CENTRO PAULA SOUZA

FATEC SANTO ANDRÉ

Tecnologia em Eletrônica Automotiva

FELIPE MAROTTI MOCHIUTI

ANÁLISE DE DESEMPENHO ECU FATEC V

SANTO ANDRÉ

2020

FELIPE MAROTTI MOCHIUTI

ANÁLISE DE DESEMPENHO ECU FATEC V

Trabalho de conclusão de curso apresentado ao Curso Tecnologia em Eletrônica Automotiva da FATEC Santo André orientado pelo professor Me. Weslley Medeiros Torres como requisito parcial para a obtenção do título tecnólogo em Eletrônica Automotiva

Santo André

2020

M688a
Mochiuti, Felipe Marotti
Análise de desempenho ECU FATEC V / Felipe Marotti Mochiuti. - Santo André, 2020. – 99f: il.

Trabalho de Conclusão de Curso – FATEC Santo André. Curso de Tecnologia em Eletrônica Automotiva, 2020.

Orientador: Prof. Weslley Medeiros Torres

1. Eletrônica. 2. Veículos. 3. Sincronismo do motor. 4. Injeção eletrônica. 5. Sincronismo ECU. I. Análise de desempenho ECU FATEC V.

629.2

FELIPE MAROTTI MOCHIUTI

ANÁLISE DE DESEMPENHO ECU FATEC V

Trabalho de Conclusão de Curso apresentado a FATEC SANTO ANDRÉ como requisito parcial à obtenção de título de Tecnólogo em Eletrônica Automotiva

BANCA EXAMINADORA

Local: Fatec Santo André
Horário: 08:00
Data: 09/07/2020

Prof. Me. Weslley Medeiros Torres
Presidente da Banca
Fatec Santo André

Prof. Fernando Dalbo Garup
Primeiro membro da Banca
Fatec Santo André

Prof. Dr. Armando Antônio Maria Laganá
Segundo Membro da Banca
Fatec Santo André

SANTO ANDRÉ

2020

DEDICATÓRIA

Dedico esse trabalho acadêmico a minha esposa e companheira Carolina, que me apoiou desde o início e compreendeu a necessidade de minha ausência em dias que precisei me ausentar para me aplicar aos estudos.

AGRADECIMENTOS

São muitas as pessoas que me ajudaram a chegar na presente data com a perseverança e conhecimento para concluir a primeira etapa deste trabalho de graduação, então, citarei apenas as pessoas que tiveram uma contribuição direta na realização deste projeto.

Começo agradecendo aos professores que tive a enorme satisfação de conhecer e poder ter o privilégio de aprender sobre o tema principal do trabalho. São eles, Professor Marco Aurélio Froés, que além de lecionar os conceitos de motores de combustão interna e sanar algumas dúvidas, ajudou também Humberto Manavella em sua obra, uma das principais referências bibliográficas deste trabalho. Professor Armando Antônio Maria Laganá, quem introduziu com maestria os conceitos dos sensores e atuadores eletrônicos utilizados na Injeção Eletrônica (IE). O Professor Adriano Ribolla que com seu conhecimento e didática, torna fácil a compreensão de assuntos complexos no que diz respeito a sincronismo e funcionamento no motor de ciclo Otto como um todo. Professor Orlando de Salvo Junior, que traduz a linguagem da central de controle eletrônico do motor fazendo com que a diagnose desta seja de simples entendimento. Professor Weslley Medeiros Torres foi quem introduziu a arte e a capacidade da programação, que fez com que eu me interessasse e buscasse me aprofundar no assunto, e que não hesitou em me orientar neste projeto.

Agradeço também quem colaborou e prestou seu tempo e atenção para me ajudar seja com material de apoio ou mesmo tirando algumas dúvidas. Paulo Costa do canal Carro & Técnica Treinamentos Automotivos no *Youtube*, que apesar de não ter formação acadêmica, mas 25 anos de experiência na área automotiva, me tirou dúvidas e apontou materiais de consulta técnica. Bruno Martin de Alcântara Dias, formado na Fatec e mestrado na Politécnica, cedeu material de sua dissertação de mestrado para eu compor em meu trabalho acadêmico.

RESUMO

Com a evolução da tecnologia automotiva, no que se refere ao controle do Motor de Combustão Interna (MCI), faz com que seu projeto seja muito complexo, envolve requisitos funcionais e não funcionais, como potência, autonomia, baixa emissão de gases poluentes *etc*. E para atingir esses requisitos, é empregado uma Unidade de Controle Eletrônico (ECU) para controlar o motor, composta por um *hardware* específico e um *software* desenvolvido para cada aplicação. Na FATEC Santo André, já foram desenvolvidos diversos projetos de ECU, e atualmente está na versão V. Então o trabalho busca apresentar um estudo comparativo da implementação do sub sistema de sincronismo da ECU, entre versões de código implementadas em linguagem C e *Assembly.*

Palavras-chave: Eletrônica, Veículos, Sincronismo do Motor, Injeção Eletrônica, Sincronismo ECU.

