

BRUNO MARTIN DE ALCÂNTARA DIAS

**Unidade microcontroladora para gerenciamento eletrônico de um motor de
combustão interna ciclo Otto**

São Paulo

2015

BRUNO MARTIN DE ALCÂNTARA DIAS

**Unidade microcontroladora para gerenciamento eletrônico de um motor de
combustão interna ciclo Otto**

Dissertação apresentada à Escola
Politécnica da Universidade de São Paulo
para a obtenção do título de Mestre em
Ciências

Orientador:
Prof. Dr. João Francisco Justo Filho

São Paulo
2015

BRUNO MARTIN DE ALCÂNTARA DIAS

**Unidade microcontroladora para gerenciamento eletrônico de um motor de
combustão interna ciclo Otto**

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção do título de Mestre em Ciências

Área de Concentração:
Engenharia Elétrica
Microeletrônica

Orientador:
Prof. Dr. João Francisco Justo Filho

São Paulo
2015

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

ALCÂNTARA DIAS, BRUNO MARTIN DE

Unidade microcontroladora para gerenciamento eletrônico de um motor de combustão interna ciclo Otto / B. M. D. ALCÂNTARA DIAS -- versão corr. -- São Paulo, 2015.

269 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Motores de combustão interna 2.Gerenciamento eletrônico
I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

ALCÂNTARA DIAS, BRUNO MARTIN DE. **Unidade microcontroladora para gerenciamento eletrônico de um motor de combustão interna ciclo Otto**. São Paulo. 2015. 269p. (Mestrado) Escola Politécnica, Universidade de São Paulo, São Paulo, 2015.

Dedico este trabalho a Lorenza Arce, mulher
honrada e trabalhadora que foi meu
alicerce durante esta jornada.

AGRADECIMENTOS

Agradeço primeiramente a Lorenza Arce que dedicou todos os dias de sua vida, desde que nasci, para que eu aqui estivesse com saúde. Muitas dificuldades foram superadas por nós com força, honra e humildade e hoje dedico este trabalho como forma de agradecimento a tudo que ela já superou por nós.

Ao Prof. Dr. João Francisco Justo Filho, orientador deste trabalho, que orientou e incentivou a execução deste trabalho, sendo uma pessoa sensata com boas sugestões, críticas, elogios e apoio durante todos os momentos de elaboração dessa dissertação.

Ao meu amigo e Prof. Dr. Armando Antônio Maria Laganá que iniciou a ideia principal deste projeto e me incentivou dia a dia para a execução do mesmo. Sem a ajuda do Professor Laganá, incluindo muitos finais de semana e feriados, não seria possível concluir as atividades propostas para este trabalho.

Ao Laboratório de Sistemas Integráveis da Escola Politécnica da USP pela estrutura cedida para a implementação do projeto.

A FATEC Santo André, pelo imenso apoio dado, cedendo o veículo e os equipamentos essenciais para os testes do projeto como, por exemplo, o veículo, o dinamômetro, a placa de aquisição de dados, osciloscópios etc.

Aos professores da Fatec Santo André Edson Caoru Kitani e Marco Aurélio Fróes pelos ensinamentos, discussões e sugestões que fizeram a elaboração final do projeto possível, proporcionando além da ideia inicial do projeto junto ao professor Laganá, melhores resultados finais ao projeto.

A Cynthia Thamires da Silva, por estar sempre ao meu lado, durante toda a jornada, sendo sempre prestativa, amorosa e atuante. O desenvolvimento do sistema de admissão e o levantamento de todo o modelo teórico da válvula borboleta não seriam possível sem sua ajuda.

Aos colegas de laboratório Marcos Antonio Carvalho Guedes, Bruno Silva Pereira, Felipe Serafim Albaladejo e Demerson Moscardini pelos auxílios prestados no desenvolvimento deste projeto.

Gostaria de agradecer a Juliana Nunes Martin Ribeiro, Daniela Giudilli, Muryel Tarifa, Davi e Rafaela Virgilio e a todos aqueles que direta e indiretamente contribuíram para a realização deste trabalho.

"A lei da mente é implacável.
O que você pensa, você cria.
O que você sente, você atrai.

O que você acredita,
torna-se realidade."

Siddhārtha Gautama

RESUMO

Nas últimas décadas, a indústria automobilística mundial vem investindo no desenvolvimento tecnológico dos motores, com o objetivo de alcançar melhor eficiência energética e atender às legislações que limitam a quantidade de resíduos tóxicos nos gases de exaustão e menor consumo de combustível. Isso resultou na implantação dos sistemas de gerenciamento eletrônico do motor, que possibilitam funcionalidades para se controlar diversas variáveis do motor, aumentando consideravelmente o rendimento do motor. Este trabalho tem como objetivos explorar a dinâmica de um motor de combustão interna ciclo Otto, os sinais elétricos associados, e os componentes de seu gerenciamento eletrônico. A partir dessas informações, o trabalho apresenta o processo de análise dos sinais elétricos e as estratégias de controle utilizadas em um sistema de gerenciamento real. Assim, são desenvolvidos o *hardware* e o *firmware* de uma unidade microcontroladora para gerenciamento eletrônico do motor. O *hardware* foi elaborado com uma concepção centralizada, ou seja, foi usado apenas um microcontrolador de 32-bit para gerenciar todas as funções. O *firmware* de controle foi desenvolvido de forma modular baseado em modelos de malha fechada. O modelo matemático do motor foi identificado utilizando técnicas de controle em um veículo real, e a validação do modelo foi obtida através de testes em um dinamômetro inercial.

Palavras-Chave: Unidade de Controle do Motor (UCM). Gerenciamento Eletrônico de Motores de Combustão Interna. Controle em Malha Fechada.

ABSTRACT

In the last few decades, the world automotive industry has invested in the technological development of the engines, aiming to get better energetic efficiency and comply with legislations that limit the amount of toxic exhaustion gases. This resulted in electronic management systems for engines, which allowed flexibility to control several engine-related variables, considerably increasing the engine efficiency. This work aims to explore the Otto cycle combustion engine dynamics, its electronic signals, and the elements of its electronic management. Based on that information, this work presents the analysis process of the electronic signals, and the control strategies used in a real management system. Then, the hardware and firmware of a microcontroller unity are developed for the electronic management of the engine. The hardware was elaborated with a centralized concept, i.e., only one 32-bit microcontroller was used to control all functions. The control firmware was developed in a modular scheme, based on the closed loop models. The engine mathematical model was identified using control techniques in a real engine. The results were obtained by tests in an inertial dynamometer.

Keywords: Electronic Control Unit (ECU). Electronic Management of Internal Combustion Engines. Closed-loop control system.

LISTA DE ILUSTRAÇÕES

Figura 1	– Principais componentes de um motor de combustão interna ciclo Otto	14
Figura 2	– Ciclo de trabalho de um motor de combustão interna ciclo Otto	14
Figura 3	– Potência, consumo específico em função de λ	16
Figura 4	– Emissões e relação com fator λ	18
Figura 5	– Visão geral de um sistema de controle de um motor ciclo Otto	19
Figura 6	– Componentes do sistema de admissão de um motor ciclo Otto	19
Figura 7	– Sistema de injeção de combustível comum	21
Figura 8	– Componentes do sistema de ignição	22
Figura 9	– Influência do avanço de ignição na pressão do cilindro	23
Figura 10	– Unidade de controle do motor e circuito eletrônico microcontrolado	26
Figura 11	– Principais sensores e atuadores de um sistema de gerenciamento	27
Figura 12	– Sensores de rotação do motor	29
Figura 13	– Sensor de fase	30
Figura 14	– Sensor de pressão absoluta	31
Figura 15	– Sensor de temperatura NTC	32
Figura 16	– Sensor posição da válvula borboleta	33
Figura 17	– Sensor posição do pedal do acelerador	34
Figura 18	– Curva característica dos sensores de oxigênio EGO e UEGO	35
Figura 19	– Sensor de oxigênio dos gases de exaustão	36
Figura 20	– Válvulas injetoras e tubo distribuidor de combustível	38
Figura 21	– Transformador de ignição	39
Figura 22	– Atuador da válvula borboleta eletrônica	40
Figura 23	– Modelo de controle de lambda proposto por Braga	48
Figura 24	– Modelo conceitual de um motor de combustão interna	51
Figura 25	– Ficha técnica dos motores da família EA 113 2.0L	59
Figura 26	– Sincronismo de comando eletrônico dos motores EA 113 2.0L	60
Figura 27	– Diagrama de blocos da UCM v3.0	62
Figura 28	– Diagrama de operação do bloco de gestão de carga	63

Figura 29	– Condicionamento do sinal de ignição	67
Figura 30	– Condicionamento do sinal do sensor de rotação e PMS	68
Figura 31	– Condicionamento do sinal do sensor de fase	69
Figura 32	– Condicionamento do sinal do sensor de pressão absoluta no coletor de admissão	70
Figura 33	– Condicionamento do sinal dos sensores de temperatura NTC	71
Figura 34	– Condicionamento do sinal dos sensores de posição da válvula borboleta e do pedal do acelerador	72
Figura 35	– Condicionamento do sinal dos sensores de oxigênio e detonação	72
Figura 36	– Condicionamento do sinal dos interruptores de embreagem e pedal de freio	73
Figura 37	– Condicionamento do sinal de carga do alternador	74
Figura 38	– Condicionamento do sinal de acionamento dos reles automotivos	75
Figura 39	– Condicionamento do sinal de acionamento do motor da válvula borboleta	76
Figura 40	– Principais Condicionamento do sinal de acionamento das válvulas injetoras e das bobinas de ignição	77
Figura 41	– Condicionamento do sinal de acionamento da bobina de ignição com <i>driver</i> externo	78
Figura 42	– Condicionamento do sinal de acionamento dos atuadores de uso geral	79
Figura 43	– Diagrama de comunicação da UCM	80
Figura 44	– Circuito de gravação do microcontrolador padrão JTAG	81
Figura 45	– Descrição geral do programa principal (main.c)	83
Figura 46	– Fluxograma do programa principal (main.c)	85
Figura 47	– Diagrama de tempos das funções do programa principal	87
Figura 48	– Fluxograma da rotina de tratamento da interrupção do sinal de ignição (Linha 15)	90
Figura 49	– Fluxograma da rotina de tratamento da interrupção do sensor de rotação do motor	92
Figura 50	– Sistema de intersecção da UCM	97

Figura 51	– <i>Layout</i> da placa de circuito impresso finalizado	100
Figura 52	– Circuito impresso finalizado	101
Figura 53	– <i>Hardware</i> finalizado	102
Figura 54	– Condicionamento do sinal de rotação	103
Figura 55	– Condicionamento do sinal do sensor de pressão no coletor de admissão	104
Figura 56	– Mapa de Avanço de Ignição Interpolado e finalizado	108
Figura 57	– Curva característica do sensor MAP utilizado neste projeto	109
Figura 58	– Função de conversão do sensor MAP	109
Figura 59	– Função de Cálculo do Avanço de Ignição	110
Figura 60	– Função de Cálculo do Tempo de Injeção	114
Figura 61	– Ilustração de Chen para o modelo da válvula borboleta	115
Figura 62	– Modelo da válvula borboleta em <i>Simulink</i>	116
Figura 63	– Sinal de entrada com degrau de 3V, posição real e simulada do modelo	117
Figura 64	– Controlador PI de posição da válvula borboleta	118
Figura 65	– Função de sincronismo do sistema de admissão	119
Figura 66	– Resultados obtidos com o controlador PI implementado	120
Figura 67	– Sistema de diagnose desenvolvido (HMI)	122
Figura 68	– Tarefa desenvolvida para enviar os dados ao sistema de diagnose	123
Figura 69	– Lógica de programação do diagrama de blocos do HMI	123
Figura 70	– Resposta da rotação ao degrau da válvula borboleta	125
Figura 71	– Simulação do modelo obtido em <i>Matlab®</i> ,	126
Figura 72	– Implementação do controlador de rotação	128
Figura 73	– Modelo de controle completo do motor ciclo Otto	129
Figura 74	– Primeiro teste de potência para validação da eficiência volumétrica	132
Figura 75	– Medição final da potência do motor	133
Figura 76	– Medição final da sincronismo de controle do motor	135

LISTA DE ABREVIATURA E SIGLAS

APMS	Antes do Ponto Morto Superior
CAD	<i>Computer Aided Design</i>
CAN	<i>Controller Area Network</i>
DAC	<i>Digital Analog Converter</i>
DPMS	Depois do Ponto Morto Superior
DSP	<i>Digital Signal Processing</i>
ECU	<i>Electronic Control Unit</i>
EGR	<i>Exhaust Gas Recirculation</i>
EPROM	<i>Erasable Programmable Read Only Memory</i>
GND	<i>Ground</i>
JTAG	<i>Joint Test Access Group</i>
L15	Linha 15 - Positivo pós-chave.
L30	Linha 30 - Fonte primária de energia elétrica do veículo.
LQG	<i>Linear Quadratic Gaussian</i>
MAF	<i>Mass Air Flow Sensor</i>
MAP	<i>Manifold Absolute Pressure</i>
MIMO	<i>Multiple Inputs Multiple Outputs</i>
MVEM	<i>Mean Value Engine Model</i>
NARX	<i>Nonlinear autoregressive exogenous</i>
PCI	<i>Peripheral Component Interconnect</i>
PI	<i>Proportional Integrative</i>
PID	<i>Proportional Integrative Derivative</i>
PMI	Ponto Morto Inferior
PMS	Ponto Morto Superior
PWM	<i>Pulse Width Modulation</i>
RTOS	<i>Real Time Operating System</i>
SMD	<i>Surface Mount Device</i>
SPI	<i>Serial Peripheral Interface</i>
UART	<i>Universal Asynchronous Receiver Transmitter</i>
UCM	Unidade de Controle do Motor
USB	<i>Universal Serial Bus</i>

SUMÁRIO

1	INTRODUÇÃO	7
1.1	OBJETIVO E MOTIVAÇÃO	9
1.2	CONTEÚDO	11
2	GERENCIAMENTO ELETRÔNICO DE MOTORES DE COMBUSTÃO INTERNA CICLO OTTO	13
2.1	MOTORES DE COMBUSTÃO INTERNA CICLO OTTO	13
2.1.1	Formação da mistura carburante nos motores ciclo Otto	15
2.2	TEORIA DE CONTROLE DOS MOTORES DE COMBUSTÃO INTERNA CICLO OTTO	18
2.2.1	Subsistema de Admissão de Ar	19
2.2.2	Subsistema dosador de combustível	20
2.2.3	Subsistema de ignição	21
2.3	GERENCIAMENTO ELETRÔNICO DOS MOTORES DE COMBUSTÃO INTERNA CICLO OTTO	24
2.3.1	Histórico e evolução dos sistemas de controle nos motores Otto	24
2.3.2	Conceitos e funções de uma unidade de controle do motor	25
2.3.2.1	Sensores	28
2.3.2.1.1	<i>Sensor de rotação do motor</i>	28
2.3.2.1.2	<i>Sensor de fase</i>	30
2.3.2.1.3	<i>Sensor de pressão do coletor de admissão</i>	30
2.3.2.1.4	<i>Sensor de temperatura</i>	32
2.3.2.1.5	<i>Sensor de posição angular da válvula borboleta</i>	33
2.3.2.1.6	<i>Sensor de posição do pedal do acelerador</i>	34
2.3.2.1.7	<i>Sensor de oxigênio dos gases de exaustão</i>	35
2.3.2.1.8	<i>Sensor de detonação</i>	37
2.3.2.2	Atuadores	37
2.3.2.2.1	<i>Válvulas injetoras de combustível</i>	38
2.3.2.2.2	<i>Bobinas de ignição</i>	38
2.3.2.2.3	<i>Atuador da válvula borboleta</i>	40
2.4	ESTRATÉGIA DE CONTROLE	41
2.5	MÉTODO DE CONTROLE E MODELOS MATEMÁTICOS DO	

	MOTOR OTTO	43
2.5.1	Modelo do motor para aplicações de controle proposto por Hendricks.....	43
2.5.2	Modelo linear de controle proporcional integral proposto por Braga	47
2.5.3	Identificação e controle da marcha lenta proposto por Nicolao, Rossi, Scattolini e Suffritti	49
2.5.4	Modelo auto-ajustável de controle aplicado na calibração de motores de combustão interna ciclo Otto proposto por Dorey e Stuart.....	52
2.6	CONSIDERAÇÕES FINAIS	55
3	METODOLOGIA	57
3.1	DESCRIÇÃO GERAL	57
3.1.1	Motores da família EA113	58
3.2	DESCRIÇÃO DO HARDWARE DA UCM	61
3.2.1	Gestão de carga.....	62
3.2.2	Microcontrolador.....	64
3.2.3	Condicionamento eletrônico dos sensores	66
3.2.3.1	Sinal de ignição (linha 15)	66
3.2.3.2	Sensor de rotação e PMS	67
3.2.3.3	Sensor de fase	69
3.2.3.4	Sensor de pressão absoluta no coletor de admissão (MAP)	69
3.2.3.5	Sensores de temperatura	70
3.2.3.6	Sensores de posição da válvula borboleta e do pedal do acelerador	71
3.2.3.7	Sensor de oxigênio e sensor de detonação	72
3.2.3.8	Interruptores de embreagem e do pedal de freio	73
3.2.3.9	Sinal de carga do alternador	73
3.2.4	Condicionamento eletrônico dos atuadores.....	74
3.2.4.1	Acionamento do relés	75
3.2.4.2	Acionamento do motor da válvula borboleta	76
3.2.4.3	Acionamento das válvulas injetoras de combustível e das bobinas de ignição com <i>driver</i> interno	77
3.2.4.4	Acionamento das bobinas de ignição com <i>driver</i> externo	78
3.2.4.5	Acionamento de atuadores de uso geral	79

3.2.4.6	Transmissores de comunicação externa	79
3.2.5	Círcuito de gravação JTAG	80
3.3	DESCRÍÇÃO DO <i>FIRMWARE</i> DA UCM	81
3.3.1	Descrição geral do <i>firmware</i> de controle do motor	82
3.3.2	Estratégia geral do <i>firmware</i> de controle do motor	84
3.3.3	Conceito da camada RTOS	86
3.3.4	Conceito da camada sincronismo	89
3.4	CONSIDERAÇÕES FINAIS	94
4	RESULTADOS	97
4.1	SISTEMA DE INTERSECÇÃO DOS SINAIS ELÉTRICOS DA UCM	97
4.2	SISTEMA DE AQUISIÇÃO DE DADOS PARA ANÁLISE DOS RESULTADOS	98
4.3	DESENVOLVIMENTO DO LAYOUT E DA PLACA DE CIRCUITO IMPRESSO	99
4.4	UNIDADE DE CONTROLE DO MOTOR FINALIZADA	101
4.5	VALIDAÇÃO DO HARDWARE	103
4.6	SINCRONISMO DOS SISTEMAS DE CONTROLE DO MOTOR	105
4.6.1	Controle do sistema de ignição	106
4.6.2	Controle do sistema de injeção	110
4.6.3	Controle do sistema de admissão	115
4.7	SISTEMA DE DIAGNOSE	121
4.8	OBTENÇÃO DO MODELO	124
4.9	DESENVOLVIMENTO DO CONTROLADOR DE ROTAÇÃO	126
4.10	TESTES FINAIS E VALIDAÇÃO DO TRABALHO	130
4.10.1	Teste de potência e validação da teoria de controle aplicada	131
4.10.2	Validação das ações de sincronismo do motor	134
4.11	CONSIDERAÇÕES FINAIS	136
5	CONCLUSÕES	139
5.1	PROPOSTAS FUTURAS	142
	REFERÊNCIAS	145
	APÊNDICE A - <i>Firmware</i> Completo Utilizado no Teste Final	149
	APÊNDICE B - Esquema Elétrico Completo da Unidade de Controle so Motor	247

1 INTRODUÇÃO

No final do século XIX, um engenheiro francês chamado Alphonse Beau de Rochas formulou a teoria termodinâmica de operação do motor de combustão interna de quatro tempos. A abordagem de Rochas enfatizou a importância de comprimir a mistura de ar e combustível admitido para dentro dos cilindros antes de iniciar a combustão. Entretanto, Rochas nunca construiu um protótipo desse modelo teórico. Somente em 1896, a teoria formulada por Rochas foi implementada com sucesso pelo engenheiro alemão Nikolaus Otto e motores de combustão interna de quatro tempos passaram a ser conhecidos como motores Otto. Atualmente, os motores de combustão interna ciclo Otto estão presentes na grande maioria dos veículos automotores de pequeno e médio porte fabricados no mundo. Estes motores utilizam combustíveis fósseis como fonte primária de energia, gerando resíduos químicos em grande quantidade, que são tóxicos e prejudicam o meio ambiente. Nas grandes cidades, as emissões de resíduos tóxicos proveniente da queima de combustíveis fósseis é alarmante e no futuro é esperado um aumento significativo na quantidade de veículos produzidos em todo o mundo (PASSARINI, 1993).

Focando unicamente em fatores técnicos, o motor ciclo Otto é um dispositivo mecânico que apresenta uma baixa eficiência energética em comparação com outros motores como, por exemplo, o motor elétrico. Entretanto, pelo menos nas próximas décadas, devido a diversos fatores, como o volume reduzido dos motores Otto, a facilidade de sua produção e a conveniência em se utilizar combustíveis fósseis, a indústria automobilística continuará utilizando esses motores como padrão para aplicações automotivas (MILHOR, 2002).

Segundo Hrovat (HROVAT e colaboradores, 1997), é esperado um aumento anual significativo na produção de veículos automotores equipados com motores Otto no mundo. Os efeitos do aumento da frota mundial vem impulsionando a indústria automobilística em duas direções: a busca por combustíveis alternativos e melhorias no gerenciamento do motor. O conceito aplicado pela indústria automobilística tem como objetivo aumentar a eficiência energética dos motores. Historicamente, este conceito aplicado fez com que a tecnologia presente no controle do motor se aprimorasse com o passar dos anos, podendo-se citar como exemplo desta evolução, o desenvolvimento de combustíveis alternativos como o

etanol e a migração dos veículos carburados para os veículos com gerenciamento eletrônico no motor (GUIMARÃES, 2007).

Além do conceito de eficiência energética aplicado pela indústria automobilística, existem legislações por todo o mundo que limitam a quantidade de resíduos considerados tóxicos e prejudiciais ao meio ambiente que um veículo pode emitir nos gases de exaustão. Essas legislações obrigam os fabricantes a investirem em tecnologias de controle de emissões nos motores de combustão interna (HROVAT e colaboradores, 1997). Os motores Otto, desde sua primeira implementação por Nikolaus Otto até meados dos anos 60, utilizaram somente soluções mecânicas para o controle da mistura carburante e do sistema de ignição. Os carburadores e distribuidores controlavam o motor de maneira satisfatória, porém devido à impossibilidade de ajustes em tempo real, essas soluções se tornaram ineficientes para atender as legislações cada vez mais rígidas. Isso levou a indústria automobilística a desenvolver os sistemas eletrônicos de gerenciamento do motor. Eles possibilitam maior flexibilidade para se controlar os sistemas de injeção de combustível, de ignição e de admissão, quando comparados aos sistemas mecânicos. Segundo Hrovat (HROVAT e colaboradores, 1997), a queima incompleta de combustível efetuada nos motores é responsável por boa parte dos problemas ambientais e de saúde da população global. Em síntese, este foi o fator principal para que a indústria automobilista migrasse para os sistemas eletrônicos de gerenciamento do motor (PUJATTI, 2007).

Segundo Kjellqvist (KJELLQVIST, 2005), o conceito e as estratégias de controle do motor podem seguir diversas teorias de controle e métodos de aplicação, e diversos modelos matemáticos podem transcrever de forma eficaz o mesmo sistema de controle. Kjellqvist ainda afirma que atualmente a maneira mais eficiente de aplicar estas teorias de controle é em sistemas eletrônicos, uma vez que é muito difícil de ajustar um sistema mecânico à atual condição de operação do motor. Sistemas de gerenciamento eletrônico de motores Otto permitem a atuação em malha fechada, fazendo com que o sistema se corrija em função do erro de estado estacionário, o que já amplia significativamente o rendimento dinâmico do motor. Motores de combustão interna ciclo Otto são sistemas cujo modelo de controle completo prevê múltiplas entradas e múltiplas saídas (MIMO) e a dinâmica de operação desses motores variam, muitas vezes, de forma imprevisível, conforme fatores atmosféricos, ambientais etc. (KJELLQVIST, 2005).

Atualmente, os sistemas eletrônicos de gerenciamento do motor são unidades microcontroladoras com memória para armazenamento de mapas de calibração, circuitos eletrônicos para condicionamento dos sensores automotivos e para adequação da potência dos atuadores. O desenvolvimento de unidades microcontroladoras para gerenciamento eletrônico de motores Otto exige a elaboração de refinados *firmware* embarcados com técnicas de controle embutidas. As unidades microcontroladoras de gerenciamento eletrônico do motor evoluem de acordo com o avanço dos microcontroladores, sensores e atuadores (CÂMARA, 2006).

Com o objetivo de compreender toda a dinâmica de um motor ciclo Otto, este trabalho apresenta o processo de análise de todos os sinais desse motor, uma discussão sobre as estratégias de controle utilizadas em um sistema de gerenciamento real e o desenvolvimento de uma unidade microcontroladora para gerenciamento eletrônico.

Ao se concluir a elaboração da unidade microcontroladora proposta, espera-se obter uma ferramenta para auxiliar outras pesquisas relacionadas ao gerenciamento eletrônico de motores Otto. A partir dessa unidade, será possível testar novas tecnologias ou controles desenvolvidos com o intuito de aprimorar todo processo de combustão e emissão de poluentes.

1.1 OBJETIVO E MOTIVAÇÃO

O crescimento contínuo da quantidade de veículos automotores circulando por todo o mundo levou a um consumo de aproximadamente um bilhão de litros de combustíveis fósseis diariamente (RIBBENS, 2003). É responsabilidade de todos, inclusive do setor acadêmico, desenvolver e propor soluções sustentáveis para diminuir consideravelmente os danos ao meio ambiente. Entretanto, a análise da dinâmica de um motor de combustão interna depende de um grande número de parâmetros, que variam desde perdas mecânicas existentes no sistema até o rendimento energético e químico do combustível presente no processo de combustão. Aprimorar o controle desse dispositivo contribui com o aumento da eficiência energética e está diretamente ligado à redução na emissão de resíduos tóxicos na atmosfera. Naturalmente, é possível afirmar que o aprimoramento do

controle de motores Otto está diretamente ligado ao estudo e compreensão de todos os sinais elétricos emitidos pelos sensores presentes no motor.

Diversas referências afirmaram que a aplicação de técnicas de controle, com o objetivo de minimizar a emissão de poluentes e maximizar o rendimento do motor, é o melhor caminho na atualidade. Entretanto, a indústria automobilística geralmente mantém sigilo sobre as unidades de gerenciamento eletrônico, incluindo as estratégias de controle e modelos utilizados em suas aplicações, dificultando que novos modelos e estratégias sejam aplicadas pela comunidade acadêmica.

O objetivo deste trabalho é desenvolver um sistema flexível de gerenciamento de motores Otto, possibilitando a compreensão de toda a sua dinâmica de operação. Nesse sentido, o trabalho apresenta todo o processo para se desenvolver uma unidade microcontroladora de gerenciamento eletrônico, incluindo uma análise completa de um sistema real, e ilustrando os sinais elétricos emitidos por todos os seus sensores e atuadores. Foi elaborado um controlador PI de rotação do motor e a função de transferência foi adquirida através de análises da resposta ao degrau da válvula borboleta. A flexibilidade, oferecida pela unidade desenvolvida neste trabalho com relação às estratégias de controle, permite que técnicas de controle moderno sejam aplicadas aos motores de combustão interna, que consiste em uma planta não linear com parâmetros variáveis no tempo, ideal para aplicações de ferramentas de identificação, estratégias de controle clássico, adaptativo, ótimo, robusto e técnicas de inteligência artificial.

A unidade microcontroladora de gerenciamento eletrônico pode condicionar e processar os sinais emitidos pelos principais sensores do motor, atuando no motor e mantendo-o em funcionamento, controlando o tempo de injeção, avanço de ignição e o acelerador eletrônico. O *hardware* elaborado possui uma concepção centralizada, ou seja, apenas um microcontrolador para gerenciar todas as funções. O projeto foi desenvolvido, testado e aplicado em um sistema veicular real.

Os modelos matemáticos do motor foram elaborados utilizando técnicas de identificação de sistemas. O modelo adquirido serviu como base para a aplicação de técnicas de controle clássico em malha fechada e a validação foi efetuada através de ensaios em um sistema real.

1.2 CONTEÚDO

Este trabalho está dividido da seguinte forma: O capítulo 2 consolida a teoria dos modelos presentes na literatura, que permitiram a elaboração de estratégias para a aplicação dos modelos de controle utilizados e permitiram elaborar toda estrutura do *hardware* e do *firmware* desenvolvidos para testar as hipóteses deste trabalho. O capítulo 3 detalha todo o processo de desenvolvimento do *hardware*, abordando os principais componentes utilizados e suas respectivas funções na execução do sistema, e do *firmware*, abordando as técnicas de controle aplicadas no gerenciamento do motor. O capítulo 4 detalha os ensaios e os resultados obtidos no projeto. Esse capítulo apresenta ainda a validação de todos os controladores aplicados com testes em dinamômetro. Finalmente, o capítulo 5 apresenta as conclusões obtidas dos resultados, construindo um referencial que fundamentou as discussões e a análise sobre os resultados finais, confirmando as hipóteses deste projeto e fundamentando propostas para trabalhos futuros.

2 GERENCIAMENTO ELETRÔNICO DE MOTORES DE COMBUSTÃO INTERNA CICLO OTTO

A fundamentação teórica necessária para se compreender o papel de um sistema de gerenciamento eletrônico de um motor de combustão interna ciclo Otto começa pela análise e compreensão dos elementos mecânicos do motor, pela teoria de operação termodinâmica do motor, os seus sensores e atuadores, as estratégias de controle usualmente utilizadas e pela dinâmica existente nesses motores, além de alguns aspectos da emissão de poluentes e como é possível minimizar este efeito. Para isto, este capítulo aborda inicialmente um breve histórico dos controles utilizados e os principais motivos que levaram com que o controle eletrônico fosse adotado. Em seguida, são discutidos os principais sensores e atuadores presentes no gerenciamento eletrônico, abordando suas vantagens e suas funções nas estratégias básicas de controle em malhas aberta e fechada dos sistemas de admissão, ignição e injeção.

2.1 MOTORES DE COMBUSTÃO INTERNA CICLO OTTO

O motor de combustão interna ciclo Otto é um conjunto de peças fixas e móveis que possui a função de disponibilizar energia mecânica em seu eixo principal. Ele é classificado como um motor térmico, pois transforma a energia calórica dos combustíveis em energia mecânica, que aparece como trabalho disponibilizado em sua árvore de manivelas.

O modelo desse motor, funcionando com máxima eficiência e economia, foi proposto pelo físico francês Alphonse Beau de Rochas em 1862. Rochas propôs uma sequência de eventos que promovia uma compressão prévia que aumentava significativamente a economia, eficiência e rendimento em relação aos motores existentes naquele período. Essa sequência, que totaliza quatro tempos, é em síntese o que ocorre nos motores de combustão interna atuais. Entretanto, Rochas apenas formulou as condições de funcionamento, não tendo construído nenhum protótipo. A construção experimental deste motor foi realizada somente em 1872 por Nikolaus Otto. Quando ele passou a fabricar estes motores de quatro tempos com êmbolos móveis ligados a uma árvore de manivelas, o mesmo passou a ser

chamado de motor ciclo Otto. A figura 1 ilustra os principais componentes de um motor de combustão interna ciclo Otto (BASSHUYSEN, 2004).

Figura 1 - Principais componentes de um motor de combustão interna ciclo Otto.

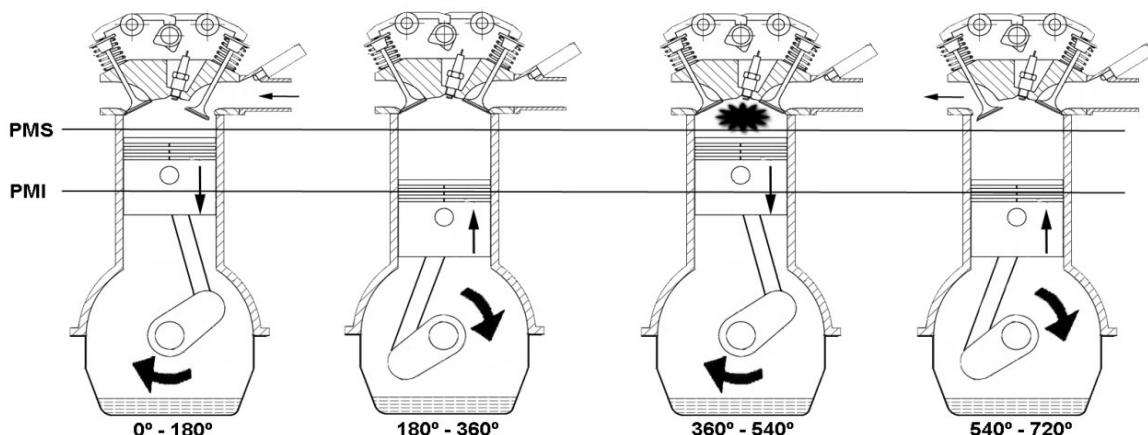


Fonte: o autor

A implementação de uma máquina térmica baseada no ciclo Otto ocorre em quatro ciclos termodinâmicos, ou seja, os motores operam em quatro ciclos em diferentes etapas que ocorrem pelo deslocamento repetitivo do êmbolo (pistão) ao longo de cilindros. Tais máquinas são classificadas como motores de quatro tempos, sendo que eles se referem aos quatro deslocamentos do êmbolo, que realizam as diferentes etapas do ciclo: admissão, compressão, combustão e exaustão, como ilustra a figura 2.

Figura 2 - Ciclo de trabalho de um motor de combustão interna ciclo Otto.

1 - ADMISSÃO 2 - COMPRESSÃO 3 - COMBUSTÃO 4 - EXAUSTÃO



Fonte: o autor

Conforme proposto por de Rochas, em 1862, um ciclo completo de um motor de combustão interna ciclo Otto corresponde a uma revolução de 720° da árvore de manivelas. Conforme a figura 2, para um único cilindro do sistema, o ciclo Otto começa pela etapa de admissão. Nessa etapa, a válvula de exaustão retorna a sua posição inicial em seu alojamento através da força de retorno da mola e a válvula de admissão é deslocada de seu alojamento inicial pelo comando de válvulas, permitindo que a mistura carburante seja aspirada para dentro do cilindro, através do deslocamento do êmbolo desde o Ponto Morto Superior (PMS) até o Ponto Morto Inferior (PMI) do cilindro (BASSHUYSEN, 2004).

Na etapa de compressão, a válvula de admissão retorna à sua posição inicial, através da força de retorno da mola da válvula, vedando totalmente a câmara de combustão. Com ambas as válvulas fechadas, o deslocamento do êmbolo desde o PMI até o PMS comprime a mistura carburante com objetivo de aumentar significativamente a pressão dentro do cilindro. Este ciclo aumenta a economia, a eficiência e o rendimento da próxima etapa.

Na combustão, a vela de ignição produz uma centelha elétrica que reage com a mistura carburante já previamente comprimida. Nesse instante, ocorre um aumento abrupto da pressão dentro da câmara de combustão, impulsionando o êmbolo do PMS para o PMI. Finalmente, a última etapa do ciclo é denominada exaustão. Nessa etapa, a válvula de exaustão é deslocada de seu alojamento inicial pelo comando de válvulas permitindo que os gases, que são o produto da combustão, sejam expelidos da câmara de combustão pelo êmbolo que se desloca do PMI para o PMS (BASSHUYSEN, 2004).

2.1.1 Formação da mistura carburante nos motores ciclo Otto

Na maioria dos motores ciclo Otto, a mistura carburante é formada no coletor de admissão entre as válvulas borboleta e de admissão. Já em motores mais modernos, esta mistura é formada dentro da própria câmara de combustão. Denomina-se mistura carburante a relação entre o ar atmosférico e o combustível admitido para dentro da câmara de combustão. Esta mistura deve ser a mais homogeneia possível. A presença de outros elementos no ar atmosférico além do oxigênio, principalmente nitrogênio, dificulta a reação química da gasolina na combustão. A relação ar/combustível admitida pelo motor não é constante e

depende de diversos fatores. Diversas situações podem fazer com que um motor, operando em uma mesma rotação e carga, apresente relação ar combustível diferente de outro, igual, nas mesmas condições (BASSHUYSEN, 2004).

Quando a mistura carburante admitida resulta em uma combustão perfeita, ou seja, todo o hidrogênio e carbono presente no combustível são convertidos em dióxido de carbono e água, está mistura passa a ser denominada uma mistura estequiométrica (MILHOR, 2002).

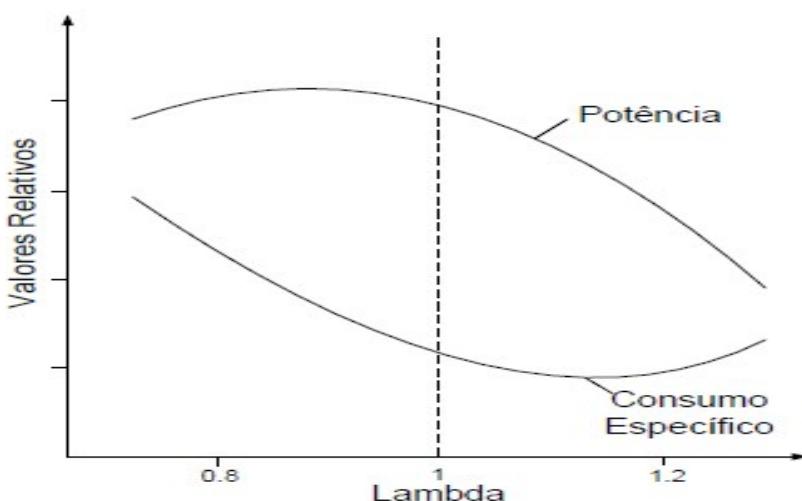
A relação entre a mistura carburante empregada e mistura carburante estequiométrica é definida pelo coeficiente de ar *lambda* (λ), conforme a equação 01.

$$\lambda = \frac{\text{MisturaCarburanteEmpregada}}{\text{MisturaCarburanteEstequiométrica}}$$

Eq.01

Desta forma, quando o coeficiente *lambda* (λ) for menor que 1.0 a mistura carburante está rica ou com deficiência de ar, quando λ for igual a 1.0 a mistura carburante está estequiométrica, quando λ for maior que 1.0 a mistura carburante está pobre ou com excesso de ar. O *lambda* é muito importante para o funcionamento do motor, pois afeta diretamente o consumo de combustível, a emissão de poluentes e o desempenho do motor, conforme ilustrado na figura 3, que considera o avanço de ignição, a rotação e a posição da válvula borboleta constantes (BRAGA, 2007).

Figura 3 - Potência, consumo específico em função de λ .



Fonte: (BRAGA, 2007).

A figura 3 mostra que a potência máxima do motor é alcançada com λ aproximadamente igual a 0.85 e que o consumo específico de combustível, assim como as emissões de poluentes, possui um melhor rendimento com λ aproximadamente igual a 1.1. A condição estequiométrica se situa na região intermediária, proporcionando um bom desempenho com baixo consumo (PUJATTI, 2007).

A relação estequiométrica entre o ar atmosférico e combustível admitido para dentro da câmara de combustão varia de acordo com o tipo de combustível utilizado, como ilustra a tabela 1, tendo sempre como referência a concentração de oxigênio presente no ar atmosférico.

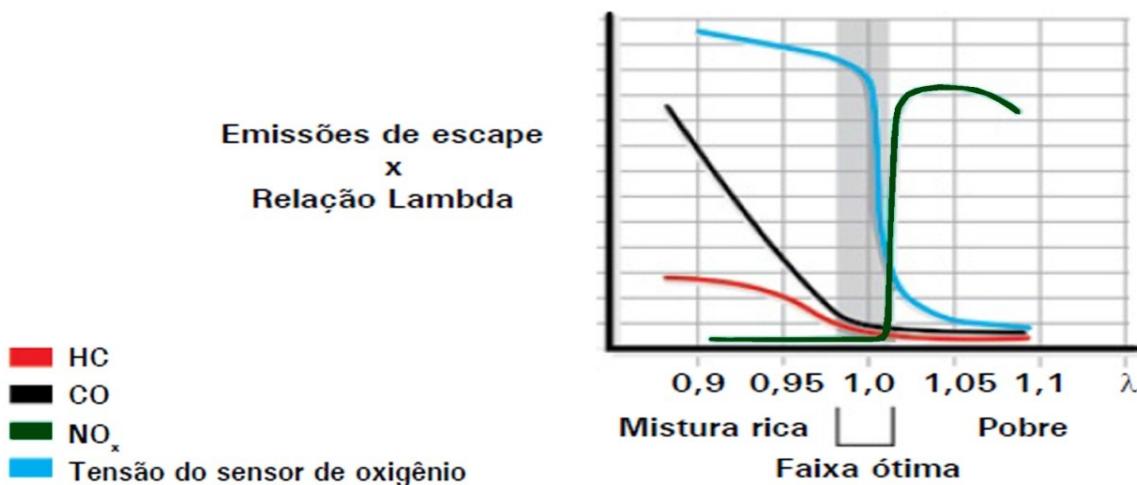
Tabela 1 - Relação estequiométrica em função do combustível.

Combustível	Ar/Combustível [Kg/Kg]
Gasolina Pura (E0)	14,7:1
Etanol (E100)	9,0:1
Gasolina Comum (E22)	13,2:1
Diesel	15,2:1
Metanol	6,4:1
Metano (CH_4)	17,2:1
Propano (C_3H_8)	15,6:1
Butano (C_4H_{10})	15,4:1
Hidrogênio	34,0:1

Fonte: (PUJATTI, 2007).

Uma mistura carburante estequiométrica, segundo Passarini (PASSARINI, 1993), só ocorreria em uma situação hipotética, ou seja, em um motor “ideal”. Em um motor real, diversos fatores ocasionam uma queima incompleta, como, por exemplo, a presença de impurezas no combustível, a presença de outros elementos além do oxigênio e o tempo reduzido para a reação entre o combustível e o oxigênio do ar. A figura 4 ilustra a relação entre o fator λ e as emissões na exaustão.

Figura 4 - Emissões e relação com fator λ .



Fonte: (VOLKSWAGEN, 2009).

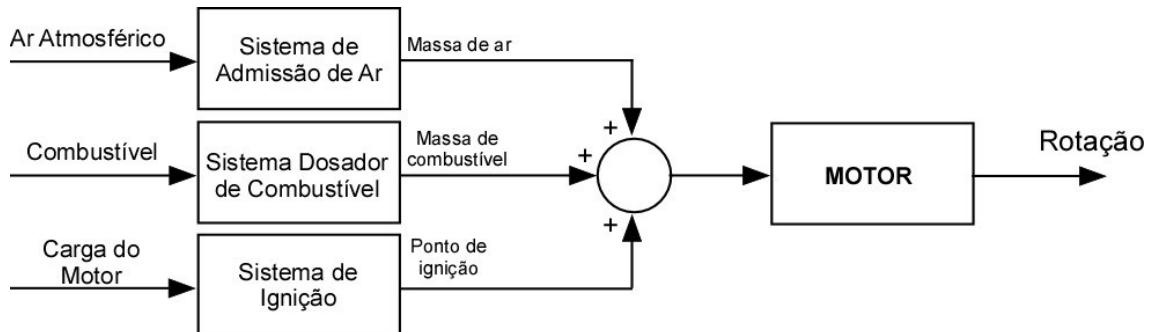
Os óxidos de nitrogênio (NO_x) são formados pela oxidação do nitrogênio atmosférico em temperaturas próximas a da combustão. A figura 4 mostra que a redução do NO_x no catalisador é favorecida com a redução do oxigênio nos gases de exaustão, e a oxidação do HC e CO, que são respectivamente hidrocarbonetos correspondentes ao combustível não oxidado durante a combustão e o monóxido de carbono, é mais eficiente com o aumento do oxigênio nos gases de exaustão.

2.2 CONTROLE DOS MOTORES DE COMBUSTÃO INTERNA CICLO OTTO

De uma forma geral, o motor Otto opera essencialmente com três parâmetros básicos para gerar torque de combustão, sendo eles: a massa de ar, a massa de combustível e a centelha elétrica da vela de ignição no momento correto, conforme o ciclo de operação descrito anteriormente.

O sistema de controle dos motores, independentemente de ser mecânico, eletromecânico ou eletrônico, deve ser capaz de estimar estes três parâmetros. Este controle pode ser em malha aberta ou fechada, dependendo das características construtivas de cada projeto. Existem diversas teorias de controle e modelos matemáticos sobre o assunto. Esta seção aborda a teoria geral de controle dos motores ciclo Otto, dividindo estes três parâmetros em três subsistemas, conforme ilustrado na figura 5 (HENDRICKS, 1997).

Figura 5 - Visão geral de um sistema de controle de um motor ciclo Otto.

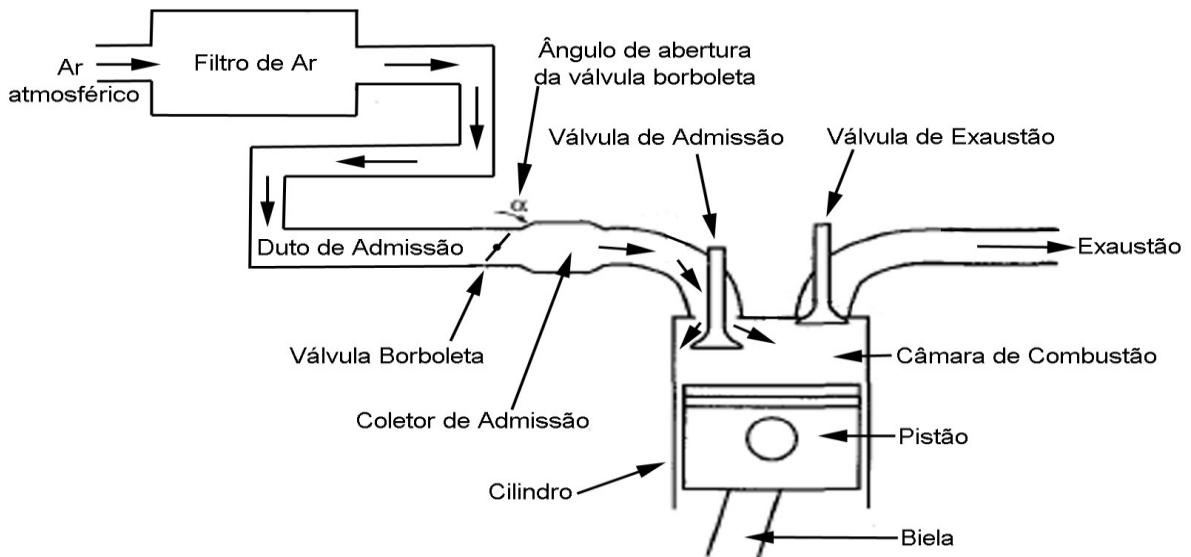


Fonte: o autor

2.2.1 Subsistema de Admissão de Ar

O fluxo de turbulência do ar atmosférico admitido nos cilindros exerce um importante papel na determinação das características da combustão e eficiência do motor. As principais funções deste subsistema consistem em determinar a massa total de ar atmosférico admitido, acelerar a velocidade de admissão dos gases e facilitar a homogeneização da mistura carburante, permitindo o enchimento do volume total dos cilindros. A turbulência e a velocidade de admissão são resultado dos processos de admissão e compressão, além da própria geometria dos dutos do sistema de admissão (POWELL e colaboradores., 1998).

Figura 6 - Componentes do sistema de admissão de um motor ciclo Otto.



Fonte: o autor.

O sistema de admissão é composto basicamente pelo filtro de ar, duto de admissão, válvula borboleta, coletor de admissão e válvula de admissão, conforme ilustra a figura 6. O filtro de ar possui a função de retirar as impurezas do ar atmosférico. O duto de admissão tem a função de guiar o ar previamente filtrado até a válvula borboleta que por sua vez tem a função de restringir a passagem de ar.

Em motores aspirados, o processo de aspiração normal do ar em pressão atmosférica ocorre em função de uma depressão no coletor de admissão do motor. Esta depressão é induzida pela sucção que o êmbolo produz ao se deslocar do PMS até o PMI (YOON e colaboradores., 2000).

Os principais parâmetros para o sistema de admissão são a pressão no coletor de admissão, a temperatura do ar admitido e a vazão total do ar. A vazão total do ar depende da geometria do duto de admissão e da posição angular da válvula borboleta (MILHOR, 2002).

Em sistemas mecânicos, o ângulo de abertura da válvula é determinado diretamente pelo motorista, através de um cabo de aço ligado ao pedal do acelerador. Em sistemas eletrônicos, a válvula borboleta é controlada indiretamente pelo motorista, pois é controlada pela unidade de gerenciamento eletrônico do motor. A vazão mássica está totalmente relacionada com o ângulo de abertura da válvula borboleta, pois, variando a área efetiva de passagem, o fluxo de massa de ar aumenta proporcionalmente.

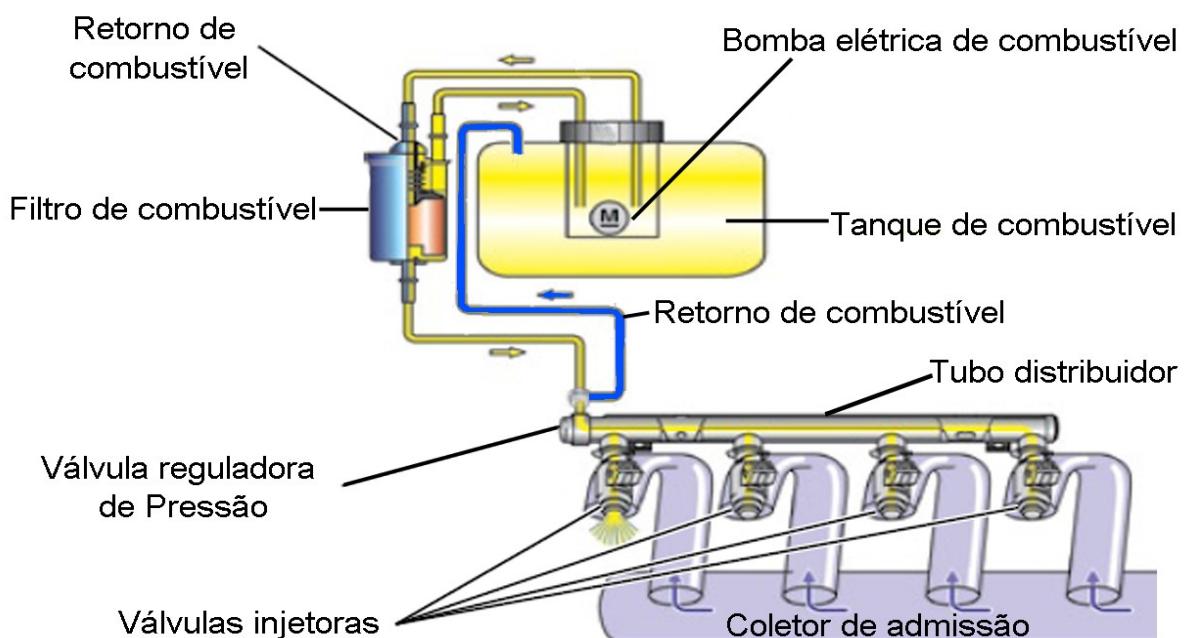
2.2.2 Subsistema dosador de combustível

O sistema dosador de combustível é responsável por controlar a quantidade ideal de combustível para cada condição de operação do motor. Seu principal objetivo é garantir que a massa de combustível admitida nos cilindros seja próxima da relação estequiométrica, tendo como principal parâmetro de entrada a massa de ar admitida pelo motor. Esse sistema deve garantir também o enriquecimento da mistura durante a partida e a fase de aquecimento do motor e limitar a rotação máxima do motor.

Os primeiros sistemas de controle de combustível eram mecânicos e baseados na sucção do combustível, atuando em função da diferença de pressão entre o coletor de admissão e a galeria de combustível. Atualmente, os sistemas de controle são, em sua grande maioria, eletrônicos e baseados no princípio da injeção de

combustível. O combustível, neste caso, é dosado através do controle do tempo de acionamento das válvulas injetoras de combustível.

Figura 7 - Sistema de injeção de combustível comum.



Fonte: (Adaptado de VOLKSWAGEN, 2009).

O sistema de combustível é composto basicamente pela bomba de combustível, pelo sistema de dutos de transmissão do combustível do tanque de combustível até as válvulas injetoras, conforme ilustra a figura 7. Independente do sistema de controle adotado, os principais parâmetros para o sistema dosador de combustível são a rotação do motor e a informação correta da massa de ar presente no coletor de admissão.

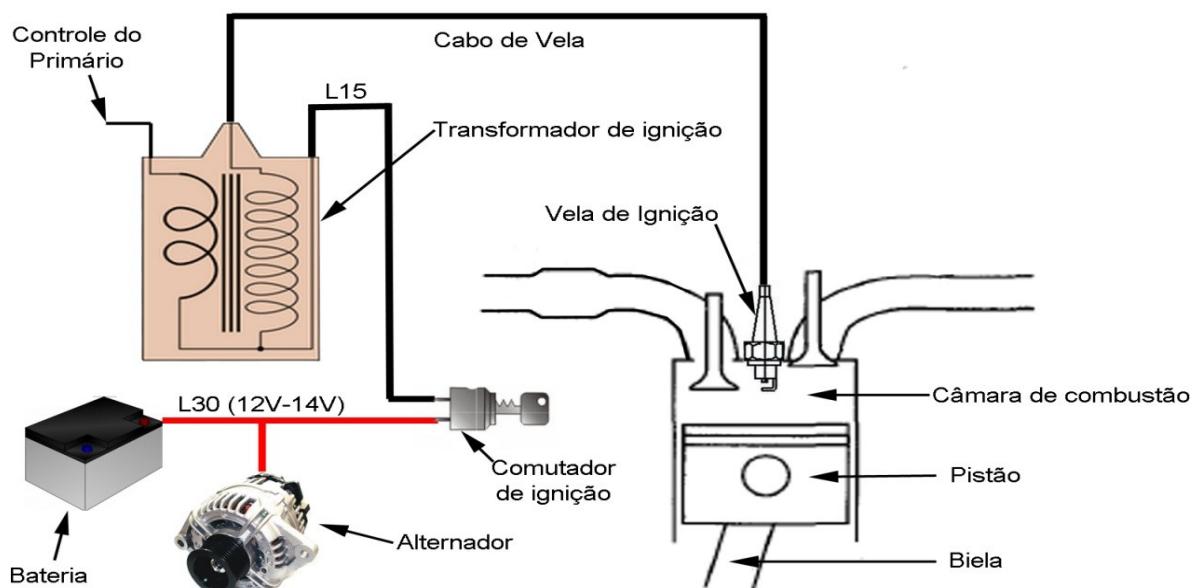
2.2.3 Subsistema de ignição

O sistema de ignição possui a função de produzir uma centelha elétrica dentro da câmara de combustão, promovendo a combustão da mistura carburante previamente admitida e comprimida nessa câmara. O sistema de ignição é responsável pelo fornecimento da energia necessária para produzir a centelha elétrica desde sua geração até o controle do instante que a centelha deve ser produzida. Segundo Pujatti (PUJATTI, 2007), a energia necessária para iniciar a

combustão da mistura carburante varia entre $200 \mu\text{J}$ para misturas estequiométricas até 3 mJ para misturas pobres ou ricas.

Para produzir corretamente a centelha elétrica, o sistema de ignição é composto pela bateria em momentos de partida ou alternador, quando o motor já está operando, pelo comutador de ignição, transformador de ignição, cabo de vela e finalmente vela de ignição, conforme ilustra a figura 8.

Figura 8 - Componentes do sistema de ignição.



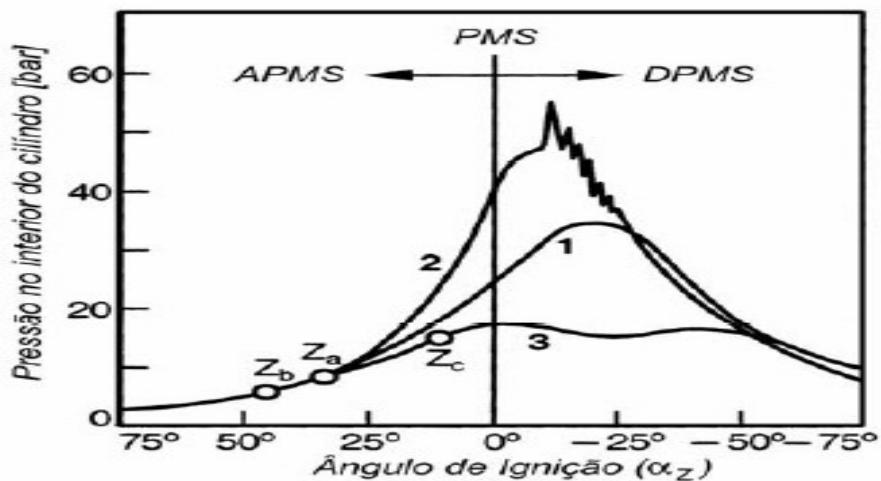
Fonte: o autor

O sistema de ignição é baseado no princípio de elevação da tensão do alternador através do transformador de ignição. O transformador é formado por dois conjuntos de espiras, definidas como primário e secundário. O enrolamento primário em uma de suas extremidades é permanentemente alimentado pela tensão positiva do pós-chave (L15), a outra extremidade é conectada ao sistema de controle, que possui a função de fornecer o sinal de aterramento, fechando o circuito elétrico e permitindo que haja circulação da corrente. A energia fornecida é diretamente proporcional à indutância do enrolamento primário e à intensidade da corrente que circula pelo primário. Como entre o primário e o secundário existe um núcleo central de aço revestido por um material isolante, a tensão no secundário é induzida e transferida para vela de ignição através dos cabos de vela que são capazes de isolar tensões da ordem de 40kV (PUJATTI, 2007). A centelha é formada pelo arco elétrico estabelecido entre a extremidade do eletrodo e a carcaça da vela.

O tempo em que o enrolamento primário permanece energizado pelo sistema de controle é denominado tempo de carga da bobina, que deve ser suficiente para garantir uma centelha capaz de iniciar a combustão da mistura carburante dentro da câmara de combustão. O sistema de controle do tempo de carga deve garantir que não ocorra a saturação do primário da bobina, evitando assim o aquecimento excessivo do enrolamento primário e a redução da vida útil da bobina de ignição.

Para que o motor possa aproveitar a máxima pressão da combustão quando o pistão estiver no PMS e, consequentemente, obter o máximo torque, a centelha elétrica deve ocorrer com pequenos graus de adiantamento em relação ao PMS. Conhecido como ângulo de ignição ao avanço de ignição, esse parâmetro é medido em graus do eixo da árvore de manivelas em relação ao PMS, durante o ciclo de compressão do motor. Este controle visa garantir que a maior parte da pressão gerada pela combustão seja convertida em trabalho pelo motor. Esse adiantamento do ponto de ignição não é constante e está diretamente ligado à rotação e carga do motor. O aumento do avanço de ignição promove uma variação crescente na pressão da mistura carburante e, consequentemente, na potência do motor. Entretanto, este avanço no ângulo de ignição é limitado pelo efeito conhecido como detonação, conforme ilustra a figura 9.

Figura 9 - Influência do avanço de ignição na pressão do cilindro.



Fonte: (PUJATTI, 2007).

Na figura 9, a curva 2 é caracterizada pelo efeito de detonação associado ao aumento do ângulo de avanço de ignição no instante Z_b . A detonação ocorre quando a temperatura e a pressão na mistura carburante não oxidada excede um valor

crítico, causando a autoignição em pontos diferentes e opostos ao sentido natural de propagação da combustão causada pela vela de ignição. A detonação gera ruídos característicos que nem sempre são percebidos pelo condutor e promove danos progressivos e irreversíveis aos componentes do motor (COOK e *colaboradores*, 1988).

2.3 GERENCIAMENTO ELETRÔNICO DOS MOTORES DE COMBUSTÃO INTERNA CICLO OTTO

Historicamente, dois tipos de sistemas de dosagem de combustível foram empregados com sucesso em motores de ciclo Otto: os carburadores, baseados na sucção de combustível, e os sistemas de injeção eletrônica, baseados na injeção de combustível. Os sistemas de gerenciamento eletrônico possibilitam o fornecimento do torque demandado pelo condutor enquanto asseguram economia de combustível, maximizam a vida útil do motor e reduzem a emissão de poluentes.

A seguir, serão descritos em detalhes todas as funções e os componentes de um sistema de gerenciamento eletrônico.

2.3.1 Histórico e evolução dos sistemas de controle nos motores Otto

Os três subsistemas, anteriormente descritos, eram controlados basicamente pelo carburador no controle da vazão de combustível em função da vazão de ar controlada pelo condutor, através de cabos de aço ligados ao pedal do acelerador e pelo distribuidor no controle do ponto de ignição.

Uma das principais limitações do carburador é que a dosagem de combustível é basicamente determinada pela abertura do giclé e, após ser calibrado, não se consegue realizar ajustes em tempo real. Por esse motivo e, considerando os diferentes regimes de operação do motor, o carburador não oferece a flexibilidade necessária para atender às rígidas legislações ambientais mais recentes (RIBBENS, 2003). A motivação para o controle eletrônico dos motores de combustão interna foi devido a dois fatores: a legislação sobre os limites na emissão dos gases tóxicos na exaustão e uma tentativa de promover economia de combustível. O controle da emissão dos gases de exaustão começou nos Estados Unidos em 1966 no estado

da Califórnia e, posteriormente, foi imposta pelo governo federal restringindo os padrões progressivamente (CÂMARA, 2006).

Gradativamente, o método de controle foi evoluindo e o primeiro sistema eletrônico de injeção foi lançado pela *Bendix* (BASSHUYSEN, 2004). Este sistema era analógico e adicionava válvulas injetoras, com o jato de combustível direcionado para as válvulas de admissão de cada cilindro. Em 1967, a *Bosch* lançou um sistema analógico de injeção de combustível, onde a pressão no coletor de admissão era medida e usada para calcular o tempo de injeção. Em 1979, foi lançado o primeiro sistema digital de gerenciamento do motor, denominado *Motronic* e desenvolvido pela *Bosch*. Este sistema integra os controladores de injeção e ignição em um único módulo. A partir deste sistema, iniciou-se o método de integração dos sistemas de controle em um único módulo de controle eletrônico denominado ECU (*Electronic Control Unit*) ou UCM (Unidade de Controle do Motor), passando a concentrar funções de controle da injeção de combustível, da marcha lenta, da ignição, do torque, recirculação dos gases, diagnose, dentre outros (MILHOR, 2002).

2.3.2 Conceitos e funções de uma unidade de controle do motor

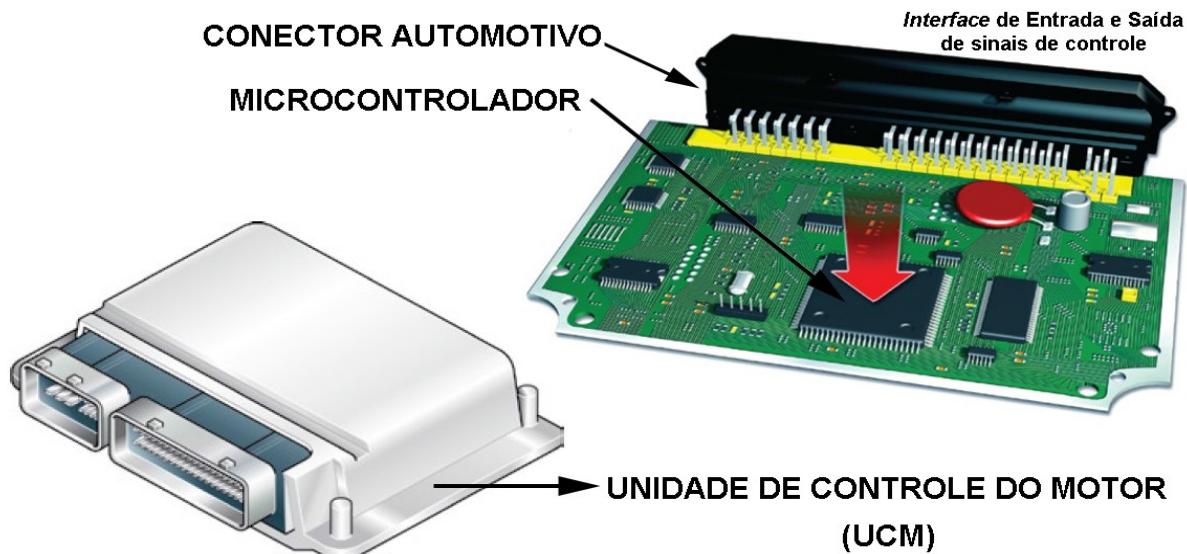
Os sistemas de injeção eletrônica são formados por múltiplos sensores e atuadores responsáveis pelo gerenciamento completo do motor. A evolução dos sistemas de gerenciamento eletrônico promoveu a redução do número de componentes mecânicos em ambos os sistemas, pela introdução de módulos eletrônicos de processamento de sinais. Os avançados sistemas existentes no mercado, aplicados aos motores de combustão interna, são capazes de controlar os parâmetros de funcionamento do motor, otimizando seu rendimento (GLIELMO, 2000).

O gerenciamento do motor, a partir dos subsistemas descritos, passou a ser possível com o desenvolvimento da eletrônica, o que viabilizou a utilização de sistemas microprocessados, realizando o controle digital do motor. Com a introdução do controle digital, através da discretização de sinais e monitoramento das condições de operação dos motores, foram desenvolvidos sistemas de controle integrados de formação e ignição da mistura carburante. Esse tipo de sistema de gerenciamento eletrônico é o principal responsável pela redução do impacto

ambiental causado pelos automóveis atuais, além de maximizar a vida útil do motor enquanto assegura economia de combustível (PUJATTI, 2007).

Para gerenciar o motor, a UCM recebe, através da interpretação de sinais elétricos fornecidos pelos diversos sensores no motor, um conjunto de informações relativas ao estado atual do motor, convertendo e discretizando essas informações de entrada para o processamento do microcontrolador. O microcontrolador é o responsável pela determinação da condição instantânea de operação do motor e pela atuação no motor através das saídas do módulo.

Figura 10 - Unidade de controle do motor e circuito eletrônico microcontrolado.

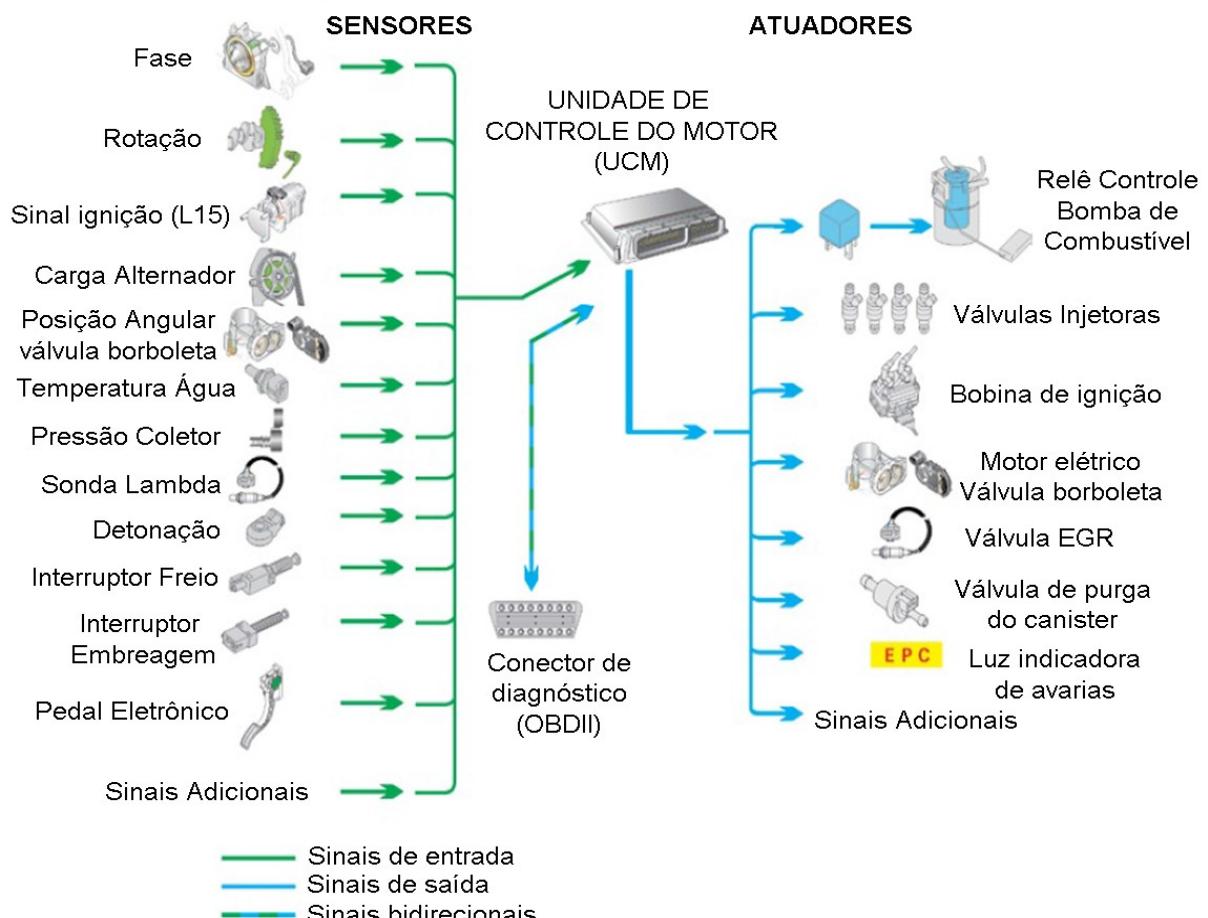


Fonte: (Adaptado de VOLKSWAGEN, 2009).

O microcontrolador possui partições de memórias dedicadas ao armazenamento do *firmware* principal. Normalmente, o *firmware* é gravado e armazenado em memórias do tipo flash EEPROM. Algumas unidades de controle do motor possuem memórias voláteis do tipo RAM para auxiliar na execução do *firmware*. Estas memórias podem ser integradas ao microcontrolador, ou externas, dependendo da característica de cada projeto. As principais saídas da unidade de controle do motor são responsáveis pelo controle do tempo de injeção de combustível, pelo tempo de carga da bobina e avanço de ignição, além da abertura da válvula borboleta. Atribui-se à UCM o controle de periféricos como, por exemplo, a bomba de combustível, o sistema de recirculação dos gases de descarga (EGR) e eletroválvula de purga do canister (BASSHUYSEN, 2004).

Um sistema de gerenciamento eletrônico digital contribui diretamente ao controle e manutenção da mistura carburante dentro dos limites definidos em todas as faixas de operação do motor, que se traduzem em um maior rendimento do motor nas regiões de economia de combustível, conforto e dirigibilidade. A divisão de tarefas do sistema físico da unidade de controle do motor (*hardware*) e do conjunto de instruções (*firmware*) é realizada de forma a maximizar o rendimento do sistema. Para que o sistema de controle possa gerenciar a operação do motor, é necessário que a UCM receba e processe os sinais de sensores indicando a condição de funcionamento do motor. Assim, sinais de comando são enviados para os atuadores, de forma que o motor opere de acordo com a condição determinada. A figura 11 ilustra os principais sensores e atuadores presentes em um sistema de gerenciamento eletrônico do motor.

Figura 11 - Principais sensores e atuadores de um sistema de gerenciamento.



Fonte: (Adaptado de VOLKSWAGEN, 2009).

Uma vez que esses sistemas são atualmente centralizados, concentrando diversas funcionalidades de controle e monitoramento do motor, pode-se atribuir as seguintes funções à UCM:

- Controle da injeção de combustível;
- Controle da ignição;
- Controle de marcha lenta;
- Controle da válvula borboleta eletrônica;
- Controle do sistema de arrefecimento do motor;
- Controle da recirculação dos gases de exaustão.

A seguir, serão descritos os principais sensores e atuadores presentes em um sistema de gerenciamento eletrônico do motor, indicando as grandezas medidas e os modos de controle para cada condição do motor. Cada modo de controle corresponde a uma determinada rotina realizada pelo programa de controle.

2.3.2.1 Sensores

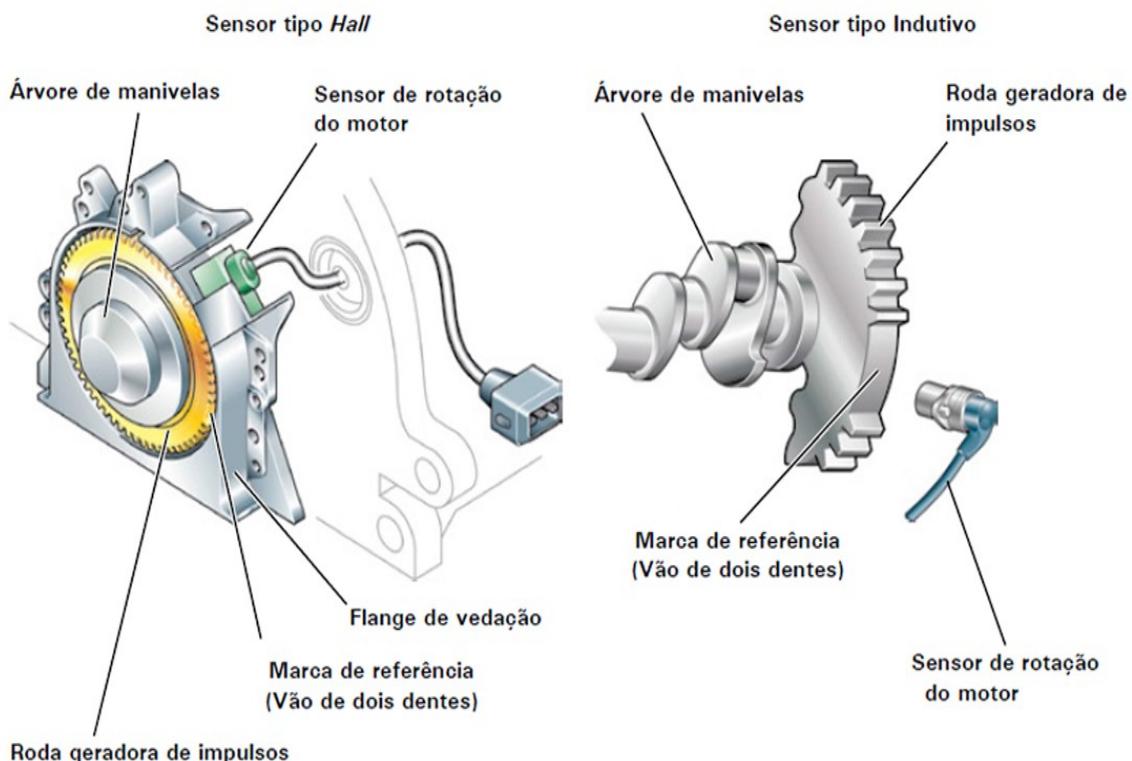
A unidade eletrônica de controle deve receber informações sobre a condição de operação atual do motor. Essas informações são enviadas para a unidade de controle através de sensores instalados no motor. A função de cada sensor e a quantidade existente no veículo varia de acordo com as especificações de cada projeto.

2.3.2.1.1 Sensor de rotação do motor

Os dois tipos de sensores mais comuns para se medir a rotação do motor são o sensor *Hall* e o de relutância variável. Eles determinam a rotação instantânea do motor e a posição da árvore de manivelas. A UCM utiliza a informação da rotação para o cálculo da massa de ar admitida, assim como para efetuar o sincronismo de todo sistema. Para isto, uma roda geradora de impulsos fixada no eixo da árvore de manivelas é utilizada para esses sensores. Através da roda geradora de impulsos, o sensor consegue determinar a posição exata da árvore de manivelas. A roda geradora de impulsos é uma peça mecânica parecida com uma engrenagem. É constituída de 60 dentes com a ausência de dois dentes, que é denominada falha. A

falla é utilizada como referência para efetuar o sincronismo do motor e a posição da árvore de manivela.

Figura 12 - Sensores de rotação do motor.



Fonte: (Adaptador de VOLKSWAGEN, 2009).

O sensor de rotação tipo *Hall* está fixado na *flange* do vedador traseiro do motor, onde também se encontra a roda geradora de impulsos, fixada à árvore de manivelas em posição predefinida. O sensor de rotação tipo indutivo está fixado na lateral do bloco do motor, próximo da embreagem. Neste ponto, também está fixada a roda geradora de impulsos em posição predefinida.

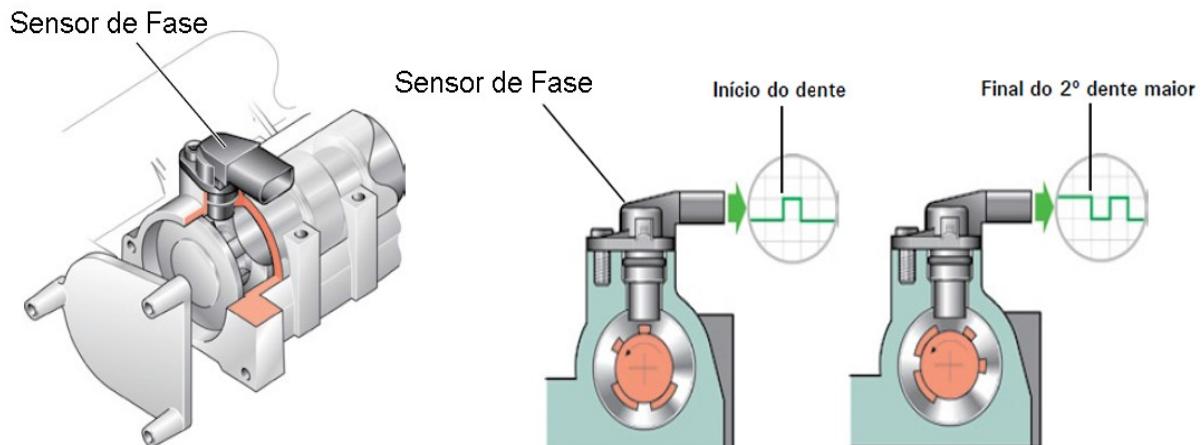
O sensor de relutância variável consiste em um imã de barra com um pino ferromagnético, sustentando uma bobina de indução com dois terminais (BOSCH, 2005). A tensão gerada por este sensor é definida pela intensidade do campo magnético que flui por esse circuito, bem como pela velocidade com que o material magnético (roda geradora de impulso) atravessa o sensor. Por outro lado, a intensidade do campo magnético é determinada pela relutância existente no circuito. Quando o dente está totalmente alinhado ao sensor, a tensão é nula, se elevando com o afastamento do dente e resultando em uma inversão no sentido da corrente, gerando uma tensão "negativa", que atingirá o ponto máximo aproximadamente na

metade do dente. Portanto, o sinal de tensão de saída de um sensor do tipo indutivo é um sinal senoidal que varia sua amplitude e frequência de acordo com a rotação do motor.

2.3.2.1.2 Sensor de Fase

Assim como o sensor de rotação do motor, o sensor de fase também é baseado no efeito *Hall*. Este sensor determina a posição do eixo do comando de válvulas de admissão para que a UCM, com a informação do sensor de rotação do motor, possa realizar a injeção de combustível de forma sequencial, isto é, liberar o combustível somente para o cilindro que iniciará a fase de admissão. Isto contribui para a redução das emissões de gases poluentes.

Figura 13 - Sensor de fase.



Fonte: (Adaptado de VOLKSWAGEN, 2009).

É fundamental que o sincronismo mecânico entre a árvore de manivelas e o eixo do comando de válvulas seja feito com critério e precisão, para que os sinais enviados pelo sensor de fase e de rotação sejam corretamente interpretados pela UCM.

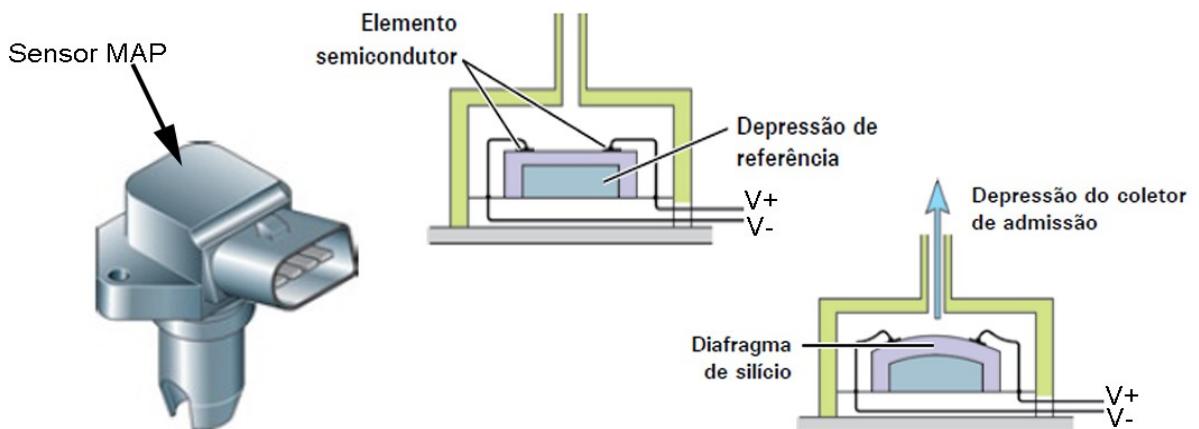
2.3.2.1.3 Sensor de Pressão do coletor de admissão

A pressão no coletor de admissão pode ser medida de forma direta pela UCM com o sensor de fluxo de ar MAF (*Manifold Air Flow*) ou de forma indireta, com o

sensor de pressão absoluta, também denominado MAP (*Manifold Absolute Pressure*) (YOON e colaboradores, 2000).

O sensor de pressão absoluta mede indiretamente o fluxo de ar admitido pelos cilindros. A função desse sensor é medir a variação de pressão absoluta no coletor de admissão, baseado num material piezo resistivo fixado a um diafragma de silício. Com o aumento da depressão no coletor de admissão, o diafragma se movimenta, gerando uma deformação nos piezo resistores que, por sua vez, alteram a resistência do seu material, que ao passar por um circuito de condicionamento eletrônico, gera uma alteração na tensão de saída.

Figura 14 - Sensor de pressão absoluta.



Fonte: (Adaptado de VOLKSWAGEN, 2009).

Este sensor é fundamental para a determinação da massa de ar admitida. Ele está localizado no coletor de admissão e está submetido às variações de depressão causadas pelas cargas impostas ao motor. É constituído por um elemento semicondutor, integrado a um diafragma de silício. A pressão interna no coletor age sobre o diafragma, causando a sua deformação. Como o elemento semicondutor está integrado a este, acaba se deformando e, conjuntamente, variando seu valor de resistência. Com a ignição ligada e o motor parado, a UCM utiliza o sinal deste sensor para medir a pressão atmosférica local, utilizada como referência nos cálculos de massa de ar admitida e do ângulo de avanço de ignição.

O sensor MAF tem como princípio físico o mesmo de um sensor de fio ou filme quente, que mede a quantidade do fluxo de massa de ar que passa por ele através do aquecimento ou resfriamento de um fio de platina, cuja variação de temperatura é

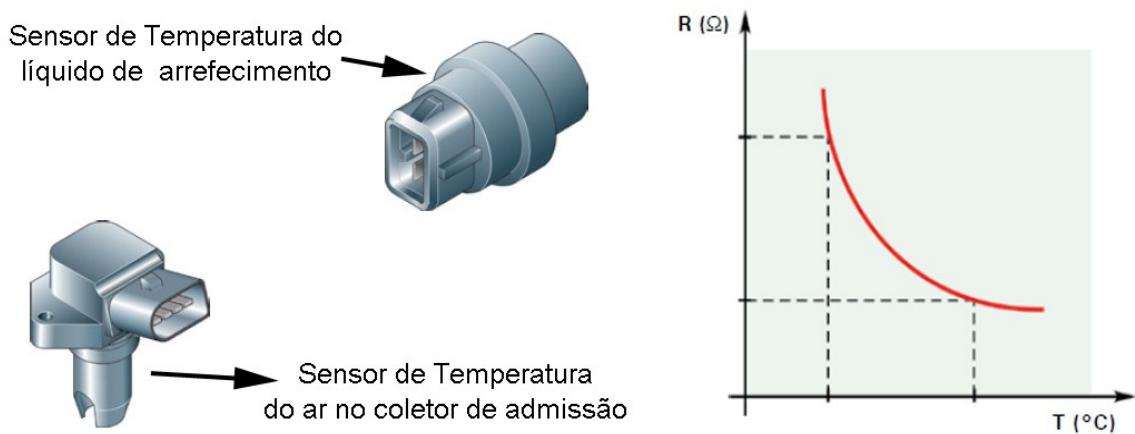
detectada por um termistor, que fica localizado logo após esse fio. Uma corrente elétrica percorre uma área de platina que, por consequência, carrega o valor resultante até um termistor localizado logo após a área de platina. Quanto maior a velocidade do fluido, maior é a taxa de transferência de calor e a mudança da temperatura do sensor (BOSCH, 2005).

