UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTOS ACADÊMICOS DE ELETRÔNICA E MECÂNICA
CURSO SUPERIOR DE TECNOLOGIA EM MECATRÔNICA INDUSTRIAL

CAROLINA LANGFELDT
ISRAEL DE PAULA TEIXEIRA

AUTOMATIZAÇÃO DE ACIONAMENTOS POR MEIO DE TAREFAS AGENDADAS USANDO COMUNICAÇÃO POR RÁDIO FREQUÊNCIA

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2013

CAROLINA LANGFELDT
ISRAEL DE PAULA TEIXEIRA

AUTOMATIZAÇÃO DE ACIONAMENTOS POR MEIO DE TAREFAS AGENDADAS USANDO COMUNICAÇÃO POR RÁDIO FREQUÊNCIA

Trabalho de Conclusão de Curso de Graduação, apresentado ao Curso Superior de Tecnologia em Mecatrônica Industrial, dos Departamentos Acadêmicos de Eletrônica e Mecânica, da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Juliano Mourão Vieira

CURITIBA
2013

TERMO DE APROVAÇÃO

CAROLINA LANGFELDT
ISRAEL DE PAULA TEIXEIRA

AUTOMATIZAÇÃO DE ACIONAMENTOS POR MEIO DE TAREFAS AGENDADAS USANDO COMUNICAÇÃO POR RÁDIO FREQUÊNCIA

Este trabalho de conclusão de curso foi apresentado no dia 13 de Novembro de 2013, como requisito parcial para obtenção do título de Tecnólogo em Mecatrônica Industrial, outorgado pela Universidade Tecnológica Federal do Paraná. Os alunos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. Dr. Milton Luiz Polli
Coordenador de Curso
Departamento Acadêmico de Mecânica

_____
Prof. Esp. Sérgio Moribe
Responsável pela Atividade de Trabalho de Conclusão de Curso
Departamento Acadêmico de Eletrônica

BANCA EXAMINADORA

_____
Prof. M.Sc. Gabriel Kovalhuk
UTFPR

_____
Prof. João Almeida de Góis
UTFPR

_____
Prof. M.Sc. João Carlos Roso
UTFPR

_____
Prof. M.Sc. Juliano Mourão Vieira
Orientador - UTFPR

"A Folha de Aprovação assinada encontra-se na Coordenação do Curso"

RESUMO

LANGFELDT, Carolina; TEIXEIRA, Israel P. **Automatização de acionamentos por meio de tarefas agendadas usando comunicação por rádio frequência.** 2013. 63 f. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Mecatrônica Industrial), Departamentos Acadêmicos de Eletrônica e Mecânica, Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

Este trabalho relata o desenvolvimento de uma aplicação que permite a um usuário comum programar um agendamento de tarefas para ligar e desligar equipamentos e aparelhos em dias e horários predefinidos. O projeto consiste em um conjunto de *hardware* e *software* integrados que podem cumprir estas funcionalidades em diferentes ambientes e com diferentes necessidades. O sistema de *hardware* é modular, sendo composto por uma central de controle que se conecta em um computador via porta USB e módulos auxiliares que comandam suas respectivas cargas. Três códigos de *software* participam da solução: o *firmware* dos módulos de acionamento, o *firmware* da central de controle e o aplicativo de interface com o usuário, este último sendo executado em um computador. O desenvolvimento teve como objetivo não apenas o funcionamento do sistema, mas também a viabilidade econômica do projeto como bem de consumo. A funcionalidade do protótipo atendeu a todos os requisitos propostos, realizando o agendamento e a execução de tarefas com relativa comodidade e praticidade para o usuário.

Palavras chave: Automação. USB. Rádio frequência. Microcontroladores.

ABSTRACT

LANGFELDT, Carolina; TEIXEIRA, Israel P. **Drive automation through scheduled tasks using radio communication.** 2013. 63 f. Trabalho de Conclusão de Curso (Curso Superior de Tecnologia em Mecatrônica Industrial), Departamentos Acadêmicos de Eletrônica e Mecânica, Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

This work relates an application development that allows a common user to program a task scheduling to turn on and off equipments and devices in predefined dates and times. The project consists in an integrated group of hardware and software that can perform these functionalities in different environments with different needs. The hardware system is modular, being composed by a central control that connects a computer through an USB port and secondary modules that command their respective loads. Three software codes belong to the solution: the drive module firmware, the central control firmware and the user's interface application, the latter of which is executed on a computer. Development has as goals not only system correctness but also its economic feasibility as a consumer's asset. The prototype functionality complied with all requisites, performing the scheduling and the execution of tasks with relative commodity and practicality to the user.

Keywords: Automation. USB. Radio frequency. Microcontrollers.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

API	Application Programming Interface
ASK	Amplitude Shift Keying
AURESIDE	Associação Brasileira de Automação Residencial
BSD	Berkeley Software Distribution
CDC	Communication Device Class
CPU	Central Proccess Unit
EEPROM	Electrically-Erasable Programmable Read-Only Memory
EP	Endpoint
FSK	Frequency Shift Keying
HID	Human Interface Device
IBM	International Business Machines
IDE	Integrated Development Environment
JDBC	Java Database Connectivity
JDK	Java Development Kit
JNI	Java Native Interface
JSR	Java Specification Request
JVM	Java Virtual Machine
LED	Light Emissor Diode
MSD	Mass Storage Device
OOK	On Off Keying
PCB	Printed Circuit Board
PHDC	Personal Healthcare Device Class
PLC	Power Line Carrier ou Power Line Communication
PSK	Phase Shift Keying
RAM	Random Access Memory
RF	Rádio Frequência
SMD	Surface Mount Device
USB	Universal Serial Bus
USB-IF	Universal Serial Bus Implementers Forum

SUMÁRIO

1 INTRODUÇÃO

1.1 TEMA

Quando se fala em automação, o mais comum é pensar nas áreas industrial e comercial pelo fato de que, historicamente, é onde a automação teve e até hoje tem seu desenvolvimento mais acentuado. Porém, as demandas e iniciativas relativas à automação vêm se expandindo também para o ambiente predial e residencial.

Ao final da década de 1970, nos Estados Unidos, surgiram as primeiras instalações onde luzes eram pontualmente acionadas através de comandos enviados pela própria rede elétrica da residência, nascendo assim o conceito de PLC (Power Line Carrier ou Power Line Communication – Comunicações através de Linha de Força). À medida que a computação, a telefonia celular e a internet vêm evoluindo e se popularizando, essas tecnologias vêm sendo incorporadas ao controle de acionamentos em residências, com crescente demanda pelo mercado (MURATORI; DAL BÓ, 2011).

Essas demandas geralmente visam incrementar o conforto, a comodidade, a praticidade e a segurança do ambiente. Ligar e desligar luzes, irrigadores, alimentadores de animais, aquecedores, ventiladores, bombas de água, persianas elétricas, portões elétricos e sistemas de calefação remotamente e/ou automaticamente são algumas das funcionalidades que um sistema automatizado pode oferecer.

Muitas vezes o consumidor adquire soluções de automação simplesmente por luxo ou *status*, não tendo consciência do impacto positivo que os recursos terão sobre seu cotidiano. Com o passar do tempo, a otimização do tempo gasto com tarefas rotineiras e os incrementos de segurança e bem estar auferidos passam a desempenhar um papel bastante satisfatório na vida do usuário.

Os modelos de sistemas para automação têm variados graus de complexidade, mas em geral se compõem de atuadores localizados que são comandados por um controle central, além de um ou mais dispositivos para interface com o usuário. Esses componentes precisam ter um meio de intercomunicação, que pode ser a própria rede elétrica da residência, cabeamentos instalados ou

previamente planejados com a arquitetura do ambiente, ou ainda tecnologias wireless.

As tecnologias de interface podem ser dedicadas, como controles remotos ou painéis fornecidos pelo fabricante do sistema, ou aproveitando os recursos tecnológicos já disponíveis no ambiente, como computadores e telefones celulares.

Cada solução tem o seu escopo de atuação, suas desvantagens e vantagens competitivas e, logicamente, preços que atendem a diferentes níveis sociais. Por isso, ao introduzir um novo produto nesse mercado, o fabricante precisa considerar o equilíbrio entre custo e desempenho, buscando atender demandas que não são devidamente ou completamente satisfeitas pelo cenário comercial.

1.2 DELIMITAÇÃO DO ESTUDO

Este projeto foi concebido, desenvolvido e custeado integralmente pelos alunos participantes da equipe; seu limite de área de trabalho foi independente de fábricas e empresas, sendo construído em ambiente residencial e acadêmico. A dupla foi responsável pela aquisição dos componentes utilizados no protótipo, assim como as ferramentas necessárias para a montagem.

O projeto como um todo foi planejado para ocupar igualmente os meses ao longo do cronograma, fazendo aos poucos a construção e as pesquisas a respeito das melhores tecnologias a empregar ao trabalho de conclusão do curso, atendendo aos requisitos propostos em um tempo equilibrado com a rotina dos integrantes do grupo.

1.3 PROBLEMA

Há basicamente dois conceitos de automação residencial atuantes no mercado (OLIVEIRA, 2005):

No amplo universo da automação residencial observa-se que dois padrões distintos se destacam na maioria das aplicações:

1) Soluções integradas – empresas especializadas no ramo de automação que se dedicam a planejar, executar e manter soluções completas com alto grau de funcionalidade, conectividade e eficiência. São os chamados *integradores* (MURATORI, 2005). Estas soluções exigem tempo de planejamento,

muitas vezes alterações significativas no ambiente, equipamentos na maioria das vezes importados e, sobretudo, disponibilidade financeira do cliente para cobrir gastos com equipamentos, serviços de diversos tipos e contratos de manutenção. Os custos podem variar entre alguns milhares e até dezenas de milhares de Reais.

2) Soluções pontuais – equipamentos destinados a executar tarefas simples em ambientes e aparelhos não planejados para receber automação. Os maiores compromissos desse tipo de automação são o baixo custo de aquisição e a facilidade de instalação. Contudo, as interfaces de controle oferecidas têm poucos recursos e as soluções são bastante inflexíveis com relação à funcionalidade.

Há situações, porém, em que recursos inteligentes são exigidos para atender determinadas necessidades, independentemente da classe social do usuário. Temos como exemplo situações em que o morador, ao estar ausente da sua residência, deseja que determinadas ações sejam executadas em dias e horários programados, como acender e apagar luzes, alimentar seus animais de estimação ou de criação, acionar sistemas de irrigação e limpeza de piscinas. Automatizar essas tarefas com base em agendamentos programados economiza tempo e deslocamentos, garante regularidade na execução das mesmas, e em alguns casos proporciona maior segurança ao patrimônio do usuário durante seu período de ausência.

Porém os equipamentos disponíveis no mercado que atendem tais necessidades pertencem ao grupo das soluções integradas, citado anteriormente. Desta maneira, pessoas com menor poder aquisitivo ficam sem boas alternativas para executar acionamentos com base em dias e horários agendados.

1.4 OBJETIVOS

Nesta seção são apresentados os objetivos geral e específicos do trabalho, relativos ao problema anteriormente apresentado.

1.4.1 Objetivo Geral

Automatizar o funcionamento de equipamentos eletro-eletrônicos residenciais através de tarefas agendadas.

1.4.2 Objetivos Específicos

● Desenvolver o hardware para módulo central e módulos auxiliares;

● Desenvolver um aplicativo desktop capaz de criar e armazenar em banco de dados uma lista de tarefas agendadas;

● Implementar funcionalidade de comunicação USB entre o computador e o sistema microcontrolado;

● Definir um protocolo de comunicação entre o aplicativo da interface e o software do microcontrolador;

● Implementar escrita na memória flash do microcontrolador com alocação dinâmica de endereços;

● Realizar comunicação bidirecional por rádio frequência entre o módulo central e os módulos auxiliares;

● Gerenciar e executar as tarefas gravadas no módulo central com base na verificação de data e hora dos agendamentos.

1.4.3 Justificativa

A solução aqui proposta disponibiliza, para pessoas de diferentes níveis sociais, uma tecnologia acessível, prática, portátil e livre da dependência de prestadores de serviços, para automatizar tarefas por meio de agendamentos.

Através de um sistema composto por central de controle, módulos de acionamento e *software* de interface, o usuário fica apto para agendar acionamentos com toda a complexidade de opções que um aplicativo *desktop* pode oferecer. Computadores e *notebooks* estão cada vez mais acessíveis para as populações de baixa renda, e aproveitar recursos computacionais e interfaces já disponíveis no ambiente significa beneficiar economicamente o consumidor pelo uso compartilhado desses recursos.

Em adição, o conjunto de *hardware* e *software* embarcado apresentados por este projeto, ao fazer um uso eficiente de componentes bem dimensionados, proporciona uma ótima relação custo/desempenho para os equipamentos.

Embora a presente proposta esteja longe de ser uma solução definitiva no âmbito da automação residencial, a mesma pretende significar um passo a mais na direção de popularizar tecnologias que já estão presentes no mercado, porém ainda muito onerosas para grande parte do público.

1.4.4 Procedimentos Metodológicos

O desenvolvimento do *hardwar*e teve o seu maior enfoque no custo de produção, contemplando uma possível fabricação em série. Isso influenciou diversas escolhas, desde os componentes eletrônicos até o processo de fabricação das placas de circuito impresso.

O desenvolvimento do software baseou-se em três premissas: a primeira para atender as necessidades do projeto, cuja característica mais crítica é a conectividade USB em multiplataforma; a segunda com relação ao programador, que consiste na escolha de linguagens e ferramentas gratuitas com amplo suporte e possíveis de serem dominadas em tempo hábil; e a terceira para o usuário final que irá operar o *software* de interface de maneira mais prática possível.

O sistema como um todo foi concebido procurando equilibrar a facilidade de instalação e manuseio, o custo de aquisição e a utilização das tecnologias mais atuais, resultando num custo-benefício atrativo para o mercado brasileiro.

1.4.5 Embasamento Teórico

Este trabalho se fundamenta primeiramente na bagagem de conhecimento adquirida ao longo do curso de Tecnologia Industrial Mecatrônica da UTFPR, através das disciplinas de Sistemas Microprocessados, Linguagem de Programação, Gestão de Projetos, entre outras. Os documentos e códigos fornecidos pela Freescale Inc. também constituem fonte de informação imprescindível para o desenvolvimento.

