FELIPE CAMPOS DE FREITAS

CONFIGURAÇÃO E GERENCIAMENTO DO KIT EKI LM3S8962 PARA ACIONAMENTO E CONTROLE DE MÁQUINAS ELÉTRICAS

São Carlos
2012

FELIPE CAMPOS DE FREITAS

CONFIGURAÇÃO E GERENCIAMENTO DO KIT EKI LM3S8962 PARA ACIONAMENTO E CONTROLE DE MÁQUINAS ELÉTRICAS

Trabalho de Conclusão de Curso
apresentado à Escola de Engenharia de São
Carlos, da Universidade de São Paulo

Curso de Engenharia Elétrica com
ênfase em eletrônica

ORIENTADOR: Manoel Luís Aguiar

São Carlos
2012

Freitas, Felipe Campos de
F862c Configuração e gerenciamento do kit EKI LM3S8962 para acionamento e controle de máquinas elétricas / Felipe Campos de Freitas; orientador Manoel Luís de Aguiar. São Carlos, 2012.

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

1. Máquinas elétricas. 2. EKI LM3S8962. 3. ARM Cortex-M3. 4. Acionamento de motores . 5. Aquisição de dados. I. Título.

FOLHA DE APROVAÇÃO

Nome: Felipe Campos de Freitas

Título: "Configuração e gerenciamento do kit EKI LM3S08962 para acionamento e controle de máquinas elétricas"

Trabalho de Conclusão de Curso defendido e aprovado em 10 / 12 / 2012,

com NOTA 6,5 (seis, cinco), *pela Comissão Julgadora:*

Prof. Dr. Manoel Luís de Aguiar (Orientador)
SEL/EESC/USP

Prof. Dr. Azauri Albano de Oliveira Júnior
SEL/EESC/USP

M.Sc. Eduardo Sylvestre Lopes de Oliveira
SEL/EESC/USP

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

Dedicatória:

Aos meus pais pelo apoio
incondicional e em especial ao meu
avô, o Professor Milton de Freitas
pelo incentivo a esta carreira.

Agradecimentos

Aos meus pais, Amauri e Mônica por deixar alguns sonhos de lado em prol da minha formação.

À minha irmã Juliana pelo apoio nos momentos difíceis.

Ao professor Manoel Aguiar pelo conhecimento a mim transmitido e pela paciência e tempo gasto na execução do presente trabalho.

Aos amigos Pedro e Ricardo pela amizade e pela força durante o ano executando este trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

O presente trabalho estuda a utilização do *kit* de desenvolvimento EKI LM3S8962 da *Texas Instruments,* para acionamento e controle de máquinas elétricas. Este *kit* de desenvolvimento possui recursos que, configurados corretamente, podem ser utilizados para este objetivo. Estes recursos, tais como geração de PWM, leitura de *encoders,* conversores analógicos digitais, serão explicados, configurados e testados no decorrer deste trabalho.

Serão apresentados circuitos auxiliares com o objetivo de acoplar o *kit* de acionamento aos circuitos de potência além de circuitos para o tratamento dos sinais de medição.

Testes acionando um motor de corrente contínua e um motor de indução trifásico foram realizados com o objetivo de validação do *kit* de desenvolvimento EKI LM3S8962 para esta aplicação.

Palavras chave: Máquinas elétricas; EKI LM3S8962; ARM Cortex-M3; Acionamento de motores; Aquisição de dados;

Abstract.

This work study the usage of the development *kit* EKI LM3S8962 form Texas instruments, to drive and control of electrical machines. This development *kit* has some features which, proper configuration, could be used to this purpose. This features, like PWM generations, encoder reading, and analogic to digital converter, will be explained, configured and tested during this work.

Will be discussed auxiliary circuits which connect the development *kit* to the power circuits beyond circuits for the measurement signal processing.

Several tests a driving direct current and an alternating current were performed for the validation of the development *kit* EKI LM3S8962 to this application.

Key words: Electrical machines; EKI LM3S8962 ARM Cortex-M3; Drive motors; Data acquisition;

1. Introdução

As máquinas motrizes em geral são muito importantes na sociedade atual desde a revolução industrial, onde o surgimento (principalmente com a energia a vapor) dos processos industriais modificou a sociedade. Com o advento da eletricidade as máquinas elétricas assumiram um papel de protagonista na indústria.

O acionamento destas máquinas pode ser realizado com velocidade variável desde que seja utilizado um circuito e uma lógica de acionamento adequada. Este acionamento com velocidade variável é utilizado tanto para operações que necessitem esta característica quanto para aumentar a eficiência dos motores, reduzindo assim o consumo de energia elétrica. Diversas técnicas ou métodos, cada qual com suas vantagens e desvantagens, podem ser utilizados para acionamento, e controle, de velocidade de um motor elétrico (corrente alternada ou corrente contínua).

O objetivo do trabalho consiste em estudar o *kit* de desenvolvimento EKI LM3S8962 [1], analisando sua utilização para a geração de sinais de acionamento de motores elétricos de corrente contínua e de corrente alternada utilizando, respectivamente, os circuitos *chopper* e inversor de tensão trifásico.

A base do *kit* de desenvolvimento EKI LM3S8962 é o micro controlador *Stellaris ARM Cortex-M3* [2], que possui uma boa capacidade de processamento mesmo sendo de baixo custo [3]. O controlador, em conjunto com os outros recursos do *kit* de desenvolvimento, como conversores analógicos digitais, acionamento via modulação largura de pulso (PWM) e leitores para *encoders*, reúne funções úteis para o acionamento e controle de máquinas elétricas. Outro recurso importante é a possibilidade de comunicação entre o *kit* de desenvolvimento e um *host,* para armazenamento de informações e impressão de resultados.

O presente trabalho explora as características do *kit* de desenvolvimento e seus recursos, realizando ensaios em cada um dos possíveis recursos a serem utilizados para este objetivo, encontrando as características de operação do *kit* desenvolvimento de modo a permitir a utilização do mesmo para controle de máquinas elétricas.

Usualmente as lógicas de controle são realizadas com uma frequência de operação de 1kHz ou 2kHz, ou seja, a cada 1ms ou 500µs. Neste aspecto, visa-se portanto investigar o desempenho do kit em realizar tarefas de aquisição, supervisão, atuação e mais os algoritmos de controle dentro destes intervalos.

A validação do trabalho foi realizada acionando os motores de corrente contínua de imã permanente e de corrente alternada de indução trifásica. Para estes ensaios foram utilizados circuitos auxiliares, de proteção e medição de grandezas físicas, como por exemplo corrente e velocidade de rotação do eixo do motor, para que seja possível o acionamento e a correta análise dos dados. Estes ensaios podem ser expandidos para uma análise sobre a utilização do referente *kit* de desenvolvimento para realização de lógicas de controle para os determinados motores.

A estrutura escrita deste trabalho possui a seguinte organização.

- Capítulo 1 - Introdução: É uma breve introdução ao assunto de acionamento de máquinas elétricas além de apresentar a motivação do presente trabalho.
- Capítulo 2 – Acionamento de motores elétricos em velocidade variável: Serão abordados os diversos métodos de acionamento dos motores com foco nos utilizados durante o presente trabalho.
- Capítulo 3 – Descrição do *kit* de desenvolvimento EKI LM3S8962: Descreve os recursos disponíveis no *kit* de desenvolvimento focando nos recursos candidatos a serem utilizados para o acionamento das máquinas elétricas em questão.
- Capítulo 4 – Programação do *kit* de desenvolvimento EKI LM3S8962: É abordada a configuração dos recursos utilizados do *kit* de desenvolvimento.
- Capítulo 5 – Materiais e métodos utilizados: Aborda a fase prática do trabalho, explicando os circuitos e os materiais utilizados.
- Capítulo 6 – Testes e ensaios realizados: Serão mostrados os resultados obtidos experimentalmente, permitindo a validação do *kit* de desenvolvimento para a aplicação proposta.
- Capítulo 7 – Conclusão: É analisada a proposta de utilização do *kit* de desenvolvimento para o acionamento de máquinas elétricas.

2. Acionamentos de motores elétricos em velocidade variável

Os motores elétricos são dispositivos capazes de transformar energia elétrica em energia mecânica. Os diferentes tipos de motores elétricos são classificados de acordo com a forma de alimentação, às características construtivas e formas de operação [4]. Quando alimentados com tensões nominais, estes motores operam com velocidade constante. Para obter velocidade variável é necessário atuar na forma de acionamento que permite impor torque variável, seja por meio da tensão, da corrente ou frequência de alimentação. O acionamento e controle da velocidade destas máquinas dependem do motor a ser acionado [5] [6].

Neste capítulo serão abordados os circuitos comumente utilizados para o acionamento dos motores de corrente contínua e de motores indução trifásico. Estes dois tipos de acionamento configuram a motivação principal deste trabalho. Neste capítulo serão também esclarecidas as necessidades de recursos para a realização destes acionamentos de forma digital.

Como resultados deste trabalho, pretende-se desenvolver protótipos de *kit*s de demonstração destes acionamentos em disciplinas da graduação e futuramente configurar um experimento de uma possível disciplina de laboratório.

2.1. Acionamento de motores CC

Os diferentes tipos de motores de corrente contínua se diferem basicamente pela forma de excitação. Existem as máquinas de corrente contínua com excitação independente, em série, composta e motores de imã permanente. Em todos os casos a velocidade de rotação do motor é proporcional ao torque, que por sua vez, é proporcional à corrente, e consequentemente à tensão, aplicada ao motor. Os acionamentos mais comuns se baseiam no princípio de controle da tensão média aplicada ao motor [7].

A Equação 1 representa o torque de um motor de corrente contínua.

$$T = K_t * I_f * I_a \tag{1}$$

Onde:

- T = Torque desenvolvido pelo motor
- K_t = Constante de torque

- I_f = Corrente no circuito de campo
- I_a = Corrente no circuito de armadura

Para o caso dos motores de corrente contínua de imã permanente não existe a da corrente de campo (I_f), o controle de velocidade é realizado por meio da corrente de armadura I_a, a qual pode ser imposta diretamente por uma fonte de corrente controlada ou indiretamente por meio do uma fonte de tensão controlada.

Os métodos mais comuns de atuação nos motores de corrente contínua com velocidade variável são por meio de resistor série no circuito de armadura, que atua na corrente efetiva no motor, por meio de retificadores ou *choppers* que podem ser configurados como fontes de tensão ou de corrente controlada [6].

2.1.1. Resistor ou reostato de partida

Tradicionalmente em sistemas de tração antigos (trólebus) [8] eram utilizados resistores para o acionamento de motores de corrente contínua de forma a controlar a corrente aplicada ao motor. Estes resistores eram retirados de acordo com o ganho de velocidade do motor. O fato de o resistor dissipar uma potência relativamente grande torna o sistema ineficiente em relação aos sistemas utilizados atualmente, por esse motivo não é muito usual [6].

2.1.2. Controle a partir de retificadores

Caso a alimentação disponível para o motor de corrente contínua seja uma fonte de corrente alternada, monofásica ou trifásica, é necessária a retificação desta fonte para alimentar o motor de corrente contínua. É possível então controlar a tensão média aplicada ao motor a partir de retificadores controlados, pois utilizando estes circuitos é possível regular o nível médio da tensão retificada [9].

Retificadores são construídos principalmente com tiristores que recebem um devido sinal de disparo no terminal de *gate*. A vantagem desse tipo de acionamento é a utilização em alta potência, no entanto o alto tempo de chaveamento dos tiristores limita a frequência de operação dos mesmos [10].

2.1.3. *Choppers*

Os circuitos *choppers* são utilizados para obter tensões variáveis a partir de uma fonte de tensão de corrente contínua. A Figura 1 ilustra o circuito de um *chopper* de primeiro quadrante [10].

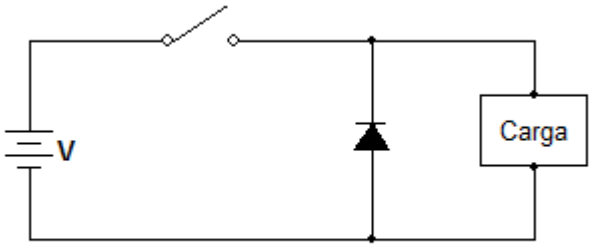


Figura 1 - *Chopper* de 1º quadrante

Controlando a posição da chave é possível controlar a tensão média aplicada à carga. Sendo T_{on}, o tempo em que a chave fica na posição fechada e T o período do sinal de controle, pode-se extrair da Figura 2 a Equação 2 que fornece a tensão média aplicada à carga.

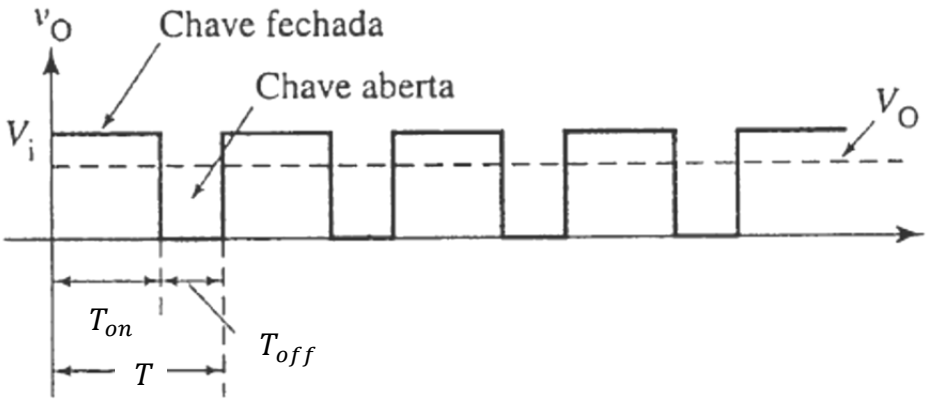


Figura 2 - Chaveamento do *chopper* [10]

$$V_o = \frac{T_{on}}{T} * V_i \tag{2}$$

Com a possibilidade de variação da tensão média, atua-se na corrente média e, portanto, no torque eletromagnético variável conforme a Equação 1.

Dois recursos para o acionamento da chave são: a modulação largura de pulso (PWM – *Pulse Width Modulation*) e a modulação por frequência de pulso (PFM – *Pulse Frequency Modulation*). A utilização do PWM é a mais usual [10] [11].

O conceito de PWM consiste em aplicar um sinal de frequência constante e variar o tempo em que a chave fica na posição fechada, ou seja, de acordo com a Figura 2, o T_{on} é variado e o período T é mantido constante.

Aplicando a Equação 1 é possível controlar a tensão média de um circuito *chopper* por este tipo de sinal.

A geração do sinal de modulação PWM é obtida pela comparação entre dois sinais, a portadora, geralmente uma onda triangular de frequência mais elevada e um sinal de referência. A alteração do sinal de referência modifica o tempo em que a chave fica fechada [12]. A Figura 3 é o sinal gerado a partir de um sinal de referência contínuo.

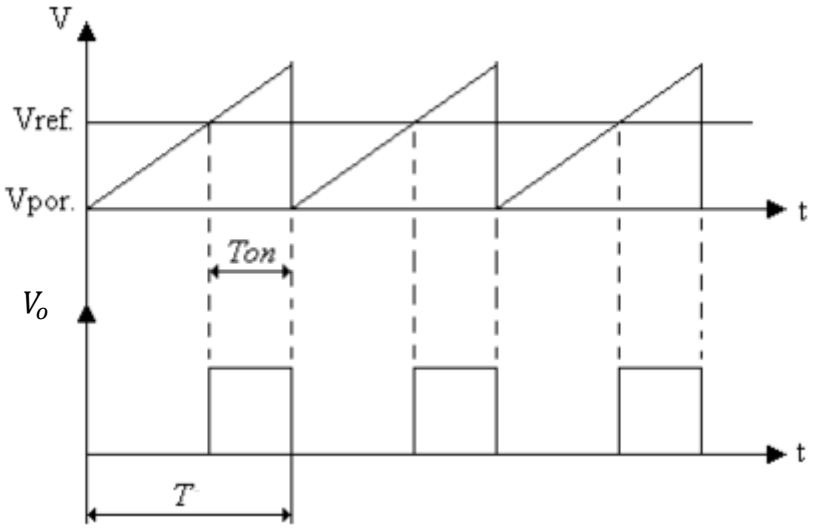


Figura 3 - Sinal PWM - Referência contínua [13]

Existem ainda outros circuitos *choppers* que operam em 2 ou 4 quadrantes, ou seja, permitem acionar o motor em dois sentidos de rotação e regeneração. [9]

Os *choppers* permitem operar com elevadas frequências de chaveamento, enquanto os retificadores exibem frequências de operação em 120 ou 360 Hz. Com elevadas frequências de chaveamento pode-se atingir melhor desempenho dinâmico do motor de corrente contínua [10].

2.2. Acionamento de motores CA

Nesta seção será abordado o acionamento para o motor de indução trifásico, que será realizado no decorrer do presente trabalho.

Motores de corrente alternada possuem velocidade diretamente relacionada à frequência da tensão de alimentação. A Equação 3a corresponde ao torque e a Equação 3b à velocidade dos motores de indução.

$$T = 3 * P * \frac{R}{s * \omega_1} * I_{rms}^2 \quad (3a)$$

$$\omega_m = \frac{2 * \omega_1}{P} * (s - 1) \quad (3b)$$

Onde:

- T = Torque
- P = Número de pares de polos do motor
- R= Resistência do rotor
- ω_1 = Frequência da tensão de alimentação
- s = Escorregamento do motor
- ω_m = Velocidade do motor

A atuação na frequência da tensão de alimentação é o mais usual e o método mais utilizado para uma excitação com frequência variável é por meio de um inversor de frequência mostrado na Figura 4. As chaves indicadas na Figura 4 podem ser implementadas com transistores bipolares, tipo MOSFET ou IGBT.

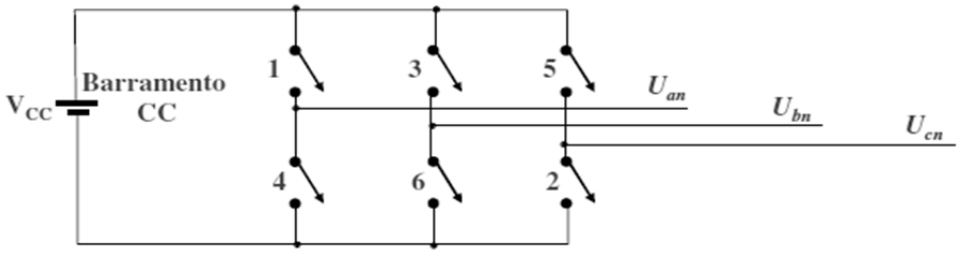


Figura 4 - Inversor de tensão [14]

Este circuito inversor de tensão trifásico opera a partir de um barramento de corrente contínua e alimenta o motor de acordo com a posição das chaves. O controle das chaves pode ser realizado utilizando os métodos de modulação PWM e PFM, além de poder ser acionado utilizando o modo 6-pulsos. No acionamento tipo 6-pulsos o motor é alimentado com forma de ondas pulsadas e periódicas, cuja componente fundamental determina a velocidade do motor [14].

O modo 6-pulsos possui dois modos de operação, o modo 2 a 2, onde apenas 2 chaves estão acionadas, simultaneamente, a cada um sexto do período da frequência desejada e o modo 3 a 3, em que 3 chaves são acionadas ao mesmo tempo. A sequência de acionamento das chaves da Figura 4 é mostrada na Figura 5 para o modo 2 a 2 e na Figura 6 para o modo 3 a 3 [14].

1 2	2 3	3 4	4 5	5 6	6 1	1 2	2 3	3 4

Figura 5 - Pulsos no modo 2 a 2 [14]

4 5 6	5 6 1	6 1 2	1 2 3	2 3 4	3 4 5	4 5 6	5 6 1	6 1 2

Figura 6 - Pulsos no modo 3 a 3 [14]

Aplicando estes pulsos é gerada uma forma de tensão teórica, entre a fase e o neutro, conforme a Figura 7 para o modo 2 a 2 e a Figura 8 para o modo 3 a 3.

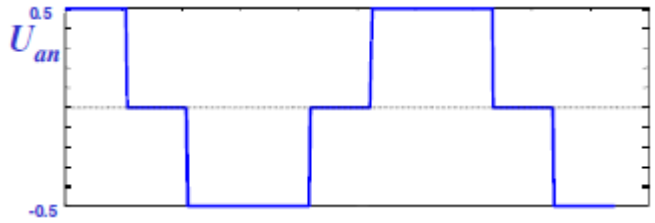


Figura 7 – Tensão teórica no modo 2 a 2 [14]

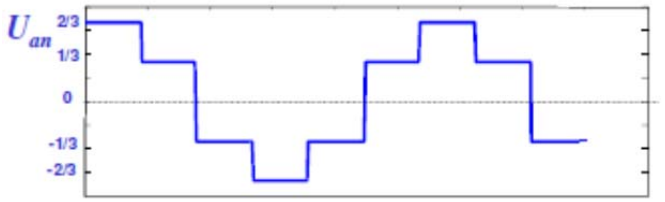


Figura 8 – Tensão teórica no modo 3 a 3 [14]

As equações 4 e 5 representam o valor RMS da tensão fundamental das formas de onda da Figura 7 e da Figura 8 respectivamente.

$$V_{RMS} = \frac{\sqrt{3}}{\pi} V_{cc} \tag{4}$$

$$V_{RMS} = \frac{2}{\pi} V_{cc} \tag{5}$$

2.2.1. Modulação por largura de pulso

A modulação por largura de pulso também pode ser utilizada para motores de corrente alternada no acionamento utilizando inversores de tal forma que o sinal modulante seja uma senoide de frequência desejada para a velocidade do motor.

