UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO DE COMPUTAÇÃO
ENGENHARIA DE COMPUTAÇÃO

HENRIQUE DALAVALE FABRETTI

DESENVOLVIMENTO DE UM SISTEMA DE COMUNICAÇÃO WIRELESS COM MATLAB/SIMULINK USANDO MICROCONTROLADOR DE 8 BITS DA FAMÍLIA HCS08.

TRABALHO DE CONCLUSÃO DE CURSO

CORNÉLIO PROCÓPIO

2016

HENRIQUE DALAVALE FABRETTI

DESENVOLVIMENTO DE UM SISTEMA DE COMUNICAÇÃO WIRELESS COM MATLAB/SIMULINK USANDO MICROCONTROLADOR DE 8 BITS DA FAMÍLIA HCS08

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina Trabalho de Conclusão de Curso 2, do curso de Engenharia de Computação da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para a obtenção do título de Bacharel.

Orientador: Prof. Dr. Marcio Aurelio Furtado Montezuma.

CORNÉLIO PROCÓPIO

2016

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Cornélio Procópio
Nome da Diretoria
Nome da Coordenação
Nome do Curso

TERMO DE APROVAÇÃO

DESENVOLVIMENTO DE UM SISTEMA DE COMUNICAÇÃO WIRELESS COM MATLAB/SIMULINK USANDO MICROCONTROLADOR DE 8 BITS DA FAMÍLIA HCS08

por

Henrique Dalavale Fabretti

Este Trabalho de Conclusão de Curso de graduação foi julgado adequado para obtenção do Título de Bacharel em Engenharia de Computação e aprovado em sua forma final pelo Programa de Graduação em Engenharia de Computação da Universidade Tecnológica Federal do Paraná.

Cornélio Procópio, 07/06/2016

_____

Prof. Dr. Marcio Aurelio Furtado Montezuma (Orientador)

_____

Prof. Dr. André Sanches Fonseca Sobrinho (Membro)

_____

Prof. Dr. Cristiano Marcos Agulhari (Membro)

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

AGRADECIMENTOS

Agradeço primeiramente a Deus, por sempre me iluminar e dar forças nos momentos mais difíceis.

Agradeço ao meu orientador Prof. Dr. Márcio Aurélio Furtado Montezuma, pela ajuda, pelo conhecimento compartilhado e pela confiança.

Aos meus pais, que sempre me apoiaram em todo o meu caminho, e nunca mediram esforços para que eu pudesse realizar meus sonhos, sempre com conselhos e incentivos que me fizeram chegar até o fim.

Aos meus irmãos, Lucas e Letícia, e minha noiva, Débora, pelos momentos e por estarem sempre ao meu lado.

Aos amigos de infância e de universidade, Murilo e Júnior.

Aos meus colegas de sala pelos momentos que juntos passamos.

A todos meus familiares, que mesmo distantes sempre estão comigo em pensamento.

Aos companheiros do laboratório LaSisC, e os alunos do mestrado Joana e Niro pela ajuda no percorrer do trabalho.

Enfim, a todos os que por algum motivo contribuíram para a realização desta pesquisa.

“Tudo tem seu tempo determinado, e há tempo para todo o propósito debaixo do céu”.
(Eclesiastes, 3-1)

RESUMO

FABRETTI, Henrique Dalavale. Desenvolvimento de um sistema de comunicação wireless com matlab/simulink usando microcontrolador de 8 bits da família HCS08. 2016. 56 f. Trabalho de Conclusão de Curso (Graduação) – Engenharia de Computação. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2016.

Este trabalho consiste na implementação de um sistema de comunicação sem fio entre um microcontrolador Freescale® ColdFire v1 e um servomotor, com o controle por Matlab/Simulink. A finalidade é criar um sistema de comunicação sem fio que controle a velocidade deste motor de forma remota. A placa utilizada é a DEMOQE128, e um módulo de comunicação radiofrequência para wireless, que por sua vez será feita de forma serial, seguindo o protocolo RS232. A medição da posição angular do servomotor é feita por um *encoder*, com o objetivo de se obter a velocidade angular. O microcontrolador faz o controle da corrente do servomotor através do *duty cycle* de um sinal PWM. Para desenvolvimento de uma interface gráfica e do controle é utilizado o software Simulink em tempo real, que facilitará a troca do sistema de controle sem a necessidade de reprogramar o microcontrolador. O resultado é um sistema completo, por meio do Matlab/Simulink, se controla a velocidade do motor à distância.

Palavras-chave: Microcontrolador. Comunicação RF. Sistema de Controle. HCS08.

ABSTRACT

FABRETTI, Henrique Dalavale. **Development a wireless communication system with Matlab/Simulink using 8-bit microcontroller HCS08 family**. 2016. 56 f. Proposta de Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Computação) - Federal Technology University - Parana. Cornelio Procopio, 2016.

This work consists in the implementation of a system of wireless communication between a Freescale® HCS08 microcontroller and a servomotor, with Matlab/Simulink being used to control the system. The purpose is to control remotely the speed of the engine. The evaluation board used is DEMOQE128, and a radio frequency communicator module for wireless, which uses serial communication following the RS232 protocol. The measurement of the angular position of the servomotor is made by an encoder in order to obtain the angular velocity. The microcontroller controls the current of the servomotor through the duty cycle of a PWM signal. To develop a graphical interface and perform the control the Simulink real-time software, which facilitate the exchange of the controller without the need to reprogram the microcontroller system used. The result is a completed system where through Matlab/Simulink you can control the motor's speed remotely.

Keywords: Microcontroller. Radiofrequency Communication. Control System. HCS08.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SIGLAS

LaSisC	Laboratório de Sistemas Automatizados e Controle

LISTA DE ACRÔNIMOS

CMOS	Semicondutor de metal-óxido complementar
dBm	Decibel miliwatt
FSK	Frequency-shift keying
LAN	Local Area Network
LED	Light Emitting Diode
PI	Proporcional e integral
PWM	Pulse Width Modulation
RPM	Rotações por minuto
RF	Radio Frequência
UART	Universal Asynchronous Receiver/Transmitter
KBPS	Quilo bit por segundo

LISTA DE SÍMBOLOS

K	Quilo
M	Mega
G	Giga
Hz	Heartz
Δt	Taxa de amostragem (Sample time)
$\Delta Counts$	Variação entre contagens
Kp	Ganho proporcional
Ti	Ganho tempo integral
$\mu(t)$	Resposta do controle em função do tempo
$e(t)$	Erro em função do tempo

Sumário

1 INTRODUÇÃO

A ideia de microcontroladores e microprocessadores surgiu em meados de 1970 com um projeto de uma calculadora eletrônica, iniciado uma empresa japonesa chamada BUSICOM. Esta empresa enviou uma equipe de engenheiros responsáveis pelo projeto à procura de Marcian Hoff, da Intel Corporation. Marcian, ao ver o projeto, teve uma ideia diferente: ao invés de desenvolver um chip que somente seria uma calculadora, projetar um chip que funcionasse de acordo com um programa/firmware. Após a compra da licença da empresa japonesa, a Intel lança o processador 4004 de 4 bits (A História..., 2014). Foi então a partir disso que os microcontroladores começaram a ganhar espaço no mercado.

Um microcontrolador (*Micro Control Unit - MCU*) é um computador em um chip. Ele possui um processador, memória e periféricos de entrada e saída (I*n/Out - IO)*. A principal diferença entre um microcontrolador e um microprocessador é a funcionalidade: para que um microprocessador possa ser usado, ele necessita de outros componentes como memória, chipsets, e periféricos para receber e enviar dados. Por outro lado, o microcontrolador foi projetado para ter todas estas funcionalidades dentro de uma única pastilha (A História..., 2014).

O rápido e intenso crescimento do mercado tecnológico das ultimas décadas, junto com a necessidade de comunicações rápidas e práticas, levaram ao surgimento das redes digitais sem fio. O mercado atual exige flexibilidade e agilidade, e muitas empresas notaram que isso pode ser alcançado quando se tem uma estrutura de comunicação que seja também ágil e flexível. Redes sem fio são redes onde a troca de dados entre dois nós é feita sem a necessidade de cabos, usando o ar como meio de propagação dos dados (ALMEIDA, 2009).

O tema abordado neste trabalho é o desenvolvimento de um sistema de comunicação *wireless* que controle um servomotor através de um microcontrolador e uma ação de controle. A comunicação *wireless* é feita utilizando-se um dispositivo de rádio frequência, e o motor é controlado através de pulsos de PWM, com realimentação através de *encoders*. A figura 1 mostra a ideia da ausência de fios e cabos para a comunicação, que é o objetivo deste trabalho.

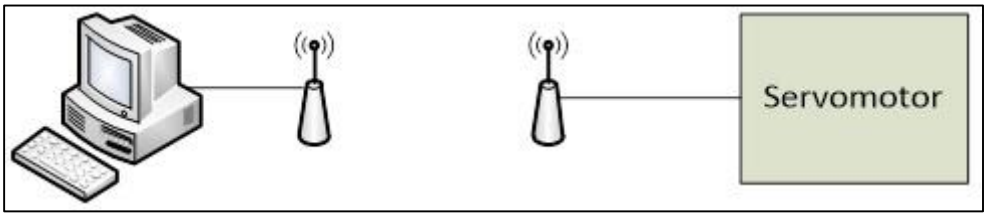


Figura 1 - Representação da comunicação entre o controle e o servomotor.
Fonte: Autoria própria.

1.1 PROBLEMA E PREMISSAS

O controle de sistemas pode ser razoavelmente problemático quando a transmissão de informações entre o controlador e o atuador é feita de forma *wireless*. Sabe-se que a propagação de dados no ar está sujeita a vários tipos de problemas, como atenuação, perda no espaço livre, ruídos, desvanecimento e absorção atmosférica, que pode promover alterações nos dados transmitidos.

O sistema de controle proposto neste trabalho sofre de tais problemas, pois será enviada a velocidade angular desejada (em rpm), e como retorno o microcontrolador dirá a velocidade atual em que está, que será calculada através da sua posição angular medida pelo decodificador de quadratura. Sendo assim, caso o computador envie uma velocidade X, e, ao passar pelo meio de transmissão (ar) este mudar para Y, o motor fará algo totalmente diferente do que foi sugerido pelo computador, e isto não pode ocorrer.

O trabalho propõe controlar a velocidade de um motor, utilizando um sistema de controle com um tempo entre máximo de *delay* (ou atraso) entre recepção e transmissão de 1 a 25 milissegundos.

Para não ocorrer um problema de processamento digital de sinal, é necessário também evitar uma discretização muito grande do sinal. A simulação desse sistema é realizada em modo contínuo; logo, a discretização sendo pequena evita-se tornar o sistema de malha fechada instável (SÁ, 2003).

1.2 JUSTIFICATIVA

Aplicar os conhecimentos adquiridos durante o curso de Engenharia de Computação, unindo disciplinas como robótica, sistemas microcontrolados, sistemas embarcados, sistemas de controle, transmissão de dados e processamento digital de sinais; Ter uma ferramenta de ensino de sistemas de controle utilizando o software Simulink, visto que a entrada e a saída (microcontrolador e motor, respectivamente) não serão alteradas, mas sim a malha de controle que será facilmente alterada utilizando o software supracitado.

A Universidade Tecnológica Federal do Paraná (UTFPR) possui diversas disciplinas de sistemas de controle e de sistemas microcontrolados em diversos cursos, e estas serão beneficiadas com o trabalho proposto, pois não haverá a necessidade de utilizar um cabo para comunicação RS232 entre os dispositivos, mas sim o sistema wireless, no qual o meio de propagação dos dados será o próprio ar. Isto auxilia um professor em efetuar o controle do seu computador e demonstrar essa ação numa bancada onde o aluno esteja (com o atuador). Esta pesquisa também acata o ambiente industrial, visto que o controle de certos equipamentos, como um motor elétrico, não precisará ter seu controlador próximo ao ambiente em que se está localizado o periférico, pois a comunicação não dependerá da interligação de fios e cabos.