Em pontos mais específicos, este trabalho se fundamenta no artigo de Ayre e Pfeiffer (2010) para uso da memória Flash, nos trabalhos de Fialho (2011), Fontes (2012) e Laskoski et al (2006), para modulações digitais, nos artigos de Baptista et al (2009) e Cesta (2009) para a plataforma Java, além de Bastos e Caproni, Friesen

(2006), Insam (2002), Jiang (2003), Muller (2012), Silicon Labs (2011) e USB-IF (2000) para a integração da plataforma Java com a porta USB.

Em termos mais conceituais, são relevantes os artigos de MURATORI (2005) e OLIVEIRA (2005).

1.4.6 Estrutura do Trabalho

O trabalho possui a estrutura apresentada a seguir.

Capítulo 1 - Introdução: São apresentados o tema, as delimitações da pesquisa, o problema e a premissa, os objetivos da pesquisa, a justificativa, os procedimentos metodológicos, as indicações para o embasamento teórico, e a estrutura geral do trabalho.

Capítulo 2 – Fundamentação teórica: neste capítulo são abordados os meios necessários para realizar o projeto, como a interface USB, a linguagem JAVA e o microcontrolador.

Capítulo 3 – Desenvolvimento do Tema: são descritos os procedimentos de desenvolvimento do *hardware*, do *software* da central de controle e do módulo de acionamento, e da interface do usuário.

Capítulo 4 – Resultados: tendo como base os procedimentos metodológicos, neste capítulo são descritos os resultados obtidos e feitas as devidas análises relacionadas.

Capítulo 5 – Considerações finais: com base nos objetivos estabelecidos, o projeto é qualificado quanto à sua utilidade, sua viabilidade e suas limitações, e são apresentadas outras possíveis soluções que podem derivar da presente pesquisa.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 USB

Em 1995 foi criado o *USB Implementers Forum*, um consórcio formado pelas empresas Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, Nec e Philips. O objetivo era padronizar a comunicação entre o computador e seus periféricos (MOTA, 2007). Como resultado, foi criado um novo barramento: o USB (Universal Serial Bus - Barramento Serial Universal).

2.1.1 Por Que Usar USB?

O USB é atualmente a forma mais comum de comunicação entre um computador e seus periféricos quando estes requerem taxas de transferência de dados de baixa ou média velocidade. Muller (2012) afirma que projetar e construir dispositivos USB ainda é uma tarefa difícil, pois a documentação pertinente é extensa e as Interfaces de Programação de Aplicativos (APIs) oferecidas para desenvolvimento são geralmente complexas. Entretanto, a principal alternativa para o uso da porta USB, o conector RS-232, está rapidamente se tornando obsoleta e mesmo ausente nos computadores comprados recentemente (INSAM, 2002). Em adição, de acordo com a USB-Implementers Forum (2000, P.11) a especificação USB proporciona diversas características ao produto que contribuem para uma melhor relação custo / desempenho do mesmo.

2.1.2 Funcionamento do USB

Toda comunicação feita através do barramento USB é iniciada pelo *host* do sistema com o envio de um pacote de dados chamado *Token Packet* (MOTA, 2007). No caso de comunicações entre computador e periférico, o computador sempre faz o papel de *host*. Uma vez que o dispositivo USB decodifica esse primeiro pacote, a

transferência dos dados acontece através do envio de *Data Packets*, que podem ser tanto no sentido do *host* para o *device* como do *device* para o *host.* A comunicação é finalizada com um *Handshake Packet* que é enviado pelo transmissor dos *Data Packets.* Observa-se, portanto, que a comunicação USB tem uma arquitetura do tipo Cliente-Servidor.

Do lado do dispositivo, a estrutura de software do USB uma composição que pode ser observada na figura 1, segundo Bastos e Caproni (2009, P.19):

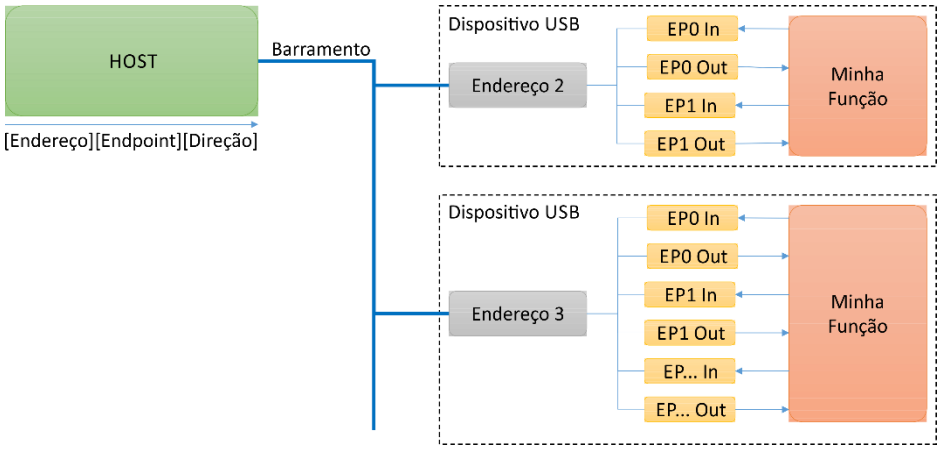


Figura 1 - Estrutura de funcionamento da comunicação USB.
Fonte: Adaptado de Bastos e Caproni (2009).

Pode-ser ver na figura 1 os *endpoints* de entrada (*EP0 In, EP1 In*) e saída (*EP0 Out, EP1 Out*). Considerando um contexto onde o dispositivo é um microcontrolador com USB integrado, de acordo com Bastos e Caproni (2009) esses *endpoints* são *buffers* de memória que armazenam os dados enviados pelo *host* até que o software do microcontrolador os leia e proceda seu tratamento, ou recebem os dados que o software do microcontrolador quer encaminhar para o *host*. Assim, o controlador USB do dispositivo não é gerenciado pelo microcontrolador, mas sim pelo *host*. “Os *endpoints* podem ser considerados uma interface entre o hardware do dispositivo USB e o software do microcontrolador.” (BASTOS e CAPRONI, 2009).

Do lado do *host*, o software cliente se utiliza de conexões lógicas entre o *host* e os *endpoints.* São os chamados *pipes* que, segundo Friesen (2006), definem o fluxo de dados entre *host* e dispositivo.

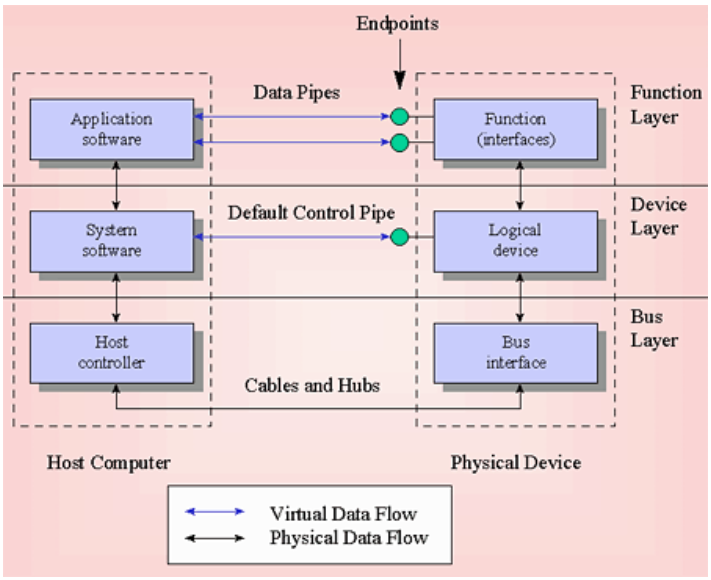


Figura 2 - Modelo de fluxo de dados na comunicação USB.
Fonte: FRIESEN (2006).

A figura 2 mostra que há um nível físico de tráfego de dados, controlado inteiramente pelo sistema de hardware, um nível lógico onde se encontra a atuação do *driver* do dispositivo, e um nível de aplicação, onde virtualmente os dados fluem entre a aplicação do *host* e a aplicação do dispositivo. Os *pipes* recebem como atributo o tipo de transferência de dados a que são destinados (FRIESEN, 2006). Esses tipos podem ser:

- *Bulk* – usados para transferir grande quantidade de dados. Este tipo garante a integridade dos dados transferidos;
- *Control* – dados estruturados na forma de comandos ou informações sobre status;
- *Isochronous* – os dados são transferidos em tempo real a uma taxa fixa, mas pode haver perda de dados;
- *Interrupt* – funciona de forma semelhante ao tipo *bulk*, porém com pequena quantidade de dados e garantia de resposta rápida.

Todo dispositivo USB utiliza descritores, que são estruturas de dados que permitem ao dispositivo ser identificado pelo software que está sendo executado na máquina *host* (FRIESEN, 2006). Há diversos tipos de descritores como: descritores

de dispositivo, de configuração, de interface e de *endpoints.* A forma como esses descritores estão configurados irá determinar a classe na qual o dispositivo se encaixa e consequentemente qual o *driver* de dispositivo que será carregado pelo sistema operacional no momento da conexão.

2.1.3 HID

A USB Implementers Forum especifica diversas classes de dispositivos. Uma delas é a HID (Human Interface Device – Dispositivo de Interface Humana). Um dispositivo USB que pertence à classe HID normalmente é usado em aplicações de interação humana como *mouses* e teclados. A classe HID, no entanto, não restringe o uso dos dispositivos, apenas define os requerimentos básicos do dispositivo e o protocolo para transferência de dados (SILICON LABS, 2011, P.4). Portanto, um dispositivo pode perfeitamente pertencer à classe HID mesmo que sua função não dependa diretamente da interação humana.

A Silicon Labs afirma que um dispositivo HID deve atender aos seguintes requerimentos:

- O *endpoint* 0 deve estar configurado para operar com *pipes* do tipo *control*;
- A conexão deve usar um único *endpoint* do tipo IN configurado como *interrupt*. Adicionalmente pode haver um *endpoint* do tipo OUT também configurado como *interrupt*;
- Todos os dados transferidos devem estar formatados de acordo com a configuração dos descritores *report* do dispositivo*;*
- Os dispositivos devem responder às requisições padrão do *driver* HID além das requisições do padrão USB.

Uma vez que o dispositivo atenda a esses requisitos, ao conectá-lo na porta USB o sistema operacional automaticamente carrega seu *driver* HID nativo, identifica o dispositivo e configura o *driver* de acordo com os descritores do dispositivo.

Além da HID, existem outras classes como PHDC (Personal Healthcare Device Class – Classe de Dispositivo de Cuidados Pessoais), MSD (Mass Storage Device – Dispositivo de Armazenamento em Massa), CDC (Communication Device Class – Classe de Dispositivo de Comunicação) e Audio.

2.2 JAVA

2.2.1 Características

Existem diversas vantagens no uso da linguagem Java. Cesta (2009) aponta algumas delas: semelhança com C e C++, portabilidade, orientação a objetos, segurança, eficiência, suporte a concorrência (múltiplas *threads*) e suporte a distribuição.

2.2.2 JVM

Quando um aplicativo escrito em Java é compilado, são gerados *bytecodes*, de forma semelhante às outras linguagens, porém esses *bytecodes* não são executados diretamente pelo sistema operacional, mas pela JVM (Java Virtual Machine – Máquina Virtual Java).

> A JVM (do inglês Java Virtual Machine) é um programa que carrega e executa os aplicativos Java, convertendo os bytecodes em código executável de máquina...
>
> A JVM é reponsável pelo gerenciamento dos aplicativos, à medida que são executados (BAPTISTA; PALANGANI; DEOTOLDO, 2009).

Assim, o mesmo aplicativo Java pode ser executado em ambiente Linux, Mac OS ou Windows, sem precisar ser previamente compilado para cada sistema operacional. Segundo Baptista, Palangani e Deotoldo (2009), as principais estruturas da JVM são:

- Carregador de classes;
- Máquina de execução;
- Áreas de dados de execução;
- Interface de métodos nativos.

A plataforma Java conta ainda com um coletor de lixo. “O coletor de lixo libera a memória mantida por um objeto se não houver mais nenhuma referência para ele” (BAPTISTA; PALANGANI; DEOTOLDO, 2009). Isso desobriga o programador de fazer manualmente a liberação de memória dos objetos que não são mais utilizados, e por isso não há necessidade de destrutores nas classes Java, como ocorre em C++.

2.2.3 Java e USB

A plataforma Java oferece pouca funcionalidade no que diz respeito ao acesso ao *hardware*. Segundo Jiang (2003), é relativamente mais fácil desenvolver uma aplicação que acesse dispositivos USB usando C ou C++ do que usando Java. Esforços foram realizados no sentido de prover a plataforma Java de acesso ao USB. A IBM iniciou o desenvolvimento do pacote JSR-80 em 1999 e em 2000 surgiu uma segunda iniciativa com o nome de jUSB. Ambos os projetos atingiram boa funcionalidade em ambiente Linux, embora com imperfeições, e houve tentativas de expandir o escopo desses projetos para o ambiente Windows, porém não houve continuidade no desenvolvimento, e até hoje a plataforma Java não tem suporte oficial para a porta USB (JIANG, 2003).

Recentemente têm surgido novos projetos Java que oferecem funcionalidade do USB em multi-plataforma, inclusive com código fonte aberto. Estas iniciativas, porém, são independentes e não recebem suporte da empresa Oracle, que é proprietária da plataforma Java.

2.3 MICROCONTROLADORES

2.3.1 Estrutura de Hardware

Entre os microcontroladores de menor custo do mercado e que possuem uma confiabilidade reconhecida, estão os modelos da família MC9S08, originalmente da Motorola, agora produzidos pela Freescale Semiconductor. Há grande similaridade na arquitetura e na forma de programação entre os produtos desta família. Os modelos, entretanto, possuem configurações bastante variadas. As figuras 3 e 4 exemplificam as semelhanças e diferenças entre dois modelos de microcontroladores da família S08.

