BRUNO IGOR RODRIGUES DOMINGUES
BRUNO NIGRO
FERNANDO NOBRE

SISTEMA VEICULAR DE CONTROLE DE VELOCIDADE E INFORMAÇÕES RODOVIÁRIAS

São Paulo
2010

BRUNO IGOR RODRIGUES DOMINGUES
BRUNO NIGRO
FERNANDO NOBRE

SISTEMA VEICULAR DE CONTROLE DE VELOCIDADE E INFORMAÇÕES RODOVIÁRIAS

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção da Graduação em Engenharia de Computação

São Paulo
2010

BRUNO IGOR RODRIGUES DOMINGUES
BRUNO NIGRO
FERNANDO NOBRE

SISTEMA VEICULAR DE CONTROLE DE VELOCIDADE E INFORMAÇÕES RODOVIÁRIAS

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção da Graduação em Engenharia de Computação

Área de concentração:
Engenharia da Computação

Orientador: Prof. Dr.
Marco Túlio Carvalho de Andrade

São Paulo
2010

Sistema veicular de controle de velocidade e informações rodoviárias / B.I.R. Domingues; B. Nigro; F. Nobre. -- São Paulo, 2010. 109 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1. Segurança rodoviária 2. Velocidade (Controle) I. Nigro, Bruno II. Nobre, Fernando III.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sis-temas Digitais II. t.

AGRADECIMENTOS

Ao professor Marco Túlio Carvalho de Andrade, pela orientação e pelo constante estímulo transmitido durante todo o trabalho.

Ao Sergio Aparecido de Oliveira do laboratório de microprocessadores e ao Daniel do laboratório digital, que, com muita paciência, nos ajudaram muito.

RESUMO

Este trabalho apresenta um sistema de automação rodoviário que visa aumentar a segurança das rodovias, auxiliando os motoristas a conduzirem seus veículos de maneira segura através da transmissão de informações sobre as condições e os limites de velocidade dos trechos das rodovias, bem como monitorar a velocidade dos veículos e avisar os motoristas quando sua velocidade exceder o limite permitido. É apresentada a especificação do sistema, destacando-se os requisitos funcionais e não-funcionais, além das restrições e suposições consideradas.
Foi desenvolvida uma arquitetura para o sistema em níveis geral e modular, especificando-se os principais componentes necessários à sua implementação. Baseado nessa arquitetura, foi construído um protótipo do sistema para demonstrar sua viabilidade. Neste trabalho, são apresentados os componentes de *hardware* e os protocolos de comunicação utilizados na implementação do protótipo, bem como e os *softwares* desenvolvidos para essa finalidade. Por fim, foram feitas algumas conclusões e sugestões de trabalhos futuros com o objetivo de aperfeiçoar o sistema.

Palavras-chave: automação rodoviária, segurança no trânsito, transmissão de dados, GSM, rádio-freqüência.

ABSTRACT

This project envolves a highway automation system that aims to increase highway security, helping drivers conduct their vehicles in a safe manner. This is accomplished through the transmission of helpful information, such as road conditions and speed limits, as well as a monitoring function that warns the driver when he exceeds the speed limit. The system specification is presented, with special emphasis on functional and non-functional requisites, along with the restrictions and pre-requisites. The system architecture was developed, both considering a general scope, evolving all system elements, and on a module-specific scope, with all the essential components specified. Based on this architecture, a prototype was constructed to demonstrate the projects viability. The hardware components and the communication protocols used in the prototypes implementation are presented, as well as the software that was developed. A few final considerations, including the projects outcome and suggestions for future developments are included.

Keywords: automation, highway, security, traffic, data transmission, GSM, radio frequency.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GPRS	*General Packet Radio Service*
GSM	*Global System for Mobile Communications*
MR	Módulo rodoviário
MV	Módulo veicular
RF	Rádio-freqüência
SIM	*Subscriber Identity Module*
SMS	*Short Message Service*
UCS2	2-byte Universal Character Set
ASCII	American Standard Code for Information Interchange
BBF	Protocolo de Comunicação RF (Bruno Bruno Fernando)

SUMÁRIO

1 Introdução

Neste capítulo será introduzido o trabalho de monografia para conclusão de curso no curso de engenharia de computação da escola Politécnica.

1.1 Apresentação do tema

Os sistemas computacionais estão sendo cada vez mais empregados em diversas áreas de atividades do ser humano, através da melhoria de processos e sistemas já existentes ou criação de novos sistemas, com o objetivo de proporcionar diversos benefícios às pessoas. O presente trabalho é uma sugestão de aplicação de tais sistemas para a automação rodoviária. Atualmente, a tecnologia aplicada nas rodovias está voltada para o controle de velocidade dos veículos através de um caráter punitivo. Os radares visam limitar a velocidade dos veículos em uma rodovia através da detecção daqueles que estão acima da velocidade máxima permitida e aplicação de multas. Porém, a tecnologia não é aplicada a favor do motorista, alertando-o quando ele ultrapassar esse limite de velocidade.

Esse trabalho apresenta o "Sistema veicular de controle de velocidade e informações rodoviárias". O sistema proposto visa transmitir aos motoristas dos veículos informações dinâmicas sobre o trecho da rodovia em que está trafegando. Essas informações incluem a velocidade máxima permitida no trecho (pois ela pode variar em diferentes trechos da mesma rodovia), buracos na pista, acidentes, obras e condições do trânsito e do tempo. Tendo essas informações, o motorista consegue se preparar melhor para as adversidades que encontrará a sua frente, prevenindo-se e evitando a ocorrência de acidentes.

1.2 Motivação

Antes da década de 1970, a maioria dos veículos produzidos no Brasil possuía apenas o espelho retrovisor no lado do motorista. Naquela época, percebeu-se que colocando um espelho também no lado do passageiro o número de colisões diminuía. Os motoristas demoraram a se acostumar com o novo espelho, mas após o período de adaptação foi bem aceito e exigido por todos. Na mesma década, por questão de estética, os bancos não possuíam apoio de cabeça. Mas percebeu-se que em colisões, muitos ferimentos graves aconteciam devido ao "efeito chicote", ou seja, sem proteção as cabeças dos passageiros faziam movimentos bruscos para trás e para frente podendo causar a quebra do pescoço. Além dos dois itens de segurança incluídos nos carros daquela época, desde a invenção do automóvel, a cada ano mais e mais itens de segurança são incluídos nos carros. *Air bags*, freios ABS, controle de tração, entre outros itens de segurança, cada vez mais fazem parte dos carros dos brasileiros.

Atualmente, a tecnologia digital está se desenvolvendo em um ritmo muito acelerado e sendo aplicada em diversas áreas de atividade do ser humano. Porém, nas rodovias, ela ainda é pouco usada para aumentar a segurança dos veículos que nela trafegam. Há poucos dispositivos que limitam a velocidade do motorista. Os mais utilizados são os radares de velocidade, uma forma que o Estado encontrou para diminuir as colisões ou atropelamentos em alta velocidade. Esses dispositivos visam detectar os veículos que estão trafegando a uma velocidade superior a velocidade máxima permitida e aplicar-lhes multas como forma de punição. Esse controle punitivo é desagradável ao motorista, uma vez que ele tem a sensação de que esse controle está sendo usado contra ele e não ao seu favor (para aumentar a segurança de sua viagem). Além disso, ele é ineficaz em diversas ocasiões, uma vez que ele não avisa ao motorista que ele está excedendo a velocidade máxima do trecho e os radares não estão presentes em toda extensão da pista, apenas em pontos específicos. Como cada trecho da rodovia pode ter uma velocidade máxima, variando inclusive com fatores climáticos, acidentes, obras, etc., o motorista gostaria de ser avisado sobre a velocidade máxima permitida e de ser alertado quando exceder esse limite, uma vez que isso pode acontecer inconscientemente. Seria

interessante ao motorista um sistema que atuasse a seu favor e o ajudasse a guiar seu veículo de maneira segura.

As rodovias também utilizam painéis luminosos para informar os motoristas sobre algumas condições dos trechos da pista. Esses painéis são de grande utilidade ao motorista. Porém, além de estarem presentes em poucos lugares da pista, suas informações só podem ser acessadas pelo motorista quando ele passar pelo painel. Além disso, se houver mais informações que não couberam no painel, ou se não houver tempo suficiente para que o motorista as leia, ou ainda se houver mudanças nessas informações, o motorista não consegue obter todas as informações que deveria.

Em carros mais modernos, encontram-se reguladores de velocidade (*Cruise Control*), onde o motorista configura a velocidade desejada e o carro mantém essa velocidade automaticamente ao longo da viagem. Como cada trecho da rodovia pode ter uma velocidade máxima diferente (variando de acordo com os fatores citados anteriormente), utilizar o regulador de velocidade pode não ser conveniente.

Tendo isso em vista, seria interessante ao motorista dispor de um sistema que atuasse a seu favor, de forma preventiva, avisando-o antecipadamente sobre as condições do trecho da rodovia em que trafega para que ele possa se prevenir e aumentar a segurança de sua viagem. Essas informações devem ser alteradas dinamicamente, conforme as condições dos trechos das rodovias mudam, e devem poder ser acessadas pelo motorista a qualquer momento de sua viagem. Esse sistema também deve monitorar a velocidade atual do motorista e alertá-lo quando ele exceder o limite de velocidade do trecho em que está localizado, aumentando sua segurança e evitando multas.

1.3 Objetivo

O objetivo deste projeto é aplicar a tecnologia atual nas rodovias a favor dos motoristas mostrando a viabilidade técnica de um sistema de automação rodoviária que visa diminuir o número de acidentes em uma rodovia através da transmissão de informações aos motoristas sobre a velocidade máxima permitida em sues trechos e

sobre mudanças nas condições da mesma, como buracos, acidentes na pista, obras, engavetamento e condições climáticas, alertando-os sobre situações de perigo para que eles possam tomar ações preventivas e, conseqüentemente, aumentando a segurança na rodovia.

Para isso foi proposto um sistema com alguns módulos ao longo da rodovia que se comunicam através da rede de telefonia móvel com a central administrativa da mesma. A central transmite as informações para os módulos rodoviários, os quais repassam essas informações para módulos instalados no interior dos veículos através de rádio-freqüência. O módulo veicular possui um display LCD que exibe as informações para o motorista. Além disso, esse módulo monitora a velocidade instantânea do veículo e a compara com a velocidade máxima permitida, alertando o motorista caso sua velocidade seja superior ao limite permitido no trecho.

2 Especificação

Neste capítulo serão apresentadas as especificações do projeto, seus requisitos funcionais e não-funcionais, assim como suas restrições.

2.1 Requisitos funcionais

O sistema proposto possui os seguintes requisitos funcionais:

1. Cadastro e atualização de informações dos trechos de uma rodovia

   O sistema deve permitir que a administradora da rodovia cadastre e atualize informações de diferentes trechos da rodovia através de uma interface Web.

   As informações que deverão ser cadastradas para um determinado trecho são:

   - Velocidade máxima permitida
   - Buracos na pista. Será informado o local (quilômetro e faixa)
   - Ocorrência de acidentes. Será informado o local (quilômetro e faixa)
   - Obras na pista. Será informado o local (quilômetro e faixa)
   - Condições do trânsito. Será informada a extensão do trânsito (quilômetro inicial e final) e alternativas se houver (quilômetro de saída e nome da rodovia)
   - Condições do tempo

2. Transmissão das informações da central para o módulo da rodovia

   Em alguns trechos da rodovia, serão instalados módulos (MR) que receberão as informações cadastradas pela administradora (relativas ao seu trecho de cobertura) e as armazenarão em sua memória.

3. Transmissão das informações do módulo da rodovia para o módulo do veículo

Nos veículos, serão instalados módulos simples (MV) que receberão os dados (citados anteriormente no primeiro requisito funcional) dos trechos ao passarem pelos MR que estarão transmitindo constantemente essas informações para que todos os veículos que entrem na área de cobertura da antena possam recebê-las.

4. Exibição das informações e alerta ao motorista

   As informações recebidas serão exibidas em um display LCD. Quando novas informações forem recebidas, um *beep* de alerta avisará o motorista que há informações novas disponíveis.

5. Manipulação do módulo de veículo

   O motorista poderá manipular as telas e opções do MV. Para isso, haverá um botão para troca de telas, sendo que cada tela corresponderá a um tipo de informação a ser exibida. Haverá outro botão para que o motorista manipule as opções do MV, como habilitar/desabilitar o *beep* e sair da rodovia.

6. Monitoração e alerta da velocidade

   O sistema deve capturar a velocidade atual do veículo e compará-la com a velocidade máxima permitida no trecho em que o motorista trafega. Em caso de excesso de velocidade, um alerta sonoro será emitido.

2.2 Requisitos não-funcionais

1. O sistema deve ser capaz de realizar a transferência de dados dos módulos da rodovia para os módulos dos veículos com velocidades de até 150km/h (25% acima da velocidade máxima das rodovias brasileiras, que é de 120km/h);

2. A transmissão dos dados deverá utilizar algum método de criptografia para garantir a validade dos dados e evitar que pessoas mal intencionadas enviem informações para os carros;

3. A transmissão das informações deverá conter algum tipo de redundância de informação para aumentar a segurança.

2.3 Suposições

1. O gerenciamento das informações transmitidas é de responsabilidade da central administrativa da rodovia (como, por exemplo, a verificação de buracos e acidentes na pista e as conseqüências que isso traz para a velocidade do trecho);

2. A central administrativa da rodovia já possui um sistema próprio de administração. O sistema proposto nesse trabalho fornecerá apenas uma interface Web segundo um protocolo para que o sistema da central possa enviar informações e atualizar a memória dos MR;

3. O sistema depende da disponibilidade da rede GPRS para a transmissão de dados

2.4 Restrições

1. A transmissão das informações da central de administração rodoviária para cada módulo de rodovia deverá ser feita através da tecnologia GPRS devido à sua ampla cobertura;

2. A transmissão das informações dos módulos das rodovias para os módulos dos veículos deverá ser feita através de rádio-freqüência. Assim, eliminam-se os custos com transferência de dados tanto para a administração da rodovia quanto para os motoristas (que não precisarão ter plano de dados com operadoras de

celular) e torna essa transmissão independente de problemas que possam ocorrer com a rede de celular.

3 Arquiteturas

Neste capítulo serão apresentadas as arquiteturas geral, do MV e do MR.

3.1 Arquitetura geral do sistema

O diagrama de blocos exibido a seguir representa a arquitetura geral do sistema

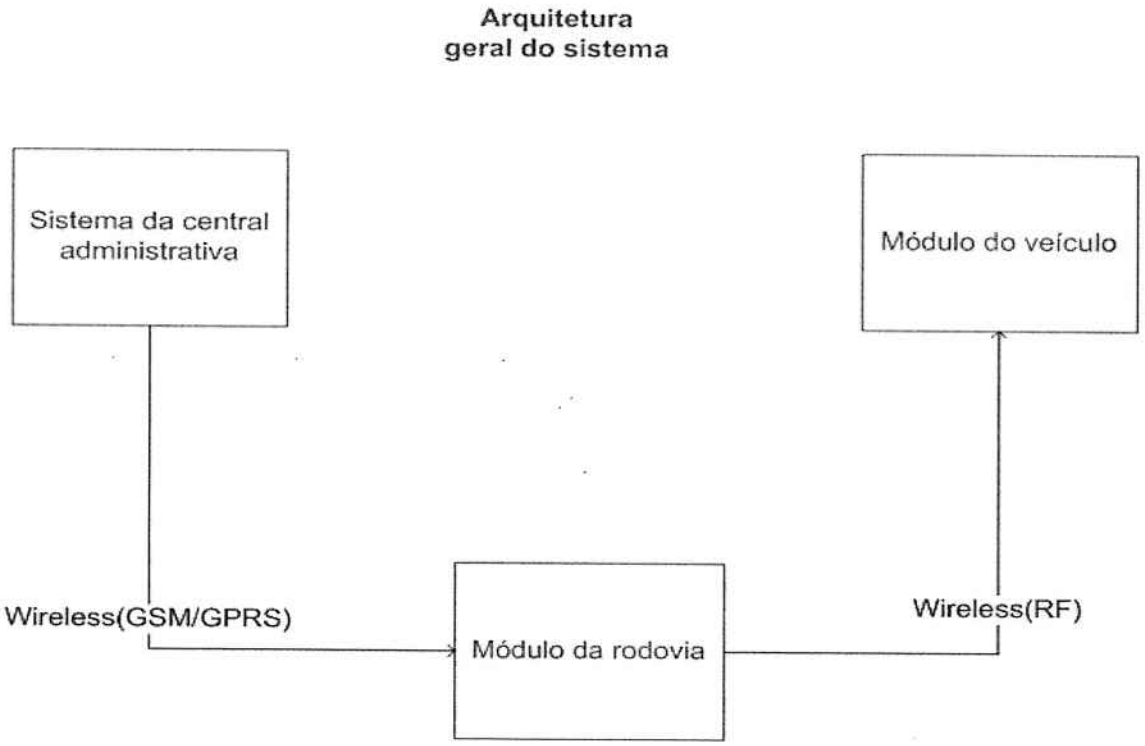


Figura 1: diagrama de blocos da arquitetura geral do sistema

A empresa responsável pela administração da rodovia possui um sistema próprio localizado em sua central administrativa. Esse sistema concentra informações dos diversos trechos da rodovia, tais como condição climática, presença de buracos, acidentes ou obras na pista, condições de trânsito, entre outras. A captura e a administração dessas informações não são cobertas pelo sistema proposto nesse trabalho. O "Sistema veicular de controle de velocidade e informações rodoviárias"

fornecerá uma interface para que o sistema da central administrativa possa enviar informações para os MR.

A comunicação entre o sistema da central administrativa e os MRs é unidirecional (do sistema da central para os MRs). Será feita através da rede GSM, utilizando a tecnologia SMS. A escolha dessa tecnologia foi feita baseando-se nos seguintes critérios:

- Comunicação *wireless* entre os sistemas (não haverá conexões entre eles através de fios devido à distância entre os mesmos)
- Área de cobertura (na data em que esse trabalho foi proposto, a tecnologia GSM/GPRS é a que possui maior área nacional de cobertura, possibilitando seu uso em regiões mais remotas da rodovia)
- Diversidade de fornecedores de equipamentos GSM
- Baixo preço dos equipamentos

Os MRs são módulos instalados em diversos trechos da rodovia, de acordo com a necessidade identificada pela empresa administradora. Assim, em regiões de maior perigo, em que as informações são constantemente atualizadas, a distância entre os módulos pode ser menor do que regiões mais seguras, em que as informações não sofrem muitas variações. O MR recebe os dados da central administrativa, realiza um processamento e os armazena em sua memória.

O MR transmitirá constantemente os dados para os MVs localizados nos veículos que trafegam pela rodovia. Essa transmissão será feita através de RF, devido às seguintes vantagens:

- Independência de empresas de telecomunicação (o canal de comunicação não precisa de nenhum serviço externo). Caso a rede de celular fique inoperante, a transmissão dos dados para os MVs continua ocorrendo.
- Baixo custo: não é necessário que o motorista ou a empresa administradora pague pela transmissão das informações como aconteceria caso fosse usada uma rede de celular para essa comunicação. Como essas informações são transmitidas continuamente e um número muito grande de veículos trafega na rodovia, economiza-se muito com a transmissão de dados. O único gasto é em relação à energia consumida na transmissão.

O MV receberá esses dados e realizará um processamento dos mesmos, armazenando-os em sua memória. As informações serão exibidas ao motorista para auxiliá-lo na condução do veículo. Além disso, o MV captura a velocidade do veículo e avisa o motorista quando sua velocidade for maior do que a velocidade permitida no trecho da rodovia em que está trafegando.

3.2 - Arquitetura do MR

O diagrama de blocos a seguir apresenta a arquitetura do hardware do MR:

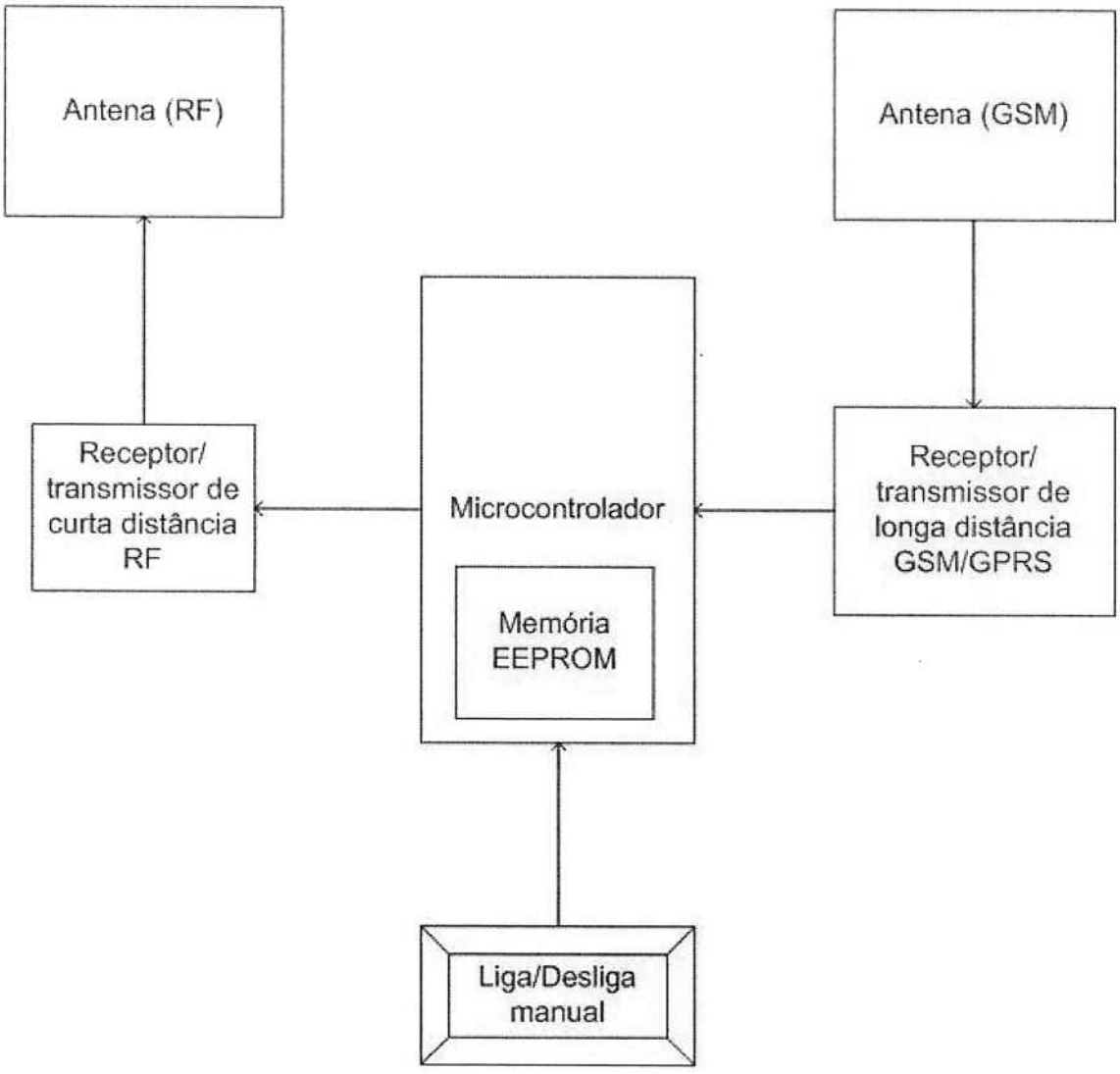


Figura 2: diagrama de blocos do hardware do módulo rodoviário

Nessa arquitetura, há um microcontrolador responsável por controlar a operação de dois componentes de comunicação sem fio.

