DOUGLAS GALHARDO PASSATUTO

SISTEMA AUTOMATIZADO MULTISSENSORIAL MICROPROCESSADO PARA CONTROLAR A INGESTÃO ALIMENTAR DO GADO EM CONFINAMENTO.

Trabalho de Conclusão de Curso apresentado

à Escola de Engenharia de São Carlos,

da Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em

Sistemas de Energia e Automação

ORIENTADOR: Evandro Luís Linhari Rodrigues

São Carlos

2012

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Passatuto, Douglas Galhardo

P286s Sistema automatizado multissensorial microprocessado para controlar a ingestão alimentar do gado em confinamento. / Douglas Galhardo Passatuto. –- orientador Evandro Luís Linhari Rodrigues. -- São Carlos, 2012.

Monografia (Graduação em Engenharia Elétrica com ênfase em Sistemas de Energia e Automação) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2012.

FOLHA DE APROVAÇÃO

Nome: Douglas Galhardo Passatuto

Título: "Sistema Automatizado Multissensorial Microprocessado para Controlar a Ingestão Alimentar do Gado em Confinamento"

Trabalho de Conclusão de Curso defendido e aprovado em 20/07/2012,

com NOTA 9,2 (Nove, Dois), **pela comissão julgadora:**

Prof. Associado Evandro Luís Linhari Rodrigues (Orientador) - EESC/USP

Prof. Dr. Marcelo Andrade da Costa Vieira - EESC/USP

Profa. Assistente Luiza Maria Romeiro Codá - EESC/USP

Prof. Associado Homero Schiabel
Coordenador da CoC-Engenharia Elétrica
EESC/USP

AGRADECIMENTOS

Agradeço a Deus, pai nosso que está no céu e me sempre guarda. Sem ele, certamente não teria superado as dificuldades e intercorrências durante o percurso.

A minha família, provas vivas que Deus existe. A estes meu obrigado pela esperança em mim depositada.

A minha esposa, Gabriela, pela paciência, incentivo e amor dispensados. Especialmente pela linda luz que trouxe ao mundo, nosso filho Carlos Eduardo, e que nos tem alegrado e mostrado a beleza da vida, da inocência, do amor gratuito. Aos meus sogros que auxiliam na educação do meu filho na minha ausência. Em especial Carlos e Shirlei, que se tornaram meus segundos pais após o nascimento daquele.

Ao professor Dr. Evandro Luís Linhari Rodrigues pela paciência e compreensão.

Aos amigos trazidos da infância e por quem guardo sinceros sentimentos de carinho, especialmente: Rodrigo, Erivaldo, Fernando, Emílio, Roger e Nelson.

Aos amigos que ganhei na universidade, em especial os que moraram ou estiveram mais próximos a mim: Juliano, Leandro, Sílvio, Marcus, Maicon, Rafael e Bruno.

À empresa Korth pela oportunidade e aos meus amigos desta empresa com quem passo maior parte do dia e que propiciam um ambiente agradável. Especial ao Marcelo, que depositou em mim sua confiança.

À Escola de Engenharia de São Carlos e a todos que de forma direta ou indireta contribuíram para minha formação.

RESUMO

Atualmente utilizado em celulares, microcomputadores e outros equipamentos de objetivos diversos, o *bluetooth* é um protocolo capaz de transmitir pacotes de dados *on-line*. Já a tecnologia RFID (*Radio-Frequency Identification*), possui procedimento para identificação por sinais de radio onde os dados são recuperados de *transponders* (*Transmitter-responder*), também chamados de etiquetas ou *tags*. Considerando que essas tecnologias estão consolidadas, o presente projeto consiste no desenvolvimento de um sistema com *software* embarcado, explorando RFID, que seja capaz de coletar informações automaticamente sobre a alimentação de gado confinado e transmiti-las para um computador via *bluetooth*. Para mapear o confinamento, utilizaram-se *tags* com valores previamente definidos e gravados em sua memória. Dessa maneira, foi possível identificar o posicionamento físico do vagão distribuidor de alimento por meio da leitura imediata do *tag*. Informações como quantidade de alimento ingerida por animal, número de refeições por dia e temperatura ambiente em cada refeição, foram transferidas para o computador e armazenadas num banco de dados. Além de coletar informações de forma automática, o equipamento, programado para ser proativo, auxiliou o tratador na distribuição do alimento deixado aos animais. Foram realizadas comparações entre o sistema automatizado e um método convencional e verificou-se a redução na quantidade de insumos utilizados, aumento da engorda, redução no tempo de trato, economia de combustível e prolongamento na vida útil do maquinário.

Palavras–chave: *Bluetooth*, UHF, *Tag*, RFID, sensores sem fio, automação de confinamento.

ABSTRACT

Commonly used on cell phones, microcomputers and other general devices, bluetooth is a protocol able to transfer data packet online. RFID, *Radio Frequency Identification, is a method that uses radio signals in order to identify data retrieved from transponders known as tags. Considering these technologies as largely tested and used, this project aim is to develop an equipment composed with an embedded software and using RFID to collect information on livestock feeding and transfer it to the computer by Bluetooth. Tags with values previously defined and stored in its memory were used to map the confinement. Thus, it allows identify the physical placement of the feed distributor truck through the immediate reading tag. Such information as the amount of food ingested by each animal, meals per day and the temperature on each occasion were transferred to the computer and stored in a database. Besides collecting information, this equipment will proactively assist distributing food left for the animals. This system was compared with the conventional and it was responsible for the reduction in amount of feedstock used, animal weight increase, reduction in treatment time, fuel economy and increasing the work life of machines.*

Keywords*: Bluetooth, UHF, Tag, RFID, wireless sensors, confinement automation.*

LISTA DE SIGLAS, SÍMBOLOS E UNIDADES

A/D	Conversor Analógico/Digital
PC	*Personal Computer*
RFID	*Radio-Frequency Identification*
RTOS	*Real-Time Operating System*
UHF	*Ultra High Frequency*
RS232	Padrão de comunicação
SCI	*Serial Comunications Interface*
SPI	*Serial Peripheral Interface*
PWM	*Pulse Widht Modulation*
%	por cento
p	pico
m	mili
k	kilo
M	Mega
G	Giga
bps	bits por segundo
Hz	*Hertz*
A	*Ampère*
F	*Faraday*
g	grama
s	segundo
V	Volt
Ω	Ohm
ºC	g*rau Celsius*

LISTA DE FIGURAS

SUMÁRIO

1 – INTRODUÇÃO

Confinamento é o método de criação bovina para engorda intensiva em que os animais são acomodados em piquetes e tratados e alimentados com tudo de que necessitam para que se desenvolvam o mais rapidamente possível de modo a acelerar e otimizar a produção, seja de corte ou leiteira. Este método pode ser subdividido em duas etapas isoladas: a fabricação e o descarregamento de ração. Este projeto, entretanto, estreita-se para a automação do descarregamento da ração.

A figura 1 ilustra o mapa de um confinamento.

Figura 1- Mapa do confinamento

Fonte: Confinamento - campo de testes.

Os piquetes são lotes de terra de área variada, tipicamente compreendida entre 12 a 15m² por cabeça[1], delimitados geralmente por cerca de arame. Nestes lotes o gado recebe alimentação através do cocho. A figura 2 ilustra o descarregamento de ração no cocho.

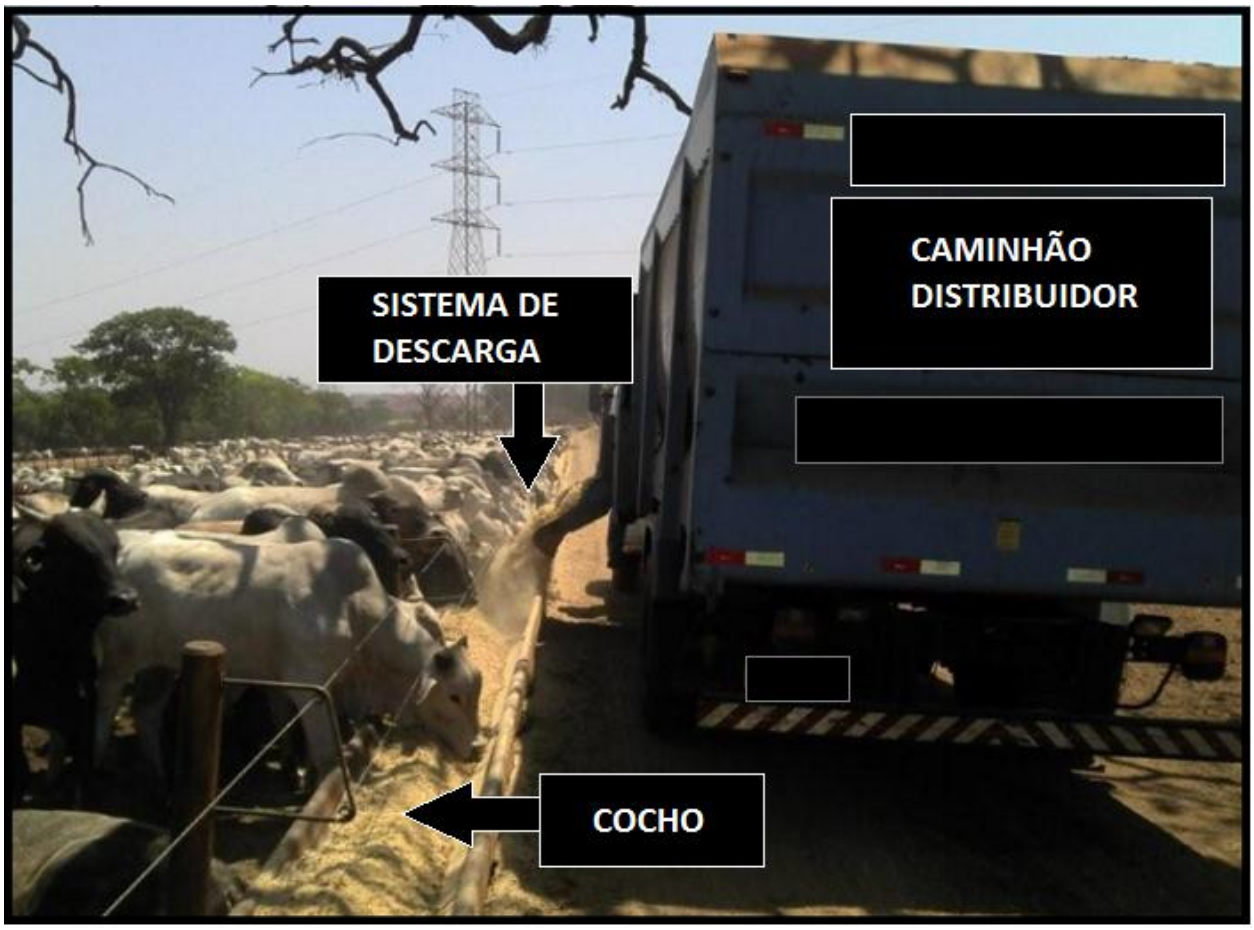

Figura 2 - Descarregamento da ração

1.1 – MOTIVAÇÃO

Há, no Brasil, basicamente três motivos para confinar:

- **Aumento do ganho de peso em relação aos incentivos nutricionais:** Os apontamentos nutrologistas atuais são embasados em estudos aprofundados de resposta de ganho de peso do animal com relação ao incentivo nutricional. Estes estudos direcionam as empresas nutricionais a orientarem seus clientes para um ponto ótimo de gasto com insumos relativo à melhor taxa de ganho de peso durante o período de confinamento. Nesse foco, foi desenvolvido pela ESALQ – USP, em parceria com a IntegraSoftware®, um software, RLM® – Ração de

Lucro Máximo, capaz de estabelecer a relação entre os valores nutricionais de diferentes ingredientes minimizando o custo da fabricação da ração e tornando-a mais eficiente[2].

