UNIVERSIDADE LUTERANA DO BRASIL

PRÓ-REITORIA DE GRADUAÇÃO

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

LEANDRO KAUER ZUCHETTO

AUTOMATIZAÇÃO DE UMA BANCADA DE TESTE PARA PARTIDA A FRIO EM MÁQUINAS AGRÍCOLAS

Canoas, Dezembro de 2008

LEANDRO KAUER ZUCHETTO

AUTOMATIZAÇÃO DE UMA BANCADA DE TESTES PARA PARTIDA A FRIO EM MÁQUINAS AGRÍCOLAS

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da ULBRA como um dos requisitos obrigatórios para a obtenção do grau de Engenheiro Eletricista

Departamento:

Engenharia Elétrica

Professor Orientador:

MSc. Eng. Eletr. André Luis Bianchi – CREA: 089197

Canoas

2008

FOLHA DE APROVAÇÃO

Nome do Autor: Leandro Kauer Zuchetto

Matrícula: 011102211-8

Título: Automatização de uma bancada de testes para partida a frio em máquinas agrícolas

Trabalho de Conclusão de Curso apresentado ao Departamento de Engenharia Elétrica da ULBRA como um dos requisitos obrigatórios para a obtenção do grau de Engenheiro Eletricista

Professor Orientador:

MSc. Eng. Eletr. André Luis Bianchi

CREA: 089197

Banca Avaliadora:

Dr. Eng. Eletr. Valner João Brusamarello

CREA-RS: 078158-D

Conceito Atribuído (A-B-C-D):

MSc. Eng. Eletr. Dalton Luiz Rech Vidor

CREA-RS: 079005-D

Conceito Atribuído (A-B-C-D):

Assinaturas:

Autor Leandro Kauer Zuchetto	Orientador André Luis Bianchi
Avaliador Valner João Brusamarello	Avaliador Dalton Luiz Rech Vidor

Relatório Aprovado em:

DEDICATÓRIA

Dedico aos meus pais Ademir e Vanda, aos meus irmãos Rodrigo e Letícia, e especialmente a minha esposa Gisele e meus filhos Maurício e Roger. Agradeço a todos pelo apoio e compreensão.

AGRADECIMENTOS

A todos que colaboraram direta ou indiretamente na elaboração deste trabalho, o meu reconhecimento.

Ao Professor Eng. Eletr. André Luis Bianchi pelo estímulo, dedicação e esforço pessoal proporcionado.

Aos inúmeros colegas pelas sugestões e observações valiosas.

Ao colega de empresa Eng. Mecatrônico Vinicius Nazário Sena pelas contribuições de grande valia.

RESUMO

ZUCHETTO, Leandro Kauer. **Título do Trabalho: Automatização de uma bancada de testes para partida a frio em máquinas agrícolas**. Trabalho de Conclusão de Curso em Engenharia Elétrica - Departamento de Engenharia Elétrica. Universidade Luterana do Brasil. Canoas, RS. 2008.

O ensaio de partida a frio executado na AGCO do Brasil tem como objetivo analisar o comportamento do sistema de partida de máquinas agrícolas, simulando condições de baixa temperatura. Através deste ensaio é possível determinar se o sistema elétrico de partida tem capacidade de prover a rotação mínima para que o motor entre em funcionamento. A proposta deste trabalho é simplificar a realização do ensaio de partida a frio, agilizando o tempo de preparação da máquina e garantindo maior segurança ao equipamento e pessoas envolvidas. Desenvolvendo um sistema microprocessado para a execução e controle do ensaio e um link de radiofreqüência para a comunicação entre a máquina e a cabine de controle.

Palavras chave: microcontrolador, radiofreqüência, aquisição de dados.

ABSTRACT

ZUCHETTO, Leandro Kauer. Title of the Work: Automation of the test to cold start in agricultural machines. Work of Conclusion of Course in Electrical Engineering - Electrical Engineering Department. Lutheran University of Brazil. Canoas, RS. 2008.

The test of cold start executed in AGCO of Brazil, has as objective to analyze the behavior of the cold start system in agricultural machines, simulating conditions of low temperature. Through this test is possible to determine if the electric system start have the capacity to provide adequate engine rotation to allow engine starting. The proposal of this work is to simplify the accomplishment of the test of cold start, activating the setup time of machine and guaranteeing larger safety to the equipment and involved people. Developing a microprocessor system for the execution and control of the test and a radio frequency link for the communication between the machine and the cab control.

Keywords: microcontroller, radio frequency, acquisition of data.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A/D: *Analog / Digital*

CRC: *Cyclic Redudance Checksum*

DCE: Data Communication equipment

DTE: Data Terminal equipment

FSK: Frequency Shift Keying

GFSK: Gaussian Frequency Shift Keying

IES: Interferência Entre Símbolos

ISM: Industrial Scientific and Medical

PC: *Personal Computer*

RF: Radiofreqüência

STOU: *Super Tractor Oil Universal*

SUMÁRIO

1. INTRODUÇÃO

Com a grande expansão das exportações de máquinas agrícolas, torna-se a cada dia mais necessário as máquinas estarem preparadas para todo tipo de região e clima. Para conseguir simular esta variedade de regiões e climas, executam-se diversos ensaios em laboratório. Trazendo dados e fatos importantes para que os projetistas possam avaliar e desenvolver novas tecnologias.

O trabalho proposto foi desenvolvido na Engenharia do Produto da AGCO do Brasil e consiste na proposição de melhorias no ensaio de partida a frio, que visa de um modo geral, verificar se o sistema elétrico de partida tem capacidade de prover a rotação mínima para que o motor entre em funcionamento.

1.1. Motivação

Na busca por dados mais confiáveis, torna-se necessário além de testes de campo, no qual a máquina é submetida a operar em condições e culturas diversas, executar ensaios em laboratório que as condições podem ser simuladas e controladas.

Os testes em laboratório permitem que diferentes modelos sejam submetidos a condições idênticas de ensaio, podendo estabelecer comparações entre os resultados obtidos em cada modelo. Gerando uma grande quantidade de informações confiáveis para uma posterior análise de engenharia.

Tendo em vista todos os fatores que influenciam nos dados obtidos de um ensaio, este trabalho foi desenvolvido, com foco no ensaio de partida a frio em máquinas agrícolas realizado na empresa AGCO do Brasil.

1.2. Objetivos

O trabalho tem como objetivo geral implementar melhorias no ensaio de partida a frio de máquinas agrícolas, visando maior confiabilidade dos dados obtidos e maior segurança ao equipamento e pessoas envolvidas.

Para cumprir com este objetivo geral, os objetivos específicos são:

- Desenvolver um sistema de controle e monitoração microcontrolado, que será responsável por comandar a partida da máquina e monitorar as variáveis envolvidas;
- Desenvolver a comunicação de dados via RF para interligar a máquina com a cabine de controle.

1.3. Estrutura do Trabalho

O presente trabalho foi organizado da seguinte forma:

O segundo capítulo apresenta uma revisão teórica dos principais temas necessários a elaboração da proposta de trabalho. O terceiro capítulo trás os detalhes técnicos dos componentes a implementar, são detalhados os circuitos utilizados, as placas de circuito impresso desenvolvidas e a interface homem-máquina (IHM). O quarto capítulo trás as considerações finais e conclusões do projeto.

2. REFERENCIAL TEÓRICO

2.1. Padrão para Comunicação digital RS-232

O RS-232, também conhecido por EIA RS-232C, é um padrão para troca série de dados binários entre um DTE (terminal de dados, de *Data Terminal equipment*) e um DCE (comunicador de dados, de *Data Communication equipment*).

No protocolo de comunicação RS-232, caracteres são enviados um a um como um conjunto de bits. A codificação freqüentemente usada é o "*start-stop* assíncrono" que usa um bit de início, seguido por sete ou oito bits de dados, possivelmente um bit de paridade, e um ou dois bits de parada, sendo então, necessários 10 bits para enviar um único caractere. Tal fato acarreta a necessidade em dividir por um fator de dez a taxa de transmissão de bits para obter a velocidade de transmissão de caracteres. A alternativa mais comum ao "*start-stop* assíncrono" é o HDLC. O padrão define os níveis elétricos correspondentes aos níveis lógicos um e zero, a velocidade de transmissão padrão e os tipos de conectores (LUEKE, 2004).

O padrão especifica 20 diferentes sinais de conexão e um conector em forma de D é comumente usado.

No conector a maioria dos pinos não são utilizados nos mais diversos dispositivos, sendo comum para que as máquinas economizem espaço e dinheiro. O conector padrão em forma de D contém apenas 9 pinos (figura 2-1).

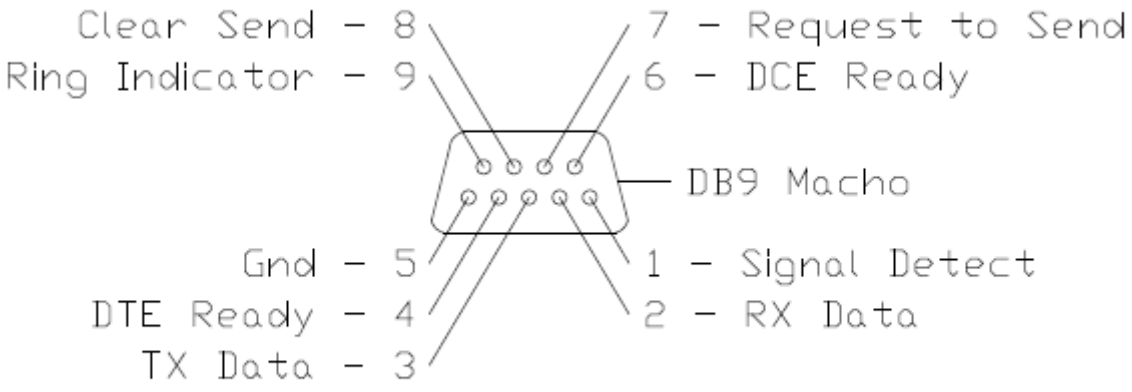

Figura 2-1 - Conector DB9 macho

O RS-232 é recomendado para conexões curtas (quinze metros ou menos). O nível lógico um é definido por ser tensão negativa, a condição de sinal é chamada marca e tem significado funcional de *OFF* (desligado). O nível lógico zero é positivo, a condição de sinal é espaço, e a função é ON (ligado). Como mostra a figura 2-2.

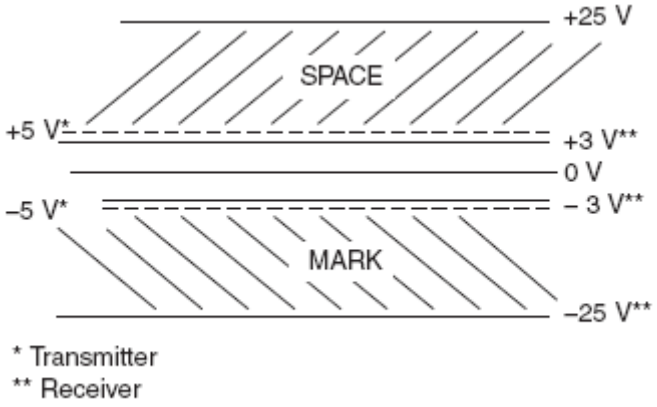

Figura 2-2 - Definição dos níveis lógicos (LUEKE, 2004).

2.2. Canais de Comunicação

Existem três maneiras de se interligar os sistemas digitais.

2.2.1. Canal Simplex

O Canal simplex (figura 2-3) é o canal no qual a direção de transmissão é inalterada. Por exemplo, transmissões de TV e rádio, que somente transmitem o sinal e nunca é permitida a transmissão inversa.

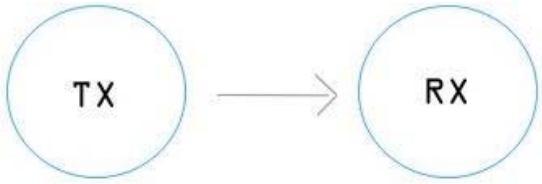

Figura 2-3 - Canal Simplex

2.2.2. Canal Half-Duplex

No Canal *half-duplex* a comunicação é feita apenas em uma direção por vez. Isto significa que ambos podem transmitir e receber dados, porém não simultaneamente. Por exemplo, os walkie-talkies, onde uma parte fala enquanto a outra escuta (figura 2-4).

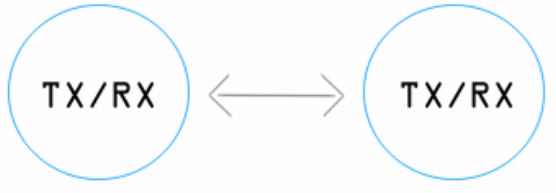

Figura 2-4 - Canal *Half-Duplex*

2.2.3. Canal Full-Duplex

O Canal *full-duplex* permite que mensagens sejam trocadas simultaneamente em ambas as direções. Ele pode ser visto como dois canais simplex, um canal direto e um canal reverso, conectados nos mesmos pontos (figura 2-5).

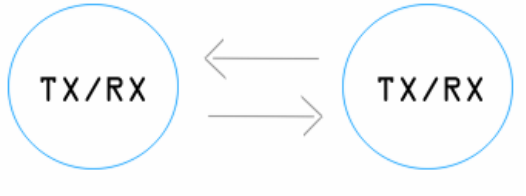

Figura 2-5 - Canal *Full-Duplex*

2.3. Modulação por Deslocamento de Freqüência

2.3.1. Modulação FSK (Frequency Shift Keying)

A modulação FSK atribui freqüências diferentes para a portadora em função do bit que é transmitido. Portanto, quando um bit 0 é transmitido, a portadora assume uma freqüência correspondente a um bit 0 durante o período de duração de um bit. Quando um bit 1 é transmitido, a freqüência da portadora é modificada para um valor correspondente a um bit 1 e analogamente, permanece nesta freqüência durante o período de duração de 1 bit, como mostrado na figura 2-6.

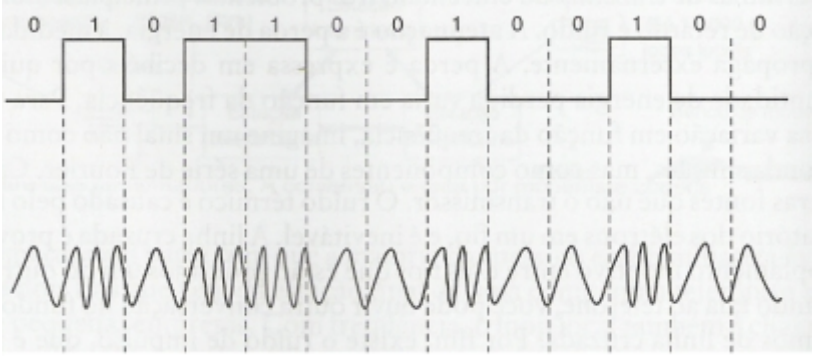

Figura 2-6 - Modulação FSK

2.3.2. Modulação GFSK (Gaussian Frequency Shift Keying)

No GFSK os dados são codificados na forma de variações de freqüência em uma portadora, de maneira similar à modulação FSK.

Os pulsos transmitidos tendem a sofrer o efeito do espalhamento durante a transmissão, o que ocasiona superposição nos pulsos adjacentes, causando uma distorção conhecida como IES. Para compensar esta interferência antes dos pulsos entrarem no modulador, eles passam por um filtro gaussiano. O filtro gaussiano é uma espécie de formatador de pulsos que serve para suavizar a transição entre os valores dos pulsos.

A figura 2-7 ilustra a transformação dos pulsos após passarem pelo filtro gaussiano.

Figura 2-7 - Filtro Gaussiano

2.4. Aquisição de Dados

A aquisição de dados é o processo pelo qual um fenômeno físico real é transformado num sinal elétrico proporcional e convertido num formato digital para posterior visualização, armazenamento, processamento e análise.

Em muitas aplicações, a aquisição de dados não se restringe apenas à aquisição, mas também compreende ações de controle sobre os sistemas. Os sinais digitais de controle correspondentes ao processo provenientes dos computadores são convertidos em sinais apropriados para atuar em diversos equipamentos de controle: atuadores, relés, válvulas, moduladores, etc.

Os sensores e transdutores fornecem a ligação direta entre o mundo real e o sistema de aquisição de dados convertendo sinais de grandezas físicas em sinais elétricos (tensões ou correntes) apropriados para os condicionadores de sinais e/ou os equipamentos de aquisição de dados. Atualmente, existem transdutores disponíveis para a medição da maioria das grandezas físicas existentes.

Os sinais elétricos gerados nos sensores e transdutores muitas vezes necessitam ser convertidos em uma forma apropriada para o equipamento de aquisição.

O condicionamento do sinal é a parte mais importante da aquisição de dados. Para isto, existem dispositivos chamados condicionadores de sinais, que são circuitos eletrônicos que adéquam os sinais analógicos para a conversão digital.

Os principais sub-componentes dos condicionadores são os amplificadores, filtros e isoladores. Através dos amplificadores, o sinal analógico é amplificado para ajustar-se à faixa de entrada do conversor A/D e, quando necessário, o amplificador responsabiliza-se também pela alimentação dos sensores. Os filtros reduzem os ruídos do sinal analógico, ou seja, diminuem eventuais interferências que podem ser originadas por diversas fontes: radiofreqüência, rede elétrica, aterramento, etc. Os isoladores, quando presentes, têm a função de proteger os outros módulos contra eventuais sobrecargas de tensão e corrente, as quais podem causar danos irreversíveis aos circuitos eletrônicos digitais.

O hardware de medição é o responsável pelas entradas e saídas de sinais na cadeia de medida. Assim, ele pode executar qualquer uma das seguintes funções:

- Entrada, processamento e conversão para o formato digital, usando conversores digitais de sinais analógicos provenientes do meio de medição. Os dados, após convertidos, são transferidos para o computador para visualização, armazenamento ou análise;
- Entrada de sinais digitais que contêm informação acerca de um sistema ou processo;
- Processamento, conversão para um formato analógico, utilizando conversores analógicos de sinais digitais do computador para controle de processos;
- Saída de sinais de controle digitais.

2.5. Sensores

2.5.1. Sensor de rotação

Os sensores de rotação podem ser de diferentes tipos e princípios, a seguir alguns modelos que podem ser implementados para medir rotação.

- **Sensores magnetoresistivos:** Fornecem uma alteração de resistência em resposta a uma alteração da intensidade de campo

magnético. A figura 2-8 demonstra a utilização destes sensores para a medida de rotação.

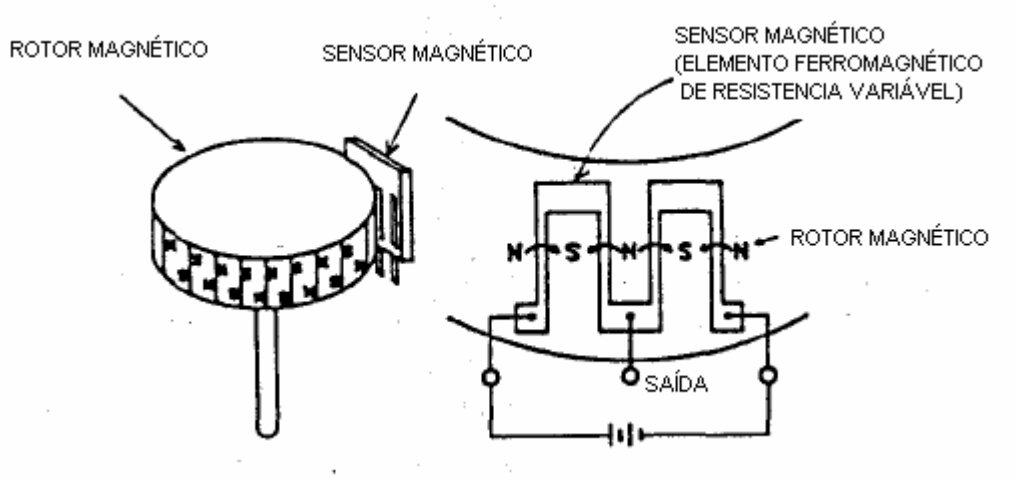

Figura 2-8 - Sensor de rotação magnético

O rotor magnético consiste de uma série de pequenos imãs com pólos norte e sul organizados conforme figura 2-8.

Quando o sensor magnético é exposto a um campo magnético, uma diferença de potencial aparecerá entre suas extremidades. A variação do campo magnético produzida pelo rotor origina na saída do sensor uma série de pulsos com certo período, o qual pode ser traduzido em rotação.

- **Sensores ópticos:** Fornecem uma variação de resistência em resposta a uma alteração da intensidade luminosa. A figura 2-9 mostra a implementação de um sensor de rotação através de sensores ópticos.

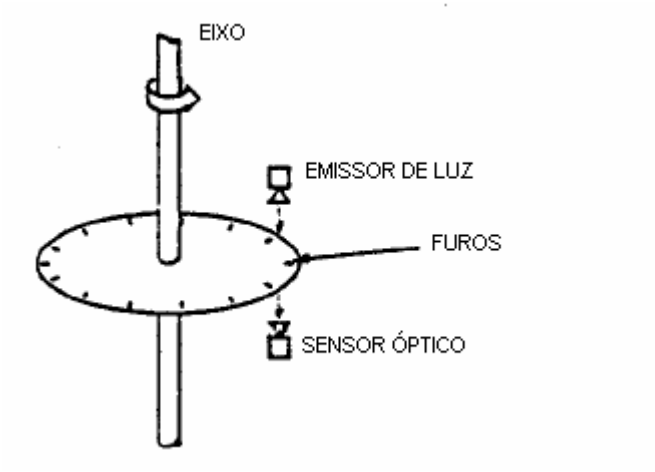

Figura 2-9 - Sensor de rotação óptico

Os furos igualmente espaçados permitem que a luz passe até o sensor óptico, gerando uma série de pulsos que permitem uma medida de rotação.

Quando se deseja saber o sentido de rotação ou a posição angular do eixo, pode utilizar um *encoder*. Existem dois tipos de *encoders*: o encoder incremental e o encoder absoluto.

Encoder incremental: Como pode ser visto na figura 2-10, a configuração das ranhuras do fotosensor é tal que, sempre um deles detecta a luz primeiro do que o outro. Conseqüentemente dois sinais digitais são produzidos, que ficam em quadratura, o que permite que a direção de rotação seja determinada.

