BRUNO CÉSAR FERNANDES PEREIRA

Unidade de Gerenciamento Eletrônico de um Motor Volkswagen 2.0L: Projeto Otto II

São Paulo
2013

BRUNO CÉSAR FERNANDES PEREIRA

Unidade de Gerenciamento Eletrônico de um Motor Volkswagen 2.0L: Projeto Otto II

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para a Conclusão do Curso de Engenharia.

Área de Concentração:
Engenharia Elétrica - Sistemas Eletrônicos

Orientador: Prof. Dr. Armando Antonio Maria Laganá

São Paulo
2013

FICHA CATALOGRÁFICA

Pereira, Bruno César Fernandes
Unidade de gerenciamento eletrônico de um motor Volkswagem 2.0L: projeto Otto II / B.C.F. Pereira. -- São Paulo, 2013.
186 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Sistemas veiculares 2.Injeção (Engenharia) 3.Eletrônica 4.Indústria automobilística I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

Agradecimentos

Primeiramente um agradecimento em especial aos meus pais, Maria da Conceição Fernandes Pereira e Nerci Florêncio Pereira, por toda dedicação, amor e apoio dados durante uma vida inteira. São verdadeiros exemplos de vida. Espero um dia retribuir tudo o que fizeram por mim.

A minha namorada, Josiane Luz Pires, por me apoiar durante toda a fase de elaboração deste trabalho, sempre com muito amor e carinho. Obrigado por ser essa pessoa tão amorosa e incrível.

Ao querido professor doutor Armando Antonio Maria Laganá, orientador e idealizador deste projeto, por uma dedicação esplêndida que certamente foi muito além das obrigações de um orientador acadêmico. O sucesso deste projeto está diretamente atrelado ao seu imenso apoio, que despejou confiança ao crer no potencial de um aluno, ao mesmo tempo que buscou sempre interagir e evoluir conjuntamente a cada etapa do trabalho. Certamente será sempre um grande exemplo de pessoa a ser seguido.

Aos grandes amigos da FATEC Santo André, Demerson Moscardini e Bruno Silva Pereira, por todo o apoio e dedicação prestados durante a implementação prática deste projeto, contribuindo com o vasto conhecimento de cada um na área automotiva. Certamente este projeto não teria alcançado o sucesso se não fosse o esforço de ambos, estando presente em todos os testes, realizados em sua maior parte nos finais de semana. Foi um prazer imenso trabalhar em uma equipe formada por pessoas tão dedicadas e talentosas. É com grande prazer que deixo o projeto para ambos darem continuidade, visando o aperfeiçoamento da presente solução. Desejo-lhes muito sucesso com a continuação do projeto.

Aos grandes amigos e ex-alunos da Escola Politécnica da USP, André Masakazu Ferreira Soares e Vitor Saiki Scarpinetti, pela eficiente solução fornecida como base para o desenvolvimento do presente projeto, além de conselhos e auxílio prestados ao longo do ano, sobretudo no que diz respeito à parte escrita deste trabalho. Bons resultados foram alcançados devido ao sucesso do projeto antecessor, que abriu caminho para que o presente trabalho se tornasse realidade.

Ao amigo e parceiro Valtemar Fernandes Cardoso, por toda ajuda e suporte prestados durante a fase de montagem do *hardware*. Graças a sua generosidade foi possível montar a placa eletrônica deste projeto de forma rápida e eficiente, fornecendo ótimos resultados. Desejo-lhe muito sucesso em seu mestrado e em sua vida profissional.

Ao professor Francisco Antonio Marino Salotti, que cedeu a excelente estrutura

presente no Instituto de Energia e Ambiente da USP para testes ao longo do semestre. Esta estrutura conta com um moderno dinamômetro inercial, fato que foi definitivo para o sucesso de diversos testes realizados no veículo. Espera-se que o presente projeto motive trabalhos em conjunto do Departamento de Engenharia de Sistema Eletrônicos da Escola Politécnica com o Instituto de Energia e Ambiente, visando futuros projetos em soluções híbridas.

A todos os parceiros da FATEC Santo André, Ricardo Andrade, Marcos Antonio de Carvalho Guedes, Felipe Serafim Albaladejo, Cynthia Thamires da Silva e demais colegas, por todo auxílio prestado durante o desenvolvimento do projeto.

Ao professor da FATEC Santo André, Marco Aurélio Fróes, pelo suporte fornecido durante testes com o dinamômetro inercial, identificando problemas e aconselhando correções.

A Freescale, por todo o apoio prestado durante o projeto. Espera-se uma aproximação maior do fabricante com a Universidade, visando futuros trabalhos.

Ao Laboratório de Sistemas Integráveis da Escola Politécnica da USP pela estrutura cedida para implementação do *firmware* inicial e testes longo do ano, além do apoio na montagem do *hardware*.

A FATEC Santo André, pelo imenso apoio dado, cedendo o veículo e equipamentos essenciais para os testes como, por exemplo, o dinamômetro inercial.

Por fim, a todos que colaboraram direta ou indiretamente na execução deste trabalho.

Resumo

O presente projeto desenvolve uma unidade de gerenciamento eletrônico (conhecida também como ECU - *Eletronic Control Unit*) para um motor a combustão interna. Gerenciando corretamente ignição, injeção, relés e válvula borboleta, a unidade deverá ser capaz de controlar adequadamente a rotação do motor, mesmo em condições de carga. A execução do projeto envolve o desenvolvimento de um *hardware* (placa de circuito impresso), com todas as entradas e saídas necessárias para interligar sensores e atuadores ao sistema, além de fornecer uma interface de comunicação para monitoramento de parâmetros em tempo real. Além do *hardware*, o projeto envolve desenvolvimento de *firmware*, correspondente à programação dos microcontroladores presentes no sistema, e desenvolvimento de *software*, correspondente a uma aplicação que, rodando em computador externo com sistema operacional *Windows*, deverá se comunicar com o *hardware* a fim de possibilitar a visualização de parâmetros e variáveis de erro. A mesma aplicação deverá também fornecer uma opção para se controlar o motor a partir do computador, simulando, neste caso, a leitura do pedal de aceleração do veículo. O projeto tem como meta final a sua implantação em um motor Volkswagen 2.0L aplicado a um veículo modelo Polo Sedan 2004, de tal forma que se possa avaliar o desempenho do motor em condições de carga, através do uso de um dinamômetro inercial. A elaboração e execução serão feitas em cooperação com o departamento de eletrônica automotiva da Faculdade de Tecnologia de Sandro André, o qual cederá o veículo e o dinamômetro para testes. É importante ressaltar também que o projeto teve início com o desenvolvimento em conjunto da parte conceitual e teórica ao projeto Otto (SCARPINETTI; SOARES, 2012) em 2012, desenvolvido por André Masakazu Ferreira Soares e Vitor Saiki Scarpinetti, justificando o fato das partes introdutória e teórica serem semelhantes em ambos os projetos.

Palavras-chaves: Unidade de gerenciamento eletrônico. Injeção eletrônica. Ignição eletrônica. Válvula borboleta. Eletrônica automotiva. Motor a combustão interna. Controle de rotação.

Abstract

The present project develops an electronic management unit (also known as ECU - Electronic Control Unit) for an internal combustion engine. The system must properly manage ignition, injection, relays and throttle valve to control the engine speed, even under load conditions. The project execution involves hardware development (printed circuit board) with all inputs and outputs necessary to interconnect sensors and actuators to the system, besides providing a communication interface for real time monitoring. In addition to the hardware, the project also involves firmware development, corresponding to the microcontroller programming present in the system, and software development, corresponding to an application which, once running on external computer with Windows operating system, it is able to communicate to the hardware in order to allow the visualization of parameters and error variables. The same application must also provide an option to control the engine from the computer, by simulating the sensor of the acceleration pedal. As a final goal, the project is intended to control an Volkswagen 2.0L engine applied to a vehicle model Polo Sedan 2004, in such a way that it is possible to evaluate the engine performance under load conditions, by using an inertial dynamometer. The elaboration and execution will be done in cooperation with Automotive Electronic Department of Faculdade de Tecnologia de Sandro André, which will provide the vehicle and dynamometer for testing. Furthermore, it is important to note that the project begun with the development of the conceptual and theoretical part related to the Otto project (SCARPINETTI; SOARES, 2012), in 2012, developed by André Masakazu Ferreira Soares and Vitor Saiki Scarpinetti, justifying the fact that the introductory and theoretical parts are similar in both projects.

Keywords: Electronic management unit. Electronic injection. Electronic ignition. Throttle valve. Automotive electronic. Internal combustion engine. Speed control.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

2D	2 Dimensões ou Bi-dimensional
3D	3 Dimensões ou Tri-dimensional
A/C	Ar/Combustível
ABS	Anti-Lock Bracking System
ADC	Analog-to-Digital Converter
b/s	Bits por Segundo
CAD	Computer Aided Design
CAN	Controller Area Network
CI	Circuito Integrado
cm	Centímetro(s)
cm^3	Centímetro(s) cúbico(s)
CO	Monóxido de carbono
cv	Cavalo(s), referente à unidade de potência
E/S	Entrada/Saída, referente à direção da pinagem do hardware
ECU	Electronic Control Unit
FATEC	Faculdade Tecnológica de São Paulo
FLEX	Flexível, referente ao sistema bicombustível presente nos automóveis
g	Grama(s)
GND	Ground
HC	Hidrocarboneto(s)
Hz	Hertz
I2C	Inter-Integrated Circuit
kB	Quilobytes

kb/s	Quilobits por Segundo
kHz	Quilohertz
kPa	Quilopascal(is)
L	Litro(s)
LCD	Liquid Cristal Display
LED	Light Emitting Diode
LSI	Laboratório de Sistemas Integráveis
MAP	Manifold Absolute Pressure
Mb/s	Megabits por Segundo
MHz	Megahertz
MISO	Master Input Slave Output
mkgf	Metro x quilograma-força, referente à unidade de torque
MOSI	Master Output Slave Input
MPU	Memory Protection Unit
ms	Milisegundo(s)
NOx	Óxido(s) Nitroso(s)
NTC	Negative Temperature Coefficient
OBD	On Board Diagnostics
PC	Personal Computer
PI	Proporcional Integral
PLL	Phase-Locked Loop
PMI	Ponto Mínimo Inferior
PMS	Ponto Máximo Superior
PSI	Departamento de Engenharia de Sistemas Eletrônicos
PTC	Positive Temperature Coefficient
PWM	Pulse Width Modulation, referente à modulação por largura de pulso

RPM	Rotações Por Minuto
SCK	Serial Clock
SMD	Surface Mount Device
SPI	Serial Peripheral Interface
SS	Slave Select
TCS	Traction Control System
TPS	Throttle Position Sensor
UART	Universal Asynchronous Receiver Transmitter
uC	Microcontrolador
USB	Universal Serial Bus
USP	Universidade de São Paulo
V	Volts
8v	8 válvulas, referente ao número de válvulas presente no motor
VB	Válvula Borboleta

Sumário

1 Introdução

Com a expansão da produção de carros movidos a motores bicombustíveis e a preocupação cada vez maior com a emissão de gases, o Brasil vive uma demanda crescente por conhecimentos na área de eletrônica automotiva, especialmente em busca de soluções inovadoras no gerenciamento eletrônico de motores, a fim de reduzir a emissão de poluentes e melhorar a eficiência do motor.

A Escola Politécnica da USP tem se dedicado ao desenvolvimento de unidades de gerenciamento eletrônico para motores a combustão interna, inicialmente com uma versão projetada e testada em *mockup* (SCARPINETTI; SOARES, 2012). Atualmente, no âmbito deste projeto, foi desenvolvida uma nova versão, testada em um motor Volkswagen 2.0L aplicado em um veículo modelo Polo. Este projeto está inserido num esforço conjunto da Faculdade de Tecnologia (FATEC) de Santo André e do Departamento de Engenharia de Sistemas Eletrônicos (PSI) da Escola Politécnica da USP de se desenvolver uma ECU didática projetada exclusivamente para o uso acadêmico e científico, de forma a possibilitar desenvolvimentos futuros. Além disto, é importante ressaltar que o projeto Otto II consiste em uma continuação do projeto Otto de 2012 (SCARPINETTI; SOARES, 2012), sendo que ambos foram iniciados em conjunto no primeiro semestre de 2012 com o desenvolvimento conceitual do trabalho, fato que justificam as partes introdutória e conceitual serem semelhantes em ambos.

Como motivação e importância para o presente projeto, assume-se como possível implementação futura o sistema de controle de tração (TCS - *Traction Control System*), item raramente presente nos veículos fabricados no Brasil. O controle de tração, sistema *dual* do sistema de freios *Anti-Lock Bracking System* (ABS), tem a função de controlar o escorregamento durante a aceleração do veículo, ou seja, evitar que o veículo gire em falso, seja por uma aceleração brusca ou por uma queda no coeficiente de atrito entre o pneu e o pavimento (BOSCH, 2005) . Assim, é possível, com o controle do escorregamento, garantir um maior coeficiente de atrito entre o pneu e o chão, o que resulta em uma maior força aplicada ao pneu. Neste caso tem-se uma força maior para acelerar o veículo, o que representa um aumento da eficiência do motor. Contudo, para se implementar esse sistema de controle é necessário uma intervenção no controle eletrônico do motor, o que exige o conhecimento da ECU que o controla.

Com isto, espera-se, com o desenvolvimento deste projeto, viabilizar projetos futuros que busquem a inovação em áreas como injeção, ignição e gerenciamento do motor como um todo, bem como melhorias no controle do veículo, de forma a otimizar fatores como segurança, desempenho e conforto dos automóveis.

1.1 Contexto do Trabalho

1.1.1 Ciclo Otto

Criado no final do século XIX, o ciclo Otto corresponde a um ciclo termodinâmico com a finalidade de se transformar a energia contida no combustível em trabalho mecânico, com o objetivo de impulsionar e criar o movimento do pistão do motor (MANAVELLA, 1996).

O ciclo é dividido em quatro principais etapas: admissão, compressão, combustão e exaustão. Para que o ciclo ocorra de forma controlada e segura, válvulas independentes são utilizadas para controlar a entrada de combustível e a saída de gases de escape (resultantes da combustão), além de uma vela usada para a geração da centelha de ignição. Ao final do ciclo, o motor terá realizado duas voltas completas. As quatro etapas do ciclo estão descritas a seguir (HEYWOOD, 1988):

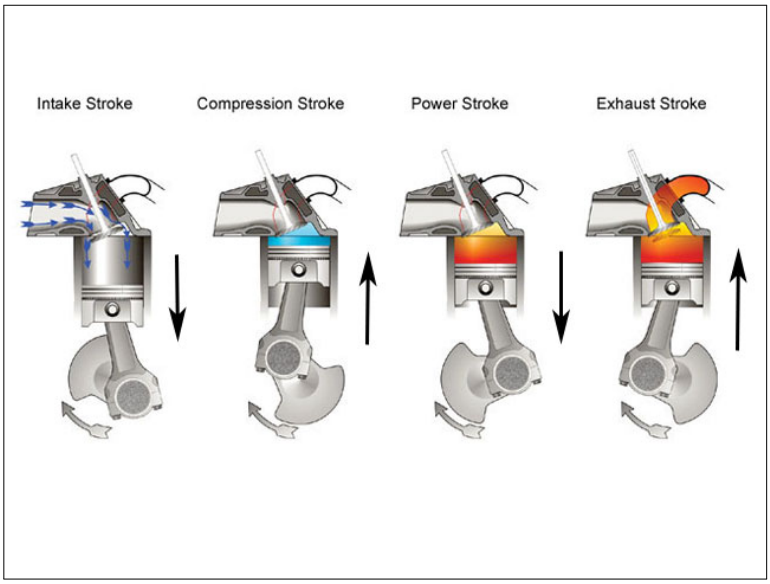
1. **Admissão**: Com o pistão na posição PMS (Ponto Máximo Superior), a válvula de admissão é aberta e, ao mesmo tempo, o bico injetor é acionado e o combustível é inserido no interior da câmara, finalizando o processo no momento em que o pistão atinge seu PMI (Ponto Máximo Inferior);

2. **Compressão**: Com as válvulas de admissão e escape fechadas, o pistão movimenta-se do PMI ao PMS, comprimindo a mistura ar/combustível;

3. **Combustão**: Com a mistura ar/combustível completamente comprimida, o pistão atinge o PMS novamente e, neste momento, é enviado um comando a vela de ignição, o que resulta em uma centelha para a combustão da mistura gasosa e, consequentemente, em um aumento iminente de pressão na câmara, o que empurra o pistão ao PMI;

4. **Exaustão**: Após a combustão, a válvula de exaustão do cabeçote é aberta e o gás resultante da queima da mistura é liberado para o sistema de escape do motor, através do retorno do pistão do PMI ao PMS.

Figura 1 – Ciclo Otto



Fonte: Yanswerz Blogspot (2009)

1.1.2 A Importância da Eletrônica no Gerenciamento do Motor

Atualmente, o controle eletrônico do motor a combustão é fundamental para se obter a máxima potência com o menor consumo de combustível e menor emissão de poluentes. Todavia, é importante ressaltar que mesmo com o gerenciamento eletrônico o motor à combustão interna não apresenta rendimentos elevados devido, principalmente, às elevadas perdas de energia que ocorrem na maior parte na forma de calor (MANAVELLA, 1996).

Integrando injeção eletrônica e ignição eletrônica em um único módulo, as primeiras ECUs surgiram com funções muito básicas. Entretanto, com as exigências de leis locais para a redução de emissão de poluentes e redução do consumo de combustível, aliado à demanda de novos veículos com maior potência e maior segurança, novas funções foram adicionadas às ECUs existentes. Como exemplo, cita-se o sistema de controle *Motronic* da Bosch, que ao longo do tempo recebeu novas funções, como regulagem da rotação de marcha lenta, controle de estequiometria, controle do eixo de comando para redução da emissão de gases de escape, dentre outros (BOSCH, 2005).

A unidade de gerenciamento eletrônico do motor, conhecida também como ECU (*Eletronic Control Unit*), corresponde a um sistema capaz de controlar o motor à combustão interna com base em informações de sensores, dosando a quantidade de combustível a ser injetada através do tempo de abertura das válvulas injetoras, controlando a ignição eletrônica, e comandando outros atuadores como a válvula borboleta (VB) de admissão de ar (CONTINENTAL, 2013).

1.2 Objetivos

1.2.1 Geral

Tem-se como principal objetivo deste projeto o desenvolvimento de uma unidade de gerenciamento eletrônico para um motor a combustão interna modelo Volkswagen 2.0L, aplicado a um veículo modelo Polo Sedan 2004, substituindo integralmente a unidade original do veículo. Constituído de *hardware*, *firmware* e *software*, o sistema eletrônico deverá ser capaz de efetuar a correta leitura dos sensores presentes no motor e calcular todos os parâmetros de atuação, como tempos de abertura dos bicos injetores, adiantamentos dos sinais enviados às velas de ignição, acionamento de relés e posição da válvula borboleta, de modo a garantir um controle estável da rotação do motor, na faixa de 800 RPM (marcha lenta) até 6000 RPM. Além disto, o motor deverá reagir quando for submetido a carga, fornecendo torque ao seu eixo.

O sistema deverá também realizar outras tarefas como condicionar os sinais provenientes dos sensores, prever na ação de controle as limitações físicas dos atuadores, fornecer uma interface para monitoramento e diagnose do sistema, além de fornecer uma opção para controle do motor a partir de um pedal simulado em computador[1].

1.2.2 Específicos

- Desenvolvimento de um *Hardware* aperfeiçoado, tomando como referência a solução implementada pelo Projeto Otto (SCARPINETTI; SOARES, 2012) e a primeira versão do *hardware* desenvolvido pela FATEC Santo André (DIAS, 2011), mantendo uma arquitetura descentralizada, porém empregando processadores automotivos mais robustos do fabricante Freescale (FREESCALE, 2012);

- Desenvolvimento de *Firmware* correspondente à programação dos microcontroladores (uCs) presentes no sistema. O *firmware* deverá levar em conta a estratégia para controlar de maneira estável a rotação do motor Volkswagen 2.0L, de tal forma que este opere adequadamente desde a partida até 6000 RPM, mesmo quando submetido à carga (neste caso o motor deverá reagir fornecendo torque ao eixo);

- Desenvolvimento de estratégia de partida do motor, incluindo partida com o motor frio;

- Gerenciar o ligamento e o desligamento de relés automotivos, necessários para o funcionamento do motor;

[1] Neste caso, o controle do motor será realizado a partir do *software* de monitoramento e diagnose, deixando de responder ao pedal de aceleração do veículo.

- Leitura dos sensores de forma rápida e confiável, com nível de ruído baixo suficiente a ponto de não causar comportamentos inesperados no controle realizado pelo *firmware*;
- Desenvolver uma estratégia de controle de rotação, de forma a garantir que esta esteja mais próximo possível da referência dada pelo sensor de posição do pedal de aceleração. Para isto será necessário adotar um controle em que o sinal de atuação irá afetar tanto o tempo de injeção de combustível, como também a posição de abertura da VB. Serão toleráveis erros de regime de até 200 RPM quando o motor estiver em vazio (sem carga);
- Desenvolver uma estratégia para controle estequiométrico da mistura ar/combustível quando a rotação estiver próximo da referência, ou seja, quando não houver aceleração nem desaceleração. Por simplificação, este controle será realizado em malha aberta;
- Desenvolver uma estratégia para manter o motor funcionando de modo estável mesmo enquanto este estiver fora da sua temperatura de operação (motor frio).

1.3 Justificativa do projeto

A demanda por conhecimentos na área de eletrônica automotiva no Brasil está cada vez maior. Projetos inovadores que necessitariam intervir diretamente no controle do motor (como, por exemplo, o controle de tração, citado anteriormente) se tornam muito complicados de serem implementados em unidades de gerenciamento presentes hoje no mercado, dado que estas são tratadas com sigilo intelectual absoluto por parte de fabricantes e montadoras de veículos. Sendo assim, se torna muito difícil o trabalho de pesquisa visando inovação na área, o que expõe uma grande deficiência do país por conhecimentos específicos na área de eletrônica automotiva. Grande parte do desenvolvimento acadêmico no setor é fundamentado em suposições teóricas e engenharia reversa.

Nota-se que após o lançamento da tecnologia de motores bicombustíveis, o avanço no conhecimento científico no ramo automotivo da eletrônica não acompanha a crescente demanda por pesquisa no setor. Ressalta-se ainda que grande parte das unidades eletrônicas presentes no mercado que são adaptadas a motores flexíveis (flex) bicombustíveis não apresentam desempenho satisfatório, o que muitas vezes se deve ao fato de tais unidades serem oriundas de países em que o bicombustível ainda não é uma realidade.

Com o projeto Otto de 2012 (SCARPINETTI; SOARES, 2012), a Escola Politécnica da USP deu início ao desenvolvimento de unidades de gerenciamento eletrônico de motores a combustão interna, dedicando-se inicialmente ao desenvolvimento de uma unidade para operar em "‘mockup"’. Com intuito de dar continuidade ao desenvolvimento,

o projeto visa agora aperfeiçoar a solução proposta em 2012 (SCARPINETTI; SOARES, 2012), desenvolvendo uma nova unidade de gerenciamento eletrônico para operar em um motor aplicado em um veículo, a fim de avaliar o seu desempenho em condições de carga.

Futuramente novas unidades de gerenciamento serão desenvolvidas, no âmbito de outros projetos. Pode-se afirmar que estas unidades futuras apresentarão novas funções como uma gestão eficiente de motores bi-combustíveis, redução do consumo de combustível, ou redução de emissões. Para tanto, torna-se fundamental o desenvolvimento de uma unidade básica, conforme proposto neste projeto, o que servirá de base para os desenvolvimentos citados anteriormente.

1.4 Organização do trabalho

O presente projeto foi organizado nas etapas destacadas abaixo, iniciando-se com o desenvolvimento do *hardware* e seguindo com o desenvolvimento do *firmware* e do *software*:

- Concepção e validação do novo *hardware*, implementado com novos microcontroladores e novos recursos;
- Desenvolvimento de *firmware* para controle de posição da VB, com testes em bancada;
- Desenvolvimento de *firmware* para controle dos instantes de acionamento da injeção e ignição, com testes em bancada;
- Desenvolvimento de *firmware* para aquisição de dados dos sensores;
- Desenvolvimento de *software* de monitoramento, que recebe e exibe na tela todas as informações de sensores e variáveis de erro do sistema, bem como tempos de atuação e a rotação calculada;
- Aperfeiçoamento de *firmware* para leitura de sensores e cálculo da variável final associada ao valor de tensão lido, tomando como referência para este cálculo a curva característica do sensor;
- Aperfeiçoamento de *software* de monitoramento, com implementação do pedal simulado para controle do motor diretamente do computador;
- Aperfeiçoamento de *firmware* para cálculos mais precisos dos sinais de atuação (injeção e ignição);
- Aperfeiçoamento de *firmware* para cálculo do tempo de injeção com base na massa de ar estimada;

- Desenvolvimento de *firmware* para controle de relés;
- Desenvolvimento de *firmware* para controle de rotação, com testes em bancada;
- Testes finais no veículo, aplicado em dinamômetro inercial a fim de avaliar o desempenho do motor em condições de carga.

2 Referencial Teórico

2.1 Estado da Arte no mundo

A área de eletrônica automotiva é atualmente caracterizada por diversos módulos eletrônicos (ECUs) interligados por um barramento regido pelo protocolo *Controller Area Network* (CAN) (BOSCH, 2013e), que será abordado posteriormente neste capítulo. O termo ECU deixou de se referir apenas ao controle do motor, abrangendo também itens como segurança, conforto e dirigibilidade. Hoje estima-se que um carro mais sofisticado possua mais de 100 ECUs, cada uma realizando uma tarefa distinta no veículo (SHEDEED et al., 2013).

Apesar de moderna e contar com inúmeros recursos, as ECUs atuais ainda não possuem, em alguns casos, mecanismos de segurança adequados para lidar com ataques e intervenções externas, mesmo nos veículos mais modernos. Segundo a BBC NEWS, dois pesquisadores americanos especializados em segurança, Charlie Miller e Chris Valasek, revelaram vulnerabilidades em ECUs automotivas. Com um controle de vídeo game e com cabos para se conectar ao sistema de diagnóstico do carro (OBD - *on board diagnostics*), eles demostraram que é possível se controlar algumas funcionalidades do veículo, como a direção, acionamento de freios e até a indicação do nível de combustível no painel frontal do motorista. Segundo os autores da pesquisa, a ideia do trabalho é aumentar a conscientização com relação à questões de segurança nos automóveis. Os testes foram realizados em um Ford Espace e um Toyota Prius (BBC NEWS, 2013).

Figura 2 – Controle utilizado pelos pesquisadores

Fonte: BBC News (2013)

2.2 Estado da Arte no Brasil

No Brasil destacam-se as ECUs para gerenciar motores flex, atendendo as necessidades de clientes locais. Como exemplo, cita-se o desenvolvimento do sistema de partida a frio para motores *flex*, proposto pela empresa Continental, que em 2012 implantou na cidade de Salto em São Paulo um novo centro tecnológico voltado para o desenvolvimento de *powertrain*, totalizando um investimento de 11 milhões de euros (CONTINENTAL, 2012).

Destaca-se também as inovações propostas pela empresa Magneti Marelli no gerenciamento de motores *flex*, focando em novos níveis de emissões e maior eficiência energética. Segundo a empresa, está em desenvolvimento uma nova ECU que terá maior capacidade de processamento de dados e integrará novos algoritmos para identificar a mistura do combustível, utilizando a informação de sensores presentes no motor (sonda lambda, sensor de detonação, rotação, velocidade e temperatura) e o ruído emitido pelo motor durante a combustão. Sendo assim, a identificação da mistura será realizada sem a necessidade de sensores adicionais, o que diminui o custo da aplicação. A empresa afirma também que já está consolidada a tecnologia para o desenvolvimento de novos bicos injetores, desenvolvidos especialmente para motores flex. Já a nova ECU estará disponível no mercado até o fim de 2014 (AUTOMOTIVE BUSINESS, 2013).

2.3 Principais conceitos abordados

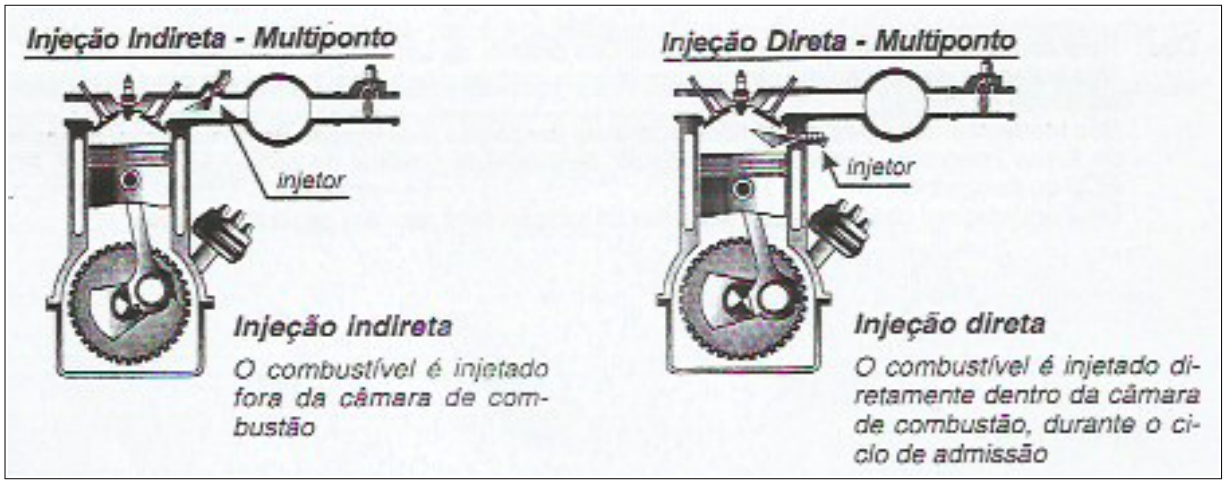
2.3.1 Sistemas de Injeção

Os sistemas de injeção para motores a combustão interna ciclo Otto podem ser classificados de acordo com o local onde se forma a mistura ar/combustível (a/c) ou de acordo com o número de válvulas de injeção utilizadas (MANAVELLA, 1996).

1. **Segundo o local onde se efetua a injeção**

   - Injeção Indireta: A injeção de combustível ocorre fora da câmara de combustão, o que aumenta o desperdício de combustível no trajeto da mistura a/c até a câmara de combustão;
   - Injeção Direta: O combustível é injetado dentro da câmara de combustão. Neste caso há economia de combustível, além da redução da emissão de poluentes.

Figura 3 – Injeção Direta e Indireta

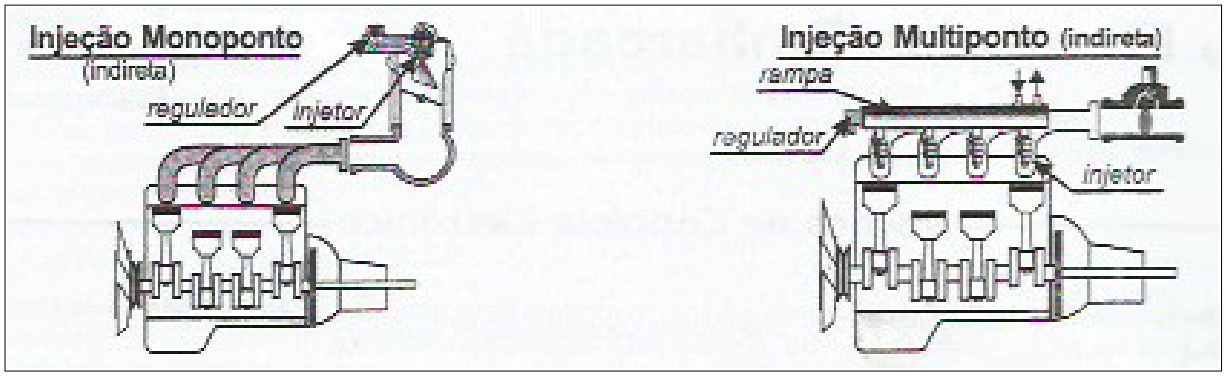


Fonte: Manavella (1996)

2. **Segundo o número de válvulas de injeção utilizadas**

   - Injeção monoponto: Neste tipo de sistema, a injeção ocorre somente em um ponto, acima da válvula borboleta. Assim, existe apenas um injetor para todos os cilindros. Apesar de ser mais simples, este sistema apresenta menor rendimento do que o sistema de injeção multiponto devido, principalmente, à condensação de combustível nas paredes do coletor;
   - Injeção Multiponto: Neste tipo de sistema, há um bico injetor por cilindro, o que permite uma dosagem de combustível mais precisa por cilindro.

Figura 4 – Injeção Monoponto e Multiponto



Fonte: Manavella (1996)

Com relação ao sistema de injeção multiponto, o acionamento da injeção pode ser feito de diversas maneiras:

- *full-group*: acionamento simultâneo da injeção em todos os cilindros. Em geral este tipo de acionamento é bastante utilizado em motores sem sensor de fase;

- banco a banco: ocorre simultaneamente o acionamento de 2 cilindros[1];
- sequencial: acionamento da injeção ocorre somente em 1 cilindro;

O método de acionamento sequencial é, de modo geral, o mais adequado, dado que ocorre injeção apenas em um cilindro, o que representa uma economia de combustível. No entanto, para motores que não possuem sensor de fase geralmente se adota a injeção *full-group* e banco a banco, principalmente durante a partida (ALBALADEJO, 2013). Especificamente neste projeto, a injeção ocorre sempre de forma sequencial desde a partida, diferentemente do projeto Otto (SCARPINETTI; SOARES, 2012) que não contava com o sensor de fase e utilizava como combustível o etanol.

2.3.2 Tipos de Arquitetura

Como opções de arquitetura de *hardware*, destacam-se a arquitetura centralizada e a arquitetura descentralizada.

Na arquitetura centralizada, tem-se uma única unidade de processamento, responsável por ler os sinais enviados por todos os sensores, processá-los e enviar os sinais de controle aos atuadores. Este fato mostra que a unidade de processamento, neste tipo de arquitetura, possui grande responsabilidade, pois caso ocorra alguma falha o funcionamento inteiro do sistema pode ser comprometido. Consequentemente, o *firmware* correspondente a esta unidade de processamento[2] acaba sendo bastante complexo, já que deve garantir a robustez do sistema, a fim de evitar situações de falha que possam comprometer o sistema.

Na arquitetura distribuída (ou descentralizada), o sistema é dividido em vários blocos, sendo cada um controlado por uma unidade de processamento. Assim, a responsabilidade de cada processador é menor do que no caso da arquitetura centralizada, pois caso ocorra falha em um processador, os demais poderão continuar trabalhando sem comprometer o funcionamento do sistema. Nesse tipo de arquitetura pode-se, por exemplo, realizar a leitura e tratamento dos sinais dos sensores por um dos processadores e deixar o processamento destes mesmos sinais para outro processador.

Além disto, a arquitetura descentralizada se mostra vantajosa quando se necessita alterar parte do projeto, visando correções ou melhorias. Caso se necessite, por exemplo, alterar um dos microcontroladores do sistema, o projetista apenas terá o trabalho de refazer o *hardware* e o *firmware* associados à ele, mantendo intacto o projeto (*hardware* e *firmware*) dos demais microcontroladores do sistema. Já na arquitetura centralizada, o projetista necessitaria alterar o *hardware* e reescrever todo o código do *firmware* da unidade de processamento central, sendo que nesta arquitetura o *firmware* é mais complexo quando comparado com o *firmware* da arquitetura descentralizada.

[1] Considera-se um motor de 4 cilindros.
[2] Esta unidade de processamento corresponde a um microcontrolador no presente projeto.

Um ponto importante na arquitetura descentralizada é a comunicação. Devido à necessidade de troca de dados entre os blocos que formam o sistema, é necessário uma via de comunicação rápida e confiável (principalmente uma via imune à ruídos), a fim de fornecer ao sistema um desempenho satisfatório. Neste caso, o sistema poderá ser comprometido caso ocorra alguma falha no protocolo de comunicação entre os blocos, o que revela uma desvantagem da arquitetura descentralizada em relação à centralizada.

Uma rápida comparação entre as duas poderia ser feita de tal forma:

- A centralizada necessita de um único controlador com grande poder de processamento; a descentralizada necessita de vários controladores com menor poder de processamento;
- A centralizada possui todos os sensores e atuadores conectados a ela; a descentralizada é dependente de um protocolo de comunicação para envio de parâmetros de um controlador para o outro;
- A centralizada possui um *firmware* mais complexo, controlando tudo; a descentralizada possui vários *firmwares* (um para cada controlador) mais simples;
- A descentralizada permite alterações mais simples no projeto do que a centralizada.

Neste projeto, optou-se por manter a arquitetura descentralizada utilizada no projeto Otto (SCARPINETTI; SOARES, 2012) devido, principalmente, ao satisfatório funcionamento da ECU daquele projeto. Apesar do *hardware* deste projeto utilizar microcontroladores mais robustos e eficientes, a solução descentralizada ainda é vantajosa, pois os microcontroladores deste projetos deverão realizar um número maior de tarefas, e ainda realizá-las de maneira eficaz, o que exige muitos cálculos. Além disto, a divisão do *firmware* também é um ponto vantajoso na arquitetura descentralizada, uma vez que é possível modificar um deles sem afetar diretamente os outros, facilitando o desenvolvimento dos programas individualmente. Fora isto, o microcontrolador (uC) utilizado no projeto suporta diversos protocolos de comunicação, o que possibilita a implementação do sistema utilizando uma arquitetura descentralizada.

É importante ressaltar que a solução centralizada não é descartada pelo projeto, sendo ainda uma alternativa para o desenvolvimento de outras ECUs, no âmbito de projetos futuros.

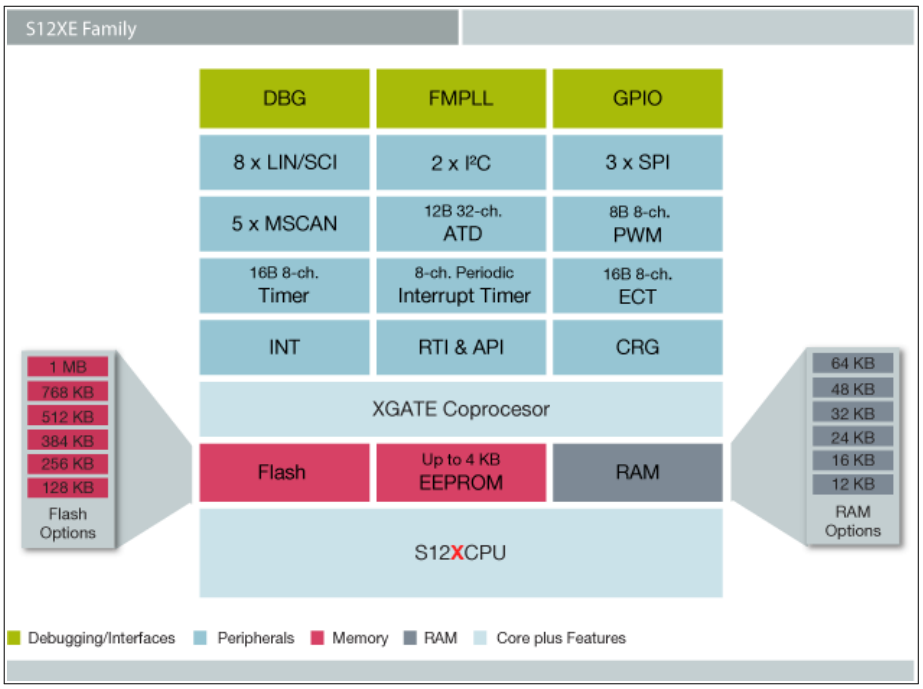
2.3.3 O microcontrolador Freescale S12X

Figura 5 – O microcontrolador automotivo S12XE da Freescale

Fonte: o autor

A família S12XE da Freescale surgiu de uma inovação em relação à família S12XD, no que diz respeito à melhor integrabilidade e aos recursos aperfeiçoados presentes no processador. Dentre as inovações, pode-se citar novos recursos como a unidade de proteção à memória (*memory protection unit* – MPU), o coprocessador XGate, um rápido e eficiente conversor analógico-digital de 12 bits (*analog-to-digital converter* – ADC), dentre outros (FREESCALE, 2013). Além disto, o microcontrolador de 16 bits possui certificação para operar no ramo automotivo, caracterizando-o como robusto, eficiente e confiável, fatores essenciais para o projeto, dado que o ambiente de operação é considerado agressivo devido às altas temperaturas, alto nível de ruído eletromagnético e alto nível de vibração mecânica.

Figura 6 – Diagrama de blocos da família de microcontroladores S12XE



Fonte: Freescale (2013)

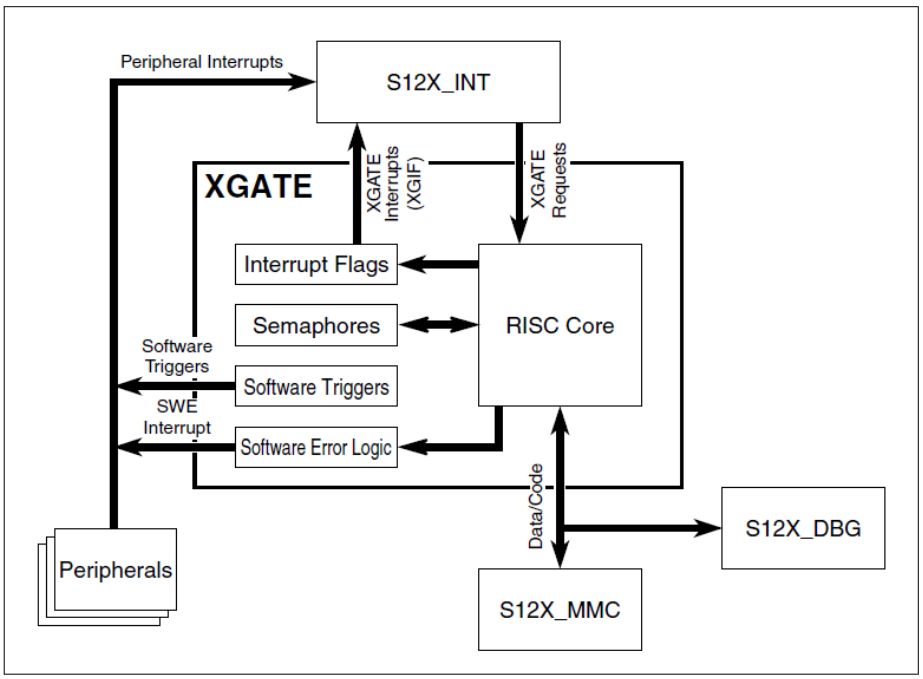
Neste projeto será utilizado o microcontrolador S12XET256. Dentre os recursos deste microcontrolador, destacam-se[3] (FREESCALE, 2012):

- 256 kB de memória flash e 16 kB de memória RAM;
- 16 canais ADC, com resolução de 12 bits e tempo de conversão de até 3 microssegundos;
- 4 *Timers* de 16 bits com canais e interrupções independentes;
- 8 canais que, associados a um *timer* de 16 bits, possuem recursos avançados como captura de entrada e comparação de saída;
- 3 interfaces CAN, 4 interfaces seriais assíncronas (UART), 3 interfaces SPI e 1 interface I2C;
- 8 canais para geração de PWM de 8 bits

[3] O leitor não deve confundir as quantidades dos itens com o diagrama da figura anterior, dado que este se refere ao microcontrolador topo da família S12XE (no caso o S12XEP100), enquanto que os itens se referem ao S12XET256.

2.3.3.1 O Coprocessador XGate

Figura 7 – Diagrama de blocos do coprocessador XGate



Fonte: Freescale (2012)

Integrado ao *hardware* do microcontrolador S12XE (FREESCALE, 2012), o coprocessador XGate corresponde a um processador RISC de 16 bits que possui a capacidade de executar instruções em até 100 MHz. O XGate tem como finalidade aumentar a capacidade de processamento do microcontrolador. Com a possibilidade de ser programado para atender pedidos de interrupção, o XGate pode ser visto como uma forma de se aliviar a carga do processador principal (S12X). Assim, quando ocorrer uma interrupção proveniente de algum periférico, o processador principal poderá deixar a cargo do coprocessador o atendimento daquela interrupção, de tal forma a se obter um paralelismo no processamento (ou seja, o processador principal não interrompe a sua tarefa para executar a rotina de interrupção, pois esta será executada pelo XGate).

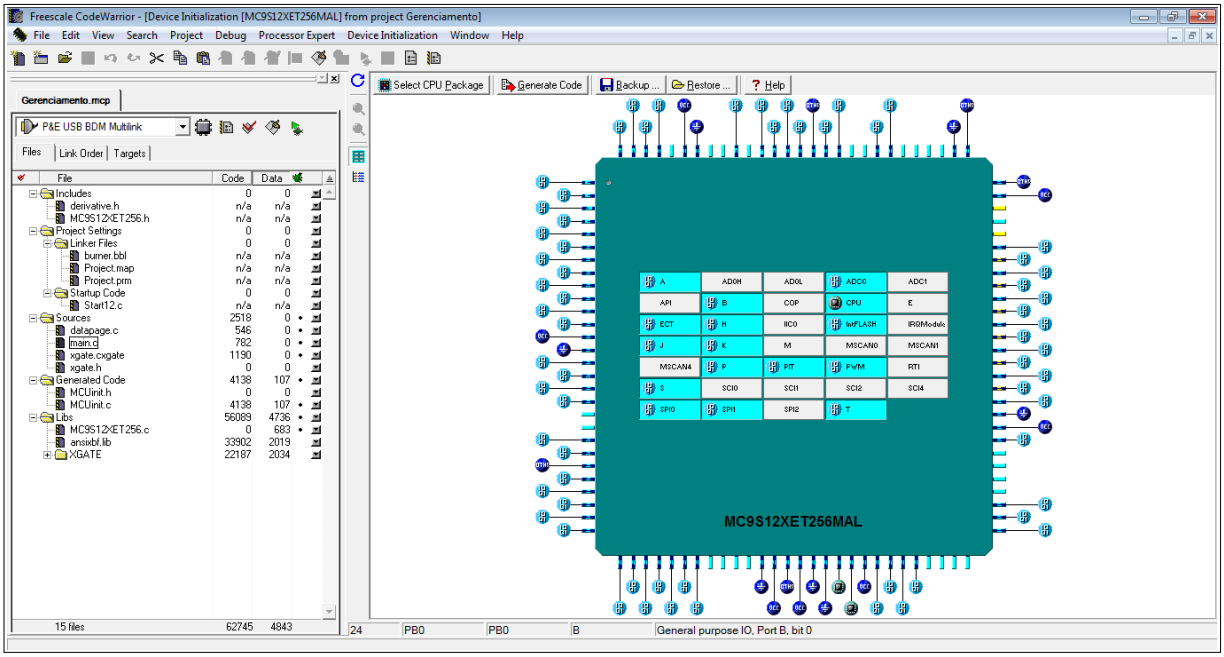
Como dito anteriormente, o núcleo do XGate é um processador RISC, com instruções adequadas para operar manipulações de bit, operações aritméticas e transferência de dados. Uma característica interessante é que tanto o processador principal quanto o coprocessador possuem mecanismos para lidar com acesso simultâneo, uma vez que pode ocorrer de ambos tentarem acessar uma variável ou rotina compartilhada. Quando isto ocorrer, o processador RISC será paralisado até que o recurso (variável ou rotina) esteja disponível novamente, evitando, consequentemente, um erro no processamento do recurso. O programador também pode utilizar "semáforos" implementados em *hardware*, para gerenciar por conta própria o compartilhamento de recursos entre o processador e o

coprocessador.

Além disto, vale ressaltar que o coprocessador possui seu próprio conjunto de instruções, que diferem do conjunto de instruções do processador principal. Sendo assim, caso seja adotada a programação em linguagem de máquina (*assembly*), o programador deverá ter conhecimento tanto do conjunto de instruções do XGate, como também do S12X. Caso seja adotada a programação em linguagem C, o programador pode, por exemplo, mover o código da rotina de uma interrupção do processador para o coprocessador sem qualquer perda, ou vice-versa, dado que as instruções nesta linguagem são as mesmas para ambos.

2.3.4 A ferramenta Codewarrior v5.1 HCS12 da Freescale

Figura 8 – A ferramenta Codewarrior v5.1 da Freescale



Fonte: o autor

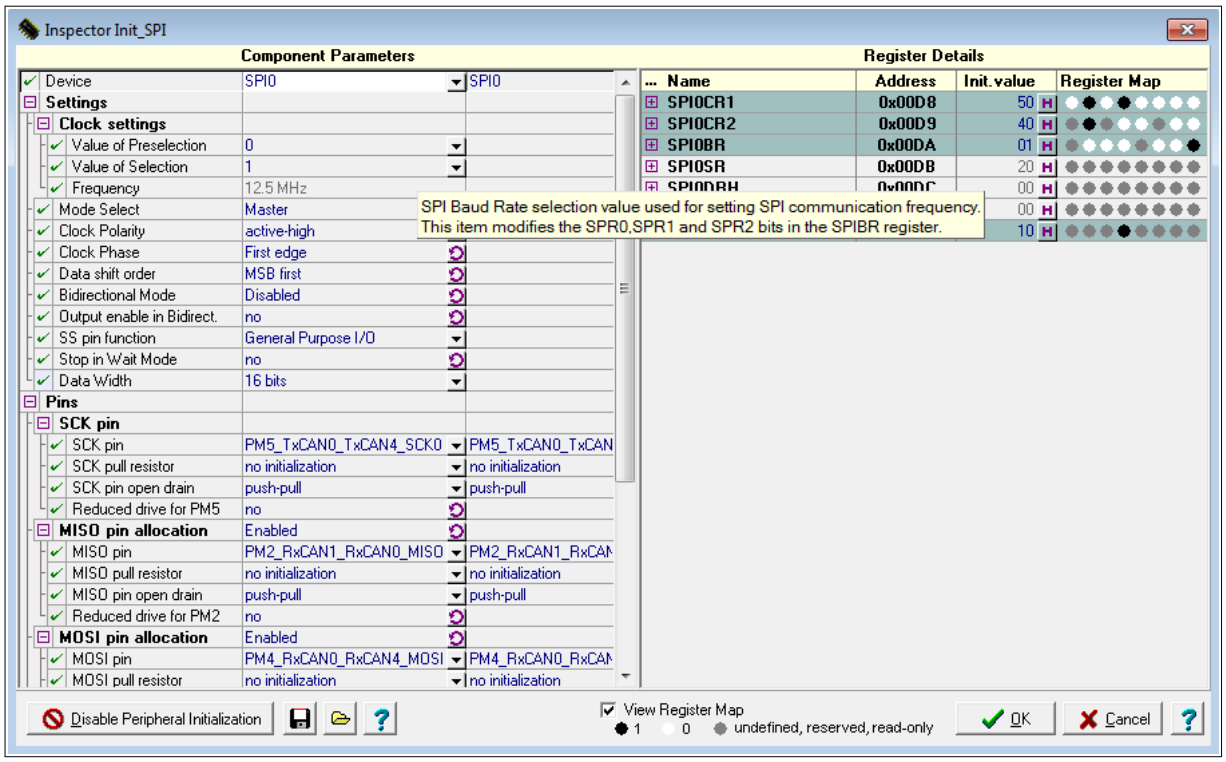
O Codewarrior é uma ferramenta de desenvolvimento para microcontroladores Freescale. No caso, a versão 5.1 é dedicada exclusivamente aos microcontroladores HCS12 e S12X. A ferramenta conta um ambiente bem estruturado e automatizado, o que contribui para o desenvolvimento de projetos embarcados complexos. Além disto, o programa fornece opção para se escrever o código fonte do *firmware* em linguagem *assembly*, C ou C++.

Quando se cria um novo projeto, a ferramenta fornece ao projetista um menu, com opções da linguagem de programação que será utilizada, do microcontrolador empregado no projeto, da arquitetura de memória, do uso de variáveis do tipo *float*, dentre outros. Assim, o projetista não precisa se preocupar com os detalhes de programação envolvidos

na determinação destes parâmetros, o que facilita muito o trabalho. Uma vez criado o projeto, o usuário pode iniciar a escrita do seu código, podendo ainda dividi-lo em vários arquivos.

Com o intuito de reduzir o tempo de trabalho, o programa possui uma ferramenta chamada *Device Initialization*. Esta ferramenta provê uma interface visual para facilitar a inicialização dos registradores do dispositivo. Com isto, o projetista não precisa se preocupar com todos os detalhes envolvidos na inicialização de um registrador. No caso, por exemplo, de um registrador para configurar um divisor de frequência, o usuário não precisaria se preocupar com as fórmulas do *datasheet* para descobrir a frequência final de operação do periférico, uma vez que a própria interface já calcularia e forneceria este valor. Além disto, a interface também é capaz de detectar conflitos na inicialização dos registradores[4]. Uma vez realizada a configuração de inicialização pela interface, gera-se de forma automática um arquivo com todos os registradores inicializados com os valores corretos, ficando a cargo do projetista apenas incluir este arquivo no programa principal (*main*).

Figura 9 – Exemplo de configuração do periférico SPI na interface *Device Initialization*



Fonte: o autor

Além do *Device Initialization*, destaca-se a interface *Processor Expert*. Com ela, é possível implementar, de forma automática, comandos para se realizar diversas tarefas

[4] Um exemplo de conflito seria um pino configurado como entrada analógica e, ao mesmo tempo, como saída digital, o que comprometeria o correto funcionamento do microcontrolador.

com os periféricos do microcontrolador. Pode-se, por exemplo, com apenas um comando ler um dado do periférico responsável pela comunicação SPI. Sendo assim, o projetista não precisa ter conhecimento da manipulação de registradores para efetuar uma leitura do SPI, ficando esta tarefa a cargo do *Processor Expert*. Assim, a ferramenta acelera muito o desenvolvimento de projetos embarcados, principalmente quando estes são bastante longos. Todavia, é importante ressaltar que, ao se utilizar o *Processor Expert*, o processo de depuração do código se torna mais complicado e pode ser comprometido.

Com relação ao uso do coprocessador XGate, o Codewarrior insere no código gerado após a criação do projeto um exemplo de como se utiliza o coprocessador para se atender uma interrupção[5]. Assim, o projetista pode facilmente alterar o exemplo, a fim de utilizar o coprocessador para atender uma interrupção de seu interesse (FREESCALE, 2005).

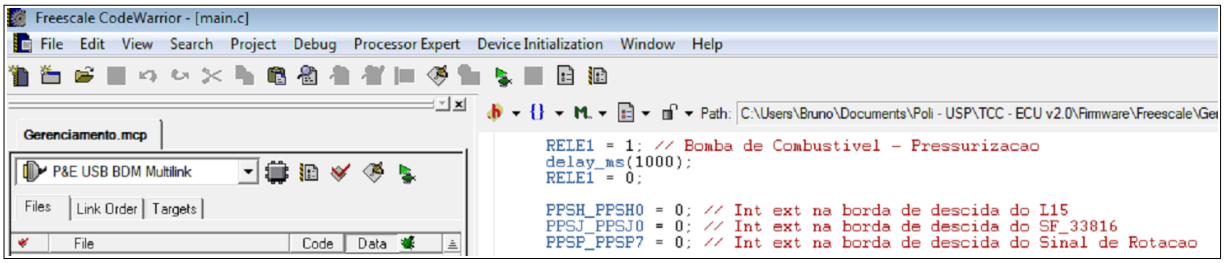
Especificamente no presente projeto, adotou-se o uso da linguagem de programação C para todos os microcontroladores, com uso do coprocessador XGate e uso da interface *Device Initialization* para configurar os valores iniciais dos registradores (devido ao grande número de registradores utilizados nos microcontroladores do sistema). Não se utilizou a interface *Processor Expert* devido, como mencionando anteriormente, à dificuldade na depuração do código.

A interface visual do Codewarrior facilita a construção do código, mesmo quando não se opta por utilizar o *Processor Expert*. Ao se criar o projeto, já se inclui no programa principal um arquivo com todos os registradores e bits do microcontrolador, de tal forma que quando se escreve o nome de um registrador ou bit, este (se escrito na ortografia correta) se destaca automaticamente na cor azul, de modo a facilitar a construção do código. Outro ponto positivo consiste no fato do nome dos registradores e bits serem iguais aos nomes utilizados no datasheet do microcontrolador. Na figura a seguir, retirada do código do uC de Gerenciamento deste projeto, tem-se vários exemplo em que o projetista escreve em determinados bits de registradores[6].

[5] Para a geração deste exemplo, é necessário que a opção para se utilizar o XGate tenha sido marcada no assistente de criação do projeto.

[6] Nota-se, por exemplo, a escrita do valor 0 no bit PPSH0 do registrador PPSH pela instrução PPSH_PPSH0 = 0.

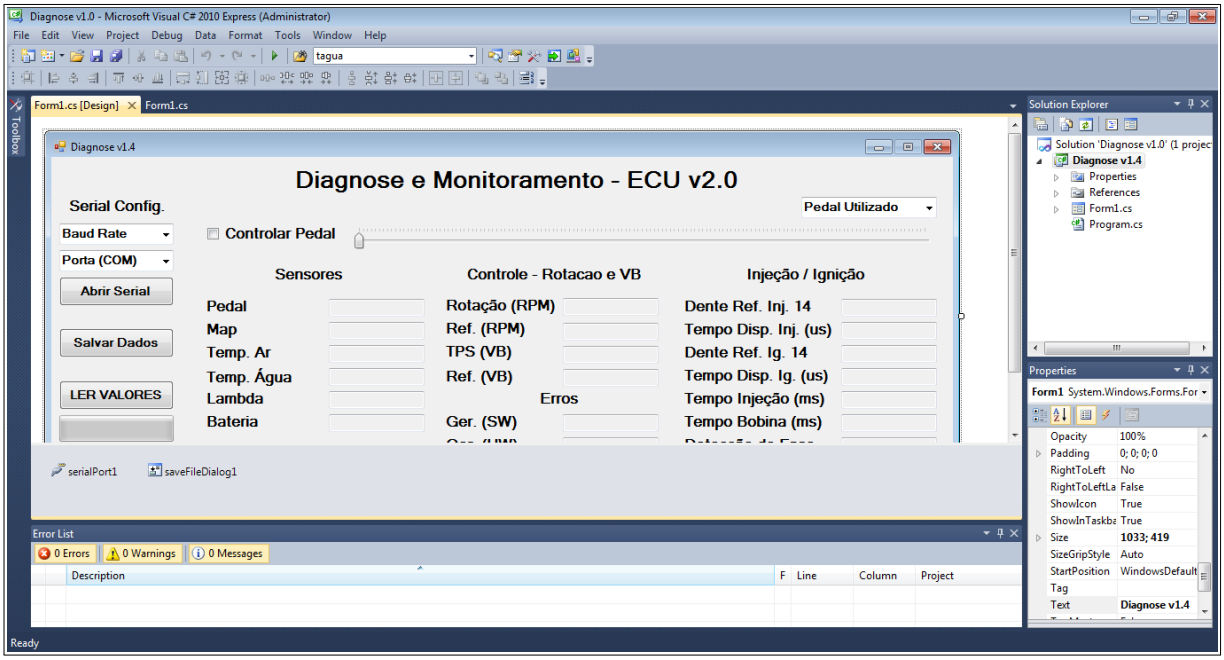
Figura 10 – Trecho de código retirado do uC de Gerenciamento



Fonte: o autor

2.3.5 A ferramenta Visual C# da Microsoft

Figura 11 – A ferramenta Visual C# da Microsoft



Fonte: o autor

O Visual C# é uma ferramenta fornecida pela Microsoft que permite a implementação da linguagem de programação C#[7]. Criada para o desenvolvimento de aplicações que executam sobre o .NET Framework, a linguagem C# é simples e poderosa, com orientação a objetos, o que permite o desenvolvimento rápido de aplicações robustas, mantendo sempre a elegância e a eficácia das linguagens baseadas no estilo C (MICROSOFT, 2013).

Com o Visual C# é possível criar aplicações completamente personalizadas, tanto do ponto de vista do código, como também do ponto de vista da aparência. Assim, o projetista pode utilizar bibliotecas prontas para acrescentar artifícios de controle, como

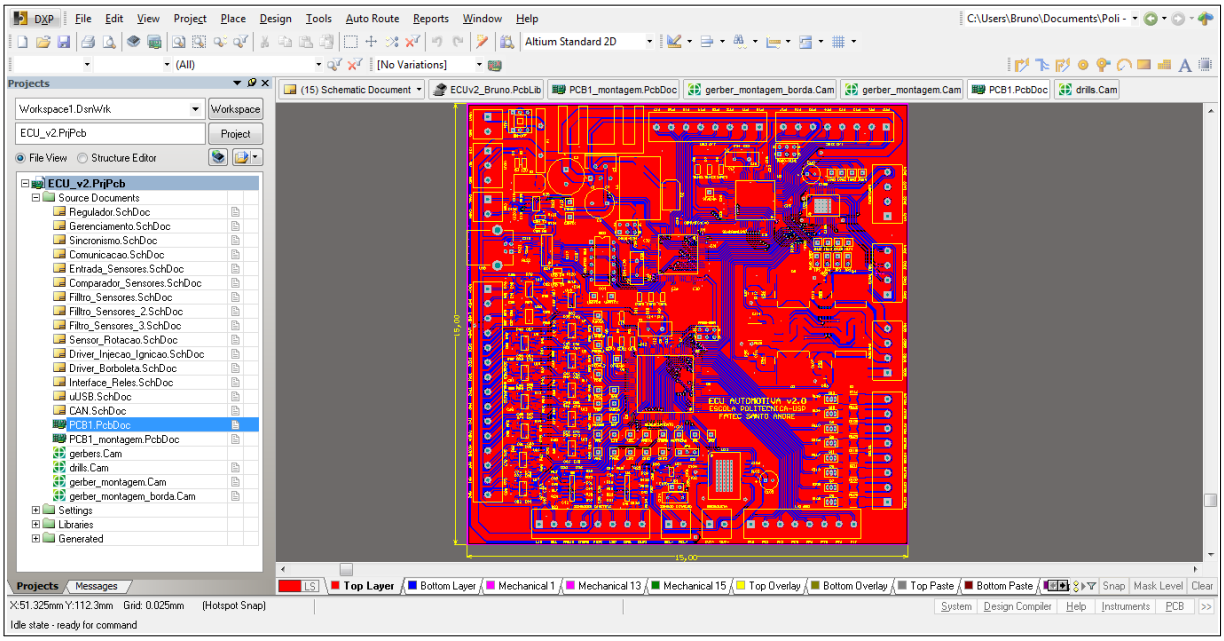
[7] Pronuncia-se "'C Sharp"'.

barras deslizantes, botões, caixas de texto ou campos de seleção, o que facilita o desenvolvimento do *software*. Além disto, o projetista pode alterar todos os parâmetros de um dado controle, utilizando-se para isto tanto a interface gráfica de design, quanto o código do programa.