Numa versão digital deste procedimento, utiliza-se de um *timer* programável, o qual é programado para contar repetidamente até um determinado valor e, em função

da frequência de *clock* escolhido do valor de contagem obtém-se a respectiva portadora. O efeito PWM é obtido por comparação entre o valor da contagem do *timer* com um valor representando o sinal modulante.

Com referência à Figura 3, do princípio da modulação largura de pulso, aplicando-se um sinal de referência senoidal é possível obter a chamada modulação por largura de pulso senoidal (SPWM – Sinusidal Pulse Width Modulation). A Figura 9 ilustra este procedimento.

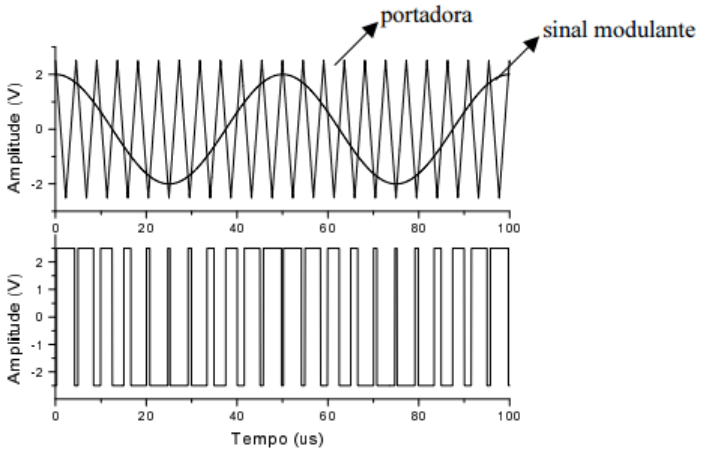


Figura 9 - Modulação por largura de pulso senoidal [15]

Para o caso do motor de indução trifásico é utilizada uma mesma portadora com três sinais modulantes senoidais defasados de 120 graus elétricos para compor uma alimentação equilibrada nos terminais do motor de indução.

Para execução do SPWM digitalmente são necessários 3 módulos de comparação para produzir os sinais de comando das chaves da ponte trifásica.

3. Descrição do *kit* de desenvolvimento

O *kit* de desenvolvimento EKI LM3S8962, mostrado na Figura 10, é uma ferramenta muito importante neste projeto já que é responsável pelos sinais de acionamento dos motores, leitura dos valores dos transdutores utilizados (através do conversor analógico digital e do *encoder*) além de transmitir estes valores via serial. Por meio do *kit* de desenvolvimento é possível também implementar o controle das máquinas.

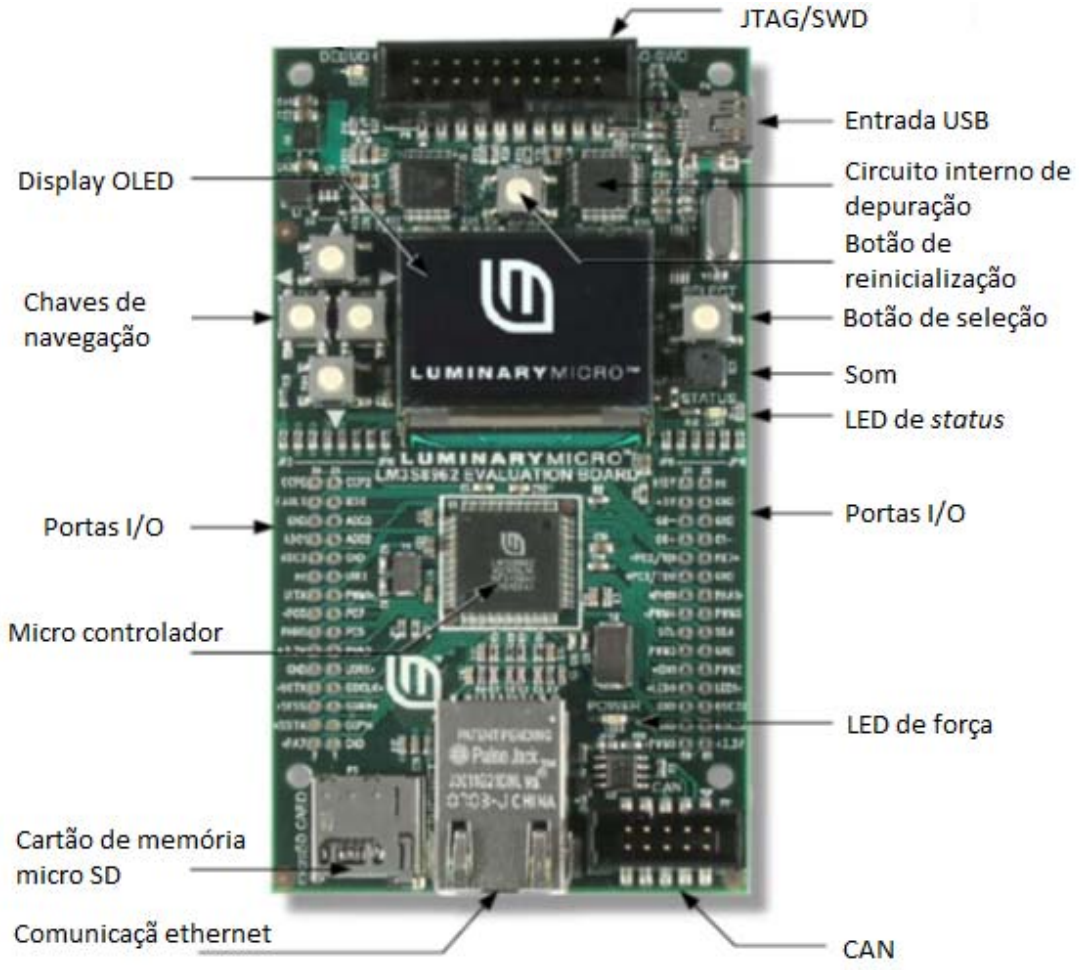


Figura 10 - *Kit* de desenvolvimento EKI LM3S8962 [1]

O *kit* de desenvolvimento é uma plataforma baseada no micro controlador *Stellaris LM3S8962 ARM® Cortex*™-M3 [2] e possui diversos recursos como:

- Comunicação ethernet
- Entrada USB (utilizada para alimentação e comunicação serial)

- *Display* gráfico
- Entrada para cartão de memória MicroSD
- Interface JTAG
- Comunicação serial (síncrona e assíncrona)
- Quatro conversores analógicos digitais
- Comparador analógico
- Protocolo de comunicação I²C
- Módulos geradores de PWM
- Interface para *encoder*
- Até 42 portas I/O

O *kit* de desenvolvimento possui também um CD para instalação do programa *IAR Embedded Workbench* [16] utilizado para compilar e descarregar os programas na memória do microprocessador. O programa possui ainda uma ferramenta de depuração (*Debug*). O código da programação pode ser escrito na linguagem *assembler*, em C ou em C++.

O código é baseado em uma biblioteca de *drivers* periféricos (*Stellaris Peripheral Driver Library*) [17] esta biblioteca possui funções já específicas para os *drives* utilizados, facilitando a programação e, principalmente, o entendimento do código, já que estas funções possuem uma linha de raciocínio lógica.

Os principais módulos constituintes do *kit* de desenvolvimento são descritos nes capítulo. No capítulo 4 serão esclarecidos como tais módulos são configurados dentro da estrutura de um programa para acionamento dos motores descritos no capítulo 2.

3.1. *Modulo display* OLED

O *kit* de desenvolvimento possui um *display* OLED (diodo orgânico emissor de luz) de resolução de 128 x 96 pixels, de alto contraste (500:1) e de brilho 120cd/m².

O *display OLED* se baseia no princípio conhecido como eletroluminescência, “[...] compostos orgânicos interagem de várias maneiras com fontes de energia diversas, e neste caso em particular, a interação ocorre quando a passagem da corrente elétrica provoca a emissão de luz pela substância. [...]“ [18].

O *display* pode ser utilizado para informar o *status* do programa além de permitir imprimir os valores amostrados, indicar possíveis falhas, e solicitar decisão do usuário. Este recurso, no entanto, não deve ser utilizado em *loop* quando se desejam elevadas taxas de aquisição e/ou controle, pois a impressão dos valores no *display* é uma operação lenta.

3.2. Modulo timer

Timers são funções baseadas na frequência de operação do processador que funcionam como um relógio (temporizador) e a cada período específico de tempo gera uma informação ou dispara uma interrupção dependo da configuração utilizada

O *kit* de desenvolvimento possui 4 módulos de *timers* independentes que podem ser utilizados também como contadores. Cada módulo pode ser configurado como 2 *timers* de 16 *bits* independentes ou como 1 *timer* de 32 *bits*.

3.3. Módulo conversor analógico digital (CAD)

Um conversor analógico digital (CAD) tem como função transformar um sinal analógico (contínuo) em uma representação digital (binária), ou seja, o CAD transforma um nível de tensão analógica em uma sequência de dígitos (0 ou 1).

O *kit* de desenvolvimento EKI LM3S8962 possui um periférico, com 4 entradas analógicas, que converte um sinal analógico, de 0V a 3V, em uma representação digital de 10 bits, ou seja, uma resolução de, aproximadamente, 0,003V. Sendo assim um sinal de 3V na entrada do conversor irá fornecer um resultado de 1023 (convertendo os 10 bits para o sistema decimal) e uma entrada de 0V fornece um resultado de 0.

O controle de amostragem é realizado utilizando um sequenciador de amostras, este recurso permite a aquisição de múltiplas entradas analógicas. A configuração do conversor analógico digital e a aquisição de dados ocorrem em cada sequência. Existem 4 sequências diferentes, cada uma com seu tamanho de acordo com a Tabela 1.

Tabela 1 - Sequências de amostragem

Sequência	Número de amostras
SS3	1
SS2	4
SS1	4
SS0	8

3.4. Módulo de Portas I/O

Portas I/O (*Input/Output*) são pinos que trocam informação entre o micro controlador e o meio externo, podendo ser tanto entrada (*Input*) como saída (*Output*), dependendo da configuração.

O *kit* de desenvolvimento possui um número variável de portas I/O já que muitas delas têm dupla função (por exemplo, PWM), o número máximo é de 42.

3.5. Módulo PWM

O *kit* de desenvolvimento EKI LM3S8962 possui 3 módulos geradores de PWM onde cada módulo possui 2 saídas que podem ser utilizadas independentes ou como pares. Cada bloco gerador possui configuração independente.

No caso de operação em pares, o módulo produz o sinal de acionamento de um dos braços da ponte completa da Figura 4. A saída aos pares é produzida de maneira complementar, possibilitando ligar a chave superior e desligar a inferior de um mesmo braço. Como forma de segurança e proteção das chaves é possível estipular um valor *Dead-band* que atrasa um destes sinais evitando colocar o barramento de corrente contínua em curto-circuito.

Não existem saídas específicas para os PWMs, ou seja, a mesma porta PWM pode ser configurada como porta I/O ou até mesmo outra função. A Tabela 2 lista as funções destas portas.

Tabela 2 - Função extra das portas PWM

PWM	Porta	Função extra
0	Porta F Pino 0	*User* LED
1	Porta G Pino 1	Som
2	Porta B Pino 0	
3	Porta B Pino 1	
4	Porta E Pino 0	Tecla acima
5	Porta E Pino 1	Tecla abaixo

Para a utilização de todos os PWMs foi desabilitada a função de som do *kit* de desenvolvimento, sendo que para isto foi cortada conexão entre eles. É possível reabilitar esta função utilizando um conector (*jump*) na posição localizada logo abaixo às chaves de navegação.

3.6. Módulo *encoder* QEI

O *encoder* é a designação de um dispositivo mecânico utilizado para realizar medidas de posição e velocidade de mecanismos rotativos. Seu funcionamento consiste em acoplar um disco perfurado ao eixo do motor, utilizando um emissor e um receptor de luz em cada lado do disco perfurado.

O movimento do eixo e consequentemente do disco perfurado faz com que o detector de luz receba pulsos que, através de sua contagem, fornece a posição do eixo do motor. O tamanho do pulso e sua frequência variam de acordo com a velocidade de rotação do eixo do motor e do número perfurações no disco. Desta forma é possível extrair a informação da velocidade e de posição de rotação do motor.

O *kit* de desenvolvimento EKI LM3S8962 possui dois módulos (*Quadrature Encoder Interface* - QEI), que faz o cálculo necessário para determinar a posição, velocidade e sentido de rotação do motor. Os pulsos devem ser de 3V para o sinal alto e 0V para o sinal baixo e são lidos pelas portas PhA0 e PhB0 (QEI módulo 0) e PhA1 e PhB1 (QEI módulo 1).

De modo semelhante às portas PWM, as portas utilizadas pelos *encoder*s também possuem mais que uma função, a Tabela 3 lista estas portas.

Tabela 3 - Função extra das portas do *encoder*

Encoder	**Porta**
PhA0	Porta C Pino 4
PhB0	Porta C Pino 6
PhA1	Porta E Pino 2
PhB1	Porta E Pino 3

3.7. Módulo de comunicação serial

A comunicação serial é aquela feita *bit* por *bit* em sequência por uma única linha. Existem dois diferentes tipos de comunicação serial, a síncrona e assíncrona.

Na transmissão assíncrona o envio da palavra é inicializado por um *bit* de partida e finalizado por um *bit* de parada (*stop bit*), de forma que o receptor possa identificar a palavra enviada. A transmissão síncrona não possui os *bits* de inicialização nem o de parada já que o sincronismo é feito por caracteres de sincronismo [19]. A que será utilizada é a assíncrona.

O *kit* de desenvolvimento EKI-LM3S8962 possui um módulo de comunicação serial que trabalha em conjunto com o componente FT232R [20] da FTDI e permite a comunicação serial via USB. Dessa forma a comunicação entre o *kit* de desenvolvimento é feito pelo mesmo cabo da alimentação.

Para a leitura dos dados (realizada pelo computador) foi utilizado o programa *Hyperterminal* [21], onde é possível ler os dados que estão sendo recebidos e armazená-los.

3.8. Configuração do h*yperterminal*

O programa *Hyperterminal* era muito utilizado na época da internet discada para comunicação entre computadores, por isso a primeira tela de configuração, mostrada na Figura 11 é referente à rede, como não será utilizado podem-se cancelar estas informações.

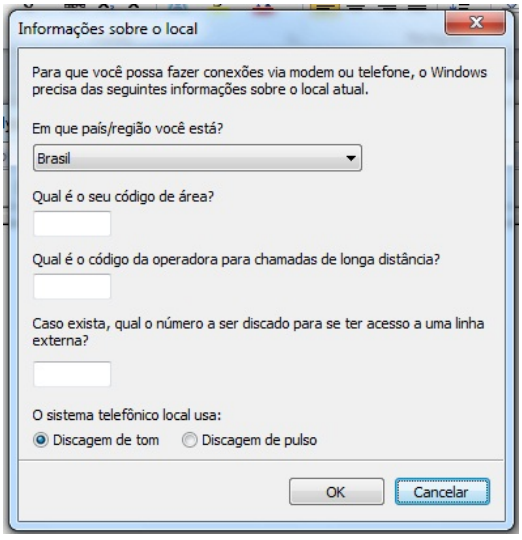


Figura 11 - *Hyperterminal* – Inicialização

Em seguida, é necessário configurar, conforme a Figura 12, o *hyperterminal* para utilizar a porta de comunicação a ser utilizada (no caso COM5).

Figura 12 - *Hyperterminal* - Configuração da porta

O próximo passo consiste na configuração da porta, conforme a Figura 13. Estes dados devem ser configurados de acordo com as informações definidas pelo emissor. Os dados necessários para configuração são:

- Velocidade de transmissão (*bits* por segundo)
- Número de *bits* de dados
- *Bit* de paridade
- *Bit* de parada
- Controle de fluxo

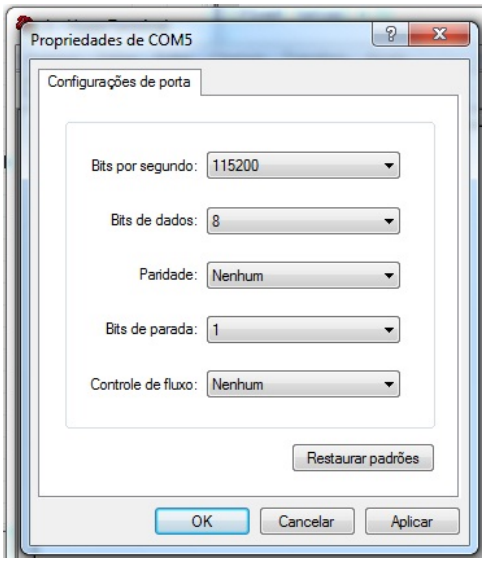


Figura 13 - *Hyperterminal* - Configuração da transmissão

Neste ponto a comunicação já estará ocorrendo e os dados, assim que recebidos, serão impressos na tela. É possível ainda armazenar esses dados em um arquivo de texto como mostra a Figura 14.

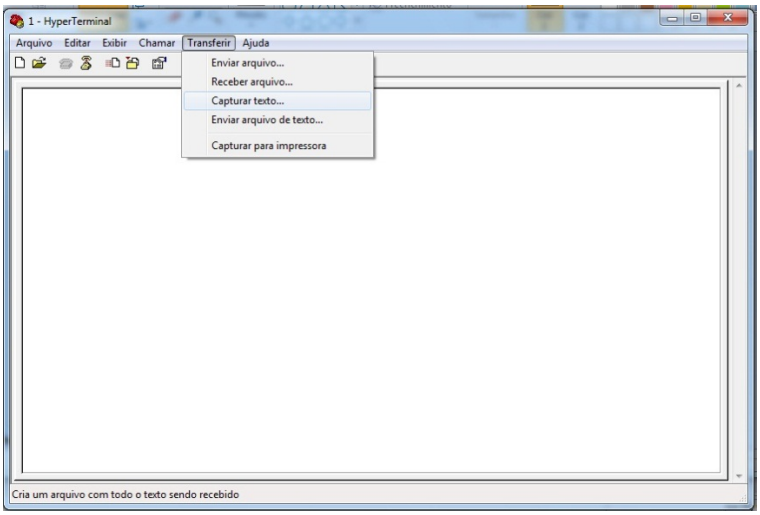


Figura 14 - *Hyperterminal* - Salvar dados

4. Programação do *kit* de desenvolvimento EKI LM3S8962

O código global de programação foi dividido em 4 partes, a inicialização, configuração dos recursos utilizados, um bloco de loop infinito e a parte de comunicação serial. O diagrama da Figura 15 exemplifica a estrutura.

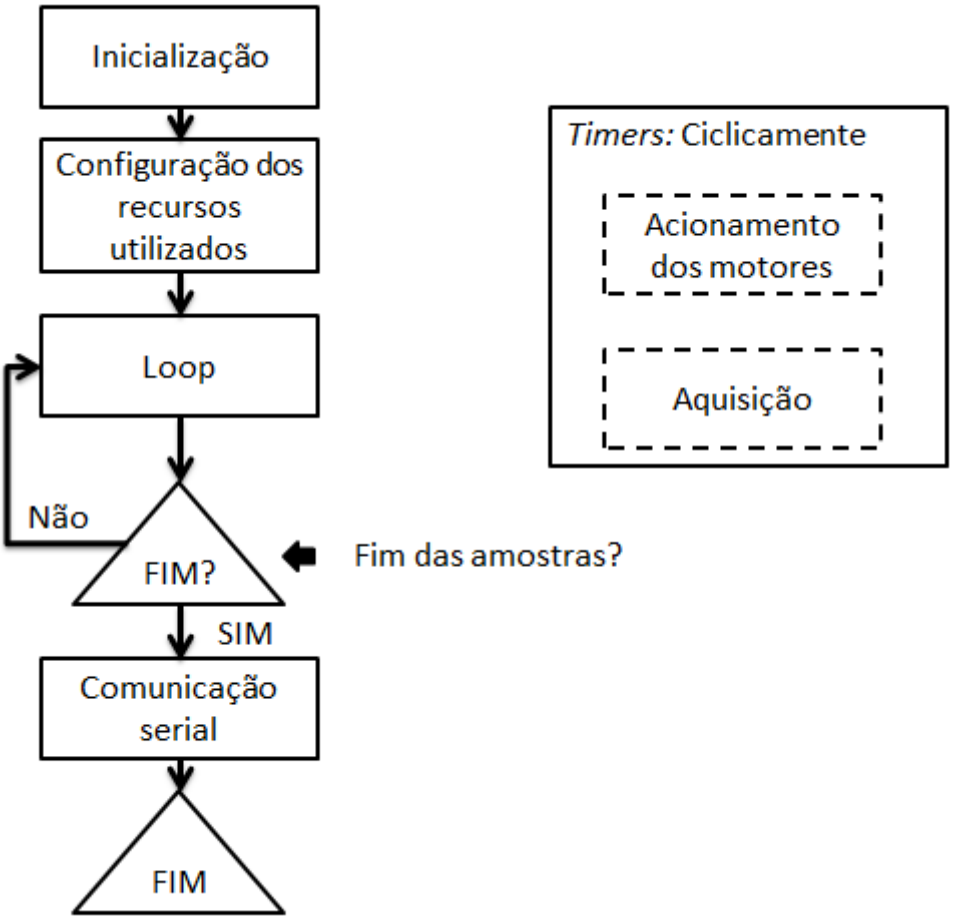


Figura 15 - Diagrama do código

Em todos os testes do *kit* em si e também nas etapas de acionamento, o código desenvolvido, em linguagem C, promove a inicialização das bibliotecas utilizadas e das variáveis globais. Em seguida, procede-se a configuração dos recursos utilizados. O próximo passo ocorre a leitura dos dados e da rotina de acionamento dos motores. Por fim ocorre a comunicação serial, onde as amostras são enviadas a um *host*. O anexo I apresenta o código implementado completo.