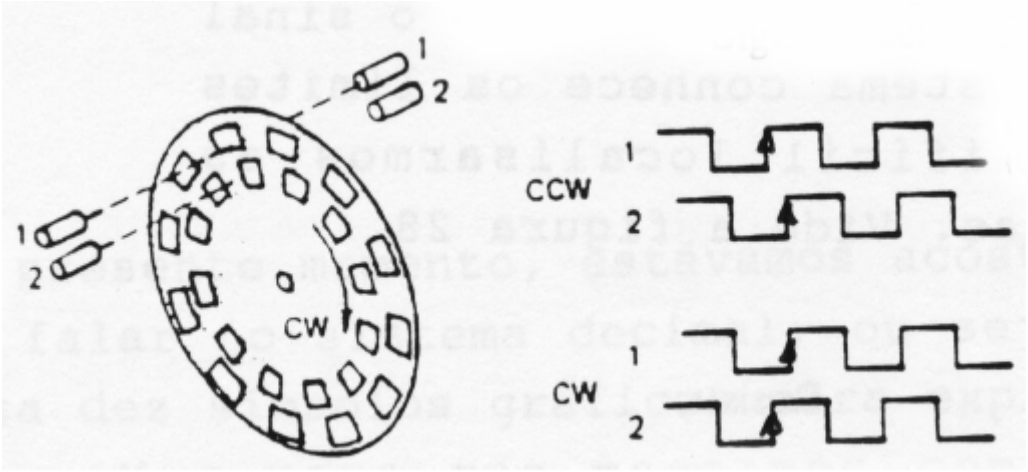

Figura 2-10 - *Encoder* incremental

Encoder absoluto: O *encoder* absoluto é constituído de um disco de vidro estampado por um padrão de trilhas concêntricas

(figura 2-11). Existem feixes de luz que atravessam cada trilha iluminando foto sensores individuais, o qual fornece sempre a posição absoluta de um objeto e não existe contato físico para que ocorra a detecção. Feita essa detecção, defini-se a posição absoluta do *encoder* através de um código. O código mais empregado é o binário, pois este é facilmente manipulado por um circuito relativamente simples e, com isso, não se faz necessário nenhum tipo de conversão para se obter a posição real do *encoder*. O código é extraído diretamente do disco (que está em rotação). O sincronismo e a aquisição da posição no momento da variação entre dois códigos tornam-se muito difíceis.

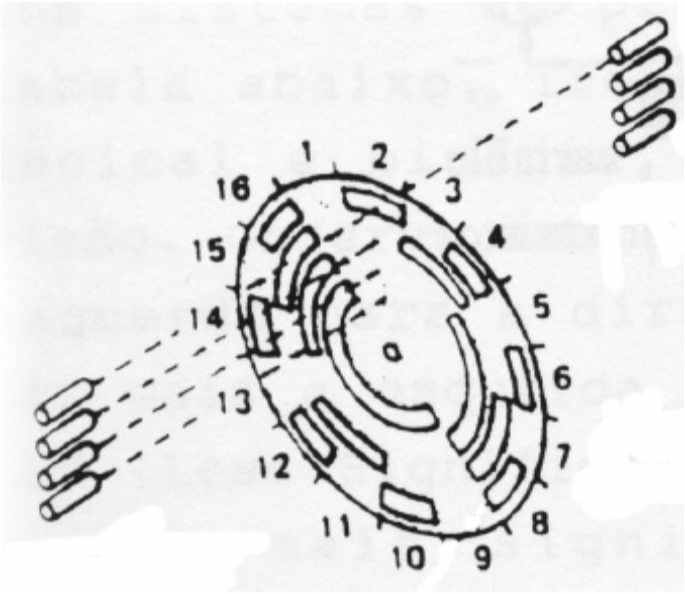

Figura 2-11 – *Encoder* absoluto

Como o mau alinhamento das fotocélulas pode causar erros de leitura, uma solução comum é o uso de um disco estampado em Código Gray, em lugar do código binário padrão. O Código Gray é um código digital com a propriedade de que duas palavras códigos consecutivas diferem apenas de 1 bit, conforme pode ser visualizado na figura 2-12.

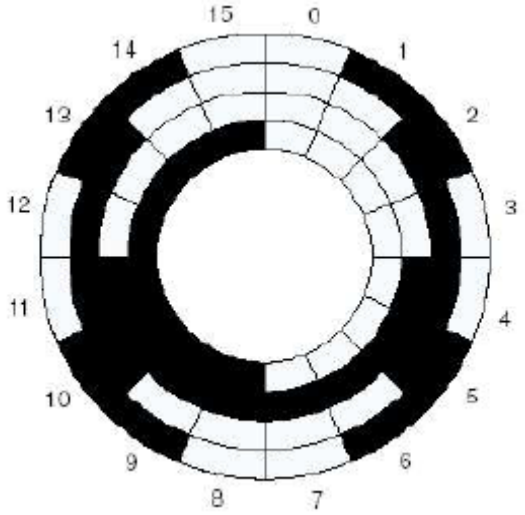

Figura 2-12-2 - *Encoder* absoluto com base no código Gray

- **Sensores indutivos:** Geram uma tensão em resposta a transição de um metal próximo ao imã, alterando o seu fluxo magnético.

A figura 2-13 mostra a montagem de um sensor indutivo para a medida de rotação.

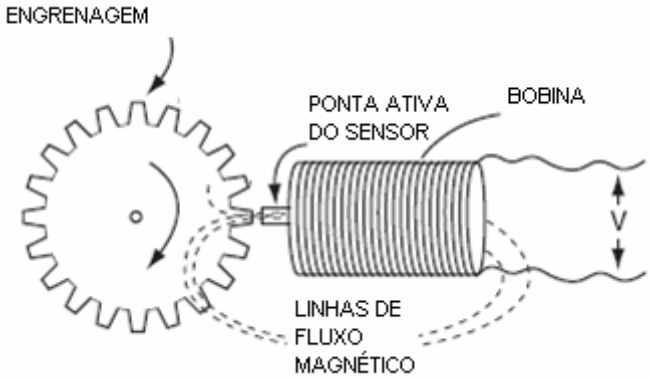

Figura 2-13 - Sensor indutivo de rotação (LUECKE)

As linhas de fluxo magnético passam pela ponta ativa do sensor, pelo dente da engrenagem e retorna a bobina. Quando um dente da engrenagem fica alinhado com a ponta ativa do sensor, a concentração de fluxo magnético é maior. Com o movimento da engrenagem o espaço entre os dentes alinha com a ponta ativa do sensor e a concentração de fluxo magnético fica menor. Em resposta a rotação da engrenagem uma série de pulsos é gerado, conforme mostra a figura 2-14.

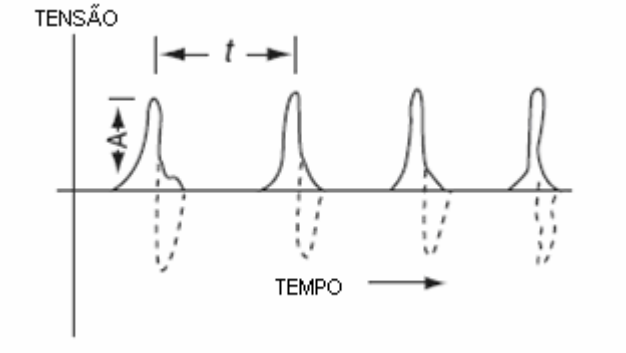

Figura 2-14 – Resposta do sensor (LUECKE)

O tempo (t) entre os pulsos varia conforme a velocidade que gira a engrenagem. Contando o número de pulsos em cima de um tempo determinado, pode obter a velocidade da engrenagem.

2.5.2. *Sensor de corrente contínua*

Para medidas de altas correntes em DC, utiliza-se geralmente derivadores shunt, que são resistores de alta exatidão. Coloca-se o shunt em série com a carga, de forma que toda a corrente consumida pela carga passe por ele. A tensão lida entre seus terminais é proporcional a corrente que flui pelo circuito. A figura 2-15 exemplifica a utilização deste sensor.

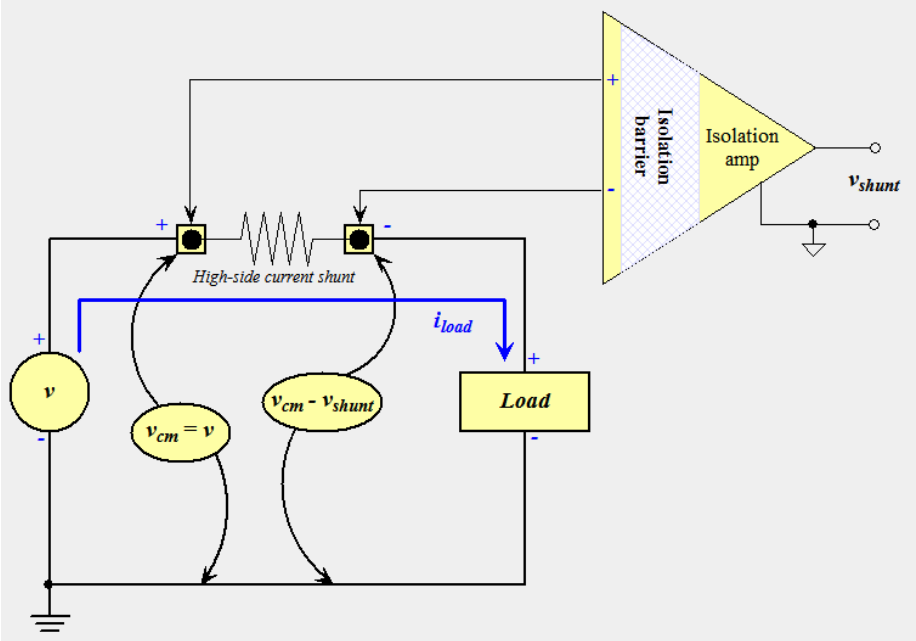

Figura 2-105 - Digrama de ligação do shunt (WIKIPEDIA, 2008)

2.6. Atuadores lineares elétricos

Os atuadores elétricos são itens muito utilizados em equipamentos industriais, hospitalares, indústria automobilística e agrícola, onde são necessários movimentos lineares, confiáveis e que exijam força.

Um atuador linear elétrico é composto basicamente por:

a) Motor elétrico de corrente contínua;

b) Engrenagens;

c) Fuso.

O movimento resultante de um atuador linear, como o próprio nome diz, será um deslocamento de avanço ou recuo, o mesmo movimento realizado por um cilindro hidráulico ou pneumático. Isto acontece porque, quando o motor elétrico entra em funcionamento, o sistema de engrenagens faz o fuso, montado dentro da haste metálica, movimentar esta, avançando ou recuando, dependendo de como é feita a ligação elétrica no motor. A figura 2-16 apresenta uma vista dos componentes de um atuador linear do fabricante (LINAK, 2008). Ele costuma substituir cilindros hidráulicos e pneumáticos.

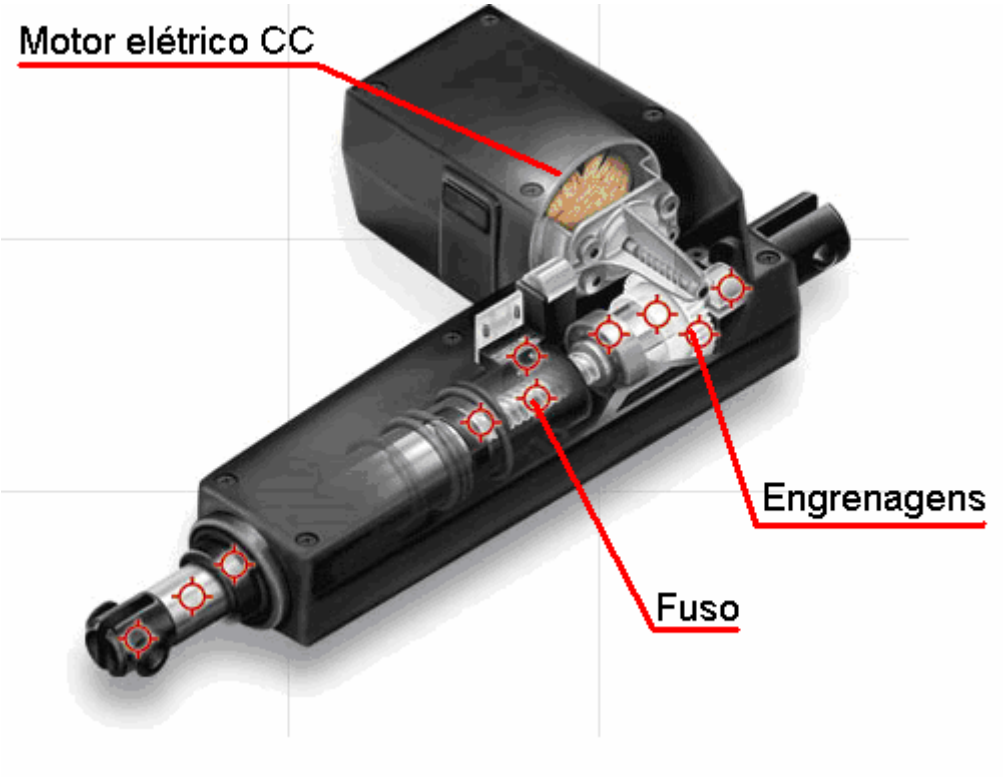

Figura 2-16 - Atuador linear (LINAK, 2008)

A força resultante no movimento do atuador é um parâmetro muito importante. Alguns modelos podem ter até 10000N de força no movimento de avanço. Estes valores eram antes conseguidos apenas com cilindros hidráulicos ou pneumáticos, tendo-se como grande vantagem a possibilidade de controle de um motor elétrico.

Alguns modelos oferecem a possibilidade de montagem de sensores de fim de curso ou de posição da haste do atuador.

2.7. Procedimento do Ensaio

A seguir, será mostrado o procedimento e detalhes de como o ensaio é realizado na empresa AGCO do Brasil. São apresentadas características e definições do procedimento GF10700001, desenvolvido pela AGCO em 2005, com uso exclusivo dos sites da corporação.

2.7.1. Preparação da máquina (GF107000001)

A máquina deve estar preparada conforme manual do operador para operar na temperatura do teste requisitado, geralmente -7 e -20°C, dependendo para qual clima a máquina esta destinada. Abaixo alguns pontos importantes para o sucesso do ensaio:

- A bateria deve estar a 80% de sua carga nominal, para simular uma condição real de carga.
- A mistura do fluido de arrefecimento deve ser na proporção de 50% de água e 50% de anticongelante.
- Os óleos lubrificantes da transmissão e motor deverão ser os recomendados para condições climáticas frias, ou seja, SAE 10W-30 STOU. Se este óleo não estiver disponível, deve-se utilizar um óleo com o mesmo número de viscosidade, por exemplo, SAE 10W-40.
- O combustível deve ser uma mistura de 50% diesel e 50% querosene.
- A máquina deve funcionar por pelo menos 15 minutos antes de iniciar o processo de resfriamento.
- O tempo de exposição mínimo à baixa temperatura é de 18 horas, para garantir que todas as partes da máquina estejam na mesma temperatura.
- Deve ser instalada instrumentação necessária para registrar os valores de corrente do motor de partida, velocidade do motor e tensões na bateria, motor de partida e bomba injetora.

2.7.2. Procedimento (GF107000001)

O procedimento padrão para ensaio de partida a frio determinado pela AGCO é descrito a seguir:

1. Para determinar a rotação que o motor de partida está provendo ao motor, a bomba injetora deve ser desligada. Se a máquina estiver equipada com sistema de aquecimento para partida a frio, este deve ser acionado por 10 segundos e após acionar o motor de partida por 10 segundos, monitorando a rotação do motor.
2. Após determinar a rotação deve-se manter a bomba injetora ligada e executar os itens abaixo:

- Máquinas equipadas com sistema de aquecimento para partida a frio: deve-ser acionado por 10 segundos o sistema de aquecimento;
- O motor de partida deve ser acionado por no máximo 15 segundos. Caso não se obtenha sucesso na primeira vez, deve-se aguardar 10 segundos para uma nova tentativa;
- Se a máquina não partir em quatro tentativas, o teste deve ser finalizado e a causa deve ser investigada, para que um novo teste possa ser realizado.

Devem ser monitorados e registrados durante este procedimento os valores de corrente do motor de partida, velocidade do motor e tensões na bateria, motor de partida e bomba injetora.

3. MATERIAIS E MÉTODOS

Neste projeto utilizou-se um módulo de comando interligado a um computador através da porta serial, um módulo para aquisição dos dados e controle da máquina, e dois módulos TRW-24G para comunicação de RF. A figura 3-1 mostra um diagrama do sistema implementado.

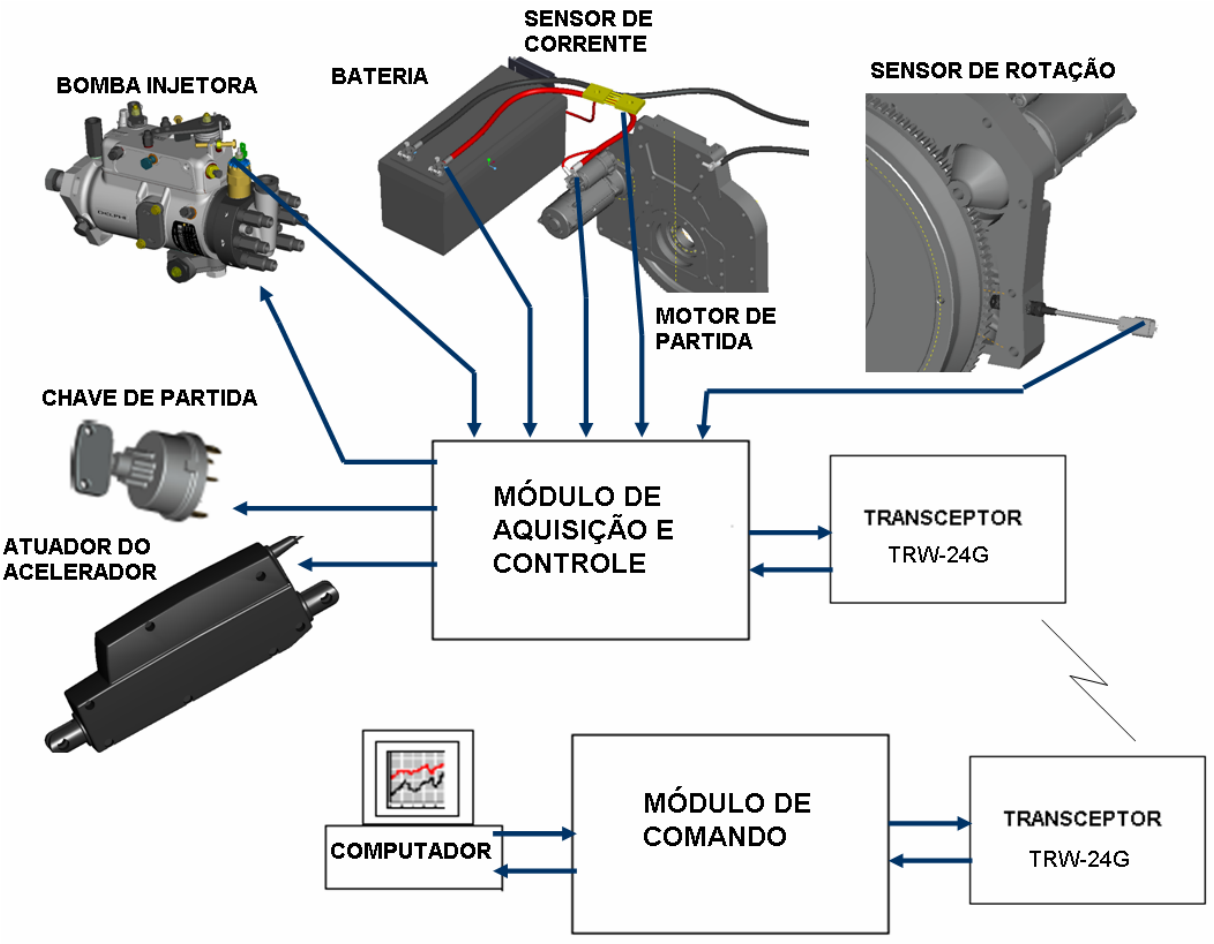

Figura 3-1 - Diagrama do sistema

3.1. Hardware

3.1.1. Módulo de aquisição e controle

Neste módulo é utilizado um microcontrolador MSP430F169 fabricado pela Texas Instruments, que é responsável por comandar a partida da máquina, adquirir os dados e enviá-los ao módulo TRW-24G, fazendo a comunicação com o módulo de comando (Figura 3.2).

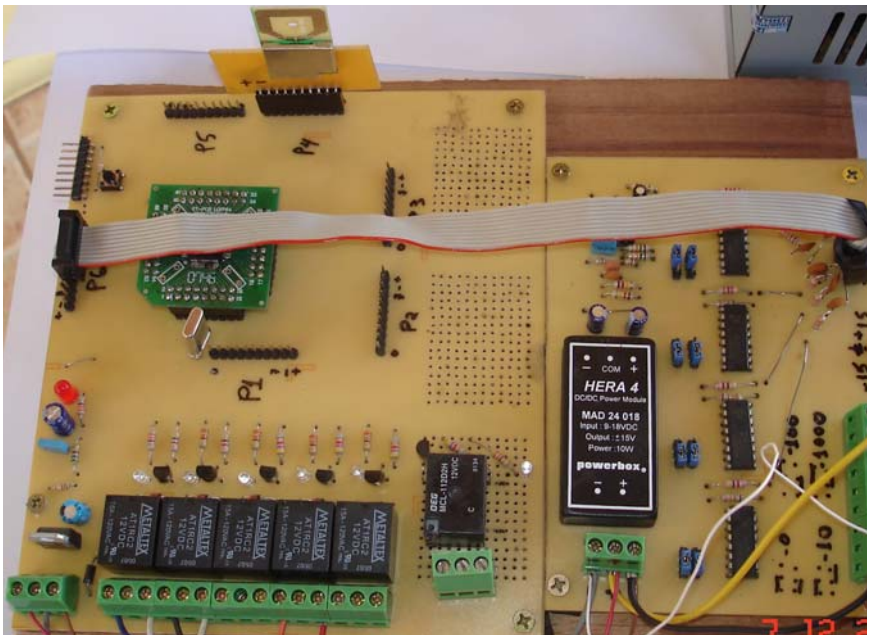

Figura 3-2 - Foto módulo de aquisição e controle

3.1.1.1. Alimentação

O MSP430F169 e o módulo TRW-24G operam em baixa tensão. Isto requer uma tensão de alimentação de no máximo 3,6 V. Utilizou-se então uma fonte baseado no CI regulador de tensão LM317, conforme figura 3-3.