Com a ferramenta também é possível desenvolver aplicações capazes de controlar a comunicação serial do computador, seguindo o padrão RS232, cujo protocolo será discutido posteriormente neste capítulo. Assim, para se utilizar este protocolo de comunicação, basta inserir na interface gráfica o objeto correspondente a porta serial e configurar os seus parâmetros, como *baud rate* (taxa de transmissão), bits de paridade, bits de parada, dentre outros. Uma vez inserido e configurado, o projetista pode utilizar comandos orientados ao objeto da porta serial, a fim de abrir ou fechar a conexão, enviar ou ler dados, dentre outros.

2.3.6 A ferramenta Altium Designer

Figura 12 – A ferramenta Altium Designer

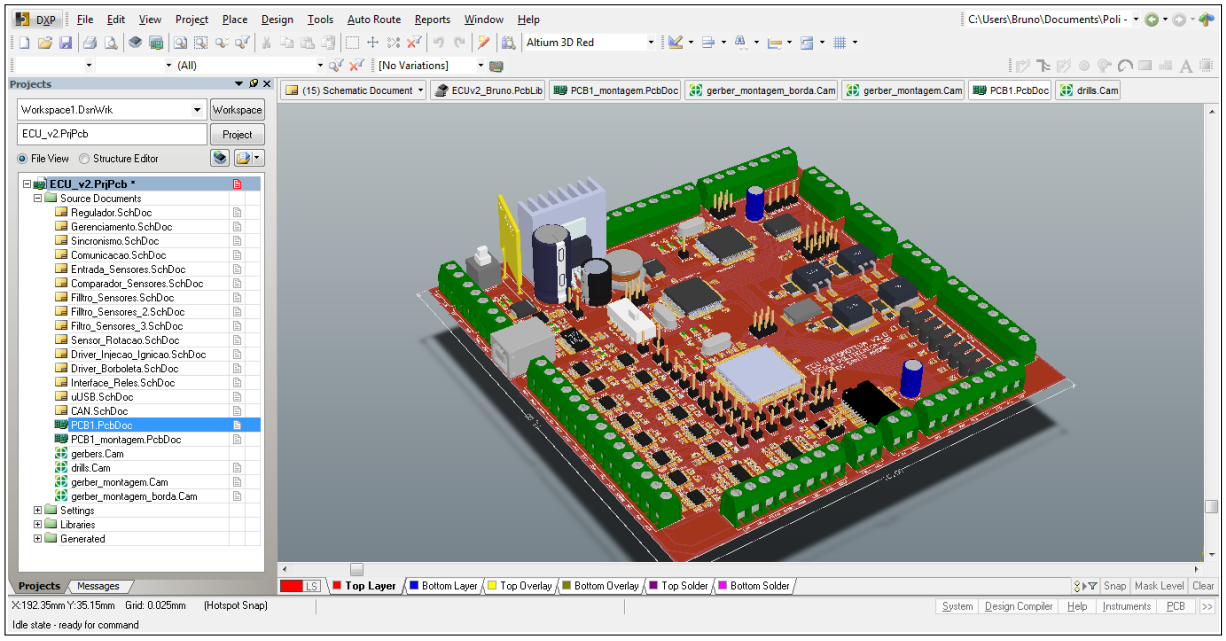


Fonte: o autor

O Altium Designer é uma ferramenta fornecida pela Altium para desenvolvimento de *hardware*, o que envolve planejamento e construção de esquema elétrico, simulação e desenho de layout da placa de circuito impresso. A ferramenta se mostrou bastante poderosa, contando com uma vasta biblioteca de componentes e opções de personalização do layout, o que agiliza muito o desenvolvimento do projeto do *hardware*. A ferramenta contém comandos para alterar espessura de várias trilhas ao mesmo tempo, opções para

se criar novos componentes, comandos para se alterar parâmetros de vários componentes ao mesmo tempo, dentre outros.

Figura 13 – Modelo 3D da ECU v2.0 projetada com a ferramenta Altium Designer



Fonte: o autor

Pela ferramenta é possível criar um modelo 3D da placa final a partir do layout de circuito impresso desenvolvido, o que fornece uma visualização de como será a placa montada. Assim, este recurso permite ao projetista prever problemas que surgiriam após a montagem do protótipo, como, por exemplo, a falta de espaço entre os componentes. Com a possibilidade de importação de desenhos desenvolvidos em outros *softwares*, é possível, para um projetista eletrônico, criar o modelo 3D de sua placa sem mesmo possuir habilidade para manusear CADs (CAD - Computer Aided Design) de desenhos mecânicos 3D.

O programa possui capacidade para realizar roteamento automático e posicionamento inteligente, porém para o presente projeto optou-se pelo roteamento e posicionamento realizados manualmente, com o objetivo de se alcançar melhores resultados[8].

2.3.7 Protocolos de Comunicação

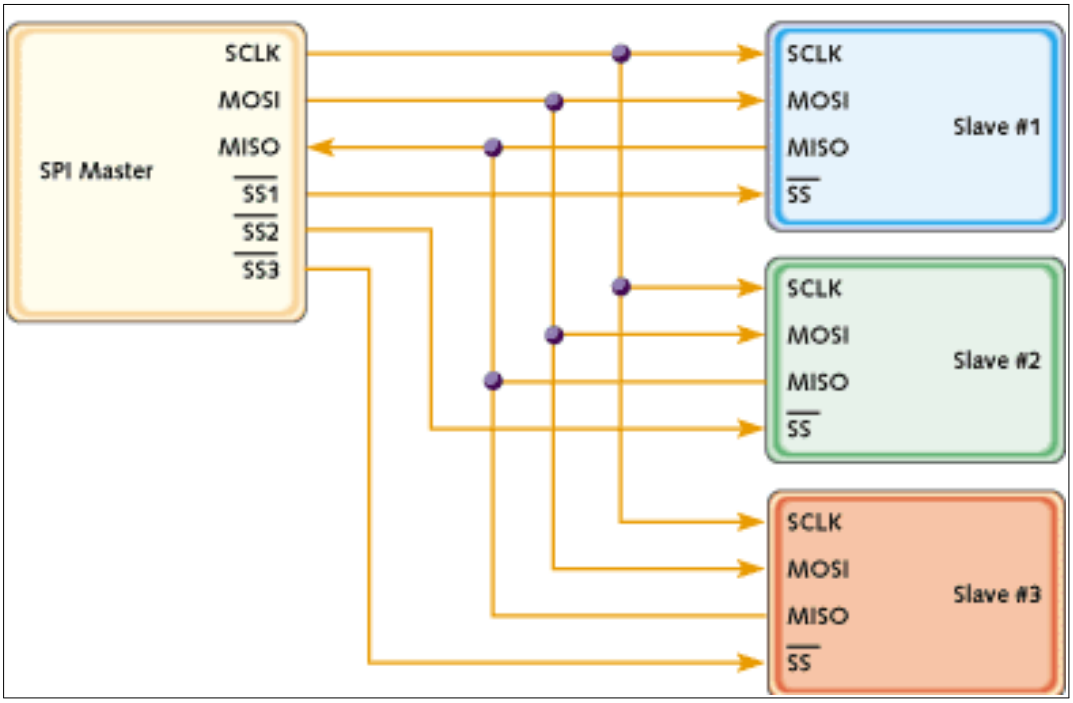
A unidade de gerenciamento eletrônico deste projeto foi desenvolvida, como explicado anteriormente, utilizando uma arquitetura descentralizada, o que exige a troca de dados entre as unidades de processamento. Para isto, se faz uso de três protocolos de comunicação: o protocolo SPI (*Serial Peripheral Interface*), que basicamente é responsável

[8] Pode-se, por exemplo, posicionar os componentes a fim de reduzir o tamanho da trilha, isolar circuitos de potência dos circuitos digitais, aumentar a espessura das trilhas de potência, dentre outros.

pela comunicação entre os microcontroladores do sistema, o protocolo UART (*Universal Asynchronous Receiver Transmitter*), responsável pela comunicação entre o microcontrolador do bloco Comunicação e a aplicação de monitoramento rodando em um computador externo, e o protocolo CAN, responsável pela comunicação da ECU v2.0 deste projeto com as demais ECUs presentes no veículo.

2.3.7.1 O protocolo SPI

Figura 14 – Esquema de ligações de uma rede de comunicação SPI



Fonte: Kalinsky e Kalinsky (2002)

Consiste de um protocolo serial síncrono para comunicação ponto-a-ponto. Basicamente é composto de três sinais: entrada de dados (*data in*), saída de dados (*data out*) e *clock*. O protocolo se diz síncrono, pois é necessário que os dois pontos (transmissor e receptor) estejam conectados com o mesmo *clock*, para que as operações ocorram no mesmo instante e estejam, portanto, sincronizadas. Além disto, o protocolo é dito *full duplex*, pois ao mesmo tempo que se transmite um dado é possível se receber outro dado[9] (FREESCALE, 2012).

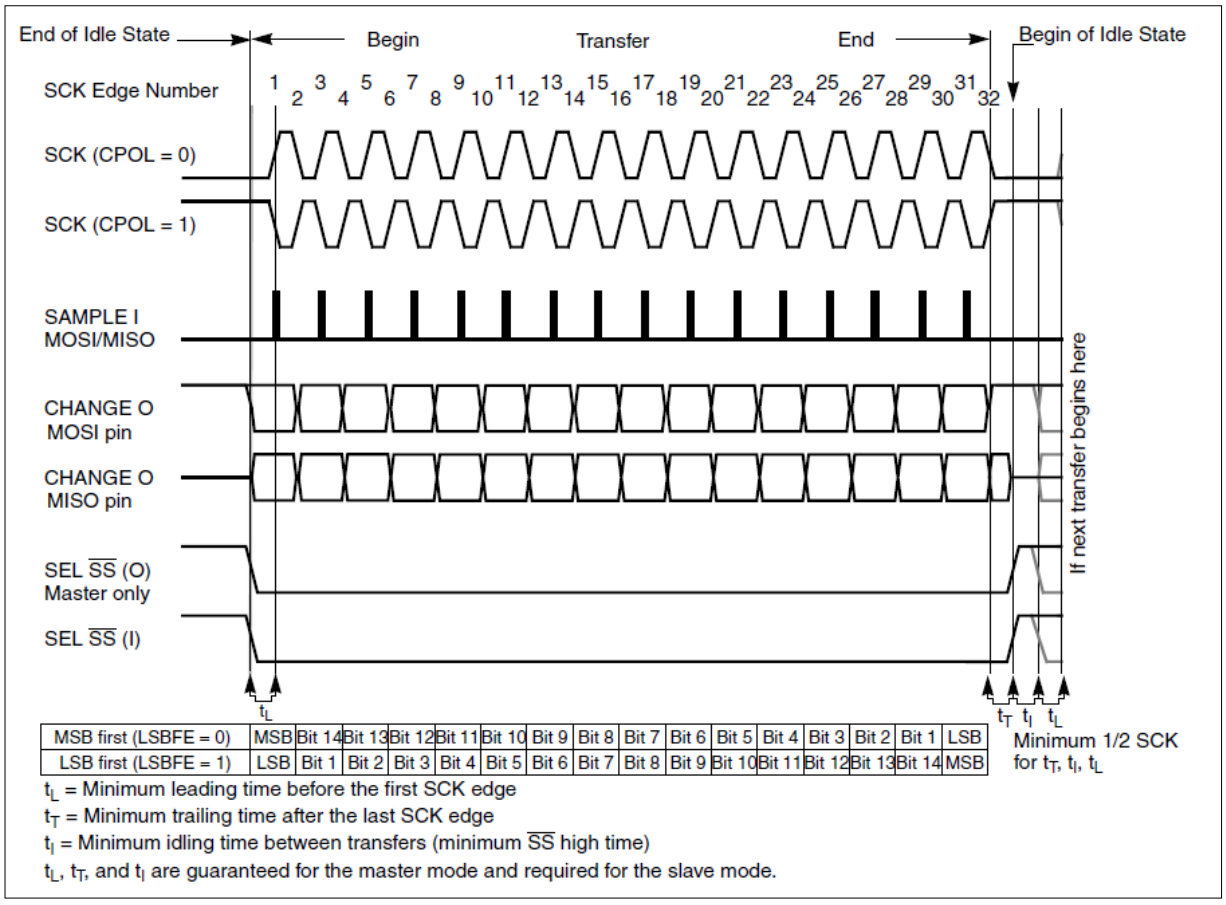
Como dito anteriormente, a comunicação é ponto-a-ponto, na qual um corresponde ao mestre e o outro, ao escravo. O mestre é designado ao dispositivo que controla a geração do sinal de *clock*. Assim, a comunicação ocorrerá no instante em que o mestre necessitar do dado, diferentemente do escravo, que deverá esperar pelo sinal de *clock* do mestre

[9] Neste caso, a transmissão e recepção são feitos por canais físicos diferentes.

para transmitir e/ou receber um dado. Além disto, é possível se estruturar uma rede de comunicação SPI, com um mestre e vários escravos, sendo que o dispositivo mestre sempre é fixo[10]. Para o caso de vários escravos, o mestre deverá utilizar sinais adicionais (designados por *slave select*) para selecionar o escravo com que deseja se comunicar em um dado instante de tempo.

O protocolo SPI é bastante simples e, por este motivo, consegue alcançar taxas elevadas de transmissão, fato essencial para o projeto em questão. No caso do S12X, o SPI pode alcançar até 12,5 Mbits/s, e pode-se transmitir e receber, simultaneamente, dados de 16 bits.

Figura 15 – Carta de tempos da comunicação SPI no S12X



Fonte: Freescale (2012)

Abaixo segue uma breve descrição dos sinais do protocolo, com base na figura anterior:

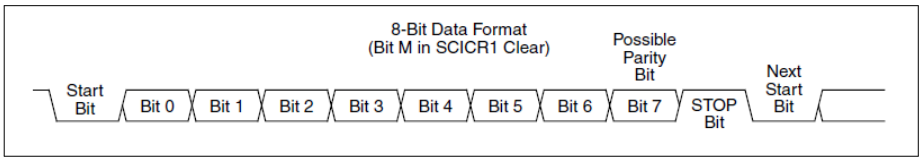
- SCK: Sinal de *clock*, com dois formatos diferentes (de acordo com o bit CPOL);

[10] A designação do mestre é dependente da conexão utilizada no *hardware*.

- MISO: Entrada de dados para mestre ou saída de dados para escravo (nota-se que a transmissão/recepção ocorre na borda de número impar do *clock*, enquanto que a mudança de dados ocorre na borda de número par);
- MOSI: Entrada de dados para escravo ou saída de dados para mestre (a observação do item anterior também é valida para este sinal);
- SS: Sinal ativo baixo de seleção do escravo.

2.3.7.2 O protocolo UART

Figura 16 – Formato do protocolo de comunicação UART



Fonte: Freescale (2012)

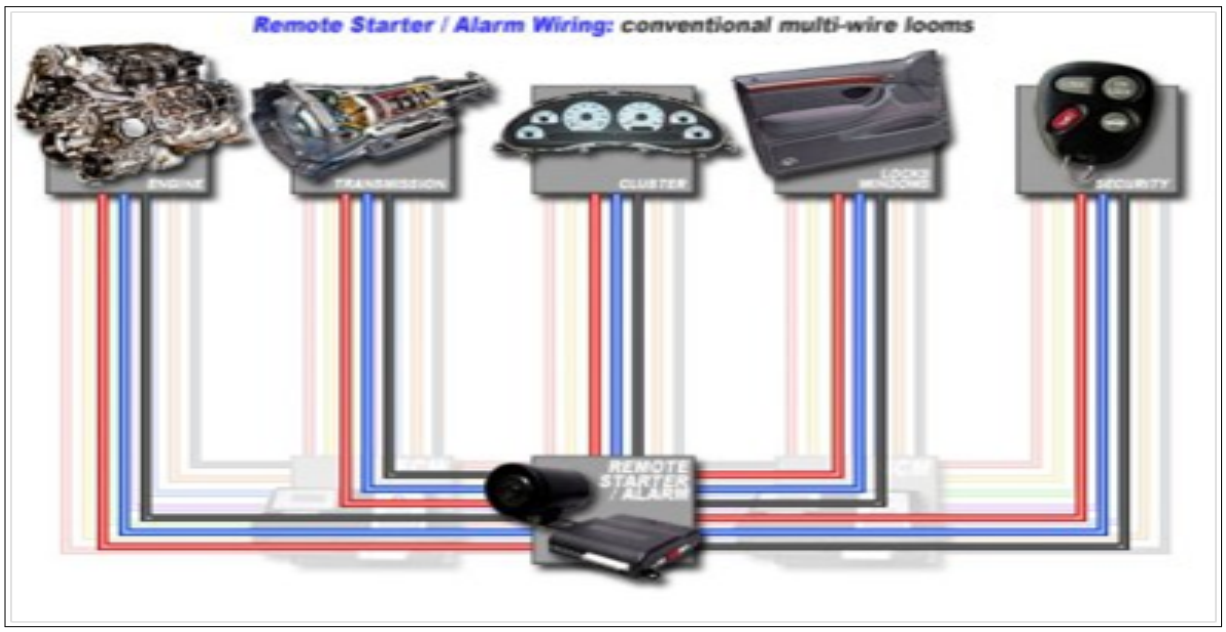
Analogamente ao SPI, este protocolo é serial ponto-a-ponto, porém não há necessidade da designação de mestre ou escravo. Além disto, a comunicação ocorre sempre entre dois dispositivos, fato que não pode ser alterado a *posteriori* devido às conexões do *hardware*. Sendo assim, em um dado instante de tempo qualquer dispositivo pode transmitir para o outro, sendo que a comunicação pode ser *half duplex*[11] ou *full duplex* (FREESCALE, 2012).

No caso deste projeto, o protocolo serial segue o padrão RS232, podendo alcançar taxas moderadas de transmissão. Utiliza-se no projeto uma taxa de transmissão (*baud rate*) de 125000 b/s, o que é suficiente para que a troca de dados do sistema com o *software* de monitoramento ocorra sem perdas relevantes. É importante destacar que é possível, com o protocolo serial, alcançar taxas de transmissão maiores, porém a probabilidade de ocorrência de erro (e, consequentemente, perda de dados) aumenta consideravelmente nestes casos. Por esta razão, o protocolo SPI foi escolhido para operar entre os microcontroladores do sistema, deixando o protocolo UART apenas para a comunicação externa com o computador (dado que neste último caso a velocidade de transmissão não precisa ser muito elevada).

[11] Neste caso não é possível transmitir e receber dados simultaneamente.

2.3.7.3 O protocolo CAN

Figura 17 – Exemplo de CAN BUS



Fonte: SCARPINETTI e SOARES (2012)

Desenvolvido pela empresa alemã Bosch, o CAN corresponde a um protocolo serial desenvolvido especialmente para interligar sensores e atuadores presentes em subsistemas eletrônicos (ECUs) de um carro (BOSCH, 2005). Atualmente, devido à sua confiabilidade e robustez, o protocolo não se restringe apenas ao ramo automotivo, podendo ser empregado também em aplicações domésticas ou industriais (FREESCALE, 2012). Apesar de simples e de baixo custo, o protocolo se mostra altamente confiável para aplicações automotivas, atendendo normas de padronização como a ISO 11898 (BOSCH, 2013e).

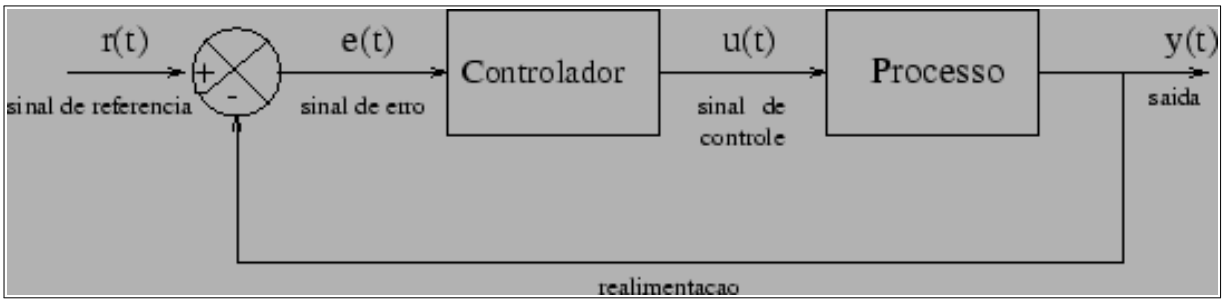
Dentre as características do protocolo destacam-se a sua capacidade para comportar vários mestres, interligação de diversos dispositivos com apenas dois fios, capacidade de se arbitrar o acesso simultâneo ao barramento CAN adotando níveis de prioridade para as mensagens, e a taxa de transmissão de até 1 Mb/s.

No âmbito deste projeto, o protocolo CAN foi utilizado para se comunicar com o painel do motorista, enviando do valor da rotação calculada e realizando a leitura da velocidade do veículo[12].

[12] A velocidade é determinada por outra ECU presente no veículo, sendo o seu valor disponibilizado no barramento CAN.

2.3.8 Controladores do tipo proporcional-mais-integral (PI)

Figura 18 – Controle em malha fechada



Fonte: Silva e Bazanella (2000)

Segundo Ogata (2003), um controlador industrial é responsável por comparar o valor real da saída do processo com o valor desejado, calcular o erro associado (diferença entre o valor desejado e o valor de saída real) e atuar no processo industrial, de forma a anular o erro ou minimizá-lo a um valor suficientemente pequeno.

Existem diversos tipos de controladores, sendo que se destaca o controlador PI para o presente projeto. Neste tipo de controlador, o sinal de controle (sinal de atuação no processo) é proporcional e integral, e a relação entre a saída do controlador $u(t)$ e a entrada $e(t)$ (sinal que corresponde ao erro) é dada pela expressão abaixo (OGATA, 2003):

$$u(t) = K_p \times e(t) + K_i \times \int e(t)dt$$

onde K_p corresponde ao ganho proporcional e K_i corresponde ao ganho integral. O ganho proporcional K_p, quando elevado, acaba diminuindo o erro $e(t)$, porém não o anula completamente em regime. Além disto, valores muito elevados desta parcela pode acarretar no surgimento de respostas oscilatórias, o que pode ser crítico dependendo do processo controlado.

De acordo com Ogata (2003), a inclusão da natureza integral no controlador proporcional tem a característica de fornecer uma sinal de controle não nulo, mesmo quando o sinal de erro for anulado. Isto se deve basicamente ao fato da saída do controlador depender, neste caso, de valores passados do erro e não apenas do valor atual. Com isto, distúrbios podem ser rejeitados mesmo em situações de erro nulo, o que não acontece com controladores puramente proporcionais (CRUZ, 2009). Assim, o principal objetivo do termo integral é a de anular erros estacionários, fato essencial para o sucesso deste projeto. Com relação ao ganho integral, valores altos ajudam a anular completamente o erro de regime, porém neste caso a resposta (saída do processo) pode se tornar oscilatória, podendo provocar a instabilização do sistema (SILVA; BAZANELLA, 2000).

Tabela 1 – Efeitos ao se alterar as constantes do PI (Adaptado de Ang, Chong e Li (2005))

Constante Incrementada	**Tempo de Subida**	**Sobressinal**	**Tempo de Estabilização**	**Erro Estacionário**	**Estabilidade**
K_P	Diminui	Aumenta	Aumenta Pouco	Diminui	Diminui
K_I	Aumenta Pouco	Aumenta	Aumenta	Diminui Muito	Diminui

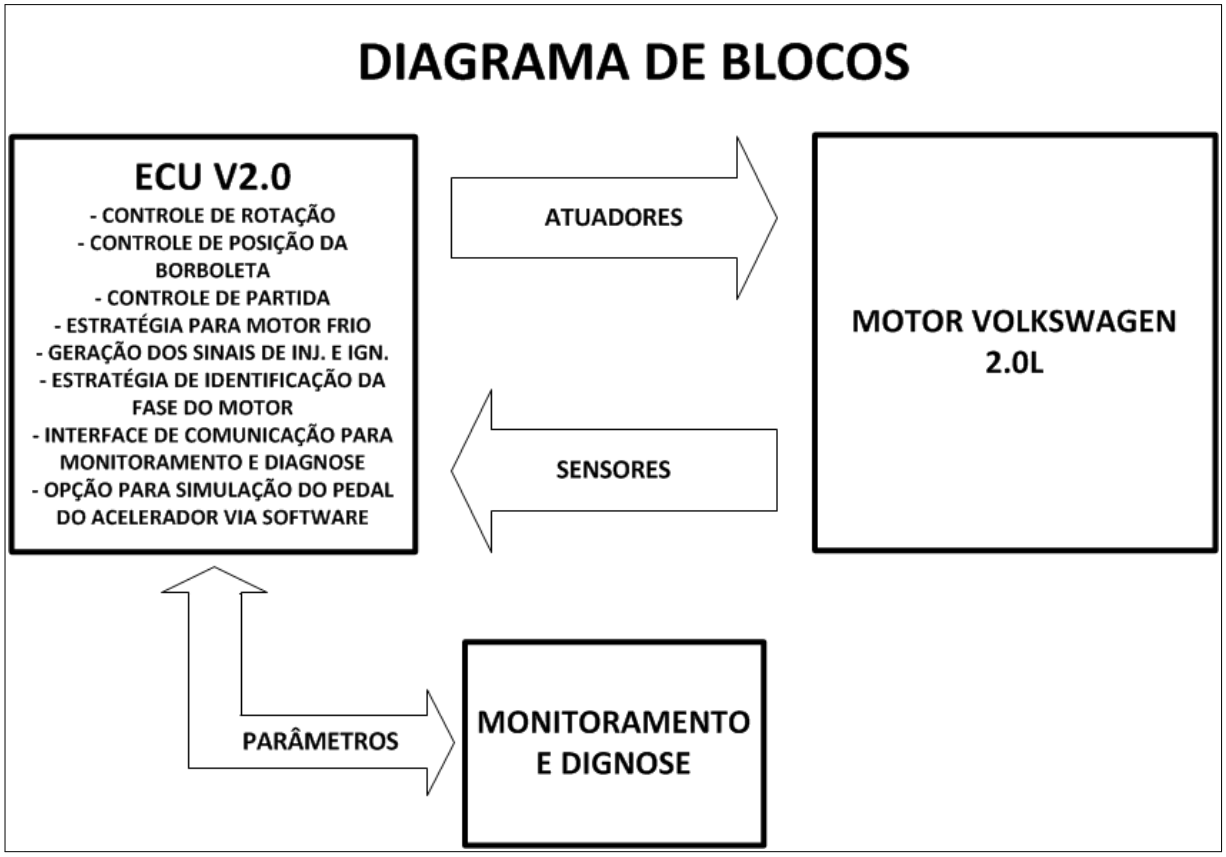
Para a sintonia do PI foi escolhido o método manual, que consiste na determinação empírica dos ganhos do controlador, com base na resposta observada. Dentro os principais motivos para se adotar este método se destaca a complexidade em se obter um modelo matemático linear para o motor a combustão interna (WONG; VONG; IP, 2010), o que levaria muito tempo para ser feito. Existem outros métodos para ajuste de controladores PI sem o conhecimento da função de transferência do processo, porém neste caso seria necessário, por exemplo, forçar uma resposta oscilatória no sistema, como sugere o método do período crítico de Ziegler-Nichols (CRUZ, 2009), o que não é seguro para um motor a combustão interna. As mesmas justificativas se aplicam ao controle de posição da válvula borboleta, que também foi sintonizado empiricamente. Detalhes relacionados à implementação destes controles serão apresentados no capítulo 3 deste trabalho.

3 Metodologia

O Projeto Otto II propõe, como explicado anteriormente, desenvolver uma unidade de gerenciamento eletrônico para um motor a combustão interna, unidade esta constituída de *hardware*, *firmware* e *software*. Além disto, o sistema será projetado com microcontroladores automotivos Freescale da família S12XE (projetados especialmente para gerenciamento de motores), além de interfaces automotivas necessárias para condicionar os sinais de atuação. Neste capítulo, será abordada a metodologia adotada no projeto para o desenvolvimento do sistema, o que envolve a descrição do *hardware*, a estratégia adotada no *firmware* e uma explicação do funcionamento do *software* de monitoramento.

3.1 Projeto Conceitual

Figura 19 – Diagrama de blocos do projeto conceitual



Fonte: o autor

De modo geral, a unidade de gerenciamento eletrônico deste projeto (denominada ECU v2.0) é capaz de ler e processar de forma rápida e eficaz os sinais dos sensores presentes no motor, de maneira a gerar corretamente os sinais de comando dos atuadores. Além disto, a unidade inclui uma interface de comunicação com finalidade de monitoramento e diagnose do sistema, sendo que também é possível pela mesma interface atuar no sistema, através da simulação do sensor do pedal de aceleração do veículo.

3.1.1 Sensores

Em grande parte dos sistemas eletrônicos se faz necessária a medição de grandezas físicas tais como pressão e temperatura. Os sensores correspondem aos elementos encarregados de medirem tais grandezas. Assim, quando a grandeza física sofre uma alteração, ocorre uma alteração no comportamento elétrico do material com que o sensor é construído[1]. Consequentemente, tem-se um sinal elétrico que varia de acordo com a grandeza medida.

A seguir serão abordados os sensores essenciais para o controle do motor deste projeto.

3.1.1.1 Sonda Lambda

Figura 20 – Sonda Lambda tipo LSM 11

Fonte: Bosch (2013d)

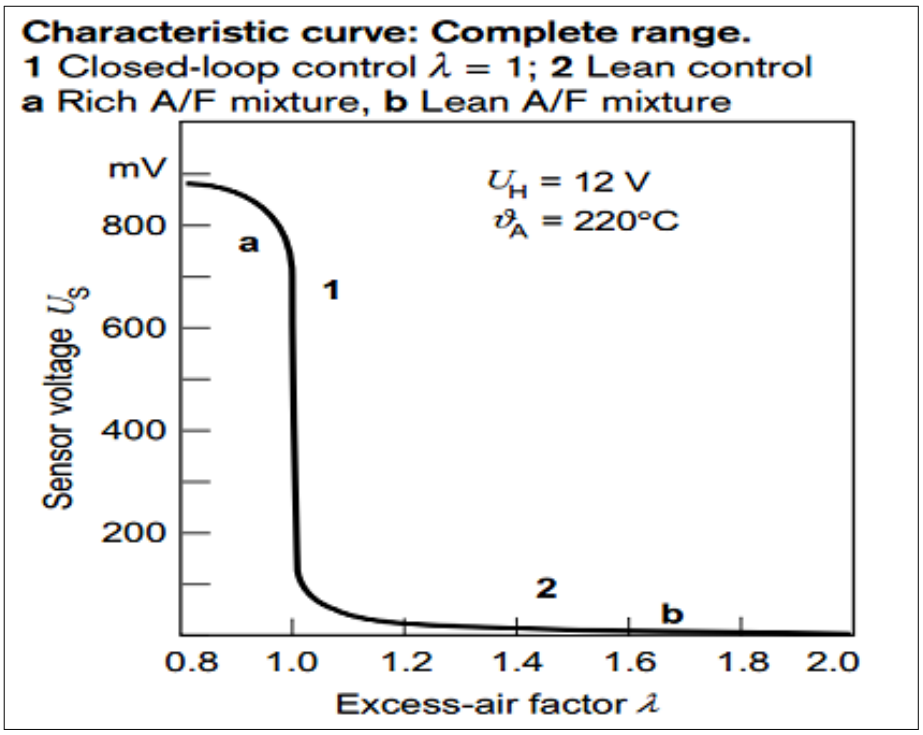
A sonda lambda é o sensor responsável pelo controle em malha fechada da mistura a/c no motor. Este sensor é capaz de medir a concentração de oxigênio nos gases resultantes da combustão (BOSCH, 2013d), informando se a mistura a/c está rica[2] ou pobre[3].

[1] Esta alteração pode ser, por exemplo, uma mudança na capacitância ou na resistência elétrica do material.

[2] Mistura rica equivale a um fator lambda (λ) menor que 1, o que indica maior quantidade de combustível em relação à quantidade de ar.

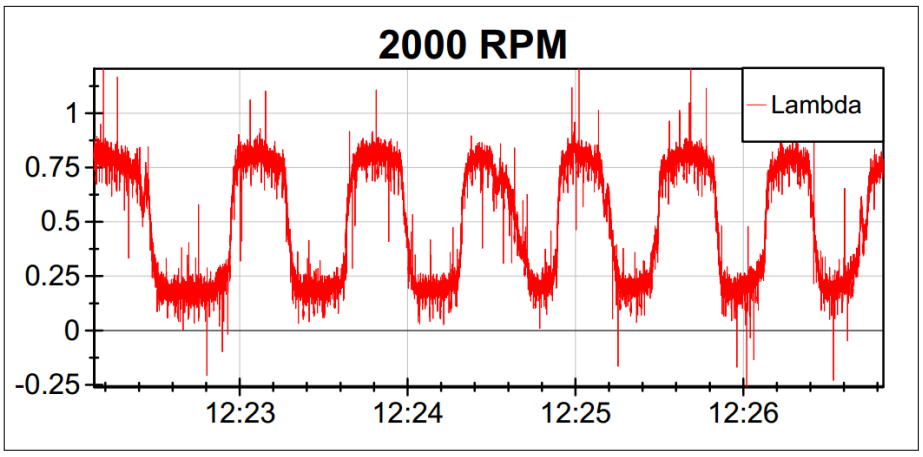
[3] Mistura pobre equivale a um fator lambda (λ) maior que 1, o que indica maior quantidade de ar em relação à quantidade de combustível.

Figura 21 – Curva característica da sonda lambda



Fonte: Bosch (2013d)

Figura 22 – Controle em malha fechada do fator lambda



Fonte: Escola Politécnica da USP; Fatec Santo André (2013)

A sonda lambda tem como finalidade fornecer uma realimentação ao controle estequiométrico da mistura a/c, principalmente quando o motor se encontra próximo de sua rotação de referência[4], a fim de reduzir as emissões de poluentes (vide figura a seguir). Já quando o motor se encontra em aceleração, a massa de combustível injetada é maior do que a quantidade estequiométrica, tornando a mistura rica (neste caso o sensor apresenta

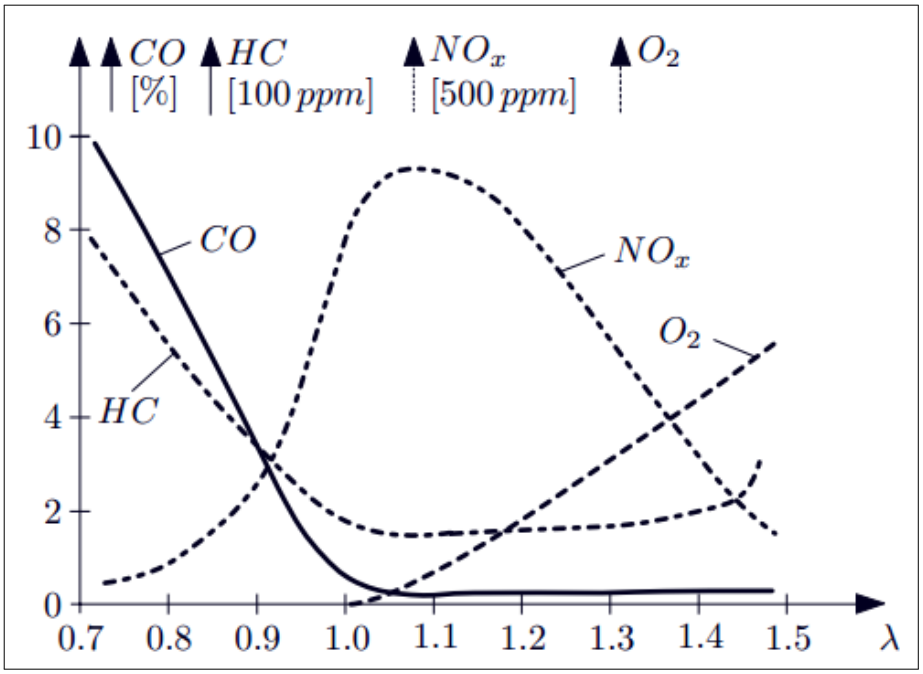
[4] O que indica que não existe aceleração nem desaceleração.

sinal constante em nível lógico alto), e a ECU deixa de realizar o controle estequiométrico em malha fechada até que a rotação do motor alcance o seu valor de referência. O mesmo procedimento ocorre na desaceleração, porém neste caso a mistura fica pobre devido a uma diminuição ou corte do combustível injetado. Vale mencionar que quando a temperatura do motor está abaixo de seu ponto de operação (em torno de 90°C), a ECU adota a estratégia de enriquecer a mistura ($\lambda < 1$), a fim de acelerar o aquecimento do motor e mantê-lo funcionando de forma estável.

Como dito anteriormente, o controle da estequiometria é essencial para se minimizar emissões de poluentes e proporcionar economia de combustível. Segundo Kiencke e Nielsen (2005), pode-se estabelecer as seguintes considerações em relação à faixa de valor em que se encontra o fator lambda (λ):

- $\lambda < 1$: Emissão elevada de hidrocarbonetos (HC) e monóxido de carbono (CO). Também entrega alta potência devido à alta quantidade de combustível injetada;
- $\lambda = 1$: Reagentes da combustão em proporção estequiométrica, com emissões de poluentes minimizadas (principalmente com uso de um catalisador para tratar os gases de escape). Além disto, fornece uma razoável potência;
- $\lambda > 1$: Apresenta boa eficiência devido à maior massa de ar admitida pelo motor. Todavia ocorre aumento considerável da emissão de óxidos nitrosos (NOx), o que pode comprometer a vida útil do catalisador do veículo.

Figura 23 – Emissão de poluentes x fator lambda



Fonte: Kiencke e Nielsen (2005)

Especificamente neste trabalho, a sonda lambda não foi utilizada, uma vez que, por simplificação, foi adotado o controle estequiométrico em malha aberta. Maiores detalhes serão apresentados na seção de *firmware*.

3.1.1.2 Sensor MAP

Figura 24 – Sensor MAP

Fonte: Bosch (2013b)

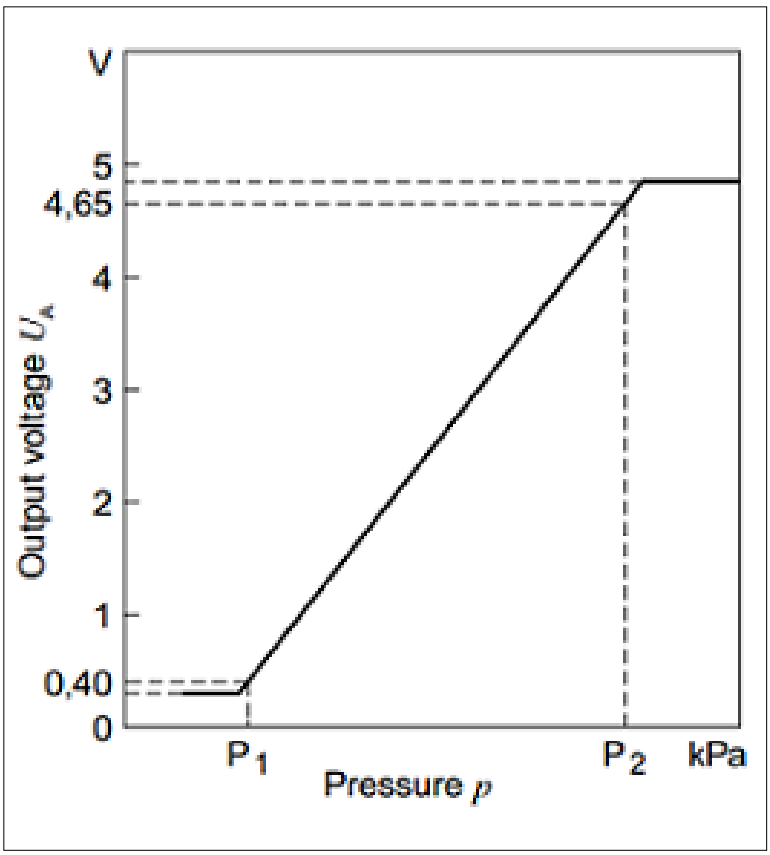
O sensor MAP (*Manifold Absolute Pressure*) é responsável por medir a pressão da massa de ar admitida, medindo-a no coletor de admissão do motor (BOSCH, 2013b). Um vantagem deste sensor é que ele mede uma pressão absoluta, ou seja, a pressão medida não é relativa à pressão atmosférica, o que descarta uso de calibrações para estimar a pressão ambiente. Com isto, a ECU é capaz de calcular corretamente a massa de combustível a ser injetada independentemente do ambiente em que se encontra o carro.

Normalmente, o sensor MAP é utilizado para se estimar a massa de ar admitida pelo motor, de modo que se possa calcular a quantidade de combustível necessária para a estequiometria da mistura ar-combustível (ou seja, evitar excesso de ar ou excesso de combustível na combustão).

Tipicamente, o sensor MAP fornece em sua saída um valor de tensão que é proporcional ao valor de pressão do ar admitido. A figura a seguir ilustra uma curva característica de um sensor MAP da Bosch. Para a unidade eletrônica gerenciadora do motor, é de extrema importância o conhecimento da curva e de seus valores críticos, dado que para estimar a massa de ar admitida pelo motor a unidade gerenciadora deste projeto neces-

sita do valor do sensor em unidade de pressão (como kPa, por exemplo), que por sua vez deve ser estimado com base no valor de tensão lido.

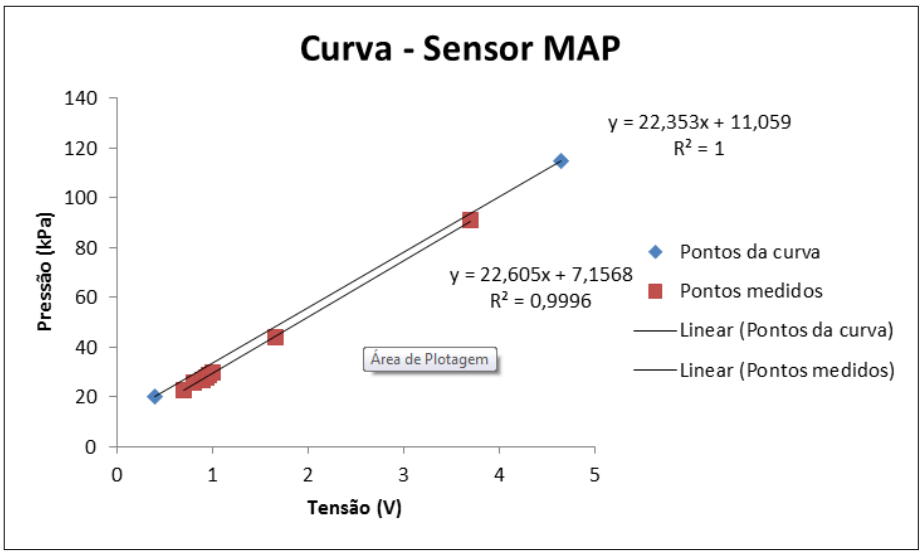
Figura 25 – Curva característica do sensor MAP



Fonte: Bosch (2013b)

Informações técnicas como curvas características e valores críticos de sensores automotivos nem sempre são fáceis de se conseguir. Muitos fabricantes omitem tais informações, apenas fornecendo algumas informações técnicas de instalação. Nestas ocasiões é necessário utilizar o aparelho de diagnóstico do automóvel em conjunto com a sua ECU original para se conseguir medir experimentalmente os pontos da curva. Com um número adequado de pontos, é possível realizar uma interpolação, a fim de se obter uma expressão matemática que relacione a grandeza física (pressão ou temperatura) com o sinal elétrico medido, e que se aproxime da curva característica do dispositivo. Especificamente neste projeto, adotou-se a interpolação da figura seguinte devido às dificuldades em se obter os valores críticos da curva característica do MAP, dada a omissão de dados técnicos pelo fabricante.

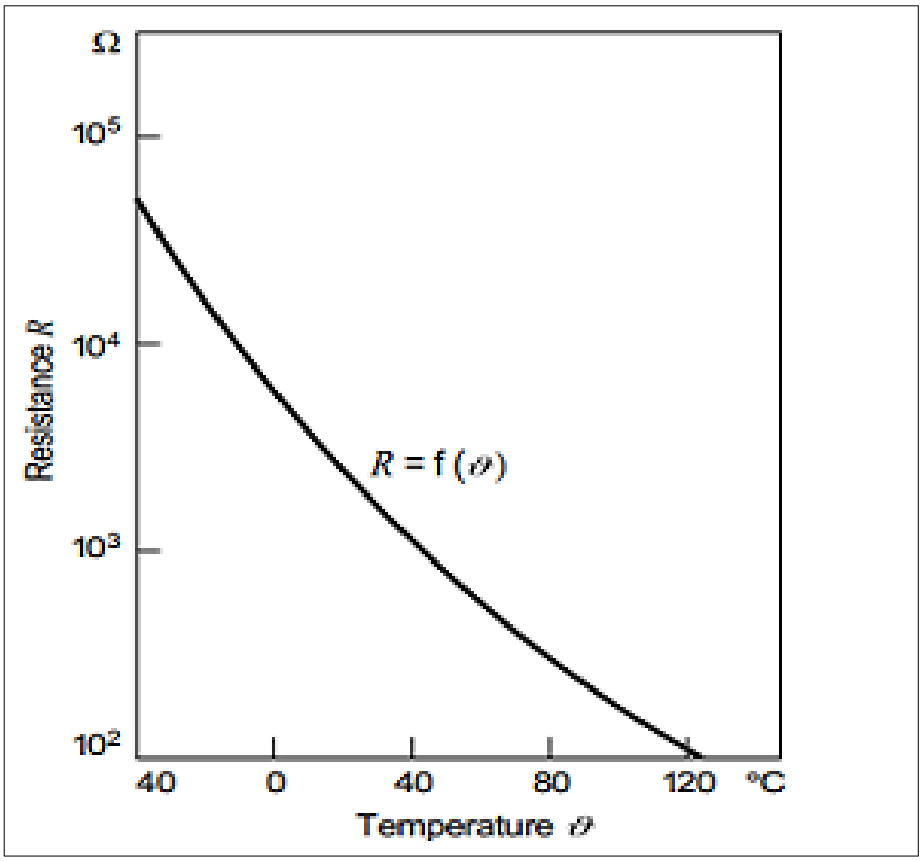
Figura 26 – Interpolação da curva característica do sensor MAP



Fonte: o autor

3.1.1.3 Sensor de temperatura do ar

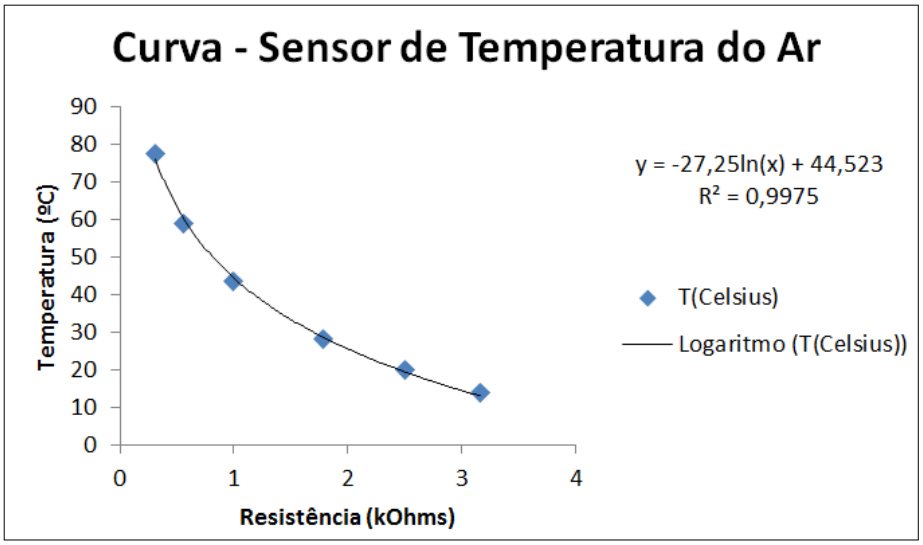
Figura 27 – Curva característica do sensor de temperatura do ar



Fonte: Bosch (2013b)

O sensor de temperatura do ar consiste de um resistor cujo valor de sua resistência varia de acordo com a temperatura. No caso deste sensor, o coeficiente de variação é negativo (*Negative Temperature Coefficient* - NTC), o que significa que o valor da resistência do sensor diminui com o aumento da temperatura (MANAVELLA, 1996). É importante notar que a relação entre a temperatura e a resistência não é linear, o que faz surgir a necessidade de se interpolar pontos da curva característica do sensor, a fim de se obter uma expressão matemática que relacione valor de resistência com valor de temperatura.

Figura 28 – Interpolação da curva característica do sensor de temperatura do ar

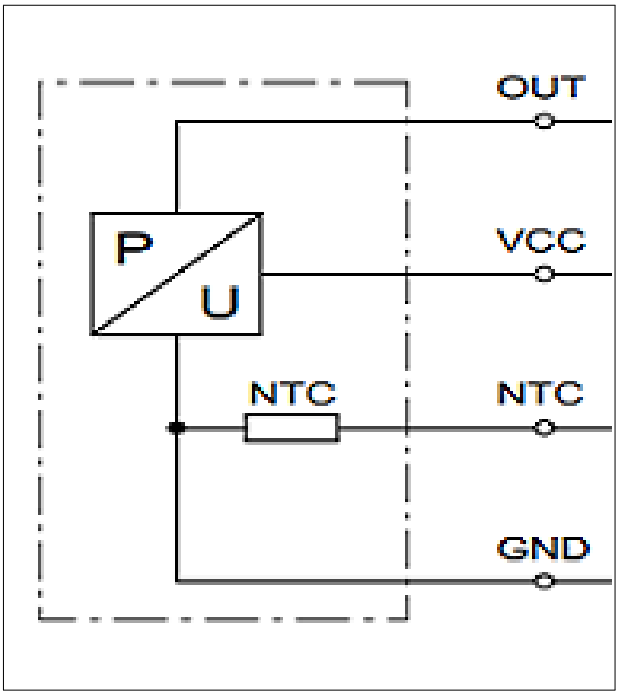


Fonte: o autor

Para se obter a informação do sensor é necessário utilizar, dentre outras opções, um circuito divisor de tensão, de modo que se possa calcular a resistência do sensor com base no valor de tensão lida do divisor resistivo. Além do divisor, utiliza-se também um circuito condicionador para filtrar o sinal, o que será discutido posteriormente com mais detalhes.

O sensor de temperatura do ar, assim como o sensor de pressão, é muito importante para se estimar a massa de ar admitida pelo motor. Em alguns motores, como no caso deste projeto, o sensor de temperatura do ar já vem integrado ao sensor de pressão MAP.

Figura 29 – Diagrama do MAP com o sensor de temperatura integrado



Fonte: Bosch (2013b)

3.1.1.4 Sensor de temperatura da água

Figura 30 – Sensor de temperatura da água

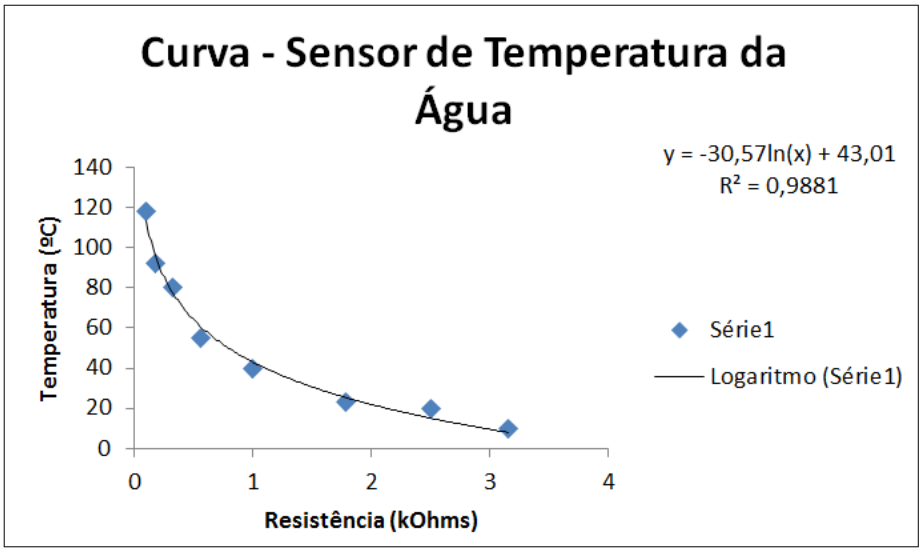
Fonte: Bosch (2013d)

Analogamente ao sensor de temperatura do ar, este sensor consiste também de um resistor NTC. Assim, quanto maior a temperatura da água, menor será o valor da resistência do sensor e, consequentemente, menor será o valor da tensão lida do divisor de tensão resistivo.

A curva característica, semelhantemente à curva do sensor de temperatura do ar, não é linear, o que se faz necessário o uso de uma interpolação de pontos da curva para

se obter uma expressão matemática.

Figura 31 – Interpolação da curva característica do sensor de temperatura da água



Fonte: o autor

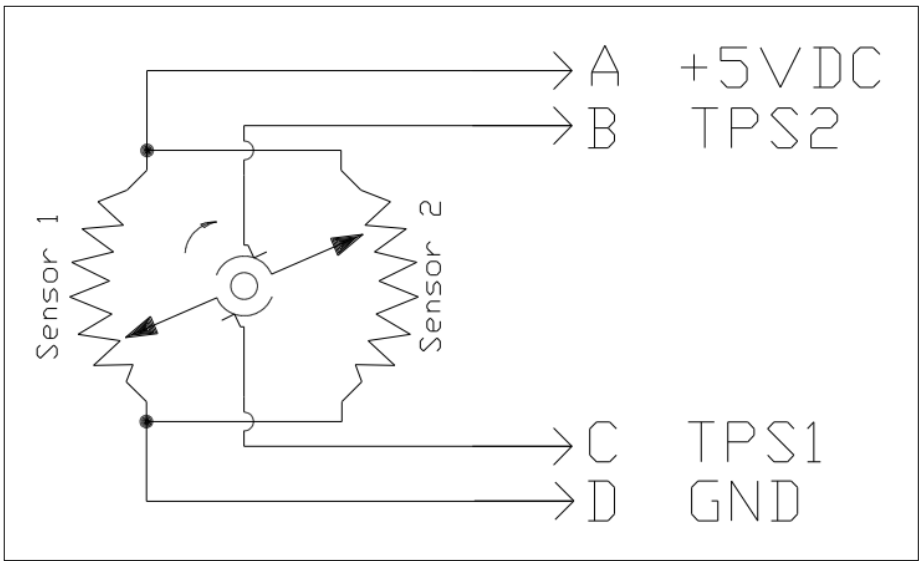
O sensor de temperatura da água é importante para informar se o motor se encontra em sua temperatura de operação. A ECU usa a informação proveniente do sensor para decidir se aplica ou não a estratégia de motor frio, que basicamente consiste em se enriquecer a mistura a/c e avançar o ponto de ignição. Em alguns motores, a ECU ainda pode acionar a ventoinha para que a temperatura do motor não exceda aproximadamente 90ºC[5].

3.1.1.5 Sensor de posição da válvula borboleta - TPS

Acoplado ao eixo da VB, o sensor de posição da válvula borboleta (TPS - *Throttle Position Sensor*) é constituído de um potenciômetro cujo valor de seu divisor resistivo varia de acordo com a posição de abertura da borboleta. Geralmente, por motivos de segurança, existem dois potenciômetros (TPS1 e TPS2) acoplados ao eixo da borboleta, porém suas leituras são complementares entre si, ou seja, quando se abre a borboleta ocorre aumento da tensão do divisor de um TPS e, ao mesmo tempo, diminuição da tensão do outro TPS (DELPHI, 2003). Tal implementação é feita para proporcionar redundância ao sistema, de modo que seja possível diagnosticar uma falha no sensor de posição da borboleta (ALBALADEJO, 2013). Caso, por exemplo, ocorra rompimento do cabo ou funcionamento incorreto de um dos sensores (TPS1 ou TPS2), o sistema pode adotar alguma estratégia como entrar em modo de segurança e/ou acender luzes indicadoras de falhas no painel, avisando ao motorista da necessidade de reparos no veículo.

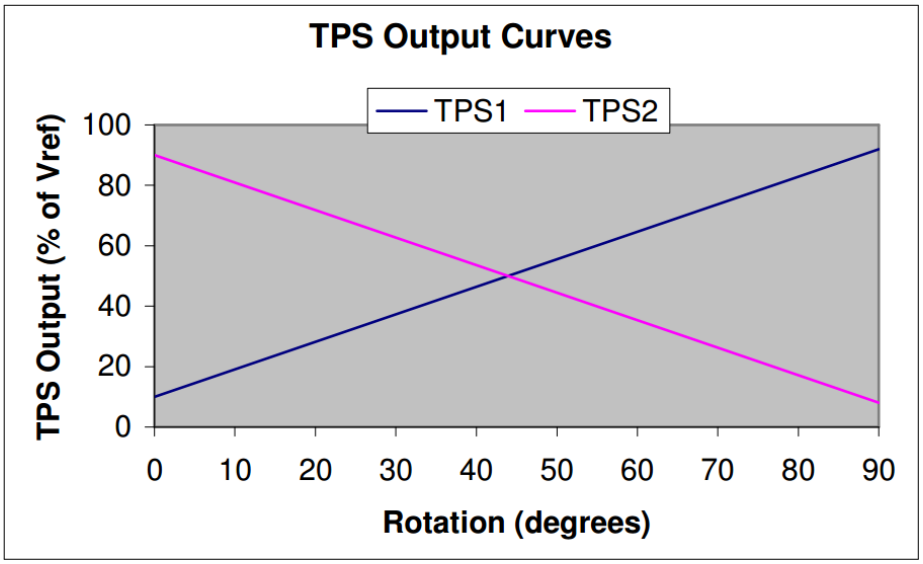
[5] Em alguns motores, como é o caso deste projeto, o acionamento da ventoinha é realizada mecanicamente através de um interruptor térmico, conhecido popularmente como "'cebolão"' (MTE-THOMSON, 2013).

Figura 32 – Diagrama do sensor TPS da válvula borboleta



Fonte: Delphi (2003)

Figura 33 – Curva da variação do TPS em função da posição da VB



Fonte: Delphi (2003)

O sensor TPS é extremamente importante para o controle do motor. Ele é responsável por fornecer a realimentação ao controle de posição da válvula borboleta, controle este realizado em malha fechada. Sendo assim, estratégias para filtrar os valores lidos ou verificar se a leitura do sensor está dentro de uma faixa correta são importantes e devem ser consideradas pela ECU.

3.1.1.6 Sensor de posição do pedal de aceleração

Analogamente ao sensor TPS, o sensor do pedal é constituído de dois potenciômetros que variam linearmente com a posição do pedal. Assim, quando o pedal for totalmente pressionado, um potenciômetro fornecerá leitura máxima de tensão, enquanto que o outro fornecerá leitura mínima.

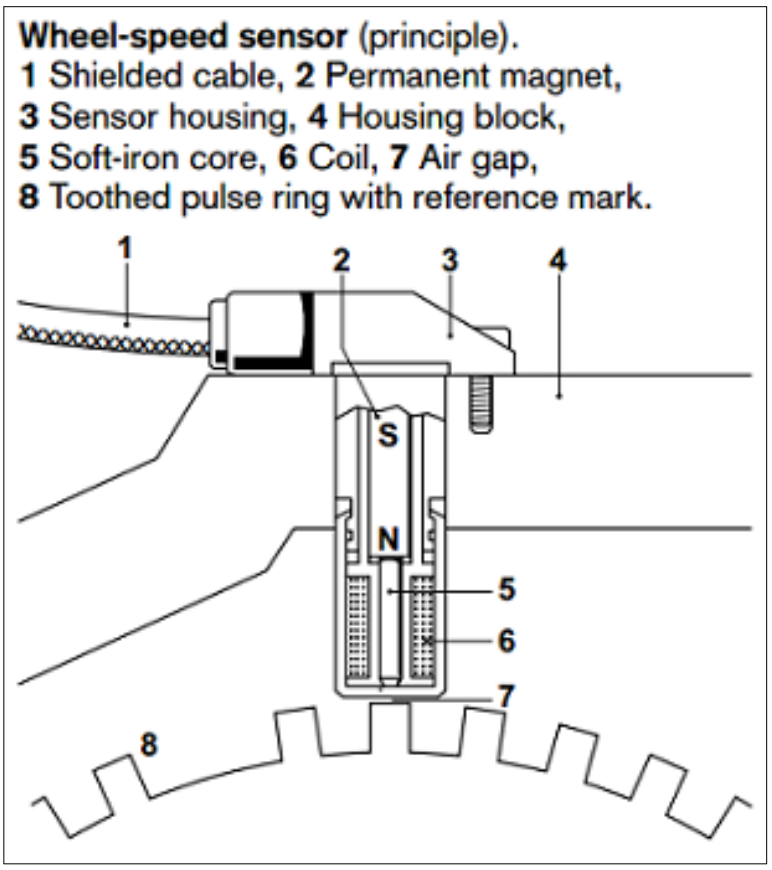
Neste projeto, o pedal eletrônico é o sensor responsável por fornecer a referência de rotação para a ECU. Sendo assim, quando este não estiver pressionado a rotação de referência será aproximadamente de 800 RPM[6], e quando estiver totalmente pressionado a referência será por volta de 6500 RPM. Assim, dado uma certa referência de rotação, a ECU deverá agir corretamente em seus atuadores de forma a buscar a rotação exigida pelo pedal.

O pedal eletrônico, junto com a válvula borboleta eletrônica, revolucionaram o controle de motores à combustão interna. Além do aumento da confiabilidade do sistema em relação ao antigo pedal a cabo, o dado conjunto (pedal e borboleta eletrônicos) fez com que o controle da massa de ar fosse controlada totalmente pela unidade eletrônica, o que aumentou a robustez do sistema. Pode-se citar outras vantagens como economia de combustível, redução da emissão de poluentes e melhorias na dirigibilidade do veículo (DEUR et al., 2004).

[6] A rotação de referência da marcha lenta poderá ser maior se o motor estiver frio.