- **Condições climáticas:** O período de estiagem do Brasil, compreendido pelas estações outono e inverno, prejudica a criação a pasto. Sem a chuva não há crescimento do pasto suficiente para garantir a demanda nutricional do gado e este é transferido para o confinamento.

- **Rastreabilidade da carne produzida:** Para garantir a qualidade dos alimentos fornecidos, grandes empresas de produção e comercialização de alimentos à base de carne bovina, Brasil Foods®, JBS®, Marfrig® e Minerva®, exigem a rastreabilidade da alimentação do gado. Durante a estadia no confinamento, é possível estabelecer um banco de dados que relacione a identificação do animal a sua alimentação.

A partir destes fatos, fica evidenciada a necessidade de controlar a ingestão de insumos do gado durante o período confinado. A rastreabilidade da alimentação permite ao pecuarista gerar relatórios para o consumidor e também comprovar a eficácia da dieta oferecida ao gado.

1.2 – PROPOSIÇÃO

A composição da dieta oferecida a cada animal durante o dia é adaptativa e depende dos fatores climáticos e da composição e distribuição anteriores. Neste cenário, é fundamental construir uma base de dados concisa relatando como é realizada cada descarga e corrigindo a próxima.

O projeto apresentado visa aprimorar a técnica de despejo de alimento nos cochos, indicando a quantidade exata que deve ser deixada em cada piquete, gerando relatórios do que foi realizado. Dados como data, hora, peso despejado por piquete e temperatura ambiente são coletados e transferidos automaticamente para um computador. O sistema multissensorial, ilustrado pela figura 3, foi composto por um equipamento embarcado que utiliza identificação por radiofrequência, doravante RFID, para identificar os piquetes e *bluetooth* para transferir os dados ao computador. Além disso, comunica-se com um indicador de peso já existente no caminhão de descarga da ração para gerar os relatórios através da diferença de peso entre início e final de cada curral.

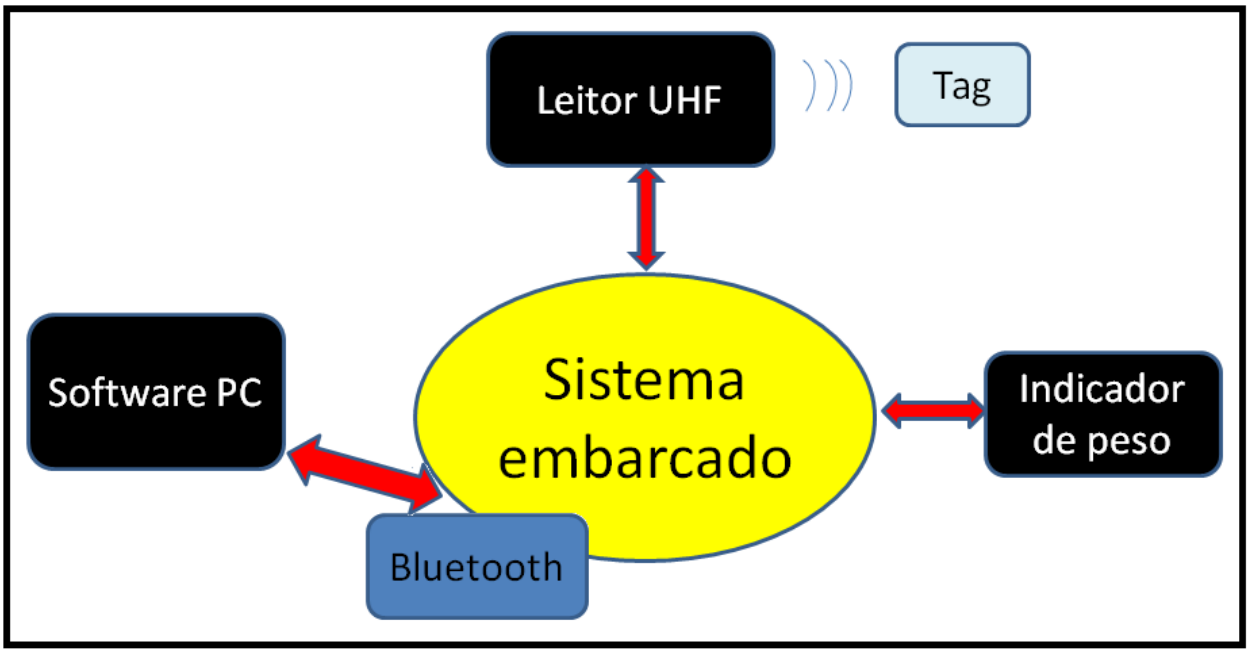

Figura 3 – Sistema multissensorial automatizado

Para garantir uma distribuição homogênea e adaptativa de ração no decorrer do dia, o sistema automatizado deve identificar o curral onde o caminhão distribuidor se encontra e informar ao tratador o montante exato a ser deixado em cada piquete. Ao final de cada piquete, um relatório é gerado e gravado na memória do equipamento. Quando o caminhão regressa à fábrica para receber mais ração, o *software* embarcado se comunica com um *software* computacional para descarregar os dados.

1.3 – ORGANIZAÇÃO DO TRABALHO

O trabalho foi organizado da seguinte maneira:

- Capítulo 2: Abordagem sobre os fundamentos teóricos que embasaram o projeto;
- Capítulo 3: Apresentação da metodologia empregada e explanação sobre o funcionamento do sistema embarcado e detalhamento das tecnologias utilizadas;
- Capítulo 4: Descrição dos resultados dos testes realizados relativos ao sistema automatizado;
- Capítulo 5: Conclusão do projeto com informações sobre viabilidade e vantagens da implantação do sistema proposto e informações sobre trabalhos futuros.

2 – FUNDAMENTAÇÃO TEÓRICA

O confinamento é geralmente utilizado na fase de terminação do gado. Neste período, cerca de 100 dias, o animal aumenta sua massa em 150 kg e obtém o acabamento da sua carcaça. A qualidade da carcaça produzida depende não só do potencial do gado, mas também da alimentação nesta fase final da vida do gado[1].

2.1 – ESTADO DA ARTE

O método usado em confinamento para tratar o animal consiste num distribuidor de ração dotado de um sistema de descarga capaz de transportar o alimento e despejá-lo no cocho. No entanto, esse procedimento não é monitorado e fica a cargo do tratador decidir a quantidade de ração que deve despejar em cada piquete, podendo fazê-lo em maior ou menor escala que o necessário, comprometendo a aferição da eficácia da dieta, culminando no ganho de peso abaixo do estimado ou no aumento do consumo de insumos na forma de desperdício. Além disso, para o *software* de gerenciamento controlar o estoque, o motorista para seu trabalho ao final de cada piquete e anota numa planilha o montante deixado naquele local. Estas pausas reduzem a eficiência de seu trabalho, aumentando o tempo que ele despende para executar suas funções; diminui a vida útil do maquinário, gastando mais freio e embreagem; aumenta o consumo de combustível, deixando o distribuidor ligado por mais tempo.

O confinamento diminui seu lucro se não garantir o ponto ótimo da engorda com relação aos incentivos nutricionais e aumenta seu custo operacional ao controlar seu estoque de insumos.

No modelo proposto, o tratador deve se basear pela quantidade de ração programada para cada piquete, levando o confinamento ao ponto ótimo de engorda. Para isso, os valores

serão transmitidos para o equipamento por *bluetooth* dispensando anotações em papel ou cabos para comunicação entre o PC e o equipamento.

2.2 – BLUETOOTH

Em 1994 a empresa Ericson® estudava a viabilidade de se desenvolver uma tecnologia que permitisse aos seus celulares trocar informações com outros dispositivos utilizando sinais de rádio. Após conclusão sobre a viabilidade, inicia-se o projeto com a tecnologia em 1997. Com a evolução do projeto, surge em 1998 o consórcio ***bluetooth*** SIG(*Special Interest Group*), formado pelas empresas Ericson®, Nokia®, IBM®, Toshiba® e Intel®. A partir disto, iniciou-se a padronização do protocolo que permitia o uso e garantia a operabilidade da tecnologia nos mais variados dispositivos[3]. Seguindo esta lógica, é possível trocar informações entre dois dispositivos afastados fisicamente através do protocolo *bluetooth*.

2.2.1 – TOPOLOGIA DA REDE *BLUETOOTH*

Cada dispositivo de comunicação de radiofrequência se enquadra em dois tipos possíveis de redes: *Piconet* ou *Scatternet*.

- ***Piconet:*** O dispositivo que realizou a conexão torna-se *master* enquanto os demais dispositivos assumem o papel de slave. O *master* tem função de regulamentar a transmissão e o sincronismo da rede. Nesta topologia, a rede suporta até 8 dispositivos, sendo 1 *master* e 7 *slaves;*

- ***Scatternet:*** Nesta topologia, uma rede *piconet* se comunica a outra dentro de uma faixa de alcance. Um dispositivo *slave*, neste caso, pode ocupar mais de uma *piconet*. Sendo assim, cada *slave* só pode ter um *master* associado.

A figura 4 ilustra a topologia de uma rede *scatternet*:

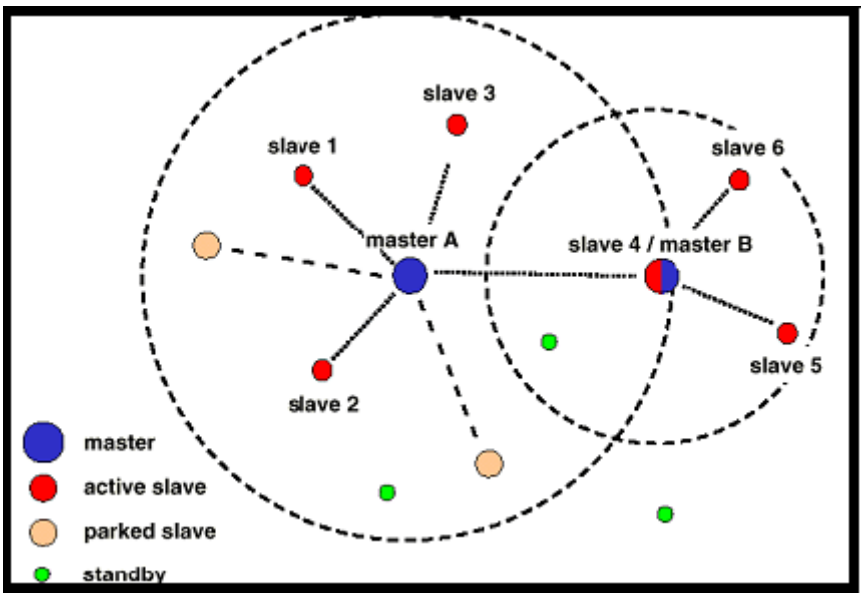

Figura 4 - Topologia scatternet

Fonte: http://www.developer.nokia.com/Community/Wiki/File:Piconet.png.