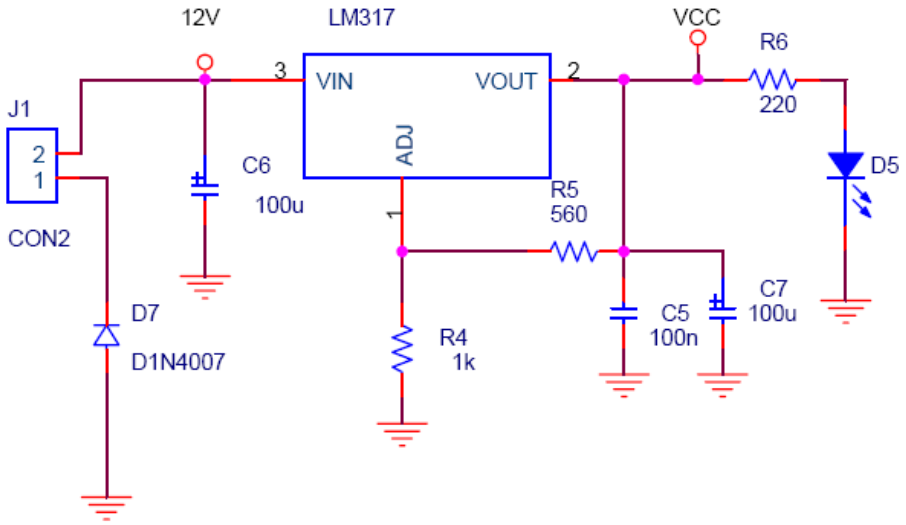

Figura 3-3 - Fonte de alimentação.

3.1.1.2. Controle da máquina

Para o acionamento da partida da máquina estão sendo utilizados seis bits da porta P1 do MSP430F169, onde cada um aciona um relé através do circuito mostrado na figura 3-4.

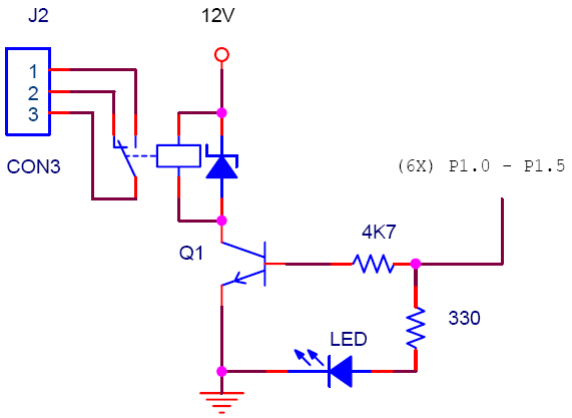

Figura 3-4 - Saída a relé.

Cada relé controla uma função durante a partida da máquina, o relé ligado a porta P1.0 controla a bomba injetora, os ligados as portas P1.1, P1.2 e P1.3 substituem a chave de partida, e os da P1.4 e P1.5 controlam a aceleração da máquina.

3.1.1.3. Aquisição de dados

Para a aquisição dos dados foram utilizados os conversores A/D de 12 bits do MSP430F169, a seguir o detalhamento dos circuitos condicionadores de cada variável a ser medida.

3.1.1.3.1. Rotação do motor

Para medir a rotação do motor, foi utilizado um sensor indutivo montado na cremalheira do motor, conforme figura 3-5.

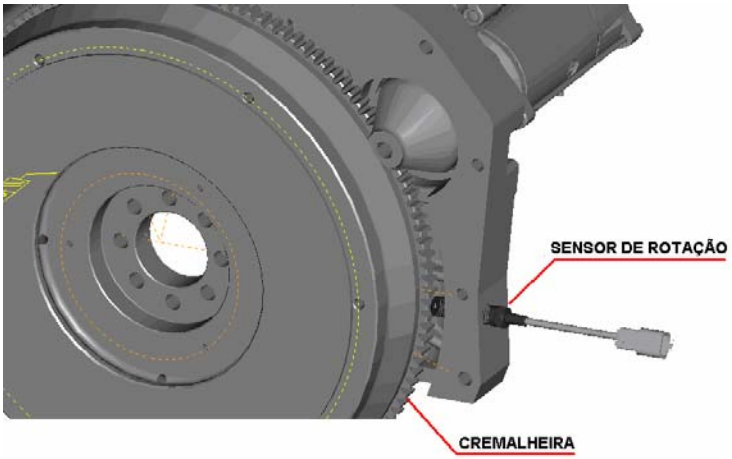

Figura 3-5 - Montagem sensor de rotação.

O sensor utilizado gera uma onda quadrada com amplitude de 12 v, com freqüência variando conforme rotação. Para o condicionamento do sinal foi utilizado um circuito conversor de freqüência para tensão, ajustado para uma saída de 0 a 3,6V, e um filtro RC passa baixa com freqüência de corte em 100Hz. Conforme figura 3-6.

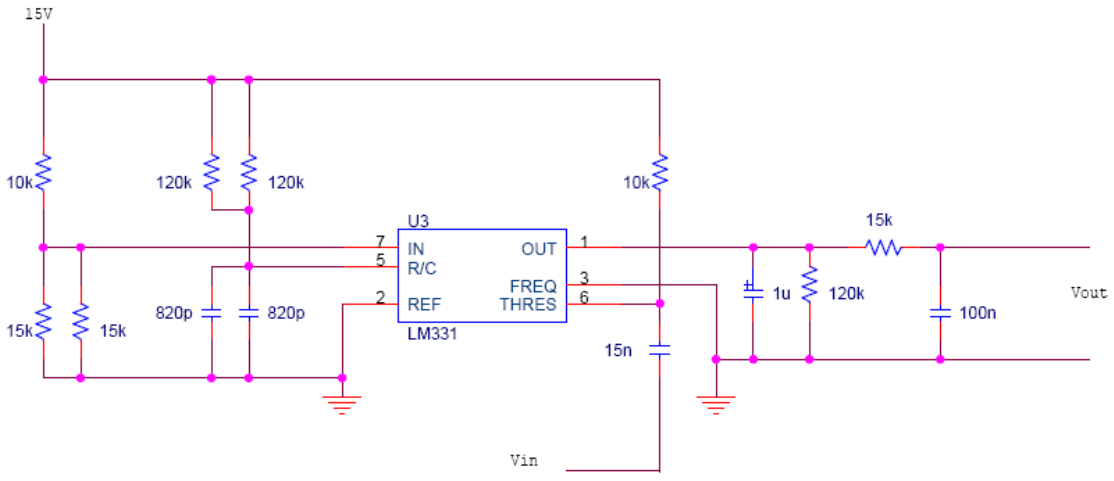

Figura 3-6 - Circuito conversor freqüência x tensão

Após montar e ajustar o circuito acima foi extraído a função de transferência do circuito, conforme mostra figura 3-7.

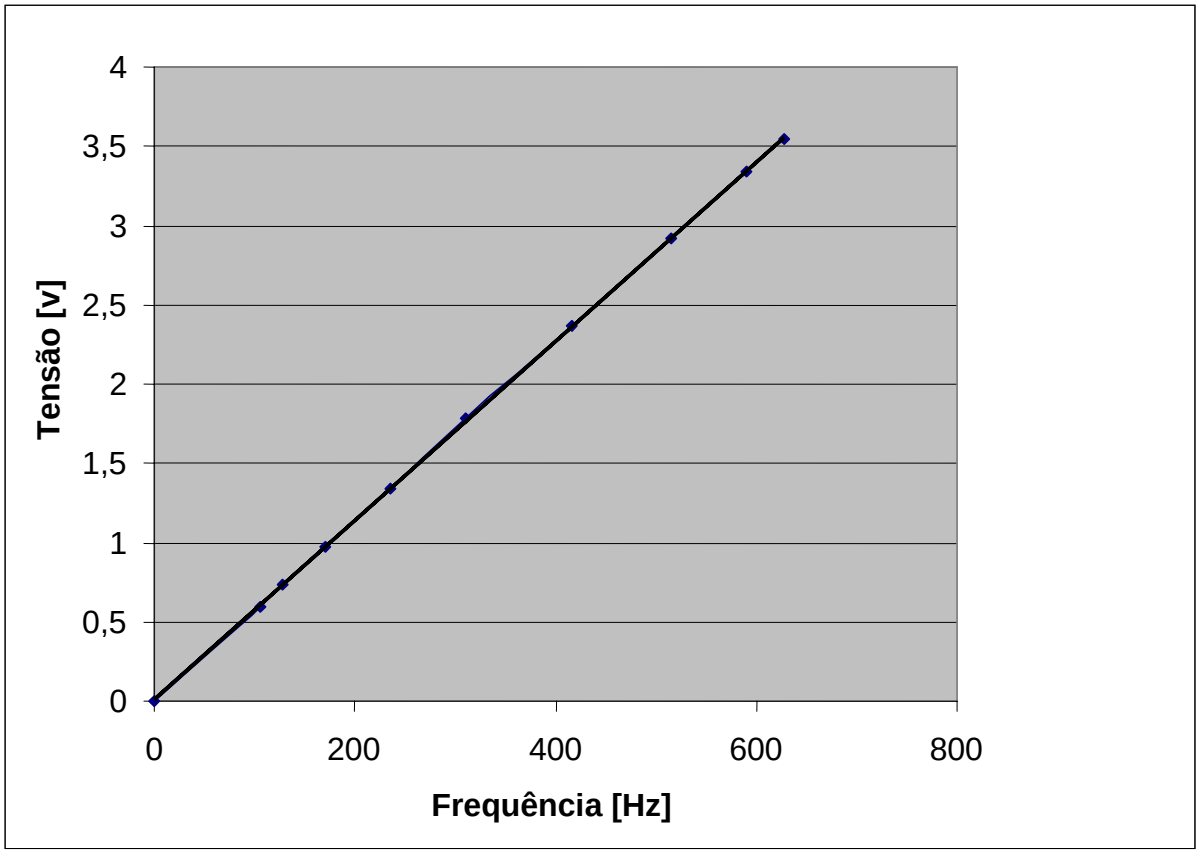

Figura 3-7 - Curva sensor de rotação

3.1.1.3.2. Corrente do motor de partida

Para a aquisição da corrente elétrica que consome o motor de partida, foi utilizado um resistor shunt em série com o motor de partida. O shunt fornece uma saída de 50 mV para uma corrente de 500 A. Para ajustar a faixa de leitura do A/D foi projetado o circuito da figura 3-8, onde o sinal é amplificado com um ganho igual a 50 e filtrado através de um filtro passa baixa RC com freqüência de corte em 100Hz.

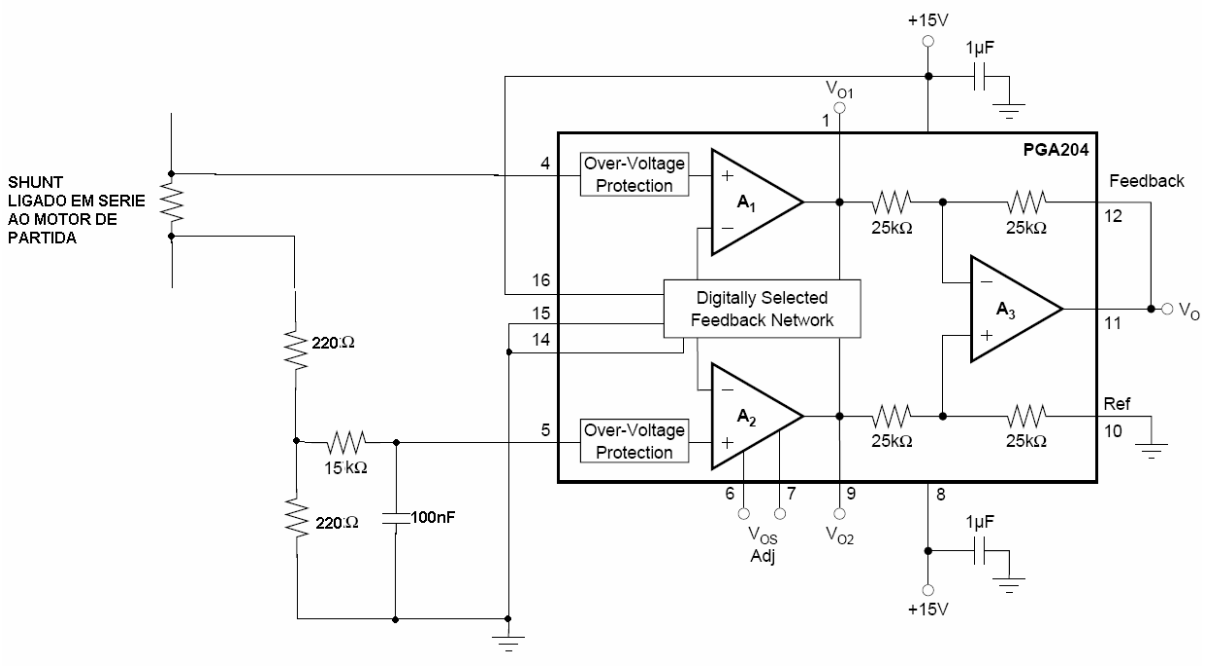

Figura 3-8 - Circuito sensor corrente.

3.1.1.3.3. Tensões

Para a leitura das tensões da bateria, motor de partida e bomba injetora, foi utilizado um circuito condicionador para ajustar a tensão, a faixa de leitura do A/D, e um filtro passa baixa RC com freqüência de corte em 100Hz foi adicionado, conforme figura 3-9.

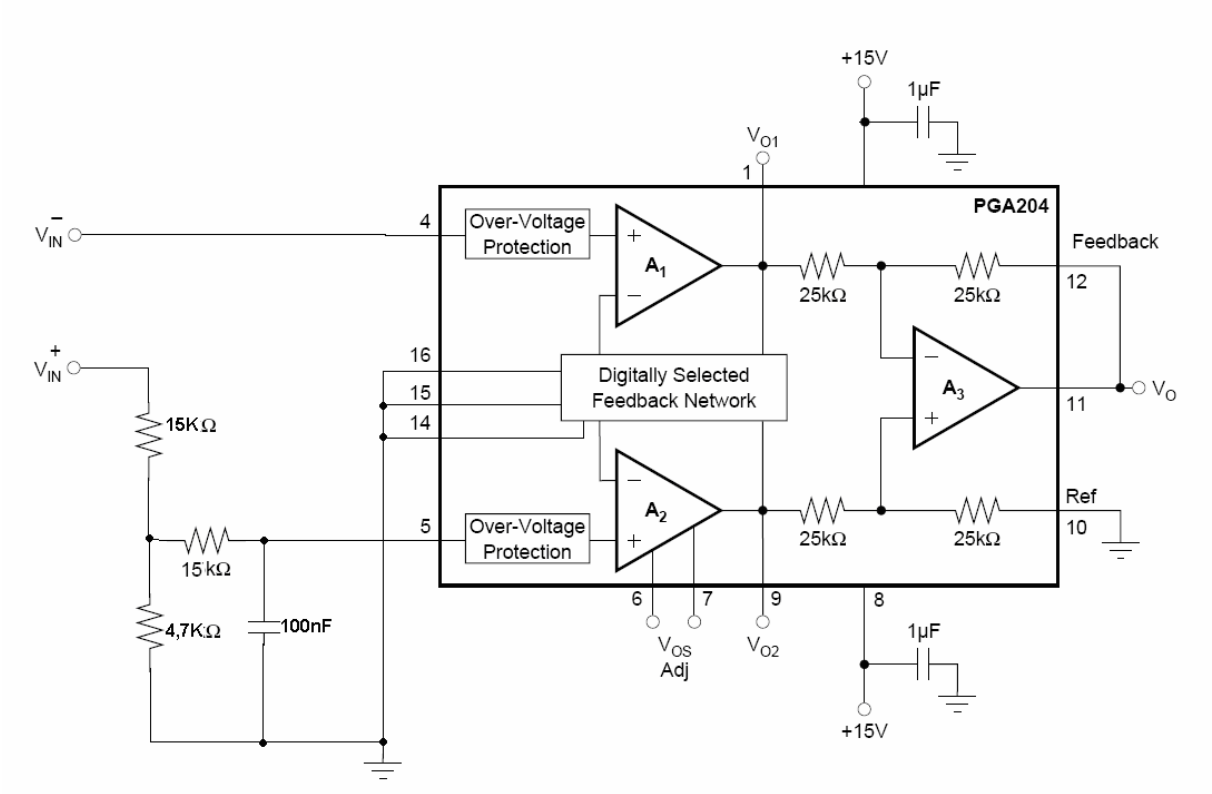

Figura 3-9 – Circuito condicionamento.

3.1.2. Módulo de comando

Neste módulo é utilizado um microcontrolador MSP430F1232 fabricado pela Texas Instruments, que é faz a interface entre o computador e o modulo de controle e aquisição.

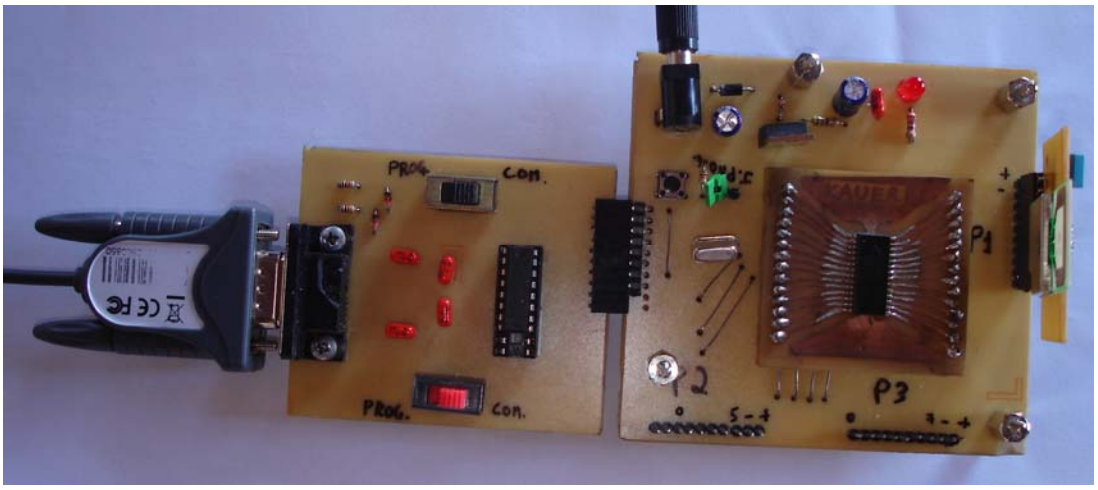

Figura 3-10 - Foto do módulo de comando

3.1.2.1. *Interface com o computador*

Para fazer a interface do microcontrolador com o computador foi desenvolvido o circuito da figura 3-11 baseado no CI MAX232, o qual é conectado na porta serial do computador.

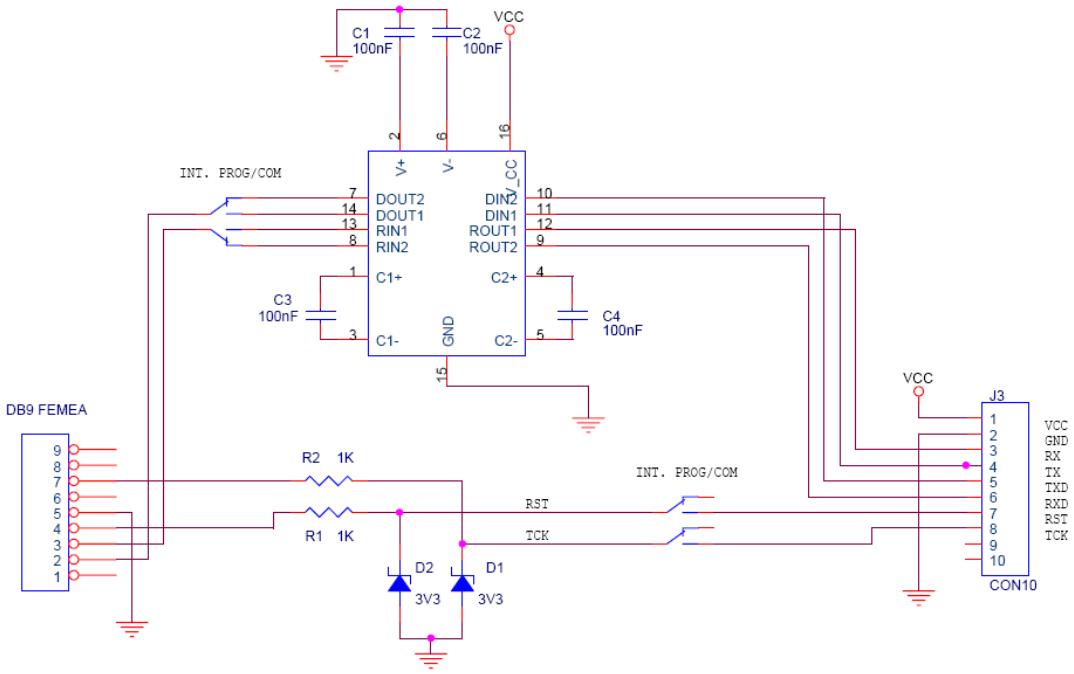

Figura 3-11 - Circuito comunicação/programação serial

Este circuito foi projetado para ser utilizado na programação dos módulos e para comunicação de dados do módulo de comando com o computador. Observa-se que existem duas chaves, quando colocadas na posição “PROG” está habilitado o modo de programação do MSP430 e quando na posição “COM” habilita o modo de comunicação de dados (figura 3-12).

Figura 3-12 – Foto da placa de comunicação/programação serial

Para programação dos módulos utilizou-se o ambiente *Embedded Workbench* é um IDE (*Integrated Development Environment* - Ambiente Integrado de Desenvolvimento), composto de um editor de arquivos, montador *assembly*, compilador C e C++, ligador, simulador e emulador. Isso significa que o programador necessita apenas de uma ferramenta de software para todo o processo de desenvolvimento utilizando microcontroladores MSP430.

Este ambiente é estruturado em *workspaces* ou espaços de trabalho que são módulos que podem agregar um ou mais projetos. Um projeto pode conter um ou mais arquivos de códigos fonte, utilizados para gerar um arquivo binário que será utilizado na simulação e programação do microcontrolador.

Para enviar o programa ao MSP430 utilizou-se o *Bootstrap Loader*, que consiste em um software gravado em uma porção de memória ROM. Este software é automaticamente executado quando se aplicam, no mínimo, dois pulsos no pino TCK, enquanto o pino reset (RST/NMI) é mantido em nível lógico baixo. A transição do nível lógico 0 para 1 no pino RST/NMI faz com que o chip saia do reset e passe a executar o BSL. Para executar o *bootstrap loader* deve-se colocar as chaves na posição “PROG” (figura 3-12).