3.1.1.7 Sensor de rotação

Figura 34 – Diagrama do sensor de relutância

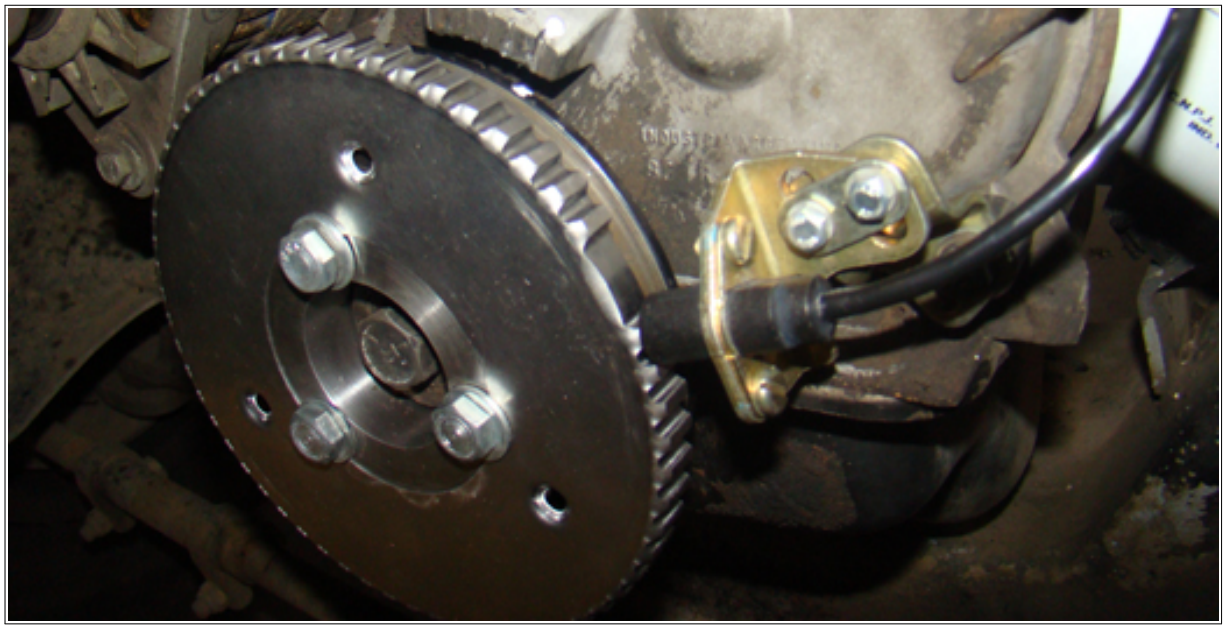


Fonte: Bosch (2013d)

O sensor de rotação é necessário para se medir a rotação instantânea do motor. No caso deste projeto, o sensor de rotação consiste de um sensor de relutância variável em conjunto com a roda fônica.

Acoplada ao eixo do motor, a roda fônica é constituída de uma roda dentada com total de 58 dentes e uma falha (vide figura a seguir) correspondente ao espaço 2 dentes. Por estarem associados à posição dos pistões do motor, os dentes são utilizados pela ECU para controle de posição da injeção e ignição, ou seja, a ECU é capaz de processar a contagem dos dentes de forma a acionar a ignição e injeção nos instantes corretos, requisito fundamental para o correto funcionamento do motor a combustão interna. Além disto, os dentes são utilizados pela unidade de gerenciamento para calcular a rotação do motor em um dado instante. Já a falha (ausência de 2 dentes na roda dentada) tem a finalidade de sincronismo, indicando o começo de uma nova volta, de modo que a ECU possa então reiniciar contagens e zerar parâmetros para serem medidos e recalculados.

Figura 35 – Roda Fônica equipada com o sensor de rotação



Fonte: Igniflex... (2013)

Como explicado anteriormente, neste projeto o sensor de rotação que está acoplado à roda fônica corresponde a um sensor de relutância variável[7]. O sensor é formado por um imã permanente e um núcleo ferromagnético, que possui uma bobina enrolada ao seu eixo. Assim, o sensor, em conjunto com o material ferro-magnético da roda fônica, forma um circuito magnético, de modo que quando o dente da roda fônica está alinhado com o sensor ocorre diminuição do entreferro[8], o que provoca uma diminuição da relutância (daí o nome sensor de relutância variável) e, consequentemente, um aumento do fluxo do circuito magnético (BOSCH, 2013d). Já quando não há presença do dente, o entreferro é maior, o que provoca uma diminuição do fluxo magnético. Com o movimento da roda fônica, a variação do fluxo magnético faz induzir, segundo a Lei de Faraday (SERWAY; JEWETT, 2004), uma tensão alternada na bobina. Assim, quando a roda fônica está em repouso, a tensão de saída do sensor é igual a zero, pois neste caso não há indução de tensão no condutor dado que o fluxo magnético é constante.

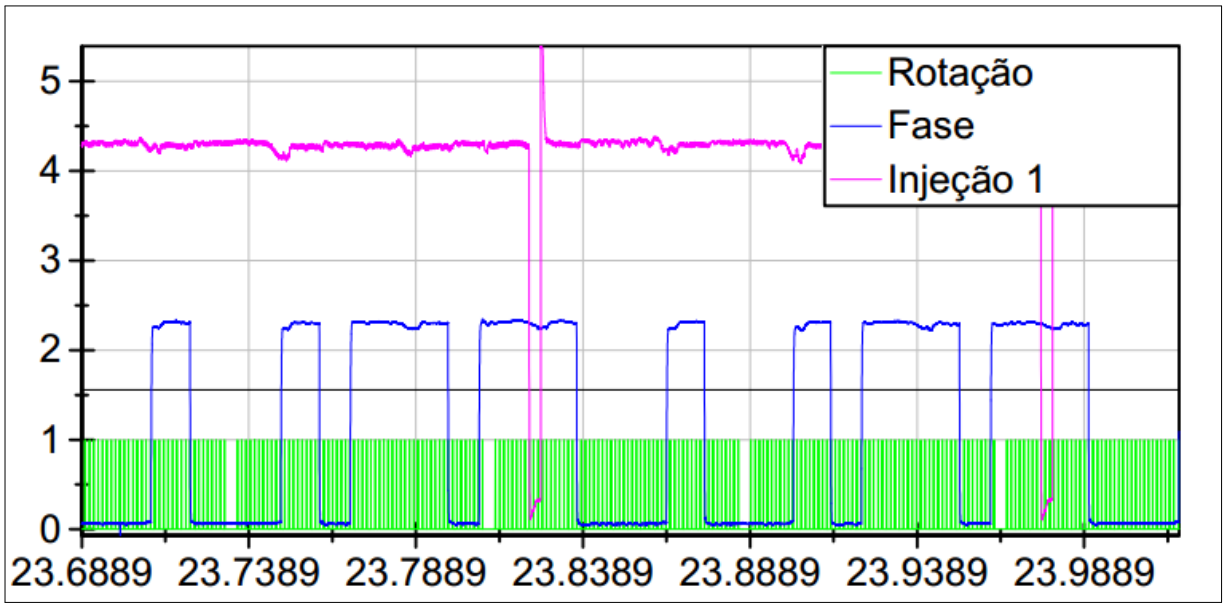
Portanto, o sensor fornece uma tensão de saída que varia de acordo com o movimento da roda fônica, seguindo a ocorrência dos dentes na posição do sensor. Vale lembrar que, uma vez que o sinal de saída deste tipo de sensor é analógico (semelhante a uma senoide), é necessário o uso de circuitos para digitalizar o sinal de saída, a fim de fornecer um sinal adequado ao processamento digital. O condicionamento do sinal de rotação será discutido posteriormente neste capítulo.

[7] Também conhecido como sensor indutivo.
[8] O entreferro corresponde ao espaço entre o dente e o núcleo ferromagnético do sensor.

3.1.1.8 Sensor de fase

O sensor de fase é responsável por fornecer a informação que relaciona a posição do virabrequim com a posição do comando de válvulas, de tal forma que a ECU consiga distinguir em qual cilindro ela deve injetar o combustível em um dado instante (MTE-THOMSON, 2013). Não se sabe a priori em qual cilindro deve ocorrer a injeção, no caso em que se utiliza somente o sinal de rotação como fonte de informação para se tomar esta decisão. Somente se sabe que a injeção deve ocorrer em uma dada sequência como, por exemplo, 1-3-4-2[9]. Também se sabe o instante (adiantamento) em que se deve injetar em uma dada volta. Com o sinal fornecido pelo sensor de fase, é possível identificar o cilindro que se deve acionar em uma dada volta. Abaixo segue um exemplo com o sinal fornecido pelo sensor de fase presente no motor a ser controlado neste projeto.

Figura 36 – Acionamento da injeção do cilindro 1 de acordo com o sinal de fase e sinal de rotação



Fonte: Escola Politécnica da USP; Fatec Santo André (2013)

Nota-se que, apesar de não ser simétrico, este sinal é complementar em alguns pontos da volta, como, por exemplo, na falha da roda dentada. Assim, com esta informação, o *firmware* é capaz de identificar a fase da injeção. É importante ressaltar que a identificação de fase não é trivial, pois o *firmware* deve levar em conta o fato do tempo de injeção ser largamente variável e não regular no tempo, além de considerar estratégias de tratamento para transições críticas[10].

No âmbito deste projeto, o motor a ser controlado é equipado com sensor de fase, como mencionado anteriormente. Assim, o bloco de Gerenciamento não precisa adotar

[9] Injeta-se primeiramente no cilindro 1, depois no 3, 4 e finalmente no 2, completando 2 voltas.

[10] Um exemplo de transição crítica corresponde ao ponto em que o instante de acionamento da injeção "'cruza"' o instante da volta em que o *firmware* detecta a fase, em caso de aceleração ou desaceleração.

estratégias complicadas para identificar a fase de injeção, como, por exemplo, a estratégia implementada no projeto de 2012 (SCARPINETTI; SOARES, 2012) que utiliza um algoritmo complexo de identificação da fase com base na rotação do motor. Todavia, neste projeto ainda se faz necessário o uso de uma estratégia de identificação da fase, estratégia esta que será mais simples do que a estratégia do projeto anterior de 2012, devido à presença do sensor de fase. A referida lógica de detecção é implementada no microcontrolador de Sincronismo e será abordada posteriormente.

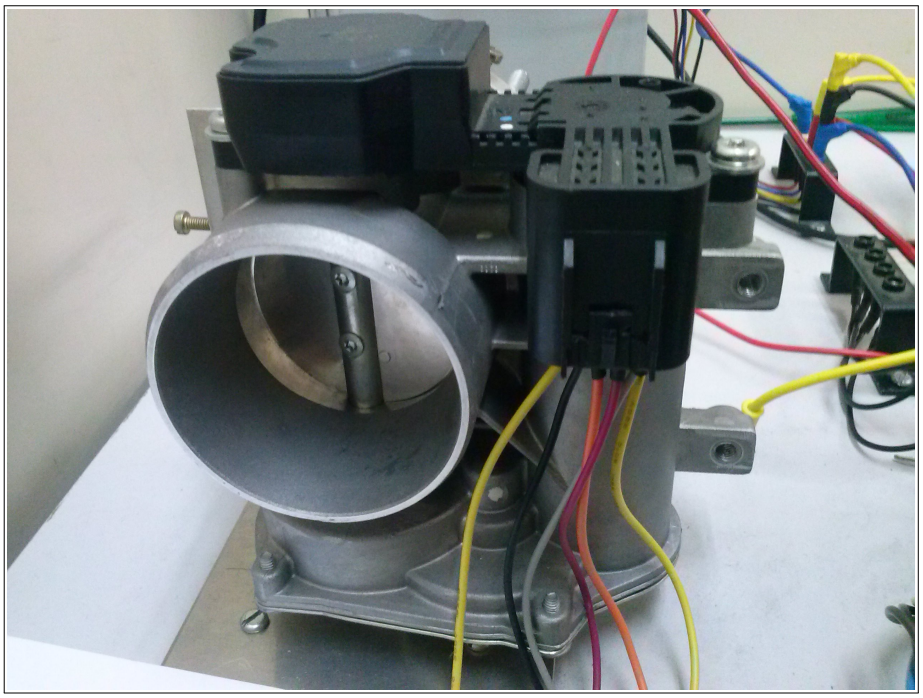
3.1.2 Atuadores

Os atuadores são dispositivos utilizados quando se deseja provocar alguma ação no motor (MANAVELLA, 1996), ação esta resultante do processamento, pela unidade de gerenciamento eletrônico, das informações oriundas dos sensores.

Nesta seção serão abordados os principais atuadores utilizados para o controle do motor deste projeto.

3.1.2.1 Válvula Borboleta

Figura 37 – Válvula Borboleta de Admissão de Ar



Fonte: o autor

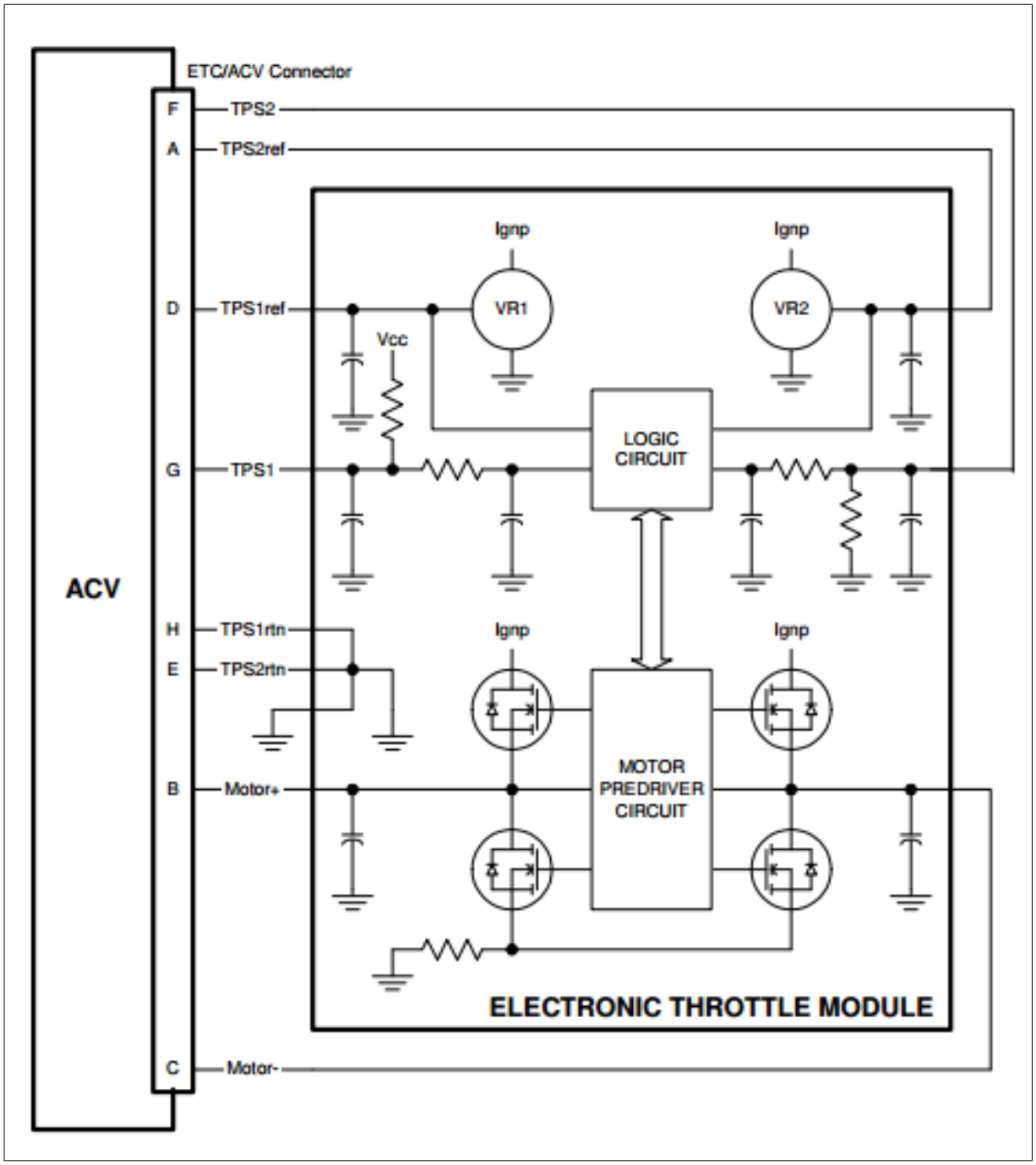
A válvula borboleta (VB) é utilizada pela unidade de comando para controle do fluxo de ar admitido pelo motor (DELPHI, 2003). Pelo fato de poder ser controlada eletronicamente, esta válvula foi fundamental para a substituição do pedal a cabo pelo pedal

eletrônico em alguns carros atuais, melhorando a dirigibilidade do veículo, diminuindo a emissão de poluentes e reduzindo o consumo de combustível. Pode-se citar, como exemplo de aplicação que surgiu com este tipo de válvula eletrônica, o controle de tração (TCS) (DEUR et al., 2004).

O controle de posição desta válvula se faz controlando-se a tensão média aplicada a um servo motor que, em conjunto com uma mola interna, controla a abertura da válvula. Em geral, esta tensão média é implementada com um pulso de amplitude fixa com largura modulada (PWM - *Pulse-Width Modulation*), dado que este tipo de sinal pode ser facilmente gerado em microcontroladores. Além disto, a válvula já contém integrado um sensor de posição[11].

[11] Este sensor corresponde ao TPS, explicado anteriormente.

Figura 38 – Diagrama de blocos da válvula borboleta eletrônica



Fonte: Delphi (2003)

3.1.2.2 Relés

Figura 39 – Relé de potência

Fonte: Bosch (2013c)

Os relés são utilizados para o acionamento de dispositivos de potência presentes no veículo tais como bombas, aquecedores elétricos, ar condicionado, dentre outros. Especificamente neste projeto, a unidade eletrônica controla apenas o relé da bomba de combustível, cuja função é retirar o combustível do reservatório do veículo e enviá-lo a linha de alimentação do injetor (MANAVELLA, 1996).

Em geral, a bomba de combustível somente é acionada com o motor em movimento, permanecendo desligada quando o motor estiver em repouso. Nas ECUs atuais, geralmente a bomba é acionada ao se ligar a chave de ignição (motor em repouso), porém é desligada após alguns segundos caso a partida não seja acionada. Este procedimento é realizado com o intuito de pressurizar a linha de combustível antes de se realizar a partida do motor (ALBALADEJO, 2013).

3.1.2.3 Válvulas Injetoras

Figura 40 – Válvula de injeção

Fonte: Bosch (2013a)

São válvulas solenoides que controlam a quantidade de combustível injetada no motor (MANAVELLA, 1996), através de comandos provenientes da ECU. Este tipo de válvula apresenta somente duas posições de operação: fechado ou aberto. Uma vez aberto, o injetor permite a entrada de combustível no motor com um certo fluxo[12], que depende das características construtivas da válvula utilizada. Sendo assim, a massa de combustível inserida no motor é determinada pelo tempo em que o injetor permanece aberto, tempo este denominado tempo de injeção. Assim, a ECU pode controlar a quantidade de combustível injetada controlando precisamente o tempo de ativação do sinal de comando do bico injetor.

[12] O valor deste fluxo pode ser obtido de manuais fornecidos pelo fabricante.

3.1.2.4 Bobinas de Ignição

Figura 41 – Tipos de bobinas de ignição

Fonte: Bosch (2013a)

As bobinas de ignição são responsáveis por transformar a tensão proveniente da bateria do automóvel (aproximadamente 12V) em uma alta tensão (superior a 8000V), necessária para produzir a centelha de ignição nas velas (MANAVELLA, 1996). Para acionamento da bobina é necessário um estágio de potência, com um transistor de potência para efetuar o chaveamento da corrente de carga da bobina. Além disto, o circuito deve conter mecanismos de proteção contra sobretensões oriundas da carga indutiva da bobina. A ECU determina o tempo de carregamento da bobina de ignição, controlando o tempo de acionamento do sinal de comando enviado ao transistor de potência que chaveia a corrente à bobina. Em geral, este tempo é constante, mas pode variar de acordo com as condições do motor, principalmente quando houver queda na tensão da bateria do veículo. No caso deste projeto, o tempo de acionamento é considerado constante, sendo apenas modificado durante a partida[13], e o transistor de potência é acoplado à bobina de ignição. Todavia, o *hardware* construído neste projeto possui transistores de potência para caso seja necessário o acionamento de bobinas de ignição que não possuam transistores acoplados.

[13] Na partida ocorre pequena redução na tensão da bateria, decorrente do acionamento do motor elétrico de arranque, o que exige que o tempo de carregamento da bobina seja maior para compensar esta queda de tensão.

3.1.3 O Motor Volkswagen 2.0L

Figura 42 – Motor Volkswagen 2.0L

Fonte: o autor

O motor a ser controlado pela unidade eletrônica deste projeto corresponde a um Volkswagen 2.0L 8v, equipado em um veículo modelo Polo Sedan 2004. Dentre as diversas características do motor, destacam-se:

- Operação com gasolina;
- Cilindrada de 1984 cm^3 (PORTAL VRUM, 2013);
- 116 cv de potência máxima a 5200 RPM (PORTAL VRUM, 2013);
- Torque máximo de 17,3 mkgf a 2400 RPM (PORTAL VRUM, 2013);
- 4 cilindros em linha (PORTAL VRUM, 2013);
- 2 válvulas por cilindro, característico do motor 8 válvulas (8v) (PORTAL VRUM, 2013);
- 2 bobinas de ignição dupla, com acionamento banco-a-banco (acionamento simultâneo nos cilindros 1-4 e 2-3, com "'centelha perdida"' em um dos cilindros);
- Injeção multiponto sequencial 1-3-4-2 (PORTAL VRUM, 2013).

3.1.4 A plataforma de projeto

Figura 43 – Plataforma de trabalho para o desenvolvimento do *firmware*

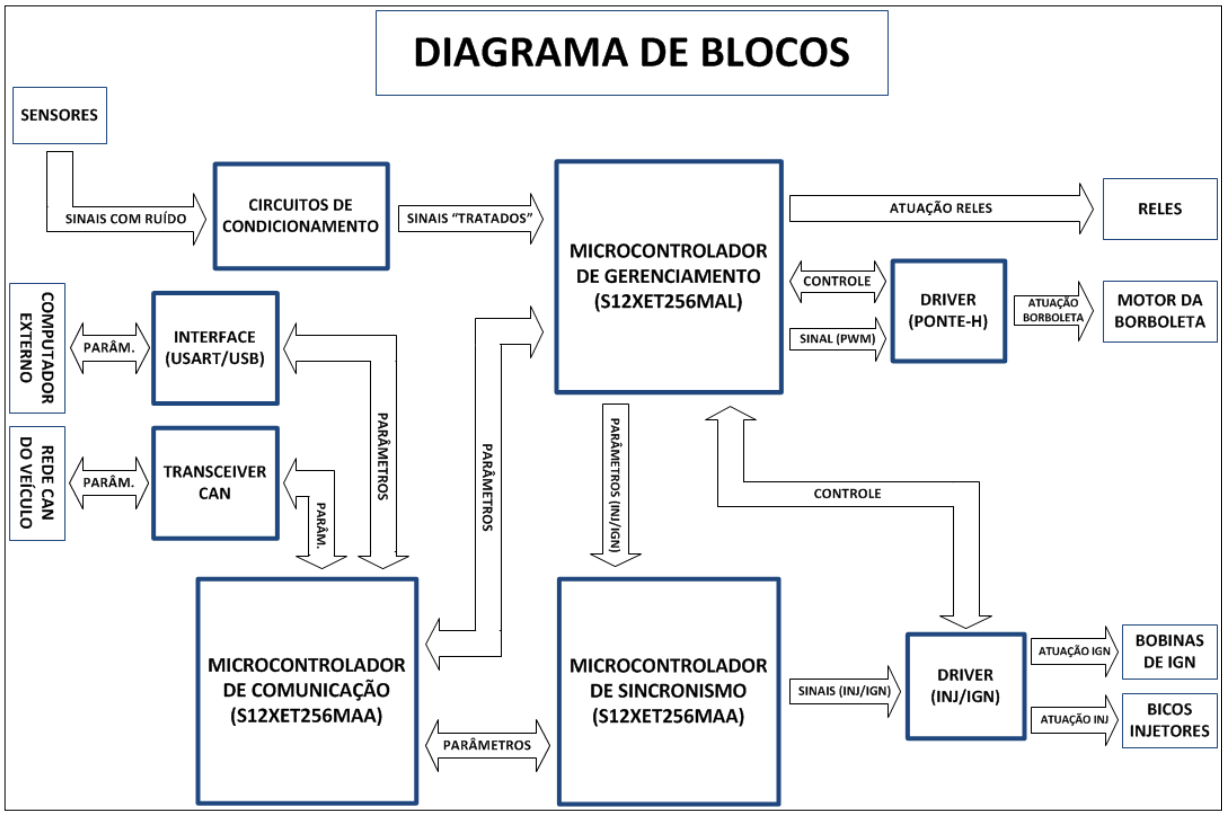
Fonte: o autor

A plataforma de projeto consiste de um veículo modelo Polo aplicado em um dinamômetro inercial, com objetivo de desenvolver a estratégia do *firmware* e avaliar o desempenho do motor em condições de carga.

3.2 Projeto Detalhado

3.2.1 *Hardware*

Figura 44 – Diagrama de blocos do *hardware*



Fonte: o autor

Na diagrama da figura acima se destacam os três principais blocos do sistema (Gerenciamento, Sincronismo e Comunicação), caracterizando uma arquitetura descentralizada com três unidades de processamento. Nota-se ainda o grande fluxo de dados entre os processadores (seguindo uma comunicação via protocolo SPI), o que é uma característica marcante da arquitetura descentralizada. Sendo assim, é necessário que a comunicação ocorra de forma rápida e confiável para que o sistema funcione corretamente, o que nem sempre é uma tarefa fácil (principalmente quando o ambiente é ruidoso como no caso deste projeto). Todavia, o *hardware* deste projeto se mostra eficiente e robusto, tendo em vista que a transmissão de dados é executada sem erros.

As tarefas executadas por cada bloco foram resumidas a seguir (mais detalhes serão apresentados ainda neste capítulo na seção de *firmware*):

- Gerenciamento: Leitura de sensores e cálculo dos parâmetros de atuação para injeção, ignição, válvula borboleta e relés;

- Sincronismo: Geração dos comandos de atuação para injeção e ignição (com parâmetros recebidos do Gerenciamento via protocolo SPI);
- Comunicação: Coleta periódica de dados dos blocos de Gerenciamento e Sincronismo, e envio dos mesmos para um computador externo (via protocolo USB) e para o painel do motorista do veículo (via protocolo CAN), além da leitura de velocidade do veículo (via protocolo CAN).

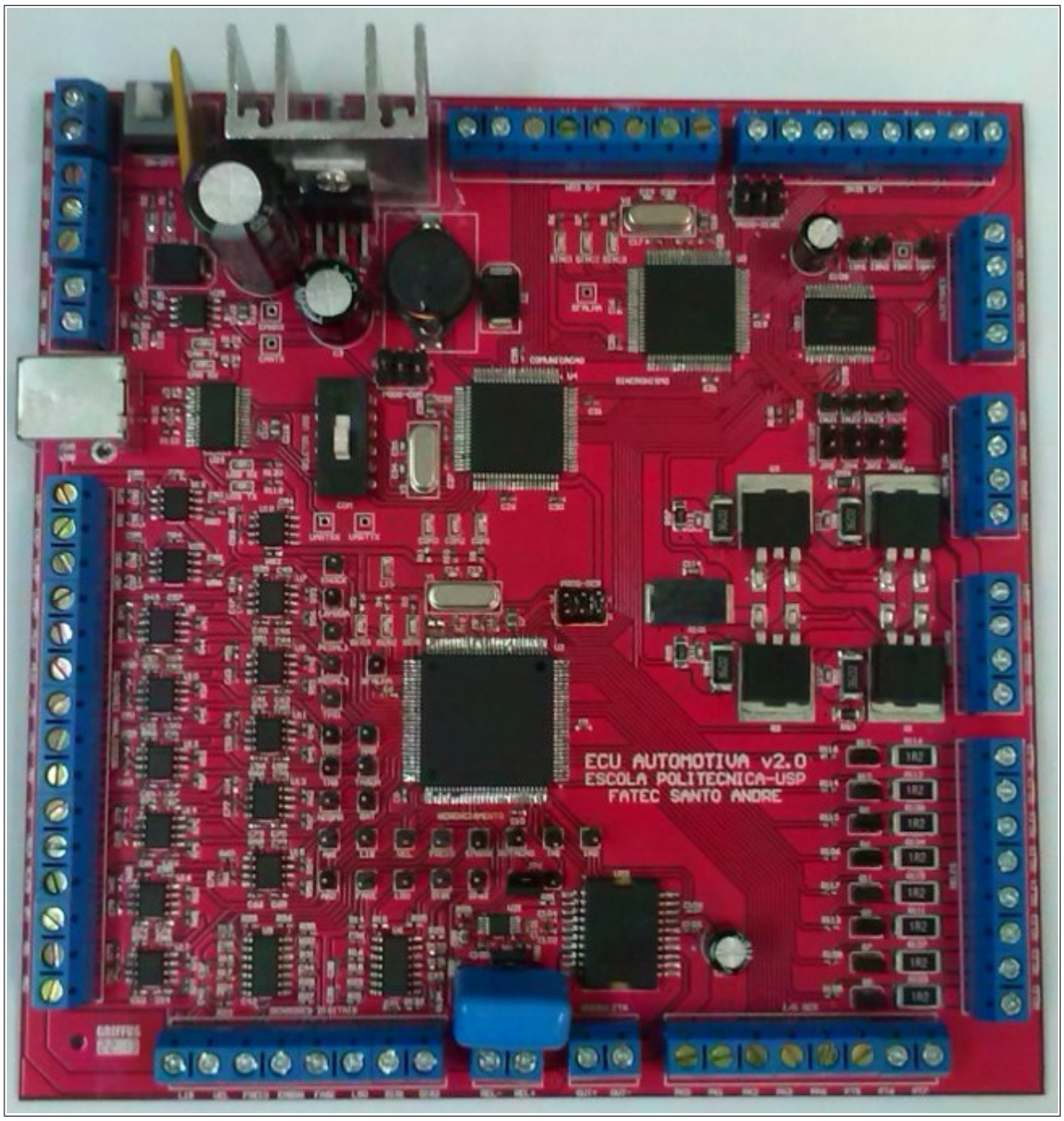
O *hardware*, projetado na ferramenta Altium Designer, usou como referência o *hardware* desenvolvido pelo Projeto Otto (SCARPINETTI; SOARES, 2012) e a primeira versão do *hardware* desenvolvido na FATEC Santo André (DIAS, 2011). No caso, ambos utilizam arquitetura descentralizada, porém empregam microcontroladores PIC da Microchip. Com relação ao *hardware* deste projeto, diversas mudanças e aperfeiçoamentos foram conduzidos, de forma a melhorar o desempenho do sistema. Dentre as principais mudanças, destacam-se:

- Emprego de microcontroladores automotivos Freescale da família S12XE, que são mais robustos e mais eficientes que o PIC18F da Microchip, sendo que o S12XE é projetado para aplicações de gerenciamento de motores automotivos (FREESCALE, 2013);
- Uso de componentes SMD menores (incluindo resistores e capacitores), o que acarretou na redução da placa de 20cm x 20cm para 15cm x 15cm;
- Uso de filtros analógicos e comparadores de tensão para o tratamento de sensores analógicos e digitais;
- Remoção do multiplexador analógico 33972, dado que o microcontrolador utilizado neste projeto possui um número elevado de entradas analógicas;
- Emprego de circuitos de proteção contra sobretensão e curto-circuito;
- *Layout* de circuito impresso mais compacto, com uso de uma camada para aterramento;
- Adição de uma comunicação USB 2.0 com um computador externo;
- Retirada do *display* LCD[14];
- Emprego de um *transceiver* CAN mais robusto;
- Adicionado pinos de medição dos sinais de sensores e atuadores;

[14] Atualmente, o monitoramento de parâmetros é realizado pela aplicação de diagnose, através da comunicação USB.

- Emprego de um circuito integrado (CI) mais robusto para tratar o sinal de rotação proveniente do sensor de relutância variável, dado que o circuito anterior (LM1815) apresentava problemas com ruídos.

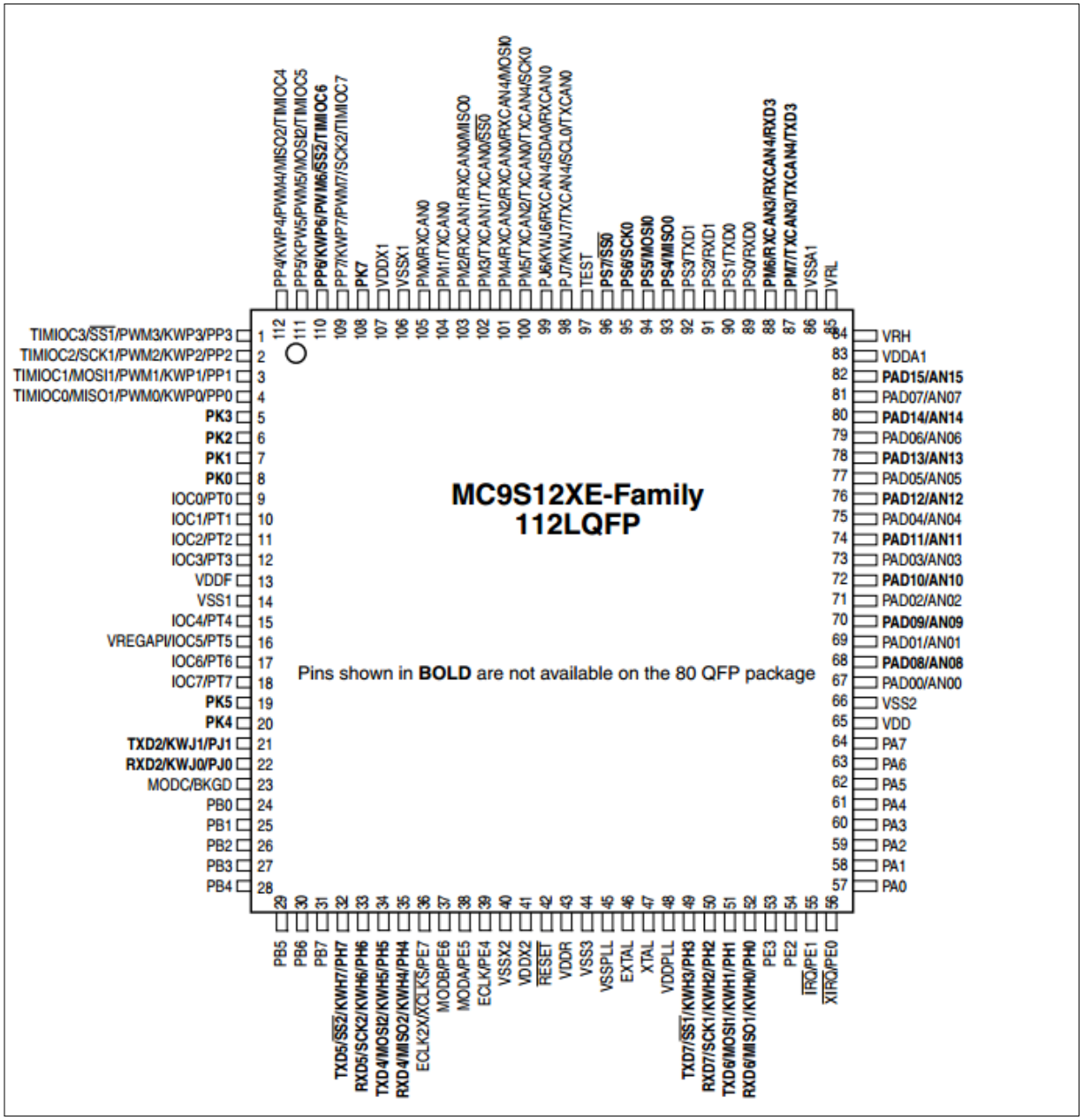
Figura 45 – Protótipo da ECU v2.0



Fonte: o autor

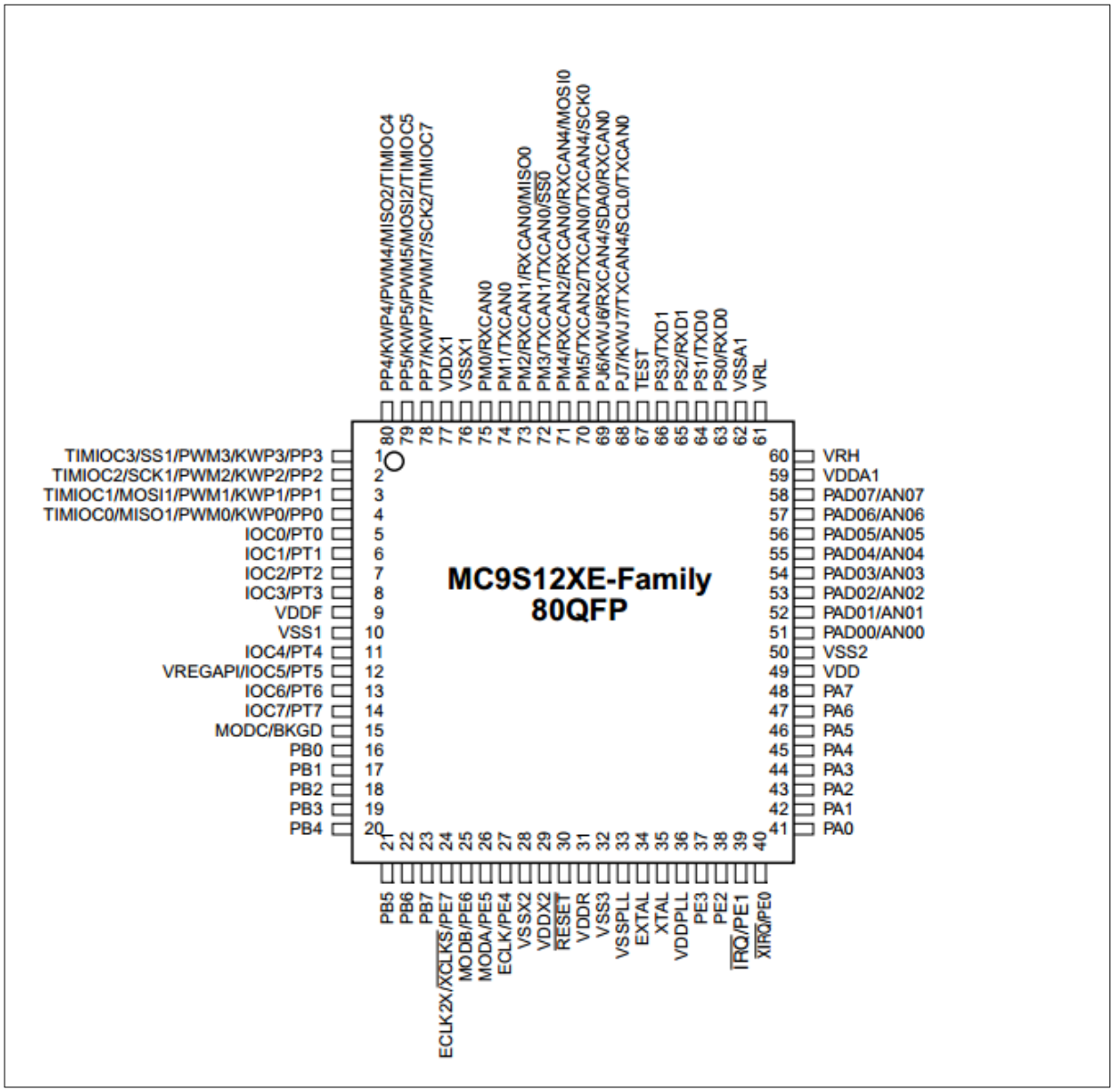
3.2.1.1 Microcontroladores

Figura 46 – S12XE com encapsulamento de 112 pinos



Fonte: Freescale (2012)

Figura 47 – S12XE com encapsulamento de 80 pinos



Fonte: Freescale (2012)

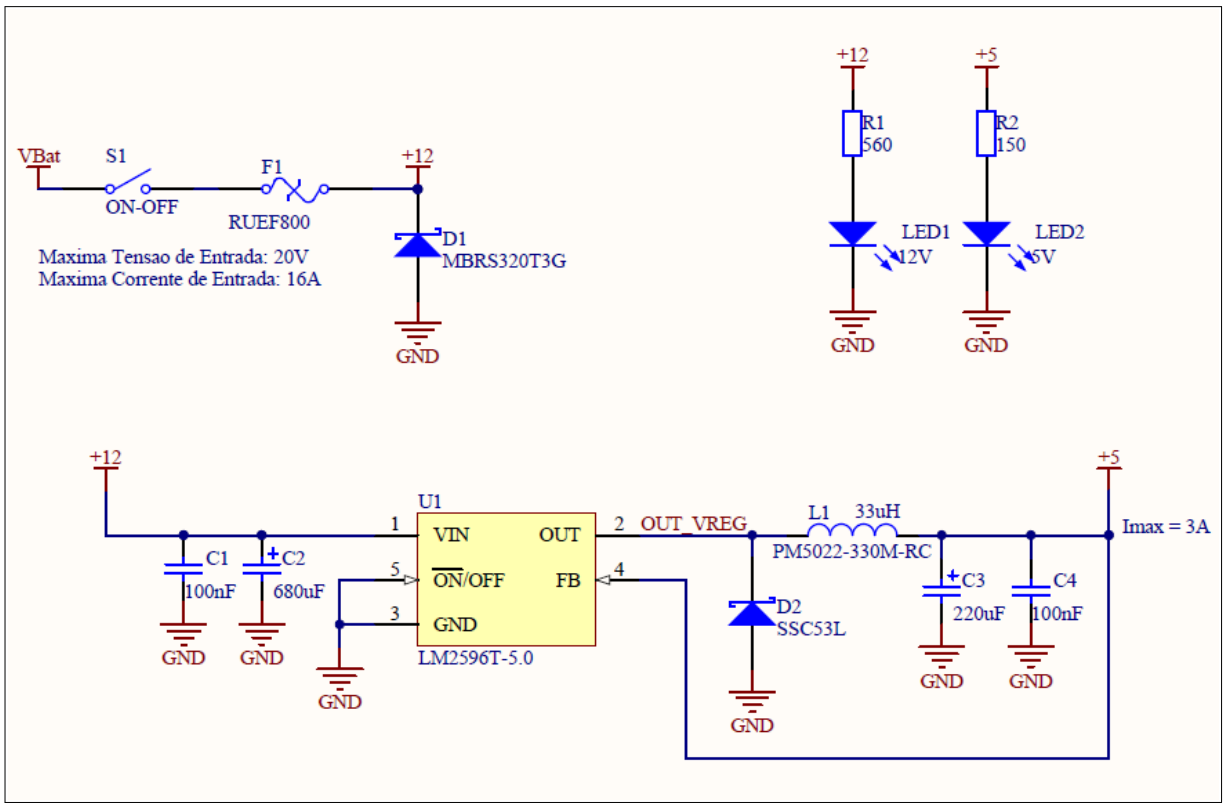
O microcontrolador escolhido para realizar as tarefas dos três blocos do sistema (Gerenciamento, Sincronismo e comunicação) foi o S12XET256, componente da família S12XE da Freescale dedicada ao ramo automotivo. No caso do Gerenciamento optou-se pelo encapsulamento com 112 pinos, diferentemente do Sincronismo e Comunicação, que utilizam a versão com 80 pinos. Todos os microcontroladores trabalham com cristais de Quartzo externos de 4 MHz, mas operam na frequência de 50 MHz resultante da amplificação da frequência proveniente do cristal[15]. Além disto, três LEDs foram ligados a cada microcontrolador para realizar funções distintas, de acordo com cada bloco. Os

[15] A amplificação da frequência de entrada é realizada pelo bloco digital PLL (*Phase-Locked Loop*) presente no *hardware* do dispositivo.

LEDs têm como finalidade notificar, através de uma simples e rápida inspeção visual, estados e erros aos operadores do sistema. Também se destaca a programação *in-circuit* realizada pelo dispositivo programador USB Multilink Universal da PE Microcomputer Systems, empresa aliada da Freescale.

3.2.1.2 Regulador de Tensão

Figura 48 – Esquema elétrico do regulador de tensão



Fonte: o autor

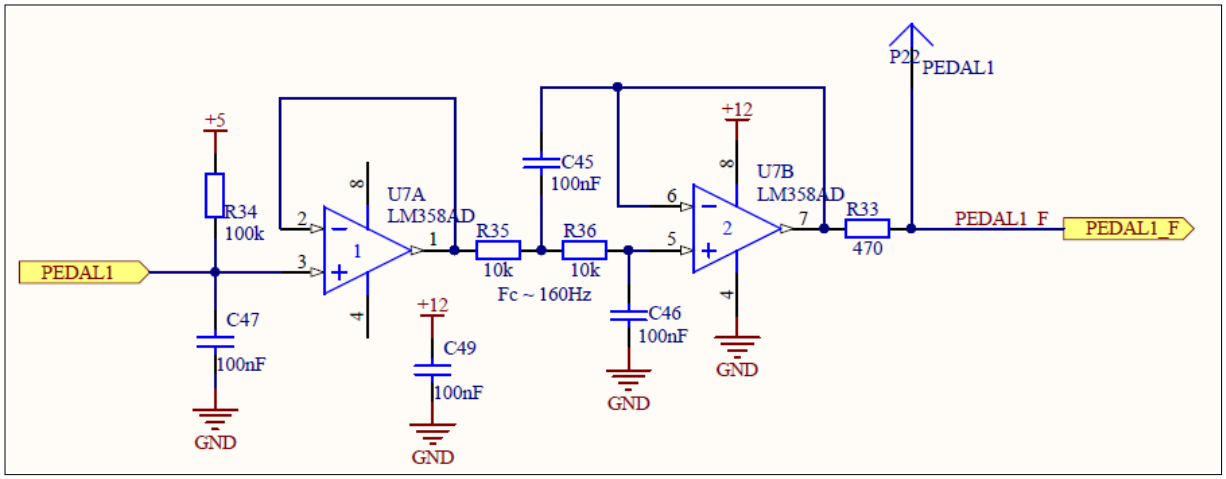
Para o regulador de tensão foi empregado o CI LM2596, cuja função na placa é converter o sinal de 12V proveniente da bateria do automóvel em um sinal de 5V estável. O circuito regulador de tensão possui capacidade de corrente de aproximadamente 3A e ainda conta com um circuito de proteção contra picos de tensão ou curto circuito.

Basicamente a proteção da placa contra alta corrente é feita por um fusível do tipo PTC (*Positive Temperature Coefficient*), de modo que quando ocorre um curto circuito a temperatura do fusível aumenta, o que faz aumentar também a sua resistência, esta que, por sua vez, bloqueia a alta corrente. O fusível se diz "'resetável"' porque o processo descrito anteriormente é reversível, ou seja, quando o dispositivo esfria (com o curto-circuito já removido), a sua resistência diminui e o circuito volta a funcionar normalmente sem necessidade da troca do fusível (LITTELFUSE, 2013). A tensão de entrada, por sua

vez, é limita por um diodo retificador. A placa também conta com LEDs para indicar a presença das tensões de 5V e 12V.

3.2.1.3 Condicionamento dos Sensores Analógicos

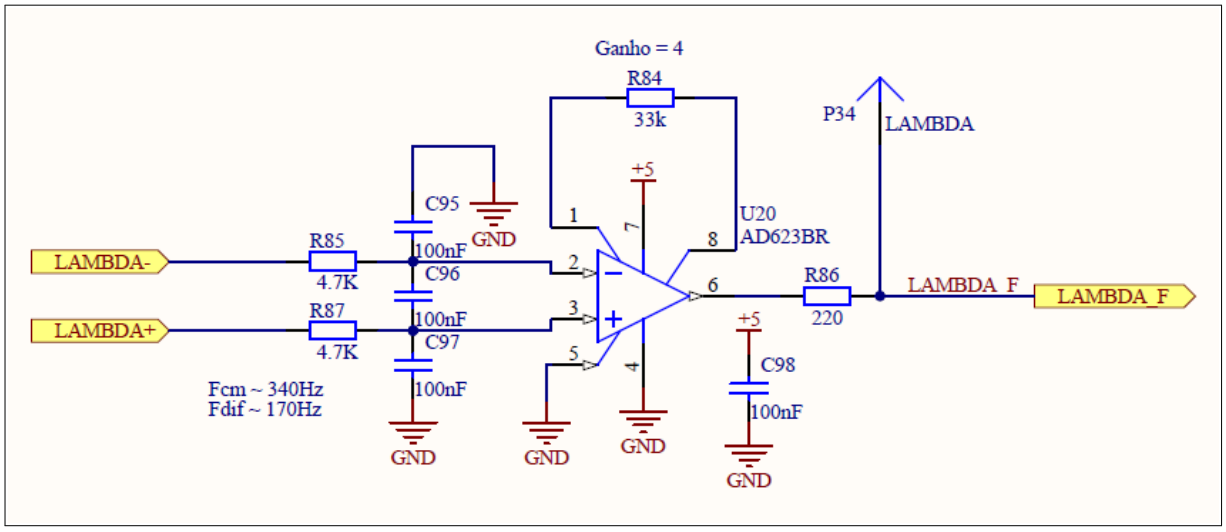
Figura 49 – Esquema elétrico do filtro analógico de segunda ordem



Fonte: o autor

A placa desenvolvida possui filtros analógicos Salent Key de segunda ordem, com frequência de corte projetada para aproximadamente 160 Hz. Além disto, nas entradas dos filtros foram colocados resistores de "'*pullup*"', com intuito de detectar, via *firmware*, sinais desconectados do sistema. Foram empregados também resistores limitadores de corrente e pinos para medição dos sinais de saída dos filtros. A implementação do filtro foi realizada com o circuito integrado LM358.

Figura 50 – Esquema elétrico do amplificador de instrumentação

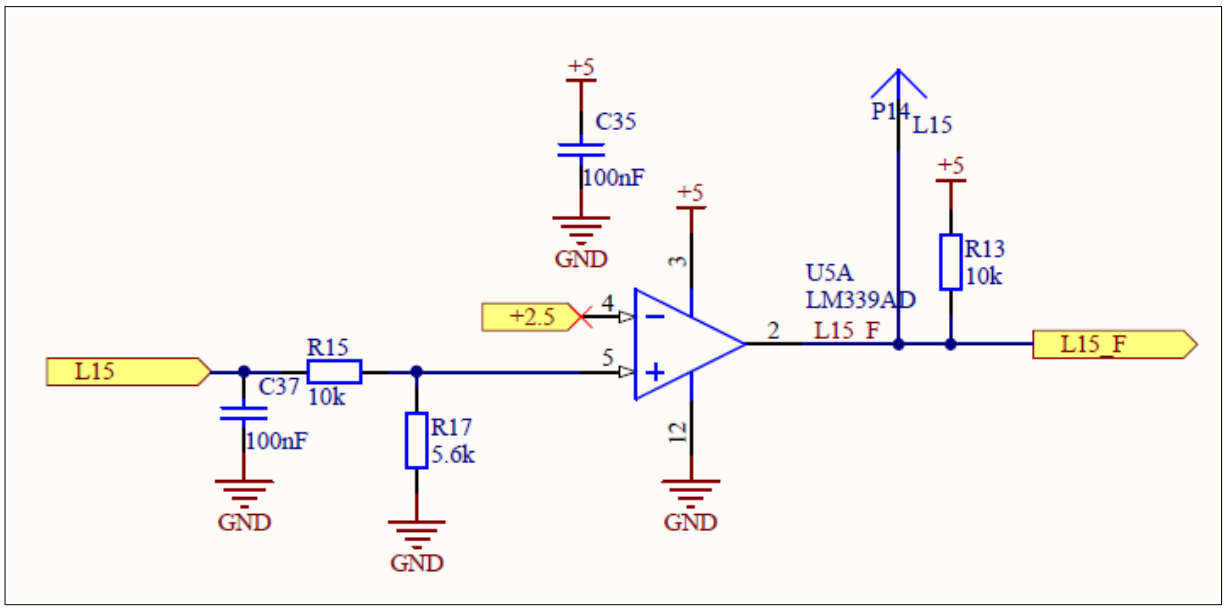


Fonte: o autor

Especificamente para os sensores Lambda e Knock[16], foram empregados amplificadores de instrumentação, uma vez que os sinais provenientes destes sensores são diferenciais e necessitam de amplificação. Os amplificadores de instrumentação foram implementados com uso do CI AD623.

3.2.1.4 Condicionamento dos Sensores Digitais

Figura 51 – Esquema elétrico do comparador de tensão



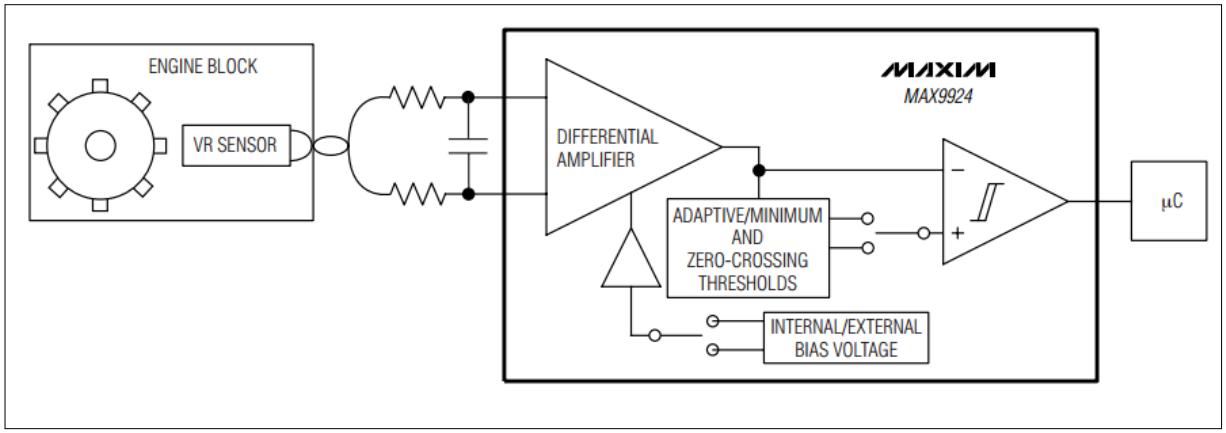
Fonte: o autor

Os sinais provenientes dos sensores digitais são tratados por um comparador de tensão com referência em um sinal de 2.5V. O objetivo é aumentar a margem de tolerância a ruído de cada entrada digital da placa. Também se destacam os pinos de medição dos sinais de saída dos comparadores, e os divisores de tensão na entrada (dado que os sinais provenientes dos sensores digitais possuem amplitude de 12V). Os comparadores de tensão foram implementados com uso do CI LM339.

[16] Apesar de não terem sido utilizados neste projeto, o *hardware* da ECU v2.0 já foi construído com interface para leitura dos sensores Lambda e Knock, o que mostra que a placa eletrônica deste projeto pode ser utilizado em projeto futuros que empregarão estes sensores.

3.2.1.5 Condicionamento do Sinal de Rotação

Figura 52 – Exemplo de aplicação do circuito integrado MAX9924



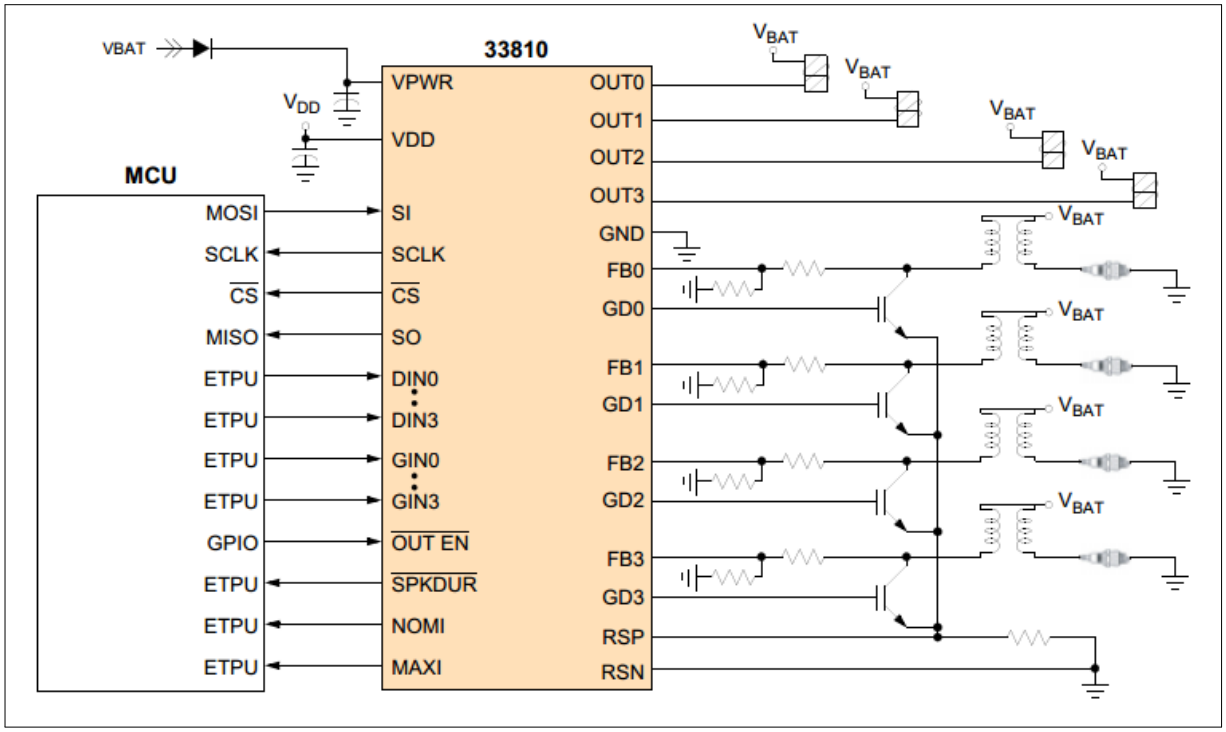
Fonte: Maxim Integrated Products (2012)

O sinal de rotação exige um condicionamento especial, dado que pequenos atrasos são cruciais para a aplicação, prejudicando o acionamento dos sinais de injeção e ignição[17]. Assim, para se condicionar o sinal de rotação proveniente do sensor de relutância variável, utilizou-se o CI MAX9924 da MAXIM. O CI possui capacidade de digitalizar a senoide de entrada de forma confiável, mesmo em um ambiente bastante ruidoso (MAXIM INTEGRATED PRODUCTS, 2012), possibilitando, desta forma, eliminar o algoritmo de filtragem utilizado no *firmware* da versão anterior, no âmbito do Projeto Otto (SCARPINETTI; SOARES, 2012). A placa também conta com um *jumper*, a fim de possibilitar o uso tanto do sinal tratado pela interface MAX9924 como do sinal proveniente do conector de entrada da placa. A finalidade desta última configuração é a de utilizar um sinal já digitalizado gerado por um circuito externo, para realização de testes em bancada antes de aplicar a unidade eletrônica ao motor propriamente dito.

[17] Atrasos da ordem de microssegundos já são suficientes para prejudicar o acionamento de injeção e ignição, comprometendo o funcionamento do motor.

3.2.1.6 *Driver* de Injeção e Ignição

Figura 53 – Exemplo de aplicação do CI 33810



Fonte: Freescale (2008)

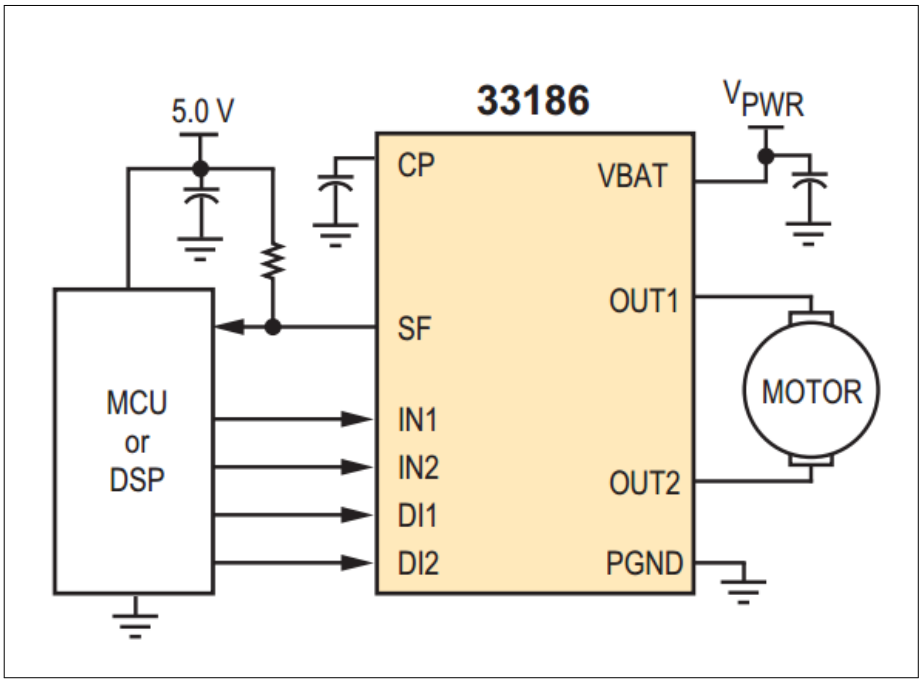
O *driver* de Injeção e Ignição corresponde ao CI 33810 da Freescale. O CI possui diversos modos de funcionamento, cuja configuração é feita pelo protocolo SPI. Além disto, o CI possui a capacidade de detectar carga em aberto ou curto-circuito, além de monitorar parâmetros úteis à diagnose da ignição ou injeção (FREESCALE, 2008).

No *hardware*, o CI pode ser empregado para acionar a ignição de dois modos. No primeiro caso (exemplificado na figura anterior), o transistor que aciona a bobina de ignição está presente na placa. Neste caso, o CI pode medir e controlar parâmetros da ignição, como a corrente do primário da bobina ou a duração da centelha. Já no segundo caso, o transistor que aciona a ignição está fora da placa, sendo que o seu acionamento é realizado por um sinal enviado à sua base.

No caso deste projeto, será utilizada a segunda opção, uma vez que o transistor de potência está integrado à bobina de ignição do motor deste projeto. Todavia, é importante ressaltar que o *hardware* pode ser utilizado em outros sistemas em que a bobina de ignição não possui nenhum *driver* acoplado. Sendo assim, o *hardware* se mostra flexível para outros tipos de aplicações, não se restringindo apenas ao atual projeto.

3.2.1.7 *Driver* para a Válvula Borboleta (Ponte H)

Figura 54 – Exemplo de aplicação do CI 33186



Fonte: Freescale (2011)

A placa foi projetada com o CI 33186 da Freescale, que corresponde a uma ponte H automotiva utilizada, neste projeto, para acionar o servo motor da válvula borboleta. Consequentemente, o projetista não precisa se preocupar em utilizar transistores de potência para acionar o servo motor, o que representa também um ganho de espaço na placa, dado que o CI ocupa menos espaço do que os transistores discretos.

O CI possui capacidade de operar em diversos modos, segundo uma tabela verdade (figura adiante) que é definida com base no controle de pinos digitais de entrada, estes que, no âmbito deste projeto, são controlados pelo uC de Gerenciamento. O CI também é capaz de detectar e se proteger de curto-circuitos e altas temperaturas (FREESCALE, 2011).