ABSTRACT

With the evolution of automotive technology with regards to thecontrol of the Internal Combustion Engine, it makes the project very complex involving functional and non-functional requirements such as power, autonomy, low emission of polluting gases etc. And to achieve these requirements an ECU (Electronic Control Unit) is used to control the engine, consisting of a specific hardware and software developed for each application. At FATEC Santo André, several ECU projects have already been developed and it is currently in version. So the work seeks to presente a comparative study of the implementation of the ECU sub-timing system, between code versions implemented in C and Assembly languages.

Keywords: Electronics, Vehicles, Engine Timing, Electronic Fuel Injection, ECU Timing.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

MCI - Motor de Combustão Interna

IE - Injeção Eletrônica

ECU - Unidade de Controle Eletrônico

PMS - Ponto Morto Superior

PMI - Ponto Morto Inferior

NTC - *Negative Temperature Coeficient*

HEX - Hexadecimal

LED - Light-Emitting Diode

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE FÓRMULAS

SUMÁRIO

1. INTRODUÇÃO

O início da eletrônica no gerenciamento dos MCI, se deu com a alta do petróleo, fazendo com que o preço do combustível disparasse, diante dessa situação, a solução encontrada foi otimizar o uso de combustível visando a eficiência no consumo.

Tal solução foi o início do controle eletrônico da injeção de combustível, que, embora fosse um sistema inviável na época, no que se refere ao custo em relação ao sistema de injeção mecânica, devido ao alto custo de desenvolvimento, produção e manutenção, se fez o melhor caminho para otimizar a eficiência na combustão dos motores.

Com exceção do custo, a substituição do carburador pela IE é significativa na melhoria em alguns aspectos, estes são:

- Menor consumo de combustível: devido a dosagem mais precisa em qualquer regime de trabalho e condições ambientais tais como o clima e altitude;
- Drástica redução na emissão de poluentes: com a utilização da sonda *lambda*, é possível corrigir a mistura ar/combustível quase que em tempo real para uma combustão mais eficiente.
- Ganho significativo de potência: através da ignição eletrônica, que substitui os avanços de ignição mecânicos e a vácuo, é possível extrair do motor a melhor performance.
- Confiabilidade: com o uso de componentes eletrônicos que tem uma maior durabilidade, devido a eliminação de desgaste mecânico, e a diagnose presente nos módulos de IE, que fazem o aprendizado do desgaste e envelhecimento de alguns componentes.

Isso traz requisitos funcionais de controle e comunicação na ECU, que devem ser convertidos em componentes de *hardware* e *software*. No desenvolvimento de *software* os requisitos funcionais são convertidos em arquivos executáveis (HEX), já no *hardware* são os circuitos integrados e microcontroladores. Então, o próximo passo é a decisão da linguagem de programação, que, é um fator muito importante visto que, implica diretamente no tempo de desenvolvimento, complexidade, manutenções e

futuras expansões do *software* desenvolvido. Naturalmente a linguagem C se torna a mais viável para sistemas embarcados.

1.1. Motivação

Dentre os trabalhos já desenvolvidos na FATEC Santo André visando o projeto da ECU, todos visaram a implementação do *software* utilizando linguagem de alto nível, Linguagem C.

Porém, é sabido que o uso de linguagens de alto nível para o desenvolvimento de *software* embarcado para o controle de sistemas de tempo crítico é um ponto importante que deve ser levado em consideração, pois durante o processo de geração do arquivo HEX, o compilador deverá traduzir a linguagem de alto nível para mnemônicos do microcontrolador.

O controle de sincronismo é o subsistema mais crítico, pois, realiza o controle da injeção de combustível e avanço do ponto de ignição, o correto controle determinará o quão eficiente será o sistema de controle de motor. A partir desse preceito, o presente trabalho tem como objetivo desenvolver um sistema de sincronismo do motor utilizando linguagem *Assembly*.