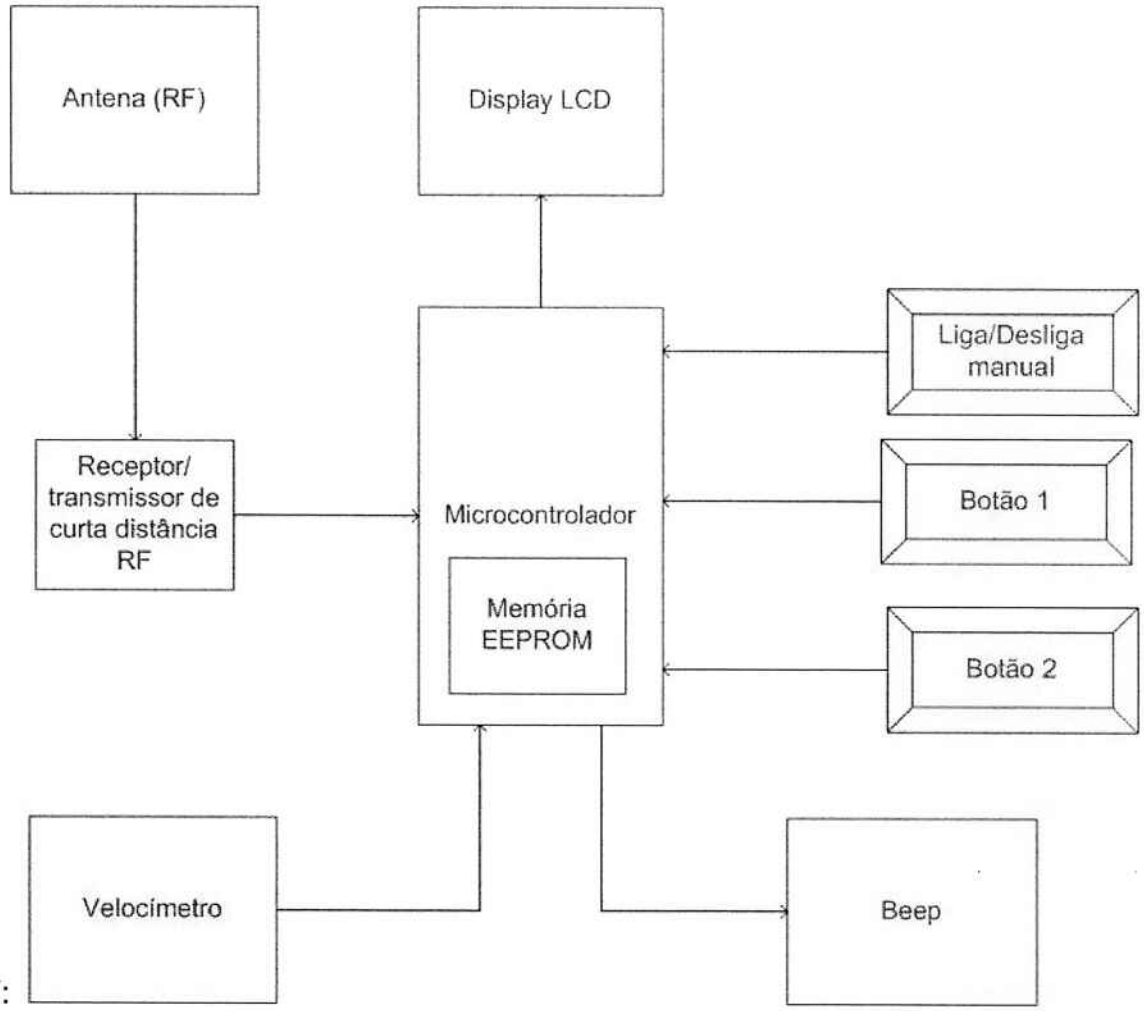
O microcontrolador envia comandos para o receptor/transmissor GSM/GPRS para coordenar a recepção das informações do trecho, enviadas pelo sistema da central administrativa. O receptor deve ser conectado a uma antena externa específica para a rede GSM, uma vez que não possui antena própria. Feita a recepção, o microcontrolador extrai os dados do receptor GPRS e, após realizar um processamento para formatá-los, armazena-os em sua memória EEPROM.

O microcontrolador irá continuamente ler os dados de sua memória e os enviar via RF para os MVs. Para isso, ele fornece esses dados para o receptor/transmissor de RF e envia comandos para controlar a operação de transmissão desse componente. Para aumentar a área de alcance, uma antena externa de RF é conectada ao transmissor.

O microcontrolador está conectado a um botão de liga/desliga, caso se necessário desativar o MR devido a algum problema ou simplesmente reiniciá-lo.

3.3 Arquitetura do MV

O diagrama de blocos a seguir apresenta a arquitetura do hardware do



MV:

Figura 3: diagrama de blocos do hardware do módulo veicular

Nessa arquitetura, há um microcontrolador responsável por coordenar a ação dos componentes a ele conectados

O microcontrolador envia comandos ao receptor/transmissor de RF para que ele fique em modo de recepção e receba os dados transmitidos pelo MR. Para aumentar a área de cobertura do equipamento, uma antena externa é conectada ao receptor RF. Após a recepção, o microcontrolador extrai os dados do receptor RF e realiza um processamento para formatá-los, armazenando-os em sua memória.

As informações recebidas são exibidas no display LCD. Como o número de caracteres que podem ser exibidos no display LCD é pequeno, não é possível

apresentar todas as informações simultaneamente no mesmo. Então, o microcontrolador deve ser capaz de formatar os dados e gerar alguns tipos de tela, cada uma associada a um tipo de informação que se deseja exibir (por exemplo, velocidade máxima ou buracos). Para realizar a troca dessas telas, o botão 1 deverá ser pressionado, gerando um sinal para o microcontrolador realizar a troca.

O microcontrolador também faz a leitura dos dados do velocímetro do veículo para obter a velocidade atual. Com essa informação, ele compara a velocidade atual com a máxima permitida no trecho. Caso a velocidade atual seja superior à máxima, o microcontrolador envia um sinal a um *beep* para que ele emita um alerta sonoro ao motorista.

Além disso, algumas configurações do aparelho podem ser feitas pelo motorista, como desligar/ligar o *beep*, por exemplo. Para isso o botão 2 é conectado ao microcontrolador. Ao ser pressionado, o microcontrolador exibirá telas de configuração, que poderão ser manipuladas através dos botões 1 e 2.

Há também um botão para ligar e desligar o equipamento.

4 Implementação

Neste capítulo serão apresentados todos os aspectos relativos à implementação do projeto.

4.1 Objetivo

A especificação e as arquiteturas apresentadas nos itens anteriores referem-se ao "Sistema veicular de controle de velocidade e informações rodoviárias" de um ponto de vista global, sem entrar em detalhes de como implementá-lo. Na verdade, há varias maneiras de realizar essa implementação, que podem variar em aspectos como: componentes de hardware utilizados, protocolos de comunicação RF, estrutura do software do MR e MV, entre outros.

O objetivo desta seção é apresentar uma sugestão de implementação para o sistema proposto nesse trabalho. Foram definidos os componentes de hardware e protocolos RF e construído um protótipo do sistema. Os próximos itens dentro desta seção visam descrever as características desse protótipo, sua implementação (hardware e software) e os resultados obtidos experimentalmente.

4.2 Escopo do protótipo

O protótipo construído para a demonstração prática do "Sistema veicular de controle de velocidade e informações rodoviárias" apresenta um escopo reduzido em relação ao escopo do sistema descrito no item 2 devido às limitações de recurso e prazo para a construção do mesmo. As seguintes reduções foram feitas:

- A velocidade do veículo será simulada em software para testar por indisponibilidade de tempo para conseguir estudar uma forma de ligar o microcontrolador ao velocímetro do carro

- O beep será substituído por um led por não ser necessário o toque do beep para chamar a atenção do motorista nos testes.
- Não foi utilizado nenhum tipo de criptografia e redundância na transmissão dos dados.

4.3 Componentes de hardware

Os componentes de hardware utilizados na elaboração do protótipo são:

- **Microcontrolador CC2510**

O microcontrolador que será utilizado tanto no MR quanto no MV será o CC2510 da Chipcon, que é um microcontrolador baseado no Intel 8051, com diversas modificações. A principal delas, que justifica a escolha do CC2510 entre os demais microcontroladores, é a adição de um transmissor/receptor de ondas de rádio-freqüência (RF), eliminando-se a necessidade de usar um componente específico para essa finalidade. Além disso, o CC2510 foi projetado para sistemas embarcados, consumindo pouca energia devido aos seus modos de operação. As principais características que justificam sua escolha são:

  - Transmissor/receptor de RF integrado na faixa de 2,4 GHz
  - Alta sensibilidade (-107 dBm a 2.4kBaud)
  - Baixo consumo de corrente (RX: 9.1 mA)
  - Trabalha com modulação FSK a uma taxa de até 76,8 kBaud
  - Otimização do núcleo do Intel 8051, permitindo um desempenho de até 2,5 vezes o 8051 original
  - Intel 8051 é um microcontrolador bem conhecido e consolidado há um tempo no mercado
  - Modos de operação idle e sleep (preserva a bateria)
  - Suporte a criptografia DES via hardware
  - 32 kB de memória flash

Apesar do microcontrolador Intel 8051 (que é o núcleo do CC2510) já estar relativamente ultrapassado, uma vez que existem microcontroladores atualmente

muito mais poderosos que ele, a aplicação não exige processamento complexo de dados, sendo que o microcontrolador é capaz de realizar suas tarefas propostas com baixo consumo de energia.

Alguns componentes externos ao CC2510 deverão ser usados para permitir seu correto funcionamento. Para mais informações sobre o CC2510, consultar o seu *datasheet* em (TEXAS INSTRUMENTS CORPORATED).

- **Display LCD WH2002A**

O display LCD WH2002A será utilizado no MV para exibir informações ao motorista. O display contém 2 linhas com 20 caracteres cada. Para que seja possível ler as mensagens à noite, esse display foi escolhido porque contém *backlight*. Suas principais características são:

  - 2 linhas x 20 caracteres
  - LED backlight
  - Controlador interno (KS0066 ou equivalente)
  - Alimentação 5V
  - 1/16 Duty Cycle
  - Alimentação do LED - 4.2V

Para mais informações sobre o display LCD WH2002A, consultar o seu *datasheet* em (WINSTAR DISPLAY).

- **Transmissor/Receptor GSM/GPRS MC35i Terminal**

O módulo GSM/GPRS que será utilizado é o MC35i Terminal da Siemens que possui todas as funcionalidades necessárias para atender aos requisitos do projeto.

O módulo MC35i pode transmitir voz, dados, SMS e fax. Nesse projeto , será utilizada a transmissão de dados.

As freqüências de banda atendidas são E-GSM 900 e GSM 1800, com potência de conexão classe 4 (2W) e classe 1 (1W) respectivamente. As operadoras de celular Claro e Tim podem ser utilizadas para fazer o envio de dados à longa distância através da rede GPRS.

A comunicação via GPRS pode realizar a recepção de dados a 85.6 kbps e seu envio a 21.4 kbps.

O terminal possui diversos conectores para alimentação de energia, interface e antena, ilustrados na figura abaixo e listados a seguir: interface de áudio, LED de status, entrada para chip SIM, interface RS-232, entrada de energia e entrada da antena de transmissão de longa distância. A interface de áudio não será utilizada neste projeto, que envolve apenas a transmissão de dados.

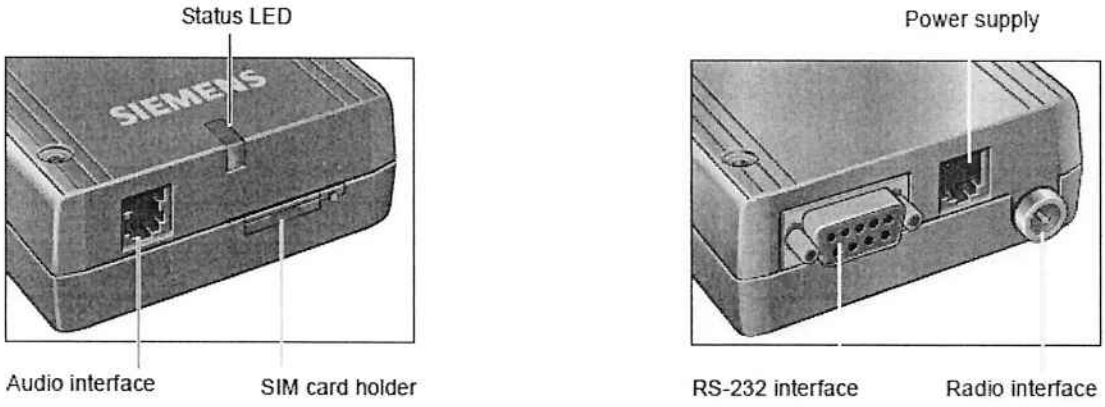


Figura 4: conectores do terminal MC35i

O principal componente do MC35i Terminal é o componente MC35i GSM/GPRS *engine*. É o componente que administra todo o processamento de áudio, sinais e dados da rede GSM/GPRS. Ele possui um software que executa internamente a aplicação de interface e os protocolos GSM e GPRS.

- **MAX232**

O componente MAX232 é um conversor de tensão de nível TTL (nível lógico 0 em 0 volts e nível lógico 1 em 3 volts) para nível RS-232 (nível lógico 0 em 12 volts e nível lógico 1 em -12 volts).

O MAX232 foi utilizado para converter a tensão entre o microcontrolador CC2510, que utiliza nível TTL, e o modem MC35i, que utiliza a interface RS232.

- **Antena RF**

A antena RF utilizada foi a antena embutida no kit de desenvolvimento do microcontrolador CC2510, que será descrito no item 4.5 deste documento.

4.4 Protocolos de comunicação

A seguir os protocolos utilizados no envio de dados entre os módulos do sistema.

4.4.1 Protocolos RF

Nesta seção serão descritos os protocolos utilizados para realizar a comunicação via Radio Freqüência (RF).

4.4.1.1 SimpliciTI

SimpliciTI é um protocolo criado pela Texas Instruments para comunicação em rádio-freqüência entre equipamentos com baixo consumo de energia, como o CC2510. Foi criado para transferências em que os dispositivos precisam economizar energia ao máximo para aumentar o tempo de vida de suas baterias.

Também foi projetado para consumir pouca memória. Ocupa 8K de memória flash e 1K de memória RAM.

A comunicação é iniciada através de links criados entre dois CC2510. O mestre, através do comando *ping*, procura por outros aparelhos que se comuniquem na mesma freqüência. Quando encontrado, inicia a criação do link entre os dois aparelhos. O escravo aceita a comunicação e a partir desse ponto pode ser iniciada a transferência de dados entre os módulos.

O protocolo SimpliciTI possui suporte a criptografia para proteger os dados contra invasões.

O pacote transferido por rádio-freqüência tem o seguinte formato e o seguinte número de bytes.

Tabela 1: pacote do protocolo simpliciTI

Preamble	Sync	Length	Misc	Dstaddr	Srcaddr	Port	Device Info	Tractid	App Payload	FCS
RD	RD	1	RD	4	4	1	1	1	N	RD

RD: dependente do rádio.

Tabela 2: campos do pacote do protocolo simpliciTI

Campo	Definição	Comentário
Preamble	Sincronização do rádio	Inserido pelo hardware do rádio
Sync	Sincronização do rádio	Inserido pelo hardware do rádio
Length	Tamanho do resto do pacote	Inserido pelo firmware na transmissão
Misc	Campos diversos	Depende do rádio, pode não existir
Dstaddr	Endereço de destino	Inserido pelo firmware
Srcaddr	Endereço fonte	Inserido pelo firmware
Port	Quadro transmitido (7), Contexto da criptografia (6), Número da porta (5-0).	Inserido pelo firmware. Para aplicação do usuário são reservadas as portas 0x20-0x3F e 0-0x1F para gerenciamento
Device Info	Informações do transmissor e do receptor	Inserido pelo firmware
Tracid	Id da transação	Inserido pelo firmware
App Payload	Dados da aplicação	0 < N < 114 bytes
FCS	"Frame Check Sequence"	CRC de checagem de dados

		recebidos

4.4.1.2 Protocolo BBF

O protocolo BBF é um protocolo de autoria dos integrantes desse trabalho (a sigla BBF é baseada nos nomes dos autores). Ele define os formatos de mensagens trocadas entre o MR e o MV e suas respectivas estruturas. A mensagem, após estar de acordo com o especificado pelo protocolo BBF, é encapsulada pelo protocolo SimpliciTI no campo *payload*.

Como o protocolo BBF foi projetado para ser encapsulado pelo SimpliciTI, o tamanho máximo da mensagem especificada pelo protocolo BBF é igual ao tamanho máximo do campo *payload* do SimpliciTI (que pode ser configurado de acordo com a aplicação). No caso do protótipo construído nesse trabalho, o tamanho máximo utilizado foi 30 bytes. Entretanto, o protocolo BBF pode ser facilmente ajustado para permitir mensagens de tamanhos maiores que 30 bytes caso seja utilizado um *payload* de tamanho maior.

Os tipos de mensagem definidos pelo protocolo BBF são:

Tabela 3: tipos de mensagem do protocolo BBF

Tipo	Descrição
0	Entrada/Saída de rodovia
1	Velocidade máxima
2	Buracos
3	Acidente/Obras
4	Trânsito (com alerta)
5	Trânsito (sem alerta)
6	Tempo (com alerta)
7	Tempo (sem alerta)

Em todos esses tipos de mensagem, há dois campos destinados ao controle de informações sobre a potência de transmissão. O campo "Potência requisitada" é utilizado para que um módulo possa requisitar ao outro módulo que ajuste sua potência de transmissão de acordo com a distância entre eles, uma vez que

potências muito altas podem saturar o sinal se a distância for pequena e potências muito baixas podem fazer com que o sinal não chegue ao outro módulo se a distância for grande. O campo "Potência atual" informa a potência do módulo transmissor para que o receptor possa calcular a potência requisitada. Com esses dois campos, os módulos conseguem configurar a potência ideal de acordo com a distância em que estão.

A seguir, serão descritos os tipos mencionados acima, apresentando-se as informações que eles transmitem e qual a estrutura de suas mensagens.

- **Tipo 0 - Entrada/Saída de rodovia**

Descrição: Esse tipo de mensagem é transmitido por MR posicionados nos pontos de entrada/saída da rodovia. Ele informa o código e o nome da rodovia na qual o motorista está trafegando.

Estrutura:

Tabela 4: entrada/saída de rodovia

Byte	0	1	2	3	4	5 a 29
Descrição	Potência requisitada	Potência atual	Tipo -> 0	Código da rodovia (+sig)	Código rodovia (-sig)	Nome da rodovia

- **Tipo 1 - Velocidade máxima**

Descrição: Esse tipo de mensagem informa a velocidade máxima permitida no trecho coberto pelo MR.