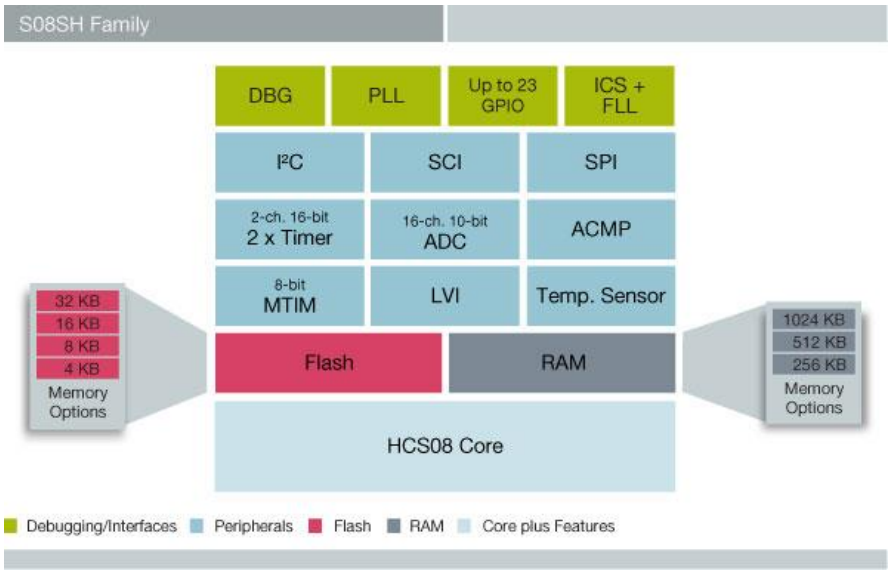


Figura 3 - Estrutura do microcontrolador S08SH.
Fonte: Freescale Semiconductor (www.freescale.com)

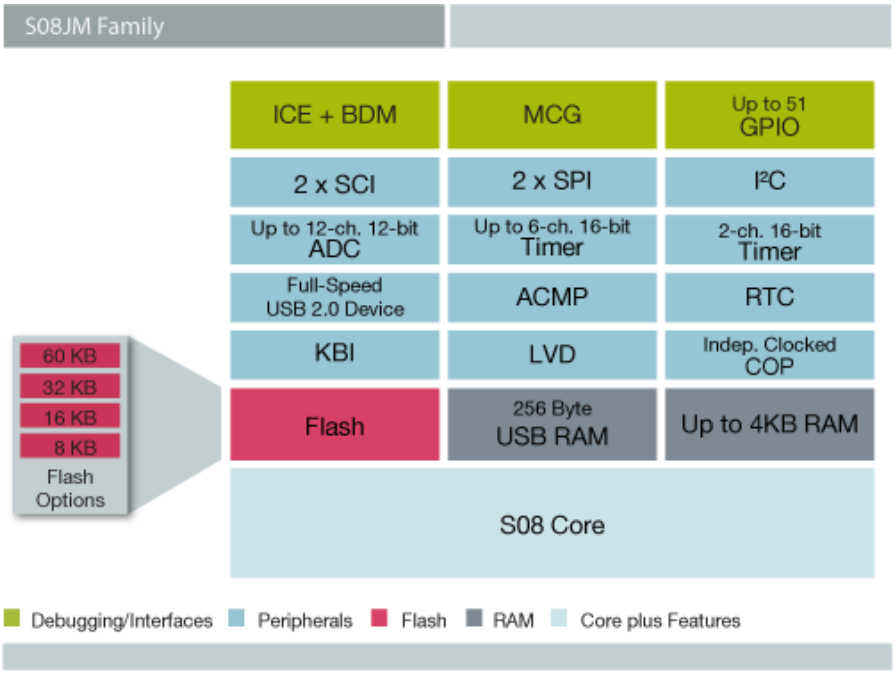


Figura 4 - Estrutura do microcontrolador S08JM.
Fonte: Freescale Semiconductor (www.freescale.com)

Como pode ser verificado na figura 4, o grupo de microcontroladores S08JM possui um dispositivo USB integrado, que possibilita a construção de aplicações dotadas de comunicação USB usando um único chip. Observa-se também que um S08JM recebe em sua construção quantidades maiores de memórias RAM (Random Access Memory – Memória de Acesso Aleatório) e *Flash*, necessárias para garantir a operabilidade de uma biblioteca USB na camada de aplicação do dispositivo.

2.3.2 Memória *Flash*

A memória *Flash* dos microcontroladores pode ser escrita em tempo de execução *byte* a *byte*, mas só pode ser apagada em blocos (AYRE e PFEIFFER, 2010). Cada modelo de memória, ou microcontrolador, possui um tamanho específico para o número de *bytes* que constituem um bloco.

Além disso, o número de ciclos de apagamento / reescrita garantidos é geralmente limitado, tipicamente em torno de 10.000 vezes ou até 100.000 vezes (AYRE e PFEIFFER, 2010).

Quando uma aplicação precisa fazer gravações frequentes na memória flash, esses fatores passam limitar consideravelmente a vida útil do equipamento, muitas vezes inviabilizando a solução, a menos que haja um gerenciamento inteligente da memória que contorne as limitações físicas presentes.

Uma técnica eficiente para economizar ciclos de apagamento / reescrita é apresentada por Ayre e Pfeiffer (2010), onde uma tabela composta por um bloco inteiro de memória é usada para armazenar a mesma informação múltiplas vezes. Quando se deseja modificar uma informação gravada na *Flash*, procedem os seguintes passos:

a) Alocar um novo endereço para o dado no final da tabela;
b) Marcar o novo endereço como atual;
c) Marcar o endereço anterior como expirado;
d) Retornar um ponteiro com o novo local da variável.

Dessa forma, quando o dado em questão precisa ser modificado ele é apenas reescrito em um novo endereço, e somente quando todos os endereços forem preenchidos é que o bloco será apagado e a tabela será utilizada novamente a partir do primeiro endereço.

O efeito desta técnica sobre a vida útil da memória será diretamente proporcional à relação entre o tamanho da tabela e o tamanho dos dados, de acordo com a seguinte fórmula:

$$(N° \; de \; bytes \; do \; bloco \;)/(N° \; de \; bytes \; do \; registro) \; \times N° \; de \; ciclos \; de \; escrita$$

Por exemplo: se o bloco de memória possui 512 *bytes*, o dado ocupa 4 *bytes*, e a memória tem 10.000 ciclos de apagamento e reescrita garantidos, então o dado poderá ser modificado 1.280.000 vezes antes que o limite de ciclos da memória seja atingido.

No passado, armazenar dados que permanecessem disponíveis após desligar e religar um aparelho exigia uma memória adicional do tipo EEPROM (Electrically-Erasable Programmable Read-Only Memory – Memória Somente Leitura Programável Apagável Eletricamente). Hoje isso não é mais necessário devido à

funcionalidade que pode ser agregada à memória *Flash* dos microcontroladores (AYRE e PFEIFFER, 2010).

2.4 MODULAÇÃO DIGITAL

O uso de uma frequência moduladora é obrigatório para realizar transmissões sem cabos, conforme Fialho (2011, p. 1):

> Se o canal de transmissão for o espaço livre, cujo comportamento em frequência é passa-banda, torna-se necessário proceder à modulação do sinal a emitir por forma a deslocar o espectro para uma banda de frequência onde a atenuação do canal seja aceitável.

Genericamente podemos definir uma transmissão modulada conforme a figura:

Figura 5 - Sistema de comunicação digital genérico.
Fonte: Fialho (2011).

Os métodos mais simples de se modular dados digitalmente para transmissão por meio de ondas eletromagnéticas são as chamadas modulações binárias, onde cada *bit* é codificado individualmente em função de uma portadora. Essa codificação é obtida em função da variação dos parâmetros fundamentais de uma senóide: amplitude, frequência e fase (FIALHO, 2011).

2.4.1 ASK e OOK

À modulação binária que varia a amplitude do sinal chamamos ASK (Amplitude Shift Keying – Chaveamento por Variação de Amplitude). Para se gerar um sinal com modulação ASK basta alterar o valor da amplitude do sinal de saída em função do bit a transmitir (FONTES, 2012). As principais características da modulação ASK são:

- Simplicidade no processo de modulação e demodulação;
- Eficiência da largura de banda;
- Baixa imunidade ao ruído.

Fontes (2012) afirma que a modulação por chaveamento de amplitude é indicada em ambientes de comunicação com pouco ruído ou quando o baixo custo do sistema é essencial.

Uma variação da modulação ASK é a modulação OOK (On Off Keying – Chaveamento Liga / Desliga). Nesta, o sinal de saída tem o *bit* 1 equivalente a uma amplitude diferente de zero enquanto o *bit* 0 tem amplitude zero. Na prática, o transmissor é ligado e desligado sequencialmente de acordo com os bits transmitidos. A figura 11 mostra a composição de um sinal OOK:

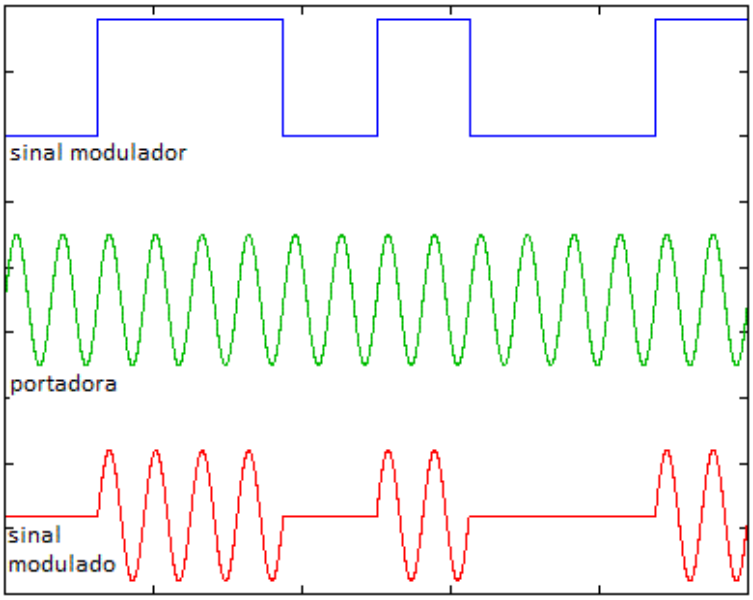


Figura 6 - Sinal modulado em OOK.
Fonte: <http://engweb.info/courses/wdt/lecture01/Wireless_Fundamentals.html>

2.4.2 FSK

A modulação FSK (Frequency Shift Keying – Chaveamento por Variação de Frequência) varia a frequência do sinal em função do *bit* transmitido. De acordo com Laskoski, Marcondes e Szeremeta (2006), a modulação FSK é mais imune a ruídos que a ASK, porém exige um circuito receptor mais elaborado.

2.4.3 PSK

Na modulação PSK (Phase Shift Keying – Chaveamento por Variação de Fase) ocorre a alteração discreta da fase da portadora conforme o sinal digital a ser modulado. Portanto, pode-se por exemplo manter a fase da portadora em 0° quando ocorrer um bit 1 e alterar a fase da portadora quando ocorrer um bit 0 (LASKOSKI; MARCONDES; SZEREMETA, 2006).

3 DESENVOLVIMENTO

3.1 HARDWARE

3.1.1 Projeto Eletrônico

Os circuitos eletrônicos da central de controle e do módulo de acionamento possuem blocos comuns, como é mostrado na figura 7.

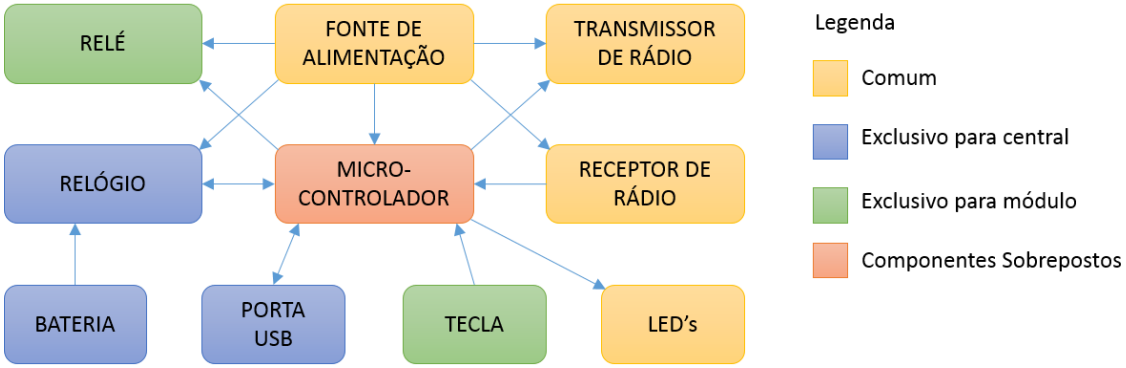


Figura 7 - Diagrama de blocos do hardware.
Fonte: Autoria própria.

Como boa parte do espaço físico do circuito é ocupada pelos elementos comuns aos dois dispositivos, tornou-se economicamente vantajoso confeccionar uma única placa de circuito híbrida que comporta todos os blocos. Os dispositivos foram diferenciados no processo de montagem da placa.

Para elaboração do esquema elétrico do projeto e desenho da placa de circuito impresso foi utilizado o *software* Altium Designer. O software foi escolhido por ser de prévio domínio técnico dos participantes da equipe, além de ser um dos melhores softwares do mercado para projeto de PCB´s.

Foram confeccionados quatro protótipos da placa de circuito impresso, no Laboratório de Tecnologia em Circuito Impresso e Eletrodeposição - LACIE, dirigido pelo professor Fernando Castaldo.

3.1.2 Fonte de Alimentação

Com base em testes realizados previamente pela equipe, ainda na fase de planejamento do projeto, verificou-se que circuitos demoduladores de rádio frequência semelhantes aos usados neste projeto, quando alimentados por fontes chaveadas, tendem a apresentar baixo alcance de recepção em relação à potência transmitida. O mesmo efeito não acontece quando a fonte é do tipo linear. Por isso, ambos os dispositivos são alimentados com fontes lineares conforme mostra o esquema elétrico em anexo. O circuito da fonte consiste em um transformador linear operando em 60Hz, um retificador em ponte completa, capacitor de filtro e reguladores para 12V e 5V.

Para manter a funcionalidade *bivolt* característica das fontes chaveadas, foi usado um transformador com enrolamento primário de 220V e secundário de 24V, seguido de um regulador de tensão de 12V. Assim, foi possível obter uma tensão de alimentação de 12V conectando o dispositivo em uma tomada de 220V ou 127V, sem a necessidade de uma chave seletora.