1.2. Objetivo

O objeto deste trabalho é desenvolver um estudo comparativo entre a implementação do *software* do sistema de sincronismo da ECU FATEC nas linguagens C e *Assembly*, baseando-se nas implementações dos trabalhos de (PEREIRA B.C.F., 2013.), (MOSCARDINI & MATA, 2014.), (JUNIOR & JATO & HIROKI, 2016.) e (SOCORRO & MERIZIO & LOPES & SILVA.). Dessa forma, podendo mensurar as diferenças de respostas temporais do sistema de sincronismo

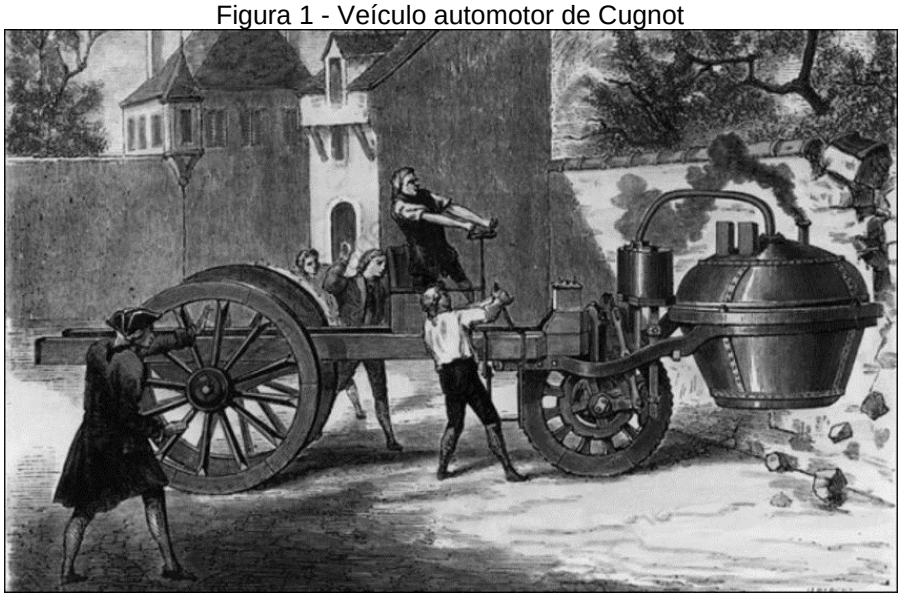
2. REFERENCIAL TEÓRICO

2.1. Breve história do automóvel

Não existe uma data exata referente a criação do primeiro automóvel, na verdade houveram diversos desbravadores que, através de adaptações sucessivas buscavam se deslocar num menor espaço de tempo com o menor consumo de energia possível. (CAPELLI, 2010)

O primeiro registro foi no ano de 1769 no reino unido, quando Nicolas Cugnot criou um veículo a vapor, ilustrado na figura 1. Equipados com caixa de velocidades e direção estes veículos chegaram a ser comercializados fazendo necessária a criação de leis de trânsito, das quais todo veículo automotor deveria transitar atrás de um indivíduo que percorria afrente do veículo sinalizando o tráfego deste. (CAPELLI, 2010 p. 15)

Figura 1 - Veículo automotor de Cugnot



Fonte: Adaptado de https://www.learning-history.com (2019)

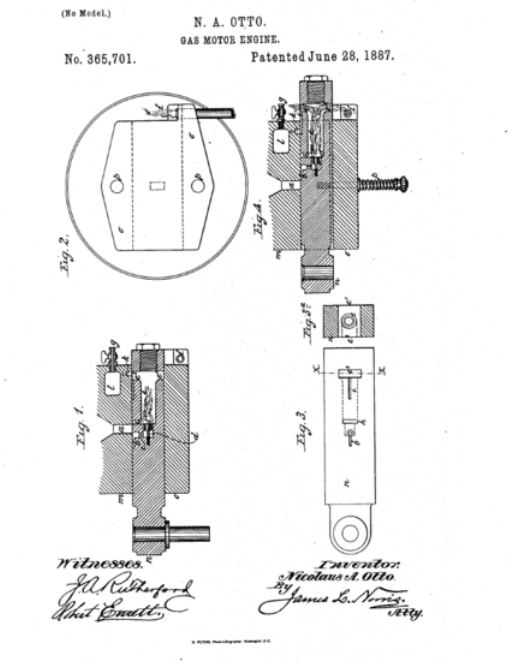
Em 1860 veio o primeiro veículo com um MCI, como mostra a figura 2, idealizado pelo belga Etienne Lenoir era movido a gás de carvão. Este veículo percorreu sete milhas em três horas. (CAPELLI, 2010 p. 15)

Figura 2 - Hippoobile

Fonte: http://www.autoentusiastasclassic.com.br (2019)

Em 1876 Nikolaus Otto otimizou o MCI que operava com quatro tempos, o combustível utilizado era o gás de iluminação. Só no ano de 1886, Karl Benz construiu o primeiro veículo com um MCI a gasolina. Contudo, a patente do MCI de quatro tempos é de Nikolaus Otto, apresentado na figura 3. (CAPELLI, 2010 p. 15)