Estrutura:

Tabela 5: velocidade máxima

Byte	0	1	2	3	4
Descrição	Potência requisitada	Potência atual	Tipo -> 1	Vmax (+sig)	Vmax (-sig)

- **Tipo 2 - Buracos**

Descrição: Esse tipo de mensagem informa a localização de buracos no trecho coberto pelo MR. O buraco pode ser especificado individualmente, informando-se somente os campos relativos ao "Km inicial" e "Faixa 1". É possível indicar até quatro buracos dessa maneira. Outro jeito de apontar buracos, quando há muitos deles em um determinado trecho, é através da indicação de um trecho esburacado que se estende por uma faixa de quilômetros e está situado entre algumas ou todas as faixas da pista Nesse caso, deve-se informar os campos "Km inicial", "Km final", "Faixa 1" e "Faixa 2". Essas faixas são as faixas que limitam a parte da pista com buracos. Caso somente uma faixa apresente buracos, essa faixa deve ser indicada nos dois campos

Estrutura:

Tabela 6: buraco

Byte	0	1	2	3	4 a 9	10 a 15	16 a 21	22 a27
Descrição	Potência requisitada	Potência atual	Tipo -> 2	Número de buracos	Buraco 0	Buraco 1	Buraco 2	Buraco 3

Buraco i (0 <= i <= 3)

Byte	4 + 6*i	5 + 6*i	6 + 6*i	7 + 6*i	8 + 6*i	9 + 6*i
Descrição	Km inicial (+sig)	Km inicial (-sig)	Km final (+sig)	Km final (-sig)	Faixa 1	Faixa 2

- **Tipo 3 - Acidente/Obras**

Descrição: Esse tipo de mensagem informa a localização da ocorrência de acidentes ou execução de obras no trecho coberto pelo MR. Podem ser informados até três desses eventos (acidentes ou obras). A delimitação do local pode ser feita por trecho (informando-se os campos "Km inicial", "Km final", "Faixa 1" e "Faixa 2") ou por uma localização específica (informando-se os campos "Km inicial", "Faixa 1")

Estrutura:

Tabela 7: acidente/obra

Byte	0	1	2	3	4 a 10	11 a 17	18 a 24
Descrição	Potência requisitada	Potência atual	Tipo -> 3	Número de ocorrências	Acidente/ Obra 0	Acidente/ Obra 1	Acidente/ Obra 2

Acidente/Obra i (0 <= i <= 2)

Byte	4 + 7*i	5 + 7*i	6 + 7*i	7 + 7*i	8 + 7*i	9 + 7*i	10 + 7*i
Descrição	0 -> Obra 1 -> Acidente	Km inicial (+sig)	Km inicial (-sig)	Km final (+sig)	Km final (-sig)	Faixa 1	Faixa 2

- **Tipos 4 e 5 - Trânsito com alerta e sem alerta**

Descrição: Esse tipo de mensagem informa um trecho da rodovia com trânsito intenso, coberto pelo MR. Devem ser especificados os campos "km inicial" e "km final" do trecho com trânsito. Além disso, pode ser informada uma rodovia alternativa. Nesse caso, devem ser especificados o nome da rodovia e o km de saída (campo "Km alternativa"). O tipo 4 é uma mensagem de trânsito com um alerta para que o motorista reduza sua velocidade antecipadamente. Ele deve ser usado em situações em que irá ocorrer uma variação brusca de velocidade, como um engarrafamento. O tipo 5 é uma mensagem de trânsito sem alerta, apenas para informar o motorista sobre uma situação de lentidão. Ele deve ser usado em situações em que apenas um tráfego intenso está causando uma lentidão, porém não há grandes variações de velocidade. A vantagem é que esse tipo de mensagem não causa troca de tela no display LCD do MV, deixando prevalecer a tela atual ou de maior prioridade

Estrutura:

Tabela 8: trânsito

Byte	0	1	2	3	4	5	6	7	8	9 a 29
Descrição	Potência requisitada	Potência atual	Tipo: 4 -> com alerta 5 -> sem alerta	Km inicial (+sig)	Km inicial (-sig)	Km final (+sig)	Km final (-sig)	Km alternativa (+sig)	Km alternativa (-sig)	Nome da rodovia alternativa

- **Tipo 6 e 7 - Tempo com alerta e sem alerta**

Descrição: Esse tipo de mensagem informa a previsão ou condição do tempo. O tipo 6 é uma mensagem com alerta. Ele é usado para prevenir o motorista das condições do tempo e da pista, alertando-o para uma redução na velocidade. Ele deve ser utilizado em situações de perigo, como, por exemplo, pista molhada ou neblina. O tipo 7 define uma mensagem sem alerta. Ele deve ser usado somente para mensagens que não necessitem que o motorista seja alertado para reduzir a velocidade do veículo, como previsão do tempo ou informação da temperatura, por exemplo.

Estrutura:

Tabela 9: tempo

Byte	0	1	2	3 a 29
Descrição	Potência requisitada	Potência atual	Tipo: 6 -> com alerta 7 -> sem alerta	Descrição do tempo

4.4.2 Protocolo Serial

Definição do protocolo serial utilizado no projeto.

4.4.2.1 RS-232

A comunicação entre o modem GSM/GPRS (Siemens MC35i) e o microcontrolador (CC2510) é realizada por meio de transmissão serial de dados, através do protocolo RS-232. O protocolo RS-232 utilizado conta com os seguintes parâmetros:

- 8 bits de dados
- 1 "stop bit"
- Sem bit de paridade
- Sem controle de fluxo
- 9600bps

O protocolo serial completo conta com o uso de nove pinos, porém para esta aplicação utilizamos apenas três: RX, TX e GND, o essencial para conseguir transmitir dados. Esta escolha foi feita porque a implementação do *driver* serial no microcontrolador CC2510 é de responsabilidade do grupo desenvolvedor. Assim, buscou-se uma simplificação do protocolo, sem sacrificar funcionalidades.

Este protocolo também define níveis de tensão que diferem dos normalmente utilizados em componentes eletrônicos. Abaixo, é apresentada a tabela dos níveis lógicos e de tensão do RS-232.

Tabela 10: níveis lógicos RS-232

Nível Lógico	Tensão
1	-3...-20V
0	3....20V

Tensões na faixa de -3V até 3V são desconsideradas. Assim, para utilizar este protocolo com o CC2510, que opera de acordo com a tabela de níveis lógicos apresentada abaixo, deve-se usar um conversor de tensão, como o MAX232.

Tabela 11: níveis lógicos CC2510

Nível Lógico	Tensão
1	3.3V
0	0V

4.4.3 Protocolo de Comunicação via Rede GSM

O protocolo para enviar os dados do sistema da central administrativa ao MR consiste em mandar SMSs com cada byte do SMS correspondendo a um byte de dados da aplicação.

O SMS possui duas codificações possíveis para o envio de mensagens. Uma delas é a GSM, que é a padrão de envio de SMS. Essa codificação utiliza apenas 7 bits para cada caractere e portanto não possui todos os caracteres ASCII enviados pelo sistema web. A outra codificação é a UCS2, utilizada para SMSs que possuem caracteres a mais que o padrão, como a língua russa ou chinesa.

Para enviar os bytes ao MR, é necessário utilizar a codificação UCS2, que suporta os 8 bits dos caracteres ASCII.

A codificação UCS2 limita o tamanho do SMS em 70 caracteres. Caso a mensagem possua um número de caracteres superior a este, ela deve ser quebrada em um número adequado de mensagens SMS.

A UCS2 transforma cada caractere ASCII em 4 bytes, sendo os dois primeiros 00 e os dois últimos o valor hexadecimal de cada algarismo do código ASCII. Por exemplo: em ASCII a letra A é 41H, convertido para UCS2 fica 30H 30H 34H 31H (em ASCII: 30H é 0, 34H é 4 e 31H é 1) .

4.5 Desenvolvimento

Nesta sessão foram discorridas todas as etapas para desenvolver os projetos, dividido entre os principais artefatos gerados no projeto.

4.5.1 Hardware do MR

Como pode ser visto na figura de arquitetura geral do MR, na seção 3.2, o MR tem os seguintes componentes de hardware:

- CC2510 – Microcontrolador e Transmissor/Receptor de RF
- MC35i – Modem GSM/GPRS
- MAX232 – Conversor de Nível de Tensão

O CC2510, por meio de seus pinos de Entrada/Saída, se comunica serialmente com o modem MC35i, utilizando o MAX232 para converter os sinais do CC2510 para os níveis do padrão RS-232.

A alimentação destes componentes também varia, de acordo com a seguinte tabela:

Tabela 12: tensão de alimentação – hardware MR

Componente	Tensão
CC2510	3V
MAX232	3V
MC35i	8...20V

Assim, é utilizada uma fonte de 12V para alimentar o modem MC35i, e duas pilhas em série para alimentar tanto o CC2510 quanto o MAX232, sendo que o terra é comum para todos os componentes do circuito. A interligação destes componentes, por se tratar de um protótipo, foi feita com uma protoboard, para permitir maior flexibilidade e modularidade ao sistema, facilitando a adição e substituição de componentes, assim como reduzir os custos de desenvolvimento do protótipo.

4.5.2 Hardware do MV

O MV conta com três componentes:

- CC2510 – Microcontrolador e Recepção RF
- Display LCD – Exibição de dados ao motorista
- *Beep* – Aviso sonoro

Assim, o MV conta com uma interligação mais simples do que o módulo rodoviário, pois não há necessidade de conversão de tensão. Tanto o display LCD quanto o *beep* são ligados diretamente na placa de desenvolvimento do CC2510.

O display de LCD tem como tensão de entrada 3V, podendo variar de 2.7V a 3.3V. Portanto a tensão do CC2510 e do display são equivalentes, podendo ser interligadas diretamente.

4.5.3 Software geral

O desenvolvimento do software que será executado no microcontrolador CC2510 foi feito utilizando o IAR Embedded Workbench IDE, ferramenta indicada pela Texas Instruments para o desenvolvimento nesse microcontrolador. O software desenvolvido pode ser testado e depurado utilizando o componente "CC Debugger", que faz parte do kit de desenvolvimento do CC2510. O "CC Debugger" é conectado a um PC pela porta USB e faz a interface com a placa de desenvolvimento do CC2510, permitindo acesso à memória, registradores, pilha, etc. em tempo de execução, o que auxilia muito o desenvolvimento do software.

4.5.3.1 Linguagem

Para o desenvolvimento do software do protótipo, tanto para o MR quanto para o MV, foi utilizada a linguagem C devido às necessidades típicas de sistemas embarcados, como:

- Acesso direto aos recursos do hardware;
- Software com desempenho otimizado devido à baixa capacidade de processamento do microcontrolador;
- Baixa alocação de memória devido à pequena memória disponível para o programa.

4.5.3.2 Bibliotecas importadas

Foram utilizadas algumas bibliotecas fornecidas pela *Texas Instruments* para o kit de desenvolvimento SmartRFCC2510 e para o protocolo SimpliciTI.

Em relação ao kit SmartRFCC2510, as bibliotecas utilizadas visam controlar os componentes que compõem a placa do kit, que são os botões, LEDs e

transmissor/receptor de RF. Além disso, há alguns arquivos com definições e funções úteis ao programador em relação ao processador Intel 8051.

Em relação ao protocolo SimpliciTI, há uma biblioteca que implementa esse protocolo e disponibiliza algumas configurações e funções de utilização do mesmo, que permitem ao programador do CC2510 enviar e receber dados através desse protocolo com poucas linhas de código.

Essas bibliotecas podem ser encontradas em (TEXAS INSTRUMENTS CORPORATED).

4.5.3.3 Bibliotecas desenvolvidas

Além das bibliotecas fornecidas pela *Texas Instruments* que foram importadas no software, foram implementadas novas bibliotecas para auxiliar o desenvolvimento dos softwares do MR e do MV.

4.5.3.3.1 Display LCD

Foi desenvolvida uma biblioteca para controlar o display LCD. Essa biblioteca contém as definições das portas (e seus respectivos pinos) do CC2510 que foram utilizadas para se conectar ao display, tanto para a transmissão de bytes de dados quanto de controle. Além disso, a biblioteca fornece métodos para inicialização do display e escrita de *strings* em ambas as suas linhas. Internamente, a biblioteca, ao receber as *strings*, coordena o envio de bytes de dados e de controle para o display, de acordo com um *delay* adequado entre os comandos, para que a informação seja exibida corretamente no display. Essa biblioteca pode ser encontrada em 8.4.3.

4.5.3.3.2 Porta Serial

Também foi desenvolvida uma biblioteca para comunicação serial (USART). Isto foi necessário uma vez que a placa de desenvolvimento utilizada não conta com uma porta RS-232, apenas 12 pinos para Entrada/Saída genéricos. Esta biblioteca fornece métodos para envio e recepção de dados serializados, por meio dos pinos de Entrada/Saída disponíveis na placa de desenvolvimento do CC2510. Algumas funcionalidades do microprocessador foram utilizadas para conseguir realizar a transferência com confiabilidade. Assim, a geração da freqüência correta de transmissão (*baud rate*) é feita por um gerador dedicado, pois não é possível gerar atrasos na ordem de poucos microsegundos com confiabilidade apenas trabalhando com instruções no processador do CC2510. A transmissão de dados é feita colocando-se os bytes a serem transmitidos em um buffer e ativando-se uma *flag* no registrador dedicado a transferências UART, sendo que ao término da transmissão dos dados, esta *flag* é alterada novamente. A recepção dos dados pode acontecer a qualquer momento, o que torna necessário o uso de uma interrupção no evento de recepção de um byte na porta serial. Assim, para realizar a recepção, basta ativar esta interrupção e observar o buffer de recepção de dados, que vai sendo preenchido com os bytes recebidos. Esta biblioteca pode ser encontrada em 8.3.5.

4.5.3.3.3 Modem

Foi desenvolvida uma biblioteca para interação com o modem MC35i, que fornece as seguintes funcionalidades:

- Inicialização do modem
- Envio de comando
- Recepção de resposta
- Recepção de SMS não lido
- Apagar SMS

A biblioteca de comunicação serial descrita no item 4.5.3.3.24.5.3.2 foi utilizada pra realizar o envio e recepção de comandos. As funções implementadas fornecem todas as medidas necessárias para a comunicação com o modem, como: esperar 300ms entre envio de comandos, verificar se cada comando foi aceito ou se gerou

um erro no modem, aguardar o tempo correto para a inicialização do modem, entre outros. O diálogo com o modem ocorre por meio de comandos AT, um padrão bem estabelecido para comunicação com dispositivos de comunicação. A referência completa dos comandos que utilizamos pode ser encontrada em (SIEMENS AG). Esta biblioteca pode ser encontrada em 8.3.3.

4.5.3.3.4 Protocolo BBF

Em relação ao protocolo BBF, foi desenvolvido um arquivo que contém todas as definições sobre os tipos e estrutura das mensagens. Esse arquivo define o dado que cada byte de cada tipo de mensagem deve conter, assim como alguns valores importantes para o protocolo, como o tamanho máximo das mensagens e a numeração dos tipos das mensagens, por exemplo. Esse arquivo é utilizado pelo MR para formatar os dados recebidos do sistema da central administrativa de acordo com a estrutura definida pelo protocolo BBF. Já o MV utiliza esse arquivo para interpretar corretamente as mensagens recebidas do MR, extraindo as informações para exibi-las no display LCD. Esse arquivo pode ser encontrado em 8.2.4.

4.5.3.4 Definições gerais e Configuração

O arquivo "definições_gerais.h" possui algumas definições que são usadas tanto pelo software do MR quanto o do MV. Ele possui as importações das bibliotecas utilizadas por ambos os módulos. Além disso, ele contém definições de tempos de espera (tanto em modo ativo quanto em modo *sleep*, em que uma interrupção é gerada ao final da espera para alternar para o modo ativo novamente) e de potência da transmissão RF (utilizada para que os módulos ajustem sua potência conforme sua distância, evitando perda ou saturação de sinal). Esse arquivo pode ser encontrado em 8.2.3.

O arquivo "configurações.h" possui as configurações (na forma de macros) para a compilação do código. Através dele, pode-se escolher compilar um código para uso

final ou para depuração. A diferença é que no modo depuração, os *power modes* são diferentes (o sistema deve ficar somente em *power mode* 1, senão o CC Debugger perde a referência ao código) e, em alguns trechos do programa, ele habilita a impressão de mensagens no Terminal IO da IDE. Além disso, nesse arquivo de configurações, é possível configurar se o código deve ser compilado para o MR ou MV. Isso é importante porque cada módulo pode utilizar bibliotecas diferentes e, dependendo do módulo para o qual o código deve ser compilado, a importação de algumas bibliotecas é desnecessária, economizando espaço da memória do microcontrolador. Dependendo dessa configuração, o programa principal (comum aos dois módulos), chamará a rotina do MV ou MR. Esse arquivo pode ser encontrado em 8.2.2.

4.5.3.5 Programa principal

O programa principal segue o seguinte fluxo:

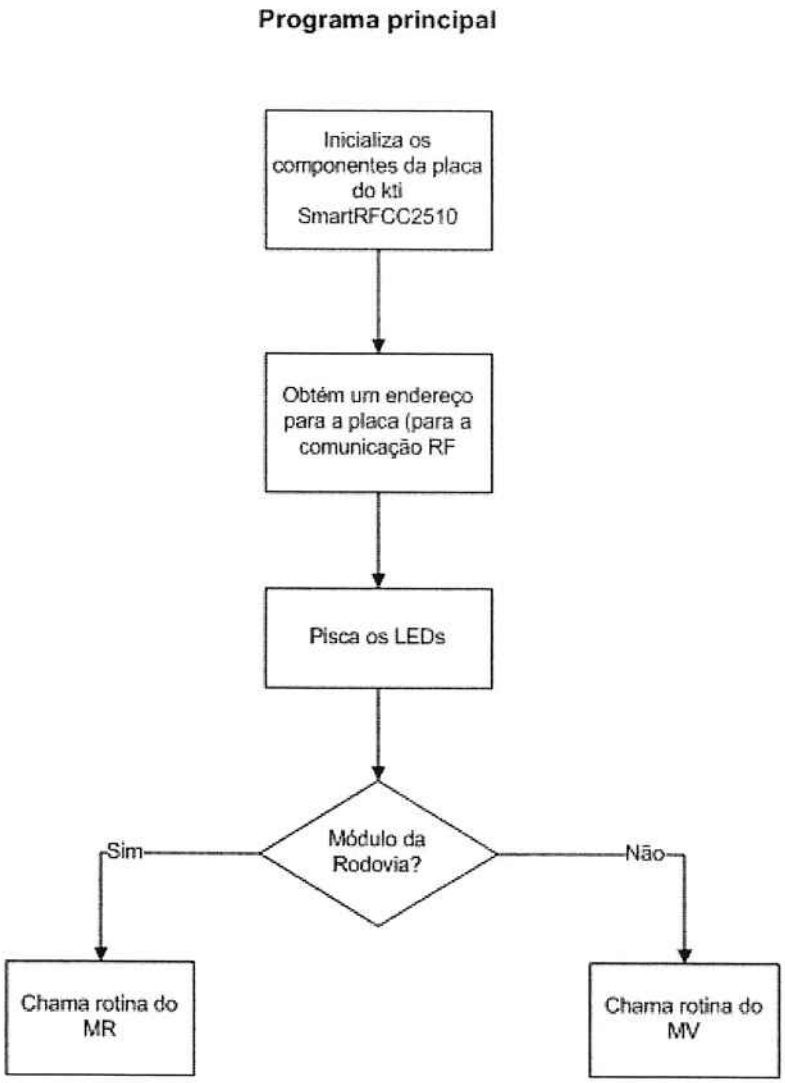


Figura 5: programa principal

O código-fonte do programa principal pode ser encontrado em 8.2.1.

4.5.4 Software do MR

O software do MR segue este fluxo:

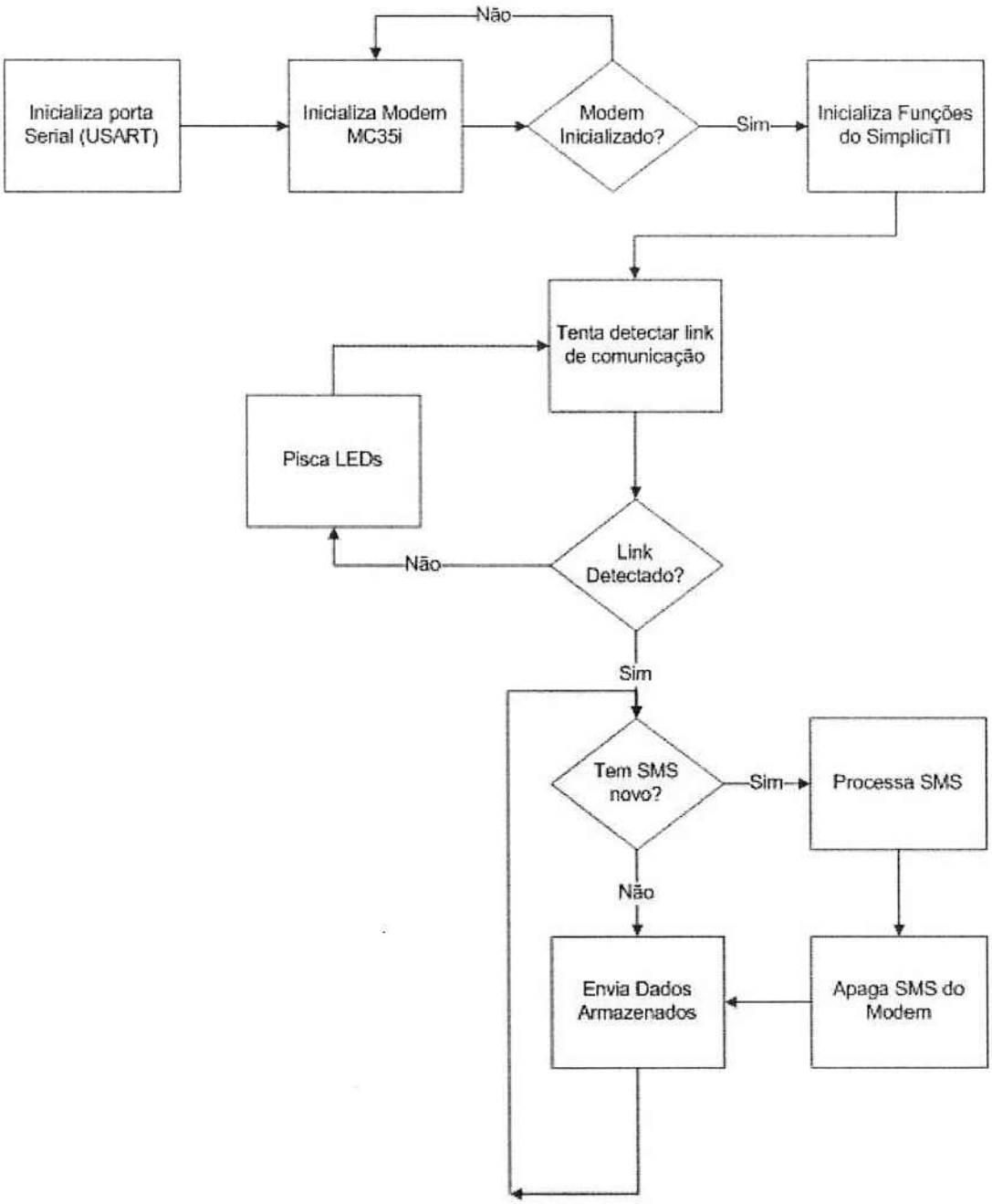


Figura 6: fluxograma do software do MR

Como pode ser visto no diagrama, a rotina do MR é relativamente simples: ao ligar o dispositivo, os componentes de comunicação serial, do modem e de comunicação por rádio-freqüência são inicializados. Após isto, o programa entra em um ciclo no qual tenta estabelecer um contato com algum MV. Uma vez estabelecido este contato, o software verifica se há novos dados recebidos via SMS. Se houver novos dados, ele empacota estes dados e transmite para o MV. Se não houver,

simplesmente transmite os dados armazenados. A freqüência com que a verificação de novas mensagens SMS é feita é de uma requisição por segundo.