4.1. Inicialização

O bloco de inicialização compreende a inclusão das bibliotecas utilizadas além da inicialização das variáveis globais. Estas bibliotecas possuem basicamente a definição

de diretivas que permitirão a devida configuração dos vários módulos que se pretende utilizar.

As bibliotecas utilizadas e os comandos utilizados para incluí-las são listados no bloco de programa a seguir:

- #include "inc/hw_memmap.h"
- #include "inc/hw_types.h"
- #include "inc/hw_ints.h"
- #include "inc/hw_qei.h"
- #include "inc/lm3s8962.h"
- #include "inc/hw_ssi.h"
- #include "driverlib/debug.h"
- #include "driverlib/gpio.h"
- #include "driverlib/pwm.h"
- #include "driverlib/sysctl.h"
- #include "driverlib/adc.h"
- #include "drivers/rit128x96x4.h"
- #include "driverlib/interrupt.h"
- #include "driverlib/sysctl.h"
- #include "driverlib/*timer*.h"
- #include "driverlib/qei.h"
- #include "driverlib/uart.h"
- #include <stdio.h>

Caso as bibliotecas adicionadas não sejam encontradas durante a programação, uma mensagem de erro é chamada. Esta verificação é feita utilizando a seguinte função.

```
#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
```

As variáveis globais utilizadas são as variáveis utilizadas durante as interrupções, além das variáveis que definem informações necessárias para a execução do programa, essas informações são o tempo de interrupção dos *timers*, número de amostras e algumas informações referentes à configuração do PWM. Ilustra-se a seguir um bloco de inicialização.

- unsigned long q=0;
- unsigned long ulPeriod;
- volatile unsigned long ulLoop;
- unsigned long ulValue;
- char buf[40];
- int e0;
- int count=0;
- float tatual = 0;
- int p=0;
- int portae=0;

- int flagv=0;
- int l=0;
- float aux;

Também neste ponto são definidos os tamanhos dos vetores utilizados para o armazenamento das amostras.

- unsigned long cad[2000];
- int vel[2000];
- int pos[2000];
- int velt[2000] = NULL;

Um exemplo de inicialização de constantes, definindo a quantidade de amostras, frequência de amostragem e frequência da portadora dos PWM a serem utilizados é:

- int amostras=7500;
- float TA=0.001;
- int fpwm = 2000;

A variável amostras representa a quantidade total de aquisições que será realizada. A variável TA é o tempo, em segundos, do ciclo de amostras, ou seja, é o inverso da frequência de aquisição. A variável fpwm é a frequência de operação do PWM utilizados e as outras 3 amostras são recursos utilizados

4.2. Configuração dos recursos utilizados

O bloco de configuração dos recursos como nome diz é onde são configurados os recursos como *timers*, módulo serial, *encoder*, portas I/O, *display*, entre outros.

4.2.1. Configuração do clock do utilizado

A definição do *clock* consiste em escolher a base de tempo de operação do processador, ou seja, quanto tempo será gasto por cada operação do processador. Existem diversas configurações diferentes, podendo ser baseado no cristal presente no *kit* de desenvolvimento EKI LM3S8962 ou baseado em um oscilador externo. Existe ainda a possibilidade de variar a frequência através de divisores. Uma configuração que pode ser utilizada é:

- SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN | SYSCTL_XTAL_8MHZ);
- SysCtlPWMClockSet(SYSCTL_PWMDIV_1);

Onde é selecionado 8 MHz de frequência de operação, proveniente do cristal.

4.2.2. Configuração do OLED *display*

A configuração do *display* OLED consiste em 3 comandos, o de inicialização, o de escrita e o de limpeza da tela. Os comandos, respectivamente, são:

- RIT128x96x4Init(1000000);

- RIT128x96x4StringDraw("Amostrando", 20, 20, 15);
- RIT128x96x4Clear();

Onde o primeiro campo da função de escrita é a palavra a ser escrita, sendo uma variável do tipo *char*, o segundo e o terceiro campo são a posição onde será escrita a palavra no eixo imaginário abaixo. O último campo é a intensidade do brilho dos caracteres impressos. A Figura 16 representa o eixo das posições do *display*.

Figura 16 - Eixo do *display* OLED

Uma função auxiliar converte um número em uma variável do tipo *char* para que a função de escrita possa enviar números:

- sprintf(buf, "%d", vel[count])

Onde buf é a variável do tipo *char* onde será armazenado o resultado da operação, %d é operador necessário para indicar a conversão de uma variável do tipo *int* para o tipo *char* e vel[count] é o valor inteiro a ser convertido. Desta maneira basta utilizar o comando de escrita da seguinte maneira:

- RIT128x96x4StringDraw(buf, 20, 20, 15);

4.2.3. Configuração dos *timers*

Os *timers* foram utilizados como interrupção de forma a garantir que as operações aconteçam sempre em um período cíclico. O *timer* 0 foi utilizado para amostragem dos sinais, ou seja, leitura dos conversores analógicos digitais e dos *encoders* e o *timer* 1 como geração dos sinais de acionamento, seja o acionamento via PWM ou pelas portas I/O.

A configuração dos *timers* é realizada da seguinte maneira.

Primeiramente, é habilitada a função do *timer*:

- SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);

Onde o objeto SYSCTL_PERIPH_*TIMER*0, representa a posição de memória referente ao *timer* 0, no caso de utilização de outros *timer* (1, 2 ou 3), deve-se utilizar a posição específica.

Em seguida é configurado o *timer* em si. Em praticamente todas as funções é utilizado o comando *TIMER*X_BASE, onde X representa o *timer* utilizado. A primeira função é referente á configuração do *timer*.

- TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);

A Tabela 4 representa as possibilidades de configuração do segundo campo da função.

Tabela 4 - Comando de configuração dos *timers*

Comando	Função
TIMER_CFG_32_BIT_OS	*Timer* de 32 *bits* com disparo único
TIMER_CFG_32_BIT_PER	*Timer* de 32 *bits* com periódico
TIMER_CFG_32_RTC	*Timer* de 32 *bits* com clock em tempo real
TIMER_CFG_16_BIT_PAIR	Utilização de dois *timers* de 16 *bits* independente

No caso de utilização do *timer* de 16 *bits* independente deve ser adicionada uma outra configuração utilizando uma operação ou entre a configuração mostrada na Tabela 4 com a Tabela 5. No caso é utilizado a configuração do *timer* A.

Tabela 5 - Comando de configuração dos *timers* 16 *bits*

Comando	Função
TIMER_CFG_A_ONE_SHOT	*Timer* de 16 *bits* com disparo único
TIMER_CFG_A_PERIODIC	*Timer* de 16 *bits* com periódico
TIMER_CFG_A_CAP_COUNT	Contador de 16 *bits*
TIMER_CFG_A_CAP_TIME	Timer de 16 bits utilizando sinal externo
TIMER_CFG_A_PWM	*Timer* de 16 *bits* em modo PWM

Um exemplo de configuração de um *timer* de 16 *bits*, e disparo único seria:

- TimerConfigure(TIMER0_BASE, TIMER_CFG_16_BIT_PAIR | TIMER_CFG_A_ONE_SHOT);

O comando responsável pela configuração do valor que o *timer* deve contar é:

- TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet()*TA);

Onde o primeiro valor é o *timer* em questão. O segundo valor depende da configuração utilizada (16 ou 32 bits), caso seja de 32 *bits*, deve ser utilizado o valor TIMER_A, no caso utilizar o de 16 *bits* deve-se utilizar o *timer* em questão (A ou B). O terceiro valor é a quantidade de pulsos do *clock* que se quer contar até que o *timer* dispare. Neste ponto é utilizada uma função auxiliar que retorna o valor do *clock* utilizado em HZ, ou seja, representa a quantidade de pulsos do clock durante 1 segundo. Dessa forma utilizando esta função no 3º campo da função e multiplicando a função por uma variável TA é possível controlar o tempo de disparo adicionando o valor desejado, em segundos, à variável.

Em seguida é necessário habilitar a interrupção do *timer* em questão além de configurar para que a interrupção seja executada quando o contador chegue ao valor adicionado na função anterior.

- IntEnable(INT_TIMER0A);
- TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);

Por fim a contagem é iniciada.

- TimerEnable(TIMER0_BASE, TIMER_A);

Após a interrupção ser chamada é necessário limpar a *flag* de interrupção com o comando abaixo, dessa forma a contagem é reiniciada.

- TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);

4.2.4. Configuração do conversor analógico digital (CAD)

O programa foi estruturado de tal forma que, quando há uma interrupção de tempo, o valor do CAD é lido e armazenado em um vetor, no entanto a configuração deste recurso e programada em conjunto com os outros recursos.

O primeiro passo é habilitar o periférico do conversor analógico digital:

- SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC);

Em seguida é configurada a sequência de amostras do conversor analógico digital:

- ADCSequenceConfigure(ADC_BASE, 0, DC_TRIGGER_PROCESSOR, 1);

O valor ADC_BASE representa a posição de memória correspondente aos conversores analógicos digitais. O segundo comando representa qual a sequência a ser configurada, 0 a 3. O terceiro valor representa quando será armazenado o valor obtido pelo conversor analógico digital. Os valores possíveis são mostrados na Tabela 6. O último valor é a prioridade do conversor analógico digital utilizado em relação aos outros conversores.

É importante ressaltar que caso seja configurado para realizar a leitura sempre, o processador perde desempenho, além de não haver a possibilidade de utilizar 2 conversores com essa configuração ao mesmo tempo.

Em seguida é realizada a configuração de cada passo da sequência:

- ADCSequenceStepConfigure(ADC_BASE, 3, 0, ADC_CTL_CH0 | ADC_CTL_END);

O valor ADC_BASE representa a posição de memória correspondente aos conversores analógicos digitais. O segundo valor representa qual sequência se deseja configurar. O terceiro valor representa qual passo da sequência será configurado. O

último valor é a configuração desejada e que é resultado de uma operação ou entre um ou mais comandos da Tabela 7.

Tabela 6 - Configuração da sequência do CAD

Comando	Função
ADC_TRIGGER_PROCESSOR	O disparo é realizado utilizando a função ADCProcessorTrigger()
ADC_TRIGGER_COMP0	O disparo é feito com base no comparador analógico configurado com a função ComparatorConfigure().
ADC_TRIGGER_COMP1	O disparo é feito com base no comparador analógico configurado com a função ComparatorConfigure().
ADC_TRIGGER_COMP2	O disparo é feito com base no comparador analógico configurado com a função ComparatorConfigure().
ADC_TRIGGER_EXTERNAL	O disparo é feito com base por um comando externo pela porta B4
ADC_TRIGGER_TIMER	O disparo é feito por um timer configurado com a função TimerControlTrigger().
ADC_TRIGGER_PWM0	O disparo é feito com base no PWM configurado com a função PWMGenIntTrigEnable().
ADC_TRIGGER_PWM1	O disparo é feito com base no PWM configurado com a função PWMGenIntTrigEnable().
ADC_TRIGGER_PWM2	O disparo é feito com base no PWM configurado com a função PWMGenIntTrigEnable().
ADC_TRIGGER_ALWAYS	A leitura é realizada sempre.

Tabela 7 - Configuração dos passos da sequência

Comando	Função
ADC_CTL_CHX	Configura qual porta deve ser amostrada, onde X pode ser 0,1, 2 ou 3.
ADC_CTL_END	Configura que o determinado passo é o último da sequência
ADC_CTL_IE	Configura que o determinado passo dispara a interrupção

Por fim é inicializada a sequência:

- ADCSequenceEnable(ADC_BASE, 0);

Onde a sequência habilitada é selecionada no segundo campo da função.

A leitura é realizada dentro da interrupção do *timer*, o disparo da leitura é realizado pelo comando:

- ADCProcessorTrigger(ADC_BASE, 0);

Onde o segundo valor representa a sequência que deve ser utilizada.

Para leitura e armazenagem da sequência o comando indicado na biblioteca é:

- ADCSequenceDataGet(ADC_BASE, 0, *cad);

Onde o segundo valor representa a sequência ser lida e o último valor representa o endereço onde será armazenado este valor. Durante a realização da programação percebeu-se o não funcionamento conforme descrito na biblioteca, do comando. Foi utilizado um recurso lendo diretamente o valor da posição de memória da sequência, no caso é armazenado o último passo da sequência.

- cad[count] = ADC_SSFIFO0_R;

Esta leitura direta da ultima posição de memória da sequência implica na necessidade de repetição do comando pelo número de passos da sequência em questão, conforme a Tabela 1, já que o armazenamento dos valores na sequência tem o conceito FIFO (*First In First Off*). O valor lido representa o valor da sequência 0, o valor de outras sequências pode ser obtido utilizando o valor ADC_SSFIFOX_R, trocando X pela sequência em questão.

4.2.5. Configuração de portas I/O

As portas I/O são configuradas independentemente, no entanto é possível ler e escrever valores nelas na forma de *byte*. A configuração consiste em configurar a porta como I/O e configurar a direção (Entrada ou saída de dados):

- GPIOPinTypeGPIOOutput(GPIO_PORTX_BASE, GPIO_PIN_Y);
- GPIODirModeSet(GPIO_PORTX_BASE, GPIO_PIN_Y , GPIO_DIR_MODE_OUT);

Onde os primeiros comandos de cada função são as portas utilizadas, trocando X pela porta desejada (A a F). Já os segundos valores das funções representam quais pinos da determinada porta serão configurados, trocando Y pelo pino (0 a 7).

Os comandos de escrita e leitura são:

- GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 | GPIO_PIN_7), 12);
- portae = GPIOPinRead(GPIO_PORTE_BASE,(GPIO_PIN_2 | GPIO_PIN_1 | GPIO_PIN_0);

Onde no de escrita são configurados a porta em que deverá ser escrito o valor, os pinos da determinada porta e o valor (*byte*) que deverá ser escrito. O comando de escrita armazena os valores da determinada porta e seus pinos em uma variável.

4.2.6. Configuração do PWM (Modulação largura de pulso)

O primeiro passo para configuração do PWM é habilitar o periférico e a porta I/O que possui a dupla função de PWM.

- SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM);
- SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

Em seguida é necessário configurar os pinos das portas I/O (no exemplo é utilizado o pino da porta F) que tem dupla função (de PWM) para que operem neste modo:

- GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_0);

Onde o primeiro valor indica qual a porta e o segundo, qual pino da determinada porta.

A configuração do PWM é realizada em dois passos. Primeiramente é realizada a configuração da geração do sinal.

- PWMGenConfigure(PWM_BASE, PWM_GEN_0, PWM_GEN_MODE_UP_DOWN | PWM_GEN_MODE_NO_SYNC);

Onde o valor PWM_BASE se repete em todas as possíveis configurações, o segundo indica qual módulo gerador de PWM será utilizado (0, 1 ou 2). O terceiro valor é obtido realizando uma função ou entre os valores mostrados na Tabela 8 e na Tabela 9.

Tabela 8 - Configuração da geração do PWM

Comando	Função
PWM_GEN_MODE_UP_DOWN	PWM gerado a partir de uma onda portadora triangular
PWM_GEN_MODE_DOWN	PWM gerado a partir de uma onda portadora do tipo dente de serra decrescente

Tabela 9 - Configuração do sincronismo do PWM

Comando	Função
PWM_GEN_MODE_SYNC	Habilita o modo de sincronização
PWM_GEN_MODE_NO_SYNC	Desabilita o modo de sincronização

O período do pulso que deve ser gerado pelo PWM é o próximo passo da programação. Para facilitar a programação é utilizado o mesmo recurso na configuração dos *timers*, ao invés de programar quantos ciclos de *clock* deve durar.

- ulPeriod = SysCtlClockGet() / fpwm

Onde fpwm representa a frequência desejada para o PWM, e ulPeriod a quantidade de ciclos necessários para a frequência desejada. O comando de configuração é:

- PWMGenPeriodSet(PWM_BASE, PWM_GEN_0, ulPeriod);

onde o primeiro valor representa a configuração de um PWM, o segundo é variado de acordo com o PWM que se deseja programar e o terceiro é o resultado do recurso utilizado.

O recurso também é utilizado para configurar o tempo em que o pulso deve ficar em nível logico alto.

- PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, ulPeriod*inipwm/100);

Onde o segundo valor da função representa qual PWM (do bloco gerador de PWM, 0 ou 1) será utilizado. Utilizando o recurso anterior é possível definir a porcentagem do pulso em nível alto em comparação ao período, este valor é alocado na variável inipwm.

Por fim a saída é habilitada:

- PWMOutputState(PWM_BASE, PWM_OUT_0_BIT , true);

Onde o segundo comando representa qual PWM deve ser habilitado, este valor varia de acordo com a Tabela 10. O último valor habilita ou não o PWM em questão

Tabela 10 - Habilitação do PWM

Comando	Função
PWM_OUT_0_BIT	PWM 0 do bloco gerador 0
PWM_OUT_1_BIT	PWM 1 do bloco gerador 0
PWM_OUT_2_BIT	PWM 0 do bloco gerador 1
PWM_OUT_3_BIT	PWM 1 do bloco gerador 1
PWM_OUT_4_BIT	PWM 0 do bloco gerador 2
PWM_OUT_5_BIT	PWM 1 do bloco gerador 3

4.2.7. *Encoder*

Do mesmo modo que o CAD, o programa foi estruturado de forma a ler o valor do módulo QEI (*Quadrature Encoder Interface*) e extrair as informações desejadas a cada intervalo de tempo pré-definido. Os valores são armazenados em um vetor e posteriormente enviados via serial.

O primeiro comando é inicialização do periférico do *encoder* e as portas que possuem a dupla função (de *encoder*), além de configurar o pino da determinada porta com dupla função para operar no modo de *encoder*.

- SysCtlPeripheralEnable(SYSCTL_PERIPH_QEI);
- SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
- GPIOPinTypeQEI(GPIO_PORTC_BASE, GPIO_PIN_4);

Em seguida é indicado desabilitar o *encoder* durante a configuração. O mesmo vale para a configuração da medição de velocidade.

- QEIDisable(QEI0_BASE);
- QEIVelocityDisable(QEI0_BASE);

O próximo passo é configurar a porta I/O como *encoder*.

- GPIOPinTypeQEI(GPIO_PORTC_BASE, GPIO_PIN_4);

Onde o primeiro valor representa a porta e o segundo valor, o pino da referente porta.

A configuração do módulo em si é realizada da seguinte maneira:

- QEIConfigure(QEI_BASE,QEI_CONFIG_CAPTURE_A | QEI_CONFIG_NO_RESET | QEI_CONFIG_QUADRATURE | QEI_CONFIG_NO_SWAP, 2999);

Onde o primeiro valor representa a configuração do *encoder* e é sempre este valor (QEI_BASE). O segundo valor é o resultado da função lógica ou dos valores mostrados na Tabela 11, Tabela 12, Tabela 13 e Tabela 14. No último valor da função deve ser colocado o número de pulsos do *encoder* (físico) subtraindo 1.

Tabela 11 - Configuração da captura do *encoder*

Comando	Função
QEI_CONFIG_CAPTURE_A	Específica se as bordas (tanto de subida quanto de descida) dos pulsos devem ser contadas apenas a partir da fase A
QEI_CONFIG_CAPTURE_A_B	Específica se as bordas (tanto de subida quanto de descida) dos pulsos devem ser contadas a partir da fase A e da fase B

Tabela 12 - Configuração do reset de contagem do *encoder*

Comando	Função
QEI_CONFIG_NO_RESET	Específica que a contagem não deve ser zerada
QEI_CONFIG_RESET_IDX	Específica se a contagem deve ser zerada caso o sinal de *index* (de posição 0) seja lido

Tabela 13 - Configuração do calculo de direção do *encoder*

Comando	Função
QEI_CONFIG_QUADRATURE	Específica se o sinal de direção é proveniente da comparação entre as duas fases.
QEI_CONFIG_CLOCK_DIR	Específica se o sinal de direção é proveniente de um sinal externo

Tabela 14 - Configuração do calculo de direção do *encoder*

Comando	Função
QEI_CONFIG_NO_SWAP	Específica que os sinais não devem ser permutados
QEI_CONFIG_SWAP	Específica que os sinais devem ser permutados

Em seguida é necessário configurar o cálculo da velocidade:

- QEIVelocityConfigure(QEI_BASE, QEI_VELDIV_1, 100000);

O primeiro valor da função é o valor base para *encoder*. O segundo é um pré-divisor que pode ser utilizado em caso de uma contagem muito alta de pulsos, os valores possíveis estão na Tabela 15. O terceiro campo é o número de ciclos que o *encoder* deve esperar para realizar o cálculo da velocidade (já que o cálculo depende do tempo).

