TRABALHO DE GRADUAÇÃO

SISTEMA DE CONTROLE DE AR CONDICIONADO DE UM AMBIENTE *EXPERIENCE*

Por,

Daniel Yoshimitsu Kuwae

Renan do Carmo Marques Ramo

Brasília, Outubro de 2012

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

SISTEMA DE CONTROLE DE AR CONDICIONADO DE UM AMBIENTE *EXPERIENCE*

POR,

Daniel Yoshimitsu Kuwae

Renan do Carmo Marques Ramos

Relatório submetido como requisito parcial para obtenção

do grau de Engenheiro de Controle e Automação.

Banca Examinadora

Prof. Lélio Ribeiro Jùnior, UnB/ ENE (Orientador)

Profª. Carla Cavalcante Koike, UnB/ CIC

Prof. Gerson Henrique Pfitscher, UnB/ CIC

Brasília, Outubro de 2012.

FICHA CATALOGRÁFICA

DANIEL YOSHIMITSU, KUWAE RENAN DO CARMO MARQUES, RAMOS Sistema de controle de ar condicionado de um ambiente *experience*, [Distrito Federal] 2012. xvii, 99p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, Ano 2012). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia. 1. Ambiente *Experience* 2.Comunicação Serial 3.Sensoriamento de mesa interativa 4. Infravermelho I. Mecatrônica/FT/UnB II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

Kuwae, DY, Ramos, RCM, 2012. Sistema de controle de ar condicionado de um ambiente *experience*. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT. TG-nº 4, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 99p.

CESSÃO DE DIREITOS

AUTOR: Daniel Yoshimitsu Kuwae e Renan do Carmo Marques Ramos

TÍTULO DO TRABALHO DE GRADUAÇÃO: Sistema de controle de ar condicionado de um ambiente *experience*.

GRAU: Engenheiro ANO: 2012

_____

Daniel Yoshimitsu Kuwae
SMPW Qd 20 cj 04 lt 03 unidade G, Park Way.
71745-004 Brasília – DF – Brasil.

_____

Renan do Carmo Marques Ramos
Sobradinho, condomínio Recanto da Serra, rua 05 lote 02/04
73270-365 Brasília – DF – Brasil.

AGRADECIMENTOS

Agradeço a oportunidade oferecida pelo ITAE – Laboratório de Inovações Tecnológicas para Ambientes *Experience* na pessoa do senhor Jorge Pereira, ao Professor Lélio Ribeiro Jùnior por sua dedicação e paciência nossos ensinamentos passados, aos meus pais por todo apoio dedicado a minha pessoa e ao parceiro Renan do Carmo Marques Ramos nessa empreitada.

Daniel Yoshimitsu Kuwae.

Agradeço ao ITAE e ao Jorge Pereira, ao Professor Lélio e sua dedicação e paciência conosco, aos meus familiares, que me deram todo o apoio durante o trabalho e ao parceiro Daniel Yoshimitsu Kuwae.

Renan do Carmo Marques Ramos.

RESUMO

Este trabalho traz o desenvolvimento de um acionador via computador de um ar condicionado a partir de um sinal infravermelho e, também, o sensoriamento de uma mesa interativa que coleta as informações das simulações de um ambiente *experience* presente no ITAE – Laboratório de Inovações Tecnológicas para Ambientes *Experience*. O ambiente *experience* é um moderno espaço para capacitação por meio de simulações da realidade.

Palavras Chave: *ambiente experience*, *comunicação serial*, *sensoriamento de mesa interativa*, *infravermelho*

ABSTRACT

This paper presents the development of an drive air conditioner via computer from an infrared signal and also sensing an interactive table that collects information from the simulations in an experience environment present in the ITAE - Technological Innovations Lab for Environments Experience. The environment experience is a modern space for training through simulations of reality.

Keywords: *experience place*; *serial comunication*; *sensing interactive desktop*; *infrared*

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

s	Tempo	[s]
V	Tensão Elétrica	[V]
L	Comprimento	[m]
Mbps	Megabits por segundo	[Mbps]
F	Farad	[F]
Hz	Hertz	[Hz]

Símbolos Gregos

Ω	Resistência	[Ohm]

Subscritos

led	led
min	mínimo
ucPB1	microcontrolador porta PB1

Siglas

ITAE	Laboratório de Inovações Tecnológicas para Ambientes *Experience*
EAP	Estrutura Analítica de Projeto
PC	*Personal Computer* (computador pessoal)
IR	*Infrared* (Infravermelho)
USART	*Universal Synchronous and Asynchronous serial Receiver and Transmitter*
RS	*Recommended Standard*
DTE	*Data Terminal Equipment*
DCE	*Data Communication Equipment*)
AGC	*Automatic Gain Control*
LSB	*Least Significant Bit*

CAPÍTULO 1 – INTRODUÇÃO

O ITAE – Laboratório de Inovações Tecnológicas para Ambientes *Experience* – é um moderno ambiente para capacitação por meio de simulações da realidade. Na Figura 1.1, apresentamos a mesa interativa e o ar condicionado presente no ambiente ITAE.

Figura 1.1. Mesa interativa do ITAE

Durante a simulação, o laboratório deve interagir nos sistemas de iluminação, de áudio, de imagens e de temperatura do ambiente, de acordo com o jogo virtual. No momento, o espaço possui o controle da iluminação, do áudio e da projeção das imagens organizado numa arquitetura de Servidor-Cliente distribuída em 5 computadores. Uma necessidade é suprir o controle de temperatura do ambiente e de uma nova interface da mesa interativa.

O presente trabalho tem por objetivo projetar um sistema de controle de sensoriamento da mesa interativa que realiza a interface homem-máquina dos participantes nas simulações/jogos virtuais e o controle remoto automatizado do ar-condicionado.

Logo, o sistema proposto deve ser capaz de monitorar a mesa interativa e acionar remotamente o ar-condicionado no ambiente via sinal infravermelho (IR – *infrared*). As ações (sensoriamento da mesa interativa e acionamento do ar-condicionado) deverão ocorrer a partir de um comando de computador, chamado de mestre, por meio de um barramento de comunicação. O mestre será capaz de monitorar as decisões dos participantes por meio da leitura das teclas ativadas na mesa interativa e acionar as funcionalidades do ar-condicionado de acordo com cada simulação.

A mesa interativa é semelhante a um teclado com dimensões ampliadas. O equipamento já existe no ambiente. Esse equipamento possui um *hardware* e um *software* com uma arquitetura fechada e sem a documentação, ou seja, um sistema que não permite alterações, ampliação e manutenção do equipamento. Assim, caso haja falha no funcionamento da placa da mesa, não seria possível ocorrer sua substituição.

Já, o controle de temperatura do ambiente realiza-se por meio de um controle remoto do equipamento. O seu acionamento é realizado pelo facilitador do jogo conforme sua percepção. Então, não há presente um sistema de intervenção direta da simulação sobre a temperatura do ambiente.

Para organizar e planejar as etapas do trabalho, utilizamos a ferramenta chamada Estrutura Analítica de Projeto (EAP). Apresentamos a configuração da EAP na Figura 1.2, que traz uma visão holística do trabalho. Iniciamos o trabalho pelo conhecimento do funcionamento da mesa e do controle remoto que acompanha o equipamento de ar-condicionado.

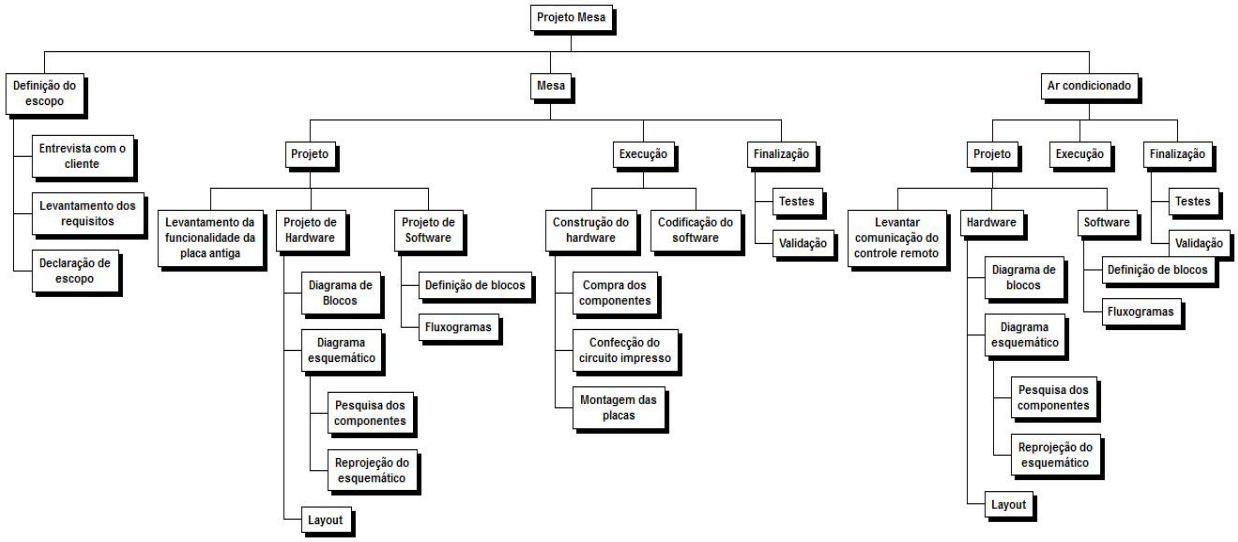

Figura 1.2. Estrutura Analítica de Projeto do Trabalho de Graduação.

Após esse estudo, definimos o projeto de *hardware*. Optamos pelo desenvolvimento de um *hardware* específico para cada sistema proposto neste trabalho com comunicação via rede RS485. Definimos o diagrama de blocos, no qual projetamos as funcionalidades do circuito eletrônico. Com esse diagrama desenvolvido, projetamos o diagrama esquemático, ou seja, a arquitetura de *hardware* do dispositivo, confrontando com os componentes disponíveis no mercado. A configuração da arquitetura do *hardware* projetada conterá um microcontrolador que possui o *software* do escravo. Além disso, temos o barramento de comunicação entre os dispositivos e o computador, e os componentes auxiliares e específicos para cada aplicação.

CAPÍTULO 2 – CONCEITUAÇÃO

Conforme exposto no Capítulo 1, identificamos duas necessidades: o desenvolvimento de um novo sistema de sensoriamento da mesa interativa presente no ambiente e a implementação do controle remoto do sistema de ar-condicionado. Em ambos os sistemas, houve o desenvolvimento tanto do dispositivo de *hardware* como de *software*.

A comunicação entre os dispositivos baseia-se em um sistema mestre-escravo. Assim, o dispositivo mestre é um computador pessoal (PC) presente no ambiente que enviará uma requisição de leitura do estado da mesa ou acionamento de funções do ar-condicionado de acordo com as demandas das simulações/jogos. Já, o escravo utiliza placas desenvolvidas com uso de microcontroladores que comunicam com o PC via rede RS 485, atendendo as demandas do mestre. Logo, o *software* mestre estará instalado no PC e o escravo, nos microcontroladores presente no sistema. O desenho da arquitetura está representado na Fig. 2.1.

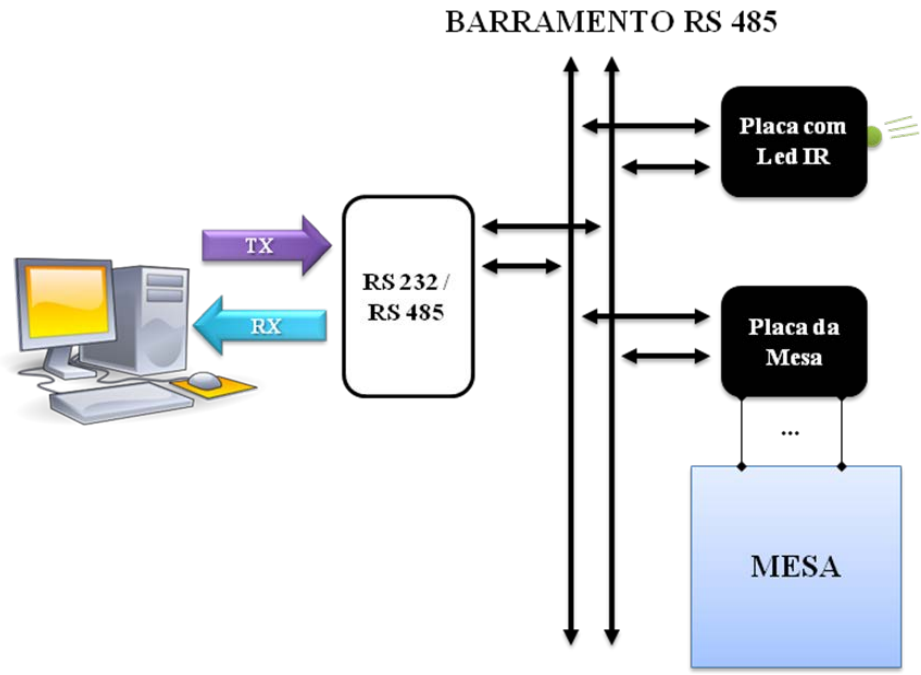

Figura 2.1. Arquitetura do sistema

No escravo, o microcontrolador é componente central, que realiza o processamento da informação e o sensoriamento/acionamento do circuito. A sua escolha impacta no desenvolvimento da placa tanto no aspecto do *hardware* como no do *software*. A seguir, apresentamos uma breve explicação sobre o ambiente *experience*, as características e os conceitos utilizados no trabalho.

2.1 AMBIENTE *EXPERIENCE*

O ambiente *experience* é um espaço no qual o ITAE desenvolve suas atividades de capacitação. Mas, o que é o ITAE?

O ITAE é um moderno ambiente para capacitação por meio de simulações da realidade. Baseado nos princípios da Economia da Experiência, o ambiente é dotado de um sistema hipermídia, que permite o compartilhamento de informações por meios diversificados, como textos, vídeos, sons, animações e outros, que favorecem a interação e o aprendizado.[7]

A Economia da Experiência pressupõe o aprendizado como resultante de um processo de transformação que ocorre a partir do envolvimento emocional, sensorial e cognitivo do indivíduo em torno de uma temática ou questão específica. Ainda com base nos princípios da Economia da Experiência, as dinâmicas educacionais vivenciadas causam impacto maior do que a simples exposição ao conteúdo, como nos modelos educacionais tradicionais, daí o que os pesquisadores da Economia da Experiência chamam de memorabilidade, onde o aprendizado é fixado mais facilmente.[7]

Com o uso de tecnologias interativas diversificadas, como telas touchscreen, mesas sensíveis ao toque, computadores de mão e controles de temperatura e iluminação, o ambiente confere a possibilidade de imersão e mobilidade dos participantes, garantindo a perfeita sintonia entre o conteúdo, o ambiente, a interatividade e os próprios participantes. A tecnologia exerce a função de catalisadora do processo de aprendizado e assume papel transparente e natural, na medida em que são oferecidas soluções de tecnologias voltadas para o ser humano.[7]

A possibilidade de inserção de novos conteúdos e customização dos já existentes faz parte da estratégia educacional adotada, possibilitando a renovação dos conteúdos, o impacto e a eficiência promovidos pela experiência educacional.[7]

2.2 MICROCONTROLADOR ATMEL ATMEGA8

O microcontrolador ATmega8, da ATMEL, possui uma boa relação custo-benefício, e é apropriado para aplicações da proposta deste trabalho. Uma de suas maiores características é a integração de várias funcionalidades em um único chip, o que permite a construção de um circuito mais simples, robusto e barato, quando comparado com um circuito similar construído com diversos dispositivos discretos. Dentre as diversas funcionalidades integradas, podemos citar:

• Um conversor AD com seis entradas multiplexadas. [5]

• Oscilador interno que pode ser ajustado entre 1 e 8 MHz, evitando a necessidade de uso de cristal ou outro circuito oscilador externo. Pode-se, entretanto, usar um cristal externo para se obter um CLK de 16 MHz e, com isso, maior desempenho. [5]

• Memória flash interna programável de 8 K bytes, usada como memória de programa. Pode-se reprogramá-la até cerca de dez mil vezes, de maneira fácil e rápida. [5]

• Memória RAM interna de 1K byte, usada como memória volátil. [5]

• Memória EEPROM de 512 bytes, para armazenar dados de maneira não-volátil. [5]

Outra característica importante é o desempenho. Esse deve ser grande o suficiente para que se consiga rodar a lógica, e que se possua uma pequena folga para permitir mudanças e melhorias futuras no algoritmo. Sobre o ATmega8, podemos citar:

• Processamento em 8 bits. [5]

• 130 instruções em *assembly*, onde a maioria realiza um ciclo de execução por ciclo de CLK. Isto, junto com o *pipeline*, permite um *throughput* de até 16 MIPS com um cristal de 16 MHZ. [5]

• Multiplicação em hardware, realizada em dois ciclos. [5]

• 32 registradores de uso geral. [5]

• Dois contadores/temporizadores de 8 bits e um de 16 bits. [5]

• Lógica de comunicação serial síncrona e assíncrona (USART) em hardware, o que reduz o custo de processamento. [5]

• Tensão de operação: 4,5 – 5,5 V (ATmega8) / 2,7 – 5,5 V (ATmega8L). [5]

• Consumo de corrente (4MHz, 3V, 25ºC): 3,6 mA (modo ativo) 1,0 mA (modo inativo) e 0,5 µA (desligado). [5]

2.3 COMUNICAÇÃO SERIAL

A comunicação do sistema deste trabalho é uma comunicação serial baseada na lógica mestre-escravo. Para haver a comunicação entre o mestre e o escravo, é necessário definir sua camada física de comunicação, chamada também de barramento. A comunicação entre o mestre e os escravos será feita com uso dos barramentos RS-232 e RS-485. O uso do barramento RS-232 é devido ao fato de que a porta serial do PC usa esse barramento, e o uso do barramento RS-485 devido à necessidade de comunicar-se com mais de um escravo. Um conversor RS-232/RS-485 será utilizado para tanto. A seguir, uma exposição sobre os barramentos envolvidos no trabalho.

2.3.1 Barramento

O barramento é o meio físico pelo qual a informação trafega entre dois ou mais dispositivos. A escolha do barramento influencia em vários aspectos da transmissão, como a taxa em que os bits são transmitidos, a imunidade a ruídos e a quantidade máxima de elementos que podem ser ligados ao barramento.

Sabe-se que o microcontrolador ATmega8 possui USART implementada via hardware. A USART significa *"Universal Synchronous "and" Asynchronous serial Receiver "and" Transmitter"* (Transmissor/Receptor Serial Universal Síncrono e Assíncrono). A USART é um dispositivo de comunicação serial altamente flexível. As principais características são [5]:

- Operação *Full Duplex* (Registradores de Transmissão e Recepção Serial Independente). [5]:
- Operação Assíncrona ou Síncrona. [5]
- Operação Síncrona Sincronizada em Mestre ou Escravo. [5]
- Alta Resolução do Gerador de Taxa de Baud. [5]
- *Frames* Serial Suportados com 5, 6, 7, 8 ou 9 de Databits e 1 ou 2 *Stop* Bits. [5]
- Detecção de saturação de dados. [5]
- Detecção de Erro de *Framing*. [5]
- Filtragem de ruído inclui Detecção de Bit de Início Falso e Filtro Passa-Baixo Digital. [5]
- Três Interrupções separados em TX Completo, Registrador Vazio de Dado TX e RX Completo. [5]
- Modo de Comunicação Multi-processador. [5]
- Modo de Comunicação Assíncrona de Velocidade Dupla. [5]

Para tornar os barramentos compatíveis com a USART, faz-se necessário o uso dos transceptores correspondentes do RS-232, RS-485 e RS-422.

2.3.1.1 RS485

O padrão RS485 foi desenvolvido em convênio entre a EIA (*Eletronic Industries Association*) e a TIA (*Telecommunications Industry Association*). O prefixo "RS" significa "*Recommended Standard"*. Posteriormente todas as normas com prefixo "RS" foram renomeadas usando o prefixo "EIA/TIA", porém o uso consagrou o nome RS485[1]. É o único padrão EIA/TIA que permite múltiplos receptores e drivers em um barramento.

O padrão RS485 faz uso de um par diferencial, sinal A e B, para representar os 2 níveis lógicos possíveis. O sinal A possui a mesma lógica TTL, enquanto o sinal B é complementar. Se a diferença entre A e B for maior que 200 mV, tem-se o nível lógico 1, e se a diferença for menor que -200 mV, tem-se o nível lógico 0. Entre -200 mV e 200 mV o

nível lógico é indefinido. Para gerar o par diferencial, faz-se uso de um transceptor (como exemplo, tem-se os transceptores DS485 e ST485, Figura 2.2), o qual gera os sinais na saída de acordo com o nível lógico da entrada TTL [2].

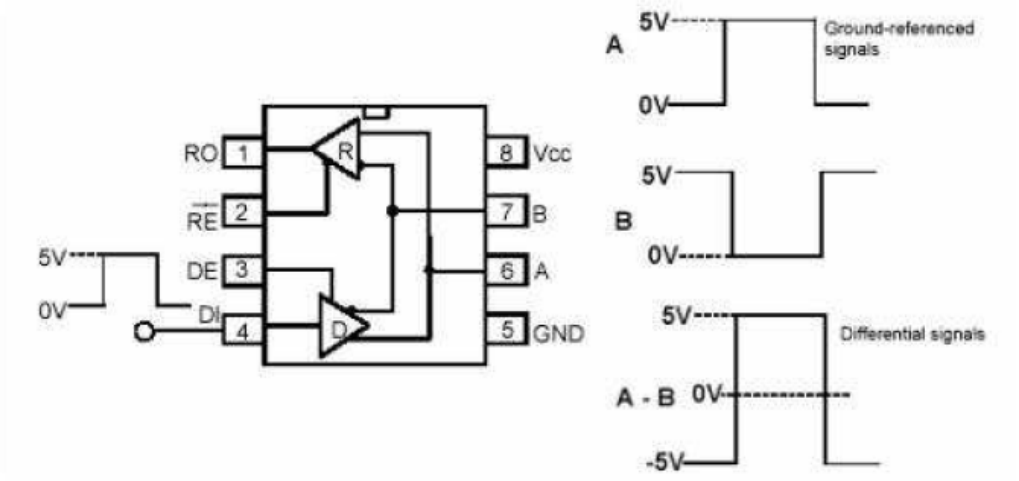

Figura 2.2. Diagrama dos transceptores DS485 e ST485, e sinais diferenciais [2].

O pino RE habilita a leitura quando ativo e o pino DE habilita a escrita. O transceptor utilizado neste trabalho será o MAX485, da Maxim.

Uma das vantagens de se usar o par diferencial é a sua robustez a interferência eletromagnética. Esta vantagem se torna mais importante a medida que maiores distâncias são necessárias para a comunicação. Outra vantagem da comunicação RS485 é a possibilidade de uma comunicação com vários dispositivos, podendo haver numa rede, dependendo do transceptor utilizado de até 200 dispositivos. O padrão RS485 permite uma taxa de transmissão de 10 Mbps a distâncias de até 1 km.

2.3.1.2 RS232

O RS232 (também conhecido por EIA RS 232C ou V.24) é um padrão para troca serial de dados binários entre um DTE (*Data Terminal Equipment*) – Equipamento de Terminação de Dados, dispositivo gerador ou consumidor de dados (ex: PC, servidor, terminal, entre outros) – e um DCE (*Data Communication Equipment*). Quaisquer equipamentos de comunicação de dados, dispositivos componentes de uma rede (ex: modem, repetidor, switch, roteador, etc). É comumente usado nas portas seriais dos Computadores Pessoais. O padrão foi originalmente usado para conectar um teletipo (equipamento eletrônico de comunicação assíncrona que usava código ASCII) a um modem, e foi padronizado pela EIA em 1969 [3].

O padrão especifica uma comunicação ponto a ponto (2 nós), e utiliza duas linhas de dados, Tx e Rx, respectivamente para o envio e a recepção dos dados. O nível lógico do sinal é determinado comparando o sinal em questão com o GND. Tensões entre -3 V e -15 V

representam nível lógico 1, enquanto tensões entre 3 V e 15 V representam nível lógico 0. Para tensões entre -3 V e 3 V o nível lógico é indefinido. Outros sinais podem ser utilizados para o controle de fluxo de dados [3].

O meio físico utilizado na comunicação é o cabo de cobre. O padrão não define distância máxima de transmissão, entretanto, define uma capacitância máxima de 2500 pF que os cabos envolvidos devem suportar. Para cabos de 50 pF/pé, isso equivale a cabo de 50 pés (15 m). O padrão suporta comunicação síncrona e assíncrona. O padrão é de baixa velocidade (máximo de 115 kbps) [3].

2.3.1.3 *Hardware* da Comunicação serial

Para a comunicação serial foram usados dois transceptores MAX485. Este circuito integrado é responsável por transformar os níveis de tensão TTL usados na USART para os níveis exigidos pelo padrão RS485. Adicionalmente, são usados dois resistores de 120 Ω, um entre os dois fios da linha de escrita e um entre os dois fios da linha de leitura.

A comunicação implementada é *full duplex*, uma linha para escrever e outra para ler. Desta forma, um dos MAX485 é habilitado para escrita somente quando é o momento de escrever e o outro permanece constantemente habilitado para leitura. Sendo assim, o pino 1 (RO) do MAX485 da leitura é conectado diretamente no pino 2 (TX) do Atmega8 e os pinos 2 (RE) e 3 (DE) são aterrados, enquanto que e o pino 4 (DI) do outro MAX485 é conectado ao pino 3 (RX) do Atmega8 e os pinos 2 e 3 são conectados ao pino 4 (INT0) do microcontrolador para habilitar a escrita. A Figura 2.3 ilustra o hardware da comunicação serial.