3.1.3 Transmissor e Receptor de Rádio

Os circuitos de transmissão e recepção de rádio frequência seguem um padrão largamente utilizado pela indústria de produtos de consumo com boa relação custo/desempenho para um alcance variando entre 15 e 50 metros, dependendo de diversas variáveis do ambiente como paredes, lajes, estruturas metálicas, posicionamento das antenas, etc. O circuito de transmissão é baseado em um ressonador SAW, com um único estágio de potência a transistor, que dissipa aproximadamente 100mW. A antena de transmissão é impressa na própria placa de circuito. Já o receptor conta com uma antena externa feita de fio flexível e quatro estágios de amplificação, sendo o primeiro responsável pelo casamento da impedância entre a antena e o oscilador LC do segundo estágio. Este se encarrega de permitir a sintonia da frequência através da variação do indutor do circuito LC. Os

dois últimos estágios produzem a digitalização do sinal analógico que será demodulado pelo microcontrolador. Um osciloscópio e um transmissor 433,92Mz foram usados para realizar a calibração dos receptores de rádio.

A frequência utilizada foi de 433,92MHz tanto para a transmissão quanto para a recepção, significando que a comunicação opera em modo *half-duplex.* O circuito utiliza uma modulação do tipo OOK. Essa configuração permitiu que os módulos de acionamento fossem compatíveis com controles remotos disponíveis no mercado.

Ambas as montagens de transmissor e receptor são *on-board*, ou seja, a placa de circuito que comporta microcontrolador e demais componentes agrega também os circuitos de rádio frequência.

3.1.4 Microcontroladores

A central de controle utiliza um microcontrolador MC9S08JM60 da Freescale, com 4kB de memória RAM e 60kB de memória *Flash*. O *chip* utiliza um cristal de 12MHz para geração do *clock*, pois embora seja possível gerar o *clock* internamente, a máquina USB integrada exige que se use um cristal externo. O microcontrolador sinaliza as operações de transmissão e recepção de rádio através de dois LED's específicos para este fim, além de manter um terceiro LED piscando apenas para indicar o funcionamento. A central também indica, através de um quarto LED, quando há ocorrência de tarefas canceladas por falta de energia elétrica.

O módulo de acionamento usa um microcontrolador mais simples e de menor custo: o MC9S08SH4, com 4kB de memória *Flash* e apenas 256 *bytes* de memória RAM, usando *clock* interno de 8MHz.

Foram vários os fatores que culminaram na escolha das CPU's da Freescale: tamanho compacto, facilidade de programação usando o ambiente CodeWarrior, ampla documentação, sobretudo em relação ao uso da porta USB, e códigos prontos ou semi-prontos para implementação no projeto. Além disso, a

equipe obteve disponibilidade para aquisição desses componentes em pequenas quantidades a preço de atacado.

3.1.5 Relógio Interno

O circuito de relógio/calendário usado na central de controle utiliza um cristal de 32,768 kHz e se mantém em funcionamento por uma bateria CR-2032 quando a central está sem energia. O *layout* da placa de circuito impresso tem compatibilidade com a disposição física dos terminais dos relógios DS1307 da Maxim, M41T82 da STMicroeletronics, PCF8583 da NXP e HT1381 da Holtek. No protótipo foi usado um *chip* HT1381.

3.1.6 Gabinete

Para acomodação das placas de circuito, tanto para central de controle como para módulo de acionamento, utilizou-se um gabinete plástico modelo PB-107 de fabricação da Patola, cujo desenho e dimensões se encontram em anexo.

Os gabinetes receberam as devidas furações para permitir a passagem do conector USB, LEDs, tecla, cabo de alimentação, porta-fusível e saída de acionamento, de acordo com a função do dispositivo. Para simular o encaixe da placa de circuito no gabinete, este foi desenhado no software SolidWorks segundo as medidas especificadas pela Patola através do site <http://www.patola.com.br> e posteriormente integrado ao projeto da PCB no Altium.

3.2 *SOFTWARE* DA CENTRAL DE CONTROLE

Funções básicas do programa:

- Cadastrar novos dispositivos de acionamento;
- Receber tarefas pela porta USB e armazená-las na memória *Flash*;

- Ler o conteúdo da memória *Flash* periodicamente comparando as datas das tarefas com a data do relógio;
- Comandar os dispositivos de acionamento de acordo com os parâmetros das tarefas;
- Informar *status* dos dispositivos e das tarefas ao aplicativo cliente quando solicitado.

De forma geral, a estrutura do *software* aplicado à central de controle segue o diagrama da figura 8.

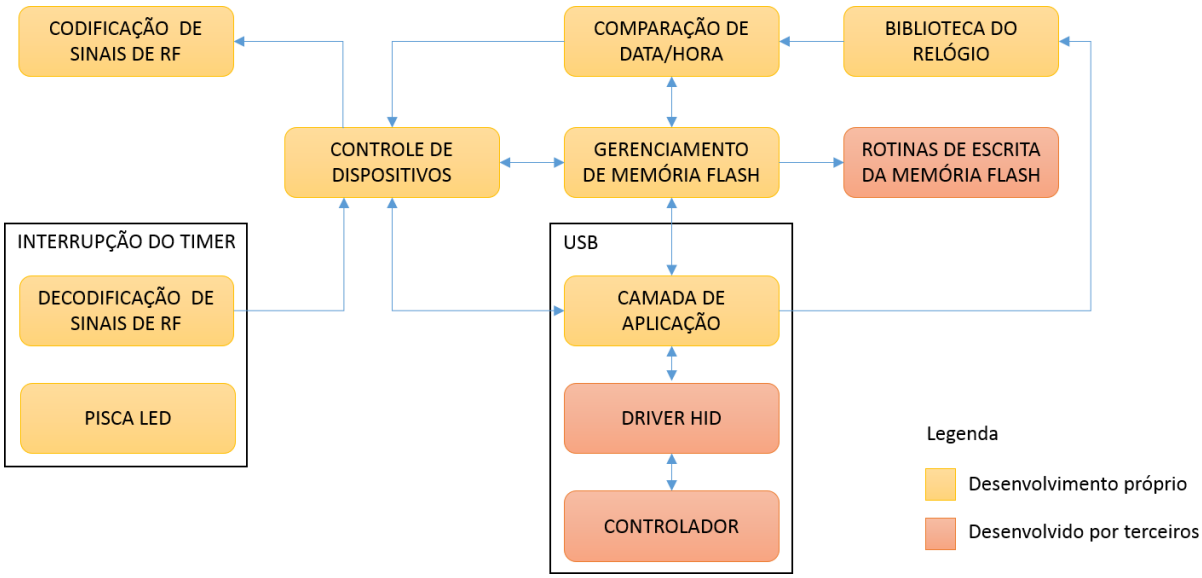


Figura 8 - Diagrama de blocos do software da central de controle.
Fonte: Autoria própria.

O código foi escrito em linguagem C, utilizando como editor o programa CodeWarrior v10.3 para Windows, compilador MinGW e, para gravação e *debug,* utilizou-se uma interface de programação da USBDM – um projeto *opensource* disponibilizado pela Freescale que usa o mesmo microcontrolador HC9S08JM60 usado neste projeto.

3.2.1 Protocolos de Comunicação

Para realizar a comunicação USB entre central de controle e computador, foi implementada uma pilha de protocolo desenvolvida pela CMX Systems. A Freescale publicou os documentos AN3492 e AN3565 que auxiliaram no entendimento e processo de implementação do protocolo. A pilha foi previamente configurada pela CMX Systems de tal forma que o *host* reconheça o *hardware* como um dispositivo HID (ver 2.1.3). Apenas os descritores USB são modificados para personalizar a aplicação de acordo com o desejado.

Para a camada de aplicação também existe um protocolo cujas definições previstas se encontram em anexo.

3.2.2 Uso da Memória *Flash*

As rotinas de escrita na memória *Flash* são fornecidas pela Freescale em códigos Assembly, porém totalmente utilizáveis em linguagem C através de funções exportadas. O documento AN3942 da Freescale explica como utilizar as rotinas durante a programação.

Tendo em vista que o programa modifica o conteúdo da memória *Flash* com relativa frequência, e de acordo com a abordagem feita no item 2.3.2 da fundamentação teórica deste projeto, existe uma função responsável por alocar dinamicamente endereços para dados de dispositivos e de tarefas, além de reciclar blocos de memória já utilizados sempre que necessário. Esta função segue o fluxograma da figura 9.

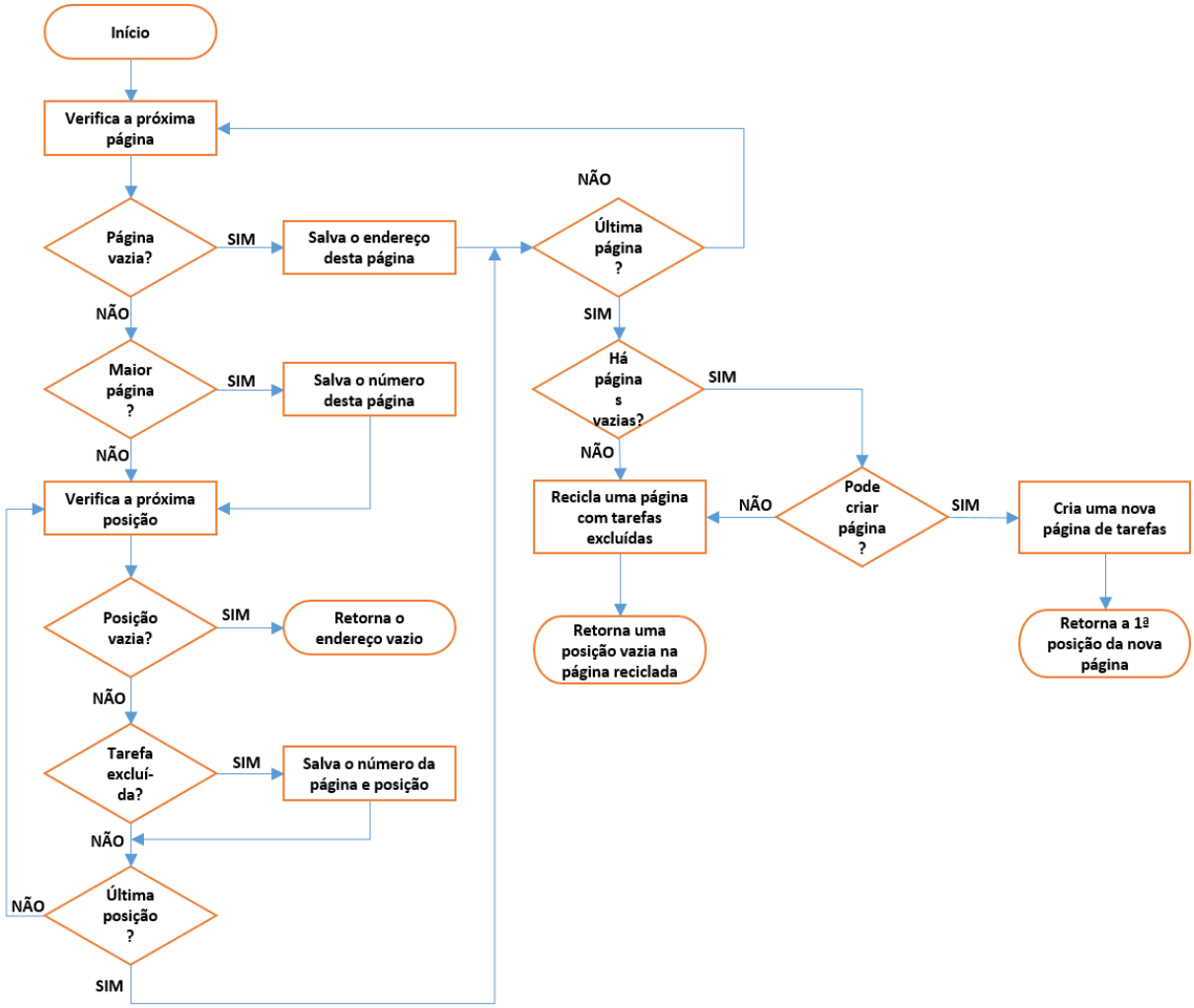


Figura 9 - Fluxograma da função de gerenciamento da memória Flash.
Fonte: Autoria própria.

3.2.3 Codificação Digital

Toda a comunicação entre a central de controle e os dispositivos de acionamento são feitas exclusivamente por meio do canal de rádio frequência. Para codificação dos dados foi usado o protocolo do *encoder* HT6P20, cujo formato pode ser visto na figura 10.

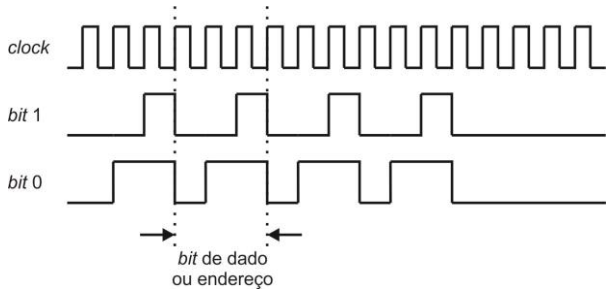


Figura 10 - Codificação do encoder HT6P20.
Fonte: Holtek

O protocolo usa 24 *bits* de endereço e 2 *bits* de dados, sendo os *bits* de dados utilizados conforme a figura 11.

Dado	Comando enviado pela central	Resposta do dispositivo
01	Ligar relé	Relé ligado
10	Desligar relé	Relé desligado
11	Informar *Status*	_____

Figura 11 - Protocolo de comunicação de rádio frequência
Fonte: Autoria própria

Como pode ser observado na figura 10, a diferenciação entre 0 e 1 no sinal de rádio é dada pela largura dos pulsos, podendo-se interpretar “pulsos largos” como 0 e “pulsos curtos” como 1, ou o contrário se forem considerados os intervalos entre pulsos. Um código do padrão HT6P20 contém 29 bits: 1 start bit, 22 bits de endereço, 2 bits de dados e 4 bits de sufixo. Cada bit tem a duração de três ciclos de *clock* do *encoder*. Esse *clock* normalmente varia entre 300 µs e 500µs. Logo, uma transmissão completa pode durar entre 26ms e 44ms, que pode ser um tempo longo demais para manter o programa dedicado exclusivamente à leitura de RF.