Tabela 15 – Pré-divisores

Comando
QEI_VELDIV_1
QEI_VELDIV_2
QEI_VELDIV_4
QEI_VELDIV_8,
QEI_VELDIV_16
QEI_VELDIV_32
QEI_VELDIV_64
QEI_VELDIV_128

Em seguida são habilitados os módulos de velocidade e o do *encoder*.

- QEIVelocityEnable(QEI0_BASE);
- QEIEnable(QEI0_BASE);

A leitura dos valores é feita com os seguintes comandos:

- dir[count] = QEIDirectionGet(QEI0_BASE);
- vel[count]= XX*QEIVelocityGet(QEI0_BASE);
- pos[count] = QEIPositionGet(QEI0_BASE);

O valor XX que multiplica a função que extrai a velocidade é o resultado da Equação 4, que diferencia o *encoder* utilizado, e a configuração utilizada.

$$RPM = \frac{Clock * 2^{VelDiv} * Velocidade * 60}{Load * PPR * Edge} \quad (4)$$

Onde:

- RPM = É o resultado da operação na unidade de rotações por minuto
- $Clock$ = Taxa de operação do microprocessador, no caso 8MHz
- $VelDiv$ = É o prédivisor, os valores possíveis estão na Tabela 15
- $Velocidade$ = Valor fornecido pela função QEIVelocityGet
- $Load$ = É a quantidade de pulsos do *clock* utilizado para o cálculo da velocidade
- PPR = Pulsos por volta do *encoder* (Físico)
- $Edge$ = Depende da quantidade de bordas contadas por pulso, ou seja, se for utilizada apenas as bordas da fase A, o valor utilizado é 2, no caso de contar as bordas das duas fases, o valor é 4.

4.2.8. Configuração da UART (Transmissão/Recepção Universal Assíncrona)

A comunicação serial é o último passo do programa, no entanto a configuração desta comunicação é realizada nesta fase.

Como os outros recursos, é necessário configurar o periférico de transmissão e a porta que possui dupla função, além de configurar o pino da determinada porta de dupla função para a função de comunicação.

- SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
- SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
- GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

Na configuração é necessário informar as características da comunicação. Estas informações devem ser de acordo com o *host* (Receptor).

- UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200, (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE | UART_CONFIG_PAR_NONE));

Onde o primeiro campo indica qual dos dois blocos de comunicação será utilizado, o segundo é a frequência do clock que a comunicação deve se basear (pode ser a mesma que a frequência de operação). O terceiro campo é a velocidade de

transmissão desejada, esses valores devem seguir o padrão de comunicação assíncrona [22].

Por fim é habilitada a interrupção necessária para o envio das informações.

- IntEnable(INT_UART0);
- UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);

Onde o campo da primeira função e o primeiro campo da segunda representam a interrupção do transmissor 0. O segundo campo da função indica quais parâmetros que disparam a interrupção, para o caso de emissão os necessários são os indicados.

O envio das informações é realizado no bloco de comunicação serial, Seção 4.1.4.

4.3. Loop

Durante a fase de *loop* o programa fica parado, esperando o disparo das interrupções dos *timers*, onde o *timer* 0 é utilizado para leitura das informações (conversor analógico digital e do *encoder*) e o *timer* 1 é utilizado para geração dos sinais de acionamento (PWM e portas I/O). Os *timers* podem, e geralmente são, configurados com tempos diferentes. Estas interrupções ocorrem ciclicamente de acordo com a configuração dos *timers*.

A declaração das interrupções é realizada da seguinte maneira.

```
void
Timer0IntHandler(void)
{
//Leitura ou geração dos sinais
}
```

É importante relembrar que a *flag* de interrupção dos *timers* da Seção 4.1.2.3 devem ser zerados.

Os valores amostrados são armazenados em vetores para serem enviados após o fim da amostragem. O programa permanece no bloco de *Loop* até atingir o número de amostras predefinido no bloco de inicialização.

4.4. Comunicação serial

O bloco de comunicação serial foi utilizado para reduzir o tempo das medidas (conversor analógico digital e *encoder*), pois caso fosse enviado cada um desses valores a cada amostra, a taxa de aquisição seria afetada drasticamente já que a comunicação consome tempo de processamento, assim os valores são alocados em

um vetor e após todas as medidas terem sido realizadas é transmitido um pacote com as informações.

O comando de envio utilizado é:

- UARTSend(buf, 1);

Onde *buf* é a variável char onde a informação está armazenada. Como é necessário informar a quantidade caracteres enviados e o tamanho da informação é variável é realizada uma lógica de verificação da palavra enviada, ou seja, se a informação é menor que 10, a palavra enviada é de tamanho 1, se entre 10 e 99, o tamanho é 2, e assim por diante.

A comunicação em si é realizada por uma interrupção. Onde o programa fica em *loop* até que todos os caracteres tenham sido enviados. A única variável do comando é a mudança do bloco utilizado para comunicação (0 ou 1).

```
void
UARTSend(const unsigned char *pucBuffer, unsigned long ulCount)
{
    while(ulCount--)
    {
        UARTCharPut(UART0_BASE, *pucBuffer++);
    }
}
```

5. Matériais e métodos utilizados

A fase prática deste trabalho foi dividida em duas etapas, os testes do *kit* de desenvolvimento EKI LM3S8962, onde serão testados os periféricos necessários para o acionamento dos motores em questão e ensaios para validação do *kit* de desenvolvimento para a aplicação desejada, realizando testes de acionamento em um motor de corrente contínua de imã permanente e em um motor de indução.

A bancada ilustrando os recursos utilizados para o acionamento do motor de indução é mostrada na 17.

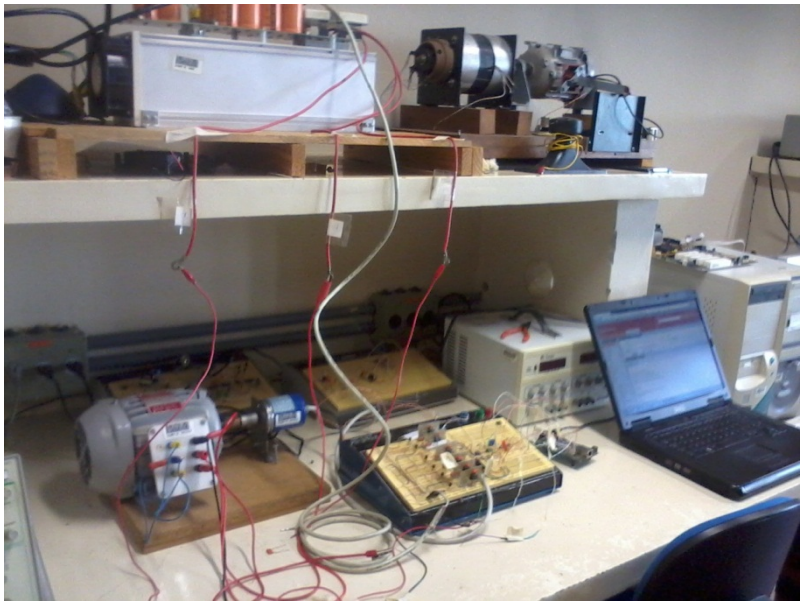


Figura 17 - Bancada de testes

5.1. Materiais utilizados

Os recursos utilizados foram:

- *Kit* de desenvolvimento EKI LM3S8962
- Motor de corrente contínua de imã permanente com tacogerador acoplado
- Motor de indução trifásico de 1CV com *encoder* acoplado
- Inversor de tensão trifásico da SEMIKRON
- Protoboard
- Computador
- Bateria de 12V
- Multímetro digital

- Tacômetro ótico

Alguns circuitos auxiliares foram utilizados na validação do *kit* de desenvolvimento realizada acionando os motores de corrente contínua e o de corrente alternada de indução. A utilidade destes circuitos foi proteção dos equipamentos (opto-acopladores), de acionamento (*Chopper*) e circuitos de medição de corrente e de velocidade.

5.2. Circuito de proteção

Visando proteção dos circuitos, principalmente do *kit* de desenvolvimento EKI LM3S8962, são utilizados circuitos opto-acopladores, que transmitem a informação via luz (LEDs), garantindo isolamento total entre os circuitos de potência e de acionamento.

Estes circuitos consistem em um emissor de luz e um receptor, a informação proveniente dos sinais do *kit* (PWM ou portas I/O) aciona o emissor de luz, esta luz é recebida pelo receptor, transmitindo assim a informação para o circuito de potência.

O opto-acoplador utilizado foi o 6N137 [23] da Figura 18 na configuração mostrada na Figura 19.

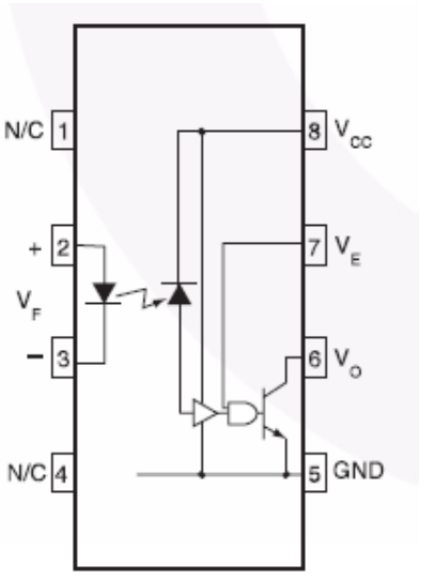


Figura 18 - Esquemático do componente 6N137 [23]

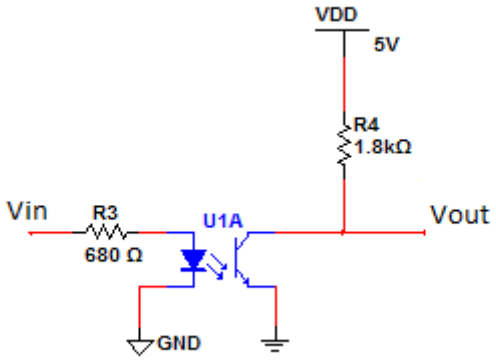


Figura 19 - Circuito opto-acoplador utilizado

O circuito da Figura 19 sofre inversão lógica, ou seja, quando o sinal da entrada está em nível lógico alto, a saída está em nível lógico baixo. No entanto esta inversão é corrigida no circuito emissor comum mostrado na Figura 20 que além de corrigir a inversão do sinal de acionamento, aumenta o nível de tensão de 5V para 15V. Este nível de tensão (15V) é necessário para operar os circuitos de potência, tanto o MOSFET IRF540N [24] utilizado no circuito *chopper* quanto as chaves do inversor de tensão trifásico utilizado precisam ser operados com 15V.

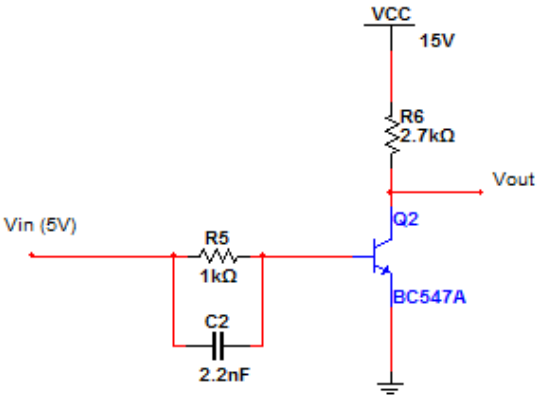


Figura 20 - Circuito elevador de tensão

Dessa forma, o circuito auxiliar para acionamento dos motores (tanto para o motor de corrente contínua quanto para o de corrente alternada) é representado pela Figura 21 e foi implementado em um protoboard para a realização dos testes. Onde no caso do motor de corrente contínua, Vout alimenta o MOSFET da Figura 22, e no caso do acionamento do motor de indução, Vout alimenta as chaves do inversor da Figura 4.

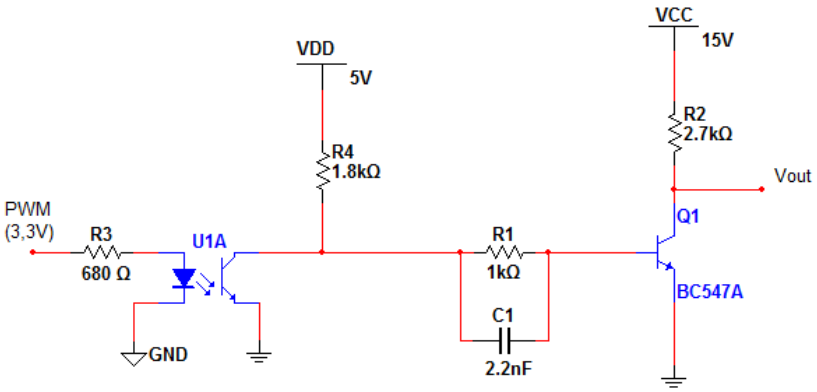


Figura 21 - Circuito de acionamento das chaves de potência

5.3. Circuito *chopper* utilizado

O circuito chopper utilizado é mostrado na Figura 22 e consiste em uma bateria automotiva de 12V, alimentando o motor através de um MOSFET de potência IRF540N, e um diodo de potência rápido MUR1520 [25] que serve de proteção à chave semicondutora e possui um tempo de recuperação de no máximo 60 ηs. O MOSFET permite alimentação de até 33ª e 100V, entre o terminal de dreno(*drain*) e o de fonte (*source*). Já o diodo rápido tem um limite de alimentação de até 600V e 15A, e a escolha deles foi para validar o acionamento de motores de corrente contínua que operam em níveis de tensão e de corrente mais elevados do que o motor utilizado.

A alimentação da chave semicondutora, o terminal In(15V) da Figura 22, é controlado pelo circuito de acionamento da Figura 21.

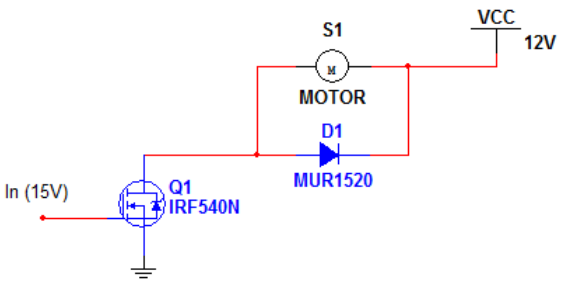


Figura 22 - Circuito utilizado – *Chopper*

5.4. Circuitos de medição

A realização de medições durante o acionamento fornece informações importantes além de possibilitar a implementação de uma lógica de controle. Para o caso das máquinas elétricas duas informações são extremamente importantes, a corrente que flui pelo motor e a velocidade de rotação do eixo do motor.

5.4.1. Medição de corrente

No caso da corrente foi utilizado um sensor de corrente do tipo *Hall*, sensor que é baseado no efeito *Hall* [26] e fornece um valor em tensão proporcionalmente à corrente que flui por ele.

O sensor de corrente do tipo *Hall* utilizado foi o sensor ACS712 [27] de 5 Amperes, que fornece uma tensão conforme a Figura 23, sendo 2.5V para 0A e variando 0,185V por Ampere.

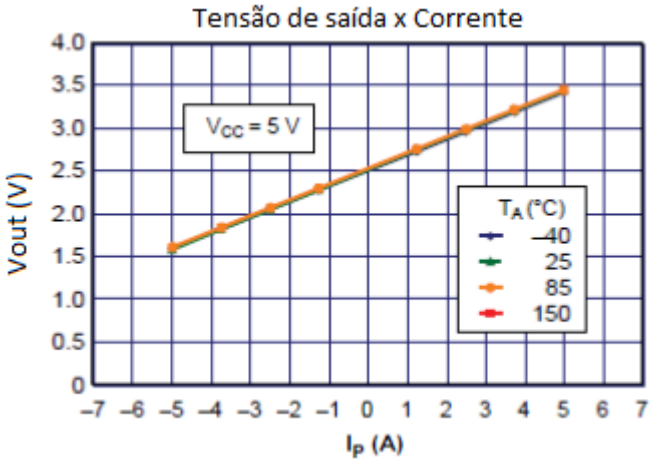


Figura 23 - Comportamento do sensor de corrente do tipo *Hall* [27]

É necessário um circuito que reduza essa faixa de operação para que seja compatível com o conversor analógico digital do *kit* de desenvolvimento EKI LM3S8962 que é de 0 a 3V. Sendo assim é possível reduzir esta faixa com a utilização de amplificadores operacionais, no caso LM324 [28], na configuração inversora. A Figura 24 ilustra a configuração utilizada. Onde Vin é o sinal do sensor de corrente do tipo *Hall* e Vout é conectado ao CAD do *kit* de desenvolvimento. Dessa forma a faixa de tensão varia de 0 a 3V, sendo 1,5V para 0A.

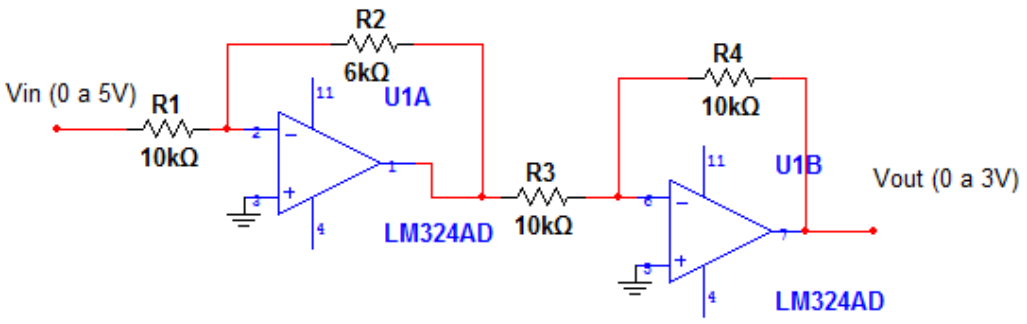


Figura 24 - Circuito auxiliar do sensor *hall*

5.4.2. Medição de velocidade

As medições de velocidade do eixo do motor foram realizadas de duas maneiras distintas. No caso do motor de corrente contínua foi utilizado um tacogerador, já no caso do motor de corrente alternada foi utilizado um *encoder* óptico. Essas diferentes soluções ocorreram pelo fato dos motores utilizados possuírem estes transdutores já acoplados aos seus eixos.

Medição via tacogerador

O tacogerador é um gerador CC de ímã permanente acoplado mecanicamente ao eixo de um motor que se deseja mediar velocidade, já que é gerada uma tensão na saída do tacogerador em função da velocidade de rotação do motor.

O tacogerador utilizado opera até 50V, ou seja, com 12V na entrada do motor é gerado um sinal (em corrente contínua) de 50V. Desse modo, foi utilizado um divisor resistivo de tensão para reduzir a faixa de tensão para 0 a 3V, realizando assim a leitura por uma entrada do conversor analógico digital. A Figura 25 representa o divisor utilizado.

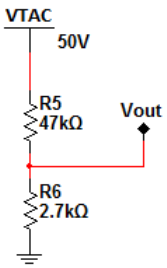


Figura 25 - Divisor de tensão resistivo

Medição via encoder

O *encoder* utilizado gera os pulsos com um valor de nível lógico alto em 15V não sendo compatível com o módulo QEI (*Quadrature Encoder Interface*) presente no *kit* de desenvolvimento EKI M3S8962, que é de 0 a 5V, dessa forma foi projetado um circuito utilizando opto acopladores para reduzir os sinais de entrada no *kit* de desenvolvimento. Foi utilizado o opto-acoplador 6N137 da Figura 18 na configuração da Figura 26.

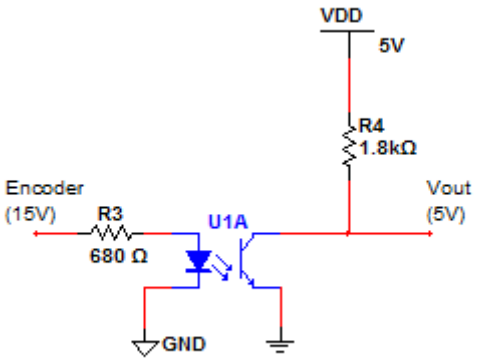


Figura 26 - Circuito abaixador de tensão para o *encoder*

6. Testes e ensaios realizados

Os testes foram realizados em duas etapas. A primeira foi a realização dos testes nos componentes do *kit* de desenvolvimento EKI LM3S8962. A segunda fase foi a validação do *kit* de desenvolvimento para o acionamento do motor de corrente contínua de imã permanente e o de corrente alternada de indução trifásica.

6.1. Testes no *kit* de desenvolvimento EKI LM3S8962

Os ensaios realizados no *kit* de desenvolvimento tem o objetivo de compreender os comandos utilizados e encontrar os limites de operação de cada periférico. São testados os periféricos utilizados no acionamento dos motores estudados, módulo PWM e portas I/O, os periféricos utilizados para aquisição de dados, CAD e módulo do *encoder*, além de recursos auxiliares, *display* OLED e módulo serial, que auxiliam o usuário do *kit* de desenvolvimento.

6.1.1. Teste display OLED

O primeiro teste realizado foi o teste do *display* OLED, baseado no exemplo presente no programa *IAR Embedded Workbench,* com o objetivo de compreender a configuração do mesmo.

O teste consiste em programar uma mensagem (“TESTE”) para aparecer na tela do *display*. A Figura 27 apresenta o resultado obtido.