Entretanto, são poucos os casos em que é utilizado um sistema wireless para o controle de algum equipamento. Geralmente são utilizados cabos como serial de nove vias, USB, ethernet ou até mesmo paralelo. Mas todos estes conjuntos de fios elétricos geram alguns problemas, como o manuseio e o custo destes fios, que aumentam de acordo com a distância entre os nós. Em contrapartida, a qualidade do serviço provido ainda é menor que a das redes cabeadas, tendo como principais razões para isso a pequena banda passante devido às limitações da rádio transmissão e a alta taxa de erro devido à interferência. O preço dos equipamentos de Redes sem Fio é mais alto que os equivalentes em redes cabeadas. Intrinsecamente, os canais sem fio são mais suscetíveis a interceptores não desejados. O uso de ondas de rádio na transmissão de dados também pode interferir em outros equipamentos de alta tecnologia, como por exemplo, equipamentos utilizados em hospitais. Além disso, equipamentos elétricos são capazes de interferir na transmissão acarretando em perdas de dados e alta taxa de erros na transmissão (TELECO, 2016).

1.3 OBJETIVOS

1.3.1 Objetivos Gerais

Desenvolver um firmware para o microcontrolador ColdFire v1 da Freescale®, para que transmita os pulsos de PWM para o servomotor, a fim de controla-lo, e se comunique com o Matlab/Simulink através da comunicação RS232. Além disso, montar todo o sistema de comunicação, que envolverá os módulos rádio frequência, um decodificador de quadratura, encoder, fonte de alimentação, e fios para as conexões.

1.3.2 Objetivos Específicos

- Desenvolver uma rotina de transmissão e recepção de dados em linguagem C.
- Montar o sistema de comunicação com todos os periféricos.
- Determinar a taxa máxima de comunicação.
- Detectar erros de transmissão.
- Desenvolver uma aplicação de controle no Simulink.
- Aplicar na prática em conjunto com um servomotor.
- Ajustar os ganhos Kp e Ki.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 MODULAÇÃO POR LARGURA DE PULSO (PWM)

A modulação por largura de pulso *(*PWM) é uma técnica usada para controlar circuitos analógicos através das saídas digitais de um microcontrolador, que consiste em representar um valor pelo *duty-cycle*, que é o tempo em nível lógico alto, de um trem de pulsos de frequência fixa (SOBRINHO, 2012). O PWM é empregado em diversas aplicações, que variam desde medição e comunicação ao controle de potência e conversão. A maior parte dessas aplicações de PWM para microcontroladores se aproveita da propriedade que a energia de um sinal retangular é proporcional ao seu *duty-cycle*, pois a energia de um sinal está relacionado com a área entre o sinal e o eixo do tempo, como mostrado na Figura 2.

Quando o *duty-cycle* é configurado com um valor de 100%, o sinal resultante será uma onda cujo tempo todo estará em nível lógico alto. Quando este está configurado com 75%, a onda resultante é uma em que em 75% do tempo ela está em nível lógico alto, e 25% em nível lógico baixo. A Figura 2 ilustra o funcionamento do *duty-cycle* (SOBRINHO, 2012).

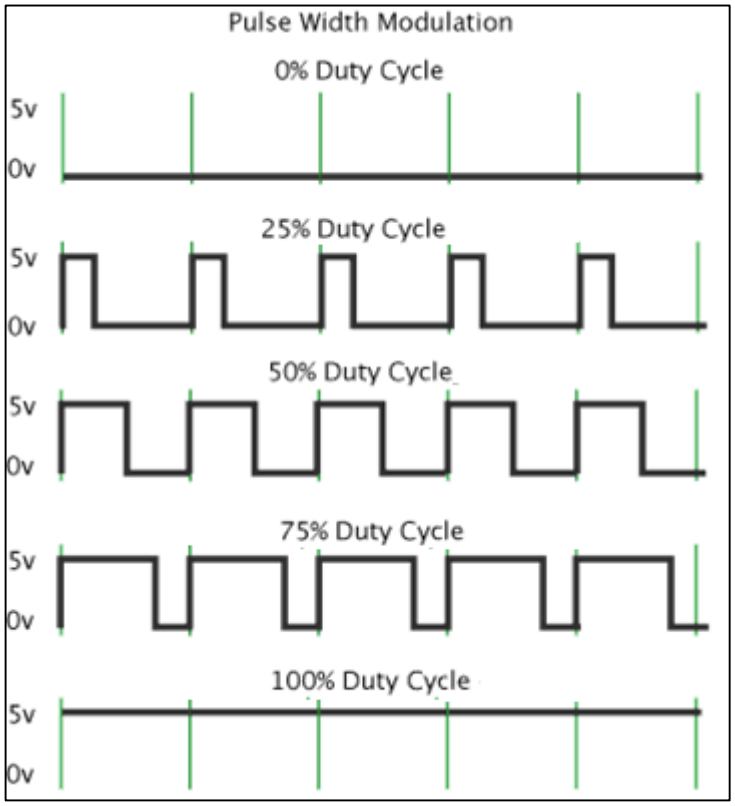


Figura 2 - Duty Cycle de um PWM.
Fonte: Adaptado de Pantech Solutions (2014).

2.2 COMUNICAÇÃO SERIAL

Em ciências da computação, comunicação em série é o processo de se enviar dados de bit em bit, sequencialmente. É um protocolo de comunicação muito comum que é utilizado por diversos dispositivos para instrumentação e também para aquisição de dados em conjunto com um dispositivo remoto de amostragem (NATIONAL INSTRUMENTS, 2014).

O conceito de comunicação serial é simples. A porta serial envia e recebe bytes de informação um bit de cada vez. Embora esta seja mais lenta que a comunicação paralela, que permite a transmissão de um byte inteiro por vez, ela é mais simples e pode ser utilizada em distâncias maiores.

Normalmente, a serial é usada para transmitir dados ASCII. A comunicação é completada usando três linhas de transmissão: Terra, Transmissão, e Recepção. A comunicação pode ser síncrona ou assíncrona, mas a maior parte dos casos

utilizado o modo assíncrono. Por ser configurada assim, a porta está apta a transmitir dados em uma linha enquanto recebe dados em outra. Os parâmetros importantes da serial são: taxa de transmissão (*baud rate*), bits de dados (*data bits*), bits de parada (*stop bits*), e paridade. Para duas portas de comunicação, estes parâmetros devem corresponder (NATIONAL INSTRUMENTS, 2014) :

a) Taxa de transmissão: uma medida de velocidade para comunicação. Isto indica o número de bits transmitidos por segundo. Por exemplo, 300 *baud* são 300 bits por segundo.
b) Bits de dados: uma medida dos bits de dados atuais em uma transmissão. Quando o computador envia um pacote de informação, a quantidade de dados pode não ser maior que 8 bits. Os valores padrão para pacotes de dados são 5, 7, e 8 bits.
c) Bits de parada: usados para sinalizar o fim da comunicação para um único pacote. Uma vez que os dados são cronometrados através da linha e cada dispositivo possui seu próprio *clock*, é possível os dois dispositivos virem a estar ligeiramente fora de sincronia. Portanto, os bits de parada não só indicam o fim da transmissão mas também dão aos computadores alguma margem de erro nas velocidades de *clock*.
d) Paridade: uma forma simples de verificação de erro que é utilizada na comunicação serial. Há quatro tipos de paridade: par, ímpar, marcada e espaçada. A opção de utilizar sem paridade está disponível.

2.3 COMUNICAÇÃO POR RADIO FREQUÊNCIA

Rádio frequência e a comunicação sem fio estão no nosso meio por mais de um século, com Alexander Popov e Oliver Lodge preparando o terreno para o rádio wireless de Guglielmo Marconi, desenvolvido no início do século 20. Em dezembro de 1901, Marconi realizou seu maior experimento, onde ele conseguiu transmitir com sucesso um código Morse de Cornwall, na Inglaterra, para St John, no Canadá.

As ondas de rádio ou radiofrequências são campos eletromagnéticos utilizados nas comunicações sem fio. Como essas ondas levam energia de um ponto

ao outro, isso permite a comunicação sem a necessidade de fios, como nas transmissões de televisão, rádio e celulares (VALLE, 2013).

Hoje em dia, RF se tornou sinônimo de comunicação sem fio e de alta frequência, que descrevem qualquer coisa de rádio AM entre 535 kHz e 1605 kHz à redes locais de computadores (LANs) à 2.4GHz. No entanto, RF tem tradicionalmente frequências definidas de alguns kHz à cerca de 1 GHz (NATIONAL INSTRUMENTS, 2016).

2.4 ENCODER

Um *encoder* é um sensor que converte um movimento angular ou linear em uma série de pulsos digitais elétricos, fornecendo para o controlador dados suficientes para transformá-los em algo útil, como posição ou velocidade angular.

A conversão desses movimentos em pulsos elétricos é feita através da detecção fotoelétrica, onde uma série de pulsos são gerados pela passagem da luz em um disco opaco, com várias aberturas transparentes. O receptor detecta a luz enviada pelo emissor e também a falta de luz, gerando assim os pulsos digitais de 0 e 1. (MECATRÔNICA ATUAL, 2014).

2.5 CONTROLADOR PROPORCIONAL-INTEGRAL

A ideia básica por trás de um controlador PI é ler um sensor, e calcular a resposta do atuador através da soma do cálculo proporcional e do cálculo integral. Este controle clássico a saída a partir o erro e da integral do erro em função do tempo. A integral no erro em função do tempo permite que saída possa acompanhar a entrada com um erro muito menor, chegando a zero, em alguns casos. Entretanto, este pode também instabilizar o sistema, se a ação integral foi muito acentuada (LIMA, 2016).

A ação proporcional produz um sinal de saída que é proporcional à amplitude do erro, usando como referência a variável Kp.

A ação integral produz um sinal de saída que trabalha com o erro em função do tempo, ou seja, o erro acumulado, usando como referência a variável Ki (NATIONAL INSTRUMENTS, 2016).

Os valores das constantes do controlador podem ser calculados utilizando vários métodos, e podem ser escritos na forma da equação (1):

$$\mu(t) = Kp\, e(t) + Ki \int_0^t e(t)dt \qquad (1)$$

2.6 TRABALHOS RELACIONADOS

Em seu trabalho, Huang (2000) afirma que a velocidade com que as comunicações wireless têm crescido nos últimos 10 anos tem surpreendido muitos na indústria de semicondutores, e sem dúvida tornar-se-á pioneiro deste ramo. Hoje, os transceptores de radiofrequência tem um papel muito mais importante do que antigamente, pois este assume atualmente uma posição crítica na comunidade de circuitos e sistemas.

Guerreiro e Oliveira (2003) realizaram em seu trabalho o controle de potência de motores DC utilizando um PIC16F876. Foi feito o controle de potência de 2 motores ligados a um veículo, com comunicação RS232 wireless entre o microcontrolador e o veículo. Para a interpretação da posição angular do motor, foram utilizados encoders.

Segundo Huang (2000) e Larson (1998), dispositivos de transmissão radiofrequência de baixo consumo para circuitos digitais, como o utilizado neste trabalho, podem ter diferenças operacionais ao alterar o consumo da corrente, sendo preciso modificar o planejamento e seleção da arquitetura do sistema como um todo.

3 MATERIAIS E MÉTODOS

Neste capítulo serão discutidas as ferramentas, os materiais utilizados na confecção do kit e os métodos para a elaboração do programa e implementação do sistema.

3.1 FERRAMENTAS

3.1.1 Codewarrior®

O Codewarrior® é um um *Integrated Development Environment* (IDE), ou Ambiente Integrado para Desenvolvimento de Software, de compilação para microcontroladores e sistemas embarcados da Freescale®. Nele é possível desenvolver firmwares, códigos em linguagem C e Assembly, alterar valores dos componentes da placa, como por exemplo o valor de *clock* através da tecnologia *Processor Expert* existente nessa IDE. A versão utilizada foi a 6.3 .

3.1.2 MATLAB®

O MATLAB® é um programa de cálculo numérico que pode ser usado interativamente. A sua estrutura de dados fundamental é a matriz, que pode ter elementos reais ou complexos. Embora na sua versão base o MATLAB já possua um vasto conjunto de funções de carácter genérico, existem várias bibliotecas de funções adicionais (designadas por toolboxes) que expandem as suas capacidades em domínios de aplicação mais específicos (MATOS, 1999). A versão utilizada foi a 2012a.

3.1.3 Simulink

O Simulink é um pacote do MATLAB® usado para modelar, simular e analisar sistemas dinâmicos. Ele suporta sistemas lineares e não-lineares modelados em tempo contínuo, tempo discreto ou ambos (BAPTISTA, 2008).

Para modelagem, o Simulink possui uma interface gráfica para construir os modelos como diagramas de blocos.