Figura 3 - Patente motor quatro tempos de Otto



Fonte: Adaptado de (CAPELLI, 2010 p. 16) (2019)

2.2. Combustão interna

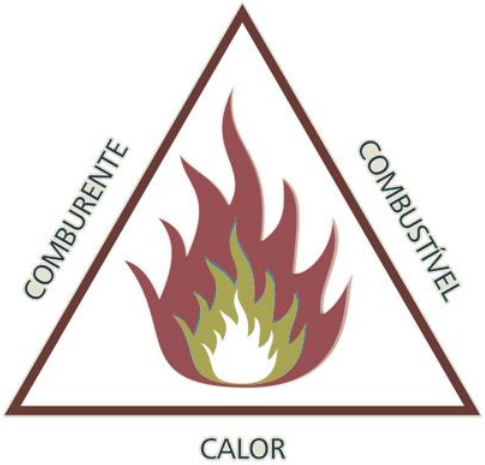
A protagonista por trás do conceito de locomoção dos veículos automotores desde os primórdios até os dias de hoje, a combustão interna tem o papel de converter o combustível em trabalho para então fazer com que o veículo se locomova.

Através da conversão da energia química, contida nos combustíveis utilizados na combustão nos motores, em energia térmica que se dá devido ao aumento da temperatura na câmara de combustão, fazendo com que o gás se expanda empurrando assim o êmbolo do Ponto Morto Superior (PMS) ao Ponto Morto Inferior (PMI). Este deslocamento linear do êmbolo, que está ligado um sistema biela manivela, converte o movimento linear em rotação que é transmitido às rodas através do sistema de transmissão. (CAPELLI, 2010 p. 186)

Para que haja combustão são necessários três componentes como ilustrado da figura 4:

- CALOR: agente de detonação, no caso a centelha da vela;
- COMBURENTE: ar atmosférico, especificamente o oxigênio;
- COMBUSTÍVEL: hidrocarbonetos, gasolina ou álcool.

Figura 4 - Triângulo do fogo



Fonte: Adaptado de https://www.cursodebombeiro.com.br/ (2019)

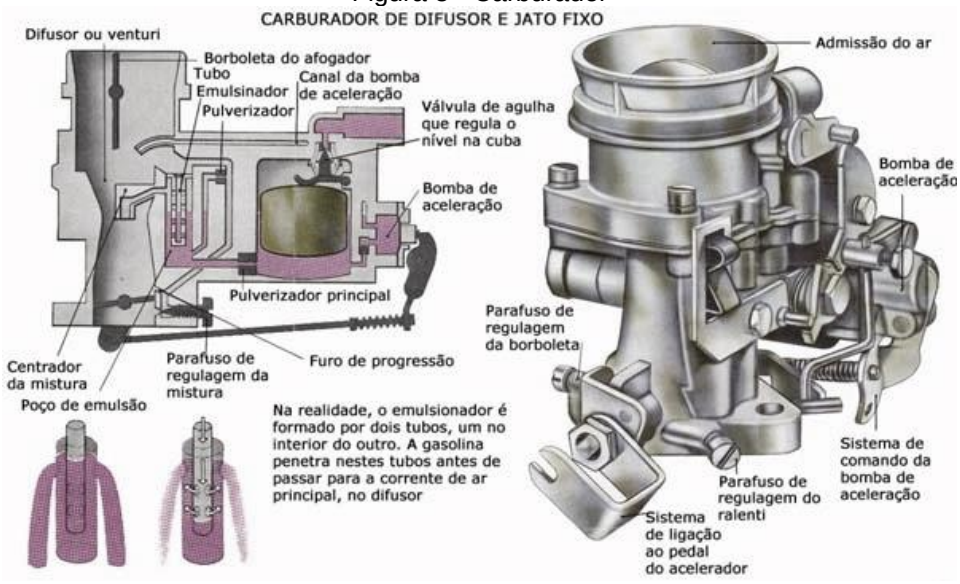
2.3. Alimentação de combustível

Como em toda e qualquer área a evolução tecnológica, no que se refere ao automóvel e seus componentes, tem como referência os primórdios. Na alimentação do motor não seria diferente, visto que a IE, embora hoje não se assemelhe tanto ao sistema de alimentação mecânica devido ao aprimoramento constante da tecnologia, teve seu início basicamente na automação do carburador e os componentes do motor.