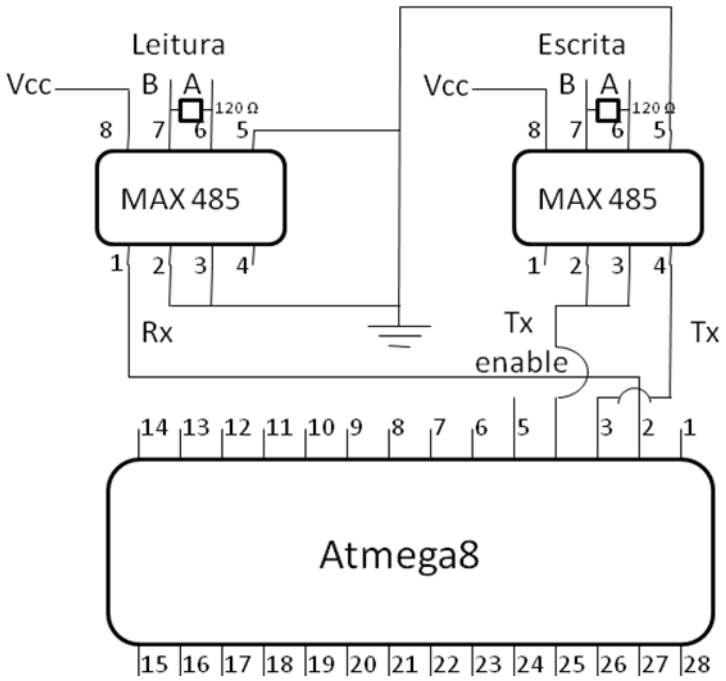

Figura 2.3. Comunicação Serial.

Um conversor RS485/232 é feito com uso de um MAX 232. São utilizados neste conversor dois MAX 485, 5 capacitores eletrolíticos de 0.1 μF, 2 resistores de 120 Ω e 1 resistor de 10 kΩ. A Figura 2.4 ilustra o conversor.

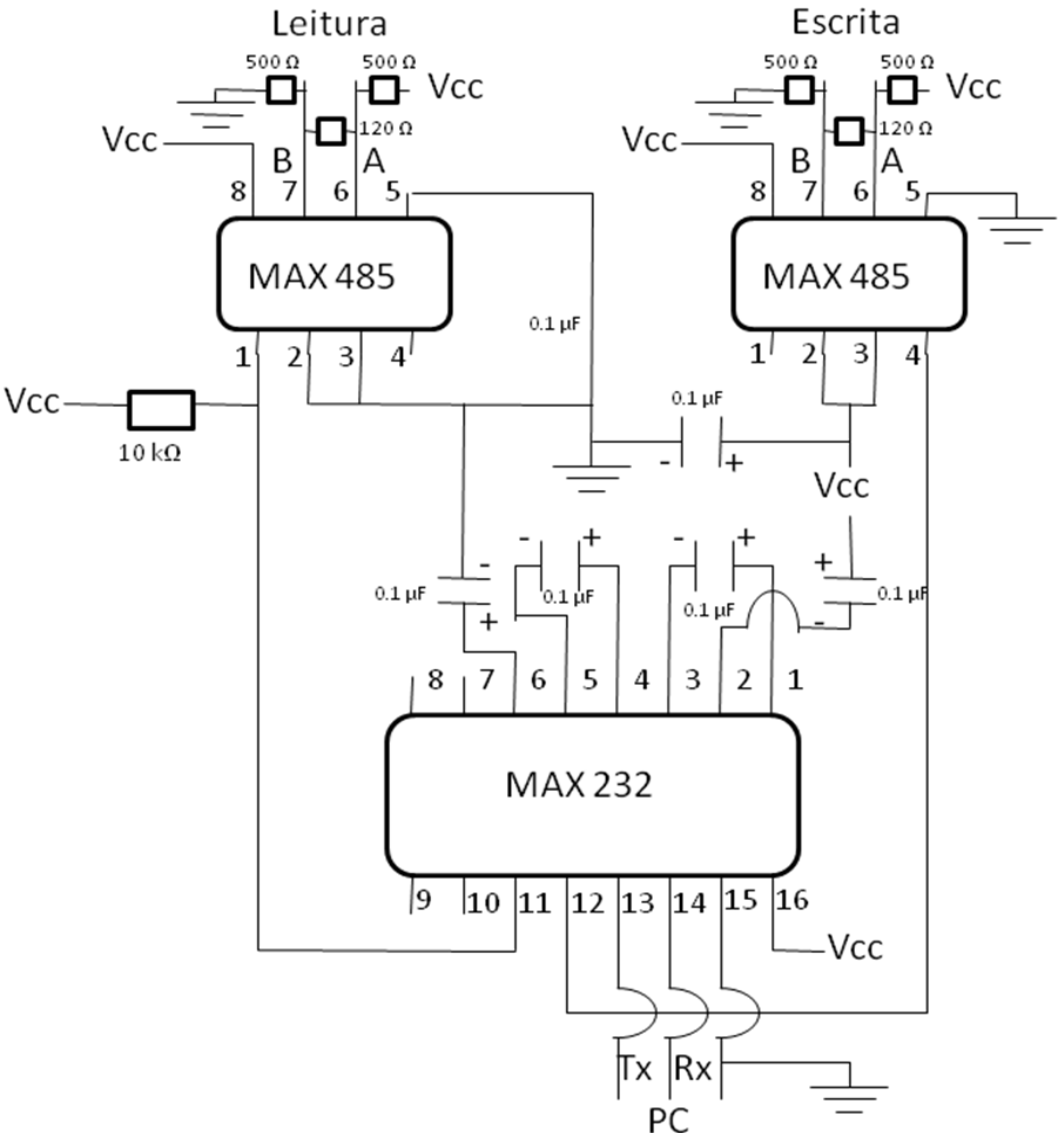

Figura 2.4. Conversor RS485/232.

2.3.2 Protocolo da Comunicação Serial

O protocolo da comunicação serial baseia-se num protocolo desenvolvido nos produtos da Jeffrey Kerr [4]. Esse é um protocolo restrito mestre-escravo, onde os pacotes de comando são enviados para um módulo do controlador pelo computador mestre, e um pacote de estado é retornado do módulo. O protocolo de comunicação usa 8 bits de dados, 1 bit de início, 1 bit de parada e nenhum de paridade.

Pacotes de comando são transmitidos pelo mestre através de uma linha de comando dedicados. Pacotes de estado são recebidos através de uma linha de estado separado, que é compartilhada por todos os módulos na rede. Devido ao fato do mestre não ter que

compartilhar a linha de comando, a porta de comunicação do mestre pode ser uma porta padrão RS232 com um simples conversor de nível de sinal de RS232 para RS485 (ou RS422). As portas dos escravos, no entanto, devem ser capazes de desativar seus transmissores para evitar colisões de dados sobre a linha de estado compartilhado.

Os pacotes de comando têm a seguinte estrutura:

- *byte* de Cabeçalho (sempre 0xAA)
- *byte* de Endereço do Módulo (0 - 15)
- *byte* de Comando
- *bytes* Adicional de Dados (0 - 15 bytes)
- *Checksum byte* (8-bits da soma do *byte* de endereço do módulo, do comando e dos *bytes* de dados adicionais)

O *byte* de Cabeçalho é usado para sinalizar o início de um pacote de comando. Quando à espera de um novo comando, cada módulo irá ignorar todos os dados de entrada até que veja um sinal semelhante ao *byte* de cabeçalho. O valor do byte do cabeçalho é 0xAA. Ao transformar para binário (0b10101010), percebe-se a constante alternância de valores. Esta alternância é difícil de ser gerada por ruído, o que reduz o acontecimento de falsos inícios de comando. O próximo *byte* é o de endereço.

O *byte* de Comando é dividido em um *nibble* superior (4 bits) e um *nibble* inferior (4 bits). O *nibble* inferior contém o valor de comando (0-15), e o *nibble* superior contém o número de bytes de dados adicionais necessários para o comando (0 - 15). Cabe ao mestre garantir que o *nibble* superior corresponda ao número de bytes de dados adicionais realmente enviados. Os *bytes* de Dados Adicionais contêm os dados específicos que podem ser exigidos para um determinado comando. Muitos comandos têm um *byte* de "controle" ou "modo" acrescido de outros parâmetros necessários para o comando. Alguns comandos não requerem dados adicionais. Cabe ao mestre certificar-se de que o número correto de *bytes* de dados adicionais seja enviado para um determinado comando, e que o *nibble* superior do *byte* do comando seja igual a este número.

O escravo para o qual o comando foi destinado deve retornar um pacote de estato para o mestre depois de ter executado o comando. Caso tenha se verificado erro de *checksum* no pacote de comando, o comando não será executado, mas o pacote de estado ainda será devolvido.

Os pacotes de status ter a seguinte estrutura:

- *byte* de Estado
- *bytes* de Dados Adicionais de Estado (programável)
- *Checksum byte* (8-bits da soma de todos os *bytes* acima)

O *byte* de Estado contém informações básicas sobre o estado do módulo, incluindo ou não se o comando em questão tinha um erro de *checksum*.

O número de *bytes* de Dados Adicionais de Estado é programável. Exatamente quais dados estão incluídos nesses *bytes* que podem ser programados usando os comandos *Define Status* ou *Read Status*.

Nenhum novo comando deve ser enviado até que o pacote de estado seja retornado para evitar sobrescrita do *buffer* de dados de comando e para evitar colisões na linha de status. Se, no entanto, o mestre não enviar nenhum dado antes que um pacote de estado seja recebido, todos os escravos na rede desativarão qualquer transmissão de dados de estado em andamento e ouvirão o novo comando a partir do mestre. Isto assegura que o mestre possa sempre dominar a atenção de todos os escravos na rede.

2.3.2.1 Comandos

Apresentamos o conjunto de comandos na Tab. 2.1. Ao todo, foi necessário o uso de 4 comandos. Como pode ser visualizado na Tab. 2.1, há disponibilidade para a configuração de novos comandos. Na coluna “Comando”, temos um mnemônico dos comandos. A coluna “Código” apresenta o número do comando. Por fim, a coluna “Descrição” possui uma breve descrição da funcionalidade do comando.

A seguir, apresentamos mais um detalhamento dos comandos no pacote comando.

Set Status

1° *byte*: Cabeçalho

2° *byte*: Endereço do módulo

3° *byte*: 0x10|0x00 = 0x10

4° *byte*: *Status Itens* (*byte* adicional)

5° *byte*: *Cheksum* = 2° *byte* + 3° *byte* + 4° *byte*

No terceiro *byte*, temos uma operação lógica “ou bit a bit” entre 0x10 (1 *byte* adicional) com 0x00(código do comando). No quarto *byte*, temos o *Status Itens*, que determina as informações do escravo que terão enviar no pacote de status em resposta a todos os comandos, excluindo-se o comando *Read Status*.

Segue abaixo a definição dos bits do pacote de *Status Itens*:

- 1° bit: Determina se o escravo deve adicionar ao pacote de status 2 *bytes* correspondentes ao número de identificação e à versão do dispositivo.
- 2° bit: Não utilizado. Disponível para ser configurado pelo usuário.

- 3° bit: Não utilizado. Disponível para ser configurado pelo usuário.
- 4° bit: Não utilizado. Disponível para ser configurado pelo usuário.
- 5° bit: Determina se o escravo deve adicionar ao pacote status 24 *bytes* correspondentes às informações das 12 linhas da mesa.

Os 3 últimos bits estão disponíveis para serem configurados de acordo com a necessidade do usuário, seguindo a codificação LSB.

Tabela 2.1. Comandos do pacote de comando

Comando	**Código**	**Descrição**
Set Status	0x00	Define-se via *byte* adicional quais informações o escravo deve retornar no status em resposta a qualquer comando, com exceção do comando *Read Status*. A leitura das teclas pode ser feita por este comando.
Read Status	0x01	Lê o status. Para isso, o *byte* adicional determina quais informações serão retornadas pelo status somente em resposta a este comando. Este comando permite também a leitura das teclas.
No Op	0x02	Não realizar nenhuma ação.
Ar Condicionado	0x03	Liga e desliga o ar-condicionado. Modifica a temperatura e potência do ar condicionado.
Livre	0x04	
Livre	0x05	
Livre	0x06	
Livre	0x07	
Livre	0x08	
Livre	0x09	
Livre	0x0A	
Livre	0x0B	
Livre	0x0C	
Livre	0x0D	
Livre	0x0E	
Livre	0x0F	

Read Status

1° *byte*: Cabeçalho

2° *byte*: Endereço do módulo

3° *byte*: 0x10|0x01 = 0x11

4° *byte*: *Status Itens* (*byte* adicional)

5° *byte*: *Cheksum* = 2° *byte* + 3° *byte* + 4° *byte*

No terceiro *byte*, temos uma operação lógica "ou *bit* a *bit*" entre 0x01(1 *byte* adicional) com 0x01(código do comando). Este comando funciona da mesma forma que o comando *Define Status*, com a diferença que as informações definidas pelo *Status Itens* só serão retornadas somente uma vez e em resposta a este comando. Este comando é, portanto, uma versão temporária do comando *Define Status*.

Ar Condicionado

1° *byte*: Cabeçalho

2° *byte*: Endereço do módulo em questão

3° *byte*: 0x20 | 0x03

4° *byte*: Operação = Ligar, Desligar, Jet Cool ou Mudar temperatura

5° *byte*: Potência(nibble inferior) e temperatura(nibble superior) desejada.

Cheksum = 2° *byte* + 3° *byte* + 4° *byte* + 5° *byte*

No terceiro *byte*, temos uma operação lógica "ou *bit* a *bit*" entre 0x20 (2 *byte* adicional) com 0x03 (código do comando). Temos o comando liga, desliga e *jet cool* do ar-condicionado. Também há possibilidade de alteração da temperatura e da potência de ventilação do equipamento. Em resposta ao comando, o escravo deve retornar no status as informações que foram definidas pelo comando *Set Status*. Portando, é necessário que tenha sido enviado anteriormente o comando *Set Status*.

2.3.2.2 Endereçamento

O endereçamento do escravo é realizado por software em uma constante com o seu endereço.

2.3.2.3 Estado

O pacote de estado é a resposta do escravo ao comando do mestre. Conforme já comentado, a composição desse pacote é um conjunto do *byte* de estado, os *bytes*

adicionais programáveis e o *byte* de *checksum*. A seguir, apresenta-se na tabela 2.2 com uma coluna do bit, o nome da funcionalidade (mneumônico) e sua definição.

Tabela 2.2. *Byte* de status.

Bit	Nome	Definição
0	*cksum_error*	O valor está em “1” se houve um erro de *checksum* no pacote de comando que acaba de receber.
1	Livre	
2	Livre	
3	Livre	
4	Livre	
5	Livre	
6	Livre	
7	Livre	

2.4 FUNCIONAMENTO MESA (TECLADO)

Será descrito nesta seção o funcionamento do módulo de aquisição de dados da mesa, tanto em nível de *hardware* quanto em nível de *software*.

O teclado é composto por uma matriz de 12 linhas e 12 colunas, e quando uma das teclas é apertada, isso faz com que se feche o circuito formado pela linha e coluna correspondentes. Desta forma, para saber quais teclas estão sendo pressionadas, o módulo deve excitar uma das linhas do teclado e depois verificar o estado de cada uma das 12 colunas. As colunas que estiverem em baixo juntamente com a linha em questão determinarão os índices das teclas. Este procedimento é repetido para cada uma das 12 linhas.

O problema é que, quando se aperta um botão com contato mecânico, é gerada uma série de ruídos aleatórios de duração pequena, mas com grande impacto na interpretação da tecla. Para evitar isto utilizamos um circuito de condicionamento de sinal, formado por um conjunto de 12 resistores de *pull-up* conectados um em cada linha, 12 resistores de *pull-down* e 12 capacitores de 100 pico Faraday conectados nas colunas.

2.4.1 Excitação das linhas

Para endereçar as 12 linhas da matriz do teclado, usamos dois decodificadores 74LS138. Quatro saídas (pinos 23, 24, 25 e 26) do microcontrolador são utilizadas como índice para a linha. Para o condicionamento do sinal, um resistor de *pull-up* é colocado em cada linha na saída do *driver* correspondente. Foram utilizados dois decodificadores de 3 para 8 para endereçar as 12 linhas. O pino 26 do microcontrolador é utilizado para habilitar e desabilitar os decodificadores. Desta forma, para o primeiro decodificador, o pino 6 é ligado ao VCC e os pinos 4 e 5 são ligados ao pino 26 do Atmega8, o que significa que este decodificador só ativará uma de suas linhas caso o índice seja menor que 8.

Da mesma forma, para o segundo, o pino 6 é ligado ao pino 26 do microcontrolador e os pinos 4 e 5 são aterrados, o que significa que esse microcontrolador só ativará uma de suas linhas caso o índice seja maior ou igual a 8. De acordo com o índice, o decodificador aterra uma de suas saídas e mantém as outras em alto. E como um *driver* se encontra logo depois do decodificador, consequentemente a linha em questão é mantida em alto enquanto as outras estão aterradas. A saída de cada *driver* é conectada a um resistor de *pull up* e à linha correspondente do teclado. A Figura 2.5 ilustra o circuito de excitação das linhas.

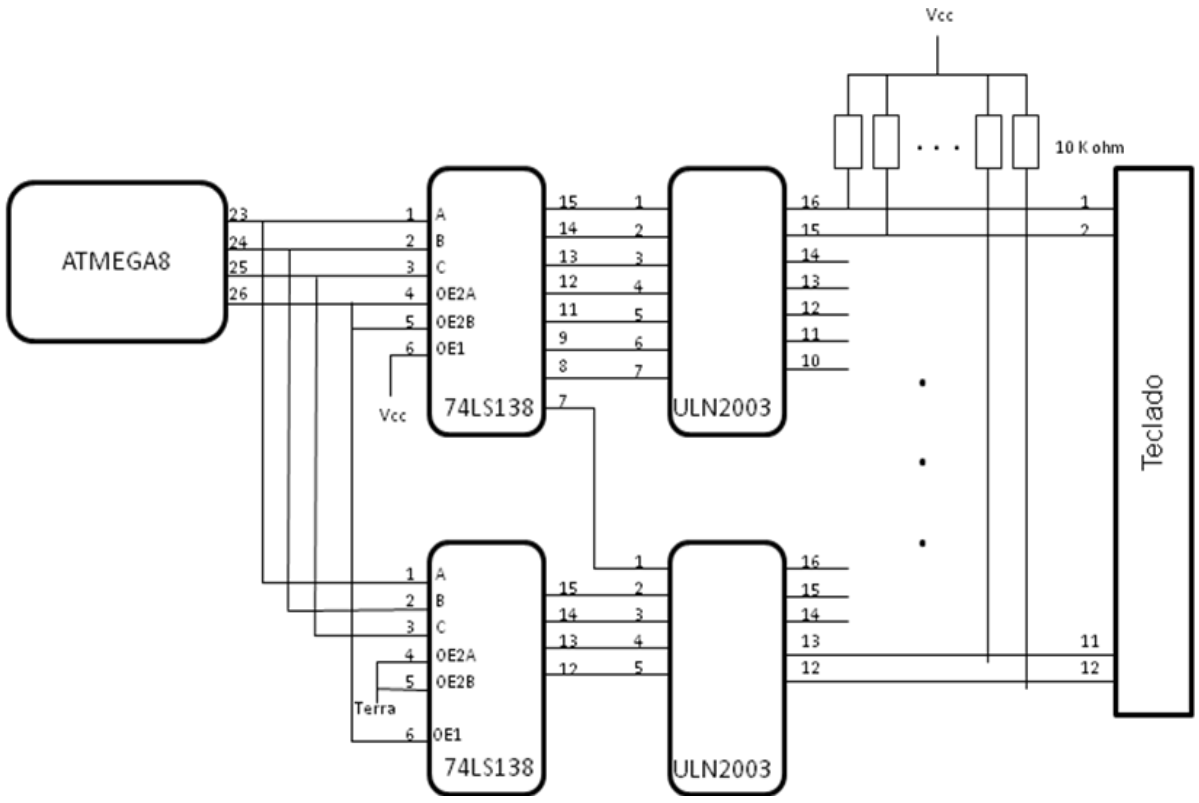

Figura 2.5. Excitação das linhas.

2.4.2 Leitura das Colunas

Para ler as 12 colunas, dois multiplexadores 74LS151 são utilizados. As 12 colunas do teclado são conectadas aos inversores 74LS14 e ao circuito de condicionamento de sinal, que é composto por 12 resistores e 12 capacitores. A saída de cada inversor é conectada a entrada correspondente do multiplexador. As saídas (pino 5) do primeiro e do segundo multiplexador são conectadas respectivamente aos pinos 12 e 13 do microcontrolador. O endereçamento dos multiplexadores é feito com uso dos pinos 14,15 e 16 do microcontrolador, que são conectados respectivamente aos pinos 11, 10 e 9 de cada multiplexador. Sendo assim, se uma tecla foi apertada, depois que a linha correspondente for excitada e que a coluna correspondente for selecionada, o valor lido pelo multiplexador deverá estar em nível baixo. A Figura 2.6 ilustra o circuito de leitura das colunas.

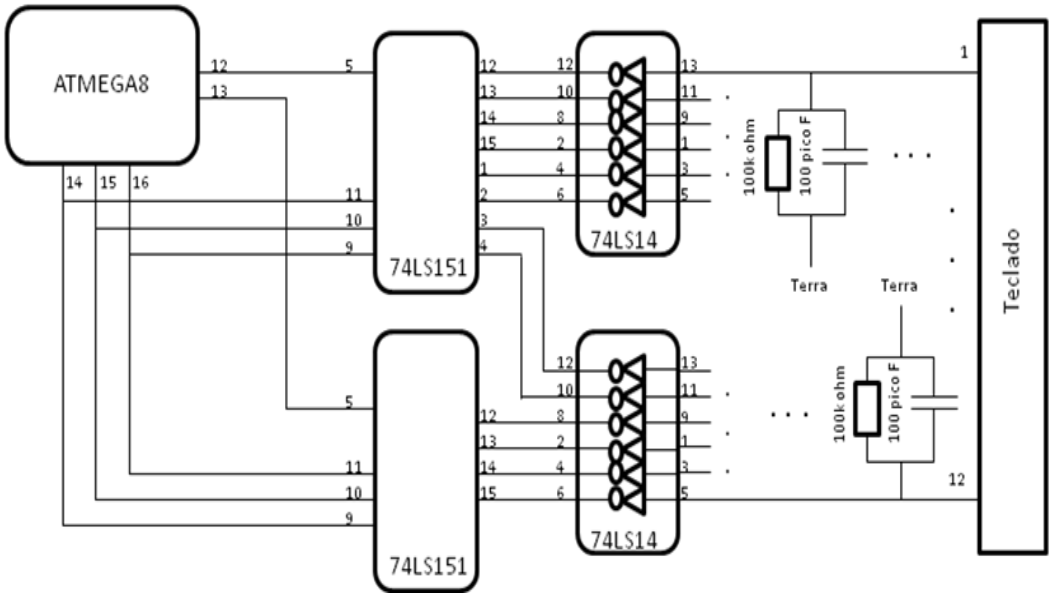

Figura 2.6. Leitura das colunas.

2.5 CONTROLE DO AR-CONDICIONADO

Para o controle do ar-condicionado, identificamos que a comunicação do sistema com o ar-condicionado TNSC2425TL0, marca LG, é via comunicação infravermelha (IR - *InfraRed*). Assim, a seguir, faremos uma breve exposição sobre essa comunicação.

2.5.1 Resumo sobre a Comunicação Infravermelha

A comunicação infravermelha é largamente utilizada nos controles remotos dos equipamentos eletrônicos. O uso da luz infravermelha é uma forma mais barata para o controle remoto dos equipamentos eletrônicos, pois os LED´s (*light-emitting diode*), um dos componentes para aplicação, são elementos de fácil confecção. Além disso, é uma luz não-

visível ao ser humano o que torna o sistema mais discreto durante o seu acionamento. O comprimento de onda é de 950nm [6].

Contudo, existe uma desvantagem: diversidade de fontes de luz infravermelha, porque tudo que irradia calor, também irradia luz infravermelha. Portanto, para garantir a correta comunicação entre o emissor e o receptor, utiliza-se a modulação de sinal. A seguir, temos uma breve explicação sobre a modulação de sinal e sobre o emissor.

2.5.1.1 Modulação

A modulação é a resposta para destacar o sinal desejado dos ruídos do ambiente. Com a modulação, a fonte de luz IR emite o sinal em uma frequência particular do sinal da portadora. Ajusta-se o receptor IR para a frequência particular, ignorando as demais frequências existem no ambiente. O sinal detectado pelo receptor na saída torna-se o sinal modulado. [6] Podemos visualizar tal situação na Fig. 2.7.

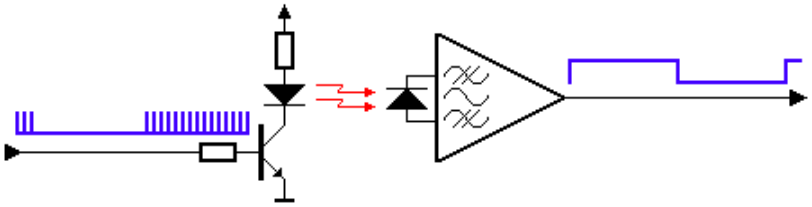

Figura 2.7. Dispositivo de Modulação [6]

Nota-se, no lado esquerdo, a presença de dois estados: “marca” e “espaço”. O espaço representa o estado desligado do emissor e é o sinal padrão. Já, a “marca” é a emissão da luz IR que pulsa na frequência definida. A faixa de frequência mais comum é entre 30 kHz e 60 kHz. No receptor, o “espaço” representa um nível alto em sua saída e a “marca”, um nível baixo. [6]

Vale ressaltar que “marcas” e “espaços” não são os 1´s e 0´s do sistema. A relação dependerá do protocolo a ser utilizado definido pelos fabricantes. [6]

2.5.1.2 Emissor

No desenvolvimento do emissor, deve haver a preocupação em produzir um sinal IR forte o suficiente para alcançar uma distância aceitável. Os diversos chips projetados podem ser usados com o emissor IR. Com o advento dos microcontroladores, houve uma redução no consumo, ampliando e flexibilizando a utilização do emissor IR. [6]

Para gerar as ondas, os ressonadores de cerâmica são mais adequados, porque eles podem resistir a grandes choques físicos. Assim, o uso dos cristais de quartzo é mais restrito devido à sua fragilidade, quebrando com facilidade quando o equipamento sofre um impacto. [6]

A variação da corrente no LED pode ser entre 100 mA a 1A. Assim, é necessário observar os parâmetros nas especificações do LED. Um circuito simples para o acionamento do LED é utilizar um transistor. O transistor serve para melhor controle da corrente e evitar danos elétricos ao controlador. [6] Na Figura 2.8, o desenho do circuito.