3.2 EQUIPAMENTOS

3.2.1 Microcontrolador MC9S08QE128

O microcontrolador utilizado para realizar o controle do motor CC foi o MC9S08QE128 de 8 bits da Freescale®. Esse microcontrolador está acoplado em uma placa de desenvolvimento DEMOQE128 (figura 3). Abaixo algumas das principais características do MC9S08QE128 (FREESCALE..., 2008):

- Mais de 50.33-MHz HCS08 CPU de 3,6 V a 2.1 v, e 20-MHz CPU de 2.1 V a 1.8 v;
- Atinge uma faixa de temperatura entre -40ºC e 85ºC;
- Suporte para até 32 interrupções/reinicialização de fontes;
- Oscilador (XOSC) - Controle de loop Pierce oscilador, cristal ou ressonador cerâmico gama de 31,25 KHz a 38,4 KHz ou 1 MHz a 16 MHz;
- Fonte de *Clock* Interno (*Internal Clock Source - ICS*) com módulo contendo uma frequência de loop fechado (FLL), controlado por referência interna ou externa;
- Precisão de corte de referência interna permite a resolução abaixo de 0,2% e desvio de 2% sobre a temperatura e a tensão;
- Suporta a frequência da CPU entre 2 MHz e 50,33 MHz;
- Conversor Analógico-Digital com 24 canais, 12 bits de resolução, até 2,5 µs no tempo de conversão e funciona de 3,6V à 1,8V;

3.2.2 DEMOQE128

A DEMOQE128 (Figura 3) é uma placa de desenvolvimento de baixo custo desenvolvida pela PE Micro que foi projetada para demonstração, avaliação e depuração dos processadores Freescale MC9S08QE128 e MCF51QE128. Esta placa oferece os seguintes recursos (DEMOQE128..., 2016):

- P&E USB Multilink, circuito embarcado que elimina a necessidade de um cabo BDM externo;
- Permite depuração, programação, alimentação, para a placa e dispositivos;
- Sinal SCI conectados ao Multilink através dos jumpers para a virtualização da porta serial on-board.
- Sinal TPM conectados ao Multilink através dos jumpers para a análise lógica on-board.
- Acelerômetro 3-eixos;
- Oito LEDs para usuário;
- Quatro *push buttons*;
- Uma sirene Piezzo;
- Uma porta serial RS-232 com conector DB9-F;
- Resistores IIC *pullups*;
- POT 10k Ohm.
- Alimentação flexível através da seleção pelos jumpers: USB Cable (5VDC, 500mA max), 5VDC à 10VDC pelo conector cilíndrico 2.5/5.5mm, positivo no centro, duas pilhas AAA, conector da porta MCU.
- Interruptor de alimentação ON/OFF com LED indicador;
- Regulador de tensão de saída VDD de 3.0V ou 2.1V
- Push button RESET e LED indicador;
- Cristal externo com circuito completo.

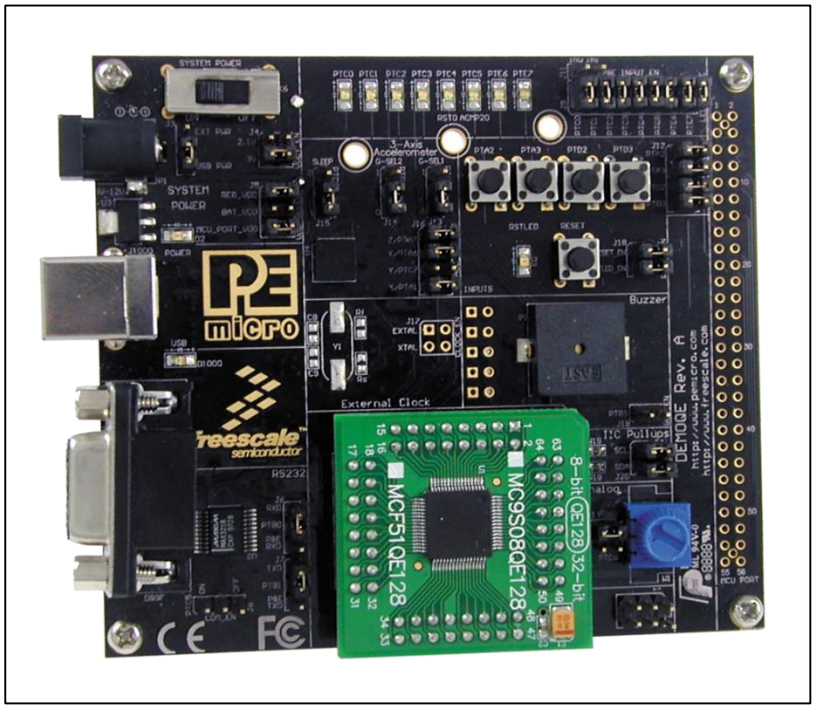


Figura 3 - Placa DEMOQE128 com MC09S08QE128.
Fonte: Bitbucket (2014).

3.2.3 Servomotor

O servomotor utilizado neste trabalho foi um motor CC da marca Pittman, com tensão máxima de 19,1V e rotação máxima de 10.000 rpm.

A velocidade do motor é calculada através da equação (2):

$$V(rpm) = \frac{60}{2048 * \Delta t} * \Delta Counts \qquad (2)$$

A constante 60 é a conversão de segundos para minutos. A constante divisória 2048 é resultando da resolução do *encoder* (512 linhas) multiplicado por 4, pois temos um decodificador de quadratura ligado a este. Temos também o Δt, que é a taxa de amostragem do Simulink, e $\Delta Counts$, que é a variação entre as contagens efetuadas do encoder/decodificador de quadratura. As Figuras 4 e 5 mostram a bancada do motor.

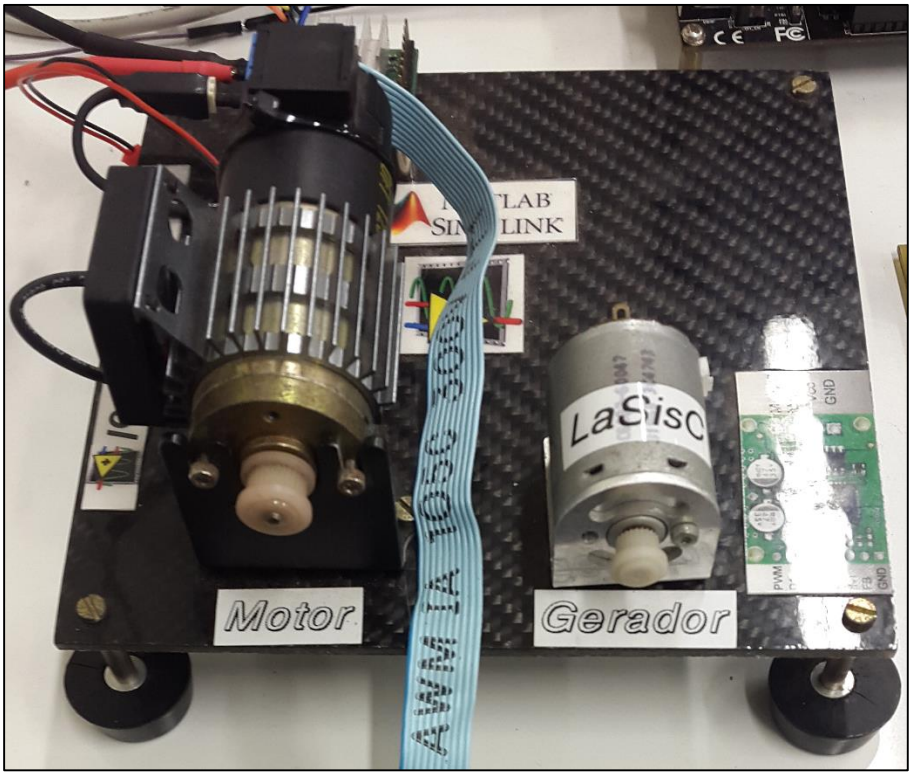


Figura 4 - Bancada do servomotor com foco no motor.
Fonte: Autoria própria.

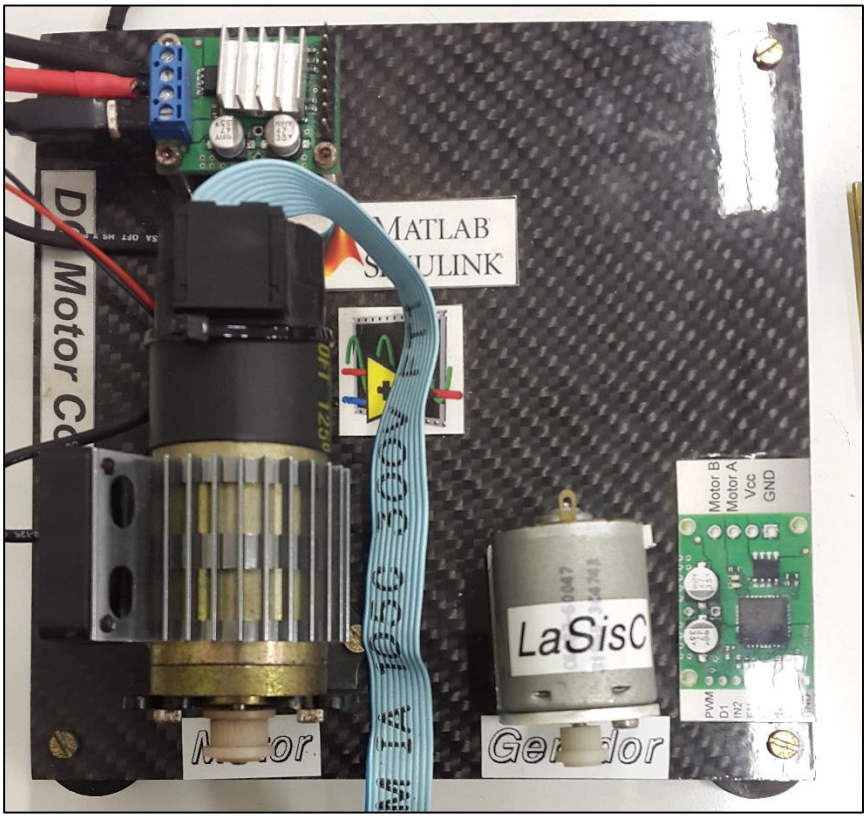


Figura 5 - Vista superior da bancada do servomotor.
Fonte: Autoria própria.

3.2.4 Driver de potência

O motor é alimentado por um driver de potência fabricado pela Pololu. O driver e seus respectivos jumpers é mostrado na Figura 6.

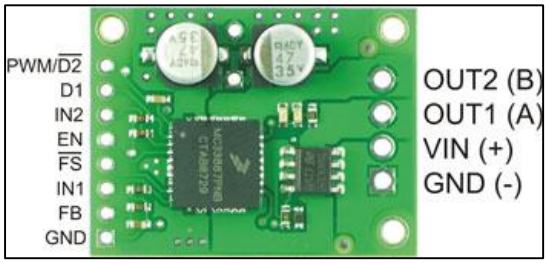


Figura 6 - Esquema do driver de potência.
Fonte: REPINALDO (2014).

A saída OUT2 é o canal B e OUT1 é o canal A do decodificador de quadratura. A alimentação da placa (VIN e GND) é feita por uma bateria de 14V e 3000mA.

Para que o motor opere no modo *forward* (sentido para frente), a configuração desses pinos deve seguir a Tabela 1. Na Figura 5 é possível ver na bancada do motor o driver, que fica logo atrás do servomotor.

Tabela 1 - Tabela verdade do driver de potência

Device State	Input Conditions				Fault Status Flag		Output States	
	EN	D1	D2	IN1	IN2	FS	OUT1	OUT2
Forward	H	L	H	H	L	H	H	L
Reverse	H	L	H	L	H	H	L	H

Fonte: Freescale Semiconductor (2012).

3.2.5 Encoder HEDS-5310

O encoder utilizado no projeto foi o HEDS-5310 da Avago. Ele possui 512 linhas e 3 canais. Ele está acoplado ao motor.

3.2.6 Decodificador de quadratura HCTL-2022

Esta placa com o decodificador de quadratura foi desenvolvida no LaSisC para simplificar o uso do próprio decodificador (Figura 7). O decodificador de quadratura recebe o sinal de um *encoder* e transforma esses dados para serem enviados para o microcontrolador. O decodifcador tem como entrada o sinal do *encoder* e como saída dois canais (A e B), recuperando o sentido de movimento e quadruplicando a resolução do *encoder*.