2.2.2 – CLASSES DO *BLUETOOTH*

No que tange à distância para a comunicação entre dois dispositivos, o *bluetooth* pode ser classificado em três classes:

Classe	Potência máxima permitida (mW/dBm)	Alcance (Aproximadamente)
Classe 1	100 mW (20 dBm)	até 100 metros
Classe 2	2.5 mW (4 dBm)	até 10 metros
Classe 3	1 mW (0 dBm)	~ 1 metro

Figura 5 - Alcance relativo à classe

Fonte: http://pt.wikipedia.org/wiki/Bluetooth.

2.3 – RFID

A utilização de radio frequência para fazer identificação, RFID, está evidentemente inserida no cotidiano. Citam-se exemplos como os sistemas de identificação de carros nos pedágios (SEM PARAR®) e de cartões em catracas de ônibus e metrôs (BILHETE ÚNICO®).

RFID, independentemente da faixa de operação, consiste em recuperar dados de um transponder através da excitação do *tag* por uma antena. O leitor de *tag*s emite uma onda eletromagnética com dados inseridos em uma portadora de radiofrequência modulada. O *tag* passivo utiliza a energia advinda da excitação para alimentar seu circuito e transmitir os dados que estão na sua memória. O *tag* ativo possui uma bateria interna que alimenta seu circuito e também responde à excitação.

As faixas de operação da radiofrequência são escolhidas de acordo com a aplicação. As aplicações típicas estão descritas na figura 6.

banda de freqüências	*designação*	*aplicações típicas*
3 - 30 kHz	very low frequency (VLF)	navegação em longas distâncias, comunicações submarinas
30 - 300 kHz	low frequency (LF)	navegação em longas distâncias, rádio farol marítimo
300 - 3.000 kHz	medium frequency (MF)	AM comercial, rádio marítimo, freqüências de emergência
3 - 30 MHz	high frequency (HF)	rádio amador, comunicações militares, *broadcasting* internacional, comunicações com aviões e navios em grandes distâncias
30 - 300 MHz	very high frequency (VHF)	televisão VHF, rádio FM, comunicação AM aérea, auxílio à navegação aérea
0,3 - 3 GHz	ultra high frequency (UHF)	televisão UHF, radar, enlaces de microondas, auxílio à navegação
3 - 30 GHz	super high frequency (SHF)	comunicações por satélite, enlaces de microondas e radar
30 - 300 GHz	extra high frequency (EHF)	radar, satélite experimental.
10^3 - 10^7 GHz	infravermelho, luz visível, ultravioleta	comunicações ópticas.

Figura 6 - Frequências e aplicações

Fonte: iaracaju.infonet.com.br/users/jfonseca/TELECOMUNICACOES.HTM.

3 – MATERIAIS E MÉTODOS

O sistema multissensorial projetado pode ser subdividido em 5 partes: Sistema embarcado, RFID em UHF, Indicador de peso, transmissão Bluetooth e Software computacional.

3.1 – SISTEMA EMBARCADO – HARDWARE

Para desenvolver o esquemático do circuito, foi utilizado o software *Altium*®. O sistema embarcado foi composto por microcontrolador, *bluetooth*, sensor de temperatura, memória *flash*, *beep*, *display*, *leds*, *realtime* e fontes de alimentação.

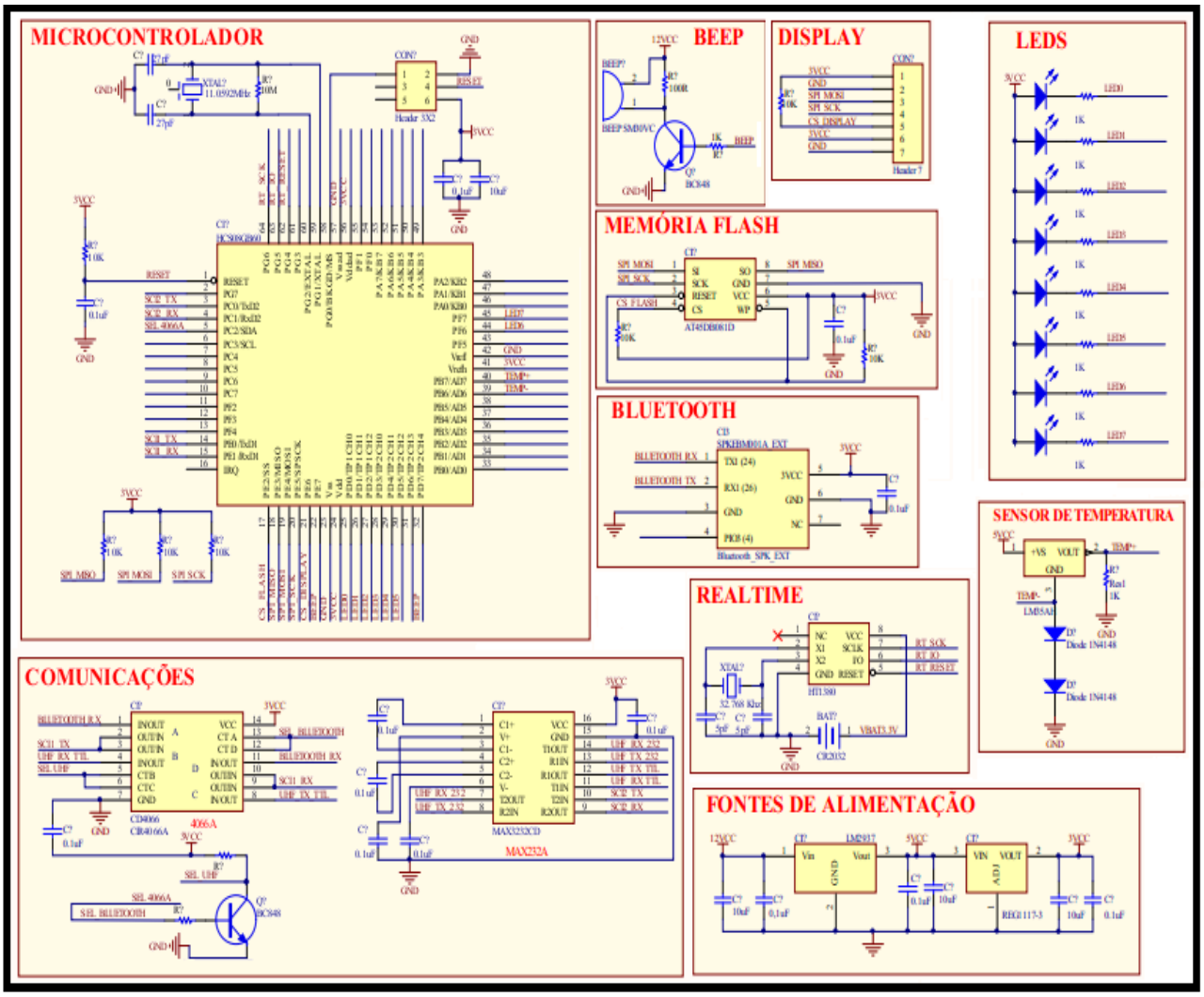

Figura 7 - Hardware

3.1.1 – MICROCONTROLADOR

O microcontrolador escolhido foi o HCS08GB60A, da *Freescale*®[6]. Este microcontrolador de 8 bits, utiliza arquitetura Von Neumann e filosofia de instruções CISC (*Complex Instruction Set Computer* ou computador com conjunto de instruções complexas). Sua unidade de processamento é capaz de operar a até 40 MHz. Possui 4 kbytes de RAM e 60 kbytes de FLASH[4]. Possui diversos periféricos e, entre eles, alguns foram preponderantes na sua escolha para o projeto:

- 2 SCI's – Serial assíncrona;
- 1 SPI – Serial síncrona;
- Conversor A/D de 10 bits – Conversor analógico digital;
- Timer e PWM – Temporizador e Modulação por largura de pulso;
- 7 Portas de entrada/saída.

Para o projeto, utilizaram-se 3 ligações seriais assíncronas. Porque este micro só possui 2, uma das seriais foi chaveada através de um circuito de seleção (*switch*), CD4066. As SCI's foram conectadas ao indicador de peso existente no caminhão, ao Leitor de *tag* UHF e ao *bluetooth*, como ilustrado na figura 8.

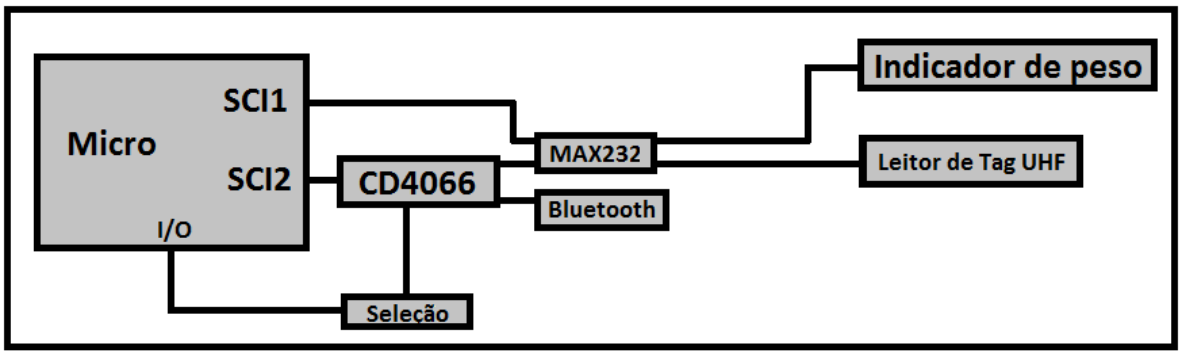

Figura 8 - Serial Assíncrona

O circuito CD4066 direciona as linhas de transmissão (TX) e recepção (RX) de dados da SCI1 para o *bluetooth* ou para o leitor de *tag* através de uma saída digital do microcontrolador ligada à base de um transistor capaz de selecionar o dispositivo desejado.

O *display* e a memória *flash* externa escolhidos utilizam comunicação:

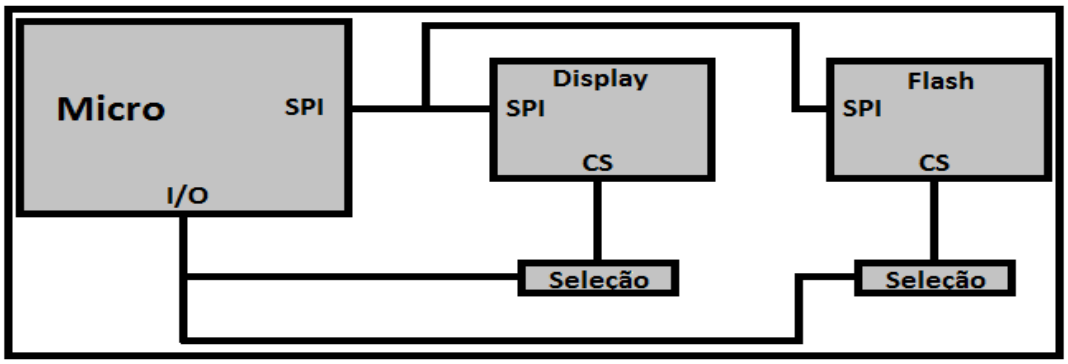

Figura 9 - Serial Síncrona

3.1.2 – *BLUETOOTH*

O módulo *bluetooth* escolhido foi o BlueMod+B2X®. Foi utilizada uma placa adaptadora para este dispositivo, conectando seus pinos de TX, RX, alimentação e terra à placa principal. O capítulo 3.5 traz informações sobre o módulo.