Figura 55 – Tabela da verdade do CI 33186

Device State	Input Conditions				Status		Outputs	
	DI1 (8)	DI2 (8)	IN1	IN2	SF(9)	SF(10)	OU1	OU2
1-Forward	L	H	H	L	H	H	H	L
2-Reverse	L	H	L	H	H	H	L	H
3-Free Wheeling Low	L	H	L	L	H	H	L	L
4-Free Wheeling High	L	H	H	H	H	H	H	H
5-Disable 1	H	X	X	X	L	H	Z	Z
6-Disable 2	X	L	X	X	L	H	Z	Z
7-IN1 Disconnected	L	H	Z	X	H	H	H	X
8-IN2 Disconnected	L	H	X	Z	H	H	X	H
9-DI1 Disconnected	Z	X	X	X	L	H	Z	Z
10-DI2 Disconnected	X	Z	X	X	L	H	Z	Z
11-Current Limit.active	L	H	X	X	H	H	Z	Z
12-Undervoltage(6)	X	X	X	X	L	L	Z	Z
13-Overtemperature(7)	X	X	X	X	L	L	Z	Z
14-Overcurrent(7)	X	X	X	X	L	L	Z	Z

Notes

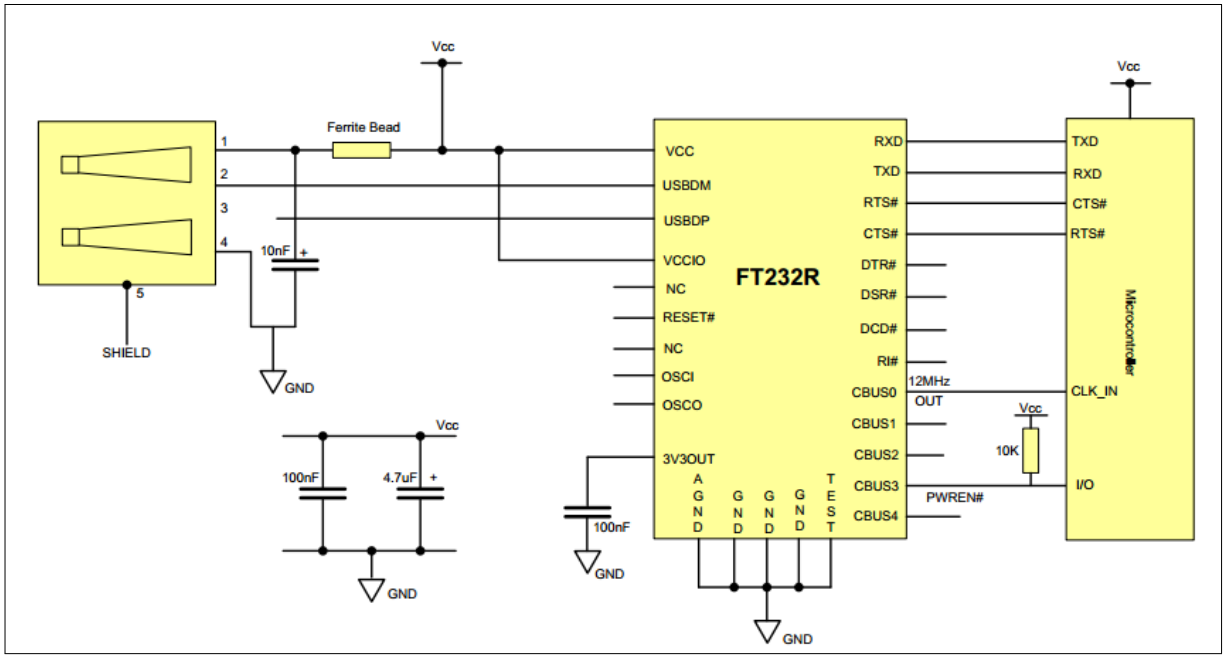
6. In case of undervoltage, tristate and status-flag are reset automatically.
7. Whenever overcurrent or overtemperature is detected, the fault is stored (i.e.status-flag remains low). The tristate conditions and the status-flag are reset via DI1 (IN1) or DI2 (IN2). Pin names in brackets refer to coding Pin (COD = VCC).
8. If COD = VCC then DI1 and DI2 are not active.
9. COD = nc or GND
10. COD = VCC

L = Low
H = High
X = High or Low
Z = High impedance (all output stage transistors are switched off).

Fonte: Freescale (2011)

3.2.1.8 Conversor UART-USB

Figura 56 – Exemplo de aplicação do FT232 com alimentação via USB



Fonte: Future Technology Devices International Limited (2010)

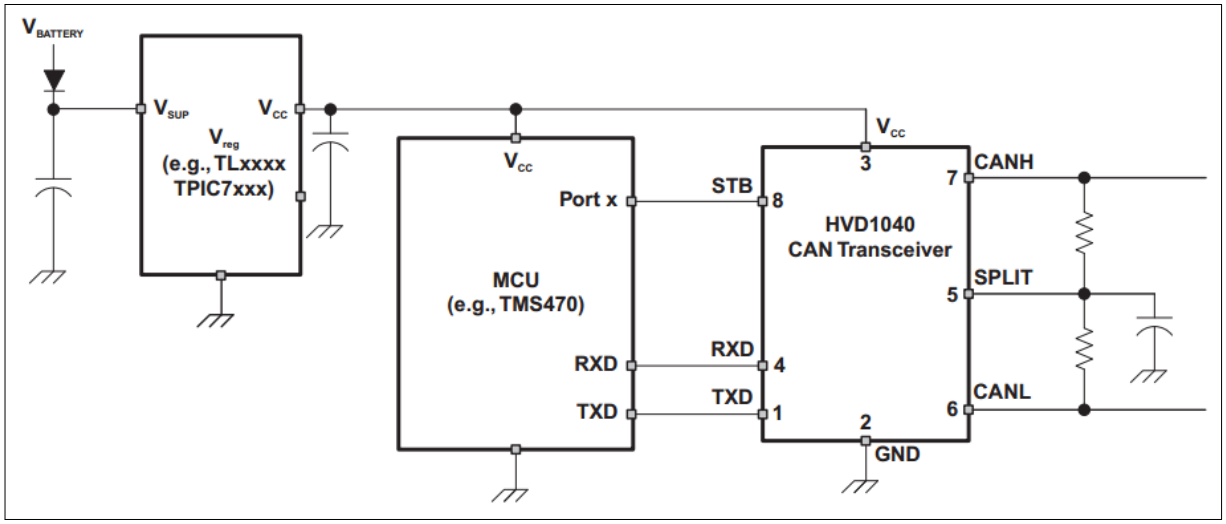
O *hardware* possui o circuito integrado FT232 da FTDI Chip, cuja função é realizar a conversão entre os protocolos serial assíncrono (UART) e USB (FUTURE TECHNOLOGY DEVICES INTERNATIONAL LIMITED, 2010). Com isto, o usuário programador não precisa se preocupar com os detalhes de programação envolvidos no protocolo USB, tanto do ponto de vista do *firmware* (uC de Comunicação) como do ponto de vista do *software* (aplicação de monitoramento). Consequentemente, a única preocupação é a programação do protocolo serial, que, por sua vez, é menos complexo do que o protocolo USB. Vale lembrar também que o CI pode ser empregado em aplicações em que a alimentação da placa se dá pelo barramento USB[18], como exemplificado na figura anterior.

O *hardware* deste projeto conta com LEDs de transmissão e recepção, pinos de testes, além de uma chave manual que permite direcionar o barramento serial tanto para o uC de Gerenciamento, como para o uC de Comunicação[19].

[18] Neste caso, o barramento USB é capaz de fornecer até 500 mA de corrente ao circuito.
[19] No presente projeto, esta chave está sempre na posição do uC de Comunicação.

3.2.1.9 *Transceiver* CAN

Figura 57 – Exemplo de aplicação do CI SN65HVD1040

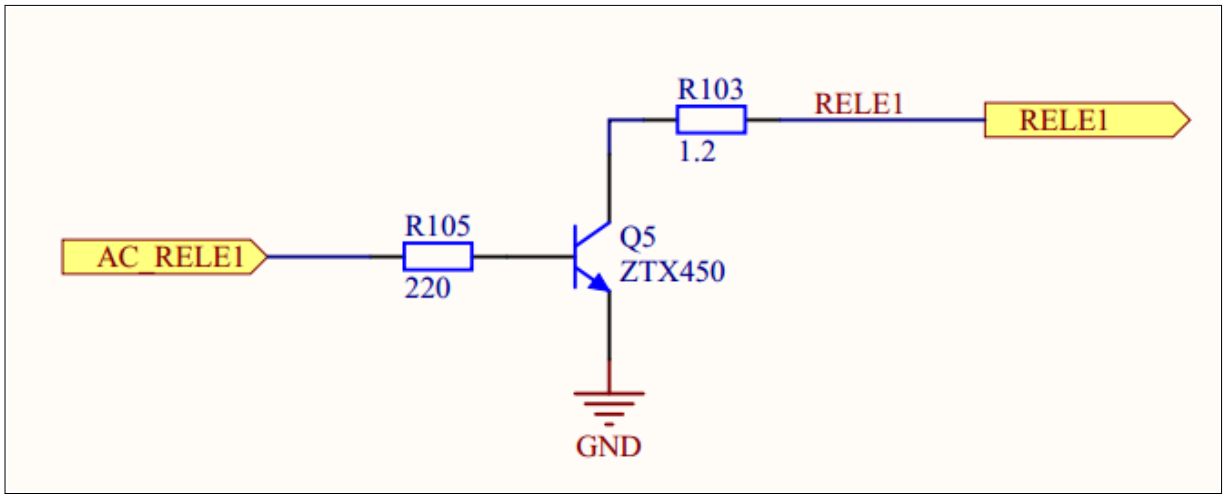


Fonte: Texas Instruments (2011)

O *hardware* da ECU v2.0 foi projetado utilizando como *transceiver* CAN o CI SN65HVD1040 da Texas Instruments, cuja função é ajustar os níveis de tensão do barramento CAN. Além disto, o integrado possui capacidade de operar com taxas de transmissão de até 1Mb/s (TEXAS INSTRUMENTS, 2011). Vale lembrar que, diferentemente do projeto anterior, o microcontroladores utilizados neste projeto já possuem o controlador CAN integrado, o que descarta o uso de CIs controladores externos.

3.2.1.10 Acionamento de Relé

Figura 58 – Esquema elétrico do acionamento de relé



Fonte: o autor

O acionamento de relés presentes no veículo é feito de forma muito simples, empregando-se transistores NPN com alta capacidade de corrente. No caso deste projeto, o *hardware* possui capacidade para acionar até 8 relés de forma independente.

3.2.1.11 Pinagem da placa

A placa foi projetada para facilitar as conexões elétricas com os sinais provenientes do motor. Para isto, escritas de identificação foram colocadas próximas de cada pino. Abaixo segue uma descrição de cada conector da placa.

Tabela 2 – Pinagem do *hardware* da ECU v2.0

Conector	**Pino**	**E/S**	**Descrição**	**Observação**
VIN	VBAT	E	Positivo da Bateria	
	GND	E	Negativo da Bateria (massa)	
VOUT	12V	S	Tensão da Bateria (aprox. 12V)	
	5V	S	Tensão de 5V controlada	
	GND	E	Massa	
CAN	CANL	E/S	Comunicação CAN	
	CANH	E/S	Comunicação CAN	
USB	-	E/S	Comunicação USB	
SENSORES ANALÓGICOS	KNO-	E	Negativo do sensor Knock	Não utilizado
	KNO+	E	Positivo do sensor Knock	Não utilizado
	LAMB-	E	Negativo da sonda Lambda	Não utilizado
	LAMB+	E	Positivo da sonda Lambda	Não utilizado
	PED1	E	Pedal 1	
	PED2	E	Pedal 2	Não utilizado
	TPS1	E	TPS1 da borboleta	Não utilizado
	TPS2	E	TPS2 da borboleta	
	MAP	E	MAP	
	TAR	E	Temperatura do ar	
	TAGUA	E	Temperatura da água	
	NCOMB	E	Nível de combustível	Não utilizado
	BAT	E	Nível da Bateria	
	AN1	E	Canal analógico reserva 1	Não utilizado
	AN2	E	Canal analógico reserva 2	Não utilizado
SENSORES DIGITAIS	L15	E	Linha 15	
	VEL	E	Velocidade	Não utilizado
	FREIO	E	Freio	Não utilizado
	EMBR	E	Embreagem	Não utilizado
	FASE	E	Fase	
	L50	E	Linha 50	Não utilizado

continua na próxima página

continuação da página anterior

	DIG1	E	Canal digital reserva 1	Não utilizado
	DIG2	E	Canal digital reserva 2	Não utilizado
SENSOR	REL-	E	Negativo do sensor de relutância v.	
	REL+	E	Positivo do sensor de relutância v.	
BORBOLETA	OUT+	S	Positivo do motor da borboleta	
	OUT-	S	Negativo do motor da borboleta	
RELES	RELE1	S	Acionamento do Relé 1	
	RELE2	S	Acionamento do Relé 2	Não utilizado
	RELE3	S	Acionamento do Relé 3	Não utilizado
	RELE4	S	Acionamento do Relé 4	Não utilizado
	RELE5	S	Acionamento do Relé 5	Não utilizado
	RELE6	S	Acionamento do Relé 6	Não utilizado
	RELE7	S	Acionamento do Relé 7	Não utilizado
	RELE8	S	Acionamento do Relé 8	Não utilizado
BOBINAS	BOB1	S	Acionamento da bobina de ign 1	Não utilizado
	BOB2	S	Acionamento da bobina de ign 2	Não utilizado
	BOB3	S	Acionamento da bobina de ign 3	Não utilizado
	BOB4	S	Acionamento da bobina de ign 4	Não utilizado
SINAL IGN	IGN1	S	Acionamento do driver de ign 1	
	IGN2	S	Acionamento do driver de ign 2	
	IGN3	S	Acionamento do driver de ign 3	
	IGN4	S	Acionamento do driver de ign 4	
INJETORES	INJ1	S	Acionamento do bico injetor 1	
	INJ2	S	Acionamento do bico injetor 2	
	INJ3	S	Acionamento do bico injetor 3	
	INJ4	S	Acionamento do bico injetor 4	
I/O GER	PK0	E/S	Pino 0 da porta K do Ger.	
	PK1	E/S	Pino 1 da porta K do Ger.	
	PK2	E/S	Pino 2 da porta K do Ger.	Portas extras
	PK3	E/S	Pino 3 da porta K do Ger.	disponibilizadas
	PP6	E/S	Pino 6 da porta P do Ger.	para eventuais
	PT5	E/S	Pino 5 da porta T do Ger.	conexões
	PT6	E/S	Pino 6 da porta T do Ger.	
	PT7	E/S	Pino 7 da porta T do Ger.	
I/O SINC	PT0	E/S	Pino 0 da porta B* do Sinc.	
	PT1	E/S	Pino 1 da porta B* do Sinc.	
	PT2	E/S	Pino 2 da porta B* do Sinc.	Portas extras
	PT3	E/S	Pino 3 da porta B* do Sinc.	disponibilizadas
	PT4	E/S	Pino 4 da porta B* do Sinc.	para eventuais
	PT5	E/S	Pino 5 da porta B* do Sinc.	conexões

continua na próxima página

continuação da página anterior

	PT6	E/S	Pino 6 da porta B* do Sinc.	*errata da placa
	PT7	E/S	Pino 7 da porta B* do Sinc.	
I/O COM	PT0	E/S	Pino 0 da porta T do Com.	Portas extras disponibilizadas para eventuais conexões
	PT1	E/S	Pino 1 da porta T do Com.	
	PT2	E/S	Pino 2 da porta T do Com.	
	PT3	E/S	Pino 3 da porta T do Com.	
	PT4	E/S	Pino 4 da porta T do Com.	
	PT5	E/S	Pino 5 da porta T do Com.	
	PT6	E/S	Pino 6 da porta T do Com.	
	PT7	E/S	Pino 7 da porta T do Com.	
PROG-GER	-	E	Programação in-circuito do GER.	
PROG-SINC	-	E	Programação in-circuito do SINC.	
PROG-COM	-	E	Programação in-circuito do COM.	

Pela tabela fornecida acima é possível notar que nem todos os recursos do *hardware* estão sendo utilizados no presente projeto, pois, como mencionado anteriormente, o *hardware* foi projetado visando também a sua aplicação em projetos futuros.

Para mais detalhes do *hardware*, recomenda-se a consulta ao esquema elétrico, que pode ser encontrado nos apêndices deste trabalho.

3.2.2 *Firmware*

Como explicado anteriormente, o desenvolvimento do *firmware* envolve a programação dos três microcontroladores presentes na placa, responsáveis por executar as tarefas de cada bloco do sistema (Gerenciamento, Sincronismo e Comunicação). Assim, toda a estratégia para o controle do motor deverá ser levada em conta no desenvolvimento do código de cada microcontrolador.

É importante ressaltar que, diferentemente do projeto Otto (SCARPINETTI; SOARES, 2012), que desenvolveu um *firmware* com base maior em tabelas e calibrações, a ECU 2.0 deste projeto foi desenvolvida com um *firmware* que utiliza mais cálculos, de modo a aproveitar também a maior capacidade de processamento dos microcontroladores utilizados neste projeto[20]. Sendo assim, os códigos dos microcontroladores de cada bloco do sistema de ambos projetos são bastante diferentes. Por ser uma unidade eletrônica com base maior em cálculos, o controle deste projeto se mostrou bastante eficiente e estável, porém vale lembrar que o fato não retira o mérito do grupo de 2012 (SCARPINETTI;

[20] É importante mencionar que o projeto anterior contava com a limitada capacidade de processamento do PIC18F da Microchip.

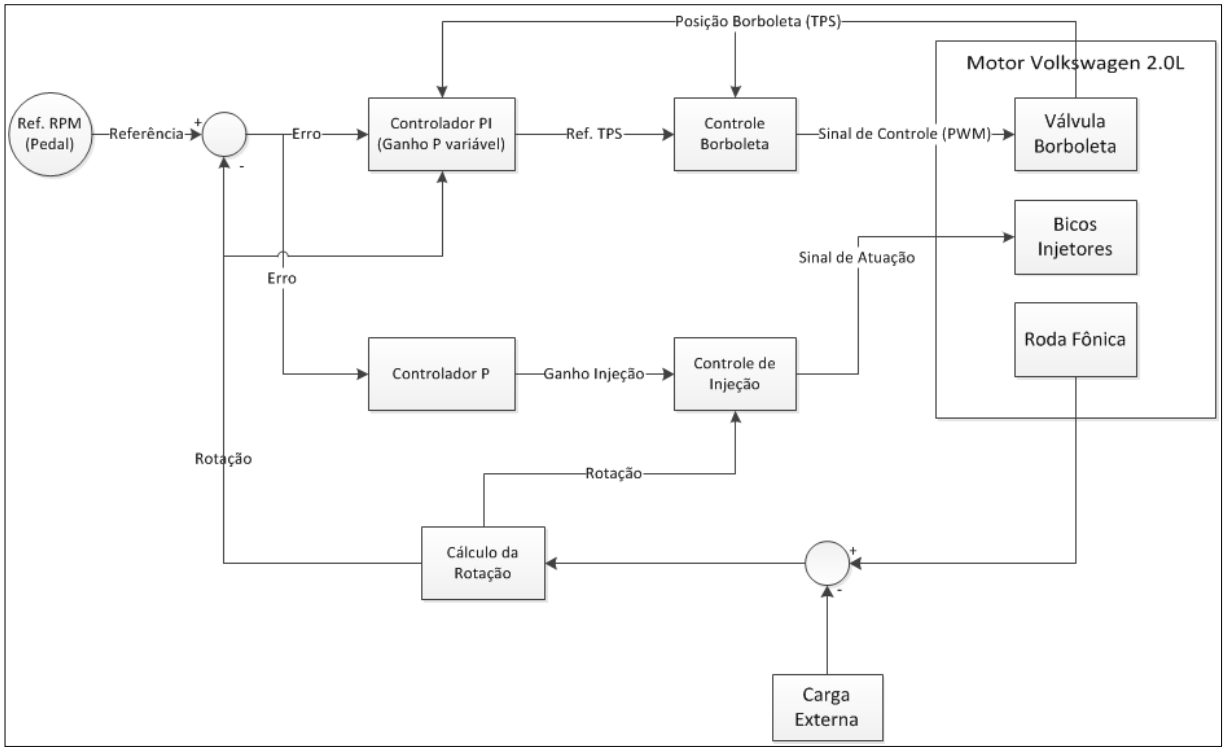
SOARES, 2012), que provaram que é possível se controlar um motor automotivo mesmo utilizando microcontroladores mais simples, o que não é uma tarefa nada fácil.

A seguir será dada uma explicação mais profunda das funções de cada bloco do sistema, bem como da estratégia de controle implementada. Fluxogramas do *firmware* de cada uC do sistema podem ser consultados nos apêndices deste trabalho. Por serem muito extensos, os códigos dos microcontroladores não foram inseridos no trabalho, porém trechos foram disponibilizados ao longo das subseções seguintes.

3.2.2.1 Gerenciamento

O microcontrolador de Gerenciamento é responsável pelo cálculo dos parâmetros de atuação do sistema, tomando como base os valores dos sensores presentes no motor. O processamento ocorre de maneira a garantir um controle de rotação estável. Sendo assim, modelou-se o controle de rotação para posterior implementação no uC de Gerenciamento.

Figura 59 – Diagrama do controle de Rotação



Fonte: o autor

Pelo diagrama acima é possível notar que para se controlar a rotação do motor é necessário atuar simultaneamente no tempo de injeção de combustível e na abertura da válvula borboleta, responsável pelo controle do fluxo de ar admitido pelo motor. Além disto, uma parte do controle do tempo de injeção consiste de um controlador puramente proporcional (P), cuja finalidade é enriquecer a mistura a/c quando a referência de rotação for maior que a rotação medida. É importante notar também que a carga imposta ao motor

atua como distúrbio no controle, de maneira a reduzir a rotação. Assim, cabe ao controle de rotação atuar de maneira a impedir esta redução, o que, consequentemente, fornece torque ao eixo do motor. Já o controle de posição da borboleta é realizado através de um controlador PI (proporcional e integral).

Abaixo foi destacado um trecho do código do uC de Gerenciamento que implementa o controle de rotação:

```
void controle_rpm() {
    float control = 0;
    float ref_vb_temp, ref_vb_base;

    // Algoritmo de controle PI
    erro_rpm = ref_rpm - (float)rotacao;

    if ((erro_rpm < 70.0) && (erro_rpm > -70.0))
        return;

    // Atualiza a somatoria
    if ((erro_rpm < 500.0) && (erro_rpm > -500.0)) // Integracao
       condicional - Anti wind-up
        soma_erro_rpm += erro_rpm;
    else
        soma_erro_rpm = 0; // Descarrega a somatoria

    // Anti wind-up (Limita a somatoria)
    if (soma_erro_rpm > 200000.0)
        soma_erro_rpm = 200000.0;
    else if (soma_erro_rpm < -200000.0)
        soma_erro_rpm = -200000.0;

    if (erro_rpm > 0.0) { // Aceleracao
        if (modo_operacao == 0) { // Modo de operacao "normal"
          if (rotacao < 1700)
              Kp_rpm = erro_rpm*0.08/5000.0; // Ajustar o primeiro
                 parametro da formula
          else if (rotacao < 3000)
              Kp_rpm = erro_rpm*(0.00013*(float)rotacao-0.142)/5000.0;
                 // Ajustar o primeiro parametro da formula
          else
              Kp_rpm = erro_rpm*0.25/5000.0; // Ajustar o primeiro
                 parametro da formula
        }
        else { // Modo de operacao "economico" (ganhos menores)
          /*if (rotacao < 1700)
              Kp_rpm = erro_rpm*0.04/5000.0; // Ajustar o primeiro
                 parametro da formula
          else if (rotacao < 3000)
```

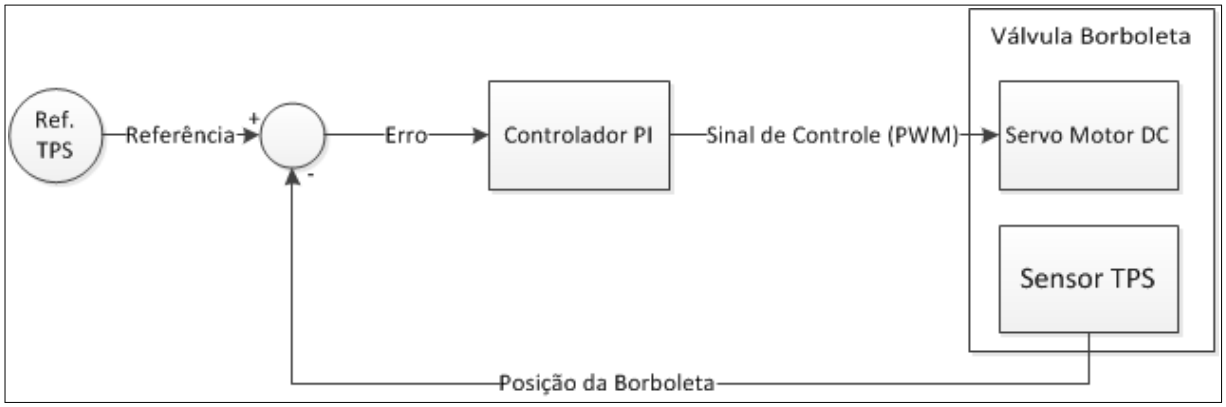
```
            Kp_rpm = erro_rpm*(0.00007*(float)rotacao-0.078)/5000.0;
                // Ajustar o primeiro parametro da formula
        else
            Kp_rpm = erro_rpm*0.13/5000.0; // Ajustar o primeiro
                parametro da formula*/
          Kp_rpm = erro_rpm*0.01/5000.0;
    }
}

else // Desaceleracao - Nao usa ganho Kp
    Kp_rpm = 0.0;

control = Kp_rpm*erro_rpm + Ki_rpm*soma_erro_rpm;
ref_vb_base = 0.0054*ref_rpm + 32.07; // Posicao da vb para manter a
    rotacao
ref_vb_temp = ref_vb_base + control;

// Limitacao da posicao max da vb para o motor nao "afogar" em
   aceleracoes grandes
//if (rotacao < 1000.0)
//    ref_vb_max = 67.3;
if (rotacao < 1700)
    ref_vb_max = 137.0;
else {
    ref_vb_max = (TPS_MAX - 137.0)*((float)rotacao - 1700.0)/1300.0
       + 137.0;
    if (ref_vb_max > TPS_MAX)
        ref_vb_max = TPS_MAX;
}

// Limita referencia para controle da borboleta
if ((ref_vb_temp < TPS_1000) && (rotacao > 1500))
    ref_vb_temp = TPS_1000;
else if (ref_vb_temp < 32.0) // Impede fechamento total da vb
    ref_vb_temp = 32.0;
else if (ref_vb_temp > ref_vb_max) // Limitacao para o motor nao "
   afogar" em aceleracoes grandes
    ref_vb_temp = ref_vb_max;

if (((erro_rpm > 1000.0) || (erro_rpm < -1000.0)) && (rotacao <
   1000))
    while ((injecao_ok < 1) && (reset == 0)); // Espera injetar
       antes de abrir a vb...

ref_vb = ref_vb_temp; // Passa p/ a variavel final

injecao_ok = 0; // reseta a variavel
```

```
}
```

Pelo código acima, é possível notar que diversos tratamentos devem ser levados em conta no controle de rotação. Este deve, por exemplo, ser robusto o suficiente para impedir que o motor "afogue"(devido ao excesso de ar) em uma aceleração brusca, o que exige limitações na abertura da borboleta. Ganhos variáveis também foram adotados, a fim de fornecer um controle que seja suave quando o pedal for pouco pressionado, e agressivo quando o pedal for bastante pressionado. Destaca-se ainda a importância do termo integral do controle para ajuste da rotação de marcha lenta.

Como mencionando anteriormente, 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 de rotação, descrito anteriormente. Assim, cabe ao controle PI garantir que a posição da válvula borboleta seja aquela determinada pelo controle de rotação (ou que seja o mais próximo possível).

Figura 60 – Diagrama do controle de posição da Válvula Borboleta



Fonte: o autor

Testes no motor revelaram que não se necessita de um controle rápido, ou seja, a borboleta não precisa abrir ou fechar rapidamente. Sobretudo, um controle estável é essencial para o sucesso do controle de rotação, o que nem sempre é uma tarefa fácil. Verifica-se que pequenas variações (da ordem de milivolts) na posição da borboleta já são suficientes para causar variações na rotação, principalmente em marcha lenta, o que pode resultar em uma instabilização do controle de rotação. Assim, apesar de simples, o controle de posição deste projeto se mostrou bastante eficiente, atendendo de modo estável as requisições do controle de rotação. É importante mencionar que o tratamento dos sensores envolvidos (pedal de aceleração e TPS) são essenciais para o sucesso do controle de posição da borboleta, dado que ruídos presentes no sistema degradariam a estabilidade do PI se não fossem devidamente filtrados pelo *hardware*.

A implementação do controlador PI é feita de forma discreta. No caso, foi utilizado

um *timer* para se controlar a frequência de amostragem em 1kHz. Em outras palavras, a cada 1 ms o laço de controle é executado de modo a minimizar o erro[21]. Abaixo segue um trecho do código do Gerenciamento que implementa o laço do controle de posição da válvula borboleta[22]:

```
interrupt void TIMER0_CONTROLE_VB_XGATE(void)
{
    float control; // Sinal de controle
    unsigned char pwm; // Valor de PWM a ser gerado
    unsigned char inv = 0;

    PITTF = (1<<0); // Limpa flag PTF0

    if (ref_vb < 10.0)
        return; // Nao Atua na borboleta

    IOGER7 = 1; // PARA TESTES

    /*----------------------Controle da Valvula Borboleta
       --------------------*/

    // Algoritmo de controle PI
    erro_vb = ref_vb - (float)tps2;

    //if ((erro_vb < 1.0) && (erro_vb > -1.0) && (tps2 < TPS_MIN)) {
    //    IOGER7 = 0;
    //    return;
    //}

    soma_erro_vb += erro_vb;

    control = Kp_vb*erro_vb + Ki_vb*soma_erro_vb;

    if (control > 0) { // Valvula precisa ser aberta
        inv = 0;
        if (control > 230.0) // Limitacao do sinal (PWM = 90%)
            control = 230.0;
    }

    else { // Valvula precisa ser fechada
        control = -control;
        inv = 1;
        if (control > 200.0)
            control = 200.0; // Limitacao do sinal (PWM = 80%)
    }
```

[21] Relembrando que o erro corresponde à diferença entre a referência e a posição medida.
[22] Este laço é executado pelo coprocessador XGate do uC de Gerenciamento.

```
    pwm = (unsigned char)control; // Passa o calculo do pwm p/ a "
        variavel final"

    if (!inv) {
        PWMDTY5 = pwm;
        PWMDTY4 = 0;
    }
    else {
        PWMDTY5 = 0;
        PWMDTY4 = pwm;
    }

    IOGER7 = 0;
}
```

Além da borboleta, o controle de rotação necessita que a injeção de combustível esteja funcionando corretamente. Para a determinação do tempo de injeção, segue-se os passos abaixo:

1. Cálculo da rotação do motor;
2. Estimativa da massa de ar admitida por cilindro com base na equação de estado dos gases ideais[23] (WYLEN; BORGNAKKE; SONNTAG, 2009);
3. Cálculo da massa de combustível a ser injetada de acordo com a relação estequiométrica de 13,3g de ar para 1g de gasolina com 22% de etanol (ROCHA, 2009);
4. Cálculo do tempo de injeção, com base no fluxo do bico injetor;
5. Aplicação de um fator multiplicativo (determinado de forma empírica), com base na aceleração e na temperatura da água.

Como mencionado anteriormente, a estequiometria da mistura a/c é controlada em malha aberta, ou seja, o *firmware* estima a massa de ar admitida pelo motor para então calcular a quantidade de combustível a ser injetada. Testes práticos demonstraram que esta estimativa funciona corretamente, de modo que quando não há aceleração nem desaceleração o motor funciona de modo estável, além do fato de não constar odor de excesso de combustível no ambiente de teste. A ideia da estimativa da massa de ar para determinação do tempo de injeção foi uma cortesia de ALBALADEJO. Abaixo segue um trecho do *firmware* que implementa as etapas descritas acima:

```
const float Kp_inj = 0.00017;
```

[23] Ou seja, adotou-se que o ar se comporta como um gás ideal.

```
...

// Calculo da rotacao
periodo_temp = (float)periodo_3 / 3.0; // Timer1 mediu a duracao de 3
    dentes
rotacao_temp = 1000000.0/periodo_temp;

rotacao = (unsigned int)rotacao_temp;

...

// Calculo da massa de ar admitida por cilindro
// OBS: Utiliza a Lei dos gases ideais (PV/4=nRT)
m_ar = 0.25 * pressao_ar * 1.984 * 28.96 / (8.314 * t_ar); // Unidade: g

// Calculo da massa de combustivel para lambda=1
m_comb = m_ar / 13.3; // Unidade: g

// Calculo do tempo de injecao (Fluxo: 176.93 g/min)
tempo_inj_temp = m_comb * 339117.17; // Unidade: us

// Ajuste do fator de injecao em caso de aceleracao/desaceleracao
fator_inj = 1.0 + Kp_inj*erro_rpm;

// Fator de correcao na injecao para motor frio
if (t_agua < 40.0)
                fator_inj += 0.2;
else if (t_agua < 70.0)
                fator_inj += 0.467 - 0.0067*t_agua;

// Limitacao superior do fator de injecao (equivale a lambda ~ 0.71)
if (fator_inj > 1.4)
                fator_inj = 1.4;

// Limitacao inferior do fator de injecao (equivale a lambda ~ 1.33)
else if (fator_inj < 0.75)
                fator_inj = 0.75;

tempo_inj_temp *= fator_inj; // Aplica o fator de correcao
```

Os acionamentos dos bicos injetores e das bobinas de ignição são realizados pelo uC de Sincronismo, porém cabe ao Gerenciamento determinar os parâmetros adequados para que estes acionamentos ocorram corretamente. A estratégia de acionamento consiste em se tabelar o término do acionamento, ou seja, o instante em que o sinal é desativado. Deste modo, é possível se controlar com maior precisão o instante em que ocorre a centelha de ignição, ao mesmo tempo em que se controla o término da injeção de combustível. Com

isto, evita-se que o combustível seja injetado com grandes atrasos em acelerações com carga[24], e ainda há um controle maior do ponto em que ocorre a centelha de ignição.

Como parâmetros, o Gerenciamento necessita determinar o instante em que os sinais de injeção ou ignição devem ser acionados, levando em conta o tempo de acionamento[25] e o instante em que o sinal deve ser desativado. De modo prático, o Gerenciamento envia ao Sincronismo uma posição de dente (que corresponde a uma borda de subida do sinal de rotação) e um tempo de disparo para o acionamento do sinal, uma vez que não necessariamente o sinal de injeção ou ignição será acionado exatamente em uma borda de subida do sinal de rotação. Abaixo segue um trecho comentado do *firmware* com o cálculo dos parâmetros de acionamento da ignição[26]:

```
n = tempo_bob_temp/periodo_temp;
ig_14_temp = ig_ref[address][address_2] - n;
if (ig_14_temp < 1.0) ig_14_temp += 60.0;
ig_dente_14_temp = (unsigned char)ig_14_temp;
if ((ig_dente_14_temp == 59) || (ig_dente_14_temp == 60))
                ig_dente_14_temp = 58; // Para o caso em que o sinal
                    sobe na falha
else if ((ig_dente_14_temp == 29) || (ig_dente_14_temp == 30))
                ig_dente_14_temp = 28; // Para o caso em que o sinal de
                    ign 23 sobe na falha
ig_tmr_disparo_temp = (ig_14_temp - (float)ig_dente_14_temp)*
    periodo_temp;
```

O Gerenciamento se baseia em uma tabela armazenada em sua memória de programa para determinar os instantes em que os sinais de injeção ou ignição devem ser desativados. No caso da injeção, estes instantes variam de acordo com a rotação. Devido a difícil modelagem destes instantes, optou-se por mapeá-los, com base em medições realizadas com a ECU original do veículo. Neste caso, as aquisições foram feitas a cada 500 RPM, porém os dados foram interpolados a fim de aumentar a resolução da tabela para 100 RPM.

Tabela 3 – Mapa dos instantes de injeção

Rotação (RPM)	**Avanço (dente)**	**Avanço (graus)**
100	12.42	9.462
200	12.42	9.462
300	12.42	9.462
400	12.42	9.462
500	12.42	9.462

continua na próxima página

[24] Geralmente acelerações com carga são caracterizadas por tempos elevados de injeção.

[25] O tempo de acionamento corresponde ao tempo de injeção, no caso da injeção, e ao tempo de carregamento da bobina, no caso da ignição.

[26] Este cálculo é análogo para injeção.

continuação da página anterior

600	12.42	9.462
700	12.42	9.462
800	12.42	9.462
900	12.42	9.462
1000	12.42	9.462
1100	12.60	8.3973
1200	12.78	7.3044
1300	12.97	6.1851
1400	13.16	5.0412
1500	13.35	3.8745
1600	13.55	2.6868
1700	13.75	1.4799
1800	13.96	0.2556
1900	14.16	-0.9843
2000	14.37	-2.238
2100	14.58	-3.5037
2200	14.80	-4.7796
2300	15.01	-6.0639
2400	15.23	-7.3548
2500	15.44	-8.6505
2600	15.66	-9.9492
2700	15.87	-11.2491
2800	16.09	-12.5484
2900	16.31	-13.8453
3000	16.52	-15.138
3100	16.74	-16.4247
3200	16.95	-17.7036
3300	17.16	-18.9729
3400	17.37	-20.2308
3500	17.58	-21.4755
3600	17.78	-22.7052
3700	17.99	-23.9181
3800	18.19	-25.1124
3900	18.38	-26.2863
4000	18.57	-27.438
4100	18.76	-28.5657
4200	18.94	-29.6676

continua na próxima página

continuação da página anterior

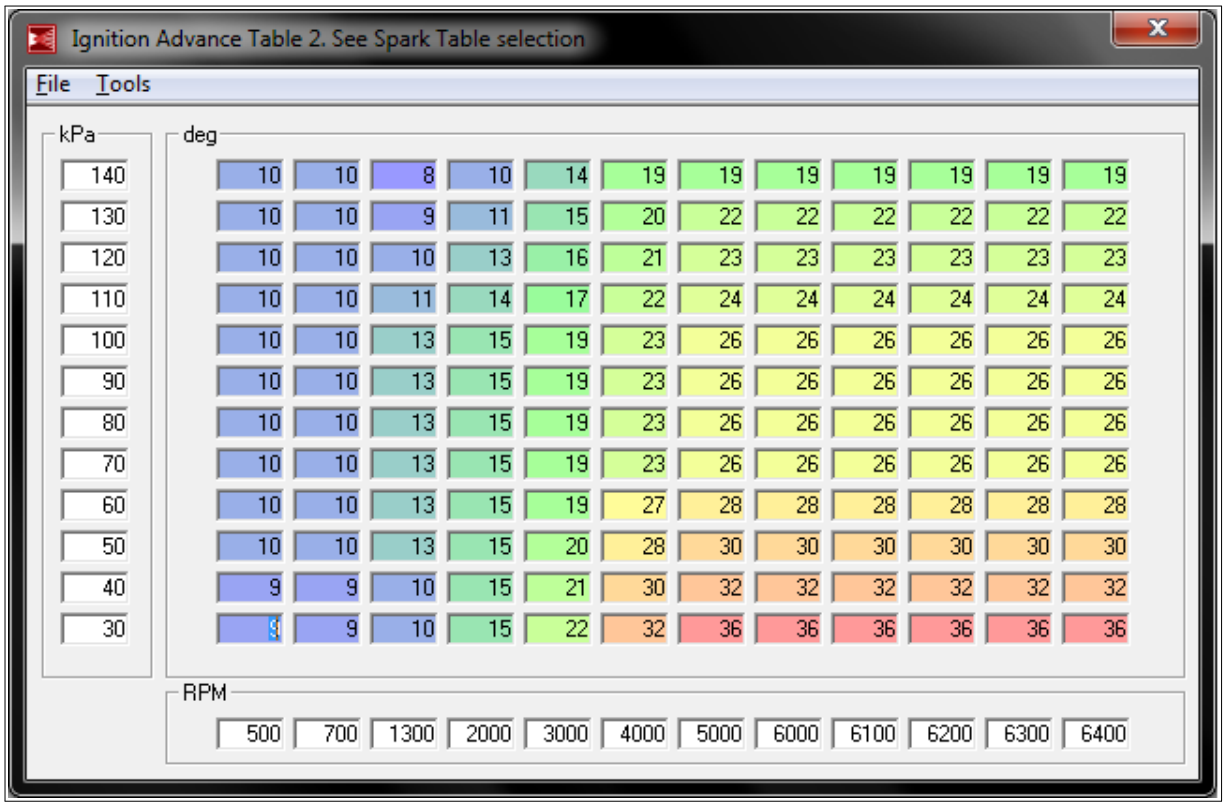
4300	19.12	-30.7419
4400	19.30	-31.7868
4500	19.47	-32.8005
4600	19.63	-33.7812
4700	19.79	-34.7271
4800	19.94	-35.6364
4900	20.08	-36.5073
5000	20.22	-37.338
5100	20.35	-38.1267
5200	20.48	-38.8716
5300	20.60	-39.5709
5400	20.70	-40.2228
5500	20.80	-40.8255
5600	20.90	-41.3772
5700	20.98	-41.8761
5800	21.05	-42.3204
5900	21.12	-42.7083
6000	21.17	-43.038
6100	21.22	-43.3077
6200	21.25	-43.5156
6300	21.28	-43.6599
6400	21.29	-43.7388
6500	200.00	-

O instante em que o sinal de ignição é desativado (instante em que ocorre a centelha) deve ser adiantado quando se aumenta a rotação. Todavia, este adiantamento também é dependente da carga imposta ao motor, esta que é estimada com base na pressão da massa de ar admitida pelo motor. Assim, o adiantamento de ignição depende tanto da rotação como também da pressão do ar. Da mesma forma que na injeção, os instantes (adiantamentos) de ignição são bastante complicados de serem modelados, e ainda variam de acordo com tipo de motor, combustível, e outros fatores. Dada esta dificuldade, optou-se inicialmente por tratar os adiantamentos de ignição apenas em função da rotação, e então foi adotado o mesmo procedimento realizado para mapear os instantes de injeção. Contudo, dado o baixo desempenho do motor nesta situação, surgiu a necessidade de se pesquisar um mapa de ignição que fosse próprio deste motor e que ao mesmo tempo fornecesse resultados consistentes com os medidos com a ECU original[27], para efeitos de

[27] Neste caso, as medições foram realizadas apenas com o motor em vazio, o que representa baixa carga

segurança. Como resultado, foi encontrado o mapa de ignição abaixo, correspondente ao mapa de um motor Volkswagen 2.0L 8v de um veículo modelo polo[28]:

Figura 61 – Mapa de ignição de um motor Volkswagen 2.0L 8v



Fonte: ClubPolo forums (2013)

Apesar de ter contribuído para uma melhora significativa do desempenho do motor, o mapa acima apresentou problemas com relação à detonação. Os testes revelaram que o mapa de ignição estava muito adiantado na faixa de rotação acima de 3000 RPM, uma vez que foi constatado um pequeno ruído de detonação[29] nesta faixa. Com isto, o mapa precisou ser reajustado para correção do problema.

Tabela 4 – Mapa de ignição (em graus) com problema de detonação

RPM kPa	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**110**	**120**
500	9	9	10	10	10	10	10	10	10	10
700	9	9	10	10	10	10	10	10	10	10
1300	10	10	13	13	13	13	13	13	11	10

continua na próxima página

e, consequentemente, uma baixa pressão de ar.

[28] É importante mencionar que o mapa da figura não corresponde ao mapa de ignição do fabricante da ECU original do veículo, dado que este corresponde a um segredo industrial e por este motivo não é revelado pelo fabricante.

[29] Também conhecido popularmente como "'batida de pino"'.

continuação da página anterior

2000	15	15	15	15	15	15	15	15	14	13
3000	22	21	20	19	19	19	19	19	17	16
4000	32	30	28	27	23	23	23	23	22	21
5000	36	32	30	28	26	26	26	26	24	23
6000	36	32	30	28	26	26	26	26	24	23

Tabela 5 – Mapa de ignição (em graus) corrigido

RPM kPa	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**100**	**110**	**120**
500	6	6	7	7	7	7	7	7	7	7
700	9	9	10	10	10	10	10	10	10	10
1300	10	10	13	13	13	13	13	13	11	10
2000	15	15	15	15	15	15	15	15	14	13
3000	22	21	20	19	16	16	16	16	16	16
4000	29	27	24	21	18	18	18	18	18	18
5000	31	29	27	24	19	19	19	19	19	19
6000	31	29	27	24	19	19	19	19	19	19

Os adiantamentos fornecidos em graus na tabela acima foram convertidos para posição de dente, utilizando-se a informação de que o PMS do motor Volkswagen deste projeto encontra-se no dente 14. Além disto, para se melhorar a resolução da tabela, optou-se por utilizar uma interpolação 2D, com auxílio da ferramenta MATLAB.

```
%% Script para interpolar a matriz de avanco da ignicao
% Usa interp2 e meshgrid
% Para usar o script e necessario que os dados tenham sido importados
    para
% o vetor "data"
% Os valores calculados estarao em Zi

rot = [700 1300 2000 3000 4000 5000]; % vetor de rotacoes da matriz
    original
pa = 30:10:120; % vetor de pressoes da matriz original
rot_i = 700:100:5000; % vetor de rotacoes da matriz interpolada
pa_i = pa; % vetor de pressoes da matriz interpolada

[X,Y] = meshgrid(pa, rot);
Z = data;
[Xi, Yi] = meshgrid(pa_i, rot_i);
Zi = interp2(X,Y,Z,Xi,Yi, 'cubic');
```

```
figure(1)
mesh(X,Y,Z)
title('Mapa de çãignio original')
xlabel('ãPresso (kPa)')
ylabel('çãRotao (RPM)')
zlabel('Dente de êreferncia')
figure(2)
mesh(Xi,Yi,Zi)
title('Mapa de çãignio interpolado')
xlabel('ãPresso (kPa)')
ylabel('çãRotao (RPM)')
zlabel('Dente de êreferncia')
```

A tabela final, com resolução na rotação de 100 RPM e resolução na pressão de 10kPa, foi disponibilizada abaixo[30]:

Tabela 6 – Mapa de ignição final (em posição de dente)

RPM kPa	30	40	50	60	70	80	90	100	110	120
100	14.50	14.50	14.50	14.50	14.50	14.50	14.50	14.50	14.50	14.50
200	14.50	14.50	14.50	14.50	14.50	14.50	14.50	14.50	14.50	14.50
300	14.50	14.50	14.50	14.50	14.50	14.50	14.50	14.50	14.50	14.50
400	14.10	14.10	14.10	14.10	14.10	14.10	14.10	14.10	14.10	14.10
500	13.10	13.10	13.10	13.10	13.10	13.10	13.10	13.10	13.10	13.10
600	13.10	13.10	13.10	13.10	13.10	13.10	13.10	13.10	13.10	13.10
700	12.50	12.50	12.33	12.33	12.33	12.33	12.33	12.33	12.33	12.33
800	12.53	12.54	12.21	12.21	12.23	12.23	12.23	12.23	12.34	12.38
900	12.54	12.55	12.11	12.11	12.14	12.14	12.14	12.14	12.34	12.41
1000	12.52	12.53	12.02	12.03	12.05	12.05	12.05	12.05	12.31	12.42
1100	12.47	12.48	11.95	11.95	11.97	11.97	11.97	11.97	12.28	12.40
1200	12.41	12.41	11.88	11.89	11.90	11.90	11.90	11.90	12.23	12.37
1300	12.33	12.33	11.83	11.83	11.83	11.83	11.83	11.83	12.17	12.33
1400	12.23	12.23	11.78	11.78	11.77	11.77	11.77	11.77	12.10	12.27
1500	12.13	12.12	11.74	11.73	11.71	11.71	11.71	11.71	12.03	12.21
1600	12.01	12.00	11.69	11.69	11.66	11.66	11.66	11.66	11.96	12.14
1700	11.88	11.88	11.65	11.65	11.61	11.61	11.61	11.61	11.88	12.06
1800	11.76	11.75	11.61	11.60	11.57	11.57	11.57	11.57	11.81	11.98
1900	11.63	11.62	11.56	11.55	11.53	11.53	11.53	11.53	11.74	11.91
2000	11.50	11.50	11.50	11.50	11.50	11.50	11.50	11.50	11.67	11.83
2100	11.38	11.38	11.44	11.44	11.47	11.47	11.47	11.47	11.61	11.76

continua na próxima página

[30] Os adiantamentos foram dados em posição de dente, ao invés de graus.

continuação da página anterior

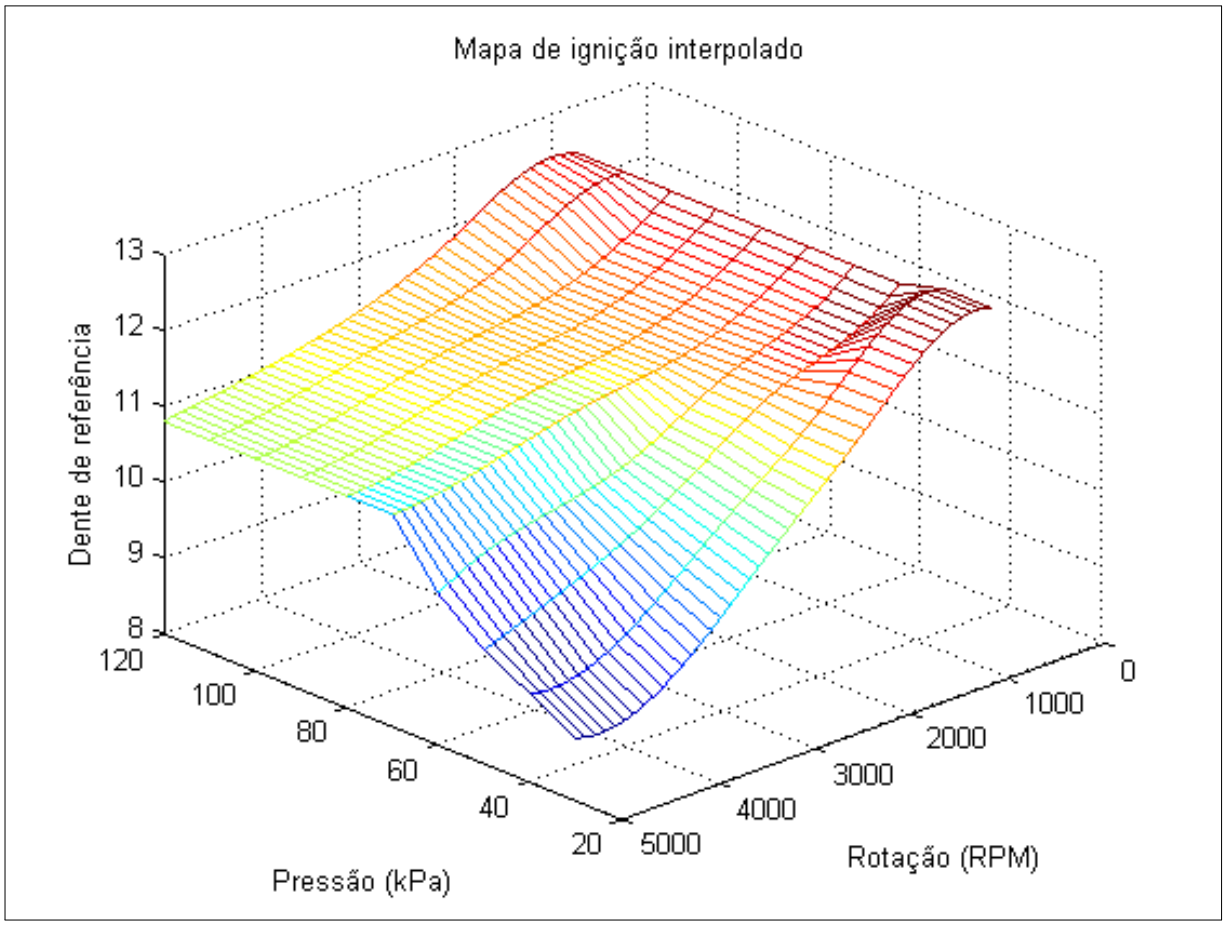
2200	11.26	11.28	11.36	11.38	11.45	11.45	11.45	11.45	11.56	11.69
2300	11.14	11.17	11.28	11.31	11.43	11.43	11.43	11.43	11.52	11.63
2400	11.02	11.07	11.20	11.24	11.41	11.41	11.41	11.41	11.48	11.57
2500	10.91	10.98	11.11	11.16	11.39	11.39	11.39	11.39	11.44	11.52
2600	10.79	10.88	11.02	11.09	11.37	11.37	11.37	11.37	11.41	11.47
2700	10.68	10.79	10.93	11.02	11.36	11.36	11.36	11.36	11.38	11.42
2800	10.56	10.69	10.84	10.95	11.34	11.34	11.34	11.34	11.35	11.38
2900	10.45	10.60	10.76	10.89	11.32	11.32	11.32	11.32	11.33	11.34
3000	10.33	10.50	10.67	10.83	11.30	11.30	11.30	11.30	11.30	11.30
3100	10.21	10.40	10.59	10.78	11.28	11.28	11.28	11.28	11.27	11.26
3200	10.09	10.30	10.51	10.73	11.25	11.25	11.25	11.25	11.24	11.23
3300	9.96	10.19	10.44	10.70	11.22	11.22	11.22	11.22	11.21	11.19
3400	9.84	10.09	10.37	10.66	11.19	11.19	11.19	11.19	11.18	11.16
3500	9.72	9.98	10.30	10.63	11.16	11.16	11.16	11.16	11.15	11.13
3600	9.60	9.88	10.23	10.60	11.13	11.13	11.13	11.13	11.12	11.10
3700	9.48	9.78	10.17	10.58	11.10	11.10	11.10	11.10	11.09	11.08
3800	9.37	9.68	10.11	10.55	11.06	11.06	11.06	11.06	11.06	11.05
3900	9.27	9.59	10.06	10.53	11.03	11.03	11.03	11.03	11.03	11.02
4000	9.17	9.50	10.00	10.50	11.00	11.00	11.00	11.00	11.00	11.00
4100	9.08	9.42	9.95	10.47	10.97	10.97	10.97	10.97	10.97	10.98
4200	9.00	9.35	9.89	10.44	10.94	10.94	10.94	10.94	10.94	10.95
4300	8.93	9.28	9.84	10.41	10.91	10.91	10.91	10.91	10.92	10.93
4400	8.87	9.23	9.79	10.37	10.89	10.89	10.89	10.89	10.90	10.91
4500	8.83	9.19	9.74	10.33	10.86	10.86	10.86	10.86	10.87	10.89
4600	8.80	9.16	9.70	10.28	10.84	10.84	10.84	10.84	10.85	10.87
4700	8.78	9.14	9.65	10.22	10.83	10.83	10.83	10.83	10.84	10.85
4800	8.78	9.13	9.60	10.16	10.81	10.81	10.81	10.81	10.82	10.84
4900	8.80	9.14	9.55	10.08	10.80	10.80	10.80	10.80	10.81	10.82
5000	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
5100	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
5200	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
5300	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
5400	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
5500	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
5600	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
5700	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
5800	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80

continua na próxima página

continuação da página anterior

5900	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
6000	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
6100	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
6200	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
6300	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
6400	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80
6500	8.83	9.17	9.50	10.00	10.80	10.80	10.80	10.80	10.80	10.80

Figura 62 – Grafico 3D do mapa de ignição final



Fonte: o autor

Com o ajuste do mapa de ignição, não se constatou mais nenhum problema relacionado à detonação, e o motor foi capaz de atingir 6000 RPM mesmo em condições de carga.

Além da injeção, ignição e válvula borboleta, o Gerenciamento é responsável também pelo controle do relé da bomba de combustível. A estratégia de acionamento deste

relé é bastante simples. Após o ligamento da linha 15, o Gerenciamento pressuriza a linha de combustível, ativando o relé por aproximadamente 0,5s. Após isto, o relé é desligado e é religado apenas na primeira detecção da falha da roda fônica[31], e então permanece ligado durante o funcionamento do motor. Caso o motor pare de funcionar por algum motivo, o relé é desligado.

Por fim, menciona-se ainda a implementação de um filtro para suavizar acelerações e desacelerações bruscas, característica inerente do pedal eletrônico (*drive-by-wire*). Este filtro foi implementado com base em limitações da referência de rotação calculada a partir da leitura do pedal de aceleração, levando em conta também valores anteriores desta referência. Nota-se que este filtro é ativado somente quando o veículo se encontra em movimento, caracterizado por uma velocidade maior do que 5 km/h[32].

```
// Filtro para impedir aceleracoes bruscas...
if (movimento != 0) {
                if (((ref_rpm_temp - ref_rpm) > 25.0) && (ref_rpm_temp >
                     rotacao))
                                ref_rpm_temp = ref_rpm + 25.0; //
                                    Suaviza aceleracao
                else if (((ref_rpm - ref_rpm_temp) > 50.0) && (
                   ref_rpm_temp < rotacao))
                                ref_rpm_temp = ref_rpm - 50.0; //
                                    Suaviza desaceleracao
}

ref_rpm = ref_rpm_temp; // Passa p/ a variavel final
```

Sumarizando os recursos utilizados no microcontrolador, o Gerenciamento utiliza quatro *timers* de 16 bits cada um, duas comunicações SPI, dois canais de PWM com 8 bits cada um, 13 canais analógicos para conversão A-D, três interrupções externas, além de diversos pinos digitais. Para mais detalhes, um fluxograma do *firmware* do uC de Gerenciamento pode ser consultado nos apêndices deste trabalho.

3.2.2.2 Sincronismo

O *firmware* do uC de Sincronismo contém a estratégia para acionar precisamente os sinais de injeção e ignição. Para isto, este microcontrolador utiliza os parâmetros de injeção e ignição calculados pelo uC de Gerenciamento, parâmetros estes enviados através do protocolo SPI.

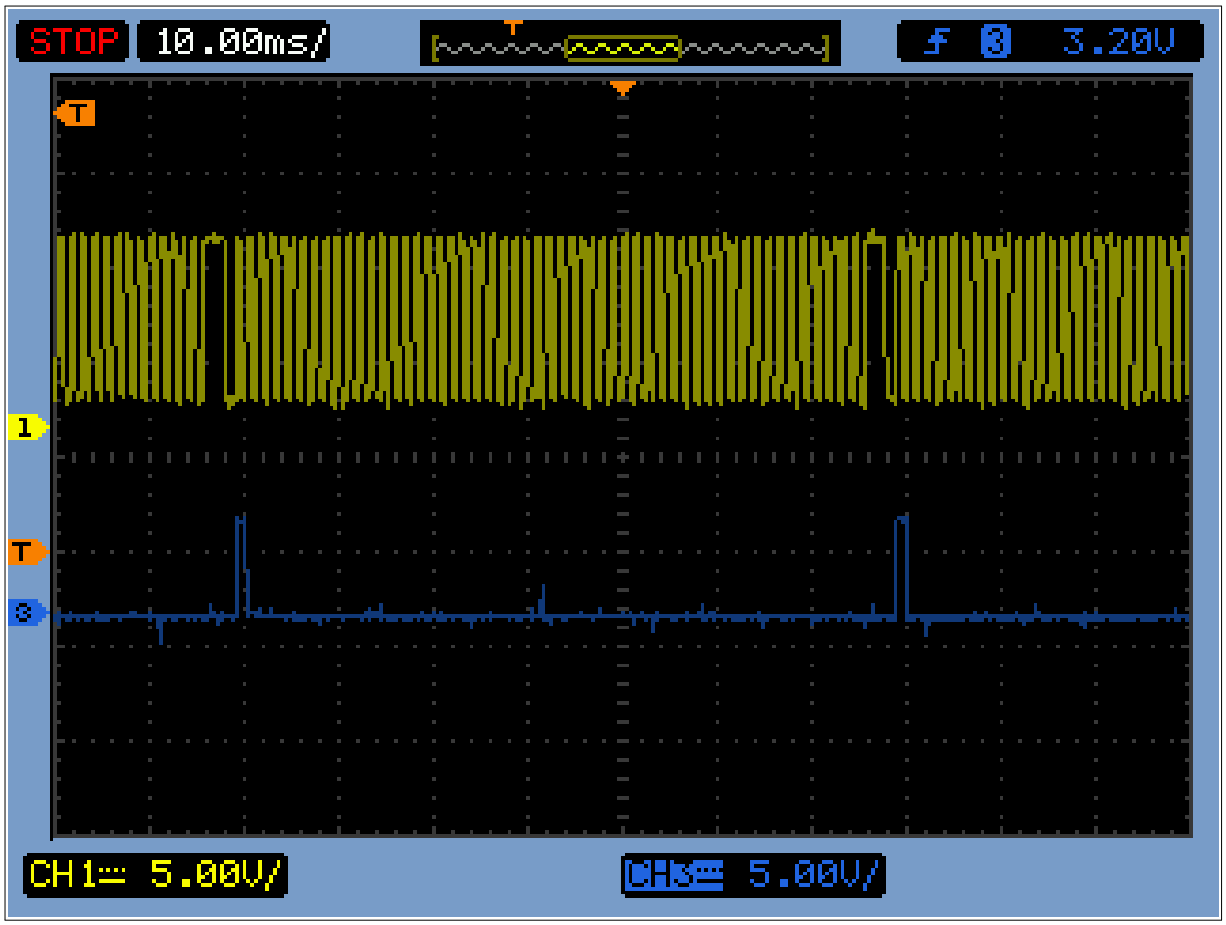
Analogamente ao Gerenciamento, o Sincronismo deve manter sincronia com o sinal de rotação, proveniente da associação da roda fônica com o sensor de relutância variável.

[31] Neste ponto a partida já foi acionada dado que já existe movimento da roda fônica.

[32] A respectiva determinação de movimento, considerando velocidades maiores do que 5 km/h, é realizada no uC de Comunicação, sendo o resultado repassado ao uC de Gerenciamento.

Para isto, o Sincronismo deve detectar a falha[33] da roda fônica, a fim de reiniciar contagens e atualizar parâmetros para a próxima volta.