A inicialização do módulo Serial (USART) pode ser resumida pelo seguinte fluxograma:

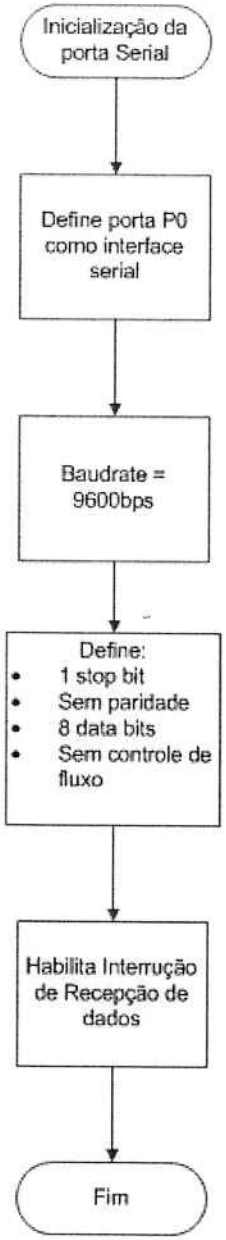


Figura 7: fluxograma de inicialização da serial

A recepção e o tratamento das mensagens SMS são feitos na rotina "processaSMS", que se encontra no código fonte do MR, que pode ser visto em 8.4. Uma mensagem SMS em UCS2 possui 70 caracteres ASCII, porém um pacote completo do MR ao

MV pode possuir 120 caracteres ASCII, por isso são enviados até dois SMSs e processados separadamente. O protocolo para envio dos dados via SMS está intimamente ligado ao protocolo de envio via rádio-freqüência, descrito em 4.4.1.2. Assim, existem cinco tipos de pacotes que podem ser enviados. O padrão adotado para transmitir estes dados via SMS é: utilizar um marcador de inicio de pacote ("$") e colocar os bytes subseqüentes após o marcador. Quando um novo marcador for encontrado pelo software do MR, ele verifica que tipo de pacote é e empacota adequadamente, segundo o protocolo BBF, para então transmitir para o MV.

Para exemplificar, será utilizado o pacote de Entrada/Saída de Rodovia, definido das seguintes maneiras no protocolo BBF e na mensagem SMS:

Entrada/Saída de rodovia - BBF

Byte	0	1	2	3	4	5 a 29
Descrição	Potência requisitada	Potência atual	Tipo -> 0	Código da rodovia (+sig)	Código rodovia (-sig)	Nome da rodovia

Entrada/Saída de rodovia - SMS

Byte	0	1	2	3	4 a 28
Descrição	$	Tipo -> 0	Código da rodovia (+sig)	Código rodovia (-sig)	Nome da rodovia

Nota-se que os bytes de "Potência requisitada" e "Potência atual" foram suprimidos da mensagem SMS, pois estes parâmetros são utilizados apenas para ajustar a potência do MR e do MV na comunicação em rádio-freqüência. No byte zero, pode-se observar o caractere iniciador de pacote. Os outros bytes são análogos aos definidos na tabela superior.

O MC35i Terminal não possui um microcontrolador programável. Ele se comunica através de comandos com o microcontrolador CC2510 do MR. Por isso, são

necessárias diversas rotinas para os comandos desejados, como, por exemplo, o comando de inicialização do modem ou inicialização do SMS.

Para ilustrar os comandos utilizados pelas rotinas do MR, são mostrados a seguir fluxogramas com cada rotina básica. Depois cada bloco básico é destrinchado para explicar os comandos utilizados.

Estes comandos são todos enviados na rotina de inicialização que está na biblioteca do modem, que pode ser encontrada em 8.3.3.

O diagrama geral de inicialização do MC35i é mostrado a seguir.

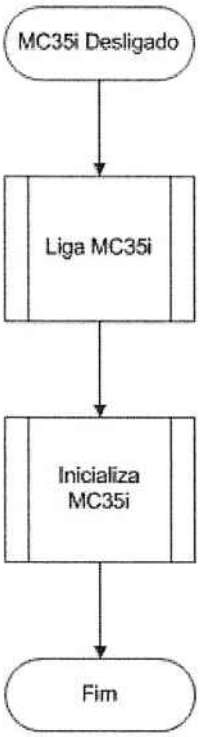


Figura 8: fluxograma de inicialização do MC35i

Inicialmente, o terminal está desligado. Então ele é ligado e a rotina de inicialização básica é executada.

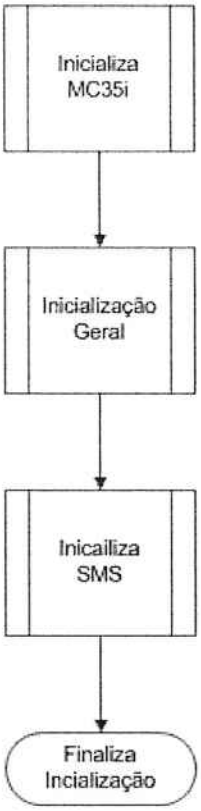


Figura 9: fluxograma de inicialização básica

A inicialização do MC35i, para a aplicação em questão, se resume a duas rotinas: inicialização de funções gerais e inicialização das funções de SMS.

Após o término da inicialização, o microcontrolador pode enviar comandos para o modem.

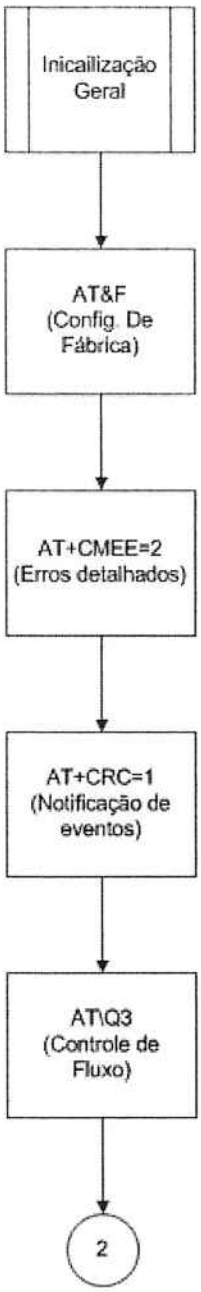


Figura 10: fluxograma da inicialização básica recomendada

Para a inicialização geral, começa-se enviando o comando AT&F para o modem, que restaura as configurações de fábrica do mesmo para os principais itens configuráveis, o que já é o suficiente para o que será necessário no protótipo, uma vez que não será utilizado nenhum comando avançado.

O segundo comando a ser executado é o AT+CMEE=2, que habilita as mensagens de erro a serem enviadas por *string*. Esse comando é opcional no projeto final, sendo usado apenas na hora de encontrar erros durante o desenvolvimento.

O próximo comando executado é o AT+CRC=1. Esse comando configura o MC35i a utilizar o formato estendido nas conexões recebidas. Ou seja, o terminal aceita conexões de voz, fax e GPRS.

O comando AT\Q3 configura o MC35i a usar o RTS/CTS *hardware flow control,* que é recomendado para ligações de voz, faz ou modo MUX. Este comando é opcional, pois neste projeto não há necessidade de ligações de voz ou de fax. Após esse comando, a inicialização básica recomendada termina.

O próximo estado a ser executado dentro da inicialização básica é a inicialização da rede, porém os comandos que desejamos executar já estão todos devidamente configurados com o comando AT&F, executado na seção anterior. Assim, descreveremos aqui os comandos essenciais de rede, mas sem necessidade de executá-los individualmente.

O primeiro comando é o AT+COPS=0, que habilita o terminal a selecionar automaticamente a rede a se conectar. Neste projeto, será utilizada a rede de celular da TIM. Com este comando, o MC35i fica procurando pela rede da operadora do SIM até encontrá-la, quando realiza a conexão.

O próximo comando é o AT+CREG=2, que habilita o módulo a reportar mudanças no status do registro de rede (caso o módulo perca conexão, por exemplo). Após esse comando, a inicialização de rede é terminada.

Após o término da inicialização básica, é possível enviar comandos de conexão para o módulo sem imprevistos.

Dentre todas as possíveis formas de transferência de dados do MC35i, será utilizado o SMS. O pacote de dados será enviado via SMS pelo módulo web ao MR.

O primeiro passo para receber os dados via SMS é fazer a inicialização da rotina SMS do MC35i.

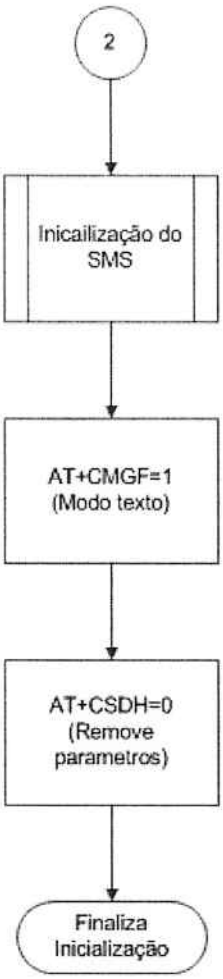


Figura 11: Inicialização do SMS

O primeiro comando a ser enviado ao MC35i é o AT+CMGF=1, que faz com que as mensagens SMS sejam retornadas em forma textual, e não numérica (modo PDU). Na seqüência, o comando AT+CSDH=0 remove parâmetros desnecessários da leitura da mensagem SMS, para reduzir o volume de dados transferidos do modem para o CC2510.

Outros parâmetros importantes do SMS, que foram configurados na inicialização geral do modem, com o comando AT&F, que definem algumas configurações padrões, são:

- AT+CSCS="GSM", que faz com que a mensagem seja recebida no alfabeto padrão GSM, para que seja mais fácil a leitura. Outra opção era converter a mensagem em caracteres hexadecimais.

- AT+CSMS=1, configura o serviço de mensagem para suportar diversos modos de mensagens.

Após as rotinas de inicialização citadas acima está terminada a inicialização completa do modem. A partir desse momento, o modem fica esperando chegar um SMS. Quando chega uma mensagem SMS, é necessário repassá-la ao CC2510 para que ele a trate de modo adequado.

Figura 12: recepção de SMS

O microcontrolador envia um comando AT+CMGL="REC UNREAD" para obter a lista de todos os SMSs que não foram lidos. Se houver mensagens não lidas, esta é

tratada pelo software do MR, e então o SMS lido é apagado, com o comando AT+CMGD=[índice da mensagem] para liberar espaço no modem. Caso não haja mensagens novas, o processo se encerra.

Quando o CC2510 obtém a mensagem, codificada em UCS2, é necessário transformar seu valor para o equivalente em inteiro. Para isso, cada grupo de 4 bytes em UCS2 é transformado da maneira em que foi apresentado em 0 (no exemplo em que 41H era convertido em 30H 30H 34H 31H). O MR pega esses 4 bytes e retira os dois primeiros que sempre serão 0 (em ASCII 30H) e, para os dois últimos, subtrai 30H de cada um, multiplica o mais significativo por 10H e soma ao menos significativo, ou seja, (34H – 30H)*10H + (31H – 30H) = 41H. Depois disso, verifica se o pacote está na forma correta para ser enviada ao MV. Se estiver certo, o MR envia o pacote via RF ao MV, terminando o processo.

O código-fonte da rotina do MR pode ser visto em 8.3.1.

4.5.5 Software do MV

A rotina do MV segue o seguinte fluxograma:

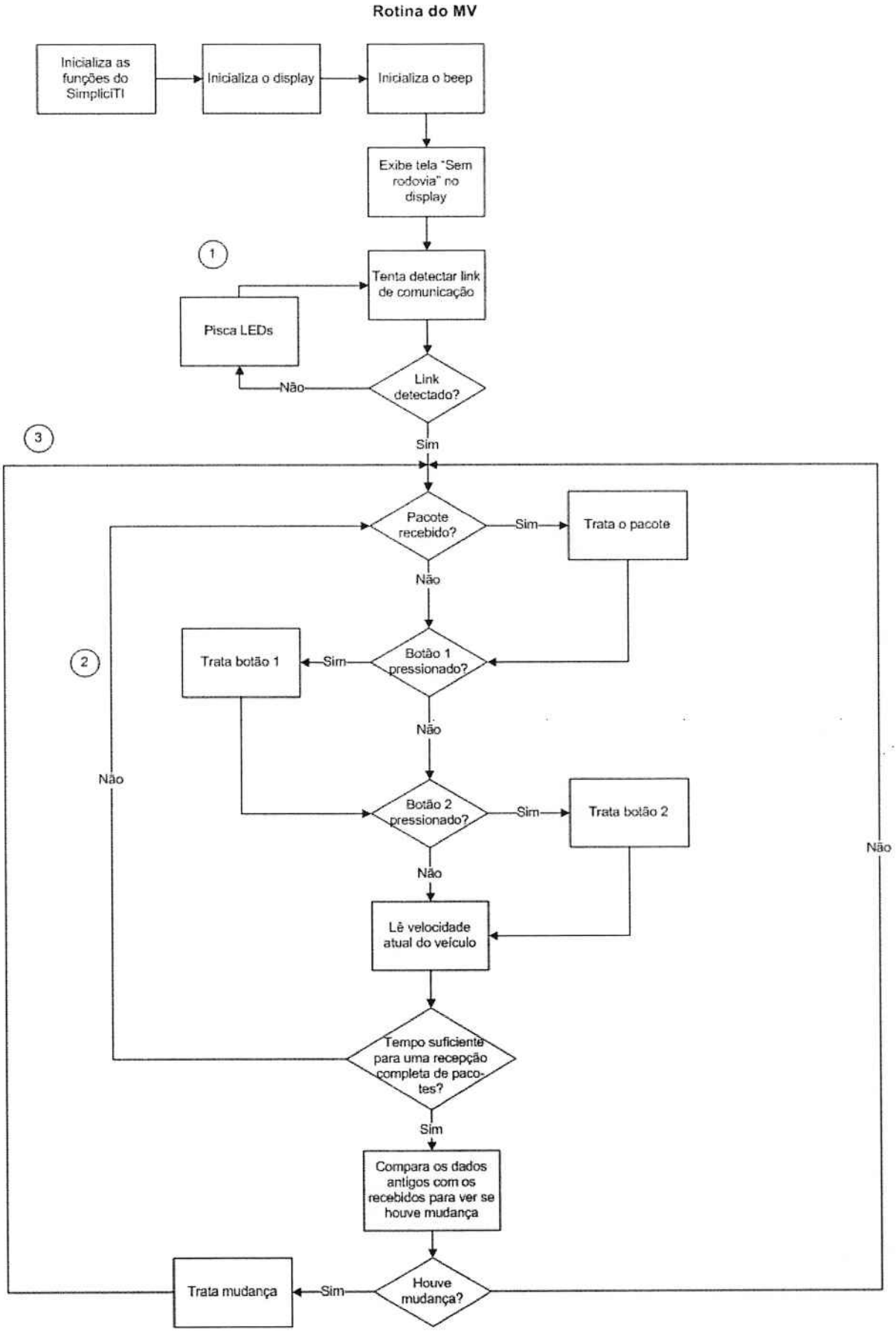


Figura 13: rotina do módulo veicular

Como pode ser visto no fluxograma apresentado, há três *loops* nessa rotina, identificados pelos números dentro dos círculos. O primeiro deles está relacionado ao link de comunicação RF. A rotina fica escutando o meio de comunicação na tentativa de fechar um link com o transmissor. Enquanto isso não ocorre, os LEDs permanecem piscando, indicando que o módulo está procurando um link. A rotina somente sairá desse *loop* quando o link entre o MR e MV for fechado.

O segundo *loop* é onde a rotina executa as funções de recepção de pacotes do MR, tratamento de botões e leitura de velocidade.

Inicialmente, a rotina verifica se houve a recepção de um novo pacote transmitido pelo MR. Se houver, o pacote será processado e algumas ações serão tomadas:

- Verificação da potência de transmissão: a rotina verifica a potência requisitada pelo MR e ajusta a potência atual do MV, para evitar perda ou saturação do sinal. Além disso, calcula a potência que o MV requisitará para o MR.
- Classifica a mensagem recebida de acordo com os tipos definidos pelo protocolo BBF e sobrescreve a mensagem anterior do mesmo tipo com os dados da mensagem recebida
- Atualiza o vetor de mudanças, indicando se houve uma mudança para o tipo de mensagem recebida.
- Monta uma mensagem de ACK contendo a potência requisitada ao MR e a potência atual do MV e a transmite ao MR.

Após a rotina executar essas ações (ou caso não tenha recebido um pacote), ela verificará se algum botão foi pressionado. A verificação do botão 1 é feita primeiramente. Caso ele tenha sido pressionado, a resposta do sistema depende de seu estado atual. Se o sistema não tiver informações da rodovia (ou porque não entrou em nenhuma rodovia que transmita informações ou porque saiu da rodovia em que estava), nada acontece. Se o sistema estiver em modo de exibição das informações da rodovia para o motorista, ocorrerá a troca de tela para que seja exibido outro tipo de informação (por exemplo, troca de tela da velocidade para a tela que informa os buracos). Se o sistema estiver em modo de configuração, a ação relacionada à tela de configuração que está sendo exibida será efetuada (por exemplo, o desligamento/ligamento do *beep* na tela de configuração do mesmo).

A próxima verificação a ser feita é sobre o botão 2. Caso ele tenha sido pressionado, a resposta do sistema depende de seu estado atual. Se o sistema não tiver informações da rodovia, nada acontece. Se o sistema estiver em modo de exibição das informações da rodovia para o motorista, ele entrará no modo de configuração e exibirá a primeira tela desse modo. Caso ele já esteja no modo de configuração, a tela de configuração será trocada para a próxima ou, se a tela de configuração for a última, o sistema voltará para o modo de exibição de informações ao motorista.

Após a verificação dos botões, a rotina faz a leitura da velocidade atual do veículo. Que no protótipo foi inserido uma variável com a velocidade atual do veículo, apenas para testes e comparação com a velocidade da rodovia. Se o MV tiver armazenado a informação sobre a velocidade máxima do trecho em sua memória, ele faz a comparação entre a velocidade atual de veículo e a velocidade máxima permitida e, caso o veículo esteja acima do limite, um alerta visual é emitido através do LED para avisar o motorista dessa situação. Se o MV não tiver armazenado nenhuma informação sobre a velocidade máxima do trecho (ou porque não entrou na rodovia ou porque já saiu dela), a velocidade atual do veículo é exibida no display apenas.

A condição de saída do segundo *loop* é o término do tempo necessário para que o MV possa fazer uma recepção completa de pacotes, ou seja, possa receber os pacotes de todos os tipos de mensagem especificados no protocolo BBF. Esse tempo é necessário para que o MV possa receber todas as informações do trecho da rodovia, analisar em quais informações ocorreram mudanças e qual a prioridade de exibição dessas informações.

O *loop* 2 está no interior do *loop* 3. Após o término do tempo necessário para a recepção total dos pacotes, a rotina sai do *loop* 2 e continua executando os comandos do *loop* 3. O sistema verifica se houve mudanças nas informações sobre a rodovia. O vetor de mudanças é analisado e, caso tenham ocorridas alterações nas informações, a rotina verifica em quais tipos de mensagem essas mudanças ocorreram. Como o display LCD só apresenta um tipo de informação por vez devido ao pequeno número de caracteres que podem ser exibidos no mesmo, a rotina estabelece uma ordem de prioridade de exibição dos tipos das novas informações. Assim, entre os tipos de informação que sofreram alteração, a rotina verifica qual é o mais prioritário e o exibe no display LCD sem que o motorista precise apertar o

botão 1, piscando o LED para avisá-lo sobre a disponibilidade de novas informações. A troca de tela é feita automaticamente para que o motorista veja a nova informação sem precisar ficar procurando qual tipo de informação foi alterada. Por esse motivo que o sistema espera o tempo necessário para a recepção total dos pacotes. Caso não houvesse esse tempo, cada pacote recebido que tivesse alguma mudança de informação causaria a troca automática de tela, deixando o motorista confuso ao olhar para o display trocando rapidamente suas telas.

O *loop* 3 não tem condição de saída, sendo executado até que o sistema seja desligado ou reiniciado.

O código-fonte da rotina do MV pode ser visto em 8.4.1.

4.5.6 Sistema Web

O sistema Web tem como principal objetivo fazer a interface entre a concessionária da rodovia e o MR, enviando as informações relevantes via GSM.

O sistema Web foi projetado de forma a fornecer uma boa usabilidade, com uma interface simples e intuitiva. Foi desenvolvido no framework ASP.NET 3.5 com a linguagem C#, utilizando o Visual Studio 2008 como ambiente de desenvolvimento. Foi utilizado AJAX para melhorar a dinâmica da interface, fazendo com que não seja necessária a troca de páginas a todo o momento.

A interface do sistema possui os campos a serem preenchidos para montar o pacote a ser enviado ao MR. É feita a verificação de validade em cada campo, evitando que sejam enviados dados inconsistentes ao módulo da rodovia.

Para enviar os dados, foi escolhido o SMS por apresentar baixo custo e tamanho adequado às necessidades do sistema. Usando a API do Skype (Skype4COM), o sistema autentica a conta do Skype e envia o SMS aos números inseridos no formulário da interface web. Ao enviar os dados, o sistema exibe uma mensagem de confirmação ou de erro para informar o usuário sobre o resultado da operação.

5 Considerações finais

Nesta sessão serão apresentadas as considerações finais do projeto.

5.1 Cumprimento dos objetivos

O projeto completo não pode ser inteiramente cumprido pelo prazo imposto pelo projeto de formatura. Porém todas as funções que foram incluídas no escopo do protótipo foram criadas. O protótipo contém grande parte das funções do projeto final, as funções mais importantes foram implementadas com sucesso e os testes foram positivos. Porém não foi possível realizar uma parte importante do projeto que era testar o protótipo em um veículo pegando a velocidade dele e comparando com a velocidade máxima transmitida, foi feito apenas uma simulação com a velocidade do veículo em código dentro do módulo veicular.

5.2 Contribuições do trabalho

Neste trabalho foi comprovada a possibilidade técnica da implantação de um sistema de controle de velocidade e informações rodoviárias. Apesar de não ter sido concluído por falta de tempo, com a implementação das características descritas na sessão 5.4 (Trabalhos futuros) o sistema se torna de grande utilidade no aumento da segurança ao motorista.

Nenhum teste foi feito para checar a eficácia do sistema quanto a diminuição de números de acidentes quando utilizado o sistema proposto.

5.3 Conclusões

Este trabalho apresentou o estudo e desenvolvimento de um sistema para ser implantado nas rodovias brasileiras visando a diminuição do número de acidentes e mortes por colisão. O trabalho foi um esforço conjunto de montar o hardware necessário para atender às necessidades do sistema e implementar o código na linguagem C para os microprocessadores do módulo rodoviário e veicular. O estudo das tecnologias necessárias para realizar o proposto pelo trabalho foi a primeira etapa realizada. Depois foi necessário o estudo dos componentes que atendessem as necessidades impostas pelas tecnologias que usaríamos, como o CC2510 que é um microcontrolador com comunicação RF embutido. Após a obtenção de todo o hardware e de feita a arquitetura do sistema foi feita a codificação dos softwares dos microcontroladores.