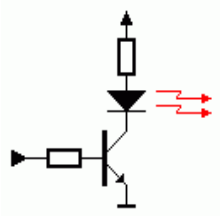

Figura 2.8. Circuito emissor [6]

No circuito, há presente dois resistores, um na base e outro no coletor. O resistor na base é para limitar a corrente do controlador ao transistor. A tensão na base é a que controla a modulação da luz IR e possui um resistor R_B. Já, o resistor R_c no coletor é utilizado como um divisor de tensão. Assim, as equações para o cálculo dos resistores R_B e R_c são os seguintes:

$$V_{cc} - V_{led} = V_C \quad (1)$$

Onde V_{cc} = 5 V e V_C é a tensão no coletor. O valor de um led de uso geral com comprimento de onda 960 nm tem V_{led} = 1,7 V, com I_c = 33 mA.

$$V_C = R_c \text{ X } I_c \quad (2)$$

No qual a corrente máxima I_C depende da corrente máxima permitida no led.

$$I_B = I_C/\beta_{min} \quad (3)$$

Onde I_B é o valor de corrente na base e I_C é o valor de corrente no coletor. O valor de β_{min} é 250. É necessário escolher o valor mínimo, pois é preciso ter uma corrente máxima para conseguir excitar a base.

$$V_{ucPB1} - V_B = V_{RB}, \quad (4)$$

Onde a tensão na saída da porta do microcontrolador é V_{ucPB1} = 5 V e a tensão da base do transistor é V_B =0,7 V.

$$V_{RB} = R_B \text{ X } I_B \quad (5)$$

Assim, os valores utilizados e de acordo com os materiais comerciais disponíveis são R_B = 27 kΩ e R_c = 100 Ω.

2.5.2 Protocolo Identificado (JVC)

O protocolo identificado, com o auxílio de um osciloscópio e um circuito integrado AVR LAB, foi o formato semelhante ao protocolo infravermelho JVC, ou simplesmente, protocolo JVC. O protocolo JVC é um desenvolvido pela Corporação JVC Kenwood. No capítulo 4, apresentaremos com mais detalhes a identificação do protocolo.

A frequência da portadora desse protocolo é de 38 kHz com um ciclo de trabalho da portadora recomendado de ¼ ou 1/3 e sua modulação é uma modulação por distância de pulso. Assim, em ambos os valores lógicos “1” e “0”, há um trem de pulso da portadora por um tempo de 526 µs (cerca de 20 ciclos). Para diferenciar os valores lógicos, é tempo de transmissão do restante do sinal. O valor lógico “1” transmite em 2,1 ms (equivalente 80 ciclos). Já, o valor lógico “0”, em 1,05 ms (equivalente 40 ciclos) [6], conforme a Fig. 2.9.

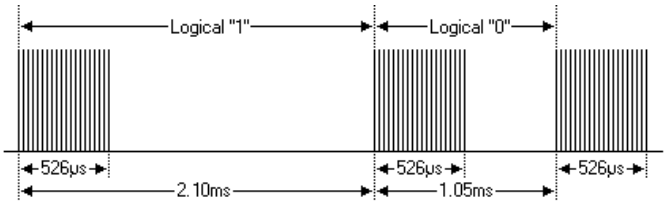

Figura 2.9. Valores lógicos do protocolo JVC. [6]

A formação do sinal ocorre da seguite forma: uma mensagem é iniciada por um pulso AGC de 8,4ms (equivalente a 320 ciclos), que é utilizado para definir o ganho dos receptores IR. Este pulso AGC é seguido por um espaço de 4,2ms (equivalente a 160 ciclos). Em seguida, acontece o envio do endereço e do comando. O comando do protocolo tem a configuração LSB - least significant bit. Na Fig. 2.10, o endereço é $59 e o comando é $59. O tempo total de transmissão é variável, porque os tempos de bit são variáveis. [6]

Figura 2.10. Exemplo de comando do protocolo JVC [6]

No caso do controle remoto, se o botão de comando mantiver-se acionado, após o primeiro envio do comando, um sinal IR é transmitido a cada 50-60 ms. Essa é forma para diferenciar no controle remoto se é o primeiro acionamento do botão ou se o botão está pressionado.[6]

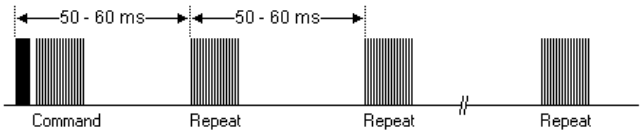

Figura 2.11. Exemplo de tecla pressionado do protocolo JVC [6]

CAPÍTULO 3 – PROJETO E EXECUÇÃO

Com base na conceituação apresentado no Capítulo 2, apresentamos o desenvolvimento de *hardware* e de *software* do trabalho. Um fator de determinante neste foi a disponibilidade dos componentes, acesso à fonte de informação e o apoio consultivo para a evolução do trabalho.

3.1 HARDWARE

O circuito do módulo constitui basicamente de um microcontrolador ATMEGA8 e dois transceptores de comunicação (MAX 485). Houve a necessidade de uso de um conversor RS 485/ RS 232 na comunicação do barramento com o computador.

No circuito da mesa, utilizamos decodificadores, multiplexadores, *drivers* ULN2003APG e inversores. No do controle remoto, usamos led e transistor.

3.1.1 Diagrama Esquemático

Apresentamos o diagrama esquemático do projeto do circuito elétrico das placas de controle remoto do ar-condicionado e de sensoriamento da mesa interativa nas Figs. 3.1 e 3.2., respectivamente. Os detalhes dos componentes das placas apresentamos abaixo.

3.1.1.1 Microcontrolador ATMEGA8

Em ambos os diagramas dos circuitos, há o uso do microcontrolador ATMEGA8. Descreveremos as suas principais funcionalidades utilizadas nos projetos.

Na comunicação entre o mestre e os escravos, utilizamos os pinos 2 (RXD) e 3 (TXD). Para alimentar, o circuito digital do sistema são utilizados os pinos 7 e 8, e para alimentar o circuito de conversão analógica/digital são utilizados os pinos 20, 21 e 22. Mesmo que a conversão analógica/digital não seja usada, é necessário se conectar os pinos 20 e 22.

Para a placa de aquisição das teclas são utilizadas as portas C, B e D. Os pinos 23, 24, 25 e 26 da porta C são utilizados para acionar os decodificadores, enquanto que os pinos 14, 15 e 16 da porta B são utilizados para acionar os multiplexadores. Para ler as 8 primeiras colunas, isto é feito lendo-se a saída do primeiro multiplexador, que é conectada ao pino 12 (AIN0). Para ler as 4 últimas colunas, isto é feito lendo-se a saída do segundo multiplexador, que é conectada ao pino 13 (AIN1).

Na placa do controle remoto, o pino 15 (PB1) é a porta utilizada, pois utilizamos a interrupção do "*timer* 1" para modulação do sinal infravermelho.

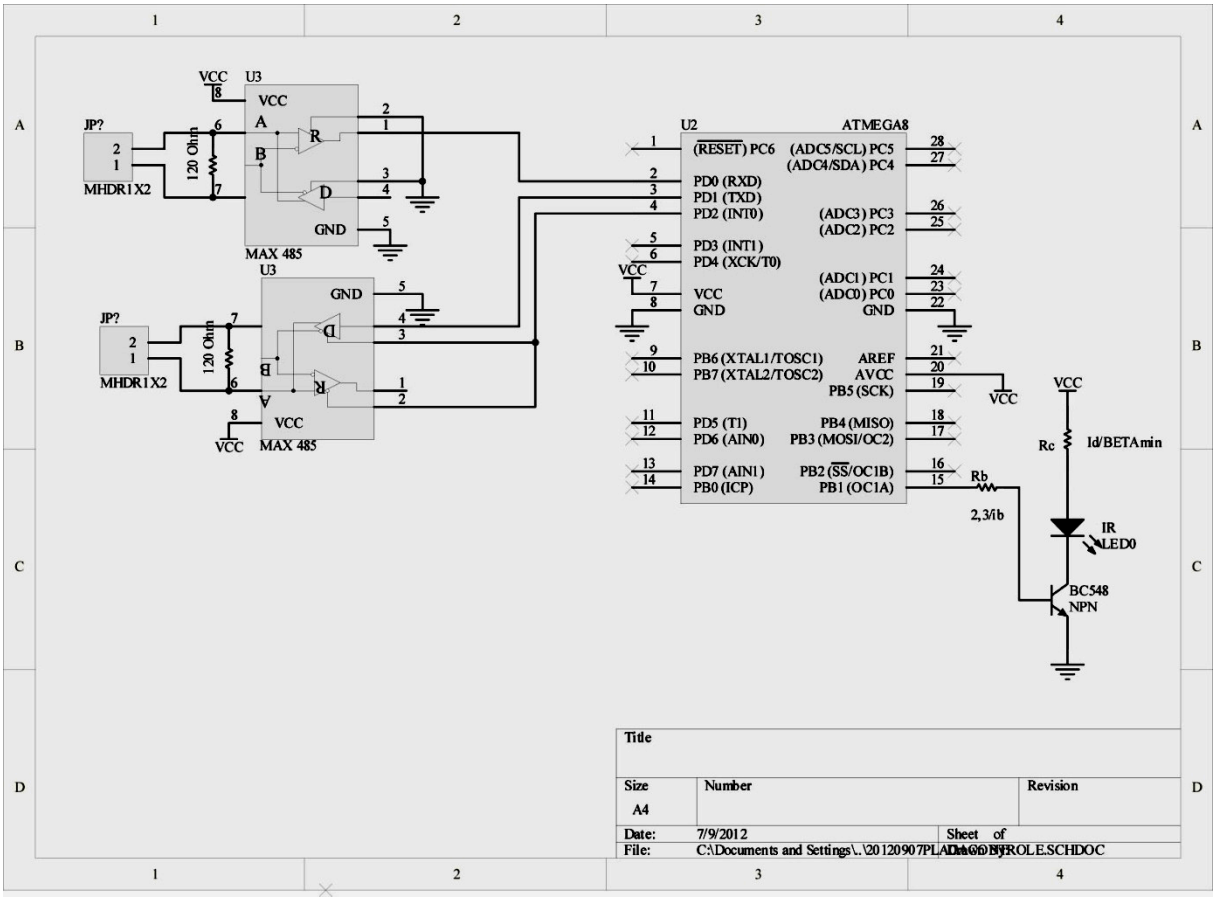

Figura 3.1. Diagrama Esquemático do Controle Remoto do ar-condicionado

3.1.1.2 TRANSCEPTOR MAX 485

Usamos dois MAX 485 como o transceptor para cada módulo, um para a escrita e outro para a leitura. No da escrita, os pinos 2 e 3 são ligados diretamente ao pino 4(INT0) do microcontrolador para o controle de escrita, e no da leitura, os pinos 2 e 3 são aterrados, habilitando-se permanentemente a leitura. Este circuito integrado é responsável por transformar os níveis de tensão TTL usados na USART para os níveis exigidos pelo padrão RS485. Adicionalmente, são usados dois resistores de 120 Ω, um entre os dois fios da linha de escrita e um entre os dois fios da linha de leitura. Os fios são conectados com ajuda de um conector parafusado.

A comunicação implementada é *full duplex*, uma linha para escrever e uma linha para ler. Desta forma, um dos transceptores é habilitado para escrita e o outro para leitura. Sendo assim, o pino 1 (RO) de um transceptor é conectado diretamente no pino 2 (TX) do microcontrolador, e o pino 4 (DI) do outro transceptor é conectado ao pino 3 (RX) do microcontrolador.

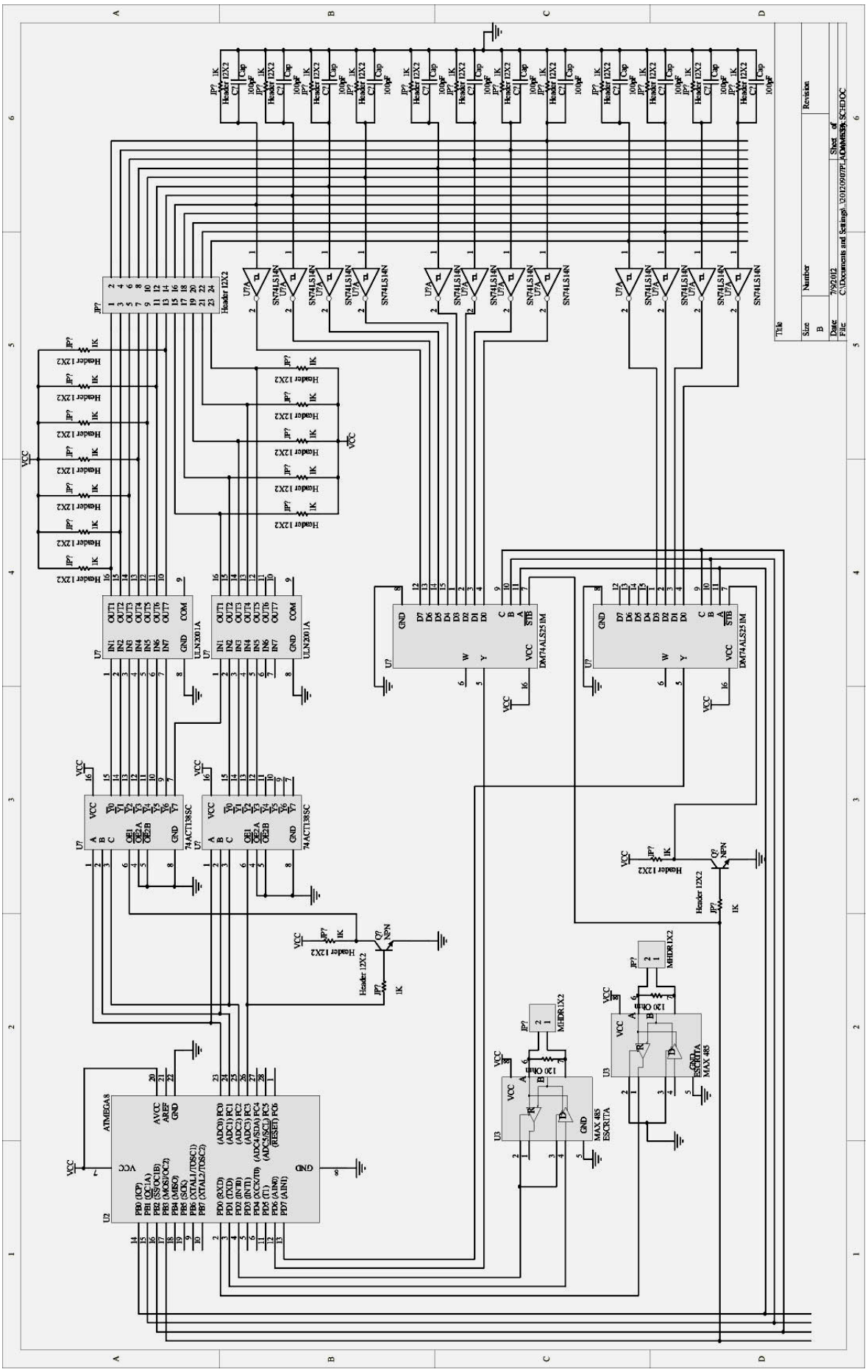

Figura 3.2. Diagrama Esquemático da Mesa Interativa

3.1.1.3 LED

Já, as particularidades do controle remoto do ar-condicionado, na Fig. 3.1, são o CI BC548, que possui o transistor com a função de proteção do microcontrolador e de controle do LED, e o LED, til 32. Não possível ter acesso ao seu *datasheet*, assim utilizamos os valores padrões, conforme presente nos cálculos no Capítulo 2. Os dois resistores utilizados são um na base do transistor com o valor de 27 kΩ e um no coletor de 100 Ω.

3.2 SOFTWARE

Usamos dois *softwares* neste trabalho, um para o mestre e outro para o escravo. O programa mestre baseia-se no *software* utilizado nos produtos da Jeffrey Kerr e o escravo, no *software* do Trabalho de Graduação, CORDEIRO, T F K, (2009). [4]

3.2.1 Mestre

O programa Mestre é que gerencia a rede. Ele é responsável por requisitar as tarefas específicas para cada módulo na rede, e por armazenar as informações sobre o estado atual dos mesmos em uma estrutura chamada *Module*.

As primeiras três funções que aparecem são *SimpleMsgBox*, *ErrorPrinting* e *ErrorMsgBox*, que controlam a exibição de mensagens do programa. A primeira é chamada para mostrar mensagens genéricas. A segunda controla se as mensagens de erro devem ou não serem exibidas. A terceira mostra mensagens de erro.

As próximas funções, de *SioOpen* até *ErrorShow* estão relacionadas com a comunicação serial do computador, incluindo inicialização e finalização da porta de comunicação, envio e recebimento de dados, esvaziamento do buffer de dados recebidos e impressão de possíveis erros relacionados à porta. Estas funções são internas ao funcionamento do programa, e não precisam ser chamadas diretamente pela função *Main*.

A função *InitiVars* é responsável por inicializar a estrutura correspondente de cada módulo. Nesta estrutura, temos as variáveis do tipo byte *GADDR*, *SDEF*, *STAT*, *ID*, *VER*, *TYPE*, e *KEYBOARD[24]*, para armazenar, respectivamente, o endereço de grupo, o status itens, o status, o número de identificação e a versão do módulo, e as informações das teclas. Esta função inicializa as estruturas com a configuração *defaut*.

As informações que cada módulo deve enviar no status devem ser definidas na função principal.

3.2.1.1 Função GET_RESPONSE

A função *GET_RESPONSE*, ilustrada pela Fig. 3.3, é responsável por receber o Status do escravo e armazenar os dados obtidos na estrutura correspondente. Para isto, ela consulta o *Status Itens* do escravo em questão, *SDEF*, e faz um "*and*" *bit* a *bit* deste com as definições *SEND_ID_VER* e *SEND_KEYBOARD,* para saber se é para receber, respectivamente, os números de identificação e versão e as informações das teclas. Para receber o Status, chama-se a função *SioGetChars* para ler da porta serial. A função *SioGetChars* salva os *bytes* no vetor *comm_string[29].* Depois de ler da porta serial, verifica se a quantidade de *bytes* recebida é condizente e se o *checksum* está correto. Depois disto, os *bytes* são transferidos para a estrutura, e por último, é feito um "*and*" *bit* a *bit* de *STAT* com a definição *Com_error* para saber se o escravo encontrou erro de *checksum* logo após receber e montar o comando.

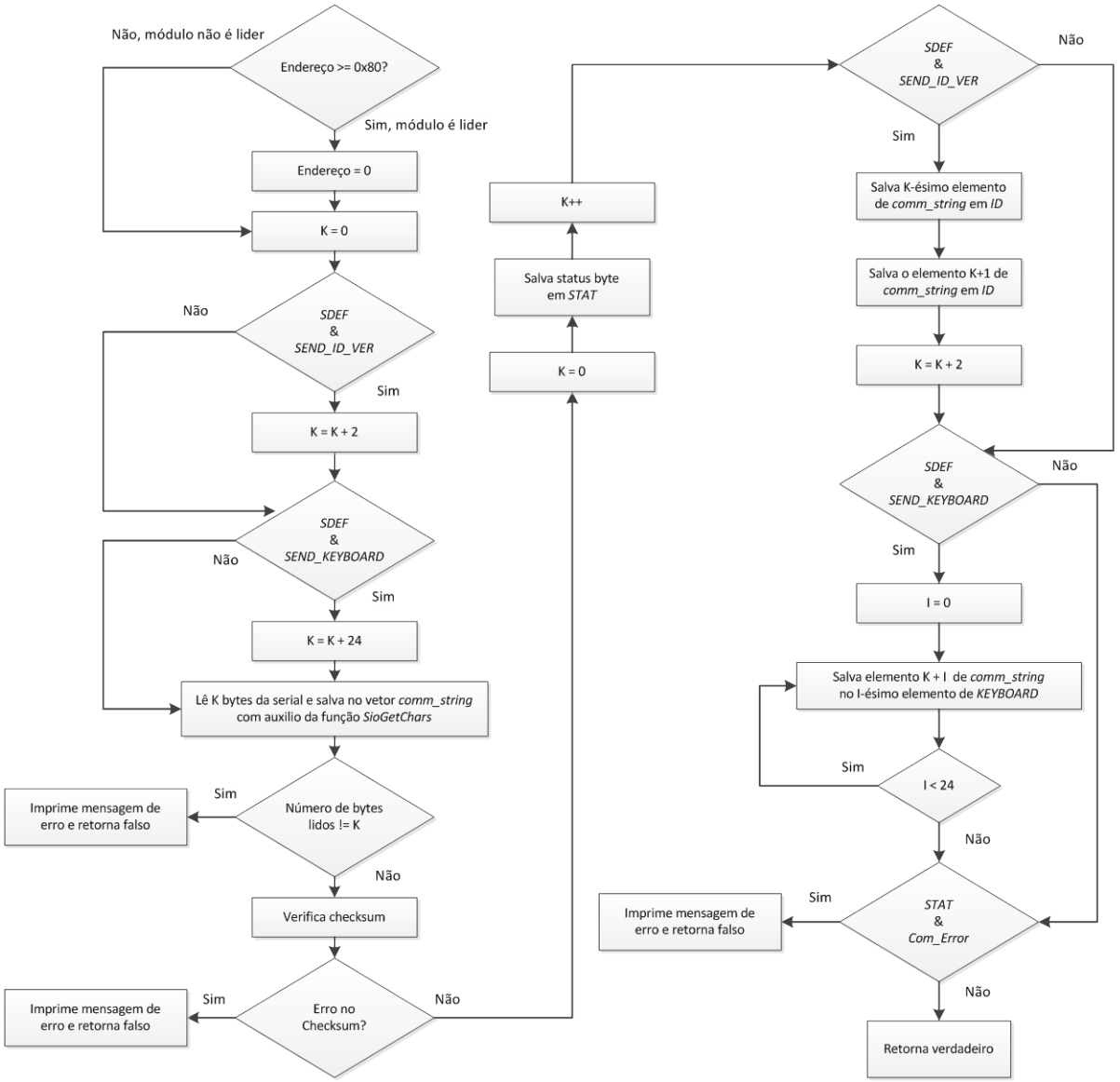

Figura 3.3. Função GET_RESPONSE.

3.2.1.2 Função M_SET_STATUS

A função *M_SET_STATUS* é responsável por enviar ao módulo em questão o comando *Set Status*. Ela recebe o endereço do módulo e o *Status Itens*, e com estas informações, monta o comando no vetor *comm_string* e o envia ao módulo por meio da função *SioPutChars*. Depois de enviar o comando ao escravo, salva o *Status Itens* na variável *SDEF* da estrutura correspondente ao módulo. Depois de alguns milissegundos, chama-se a função *GET_RESPONSE* para receber o status. A Figura 3.4 ilustra a função.

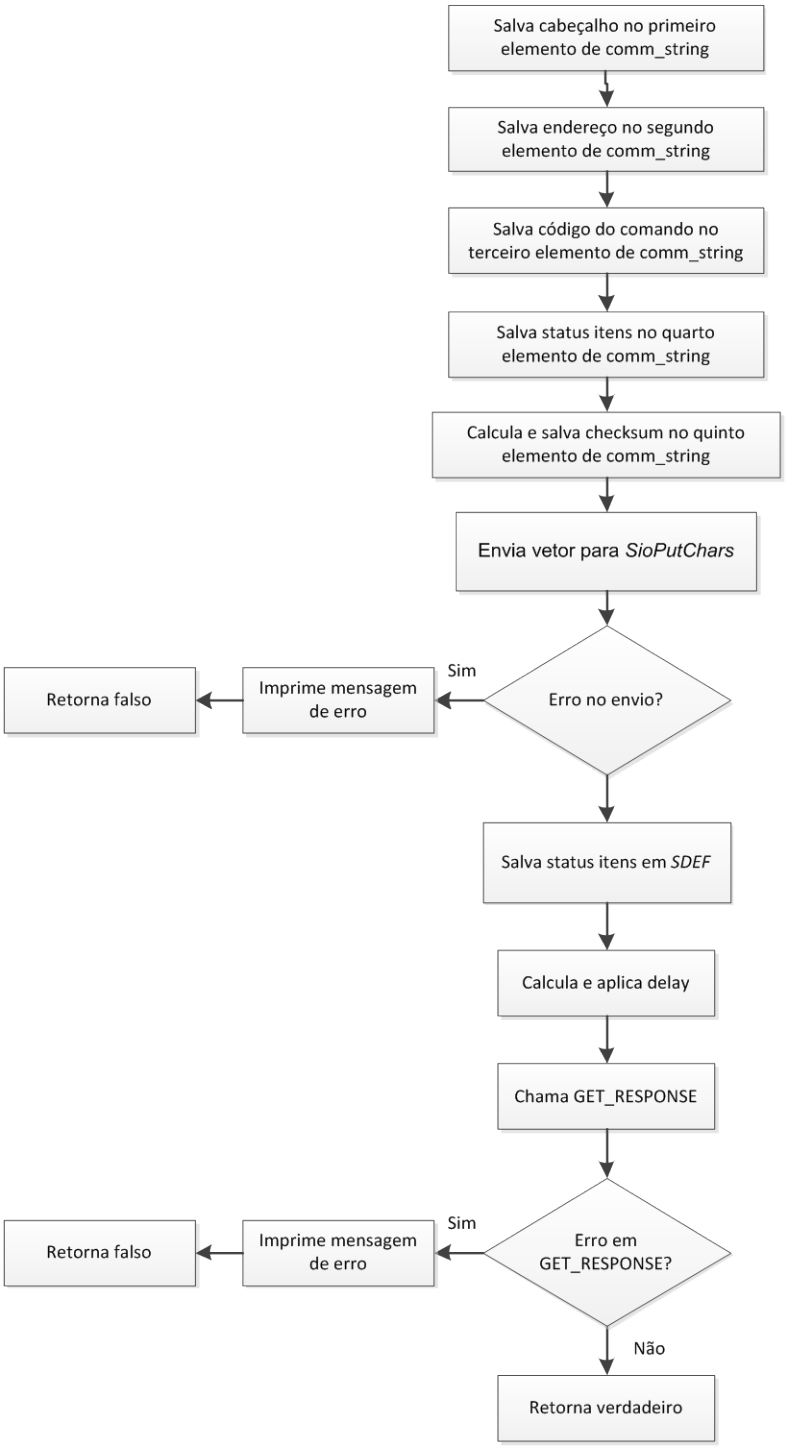

Figura 3.4. Função M_SET_STATUS.