Para proporcionar ao *software* a capacidade de decodificar essas transmissões em tempo real, rejeitando automaticamente qualquer ruído presente no canal de RF e sem perdas de desempenho para as outras funções do programa, foi desenvolvido um novo método para decodificação do HT6P20. O método consiste em ler o estado da porta de RF a cada 31.25µs, usando a interrupção do *timer* do microcontrolador. Pela comparação entre a leitura da última interrupção e a imediatamente anterior, pode-se identificar a ocorrência de rampas de subida e

descida na porta, e consequentemente a largura dos pulsos e a largura dos intervalos entre pulsos contando-se o número de interrupções consecutivas que antecedem uma rampa. Assim, os bits podem ser diferenciados comparando as contagens de interrupções. Esse método é particularmente eficaz porque permite que os bits sejam lidos independentemente da largura padronizada para a transmissão.

Para melhor compreensão é apresentada parte da rotina de interrupção do *timer*, responsável pela recepção de rádio:

```
if(!RX && rxdw < 255) rxdw++; //counts interruptions with the RX port on low level
if(RX && rxup < 255) rxup++; //counts interruptions with the RX port on high level
if(rxdw > 100) rfbits = 0; //pulse width is too long, code was interrupted
if(!RX && rxant) { //down slope -> bit end
        //requires a minimum pulse width of 900us, for noise suppression
        if((rxup + rxdw) < 20) rfbits = 0;
        rxbuff *= 2; //rolls the code left once
        rxbuff &= 0x1FFFFFFF; //clear 3 MSB bits
        if(rxup < rxdw) rxbuff |= 0x00000001; //  __|¯| = 1          _|¯¯| = 0
        rxup = rxdw = 0; //clears interruption counters
        if(++rfbits == 29) { //checks whether the code is complete
        //checks code integrity and return it, if it's ok
        if((rxbuff & 0xF000000F) == 0x10000005) rfCode = rxbuff;
                rxbuff = 0x00000000; //clears RF incoming buffer
                rfbits = 0; //reset bit counter
        }
}
rxant = RX; //stores the RX port state to be read on next interruption
```

3.3 *SOFTWARE* DO MÓDULO DE ACIONAMENTO

Funções do programa:

- Cadastrar um ou mais endereços de rádio na memória *Flash* através do botão de programação;
- Executar os comandos do relé de acordo com o protocolo estabelecido e sinalizar através do LED;
- Informar o *status* do dispositivo quando solicitado pela central;

A estrutura do programa pode ser vista na figura12.

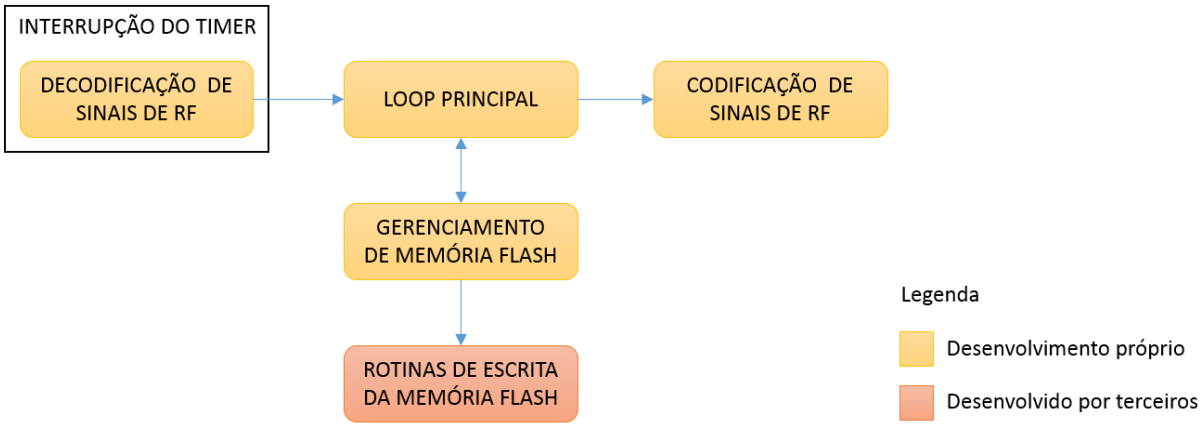


Figura 12 - Diagrama de blocos do software do módulo de acionamento.
Fonte: Autoria própria.

As ferramentas de desenvolvimento, a codificação e a decodificação de dados e o uso da memória *Flash* seguem o mesmo padrão adotado para a central de controle.

3.4 INTERFACE DO USUÁRIO

Estão presentes no aplicativo de interface as seguintes funcionalidades:

- Monitorar a presença da central de controle na porta USB;
- Incluir, renomear, remover e testar dispositivos de automação;
- Listar, criar, modificar e excluir tarefas agendadas;
- Atualizar data e hora do relógio da central de controle;
- Transferir as tarefas para a central de controle e consultar os *status* das mesmas.

O código do programa foi escrito em Java, utilizando-se o software NetBeans IDE 7.4 como ambiente de desenvolvimento e a JDK 1.7 como biblioteca padrão. O diagrama da figura 13 mostra a estrutura funcional do aplicativo.

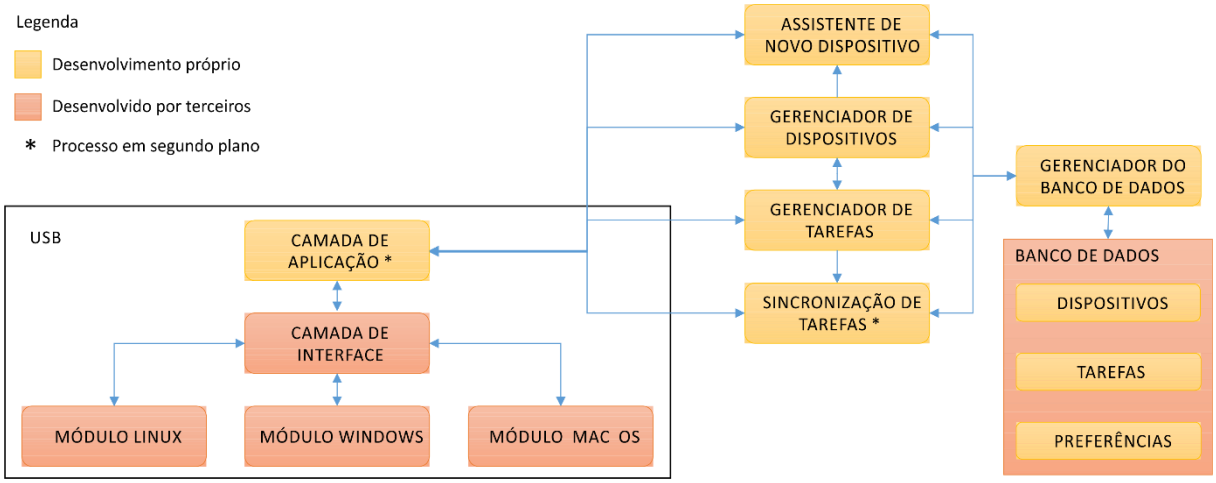


Figura 13 - Diagrama de blocos do aplicativo de interface com o usuário.
Fonte: Autoria própria.

Os blocos da figura 13 correspondem às principais classes do programa. Também foram utilizadas algumas classes auxiliares, como um *buffer* USB em nível de aplicação, com acesso controlado por semáforo, para impedir que duas classes acessem o mesmo dado simultaneamente. São necessárias ainda classes para definição de objetos do tipo “tarefa” e do tipo “dispositivo”, bem como classes que estendam a *DefaultTableModel* presente no *kit* de desenvolvimento Java, para que o usuário possa visualizar o conteúdo das tabelas.

As classes do programa são aqui descritas em sua totalidade:

- *Device*: armazena os dados de cada dispositivo que irá compor a lista de módulos de acionamento cadastrados pela central de controle;
- *DeviceTableModel*: subclasse da *DefaultTableModel*, criada para exibir a lista de módulos de acionamento cadastrados pela central de controle;
- *FrmDevices*: formulário onde o usuário pode gerenciar os módulos de acionamento;
- *FrmRegisterDevice*: formulário em formato de assistente, usado para cadastrar um novo módulo na central de controle;
- *FrmTask*: formulário onde o usuário pode selecionar todas as opções para criar uma nova tarefa ou alterar uma já existente;
- *FrmTaskList*: formulário principal do programa que contém o método *“main()”*. Nele é exibida a lista de tarefas e os principais comandos e informações do programa, como status e barra de progresso da conexão USB;

- *SQLite*: contém os métodos de acesso ao banco de dados, usados por todas as outras classes do programa;
- *SyncThread*: processo em segundo plano que prepara uma lista de ações as quais deverão ser executadas pela central, incluindo alterações na lista de tarefas e atualização de status das tarefas pendentes;
- *Task*: armazena os dados de cada tarefa presente no banco de dados do programa;
- *TaskTableModel*: subclasse da *DefaultTableModel*, criada para exibir a lista de tarefas criadas pelo usuário;
- *UsbBuffer*: conjunto de oito *bytes* com seus respectivos métodos de acesso, usado transferência via USB. Inclui um semáforo para impedir que diferentes *threads* o acessem simultaneamente;
- *UsbCommands*: lista de comandos USB idêntica ao *Enum* do software embarcado, usada para padronizar os *aliases* dos comandos em hexadecimal que correspondem ao protocolo de comunicação em anexo;
- *UsbHidApi*: processo em segundo plano faz a interface com pacote *hidapi-1.1.jar*, que implementa o driver HID. Esta classe responsável por monitorar a conexão USB e executar todos os comandos de leitura e escrita na USB solicitados pelas outras classes.

3.4.1 Processos em Segundo Plano

O uso de processos em segundo plano, neste projeto, têm por finalidade manter os formulários do programa ativos para interação com o usuário enquanto tarefas de comunicação são executadas. Por exemplo: enquanto dados de tarefas estão sendo transferidos para a central de controle, o usuário pode preencher os dados para uma nova tarefa.

3.4.2 Banco de Dados

Uma vez que o aplicativo permite ao usuário interagir com a lista de tarefas mesmo que a central esteja desconectada da porta USB, é conveniente que os dados gerados fiquem armazenados no disco rígido do computador. Assim, o usuário tem a opção de realizar os agendamentos *off-line*, e ficar *online* com a central apenas no momento em que desejar transmitir as tarefas.

Para proporcionar esta funcionalidade foi utilizado o SQLite3. Trata-se de um banco de dados bastante leve, multi-plataforma, que requer apenas a adição de um *driver* JDBC ao projeto Java.

3.4.3 Comunicação USB

Como a plataforma Java não tem suporte nativo ao USB, faz-se necessária a implementação de uma API independente para realizar tal acesso. Foi utilizada uma API Java desenvolvida pela empresa Codeminders, disponibilizada com licença BSD através do link <http://code.google.com/p/javahidapi/>. A API, que faz uso de módulos específicos para cada sistema operacional, utiliza a interface JNI para listar os dispositivos HID presentes no barramento e se conectar a um deles a partir das informações de *Vendor ID* e *Product ID*, que fazem parte dos descritores USB do dispositivo.

4 RESULTADOS

4.1 DESEMPENHO DO HARDWARE

4.1.1 Fonte de Alimentação

As tensões de alimentação dos circuitos mantiveram estabilidade suficiente para garantir a funcionalidade completa dos módulos, inclusive podendo utilizar ambas as tensões de rede: 127V e 220V. No entanto, ao alimentar o módulo de acionamento com 127V e após atracar o relé da placa, verificou-se queda na tensão de 12V para 10,6V. A mesma queda não acontece com a bobina do relé desligada. Os transformadores utilizados possuem corrente nominal de 200mA, e a carga aplicada aos mesmos é de aproximadamente 70mA, dos quais 50mA são destinados ao funcionamento do relé. Logo, conclui-se que essa queda na tensão 12V se deve ao baixo desempenho dos transformadores lineares adquiridos para a montagem dos protótipos.

A variação observada se limita apenas à tensão de 12V, não interferindo na linha de 5V. Com isso, ficam garantidas as funções do módulo embora, em tese, possa haver uma pequena diminuição no alcance dos sinais de rádio transmitidos pelo módulo, uma vez que o circuito transmissor é alimentado com 12V.

As figuras 14 e 15 representam as medições feitas com osciloscópio da tensão de 12V em diferentes situações.

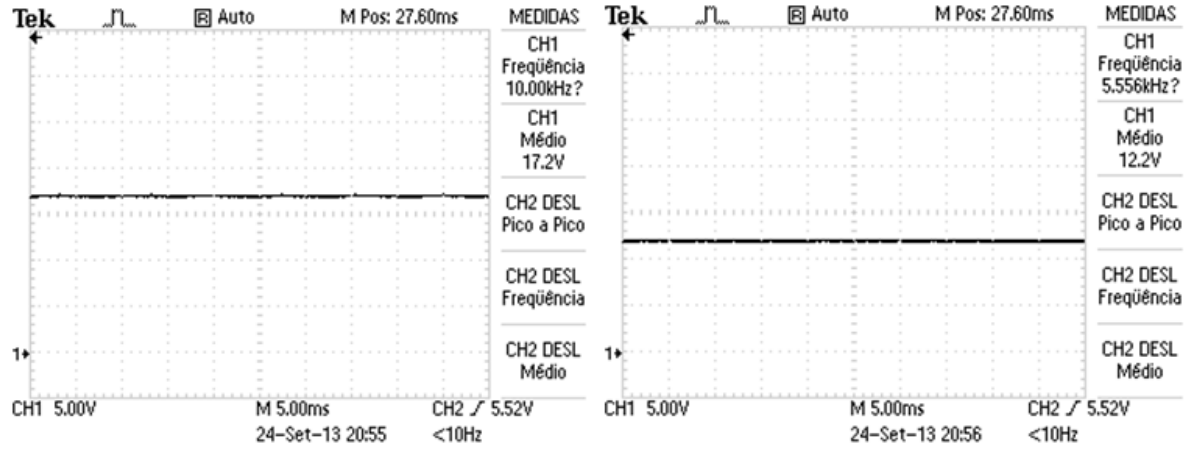


Figura 14 - Tensão não regulada (à esquerda) e regulada (à direita) com alimentação de 127V e relé não acionado.

Fonte: Autoria própria.