2.3.2.1.4 Sensor de Temperatura

A medição da temperatura em motores ciclo Otto é geralmente realizada por sensores que variam sua resistência elétrica em função da temperatura. Nos motores ciclo Otto, a UCM normalmente realiza leituras nas temperaturas do líquido de arrefecimento e do ar dentro do coletor de admissão.

O sensor do tipo NTC (*Negative Temperature Coefficient*) emite um sinal elétrico inversamente proporcional à temperatura medida. Geralmente, sua resistência é construída de óxidos de metais que formam um material semicondutor cuja resistência varia rapidamente em função da temperatura.

Figura 15 - Sensor de temperatura NTC.



Fonte: (Adaptado de VOLKSWAGEN, 2009).

O sensor de temperatura do ar de admissão é de fundamental importância para o cálculo da massa de ar. Ele está localizado no coletor de admissão e está submetido às variações de temperatura do ar. Geralmente, é integrado ao corpo do sensor de pressão absoluta descrito anteriormente. A variação de sua resistência em função da temperatura do ar provoca uma queda de tensão no seu circuito de

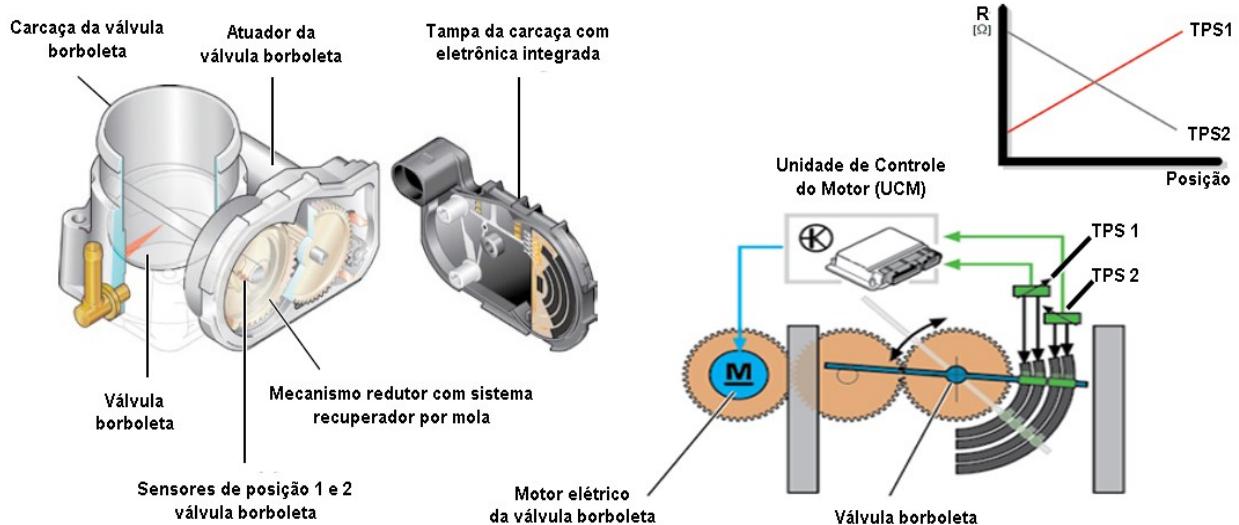
alimentação o que é interpretado pela UCM como a leitura da temperatura do ar de admissão.

O sensor de temperatura do líquido de arrefecimento permite à UCM realizar a estratégia de correção do tempo de injeção e ignição pois, com o motor frio, estes valores devem ser maiores, para acelerar o processo de aquecimento do motor. Esta informação é também importante para que a UCM comande o acionamento do eletroventilador de arrefecimento via relés convencionais.

2.3.2.1.5 Sensor de posição angular da válvula borboleta

O sensor de posição angular da válvula borboleta, também denominado TPS (*Throttle Position Sensor*), é utilizado para indicar o ângulo de abertura da válvula borboleta. Este sensor é constituído de um potenciômetro fixado no eixo da válvula borboleta e altera sua resistência elétrica proporcionalmente à variação angular da borboleta (GUEDES e colaboradores, 2012).

Figura 16 - Sensor posição da válvula borboleta.



Fonte: (Adaptado de VOLKSWAGEN, 2009).

Por razões de segurança, empregam-se dois sensores, que são potenciômetros variáveis. Os contatos dos cursores estão situados na engrenagem solidária com o eixo da válvula borboleta. Alterando a posição da válvula borboleta, variam as resistências nas pistas de contato dos potenciômetros e, assim, variam também as tensões dos sinais que são transmitidos à UCM. Os dois sensores usam

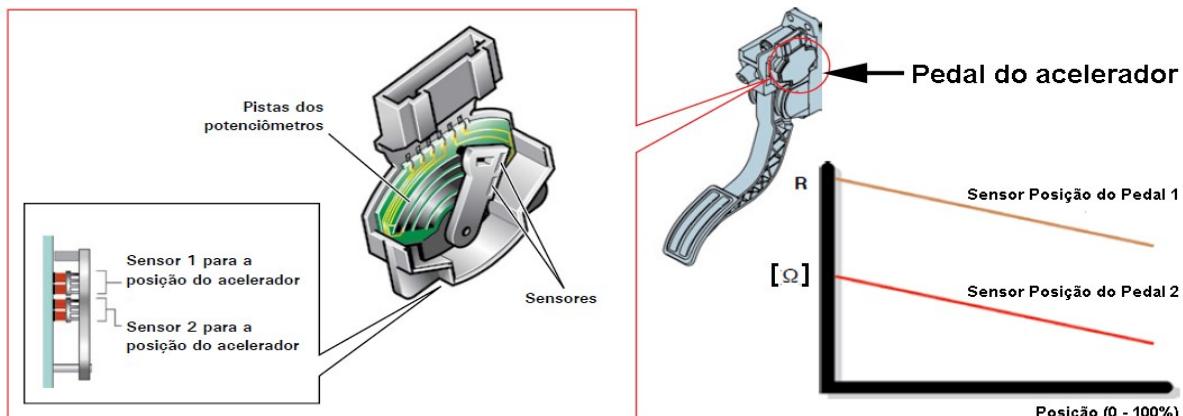
curvas de respostas opostas. Enquanto um sinal aumenta, o outro sinal diminui a resistência com a variação rotacional da válvula borboleta.

2.3.2.1.6 Sensor de posição do pedal do acelerador

Antigamente, o controle da válvula borboleta era efetuado através do pedal do acelerador. O acionamento era mecânico, através de um cabo de aço. Atualmente, esse sistema é elétrico, ou seja, a posição da válvula borboleta é regulada por um motor elétrico de corrente contínua. Nesse sistema, o pedal do acelerador passou a ter um sensor de posição e conforme o condutor pisa no pedal do acelerador sua posição é detectada pela UCM, que utiliza esse sinal como referência para determinar o torque do motor. A vantagem em utilizar o pedal como sensor é permitir que UCM defina a posição da válvula borboleta em função da determinação do condutor, mas atendendo também à dinâmica de enchimento do motor e às exigências de controle das emissões, consumo e segurança.

O sensor de posição do pedal do acelerador é constituído por dois potenciômetros. Empregam-se dois sensores para se obter os níveis máximos de confiabilidade dos sinais emitidos, através do sistema de redundância.

Figura 17 - Sensor posição do pedal do acelerador



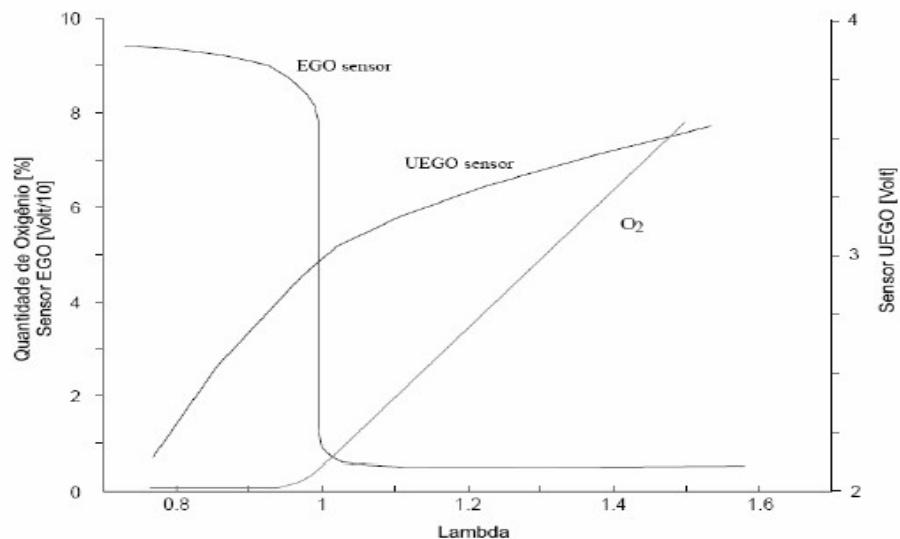
Fonte: (Adaptado de VOLKSWAGEN, 2009).

Através dos sinais procedentes de ambos os sensores, a UCM detecta a posição momentânea do pedal do acelerador. Alterando a posição do pedal do acelerador, as resistências dos potenciômetros se alteram variando assim a tensão transmitida à UCM.

2.3.2.1.7 Sensor de oxigênio dos gases de exaustão

O sensor de oxigênio, também denominado de sonda *lambda*, é capaz de medir a concentração de oxigênio nos gases de exaustão e, com base nessa informação, é possível estimar o valor real do fator *lambda* descrito na secção 2.2.2. O sensor mais utilizado atualmente é o EGO (*Exhaust Gas Oxygen*), também conhecido como sonda *lambda* de banda estreita (*Narrow Band Lambda Sensor*). Outro sensor de oxigênio é o UEGO (*Universal Exhaust Gas Oxigen*), também chamado de sonda *lambda* linear ou sensor *lambda* de banda larga (*Wideband Lambda Sensor*). Com esse sensor, é possível medir o valor exato do fator *lambda*, o que possibilita o controle da mistura também fora da região estequiométrica. A figura 18 ilustra os sinais elétricos fornecidos por esses sensores. Eles precisam de uma temperatura de operação acima de 400°C para que sua saída seja válida. Deste modo, os sensores usados atualmente são pré-aquecidos eletricamente.

Figura 18 - Curva característica dos sensores de oxigênio EGO e UEGO



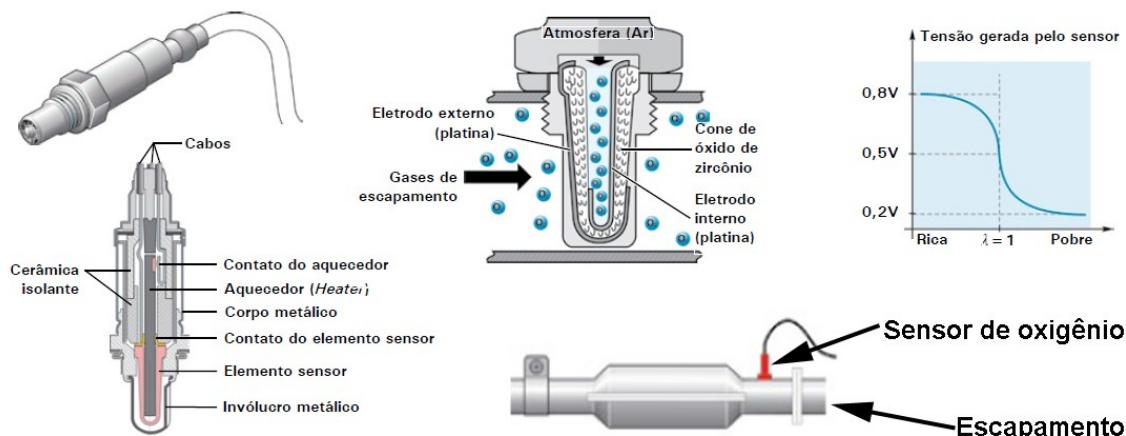
Fonte: (BRAGA, 2007).

O sensor de oxigênio está localizado no sistema de escapamento, geralmente próximo ao motor antes do catalisador. Através da análise adequada da presença de oxigênio na saída dos gases é possível equilibrar a relação da mistura carburante próximo da região estequiométrica. As mudanças de composição dos gases são consideradas pela UCM no controle das diversas funções e servem, geralmente,

como aviso para possíveis falhas. Por exemplo, caso algum sensor esteja fazendo uma medição incorreta para a UCM, esta também estará realizando os cálculos do tempo de injeção e do avanço de ignição incorretamente. O resultado disso será a formação de um mistura carburante fora dos parâmetros esperados que será detectada quando a sonda lambda informar a concentração de oxigênio nos gases de exaustão para UCM.

Para atingir a temperatura ideal de operação do sensor em poucos segundos, a sonda *lambda* possui quatro ligações, sendo duas para enviar o sinal elétrico do sensor até a UCM e as outras duas para controlar o aquecedor elétrico. O sensor de oxigênio opera em função da diferença da concentração de oxigênio entre o gás de exaustão e uma amostra de referência do ar externo, gerando uma diferença de potencial elétrico entre os eletrodos de platina, oscilando entre 200 e 800mV, dependendo do teor de oxigênio presente nos gases de exaustão (BRAGA, 2007).

Figura 19 - Sensor de oxigênio dos gases de exaustão.



Fonte: (Adaptado de VOLKSWAGEN, 2009).

Quando o motor trabalha com a mistura carburante rica, ou seja, *lambda* menor que 1, a diferença da concentração de oxigênio entre o gás de escape e a amostra de referência do ar externo é maior. Se o óxido de zircônio estiver com a temperatura acima de 400°C, ocorrerá a passagem de uma grande quantidade de íons de oxigênio entre os eletrodos de platina, causando um aumento na tensão gerada, que estará entre 500 e 800mV. Quando a UCM recebe esse sinal de tensão do sensor, corrige a mistura carburante, empobrecendo-a, buscando sempre atuar próximo da região estequiométrica.

Quando o motor trabalha com a mistura carburante pobre, ou seja, *lambda* maior que 1, a diferença da concentração de oxigênio entre o gás de escape e a amostra de referência do ar externo é menor. Se o óxido de zircônio estiver com a temperatura acima de 400°C, ocorrerá uma redução na passagem de íons de oxigênio entre os eletrodos de platina causando uma redução na tensão gerada, que estará entre 200 e 500mV.

2.3.2.1.8 Sensor de detonação

Para evitar que algum cilindro acabe gerando detonação por fatores externos fora do controle da UCM, este sensor é fixado diretamente ao bloco do motor e trabalha conforme o princípio piezo elétrico. Essa detonação pode ser proveniente de reações espontâneas de oxidação nas regiões quentes de carga não queimada, sendo que a combustão que caracteriza a detonação implica em uma ignição quase instantânea de parte da mistura remanescente, elevadas pressões localizadas, por ondas de choque e oscilações súbitas de pressão. Isto ocorre por conta deste indevido processo acelerado de combustão.

A detonação é extremamente prejudicial ao motor porque produz altíssimas pressões e temperaturas dentro do cilindro, podendo danificar irreversivelmente o pistão, os anéis de pistão, as válvulas, a junta de cabeçote ou a vela de ignição do cilindro em questão. Quando a UCM detecta o sinal de detonação do sensor, esta identifica qual cilindro estava em ignição e imediatamente atrasa o seu ponto de ignição em etapas, até a extinção do problema. Uma vez extinta a detonação no cilindro, a UCM volta a avançar o ponto de ignição, também em etapas, até o valor nominal de seu mapa característico (TERRINI, 1986).

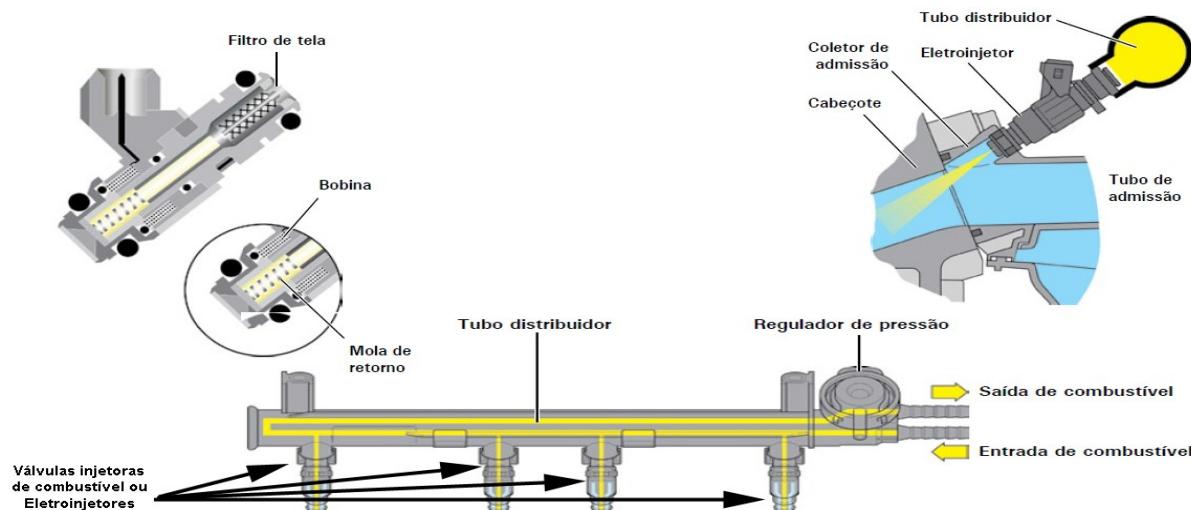
2.3.2.2 Atuadores

Atuadores são todos os componentes do sistema de controle responsáveis por gerar uma ação sobre a planta, a partir de um sinal de controle. De forma contrária aos sensores, os atuadores recebem da unidade de controle comandos para atuar sobre o motor de maneira a adequá-lo às condições desejadas de torque, potência e consumo. A seguir serão descritos os principais atuadores utilizados e suas respectivas funções (WANG, 2012).

2.3.2.2.1 Válvulas injetoras de combustível

Esses atuadores controlam a quantidade de combustível presente em uma linha pressurizada através da modulação da largura de um pulso de tensão.

Figura 20 - Válvulas injetoras e tubo distribuidor de combustível.



Fonte: (Adaptado de VOLKSWAGEN, 2009).

Em função da largura de pulso aplicado à bobina de indução, o campo magnético gerado desloca a agulha, permitindo a passagem do combustível. No instante em que a bobina de indução é desenergizada, a mola de retorno comprime novamente a agulha contra a válvula de vedação, bloqueando a passagem de combustível. No momento da desenergização, aparece uma tensão reversa nos terminais da válvula injetora de combustível, induzindo na agulha uma força no sentido contrário, que reduz o tempo de fechamento da válvula injetora.

O tempo de injeção é o termo que define a largura do pulso, durante o qual a válvula injetora permanece aberta e fornecendo combustível ao motor. Os principais fatores que limitam a definição do tempo máximo de injeção são a vazão máxima de combustível por milésimo de segundo de acionamento e a rotação máxima de operação do motor (PUJATTI, 2007).

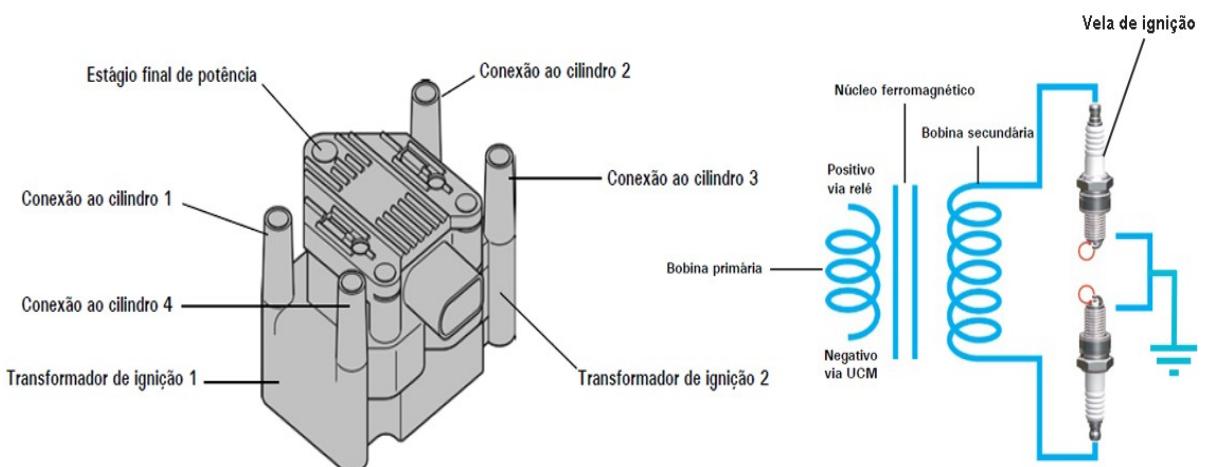
2.3.2.2.2 Bobinas de ignição

O controle eletrônico da ignição trabalha a partir do mapa de avanço da ignição do motor. Uma vez detectada a condição de operação, as informações armazenadas nas tabelas são recuperadas, corrigindo-se o ponto de ignição, que é função da

rotação do motor, pressão no coletor de admissão e temperatura do motor. Além de corrigir o avanço de ignição em função da condição de operação do motor, o gerenciamento eletrônico da ignição previne o fenômeno de detonação, de modo a atrasar o ponto de ignição quando a detonação aparece.

O principal atuador do sistema de controle eletrônico da ignição é o transformador de ignição, que também pode ser denominado bobina de ignição. Ela tem a função de transformar a baixa tensão, de aproximadamente 12V do circuito primário, na alta tensão do circuito secundário. O transformador de ignição é constituído de uma bobina primária, diretamente comandada pela UCM, um núcleo ferromagnético e uma bobina secundária, por onde sairá a centelha para as velas dos cilindros. Para elevar a tensão, a bobina secundária possui um número de espiras centenas de vezes maior do que o da bobina primária (TERRINI, 1986).

Figura 21 - Transformador de ignição.



Fonte: (Adaptado de VOKSWAGEN, 2009).

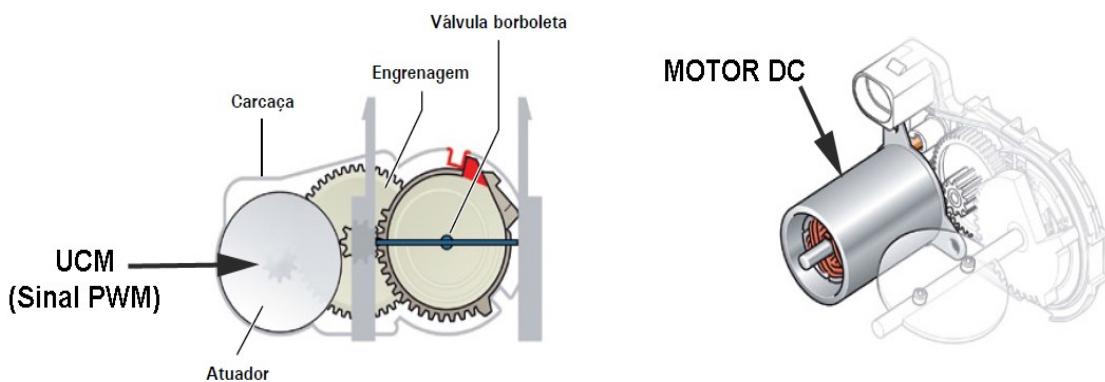
Quando um cilindro está em fase final de compressão, a UCM comanda o estágio final de potência, energizando a bobina primária do respectivo transformador, formando um campo magnético ao redor do núcleo ferromagnético. A UCM deve manter energizado o primário da bobina por um período suficiente para que esse campo magnético se forme com alta intensidade. No momento em que o adiantamento predefinido pelo mapa de ignição ocorre, a UCM desenergiza a bobina primária, fazendo com que o campo magnético criado atue agora sobre a bobina secundária, gerando uma corrente induzida. Em virtude da enorme quantidade de espiras da bobina secundária, a tensão induzida neste circuito é da ordem de milhares de Volts, sendo suficiente para que a resistência do dielétrico da vela de

ignição do cilindro de ignição seja superada, gerando uma centelha elétrica e iniciando a combustão da mistura carburante (BASSHUYSEN, 2004).

2.3.2.2.3 Atuador da válvula borboleta

A válvula borboleta é o principal atuador do sistema de admissão de ar. Com o intuito de propiciar uma melhor eficiência do motor e visando ainda atender às legislações ambientais, a válvula borboleta com acionamento eletrônico foi desenvolvida. Ela é formada por um motor DC, molas de retorno, engrenagens de transmissão e potenciômetros, para indicar a posição angular de seu movimento de abertura.

Figura 22 - Atuador da válvula borboleta eletrônica.



Fonte: (Adaptado de VOLKSWAGEN, 2009).

O motor DC da válvula borboleta eletrônica é acionado por um sinal de tensão. Para facilitar o controle deste atuador, a UCM geralmente utiliza um sinal de tensão contínuo, modulado por largura de pulso denominado PWM (*Pulse Width Modulation*). O PWM é um sinal pulsado de frequência fixa que possui uma modulação de sua razão cíclica com objetivo de modificar a tensão média final de saída. O motor DC atua através das engrenagens de transmissão, alterando a abertura da válvula borboleta em função do sinal PWM de entrada, ou seja, em função do valor de tensão média em sua entrada. Quando a tensão média do sinal elétrico de entrada for suficiente para que o motor DC vença as forças de atrito e da mola de retorno, a válvula borboleta irá se movimentar alterando sua posição angular. O sinal PWM é controlado pela UCM que, de acordo com a exigência de torque demandado pelo motor, atua na válvula borboleta, alterando sua posição

angular, e controlando a passagem de ar para o sistema de admissão (GUEDES, 2014).

2.4 ESTRATÉGIA DE CONTROLE

As seções anteriores descreveram os principais sensores encarregados de fornecer informações para que a UCM efetue todos os cálculos necessários para o gerenciamento dos subsistemas de combustível, ignição e admissão de ar. Também foram descritos os principais atuadores do sistema de gerenciamento eletrônico de um motor de combustão interna ciclo Otto. Para que sejam abordadas as estratégias de gerenciamento do motor, é preciso compreender quais os parâmetros utilizados por uma UCM para criar essas estratégias e como a UCM trata todos esses parâmetros. O gerenciamento eletrônico do motor, através da UCM, monitora, processa, calcula e atua no motor com o objetivo de otimizar seu rendimento. Entretanto, o motor possui diversos regimes de operação e, para cada regime, é necessária uma estratégia de controle diferente. Os principais regimes de operação que a UCM deve identificar e atuar são:

- Marcha lenta;
- Aceleração;
- Desaceleração;
- Partida do motor (motor frio e motor quente);
- Velocidade de cruzeiro;

O regime de marcha lenta se caracteriza pelo veículo parado com o motor ainda em regime de trabalho. Nessa condição, o motor permanece em uma rotação aproximadamente constante sem que o usuário pressione o pedal do acelerador ou engate alguma marcha. Toda energia produzida pela combustão é dissipada pelas perdas internas e pelo atrito. O objetivo do controle de marcha lenta é manter uma rotação de referência do motor no menor valor possível, para que o mesmo vença todas as perdas existentes e permaneça em funcionamento, produzindo uma combustão com qualidade aceitável, induzindo pouco ruído e vibração, além de suportar as cargas dos acessórios existentes no veículo. O controle de marcha lenta se torna complexo, pois em baixas rotações o torque disponível para rejeitar

perturbações é pequeno. O desafio é ainda maior pelo fato de se esperar uma grande variação nos parâmetros existentes de um veículo para o outro.

Quando o condutor pressiona o pedal do acelerador, caracteriza-se o regime de aceleração. Nessa fase, a massa de combustível fornecida ao motor deve ser incrementada adequadamente para se obter o torque solicitado pelo condutor. Para permitir que o motor forneça potência máxima, a UCM deve enriquecer a mistura carburante e, consequentemente, trabalhar fora da relação estequiométrica.

Quando motor está quente e com elevadas rotações e o condutor não pressiona o pedal do acelerador, deixando o mesmo em sua posição inicial, caracteriza-se o regime de desaceleração. A UCM deve efetuar um fechamento gradual da válvula borboleta para evitar solavancos indesejáveis no veículo e o corte da injeção de combustível visando eliminar a emissão de poluentes e reduzir o consumo de combustível, uma vez que, nesta situação, a inércia do motor é aproveitada pela UCM.

No regime de partida do motor, a UCM, após identificar o sinal de acionamento da chave de ignição, efetua a medida do sensor de temperatura do líquido de arrefecimento para determinar se o motor está frio ou quente. Com essa informação, a UCM aguarda o sinal de rotação do motor, que está girando com auxílio do motor de partida, e aciona todas as válvulas injetoras para garantir uma mistura carburante rica o suficiente para que o motor comece a operar nos quatro ciclos previamente descritos. Após a partida, caso o motor esteja frio, o motor entra em um estado de aquecimento, onde a UCM mantém a mistura carburante rica para aquecer rapidamente o motor. Conforme o motor aquece, a UCM corrige a mistura carburante, diminuindo o tempo de injeção até que o motor entre em regime de marcha lenta.

O regime de velocidade de cruzeiro é caracterizado pelo motor em regime de rotação constante. Geralmente ocorre quando o veículo está em viagem com velocidade aproximadamente constante e o condutor mantém o pedal do acelerador parcialmente acionado. Nessa condição, a UCM opera o motor em função da sonda lambda, buscando sempre a relação estequiométrica.

A principal função de um sistema de gerenciamento eletrônico de motores é fornecer o torque requerido pelo condutor enquanto assegura economia de combustível, maximizar a vida útil e reduzir a emissão de poluentes. O projeto de um controlador, ou seja, uma UCM, não está submetido apenas aos requisitos de

desempenho, mas também a medidas subjetivas, como a aspereza do funcionamento do motor, expresso pelo grau de ruído e vibração comunicado aos passageiros através da estrutura do veículo (ONO, 1989).

A abordagem do problema de controle de um motor de combustão interna ciclo Otto permite a utilização de várias modalidades de controle, que vão do clássico PID até o moderno LQG (*Linear Quadratic Gaussian*), passando pelos não convencionais, como lógica “fuzzy” e redes neurais. A seguir, são descritos alguns métodos e modelos matemáticos para controle do motor apresentados na literatura, que permitiram definir o modelo apropriado para atender aos requisitos específicos deste trabalho (BALANOVIC, 2002).

2.5 MÉTODO DE CONTROLE E MODELOS MATEMÁTICOS DO MOTOR OTTO

Diversas técnicas podem ser utilizadas para controlar o motor, apresentando respostas de controle bastante satisfatórias. A seguir serão detalhados métodos desenvolvidos previamente por autores que abordam técnicas PID (*Proportional-Integrative-Derivative*), com técnicas não lineares e compensadores, baseando predominantemente em teorias de controle clássico e adaptativo para modelar e controlar o motor de combustão interna.

2.5.1 Modelo do motor para aplicações de controle proposto por Hendricks

Em 1997, Elbert Hendricks da Universidade da Dinamarca (DTU) propôs um modelo matemático para aplicações de controle em motores de combustão interna ciclo Otto. Hendricks elaborou um modelo incluindo a dinâmica da árvore de manivelas e denominou o mesmo como Modelo de Valor Médio do Motor (MVEM). A essência do modelo é expressar a dinâmica completa do motor em função do tempo com precisão entre 1% e 3%, capturando um determinado comportamento do sistema de uma forma lógica, autoconsistente e compacta. O modelo desejado tem poucos parâmetros ajustáveis e é facilmente adaptado para outros motores (HENDRICKS, 1997).

Um motor de combustão interna a gasolina e com injeção eletrônica possui basicamente três subsistemas principais: dinâmicas do combustível, da árvore de manivelas e do coletor de admissão. Devido à natureza desses motores, estes

subsistemas são descritos de maneiras diferentes em seus modelos físicos. Hendricks (HENDRICKS, 1997) descreve o modelo simplificado da dinâmica do fluxo de injeção de combustível como:

$$\ddot{m}_{ff} = \frac{1}{\tau_f}(-\dot{m}_{ff} + X\dot{m}_{fi}) \quad \text{Eq.02}$$

$$\dot{m}_{fv} = (1 - X)\dot{m}_{fi} \quad \text{Eq.02a}$$

$$\dot{m}_f = \dot{m}_{fv} + \dot{m}_{ff} \quad \text{Eq.02b}$$

onde:

\dot{m}_{ff} Fluxo da massa de combustível $\left[\frac{kg}{s}\right]$,

τ_f Constante de tempo da evaporação do combustível [s],

\dot{m}_{fi} Fluxo da massa de combustível injetada $\left[\frac{Kg}{s}\right]$,

\dot{m}_{fv} Fluxo da massa de evaporação do combustível $\left[\frac{Kg}{s}\right]$,

\dot{m}_f Fluxo da massa de combustível dentro do coletor de admissão $\left[\frac{KG}{s}\right]$,

X Fração líquida do combustível depositada nas paredes do coletor [-]

O modelo do subsistema de injeção de combustível mantém o controle do fluxo de combustível (em vez da massa de combustível) no coletor de admissão ou próximo das válvulas de admissão de um motor.

A equação de estado da dinâmica da árvore de manivelas pode ser descrita como:

$$\dot{n} = \frac{1}{I_n}(-(P_1 + P_b) + H_u n_i \dot{m}_f (t - \tau_d)) \quad \text{Eq.03}$$

onde:

\dot{n} Velocidade do motor [RPM]

I Inércia do motor ($I_{ac} \cdot \left(\frac{\pi}{30}\right)^2 1000$)

I_{ac} Inércia atual do motor [Kgm^2]

P_1	<i>Perdas de potência do motor ($= P_f + P_p$) [kW]</i>
P_f	<i>Perdas mecânicas por atrito do motor [kW]</i>
P_p	<i>Perdas de potência previstas no ciclo de trabalho do motor [kW]</i>
P_b	<i>Perdas de potência do motor impostas pela carga [kW]</i>
H_u	<i>Pontos de baixa temperatura do combustível $\left[\frac{kJ}{Kg} \right]$</i>
n_i	<i>Eficiência térmica $\left[\frac{kW}{kJ} \right]$</i>
t	<i>Tempo [s]</i>
τ_d	<i>Tempo de atraso entre a injeção e o torque efetivo [s]</i>

O tempo de atraso entre a injeção e o torque efetivo é descrito por:

$$\tau_d = \frac{60}{n} \left[1 + \frac{1}{n_{cyl}} \right] + \frac{\rho_i l_e A_i}{\dot{m}_{ax}} \quad \text{Eq.04}$$

onde:

n_{cyl}	<i>Número de cilindros do motor</i>
ρ_i	<i>Densidade do ar admitido para o coletor de admissão</i>
l_e	<i>Distância efetiva entre o injetor e a válvula de admissão</i>
A_i	<i>Área efetiva do coletor de admissão</i>

O fluxo de massa de ar que passa pela válvula borboleta determina um tempo de atraso secundário, dependendo do tipo de sistema de injeção adotado. As perdas que ocorrem em todo o processo de combustão podem ser descritas empiricamente seguindo as teorias de atrito e fricção.

O último subsistema equaciona a dinâmica da pressão no coletor de admissão. Este modelo é uma derivação da conservação da massa de ar dentro do coletor de admissão e da lei dos gases ideais:

$$p_l = \frac{RT_l}{V_l} (-\dot{m}_{ap} + \dot{m}_{at}) + p_i \frac{\dot{T}_i}{T_i} \quad \text{Eq.05}$$

onde:

p_i Pressão absoluta no coletor de admissão [bar]

R Constante dos gases ideais $\left[\frac{kJ}{KgK} \right]$

T_i Temperatura no coletor de admissão [K]

V_i Volume do coletor + passagem da válvula borboleta [m^3]

\dot{m}_{ap} Fluxo da massa de ar dentro do coletor de admissão $\left[\frac{Kg}{s} \right]$

\dot{m}_{at} Fluxo da massa de ar sobre a válvula borboleta $\left[\frac{Kg}{s} \right]$

O fluxo de massa de ar pode ser expresso pela equação de velocidade e densidade do coletor de admissão:

$$\dot{m}_{ap}(n, p_i) = \frac{V}{120RT_i} (e_v p_i) n \quad \text{Eq.06}$$

$$\dot{m}_{ap}(n, p_i) = \frac{1}{30} \frac{V_d}{4RT_i} m'_i \cdot n \quad \text{Eq.06a}$$

$$\dot{m}_{ap}(n, p_i) = \frac{1}{30} m_i \cdot n \quad \text{Eq.06b}$$

Segundo Hendricks (HENDRICKS, 1997), uma das principais vantagens de se utilizar esse modelo é a descrição genérica e separada dos modelos de eficiência volumétrica, fluxo de massa de ar, corpo de borboleta e coletor de admissão. Esses modelos apresentam configuração modular e de fácil adaptação para outros motores. Existem muitos modelos MVEM na literatura que possuem um grande número de elementos em comum e parâmetros ajustáveis. Hendricks propõe a aplicação desses modelos para diversos motores, cuja validação é checada em regimes de estado estacionário e transitório.

O modelo proposto por Hendricks foi utilizado como base para se definir os cálculos do sistema de injeção e da eficiência volumétrica do motor. Os testes efetuados utilizando este modelo se comportaram satisfatoriamente e os valores teóricos calculados se aproximaram dos valores obtidos em testes práticos.

2.5.2 Modelo linear de controle proporcional integral proposto por Braga

Gabriel Braga (BRAGA, 2007) propôs obter o modelo diretamente através de ensaios. As ações de controle diretas abordadas são capazes de proporcionar boa resposta dinâmica às perturbações de vazão mássica de ar e rotação, compensando seus efeitos. Braga implementou um controlador proporcional integral. Para ajustar os ganhos proporcionais e integrais, foi utilizada uma abordagem clássica baseada em um modelo linear que caracteriza a relação entre o tempo de injeção aplicado e o valor de lambda obtido (BRAGA, 2007).

Para a determinação do modelo, Braga realizou ensaios variando o tempo de injeção e avaliando a resposta do sensor de oxigênio. Nos ensaios, o mapa de avanço de ignição e a posição da válvula borboleta foram mantidos constantes. Os ensaios foram efetuados utilizando-se de uma mistura rica, devido à oscilação natural do motor quando o mesmo trabalha na condição estequiométrica e, no caso específico da região de marcha lenta, o tempo de injeção que gera lambda igual a 1 é inversamente proporcional à rotação do motor.

Foram realizados testes com degraus positivos e negativos, com amplitudes diferentes, analisando a resposta do sistema. Ao final, foi obtida uma média dos valores, o que possibilitou estabelecer uma função de transferência do modelo, conforme equações de controle clássico (OGATA, 1985), em função do tempo de acomodação, valores de pico e constantes de tempo. O modelo obtido por Braga é descrito como:

$$G(s) = \frac{K}{\tau.s+1} \cdot e^{-\theta s} = \frac{-0,34}{0,22.s+1} \cdot e^{-0,2s} \quad \text{Eq.07}$$

onde:

$$K \quad \text{Ganho do sistema} \left[\frac{\text{lambda}}{\text{ms}} \right]$$

τ Constante de tempo [s]

θ Tempo morto [s]

Com base no modelo obtido Braga, um controlador PI foi elaborado pelo método da síntese direta elevada. A constante de tempo em malha fechada adotada foi de 0,2 segundos. Como este sistema apresenta uma razão entre o tempo morto e a constante de tempo elevada, a escolha da constante de tempo em malha fechada

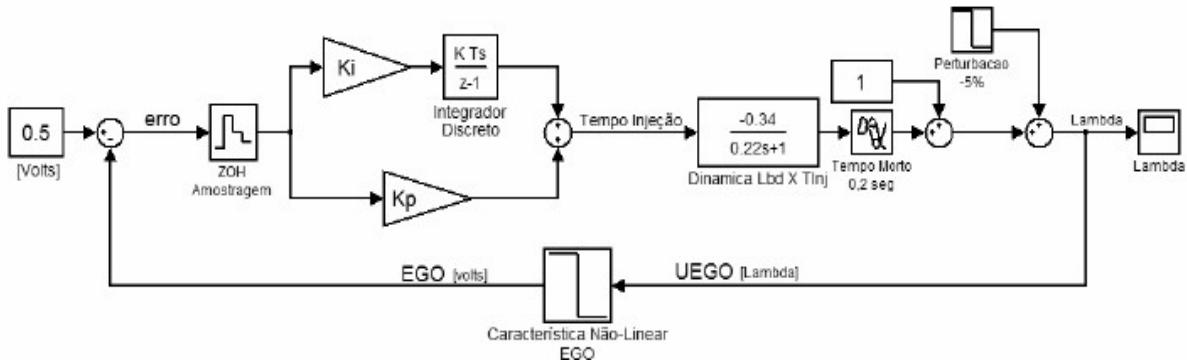
deve ser conservativa, evitando levar o sistema à instabilidade. Os cálculos dos ganhos proporcional (K_p) e integral (K_i) estão apresentados nas equações:

$$K_p = \frac{1}{K} \cdot \frac{\tau}{\theta + \tau_c} = \frac{0,22}{-0,34 \cdot (0,20 + 0,2)} = -1,62 \quad \text{Eq.08}$$

$$K_i = \frac{K_p}{\tau_i} \Rightarrow K_i = \frac{K_p}{\tau} = \frac{1,62}{0,22} = 7,36 \quad \text{Eq.08a}$$

Entretanto, Braga afirma que estes ganhos podem somente ser aplicados caso a realimentação seja feita através do sensor UEGO. A utilização do EGO implica em uma característica não linear na malha de realimentação. Deste modo, Braga usou uma estratégia que consiste em simular a malha de controle com um controlador PI e o modelo obtido com o sensor UEGO. O modelo de controle do motor em marcha lenta proposto por Braga é apresentado na figura 23.

Figura 23 - Modelo de controle do motor em marcha lenta proposto por Braga.



Fonte: (BRAGA, 2007).

O modelo consiste em um controlador PI ajustado para a função de transferência mostrada na equação (07). A medição do EGO é simulada através de uma aproximação da curva não linear do sensor. O controlador recebe o valor de erro, que consiste na diferença do valor atual da sonda lambda da referência (0,5), sendo utilizado um retentor de ordem zero (ZOH) para simular o efeito de amostragem (BRAGA, 2007).

O modelo de controle de *lambda* aplica sobre o tempo de injeção uma correção percentual, gerada pela tabela de injeção previamente calibrada com correções em função da tensão da bateria, cujo objetivo é manter uma relação direta entre o tempo de injeção e a massa de combustível injetada. Braga ressalta que o sistema de

controle desenvolvido é focado na condição de marcha lenta e, deste modo, não são abordadas condições de partida a frio, aquecimento do motor ou aceleração rápida. Entretanto, o desempenho do sistema proposto por Braga fica dependente de ações de controle através de calibrações prévias de mapas, tabelas e parâmetros, e tende a se deteriorar com o desgaste do sistema. Além disso, não são capazes de manter a mistura em torno da região estequiométrica com boa precisão, devido a fenômenos complexos da dinâmica do motor, que fazem com que o sistema apresente variações entre os ciclos, mesmo com todos os parâmetros controláveis constantes.

Apesar de mostrar vulnerabilidades em alguns pontos de operação do motor, o modelo se aquisição e o controlador proposto por Braga se comporta de maneira satisfatória dentro de regiões específicas de operação. Apesar de motores Otto serem sistemas MIMO, é possível realizar aproximações baseadas em sistemas lineares de primeira ordem e em síntese este é o método utilizado para se levantar o modelo de rotação do motor e elaborar o controlador PI de rotação proposto neste trabalho.

2.5.3 Identificação e controle da marcha lenta proposto por Nicolao, Rossi, Scattolini e Suffritti

O trabalho elaborado pelos autores apresenta uma aproximação integrada para a identificação e controle de motores de combustão interna na condição de marcha lenta. As entradas do modelo não linear de identificação são a posição do atuador de marcha lenta e o avanço de ignição, enquanto as saídas do modelo são a pressão no interior do coletor de admissão e a velocidade de rotação da árvore de manivelas. O modelo estimado é utilizado para sintetizar o controlador de marcha lenta utilizando técnicas de controle linear quadrático (NICOLAO e colaboradores, 1999).

Os autores desenvolveram um modelo do motor recorrendo às leis da física, combinando com procedimentos de identificação de sistemas para estimar alguns parâmetros desconhecidos como, por exemplo, a eficiência volumétrica do motor. Devido ao comportamento não linear do motor, os autores adotaram o método de estimação não linear autorregressiva exógena (NARX, *Nonlinear Autoregressive Exogenous*). Considerando um sistema não linear de tempo discreto com uma saída y e m entradas $u_1 \dots, u_m$, obtém-se:

$$y(T) = f(y(t-1), \dots, y(t-n_y), u_1(t-1), \dots, u_m(t-n_{u1}), \dots, u_m(t-n_{um})) \quad \text{Eq.09}$$

onde n_y e $n_{ui}, i = 1, \dots, m$, denotam o atraso máximo nas entradas e saídas. Se a função $y(T)$ se aproximar do polinômio de degrau M apresentado em (BILLINGS e colaboradores, 1989), a equação 09 pode ser escrita como:

$$y(t) = \sum_{i=0}^r \beta_i x_i(t) \quad \text{Eq.10}$$

onde r depende de $m, M, n_y, n_{ui}, i = 1, \dots, m$. Os parâmetros $\beta_i, i = 0, \dots, r$, são coeficientes a serem identificados. Os parâmetros $x_0 = 1$ e $x_i(t), i = 1, \dots, r$, são monômios formados por entradas e/ou saídas atrasadas. Por exemplo, se $n_y = m = n_u = 1, M = 2$ e se as forças forem positivas o modelo poderia ser descrito como:

$$y(t) = \beta_0 + \beta_1 y(t-1) + \beta_2 y(t-1)^2 + \beta_3 u(t-1) + \beta_4 u(t-1)^2 + \beta_5 y(t-1)u(t-1) \quad \text{Eq.11}$$

sendo que $\beta = [\beta_0 \ \beta_1 \ \dots \ \beta_r]$ é o vetor de parâmetros desconhecidos e denota uma sequência residual.

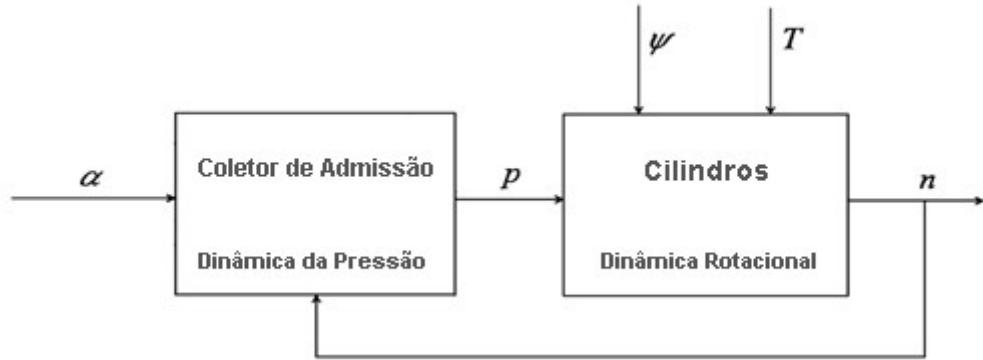
$$\varepsilon(t) = y(t) - \sum_{i=0}^r \beta_i x_i(t) \quad \text{Eq.12}$$

Assumindo que N observações das entradas e variáveis de saída estão disponíveis, os mínimos quadrados (LS) estima β^{LS} de β :

$$\beta^{LS} = \arg \min_{\beta} \sum_{t=1}^N \varepsilon(t)^2 \quad \text{Eq.13}$$

A aproximação proposta pode ser usada para determinar o modelo do motor como duas entradas e duas saídas. O esquema desenvolvido para o controlador de marcha lenta e para o modelo de produção do torque do motor é conceitualmente ilustrado na figura 24.

Figura 24 - Modelo conceitual de um motor de combustão interna.



Fonte: (NICOLAO e colaboradores, 1999).

Na figura 24, T representa as perturbações causadas pelo torque, p é a pressão do ar dentro do coletor de admissão, n é a velocidade da árvore de manivelas (RPM), Ψ é o avanço de ignição e α é a posição do atuador de marcha lenta. O modelo conceitual é composto de dois blocos, sendo o primeiro denominado Coletor de admissão, incluindo a válvula borboleta e o atuador de marcha lenta, e o segundo denominado Cilindros, que compõem a dinâmica rotacional do motor. De acordo com as tradicionais aproximações de desenvolvimento do modelo do motor, o modelo NARX do motor é composto de duas equações que descrevem a dinâmica da pressão no coletor de admissão p e a dinâmica rotacional do motor n em função das variáveis manipuladas α e ψ . O modelo da pressão é descrito por:

$$p(\theta) = f_p(p(\theta - i), n(\theta - i), \alpha(\theta - i), \psi(\theta - i), i > 0) \quad \text{Eq.14}$$

A identificação dos parâmetros desconhecidos do modelo da dinâmica rotacional é descrito por:

$$n(\theta) = f_n(n(\theta - i), p(\theta - i), \alpha(\theta - i), \psi(\theta - i), i > 0) \quad \text{Eq.15}$$

Como resultado, o método de identificação apresentado pelos autores e o desenvolvimento do controlador de marcha lenta foram utilizados pela empresa *Magneti Marelli* em aplicações comerciais. A adoção desta metodologia trouxe economia de aproximadamente 50% no tempo estimado necessário para o ajuste do ganho do controlador K .

Este método de identificação apresentou resultados bastante satisfatórios em simulações efetuadas com *software*, embora não tenha sido possível implementá-los na unidade microcontroladora para gerenciamento eletrônico desenvolvida neste trabalho. Embora o microcontrolador escolhido neste trabalho possua um poderoso núcleo de processamento, conforme descrito no capítulo 3, ao se adicionar as equações deste modelo no *firmware* elaborado, não se obtiveram resultados satisfatórios devido a atrasos provocados pelo processamento do microcontrolador, que não conseguiu atender à demanda do sistema de sincronismo do motor com os cálculos carregados deste modelo proposto.

2.5.4 Modelo auto-ajustável de controle aplicado na calibração de motores de combustão interna ciclo Otto proposto por Dorey e Stuart

O modelo proposto por Robert Dorey e Gregory Stuart descreve a aplicação de um conceito de controle auto-ajustável para a calibração de motores de combustão interna ciclo Otto. O objetivo é desenvolver um modelo de estado estacionário para calibração do avanço de ignição, utilizando a informação de pressão interna do cilindro enquanto o veículo está em movimento. O modelo avança o ponto de ignição ao ponto máximo de produção de torque do motor, observando o limite do fenômeno da detonação e auto-ajustando o ponto de ignição. Para isto, é aplicado o método de controle, empregando um estimador de mínimos quadrados recursivo e um controlador de extremos, suplementado com um regulador de P+I para acelerar a calibração inicial (DOREY e colaboradores, 1994).

Para os casos onde um sistema de apenas uma variável $u(t)$ é ajustada para maximizar a índice de desempenho escalar da função $J(t)$, é possível assumir as variáveis e correlacionar as funções por uma relação quadrática, então:

$$y(t) = au(t)^2 + bu(t) + c \quad \text{Eq.16}$$

onde a, b e c são os parâmetros da curva quadrática. Se a derivada dy/du é movimento conhecido entre u_1 e u_2 , pode-se calcular:

$$u_2 = u_1 + g \left[\frac{dy}{du} \right]_{u=u_1} \quad \text{Eq.17}$$

onde g é o ganho que controla a taxa de convergência. Entretanto, se a segunda derivada d^2y / du^2 é também acessível, o movimento é dado por:

$$u_2 = u_1 - \left[\frac{\frac{dy}{du}}{\frac{d^2y}{du^2}} \right]_{u=u_1} \quad \text{Eq.18}$$

Na ausência de ruído e variável de restrição, o algoritmo irá convergir em uma única etapa:

$$\left[\frac{dy}{du} \right]_{u_1(t)} = 2au_1(t) + b \quad ; \quad \frac{d^2y}{du^2} = 2a \quad \text{Eq.19}$$

e, aplicando, na equação 19, chega-se a:

$$u_2(t) = u_1(t) - \frac{2au_1(t)+b}{2a} = -\frac{b}{2a} \quad \text{Eq.20}$$

que fornece o valor da variável correspondente ao pico do índice de desempenho.

O somatório do quadrado dos erros é o melhor ajuste que minimiza o índice de desempenho. O somatório sobre n pontos da curva do índice de desempenho é obtida por:

$$J = \sum_{i=1}^n e_i^2 \quad \text{Eq.21}$$

Introduzindo o vetor θ , contendo os parâmetros do modelo, e o vetor X , contendo os dados de entrada medidos, a equação é descrita como:

$$y_1 = x_i^T \theta + e_1 \quad \text{Eq.22}$$

onde:

$$x_i = \begin{pmatrix} u_i^2 \\ u_1 \\ 1 \end{pmatrix} \quad \text{e} \quad \theta = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

Montando a sequência de saídas medidas dentro do vetor y , os erros dentro do vetor e , e as entradas dentro da matriz X , a informação de n ciclos do motor pode ser combinada em uma única equação:

$$y = X^T \theta + e \quad \text{Eq.23}$$

onde

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \quad X^T = \begin{pmatrix} u_1^2 & u_1 & 1 \\ u_2^2 & u_2 & 1 \\ \vdots & \vdots & \vdots \\ u_n^2 & u_n & 1 \end{pmatrix} \quad e = \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix}$$

Reescrevendo a equação 21 como:

$$J = e^T e = (y - X\theta)^T (y - X\theta) \quad \text{Eq.24}$$

Diferenciando em relação a θ para encontrar os parâmetros do modelo que minimizam J :

$$\frac{dJ}{d\theta} = -2X^T y + 2X^T X\theta \quad \text{Eq.25}$$

Definindo a derivada como zero e reorganizando a equação, é possível obter a estimativa de mínimos quadrados:

$$\theta = (X^T X)^{-1} X^T y \quad \text{Eq.26}$$

Uma vez que a , b e c foram estimados, a entrada que deverá maximizar a saída pode ser encontrada diferenciando a equação 16 que fornece:

$$u|_{max} = -\frac{b}{2a} \quad \text{Eq.27}$$

Os resultados dos experimentos demonstram que a estimativa do avanço de ignição ótimo varia de acordo com a iteração para um único elemento. O controle do valor de ignição torna-se uma estimativa otimizada e, assim, atinge um valor constante. O modelo proposto mostrou que o avanço de ignição pode ser calibrado com o motor em funcionamento utilizando a pressão interna dos cilindros. A combinação entre um controlador auto-ajustável e um regulador do avanço de

ignição é extremamente satisfatória e demonstra bom desempenho. Ao final do experimento, o motor ficou imune ao fenômeno de detonação, apresentando um torque elevado.

A forma incremental do estimador de mínimos quadrados, e os fatores não considerados no modelo apresentando, foram negativamente afetados pela variação no regime de operação do motor. A aplicação requer métodos robustos para variações nas medições e nos distúrbios do sistema.

A aplicação deste método de correção do sistema de ignição em tempo real foi o objetivo inicial deste trabalho. Entretanto, não foi possível aplicá-lo nos testes efetuados devido à dificuldades encontradas em processar de maneira adequada o sinal obtido do sensor de detonação. A aplicação deste modelo permitiu atuar em malha fechada no sistema de ignição, porém requereu a elaboração de complexas funções de tratamento digital do sinal do sensor de detonação para a correta identificação da ocorrência de detonações no motor.

2.6 CONSIDERAÇÕES FINAIS

As unidades de gerenciamento eletrônico controlam as funções do motor, recebendo sinais emitidos pelos diversos sensores e enviando sinais de comando para os atuadores existentes no motor, com a finalidade de ajustar o torque do motor em função da demanda do condutor visando diminuir as emissões, melhorar o rendimento e aumentar as condições de conforto e segurança.

Estas unidades de gerenciamento eletrônico consistem da combinação de *hardware* e *firmware*, que possuem a função de processar e controlar com precisão a formação da mistura carburante e a combustão através da massa de combustível, da vazão mássica de ar e do adiantamento da centelha de ignição.

O *hardware* da UCM deve ser capaz de condicionar os sinais elétricos de todos os sensores presentes no motor, adequando-os aos limites de operação do microcontrolador e atenuando, ainda, os ruídos presentes nesses sinais. Outra função atribuída ao *hardware* é amplificar a corrente elétrica dos sinais de comando aos atuadores, para que seja possível acioná-los devidamente, em função da condição atual de operação do motor.

O *firmware* de uma unidade de gerenciamento eletrônico deve ser capaz de estimar com a máxima precisão o estado atual de operação do motor e calcular

todos os parâmetros de operação do sistema. O *firmware* deve ser baseado em modelos matemáticos de controle de todos os subsistemas do motor, sendo que a integração do *firmware* com técnicas de controle digital melhoram significativamente o rendimento do motor.

Diversos modelos matemáticos e técnicas de controle visam aprimorar o rendimento térmico do motor. Em sua grande maioria, estes modelos apresentaram resultados satisfatórios em sua hipótese principal. Entretanto, devido ao grande número de variáveis, a dinâmica instável do motor e o reduzido tempo em cada ciclo de operação, os modelos apresentam vulnerabilidades em algumas condições de operação do motor. Este trabalho visa desenvolver um modelo através de ensaios e obtenção do sinal de resposta do sistema, em função do tempo de acomodação, dos valores de pico e constantes de tempo conforme equações de controle clássico (OGATA, 1985), contribuindo como ferramenta inicial para o desenvolvimento de novos modelos e técnicas de controle para aplicações em motores de combustão interna convencionais ou bicompostíveis.

O próximo capítulo apresenta toda a metodologia empregada neste trabalho, mencionando as ferramentas utilizadas para confecção do *hardware* e *firmware*, abordando a plataforma e os circuitos eletrônicos utilizados e suas respectivas funções na execução do sistema.

3 METODOLOGIA

A literatura apresenta diversos modelos matemáticos e métodos de controle que apresentam resultados satisfatórios. Entretanto, a aplicação prática destes modelos é dificultada devido à estrutura inicial exigida para a aplicação. A maioria dos modelos propostos foca no controle de apenas um atuador de todo o motor, deixando os demais atuadores constantes. Em alguns casos, o modelo proposto é tão específico que os demais atuadores simplesmente são ignorados. Para aplicar estes modelos, muitos pesquisadores elaboraram alternativas para controlar o respectivo atuador, deixando os demais atuadores sobre responsabilidade do próprio sistema original de controle do motor. Isto pode mascarar os resultados obtidos, além de limitar as possibilidades de novas formulações do modelo proposto. A ideia de se elaborar uma UCM tem como objetivo viabilizar a criação de novas estratégias de controle e permitir a aplicação dos modelos teóricos encontrados na literatura. Todo o processo de desenvolvimento da UCM proposta prevê uma análise sobre os sistemas que integram um motor ciclo Otto real. Deste modo, é possível aplicar variados métodos de controle, variando todos os atuadores, de acordo com as conveniências da aplicação, permitindo elaborar ou comprovar as estratégias presentes na literatura.

Tendo em vista o aumento contínuo de sistemas eletrônicos embarcados, a proposta deste trabalho é fornecer uma metodologia que resulte em uma ferramenta aberta para o desenvolvimento de uma plataforma unificada para o controle de motores a combustão. Com este intuito, inicia-se o estudo teórico e a aplicação prática dos conhecimentos adquiridos para a confecção de um *hardware* capaz de efetuar o gerenciamento completo de um motor de combustão interna. Descreve-se, a seguir, toda a metodologia aplicada para elaborar o *hardware* proposto e o respectivo *firmware* de controle embarcado utilizado neste projeto.

3.1 DESCRIÇÃO GERAL

O projeto foi elaborado, testado e aplicado em um veículo automotor equipado com um motor de combustão interna ciclo Otto. O modelo escolhido como plataforma de desenvolvimento foi o Volkswagen Polo Sedan 2.0L. Entretanto, os

conceitos fundamentais de toda a metodologia são gerais e extensíveis a qualquer outro veículo.

Foi estabelecido para o desenvolvimento desta unidade de controle do motor um processo evolutivo, passando por várias versões de *hardware*, adicionando-se a cada etapa novas funções. A primeira versão tinha como objetivo fornecer condições mínimas para efetuar o gerenciamento de um *mock-up*, com blocos básicos e concepção didática cujo desenvolvimento foi descrito em (ALCANTARA DIAS, 2011) e (ALBALADEJO, 2013). A segunda versão, que é o foco do trabalho descrito por (PEREIRA, 2013), tinha como objetivo fornecer condições de analisar os sinais emitidos pelos principais sensores do veículo, tentando colocá-lo e mantê-lo em funcionamento, controlando o tempo de injeção, avanço de ignição e a válvula borboleta eletrônica, tendo como principal referência a rotação do motor. A terceira versão, que é o foco deste trabalho, contempla todas as funções da segunda versão e ainda possui uma concepção centralizada, ou seja, apenas um microcontrolador para gerenciar todas as funções.

Os princípios de gerenciamento utilizados neste trabalho são extensíveis a todos os veículos equipados com motores de combustão interna ciclo Otto. O veículo utilizado neste trabalho, um Volkswagen Polo Sedan, foi fabricado no ano de 2004 e está equipado com um motor da família EA 113. As principais características deste motor, que será utilizado como referência neste projeto, são descritas a seguir.

3.1.1 Motores da família EA 113

Os motores da família EA 113 (VOLKSWAGEN, 2007) foram desenvolvidos com o objetivo de reduzir o peso dos componentes, aumentar a suavidade de funcionamento, aumentar o conforto na condução, otimizar as curvas de torque e potência e manter mínimo o impacto no custo de produção, através da utilização de componentes comuns a outras versões de motores. O gerenciamento nesta família de motores é em função do torque, originando a necessidade de inúmeros testes experimentais até se chegar ao produto final.

Os motores EA 113 possuem montagem transversal e, com rotações de 2400 RPM, desenvolve um torque máximo de 170Nm. Sua potência máxima é de 85 kW (116 cv) quando a rotação alcança 5200 RPM. A figura 25 ilustra o motor em uma visão geral, além de informar os dados técnicos como, por exemplo, curva de

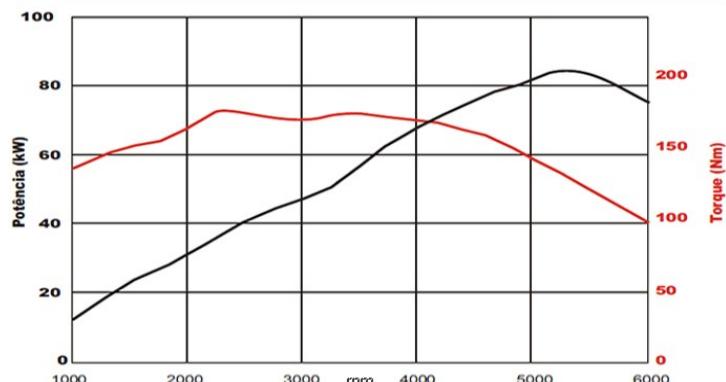
potência, curva de torque, cilindrada, taxa de compressão e sistema de gerenciamento original do motor. Todos os dados ilustrados na figura 25 foram fornecidos pelo próprio fabricante do motor.

Figura 25 - Ficha técnica dos motores da família EA 113 2.0L

EA 113 2.0L - Gasolina Ficha Técnica



Prefixo	BBX
Família	EA 113
Cilindrada	1984 cm ³
Diâmetro dos cilindros	82,5 mm
Curso dos pistões	92,8 mm
Taxa de compressão	10,5:1
Potência específica	43 kW/ℓ (58 cv/ℓ)
Torque específico	85,5 Nm/ℓ
Sistema de gerenciamento do motor	Bosch Motronic ME 7.5.10
Controle de emissão de poluentes	Regulagem Lambda e Catalisador

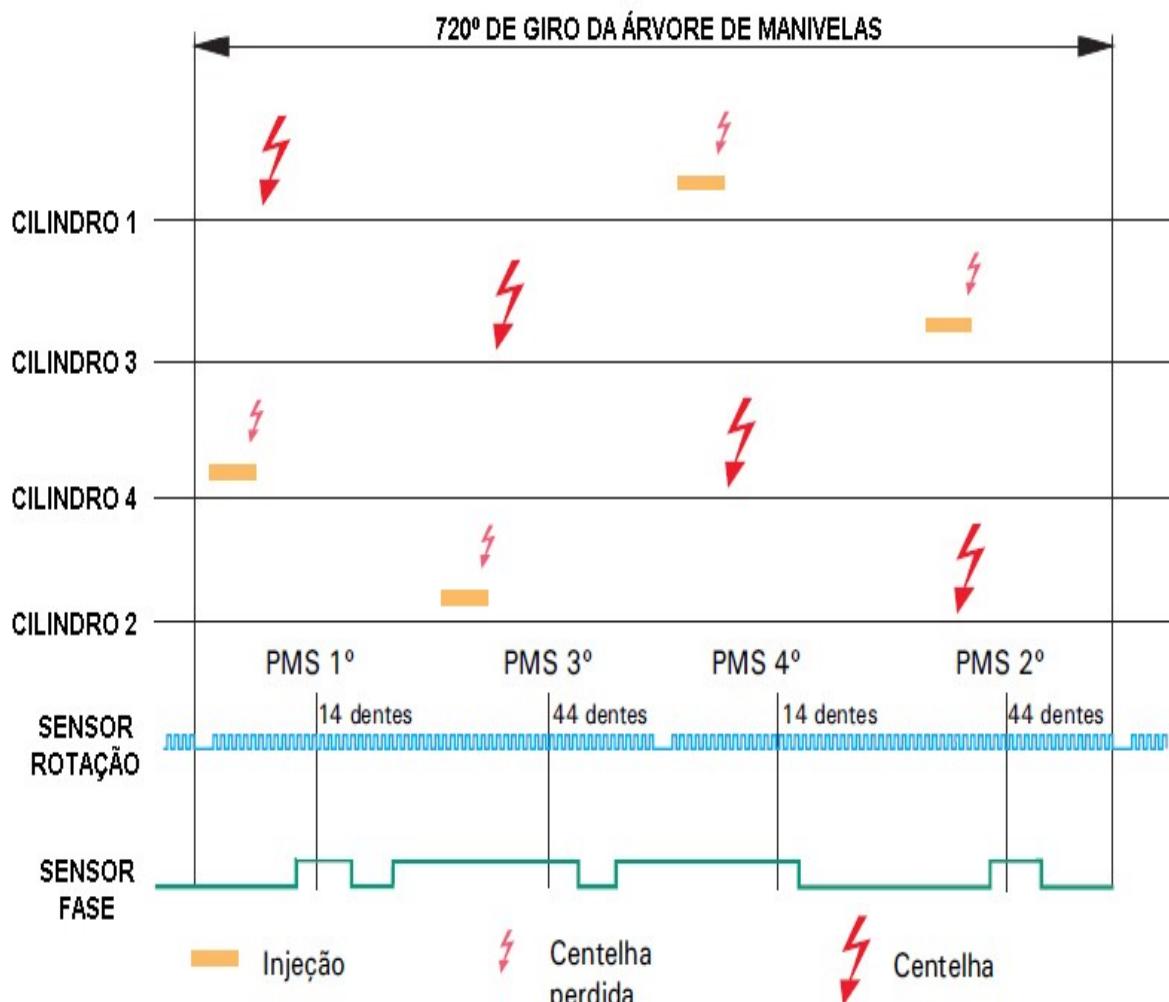


Fonte: (Adaptado de VOLKSWAGEN, 2009).

Eles são equipados com o sistema de gerenciamento eletrônico BOSCH, denominado ME 7.5.10. Nesses sistemas, a vazão de ar medida pelo sensor de pressão e temperatura deixa de ser o parâmetro central para o controle do motor, ou seja, o cálculo da massa de combustível injetada e o avanço da ignição passam a ser um parâmetro determinado pela unidade de controle do motor, a partir do torque que o motorista requer ao pisar no pedal do acelerador. Neste novo conceito de *firmware*, o acionamento do pedal do acelerador não será transformado diretamente em ângulo de borboleta, a UCM primeiramente determina o "Torque Objetivo" em função da porcentagem de acionamento do pedal do acelerador. Com o "Torque Objetivo" definido, a UCM fará o cálculo da quantidade de ar (ângulo de abertura da válvula borboleta), da vazão de combustível (tempo de injeção) e do avanço de ignição necessários para obter o torque solicitado através do pedal do acelerador na atual condição de operação do motor (VOLKSWAGEN, 2009).

A UCM dos motores da família EA 113 2.0L sincroniza a injeção de combustível, com base na informação da posição angular da árvore de manivelas em relação à posição angular da árvore do comando de válvulas utilizando como referência os sinais dos sensores de relutância variável (rotação) e fase, respectivamente.

Figura 26 - Sincronismo de comando eletrônico dos motores EA 113 2.0L.



Fonte: (Adaptado de VOLKSWAGEN, 2009).

A figura 26 mostra que os motores da família EA 113 operam em um sistema de bobina dupla, onde cada uma das bobinas atende dois cilindros (1-4 e 2-3). Enquanto um cilindro recebe a centelha no final do ciclo de compressão, o seu cilindro gêmeo recebe a centelha no final do ciclo de exaustão. O sinal elétrico do sensor de rotação já está previamente condicionado pela unidade de controle do motor UCM. O condicionamento do sensor de rotação e dos demais sensores e

atuadores também será executado na UCM desenvolvida neste trabalho. A seguir, são descritos todos os circuitos eletrônicos de condicionamento, além das informações relevantes ao desenvolvimento do *hardware*.

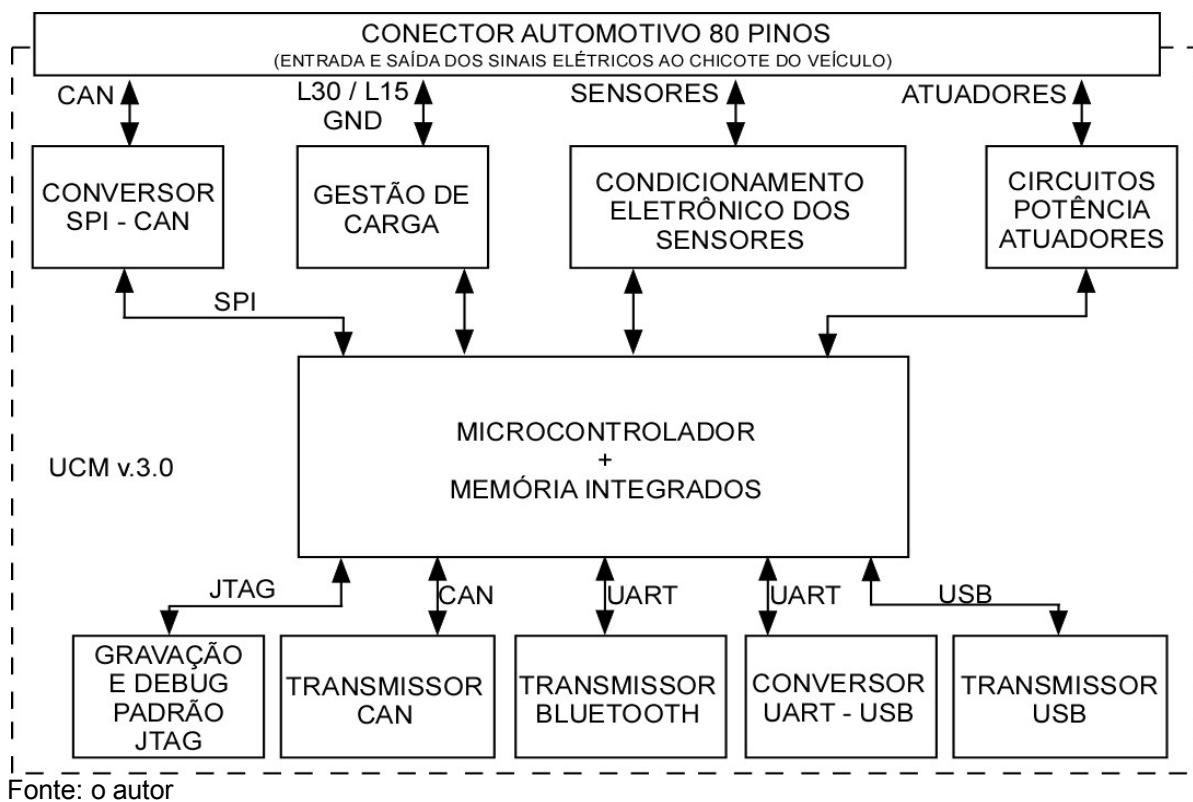
3.2 DESCRIÇÃO DO *HARDWARE* DA UCM

A unidade de comando do motor desenvolvida neste trabalho consiste de um sistema eletrônico microcontrolado capaz de gerenciar sinais elétricos provenientes dos sensores instalados em um motor de combustão interna e, a partir destas informações, fazer o seu controle em malha fechada. Os parâmetros de configuração do sistema, assim como a estratégia de controle do motor, serão disponibilizados em uma Interface Homem-Máquina (HMI).

Seguindo o processo evolutivo, em desenvolver diversas versões de *hardware* anteriormente descrito, a unidade de controle do motor atende às seguintes especificações de projeto:

- *Hardware* com concepção centralizada.
- Capacidade de armazenar mapas de ignição e injeção em memórias.
- *Design* de circuito impresso seguindo padrão automotivo.
- Conector automotivo (80 vias - padrão Motronic ME 7.5.10).
- Aplicação compatível para o Volkswagen Polo Sedan.
- Apenas aplicação de componentes eletrônicos padrão SMD.
- Interface Homem-máquina por comunicação externa.
- Circuito de gravação externo ao *hardware* (SEGGER J-LINK JTAG).
- Capacidade de debug de *firmware* durante a execução.
- Desenvolvimento de circuitos específicos e dedicados para cada sensor e atuador do projeto.

O *hardware* da UCM constitui-se de circuitos para condicionamento dos sinais dos sensores, circuitos de potência para acionamento dos atuadores, componentes para comunicação externa, além de circuitos de gestão de carga e memórias. A figura 27 ilustra o diagrama de blocos geral do *hardware*.

Figura 27 - Diagrama de blocos da UCM v3.0.

Fonte: o autor

A UCM foi projetada para atender às especificações deste projeto, porém estão previstas aplicações futuras nos campos de diagnose veicular e aquisição de dados em tempo real. Desta forma, o *hardware* possui a capacidade de se comunicar em duas redes USB e duas redes CAN, sendo uma rede externa para aplicações gerais e outra disponível no conector automotivo, que possibilita integração do *hardware* com a rede CAN do veículo. O *hardware* poderá ainda se comunicar em uma rede sem fio, utilizando o protocolo *Bluetooth*.

As seções seguintes apresentam o processo de desenvolvimento da unidade de controle, em que a função de cada bloco do diagrama, apresentado na figura 27, é detalhada.

3.2.1 Gestão de carga

O processo de levantamento de carga elétrica é importante e se faz essencial para o desenvolvimento de qualquer unidade eletrônica. O cálculo correto de todas as cargas previstas e a estipulação das possíveis sobrecargas é o primeiro processo para elaboração de circuitos eletrônicos microcontroladores. É fundamental determinar a potência de alimentação que garanta economia e segurança dentro dos

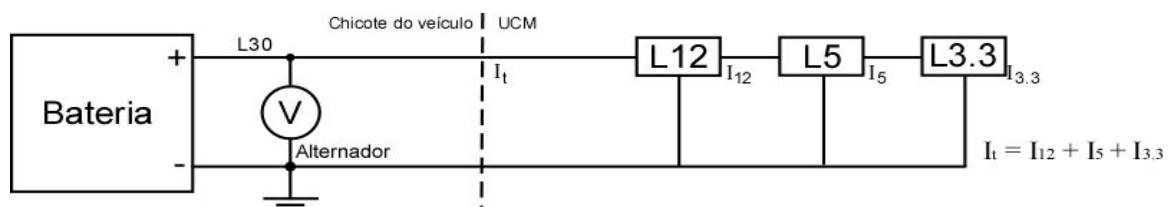
limites de temperatura e queda de tensão. Para isso, é necessário estimar quais componentes serão utilizados, assim como garantir uma reserva para futuras aplicações.

A previsão da carga de uma unidade eletrônica é a soma de todas as potências nominais dos componentes, utilizando como base o próprio catálogo técnico do componente. Nos casos em que o fabricante não fornece essa informação, é necessário calculá-la a partir de testes em bancada.

No setor automotivo, as UCM geralmente utilizam a bateria e o alternador como fonte de energia primária. A bateria, na grande maioria dos casos, gera uma diferença de potencial de aproximadamente 12.6V. O alternador, que opera quando o motor está funcionando, possui um regulador de tensão em sua saída que limita a diferença de potencial em 14V. Desta forma, foi definido que a UCM elaborada opere entre 9V e 15V.

O hardware possui circuitos integrados que operam com 12V, 5V e 3.3V. Deste modo, foram projetadas três linhas de tensão, utilizando circuitos integrados reguladores de tensão. A figura 28 ilustra o diagrama de operação do bloco de gestão de carga.

Figura 28 - Diagrama de operação do bloco de gestão de carga.



Fonte: o autor

Na entrada da UCM, existem componentes que operam com 12V como, por exemplo, o circuito de potência da válvula borboleta, que será descrito posteriormente. A tensão de entrada da UCM passa por um regulador de tensão que disponibiliza em sua saída 5V, que é utilizada para suprir grande parte dos componentes da placa, além de ser a tensão de referência para os sensores do motor. Assim, a tensão de 5V passa para um regulador de 3.3V que fornece energia para os componentes digitais da placa como, por exemplo, o microcontrolador. Para determinar o circuito integrado ideal para regular a tensão, foi estimado o consumo máximo, conforme catálogo técnico dos componentes, que segue:

Tabela 2 - Consumo máximo estimado da UCM desenvolvida

Componentes da linha 3.3V		
		Corrente máxima
01	Microcontrolador	100mA
02	Interface injetores	14mA
01	Interface sensor rotação	6mA
01	Controlador CAN	10mA
02	Led	10mA
Total de corrente estimada		164mA
Componentes da linha 5V		
		Corrente máxima
01	Componentes Linha 3.3V	164mA
11	Amplificador Operacional Duplo	30mA
01	Condicionador digital	16mA
02	Transmissor CAN	80mA
01	Transmissor USB	15mA
01	Transmissor Bluetooth	35mA
01	Gravador JTAG	10mA
01	Carga dos sensores	50mA
03	Amplificador diferencial	2mA
Total de corrente estimada		786mA

Fonte: o autor

A corrente máxima estimada para o regulador de 5V é de 786mA. Deste modo, foi definida uma reserva e o regulador escolhido possui corrente nominal máxima de 3A. Com o regulador de 3.3V, que possui corrente máxima estimada em 164mA, também foi definida uma reserva e escolhido um regulador que possui corrente nominal máxima de 800mA.

3.2.2 Microcontrolador

As especificações do projeto do *hardware* usaram uma concepção centralizada, ou seja, apenas um microcontrolador para gerenciar todas as funções da UCM. Tendo em vista a complexidade existente em um gerenciamento eletrônico do motor e a quantidade de sensores e atuadores a serem analisados e processados, o microcontrolador escolhido é da família Kinetis K40, modelo MK40X256ZLQ100, fabricado pela *Freescale Semiconductor*. Este microcontrolador possui um núcleo ARM CORTEX M4 de 32bits com possibilidade de instruções DSP, com alto desempenho e com baixo consumo de energia.

Todos os microcontroladores da família Kinetis incluem poderosos periféricos dedicados para efetuar o processamento de sinais analógicos, se comunicar com outros dispositivos e controlar processos e rotinas em função do tempo. (Freescale, 2010).

Algumas das especificações deste microcontrolador são descritas a seguir:

- Núcleo ARM CORTEX-M4 com 1.25*Dhrystone* MIPS por MHz.
- Instruções de processamento digital DSP.
- Periférico de acesso direto a memórias DMA com 16 canais.
- Frequência de operação do núcleo de até 100MHz configurável.
- Oscilador interno e multiplicador de frequência PLL e FLL.
- Microcontrolador com núcleo e barramento de 32bits.
- 2 periféricos de conversão analógico/digital de 16bits.
- 2 periféricos de conversor digital/analógico de 12bits.
- 3 periféricos de conversor digital/analógico de 6bits.
- 3 periféricos de gestão de tempo (TIMERS) com 12 canais cada.
- 6 periféricos de comunicação UART.
- 2 periféricos de comunicação CAN.
- 3 periféricos de comunicação SPI.
- 2 periféricos de comunicação I2C.
- Periférico avançado de comunicação USB.
- 512KB flash memory.
- 256KB flash.
- 256KB flexNVM.
- 4KB EEPROM/FlexRAM.
- 64KB SRAM.
- DEBUG padrão JTAG, cJTAG, SWD.
- 120 fontes de interrupção de *software* com 16 níveis de prioridade.
- Possibilidade de operação com sistemas operacionais RTOS.
- Compatível com ambiente integrado de desenvolvimento CodeWarrior™.

3.2.3 Condicionamento eletrônico dos sensores

Os sensores automotivos enviam sinais elétricos para a unidade de controle do motor. Para serem processados pelo microcontrolador, esses sinais devem receber um condicionamento prévio para adequá-los dentro das faixas de operação dos mesmos do microcontrolador. O microcontrolador opera com uma tensão de 3.3V e a maioria dos sensores operam com sinais que podem chegar a 5V, sendo que alguns sensores e interruptores operam com sinais que podem chegar a 12V. Para esta UCM, ficou determinado que os seguintes sensores devem conter condicionamento:

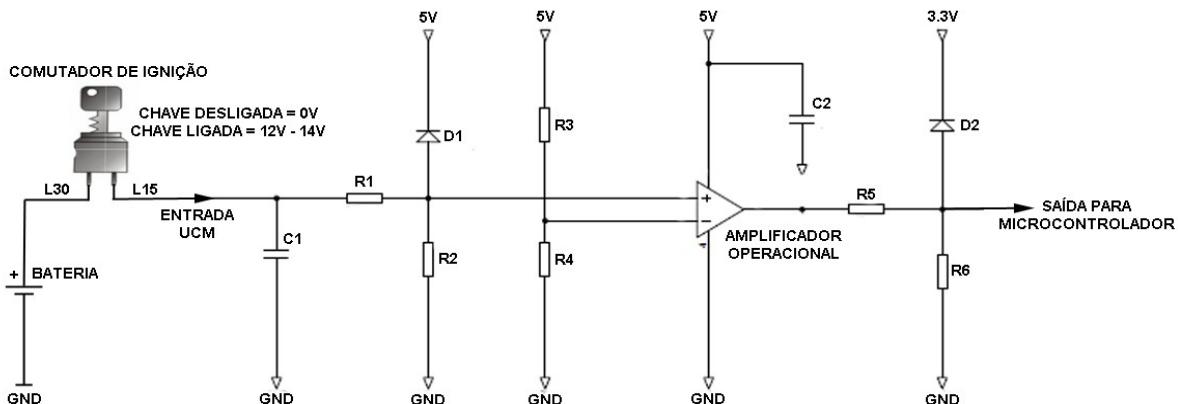
- Sinal de ignição (linha 15).
- Sensor de rotação e PMS.
- Sensor de fase.
- Sensor de pressão absoluta no coletor de admissão (MAP).
- Sensor de temperatura do líquido de arrefecimento.
- Sensor de temperatura do ar de admissão.
- Sensor de posição da válvula borboleta (TPS).
- Sensor de posição do pedal do acelerador.
- Sensor de oxigênio (Sonda lambda).
- Sensor de detonação.
- Interruptor de embreagem.
- Interruptor duplo do pedal de freio.
- Sinal de carga do alternador.

3.2.3.1 Sinal de ignição (Linha 15)

O sinal de ignição, que é a referência para UCM, tem como objetivo informar o microcontrolador que o motor está prestes a partir. Ele é proveniente do comutador de ignição. Quando o condutor vira a chave do veículo da posição inicial até a posição intermediária, o comutador envia um sinal elétrico de aproximadamente 12V para a UCM. Nesse instante, o microcontrolador deve efetuar a leitura prévia de todos os sensores e aguardar o sinal de rotação do motor para começar a controlar a injeção e a ignição, para que o motor entre em funcionamento sem auxílio do

motor de partida. O sinal da ignição é similar ao sinal de um interruptor elétrico comum atuando como uma chave elétrica. O esquema elétrico de condicionamento do sinal de ignição é ilustrado na figura 29.

Figura 29 - Condicionamento do sinal de ignição.