3.1.3 – SENSOR DE TEMPERATURA

A *National Semiconductor*® possui uma série de sensores de temperatura de precisão: LM35. O circuito integrado LM35 tem como característica tensão de saída linear proporcional à temperatura em graus Celsius medida (10mV/ºC). Além disso, não requer um circuito externo de calibração para fornecer curvas com precisão de +-1/4ºC para temperatura ambiente e +-3/4ºC em toda sua faixa de medição: -55ºC a 150ºC[7].

A topologia utilizada possui duas saídas: V+ e V-. Cada uma destas saídas foi ligada a um conversor A/D. Sendo assim, a tensão de saída é a diferença entre os valores obtidos em cada conversor A/D. A partir da tensão de saída, pode-se calcular a temperatura que é numericamente proporcional à tensão.

3.1.4 – OUTROS DISPOSITIVOS

Os demais dispositivos utilizados foram:

- Memória flash: AT45DB081D da *Atmel*®;
- Beep: SM30VC da WALTRONICA®;
- Display de 7 segmentos;
- *Led*'s;
- Realtime HT1380 da HOLTEK®;
- Fontes de alimentação LM2937 da National Semiconductor® e REG1117-3 da Texas Instruments®.

3.2 – SISTEMA EMBARCADO – SOFTWARE

Para o sistema embarcado foi desenvolvido um *software* em linguagem C utilizando a plataforma CodeWarrior® V10.2. O equipamento de automação trabalha conforme o fluxograma da figura 10. O código-fonte relativo a ele está disponibilizado no apêndice:

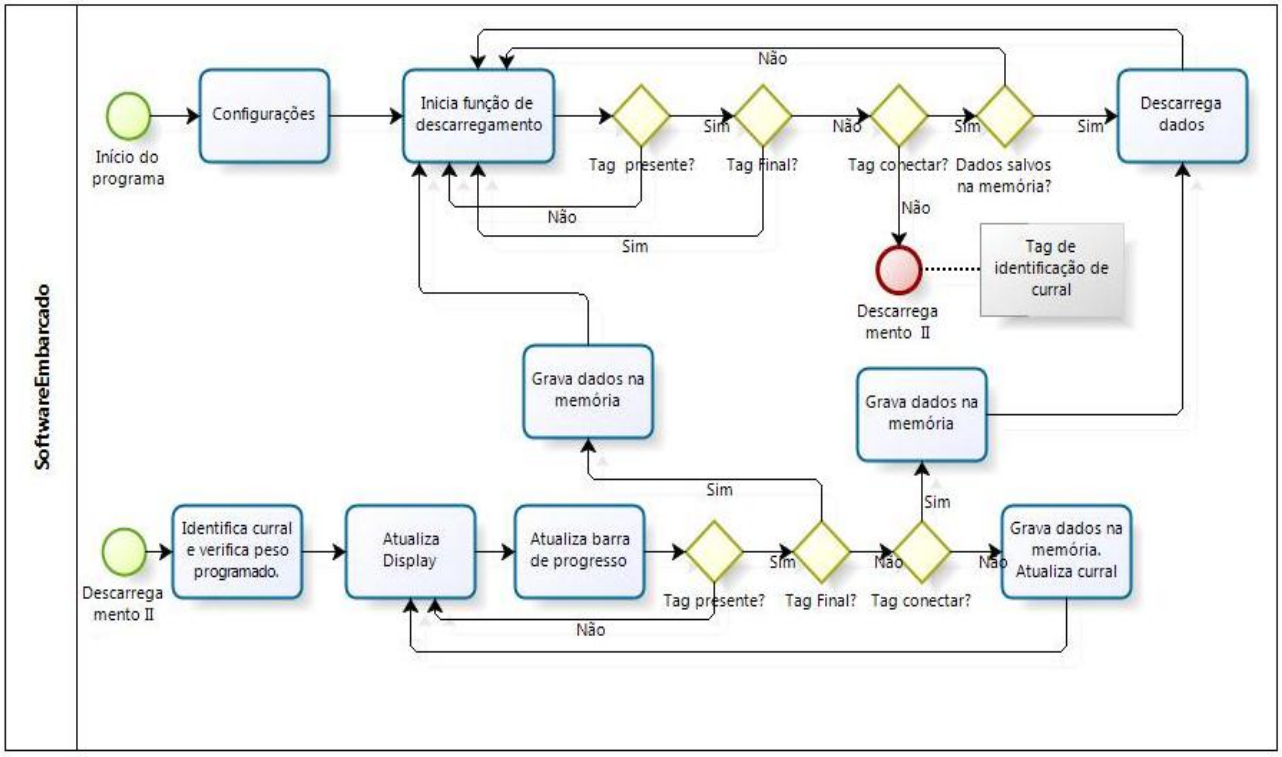

Figura 10 - Fluxograma de funcionamento do software embarcado

Ao iniciar, o equipamento é configurado e automaticamente entra no modo de descarga inicial. O sistema envia comando de leitura ao leitor UHF e aguarda a resposta. Quando a resposta for *tag* lido, o equipamento relaciona a resposta com o tipo de *tag*. Existem três tipos de *tag* previstos: *tag* normal da linha, *tag* finalizador de linha e *tag* de conexão.

- *Tag* normal: o equipamento faz a identificação do curral que este tag representa, atualiza o display com o montante a ser jogado e entra no modo de descarregamento efetivo;
- *Tag* finalizador: não tem alteração no funcionamento;
- *Tag* de conexão: se houver registros na memória, transfere os dados para o computador. Ao final deste procedimento, ou por timeout, volta para o modo de descarga inicial.

Quando o equipamento entra no estado de descarregamento efetivo, o *display* é atualizado constantemente. Esta atualização leva em conta o peso do caminhão no início do piquete, o peso atual do caminhão e o peso programado para o piquete. Além do *display*, a barra de progresso da descarga, composta por oito *leds,* também é atualizada. A barra de progresso indica a relação entre o peso já despejado e o peso programado para o piquete. Quando houver uma nova leitura de *tag*, um relatório será salvo para o piquete em aberto.

- *Tag* normal: os dados no novo curral serão exibidos no display e o equipamento permanece no modo de descarga efetivo;
- *Tag* finalizador: o trato será encerrado e o caminhão volta ao estado de descarga inicial;
- *Tag* de conexão: o trato será encerrado e o equipamento se comunica com o PC e transmite seus dados. Ao final da transmissão, ou por timeout, o equipamento volta ao estado de descarga inicial.

3.3 – LEITOR E *TAG* UHF

A frequência UHF, compreendida entre 300 MHz e 3 GHz, foi escolhida para o projeto para não haver interferência entre o *tag* UHF e o *tag* de identificação animal, afixado na orelha do animal, comumente na faixa de 134,5 kHz, que é amplamente utilizado no Brasil para o rastreamento e o manejo do gado. A distância de leitura que esta tecnologia proporciona, maior que 10 m, também foi determinante na escolha da frequência UHF.

3.3.1 – O LEITOR UHF

Utilizou-se o leitor UHF integrado VI-85 da empresa *Vanch*®[9]. Este produto possui leitor e antena integrados. Este leitor se comunica por comando serial com o *software* embarcado.

De acordo com o fabricante, a figura 11 caracteriza o leitor utilizado:

Model(Order Code)	VI-85	
Physical Characteristics	Dimensions	285(L)*205(W)*77(H)mm
	Weight	2KG
Environment	Operating Temp	-20°C to 55°C (-4°F to 131°F)
	Storage Temp	-40°C to 80°C (-40°F to 176°F)
	Humidity	10% to 95%
	non-condensing Enclosure	IP-65
Antenna Connection	1 SMA (Female/Inbuilt Antenna)	
Power	External 110 - 240 VAC auto-ranging RFID	
Frequency Ranges	902 or 928 MHz band (We can do as per your requirements)	
RF power control	20-30dBm	
Tag Air Interfaces	EPC Class 1 Gen 2 or ISO 18000-6B	
Communication Interface	RS232/RS485;Wiegand26/34	

Figura 11 – Especificações do leitor VI-85

Fonte: www.vanch.cn/en/products_view.asp?id=552

O leitor utilizado possui um protocolo de comunicação padrão que traz em sua estrutura:

- Preâmbulo: indica o início da comunicação;

- Comprimento: indica a quantidade de bytes da mensagem;
- Comando: indica o tipo de mensagem;
- Parâmetros ou dados: pacote de mensagem útil;
- *Checksum*: byte utilizado para conferência dos dados anteriores.

O principal comando utilizado foi o de leitura de *tags*. Ao enviar esse comando para o leitor integrado, 0xEE, os dados recebidos seguem a estrutura da figura 12:

Quantidade de *tags*	0x02
Número de *words* do primeiro *tag*	0x02
Primeiro byte do primeiro *word*	0x41 ou 'A'
Segundo byte do primeiro *word*	0x42 ou 'B'
Primeiro byte do segundo *word*	0x43 ou 'C'
Segundo byte do segundo *word*	0x44 ou 'D'
Número de *words* do segundo *tag*	0x01
Primeiro byte do primeiro *word*	0x41 ou 'A'
Segundo byte do primeiro *word*	0x42 ou 'B'

Figura 12 - Estrutura dos dados do tag

Interpretação dos dados:

- Foram lidos dois tags;
- O primeiro tag possui dois words (word = 2 bytes);
- Memória do primeiro: 0x41, 0x42, 0x43, 0x44;
- O segundo tag possui um word:
- Memória do segundo: 0x41, 0x42.

É importante observar que dois comandos consecutivos devem ser enviados ao leitor respeitando o tempo mínimo de 50 ms. Esse tempo é chamado *deadline.* A figura 13, capturada com um osciloscópio, ilustra a ocorrência quando se usa o espaçamento de 40 ms entre dois envios de comando para o leitor:

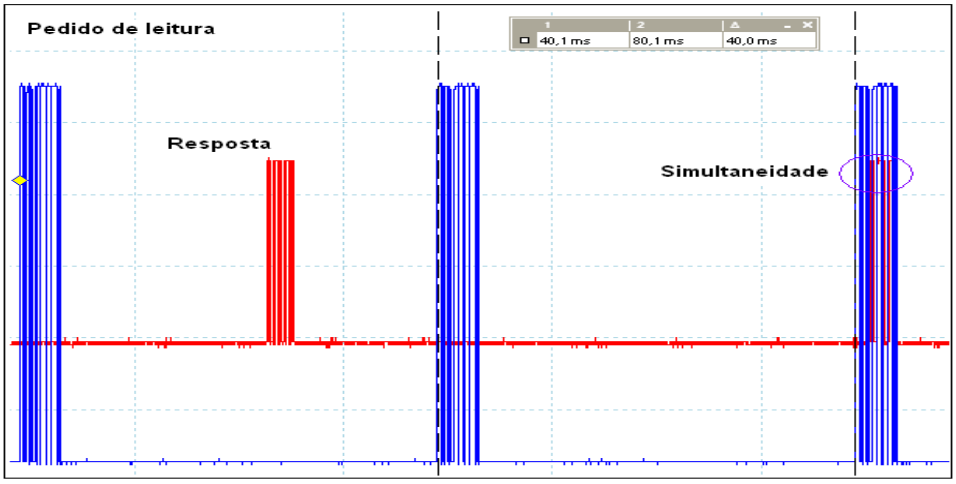

Figura 13 - Simultaneidade

3.3.2 – O TAG UHF

O *tag* escolhido foi o UHF EPC-C1 G2 da marca CONFIDEX®[10]. Este *tag*, classificado como *survivor* (característica de *tag* cujo sinal não sofre interferência por estruturas metálicas), possui 240 bits para identificação e 512 destinados ao usuário, sendo possível armazenar dados nesses *tags*. As informações dos piquetes foram alocadas nesses *tags* e o software embarcado processa essas informações e manipula os dados programados e realizados para cada piquete.