A segurança dos passageiros foi o principal motivo da proposta desse trabalho. Com base nisso, foi projetado um sistema que transmite informações para o motorista se prevenir de possíveis incidentes durante a viagem que possam causar acidentes, ferindo ou tirando a vida de pessoas nas rodovias. Tendo isso em vista, o sistema transmite informações como buracos, acidentes, obras, trânsito, condições do tempo e velocidade máxima do trecho aos motoristas para que eles sejam alertados em casos de adversidades.

Neste trabalho obteve-se como resultado o protótipo completo como especificado nesta monografia. Com a construção total do módulo veicular, rodoviário e sistema web e a especificação completa nesta monografia. Os testes feitos mostraram o funcionamento do sistema completo, com as informações sendo mostradas no display de LCD de forma correta.

5.4 Trabalhos futuros

Além da monitoração da velocidade instantânea do veículo que trafega pela rodovia, para compará-la com a velocidade máxima permitida no trecho e alertar o motorista caso sua velocidade seja superior ao limite permitido, uma possível funcionalidade nova para o sistema é a integração com o *Cruise Control* (sistema que atua na velocidade do veículo, mantendo-a em um valor estabelecido pelo motorista) de um veículo que o possua. Como visto nos itens anteriores, a velocidade máxima ao longo de uma rodovia pode variar de acordo com a localidade do trecho ou das condições da pista. Como o *Cruise Control* necessita que o motorista regule a velocidade a ser mantida, seu uso pode não ser conveniente. Assim, o MV atuaria diretamente no *Cruise Control* do veículo, enviando informações sobre o limite de velocidade do trecho e ajustando a velocidade do veículo a esse limite, sem que o motorista precisasse fazer a regulagem manual.

Poderia ser integrado também um módulo GPS ao MV, a fim de calcular a posição do veículo na rodovia. Essa posição seria exibida em um mapa no display do MV (que necessitaria ser um display mais sofisticado do que o usado no protótipo) para que o motorista pudesse ver sua posição em relação aos problemas enviados pelo MR ao seu MV, como os buracos, obras, acidentes. Assim, ele poderia ser alertado ao se aproximar desses problemas para tomar uma ação preventiva. Além disso, essa informação poderia ser usada para avisá-lo sobre a aproximação de saídas para rodovias alternativas em caso de trânsito.

Outra funcionalidade que pode facilmente ser adicionada ao sistema é o armazenamento do histórico das informações sobre a condução do veículo pelo motorista. Essas informações seriam armazenadas em cada MV e poderiam ser transmitidas aos MR ou inspecionadas em alguns estabelecimentos de empresas interessadas em obter o perfil do motorista para realizar promoções. O MV poderia armazenar a porcentagem do tempo em que o motorista conduziu o veículo na velocidade sugerida pelo MR para o trecho e o número de vezes em que ele excedeu essa velocidade. Essa informação é útil às seguradoras, por exemplo, que poderiam incentivar a direção segura fornecendo descontos e bônus aos motoristas com perfil de direção defensiva. Nesse caso, tanto a seguradora (que incentivaria um comportamento que leva à redução do número de acidentes e, conseqüentemente, redução dos seus gastos) quanto os motoristas (que receberiam descontos maiores no seguro de seus veículos) seriam beneficiados.

Outras possíveis melhorias poderiam estar relacionadas ao aumento de funcionalidades do MV para que haja outras finalidades além da segurança como, por exemplo, adicionar a funcionalidade de controle de pedágio (semelhante ao Sem Parar) ou de navegador veicular (funcionalidade possibilitada pela adição do módulo GPS, como foi citado anteriormente).

6 Referências

ESCOLA POLITÉCNICA – DIVISÃO DE BIBLIOTECA. Diretrizes para monografia. Disponivel em: <http://www.poli.usp.br/media/biblioteca/diretrizes3.pdf>. Acesso em: 21 Ago 2010.

PEACOCK, C. Interfacing the serial / RS-232 port. Disponivel em: <http://www.beyondlogic.org/serial/serial.htm>. Acesso em: 21 Ago 2010.

SIEMENS AG. Application Developer's Guide – Application Note 24. Disponivel em: <http://www.mcs-nl.com/files/nl/m2m/siemens/WM_AN_24_Dev_Guide.pdf>. Acesso em: 21 Ago 2010.

SIEMENS AG. MC35i AT Command Set. Disponivel em: <http://read.pudn.com/downloads80/doc/comm/311558/mc35i_atc_v0103.pdf>. Acesso em: 21 Ago 2010.

SIEMENS AG. MC35i Terminal Siemens Cellular Engine Hardware Interface Description. Disponivel em: <http://www.mobilesolutions.ch/fileadmin/user_upload/pdf/wm/siemens/MC35i_Terminal_User_guide.pdf>. Acesso em: 21 Ago 2010.

SIEMENS AG. MC35i Terminal User Guide. Disponivel em: <http://www.mobilesolutions.ch/fileadmin/user_upload/pdf/wm/siemens/MC35i_Terminal_User_guide.pdf>. Acesso em: 21 Ago 2010.

TEXAS INSTRUMENTS CORPORATED. CC2510Fx / CC2511Fx. Disponivel em: <http://focus.ti.com/lit/ds/symlink/cc2510f32.pdf>. Acesso em: 21 Ago 2010.

TEXAS INSTRUMENTS CORPORATED. Design Note DN112. Disponivel em: <http://focus.ti.com/lit/an/swra222b/swra222b.pdf>. Acesso em: 21 Ago 2010.

TEXAS INSTRUMENTS CORPORATED. MAX232, MAX232I, dual EIA-232 drivers/receivers. Disponivel em: <http://focus.ti.com/lit/ds/symlink/max232.pdf>. Acesso em: 21 Ago 2010.

TEXAS INSTRUMENTS CORPORATED. SimpliciTI™ - RF Made Easy. Disponivel em: <http://www.ti.com/corp/docs/landing/simpliciTI/index.htm?DCMP=hpa_rf_general&HQS=NotApplicable+OT+simpliciti>. Acesso em: 22 Ago 2010.

WINSTAR DISPLAY. WH2002A Character 20x2. Disponivel em: <http://www.crestcomponents.com.au/images/LCD/WH2002A.pdf>. Acesso em: 25 Ago 2010.

7 Bibliografia

ESCOLA POLITÉCNICA – DIVISÃO DE BIBLIOTECA. Diretrizes para monografia. Disponivel em: <http://www.poli.usp.br/media/biblioteca/diretrizes3.pdf>. Acesso em: 21 Ago 2010.

PEACOCK, C. Interfacing the serial / RS-232 port. Disponivel em: <http://www.beyondlogic.org/serial/serial.htm>. Acesso em: 21 Ago 2010.

SIEMENS AG. Application Developer's Guide – Application Note 24. Disponivel em: <http://www.mcs-nl.com/files/nl/m2m/siemens/WM_AN_24_Dev_Guide.pdf>. Acesso em: 21 Ago 2010.

SIEMENS AG. MC35i AT Command Set. Disponivel em: <http://read.pudn.com/downloads80/doc/comm/311558/mc35i_atc_v0103.pdf>. Acesso em: 21 Ago 2010.

SIEMENS AG. MC35i Terminal Siemens Cellular Engine Hardware Interface Description. Disponivel em: <http://www.mobilesolutions.ch/fileadmin/user_upload/pdf/wm/siemens/MC35i_Terminal_User_guide.pdf>. Acesso em: 21 Ago 2010.

SIEMENS AG. MC35i Terminal User Guide. Disponivel em: <http://www.mobilesolutions.ch/fileadmin/user_upload/pdf/wm/siemens/MC35i_Terminal_User_guide.pdf>. Acesso em: 21 Ago 2010.

TEXAS INSTRUMENTS CORPORATED. CC2510Fx / CC2511Fx. Disponivel em: <http://focus.ti.com/lit/ds/symlink/cc2510f32.pdf>. Acesso em: 21 Ago 2010.

TEXAS INSTRUMENTS CORPORATED. Design Note DN112. Disponivel em: <http://focus.ti.com/lit/an/swra222b/swra222b.pdf>. Acesso em: 21 Ago 2010.

TEXAS INSTRUMENTS CORPORATED. MAX232, MAX232I, dual EIA-232 drivers/receivers. Disponivel em: <http://focus.ti.com/lit/ds/symlink/max232.pdf>. Acesso em: 21 Ago 2010.

TEXAS INSTRUMENTS CORPORATED. SimpliciTI™ - RF Made Easy. Disponivel em: <http://www.ti.com/corp/docs/landing/simpliciTI/index.htm?DCMP=hpa_rf_general&HQS=NotApplicable+OT+simpliciti>. Acesso em: 22 Ago 2010.

WINSTAR DISPLAY. WH2002A Character 20x2. Disponivel em: <http://www.crestcomponents.com.au/images/LCD/WH2002A.pdf>. Acesso em: 25 Ago 2010.

8 Apêndice

Nesta sessão são apresentados os códigos produzidos no trabalho.

8.1 Código do Sistema Web

8.1.1 Design

Configurador de Módulo Rodoviário

Configurações:
DDD: Telefone: Digite os números, separados por vírgula se tiver mais de um.

Entrada/Saída
Código da Rodovia:
Nome da Rodovia:

Velocidade
Velocidade Máxima:

Buracos
Número de Buracos (0-3): 1
Buraco 1: Km Inicial: Km Final:
Faixa Inicial: Faixa Final:

Acidentes/Obras
Número de Ocorrências (0-3): 1
Ocorrência 1: Tipo: Obra Acidente Km Inicial: Km Final:
Faixa Inicial: Faixa Final:

Trânsito
Tipo: Com Alerta Sem Alerta
Km Inicial: Km Final:
Km Alternativa: Nome da Rodovia Alternativa:

Tempo
Tipo: Com Alerta Sem Alerta
Descrição do Tempo:

Enviar Dados

8.1.2 Lógica

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using SKYPE4COMLib;
using System.Text;

public partial class Configurador : System.Web.UI.Page
{
    private enum Tipos
    {
        Entrada_Saida = 0,
        Velocidade = 1,
        Buracos = 2,
        Acidentes = 3,
        Transito_Alerta = 4,
        Transito_Sem_Alerta = 5,
        Tempo_Alerta = 6,
        Tempo_Sem_Alerta = 7
    }
    List<byte> payload;

    #region Skype Objects
    private static Skype skype;
    private static Command command;

    #endregion

    private const Int32 MARCADOR = 0xFF;

    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            toggleBuracos(false, 3);
            toggleOcorrencias(false, 3);
        }

        verificaBuracos();
        verificaOcorrencias();

    }

    #region [ Visibilidade ]

    protected void verificaBuracos()
    {
        int numBuracos = 0;
```

```
        try
        {
            numBuracos = Convert.ToInt32(txtNumBuracos.Text);
        }
        catch (Exception)
        {
        }

        toggleBuracos(true, numBuracos);
    }

    protected void verificaOcorrencias()
    {
        int numOcorrencias = 0;
        try
        {
            numOcorrencias = Convert.ToInt32(txtNumAcidentes.Text);
        }
        catch (Exception)
        {
        }

        toggleOcorrencias(true, numOcorrencias);
    }

    protected void toggleBuracos(bool toggle, int numBuracos)
    {
        if (numBuracos == 1)
        {
            divBuraco1.Visible = toggle;
            divBuraco2.Visible = !toggle;
            divBuraco3.Visible = !toggle;

        }
        else if (numBuracos == 2)
        {
            divBuraco1.Visible = toggle;
            divBuraco2.Visible = toggle;
            divBuraco3.Visible = !toggle;
        }
        else if (numBuracos == 3)
        {
            divBuraco1.Visible = toggle;
            divBuraco2.Visible = toggle;
            divBuraco3.Visible = toggle;
        }
        else
            toggleBuracos(false, 3);

    }

    protected void toggleOcorrencias(bool toggle, int numOcorrencias)
    {
        if (numOcorrencias == 1)
        {
            divOcorrencia1.Visible = toggle;
            divOcorrencia2.Visible = !toggle;
            divOcorrencia3.Visible = !toggle;

        }
        else if (numOcorrencias == 2)
```

```
        {
            divOcorrencia1.Visible = toggle;
            divOcorrencia2.Visible = toggle;
            divOcorrencia3.Visible = !toggle;
        }
        else if (numOcorrencias == 3)
        {
            divOcorrencia1.Visible = toggle;
            divOcorrencia2.Visible = toggle;
            divOcorrencia3.Visible = toggle;
        }
        else
            toggleOcorrencias(false, 3);

    }
    #endregion

    #region [ Eventos ]

    protected void btnSubmit_Click(Object sender, EventArgs e)
    {
        try
        {
            MontaPacoteSMS();
        }
        catch (Exception ex)
        {
            ClientScript.RegisterStartupScript(this.GetType(), "myalert",
"alert('Erro ao processar dados: " + ex.Message + "');", true);
            return;
        }

        try
        {
            EnviaPacoteSMS();
        }
        catch (Exception ex)
        {
            ClientScript.RegisterStartupScript(this.GetType(), "myalert",
"alert('Erro ao enviar SMS: " + ex.Message + "');", true);
            return;
        }

        ClientScript.RegisterStartupScript(this.GetType(), "myalert",
"alert('Dados enviados com sucesso!');", true);
    }

    #endregion

    protected void MontaPacoteSMS()
    {
        payload = new List<byte>();

        #region Variaveis
        //bool enviar = false;

        //ES
        byte[] codRodovia;
        String NomeRodovia;
        //Buracos
        Int32 numeroBuracos;
```

```
        //Acidentes
        Int32 numeroAcidentes;
        #endregion

        #region Entrada/Saida
        //Entrada/Saída
        try
        {
            //Verifica se uma rodovia foi especificada:
            if (String.IsNullOrEmpty(txtEsCodRodovia.Text))
            {
                throw new Exception("É necessário informar uma rodovia.");
            }

            codRodovia =
BitConverter.GetBytes(Convert.ToInt32(txtEsCodRodovia.Text));

            payload.Add(MARCADOR);//Inicio de pacote

payload.Add(BitConverter.GetBytes((int)Tipos.Entrada_Saida)[0]);//Tipo do
pacote
            payload.Add(codRodovia[1]); //Bit high
            payload.Add(codRodovia[0]); //Bit low
            NomeRodovia = txtEsNomRodovia.Text;
            NomeRodovia = NomeRodovia.Length > 24 ?
NomeRodovia.Substring(0, 24) : NomeRodovia; //Adequa o tamanho
            for (int i = 0; i < NomeRodovia.Length; i++)
            {
                payload.Add((byte)NomeRodovia[i]);
            }
            payload.Add((byte)'\0'); //Terminador da string

        }
        catch (Exception)
        {
            throw;
        }
        #endregion

        #region Velocidade
        //Velocidade
        try
        {
            if (!String.IsNullOrEmpty(txtVelMax.Text))
            {
                payload.Add(MARCADOR);

payload.Add(BitConverter.GetBytes((int)Tipos.Velocidade)[0]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtVelMax.Text)))[1]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtVelMax.Text)))[0]);

            }
        }
        catch (Exception)
        {
            throw;
        }
        #endregion
```

```
        #region Buracos
        //Buracos
        try
        {
            if (!String.IsNullOrEmpty(txtNumBuracos.Text))
            {
                payload.Add(MARCADOR);
                payload.Add(BitConverter.GetBytes((int)Tipos.Buracos)[0]);
                numeroBuracos = Convert.ToInt32(txtNumBuracos.Text);
                payload.Add((BitConverter.GetBytes(numeroBuracos))[0]);

                if (numeroBuracos == 1 || numeroBuracos == 2 ||
numeroBuracos == 3)
                {
                    //Km Inicial

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco1KmInicial.Text
)))[1]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco1KmInicial.Text
)))[0]);

                    //Km Final

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco1KmFinal.Text))
)[1]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco1KmFinal.Text))
)[0]);

                    //Faixas

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco1FaixaInicial.T
ext)))[0]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco1FaixaFinal.Tex
t)))[0]);

                    if (numeroBuracos == 2 || numeroBuracos == 3)
                    {
                        //Km Inicial

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco2KmInicial.Text
)))[1]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco2KmInicial.Text
)))[0]);

                        //Km Final

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco2KmFinal.Text))
)[1]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco2KmFinal.Text))
)[0]);

                        //Faixas

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco2FaixaInicial.T
ext)))[0]);
```

```
payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco2FaixaFinal.Tex
t)))[0]);
                        if (numeroBuracos == 3)
                        {
                            //Km Inicial

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco3KmInicial.Text
)))[1]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco3KmInicial.Text
)))[0]);

                            //Km Final

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco3KmFinal.Text))
)[1]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco3KmFinal.Text))
)[0]);

                            //Faixas

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco3FaixaInicial.T
ext)))[0]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtBuraco3FaixaFinal.Tex
t)))[0]);
                        }
                    }
                }
            }
        }
        catch (Exception)
        {
            throw;
        }
        #endregion

        #region Acidentes/Obras
        try
        {
            if (!String.IsNullOrEmpty(txtNumAcidentes.Text))
            {
                payload.Add(MARCADOR);

payload.Add(BitConverter.GetBytes((int)Tipos.Acidentes)[0]);
                numeroAcidentes = Convert.ToInt32(txtNumAcidentes.Text);
                payload.Add((BitConverter.GetBytes(numeroAcidentes))[0]);
                if (numeroAcidentes == 1 || numeroAcidentes == 2 ||
numeroAcidentes == 3)
                {
                    //Tipo

payload.Add(BitConverter.GetBytes(Convert.ToInt32(rblOcorrencia1.SelectedVa
lue))[0]);

                    //Km Inicial

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmInicial1.Text)))
[1]);
```

```
payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmInicial1.Text)))
[0]);

                    //Km Final

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmFinal1.Text)))[1
]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmFinal1.Text)))[0
]);

                    //Faixas

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoFaixaInicial1.Text
)))[0]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoFaixaFinal1.Text))
)[0]);

                    if (numeroAcidentes == 2 || numeroAcidentes == 3)
                    {
                        //Tipo

payload.Add(BitConverter.GetBytes(Convert.ToInt32(rblOcorrencia2.SelectedVa
lue))[0]);

                        //Km Inicial

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmInicial2.Text)))
[1]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmInicial2.Text)))
[0]);

                        //Km Final

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmFinal2.Text)))[1
]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmFinal2.Text)))[0
]);

                        //Faixas

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoFaixaInicial2.Text
)))[0]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoFaixaFinal2)))[0])
;

                        if (numeroAcidentes == 3)
                        {
                            //Tipo

payload.Add(BitConverter.GetBytes(Convert.ToInt32(rblOcorrencia3.SelectedVa
lue))[0]);

                            //Km Inicial
```

```
payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmInicial3.Text)))
[1]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmInicial3.Text)))
[0]);

                                //Km Final

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmFinal3.Text)))[1
]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoKmFinal3.Text)))[0
]);

                                //Faixas

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoFaixaInicial3.Text
)))[0]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtOcoFaixaFinal3.Text))
)[0]);

                            }
                        }
                    }
                }

            }
            catch (Exception)
            {
                throw;
            }

            #endregion

            #region Trânsito

            try
            {
                if (!String.IsNullOrEmpty(rblTransitoAlerta.SelectedValue))
                {
                    payload.Add(MARCADOR);

payload.Add(BitConverter.GetBytes(Convert.ToInt32(rblTransitoAlerta.Selecte
dValue))[0]);

                    //Km Inicial

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtTransitoKmInicial.Tex
t)))[1]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtTransitoKmInicial.Tex
t)))[0]);


                    //Km Final

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtTransitoKmFinal.Text)
))[1]);
```

```
payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtTransitoKmFinal.Text)
))[0]);


                //Km Alternativa

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtTransitoKmAlternativa
.Text)))[1]);

payload.Add((BitConverter.GetBytes(Convert.ToInt32(txtTransitoKmAlternativa
.Text)))[0]);

            .   //Nome Alternativa
                NomeRodovia = txtEsNomRodovia.Text;
                NomeRodovia = NomeRodovia.Length > 20 ?
NomeRodovia.Substring(0, 20) : NomeRodovia; //Adequa o tamanho
                for (int i = 0; i < NomeRodovia.Length; i++)
                {
                    payload.Add((byte)NomeRodovia[i]);
                }
                payload.Add((byte)'\0'); //Terminador da string
            }
        }
        catch (Exception)
        {
            throw;
        }


        #endregion

        #region Tempo
        string descTempo;

        try
        {
            if (!String.IsNullOrEmpty(rblTempoAlerta.SelectedValue))
            {
                payload.Add(MARCADOR);

payload.Add(BitConverter.GetBytes(Convert.ToInt32(rblTempoAlerta.SelectedVa
lue))[0]);
                //Descricao do Tempo
                descTempo = txtEsNomRodovia.Text;
                descTempo = descTempo.Length > 26 ? descTempo.Substring(0,
26) : descTempo; //Adequa o tamanho
                for (int i = 0; i < descTempo.Length; i++)
                {
                    payload.Add((byte)descTempo[i]);
                }
                payload.Add((byte)'\0'); //Terminador da string

            }
        }
        catch (Exception)
        {
            throw;
        }

        #endregion
```

```
        payload.Add(MARCADOR);
        payload.Add(MARCADOR);//Marca o fim do payload
    }

    protected void EnviaPacoteSMS()
    {
        skype = new Skype();

        skype.Attach(8, true);

        string DDI = "+55";
        string DDD = txtDDD.Text;
        string[] telefones = txtTelefones.Text.Split(',');
        StringBuilder numero_envio = new StringBuilder();
        string mensagem;
        string numero_resposta;

        foreach (string telefone in telefones)
        {
            numero_envio.Append(DDI + DDD + telefone.Trim());
            numero_envio.Append(',');
        }
        numero_envio.Remove(numero_envio.Length - 1, 1);

        numero_resposta = "";
        mensagem = new String((payload.ConvertAll(new Converter<byte,
char>(Byte2Char))).ToArray());

        skype.SendSms(numero_envio.ToString(), mensagem, numero_resposta);
    }

   // private void TesteSkype()
   // {
   //     Skype skype = new Skype();

   //     skype.Attach(8, true);