3.2.1.3 Função M_READ_STATUS

A função *M_READ_STATUS* tem o papel de enviar o comando *Read Status* ao escravo. Alem de enviar o *Status Itens* para o escravo, é preciso acessar a estrutura do módulo em questão, e modificar o *Status Itens* de acordo com o comando, para que, quando a função *GET_RESPONSE* for chamada, ela saiba que informações e quantos *bytes* terá que ler. Para isto, é preciso salvar o *Status Itens* atual, *SDEF,* na variável *temp*, e, depois da recepção do status via *GET_RESPONSE*, restaurar o *Status Itens* anterior. A Figura 3.5 ilustra a função.

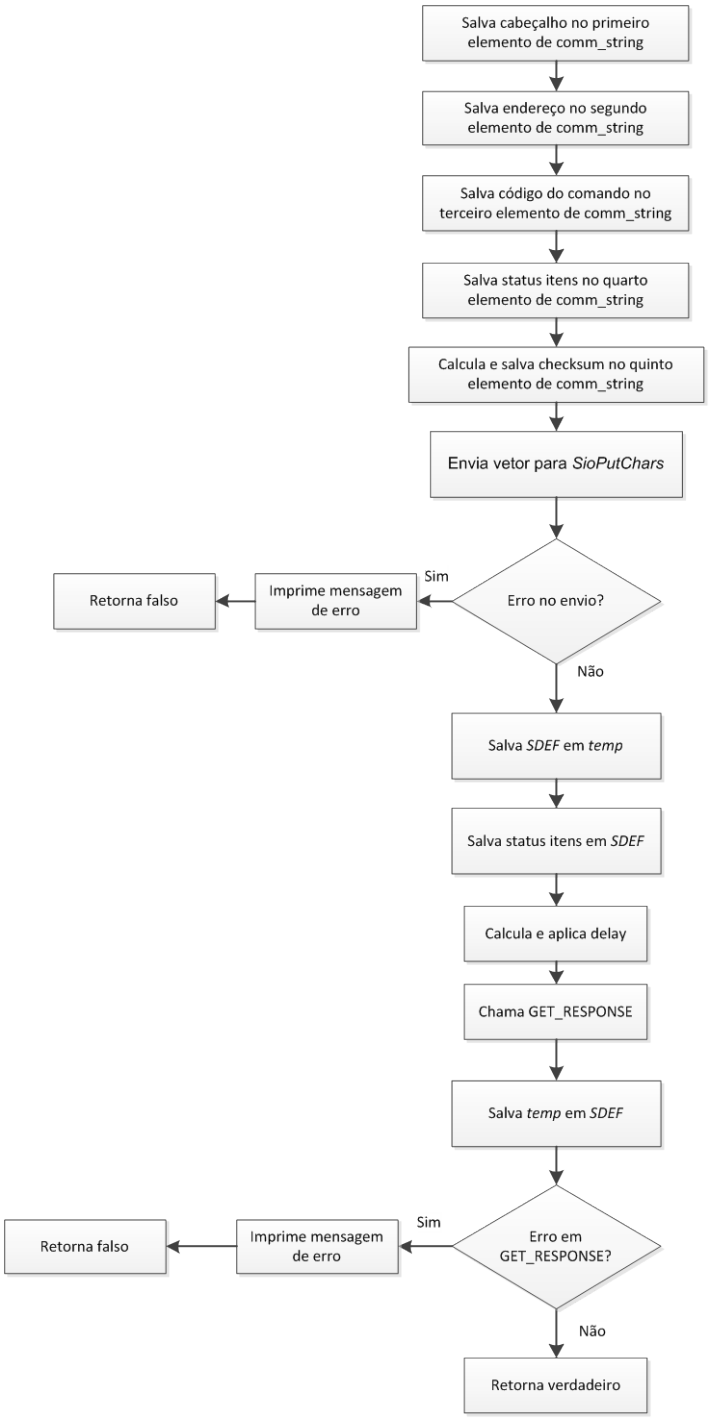

Figura 3.5. Função M_READ_STATUS.

3.2.1.4 Função M_NO_OP e ARCONDICIONADO

A função que envia o comando No Op funciona de maneira semelhante às funções *M_SET_STATUS* e *M_READ_STATUS*. Para o comando relacionado ao ar condicionado, a função *ARCONDICIONADO* funciona da mesma maneira que as funções de comando ilustradas anteriormente. A Figura 3.6 ilustra a função.

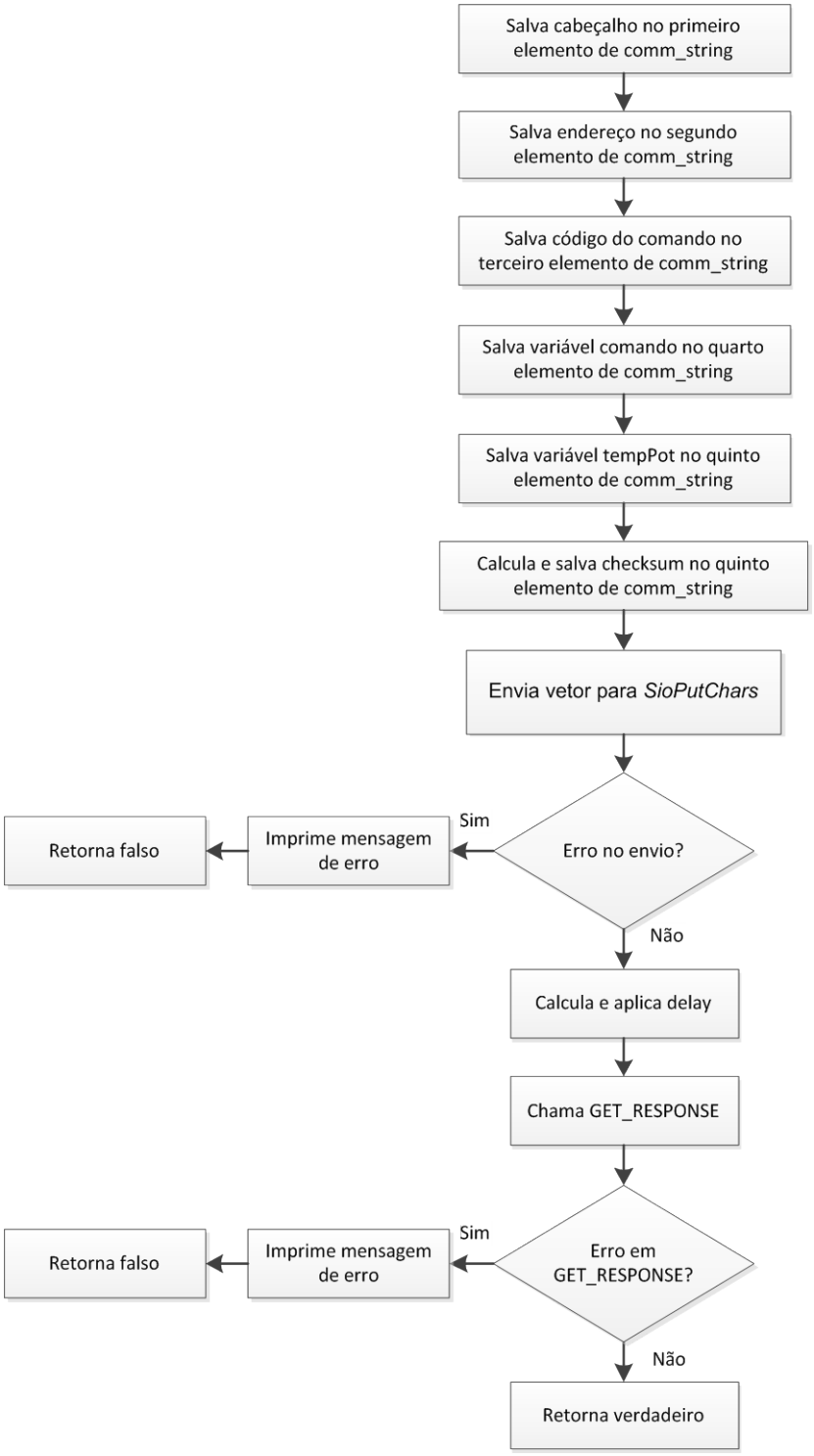

Figura 3.6. Função ARCONDICIONADO.

3.2.2 Escravo

O *software* do escravo possui dois arquivos: escravo.h e escravo.c. O arquivo escravo.h, contem algumas definições e variáveis. As funções utilizadas no arquivo escravo.c serão detalhadas mais a frente, e as variáveis declaradas neste arquivo serão detalhadas na explicação das funções que fazem uso delas.

As primeiras definições são *TESTE_1MHZ* e *TESTE_8MHZ*, que determinam a frequência do *clock* a ser utilizada no microcontrolador, sendo que duas delas devem ser comentadas. A próxima definição que se segue é *tamanhoBufferSerial,* que determina o tamanho do buffer da comunicação serial em 32 bytes.

Para cada uma das frequências de *clock* possíveis, pode se ver a definição *CONSTANTE_TEMPO,* que determina quantos ciclos o *TIMER/COUNTER0* deve executar para efetivar um tempo de 512µs. Também pode se ver as definições *br_4800*, *br_9600*, *br_19200* e *br_3800*, onde uma delas é escolhida para auxiliar a função de inicialização da USART a escolher os valores apropriados para o registrador UBRR, necessário para a configuração da velocidade da comunicação serial.

A seguir, as definições usadas para criar novos tipos de variáveis: *uint8* para *unsigned char, int8* para *signed char*, *uint16* para *unsigned int*, *int16* para *int*, *uint32* para *unsigned long*, *int32* para *long*, *bool* para *char*, *true* para 1, e *false* para 0.

As próximas definições são utilizadas para o funcionamento do protocolo de comunicação. *Header* é utilizada para o cabeçalho, *checksumError* para o erro de *checksum*, *sendDeviceTypeVersion* para indicar se o módulo deve retornar o número de identificação e versão e *sendKeyboard* para indicar que o módulo deve retornar as informações do teclado.

Logo em seguida, aparecem algumas variáveis do tipo *enum*: *estadoComunicacao* e *destinoComunicacao*. Ambos funcionam como uma máquina de estados, sendo que o primeiro determina os estados possíveis para a recepção dos bytes do comando e o segundo determina os destinatários possíveis. Por exemplo: Se *estadoComunicacao* se encontra em *aguardandoEndereco*, então o módulo fica a espera da chagada do byte de endereço. E se o endereço for diferente do endereço do módulo e diferente do endereço global, então o estado de *destinoComunicacao* é forçado em *destinoOutro*, para que, então, o módulo descarte os próximos bytes do comando.

3.2.2.1 Programa Principal

A função Programa Principal, ilustrada pela Fig. 3.7, começa com a chamada de algumas funções para configuração de hardware e inicialização de variáveis, e depois disto, entra em um laço infinito. Neste laço ocorrem várias questões importantes, como a interpretação de comandos vindos da serial. Dentro desse laço, são exigidos alguns milissegundos para que

o programa esteja livre para executar o protocolo e as funcionalidades varredura do teclado e o acionamento do ar condicionado. Consegue-se esse tempo após a passagem de n vezes do timer de 512µs, onde n é determinado pela variável *stepRateMultiplier*. A escolha foi de um tempo de 5.1 ms, esta variável deve ser inicializada e mantida igual a 10.

A primeira ação que a função faz ao entrar no laço é verificar se há um novo byte vindo da porta serial. Se houver, a função *montaVetorComando* é chamada para se montar o vetor do comando a ser executado. Depois disto, a *flag iniciouNovociclo* é verificada para se saber se já se passaram os 5.1 ms exigidos para a execução das demais funções. Se não, então o laço termina. Se já se passaram os 5.1 ms, a *flag* é desativada e a *flag recebeuNovoComando* é verificada para saber se já chegou um comando.

Se o comando chegou, a *flag* é desativada e então é verificado o *checksum*, e caso o *checksum* esteja correto, então é chamada a função *interpretaComando* para saber qual é o comando e executá-lo. Depois de executar o comando a *flag responderComando* é verificada para saber se o escravo deve responder ao comando. Ou seja, se o comando é mesmo para o escravo em questão. Depois, verifica a flag *controleIR* para identificar a necessidade de modular o sinal infra vermelho. Por último, é chamada a função *varredura* para ler e salvar a última linha excitada do teclado excitar a próxima linha.

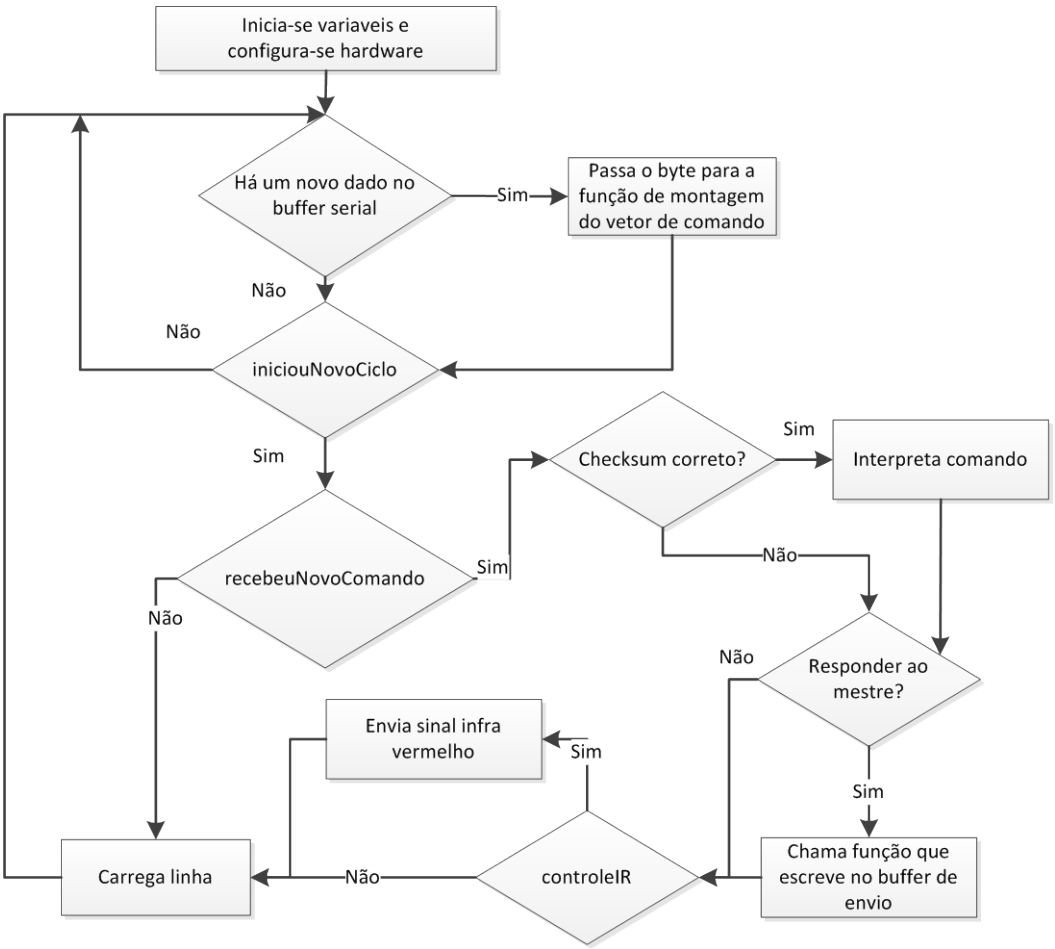

Figura 3.7. Função do programa principal

3.2.3 Configurações básicas de *hardware*

A função *iniciaUSART* configura a comunicação serial. Escolhe-se o valor do registrador UBRR de acordo com as definições do *clock* e do *baud rate* previamente selecionadas. Habilita-se a leitura e escrita. O formato do *byte* é de 8 bits sem paridade e 1 *bit* de *stop bit*.

A função *iniciaPortas* configura a porta B (controle dos multiplexadores), a porta C (controle dos decadificadores), e a porta D (saída dos multiplexadores, escrita e leitura serial, e Tx enable).

A função *iniciaPWM* configura o *TIMER/COUNTER1* para gerar uma onda de 38 kHz com ¼ de ciclo de trabalho na porta 16(PB2).

A função *iniciaVariaveis* inicia as variáveis necessárias para a execução do protocolo de comunicação, e as variáveis linha e coluna, necessárias para a varredura do teclado.

A função *iniciaTempoServo* configura o *Prescaler* de acordo com a frequência utilizada para que o *TIMER/COUNTER0* funcione corretamente, e em seguida, configura a máscara de interrupções do overflow.

3.2.4 Comunicação serial

Duas interrupções foram utilizadas, uma para a recepção, que está sempre ativada e ocorre quando um *byte* chega pela serial, e outra para o envio. Foram utilizados dois *buffers* circulares, um para armazenar os *bytes* que chegam na recepção serial e outro para guardar os *bytes* a serem enviados. Cada *buffer* possui dois ponteiros, um de início de *buffer* e um de fim de *buffer*. Quando o ponteiro de início é igual ao de fim, não existem mais dados no *buffer*. As macros *existeDadosBufferRX* e *existeDadosBufferTX* testam se os *buffers* de recepção e de envio estão vazios. Quando um ponteiro se torna igual ao tamanho do *buffer*, este é feito igual a zero.

Na recepção, quando um *byte* chega ao registrador UDR, é disparada a interrupção de recepção, onde este *byte* é salvo no *buffer* de recepção. Depois de salvar o *byte* no *buffer*, o ponteiro de fim de *buffer* é então incrementado ou é feito apontar para o início do *buffer*. A Figura 3.8 ilustra a recepção da comunicação serial para o *buffer* de leitura.

Em momentos apropriados, o programa verifica a macro *existeDadosBufferRX* para saber se existem dados no *buffer* de recepção. Quando existem dados, estes podem ser lidos através da função *leDadoSerial*. Esta função lê o *byte* do *buffer*, incrementa o ponteiro de início de *buffer*, e retorna os *bytes* para serem mais tarde manipulados. A Figura 3.9 ilustra a função.

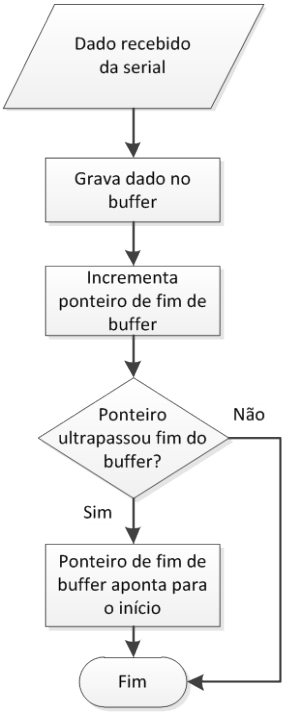

Figura 3.8. Recepção da comunicação serial.

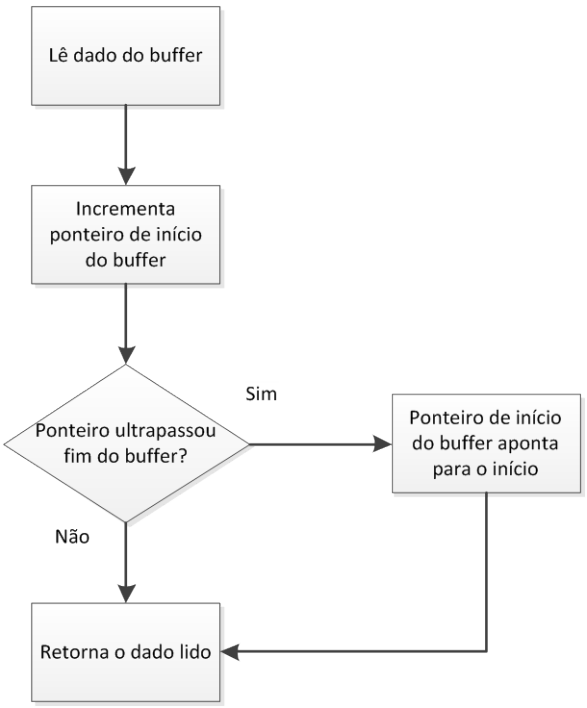

Figura 3.9. Recebimento do dado do buffer de recepção para o programa principal.

Quando o programa principal precisa enviar o vetor de status para o mestre, os *bytes* precisam ser enviados um de cada vez para o *buffer* de envio da serial. Para cada byte do vetor, o programa principal então chama a função *escreveDadoSerial*, a qual é responsável por gravar o *byte* no *buffer* de envio incrementar o ponteiro de fim de *buffer*, habilitar o MAX 485, e ativar a interrupção de envio. A Figura 3.10 ilustra a função.

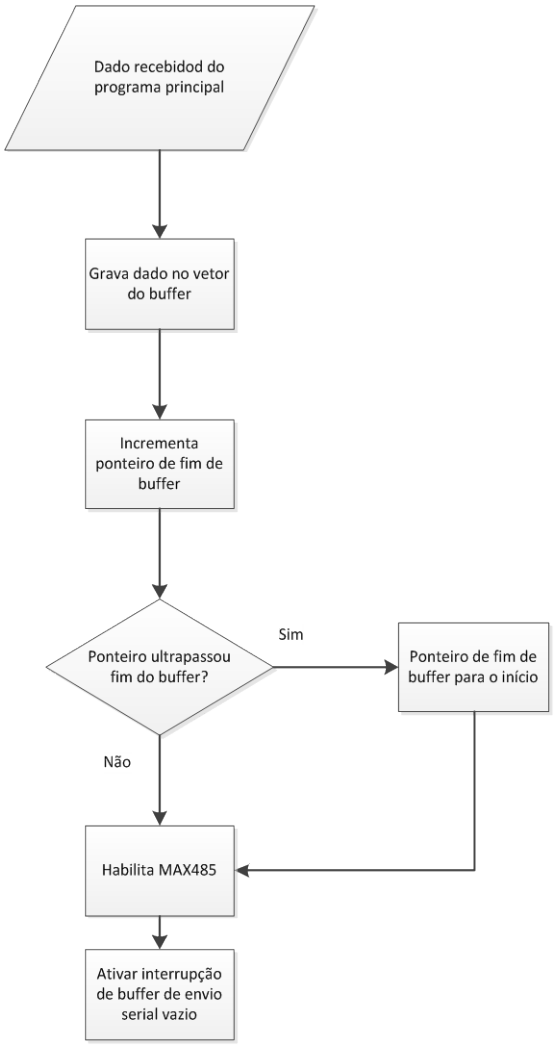

Figura 3.10. Recebimento do dado do programa principal para o buffer de envio da serial.

Se não há mais dados no registrador UDR, e se a interrupção de envio estiver ativada, acontece a interrupção de envio. Nessa, uma rotina é responsável por salvar o *byte* no registrador UDR, incrementar o ponteiro de início do *buffer*, e chamar a macro *existeDadosBufferTx*, para fazer o teste para saber se o *buffer* está vazio. Somente depois que o *buffer* de envio estiver vazio e depois de a transmissão ter sido completada, é que a macro desabilita a interrupção de envio, a *flag* de transmissão completa (TXC) do registrador de controle de status A(UCSRA), e o MAX 485. A Figura 3.11 ilustra a interrupção de envio.

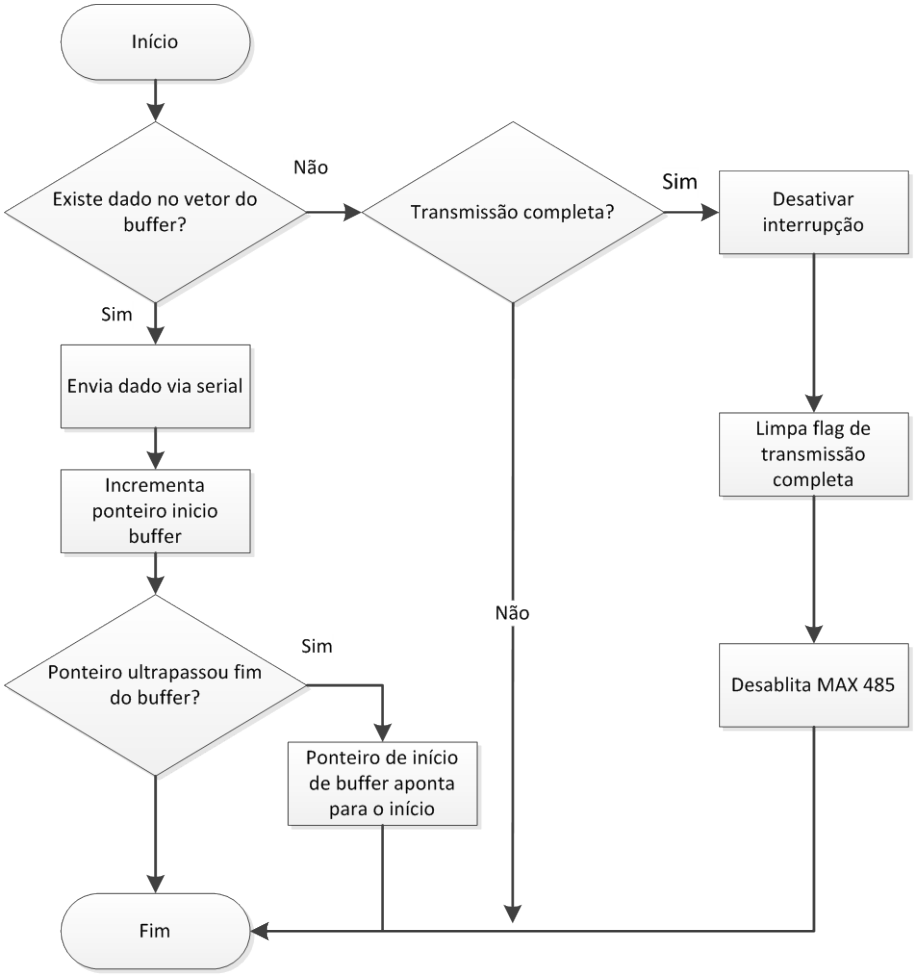

Figura 3.11. Interrupção de envio da comunicação serial.