Figura 63 – Sinal de rotação e sinal de detecção da falha da roda fônica

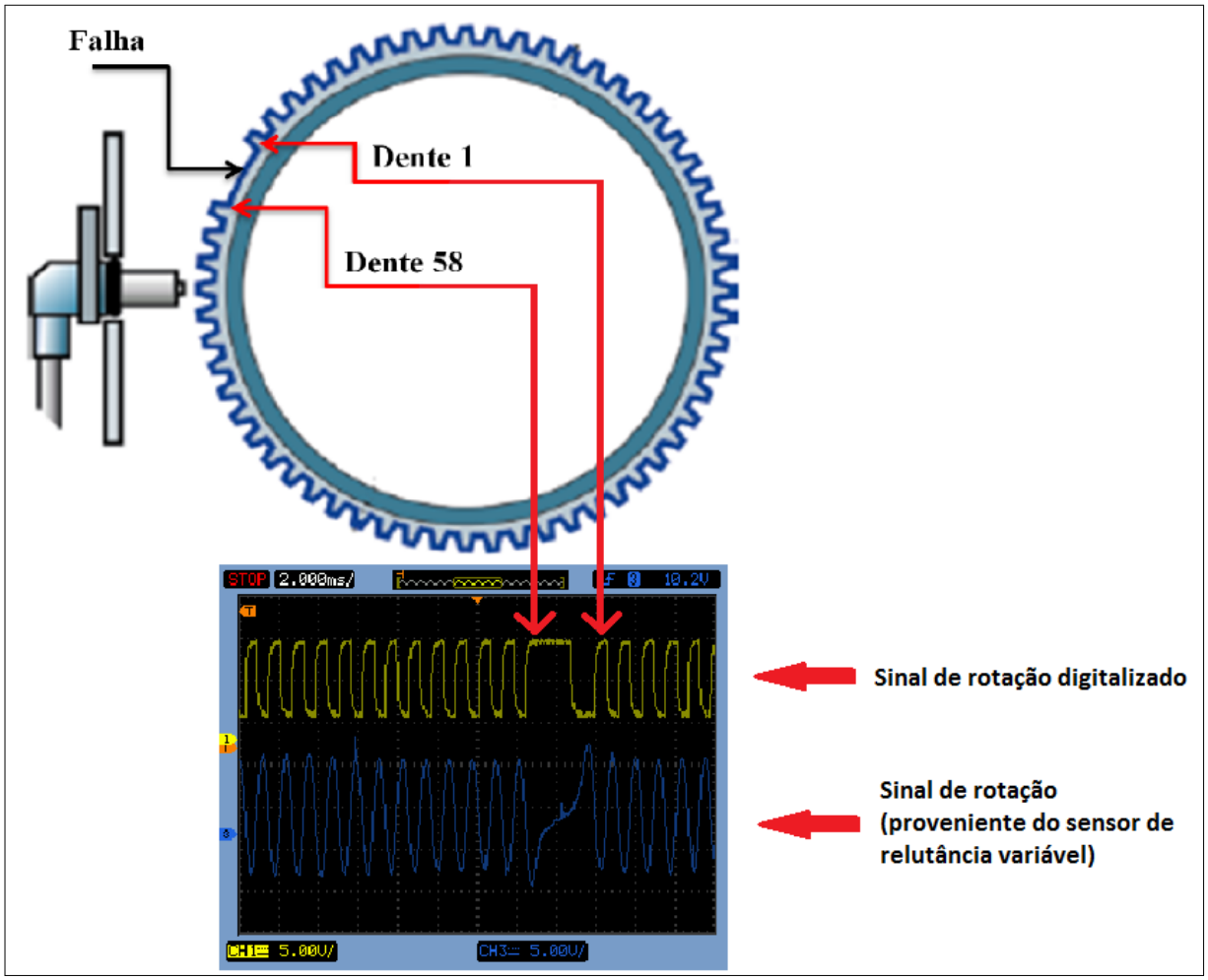


Fonte: o autor

O sinal de rotação está associado à posição do dente da roda fônica. Dado este fato, o programa detecta e processa cada transição de subida do sinal de rotação. Diferentemente do Gerenciamento, que necessita apenas dos 3 primeiros dentes para calcular a rotação, o Sincronismo necessita contar todas as transições de subida do sinal de rotação, ou seja, necessita identificar todos os dentes para disparar os sinais de injeção e ignição no momento correto. Deste modo é importante que o Sincronismo não execute muito cálculos (ao contrário do Gerenciamento), uma vez que contas complexas (como multiplicação e divisão com ponto flutuante) levariam muito tempo para serem executadas e, consequentemente, alguns dentes poderiam não ser contabilizados (ou poderiam ser contabilizados com atraso). Este fato ainda se agravaria com o aumento da rotação, dado que o período de um dente diminuiria significativamente.

[33] Correspondente à ausência de dois dentes.

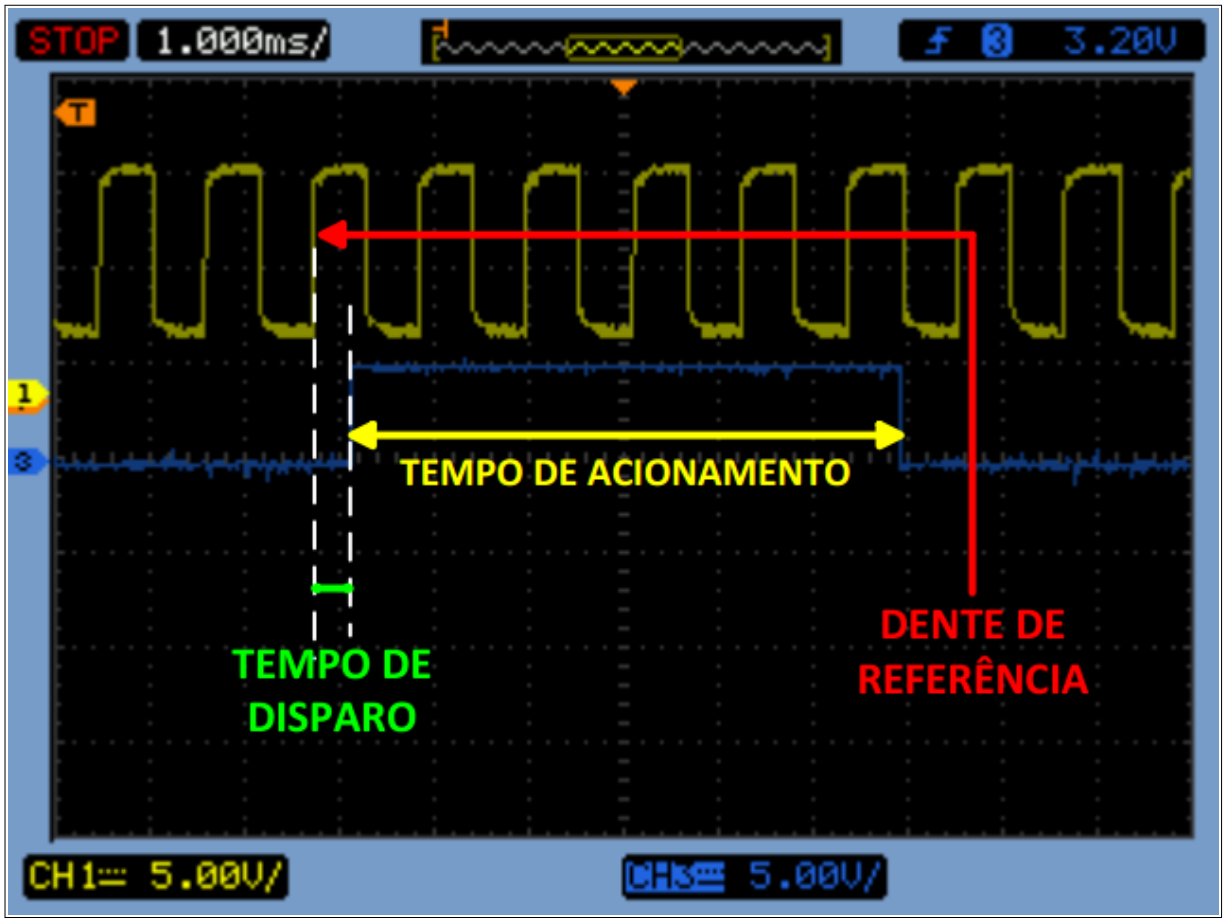
Figura 64 – Relação da roda fônica com o sinal de rotação



Fonte: adaptado de Albaladejo (2013)

Os parâmetros recebidos do Gerenciamento correspondem a duas posições de dente (um para injeção e outro para ignição), dois tempos de disparo a partir deste dente (um para injeção e outro para ignição), dois tempos de acionamento (tempo de injeção e tempo de carregamento da bobina de ignição) e a rotação. Uma vez recebido os parâmetros da volta, o Sincronismo verifica se o dente atual (identificado pela contagem das bordas de subida do sinal de rotação) corresponde a um dos dentes recebidos. Em caso afirmativo, o programa habilita um *timer* para gerar uma interrupção após o tempo de disparo recebido. Uma vez gerada a interrupção, o sinal correspondente (injeção ou ignição) é ativado, e permanece assim por um tempo determinado pelo tempo de acionamento, que também é programado na interrupção do mesmo *timer*. Assim, após esta última interrupção, o sinal que estava ativado é então desativado.

Figura 65 – Sinal de ignição com ilustração dos parâmetros de acionamento



Fonte: o autor

É importante mencionar que os mesmos tempos (disparo e acionamento) são utilizados para a injeção e ignição dos quatros cilindros, diferindo apenas no fato do dente de referência para os sinais referentes aos cilindros 2 e 3 serem defasados de 30 dentes (o que equivale a 180°) em relação aos cilindros 1 e 4. Abaixo segue um trecho da rotina que realiza a contagem dos dentes e verifica se deve ocorrer acionamento de injeção ou ignição.

```
void conta_dente() {
    dente++; // Contagem das dentes

                ...

    if ((dente == ig_dente_14) && (ig14_pos == 0)) {
        if (ig_tmr_disparo > 9) {
            PITLD0 = ig_tmr_disparo;
            PITFLT_PFLT0 = 1;
            PITTF = (1<<0); // Limpa flag PTF0
            PITINTE_PINTE0 = 1; // Habilita interrupcao timer 0
            ig14_pos = 14;
```

```
        }
    }

    if ((dente == ig_dente_23) && (ig23_pos == 0)) {
        if (ig_tmr_disparo > 9) {
            PITLD3 = ig_tmr_disparo;
            PITFLT_PFLT3 = 1;
            PITTF = (1<<3); // Limpa flag PTF3
            PITINTE_PINTE3 = 1; // Habilita interrupcao timer 3
            ig23_pos = 23;
        }
    }

    // Se nao ident. a fase uma vez nao deixa dar os sinais de inj
    if (rotacao > 6500 || fase_ok == 0) // O GER corta a inj, mas esta
        condicao reforca o corte (seguranca)
        return;

    if ((dente == ij_dente_14) && (!volta14) && (ij14_pos == 0)) {
        if (ij_tmr_disparo > 9) {
            PITLD1 = ij_tmr_disparo;
            PITFLT_PFLT1 = 1;
            PITTF = (1<<1); // Limpa flag PTF1
            PITINTE_PINTE1 = 1; // Habilita interrupcao timer 1
            ij14_pos = 1;
        }
    }

    if ((dente == ij_dente_23) && (volta23) && (ij23_pos == 0)) {
        if (ij_tmr_disparo > 9) {
            PITLD2 = ij_tmr_disparo;
            PITFLT_PFLT2 = 1;
            PITTF = (1<<2); // Limpa flag PTF2
            PITINTE_PINTE2 = 1; // Habilita interrupcao timer 2
            ij23_pos = 2;
        }
    }

    if ((dente == ij_dente_23) && (!volta23) && (ij23_pos == 0)) {
        if (ij_tmr_disparo > 9) {
            PITLD2 = ij_tmr_disparo;
            PITFLT_PFLT2 = 1;
            PITTF = (1<<2); // Limpa flag PTF2
            PITINTE_PINTE2 = 1; // Habilita interrupcao timer 2
            ij23_pos = 3;
        }
    }
```

```
    if ((dente == ij_dente_14) && (volta14) && (ij14_pos == 0)) {
        if (ij_tmr_disparo > 9) {
            PITLD1 = ij_tmr_disparo;
            PITFLT_PFLT1 = 1;
            PITTF = (1<<1); // Limpa flag PTF1
            PITINTE_PINTE1 = 1; // Habilita interrupcao timer 1
            ij14_pos = 4;
        }
    }
    erro = erro & 0b11111101;

} // void conta_dente()
```

Com relação aos modos de acionamento, adotou-se para ignição a centelha perdida, ou seja, os sinais de ignição são acionados simultaneamente em cada banco[34], o que facilita a estratégia de ignição. Já a injeção é acionada apenas uma vez em cada cilindro, a cada duas voltas, o que caracteriza a injeção multiponto sequencial. Para isto, deve-se detectar a fase de injeção, ou seja, em um dada volta o Sincronismo necessita identificar em qual dos cilindros do banco deve ocorrer a injeção. Assim, a estratégia adotada consiste em se identificar a fase do motor quando a rotação está abaixo de 600 RPM (partida), e então continuar acionando a injeção de modo a manter sempre a sequência 1-3-4-2. Caso ocorra perda da fase, o Sincronismo é capaz de corrigi-la, tomando como base a descida do sinal de injeção (que é mapeada) e o sinal de fase. Em outras palavras, quando o sinal de injeção é desativado, o Sincronismo verifica se o sinal de fase é aquele esperado e, em caso negativo, inverte a variável que controla a sequencia de injeção, a fim de corrigir a sua fase. Como exemplo de implementação, foi destacado a seguir um trecho do Sincronismo com a interrupção do *timer* que controla a injeção do banco 1-4. É possível verificar também a implementação da estratégia para correção da fase de injeção.

```
__interrupt void TIMER_INJ14_PIT1(void)
{
    PITINTE_PINTE1 = 0; // Desabilita int timer 1

    /* Pulsos de injecao p sequencial*/
    switch(ij14_pos) {

        case 0:
                break;

        case 10:
                SINAL_IJ1 = 0; /* Baixa o Sinal da Injecao 1 */

                if (rotacao >= 600 && rotacao_ant >= 600) {
```

[34] Um banco é composto pelos cilindros 1 e 4, e o outro, pelos cilindros 2 e 3.

```
                volta14 = ~volta14;

                // Verifica a logica da fase...
                if (rotacao < 2000) { // OBS: Lembrar que a falha eh
                    detectada no primeiro dente!
                    // Neste caso o sinal de injecao desce DEPOIS da
                        falha
                    if (sinal_fase == 0) // a fase estava errada!!!
                        volta14 = ~volta14;
                }
                else {
                    // Neste caso o sinal de injecao desce ANTES da
                       falha
                    if (sinal_fase == 1) // a fase estava errada!!!
                        volta14 = ~volta14;
                }
            }

            ij14_pos = 0;
            break;

    case 40:
            SINAL_IJ4 = 0; /* Baixa o Sinal da Injecao 4 */

            if (rotacao >= 600 && rotacao_ant >= 600)
                volta14 = ~volta14;

            ij14_pos = 0;
            break;

    case 1:
            if (tempo_inj > 50) {

                SINAL_IJ1 = 1; /* Ativa Sinal da injecao 1 */

                /* Conta e entao baixa o sinal de injecao por
                   interrupcao */
                PITLD1 = tempo_inj;
                PITFLT_PFLT1 = 1;
                PITTF = (1<<1); // Limpa flag PTF1
                PITINTE_PINTE1 = 1; // Habilita interrupcao timer 1

                ij14_pos = 10; // Proxima interrupcao vai desligar o
                    sinal
            }
            else
                ij14_pos = 0;
```

```
                break;

        case 4:
                if (tempo_inj > 50) {

                    SINAL_IJ4 = 1; /* Ativa Sinal da injecao 4 */

                    /* Conta e entao baixa o sinal de injecao por
                       interrupcao */
                    PITLD1 = tempo_inj;
                    PITFLT_PFLT1 = 1;
                    PITTF = (1<<1); // Limpa flag PTF1
                    PITINTE_PINTE1 = 1; // Habilita interrupcao timer 1

                    ij14_pos = 40; // Proxima interrupcao vai desligar o
                        sinal
                }
                else
                    ij14_pos = 0;
                break;
    }
    erro = erro & 0b11101111;
}
```

Sumarizando os recursos utilizados no microcontrolador, o Sincronismo utiliza cinco *timers* de 16 bits cada um, duas comunicações SPI, duas interrupções externas, e diversos pinos digitais. Para mais detalhes, um fluxograma do *firmware* do Sincronismo pode ser consultado nos apêndices deste trabalho.

3.2.2.3 Comunicação

O uC de Comunicação é, de modo geral, responsável por estabelecer a conexão entre a ECU e outros dispositivos externos, como computadores, aparelhos automotivos de diagnose ou até outras ECUs presentes no veículo[35]. Deste modo, o uC de Comunicação pode ser visto como uma verdadeira ponte que interliga a ECU deste projeto ao meio externo.

O bloco de Comunicação trabalha com auxílio de três protocolos de comunicação: UART, SPI e CAN (os detalhes de cada protocolo podem ser encontrados no capítulo 2 deste trabalho). O SPI é utilizado quando a comunicação ocorre internamente, ou seja, entre os blocos do sistema. Neste caso, a troca de dados necessita ocorrer de forma rápida, interferindo o menor tempo possível no Gerenciamento ou no Sincronismo, fato que justifica o uso de um protocolo simples e rápido, como o SPI. No âmbito deste projeto, a comunicação SPI foi configurada para operar em 12,5 MHz.

[35] Como, por exemplo, a ECU do painel do motorista.

Para se comunicar com a aplicação (*software*) de monitoramento, o bloco de Comunicação utiliza o protocolo UART, dado que este é um protocolo de fácil integração no *software* e possui suporte no ambiente *Windows*. Para facilitar conexões com computadores que não possuem a porta serial RS-232, foi empregado um conversor UART/USB (descrito anteriormente), responsável por realizar a conversão entre os protocolos. Assim o projetista não precisa se preocupar com os detalhes de programação do protocolo USB, sendo ainda possível com um simples cabo USB ligar a ECU v2.0 ao *software*, o que dispensa o uso de cabos RS-232 que são mais raros no mercado. No caso deste projeto, utiliza-se uma taxa de transmissão de 125000b/s, suficiente para a comunicação ocorrer de forma rápida e confiável, sem perda relevante de dados.

Quando se necessita que a ECU envie um dado ao painel do motorista ou ainda a um aparelho automotivo de diagnose, utiliza-se o protocolo CAN. Especificamente neste projeto, o protocolo CAN é utilizado para se enviar o valor da rotação ao painel do motorista e para leitura da velocidade do veículo, utilizando para isto uma taxa de transmissão de 500kb/s.

Para requisitar um dado ao bloco de Gerenciamento ou Sincronismo, o uC de Comunicação utiliza um protocolo bastante simples:

Tabela 7 – Protocolo entre o uC Comunicação e os demais uCs (usa SPI1)

Comunicação	**GER/SINC**	***Byte High***	***Byte Low***
"a"	GER	ref rotação(H)	ref rotação(L)
"b"	GER	rotação(H)	rotação(L)
"c"	GER	TPS	ref borboleta
"d"	GER	pedal	MAP
"e"	GER	Temp Ar	Temp Agua
"f"	GER	Lambda	Bateria
"g"	GER	Erro GER SW	Erro GER HW
"h"	GER	pedal "simulado"(envio)	"h"
"i"	GER	status (envio)	"i"
"j"	SINC	Inj dente ref	Ign dente ref
"k"	SINC	Inj disparo(H)	Inj disparo(L)
"l"	SINC	Ign disparo(H)	Ign disparo(L)
"m"	SINC	Tempo Inj(H)	Tempo Inj(L)
"n"	SINC	Tempo Ign(H)	Tempo Ign(L)
"o"	SINC	Detecção Fase	Erro SINC SW

Quando se deseja um dado, o uC de Comunicação envia um caracter e espera pela resposta, esta que é enviada pelo SPI1. Já quando o Comunicação deseja enviar

um dado (como, por exemplo, o valor do pedal "'simulado"'), ele envia um carácter e envia, ao mesmo tempo, o dado, de modo a aproveitar o fato de que a comunicação SPI no microcontrolador S12XE opera com registradores de 16 bits, ou seja, em uma transmissão é possível se enviar 2 *bytes* de informação de uma só vez (1 *byte* é o caracter e o outro é a informação a ser enviada). Além disto, os microcontroladores de Gerenciamento e Sincronismo respondem às requisições do microcontrolador de Comunicação através de interrupção, atendida no caso pelo coprocessador XGate, de modo que não ocorra interferência no processamento do *core* principal S12X.

Para requisitar um dado ao *software*, o uC de Comunicação utiliza o protocolo abaixo:

Tabela 8 – Protocolo entre o uC Comunicação e a aplicação no PC (usa SCI0)

Aplicação (PC)	**Comunicação (uC)**
"a"	rotação
"b"	pedal (PEDAL1 ou PEDAL2)
"c"	válvula (TPS1 ou TPS2)
"d"	referência de rotação (pedal)
"e"	MAP
"f"	Temperatura do Ar
"g"	Temperatura da Água
"h"	Lambda
"i"	Tensão da Bateria
"j"	referência de posição para a Borboleta
"k"	Injeção dente 14
"l"	Injeção tempo disparo
"m"	Ignição dente 14
"n"	Ignição tempo disparo
"o"	Tempo de Injeção
"p"	Tempo de Carregamento da Bobina de Ignição
"q"	Detecção de Fase
"r"	Erro Gerenciamento Software
"s"	Erro Gerenciamento Hardware
"t"	Erro Sincronismo Software
"u"	Controlar Pedal = True
"v"	Controlar Pedal = False
"w"	Leitura do Pedal Simulado (Enviar)
"x"	Modo de operação "Normal"
"y"	Modo de operação "Econômico"

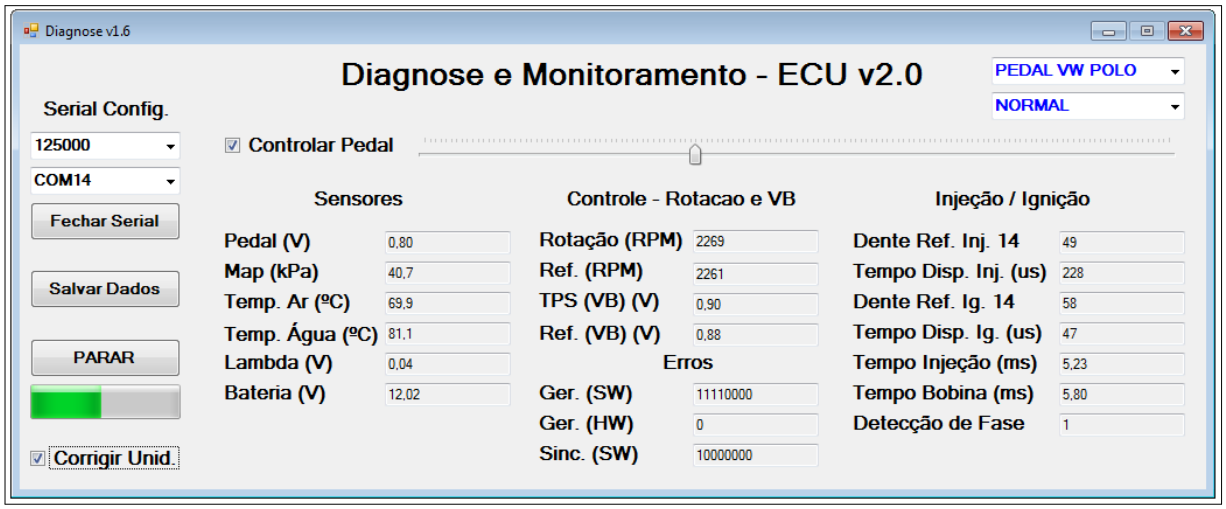
Nota-se que o protocolo opera de maneira semelhante ao protocolo descrito anteriormente, porém neste caso é possível se transmitir apenas 1 *byte* de informação por vez, característica do protocolo serial UART. Deste modo, quando a aplicação envia um caracter da lista acima, o uC de Comunicação responde com a respectiva informação, através de interrupção atendida também pelo XGate.

É importante ressaltar que o bloco de Comunicação opera todo o tempo, independente do regime do motor ou do estado da linha 15. Assim, o monitoramento ocorre normalmente mesmo quando o motor está desligado. Já o painel do motorista recebe a informação via CAN somente com a linha 15 ligada (chave ligada no contato).

Sumarizando os recursos utilizados no microcontrolador, o Comunicação utiliza dois *timers* de 16 bits cada um, duas comunicações SPI, uma comunicação serial UART, uma comunicação CAN e diversos pinos digitais. Para mais detalhes, um fluxograma do *firmware* do Comunicação pode ser consultado nos apêndices deste trabalho.

3.2.3 *Software*

Figura 66 – Aplicação de Monitoramento desenvolvida no Visual C#



Fonte: o autor

Desenvolvida em linguagem C# e concebida para rodar em ambiente *Windows*, a aplicação da figura acima tem a finalidade de monitorar em tempo real parâmetros do sistema, como informações de sensores e parâmetros calculados. O *software* desenvolvido se mostrou bastante útil no que diz respeito à verificação e depuração do sistema. Ao se constatar, por exemplo, uma falha durante os testes no veículo, é possível, pela interface, mapear todos os parâmetros que foram medidos e/ou calculados no momento em que ocorreu o problema, facilitando a reprodução e a correção da falha em laboratório. Para isto, foi integrado à aplicação uma opção para se salvar parte dos dados monitorados, o

que viabiliza também a confecção de tabelas e gráficos com os parâmetros retirados do sistema.

A interface também fornece opção para se alterar o formato de visualização dos valores dos sensores. No caso é possível visualizar o dado lido diretamente pelo microcontrolador (que corresponde a um valor de 8 bits no intervalo 0 a 255), ou ainda é possível visualizar o dado já na unidade associada à variável final[36]. Neste último caso, com a opção "‘Corrigir Unid."’ marcada, a aplicação calcula o valor final com base na curva característica do sensor, tarefa equivalente à do uC de Gerenciamento.

Além de monitoramento, o *software* desenvolvido permite também atuar no sistema, simulando a leitura do pedal de aceleração. Neste caso, o sistema deixa de processar a leitura do pedal do veículo e passa a considerar o valor enviado pela aplicação, valor este determinado com base na posição do controle deslizante do *software*. Vale lembrar que o controle deslizante possui os mesmos limites do pedal do acelerador, de tal forma que a referência de rotação seja equivalente em ambos os casos. Assim, é possível se controlar o motor do veículo diretamente do computador, para fins de testes.

Por fim, a aplicação permite também alterar o modo de operação do controle de rotação. O modo "‘Normal"’ corresponde aos ganhos do controle de rotação em que a ECU v2.0 funciona de maneira similar à ECU original do veículo. Já o modo "‘Econômico"’ corresponde a ganhos menores, de modo que o motor funcione com aberturas menores na borboleta, o que resulta em uma maior economia de combustível, apesar de resultar, ao mesmo tempo, em uma perda de potência. Uma vez alterado este modo de operação no *software*, o uC de Comunicação é avisado pela aplicação e repassa esta "‘mudança"’ ao uC de Gerenciamento, este que, por sua vez, altera o ganho proporcional do controle de rotação. A finalidade desta opção é demonstrar que, uma vez conhecido os detalhes de implementação do sistema, é possível implementar facilmente novas funções em uma ECU automotiva.

Para mais detalhes da implementação do *software*, recomenda-se a consulta ao código fonte, que pode ser encontrado nos apêndices deste trabalho.

[36] Pode-se, por exemplo, ler o valor de temperatura em graus *Celsius*.

4 Resultados e Discussão

O OBJETIVO FINAL DESTE PROJETO consistiu em desenvolver uma unidade de gerenciamento eletrônico que, apesar de ser voltada para desenvolvimentos acadêmicos, pode ao mesmo tempo ser aplicada a um veículo real, para fins de testes. Para isto, a ECU v2.0 desenvolvida neste projeto foi capaz de gerenciar corretamente ignição, injeção, relés e válvula borboleta, a fim de controlar satisfatoriamente a rotação do motor, mesmo em situações de carga. Para que este controle fosse realizado com sucesso, alguns requisitos de engenharia necessitaram ser atendidos. O *hardware* necessitou, por exemplo, filtrar ruídos presentes nos sinais dos sensores, dado que um ruído de magnitude elevada poderia causar um comportamento inesperado no controle do motor, podendo até provocar a sua instabilidade.

Sendo assim, os requisitos de aprovação foram especificados como:

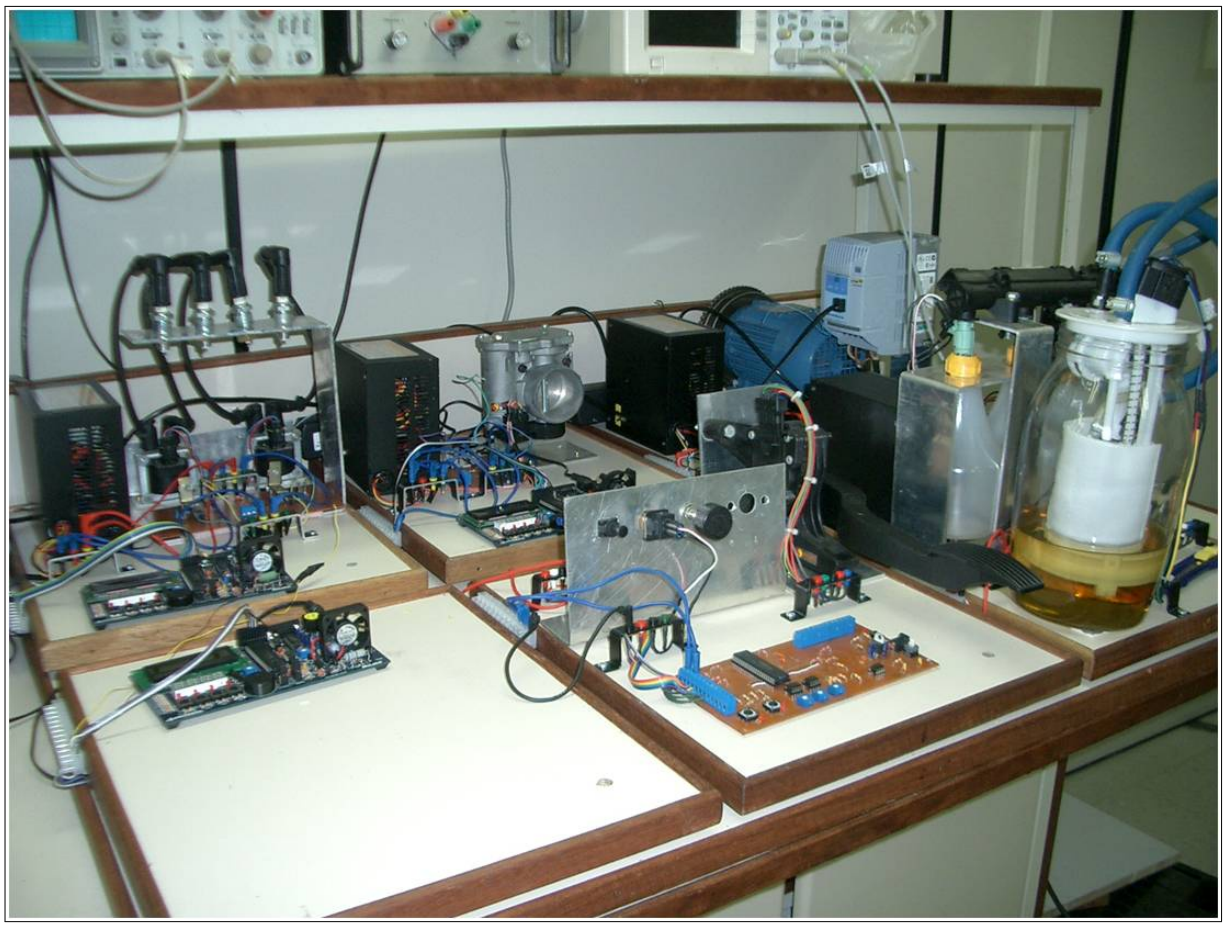
- *Firmware* capaz de manter o funcionamento do motor a até 6000 RPM, mesmo quando o motor estiver fora da sua temperatura de operação (motor frio);
- *Hardware* resistente a ruídos presentes nos sinais dos sensores;
- *Hardware* com processamento da ordem de microssegundos, de forma a possibilitar a execução de todas as tarefas, mesmo em rotações elevadas;
- *Hardware* com alta velocidade de aquisição de dados (capaz de ler todos os sensores em uma volta, mesmo em elevadas rotações) e capaz de gerar saídas compatíveis com os atuadores de um motor real;
- *Hardware* com baixo consumo de energia quando o motor estiver desligado, o suficiente para que a placa possa ser alimentada por uma bateria de um veículo de passeio comercial (12V);
- *Hardware* flexível, de forma que a mesma placa possa ser utilizada para controle de outros motores, no âmbito de projetos futuros. Esta flexibilidade possui certos limites;
- *Hardware* capaz de drenar correntes elevadas provenientes de atuadores;
- *Software* com capacidade de monitorar parâmetros em tempo real, contando com a possibilidade de se controlar o motor a partir do computador, para fins de testes.
- *Firmware* capaz de controlar a rotação do motor em condições de carga, fornecendo uma potência máxima de no mínimo 50% do valor nominal (116 cv), o que corresponde ao valor de 58 cv.

A validação do projeto envolve diversos testes realizados em bancada e testes realizados no veículo, sendo que neste último caso será utilizado um dinamômetro inercial para avaliação do desempenho do motor em condições de carga.

4.1 Testes em bancada

Os testes em bancada são de extrema importância para se validar o correto funcionamento dos *drivers* de potência do *hardware*, antes de aplicá-los aos atuadores presentes no motor. Com isto, problemas ou comportamentos inesperados relacionados aos atuadores podem ser corrigidos de forma segura em bancada.

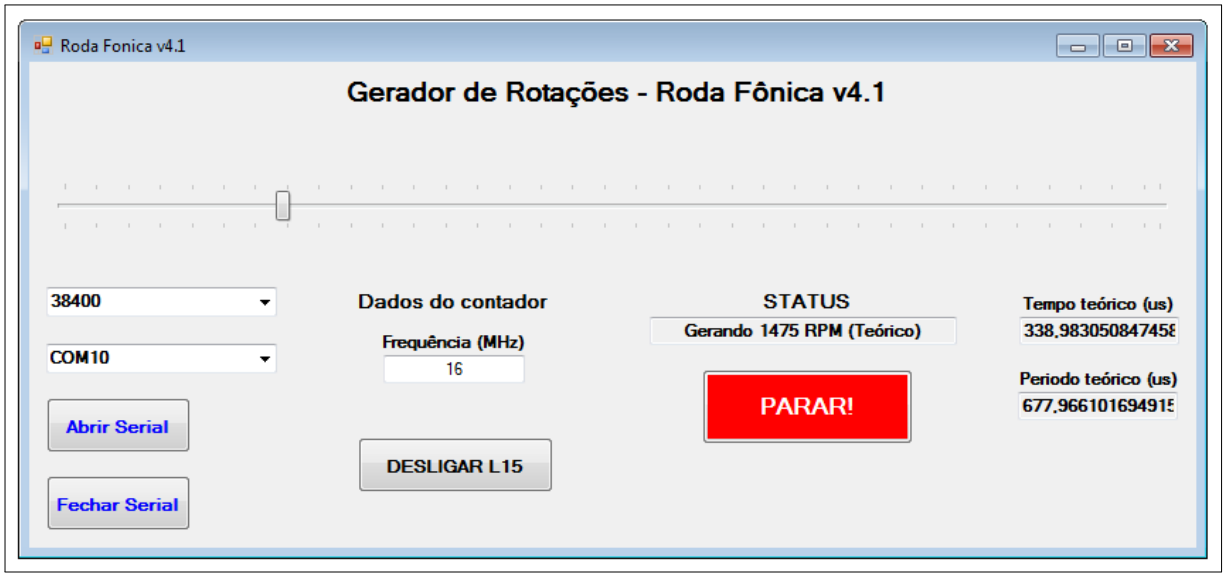
Figura 67 – Bancadas com kits de simulação



Fonte: SCARPINETTI e SOARES (2012)

4.1.1 Emulação do sinal de rotação

Figura 68 – Aplicação utilizada para a geração do sinal de rotação

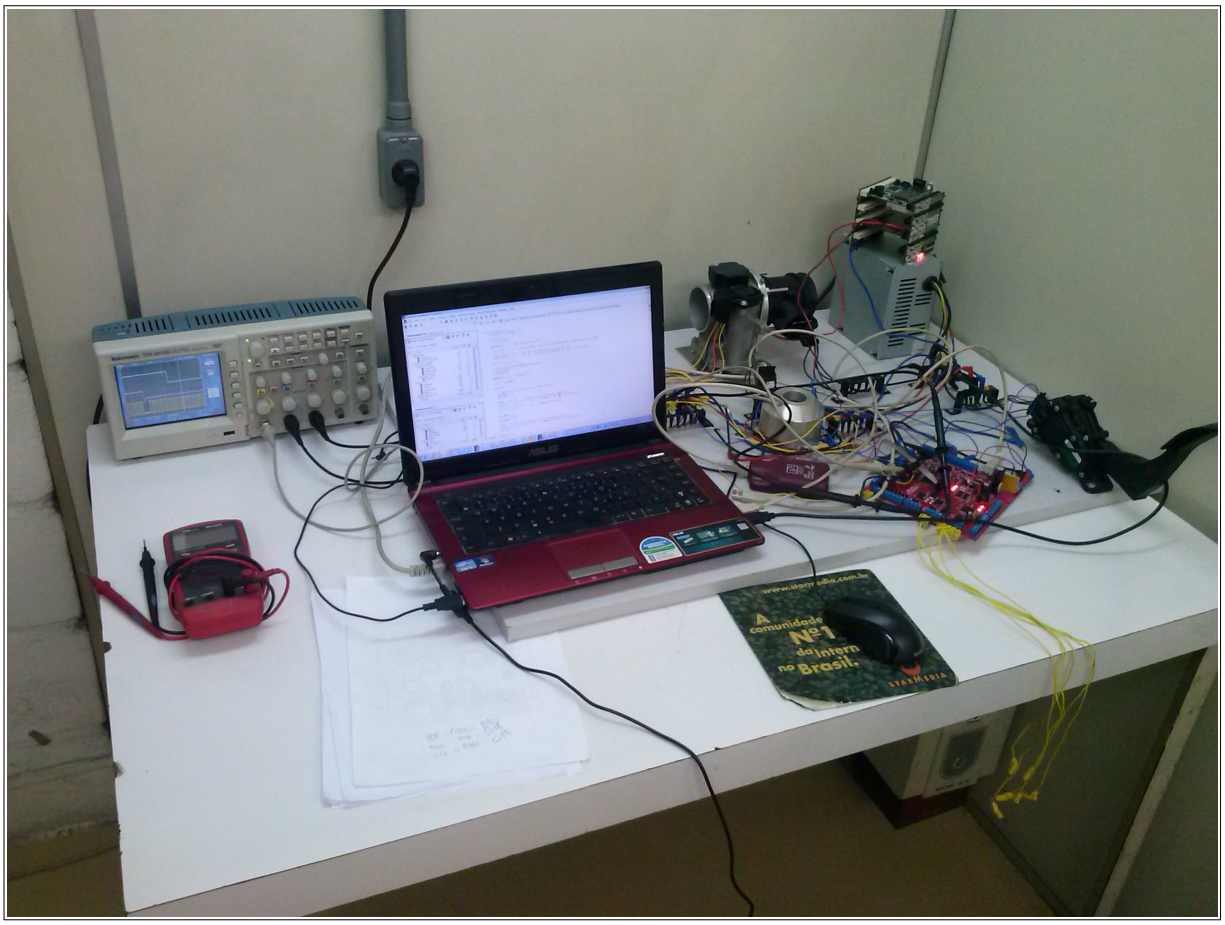


Fonte: o autor

Para validar o correto funcionamento do *firmware*, optou-se por emular o sinal de rotação, correspondente ao sinal de saída do CI MAX9924, responsável por digitalizar o sinal senoidal proveniente do sensor de relutância variável. O sinal de rotação é fundamental para se validar o correto cálculo da rotação, bem como o correto acionamento dos sinais de injeção e ignição.

A geração do sinal é feita com auxílio de um kit Freescale que contém o microcontrolador de 8 bits S08LH64. Através de uma comunicação serial RS-232, a aplicação, rodando em ambiente *Windows*, envia ao microcontrolador um valor que corresponde ao período de duração de um dente do sinal, período este que é calculado de acordo com a rotação que se deseja emular. Com o dado recebido da serial, o microcontrolador gera, com auxílio da interrupção de um timer, o sinal de rotação desejado, alterando no instante correto o estado lógico de um pino de saída. O *software* mencionado foi desenvolvido na ferramenta *Visual Basic* da *Microsoft* em 2011 para desenvolvimentos e testes em laboratório.

Figura 69 – Laboratório para Testes (LSI-USP)



Fonte: o autor

Além de ser útil no desenvolvimento do *firmware*, esta etapa é extremamente importante na correção de erros durante os testes no motor. Caso se verifique uma falha no veículo, mapeia-se a faixa de rotação onde ocorreu o problema e então se reproduz a falha em laboratório, corrigindo-a de forma segura.

4.1.2 Simulação com roda fônica em bancada

A finalidade desta etapa consiste na validação do correto funcionamento do CI MAX9924. Pode-se conferir o sinal digitalizado na saída do CI em diferentes rotações, por meio da indução de uma onda senoidal a partir de um sensor de relutância variável acoplado à roda fônica do kit.

4.1.3 Simulação com kits de injeção e ignição em bancada

Estes kits tem como objetivo validar o correto funcionamento do *driver* de injeção e ignição (CI 33810) presente na placa, além de testar a resistência do *hardware* à ruídos gerados pelas correntes elevadas dos bicos injetores e das velas de ignição. Os kits também

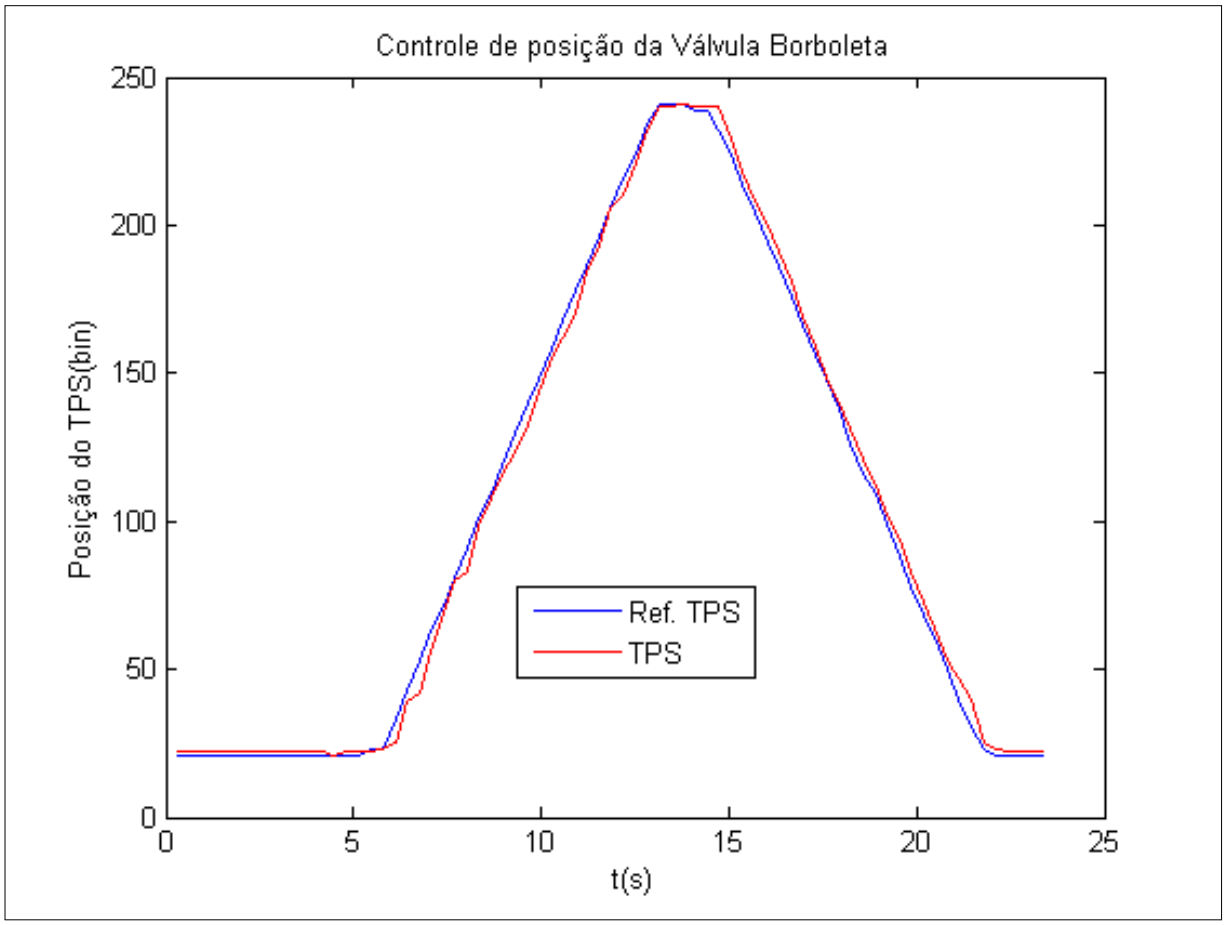
permitem avaliar a capacidade da placa em drenar estas correntes elevadas oriundas das bobinas de ignição e das válvulas injetoras.

4.1.4 Simulação com kit de válvula borboleta em bancada

Este kit é fundamental para se validar o correto funcionamento do controle de posição da válvula borboleta, implementado no uC de Gerenciamento. Assim, quando surge a necessidade de se alterar o algoritmo de controle ou ajustar os ganhos deste controlador, é necessário realizar testes em bancada com este kit antes de aplicar a ECU ao motor. Além disto, é possível verificar, analogamente aos kits de injeção e ignição, a capacidade em se drenar alta corrente e a resistência do *hardware* à ruídos. Outra função deste kit é a de validar o correto funcionamento da ponte H (CI 33186), *driver* utilizado para alimentar o servo motor da válvula borboleta.

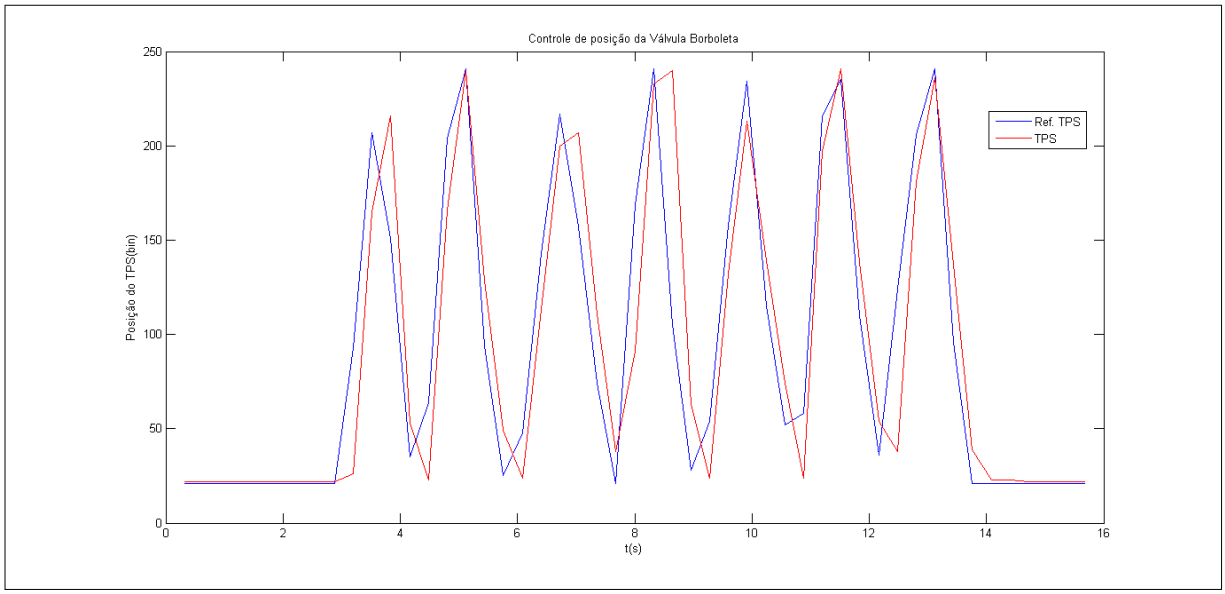
Os gráficos abaixo mostram alguns resultados de atuação do controle de posição da válvula borboleta. Dois ensaios foram feitos para avaliar o desempenho do controle. Os dados foram adquiridos com auxílio do *software* de monitoramento desenvolvido neste projeto. Os gráficos, por sua vez, foram construídos com auxílio da ferramenta MATLAB.

Figura 70 – Gráfico do controle de posição da válvula borboleta com variações suaves na referência



Fonte: o autor

Figura 71 – Gráfico do controle de posição da válvula borboleta com variações bruscas na referência



Fonte: o autor

Pelas figuras anteriores é possível notar que o controle de posição da borboleta funciona corretamente, com erros pequenos e sem oscilações, o que confere estabilidade ao controle. Além disto, a resposta do controle de posição é considerada adequada à aplicação envolvida. Os ensaios anteriores foram realizados em laboratório.

4.2 Testes no veículo

Os testes de integração do *firmware* foram todos executados no veículo cedido pela FATEC Santo André ao Projeto Otto II, separados em etapas distintas:

- **Verificação do correto cálculo da rotação**: A primeira etapa consiste em verificar se a ECU desenvolvida calcula de forma correta a rotação do motor, sendo que neste caso é utilizado a ECU original do veículo para comparações;
- **Atuação nos bicos injetores**: Esta etapa valida o correto acionamento dos sinais de injeção, que devem ser acionados em instantes específicos. Inicialmente é realizado uma comparação, com auxílio de um osciloscópio, entre os sinais gerados pela ECU original do veículo e a ECU deste projeto, e em sequência o motor é acionado com os sinais gerados pela unidade deste projeto;
- **Atuação na válvula borboleta**: Com o algoritmo de controle da borboleta validado em laboratório, o mesmo é aplicado à borboleta, porém inicialmente com o motor desligado. Após calibrado os ganhos, o motor é acionado com o controle

da válvula borboleta da ECU deste projeto, respondendo a pequenas variações no pedal do acelerador;

- **Atuação na ignição**: Analogamente à segunda etapa, valida-se o correto acionamento dos sinais de ignição. Inicialmente é realizado uma comparação com os sinais de ignição da ECU original, e em seguida a ignição é acionada pela ECU deste projeto;

- **Leitura dos sensores**: Nesta etapa, valida-se a correta leitura dos sensores que serão utilizados neste projeto, exceto o sensor de rotação, que já foi validado na primeira etapa. Nesta etapa é validado também o correto funcionamento do *software* de monitoramento, no que diz respeito à aquisição em tempo real dos dados;

- **Acionamento de Relé e atuação completa**: Nesta etapa a meta corresponde à validação do acionamento do relé da bomba de combustível. Neste momento é realizado também um teste, acionando-se simultaneamente todos os atuadores validados nas etapas anteriores. Portanto, após esta etapa, espera-se que o motor seja completamente controlado pela ECU v2.0 deste projeto, não dependendo, portanto, de nenhuma atuação da ECU original;

- **Inserção do cálculo da massa de combustível**: Esta etapa consiste em melhorar o desempenho do motor, calculando-se a massa de combustível que deve ser injetada em uma dada volta. Utiliza-se, neste caso, as informações dos sensores MAP e temperatura do ar;

- **Inserção do controle de rotação**: Esta etapa consiste em inserir e validar o correto funcionamento do controle de rotação. Espera-se ao final desta etapa que seja possível se controlar de forma estável a rotação do motor de acordo com a leitura do pedal de aceleração. O motor deve responder até 6000 RPM, ainda sem aplicação de carga (motor em vazio), com erros de regime de até 200 RPM;

- **Ajuste do tempo de injeção para acelerações bruscas**: Nesta etapa, são realizados diversos testes de acelerações bruscas, de forma que o motor responda de forma satisfatória com o controle de rotação. Também é validado o corte de combustível em caso de desacelerações;

- **Ajuste para partida a frio**: Testes são realizados para verificar o correto funcionamento do motor em baixas temperaturas;

- **Aplicação de carga ao motor**: Nesta etapa espera-se validar o correto funcionamento do motor em condições de carga, utilizando o veículo em conjunto com o dinamômetro inercial. Todos os ajustes e correções são realizados de forma a melhorar o desempenho do motor nestas condições. Como resultado, espera-se que o

motor consiga alcançar 6000 RPM em terceira marcha com a carga fixa fornecida pelo dinamômetro, desenvolvendo uma potência máxima de no mínimo 50% do valor nominal (116 cv) do motor, o que corresponde ao valor de 58 cv.

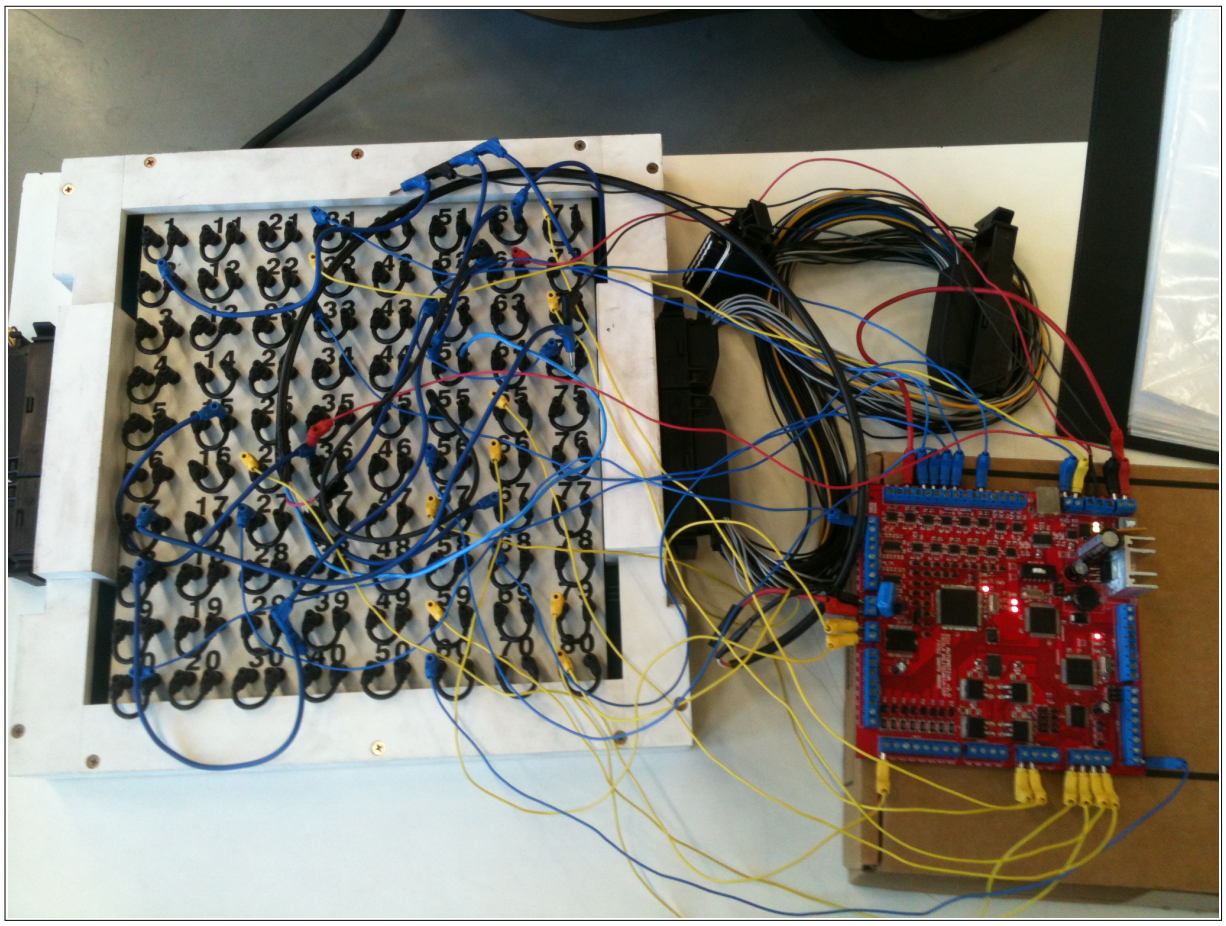
- **Comunicação com a rede CAN**: Nesta etapa é implementado a comunicação com a rede CAN presente no veículo, enviando a rotação calculada para o painel do motorista e lendo a velocidade instantânea do veículo. Espera-se, ao final desta etapa, que a ECU original possa ser completamente desligada do veículo, sem prejuízos para o funcionamento do motor;

Para obter acesso aos sinais de sensores e atuadores presentes no motor foi utilizado uma intersecção, construída na FATEC Santo André. Além de possibilitar o monitoramento destes sinais, esta intersecção permite também, através de *jumpers*, aplicar sinais de atuação ao motor, desligando, neste caso, sinais de atuação provenientes da ECU original do veículo. Deste modo, pode-se controlar parcialmente o motor, deixando somente alguns atuadores para serem controlados pela ECU original. Ao final, espera-se que todos os atuadores da ECU original possam ser desligados, sendo, neste momento, a ECU v2.0 responsável por todo o controle do motor.

Figura 72 – Intersecção dos sinais do Motor Volkswagen 2.0L

Fonte: o autor

Figura 73 – ECU v2.0 conectada à intersecção

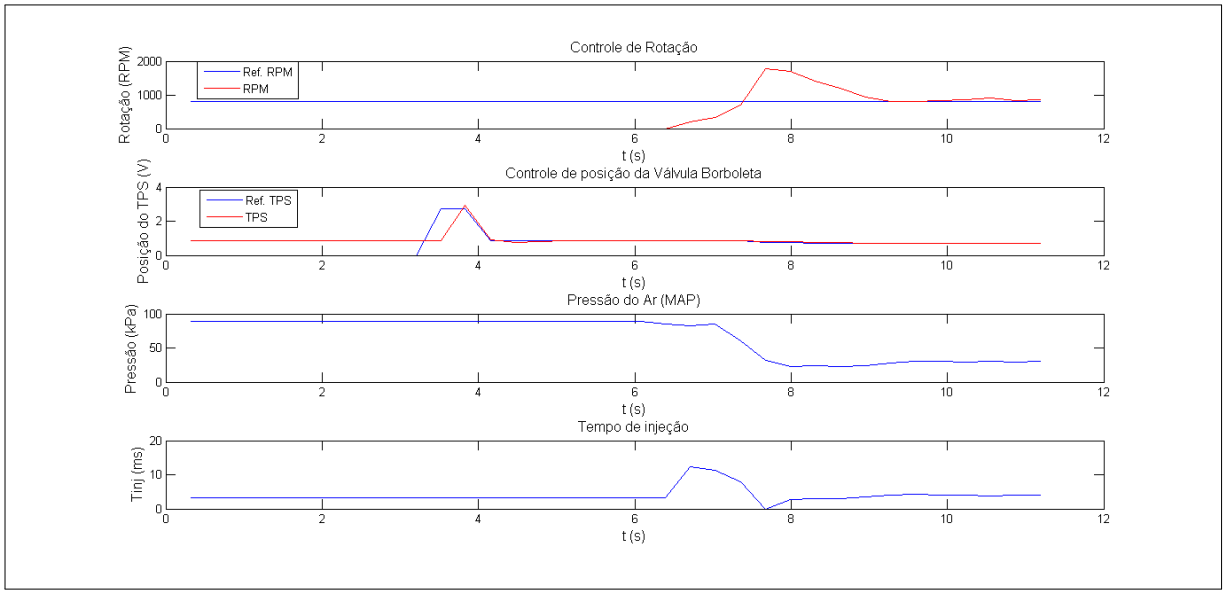


Fonte: o autor

Diversos testes foram efetuados no veículo para se validar o correto funcionamento da ECU v2.0 deste projeto. Os gráficos a seguir mostram como o controle de rotação opera em conjunto com o controle de posição da borboleta, com o tempo de injeção e com a pressão da massa de ar admitida. Os gráficos foram construídos com uso da ferramenta MATLAB e com dados obtidos em tempo real pela aplicação de monitoramento desenvolvida neste projeto.

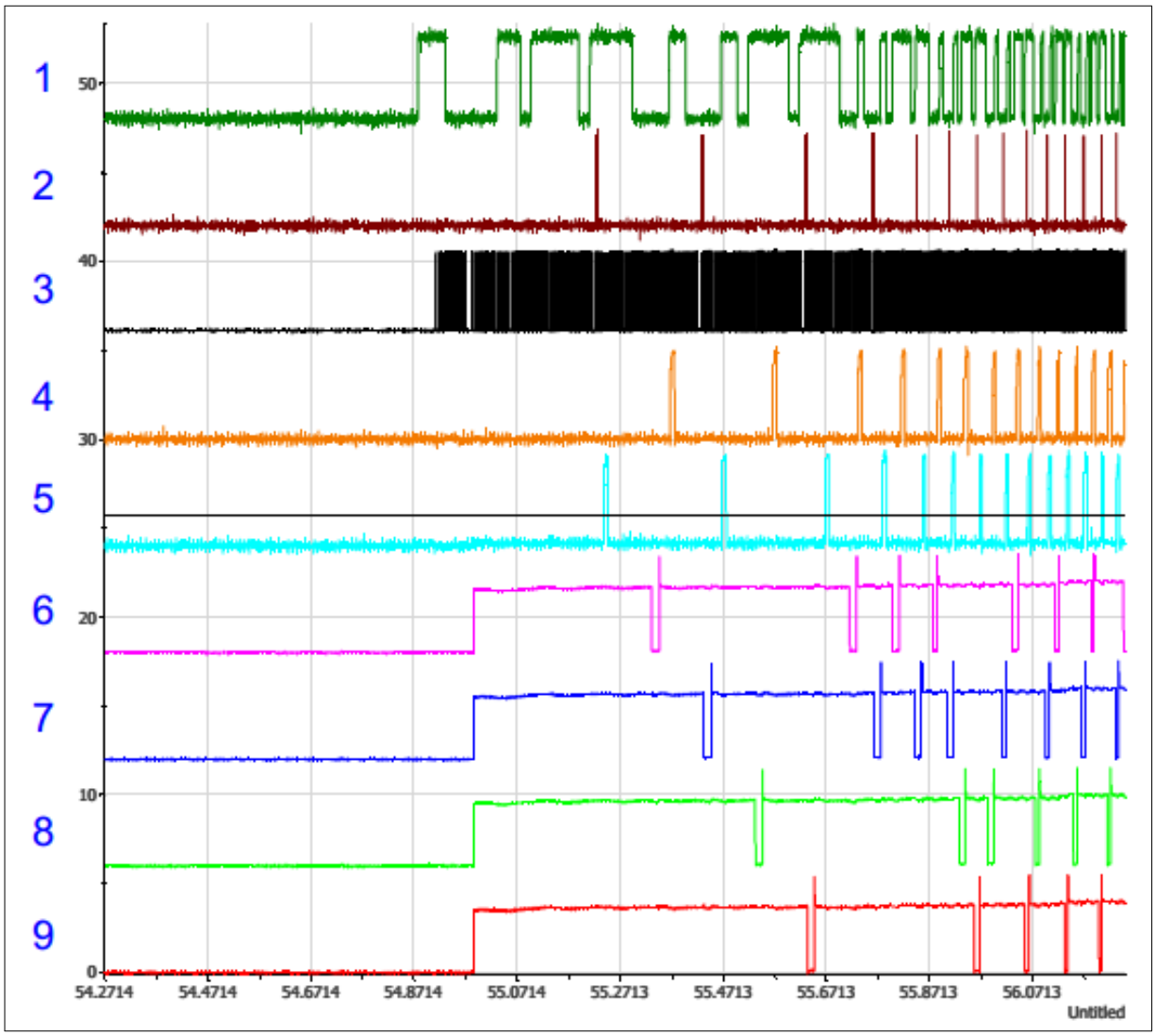
4.2.1 Regime de partida

Figura 74 – Monitoramento - Regime de partida



Fonte: o autor

Figura 75 – Sinais de acionamento - Regime de partida



Fonte: o autor

Com relação aos sinais da figura anterior, obtidos com auxílio da ferramenta LabView, uma legenda foi disponibilizada a seguir. É importante mencionar que a mesma legenda é válida para todas as futuras figuras que contêm sinais de acionamento.