Este decodificador está acoplado a um contadores de 32 bits que armazena a contagem de pulsos entre leituras. Os pinos utilizados neste decodificador e suas respectivas descrições são:

- X/Y : Seleciona os dados do primeiro ou do segundo eixo para serem lidos. Nível lógico baixo habilita o primeiro eixo (X), enquanto nível alto habilita o segundo eixo (Y).
- OEN: Esse CMOS ativo de nível baixo habilita os *buffers* de saída *tri-state*. SEL1, e as entradas OE/SEL2 são recolhidos pela lógica de inibição interna na borda de descida do *clock* para controlar o carregamento dos dados.
- EN1 e EN2: Esses pinos são configurados em nível lógico alto ou baixo para selecionar o modo de contagem de acordo com a Tabela 2.
- SEL1 e SEL2: Esse CMOS controla diretamente quais *bytes* de dados que serão transmitidos no *buffer* de 8 bits de saída. O modo de seleção é dado de acordo com a Tabela 3.
- RSTNx e RSTNy: Essas entradas ativas no nível lógico baixo limpam o contador de posição interna e a trava de posição e também redefinme a lógica de inibição. Ambos são assíncronos em relação à qualquer outro sinal de entrada. RSTx reseta o primeiro eixo do contador e RSTy reseta o segundo eixo do contador.

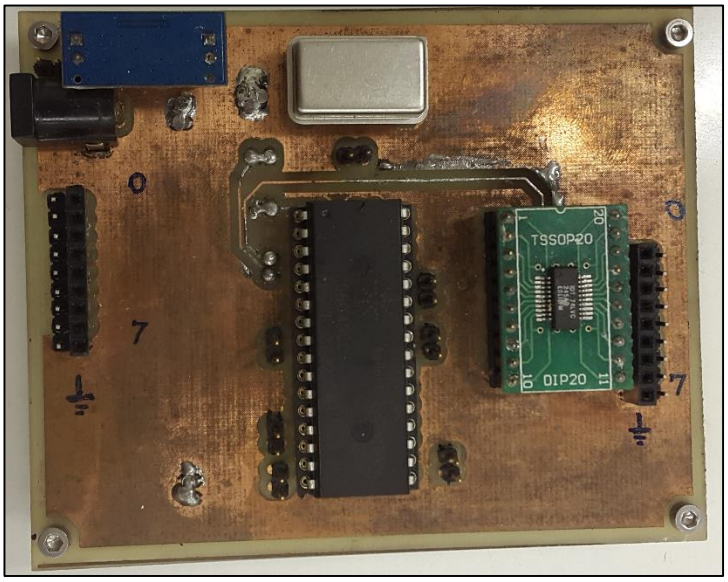


Figura 7 - Placa com decodificador de quadratura feita.
Fonte: Autoria própria.

Tabela 2 - Modo contagem EN1 e EN2

		Count Modes		
EN1	EN2	4x	2x	1x
0	0		Illegal Mode	
1	0	On		
0	1		On	
1	1			On

Fonte: Avago Technologies (2014).

Tabela 3 - Modo de seleção SEL1 e SEL2

		Byte Selected			
SEL1	SEL2	MSB	2ND	3RD	LSB
0	1	D4			
1	1		D3		
0	0			D2	
1	0				D1

Fonte: Avago Technologies (2014).

3.2.7 Módulo comunicação rádio frequência XL02-232AP1

O XL02-232AP1 (Figura 8) é um módulo de transmissão com interface UART, que trabalha a até 433 MHZ de frequência. Esse módulo é desenvolvido para todos os tipos de comunicações wireless entre portas seriais, como computador, microcontrolador, todo tipo de maquinaria e equipamentos de portas seriais. Suas especificações são:

- Distância máxima de comunicação de 300 metros;
- Frequência de trabalho de 433-435 MHz (Padrão 433.92MHz);
- ID pode ser ajustada entre 0-65535. (Padrão: 12345);
- Faixa serial de 1.2k-39.4 KBPS (Padrão 9.6 KBPS);
- Formato de dados 8N1;
- Modo FSK;
- Trabalho máximo de emissão: 13 dBm
- Sensibilidade de recepção: -110 dBm @ 50 KBPS
- Tensão de trabalho: 5V;
- Temperatura de trabalho: -30 à 60 °C.

Figura 8 - Módulo XL02-232AP1.
Fonte: Autoria própria.

4 RESULTADOS

4.1.1 Montagem do experimento

Usando os equipamentos citados na seção acima, montamos o experimento para que a comunicação e o controle pudessem ser realizados, como mostra a Figura 9.

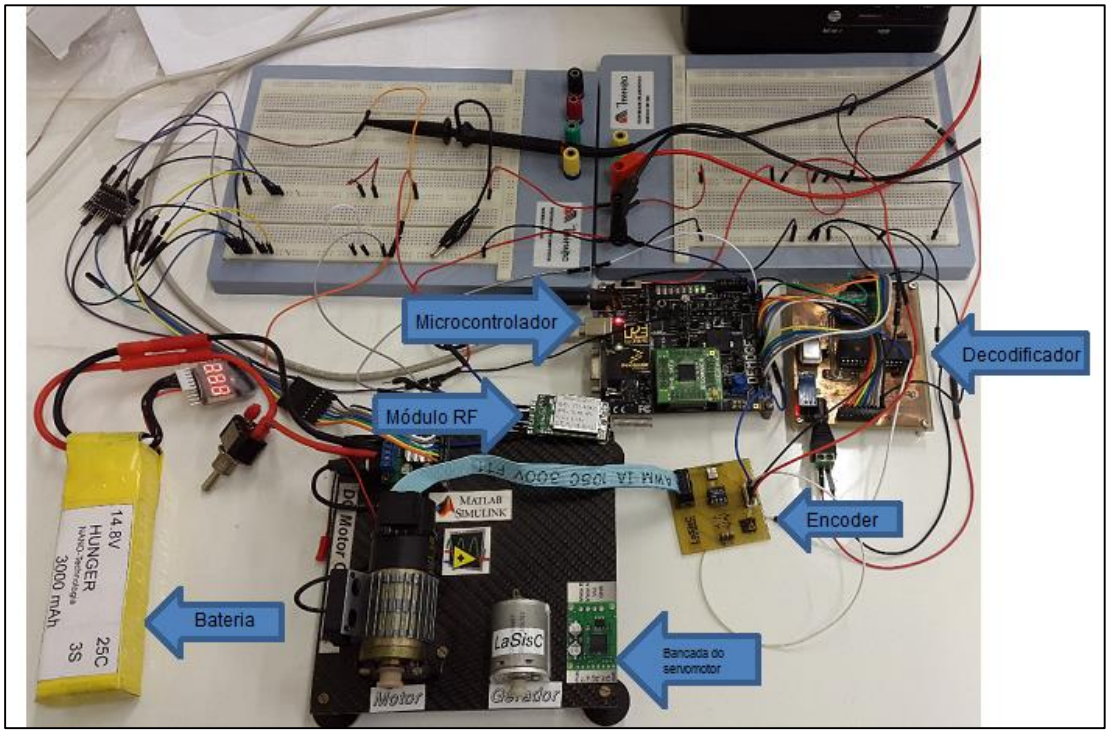


Figura 9 – Microcontrolador e Servomotor com os periféricos conectados.
Fonte: Autoria própria.

No outro nó temos o computador, que também está utilizando um módulo RF. A Figura 10 mostra o módulo RF ligado a um adaptador serial, para que este possa ser conectado na porta serial do computador (Figura 11).

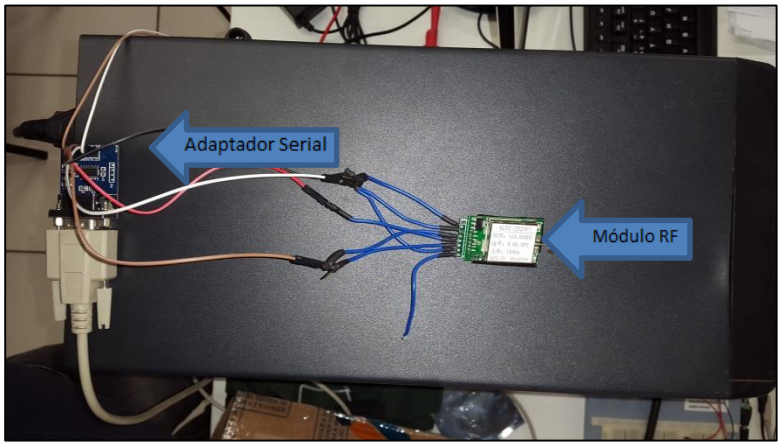


Figura 10 - Módulo RF e adaptador serial.
Fonte: Autoria Própria.

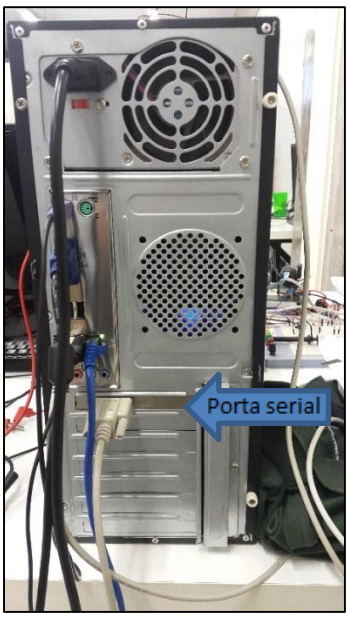


Figura 11 - Porta serial do computador.
Fonte: Autoria própria.

A Figura 12 mostra a fonte utilizada para alimentação dos equipamentos.

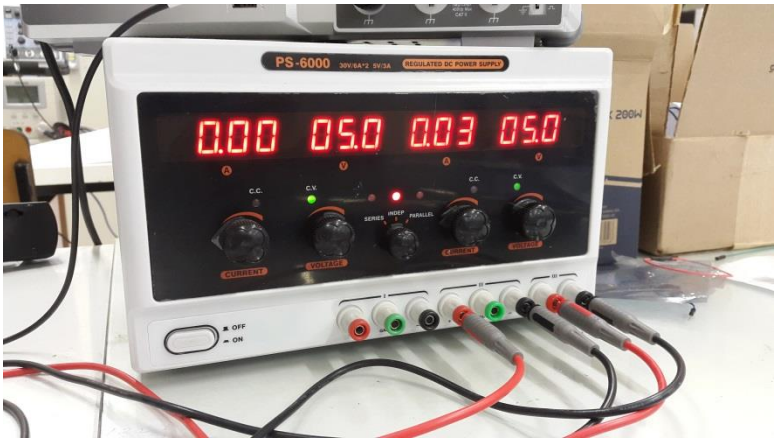


Figura 12 - Fonte de alimentação.
Fonte: Autoria própria.

As portas de configurações do decodificador de quadratura XY, OEN, EN1, EN2, SEL1, SEL2, RSTx, RSTy foram conectadas nas portas D0, D1, D2, D3, D4, D5, D6, e D7, respectivamente, da DEMOQE. O retorno do decodificador de quadratura com a posição angular está conectada no PORTF de 0 a 7. A saída do sistema, que é o PWM que controla a velocidade do motor, está configurado na porta A0. A entrada Rx do módulo RF está conectada na porta B0, e a entrada Tx está na porta B1. Todas as portas seguem o esquema referenciado na Figura 13.