3.2.5 Recepção, interpretação e resposta aos comandos

As funções deste grupo manipulam os *bytes* vindos das funções de comunicação serial, montam, interpretam e executam os comandos, e enviam o status para o mestre.

A primeira função deste grupo é a *montaVetorComando*, quem recebe os *bytes* vindos da comunicação e, a partir deles, monta o vetor-comando. Neste vetor, não estão os *bytes* do cabeçalho e do endereço. Esta função é chamada cada vez que um *byte* é recebido da serial. A Figura 3.12 ilustra a função.

Esta função funciona como uma máquina de estados. Os estados possíveis estão em uma variável do tipo *enum*, e são: *aguardandoCabecalho*, *aguardandoEndereco*, *aguardandoComando* e *aguardandoBytesDados*. A variável enum destino juntamente com a variável global *enderecoRecebido* e a *flag recebendoBytesOutro* complementam a definição da máquina de estados, criando assim um estado alternativo onde os *bytes* recebidos são descartados. Isto acontece quando o comando não é para o módulo em questão. Como a função precisa ser chamada várias vezes para completar a montagem do vetor de comando,

as variáveis de estado devem ser declaradas como estáticas, pois, se assim não fosse, a máquina de estados não sairia do estado inicial.

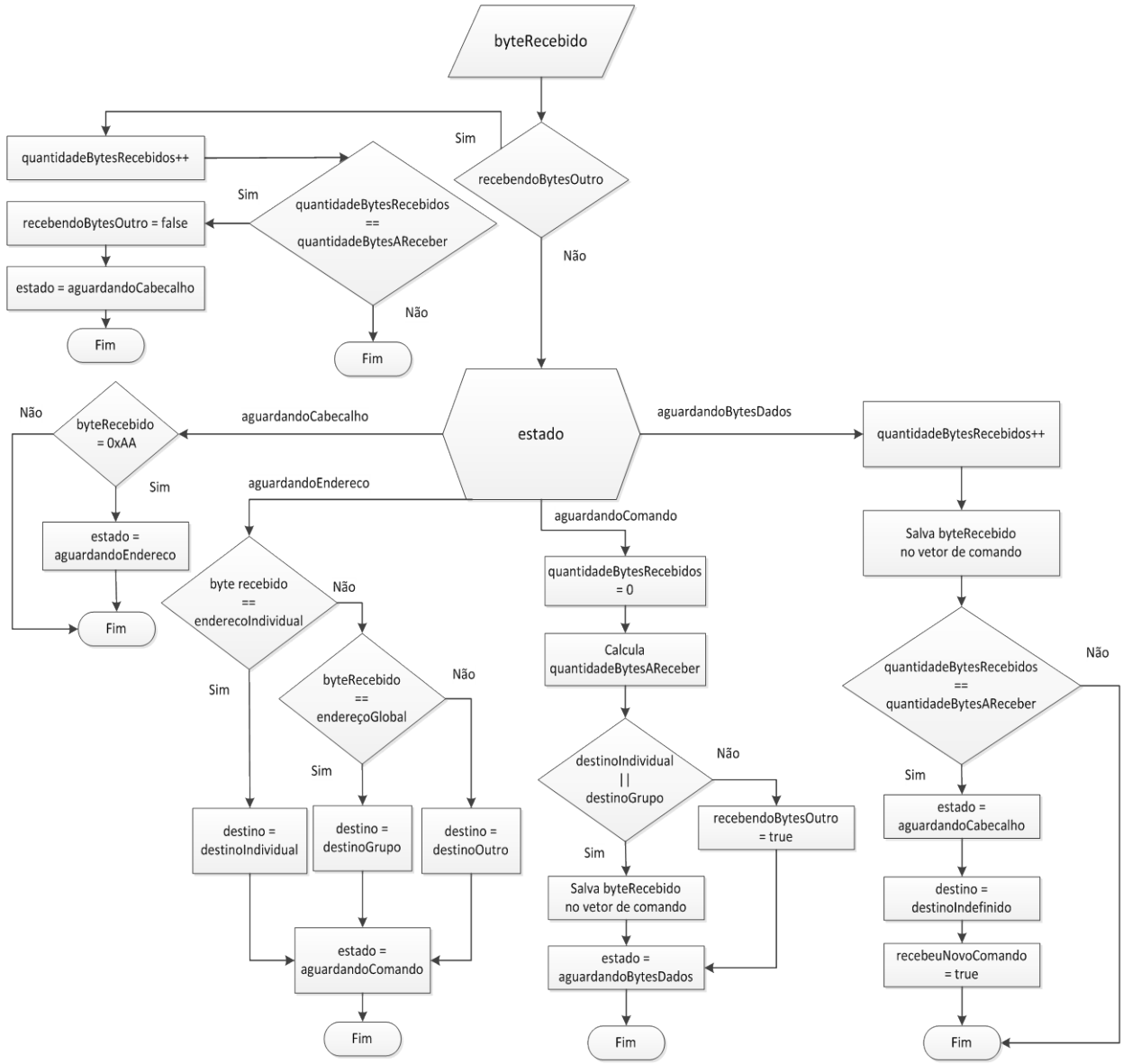

Figura 3.12. Função montaVetorComando.

Esta função funciona como uma máquina de estados. Os estados possíveis estão em uma variável do tipo enum, e são: *aguardandoCabecalho*, *aguardandoEndereco, aguardandoComando* e *aguardandoBytesDados*. Outra variável enum destino juntamente com a variável booleana *recebendoBytesOutro* complementam a definição da máquina de estados, criando assim um estado alternativo onde os bytes recebidos são descartados. Isto acontece quando o comando não é para o módulo em questão. Como a função precisa ser chamada várias vezes para completar a montagem do vetor de comando, as variáveis de estado devem ser declaradas como estáticas, pois, se assim não fosse, a máquina de estados não sairia do estado inicial.

No estado *aguardandoCabecalho,* a função espera pelo *byte* de cabeçalho. A transição para o próximo estado é feita somente se o *byte* recebido é igual a 0xAA. O segundo estado é o *aguardandoEndereco.* Neste estado, o byte que chega é verificado para saber se é o endereço do módulo. O próximo estado é o *aguardandoComando.* Este estado é executado independente se o comando é ou não para o módulo. Isto se deve ao fato de que não se sabe ainda quantos *bytes* terão que ser descartados ou montados no vetor de comando. A metade superior do *byte* recebido indica a quantidade de *bytes* adicionais a esperar e a metade inferior contém o código do comando. Caso o comando seja para o módulo, o *byte* recebido é salvo na primeira posição do vetor. Caso não seja, a *flag recebendoBytesOutro* é ativada, para que mais tarde os *bytes* recebidos sejam descartados.

O próximo estado é o *aguardandoBytesDados.* Este estado só é executado se o comando é para o módulo. Neste estado, o *byte* recebido é salvo no vetor, e este é incrementado. Depois disto, é feita a comparação entre as variáveis *quantidadeBytesRecebidos* e *quantidadeBytesAReceber* para determinar se o comando já foi salvo completamente no vetor. Este estado é executado quantas vezes for necessário para montar o vetor de comando, e, na última vez que é executado, a *flag recebeuNovoComando* é ativada, indicando que o comando está montado. Por último, as variáveis de estado são reinicializadas.

A segunda função, *verificaCheckSum*, soma os elementos do vetor comando e o endereço para verificar se o *checksum* está correto.

A próxima função é *interpretaComando*. Esta faz um "*and*" *bit* a *bit* do primeiro elemento do vetor de comando com 0x0F para retirar a metade inferior, necessária para saber o tipo do comando. A Figura 3.13 ilustra a função.

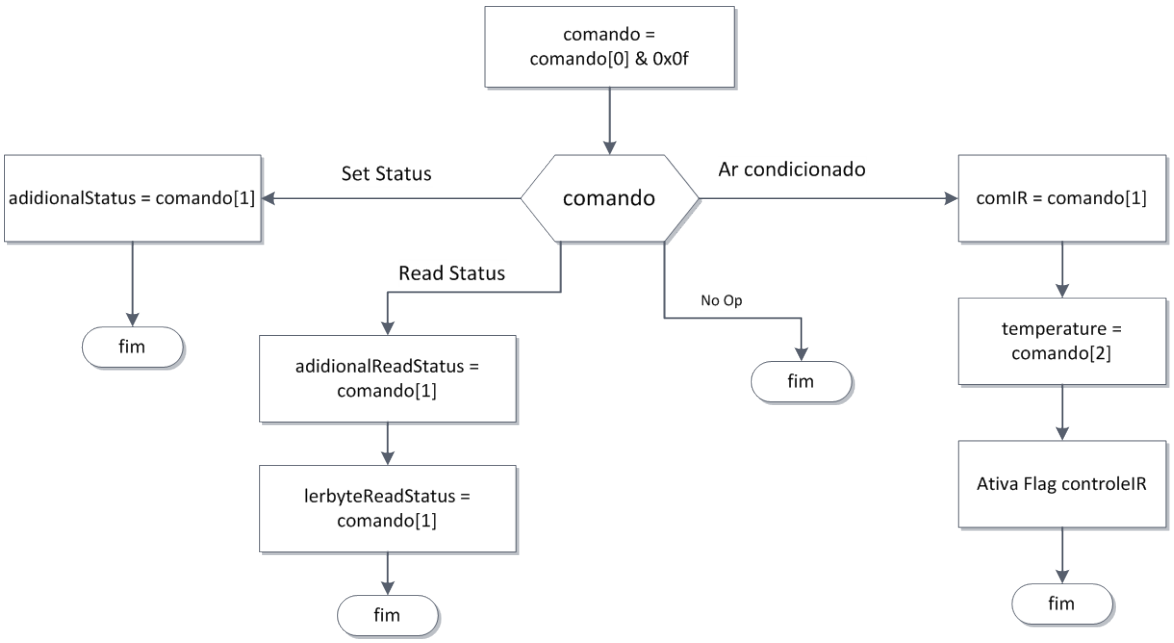

Figura 3.13. InterpretaComando.

A próxima função é *enviaDadosResposta*. Esta função tem o papel de montar o status e enviá-lo para o buffer da comunicação serial. A Figura 3.14 ilustra a função. Esta faz o uso da *flag lerByteReadStatus* para diferenciar se o status será montado de acordo com o *Status Itens* recebido pelo comando *Define Status*, que foi salvo previamente na variável *additionalReadStatus*.

A variável local st recebe um dos possíveis Status Itens de acordo com a *flag lerByteReadStatus*, e seus bits são verificados para determinar quais dados serão enviados para a função *escreveDadoSerial*, que é responsável por armazenar os dados no *buffer* de envio da serial. O primeiro *bit* indica que é para enviar os números de identificação e versão do módulo e o quinto que é para enviar as informações do teclado. Os outros *bits* estão disponíveis para configuração. A verificação destes *bits* é feita através de um "*and*" *bit* a *bit* da variável com as definições *sendDeviceTypeVersion* e *sendKeyboard*.

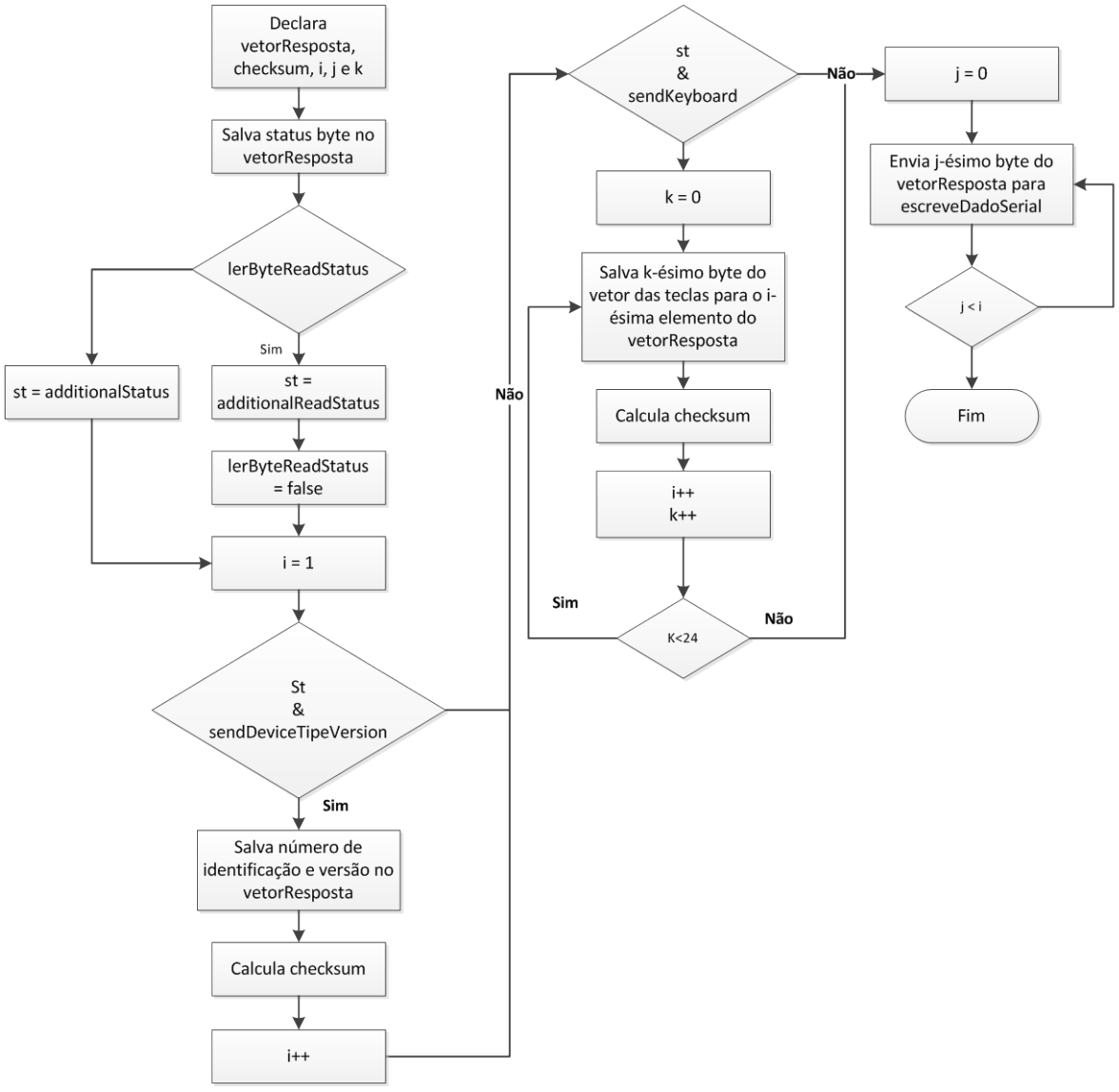

Figura 3.14. Função enviaDadosResposta.

A função varredura lê uma linha do teclado, transfere esta linha para o vetor *keyboard*, e excita a próxima linha. Para ler o valor de uma coluna, é preciso selecionar a entrada específica do multiplexador – o que é feito com auxilio da porta B –, e ler a saída do multiplexador – o que é feito através dos pinos AIN0 e AIN1 da porta D – e aplicar uma máscara ao registrador PIND para filtrar os valores lidos nos demais pinos da porta. Para o controle do multiplexador, são usados os 3 primeiros pinos da porta B. As primeiras 8 colunas são lidas via AIN0, e as demais via AIN1. Duas variáveis temporárias são usadas para armazenar os valores lidos da porta enquanto todas as colunas ainda não foram lidas. A primeira, *linhaLow*, é usada para receber os valores lidos das 8 primeiras colunas, enquanto que a segunda, *linhaHight*, para receber os valores lidos das 4 últimas. Cada *bit* destas variáveis deve receber o valor da respectiva coluna. Sendo assim, o procedimento de leitura é repetido 12 vezes, uma para cada coluna, com um *delay* de 12 ciclos de *clock* entre cada. Por último, os valores previamente armazenados em *linhaLow* e *linhaHight* são transferidos para os índices n e n+1 do vetor *keyboard*, onde n é igual a linha. Ou seja, para armazenar as 12 linhas – o que é feito ao longo de 12 chamadas desta função e com um *timer* mínimo de 5,1 ms entre elas –, são necessários 24 bytes do vetor. Depois de ler e salvar as 12 colunas, a porta C é incrementada, fazendo com que a próxima linha seja excitada. A Figura 3.15 ilustra a função.

A função responsável por enviar o sinal infravermelho é *ir*. O *ir* recebe as informações desejadas de alteração do ar-condicionador. Assim, ativa-se a contagem do *timer* 1 do microcontrolador. Verifica-se qual o comando desejado, carregando o código IR a ser enviado. Chama-se função de modulação. Modula-se primeiro o sinal AGC. Após essa ação, há a leitura de cada bit e sua devida modulação, com o envio do sinal pela Porta 15 (PB1).

O primeiro conjunto de sinal enviado é o pulso ACG de 8,4 ms seguido de um espaço de 4,2 ms. Após isso, identifica-se o valor do bit (“0” ou “1”) para realizar a modulação devida de cada sinal. Ao término do envio dos 29 bit´s, desliga-se o contador timer 1.

Figura 3.15. Função varredura.

CAPÍTULO 4 – TESTES E VALIDAÇÃO

Realizamos a identificação do sinal infravermelho e dois conjuntos de teste. A análise do protocolo infravermelho emitido pelo controle remoto do o ar-condicionado TNSC2425TL0, marca LG, realizamos com um osciloscópio. Quantos aos conjuntos de teste, o primeiro foi verificar o funcionamento do sensoriamento da mesa. Já, o outro conjunto de teste corresponde ao funcionamento na emissão do sinal infravermelho do ar-condicionado.

4.1 CONFIGURAÇÃO DO HARDWARE E SOFTWARE

Buscamos verificar o impacto da configuração do oscilador interno dos escravos na comunicação serial. Configuramos o programa tanto em 1 MHz como em 8MHz no microcontrolador do circuito da mesa e do ar condicionado respectivamente. Os endereços das placas são fixos dentro do programa do escravo. O microcontrolador do sensoriamento da mesa possui o endereço 1 e do controle remoto, o endereço 2. Ajustamos a comunicação em 4800 Kbps. Colocamos ambas as placas no barramento. Constatamos não haver impacto na comunicação e no funcionamento de ambos os escravos.

4.2 SENSORIAMENTO DA MESA INTERATIVA

Por meio do programa mestre, enviamos o comando DEFINE STATUS, escolhendo a opção mesa. O módulo escravo envia ao mestre o estado da mesa que é apresentado numa matriz 12 X 12, conforme Figura 4.1. Quando acionamos um botão da mesa, há uma mudança no valor do elemento da matriz correspondente, que representa o acionamento do botão da mesa. Podemos verificar a diferença no elemento na linha 1 – coluna 1.

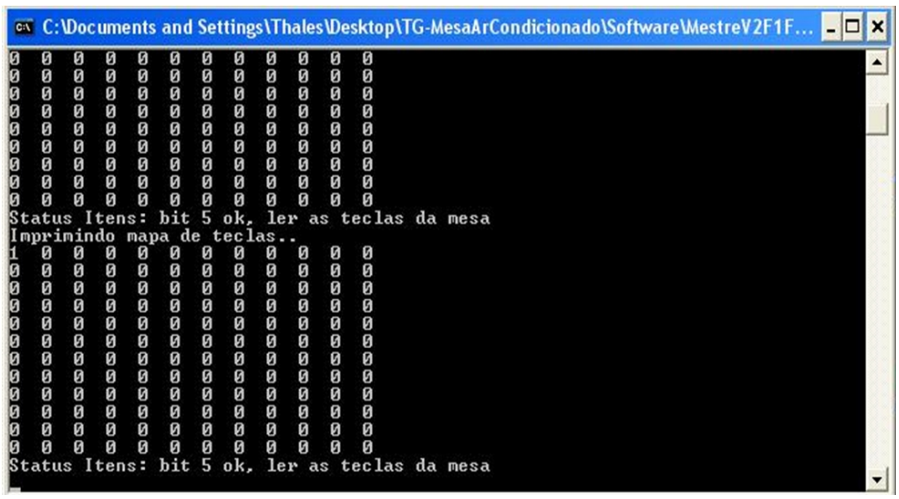

Figura 4.1. Resposta do escravo ao acionamento do teclado da mesa interativa

4.4 EMISSÃO DE PROTOCOLO INFRAVERMELHO

Apresentamos a identificação do protocolo infravermelho do controle remoto do ar-condicionado TNSC2425TL0, marca LG, realizado com o uso de um osciloscópio e um circuito integrado AVR LAB. O AVR LAB possui um receptor que possibilitou captar o sinal infravermelho do controle remoto e monitorá-lo no osciloscópio, conforme a Fig. 4.2. Lembramos que o sinal no receptor apresenta-se de forma invertida do sinal gerado no emissor.

O osciloscópio também permitia coletar os valores dos gráficos tabulados com a tensão e o tempo respectivo. Capturamos os sinais conforme a Tab. 4.1 que possui a seguinte linha de raciocínio:

- Primeiro, definimos os conjuntos das funções de interesse (liga, desliga, jetcool, mudança de temperatura e mudança de ventilação) presente no controle remoto.
- Percebemos que o controle tem uma memória da última ação realizada que pode ser identificado no seu visor. Assim, escolhemos a situação de referência e emitimos o infravermelho variando uma variável.

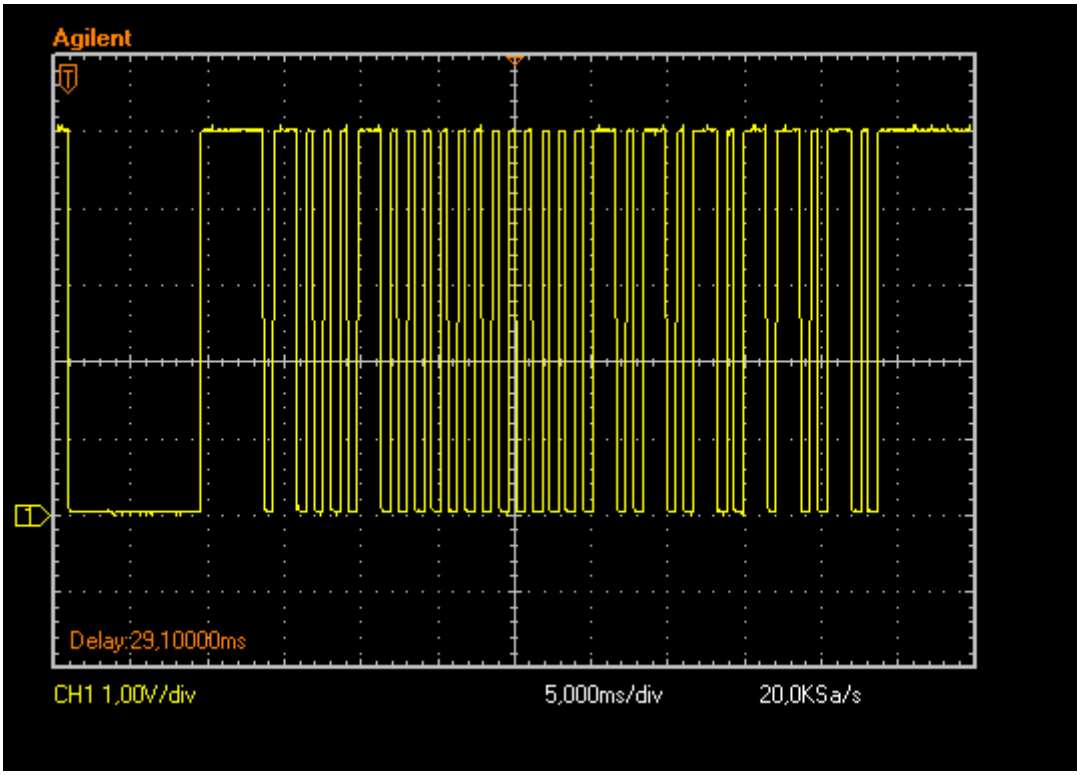

Figura 4.2. Sinal Infravermelho “Ligar” do controle remoto

Tabela 4.1. Ações da coleta de sinal do controle remoto do ar-condicionado

Nº	Situação de Referência	Mudança
1	Temperatura: 20 Ventilação: automática	Acionamento do comando Jet Cool
2	Temperatura: 20 Ventilação: automática	Mudança da ventilação para “LOW”
3	Temperatura: 20 Ventilação: “LOW”	Mudança da ventilação para “MED”
4	Temperatura: 20 Ventilação: “MED”	Mudança da ventilação para “HIGH”
5	Temperatura: 20 Ventilação: “HIGH”	Mudança da ventilação para automática
6	Temperatura: 18 Ventilação: automática	Mudança de temperatura para 19
7	Temperatura: 19 Ventilação: automática	Mudança de temperatura para 18
8	Temperatura: 19 Ventilação: automática	Mudança de temperatura para 20
9	Temperatura: 20 Ventilação: automática	Mudança de temperatura para 19
10	JET COOL	Desacionamento da função JET COOL (modo de funcionamento mais intenso do ar-condicionado) leva o funcionamento para temperatura em 18 e ventilação em “HIGH”.
11	Temperatura: 19 Ventilação: automática	Ativa a função “desliga”
12	Temperatura: 20 Ventilação: automática	Ativa a função “desliga”
13	Temperatura: 19	Ativa a função “liga”

Nº	Situação de Referência	Mudança
	Ventilação: automática	
14	Temperatura: 20 Ventilação: automática	Ativa a função “liga”
15	Temperatura: 21 Ventilação: automática	Ativa a função “liga”

Para a análise, identificamos a variação de tensão e tempo relacionado. Verificamos a diferença no tempo como pode ser visto na Tab. 4.2, que representa os dados da coleta do item 1 da Tab. 4.1. Com essa análise, anotamos que o formato de protocolo capturado se assemelha ao da Corporação JVC Kenwood, pois temos um trem de pulso da portadora em torno de 600 μs e os espaços variando aproximadamente aos tempos do valor lógico “1” e “0” desse protocolo.