2.3.1. Mecânica / Carburador

O carburador, representado na figura 5, nada mais é do que um dosador mecânico, que através de orifícios calibrados de acordo com o volume do motor e o combustível utilizado, regula a quantidade de combustível a ser diluído ao ar admitido através da abertura da válvula borboleta, que conforme abre aumenta o fluxo de ar, aumentado assim a quantidade de combustível arrastado pela depressão gerada pelo êmbolo do motor no curso do PMI ao PMI. (CAPELLI, 2010 p. 241)

Figura 5 - Carburador



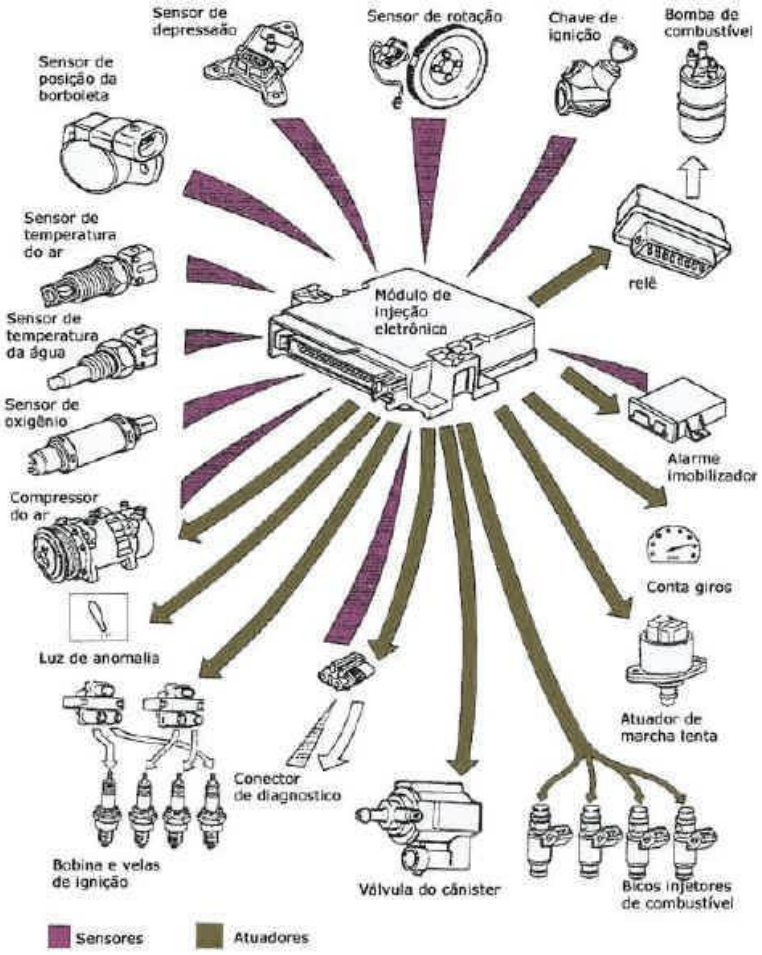
Fonte: https://megaarquivo.wordpress.com (2019)

2.3.2. Eletrônica

Assim como na injeção mecânica, a IE controla quantidade de combustível de acordo com a quantidade de ar admitido e/ou a depressão do coletor de admissão, depressão esta que determina a carga do motor.

Diferentemente do carburador, onde a alimentação é feita através do arrasto do combustível devido ao fluxo de ar, a IE mede, por meio de sensores, o fluxo de ar admitido e dosa uma quantidade de massa de combustível equivalente a essa massa de ar, como mostrado no esquemático da figura 6.

Figura 6 - IE



Fonte: http://blogdispemec.rdstation.com.br (2019)

Da mesma maneira, a depressão no coletor é medida por um sensor de pressão instalado no coletor de admissão. A principal vantagem do uso do sensor de pressão na IE em relação ao sistema mecânico, é o fato de que através do sensor de pressão, a IE ajusta automaticamente a quantidade de combustível de acordo com a altitude que interfere diretamente na proporção da mistura. No carburador, essa alteração teria que ser feita por um profissional manualmente.

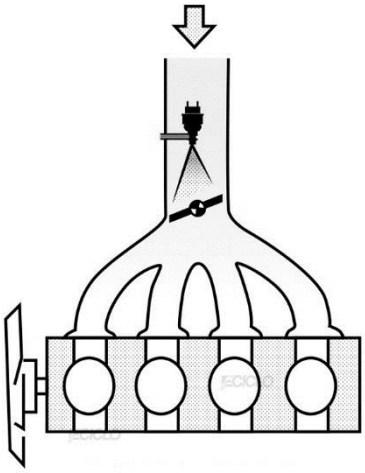
Por último, mas não menos importante, o sensoriamento da rotação do motor. Este dado é o mais importante para o funcionamento do motor, tendo em mente que

sem ele, não seria possível sincronizar o tempo de injeção e nem mesmo o avanço de ignição.