Figura 27 - Teste no *display* OLED

6.1.2. Teste no módulo CAD

O teste no módulo CAD tem como objetivo verificar o comportamento do conversor e encontrar a precisão do sistema de aquisição. Utilizando o circuito da Figura 28 foram aplicados diferentes níveis de tensão na entrada do CAD através de

um potenciômetro, comparando o valor amostrado pelo CAD, e impresso no *display*, com o valor medido por um voltímetro. O resistor em questão utilizado foi um potenciômetro de 1kΩ.

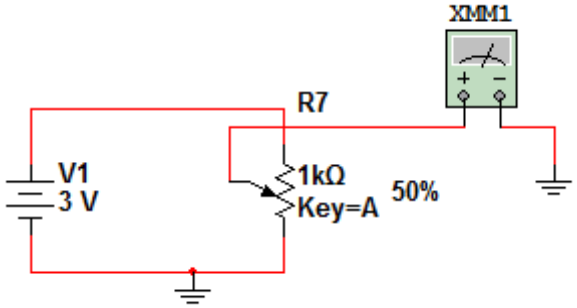


Figura 28 - Teste do CAD

A Tabela 16 apresenta os resultados obtidos. A primeira coluna representa a tensão obtida pelo multímetro, a segunda coluna representa o número decimal fornecido pelo CAD e impresso no *display* OLED. A terceira coluna é o resultado da conversão dos valores da coluna anterior utilizando a Equação 5. Por fim, a quarta coluna representa o erro comparativo entre a primeira e terceira colunas.

Tabela 16 - Resultado do teste CAD

Tensão (multímetro)	Valor ADC (0 a 1024)	Conversão (V)	Erro (%)
0,00	2	0,01	-
0,25	86	0,25	0,00
0,50	172	0,50	0,00
0,75	257	0,75	0,00
1,00	338	0,99	0,01
1,25	425	1,25	0,00
1,50	510	1,49	0,01
1,75	592	1,73	0,01
2,00	675	1,98	0,01
2,25	764	2,24	0,00
2,50	846	2,48	0,01
2,75	933	2,73	0,01
3,00	1022	2,99	0,00

$$V = \frac{V_{max}}{2^n} * Valor_{ADC} \quad (5)$$

Onde:

- V_{max} = Tensão máxima de entrada = 3V
- n = Número de bits do conversor = 10
- $Valor_{ADC}$= Valor que o conversor fornece.

A precisão do CAD é satisfatória visto a pequena porcentagem de erro mostrada na última coluna da Tabela 16.

6.1.3. Ensaios com a porta serial

Com o objetivo de testar a comunicação serial, o envio de informações para o computador, foi realizado um ensaio com o envio de uma mensagem “TESTE” via serial e receber a informação salvando-a em um arquivo de texto.

O arquivo recebido pelo computador (host) via porta USB é mostrado na Figura 29.

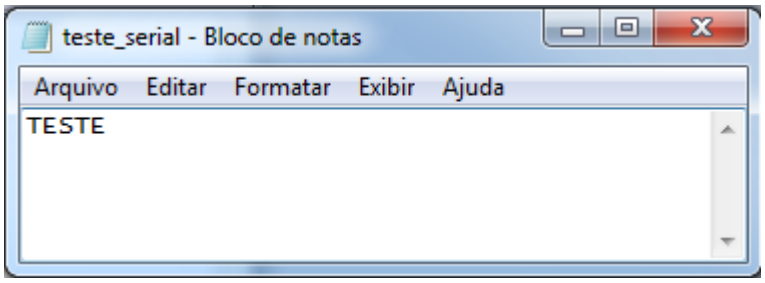


Figura 29 - Teste serial

6.1.4. Teste da porta serial com envio amostras do CAD

O primeiro teste envolvendo dois periféricos em conjunto foi a utilização da comunicação serial para o envio das informações lidas e armazenadas pelo conversor analógico digital (CAD). O código foi estruturado de forma a ler a informação a cada interrupção do *timer*, a cada 0.5 segundos, armazenar esta informação em um vetor e ao fim do programa, que termina após um número de amostras pré-determinado (10 amostras), enviando as amostras via serial.

Os sinais utilizados para realizar a conversão são os sinais de 3,3V e 0V provenientes da própria placa de desenvolvimento EKI LM3S862.

Utilizando o programa computacional *Matlab* é possível gerar o gráfico a partir das amostras recebidas e armazenadas em um arquivo de texto. A Figura 30 compreende o resultado deste gráfico.

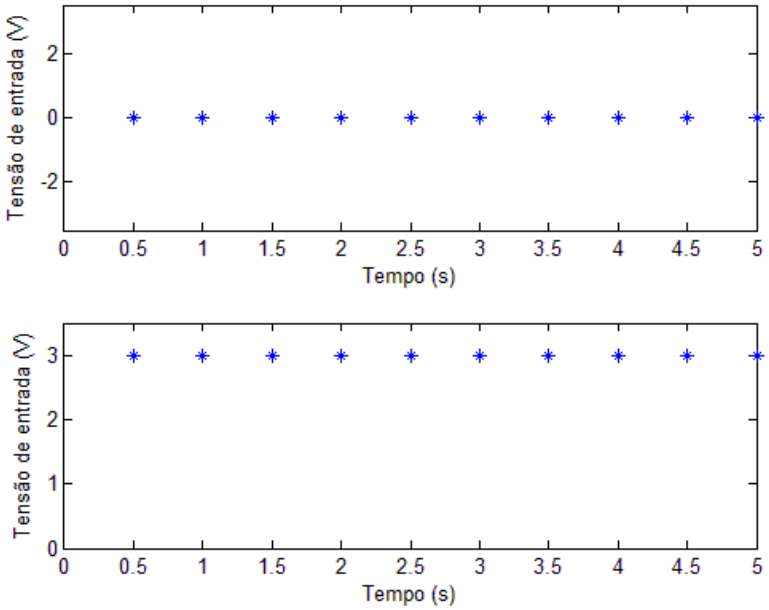


Figura 30 - Resultado do teste CAD

6.1.5. Ensaios do módulo PWM

Com o objetivo de compreender a programação do periférico PWM do *kit* de desenvolvimento EKI LM3S8962, foram realizados ensaios a partir do exemplo do programa *IAR Embedded Workbench*, onde é configurado um pulso de tamanho fixo.

A princípio foi gerado este pulso fixo, mas como o objetivo é criar um acionamento para motores, foi gerado um perfil como da Figura 31, onde o tamanho do pulso aumenta ou diminui de acordo com o tempo. O eixo “perfil (%)” representa a porcentagem do pulso em nível lógico alto.

O perfil da Figura 31 foi gerado e observado por um osciloscópio. O resultado obtido foi satisfatório, sendo observado o perfil proposto, a largura do pulso crescendo, se mantendo em 100% e diminuindo novamente.

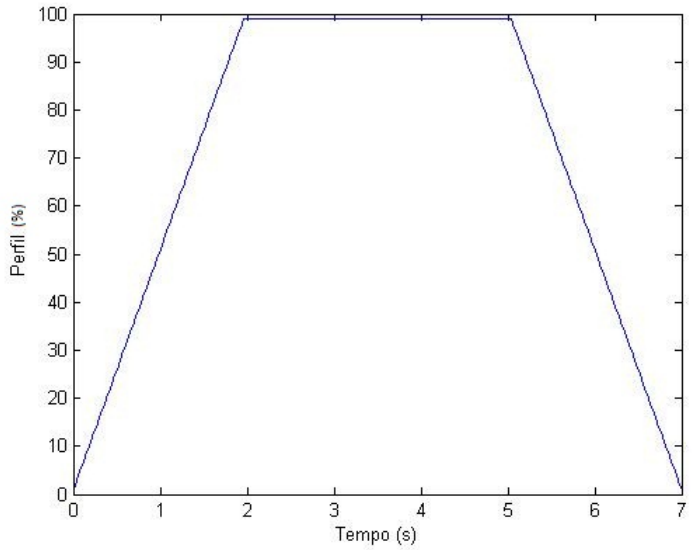


Figura 31 - Perfil para o PWM

6.1.6. Teste da interface de encoder

Os ensaios no *encoder* têm como objetivo analisar o comportamento da ferramenta QEI (*Quadrature Encoder Interface*) do *kit* de desenvolvimento EKI LM3S8962. Para isto se deve acionar o motor de indução trifásico que possui *encoder* acoplado a ele, através de um *variac* ajustado com 80Vpp, e comparar o valor amostrado pelo QEI, impresso no *display* OLED, comparando ao valor obtido por um tacômetro ótico.

O teste na interface do *encoder* foi realizado utilizando o *encoder* da Figura 32 acoplado ao eixo motor e o circuito auxiliar da Figura 26 para conectar o sinal às entradas QEI do *kit* de desenvolvimento afim de comparar o valor lido pelo módulo QEI com o valor medido utilizando um tacômetro ótico. Para o cálculo da velocidade em rotações por minuto foi utilizada a Equação 4, que para o caso utilizado resulta no valor de 1,6. A Tabela 17 é o resultado do teste, onde a primeira coluna representa o valor lido pelo tacômetro, a segunda coluna mostra o valor obtido pelo módulo QEI já aplicando o fator de 1,6 e a última coluna representa o erro percentual comparando as duas medições.

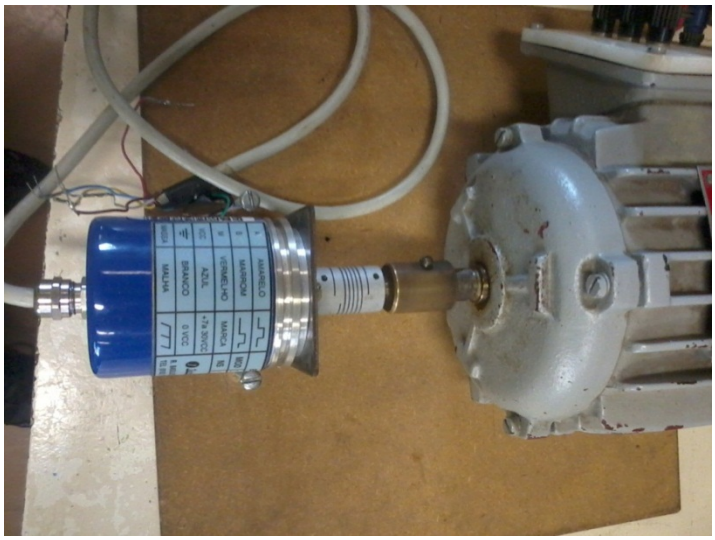


Figura 32 - *Encoder* utilizado acoplado ao motor de indução

Tabela 17 - Teste QEI

Valor Tacômetro (RPM)	Valor QEI (RPM)	Erro (%)
3581	3577,60	0,1%

6.1.7. Ensaio da constante do tacogerador

Os ensaios envolvendo o tacogerador foram realizados durante os testes de acionamento do motor corrente contínua. O objetivo dos testes é encontrar a constante linear do tacogerador que, multiplicado pelo valor lido no CAD, fornece a velocidade real do motor.

Para encontrar o coeficiente linear deste tacogerador foi realizado um ensaio acionando o motor com diferentes velocidades, a partir do acionamento via PWM, utilizando o circuito da Figura 21 para proteção do *kit* de desenvolvimento e o circuito, de potência da Figura 22, realizar medições com um tacômetro óptico e compará-los com os valores obtidos no CAD impressos no *display*.

A constante K pode ser obtida pela divisão do valor medido no tacômetro ótico pelo valor amostrado pelo conversor analógico digital. Com a realização de 5 medidas com velocidades diferentes é possível encontrar o valor da constante K médio.

A Tabela 18 é o resultado deste ensaio com as velocidades referentes à largura de pulso do PWM referenciadas na primeira coluna, como porcentagem do pulso em nível lógico alto, estas medições foram realizadas a partir de 60% pois abaixo desta porcentagem, a medição pelo tacômetro se tornou imprecisa, já que os valores indicados não eram constantes. A segunda coluna representa a velocidade obtida

pelo tacômetro ótico. A terceira representa o valor amostrado pelo CAD e a quarta coluna o resultado da divisão da segunda coluna pela terceira. Realizando a média dos valores obtidos para constante K, foi obtido o valor K médio de 3,37, utilizado como constante para obter a quinta coluna da Tabela 18. A última coluna mostra o erro percentual do valor utilizando a constante K encontrada em comparação às medições realizadas com o tacômetro ótico.

Tabela 18 - Cálculo da constante K do tacogerador

Medição	Tacômetro (RPM)	CAD	K	Convertido (RPM)	Erro (%)
60%	571	171	3,34	577,04	1,05
70%	1200	354	3,39	1194,58	0,45
80%	1627	477	3,42	1609,64	1,08
90%	1992	593	3,36	2001,09	0,45
100%	2493	739	3,37	2493,77	0,03

6.1.8. Tempo gasto pelas funções de aquisição de dados

Todas as operações em um microprocessador gastam um determinado tempo que varia com a complexidade da operação e com a frequência de trabalho, por exemplo, uma multiplicação de um número real demora mais tempo que a soma de um número inteiro. Deste modo é realizado um ensaio com o objetivo de encontrar o tempo gasto pelas funções de aquisição e armazenagem dos dados. Com esta informação é possível escolher uma taxa adequada para uma possível lógica de controle.

O ensaio é realizado com base na estrutura em que o código foi montado. Como o programa foi estruturado de forma a realizar as aquisições dentro da interrupção de um *timer*, é possível medir o tempo gasto dentro do *timer* acionando uma saída apenas durante o período da interrupção. Utilizando um osciloscópio é possível medir o tempo em que a saída permanece em nível lógico alto.

Os testes consistem em encontrar o tempo gasto em 3 casos. O primeiro amostrando os valores de um CAD e do *encoder*, imprimindo estes valores no *display* OLED. Em seguida foi realizado o teste apenas amostrando os valores do CAD e do *encoder*. Por fim é realizado o ensaio para descobrir o tempo gasto apenas da função de aquisição do CAD.

O primeiro teste foi realizado com as informações do CAD e do *encoder* e imprimindo estas informações no *display* OLED. A Figura 33 mostra o valor medido pelo osciloscópio.

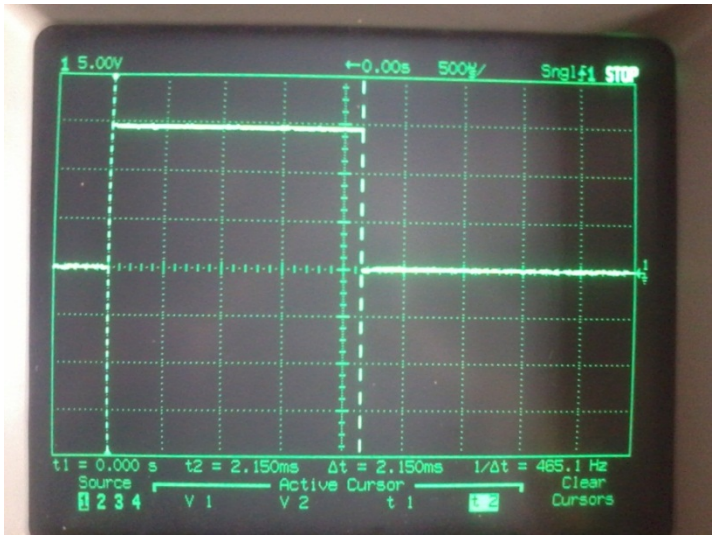


Figura 33 - Teste da taxa máxima de aquisição – *Display*

Como era de se esperar o tempo gasto, 2,150ms é muito alto já que as funções do *display* OLED são lentas. Por este motivo, para frequências de aquisições altas, a função de impressão no *display* não deve ser utilizada.

O segundo ensaio foi realizado utilizando o CAD em conjunto com o *encoder* obtendo o resultado mostrado na Figura 34.

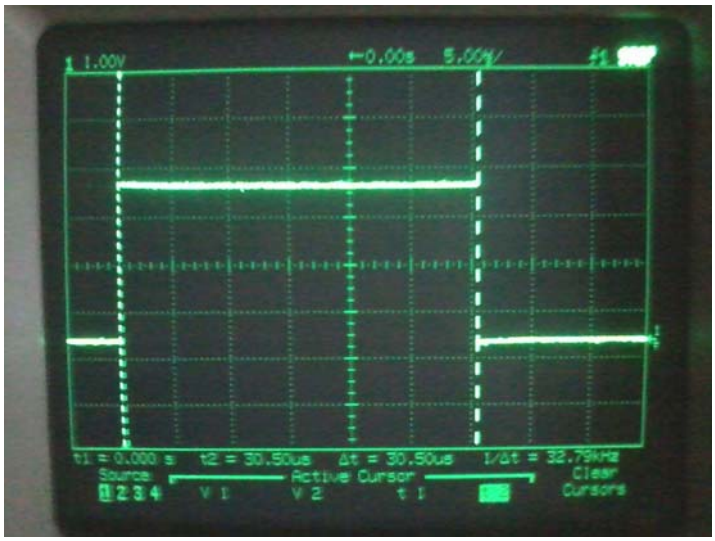


Figura 34 - Taxa máxima de aquisição - CAD e QEI

O tempo gasto para estas duas funções é de 30,5µs.

Por fim, utilizando apenas o conversor analógico digital (CAD) obtém-se a Figura 35.

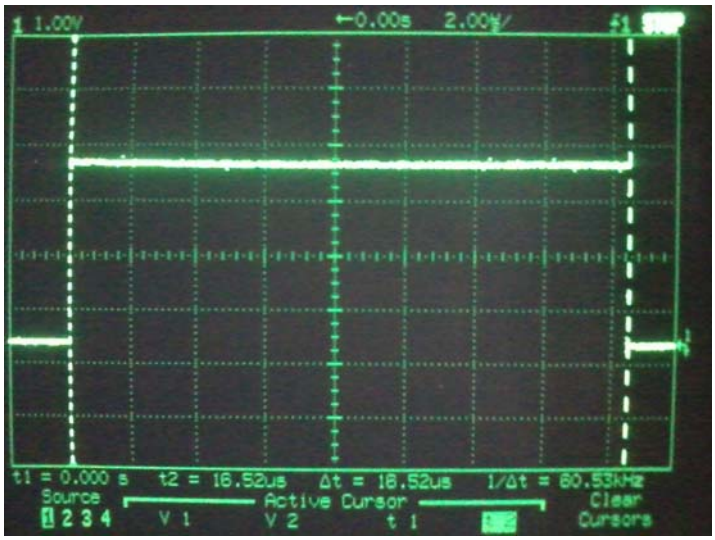


Figura 35 - Taxa máxima de aquisição – CAD

Como era de se esperar o tempo gasto, 16,52 µs, por esta função é menor do que nos outros ensaios.

É possível ainda extrair o tempo da função de aquisição de dados do *encoder* comparando os resultados dos ensaios realizados. Realizando a subtração do tempo gasto pela função do conversor analógico digital em conjunto com o *encoder*, pelo tempo obtido no ensaio utilizando apenas o CAD, é obtido o tempo gasto apenas pelo *encoder*, que é 13,89 µs.

6.1.9. Número máximo de amostras

A comunicação do *kit* de desenvolvimento é realizada via porta serial, no entanto, para não diminuir a taxa de amostragem, a transmissão serial inicia após todas as leituras, deste modo as leituras são armazenadas na memória do *kit* de desenvolvimento. O objetivo deste ensaio é encontrar o valor máximo disponível para armazenamento das amostrar.

O ensaio visa encontrar o máximo valor possível para o tamanho dos vetores utilizados para o armazenamento. Na fase de compilação do programa é possível variar o tamanho destes vetores. Caso o vetor seja maior que a memória disponível, ocorre uma mensagem de erro. O ensaio consiste em encontrar o maior número possível deste vetor.

A Tabela 19 apresenta os resultados obtidos. A primeira coluna representa os valores testados para um vetor e a coluna de resultados indica se o ensaio foi satisfatório ou não, o resultado OK representa que o programa compilou corretamente, já a palavra Erro representa uma falha neste ponto do programa por causa do estouro da memória disponível.

Observando a Tabela 19 é possível considerar um número máximo de 16000 amostras, ou seja, é possível armazenar 16000 amostras de um vetor (CAD, *encoder*, etc.), caso sejam 4 amostras o número cai para 4000 amostras.

Tabela 19 - Teste do número máximo de amostras

Tamanho do Vetor	Resultado
3000	OK
10000	OK
15000	OK
20000	Erro
17500	Erro
16000	Ok
17000	Erro
16500	Erro

6.2. Ensaios nos motores em estudo

Após os testes realizados no *kit* de desenvolvimento EKILM3S8962 é possível integrá-los com o objetivo de acionar os motores, realizar medições, comunicar com o computador e plotar estas medidas. São realizados ensaios com o motor de corrente contínua com imã permanente e com um motor de indução

6.2.1. Ensaio no motor de corrente contínua

O motor de corrente contínua de imã permanente da Figura 36 é acionado de duas maneiras. Diretamente de uma bateria automotiva de 12V e utilizando o *chopper* da Figura 22 em conjunto com circuito o da Figura 21, aplicando o perfil do PWM da Figura 31. Para validar os conversores analógicos digitais são amostrados os valores da corrente, gerada pelo sensor de corrente do tipo *Hall* condicionado utilizando o circuito da Figura 24, e de velocidade, gerado pelo tacogerador acoplado ao motor de corrente contínua com o tratamento mostrado na Figura 25. Estes valores são enviados via serial para serem impressos via *Matlab*.