Com base nisso, identificamos os valores coletados e traduzimos no formato de bit, chegando na Tab. 4.6. Avaliando a Tab. 4.6, chegamos à conclusão apresentada na Tab. 4.3 sobre os bit´s enviados.

Tabela 4.2. Valores do item1 da tabela 4.1

CH1 1,00V/div 5,000ms/div Size=1200 [Date:14/2/2012]						
NO.	Tensão [V]	Tempo [s]	Variação do tempo [s]	Modulação [s]	Valor do Bit	Identificação
190	0,00	0,00860	0,00855	0,00855		Header up
273	5,00	0,01275	0,00415	0,00415		Header down
286	0,00	0,01340	0,00060			
316	5,04	0,01490	0,00145	0,00205	1	Bit 0
330	0,00	0,01560	0,00065			
338	5,00	0,01600	0,00035	0,00100	0	Bit 1
351	0,00	0,01665	0,00060			
360	5,00	0,01710	0,00040	0,00100	0	Bit 2
373	0,00	0,01775	0,00060			
382	4,96	0,01820	0,00040	0,00100	0	Bit 3
395	0,00	0,01885	0,00060			
426	5,00	0,02040	0,00150	0,00210	1	Bit 4
439	0,00	0,02105	0,00060			
447	5,00	0,02145	0,00035	0,00095	0	Bit 5
460	0,00	0,02210	0,00060			
469	4,96	0,02255	0,00040	0,00100	0	Bit 6
483	0,00	0,02325	0,00065			

CH1 1,00V/div 5,000ms/div Size=1200 [Date:14/2/2012]						
NO.	Tensão [V]	Tempo [s]	Variação do tempo [s]	Modulação [s]	Valor do Bit	Identificação
491	4,96	0,02365	0,00035	0,00100	0	Bit 7
505	0,00	0,02435	0,00065			
513	5,00	0,02475	0,00035	0,00100	0	Bit 8
527	0,00	0,02545	0,00065			
535	5,00	0,02585	0,00035	0,00100	0	Bit 9
549	0,00	0,02655	0,00065			
557	5,00	0,02695	0,00035	0,00100	0	Bit 10
570	0,00	0,02760	0,00060			
601	5,00	0,02915	0,00150	0,00210	1	Bit 11
614	0,00	0,02980	0,00060			
623	4,96	0,03025	0,00040	0,00100	0	Bit 12
636	0,00	0,03090	0,00060			
644	5,00	0,03130	0,00035	0,00095	0	Bit 13
657	0,00	0,03195	0,00060			
666	5,00	0,03240	0,00040	0,00100	0	Bit 14
680	0,00	0,03310	0,00065			
688	5,00	0,03350	0,00035	0,00100	0	Bit 15
701	0,00	0,03415	0,00060			
710	5,00	0,03460	0,00040	0,00100	0	Bit 16
723	0,00	0,03525	0,00060			
732	4,96	0,03570	0,00040	0,00100	0	Bit 17
746	0,00	0,03640	0,00065			
754	5,00	0,03680	0,00035	0,00100	0	Bit 18
767	0,00	0,03745	0,00060			
776	4,96	0,03790	0,00040	0,00100	0	Bit 19
789	0,00	0,03855	0,00060			
820	4,96	0,04010	0,00150	0,00210	1	Bit 20
833	0,00	0,04075	0,00060			
841	5,00	0,04115	0,00035	0,00095	0	Bit 21
855	-0,04	0,04185	0,00065			
863	4,96	0,04225	0,00035	0,00100	0	Bit 22
876	0,00	0,04290	0,00060			
885	5,00	0,04335	0,00040	0,00100	0	Bit 23
899	0,00	0,04405	0,00065			
929	4,96	0,04555	0,00145	0,00210	1	Bit 24
942	0,00	0,04620	0,00060			
951	5,00	0,04665	0,00040	0,00100	0	Bit 25
964	0,00	0,04730	0,00060			
973	4,96	0,04775	0,00040	0,00100	0	Bit 26
986	0,00	0,04840	0,00060			
1016	4,96	0,04990	0,00145	0,00205	1	Bit 27
1030	0,00	0,05060	0,00065			
1200	5,00	0,05910	0,00845	0,00910	1	Bit 28

Tabela 4.3. Estrutura do protocolo infravermelho

BYTE DE ENDEREÇO								NIBLE COMANDO				NIBLE COMANDO				NIBLE TEMP				NIBLE VENT				NIBLE CHECK SUM				Bit Paridade
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	0	0	0	1	0	0	0	BIT DE DESLIGAMENTO			BIT DE ACIONAMENTO JET COOL	BIT DE MUDANÇA DE TEMPERATURA E/OU DE VENTILAÇÃO																

A seguir, a Tabela 4.4 traz as informações da temperatura.

Tabela 4.4. Valores de temperatura

	NIBLE TEMPERATURA			
	Bit 16	**Bit 17**	**Bit 18**	**Bit 19**
temp 18	0	0	1	1
temp 19	0	1	0	0
temp 20	0	1	0	1
temp 21	0	1	1	0
temp 22	0	1	1	1
temp 23	1	0	0	0
temp 24	1	0	0	1
temp 25	1	0	1	0
temp 26	1	0	1	1
temp 27	1	1	0	0
temp 28	1	1	0	1
temp 29	1	1	1	0
temp 30	1	1	1	1

A Tabela 4.5 traz as informações da temperatura.

	NIBLE VENTILAÇÃO			
Ventilação	**Bit 20**	**Bit 21**	**Bit 22**	**Bit 23**
LOW	0	0	0	0
MED	0	0	1	0
HIGH	0	1	0	0
AUTO	0	1	0	1
JET COOL	1	0	0	0

Tabela 4.5. Valores de ventilação

Tabela 4.6. Mapa dos bit´s dos dados coletados

	Pulso AGC		BYTE 0								BYTE 1								BYTE 2								NIBBLE			
nº	Ação	Header up	Header down	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
1	1819auto	0,0085	0,0041	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	1
2	1918auto	0,0085	0,0041	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	1
3	1920auto	0,00855	0,00405	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	1
4	2019auto	0,00855	0,0041	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	1	0	1	0	0	0	1	1
5	20autolow	0,0085	0,0041	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	1	1	0	1	1
6	20lowmed	0,0085	0,0041	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	1	0	1	1	1	1	1
7	20medhigh	0,00855	0,00405	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0	0	0	0	0	1	1
8	20highauto	0,0085	0,0041	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	1
9	20autojet	0,00855	0,00415	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1
10	jet18high	0,00855	0,00405	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	1	0	1	0	0	1	1	1	1	1
11	OFF19auto	0,0085	0,0041	1	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	1
12	OFF20auto	0,0085	0,0041	1	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	1	1
13	ON19auto	0,00855	0,00405	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	1	1	0	0	1	1
14	ON20auto	0,00855	0,00405	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0	1	1	0	1	0	1
15	ON21auto	0,0085	0,0041	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	1	1	1

Conforme explicitado no Capítulo 3, utilizamos o *timer 1* do ATMEGA8 modulamos o sinal infravermelho. O programa mestre enviou os comandos liga, desliga, mudança de temperatura e jet cool. Para demonstrar a modulação do sinal, os gráficos obtidos diretamente da porta 15 (PB1) do microcontrolador das funções liga e *jetcool*.

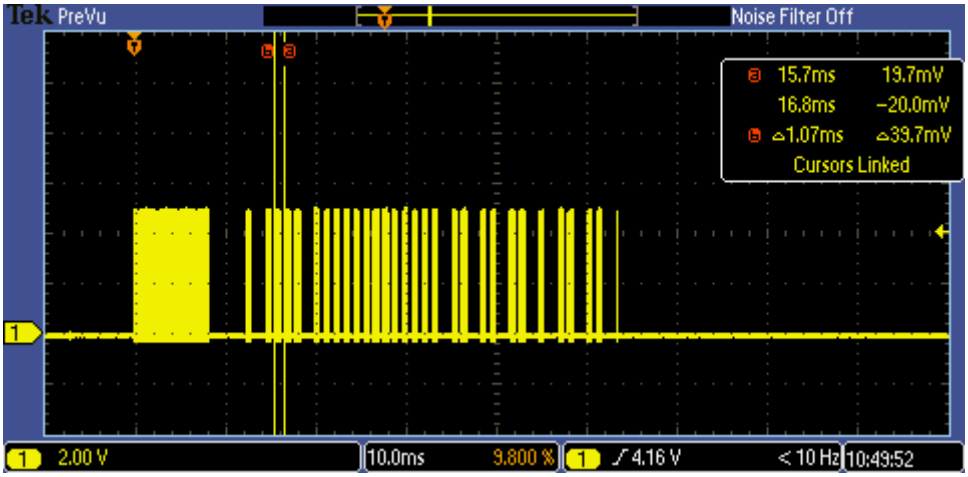

Figura 4.3. Função “Ligar”

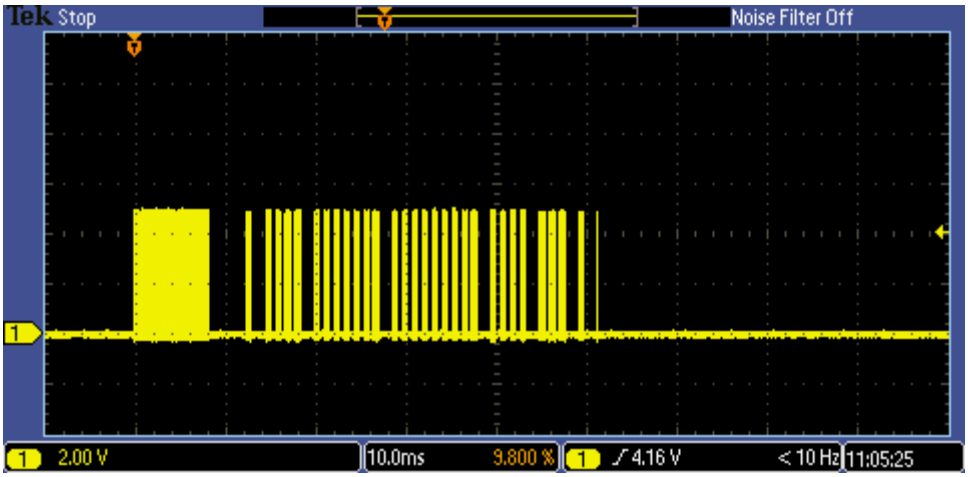

Figura 4.4. Função “Jet Cool”

Notamos que as Fig. 4.3 a 4.4 se assemelham com a Fig. 4.2. Após isso, realizamos o teste no próprio ar-condicionado e houve a resposta do equipamento conforme o esperado.

CAPÍTULO 5 - CONCLUSÃO

5.1 CONSIDERAÇÕES FINAIS

Desenvolvemos um projeto de sistema, composto de hardware e software, o qual monitora a mesa interativa e aciona remotamente via infravermelho o ar-condicionado, por meio de uma rede RS 485 de um computador mestre. O computador mestre envia o comando e recebe as respostas das ações dos escravos. Definimos o protocolo de comunicação e os *hardware´*s envolvidos para atendimento do trabalho.

Vimos que a comunicação funcionou conforme o esperado. O sistema foi capaz de identificar as mensagens destinadas para cada módulo, permitindo que os comandos desejados chegassem ao escravo e o recebimento do resultado de sua ação. Além disso, verificamos o funcionamento dos escravos de acordo com a solicitação do mestre. Pudemos observar a verificação de erros de comunicação por meio do *checksum* presente no vetor de *bytes* de resposta.

A funcionalidade do sensoriamento da mesa respondeu de acordo com o proposto, possibilitando a eventual substituição e otimização do sistema, pois há uma documentação necessária sobre o *hardware* e o *software*. Já, o controle remoto do ar-condicionado foi implementado com a devida identificação do seu protocolo infravermelho tanto em nível de *hardware* como de *software*. Assim, o conjunto de conhecimento desenvolvido poderá servir de base para um controle pelas simulações/jogos virtuais do ambiente ITAE.

5.2 TRABALHOS FUTUROS

O sistema permite a inclusão de funcionalidades em seu protocolo, pois, das 16 funcionalidades, utilizamos apenas 4 funções. Ainda é possível desenvolver uma placa com o sensoriamento da mesa e o acionamento do ar-condicionado integrado, conforme pode ser visualizado no projeto no apêndice.

Uma outra melhoria é a forma de endereçamento dos escravos, que é realizado no programa. Seria possível utilizar um conjunto de chaves para programação do escravo da mesma forma que ocorre na programação de controle remoto para portão eletrônico.

Além disso, é possível implementar mais funções do protocolo infravermelho e utilizá-lo em outros equipamento que utilizam o padrão de protocolo JVC. Na mesa, há possibilidade de customização da matriz sensoriada.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] LOPEZ, R.A., (2000). “Sistemas de Redes para Controle e Automação – Rede Industrial, Tecnologias de Controle, Meios de Transmissão, Modelo OSI Rede Fieldbus Industriais, Sistemas Residenciais e rede Ethernet”, Editora Book Express, 276p.

[2] RANDAZZO, A. ST485: AN RS-485 BASED INTERFACE WITH LOWER DATA BIT ERRORS. 2004. Application Note 1348. Disponível em www.st.com/stonline/products/literature/an/7628.pdf

[3] PINHEIRO, Gil. Apresentação “A interface Serial e o Padrão RS-232”. 2011. Disponível em http://www.lee.eng.uerj.br/~gil/filas/Padrao%20RS-232.pdf

[4] CORDEIRO, T F K, (2009). Controlador de motor DC sob um barramento de comunicação RS485 - Algoritmo de controle e protocolo de comunicação. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-nº 10/2009, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 93p

[5] ATMEL. ATMEGA8. Disponível em http://www.atmel.com/images/doc8159.pdf> Acesso em 21 de maio de 2012

[6] SBPROJECTS. Infravermelho. http://www.sbprojects.com/knowledge/ir/index.php> Acesso em 21 de janeiro de 2011

[7] ITAE. Ambiente *Experience.* http://www.cdt.unb.br/programaseprojetos/itae/index> Acesso em outubro de 2011

ANEXOS

Anexo 1 – Código do arquivo .h do programa Escravo.

```
/*====================================================================
                                    Defines de configuração de compilação
====================================================================*/
//#define __TESTE_1MHZ
#define __TESTE_8MHZ
#define tamanhoBufferSerial 32      //Usar apenas potência de 2  (2 a 128)
/*****************************************************************************
                                Configurações baseadas nos defines anteriores
*****************************************************************************/
#ifdef __TESTE_1MHZ
        #define F_CPU 1000000UL  //  1 MHz
        #define CONSTANTE_TEMPO      2
                // #define          br_2400
                #define  br_4800
#endif
#ifdef __TESTE_8MHZ
        #define F_CPU 8000000UL  //  8 MHz
        #define CONSTANTE_TEMPO      2
                // #define br_2400
                //#define br_4800
                //#define br_9600
                #define br_19200
                // #define br_38400
#endif
#ifndef F_CPU
        #define F_CPU 16000000UL // 16 MHz
        #define CONSTANTE_TEMPO      4
        #define __16MHZ
                // #define br_2400
                #define br_4800
                // #define br_9600
                // #define br_19200
                // #define br_38400
#endif
/*****************************************************************************
                Criando nomes mais simples e intuitivos para as variáveis
*****************************************************************************/
#define uint8       unsigned char
#define int8        signed char
#define  uint16     unsigned int        // Compilador, o int -> 16 bits e long -> 32
#define  int16      int
#define uint32      unsigned long
#define int32       long
#define bool        char
#define true        1
#define false       0
/*****************************************************************************
                                Valores de configuração do baud rate
*****************************************************************************/
#define B2400               0
#define B4800               1
#define B9600               2
#define B19200              3
#define B38400              4
/*****************************************************************************
                                                Definindo bits de portas
*****************************************************************************/
// Pinos porta B
// Pinos porta C
```

```
// Pinos porta D
/*****************************************************************************
                                Definindo parametros do protocolo
*****************************************************************************/
#define Header                          0xAA
#define endIndividual           0x02    // Mudar caso necessario
#define endGrupo                        0xFF    // Mudar caso necessario
#define lider                           false   // Mudar para "true" se for lider

// Bits do Status Byte
#define checksumError                   0x01
#define servoOverrun                    0x02

// Bits do Byte de configuracao dos comandos defineStatus e readStatus
#define sendDeviceTypeVersion   0x01
#define sendTemp                                0x02
#define sendInformation3        0x04
#define sendInformation4        0x08
#define sendKeyboard                    0x10
#define sendInformation6        0x20
#define sendInformation7        0x40
#define sendInformation8        0x80

// Enumeracoes do protocolo (estados, comandos e etc.)
enum estadoComunicacao          {aguardandoCabecalho, aguardandoEndereco,
                                                        aguardandoComando, aguardandoBytesDados};
enum destinoComunicacao {destinoIndefinido, destinoIndividual, destinoGrupo,
                                                        destinoOutro};
enum tipoComando                {defineStatus, readStatus, noOP, comando3};

/*=============================================================================
                                                                Prototipos de funcoes
=============================================================================*/

// Varredeura do teclado
void varredura(void);
/*****************************************************************************
                                                        Buffers da comunicacao serial
*****************************************************************************/
uint8 leDadoSerial(void);
void escreveDadoSerial(uint8);
/*****************************************************************************
                        Configuracoes basicas de hardware e inicializacao de variaveis
*****************************************************************************/
void iniciaPortas(void);
void iniciaUSART(void);
void iniciaTempoServo(void);
void iniciaVariaveis(void);
void inicializa_timer1(void);
/*****************************************************************************
                                                                Protocolo
*****************************************************************************/
void montaVetorComando(uint8);
void interpretaComando(void);
bool verificaCheckSum(void);
void enviaDadosResposta(void);
void IR (unsigned char, unsigned char);
void power (int);
```

Anexo 2 – Código do arquivo .c do programa Escravo.

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include <avr/delay.h>
#include <avr/wdt.h>
#include "Escravo.h"

/*****************************************************************************
                                        Variaveis globais
*****************************************************************************/
// Comunicacao
volatile uint8          bufferDadosRX[tamanhoBufferSerial];
volatile uint8          bufferDadosTX[tamanhoBufferSerial];
volatile uint8          ptEscritaBufferRX, ptLeituraBufferRX;
volatile uint8          ptEscritaBufferTX, ptLeituraBufferTX;

// Temporizacao
volatile uint8          stepRateMultiplier;
volatile bool           iniciouNovoCiclo;

// Protocolo
uint8                   comando[16];                    // Vetor comando
volatile bool           recebeuNovoComando;
uint8                   enderecoRecebido, enderecoIndividual, enderecoGrupo;
bool                    liderDoGrupo;
uint8                   additionalStatus;       // Para defineStatus
uint8                   additionalReadStatus;   // Para readStatus
bool                    lerByteReadStatus;
uint8                   Status;                         // Status basico

// Teclado
uint8                   keyboard[24];                   //Vetor para armazenar
as 12 linhas do teclado
uint8                   coluna;
uint8                   linha;

// Temperatura ambiente
int                             sinal;
volatile int            i=0;                    //variável do controle para a função modulação
volatile int            ciclo=0;                //variável para contagem de ciclo do IR
uint8                   comIR;
```

```
uint8                                   temperature;
unsigned char                   controleIR=0;       // variável para o controle de entrada na função IR
/*****************************************************************************
                                                Macros
*****************************************************************************/
#define existeDadosBufferRX()       (ptLeituraBufferRX != ptEscritaBufferRX)
#define existeDadosBufferTX()       (ptLeituraBufferTX != ptEscritaBufferTX)
/*****************************************************************************
                                                Interrupcoes
*****************************************************************************/
ISR(USART_RXC_vect) // Recepcao serial
{
        bufferDadosRX[ptEscritaBufferRX++] = UDR;
        ptEscritaBufferRX &= (tamanhoBufferSerial - 1); // (0 ... 31 0 ... 31)
}
// -----------------------------------------------------------------------
ISR(USART_UDRE_vect) // Transmissao serial
{
    if(existeDadosBufferTX())
    {
        UDR = bufferDadosTX[ptLeituraBufferTX++];
        ptLeituraBufferTX &= (tamanhoBufferSerial - 1); // (0 ... 31 0 ... 31)
    }
    else // nao ha mais dados para enviar. Desligar a interrupcao.
    {
        if (UCSRA & _BV(TXC))
        {
            UCSRB &= ~(_BV(UDRIE)); // Desliga interrupcao de envio de dados
            UCSRA |= _BV(TXC);          // Limpa o flag de transmissao completa
            PORTD &= ~_BV(PD2);         // Desabilita RS485
        }
    }
}
// -----------------------------------------------------------------------
ISR(TIMER0_OVF_vect) // Overflow do timer 0 (8 bits)
{
        static uint8        i = 0, j = 0;
        i++;
        if(i == CONSTANTE_TEMPO)                        // Tick basico = 0.510ms.
        {
                i = 0;
                j++;
                if(j == stepRateMultiplier)             // Numero de ticks basicos = 5.1ms
                {
```

```
                        if(iniciouNovoCiclo) Status |= servoOverrun;
                        j = 0;
                        iniciouNovoCiclo = true;
                }
        }
}
// ----------------------------------------------------------------
// Esta funcao e executada antes do main, de forma que o watchdog
// seja desligado o mais rapido possível durante a inicializacao.
// uint8_t mcusr_mirror _attribute_ ((section (".noinit")));

void get_mcucsr(void) \
 __attribute__((naked)) \
 __attribute__((section(".init3")));
void get_mcucsr(void)
{
        //mcusr_mirror = MCUSR;
        MCUCSR = 0;
        wdt_disable();
}
/*************************************************************************
                                                                    main
*************************************************************************/
int main(void)
{
        bool checkSumOk;
        bool responderComando;
        iniciaVariaveis();
        iniciaPortas();
        iniciaTempoServo();
        iniciaUSART();
        inicializa_timer1();        //ajusta os valores do microcontrolador para o envio do sinal IR
        sei();
        while(1)
        {
                if(existeDadosBufferRX())
                {
                        montaVetorComando(leDadoSerial());
                }
                // ------------------- 510us * stepRateMultiplier = 5.1ms
                if(iniciouNovoCiclo)
                {
                        iniciouNovoCiclo = false;
                        // -------------------
```

```
                        if(recebeuNovoComando)
                        {
                                recebeuNovoComando = false;
                                checkSumOk = verificaCheckSum();
                                if(checkSumOk)
                                {
                                        interpretaComando();
                                        Status &= ~checksumError;
                                }
                                else
                                {
                                        Status |= checksumError;
                                }
                        responderComando = (enderecoRecebido == enderecoIndividual) ||
((enderecoRecebido == enderecoGrupo) && liderDoGrupo);
                                if(responderComando) enviaDadosResposta();
                        }
                        // --------------------
                        if (controleIR==1){
                                IR (comIR,temperature); // caso se escolha do comando seja
                                                        com = 4 (mudança de temp e vent), o valor
                                                        será 18ºC (3) de temp e ventilação em low
                                                        (0)
                                controleIR=0;
                        }
                        varredura(); // lê um linha do teclado
                }
                // --------------------
        }
        return 0;
}
/*****************************************************************************
                        Configurações basicas de hardware
*****************************************************************************/
void iniciaUSART(void)     // Porta de comunicacao serial assincrona
{
        UBRRH = 0;
#ifdef __TESTE_1MHZ
        #ifdef br_2400
                UBRL = 25;              // Erro: 0.2% (U2X=0)
        #endif
        #ifdef br_4800
                UBRRL = 12;             // Erro: 0.2% (U2X=0)
        #endif
#else
```

```
	#ifdef __TESTE_8MHZ
		#ifdef br_2400
			UBRL = 207;			// Erro = 0.2% (U2X=0)
		#endif
		#ifdef br_4800
			UBRRL = 103;	// Erro = 0.2% (U2X=0)
		#endif
		#ifdef br_9600
			UBRRL = 51;			// Erro = 0.2% (U2X=0)
		#endif
		#ifdef br_19200
			UBRRL = 25;			// Erro = 0.2% (U2X=0)
		#endif
		#ifdef br_38400
			UBRRL = 12;			// Erro = 0.2% (U2X=0)
		#endif
	#else	// 16MHz
		#ifdef br_2400
			// UBRR = 416	// Erro = -0.1%  (U2X=0)
			UBRRH = 0x01;
			UBRRL = 0xA0;
		#endif
		#ifdef br_4800
			UBRRL = 207;	// Erro = 0.2% (U2X=0)
		#endif
		#ifdef br_9600
			UBRRL = 103;	// Erro = 0.2% (U2X=0)
		#endif
		#ifdef br_19200
			UBRRL = 51;			// Erro = 0.2% (U2X=0)
		#endif
		#ifdef br_38400
			UBRRL = 25;			// Erro = 0.2% (U2X=0)
		#endif
	#endif
#endif
	UCSRB  = 0b10011000;	// RXCIE=1, UDRIE=0, RXEN=1, TXEN=1
	UCSRC  = 0b10000110;	// Sem paridade, 1 stop bit, 8 bits de dados
}
// ---------------------------------------------------------------------
void iniciaPortas(void)			// Configura portas B, C e D
{
	//DDRx  -	Data Direction: 0 entrada, 1 saída de dados.
	//PORTx-	Valor da saída. No caso de entrada de dados,
```