VDD	1	2	PTA5/IRQ/ TPM1CLK /RESET
VSS	3	4	PTA5/IRQ/TPM1CLK/RESET
PTB1/KBI1P5/TxD1/ADP5	5	6	PTA4/ACMP1O/BKGD/MS
PTB0/KBI1P4/RxD1/ADP4	7	8	PTE7/TPM3CLK (n/c for 32 LQFP)
PTA2/KBI1P2/SDA1/ADP2	9	10	VREFH
PTA3/KBI1P3/SCL1/ADP3	11	12	VREFL
PTC0/TPM3CH0	13	14	PTA0/KBI1P0/TPM1CH0/ADP0/ACMP1+
PTC1/TPM3CH1	15	16	PTA1/KBI1P1/TPM2CH0/ADP1/ACMP1-
PTB3/KBI1P7/MOSI1/ADP7	17	18	PTF0/ADP10 (n/c for 32 LQFP)
PTB4/TPM2CH1/MISO1	19	20	PTF1/ADP11 (n/c for 32 LQFP)
PTB2/KBI1P6/SPSCK1/ADP6	21	22	PTA6/TPM1CH2/ADP8
PTB5/TPM1CH1/SS1	23	24	PTA7/TPM2CH2/ADP9
PTD1/KBI2P1/MOSI2	25	26	PTH6/SCL2 (n/c for 32 LQFP)
PTD2/KBI2P2/MISO2	27	28	PTH7/SDA2 (n/c for 32 LQFP)
PTD0/KBI2P0/SPSCK2	29	30	PTD4/KBI2P4 (n/c for 32 LQFP)
PTD3/KBI2P3/SS2	31	32	PTD5/KBI2P5 (n/c for 32 LQFP)
PTC2/TPM3CH2	33	34	PTD6/KBI2P6 (n/c for 32 LQFP)
PTC3/TPM3CH3	35	36	PTD7/KBI2P7 (n/c for 32 LQFP)
PTC4/TPM3CH4/RSTO	37	38	PTC7/TxD2/ACMP2-
PTC5/TPM3CH5/ACMPO	39	40	PTC6/RxD2/ACMP2+
(n/c for 32 LQFP) PTF2/ADP12	41	42	PTB7/SCL1/EXTAL
(n/c for 32 LQFP) PTF3/ADP13	43	44	PTB6/SDA1/XTAL
(n/c for 32 LQFP) PTF4/ADP14	45	46	PTG0 (n/c for 32 LQFP)
(n/c for 32 LQFP) PTF5/ADP15	47	48	PTG1 (n/c for 32 LQFP)
(n/c for 32 LQFP) PTF6/ADP16	49	50	PTH0 (n/c for 32 LQFP)
(n/c for 32 LQFP) PTF7/ADP17	51	52	PTH1 (n/c for 32 LQFP)
(n/c for 32 LQFP) PTG2/ADP18	53	54	PTE6 (n/c for 32 LQFP)
(n/c for 32 LQFP) PTG3/ADP19	55	56	NC

Figura 13 - Jumpers da DEMOQE.
Fonte: Freescale (2007).

4.1.2 Firmware

O *firmware* (rotina ou código-fonte) foi desenvolvido utilizando o Codewarrior®, por se tratar de uma ferramenta própria para microcontroladores Freescale®. No *firmware* deste projeto está contida a biblioteca de controle e leitura do decodificador de quadratura, as diretrizes para a comunicação RS-232, e a configuração e ajuste do PWM através dos dados recebidos. O código fonte deste projeto encontra-se anexado no apêndice A. Basicamente, o programa atua em função do controle efetuado pelo Simulink. A Figura 14 ilustra este processo. O microcontrolador requesta uma leitura da posição angular (passo 1 – Figura 14). O motor então passa para o *encoder* os dados necessários para se fazer essa leitura (passo 2– Figura 14). O decodificador recebe esse dado (passo 3 – Figura 14), faz a decodificação, e transmite este dado para o microcontrolador (passo 4 – Figura 14).

O microcontrolar passa então esse valor para o Simulink (passo 5 – Figura 14), que aplica o controle PI de acordo com a velocidade do motor, e então calcula um novo valor de *duty cycle* para ser aplicado no PWM do motor. Este dado é passado para o microcontrolador (passo 6 – Figura 14), que transforma isso em PWM para o motor (passo 7 – Figura 14). Os passos 5 e 6 da Figura 14 são feitos de forma *wireless*. Isto se repete até que o microcontrolador ou o controle seja desligado.

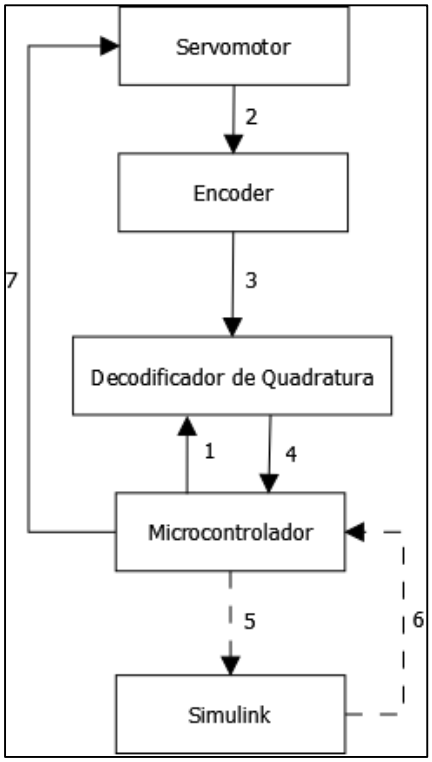


Figura 14 - Funcionamento do sistema em geral.
Fonte: Autoria própria.

Os sinais enviados por cada elemento estão demonstrados na Figura 15. O encoder envia a contagem de pulsos (passo 1 – Figura 15); o microcontrolador envia através da RS232 o valor para o Simulink (passo 2 – Figura 15); o Simulink o novo valor de *duty cicle* para o microcontrolador (passo 3 – Figura 15); o microcontrolador envia o novo PWM para o motor (passo 4 – Figura 15); o motor altera sua velocidade, e assim por diante. Os passos 2 e 3 são realizados de forma *wireless*.

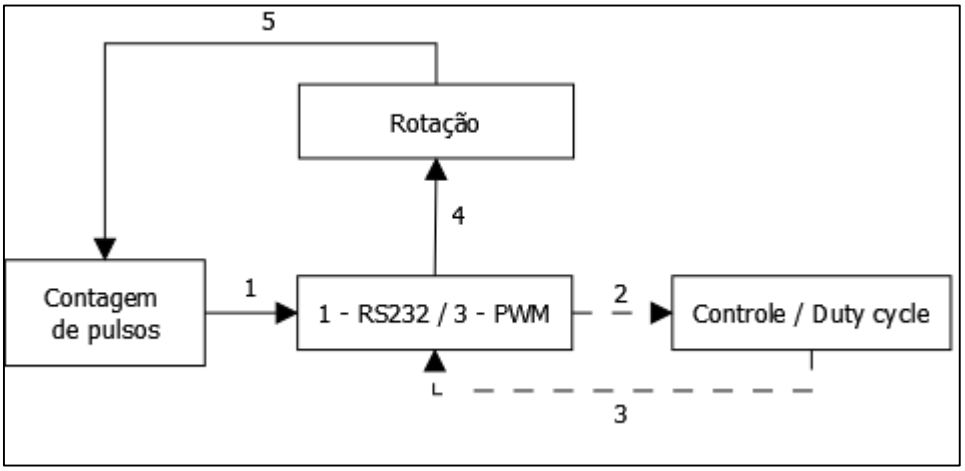


Figura 15 - Sinais enviados pelo sistema.
Fonte: Autoria própria.

O sinal PWM deste projeto foi configurado usando a ferramenta *Processor Expert*. A onda gerada pelo PWM tem um período de 5 ms, e os passos aceitados por essa ferramenta, dado essa limitação de período, são de 0,2384186 µs. Em outras palavras, para se alterar o valor do PWM, o valor colocado precisa ser proporcional ao passo definido pelo *Processor Expert* (0,2384186 µs) Este módulo de PWM, trabalha com o tempo invertido, ou seja, se o PWM é configurado para ter um *duty cycle* de 5 ms, a resposta é 0%. Se o PWM está configurado para um *duty cycle* com 0 ms, a resposta é 100%.

4.1.3 Ação de controle – Simulink

O programa desenvolvido para efetuar o controle do PWM enviado ao microcontrolador foi feito usando o Simulink. O controle usado foi o proporcional-integral.

O programa desenvolvido no Simulink trabalha com um uma taxa de amostragem de 40Hz. Esta taxa foi obtida através de experimentos com o projeto, sendo esta a máxima obtida sem perda de dados. A imagem com todo o processo de controle feito no Simulink está no apêndice B. Para um melhor esclarecimento, o controle foi quebrado em blocos, mostrados abaixo. A Figura 16 mostra onde os dados são coletados, ou seja, a entrada do sistema.

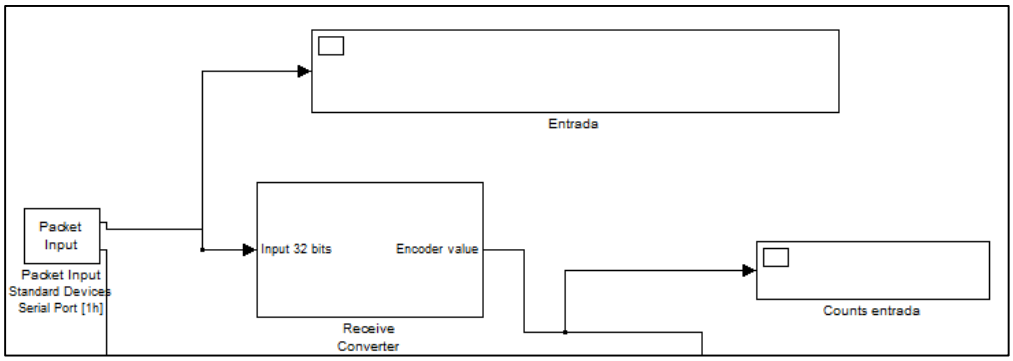


Figura 16 - Entrada dos dados no Simulink.
Fonte: Autoria própria.

A partir deste dado de entrada, a velocidade em rpm do motor é calculada. A Figura 17 mostra o bloco onde é feito esse cálculo, segundo a equação (2) .

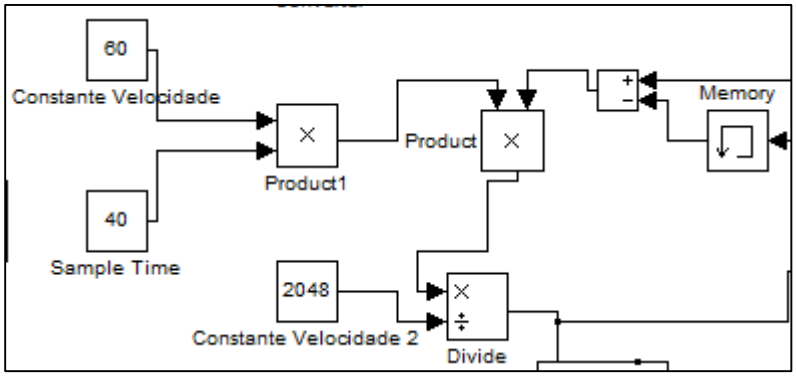


Figura 17 – Cálculo da velocidade em rpm no Simulink.
Fonte: Autoria própria.

As ondas usadas como entradas de referência são mostradas na Figura 18, onde um *switch* é utilizado para trocar de quadrada para senoidal, ou senoidal para quadrada por exemplo. Existe ali também o modo de gerar um sinal constante.

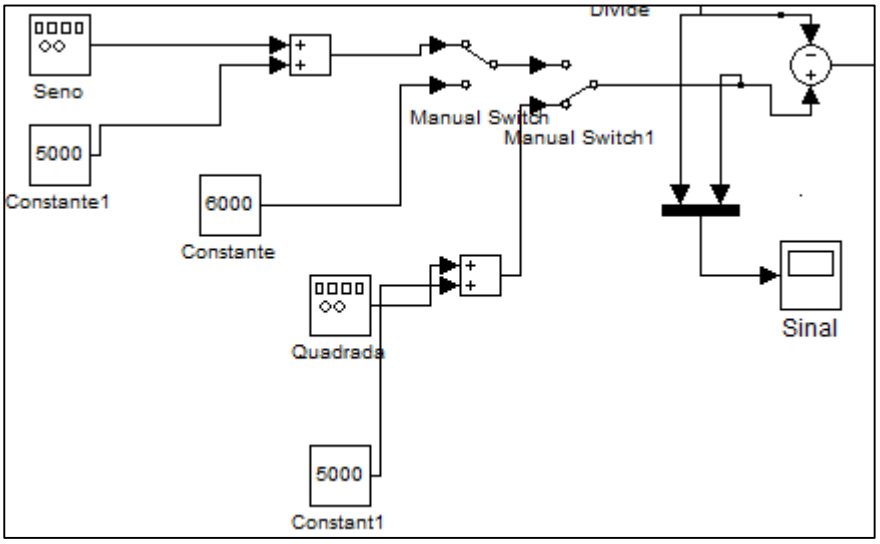


Figura 18 - Gerador de ondas no Simulink.
Fonte: Autoria própria.

O bloco onde o controle PI é efetuado está na Figura 19. Na parte superior está o ganho proporcional Kp, e na inferior o ganho integral Ti.