Figura 36 - Motor utilizado

6.2.1.1. Acionamento do motor CC diretamente a uma bateria

A conexão direta da bateria automotiva de 12V ao motor fornece os resultados da velocidade por meio do tacogerador acoplado ao motor, Figura 37, e da corrente, Figura 38, utilizando o sensor de corrente do tipo *Hall*.

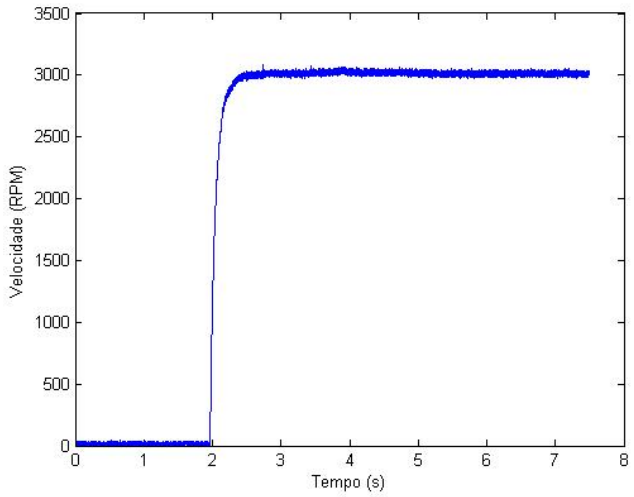


Figura 37 - Motor CC - Partida direta – Velocidade

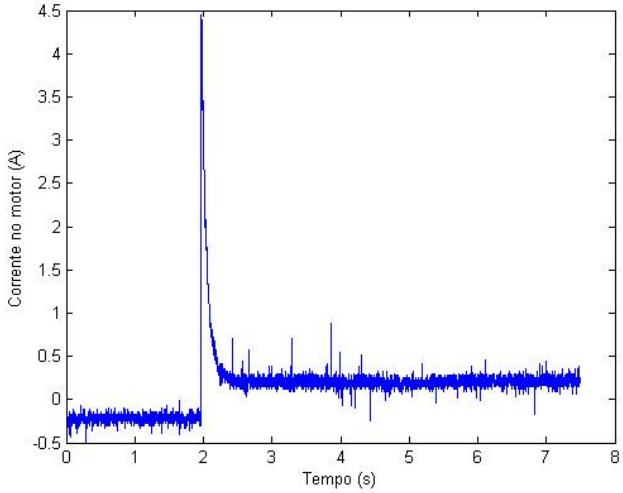


Figura 38 - Motor CC - Partida direta – Corrente

A conexão do motor à bateria e o acionamento do *kit* de desenvolvimento par realização das medições foram realizadas manualmente. O tempo de 2 segundos compreende o tempo gasto entre as duas operações manuais.

6.2.1.2. Acionamento do motor CC utilizando PWM

Utilizando o circuito de acionamento para o motor de corrente contínua (Figura 21 e Figura 22), foi aplicado um sinal PWM com o perfil da Figura 31. O resultado obtido para velocidade na Figura 39. O resultado da corrente é observado na Figura 40.

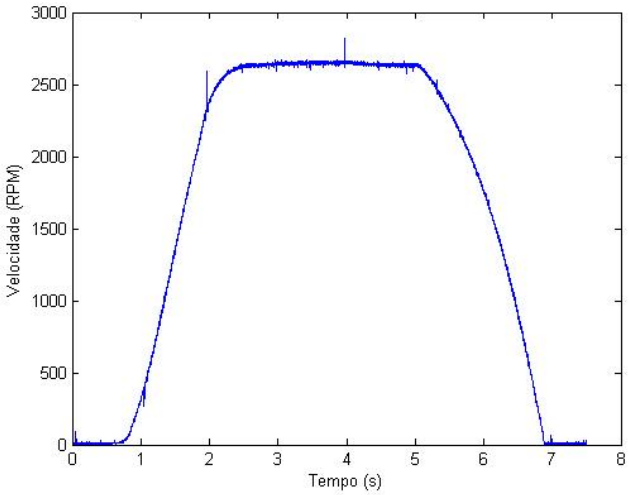


Figura 39 - Motor CC - Partida PWM – Velocidade

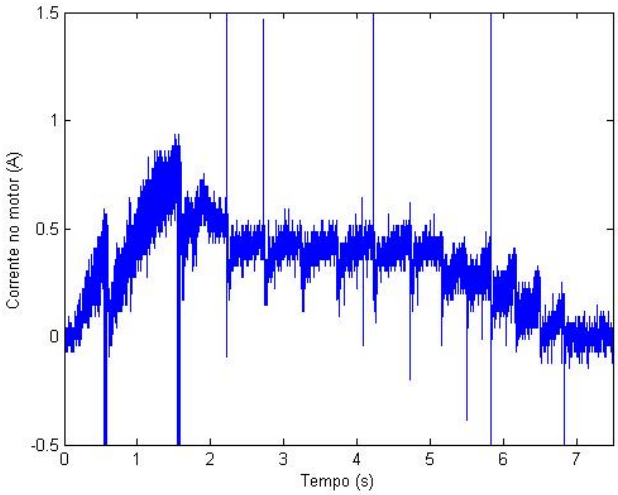


Figura 40 - Motor CC - Partida PWM – Corrente

6.2.2. Ensaio no motor de indução trifásico

De modo semelhante ao motor de corrente contínua foram realizados dois diferentes acionamentos no motor de indução. A partida diretamente da rede trifásica por meio de um *variac* ajustado para 80V de pico a pico, e a partida utilizando o modo 6-pulsos (modos 2 a 2 e 3 a 3). O valor de corrente de uma das fases do motor deve ser amostrado pelo CAD, por meio do sensor de corrente do tipo *Hall*, transmitido via serial e impressos via *Matlab*.

A partida utilizando o modo 6-pulsos aciona o inversor da Figura 41 utilizando as sequências da Figura 5, para o modo 2 a 2, e da Figura 6, para o modo 3 a 3, com frequências de rotação de 0,33Hz e 60Hz.

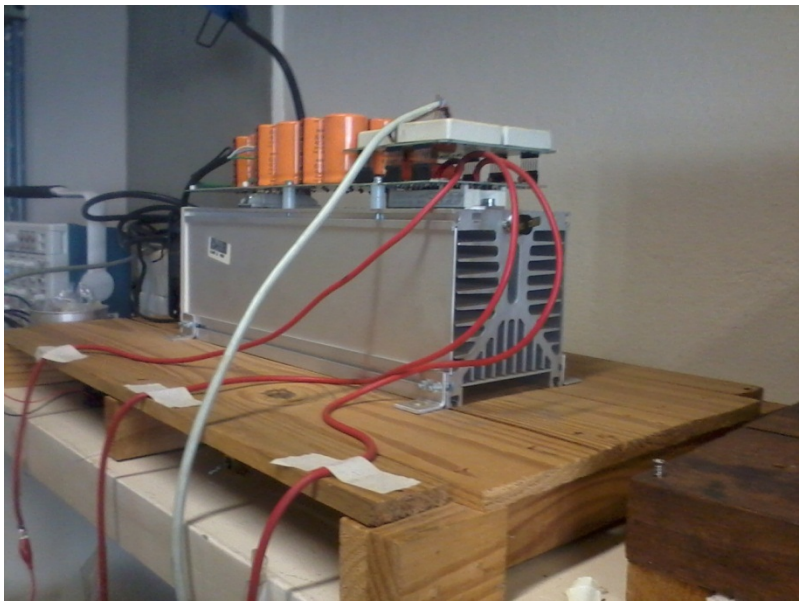


Figura 41 - Inversor trifásico utilizado

O inversor utilizado é um inversor da SEMIKRON constituído de um circuito retificador de tensão SKD51/52 [29] e um filtro para alimentação do barramento de corrente contínua, de semicondutores IGBTs SKM40GDL123D [30] e módulos de acionamentos para o Gate dos IGBTs SKIH22 [31], que tem como objetivo tratar o sinal para o acionamento e prover proteção aos semicondutores. O filtro é formado por um banco de capacitores que juntos possuem o valor de 8,16mF e suportam até 400V de alimentação.

6.2.2.1. Partida direta

Conectando o motor de indução, por meio de um disjuntor trifásico, a um *variac* ajustado para 80Vpp foram realizadas medições de corrente em uma das fases utilizando um sensor de corrente do tipo *Hall* e de velocidade de rotação do eixo do motor. O valor lido pelo sensor de corrente do tipo *Hall* foi amostrado com o osciloscópio. A Figura 42 mostra a o valor da corrente em uma das fases do motor de indução obtidos com um osciloscópio. Estes valores de corrente foram amostrados pelo CAD e transmitidos via comunicação serial. Utilizando o *software Matlab* para plotar estes dados em um gráfico foi gerada a Figura 43.

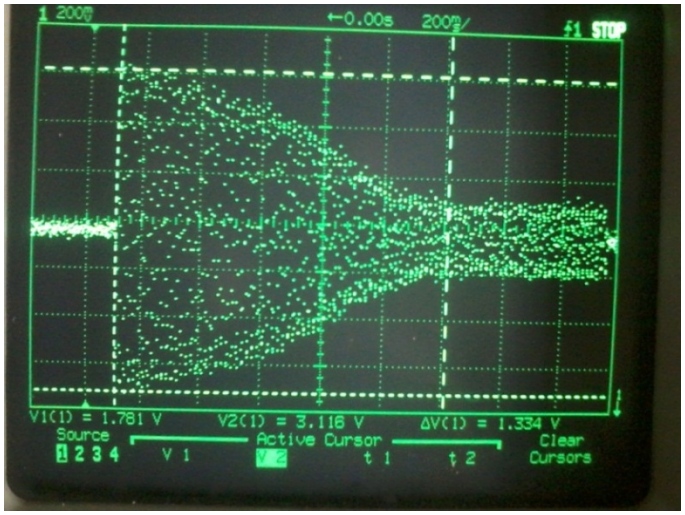


Figura 42 - Motor de indução - Partida direta – Corrente

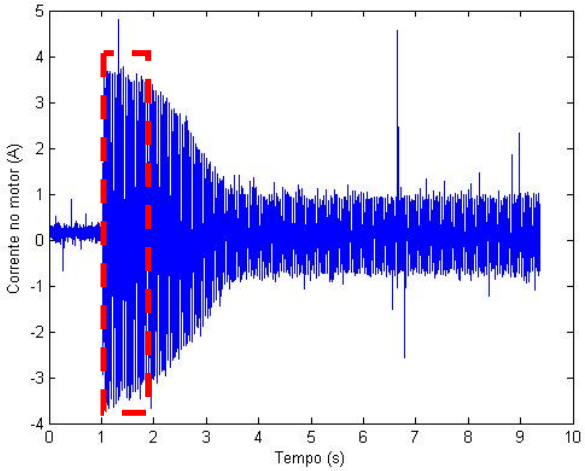


Figura 43 - Motor de indução - Parida direta – Corrente

Na Figura 43 foi aplicado um zoom do entre o 1º e 2º segundo para verificar se o comportamento está dentro do esperado (senoidal). A Figura 44 mostra que o resultado é satisfatório.

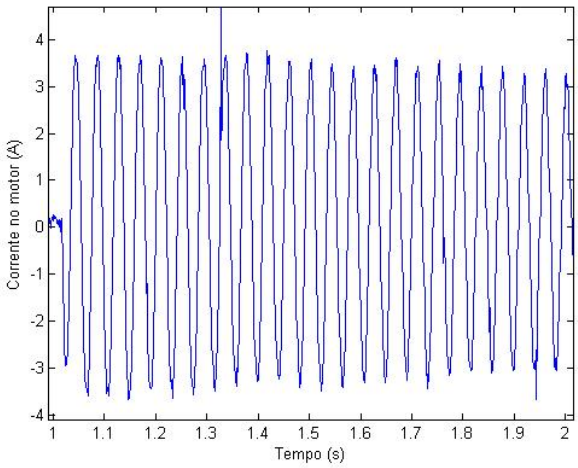


Figura 44 - Motor CA - Comportamento da corrente

O valor amostrado correspondente à velocidade é exposto na Figura 45.

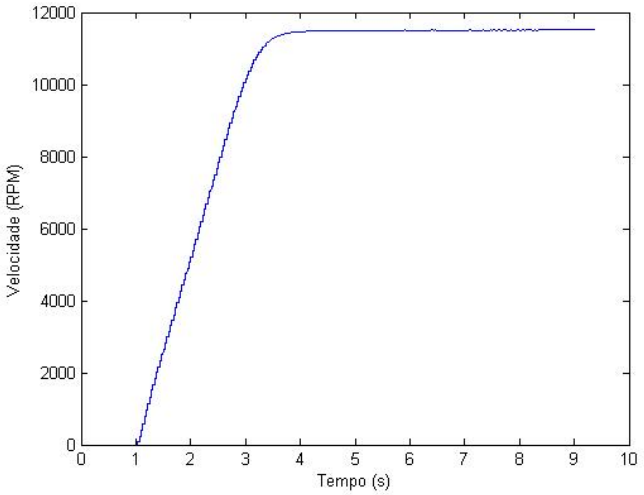


Figura 45 - Motor CA - Parida direta – Velocidade

Os disparos, tanto do programa quanto dos motores, foram realizados manualmente. O tempo de 0 a 1s representam este tempo gasto até o acionamento do motor de indução.

6.2.2.2. Acionamento através do *kit* no modo 6-pulsos

A segunda etapa de teste no motor de indução foi o acionamento do mesmo utilizando o inversor da Figura 41 no modo 6-pulsos, tanto nos modos 2 a 2 quanto nos modos 3 a 3.

O chaveamento do inversor trifásico deve ser realizado em todas as chaves simultaneamente, de modo a garantir que as chaves de um mesmo braço não conduzam ao mesmo tempo. Desta forma o programa utilizado no microprocessador ARM Cortex-M3 gera os sinais ao mesmo tempo, utilizando a porta A do *kit* de desenvolvimento EKI LM3S8962 para gerar uma palavra de 8 bits onde 2 não são utilizados.

Visando a proteção do inversor, a tensão de alimentação do inversor utilizada não foi a tensão nominal no motor, visando manter a corrente do motor dentro dos valores nominais dele. Foi utilizado 50V para baixas frequências (0,33Hz) e 115V para as altas (60Hz).

6.2.2.2.1. Acionamento no modo 2 a 2

Seguindo a lógica de acionamento da Figura 5 foi gerada a Tabela 20 para a geração da palavra a ser gerada. Os bits menos significativos não são utilizados para o acionamento. A primeira coluna representa os 6 pulsos possíveis. A segunda representa quais chaves, da Figura 4, devem estar acionadas, no respectivo pulso. A terceira coluna representa como os pinos da porta A devem estar configurados para acionar as determinadas chaves. Por fim a ultima coluna representa o valor decimal que se deve escrever à porta.

Tabela 20 - Modo 2 a 2 - Palavras geradas

Pulso	Saídas	Binário	Decimal
0	1,2	10000100	132
1	2,3	10001000	136
2	3,4	00101000	40
3	4,5	00110000	48
4	5,6	01010000	80
5	6,1	01000100	68

Deste modo, a cada intervalo de tempo os valores na saída da porta A variam de acordo com a Tabela 20, acionando 2 chaves. Os tempos de acionamentos para cada pulso foram 0,5s e 1/360s, que representam 0,33 e 60Hz, respectivamente, de rotação do eixo do motor de indução.

As grandezas amostradas foram a corrente que passa por uma das fases do motor de indução e a tensão fase-neutro que alimenta o motor. Os valores foram amostrados pelo osciloscópio.

Frequência do motor em 0,33Hz

A Figura 46 mostra o valor amostrado pelo sensor de corrente do tipo *Hall* correspondente a corrente de uma fase do motor.

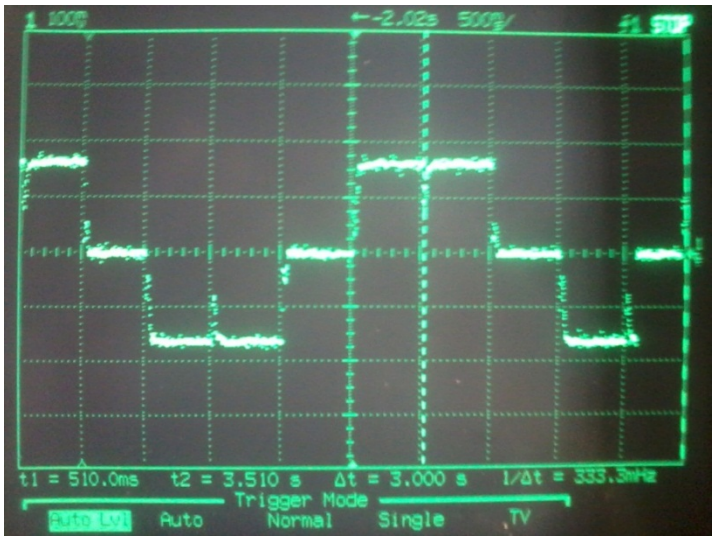


Figura 46 - Modo 2 a 2 – 0,33Hz – Corrente do motor de indução

A Figura 47 mostra o valor de tensão aplicada aos terminais do motor. No caso a tensão que alimenta o motor foi de aproximadamente 50V.

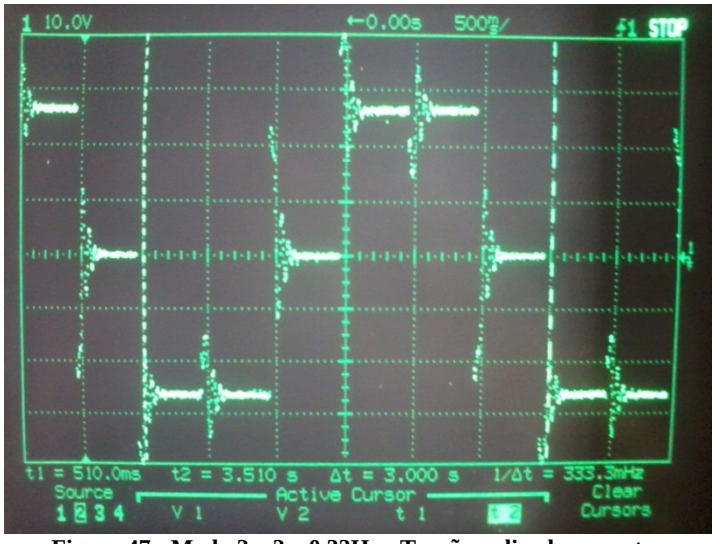


Figura 47 - Modo 2 a 2 – 0,33Hz - Tensão aplicada ao motor

Frequência do motor em 60Hz

Aumentando a frequência de rotação do motor para 60Hz são obtidos os resultados amostrados na Figura 48 para a corrente de uma das fases do motor e na Figura 49 para a tensão de alimentação do motor de indução. No caso foi utilizado aproximadamente 115V.

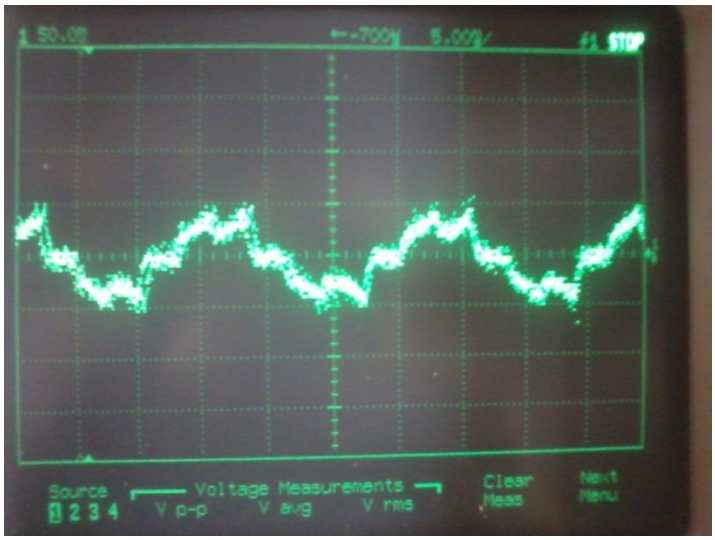


Figura 48 - Modo 2 a 2 – 60Hz – Corrente no motor

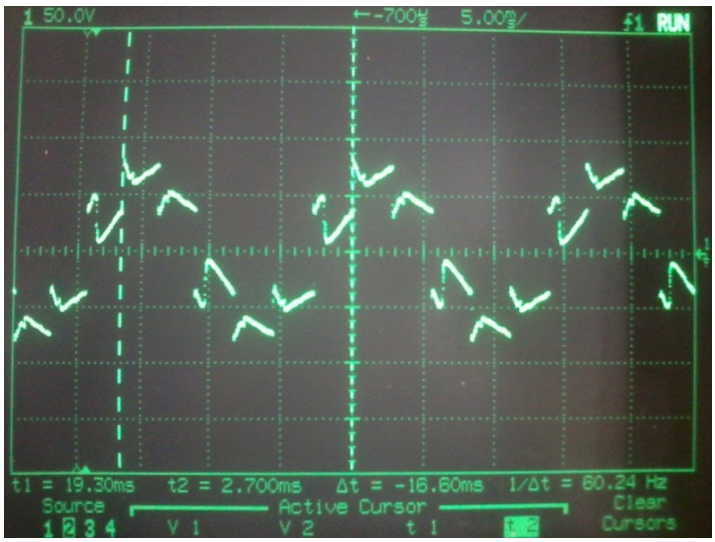


Figura 49 - Modo 2 a 2 – 60Hz – Tensão aplicada ao motor

6.2.2.2.2. Acionamento no modo 3 a 3

Analogamente ao modo 2 a 2, foi gerada a Tabela 21 com os valores das palavras utilizadas conforme a Figura 6.