```
        //deixar valor 1 para ativar pull-up
        //Pinos nao usados: deixar o pull up ativado.
        DDRB   = 0b00001111;    //Controle Multiplexadores
        PORTB  = 0b00000000;
//      DDRC   |= 0b1110000;    // teste IR
//      PORTC  |= 0b0001111;    // teste IR
        DDRC   = 0b11111111;    //Controle Decodificadores
        PORTC  = 0b00000000;
        DDRD   = 0b00000110;    // TxD (bit 1), RxD (bit 0) e Tx enable(bit2)
        PORTD  = 0b11111001;
}
// ---------------------------------------------------------------------
void iniciaTempoServo(void)
{
        TCNT0 = 0;                      // Zerando contador (Timer 0)
#ifdef __TESTE_1MHZ
        TCCR0 = 0b00000001;     // Prescaler = 001.
                                        // Interrupcoes com periodo 255useg
#endif
#ifdef __TESTE_8MHZ
        TCCR0 = 0b00000010;     // Prescaler = 010.
                                        // Interrupcoes com periodo 255useg
#endif
#ifdef _16MHZ
        TCCR0 = 0b00000010;     // Prescaler = 010.
                                        // Interrupcoes com periodo 127.5useg
#endif
        TIMSK |= 0x01;          // Ativar interrupcao de overflow.
}
// ---------------------------------------------------------------------
void iniciaVariaveis(void)
{
        stepRateMultiplier = 10;        // Tick = 5.1ms
        enderecoIndividual = endIndividual;// Verificar como foi configurado
        enderecoGrupo = endGrupo;               // Verificar como foi configurado
        liderDoGrupo = lider;                   // Verificar como foi configurado
        additionalStatus = 0x00;
        additionalReadStatus = 0x00;
        Status = 0x00;                          // Status zerado, nenhum erro.
        linha = coluna = 0;
        int k;
        for(k=0;k<tamanhoBufferSerial;k++)bufferDadosRX[k] = 0x00;
        for(k=0;k<tamanhoBufferSerial;k++)bufferDadosTX[k] = 0x00;
        ptEscritaBufferRX = ptLeituraBufferRX = 0;
```

```
	ptEscritaBufferTX = ptLeituraBufferTX = 0;
}
// ---------------------------------------------------------------------
void inicializa_timer1(void) {
	DDRB   |= 0b00000010;		//PB 2 (pino 16) e OC1A (PB1 - pino 15)
	PORTB  |= 0b00000000;
	TCCR1A |= 0b11000010;		/* Definindo o modo de operação 10 (bit 1 –
					WGM11 e 0 - WGM10) e mode do Compare Output (bit 7:6
					- COM1A1:0)*/
	TCCR1B |= 0b00010000;		/* Definindo o modo de operação (bit 4 - WGM13
					e 3 - WGM12)*/
	TIMSK  |= 0b00000100;		// Ativar OCIE1A.
	ICR1 = 106;			//adequadando o valor da portadora em 38 kHz
	OCR1A = 79;			//adequadando o ciclo de trabalho em 1/3
}
ISR (TIMER1_OVF_vect){
	ciclo ++;			//contagem do número de ciclo
}
/*****************************************************************************
			Buffers da comunicacao serial (leitura e escrita)
*****************************************************************************/
uint8 leDadoSerial(void)    // Retira do buffer
{
	uint8 resposta;
	resposta = bufferDadosRX[ptLeituraBufferRX++];
	ptLeituraBufferRX &= (tamanhoBufferSerial - 1);		// (0 ... 31 0 ... 31)
	return resposta;
}
// ---------------------------------------------------------------------
void escreveDadoSerial(uint8 dado)		// Insere no buffer
{
	bufferDadosTX[ptEscritaBufferTX++] = dado;
	ptEscritaBufferTX &= (tamanhoBufferSerial - 1);		// (0 ... 31 0 ... 31)
	PORTD |= _BV(PD2);
	UCSRB |= _BV(UDRIE); //Liga a interrupcao para enviar dados.
}
/*****************************************************************************
						Protocolo
*****************************************************************************/
void montaVetorComando(uint8 dadoRecebido)
{
	static enum	estadoComunicacao	estado = aguardandoCabecalho;
	static enum	destinoComunicacao	destino = destinoIndefinido;
	static uint8	quantidadeBytesAReceber = 0;
	static uint8	quantidadeBytesRecebidos = 0;
```

```
static bool		recebendoBytesOutro = false;
if(recebendoBytesOutro)
{
		quantidadeBytesRecebidos++;
		if(quantidadeBytesRecebidos == quantidadeBytesAReceber)
		{
			recebendoBytesOutro = false;
			estado = aguardandoCabecalho;
		}
		return;
}
switch (estado)			// Maquina de estados
{
		case aguardandoCabecalho:
			if(dadoRecebido == Header)
			{
				estado = aguardandoEndereco;
			}
			break;
		case aguardandoEndereco:
			enderecoRecebido = dadoRecebido;
			if(enderecoRecebido == enderecoIndividual)
			{
				destino = destinoIndividual;
			}
			else if (enderecoRecebido == enderecoGrupo)
			{
				destino = destinoGrupo;
			}
			else
			{
				destino = destinoOutro;
			}
			estado = aguardandoComando;
			break;
		case aguardandoComando:
			quantidadeBytesRecebidos = 0;
			quantidadeBytesAReceber = (dadoRecebido >> 4) + 1; // + CheckSum
			if(destino == destinoIndividual || destino == destinoGrupo)
			{
				comando[0] = dadoRecebido;
			}
			else
			{
```

```
                        recebendoBytesOutro = true;
                    }
                    estado = aguardandoBytesDados;
                    break;
                case aguardandoBytesDados:
                    quantidadeBytesRecebidos++;
                    comando[quantidadeBytesRecebidos] = dadoRecebido;
                    if(quantidadeBytesRecebidos == quantidadeBytesAReceber)
                    {
                        estado = aguardandoCabecalho;
                        destino = destinoIndefinido;
                        recebeuNovoComando = true;
                    }
                    break;
            }
}
// ---------------------------------------------------------------------
bool verificaCheckSum(void)
{
        uint8   quantidadeBytes, i, soma;
        quantidadeBytes = (comando[0] >> 4);
        soma = enderecoRecebido;
        // O endereço recebido faz parte da soma do CheckSum
        for(i = 0; i <= quantidadeBytes; i++)
        // comando+quantidadeBytes (por isso o =, para considerar um valor a mais)
        {
            soma += comando[i];
        }
        return (soma == comando[quantidadeBytes + 1]);
        // o byte extra de comando é o CheckSum
}
// ---------------------------------------------------------------------
void interpretaComando(void)
{
        enum tipoComando        com = (enum tipoComando)(comando[0] & 0x0f);
        switch (com)
        {
            // -----------------------------------------------------------------
            case defineStatus:
                additionalStatus = comando[1];
                break;
            // -----------------------------------------------------------------
            case readStatus:
                additionalReadStatus = comando[1];
```

```
                        lerByteReadStatus = true;
                        break;
                // ---------------------------------------------------------------------
                case noOP:
                        // Nao faz nada
                        break;
                // ---------------------------------------------------------------------
                case comando3:
                        comIR = comando[1];
                        temperature = comando[2];
                        controleIR=1;
                        break;
        }
}
// ---------------------------------------------------------------------
void enviaDadosResposta(void)
{
        uint8   vetorResposta[29]; // Status + Checksum + Tipo(ID) + Versão + Teclas
        uint8   i, j, chksum;
        uint8   st;
        chksum = vetorResposta[0] = Status;
        if(lerByteReadStatus)
        {
                st = additionalReadStatus;
                lerByteReadStatus = false;
        }
        else
        {
                st = additionalStatus;
        }
        i = 1;
        if(st & sendDeviceTypeVersion)
        {
                chksum += vetorResposta[i++] = 0x92; // Tipo 0
                chksum += vetorResposta[i++] = 0x6D; // Versao 0
        }
        if(st & sendKeyboard)
        {
                int k;
                for(k=0; k<24; k++) chksum += vetorResposta[i++] = keyboard[k];
        }
        if(st & sendInformation4)
        {
        }
```

```
		if(st & sendInformation6)
		{
		}
		if(st & sendInformation7)
		{
		}
		if(st & sendInformation8)
		{
		}
		for(j = 0; j < i; j++)
		{
			escreveDadoSerial(vetorResposta[j]);
		}
		escreveDadoSerial(chksum);
}
void varredura(void)
{
		uint8	linhaLow = 0xFF;
		uint8	linhaHight = 0xFF;
		while(coluna<12)
		{
			PORTB = coluna;		// seleciona próxima entrada do Mux
			_delay_loop_2(3);		//delay for 12 cycles of clock = 12us
			switch(coluna)
			{
				case 0:
					linhaLow &= ((PIND & 0b01000000)>>6)|0b11111110;
					break;

				case 1:
					linhaLow &= ((PIND & 0b01000000)>>5)|0b11111101;
					break;
				case 2:
					linhaLow &= ((PIND & 0b01000000)>>4)|0b11111011;
					break;
				case 3:
					linhaLow &= ((PIND & 0b01000000)>>3)|0b11110111;
					break;
				case 4:
					linhaLow &= ((PIND & 0b01000000)>>2)|0b11101111;
					break;
				case 5:
					linhaLow &= ((PIND & 0b01000000)>>1)|0b11011111;
					break;
```

```
                        case 6:
                                linhaLow &= (PIND & 0b01000000)          |0b10111111;
                                break;
                        case 7:
                                linhaLow &= ((PIND & 0b01000000)<<1)|0b01111111;
                                break;
                        case 8:
                                linhaHight &= ((PIND & 0b10000000)>>7)|0b11111110;
                                break;
                        case 9:
                                linhaHight &= ((PIND & 0b10000000)>>6)|0b11111101;
                                break;
                        case 10:
                                linhaHight &= ((PIND & 0b10000000)>>5)|0b11111011;
                                break;
                        case 11:
                                linhaHight &= ((PIND & 0b10000000)>>4)|0b11110111;
                                break;
                }
                coluna++;
        }
        keyboard[2*linha] = linhaHight;
        keyboard[2*linha + 1] = linhaLow; // salva linha no vetor
        linha++;
        PORTC = linha;
        coluna = 0;
        if(linha==12)
        {
                linha = 0;
                PORTC = 0b00000000;
        }
}
void IR (unsigned char comando, unsigned char byte2){       //recebe o comando e valor de temp e vent, se for o caso
        unsigned char b[4];                     // variável para receber dados de envio
        unsigned char j, k;                     // variável de controle de envio dos bit´s do IR
        switch (comando){
                case 1:             //comando on
                        b[0] = 0b10001000; //bit IR de 0 a 7
                        b[1] = 0b00000000; //bit IR de 8 a 15
                        b[2] = 0b01010101; //bit IR de 16 a 23 com a temperatura em 20ºC (nibble superior - 0101) e ventilação automática (nibble inferior - 0101)
                        b[3] = 0b10101000; //bit IR de 24 a 31
                        power (1);                      // liga o contador
                        i=0;
```

```
            mod (-2);                   // header
            i=0;
            for (j=0;j<4;j++)   //envio dos bit´s, controle do vetor
                  for (k=0; k<=7; k++)      //controle do bit do vetor
                        if (j!=3||k<5)      //controle para finalização no bit 29
                              if ((b[j]<<k)&0x80){ //verificação do valor do bit do
                                                      vetor
                              i=0;
                              mod(1);//modulação do bit no valor 1
                              }
                              else{
                              i=0;
                              mod(0);//modulação do bit no valor 0
                              }
                        else;
                              power (0);          // desliga o contador
break;
case 2:                     //off
            b[0] = 0b10001000; //bit IR de 0 a 7
            b[1] = 0b11000000; //bit IR de 8 a 15
            b[2] = 0b00000101; //bit IR de 16 a 23
            b[3] = 0b00011000; //bit IR de 24 a 31
            power (1);                  // liga o contador
            i=0;
            mod (-2);                   // header
            i=0;
            for (j=0;j<4;j++)   //mesma rotina presente no case 1
                  for (k=0; k<=7; k++)
                        if (j!=3||k<5)
                              if ((b[j]<<k)&0x80){
                              i=0;
                              mod(1);
                              }
                              else{
                              i=0;
                              mod(0);
                              }
                        else;
                              power (0);          // desliga o contador
break;
case 3:                             //muda temperatura e ventilação
            b[0] = 0b10001000;      //bit IR de 0 a 7
            b[1] = 0b00001000;      //bit IR de 8 a 15
            b[2] = byte2;           //bit IR de 16 a 23, com a aquisição do valor
```

```
                                        desejado pelo mestre
//calculo do checksum
                unsigned char a1, a2; //variável auxiliar para o cálculo do checksum
                a1 = b[1]<<4;           //valor dos bit´s IR de 12 a 15
                a2 = byte2<<4;    //valor dos bit´s IR de 20 a 23
                b[3] = ((b[1]+a1+byte2+a2)&0b11110000)+0b00001000; //checksum a
                                                                        ser inserido nos
                                                                        bit´s de 24 a 31
                power (1);                      // liga o contador
                i=0;
                mod (-2);                       // header
                i=0;
                for (j=0;j<4;j++)               //mesma rotina presente no case 1
                        for (k=0; k<=7; k++)
                                if (j!=3||k<5)
                                        if ((b[j]<<k)&0x80){
                                                i=0;
                                                mod(1);
                                        }
                                        else{
                                                i=0;
                                                mod(0);
                                        }
                                else;
                                        power (0);          // desliga o contador
        break;
        case 4:                         //JET COOL
                b[0] = 0b10001000; //bit IR de 0 a 7
                b[1] = 0b00010000; //bit IR de 8 a 15
                b[2] = 0b00001000; //bit IR de 16 a 23
                b[3] = 0b10011000; //bit IR de 24 a 31
                power (1);                      // liga o contador
                i=0;
                mod (-2);               // header
                i=0;
                for (j=0;j<4;j++)               //mesma rotina presente no case 1
                        for (k=0; k<=7; k++)
                                if (j!=3||k<5)
                                        if ((b[j]<<k)&0x80){
                                                i=0;
                                                mod(1);
                                        }
                                        else{
                                                i=0;
                                                mod(0);
```

```
                                        }
                                else;
                                        power (0);          // desliga o contador
                        break;
                }
}
void power (int posi){                  //função que liga e desliga o contador
        switch (posi){
                case 0:
                        TCCR1B &= 0b11111000;/* O prescaler (bit 2:0 - sem fonte de relógio)*/
                        TCCR1A &= 0b00111111;// desconecta OC1A e volta a forma
                                                normal de operação
                        PORTB  &= 0b11111101;  // desliga a porta
                break;
                case 1:
                        TCCR1A |= 0b11000010;  // conecta o OC1A
                        TCCR1B |= 0b00010001;  /* O prescaler (bit 2:0 - prescaler 1)*/
                break;
        }
}
void mod (int sinal){
        while (i<1){
                switch (sinal){
                        case -2:        //Header com luz, totalizando 318 sinais de
                                        ondas = 8,4ms
                                if (ciclo<=318);
                                else {
                                        ciclo = 0; //inicializando a contagem do ciclo para o
                                                envio do próximo sinal
                                        sinal=-1; //Definindo o valor para envio do Header sem
                                                luz
                                }
                                break;
                        case -1:        //Header sem luz, equivalente ao envio de 162
                                        sinais de onda = 4,2ms
                                TCCR1A &= 0b00111111;// Desativando a saída da porta de
                                                        comparação do timer 1
                                if (ciclo<=162);
                                else {
                                        TCCR1A |= 0b11000000;  // Ativando a saída da
                                                                porta de comparação do timer 1
                                        ciclo = 0;      //inicializando a contagem do ciclo para o
                                                        envio do próximo sinal
                                        i=1;            //Saindo da função mod
```

```
                }
                break;
            case 0:                 //bit 0, com execução em 1,079 ms para
                                    seu total envio
                if (ciclo<=41){
                    if (ciclo<=19);     //Parte do bit com luz, com execução de
                                        500us
                    else {
                        TCCR1A &= 0b00111111; // Desativando a
                                                saída da porta de
                                                comparação do timer 1
                    }
                }
                else {
                    TCCR1A |= 0b11000000;       // Ativando a saída da porta de
                                                comparação do timer 1
                    ciclo = 0;          //inicializando a contagem do ciclo para o envio do
                                        próximo sinal
                    i=1;
                }
                break;
            case 1: //bit 1, com execução em 2,132 ms para seu total envio
                if (ciclo<=81){
                    if (ciclo<=19);             //Parte do bit com luz, com execução de
                                                500us
                    else {
                        TCCR1A &= 0b00111111;       // Desativando a saída
                                                    da porta de comparação
                                                    do timer 1
                    }
                }
                else {
                    TCCR1A |= 0b11000000;       // Ativando a saída da porta de
                                                comparação do timer 1
                    ciclo = 0;          //inicializando a contagem do ciclo para o envio do
                                        próximo sinal
                    i = 1;
                }
                break;
        }
    }
}
```

Anexo 3 – Código do arquivo .h do programa Mestre.

```
#include <windows.h>
#include <stdio.h>
// ====================================================================
// Microcontroller clock (select one of then)
// #define    Clock_1MHz
 #define    Clock_8MHz
// #denine    Clock_16MHz

// DEFINITONS:
#define Header          0xAA    //Header byte (*)
#define MAX_UNITS       32      // Maximum number of modules on the bus

// Commands:
#define SET_STATUS        0x00   //Define what is returned as default status
#define READ_STATUS       0x01   //Read specified status items one time only
#define NO_OP             0x02           //Simply reports default status
//#define SET_ADDRESS     0x00   //Set individual and group addresses
//#define SET_STATUS      0x01   //Define what is returned as default status
//#define READ_STATUS     0x02   //Read specified status items one time only
//#define SET_BAUD        0x03   //Set baud rate (group command only)
//#define NO_OP           0x04   //Simply reports default status
//#define HARD_RESET      0x05   //Complete reset of controller
//#define IDENTIFICATION  0x06     //Read type of
#define AR_CONDICIONADO   0x03           //Ar condicionado
// Comandos do ar condicionado
#define LIGA                            0x01     //Liga ar condicionado
#define DESLIGA                          0x02     //Desliga ar condicionado
#define MUDA                            0x03     //Muda temperatura e potencia da ventilação
#define JET_COOL                        0x04     //Habilita Jet Cool

// Status byte bit definitions:
#define COM_ERROR       0x01    //Set if UART or cksum error, clear otherwise
#define Servo_Overrun1  0x02    //Set if used time > 0.512mSeg (*)
#define Servo_Overrum2  0x04      //Set if used time > 5.12mSeg. Slave stop on keyboard reading

// Define baud rate divisors:
#define PB1200                  0x00
#define PB2400                  0x01
#define PB4800          0x02
#define PB9600          0x03
#define  PB19200        0x04
```

```
#define PB38400         0x05
#define  PB57600         0x06
#define  PB115200        0x07
#define  PB230400        0x08

// Status items byte bit definitions (Adittional data sent):
#define SEND_ID_VER     0x01    //Send identificatin, version (2 bytes)
#define SEND_TEMP       0x02    //Send temperature(1 byte)
#define SEND_TYPE                   0x04    //Send ?
#define SEND_INFO3      0x08    //Send ?
#define SEND_KEYBOARD           0x10      //Send information keyboard(24 bytes)
// ====================================================================
typedef struct MODULE
{
      BOOL LEADER;     //flag to a group leader
      byte GADDR;      //group address
                  byte SDEF;       //defines default status items returned
                  byte STAT;       //status byte
                  byte ID;         //device type
                  byte VER;        //version number
                  byte temperature;
                  byte keyboard[24];
                  byte TYPE;
                  byte   bd;       // baudrate
};

// Prototypes:
void    ErrorPrinting(BOOL);
int     ErrorMsgBox(char *);
int     SimpleMsgBox(char *);
HANDLE  SioOpen(char *, unsigned int);
BOOL    SioPutChars(HANDLE, char *, int);
DWORD   SioGetChars(HANDLE, char *, int);
DWORD   SioTest(HANDLE);
BOOL    SioClrInbuf(HANDLE);
BOOL    SioChangeBaud(HANDLE, unsigned int);
BOOL    SioClose(HANDLE);
void    ErrorShow(void); // (*)
BOOL    GET_RESPONSE(MODULE, byte);       // For Modules
BOOL    M_SET_STATUS(MODULE, byte, byte);
BOOL    M_READ_STATUS(MODULE, byte, byte);
BOOL    M_NO_OP(MODULE, byte);
void    InitVars(MODULE, int);
void interpretaTecla(byte, int);
```

```
BOOL ARCONDICIONADO(MODULE, byte, byte, byte);
// ====================================================================
// Global variables:

#define     no_modules  1       // Number of modules to be considered
static BOOL        printerrors = true; // Print errors (true) or not (false)
static byte        comm_string[29] = {0};    // Command string
static BOOL        wait_stat = true;   // Flag: Wait status (true) or not (false)
static BOOL                    wait_keyboard = false; // Flag: Wait keyboard
static int         nSrv = no_modules;  // Number of modules
static byte        Group_Leader = 0;   // Group leader address
static HANDLE      Com_Port;           // Handle to Serial COM port
//static byte                  keyboard[24] = {0};           // Keyboard
//static int                   keyboardMap[11][11] = {0};
```

Anexo 4 – Código do arquivo .c do programa Mestre.

```
#include <cstdlib>
#include <iostream>
#include <stdio.h>
#include "master.h"

using namespace std;
// ########################################################################## //
// ########################################################################## //
// ##     FUNCOES DE COMUNICACAO SERIAL                                    ## //
// ########################################################################## //
// ########################################################################## //
int SimpleMsgBox(char *msgstr)
{
   return MessageBox(NULL, msgstr,"",MB_TASKMODAL | MB_SETFOREGROUND);
}
void ErrorPrinting(BOOL f)
{
    printerrors = f;
}
int ErrorMsgBox(char *msgstr)
{
   if (printerrors)
      return MessageBox(NULL, msgstr,"",MB_TASKMODAL | MB_SETFOREGROUND);
   else return(0);
}
//---------------------------------------------------------------------------//
// ABRIR A PORTA SERIAL
//---------------------------------------------------------------------------//
HANDLE SioOpen(char *name, unsigned int baudrate)
{
      BOOL RetStat;
      COMMCONFIG cc;
      COMMTIMEOUTS ct;
      HANDLE ComHandle;
      DWORD winrate;
      char msgstr[50];
      //Open COM port as a file
      ComHandle = CreateFile(name, GENERIC_READ | GENERIC_WRITE ,0, NULL,
      OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL , NULL);
      while (TRUE)
      {
```

```
if (ComHandle == INVALID_HANDLE_VALUE)
                {
  sprintf(msgstr,"%s failed to open",name);
                  ErrorMsgBox(msgstr);
  break;
}
switch (baudrate)
{
                                        case 1200:              winrate = CBR_1200; break;
                                        case 2400:              winrate = CBR_2400; break;
                                        case 4800:              winrate = CBR_4800; break;
                    case 9600:                  winrate = CBR_9600; break;
                    case 19200:     winrate = CBR_19200; break;
                    case 38400:     winrate = CBR_38400; break;
                    case 57600:     winrate = CBR_57600; break;
                    case 115200:    winrate = CBR_115200; break;
                    case 230400:    winrate = 230400; break;
          default: ErrorMsgBox("Baud rate not supported - using default of 48000");
                                        winrate = CBR_4800;
}
//Fill in COM port config. structure & set config.
cc.dwSize = sizeof(DCB) + sizeof(WCHAR) + 20;
cc.wVersion = 1;
cc.dcb.DCBlength = sizeof(DCB);
cc.dcb.BaudRate = winrate;
cc.dcb.fBinary = 1;
cc.dcb.fParity = 0;
cc.dcb.fOutxCtsFlow = 0;
cc.dcb.fOutxDsrFlow = 0;
cc.dcb.fDtrControl = DTR_CONTROL_DISABLE;
cc.dcb.fDsrSensitivity = 0;
cc.dcb.fTXContinueOnXoff = 0;
cc.dcb.fOutX = 0;
cc.dcb.fInX = 0;
cc.dcb.fErrorChar = 0;
cc.dcb.fNull = 0;
cc.dcb.fRtsControl = RTS_CONTROL_DISABLE;
cc.dcb.fAbortOnError = 0;
cc.dcb.XonLim = 100;
cc.dcb.XoffLim = 100;
cc.dcb.ByteSize = 8;
cc.dcb.Parity = NOPARITY;
cc.dcb.StopBits = ONESTOPBIT;
cc.dcb.XonChar = 'x';
```

```
            cc.dcb.XoffChar = 'y';
            cc.dcb.ErrorChar = 0;
            cc.dcb.EofChar = 0;
            cc.dcb.EvtChar = 0;
            cc.dwProviderSubType = PST_RS232;
            cc.dwProviderOffset = 0;
            cc.dwProviderSize = 0;
            RetStat = SetCommConfig(ComHandle, &cc, sizeof(cc));
            if (RetStat == 0)
            {
              ErrorMsgBox("Failed to set COMM configuration");
              break;
            }
            //Set read/write timeout values for the file
            ct.ReadIntervalTimeout = 0;                    //ignore interval timing
            ct.ReadTotalTimeoutMultiplier = 2; //2 msec per char
            ct.ReadTotalTimeoutConstant = 50;  //plus add'l 50 msec
            ct.WriteTotalTimeoutMultiplier = 2;//Set max time per char written
            ct.WriteTotalTimeoutConstant = 50; //plus additional time
            RetStat = SetCommTimeouts(ComHandle, &ct);
            if (RetStat == 0)
            {
              ErrorMsgBox("Failed to set Comm timeouts");
              break;
            }
            break;
        }
        return(ComHandle);
}

//---------------------------------------------------------------------------//
 //MUDAR BAUDRATE
//---------------------------------------------------------------------------//
BOOL SioChangeBaud(HANDLE ComPort, unsigned int baudrate)
{
    BOOL RetStat;
    DWORD winrate;
    DCB cs;
    RetStat = GetCommState(ComPort, &cs);
    if (RetStat == false) return RetStat;
    switch (baudrate)
    {
            case 1200:                  winrate = CBR_1200; break;
            case 2400:                  winrate = CBR_2400; break;
```