Fonte: o autor

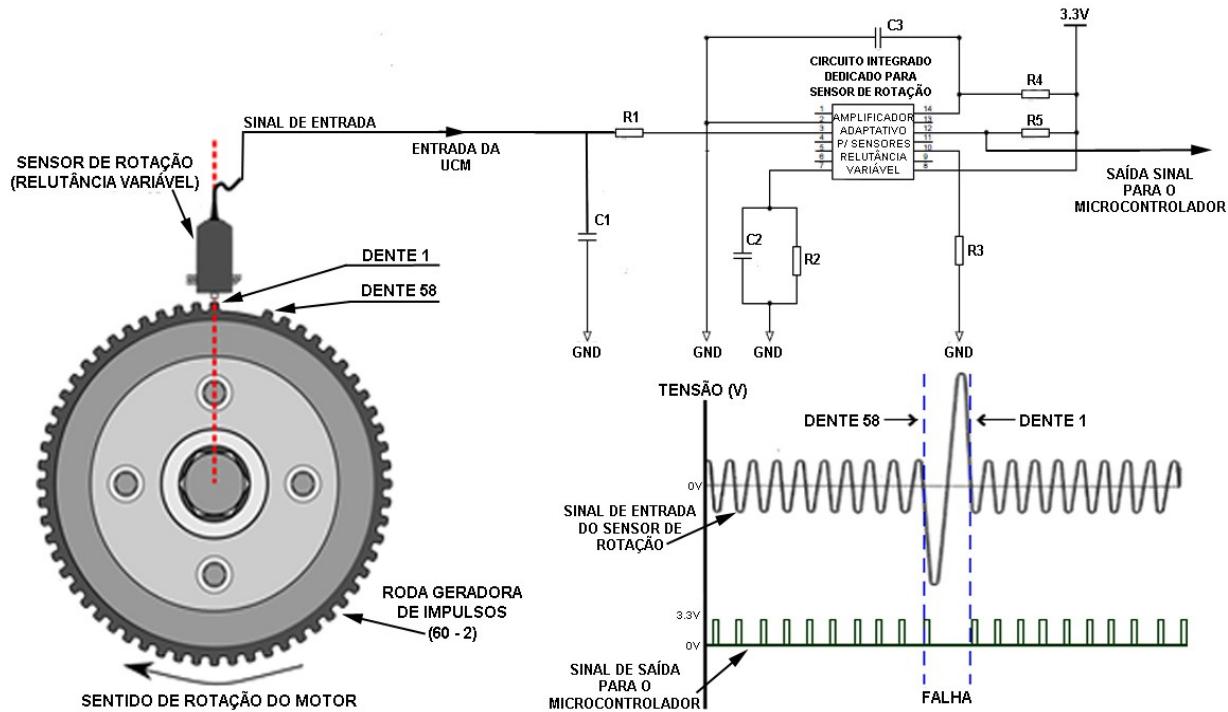
Nesse circuito, o sinal de entrada da UCM, que possui aproximadamente 12V, passa por um divisor resistivo formado por R1 e R2 que, com auxílio do diodo D1, tem como objetivo limitar a tensão para aproximadamente 5V na entrada do amplificador operacional. Ela compara a tensão de entrada no terminal positivo com uma tensão de referência em seu terminal negativo, formado pelo divisor resistivo de R3 e R4. Caso a tensão no terminal positivo seja maior que a tensão no terminal negativo, o amplificador operacional disponibilizará em sua saída uma tensão de 90% de 5V. Antes de entrar no microcontrolador, esse sinal passa por mais um divisor resistivo, formado por R5 e R6 com auxílio do diodo D2, para limitar a tensão de saída para o microcontrolador para o máximo valor compatível com o microcontrolador de 4V.

3.2.3.2 Sensor de rotação e PMS

O sinal deste sensor é de extrema importância para toda operação do gerenciamento eletrônico do motor. A UCM sincroniza todo o sistema, determina o momento exato para acionamento das válvulas injetoras de combustível e o avanço de ignição em função do sinal do sensor de rotação e PMS. Os motores da família EA 113 2.0L possuem o sensor de relutância variável e, conforme descrito anteriormente, o sinal deste sensor possui uma forma senoidal que é inadequado

para o processamento em alta velocidade por parte do microcontrolador. O ideal, neste caso, é que este sinal senoidal seja convertido para um sinal do tipo quadrado para que o microcontrolador, através das interrupções por borda de sinal, possa processá-lo de forma eficaz. O esquema elétrico do circuito de condicionamento do sensor de rotação e PMS é ilustrado na figura 30.

Figura 30 - Condicionamento do sinal do sensor de rotação e PMS.



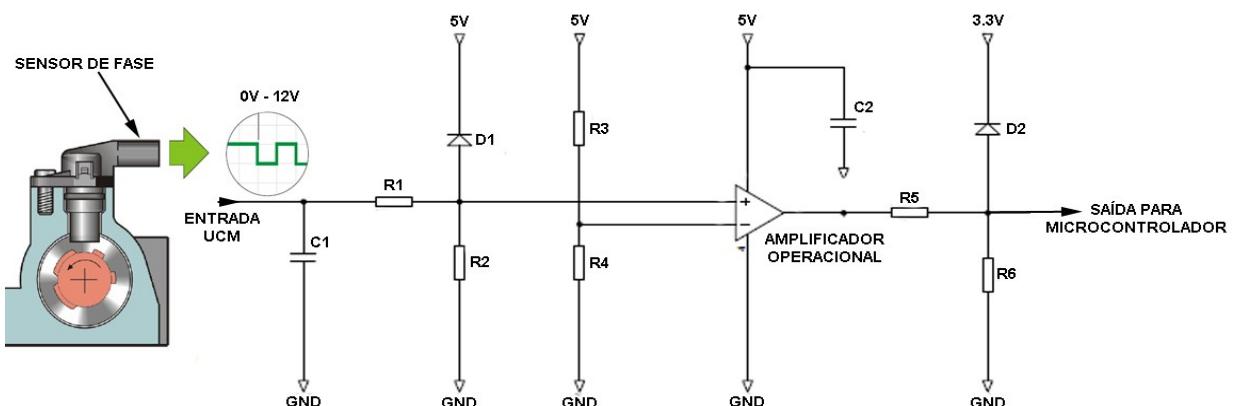
Fonte: o autor

O condicionamento do sinal senoidal proveniente do sensor de rotação do motor é efetuado pelo LM1815, um circuito integrado dedicado ao condicionamento de sensores de rotação. Trata-se de um amplificador adaptativo para sensores de relutância variável, que atua disparando o sinal em sua saída para um nível de tensão igual a sua tensão de VDD, que neste caso é de 3.3V, quando o sinal senoidal de entrada do sensor efetua o cruzamento com o sinal referência (GND), passando do ciclo positivo para o ciclo negativo. O tempo que o sinal de saída permanece igual a VDD é determinado pela constante de tempo formada entre o resistor R4 e o capacitor C3 que, nesta unidade, está definida como 100 μ s. Na entrada do circuito integrado, o resistor R1 possui a função de limitar a corrente do sinal do sensor de rotação (NATIONAL SEMICONDUCTOR, 2005).

3.2.3.3 Sensor de fase

O sensor de fase nos motores da família EA 113 2.0L são do tipo Hall. O sinal gerado é digital e possui uma amplitude de 12V e, deste modo, o objetivo do circuito é apenas reduzir esta amplitude para 3.3V sem que o sinal perca suas propriedades temporais. O circuito de condicionamento é semelhante ao circuito do sinal de ignição, conforme ilustrado na figura 31.

Figura 31 - Condicionamento do sinal do sensor de fase



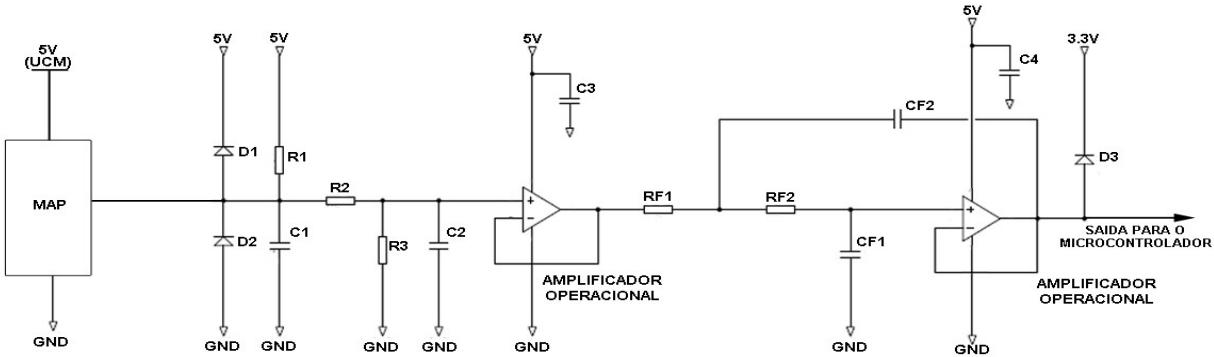
Fonte: o autor

Os componentes desse circuito possuem exatamente a mesma função descrita para o condicionamento do sinal de ignição. O sinal do sensor de fase é muito importante para sistemas que utilizam injeção sequencial de combustível onde a válvula injetora é aberta no exato instante em que a válvula de admissão se abre. A UCM utiliza este sinal para determinar qual dos cilindros gêmeos está no ciclo de admissão e qual está no ciclo de combustão.

3.2.3.4 Sensor de pressão absoluta no coletor de admissão (MAP)

O sensor de pressão no coletor de admissão disponibiliza, neste caso, um sinal analógico em sua saída referente à pressão no coletor. Este sinal possui uma faixa de operação de até 5V. A figura 32 ilustra o circuito de condicionamento elaborado para o sensor MAP.

Figura 32 - Condicionamento do sinal do sensor de pressão absoluta no coletor de admissão.



Fonte: o autor

O sinal do sensor entra no terminal positivo do primeiro amplificador operacional, configurado como seguidor de tensão (*buffer*), que tem a função de casar a impedância do sensor do resto do circuito da UCM, após ser atenuado pelo divisor de tensão, formado pelos resistores R2 e R3. O segundo amplificador operacional possui a função de filtrar ruídos presentes no sinal do sensor. O amplificador está configurado como filtro ativo de segunda ordem na topologia *sallen key* passa baixas. A frequência de corte (F_c) deste circuito é dada pela equação 28.

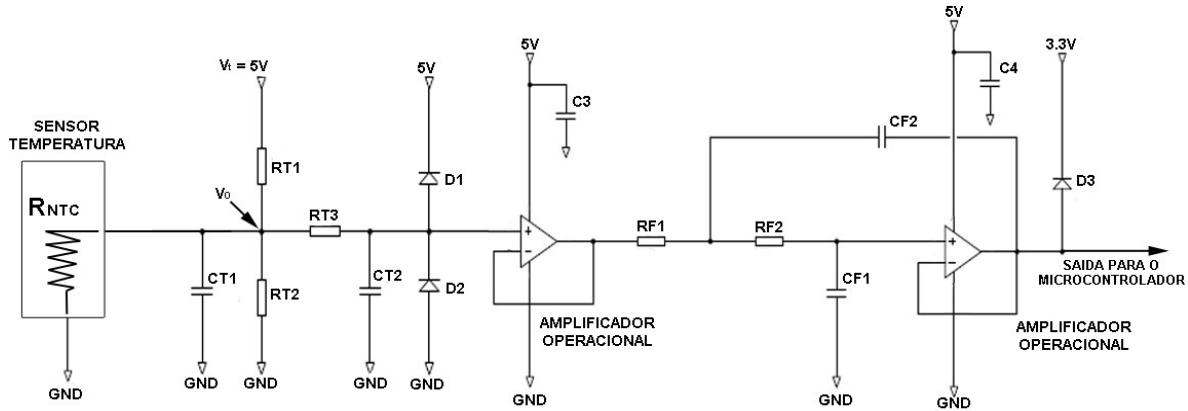
$$F_c = \frac{1}{2\pi\sqrt{RF1.RF2.CF1.CF2}} \quad \text{Eq.28}$$

O amplificador operacional configurado na topologia *sallen key* deve possuir uma banda passante dez vezes maior que a frequência de corte para permanecer com ganho unitário. Neste caso, a frequência de corte foi definida em 20KHz (PERTENCE JUNIOR, 2003).

3.2.3.5 Sensores de temperatura

Conforme descrito anteriormente, os sensores de temperatura, tanto do líquido de arrefecimento quanto do ar de admissão, variam a resistência inversamente proporcional à temperatura. Como as curvas fornecidas pelos dois sensores em seus respectivos catálogos técnicos são idênticas, o circuito de condicionamento de ambos também foi desenvolvido de forma idêntica, de acordo com a figura 33.

Figura 33 - Condicionamento do sinal dos sensores de temperatura NTC.



Fonte: o autor

Para condicionar o sinal de entrada dos sensores de temperatura, que na figura 33 é caracterizado por V_o , os limites de operação do microcontrolador (0V - 3.3V) e as resistências mínima (10Ω) e máxima (6100Ω) do sensor NTC foram considerados para equacionar a tensão de entrada no terminal positivo do amplificador operacional. A tensão de entrada, em função das resistências do circuito, foi calculada utilizando o método de análise de circuitos elétricos conforme leis de Kirchhoff (HALLIDAY, 2012). A equação 29 ilustra e equação da tensão de entrada do circuito elaborado.

$$V_o = V_t - \left[\frac{V_t}{\left(\frac{RT_2 \cdot R_{NTC}}{RT_2 + R_{NTC}} \right) + RT_1} \cdot RT_1 \right] \quad \text{Eq.29}$$

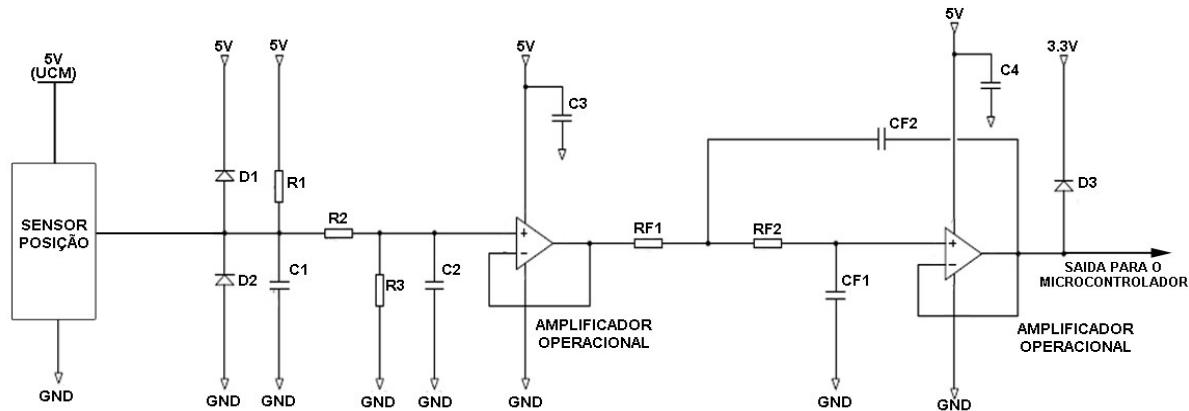
Nesse caso, V_t é a tensão positiva do circuito (5V). O primeiro amplificador operacional, configurado como seguidor de tensão (*buffer*), tem a função de casar a impedância do sensor do resto do circuito da UCM, o segundo amplificador operacional possui a função de filtrar ruídos presentes no sinal do sensor e possui uma frequência de corte de 20KHz, conforme equação 28. A não linearidade característica dos sensores de temperatura deve ser tratada por software para a correta aplicação no gerenciamento do motor.

3.2.3.6 Sensores de posição da válvula borboleta e do pedal do acelerador

Os sensores de posição da válvula borboleta e do pedal do acelerador são potenciômetros. Esses sensores disponibilizam um sinal analógico em sua saída

referente às posições da válvula borboleta e do pedal do acelerador. Este sinal possui uma faixa de operação de até 5V. O circuito de condicionamento dos sensores é ilustrado na figura 34.

Figura 34 - Condicionamento do sinal dos sensores de posição da válvula borboleta e do pedal do acelerador.



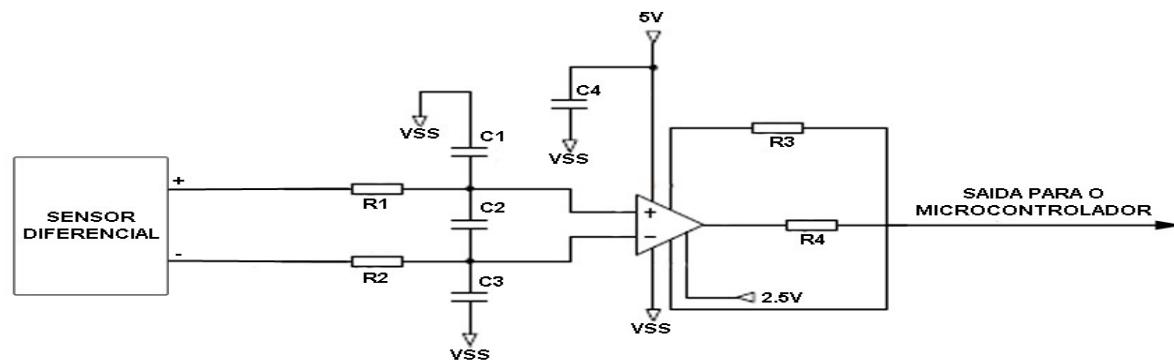
Fonte: o autor

O circuito de condicionamento destes sensores segue o mesmo padrão apresentado para o circuito condicionamento do sinal do sensor MAP.

3.2.3.7 Sensores de oxigênio e de detonação

Os sensores de oxigênio e de detonação possuem funções completamente diferentes no gerenciamento do motor, conforme descrito no capítulo 2. Entretanto, ambos apresentam padrões elétricos semelhantes e, por esse motivo, os circuitos de condicionamento são similares, conforme ilustrado na figura 35.

Figura 35 - Condicionamento do sinal dos sensores de oxigênio e de detonação.



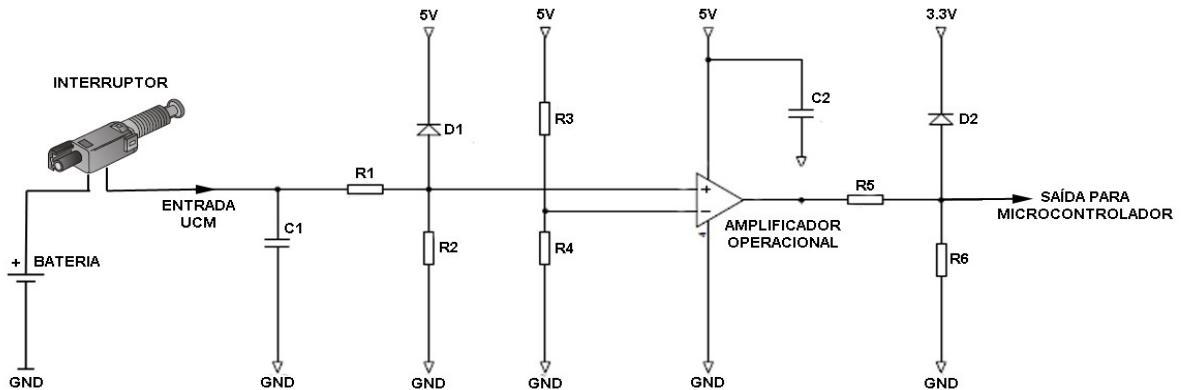
Fonte: o autor

Para efetuar o condicionamento dos sinais elétricos enviados pelos sensores de oxigênio e de detonação, é necessário utilizar um amplificador operacional de instrumentação com ganho ajustável. O ganho do circuito é definido pelos resistores R3 e R4.

3.2.3.8 Interruptores de embreagem e do pedal de freio

A unidade de controle do motor precisa identificar quais ações o condutor está executando, para tomar ações que suavizem o funcionamento do motor, aumentando seu rendimento e economia. Esses interruptores são integrados aos seus respectivos pedais e operam de forma semelhante ao comutador de ignição, enviando um sinal elétrico de aproximadamente 12V quando o condutor pressiona o pedal. O circuito de condicionamento dos interruptores é ilustrado na figura 36.

Figura 36 - Condicionamento do sinal dos interruptores de embreagem e pedal de freio.



Fonte: o autor

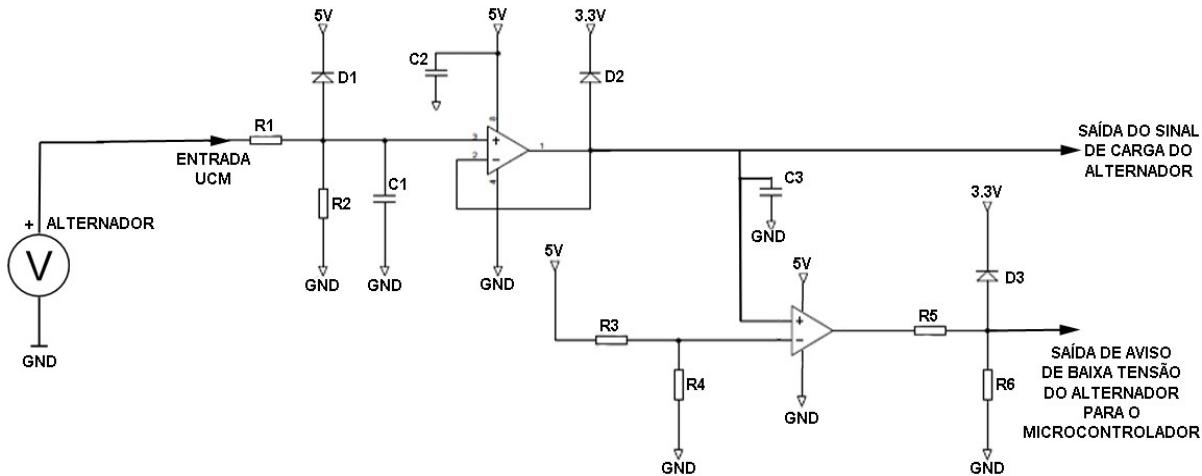
Os circuitos de condicionamento dos interruptores de embreagem e do pedal de freio seguem o mesmo padrão apresentado para o circuito condicionamento do sinal de ignição.

3.2.3.9 Sinal de carga do alternador

O sinal de carga do alternador é constantemente monitorado pela UCM. É de extrema importância para o microcontrolador determinar exatamente qual a tensão fornecida pelo alternador, pois influencia diretamente nos cálculos do tempo de carga da bobina de ignição e do tempo de injeção de combustível. Caso o valor

fornecido pelo alternador seja abaixo do necessário, os valores de carga da bobina e de injeção devem ser compensados pelo sistema de gerenciamento, para que se garanta energia suficiente para produzir uma centelha de qualidade e a injeção correta da massa de combustível específica para a condição do motor. Desta maneira, o sinal do alternador, que é de aproximadamente 14V, deve ser condicionado e atenuado para valores de no máximo 3.3V, se adequando ao microcontrolador. O circuito de condicionamento do sinal de carga do alternador é apresentado na figura 37.

Figura 37 - Condicionamento do sinal de carga do alternador.



Fonte: o autor

O circuito de condicionamento do sinal de carga do alternador possui um divisor resistivo na entrada, que atenua a tensão de entrada máxima do alternador (15V) para 3.3V. A tensão de saída do amplificador operacional é diretamente conectada em uma entrada analógica do microcontrolador. O segundo amplificador operacional compara a tensão na entrada analógica do microcontrolador com uma tensão de referência ajustada pelos resistores R3 e R4. Neste projeto, a tensão de referência foi calculada para alterar o sinal de saída do segundo amplificador operacional quando a tensão fornecida pelo alternador for menor que 12.6V.

3.2.4 Condicionamento eletrônico dos atuadores

Conforme citado anteriormente, os atuadores automotivos recebem sinais elétricos de comando da unidade de controle do motor. Para que a UCM possa atuar no motor, estes sinais de comando devem receber um condicionamento prévio para

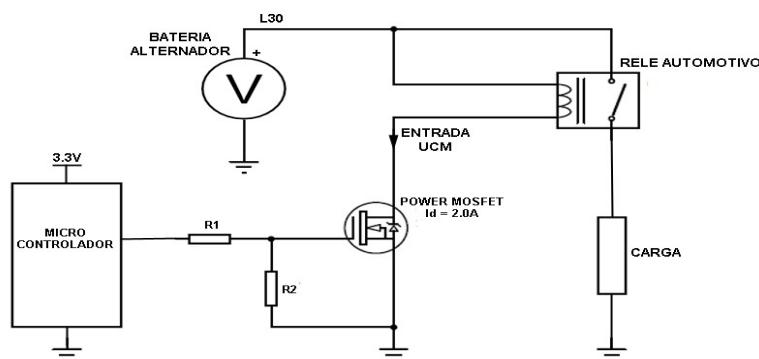
adequar o sinal para seu respectivo atuador. O microcontrolador possui a capacidade de fornecer no máximo 20mA de corrente em seus sinais de saída (FREESCALE, 2010). Os atuadores do motor necessitam de correntes da ordem de ampères para operar. Deste modo, o *hardware* elaborado neste projeto deverá conter circuitos eletrônicos que possam condicionar o sinal de comando, para que os atuadores operem de forma satisfatória. Para esta UCM, foi determinado que os seguintes atuadores devem conter condicionamento:

- Possibilidade de acionar até quatro relés.
- Possibilidade de acionar o motor da válvula borboleta.
- Possibilidade de acionar oito válvulas injetoras.
- Possibilidade de acionar oito bobinas de ignição com *driver* interno
- Possibilidade de acionar oito bobinas de ignição com *driver* externo
- Possibilidade de acionar quatro atuadores de uso geral de até 2A.

3.2.4.1 Acionamento de relés

Desde o inicio da evolução na arquitetura elétrica dos veículos, os relés automotivos sempre foram utilizados para acionar os itens elétricos do veículo como, por exemplo, as bombas de combustível e de ignição. Este projeto prevê o acionamento por parte da UCM de até quatro relés automotivos, conforme circuito ilustrado na figura 38.

Figura 38 - Condicionamento do sinal de acionamento dos reles automotivos.



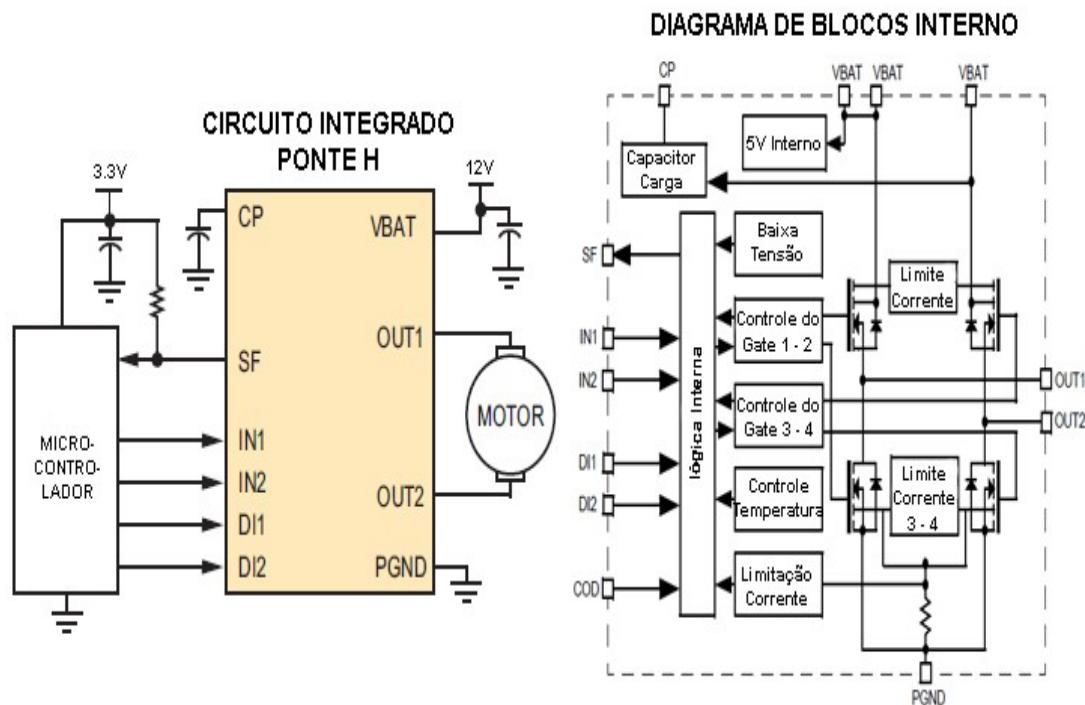
Fonte: o autor

Geralmente, os relés automotivos limitam a corrente de acionamento da bobina primária internamente. A resistência encontrada em sua bobina primária varia entre 50Ω e 100Ω , dependendo da aplicação. Na UCM desenvolvida, deve ser possível acionar relés que consumam no máximo 2A de corrente na bobina primária.

3.2.4.2 Acionamento do motor da válvula borboleta

A posição angular da válvula borboleta é modificada através de um motor elétrico em seu eixo de rotação. Este motor DC é acionado por uma tensão modulada do tipo PWM. O microcontrolador controla a variação no sinal PWM por software, mas a corrente máxima fornecida pelo microcontrolador não é suficiente para acionar o motor DC da válvula borboleta. Desta maneira, o circuito de condicionamento deve utilizar um circuito integrado que utilize uma lógica de ponte H para acionar o motor DC. O circuito de condicionamento utilizado na UCM é ilustrado na figura 39.

Figura 39 - Condicionamento do sinal de acionamento do motor da válvula borboleta.



Fonte: (Adaptado de FREESCALE, 2011).

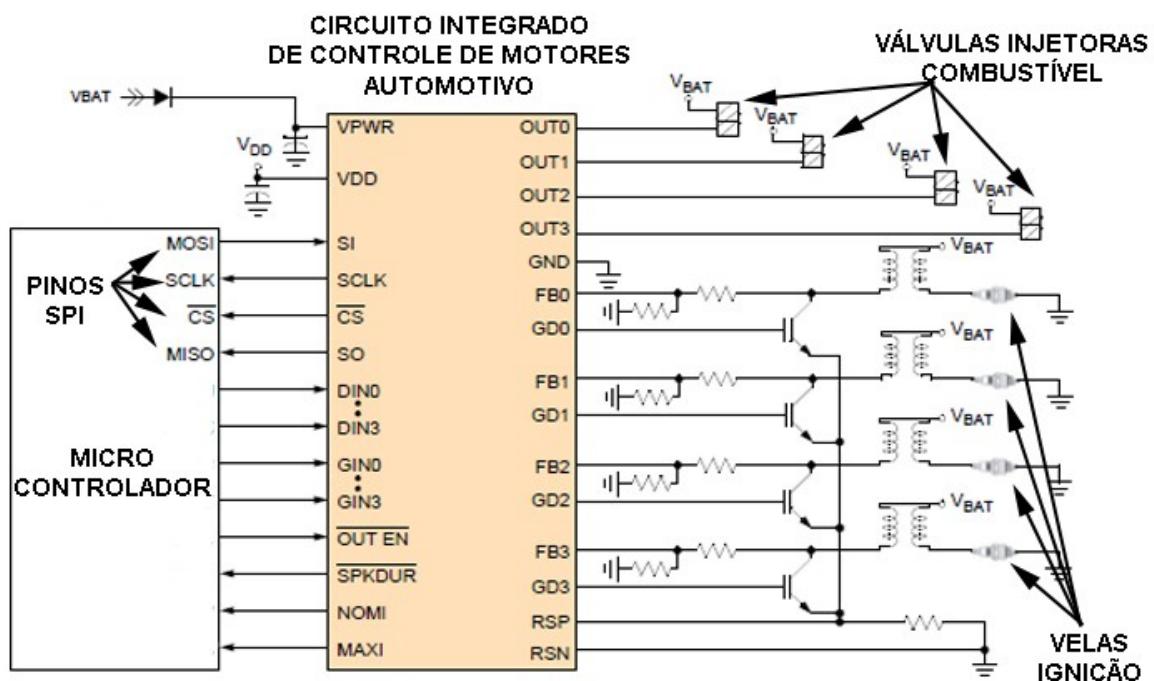
O circuito de condicionamento é baseado em um circuito integrado desenvolvido especificamente para efetuar o controle da válvula borboleta, utilizando

uma lógica baseada no acionamento por MOSFETS de potência, com configuração em uma topologia de Ponte H. O circuito integrado possui gerenciamento da corrente que circula no motor DC da válvula borboleta e é acionado diretamente pelo microcontrolador através nos pinos IN1 e IN2.

3.2.4.3 Acionamento das válvulas injetoras de combustível e das bobinas de ignição com driver interno

A UCM desenvolvida neste projeto pode controlar até oito válvulas injetoras de combustível e oito bobinas de ignição. Este controle é efetuado por um circuito integrado que foi especialmente desenvolvido pela *Freescale®* para efetuar o seu controle. O circuito de condicionamento utilizado para acionar quatro injetores e quatro bobinas é ilustrado na figura 40.

Figura 40 - Condicionamento do sinal de acionamento das válvulas injetoras e das bobinas de ignição.



Fonte: (Adaptado de FREESCALE, 2009).

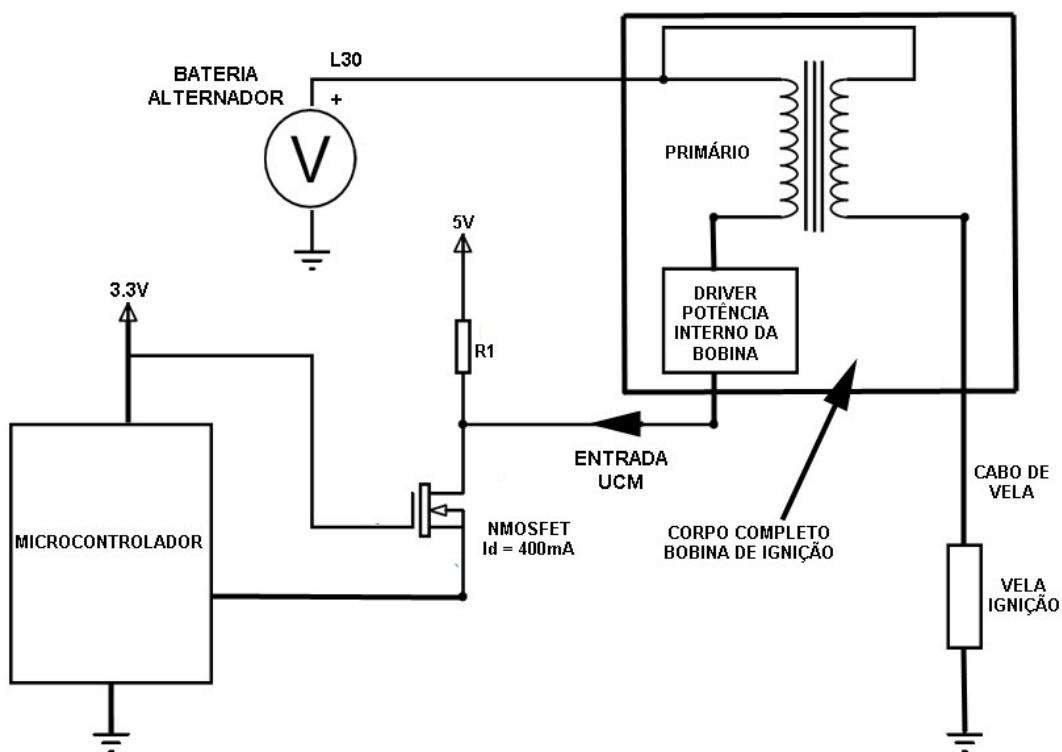
O circuito integrado utilizado para o condicionamento das válvulas injetoras de combustível e da bobina de ignição foi especialmente desenvolvido para esta função. O mesmo pode se comunicar com o microcontrolador através do protocolo de comunicação SPI (*Serial Peripheral Interface*). A comunicação pode transmitir

dados, como tempo de carga da bobina, corrente máxima de acionamento, dentre outros parâmetros.

3.2.4.4 Acionamento das bobinas de ignição com driver externo

As bobinas de ignição modernas, inclusive a utilizada no Volkswagen Polo Sedan deste projeto, não necessitam de um condicionamento de potência interno na UCM. Nesses casos, o condicionamento é efetuado internamente na bobina, sendo que a UCM apenas envia o pulso de controle para a bobina de ignição. O circuito de condicionamento está ilustrado na figura 41.

Figura 41 - Condicionamento do sinal de acionamento da bobina de ignição com driver externo



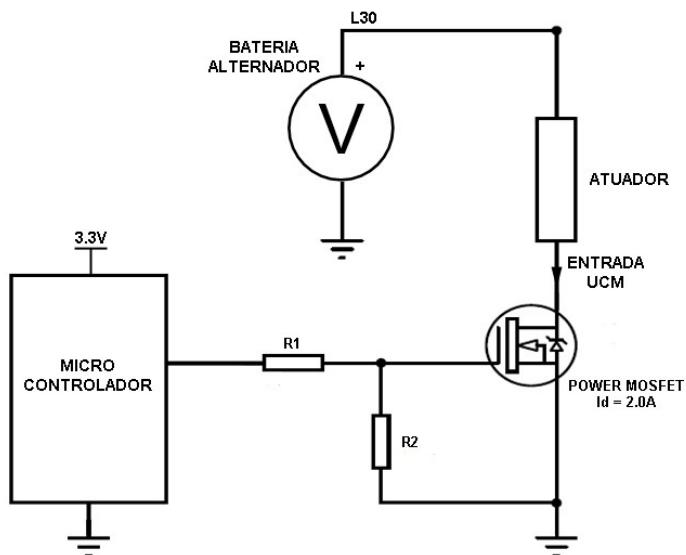
Fonte: o autor

A corrente fornecida pela UCM, na maioria dos casos, é insuficiente para que a bobina produza uma centelha na vela de ignição. Em alguns casos, o próprio microcontrolador é capaz de fornecer esta corrente sem a necessidade de um circuito de potência. Na UCM desenvolvida neste projeto, a corrente de acionamento da bobina de ignição é determinada pela relação entre a tensão de 5V e o valor do resistor R1.

3.2.4.5 Acionamento de atuadores de uso geral

Os veículos modernos possuem atuadores de uso geral. É possível citar, como exemplo, o acionamento da resistência aquecedora do sensor de oxigênio, a válvula injetora de combustível para auxílio da partida quando o motor está frio e o acionamento da eletroválvula de purga do *canister*. Neste projeto, a UCM é capaz de acionar até quatro atuadores de uso geral com correntes de até 2A. A figura 42 ilustra o circuito de condicionamento desses atuadores.

Figura 42 - Condicionamento do sinal de acionamento dos atuadores de uso geral.



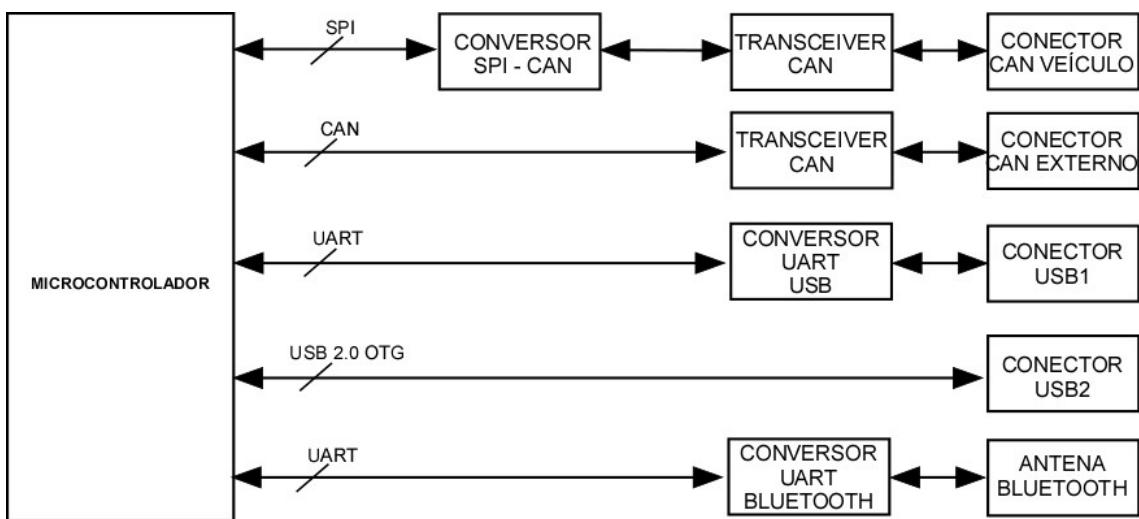
Fonte: o autor

No circuito de condicionamento, o microcontrolador fornece uma tensão no *gate* do *MOSFET*, que tem capacidade de suportar corrente de até 2A.

3.2.4.6 Transmissores de comunicação externa

O *hardware* pode se comunicar em duas redes USB e duas redes CAN, sendo uma rede externa para aplicações gerais e outra rede disponível no conector automotivo, que possibilita integração do *hardware* com a rede CAN do veículo. O *hardware* proposto também é capaz de se comunicar em uma rede sem fio, utilizando o protocolo *Bluetooth*. O diagrama de blocos de comunicação externa é ilustrado na figura 43.

Figura 43 - Diagrama de comunicação da UCM.



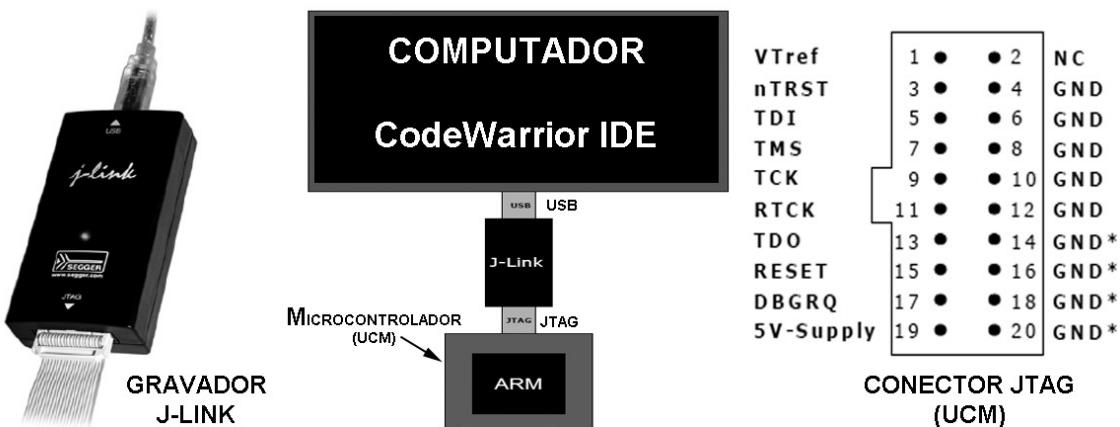
Fonte: o autor

Como o microcontrolador possui periféricos de comunicação CAN e USB, foram disponibilizados conectores externos que possibilitam comunicação direta com o microcontrolador. As demais redes USB e CAN se comunicam indiretamente com o microcontrolador, pois antes passam por um circuito integrado conversor. No caso da rede de comunicação CAN, o circuito converte os dados enviados via SPI pelo microcontrolador, enquanto que no caso da comunicação USB, o circuito integrado converte os dados via UART pelo microcontrolador. A comunicação *Bluetooth* também é efetuada de forma indireta pelo microcontrolador. Os dados enviados pela antena de *Bluetooth* devem ser convertidos por um circuito integrado que recebe do microcontrolador comando e dados via protocolo UART.

3.2.5 Circuito de gravação JTAG

O microcontrolador utilizado neste projeto possui um núcleo ARM CORTEX-M4 para que o *firmware* elaborado seja gravado na sua memória *flash*. Desta forma, o microcontrolador necessita de um circuito externo com a função de executar a interface de gravação com computador utilizado para elaborar o *firmware*. O circuito utilizado para realizar esta função é ilustrado na figura 44.

Figura 44 - Circuito de gravação do microcontrolador padrão JTAG.



Fonte: o autor

O ambiente de desenvolvimento integrado para a elaboração e compilação do *firmware* de controle do microcontrolador escolhido para este projeto é o *CodeWarrior®* produzido pela *Freescale®*. O mesmo possui integração com o gravador produzido pela *Segger®*, denominado J-LINK. Este gravador é capaz de efetuar a gravação na memória *flash* e RAM do microcontrolador, além de ser capaz de efetuar *debug* em tempo real do *firmware*. O gravador J-LINK opera utilizando o padrão JTAG e o *hardware*, assim como o conector de gravação, seguem o padrão JTAG.

3.3 DESCRIÇÃO DO FIRMWARE DA UCM

Desde o início dos anos 90, a aplicação de sistemas eletrônicos de gerenciamento e controle do motor ciclo Otto tornou-se o principal método para melhorar a dinâmica, a economia de combustível e as emissões de poluentes. Um sistema de gerenciamento e controle eletrônico do motor Otto geralmente consiste de *hardware* e *firmware*. Após o correto condicionamento dos sinais elétricos dos sensores, o microcontrolador processa estes sinais para determinar os parâmetros de operação para o motor.

Nesta subseção, um novo *firmware* de controle eletrônico do motor é desenvolvido, baseado em sistemas de controle em malha fechada e em eventos externos de sincronismo do motor. Este *firmware* é projetado com uma concepção modular, utilizando um método hierárquico. O programa executa as funções básicas de aquisição de sinais dos sensores, o cálculo de injeção de combustível e avanço do ponto ignição, além de controlar a atuação da válvula borboleta eletrônica. Esta

subseção descreve os conteúdos relacionados ao desenvolvimento do *firmware* e a lógica de controle baseado em novas funções e características do microcontrolador de 32-bit. Os princípios de gerenciamento utilizados neste trabalho são extensíveis a todos os veículos equipados com motores de combustão interna ciclo Otto.

3.3.1 Descrição geral do *firmware* de controle do motor

Um *firmware* eficaz de controle eletrônico implementado em um microcontrolador deve, além de executar as tarefas de adquirir os sinais elétricos dos sensores através da conversão A/D, calcular os parâmetros e controlar os atuadores, ainda possuir as características de resposta em tempo real, estabilidade e portabilidade. Para atingir esses objetivos, a metodologia de desenvolvimento é um dos fatores mais importantes. Este trabalho utilizou o *software* CodeWarrior® IDE da Freescale® para desenvolver toda a lógica de programação utilizando a linguagem C de forma hierárquica para se projetar o *firmware* a ser gravado no microcontrolador.

Todas as funções inicialmente projetadas foram testadas e validadas, com o objetivo de avaliar se a unidade de controle do motor realmente atende às suas especificações iniciais. Além disso, todos os sensores e atuadores especificados no projeto foram testados. Para realizar os testes dos circuitos projetados, foram desenvolvidos programas de testes com a função de controlar, através de pulsos de acionamento gerados pelo microcontrolador, os atuadores ou adquirir e processar os sinais emitidos pelos sensores. Com o resultado satisfatório em todos os testes de *hardware* e funções do *firmware*, iniciou-se o planejamento da arquitetura principal do programa principal "main.c".

O programa principal é dividido em três camadas: a camada principal possui função de lidar com a inicialização do programa, dos pinos e dos periféricos do microcontrolador e dar suporte às outras duas camadas existentes. A camada de sincronismo possui a função de gerenciar as interrupções de *software* e garantir o sincronismo dos atuadores com o motor em tempo real. Finalmente, a camada do sistema operacional, denominada camada RTOS, possui a função de gerenciar todas as tarefas do *firmware* em função do tempo de execução de cada tarefa.

Figura 45 - Descrição geral do programa principal (main.c)



Fonte: o autor

Para criar um ambiente operacional adequado para o programa, as tarefas da camada principal devem ser executadas no início da aplicação. Essas tarefas efetuam as configurações iniciais de todos os registradores disponíveis para o microcontrolador. As tarefas da camada principal executam as configurações extremamente importantes de toda a aplicação, incluindo a alocação de memória para os mapas de ignição e sincronismo, as configurações da frequência de operação da CPU do microcontrolador (100MHz), dos pinos de entrada e saída do microcontrolador, das interrupções de *software* para camada de sincronismo e de todos os periféricos do microcontrolador para que seja utilizado pelo sistema operacional na camada RTOS.

A camada principal, após executar todas suas tarefas, suporta as duas outras camadas existentes. Isto significa que as camadas de sincronismo e RTOS rodam dentro da camada principal. Após efetuar todas as configurações iniciais do microcontrolador, a camada principal habilita a interrupção do pino interligado ao sinal da linha 15 (positivo pós-chave) para dar inicio à aplicação da camada de sincronismo e entrega o controle da aplicação ao sistema operacional, iniciando a operação da camada RTOS. As camadas de sincronismo e RTOS rodam em paralelo. Apesar de ser um microcontrolador com apenas um núcleo de

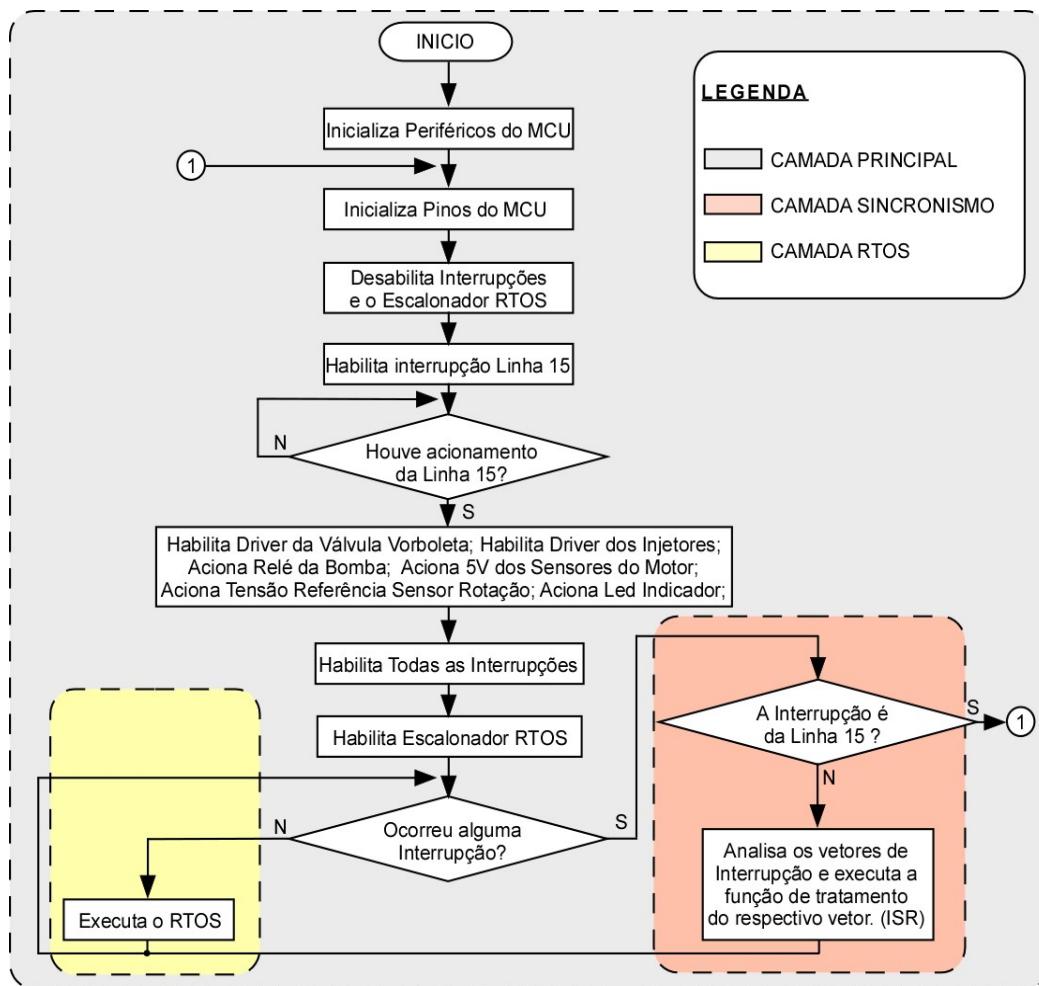
processamento, um método de hierarquia foi implementado, permitindo que as duas camadas executem suas tarefas simultaneamente.

3.3.2 Estratégia geral do *firmware* de controle do motor

Visando uma melhoria na eficiência de toda a aplicação do *firmware* e levando em consideração toda a dinâmica dos motores Otto, todas as funções elaboradas receberam uma camada de destino e um número de prioridade na execução. Naturalmente, as funções elaboradas para serem executadas na camada de sincronismo possuem uma prioridade maior do que aquelas elaboradas para serem executadas na camada RTOS. Para gerenciar eletronicamente um motor Otto, o *firmware* de controle deve priorizar os comandos diretamente relacionados com o sincronismo do motor, garantindo que o motor terá, por exemplo, a centelha de ignição produzida com a energia necessária, no exato instante e com o exato avanço do ponto de ignição para produzir uma combustão satisfatória. Para garantir este correto sincronismo do *firmware* com a dinâmica do motor de combustão interna, todas as funções da camada sincronismo ocorrem em função de eventos externos (interrupções externas de *software*). Como as funções da camada de sincronismo ocorrem em função de eventos externos, é possível afirmar que estas possuem maior prioridade sobre as funções da camada RTOS.

Quando não ocorrer nenhuma interrupção externa, a aplicação entrega o controle do *firmware* ao sistema operacional de tempo real RTOS. Esta camada RTOS possui autonomia para gerenciar suas tarefas em função do tempo de cada aplicação. Se a camada de sincronismo requisitar a execução de outra função, a camada RTOS interrompe a execução da tarefa atual e, após executar a função da camada sincronismo, retorna exatamente na linha de comando seguinte à solicitação de pausa do RTOS. A figura 46 ilustra de forma geral a estratégia do *software* na camada principal com destaque para as camadas de sincronismo e RTOS.

Figura 46 - Fluxograma do programa principal (main.c)



Fonte: o autor

O fluxograma da figura 46 mostra, de forma generalizada, a estrutura e a estratégia geral de toda a aplicação. Após energizar a UCM desenvolvida, o microcontrolador, ao sair de seu vetor de *reset*, inicia as configurações iniciais dos registradores principais para determinar a frequência de operação, habilitar *Timers* e periféricos utilizados como, por exemplo, conversor ADC, módulo PWM, *FlexRam*, UART, USB, CAN, dentre outros. Após configurar os periféricos, o microcontrolador inicia o processo de configuração de seus pinos de entrada e saída para se adequar ao *hardware* desenvolvido. O esquema elétrico completo do *hardware* desenvolvido está disponível nos apêndices desta dissertação.

Com os pinos configurados e com os estados iniciais de cada pino também determinados, a camada principal garante que todas as interrupções estão desabilitadas e que o escalonador da camada RTOS também está desabilitado para que se evite execução inapropriada das outras duas camadas da aplicação. Neste

instante, a camada principal entra em um *loop* aguardando com que o sinal da linha 15 (positivo pós-chave) seja acionado, indicando a intenção de partida do motor pelo condutor. Ao se acionar a linha 15, ocorre a primeira interrupção externa da aplicação. Se o sinal da linha 15 foi ligado à camada principal, prepara o motor para dar a partida, acionando o relé da bomba de combustível, acionando a tensão positiva de 5V de alimentação de todos os sensores do motor, acionando a tensão 2,5V de referência para o sensor de rotação e habilitando os circuitos integrados responsáveis por acionar a válvula borboleta e os injetores de combustível.

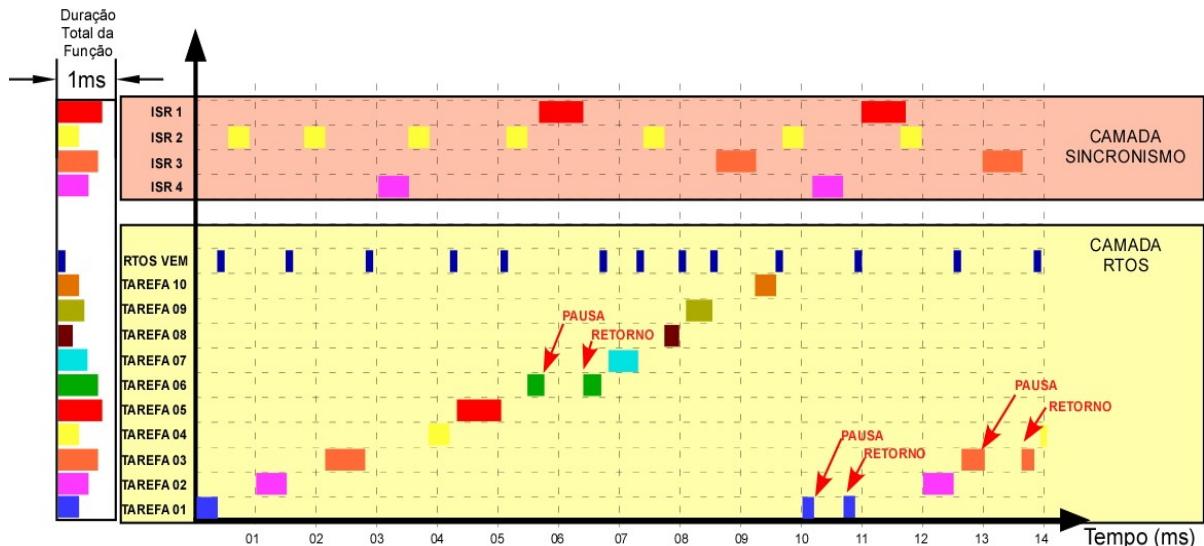
Após preparar o motor para dar a partida, a camada principal habilita todas as interrupções possíveis na aplicação para iniciar a operação da camada de sincronismo e habilita o escalonador do RTOS para dar início à operação da camada RTOS. As camadas de sincronismo e RTOS somente irão parar suas execuções quando a linha 15 for desligada, fazendo com que a camada principal retorne aos valores iniciais de todos os pinos e reiniciando a aplicação para aguardar novo acionamento da linha 15. Para compreender melhor a execução das camadas de sincronismo e RTOS, as próximas subseções abordam a estratégia destas camadas, exemplificando o conceito utilizado.

3.3.3 Conceito da camada RTOS

Sistemas operacionais de tempo real ou RTOS (*Real Time Operating Systems*) são voltados para aplicações onde é essencial a confiabilidade e a execução de tarefas em prazos compatíveis com a ocorrência de eventos externos. Tempo real é um termo que pode ter significados muito diferentes, dependendo da velocidade da ocorrência dos eventos de um sistema e dos tempos de resposta necessários para atendê-los. Há sistemas que podem levar muito tempo para modificar o seu estado. Em geral motores de combustão interna ciclo Otto dependem do número de cilindros existentes no motor, da carga imposta ao motor, da rotação atual do motor e de fatores físicos da construção do motor como, por exemplo, a capacidade volumétrica e a taxa de compressão para alterar sua resposta. Em testes de bancada, foi possível concluir que, para esta aplicação, o motor demora aproximadamente 100ms em média para alterar seu estado atual. Deste modo, foi determinado que um ciclo completo do escalonador do RTOS deveria durar 10% do tempo de resposta do motor, ou seja, 10ms.

Em sistemas operacionais, as rotinas de processamento, em geral, são bastante especializadas e curtas, pois devem executar a sua tarefa no menor tempo possível. Há um forte paralelismo na execução das atividades e um estabelecimento hierárquico de prioridades, onde as atividades mais prioritárias interrompem aquelas menos prioritárias e assumem o controle do processador. A divisão de tarefas em rotinas especializadas requer que haja alguma forma de sincronismo ou troca de informação entre elas, por exemplo, para que seja informada para outra rotina qual tarefa já foi concluída ou não. Nesta aplicação, um vetor de estados do motor, denominado RTOS_VEM, foi desenvolvido, e executa a troca de informações entre as tarefas da aplicação. A figura 47 mostra o diagrama de tempo da aplicação, ilustrando como o RTOS se comporta para gerenciar a prioridade entre as tarefas da camada RTOS e as funções da camada sincronismo (ISR) ocorridas em função de eventos externos.

Figura 47 - Diagrama de tempos das funções do programa principal (main.c)



Fonte: o autor

O tempo máximo para execução de cada tarefa criada para ser executada na camada do RTOS é de 1ms. Conforme mencionado anteriormente, ficou determinado que um ciclo completo do escalonador do RTOS deve durar 10ms. Este método de construção do *firmware* permite criar até 10 tarefas com tempo limite de 1ms. O escalonador do RTOS se encarrega de gerenciar a latência de cada tarefa, de acordo com sua prioridade e execução. Ao final de cada tarefa, o vetor de estados do motor (RTOS_VEM) é atualizado com novos parâmetros. Isso permite

que as outras tarefas utilizem os parâmetros atualizados quando necessitarem. A figura 47 mostra como a tarefa é pausada quando ocorre uma interrupção externa. Ao final da execução da rotina da interrupção, o sistema retorna exatamente onde tinha parado a execução da tarefa. Em casos onde ocorram interrupções simultâneas, ambas serão executadas em sua totalidade antes que o RTOS retorne com suas operações. Se o escalonador RTOS concluir que está ocorrendo frequentemente latências superiores a 100ms, será enviada uma mensagem de aviso ao sistema de diagnóstico (HMI). A tabela 3 mostra as funções que foram desenvolvidas para essa aplicação.

Tabela 3 - Tarefas implementadas para camada RTOS

Tarefa	Nome	Descrição
00	AtualizaRTOS	Atualiza Parâmetros RTOS
01	ReferenciaRPM	Determina referencia rotação (Desejo motorista)
02	CalculoAvancoIGN	Calcula Avanço de ignição
03	CalculoTempoInjecao	Calcula Tempo de injeção de combustível
04	PID_RPM	Controle PID que calcula a referencia correta VB
05	SincronismoSistemaADM	Controle PI para que a VB siga a referência
06	Não implementado	Nenhuma tarefa foi implementada
07	HMI	Envia dados via USB e Bluetooth para diagnose
08	EstadoAtualMotor	Determina o estado atual do motor
09	Não implementado	Nenhuma tarefa foi implementada

Fonte: o autor

Quanto menor for o número determinado para a respectiva tarefa, maior é a sua prioridade de execução pelo escalonador. O escalonador RTOS verifica sequencialmente quais tarefas foram executadas e quais ainda devem ser executadas. Ou seja, se em sua verificação for constatado que a tarefa zero está pendente junto com a tarefa três, a camada RTOS irá terminar a execução da tarefa zero para iniciar a execução da tarefa três. Nesta aplicação, não foram implementadas as tarefas seis e nove, sendo uma reserva técnica para futuras aplicações nesta área.

A implementação da camada RTOS aumentou significativamente a eficiência do *firmware* desenvolvido. Esse método garante que as tarefas e funções essenciais para que o motor se mantenha em regime de operação sejam executadas prioritariamente, respeitando a hierarquia e a dinâmica dos motores ciclo Otto. Este método apresenta um paralelismo bastante eficaz entre as camadas RTOS e de sincronismo, permitindo que ambas possam executar suas tarefas em paralelo, mesmo com apenas um núcleo de processamento disponível.

3.3.4 Conceito da camada de sincronismo

A camada de sincronismo possui rotinas que foram desenvolvidas para ocorrer em função de eventos externos. Conforme descrito anteriormente, esta camada está inserida dentro da camada principal, e as principais rotinas de tratamento das interrupções desenvolvidas foram as de tratamento da linha 15, do sensor de fase, do sensor de rotação e de disparo e controle dos *Timers* para acionamento das válvulas injetoras de combustível e bobina de ignição.

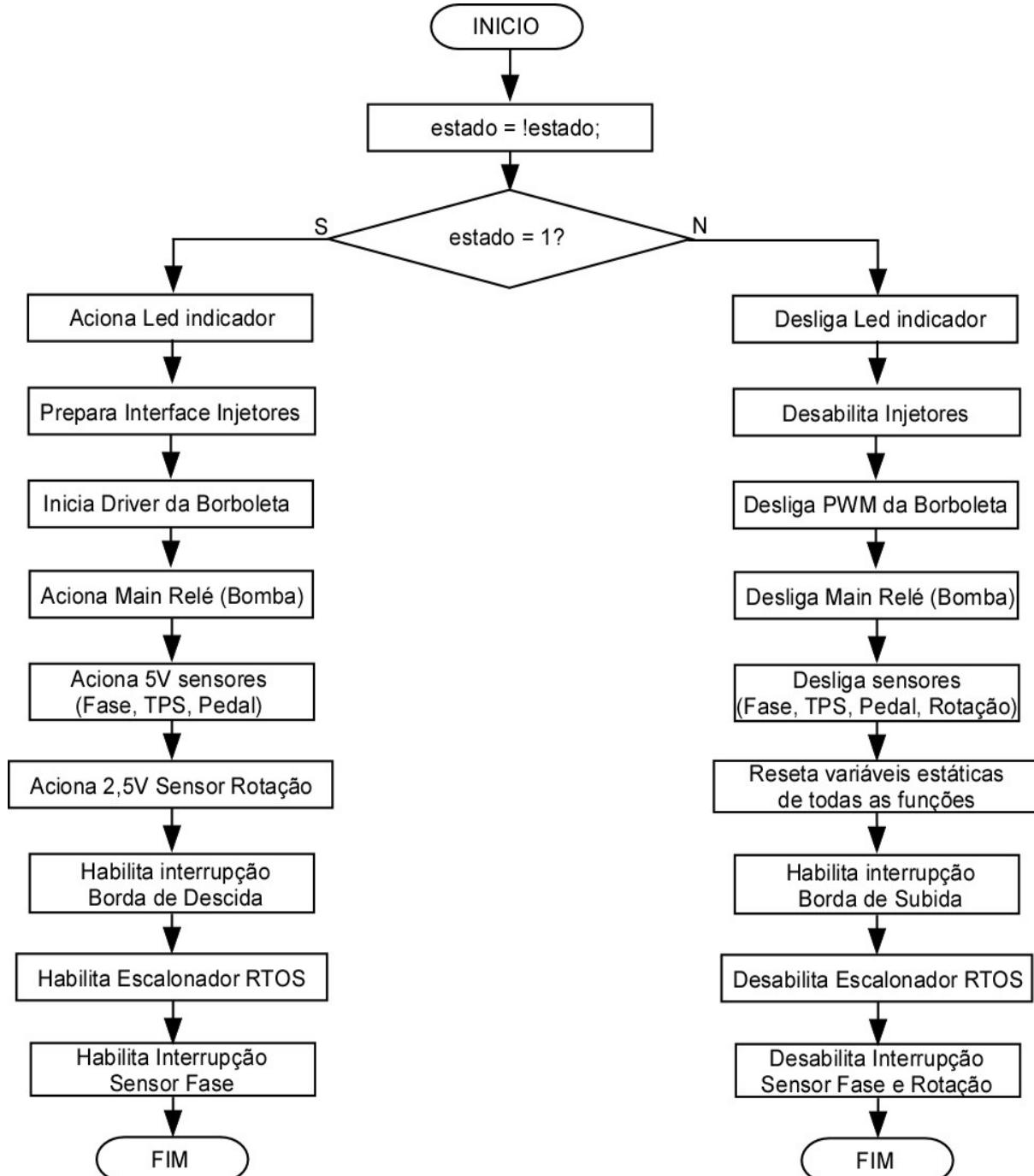
Essas funções possuem grande importância para toda a aplicação, pois são elas que garantem o correto sincronismo do *firmware* desenvolvido com o motor controlado. Por esse motivo, as rotinas de tratamento das interrupções externas possuem maior prioridade do que as tarefas da camada RTOS. Para esta aplicação, é mais importante garantir o correto acionamento das válvulas injetoras e das bobinas de ignição do que recalcular um novo tempo de injeção ou até mesmo transmitir algum dado para o sistema de diagnose.

A função que permite que a camada de sincronismo inicie suas funções é a rotina de tratamento do sinal da chave de ignição (linha 15). Após a camada principal efetuar todas as configurações iniciais do microcontrolador e desabilitar todas as interrupções possíveis, além de desabilitar o escalonador RTOS, a camada principal habilita somente a interrupção externa do sinal da linha 15, habilitando vetor de interrupção quando houver uma borda de subida do sinal do pino de entrada. Ou seja, quando o pino de entrada do sinal da linha 15, descrito na seção 3.2.3.1, alterar seu estado lógico de zero para um. Nesse instante, a variável estática “estado” irá inverter seu nível lógico e proceder conforme o lado esquerdo do fluxograma.

A figura 48 ilustra a lógica de controle da função de tratamento do sinal de ignição. Esta função prepara o motor para a partida, acionando a bomba de combustível e preparando os circuitos integrados responsáveis por efetuar a interface entre o microcontrolador e os atuadores. Entretanto, a função mais importante desta rotina de tratamento é a habilitação das interrupções dos sensores de fase e de rotação, para que a camada de sincronismo possa iniciar sua operação. Após habilitar as interrupções dos sensores, a rotina de tratamento do sinal de ignição altera sua borda de interrupção para borda de descida para que, quando o motorista desligar a chave de ignição, ocorra uma nova interrupção do vetor da linha

15 para que a função seja executada, conforme o lado direito do fluxograma, desligando o relé da bomba, efetuando *reset* de todas as funções executadas e preparando o *firmware* para iniciar nova interação.

Figura 48 - Fluxograma da rotina de tratamento da interrupção do sinal de ignição (Linha 15)



Fonte: o autor

Ao término da execução da rotina de tratamento do sinal de ignição, o programa inicia a execução da camada RTOS, enquanto que a camada de

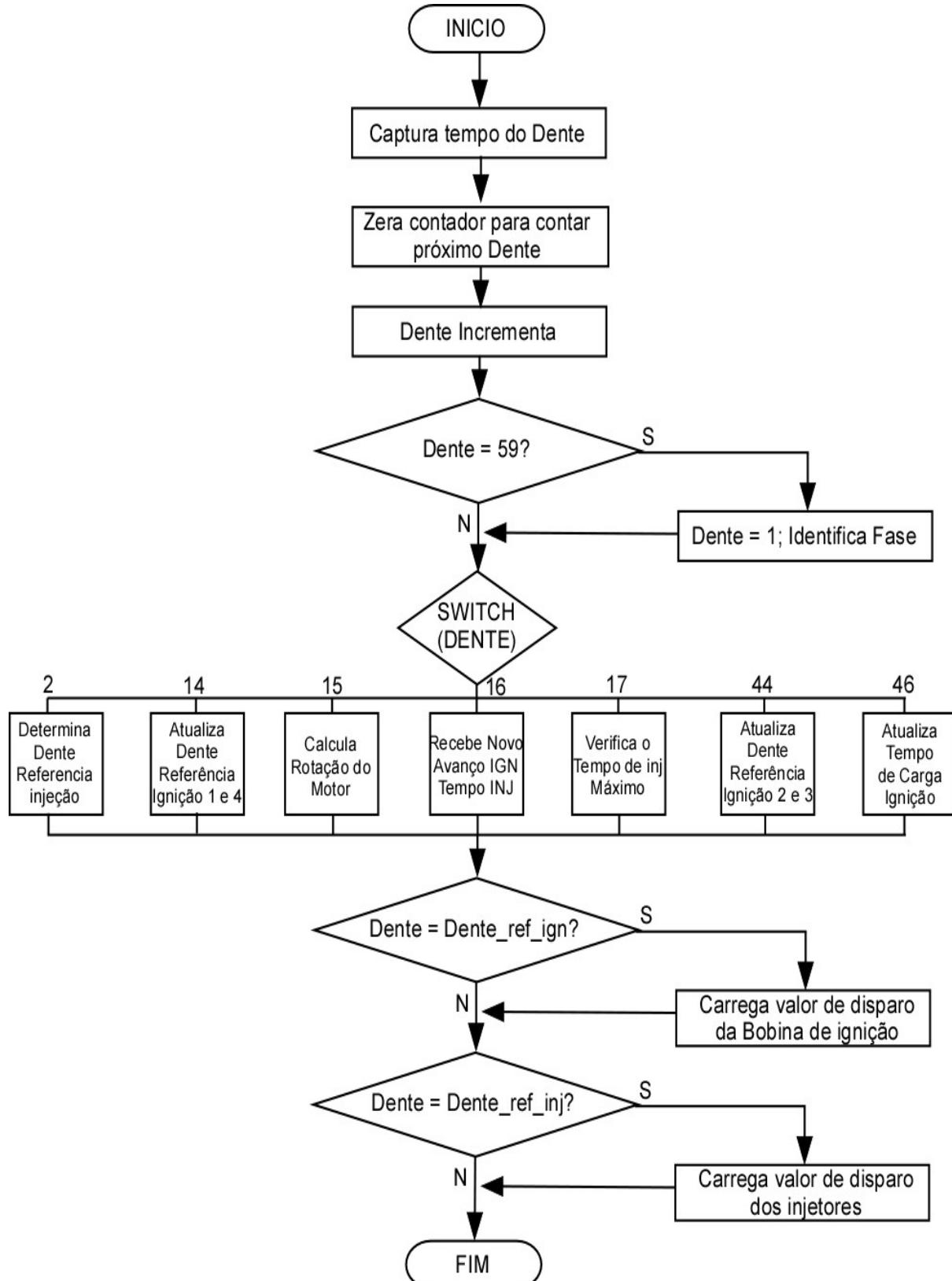
sincronismo fica aguardando ocorrer a primeira interrupção externa do sinal do sensor de fase. Esta interrupção indica que o motorista está tentando dar a partida e o motor está girando através do motor de partida do veículo. Nesse instante, a camada de sincronismo deve se sincronizar com a fase atual do motor e, para isto, foi desenvolvida uma rotina de tratamento do sinal do sensor de fase, que possui a função de efetuar o primeiro sincronismo entre o motor e o programa. Somente após sincronismo do programa com o motor, que acontece em média em uma volta completa do motor, é que a rotina de tratamento do sensor de fase habilita a interrupção externa do sensor de rotação.

Com o programa sincronizado e com a interrupção do sensor de rotação habilitada, o programa inicia a lógica de controle para comandar o acionamento das válvulas injetoras e da bobina de ignição, conforme a figura 49.

Conforme descrito na seção 3.2.3.2, o sinal do sensor de rotação do motor é de extrema importância para todas as operações do gerenciamento eletrônico do motor. A UCM sincroniza todo o sistema e determina o momento exato para acionamento das válvulas injetoras de combustível e o avanço de ignição em função do sinal de rotação do motor. O microcontrolador processa esse sinal através das interrupções por borda de subida. Ou seja, a rotina de tratamento da interrupção externa do sinal do sensor de rotação é executada a cada vez que um novo dente cruza pelo pino de entrada do microcontrolador.

Conforme fluxograma apresentado na figura 49, o tempo do dente é medido a toda nova execução da função. Este tempo é medido pelo *timer* FTM0 que incrementa seu valor a cada $1.28\mu\text{s}$. Após capturar o tempo atual do dente, a rotina zera o contador do *timer* para que o próximo dente possa ser medido pelo timer. Nesse instante, a rotina incrementa o valor da variável estática “dente”. Esta variável é o principal parâmetro para toda lógica desta rotina de tratamento, pois com este parâmetro o programa consegue determinar a posição angular exata do motor. Para aumentar a eficiência dessa rotina e tendo em vista que uma volta completa do motor dura de 10ms à 100ms, optou-se por executar apenas uma ação por dente. Deste modo, foi utilizado o laço padrão da linguagem C “switch case” para determinar qual ação deverá ser tomada pela rotina de tratamento do sinal do sensor de rotação. Isto possibilitou criar uma lógica eficiente e eficaz para sincronismo do programa com o motor.

Figura 49 - Fluxograma da rotina de tratamento da interrupção do sensor de rotação do motor



Fonte: o autor

Esta rotina de tratamento determina o dente de referência para os injetores no segundo dente, determina o dente de referência para a bobina de ignição do banco 1 e 4 no décimo quarto dente e para a bobina de ignição do banco 2 e 3 no quadragésimo quarto dente. Esses dentes de referência foram calculados externamente em função da rotação do motor e armazenados em mapas alocados dentro da memória EEPROM do microcontrolador.

No décimo quinto dente, essa rotina calcula a rotação do motor através do valor do dente obtido do *timer* FTM0 no início da rotina de tratamento. Esse parâmetro é carregado no vetor de estados do motor RTOS_VEM, para que seja utilizado pelas tarefas da camada RTOS. No décimo sexto dente, a rotina de tratamento recebe da camada RTOS os novos valores calculados para o avanço do ponto de ignição e para o tempo de injeção de combustível. No décimo sétimo dente, esta rotina verifica se o tempo de injeção atual é compatível com a rotação atual do motor para que se evite cruzamento de interrupções para acionamento das válvulas injetoras. Como duas válvulas injetoras são acionadas por *timer*, se o tempo que uma válvula injetora deve permanecer aberta for maior que a própria rotação atual do motor, então a rotina de sincronismo irá ativar uma ação corretiva, para que não ocorra aberturas indevidas das válvulas injetoras. O sincronismo indevido entre as válvulas injetoras e o motor prejudicaria todas as ações de controle propostas neste projeto.

No décimo sexto dente, o tempo de carga da bobina é calculado em função do regime de operação e rotação do motor e da tensão de carga do alternador. Após efetuar as ações do laço "switch case", a rotina de tratamento verifica se o dente atual é igual ao dente de referência para o sistema de injeção e ignição. Em caso afirmativo, a rotina carrega o valor de disparo previamente recebido no *timer* designado para controlar o respectivo injetor ou bobina, e em seguida habilita sua interrupção. O *timer* escolhido para controlar as válvulas injetoras e as bobinas de ignição é o PIT. O PIT0 controla as válvulas injetoras 1 e 4, o PIT1 controla as válvulas 2 e 3, o PIT2 controla a bobina do banco 1 e 4 e o PIT3 controla a bobina do banco 2 e 3. Esse *timer* possui uma contagem decrescente e a interrupção ocorre quando ele atinge o valor zero. Ao ocorrer a interrupção, o respectivo atuador é acionado e o *timer* é carregado novamente com o valor do tempo de injeção para as válvulas injetoras ou com o tempo de carga para as bobinas de ignição. Ao ocorrer a próxima interrupção do *timer*, o respectivo atuador é desligado, dando inicio a uma

nova interação da camada de sincronismo. O *firmware* completo desenvolvido está disponível nos apêndices desta dissertação.

3.4 CONSIDERAÇÕES FINAIS

A metodologia empregada no desenvolvimento do *hardware* e *firmware* utilizou uma estrutura analítica de projetos. O conceito aplicado prevê a elaboração progressiva de cada componente no escopo geral do projeto. Desta forma, para desenvolver o circuito eletrônico de condicionamento de cada sensor ou atuador, o circuito foi projetado baseando-se nos valores medidos em ensaios individuais de cada componente. Inicialmente, em bancada e posteriormente no próprio veículo, foi analisado o sinal elétrico emitido por cada sensor e atuador. A análise teve como objetivo coletar amplitudes elétricas e avaliar os ruídos embutidos no sinal de sensor e atuador. Foi elaborado um extenso banco de dados com testes em variadas rotações e condições de operação do motor. Com todos os dados coletados e analisados, cada circuito eletrônico foi projetado e testado em bancada separadamente. Após a conclusão de todos os circuitos, o processo de fabricação de um *hardware* final foi concretizado. Todas as trilhas do circuito impresso foram desenvolvidas levando em consideração a interferência eletromagnética, utilizando a teoria da indução por corrente elétrica.

Seguindo a estrutura analítica e progressiva de desenvolvimento do projeto, o *firmware* para o controle do motor foi elaborado. O desenvolvimento de um algoritmo de gerenciamento eletrônico de motores a combustão interna é extenso e optou-se por utilizar aplicações automotivas de tempo real, integradas ao sistema. O ideal nesta aplicação seria utilizar o AUTOSAR (*Automotive Open System Architecture*), que é uma arquitetura de *firmware* automotivo padronizado. O objetivo do AUTOSAR é criar e estabelecer padrões para sistemas automotivos, que irá proporcionar uma infraestrutura básica para ajudar com o desenvolvimento do *firmware*. Entretanto, o RTOS (*Real-Time Operating System*), que possui algumas aplicações com licenças gratuitas que são compatíveis com o microcontrolador escolhido neste projeto, acabou sendo a escolha para o projeto. Ele também apresenta excelentes resultados em testes práticos, se mostrando eficaz para atender a toda estrutura de *firmware* proposta. Um RTOS facilita a divisão do trabalho entre interrupções e tarefas, melhorando o desempenho e diminuindo consid-

eravelmente o tempo de latência do sistema. Com um RTOS, a arquitetura do *firmware* pode ser toda baseada em eventos e interrupções, economizando processamento e melhorando a eficiência da aplicação.

O *firmware* proposto deve gerenciar um grande número de tarefas e interrupções que poderão ocorrer ao mesmo tempo. O *firmware* desenvolvido segue uma lógica de prioridades atendendo primeiramente às tarefas de sincronismo do motor e posteriormente às tarefas de comunicação externa, diagnose, conforto e conveniência do gerenciamento. Devido ao curto tempo para execução de cada tarefa, o microcontrolador escolhido possui um poderoso núcleo de processamento que opera com 100 MHz, executando em média uma instrução a cada 20ns. A ideia na elaboração do algoritmo é desenvolver tarefas totalmente modulares e específicas para aquisição ou controle de cada componente presente no gerenciamento eletrônico.

Pequenos algoritmos foram elaborados e implementados no *hardware* deste trabalho. Todos os algoritmos desenvolvidos tiveram como objetivo testar individualmente cada circuito eletrônico da UCM e os testes apresentaram resultados satisfatórios em experimentos de bancada.

Para validar a metodologia proposta neste trabalho, o próximo capítulo apresenta os resultados alcançados. Com a análise dos resultados, será possível compreender toda a dinâmica de um motor ciclo Otto e refletir sobre os reais motivos que levaram com que o gerenciamento eletrônico fosse implementado, validando os modelos teóricos em ensaios práticos.

4 RESULTADOS

Este capítulo apresenta os resultados alcançados neste projeto. Seguindo o conceito de elaboração progressiva de cada componente, previsto no escopo geral do projeto, inicialmente foram especificadas as condições dos testes realizados e os equipamentos utilizados. Os resultados serão apresentados de forma quantitativa e analisados com o objetivo de concretizar a metodologia proposta.

4.1 SISTEMA DE INTERSECÇÃO DOS SINAIS ELÉTRICOS DA UCM

A arquitetura elétrica, normalmente encontrada em um automóvel, é constituída por componentes selados como, por exemplo, conectores e terminais de contato elétricos. Esses componentes tem o objetivo de proteger todo o circuito elétrico contra infiltrações, curtos-circuitos, interferência eletromagnética e ações de agentes externos. Adicionalmente, estes conectores dificultam qualquer ação de controle sobre o motor ou até mesmo aquisição de dados e comparação com outros sistemas de controle do motor. Como o projeto precisa medir os sinais elétricos disponíveis nos sensores e atuadores do motor para análise e um posterior controle, foi projetado um sistema de intersecção dos sinais elétricos. A figura 50 ilustra o resultado final do sistema de intersecção dos sinais elétricos do motor elaborado.

Figura 50 - Sistema de intersecção da UCM.

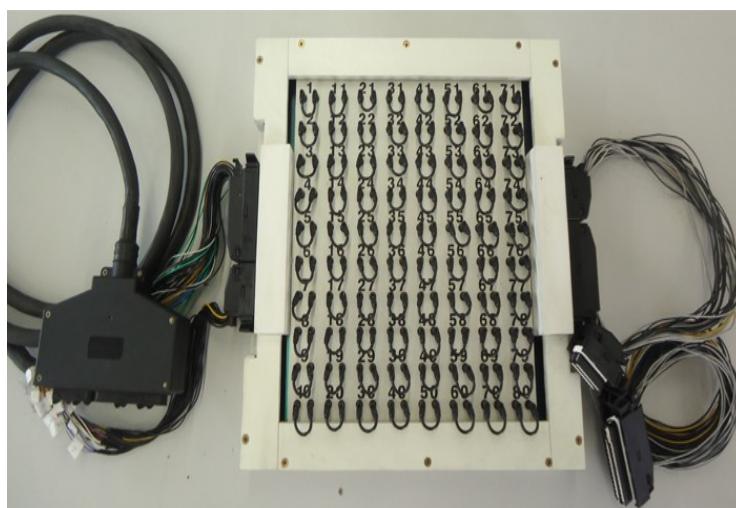
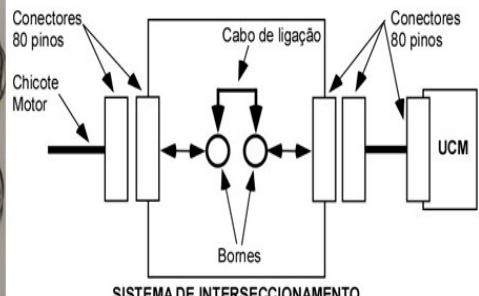


ILUSTRAÇÃO DO ESQUEMA DE LIGAÇÃO ELÉTRICA



Fonte: o autor

Para obter acesso aos sinais dos sensores e atuadores presentes no motor, foi utilizada uma intersecção dos sinais elétricos. Além de possibilitar o monitoramento completo de até 80 sinais elétricos através dos 80 canais disponíveis no sistema de intersecção, este componente possibilita, através de *jumpers*, aplicar sinais de atuação no motor, desligando, neste caso, sinais de atuação provenientes da UCM original do veículo. Deste modo, foi possível controlar parcialmente o motor, deixando somente alguns atuadores para serem controlados pela UCM original do veículo, dando assim, uma ordem cronológica e progressiva de desenvolvimento de toda a estratégia de controle do motor. Durante a elaboração do sistema de sincronismo do motor, um controle de atuador era adicionado isoladamente. Esta ação visou garantir a integridade das peças mecânicas e elétricas do motor em possíveis erros de controle do respectivo atuador em elaboração. Nos testes finais realizados, a UCM desenvolvida neste projeto foi responsável por todo o controle do motor e não foi necessário utilizar o sistema de intersecção desenvolvido, tendo em vista que a UCM foi elaborada para uma aplicação direta no chicote original do veículo utilizado.

4.2 SISTEMA DE AQUISIÇÃO DE DADOS PARA ANÁLISE DOS RESULTADOS

Com o sistema de intersecção concluído, foi iniciado o desenvolvimento de um sistema capaz de adquirir todos os sinais em tempo real dos testes efetuados e armazená-los para posterior análise de todos os sinais elétricos que compreendem o sistema de gerenciamento eletrônico do motor. Este sistema é composto por um computador para a aquisição dos dados e pelo sistema de intersecção, anteriormente descrito.

Para o processo de aquisição de dados, foi utilizado um computador equipado com um *hardware* dedicado de aquisição de dados, fabricado pela empresa *National Instruments*. O modelo escolhido para aquisição de dados foi o NI PCI 6514 com 32 entradas e 32 saídas. O *hardware* DAQ atua como a *interface* entre um computador e sinais do mundo exterior. Ele funciona basicamente como um dispositivo que digitaliza sinais analógicos de entrada, de forma que um computador possa interpretá-los. Os três componentes principais de um dispositivo DAQ, usados para medir um sinal, são os circuitos eletrônicos de condicionamento de sinais, conversor analógico-digital (ADC) e o barramento do computador.