   //     char[] teste = new char[12];
   //     for (int i = 0; i <= 10; i++)
   //         teste[i] = (char)(i + 127);
   //     string teste_string = new String(teste);
   //  skype.SendSms("+551172303175", teste_string, "");
   ////     skype.SendMessage("bruno.igor.rodrigues.domingues",
teste_string);
   // }

    private static char Byte2Char(byte b)
    {
        return (char)b;
    }
}
```

8.2 Sistema Geral

8.2.1 Main.c

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

Filename:   main.c

Description:      Este arquivo contém a função main utilizada tanto no MR
(Módulo
                da Rodovia) quanto no MV (Módulo do Veículo).

Autores:          Bruno Nigro
                  Fernando Nobre
                  Bruno Igor Rodrigues Domingues

*****************************************************************************
********
* INCLUDES
*/
#include "configuracoes.h"
#include "definicoes_gerais.h"
#if MR_MV
#include "mod_rodovia.h"
#else
#include "mod_veiculo.h"
#endif


/*****************************************************************************
*********
* @fn          main
*
* @brief       This is the main entry of the SMPL link application. It sets
*              random addresses for the nodes, initalises and runs
*              MASTER and SLAVE tasks sequentially in an endless loop.
*
* @return      none
*/
void main (void)
{
  printf("Entrou no main.\n");
  BSP_Init();

  /* Create and set random address for this device. */
  addr_t lAddr;
  BSP_createRandomAddress(&lAddr);
  SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, &lAddr);

  /* Turn on LEDs indicating power on */
  BSP_TURN_ON_LED1();
  BSP_TURN_ON_LED2();

  BSP_SleepFor( _POWER_MODE_2, SLEEP_1_MS_RESOLUTION, 1000);

  BSP_TURN_OFF_LED1();
  BSP_TURN_OFF_LED2();

#if MR_MV
```

```
  modulo_rodovia();
#else
  modulo_veiculo();
#endif
  while (1);
}
```

8.2.2 Configuracoes.h

```
#define MR_MV                           1  /* MR_MV = 1 => Módulo da Rodovia
                                              MR_MV = 0 => Módulo do Veículo
*/
#define DEBUG_MODE                      1  /* DEBUG_MODE = 1 => Modo usado
para
                                              debug */

/* Define as configurações de Power Mode usadas pela aplicação em
diferentes
  modos de operação */
#if DEBUG_MODE
// Essa configuração deve ser usada com o debugger (que só funciona com
PM1) */
#define _POWER_MODE_1  POWER_MODE_1
#define _POWER_MODE_2  POWER_MODE_1
#define _POWER_MODE_3  POWER_MODE_1

#else
// Essa configuração deve ser usada em utilização normal
#define _POWER_MODE_1  POWER_MODE_1
#define _POWER_MODE_2  POWER_MODE_2
#define _POWER_MODE_3  POWER_MODE_3
#endif
```

8.2.3 Definicoes_gerais.h

```
#include <stdio.h>  /* Necessário para utlizar o printf() do debugger */
#include <string.h>
#include "bsp.h"
#include "mrfi.h"
#include "nwk_types.h"
#include "nwk_api.h"
#include "bsp_leds.h"
#include "bsp_buttons.h"
#include "bsp_extended.h"

/************************************************************************
*********
* CONSTANTS and DEFINITIONS
*/
#define SPIN_ABOUT_HALF_SECOND          NWK_DELAY(500)
#define SPIN_ABOUT_QUARTER_A_SECOND     NWK_DELAY(250)
#define SPIN_ABOUT_100_MS               NWK_DELAY(100)
#define SPIN_ABOUT_5_MS                 NWK_DELAY(5)

#define NUM_TX_RETRIES                  3

#define RSSI_UPPER_THRESHOLD            -40
#define RSSI_LOWER_THRESHOLD            -70

#define MINIMUM_OUTPUT_POWER            0
#define MEDIUM_OUTPUT_POWER             1
```

```
#define MAXIMUM_OUTPUT_POWER          2

#define SLEEP_31_25_US_RESOLUTION     0
#define SLEEP_1_MS_RESOLUTION         1
#define SLEEP_32_MS_RESOLUTION        2
#define SLEEP_1_S_RESOLUTION          3

#define MASTER_BUTTON                 1
#define SLAVE_BUTTON                  2
#define BOTH_BUTTONS                  3
```

8.2.4 Protocolo_inf_rodoviarias.h

```
/*************************************************************************
*********
* Definições gerais
*/
#define REQ_PWR_LEVEL           0
#define CUR_PWR_LEVEL           1
#define TIPO_MENSAGEM           2
#define MAX_LENGTH              30
#define MAX_LENGTH_SMS          300
#define DELIMITADOR_PACOTE      255

/*************************************************************************
*********
* Entrada/Saída de rodovia
*/
#define ES_LENGTH_VAL           MAX_LENGTH
#define ES_TIPO_VAL             0
#define ES_COD_ROD_H            3
#define ES_COD_ROD_L            4
#define ES_NOME_ROD_INICIAL     5
#define ES_NOME_ROD_MAX_LENGTH  24
/* OBS: o último caracter do nome da rodovia deve ser '\0' (para ser
identificado como string pelo receptor) */

/*************************************************************************
*********
* Velocidade Máxima
*/
#define VM_LENGTH_VAL           5
#define VM_TIPO_VAL             1
#define VM_H                    3
#define VM_L                    4

/*************************************************************************
*********
* Buraco
*/
#define B_LENGTH_VAL            28
#define B_TIPO_VAL              2
#define B_NUM_BURACOS           3
#define B_NUM_MAX_BURACO_VAL    4
#define B_BURACOS_INICIO        4
#define B_BURACOS_LENGTH_VAL    6
#define B_KM_INICIAL_H_REL      0
#define B_KM_INICIAL_L_REL      1
#define B_KM_FINAL_H_REL        2
#define B_KM_FINAL_L_REL        3
#define B_FAIXA_INICIAL_REL     4
```

```
#define B_FAIXA_FINAL_REL       5

/*************************************************************************
*********
* Acidente/Obras
*/
#define AO_LENGTH_VAL           25
#define AO_TIPO_VAL             3
#define AO_ACIDENTE_VAL         0
#define AO_OBRA_VAL             1
#define AO_NUM_AOS              3
#define AO_NUM_MAX_AO_VAL       3
#define AO_ACID_OBR_INICIO      4
#define AO_ACID_OBR_LENGTH_VAL  7
#define AO_ACIDENTE_OBRA_REL    0
#define AO_KM_INICIAL_H_REL     1
#define AO_KM_INICIAL_L_REL     2
#define AO_KM_FINAL_H_REL       3
#define AO_KM_FINAL_L_REL       4
#define AO_FAIXA_INICIAL_REL    5
#define AO_FAIXA_FINAL_REL      6

/*************************************************************************
*********
* Trânsito
*/
#define TR_LENGTH_VAL           MAX_LENGTH
#define TR_TIPO_C_ALERTA_VAL    4
#define TR_TIPO_S_ALERTA_VAL    5
#define TR_KM_INICIAL_H         3
#define TR_KM_INICIAL_L         4
#define TR_KM_FINAL_H           5
#define TR_KM_FINAL_L           6
#define TR_KM_ALT_H             7
#define TR_KM_ALT_L             8
#define TR_NOME_ROD_INICIAL     9
#define TR_NOME_ROD_MAX_LENGTH  21
/* OBS: o último caracter do nome da rodovia alternativa deve ser '\0'
(para ser
identificado como string pelo receptor) */

/*************************************************************************
*********
* Tempo
*/
#define TE_LENGTH_VAL           MAX_LENGTH
#define TE_TIPO_C_ALERTA_VAL    6
#define TE_TIPO_S_ALERTA_VAL    7
#define TE_DESCRICAO_INICIAL    3
#define TE_DESCRICAO_MAX_LENGTH 27
```

8.3 Módulo da rodovia

8.3.1 Mod_rodovia.h

```
void        modulo_rodovia(void);
```

8.3.2 Mod_rodovia.c

```
#include "configuracoes.h"
```

```
#include "definicoes_gerais.h"
#include "mod_rodovia.h"
#include "protocolo_inf_rodoviarias.h"
#include "modem.h"
#include "usart.h"
#include "display.h"
#include "mrfi.h"
#include "nwk_types.h"

/*************************************************************************
*********
* LOCAL VARIABLES
*/
static          linkID_t              sLinkID;
static          ioctlRadioSiginfo_t   info;
static volatile uint8_t               sSemaphore;
static          uint8_t               sCurrentPwrLevel;
static          uint8_t               sRequestPwrLevel;
static          uint8_t               entradaSaidaMsg[ES_LENGTH_VAL];
static          uint8_t               velocidadeMaxMsg[VM_LENGTH_VAL];
static          uint8_t               buracoMsg[B_LENGTH_VAL];
static          uint8_t               acidenteObraMsg[AO_LENGTH_VAL];
static          uint8_t               transitoMsg[TR_LENGTH_VAL];
static          uint8_t               tempoMsg[TE_LENGTH_VAL];
static          uint8_t               pacote_sms[MAX_LENGTH_SMS];

/*************************************************************************
*********
* LOCAL FUNCTIONS
*/
static uint8_t    sRxCallback(linkID_t);
static uint8_t    enviaPacote(uint8_t *radioMsg, uint8_t tamanho);
static void       processaSMS(uint8_t *dados);

extern unsigned char __xdata uartRxBuffer[SIZE_OF_UART_RX_BUFFER];

void modulo_rodovia(void)
{
  printf("Entrou no modulo_rodovia.\n");

  //Inicializa a porta serial
  USARTInit();
  //Espera 2s para o modem inicializar
  NWK_DELAY(2000);
  //Inicializa o modem
  if(!inicializaModem())
  {
    printf("Falha ao inicializar modem.\n");
    //Falha na comunicação com modem
    while(1)
    {
      BSP_TOGGLE_LED2();
    }
  }


  printf("Modem inicalizado com sucesso!\n");

  sCurrentPwrLevel  = MAXIMUM_OUTPUT_POWER;
  sRequestPwrLevel  = MAXIMUM_OUTPUT_POWER;
```

```
  /* Initialize SimpliciTI and provide Callback function */
  SMPL_Init(sRxCallback);

  /* Continue to try to link until success */
  while (SMPL_SUCCESS != SMPL_Link(&sLinkID))
  {
    BSP_TOGGLE_LED1();
    BSP_TOGGLE_LED2();
  }

  BSP_TURN_OFF_LED1();
  BSP_TURN_OFF_LED2();

  while(1)
  {
    /*Busca dados novos do modem*/
    if(recebeSMS(pacote_sms))
    {
      processaSMS(pacote_sms); //Envia os dados novos
    }
    else if(pacote_sms[0] == DELIMITADOR_PACOTE)
      processaSMS(pacote_sms); //Envia os dados armazenados

    BSP_TURN_ON_LED1();
    BSP_SleepFor( _POWER_MODE_2, SLEEP_1_MS_RESOLUTION, 500);
    BSP_TURN_OFF_LED1();

    NWK_DELAY(1000); //Espera 1segundo para verificar se novos dados foram
enviados

    /* Reenvia os pacotes somente quando o MASTER_BUTTON é pressionado */
    //BSP_SleepUntilButton( _POWER_MODE_3, MASTER_BUTTON);
  }
}

/*************************************************************************
*********
* @fn          sRxCallback
*
* @brief
*
* @param       lid - link id message receive at
*
* @return      0 - frame left for application to read
*              1 - frame could be overwritten
*/
static uint8_t sRxCallback(linkID_t lid)
{
  if(lid)
  {
    sSemaphore = 1;

    /* Radio IDLE to save power */
    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXIDLE, 0);
  }

  /* Leave frame to be read by application. */
  return 0;
}

static void processaSMS(uint8_t *dados)
```

```
{

  uint8_t i, indiceBase;
  uint8_t nomeRodoviaLength, descricaoTempoLength;
  char nomeRodovia[] = "Bandeirantes";
  char nomeRodoviaAlternativa[] = "Castelo Branco";
  char descricaoTempo[] = "Chuvoso";
  uint8_t *ptr = "";
  uint8_t acabou = 0;

  ptr = (uint8_t*)memchr(dados, 255, MAX_LENGTH_SMS); //Encontra o primeiro
iniciador de pacote '0xFF'

  while(!acabou) //Enquanto nao ler o SMS todo
  {
    if(ptr[0] == DELIMITADOR_PACOTE) //Marcador de inicio de pacote
    {
      if(ptr[1] == DELIMITADOR_PACOTE) //Acabou o sms (marcador de fim =
dois delimitadores de inicio)
      {
        acabou = 1;
        break;
      }

      ptr++;
      switch(ptr[0])
      {
      case 0: //Entrada/Saida
        /* Monta a mensagem de entrada/saída da rodovia */
        entradaSaidaMsg[REQ_PWR_LEVEL] = sRequestPwrLevel;
        entradaSaidaMsg[CUR_PWR_LEVEL] = sCurrentPwrLevel;
        entradaSaidaMsg[TIPO_MENSAGEM] = ES_TIPO_VAL;
        entradaSaidaMsg[ES_COD_ROD_H] = ptr[1];
        entradaSaidaMsg[ES_COD_ROD_L] = ptr[2];
        strcpy(nomeRodovia, (const char*)ptr+3);
        nomeRodoviaLength = strlen(nomeRodovia);
        if(nomeRodoviaLength > ES_NOME_ROD_MAX_LENGTH)
          nomeRodoviaLength = ES_NOME_ROD_MAX_LENGTH;
        memmove(entradaSaidaMsg + ES_NOME_ROD_INICIAL, nomeRodovia,
                nomeRodoviaLength);

        enviaPacote(entradaSaidaMsg, sizeof(entradaSaidaMsg));
        break;
      case 1: //Velocidade
        /* Monta a mensagem de velocidade máxima */
        velocidadeMaxMsg[REQ_PWR_LEVEL] = sRequestPwrLevel;
        velocidadeMaxMsg[CUR_PWR_LEVEL] = sCurrentPwrLevel;
        velocidadeMaxMsg[TIPO_MENSAGEM] = VM_TIPO_VAL;
        velocidadeMaxMsg[VM_H] = ptr[1];
        velocidadeMaxMsg[VM_L] = ptr[2];

        enviaPacote(velocidadeMaxMsg, sizeof(velocidadeMaxMsg));
        break;
      case 2: //Buraco
        /* Monta a mensagem de buracos */
        buracoMsg[REQ_PWR_LEVEL] = sRequestPwrLevel;
        buracoMsg[CUR_PWR_LEVEL] = sCurrentPwrLevel;
        buracoMsg[TIPO_MENSAGEM] = B_TIPO_VAL;
        buracoMsg[B_NUM_BURACOS] = ptr[1];
        for(i = 0; i < buracoMsg[B_NUM_BURACOS]; i++)
        {
```

```
          indiceBase = B_BURACOS_INICIO + i*B_BURACOS_LENGTH_VAL;
          buracoMsg[indiceBase + B_KM_INICIAL_H_REL]   =   ptr[2 + ( i *
B_BURACOS_LENGTH_VAL)];
          buracoMsg[indiceBase + B_KM_INICIAL_L_REL]   =   ptr[3 + ( i *
B_BURACOS_LENGTH_VAL)];
          buracoMsg[indiceBase + B_KM_FINAL_H_REL]     =   ptr[4 + ( i *
B_BURACOS_LENGTH_VAL)];
          buracoMsg[indiceBase + B_KM_FINAL_L_REL]     =   ptr[5 + ( i *
B_BURACOS_LENGTH_VAL)];
          buracoMsg[indiceBase + B_FAIXA_INICIAL_REL] =   ptr[6 + ( i *
B_BURACOS_LENGTH_VAL)];
          buracoMsg[indiceBase + B_FAIXA_FINAL_REL]    =   ptr[7 + ( i *
B_BURACOS_LENGTH_VAL)];
        }

        enviaPacote(buracoMsg, sizeof(buracoMsg));
        break;
      case 3: //Acidente/Obra
        /* Monta a mensagem de acidentes/obras */
        acidenteObraMsg[REQ_PWR_LEVEL] = sRequestPwrLevel;
        acidenteObraMsg[CUR_PWR_LEVEL] = sCurrentPwrLevel;
        acidenteObraMsg[TIPO_MENSAGEM] = AO_TIPO_VAL;
        acidenteObraMsg[AO_NUM_AOS] = ptr[1];
        for(i = 0; i < acidenteObraMsg[AO_NUM_AOS]; i++)
        {
          indiceBase = AO_ACID_OBR_INICIO + i*AO_ACID_OBR_LENGTH_VAL;
          acidenteObraMsg[indiceBase + AO_ACIDENTE_OBRA_REL ]    =     ptr[2
+ ( i * AO_ACID_OBR_LENGTH_VAL)];
          acidenteObraMsg[indiceBase + AO_KM_INICIAL_H_REL]      =     ptr[3
+ ( i * AO_ACID_OBR_LENGTH_VAL)];
          acidenteObraMsg[indiceBase + AO_KM_INICIAL_L_REL]      =     ptr[4
+ ( i * AO_ACID_OBR_LENGTH_VAL)];
          acidenteObraMsg[indiceBase + AO_KM_FINAL_H_REL]        =     ptr[5
+ ( i * AO_ACID_OBR_LENGTH_VAL)];
          acidenteObraMsg[indiceBase + AO_KM_FINAL_L_REL]        =     ptr[6
+ ( i * AO_ACID_OBR_LENGTH_VAL)];
          acidenteObraMsg[indiceBase + AO_FAIXA_INICIAL_REL]     =     ptr[7
+ ( i * AO_ACID_OBR_LENGTH_VAL)];
          acidenteObraMsg[indiceBase + AO_FAIXA_FINAL_REL]       =     ptr[8
+ ( i * AO_ACID_OBR_LENGTH_VAL)];
        }

        enviaPacote(acidenteObraMsg, sizeof(acidenteObraMsg));
        break;
      case 4: //Trânsito
      case 5:
        /* Monta a mensagem de trânsito */
        transitoMsg[REQ_PWR_LEVEL] = sRequestPwrLevel;
        transitoMsg[CUR_PWR_LEVEL] = sCurrentPwrLevel;
        transitoMsg[TIPO_MENSAGEM] = ptr[0];
        transitoMsg[TR_KM_INICIAL_H] = ptr[1];
        transitoMsg[TR_KM_INICIAL_L] = ptr[2];
        transitoMsg[TR_KM_FINAL_H] = ptr[3];
        transitoMsg[TR_KM_FINAL_L] = ptr[4];
        transitoMsg[TR_KM_ALT_H] = ptr[5];
        transitoMsg[TR_KM_ALT_L] = ptr[6];
        strcpy(nomeRodoviaAlternativa, (const char*)ptr+7);
        nomeRodoviaLength = strlen(nomeRodoviaAlternativa);
        if (nomeRodoviaLength > TR_NOME_ROD_MAX_LENGTH)
          nomeRodoviaLength = TR_NOME_ROD_MAX_LENGTH;
        memmove(transitoMsg + TR_NOME_ROD_INICIAL, nomeRodoviaAlternativa,
```

```
                nomeRodoviaLength);

        enviaPacote(transitoMsg, sizeof(transitoMsg));
        break;
      case 6: //Tempo
      case 7:
        /* Monta a mensagem de previsão/condição do tempo */
        tempoMsg[REQ_PWR_LEVEL] = sRequestPwrLevel ;
        tempoMsg[CUR_PWR_LEVEL] = sCurrentPwrLevel;
        tempoMsg[TIPO_MENSAGEM] = ptr[0];
        strcpy(descricaoTempo, (const char*)ptr+1);
        descricaoTempoLength = strlen(descricaoTempo);
        if (descricaoTempoLength > TE_DESCRICAO_MAX_LENGTH)
          descricaoTempoLength = TE_DESCRICAO_MAX_LENGTH;
        memmove(tempoMsg + TE_DESCRICAO_INICIAL, descricaoTempo,
                descricaoTempoLength);

        enviaPacote(tempoMsg, sizeof(tempoMsg));
        break;
      }
    }
    else
      ptr++;
  }
}

static uint8_t enviaPacote(uint8_t *radioMsg, uint8_t tamanho)
{
  uint8_t ackMsg[2], ackMsgLength;

  /* Verifica o tipo de mensagem a ser enviada */
  /*switch(radioMsg[TIPO_MENSAGEM])
  {
  case 0:
    radioMsg = (uint8_t *)malloc(sizeof(entradaSaidaMsg));
    radioMsg = entradaSaidaMsg;
    //copiarVetores(entradaSaidaMsg, radioMsg, ES_LENGTH_VAL);
    break;
  case 1:
    copiarVetores(velocidadeMaxMsg, radioMsg, VM_LENGTH_VAL);
    break;
  case 2:
    copiarVetores(buracoMsg, radioMsg, B_LENGTH_VAL);
    break;
  case 3:
    copiarVetores(acidenteObraMsg, radioMsg, AO_LENGTH_VAL);
    break;
  case 4:
    copiarVetores(transitoMsg, radioMsg, TR_LENGTH_VAL);
    break;
  case 5:
    copiarVetores(transitoMsg, radioMsg, TR_LENGTH_VAL);
    break;
  case 6:
    copiarVetores(tempoMsg, radioMsg, TE_LENGTH_VAL);
    break;
  case 7:
    copiarVetores(tempoMsg, radioMsg, TE_LENGTH_VAL);
    break;
  }
  */
```

```
  for( uint8_t x = 0; x < NUM_TX_RETRIES; x++ )
  {
    SMPL_Send(sLinkID, radioMsg, tamanho);

    /* Turn on RX. default is RX Idle. */
    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, 0);

    SPIN_ABOUT_QUARTER_A_SECOND;  /* Might have to be longer for bigger
payloads */
    if( sSemaphore )
      break;
  }

  /* Radio IDLE to save power */
  SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXIDLE, 0);

  if( sSemaphore )                    /* Acknowledge successfully received
*/
  {
    SMPL_Receive(sLinkID, ackMsg, &ackMsgLength);
    sSemaphore = 0;

    /* Check and set desired output power */
    if ( sCurrentPwrLevel != ackMsg[0] )
    {
      sCurrentPwrLevel = ackMsg[0];
      SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SETPWR,
&sCurrentPwrLevel);
    }

    /* Check and adjust wanted output power */
    info.lid = sLinkID;
    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SIGINFO, &info);
    if( sRequestPwrLevel > MINIMUM_OUTPUT_POWER )
      if( info.sigInfo.rssi  > RSSI_UPPER_THRESHOLD )
        sRequestPwrLevel--;

    if( sRequestPwrLevel < MAXIMUM_OUTPUT_POWER )
      if( info.sigInfo.rssi  < RSSI_LOWER_THRESHOLD )
        sRequestPwrLevel++;

    BSP_TURN_ON_LED1();
    BSP_SleepFor( _POWER_MODE_2, SLEEP_1_MS_RESOLUTION, 500);
    BSP_TURN_OFF_LED1();

    return 1;
  }
  else /* No ACK */
  {
    return 0;
  }
}
```