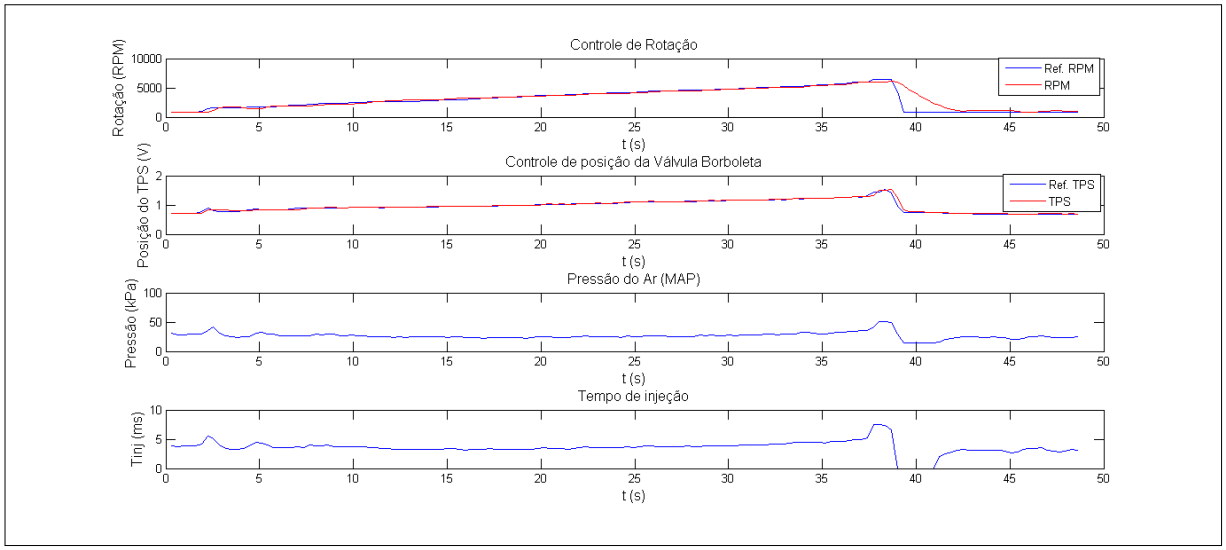
1. Sinal do sensor de fase
2. Sinal de detecção da falha da roda fônica
3. Sinal de rotação digitalizado
4. Comando de ignição do banco 2-3
5. Comando de ignição do banco 1-4
6. Comando de injeção do cilindro 2

7. Comando de injeção do cilindro 4
8. Comando de injeção do cilindro 3
9. Comando de injeção do cilindro 1

Pelas figuras anteriores é possível notar algumas características da partida do motor. Nota-se, por exemplo, que o acionamento de injeção se mantém sempre no modo sequencial. É possível verificar também que a partida não envolve nenhuma abertura adicional na válvula borboleta, esta que se mantém com uma abertura pequena e constante durante todo o regime, apenas fechando um pouco quando a rotação do motor supera a rotação de marcha lenta, de modo a seguir o controle de rotação. É importante mencionar que a primeira abertura da borboleta (que ocorreu por volta de 4s) se deve apenas a um teste inicial de funcionamento, não sendo, portanto, uma abertura necessária para a partida do motor. Observa-se também um tempo de injeção maior no início da partida, necessário para que o motor vença o atrito e alcance a rotação de marcha lenta.

4.2.2 Aceleração suave com o motor em vazio

Figura 76 – Monitoramento - Aceleração suave com o motor em vazio

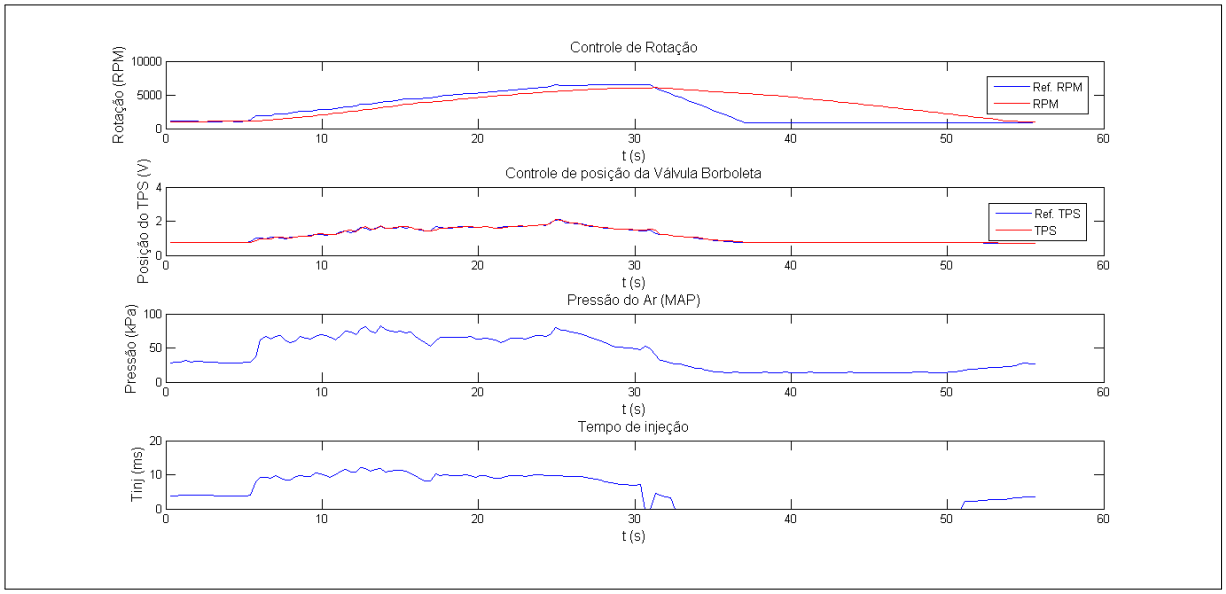


Fonte: o autor

O teste da figura anterior foi realizado com intuito de analisar a estabilidade do controle de rotação quando o motor não opera com carga. Observa-se que o controle de rotação funciona adequadamente, dado que a rotação segue fielmente (erros pequenos) a sua referência. Nota-se ainda a operação correta do controle de posição da borboleta. Observa-se, por fim, que o tempo de injeção não sofre muita alteração, decorrente da aceleração suave.

4.2.3 Aceleração suave com o motor submetido a carga

Figura 77 – Monitoramento - Aceleração suave com carga

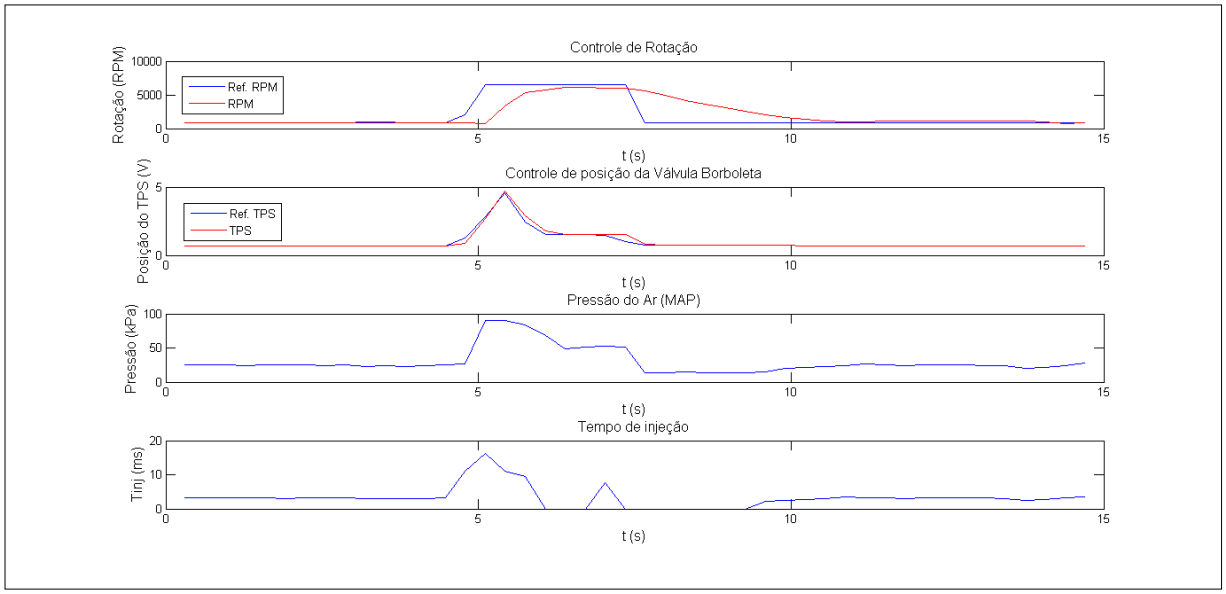


Fonte: o autor

É possível notar que, em condições de carga, aumenta-se ligeiramente o erro do controle de rotação, além de aumentar também o tempo de injeção. Este fato, porém, já é esperado e não compromete o correto funcionamento do motor. Observa-se ainda que o tempo de injeção acompanha a massa de ar em ambos os testes (com e sem carga), o que valida o correto funcionamento do controle estequiométrico da mistura a/c, demonstrando, ao mesmo tempo, que a ideia deste controle operar em malha aberta (através da estimativa da massa de ar) funciona corretamente, mesmo sem o uso da sonda lambda.

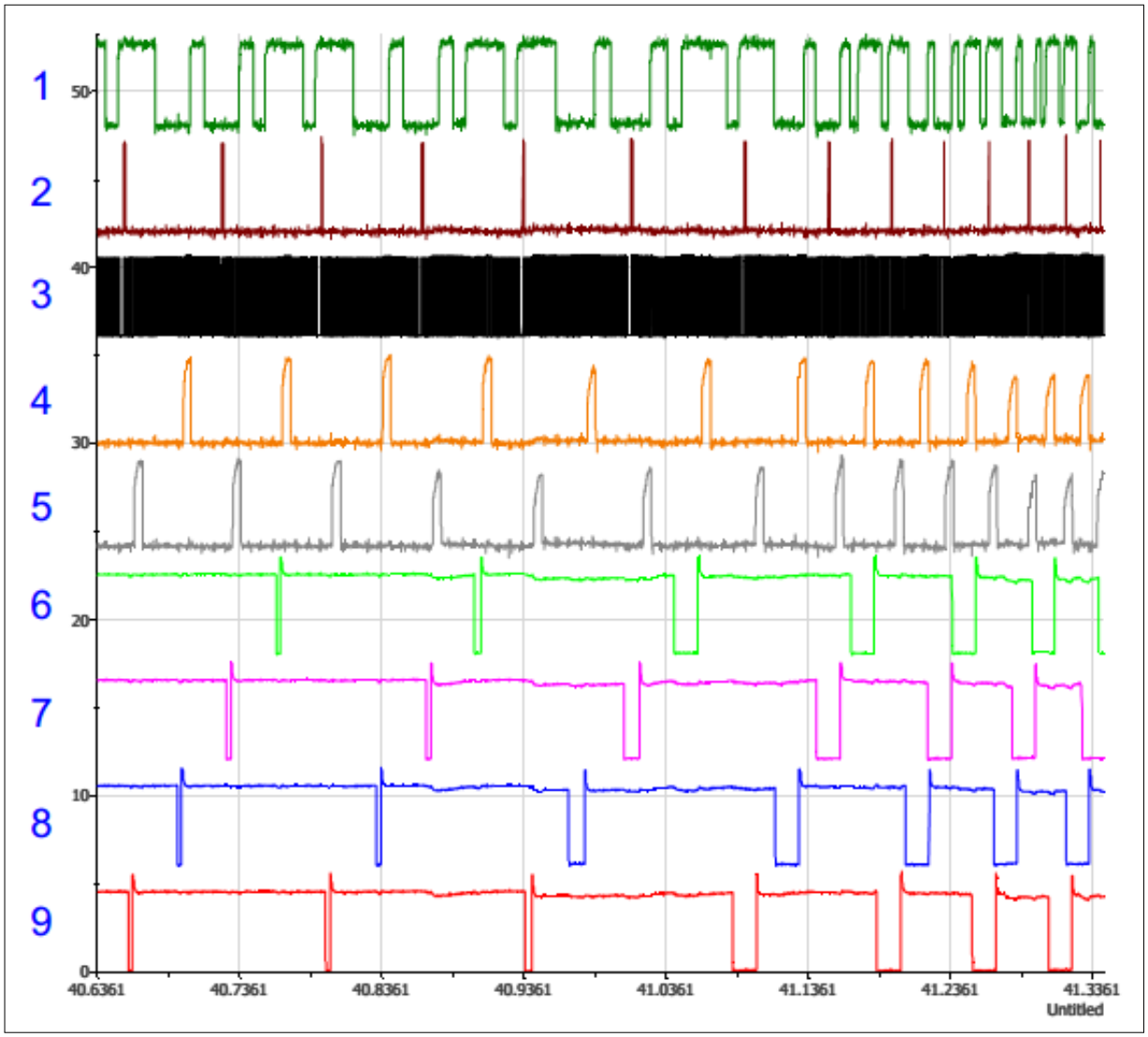
4.2.4 Aceleração brusca com o motor em vazio

Figura 78 – Monitoramento - Aceleração brusca com o motor em vazio



Fonte: o autor

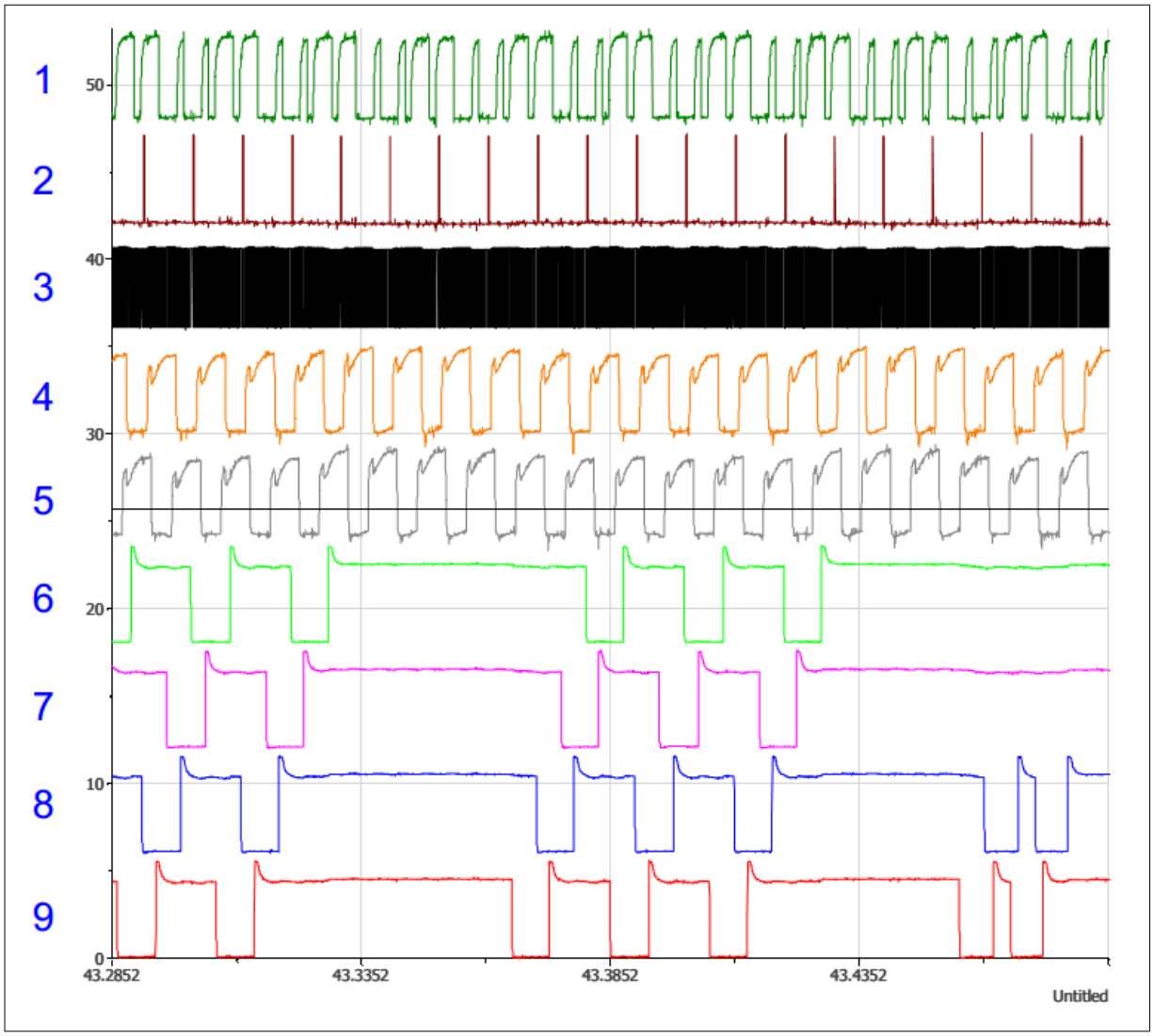
Figura 79 – Sinais de acionamento - Instante da aceleração brusca



Fonte: o autor

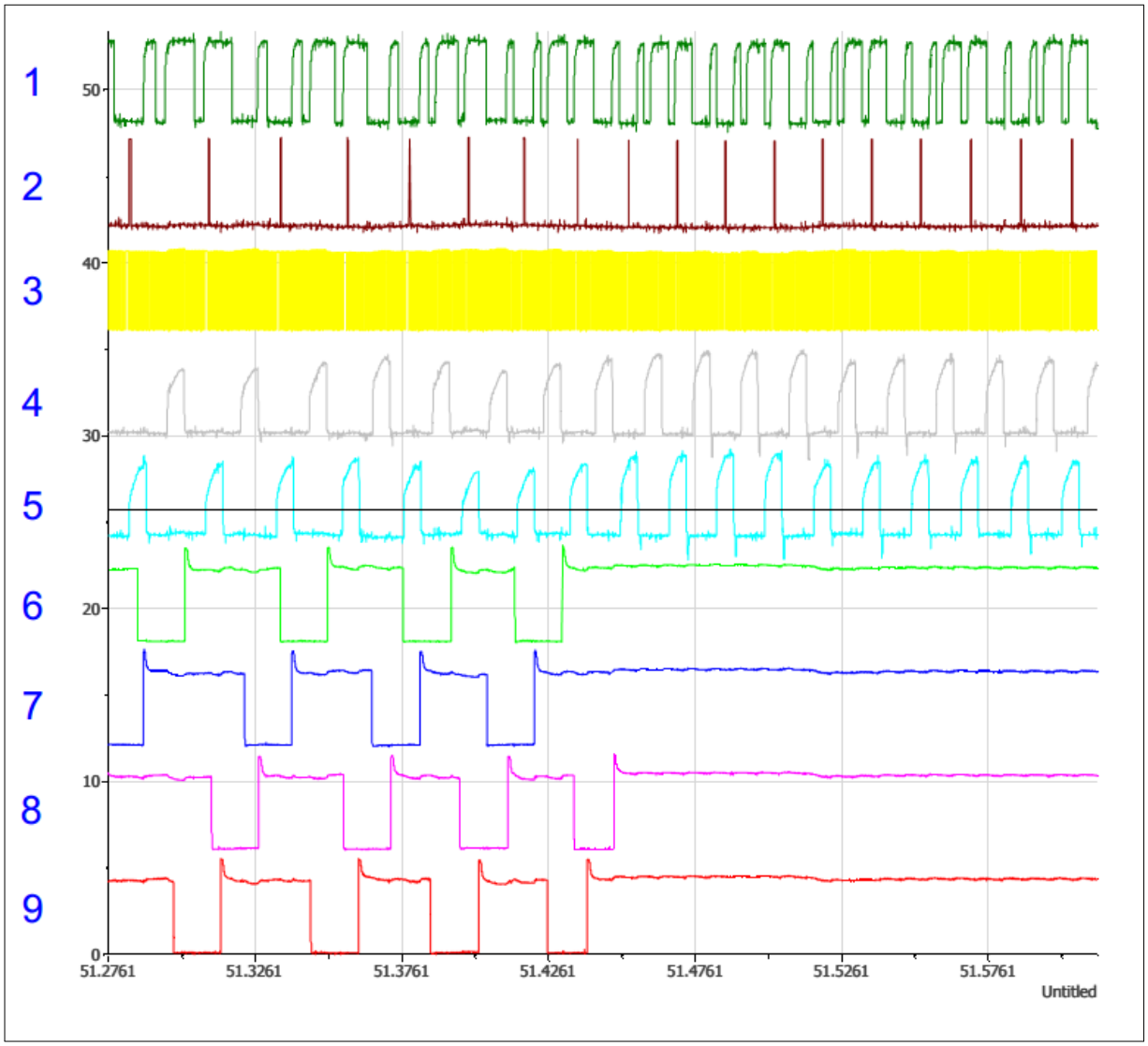
Os resultados com aceleração brusca (figuras 78 e 79) mostram que o tempo de injeção aumenta significativamente, decorrente do maior enriquecimento da mistura e da elevada massa de ar admitida, esta que, por sua vez, resultou da maior abertura da válvula borboleta. Nota-se que a borboleta chega a abrir completamente (90°) neste ensaio, quando o pedal é completamente pressionado e a rotação passa dos 3000 RPM. É possível observar também o corte de combustível em duas ocasiões, sendo a primeira (por volta de 6s a 7s) devido a rotação ter alcançado o limite imposto de 6050 RPM, e a segunda (de 7,5s a 9s) devido à desaceleração.

Figura 80 – Sinais de acionamento - Corte de injeção a 6050 RPM



Fonte: o autor

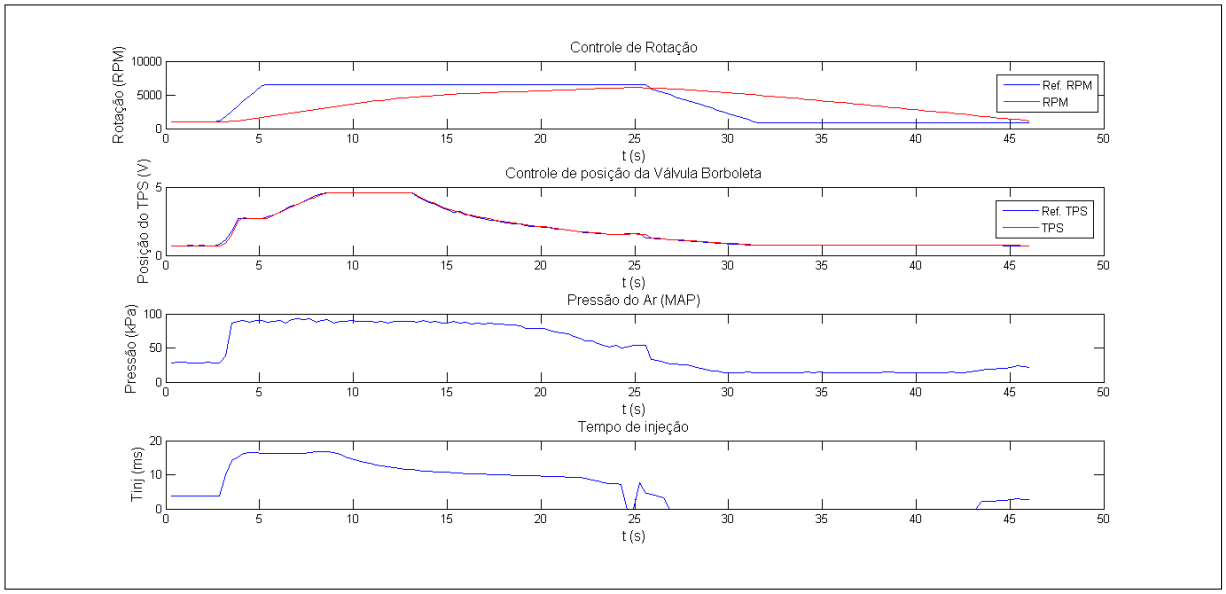
Figura 81 – Sinais de acionamento - Corte de injeção na desaceleração



Fonte: o autor

4.2.5 Aceleração brusca com o motor submetido a carga - plena potência

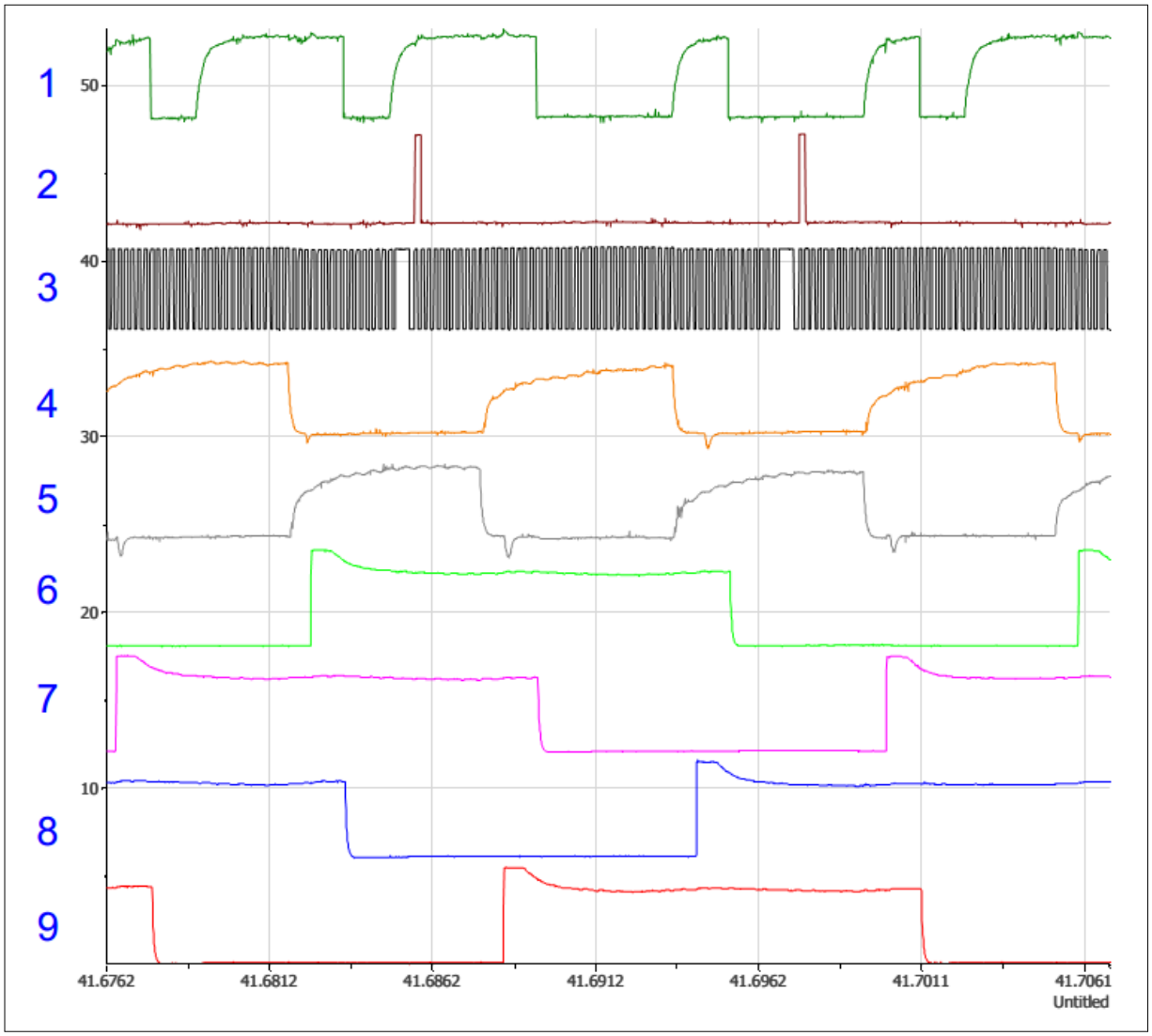
Figura 82 – Monitoramento - Aceleração brusca com carga



Fonte: o autor

A figura anterior mostra os resultados do teste de plena potência, no qual o pedal foi totalmente pressionado durante o teste com carga. Nota-se, sobretudo, que foi necessário um tempo maior para a rotação atingir aproximadamente 6000 RPM, quando comparado ao mesmo teste com o motor em vazio, fato decorrente da carga imposta ao motor. É possível notar a limitação na abertura da válvula borboleta, por volta de 4s, limitação esta necessária para evitar um "‘afogamento"’ do motor devido ao excesso de ar e à baixa rotação neste instante.

Figura 83 – Sinais de acionamento - Aceleração brusca com carga



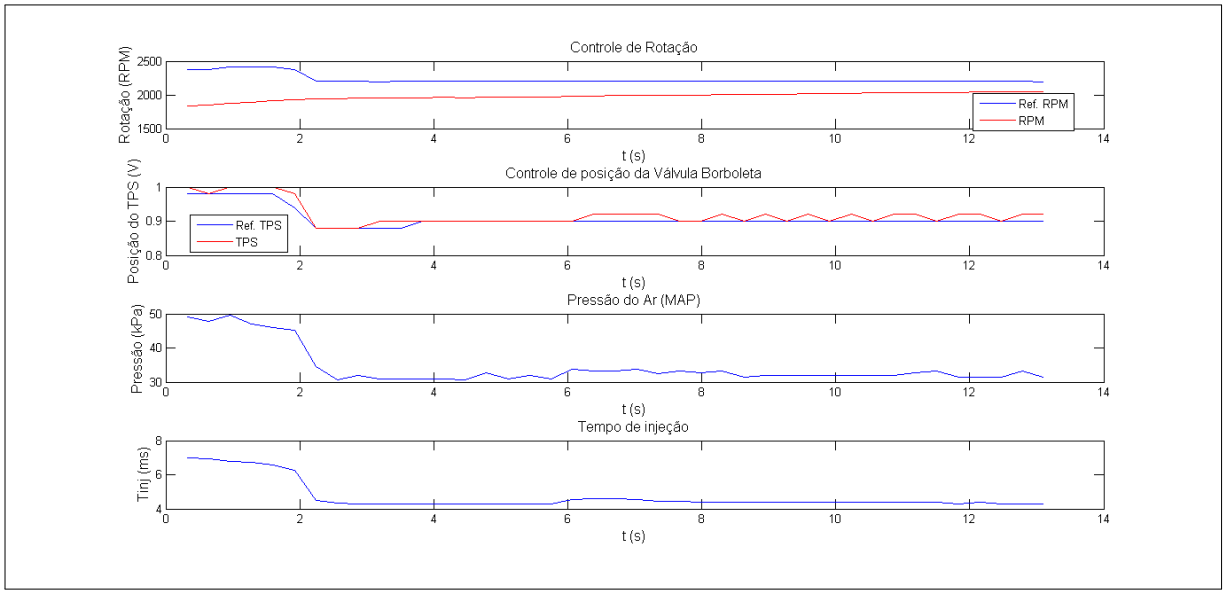
Fonte: o autor

Além disto, nota-se que o tempo de injeção permanece elevado na maior parte do tempo (por volta de 16 ms), de modo a acompanhar a elevada massa de ar admitida pelo motor durante este teste, fato evidenciado pelo elevado valor de pressão fornecido pelo sensor MAP.

Por volta de 4500 RPM, a válvula borboleta começa a fechar, o que dá início a uma diminuição da pressão de ar, e, consequentemente, faz diminuir o tempo de injeção. Este procedimento é realizado através de ajustes do ganho proporcional do controle de rotação e tem como objetivo diminuir a potência do motor quando a rotação já está elevada, por motivos de segurança.

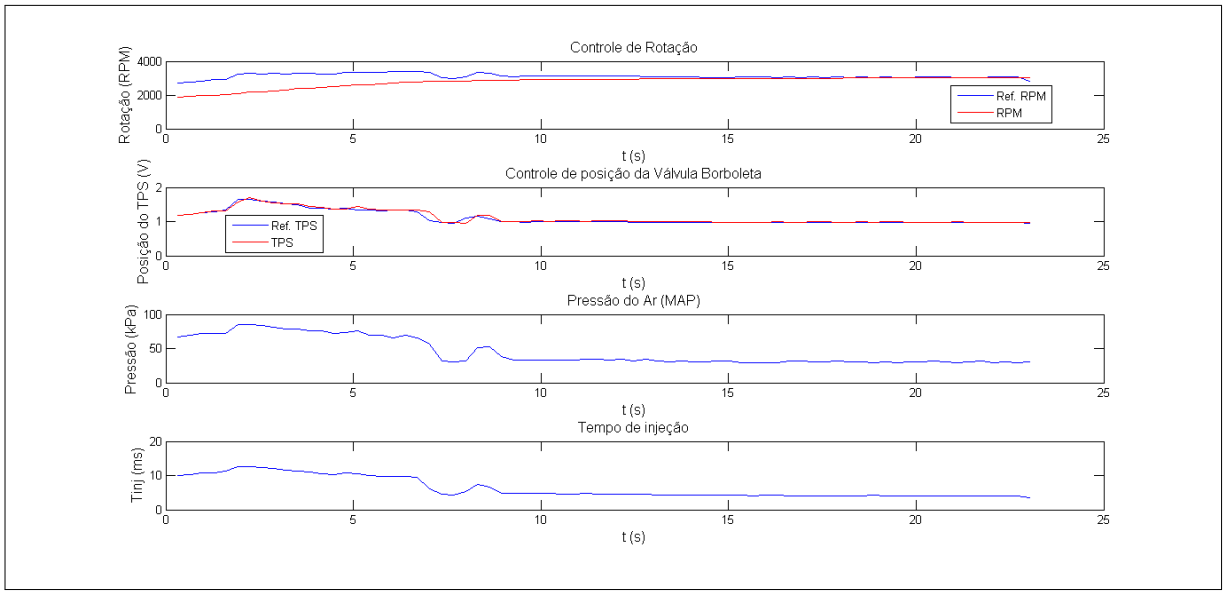
4.2.6 Rotação constante com o motor submetido a carga

Figura 84 – Monitoramento - 2000 RPM com o motor em carga



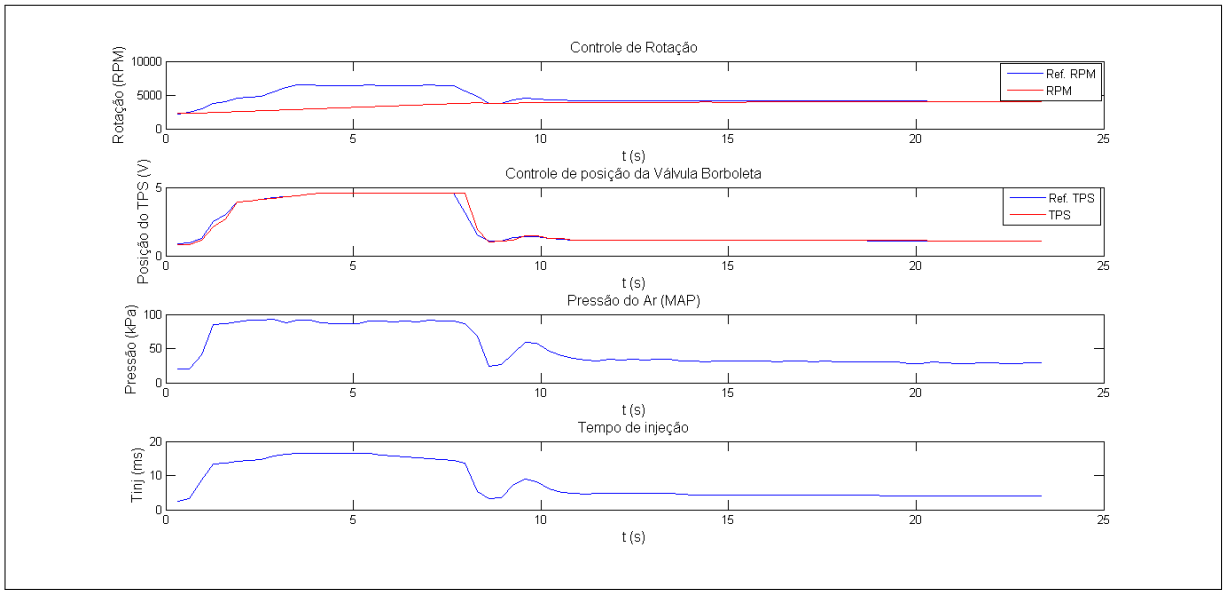
Fonte: o autor

Figura 85 – Monitoramento - 3000 RPM com o motor em carga



Fonte: o autor

Figura 86 – Monitoramento - 4000 RPM com o motor em carga

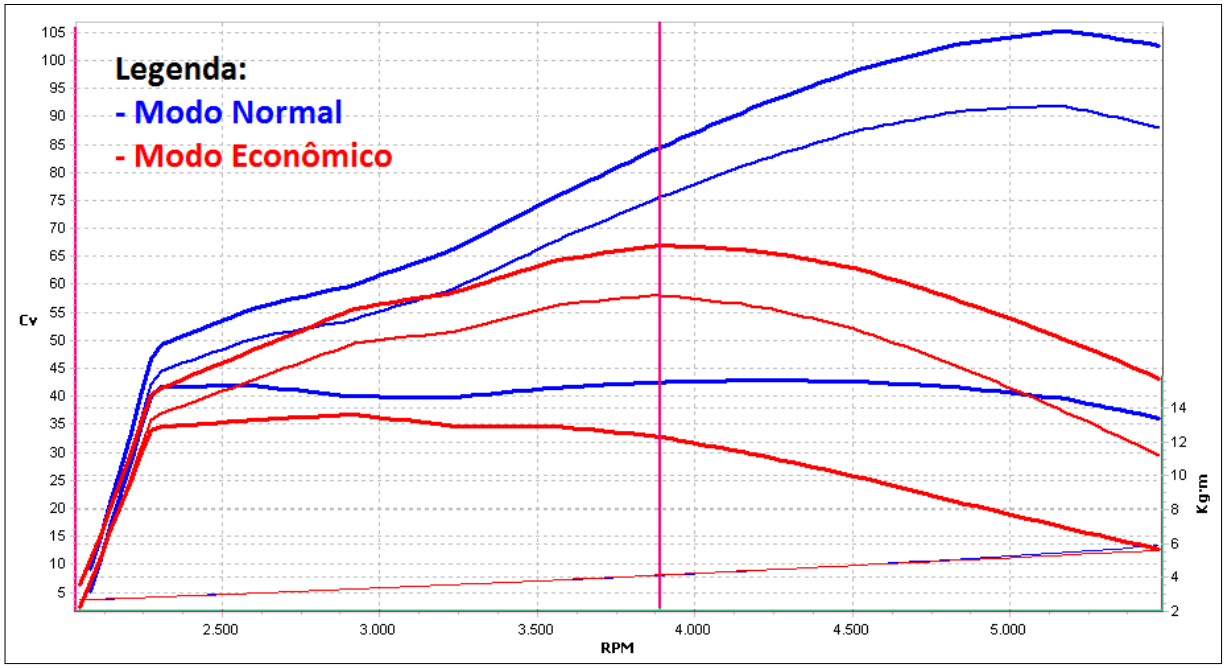


Fonte: o autor

Testes foram realizados para manter o motor em 2000, 3000 e 4000 RPM, de modo a analisar a estabilidade do controle de rotação quando o motor se encontra submetido a carga. Pelas figuras acima, desconsiderando a aceleração inicial, é possível notar que de modo geral se conseguiu manter a rotação desejada, mesmo com um pequeno erro de regime decorrente da carga imposta, fato que, por sua vez, não prejudica a aplicação. Analogamente ao teste de aceleração suave, o tempo de injeção se manteve pequeno e constante, da mesma forma que a válvula borboleta se manteve com uma abertura pequena.

4.2.7 Curva de Potência: Modo Econômico x Modo Normal

Figura 87 – Curva de potência e torque dos modos normal e econômico



Fonte: o autor

Para comparar os dois modos de operação da ECU v2.0, testes de potência[1] foram realizados. Nota-se, pela figura anterior, que o modo econômico fornece potência e torque menores do que o modo normal. Este fato já era esperado, dado que no modo econômico a VB trabalha com aberturas menores, o que diminui a quantidade de combustível injetado e, consequentemente, reduz a potência e o torque do motor. Os pontos máximos de cada curva foram destacados na tabela a seguir.

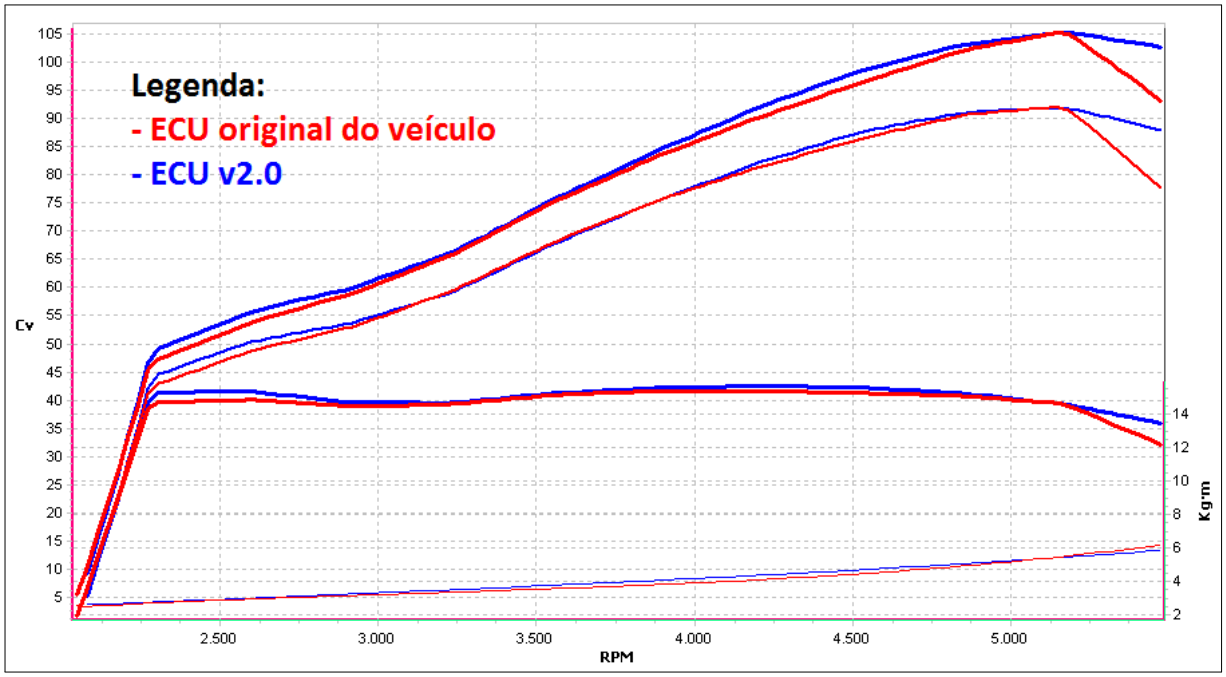
Tabela 9 – Potência e torque máximos (econômico x normal)

Modo de operação	**Potência máxima**	**Torque máximo**
Normal	105.2 cv a 5182 RPM	15.7 mkgf a 4213 RPM
Econômico	66.9 cv a 3890 RPM	13.6 mkgf a 2920 RPM

[1] No teste de potência, realizado com auxílio do dinamômetro, o pedal de aceleração é totalmente pressionado até que a rotação do motor alcance aproximadamente 5500 RPM, com o veículo engatado em terceira marcha.

4.2.8 Curva de Potência: ECU original do veículo x ECU v2.0

Figura 88 – Curva de potência e torque da ECU original e da ECU v2.0



Fonte: o autor

Testes de potência foram realizados com a ECU original do veículo e com a ECU v2.0 deste projeto, a fim de comparar o desempenho de ambos os sistemas. Nota-se que, de modo geral, a ECU deste projeto apresenta desempenho similar à ECU original do veículo, dado que as curvas de ambos são bastante similares. Os pontos máximos de cada curva foram apresentados na tabela a seguir.

Tabela 10 – Potência e torque máximos (original x v2.0)

Unidade Eletrônica	**Potência máxima**	**Torque máximo**
ECU original	105.2 cv a 5150 RPM	15.4 mkgf a 3890 RPM
ECU v2.0	105.2 cv a 5182 RPM	15.7 mkgf a 4213 RPM

5 Conclusões

O projeto Otto II cumpriu com a sua meta inicial de desenvolver uma unidade de gerenciamento eletrônico para um motor Volkswagen 2.0L, operado inclusive em condições de carga. O projeto demonstrou que é possível se controlar um motor à combustão interna por meio de um controle de rotação, responsável também pelo fornecimento de torque ao eixo do motor em condições de carga. Pelos resultados obtidos é possível notar que, com a implementação do controle de rotação, a ECU v2.0 deste projeto responde semelhantemente a ECU original do veículo, incluindo a característica de filtragem do pedal de aceleração, de modo a suavizar acelerações bruscas quando o veículo se encontra em movimento. Além disto, é importante ressaltar que a unidade eletrônica desenvolvida neste projeto atua no motor de forma totalmente independente e sem qualquer auxílio da ECU original[1].

Consequentemente, o projeto também atingiu a meta de se desenvolver um *hardware* robusto e eficiente para processar de forma rápida e segura todas as informações envolvidas no controle. Além disto, por ser um *hardware* bastante flexível, espera-se que o mesmo possibilite desenvolvimentos futuros na área de eletrônica automotiva da Escola Politécnica da USP, com intuito de implementar inovações e melhorias no setor automotivo.

Apesar de não ser original[2], o trabalho se mostrou bastante desafiador, pois se trata de uma aplicação real de grande porte, que envolve diversos conceitos como eletrônica analógica, eletrônica de potência, processamento de sinais, teoria de controle, programação, projeto de *hardware*, protocolos de comunicação e conceitos de engenharia automotiva. Assim, o projeto se constituiu em uma grande oportunidade de aplicação de parte dos conceitos adquiridos no curso de graduação de sistemas eletrônicos da Escola Politécnica da USP, além de estimular a busca por novos conhecimentos.

Espera-se com o resultados obtidos neste projeto que se desperte um interesse maior de alunos e professores da Escola Politécnica da USP em participar e trabalhar, conjuntamente ao departamento PSI da Escola Politécnica da USP e à Fatec Santo André, em novos projetos no setor automotivo, envolvendo a área de eletrônica automotiva. Alunos que buscam desafios em projetos práticos, com o objetivo de aprimorar seus conhecimentos em eletrônica, tais como o desenvolvimento de *hardware*, *firmware* e *software*, certamente terão oportunidade de adquirí-los em um projeto deste tipo, envolvendo eletrônica automotiva.

[1] Relembrando que a ECU original foi completamente desligada do motor.
[2] Dado que atualmente ECUs já existem no mercado.

5.1 Sugestões para projetos futuros

- Desenvolvimento de modelos matemáticos para aplicação da metodologia de projeto por modelos, visando o desenvolvimento de futuros projetos em diferentes motores;
- Desenvolvimento de controle de detonação em malha fechada, o que envolve estratégia para se retirar informação do sensor de detonação[3] e atuar, principalmente, no adiantamento de ignição;
- Desenvolvimento de controle estequiométrico da mistura a/c em malha fechada, envolvendo o uso da sonda lambda[4] e aprimoramentos na estratégia do *firmware*, a fim de diminuir o consumo de combustível e reduzir a emissão de poluentes;
- Desenvolvimento de uma ECU para motores bicombustíveis (flex), com uso de algoritmos e sensores adequados, a fim de aprimorar a identificação da mistura do combustível, além de ajustar de modo adaptativo os mapas de injeção e ignição;
- Aprimoramentos no *software* de monitoramento e diagnose;
- Aprimoramentos no *hardware* e no *firmware*, a fim de inserir um controle de torque e aumentar ainda mais a segurança da ECU v2.0, o que envolve estratégias como operação em modo de segurança;
- Desenvolvimento de uma ECU para controle de transmissão, interagindo diretamente com a ECU v2.0 desenvolvida neste projeto.

[3] Apesar de estar presente no motor Volkswagen 2.0L estudado, este sensor, conhecido também por sensor *knock*, não foi utilizado no presente projeto.

[4] Analogamente ao *knock*, este sensor também não foi utilizado neste projeto.

Referências

ALBALADEJO, F. S. *Desenvolvimento de uma unidade de gerenciamento eletrônico para motores de combustão interna do ciclo Otto*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, 2013.

ANG, K. H.; CHONG, G.; LI, Y. Pid control system analysis, design, and technology. *Control Systems Technology, IEEE Transactions on*, v. 13, n. 4, p. 559–576, 2005.

AUTOMOTIVE BUSINESS. *Magneti Marelli prepara inovações para o sistema flex*. 2013. Notícia publicada em 08 de abril de 2013. Disponível em: http://www.automotivebusiness.com.br/noticia/16720/magneti-marelli-prepara-inovacoes-para-o-sistema-flex-. Acesso em: agosto/2013.

BBC NEWS. *Car hackers use laptop to control standard car*. 2013. Notícia publicada em 25 de julho de 2013. Disponível em: http://www.bbc.co.uk/news/technology-23443215. Acesso em: setembro/2013.

BOSCH. *Bosch Auto Parts: Gasoline*. 2013. Disponível em: http://de.bosch-automotive.com/parts/parts_and_accessories/motor_and_sytems/benzin/benzin_1. Acesso em: maio/2013.

BOSCH. *MAP Sensor Tecnical Specifications*. [S.l.], 2013. Disponível em: http://www.bosch.com.au/car_parts/en/downloads/Map_Sensor_Technical_Specification.pdf. Acesso em: abril/2013.

BOSCH. *Relay online catalog*. 2013. Disponível em: http://rb-aa.bosch.com/boaarocs/index.jsp;jsessionid=687C781D5322090B862DA6CE81608728.sundoro2?ccat_id=33&prod_id=111&lang=en. Acesso em: maio/2013.

BOSCH. *Sensors*. [S.l.], 2013. Disponível em: http://jenniskens.livedsl.nl/Technical/Tips/Files/Bosch%20Sensor%20information.pdf. Acesso em: maio/2013.

BOSCH. *What is CAN?* 2013. Disponível em: http://www.bosch-semiconductors.de/en/ubk_semiconductors/safe/ip_modules/what_is_can/what_is_can.html. Acesso em: junho/2013.

BOSCH, R. *Manual de tecnologia automotiva*. [S.l.]: Edgard Blücher, 2005. ISBN 8521203780.

CLUBPOLO FORUMS. *NA Ignition maps*. 2013. Disponível em: http://www.clubpolo.co.uk/forum/index.php?showtopic=303127. Acesso em: junho/2013.

CONTINENTAL. *Continental inaugura Centro Tecnológico de Powertrain no interior de São Paulo*. 2012. Notícia publicada em 22 de agosto de 2012. Disponível em: http://www.continental-corporation.com/www/pressportal_br_pt/themes/press_releases/3_automotive_group/powertrain/pr_2012_08_22_centro_tecnologico_inauguracao_br_pt.html. Acesso em: agosto/2013.

CONTINENTAL. *Sistema de Gerenciamento de Motor*. 2013. Disponível em: http://www.conti-online.com/www/automotive_br_pt/themes/passenger_cars/powertrain/pi_ecu_pt.html. Acesso em: julho/2013.

CRUZ, J. J. da. *PTC2413 Controle I - Notas de Aula*. 2009. Apostila do curso PTC2413 oferecido na graduação da Escola Politécnica da USP.

DELPHI. *ETC air Control Valve (ETC-ACV) application manual*. [S.l.], 2003.

DEUR, J. et al. An electronic throttle control strategy including compensation of friction and limp-home effects. *Industry Applications, IEEE Transactions on*, v. 40, n. 3, p. 821–834, 2004.

DIAS, B. M. de A. *Plataforma didática de injeção eletrônica para controle de motores a combustão interna*. Monografia (Trabalho de Conclusão de Curso) — Faculdade de Tecnologia de Santo André, 2011.

ESCOLA POLITéCNICA DA USP; FATEC SANTO ANDRé. *Mapeamento Polo*. 2013.

FREESCALE. *Using XGATE to Implement a Simple Buffered SCI - Application Note*. 2005. Disponível em: http://www.freescale.com/files/microcontrollers/doc/app_note/AN3144.pdf. Acesso em: julho/2013.

FREESCALE. *Datasheet MC33810*. [S.l.], 2008.

FREESCALE. *Datasheet MC33186*. [S.l.], 2011.

FREESCALE. *Datasheet da família MC9S12XE*. [S.l.], 2012.

FREESCALE. *S12XE: 16-Bit Automotive Microcontroller - Overview*. 2013. Disponível em: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=S12XE&nodeId=0162468636K100. Acesso em: março/2013.

FUTURE TECHNOLOGY DEVICES INTERNATIONAL LIMITED. *Datasheet FT232R*. [S.l.], 2010.

HEYWOOD, J. *Internal combustion engine fundamentals*. [S.l.]: McGraw-Hill, 1988. ISBN 9780070286375.

IGNIFLEX - Ignição e Componentes Automotivos. 2013. Disponível em: http://www.igniflex.com.br/midia. Acesso em: abril/2013.

KALINSKY, D.; KALINSKY, R. *Introduction to Serial Peripheral Interface*. 2002. Disponível em: http://www.embedded.com/electronics-blogs/beginner-s-corner/4023908/Introduction-to-Serial-Peripheral-Interface. Acesso em: abril/2013.

KIENCKE, U.; NIELSEN, L. *Automotive Control Systems*. [S.l.]: Springer Berlin Heidelberg, 2005. ISBN 9783540231394.

LITTELFUSE. *Resettable PTCs*. 2013. Disponível em: http://www.littelfuse.com/products/resettable-ptcs.aspx. Acesso em: março/2013.

MANAVELLA, J. H. *Controle integrado do motor*. [S.l.: s.n.], 1996.

MAXIM INTEGRATED PRODUCTS. *Datasheet MAX9924*. [S.l.], 2012.

MICROSOFT. *Visual C#*. 2013. Disponível em: http://msdn.microsoft.com/pt-br/library/kx37x362(v=vs.90).aspx. Acesso em: junho/2013.

MTE-THOMSON. *Conhecimentos básicos sobre Injeção Eletrônica*. 2013. Disponível em: http://www.mte-thomson.com.br/site/faq/conhecimentos-basicos-sobre-injecao-eletronica/. Acesso em: maiol/2013.

OGATA, K. *Engenharia de controle moderno*. Prentice Hall, 2003. ISBN 9788587918239. Disponível em: http://books.google.com.br/books?id=fXmIPQAACAAJ. Acesso em: abril/2013.

PORTAL VRUM. *Ficha Técnica: Polo Sed./ Sed. COMFORT. 2.0 Mi 8V 4p*. 2013. Disponível em: http://www.vrum.com.br/Autos?tp=VRUM&name=db:FichaTecnica&fabricante=Volkswagen&modelobase=POLO&anomodelo=2004&codigo=005186-1. Acesso em: maio/2013.

ROCHA, G. *Estequiometria do Motor de Combustão Interna*. 2009. Disponível em: http://www.infomotor.com.br/site/2009/03/esteqiometria-do-motor-de-combustao-interna. Acesso em: maio/2013.

SCARPINETTI, V. S.; SOARES, A. M. F. *Unidade eletrônica de controle de um motor a combustão: Projeto Otto*. Monografia (Trabalho de Conclusão de Curso) — Escola Politécnica da Universidade de São Paulo, 2012.

SERWAY, R.; JEWETT, J. *Princípios de Física: Eletromagnetismo*. [S.l.]: Pioneira Thomson Learning, 2004. ISBN 9788522104147.

SHEDEED, M. et al. Functional design and verification of automotive embedded software: An integrated system verification flow. In: *Electronics, Communications and Photonics Conference (SIECPC), 2013 Saudi International*. [S.l.: s.n.], 2013. p. 1–5.

SILVA, J. M. G. da; BAZANELLA, A. S. *Ajuste de Controladores PID*. 2000. Apostila de curso da Universidade Federal do Rio Grande do Sul. Disponível em: http://www.ece.ufrgs.br/~jmgomes/pid/Apostila/apostila/apostila.html. Acesso em: agosto/2013.

TEXAS INSTRUMENTS. *Datasheet SN65HVD1040*. [S.l.], 2011.

WONG, P.-K.; VONG, C.-M.; IP, W.-F. Modelling of petrol engine power using incremental least-square support vector machines for ecu calibration. In: *Optoelectronics and Image Processing (ICOIP), 2010 International Conference on*. [S.l.: s.n.], 2010. v. 2, p. 12–15.

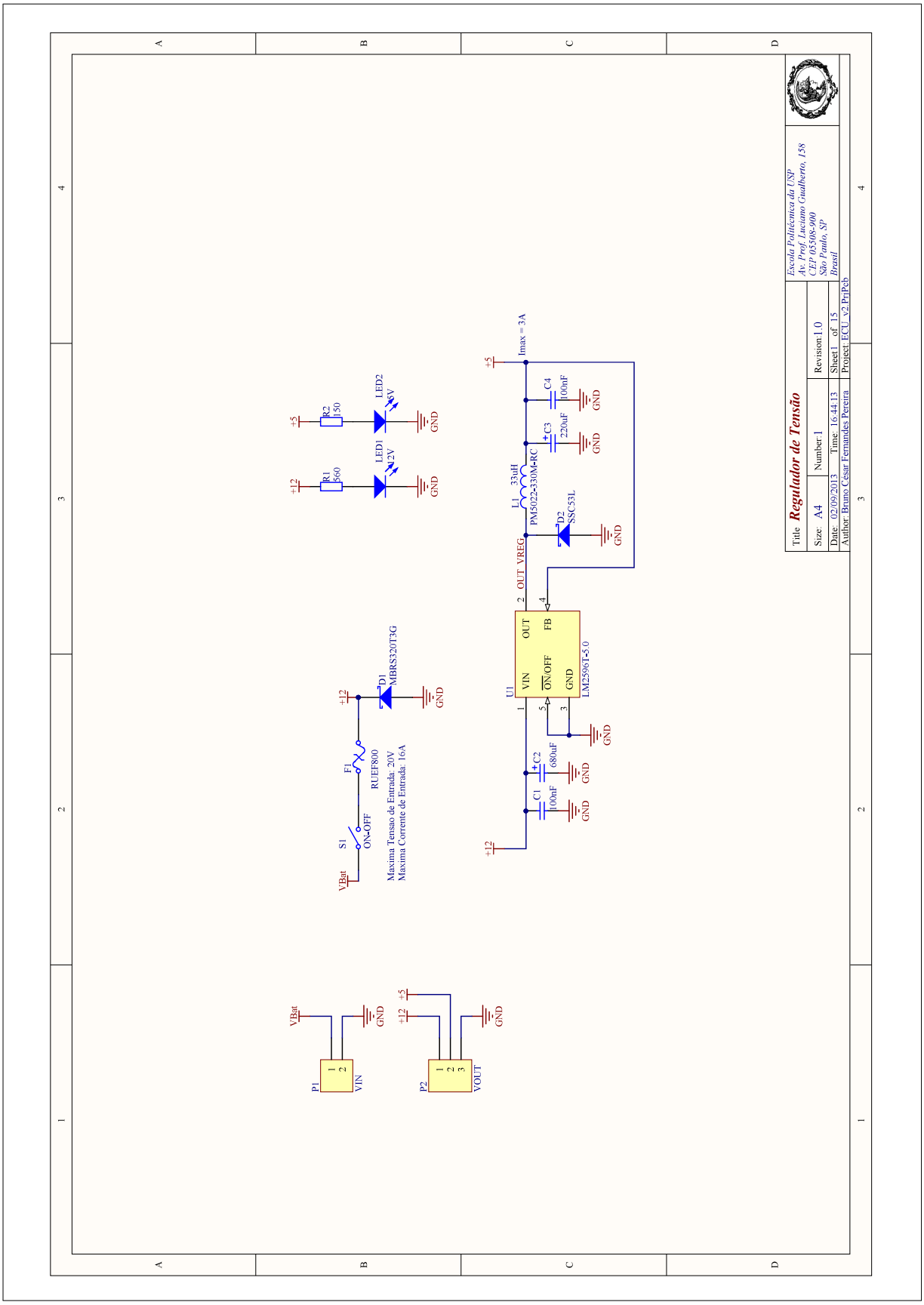
WYLEN, G. V.; BORGNAKKE, C.; SONNTAG, R. *Fundamentos da Termodinâmica*. [S.l.]: Edgard Blucher, 2009. ISBN 9788521204909.

YANSWERZ BLOGSPOT. *Four Stroke Engine / Spark Ignition Engine / Otto Cycle*. 2009. Disponível em: http://yanswerz.blogspot.com.br/2009/12/four-stroke-engine-spark-ignition.html. Acesso em: abril/2013.