Atuando em conjunto com o *hardware* DAQ, o *software* de aplicação é parte vital dos modernos sistemas de aquisição de dados. A ferramenta de *software* adotada nesta aplicação é denominada *LabVIEW®*, também desenvolvida pela *National Instruments*. O código desenvolvido armazena e disponibiliza os sinais do motor em arquivos eletrônicos, podendo ser exportados para diversos programas de análises gráficas. Para facilitar a interpretação dos sinais adquiridos, foi utilizado o *software* *DIAdem®*, também desenvolvido pela *National Instruments*. Este *software* criou gráficos de todos os sinais armazenados da maneira conveniente, permitindo comparar os sinais obtidos nos testes efetuados com o sistema de gerenciamento eletrônico elaborado neste projeto, com os sinais do sistema de gerenciamento eletrônico original do motor EA 113.

Os dados foram adquiridos em todos os testes efetuados, tanto com o sistema original quanto com o sistema desenvolvido neste projeto. Foram utilizadas todas as faixas de rotações, simulando situações no dinamômetro como, por exemplo, aplicação de carga e alteração de marchas do sistema de transmissão do veículo. Um extenso banco de dados de aproximadamente 3 Gigabytes foi criado. Com o banco de dados construído, não houve mais a necessidade da presença do veículo durante o desenvolvimento do *hardware* e do *firmware* proposto neste projeto, facilitando o desenvolvimento e os testes em bancadas da UCM. O veículo foi novamente utilizado somente nos testes finais para validação do modelo proposto e ajustes finais.

4.3 DESENVOLVIMENTO DO LAYOUT E DA PLACA DE CIRCUITO IMPRESSO

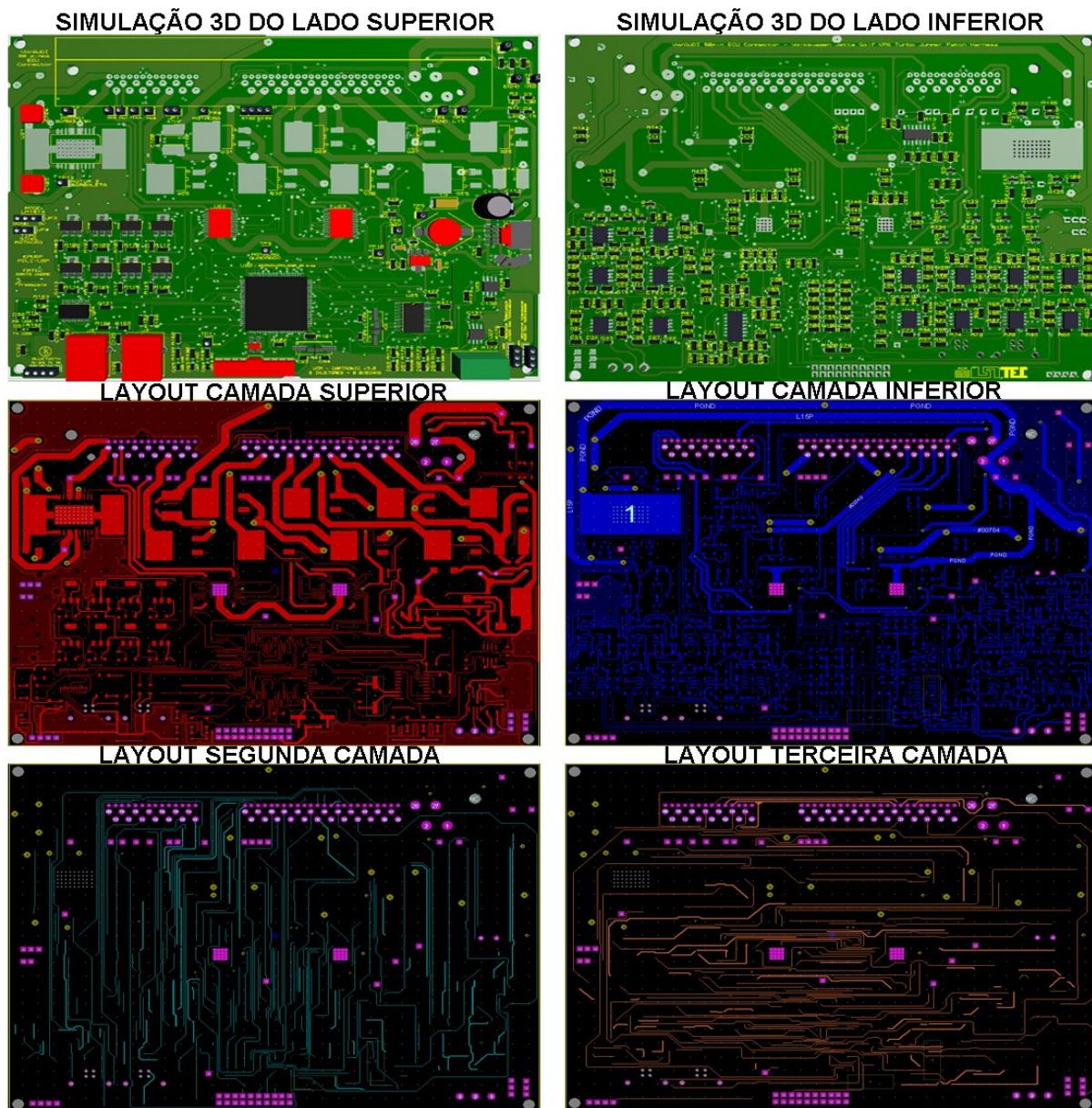
Baseando-se nos dados práticos adquiridos com o sistema de aquisição, os circuitos de condicionamento dos sensores e atuadores foram elaborados e testados em bancada, conforme descrição no capítulo 3 deste trabalho. A densidade de conexões elétricas existentes no projeto elaborado inviabilizou o desenvolvimento de um *hardware* utilizando técnicas manuais de elaboração das placas de circuito impresso.

Seguindo a proposta inicial do projeto, a plataforma CAD adotada para confeccionar o *layout* da placa de circuito impresso foi o *software* *Proteus®*, desenvolvido pela *Labcenter Electronics®*. Ela foi utilizada para elaborar o esquema elétrico, simular a funcionalidade do sistema, definir a área útil, realizar o

posicionamento dos componentes e rotear as trilhas. Deste modo, todo o processo de desenvolvimento da UCM foi agilizado.

A figura 51 ilustra o *layout* das trilhas e o posicionamento dos componentes da UCM finalizada. A figura 51 apresenta a simulação 3D da face superior do *layout* do circuito impresso, a simulação 3D da face inferior do *layout* do circuito impresso, além de ilustrar exatamente o *design* efetuado em todas as trilhas das quatro camadas que o *layout* desenvolvido apresenta.

Figura 51 - Layout da placa de circuito impresso finalizado.



Fonte: o autor

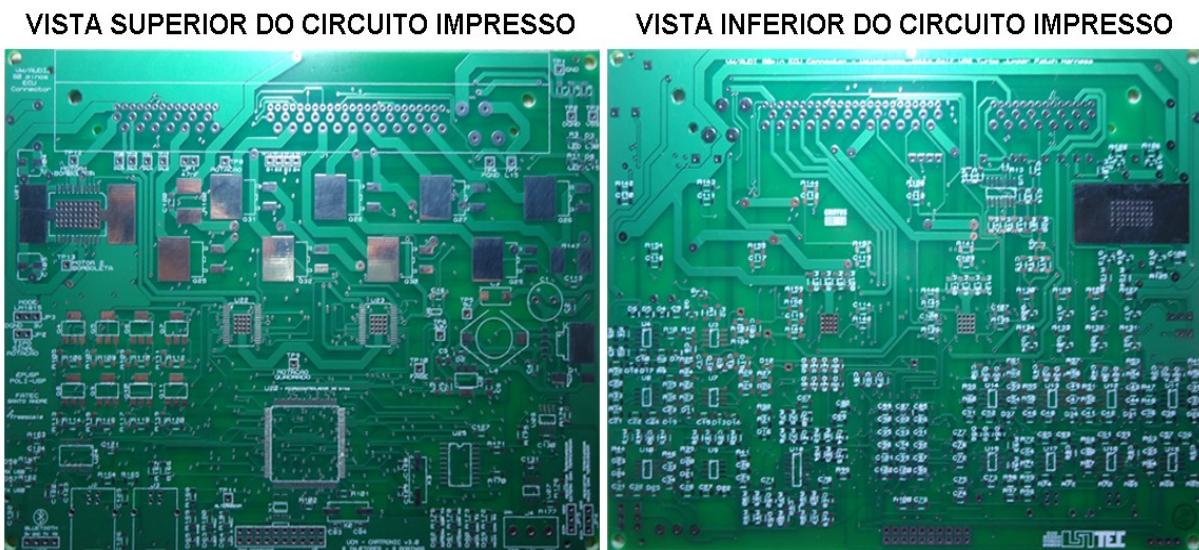
Devido à densidade elevada no total de trilhas do circuito impresso elaborado, o circuito foi desenvolvido em quatro camadas de aplicação. Nas camadas

superiores e inferiores, estão projetadas as trilhas cuja corrente percorrida é superior à 1A. Nas camadas centrais foram projetadas as trilhas digitais e de comunicação. Um fato importante no desenvolvimento deste *layout* foi a preocupação com a interferência eletromagnética e seus efeitos indesejados no desempenho final do projeto.

4.4 UNIDADE DE CONTROLE DO MOTOR FINALIZADA

Conforme mencionado anteriormente, o circuito impresso foi desenvolvido na plataforma *Proteus®*. Devido à grande densidade de componentes, trilhas e ilhas existentes neste projeto, o circuito impresso foi desenvolvido externamente por uma empresa especializada na confecção de circuitos impressos. Este circuito foi produzido em uma placa de fibra de vidro com quatro faces impressas, furos metalizados e acabamento “hot air”. A figura 52 apresenta o circuito impresso finalizado.

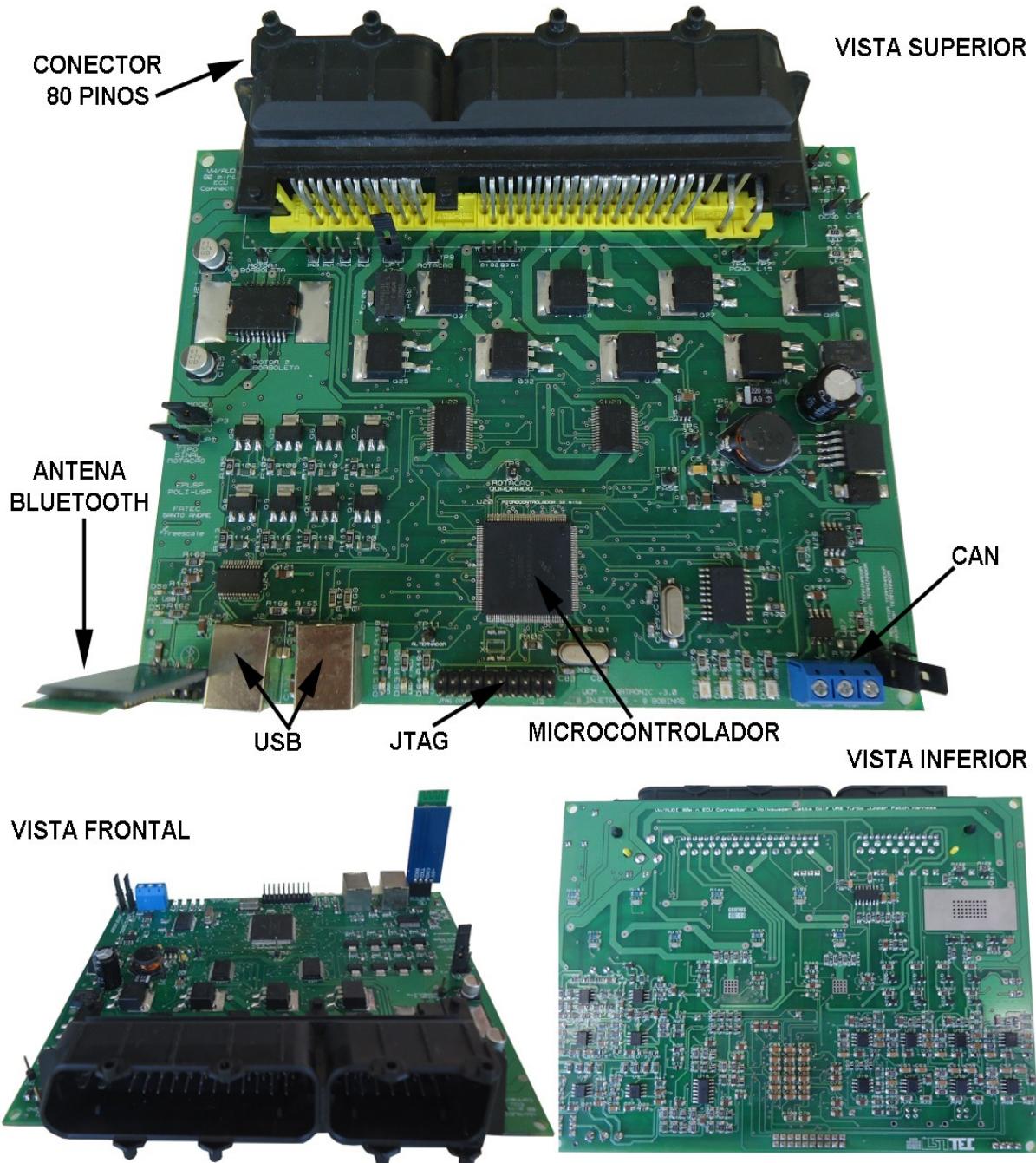
Figura 52 - Circuito impresso finalizado.



Fonte: o autor

Com o circuito impresso finalizado, iniciou-se o processo de solda dos componentes. Nesta versão, optou-se pela utilização de componentes eletrônicos SMD para reduzir as dimensões da placa de circuito impresso. A figura 53 apresenta a UCM finalizada.

Figura 53 - Hardware finalizado.



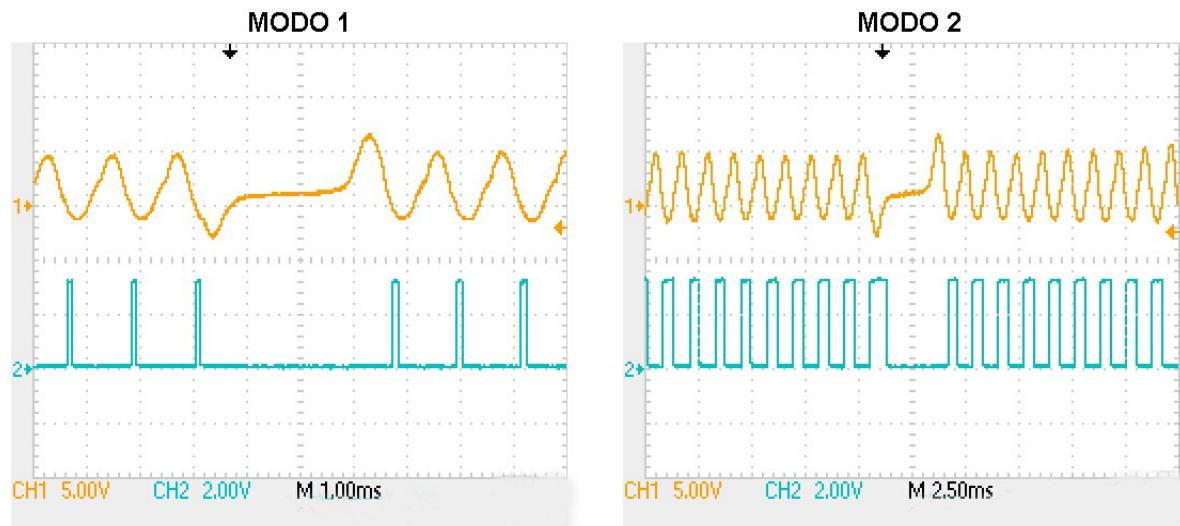
Fonte: o autor

A figura 53 ilustra claramente como foram posicionados os componentes eletrônicos, o microcontrolador, o conector de gravação JTAG, os conectores USB, o conector automotivo de entrada e saída dos sinais. Adicionalmente, é possível visualizar o conector da rede CAN e a antena de comunicação *bluetooth*.

4.5 VALIDAÇÃO DO HARDWARE

Todas as funções inicialmente projetadas foram testadas e validadas, com o objetivo de avaliar se a unidade de controle do motor construída realmente atende às suas especificações iniciais. Além disso, todos os sensores e atuadores especificados no projeto foram testados. Para realizar os testes dos circuitos projetados, foram desenvolvidos pequenos *firmware* de testes com a função de controlar, através de pulsos de acionamento gerados pelo microcontrolador, os atuadores ou adquirir e processar os sinais emitidos pelos sensores. Para validar os testes, um osciloscópio digital foi utilizado para medir os sinais da entrada e saída. Em todos os testes, foram utilizados sensores e atuadores reais. A seguir, são descritos os principais testes efetuados no processo de validação do *hardware*.

Figura 54 - Condicionamento do sinal de rotação.



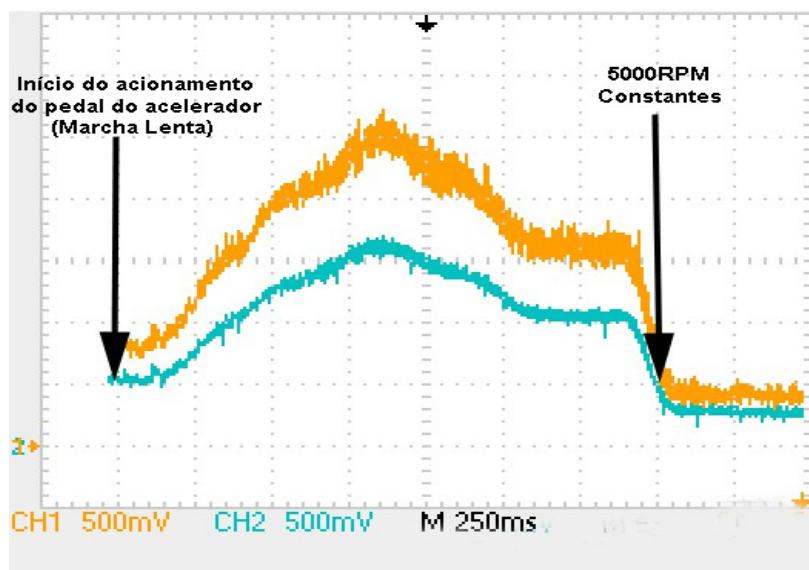
Fonte: o autor

A figura 54 ilustra o resultado do circuito eletrônico de condicionamento do sinal de rotação do motor, desenvolvido e descrito na seção 3.2.3.2. O teste aplicado para validar o circuito foi efetuado utilizando o próprio sensor de rotação do veículo. Um *firmware* de testes foi desenvolvido e gravado no microcontrolador. Este *firmware* possui a função de configurar o pino de entrada do sinal do microcontrolador. Nos dois modos de operação ilustrados na figura 54, o motor está na condição de marcha lenta, e o sinal senoidal presente no canal 1 do osciloscópio é o sinal de entrada da UCM desenvolvida. O sinal digital presente no canal 2 do osciloscópio

representa a saída do circuito de condicionamento. Este sinal de saída do circuito de condicionamento entra diretamente no pino do microcontrolador. O circuito desenvolvido possui dois modos de operação, que podem ser selecionados através de um *jumper* na placa de circuito impresso. No primeiro modo de operação, o sinal de saída do circuito assume valor igual a VDD (3.3V) quando o sinal senoidal do sensor cruza sua referência do semi-ciclo positivo para o semi-ciclo negativo, em seguida permanece com valor inalterado por 100 μ s, independentemente da rotação do motor e somente após 100 μ s assume valor igual a VSS (GND). O segundo modo de operação atua de forma semelhante, porém o valor de saída do circuito de condicionamento somente assumirá o valor igual a VSS quando o sinal senoidal do sensor de rotação cruzar sua referência do semi-ciclo negativo para o semi-ciclo positivo.

Além do circuito de condicionamento do sinal do sensor de rotação do motor, o circuito eletrônico de condicionamento do sinal do sensor de pressão no coletor foi validado. A figura 55 ilustra o resultado obtido no teste de validação do circuito descrito na seção 3.2.3.4.

Figura 55 - Condicionamento do sinal do sensor de pressão no coletor de admissão.



Fonte: o autor

Para validar o circuito eletrônico de condicionamento do sensor de pressão, o teste foi efetuado utilizando o próprio sensor de pressão MAP do veículo. O teste teve como objetivo avaliar a variação do sensor de pressão em uma situação de aceleração abrupta do motor sem carga. Na figura 55, o canal 1 ilustra o sinal de

entrada do sensor de pressão, o canal 2 ilustra o sinal de saída do circuito de condicionamento. Os dois canais estão na mesma referência, mostrando que o circuito desenvolvido atua de forma satisfatória, o sinal no canal 2 acompanha o sinal do canal 1 com uma atenuação na amplitude do sinal devido ao divisor resistivo na entrada do circuito que possui a função de atenuar o sinal de entrada para 3.3V adequando aos limites impostos pelo microcontrolador. Adicionalmente, é possível observar uma pequena redução no nível de ruído presente no sinal em função do filtro eletrônico projetado no circuito.

Todos os outros circuitos eletrônicos de condicionamento dos sensores e de interface para os atuadores, projetados e descritos no capítulo 3, foram testados de forma individual e similar aos testes apresentados para o sensor de rotação e pressão no coletor de admissão. Estes pequenos testes efetuados mostraram que o hardware projetado atende as expectativas satisfatoriamente.

Essa validação foi extremamente importante para a continuidade do projeto. Como este projeto propõe, além do controle do motor, o desenvolvimento de um hardware capaz de efetuar este controle, se os resultados obtidos nesta etapa do projeto não fossem satisfatórios, consequentemente não seria possível prosseguir com o restante das atividades previstas para conclusão do projeto como, por exemplo, o levantamento do modelo em primeira ordem para o controle do motor, o sincronismo dos sistemas de ignição, injeção e admissão, o desenvolvimento da interface homem-máquina, dentre outros.

4.6 SÍNCRONISMO DOS SISTEMAS DE CONTROLE DO MOTOR

Conforme descrito no capítulo 2, o motor de combustão interna ciclo Otto opera essencialmente com três parâmetros básicos para gerar torque: a massa de ar, a massa de combustível e a centelha elétrica da vela de ignição no momento correto, conforme os quatro ciclos de operação. Neste projeto, estes três sistemas são os principais parâmetros de controle da UCM desenvolvida, e os três sistemas foram desenvolvidos de forma paralela e independente. Esta seção descreve a lógica de sincronismo e os modelos de controle, assim como as técnicas de controle utilizadas em cada um dos três sistemas.

4.6.1 Controle do Sistema de Ignição

O sistema de ignição deve realizar, com confiabilidade, a combustão da mistura comprimida com precisão, mesmo sob condições operacionais dinâmicas, com as flutuações substanciais nos padrões de fluxo da mistura ar/combustível. Conforme apresentado no capítulo 2, o tempo que o primário deve permanecer energizado pelo sistema de controle, o tempo de carga da bobina deve ser o suficiente para garantir a centelha capaz de iniciar a combustão da mistura carburante dentro da câmara de combustão. Em análises ao sistema de controle original do motor utilizado neste trabalho e com base nos dados existentes no catálogo técnico da bobina utilizada neste motor, o tempo de carga mínimo para se realizar uma combustão satisfatória é de 5.8ms. Este tempo de carga pode variar de acordo com a tensão de alimentação primária fornecida pelo conjunto bateria/alternador do veículo, podendo chegar à 8ms em situações extremas como é o caso da partida do motor.

Além do tempo de carga da bobina, o sistema de controle deve atuar no ângulo de ignição ao avanço de ignição. Esse é principal parâmetro de controle do sistema de ignição, e foi detalhado no capítulo 2. Esse parâmetro não é constante e está diretamente associado à rotação e carga aplicada ao motor. O avanço do ponto de ignição promove uma variação crescente na pressão da mistura carburante e um consequente aumento na potência do motor. Entretanto, ele é limitado pelo fenômeno conhecido como detonação.

A modelagem matemática deste sistema de controle é bastante complexa e envolve diversos parâmetros de controle do motor como, por exemplo, a variação da tensão da bateria, a qualidade do combustível, a pressão atmosférica, o desgaste natural do motor, dentre outros fatores. Devido a este fato, optou-se por mapear e armazenar em memórias dentro do microcontrolador o avanço de ignição para todas as faixas de rotação e carga do motor em testes efetuados em um dinamômetro inercial.

Para realizar a aquisição dos pontos de avanço de ignição, foram utilizados os sistemas de controle original do veículo, de intersecção do chicote do motor, de aquisição de dados desenvolvido no *LabVIEW®* e um dinamômetro inercial. A ideia neste teste era variar apenas o avanço de ignição e monitorar o sinal do sensor de detonação com o sistema de aquisição. Para controlar apenas o avanço, foi

desenvolvido um pequeno *firmware* na UCM elaborada neste projeto, com o objetivo de efetuar o controle somente da bobina de ignição. O sistema de intersecção foi utilizado para interromper o sinal de controle da UCM original apenas nos pinos referentes ao controle das duas bobinas de ignição. Os demais parâmetros de controle do motor, como as válvulas injetoras e a válvula borboleta, foram mantidos sobre controle da UCM original. Desta maneira, foi possível variar a rotação do motor normalmente no dinamômetro e alterar de forma própria o avanço de ignição pela UCM desenvolvida neste projeto e monitorar a ocorrência do fenômeno de detonação pelo sistema de aquisição de dados do sinal do sensor de detonação. O avanço escolhido para o mapa efetuado foi o máximo avanço de ignição possível sem que houvesse ocorrências do fenômeno de detonação no motor. A tabela 4 ilustra o mapa adquirido com os testes efetuados no dinamômetro.

Tabela 4 - Tabela de avanço de ignição adquirida em ensaios no dinamômetro

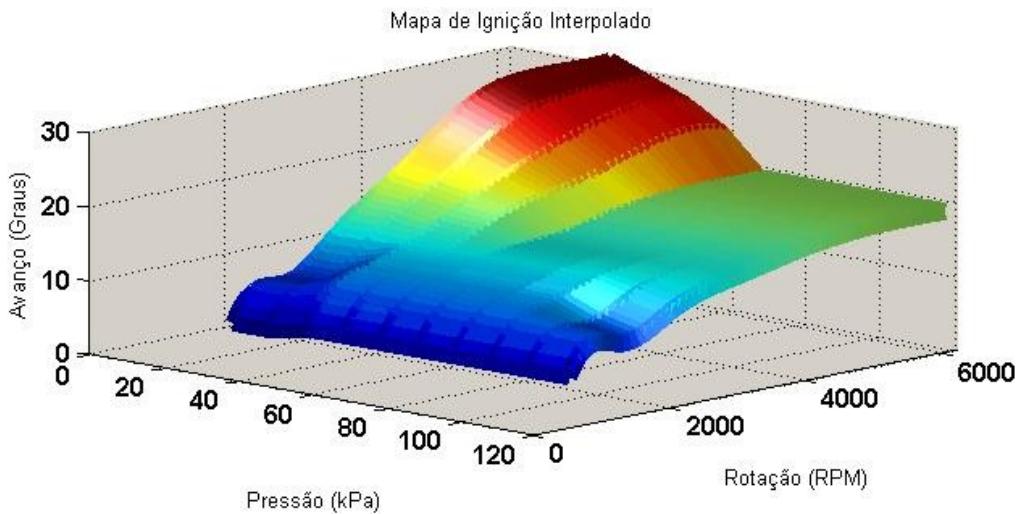
RPM \ MAP(KPa)	10	20	30	40	50	60	70	80	90	100	110	120
800	09	09	09	09	10	10	10	10	10	10	10	10
1000	09	09	10	10	12	12	12	12	12	12	11	12
1500	10	10	10	10	13	13	13	13	13	13	11	10
2000	15	15	15	15	15	15	15	15	15	15	14	13
2500	18	18	18	18	17	17	15	15	15	15	15	14
3000	22	22	22	21	20	19	16	16	16	16	16	16
3500	25	25	25	24	22	20	16	16	16	16	16	16
4000	29	29	29	27	24	21	18	18	18	18	18	18
4500	30	30	30	28	25	22	18	18	18	18	18	18
5000	30	30	30	29	27	24	19	19	19	19	19	19
5500	30	30	30	28	27	24	19	19	19	19	19	19
6000	30	30	30	29	27	24	19	19	19	19	19	19

Fonte: o autor

Para realizar os ensaios no dinamômetro, o veículo foi acelerado em rotações determinadas conforme a tabela 4. O teste foi efetuado em terceira marcha do sistema de transmissão. Esta marcha foi determinada por apresentar a maior linearidade na relação de transmissão entre o motor e a roda. Quando a rotação se estabilizava nos valores determinados conforme a tabela, o avanço de ignição era incrementado um grau por vez pela UCM desenvolvida neste projeto. O comando para se avançar o ponto de ignição foi fornecido pela interface Homem-Maquina (HMI) elaborada, que será descrita nas próximas seções. Quando o sensor de detonação começava a indicar a ocorrência de detonação, o ponto de avanço era determinado como sendo um grau anterior ao ponto de avanço atual. O teste foi efetuado a cada 500 RPM. Com a tabela determinada, foi iniciado o processo de interpolação dos pontos da tabela. Com o auxílio do software *Matlab®*, a tabela foi

interpolada, utilizando o método de interpolação com duas dimensões, com 61 linhas. Para isto, foram utilizados os comandos "*meshgrid*" e "*interp2*" no *Matlab®*. O resultado foi um mapa de avanço de ignição interpolado, com uma resolução de 100 RPM, conforme ilustra a figura 56.

Figura 56 - Mapa de Avanço de Ignição Interpolado e finalizado.



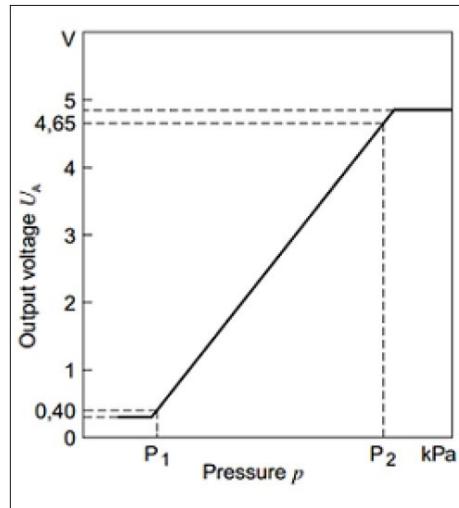
Fonte: o autor

Para implementar o sistema de ignição no *firmware* desenvolvido para o microcontrolador da UCM deste projeto, foi necessário se calcular a rotação atual do motor e se converter o sinal do sensor MAP. Para se calcular essa rotação, converte-se o valor obtido pelo *timer* FTM0 do microcontrolador, que incrementa seu valor a cada $1.28\mu\text{s}$, em rotações por minutos dividido por cem (RPM/100). A rotação é calculada com base no intervalo de tempo de cada dente do sinal do sensor de rotação. A equação 30 retorna o valor da rotação do motor convertido, ou seja, quando o motor estiver com 1000 RPM está equação irá retornar o valor 10.

$$RPM_{/100} = \frac{1000000}{t_{dente} \cdot 128} \quad \text{Eq.30}$$

O sensor MAP é responsável por medir a pressão da massa de ar admitida no coletor de admissão. Tipicamente, esse sensor fornece em sua saída um valor de tensão que é proporcional ao valor de pressão do ar admitido. A figura 57 ilustra a curva característica desse sensor, fornecida no seu catálogo técnico produzido pela Bosch.

Figura 57 - Curva característica do sensor MAP utilizado neste projeto



Fonte: (BOSCH, 2005)

Conforme o circuito de condicionamento, apresentado na figura 32 no capítulo 3, o sinal de entrada do sensor possui uma atenuação de 66% até entrar no microcontrolador. Considerando que o conversor ADC utilizado no microcontrolador foi configurado para possuir uma resolução de 8 bits, ou seja, quando a tensão de entrada no pino do microcontrolador possuir 3.3V, o valor digital de saída do ADC será de 255, a função desenvolvida para se converter o sinal de entrada do sensor MAP para (KPa/10) é ilustrada na figura 58.

Figura 58 - Função de conversão do sensor MAP.

```
uint8_t SensorMAP (void) {
    uint8_t MAP = 0; // Inicia nova variável MAP

    ADC0_SC1A = 10 & ADC_SC1_ADCH_MASK; // Inicia conversao canal SE10
    while ((ADC0_SC2 & ADC_SC2_ACTR_MASK)); // Aguarda ADC em progresso
    while (! (ADC0_SC1A && ADC_SC1_COCO_MASK)); // aguarda conversao terminar
    MAP = ADC0_RA; // Captura o resultado;

    MAP = (0.0435 * MAP) + 1.08; // Converte Saida em (kPa / 10)

    return MAP; // Retorna valor de MAP convertido
}
```

Fonte: o autor

O cálculo de conversão efetuado segue a equação da reta ($y = ax + b$) e os pontos para o cálculo foram retirados da curva característica do sensor MAP, apresentados na figura 57. Após calcular o valor atual da rotação do motor e converter o valor de entrada do sensor MAP no conversor ADC, o microcontrolador acessa o mapa de avanço de ignição gravado na memória interna. A função

implementada para se calcular o avanço de ignição, executada na camada RTOS do *firmware* a cada 10ms, é ilustrada na figura 59.

Figura 59 - Função de Cálculo do Avanço de Ignição.

```
void CalculoAvancoIGN(uint16_t *t_dente, uint8_t *avancoIGN) {
    uint8_t Avanco = 0;           //Variavel do Avanco de Ignicao
    uint8_t MAP = 0;             //Variavel de aquisicao do valor de MAP
    uint8_t RPM = 0;             //Variavel de aquisicao do valor de RPM

    MAP = SensorMAP();          //Calcula pressao em (kPa/10)
    RPM = 1000000 / ((*t_dente) * 128); //Retorna valor da rotacao(RPM/100)
    Avanco = MapaIgnicao [RPM][MAP]; //Calcula Avanco de Ignicao
    *avancoIGN = Avanco;         //Atualiza ponteiro de saida da funcao

    RTOS_VEM [AVANCO_VEM] = Avanco; //Atualiza Vetor de Estados RTOS
    RTOS_VEM [MAP_VEM] = MAP;      //Atualiza Vetor de Estados RTOS
}

//Final da Funcao
```

Fonte: o autor

O avanço foi armazenado internamente em uma matriz denominada "Mapalgnicao [61][13]" na memória *FlexRAM* do microcontrolador, durante a inicialização do *firmware* realizado na camada principal do programa, conforme descrito no capítulo 3. A função "CalculoAvancoIGN" deve receber os endereços (&ponteiros) das variáveis "t_dente" e "avancoIGN" que são parâmetros originários da camada de sincronismo e, após calcular o valor do sensor MAP e a rotação atual do motor, a função deve utilizar as variáveis "RPM" e "MAP" para indexar a matriz "Mapalgnião" e receber o novo avanço de ignição. Este novo parâmetro é carregado no ponteiro "avancoIGN" para ser utilizado pela camada de sincronismo e carregado no vetor de estados do motor "RTOS_VEM", para que seja utilizado pelas tarefas do sistema operacional na camada RTOS.

4.6.2 Controle do Sistema de Injeção

O algoritmo utilizado para se efetuar o controle do sistema de injeção de combustível segue os modelos físicos propostos por Elbert Hendricks (HENDRICKS, 1997), descrito na seção 2.5.1. Hendrincks também propõe um modelo com três subsistemas principais (ignição, injeção e admissão). O modelo simplificado da dinâmica do fluxo de injeção de combustível, proposto por Hendricks, mantém o controle do fluxo de combustível (em vez da massa de combustível) no coletor de admissão, assim como a UCM desenvolvida neste projeto. Por estes motivos e por ser facilmente adaptado para outros motores, conforme afirma Hendricks, foi

adotada esta teoria para desenvolver todo sistema de controle da injeção de combustível.

No algoritmo implementado, foi utilizada a equação dos gases ideais para aproximar a relação entre a densidade do ar, a pressão do ar e a temperatura do ar no coletor de admissão. Baseando-se na equação 05, é possível deduzir que a densidade do ar admitido é proporcional à pressão absoluta do ar dentro do coletor de admissão e inversamente proporcional à temperatura do ar. Desta mesma equação, é possível determinar a massa de combustível necessária para o regime de operação do motor. Nos motores equipados com injeção eletrônica de combustível, esta massa de combustível é inserida para dentro de cada cilindro quando as válvulas injetoras de combustível são abertas pela unidade controladora. Diferentes motores possuem diferentes números de cilindros e diferentes números de injetores. Levando isto em conta e tendo em mente que o único parâmetro de controle da UCM é o tempo que a válvula injetora permanece aberta, a equação 31 relaciona a massa de combustível com o número de injetores e cilindros, além de considerar o fluxo de combustível da válvula injetora, que é um parâmetro físico de construção de cada válvula injetora e não há como ser alterado.

$$t_{inj} = \frac{m_{combustivel} \cdot N_{cilindros}}{V_{fluxo_injetor} \cdot N_{injetores}} \quad \text{Eq.31}$$

Substituindo a equação 31 na equação 05, é possível chegar à equação que determina o tempo de injeção em função da massa de combustível seguindo a estratégia *speedy-density*.

$$T_{inj} = \frac{\rho_0 \left(\frac{P_{MAP}}{P_0} \right) \left(\frac{T_0}{T_{MAP}} \right) N_{cilindros} V_{pistao} E_V}{V_{fluxo_injetor} N_{injetores} (A/C)} \quad \text{Eq.32}$$

onde:

T_{inj} *Tempo de permanência da válvula injetora aberta [μs]*

ρ_0 Densidade do ar atmosférico a nível do mar = 1.299 $\left[\frac{kg}{m^3} \right]$

P_0 Pressão do ar atmosférico a nível do mar = 101.3 [KPa]

T_0	Temperatura ambiente = 288 [Kelvin]
P_{MAP}	Pressão Absoluta do ar no coletor de admissão [KPa]
T_{MAP}	Temperatura do ar no coletor de admissão [Kelvin]
$N_{cilindros}$	Número de cilindros do motor = 4 [*]
$V_{pistão}$	Volume do pistão do motor = 0.496 [m^3]
E_v	Eficiência volumétrica [*]
$V_{fluxo_injetor}$	Fluxo dos injetores do motor = 2.948 $\left[\frac{\mu g}{\mu s} \right]$
$N_{injetores}$	Número de injetores do motor = 4 [*]
$\left(\frac{A}{C}\right)$	Fator de lambda estequimétrico para o combustível utilizado = 13.8 [*]

A equação 32 também pode ser encontrada em (Kjellqvist, 2005). Quase todos os parâmetros desta equação são constantes e dependem de fatores construtivos do motor, os únicos parâmetros que variam de acordo com o estado atual do motor são a pressão absoluta do ar admitido, a temperatura do ar admitido e a eficiência volumétrica. A eficiência volumétrica é definida como a massa de mistura ambiente que entra no cilindro no tempo de admissão dividida pela massa de mistura que vai preencher o espaço disponível na câmara, considerando-se a densidade do ar na admissão (CARVALHO, 2011). A eficiência volumétrica pode ser expressa como:

$$E_v = \frac{E_{real}}{E_{teórica}} \quad \text{Eq.33}$$

$$E_{real} = V_t \left(\frac{P_{MAP}}{R \cdot T_{MAP}} \right) \left(\frac{RPM}{2} \right) \quad \text{Eq.34}$$

$$E_{teórica} = V_t \left(\frac{RPM_{max}}{2} \right) E_{vmax} \quad \text{Eq.35}$$

onde:

V_t	Volume total do motor [m ³]
Ev_{max}	Eficiência volumétrica máxima [*]
P_{MAP}	Pressão Absoluta do ar no coletor de admissão [KPa]
T_{MAP}	Temperatura do ar no coletor de admissão [°C]
RPM	Rotações por minuto do motor [min ⁻¹]
RPM_{max}	Rotação máxima do motor [min ⁻¹]
V_t	Volume total do motor [m ³]
R	Constante universal dos gases = 8.134 $\left[\frac{\text{J}}{\text{mol} \cdot \text{°C}} \right]$

Em motores de combustão interna ciclo Otto, o efeito desejado é o aumento da eficiência volumétrica. Ela tem um efeito direto na potência de saída do motor, na medida em que a massa de ar no cilindro determina a quantidade de combustível que pode injetada num determinado instante de operação do motor. A melhor seleção dos tempos de abertura e fechamento das válvulas de admissão e escapamento de cada projeto é fundamental para maximizar-se a eficiência volumétrica. Para rotações inferiores a 2000 RPM, observa-se que menores ângulos de fechamento da válvula melhoram a eficiência volumétrica. No entanto, conforme a faixa de rotação aumenta, o atraso no fechamento da válvula de admissão melhora a relação da eficiência volumétrica (CARVALHO, 2011). Nos motores da família EA 113, utilizados neste projeto, o diagrama de válvula é constante e característico do projeto de desenvolvimento, não tendo como ser alterado durante o regime de operação do motor.

Para implementação do sistema de controle da injeção de combustível na UCM, desenvolveu-se uma função de cálculo do tempo de injeção conforme ilustra a figura 60.

Figura 60 - Função de Cálculo do Tempo de Injeção.

```

void CalculoTempoInjecao(uint8_t *Tinj) {

    float Ev = 0;           //Variavel p/ calculo da eficiencia volumetrica
    uint16_t Tus = 0;        //Variavel p/ calculo do tempo de injecao (us)
    uint8_t MAP = 0;         //Variavel de aquisicao do valor de MAP
    uint8_t RPM = 0;         //Variavel de aquisicao do valor de RPM
    uint8_t TAR = 0;         //Variavel de aquisicao do valor de TAR
    uint8_t Tinjecao = 0;    //Variavel de saida do Tempo de injecao

    MAP = RTOS_VEM [MAP_VEM];      //Recebe do RTOS pressao em [kPa/10]
    RPM = RTOS_VEM [RPM_VEM];      //Recebe do RTOS Rotacao em [RPM/100]
    TAR = RTOS_VEM [TAR_VEM];      //Recebe do RTOS Temp AR em [*C/10]

    //Calcula o Tempo de injecao em [us]
    Tus = (2.57 * ((MAP/10.13) * (288/((TAR*10)+273))) / (0.0001627296));
    //Tus = (Rho*(MAP/Po)*(To/TAR)*Ncil*Vpistao)/(Vinjetor*Ninjetores*Lambda)

    //Rho = 1.299[Kg/m3] - Densidade do ar atmosferico a nivel do mar
    //Po = (101.3/10) [kPa] - Pressao do ar atmosferico a nivel do mar
    //To = 288 [K] - Temperatura ambiente (15*C)
    //Ncil = 4 - Numero de cilindros do motor
    //Vpistao = 0.496 [m3]
    //Vinjetor = 2.948 [ug/us] - Fluxo do injetor (microgramas por us)
    //Ninjetores = 4 - Numero de injetores
    //Lambda = 13.8 - Lambda para estequiometria (gasohol brasileiro)
    /////////////////////////////////



    //Calcula Eficiencia Volumetrica do motor
    Ev = (0.001984 * ((MAP*10000)/(8.134*(TAR*10))) * ((RPM*100)/120)) / (5.952);
    //Ev = (Ereal/Eteorico)
    //Ereal = Vt * (MAP/(R*TAR)) * (RPM/120)
    //Eteorico = Vt * (RPMmax/2) * Evmax
    //Vt = 1.984 [cm3] - Volume total do motor (converter para [m3])
    //Evmax = 1 - Eficiencia volumetrica maxima
    //RPMmax = 6000 [RPM] - Rotacao maxima do motor
    //R = 8.134 [J/mol*C] - Constante do Gas
    /////////////////////////////////



    Tinjecao = Tus * Ev;          //Tempo de injecao corrigido [us/100]
    *Tinj = Tinjecao;            //Atualiza ponteiro de saida da função

    if(RPM <= 6) Tinjecao = 120;   //Caso Rotacao menor que 700RPM
    //Determina Tinj = 12ms para partida do motor
    if(Tinjecao >= 180) Tinjecao = 180; //Determina limite maximo de 18ms
    if(RTOS_VEM [REGIME_VEM] == 3) Tinjecao = 0; //Se o motor esta no regime de
    //desaceleraçao exerce CUT-OFF

    RTOS_VEM [TINJ_VEM] = Tinjecao; //Atualiza Vetor de estados
}                                //Final da Funcao

```

Fonte: o autor

Essa função é executada dentro da camada RTOS do *firmware* a cada 10ms. A função recebe os parâmetros de pressão do coletor de admissão, a rotação do motor e a temperatura do ar de admissão do vetor de estados do motor "RTOS_VEM []" e efetua o cálculo do tempo de injeção em μs , conforme equação 32. Após calcular o tempo de injeção, a função calcula a eficiência volumétrica, conforme equação 33. O tempo de injeção final é corrigido pela multiplicação entre o tempo de injeção calculado e a atual eficiência volumétrica que varia entre zero e um. O tempo de injeção final é uma representação do tempo de injeção. Por exemplo, se a função retornar o valor de 146, isto significa que o tempo de injeção calculado é de 14.6 ms, o que fornece uma resolução de 100 μs . Esta conversão foi adotada para agilizar o processamento da função, considerando que utilizar

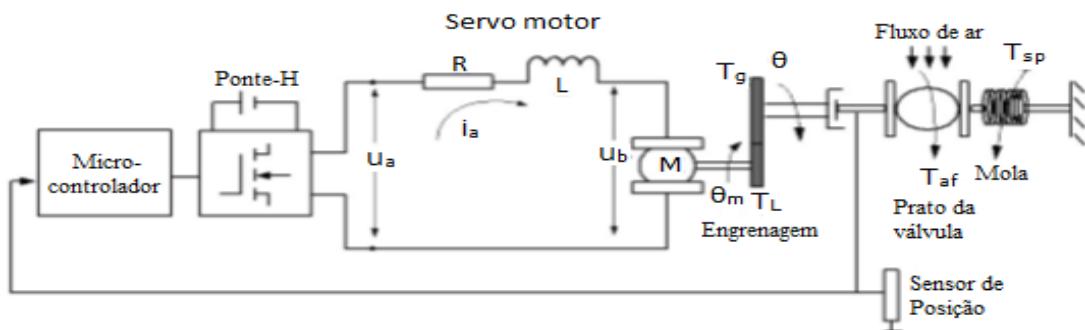
variáveis do tipo "float" atrasa em até quatro vezes a velocidade de processamento da função. O resultado final é atualizado no ponteiro de saída para ser utilizado na camada de sincronismo e o vetor de estados do motor também é atualizado para que este parâmetro possa ser utilizado por outras tarefas da camada RTOS.

4.6.3 Controle do Sistema de Admissão

A válvula borboleta é o principal componente do sistema de admissão de ar e o único atuador. A principal referência de todo sistema é a vazão do ar, geralmente determinada pelo ângulo de abertura da válvula borboleta uma vez conhecendo toda geometria do duto de admissão. Em sistemas eletrônicos, a válvula borboleta é controlada indiretamente pelo motorista. A vazão mássica está totalmente relacionada com o ângulo de abertura da válvula borboleta, pois, variando a área efetiva de passagem, o fluxo de massa de ar aumenta proporcionalmente.

A válvula borboleta é composta por um motor DC, uma mola de retorno, engrenagens de transmissão, prato da válvula e um sensor de posição angular, denominado TPS. No momento em que o pedal acelerador é acionado, um sinal é enviado para a UCM, indicando a posição do pedal e, de acordo com a sua posição e as condições de carga do motor, a UCM gera um sinal de controle PWM. Para a realização de um controle correto da válvula borboleta, é necessário desenvolver um modelo matemático da dinâmica da válvula. Esse modelo matemático, apresentado inicialmente por Chen (CHEN e colaboradores, 2012) e, posteriormente desenvolvido por Silva (THAMIRES DA SILVA, 2015) apresenta as equações e os parâmetros de uma forma bem simples, sendo possível identificar as equações a partir da ilustração do modelo da válvula borboleta, conforme a figura 61.

Figura 61 - Ilustração de Chen para o modelo da válvula borboleta.



Fonte: (Adaptado de CHEN e colaboradores, 2012).

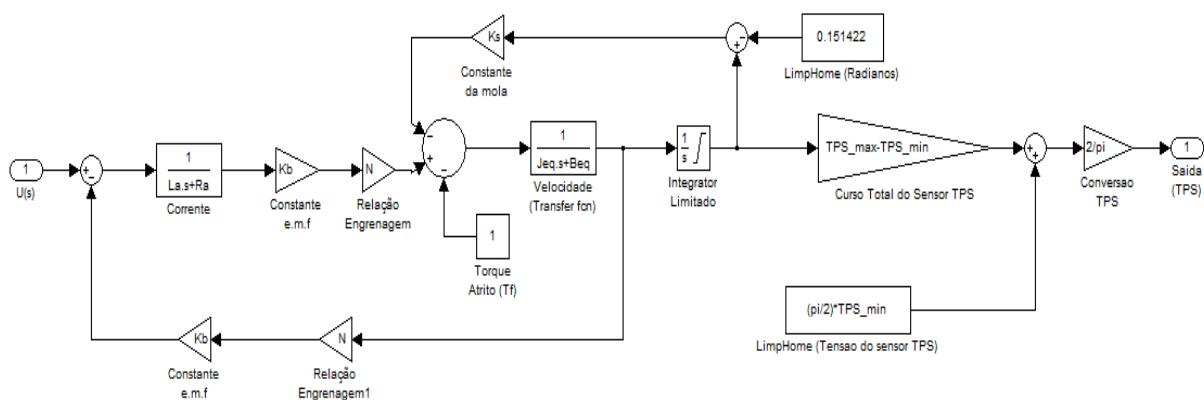
Na figura 61, u , R , i_a e L representam a tensão da armadura, resistência, corrente e indutância, respectivamente, K_b é a constante contra eletromotriz em relação à rotação do motor, θ_m é a posição angular do eixo do motor, T_m é o torque de saída do eixo do motor, T_L é o torque de carga do motor, C_m é o coeficiente de atrito viscoso do eixo do motor, C_{m0} é o coeficiente de atrito estático do eixo do motor, J_m é a inércia do motor, T_g é o torque transmitido das engrenagens, T_{af} é o torque do fluxo de ar, T_{sp} é o torque de retorno da mola, C_t é o coeficiente de atrito viscoso da borboleta, C_{t0} é o coeficiente de atrito estático da borboleta, J_g é a inércia da borboleta, θ é a posição angular do prato da borboleta, $T_{sp} = K_{sp}(\theta - \theta_0)$, θ_0 é o ângulo inicial do torque de retorno da mola, K_{sp} é o coeficiente elástico da mola.

Com as três equações do modelo matemático proposto por Chen (CHEN e colaboradores, 2012), é possível desenvolver uma única equação correspondente à dinâmica da válvula borboleta, resultando na rotação do eixo da válvula borboleta, conforme a equação 36.

$$\left(\left((U(s) - k_b \Omega_t(s) N) * \left(\frac{1}{L_s + R} \right) * k_b \right) N - c_{m0} N - T_{SP} - c_{t0} \right) * \left(\frac{1}{(J_g + N^2 J_m) s + c_t + N^2 c_m} \right) = \Omega_t(s) \quad \text{Eq.36}$$

Utilizando a equação 36, foi possível desenvolver o modelo matemático no *Simulink* do software *Matlab®*, conforme ilustra a figura 62.

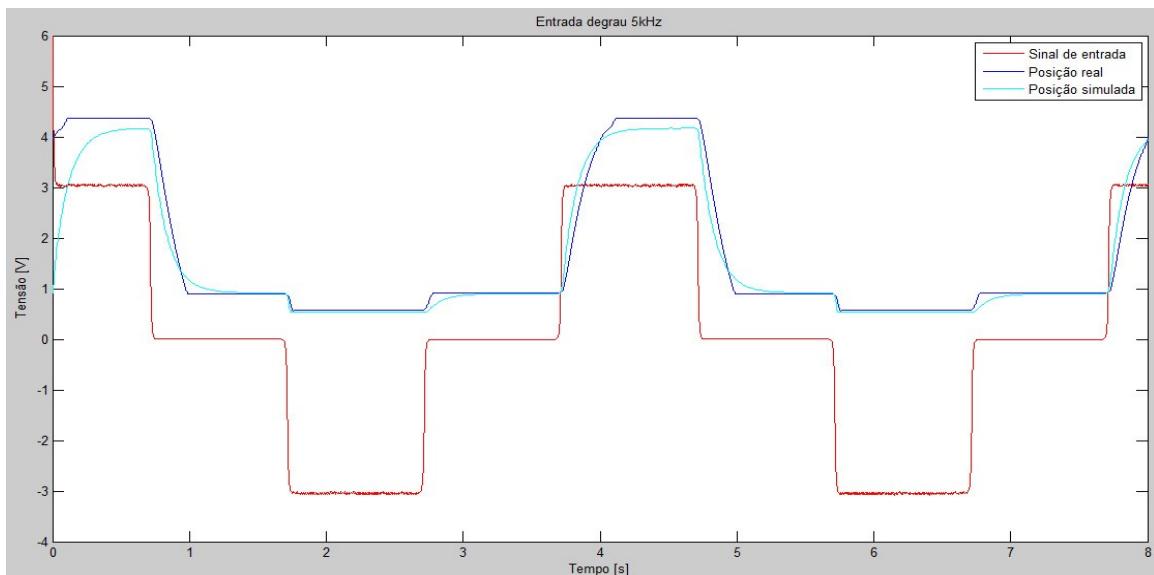
Figura 62 - Modelo da válvula borboleta em *Simulink*.



Fonte: (Adaptado de THAMIREZ DA SILVA, 2015).

Para validar este modelo, foram efetuados testes de bancada utilizando um pedal eletrônico e uma válvula borboleta idêntica ao modelo utilizado pelo motor da família EA 113. Os testes efetuados foram satisfatórios e o modelo desenvolvido se mostrou bastante confiável. Foram realizados testes com diferentes frequências de PWM, pois quanto maior a frequência, menor é o tempo de abertura da válvula borboleta. Neste projeto, adotou-se a frequência de 2KHz para se controlar a válvula borboleta. A figura 63 mostra o sinal da tensão correspondente ao sensor de posição da válvula borboleta (TPS), resultante da aplicação do degrau de acionamento de 3V, juntamente com o sinal resultante da simulação.

Figura 63 - Sinal de entrada com degrau de 3V, posição real e simulada do modelo proposto.

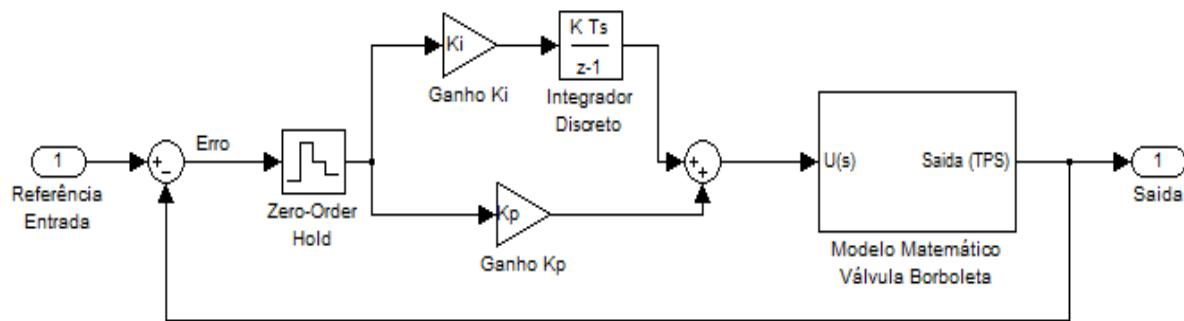


Fonte: (THAMIREZ DA SILVA, 2015).

A figura 63 ilustra um comportamento satisfatório do modelo matemático em malha aberta, ou seja, sem controle. Desta forma, foi validado o modelo matemático da válvula borboleta, obtendo uma boa base para que o controle seja desenvolvido. A partir do modelo matemático, foi calculada a equação de estados da válvula borboleta, permitindo adquirir uma função de transferência, fazendo com que o controle pudesse ser realizado. Neste projeto, o controle de posição da válvula borboleta foi implementado com um controlador PI (proporcional e integral), sendo que a referência de posição para este controlador é determinada pelo controle da rotação do motor que será descrito nas próximas seções. Desta forma, a função do controlador PI, implementado para esta válvula borboleta, é garantir que a posição da válvula seja aquela determinada pelo controle de rotação.

Testes no motor e análises ao sistema de controle original revelaram que não é necessário um controle rápido, ou seja, a válvula borboleta não precisa abrir ou fechar rapidamente, ou seja, um controle estável é essencial para o sucesso do controle de rotação. Pequenas alterações na posição angular da válvula borboleta já são suficientes para causar variações na rotação do motor, principalmente em marcha lenta, o que pode resultar numa instabilidade do controle da rotação do motor. O controlador de posição desenvolvido para este projeto é ilustrado na figura 64.

Figura 64 - Controlador PI de posição da válvula borboleta.



Fonte: o autor

A implementação do controlador PI é feita de forma discreta. O microcontrolador utiliza o módulo PWM do periférico FMT1 que disponibiliza um sinal PWM com uma frequência de 2KHz. A figura 65 ilustra a função utilizada para implementar o controlador.

Figura 65 - Função de sincronismo do sistema de admissão.

```

void SincronismoSistemaADM(void) {
    static short int Erro;
    static short int Errol;
    unsigned short int Pwm;
    short int Saida;
    uint8_t Tps;
    uint8_t Referencia; //Referencia de posicao para Borboleta

    //Variaveis utilizadas na funcao que nao necessitam ser declaradas
    //ErroMaximo = 220
    //SaidaMaxima = 220 (Final de curso da Borboleta)
    //Resolucao ADC = 8bits (255)
    //Kp = 12
    //Ki = 0.07

    Tps = SensorTPS2(); //Captura novo valor de Tps
    Referencia = RTOS_VEM [REF_VB_VEM]; //Captura referencia atual para VB
    RTOS_VEM [TPS2_VEM] = Tps; //Atualiza Vetor de estados

    Erro = Referencia - Tps; //Calcula Erro do controlador PI

    //Limita Erro em funcao dos valores de ErroMaximo
    if (Erro >= 220) Erro = 220;
    if (Erro <= (-220)) Erro = (-220);

    //Calcula nova saida
    Saida = (12 * Erro) + (0.07 * (Erro + Errol)); //Calculo do PI com (n-1)
    // (Kp * Erro) + (Ki * (Erro + Errol))

    //Atualiza Variaveis
    Errol = Erro;

    //Limita saida em funcao dos valores de ErroMaximo
    if (Saida > 220) Saida = 220;
    if (Saida < (-220)) Saida = (-220);

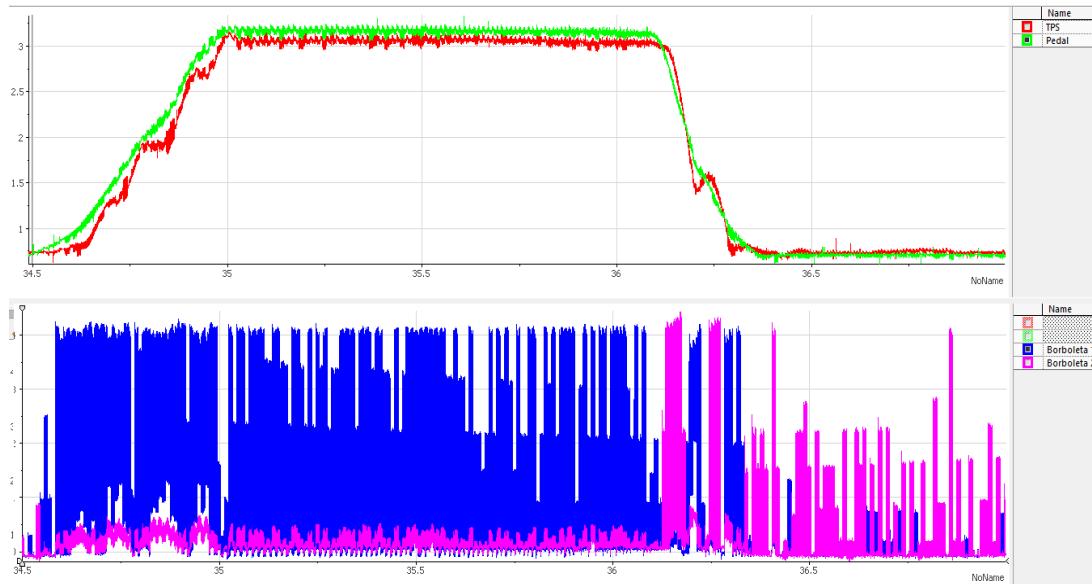
    //Atualiza valor da nova saida para atualizar registrador do PWMs
    if (Saida < 0) { //Executa de saida eh menor que zero (Erro negativo)
        Pwm = ((12500 * (Saida * (-1))) / 220); //Converte PWM dentro do limite do
        //registrador (12500) 2Khz
        // Controla Valvula Borboleta atualizando PWMs
        FTM1_COV = 0; //PWM 0%
        FTM1_C1V = Pwm; //PWM de acordo com o valor do calculo PID
    }
    else { //Executa de saida eh maior que zero (Erro positivo)
        Pwm = (12500 * Saida) / 220; //Converte PWM dentro do limite do
        //registrador (12500) 2Khz
        // Controla Valvula Borboleta atualizando PWMs
        FTM1_COV = Pwm; //PWM de acordo com o valor do calculo PID
        FTM1_C1V = 0; //PWM 0%
    }
} //Final da funcao

```

Fonte: o autor

Esta função é executada a cada 10ms pela camada RTOS. Para obter os ganhos K_p e K_i , foi adotada uma metodologia ajustável, baseada em testes de bancada e no próprio veículo. A carga imposta pela aspiração do motor e a instabilidade do motor na condição de marcha lenta fez com que o ganho proporcional fosse relativamente elevado. Escolheu-se um ganho proporcional elevado para que o controlador fosse capaz de atingir sua referência no menor tempo possível. É importante mencionar que o tratamento dos sensores envolvidos (pedal de aceleração e TPS) é essencial para o sucesso do controlador de posição implementado, dado que ruídos presentes no sistema degradariam a estabilidade do PI. A figura 66 ilustra o resultado final obtido com o controlador PI desenvolvido.

Figura 66 - Resultados obtidos com o controlador PI implementado.



Fonte: o autor

A figura 66 mostra o resultado do controlador PI implementado. Para adquirir estes resultados, foram utilizados o sistema de intersecção, o *software* de aquisição de dados desenvolvido em *LabVIEW®*, e os gráficos foram gerados no *software* *DIAdem®*. O gráfico superior mostra o sinal de referência do controlador implementado no canal 2 (Pedal, cor verde). Neste teste, o pedal estava em sua posição de repouso (completamente solto) e no instante 34.5s foi totalmente acionado, permanecendo acionado até o instante 36.2s, quando é solto, retornando a sua posição inicial. O canal 1(TPS, cor vermelha) mostra o sinal do sensor de posição angular da válvula borboleta acompanhando sua referência. Este resultado se mostrou bastante eficaz, tendo em vista a dinâmica e a carga imposta pelo motor em regime de operação. O gráfico inferior mostra o sinal PWM de saída do controlador diretamente nos dois polos do motor DC da válvula borboleta. O canal 3 (Borboleta 1, cor azul) mostra o sinal de controle PWM no polo 1 do motor DC, o canal 4 (Borboleta 2, cor rosa) mostra o sinal de controle PWM no polo 2 do motor DC. É possível observar que quando o erro do controlador é positivo, ou seja, a referência é maior que a posição atual da válvula borboleta, o controlador apenas atua no PWM do polo 1 do motor DC, fazendo com que a válvula borboleta aumente sua posição angular. Quando o erro do controlador é negativo, ou seja, a referência

é menor que a posição atual da válvula borboleta, o controlador apenas atua no PWM do polo 2 do motor DC.

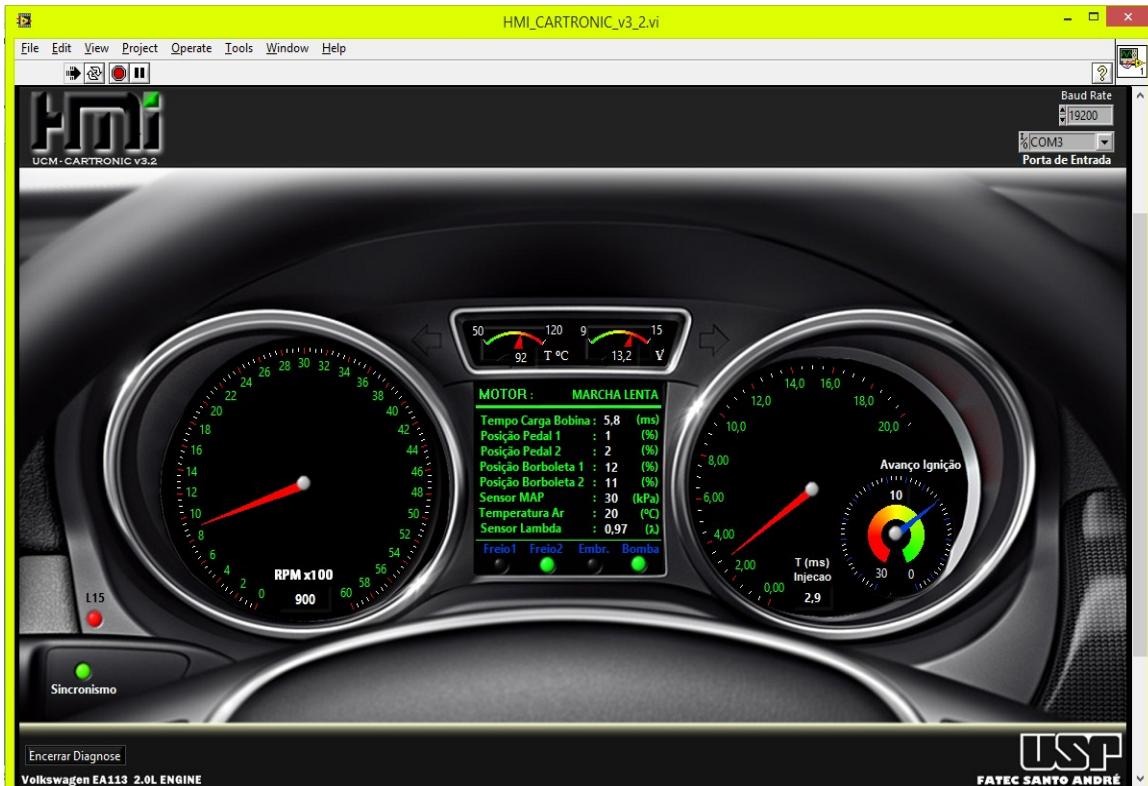
O pequeno erro de estado estacionário ocorre devido à perturbações, como os próprios atritos e molas não lineares do sistema. A dinâmica real da válvula borboleta foi comparada com o resultado da dinâmica simulada utilizando um mesmo sinal de entrada, sendo possível validar o modelo desenvolvido, e validar os parâmetros coletados. A partir do modelo matemático, foi desenvolvida a equação de estados e a função de transferência. O modelo aqui desenvolvido obteve sucesso acompanhando o sinal real da válvula borboleta com diferentes sinais de entrada.

4.7 SISTEMA DE DIAGNOSE

Como parte do trabalho foi desenvolvido um *software* de *interface* entre o usuário e a UCM. Este sistema de diagnose foi denominado HMI (*Human Machine Interface*), e seu objetivo principal é possibilitar o ajuste das estratégias de controle e a visualização do comportamento do sistema através da aquisição de dados enviados em tempo real pela UCM.

Para o sistema de diagnose (HMI) desenvolvido, foram assumidas algumas prioridades com relação às suas características construtivas. O *software* deve possuir estrutura modular, para permitir expansão posterior, deve permitir comunicação via USB e *Bluetooth®* com a UCM, garantindo integridade dos dados e a visualização dos parâmetros em tempo real, além de possuir uma *interface* gráfica simples e amigável, permitindo a integração de novas funções no futuro.

Para atingir estes objetivos e agilizar o processo de desenvolvimento do sistema de diagnose, foi adotada a linguagem "G", que é a linguagem de programação gráfica do *LabVIEW®*. Neste *software*, a programação é feita de acordo com o modelo de fluxo de dados, o que oferece a esta linguagem vantagens para a aquisição de dados e para a sua manipulação. O *LabVIEW®* é composto pelo painel frontal, que contém a *interface*, e pelo diagrama de blocos, que contém o código gráfico do programa. O programa não é processado por um interpretador, mas sim compilado. Deste modo, a sua execução é comparável à exibida pelas linguagens de programação de alto nível. A figura 67 ilustra a tela principal do sistema de diagnose, desenvolvido para possibilitar uma completa interação entre usuário e UCM.

Figura 67 - Sistema de diagnose desenvolvido (HMI)

Fonte: o autor

O sistema de diagnose permite visualizar todas as variáveis em tempo real, permite selecionar qual a porta de entrada de dados e qual a frequência de transmissão durante a execução. Para se comunicar com a UCM, o sistema de diagnose utiliza um bloco de dados padronizado de 19 parâmetros do motor, além de um parâmetro de sincronismo. A UCM envia um parâmetro novo a cada 10ms, conforme execução da camada RTOS. Estes parâmetros foram determinados como sendo os dezenove primeiros parâmetros do vetor de estados do motor "RTOS_VEM". O vigésimo parâmetro é o de sincronismo, que foi definido como 0xFF. A tarefa desenvolvida para enviar os parâmetros para o sistema de diagnose é apresentada na figura 68.

Figura 68 - Tarefa desenvolvida para enviar os dados ao sistema de diagnose

```
void HMI (void) {
    static uint8_t i = 0;           //Variavel indexadora da tarefa

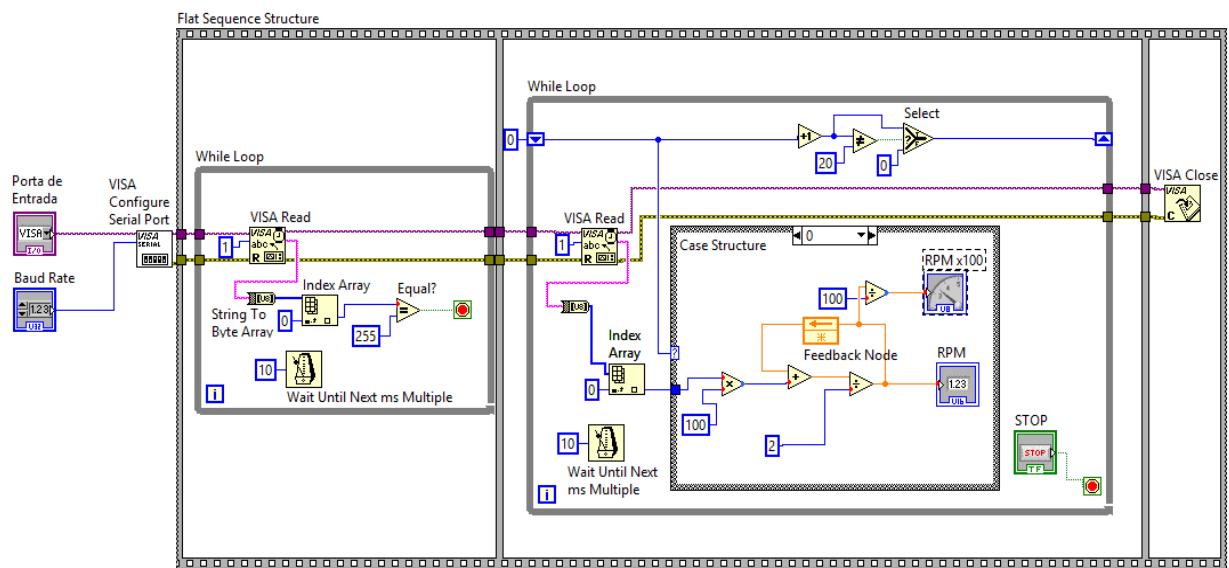
    if(i == 19) EnviaDadoUSB1(0xFF);      //Envia Byte de sincronismo
    else EnviaDadoUSB1(RTOS_VEM[i]);     //Envia dados indexados por "i"

    i++;
    if (i == 20) i = 0;                //Caso i = 20, resetamos a funcao zerando i
}
```

Fonte: o autor

No diagrama de blocos do *LabVIEW®*, os blocos de funções são designados por instrumentos virtuais. Isto porque, em princípio, cada programa (Sub-IV) pode ser usado como subprograma por qualquer outro ou pode, simplesmente, ser executado isoladamente. A lógica da programação do diagrama de blocos do *LabVIEW®* é apresentada na figura 69.

Figura 69 - Lógica de programação do diagrama de blocos do HMI



Fonte: o autor

A execução do diagrama de blocos é dependente do fluxo de dados, ou seja, o diagrama não executa da esquerda para a direita e o um bloco somente executa sua função quando todos os dados estiverem disponíveis nos terminais de entrada do bloco. Por este motivo, foi utilizado o laço "*Flat Sequence Structure*", que permite definir ordens para a execução dos blocos. Inicialmente, este programa deve abrir os *plugins* de comunicação com a respectiva porta de entrada de dados do computador. Com a comunicação estabelecida, o programa entra em um "*while loop*" até receber o dado de sincronismo da UCM. Após receber o dado de sincronismo, a função entra em seu "*while loop*" principal, e neste laço o programa aguarda receber um novo dado para indexar o laço "*Case Structure*". Na figura 69, é possível visualizar o tratamento efetuado para converter o dado relativo à rotação do motor, porém o laço "*Case Structure*" possui vinte camadas, cada camada efetua a conversão de um parâmetro para ser visualizado na interface gráfica do programa. A

cada novo dado que o programa recebe, o laço "Case Structure" incrementa sua camada de aplicação. O programa somente se encerrará quando o usuário pressionar o botão "STOP" para que o fluxo de dados do programa permita que a comunicação entre o sistema de diagnose e a UCM seja finalizada.

O sistema de diagnose desenvolvido se mostrou bastante eficaz, permitindo verificar em tempo real os principais parâmetros do motor. Esta ação facilitou os desenvolvimentos, as tarefas e os testes durante toda a elaboração deste projeto. A estrutura do programa foi baseada na arquitetura Documento/Visão, que é uma versão simplificada da MVC (Modelo/Visão/Controle). Esta arquitetura proporciona a separação entre o código que gerencia os dados do programa e o código necessário para apresentá-los ao usuário.

4.8 OBTENÇÃO DO MODELO

Devido a fenômenos complexos da dinâmica do motor, que fazem com que o sistema apresente variações entre ciclos mesmo quando todos os parâmetros controláveis sejam mantidos constantes, as ações de controle no motor devem proporcionar boa resposta dinâmica às perturbações de vazão mássica de ar e rotação, compensando os seus efeitos. O desempenho destas ações de controle depende de calibrações de mapas, tabelas e parâmetros, e tende a se deteriorar com o desgaste do sistema. Deste modo, independentemente do tipo de controlador utilizado, a realimentação torna-se indispensável para garantir que a rotação permaneça próxima da referência imposta pelo motorista.

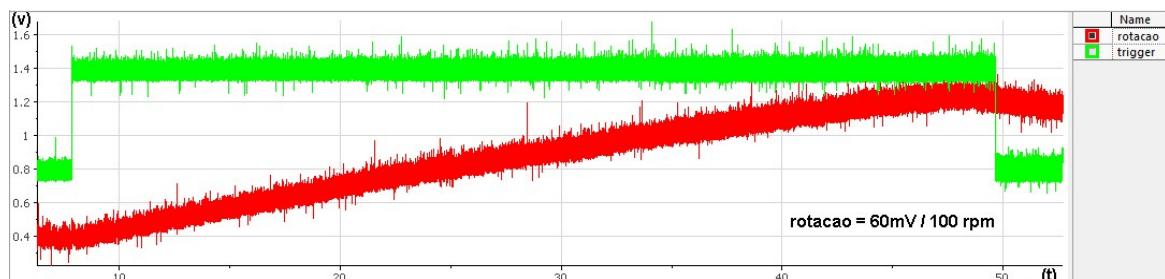
Neste trabalho, foi implementado um controlador proporcional integral seguindo o modelo linear de controle proporcional integral proposto por Braga (BRAGA, 2007). Para obter o modelo, foram efetuados ensaios para obtenção do sinal de resposta do sistema, utilizando uma abordagem clássica baseada em um modelo linear que caracteriza a relação entre a posição angular da válvula borboleta e a rotação do motor. Apesar da dinâmica não linear dos motores ciclo Otto, é possível utilizar um modelo linear para descrever as condições de operação do motor.

Para a determinação do modelo, foram realizados testes de resposta ao degrau da rotação do motor em função da posição angular da válvula borboleta. A condição de operação adequada para a obtenção do modelo requer uma aplicação de carga no motor. Para isto, todos os testes foram efetuados com o veículo fixo em um

dinamômetro inercial e em terceira marcha. Além do dinamômetro, para efetuar o levantamento do modelo, foi necessário o sistema de interseção e o software de aquisição de dados desenvolvido em *LabVIEW®*. Adicionalmente, foi necessário estar com todas as funções da camada de sincronismo implementadas e extremamente confiáveis no *firmware* desenvolvido na UCM.

Para obter um modelo aproximado da influência da variação angular da válvula borboleta na rotação do motor, os sistemas de injeção e ignição devem permanecer constantes. As condições iniciais do teste exigem que o veículo seja controlado de forma própria. Ao se estabilizar a rotação do motor em 1000 RPM com o veículo em terceira marcha, sobre a carga imposta pelo dinamômetro, com o tempo de injeção fixado em 10ms, com o avanço de ignição fixado em 10 graus e com a válvula borboleta posicionada em sua posição angular inicial (*Limp Home*), um degrau de 20 graus na posição angular da válvula borboleta é aplicado para iniciar o teste. A figura 70 ilustra o resultado obtido com o teste.

Figura 70 - Resposta da rotação ao degrau da válvula borboleta



Fonte: o autor

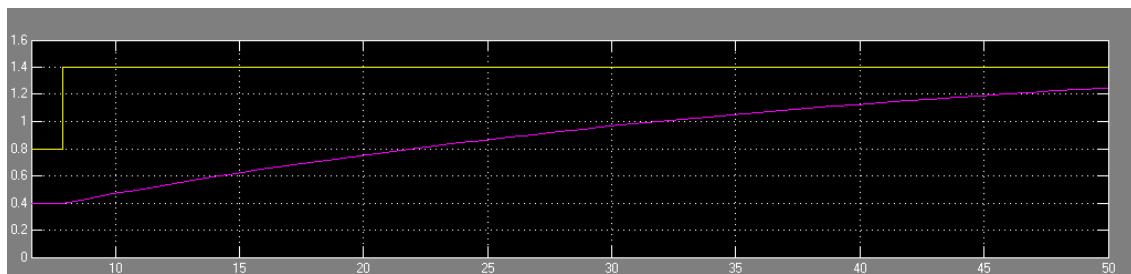
Na figura 70, o eixo horizontal representa o tempo em segundos e o eixo vertical representa a tensão em volts. Para conseguir medir a rotação de uma forma analógica, foi utilizado o DAC (*Digital Analogic Converter*) do microcontrolador. Este periférico transforma valores digitais de 12 bits dentro da memória de programa do microcontrolador em uma tensão analógica disponível em um pino de saída do mesmo. A resolução definida para esta conversão foi de 60mV/100RPM, ou seja, a cada 100 RPM do motor a tensão no pino de saída do DAC aumentará 60mV. O canal 1 (rotação, vermelho) mostra a resposta do sinal de rotação, o canal 2 (trigger, verde) mostra o sinal de gatilho do degrau, indicando quando o degrau foi aplicado. Nestas condições, é possível observar que o motor demora aproximadamente 40 segundos para se estabilizar a 5000 RPM. Os resultados mostram que, na faixa de

operação considerada, o sistema apresenta comportamento de primeira ordem com tempo morto. Com os valores obtidos neste teste, foi estabelecida a função de transferência do modelo em função do tempo de acomodação, dos valores de pico e constantes de tempo, conforme as equações de controle clássico (OGATA, 1985). O modelo obtido é apresentado como:

$$G(s) = \frac{K}{\tau.s+1} \cdot e^{-\theta s} = \frac{1,55}{35,2.s+1} \cdot e^{-0,08s} \quad \text{Eq.37}$$

Este modelo pode ser aplicado em casos onde as condições de carga, temperatura e pressão do ar admitido pelo motor forem iguais às condições do teste efetuado. Com a função de transferência concluída, foi simulado o resultado do degrau em malha aberta pelo *Matlab®*, e o resultado é ilustrado na figura 71.

Figura 71 - Simulação do modelo obtido em *Matlab®*,



Fonte: o autor

4.9 DESENVOLVIMENTO DO CONTROLADOR DE ROTAÇÃO

Com base no modelo obtido, foi projetado um controlador PI para a rotação do motor, utilizando o método da síntese direta. Como o motor apresenta um tempo morto elevado, a escolha da constante de tempo em malha fechada deve ser conservativa. Se optasse por utilizar ganhos elevados, o motor poderia apresentar instabilidades, principalmente em condições de marcha lenta. Portanto, optou-se por utilizar uma constante de tempo em malha fechada de 0,2 segundos. As equações 38 e 39 apresentam os cálculos efetuados para determinar o ganho proporcional (K_p) e o ganho integral (K_i) do controlador de rotação implementado.

$$Kp = \frac{1}{K} \cdot \frac{\tau}{\theta + \tau_f} = \frac{35,2}{1,55 \cdot (0,08 + 0,2)} = 81,1 \quad \text{Eq.38}$$

$$Ki = \frac{Kp}{\tau} = \frac{81,1}{35,2} = 2,3 \quad \text{Eq.39}$$

Estes ganhos somente podem ser aplicados caso o motor esteja em terceira marcha e com a carga semelhante à carga aplicada pelo dinamômetro inercial (2500Kg). O motor apresenta variações em seu modelo de acordo com a variação da carga imposta ao motor. Deste modo, este modelo somente se aplica nesta condição específica.

A ideia aplicada consiste em um controlador PI gerando a ação de controle para o modelo dado pela função de transferência apresentada na equação 37. O controlador tem como parâmetro de entrada a referência de rotação calculada a partir da leitura do pedal. O cálculo para determinar a referência de rotação é linear, ou seja, se o pedal estiver totalmente solto, a referência de rotação deverá ser 800 RPM, se o pedal for totalmente acionado, a rotação de referência deverá ser de 6000 RPM. A partir do erro existente entre as rotações de referência e atual, o controlador executa ações proporcionais e integrais para que seu parâmetro de saída ajuste a referência da posição angular da válvula borboleta. Este é o parâmetro de entrada do controlador de posição da válvula borboleta. Como todos os controladores desenvolvidos neste trabalho, o controlador de rotação é discreto e a tarefa desenvolvida na camada RTOS para implementar este controlador é ilustrada na figura 72.

Figura 72 - Implementação do controlador de rotação

```

void PID_RPM (void) {
    static short int Erro;           //Variavel para obtenção do erro
    static short int Erro1;          //Memoria para Erro anterior
    unsigned short int Referencia; //Variavel de saida da função
    short int Saida;               //Variavel para calculo da função

    //Variaveis utilizadas na função que não necessitam ser declaradas
    //Kp = 81.1
    //Ki = 2.3

    Erro = RTOS_VEM [REF_RPM_VEM] - RTOS_VEM [RPM_VEM];
    //Erro = Referencia Rotacao - Rotacao Atual

    //Calcula nova saida
    Saida = (81.1 * Erro) + (2.3 * (Erro + Erro1)); //Calculo do PI com (n-1)
    //((Kp * Erro) + (Ki * (Erro + Erro1)))

    //Limita saida em função do curso total da válvula borboleta
    if (Saida > 220) Saida = 220;
    if (Saida < (-220)) Saida= (-220);

    //Atualiza Variaveis
    Erro1 = Erro;

    //Atualiza valor da nova saida para atualizar referencia borboleta
    Referencia = ((220 * Saida) / 220) + 32;

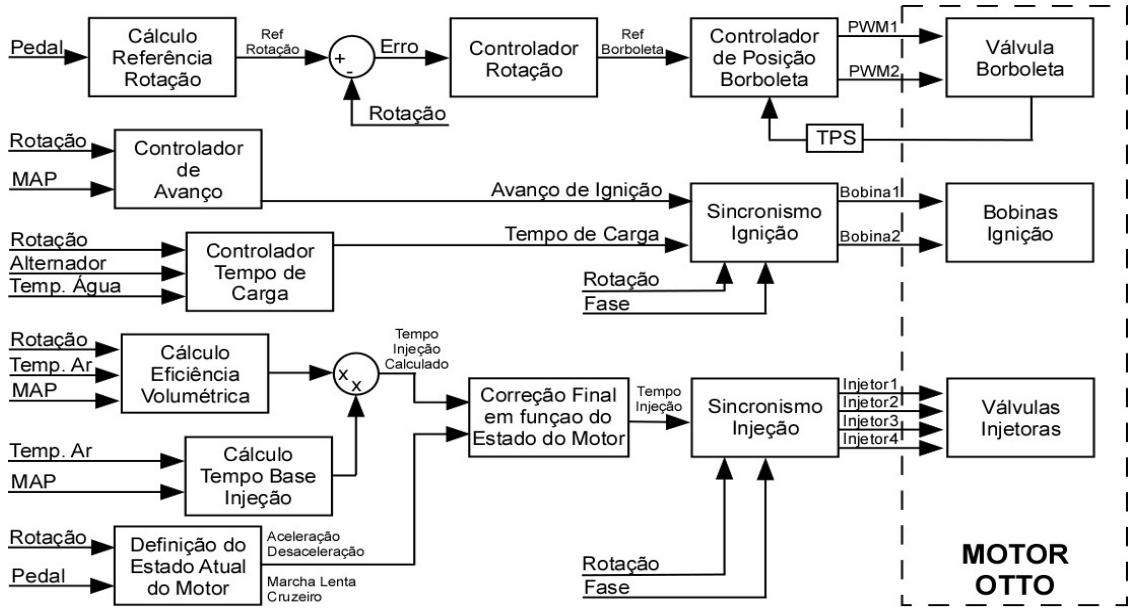
    RTOS_VEM [REF_VB_VEM] = Referencia;      //Atualiza a referencia para borboleta
} //Final da Função

```

Fonte: o autor

Esta tarefa possui como parâmetro de saída a referência calculada para a posição angular da válvula borboleta. Este parâmetro de saída é o parâmetro de entrada da função do controlador de posição da válvula borboleta. O ajuste deste controlador é delicado, e isto ocorre devido à dinâmica do motor. Variações muito rápidas na posição angular da válvula borboleta ocasionam uma perca considerável da eficiência volumétrica do motor, causando o efeito conhecido como "afogamento". Este efeito é indesejado nos motores ciclo Otto e pode, em muitos dos casos, interromper o funcionamento do motor. A visão completa do controlador desenvolvido para atuar no motor deste trabalho é ilustrada na figura 73.