8.3.3 Modem.h

```
#include "bsp.h"

uint8_t inicializaModem();
uint8_t recebeSMS(uint8_t*);
void apagaSMS();
uint8_t recebeResposta();
```

```
void enviaComando(char*);
```

8.3.4 Modem.c

```
#include "modem.h"
#include "usart.h"
#include <string.h>
#include "mrfi.h"
#include "nwk_types.h"

#define WAIT_COMMAND_INTERVAL          NWK_DELAY(300)
#define VERIFICA_OK                    st(if(!recebeResposta()) return 0;
else LIMPA_RX_BUFFER;)
#define LIMPA_RX_BUFFER                memset(uartRxBuffer, 0,
SIZE_OF_UART_RX_BUFFER)

extern unsigned char __xdata uartRxBuffer[SIZE_OF_UART_RX_BUFFER];
extern unsigned char __xdata uartTxBuffer[SIZE_OF_UART_TX_BUFFER];
extern unsigned short __xdata uartRxIndex, uartTxIndex;

uint8_t inicializaModem()
{
  uint8_t alive = 0;
  uint8_t tries = 0;

 //Verifica se o Modem está vivo
  while(!alive && tries++ < 20)
  {
    enviaComando("AT\r");
    alive = recebeResposta();
    NWK_DELAY(500); //Espera aproximadamente 0.5s para tentar novamente
  }

  if(!alive)      //Não conseguiu comunicar com o modem por 10s
    return 0;

  //Modem está vivo, envia comandos de inicialização
  LIMPA_RX_BUFFER;

  //Inicialização Geral
  enviaComando("AT&F\r"); //Factory settings
  VERIFICA_OK;
  WAIT_COMMAND_INTERVAL;
  enviaComando("AT+CMEE=2\r"); //Extended error reporting
  VERIFICA_OK;
  WAIT_COMMAND_INTERVAL;
  enviaComando("AT+CRC=1\r"); //Call notification
  VERIFICA_OK;
  WAIT_COMMAND_INTERVAL;
  enviaComando("AT\\Q3\r"); //Hardware flow control
  VERIFICA_OK;
  WAIT_COMMAND_INTERVAL;

  //Inicailização do SMS
  enviaComando("AT+CMGF=1\r"); //Text mode
  VERIFICA_OK;
  WAIT_COMMAND_INTERVAL;
  enviaComando("AT+CSDH=0\r"); //Remove parameters
  VERIFICA_OK;
  WAIT_COMMAND_INTERVAL;
  //enviaComando("AT+CSCA=1"); //Service center number
```

```
  //VERIFICA_OK;
  //WAIT_COMMAND_INTERVAL;

  enviaComando("AT&W\r"); //Grava configurações
  VERIFICA_OK;
  WAIT_COMMAND_INTERVAL;

  return 1;
}

void enviaComando(char* comando)
{
  int length = strlen(comando);
  uart0Send((unsigned char*)comando, strlen(comando));
}

uint8_t recebeResposta()
{
  char* ok = 0;
  char* error = 0;

  //Habilita interrupcao do UART.RX
  IEN0 |= 0x04;

  while(!ok || !error)
  {
    //Fica olhando o vetor de retorno para ver se terminou a resposta ("OK"
ou "ERROR")
    ok = strstr ((const char*)uartRxBuffer,"OK\r\n");
    error = strstr ((const char*)uartRxBuffer,"ERROR");

    if(ok)
    {
      //Terminou a resposta, e o comando foi aceito.

      IEN0 &= ~0x04; //Desabilita interrupcao usart
      uartRxBuffer[uartRxIndex] = '\0'; //termina a string
      uartRxIndex = 0; //zera o indice do buffer
      return 1;
    }
    else if (error)
    {
      //Terminou a resposta, mas deu erro.
      uartRxIndex = 0; IEN0 &= ~0x04;
      return 0;
    }

    WAIT_COMMAND_INTERVAL;
  }
  return 1;

}

uint8_t recebeSMS(uint8_t* p_sms)
{
  uint8_t* p_buf = 0;
  enviaComando("AT+CMGL=\"REC UNREAD\"\r");
  if(recebeResposta())
  {
    //Pega a última ocorrência de "
    p_buf = (uint8_t*)(strrchr((const char*)uartRxBuffer, 34));
```

```
    if(p_buf)
    {
      p_buf++; //Avança ponteiro para começo do SMS
      strcpy((char*)p_sms, (const char*)p_buf); //Copia SMS para uma string
nova;
      LIMPA_RX_BUFFER; //Apaga o Buffer

      //Apaga SMS no modem
      WAIT_COMMAND_INTERVAL;
      apagaSMS();

      return 1;
    }
  }
  return 0;
}

void apagaSMS()
{
  enviaComando("AT+CMGD=1\r");
}
```

8.3.5 Usart.h

```
#include "bsp.h"

/* Configurações utilizadas para comunicação RS-232*/
#define SERIAL_PORT                    P1
#define TXD_PIN                        4
#define RXD_PIN                        5
#define RTS_PIN                        6
#define CTS_PIN                        7
#define TXD                            P1_4
#define RXD                            P1_5
#define RTS                            P1_6
#define CTS                            P1_7
#define BAUD_RATE                      1000.0

// Define size of allocated UART RX/TX buffer
#define SIZE_OF_UART_RX_BUFFER 1000
#define SIZE_OF_UART_TX_BUFFER SIZE_OF_UART_RX_BUFFER

void USARTInit();
//void USARTWriteByte(uint8_t);
//void USARTWriteString(char string[]);
//void USARTWriteInt(int , uint8_t);
//uint8_t USARTReadByte();
//void USARTReadString(char*);

void uart0Send(unsigned char*, unsigned short);
void uart0Receive(unsigned char*, unsigned short);
```

8.3.6 Usart.c

```
#include "usart.h"
#include <string.h>

// Define and allocate a setup structure for the UART protocol:
typedef struct {
  unsigned char uartNum          : 1; // UART peripheral number (0 or 1)
  unsigned char START            : 1; // Start bit level (low/high)
```

```
  unsigned char STOP              : 1; // Stop bit level (low/high)
  unsigned char SPB               : 1; // Stop bits (0 => 1, 1 => 2)
  unsigned char PARITY            : 1; // Parity control (enable/disable)
  unsigned char BIT9              : 1; // 9 bit enable (8bit / 9bit)
  unsigned char D9                : 1; // 9th bit level or Parity type
  unsigned char FLOW              : 1; // HW Flow Control (enable/disable)
  unsigned char ORDER             : 1; // Data bit order(LSB/MSB first)
} UART_PROT_CONFIG;

UART_PROT_CONFIG __xdata uartProtConfig;

// Allocate buffer+ind
unsigned char __xdata uartRxBuffer[SIZE_OF_UART_RX_BUFFER];
unsigned char __xdata uartTxBuffer[SIZE_OF_UART_TX_BUFFER];
unsigned short __xdata uartRxIndex, uartTxIndex;

void uartMapPort(unsigned char, unsigned char);
void uartInitBitrate(unsigned char, unsigned char);
void uartInitProtocol(UART_PROT_CONFIG*);

void uartStartRxForIsr(unsigned char);
_Pragma("vector=0x13") __near_func __interrupt void UART0_RX_ISR(void);

void USARTInit()
{
  //Mapeia o USART para P0
  uartMapPort(1, 0);

  //Define Baudrate = 9600
  uartInitBitrate(131, 8);

  uartProtConfig.uartNum = 0;
  uartProtConfig.START = 0;
  uartProtConfig.STOP = 1;
  uartProtConfig.SPB = 0;
  uartProtConfig.PARITY = 0;
  uartProtConfig.BIT9 = 0;
  uartProtConfig.D9 = 0;
  uartProtConfig.FLOW = 0;
  uartProtConfig.ORDER = 0;

  //Inicializa o protocolo RS-232
  uartInitProtocol(&uartProtConfig);

  //Habilita Interrupcoes de RX
  uartStartRxForIsr(0);
}


// This function maps/connects the UART to the desired SoC I/O port.
// The application should call this function with "uartPortAlt" = 1 or 2,
// and "uartNum" = 0 or 1.
void uartMapPort(unsigned char uartPortAlt, unsigned char uartNum) {
  // If UART Port Alternative 1 desired
  if(uartPortAlt == 1) {
    // If UART0 desired
    if (uartNum == 0) {
      // Configure UART0 for Alternative 1 => Port P0 (PERCFG.U0CFG = 0)
      PERCFG &= ~0x01;
      // Configure relevant Port P0 pins for peripheral function:
```

```
      // P0SEL.SELP0_2/3/4/5 = 1 => RX = P0_2, TX = P0_3, CT = P0_4, RT =
P0_5
      P0SEL |= 0x3C;
      // Configure relevant Port P1 pins back to GPIO function
      P1SEL &= ~0x3C;
      // Else (UART1 desired)
    } else {
      // Configure UART1 for Alternative 1 => Port P0 (PERCFG.U1CFG = 0)
      PERCFG &= ~0x02;
      // Configure relevant Port P0 pins for peripheral function:
      // P0SEL.SELP0_2/3/4/5 = 1 => CT = P0_2, RT = P0_3, TX = P0_4, RX =
P0_5
      P0SEL |= 0x3C;
      // Configure relevant Port P1 pins back to GPIO function
      P1SEL &= ~0xF0;
    }
    // Else (UART Port Alternative 2 desired)
  } else {
    // If UART0 desired
    if (uartNum == 0) {
      // Configure UART0 for Alternative 2 => Port P1 (PERCFG.U0CFG = 1)
      PERCFG |= 0x01;
      // P1SEL.SELP1_2/3/4/5 = 1 => CT = P1_2, RT = P1_3, RX = P1_4, TX =
P1_5
      P1SEL |= 0x3C;
      // Configure relevant Port P0 pins back to GPIO function
      P0SEL &= ~0x3C;
      // Else (UART1 desired)
    } else {
      // Configure UART1 for Alternative 2 => Port P1 (PERCFG.U1CFG = 1)
      PERCFG |= 0x02;
      // P1SEL.SELP1_4/5/6/7 = 1 => CT = P1_4, RT = P1_5, TX = P1_6, RX =
P1_7
      P1SEL |= 0xF0;
      // Configure relevant Port P0 pins back to GPIO function
      P0SEL &= ~0x3C;
    }
  }
}

// This function initializes the UART bit rate.
void uartInitBitrate(unsigned char uartBaudM, unsigned char uartBaudE) {
  //////////////////////////////////////////////////////////////
  // This initial code section ensures that the SoC system clock is driven
  // by the HS XOSC:
  // Clear CLKCON.OSC to make the SoC operate on the HS XOSC.
  // Set CLKCON.TICKSPD/CLKSPD = 000 => system clock speed = HS RCOSC
speed.
  CLKCON &= 0x80;
  // Monitor CLKCON.OSC to ensure that the HS XOSC is stable and actually
  // applied as system clock source before continuing code execution
  while(CLKCON & 0x40);
  // Set SLEEP.OSC_PD to power down the HS RCOSC.
  SLEEP |= 0x04;
  //////////////////////////////////////////////////////////////
  // Initialize bitrate (U0BAUD.BAUD_M, U0GCR.BAUD_E)
  U0BAUD = uartBaudM;
  U0GCR = (U0GCR&~0x1F) | uartBaudE;
}

// This function initializes the UART protocol (start/stop bit, data bits,
```

```
// parity, etc.). The application must call this function with an
initialized
// data structure according to the code in Figure 12.
void uartInitProtocol(UART_PROT_CONFIG* uartProtConfig) {
  // Initialize UART protocol for desired UART (0 or 1)
  if (uartProtConfig->uartNum == 0) {
    // USART mode = UART (U0CSR.MODE = 1)
    U0CSR |= 0x80;
    // Start bit level = low => Idle level = high (U0UCR.START = 0)
    // Start bit level = high => Idle level = low (U0UCR.START = 1)
    U0UCR = (U0UCR&~0x01) | uartProtConfig->START;
    // Stop bit level = high (U0UCR.STOP = 1)
    // Stop bit level = low (U0UCR.STOP = 0)
    U0UCR = (U0UCR&~0x02) | (uartProtConfig->STOP << 1);
    // Number of stop bits = 1 (U0UCR.SPB = 0)
    // Number of stop bits = 2 (U0UCR.SPB = 1)
    U0UCR = (U0UCR&~0x04) | (uartProtConfig->SPB << 2);
    // Parity = disabled (U0UCR.PARITY = 0)
    // Parity = enabled (U0UCR.PARITY = 1)
    U0UCR = (U0UCR&~0x08) | (uartProtConfig->PARITY << 3);
    // 9-bit data disable = 8 bits transfer (U0UCR.BIT9 = 0)
    // 9-bit data enable = 9 bits transfer (U0UCR.BIT9 = 1)
    U0UCR = (U0UCR&~0x10) | (uartProtConfig->BIT9 << 4);
    // Level of bit 9 = 0 (U0UCR.D9 = 0), used when U0UCR.BIT9 = 1
    // Level of bit 9 = 1 (U0UCR.D9 = 1), used when U0UCR.BIT9 = 1
    // Parity = Even (U0UCR.D9 = 0), used when U0UCR.PARITY = 1
    // Parity = Odd (U0UCR.D9 = 1), used when U0UCR.PARITY = 1
    U0UCR = (U0UCR&~0x20) | (uartProtConfig->D9 << 5);
    // Flow control = disabled (U0UCR.FLOW = 0)
    // Flow control = enabled (U0UCR.FLOW = 1)
    U0UCR = (U0UCR&~0x40) | (uartProtConfig->FLOW << 6);
    // Bit order = MSB first (U0GCR.ORDER = 1)
    // Bit order = LSB first (U0GCR.ORDER = 0) => For PC/Hyperterminal
    U0GCR = (U0GCR&~0x20) | (uartProtConfig->ORDER << 5);
  } else {
    // USART mode = UART (U1CSR.MODE = 1)
    U1CSR |= 0x80;
    // Start bit level = low => Idle level = high (U1UCR.START = 0)
    // Start bit level = high => Idle level = low (U1UCR.START = 1)
    U1UCR = (U1UCR&~0x01) | uartProtConfig->START;
    // Stop bit level = high (U1UCR.STOP = 1)
    // Stop bit level = low (U1UCR.STOP = 0)
    U1UCR = (U1UCR&~0x02) | (uartProtConfig->STOP << 1);
    // Number of stop bits = 1 (U1UCR.SPB = 0)
    // Number of stop bits = 2 (U1UCR.SPB = 1)
    U1UCR = (U1UCR&~0x04) | (uartProtConfig->SPB << 2);
    // Parity = disabled (U1UCR.PARITY = 0)
    // Parity = enabled (U1UCR.PARITY = 1)
    U1UCR = (U1UCR&~0x08) | (uartProtConfig->PARITY << 3);
    // 9-bit data enable = 8 bits transfer (U1UCR.BIT9 = 0)
    // 9-bit data enable = 8 bits transfer (U1UCR.BIT9 = 1)
    U1UCR = (U1UCR&~0x10) | (uartProtConfig->BIT9 << 4);
    // Level of bit 9 = 0 (U1UCR.D9 = 0), used when U1UCR.BIT9 = 1
    // Level of bit 9 = 1 (U1UCR.D9 = 1), used when U1UCR.BIT9 = 1
    // Parity = Even (U1UCR.D9 = 0), used when U1UCR.PARITY = 1
    // Parity = Odd (U1UCR.D9 = 1), used when U1UCR.PARITY = 1
    U1UCR = (U1UCR&~0x20) | (uartProtConfig->D9 << 5);
    // Flow control = disabled (U1UCR.FLOW = 0)
    // Flow control = enabled (U1UCR.FLOW = 1)
    U1UCR = (U1UCR&~0x40) | (uartProtConfig->FLOW << 6);
    // Bit order = MSB first (U1GCR.ORDER = 1)
```

```
    // Bit order = LSB first (U1GCR.ORDER = 0) => For PC/Hyperterminal
    U1GCR = (U1GCR&~0x20) | (uartProtConfig->ORDER << 5);
  }
}

// The two functions below send a range of bytes on the UARTx TX line. Note
// that, before the relevant function is called the application must
execute
// the initialization code in Figure 3, Figure 11, Figure 12, and Figure
13.
// The code implements the following steps:
// 1. Clear TX interrupt request (UTXxIF = 0).
// 2. Loop: send each UARTx source byte on the UARTx TX line.
// 2a. Read byte from the allocated UART TX source buffer and write to
UxDBUF.
// 2b. Wait until UART byte has been sent (UTXxIF = 1).
// 2c. Clear UTXxIF.
void uart0Send(unsigned char* uartTxBuf, unsigned short uartTxBufLength) {
  unsigned short uartTxIndex;
  UTX0IF = 0;
  for (uartTxIndex = 0; uartTxIndex < uartTxBufLength; uartTxIndex++) {
    U0DBUF = uartTxBuf[uartTxIndex];
    while( !UTX0IF );
    UTX0IF = 0;
  }
}



// The code implements the following steps:
// 1. Enable UARTx RX (UxCSR.RE = 1)
// 2. Clear RX interrupt request (set URXxIF = 0)
// 3. Loop: receive each UARTx sample from the UARTx RX line.
// 3a. Wait until data received (URXxIF = 1).
// 3b. Read UxDBUF and store the value in the allocated UART RX target
buffer.
void uart0Receive(unsigned char* uartRxBuf, unsigned short uartRxBufLength)
{
  unsigned short uartRxIndex;
  U0CSR |= 0x40; URX0IF = 0;
  for (uartRxIndex = 0; uartRxIndex < uartRxBufLength; uartRxIndex++) {
    while( !URX0IF );
    uartRxBuf[uartRxIndex] = U0DBUF;
    URX0IF = 0;
  }
    uartRxBuf[uartRxBufLength] = '\0';
}

// This function initializes the UART RX session, by simply enabling the
// corresponding UART interrupt, and leave the sample reception to the
// UART ISR shown in Figure 19. Before this function is called the
// application must initialize the UART peripheral according to the
// code shown in Figure 3, Figure 11, Figure 12, and Figure 13.
// The code implements the following steps:
// 1. Initialize the UART RX buffer index.
// 2. Clear UART RX Interrupt Flag (TCON.URXxIF = 0)
// 3. Enable UART RX and Interrupt (IEN0.URXxIE = 1, UxCSR.RE = 1)
// 4. Enable global interrupt (IEN0.EA = 1)

void uartStartRxForIsr(unsigned char uartNum) {
  uartRxIndex = 0;
```

```
  if (uartNum == 0) {
    URX0IF = 0;
    U0CSR |= 0x40;
    IEN0 |= 0x04;
  } else {
    URX1IF = 0;
    U1CSR |= 0x40;
    IEN0 |= 0x08;
  }
  IEN0 |= 0x80;
}

// The UARTx RX ISR assumes that the code in Figure 18 has initialized the
// UART RX session, by enabling the UART RX interrupt. Then this UART RX
ISR
// will receive the data based in interrupt request generation by the
// USART peripheral.
// The code implements the following steps:
// 1. Clear UARTx RX Interrupt Flag (TCON.URXxIF = 0)
// 2. Read UxDBUF and store the value in the allocated UART RX target
buffer
// 3. If all UART data received, stop this UART RX session
// Note that in order to start another UART RX session the application
// just needs to re-enable the UART RX interrupt(IEN0.URXxIE = 1).

_Pragma("vector=0x13") __near_func __interrupt void UART0_RX_ISR(void){
  URX0IF = 0;
  uartRxBuffer[uartRxIndex++] = U0DBUF;
  if (uartRxIndex >= SIZE_OF_UART_RX_BUFFER) {
    uartRxIndex = 0; IEN0 &= ~0x04;
  }
}
```

8.4 Módulo veicular

8.4.1 Mod_veiculo.h

```
/* Telas exibidas no display de LCD */
#define   NUMERO_TELAS              9
#define   TELA_NOME_RODOVIA         0
#define   TELA_VELOCIDADE           1
#define   TELA_BURACO               2
#define   TELA_ACIDENTE_OBRA        3
#define   TELA_TRANSITO             4
#define   TELA_TEMPO                5
#define   TELA_BEEP                 6
#define   TELA_SAIR_RODOVIA         7
#define   TELA_SEM_RODOVIA          8

#define   PORT_BEEP                 P1
#define   BEEP_PIN                  7

void       modulo_veiculo(void);
```