2.3.2.1. Injeção monoponto

Este conceito de injeção é bem parecido com o carburador, tanto na aparência como na posição em que se injeta o combustível. Basicamente o antigo sistema de arrasto foi substituído por um eletro injetor, sendo assim o combustível é pulverizado ao fluxo de ar no mesmo ponto como visto na figura 7. Neste sistema, após o combustível ser misturado ao ar admitido, o que determina o cilindro de destino da mistura ar combustível é a depressão gerada com a abertura da válvula de admissão do respectivo cilindro, que está em fase de admissão. (CAPELLI, 2010 p. 264)

Figura 7 - Injeção Monoponto



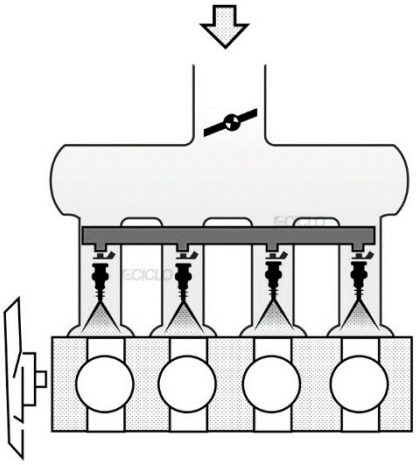
Fonte: Adaptada de http://blog.ciclo.eng.br/ (2019)

2.3.2.2. Injeção multiponto simultânea

Precursor do sistema multiponto, este tipo de IE é bem simples, onde a diferença em relação ao monoponto é a quantidade e disposição das eletroválvulas injetoras de combustível. São instaladas uma eletroválvula de injeção por cilindro, onde cada uma destas está próxima e direcionada para a válvula de admissão do respectivo cilindro. Ao ser acionado, o eletro injetor de combustível pulveriza o combustível na válvula de admissão, evitando assim que o combustível condense nas paredes do coletor de admissão. (BRUNETTI, 2012 p. 468)

De acordo com a figura 8, todas as eletroválvulas injetoras são acionadas simultaneamente, e a cada volta do motor é injetada metade da quantidade ideal de combustível e ao fim de duas voltas cada cilindro terá a quantidade exata de combustível no duto de entrada da admissão. Um ponto negativo é que a maior parte das injeções são feitas com a válvula de admissão fechada fazendo com que parte da mistura condense na válvula e também no duto de admissão do cabeçote. (BRUNETTI, 2012 p. 468)

Figura 8 - Injeção Multiponto Simultânea

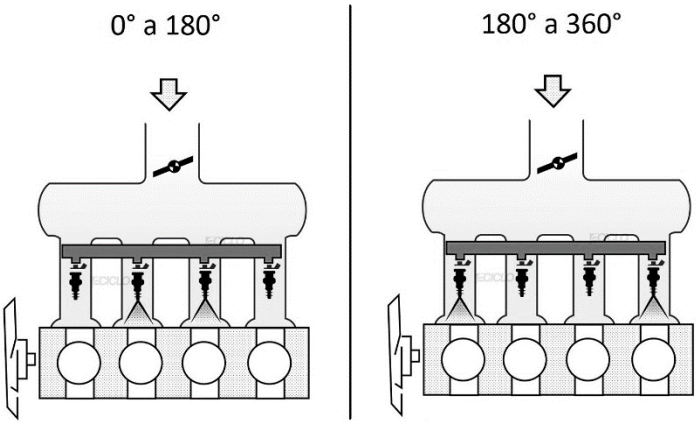


Fonte: Adaptado de *http://blog.ciclo.eng.br (2019)*

2.3.2.3. Injeção multiponto semissequencial

O sistema semissequencial, também conhecido como banco a banco, se assemelha ao sistema de IE simultâneo, com duas injeções de combustível para formar a mistura completa. Contudo, este sistema faz uma discretização nas injeções acionando pares de injetores, os cilindros gêmeos 1-4 e 2-3, sequência está ilustrada na figura 9.