Apêndices

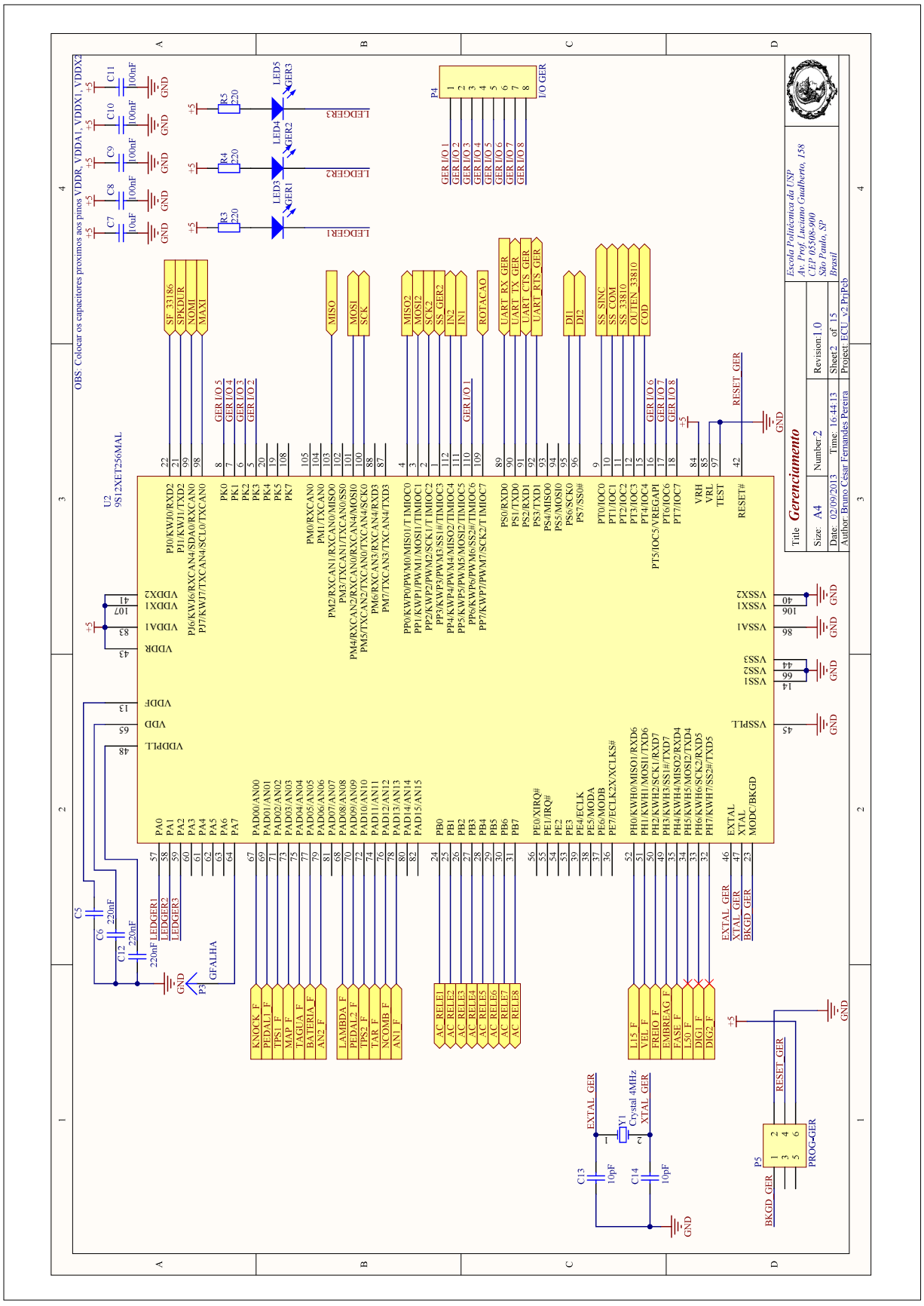
APÊNDICE A – *Hardware*

Figura 89 – Regulador de Tensão



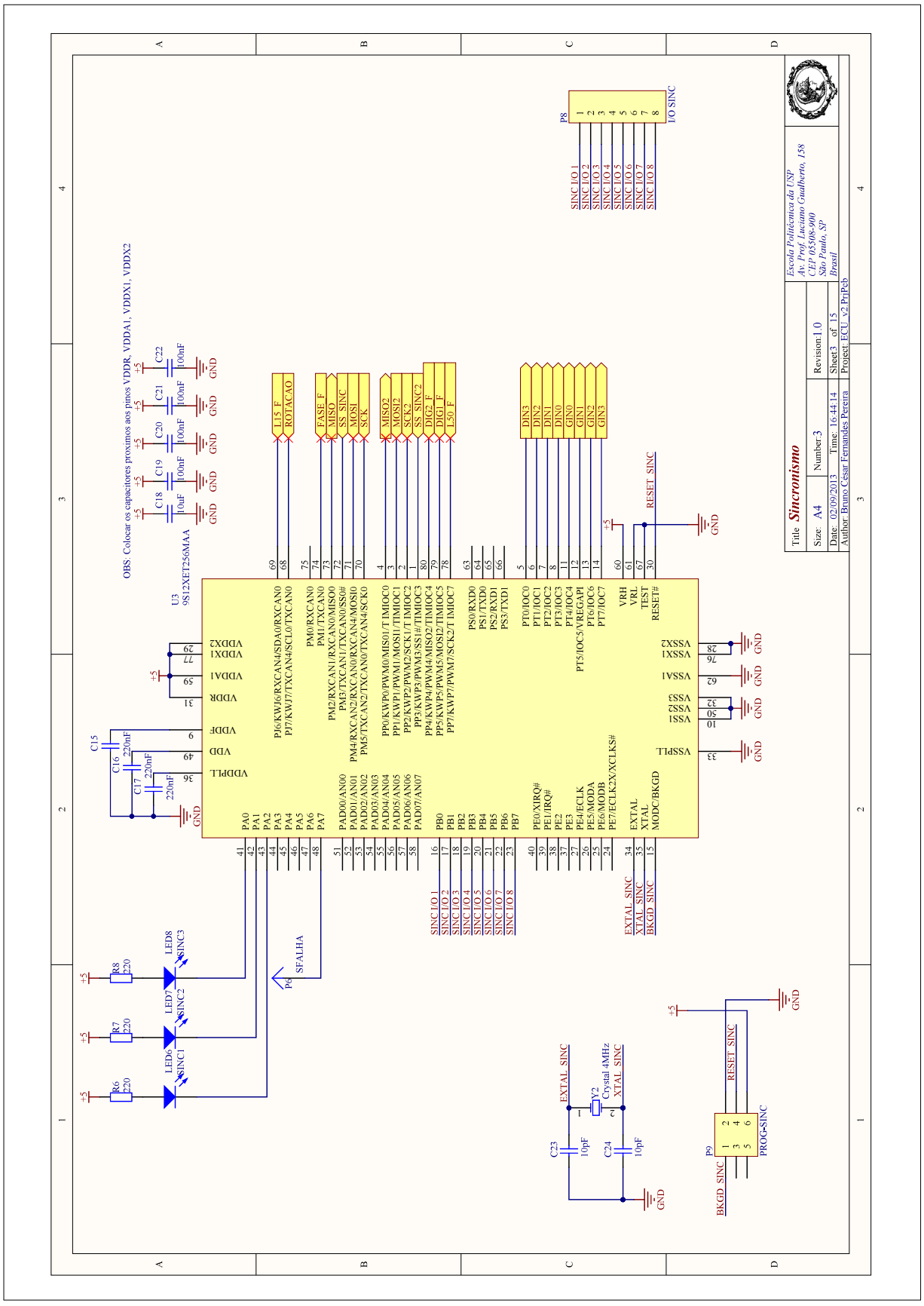
Fonte: o autor

Figura 90 – Microcontrolador de Gerenciamento



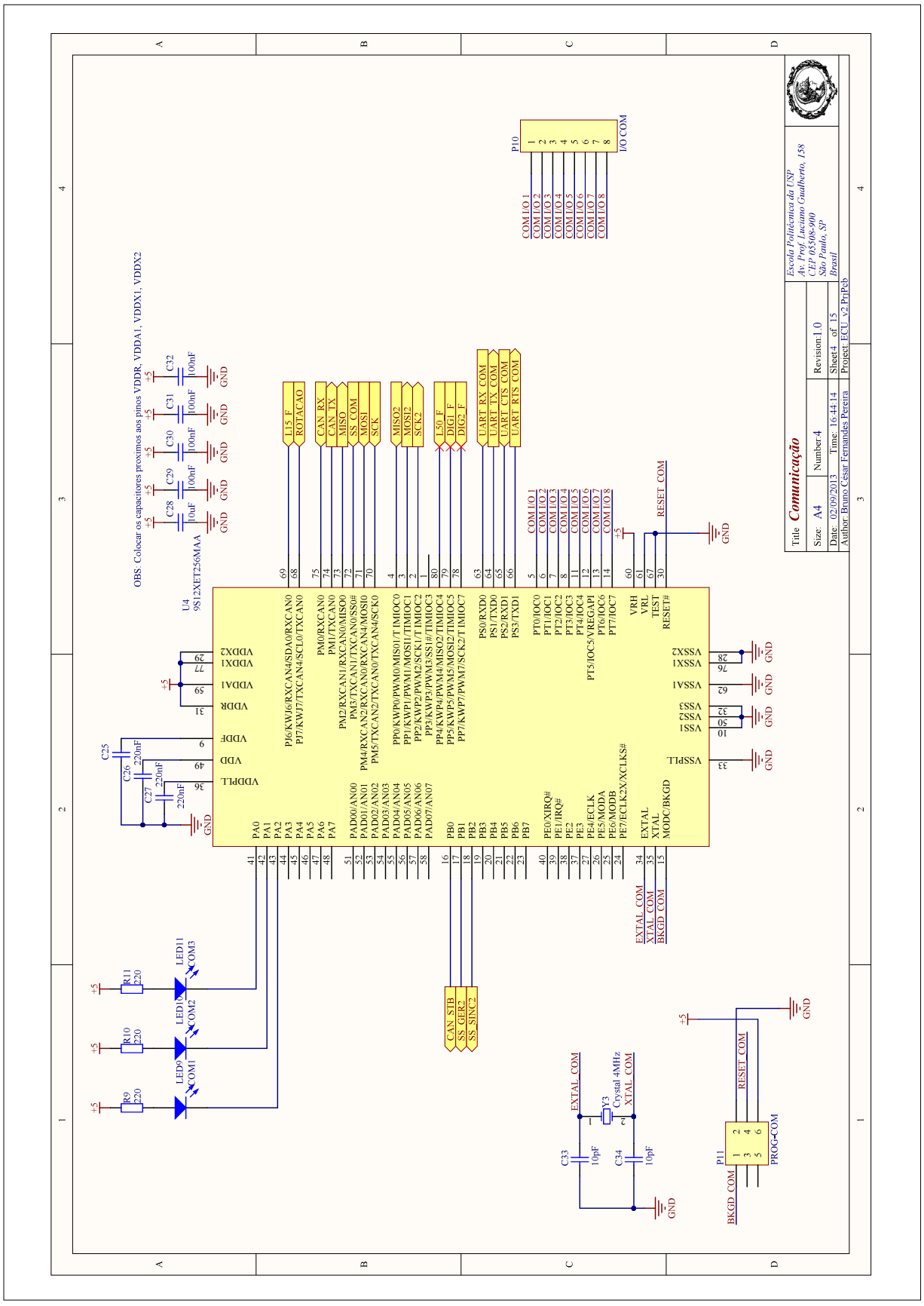
Fonte: o autor

Figura 91 – Microcontrolador de Sincronismo



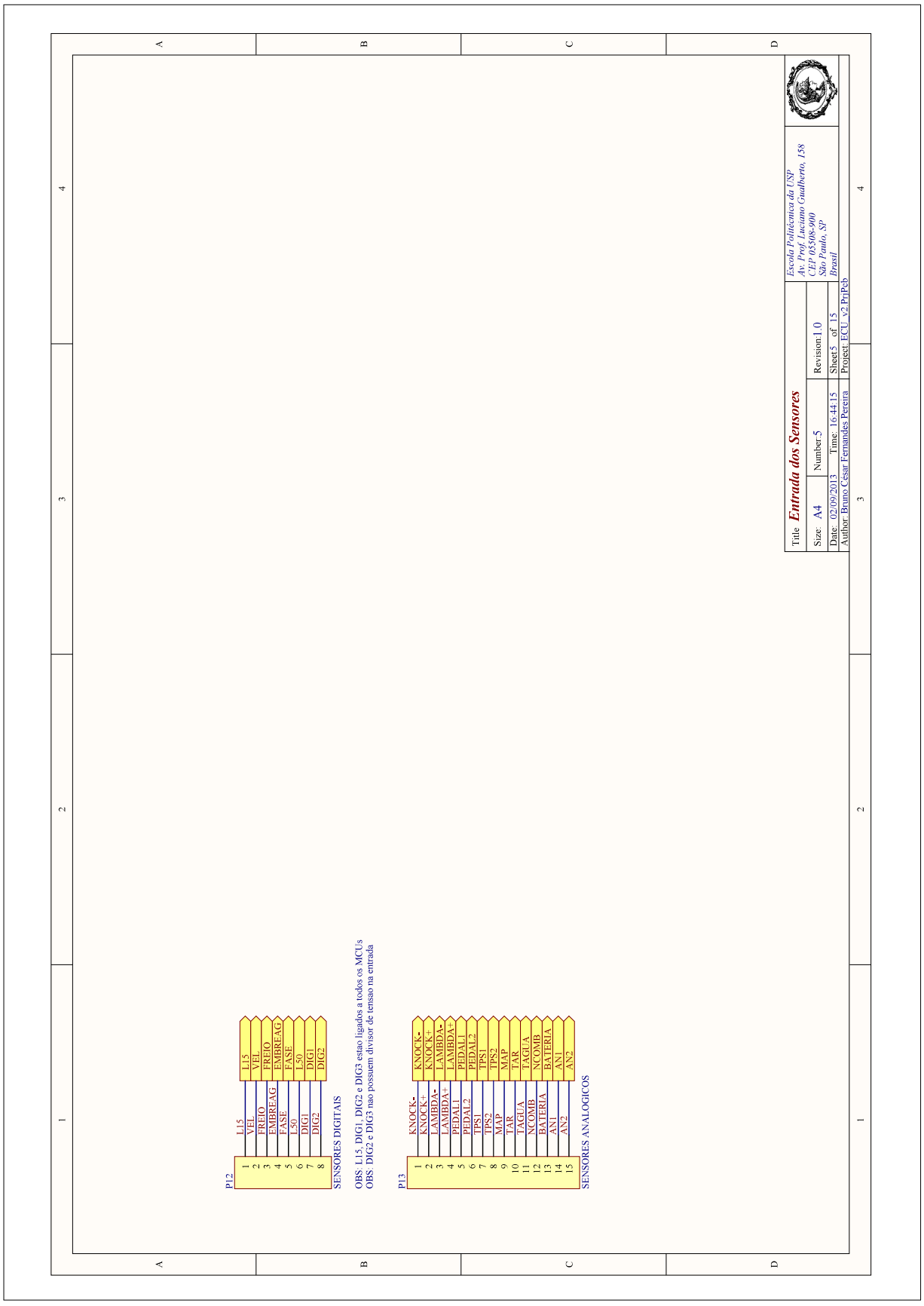
Fonte: o autor

Figura 92 – Microcontrolador de Comunicação



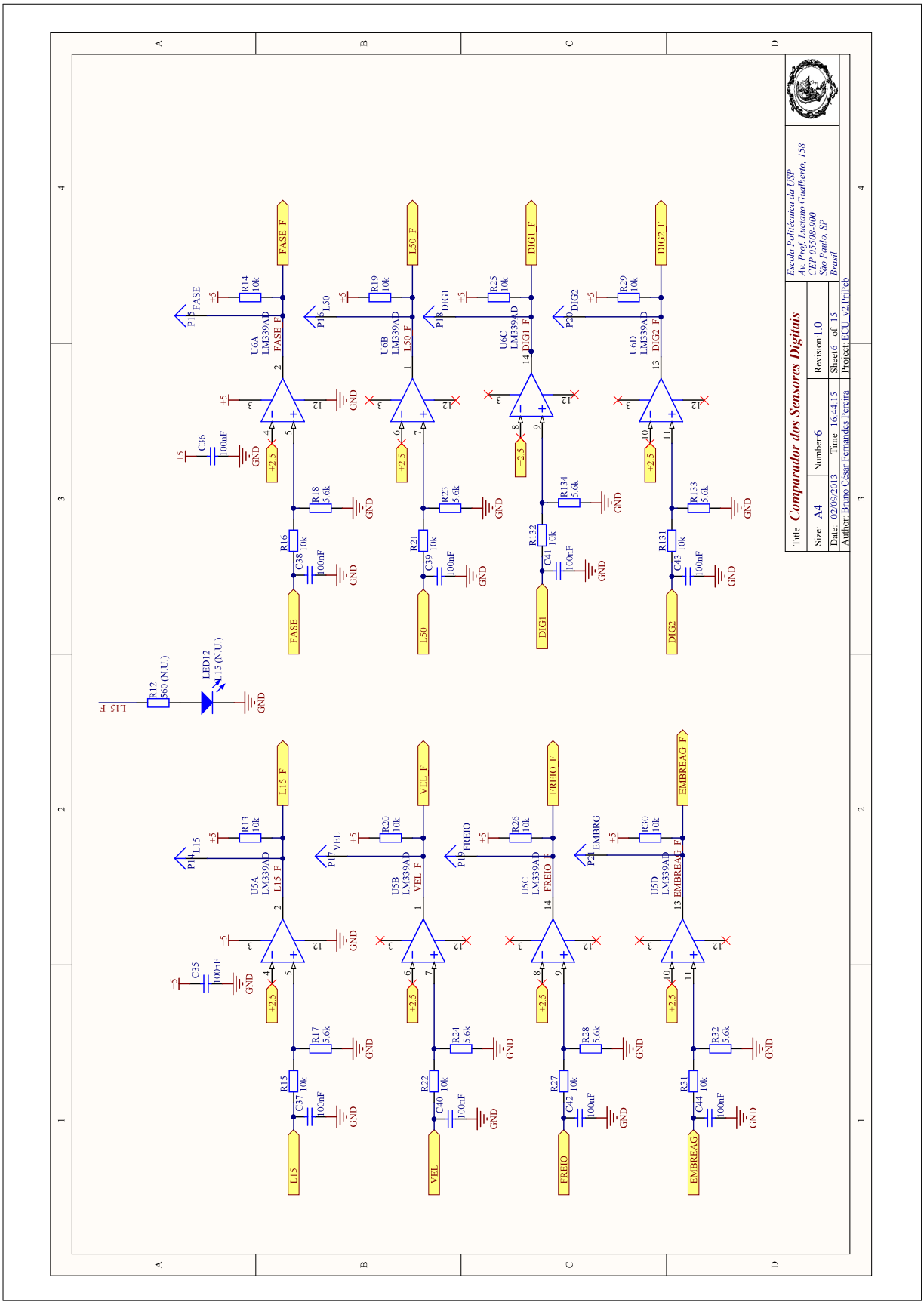
Fonte: o autor

Figura 93 – Sinais de Entrada dos Sensores



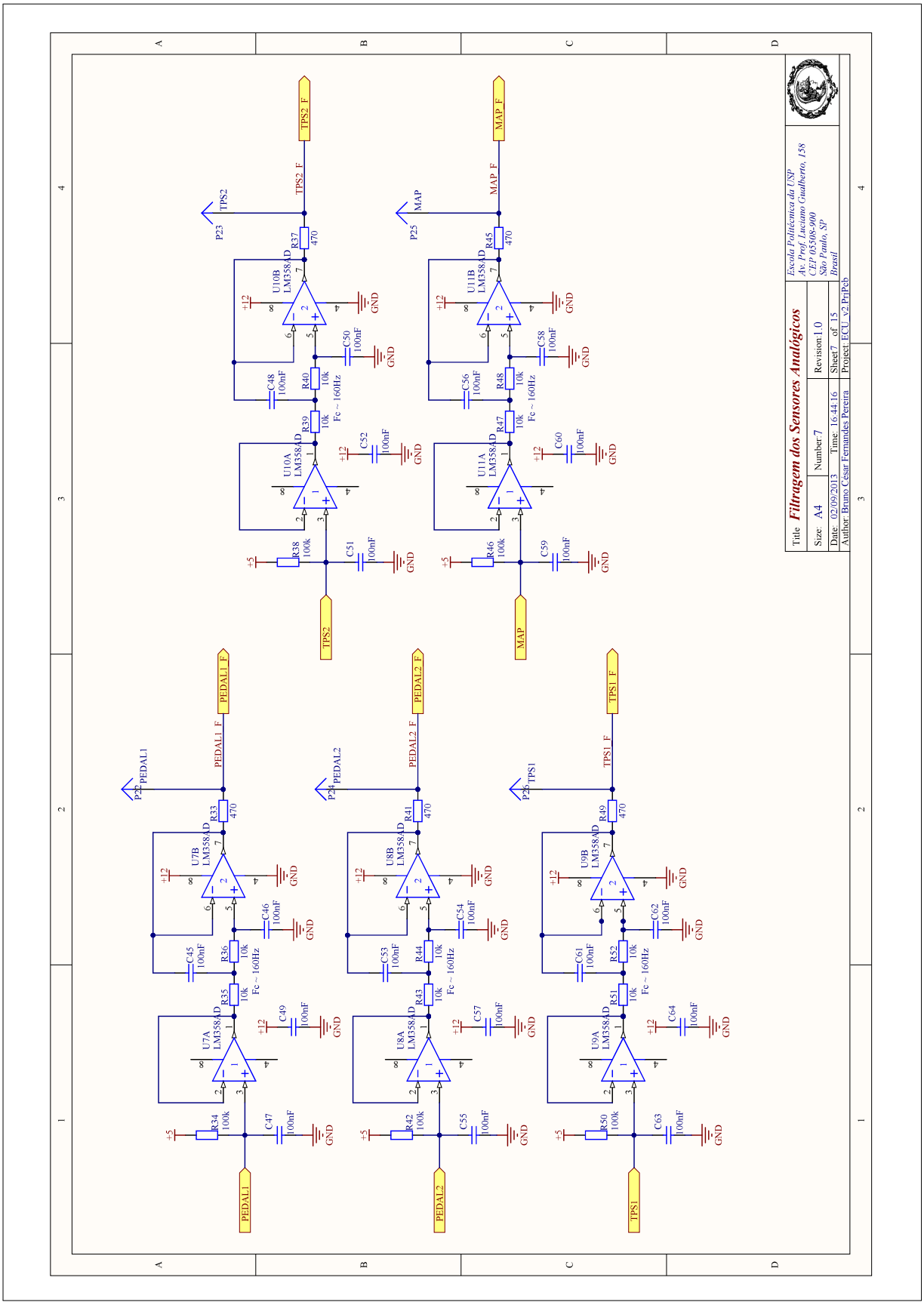
Fonte: o autor

Figura 94 – Comparadores de Tensão



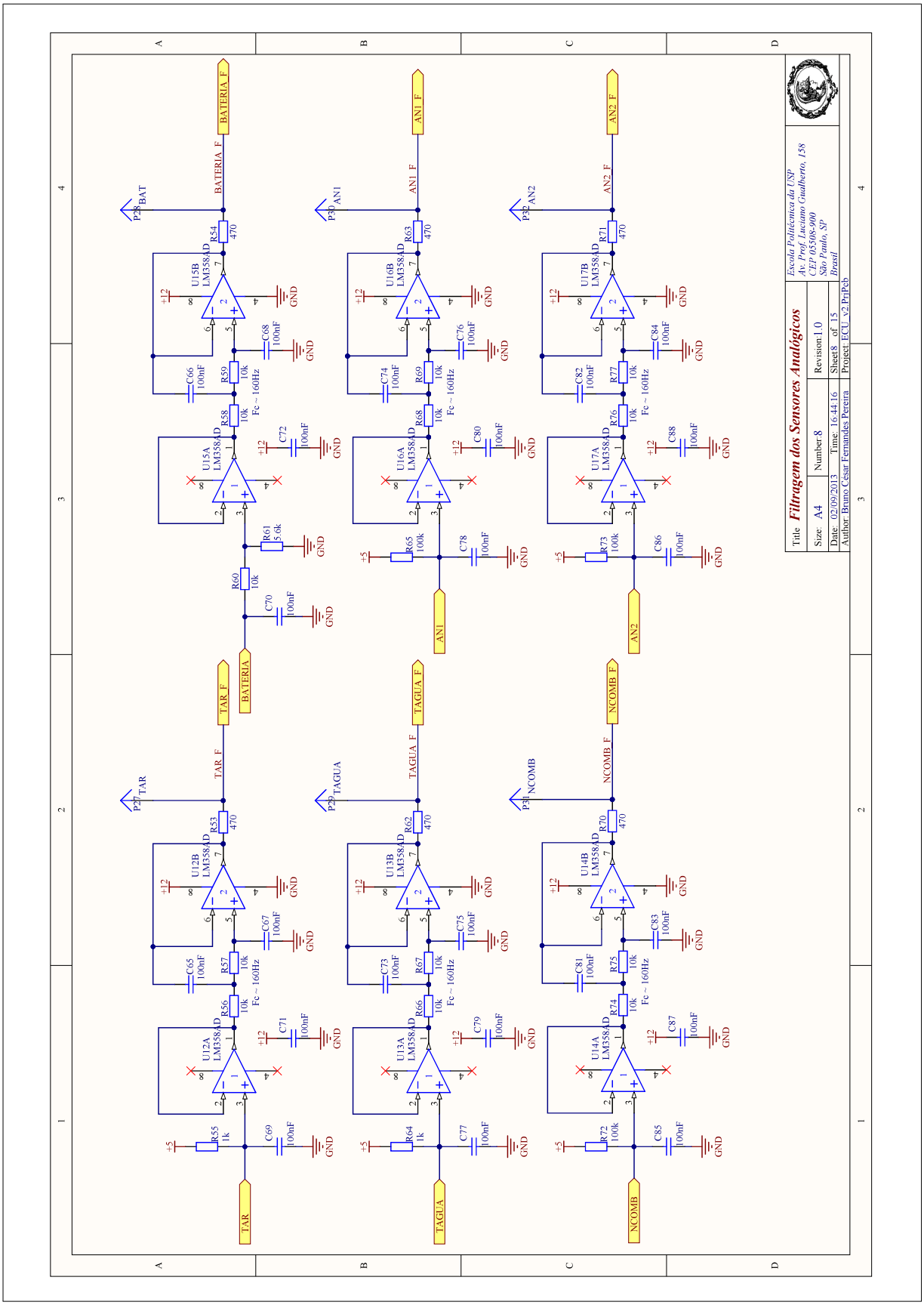
Fonte: o autor

Figura 95 – Filtragem Analógica



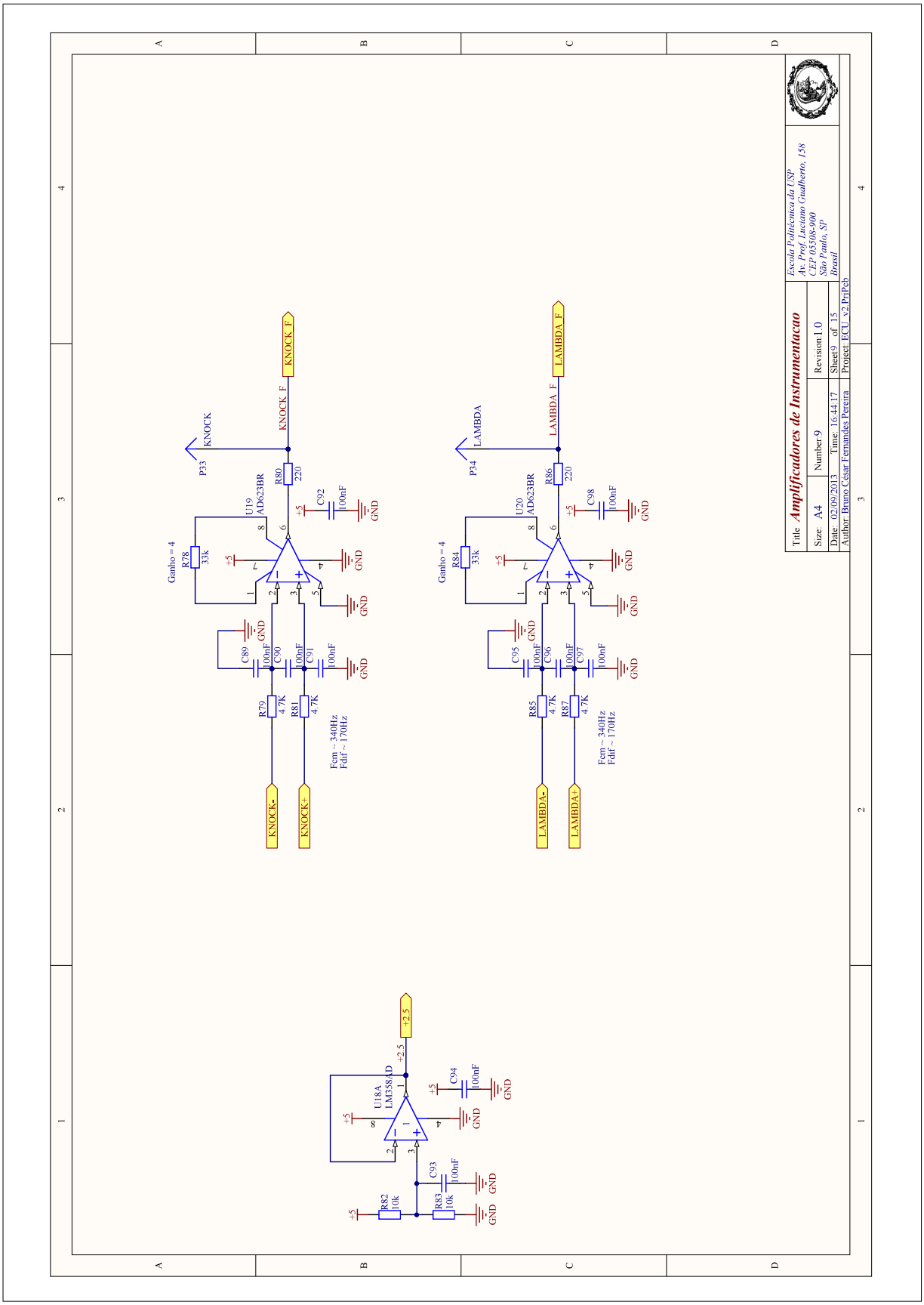
Fonte: o autor

Figura 96 – Filtragem Analógica II



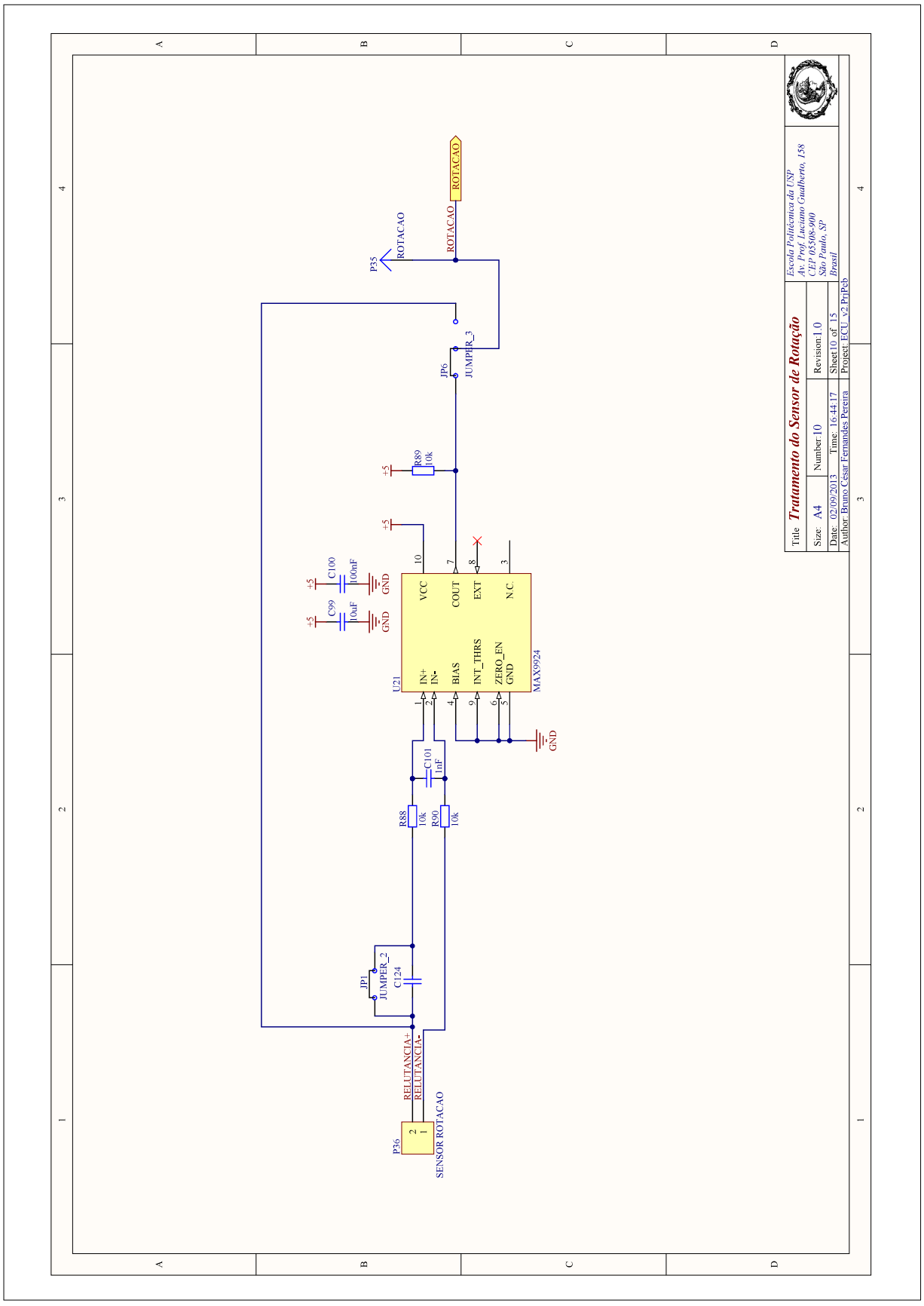
Fonte: o autor

Figura 97 – Amplificadores de Instrumentação



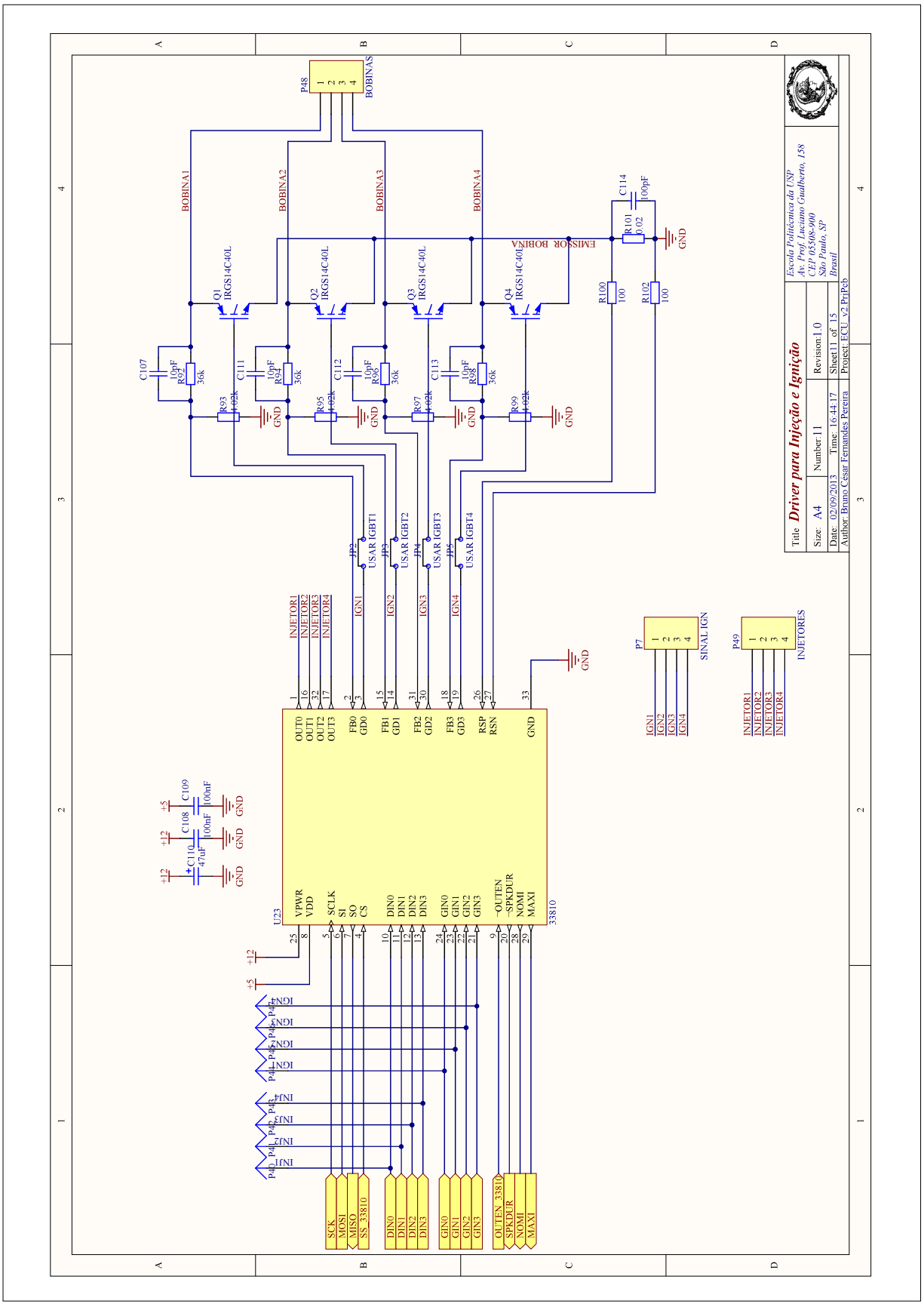
Fonte: o autor

Figura 98 – Tratamento do Sinal do Sensor de Rotação



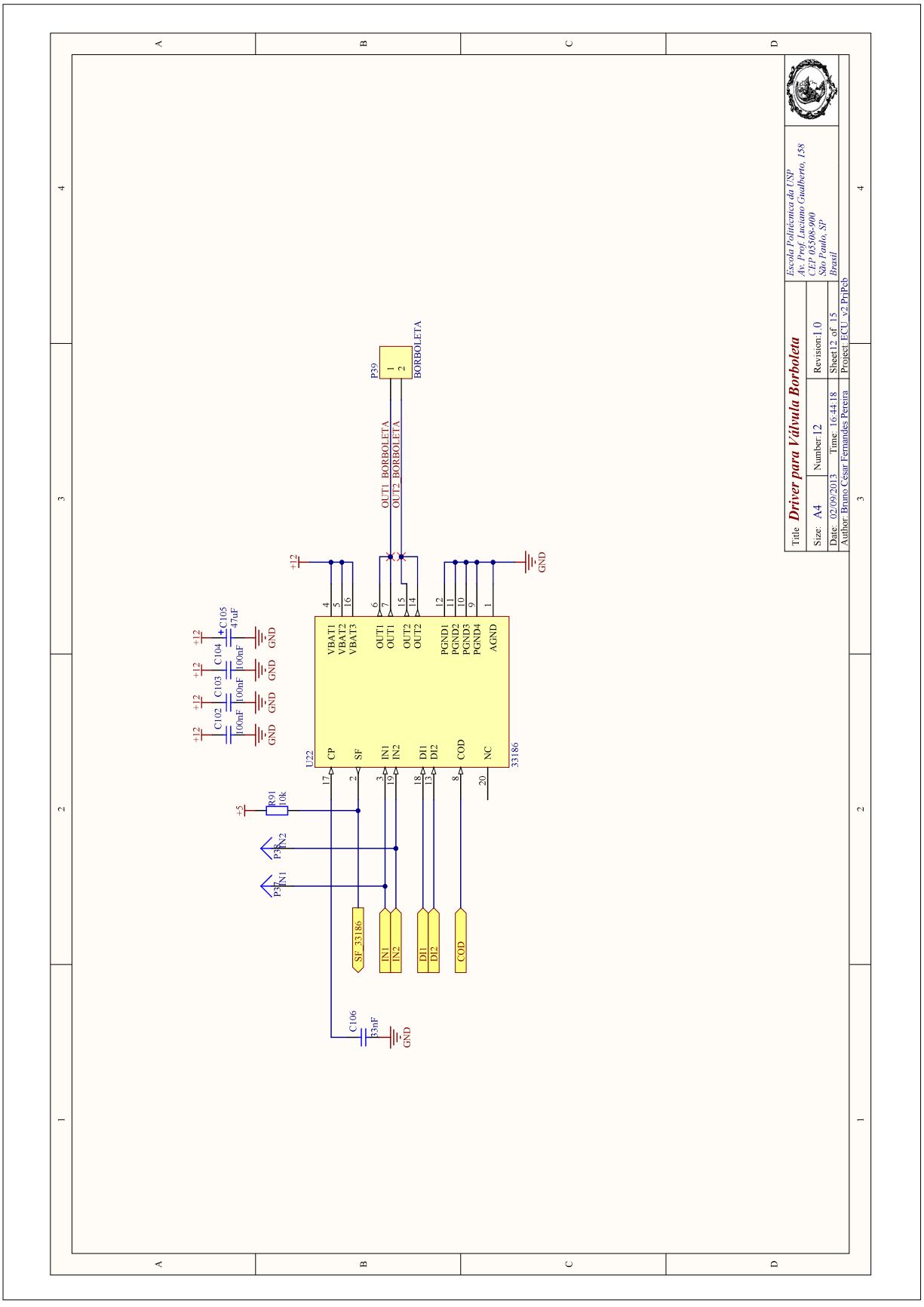
Fonte: o autor

Figura 99 – Driver para Injeção e Ignição



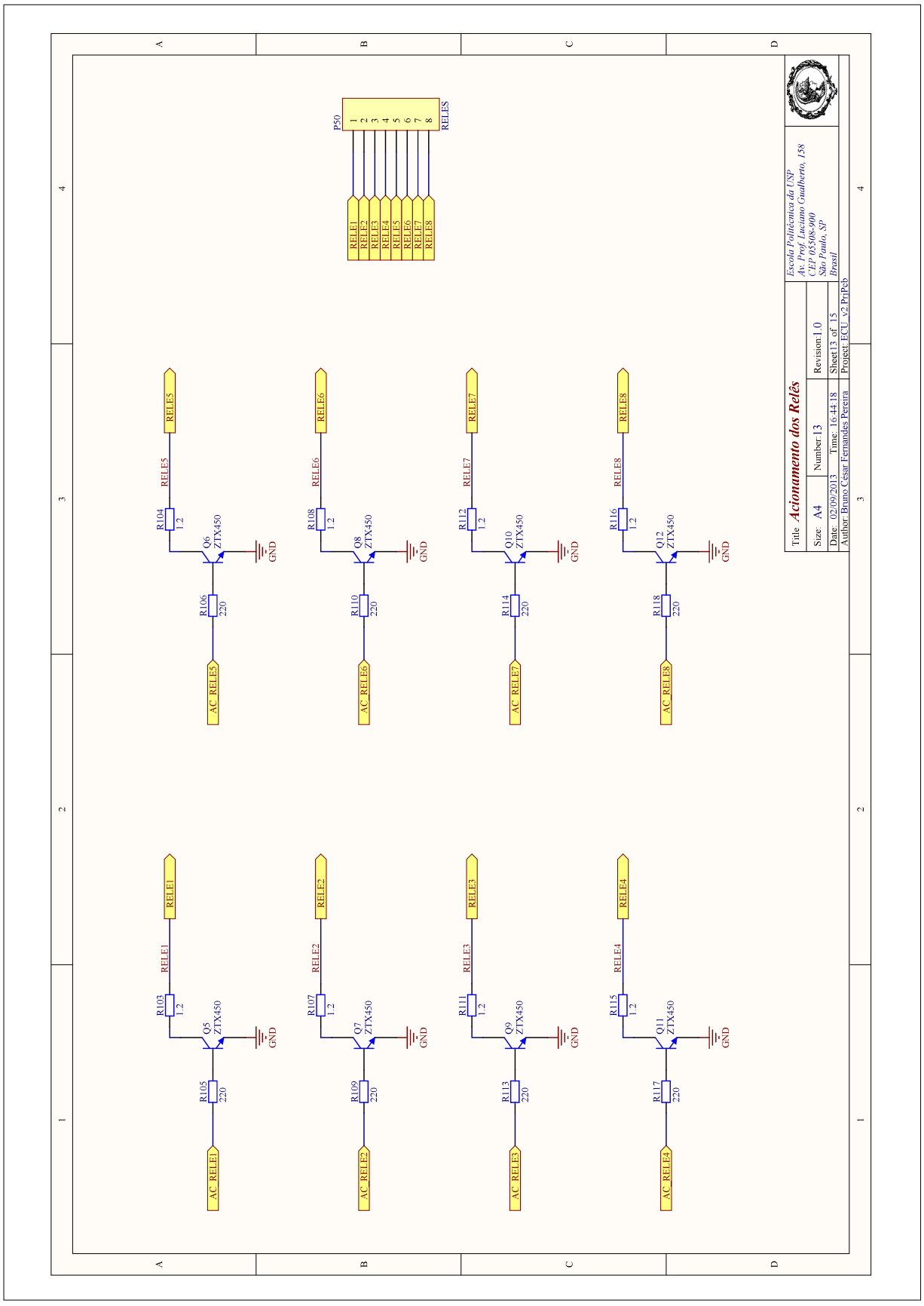
Fonte: o autor

Figura 100 – Driver para Válvula Borboleta (Ponte "‘H"’)