Figura 73 - Modelo de controle completo do motor ciclo Otto



Fonte: o autor

É importante ressaltar que este modelo corresponde a uma aproximação bastante simplificada da dinâmica do motor e ilustra de maneira completa a teoria de controle aplicado para controlar o motor deste trabalho. Como este motor possui um sistema eletrônico de controle da válvula borboleta, o motorista não atua diretamente na abertura da válvula borboleta, o pedal passa a ser a referência principal para o sistema de controle. A partir do pedal, o sistema determina qual é a rotação que o motorista deseja. O controlador de rotação avalia o erro entre as rotações de referência e atual do motor e determina a posição ideal para a válvula borboleta. A posição de referência é o parâmetro de entrada da função de controle da posição da válvula borboleta. O sistema de ignição é calculado com base em mapas previamente armazenados e o tempo de carga da bobina é calculado com base na tensão do alternador e na temperatura do líquido de arrefecimento do motor. O sistema de injeção é calculado com base na densidade do ar admitido, na eficiência volumétrica do motor e corrigido em função do estado atual do motor. Esta correção aumenta o tempo de injeção em condições de aceleração e zera o tempo de injeção em situações de desaceleração (*cut-off*). Todos os parâmetros calculados são transmitidos para a camada de sincronismo que se encarrega de sincronizar o acionamento das válvulas injetoras e bobinas de ignição com o motor em função do sinal do sensor de rotação do motor e o sinal do sensor de fase.

Algumas considerações devem ser feitas com relação ao ajuste dos ganhos deste controlador. Como o desempenho dinâmico do controlador é prejudicado pelos erros de medição por parte do microcontrolador, pelo elevado tempo morto e pela característica não linear do motor, a definição destes ganhos deve ser conservadora para garantir um desempenho satisfatório do motor. A rejeição a perturbações são compensadas pelas ações diretas dos controladores PI e o sistema apresenta maiores dificuldades para atuar no motor na condição de marcha lenta.

Finalmente, o controlador aplicado neste trabalho se mostrou eficaz em condições de cruzeiro, plena carga e desaceleração. Em marcha lenta, o sistema se comportou de maneira satisfatória com pequenas oscilações entre 800 RPM e 900 RPM, embora testes efetuados com a referência de marcha lenta em 1000 RPM se mostraram mais estáveis. Isto ocorre devido às perdas mecânicas do motor, que exercem maior influência sobre o controlador quanto menor for a rotação do motor. Em condições de acelerações abruptas, o controlador se mostrou ineficiente sem uma correção no percentual da massa de combustível injetada para dentro do cilindro. Por este motivo, foi adicionado o bloco "Correção final em função do estado atual do motor". Este bloco corrige em até 50% o valor do tempo de injeção calculado se o motorista pressionar o pedal do acelerador de forma abrupta. Deve-se lembrar que o tempo máximo de injeção estabelecido para este trabalho é de 18ms. A próxima seção ilustra os resultados obtidos nos testes finais deste trabalho.

4.10 TESTES FINAIS E VALIDAÇÃO DO TRABALHO

Para realizar os testes finais na unidade microcontroladora, para o gerenciamento eletrônico de um motor de combustão interna ciclo Otto desenvolvida neste trabalho, foi necessário efetuar a instrumentação de todos os sensores do veículo e acelerá-lo em um dinamômetro inercial. Para adquirir os dados dos testes finais, foi utilizado o sistema de interseção e o computador equipado com a placa de aquisição de dados, conforme mencionado na seção 4.2. O dinamômetro inercial utilizado nos testes é denominado *Dynotech 7201* e os cilindros principais deste dinamômetro possuem uma massa conjunta de aproximadamente 2500 Kg. Estes equipamentos possibilitaram realizar uma série de testes para validar toda a teoria aplicada neste trabalho. Foi possível obter resultados com o motor no momento de partida, em marcha lenta, em plena carga e desaceleração, utilizando a estratégia

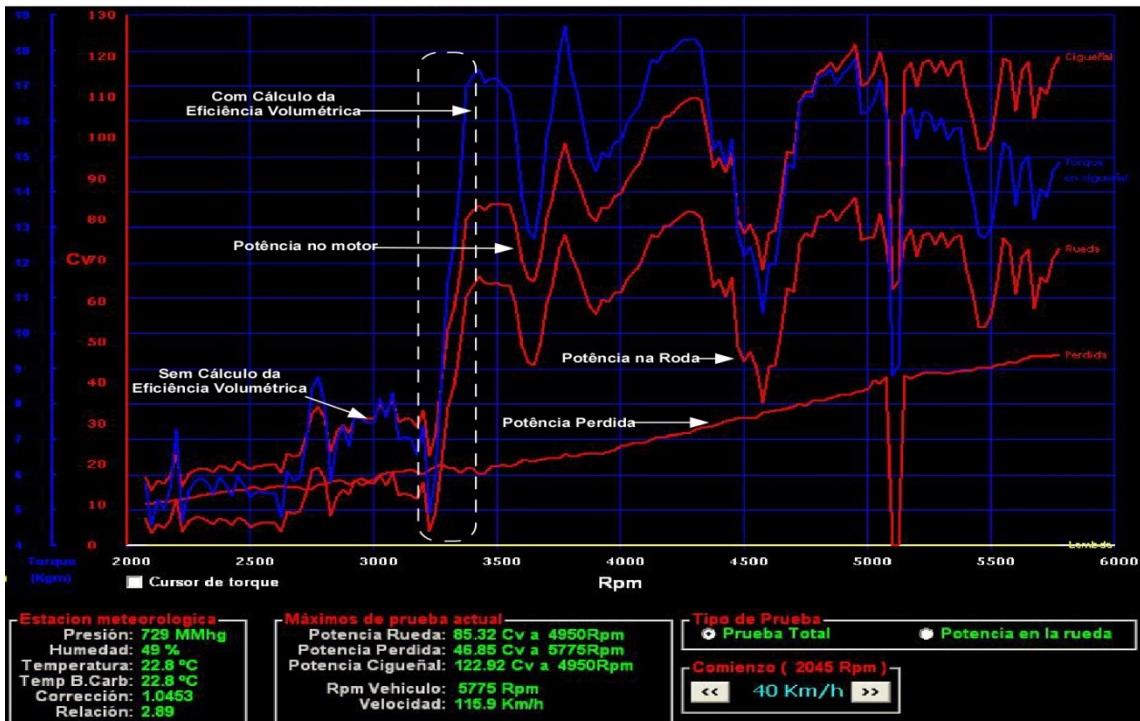
de *cut-off*. Além disso, foi possível validar todas as funções do sistema de sincronismo e medir a potência real existente nas rodas do veículo. Esta seção ilustra os resultados obtidos nos testes finais, que permitiram validar a teoria de controle do motor aplicado neste trabalho.

4.10.1 Teste de potência e validação da teoria de controle aplicada

Para realizar o teste de potência, foi utilizado o *software Dynercial 2*, desenvolvido pela empresa fornecedora do dinamômetro. Primeiramente, se calibra o dinamômetro ajustando o *software* a referência do veículo. A calibração é executada com o motor em 2000 RPM e o veículo em terceira marcha, que foi escolhido por apresentar a relação de transmissão mais próxima de 1. Após calibrar o dinamômetro, o rolo deve ser totalmente paralisado para que o *software* inicie o teste. O teste se inicia quando o veículo ultrapassa 2000 RPM em terceira marcha. O pedal do acelerador deve permanecer totalmente pressionado até que o motor atinja aproximadamente 6000 RPM. Nesse instante, o pedal deve permanecer solto e o motor deve desacelerar naturalmente. Isto deve ser efetuado para o *software* do dinamômetro calcular as perdas mecânicas existentes entre a roda do veículo e o eixo de saída do motor, através da verificação da inércia do sistema. Quando o motor atingir uma rotação inferior a 2000 RPM, o teste é encerrado e os valores são armazenados em gráficos.

O primeiro teste efetuado teve como objetivo validar o sistema de sincronismo e a influência do cálculo da eficiência volumétrica no sistema de injeção. Para isto, uma pequena alteração no *firmware* gravado no microcontrolador foi efetuada. Esta alteração obrigou o *firmware* a calcular a eficiência volumétrica do motor e corrigir o tempo de injeção somente quando a rotação do motor fosse superior a 3000 RPM. Nesse teste, ainda não havia sido implementado o controlador de rotação. O resultado deste primeiro teste é ilustrado na figura 74.

Figura 74 - Primeiro teste de potência para validação da eficiência volumétrica



Fonte: o autor

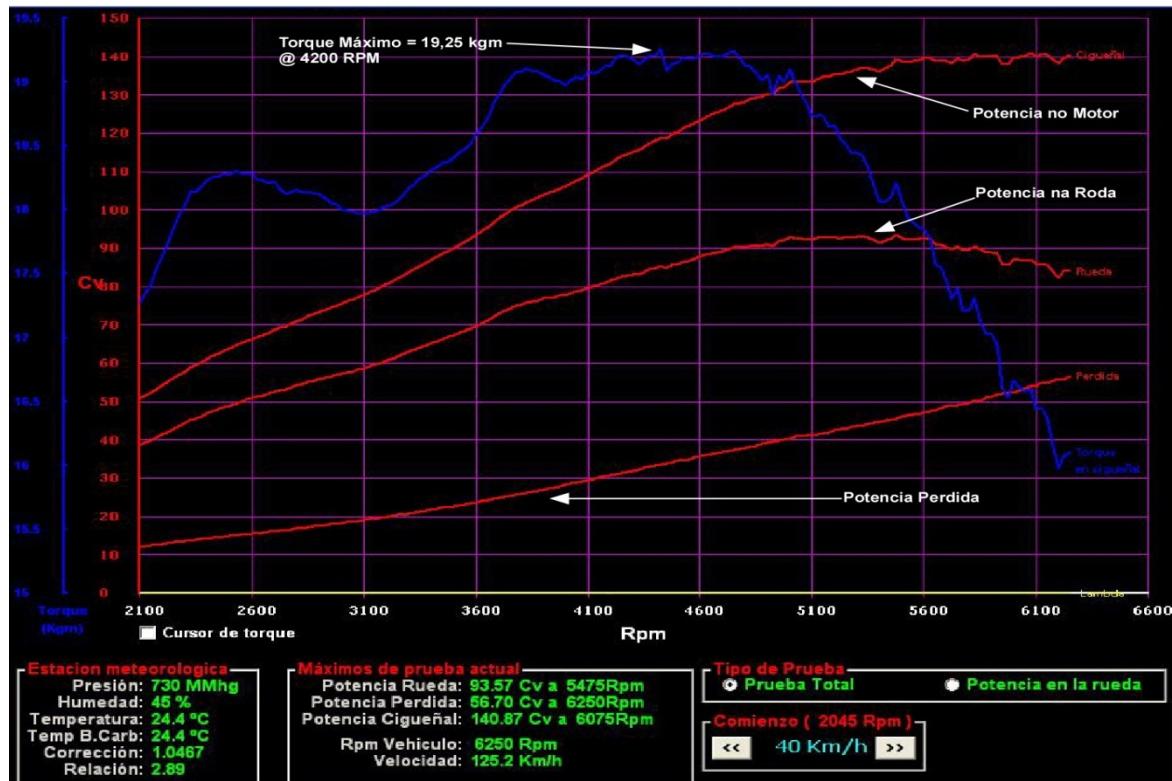
A figura 74 mostra um aumento significativo da potência no motor quando o *firmware* desenvolvido começou a calcular a eficiência volumétrica e passou a corrigir o tempo de injeção em função da dinâmica de enchimento do motor. A diferença apresentada no comportamento do motor foi amplamente nítida tanto fisicamente quanto nos resultados obtidos nos gráficos. A área selecionada em pontilhado indica o instante da transição entre o sistema de injeção sem o cálculo da eficiência volumétrica e após iniciar a correção do tempo de injeção em função da eficiência volumétrica. Sem calcular a eficiência volumétrica, foi obtida uma potência máxima de 20 cavalos na roda. Quando o motor ultrapassou os 3000 RPM e passou a corrigir o tempo de injeção em função da eficiência volumétrica houve uma ganho expressivo na potência e foi atingida uma potência máxima na roda de 85.32 cavalos a 4950 RPM.

Entretanto, mesmo com o nítido aumento na potência do motor com o cálculo da eficiência volumétrica, o motor apresentou muitas oscilações em sua potência medida, chegando a reduzir sua potência de 80 cavalos a 4200 RPM para 37 cavalos a 4600 RPM. Atribuiu-se esta oscilação ao incorreto controle da posição angular da válvula borboleta. Neste primeiro teste, a referência para a válvula borboleta era o pedal do acelerador. Como o teste exige que o pedal do acelerador

esteja totalmente pressionado, isto significa afirmar que o teste foi efetuado com a válvula borboleta totalmente aberta durante toda sua execução. Ao comparar o resultado obtido neste teste com a curva nominal de potência do sistema original de controle eletrônico deste motor, conforme ilustra a figura 25, é possível concluir que a curva de potência medida neste teste está muito abaixo da capacidade nominal deste motor.

Visando aprimorar o resultado obtido no primeiro teste e tendo como objetivo linearizar ao máximo a curva de potência medida, foi desenvolvido o controlador de rotação, conforme discutido na seção 4.7, permitindo calcular a eficiência volumétrica durante todo o teste. As ações diretas de compensação do controlador PI desenvolvido resultaram em uma melhoria significativa dos resultados em relação ao primeiro teste. A figura 75 ilustra o resultado obtido no teste final do sistema com o controlador de rotação implementado, que corrige a posição angular da válvula borboleta.

Figura 75 - Medição final da potência do motor



Fonte: o autor

No teste final realizado, a potência máxima obtida na roda foi de 93.57 cavalos a 5475 RPM. Neste projeto, optou-se por linearizar ao máximo a curva final de

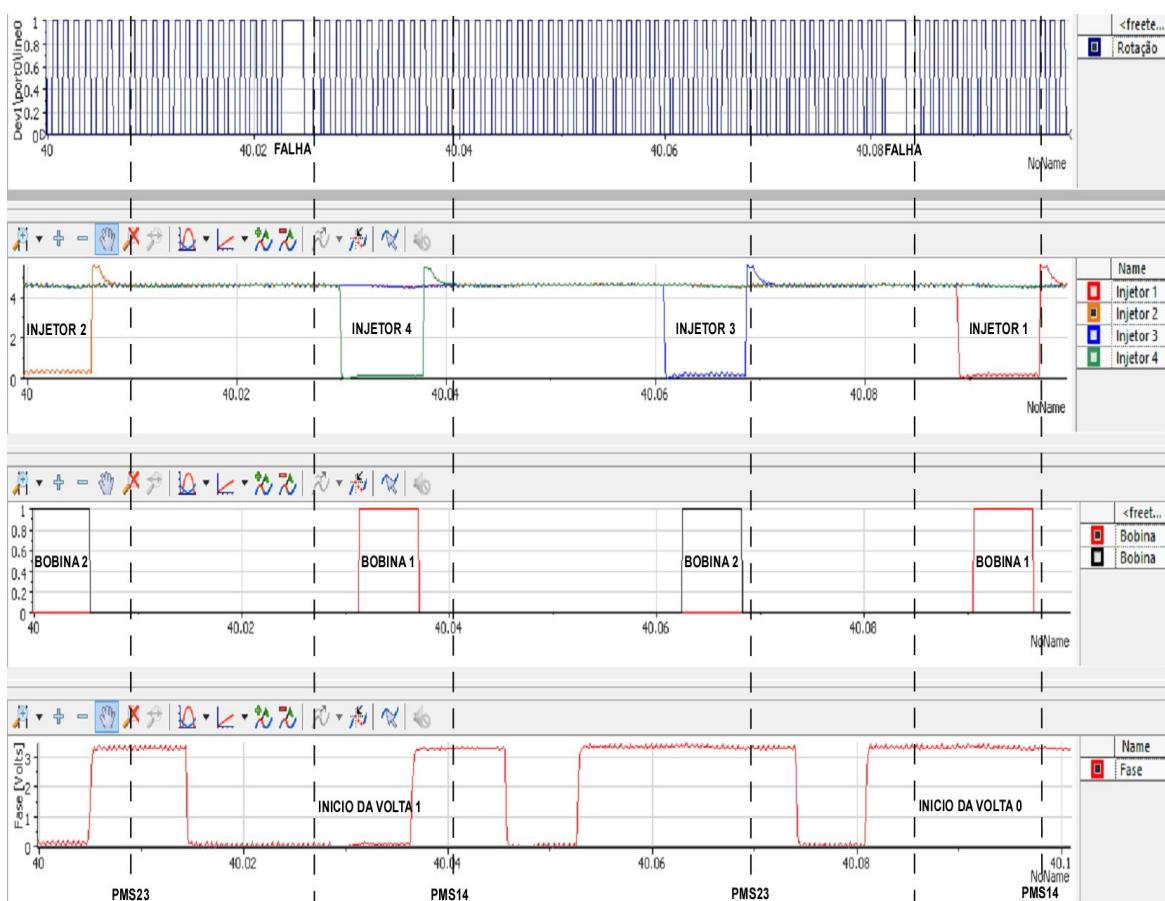
potência do motor. Antes de realizar o teste final, uma série de outros testes foram executados, variando o ganho proporcional e integral do controlador de rotação com objetivo de aumentar a potência máxima na roda e se aproximar ao máximo dos 116 cavalos de potência nominal deste motor. Foi possível atingir 100 cavalos de potência, porém o gráfico resultante apresentou instabilidades e oscilações na potência medida. Deste modo, optou-se por agir de forma conservadora, reduzindo os ganhos do controlador para linearizar a curva de potência. Durante a realização dos testes, ficou nítido que a redução dos ganhos do controlador reduz a potência máxima do motor, embora mantenha a curva de potência linear e com poucas oscilações, o que é algo desejado quando se controla motores ciclo Otto.

4.10.2 Validação das ações de sincronismo do motor

Conforme mencionado no capítulo 3, a UCM dos motores da família EA 113 2.0L sincroniza a injeção de combustível, com base na informação da posição angular da árvore de manivelas em relação à posição angular da árvore do comando de válvulas, utilizando como referência os sinais dos sensores de relutância variável (rotação) e do sensor de fase respectivamente. A implementação da camada de sincronismo resultou em um grande esforço no processo de aplicação prática. Os sensores elétricos instalados no motor atuam em um ambiente carregado de ações externas que influenciam negativamente, e podem gerar ruídos e em casos extremos erros de medição das grandezas físicas. Desta maneira, os testes iniciais na camada de sincronismo apresentaram muitas falhas no sincronismo do *firmware* com o motor, principalmente quando um ruído ocorria no sinal do sensor de rotação. Este cenário piora quando o ruído ocorre durante a falha de dois dentes do sinal do sensor de rotação. Por este motivo, foi necessário atuar em conjunto com o sensor de fase para se alcançar resultados satisfatórios. O sensor de fase deste motor é do tipo Hall e nos testes efetuados apresentou um baixo índice de ruídos em seu sinal. A ideia central do sistema de sincronismo é utilizar o sinal do sensor de fase em paralelo ao sensor de rotação. Algoritmos foram elaborados para sincronizar o sinal de ambos os sensores e utilizar a função do sensor de fase para corrigir o dente atual da função do sensor de rotação em toda volta completa do motor. Esta foi a ação que solucionou os problemas com os ruídos embutidos no sinal do sensor de rotação. Basicamente, a função do sensor de fase identifica o dente 28 da volta 0 e

o dente 57 da volta 1 e atualiza o valor da variável dente da função do sensor de rotação. Como a função do sensor de rotação é acionada a cada borda de subida do sinal do sensor de relutância variável e a variável dente é incrementada em toda execução da função, mesmo que haja um ruído no sinal e este ruído gere um incremento indevido da variável dente, a função do sensor de fase irá corrigir este valor a toda volta completa do motor. A figura 76 ilustra o resultado obtido com o sistema de sincronismo aplicado neste projeto.

Figura 76 - Medição final da sincronismo de controle do motor



Fonte: o autor

Os gráficos apresentados na figura 76 foram extraídos do banco de dados adquirido com o sistema de aquisição desenvolvido no *LabVIEW®* em conjunto com o sistema de intersecção. Todos os testes efetuados no dinamômetro inercial foram arquivados e os sinais de todos os sensores foram armazenados, os gráficos ilustrados na figura foram retirados do teste final no dinamômetro. Aleatoriamente, foi escolhido o instante equivalente ao quadragésimo segundo do teste para demonstrar a ação do sistema de sincronismo. Os sinais de acionamento das quatro

válvulas injetoras e das duas bobinas são ilustrados em um ciclo completo do motor Otto, ou seja, em duas voltas completa da árvore de manivelas. Para ilustrar o sincronismo com o motor, os sinais dos sensores de rotação e de fase também foram ilustrados. É possível notar que o valor de avanço do ponto de ignição das bobinas 1 e 2 se alteram após o décimo sexto dente do sinal da volta 1. Esta ação está de acordo com fluxograma apresentado na figura 49 do sistema de sincronismo, onde os novos parâmetros calculados pela camada RTOS são transferidos para a camada de sincronismo, fazendo com que os novos valores de avanço de ignição, tempos de carga da bobina e de injeção sejam aplicados aos seus respectivos atuadores.

4.11 CONSIDERAÇÕES FINAIS

Neste capítulo foram analisados os resultados obtidos durante os testes realizados com o sistema totalmente implementado. Para conseguir realizar todos os testes finais para validação da teoria aplicada, foi necessário desenvolver antecipadamente ferramentas que permitiram e facilitaram a execução dos testes. Foi fundamental para todo o desenvolvimento das atividades do projeto o sistema de intersecção. Este sistema permitiu sequenciar de forma lógica e gradual as atividades do projeto, pois permitiu analisar os sinais elétricos emitidos por todos os sensores do motor sem a necessidade de realizar ajustes técnicos ou alterar o chicote do veículo, além de permitir analisar o método de controle original aplicado nos atuadores do motor.

Propositalmente, priorizou-se planejar as ações e tarefas deste projeto, grande parte do tempo demandado neste projeto foi dedicado a análises das ações de controle do sistema original. Paralelamente à pesquisa acadêmica, foi possível aprimorar a teoria de controle e as estratégias utilizadas neste projeto. Estas ações foram de extrema importância para se projetar e desenvolver os circuitos eletrônicos contidos na UCM. Todos os componentes da unidade desenvolvida foram selecionados e testados anteriormente ao desenvolvimento do *layout* do circuito impresso. Todas as trilhas das quatro camadas do circuito impresso foram revisadas manualmente e após a fabricação do circuito impresso os componentes foram soldados na placa. Antes de se iniciar o desenvolvimento do *firmware*, o *hardware* foi analisado e validado com pequenos testes de bancada. Controlar motores de

combustão interna ciclo Otto demandam extremo cuidado, pois ações indevidas no sincronismo do motor danificariam de forma irreversível o motor utilizado no projeto.

É importante afirmar que as ações de controle aplicadas neste projeto somente foram possíveis após concluir o sistema de sincronismo entre o *firmware* e o motor. A conclusão do sistema de sincronismo demanda garantir que os pulsos de comando de todos os atuadores sejam efetuados no instante exato, respeitando a dinâmica de operação dos motores Otto. Falhas no sincronismo não são toleráveis em um sistema de gerenciamento eletrônico. Outro ponto que se deve considerar é que falhas no sistema de sincronismo prejudicariam qualquer modelo teórico de controle desenvolvido, pois possivelmente o modelo teórico e o resultado prático não se aproximariam conforme o esperado. Deste modo, grande parte do tempo utilizado no desenvolvimento do *firmware* foi dedicado ao sistema de sincronismo. Somente depois que o sistema de sincronismo estava totalmente satisfatório e confiável é que se iniciaram as atividades de levantamento do modelo matemático e desenvolvimento dos controladores apresentados neste capítulo.

Os controladores desenvolvidos se mostraram bastante eficazes, comprovando toda a teoria desenvolvida para cada controlador. O controlador de rotação apresentou o papel mais importante entre os controladores aplicados, uma vez que a variação da posição angular da válvula borboleta exerce grande influência sobre a dinâmica de operação do motor. Os ganhos aplicados a este controlador podem suavizar e linearizar a curva de potência do motor, porém em contrapartida limitam a potência final. O ideal é ajustar os ganhos proporcionais e integrais com coerência, dosando-os de maneira adequada, buscando a maior potência final com o menor índice de oscilações na curva de potência. Grandes oscilações na curva final de potência reduzem significativamente a eficiência energética dos motores.

Os resultados apresentados neste capítulo mostram que a unidade microcontroladora para gerenciamento eletrônico de motores de combustão interna ciclo Otto apresenta um comportamento similar aos sistemas convencionais de gerenciamento eletrônico de motores. Entretanto o sistema aqui desenvolvido possui a vantagem de proporcionar condições de acesso, elaboração e alterações nas estratégias de controle do motor. Em conjunto com o sistema de intersecção desenvolvido, a UCM se torna uma ferramenta completa para demais aplicações e testes de novos modelos de controle, não somente no motor utilizado neste projeto mas em qualquer outro motor ciclo Otto.

5 CONCLUSÃO

O trabalho apresentado integra uma metodologia elaborada com o objetivo de fornecer uma ferramenta completa para analisar, pesquisar, controlar, modificar, discutir e gerenciar de forma completa qualquer motor de combustão interna ciclo Otto. Seguindo o conceito de elaboração progressiva de cada componente, a metodologia elaborada executou de forma satisfatória sua hipótese. O resultado desta metodologia permitiu compreender claramente a dinâmica dos motores ciclo Otto e os motivos pelo qual a indústria automobilística migrou para os sistemas eletrônicos de gerenciamento do motor.

Motores de combustão interna ciclo são sistemas multivariáveis. Muitos são os fatores ou parâmetros que fazem que os motores alterem rotação, torque, potência, eficiência energética, consumo e emissão de poluentes. Motores ciclo Otto equipados com carburadores e distribuidor apresentam resultados satisfatórios ao analisar a curva de potência, porém ao analisar parâmetros como consumo de combustível, emissão de poluentes, dentre outros, os motores carburados não são eficientes. Testes realizados neste projeto mostraram que variações mínimas em qualquer parâmetro de controle do motor resultam em grandes variações na instabilidade do motor, no consumo de combustível e na eficiência energética do motor. Isto permite afirmar que ajustes nos principais parâmetros de controle do motor devem ser constantemente efetuados e isto somente é possível com a elaboração de unidades microcontroladoras contendo algoritmos para gerenciamento eletrônico de todos os parâmetros de calibração dos motores.

Inicialmente a indústria automobilística migrou para sistemas eletrônicos de gerenciamento dos motores Otto para atender as exigências vigentes nas legislações sobre emissões de resíduos tóxicos, porém posteriormente notou-se que sistemas eletrônicos de controle são excelentes para aprimorar o rendimento térmico, incrementar a potência e reduzir o consumo de combustível dos motores. Como estes fatores são diferenciais de venda para veículos novos, a indústria automobilística investe na elaboração de novas estratégias de controle e na aplicação de novos sistemas de gerenciamento. Atualmente, é extremamente raro um veículo sair de fábrica com controladores mecânicos e os sistemas de gerenciamento eletrônico de aplicação comercial são complexos algoritmos de controle elaborado por dezenas de profissionais, atuando com rendimento muito

superior aos motores do início do século XX. Entretanto, toda esta evolução impulsionada por fatores comerciais não é divulgada para a sociedade, e os sistemas comerciais são unidades cuja tecnologia embarcada é restrita e alterações não são permitidas pela indústria automobilística. A unidade microcontroladora para gerenciamento eletrônico, elaborada neste trabalho, apresenta um comportamento similar aos sistemas comerciais de gerenciamento eletrônico de motores, porém possui a vantagem de proporcionar condições de acesso, elaboração e alterações nas estratégias de controle do motor.

Em conjunto com o sistema de intersecção, com o sistema de diagnose e com *software* de aquisição de dados elaborado, a UCM se torna uma ferramenta completa para elaboração de novas aplicações, testes de novos modelos de controle e testes de novas estratégias de gerenciamento, não somente no motor utilizado neste projeto, mas em qualquer outro motor ciclo Otto, comprovando o sucesso da metodologia proposta neste projeto.

Para validação do *firmware* elaborado, foram realizados inúmeros testes em bancada e posteriormente no veículo utilizado neste trabalho. Estes testes apresentaram divergências entre os resultados alcançados em bancada e entre os resultados alcançados no veículo. As cargas impostas por um motor real são expressivamente maiores do que os valores apresentados em modelos teóricos e influenciaram negativamente na calibração e no sincronismo entre o *firmware* e o motor. Deste modo, foi necessária grande esforço para a elaboração do *firmware* em atividades de desenvolvimento e ajustes das funções elaboradas para realização do sincronismo do motor. A implementação de um sistema operacional de tempo real (RTOS) foi fundamental para o sucesso do *firmware* elaborado. O sistema se mostrou robusto e confiável com a aplicação do RTOS, e é possível afirmar que a complexidade das tarefas de sincronismo foram consideravelmente facilitadas após a implementação do sistema operacional. Ao atuar apenas com um microcontrolador para gerenciar todas as funções e parâmetros de controle do motor e levando em consideração a dinâmica dos motores Otto, a aplicação de um sistema operacional de tempo real é extremamente útil para agilizar todo o processo de desenvolvimento.

Os controladores elaborados utilizaram técnicas de controle clássico e para isto foi necessário concluir o sistema de sincronismo para se levantar o modelo. O principal controlador elaborado foi o de rotação do motor. A função de transferência

deste controlador foi adquirida através de ensaios em dinamômetro e os ganhos foram calculados e ajustados durante a execução dos testes finais.

O sistema de injeção elaborado neste trabalho é calculado para que a massa de combustível injetada para dentro do cilindro seja o mais próxima possível da relação estequiométrica. Durante os testes efetuados, concluiu-se que pequenas variações no tempo de injeção não resultaram em grandes variações na rotação do motor, houve apenas um aumento no índice de resíduos tóxicos produzidos na exaustão. O sistema de injeção apresenta deficiências em acelerações e a correção aplicada no tempo de injeção foi necessária na condição de aceleração para que o motor apresentasse funcionamento adequado.

O sistema de ignição foi mapeado e armazenado em memórias embutidas no microcontrolador. O teste efetuado requer cautela e o sistema de sincronismo deve ser extremamente confiável. A ocorrência de detonações prejudica os componentes mecânicos do motor. Deste modo, uma rápida detecção da detonação pelo software de análise utilizado foi necessária para a elaboração do mapa de ignição. Neste projeto, não foi possível atuar com o sensor de detonação, porém recomenda-se elaborar o sistema de ignição com o fechamento de malha e a correção automática do avanço de ignição em função da ocorrência da detonação detectada pela UCM.

Para o sistema de admissão, foi elaborado um controlador PI para gerenciar a variação angular da válvula borboleta eletrônica. Este parâmetro apresentou grande influência no motor. Em todos os testes efetuados, foi possível observar que pequenas variações no ângulo de abertura da válvula borboleta geraram grandes variações no comportamento do motor. Em testes efetuados com o motor sem carga e em marcha lenta, o sistema de admissão se mostrou o principal parâmetro de controle. Nesta condição, a estabilidade do motor oscila bastante, mesmo com pequenas variações na válvula borboleta. O correto controle do sistema de admissão é essencial para que o motor apresente resultados satisfatórios em testes de potência, consumo, estabilidade e emissão de poluentes.

Os resultados obtidos permitem concluir que o sistema desenvolvido é capaz de controlar a rotação do motor em torno de um valor de referência e ao mesmo tempo manter ótimos resultados na análise da curva de potência com uma carga aplicada no eixo principal do motor. O controlador de rotação elaborado exerceu um aumento considerável da linearidade da curva de potência do motor e a metodologia de construção do controlador proposta se confirmou satisfatória. Algumas questões

devem ser consideradas com relação aos resultados obtidos. Primeiramente, o avanço de ignição foi mapeado, esta limitação prejudica o desempenho do sistema, que não consegue otimizar o torque do motor em todas as faixas de operação. Deve ser considerado também que não foram realizadas análises de emissões de poluentes. O controle da mistura na região estequiométrica não garante que as emissões estejam dentro dos limites impostos pela legislação atual. Por fim, deve ser observado que os testes dinâmicos de variação de carga no eixo do motor foram baseados apenas na carga imposta pelo dinamômetro inercial e pela relação de transmissão do veículo em terceira marcha. Não foi avaliado o desempenho do sistema de controle diante de outras cargas, e isto permite afirmar que o desempenho do controlador de rotação elaborado está condicionado às condições de carga imposta nos testes realizados.

A UCM desenvolvida neste trabalho se mostrou uma ferramenta completa para elaboração de novas estratégias de controle. Os parâmetros e configurações do motor podem ser visualizados de forma amigável em tempo real pelo sistema de diagnose elaborado e a metodologia empregada durante toda a execução do projeto cumpriu com sucesso o objetivo inicial deste trabalho, podendo-se afirmar que o objetivo foi alcançado.

5.1 PROPOSTAS FUTURAS

Os resultados obtidos com a metodologia empregada na elaboração deste trabalho cria inúmeras possibilidades de pesquisas, sendo apresentadas algumas sugestões:

- Desenvolver um controlador do sistema de injeção de combustível, fechando a malha em função do sinal do sensor de oxigênio.
- Desenvolver um controlador do sistema de ignição, fechando a malha em função do sinal do sensor de detonação.
- Aprimorar o controlador de rotação, aplicando novas teorias de controle moderno.
- Aprimorar a interface Homem-maquina (HMI), para que seja possível alterar em tempo real parâmetros de calibração e os ganhos dos controladores.

- Desenvolver completa integração da UCM com o protocolo CAN do veículo.
- Desenvolver um sistema de diagnose via rede CAN.
- Aprimorar o *firmware*, utilizando novas linguagens de programação.
- Calibrar o motor em dinamômetros de bancada com variações de carga.
- Elaborar um controlador para um sistema bi-combustível.
- Implementar da arquitetura de *software* padronizada AUTOSAR.
- Levantar novos modelos, variando de forma individual todos os sensores do veículo.
- Aprimorar a curva de potência nominal do motor.
- Analisar a influência do tempo de carga da bobina na emissão de poluentes.
- Aprimorar e reduzir dos índices nominais de poluentes deste motor.
- Elaborar um controlador de consumo de combustível.
- Aplicar a unidade elaborada em motores de três, cinco, seis e oito cilindros.

REFERÊNCIAS

- ALBALADEJO, F. S.; **Desenvolvimento de uma unidade de gerenciamento eletrônico para motores de combustão interna de ciclo Otto.** 2013. 142p. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2013.
- ALCÂNTARA DIAS, B. M. de. **Plataforma didática de injeção eletrônica para controle de motores de combustão interna.** 2011. 77p. Monografia (Graduação Eletrônica Automotiva) - Faculdade de Tecnologia de Santo André, Santo André, 2011.
- AONO S. The Next Step in Automotive Electronic Control. In: VEHICULAR TECHNOLOGY CONFERENCE, IEEE 39th., 1989, San Francisco, CA. **Proceedings...** IEEE Cat. No.89CH2739-1, 1989. v.2, p.567-573.
- BALANOVIC, M. **Modeling and model-based control of a three-way catalytic converter.** 2002. 179p. Thesis (Ph.D) - Technische Universiteit Eindhoven, Eindhoven, Holanda, 2002.
- BASSHUYSEN, R. V. **Internal Combustion Engine Handbook.** 1.ed. SAE International. 2004, 868p.
- BILLINGS, S. A.; CHEN, S. The Identification of linear and nonlinear models of a turbo charged automotive diesel engine. In: MECHANICAL SYSTEM AND SIGNAL PROCESSING, v.3(2). 1989. **Proceedings...** doi:10.1016/0888-3270(89)90012-5, 1989. p.123-142.
- BOSCH. **Manual de Tecnologia Automotiva.** 25. ed. São Paulo: Edgard Blucher, 2005. 1232p.
- BRAGA, G. T. **Uma contribuição ao controle de motores de combustão interna de ignição por centelha na condição de marcha lenta.** 2007. 130p. Dissertação (Mestrado) - Universidade Federal Minas Gerais, Belo Horizonte, 2007.
- CÂMARA, J. C. C. **Monitoramento eletrônico da mistura ar/combustível em motores a combustão interna ciclo otto.** 2006. 171p. Dissertação (Mestrado) - Universidade Federal da Bahia, Salvador, 2006.
- CARVALHO, M. A. S. **Avaliação de um motor de combustão interna ciclo otto utilizando diferentes tipos de combustíveis.** 2011. 168p. Dissertação (Mestrado) - Universidade Federal da Bahia, Salvador, 2011.
- CHEN, R.; MI, L.; TAN, W. Adaptive Fuzzy Logic Based Sliding Mode Control of Electronic Throttle. **Journal of Computational Information Systems.** Chongqing 400050, China, Ago. 2012. Disponível em : http://www.jofcis.com/publishedpapers/2012_8_8_3253_3260.pdf Acesso em: 09 fev. 2015.

COOK J. A.; POWELL B. K. Modeling of an Internal Combustion Engine for Control Analysis. **IEEE CONTROL SYSTEMS MAGAZINE**. v.8, n.4, p.20-26, 1988.

DOREY R. E.; STUART G. Self-Tuning Control Applied to the In-Vehicle Calibration of a Spark Ignition Engine. In: THIRD IEEE CONFERENCE ON CONTROL APPLICATIONS, 1., 1994, Glasgow. **Proceedings...** 10.1109/CCA.1994.381239, 1994. p.121-126.

FREESCALE . **K40 Sub-Family Reference Manual**. Document Number: K40P144M100SF2RM. Rev. 3, 4 Nov 2010.

FREESCALE. **MC33186 H-Bridge Driver - Technical Data**. Document Number: MC33186 Rev. 7.0, 10/2011.

FREESCALE. **MC33810 Automotive Engine Control IC - Technical Data**. Document Number: MC33810 Rev. 9.0, 2/2011

GLIELMO, L.; VASCA, F.; ROSSI, C. Architecture for Electronic Control Unit Tasks in Automotive Engine Control. In: IEEE INTERNATIONAL SYMPOSIUM ON COMPUTER-AIDED CONTROL SYSTEM DESIGN, 2000, Anchorage, AK. **Proceedings...** 10.1109/CACSD.2000.900184, 2000. p.42-47.

GUEDES, M. A. C. **Estudo de compensadores para a válvula borboleta eletrônica automotiva**. 2014. 155p. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2014.

GUEDES, M. A. C.; ALBALADEJO. F. S.; LAGANA, A. A. M.; JUSTO, J. F. A Didactic kit to Study Intake Air System in Internal Combustion Engines. In: ICEE - INTERNATIONAL CONFERENCE ON ENGINEERING EDUCATION, 2012, Turku - Finland, **Proceedings...** ICEE2012, 2012. pg.838 -845.

GUIMARÃES, A. de A. **Eletrônica Embarcada Automotiva**, 1. ed., São Paulo: Érica, 2007. 326p.

HALLIDAY, D., RESNICK, R., WALKER, J. **Fundamentos de Física III: Eletromagnetismo**, 9. ed. São Paulo: LTC, 2012. v.3, 388p.

HENDRICKS, E. Engine Modeling for Control Applications: A Critical Survey. **An International Journal of Theoretical and Applied Mechanics**, v. 32, n. 5, p.387-396, 1997. ISSN 1572-9648.

HROVAT, D.; SUN, J. Models and control methodologies for IC engine idle speed control design. **Control Engineering Practice**, v. 5, n. 8, p. 1093-1100, 1997.

KJELLQVIST, D. **Concepts, Strategies and Controller for Gasoline Engine Management**. 2005. 80p. Dissertação (Mestrado) - Luleá University of Technology, Suécia. 2005.

MILHOR, C. E. **Sistema de desenvolvimento para controle eletrônico dos motores a combustão interna ciclo otto.** 2002. 101p. Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos - SP, 2002.

NATIONAL SEMICONDUCTOR. **Adaptive variable reluctance sensor amplifier.** 2005. p.11. DS007993. Disponível em :
<http://www.national.com> Acesso em: 09 fev. 2015.

NICOLAO, G. de; ROSSI, C.; SCATTOLINI R.; SUFFRITTI M. Identification and Idle Speed Control of Internal Combustion Engines. **Control Engineering Practice**, v.7, n. 9, p. 1061-1069, 1999.

OGATA, K. **Engenharia de Controle Moderno.** 3. ed. Rio de Janeiro: Prentice-Hall do Brasil, 1985. 930p.

PASSARINI, L. C. **Projeto e análise de válvulas eletromagnéticas injetoras de combustível: Uma nova proposta.** Tese (Doutorado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos - SP, 1993.

PEREIRA, B. C. F. **Unidade de gerenciamento eletrônico de um motor volkswagen 2.0L: Projeto Otto II.** 2013. 186p. Trabalho de Conclusão de Curso (Graduação) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2013.

PERTENCE JÚNIOR, A. **Eletrônica analógica: amplificadores operacionais e filtros ativos: teoria, projetos, aplicações e laboratório,** 6. ed. Porto Alegre: Bookman, 2003. 308p.

POWELL J. D.; FEKETE N. P.; CHANG C. Observer-Based Air-Fuel Ratio Control. **IEEE Control Systems Magazine**, v. 18, n.5, p.72-83, 1998. ISSN: 1066-033X.

PUJATTI, F. J. P. **Desenvolvimento de um sistema de gerenciamento eletrônico para motores de ignição por centelha.** 2007. 221p. Tese (Doutorado) - Universidade Federal de Minas Gerais, Belo Horizonte, 2007.

RIBBENS, W. B; **Understanding Automotive Electronics.** 6. ed. [S.I.]: SAE R-350: Newnes, 2003. 480p.

TERRENI, P.; GENTILI R. Closed-Loop Electronic Fuel Injection for Spark-Ignited Engines. **IEEE Transactions on Vehicular Technology**. v. 35, n. 1, p. 30-38, 1986. ISSN: 0018-9545

THAMIREZ DA SILVA, C. **Modelamento matemático e controle PI de uma válvula borboleta eletrônica,** 2015. 138p. Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo. 2015.

VOLKSWAGEN (Brasil). **Motores 1.6I e 2.0I Polo.** São Bernardo do Campo: Academia Volkswagen, 2007. 47p. Apostila técnica para treinamento do produto.

VOLKSWAGEN (Brasil). **Sistema de Gerenciamento de Motores Aspirados.** São Bernardo do Campo: Academia Volkswagen, 2009. 96p. Apostila técnica para treinamento do produto.

WANG, C.; LUO, F. Software Development of Automotive Engine Electronic Control Unit. In: INTERNATIONAL CONFERENCE ON POWER AND ENERGY SYSTEMS, v.13. 2012. **Proceedings...** ICPES2012, 2012. p.147-152.

YOON, P.; PARK S.; SUNWOO M. Closed-Loop Control of Spark Advance and Air-Fuel Ratio in SI Engines Using Cylinder Pressure. **SAE TECHNICAL PAPER**, 2000-01-0933, 2000. doi:10.4271/2000-01-0933.

APÊNDICE A - *Firmware Completo Utilizado no Teste Final*

MAIN.C

```
/****************************************************************************
 *          Escola Politécnica da Universidade de São Paulo
 *          POLI - USP - EPUSP
 *
 *          Mestrado em Engenharia Elétrica
 *          Microeletrônica
 *
 *          Firmware da Unidade de Gerenciamento Eletrônico do motor EA113 2.0L*****
 */

/****************************************************************************
 *          Autor ****
 *          Bruno Martin de Alcântara Dias
 */

/****************************************************************************
 *          Modulo:      cartronic_v3_2.c
 *
 *          Descrição:   Modulo principal (main.c)
 *
 *          Projeto:     UCM - CARTRONIC v3.0
 *
 *          Compilador:  CodeWarrior for MCU - Version: 10.2 - Build Id:120126
 *
 *          Hardware:   Cartronic v3.2 32bits - Freescale Kinetis K40 256k
 *
 *                      MCU = K40X256VLQ100
 *
 *
 *          Nucleo:      ARM CORTEX M4 32bits
 *
 *          Data: <10/05/2014>
 *
 *          Historico:  Iniciais      Motivo da Mudança
 *                      do Projetista
 *
 *          10/05/2014    BMAD        Inicial version.
 *          19/05/2014    BMAD        Sistema USB 1
 *          24/05/2014    BMAD        Sistema Injeção
 *          25/05/2014    BMAD        Conclusão SistemalIGN
 *          28/05/2014    BMAD        Sistema Admissão
 *          06/06/2014    BMAD        Controle Partida 01
 *          19/08/2014    BMAD        Aplicação Filtro ROT
 *          22/08/2014    BMAD        Inclusão Escalonador
 *          28/08/2014    BMAD        Sincronismo INJ,IGN
 *          09/09/2014    BMAD        Controle PID borboleta
 *          12/09/2014    BMAD        Alteração Injeção
 *          12/10/2014    BMAD        Partiu Motor 1a vez
 *          27/10/2014    BMAD        RTOS_VEM[RTOS_BUS]
 *          29/10/2014    BMAD        Alt Sistema Admissão
 *          29/10/2014    BMAD        Calculo Tinj
 *          20/12/2014    BMAD        Controle PID rotação e conclusão da versão final
 */

```

```
*****
*          DESCRIÇÃO GERAL DO PROGRAMA
*****/
```

```
// O programa principal possui a função de efetuar a partida no
// motor EA113 2.0L mantendo o mesmo em regime de operação.
// Buscando a maior potencia e a maior eficiência.
// Para isto o firmware controlará o sistema de injeção , ignição e admissão.
// O firmware receberá o sinal da linha 15, acionara a bomba de combustível,
// e efetuará a leitura do sensor de rotação e fase (ambos por interrupção
// externa) e acionara o injetor e a bobina em sincronismo com o motor. A
// válvula borboleta será controlada em função do TPS e do PEDAL e sua função
// ocorrerá no RTOS periodicamente a cada 10ms. O firmware tem como
// objetivo principal controlar a partir do modelo matemático dos três
// sistemas adquirido por pesquisa teórica e ensaios práticos, conforme
// descrição através dissertação deste mestrado.
```

```
*****
*  1. ARQUIVOS DE DEFINIÇÕES (INCLUDES)
*  1.1 Arquivos includes padroes do Compilador
*****/
```

```
#define          PE_MCUINIT
//Definição do DEVICE INICIALIZATION
```

```
*****
*  1.2 Arquivos includes padroes do C-ANSI
*****/
```

```
#include      <stdio.h>           //Biblioteca padrão de entrada e saídas
#include      <stdlib.h>          //Biblioteca padrão da linguagem C-ANSI
#include      <string.h>           //Biblioteca padrão de strings e chars
```

```
*****
*  1.3 Arquivos includes de usuário
*****/
```

```
#include "derivative.h"           //Contém as declarações dos periféricos
#include "definicoes.h"            //Contém as definições do programa
#include "ENGINE_MAPS.h"           //Contém os mapas de injeção e ignição
#include "prototipagem.h"          //Declara o protótipo de todas funções
#include "ComunicacaoDiagnose_tasks.h" //Contém as funções de comunicação "HMI"
#include "Comum_tasks.h"           //Contém as funções comuns ao projeto
#include "ComandoAtuadores_tasks.h" //Contém as funções dos atuadores
#include "AquisicaoSensores_tasks.h" //Contém as funções dos sensores
#include "Interrupcoes_tasks.h"     //Contém as funções das interrupções
#include "EngineManage_tasks.h"     //Funções de gerenciamento do Motor
```

```
*****
*  3. Variáveis Globais
*****/
```

```
// O firmware automotivo não possui variáveis globais. Padrão MISRA C.
```

```

/*********************4. INTERRUPCOES*********************/
*   4. INTERRUPCOES
*********************4. INTERRUPCOES*********************/



// Os vetores de interrupção estão no arquivo "MCUinit.c"

/*********************5. FUNCAO PRINCIPAL*********************/
*   5. FUNCAO PRINCIPAL
*********************5. FUNCAO PRINCIPAL*/



int main(void) {

/*
**----- Declaração de variáveis locais
**-----
*/
    //não ha variáveis locais
/*
**----- Limpeza dos PORTS
**-----
*/
    GPIOA_PDOR = 0;          //Zera pinos do PORT A configurados como GPIO
    GPIOB_PDOR = 0;          //Zera pinos do PORT B configurados como GPIO
    GPIOC_PDOR = 0;          //Zera pinos do PORT C configurados como GPIO
    GPIOD_PDOR = 0;          //Zera pinos do PORT D configurados como GPIO
    GPIOE_PDOR = 0;          //Zera pinos do PORT E configurados como GPIO
/*
**----- Configurações iniciais do microcontrolador
**-----
*/
    MCU_init();              //Executa as instruções de inicialização do MCU
    init_MCU_pins();         //Posiciona os pinos em seus valores iniciais
    USB1(HABILITA);         //Habilita tx e rx de dados no USB1

/*
**----- Inicializacao das variaveis
**-----
*/
    // Não ha variáveis na função principal

/*
**----- Habilita as interrupções
**-----
*/
    DesabilitaInterrupcao(ISR_ROTACAO);           //Desabilita sinal Rotação
    DesabilitaInterrupcao(ISR_FASE);               //Desabilita sinal Fase
    DesabilitaInterrupcao(ISR_PITO);               //Desabilita Interrupção PIT0
    PIT_TCTRL0 &= (!PIT_TCTRL_TEN_MASK);          //Desabilita PIT0
    DesabilitaInterrupcao(ISR_PIT1);               //Desabilita Interrupção PIT1
    PIT_TCTRL1 &= (!PIT_TCTRL_TEN_MASK);          //Desabilita PIT1
}

```

```

DesabilitaInterrupcao(ISR_PIT2);           //Desabilita Interrupção PIT2
PIT_TCTRL2 &= (!PIT_TCTRL_TEN_MASK);       //Desabilita PIT2
DesabilitaInterrupcao(ISR_PIT3);           //Desabilita Interrupção PIT3
PIT_TCTRL3 &= (!PIT_TCTRL_TEN_MASK);       //Desabilita PIT3
DesabilitaInterrupcao(ISR_PDB0);           //Desabilita Interrupção PDB0
PDB0_SC |= PDB_SC_SWTRIG_MASK;             //Software trigger

HabilitaInterrupcao(LINHA15);              //Habilita interrupção da Linha 15

/*
**-----
** Loop principal
**-----
*/
for(;;) {
    RTOS_RUN();    //Entrega o controle ao sistema operacional

}

return 0;                                // Retorna ZERO caso função MAIN se encerre

/*
**-----
** Fim do Programa
**-----
*/
}
}/* **** */
/* **** */
/* **** */
/* **** */

/* **** */
*          FIM DO ARQUIVO          *
***** */

```

DERIVATIVE.H

```

/*
*          Escola Politécnica da Universidade de São Paulo      */
*          POLI - USP - EPUSP                                     */
*          */                                                 */
*          Mestrado em Engenharia Elétrica                      */
*          Microeletrônica                                     */
*          */                                                 */
*****Firmware da Unidade de Gerenciamento Eletrônico do motor EA113 2.0L***** */
/*
*      Note: This file is recreated by the project wizard whenever the MCU is
*      changed and should not be edited by hand
*/
/* Include the derivative-specific header file */
#include <MK40DZ10.h>
***** */
*          FIM DO ARQUIVO          */
***** */

```

DEFINICOES.H

```
/****************************************************************************
 *          Escola Politécnica da Universidade de São Paulo
 *          POLI - USP - EPUSP
 *
 *          Mestrado em Engenharia Elétrica
 *          Microeletrônica
 *
 ****Firmware da Unidade de Gerenciamento Eletrônico do motor EA113 2.0L****

 **** Autor ****
 *          Bruno Martin de Alcântara Dias
 */

/*
 * Modulo:      cartronic_v3_2.c
 *
 * Descrição:   Biblioteca de defines (definicoes.h)
 *
 * Projeto:     UCM - CARTRONIC v3.0
 *
 * Compilador:  CodeWarrior for MCU - Version: 10.2 - Build Id:120126
 *
 * Hardware:    Cartronic v3.2 32bits - Freescale Kinetis K40 256k
 *
 *                  MCU = K40X256VLQ100
 *
 *
 * Nucleo:      ARM CORTEX M4 32bits
 *
 * Data: <10/05/2014>
 *
 * Historico:   Iniciais      Motivo da Mudança
 *               do Projetista
 *
 * 10/05/2014    BMAD        Inicial version.
 * 15/05/2014    BMAD        Defines p/ estruturar as funções
 * 24/05/2014    BMAD        Sistema Injeção
 * 25/05/2014    BMAD        Sistema Ignição
 * 28/05/2014    BMAD        Sistema Admissão
 * 06/06/2014    BMAD        Sistema Partida 01
 * 19/08/2014    BMAD        Aplicação Filtro ROT
 * 28/08/2014    BMAD        Sincronismo INJ,IGN
 * 09/09/2014    BMAD        Controle PID borboleta
 * 12/09/2014    BMAD        Alteração Injeção
 * 12/10/2014    BMAD        Partiu motor 1a vez!
 * 27/10/2014    BMAD        Definição RTOS_VEM
 * 18/12/2014    BMAD        Controle PID rotação
 *
 **** DESCRIÇÃO GERAL DO HEADER ****
 */
```

//Esta biblioteca tem como objetivo incluir dentro do programa principal
 //todos os defines padrões e de utilização comum dentro do projeto

```
*****
* 1. ARQUIVOS DE DEFINICOES (INCLUDES)
*****
```

```
*****
/*
 * Descrição: Defines de acesso rápido aos pinos PTxY.
 *           x = Indicação do respectivo PORT
 *           Y = Indicação do respectivo pino do PORT
 */
*****
```

```
#define      PTx0      0x1
#define      PTx1      0x2
#define      PTx2      0x4
#define      PTx3      0x8
#define      PTx4      0x10
#define      PTx5      0x20
#define      PTx6      0x40
#define      PTx7      0x80
#define      PTx8      0x100
#define      PTx9      0x200
#define      PTx10     0x400
#define      PTx11     0x800
#define      PTx12     0x1000
#define      PTx13     0x2000
#define      PTx14     0x4000
#define      PTx15     0x8000
#define      PTx16     0x10000
#define      PTx17     0x20000
#define      PTx18     0x40000
#define      PTx19     0x80000
#define      PTx20     0x100000
#define      PTx21     0x200000
#define      PTx22     0x400000
#define      PTx23     0x800000
#define      PTx24     0x1000000
#define      PTx25     0x2000000
#define      PTx26     0x4000000
#define      PTx27     0x8000000
#define      PTx28     0x10000000
#define      PTx29     0x20000000
#define      PTx30     0x40000000
#define      PTx31     0x80000000
*****
```

```

/*
 *   Descrição: Defines e constantes de afirmação e negação
 */

#define ON 1
#define OFF 0
#define TRUE 1
#define FALSE 0
#define HABILITA 1
#define DESABILITA 0

/*
 *   Descrição: Defines utilizados para estruturar as funções
 */

#define ARMAZENA 0 //Indica modo
#define AQUISITA 1 //Indica modo
#define CALCULA 0 //Indica modo
#define RETORNA 1 //Indica modo
#define ZERATIMER 1 //indica modo
#define ATUALIZA 2 //Indica modo
#define RTOS_BUS 61 //Total de variáveis do BUS RTOS

/*
 *   Descrição: Defines contendo o numero das interrupções de software
 */

#define LINHA15 0 //Interrupção externa PTA24
#define ISR_USB1_RX 1 //Interrupção RX UART4
#define ISR_ROTACAO 2 //interrupção externa PTD4
#define ISR_PITO 3 //Interrupção PIT0 (IJN)
#define ISR_PIT1 4 //Interrupção PIT1 (IJN)
#define ISR_PIT2 5 //Interrupção PIT2 (IGN)
#define ISR_PIT3 6 //Interrupção PIT3 (IGN)
#define ISR_FTM2 7 //Interrupção FMT2 (IGN)
#define ISR_PDB0 8 //Interrupção PDB0 (ADM)
#define ISR_FASE 9 //Interrupção externa PTD5

/*
 *   Descrição: Defines contendo o nome e o endereço das variáveis do vetor
 *   de estados do moto (VEM) utilizado pelo sistema RTOS
 */

```

```
*****
#define RPM_VEM 0 //Rotacao do Motor (RPM)
#define TINJ_VEM 1 //Indicacao do tempo atual inj
#define TCARGA_VEM 2 //Indicacao de tempo de CARGA IGN
#define AVANCO_VEM 3 //Indicacao do valor de AVANCO IGN
#define MAP_VEM 4 //Indicacao do valor de MAP
#define PEDAL1_VEM 5 //Indicacao do valor de PEDAL 1
#define PEDAL2_VEM 6 //Indicacao do valor de PEDAL 2
#define TPS1_VEM 7 //Indicacao do valor de TPS 1
#define TPS2_VEM 8 //Indicacao do valor de TPS 2
#define TAR_VEM 9 //Indicacao do valor de TEMP AR
#define TAGUA_VEM 10 //Indicacao do valor de TEMP AGUA
#define LAMBDA_VEM 11 //Indicacao do valor de LAMBDA
#define ALTERNADOR_VEM 12 //Indicacao do valor do ALTERNADOR
#define LINHA15_VEM 13 //Indicacao do estado da LINHA 15
#define FREIOL30_VEM 14 //Indicacao do estado do FREIO 1
#define FREIOL15_VEM 15 //Indicacao do estado do FREIO 2
#define EMBREAGEM_VEM 16 //Indicacao do estado da EMBREAGEM
#define MAIN_RELAY_VEM 17 //Indicacao do estado do MAIN RELAY
#define REGIME_VEM 18 //Indicacao do regime de operacao do motor
#define DENTE_VEM 19 //Dente atual do Motor
#define FASE_VEM 20 //Indicacao de fase do motor
#define KNOCK1_VEM 21 //Indicacao do valor de KNOCK 1
#define KNOCK2_VEM 22 //Indicacao do valor de KNOCK 2
#define INT_ALTERNADOR_VEM 23 //Flag de erro do ALTERNADOR
#define WARN_LAMBDA_VEM 24 //Estado do aquecimento de lambda
#define POSITIVO_RPM_VEM 25 //Estado do positivo sensor de rotacao
#define POSITIVO_SENSOR_VEM 26 //Estado do positivo sensor do motor 5V
#define FUelpump_RELAY_VEM 27 //Indicacao do estado do FUEL PUMP RELAY
#define COLDSTART_RELAY_VEM 28 //Indicacao do estado do COLD START RELAY
#define RESERVA_RELAY_VEM 29 //Indicacao do estado do RESERVA RELAY
#define COLDINJECTOR_VEM 30 //Indicacao do estado do Injetor COLD START
#define INJETOR1_VEM 31 //Indicacao do estado do INJETOR 1
#define INJETOR2_VEM 32 //Indicacao do estado do INJETOR 2
#define INJETOR3_VEM 33 //Indicacao do estado do INJETOR 3
#define INJETOR4_VEM 34 //Indicacao do estado do INJETOR 4
#define INJETOR5_VEM 35 //Indicacao do estado do INJETOR 5
#define INJETOR6_VEM 36 //Indicacao do estado do INJETOR 6
#define INJETOR7_VEM 37 //Indicacao do estado do INJETOR 7
#define INJETOR8_VEM 38 //Indicacao do estado do INJETOR 8
#define ATUADOR_VEM 39 //Indicacao do estado do ATUADOR RESERVA
#define PURGA_VEM 40 //Indicacao do estado do PURGA CANISTER
#define BOBINA1_VEM 41 //Indicacao do estado do BOBINA 1
#define BOBINA2_VEM 42 //Indicacao do estado do BOBINA 1
#define BOBINA3_VEM 43 //Indicacao do estado do BOBINA 1
#define BOBINA4_VEM 44 //Indicacao do estado do BOBINA 1
#define BOBINA5_VEM 45 //Indicacao do estado do BOBINA 1
#define BOBINA6_VEM 46 //Indicacao do estado do BOBINA 1
#define BOBINA7_VEM 47 //Indicacao do estado do BOBINA 1
#define BOBINA8_VEM 48 //Indicacao do estado do BOBINA 1
#define SENSOR_DIG1_VEM 49 //Estado do SENSOR DIGITAL RESERVA 1
#define SENSOR_DIG2_VEM 50 //Estado do SENSOR DIGITAL RESERVA 2
#define USB1_VEM 51 //Valor do dado do USB 1
#define USB1_FLAG_VEM 52 //Estado da flag para polling do USB 1
#define USB2_VEM 53 //Valor do dado do USB 2
#define USB2_FLAG_VEM 54 //Estado da flag para polling do USB 2
#define BLUE_VEM 55 //Valor do dado do BLUETOOTH
```

```

#define BLUE_FLAG_VEM      56 //Estado da flag para polling do BLUETOOTH
#define CAN1_VEM            57 //Valor do dado do CAN 1
#define CAN1_FLAG_VEM       58 //Estado da flag para polling do CAN 1
#define REF_RPM_VEM         59 //Referencia do motor calculada pelo
pedal
#define REF_VB_VEM          60 //Valor de referencia para Borboleta

//REGIME_VEM = 0 - MOTOR DESLIGADO
//REGIME_VEM = 1 - MOTOR MARCHA LENTA
//REGIME_VEM = 2 - MOTOR ACELERACAO
//REGIME_VEM = 3 - MOTOR DESACELERACAO
//REGIME_VEM = 4 - MOTOR CRUZEIRO
//REGIME_VEM = 5 - MOTOR PLENA CARGA
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
*                      FIM DO ARQUIVO
*****

```

ENGINE_MAPS.H

```

//****************************************************************************
/*
*           Escola Politécnica da Universidade de São Paulo
*           POLI - USP - EPUSP
*/
/*
*           Mestrado em Engenharia Elétrica
*           Microeletrônica
*/
/*
*****Firmware da Unidade de Gerenciamento Eletrônico do motor EA113 2.0L*****
```

```

***** Autor *****
/*
*           Bruno Martin de Alcântara Dias
*/
*****
```

```

*****
```

```

/* Modulo:      cartronic_v3_2.c
*/
/*
* Descrição:    Biblioteca contendo os mapas de sincronismo do motor (ENGINE_MAPS.h)
*/
/*
* Projeto:      UCM - CARTRONIC v3.0
*/
/*
* Compilador:   CodeWarrior for MCU - Version: 10.2 - Build Id:120126
*/
/*
* Hardware:    Cartronic v3.2 32bits - Freescale Kinetis K40 256k
*/
/*
*               MCU = K40X256VLQ100
*/
/*
* Nucleo:      ARM CORTEX M4 32bits
*/
/*
* Data: <10/05/2014>
*/
/*
* Historico:   Iniciais      Motivo da Mudança
*/

```

```
/*
 * do Projetista
 */
/*
 * 28/08/2014    BMAD      Inicial version.
 * 28/08/2014    BMAD      Sincronismo INJ,IGN
 * 12/10/2014    BMAD      Partiu motor 1a vez!
 * 27/10/2014    BMAD      RTOS_VEM [RTOS_BUS]
 * 10/11/2014    BMAD      UINT32_T -> UINT16_T
 * 18/12/2014    BMAD      Controle PID rotacao
*/
*****
```

```
const uint16_t DisparoBobina1_PIT2 [61][31] = {
```



```

{5455, 3561, 1667, 11136, 9242, 7348, 5455, 3561, 1667, 11136, 9242, 7348, 5455, 3561, 1667, 11136, 9242, 7348, 5455, 3561, 1667,
11136, 9242, 7348, 5455, 3561, 1667, 11136, 9242, 7348, 5455},
{10000, 8148, 6296, 4444, 2593, 741, 10000, 8148, 6296, 4444, 2593, 741, 10000, 8148, 6296, 4444, 2593, 741, 10000, 8148, 6296,
4444, 2593, 741, 10000, 8148, 6296, 4444, 2593, 741, 10000},
{3478, 1667, 10725, 8913, 7101, 5290, 3478, 1667, 10725, 8913, 7101, 5290, 3478, 1667, 10725, 8913, 7101, 5290, 3478, 1667, 10725,
8913, 7101, 5290, 3478, 1667, 10725, 8913, 7101, 5290, 3478},
{7872, 6099, 4326, 2553, 780, 9645, 7872, 6099, 4326, 2553, 780, 9645, 7872, 6099, 4326, 2553, 780, 9645, 7872, 6099, 4326, 2553,
780, 9645, 7872, 6099, 4326, 2553, 780, 9645, 7872},
{1667, 10347, 8611, 6875, 5139, 3403, 1667, 10347, 8611, 6875, 5139, 3403, 1667, 10347, 8611, 6875, 5139, 3403, 1667, 10347, 8611,
6875, 5139, 3403, 1667, 10347, 8611, 6875, 5139, 3403, 1667},
{5918, 4218, 2517, 816, 9320, 7619, 5918, 4218, 2517, 816, 9320, 7619, 5918, 4218, 2517, 816, 9320, 7619, 5918, 4218, 2517, 816,
9320, 7619, 5918, 4218, 2517, 816, 9320, 7619, 5918},
{0, 8333, 6667, 5000, 3333, 1667, 0, 8333, 6667, 5000, 3333, 1667, 0, 8333, 6667, 5000, 3333, 1667, 0, 8333, 6667, 5000, 3333, 1667, 0,
8333, 6667, 5000, 3333, 1667, 0},
{4118, 2484, 850, 9020, 7386, 5752, 4118, 2484, 850, 9020, 7386, 5752, 4118, 2484, 850, 9020, 7386, 5752, 4118, 2484, 850, 9020,
7386, 5752, 4118, 2484, 850, 9020, 7386, 5752, 4118},
{8077, 6474, 4872, 3269, 1667, 64, 8077, 6474, 4872, 3269, 1667, 64, 8077, 6474, 4872, 3269, 1667, 64, 8077, 6474, 4872, 3269, 1667,
64, 8077, 6474, 4872, 3269, 1667, 64, 8077},
{2453, 881, 8742, 7170, 5597, 4025, 2453, 881, 8742, 7170, 5597, 4025, 2453, 881, 8742, 7170, 5597, 4025, 2453, 881, 8742, 7170,
5597, 4025, 2453, 881, 8742, 7170, 5597, 4025, 2453},
{6296, 4753, 3210, 1667, 123, 7840, 6296, 4753, 3210, 1667, 123, 7840, 6296, 4753, 3210, 1667, 123, 7840, 6296, 4753, 3210, 1667,
123, 7840, 6296, 4753, 3210, 1667, 123, 7840, 6296},
{909, 8485, 6970, 5455, 3939, 2424, 909, 8485, 6970, 5455, 3939, 2424, 909, 8485, 6970, 5455, 3939, 2424, 909, 8485, 6970, 5455,
3939, 2424, 909, 8485, 6970, 5455, 3939, 2424, 909},
{4643, 3155, 1667, 179, 7619, 6131, 4643, 3155, 1667, 179, 7619, 6131, 4643, 3155, 1667, 179, 7619, 6131, 4643, 3155, 1667, 179,
7619, 6131, 4643, 3155, 1667, 179, 7619, 6131, 4643},
{8246, 6784, 5322, 3860, 2398, 936, 8246, 6784, 5322, 3860, 2398, 936, 8246, 6784, 5322, 3860, 2398, 936, 8246, 6784, 5322, 3860,
2398, 936, 8246, 6784, 5322, 3860, 2398, 936, 8246},
{3103, 1667, 230, 7414, 5977, 4540, 3103, 1667, 230, 7414, 5977, 4540, 3103, 1667, 230, 7414, 5977, 4540, 3103, 1667, 230, 7414,
5977, 4540, 3103, 1667, 230, 7414, 5977, 4540, 3103},
{6610, 5198, 3785, 2373, 960, 8023, 6610, 5198, 3785, 2373, 960, 8023, 6610, 5198, 3785, 2373, 960, 8023, 6610, 5198, 3785, 2373,
960, 8023, 6610, 5198, 3785, 2373, 960, 8023, 6610},
{1667, 278, 7222, 5833, 4444, 3056, 1667, 278, 7222, 5833, 4444, 3056, 1667, 278, 7222, 5833, 4444, 3056, 1667, 278, 7222, 5833,
4444, 3056, 1667, 278, 7222, 5833, 4444, 3056, 1667},
};

```

```
const uint8_t Mapalgnciao [61][13] = {  
{0, 7, 7, 7, 7, 8, 8, 8, 8, 8, 8, 8},  
{0, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10, 10, 10},  
{0, 9, 9, 9, 9, 10, 10, 10, 10, 10, 10, 10, 10},  
{0, 9, 9, 9, 9, 11, 11, 11, 11, 11, 11, 11, 10},  
{0, 9, 9, 10, 10, 12, 12, 12, 12, 12, 12, 11, 10},  
{0, 9, 9, 9, 9, 12, 12, 12, 12, 12, 12, 11, 10},  
{0, 9, 9, 9, 9, 12, 12, 12, 12, 12, 12, 11, 10},  
{0, 10, 10, 10, 10, 13, 13, 13, 13, 13, 13, 11, 10},  
{0, 10, 10, 10, 10, 13, 13, 13, 13, 13, 11, 10},  
{0, 10, 10, 10, 10, 13, 13, 13, 13, 13, 11, 10},  
{0, 11, 11, 11, 11, 13, 13, 13, 13, 13, 13, 11, 10},  
{0, 12, 12, 12, 12, 13, 13, 14, 14, 14, 14, 12, 11},  
{0, 13, 13, 13, 13, 14, 14, 14, 14, 14, 14, 12, 11},  
{0, 14, 14, 14, 14, 14, 14, 14, 14, 14, 14, 13, 12},  
{0, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 14, 13},  
{0, 15, 15, 15, 15, 15, 15, 15, 15, 15, 15, 14, 13},  
{0, 16, 16, 16, 16, 15, 15, 15, 15, 15, 15, 14, 13},  
{0, 17, 17, 17, 17, 16, 16, 15, 15, 15, 15, 15, 14},  
{0, 18, 18, 18, 17, 16, 16, 15, 15, 15, 15, 15, 14},  
{0, 18, 18, 18, 18, 17, 17, 15, 15, 15, 15, 15, 14},  
{0, 19, 19, 19, 18, 17, 17, 15, 15, 15, 15, 15, 15},  
{0, 19, 19, 19, 19, 18, 17, 15, 15, 15, 15, 15, 15},  
{0, 20, 20, 20, 19, 18, 18, 15, 15, 15, 15, 15, 15},  
{0, 21, 21, 21, 20, 19, 18, 15, 15, 15, 15, 15, 15},  
{0, 22, 22, 22, 21, 20, 19, 16, 16, 16, 16, 16, 16},  
{0, 22, 22, 22, 21, 20, 19, 16, 16, 16, 16, 16, 16},  
{0, 23, 23, 23, 22, 20, 19, 16, 16, 16, 16, 16, 16},  
{0, 24, 24, 24, 23, 21, 19, 16, 16, 16, 16, 16, 16},  
{0, 25, 25, 25, 24, 21, 19, 16, 16, 16, 16, 16, 16},  
{0, 25, 25, 25, 24, 22, 20, 16, 16, 16, 16, 16, 17},  
{0, 26, 26, 26, 25, 22, 20, 17, 17, 17, 17, 17, 17},  
{0, 27, 27, 27, 25, 22, 20, 17, 17, 17, 17, 17, 17},  
{0, 27, 27, 27, 26, 23, 20, 17, 17, 17, 17, 17, 17},  
{0, 28, 28, 28, 27, 23, 20, 17, 17, 17, 17, 17, 17},
```



```

{6, 1, 55, 50, 45, 41, 32, 32, 27, 22, 21, 21, 21, 21, 21, 21, 21, 21},  

{5, 1, 54, 49, 45, 40, 31, 31, 26, 21, 21, 21, 21, 21, 21, 21, 21},  

{5, 58, 54, 49, 44, 39, 30, 30, 25, 21, 21, 21, 21, 21, 21, 21, 21},  

{5, 58, 53, 48, 43, 38, 29, 29, 24, 21, 21, 21, 21, 21, 21, 21, 21},  

{5, 58, 53, 48, 42, 38, 28, 28, 23, 21, 21, 21, 21, 21, 21, 21, 21},  

{4, 57, 52, 47, 42, 37, 27, 27, 21, 21, 21, 21, 21, 21, 21, 21},  

{4, 57, 52, 46, 41, 36, 26, 26, 21, 21, 21, 21, 21, 21, 21, 21},  

{5, 57, 52, 47, 41, 36, 26, 26, 22, 22, 22, 22, 22, 22, 22, 22},  

{4, 57, 52, 46, 41, 35, 25, 25, 22, 22, 22, 22, 22, 22, 22},  

{4, 57, 51, 46, 40, 35, 24, 24, 22, 22, 22, 22, 22, 22, 22},  

{4, 56, 51, 45, 39, 34, 23, 23, 22, 22, 22, 22, 22, 22, 22},  

{3, 56, 50, 44, 39, 33, 23, 22, 22, 22, 22, 22, 22, 22, 22},  

{3, 55, 50, 44, 38, 32, 22, 22, 22, 22, 22, 22, 22, 22, 22},  

{3, 55, 49, 43, 37, 31, 22, 22, 22, 22, 22, 22, 22, 22, 22},  

{4, 56, 50, 44, 38, 32, 23, 23, 23, 23, 23, 23, 23, 23, 23},  

};

const uint8_t DenteRef_INJ23 [61][17] = {
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},  

{41, 41, 41, 41, 41, 41, 41, 40, 40, 40, 40, 40, 40, 40, 40, 40},  

{41, 41, 40, 40, 40, 40, 40, 39, 39, 39, 39, 39, 38, 38, 38},  

{41, 40, 40, 40, 39, 39, 39, 39, 38, 38, 38, 37, 37, 37, 36, 36},  

{40, 40, 40, 39, 39, 38, 38, 38, 37, 37, 36, 36, 35, 35, 34},  

{40, 40, 39, 39, 38, 38, 37, 37, 36, 36, 35, 35, 34, 34, 33, 32},  

{40, 39, 39, 38, 37, 37, 36, 36, 35, 34, 34, 33, 32, 32, 31, 30},  

{39, 39, 38, 37, 37, 36, 35, 35, 34, 33, 32, 32, 31, 30, 30, 28},  

{39, 38, 38, 37, 36, 35, 34, 34, 33, 32, 31, 30, 30, 29, 28, 26},  

{39, 38, 37, 36, 35, 34, 33, 33, 32, 31, 30, 29, 28, 27, 26, 24},  

{39, 38, 37, 36, 35, 34, 32, 32, 31, 30, 29, 28, 27, 25, 25, 22},  

{39, 38, 37, 36, 35, 34, 32, 32, 30, 29, 28, 27, 26, 25, 24, 21},  

{39, 38, 37, 35, 34, 33, 31, 31, 29, 28, 27, 26, 25, 23, 22, 19},  

{39, 37, 36, 35, 33, 32, 30, 30, 28, 27, 26, 24, 23, 22, 20, 17},  

{38, 37, 36, 34, 33, 31, 29, 29, 27, 26, 24, 23, 22, 20, 19, 15},  

{38, 37, 35, 34, 32, 31, 28, 28, 26, 25, 23, 22, 20, 19, 17, 13},  

{39, 37, 36, 34, 32, 31, 28, 28, 26, 24, 23, 21, 20, 18, 16, 12},  

{38, 37, 35, 33, 32, 30, 27, 27, 25, 23, 21, 20, 18, 16, 15, 10},  

{38, 36, 35, 33, 31, 29, 26, 26, 24, 22, 20, 18, 17, 15, 13, 8},  

{38, 36, 34, 32, 30, 28, 25, 25, 23, 21, 19, 17, 15, 13, 11, 6},  

{38, 36, 34, 32, 30, 27, 24, 24, 22, 20, 18, 16, 14, 12, 10, 4},  

{38, 36, 34, 32, 30, 28, 24, 24, 21, 19, 17, 15, 13, 11, 9, 3},  

{38, 36, 34, 31, 29, 27, 23, 23, 20, 18, 16, 14, 12, 9, 7, 1},  

{38, 35, 33, 31, 28, 26, 22, 22, 19, 17, 15, 12, 10, 8, 5, 57},  

{37, 35, 33, 30, 28, 25, 21, 21, 18, 16, 13, 11, 9, 6, 4, 55},  

{37, 35, 32, 30, 27, 25, 20, 20, 17, 15, 12, 10, 7, 5, 2, 53},  

{38, 35, 33, 30, 27, 25, 20, 20, 17, 14, 12, 9, 7, 4, 1, 52},  

{37, 34, 32, 29, 27, 24, 19, 19, 16, 13, 10, 8, 5, 2, 58, 50},  

{37, 34, 32, 29, 26, 23, 18, 18, 15, 12, 9, 6, 4, 1, 56, 48},  

{37, 34, 31, 28, 25, 22, 17, 17, 14, 11, 8, 5, 2, 57, 54, 47},  

{38, 35, 32, 29, 26, 23, 17, 17, 14, 11, 8, 5, 2, 57, 54, 48},  

{37, 34, 31, 28, 25, 22, 16, 16, 12, 9, 6, 3, 58, 55, 52, 48},  

{37, 34, 31, 27, 24, 21, 15, 15, 11, 8, 5, 2, 57, 53, 50, 48},  

{37, 33, 30, 27, 23, 20, 14, 14, 10, 7, 4, 58, 55, 52, 48, 48},  

{36, 33, 30, 26, 23, 19, 13, 13, 9, 6, 2, 57, 54, 50, 48, 48},  

{37, 34, 30, 27, 23, 20, 13, 13, 9, 6, 2, 57, 53, 50, 49, 49},  

{37, 33, 30, 26, 22, 19, 12, 12, 8, 4, 1, 55, 52, 49, 49, 49},  

{36, 33, 29, 25, 22, 18, 11, 11, 7, 3, 57, 54, 50, 49, 49, 49},  

{36, 32, 29, 25, 21, 17, 10, 10, 6, 2, 56, 52, 49, 49, 49, 49},  

{36, 32, 28, 24, 20, 16, 9, 9, 5, 1, 55, 51, 49, 49, 49, 49},  

{36, 32, 28, 24, 20, 16, 8, 8, 4, 58, 54, 50, 49, 49, 49, 49},  

{36, 32, 28, 24, 20, 16, 8, 8, 3, 57, 53, 50, 50, 50, 50, 50},  

{36, 32, 28, 23, 19, 15, 7, 7, 2, 56, 52, 50, 50, 50, 50, 50},  

{36, 31, 27, 23, 18, 14, 6, 6, 1, 55, 51, 50, 50, 50, 50, 50},  

{35, 31, 27, 22, 18, 13, 5, 5, 58, 54, 50, 50, 50, 50, 50, 50},  

{35, 31, 26, 22, 17, 13, 4, 4, 57, 53, 50, 50, 50, 50, 50, 50},  

{36, 31, 27, 22, 17, 13, 4, 4, 57, 52, 51, 51, 51, 51, 51, 51},  

{35, 31, 26, 21, 17, 12, 3, 3, 56, 51, 51, 51, 51, 51, 51, 51},  

{35, 30, 26, 21, 16, 11, 2, 2, 55, 51, 51, 51, 51, 51, 51, 51},  

{35, 30, 25, 20, 15, 10, 1, 1, 54, 51, 51, 51, 51, 51, 51, 51},  

{35, 30, 25, 20, 14, 10, 58, 58, 53, 51, 51, 51, 51, 51, 51, 51},  

{34, 29, 24, 19, 14, 9, 57, 57, 51, 51, 51, 51, 51, 51, 51, 51},  

{34, 29, 24, 18, 13, 8, 56, 56, 51, 51, 51, 51, 51, 51, 51, 51},
};

```

PROTOTIPAGEM.H

```
/*
 *          Escola Politécnica da Universidade de São Paulo
 *          POLI - USP - EPUSP
 */
/*
 *          Mestrado em Engenharia Elétrica
 *          Microeletrônica
 */
/*
 ****Firmware da Unidade de Gerenciamento Eletrônico do motor EA113 2.0L****

 **** Autor ****
 */
/*
 *          Bruno Martin de Alcântara Dias
 */
/*
 */
/* Modulo:      cartronic_v3_2.c
 */
/* Descrição:   Biblioteca de prototipagem das funções (prototipagem.h)
 */
/* Projeto:     UCM - CARTRONIC v3.0
 */
/* Compilador:  CodeWarrior for MCU - Version: 10.2 - Build Id:120126
 */
/* Hardware:    Cartronic v3.2 32bits - Freescale Kinetis K40 256k
 */
/*           MCU = K40X256VLQ100
```

```

/*
/*
/* Nucleo:      ARM CORTEX M4 32bits
/*
/* Data: <10/05/2014>
/*
/* Historico:      Iniciais      Motivo da Mudança
/* do Projetista
/*
/* 10/05/2014      BMAD          Inicial version.
/* 15/05/2014      BMAD          Task 04, 42
/* 24/05/2014      BMAD          task 43, 44, 84, 85, 86, 87, 88, 102, 103, 105 e 108;
/* 25/05/2014      BMAD          task 89, 106, 83, 109;
/* 28/05/2014      BMAD          task 24, 61, 90, 107;
/* 06/06/2014      BMAD          task 111, 112;
/* 07/06/2014      BMAD          task 25;
/* 19/08/2014      BMAD          Aplicacao Filtro ROT
/* 22/08/2014      BMAD          task 26
/* 28/08/2014      BMAD          Sincronismo INJ,IGN
/* 28/08/2014      BMAD          task 62, 63
/* 09/09/2014      BMAD          Controle PID borboleta
/* 09/09/2014      BMAD          task 27, 64
/* 12/09/2014      BMAD          Alteracao Injecao
/* 12/10/2014      BMAD          Partiu motor 1a vez!
/* 29/10/2014      BMAD          task 107
/* 05/11/2014      BMAD          Atualizacao task 85,86
/* 18/12/2014      BMAD          Controle PID rotacao
/* 20/12/2014      BMAD          task 110
/* 20/12/2014      BMAD          task 111
//****************************************************************************
/*
*                               DESCRICAO GERAL DO HEADER
*/

```

//Esta biblioteca contem os protótipos de todas as funções existentes no
//projeto, contendo o numero da função e uma breve descrição

```

//****************************************************************************
/*
*   Prototipagem de funções
*/
//****************************************************************************
void MCU_init(void);                                // Função que contem a configuração inicial do MCU
// Desenvolvido pelo DEVICE INITIALIZATION

//****************************************************************************
*   ComunicacaoDiagnose_tasks.h
*/
//****************************************************************************
void USB1 (uint8_t comando);                         //task: 01
void EnviaDadoUSB1 (char dado);                     //task: 02
void EnviaMensagemUSB1 (char* texto);               //task: 03
unsigned char RecebeUSB1 (uint8_t modo);             //task: 04
void Diagnose (uint16_t erro);                      //task: 17
void HMI (void);                                    //task: 18

```

```
*****
* Comum_tasks.h
*****
```

```
void delay_us (uint32_t valor);                                //task: 21
char* ConverteINTtoCHAR(int i, char texto[]);                //task: 22
void init_MC33810 (uint8_t numero_ci);                         //task: 23
void init_MC33186 (void);                                     //task: 24
void init MCU_pins (void);                                    //task: 25
void escalonador (uint8_t modo, uint16_t *scheduler);        //task: 26
void RTOS_RUN(void);                                         //task: 27
```

```
*****
* ComandoAtuadores_tasks.h
*****
```

```
void ControlaLed (uint8_t comando, uint8_t led);             //task: 41
void ControlaMainRele (uint8_t comando);                      //task: 42
void ControlaInjetores(uint8_t comando, uint8_t injetor);   //task: 43
void ControlaBobinas(uint8_t comando, uint8_t bobina);      //task: 44
void ControlaSensores (uint8_t comando);                     //task: 45
void ControlaSensorRotacao (uint8_t comando);               //task: 46
```

```
*****
* AquisicaoSensores_tasks.h
*****
```

```
uint8_t SensorTPS1 (void);                                  //task: 61
uint8_t SensorTPS2 (void);                                  //task: 62
uint8_t SensorPEDAL1 (void);                             //task: 63
uint8_t SensorPEDAL2 (void);                             //task: 64
uint8_t SensorMAP (void);                                //task: 65
uint8_t SensorTAGUA (void);                            //task: 66
uint8_t SensorTAR (void);                               //task: 67
```

```
*****
* Interrupcoes_tasks.h
*****
```

```
void HabilitaInterrupcao(uint8_t NomeInterrupcao);        //task: 81
void DesabilitaInterrupcao(uint8_t NomeInterrupcao);       //task: 82
void Vetor_Linha15(void);                                 //task: 83
void Vetor_SensorRotacao(uint8_t TCTRL, char *input);     //task: 84
void Vetor_PIT0(uint8_t *Tinj, uint8_t Injetor);          //task: 85
void Vetor_PIT1(uint8_t *Tinj, uint8_t Injetor);          //task: 86
void Vetor_PIT2(uint32_t *TempoCarga, uint8_t TCTRL);    //task: 87
void Vetor_PIT3(uint32_t *TempoCarga, uint8_t TCTRL);    //task: 88
void Vetor_PDB0(void);                                   //task: 89
```

```
*****
* EngineManage.h
*****
```

```
void HabilitaInjetores(uint8_t comando, uint8_t CI);        //task: 101
void IdentificaFase (char *volta);                          //task: 102
void Rotacao (uint8_t TCTRL, uint16_t *t_dente, uint8_t *RPM); //task: 103
void SincronismoSistemaADM(void);                           //task: 104
```

```

void CalculoTempoInjecao(uint8_t TCTRL, uint8_t *Tinj);           //task: 105
void CalculoAvancoIGN(uint8_t TCTRL, uint8_t *avancoIGN);        //task: 106
void init_engine(uint8_t TCTRL);                                    //task: 107
void AtualizaRTOS (void);                                         //task: 108
void TempoMaximoinjecao (uint8_t *TMAX);                         //task: 109
void PID_RPM (void);                                              //task: 110
void EstadoAtualMotor (void);                                     //task: 111
void ReferenciaRPM(void);                                       //task: 112
//****************************************************************************
*                                FIM DO ARQUIVO
*****