1.1 SPECIFICATION DATA

Device type	Class 1 Generation 2 passive UHF RFID transponder
Air interface protocol	EPCGlobal Class1 Gen2 ISO 18000-6C
Operational frequency	885-869 MHz (EU), 902-928MHz (US), 952-955 MHz (JPN)
IC options	NXP UCODE G2XM
EPC memory	up to 240 bit (G2XM)
EPC memory content	Unique number encoded as a default
Extended memory	512 bit (G2XM)
Read range	up to 8-12 m (26-39 ft) with reader power 2W ERP (dependent on application)
Applicable surface materials	Any surfaces, incl. metal, plastic and wood
Encapsulation material	PC/ABS
Color	Dark grey
Weight	25 g
Delivery format	Single
Amount in box	250 pcs (default)
Product is RoHS compliant	

Figura 14 - Especificação do tag

Fonte: www.confidex.fi/images/stories/pdf/product_datasheets/Survivor_Datasheet.pdf

A figura 15 ilustra o gráfico da variação da distância de resposta do *tag* em relação à frequência da antena do leitor:

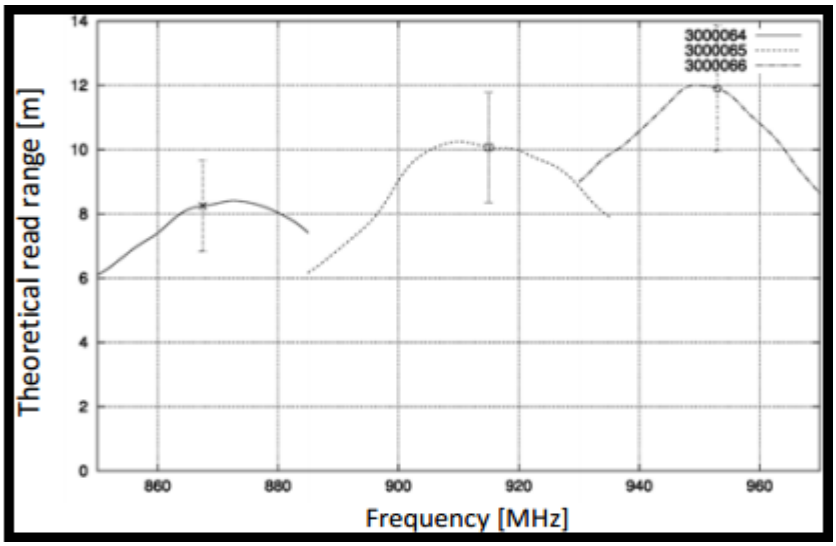

Figura 15 - Faixa de operação do tag

Fonte: www.confidex.fi/images/stories/pdf/product_datasheets/Survivor_Datasheet.pdf

3.3.3 – POSICIONAMENTO DA ANTENA E DO *TAG*

Tão importante quanto a escolha da tecnologia RFID é o posicionamento do leitor em relação ao sistema de descarga e o posicionamento do *tag* em relação ao cocho. O leitor deve ser posicionado próximo ao sistema de descarga na parte superior ao fluxo de ração e o *tag* deve ser posicionado a uma altura mínima de 50 cm em relação ao cocho. Isso ocorre já que a vazão do alimento, rico em água, é alta e pode causar interferência na leitura. A figura 16 e a figura 17 ilustram o posicionamento do leitor UHF e do *tag* respectivamente:

Figura 16 - Posicionamento da antena

Figura 17 - Posicionamento do tag

3.3.4 – ENVIO E RECEPÇÃO DE DADOS

Para utilizar o protocolo descrito no capítulo 3.3.1, aplica-se um filtro sintático capaz de alocar os pacotes de dados decodificados em estruturas inteligíveis.

O fluxograma da figura 18 representa a estrutura lógica criada para enviar um comando ao leitor UHF:

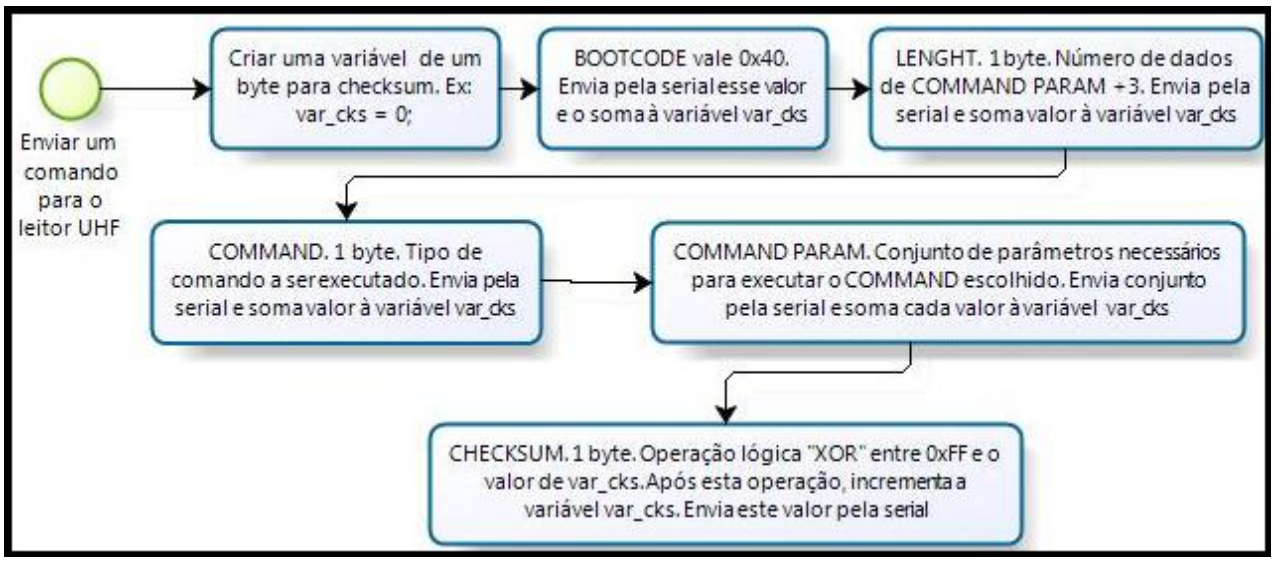

Figura 18- Fluxograma de encapsulamento de dados enviados pela serial

O fluxograma da figura 19 representa a estrutura lógica criada para receber um comando do leitor UHF:

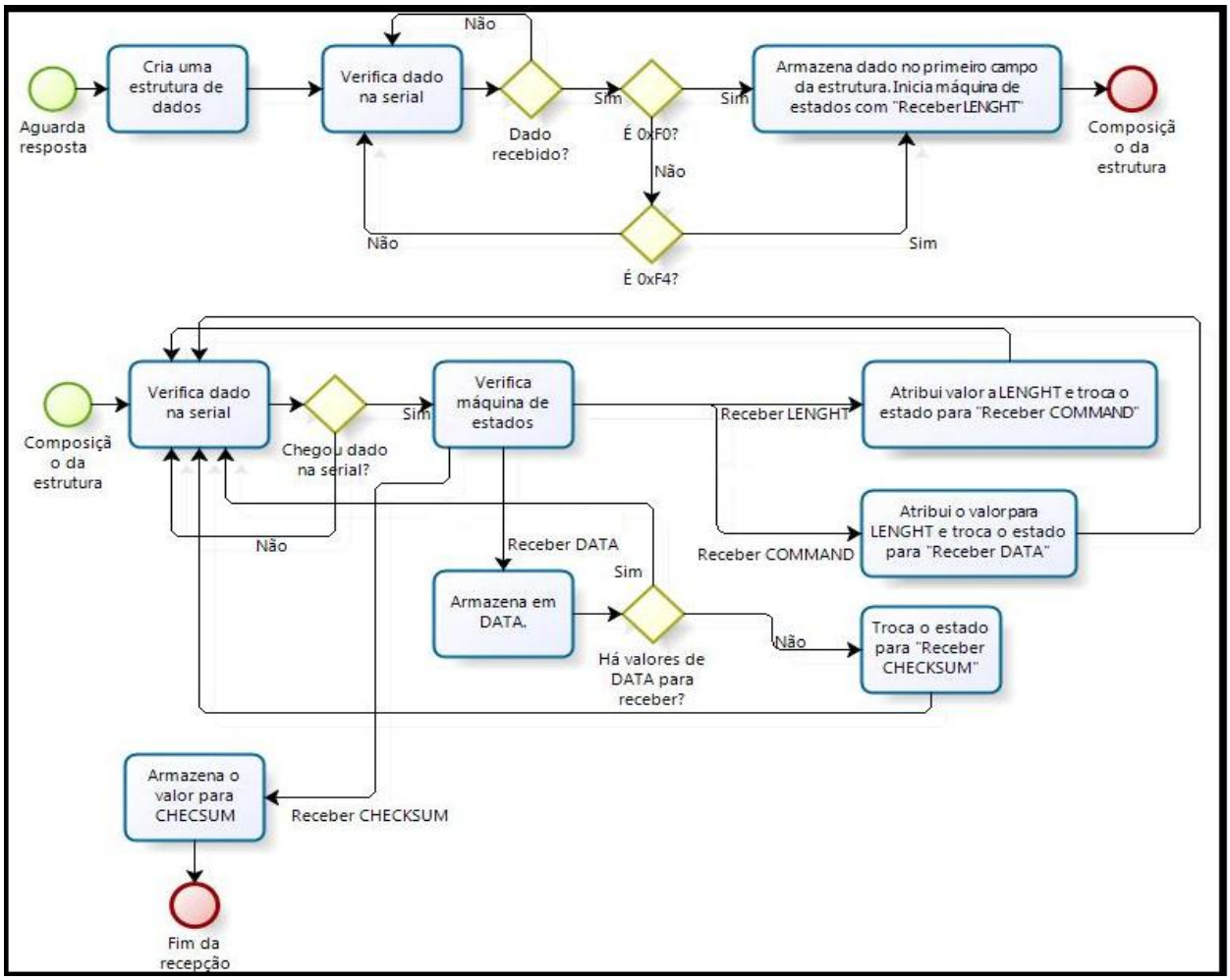

Figura 19 - Fluxograma de dados encapsulados recebidos pela serial

3.4 – INDICADOR DE PESO

O sistema proposto é dependente de um indicador de peso para capturar informações e gerar relatórios relevando as diferenças de peso do caminhão em cada instante. No caminhão onde fora montada a automação, havia o indicador de peso 3101C conectado a 4 células de carga de 3 mV/V com capacidade até 4000 kg cada uma. Seu protocolo de comunicação foi rastreado utilizando-se um osciloscópio e o software Realterm®.