3.1.3. Módulos de RF

O módulo RF utilizado para a comunicação entre o módulo de aquisição e o módulo de controle é o Módulo TRW-24G (figura 3-13).

O TRW-24G é um módulo da Whensing que opera em uma banda de freqüência reservada para o uso comercial ISM (*Industrial Scientific and Medical*) de 2.4 GHz e usa o chip da Nordic nRF2401VFSI com um cristal oscilador de 16MHz, uma antena dipolo embutida e um amplificador de potência. As vantagens deste módulo transmissor são:

- Conter em um só chip o módulo de recepção e transmissão;
- Usar modulação GFSK e sua comunicação é *Full Duplex*;
- Possuir dois canais de operação;
- O hardware gera o código CRC (*Cyclic Redundancy Checksum*) e confere se há erros;
- Ter uma alta velocidade na transmissão no modo *ShockBurst*;
- Ter um alcance de 280 metros para uma taxa de transferência de 250Kbps e de 150 metros se a taxa de transferência é de 1Mbps;
- Ter um baixo consumo de energia.
- Ter um codificador, decodificador e "data Buffer";
- Pelos mesmos pinos o módulo é configurado e os dados são enviados e adquiridos.

Figura 3-13 - Módulo TRW 24G (WENSHING, 2008)

Os pinos do Módulo TRW-24G são descritos abaixo (figura 3-14):

- Pino 1 – GND – *Ground* (0V);
- Pino 2 – CE – *Chip Enable*, ativo no modo recepção ou transmissão;
- Pino 3 – CLK2 – *Clock output / input* para o modo recepção do canal 2 (não utilizado neste projeto);
- Pino 4 – CS – *Chip Select*, ativo durante a configuração do Módulo TRW-24G;

- Pino 5 – CLK1 – *Clock Input* (transmissão) e I/O (recepção) do canal 1;
- Pino 6 – DATA – Recepção do canal 1 e transmissão para ambos os canais;
- Pino 7 – DR1 – Sinaliza, no modo recepção, quando os dados são recebidos pelo canal 1, somente no modo *ShockBurst*;
- Pino 8 – DOUT2 – Recepção do canal 2 (não utilizado neste projeto);
- Pino 9 – DR2 - Sinaliza, no modo recepção, quando os dados são recebidos pelo canal 2, somente no modo *ShockBurst* (não utilizado neste projeto);
- Pino 10 – VCC – Power Supply (+3V).

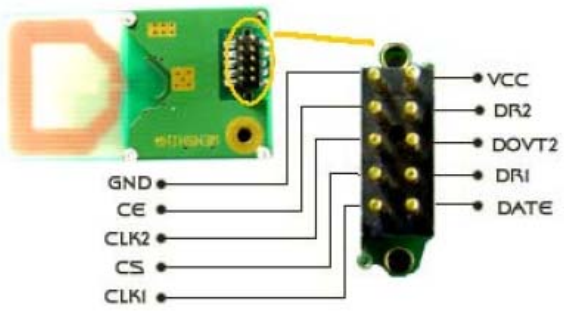

Figura 3-14 - Pinos do TRW 24G (WENSHING, 2008)

A ligação entre o MSP430 e o módulo TRW-24G é bem simples, pois ambos trabalham com alimentação e níveis lógicos na faixa de 3 v. Foi utilizada uma porta de I/O do microcontrolador conectado diretamente ao módulo de RF (figura 3-15).

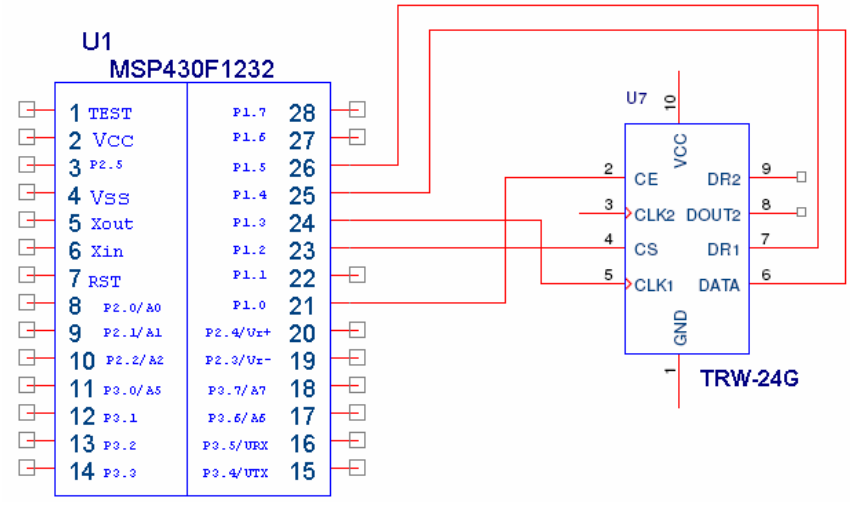

Figura 3-15 - Circuito demonstrativo da ligação entre o MSP430 e o módulo TRW-24G

Neste projeto, é utilizada a comunicação *half-duplex* para envio dos dados do microcontrolador MSP430 para o módulo TRW-24G e recebimento dos dados do módulo pelo MSP.

3.2. Software

O fluxograma da figura 3-16 mostra a rotina do ensaio que deve ser implementado. A seguir serão comentadas com maiores detalhes as rotinas de cada módulo.

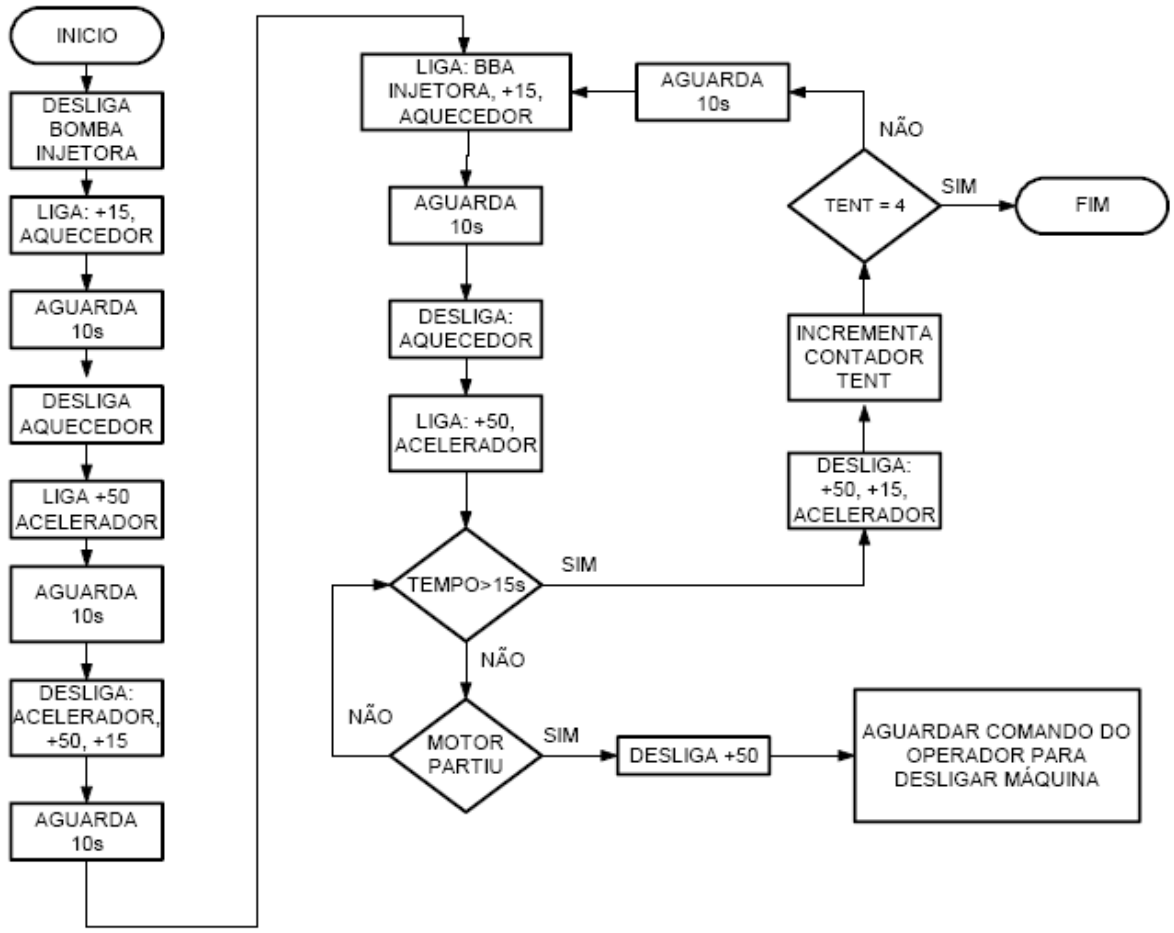

Figura 3-16 - Fluxograma geral do sistema

3.2.1. Módulo TRW-24G

Neste projeto são utilizados dois módulos TRW-24G, um ligado ao módulo de comando conectado ao PC e outro ligado ao módulo de controle e aquisição.

O módulo de comando inicialmente é configurado para selecionar e após transmitir os dados de comando para o módulo de controle e aquisição. Assim que os dados de comando são enviados, o módulo de comando é configurado para o modo de recepção e fica aguardando receber os dados adquiridos pelo módulo de aquisição e enviá-los ao PC.

Da mesma forma acontece com o módulo de controle e aquisição. Inicialmente o módulo é configurado para o modo de recepção e fica comunicando a

espera dos comandos enviados pelo módulo de comando. Assim que o módulo de controle e aquisição recebe os comandos, os interpreta e após é configurado para o modo de transmissão para que possa enviar os dados solicitados pelo módulo de comando. Após o envio dos dados, o módulo é configurado para o modo de recepção para que o mesmo fique monitorando o ar a espera dos comandos do módulo de comando.

3.2.1.1. Configuração

O Módulo TRW-2.4G é configurado com uma palavra de 144 bits (18 bytes). Nestes 144 bits são configurados o tamanho dos dados do canal 2, o tamanho dos dados do canal 1, o endereço do canal 2, o endereço do canal 1, o tamanho do endereço, o tamanho do CRC, o número de canais de recepção, modo de operação, velocidade de transmissão, freqüência de *clock*, potência de transmissão, freqüência de transmissão e o modo de operação do Módulo TRW-24G.

Enviar os dados ao Módulo TRW-24G é razoavelmente simples. A palavra da configuração é emitida ao dispositivo em série (primeiro MSB) através do pino DATA pelo microcontrolador MSP e este dá os pulsos de *clock*, no pino CLK1, de forma ordenada sucessivamente após cada bit enviado. É muito importante respeitar as exigências de temporização entre os dados enviados e o *clock* para que o Módulo TRW-24G compreenda os dados enviados a ele. No **Anexo B** foi detalhada esta temporização. O Módulo requer um *delay* mínimo de 500ns entre o envio do dado e o pulso de *clock*.

Antes de configurar o Módulo TRW-24G os pinos de CE e CS devem ser colocados para o nível lógico 0. Somente após 5cs o módulo poderá ser colocado em modo de configuração, colocando o pino CE para nível lógico 0 e o pino CS para nível lógico 1, conforme tabela 3-1. Estes comandos serão dados pelo MSP através das suas saídas digitais.

Tabela 3-1 - Modos principais do módulo TRW-24G.

	CE	CS
Modo Ativo	1	0
Configuração	0	1
Standby	0	0

Após o Módulo TRW-24G ser colocado em modo de configuração, o mesmo já está pronto para receber os bits de configuração. Assim que o módulo receber todos

os bits de configuração, o pino CS deve ser colocado para o nível lógico 0 e o pino CE deve ser colocado em nível lógico 1 para que o módulo entre em funcionamento. O fluxograma da rotina de configuração é demonstrado na figura 3-17.

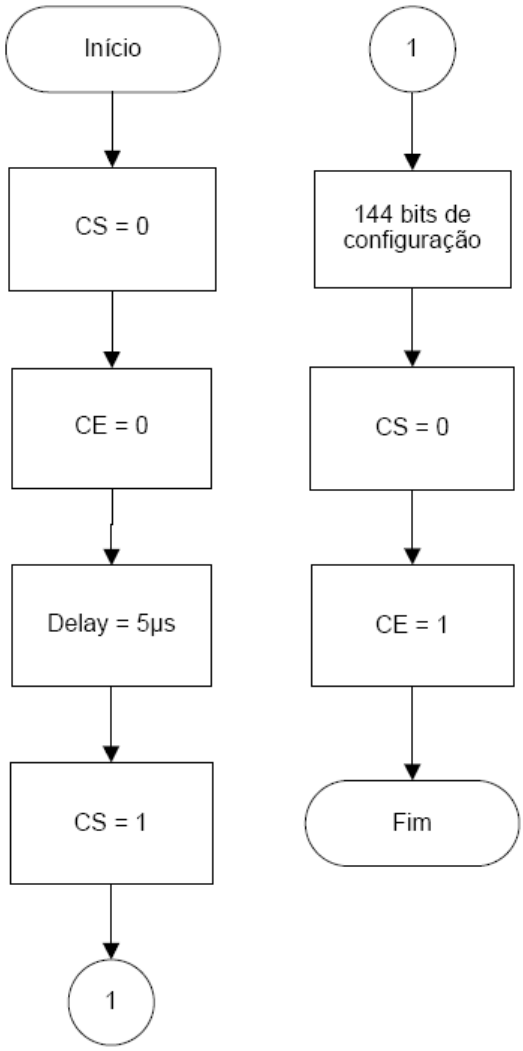

Figura 3-17 - Fluxograma da rotina de configuração

Para este projeto, o Módulo TRW-24G foi configurado para operar em uma freqüência de 2410MHz, modo *ShockBurst*, velocidade de transmissão de 1Mbps, potência de transmissão 0dB, endereço de 40-bit, dados 60 bit e 8 bit e comprimento do CRC de 16 bits.

Os 144 bits de configuração do Módulo TRW-24G são divididos e descritos da seguinte forma:

- Os Bits de Teste, do bit 143 ao bit 120, estes são reservados para testes do módulo. Estes bytes são os mais significativos, conseqüentemente são os primeiros a serem enviados pelo microcontrolador MSP430 para o módulo.

- Os Bits de Tamanho dos Dados do Canal 2, do bit 119 ao bit 112, configuram o tamanho dos dados que são enviados pelo canal 2 do Módulo TRW-24G. Neste projeto não será utilizado o canal 2.
- Os Bits de Tamanho dos Dados do Canal 1, do bit 111 ao bit 104, configuram o tamanho dos dados que são enviados pelo canal 1 do Módulo TRW-24G. Neste projeto foram utilizados 8 bits de tamanho de dados para enviar os comandos ao módulo de aquisição e 60 bits para a transmissão dos dados lidos nos canais A/D do microprocessador MSP430F169.
- O tamanho da palavra de configuração do endereço do canal 2 do módulo TRW-24G é de 5 bytes, do bit 103 ao bit 64. Neste projeto não será utilizado o canal 2.
- O tamanho da palavra de configuração do endereço do Canal 1 do Módulo TRW-24G é de 5 bytes, do bit 63 ao bit 24.
- Os Bits de ADDR_W & CRC configuram o tamanho do endereço do Módulo TRW-24G, o tamanho do CRC e a habilitação ou não do CRC. Estes bits somente serão utilizados se os módulos estiverem comunicando no modo *ShockBurst*. Os bits 23 a 18 são reservados para o ADDR_W, isto é, o tamanho do endereço do módulo. O tamanho máximo do endereço é 40 bits (5 bytes). O bit 17, CRC_L, é reservado para o tamanho do CRC que será calculado pelo módulo após o envio do ultimo bit de dados, sendo os dados enviados do microcontrolador MSP430 ao módulo. Assim que o módulo receptor receber os dados, o mesmo irá retirar o CRC enviar ao MSP somente os dados. Se este bit estiver com nível lógico 0, será configurado um tamanho de CRC de 8 bits e se o bit estiver com nível lógico 1, será configurado um tamanho de CRC de 16 bits. O bit 16, CRC_EN, é reservado para a habilitação ou não da geração do CRC quando o dado é enviado pelo módulo transmissor e para a verificação no módulo Receptor. Se este bit estiver com nível lógico 0, o CRC não será habilitado quando o módulo for configurado e se este bit estiver com nível lógico 1, o CRC será habilitado quando o módulo for configurado. Neste projeto serão utilizados 40 bits de endereço, 16 bits de CRC e o CRC está habilitado.

- Os Bits de Programação configuram o número de canais de recepção, o modo de transmissão, a velocidade de transmissão, a freqüência de oscilação do cristal e a potência de saída. O bit 15, RX2_EN, é reservado para a configuração do número de canais de recepção do Módulo TRW-24G. Se este bit estiver com o nível lógico 0, será configurado somente um canal de recepção, se este bit estiver com o nível lógico 1, serão configurados dois canais de recepção. O bit 14, CM, é reservado para a configuração do modo de transmissão. Se este bit estiver com nível lógico 0, o módulo será configurado para a transmissão no Modo Direto, se este bit estiver com nível lógico 1, o módulo será configurado para a transmissão no modo *ShockBurst*. O bit 13, RFDR_SB, é reservado para a configuração da velocidade de transmissão dos dados. Se este bit estiver com nível lógico 0, o Módulo TRW-24G será configurado para transmitir a uma velocidade de 250kbps, se este bit estiver com nível lógico 1, o Módulo TRW-24G será configurado para transmitir a uma velocidade de 1Mbps. Os bits 12 a 10, XO_F, são reservados para a configuração da freqüência do cristal. A freqüência do cristal padrão é 16MHz. Os bits 9 e 8, RF_PWR, são reservados para a configuração da potência de saída. Neste projeto será utilizado um canal de recepção, o modo de transmissão é o *ShockBurst*, a velocidade de transmissão é de 1Mbps, a freqüência do cristal é 16MHz e a potência de saída é 0dB.
- Os Bits de Freqüência de Transmissão e Direção configuram a freqüência de transmissão do canal e a direção do Módulo TRW-24G. Os bits 7 a 1, RF_CH#, são reservados para a configuração da freqüência de operação do módulo, transmissão e recepção. O bit 0, RXEN, é reservado para a configuração da direção do módulo. Se este bit estiver com nível lógico 0, o módulo será configurado para transmissor, e se o bit estiver com nível lógico 1, o módulo será configurado para receptor. Neste projeto será utilizada uma freqüência de transmissão e recepção igual a 2410MHz.

3.2.1.2. Estrutura de Envio dos Dados

O Módulo TRW-24G envia os dados de uma forma estruturada. A estrutura é formada do *Preamble*, Endereço, *Payload* e CRC. O pacote Endereço + *Payload* + CRC pode ter o tamanho máximo de 256 bits. Quando o outro módulo recebe o pacote, automaticamente retira o *Preamble*, o Endereço e o CRC e envia apenas o *Payload* para o microcontrolador MSP430.

- *Preamble* – É automaticamente calculado e inserido ao pacote de dados no modo *ShockBurst*, pelo Módulo TRW-24G, antes do envio. O *Preamble* tem o tamanho de 8 bits.
- Endereço – O tamanho do endereço pode ter no máximo 5 bytes, é adicionado no modo *ShockBurst*, para que o Módulo TRW-24G de destino saiba que o pacote foi enviado a ele.
- *Payload* – Este é o dado a ser enviado. O tamanho máximo do pode variar, levando em consideração o tamanho do endereço e do CRC que são enviados juntamente no pacote. O tamanho máximo do *Payload* é calculado da seguinte forma: Payload = 256 – Tamanho do Endereço – CRC.
- CRC – O CRC, *Cyclic redundancy check*, é um código detector de erros. É calculado e anexado à informação que será transmitida e verificada após a recepção, para confirmar se não ocorreram alterações. O CRC é popular por ser simples de implementar em hardware binário, simples de ser analisado matematicamente, e pela eficiência em detectar erros típicos causados por ruído em canais de transmissão. Pode ter o tamanho de 8 ou 16 bits.

A figura 3–18 mostra o fluxograma de como é montada a estrutura de envio de dados pelo Módulo TRW-24G.

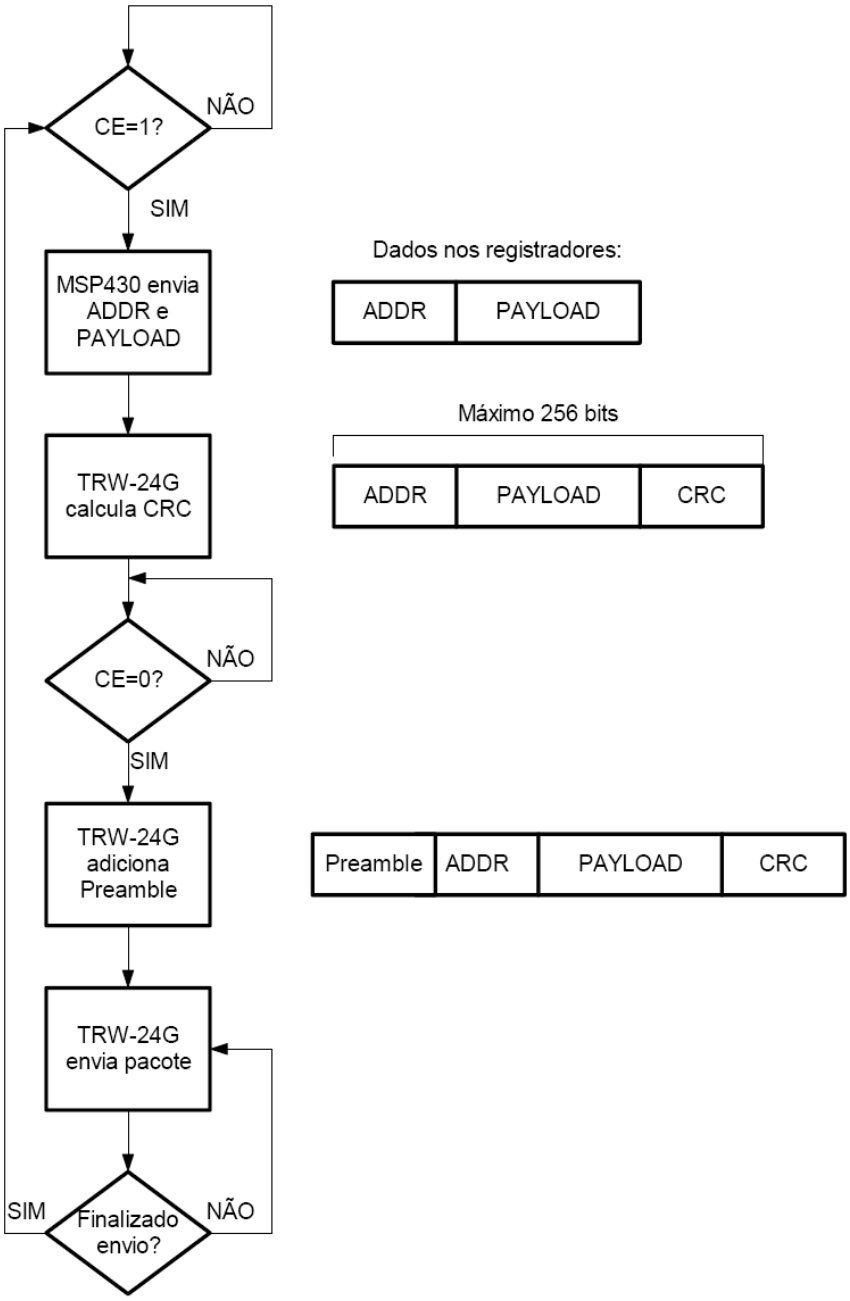

Figura 3-18 – Fluxograma de montagem da estrutura de envio de dados no TRW-24G

3.2.1.3. Transmissão

Para dar início à transmissão o pino CE deve ser colocado em nível lógico 1 e o pino CS em nível lógico 0. No modo ShockBurst o Módulo TRW-24G recebe um pacote dos dados que inclui o endereço de destino e um payload do microcontrolador MSP430. O pacote é transmitido em série (primeiro MSB), sempre

dando os pulsos de *clock* temporizados pelo MSP, da mesma maneira que é feito quando o módulo é configurado.