```
            case 4800:              winrate = CBR_4800; break;
            case 9600:              winrate = CBR_9600; break;
            case 19200:     winrate = CBR_19200; break;
            case 38400:     winrate = CBR_38400; break;
            case 57600:     winrate = CBR_57600; break;
            case 115200:    winrate = CBR_115200; break;
            case 230400:    winrate = 230400; break;
            default:        ErrorMsgBox("Baud rate not supported");
            return false;
    }
    cs.BaudRate = winrate;
    RetStat = SetCommState(ComPort, &cs);
    if (RetStat == false) return RetStat;
    return true;
}
//---------------------------------------------------------------------------//
// ENVIAR n CARACTERS PARA A PORTA
//---------------------------------------------------------------------------//
BOOL SioPutChars(HANDLE ComPort, char *stuff, int n)
{
    BOOL RetStat;
    DWORD nums;
    RetStat = WriteFile(ComPort, stuff,n, &nums, NULL);
    if (RetStat == 0) ErrorMsgBox("SioPutChars failed");
    return RetStat;
}
//---------------------------------------------------------------------------//
// LE n CARACTERES E RETORNA A QUANTIDADE LIDA ATÉ ENTÃO(numread)
//---------------------------------------------------------------------------//
DWORD SioGetChars(HANDLE ComPort, char *stuff, int n)
{
     BOOL RetStat;
     DWORD numread;
     RetStat = ReadFile(ComPort, stuff, n, &numread, NULL);
     if (RetStat == 0) ErrorMsgBox("SioReadChars failed");
     return numread;
}
//---------------------------------------------------------------------------//
// RETORNA O NUMERO DE CARACTERES NO BUFFRER DE ENTRADA
//---------------------------------------------------------------------------//
DWORD SioTest(HANDLE ComPort)
{
    COMSTAT cs;
    DWORD Errors;
```

```
    BOOL RetStat;
    RetStat = ClearCommError(ComPort, &Errors, &cs);
    if (RetStat == 0) ErrorMsgBox("SioTest failed");
    return cs.cbInQue;
}
//---------------------------------------------------------------------------//
//  Function Name:  SioClrInBuf (Internal Library Function)                  //
//  Return Value:   0=Fail, 1=Success                                        //
//  Parameters:     ComPort: COM port handle                                 //
//  Description:    Purge all chars from a port's input buffer.              //
//---------------------------------------------------------------------------//
BOOL SioClrInbuf(HANDLE ComPort)
{
    BOOL RetStat;
    RetStat = PurgeComm(ComPort, PURGE_RXCLEAR);
    if (RetStat == 0) ErrorMsgBox("SioClrInbuf failed");
    return RetStat;
}
//---------------------------------------------------------------------------//
//  Description:    Close a previously opened COM port.                      //
//---------------------------------------------------------------------------//
BOOL SioClose(HANDLE ComPort)
{
    return(CloseHandle(ComPort));
}
//---------------------------------------------------------------------------//
//  Description:    Show the error on communication link                     //
//---------------------------------------------------------------------------//
void ErrorShow(void)
{
    char lastError[1024];

    FormatMessage(
                FORMAT_MESSAGE_FROM_SYSTEM | FORMAT_MESSAGE_IGNORE_INSERTS,
                NULL,
                GetLastError(),
                MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT),
                lastError,
                1024,
                NULL);
    printf(" \n\n Erro de comunicacao = %s \n\n",lastError);

}
// ########################################################################## //
```

```
// ########################################################################## //
// ##     FUNCOES DE COMANDO BASICAS                                       ## //
// ########################################################################## //
// ########################################################################## //

// ========================================================================== //
// GET_RESPONSE:
// GET STATUS BYTE AND ADDITIONAL BYTES (IF CONFIGURED)
// ADDR: Address of the module that responds
//
// Returns true or false
// ========================================================================== //
BOOL GET_RESPONSE(MODULE modulo[], byte ADDR)
{
   int     k;
   DWORD   nbytes;
   byte    cksum;
           k = 2;
           if (modulo[ADDR].SDEF & SEND_ID_VER) k = k+2; // lê 4 bytes: STAT, ID, VER e Checksum
           if (modulo[ADDR].SDEF & SEND_TYPE) k = k+1;       // lê 1 byte: tipo do dispositivo
           if (modulo[ADDR].SDEF & SEND_TEMP) k = k+1;       // lê 1 byte: Temperature
           if (modulo[ADDR].SDEF & SEND_KEYBOARD) k = k+24;  // lê 24 bytes: teclas

           //temporario = modulo[ADDR].SDEF;
           for(int i = 0; i<k; i++) comm_string[i] = 0;
   nbytes = SioGetChars(Com_Port, (char *)comm_string, k);
   for(int i = 0; i<k; i++) printf("%d\n",comm_string[i]);
           if (nbytes != k)                          // No. of bytes test
   {
      SimpleMsgBox("Error on SioGetChars!");
      return false;
   }

           //modulo[ADDR].SDEF = temporario;
   cksum = 0;
   for (k=0; k<=nbytes-2; k++)
      cksum = cksum + comm_string[k];
           if (cksum != comm_string[nbytes-1])              // Checksum test
   {
      SimpleMsgBox("Checksum Error on Module Response!");
      return false;
   }
   k = 0;
   modulo[ADDR].STAT = comm_string[k];
```

```
    k = k + 1;
        if (modulo[ADDR].SDEF & SEND_ID_VER)
        {
            modulo[ADDR].ID = comm_string[k];
            modulo[ADDR].VER = comm_string[k+1];
            k = k + 2;
        }
        if (modulo[ADDR].SDEF & SEND_TYPE)
        {
            modulo[ADDR].TYPE = comm_string[k];
            k++;
        }
        if (modulo[ADDR].SDEF & SEND_TEMP)
        {
            modulo[ADDR].temperature = comm_string[k];
            k++;
        }
        if (modulo[ADDR].SDEF & SEND_KEYBOARD)
        {
            for(int j=0;j<24;j++)
                modulo[ADDR].keyboard[j] = comm_string[k+j];
        }
    if (modulo[ADDR].STAT & COM_ERROR)
    {
       SimpleMsgBox("Checksum Error on Module Command!");
       return false;
    }
        return true;
}
// ================================================================== //
// M_SET_STATUS:
// DEFINES ADDITIONAL STATUS DATA
// ADDR: Address
// StatusItens: Data itens to be returned
// Returns true or false
// ================================================================== //
BOOL M_SET_STATUS(MODULE modulo[], byte ADDR, byte StatusItens)
{
    byte cksum;
    BOOL ckcomm;
    byte address;
    unsigned int nbytes = 2;
    comm_string[0] = Header;
    comm_string[1] = ADDR;
```

```
cksum = comm_string[1];
comm_string[2] = (0x10 | SET_STATUS);
cksum = cksum + comm_string[2];
comm_string[3] = StatusItens;
cksum = cksum + comm_string[3];
comm_string[4] = cksum;
SioClrInbuf(Com_Port);  //Get rid of any old input chars
ckcomm = SioPutChars(Com_Port, (char *)comm_string, 5);
//Sleep(100);
if (ckcomm == false)
{
   SimpleMsgBox("Error on SioPutChars!");
   return false;
}
if (!wait_stat) return true; // No status byte
if (ADDR>=0x80) // Caso seja endereço de grupo, atualizar o status itens de todo o grupo
{              // e modificar ADDR para receber o status do lider
   address = ADDR;
   ADDR = Group_Leader;
   int i;
   for(i=1;i<=32;i++) if(modulo[i].GADDR = address) modulo[i].SDEF = StatusItens;
        }
        modulo[ADDR].SDEF = StatusItens;
        if (modulo[ADDR].SDEF & SEND_ID_VER)
        {
                printf("Status Itens: bit 1 ok, ler numero de identificacao e versao\n");
                nbytes = nbytes + 2;
        }
        if (modulo[ADDR].SDEF & SEND_TYPE)
        {
                printf("Status Itens: bit 3 ok, ler o tipo do dispositivo\n");
                nbytes = nbytes + 1;
        }
        if (modulo[ADDR].SDEF & SEND_TEMP)
        {
                printf("Status Itens: bit 2 ok, ler a temperatura ambiente\n");
                nbytes = nbytes + 1;
        }
        if (modulo[ADDR].SDEF & SEND_KEYBOARD)
        {
                printf("Status Itens: bit 5 ok, ler as teclas da mesa\n");
                nbytes = nbytes + 24;
        }
        // Delay de 2*(10/bd)msec por byte + 5msec para execução do protocolo pelo escravo
```

```
    Sleep((unsigned int)(1.1*(2*nbytes*10)/modulo[ADDR].bd) + 5);
            ckcomm = GET_RESPONSE(modulo, ADDR);
    if (ckcomm == false)
    {
        SimpleMsgBox("Error on GET_RESPONSE!");
        return false;
    }
            return true;
}

// ================================================================== //
// M_READ_STATUS:
// READS CONFIGURED ADDITIONAL STATUS DATA
// ADDR: Address
// StatusItens: Data itens to be returned
// Returns true or false
// ================================================================== //
BOOL M_READ_STATUS(MODULE modulo[], byte ADDR, byte StatusItens)
{
    byte cksum;
    BOOL ckcomm;
    byte temp;
    unsigned int nbytes = 2;
    comm_string[0] = Header;
    comm_string[1] = ADDR;
    cksum = comm_string[1];
    comm_string[2] = (0x10 | READ_STATUS);
    cksum = cksum + comm_string[2];
    comm_string[3] = StatusItens;
    cksum = cksum + comm_string[3];
    comm_string[4] = cksum;
    SioClrInbuf(Com_Port);  //Get rid of any old input chars
    ckcomm = SioPutChars(Com_Port, (char *) comm_string, 5);
    //Sleep(100);
    if (ckcomm == false)
    {
        SimpleMsgBox("Error on SioPutChars!");
        return false;
    }
    if (ADDR>=0x80) ADDR = Group_Leader;        // Group Leader that returns status
    temp = modulo[ADDR].SDEF;
    modulo[ADDR].SDEF = StatusItens;
            if (modulo[ADDR].SDEF & SEND_ID_VER)
            {
```

```
                printf("Status Itens: bit 1 ok, ler numero de identificacao e versao\n");
                nbytes = nbytes + 2;
        }
        if (modulo[ADDR].SDEF & SEND_TYPE)
        {
                printf("Status Itens: bit 3 ok, ler o tipo do dispositivo\n");
                nbytes = nbytes + 1;
        }
        if (modulo[ADDR].SDEF & SEND_TEMP)
        {
                printf("Status Itens: bit 2 ok, ler a temperatura ambiente\n");
                nbytes = nbytes + 1;
        }
        if (modulo[ADDR].SDEF & SEND_KEYBOARD)
        {
                printf("Status Itens: bit 5 ok, ler as teclas da mesa\n");
                nbytes = nbytes + 24;
        }
        // Delay de 2*(10/bd)msec por byte + 5msec para execução do protocolo pelo escravo
   Sleep((unsigned int)(1.1*(2*nbytes*10)/modulo[ADDR].bd) + 5);
   ckcomm = GET_RESPONSE(modulo, ADDR);
   modulo[ADDR].SDEF = temp;
   if (ckcomm == false)
   {
      SimpleMsgBox("Error on GET_RESPONSE!");
      return false;
   }
        return true;
}
// ========================================================================= //
// M_NO_OP:
// NO OPERATION BUT DEFINED STATUS DATA ARE RETURNED
// ADDR: Address (individual or group)
// Returns true or false
// ========================================================================= //
BOOL M_NO_OP(MODULE modulo[], byte ADDR)
{
   byte cksum;
   BOOL ckcomm;
   unsigned int nbytes = 2;
   comm_string[0] = Header;
   comm_string[1] = ADDR;
   cksum = comm_string[1];
   comm_string[2] = (0x00 | NO_OP);
```

```
    cksum = cksum + comm_string[2];
    comm_string[3] = cksum;
    SioClrInbuf(Com_Port);  //Get rid of any old input chars
    ckcomm = SioPutChars(Com_Port, (char *) comm_string, 4);
    if (ckcomm == false)
    {
       SimpleMsgBox("Error on SioPutChars!");
       return false;
    }
    if (!wait_stat) return true; // No status byte
    if (ADDR>=0x80) ADDR = Group_Leader;
            if (modulo[ADDR].SDEF & SEND_ID_VER)
            {
                    printf("Status Itens: bit 1 ok, ler numero de identificacao e versao\n");
                    nbytes = nbytes + 2;
            }
            if (modulo[ADDR].SDEF & SEND_TYPE)
            {
                    printf("Status Itens: bit 3 ok, ler o tipo do dispositivo\n");
                    nbytes = nbytes + 1;
            }
            if (modulo[ADDR].SDEF & SEND_TEMP)
            {
                    printf("Status Itens: bit 2 ok, ler a temperatura ambiente\n");
                    nbytes = nbytes + 1;
            }
            if (modulo[ADDR].SDEF & SEND_KEYBOARD)
            {
                    printf("Status Itens: bit 5 ok, ler as teclas da mesa\n");
                    nbytes = nbytes + 24;
            }
            // Delay de 2*(10/bd)msec por byte + 5msec para execução do protocolo pelo escravo
    Sleep((unsigned int)(1.1*(2*nbytes*10)/modulo[ADDR].bd) + 5);
    //Sleep(100);
    ckcomm = GET_RESPONSE(modulo, ADDR);
    if (ckcomm == false)
    {
       SimpleMsgBox("Error on GET_RESPONSE!");
       return false;
    }
}
//---------------------------------------------------------------------------//
//  InitVars:
//  Initialize misc network variables
```

```
//  N: Number of modules
//  Returns nothing
//-------------------------------------------------------------------------------//
void InitVars(MODULE modulo[], int N)
{
      int i;
      modulo[0].ID = 0;       //Default to a known module type for module 0
      modulo[0].VER = 0;
      modulo[0].STAT = 0;
      modulo[0].SDEF = 0;
      modulo[0].GADDR = 0xFF;
      modulo[0].LEADER = false;
            modulo[0].temperature = 0;
            modulo[0].TYPE = 0xFF;
            modulo[0].bd = 4800;
            for(int j=0; j<24; j++) modulo[0].keyboard[j] = 0;
      for (i=1; i<=N; i++)
      {
         modulo[i].ID = 0xFF; // N modules
         modulo[i].VER = 0;
         modulo[i].STAT = 0;
         modulo[i].SDEF = 0;
         modulo[i].GADDR = 0xFF;
         modulo[i].LEADER = false;
                     modulo[i].temperature = 0;
                     modulo[i].TYPE = 0xFF;
                     modulo[i].bd = 4800;
                     for(int j=0; j<24; j++) modulo[i].keyboard[j] = 0;
      }
}
BOOL ARCONDICIONADO(MODULE modulo[], byte ADDR, byte comando, byte tempPot)
{
    byte cksum;
    BOOL ckcomm;
    unsigned int nbytes = 2;
    comm_string[0] = Header;
    comm_string[1] = ADDR;
    cksum = comm_string[1];
    comm_string[2] = (0x20 | AR_CONDICIONADO);
    cksum = cksum + comm_string[2];
    comm_string[3] = comando;
    cksum = cksum + comm_string[3];
    comm_string[4] = tempPot;
    cksum = cksum + comm_string[4];
```

```
    comm_string[5] = cksum;
    SioClrInbuf(Com_Port);  //Get rid of any old input chars
    ckcomm = SioPutChars(Com_Port, (char *)comm_string, 6);
    Sleep(100);
    if (ckcomm == false)
    {
       SimpleMsgBox("Error on SioPutChars!");
       return false;
    }
    if (!wait_stat) return true; // No status byte
            if (ADDR>=0x80) ADDR = Group_Leader;
            if (modulo[ADDR].SDEF & SEND_ID_VER)
            {
                    printf("Status Itens: bit 1 ok, ler numero de identificacao e versao\n");
                    nbytes = nbytes + 2;
            }
            if (modulo[ADDR].SDEF & SEND_TYPE)
            {
                    printf("Status Itens: bit 3 ok, ler o tipo do dispositivo\n");
                    nbytes = nbytes + 1;
            }
            if (modulo[ADDR].SDEF & SEND_TEMP)
            {
                    printf("Status Itens: bit 2 ok, ler a temperatura ambiente\n");
                    nbytes = nbytes + 1;
            }
            if (modulo[ADDR].SDEF & SEND_KEYBOARD)
            {
                    printf("Status Itens: bit 5 ok, ler as teclas da mesa\n");
                    nbytes = nbytes + 24;
            }
// Delay de 2*(10/bd)msec por byte + 5msec para execução do protocolo pelo escravo
    Sleep((unsigned int)(1.1*(2*nbytes*10)/modulo[ADDR].bd) + 5);
    ckcomm = GET_RESPONSE(modulo, ADDR);
    if (ckcomm == false)
    {
       SimpleMsgBox("Error on GET_RESPONSE!");
       return false;
    }
            return true;
}
// ############################################################################ //
// ############################################################################ //
// ##     INTERFACE COM O TECLADO                                              ## //
```

```
// ############################################################################ //
// ############################################################################ //
void interpretaTecla(byte keyboard[], int mapa[12][12])
{
	int buffer[12] = {0};
	int aux1, aux2;
	for(int i=0;i<23;i+=2)
	{
		aux2 = keyboard[i];
		aux1 = keyboard[i+1];
		buffer[i/2] = aux2*256 + aux1;

	}
	for(int i=0;i<12;i++)
	{
		for(int j=0; j<12;j++)
		{
			mapa[i][j] = (buffer[i]>>(j))&1;
			printf("%d  ",mapa[i][j]);
		}
		printf("\n");
	}
}
// ############################################################################ //
// ############################################################################ //
// ##     PROGRAMA PRINCIPAL                                          ## //
// ############################################################################ //
// ############################################################################ //
int main()
{
	MODULE modulo[32];

	DWORD    nbytes;                // numero de bytes lidos pelo SioGetChars
	BOOL     resultado;             // recebe o valor retornado pelas funcoes booleanas
   char    baudrate[10];           // baudrate a ser mudado
	byte     address;               // endereço do módulo
	byte     informacoes;           // informações para o Status Itens
	char     endereco[10];          // entrada do usuario contendo o endereço
	char     info[50];          // entrada do usuario contendo as informações
	char     subInfo[10];
	char     *extraiInfo;
	char     funcionalidade[10];    // flag para saber a funionalidade escolhida pelo usuario
	char     continuar;             // flag
	char     comandoArCondicionado[10];    // Entrada do usuário para o código do a ser enviado para o
ar condicionado
```

```
        char    temperatura[10]; // Entrada do usuário para a temperatura
        char    potencia[10];           // Entrada do usuário para a potencia de ventilação
        byte    temperaturaPotencia;    // Byte a ser mondado com o código referente a
                                            temperatura e a potência da ventilação escolhidas pelo
                                            usuário.
        int     mapa[12][12];           // matriz para imprimir as teclas
        wait_stat = true;
   //SYSTEM_INIT(modulo, 2);     // inicializa a rede
   Com_Port = SioOpen("COM1", 4800);
   informacoes = SEND_ID_VER;
  /*  address = 0x01;
   resultado = M_SET_STATUS(modulo, address, informacoes);
   if(resultado == false) return false;
   address = 0x02;
   resultado = M_SET_STATUS(modulo, address, informacoes);
   if(resultado == false) return false;
*/
        do
        {
                printf("Digite ESCRAVO1 para enviar para enviar comando para o endereco 0x01 e
ESCRAVO2 para o endereco 0x02, e Grupo para enviar o comando para o grupo. \n");
                gets(endereco);
                extraiInfo = strstr(endereco, "ESCRAVO1");
                if(extraiInfo) address = 0x01;
                extraiInfo = strstr(endereco, "ESCRAVO2");
                if(extraiInfo) address = 0x02;
                extraiInfo = strstr(endereco, "Grupo");
                if(extraiInfo) address = 0x81;
                do
                {
                        printf("Digite DEFINE para definir que informacao a ser retornada no Status, ou
ARCONDICIONADO para acionar o ar condicionado\n\n");
                        gets(funcionalidade); //Usuario escolhe funcionalidade
                        extraiInfo = strstr(funcionalidade, "DEFINE");
                        if(extraiInfo)
                        {
                                do
                                {
                                        informacoes = 0x00;
                                        printf("\n");
                                        printf("Digite as informacoes que devem ser retornadas pelo
status\n");
                                        printf("Digite as strings seguidas e separadas por uma barra |\n");
                                        printf("Um exemplo de informacoes:
TECLADO|TEMPERATURA|TIPO|ID_VER\n\n");
                                        gets(info);
```

```
                                        extraiInfo = strstr(info, "ID_VER");
                                        if(extraiInfo) informacoes |= SEND_ID_VER;
                                        extraiInfo = strstr(info, "TIPO");
                                        if(extraiInfo) informacoes |= SEND_TYPE;
                                        extraiInfo = strstr(info, "TEMPERATURA");
                                        if(extraiInfo) informacoes |= SEND_TEMP;
                                        extraiInfo = strstr(info, "TECLADO");
                                        if(extraiInfo) informacoes |= SEND_KEYBOARD;
                                        printf("Enviando comando Define Status\n\n");
                                        resultado = M_SET_STATUS(modulo, address, informacoes);
                                        if(resultado == false) return false;
                                        if (modulo[address].SDEF & SEND_ID_VER) printf("VER: %x ID: %x\n", modulo[address].VER, modulo[address].ID);
                                        if (modulo[address].SDEF & SEND_TYPE) printf("TYPE: %x\n", modulo[address].TYPE);
                                        if (modulo[address].SDEF & SEND_TEMP) printf("TEMP: %x\n", modulo[address].temperature);
                                        if (modulo[address].SDEF & SEND_KEYBOARD)
                                        {
                while(1)
                {
                      resultado = M_SET_STATUS(modulo, address, informacoes);
                         printf("Imprimindo mapa de teclas..\n");
                         interpretaTecla(modulo[address].keyboard, mapa);
                         Sleep(1000);
                }
                                        }
                   printf("Digite S para coantinuar e N para escolher outra funcionalidade ou escolher outro escravo\n");
                   gets(&continuar);
                   }while(continuar == 'S');
                   }
                   extraiInfo = strstr(funcionalidade, "ARCONDICIONADO");
                   if(extraiInfo)
                   {
                            temperaturaPotencia = 0x00;
                            printf("Digite LIGA para ligar, digite DESLIGA para desligar, digite MUDA para mudar a temperatura e a potencia da ventilacao e digite JET_COOL para mudar para acionar modo Jet Cool\n");
                            gets(comandoArCondicionado);
                            extraiInfo = strstr(comandoArCondicionado, "LIGA");
                            if(extraiInfo)resultado = ARCONDICIONADO(modulo, address, LIGA, 0x00);
                            extraiInfo = strstr(comandoArCondicionado, "DESLIGA");
                            if(extraiInfo)resultado = ARCONDICIONADO(modulo, address, DESLIGA, 0x00);
                            extraiInfo = strstr(comandoArCondicionado, "JET_COOL");
                            if(extraiInfo)resultado = ARCONDICIONADO(modulo, address, JET_COOL, 0x00);
                            extraiInfo = strstr(comandoArCondicionado, "MUDA");
```

```
if(extraiInfo)
{
	printf("Digite a temperatura desejada, qualquer valor entre 18 e 30.\n");
	gets(temperatura);
	extraiInfo = strstr(temperatura, "18");
	if(extraiInfo)temperaturaPotencia |= 0x30;
	extraiInfo = strstr(temperatura, "19");
	if(extraiInfo)temperaturaPotencia |= 0x40;
	extraiInfo = strstr(temperatura, "20");
	if(extraiInfo)temperaturaPotencia |= 0x50;
	extraiInfo = strstr(temperatura, "21");
	if(extraiInfo)temperaturaPotencia |= 0x60;
	extraiInfo = strstr(temperatura, "22");
	if(extraiInfo)temperaturaPotencia |= 0x70;
	extraiInfo = strstr(temperatura, "23");
	if(extraiInfo)temperaturaPotencia |= 0x80;
	extraiInfo = strstr(temperatura, "24");
	if(extraiInfo)temperaturaPotencia |= 0x90;
	extraiInfo = strstr(temperatura, "25");
	if(extraiInfo)temperaturaPotencia |= 0xA0;
	extraiInfo = strstr(temperatura, "26");
	if(extraiInfo)temperaturaPotencia |= 0xB0;
	extraiInfo = strstr(temperatura, "27");
	if(extraiInfo)temperaturaPotencia |= 0xC0;
	extraiInfo = strstr(temperatura, "28");
	if(extraiInfo)temperaturaPotencia |= 0xD0;
	extraiInfo = strstr(temperatura, "29");
	if(extraiInfo)temperaturaPotencia |= 0xE0;
	extraiInfo = strstr(temperatura, "30");
	if(extraiInfo)temperaturaPotencia |= 0xF0;
	printf("Digite LOW para ventilacao fraca, MED para ventilacao media, HIGH
para ventilacao forte e AUTO para automatico.\n");
	gets(potencia);
	extraiInfo = strstr(potencia, "LOW");
	if(extraiInfo)temperaturaPotencia |= 0x00;
	extraiInfo = strstr(potencia, "MED");
	if(extraiInfo)temperaturaPotencia |= 0x02;
	extraiInfo = strstr(potencia, "HIGH");
	if(extraiInfo)temperaturaPotencia |= 0x04;
	extraiInfo = strstr(potencia, "AUTO");
	if(extraiInfo)temperaturaPotencia |= 0x05;
	resultado = ARCONDICIONADO(modulo, address, MUDA,
temperaturaPotencia);
	}
}
```

```
                                printf("Digite S para escolher outra funcionalidade ou N para escolher outra funcionalidade ou outro escravo\n");
                                gets(&continuar);
                        }while(continuar == 'S');
                }while(1);
// ========================================================================
    SioClose(Com_Port);
// ========================================================================
    return 0;
}
```

APÊNDICES

Apêndice 1 – Placa integrada do sensoriamento da mesa e acionamento do ar-condicionado.