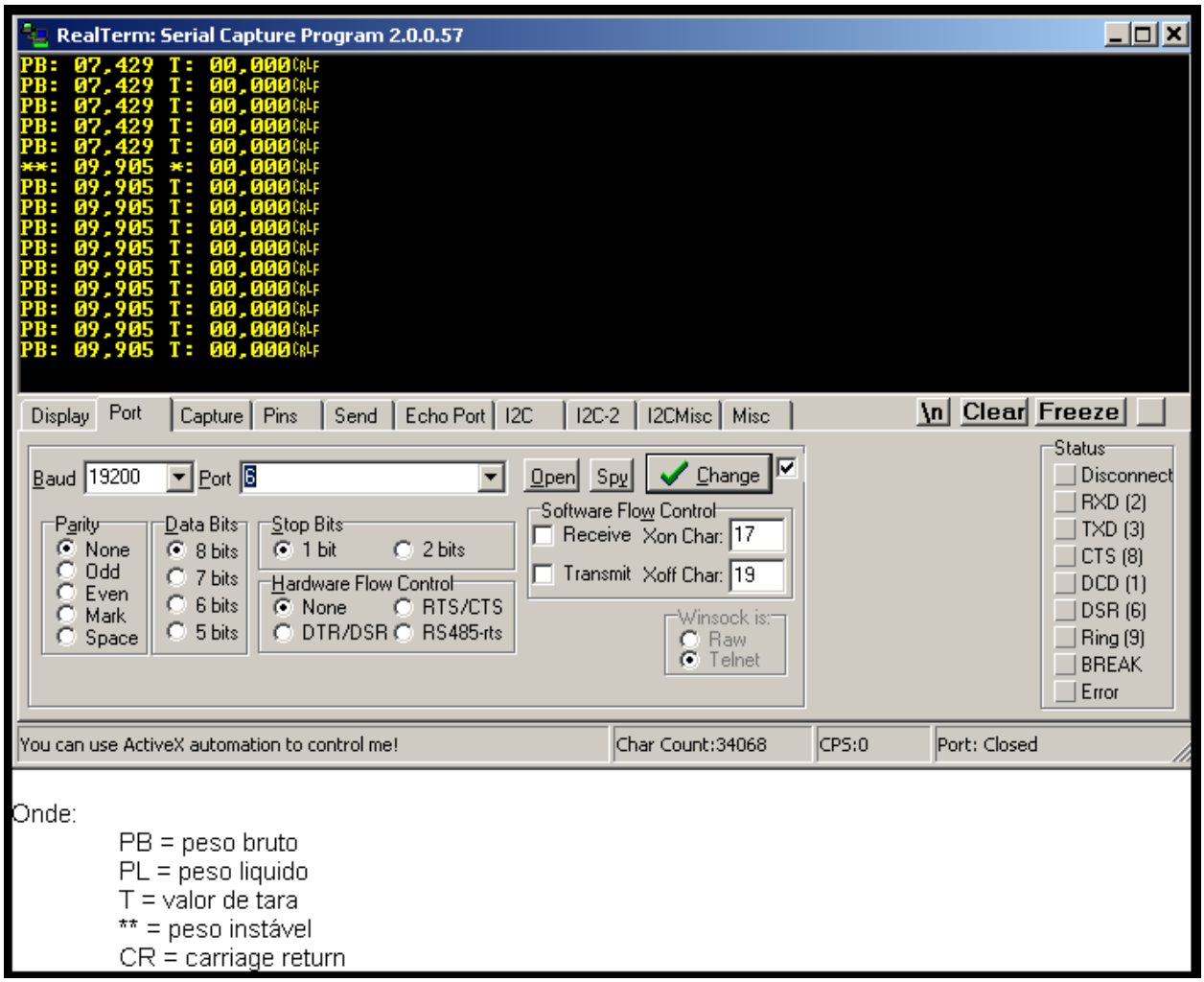

Figura 20 - Captura da comunicação – Indicador 3101C

A partir deste levantamento, foi programada no software embarcado uma rotina capaz de interpretar sintaticamente essa comunicação serial e transformar os caracteres que indicam o peso bruto numa variável do sistema.

3.5 – MÓDULO *BLUETOOTH*

O módulo utilizado para o projeto é o BlueMod+B2X®[8]. Este módulo é um dispositivo de classe 2 com alcance de 20m.

A configuração do *bluetooth* é feita por comandos AT. Um comando AT é uma sequência de bytes enviada pela serial do microcontrolador iniciada pelos bytes “AT” em ASCII ou, em hexadecimal, 0x41 0x54.

Dentre os comandos dispostos no *datasheet**, os seguintes foram utilizados:

- ATS323=19200<n>: Configuração de *baud rate* para 19200 bps;
- AT+BINQ<n>: Varredura dos dispositivos dentro da faixa de alcance;
- ATD<*Mac Address*><n>: Conectar-se a um dispositivo de endereço conhecido;
- ATH<n>: Desconectar-se.

3.6 – *SOFTWARE* COMPUTACIONAL

O sistema automatizado proposto não prevê a construção de um software de gerenciamento dos dados coletados. Para isso, há softwares no mercado, embasados em estudos nutricionais, capazes de fazer gerenciamento de tratos e fabricação de ração. A proposta do presente projeto é construir um *software* computacional de *background* capaz de se comunicar com o sistema embarcado e fornecer relatórios em documentos de texto onde um *software* de gerência compartilhe dessas informações.

O *software* computacional *background*, implementado em Visual C#®, recebe os dados coletados pelo equipamento através de um *bluetooth*, semelhante ao do equipamento embarcado, e salva relatórios numa pasta determinada.

Figura 21 - Software computacional de background

O *software* do PC é *slave*, ou seja, quem inicia a comunicação é sempre o equipamento embarcado, e obedece ao protocolo seguinte:

Comprimento da mensagem	Comando	Dado	CrLf

Figura 22 – Pacote de dados do protocolo interno

Toda comunicação exige uma resposta para ser validada. A resposta segue o mesmo padrão de pacote acima.

3.6.1 – COMPRIMENTO DA MENSAGEM

O comprimento da mensagem, 1 byte, indica quantos bytes devem compor a *string* incluindo ele mesmo.

3.6.2 – COMANDO

O comando, 1 byte, indica qual tipo de pacote de dados será trocado.

- 0xAA: cabeça de comunicação;
- 0x48 ou ‘H’: pedido de atualização de data e hora;
- 0x51 ou ‘Q’: quantidade de currais tratados;
- 0x44 ou ‘D’: relatório de descarga;
- 0x50 ou ‘P’: novo valor programado de peso.

3.6.3 – DADO

Este campo depende do comando e do *status* (envio ou recepção). Tem comprimento variado de acordo com o comprimento do pacote.

3.6.4 – CrLf

Campo, de dois bytes, finalizador de pacote redundante: 0x0D ou *carriage return*, 0x0A ou *line feed*.

3.6.5 – TRANSMISSÃO

A partir do protocolo descrito foi criado um diagrama de blocos ilustrativo de uma transmissão de dados entre o software embarcado e o software computacional. O diagrama de blocos está ilustrado na figura 23.

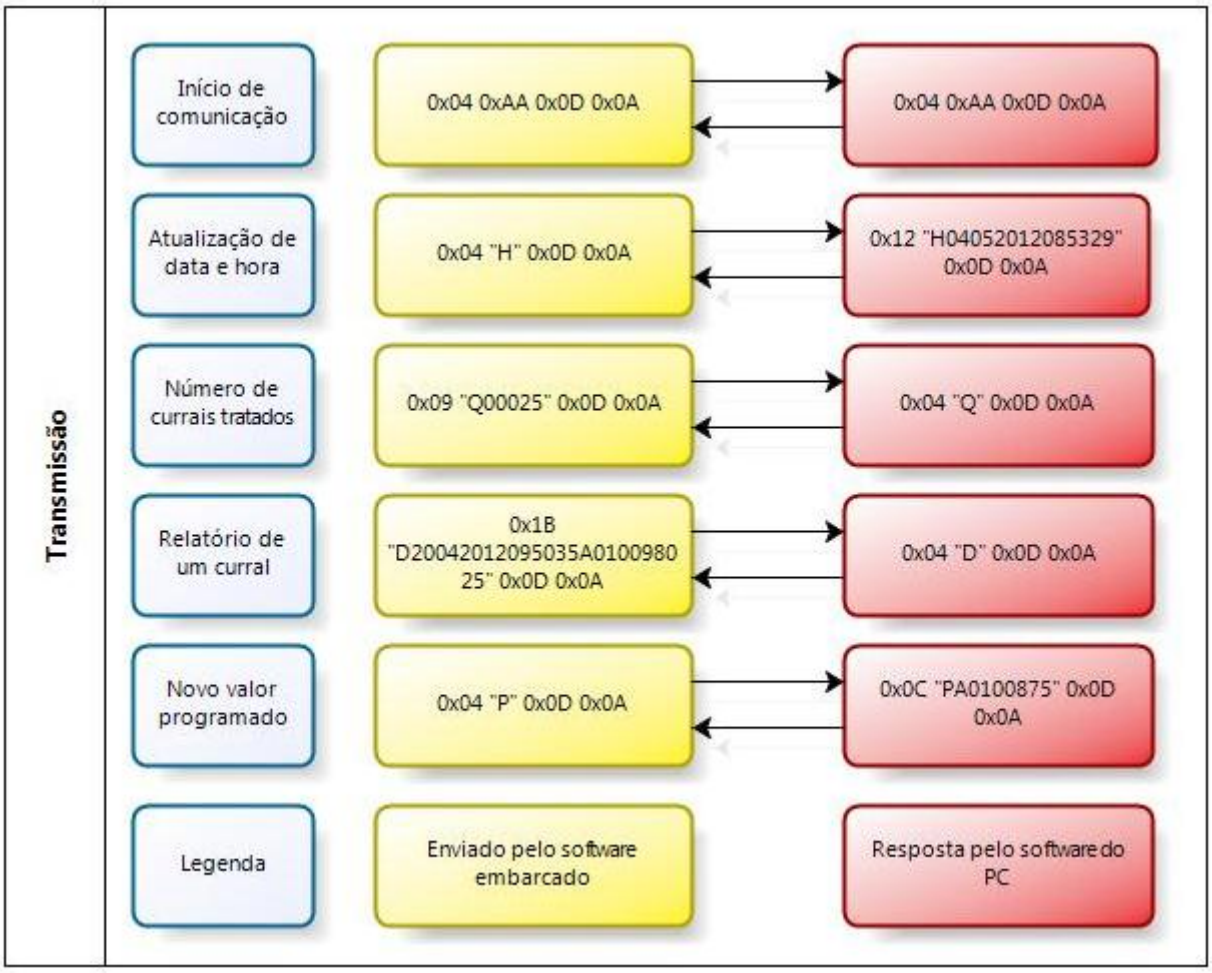

Figura 23 - Diagrama de blocos representando uma transmissão de dados

O software do PC deve processar esses dados e gerar um relatório como ilustrado na figura 24:

Data (DD/MM/AAAA)	Hora (HH:MM:SS)	Curral	Peso (kg)	Temperatura (°C)
20/04/2012	09:50:35	A01	00980	25

Figura 24 - Representação do arquivo gerado a partir da transmissão de dados

Os dados programados, enviados ao equipamento embarcado, como observado na comunicação, devem seguir o seguinte padrão:

Curral	Peso programado (kg)
A01	00875

Figura 25 - Representação do arquivo utilizado para peso programado

4 – RESULTADOS

O sistema proposto foi utilizado entre os dias 15/10/2011 e 09/02/2012. Durante este período os relatórios foram gerados de forma automática, como proposto, e obteve-se um banco de dados para comparar o método implementado ao método convencional.