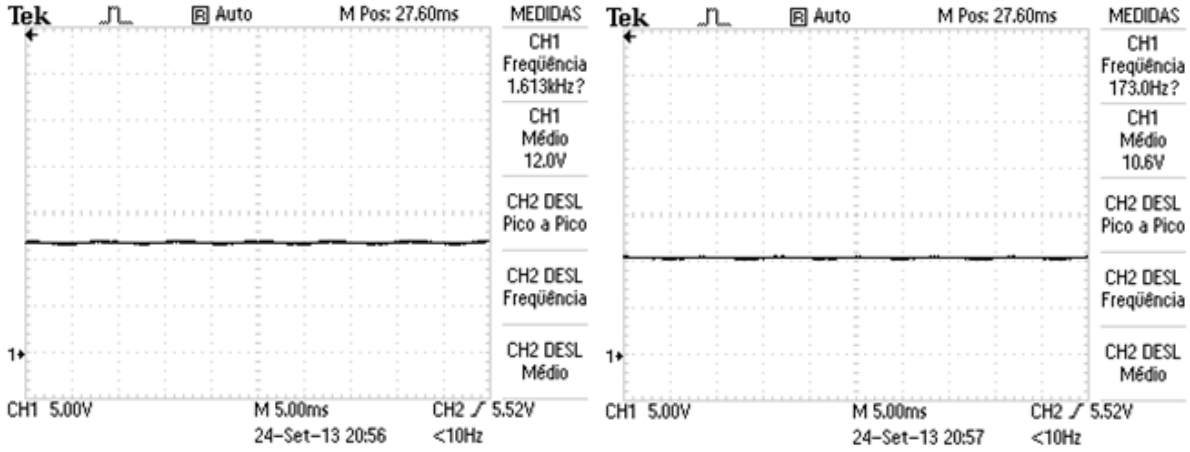


Figura 15 - Tensão não regulada (à esquerda) e regulada (à direita) com alimentação de 127V e relé acionado.

Fonte: Autoria própria.

4.1.2 Comunicação de Rádio Frequência

Os circuitos de rádio frequência têm razoável capacidade de comunicação desde que não haja interferências no canal de RF e que seja respeitada uma distância máxima de 25 metros em ambiente *indoor*. A figura 16 representa uma captura de código enviado por um dos módulos de acionamento e recebido pela central de controle.

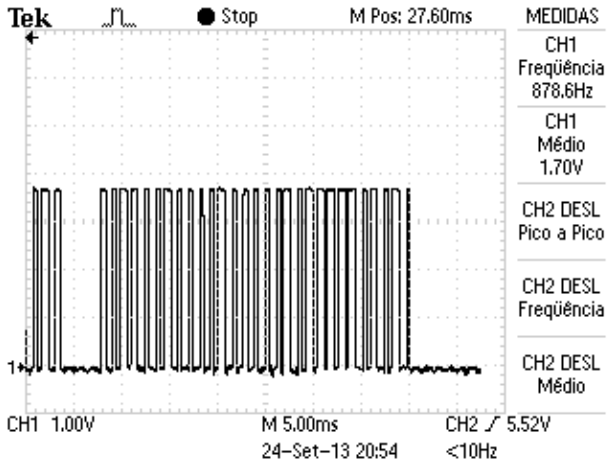


Figura 16 – Transmissão de rádio frequência capturada no circuito receptor.
Fonte: Autoria Própria.

Observou-se que as larguras de pulso de mesmo valor lógico identificadas na recepção de rádio pelo software da central de controle variam dentro de um mesmo código, mesmo que o *clock* da transmissão seja constante. Isso se deve ao fato de que as interrupções da porta USB têm prioridade sobre as interrupções do *timer*, fazendo com que a periodicidade do *timer* não seja constante. Mesmo assim, o software é capaz de decodificar o sinal corretamente, uma vez que não depende de sincronismo de *clock* entre transmissor e receptor.

Ao se utilizar o padrão de codificação do HT6P20, que é amplamente empregado comercialmente, permitiu-se que os módulos de acionamento pudessem ser comandados diretamente por controles remotos comuns, disponíveis no mercado. Portanto, o método de recepção assíncrona desenvolvido para a central de controle também é útil para o módulo de acionamento, pois permite a compatibilidade com controles remotos de diferentes fabricantes que usam diferentes larguras de pulso na transmissão dos dados.

4.1.3 Placa de circuito impresso

A placa de circuito atendeu bem aos requisitos do projeto, inclusive quanto ao *layout* híbrido que comporta ambos os circuitos do módulo central e do módulo auxiliar. Houve, entretanto, um defeito de corrosão em uma das unidades, embora

não tenha sido necessário rejeitá-la, uma vez que o defeito se limitava apenas ao circuito da porta USB, sendo possível utilizar a placa para a montagem de um módulo de acionamento.

Ao fixar as placas nos respectivos gabinetes, observou-se também que as dimensões das mesmas ficaram ligeiramente acima do especificado no projeto, sendo necessário o desbaste de alguns pontos para possibilitar o encaixe no gabinete.

As fotos 17 e 18 ilustram uma das placas confeccionadas.

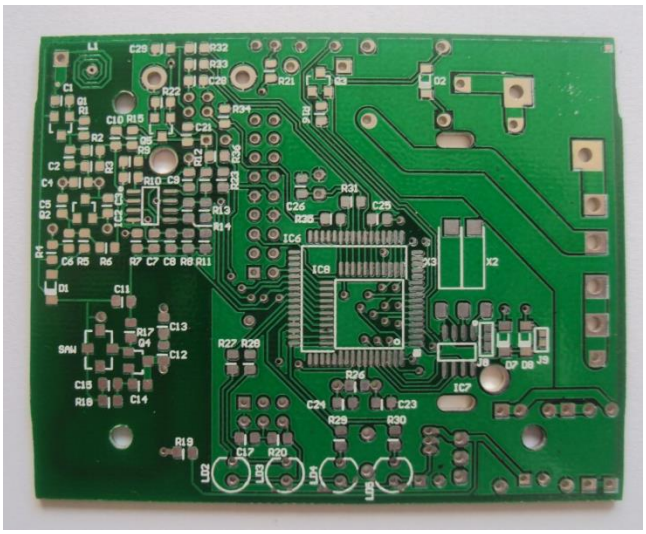


Figura 17 - Placa de Circuito Impresso (face superior).
Fonte: Autoria própria.

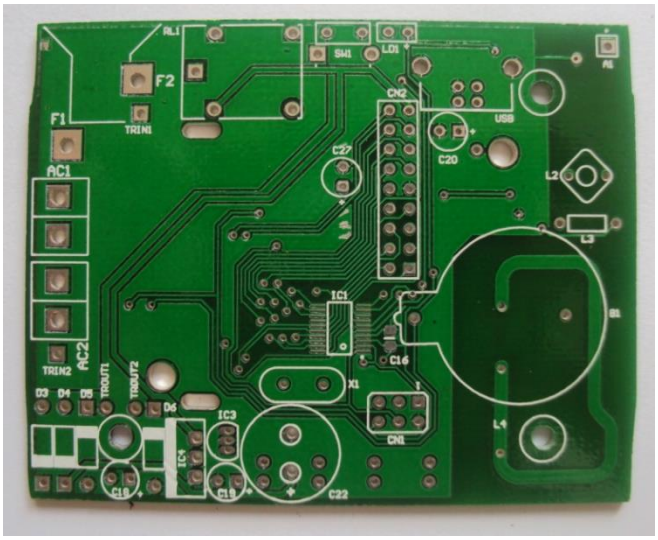


Figura 18 - Placa de Circuito Impresso (face inferior).
Fonte: Autoria própria.

4.1.4 Montagem dos módulos

A montagem física dos módulos de hardware não apresentou grandes surpresas. No entanto, verificou-se que para uma eventual produção em série seria imprescindível o desenvolvimento de ferramentas mecânicas para tornar mais rápidas as diversas furações necessárias ao encaixe de teclas, LEDs, fios e conectores. Os resultados obtidos com a montagem dos módulos são ilustrados nas fotos 19 a 24.

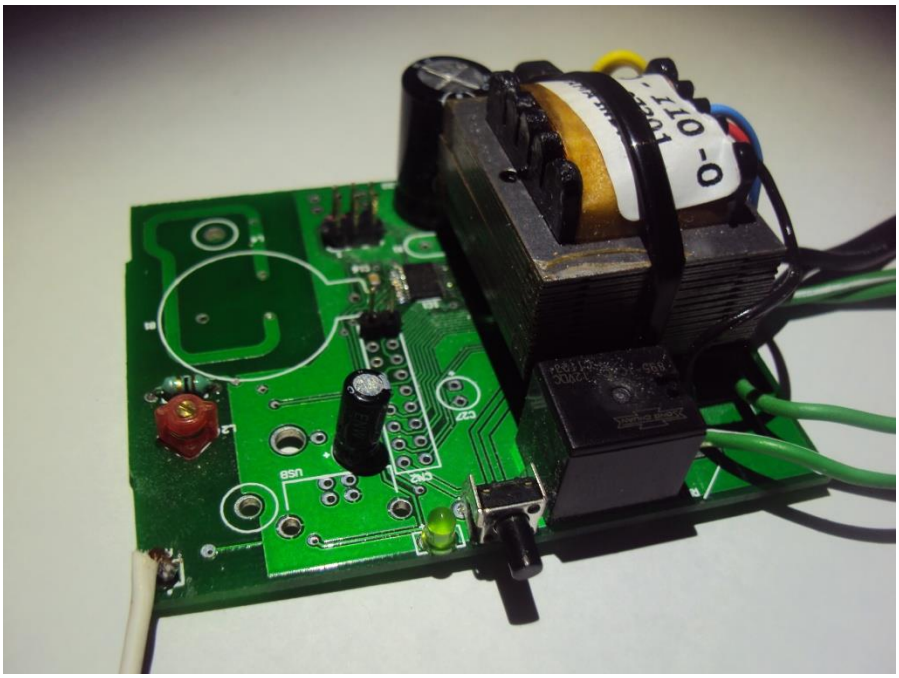


Figura 19 - Placa do módulo auxiliar.
Fonte: Autoria própria.

Figura 20 - Placa do módulo central montada em seu compartimento final.
Fonte: Autoria própria.

Figura 21 – Conjunto Módulo Auxiliar.
Fonte: Autoria própria.

Figura 22 - Módulo Auxiliar com foco no botão de programação.
Fonte: Autoria própria.

Figura 23 - Conjunto Módulo Central
Fonte: Autoria própria.

Figura 24 - Módulo Central com foco no conector USB
Fonte: Autoria própria.

4.2 *SOFTWARE* EMBARCADO

4.2.1 *Driver* HID

A implementação dos códigos para acesso ao driver HID nativo do sistema operacional foi bem sucedida para realizar a conexão, tanto do lado do *device* (código em C) quanto do lado do *host* (código em Java). As bibliotecas prontas utilizadas não tiveram problemas para estabelecer uma conexão usando a porta USB, muito embora ainda permaneçam algumas instabilidades momentâneas nessa conexão. Já no quesito transferência de dados, o driver HID se mostrou pouco eficiente para grandes volumes de informação, uma vez que apenas 8 bytes podem ser transferidos em cada pacote. Isso não impede a funcionalidade do sistema, mas torna um pouco demorado o processo de sincronização quando há várias tarefas para serem transmitidas.

4.2.2 Memória *Flash*

A rotina escrita em Assembly, fornecida pela Freescale, para gravação de dados em memória *Flash*, tem certas limitações não documentadas pela empresa. Isso que exigiu um processo de *debug* relativamente trabalhoso até que se essas limitações pudessem ser contornadas pelo uso de *delays* e reestruturação em partes específicas do código. Após essas correções não houve mais nenhum tipo de falha na manipulação da memória *Flash*.

4.2.3 Controle de Agendamento

A varredura cíclica de toda a memória *Flash* disponível para dados, combinada com a verificação de data e hora em uma CPU de 8 bits, é uma tarefa um tanto extensa para o microcontrolador, sobretudo porque o acesso à memória *Flash* é limitado em 200kHz, independentemente do *clock* utilizado para o processamento. Com isso, observou-se que quando há muitas tarefas agendadas, pode haver atrasos superiores a um minuto na execução de algumas tarefas, o que pode ocasionar o cancelamento da tarefa se esta não permitir atrasos. Outro fator que pode causar atrasos é a demora na comunicação de rádio devido a interferências externas ou módulos fora da área de alcance da central USB.

4.2.4 Memória RAM

Enquanto a central USB usa um microcontrolador com 4kB de memória RAM, os módulos de acionamento recebem apenas 256 bytes desse recurso. Com isso, algumas funções que poderiam ser exatamente iguais nos dois tipos de módulo, como a transmissão de rádio, tiveram que ser reformuladas para racionar o uso da memória RAM, sob pena de extrapolar o limite de memória pelo simples emprego de estruturas de laço. Esse tipo de falha é particularmente difícil de ser

detectada já que não é identificado nem pelo compilador nem pelo software de *debug*.

Ficou claro que os módulos de acionamento dificilmente poderão receber novas funcionalidades se estiverem equipados com o mesmo microcontrolador. Felizmente o MC9S08SH4 também é fabricado em versões com até 1kB de memória RAM.

4.3 INTERFACE DO USUÁRIO

4.3.1 SQLite

Por se tratar de uma aplicação simples do ponto de vista da persistência de dados no *host*, sem muitas tabelas e sem relacionamentos complexos, o SQLite se encaixa perfeitamente à necessidade em questão, e foi uma opção muito prática para o comissionamento da interface, visto que seu *driver* JDBC é bem funcional e de fácil utilização.

4.3.2 NetBeans IDE

O uso do *software* NetBeans para programação foi de grande utilidade visto que a automatização na elaboração de interfaces gráficas garante uma economia de tempo significativa, especialmente quando não se tem domínio completo da linguagem Java. As figuras 25 a 27 representam as principais telas do *software* de interface.