Assim que todo o pacote foi enviado ao dispositivo, o pino CE deve ser colocado em nível lógico 0, para ativar o processo *onboard* no Módulo TRW-24G. O módulo irá calcular e inclui, ao endereço e aos dados, o *preamble* e o CRC e só assim o módulo iniciará a transmissão dos dados. O fluxograma da rotina de transmissão é demonstrado na figura 3-19.

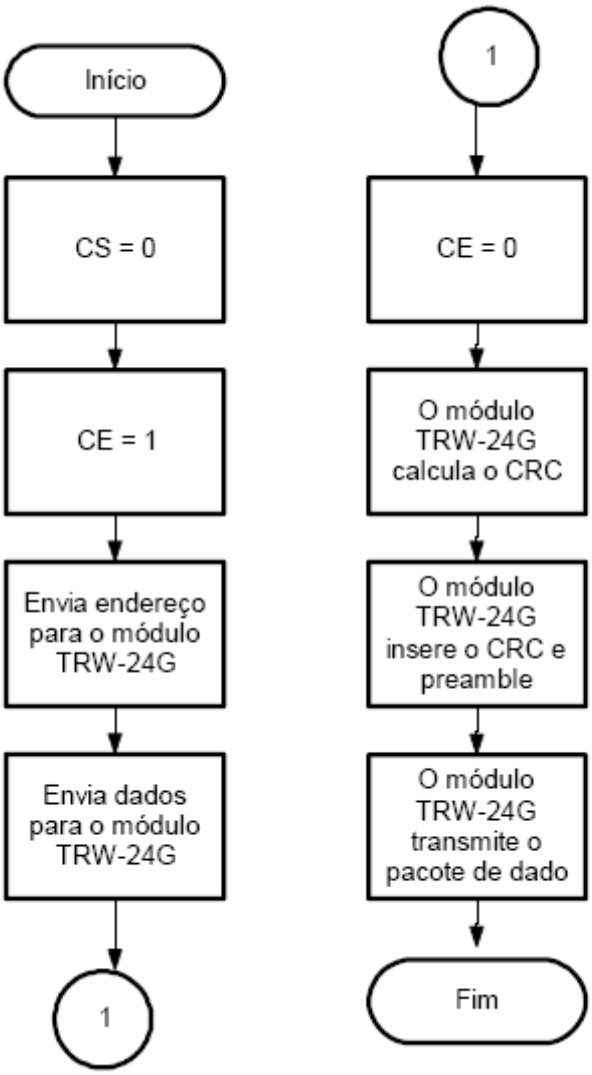

Figura 3-19 - Fluxograma de Transmissão do Módulo TRW-24G

3.2.1.4. Recepção

Para dar início à recepção dos dados, o pino CE deve ser colocado em nível lógico 1 e o pino CS em nível lógico 0. No modo *ShockBurst* o Módulo TRW-24G recebe um pacote de dados que inclui o endereço de destino, o *payload*, o *preamble* o CRC. Assim que todo o pacote for recebido, o módulo retira automaticamente o

endereço de destino, o *preamble* e o CRC. Acabado este processo ele coloca o pino DR1 para nível lógico 1, sinalizando que dados foram recebidos.

Assim que o microcontrolador MSP430 detectar que o pino DR1 do Módulo TRW-24G está em nível lógico 1, o MSP deverá colocar o pino CE para nível lógico 0 durante a recepção dos dados pelo MSP. Porém o pino CE poderá ficar com o nível lógico em 1 durante a recepção dos dados pelo MSP, pois isso somente acarretará um maior consumo de corrente elétrica, aproximadamente 18mA.

O pacote de dados é lido, pelo microcontrolador MSP430, em série (primeiro MSB), sempre sincronizando os pulsos de *clock* com o bit recebido. Assim que o Módulo TRW-24G enviar todos os dados para o MSP, o pino DR1 do módulo será levado para o nível lógico 0 e o MSP deverá colocar o pino CE para o nível lógico 1 para que o módulo comece a monitorar o ar a espera de novos pacotes de dados. O fluxograma da rotina de recepção é demonstrado na figura 3-20.

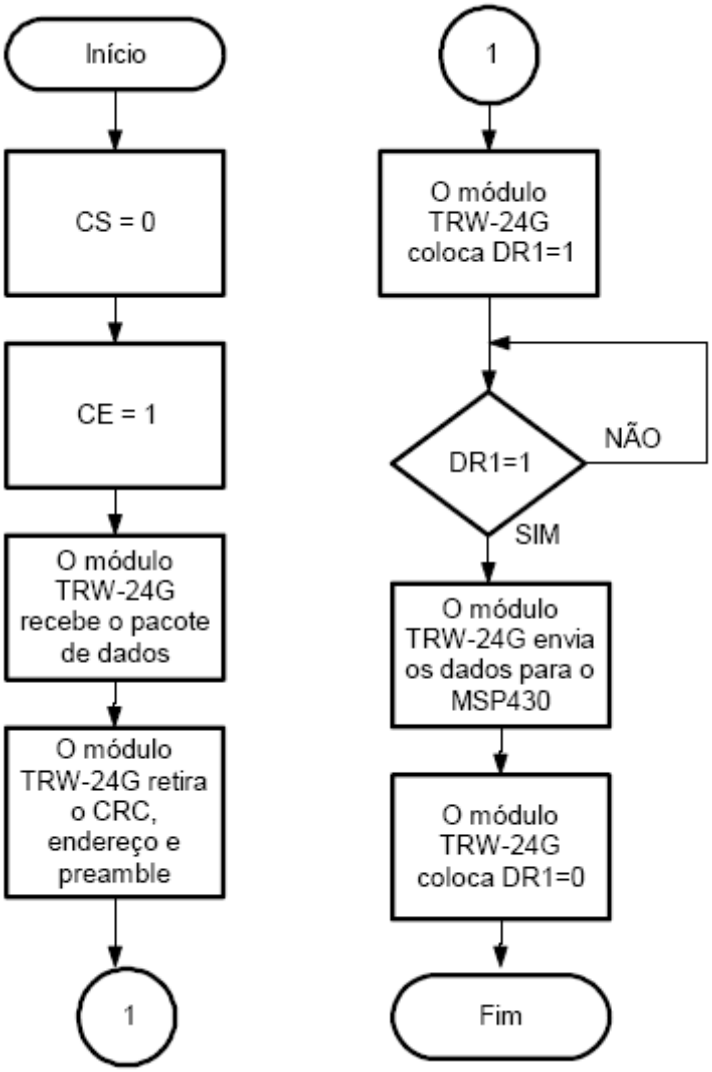

Figura 3-20 - Fluxograma de recepção do módulo TRW-24G

3.2.2. Módulo de controle e aquisição

A inicialização do "firmware" é uma parte do programa bastante importante, pois ali devem estar as configurações de funcionamento do micro-controlador que, se não estiverem de acordo com a necessidade do programador, podem causar problemas no algoritmo ou mau funcionamento das rotinas. Inclusive, possíveis problemas que podem parecer estar em funções, podem estar em algum erro de parametrização.

A configuração dos A/Ds foi feita através dos registradores ADC12CTL0 e ADC12CTL1. No primeiro, os seguintes bits são ajustados: MSC, que significa "modo de conversão repetitiva", ou seja, no momento da aquisição, todas as portas (de A/D) serão lidas, a uma freqüência de 200KHz; ADC12ON, que liga o módulo ADC12; e o SHTO que é o tempo de amostragem, configurado para amostrar o sinal durante 256 ciclos de *clock*. No segundo registrador é configurado o bit SHP, que ajusta a amostragem para durar o tempo definido pelo bit SHTO, do primeiro registrador, e o CONSEQ_3, que também significa "modo de conversão repetitiva".

Os registradores ADC12MCTLx ("x" de 0 a 4) são configurados para ter referência de tensão com AVcc e AVss, ou seja, a alimentação do micro-controlador (0 a 3,48v), e ter como entrada os canais de A0 a A4, respectivamente.

Também é configurado o módulo TRW-24G como receptor, para que fique monitorando a espera dos comandos do módulo de comando.

Após as configurações descritas o MSP430 inicia as conversões dos A/Ds e a cada oito leituras é feita uma média destes valores, a fim de efetuar um filtro digital. Assim que os comandos são recebidos, o MSP os interpreta e configura o módulo TRW-24G como transmissor, enviando os dados lidos pelos A/Ds. O fluxograma da rotina do módulo de aquisição e controle é visto na figura 3-21.

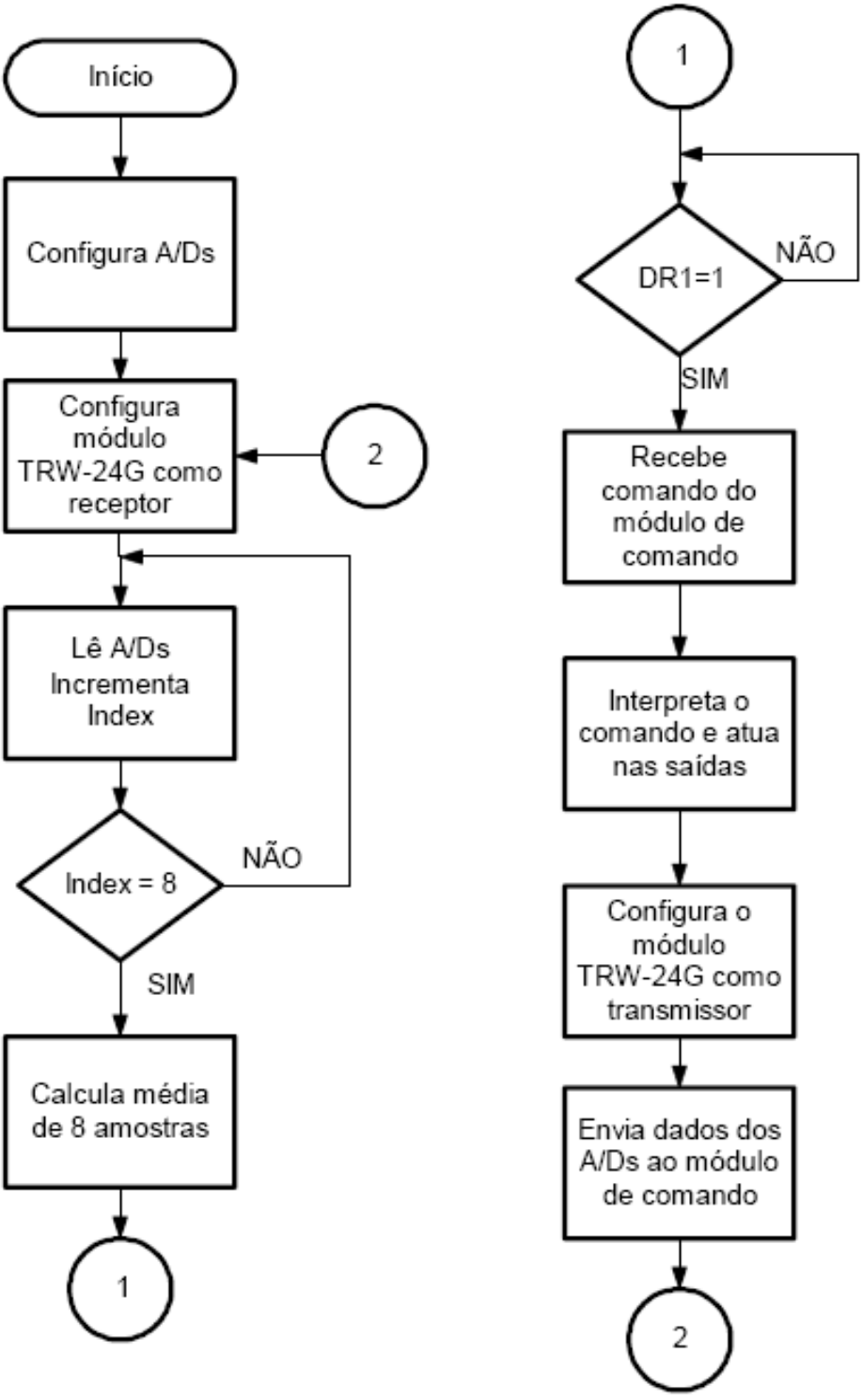

Figura 3-21 - Fluxograma da rotina do módulo de controle e aquisição

3.2.3. Módulo de comando

Inicialmente é ativado o *watchdog* através do registrador WDTCTL, são ajustados os seguintes bits: WDTPW que é a senha para habilitar a escrita no registrador; WDTCNTCL que apaga a contagem do *watchdog*; para os demais bits foi mantida a configuração padrão utilizando o *watchdog* para reiniciar o sistema caso alguma falha ocorra.

Também foi realizada a configuração do registrador "BCSCTL1", que é um dos responsáveis pelo controle do módulo oscilador. Foi selecionada a configuração para a operação do micro-controlador com "*clock*" externo, que é gerado pelo cristal de 8Mhz. Outras configurações, referentes aos registradores "IFG1" e "BCSCTL2", também são referentes ao "*clock*" do dispositivo, e realizam a conferência da integridade do mesmo.

Os parâmetros da porta serial também foram ajustados: habilitação da recepção e envio de dados pela "UART 0", no registrador "ME1"; ajuste de 8 bits de dados por byte, paridade ímpar e controle de pacotes por inatividade de linha, pelo registrador "U0CTL"; utilização do cristal externo, pelo "U0TCTL" e ajuste da taxa de transmissão através dos registradores "UBR00", "UBR10" e "UMCTL0" para 115200 bps. Ainda sobre a comunicação serial, a linha "U0CTL&=~SWRST;" inicializa o funcionamento da porta.

Também é configurado o módulo TRW-24G como transmissor, para que assim que receba um comando do computador envie os comandos do ao módulo de aquisição e controle. Após envio do comando ele é configurado como receptor para fique aguardando o recebimento dos dados.

Assim que os dados são recebidos pelo módulo TRW-24G ele informa o MSP através do pino DR1, que os dados estão prontos para serem lidos. O MSP lê os dados e envia através da serial para que seja registrado no computador. Na figura 3-22 o fluxograma da rotina do módulo de comando.

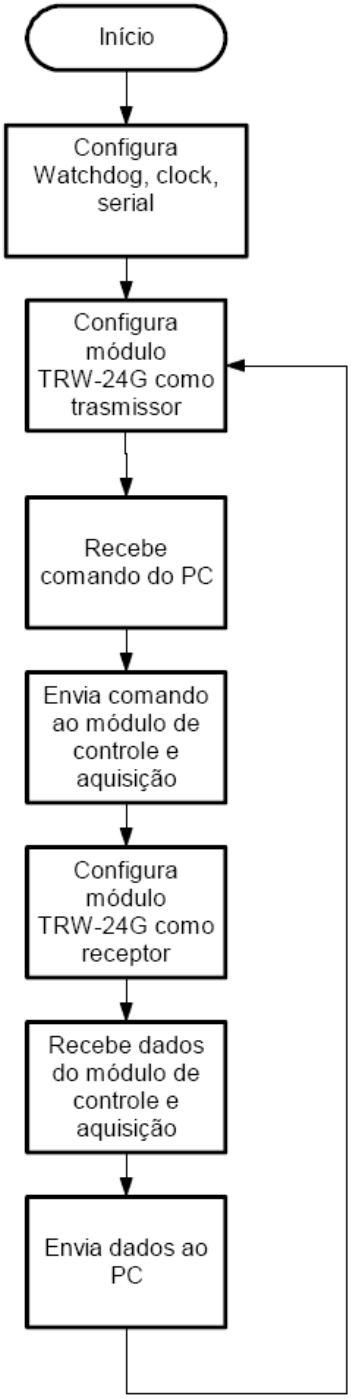

Figura 3-22 - Fluxograma da rotina do módulo de comando

3.2.4. IHM (Interface Homem Máquina)

Foi desenvolvida uma interface visual no software Borland C++ Builder, para permitir a interação do operador em tempo real à execução do teste. Este software apresenta uma tela gráfica, figura 3-23, onde é feito a inicialização da porta serial e o controle do teste através dos botões “Inicia Teste” e “Parar Teste”. Também são apresentados em forma de gráficos os valores das seguintes variáveis:

tensão na bateria, no motor de partida e na bomba injetora, corrente no motor de partida e rotação do motor. Nesta tela também são apresentados o estado das saídas controladas pelo sistema e o andamento do teste.

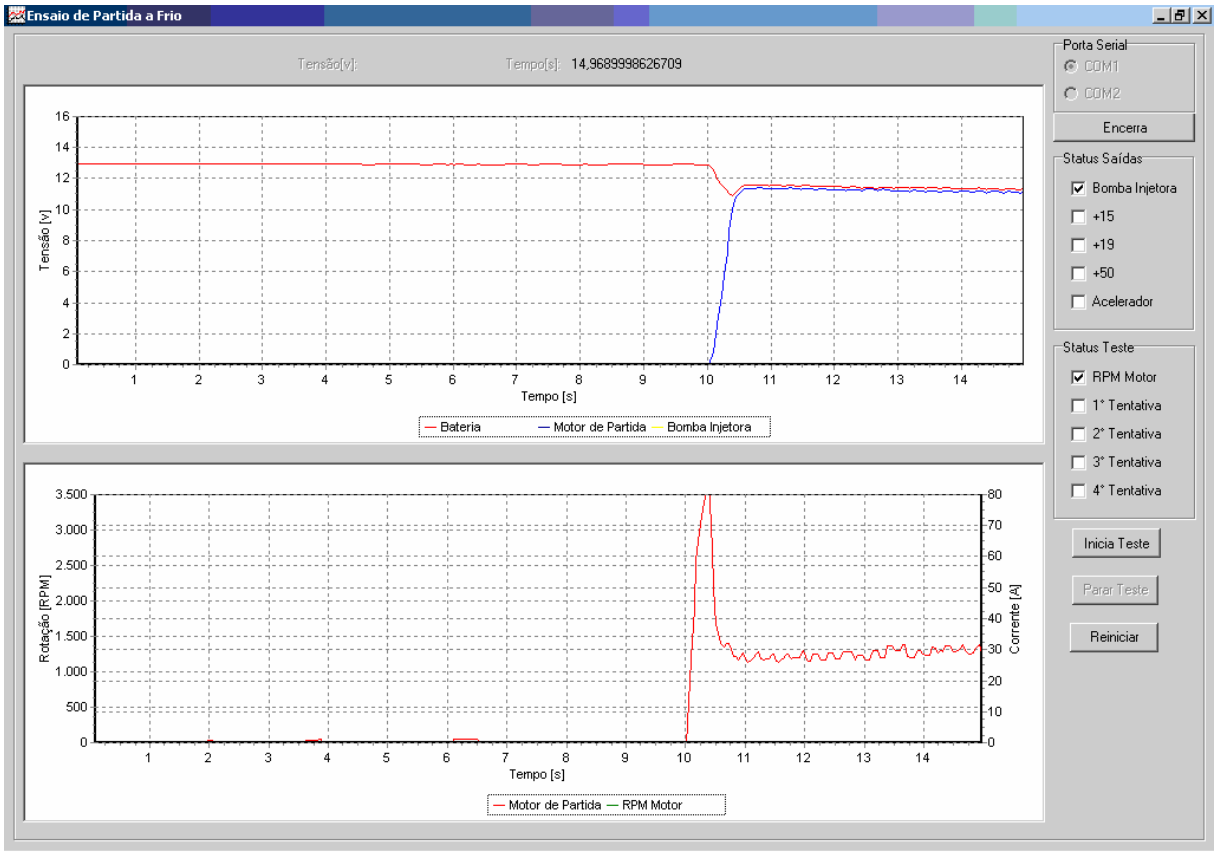

Figura 3-23 - Interface IHM

3.3. *Testes e Validação*

3.3.1. *Testes preliminares*

Inicialmente o sistema foi montado sem os módulos de RF, sendo o módulo de aquisição e controle ligado diretamente na serial do PC. Foi montado um circuito com potenciômetros variando de 0 a 12 v, para simular os valores de tensão a serem lidos durante o ensaio. Para a simulação da corrente no motor de partida foi montado um potenciômetro variando de 0 a 50mV, que é a faixa de saída do shunt para uma corrente de 0 a 500A. Um gerador de funções foi utilizado para simular a saída do sensor de rotação, aplicando-se uma onda quadrada de 12 V de amplitude e freqüência variando de 0 a 600 Hz.

Foram corrigidos alguns erros iniciais de projeto e certificou-se que os valores mostrados na tela correspondiam aos valores nas entradas.

Para inclusão dos módulos de RF foi desenvolvida as rotinas de configuração, transmissão e recepção. Com o auxílio do osciloscópio foram medidos os valores que saiam do MSP até o módulo, após alguns ajustes certificou-se que as temporizações exigidas pelo módulo estavam corretas.