Os 54 piquetes do confinamento, com 7506 cabeças, são divididos em 4 linhas de 14, 13, 14 e 13 piquetes, respectivamente. Duas linhas, 27 piquetes, foram automatizadas e duas mantiveram o método antigo de trato.

A figura 26 ilustra os resultados obtidos nos testes relativos ao consumo e ao ganho de peso.

	Linha automatizada (kg/cabeça)	Linha não automatizada (kg/cabeça)
Ração consumida (dia)	12,5	13,4
Ganho de peso (100 dias)	159,5	152

Figura 26 - Relação de consumo e ganho de peso entre as linhas

A partir dos dados apresentados, a economia de ração distribuída foi de 3378 kg por dia. No entanto, o ganho de peso médio das linhas automatizadas foi 7,5 kg maior em relação ao ganho de peso médio das linhas não automatizadas. Este fato comprova que, apesar de se despejar mais ração para os animais no modo não automatizado, essa distribuição não foi homogênea e alguns animais obtiveram uma ingestão maior de alimentos do que demandavam para chegar ao ponto ótimo de engorda e esse aumento na inclusão calórica não foi refletida na engorda.

Outro resultado importante obtido foi a diminuição no tempo para executar o serviço. A figura 27 ilustra esse resultado.

	Linha automatizada (s/piquete)	Linha não automatizada (s/piquete)
Tempo demandado	35	57

Figura 27 - Relação de tempo utilizado para tratar cada piquete.

De acordo com esses dados, foram economizados 39 minutos por dia de hora-máquina do distribuidor de ração. Esse tempo reflete na economia de combustível utilizado, cerca de 50 l de diesel por dia, e na redução do tempo em que o distribuidor trabalha em marcha lenta, diminuindo o aquecimento do motor e aumentando sua vida útil.

5 – CONCLUSÃO E TRABALHOS FUTUROS

Os testes realizados em campo com o sistema automatizado proposto neste trabalho norteou à economia de insumos utilizados na alimentação do gado, ao aumento do ganho de peso obtido, à redução do tempo utilizado para o descarregamento de ração e ao aumento da vida útil dos maquinários. Desta forma, o sistema mostrou-se eficiente e trouxe vantagens para esta área.

Atualmente, a rotina de trato é relatada manualmente, fazendo-se paradas ao final de cada curral. Com o monitor embarcado, sem paradas necessárias à anotação, corroborada pela quantidade pré-definida a cada animal, o operador é auxiliado pelo *feedback* do seu trabalho através de um *display* e de uma barra de progresso, feita de *leds*, simples e baratos.

A transmissão de dados, confiável e rápida, é o fator determinístico para o sucesso da aplicação. O equipamento transfere os dados de trato em tempo real para o computador e possibilita o recálculo do próximo trato ao mesmo tempo em que o caminhão de trato é reabastecido de ração.

O *software* do computador não possui interface com o usuário. É apenas um monitor de dados que trabalha em plano de fundo, *background*, e aloca os relatórios em arquivos predeterminados.

A alimentação do gado é realizada a partir de dois processos: o trato, descrito no projeto, e a composição da ração. Notou-se a necessidade de relacionar, então, a composição da ração à descarga ou trato. Desta maneira, o relatório gerado não apresentaria o montante de ração distribuído, e sim a designação de cada componente que foi jogado em cada curral. A partir deste novo sistema de trabalho, seria possível relacionar a engorda aos insumos utilizados de maneira mais assertiva.

A proposta deste trabalho permite desenvolver um equipamento que trabalhe nos dois modos distintos: composição e descarga.

Para um novo modelo, o equipamento de descarga deve se comunicar com o equipamento de composição para trocar informações sobre a dieta fabricada. Para isso, a proposta é utilizar o *zigbee*, sendo este capaz de estabelecer uma rede *mesh*, e, desta maneira, não usar apenas comunicação ponto a ponto.

Além da rede de comunicação, nota-se a necessidade de o equipamento se comunicar diretamente com uma célula de carga e interpretar os dados de peso, excluindo-se a necessidade de um indicador de peso intermediário.

A partir destas mudanças, o processamento pode ser alterado para o microcontrolador *cold fire* V1 MCF51EM256 de 32 bits. Para este micro, também da *Freescale*®, pode-se trabalhar com o sistema operacional de tempo real ou *real time operating system*, RTOS, disponibilizado pela fabricante: MQX. Este microcontrolador possui *realtime* interno e conversor A/D de 16 bits, ideal para pesagens até 16000kg (com precisão de 1kg).

Por fim, podem ser feitas alterações no *software* do PC possibilitando alterações na composição e no trato através de uma interface simples e objetiva.

REFERÊNCIAS BIBLIOGRÁFICAS

[1]. Luiz Roberto Lopes de S. Thiago, Fernando Paim Costa. 1994. Confinamento na prática: Sistemas alternativos. [*Online*].www.cnpgc.embrapa.br/publicacoes/cot/COT50.html. Acessado em 19/04/2012.

[2]. Marcelo Saade Amato. 2010. [*Online*]. http://portalbiossistemas.wordpress.com. Acessado em 19/04/2012.

[3]. CARLOS YASSUNORI KOBAYASHI. [2004]. A Tecnologia Bluetooth e aplicações.

[4]. Fábio Pereira. 2005. Microcontroladores HCS08. Teoria e prática.

[5]. GUSTAVO WEBER DENARDIN. Tutorial para Programação de Microcontroladores HCS08 Baseado no MC9S08AW60. http://pessoal.utfpr.edu.br/gustavo/tutorial_MC9S08AW60.pdf.

[6].FREESCALE.2004.[*Online*].www.freescale.com/webapp/sps/site/homepage.jsp?code=8BITMCU. Acessado em 20/04/2012

[7]. NATIONAL SEMICONDUCTOR.[*Online*] www.national.com/ds/LM/LM35.pdf. Acessado em 13/04/2012.

[8].BLUEMODEB2X.2005. [*Online*].www.stollmann.de/uploads/media/BlueMod_B2x_AT_Command_Reference_r11.pdf.Acessado em 12/11/2011.

[9]. VANCH. www.vanch.cn/en/products_view.asp?id=519. Acessado em 20/03/2012.

[10]. CONFIDEX. www.confidex.fi/products-and-services/uhf-rfid-hard-tags/confidex-survivor. Acessado em 14/02/2012.