```

COMUNICAÇÃOEDIAGNOSE_TASKS.H

```

//****************************************************************************
/*
Escola Politécnica da Universidade de São Paulo
POLI - USP - EPUSP
*/
/*
Mestrado em Engenharia Elétrica
Microeletrônica
*/
*****
*****Firmware da Unidade de Gerenciamento Eletrônico do motor EA113 2.0L*****


***** Autor *****
/*
Bruno Martin de Alcântara Dias
*/
*****


/*
Modulo:      cartronic_v3_2.c
*/
/*
Descrição:   Biblioteca de Comunicação e Diagnose (ComunicacaoDiagnose_tasks.h)
*/
/*
Projeto:     UCM - CARTRONIC v3.0
*/
/*
Compilador:  CodeWarrior for MCU - Version: 10.2 - Build Id:120126
*/
/*
Hardware:   Cartronic v3.2 32bits - Freescale Kinetis K40 256k
*/
/*
MCU = K40X256VLQ100
*/
/*
Nucleo:      ARM CORTEX M4 32bits
*/
/*
Data: <10/05/2014>
*/
/*
Historico:      Iniciais      Motivo da Mudança
do Projetista
*/
/* 10/05/2014    BMAD         Inicial version.
/* 15/05/2014    BMAD         Conclusao RecebeUSB1
/* 24/05/2014    BMAD         Sistema Injecao
/* 25/05/2014    BMAD         Sistema Ignicao
/* 28/05/2014    BMAD         Sistema Admissao
/* 31/05/2014    BMAD         Aprimorando task 03
/* 06/06/2014    BMAD         Aprimorando task 04
/* 19/08/2014    BMAD         Aplicacao Filtro ROT
/* 28/08/2014    BMAD         Sincronismo INJ,IGN
*/

```

```

/* 09/09/2014      BMAD      Controle PID borboleta      */
/* 12/09/2014      BMAD      Alteracao Injecao          */
/* 12/10/2014      BMAD      Partiu motor 1a vez!        */
/* 18/12/2014      BMAD      Controle PID rotacao       */
//*****************************************************************************/
//*****************************************************************************/
/*
*           DESCRICAO GERAL DA BIBLIOTECA
*/
***** */

//Esta biblioteca agrupa todas as tarefas de comunicação externa e diagnose
***** */

//*****************************************************************************/
/*
*           FUNCOES DA COMUNICACAO USB 1
*
*   A COMUNICACAO USB 1 É INDIRETA. ISTO SIGNIFICA QUE A STACK DO PROTOCOLO
*   USB É IMPLEMENTADA PELO C.I FT232RL. ESTE C.I CONVERTE O PROTOCOLO UART
*   EM MENSAGENS PADRAO USB. PARA ENVIAR E RECEBER MENSAGENS O MCU UTILIZA O
*   PERIFERICO "UART4". SUA CONFIGURACAO É:
*       BAUD RATE: 19200 bps
*       PARIDADE : NAO
*       DATA BITS: 8
*       STOP BIT : 1
*       ORDEM TX : LSB FIRST
*       RX     : PTC14
*       TX     : PTC15
*/
***** */

/*
*           Funcao  : void USB1 (uint8_t comando)
*/
/*
*           TaskNumber: 01
*/
/*
*           Entradas : uint8_t comando (Indica ação a ser executada)
*/
/*
*           Saida   : nenhuma
*/
/*
*           Descricao : Esta função Habilita e Desabilita a comunicação USB 1
*/
***** */

void USB1 (uint8_t comando) {

    switch (comando) {
        case 0:                                //Comando = DESABILITA
            UART4_C2 &= (!UART_C2_TE_MASK | !UART_C2_RE_MASK);
            break;

        case 1:                                //Comando = ABILITA
            UART4_C2 |= (UART_C2_TE_MASK | UART_C2_RE_MASK);
            break;
    }
} //Final da Função
***** */
***** */
***** */
***** */

```



```

/*
/* Entradas : uint8_t modo (Indica se armazenara ou retorna saida)
/*          uint8_t sistema (Indica qual sistema sera utilizado)
*/
/*
/* Saída : Retorna o dado lido no RX do USB1
*/
/*
/* Descricao : Esta funcao possui duas condicoes. A primeira eh apenas
/*      receber o dado do USB1 e armazenar em uma variavel static
/*      A segunda eh retornar o dado previamente armazenado
/*      Ela foi desenvolvida para atuar na interrupcao de RX do USB1
/*      Sistema 0 = Leitura diversa
/*      Sistema 1 = Leitura sistema injecao
/*      Sistema 2 = Leitura sistema ignicao
/*      Sistema 3 = Leitura sistema admissao
//****************************************************************************
//****************************************************************************

unsigned char RecebeUSB1 (uint8_t modo) {

    static uint8_t dado = 0x61;      //Dado estatico de recepcao do USB1

    switch (modo) {

        case 0:                //Caso modo = ARMAZENA

            dado = UART4_S1; //Passo para limpar flag interrupcao
            dado = UART4_D; //Captura dado do buffer UART4
        break;
        case 1:                //Caso modo = AQUISITA

            return dado;
        break;
    }
} //Final da Funcao
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************

/*
*           FUNCOES DE DIAGNOSE E HMI
*
* REUNIRA AS FUNCOES DE DIAGNOSE E DE COMUNICACAO COM SOFTWARE PCI EXTERNOS
* DE AQUISICACAO DE DADOS E INTERFACE HOMEM-MAQUINA USANDO QUALQUER UM DOS
* PROTOCOLOS.
*****
*****
```

```

/*
 *      e devera estar preparada para receber via comunicação
 *      externa (USB ou Bluetooth), decodificar e apresentar os
 *      parâmetros de controle do motor. Esta função utiliza como
 *      base os dados do Vetor de estado do motor RTOS_VEM().
 *      Esta função foi desenvolvida para trabalhar dentro do RTOS
 *      com um byte enviado a cada 10ms. Totalizando 20 bytes.
 */
void HMI (void) {

    static uint8_t i = 0;           //Variável indexadora da tarefa

    if(i == 19) EnviaDadoUSB1(0xFF); //Envia Byte de sincronismo
    else EnviaDadoUSB1(RTOS_VEM[i]); //Envia dados indexados por "i"
    i++;                         //Incrementa "i"
    if (i == 20) i = 0;           //Caso i = 20, resetamos a função zerando i
} //Final da Função
/*
 *                                FIM DO ARQUIVO
 */

```

COMUN_TASKS.H

```

/*
 *      Escola Politécnica da Universidade de São Paulo
 *      POLI - USP - EPUSP
 *
 *      Mestrado em Engenharia Elétrica
 *          Microeletrônica
 *
 *      Firmware da Unidade de Gerenciamento Eletrônico do motor EA113 2.0L
 */

/*
 *      Autor
 */
/*
 *      Bruno Martin de Alcântara Dias
 */

/*
 *      Modulo:      cartronic_v3_2.c
 */
/*
 *      Descrição:   Biblioteca de Tarefas comuns (Comun_tasks.h)
 */
/*
 *      Projeto:     UCM - CARTRONIC v3.0
 */
/*
 *      Compilador:  CodeWarrior for MCU - Version: 10.2 - Build Id:120126
 */
/*
 *      Hardware:   Cartronic v3.2 32bits - Freescale Kinetis K40 256k
 */
/*
 *                  MCU = K40X256VLQ100
 */
/*
 *      Nucleo:     ARM CORTEX M4 32bits
 */

```

```

/*
/* Data: <10/05/2014>
*/
/*
/* Historico:           Iniciais          Motivo da Mudança
/* do Projetista
/* 10/05/2014      BMAD      Inicial version.
/* 16/05/2014      BMAD      init_MC33810
/* 24/05/2014      BMAD      Sistema Injecao
/* 25/05/2014      BMAD      Sistema Ignicao
/* 28/05/2014      BMAD      Sistema Admissao
/* 06/06/2014      BMAD      Controle Partida 01
/* 07/06/2014      BMAD      Init MCU pins
/* 19/08/2014      BMAD      Aplicacao Filtro ROT
/* 22/08/2014      BMAD      Inclusao Escalonador
/* 28/08/2014      BMAD      Sincronismo INJ,IGN
/* 09/09/2014      BMAD      Controle PID borboleta
/* 12/09/2014      BMAD      Alteracao Injecao
/* 12/10/2014      BMAD      Partiu motor 1a vez!
/* 30/10/2014      BMAD      RTOS_VEM [RTOS_BUS]
/* 18/12/2014      BMAD      Controle PID rotacao
//****************************************************************************

/*
*          DESCRIÇÃO GERAL DA BIBLIOTECA
*/
***** */

//Esta biblioteca agrupa o conjunto de tarefas que tarefas comuns ao projeto
//************************************************************************

/*
*          Funcao : void delay_us (uint32_t int valor)
*/
/*
*          TaskNumber: 21
*/
/*
*          Entradas : uint32_t (Indica o valor em microsegundos)
*/
/*
*          Saída : Nenhuma
*/
/*
*          Descrição : Esta função trava o MCU pelo tempo determinado pela int
*                      valor em microsegundos
*/
***** */

void delay_us (uint32_t valor) {
    uint32_t delay = 0;                                //Receber o resultado da conversao para us
    uint32_t i = 0;                                   //Indexador para funcao FOR

    delay = valor * 20;                             // Realiza conversão para us (MCU CLOCK = 100MHz)
    for (i = 0; i <= delay; i++);                  //Aguarda o tempo desejado em us
} //Final da Função
***** */
***** */
***** */
***** */

```

```

/*****************/
/*      Funcao  : char ConverteINTtoCHAR (int i, char texto[])
/*
/* TaskNumber: 22
/*
/* Entradas : int i (Entrada do numero inteiro a ser convertido)
/*
/* Saída   : char texto [] (Retorna int convertido em char)
/*
/* Descricao : Esta funcao converte uma variavel inteira (INT) em uma
/*             matrix de char (STRING)
/*****************/
/*****************/
char* ConverteINTtoCHAR(int i, char texto[]){
    char const numeros[] = "0123456789";           //Números para conversão
    char *p = texto;                                //Cria o ponteiro de indexação
    int shift;                                     //Variável de avanço da matrix

    if(i<0){                                       //Caso o numero INT seja negativo
        *p++ = '-';
        i *= -1;
    }
    shift = i;
    do{                                            //Avanço a o final do numero
        ++p;
        shift = shift/10;
    }while(shift);
    *p = '\0';                                     //Inseri comando para pular a linha
    do{                                            //Regredi o avanço inserindo os números na matrix
        *--p = numeros[i%10];
        i /= 10;
    }while(i);
    return texto;                                  //Retorna a matrix convertida
}
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*
/*      Funcao  : void init_MC33810 (uint8_t numero_ci)
/*
/* TaskNumber: 23
/*
/* Entradas : uint8_t numero_ci (Indica qual c.i sera iniciado)
/*
/* Saída   : Nenhuma
/*
/* Descrição : Esta função inicia a comunicação SPI entre o MCU e o c.i
/*             MC33810. A UCM deste projeto possui 2 c.i. Um para cada
/*             4 injetores e cada 4 bobinas. A variável numero_ci indica
/*             qual c.i sera inicializado com as configurações necessária
/*****************/
/*****************/

```

```

void init_MC33810 (uint8_t numero_ci) {
    uint8_t flag = 0;                                //flag interna da funcao
    uint8_t i = 0;                                   //Indexador para nao travar funcao
    uint16_t dado = 0;                               //Recebe resposta do C.I

    HabilitaInjetores(DESABILITA, numero_ci);        //Desabilita saida injetores

    if(numero_ci == 1) GPIOA_PSOR = PTx14;           // CS_33810_1 = 1 ---- PTA14
    if(numero_ci == 2) GPIOC_PSOR = PTx3;             // CS_33810_2 = 1 ---- PTC3

    if(numero_ci == 1) GPIOA_PCOR = PTx14;           // CS_33810_1 = 0 ---- PTA14
    if(numero_ci == 2) GPIOC_PCOR = PTx3;             // CS_33810_2 = 0 ---- PTC3

    delay_us(1);                                     //aguarda 1 us
    while (!SPIO_SR && SPI_SR_TFFF_MASK);          //espera o buffer TX FIFO esvaziar
    SPIO_PUSHR = 0b1110000000000000;                  //carrega comando de calibracao clock
    while (!(SPIO_SR && SPI_SR_TCF_MASK));          //espera a transmissao acabar...
    delay_us(1);                                     //aguarda 1 us
    if(numero_ci == 1) GPIOA_PSOR = PTx14;           // CS_33810_1 = 1 ---- PTA14
    if(numero_ci == 2) GPIOC_PSOR = PTx3;             // CS_33810_2 = 1 ---- PTC3
    delay_us(2);                                     //aguarda 2 us

    /* Pulso de 32us no CS do SPI */
    if(numero_ci == 1) GPIOA_PCOR = PTx14;           // CS_33810_1 = 0 ---- PTA14
    if(numero_ci == 2) GPIOC_PCOR = PTx3;             // CS_33810_2 = 0 ---- PTC3
    delay_us(32);                                    //aguarda 32 us

    /* Requisito de calibracao do MC33810 */
    if(numero_ci == 1) GPIOA_PSOR = PTx14;           // CS_33810_1 = 1 ---- PTA14
    if(numero_ci == 2) GPIOC_PSOR = PTx3;             // CS_33810_2 = 1 ---- PTC3
    delay_us(20);                                    //aguarda 20 us

    /* Mantem o SPI do MC33810 ativo */
    if(numero_ci == 1) GPIOA_PCOR = PTx14;           // CS_33810_1 = 0 ---- PTA14
    if(numero_ci == 2) GPIOC_PCOR = PTx3;             // CS_33810_2 = 0 ---- PTC3

    while (!flag) {
        while (!SPI_SR && SPI_SR_TFFF_MASK); //Para buffer TX FIFO esvaziar
        SPIO_PUSHR = 0b0000111000000000; // SPI check comand
        while (!(SPI_SR && SPI_SR_TCF_MASK)); //espera a transmissao acabar
        delay_us(1);                           //aguarda 1 us
        while (!(SPI_SR && SPI_SR_RFDF_MASK)); //RX FIFO estar j vazio ~RFDF
        dado = SPIO_POPR;                      //Recebe a resposta MC33810
        if (dado == 0b0000110100001010) {
            flag = 1; //Variavel liberar funcao
        }
        else {
            flag = 0;      //Nao Libera funcao
            i++;          //Incrementa i
            if(i == 30) {
                flag = 1;      //Tenta por 30 vezes
            }
        }
        dado = 0;                      //Zera variável dado
        delay_us(32);                 //aguarda 32 us
    }
}

```

```

/* Requisito de calibracao do MC33810 */
if(numero_ci == 1) GPIOA_PSOR = PTx14; // CS_33810_1 = 1 ---- PTA14
if(numero_ci == 2) GPIOC_PSOR = PTx3; // CS_33810_2 = 1 ---- PTC3
delay_us(32); //aguarda 32 us

/* Mantem o SPI do MC33810 ativo */
if(numero_ci == 1) GPIOA_PCOR = PTx14; // CS_33810_1 = 0 ---- PTA14
if(numero_ci == 2) GPIOC_PCOR = PTx3; // CS_33810_2 = 0 ---- PTC3

/* Modo de Operacao - MC33810 */
while (!SPI0_SR && SPI_SR_TFFF_MASK); //Espera o buffer TX FIFO esvaziar
SPI0_PUSHR = 0b0001111100000000; /* GENERAL PURPOSE GATE DRIVER MODE */
while (!(SPI0_SR && SPI_SR_TCF_MASK)); //Espera a transmissao acabar
delay_us(2); //aguarda 2 us

/* Requisito de calibracao do MC33810 */
if(numero_ci == 1) GPIOA_PSOR = PTx14; // CS_33810_1 = 1 ---- PTA14
if(numero_ci == 2) GPIOC_PSOR = PTx3; // CS_33810_2 = 1 ---- PTC3
delay_us(20); //aguarda 20 us

/* Mantem o SPI do MC33810 ativo */
if(numero_ci == 1) GPIOA_PCOR = PTx14; // CS_33810_1 = 0 ---- PTA14
if(numero_ci == 2) GPIOC_PCOR = PTx3; // CS_33810_2 = 0 ---- PTC3

/* LSD Fault Command - MC33810 */
while (!SPI0_SR && SPI_SR_TFFF_MASK); //Espera buffer TX FIFO esvaziar
SPI0_PUSHR = 0b0010111000000000; //Combina2 mecanismos contra curtos
while (!(SPI0_SR && SPI_SR_TCF_MASK)); //Espera a transmissao acabar
delay_us(2); //aguarda 2 us

if(numero_ci == 1) GPIOA_PSOR = PTx14; // CS_33810_1 = 1 ---- PTA14
if(numero_ci == 2) GPIOC_PSOR = PTx3; // CS_33810_2 = 1 ---- PTC3

}

//FINAL DA FUNCAO
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************

/*
    Funcao : void init_MC33186 (void)
*/
/*
TaskNumber: 24
*/
/*
Entradas : Nenhuma
*/
/*
Saida : Nenhuma
*/
/*
Descricao : Esta função inicia o c.i MC33186 (Ponte H - Borboleta)
A UCM deste projeto possui 1 c.i. que possui a função de
efetuar a interface entre a Válvula borboleta do motor e
MCU utilizando uma lógica de ponte H passeando-se em si-
nais PWM. Esta função inicia o c.i efetuando reset do mes-
e removendo-o de um modo de segurança caso o mesmo esteja.
*/
/*
HARDWARE:
IN1 - FTM1_CH0 - PTA8 - PINO 60 (PWM)
*/

```

```

/*
 *      IN2 - FTM1_CH1 - PTA9 - PINO 61 (PWM)          */
/*
 *      DI1 - FTM2_CHO - PTA10 - PINO 62 (OUTPUT) [0V]   */
/*
 *      DI2 - FTM2_CH1 - PTA10 - PINO 63 (OUTPUT) [5V]    */
/*
 *      COD - PTA12 - PINO 64 (OUTPUT) [0V]            */
/*
 *      SF - PTA13 - PINO 65 (INPUT) [C.I WARNING FLAG]  */
//*****************************************************************/
void init_MC33186 (void) {

    //----- Inicia PONTE H - MC33186 (2Khz PWM 0 - 12500 (0%-100%))
    FTM1_COV = 12500;           // IN1 = 100% PWM 100%
    FTM1_C1V = 12500;           // IN2 = 100% PWM 100%
    GPIOA_PSOR = PTx10;         // DI1 = 1 - PTA10
    GPIOA_PCOR = PTx11;         // DI2 = 0 - PTA11
    GPIOA_PCOR = PTx12;         // COD = 0 - PTA12
    delay_us(10000);           // Aguarda 10ms
    GPIOA_PCOR = PTx10;         // DI1 = 0 - PTA10
    GPIOA_PSOR = PTx11;         // DI2 = 1 - PTA11
} //FINAL DA FUNCAO
//*****************************************************************/
//*****************************************************************/
//*****************************************************************/
//*****************************************************************/
//*****************************************************************/
//*****************************************************************/
//*****************************************************************/
/*
 *      Funcao : void init MCU_pins (void)           */
/*
 *      TaskNumber: 25                                */
/*
 *      Entradas : Nenhuma                          */
/*
 *      Saida   : Nenhuma                          */
/*
 *      Descricao : Esta funcao inicia todos os pinos do MCU para evitar que
 *                  no inicio ou no reset do software algum atuador permaneça
 *                  acionado indevidamente.           */
//*****************************************************************/
void init MCU_pins (void) {

    //PORT A init
    //01      10      20      30
    //012345678901234567890123456789012
    //x xx x1x00000001000 x xxxxxxxxxxxxxxxx
    GPIOA_PDOR = 0x00004010;           //Valores conforme hardware Cartronic v3.0

    //PORT B init
    //01      10      20      30
    //012345678901234567890123456789012
    //xxxxxxxx011xxxx10001xxxxxxxxxxxxxx
    GPIOB_PDOR = 0x00110C00;           //Valores conforme hardware Cartronic v3.0

    //PORT C init
    //01      10      20      30
    //012345678901234567890123456789012
    //00010000000000x0xx1xxxxxxxxxxxxxx
    GPIOC_PDOR = 0x00040008;           //Valores conforme hardware Cartronic v3.0

```

```

//PORT D init
//01      10      20      30
//012345678901234567890123456789012
//0000xxxxx000000xxxxxxxxxxxxxx
GPIOD_PDIR = 0x00000000;                                //Valores conforme hardware Cartronic v3.0

//PORT E init
//01      10      20      30
//012345678901234567890123456789012
//000000000011xxxxxxxxxx0x00xxxx
GPIOE_PDIR = 0x00000C00;                                //Valores conforme hardware Cartronic v3.0

}//FINAL DA FUNCAO
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************

//****************************************************************************
/*
     Funcao  : void escalonador (uint8_t modo, uint16_t *scheduler)
*/
/*
/* TaskNumber: 26
*/
/*
/* Entradas : uint16_t *scheduler (Variavel de entrada da funcao)
/*      : uint8_t modo(Indica se armazena ou retorna o scheduler)
*/
/*
/* Saída   : Ponteiro scheduler atualizado caso seja solicitado
*/
/*
/* Descricao : Esta função atua no RTOS e efetua o escalonamento de todo
/*      sistema através do tempo gerado na interrupção do PDB0()
/*      Gerado para atuar com 1ms de latência.
*/
//****************************************************************************
void escalonador (uint8_t modo, uint16_t *scheduler) {

static uint16_t saida = 0;                                //Memoria para o ponteiro scheduler

    if(!modo)                                              //caso modo = ARMAZENA
        saida = *scheduler;
    else                                                    //caso modo = RETORNA
        *scheduler = saida;

}//FINAL DA FUNCAO
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************

//****************************************************************************
/*
     Funcao  : void RTOS_RUN(void)
*/
/*
/* TaskNumber: 27
*/
/*
/* Entradas : Nenhuma
*/
/*
/* Saída   : Nenhuma
*/
/*
/* Descricao : Esta função é o coração do sistema. Desenvolvida para
/*      atuar como um sistema operacional de tempo real que possui
*/
/*

```



```

        SincronismoSistemaADM();//Efetua controle da Borboleta
    }

/*
-----*/
/*
-----*/
    if (VetorTask[7] == 1) {
        VetorTask[7] = 0;           //Reseta função
        HMI();                     //Envia bloco de dados para HMI
    }
/*
-----*/
/*
-----*/

    if (VetorTask[8] == 1) {
        VetorTask[8] = 0;           //Reseta função
        EstadoAtualMotor();        //Determina estado atual do motor
    }

    j = i;                      //Armazena ultimo estado do RTOS
}//FINAL DA FUNCAO
*****
***** FIM DO ARQUIVO *****
*****
```

COMANDOATUADORES.H

```

/*
*          Escola Politécnica da Universidade de São Paulo
*          POLI - USP - EPUSP
*
*          Mestrado em Engenharia Elétrica
*          Microeletrônica
*
*****Firmware da Unidade de Gerenciamento Eletrônico do motor EA113 2.0L*****
```

```

***** Autor *****
/*
*          Bruno Martin de Alcântara Dias
*/

/*
*  Modulo:      cartronic_v3_2.c
*
*  Descrição:   Biblioteca de Acionamento Atuadores (ComandoAtuadores_task.h)
*
*  Projeto:     UCM - CARTRONIC v3.0
*
*  Compilador:  CodeWarrior for MCU - Version: 10.2 - Build Id:120126
*
*  Hardware:    Cartronic v3.2 32bits - Freescale Kinetis K40 256k
*
*          MCU = K40X256VLQ100
*
*/
```

```

/*
 * Nucleo:      ARM CORTEX M4 32bits
 */
/*
 * Data: <10/05/2014>
 */
/*
 * Historico:      Iniciais          Motivo da Mudança
 */
/*
 *                      do Projetista
 */
/*
 * 10/05/2014        BMAD            Inicial version.
 * 15/05/2014        BMAD            ControlaMainRele()
 * 22/05/2014        BMAD            Injetor e bobina
 * 24/05/2014        BMAD            Sistema Injeção
 * 25/05/2014        BMAD            Sistema Ignicao
 * 28/05/2014        BMAD            Sistema Admissao
 * 06/06/2014        BMAD            Controle Partida 01
 * 19/08/2014        BMAD            Aplicacao Filtro ROT
 * 28/08/2014        BMAD            Sincronismo INJ,IGN
 * 09/09/2014        BMAD            Controle PID borboleta
 * 12/09/2014        BMAD            Alteracao Injecao
 * 12/10/2014        BMAD            Partiu motor 1a vez!
 * 27/10/2012        BMAD            RTOS_VEM [RTOS_BUS]
 * 18/12/2014        BMAD            Controle PID rotacao
 ****
 */

```

* DESCRICAO GERAL DA BIBLIOTECA *

// Esta biblioteca agrupa o conjunto de funções de acionamento e de desacionamento dos atuadores presentes no projeto

```
*****  
*****  
/*          Funcao  : void ControlaLed (uint8_t comando, uint8_t led)      */  
/*  
/* TaskNumber: 41  
/*  
/* Entradas : uint8_t led (Indica qual led sera controlado)  
/*           uint8_t comando (Indica acao a ser executada)  
/*  
/* Saída   : Nenhuma  
/*  
/* Descricao : Esta funcao aciona o led 1 ou led 2 dependendo da variavel  
/*           de comando (ON e OFF) [LED 1 = PTB10 ; LED 2 = PTB11]  
*****  
*****
```

```

void ControlaLed (uint8_t comando, uint8_t led) {

    if (!comando) { //Caso comando = OFF
        if (led)
            GPIOB_PSOR |= PTx10; //Desliga Led1
        if(led == 2)    GPIOB_PSOR |= PTx11; //Desliga Led2
    } else {
        //Caso comando = ON
        if (led)      GPIOB_PCOR |= PTx10; //Aiona Led1
        if (led == 2)  GPIOB_PCOR |= PTx11; //Aiona Led2
    }
} //Final da Funcao
*****
```

```

/*****************/
/*****************/
/*****************/

/*****************/
/*****************/
/*
 *      Funcao  : void ControlaMainRele(uint8_t comando)
 */
/*
 * TaskNumber: 42
 */
/*
 * Entradas : uint8_t comando (Indica acao a ser executada)
 */
/*
 * Saida   : Nenhuma
 */
/*
 * Descricao : Esta funcao aciona o MAIN RELE (RELE BOMBA) dependendo
 *              da variavel de comando (ON e OFF) [MAIN RELE = PTB16]
 *              HARDWARE = PINO 26 UCM [Possui interface eletronica]
 */
/*****************/
/*****************/

void ControlaMainRele (uint8_t comando) {

    RTOS_VEM [MAIN_RELAY_VEM] = comando;           //Atualiza Vetor estados RTOS

    if (!comando) {                                //Caso comando = OFF
        GPIOB_PCOR |= PTx16;                         //Desliga Main Rele
    }else {                                         //Caso comando = ON
        GPIOB_PSOR |= PTx16;                         //Aciona Main Rele
    }
}//Final da Funcao
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*
 *      Funcao  : void ControlaInjetores(uint8_t comando,uint8_t injetor)
 */
/*
 * TaskNumber: 43
 */
/*
 * Entradas : uint8_t injetor(Indica qual injetor sera controlado)
 *             uint8_t comando (Indica acao a ser executada)
 */
/*
 * Saida   : Nenhuma
 */
/*
 * Descricao : Esta funcao aciona os injetores 1,2,3,4,5,6,7 e 8
 *              dependendo da variavel de comando (ON e OFF)
 *              HARDWARE = INJETOR 1 = PTE0(01) [DIN1 - MC33810 1] - PINO 79
 *              HARDWARE = INJETOR 2 = PTE1(02) [DIN2 - MC33810 1] - PINO 59
 *              HARDWARE = INJETOR 3 = PTE2(03) [DIN3 - MC33810 1] - PINO 73
 *              HARDWARE = INJETOR 4 = PTE3(04) [DIN4 - MC33810 1] - PINO 65
 *              HARDWARE = INJETOR 5 = PTE4(07) [DIN1 - MC33810 2] - PINO 41
 *              HARDWARE = INJETOR 6 = PTE5(08) [DIN2 - MC33810 2] - PINO 42
 *              HARDWARE = INJETOR 7 = PTE6(09) [DIN3 - MC33810 2] - PINO 43
 *              HARDWARE = INJETOR 8 = PTE7(10) [DIN4 - MC33810 2] - PINO 44
 */
/*****************/
/*****************/

```

```

void ControlaInjetores(uint8_t comando, uint8_t injetor) {

    RTOS_VEM[injutor + 30] = comando; //Atualiza Vetor estados RTOS

    if(!comando){ //Caso comando = OFF

        switch (injutor) {

            case 1:
            GPIOE_PCOR |= PTx0; //Desliga Injetor 1
            break;

            case 2:
            GPIOE_PCOR |= PTx1; //Desliga Injetor 2
            break;

            case 3:
            GPIOE_PCOR |= PTx2; //Desliga Injetor 3
            break;

            case 4:
            GPIOE_PCOR |= PTx3; //Desliga Injetor 4
            break;

            case 5:
            GPIOE_PCOR |= PTx4; //Desliga Injetor 5
            break;

            case 6:
            GPIOE_PCOR |= PTx5; //Desliga Injetor 6
            break;

            case 7:
            GPIOE_PCOR |= PTx6; //Desliga Injetor 7
            break;

            case 8:
            GPIOE_PCOR |= PTx7; //Desliga Injetor 8
            break;
        }
    } else { //Caso comando = ON
        switch (injutor) {

            case 1:
            GPIOE_PSOR |= PTx0; //Aciona Injetor 1
            break;

            case 2:
            GPIOE_PSOR |= PTx1; //Aciona Injetor 2
            break;

            case 3:
            GPIOE_PSOR |= PTx2; //Aciona Injetor 3
            break;
        }
    }
}

```

```

case 4:
GPIOE_PSOR |= PTx3; //Aiona Injetor 4
break;
```



```

case 5:
GPIOE_PSOR |= PTx4; //Aiona Injetor 5
break;
```



```

case 6:
GPIOE_PSOR |= PTx5; //Aiona Injetor 6
break;
```



```

case 7:
GPIOE_PSOR |= PTx6; //Aiona Injetor 7
break;
```



```

case 8:
GPIOE_PSOR |= PTx7; //Aiona Injetor 8
break;
```



```

}
```



```

} //Final da Funcao
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
```



```

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



```

/*
    Funcao : void ControlaBobinas(uint8_t comando, uint8_t bobina)
*/
/*
TaskNumber: 44
*/
/*
Entradas : uint8_t bobina(Indica qual bobina sera controlada)
        uint8_t comando (Indica acao a ser executada)
*/
/*
Saida : Nenhuma
*/
/*
Descricao : Esta funcao aciona a bobina ED 1,2,3,4,5,6,7 e 8
dependendo da variavel de comando (ON e OFF)
HARDWARE = BOBINA 1 = PTA6(058) - PINO 57
HARDWARE = BOBINA 2 = PTC0(103) - PINO 71
HARDWARE = BOBINA 3 = PTC5(114) - PINO 03
HARDWARE = BOBINA 4 = PTC6(115) - PINO 09
HARDWARE = BOBINA 5 = PTC7(116) - PINO 12
HARDWARE = BOBINA 6 = PTC8(117) - PINO 16
HARDWARE = BOBINA 7 = PTC9(118) - PINO 17
HARDWARE = BOBINA 8 = PTC10(119) - PINO 18
*/
/*
OBS: Esta funcao foi implementada para controla bobina ED, ou seja,
bobinas que possuem o driver de potencia externo ao hardware d UCM
(bobinas com driver de potencia integrado ao seu corpo fisico)
deste modo a UCM envia nivel logico 1 para carregar e nivel logico
zero para desligar ou interromper o tempo de carga.
Esta funcao nao se aplica a bobinas comuns (ID), caso seja
necessario esta funcao deve ser modificada.
*/
*****
```

```

/***********************/
void ControlaBobinas(uint8_t comando, uint8_t bobina) {

    RTOS_VEM [bobina + 40] = comando;           //Atualiza Vetor estados RTOS

    if (!comando) {                           //Caso comando = OFF

        switch (bobina) {

            case 1:
                GPIOA_PCOR |= PTx6;             //Desliga bobina 1
                break;
            case 2:
                GPIOC_PCOR |= PTx0;            //Desliga bobina 2
                break;
            case 3:
                GPIOC_PCOR |= PTx5;            //Desliga bobina 3
                break;
            case 4:
                GPIOC_PCOR |= PTx6;            //Desliga bobina 4
                break;
            case 5:
                GPIOC_PCOR |= PTx7;            //Desliga bobina 5
                break;
            case 6:
                GPIOC_PCOR |= PTx8;            //Desliga bobina 6
                break;
            case 7:
                GPIOC_PCOR |= PTx9;            //Desliga bobina 7
                break;
            case 8:
                GPIOC_PCOR |= PTx10;           //Desliga bobina 8
                break;
        }
    }
    else {                                     //Caso comando = ON

        switch (bobina) {

            case 1:
                GPIOA_PSOR |= PTx6;          //Aciona bobina 1
                break;
            case 2:
                GPIOC_PSOR |= PTx0;          //Aciona bobina 2
                break;
            case 3:
                GPIOC_PSOR |= PTx5;          //Aciona bobina 3
                break;
            case 4:
                GPIOC_PSOR |= PTx6;          //Aciona bobina 4
                break;
            case 5:
                GPIOC_PSOR |= PTx7;          //Aciona bobina 5
                break;
            case 6:
                GPIOC_PSOR |= PTx8;          //Aciona bobina 6
                break;
            case 7:
                GPIOC_PSOR |= PTx9;          //Aciona bobina 7
                break;
        }
    }
}

```

```

        break;
    case 8:
        GPIOC_PSOR |= PTx10; //Aciona bobina 8
        break;
    }
}

}//Final da Funcao
//********************************************************************/
//********************************************************************/
//********************************************************************/
//********************************************************************/
//********************************************************************/
/*
    Funcao : void ControlaSensores(uint8_t comando)
*/
/*
/* TaskNumber: 45
*/
/*
/* Entradas : uint8_t comando (Indica acao a ser executada)
*/
/*
/* Saida : Nenhuma
*/
/*
/* Descricao : Esta funcao aciona a tensao positiva aos sensores FASE,
/*             TPS e PEDAL dependendo da variavel de comando (ON e OFF)
/* [CONTROL_SENSOR MCU = PTA26]
/* HARDWARE = PINO 77 UCM [Possui interface eletronica]
//********************************************************************/
//********************************************************************/
void ControlaSensores (uint8_t comando) {

    RTOS_VEM [POSITIVO_SENSOR_VEM] = comando; //Atualiza Vetor estados RTOS

    if (!comando) { //Caso comando = OFF
        GPIOA_PCOR |= PTx26; //Desliga 5V sensores
    }else { //Caso comando = ON
        GPIOA_PSOR = PTx26; //Aciona 5V sensores
    }

}

}//Final da Funcao
//********************************************************************/
//********************************************************************/
//********************************************************************/
//********************************************************************/
/*
    Funcao : void ControlaSensorRotacao(uint8_t comando)
*/
/*
/* TaskNumber: 46
*/
/*
/* Entradas : uint8_t comando (Indica acao a ser executada)
*/
/*
/* Saida : Nenhuma
*/
/*
/* Descricao : Esta funcao aciona a tensao positiva de referencia do
/*             sensor rotacao dependendo da variavel de comando(ON e OFF)
/* [CONTROL_ROT67 MCU = PTA25]
/* HARDWARE = PINO 76 UCM [Possui interface eletronica]
*/
*/

```

```
*****
/*
void ControlaSensorRotacao (uint8_t comando) {

    RTOS_VEM [POSITIVO_RPM_VEM] = comando;      //Atualiza Vetor estados RTOS

    if (!comando) {                            //Caso comando = OFF
        GPIOA_PSOR = PTx25;                   //Desliga Referencia 2.5V Sensor RPM
    }else {
        GPIOA_PCOR = PTx25;                   //Aciona Referencia 2.5V Sensor RPM
    }
}//Final da Funcao
*****
*/
*****
```

* FIM DO ARQUIVO *

```
*****
```

AQUISICAOSENSORES_TASKS.H

```
*****
/*
* Escola Politécnica da Universidade de São Paulo
* POLI - USP - EPUSP
*
* Mestrado em Engenharia Elétrica
* Microeletrônica
*
*****Firmware da Unidade de Gerenciamento Eletrônico do motor EA113 2.0L*****
```

```
*****
/* Autor ****/
/*
* Bruno Martin de Alcântara Dias
*****
*/
```

```
*****
/* Modulo: cartronic_v3_2.c */
/*
* Descrição: Biblioteca de Aquisição Sensores (AquisicaoSensores_task.h)
*/
/*
* Projeto: UCM - CARTRONIC v3.0
*/
/*
* Compilador: CodeWarrior for MCU - Version: 10.2 - Build Id:120126
*/
/*
* Hardware: Cartronic v3.2 32bits - Freescale Kinetis K40 256k
*/
/*
* MCU = K40X256VLQ100
*/
/*
* Nucleo: ARM CORTEX M4 32bits
*/
/*
* Data: <10/05/2014>
*/
/*
* Historico: Iniciais Motivo da Mudança
* do Projetista
* 10/05/2014 BMAD Inicial version.
* 24/05/2014 BMAD Sistema Injecao
*/
*/
```

```
/* 25/05/2014          BMAD      Sistema Ignicao
/* 28/05/2014          BMAD      TPS1, PEDAL
/* 06/06/2014          BMAD      Controle Partida 01
/* 19/08/2014          BMAD      Aplicacao Filtro ROT
/* 28/08/2014          BMAD      Sincronismo INJ,IGN
/* 28/08/2014          BMAD      TPS2
/* 09/09/2014          BMAD      Controle PID borboleta
/* 09/09/2014          BMAD      Pedal2
/* 12/09/2014          BMAD      Alteracao Injecao
/* 12/10/2014          BMAD      Partiu motor 1a vez!
/* 30/10/2014          BMAD      RTOS_VEM [RTOS_BUS]
/* 18/12/2014          BMAD      Controle PID rotacao
/*****
/***** DESCRICAO GERAL DA BIBLIOTECA
*****
```

```

// Esta biblioteca agrupa o conjunto de funcoes de aquisicao dos sensores
// de gerencianento eletronico do motor
//****************************************************************************
/*
 *      Funcao : uint8_t SensorTPS1 (void)
 */
/*
 * TaskNumber: 61
 */
/*
 * Entradas : Nenhuma
 */
/*
 * Saída : Sinal convertido do Sensor TPS1 (Variacao negativa)
 */
/*
 * Descricao : Esta função utiliza o conversor ADC0 em seu canal 12 para
 *              retornar o valor digital do sensor apos a conversao.
 */
/*
 * HARDWARE :
 */
/*
 * PINO83 - PTB2 - ADC0_SE12 (single ended conversion)
 */
//****************************************************************************
/*
 * uint8_t SensorTPS1 (void) {
 */

    uint8_t TPS;

    ADC0_SC1A = 12 & ADC_SC1_ADCH_MASK;           // Inicia conversao canal SE12
    while ((ADC0_SC2 & ADC_SC2_ADACT_MASK));     // Aguarda ADC em progresso
    while (!(ADC0_SC1A && ADC_SC1_COCO_MASK)); // Aguarda conversao terminar
    TPS = ADC0_RA;                                // Captura o resultado

    TPS = (-0.48 * TPS) + 107.18;                 // Converte Set point de 0% a 100%

    return TPS;
}

//Final da Funcao
*****
*****
```

```

/******************************************************************************/
/*
 *      Funcao : uint8_t SensorTPS2 (void)
 */
/*
 * TaskNumber: 62
 */
/*
 * Entradas : Nenhuma
 */
/*
 * Saída : Sinal convertido do Sensor TPS2 (Variacao positiva)
 */
/*
 * Descrição : Esta função utiliza o conversor ADC0 em seu canal 13 para
 *              retornar o valor digital do sensor após a conversão.
 */
/*
 * HARDWARE :
 */
/*
 * PINO84 - PTB3 - ADC0_SE13 (single ended conversion)
 */
/************************************************************************
/************************************************************************

uint8_t SensorTPS2 (void) {

    uint8_t TPS;

    ADC0_SC1A = 13 & ADC_SC1_ADCH_MASK;           // Inicia conversão canal SE13
    while ((ADC0_SC2 & ADC_SC2_ADACT_MASK));     // Aguarda ADC em progresso
    while (!(ADC0_SC1A && ADC_SC1_COCO_MASK));   // Aguarda conversão terminar
    TPS = ADC0_RA;                                // Captura o resultado

    TPS = (0.488 * TPS) - 17.09;                  // Converte Set point de 0% a 100%

    return TPS;
}//Final da Função
/************************************************************************
/************************************************************************
/************************************************************************
/************************************************************************

/*
 *      Funcao : uint8_t SensorPEDAL1 (void)
 */
/*
 * TaskNumber: 63
 */
/*
 * Entradas : Nenhuma
 */
/*
 * Saída : Sinal convertido do Sensor PEDAL1
 */
/*
 * Descrição : Esta função utiliza o conversor ADC1 em seu canal 10 para
 *              retornar o valor digital do sensor após a conversão.
 */
/*
 * HARDWARE :
 */
/*
 * PINO85 - PTB4 - ADC1_SE10 (single ended conversion)
 */
/************************************************************************
/************************************************************************

uint8_t SensorPEDAL1 (void) {

    uint8_t PEDAL;

    ADC1_SC1A = 10 & ADC_SC1_ADCH_MASK;           // Inicia conversão canal SE10
    while ((ADC1_SC2 & ADC_SC2_ADACT_MASK));     // Aguarda ADC em progresso
    while (!(ADC1_SC1A && ADC_SC1_COCO_MASK));   // Aguarda conversão terminar
}

```



```

/*
 *      HARDWARE :
 *          PINO59 - PTA7 - ADC0_SE10 (single ended conversion)
 */
/*****************/
/*uint8_t SensorMAP (void) {

    uint8_t MAP = 0;

    ADC0_SC1A = 10 & ADC_SC1_ADCH_MASK;           // Inicia conversao canal SE10
    while ((ADC0_SC2 & ADC_SC2_ADACT_MASK));     // Aguarda ADC em progresso
    while (!(ADC0_SC1A && ADC_SC1_COCO_MASK));   // aguarda conversao terminar
    MAP = ADC0_RA;                                // Captura o resultado;

    MAP = (0.0435 * MAP) + 1.08;                  //Converte Saida em (kPa / 10)

    return MAP;
} //Final da Funcao
/*****************/
/*****************/
/*      Funcao : uint8_t SensorTAGUA (void)
 */
/* TaskNumber: 66
 */
/* Entradas : Nenhuma
 */
/* Saída : Sinal convertido do Sensor Temp AGUA
 */
/* Descricao : Esta funcao utiliza o conversor AD00 em seu canal 8 para
 *             retornar o valor digital do sensor apois a conversao.
 */
/* HARDWARE :
 */
/*      PINO81 - PTB0 - ADC0_SE8 (single ended conversion)
 */
/*****************/
/*uint8_t SensorTAGUA (void) {

    uint8_t TAGUA = 0;

    ADC0_SC1A = 8 & ADC_SC1_ADCH_MASK;           // Inicia conversao canal SE8
    while ((ADC0_SC2 & ADC_SC2_ADACT_MASK));     // Aguarda ADC em progresso
    while (!(ADC0_SC1A && ADC_SC1_COCO_MASK));   // Aguarda conversao terminar
    TAGUA = ADC0_RA;                                // Captura o resultado

    // Converte a saida do ADC para retornar valor em graus Celsius divido por 10
    // Ex:Se retornar 23 graus Celsius entao a funcao retornara "(int)23/10 = 2"
    if (TAGUA >= 85) TAGUA = ((-0.295 * TAGUA) + 75)/10;
    else TAGUA = (135 - TAGUA) / 10;

    return TAGUA;
} //Final da Funcao
/*****************/
/*****************/

```

```

/*****************/
/*****************/
/*      Funcao  : uint8_t SensorTAR (void) */
/*
/* TaskNumber: 67
/*
/* Entradas : Nenhuma
/*
/* Saída   : Sinal convertido do Sensor Temp AR
/*
/* Descrição : Esta função utiliza o conversor AD00 em seu canal 9 para
/*             retornar o valor digital do sensor após a conversão.
/*
/* HARDWARE :
/*     PINO82 - PTB1 - ADC0_SE9 (single ended conversion)
/*****************/
/*****************/
uint8_t SensorTAR (void) {

    uint8_t TAR = 0;

    ADC0_SC1A = 9 & ADC_SC1_ADCH_MASK;           // Inicia conversão canal SE9
    while ((ADC0_SC2 & ADC_SC2_ACTL_MASK));      // Aguarda ADC em progresso
    while (!(ADC0_SC1A && ADC_SC1_COCO_MASK)); // Aguarda conversão terminar
    TAR = ADC0_RA;                                // Captura o resultado

    // Converte a saída do ADC para retornar valor em graus Celsius dividido por 10
    // Ex: Se retornar 23 graus Celsius então a função retornará "(int)23/10 = 2"
    if (TAR >= 85) TAR = ((-0.295 * TAR) + 75)/10;
    else TAR = (135 - TAR) / 10;

    return TAR;
}//Final da Função
/*****************/
/*****************/
/*****************/
/*****************/
/*****************/
/*
*          FIM DO ARQUIVO
*/
/*****************/

```

INTERRUPCOES_TASKS.H

```

/*****************/
/*
*          Escola Politécnica da Universidade de São Paulo
*          POLI - USP - EPUSP
*/
/*
*          Mestrado em Engenharia Elétrica
*          Microeletrônica
*/
/*
*          Firmware da Unidade de Gerenciamento Eletrônico do motor EA113 2.0L*****
*/

/****************** Autor *****/
/*
*/
/*
*          Bruno Martin de Alcântara Dias
*/

```

```
*****
/*
/* Modulo: cartronic_v3_2.c */
/*
/* Descrição: Biblioteca de tarefas relacionadas as interrupcoes (Interrupcoes_tasks.h) */
/*
/* Projeto: UCM - CARTRONIC v3.0 */
/*
/* Compilador: CodeWarrior for MCU - Version: 10.2 - Build Id:120126 */
/*
/* Hardware: Cartronic v3.2 32bits - Freescale Kinetis K40 256k */
/*
/*
MCU = K40X256VLQ100
/*
/*
/* Nucleo: ARM CORTEX M4 32bits */
/*
/*
Data: <10/05/2014>
/*
/*
/* Historico: Iniciais Motivo da Mudança */
/* do Projetista */
/* 10/05/2014 BMAD Inicial version. */
/* 24/05/2014 BMAD Conclusao SistemaINJ */
/* 25/05/2014 BMAD Conclusao SistemaIGN */
/* 28/05/2014 BMAD Sistema Admissao */
/* 06/06/2014 BMAD Controle Partida 01 */
/* 07/06/2014 BMAD Init MCU pins */
/* 19/08/2014 BMAD Aplicacao Filtro ROT */
/* 22/08/2014 BMAD Inclusao Escalonador */
/* 28/08/2014 BMAD Sincronismo INJ,IGN */
/* 12/09/2014 BMAD Alteracao Injecao */
/* 12/10/2014 BMAD Partiu motor 1a vez! */
/* 27/10/2014 BMAD RTOS_VEM [RTOS_BUS] */
/* 29/10/2014 BMAD Sistema Admissao */
/* 04/11/2014 BMAD Inclusao TinjFlag */
/* 05/11/2014 BMAD Atualizacao task 85,86 */
/* 18/12/2014 BMAD Controle PID rotacao */
*****
```

```
*****
* DESCRICAO GERAL DA BIBLIOTECA */
*****
```

```
//Esta biblioteca agrupa o conjunto de tarefas que sao comuns as interrupcoes
// de todo o projeto
*****
```

```
*****
/* Funcao : void HabilitaInterrupcao(uint8_t NomeInterrupcao) */
/*
/* TaskNumber: 81 */
/*
/* Entradas : uint8_t NomeInterrupcao (Indica a interrupcao a controlar)
/*
/* Saida : Nenhuma */
*/
```



```

/*****************/
void DesabilitaInterrupcao(uint8_t NomeInterrupcao) {

    switch (NomeInterrupcao) {

        case 0:                                //INTERRUPCAO = LINHA15 - PTA24
            PORTA_PCR24 &= 0x0FFF;             //Desabilita interrupcao
            break;
        case 1:                                //INTERRUPCAO = USB1 - "UART4"
            UART4_C2 &= 0xDF;              //Zera RIE Desabilita interrupcao
            break;
        case 2:                                //INTERRUPCAO = SENSOR ROTACAO - PTD4
            PORTD_PCR4 &= 0x0FFF;           //Desabilita interrupcao
            break;
        case 3:                                //INTERRUPCAO = PIT0
            PIT_TCTRL0 &= (!PIT_TCTRL_TIE_MASK); //Desabilita interrupcao
            break;
        case 4:                                //INTERRUPCAO = PIT1
            PIT_TCTRL1 &= (!PIT_TCTRL_TIE_MASK); //Desabilita interrupcao
            break;
        case 5:                                //INTERRUPCAO = PIT2
            PIT_TCTRL2 &= (!PIT_TCTRL_TIE_MASK); //Desabilita interrupcao
            break;
        case 6:                                //INTERRUPCAO = PIT3
            PIT_TCTRL3 &= (!PIT_TCTRL_TIE_MASK); //Desabilita interrupcao
            break;
        case 7:                                //INTERRUPCAP = FTM2
            FTM2_SC &= 0xBF;               //Desabilita interrupcao
            break;
        case 8:                                //INTERRUPCAO = PDB0 (RTOS SCHEDULER)
            PDB0_SC |= PDB_SC_SWTRIG_MASK;   //software trigger
            PDB0_SC &= (~PDB_SC_PDBIE_MASK); //Desabilita interrupcao
            break;
        case 9:                                //INTERRUPCAO = SENSOR FASE - PTD5
            PORTD_PCR5 &= 0x0FFF;           //Desabilita interrupcao
            break;
    }
}

//Final da Funcao
/*****************/
/*          Funcao  : void Vetor_Linha15(void) */
/*          TaskNumber: 83 */
/*          Entradas : Nenhuma */
/*          Saida   : Nenhuma */
/*          Descricao : Esta funcao ocorre na interrupcao externa do pino que
/*                         recebe o sinal de acinamento da linha 15. Este sinal eh
/*                         vital para o funcionamento do sistema. A interrupcao
/*                         ocorre inicialmente na borda de subida quando se aciona
/*                         a chave no comutador de ignicao. Neste instante a UCM deve
/*                         */

```

```

/*
    preparar todo o sistema deixando o motor preparado e
    aguardando o motor comecar a girar pelo motor de partida.
    Ao final da execucao de toda preparacao a interrupcao eh
    alterada para borda de descida aguardando a chave de
    ignicao ser desligada. Quando a chave eh desligada a UCM
    deve memorizar os dados para que seja usada na proxima
    partida iniciando todo o processo.
*/
DADOS DA INTERRUPCAO
INTERRUPCAO = EXTERNA (AMBAS AS BORDAS)
PERIODO = EM FUNCAO DA LINHA 15
HARDWARE = PTA24
******/



void Vetor_Linha15(void) {

    static uint8_t estado = 0;          // ESTADO 0 - DESLIGADO | ESTADO 1 - LIGADO
    uint8_t i = 0;                     // Variavel indexadora da tarefa

    estado = !estado;                //inverte estado informando ON e OFF

    if (estado) {                    // Executa se LINHA 15 acionada

        ControlaLed(ON,1);           //Liga Led 1

        init_MC33810 (1);           //Prepara Interface Injetor
        init_MC33186 ();            //Inicia Driver da Borboleta
        ControlaMainRele(ON);        //Aciona Main Rele (Bomba)
        ControlaSensores(ON);        //Liga 5V sensores (FASE,TPS,PEDAL)
        ControlaSensorRotacao(ON);   //Liga 2,5V sensor rotacao pino 67

        DesabilitaInterrupcao(LINHA15); //Desabilita interrupcao
        PORTA_PCR24 |= (PORT_PCR IRQC_MASK & 0xA0000);           //Habilita borda descida

        HabilitaInterrupcao(ISR_PDB0);          //Habilita interrupcao PDB0
        HabilitaInterrupcao(ISR_FADE);           //Habilita interrupcao PTD5

        RTOS_VEM [LINHA15_VEM] = ON;             //Atualiza vetor de estado RTOS

    }else {                           // Executa se LINHA 15 desligada

        RTOS_VEM [LINHA15_VEM] = OFF;          //Atualiza vetor de estado RTOS
        ControlaLed(OFF,1);                  //Desliga Led1
        ControlaMainRele(OFF);               //Desliga Main Rele (Bomba)
        HabilitaInjetores(DESABILITA,1);     //Desabilita injetore 1,2,3 e 4
        DesabilitaInterrupcao(ISR_PDB0);      //Desabilita Interrupcao PDB0
        DesabilitaInterrupcao(ISR_FADE);       //Desabilita Interrupcao FASE
        DesabilitaInterrupcao(ISR_ROTACAO);    //Desabilita Interrupcao ROTACAO

        Vetor_SensorRotacao(OFF, 0);          //Reseta valores estaticos da funcao
        init_engine(OFF);                   //Reseta valores da interrupcao FASE
        Vetor_PIT2(0, OFF);                 //Reseta funcao de controle Bobina 1
        Vetor_PIT3(0, OFF);                 //Reseta funcao de controle Bobina 2
        Rotacao(OFF,0,0);                  //Reseta funcao de calculo da rotacao

        FTM1_COV = 12500; //PWM 100%//Certifica que a borboleta de se desligara
        FTM1_C1V = 12500; //PWM 100% //Certifica que a borboleta de se desligara
    }
}

```

```

delay_us(100000);           //Delay para zerar o sistema

init MCU_pins();           //Reseta pinos em sua posicao inicial

ControlaSensores(OFF);     //Desliga 5V sensores (FASE,TPS,PEDAL)
ControlaSensorRotacao(OFF); //Desliga 2,5V sensor rotacao pino 67

delay_us(3000000);          //Delay para zerar o sistema 3s

ControlaInjetores(OFF,1);   //Garante todos os injetores desligados
ControlaInjetores(OFF,2);
ControlaInjetores(OFF,3);
ControlaInjetores(OFF,4);

ControlaBobinas(OFF,1);     //Garante todas as bobinas desligadas
ControlaBobinas(OFF,2);
ControlaBobinas(OFF,3);
ControlaBobinas(OFF,4);

for (i = 0; i <= 21; i++) {
    EnviaDadoUSB1(0);        //Reseta valores da interface HMI
    delay_us(10000);          //Aguarda 10ms
}
RTOS_VEM [REGIME_VEM] = 0;   //Atualiza Estado Atual do motor

DesabilitaInterrupcao(LINHA15);
//Desabilita interrupcao
PORTA_PCR24 |= (PORT_PCR_IQC_MASK & 0x90000);           //Habilita borda subida
}

} //Final da Funcao
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************

/*
*       Funcao : void Vetor_SensorRotacao(void) *
*       TaskNumber: 84 *
*       Entradas : Nenhuma *
*       Saída : Nenhuma *
*       Descrição : Esta função ocorre na interrupção externa do pino que *
*                   recebe o sinal do sensor de rotação. Para o sistema de *
*                   gerenciamento do motor, esta é a função mais importante. *
*                   Esta função é responsável por todo o sincronismo do *
*                   motor e sua metodologia de operação segue o conceito de *
*                   evento trigger *
*       DADOS DA INTERRUPÇÃO *
*       INTERRUPÇÃO = EXTERNA (BORDA SUBIDA) *
*       PERÍODO = EM FUNÇÃO DO SINAL DE ROTAÇÃO DO MOTOR *
*       HARDWARE = PTD4 *
*
*       OBS: Como a função deve estar no vetor de interrupção do PORT D *
*       Deve-se verificar primeiramente se a interrupção realmente *
*       ocorreu no pino PTD4. *
*/

```

```

/***********************/
/***********************/

void Vetor_SensorRotacao(uint8_t TCTRL, char *input) {

    static char dente = 0;                                //Variavel indicacao do dente atual
    static char volta = 0;                               //Sincronismo injecao sequencial
    static uint16_t t_dente = 0;                         //Tempo do dente atual
    static uint32_t TempoCarga = 400000;                //Tempo de carga da bobina de ign
    static uint8_t dente_ref_ign14 = 0;                  //Dente de referencia ignicao 1 e 4
    static uint8_t dente_ref_ign23 = 0;                  //Dente de referencia ignicao 2 e 3
    static uint8_t dente_ref_inj14 = 0;                  //Dente de referencia injecao 1 e 4
    static uint8_t dente_ref_inj23 = 0;                  //Dente de referencia injecao 2 e 3
    static uint8_t dente_final_inj14 = 0;                //Dente de encerramento de injecao
    static uint8_t dente_final_inj23 = 0;                //Dente de encerramento de injecao
    static uint8_t synchro14 = 0;                        //Correcao do sincronismo de injecao
    static uint8_t synchro23 = 0;                        //Correcao do sincronismo de injecao
    static uint8_t RPM = 0;                             //Rotacao do motor
    static uint8_t Avanco = 0;                          //Avanco de ignicao
    static uint8_t Tinj = 0;                            //Tempo de injecao [us/100]
    static uint8_t iTinj = 0;                           //Indexadora do tempo de injecao
    static uint8_t TinjMaximo = 0;                      //Variavel que limite Tinj em funcao de RPM
    static uint8_t TinjRestante = 0;                    //Carrega do tempo de injecao restante
    static uint8_t TinjFlag = 0;                         //Indica necessidade de acionar injetor gêmeo

    switch (TCTRL) {                                  //Verifica se a funcao ON, OFF ou ATUALIZA

        case 1:                                     //Caso funcao ON

            t_dente = FTM0_CNT;                     //Captura tempo do Dente
            FTM0_CNT = 1;                          //Zera contador para contar novo Dente

            dente++;                                //Incrementa Dente
            RTOS_VEM[DENTE_VEM] = dente;           //Atualiza Vetor de Estados RTOS

            if (dente == 59) dente = 1;             //Limita dente caso haja ruido no sinal
            //para que nao ocorra perda do sincronismo

            if(dente == 1) {                      //Executa caso a falha tenha ocorrido
                IdentificaFase(&volta);           //Atualiza a fase do motor para injecao sequencial
            }
    ////////////////////////////////Efetua Atualizacao dos parametros de Ignicao e Injecao
    ////////////////////////////////Efetua Atualizacao dos parametros de Ignicao e Injecao

    switch (dente) { //Avalie qual o dente atual do sinal de rotacao

        case 2:                                     //Caso dente = 2
        dente_final_inj14 = DenteFINAL_INJ14[RPM]; //Determina Dente final de injecao
        dente_final_inj23 = DenteFINAL_INJ23[RPM]; //Determina Dente final de injecao
        break;

        case 14:                                    //Dente PMS cilindro 1 e 4
        dente_ref_ign14 = DenteRef_IGN1 [RPM][Avanco]; //Atualiza Dente referencia IGN
        break;
    }
}

```



```

///////////////////////////////Sincronismo INJECAO //////////////////
if (dente == dente_final_inj14) { // Executa quando acaba um ciclo de inj. e carrega proximo
    iTinj = (Tinj - 30)/10;           //Ajusta Tinj para indexar matriz a seguir
    dente_ref_inj14 = DenteRef_INJ14[RPM][iTinj];
                                //Atualiza novo dente de referencia injetores 1 e 4
    if(dente_ref_inj14 >= 12) synchro14 = 1;
                                //Avalia sincronismo de volta (ajuste injecao)
    else synchro14 = 0;
}

if (dente == dente_final_inj23) { // Executa quando acaba um ciclo de inj. e carrega proximo
    iTinj = (Tinj - 30)/10;           //Ajusta Tinj para indexar matriz a seguir
    dente_ref_inj23 = DenteRef_INJ23[RPM][iTinj];
                                //Atualiza novo dente de referencia injetores 3 e 2
    if(dente_ref_inj23 >= 42) synchro23 = 1;
                                //Avalia sincronismo de volta (ajuste injecao)
    else synchro23 = 0;
}

//////////////////Disparo Sistema Injecao Banco 1 e 4////////////////
if (dente == dente_ref_inj14){ // Executa quando for o dente de referencia injetores 1 e 4
    if(!TinjFlag){ //Executa caso Tinj menor que TinjMaximo
        PIT_LDVAL0 = Tinj / 0.0002;      // Carrega valor de t_inj do PIT0
        PIT_TCTRL0 |= PIT_TCTRL_TIE_MASK; // Habilita interrupcao
        PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK; // Habilita PIT0
        if (!volta) { //Executa caso VOLTA = 0
            if (!synchro14) ControlaInjetores(ON,1); //Aciona Injetor 1
            else ControlaInjetores(ON,4); //Aciona Injetor 4
        } else {
            //Executa caso VOLTA = 1
            if (!synchro14) ControlaInjetores(ON,4); //Aciona Injetor 4
            else ControlaInjetores(ON,1); //Aciona Injetor 1
        }
    } //Final Else if (!volta)

    } else { //Executa caso Tinj maior que TinjMaximo
        PIT_LDVAL0 = TinjRestante / 0.0002; // Carrega valor de t_inj
        PIT_TCTRL0 |= PIT_TCTRL_TIE_MASK; // Habilita interrupcao
        PIT_TCTRL0 |= PIT_TCTRL_TEN_MASK; // Abilita PIT0
        ControlaInjetores(ON,1); //Aciona injetor 1
        ControlaInjetores(ON,4); //Aciona injetor 4

        if (!volta) { //Executa caso VOLTA = 0
            if (!synchro14) Vetor_PIT0(&Tinj, 4); //Indica injetor auxiliar e Tinj
            else Vetor_PIT0(&Tinj, 1); //Indica injetor auxiliar e Tinj
        } else {
            //Executa caso VOLTA = 1
            if (!synchro14) Vetor_PIT0(&Tinj, 1); //Indica injetor auxiliar e Tinj
            else Vetor_PIT0(&Tinj, 4); //Indica injetor auxiliar e Tinj
        }
    } //Final Else if (!volta)
} //Final Else if (!TinjFlag)
} //Final if(dente == dente_ref_inj14)

```

```
///////////////////////////////Disparo Sistema Injecao Banco 2 e 3////////////////////
```

```
if (dente == dente_ref_inj23){ // Executa quando for o dente de referencia injetores 1 e 4
    if (!TinjFlag){ //Executa caso Tinj menor que TinjMaximo
        PIT_LDVAL1 = Tinj / 0.0002; // Carrega valor de t_inj do PIT1
        PIT_TCTRL1 |= PIT_TCTRL_TIE_MASK; //Habilita interrupcao
        PIT_TCTRL1 |= PIT_TCTRL_TEN_MASK; // Habilita PIT1

        if (!volta) { // Executa caso VOLTA = 0
            if (!synchro23) ControlalInjetores(ON,2); //Aiona Injetor 2
            else ControlalInjetores(ON,3); //Aiona Injetor 3
        } else { // Executa caso VOLTA = 1
            if (!synchro23) ControlalInjetores(ON,3); //Aiona Injetor 3
            else ControlalInjetores(ON,2); //Aiona Injetor 2
        } //Final Else if (!volta)

        else { //Executa caso Tinj maior que TinjMaximo
            PIT_LDVAL1 = TinjRestante / 0.0002; // Carrega valor de t_inj do PIT1
            PIT_TCTRL1 |= PIT_TCTRL_TIE_MASK; //Habilita interrupcao
            PIT_TCTRL1 |= PIT_TCTRL_TEN_MASK; //Habilita PIT1
            ControlalInjetores(ON,2); //Aiona injetor 2
            ControlalInjetores(ON,3); //Aiona injetor 3

            if (!volta) { // Executa caso VOLTA = 0
                if (!synchro23) Vetor_PIT1(&Tinj, 3); //Indica injetor auxiliar e Tinj
                else Vetor_PIT1(&Tinj, 2); //Indica injetor auxiliar e Tinj
            } else { // Executa caso VOLTA = 1
                if (!synchro23)Vetor_PIT1(&Tinj, 2); //Indica injetor auxiliar e Tinj
                else Vetor_PIT1(&Tinj, 3); //Indica injetor auxiliar e Tinj
            } //Final Else if (!volta)
        } //Final Else if (!TinjFlag)
    } //Final if(dente == dente_ref_inj23)
///////////////////////////////Disparo Sistema Injecao Banco 2 e 3////////////////////
```

break; //Final Case funcao ON = case:1

```
case 0: //Caso TCTRL = OFF
dente = 0; //Variavel indicacao do dente atual
volta = 0; //Sincronismo injecao sequencial
t_dente = 0; //Tempo do dente atual
TempoCarga = 400000; //Tempo de carga da bobina de ign
dente_ref_ign14 = 0; //Dente de referencia ignicao 1 e 4
dente_ref_ign23 = 0; //Dente de referencia ignicao 2 e 3
dente_ref_inj14 = 0; //Dente de referencia injecao 1 e 4
dente_ref_inj23 = 0; //Dente de referencia injecao 2 e 3
dente_final_inj14 = 0; //Dente de encerramento de injecao
dente_final_inj23 = 0; //Dente de encerramento de injecao
synchro14 = 0; //Correcao do sincronismo de injecao
synchro23 = 0; //Correcao do sincronismo de injecao
RPM = 0; //Rotacao do motor
Avanco = 0; //Avanco de ignicao
Tinj = 0; //Tempo de injecao [us/100]
iTinj = 0; //Indexadora do tempo de injecao
TinjMaximo = 0; //Variavel que limita Tinj em funcao de RPM
TinjRestante = 0; //Carrega do tempo de injecao restante
```

```

TinjFlag = 0;                                //Indica necessidade de acionar injetor gêmeo
break;                                     //Final Case função OFF

case 2:                                    //Caso Task atualiza Dente TCTRL = 2
    dente = *input;
break;
}                                              //Final Switch TCTRL

}//Final da Função
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************

//****************************************************************************
/*
    Função : void Vetor_PIT0(uint8_t *Tinj, uint8_t Injetor)
*/
/*
/* TaskNumber: 85
*/
/*
/* Entradas : uint8_t Injetor - Indica qual é o injetor auxiliar atual
/*     uint8_t *Tinj - Ponteiro para tempo de injeção auxiliar
*/
/*
/* Saída : Nenhuma
*/
/*
/* Descrição : Esta função ocorre na interrupção do PIT0. O PIT 0 é usado exclusivamente para controle do tempo dos injetores 1 e 4.
/* O PIT é carregado com um novo valor na função de sincronismo do sistema de injeção e decremente sua contagem até interromper quando atinge zero. No momento da interrupção, a única função desta tarefa é desligar os injetores.
/* Quando o sincronismo determinar que há necessidade de acionar o injetor auxiliar (gêmeo) a função será chamada com a entrada Injetor indicando qual injetor é o auxiliar. A função será chamada apenas para que se memorize o estado atual e o valor de injeção restante
*/
/*
/* DADOS DA INTERRUPÇÃO
/* INTERRUPÇÃO = SOFTWARE (TIMER DECREMENTA ATÉ ZERO)
/* PERÍODO = EM FUNÇÃO DO TEMPO DE INJEÇÃO
/* HARDWARE = *-*-
*/
//****************************************************************************
//****************************************************************************

void Vetor_PIT0(uint8_t *Tinj, uint8_t Injetor) {

    static uint8_t estado = 0;                      //Variável de estado e memória da tarefa
    static uint8_t Tinj14 = 0;                       //Memória Tempo injeção restante

    if (!Injetor) {                                //Caso injetores sem necessidade de limite
        PIT_TFLG0 |= PIT_TFLG_TIF_MASK;             //Limpa flag escrevendo 1 TIF
        PIT_TCTRL0 &= (!PIT_TCTRL_TEN_MASK);         //Desabilita PIT0
        PIT_TCTRL0 &= (!PIT_TCTRL_TIE_MASK);         //Desabilita interrupção
    }

    if (!estado) {
        ControlaInjetores(OFF,1);                  //Desliga injetor 1
        ControlaInjetores(OFF,4);                  //Desliga injetor 4
    } else {

        switch (estado) {
            case 1:

```



```

void Vetor_PIT1(uint8_t *Tinj, uint8_t Injetor) {

    static uint8_t estado = 0;                                //Variavel de estado e memoria da tarefa
    static uint8_t Tinj23 = 0;                             //Memoria Tempo injecao restante

    if (!Injetor) {                                         //Caso injetores sem necessidade de limite
        PIT_TFLG1 |= PIT_TFLG_TIF_MASK;                      //Limpa flag escrevendo 1 TIF
        PIT_TCTRL1 &= (!PIT_TCTRL_TEN_MASK);                  //Desabilita PIT1
        PIT_TCTRL1 &= (!PIT_TCTRL_TIE_MASK);                  //Desabilita interrupcao

        if (!estado) {
            ControlaInjetores(OFF,2);                         //Desliga injetor 2
            ControlaInjetores(OFF,3);                         //Desliga injetor 3
        } else {

            switch (estado) {
                case 2:
                    ControlaInjetores(OFF,2);                   // Desliga injetor gêmeo (auxiliar)
                    PIT_LDVAL1 = Tinj23 / 0.0002;               // Carrega valor de t_inj do PIT1
                    PIT_TCTRL1 |= PIT_TCTRL_TIE_MASK;           // Habilita interrupcao
                    PIT_TCTRL1 |= PIT_TCTRL_TEN_MASK;           // Habilita PIT1
                    estado = 0;                                // Reseta variavel estado
                    Tinj23 = 0;                               // Reseta variavel estado
                    break;

                case 3:
                    ControlaInjetores(OFF,3);                   // Desliga injetor gêmeo (auxiliar)
                    PIT_LDVAL1 = Tinj23 / 0.0002;               // Carrega valor de t_inj do PIT1
                    PIT_TCTRL1 |= PIT_TCTRL_TIE_MASK;           // Habilita interrupcao
                    PIT_TCTRL1 |= PIT_TCTRL_TEN_MASK;           // Habilita PIT1
                    estado = 0;                                // Reseta variavel estado
                    Tinj23 = 0;                               // Reseta variavel estado
                    break;
            }
        } //Final switch (estado)
    } //Final else if (!estado)
    else {                                                 //Caso injetores limitados TinjFlag = 1
        Tinj23 = *Tinj;                                    //Armazena valor de injecao restante
        estado = Injetor;                                 //Armazena estado e injetor gêmeo
    } //Final else if (!Injetor)

} //Final da Funcao
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************

//****************************************************************************
/*          Funcao  : void Vetor_PIT2(void) */          */
/* */                                                 */
/* TaskNumber: 87 */                                     */
/* */                                                 */
/* Entradas : uint32_t*TempoCarga(Ponteiro com tempo de carga da bobina) */
/*          uint8_t TCTRL (Variavel de controle e comando da funcao)      */
/* */                                                 */
/* Saida   : Nenhuma */                                 */
/* */                                                 */

```

```

/*
 * Descricao : Esta funcao ocorre na interrupcao do PIT2. O PIT 2 eh de
 * uso exclusivo para controle do acionamento da bobina 1.
 *
 * O PIT eh carregado com novo valor na funcao de sincronismo
 * do sensor de rotacao. Basicamente apos o dente de referencia
 * este TIMER eh carregado pela funcao do sensor de rotacao com o tempo de disparo calculado e tabelado. Quando
 * ocorre a primeira interrupcao (volta = 0) a funcao aciona
 * a bobina e carrega nova interrupcao com o tempo de carga
 * da bobina. Quando ocorrer a proxima interrupcao a funcao
 * desliga a bobina.
 *
 * DADOS DA INTERRUPCAO
 * INTERRUPCAO = SOFTWARE (TIMER DECREMENTA ATÉ ZERO)
 * PERÍODO = EM FUNCAO DO TEMPO DE DISPARO e CARGA
 * HARDWARE = *-*-
 *****/
*****/

void Vetor_PIT2(uint32_t *TempoCarga, uint8_t TCTRL) {

    static uint8_t volta = 1;                                //Indica estado da funcao
    static uint32_t carga = 0;                             //Memoria do tempo de carga da bobina

    switch (TCTRL) {
        case 0:                                         //Caso funcao OFF
            volta = 1;                                 //Reinicia variaveis
            carga = 0;
            break;

        case 1:                                         //Caso funcao ON
            PIT_TFLG2 |= PIT_TFLG_TIF_MASK; //limpa flag escrevendo 1 TIF
            PIT_TCTRL2 &= (!PIT_TCTRL_TEN_MASK);      //Desabilita PIT2
            PIT_TCTRL2 &= (!PIT_TCTRL_TIE_MASK);      //Desabilita interrupcao

            volta = !volta;                            //Altera valor de volta

            if (!volta) {
                ControlaBobinas(ON,1);                  //Liga bobina 1
                PIT_LDVAL2 = carga; // Carrega o novo valor timer (t carga bobina)
                PIT_TCTRL2 |= PIT_TCTRL_TIE_MASK;        //Habilita interrupcao
                PIT_TCTRL2 |= PIT_TCTRL_TEN_MASK;        //Habilita PIT2
            }
            else ControlaBobinas(OFF,1);               //Desliga bobina 1
            break;

        case 2:
            carga = *TempoCarga;                   //Armazena novo valor de carga bobina
            break;
    }

} //Final da Funcao
*****/
*****/
*****/
*****/

```



```

}//Final da Funcao
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************

//****************************************************************************
/*
     Funcao  : void Vetor_PDB0(void)
*/
/*
/* TaskNumber: 89
*/
/*
/* Entradas : Nenhuma
*/
/*
/* Saída   : Atualiza o escalonador do sistema RTOS
*/
/*
/* Descrição : Esta função ocorre na interrupção do PDB0. A interrupção
/*          ocorre de forma periódica com trigger por software.
/*          uma nova interrupção ocorrerá a cada 1ms. A prioridade
/*          definida é 2. O objetivo é atualizar o escalonador do
/*          RTOS.
*/
/*
/* DADOS DA INTERRUPÇÃO
/*
/* INTERRUPÇÃO = SOFTWARE (TIMER PDB0)
/*
/* PERÍODO    = 1ms
/*
/* HARDWARE   = *-*-
*/
//****************************************************************************
//****************************************************************************

void Vetor_PDB0(void) {

    static uint16_t scheduler = 0;      //variável direcionadora

    PDB0_SC &= 0xFFBF;                //Limpa flag, zera bit PDBIF = bit 6

    scheduler++;                      //Incrementa scheduler
    if (scheduler == 10) scheduler = 0; //Limita scheduler
    escalonador (ARMAZENA, &scheduler); //Armazena scheduler dentro do RTOS
}

//Final da função
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************

//****************************************************************************
/*
     FIM DO ARQUIVO
*/
*****

```

ENGINEMANAGE_TASKS.H

```

//****************************************************************************
/*
     Escola Politécnica da Universidade de São Paulo
*/
/*
     POLI - USP - EPUSP
*/
/*
     Mestrado em Engenharia Elétrica
*/
/*
     Microeletrônica
*/
/*
*****Firmware da Unidade de Gerenciamento Eletrônico do motor EA113 2.0L*****

```

```
***** Autor *****/
/*
/*
* Bruno Martin de Alcântara Dias
*/
***** */

/*
* Modulo: cartronic_v3_2.c
*/
/*
* Descrição: Biblioteca de Gerenciamento do Motor (EngineManage_tasks.h)
*/
/*
* Projeto: UCM - CARTRONIC v3.0
*/
/*
* Compilador: CodeWarrior for MCU - Version: 10.2 - Build Id:120126
*/
/*
* Hardware: Cartronic v3.2 32bits - Freescale Kinetis K40 256k
*/
/*
* MCU = K40X256VLQ100
*/
/*
*/
/*
* Nucleo: ARM CORTEX M4 32bits
*/
/*
* Data: <10/05/2014>
*/
/*
* Historico: Iniciais Motivo da Mudança
* do Projetista
*/
/*
* 17/05/2014 BMAD Inicial version.
* 24/05/2014 BMAD Sistema Injecao
* 25/05/2014 BMAD Conclusao SistemalGN
* 28/05/2014 BMAD Sistema Admissao
* 06/06/2014 BMAD Controle Partida 01
* 19/08/2014 BMAD Aplicacao Filtro ROT
* 28/08/2014 BMAD Sincronismo INJ,IGN
* 12/09/2014 BMAD Alteracao Injecao
* 12/10/2014 BMAD Partiu motor 1a vez!
* 29/10/2014 BMAD Alteracao Admissao
* 30/10/2014 BMAD Calculo Tinj
* 18/12/2014 BMAD Controle PID rotacao
*/
***** */

/*
* DESCRICAO GERAL DA BIBLIOTECA
*/
***** */


```

// Esta biblioteca agrupa o conjunto de funções relacionadas ao gerenciamento
// eletrônico do motor.