Tabela 21 - Modo 3 a 3 - Palavras geradas

Pulso	Saídas	Binário	Decimal
0	4,5,6	01110000	112
1	5,6,1	01010100	84
2	6,1,2	11000100	196
3	1,2,3	10001100	140
4	2,3,4	10101000	168
5	3,4,5	00111000	56

O acionamento foi realizado para as mesmas frequências de acionamento do modo 2 a 2, 0,33Hz e 60Hz.

Frequência do motor em 0,33Hz

Utilizando 0,5 segundos de duração em cada pulso é obtida a frequência de rotação do eixo do motor de 0,33Hz. A Figura 50 apresenta a corrente em uma das fases do motor e a Figura 51 a tensão fase-neutro aplicada aos terminais do motor, no caso foi utilizado aproximadamente 50V.

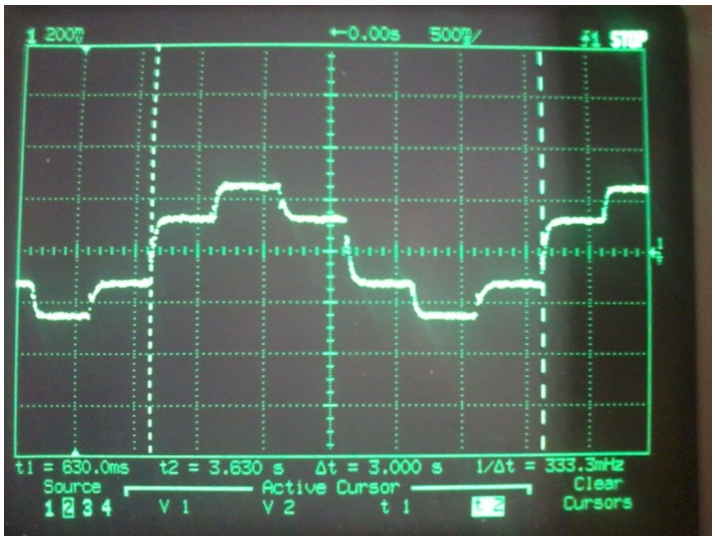


Figura 50 - Modo 3 a 3 – 0,33Hz – Corrente no motor

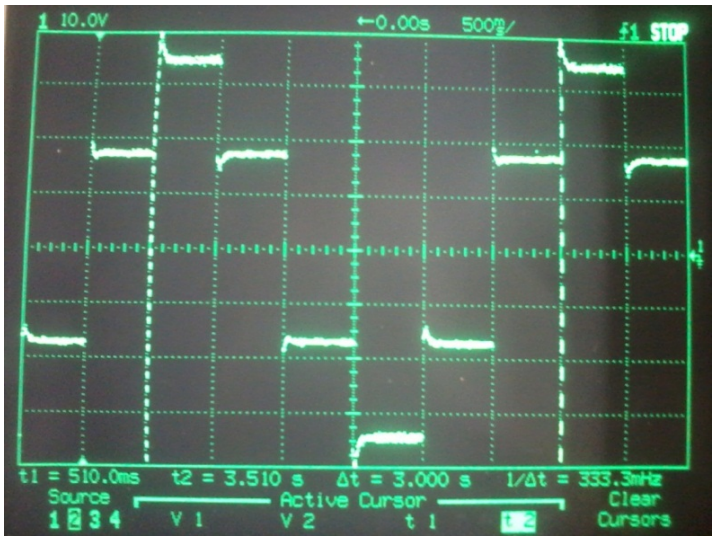


Figura 51 - Modo 3 a 3 – 0,33Hz – Tensão aplicada ao motor

Frequência do motor em 60Hz

Utilizando 2,77ms de duração dos pulsos é obtida a frequência de 60Hz de rotação do eixo do motor de indução. A Figura 52 apresenta a corrente de um das fases do motor e a Figura 53 mostra a tensão fase-neutro aplicada aos terminais do motor, que foi de aproximadamente 115V.

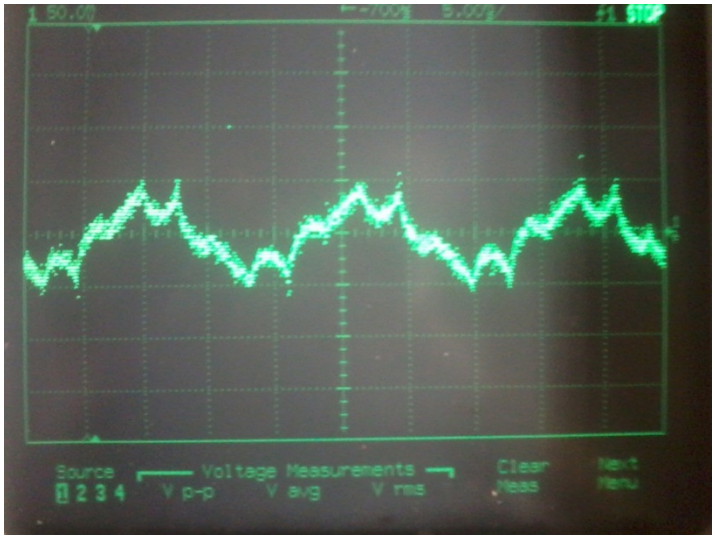


Figura 52 - Modo 3 a 3 – 60Hz – Corrente no motor

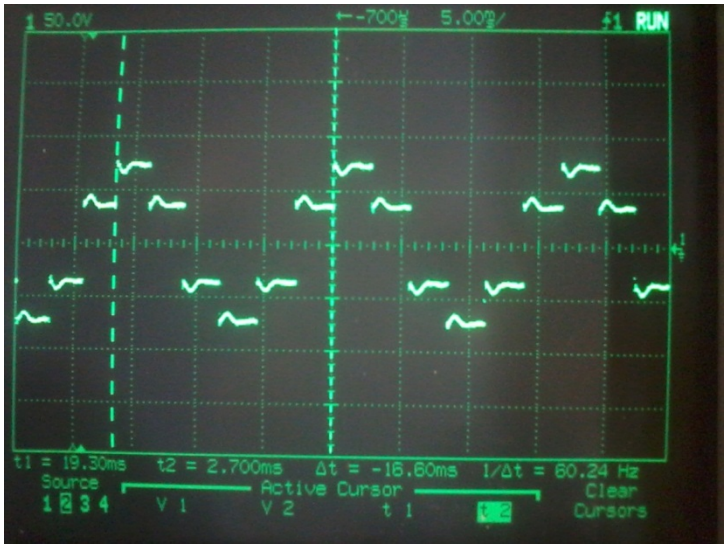


Figura 53 - Modo 3 a 3 – 60Hz – Tensão aplicada ao motor

7. Conclusões

O trabalho em questão analisou a utilização do *kit* desenvolvimento EKI LM3S8962 da *Texas Instruments* para o acionamento, e um possível controle, de máquinas elétricas de correntes contínua (imã permanente) e de corrente alternada (motor de indução trifásico). O *kit* em questão possui recursos, como módulo gerador de PWM, leitura de *encoder*, entre outros, que podem ser úteis para esta aplicação. A utilização deste *kit* específico se deu pela integração destes diversos recursos em uma única plataforma, aumentando a eficácia e reduzindo o custo em comparação a estes recursos implementados separadamente.

O foco do trabalho foi detalhar o *kit* de desenvolvimento, formas de operação e de configuração, para o acionamento das máquinas elétricas por meio de 2 circuitos de acionamento. O circuito *chopper* para o acionamento com velocidade variável do motor de corrente contínua por meio do controle do nível de tensão média aplicada ao circuito de campo do motor e o circuito inversor de tensão trifásicos para o acionamento do motor de corrente alternada controlando a frequência da tensão de alimentação do determinado motor.

A estrutura escrita deste trabalho foi realizada para prover informações sobre o *kit* de desenvolvimento EKI LM3S8962, em seguida foi abordada a configuração dos recursos presentes no *kit*. Foram realizado testes no *kit* de desenvolvimento e por fim foi realizada a validação do *kit* para o objetivo proposto acionando dois motores com velocidade variável, um motor de corrente contínua de imã permanente e um motor de corrente alternada de indução trifásica. Para esta validação foram implementados os circuitos de potência (*Chopper*) para o acionamento do motor de corrente contínua, circuitos de acionamento, para o *chopper* e para o inversor de tensão trifásico, e circuitos auxiliares, para medição de grandezas físicas como velocidade e corrente, e circuitos de proteção.

Os ensaios realizados no *kit* de desenvolvimento foram satisfatórios vistos que os recursos disponíveis são suficientes para os acionamentos propostos e ainda possibilitam outras abordagens, como controle das referidas máquinas elétricas ou acionamento do inversor utilizando modulação vetorial, já que utilizando os 6 PWMs presentes no *kit* de desenvolvimento é possível gerar esta modulação, conhecida como SPVM (*Space Vector Modulation*).

Foi levantado também o tempo gasto para as funções de aquisição do *kit* de desenvolvimento, onde foi encontrado um valor satisfatório, já que utilizando um

conversor analógico digital para a medição da corrente de um determinado motor CC por exemplo, em conjunto com a medição da velocidade e da posição do motor utilizando um *encoder* é gasto o tempo de 30,5μs. Considerando um controle digital com uma frequência de 1kHz, tem-se uma janela de tempo de 969,5μs para realização das operações de controle, tempo suficiente visto a velocidade das operações do microprocessador. Para o caso de um motor de indução trifásico (3 medições de corrente e aquisição do encoder) gasta-se 63,54μs, e portanto, numa taxa de amostragem de 1kHz restam 936μs para as tarefas de controle e supervisão.

O número máximo de amostras obtido, 16000 amostras, é suficiente para a utilização didática proposta. Uma ampliação deste ambiente pode resultar na necessidade de aumentar este número de amostras, outra abordagem, como utilização de uma memória externa, pode ser utilizada.

A validação do *kit* de desenvolvimento para esta aplicação foi realizada implementando os circuitos propostos e obtendo resultado satisfatório para o acionamento de motores de corrente contínua de imã permanente e de corrente alternada de indução trifásica. Os resultados destes ensaios podem ser expandidos para outros tipos de motores que utilizem o princípio de controle da tensão média através de um *chopper* para os motores de corrente contínua e os que utilizem o acionamento variável da velocidade através de inversores de tensão.

O anexo II é mostrado um esquemático reunindo todos os circuitos proposto com o objetivo de criação de um protótipo para, possivelmente, ser utilizado em uma matéria de graduação da Escola de Engenharia de São Carlos de acionamento de máquinas elétricas, servindo de base para um possível laboratório desta disciplina.

Anexos I – Código implementado

```
//#####################################################
//
// INCLUSÃO DE BIBLIOTECAS
//
//#####################################################


#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/gpio.h"
#include "driverlib/pwm.h"
#include "driverlib/sysctl.h"
#include "inc/hw_ints.h"
#include "inc/hw_qei.h"
#include "inc/hw_ssi.h"
#include "driverlib/adc.h"
#include "drivers/rit128x96x4.h"
#include "inc/lm3s8962.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/timer.h"
#include "driverlib/qei.h"
#include "driverlib/uart.h"
#include <stdio.h>

//#####################################################
//
// ROTINA DE ERRO
//
//#####################################################

#ifdef DEBUG
void
__error__(char *pcFilename, unsigned long ulLine)
{
}
#endif

 //#####################################################
//
// DECLARAÃO DE VARIAVEIS
//
//#####################################################

unsigned long q=0;
unsigned long ulPeriod;        //tempo
volatile unsigned long ulLoop;
   unsigned long ulValue;
char buf[40];
int e0;
int count=0;                 //Variavel utilizada para contagem de amostras
unsigned long cad[8100];      //vetor contendo informações do Conversor Analogico Digital
//int dir[4000];              //Vetor contendo informações da Direção do Motor
//int vel[8100];              //Vetor contendo informações da Velocidade de rotação do motor
//int pos[4000];               //Vetor contendo informações da Posição do eixo do motor
int velt[8100] = NULL;
int difPWM2 = 5;
float tatual = 0;
int p=0;
   int portae=0;
   int flagv=0;
   int l=0;
   float aux;
//#####################################################
//
// RDEFINIÇÕES DO USUÁRIO
//
//#####################################################

//O máximo de amostras é de cerca de 15000 amostras no total, ou seja, 15000 de apenas uma informação
//ou 3000 caso seja necessárias 5 informações. Para isso deve ser ajustado o tamanho dos vetores acima.
```

```
int amostras=7500;          //Definir quantidade de amostras
float TA=0.001;             //taxa de amostragem em segundos, minima de 100us

float TPWM = 0.02;          //tempo de variação do PWM (0.02) ou pulsos
int inipwm = 1;             //Porcentagem inicial do PWM
int fpwm = 2000;
int difPWM = 1;


//#####################################################
//
// SERIAL - Rotina para enviar strings via serial - Caracter por caracter
//
//#####################################################


void
UARTSend(const unsigned char *pucBuffer, unsigned long ulCount)
{
   //
   // Loop enquanto existem caracteres sendo enviados
   //
   while(ulCount--)
   {
      UARTCharPut(UART0_BASE, *pucBuffer++); // Escreve o próximo caracter na seroal
   }
}


//#####################################################
//
// Interrupção por tempo - AQUISIÇÃO
//
//#####################################################
void
Timer0IntHandler(void)
{
TimerIntClear(TIMER0_BASE, TIMER_TIMA_TIMEOUT); //limpa a flag de interrupção por tempo
ADC_ISC_R = 0x000000000000000000000000000ffff;
//RIT128x96x4Clear();

//Caso a taxa de amostragem seja pequena, é possivel imprimir os valores no Display,
//para isso basta descomentar os comandos abaixo referentes ao display

//########################CAD############################################
ADCProcessorTrigger(ADC_BASE, 0);
cad[count] = ADC_SSFIFO0_R;          //le o valor do conversor e coloca no vetor
cad[count] = ADC_SSFIFO0_R;          //le o valor do conversor e coloca no vetor
cad[count] = ADC_SSFIFO0_R;          //le o valor do conversor e coloca no vetor
cad[count] = ADC_SSFIFO0_R;          //le o valor do conversor e coloca no vetor
cad[count] = ADC_SSFIFO0_R;          //le o valor do conversor e coloca no vetor
cad[count] = ADC_SSFIFO0_R;          //le o valor do conversor e coloca no vetor
cad[count] = ADC_SSFIFO0_R;          //le o valor do conversor e coloca no vetor
cad[count] = ADC_SSFIFO0_R;          //le o valor do conversor e coloca no vetor
cad[count] = ADC_SSFIFO0_R;          //le o valor do conversor e coloca no vetor

//sprintf(buf, "%d", cad[count]);       //transforma em char
//RIT128x96x4StringDraw(buf, 20, 20, 15);


//##################ENCODER########################
//dir[count] = QEIDirectionGet(QEI0_BASE);  //Direção
////sprintf(buf, "%d", dir[count] );          //transforma em char
////RIT128x96x4StringDraw(buf, 20, 50, 15); //Coloca no display
//
//pos[count] = QEIPositionGet(QEI0_BASE);   //Posição
////sprintf(buf, "%d", pos[count] );          //transofrma em char
////RIT128x96x4StringDraw(buf, 20, 80, 15); //Coloca no display
////
//velocidade
//vel[count]= 1.6*QEIVelocityGet(QEI0_BASE);//Velocidade
//sprintf(buf, "%d", vel[count] );          //transorma em char
//RIT128x96x4StringDraw(buf, 80, 80, 15); //Coloca no display

//adc- 2 tacogerador
ADCProcessorTrigger(ADC_BASE, 1);
```

```
velt[count] = ADC_SSFIFO1_R;
velt[count] = ADC_SSFIFO1_R;
velt[count] = ADC_SSFIFO1_R;
velt[count] = ADC_SSFIFO1_R;
velt[count] = ADC_SSFIFO1_R;
velt[count] = ADC_SSFIFO1_R;
velt[count] = ADC_SSFIFO1_R;
velt[count] = ADC_SSFIFO1_R;
velt[count] = ADC_SSFIFO1_R;
sprintf(buf, "%d", velt[count] );             //transorma em char
//RIT128x96x4StringDraw(buf, 80, 80, 15); //Coloca no display
//###################Verificação de erro de leitura do encoder###################
//Comentado pois serve apenas para testes

//*erro0= QEIErrorGet(QEI0_BASE);       //bit de erro de leitura
//if (erro0==1)
//{
//  e0++;
//  erro0 = 0;
//}
//sprintf(buf, "%d", e0 );
//RIT128x96x4StringDraw(buf, 80, 50, 15);


count++;            //Incrementa a contagem e a posição dos vetores
}

//#####################################################
//
// Interrupção por tempo - ACIONAMENTO
//
//#####################################################

void
Timer1IntHandler(void)
{

//  // ########################PWM Acionamento de motor DC######################
  TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT); //limpa a flag de interrupção por tempo
  tatual = TPWM + tatual;

  if (tatual < 1.88)
  {
  PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, ulPeriod*(inipwm+difPWM2)/100);
  PWMPulseWidthSet(PWM_BASE, PWM_OUT_1, ulPeriod*(inipwm+difPWM2)/100);
  difPWM2=difPWM+difPWM2;
  }
  else if (tatual <5)
  {
  }
  else if (tatual <6.98)
  {
  PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, ulPeriod*(difPWM2)/100);
  PWMPulseWidthSet(PWM_BASE, PWM_OUT_1, ulPeriod*(difPWM2)/100);
  difPWM2=difPWM2-difPWM;
  }
  //#########################################################################

  //################### MODO 6 PULSOS ######################
//  if (p==0)
//  {
//    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 132);
//    p++;
////    for(l=0;l<=(80000); l++)
////    {
////      aux=aux*0;
////    }
////    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 0);
//  }
//  else if (p==1)
//  {
//    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 136);
//    p++;
```

```
////    for(l=0;l<=(80000); l++)
////    {
////      aux=aux*0;
////    }
////    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 0);
//  }
//  else if (p==2)
//  {
//    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 40);
//    p++;
////    for(l=0;l<=(80000); l++)
////    {
////      aux=aux*0;
////    }
////    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 0);
//  }
//  else if (p==3)
//  {
//    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 48);
//    p++;
////    for(l=0;l<=(80000); l++)
////    {
////      aux=aux*0;
////    }
////    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 0);
//  }
//  else if (p==4)
//  {
//    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 80);
//    p++;
////    for(l=0;l<=(80000); l++)
////    {
////      aux=aux*0;
////    }
////    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 0);
//  }
//  else if (p==5)
//  {
//    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 68);
//    p=0;
//    for(l=0;l<=(80000); l++)
//    {
//      aux=aux*0;
//    }
//    GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6 |
GPIO_PIN_7), 0);
  }




//  RIT128x96x4Clear();          //Limpa o display
//RIT128x96x4StringDraw("timer 1", 80, 50, 15);
//}

//*****************************************************
//
// CONFIGURAÇÃO DOS RECURSOS
//
//*****************************************************
int
main(void)
{
    volatile unsigned long ulLoop;
    int i;

//LED
```

```
SYSCTL_RCGC2_R = SYSCTL_RCGC2_GPIOF;
ulLoop = SYSCTL_RCGC2_R;
GPIO_PORTF_DIR_R = 0x01;
GPIO_PORTF_DEN_R = 0x01;

RIT128x96x4Init(1000000);


  //
  // Ajustar o clock a partir do cristal
  //
  SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_OSC | SYSCTL_OSC_MAIN |
                 SYSCTL_XTAL_8MHZ);
  SysCtlPWMClockSet(SYSCTL_PWMDIV_1);


  //
  // Bail out if there is not a PWM peripheral on this part.
  //
  if(!SysCtlPeripheralPresent(SYSCTL_PERIPH_PWM))
  {
    while(1);
  }

  //
  // Enable the peripherals used by this example.
  //
  SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);    //utilizar saída B
  SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);    //utilizar saída E

  //
  // Set GPIO F0 and G1 as PWM pins.  They are used to output the PWM0 and
  // PWM1 signals.
  //
  GPIOPinTypePWM(GPIO_PORTF_BASE, GPIO_PIN_0);
  GPIOPinTypePWM(GPIO_PORTG_BASE, GPIO_PIN_1);
  GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_0);   //Pino 0 da porta B como PWM (PWM 2)
  GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_1);   //Pino 1 da porta B como PWM (PWM 3)
  GPIOPinTypePWM(GPIO_PORTE_BASE, GPIO_PIN_0);   //Pino 0 da porta E como PWM (PWM 4)
  GPIOPinTypePWM(GPIO_PORTE_BASE, GPIO_PIN_1);   //Pino 1 da porta E como PWM (PWM 5)

  //
  // Compute the PWM period based on the system clock.
  //
//   ulPeriod = SysCtlClockGet() / 440;
     ulPeriod = SysCtlClockGet() / fpwm;

  //
  // Set the PWM period to 440 (A) Hz.
  //
  PWMGenConfigure(PWM_BASE, PWM_GEN_0,
                  PWM_GEN_MODE_UP_DOWN | PWM_GEN_MODE_NO_SYNC);
  PWMGenPeriodSet(PWM_BASE, PWM_GEN_0, ulPeriod);
  PWMGenConfigure(PWM_BASE, PWM_GEN_1,
                  PWM_GEN_MODE_UP_DOWN | PWM_GEN_MODE_NO_SYNC);
  PWMGenPeriodSet(PWM_BASE, PWM_GEN_1, ulPeriod);
  PWMGenConfigure(PWM_BASE, PWM_GEN_2,
                  PWM_GEN_MODE_UP_DOWN | PWM_GEN_MODE_NO_SYNC);
  PWMGenPeriodSet(PWM_BASE, PWM_GEN_2, ulPeriod);

  //
  // Set PWM0 to a duty cycle of 25% and PWM1 to a duty cycle of 75%.
  //
  PWMPulseWidthSet(PWM_BASE, PWM_OUT_0, ulPeriod*inipwm/100);
  PWMPulseWidthSet(PWM_BASE, PWM_OUT_1, ulPeriod*inipwm/100);
  PWMPulseWidthSet(PWM_BASE, PWM_OUT_2, ulPeriod*inipwm/100);
  PWMPulseWidthSet(PWM_BASE, PWM_OUT_3, ulPeriod*inipwm/100);
  PWMPulseWidthSet(PWM_BASE, PWM_OUT_4, ulPeriod*inipwm/100);
  PWMPulseWidthSet(PWM_BASE, PWM_OUT_5, ulPeriod*inipwm/100);

  //
  // Enable the PWM0 and PWM1 output signals.
  //
```