APÊNDICE

CÓDIGO-FONTE: ARQUIVOS DE FUNÇÕES

O código abaixo representa as funções do equipamento no que tange às rotinas funcionais. Neste aspecto, o código está numa camada acima da dos registradores. Não será apresentado o baixo nível por questões de sigilo industrial.

```
ProcessorExpert.c
/** ###################################################################
**    Filename    : ProcessorExpert.c
**    Project     : ProcessorExpert
**    Version     : Driver 01.00
**    Compiler    : CodeWarrior C Compiler
**    Date/Time   : 2012-02-17, 15:04, # CodeGen: 0
**    Abstract    :
**      Main module.
**      This module contains user's application code.
**    Settings    :
**    Contents    :
**      No public methods
** ###################################################################*/
/* MODULE ProcessorExpert */
/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "SM1.h"
#include "RealTime.h"
#include "Serial1_UHF.h"
#include "Serial3_Bal_Displ_Cel.h"
#include "Serial_bluetooth.h"
#include "Timer_Cyclic.h"
#include "BEEP_PWM.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "dpl.h"
#include "grl.h"
#include "fls.h"
#include "def.h"
#include "bluetooth.h"
#include "cfg.h"
#include "sch.h"
#include "fun.h"
#include "flag.h"
#include "prt.h"
#include "UHF.h"
#include "string.h"
/* User includes (#include below this line is not maintained by Processor Expert) */
void main(void)
{
  /* Write your local variable definition here */
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
                PE_low_level_init();
  /*** End of Processor Expert internal initialization.              ***/
  /* Write your code here */
  /* For example: for(;;) { } */
                cfg_Ports();
                Dpl_Init();
                BEEP_PWM_InitTO();
                BEEP_PWM_ClrValue();
                for(;;){
                                switch (menu_mode) {
                                case MENU2_DO_DISCHARGE:
                                                fun_Do_Discharge1();
                                                if (menu_mode != MENU3_CONNECT){
                                                                uhf_Seek_Tags();
                                                }
                                                break;//case MENU2_DO_DISCHARGE
                                case MENU3_DO_DISCHARGE:
                                                fun_Do_Discharge2();
                                                if (menu_mode != MENU3_CONNECT)
                                                                uhf_Seek_Tags();
                                                break;//case MENU3_DO_DISCHARGE
                                case MENU3_CONNECT:
                                                fun_Do_Connect();
```

```
                                        break;//case MENU3_CONNECT
                                case MENU4_TRANSMIT:
                                        fun_Do_Trasmit();
                                        break;//case MENU4_TRANSMIT
                                case MENU5_DISCONNECT:
                                        fun_Do_Disconnect();
                                        break;//case MENU2_DISCONNECT
                                }
                                grl_Check_Beep();//Verifica se há beep para executar por PWM
                                Flag.DoubleBeep = FALSE;
                                Flag.KeyBeep = FALSE;
                                while (cyclic_timer < 40);//Tempo de cada ciclo
                                LEDS = LED_00;
                                cyclic_timer = 0;//Zera o contador de ciclo...
                                Dpl_Refresh();//Atualiza o Display...
                                bluetooth_PullParserPendente();//Verifica frames do bluetooth...
                }

 /*** Don't write any code pass this line, or it will be deleted during code generation. ***/
 /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
 for(;;){}
 /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END ProcessorExpert */
/*
** ###################################################################
**
**    This file was created by Processor Expert 5.3 [05.01]
**    for the Freescale MC9HCS08 series of microcontrollers.
**
** ###################################################################
*/

Fun.c
#include "fun.h"
Flag_T Flag;
unsigned char read_tag[10];
void fun_Do_Discharge1(void) {
                uint8_T i;
                switch (fun_Check_Tag()) {
                case TAG_FALSE:
                                sprintf_(Display.String2, current_weight, (sizeof(Display.String2)) - 1);//Atualiza o display com o valor do peso atual...

                case TAG_INIT:
                case TAG_NORMAL:
                                count = 0;
                                grl_Beep_Key();
                                init_weight_truck = current_weight;
                                fun_Check_Corral(); //Verifica a ID do curral, o peso programado e a nota...
                                for(i=0; i<4; i++){
                                        bkp_corral[i] = tag[i];
                                }
                                fun_Init_Discharge2();
                                menu_mode = MENU3_DO_DISCHARGE;
                                break;//case TAG_INIT ; case TAG_NORMAL
                case TAG_FINAL:
                                break;//case TAG_FINAL
                case TAG_CONNECT:
                                if (total_discharge) {
                                        menu_mode = MENU3_CONNECT;
                                        fun_Init_Connect();
                                } else {
                                        menu_mode = MENU2_DO_DISCHARGE;
                                        fun_Init_Discharge1();
                                }
                                break;//case TAG_CONNECT
                case TAG_DISCONNECT:
                                menu_mode = MENU5_DISCONNECT;
                                fun_Init_Disconnect();
                                break;//case TAG_CONNECT
                default:
                                break;//default
                }
}
void fun_Do_Discharge2(void) {
                uint16_T aux_weight;
                switch (fun_Check_Tag()) {
                case TAG_FALSE:
                                //Função que compara os pesos e seta um valor para weight_display(peso que será exibido no display)...
                                if (init_weight_truck > current_weight) {
                                        aux_weight = init_weight_truck - current_weight;
                                        if (aux_weight <= 10)//Atualiza o display somente quando foram deixados mais de 10 Kg...
                                                aux_weight = 0;
                                        if (aux_weight <= set_weight) {
                                                weight_display = set_weight - aux_weight;
                                                Display.Char[0] = ' ';
                                        } else {
                                                weight_display = aux_weight - set_weight;
                                                Display.Char[0] = '-';
                                        }
                                } else {
                                        weight_display = set_weight;
```

```
                }
                FUN_UPDATE_DISPLAY();
                BREAK;//CASE TAG_FALSE
        CASE TAG_INIT:
        CASE TAG_NORMAL://FINALIZOU O CURRAL ANTERIOR....
                GRL_BEEP_KEY();
                FUN_UP_MEMORY_TREATMENT();
                IF (FLAG.UP_MEMORY_TREATMENT == TRUE)
                        INIT_WEIGHT_TRUCK = CURRENT_WEIGHT;
                FUN_CHECK_CORRAL();
                BREAK;//CASE TAG_INIT; CASE TAG_NORMAL
        CASE TAG_FINAL:
                GRL_BEEP_END_FUNCTION();
                FUN_UP_MEMORY_TREATMENT();
                MENU_MODE = MENU2_DO_DISCHARGE;
                FUN_INIT_DISCHARGE1();
                TAG[0] = 0;
                BREAK;//CASE TAG_FINAL

        CASE TAG_CONNECT:
                FUN_UP_MEMORY_TREATMENT();
                FUN_INIT_CONNECT();
                MENU_MODE = MENU3_CONNECT;
                BREAK;//CASE TAG_CONNECT

        CASE TAG_DISCONNECT:
                FUN_UP_MEMORY_TREATMENT();
                FUN_INIT_CONNECT();
                MENU_MODE = MENU3_CONNECT;
                BREAK;//CASE TAG_DISCONNECT
        }
}
UINT8_T FUN_CHECK_TAG(VOID) {
        UINT8_T CURRENT_TAG[4];
        UINT8_T ID_TAG, CA, CKS;
        UINT8_T I;
        IF(FLAG.TAG != TRUE){
                RETURN TAG_FALSE;
        }ELSE{
                FLAG.TAG = FALSE;
        }
        //RESPOSTA AO LER TAG CORRETAMENTE:
        // F0 - TAG LIDO CORRETAMENTE...
        // 08 - NÚMERO DE BYTES DA LEITURA, EXCLUINDO O F0...
        // EE - CMD...
        // 01 - 01 TAG LIDO...
        // 02 - O PRIMEIRO TAG LIDO POSSUI 02 WORDS, OU SEJA, 4 BYTES...
        // 01 - DADO [0] - TIPO DE CURRAL...
        // 41 - DADO [1] - 'A'...
        // 30 - DADO [2] - '0'...
        // 31 - DADO [3] - '1'...
        // 72 - CKS...
        ID_TAG = READ_TAG[0];
        FOR(I = 0; I < 4; I++){
                CURRENT_TAG[I] = READ_TAG[I];
        }
        IF (CURRENT_TAG[2] == TAG[2]) {
                IF (CURRENT_TAG[1] == TAG[1]) {
                        IF (CURRENT_TAG[0] == TAG[0]) {
                                IF (ID_TAG == TAG_CONNECT){
                                        RETURN TAG_CONNECT;
                                }
                                RETURN TAG_FALSE;
                        }
                }
        }
        IF (ID_TAG == TAG_FINAL) {
                RETURN TAG_FINAL;
        } ELSE {
                FOR(I = 0; I < 4; I++){
                        TAG[I] = CURRENT_TAG[I];
                }
                IF(ID_TAG <= 0X08){
                        IF (ID_TAG != TAG_FALSE && ID_TAG != TAG_CONNECT) {
                                FOR(I = 0; I <= 2; I++){
                                        DISPLAY.STRING2[I] = TAG[I];
                                }
                        }
                }
                IF (ID_TAG == TAG_INIT) {//TAG INICIAL
                        RETURN TAG_INIT;
                } ELSE IF (ID_TAG == TAG_NORMAL) {//TAG NORMAL
                        RETURN TAG_NORMAL;
                } ELSE IF (ID_TAG == TAG_CONNECT) {
                        RETURN TAG_CONNECT;
                } ELSE IF (ID_TAG == TAG_DISCONNECT) {
                        RETURN TAG_FALSE;
                } ELSE
                        RETURN TAG_FALSE;
        }
}
VOID FUN_CHECK_CORRAL(VOID){
        UNSIGNED CHAR LAST_BYTE[1], BUFFER[9];
```

```
                unsigned long int i, offset_leitura;
                Display.String2[0] = ' ';
                Display.String2[1] = ' ';
                Display.String2[2] = ' ';
                Display.String2[3] = tag[0];
                Display.String2[4] = tag[1];
                Display.String2[5] = tag[2];
                spi_Cfg(SPI_FLASH);
                for (i = 0; i < MAX_CORRAL; i++) {
    offset_leitura = (unsigned long int) (2 + (unsigned long int) (i) * 7);
    fls_Read(MEM_TRATO_PROG + offset_leitura, sizeof (last_byte), &last_byte);
    if (last_byte[0] == tag[2]) {
                offset_leitura--;
                fls_Read(MEM_TRATO_PROG + offset_leitura, sizeof (last_byte), &last_byte);
                if (last_byte[0] == tag[1]) {
                offset_leitura--;
                fls_Read(MEM_TRATO_PROG + offset_leitura, sizeof (last_byte), &last_byte);
                if (last_byte[0] == tag[0]) {//Identificou o curral...
                fls_Read(MEM_TRATO_PROG + offset_leitura, 7, &buffer);
                                                        set_weight = (buffer[3] - 0x30)*1000;//Transformação de um dado em ASCII para inteiro...
                 set_weight += (buffer[4] - 0x30)*100;
                 set_weight += (buffer[5] - 0x30)*10;
                 set_weight += (buffer[6] - 0x30);
                break;
                } else
                                set_weight = 1000;
                                } else
        set_weight = 1000;
                } else
      set_weight = 1000;
                }
                sprintf_(Display.String1, set_weight, (sizeof (Display.String1)) - 1);
                spi_Cfg(SPI_DISPLAY);
}
void fun_Update_Display(void) {
    sprintf_(Display.String1, weight_display, (sizeof (Display.String1)) - 1);
    sprintf_(Display.String2, current_weight, (sizeof (Display.String2)) - 1);
}
void fun_Update_Progress_bar(void) {
  unsigned long int prog_bar;
  unsigned int aux_weight;
  if (init_weight_truck > current_weight) {
    aux_weight = init_weight_truck - current_weight;
    if (aux_weight <= 10)
      aux_weight = 0;
  } else
    aux_weight = 0;
  if (set_weight != 0) {
    if (aux_weight <= set_weight) {
      prog_bar = (unsigned long int) ((unsigned long int) (aux_weight)*100);
      prog_bar = (unsigned long int) (prog_bar) / (unsigned long int) (set_weight);
    } else
      prog_bar = 100;
  } else
    prog_bar = 0;

  if (prog_bar <= 4)
    LEDS = LED_00;
  else if (prog_bar <= 16)
    LEDS = LED_01;
  else if (prog_bar <= 28)
    LEDS = LED_02;
  else if (prog_bar <= 40)
    LEDS = LED_03;
  else if (prog_bar <= 52)
    LEDS = LED_04;
  else if (prog_bar <= 64)
    LEDS = LED_05;
  else if (prog_bar <= 76)
    LEDS = LED_06;
  else if (prog_bar <= 92)
    LEDS = LED_07;
  Else if (prog_bar > 92)
    LEDS = LED_08;
}
void fun_Do_Trasmit(void) {
                static uint8_t sender = 0;
                //Se o número de tentativas de comunicação com o PC for maior que 15, sai do modo de transmissão...
                if(transmition_tries >= 15){
                                sender = transmition_tries = 0;
                                prt_bluetooth_master = SEND_HEAD;
                                menu_mode = MENU5_DISCONNECT;
                                fun_Init_Disconnect();
                                return;
                }
                if(sender >= 10){
                                sender = 0;
                                switch (prt_bluetooth_master) {
                                case SEND_HEAD:
                                                bluetooth_enviaTransmissionRequest(bluetooth_monta_head);
                                                transmition_tries++;
                                                break;//case SEND_HEAD
                                case GET_TIMER:
```

```
                                        BLUETOOTH_ENVIATRANSMISSIONREQUEST(BLUETOOTH_MONTA_TIMER);
                                        TRANSMITION_TRIES++;
                                        BREAK;//CASE GET_TIMER
                                CASE SEND_TOTAL_DISCHARGE:
                                        IF (TOTAL_DISCHARGE) {
                                                BLUETOOTH_ENVIATRANSMISSIONREQUEST(BLUETOOTH_MONTA_TOTAL_DISCHARGE);
                                                TRANSMITION_TRIES++;
                                        }ELSE{
                                                PRT_BLUETOOTH_MASTER = SEND_END;
                                        }
                                        BREAK;//CASE SEND_TOTAL_DISCHARGE
                                CASE SEND_MEM_DISCHARGE:
                                        BLUETOOTH_ENVIATRANSMISSIONREQUEST(BLUETOOTH_MONTA_MEM_DISCHARGE);
                                        TRANSMITION_TRIES++;
                                        BREAK;//CASE SEND_MEM_DISCHARGE
                                CASE GET_TREAT:
                                        BLUETOOTH_ENVIATRANSMISSIONREQUEST(BLUETOOTH_MONTA_TRATO_PROGRAMADO);
                                        TRANSMITION_TRIES++;
                                        BREAK;//CASE GET_TREAT
                                DEFAULT:
                                        PRT_BLUETOOTH_MASTER = SEND_HEAD;
                                        BREAK;//DEFAULT
                                }
                        }ELSE{//SE (SENDER < 10)
                                SENDER++;
                        }
}
VOID FUN_DO_CONNECT(VOID) {
                        IF (TOTAL_DISCHARGE) {
                        GRL_BEEP_END_FUNCTION();
                        SCI_CHOOSE1(SCI1_BLUETOOTH);
                                SPRINTF(CONECTA_BLUETOOTH, "ATD%S\0", ENDERECO);
                        SCI_TX1_CST((CONECTA_BLUETOOTH);
                        FUN_CLEAR_BUFFER();
                        MENU_MODE = MENU4_TRANSMIT;
                        } ELSE {
                        MENU_MODE = MENU3_DO_DISCHARGE;
                        FUN_INIT_DISCHARGE1();
                                SCI_CHOOSE1(SCI1_UHF);
                        }
}
VOID FUN_DO_DISCONNECT(VOID) {
                        SCI_CHOOSE1(SCI1_BLUETOOTH);
                        DTR_PIO4_SPK = ON;
                        GRL_DELAY_MS(100);//LARGURA DO PULSO...
                        DTR_PIO4_SPK = OFF; // PULSAR EM 1 PARA SAIR DO MODO TRANSPARENTE NO MODULO SPK BLUETOOTH
                        SCI_TX1_CST("ATH\0");
                        MENU_MODE = MENU3_DO_DISCHARGE;
                        FUN_INIT_DISCHARGE1();
                        SCI_CHOOSE1(SCI1_UHF);
}
```