8.4.2 Mod_veiculo.c

```
#include "configuracoes.h"
#include "definicoes_gerais.h"
#include "mod_veiculo.h"
#include "display.h"
```

```
#include "protocolo_inf_rodoviarias.h"

/*****************************************************************************
*********
* LOCAL VARIABLES
*/
static          linkID_t              sLinkID;
static          ioctlRadioSiginfo_t   info;
static volatile uint8_t               sSemaphore;
static          uint8_t               sCurrentPwrLevel;
static          uint8_t               sRequestPwrLevel;
static          uint8_t               velocidadeAtual;
static          uint8_t               telaAtualDisplay;
static          uint8_t               telaInfoAtualDisplay;
static          uint8_t               estadoBeep;
static          uint8_t               entradaSaidaMsg[ES_LENGTH_VAL];
static          uint8_t               velocidadeMaxMsg[VM_LENGTH_VAL];
static          uint8_t               buracoMsg[B_LENGTH_VAL];
static          uint8_t               acidenteObraMsg[AO_LENGTH_VAL];
static          uint8_t               transitoMsg[TR_LENGTH_VAL];
static          uint8_t               tempoMsg[TE_LENGTH_VAL];
static          uint8_t               mudanca[NUMERO_TELAS];
static          char                  linha_1_display[50];
static          char                  linha_2_display[50];
static          char                  numeroASCII[5];
static          char                  msgBotoesConfig[21];

/*****************************************************************************
*********
* LOCAL FUNCTIONS
*/
static uint8_t    sRxCallback(linkID_t);
static void       receberPacotes(void);
static uint8_t    copiarVetoresDetectarMudanca(uint8_t vetor_1[],
                                               uint8_t vetor_2[], uint8_t
tamanho);
static void       trocarTelaInfoDisplay(void);
static void       exibeLCDNomeRodovia(void);
static void       exibeLCDVelocidades(void);
static void       exibeLCDBuracos(void);
static void       exibeLCDAcidentesObras(void);
static void       exibeLCDTransito(void);
static void       exibeLCDTempo(void);
static void       inteiroParaASCII(uint8_t int_H, uint8_t int_l);
static void       atualizarVelocidadeAtual(void);
static void       verificarMudancas(void);
static void       inicializa_beep(void);
static void       beep(void);
static void       zerarVetor(uint8_t *vetor, uint8_t tamanho);
static void       tratarBotao1(void);
static void       tratarBotao2(void);
static void       exibeLCDAlterarBeep(void);
static void       exibeLCDSairRodovia(void);
static void       exibeLCDSemRodovia(void);
static void       alterarEstadoBeep(void);
static void       sairRodovia(void);
static void       exibirTelaInfo(uint8_t codTela);

void modulo_veiculo(void)
{
  printf("Entrou no modulo_veiculo.\n");
```

```
uint8_t i, j, k, pacoteRecebido;
sCurrentPwrLevel  = MAXIMUM_OUTPUT_POWER;
sRequestPwrLevel  = MAXIMUM_OUTPUT_POWER;

/* Initialize SimpliciTI and provide Callback function */
SMPL_Init(sRxCallback);

/* Inicializa o display de LCD */
inicializa_display();

/* Inicializa o beep */
inicializa_beep();

telaAtualDisplay = TELA_SEM_RODOVIA;
exibeLCDSemRodovia();

strcpy(msgBotoesConfig, "B1:OK     B2:Proximo");

/* Listen for link forever... */
BSP_TURN_ON_LED1();
BSP_TURN_ON_LED2();
while (SMPL_LinkListen(&sLinkID) != SMPL_SUCCESS)
{
  BSP_TOGGLE_LED1();
  BSP_TOGGLE_LED2();
}

/* Turning on LED2 to show that we have link*/
BSP_TURN_OFF_LED1();
BSP_TURN_ON_LED2();

/* turn on RX. default is RX off. */
SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, 0 );


while(1){

  pacoteRecebido = 0;

  for(i = 0; i < 17; i++)
  {
    for (j = 0; j < 255; j++)
    {
      for(k = 0; k < 255; k++){
        /* Verifica se recebeu um pacote */
        if(sSemaphore)
        {
          receberPacotes();
          if(!pacoteRecebido)
          {
            i = 0;
            j = 0;
            pacoteRecebido = 1;
          }
        }

        /* Verifica se o botão 1 foi pressionado */
        if(BSP_BUTTON1())
          tratarBotao1();
```

```
            if(BSP_BUTTON2())
              tratarBotao2();
          }

          /* Atualiza a velocidade atual */
          atualizarVelocidadeAtual();
        }
      }
      /* Verifica se houve mudanças */
      verificarMudancas();
      zerarVetor(mudanca, NUMERO_TELAS);
    }
  }

/**************************************************************************
*********
* LOCAL FUNCTIONS
*/


/**************************************************************************
*********
* @fn          sRxCallback
*
* @brief
*
* @param       lid - link id message receive at
*
* @return      0 - frame left for application to read
*              1 - frame could be overwritten
*/
static uint8_t sRxCallback(linkID_t lid)
{
  if(lid)
  {
    sSemaphore = 1;

    /* Radio IDLE to save power */
    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXIDLE, 0);
  }

  /* Leave frame to be read by application. */
  return 0;
}

static void receberPacotes(void)
{
  uint8_t radioMsg[30], len;
  uint8_t ackMsg[2];

  if( sSemaphore )                    /* Recebeu dados do MR */
  {
    /* Radio IDLE to save power */
    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXIDLE, 0);
    SMPL_Receive(sLinkID, radioMsg, &len);
    sSemaphore = 0;

    /* Check and set desired output power */
    if ( sCurrentPwrLevel != radioMsg[REQ_PWR_LEVEL] )
    {
      sCurrentPwrLevel = radioMsg[REQ_PWR_LEVEL];
```

```
      SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SETPWR,
&sCurrentPwrLevel);
    }

    /* Check and adjust wanted output power */
    info.lid = sLinkID;
    SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_SIGINFO, &info);
    if( sRequestPwrLevel > MINIMUM_OUTPUT_POWER )
      if( info.sigInfo.rssi  > RSSI_UPPER_THRESHOLD )
        sRequestPwrLevel--;

    if( sRequestPwrLevel < MAXIMUM_OUTPUT_POWER )
      if( info.sigInfo.rssi  < RSSI_LOWER_THRESHOLD )
        sRequestPwrLevel++;

    /* Analisa o tipo de mensagem recebida */
    switch(radioMsg[TIPO_MENSAGEM])
    {
    case 0:
      if(copiarVetoresDetectarMudanca(entradaSaidaMsg, radioMsg,
ES_LENGTH_VAL))
        mudanca[TELA_NOME_RODOVIA] = 1;
      break;
    case 1:
      if(copiarVetoresDetectarMudanca(velocidadeMaxMsg, radioMsg,
VM_LENGTH_VAL))
        mudanca[TELA_VELOCIDADE] = 1;
      break;
    case 2:
      if(copiarVetoresDetectarMudanca(buracoMsg, radioMsg, B_LENGTH_VAL))
        mudanca[TELA_BURACO] = 1;
      break;
    case 3:
      if(copiarVetoresDetectarMudanca(acidenteObraMsg, radioMsg,
AO_LENGTH_VAL))
        mudanca[TELA_ACIDENTE_OBRA] = 1;
      break;
    case 4:
      if(copiarVetoresDetectarMudanca(transitoMsg, radioMsg,
TR_LENGTH_VAL))
        mudanca[TELA_TRANSITO] = 1;
      break;
    case 5:
      if(copiarVetoresDetectarMudanca(transitoMsg, radioMsg,
TR_LENGTH_VAL))
        mudanca[TELA_TRANSITO] = 1;
      break;
    case 6:
      if(copiarVetoresDetectarMudanca(tempoMsg, radioMsg, TE_LENGTH_VAL))
        mudanca[TELA_TEMPO] = 1;
      break;
    case 7:
      if(copiarVetoresDetectarMudanca(tempoMsg, radioMsg, TE_LENGTH_VAL))
        mudanca[TELA_TEMPO] = 1;
      break;
    }

    /* Build and send acknowledge */
    ackMsg[0] = sRequestPwrLevel;
    ackMsg[1] = sCurrentPwrLevel;
```

```
      SMPL_Send(sLinkID, ackMsg, sizeof(ackMsg));

      /* Turn on RX. default is RX off. */
      SMPL_Ioctl( IOCTL_OBJ_RADIO, IOCTL_ACT_RADIO_RXON, 0);

      BSP_TURN_ON_LED1();
      SPIN_ABOUT_QUARTER_A_SECOND;
      BSP_TURN_OFF_LED1();
    }
  }

  static void tratarBotao1(void)
  {
    if(telaAtualDisplay < TELA_BEEP)
      trocarTelaInfoDisplay();
    else if (telaAtualDisplay == TELA_BEEP)
    {
      alterarEstadoBeep();
      telaInfoAtualDisplay--;
      trocarTelaInfoDisplay();
    }
    else if (telaAtualDisplay == TELA_SAIR_RODOVIA)
    {
      sairRodovia();
      exibeLCDSemRodovia();
    }
  }

  static void tratarBotao2(void)
  {
    if(telaAtualDisplay < TELA_BEEP)
    {
      exibeLCDAlterarBeep();
    }
    else if (telaAtualDisplay == TELA_BEEP)
    {
      exibeLCDSairRodovia();
    }
    else if (telaAtualDisplay == TELA_SAIR_RODOVIA){
      telaInfoAtualDisplay--;
      trocarTelaInfoDisplay();
    }
  }

  static void trocarTelaInfoDisplay(void)
  {
    uint8_t proximaTela = telaInfoAtualDisplay + 1;
    if(proximaTela >= TELA_BEEP)
      proximaTela = TELA_NOME_RODOVIA;
    exibirTelaInfo(proximaTela);
  }

  static void exibirTelaInfo(uint8_t codTela)
  {
    switch(codTela)
    {
    case TELA_NOME_RODOVIA:
      exibeLCDNomeRodovia();
      break;
    case TELA_VELOCIDADE:
      exibeLCDVelocidades();
```

```
    break;
  case TELA_BURACO:
    exibeLCDBuracos();
    break;
  case TELA_ACIDENTE_OBRA:
    exibeLCDAcidentesObras();
    break;
  case TELA_TRANSITO:
    exibeLCDTransito();
    break;
  case TELA_TEMPO:
    exibeLCDTempo();
    break;
  }
}

static void verificarMudancas(void)
{
  if(mudanca[TELA_NOME_RODOVIA])
  {
    beep();
    exibeLCDNomeRodovia();
  }
  else if((mudanca[TELA_VELOCIDADE]))
  {
    beep();
    exibeLCDVelocidades();
  }
  else if((mudanca[TELA_BURACO]))
  {
    beep();
    exibeLCDBuracos();
  }
  else if((mudanca[TELA_ACIDENTE_OBRA]))
  {
    beep();
    exibeLCDAcidentesObras();
  }
  else if((mudanca[TELA_TRANSITO]))
  {
    beep();
    exibeLCDTransito();
  }
  else if((mudanca[TELA_TEMPO]))
  {
    beep();
    exibeLCDTempo();
  }
}

static void exibeLCDNomeRodovia(void)
{
  telaAtualDisplay = TELA_NOME_RODOVIA;
  telaInfoAtualDisplay = TELA_NOME_RODOVIA;
  strcpy(linha_1_display, "------Rodovia-------");
  strcpy(linha_2_display, (char *)entradaSaidaMsg + ES_NOME_ROD_INICIAL);
  escreve_string_display(linha_1_display, linha_2_display);
}

static void exibeLCDVelocidades(void)
{
```

```
  telaAtualDisplay = TELA_VELOCIDADE;
  telaInfoAtualDisplay = TELA_VELOCIDADE;
  strcpy(linha_1_display, "Vel. Maxima: ");
  inteiroParaASCII(velocidadeMaxMsg[VM_H], velocidadeMaxMsg[VM_L]);
  strcat(linha_1_display, numeroASCII);
  strcat(linha_1_display, "km/h");
  strcpy(linha_2_display, "Vel. Atual: ");
  inteiroParaASCII(0, velocidadeAtual);
  strcat(linha_2_display, numeroASCII);
  strcat(linha_2_display, "km/h");
  escreve_string_display(linha_1_display, linha_2_display);
}

static void exibeLCDBuracos(void)
{
  telaAtualDisplay = TELA_BURACO;
  telaInfoAtualDisplay = TELA_BURACO;
  strcpy(linha_1_display, "------Buraco");
  strcpy(linha_2_display, "km:");
  inteiroParaASCII(buracoMsg[B_BURACOS_INICIO + B_KM_INICIAL_H_REL],
                   buracoMsg[B_BURACOS_INICIO + B_KM_INICIAL_L_REL]);
  strcat(linha_2_display, numeroASCII);
  if((buracoMsg[B_BURACOS_INICIO + B_KM_FINAL_H_REL]<<8) +
     buracoMsg[B_BURACOS_INICIO + B_KM_FINAL_L_REL] == 0)
  {
    strcat(linha_1_display, "--------");
  }
  else
  {
    strcat(linha_1_display, "s-------");
    strcat(linha_2_display, "-");
    inteiroParaASCII(buracoMsg[B_BURACOS_INICIO + B_KM_FINAL_H_REL],
                     buracoMsg[B_BURACOS_INICIO + B_KM_FINAL_L_REL]);
    strcat(linha_2_display, numeroASCII);
  }
  strcat(linha_2_display, " fx:");
  inteiroParaASCII(0, buracoMsg[B_BURACOS_INICIO + B_FAIXA_INICIAL_REL]);
  strcat(linha_2_display, numeroASCII);
  if(buracoMsg[B_BURACOS_INICIO + B_FAIXA_FINAL_REL] != 0)
  {
    strcat(linha_2_display, "-");
    inteiroParaASCII(0, buracoMsg[B_BURACOS_INICIO + B_FAIXA_FINAL_REL]);
    strcat(linha_2_display, numeroASCII);
  }
  escreve_string_display(linha_1_display, linha_2_display);
}

static void exibeLCDAcidentesObras(void)
{
  telaAtualDisplay = TELA_ACIDENTE_OBRA;
  telaInfoAtualDisplay = TELA_ACIDENTE_OBRA;
  if(acidenteObraMsg[AO_ACID_OBR_INICIO + AO_ACIDENTE_OBRA_REL] ==
AO_ACIDENTE_VAL)
    strcpy(linha_1_display, "------Acidente------");
  else strcpy(linha_1_display, "--------Obra--------");
  strcpy(linha_2_display, "km:");
  inteiroParaASCII(acidenteObraMsg[AO_ACID_OBR_INICIO +
AO_KM_INICIAL_H_REL],
                   acidenteObraMsg[AO_ACID_OBR_INICIO +
AO_KM_INICIAL_L_REL]);
  strcat(linha_2_display, numeroASCII);
```

```
  if((acidenteObraMsg[AO_ACID_OBR_INICIO + AO_KM_FINAL_H_REL]<<8) +
     acidenteObraMsg[AO_ACID_OBR_INICIO + AO_KM_FINAL_L_REL])
  {
    strcat(linha_2_display, "-");
    inteiroParaASCII(acidenteObraMsg[AO_ACID_OBR_INICIO +
AO_KM_FINAL_H_REL],
                     acidenteObraMsg[AO_ACID_OBR_INICIO +
AO_KM_FINAL_L_REL]);
    strcat(linha_2_display, numeroASCII);
  }
  strcat(linha_2_display, " fx:");
  inteiroParaASCII(0, acidenteObraMsg[AO_ACID_OBR_INICIO +
AO_FAIXA_INICIAL_REL]);
  strcat(linha_2_display, numeroASCII);
  if(acidenteObraMsg[AO_ACID_OBR_INICIO + AO_FAIXA_FINAL_REL] != 0)
  {
    strcat(linha_2_display, "-");
    inteiroParaASCII(0, acidenteObraMsg[AO_ACID_OBR_INICIO +
AO_FAIXA_FINAL_REL]);
    strcat(linha_2_display, numeroASCII);
  }
  escreve_string_display(linha_1_display, linha_2_display);
}

static void exibeLCDTransito(void)
{
  telaAtualDisplay = TELA_TRANSITO;
  telaInfoAtualDisplay = TELA_TRANSITO;
  strcpy(linha_1_display, "Trans:km ");
  inteiroParaASCII(transitoMsg[TR_KM_INICIAL_H],
transitoMsg[TR_KM_INICIAL_L]);
  strcat(linha_1_display, numeroASCII);
  if((transitoMsg[TR_KM_FINAL_H]<<8) + transitoMsg[TR_KM_FINAL_L] != 0)
  {
    strcat(linha_1_display, "-");
    inteiroParaASCII(transitoMsg[TR_KM_FINAL_H],
transitoMsg[TR_KM_FINAL_L]);
    strcat(linha_1_display, numeroASCII);
  }
  if((transitoMsg[TR_KM_ALT_H]<<8) + transitoMsg[TR_KM_FINAL_L] == 0)
    strcpy(linha_2_display, "Nao ha alternativa");
  else
  {
    strcpy(linha_2_display, "Alt:km ");
    inteiroParaASCII(transitoMsg[TR_KM_ALT_H], transitoMsg[TR_KM_ALT_L]);
    strcat(linha_2_display, numeroASCII);
    strcat(linha_2_display, " ");
    strcat(linha_2_display, (char *)transitoMsg + TR_NOME_ROD_INICIAL);
  }
  escreve_string_display(linha_1_display, linha_2_display);
}

static void exibeLCDTempo(void)
{
  telaAtualDisplay = TELA_TEMPO;
  telaInfoAtualDisplay = TELA_TEMPO;
  strcpy(linha_1_display, "-------Tempo--------");
  strcpy(linha_2_display, (char *)tempoMsg + TE_DESCRICAO_INICIAL);
  escreve_string_display(linha_1_display, linha_2_display);
}
```

```
static void exibeLCDAlterarBeep(void)
{
  telaAtualDisplay = TELA_BEEP;
  if(!estadoBeep)
    strcpy(linha_1_display, "L");
  else strcpy(linha_1_display, "Desl");
  strcat(linha_1_display, "igar beep?");
  strcpy(linha_2_display, msgBotoesConfig);
  escreve_string_display(linha_1_display, linha_2_display);
}

static void exibeLCDSairRodovia(void)
{
  telaAtualDisplay = TELA_SAIR_RODOVIA;
  strcpy(linha_1_display, "Sair da rodovia?");
  strcpy(linha_2_display, msgBotoesConfig);
  escreve_string_display(linha_1_display, linha_2_display);
}

static void exibeLCDSemRodovia(void)
{
  telaAtualDisplay = TELA_SEM_RODOVIA;
  strcpy(linha_1_display, "Rodovia sem suporte");
  strcpy(linha_2_display, "Vel. Atual: ");
  inteiroParaASCII(0, velocidadeAtual);
  strcat(linha_2_display, numeroASCII);
  strcat(linha_2_display, "km/h");
  escreve_string_display(linha_1_display, linha_2_display);
}

static void atualizarVelocidadeAtual(void)
{
  velocidadeAtual = 10;
}

static void inicializa_beep(void)
{
  P1DIR |= 0x80;
  estadoBeep = 1;
  beep();

}

static void beep(void)
{
  if(estadoBeep)
  {
    PORT_BEEP |= 0x80;
    SPIN_ABOUT_HALF_SECOND;
    PORT_BEEP &= 0x7F;

  }
}

static void alterarEstadoBeep(void)
{
  if(!estadoBeep){
    estadoBeep = 1;
    beep();
  }
  else estadoBeep = 0;
```

```
}

static void sairRodovia(void)
{
  zerarVetor(entradaSaidaMsg, ES_LENGTH_VAL);
  zerarVetor(velocidadeMaxMsg, VM_LENGTH_VAL);
  zerarVetor(buracoMsg, B_LENGTH_VAL);
  zerarVetor(acidenteObraMsg, AO_LENGTH_VAL);
  zerarVetor(transitoMsg, TR_LENGTH_VAL);
  zerarVetor(tempoMsg, TE_LENGTH_VAL);
}

static void inteiroParaASCII(uint8_t int_H, uint8_t int_L)
{
  uint8_t posicaoNumASCII = 0;
  uint8_t posicaoAux = 0;
  int valor = (int_H<<8) + int_L;
  char vetorAux[6];

  if(valor == 0){
    numeroASCII[0] = '0';
    numeroASCII[1] = '\0';
  }
  else{
    while(valor != 0)
    {
      vetorAux[posicaoAux] = '0' + valor%10;
      valor = valor/10;
      posicaoAux++;
    }

    while(posicaoAux > 0)
    {
      posicaoAux--;
      numeroASCII[posicaoNumASCII] = vetorAux[posicaoAux];
      posicaoNumASCII++;
    }

    numeroASCII[posicaoNumASCII] = '\0';
  }
}

static uint8_t copiarVetoresDetectarMudanca(uint8_t vetor_1[], uint8_t
vetor_2[],
                                            uint8_t tamanho)
{
  uint8_t i;
  uint8_t mudanca = 0;
  for(i = 0; i < tamanho; i++)
  {
    if (!mudanca && vetor_1[i] != vetor_2[i])
      mudanca = 1;
    vetor_1[i] = vetor_2[i];
  }
  return mudanca;
}

static void zerarVetor(uint8_t *vetor, uint8_t tamanho)
{
  uint8_t i;
  for(i = 0; i < tamanho; i++)
```

```
  {
    *(vetor + i) = 0;
  }
}
```

8.4.3 Display.h

```
#include "bsp.h"

/* Configurações utilizadas para comunicação com o display de LCD */
#define PORT_CONTROL_DISPLAY          P1
#define RS_PIN                        4
#define RW_PIN                        5
#define E_PIN                         6
#define PORT_DATA_DISPLAY             P0

void       inicializa_display(void);
void       escreve_string_display(char string_1[], char string_2[]);
```

8.4.4 Display.c

```
#include <string.h>
#include "configuracoes.h"
#include "display.h"
#include "mrfi.h"
#include "nwk_types.h"

#define WAIT_DISPLAY_TIME                NWK_DELAY(5)

static void escreve_byte_display(uint8_t instr, int tipo_dado);

void inicializa_display(void)
{
  P1DIR |= 0x70;
  P0DIR = 0xFF;
  escreve_byte_display(0x38, 0);
  WAIT_DISPLAY_TIME;
  escreve_byte_display(0x38, 0);
  WAIT_DISPLAY_TIME;
  escreve_byte_display(0x38, 0);
  WAIT_DISPLAY_TIME;
  escreve_byte_display(0x06, 0);
  WAIT_DISPLAY_TIME;
  escreve_byte_display(0x0C, 0);
  WAIT_DISPLAY_TIME;
  escreve_byte_display(0x01, 0);
  WAIT_DISPLAY_TIME;
  escreve_byte_display(0x80, 0);
}

void escreve_string_display(char string_1[], char string_2[])
{
  uint8_t i, stringSize;

  /* Limpa o display */
  escreve_byte_display(0x01, 0);
  WAIT_DISPLAY_TIME;

  /* Escreve na primeira linha do display */
  if(string_1 != 0)
  {
```

```
    escreve_byte_display(0x80, 0);
    WAIT_DISPLAY_TIME;
    stringSize = strlen(string_1);
    for(i = 0; i < stringSize; i++)
    {
      escreve_byte_display(string_1[i], 1);
      WAIT_DISPLAY_TIME;
    }
  }

  /* Escreve na segunda linha do display */
  if(string_2 != 0)
  {
    escreve_byte_display(0xC0, 0);
    WAIT_DISPLAY_TIME;
    stringSize = strlen(string_2);
    for(i = 0; i < stringSize; i++)
    {
      escreve_byte_display(string_2[i], 1);
      WAIT_DISPLAY_TIME;
    }
  }
}

static void escreve_byte_display(uint8_t instr, int tipo_dado)
{
  PORT_CONTROL_DISPLAY &= ~(BV(E_PIN));
  if(tipo_dado)
    PORT_CONTROL_DISPLAY |= BV(RS_PIN);
  else PORT_CONTROL_DISPLAY &= ~(BV(RS_PIN));
  PORT_CONTROL_DISPLAY &= ~(BV(RW_PIN));
  PORT_DATA_DISPLAY = instr;
  PORT_CONTROL_DISPLAY |= BV(E_PIN);
  WAIT_DISPLAY_TIME;
  PORT_CONTROL_DISPLAY &= ~(BV(E_PIN));
  PORT_CONTROL_DISPLAY |= BV(RW_PIN);
}
```