Repetiram-se os mesmos testes executados anteriormente sem os módulos de RF, e após algumas correções foi verificado o correto funcionamento do sistema.

3.3.2. Funcionamento do projeto

O sistema desenvolvido foi montado em motor a combustão, a fim de simular as condições reais das variáveis a serem medidas. As figuras 3-24 3-25 e 3-26 mostram a montagem do sistema para os testes de aquisição.

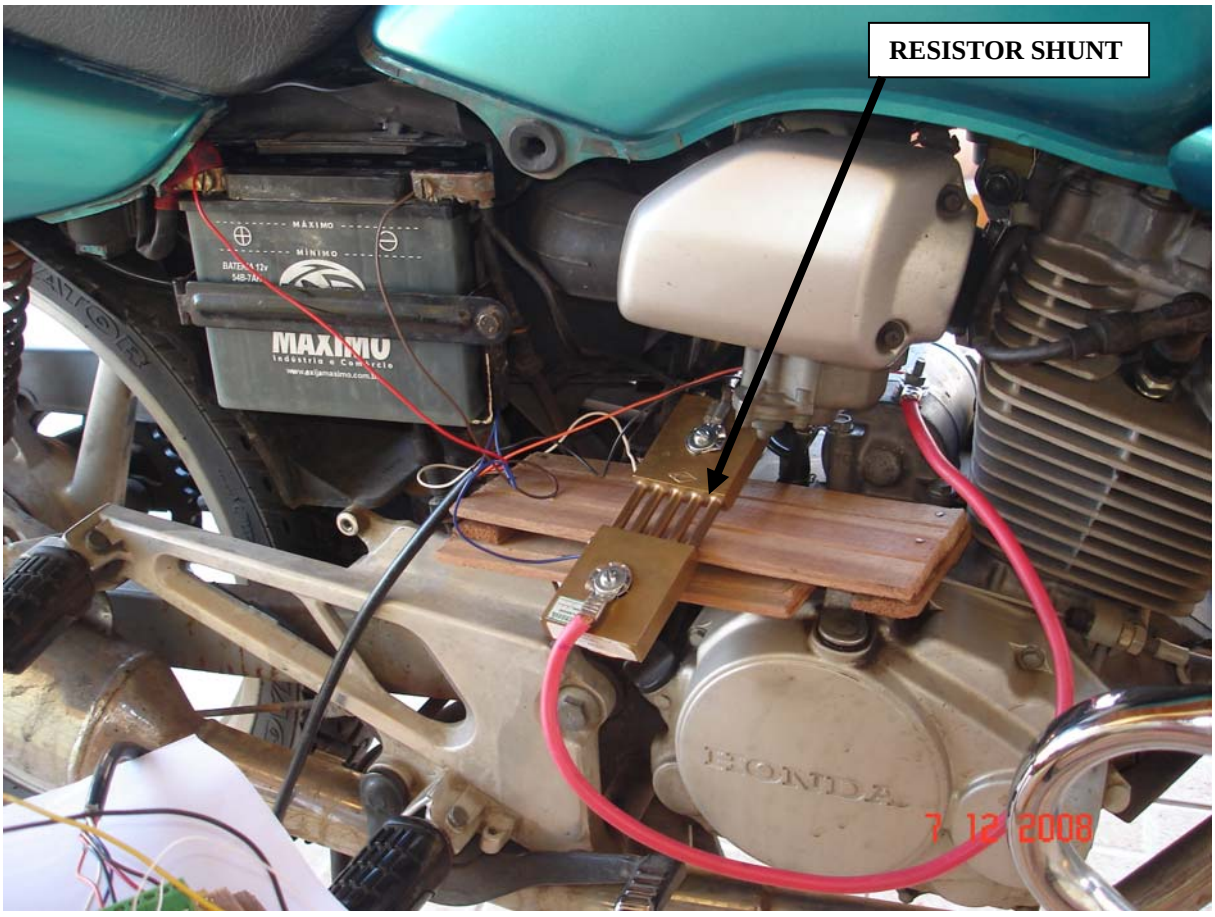

Figura 3-24 - Montagem do resistor shunt

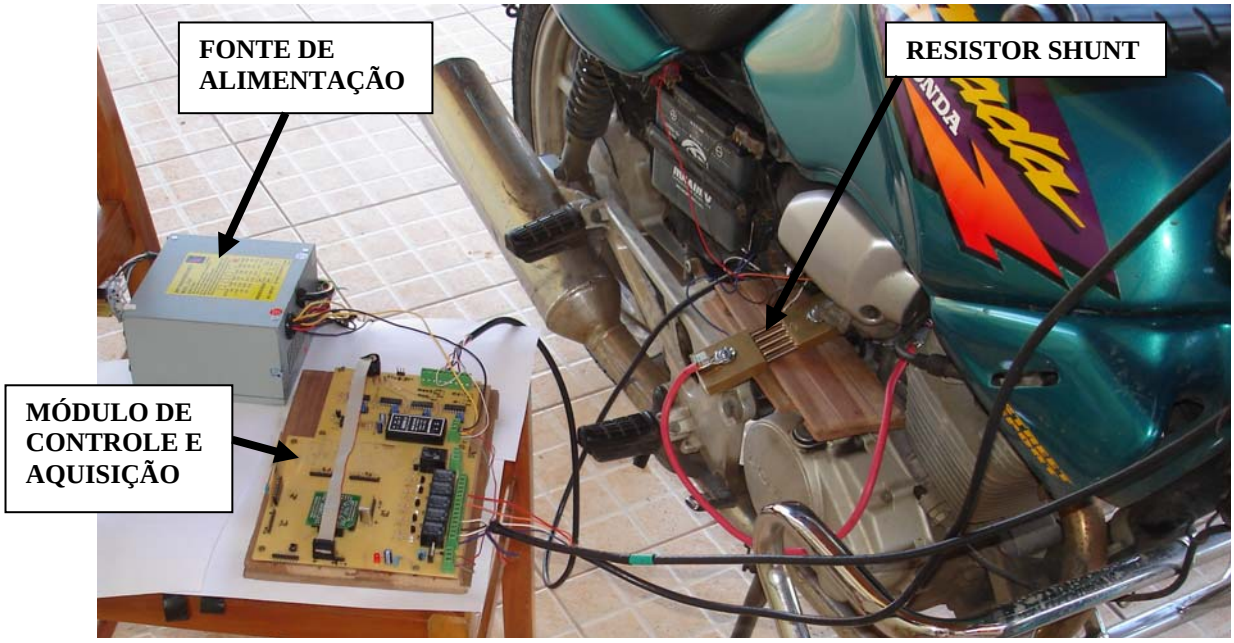

Figura 3-25 - Módulo de controle e aquisição

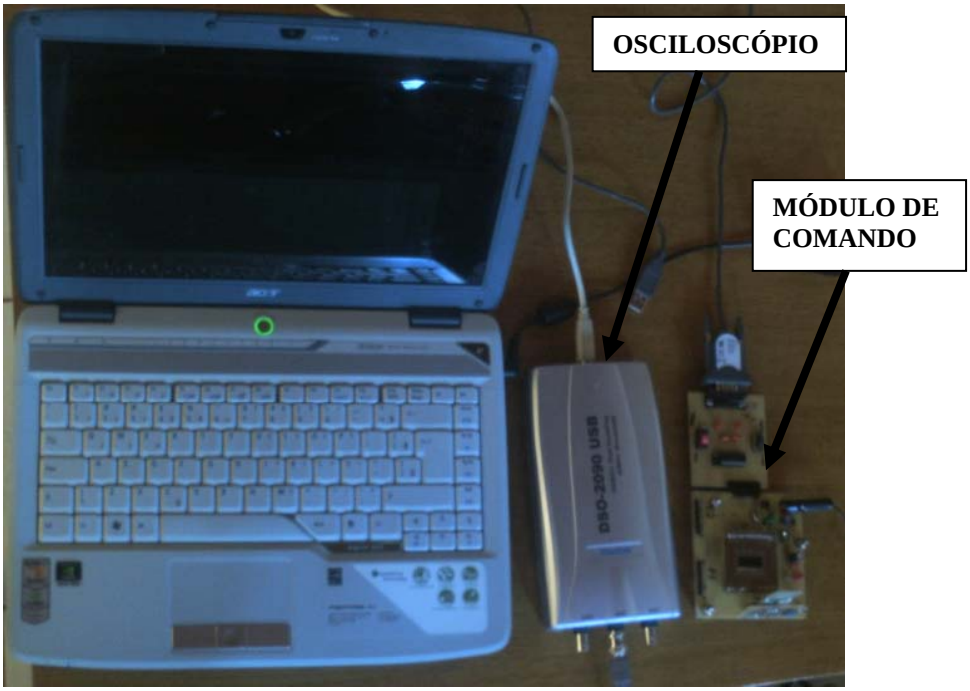

Figura 3-26 - Módulo de comando e osciloscópio

Um osciloscópio foi utilizado para ler os valores a serem adquiridos comparando-os com os valores obtidos no sistema desenvolvido. Após alguns ajustes obteve-se a resposta esperada conforme mostram figuras 3-27 e 3-28. Nestas figuras o gráfico da parte superior foi obtido através do osciloscópio e o gráfico da parte inferior através do sistema desenvolvido.

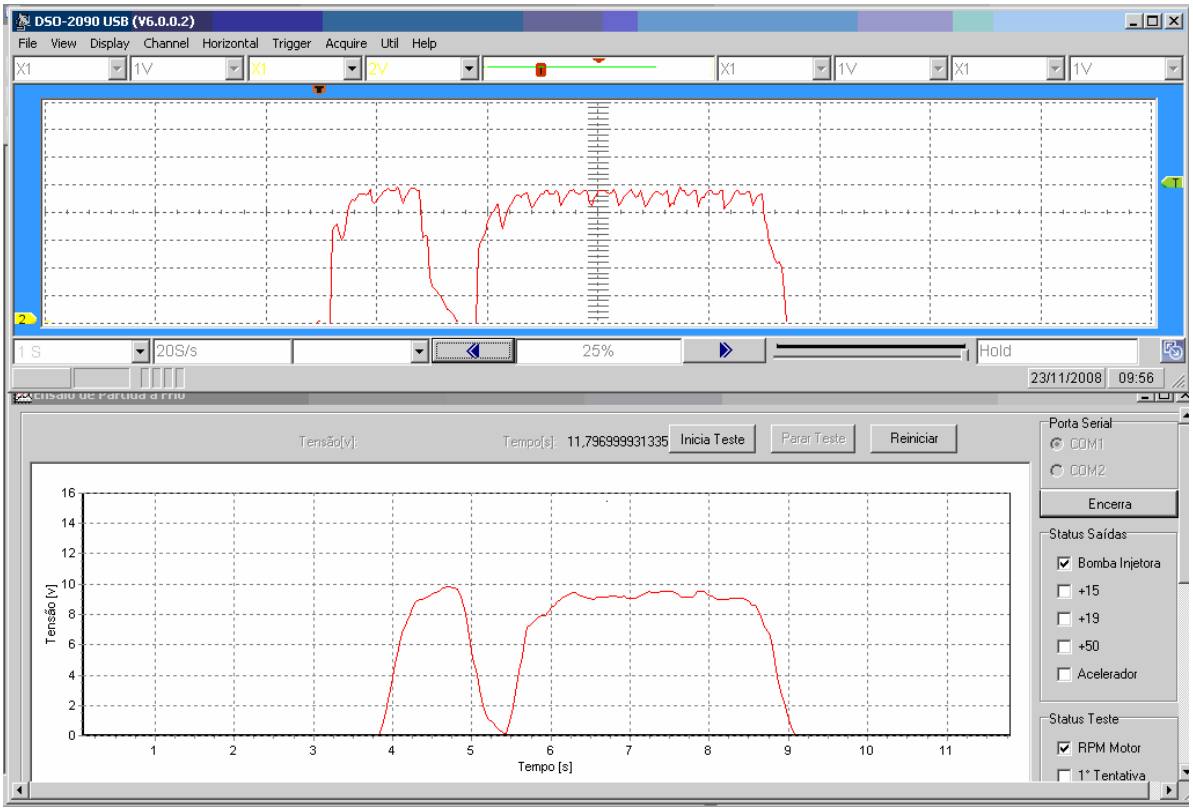

Figura 3-27 - Gráficos da tensão no motor de partida.

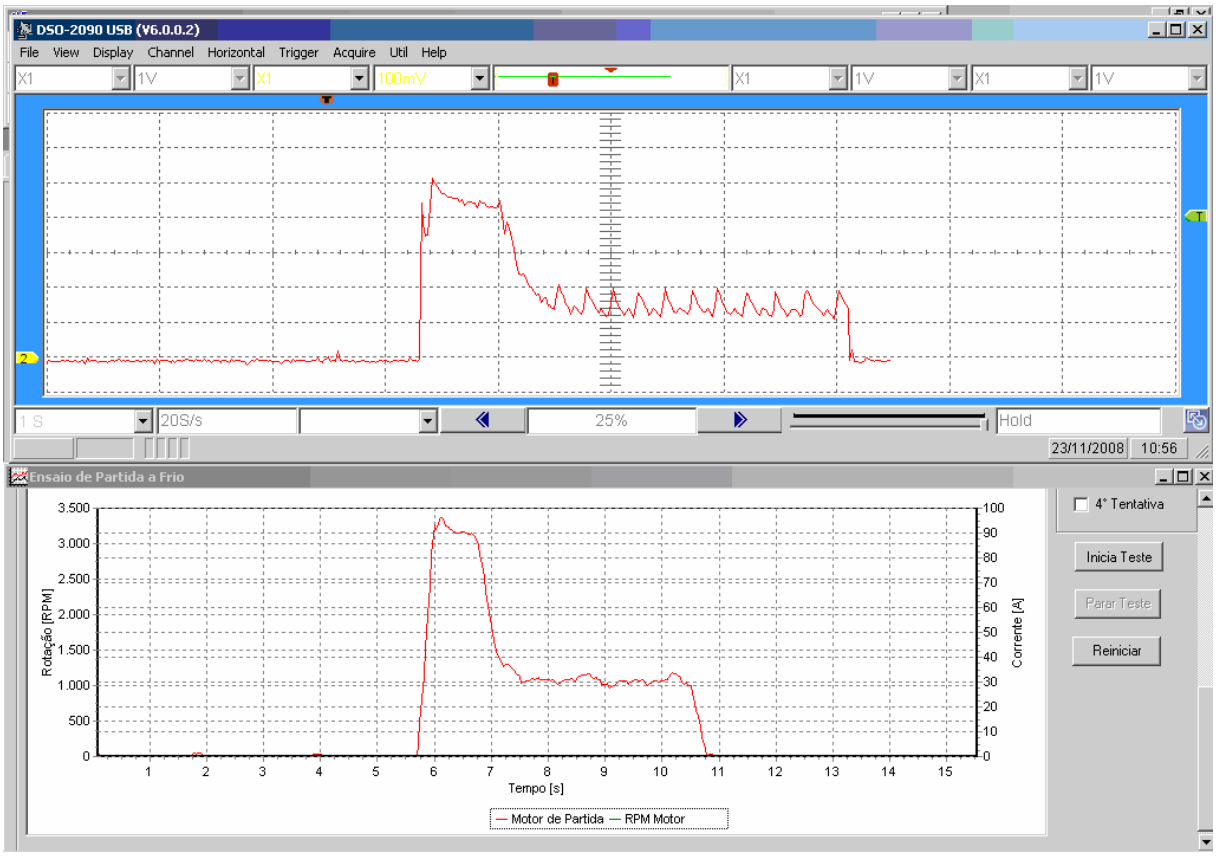

Figura 3-28 - Gráficos da corrente no motor de partida

Como pode ser visto nos gráficos das figuras 3-27 e 3-28, o sistema desenvolvido elimina a oscilação produzida durante a partida, introduzindo assim um erro máximo de 4% a medida, que é considerado aceitável para este projeto. Este erro foi calculado comparando os valores lidos com o osciloscópio DSO-2090 e o sistema desenvolvido, considerou-se este osciloscópio como referência para a calibração do sistema.

3.3.3. Análise dos dados

Foram injetados na entrada do sistema valores conhecidos e analisou-se a resposta do sistema. Para cada valor colocado na entrada, foi monitorada a saída através do software desenvolvido realizando 45 leituras, sendo 15 em cada uma das temperaturas de 29, 12 e 2 ºC. Destas 45 leituras obteve-se a média, o desvio padrão e o desvio padrão da média. Nas tabelas 3-2, 3-3 e 3-4 cada linha representa um valor de medida, foram escolhidos 9 pontos visando alcançar todo o *range* de leitura do sistema para cada variável.

Tabela 3-2 - Análise dos dados das leituras de tensão.

Média	Desvio Padrão	Desvio Padrão da Média	Percentual Desvio Padrão da Média
0.8127	0.0108	0.0020	0.2%
1.7193	0.0187	0.0035	0.2%
3.9300	0.0214	0.0040	0.1%
5.0438	0.0825	0.0153	0.3%
7.1221	0.0927	0.0172	0.2%
8.0393	0.0304	0.0045	0.1%
9.2514	0.0663	0.0123	0.1%
10.0110	0.0263	0.0048	0.0%
11.8390	0.0760	0.0141	0.1%

Tabela 3-3 - Análise dos dados da leitura de corrente.

Média	Desvio Padrão	Desvio Padrão da Média	Percentual Desvio Padrão da Média
50.8800	1.2622	0.3259	0.6%
96.8746	1.5743	0.2874	0.3%
135.9367	0.9021	0.2329	0.2%
200.6667	1.3973	0.3608	0.2%
269.7333	1.7099	0.4415	0.2%
351.5200	3.4349	0.8869	0.3%
400.8887	1.3228	0.3415	0.1%
451.6720	2.7255	0.7037	0.2%
483.2667	3.3481	0.8645	0.2%

Tabela 3-4 - Análise dos dados da leitura de rotação.

Média	Desvio Padrão	Desvio Padrão da Média	Percentual Desvio Padrão da Média
67.3600	1.6932	0.4372	0.6%
114.0853	2.0710	0.5347	0.5%
200.4427	3.1096	0.8029	0.4%
353.2267	2.4271	0.6267	0.2%
528.0933	2.0422	0.5273	0.1%
1014.2827	16.8612	4.3535	0.4%
1554.7433	16.6058	4.2876	0.3%
2331.7333	25.2487	6.5192	0.3%
3746.7333	18.0494	4.6603	0.1%

Obteve-se um desvio padrão da média de no máximo 0,6% para a corrente e para a rotação, para os valores de tensão o desvio padrão da média é de no máximo 0,3%. Pode-se então caracterizar a medidas do sistema da seguinte forma:

V= Valor lido ± 0,3% V

I= Valor lido ± 0,6% V

RPM= Valor lido ± 0,6% rpm

4. CONCLUSÕES

Este trabalho teve como objetivo geral implementar melhorias no ensaio de partida a frio de máquinas agrícolas. Os objetivos específicos propostos para cumprir com este objetivo foram:

- Desenvolver um sistema de controle e monitoração microcontrolado, responsável por comandar a partida da máquina e monitorar as variáveis envolvidas.

  Este objetivo foi atendido com o desenvolvimento dos módulos de comando e de controle e aquisição, bem como os softwares destes módulos e da IHM.

- Desenvolver a comunicação de dados via RF para interligar a máquina com a cabine de controle.

  Para atender este objetivo foram utilizados os módulos TRW-24G, que realiza a transmissão dos dados entre o módulo de comando e o módulo de controle e aquisição.

O projeto apresentou os resultados esperados e atendeu o que inicialmente havia sido proposto a ser desenvolvido.

Este trabalho pode ser adaptado para diversas aplicações em uma planta industrial que tenha a necessidade da aquisição de dados sem fio. Um exemplo seria aplicar em testes de campo em máquinas agrícolas, onde pode-se visualizar os resultados a distância durante o trabalho da máquina no campo.

5. REFERÊNCIAS

LUECKE, J. **Analog and Digital Circuits for Eletronic Control System Applications**. Burlington: Elsevier, 2004. 312p. ISBN 0-7506-7810-0.

PEREIRA, F. **Microcontroladores MSP430: Teoria e Prárica**. 1. Ed. São Paulo: Érica, 2005. 414 p. ISBN 85-365-0067-0.

LINAK Group. **Sobre os Atuadores Lineares**. Disponível em: <http://www.linak.com.br/Products/?id3=1630>. Acesso em: 10 out. 2008.

WIKIPEDIA. **Shunt (electrical).** Disponível em: <http://en.wikipedia.org/wiki/Shunt_(electrical)>. Acesso em 15 set. 2008.

WENSHING, Electronics. **TRW-24G – Wireless High Frequency Transceiver Module (RF GFSK).** Disponível em: <http://www.wenshing.com.tw/data/rf-module/TRW-24G.pdf> Acesso em: 10 ago. 2008.

TEXAS, Instruments. **MSP430x15x, MSP430x16x, MSP430x161x Mixed Signal Microcontroller.** Disponível em: <http://focus.ti.com/lit/ds/symlink/msp430f1611.pdf> Acesso em: 10 ago. 2008.

TEXAS, Instruments. **MSP430x11x2, MSP430x12x2 Mixed Signal Microcontroller.** Disponível em: <http://focus.ti.com/lit/ds/symlink/msp430f1232.pdf> Acesso em: 10 ago. 2008.

AGCO, Corporation. **Machine Cold Starting Procedure:GF10700001.** Hesston, 2005.

OBRAS CONSULTADAS

HAYKIN, S. **Sistemas de Comunicação – Analógicos e Digitais.** 4. Ed. São Paulo: Bookman,2004. 840 p. ISBN 85-730-7936-3.

YOUNG, P. H. **Técnicas de Comunicação Eletrônica.** 5. Ed. São Paulo: Pearson Prentice Hall, 2006. 704 p. ISBN 85-760-5049-8.

REGTIEN, P. P. L.. **Instrumentation Electronics**. 5. Ed. New York: Prentice Hall, 1992. 489 p.

MARIN, R. **Aquisição de dados por uma rede sem fio.** 2007. 124 f. Monografia (Graduação) – Faculdade de Engenharia Elétrica, Universidade Luterana do Brasil. Canoas. 2007.