Figura 9 - Injeção Multiponto Semissequencial



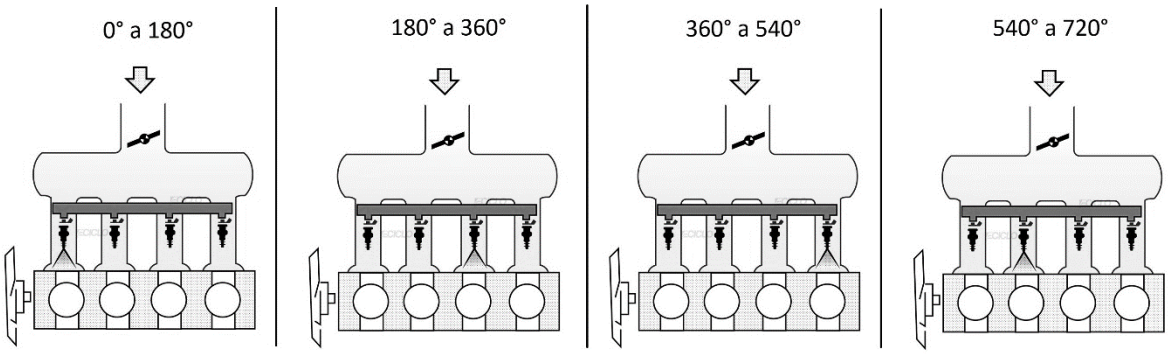
Fonte: Adaptado de *http://blog.ciclo.eng.br (2019)*

As eletroválvulas injetoras sempre são acionadas nas fases de expansão e admissão, otimizando assim a injeção que metade das vezes é feita com a válvula de admissão aberta reduzindo a condensação do combustível nas paredes dos dutos de admissão do cabeçote. É importante ressaltar que para este tipo de estratégia de injeção é necessário que a central eletrônica conheça o PMI de todos os cilindros. (GLEHN, 2001 p. 85)

2.3.2.4. Injeção multiponto sequencial

Devido ao avanço tecnológico e principalmente as exigências ambientais para um índice de emissões cada vez menor, a injeção de combustível evolui para uma estratégia sequencial onde as válvulas injetoras passaram a ser acionadas independentemente umas das outras, de acordo com a figura 10, aumentando assim a eficiência, o desempenho e o consumo. Nesta estratégia os injetores trabalham somente quando a válvula de admissão está aberta, evitando assim a condensação no duto de admissão do cabeçote.

Figura 10 - Injeção Multiponto Sequencial



Fonte: Adaptado de *http://blog.ciclo.eng.br (2019)*

Para garantir um perfeito sincronismo na injeção de combustível e também no avanço da ignição é necessário que a central eletrônica conheça a fase em que os quatro cilindros se encontram, para isso se faz o uso de um sensor de fase instalado no comando de válvulas, que aliado ao sinal de rotação permite determinar cada fase dos quatro cilindros.

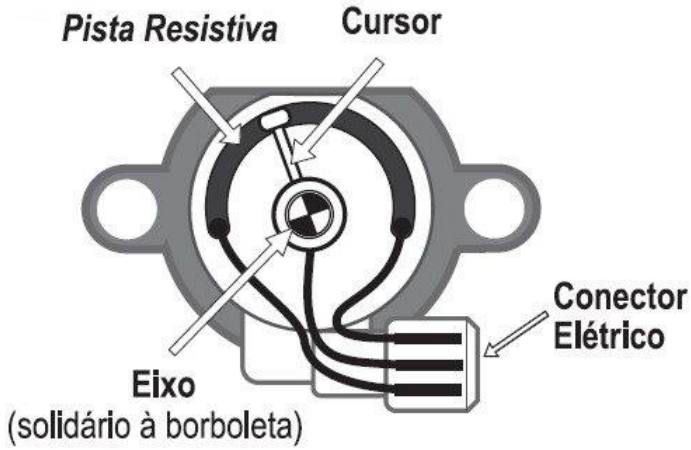
2.3.3. Sensores

Para que a central eletrônica realize uma dosagem perfeita de ar combustível, por exemplo, é necessário que esta tenha informações da massa de ar admitida para definir assim a massa de combustível necessária. Também são necessárias mais informações para definir a relação ar combustível para cada regime de trabalho, como rotação do motor, pressão no coletor de admissão, temperatura do motor, entre outros. Para cada uma dessas informações existe um sensor responsável por converter as grandezas físicas em sinais eletrônicos que serão disponibilizados para a central eletrônica, que serão utilizados para interpretar a condição de operação e então, controlar os atuadores para um perfeito sincronismo.

2.3.3.1. Sensor de posição da válvula borboleta

Nos sistemas de IE a válvula de aceleração, também conhecida como válvula borboleta, se encontra o corpo de borboleta. O eixo da válvula de aceleração é acoplado o sensor de posição resistivo da válvula borboleta, cujo funcionamento é similar a um potenciômetro, cujo cursor é solidário ao eixo da válvula borboleta que ao abrir ou fechar varia a sua resistência, conforme ilustrado na figura 11. Esta informação de abertura da válvula de borboleta é importante para determinar o avanço de ignição e controlar a relação ar combustível. (MANAVELLA, 2003)