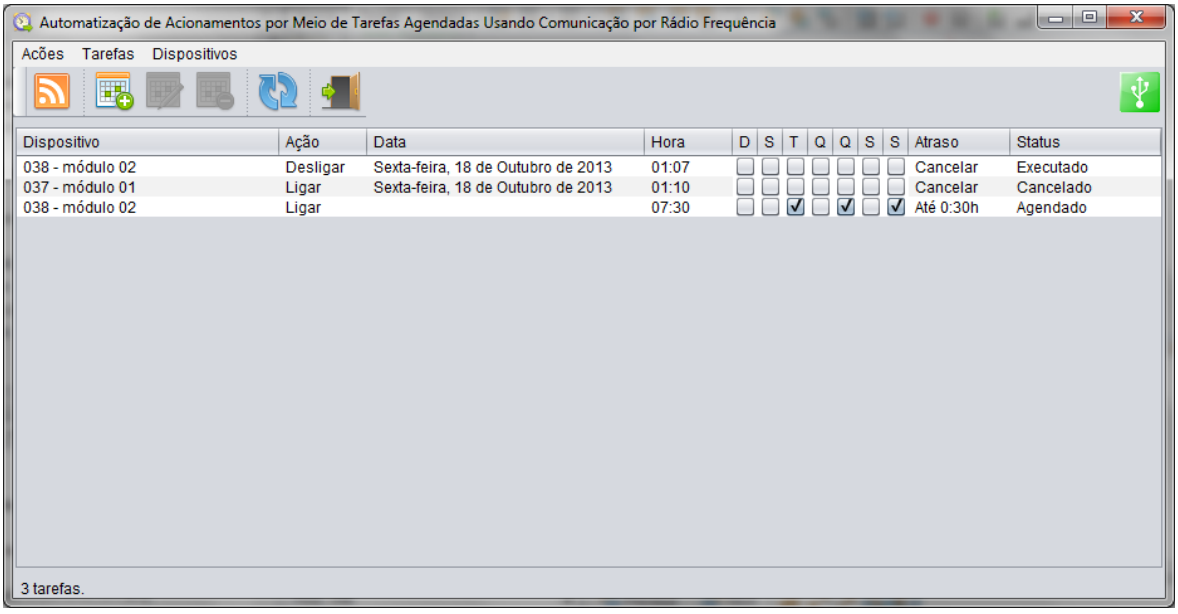


Figura 25 - Tela principal do aplicativo JAVA.
Fonte: Autoria própria.

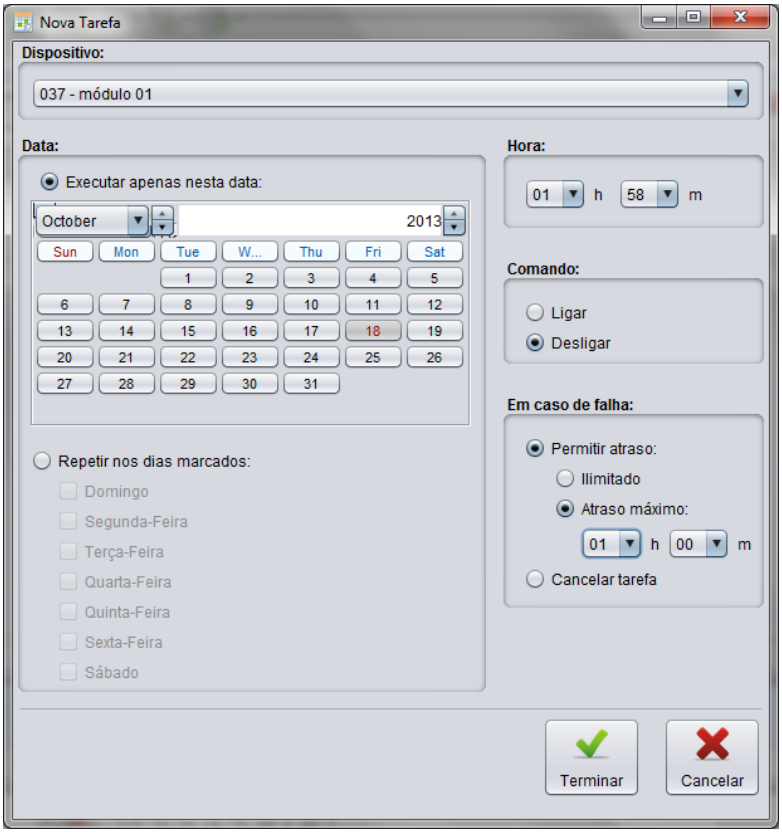


Figura 26 - Tela de preenchimento de tarefas.
Fonte: Autoria própria.

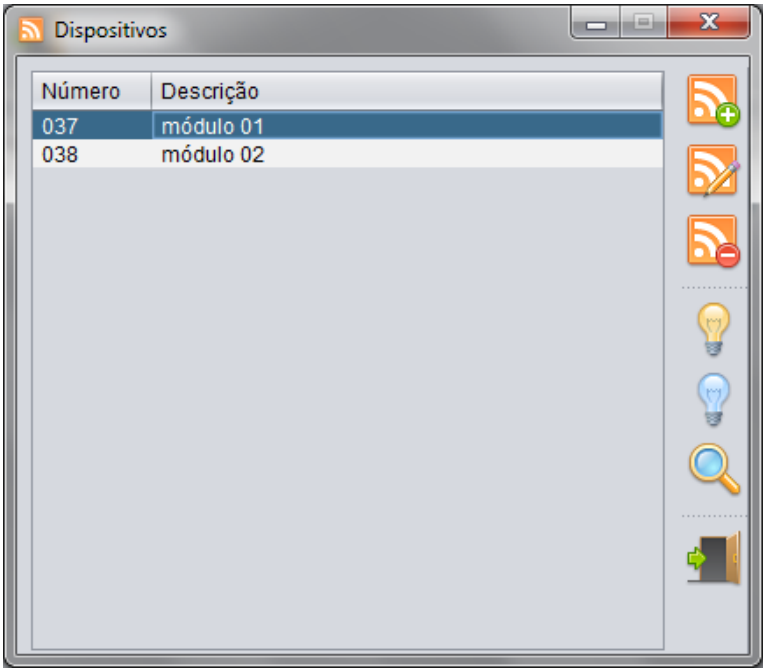


Figura 27 - Tela de Gerenciamento de Dispositivos.
Fonte: Autoria própria.

4.3.3 Processos em Segundo Plano

O uso de *Threads* separadas para executar processos mais demorados, como transferência de dados para a central USB e interação direta com os módulos de acionamento, é muito conveniente, pois permite que o usuário continue interagindo com a interface mesmo enquanto esses processos estão em andamento. Sem esse recurso, usar o aplicativo seria muito desconfortável: o usuário teria a impressão de travamento do programa sempre que um desses processos estivesse em execução.

4.3.4 *DefaultTableModel*

A *DefaultTableModel*, superclasse Java destinada a permitir a manipulação de tabelas em formulários, é conhecida por apresentar falhas diversas. A alternativa mais correta seria herdar a classe de acesso à tabela diretamente da *TableModel*,

superclasse da qual é derivada a própria *DefaultTableModel*. No desenvolvimento deste aplicativo, no entanto, não se verificou falhas na *DefaultTableModel,* e sua utilização também contribuiu para economia de tempo de programação.

5 CONSIDERAÇÔES FINAIS

Dado o grau de funcionalidade atingido pelo protótipo, percebe-se que este projeto cumpriu todas as metas propostas sendo viável, inclusive, para produção em maior quantidade. Pode-se afirmar que a presente pesquisa fornece base tecnológica suficiente para fabricar e comercializar um produto com as funcionalidades do protótipo apresentado.

De acordo com Pradella (2013), o sistemas de domótica mais acessíveis do mercado são os controles remotos universais, destinados ao comando de múltiplos equipamentos, e que custam entre R$ 400,00 e R$ 800,00. Esses aparelhos conferem praticidade e conforto ao usuário quando este se encontra na residência. No entanto, quando o usuário está ausente, apenas soluções mais caras, a partir de R$ 4.000,00, são capazes de executar acionamentos, seja por celular ou por ações pré-programadas.

Os protótipos desenvolvidos a partir desta pesquisa tornam a automação por tarefas agendadas mais atrativa pois, se fabricados em série, teriam custo unitário inferior a R$ 30,00, significando que o custo de desenvolvimento poderia ser absorvido rapidamente ao mesmo tempo que se poderia oferecer a solução a um preço bem acessível.

Não obstante, o sistema possui limitações que ainda o impedem de ser classificado como um produto de consumo. Por exemplo, uma vez que os módulos de automação são instalados, fica estabelecido um vínculo entre os dados da central USB e o banco de dados da máquina *host*. O sistema, em sua forma atual, não prevê que uma mesma central USB possa ser acessada por diferentes computadores sem prejuízo da configuração existente. O sistema também não está adequadamente preparado para operar com mais de uma central USB no mesmo computador. Estas correções, no entanto, exigem apenas tempo de desenvolvimento, e não correm o risco de aumentar o custo de produção ou inviabilizar o projeto tecnicamente.

Existem ainda questões mercadológicas que mereceriam ser analisadas antes que um produto com as características deste projeto pudesse ser lançado. Um exemplo disso seria o uso de frequências e modulações diferentes para a comunicação wireless, o que provavelmente impactaria o custo da solução, mas

também melhoraria o desempenho do sistema. Por exemplo, um sistema operando em 2,4GHz poderia incrementar a área de cobertura do sinal de rádio e aumentar a velocidade da comunicação. Outra possibilidade seria a implementação de um módulo GSM em substituição (ou complementação) à porta USB, associada ao desenvolvimento de um aplicativo para telefones celulares, possibilitando a operação remota do sistema. É evidente que tal modificação elevaria muito o custo do *hardware*, razão pela qual não foi escolhida para este projeto.

Estas e outras melhorias poderiam ser facilmente incorporadas ao projeto sem a necessidade de reestruturação do hardware. Isso implica dizer que uma das virtudes deste projeto é que, embora esteja longe de ser uma solução definitiva, o mesmo constitui base tecnológica suficientemente desenvolvida para a criação de diversas outras soluções, bastando para isso que se tracem novos objetivos e que se invista no potencial desta tecnologia.

REFERÊNCIAS

AYRE, Andrew; PFEIFFER, Olaf. **Using Flash Memory in Embedded Applications.** 2010. Disponível em: <http://www.esacademy.com/en/library/technical-articles-and-documents/8051-programming/using-flash-memory-in-embedded-applications.html>. Acesso em: 16 jun. 2013.

BAPTISTA, Leandro; PALANGANI, Marlon; DEOLTODO, Tiago. **Java Virtual Machine (JVM).** Abril/2009. Disponível em:<adscesumar2008.files.wordpress.com/2009/04/java-virtual-machine.pdf>. Acesso em: 10 mai. 2013.

BASTOS, Bruno; CAPRONI, Rodolfo. **Controlando o Microcontrolador JM60 via USB com LabView.** Revista Saber Eletrônica. V45. Nº438. P.18-24.

CESTA, André A. **Tutorial: A Linguagem de Programação Java e Orientação a Objetos.** Instituto de Computação da UNICAMP. 2009.

FIALHO, Vitor Manuel de Oliveira. **Apontamentos Sobre Modulações Digitais.** Instituto Superior de Engenharia de Lisboa. Maio/2011. Disponível em: <http://www.deetc.isel.ipl.pt/sistemastele/cm/Bibliografia/MaterialDeApoio/ModulacoesDigitais.pdf>. Acesso em: 10 jul. 2013.

FONTES, Aluisio I. R. **Classificação Automática de Modulação Digital com uso de Correntropia para Ambientes de Rádio Cognitivo.** Mestrado. UFRN. Natal. 2012.

FRIESEN, Jeff. **Java and USB.** Julho/2006. Disponível em <https://today.java.net/pub/a/today/2006/07/06/java-and-usb.html>. Acesso em: 20 mai. 2013.

INSAM, Eddie. **A Simple USB Oscilloscope for the PC – Part I.** Revista Eletronics World. Fevereiro/Março 2002.

JIANG, Qingye. **Access USB Devices from Java Applications.** An Introduction to USB, jUSB and JSR-80. Setembro/2003. Disponível em <http://www.ibm.com/developerworks/linux/library/j-usb/index.html>. Acesso em: 25 mai. 2013.

JIANG, William. **USB and Using the CMX USB Stack with 9S08JM Devices.** Freescale Semiconductor. Documento: AN3565. Fevereiro/2008.

LASKOSKI, Gustavo; MARCONDES, Maicon; SZEREMETA, Oscar. **Modulações Digitais**. Tópicos de Comunicações. UTFPR. Curitiba. 2006.

MOTA, Diego Maciel. **Utilização de Comunicação USB em Sistemas Embutidos.** 2007. 25 f. Monografia – Universidade Federal de Ouro Preto. 2007. Disponível em: <http://www.em.ufop.br/cecau/monografias/2007/DIEGO%20MACIEL%20MOTA.pdf>. Acesso em: 10 ago. 2013.

MULLER, Henk. **How to Create and Program USB Devices.** Julho/2012. Disponível em <http://electronicdesign.com/boards/how-create-and-program-usb-devices>. Acesso em: 18 jun. 2013.

MURATORI, José Roberto. **Integrador de Sistemas Residenciais: um Novo Profissional.** Março/2005. Disponível em: <http://aureside.com.br/artigos/default.asp?file=01.asp&id=64>. Acesso em 25 set. 2013.

MURATORI, José Roberto; DAL BÓ, Paulo Henrique. **Automação Residencial: Histórico, Definições e Conceitos.** 2011. Disponível em:

<http://www.osetoreletrico.com.br/web/documentos/fasciculos/Ed62_fasc_automacao_capI.pdf>. Acesso em: 22 ago. 2013.

OLIVEIRA, Adriano Marcio de. **Automação Residencial.** 2005. 44 f. Monografia – Centro Universitário de Araraquara – UNIARA, 2005. Disponível em: <http:// www.aureside.org.br//temastec/TCC_AutoRes10.pdf>. Acesso em: 20 set. 2013.

PRADELLA, Arthur. **Automação Residencial que Cabe no Bolso.** 2013. Disponível em: <http://revistahometheater.uol.com.br/site/tec_artigos_02.php?id_lista_txt=8367>. Acesso em: 02 dez. 2013.

SILICON LABS. **Human Interface Device Tutorial.** Documento AN249. Março/2011. Disponível em: <www.silabs.com/Support%20Documents/TechnicalDocs/AN249.pdf>. Acesso em: 10 jul. 2013.

USB-Implementers Forum. **Universal Serial Bus Specification Revision 2.0**. Abril/2000.