APÊNDICE A – ROTINAS DO SOFTWARE MSP430F169

```
#include  <io430x14x.h>
#include  <intrinsics.h>
#define CE P4OUT_bit.P4OUT_0
#define CS P4OUT_bit.P4OUT_2
#define CLK1 P4OUT_bit.P4OUT_3
#define DATA P4OUT_bit.P4OUT_4
#define DATAIN P4IN_bit.P4IN_4
#define DR1 P4IN_bit.P4IN_5
#define   Num_of_Results   8 //para ad
void delayus(void);
void delayms(unsigned int j);
void init_24g(void);
void write_24g(unsigned char dado);;
void write_24g_12(unsigned int dado);
void configura_24g(unsigned char mode);
void configura_24g_12(unsigned char mode);
void send_24g(unsigned char dado);
void send_24g_12(unsigned int dado);
unsigned char read_24g(void);
unsigned int read_24g_12(void);
void teste(void);
void send_24g_60(void);
void write_24g_60(void);
void configura_24g_60(unsigned char mode);
unsigned char leitura;
static unsigned int A0results[Num_of_Results];  //vetores para armazenar
```

```
static unsigned int A1results[Num_of_Results];  //os valores lidos nos ads
static unsigned int A2results[Num_of_Results];
static unsigned int A3results[Num_of_Results];
static unsigned int A4results[Num_of_Results];
static unsigned int resultados[5];
unsigned int ad,ad1,ad2,ad3,ad4,conta=0;

void main(void)
{
  // Stop watchdog timer to prevent time out reset
  WDTCTL = WDTPW + WDTHOLD;
  P1DIR |= 0XFF;
  P1OUT = 0;
    //configura ad
  P6SEL = 0x0F;                          // Enable A/D channel inputs
  ADC12CTL0 = ADC12ON+MSC+SHT0_8;// Turn on ADC12, extend sampling time
                                          // to avoid overflow of results
  ADC12CTL1 = SHP+CONSEQ_3;          // Use sampling timer, repeated sequence
  ADC12MCTL0 = INCH_0;                // ref+=AVcc, channel = A0
  ADC12MCTL1 = INCH_1;                // ref+=AVcc, channel = A1
  ADC12MCTL2 = INCH_2;                // ref+=AVcc, channel = A2
  ADC12MCTL3 = INCH_3;                // ref+=AVcc, channel = A3
  ADC12MCTL4 = INCH_4+EOS;            // ref+=AVcc, channel = A4, end seq.
  ADC12IE = 0x10;                       // Enable ADC12IFG.4
  ADC12CTL0 |= ENC;                     // Enable conversions
//fim configura ad
  P4DIR = 0X1D;
  init_24g();
  delayms(3);
  configura_24g(0);
  P4DIR = 0X0D;
  CS=0;
  delayus();
  CE=1;
  delayus();
  ADC12CTL0 |= ADC12SC;                 // Start conversion
  __bis_SR_register(LPM0_bits + GIE);
```

```
}

//Configura todos os pinos para zero
void init_24g(void)
{
  delayms(10);
  CE=0;
  CS=0;
  CLK1=0;
  DATA=0;
  delayms(10);
}

//Rotina para escrever no módulo TRW24G em 8 bit
void write_24g(unsigned char dado)
{
  unsigned char i;
  int out=0;
  CLK1=0;
  for(i=0;i<8;i++)
  {
    out=dado&0x80;
    if (out)
      DATA=1;
    else
      DATA=0;
    delayus();  //Ts - Setup Time 100us
    CLK1=1;
    delayus();  //Th - Hold Time 100us
    CLK1=0;
    dado<<=1;
  }
}

//Rotina para escrever no módulo TRW24G em 12 bit
void write_24g_12(unsigned int dado)
{
```

```
  unsigned char i;
  int out=0;
  CLK1=0;
  for(i=0;i<12;i++)
  {
    out=dado&0x800;
    if (out)
      DATA=1;
    else
      DATA=0;
    delayus();  //Ts - Setup Time 100us
    CLK1=1;
    delayus();  //Th - Hold Time 100us
    CLK1=0;
    dado<<=1;
  }
}

//Rotina para escrever no módulo TRW24G em 60 bit
void write_24g_60(void)
{
  unsigned char i,j;
  int out=0;
  CLK1=0;
  for (j=0;j<5;j++)
  {
  for(i=0;i<12;i++)
  {
    out=resultados[j]&0x800;
    if (out)
      DATA=1;
    else
      DATA=0;
    delayus();  //Ts - Setup Time 100us
    CLK1=1;
    delayus();  //Th - Hold Time 100us
    CLK1=0;
```

```
    resultados[j]<<=1;
  }
  }
}

//Rotina de Configuração do Módulo TRW24G, 8 bit
void configura_24g(unsigned char mode)
{
  P4DIR |= 0X1D; // define P4 como saida
  //delayms(1);
  CE=0;
  delayus(); //Td - delay between edges
  CS=1;
  delayus(); //Tcs2data - delay from CS to data
  write_24g(0x8E);
  write_24g(0x08);
  write_24g(0x1C);
  write_24g(0x08);//8 bit
  write_24g(0x08);//8 bit
  write_24g(0xC0);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0xA3);
  write_24g(0x6F);
  if (mode==1)
    write_24g(0x14);
  if (mode==0)
    write_24g(0x15);
  delayus();
  CS=0;
```

```
}

//Rotina de Configuração do Módulo TRW24G, 12 bit
void configura_24g_12(unsigned char mode)
{
  P4DIR |= 0X1D; // define P4 como saida
  delayms(1);
  CE=0;
  delayus(); //Td - delay between edges
  CS=1;
  delayus(); //Tcs2data - delay from CS to data
  write_24g(0x8E);
  write_24g(0x08);
  write_24g(0x1C);
  write_24g(0x0C); //12 bit
  write_24g(0x0C); //12 bit
  write_24g(0xC0);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0xA3);
  write_24g(0x6F);
  if (mode==1)
    write_24g(0x14);
  if (mode==0)
    write_24g(0x15);
  delayus();
  CS=0;
}
```

```
//Rotina de Configuração do Módulo TRW24G, 60 bit
void configura_24g_60(unsigned char mode)
{
  P4DIR |= 0X1D; // define P4 como saida
  //delayms(1);
  CE=0;
  delayus(); //Td - delay between edges
  CS=1;
  delayus(); //Tcs2data - delay from CS to data
  write_24g(0x8E);
  write_24g(0x08);
  write_24g(0x1C);
  write_24g(0x3C); //60 bit
  write_24g(0x3C); //60 bit
  write_24g(0xC0);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0xA3);
  write_24g(0x6F);
  if (mode==1)
    write_24g(0x14);
  if (mode==0)
    write_24g(0x15);
  delayus();
  CS=0;
}
```

```
//rotina envia dados para o modulo TRW24G, 8 bit
void send_24g(unsigned char dado)
{
  delayus();
  CS=0;
  delayus();
  CE=1;
  delayus();
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(dado);
  delayus();
  CE=0;
  delayus();
  delayus();
}

//rotina envia dados para o modulo TRW24G, 12 bit
void send_24g_12(unsigned int dado)
{
  delayus();
  CS=0;
  delayus();
  CE=1;
  delayus();
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g_12(dado);
  delayus();
```

```
  CE=0;
  delayus();
  delayus();
}

//rotina envia dados para o modulo TRW24G, 12 bit
void send_24g_60(void)
{
  delayus();
  CS=0;
  delayus();
  CE=1;
  delayus();
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g_60();
  delayus();
  CE=0;
  delayus();
  delayus();
}

//rotina que le os dados enviados para o modulo TRW24G, 8 bit
unsigned char read_24g(void)
{
 unsigned char i, temp=0;
 delayus();
 CLK1=0;
 delayus();
 CLK1=1;
 for(i=0;i<8;i++)
 {
   temp<<=1;
   CLK1=1;
```

```
   delayus();
   if (DATAIN==1)
     temp=temp+0x01;
   else
     temp=temp+0x00;
   CLK1=0;
   delayus();
 }
 return(temp);
}

//rotina que le os dados enviados para o modulo TRW24G, 12 bit
unsigned int read_24g_12(void)
{
 unsigned int i, temp=0;
 delayus();
 CLK1=0;
 delayus();
 CLK1=1;
 for(i=0;i<12;i++)
 {
   temp<<=1;
   CLK1=1;
   delayus();
   if (DATAIN==1)
     temp=temp+0x01;
   else
     temp=temp+0x00;
   CLK1=0;
   delayus();
 }
 return(temp);
}

void delayus(void)//Aproximadamente 100us (ok)
   {
       unsigned int i;
```

```
        i = 1;
         do (i--);
         while (i != 0);
  }
void delayms(unsigned int j) //se j=1 aproxemadamente 1ms
  {
      unsigned int i;
        i = 170*j;
         do (i--);
         while (i != 0);
  }

#pragma vector=ADC12_VECTOR
__interrupt void ADC12ISR (void)
{
  static unsigned int index = 0;

  A0results[index] = ADC12MEM0;           // Move A0 results, IFG is cleared
  A2results[index] = ADC12MEM2;           // Move A2 results, IFG is cleared
  A1results[index] = ADC12MEM1;           // Move A1 results, IFG is cleared

  A3results[index] = ADC12MEM3;           // Move A3 results, IFG is cleared
  A4results[index] = ADC12MEM4;           // Move A4 results, IFG is cleared
  index = (index+1)%Num_of_Results;         // Increment results index, modulo; Set
Breakpoint here

resultados[0]=(A0results[0]+A0results[1]+A0results[2]+A0results[3]+A0results[4]+A0
results[5]+A0results[6]+A0results[7])/8;//Filtro digital

resultados[2]=(A2results[0]+A2results[1]+A2results[2]+A2results[3]+A2results[4]+A2
results[5]+A2results[6]+A2results[7])/8;//Filtro digital

resultados[1]=(A1results[0]+A1results[1]+A1results[2]+A1results[3]+A1results[4]+A1
results[5]+A1results[6]+A1results[7])/8;//Filtro digital

resultados[3]=(A3results[0]+A3results[1]+A3results[2]+A3results[3]+A3results[4]+A3
results[5]+A3results[6]+A3results[7])/8;//Filtro digital

resultados[4]=(A4results[0]+A4results[1]+A4results[2]+A4results[3]+A4results[4]+A4
results[5]+A4results[6]+A4results[7])/8;//Filtro digital
  teste();
```

```
}
void teste(void)
{
  configura_24g(0);
  P4DIR = 0X0D;
  CS=0;
  delayus();
  CE=1;
  delayus();
while (DR1==0)
{
 conta++;
if (conta>20000)
P1OUT = 0X00;
}
conta=0;
if (DR1==1)
{
leitura=read_24g();
if (leitura >=66)
{
switch(leitura)
{
case 0x66:
  P1OUT = 0X07;
break;
case 0x77:
  P1OUT = 0X3B;
break;
case 0x88:
  P1OUT = 0X00;
break;
case 0x99:
  P1OUT = 0X06;
break;
case 0xAA:
  P1OUT = 0X3A;
```

```
break;
}
}
}
if (leitura==0x11)
{
configura_24g_60(1);
send_24g_60();
}
}
```

APÊNDICE B– ROTINAS DO SOFTWARE MSP430F1232

```
#include  <io430x12x.h>
#include  <intrinsics.h>
#define CE P1OUT_bit.P1OUT_0
#define CS P1OUT_bit.P1OUT_2
#define CLK1 P1OUT_bit.P1OUT_3
#define DATA P1OUT_bit.P1OUT_4
#define DATAIN P1IN_bit.P1IN_4
#define DR1 P1IN_bit.P1IN_5
void delayus(void);
void delayms(unsigned int j);
void init_24g(void);
void write_24g(unsigned char dado);;
void write_24g_12(unsigned int dado);
void configura_24g(unsigned char mode);
void configura_24g_12(unsigned char mode);
void send_24g(unsigned char dado);
void send_24g_12(unsigned int dado);
unsigned char read_24g(void);
unsigned int read_24g_12(void);
void read_24g_12_st(void);
void configura_24g_60(unsigned char mode);
void read_24g_60_st(void);
void envia_serial(unsigned int dado);
void recebe_serial(void);
void compara(void);
unsigned char recebe;
static char string1[]={"                                                \r\n"};
```

```
char i,teste,j,k;
unsigned int leitura,conta=10,tempo;

void main(void)
{
  WDTCTL = WDTPW + WDTCNTCL;
  volatile unsigned int i;
  P3SEL |= 0x30;                        // P3.4,5 = USART0 TXD/RXD
  BCSCTL1 |= XTS;                       // ACLK= LFXT1= HF XTAL
  do
  {
  IFG1 &= ~OFIFG;                       // Clear OSCFault flag
  for (i = 0xFF; i > 0; i--);           // Time for flag to set
  }
  while ((IFG1 & OFIFG));               // OSCFault flag still set?
  BCSCTL2 |= SELM_3;                    // MCLK = LFXT1 (safe)
  ME2 |= UTXE0 + URXE0;                 // Enable USART0 TXD/RXD
  UCTL0 |= CHAR;                        // 8-bit character
  UTCTL0 |= SSEL0;                      // UCLK= ACLK
  UBR00 = 0x45;
  UBR10 = 0x00;
  UMCTL0 = 0x2c;
  UCTL0 &= ~SWRST;                      // Initialize USART state machine
  IE2 = URXIE0 + UTXIE0;                // Enable USART0 RX/TX interrupt
  P2DIR |= 0XFF;
  P2OUT = 0;
  P1DIR = 0X1D;
  init_24g();
  delayms(3);
  configura_24g(1);
  P1DIR = 0X1D;
  CS=0;
  delayus();
  CE=1;
  delayus();
    __bis_SR_register(LPM0_bits + GIE);             // Enter LPM3 w/ interrupt
while(1)
```

```
{
 IE2 = URXIE0;
}
}
//Configura todos os pinos para zero
void init_24g(void)
{
  delayms(10);
  CE=0;
  CS=0;
  CLK1=0;
  DATA=0;
  delayms(10);

}

//Rotina para escrever no módulo TRW24G em 8 bit
void write_24g(unsigned char dado)
{
  unsigned char i;
  int out=0;
  CLK1=0;
  for(i=0;i<8;i++)
  {
    out=dado&0x80;
    if (out)
      DATA=1;
    else
      DATA=0;
    delayus();  //Ts - Setup Time 100us
    CLK1=1;
    delayus();  //Th - Hold Time 100us
    CLK1=0;
    dado<<=1;
  }
}
```

```
//Rotina para escrever no módulo TRW24G em 12 bit
void write_24g_12(unsigned int dado)
{
  unsigned char i;
  int out=0;
  CLK1=0;
  for(i=0;i<12;i++)
  {
    out=dado&0x800;
    if (out)
      DATA=1;
    else
      DATA=0;
    delayus();  //Ts - Setup Time 100us
    CLK1=1;
    delayus();  //Th - Hold Time 100us
    CLK1=0;
    dado<<=1;
  }
}

//Rotina de Configuração do Módulo TRW24G, 8 bit
void configura_24g(unsigned char mode)
{
  P1DIR |= 0X1D; // define P1 como saida
  //delayms(1);
  CE=0;
  delayus(); //Td - delay between edges
  CS=1;
  delayus(); //Tcs2data - delay from CS to data
  write_24g(0x8E);
  write_24g(0x08);
  write_24g(0x1C);
  write_24g(0x08);//8 bit
  write_24g(0x08);//8 bit
  write_24g(0xC0);
  write_24g(0xAA);
```

```
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0xA3);
  write_24g(0x6F);
  if (mode==1)
    write_24g(0x14);
  if (mode==0)
    write_24g(0x15);
  delayus();
  CS=0;
}

//Rotina de Configuração do Módulo TRW24G, 12 bit
void configura_24g_12(unsigned char mode)
{
  P1DIR |= 0X1D; // define P1 como saida
  delayms(1);
  CE=0;
  delayus(); //Td - delay between edges
  CS=1;
  delayus(); //Tcs2data - delay from CS to data
  write_24g(0x8E);
  write_24g(0x08);
  write_24g(0x1C);
  write_24g(0x0C); //12 bit
  write_24g(0x0C); //12 bit
  write_24g(0xC0);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
```

```
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0xA3);
  write_24g(0x6F);
  if (mode==1)
    write_24g(0x14);
  if (mode==0)
    write_24g(0x15);
  delayus();
  CS=0;
}
//Rotina de Configuração do Módulo TRW24G, 60 bit
void configura_24g_60(unsigned char mode)
{
  P1DIR |= 0X1D; // define P4 como saida
  //delayms(1);
  CE=0;
  delayus(); //Td - delay between edges
  CS=1;
  delayus(); //Tcs2data - delay from CS to data
  write_24g(0x8E);
  write_24g(0x08);
  write_24g(0x1C);
  write_24g(0x3C); //60 bit
  write_24g(0x3C); //60 bit
  write_24g(0xC0);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
```

```
  write_24g(0xAA);
  write_24g(0xA3);
  write_24g(0x6F);
  if (mode==1)
    write_24g(0x14);
  if (mode==0)
    write_24g(0x15);
  delayus();
  CS=0;
}

//rotina envia dados para o modulo TRW24G, 8 bit
void send_24g(unsigned char dado)
{
  delayus();
  CS=0;
  delayus();
  CE=1;
  delayus();
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(dado);
  delayus();
  CE=0;
  delayus();
  delayus();
}

//rotina envia dados para o modulo TRW24G, 12 bit
void send_24g_12(unsigned int dado)
{
  delayus();
  CS=0;
  delayus();
```

```
  CE=1;
  delayus();
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g(0x55);
  write_24g(0xAA);
  write_24g_12(dado);
  delayus();
  CE=0;
  delayus();
  delayus();
}

//rotina que le os dados enviados para o modulo TRW24G, 8 bit
unsigned char read_24g(void)
{
 unsigned char i, temp=0;
 delayus();
 CLK1=0;
 delayus();
 CLK1=1;
 for(i=0;i<8;i++)
 {
   temp<<=1;
   CLK1=1;
   delayus();
   if (DATAIN==1)
     temp=temp+0x01;
   else
     temp=temp+0x00;
   CLK1=0;
   delayus();
 }
 return(temp);
}
```

```
//rotina que le os dados enviados para o modulo TRW24G, 12 bit
unsigned int read_24g_12(void)
{
 unsigned int i, temp=0;
 delayus();
 CLK1=0;
 delayus();
 CLK1=1;
 for(i=0;i<12;i++)
 {
   temp<<=1;
   CLK1=1;
   delayus();
   if (DATAIN==1)
     temp=temp+0x01;
   else
     temp=temp+0x00;
   CLK1=0;
   delayus();
 }
 return(temp);
}

//rotina que le os dados enviados para o modulo TRW24G, 12 bit - string
void read_24g_12_st(void)
{
 unsigned int i;
 string1[0]='4';
  string1[1]='7';
  string1[2]='3';
  string1[3]='1';
  string1[16]='9';
 delayus();
 CLK1=0;
 delayus();
 CLK1=1;
 for(i=4;i<16;i++)
```

```
 {
   CLK1=1;
   delayus();
   if (DATAIN==1)
     string1[i]='1';
   else
     string1[i]='0';
   CLK1=0;
   delayus();
 }
}

//rotina que le os dados enviados para o modulo TRW24G, 60it - string
void read_24g_60_st(void)
{
 unsigned int i;
 string1[0]='4';
  string1[1]='7';
  string1[2]='3';
  string1[3]='1';
  string1[64]='9';
 delayus();
 CLK1=0;
 delayus();
 CLK1=1;
 for(i=4;i<64;i++)
 {
   CLK1=1;
   delayus();
   if (DATAIN==1)
     string1[i]='1';
   else
     string1[i]='0';
   CLK1=0;
   delayus();
 }
}
```

```
void delayus(void)
  {
      unsigned int i;
       i = 1;
        do (i--);
        while (i != 0);
  }
void delayms(unsigned int j) //se j=1 aproxemadamente 1ms
  {
      unsigned int i;
       i = 170*j;
        do (i--);
        while (i != 0);

  }

// UART0 TX ISR
#pragma vector=USART0TX_VECTOR
__interrupt void usart0_tx (void)
{
  if (i < sizeof string1-1)
     TXBUF0 = string1[i++];
 else
  IE2 = URXIE0;
  }

// UART0 RX ISR
#pragma vector=USART0RX_VECTOR
__interrupt void usart0_rx (void)
{

  recebe = RXBUF0;
  compara();

}
```

```
void envia_serial(unsigned int dado)
{
unsigned int j=0;
  int out=0;
    string1[0]='4';
  string1[1]='7';
  string1[2]='3';
  string1[3]='1';
  string1[16]='9';
    for (j=4;j<16;j++)
    {
     out=dado&0x800;
     if (out==0x800)
       string1[j]='1';

     else
       string1[j]='0';
    dado<<=1;
    }
    i=0;
     TXBUF0 = string1[i++];

}
void compara(void)
{
 WDTCTL = WDTPW + WDTCNTCL;
if ((recebe=='a')|(recebe=='z')|(recebe=='f')|(recebe=='h')|(recebe=='w')|(recebe=='u'))
IE2 = UTXIE0;

 switch(recebe)
{
case 'a':
  {
    configura_24g(1);
  P1DIR = 0X1D;
  CS=0;
  delayus();
```

```
  CE=1;
  delayus();
  send_24g(0x88);
  IE2 = URXIE0;
  }
break;
  case 'z':
    {
  configura_24g(1);
  P1DIR = 0X1D;
  CS=0;
  delayus();
  CE=1;
  delayus();
  send_24g(0x66);
 IE2 = URXIE0;
  }
 break;
  case 'f':
    {
  configura_24g(1);
  P1DIR = 0X1D;
  CS=0;
  delayus();
  CE=1;
  delayus();
  send_24g(0x77);
 IE2 = URXIE0;
  }
  break;
  case 'h':
    {
  configura_24g(1);
  P1DIR = 0X1D;
  CS=0;
  delayus();
  CE=1;
```

```
  delayus();
  send_24g(0x99);
 IE2 = URXIE0;
  }
  break;
  case 'w':
   {
  configura_24g(1);
  P1DIR = 0X1D;
  CS=0;
  delayus();
  CE=1;
  delayus();
  send_24g(0xAA);
  IE2 = URXIE0;
    }
   break;
  case 'u':
  {
IE2 = UTXIE0;
  configura_24g(1);
  P1DIR = 0X1D;
  CS=0;
  delayus();
  CE=1;
  delayus();
  send_24g(0x11);
  configura_24g_60(0);
  P1DIR =0X0D;
  CS=0;
  delayus();
  CE=1;
  delayus();
   while (DR1==0)
  {
  }
```

```
  if (DR1==1)
  {
    read_24g_60_st();
   i=0;
     TXBUF0 = string1[i++];
  }
  }

}
}
```