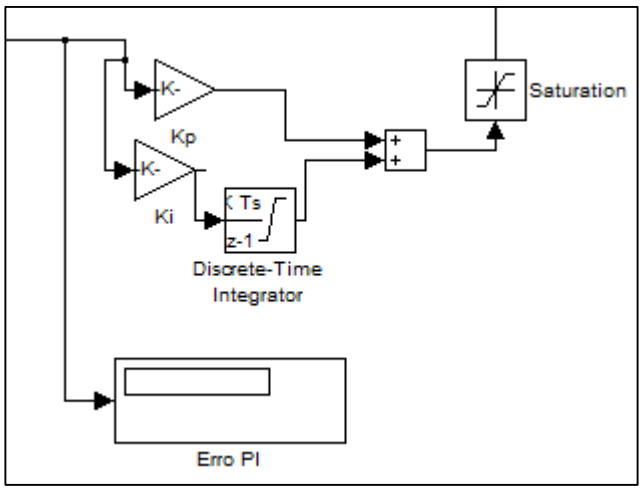


Figura 19 - Controle PI no Simulink.
Fonte: Autoria própria.

A saída do sistema, que é enviada para o microcontrolador, está na Figura 20.

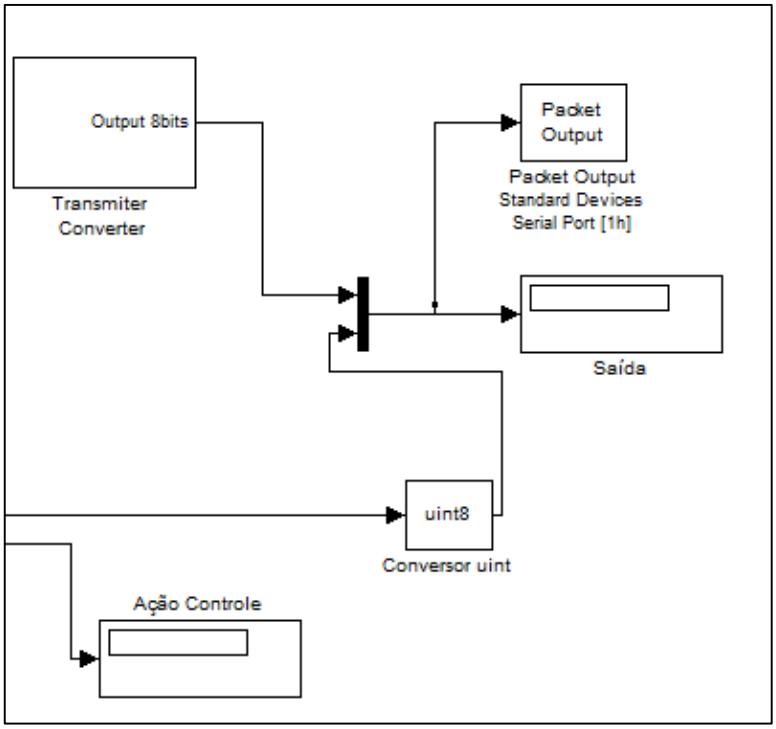


Figura 20 - Saída do sistema no Simulink.
Fonte: Autoria própria.

Com base nos estudos dos conceitos relacionados com controle clássico, foram realizados testes para o ajuste dos ganhos que possibilitassem um desempenho satisfatório do sistema.

Um método empírico foi utilizado para aquisição dos melhores ganhos Kp e Ki para o sistema de controle PI, que embora seja pouco científico, para este projeto apresentou resultados satisfatórios. Este método consiste nas seguintes etapas (REPINALDO, 2015):

a) Aumentar o ganho proporcional, zerando o ganho integral, até a saída responder as mudanças de *setpoint*.
b) Inserir ganho integral até que a saída responda com baixos *overshoots*.

Os ajustes do ganhos foram realizados baseados em duas curvas, uma quadrada e uma senoidal, ambas em relação a velocidade do motor. A onda senoidal tem por características uma amplitude de 3000 rpm, *offset* de 5000 rpm e uma frequência de 0,15 Hz. A onda quadrada tem por características uma amplitude de 1500 rpm, *offset* de 5000 rpm e frequência de 0,2 Hz.

A primeira taxa de amostragem utilizada foi de 45 Hz, pois taxas maiores se mostraram ineficazes. Entretanto, a comunicação entre o microcontrolador e o Simulink simplesmente não aconteceu. Nenhum dado era recebido, e nem transmitido. Sendo assim, mudamos a frequência para 45Hz. O ganho proporcional utilizado foi Kp = 0,03 e o ganho integral Ki = 0,009. A resposta à referencia quadrada pode ser observada na Figura 21.

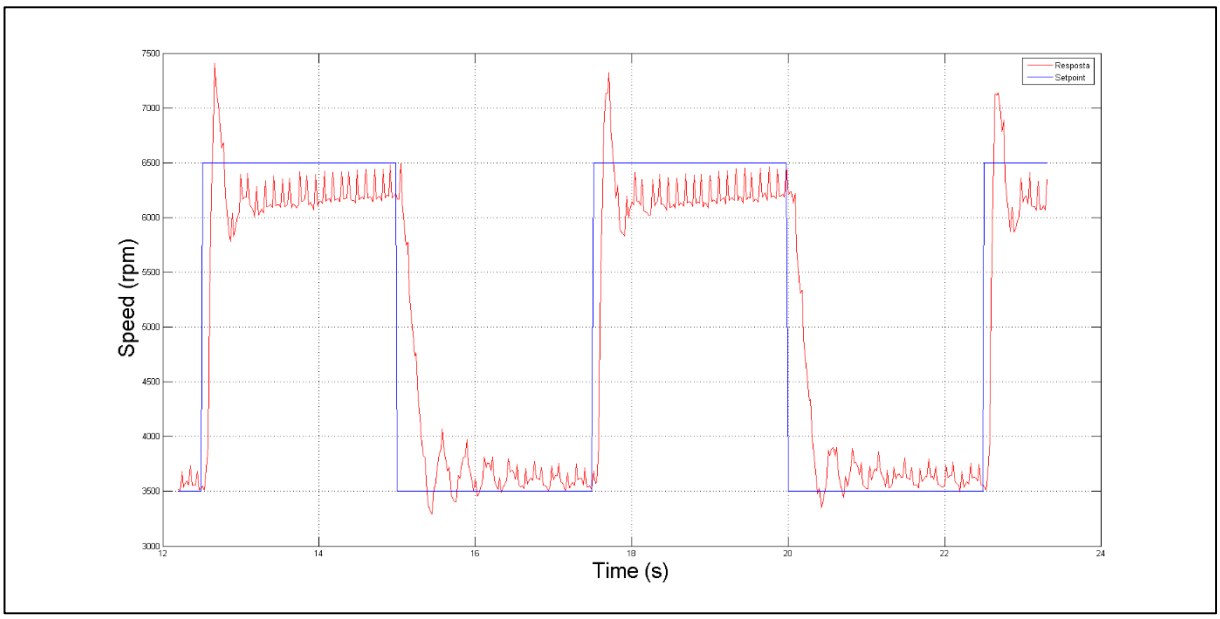


Figura 21 - Onda quadrada para taxa de amostragem de 45 Hz.
Fonte: Autoria própria.

Pode-se observar vários picos na resposta do sistema, que é o resultado de uma falha na comunicação entre o microcontrolador e o Simulink. Ao tentar alterar os ganhos Kp e Ki, o sistema tornou-se totalmente instável, com picos ainda maiores e totalmente desproporcionais ao *setpoint*. O mesmo se observa na curva senoidal, mostrada na Figura 22.

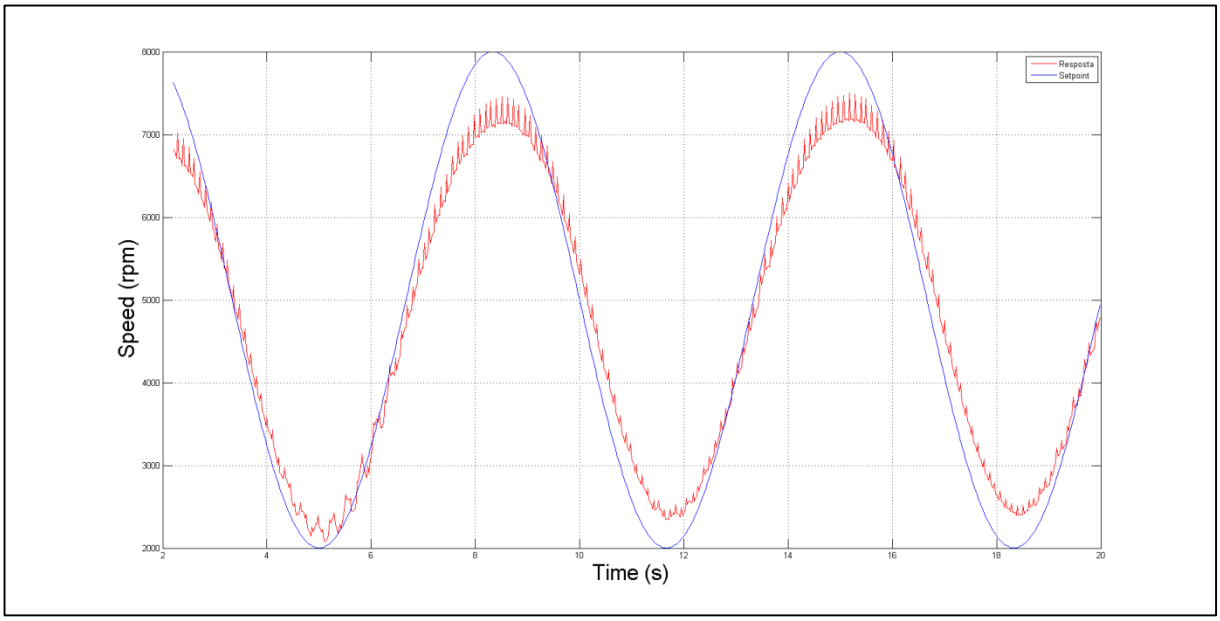


Figura 22 – Onda senoidal para taxa de amostragem de 45 Hz.
Fonte: Autoria própria.

Sendo assim, alteramos a taxa de amostragem para 40 Hz. O sistema ficou muito próximo do *setpoint*, respondendo satisfatoriamente com um baixo *overshoot*. Para tal frequência, foi utilizado um ganho proporcional Kp = 0,018, e o ganho integral Ki = 0,02. A Figura 23 mostra a onda quadrada para tal taxa.

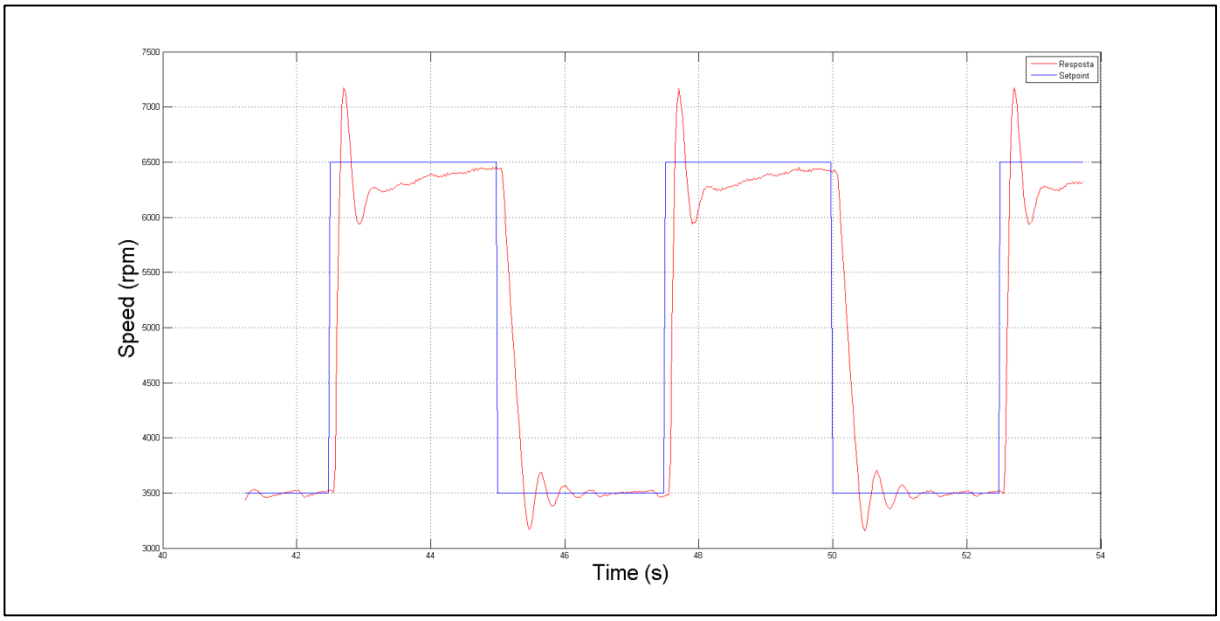


Figura 23 - Onda quadrada para taxa de amostragem de 40 Hz.
Fonte: Autoria própria.