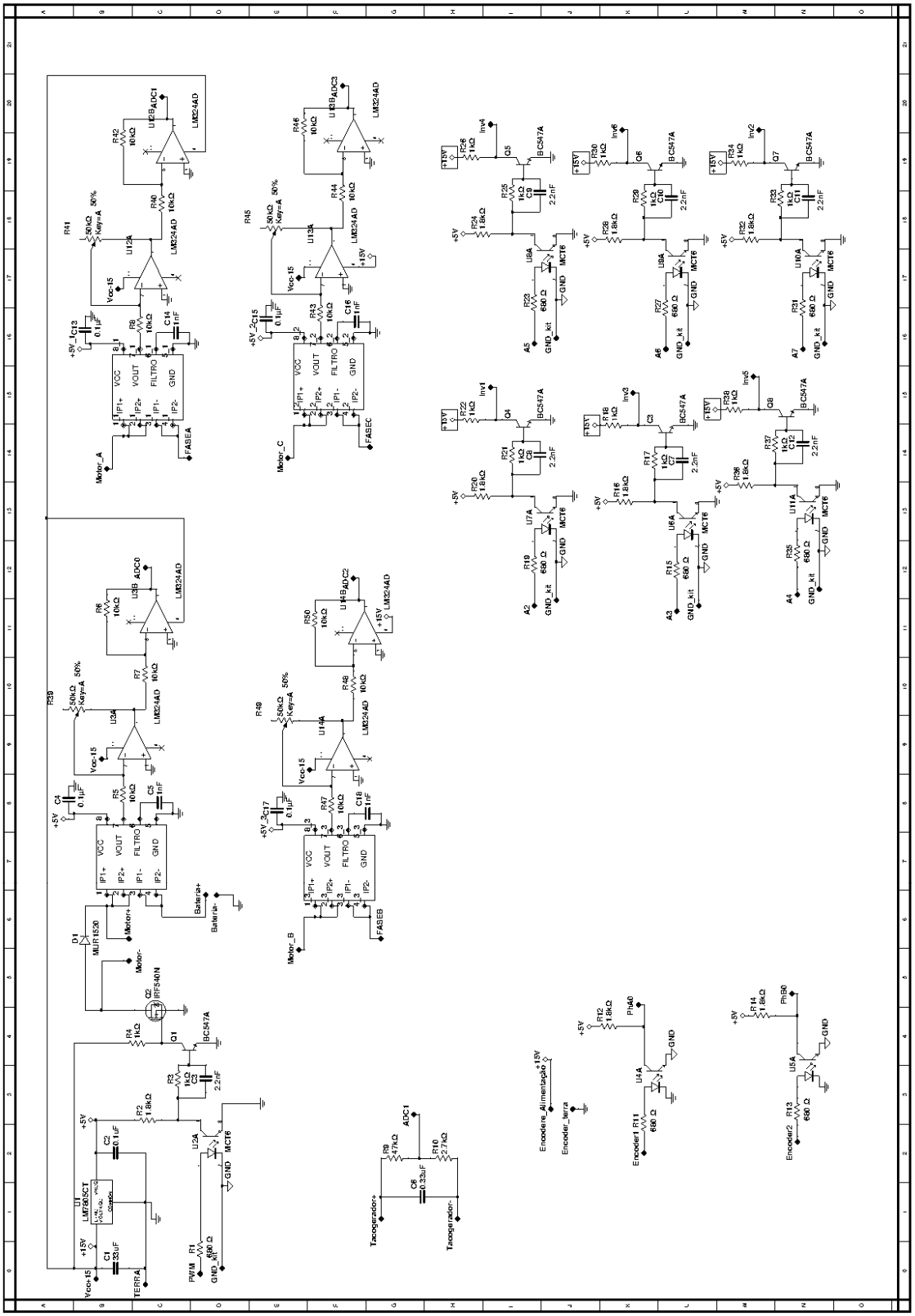
```
    PWMOutputState(PWM_BASE, PWM_OUT_0_BIT | PWM_OUT_1_BIT | PWM_OUT_2_BIT | PWM_OUT_3_BIT |
            PWM_OUT_4_BIT | PWM_OUT_5_BIT , true);

    //
    // Enable the PWM generator.
    //
    PWMGenEnable(PWM_BASE, PWM_GEN_0);
    PWMGenEnable(PWM_BASE, PWM_GEN_1);
    PWMGenEnable(PWM_BASE, PWM_GEN_2);

//###############   CAD    ##################################################


SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC);
ADCSequenceConfigure(ADC_BASE, 0, ADC_TRIGGER_PROCESSOR, 1);
ADCSequenceStepConfigure(ADC_BASE, 3, 0, ADC_CTL_CH0 | ADC_CTL_END);
ADCProcessorTrigger(ADC_BASE, 0);

ADCSequenceConfigure(ADC_BASE, 1, ADC_TRIGGER_PROCESSOR, 0);
ADCProcessorTrigger(ADC_BASE, 1);
ADCSequenceStepConfigure(ADC_BASE, 1, 0, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 1, 1, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 1, 2, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 1, 3, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 1, 4, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 1, 5, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 1, 6, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 1, 7, ADC_CTL_CH1);
ADCSequenceStepConfigure(ADC_BASE, 1, 8, ADC_CTL_CH1 | ADC_CTL_END);

ADCSequenceEnable(ADC_BASE, 0);
ADCSequenceEnable(ADC_BASE, 1);

//#################timer - interrupção  ####################
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER0);     //habilita o timer 0
SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);     //habilita o timer 1

IntMasterEnable();
//timer 0
TimerConfigure(TIMER0_BASE, TIMER_CFG_32_BIT_PER);       //Configura o timer 0
TimerLoadSet(TIMER0_BASE, TIMER_A, SysCtlClockGet()*TA); //Ajuste de tempo,
//para variar o tempo de interrução deve mudar o valor de TA, onde a interrupçaõ ocorrerá a cada TA segundos
IntEnable(INT_TIMER0A);                          //habilita a interução do timer
TimerIntEnable(TIMER0_BASE, TIMER_TIMA_TIMEOUT);      //habilita interrupção do timer0
TimerEnable(TIMER0_BASE, TIMER_A);                  //iniicializa o timer 0
//timer 1

TimerConfigure(TIMER1_BASE, TIMER_CFG_32_BIT_PER);    //Configura o timer 1
TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()*TPWM); //Ajuste de tempo, para variar o tempo de
interrução deve mudar o valor de TPWM, onde a

interrupçaõ ocorrerá a cada TPWM segundos
//IntEnable(INT_TIMER1A);     //habilita a interução do timer1, no caso está comentado pois só sera inicializado
no loop infinito, quando p botão for

acionado
TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);    //habilita interrupção do timer0
TimerEnable(TIMER1_BASE, TIMER_A);    //iniicializa o timer 1
IntEnable(INT_TIMER1A);
//#################encoder#####################################

//QEIEnable(QEI_BASE);
SysCtlPeripheralEnable(SYSCTL_PERIPH_QEI);    //utilizar encoder
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);  //utilizar porta C
QEIDisable(QEI0_BASE);                    //desabilitar encoder 0
QEIVelocityDisable(QEI0_BASE);              //desabilitar velocidade
GPIOPinTypeQEI(GPIO_PORTC_BASE, GPIO_PIN_4);  //Pino 4 da porta C como encoder
GPIOPinTypeQEI(GPIO_PORTC_BASE, GPIO_PIN_6);  //Pino 6 da porta C como encoder
QEIConfigure(QEI_BASE,QEI_CONFIG_CAPTURE_A |   //configuração
        QEI_CONFIG_NO_RESET |
        QEI_CONFIG_QUADRATURE |
        QEI_CONFIG_NO_SWAP, 2999);
QEIVelocityConfigure(QEI_BASE, QEI_VELDIV_1, 100000);
QEIVelocityEnable(QEI0_BASE);               //habilita velocidade

QEIEnable(QEI0_BASE);                     //habilita encoder
```

```
//#########################serial#################################
SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);    //habilita serial
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);    //habilita porta A
GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);  //Pinos 0 e 1 da porta A como serial
UARTConfigSetExpClk(UART0_BASE, SysCtlClockGet(), 115200,     //configuração
                    (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                     UART_CONFIG_PAR_NONE));
IntEnable(INT_UART0);                           //Habilita serial
UARTIntEnable(UART0_BASE, UART_INT_RX | UART_INT_RT);
//UARTSend((unsigned char *)"Enter text: ", 12);       //Chama rotina para enviar o texto "Enter text"


RIT128x96x4StringDraw("Amostrando", 20, 20, 15);

//teste tempo
//GPIOPinTypeGPIOOutput(GPIO_PORTC_BASE, (GPIO_PIN_4));
//GPIODirModeSet(GPIO_PORTC_BASE, (GPIO_PIN_4), GPIO_DIR_MODE_OUT);

//Modo 6 pulsos
//GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, (GPIO_PIN_2 ));
//GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, (GPIO_PIN_3 ));
//GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, (GPIO_PIN_4 ));
//GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, (GPIO_PIN_5 ));
//GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, (GPIO_PIN_6 ));
//GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, (GPIO_PIN_7 ));
//GPIODirModeSet(GPIO_PORTA_BASE, (GPIO_PIN_2) , GPIO_DIR_MODE_OUT);
//GPIODirModeSet(GPIO_PORTA_BASE, (GPIO_PIN_3) , GPIO_DIR_MODE_OUT);
//GPIODirModeSet(GPIO_PORTA_BASE, (GPIO_PIN_4) , GPIO_DIR_MODE_OUT);
//GPIODirModeSet(GPIO_PORTA_BASE, (GPIO_PIN_5) , GPIO_DIR_MODE_OUT);
//GPIODirModeSet(GPIO_PORTA_BASE, (GPIO_PIN_6) , GPIO_DIR_MODE_OUT);
//GPIODirModeSet(GPIO_PORTA_BASE, (GPIO_PIN_7) , GPIO_DIR_MODE_OUT);


//Chave
//

//#####################################################
//
// LOOP
//
//#####################################################
   while(1)
   {

//     portae=0;
//     portae = GPIOPinRead(GPIO_PORTE_BASE,(GPIO_PIN_2 | GPIO_PIN_1 | GPIO_PIN_0 ));
//     if (portae==1)
//     {
//       IntEnable(INT_TIMER1A);
//
//     }
//     if (portae==4)
//     {
//       IntDisable(INT_TIMER1A);      //Desabilita a interrupção 0 (amostras)
//       GPIOPinWrite(GPIO_PORTA_BASE, (GPIO_PIN_2 | GPIO_PIN_3 | GPIO_PIN_4 | GPIO_PIN_5 | GPIO_PIN_6
| GPIO_PIN_7), 0);
//
//     }
//     if (portae==2)
//     {
//       TPWM = 0.1;
//       TimerLoadSet(TIMER1_BASE, TIMER_A, SysCtlClockGet()*TPWM); //Ajuste de tempo
//       flagv++;
//       if (flagv==1)
//       {
//         TPWM = 0.5;
//       }
//       if (flagv==2)
//       {
//         TPWM = 0.1;
//       }
//       if (flagv==3)
//       {
//         TPWM = 1/360;
//       }
```

```
//     }
     if (count > amostras-1)
{

RIT128x96x4Clear();
IntDisable(INT_TIMER0A);      //Desabilita a interrupção 0 (amostras)
IntDisable(INT_TIMER1A);    //Desabilita a interrupção 1 (PWM)
RIT128x96x4StringDraw("enviando via serial", 20, 20, 15);

//#####################################################
//
// COMUNICAÇÃO SERIAL
//
//#####################################################
UARTSend("CAD:",5);
            for(i=0;i<=(amostras-1); i++)
            {
              sprintf(buf, "%i", cad[i]);
              UARTSend(" ", 1); // manda espaço para separar
                 if (cad[i] <= 9)
                 {
                  UARTSend(buf, 1);
                 }
                 else if (cad[i] <= 99)
                 {
                 UARTSend(buf, 2);
                 }
                 else if  (cad[i] <= 999)
                 {
                 UARTSend(buf, 3);
                 }
                 else
                 {
                 UARTSend(buf, 4);
                 }
              }
//////################### Posição ################
//UARTSend(" POS=",5);
//for(i=0;i<=(amostras-1); i++)
//{
//sprintf(buf, "%d", pos[i]);
//UARTSend(" ", 1); // manda espaço para separar
//  if (pos[i] <= 9)
//  {
//    UARTSend(buf, 1);
//     }
//  else if (pos[i] <= 99)
//  {
//    UARTSend(buf, 2);
//  }
//  else if  (pos[i] <= 999)
//  {
//    UARTSend(buf, 3);
//  }
//  else
//  {
//    UARTSend(buf, 4);
//  }
//}
//#########################################
            UARTSend(" VELT =",7);
for(i=0;i<=(amostras-1); i++)
{
sprintf(buf, "%d", velt[i]);
UARTSend(" ", 1); // manda espaço para separar
  if (velt[i] <= 9)
  {
    UARTSend(buf, 1);
  }
  else if (velt[i] <= 99)
  {
    UARTSend(buf, 2);
  }
  else if  (velt[i] <= 999)
  {
    UARTSend(buf, 3);
```

```
  }
  else
  {
    UARTSend(buf, 4);
  }
}

//###################
//UARTSend(" VEL=",5);
//for(i=0;i<=(amostras-1); i++)
//{
//sprintf(buf, "%d", vel[i]);
//UARTSend(" ", 1); // manda espaço para separar
//  if (vel[i] <= 9)
//  {
//    UARTSend(buf, 1);
//  }
//  else if (vel[i] <= 99)
//  {
//    UARTSend(buf, 2);
//  }
//  else if  (vel[i] <= 999)
//  {
//    UARTSend(buf, 3);
//  }
//  else
//  {
//    UARTSend(buf, 4);
//  }
//}
//###################
//UARTSend(" DIR=",5);
//for(i=0;i<=(amostras-1); i++)
//{
//sprintf(buf, "%d", dir[i]);
//UARTSend(" ", 1); // manda espaço para separar
//  if (dir[i] <= 9)
//  {
//    UARTSend(buf, 1);
//  }
//  else if (dir[i] <= 99)
//  {
//    UARTSend(buf, 2);
//  }
//  else if  (dir[i] <= 999)
//  {
//    UARTSend(buf, 3);
//  }
//  else
//  {
//    UARTSend(buf, 4);
//  }
//}
RIT128x96x4Clear();
RIT128x96x4StringDraw("FIM", 20, 20, 15);
//sprintf(buf, "%d", count );
//RIT128x96x4StringDraw(buf, 20, 40, 15);
RIT128x96x4Clear();
RIT128x96x4StringDraw("FIM", 20, 20, 15);
count=0;
}
//if (difPWM2>=99)
//{
//  IntDisable(INT_TIMER1A);    //Desabilita a interrupção 1 (PWM)
//}
   }
}
```

Anexos II - Esquemático



Bibliografia

1. TEXAS INSTRUMENTS. **Manual do usuário: Stellaris LM3S8962 Evaluation Board**. Austin. 2009.
2. TEXAS INSTRUMENTS. **Data Sheet: Stellaris® LM3S8962 Evaluation Board**. Austin. 2009.
3. YIU, J. **The Definitive guide to the ARM CORTEX-M3**. Oxford: Elsevier, 2007.
4. VIEIRA, L. **Motores elétricos - Princípios e fundamentos**. Disponivel em: <http://www.dea.uem.br/disciplinas/eletrotecnica/motoreseletricos.pdf>. Acesso em: 03 Outubro 2012.
5. KRISHNAN, R. R. **Electric motor drives modeling, analysis, and control**. Upper Saddle River: N.J. Prentice Hall, 2001.
6. LANDER, C. **Eletrônica Industrial Teoria e Aplicações**. Tradução de Maurício Eduardo Bernardino Ribeiro. 2ª. ed. São Paulo: Makron books, 1996.
7. RASHID, M. H. **Eletrônica de Potência:** Circuitos, dispositivos e aplicações. Tradução de Carlos Alberto Favato. São Paulo: Makron books, 1999.
8. MELO, G. A. et al. Sistema de tração elétrica flexível baseada em veículos trólebus para alimentação com redes cc ou ca. **Revista Controle & Automação** , Ilha Solteira, v. 23, n. 5, Setembro e Outubro 2012.
9. ALMEIDA, J. L. A. **Eletrônica de potência**. 2ª. ed. São Paulo: Érica, 1986.
10. AHMED, A. **Eletrônica de Potência**. Tradução de Eduardo Vernes Mack. São Paulo: Prentice Hall, 2000.
11. KAZIMIERCZUK, M. K. **Pulse-width Modulated DC–DC**. Wright State University. Dayton. 2008.
12. NEACSU, D. O. **Power Switching Converters:** Medium and high power. Boca Raton: Taylor & Francis, 2006.
13. MEZAROBA, M. **Material didático:** Modulação PWM. Disponivel em: <http://www.joinville.udesc.br/portal/professores/mezaroba/materiais/Modulacao_PWM.pdf>. Acesso em: 01 Novembro 2012.
14. AGUIAR, M. L. **Material didático:** Acionamento e controle de máquinas elétricas. Disponivel em: <http://www.sel.eesc.usp.br:8085/Disciplinas/disciplinas/disc_login.jsp?id=24>. Acesso em: 2012 Outubro 29.
15. JÚNIOR, W. D. S. **ASIC para geração de senpide com frequência variável baseada em PWM**. UNICAMP. Campinas. 2002.
16. IAR SYSTEM AB. **Manual do usuário: IAR Embedded Workbench IDE**. Uppsala. 2009.
17. LUMINARY MICRO, INC. **Manual do usuário: Stellaris Peripheral Driver Library**. Austin. 2007.
18. ELIAS, P. R. **OLED:** O futuro vem aí. Disponivel em: <http://webinsider.uol.com.br/2012/06/03/oled-o-futuro-vem-por-ai/>. Acesso em: 03 Outubro 2012.
19. GONZAGA, A.; RODRIGUES, E. L. L.; PAIVA, M. S. V. D. **Aplicação de microprocessadores II**. Disponivel em: <http://iris.sel.eesc.sc.usp.br/sel337/serial.pdf>. Acesso em: 29 outubro 2012.
20. FTDI. **Datasheet: FT232R USB UART IC**. Glasgow. 2012.
21. HILGRAEVE. **Hyperterminal**. Disponivel em:

. <http://www.hilgraeve.com/hyperterminal/>. Acesso em: 01 Dezembro 2012.

22 NETO, C. C.; THOMÉ, A. G. **Material didático:** Comunicação de dados, 2000. . Disponivel em: <http://equipe.nce.ufrj.br/thome/comdados/comdad.htm>. Acesso em: 31 Novembro 2012.

23 FAIRCHILD. **Datasheet: 6N137**. San Jose. 2011.
.

24 INTERNATIONAL RECTIFIER. **Datasheet: IRF540N**. Kansas. 2001.
.

25 ON SEMICONDUCTOR. **Datasheet: MUR1520**. [S.l.].
.

26 NIST. Physical Measurement Laboratory. Disponivel em: . <http://www.nist.gov/pml/div683/hall_effect.cfm>. Acesso em: 01 Dezembro 2012.

27 ALLEGRO. **Datasheet: ACS712**. Worcester. 2011.
.

28 INTERSIL. **Datasheet: LM324**. Palm Bay. 2001.
.

29 SEMIKRON. **Datashet: SKD51/12**. Nürnberg. 2007.
.

30 SEMIKRON. **Datasheet: SKM40GDL123D**. Nürnberg. 2009.
.

31 SEMIKRON. **Datasheet: SKHI22**. Nürnberg. 2008.
.