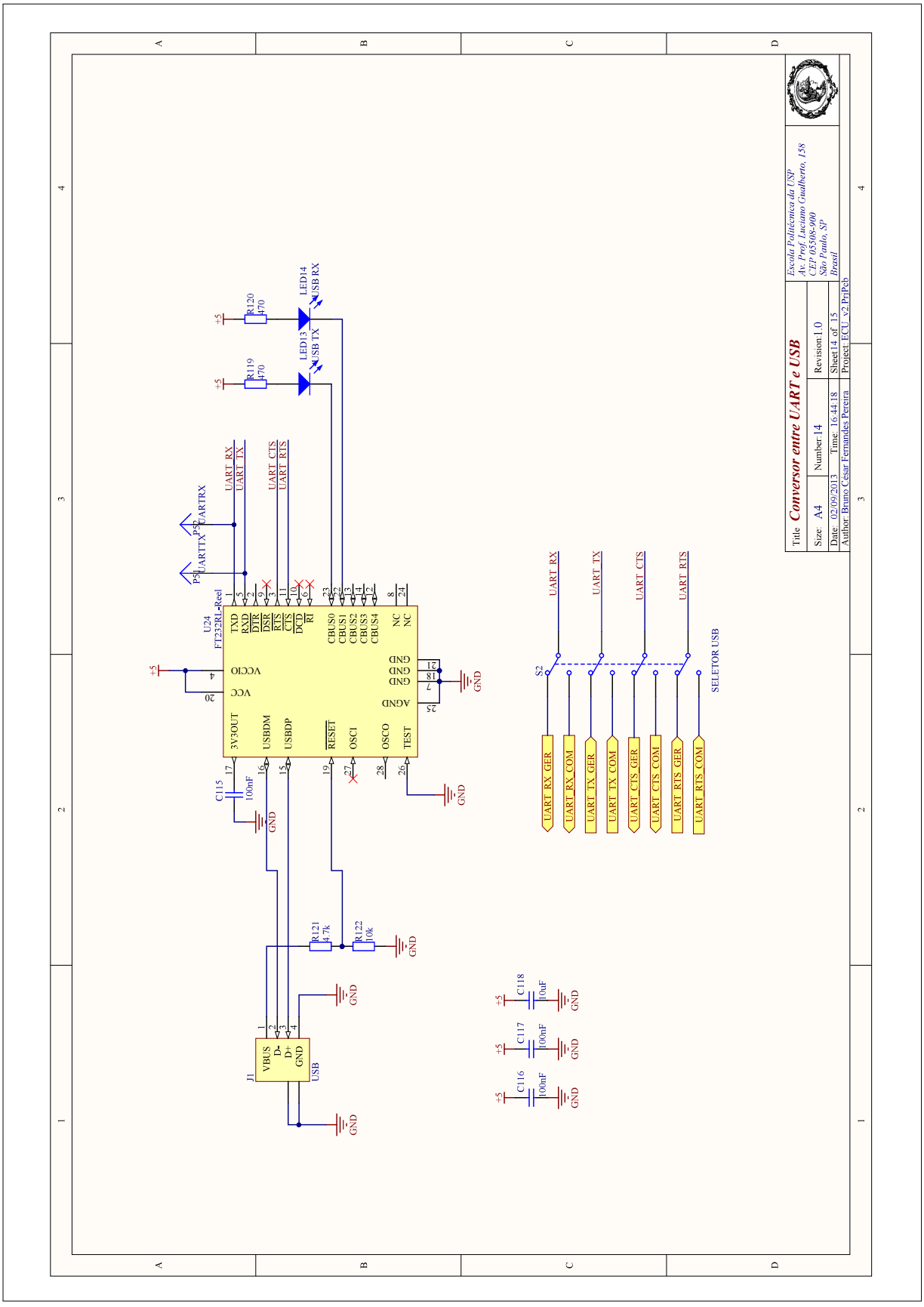
Fonte: o autor

Figura 101 – Acionamento de Relés



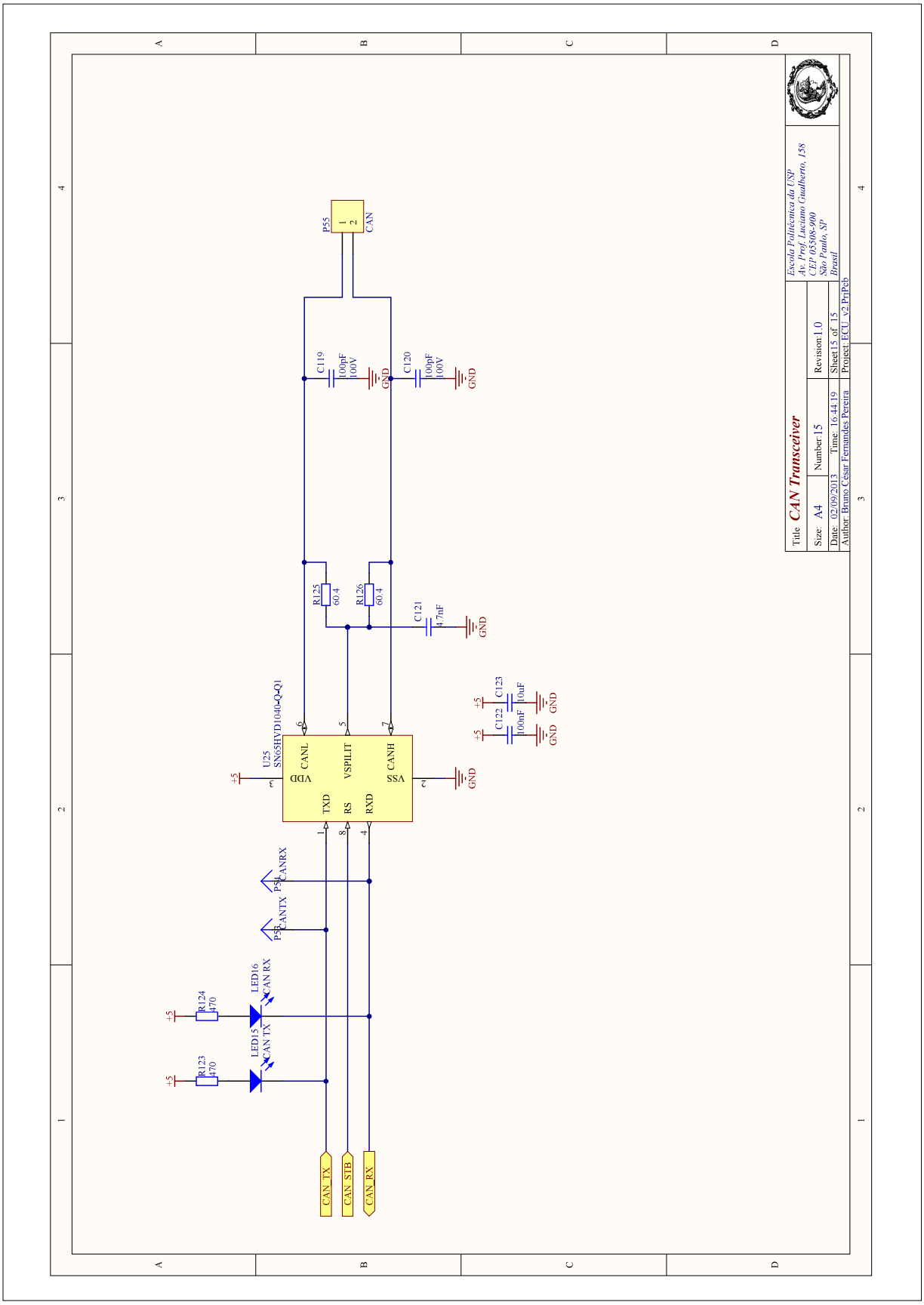
Fonte: o autor

Figura 102 – Conversor UART-USB



Fonte: o autor

Figura 103 – Transceiver CAN

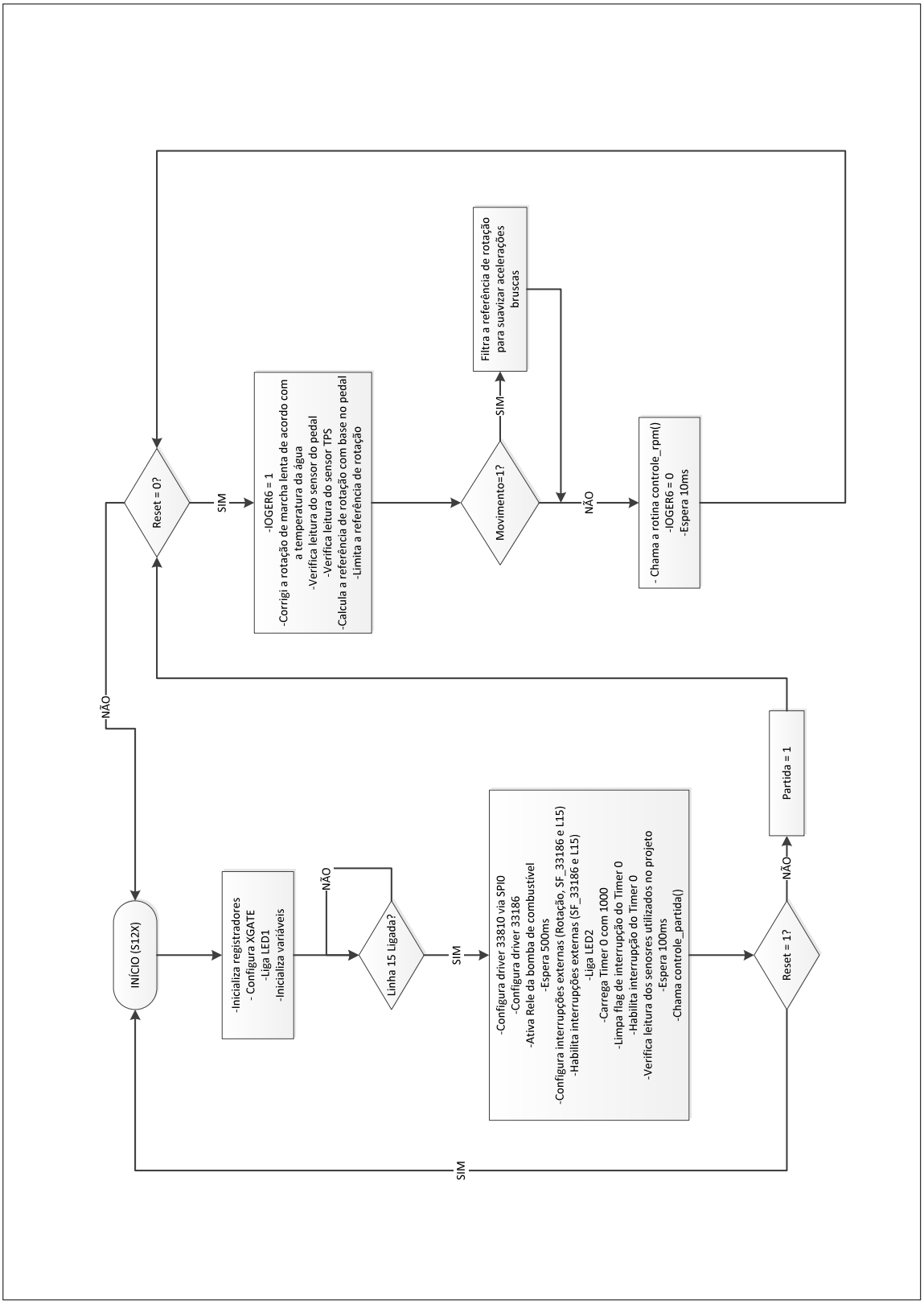


Fonte: o autor

APÊNDICE B – *Firmware*

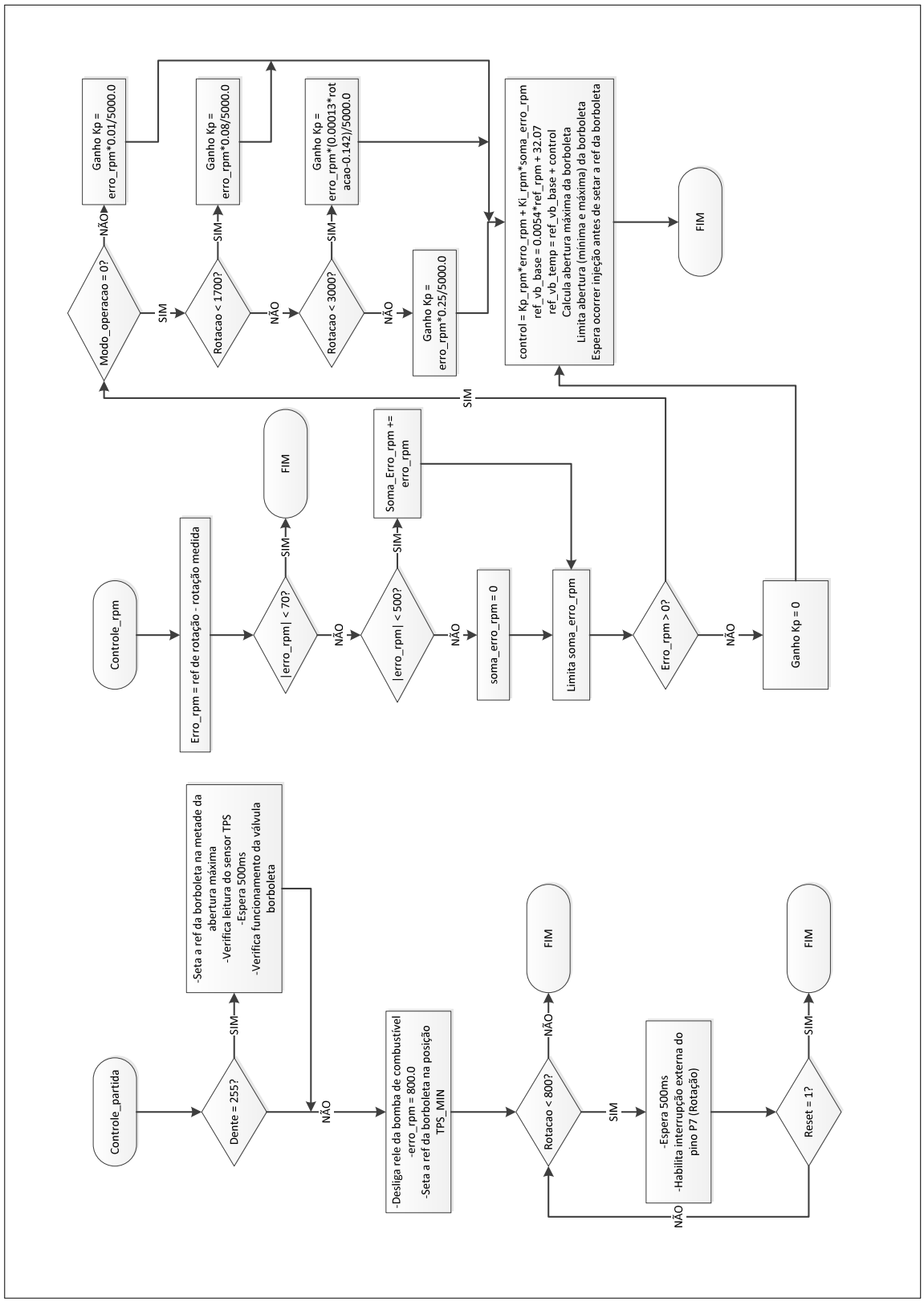
B.1 Gerenciamento

Figura 104 – Fluxograma do *firmware* implementado no uC de Gerenciamento



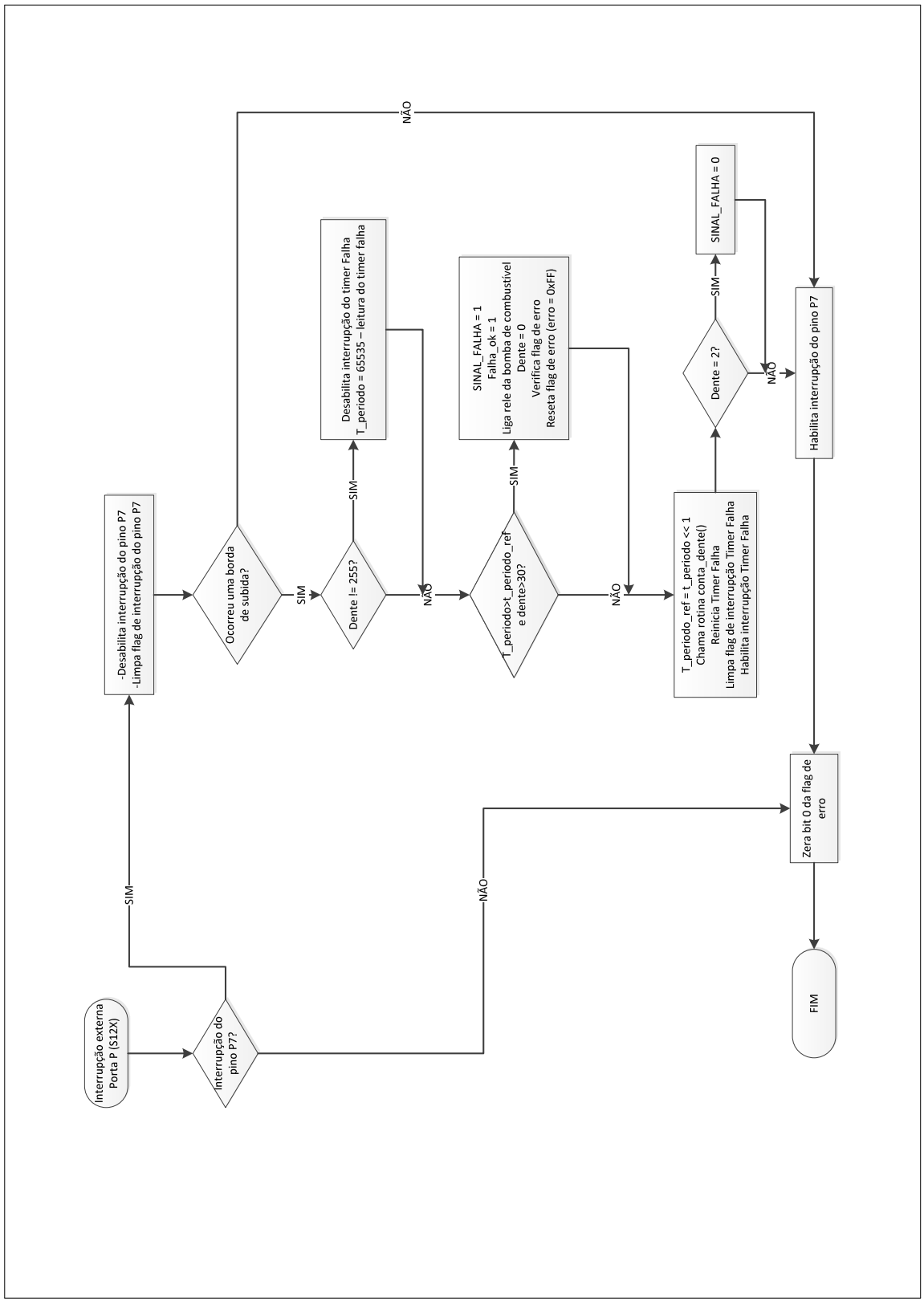
Fonte: o autor

Figura 105 – Fluxograma do *firmware* implementado no uC de Gerenciamento



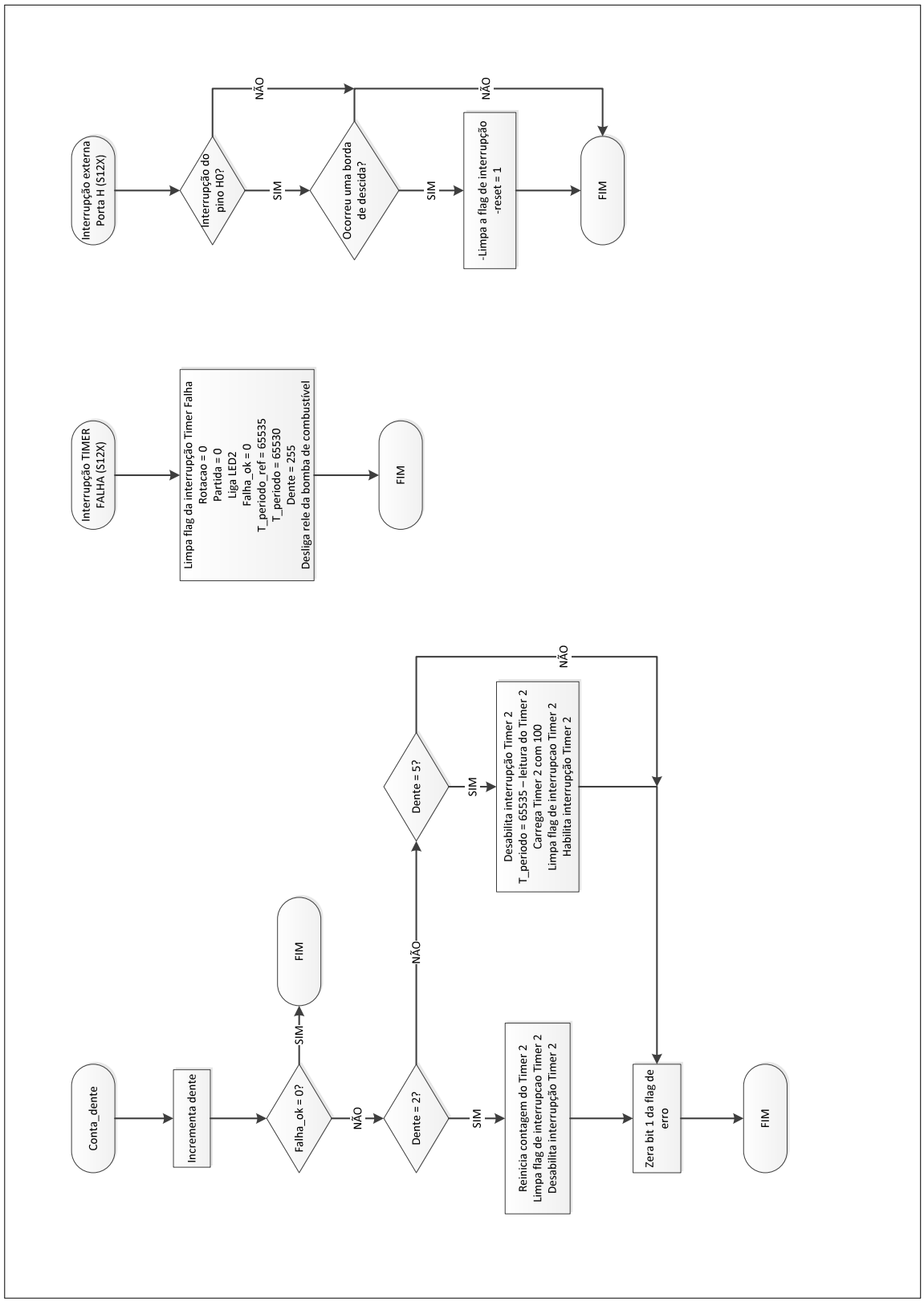
Fonte: o autor

Figura 106 – Fluxograma do *firmware* implementado no uC de Gerenciamento



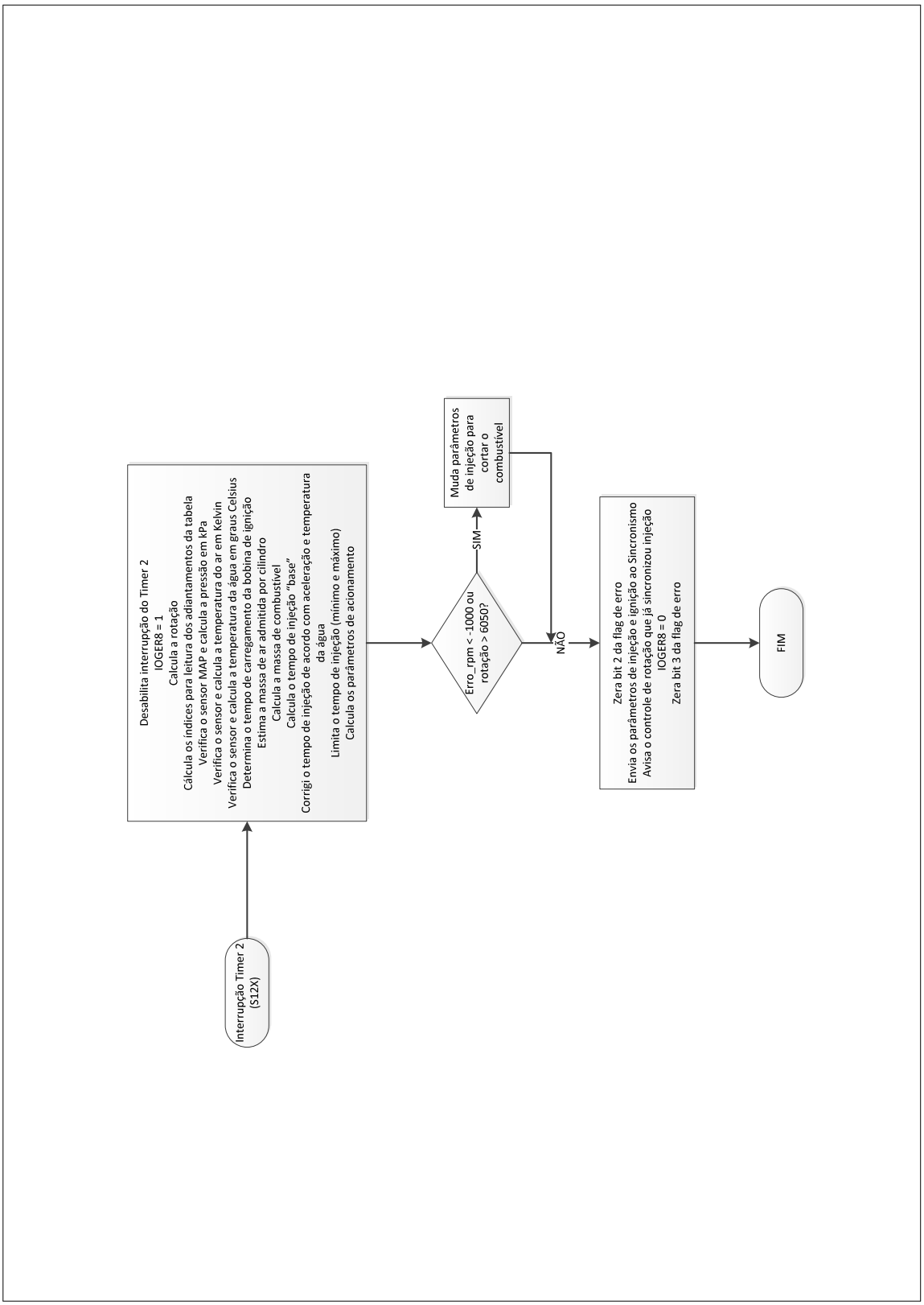
Fonte: o autor

Figura 107 – Fluxograma do *firmware* implementado no uC de Gerenciamento



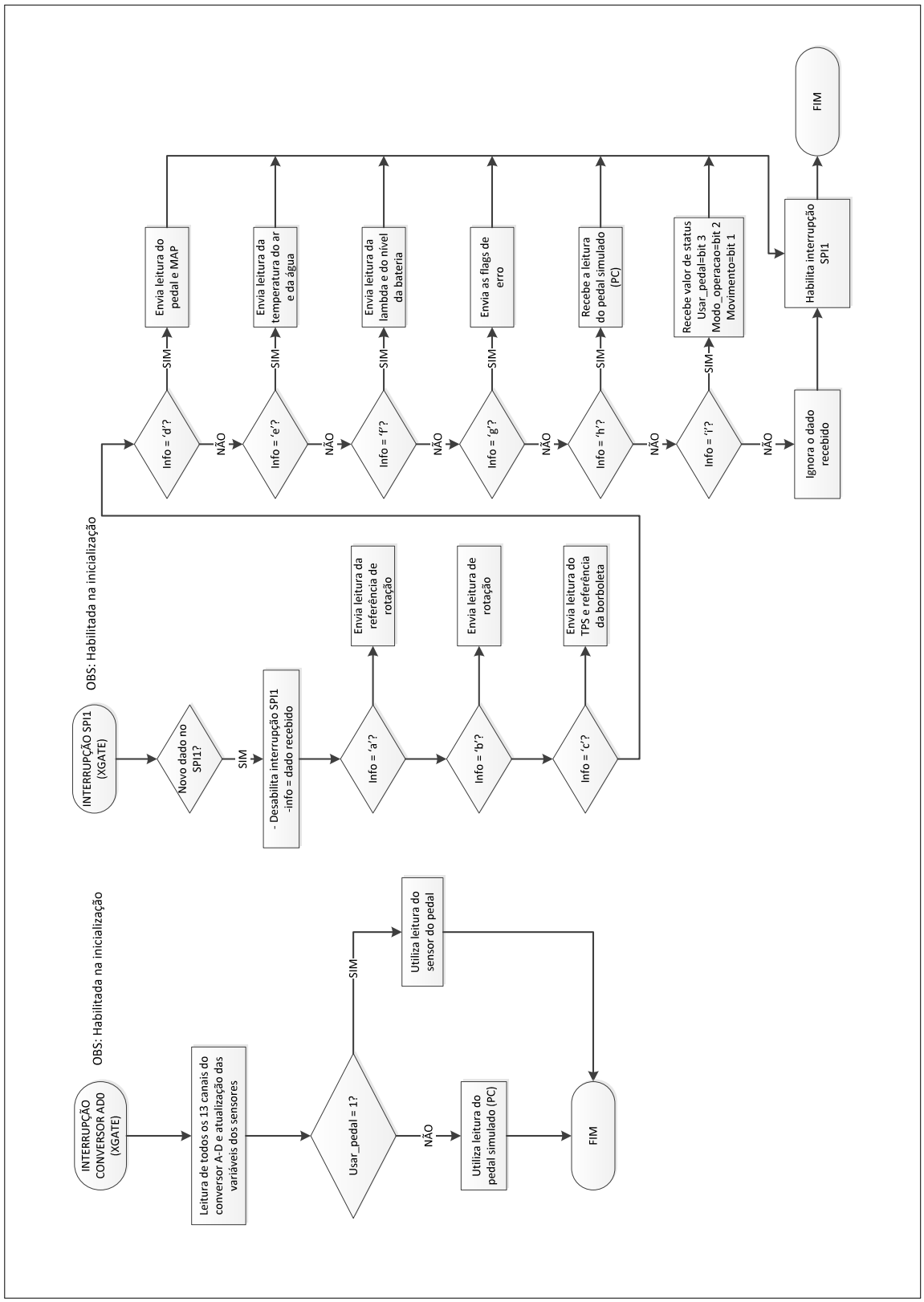
Fonte: o autor

Figura 108 – Fluxograma do *firmware* implementado no uC de Gerenciamento



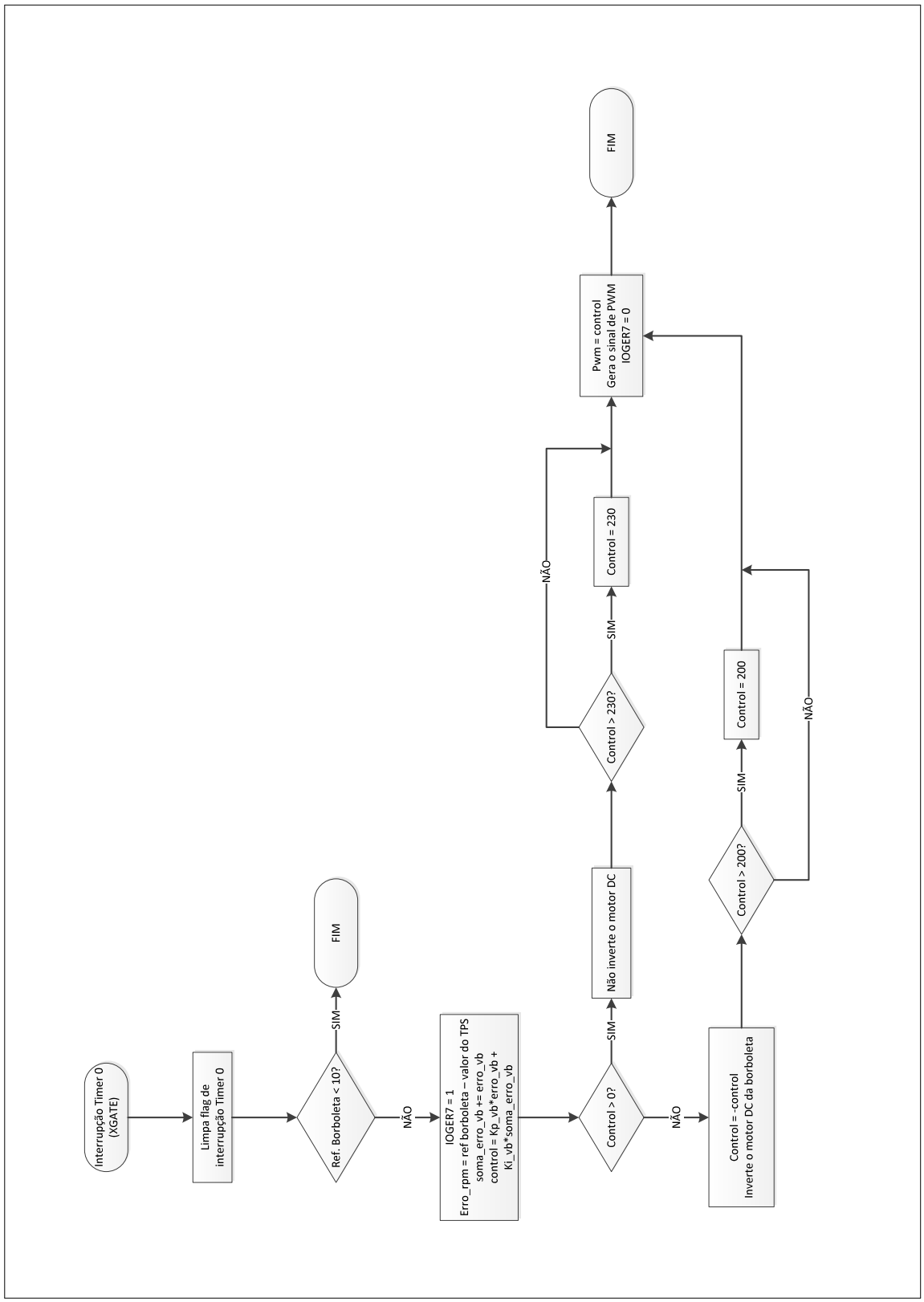
Fonte: o autor

Figura 109 – Fluxograma do *firmware* implementado no uC de Gerenciamento



Fonte: o autor

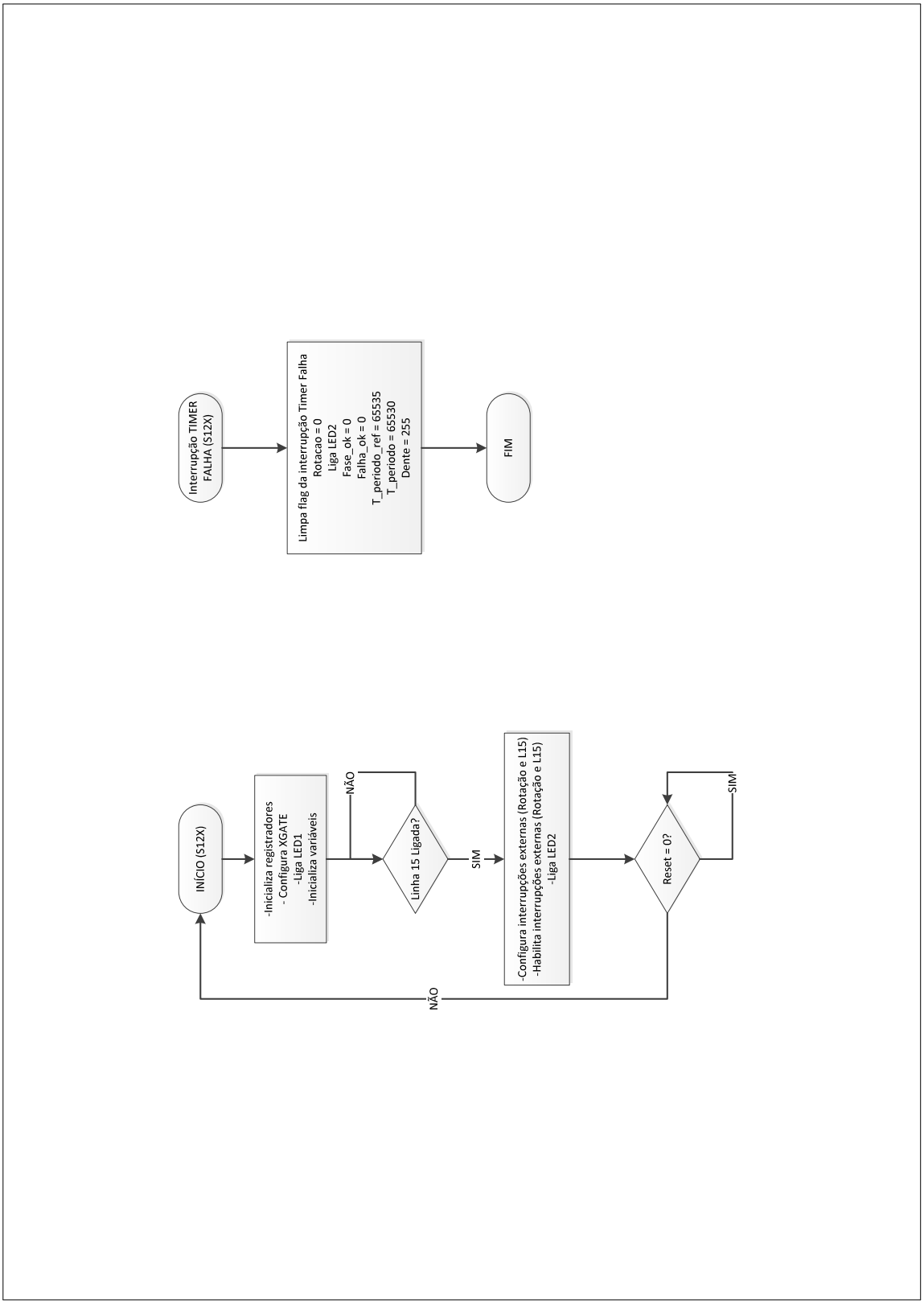
Figura 110 – Fluxograma do *firmware* implementado no uC de Gerenciamento



Fonte: o autor

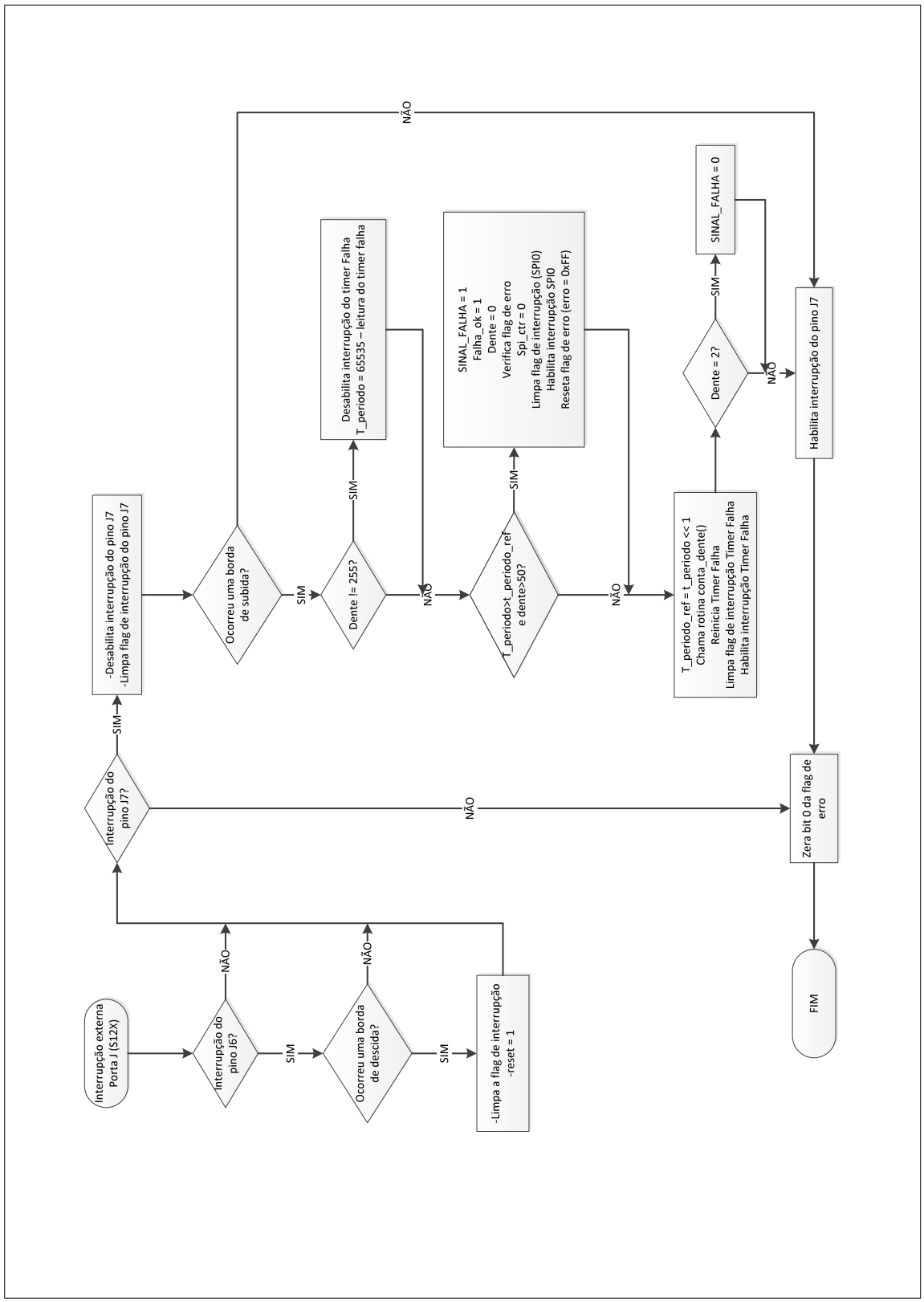
B.2 Sincronismo

Figura 111 – Fluxograma do *firmware* implementado no uC de Sincronismo



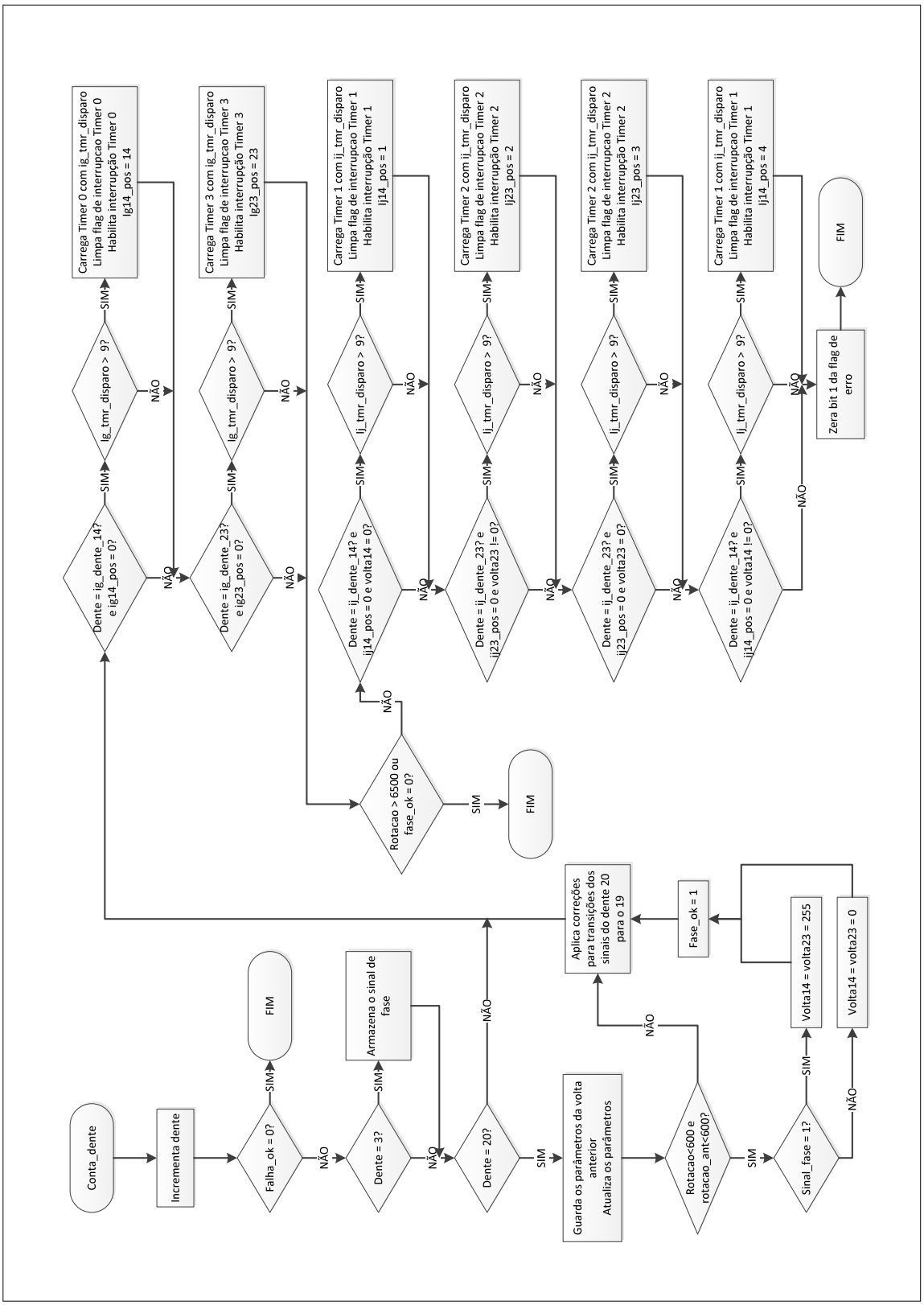
Fonte: o autor

Figura 112 – Fluxograma do *firmware* implementado no uC de Sincronismo



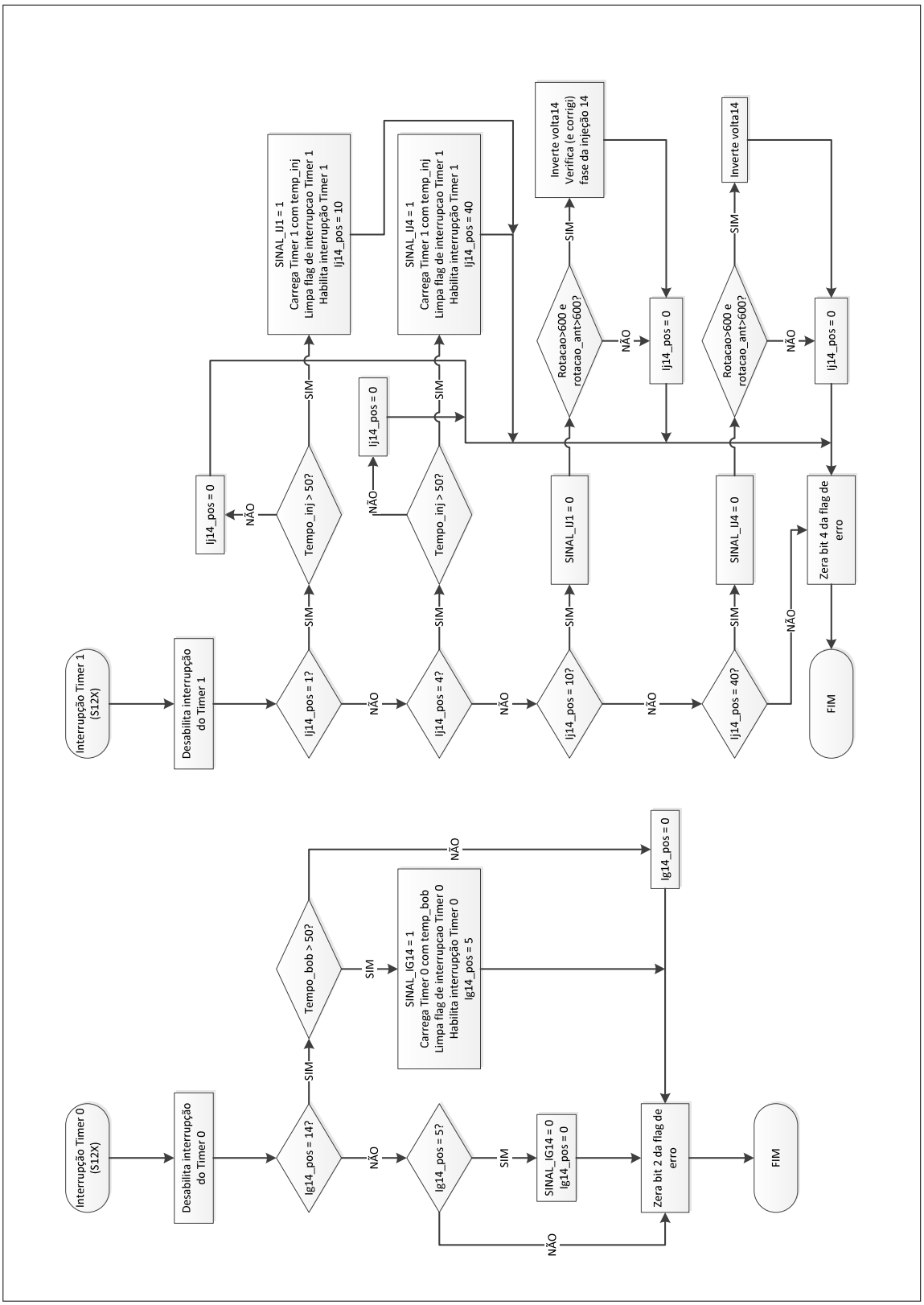
Fonte: o autor

Figura 113 – Fluxograma do *firmware* implementado no uC de Sincronismo



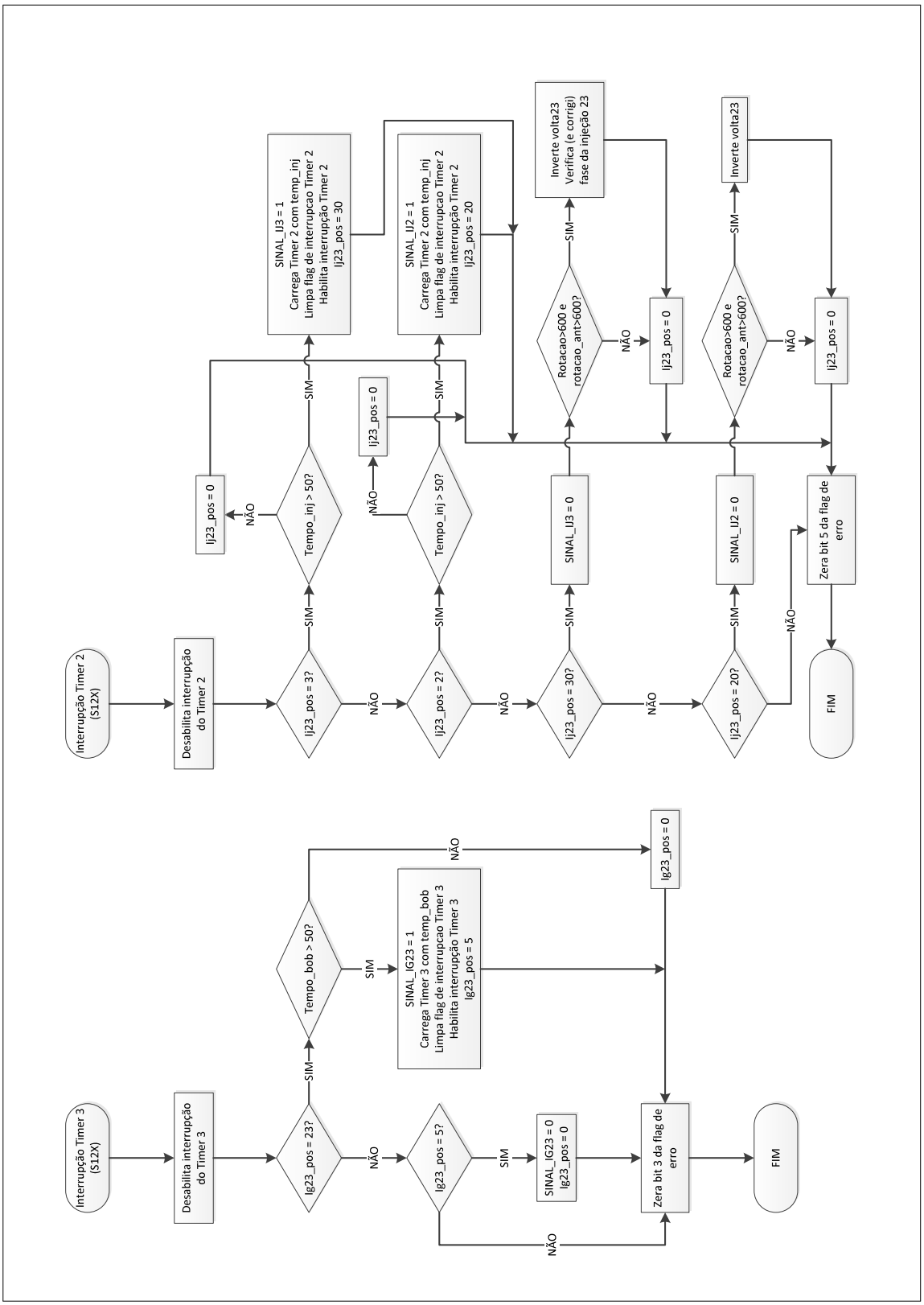
Fonte: o autor

Figura 114 – Fluxograma do *firmware* implementado no uC de Sincronismo



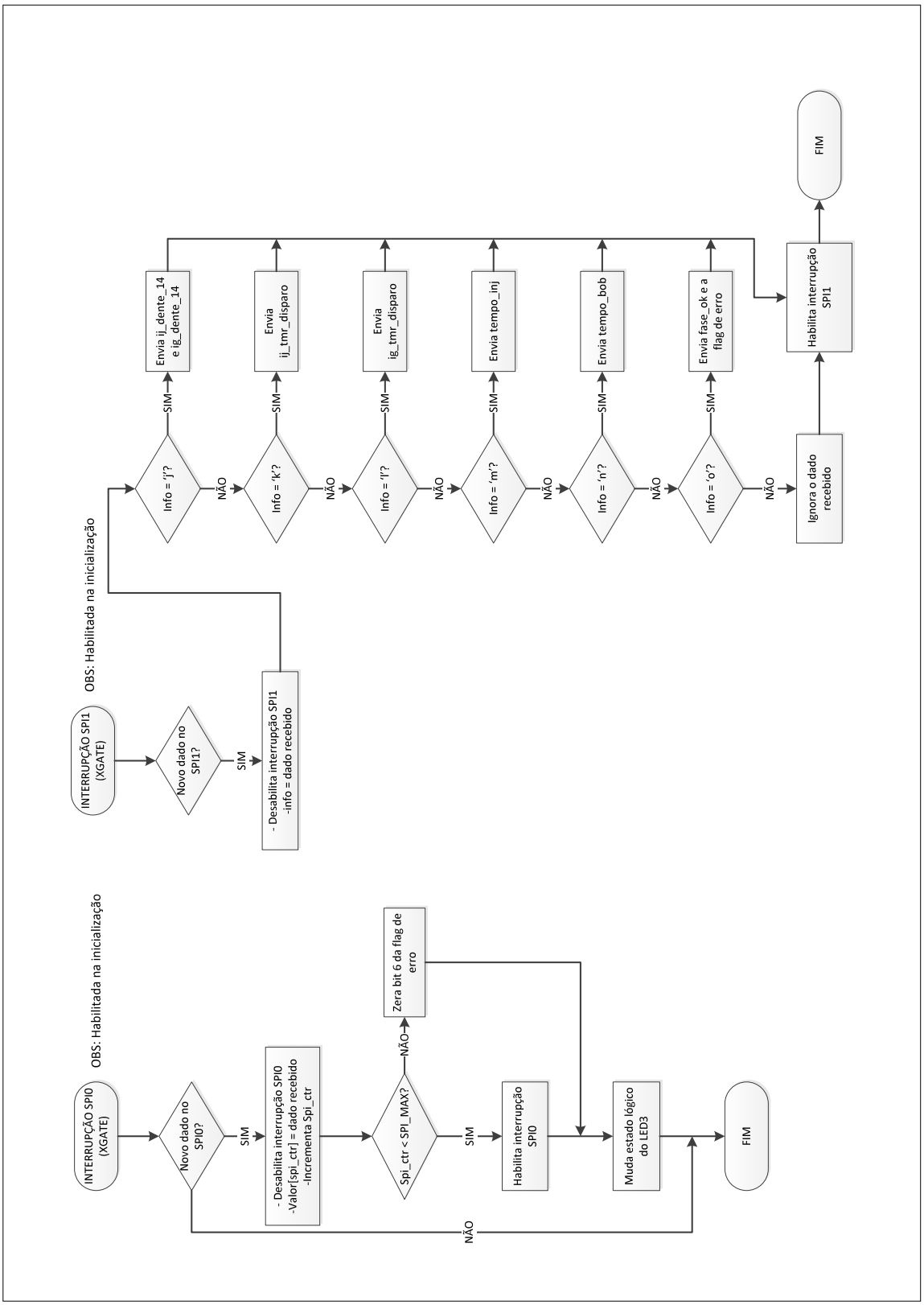
Fonte: o autor

Figura 115 – Fluxograma do *firmware* implementado no uC de Sincronismo



Fonte: o autor

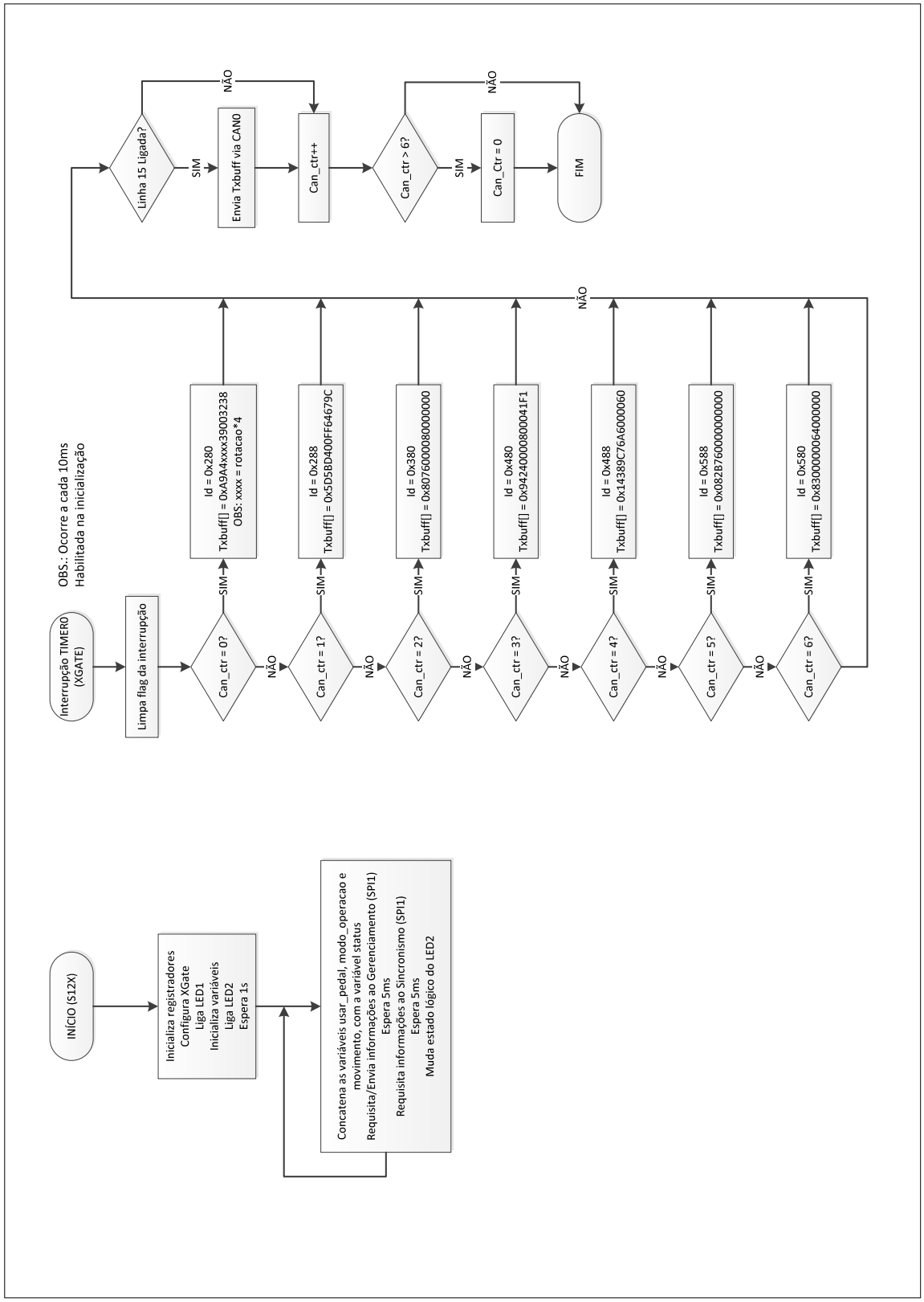
Figura 116 – Fluxograma do *firmware* implementado no uC de Sincronismo



Fonte: o autor

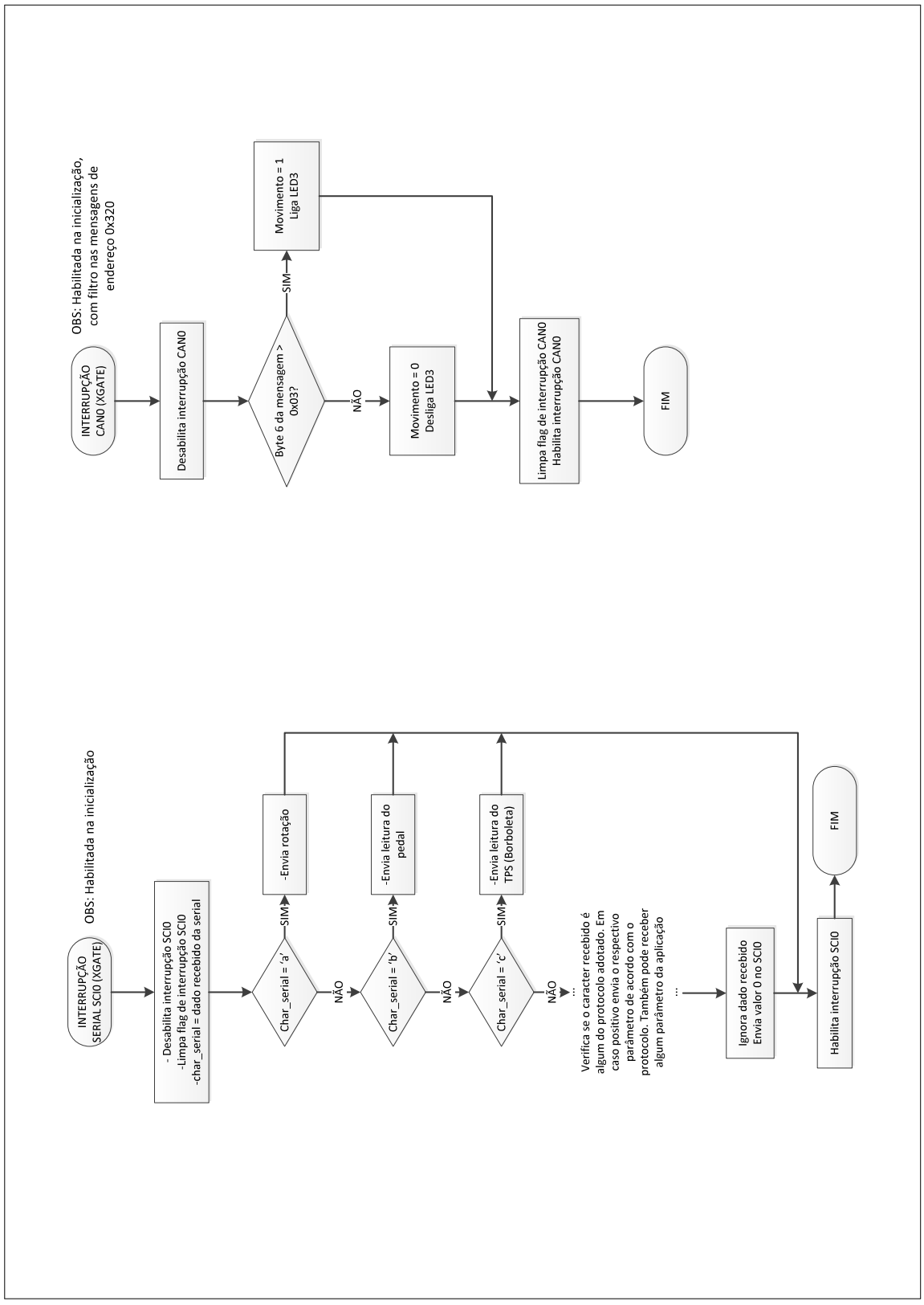
B.3 Comunicação

Figura 117 – Fluxograma do *firmware* implementado no uC de Comunicação



Fonte: o autor

Figura 118 – Fluxograma do *firmware* implementado no uC de Comunicação



Fonte: o autor

APÊNDICE C – *Software* de Monitoramento

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
//using ZedGraph;
using System.Diagnostics;
using System.Threading;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        bool medindo;
        uint erro;
        int dado;
        double rs, dadof, t;
        byte[] rxbuffer = new byte[1];
        string data_to_file;

        // Calibracoes - Polo
        const int PEDAL_MAX_POLO = 180; // Pedal pressionado
        const int PEDAL_MIN_POLO = 40; // Pedal solto

        // Calibracoes - GM
        const int PEDAL_MAX_GM = 236; // Pedal pressionado
        const int PEDAL_MIN_GM = 40; // Pedal solto

        Stopwatch stopwatch = new Stopwatch();

        public Form1()
        {
            InitializeComponent();

            // Serial Configuration
            portacom_comboBox.Items.AddRange(SerialPort.GetPortNames());
```

```
            // Pega as portas COM disponiveis
        baudrate_comboBox.Items.Add("2400");
        baudrate_comboBox.Items.Add("4800");
        baudrate_comboBox.Items.Add("7200");
        baudrate_comboBox.Items.Add("9600");
        baudrate_comboBox.Items.Add("14400");
        baudrate_comboBox.Items.Add("19200");
        baudrate_comboBox.Items.Add("38400");
        baudrate_comboBox.Items.Add("57600");
        baudrate_comboBox.Items.Add("115200");
        baudrate_comboBox.Items.Add("125000");
        baudrate_comboBox.Items.Add("230400");
        baudrate_comboBox.Items.Add("256000");
        baudrate_comboBox.Items.Add("460800");
        baudrate_comboBox.Items.Add("921600");
        baudrate_comboBox.Items.Add("1562500");
        baudrate_comboBox.Items.Add("3125000");
        baudrate_comboBox.SelectedIndex = 9; // Baud Rate default
        serialPort1.DataBits = 8;
        serialPort1.ReadTimeout = 500; // 500ms de timeout

        pedal_comboBox.Items.Add("PEDAL VW POLO");
        pedal_comboBox.Items.Add("PEDAL GM");

        modo_comboBox.Items.Add("NORMAL");
        modo_comboBox.Items.Add("ÔECONMICO");

        pedal_comboBox.SelectedIndex = 0; // Default corresponde ao
            pedal do Polo
        //modo_comboBox.SelectedIndex = 0; // Default corresponde ao
            modo Normal

        medindo = false;
        progressBar1.Value = progressBar1.Minimum;
        erro = 0;
    }

    private void abrirserial_button_Click(object sender, EventArgs e
        )
    {
        if (serialPort1.IsOpen == false)
        {
            if (portacom_comboBox.SelectedItem == null ||
                baudrate_comboBox.SelectedItem == null)
                MessageBox.Show("Selecione a porta COM e/ou o Baud
                    Rate!", "Erro de Comunicacao!", MessageBoxButtons
                    .OK, MessageBoxIcon.Warning);
```

```
            else
            {
                abrirserial_button.Text = "Fechar Serial";
                serialPort1.PortName = (string)portacom_comboBox.
                    SelectedItem; // Set the serial Port
                serialPort1.BaudRate = Convert.ToInt32(
                    baudrate_comboBox.SelectedItem.ToString());
                serialPort1.Open();
            }
        }
        else
        {
            abrirserial_button.Text = "Abrir Serial";
            medindo = false;
            serialPort1.Close();
        }
    }

    private void salvar_button_Click(object sender, EventArgs e)
    {
        if (serialPort1.IsOpen == true)
        {
            if (data_to_file == null)
                MessageBox.Show("Nao áh dados para salvar!", "Erro!"
                    , MessageBoxButtons.OK, MessageBoxIcon.Warning);
            else {
                if (saveFileDialog1.ShowDialog() == DialogResult.OK)
                {
                    if (saveFileDialog1.FileName != "")
                    {
                        using (System.IO.TextWriter tw = new System.
                            IO.StreamWriter(saveFileDialog1.FileName)
                            )
                        {
                            tw.WriteLine(data_to_file);
                        }
                    }
                    else
                        MessageBox.Show("Nome áInvlido!", "Erro",
                            MessageBoxButtons.OK, MessageBoxIcon.
                            Error);
                }
            }
        }
        else
            MessageBox.Show("A porta esta fechada!", "Erro de
                Comunicacao!", MessageBoxButtons.OK, MessageBoxIcon.
```

```
                Warning);
        }

        private void lervalores_button_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen == true)
            {
                if (medindo == false) // Caso ãno esteja medindo
                {
                    // Habilita timer para medir o tempo...
                    stopwatch.Reset();
                    stopwatch.Start();
                    t = 0;

                    // Preparando a string que sera escrita no arquivo
                    data_to_file = "Diagnose e Monitoramento - ECU v2.0\
                        r\n\r\n";
                    if (modo_comboBox.SelectedIndex == 1)
                        data_to_file += "Modo de çãoperao: ôEconmico\r\n
                            \r\n";
                    else
                        data_to_file += "Modo de çãoperao: Normal\r\n\r\
                            n";
                    data_to_file += "Tempo(s)";
                    data_to_file += "\t\tReferencia(RPM)";
                    data_to_file += "\tRotacao(RPM)";
                    data_to_file += "\tRef. (TPS)";
                    if (medidas_checkBox.Checked == true)
                        data_to_file += "(V)";
                    data_to_file += "\t\tBorboleta";
                    if (medidas_checkBox.Checked == true)
                        data_to_file += "(V)";
                    data_to_file += "\t\tMAP";
                    if (medidas_checkBox.Checked == true)
                        data_to_file += "(kPa)";
                    data_to_file += "\t\tT_Injecao(ms)";
                    //data_to_file += "\tLambda";
                    //if (medidas_checkBox.Checked == true)
                    //    data_to_file += "(V)";
                    data_to_file += "\r\n\r\n";


                    medindo = true;
                    lervalores_button.Text = "PARAR"; // Aproveita o
                        mesmo botao para parar a aquisicao

                    while (medindo == true)
```

```
        {
            Application.DoEvents();

            try
            {
                // Requisita çõinformaes ao Comunicacao
                   segundo o protocolo

                serialPort1.Write("a");
                // espera resposta...
                dado = serialPort1.ReadByte();
                dado = dado << 8;
                dado += serialPort1.ReadByte();
                rotacao_textBox.Text = dado.ToString("F0");

                Application.DoEvents();

                serialPort1.Write("b");
                // espera resposta...
                dado = serialPort1.ReadByte();
                if (medidas_checkBox.Checked == true)
                {
                    dadof = (5.0 * dado) / 255.0;
                    pedal_textBox.Text = dadof.ToString("F2"
                       );
                }
                else
                    pedal_textBox.Text = dado.ToString("F0")
                       ;



                Application.DoEvents();

                serialPort1.Write("c");
                // espera resposta...
                dado = serialPort1.ReadByte();
                if (medidas_checkBox.Checked == true)
                {
                    dadof = (5.0 * dado) / 255.0;
                    valvula_textBox.Text = dadof.ToString("
                       F2");
                }
                else
                    valvula_textBox.Text = dado.ToString("F0
                       ");
```

```
                Application.DoEvents();

                serialPort1.Write("d");
                // espera resposta...
                dado = serialPort1.ReadByte();
                dado = dado << 8;
                dado += serialPort1.ReadByte();
                refrpm_textBox.Text = dado.ToString("F0");

                Application.DoEvents();

                serialPort1.Write("e");
                // espera resposta...
                dado = serialPort1.ReadByte();
                if (medidas_checkBox.Checked == true)
                {
                    dadof = 0.442 * dado + 7.157; // Unidade
                        : kPa
                    map_textBox.Text = dadof.ToString("F1");
                }
                else
                    map_textBox.Text = dado.ToString("F0");

                Application.DoEvents();

                serialPort1.Write("f");
                // espera resposta...
                dado = serialPort1.ReadByte();
                if (medidas_checkBox.Checked == true)
                {
                    // Calculo da resistencia do sensor
                    rs = dado / (255.0 - dado); // Unidade:
                        KOhms

                    // Calculo da temperatura com base na
                        interpolacao (curva do sensor)
                    dadof = -27.25 * Math.Log(rs) + 44.52;
                        // Unidade: Celsius

                    tar_textBox.Text = dadof.ToString("F1");
                }
                else
                    tar_textBox.Text = dado.ToString("F0");

                Application.DoEvents();

                serialPort1.Write("g");
```

```
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    if (medidas_checkBox.Checked == true)
                    {
                        // Calculo da resistencia do sensor
                        rs = dado / (255.0 - dado); // Unidade:
                            KOhms

                        // Calculo da temperatura com base na
                            interpolacao (curva do sensor)
                        dadof = -30.57 * Math.Log(rs) + 43.01;
                            // Unidade: Celsius

                        tagua_textBox.Text = dadof.ToString("F1"
                            );
                    }
                    else
                        tagua_textBox.Text = dado.ToString("F0")
                            ;

                    Application.DoEvents();

                    serialPort1.Write("h");
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    if (medidas_checkBox.Checked == true)
                    {
                        dadof = (5.0 * dado) / 255.0;
                        lambda_textBox.Text = dadof.ToString("F2
                            ");
                    }
                    else
                        lambda_textBox.Text = dado.ToString("F0"
                            );

                    Application.DoEvents();

                    serialPort1.Write("i");
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    if (medidas_checkBox.Checked == true)
                    {
                        dadof = (5.0 * 15.6 * dado) / (255.0 *
                            5.6);
                        bateria_textBox.Text = dadof.ToString("
                            F2");
                    }
```

```
                    else
                        bateria_textBox.Text = dado.ToString("F0
                            ");

                    Application.DoEvents();

                    serialPort1.Write("j");
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    if (medidas_checkBox.Checked == true)
                    {
                        dadof = (5.0 * dado) / 255.0;
                        refvb_textBox.Text = dadof.ToString("F2"
                            );
                    }
                    else
                        refvb_textBox.Text = dado.ToString("F0")
                            ;

                    Application.DoEvents();

                    serialPort1.Write("k");
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    ijdente14_textBox.Text = dado.ToString("F0")
                        ;

                    Application.DoEvents();

                    serialPort1.Write("l");
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    dado = dado << 8;
                    dado += serialPort1.ReadByte();
                    ijtempo_textBox.Text = dado.ToString("F0");

                    Application.DoEvents();

                    serialPort1.Write("m");
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    igdente14_textBox.Text = dado.ToString("F0")
                        ;

                    Application.DoEvents();

                    serialPort1.Write("n");
```

```
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    dado = dado << 8;
                    dado += serialPort1.ReadByte();
                    igtempo_textBox.Text = dado.ToString("F0");

                    Application.DoEvents();

                    serialPort1.Write("o");
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    dado = dado << 8;
                    dado += serialPort1.ReadByte();
                    dadof = dado / 1000.0;
                    tempoinjecao_textBox.Text = dadof.ToString("
                        F2");

                    Application.DoEvents();

                    serialPort1.Write("p");
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    dado = dado << 8;
                    dado += serialPort1.ReadByte();
                    dadof = dado / 1000.0;
                    tempobob_textBox.Text = dadof.ToString("F2")
                        ;

                    Application.DoEvents();

                    serialPort1.Write("q");
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    fase_textBox.Text = dado.ToString("F0");

                    Application.DoEvents();

                    serialPort1.Write("r");
                    // espera resposta...
                    dado = serialPort1.ReadByte();
                    erro_ger_sw_textBox.Text = Convert.ToString(
                        dado, 2);

                    Application.DoEvents();

                    serialPort1.Write("s");
                    // espera resposta...
```

```
                dado = serialPort1.ReadByte();
                erro_ger_hw_textBox.Text = dado.ToString("X"
                    );

                Application.DoEvents();

                serialPort1.Write("t");
                // espera resposta...
                dado = serialPort1.ReadByte();
                erro_sinc_sw_textBox.Text = Convert.ToString
                    (dado, 2);

                Application.DoEvents();

                t = stopwatch.ElapsedMilliseconds * 0.001;
                    // Pega leitura do timer

                // escreve todos os dados na string do
                    arquivo
                data_to_file += t.ToString("F2") + "\t\t";
                data_to_file += refrpm_textBox.Text + "\t\t"
                    ;
                data_to_file += rotacao_textBox.Text + "\t\t
                    ";
                data_to_file += refvb_textBox.Text + "\t\t";
                data_to_file += valvula_textBox.Text + "\t\t
                    ";
                data_to_file += map_textBox.Text + "\t\t";
                data_to_file += tempoinjecao_textBox.Text +
                    "\t\t";
                //data_to_file += lambda_textBox.Text + "\t\
                    t";
                data_to_file += "\r\n";

                Application.DoEvents();

                // Atualiza barra de progresso
                if (progressBar1.Value == progressBar1.
                    Maximum)
                    progressBar1.Value = progressBar1.
                        Minimum;
                else
                    progressBar1.Value += 10;

                Application.DoEvents();
            }
            catch (Exception)
```

```
                    {
                        erro++;
                        if (erro > 5)
                        {
                            MessageBox.Show("ãNo áh dados na Serial!
                                ", "Erro de Comunicacao!",
                                MessageBoxButtons.OK, MessageBoxIcon.
                                Warning);
                            medindo = false;
                            lervalores_button.Text = "LER VALORES";
                            progressBar1.Value = progressBar1.
                                Minimum;
                            erro = 0;
                            return;
                        }
                    }
                }
            }
            else // medindo = true, parar aquisicao
            {
                medindo = false;
                lervalores_button.Text = "LER VALORES";
                progressBar1.Value = progressBar1.Minimum;
                erro = 0;
            }
        }
        else
            MessageBox.Show("A porta esta fechada!", "Erro de
                Comunicacao!", MessageBoxButtons.OK, MessageBoxIcon.
                Warning);
    }

    private void controlpedal_checkBox_CheckedChanged(object sender,
        EventArgs e)
    {
        if (serialPort1.IsOpen == true)
        {
            if (controlpedal_checkBox.Checked == true)
                serialPort1.Write("u");
            else
                serialPort1.Write("v");
        }
        else
            MessageBox.Show("A porta esta fechada!", "Erro de
                Comunicacao!", MessageBoxButtons.OK, MessageBoxIcon.
                Warning);
    }
```

```
        private void trackBar1_Scroll(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen == true)
            {
                if (controlpedal_checkBox.Checked == true)
                {
                    serialPort1.Write("w");
                    rxbuffer = BitConverter.GetBytes(pedal_trackBar.
                        Value);
                    serialPort1.Write(rxbuffer, 0, 1);
                }
                else
                    MessageBox.Show("Selecione a çãopo 'Controlar Pedal
                        '!", "Erro!", MessageBoxButtons.OK,
                        MessageBoxIcon.Warning);
            }
            else
                MessageBox.Show("A porta esta fechada!", "Erro de
                    Comunicacao!", MessageBoxButtons.OK, MessageBoxIcon.
                    Warning);
        }

        private void medidas_checkBox_CheckedChanged(object sender,
            EventArgs e)
        {
            if (medidas_checkBox.Checked == true)
            {
                pedal_label.Text += " (V)";
                valvula_label.Text += " (V)";
                map_label.Text += " (kPa)";
                tar_label.Text += " °(C)";
                tagua_label2.Text += " °(C)";
                lambda_label.Text += " (V)";
                bateria_label.Text += " (V)";
                refvb_label.Text += " (V)";
            }
            else
            {
                pedal_label.Text = pedal_label.Text.Replace(" (V)","");
                valvula_label.Text = valvula_label.Text.Replace(" (V)",
                    "");
                map_label.Text = map_label.Text.Replace(" (kPa)", "");
                tar_label.Text = tar_label.Text.Replace(" °(C)", "");
                tagua_label2.Text = tagua_label2.Text.Replace(" °(C)", "
                    ");
                lambda_label.Text = lambda_label.Text.Replace(" (V)", ""
```

```
                    );
                bateria_label.Text = bateria_label.Text.Replace(" (V)",
                    "");
                refvb_label.Text = refvb_label.Text.Replace(" (V)", "");
            }
        }

        private void pedal_comboBox_SelectedIndexChanged(object sender,
            EventArgs e)
        {
            // Seleciona a medidas limites do scroll de acordo com o
                pedal utilizado
            if (pedal_comboBox.SelectedIndex == 0)
            {
                pedal_trackBar.Minimum = PEDAL_MIN_POLO;
                pedal_trackBar.Maximum = PEDAL_MAX_POLO;
            }
            else
            {
                pedal_trackBar.Minimum = PEDAL_MIN_GM;
                pedal_trackBar.Maximum = PEDAL_MAX_GM;
            }
        }

        private void modo_comboBox_SelectedIndexChanged(object sender,
            EventArgs e)
        {
            if (serialPort1.IsOpen == true)
            {
                if (modo_comboBox.SelectedIndex == 0) // Modo de
                    operacao "normal"
                     serialPort1.Write("x");
                else // Modo de operacao "economico"
                     serialPort1.Write("y");
            }
            else
                MessageBox.Show("A porta esta fechada!", "Erro de
                    Comunicacao!", MessageBoxButtons.OK, MessageBoxIcon.
                    Warning);
        }
    }
}
```