A melhora na resposta do sistema é nítida. Os picos existentes durante a ação de controle e resposta praticamente não existem quando a frequência é alterada. Em relação aos *overshoots* o sistema não mudou significativamente, mas a estabilização da velocidade foi muito melhor. Atribuímos os picos existentes nos 45 Hz à perda de dados existente quando a frequência está muito próxima dos 50 Hz. A melhora é vista também na curva senoidal, mostrada na Figura 24.

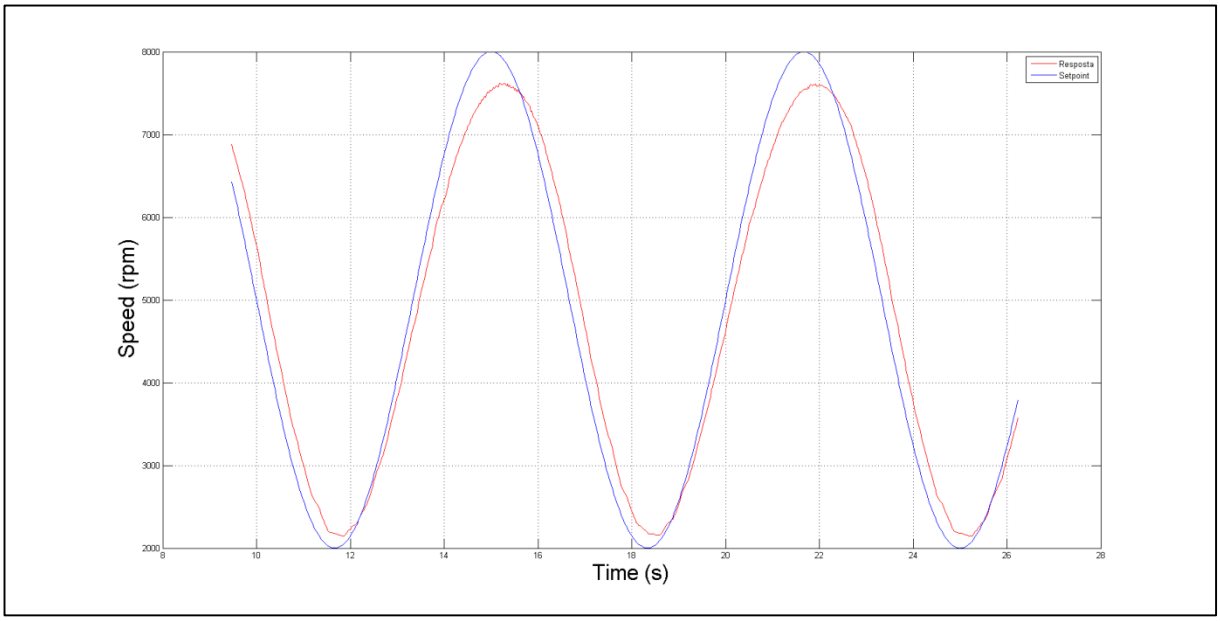


Figura 24 - Onda senoidal para taxa de amostragem de 40 Hz.
Fonte: Autoria própria.

A resposta senoidal ficou mais suavizada. Não houve perda de dados, e consequentemente a curva ficou mais próxima do *setpoint*. Em ambas curvas senoidais pode ser observado que a velocidade máxima do motor neste projeto foi de 7500 rpm.

5 CONCLUSÃO

Este trabalho contribuiu para o aprendizado prático do aluno, como a programação de microcontroladores, interface e bibliotecas para circuitos integrados, programação e controle utilizando Matlab/Simulink. Como resultado, foi construído um kit didático para controle de velocidade de um servomotor, utilizando um sistema de controle baseado no controle clássico PI, que estruturado pelos ganhos proporcional e integral. Com o auxílio do *software* Simulink, foi possível criar esse sistema de controle, onde a alteração dos ganhos é fácil de se fazer, sendo muito intuitiva e muito prática.

Os objetivos traçados foram alcançados com sucesso. Embora a taxa de amostragem não tenha sido muito alta (40 Hz), o sistema respondeu satisfatoriamente a esta frequência.

Como trabalhos futuros, sugere-se alterar o módulo de comunicação wireless, para que uma taxa maior de amostragem seja alcançada. Pode-se utilizar *bluetooth*, ou até mesmo um outro sistema radiofrequência. Outro detalhe que pode ser testado para aumentar a taxa de transferência é mudar o tipo de entrada, que neste projeto são 32 bits de um inteiro, para 16 bits de um inteiro, usando apenas os 16 bits menos significativos, pois assim, o microcontrolador ganhará tempo que era gasto transmitindo 16 bits a mais do que o necessário.

Sugere-se também construir uma placa para substituir a imensa quantidade de fios utilizadas nas conexões da DEMOQE com os outros equipamentos.

REFERÊNCIAS

A história e as diferenças entre um microcontrolador e um microprocessador. Disponível em: <http://www.cerne-tec.com.br/Artigo3_Historia.pdf>. Acesso em: 20 maio 2014.

ALMEIDA, Felipe Biriba de. **PADRONIZAÇÃO DA COMUNICAÇÃO SEM FIO EM AMBIENTES INDUSTRIAIS – ISA SP100.** 54 f. Tese (Doutorado) - Curso de Curso de Especialização em Automação Industrial Com ênfase em Informática Industrial, Instrumentação, Controle e Otimização de Processos Contínuos., Universidade Federal da Bahia, Salvador, 2009. Disponível em: <http://www.pei.ufba.br/doc/monografias/Monografia_versao_final_Felipe_Biriba.pdf >. Acesso em: 20 jun. 2016.

AVAGO TECHNOLOGIES. **HCTL-2032, HCTL-2032-SC, HCTL-2032-SCT, HCTL-2022.** 2014.

BAPTISTA, Luis Filipe. **Manual de Instrução ao MATLAB/Simulink**. Paço de Arcos, Portugal. ENIDH, 2008. 29 p.

DEMOQE128: MCF51QE128 Demonstration Board. Disponível em: <http://www.nxp.com/products/sensors/touch-sensors/mcf51qe128-demonstration-board:DEMOQE128> . Acesso em: 06 abril 2016.

DFROBOT. Bluetooth 2.0 Module V3 for Arduino. Disponível em: <http://www.dfrobot.com/index.php?route=product/product&product_id=360#.Vyewi_krKM8>. Acesso em: 02 maio 2016.

FREESCALE SEMICONDUCTOR. **5.0 A H-Bridge with Load Current Feedback.** 2012. 37 p. Disponível em: <https://www.sparkfun.com/datasheets/Robotics/MC33887.pdf>. Acesso em: 01 abril 2016

FREESCALE SEMICONDUCTOR. **Mc9s08qe128rm: Hcs08 Microcontrollers Reference Manual**. 2007. Disponível em: < http://www.nxp.com/doc/MC9S08QE128RM>. Acesso em: 11 nov. 2015

FREESCALE SEMICONDUCTOR. **MC9S08QE128 Series Data Sheet**. 2008.
GUERREIRO, Júlio; OLIVEIRA, Luís. **Sistema de Controlo de Potência de Motores DC**. 2003. 58 f. TCC (Graduação) - Curso de Engenharia Electrónica e Computadores, Escola Superior de Tecnologia Setúbal, Setúbal, 2003.

HUANG, Qiuting. **Radio frequency transceivers for wireless communications**. Circuits and Systems, 2000. IEEE APCCAS 2000. The 2000 IEEE Asia-Pacific Conference, p.859-863, 2000.

LARSON, Lawrence E.. **Radio frequency integrated circuit technology for low-power wireless communications**. Personal Communications, IEEE , vol.5, no.3, p.11-19, Jun 1998.

LIMA, Luis Eduardo Martins de. **Sistemas Embarcados:** Vitória, 2016. 37 slides. Disponível em: <ftp://ftp.ifes.edu.br/cursos/EngenhariaEletrica/LuisEduardo/Sistemas Embarcados/SE 5 - Controladores PI PD e PID.pdf>. Acesso em: 25 maio 2016.

MATOS, Aníbal Castilho Coimbra de. **Introdução ao MATLAB**. Porto. Feup, 1999. Disponível em: <http://paginas.fe.up.pt/~anibal/matlab/matlab_intro.pdf>. Acesso em: 01 jun. 2014.

MECATRÔNICA ATUAL. **Encoders: Saiba como funcionam os sensores mais usados na automação industrial**. São Paulo, 2013. Disponível em: <http://www.mecatronicaatual.com.br/educacao/1689-encoders-saiba-como-funcionam-os-sensores-mais-usados-na-automao-industrial>. Acesso em: 02 jun. 2014.

NATIONAL INSTRUMENTS. **Conceitos Gerais de Comunicação Serial**. Disponível em: <http://digital.ni.com/public.nsf/allkb/32679C566F4B9700862576A20051FE8F>. Acesso em: 05 jun. 2014.

NATIONAL INSTRUMENTS. **Introduction to RF & Wireless Communications Systems.** Disponível em: <http://www.ni.com/tutorial/3541/en/>. Acesso em: 10 maio 2016.

PANTECH SOLUTIONS. **How to Generate a PWM with LPC2148 ARM.** 2014. Disponível em: <https://www.pantechsolutions.net/microcontroller-boards/pwm-interfacing-with-lpc2148-arm7-primer>. Acesso em: 10 mar. 2014.

PEREIRA, Fábio. **HCS08 Unleashed: Designer's Guide to the HCS08 Microcontrollers**. USA: Booksurge Publishing, 2009.

P&E MICROCOMPUTER SYSTEM INC. **Demoqe128 Base Board Schematic**. 2007.

PINHEIRO, Rhafael Meassi. **Controle da velocidade de um motor de corrente continua usando microcontroladores aplicando técnicas de controle PI**. 8 p. UTFPR, Cornélio Procópio, 2013.

REPINALDO, Joana Pereira. **Controle de Velocidade de um Servomotor Utilizando Software LabVIEW Real-Time**. 2015. 80 f. Trabalho de Conclusão de Curso (Graduação) – Engenharia Mecânica. Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2015.

SÁ, J. P. Marques de. **Processamento de Sinal**. Porto. Faculdade de Engenharia da Universidade do Porto, 2003. Disponível em: <http://paginas.fe.up.pt/~jmsa/apsi/APSI-2.PDF>. Acesso em: 25 maio 2014.

SOBRINHO, André Sanches Fonseca. **IF66D/ET37I - Sistemas Microcontrolados**. 28 f. Curso de Engenharia de Computação, UTFPR, Cornélio Procópio, 2012.

TELECO. **LAN / MAN Wireless I: Redes sem Fio.** Disponível em: <http://www.teleco.com.br/tutoriais/tutorialrwlanman1/pagina_2.asp>. Acesso em: 01 jun. 2016.

TERRONES, Angel. **OS-DEMOQE128: Operating System for Freescale microcontroller MC9S08QE128.** 2013. Disponível em: <https://bitbucket.org/NHT/os-demoqe128/wiki/Home>. Acesso em: 24 maio 2016.

TOMASCHITZ, Flávio. **Controle e análise da velocidade de servos-motores de manipuladores cartesianos em uma linha automatizada de estamparia**. 2013. Monografia (Especialização em Automação industrial) – Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná, Curitiba, 2013.

VALLE, Carlos Magno Catharino Olsson. **Comunicação por radio frequência para controladores lógicos programáveis.**2013. 60 f. Tese - Curso de Engenharia de Controle e Automação, Puc-rio, Rio de Janeiro, 2013.

WUMING, Luo; PINGYANG, Han; RUILIN, Zhao. **Study on Design and Application of Wireless Sensor Network Based on Communication of Radio Frequency Identification System.** Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference, p.1-6, 2009.

YABIM, Liu.; et al. **Wireless Communication Technology Based on Bluetooth and Its Application to a Manipulator.** Industrial Informatics, 2006 IEEE International Conference, p.1251-1256, 2006.