```
***** */
/*
* Funcao : uint16_t AbilitaInjetores(uint8_t comando, uint8_t CI)
*/
/*
* TaskNumber: 101
*/
/*
* Entradas : uint8_t comando (Indica se Abilita ou desabilita)
* uint8_t CI(Indica qual circuito integrado sera controlado)
*/
/*
* Saída : Indicação de erro -> 0 = OK ; 101x = ERRO
*/
/*
* Descrição : Esta função Abilita ou desabilita o acionamento dos
*/
***** */


```



```

void IdentificaFase (char *volta) {
    unsigned int entrada; //Variavel que assume valor de PTD5

    entrada = GPIOD_PDIR & PTx5; //Caso sensor fase = 1 - entrada = 0x20

    if(entrada == PTx5){ //Avalia qual a fase do motor
        *volta = 0; //Fase = 1 na falha volta = 0
    }else{
        *volta = 1; //Fase = 0 na falha volta = 1
    }

    RTOS_VEM [FASE_VEM] = *volta; //Atualiza Vetor Estados do RTOS
}//Final da Funcao
/*****
/*****
/*****
/*****



/*****
/*      Funcao : Rotacao (uint8_t TCTRL, uint16_t *t_dente, uint8_t *RPM) */
/*
/* TaskNumber: 103
/*
/* Entradas : uint8_t TCTRL (Variavel de comando e controle da funcao)
/*      uint16_t *t_dente (Ponteiro informa tempo de dente atual)
/*
/* Saída : Ponteiro RPM com valor da rotacao atualizado apos calculo
/*
/* Descrição : Esta função foi desenvolvida para calcular a atual rotacao
/*      do motor tendo como entrada o valor do timer FTM0 que
/*      incrementa a cada 1.28us. Este valor reflete o tempo que
/*      um dente levou para passar pelo sensor de rotacao. Com
/*      isto eh possivel calcular a rotacao do motor (RPM) e retornar
/*      o valor de RPM/100, ou seja caso o motor gire a 1000RPM
/*      a função irá retornar 10. Caso seja 1500RPM retornara 15.
/*****



void Rotacao (uint8_t TCTRL, uint16_t *t_dente, uint8_t *RPM) {

    static uint8_t valor = 0; //Variavel de memoria da funcao

    switch (TCTRL) { //Variavel de comando e controle da funcao

        case 0: //Caso OFF
            valor = 0; //Reseta memoria da funcao
            break; //Final OFF

        case 1: //Caso RETORNA
            *RPM = valor; //Retorna ponteiro RPM previamente calculado
            break; //Final RETORNA

        case 2: //Caso ATUALIZA
            *RPM = 1000000 / ((*t_dente) * 128); //Calcula rotacao do motor
            valor = *RPM; //Armazena Valor de RPM
            RTOS_VEM [RPM_VEM] = valor; //Atualiza vetor de estados do RTOS
            break; //Final ATUALIZA
    }
}//Final da Funcao

```

```

/*****************/
/*****************/
/*****************/
/*****************/

/*****************/
/*      Funcao : void SincronismoSistemaADM(char TCTRL) */
/*
/* TaskNumber: 104
/*
/* Entradas : void
/*
/* Saída : Controla da valvula borboleta (PWM)
/*
/* Descricao : Esta funcao foi desenvolvida para atuar taxa de amostragem
/*           de 10ms. O objetivo eh efetuar o controle e aplicar
/*           tecnicas de controle classico (PI) da valvula borboleta.
/*           Inicialmente o controle eh em malha fechada utilizando um
/*           controle proporcional e integral em tempo discreto.
/*****************/
void SincronismoSistemaADM(void) {

    static short int Erro;
    static short int Erro1;
    unsigned short int Pwm;
    short int Saída;

    uint8_t Tps;
    uint8_t Referencia;           //Referencia de posicao para Borboleta

    //Variaveis utilizadas na funcao que nao necessitam ser declaradas
    //ErroMaximo = 255
    //SaídaMaxima = 220 (Final de curso da Borboleta)
    //Resolucao ADC = 8bits (255)
    //Kp = 12
    //Ki = 0.07

    Tps = SensorTPS2();          //Captura novo valor de Tps

    RTOS_VEM [TPS2_VEM] = Tps;   //Atualiza Vetor de estados

    Referencia = RTOS_VEM [REF_VB_VEM]; //Captura referencia atual para VB

    Erro = Referencia - Tps;     //Calcula Erro do PI

    //Limita Erro em funcao dos valores de ErroMaximo adquiridos na funcao CalibraBorboleta
    if (Erro >= 255) Erro = 255;
    if (Erro <= (-255)) Erro = (-255);

    //Calcula nova saida
    Saída = (12 * Erro) + (0.07 * (Erro + Erro1)); //Calculo do PI com (n-1)
                                                //((Kp * Erro) + (Ki * (Erro + Erro1)))

    //Atualiza Variaveis
    Erro1 = Erro;

    //Limita saida em funcao dos valores de ErroMaximo
}

```

```

if (Saida > 220) Saida = 220;
if (Saida < (-220)) Saida= (-220);

//Atualiza valor da nova saida para atualizar registrador do PWMs
if (Saida < 0) { //Executa de saida eh menor que zero (Erro negativo)
    Pwm = ((12500 * (Saida * (-1))) / 205);
    //Converte PWM dentro do limite do registrador (12500) 2Khz
    // Controla Valvula Borboleta atualizando PWMs
    FTM1_COV = 0; //PWM 0%
    FTM1_C1V = Pwm;//PWM de acordo com o valor do calculo PID
}
else { //Executa de saida eh maior que zero (Erro positivo)
    Pwm = (12500 * Saida) / 205;
    //Converte PWM dentro do limite do registrador (12500) 2Khz
    // Controla Valvula Borboleta atualizando PWMs
    FTM1_COV = Pwm;//PWM de acordo com o valor do calculo PID
    FTM1_C1V = 0; //PWM 0%
}
//Final da funcao
/******
*   Funcao  : void CalculoTempoInjecao(uint8_t TCTRL, uint8_t *Tinj)
*/
/* TaskNumber: 105 */
/* Entradas : uint8_t TCRL(Indica se calcula ou retorna tempo d injecao) */
/* Saída   : Tempo de Injecao caculado */
/* Descrição : Esta função foi desenvolvida para calcular o tempo de
   injecao de combustivel. A mesma dependendo da variavel
   TCTRL (CALCULA ou RETORNA) calcula o tempo de injecao
   ou retorna o valor do tempo de injecao ja calculado.
   O calculo ocorre em função da pressao absoluta no coletor
   (MAP) e da temperatura do ar de admissao (TAR). O tempo de
   injecao eh calculado conforme equacoes da estrategia SPEED
   DENSITY. Esta equacao leva em conta o volume do cilindro
   e a densidade do ar de admissao alem de corrigir o Tinj em
   função da eficiencia volumetrica (VE) que eh calculada em
   função do MAP,TAR e RPM. Com o MCU a 100MHz o calculo foi
   executado em 20us. Todas as equacoes sao devidamente
   descritas e explicadas na disertação deste mestrado.
*/
void CalculoTempoInjecao(uint8_t TCTRL, uint8_t *Tinj) {

    static uint8_t Tinjacao = 40;      //Memoria do Tempo de injecao
    uint8_t MAP = 0;                  //Variavel de aquisicao do valor de MAP
    static uint8_t RPM = 0;           //Variavel de aquisicao do valor de RPM
    uint8_t TAR = 0;                 //Variavel de aquisicao do valor de TAR

    if (!TCTRL) { //Caso modo = CALCULA
        MAP = RTOS_VEM [MAP_VEM];          //Recebe do RTOS pressao em [kPa/10]
        RPM = RTOS_VEM [RPM_VEM];          //Recebe do RTOS Rotacao em [RPM/100]
    }
}

```

```

TAR = RTOS_VEM [TAR_VEM];           //Recebe do RTOS Temp AR em [*C/10]

////////////////////////////// //Calcula o Tempo de injecao em [us]

Tus = (2.57 * (MAP/10.13) * (288/((TAR*10)+273))) / (0.0001627296);
//Tus = (Rho*(MAP/Po)*(To/TAR)*Ncil*Vpistao)/(Vinjeto*Ninjetores*Lambda)

//Rho = 1.299[Kg/m3] - Densidade do ar atmosferico a nivel do mar
//Po = (101.3/10) [kPa] - Pressao do ar atmosferico a nivel do mar
//To = 288 [K] - Temperatura ambiente (15*C)
//Ncil = 4 - Numero de cilindros do motor
//Vpistao = 0.496 [m3]
//Vinjeto = 2.948 [ug/us] - Fluxo do injetor (microgramas por us]
//Ninjetores = 4 - Numero de injetores
//Lambda = 13.8 - Lambda para estequiometria (gasohol brasileiro)
//////////////////////////////

////////////////////////////// //Calcula Eficiencia Volumetrica do motor

VE = (0.001984 * ((MAP*10000)/(8.134*(TAR*10))) * ((RPM*100)/120)) / (5.952);
//VE = (Vreal/Vteorico)
//Vreal = Vt * (MAP/(R*TAR)) * (RPM/120)
//Vteorico = Vt * (RPMmax/2) * VEmax

//Vt = 1.984 [cm3] - Volume total do motor (converter para [m3])
//VEmax = 1 - Eficiencia volumetrica maxima
//RPMmax = 6000 [RPM] - Rotacao maxima do motor
//R = 8.134 [J/mol*C] - Constante do Gas

//////////////////////////////

Tinjecao = Tus * VE;           //Tempo de injecao corrigido [us/100]

if(RPM <= 6) Tinjecao = 120;    //Caso Rotacao menor que 700RPM

//Determina Tinj = 12ms para partida do motor
if (Tinjecao <= 29) Tinjecao = 29; //Determina limite minimo de 2.9ms
if(Tinjecao >= 180) Tinjecao = 180; //Determina limite maximo de 18ms

RTOS_VEM [TINJ_VEM] = Tinjecao; //Atualiza Vetor de estados

}else                         //Caso RETORNA
*Tinj = Tinjecao;             //Retorna Tempo de injecao calculado
//em microsegundo dividido por cem. (us/100)

}//Final da Funcao
***** */
***** */
***** */
***** */

***** */
/*      Funcao : CalculoAvancoIGN(uint8_t TCTRL, uint8_t *avancoIGN) */
/*
/* TaskNumber: 106
/*
/* Entradas : uint8_t TCTRL(Indica se calcula ou retorna Avanco de IGN)
*/

```

```

/*
/* Saída : uint8_t *avancoIGN (Ponteiro do Avanco de ignicao)
*/
/*
/* Descrição : Esta função foi desenvolvida para calcular o avanco de
/* ignicao das bobinas. A função dependendo da variável
/* TCTRL(CALCULA ou RETORNA) calcula o avanco de ignicao
/* ou retorna o valor de avanco previamente calculado.
/* Na realidade o cálculo foi efetuado externamente e mapeado
/* Este mapa está contido na header "ENGINE_MAPS.h". O mapa
/* é indexado pelo valor de RPM e de MAP. Para chegar aos
/* valores finais de avanco efetuamos uma série de ensaios
/* em dinamômetro tendo como referência o inicio da presença
/* de detonação no motor. Após efetuar o mapa inicial, o mesmo
/* foi interpolado no MATLAB para conseguirmos uma resolução
/* de 100RPM e 10kPa em todo o mapa. Esta função efetua leitura
/* do sensor MAP e adquire o valor de RPM do RTOS para indexar
/* a matriz Mapalgnicao.
//*****************************************************************************
void CalculoAvancoIGN(uint8_t TCTRL, uint8_t *avancoIGN) {

    static uint8_t Avanco = 0;           //Memória do Avanco de Ignicao
    uint8_t MAP = 0;                   //Variável de aquisição do valor de MAP
    uint8_t RPM = 0;                  //Variável de aquisição do valor de RPM

    if (!TCTRL) {                     //Caso modo = CALCULA
        MAP = SensorMAP();           //Calcula pressão em (kPa/10)
        Rotacao(RETORNA, 0, &RPM);   //Retorna valor da rotação(RPM/100)
        Avanco = Mapalgnicao [RPM][MAP]; //Calcula Avanco de Ignicao
        RTOS_VEM [AVANCO_VEM] = Avanco; //Atualiza Vetor de Estados RTOS
        RTOS_VEM [MAP_VEM] = MAP;     //Atualiza Vetor de Estados RTOS
    }else {                           //Caso RETORNA
        *avancoIGN = Avanco;         //Retorna Avanco ignicao
    }
} //Final da Função
//*****************************************************************************
//*****************************************************************************
//*****************************************************************************
//*****************************************************************************
//*****************************************************************************
/*
/* Função : void init_engine(char TCRTL)
/*
/* TaskNumber: 107
/*
/* Entradas : char TCTRL (Controle da tarefa)
/*
/* Saída : Indicação de erro -> 0 = OK ; 112x = ERRO
/*          Retorna para o RTOS somente quando o motor estiver
/*          operando com rotação superior a 800RPM
/*
/* Descrição : Esta função foi desenvolvida para atuar no momento da
/* partida do motor. Devido à condição desfavorável da partida do motor todas as interrupções estão desabilitadas e o
/* RTOS não opera. Antes de entregarmos o controle do motor ao RTOS esta função atua identificando a falha de forma
/* manual e sincronizando o motor. Quando a função identificar
/* que o motor conseguir pegar ela se encerra e entrega o
/* controle do sistema ao RTOS.
*/

```

```

/*
OBS: Utiliza timer FTMO com incremento a cada 1.28us
****/
void init_engine(uint8_t TCTRL) { //Primeira borda descida

    static uint8_t estado = 0; //Variavel que indica estado da task
    static uint8_t borda = 1; //Borda 0 = Descida | Borda 1 = Subida
    static uint16_t t_borda_descida = 0; //Tempo de borda de descida
    static uint8_t borda_descida = 0; //Indica qual borda o sistema se encontra
    static uint8_t borda_subida = 0; //Indica qual borda o sistema se encontra
    static uint8_t objetivo = 0;
    uint16_t t_referencia = 0; //Tempo de referencia para sincronismo
    char Dente; //Indica dente para atualizacao do sistema

    if (!TCTRL) { //Executa esta condicao para resetar funcao
        DesabilitaInterrupcao(ISR_FASE); //Desabilita Interrupcao FASE
        estado = 0; //Zera estada da task
        borda = 1; //Borda 0 = Descida | Borda 1 = Subida
        t_borda_descida = 0; //Tempo de borda de descida
        borda_descida = 0; //Indica qual borda o sistema se encontra
        borda_subida = 0; //Indica qual borda o sistema se encontra
        objetivo = 0; //Zera objetivo da funcao indicando motor Desligado
    } else {

        borda = !borda; //Altera borda

        /////////////////////////////////Altera registrador de Borda do sinal
        if (!borda) {
            PORTD_PCR5 &= 0xFFFF; //Desabilita interrupcao PTD5 - Fase
            PORTD_PCR5 |= (PORT_PCR IRQC_MASK & 0x90000); //Altera borda para subida
        } else {
            PORTD_PCR5 &= 0xFFFF; //Desabilita interrupcao PTD5 - Fase
            PORTD_PCR5 |= (PORT_PCR IRQC_MASK & 0xA0000); //Altera borda para descida
        }
        /////////////////////////////////

        ////////////////////////////////Avalie Estado da task/////////////////////////////
        //ESTADO = 0 - MOTOR SEM SINCRONISMO (BUSCANDO SINCRONISMO)
        //ESTADO = 1 - MOTOR COM SINCRONISMO (ATUALIZANDO SINCRONISMO)
        switch (estado) {

            case 0: // Estado = 0 - Identificar fase
            if (!borda) { //Executa caso borda de descida
                FTM2_CNT = 1; //Zera contador para medir tempo
            } else { //Executa caso borda de subida
                t_referencia = FTM2_CNT; //Captura tempo entre bordas
                if (t_referencia <= (t_borda_descida / 2)) { //Busca sincronismo do sistema
                    estado = 1; //Caso sincronizado altera estado para 1
                    borda_subida = 4; //Atualiza borda atual e sincroniza sistema
                    borda_descida = 3; //Atualiza borda atual e sincroniza sistema
                }
                t_borda_descida = t_referencia; //Guarda tempo de borda de descida
            }
            break; //Final da busca pelo sincronismo.

            case 1: //Estado = 1 - Sincronismo injecao e ignicao
            if (!borda) { //Executa caso borda de descida

```

```

        borda_descida++; //Incrementa borda de descida
        if (borda_descida == 2) {
            FTM2_CNT = 1; //Zera contador para medir tempo
            Dente = 50; //Resincroniza sistema

            Vetor_SensorRotacao(ATUALIZA, &Dente); //Atualiza Dente
        }
        if (borda_descida == 3) FTM2_CNT = 1; //Zera contador tempo
        if (borda_descida == 5) borda_descida = 1; //Reseta borda de descida
        else {
            //Executa caso borda de subida
            borda_subida++; //Incrementa borda de subida
            if(borda_subida == 3) t_borda_descida = FTM2_CNT;
            //Captura tempo borda descida
            if (borda_subida == 4) {
                t_referencia = FTM2_CNT;
                if (!(t_referencia <= (t_borda_descida / 2))) {
                    borda_subida++; //Atualiza borda atual e sincroniza sistema
                    borda_descida++; //Atualiza borda atual e sincroniza sistema
                }
            }
            if (borda_subida == 5) {
                borda_subida = 1; //Reseta borda de descida
                Dente = 57;
                Vetor_SensorRotacao(ATUALIZA, &Dente); //Atualiza Dente
                if (!objetivo) { //Libera interrupcao do sensor de rotacao (init engine)
                    HabilitaInterrupcao(ISR_ROTACAO); //Interrupcao sensor
                    objetivo = 1; //Flag setada
                    HabilitaInjetores(HABILITA,1);
                    //Habilita os injetores 1,2,3 e 4
                }
            }
        }
    } //Final do ELSE BORDA
    break; //Final da atualizacao do sincronismo.
} //Final switch (estado)
} //Final ELSE TCTRL
} //Final da Funcao
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************
/*
     Funcao : void AtualizaRTOS(void)
*/
/*
TaskNumber: 108
*/
/*
Entradas : Nenhuma
*/
/*
Saida : RTOS_VEM[RTOS_BUS] Atualizado
*/
/*
Descricao : Esta funcao foi desenvolvida para atualizar os parametros
do vetor de estados do motor (VEM). O objetivo e ler todos
os sensores e garantir que todos os valores do RTOS_VEM
se atualizem para que as demais funcoes do RTOS operem de
maneira correta.
*/
void AtualizaRTOS (void) {

```

```

//RPM_VEM
//DENTE_VEM
//FASE_VEM
//TINJ_VEM
//TCARGA_VEM
//AVANCO_VEM
//MAP_VEM
//PEDAL1_VEM
RTOS_VEM [PEDAL1_VEM] = SensorPEDAL1(); //Atualiza Vetor de estados RTOS
//PEDAL2_VEM
RTOS_VEM [PEDAL2_VEM] = SensorPEDAL2(); //Atualiza Vetor de estados RTOS
//TPS1_VEM
RTOS_VEM [TPS1_VEM] = SensorTPS1(); //Atualiza Vetor de estados RTOS
//TPS2_VEM
//TAR_VEM
RTOS_VEM [TAR_VEM] = 4; //Atualiza Vetor de estados RTOS
//TAGUA_VEM
RTOS_VEM [TAGUA_VEM] = SensorTAGUA(); //Atualiza Vetor de estados RTOS
//LAMBDA_VEM - Ainda Nao Implementado
//KNOCK1_VEM - Ainda Nao Implementado
//KNOCK2_VEM - Ainda Nao Implementado
//ALTERNADOR_VEM - Ainda Nao Implementado *****
RTOS_VEM [ALTERNADOR_VEM] = 135;
//INT_ALTERNADOR_VEM- Ainda Nao Implementado *****
//LINHA15_VEM - Atualizado na task 83
//FREIOL30_VEM - Ainda Nao Implementado *****
RTOS_VEM [FREIOL30_VEM] = 1;
//FREIOL15_VEM - Ainda Nao Implementado *****
RTOS_VEM [FREIOL15_VEM] = 0;
//EMBREAGEM_VEM - Ainda Nao Implementado *****
RTOS_VEM [EMBREAGEM_VEM] = 0;
//WARN_LAMBDA_VEM - Ainda Nao Implementado
//POSITIVO_RPM_VEM - Atualizado na task 46
//POSITIVO_SENSOR_VEM - Atualizado na task 45
//MAIN_RELAY_VEM - Atualizado na task 42
//FUELPUMP_RELAY_VEM - Ainda Nao Implementado
//COLDSTART_RELAY_VEM - Ainda Nao Implementado
//RESERVA_RELAY_VEM - Ainda Nao Implementado
//INJETOR1_VEM - Atualizado na task 43
//INJETOR2_VEM - Atualizado na task 43
//INJETOR3_VEM - Atualizado na task 43
//INJETOR4_VEM - Atualizado na task 43
//INJETOR5_VEM - Atualizado na task 43
//INJETOR6_VEM - Atualizado na task 43
//INJETOR7_VEM - Atualizado na task 43
//INJETOR8_VEM - Atualizado na task 43
//BOBINA1_VEM - Atualizado na task 44
//BOBINA2_VEM - Atualizado na task 44
//BOBINA3_VEM - Atualizado na task 44
//BOBINA4_VEM - Atualizado na task 44
//BOBINA5_VEM - Atualizado na task 44
//BOBINA6_VEM - Atualizado na task 44
//BOBINA7_VEM - Atualizado na task 44
//BOBINA8_VEM - Atualizado na task 44
//COLDINJECTOR_VEM - Ainda Nao Implementado
//PURGA_VEM - Ainda Nao Implementado
//ATUADOR_VEM - Ainda Nao Implementado

```

```

//SENSOR_DIG1_VEM           - Ainda Nao Implementado
//SENSOR_DIG2_VEM           - Ainda Nao Implementado
//USB1_VEM                  - Ainda Nao Implementado
//USB1_FLAG_VEM             - Ainda Nao Implementado
//USB2_VEM                  - Ainda Nao Implementado
//USB2_FLAG_VEM             - Ainda Nao Implementado
//BLUE_VEM                  - Ainda Nao Implementado
//BLUE_FLAG_VEM             - Ainda Nao Implementado
//CAN1_VEM                  - Ainda Nao Implementado
//CAN1_FLAG_VEM             - Ainda Nao Implementado
//CAN2_VEM                  - Ainda Nao Implementado
//REF_VB_VEM                - Atualizado na task 110

//REGIME_VEM                - Atualizado na task 111
} //Final da Funcao
//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************

//****************************************************************************
/*          Funcao  : void TempoMaximoInjecao (uint8_t *TMAX) */
/*
/* TaskNumber: 109
/*
/* Entradas : Nenhuma
/*
/* Saida   : Ponteiro de saida para atualizar o Tinj maximo
/*
/* Descricao : Esta funcao foi desenvolvida para limitar o tempo de inje
/*      cao em funcao da rotacao do motor. Esta medida eh necessa
/*      ria para que uma injecao nao "atropelie" a outra do mesmo
/*      banco, visto que um timer atua em dois injetores.
/*
void TempoMaximoInjecao (uint8_t *TMAX) {

    uint8_t RPM = 0;

    RPM = RTOS_VEM [RPM_VEM]; //Recebe do RTOS o valor atual de rotacao do motor

    if(RPM <= 28) *TMAX = 180;                                //Limita para 18ms
    else if ((RPM >= 33) && (RPM < 35)) *TMAX = 170;    //Limita para 17ms
    else if ((RPM >= 35) && (RPM < 38)) *TMAX = 160;    //Limita para 16ms
    else if ((RPM >= 38) && (RPM < 40)) *TMAX = 150;    //Limita para 15ms
    else if ((RPM >= 40) && (RPM < 43)) *TMAX = 140;    //Limita para 14ms
    else if ((RPM >= 43) && (RPM < 47)) *TMAX = 130;    //Limita para 13ms
    else if ((RPM >= 47) && (RPM < 51)) *TMAX = 120;    //Limita para 12ms
    else if ((RPM >= 51) && (RPM < 57)) *TMAX = 110;    //Limita para 11ms
    else *TMAX = 90;                                         //Limita para 09ms
}

//****************************************************************************
//****************************************************************************
//****************************************************************************
//****************************************************************************

//****************************************************************************
/*          Funcao  : void PID_RPM (void) */
*/

```

```

/*
/* TaskNumber: 110 */
/*
/* Entradas : Nenhuma */
/*
/* Saida   : Referencia digital para controle da valvula borboleta */
/*
/* Descricao : Esta funcao foi desenvolvida calcular a referencia correta
/*      para que seja utilizada na funcao de controle da valvula
/*      borboleta. A saida desta funcao eh calculado a partir de
/*      tecnicas de controle PID. Devido a dinamica do motor ciclo
/*      Otto o ganho proporcional varia em funcao da rotacao do
/*      motor e em funcao do erro estacionario de rotacao.
/*      Utilizamos tres equacoes para definirmos o ganho KP. Uma
/*      equacao para baixa rotacoes, outra para medias rotacoes e
/*      a ultima para altas rotacoes. O saida da funcao eh carre-
/*      gada no vetor de estados do motor para ser utilizado por
/*      todo sistema RTOS. Inclusive pela task 104 (PID Borboleta)
//*****
void PID_RPM (void) {

    static short int Erro;           //Variavel para obtenção do erro
    static short int Erro1;          //Memoria para Erro anterior
    unsigned short int Referencia;  //Variavel de saida da funcao
    short int Saida;               //Variavel para calculo da funcao

    //Variaveis utilizadas na funcao que nao necessitam ser declaradas
    //Kp = 81.1
    //Ki = 2.3

    Erro = RTOS_VEM [REF_RPM_VEM] - RTOS_VEM [RPM_VEM];
    //Erro = Referencia Rotacao - Rotacao Atual

    //Calcula nova saida
    Saida = (81.1 * Erro) + (2.3 * (Erro + Erro1)); //Calculo do PI com (n-1)
    //((Kp * Erro) + (Ki * (Erro + Erro1)))

    //Limita saida em funcao do curso total da valvula borboleta
    if (Saida > 220) Saida = 220;
    if (Saida < (-220)) Saida= (-220);

    //Atualiza Variaveis
    Erro1 = Erro;

    //Atualiza valor da nova saida para atualizar referencia borboleta
    Referencia = ((220 * Saida) / 220) + 32;

    RTOS_VEM [REF_VB_VEM] = Referencia;           //Atualiza a referencia para borboleta
}
//Final da Funcao
//*****
//*****
//*****
//*****
```

```

//*****
/*      Funcao  : void EstadoAtualMotor (void)
*/
*/
```

```

/*
 * TaskNumber: 111
 */
/*
 * Entradas : Nenhuma
 */
/*
 * Saída : Estado atual de operação do motor
 */
/*
 * Descrição : Esta função foi desenvolvida para determinar o estado
 *              atual de operação do motor. Foram definidos 6 estados
 *              possíveis para o motor. Para determinar o estado do motor
 *              esta função verifica a rotação atual, a posição do pedal e
 *              a rotação anterior do motor a 200ms passados. Após determinar
 *              o estado atual a função atualiza o vetor de estados do
 *              motor no RTOS para que outras funções utilizem deste para
 *              metro.
 */
/*
 *          //REGIME_VEM = 0 - MOTOR DESLIGADO
 *          //REGIME_VEM = 1 - MOTOR MARCHA LENTA
 *          //REGIME_VEM = 2 - MOTOR ACCELERACAO
 *          //REGIME_VEM = 3 - MOTOR DESACCELERACAO
 *          //REGIME_VEM = 4 - MOTOR CRUZEIRO
 *          //REGIME_VEM = 5 - MOTOR PLENA CARGA
 */
*****void EstadoAtualMotor (void) {

    static uint8_t MemoriaRPM = 0; //Variável que armazena rotação anterior
    uint8_t RegimeAtual = 4; //Variável de saída

    // PARA REGIME = (MOTOR DESLIGADO) O ESTADO EH ATUALIZADO JUNTO COM L15

    if((RTOS_VEM [RPM_VEM] <= 14) && (RTOS_VEM [PEDAL2_VEM] <= 14))
        RegimeAtual = 1; //MOTOR EM MARCHA LENTA

    if((RTOS_VEM [PEDAL2_VEM] > 49) && (RTOS_VEM [RPM_VEM] >= (MemoriaRPM + 1)))
        RegimeAtual = 2; //MOTOR EM ACCELERACAO

    if((RTOS_VEM [PEDAL2_VEM] <= 15) && (RTOS_VEM [RPM_VEM] >= 15))
        RegimeAtual = 3; //MOTOR EM DESACCELERACAO

    if ((RTOS_VEM [PEDAL2_VEM] >= 16) && (RTOS_VEM [PEDAL2_VEM] <= 79)
        && (RTOS_VEM [RPM_VEM] >= 15) && (RTOS_VEM [RPM_VEM] <= 49))
        RegimeAtual = 4; //MOTOR EM CRUZEIRO

    if ((RTOS_VEM [PEDAL2_VEM] >= 80) && (RTOS_VEM [RPM_VEM] >= 50))
        RegimeAtual = 5; //MOTOR EM PLENA CARGA

    RTOS_VEM [REGIME_VEM] = RegimeAtual; //Atualiza RTOS_VEM
    MemoriaRPM = RTOS_VEM [RPM_VEM];
    //Armazena valor de rotação
} //Final da Função
*****
*****
*****
/

```

```

void ReferenciaRPM(void) {
    //Calcula qual a rotacao de referencia desejada pelo motorista
    RTOS_VEM [REF_RPM_VEM] = (0.52 * RTOS_VEM [PEDAL1_VEM]) + 8;
    //Retorna ref_rpm pelo pedal
} //Final da Funcao
/* ***** FIM DO ARQUIVO *****/

```

MCUinit.c

```

/*
** ##### This code is generated by the Device Initialization Tool.
** It is overwritten during code generation.
** USER MODIFICATION ARE PRESERVED ONLY INSIDE INTERRUPT SERVICE ROUTINES
** OR EXPLICITLY MARKED SECTIONS
**
** Project : DeviceInitialization
** Processor : MK40DX256ZVLQ10
** Version : Component 01.000, Driver 01.03, CPU db: 3.00.001
** Datasheet : K40P144M100SF2RM, Rev. 5, 8 May 2011
** Compiler : CodeWarrior ARM C Compiler
** Date/Time : 2014-12-21, 09:40, # CodeGen: 4
** Abstract :
**
** Contents :
** Function "MCU_init" initializes selected peripherals
**
** Copyright : 1997 - 2011 Freescale Semiconductor, Inc. All Rights Reserved.
**
** http : www.freescale.com
** mail : support@freescale.com
** #####
*/
/* MODULE MCUinit */

#define PE_MCUINIT

#include <MK40DZ10.h>          /* I/O map for MK40DX256ZVLQ10 */
#include "MCUinit.h"

typedef void (*const tIsrFunc)(void);
typedef struct {
    uint32_t * __ptr;
    tIsrFunc __fun[119];
} tVectorTable;

/* User declarations and definitions */
/* Code, declarations and definitions here will be preserved during code generation */
/* End of user declarations and definitions */

/*
** =====
** Method : Cpu_SetBASEPRI (component MK40N512LQ100)
**

```

```

**  Description :
**      This method sets the BASEPRI core register.
**      This method is internal. It is used by Processor Expert only.
** =====
*/
/*lint -save -e586 -e950 Disable MISRA rule (2.1,1.1) checking.*/
#ifndef _lint
#define Cpu_SetBASEPRI(Level) /* empty */
#else
asm void Cpu_SetBASEPRI(register uint32_t Level){
    MSR BASEPRI, R0;
    MOV PC, LR
}
#endif
/*lint -restore Enable MISRA rule (2.1,1.1) checking.*/
/*
** =====
** Method   : __init_hw (component MK40N512LQ100)
**
** Description :
**     Initialization code for CPU core and a clock source.
** =====
*/
extern uint32_t __vector_table[];
void __init_hw(void)
{
    /**## MK40DX256ZVLQ10 "Cpu" init code ... ***/
    /** PE initialization code after reset ***/
    SCB_VTOR = (uint32_t) __vector_table; /* Set the interrupt vector table position */
    /* SIM_SCGC6: RTC=1 */
    SIM_SCGC6 |= (uint32_t)0x20000000UL;
    if ((RTC_CR & RTC_CR_OSCE_MASK) == 0u) { /* Only if the OSCILLATOR is not already enabled */
        /* RTC_CR: SC2P=0,SC4P=0,SC8P=0,SC16P=0 */
        RTC_CR &= (uint32_t)~0x3C00UL;
        /* RTC_CR: OSCE=1 */
        RTC_CR |= (uint32_t)0x0100UL;
        /* RTC_CR: CLKO=0 */
        RTC_CR &= (uint32_t)~0x0200UL;
    }
    /* Disable the WDOG module */
    /* WDOG_UNLOCK: WDOGUNLOCK=0xC520 */
    WDOG_UNLOCK = (uint16_t)0xC520U; /* Key 1 */
    /* WDOG_UNLOCK : WDOGUNLOCK=0xD928 */
    WDOG_UNLOCK = (uint16_t)0xD928U; /* Key 2 */
    /* WDOG_STCTRLH:
DISTESTWDOG=0,BYTESEL=0,TESTSEL=0,TESTWDOG=0,STNDBYEN=1,WAITEN=1,STOPEN=1,DBGEN=0,ALLOWU
PDATE=1,WINEN=0,IRQRSTEN=0,CLKSRC=1,WDOGEN=0 */
    WDOG_STCTRLH = (uint16_t)0x01D2U;
    /* System clock initialization */
    /* SIM_SCGC5: PORTE=1,PORTD=1,PORTC=1,PORTB=1,PORTA=1 */
    SIM_SCGC5 |= (uint32_t)0x3E00UL; /* Enable clock gate for ports to enable pin routing */
    /* SIM_CLKDIV1: OUTDIV1=0,OUTDIV2=1,OUTDIV3=1,OUTDIV4=3 */
    SIM_CLKDIV1 = (uint32_t)0x01130000UL; /* Update system prescalers */
    /* SIM_CLKDIV2: USBDIV=0,USBFRAC=0 */
    SIM_CLKDIV2 &= (uint32_t)~0x0FUL; /* Update USB clock prescalers */
    /* SIM_SOPT2: PLLFLLSEL=0 */
    SIM_SOPT2 &= (uint32_t)~0x00010000UL; /* Select FLL as a clock source for various peripherals */

```

```

/* SIM_SOPT1: OSC32KSEL=0 */
SIM_SOPT1 &= (uint32_t)~0x00080000UL; /* System oscillator drives 32 kHz clock for various peripherals */
/* PORTA_PCR18: ISF=0,MUX=0 */
PORTA_PCR18 &= (uint32_t)~0x01000700UL;
/* PORTA_PCR19: ISF=0,MUX=0 */
PORTA_PCR19 &= (uint32_t)~0x01000700UL;
/* Switch to FBE Mode */
/* OSC_CR: ERCLKEN=1,EREFSTEN=0,SC2P=1,SC4P=0,SC8P=1,SC16P=0 */
OSC_CR = (uint8_t)0x8AU;
/* SIM_SOPT2: MCGCLKSEL=0 */
SIM_SOPT2 &= (uint32_t)~0x01UL;
/* MCG_C2: RANGE=1,HGO=1,EREFS=1,LP=0,IRCS=0 */
MCG_C2 = (uint8_t)0x1CU;
/* MCG_C1: CLKS=2,FRDIV=2,IREFS=0,IRCLKEN=1,IREFSTEN=0 */
MCG_C1 = (uint8_t)0x92U;
/* MCG_C4: DMX32=0,DRST_DRST=0 */
MCG_C4 &= (uint8_t)~(uint8_t)0xE0U;
/* MCG_C5: PLLCLKEN=0,PLLSTEN=0,PRDIV=1 */
MCG_C5 = (uint8_t)0x01U;
/* MCG_C6: LOLIE=0,PLLS=0,CME=0,VDIV=0x1A */
MCG_C6 = (uint8_t)0x1AU;
while((MCG_S & MCG_S_OSCINIT_MASK) == 0x00U) { /* Check that the oscillator is running */
}
while((MCG_S & MCG_S_IREFST_MASK) != 0x00U) { /* Check that the source of the FLL reference clock is the
external reference clock. */
}
while((MCG_S & 0x0CU) != 0x08U) { /* Wait until external reference clock is selected as MCG output */
}
/* Switch to PBE Mode */
/* OSC_CR: ERCLKEN=1,EREFSTEN=0,SC2P=1,SC4P=0,SC8P=1,SC16P=0 */
OSC_CR = (uint8_t)0x8AU;
/* SIM_SOPT2: MCGCLKSEL=0 */
SIM_SOPT2 &= (uint32_t)~0x01UL;
/* MCG_C1: CLKS=2,FRDIV=2,IREFS=0,IRCLKEN=1,IREFSTEN=0 */
MCG_C1 = (uint8_t)0x92U;
/* MCG_C2: RANGE=1,HGO=1,EREFS=1,LP=0,IRCS=0 */
MCG_C2 = (uint8_t)0x1CU;
/* MCG_C5: PLLCLKEN=0,PLLSTEN=0,PRDIV=1 */
MCG_C5 = (uint8_t)0x01U;
/* MCG_C6: LOLIE=0,PLLS=1,CME=0,VDIV=0x1A */
MCG_C6 = (uint8_t)0x5AU;
while((MCG_S & 0x0CU) != 0x08U) { /* Wait until external reference clock is selected as MCG output */
}
while((MCG_S & MCG_S_LOCK_MASK) == 0x00U) { /* Wait until locked */
}
/* Switch to PEE Mode */
/* OSC_CR: ERCLKEN=1,EREFSTEN=0,SC2P=1,SC4P=0,SC8P=1,SC16P=0 */
OSC_CR = (uint8_t)0x8AU;
/* SIM_SOPT2: MCGCLKSEL=0 */
SIM_SOPT2 &= (uint32_t)~0x01UL;
/* MCG_C1: CLKS=0,FRDIV=2,IREFS=0,IRCLKEN=1,IREFSTEN=0 */
MCG_C1 = (uint8_t)0x12U;
/* MCG_C2: RANGE=1,HGO=1,EREFS=1,LP=0,IRCS=0 */
MCG_C2 = (uint8_t)0x1CU;
/* MCG_C5: PLLCLKEN=0,PLLSTEN=0,PRDIV=1 */
MCG_C5 = (uint8_t)0x01U;
/* MCG_C6: LOLIE=0,PLLS=1,CME=0,VDIV=0x1A */

```

```

MCG_C6 = (uint8_t)0x5AU;
while((MCG_S & 0x0CU) != 0x0CU) { /* Wait until output of the PLL is selected */
}
}

/*
** =====
** Method   : MCU_init (component MK40N512LQ100)
**
** Description :
**     Device initialization code for selected peripherals.
** =====
*/
void MCU_init(void)
{
    /* Initialization of the SIM module */
    /* SIM_SOPT6: RSTFLTEN=0,RSTFLTSEL=0 */
    SIM_SOPT6 = (uint32_t)0x00UL; /* Set reset pin filter */
    /* Initialization of the FMC module */
    /* FMC_PFAPR:
M5PFD=1,M4PFD=1,M2PFD=0,M1PFD=0,MOPFD=0,M5AP=0,M4AP=0,M2AP=3,M1AP=3,M0AP=3 */
    FMC_PFAPR = (uint32_t)((FMC_PFAPR & (uint32_t)~0x00070F00UL) | (uint32_t)0x0030003FUL);
    /* FMC_PFB0CR:
BORWSC=3,CLK_WAY=0,CINV_WAY=0,S_B_INV=0,B0MW=1,CRC=0,B0DCE=1,B0ICE=1,B0DPE=1,B0IPE=1,BOSEB
E=1 */
    FMC_PFB0CR = (uint32_t)0x3002001FUL;
    /* FMC_PFB1CR: B1RWSC=3,B1MW=1,B1DCE=1,B1ICE=1,B1DPE=1,B1IPE=1,B1SEBE=1 */
    FMC_PFB1CR = (uint32_t)0x3002001FUL;
    /* Initialization of the FTFL_FlashConfig module */
    /* SIM_SCGC7: MPU=1 */
    SIM_SCGC7 |= (uint32_t)0x04UL;
    /* Initialization of the MPU module */
    /* MPU_CESR: SPERR=0,VLD=0 */
    MPU_CESR &= (uint32_t)~0xF8000001UL;
    /* MPU_RGDAAC0:
M5RE=0,M5WE=0,M4RE=0,M4WE=0,M2SM=3,M2UM=7,M1SM=3,M1UM=7,M0SM=3,M0UM=7 */
    MPU_RGDAAC0 = (uint32_t)((MPU_RGDAAC0 & (uint32_t)~0x0F000000UL) | (uint32_t)0x0001F7DFUL);
    /* MPU_RGD1_WORD3: VLD=0 */
    MPU_RGD1_WORD3 &= (uint32_t)~0x01UL;
    /* MPU_RGD2_WORD3: VLD=0 */
    MPU_RGD2_WORD3 &= (uint32_t)~0x01UL;
    /* MPU_RGD3_WORD3: VLD=0 */
    MPU_RGD3_WORD3 &= (uint32_t)~0x01UL;
    /* MPU_RGD4_WORD3: VLD=0 */
    MPU_RGD4_WORD3 &= (uint32_t)~0x01UL;
    /* MPU_RGD5_WORD3: VLD=0 */
    MPU_RGD5_WORD3 &= (uint32_t)~0x01UL;
    /* MPU_RGD6_WORD3: VLD=0 */
    MPU_RGD6_WORD3 &= (uint32_t)~0x01UL;
    /* MPU_RGD7_WORD3: VLD=0 */
    MPU_RGD7_WORD3 &= (uint32_t)~0x01UL;
    /* MPU_RGD8_WORD3: VLD=0 */
    MPU_RGD8_WORD3 &= (uint32_t)~0x01UL;
    /* MPU_RGD9_WORD3: VLD=0 */
    MPU_RGD9_WORD3 &= (uint32_t)~0x01UL;
    /* MPU_RGD10_WORD3: VLD=0 */
    MPU_RGD10_WORD3 &= (uint32_t)~0x01UL;
}

```

```

/* MPU_RGD11_WORD3: VLD=0 */
MPU_RGD11_WORD3 &= (uint32_t)~0x01UL;
/* MPU_CESR: SPERR=0,VLD=1 */
MPU_CESR = (uint32_t)((MPU_CESR & (uint32_t)~0xF8000000UL) | (uint32_t)0x01UL);
    /* Initialization of the PMC module */
/* PMC_LVDSC1: LVDACK=1,LVDIE=0,LVDRE=1,LVDV=0 */
PMC_LVDSC1 = (uint8_t)((PMC_LVDSC1 & (uint8_t)~(uint8_t)0x23U) | (uint8_t)0x50U);
/* PMC_LVDSC2: LVWACK=1,LVWIE=0,LVWV=0 */
PMC_LVDSC2 = (uint8_t)((PMC_LVDSC2 & (uint8_t)~(uint8_t)0x23U) | (uint8_t)0x40U);
/* PMC_REGSC: TRAMPO=0,BGBE=0 */
PMC_REGSC &= (uint8_t)~(uint8_t)0x13U;
/* MC_PMPROT: AVLP=0,ALLS=0,AVLLS3=0,AVLLS2=0,AVLLS1=0 */
MC_PMPROT = (uint8_t)0x00U;      /* Setup Power mode protection register */
/* Common initialization of the CPU registers */
/* NVICIP57: PRI57=0 */
NVICIP57 = (uint8_t)0x00U;
/* NVICIP58: PRI58=0 */
NVICIP58 = (uint8_t)0x00U;
/* NVICIP62: PRI62=0 */
NVICIP62 = (uint8_t)0x00U;
/* NVICIP81: PRI81=0 */
NVICIP81 = (uint8_t)0x00U;
/* NVICIP82: PRI82=0 */
NVICIP82 = (uint8_t)0x00U;
/* NVICIP63: PRI63=0 */
NVICIP63 = (uint8_t)0x00U;
/* NVICIP64: PRI64=0x70 */
NVICIP64 = (uint8_t)0x70U;
/* NVICIP72: PRI72=0x90 */
NVICIP72 = (uint8_t)0x90U;
/* NVICIP68: PRI68=0x60 */
NVICIP68 = (uint8_t)0x60U;
/* NVICIP69: PRI69=0x60 */
NVICIP69 = (uint8_t)0x60U;
/* NVICIP70: PRI70=0x60 */
NVICIP70 = (uint8_t)0x60U;
/* NVICIP71: PRI71=0x60 */
NVICIP71 = (uint8_t)0x60U;
/* NVICIP87: PRI87=0x40 */
NVICIP87 = (uint8_t)0x40U;
/* NVICIP90: PRI90=0x50 */
NVICIP90 = (uint8_t)0x50U;
/* NVICIP26: PRI26=0 */
NVICIP26 = (uint8_t)0x00U;
/* NVICIP53: PRI53=0xA0 */
NVICIP53 = (uint8_t)0xA0U;
/* NVICIP54: PRI54=0xF0 */
NVICIP54 = (uint8_t)0xF0U;
/* NVICIP20: PRI20=0 */
NVICIP20 = (uint8_t)0x00U;
/* PORTA_PCR7: ISF=0,MUX=0 */
PORTA_PCR7 &= (uint32_t)~0x01000700UL;
/* PORTB_PCR2: ISF=0,MUX=0 */
PORTB_PCR2 &= (uint32_t)~0x01000700UL;
/* PORTB_PCR3: ISF=0,MUX=0 */
PORTB_PCR3 &= (uint32_t)~0x01000700UL;
/* PORTB_PCR0: ISF=0,MUX=0 */

```

```

PORTB_PCR0 &= (uint32_t)~0x01000700UL;
/* PORTB_PCR4: ISF=0,MUX=0 */
PORTB_PCR4 &= (uint32_t)~0x01000700UL;
/* PORTB_PCR5: ISF=0,MUX=0 */
PORTB_PCR5 &= (uint32_t)~0x01000700UL;
/* PORTA_PCR8: ISF=0,MUX=3 */
PORTA_PCR8 = (uint32_t)((PORTA_PCR8 & (uint32_t)~0x01000400UL) | (uint32_t)0x0300UL);
/* PORTA_PCR9: ISF=0,MUX=3 */
PORTA_PCR9 = (uint32_t)((PORTA_PCR9 & (uint32_t)~0x01000400UL) | (uint32_t)0x0300UL);
/* NVICISER2: SETENA |= 0x048001F1 */
NVICISER2 |= (uint32_t)0x048001F1UL;
/* PORTA_PCR4: ISF=0,IRQC=0,MUX=1,ODE=0,PE=1,PS=1 */
PORTA_PCR4 = (uint32_t)((PORTA_PCR4 & (uint32_t)~0x010F0620UL) | (uint32_t)0x0103UL);
/* PORTA_PCR6: ISF=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTA_PCR6 = (uint32_t)((PORTA_PCR6 & (uint32_t)~0x01000626UL) | (uint32_t)0x0140UL);
/* PORTA_PCR10: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=1,SRE=0,PE=0 */
PORTA_PCR10 = (uint32_t)((PORTA_PCR10 & (uint32_t)~0x010F0606UL) | (uint32_t)0x0160UL);
/* PORTA_PCR11: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=1,SRE=0,PE=0 */
PORTA_PCR11 = (uint32_t)((PORTA_PCR11 & (uint32_t)~0x010F0606UL) | (uint32_t)0x0160UL);
/* PORTA_PCR12: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=1,SRE=0,PE=0 */
PORTA_PCR12 = (uint32_t)((PORTA_PCR12 & (uint32_t)~0x010F0606UL) | (uint32_t)0x0160UL);
/* PORTA_PCR13: ISF=0,IRQC=0,MUX=1,DSE=0,ODE=1,SRE=0 */
PORTA_PCR13 = (uint32_t)((PORTA_PCR13 & (uint32_t)~0x010F0644UL) | (uint32_t)0x0120UL);
/* PORTA_PCR14: ISF=0,IRQC=0,MUX=1,ODE=0,PE=1,PS=1 */
PORTA_PCR14 = (uint32_t)((PORTA_PCR14 & (uint32_t)~0x010F0620UL) | (uint32_t)0x0103UL);
/* PORTA_PCR24: ISF=0,IRQC=0,MUX=1,ODE=1,PFE=0,SRE=0,PE=0 */
PORTA_PCR24 = (uint32_t)((PORTA_PCR24 & (uint32_t)~0x010F0616UL) | (uint32_t)0x0120UL);
/* PORTA_DFER: DFE&=~0x01000000 */
PORTA_DFER &= (uint32_t)~0x01000000UL;
/* PORTA_DFCR: CS=0 */
PORTA_DFCR &= (uint32_t)~0x01UL;
/* PORTA_DFWR: FILT=0 */
PORTA_DFWR &= (uint32_t)~0x1FUL;
/* PORTB_PCR10: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=1,PFE=0,SRE=0,PE=0 */
PORTB_PCR10 = (uint32_t)((PORTB_PCR10 & (uint32_t)~0x010F0616UL) | (uint32_t)0x0160UL);
/* PORTB_PCR11: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=1,PFE=0,SRE=0,PE=0 */
PORTB_PCR11 = (uint32_t)((PORTB_PCR11 & (uint32_t)~0x010F0616UL) | (uint32_t)0x0160UL);
/* PORTB_PCR16: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTB_PCR16 = (uint32_t)((PORTB_PCR16 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTB_DFER: DFE&=~0x0C00 */
PORTB_DFER &= (uint32_t)~0x0C00UL;
/* PORTB_DFCR: CS=0 */
PORTB_DFCR &= (uint32_t)~0x01UL;
/* PORTB_DFWR: FILT=0 */
PORTB_DFWR &= (uint32_t)~0x1FUL;
/* PORTC_PCR0: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTC_PCR0 = (uint32_t)((PORTC_PCR0 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTC_PCR5: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTC_PCR5 = (uint32_t)((PORTC_PCR5 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTC_PCR6: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTC_PCR6 = (uint32_t)((PORTC_PCR6 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTC_PCR7: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTC_PCR7 = (uint32_t)((PORTC_PCR7 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTC_PCR8: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=1,SRE=0,PE=0 */
PORTC_PCR8 = (uint32_t)((PORTC_PCR8 & (uint32_t)~0x010F0606UL) | (uint32_t)0x0160UL);
/* PORTC_PCR9: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTC_PCR9 = (uint32_t)((PORTC_PCR9 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);

```

```

/* PORTC_PCR10: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTC_PCR10 = (uint32_t)((PORTC_PCR10 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTC_DFCR: CS=0 */
PORTC_DFCR &= (uint32_t)~0x01UL;
/* PORTC_DFWR: FILT=0 */
PORTC_DFWR &= (uint32_t)~0x1FUL;
/* PORTD_PCR4: ISF=0,IRQC=9,MUX=1,ODE=1,PFE=0,SRE=0,PE=0 */
PORTD_PCR4 = (uint32_t)((PORTD_PCR4 & (uint32_t)~0x01060616UL) | (uint32_t)0x00090120UL);
/* PORTD_PCR5: ISF=0,IRQC=0,MUX=1,ODE=1,SRE=0,PE=0 */
PORTD_PCR5 = (uint32_t)((PORTD_PCR5 & (uint32_t)~0x010F0606UL) | (uint32_t)0x0120UL);
/* PORTD_DFER: DFE&=~0x10 */
PORTD_DFER &= (uint32_t)~0x10UL;
/* PORTD_DFCR: CS=0 */
PORTD_DFCR &= (uint32_t)~0x01UL;
/* PORTD_DFWR: FILT=0 */
PORTD_DFWR &= (uint32_t)~0x1FUL;
/* PORTE_PCR0: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTE_PCR0 = (uint32_t)((PORTE_PCR0 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTE_PCR1: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTE_PCR1 = (uint32_t)((PORTE_PCR1 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTE_PCR2: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTE_PCR2 = (uint32_t)((PORTE_PCR2 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTE_PCR3: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTE_PCR3 = (uint32_t)((PORTE_PCR3 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTE_PCR4: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTE_PCR4 = (uint32_t)((PORTE_PCR4 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTE_PCR5: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTE_PCR5 = (uint32_t)((PORTE_PCR5 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTE_PCR6: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTE_PCR6 = (uint32_t)((PORTE_PCR6 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTE_PCR7: ISF=0,IRQC=0,MUX=1,DSE=1,ODE=0,SRE=0,PE=0 */
PORTE_PCR7 = (uint32_t)((PORTE_PCR7 & (uint32_t)~0x010F0626UL) | (uint32_t)0x0140UL);
/* PORTE_DFCR: CS=0 */
PORTE_DFCR &= (uint32_t)~0x01UL;
/* PORTE_DFWR: FILT=0 */
PORTE_DFWR &= (uint32_t)~0x1FUL;
/* PORTA_PCR17: ISF=0,MUX=2 */
PORTA_PCR17 = (uint32_t)((PORTA_PCR17 & (uint32_t)~0x01000500UL) | (uint32_t)0x0200UL);
/* PORTA_PCR16: ISF=0,MUX=2 */
PORTA_PCR16 = (uint32_t)((PORTA_PCR16 & (uint32_t)~0x01000500UL) | (uint32_t)0x0200UL);
/* PORTA_PCR15: ISF=0,MUX=2 */
PORTA_PCR15 = (uint32_t)((PORTA_PCR15 & (uint32_t)~0x01000500UL) | (uint32_t)0x0200UL);
/* PORTC_PCR14: ISF=0,MUX=3 */
PORTC_PCR14 = (uint32_t)((PORTC_PCR14 & (uint32_t)~0x01000400UL) | (uint32_t)0x0300UL);
/* PORTC_PCR15: ISF=0,MUX=3 */
PORTC_PCR15 = (uint32_t)((PORTC_PCR15 & (uint32_t)~0x01000400UL) | (uint32_t)0x0300UL);
/* NVICISER1: SETENA|=0x00200000 */
NVICISER1 |= (uint32_t)0x00200000UL;
/* ### Init_ADC init code */
/* SIM_SCGC6: ADC0=1 */
SIM_SCGC6 |= (uint32_t)0x08000000UL;
/* ADC0_CFG1: ADLPC=0,ADIV=0,ADLSMP=0,MODE=0,ADICLK=1 */
ADC0_CFG1 = (uint32_t)0x01UL;
/* ADC0_CFG2: ADACKEN=0,ADHSC=0,ADLSTS=0 */
ADC0_CFG2 &= (uint32_t)~0xFFFFFEFUL;
/* ADC0_CV1: CV=0 */
ADC0_CV1 = (uint32_t)0x00UL;

```

```

/* ADC0_CV2: CV=0 */
ADC0_CV2 = (uint32_t)0x00UL;
/* ADC0_SC2: ADACT=0,ADTRG=0,ACFE=0,ACFGT=0,ACREN=0,DMAEN=0,REFSEL=0 */
ADC0_SC2 = (uint32_t)0x00UL;
/* ADC0_SC3: CAL=0,CALF=0,ADCO=0,AVGE=0,AVGS=0 */
ADC0_SC3 = (uint32_t)0x00UL;
/* ADC0_SC1A: COCO=0,AIEN=0,DIFF=0,ADCH=0x1F */
ADC0_SC1A = (uint32_t)0x1FUL;
/* ADC0_SC1B: COCO=0,AIEN=0,DIFF=0,ADCH=0x1F */
ADC0_SC1B = (uint32_t)0x1FUL;
/* ### Init_ADC init code */
/* SIM_SCGC3: ADC1=1 */
SIM_SCGC3 |= (uint32_t)0x08000000UL;
/* ADC1_CFG1: ADLPC=0,ADIV=0,ADLSMP=0,MODE=0,ADICLK=1 */
ADC1_CFG1 = (uint32_t)0x01UL;
/* ADC1_CFG2: ADACKEN=0,ADHSC=0,ADLSTS=0 */
ADC1_CFG2 &= (uint32_t)~0xFFFFFFFUL;
/* ADC1.CV1: CV=0 */
ADC1.CV1 = (uint32_t)0x00UL;
/* ADC1.CV2: CV=0 */
ADC1.CV2 = (uint32_t)0x00UL;
/* ADC1_SC2: ADACT=0,ADTRG=0,ACFE=0,ACFGT=0,ACREN=0,DMAEN=0,REFSEL=0 */
ADC1_SC2 = (uint32_t)0x00UL;
/* ADC1_SC3: CAL=0,CALF=0,ADCO=0,AVGE=0,AVGS=0 */
ADC1_SC3 = (uint32_t)0x00UL;
/* ADC1_SC1A: COCO=0,AIEN=0,DIFF=0,ADCH=0x1F */
ADC1_SC1A = (uint32_t)0x1FUL;
/* ADC1_SC1B: COCO=0,AIEN=0,DIFF=0,ADCH=0x1F */
ADC1_SC1B = (uint32_t)0x1FUL;
/* ### Init_FTM init code */
/* SIM_SCGC6: FTM0=1 */
SIM_SCGC6 |= (uint32_t)0x01000000UL;
(void)(FTM0_SC == 0U);           /* Dummy read of the FTM0_SC register to clear the interrupt flag */
/* FTM0_SC: TOF=0,TOIE=0,CPWMS=0,CLKS=0,PS=0 */
FTM0_SC = (uint32_t)0x00UL;      /* Stop the counter */
/* FTM0_COSC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM0_COSC = (uint32_t)0x00UL;
/* FTM0_C1SC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM0_C1SC = (uint32_t)0x00UL;
/* FTM0_C2SC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM0_C2SC = (uint32_t)0x00UL;
/* FTM0_C3SC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM0_C3SC = (uint32_t)0x00UL;
/* FTM0_C4SC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM0_C4SC = (uint32_t)0x00UL;
/* FTM0_C5SC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM0_C5SC = (uint32_t)0x00UL;
/* FTM0_C6SC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM0_C6SC = (uint32_t)0x00UL;
/* FTM0_C7SC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM0_C7SC = (uint32_t)0x00UL;
/* FTM0_MODE: WPDIS=1 */
FTM0_MODE |= (uint32_t)0x04UL;    /* Disable write protection */
/* FTM0_MODE: FTMEN=1 */
FTM0_MODE |= (uint32_t)0x01UL;

```

```

/* FTM0_COMBINE:
FAULTEN3=0,SYNCEN3=0,DTEN3=0,COMP3=0,FAULTEN2=0,SYNCEN2=0,DTEN2=0,COMP2=0,FAULTEN1=0,SYNC
EN1=0,DTEN1=0,COMP1=0,FAULTENO=0,SYNCENO=0,DTENO=0,COMP0=0 */
FTM0_COMBINE &= (uint32_t)~0x72727272UL;
/* FTM0_INVCTRL: INV3EN=0,INV2EN=0,INV1EN=0,INV0EN=0 */
FTM0_INVCTRL &= (uint32_t)~0x0FUL;
/* FTM0_COSC: CHF=0,MSB=0,MSA=0,ELSB=0,ELSA=0 */
FTM0_COSC &= (uint32_t)~0xFFFFFBEBUL;
/* FTM0_C1SC: CHF=0,MSB=0,MSA=0,ELSB=0,ELSA=0 */
FTM0_C1SC &= (uint32_t)~0xFFFFFBEBUL;
/* FTM0_C2SC: CHF=0,MSB=0,MSA=0,ELSB=0,ELSA=0 */
FTM0_C2SC &= (uint32_t)~0xFFFFFBEBUL;
/* FTM0_C3SC: CHF=0,MSB=0,MSA=0,ELSB=0,ELSA=0 */
FTM0_C3SC &= (uint32_t)~0xFFFFFBEBUL;
/* FTM0_C4SC: CHF=0,MSB=0,MSA=0,ELSB=0,ELSA=0 */
FTM0_C4SC &= (uint32_t)~0xFFFFFBEBUL;
/* FTM0_C5SC: CHF=0,MSB=0,MSA=0,ELSB=0,ELSA=0 */
FTM0_C5SC &= (uint32_t)~0xFFFFFBEBUL;
/* FTM0_C6SC: CHF=0,MSB=0,MSA=0,ELSB=0,ELSA=0 */
FTM0_C6SC &= (uint32_t)~0xFFFFFBEBUL;
/* FTM0_C7SC: CHF=0,MSB=0,MSA=0,ELSB=0,ELSA=0 */
FTM0_C7SC &= (uint32_t)~0xFFFFFBEBUL;
/* FTM0_OUTINIT: */
FTM0_OUTINIT &= (uint32_t)~0xFFFFFFFF00UL;
/* FTM0_FLTCTRL: FFVAL=0 */
FTM0_FLTCTRL &= (uint32_t)~0x0F00UL;
/* FTM0_FLTPOL: FLT3POL=0,FLT2POL=0,FLT1POL=0,FLT0POL=0 */
FTM0_FLTPOL &= (uint32_t)~0x0FUL;
/* FTM0_FLTCTRL:
FFLTR3EN=0,FFLTR2EN=0,FFLTR1EN=0,FFLTR0EN=0,FAULT3EN=0,FAULT2EN=0,FAULT1EN=0,FAULT0EN=0 */
FTM0_FLTCTRL &= (uint32_t)~0xFFUL;
/* FTM0_SYNC: SWSYNC=0,TRIG2=0,TRIG1=0,TRIG0=0,SYNCHOM=0,REINIT=0,CNTMAX=0,CNTMIN=0 */
FTM0_SYNC &= (uint32_t)~0xFFUL;
/* FTM0_EXTTRIG: INITTRIGEN=0,CH1TRIG=0,CH0TRIG=0,CH5TRIG=0,CH4TRIG=0,CH3TRIG=0,CH2TRIG=0 */
FTM0_EXTTRIG &= (uint32_t)~0x7FUL;
/* FTM0_MOD: MOD=0xFFFF */
FTM0_MOD |= (uint32_t)0xFFFFFUL;
/* FTM0_CNTIN: INIT=0 */
FTM0_CNTIN &= (uint32_t)~0xFFFFFUL;
/* FTM0_DEADTIME: DTPS=0,DTVAL=0 */
FTM0_DEADTIME &= (uint32_t)~0xFFUL;
/* FTM0_CNT: COUNT=0 */
FTM0_CNT &= (uint32_t)~0xFFFFFUL;
/* FTM0_MODE: FAULTIE=0,FAULTM=0,PWMSYNC=0,INIT=0,FTMEN=1 */
FTM0_MODE = (uint32_t)((FTM0_MODE & (uint32_t)~0xFFFFFEAUL) | (uint32_t)0x01UL);
/* FTM0_SYNCONF: SYNCMODE=0,SWOC=0,INV=0,CNTINC=0,HWTRIGMODE=0 */
FTM0_SYNCONF &= (uint32_t)~0xFFE0E0FFUL;
/* FTM0_SC: TOF=0,TOIE=0,CLKS=1,PS=6 */
FTM0_SC = (uint32_t)((FTM0_SC & (uint32_t)~0xD1UL) | (uint32_t)0x0EUL);
/* FTM0_FMS: FAULTF=0,WPEN=0,FAULTF3=0,FAULTF2=0,FAULTF1=0,FAULTF0=0 */
FTM0_FMS &= (uint32_t)~0xCFUL;
/* FTM0_CONF: GTBEOUT=0,GTBEEN=0,BDMMODE=3,NUMTOF=0 */
FTM0_CONF = (uint32_t)((FTM0_CONF & (uint32_t)~0x061FUL) | (uint32_t)0xC0UL);
/* FTM0_PWMLOAD:
LDOK=1,CH7SEL=0,CH6SEL=0,CH5SEL=0,CH4SEL=0,CH3SEL=0,CH2SEL=0,CH1SEL=0,CHOSEL=0 */
FTM0_PWMLOAD = (uint32_t)((FTM0_PWMLOAD & (uint32_t)~0xFFUL) | (uint32_t)0x0200UL);
/* ### Init_DAC init code */

```

```

/* SIM_SCGC2: DAC0=1 */
SIM_SCGC2 |= (uint32_t)0x1000UL;
/* DAC0_CO:
Dacen=0,DACRFS=0,DACTRGSEL=0,DACSWTRG=0,LPEN=0,DACBWIEN=0,DACBTIEN=0,DACBBIEN=0 */
DAC0_CO = (uint8_t)0x00U;
/* DAC0_SR: DACBFWMF=0,DACBFRPTF=0,DACBFRPBF=0 */
DAC0_SR = (uint8_t)0x00U;
/* DAC0_DATOL: DATA=0 */
DAC0_DATOL = (uint8_t)0x00U;
/* DAC0_DATOH: DATA=8 */
DAC0_DATOH = (uint8_t)0x08U;
/* DAC0_C1: DMAEN=0,DACBFWM=0,DACBFMD=0,DACBFEN=0 */
DAC0_C1 = (uint8_t)0x00U;
/* DAC0_CO:
Dacen=1,DACRFS=0,DACTRGSEL=1,DACSWTRG=0,LPEN=0,DACBWIEN=0,DACBTIEN=0,DACBBIEN=0 */
DAC0_CO = (uint8_t)0xA0U;
/* ### Init_DAC init code */
/* SIM_SCGC2: DAC1=1 */
SIM_SCGC2 |= (uint32_t)0x2000UL;
/* DAC1_CO:
Dacen=0,DACRFS=0,DACTRGSEL=0,DACSWTRG=0,LPEN=0,DACBWIEN=0,DACBTIEN=0,DACBBIEN=0 */
DAC1_CO = (uint8_t)0x00U;
/* DAC1_SR: DACBFWMF=0,DACBFRPTF=0,DACBFRPBF=0 */
DAC1_SR = (uint8_t)0x00U;
/* DAC1_DATOL: DATA=0 */
DAC1_DATOL = (uint8_t)0x00U;
/* DAC1_DATOH: DATA=8 */
DAC1_DATOH = (uint8_t)0x08U;
/* DAC1_C1: DMAEN=0,DACBFWM=0,DACBFMD=0,DACBFEN=0 */
DAC1_C1 = (uint8_t)0x00U;
/* DAC1_CO:
Dacen=1,DACRFS=0,DACTRGSEL=1,DACSWTRG=0,LPEN=0,DACBWIEN=0,DACBTIEN=0,DACBBIEN=0 */
DAC1_CO = (uint8_t)0xA0U;
/* ### Init_FTM init code */
/* SIM_SCGC6: FTM1=1 */
SIM_SCGC6 |= (uint32_t)0x20000000UL;
(void)(FTM1_SC == 0U); /* Dummy read of the FTM1_SC register to clear the interrupt flag */
/* FTM1_SC: TOF=0,TOIE=0,CPWMS=0,CLKS=0,PS=0 */
FTM1_SC = (uint32_t)0x00UL; /* Stop the counter */
(void)(FTM1_COSC == 0U); /* Dummy read of the FTM1_COSC register to clear the interrupt flag */
(void)(FTM1_C1SC == 0U); /* Dummy read of the FTM1_C1SC register to clear the interrupt flag */
/* FTM1_COSC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM1_COSC = (uint32_t)0x00UL;
/* FTM1_C1SC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM1_C1SC = (uint32_t)0x00UL;
/* FTM1_SC: TOF=0,CPWMS=1 */
FTM1_SC = (uint32_t)((FTM1_SC & (uint32_t)~0x80UL) | (uint32_t)0x20UL);
/* FTM1_MODE: WPDIS=1 */
FTM1_MODE |= (uint32_t)0x04UL; /* Disable write protection */
/* FTM1_MODE: FTMEN=0 */
FTM1_MODE &= (uint32_t)~0x01UL;
/* FTM1_INVCTRL: INVOEN=0 */
FTM1_INVCTRL &= (uint32_t)~0x01UL;
/* FTM1_COMBINE: FAULTENO=0,SYNCENO=0,DTENO=0,DECAPENO=0,COMP0=0,COMBINE0=0 */
FTM1_COMBINE &= (uint32_t)~0x77UL;
/* FTM1_COSC: CHF=0,CHIE=0,ELSB=1,ELSA=0,DMA=0 */
FTM1_COSC = (uint32_t)((FTM1_COSC & (uint32_t)~0xFFFFFC7UL) | (uint32_t)0x08UL);

```

```

/* FTM1_C1SC: CHF=0,CHIE=0,ELSB=1,ELSA=0,DMA=0 */
FTM1_C1SC = (uint32_t)((FTM1_C1SC & (uint32_t)~0xFFFFFC7UL) | (uint32_t)0x08UL);
/* FTM1_COV: VAL=0 */
FTM1_COV = (uint32_t)0x00UL;
/* FTM1_C1V: VAL=0 */
FTM1_C1V = (uint32_t)0x00UL;
/* FTM1_FILTER: CH1FVAL=0,CH0FVAL=0 */
FTM1_FILTER &= (uint32_t)~0xFFUL;
/* FTM1_OUTINIT: */
FTM1_OUTINIT &= (uint32_t)~0xFFFFFFFF00UL;
/* FTM1_FLTCTRL: FFVAL=0 */
FTM1_FLTCTRL &= (uint32_t)~0x0F00UL;
/* FTM1_FLTPOL: FLT2POL=0,FLT1POL=0,FLT0POL=0 */
FTM1_FLTPOL &= (uint32_t)~0x07UL;
/* FTM1_FLTCTRL: FFLTR2EN=0,FFLTR1EN=0,FFLTR0EN=0,FAULT2EN=0,FAULT1EN=0,FAULT0EN=0 */
FTM1_FLTCTRL &= (uint32_t)~0x77UL;
/* FTM1_SYNC: SWSYNC=0,TRIG2=0,TRIG1=0,TRIG0=0,SYNCHOM=0,REINIT=0,CNTMAX=0,CNTMIN=0 */
FTM1_SYNC &= (uint32_t)~0xFFUL;
/* FTM1_EXTTRIG: INITTRIGEN=0,CH1TRIG=0,CH0TRIG=0 */
FTM1_EXTTRIG &= (uint32_t)~0x70UL;
/* FTM1_MOD: MOD=0x30D4 */
FTM1_MOD = (uint32_t)((FTM1_MOD & (uint32_t)~0xCF2BUL) | (uint32_t)0x30D4UL);
/* FTM1_CNTIN: INIT=0 */
FTM1_CNTIN &= (uint32_t)~0xFFFFFUL;
/* FTM1_DEADTIME: DTPS=0,DTVAL=0 */
FTM1_DEADTIME &= (uint32_t)~0xFFUL;
/* FTM1_CNT: COUNT=0 */
FTM1_CNT &= (uint32_t)~0xFFFFFUL;
/* FTM1_MODE: FAULTIE=0,FAULTTM=0,PWMSYNC=0,INIT=0,FTMEN=0 */
FTM1_MODE &= (uint32_t)~0xFFFFFEFUL;
/* FTM1_SYNCONF: SYNCMODE=0,SWOC=0,INV=0,CNTINC=0,HWTRIGMODE=0 */
FTM1_SYNCONF &= (uint32_t)~0xFFE0E0FFUL;
/* FTM1_SC: TOF=0,TOIE=0,CPWMS=1,CLKS=1,PS=0 */
FTM1_SC = (uint32_t)((FTM1_SC & (uint32_t)~0xD7UL) | (uint32_t)0x28UL);
/* FTM1_FMS: FAULTF=0,WOPEN=0,FAULTF3=0,FAULTF2=0,FAULTF1=0,FAULTF0=0 */
FTM1_FMS &= (uint32_t)~0xCFUL;
/* FTM1_QDCTRL:
PHAFLTREN=0,PHBFLTREN=0,PHAPOL=0,PHBPOL=0,QUADMODE=0,QUADIR=0,TOFDIR=0,QUADEN=0 */
FTM1_QDCTRL = (uint32_t)0x00UL;
/* FTM1_SWOCTRL: CH1OC=0,CH0OC=0 */
FTM1_SWOCTRL &= (uint32_t)~0x03UL;
/* FTM1_CONF: GTBEOUT=0,GTBEEN=0,BDMMODE=0,NUMTOF=0 */
FTM1_CONF &= (uint32_t)~0x06DFUL;
/* FTM1_PWMLOAD: LDOK=0,CH1SEL=0,CHOSEL=0 */
FTM1_PWMLOAD &= (uint32_t)~0x0203UL;
/* ### Init_FTM init code */
/* SIM_SCGC3: FTM2=1 */
SIM_SCGC3 |= (uint32_t)0x010000000UL;
(void)(FTM2_SC == 0U); /* Dummy read of the FTM2_SC register to clear the interrupt flag */
/* FTM2_SC: TOF=0,TOIE=0,CPWMS=0,CLKS=0,PS=0 */
FTM2_SC = (uint32_t)0x00UL; /* Stop the counter */
/* FTM2_COSC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM2_COSC = (uint32_t)0x00UL;
/* FTM2_C1SC: CHF=0,CHIE=0,MSB=0,MSA=0,ELSB=0,ELSA=0,DMA=0 */
FTM2_C1SC = (uint32_t)0x00UL;
/* FTM2_MODE: WPDIS=1 */
FTM2_MODE |= (uint32_t)0x04UL; /* Disable write protection */

```

```

/* FTM2_MODE: FTMEN=1 */
FTM2_MODE |= (uint32_t)0x01UL;
/* FTM2_INVCTRL: INVOEN=0 */
FTM2_INVCTRL &= (uint32_t)~0x01UL;
/* FTM2_COMBINE: FAULTENO=0,SYNCENO=0,DTENO=0,COMP0=0 */
FTM2_COMBINE &= (uint32_t)~0x72UL;
/* FTM2_COSC: CHF=0,MSB=0,MSA=0,ELSB=0,ELSA=0 */
FTM2_COSC &= (uint32_t)~0xFFFFFFFFBEUL;
/* FTM2_C1SC: CHF=0,MSB=0,MSA=0,ELSB=0,ELSA=0 */
FTM2_C1SC &= (uint32_t)~0xFFFFFFFFBEUL;
/* FTM2_OUTINIT: */
FTM2_OUTINIT &= (uint32_t)~0xFFFFFFFF00UL;
/* FTM2_FLTCTRL: FFVAL=0 */
FTM2_FLTCTRL &= (uint32_t)~0x0F00UL;
/* FTM2_FLTPOL: FLT2POL=0,FLT1POL=0,FLT0POL=0 */
FTM2_FLTPOL &= (uint32_t)~0x07UL;
/* FTM2_FLTCTRL: FFLTR2EN=0,FFLTR1EN=0,FFLTR0EN=0,FAULT2EN=0,FAULT1EN=0,FAULT0EN=0 */
FTM2_FLTCTRL &= (uint32_t)~0x77UL;
/* FTM2_SYNC: SWSYNC=0,TRIG2=0,TRIG1=0,TRIG0=0,SYNCHOM=0,REINIT=0,CNTMAX=0,CNTMIN=0 */
FTM2_SYNC &= (uint32_t)~0xFFUL;
/* FTM2_EXTRIG: INITTRIGEN=0,CH1TRIG=0,CH0TRIG=0 */
FTM2_EXTRIG &= (uint32_t)~0x70UL;
/* FTM2_MOD: MOD=0xFFFF */
FTM2_MOD |= (uint32_t)0xFFFFUL;
/* FTM2_CNTIN: INIT=0 */
FTM2_CNTIN &= (uint32_t)~0xFFFFUL;
/* FTM2_DEADTIME: DTPS=0,DTVAL=0 */
FTM2_DEADTIME &= (uint32_t)~0xFFUL;
/* FTM2_CNT: COUNT=0 */
FTM2_CNT &= (uint32_t)~0xFFFFUL;
/* FTM2_MODE: FAULTIE=0,FAULTM=0,PWMSYNC=0,INIT=0,FTMEN=1 */
FTM2_MODE = (uint32_t)((FTM2_MODE & (uint32_t)~0xFFFFFFFFEAUL) | (uint32_t)0x01UL);
/* FTM2_SYNCONF: SYNCMODE=0,SWOC=0,INV=0,CNTINC=0,HWTRIGMODE=0 */
FTM2_SYNCONF &= (uint32_t)~0xFFE0E0FFUL;
/* FTM2_SC: TOF=0,TOIE=0,CLKS=1,PS=6 */
FTM2_SC = (uint32_t)((FTM2_SC & (uint32_t)~0xD1UL) | (uint32_t)0x0EUL);
/* FTM2_FMS: FAULTF=0,WOPEN=0,FAULTF3=0,FAULTF2=0,FAULTF1=0,FAULTF0=0 */
FTM2_FMS &= (uint32_t)~0x0CFUL;
/* FTM2_QDCTRL: */
PHAFLTREN=0,PHBFLTREN=0,PHAPOL=0,PHBPOL=0,QUADMODE=0,QUADIR=0,TOFDIR=0,QUADEN=0 */
FTM2_QDCTRL = (uint32_t)0x00UL;
/* FTM2_CONF: GTBEO=0,GTBEE=0,BDMMODE=3,NUMTOF=0 */
FTM2_CONF = (uint32_t)((FTM2_CONF & (uint32_t)~0x061FUL) | (uint32_t)0xC0UL);
/* FTM2_PWMLOAD: LDO=1,CH1SEL=0,CHOSEL=0 */
FTM2_PWMLOAD = (uint32_t)((FTM2_PWMLOAD & (uint32_t)~0x03UL) | (uint32_t)0x0200UL);
/* ### Init_PDB init code */
/* SIM_SCGC6: PDB=1 */
SIM_SCGC6 |= (uint32_t)0x00400000UL;
/* PDB0_SC: */
LDMOD=1,PDBEIE=0,SWTRIG=0,DMAEN=0,PRESCALER=0,TRGSEL=0x0F,PDBEN=0,PDBIF=0,PDBIE=1,MULT=1,CO
NT=1,LDO=0 */
PDB0_SC = (uint32_t)0x00040F26UL;
/* PDB0_MOD: MOD=0x1388 */
PDB0_MOD = (uint32_t)0x1388UL;
/* PDB0_IDLY: IDLY=0x1388 */
PDB0_IDLY = (uint32_t)0x1388UL;
/* PDB0_CH0C1: BB&=~3,TOS&=~3,EN&=~3 */

```

```

PDB0_CH0C1 &= (uint32_t)~0x00030303UL;
/* PDB0_CHOS: CF=0,ERR=0xFF */
PDB0_CHOS = (uint32_t)0xFFUL;
/* PDB0_CHODLY0: DLY=0 */
PDB0_CH0DLY0 &= (uint32_t)~0xFFFFUL;
/* PDB0_CH0DLY1: DLY=0 */
PDB0_CH0DLY1 &= (uint32_t)~0xFFFFUL;
/* PDB0_CH1C1: BB&=~3,TOS&=~3,EN&=~3 */
PDB0_CH1C1 &= (uint32_t)~0x00030303UL;
/* PDB0_CH1S: CF=0,ERR=0xFF */
PDB0_CH1S = (uint32_t)0xFFUL;
/* PDB0_CH1DLY0: DLY=0 */
PDB0_CH1DLY0 &= (uint32_t)~0xFFFFUL;
/* PDB0_CH1DLY1: DLY=0 */
PDB0_CH1DLY1 &= (uint32_t)~0xFFFFUL;
/* PDB0_DACINT0: INT=0 */
PDB0_DACINT0 = (uint32_t)0x00UL;
/* PDB0_DACINT1: INT=0 */
PDB0_DACINT1 = (uint32_t)0x00UL;
/* PDB0_DACINTC0: EXT=0,TOE=0 */
PDB0_DACINTC0 = (uint32_t)0x00UL;
/* PDB0_DACINTC1: EXT=0,TOE=0 */
PDB0_DACINTC1 = (uint32_t)0x00UL;
/* PDB0_POODLY: DLY1=0,DLY2=0 */
PDB0_POODLY = (uint32_t)0x00UL;
/* PDB0_POOPEN: POEN&=~1 */
PDB0_POOPEN &= (uint32_t)~0x01UL;
/* PDB0_SC: PDBEN=1,LDOK=1 */
PDB0_SC |= (uint32_t)0x81UL;
/* ### Init_PIT init code */
/* SIM_SCGC6: PIT=1 */
SIM_SCGC6 |= (uint32_t)0x008000000UL;
/* PIT_MCR: MDIS=0,FRZ=1 */
PIT_MCR = (uint32_t)0x01UL;
/* PIT_TFLG0: TIF=1 */
PIT_TFLG0 = (uint32_t)0x01UL;
/* PIT_TFLG1: TIF=1 */
PIT_TFLG1 = (uint32_t)0x01UL;
/* PIT_TFLG2: TIF=1 */
PIT_TFLG2 = (uint32_t)0x01UL;
/* PIT_TFLG3: TIF=1 */
PIT_TFLG3 = (uint32_t)0x01UL;
/* PIT_LDVAL0: TSV=0xFFFFFFFF */
PIT_LDVAL0 = (uint32_t)0xFFFFFFFFUL;
/* PIT_LDVAL1: TSV=0xFFFFFFFF */
PIT_LDVAL1 = (uint32_t)0xFFFFFFFFUL;
/* PIT_LDVAL2: TSV=0xFFFFFFFF */
PIT_LDVAL2 = (uint32_t)0xFFFFFFFFUL;
/* PIT_LDVAL3: TSV=0xFFFFFFFF */
PIT_LDVAL3 = (uint32_t)0xFFFFFFFFUL;
/* PIT_TCTRL0: TIE=0,TEN=0 */
PIT_TCTRL0 = (uint32_t)0x00UL;
/* PIT_TCTRL1: TIE=0,TEN=0 */
PIT_TCTRL1 = (uint32_t)0x00UL;
/* PIT_TCTRL2: TIE=0,TEN=0 */
PIT_TCTRL2 = (uint32_t)0x00UL;
/* PIT_TCTRL3: TIE=0,TEN=0 */

```

```

PIT_TCTRL3 = (uint32_t)0x00UL;
/* ### Init_GPIO init code */
/* GPIOA_PSOR: PTSO&=~0x01001440,PTSO|=0x4810 */
GPIOA_PSOR = (uint32_t)((GPIOA_PSOR & (uint32_t)~0x01001440UL) | (uint32_t)0x4810UL);
/* GPIOA_PCOR: PTCO&=~0x4810,PTCO|=0x01001440 */
GPIOA_PCOR = (uint32_t)((GPIOA_PCOR & (uint32_t)~0x4810UL) | (uint32_t)0x01001440UL);
/* GPIOA_PDDR: PDD&=~0x01002000,PDD|=0x5C50 */
GPIOA_PDDR = (uint32_t)((GPIOA_PDDR & (uint32_t)~0x01002000UL) | (uint32_t)0x5C50UL);
/* ### Init_GPIO init code */
/* GPIOB_PSOR: PTSO&=~0x00010000,PTSO|=0x0C00 */
GPIOB_PSOR = (uint32_t)((GPIOB_PSOR & (uint32_t)~0x00010000UL) | (uint32_t)0x0C00UL);
/* GPIOB_PCOR: PTCO&=~0x0C00,PTCO|=0x00010000 */
GPIOB_PCOR = (uint32_t)((GPIOB_PCOR & (uint32_t)~0x0C00UL) | (uint32_t)0x00010000UL);
/* GPIOB_PDDR: PDD|=0x00010C00 */
GPIOB_PDDR |= (uint32_t)0x00010C00UL;
/* ### Init_GPIO init code */
/* GPIOC_PSOR: PTSO&=~0x07E1 */
GPIOC_PSOR &= (uint32_t)~0x07E1UL;
/* GPIOC_PCOR: PTCO|=0x07E1 */
GPIOC_PCOR |= (uint32_t)0x07E1UL;
/* GPIOC_PDDR: PDD|=0x07E1 */
GPIOC_PDDR |= (uint32_t)0x07E1UL;
/* ### Init_GPIO init code */
/* GPIOD_PSOR: PTSO&=~0x30 */
GPIOD_PSOR &= (uint32_t)~0x30UL;
/* GPIOD_PCOR: PTCO|=0x30 */
GPIOD_PCOR |= (uint32_t)0x30UL;
/* GPIOD_PDDR: PDD&=~0x30 */
GPIOD_PDDR &= (uint32_t)~0x30UL;
/* ### Init_GPIO init code */
/* GPIOE_PSOR: PTSO&=~0xFF */
GPIOE_PSOR &= (uint32_t)~0xFFUL;
/* GPIOE_PCOR: PTCO|=0xFF */
GPIOE_PCOR |= (uint32_t)0xFFUL;
/* GPIOE_PDDR: PDD|=0xFF */
GPIOE_PDDR |= (uint32_t)0xFFUL;
/* ### Init_SPI init code */
/* SIM_SCGC6: DSPIO=1 */
SIM_SCGC6 |= (uint32_t)0x1000UL;
/* SPI0_MCR:
MSTR=0,CONT_SCKE=0,DCONF=0,FRZ=0,MTFE=0,PCSSE=0,ROOE=0,PCSIS=0,DOZE=0,MDIS=0,DIS_TXF=0,DIS_RX
F=0,CLR_TXF=0,CLR_RXF=0,SMPL_PT=0,HALT=1 */
SPI0_MCR = (uint32_t)0x01UL;
/* SPI0_MCR:
MSTR=1,CONT_SCKE=0,DCONF=0,FRZ=0,MTFE=0,PCSSE=0,ROOE=1,PCSIS=0,DOZE=0,MDIS=0,DIS_TXF=0,DIS_RX
F=0,CLR_TXF=1,CLR_RXF=1,SMPL_PT=0,HALT=1 */
SPI0_MCR = (uint32_t)0x81000C01UL;
/* SPI0_TCR: SPI_TCNT=0 */
SPI0_TCR = (uint32_t)0x00UL;
/* SPI0_CTAR0:
DBR=1,FMSZ=0x0F,CPOL=0,CPHA=0,LSBFE=0,PCSSCK=0,PASC=0,PDT=0,PBR=2,CSSCK=0,ASC=0,DT=0,BR=4 */
SPI0_CTAR0 = (uint32_t)0xF8020004UL;
/* SPI0_CTAR1:
DBR=1,FMSZ=0x0F,CPOL=0,CPHA=0,LSBFE=0,PCSSCK=0,PASC=0,PDT=0,PBR=2,CSSCK=0,ASC=0,DT=0,BR=4 */
SPI0_CTAR1 = (uint32_t)0xF8020004UL;
/* SPI0_SR:
TCF=1,TXRXS=0,EOQF=1,TFUF=1,TFFF=1,RFOF=1,RFDF=1,TXCTR=0,TXNXTPTR=0,RXCTR=0,POPNXTPTR=0 */
TCF=1,TXRXS=0,EOQF=1,TFUF=1,TFFF=1,RFOF=1,RFDF=1,TXCTR=0,TXNXTPTR=0,RXCTR=0,POPNXTPTR=0 */

```

```

SPIO_SR = (uint32_t)0x9AOA0000UL;
/* SPIO_RSER:
TCF_RE=0,EOQF_RE=0,TFUF_RE=0,TFFF_RE=0,TFFF_DIRS=0,RFOF_RE=0,RFDF_RE=0,RFDF_DIRS=0 */
SPIO_RSER = (uint32_t)0x00UL;
/* SPIO_MCR: HALT=0 */
SPIO_MCR &= (uint32_t)~0x01UL;
/* ### Init_UART init code */
/* SIM_SCGC1: UART4=1 */
SIM_SCGC1 |= (uint32_t)0x0400UL;
/* UART4_C2: TE=0,RE=0 */
UART4_C2 &= (uint8_t)~(uint8_t)0x0CU;
/* UART4_BDL: SBR=0xA2 */
UART4_BDL = (uint8_t)0xA2U;
/* UART4_BDH: LBKDE=0,RXEDGIE=0,SBR=0 */
UART4_BDH = (uint8_t)0x00U;
/* UART4_MA1: MA=0 */
UART4_MA1 = (uint8_t)0x00U;
/* UART4_MA2: MA=0 */
UART4_MA2 = (uint8_t)0x00U;
/* UART4_C4: MAEN1=0,MAEN2=0,M10=0,BRFA=0x18 */
UART4_C4 = (uint8_t)0x18U;
/* UART4_C1: LOOPS=0,UARTSWAI=1,RSRC=0,M=0,WAKE=0,ILT=0,PE=0,PT=0 */
UART4_C1 = (uint8_t)0x40U;
/* UART4_S2: LBKDIF=1,RXEDGIF=1,MSBF=0,RXINV=0,RWUID=0,BRK13=0,LBKDE=0,RAF=0 */
UART4_S2 = (uint8_t)0xCOU;
/* UART4_MODEM: RXRTSE=0,TXRTSPOL=0,TXRTSE=0,TXCTSE=0 */
UART4_MODEM = (uint8_t)0x00U;
/* UART4_IR: IREN=0,TNP=0 */
UART4_IR = (uint8_t)0x00U;
/* UART4_TWFIFO: TXWATER=1 */
UART4_TWFIFO = (uint8_t)0x01U;
/* UART4_RWFIFO: RXWATER=1 */
UART4_RWFIFO = (uint8_t)0x01U;
/* UART4_SFIFO: TXEMPT=0,RXEMPT=0,TXOF=1,RXUF=1 */
UART4_SFIFO = (uint8_t)0x03U;
/* UART4_CFIFO: TXFLUSH=1,RXFLUSH=1,TXOFE=0,RXUFE=0 */
UART4_CFIFO = (uint8_t)0xCOU;
/* UART4_P FIFO: TXFE=1,RXFE=0 */
UART4_P FIFO = (uint8_t)((UART4_P FIFO & (uint8_t)~(uint8_t)0x08U) | (uint8_t)0x80U);
(void)UART4_S1;           /* Dummy read of the UART4_S1 register to clear flags */
(void)UART4_D;           /* Dummy read of the UART4_D register to clear flags */
/* UART4_C5: TDMAS=0,RDMAS=0 */
UART4_C5 = (uint8_t)0x00U;
/* UART4_C3: R8=0,T8=0,TXDIR=0,TXINV=0,ORIE=0,NEIE=0,FEIE=0,PEIE=0 */
UART4_C3 = (uint8_t)0x00U;
/* UART4_C2: TIE=0,TCIE=0,RIE=1,ILIE=0,TE=0,RE=0,RWU=0,SBK=0 */
UART4_C2 = (uint8_t)0x20U;
/* ### Init_VREF init code */
/* SIM_SCGC4: VREF=1 */
SIM_SCGC4 |= (uint32_t)0x00100000UL;
/* VREF_SC: VREFEN=1,REGEN=0,VREFST=0,MODE_LV=1 */
VREF_SC = (uint8_t)0x81U;
/* ### */
Cpu_SetBASEPRI(0U);
} /* MCU_init */

```

```

/*
** =====
**   Interrupt handler : isr_default
**
**   Description :
**     User interrupt service routine.
**   Parameters : None
**   Returns   : Nothing
** =====
*/
PE_ISR(isr_default)
{
    Diagnose(001);           //ERRO 001 = ISR DEFAULT (SOFTWARE FATAL ERROR)
}
/* end of isr_default */

/*
** =====
**   Interrupt handler : isrINT_NMI
**
**   Description :
**     User interrupt service routine.
**   Parameters : None
**   Returns   : Nothing
** =====
*/
PE_ISR(isrINT_NMI)
{
    Diagnose(002); //ERRO 002 = ISR INT_NMI (MEMORY INTERRUPT ERROR)
}
/* end of isrINT_NMI */

/*
** =====
**   Interrupt handler : isrINT_SPIO
**
**   Description :
**     User interrupt service routine.
**   Parameters : None
**   Returns   : Nothing
** =====
*/
PE_ISR(isrINT_SPIO)
{
    Diagnose(004);           // ERRO 004 = INTERRUPCAO INVALIDA
}
/* end of isrINT_SPIO */

/*
** =====
**   Interrupt handler : isrINT_UART4_RX_TX
**
**   Description :
**     User interrupt service routine.

```

```

** Parameters : None
** Returns   : Nothing
** =====
*/
PE_ISR(isrINT_UART4_RX_TX)
{
    //Interrupcao programada somente para ocorrer quando uma mensagem
    //valida ocorre no usb1 - Prioridade definida como 10

    //RecebeUSB1(0);           //Armazena dado recebido

    //Funcao deve limpar a flag de interrupcao
}
/* end of isrINT_UART4_RX_TX */

/*
** =====
** Interrupt handler : isrINT_UART4_ERR
**
** Description :
**     User interrupt service routine.
** Parameters : None
** Returns   : Nothing
** =====
*/
PE_ISR(isrINT_UART4_ERR)
{
    Diagnose(003); //ERRO 003 = UART4_ERR (ERRO NO USB1)
}
/* end of isrINT_UART4_ERR */

/*
** =====
** Interrupt handler : isrINT_ADC0
**
** Description :
**     User interrupt service routine.
** Parameters : None
** Returns   : Nothing
** =====
*/
PE_ISR(isrINT_ADC0)
{
    Diagnose(004);           // ERRO 004 = INTERRUPCAO INVALIDA

}
/* end of isrINT_ADC0 */

/*
** =====
** Interrupt handler : isrINT_ADC1
**
** Description :
**     User interrupt service routine.

```

```

** Parameters : None
** Returns   : Nothing
** =====
*/
PE_ISR(isrINT_ADC1)
{
    Diagnose(004);                                // ERRO 004 = INTERRUPCAO INVALIDA