ANEXO(S)

SISTEMA DE AUTOMAÇÃO RESIDENCIAL

PROTOCOLO DE COMUNICAÇÃO ENTRE APLICATIVO DE INTERFACE E MÓDULO CENTRAL

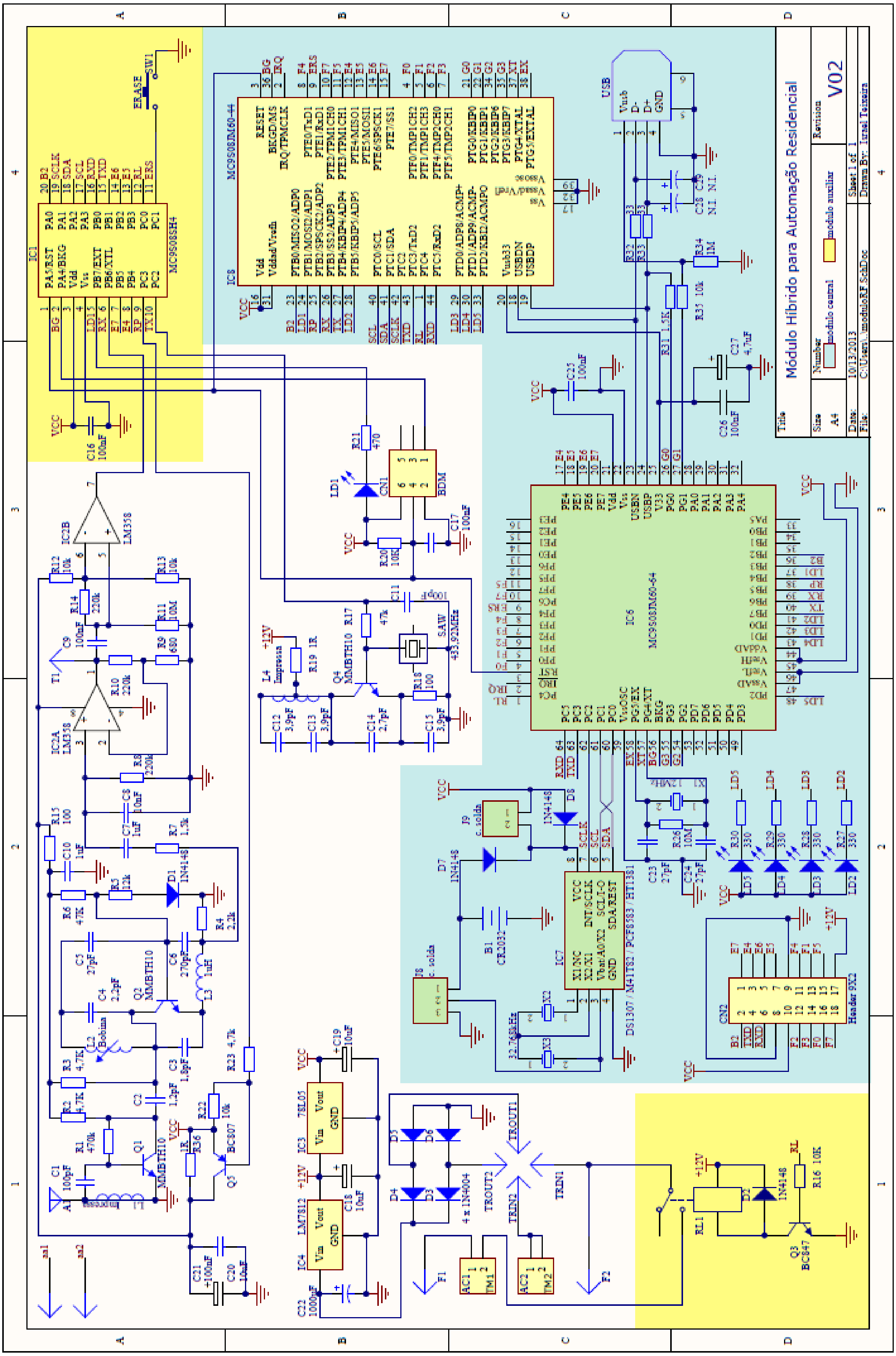
	COMANDO (host -> device)							
	byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
SET_CALENDAR	0x01	10s | 1s	10m | 1m	10h | 1h	10d | 1d	10M | 1M	dia_sem	10a | 1a
GET_CALENDAR	0x02	X	X	X	X	X	X	X
ADD_DEVICE	0x03	X	X	X	X	X	X	X
REG_DEVICE	0x04	código RF			X	X	X	X
REMOVE_DEVICE	0x05	disp	X	X	X	X	X	X
GET_DEVICE_STATUS	0x06	disp	X	X	X	X	X	X
ACTIVATE_DEVICE	0x07	disp	X	X	X	X	X	X
DEACTIVATE_DEVICE	0x08	disp	X	X	X	X	X	X
ADD_TASK_1	0x09	disp	valor	ano	mês	dia	hora	minuto
ADD_TASK_2	0x0A	seg	semanal	atrasoH	atrasoM	X	X	X
REMOVE_TASK	0x0B	pg	pos	X	X	X	X	X
GET_TASK_STATUS	0x0C	pg	pos	X	X	X	X	X
NULL_ACTION	0xFF	X	X	X	X	X	X	X

	RESPOSTA (device -> host)							
	byte 0	byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7
SET_CALENDAR	0x01	X	X	X	X	X	X	X
GET_CALENDAR	0x02	10s | 1s	10m | 1m	10h | 1h	10d | 1d	10M | 1M	dia_sem	10a | 1a
ADD_DEVICE	0x03	disp	código RF			X	X	X
REG_DEVICE	0x04	disp	X	X	X	X	X	X
REMOVE_DEVICE	0x05	X	X	X	X	X	X	X
GET_DEVICE_STATUS	0x06	status	X	X	X	X	X	X
ACTIVATE_DEVICE	0x07	X	X	X	X	X	X	X
DEACTIVATE_DEVICE	0x08	X	X	X	X	X	X	X
ADD_TASK_1	0x09	X	X	X	X	X	X	X
ADD_TASK_2	0x0A	pg	pos	endereço		X	X	X
REMOVE_TASK	0x0B	pg	pos	X	X	X	X	X
GET_TASK_STATUS	0x0C	status	X	X	X	X	X	X
NULL_ACTION	0xFF	X	X	X	X	X	X	X

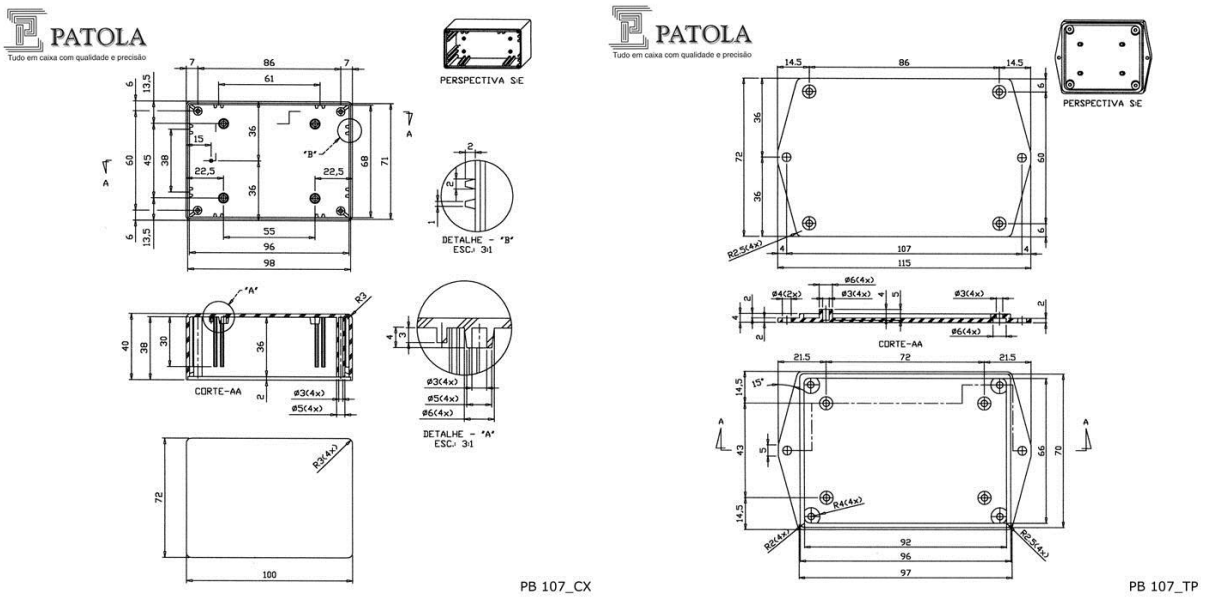
SISTEMA DE AUTOMAÇÃO RESIDENCIAL

PROTOCOLO DE COMUNICAÇÃO ENTRE APLICATIVO DE INTERFACE E MÓDULO CENTRAL

Legenda		
Campo	**Descrição**	**Escopo de valores**
10s	dezenas de segundos	0 ~ 5
1s	unidades de segundos	0 ~ 9
10m	dezenas de minutos	0 ~ 5
1m	unidades de minutos	0 ~ 9
10h	dezenas de horas	0 ~ 2
1h	unidades de horas	0 ~ 9
10d	dezenas de dias	0 ~ 3
1d	unidades de dias	0 ~ 9
10M	dezenas de meses	0 ~ 1
1M	unidades de meses	0 ~ 9
dia_sem	número do dia da semana	1 ~ 7
10a	dezenas de anos	0 ~ 9
1a	unidades de anos	0 ~ 9
código RF	três bytes para endereço de RF	3 x 1 ~ 255
disp	número do dispositivo	1 ~ 170
valor	informa se a ação é ligar ou desligar	0 = desligar | 1 = ligar
ano	ano agendado para a tarefa	0 ~ 99
mês	mês agendado para a tarefa	1 ~ 12
dia	dia agendado para a tarefa	1 ~ 31
hora	hora agendado para a tarefa	0 ~ 23
minuto	minuto agendado para a tarefa	0 ~ 59
segundo	segundo agendado para a tarefa	0 ~ 59
semanal	bit 7 : executar a tarefa aos domingos	0 = não | 1 = sim
	bit 6 : executar a tarefa às segundas-feiras	0 = não | 1 = sim
	bit 5 : executar a tarefa às terças-feiras	0 = não | 1 = sim
	bit 4 : executar a tarefa às quartas-feiras	0 = não | 1 = sim
	bit 3 : executar a tarefa às quintas-feiras	0 = não | 1 = sim
	bit 2 : executar a tarefa às sextas-feiras	0 = não | 1 = sim
	bit 1 : executar a tarefa aos sábados	0 = não | 1 = sim
	bit 0 : flag de tarefa executada	0 = cancelada | 1 = executada
atrasoH	horas de atraso permitido para a tarefa	0 ~ 72
atrasoM	minutos de atraso permitido para a tarefa	0 ~ 59
pg	número da página que contém a tarefa	1 ~ 87
pos	posição relativa da tarefa dentro da página	0 ~ 45
endereço	endereço físico do 1º byte da tarefa	0x4801 ~ 0xFBFA

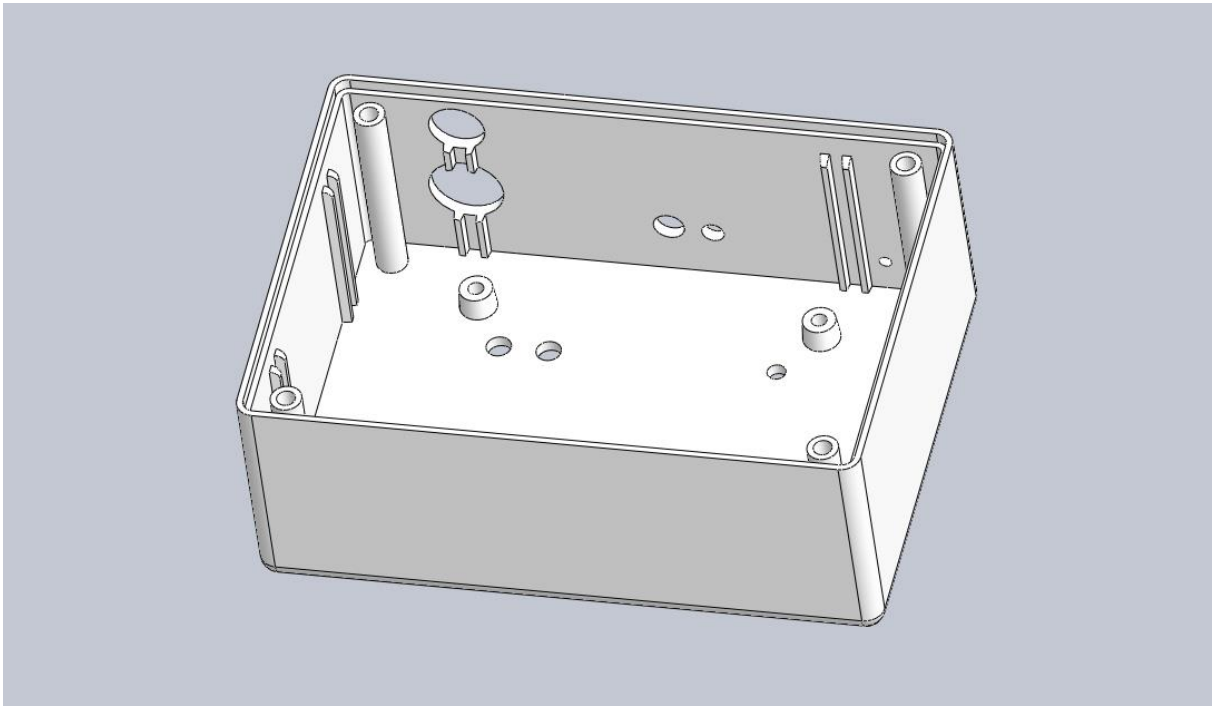


Módulo Híbrido para Automação Residencial
V02
A4
10/13/2013
Sheet 1 of 1
Drawn By: Israel Teixeira



Desenhos cotados da caixa PB-107 da Patola

Fonte: <http://www.patola.com.br>



Caixa PB-107 da Patola reproduzida em SolidWorks