APÊNDICE A – Código fonte

ProcessorExpert.c

```
/** ###################################################################
**     Filename    : ProcessorExpert.c
**     Project     : ProcessorExpert
**     Processor   : MC9S08QE128CLH
**     Version     : Driver 01.12
**     Compiler    : CodeWarrior HCS08 C Compiler
**     Date/Time   : 2016-03-29, 22:20, # CodeGen: 0
**     Abstract    :
**         Main module.
**         This module contains user's application code.
**     Settings    :
**     Contents    :
**         No public methods
**
** ###################################################################*/
/* MODULE ProcessorExpert */


/* Including needed modules to compile this module/procedure */
#include "Cpu.h"
#include "Events.h"
#include "PWM1.h"
#include "AS1.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"


/* User includes (#include below this line is not maintained by Processor Expert) */
#include <hidef.h>       /* for EnableInterrupts macro */
#include "hcs08.h"       // This is our header file!
//Encoder definitions
typedef long int INT32;
typedef short int INT16;
#define Data        PTFD
#define XY          PTDD_PTDD0
#define OE          PTDD_PTDD1
#define EN1         PTDD_PTDD2
#define EN2         PTDD_PTDD3
#define SEL1        PTDD_PTDD4
#define SEL2        PTDD_PTDD5
#define RSTx        PTDD_PTDD6
#define RSTy        PTDD_PTDD7
#define BtReset     PTAD_PTAD2
#define BtSP        PTAD_PTAD3
//Definições Ponte H
#define GND                     PTGD_PTGD0
#define FB                      PTGD_PTGD1
#define IN1                     PTGD_PTGD2
#define EN          PTHD_PTHD0
#define IN2         PTHD_PTHD1
#define D1          PTHD_PTHD0
#define PWM1        PTCD_PTCD1
//End Encoder definitions

//PWM definition
```

```
//#define CICLO_ATIVO TPM1C1V
//End PWM definition

//Encoder variables
INT32 valor = 1234567890;
INT32 counter = 0;
INT32 rxint=0;
char var;
int var2=2500;
byte controle = 1;

void SendChar(char s_char) {

  //SCI1C2 = 0x08;    // enable Tx
  while(!SCI1S1_TDRE){ }
  SCI1D = (byte) s_char;   // 2nd half of TDRE clear procedure
} //end SendChar

void inttostr(INT32 n, char s[])
 {
     INT32 i, sign;

     i = 0;
     do {       /* generate digits in reverse order */
         s[i++] = n % 10 + '0';   /* get next digit */
     } while ((n /= 10) > 0);     /* delete it */
     s[i] = '\0';
     //reverse(s);
 }


void SCI_send_int(INT32 var)
{
  int p;
  inttostr(var, cBuffer);

  for(p=5;p>-1;p--)         // while the current char of the string is not null
  {
    while (!SCI1S1_TDRE);  // wait for the transmit buffer to be empty
    SCI1D = cBuffer[p];       // write the current char into the transmit buffer
                // increment the current char position within the string
  }
}

INT32 readEncoderX() {
           PTDD = 0xE6; // 0b11100110
               PTDD = 0xE4; // RSTy = 1, RSTx = 1, SEL2 = 1, SEL1 = 0, EN2 = 0, EN1 = 1, OE = 0, XY
= 0
               counter = 0;

               PTDD = 0xD4;
               counter = counter | Data;

               while(!SCI1S1_TDRE){ }
               SCI1D = Data;

               PTDD = 0xC4;
               counter = counter | ((ulong)Data << 8);

               while(!SCI1S1_TDRE){ }
```

```
            SCI1D = Data;


            PTDD = 0xF4;
            counter = counter | ((ulong)Data << 16);

            while(!SCI1S1_TDRE){ }
            SCI1D = Data;

            PTDD = 0xE4;
            counter = ((ulong)Data << 24);

            while(!SCI1S1_TDRE){ }
            SCI1D = Data;

            PTDD = 0xD6;

            return counter;
}



void main(void)
{
  /* Write your local variable definition here */
          INT32 compare = 2;
          int pwmtst = 0, i;
          INT16 simulink = 0;
          INT32 buffer[4];
          char *buff;
          word tempo=0;

  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
  /* For example: for(;;) { } */
          SOPT1 = bBKGDPE;        // enable the debug pin
          PTBDD_PTBDD7 = 1;       // PTB7 as an output

    //IO configs
    PTFDD = 0x00;    // Configura todo o PORTF como entrada        *
    PTDDD = 0xFF;    //configura todo o PORTD como saída          *



    PTEDD = 0xFF;
    PTFPE = 0xFF;
    PTCD  = 0x00;
    PTED  = 0x00;
    //End IO configs



    SCI_send_string("Starting libraries!\r\n");  // write into the serial port ...
    SCI_send_string("Starting encoder..\r\n");  // write into the serial port ...

    //Aguarda comando Simulink, que gera interrupção
```

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

/* END ProcessorExpert */
/*
** ###################################################################
**
**     This file was created by Processor Expert 5.3 [05.01]
**     for the Freescale HCS08 series of microcontrollers.
**
** ###################################################################
*/
```

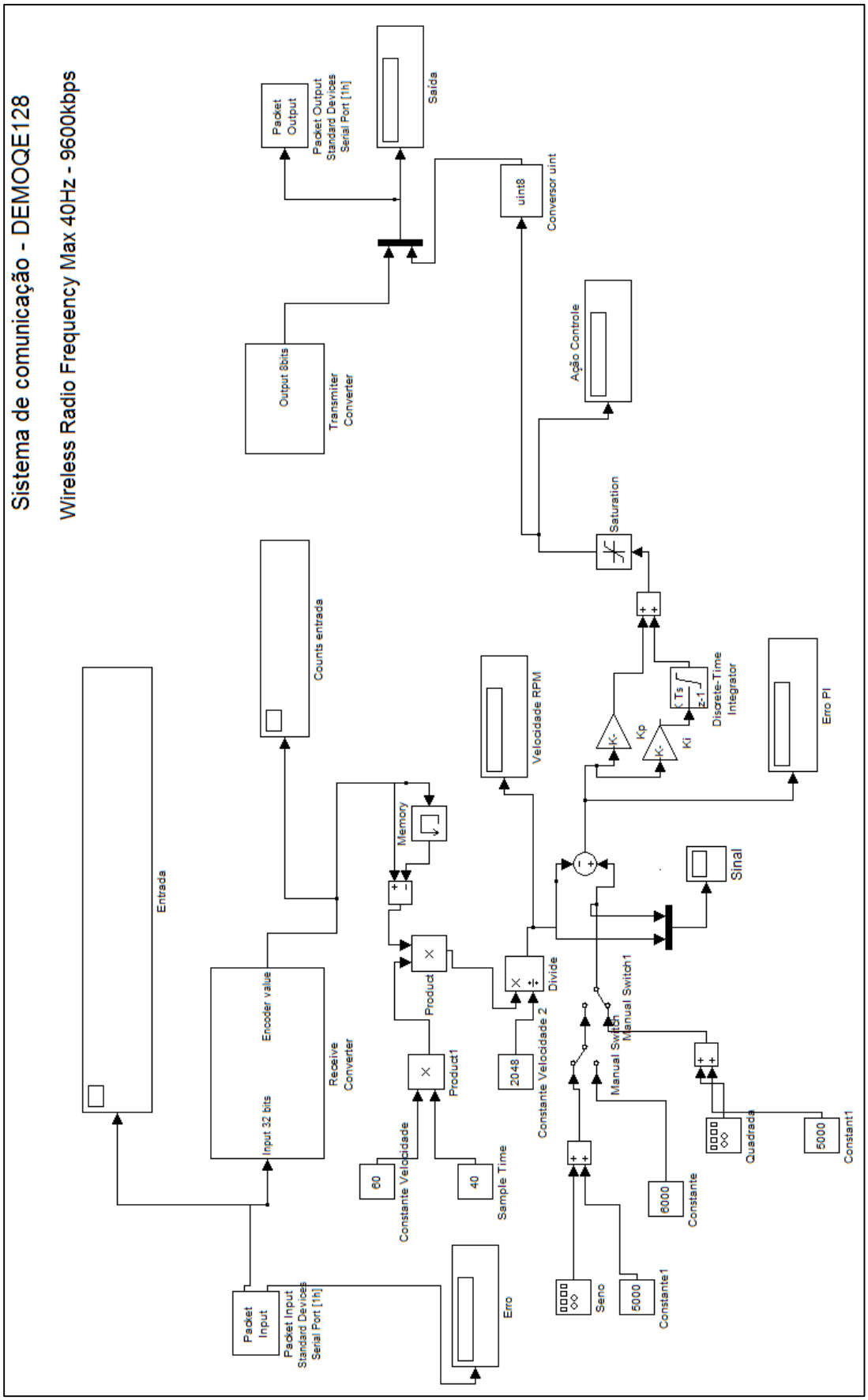
Events.c

```
/** ###################################################################
**     Filename    : Events.c
**     Project     : ProcessorExpert
**     Processor   : MC9S08QE128CLH
**     Component   : Events
**     Version     : Driver 01.02
**     Compiler    : CodeWarrior HCS08 C Compiler
**     Date/Time   : 2016-04-12, 22:50, # CodeGen: 0
**     Abstract    :
**         This is user's event module.
**         Put your event handler code here.
**     Settings    :
**     Contents    :
**         No public methods
**
** ###################################################################*/
/* MODULE Events */


#include "Cpu.h"
#include "Events.h"

/* User includes (#include below this line is not maintained by Processor Expert) */

/*
** ===================================================================
**     Event       :  AS1_OnError (module Events)
**
**     Component   :  AS1 [AsynchroSerial]
**     Description :
**         This event is called when a channel error (not the error
**         returned by a given method) occurs. The errors can be read
**         using <GetError> method.
**         The event is available only when the <Interrupt
**         service/event> property is enabled.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void  AS1_OnError(void)
{
  /* Write your code here ... */
}

/*
** ===================================================================
**     Event       :  AS1_OnRxChar (module Events)
**
**     Component   :  AS1 [AsynchroSerial]
**     Description :
**         This event is called after a correct character is received.
**         The event is available only when the <Interrupt
**         service/event> property is enabled and either the <Receiver>
**         property is enabled or the <SCI output mode> property (if
**         supported) is set to Single-wire mode.
**     Parameters  : None
```

```
**     Returns     : Nothing
** ===================================================================
*/
void  AS1_OnRxChar(void)
{
  /* Write your code here ... */
          AS1_TComData ch;                    // TComData type is defined in the AS1.h header file
          char input;
          char text[] = "Enviado";
          int controle;
          int transf;
          //if(AS1_RecvChar(&ch) == ERR_OK)
             //AS1_SendChar(ch);
                    //Read received character and send it if no error is detected
                    //if(controle != 1){
                    if((AS1_RecvChar(&input) == ERR_OK)){
                              if(input == '{'){
                                        readEncoderX();
                                        PTED = PTCD & 0b11000000;
                                        controle = 1;
                              }
                    //}
                   // if(controle == 1){
                             //  if((AS1_RecvChar(&input) == ERR_OK))
                               else if(input > 0x0 && input < 0xFF && input != '{'){
                                        PWM1_SetDutyUS((238/100)*10*(240-(int)input));

                              }
                    //}
                    }
}

/*
** ===================================================================
**     Event       :  AS1_OnTxChar (module Events)
**
**     Component   :  AS1 [AsynchroSerial]
**     Description :
**         This event is called after a character is transmitted.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void  AS1_OnTxChar(void)
{
  /* Write your code here ... */
}

/*
** ===================================================================
**     Event       :  Cpu_OnSwINT (module Events)
**
**     Component   :  Cpu [MC9S08QE128_64]
**     Description :
**         This event is called when the SWI interrupt had occurred.
**     Parameters  : None
**     Returns     : Nothing
** ===================================================================
*/
void Cpu_OnSwINT(void)
```

```
{
  /* Write your code here ... */
}

/* END Events */

/*
** ###################################################################
**
**     This file was created by Processor Expert 5.3 [05.01]
**     for the Freescale HCS08 series of microcontrollers.
**
** ###################################################################
*/
```

APÊNDICE B – Controle Simulink


Sistema de comunicação - DEMOQE128
Wireless Radio Frequency Max 40Hz - 9600kbps
Packet Output
Packet Output Standard Devices Serial Port [1h]
Saída
uint8
Conversor uint
Output 8bits
Transmiter Converter
Ação Controle
Saturation
Counts entrada
Velocidade RPM
Erro PI
Kp
Ki
Ts
z-1
Discrete-Time Integrator
Memory
Entrada
Sinal
Encoder value
Input 32 bits
Receive Converter
Product
Product1
Divide
2048
Constante Velocidade 2
Manual Switch
Manual Switch1
60
Constante Velocidade
40
Sample Time
6000
Constante
Quadrada
5000
Constant1
Seno
5000
Constante1
Erro
Packet Input
Packet Input Standard Devices Serial Port [1h]