APÊNDICE C – ROTINA DO SOFTWARE BUILDER

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <time.h>
#include <stdio.h>
#include <dos.h>
#pragma hdrstop
#include "UPrincipal.h"
#include "Serial.h"
#include <math.h>
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TfrmPrincipal *frmPrincipal;
AnsiString sPorta;
String mensagem_serial;
float cont=0;
int cond=1,tentativa=0,tempo=0,r=0,s=0;
unsigned int a,b,c,d,e,i,j,k,m,dado[12],conta=0;
float valor;
  static char buffer[TAM_MAX_BUFFER];
BYTE aux;
clock_t start, end;

//---------------------------------------------------------------------------
__fastcall TfrmPrincipal::TfrmPrincipal(TComponent* Owner)
  : TForm(Owner)
{
```

```
}
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::FormCreate(TObject *Sender)
{
   // Inicializa porta serial padrão: COM1.
   rdgPortaSerial->ItemIndex = 0;

}
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::btnInicializaSerialClick(TObject *Sender)
{
  if (btnInicializaSerial->Caption == "Inicializa")
  {
   if (bInicializaSerial(sPorta.c_str())==ERRO_ABERTURA_PORTA_COMUNICACAO)
       ShowMessage("Erro na abertura da porta");
   else
   {
      btnEnvia->Enabled = true;
      rdgPortaSerial->Enabled = false;
      btnInicializaSerial->Caption = "Encerra";
    }
   }
     else
     {
       bEncerraSerial();
       btnEnvia->Enabled = false;
      rdgPortaSerial->Enabled = true;
       btnInicializaSerial->Caption = "Inicializa";
       }
}
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::rdgPortaSerialClick(TObject *Sender)
{
   sPorta = rdgPortaSerial->Items->Strings[rdgPortaSerial->ItemIndex];
```

```
}
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::btnEnviaClick(TObject *Sender)
{
Button2->Enabled = false;
Timer2->Enabled = false;
//RPM motor - aquecimento
btnEnvia->Enabled = false;
Button1->Enabled = true;
rpm_motor->State=cbChecked;
chave_15->State=cbChecked;
chave_19->State=cbChecked;
//----------------------------------------
edtMensagemTransmitir->Text="z";
bTransmiteMensagem(edtMensagemTransmitir->Text.c_str());
//----------------------------------------
Timer1->Enabled = true;
start = clock();
conta=0;

}
//---------------------------------------------------------------------------


void __fastcall TfrmPrincipal::Timer1Timer(TObject *Sender)
{

end = clock();
cont=(end-start)*0.001;
//RPM motor - partindo

if (cont<0.05)
{
//----------------------------------------
edtMensagemTransmitir->Text="z";
bTransmiteMensagem(edtMensagemTransmitir->Text.c_str());
```

```
//---------------------------------------

}
if (cont>10&cont<10.05)

{
chave_15->State=cbChecked;
chave_19->State=cbUnchecked;
chave_50->State=cbChecked;
acelerador->State=cbChecked;



//---------------------------------------
edtMensagemTransmitir->Text="f";
bTransmiteMensagem(edtMensagemTransmitir->Text.c_str());
//---------------------------------------

}
 // pausa
if (cont>20&cont<20.05)
//if (cont>12&cont<12.05) //Moto
{
bba_inj->State=cbChecked;
chave_15->State=cbUnchecked;
chave_19->State=cbUnchecked;
chave_50->State=cbUnchecked;
acelerador->State=cbUnchecked;
cond=3;
tentativa=1;
tempo=30;
//---------------------------------------
edtMensagemTransmitir->Text="a";
bTransmiteMensagem(edtMensagemTransmitir->Text.c_str());
//---------------------------------------
}
if (cont>30&tentativa<5)
{
```

```
if (cont>tempo&cond==3)
{
tentativa1->State=cbChecked;
if (tentativa==2)
tentativa2->State=cbChecked;
if (tentativa==3)
tentativa3->State=cbChecked;
if (tentativa==4)
tentativa4->State=cbChecked;
chave_15->State=cbChecked;
chave_19->State=cbChecked;
r++;
if (r>10)
{
cond=4;
tempo=tempo+10;

r=0;
}
//----------------------------------------
edtMensagemTransmitir->Text="h";
bTransmiteMensagem(edtMensagemTransmitir->Text.c_str());
//----------------------------------------

}
if (cont>tempo&cond==4)

{
chave_15->State=cbChecked;
chave_19->State=cbUnchecked;
chave_50->State=cbChecked;
acelerador->State=cbChecked;
 r++;
if (r>10)
{
cond=5;
```

```
  tempo=tempo+15;
  //tempo=tempo+2;//Moto
  r=0;
  }
  //----------------------------------------
  edtMensagemTransmitir->Text="w";
  bTransmiteMensagem(edtMensagemTransmitir->Text.c_str());
  //----------------------------------------
  }
    if (cont>tempo&cond==5)  // pausa

  {
  bba_inj->State=cbChecked;
  chave_15->State=cbUnchecked;
  chave_19->State=cbUnchecked;
  chave_50->State=cbUnchecked;
  acelerador->State=cbUnchecked;

   r++;
  if (r>10)
  {
  cond=3;
  tempo=cont+10;
  tentativa++;
  r=0;
  }
  //----------------------------------------
  edtMensagemTransmitir->Text="a";
  bTransmiteMensagem(edtMensagemTransmitir->Text.c_str());
  //----------------------------------------
  }

}
     edtMensagemTransmitir->Text = "u";
     bTransmiteMensagem(edtMensagemTransmitir->Text.c_str());
      Sleep(15);
   aux = bRecebeMensagem(buffer);
```

```
  if (aux == SS_SERVICO_EM_ANDAMENTO)
  {
     mensagem_serial=((AnsiString)buffer);
     String InicioStr = mensagem_serial.SubString(1, 3);
     String FimStr=mensagem_serial.SubString(65,1);
     if ((InicioStr=="473")&(FimStr=="9"))
     {
     conta=0;
      e=0;
      j=5;
      k=11;
      for(i=0;i<12;i++)
      {
      dado[i]=StrToInt(mensagem_serial.SubString(j,1));
      a=dado[i]*pow(2,k)+e;
      e=a;
      j++;
      k--;
      }
      valor=(((3.48*a)/4096)+0.0027)/0.2322;
        end = clock();
cont=(end-start)*0.001;
Series1->AddXY(cont,valor,"",clRed);
     }

     if ((InicioStr=="473")&(FimStr=="9"))
     {
      e=0;
      k=11;
      for(i=0;i<12;i++)
      {
      dado[i]=StrToInt(mensagem_serial.SubString(j,1));
      a=dado[i]*pow(2,k)+e;
      e=a;
      j++;
      k--;
      }
```

```
      valor=(((3.48*a)/4096)+0.0017)/0.2321;
       end = clock();
cont=(end-start)*0.001;
Series2->AddXY(cont,valor,"",clYellow);
      }

     if ((InicioStr=="473")&(FimStr=="9"))
     {
      e=0;
      k=11;
      for(i=0;i<12;i++)
      {
      dado[i]=StrToInt(mensagem_serial.SubString(j,1));
      a=dado[i]*pow(2,k)+e;
      e=a;
      j++;
      k--;
      }
      valor=(((3.48*a)/4096)+0.0001)/0.2365;
        end = clock();
cont=(end-start)*0.001;
Series3->AddXY(cont,valor,"",clBlue);
     }

     if ((InicioStr=="473")&(FimStr=="9"))
     {
      e=0;
      k=11;
      for(i=0;i<12;i++)
      {
      dado[i]=StrToInt(mensagem_serial.SubString(j,1));
      a=dado[i]*pow(2,k)+e;
      e=a;
      j++;
      k--;
      }
      valor=(((3.48*a)/4096)-0.004)*197.34;
```

```
       end = clock();
cont=(end-start)*0.001;
LineSeries2->AddXY(cont,valor,"",clRed);
      }

      if ((InicioStr=="473")&(FimStr=="9"))
      {
       e=0;

       k=11;
       for(i=0;i<12;i++)
       {
       dado[i]=StrToInt(mensagem_serial.SubString(j,1));
       a=dado[i]*pow(2,k)+e;
       e=a;
       j++;
       k--;
       }
       valor=60/(12/(((3.48*a)/4096)/0.0057));
        end = clock();
cont=(end-start)*0.001;
Series4->AddXY(cont,valor,"",clGreen);

      }
   }
 conta++;
 if (conta>50)
 {
   Timer1->Enabled = false;
    ShowMessage("Erro na comunicação");

}

Label2->Caption=cont;

}
```

```
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::Button1Click(TObject *Sender)
{

Timer1->Enabled = false;
Timer2->Enabled = true;
btnEnvia->Enabled = true;
Button1->Enabled = false;
bba_inj->State=cbChecked;
chave_15->State=cbUnchecked;
chave_19->State=cbUnchecked;
chave_50->State=cbUnchecked;
acelerador->State=cbUnchecked;
Button2->Enabled = true;
}
//---------------------------------------------------------------------------

void __fastcall TfrmPrincipal::Timer2Timer(TObject *Sender)
{

//---------------------------------------
  edtMensagemTransmitir->Text="a";
  bTransmiteMensagem(edtMensagemTransmitir->Text.c_str());
  //---------------------------------------
}
//---------------------------------------------------------------------------
void __fastcall TfrmPrincipal::Button2Click(TObject *Sender)
{
Series1->Clear();
Series2->Clear();
Series3->Clear();
LineSeries2->Clear();
Series4->Clear();
bba_inj->State=cbUnchecked;
rpm_motor->State=cbUnchecked;
tentativa1->State=cbUnchecked;
```

```
tentativa2->State=cbUnchecked;
tentativa3->State=cbUnchecked;
tentativa4->State=cbUnchecked;

}
```

ANEXO B – TEMPORIZAÇÃO – MÓDULO TRW-24G

Ao longo deste projeto são citadas as temporizações a serem seguidas e que são descritas abaixo: (Wenshing, 2008)

- Tpd2cfgm – Tempo entre ligar o Módulo TRW-24G e o início do modo de configuração. Este tempo também deve ser respeitado quando o módulo está no modo *standby* e a configuração será iniciada, conforme figura 5-1. Tempo máximo de 3ms.

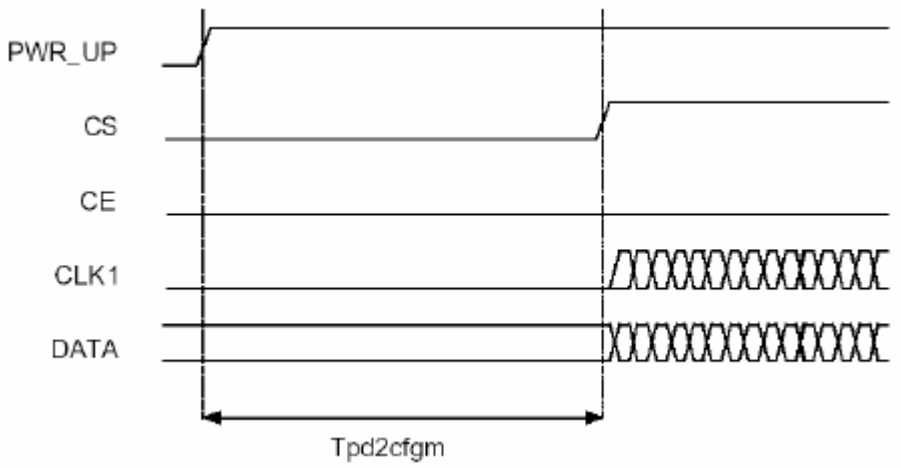

Figura 5-1 - Diagrama do Tpd2cfgm.

- Tpd2a – Tempo entre ligar o Módulo TRW-24G e o início do modo ativo, a monitoração do ar, conforme figura 5-2. Tempo máximo 3ms.

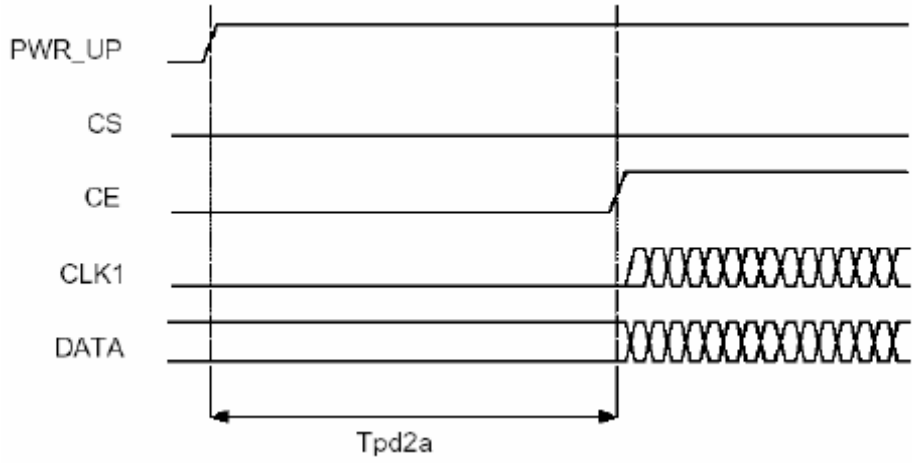

Figura 5-2 - Diagrama do Tpd2a.

- Tsby2txSB – Tempo que o Módulo TRW-24G leva para transmitir os dados, no modo *ShockBurst*, após o pino CE ser levado de nível lógico 1 para nível lógico 0, conforme figura 5-5. Tempo mínimo 195µs.
- Tsby2rx – Tempo que o Módulo TRW-24G leva para começar a monitorar o ar, no modo *ShockBurst*, após o pino CE ser levado de nível lógico 0 para nível lógico 1, conforme figura 5-3. Tempo mínimo 202µs.

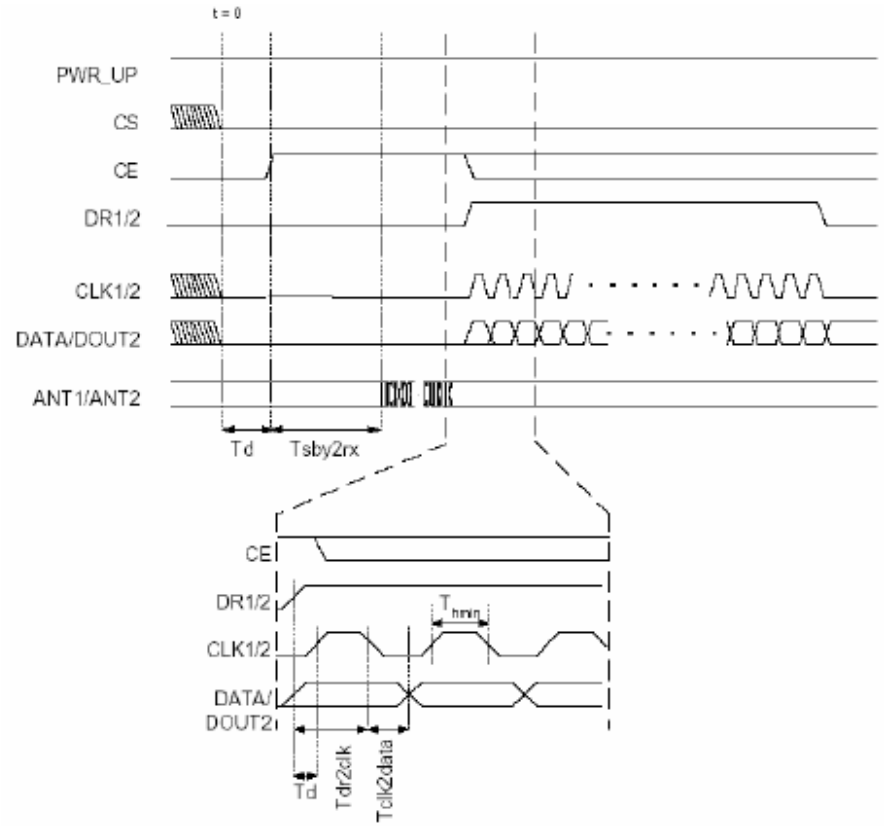

Figura 5-3 - Diagrama dos Tempos de Recepção no Modo *ShockBurst*.

- Tcs2data – Tempo de *delay* antes que a palavra de configuração seja enviada ao Módulo TRW-24G após o pino CS ser levado de nível lógico 0 para nível lógico 1, conforme figura 5-4. Tempo mínimo 5µs.

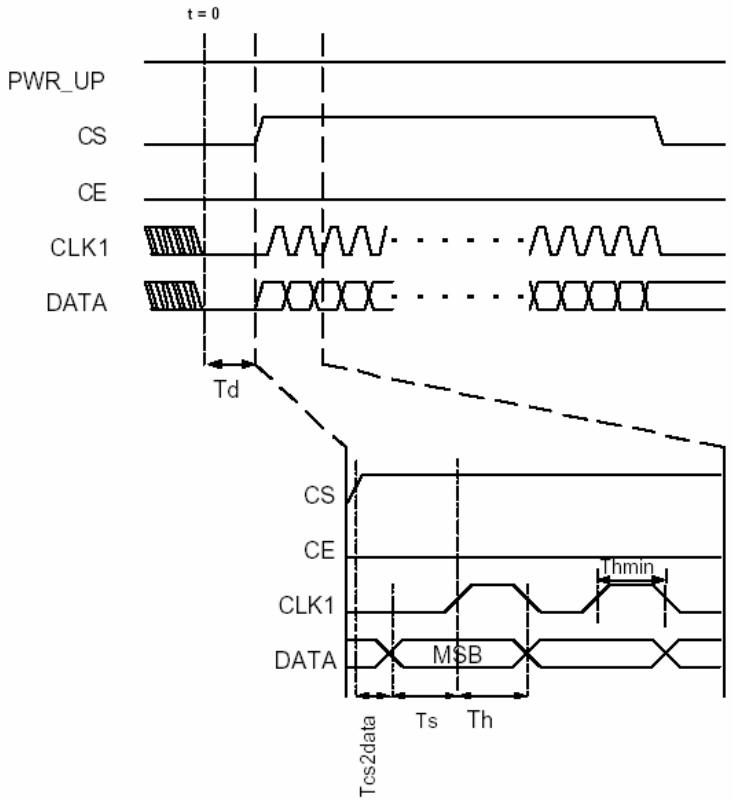

Figura 5-4 - Diagrama de Tempos do Modo de Configuração.

- Tce2data – Tempo de *delay* antes que os dados sejam enviados ao Módulo TRW-24G após o pino CE ser levado de nível lógico 0 para nível lógico 1, conforme figura 5-5. Tempo mínimo 5µs.

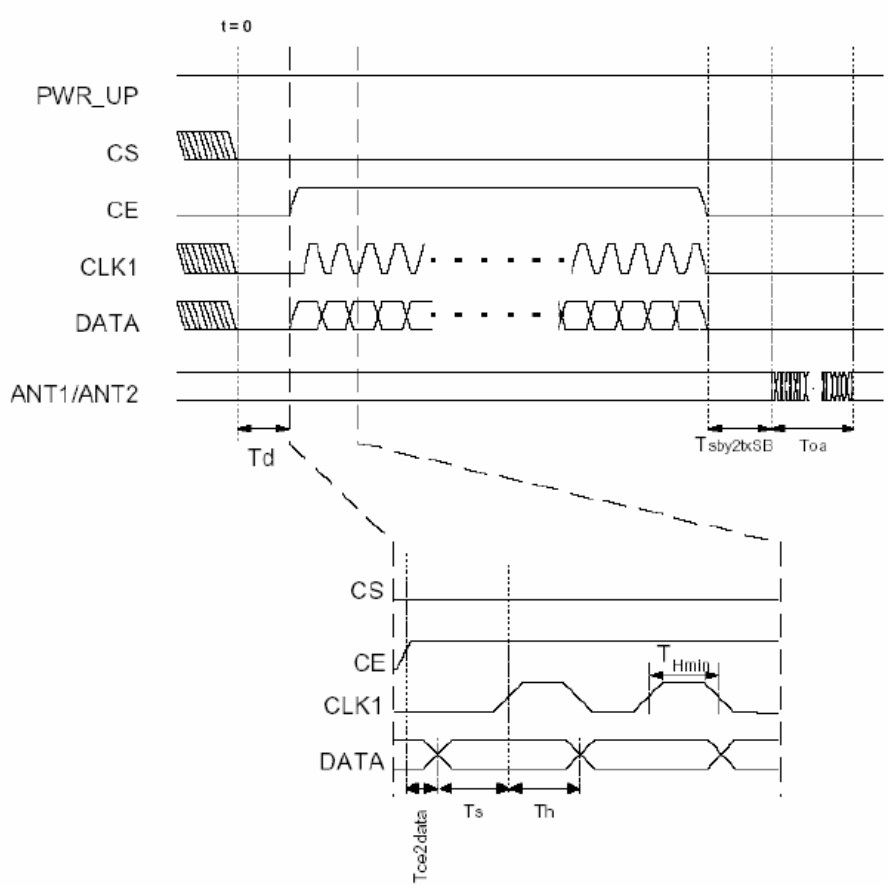

Figura 5-5 - Diagrama dos Tempos de Transmissão no Modo *ShockBurs*t.

- Tdr2clk – Tempo que o pino CLK, do Módulo TRW-24G, deve permanecer em nível lógico 1 durante a recepção de cada bit, conforme figura 5-3. Tempo mínimo 50ns.
- Tclk2data – Tempo que o pino CLK, do Módulo TRW-24G, deve permanecer em nível lógico 0 durante a recepção de cada bit, conforme figura 5-3. Tempo mínimo 50ns.
- Td – Tempo entre o pino CS ser levado de nível lógico 0 para nível lógico 1, do Módulo TRW-24G, com o pino CE estando em nível lógico 0, conforme figura 5-5. Os pinos CE e CS nunca podem ser levados para nível lógico 1 ao mesmo tempo. Tempo mínimo 50ns.
- Ts – Tempo que o pino CLK, do Módulo TRW-24G, deve permanecer em nível lógico 0 durante o envio do bit mais significativo ao módulo, conforme figura 5-5. Tempo mínimo 500ns.
- Th – Tempo que o pino CLK, do Módulo TRW-24G, deve permanecer em nível lógico 1 durante o envio do bit mais significativo ao módulo, conforme figura 5-5. Tempo mínimo 500ns.

- Thmin – Tempo que o pino CLK, do Módulo TRW-24G, deve permanecer em nível lógico 1 durante o envio de um bit ao módulo, conforme figura 5-5. Tempo mínimo 500ns.