}

/* end of isrINT_ADC1 */



/*
** =====
** Interrupt handler : isrINT_FTM0
**
** Description :
**     User interrupt service routine.
** Parameters : None
** Returns   : Nothing
** =====
*/
PE_ISR(isrINT_FTM0)
{
    Diagnose(004);                                // ERRO 004 = INTERRUPCAO INVALIDA
}
/* end of isrINT_FTM0 */



/*
** =====
** Interrupt handler : isrINT_FTM1
**
** Description :
**     User interrupt service routine.
** Parameters : None
** Returns   : Nothing
** =====
*/
PE_ISR(isrINT_FTM1)
{
    Diagnose(004);                                // ERRO 004 = INTERRUPCAO INVALIDA

}
/* end of isrINT_FTM1 */



/*
** =====
** Interrupt handler : isrINT_FTM2
**
** Description :
**     User interrupt service routine.
** Parameters : None
** Returns   : Nothing
** =====
*/

```

```

PE_ISR(isrINT_FTM2)
{
    //Vetor_FTM2(ON);           //Interrupcao que controla o acionamento da bobina
}
/* end of isrINT_FTM2 */

/*
** =====
** Interrupt handler : isrINT_PITO
**
** Description :
**     User interrupt service routine.
** Parameters : None
** Returns   : Nothing
** =====
*/
PE_ISR(isrINT_PITO)
{
    Vetor_PITO(0,0);           //Sincronismo do injetor 1
}
/* end of isrINT_PITO */

/*
** =====
** Interrupt handler : isrINT_PIT1
**
** Description :
**     User interrupt service routine.
** Parameters : None
** Returns   : Nothing
** =====
*/
PE_ISR(isrINT_PIT1)
{
    Vetor_PIT1(0,0);           //Sincronismo do injetor 2
}
/* end of isrINT_PIT1 */

/*
** =====
** Interrupt handler : isrINT_PIT2
**
** Description :
**     User interrupt service routine.
** Parameters : None
** Returns   : Nothing
** =====
*/
PE_ISR(isrINT_PIT2)
{
    Vetor_PIT2(0,1);           //Sincronismo da Bobina 1
}

```

```

        //Vetor_PIT2(0,ON);                                //Sincronismo da Bobina 1
    }
/* end of isrINT_PIT2 */

/*
** =====
**   Interrupt handler : isrINT_PIT3
**
**   Description :
**       User interrupt service routine.
**   Parameters : None
**   Returns   : Nothing
** =====
*/
PE_ISR(isrINT_PIT3)
{
    Vetor_PIT3(0,1);                                //Sincronismo da Bobina 2
    //Vetor_PIT3(0,ON);                            //Sincronismo da Bobina 2
}
/* end of isrINT_PIT3 */

/*
** =====
**   Interrupt handler : isrINT_PDB0
**
**   Description :
**       User interrupt service routine.
**   Parameters : None
**   Returns   : Nothing
** =====
*/
PE_ISR(isrINT_PDB0)
{
    Vetor_PDB0();    //Interrupcao periodica (100us) Valvula Borboleta e escalonador da main()

}
/* end of isrINT_PDB0 */

/*
** =====
**   Interrupt handler : isrINT_DAC0
**
**   Description :
**       User interrupt service routine.
**   Parameters : None
**   Returns   : Nothing
** =====
*/
PE_ISR(isrINT_DAC0)
{
    /* Write your interrupt code here ... */
}

```

```

/* end of isrINT_DAC0 */

/*
** =====
** Interrupt handler : isrINT_PORTA
**
** Description :
**   User interrupt service routine.
** Parameters : None
** Returns  : Nothing
** =====
*/
PE_ISR(isrINT_PORTA)
{
    switch (PORTA_ISFR){ //como é um registrador para todos os pinos, verifica em qual ocorreu

        case 0x1000000:
            //CHAVE IGNICAO PTA24
            Vetor_Linha15(); //A funcao deve limpar sua flag ao final

            break;
    }

}
/* end of isrINT_PORTA */

/*
** =====
** Interrupt handler : isrINT_PORTD
**
** Description :
**   User interrupt service routine.
** Parameters : None
** Returns  : Nothing
** =====
*/
PE_ISR(isrINT_PORTD)
{
    switch (PORTD_ISFR){ //como eh um registrador para todos os pinos,
                        //verifica se a interrupcao realmente eh no sensor Rotacao

        case 0x10: //Caso Interrupcao tenha ocorrido no pino PTD4 (0x10)
                    PORTD_PCR4 |= PORT_PCR_ISF_MASK;
                    //limpa flag PTD4
                    Vetor_SensorRotacao(1, 0);
                    //Sincronismo do Motor
                    break;

        case 0x20: //Caso Interrupcao tenha ocorrido no pino PTD5 (0x20)
                    PORTD_PCR5 |= PORT_PCR_ISF_MASK;
                    //limpa flag PTD5
                    init_engine(1);
                    //Partida no motor com sensor FASE
    }
}

```

```

        break;

    default:
        PORTD_PCR4 |= PORT_PCR_ISF_MASK;
        //limpa flag PTD4
        PORTD_PCR5 |= PORT_PCR_ISF_MASK;
        //limpa flag PTDS
        Vetor_SensorRotacao(1, 0);
        //Sincronismo do Motor
        init_engine(1);
        //Partida no motor com sensor FASE
        break;
    }
}

/* end of isrINT_PORTD */

#endif __cplusplus
extern "C" {
#endif
extern uint32_t __SP_INIT[];
extern void __thumb_startup( void );
#endif __cplusplus
}
#endif

/* Interrupt vector table */
#ifndef UNASSIGNED_ISR
#define UNASSIGNED_ISR isr_default /* unassigned interrupt service routine */
#endif

/* Pragma to place the interrupt vector table on correct location location defined in linker file.*/
#pragma define_section vectortable ".vectortable" ".vectortable" ".vectortable" far_abs R
static __declspec(vectortable) tVectorTable __vect_table = { /* Interrupt vector table */
    __SP_INIT, /* 0 (0x00000000) (prior: -) */
{
    (tIsrFunc)&__thumb_startup,
    /* 1 (0x00000004) (prior: -) */
    (tIsrFunc)&isrINT_NMI,
    /* 2 (0x00000008) (prior: -2) */
    /* 3 (0x0000000C) (prior: -1) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 4 (0x00000010) (prior: -) */
    /* 5 (0x00000014) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 6 (0x00000018) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 7 (0x0000001C) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 8 (0x00000020) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 9 (0x00000024) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 10 (0x00000028) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 11 (0x0000002C) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 12 (0x00000030) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 13 (0x00000034) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 14 (0x00000038) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 15 (0x0000003C) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 16 (0x00000040) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 17 (0x00000044) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 18 (0x00000048) (prior: -) */
    (tIsrFunc)&UNASSIGNED_ISR,
    /* 19 (0x0000004C) (prior: -) */
}

```

```

(tlsrFunc)&UNASSIGNED_ISR,          /* 20 (0x00000050) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 21 (0x00000054) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 22 (0x00000058) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 23 (0x0000005C) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 24 (0x00000060) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 25 (0x00000064) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 26 (0x00000068) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 27 (0x0000006C) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 28 (0x00000070) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 29 (0x00000074) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 30 (0x00000078) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 31 (0x0000007C) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 32 (0x00000080) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 33 (0x00000084) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 34 (0x00000088) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 35 (0x0000008C) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 36 (0x00000090) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 37 (0x00000094) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 38 (0x00000098) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 39 (0x0000009C) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 40 (0x000000A0) (prior: -) */
(tlsrFunc)&UNASSIGNED_ISR,          /* 41 (0x000000A4) (prior: -) */
/* 42 (0x000000A8) (prior: 0) */
/* 43 (0x000000AC) (prior: -) */
/* 44 (0x000000B0) (prior: -) */
/* 45 (0x000000B4) (prior: -) */
/* 46 (0x000000B8) (prior: -) */
/* 47 (0x000000BC) (prior: -) */
/* 48 (0x000000C0) (prior: -) */
/* 49 (0x000000C4) (prior: -) */
/* 50 (0x000000C8) (prior: -) */
/* 51 (0x000000CC) (prior: -) */
/* 52 (0x000000D0) (prior: -) */
/* 53 (0x000000D4) (prior: -) */
/* 54 (0x000000D8) (prior: -) */
/* 55 (0x000000DC) (prior: -) */
/* 56 (0x000000E0) (prior: -) */
/* 57 (0x000000E4) (prior: -) */
/* 58 (0x000000E8) (prior: -) */
/* 59 (0x000000EC) (prior: -) */
/* 60 (0x000000F0) (prior: -) */
/* 61 (0x000000F4) (prior: -) */
/* 62 (0x000000F8) (prior: -) */
/* 63 (0x000000FC) (prior: -) */
/* 64 (0x00000100) (prior: -) */
/* 65 (0x00000104) (prior: -) */
/* 66 (0x00000108) (prior: -) */
/* 67 (0x0000010C) (prior: -) */
/* 68 (0x00000110) (prior: -) */
/* 69 (0x00000114) (prior: 10) */
/* 70 (0x00000118) (prior: 15) */
/* 71 (0x0000011C) (prior: -) */
/* 72 (0x00000120) (prior: -) */
/* 73 (0x00000124) (prior: 0) */
/* 74 (0x00000128) (prior: 0) */
/* 75 (0x0000012C) (prior: -) */
/* 76 (0x00000130) (prior: -) */

```



```

0xFFU,
/* NV_BACKKEY0: KEY=0xFF */
0xFFU,
/* NV_BACKKEY7: KEY=0xFF */
0xFFU,
/* NV_BACKKEY6: KEY=0xFF */
0xFFU,
/* NV_BACKKEY5: KEY=0xFF */
0xFFU,
/* NV_BACKKEY4: KEY=0xFF */
0xFFU,
/* NV_FPROT3: PROT=0xFF */
0xFFU,
/* NV_FPROT2: PROT=0xFF */
0xFFU,
/* NV_FPROT1: PROT=0xFF */
0xFFU,
/* NV_FPROT0: PROT=0xFF */
0xFFU,
/* NV_FSEC: KEYEN=1,MEEN=3,FSLACC=3,SEC=2 */
0x7EU,
/* NV_FOPT: EZPORT_DIS=1,LPBOOT=1 */
0xFFU,
/* NV_FEPROT: EPROT=0xFF */
0xFFU,
/* NV_FDPROT: DPROT=0xFF */
0xFFU
};

/*
** =====
**   Interrupt handler : isrINT_PORTB
**
**   Description :
**       User interrupt service routine.
**   Parameters : None
**   Returns   : Nothing
** =====
*/
PE_ISR(isrINT_PORTB)
{
    // Tarefa nao implementada
}

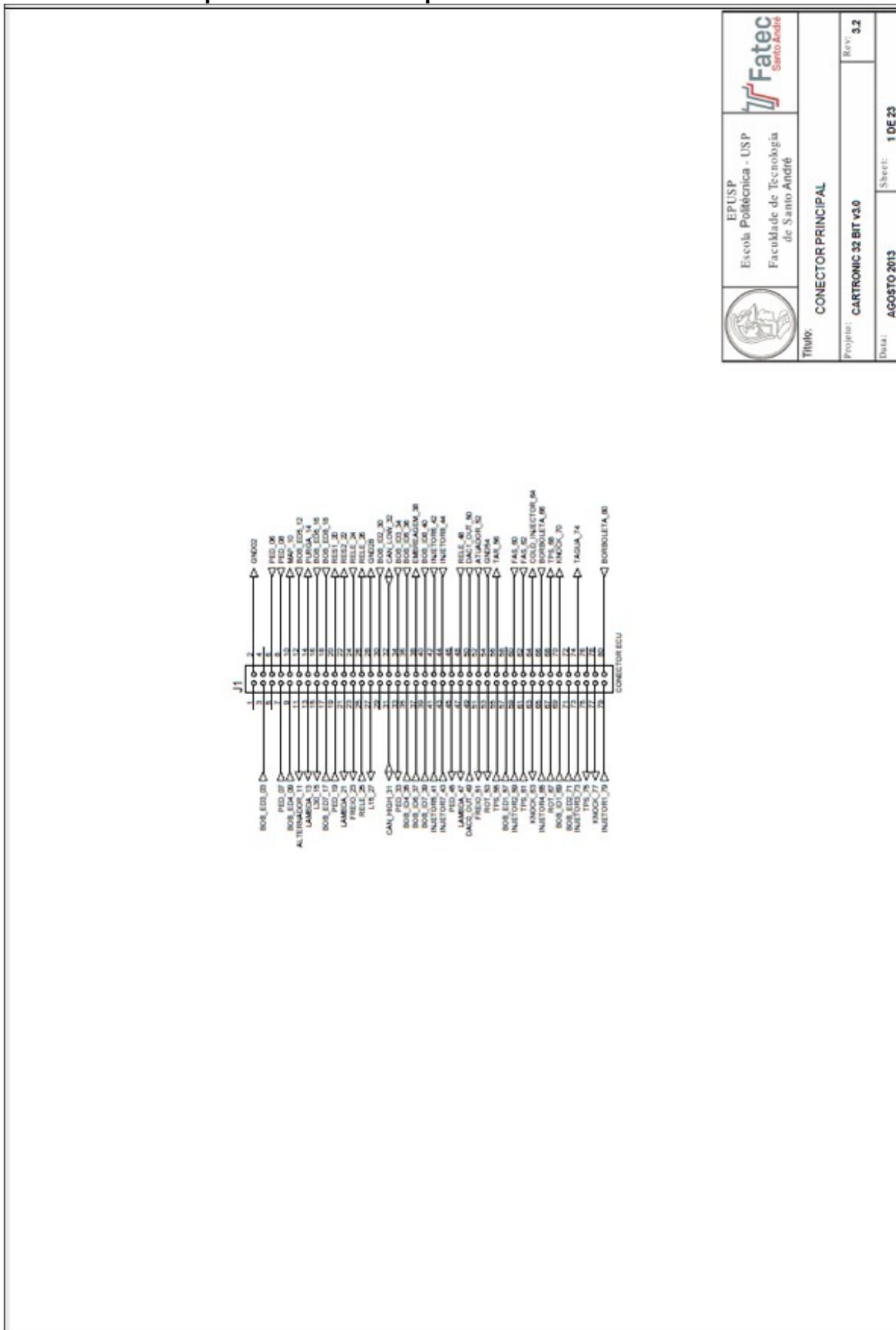
/* end of isrINT_PORTB */

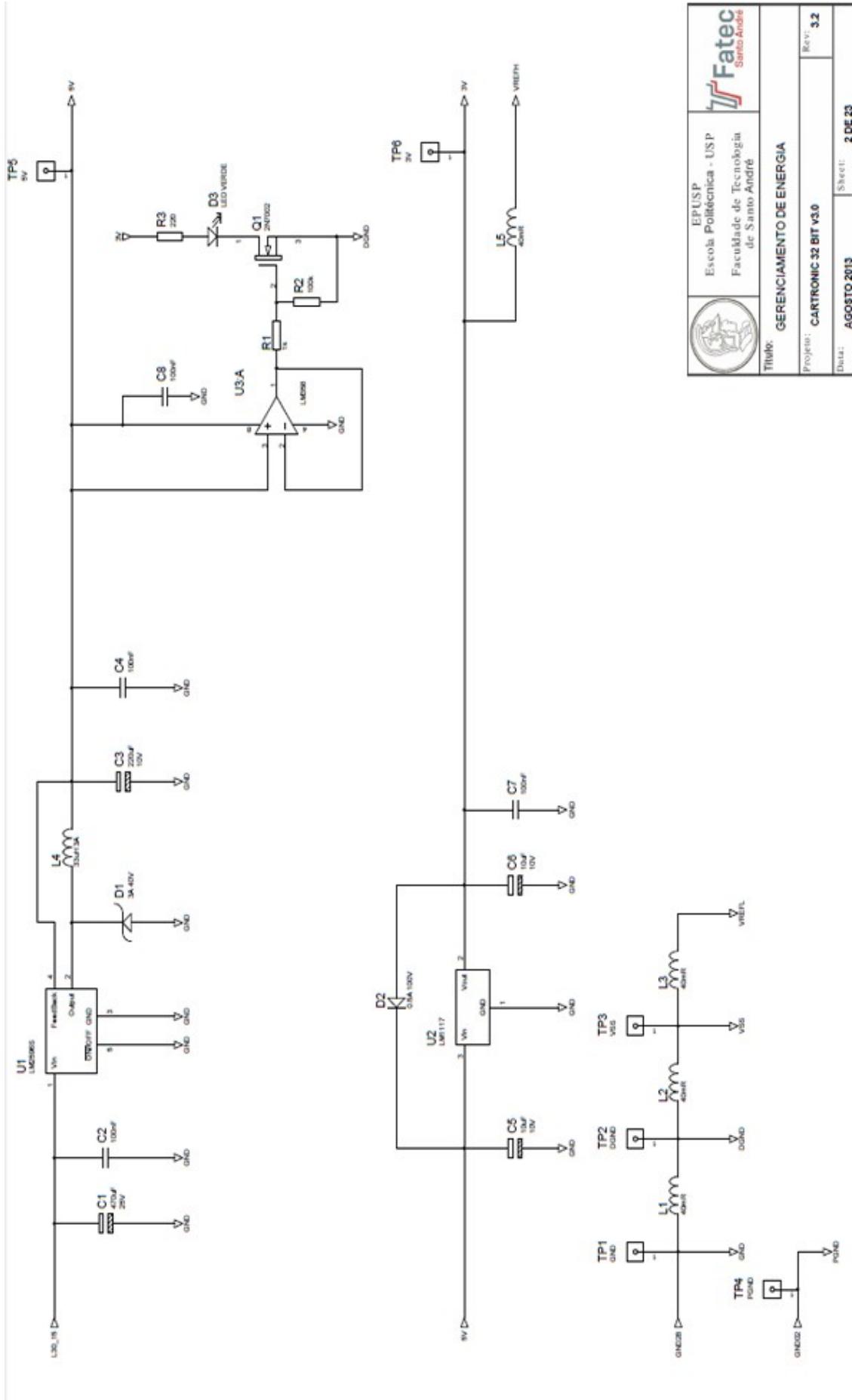
/* END MCUinit */

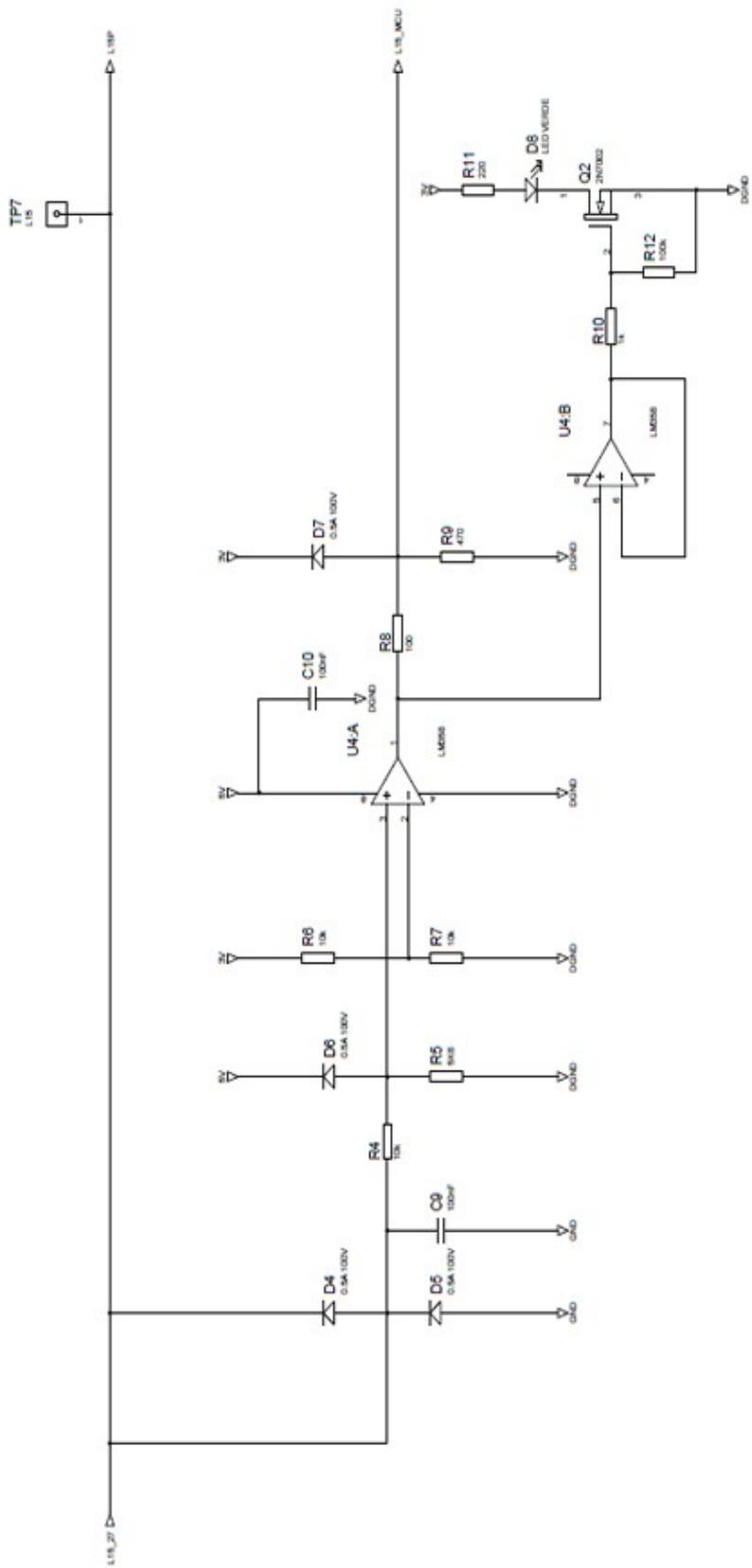
/*
** #####
** This file was created by Processor Expert 5.3 [05.01]
** for the Freescale Kinetis series of microcontrollers.
**
** #####

```

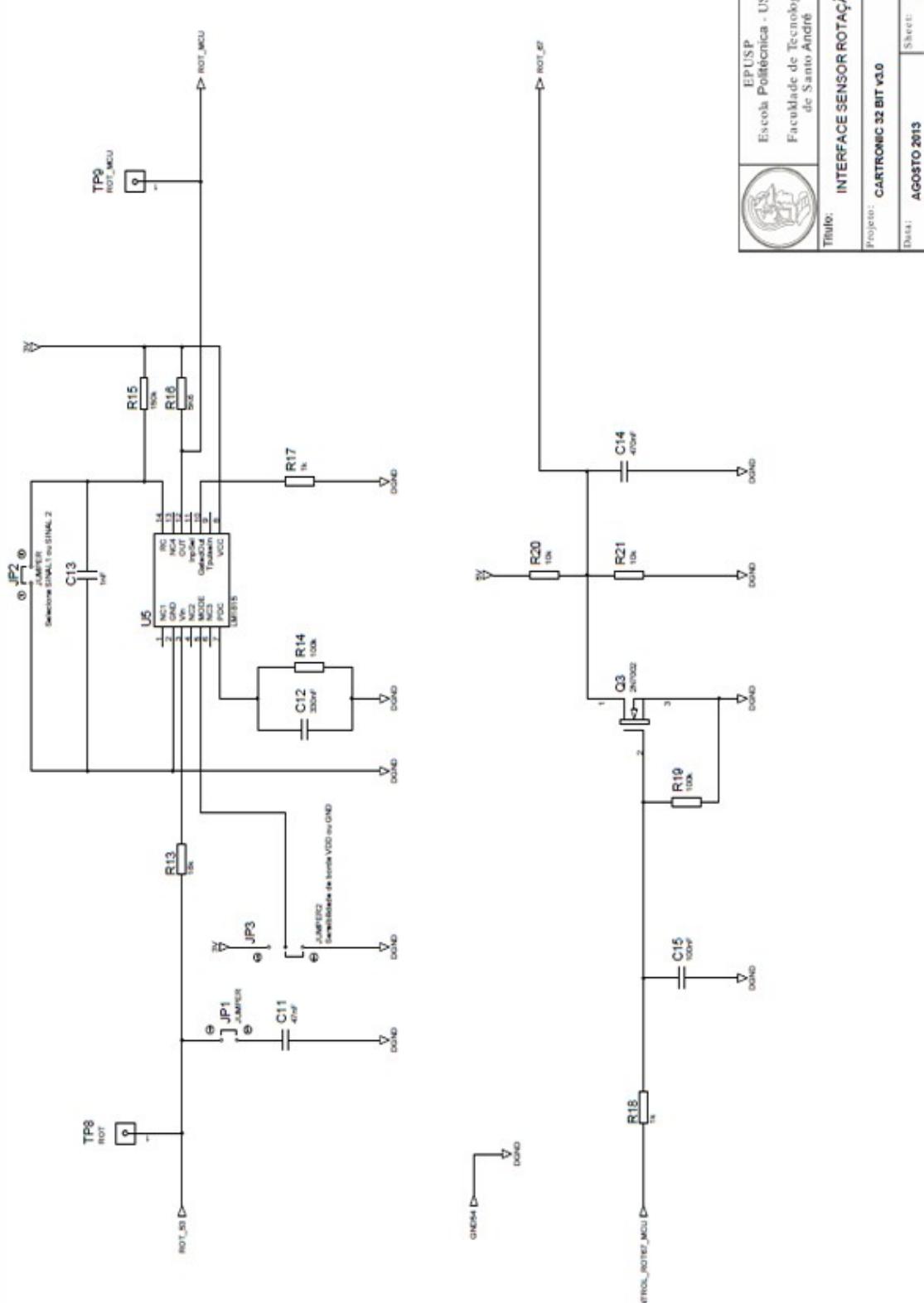
APÊNDICE B - Esquema Elétrico Completo do Unidade de Controle do Motor



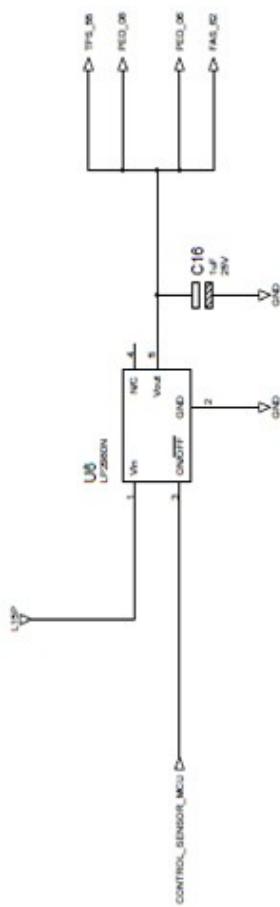


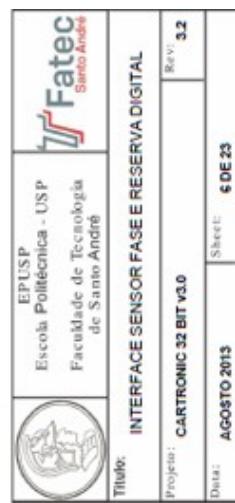
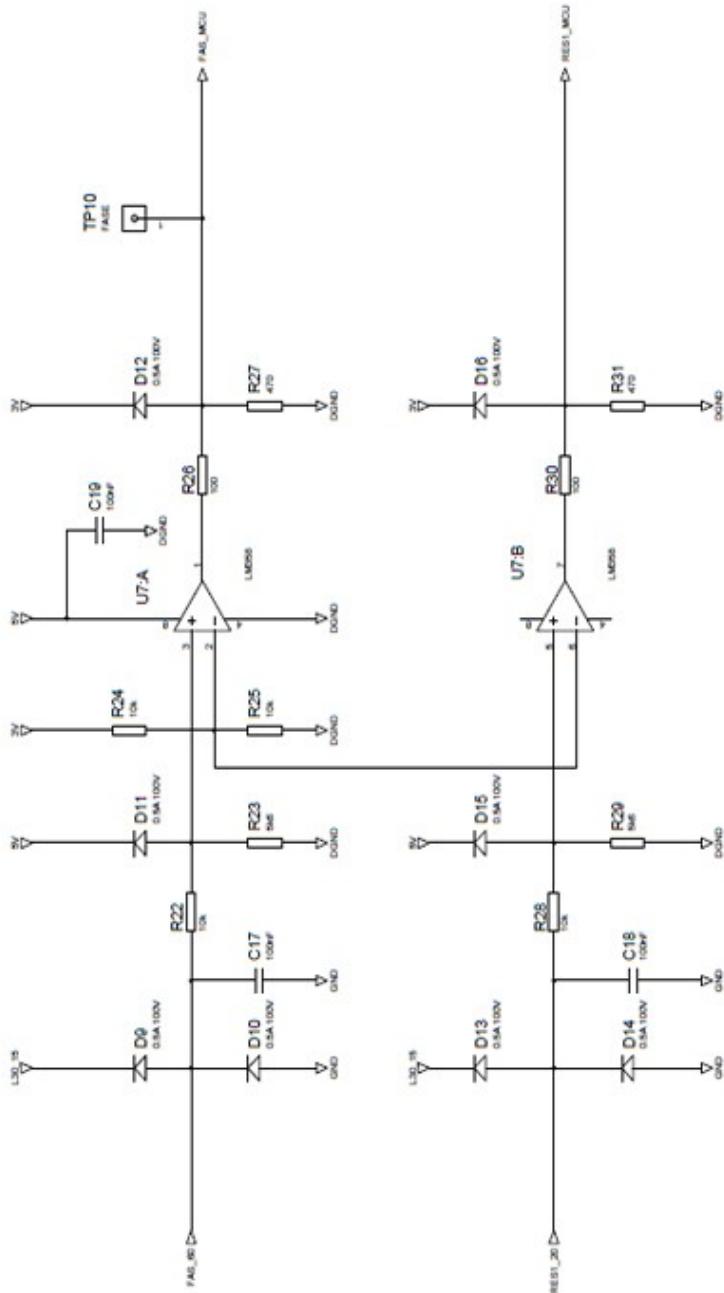


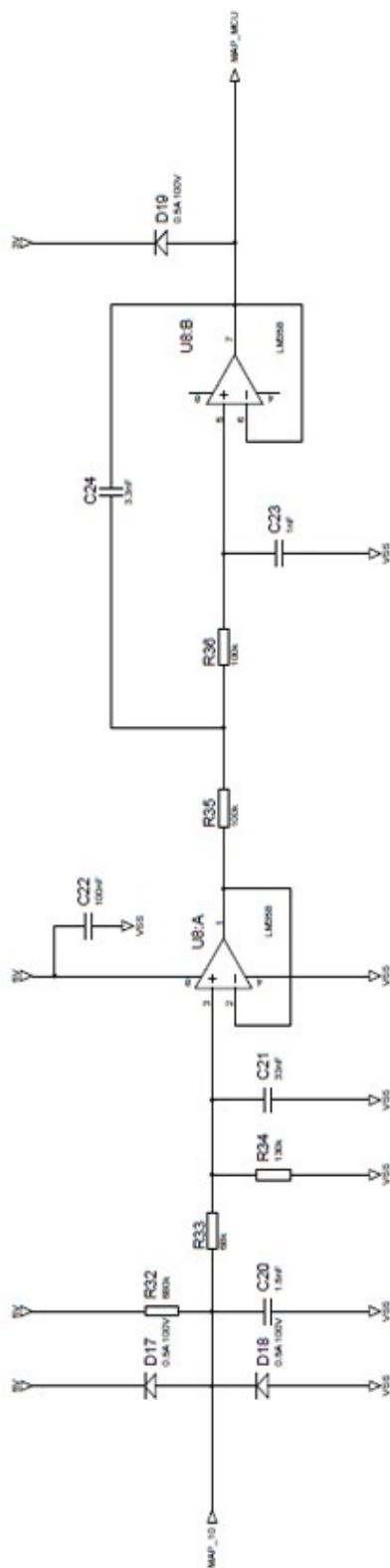
	EPUSP Escola Politécnica - USP Faculdade de Tecnologia de Santo André	
Título: GERENCIAMENTO LINHA 15		
Projeto: CARTRONIC 32 BIT v3.0	Rev: 3.2	
Data: AGOSTO 2013	Serviç: 3 DE 23	



	EPUSP Faculdade de Tecnologia de Santo André	Fatec Santo André
Projeto: INTERFACE SENSOR ROTAÇÃO		
Data: AGOSTO 2013	Série: 4 DE 23	Rev: 3.2

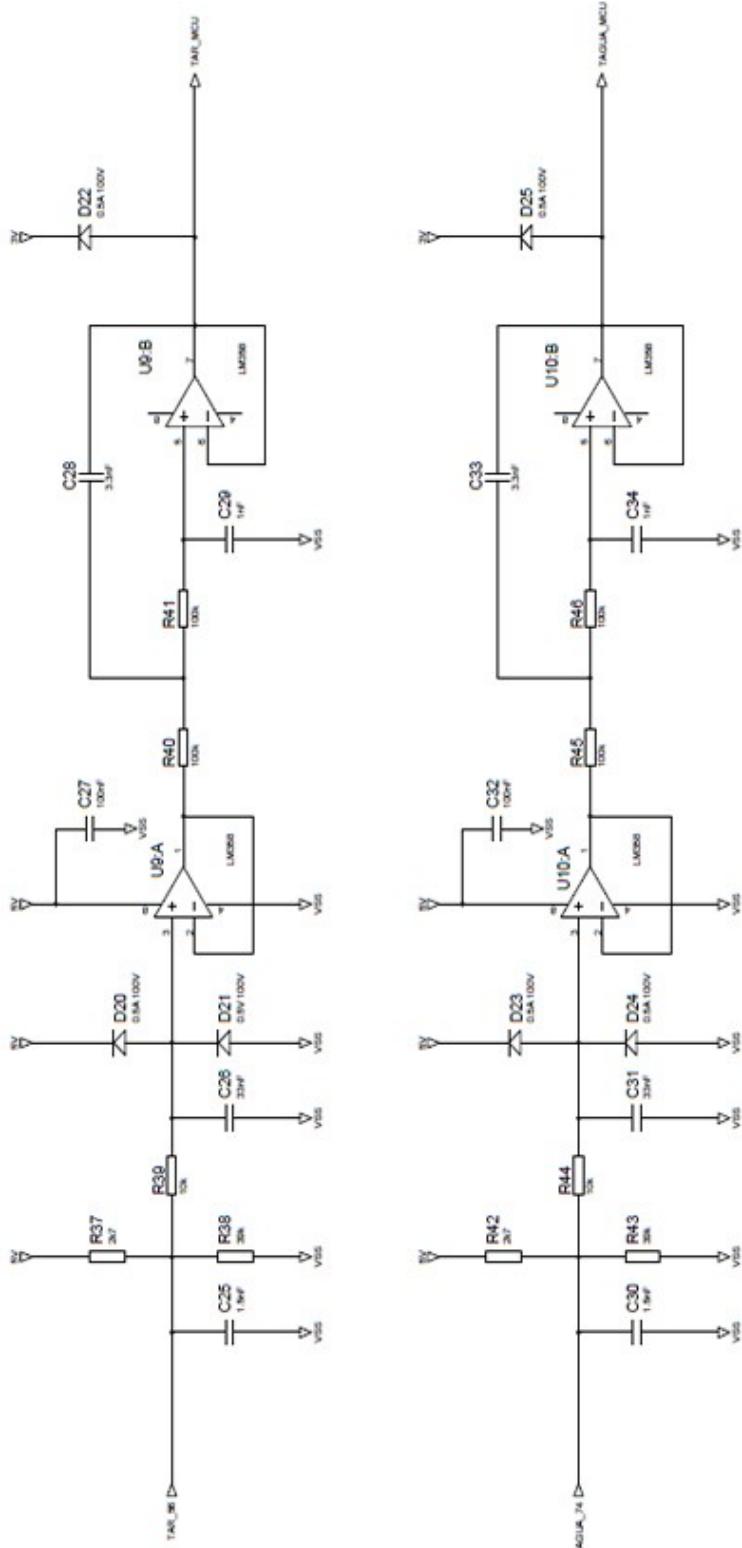


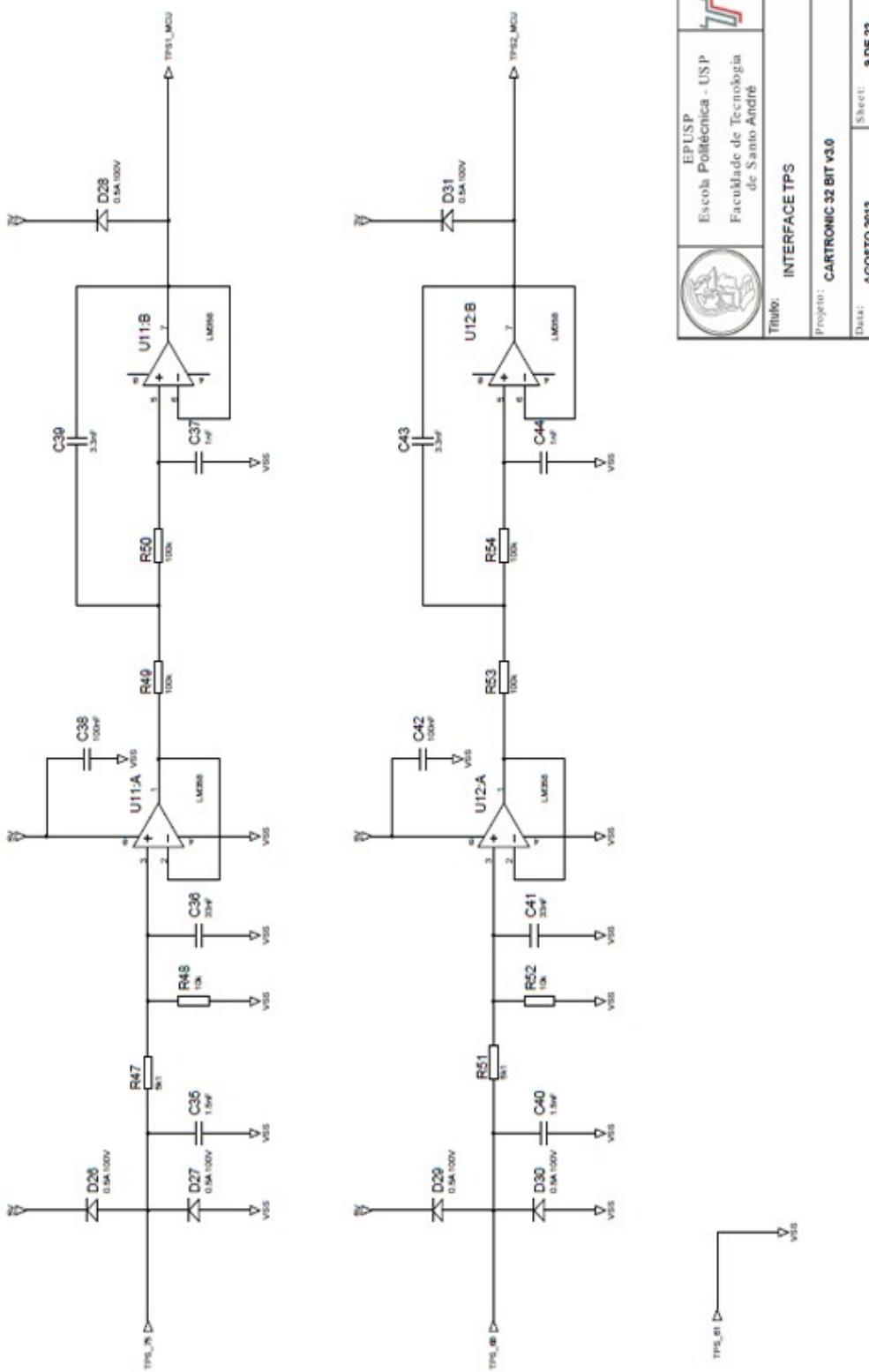


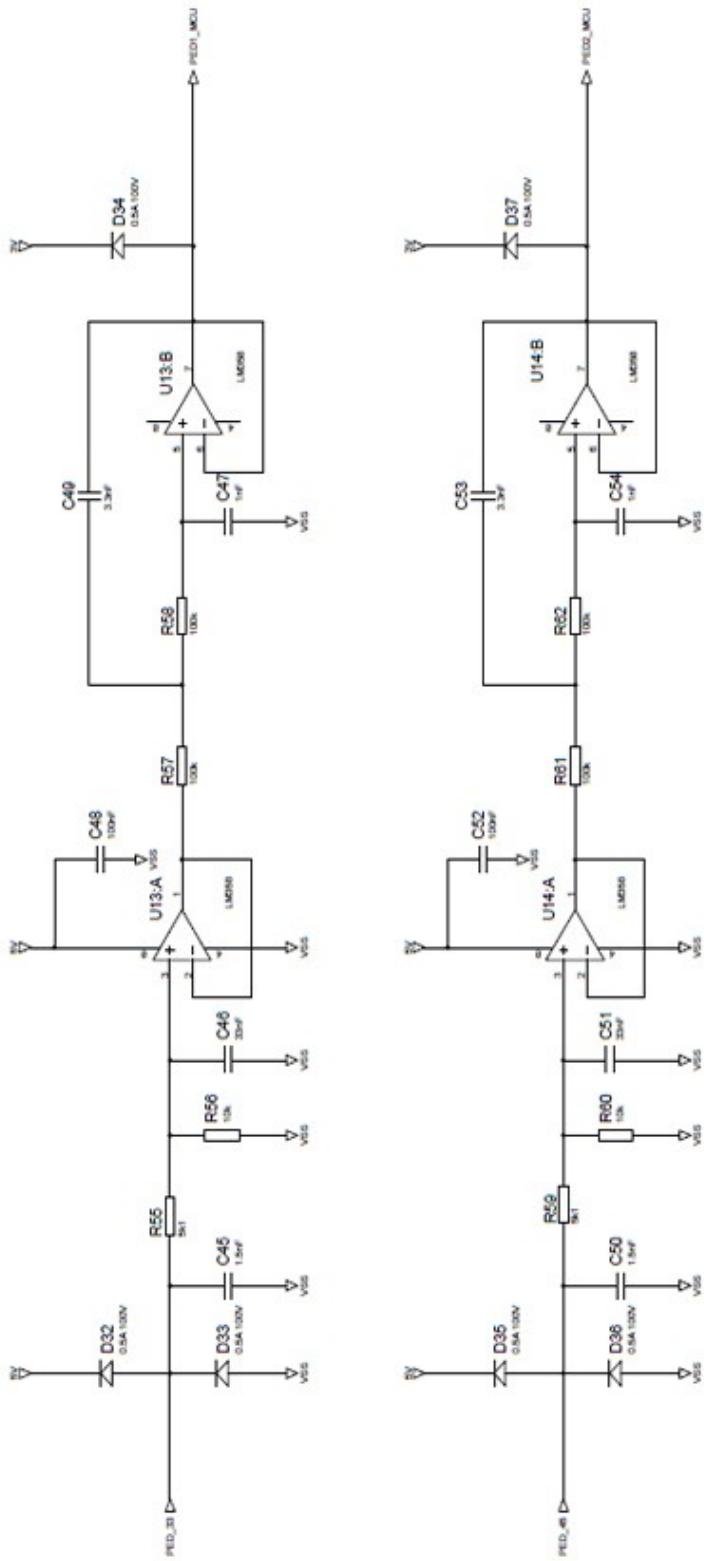


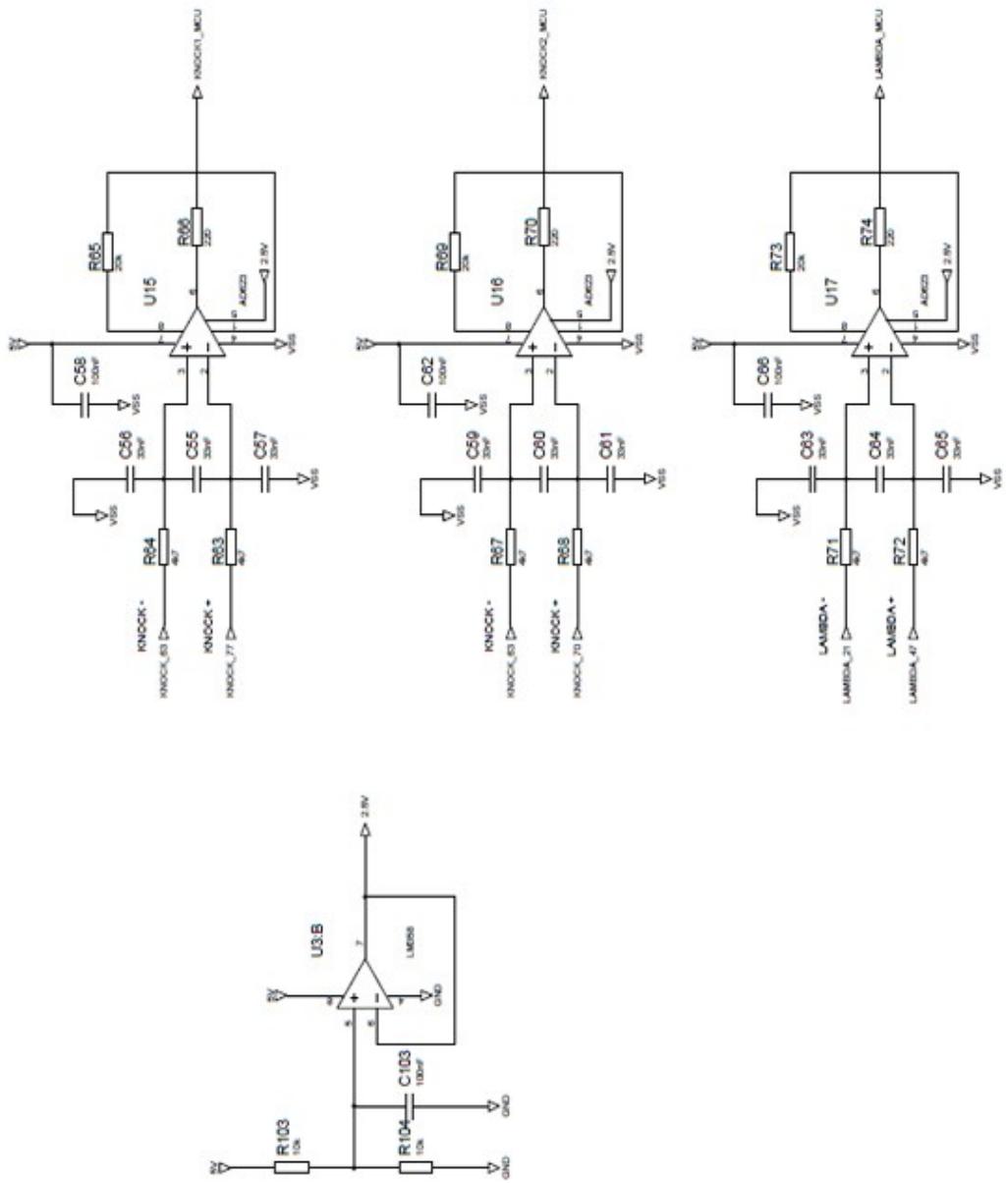
	EPUISP Escola Politécnica - USP Faculdade de Tecnologia de Santo André	
Projeto:	CARTRONIC 32 BIT v3.0	Rev.: 3.2
Data:	AGOSTO 2013	Sheet: 7 DE 23

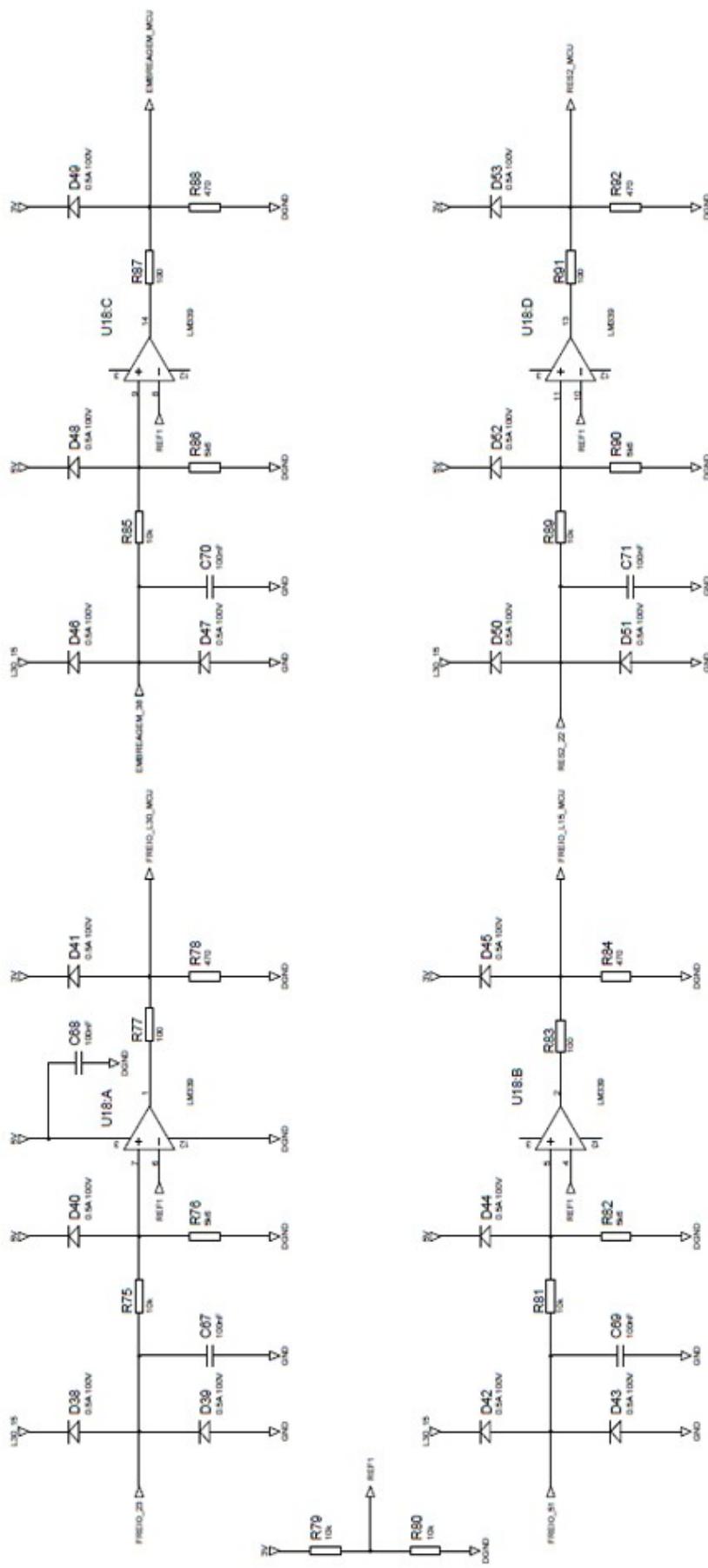
INTERFACE SENSOR DE PRESSÃO

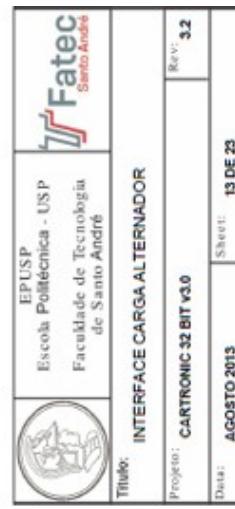
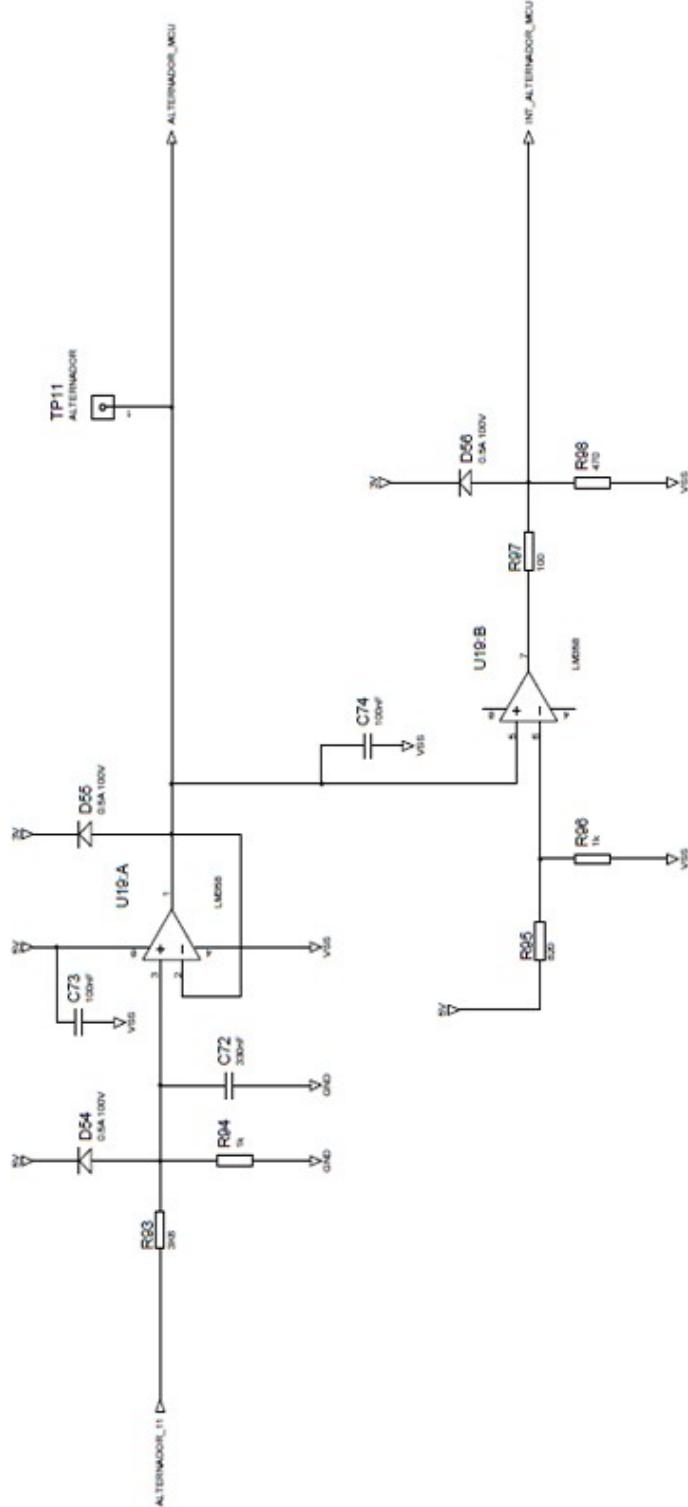


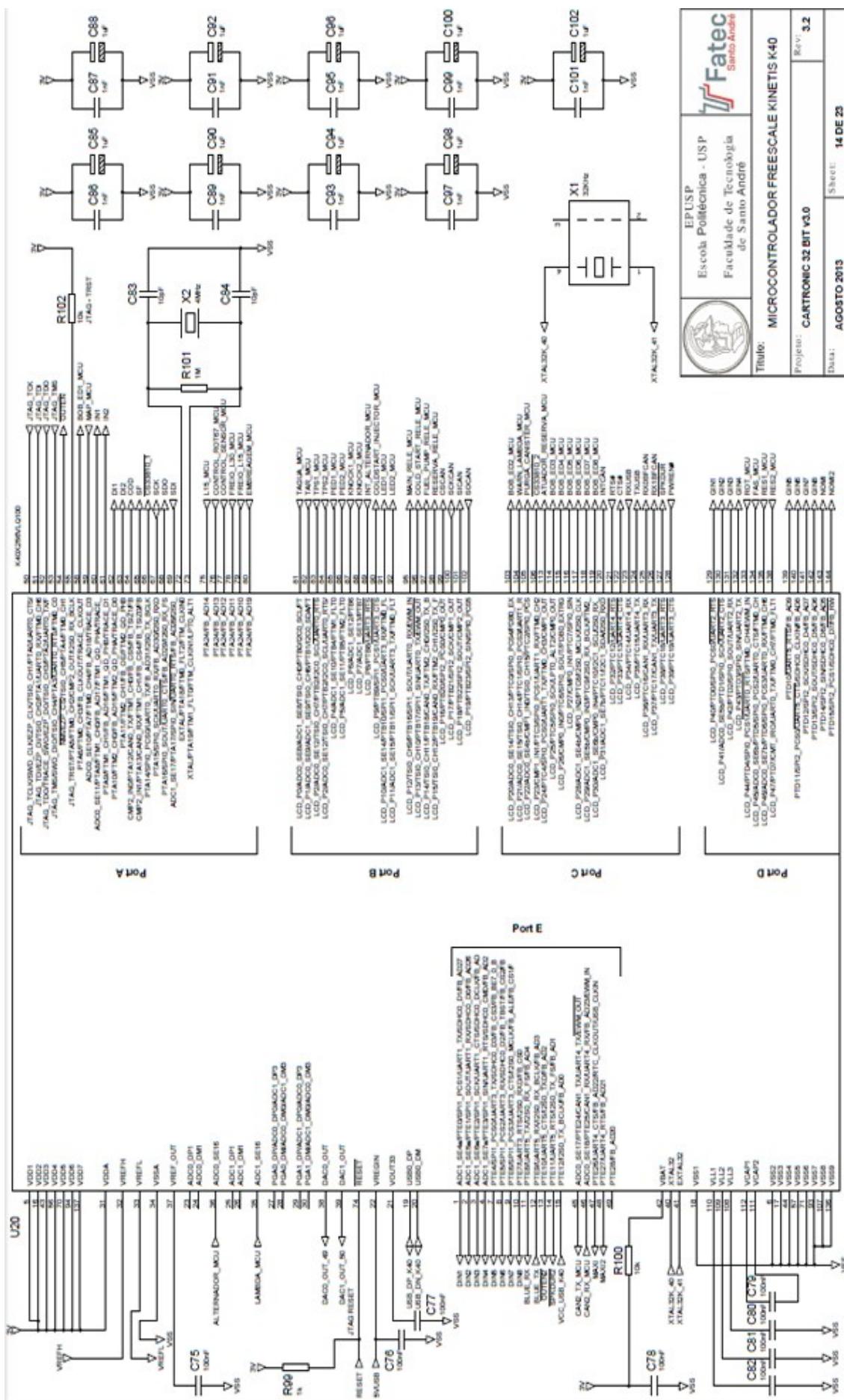


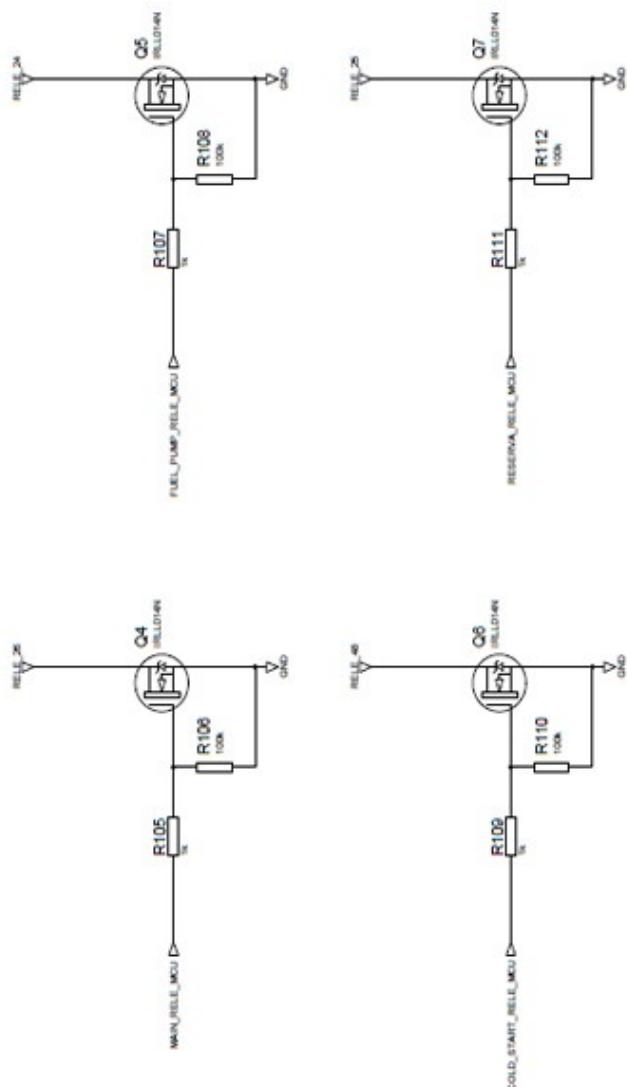


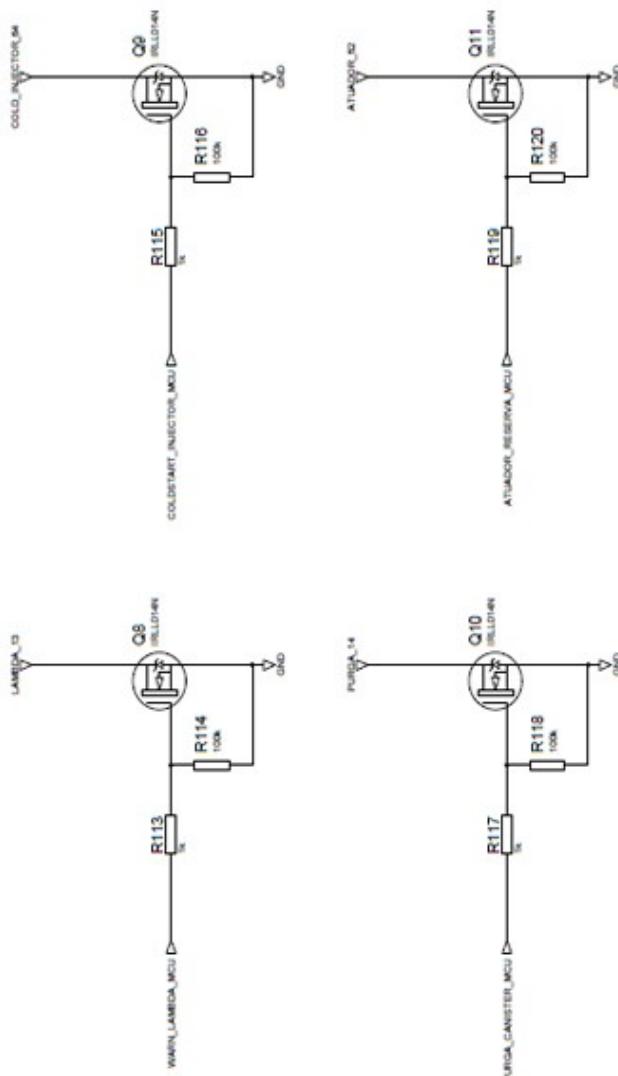


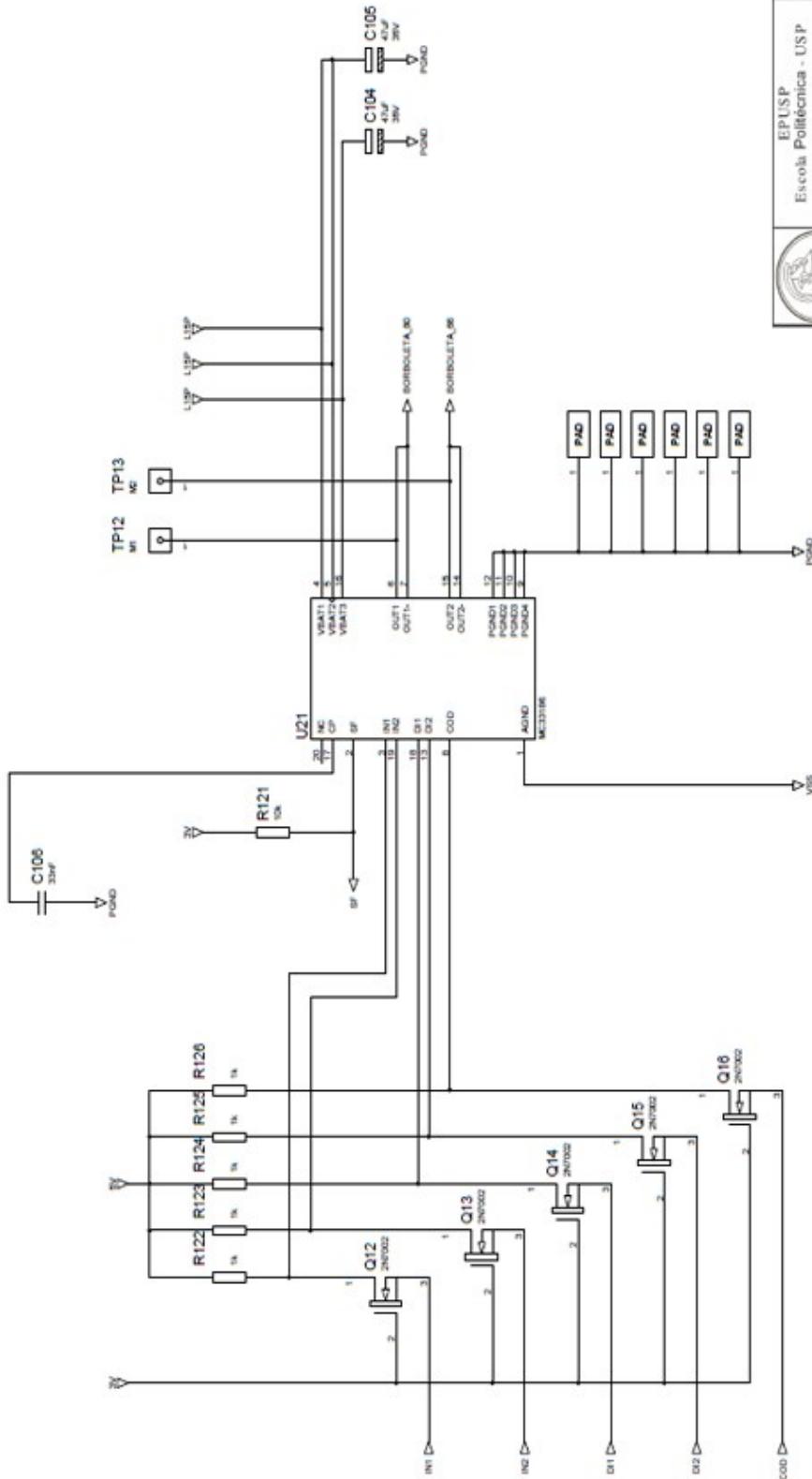










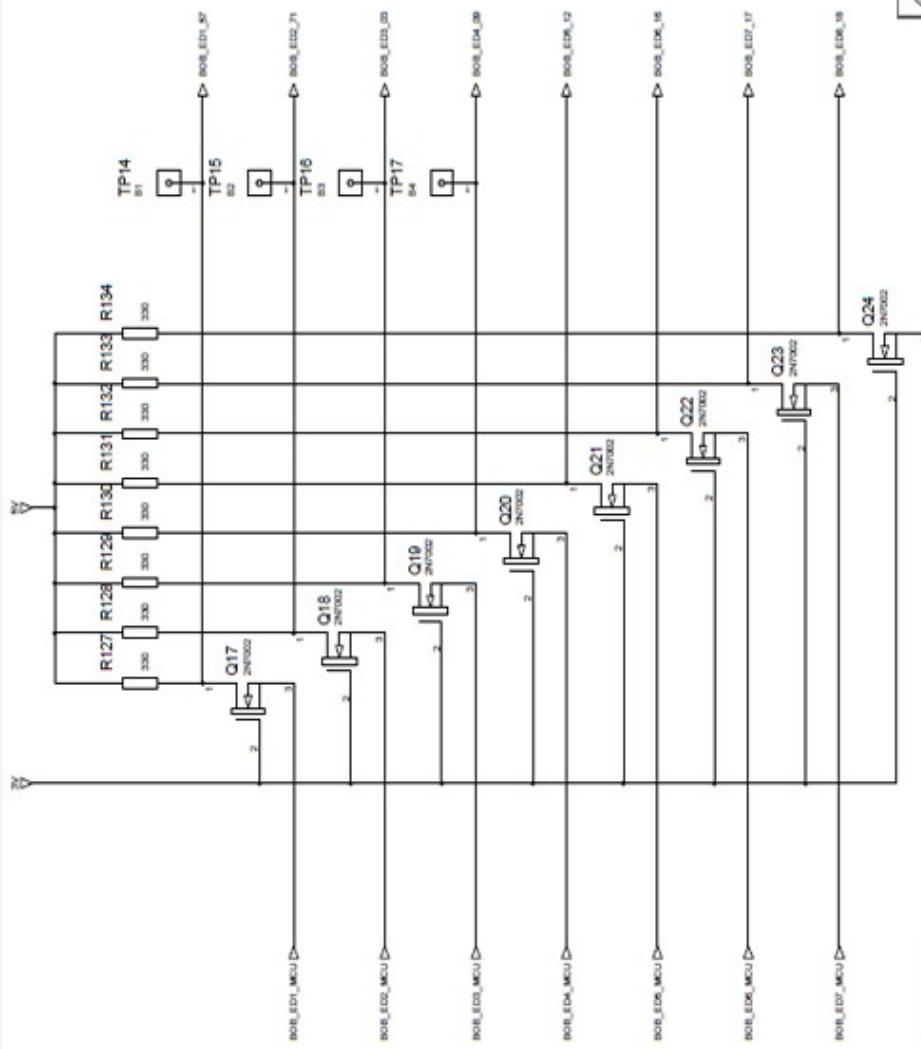


	EPTUSP Faculdade de Tecnologia de Santo André
Fatec Santo André	

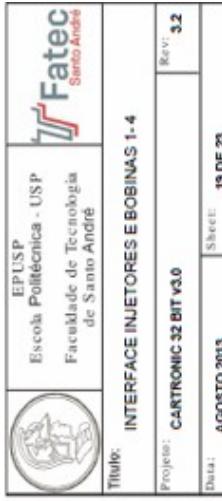
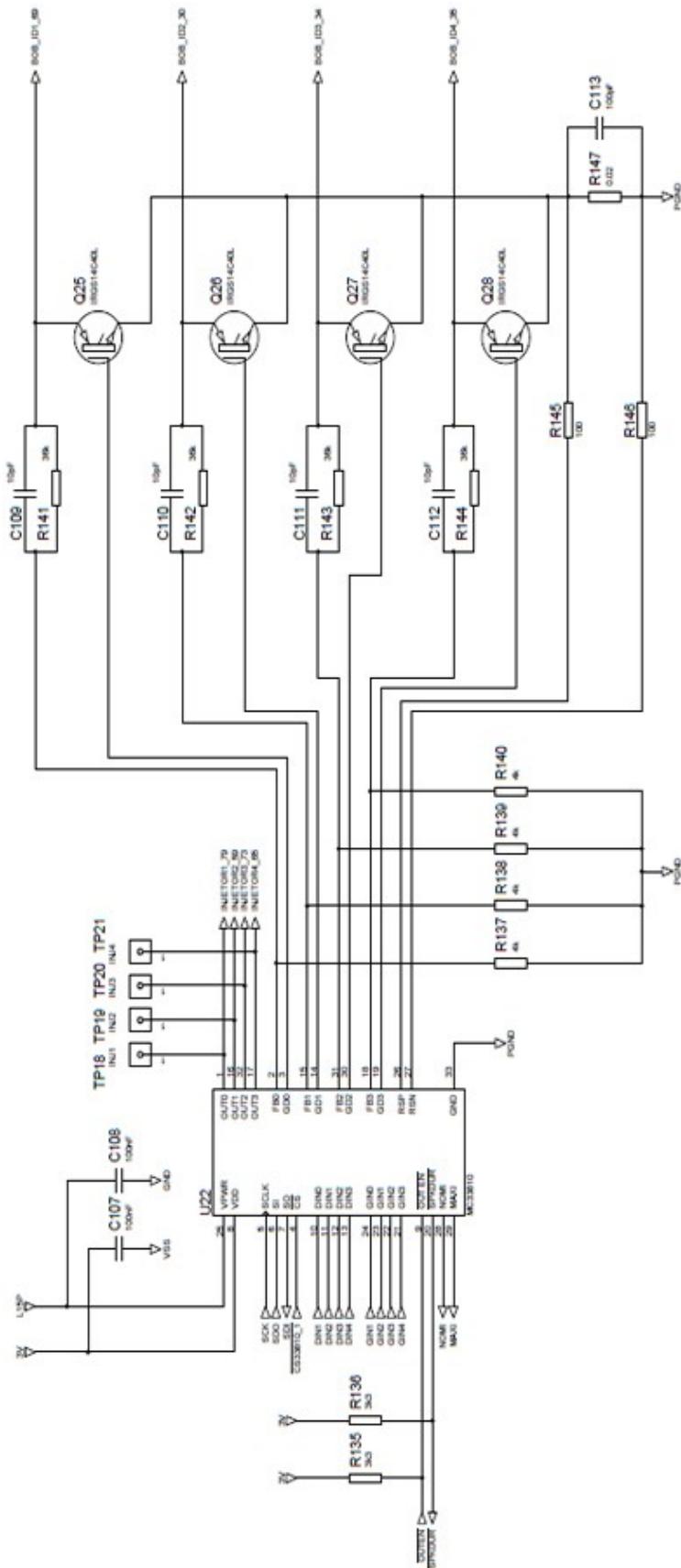
Title: INTERFACE VÁLVULA BORBOLETA

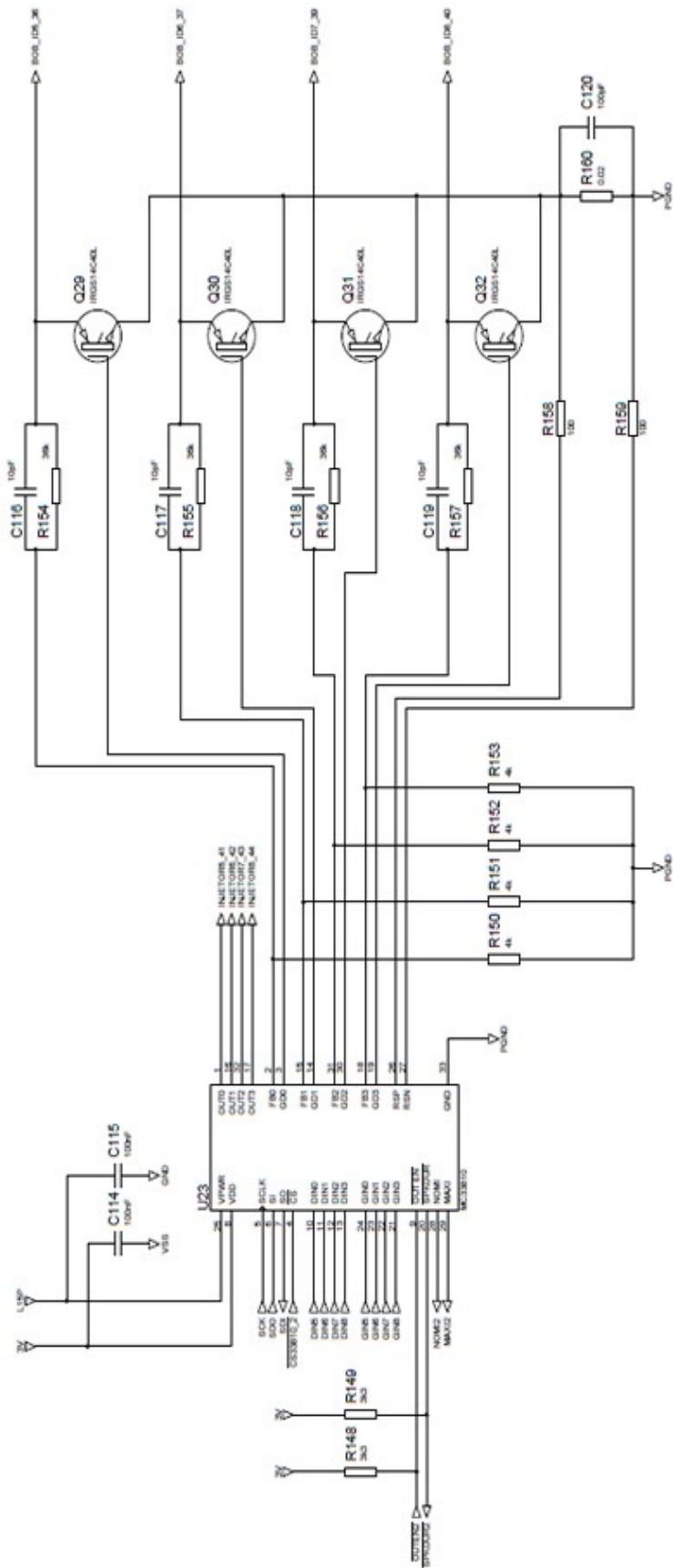
Project: CARTRONIC 32 BIT v3.0

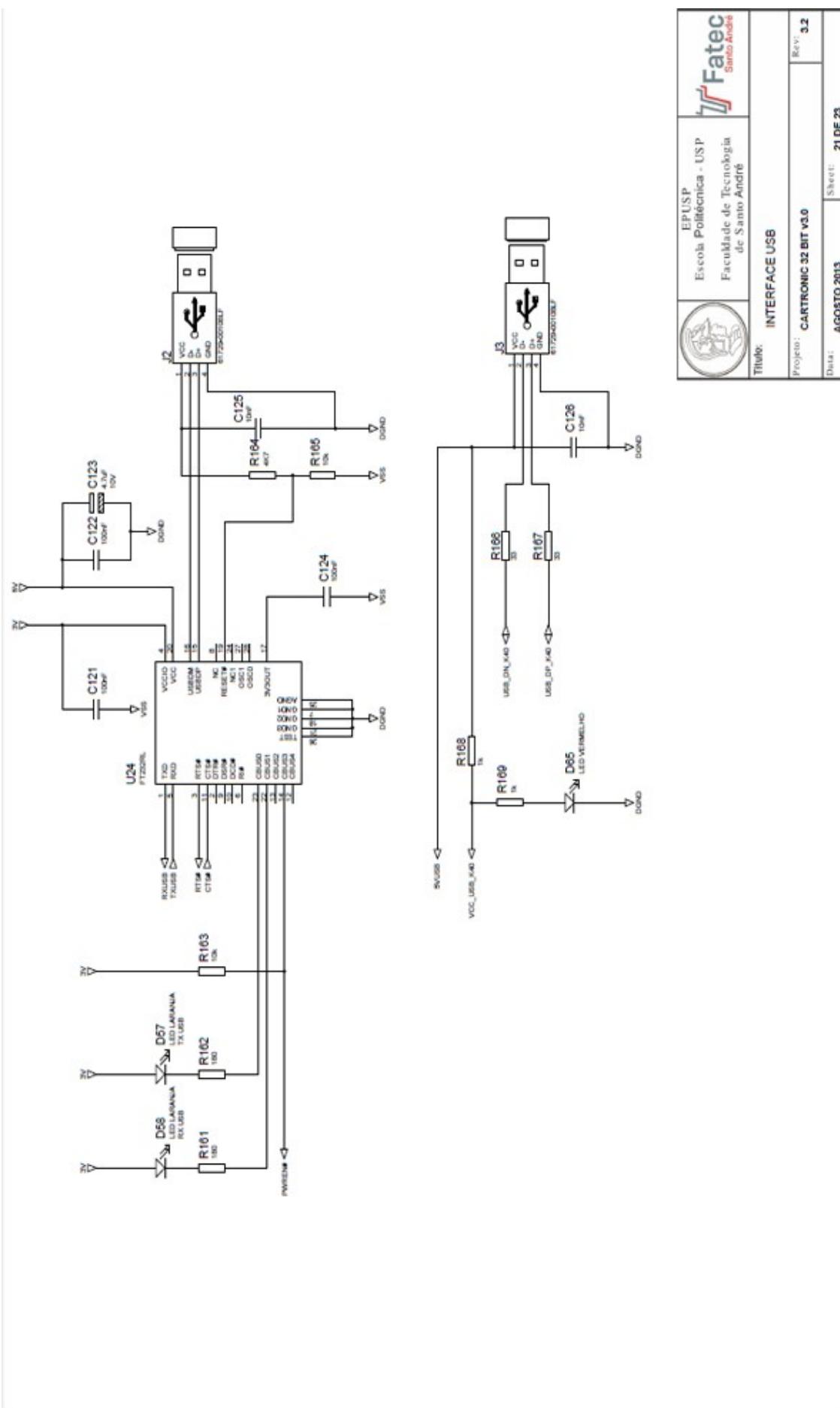
Data: AGOSTO 2013	Sheet: 17 DE 23
Rev.: 3.2	

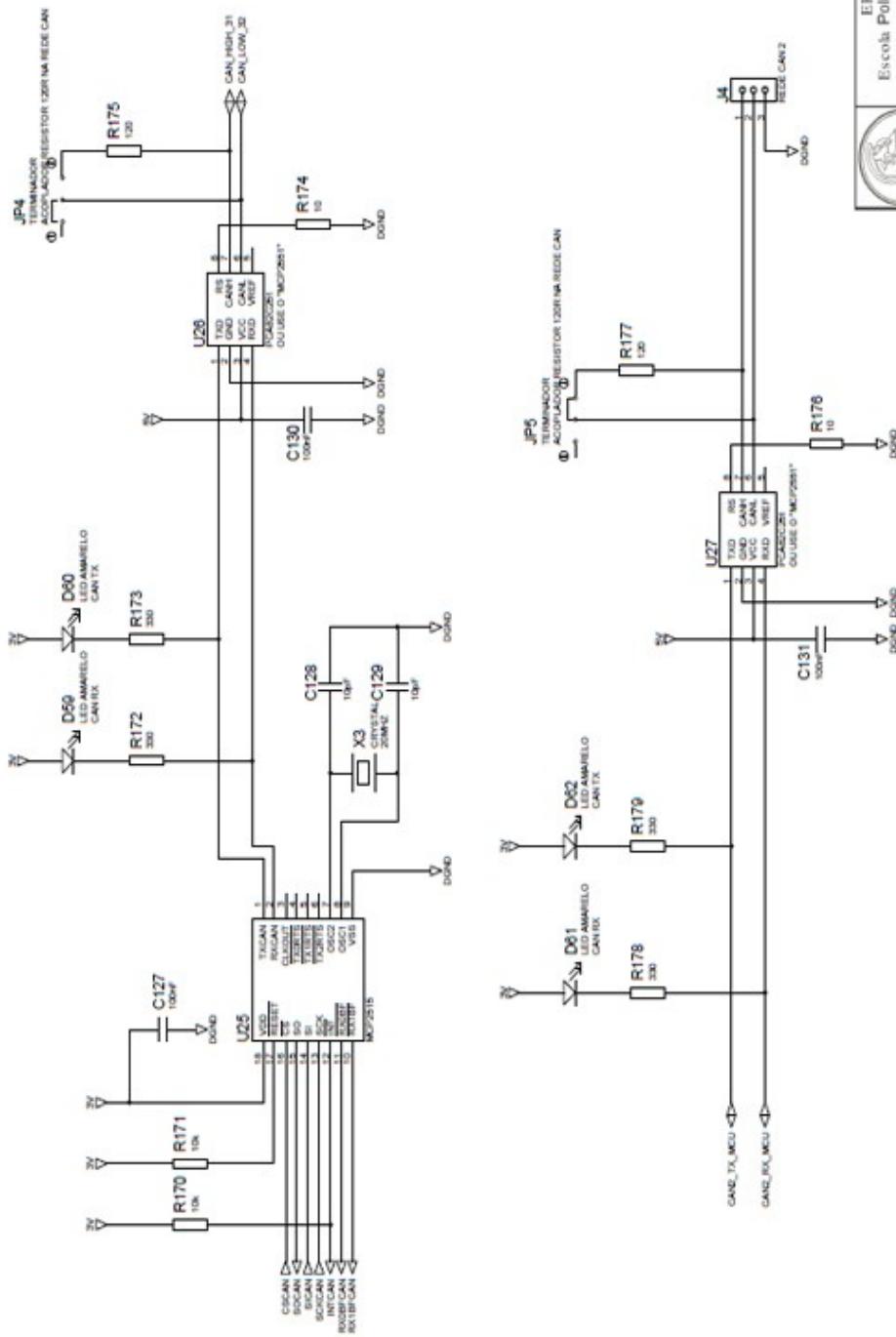


	Fatec Escola Politécnica - USP Faculdade de Tecnologia de Santo André
Projeto: INTERFACE BOBINA IGNÍCIAO (ED)	Rev.: 3.2
Data: AGOSTO 2013	Série: 18 DE 29

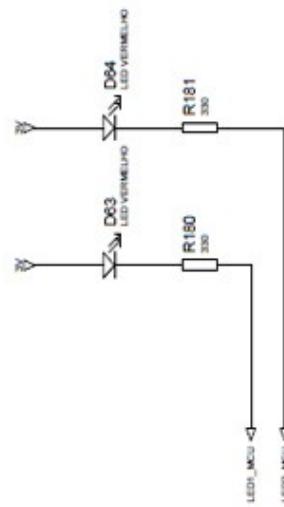
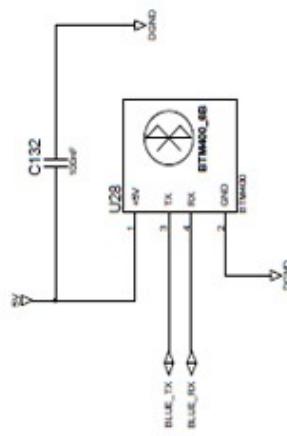
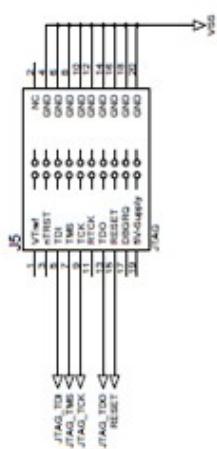








	EPUSP Faculdade de Tecnologia de São André	
Projeto: CANTRONIC 32 BIT v3.0	Série: 22 DE 23	Rev. 3.2
Data: AGOSTO 2013		



	Fatec Santo André
EPUSP Faculdade de Tecnologia de São André	
Projeto: CARTRONIC 32 BIT v3.0	Rev.: 3.2
Data: AGOSTO 2013	Série: 23 DE 23