Figura 11 - Sensor de Posição da Válvula Borboleta

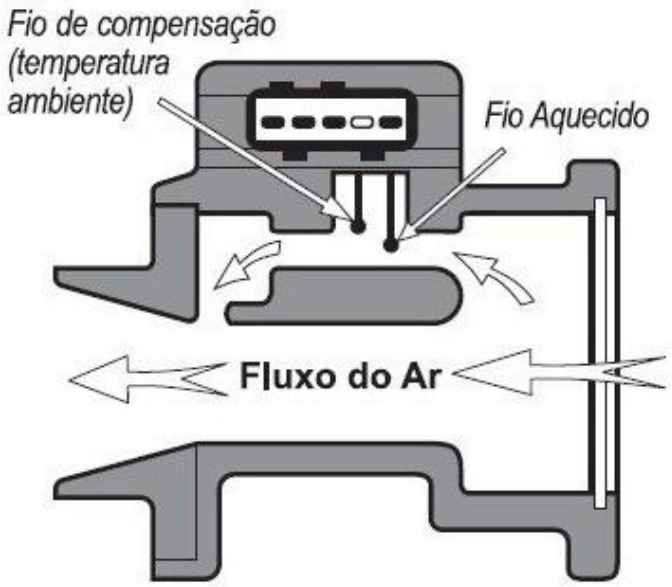


Fonte: Adaptado de https://cursosonline.mte-thomson.com.br (2019)

2.3.3.2. Sensor de massa de ar

Também conhecido como debímetro, este sensor é uma resistência que aquece uma membrana a uma determinada temperatura. O fluxo de ar troca calor ao passar por entre a membrana, que para manter a temperatura específica de trabalho demanda uma maior corrente na resistência de aquecimento, como se vê na figura 12. Esta corrente é medida por uma ponte de *Wheatstone* e é proporcional a massa de ar que flui no sensor em questão. O sensor mede a massa de ar e não o volume, eliminando os problemas de temperatura do ar, altitude e a pressão atmosférica. (CAPELLI, 2010 p. 281)

Figura 12 - Sensor de Massa de Ar

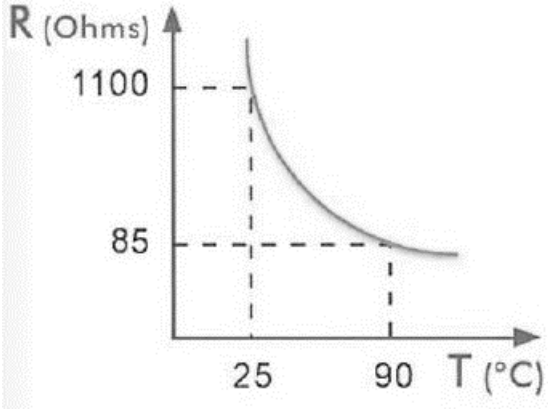


Fonte: Adaptado de https://cursosonline.mte-thomson.com.br (2019)

2.3.3.3. Sensor de temperatura de ar

Este sensor é constituído de um termistor com coeficiente de temperatura negativo ou *Negative Temperature Coeficient* (NTC), representado no gráfico 1, e é montado no coletor de admissão ou no corpo de borboleta.

Gráfico 1 - Curva Característica NTC



Fonte: Adaptado de https://cursosonline.mte-thomson.com.br (2019)

Este termistor possui alta resistência em baixas temperaturas e à medida que a temperatura aumenta a resistência diminui. Com a informação da temperatura do ar admitido e a da pressão no coletor de admissão a central eletrônica calcula a densidade a massa do ar admitido. Na figura 13, é possível verificar os diferentes tipos de sensores de temperaturas com distintos meios de instalação. (CAPELLI, 2010 p. 282)

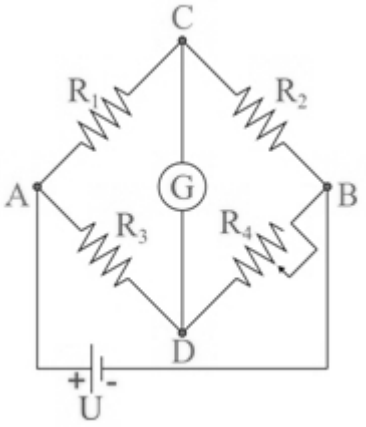
Figura 13 - Sensores de Temperatura de Ar

Fonte: Adaptado de Catálogo MTE-Thompson (2018)

2.3.3.4. Sensor de pressão absoluta

Composto por um diafragma de cerâmica com propriedades piezo resistivas, que de acordo com a deformação da membrana apresenta uma variação de resistência. A resistência é ligada a uma ponte de *Wheatstone*, representada pelo circuito da figura 14. (MANAVELLA, 2003)

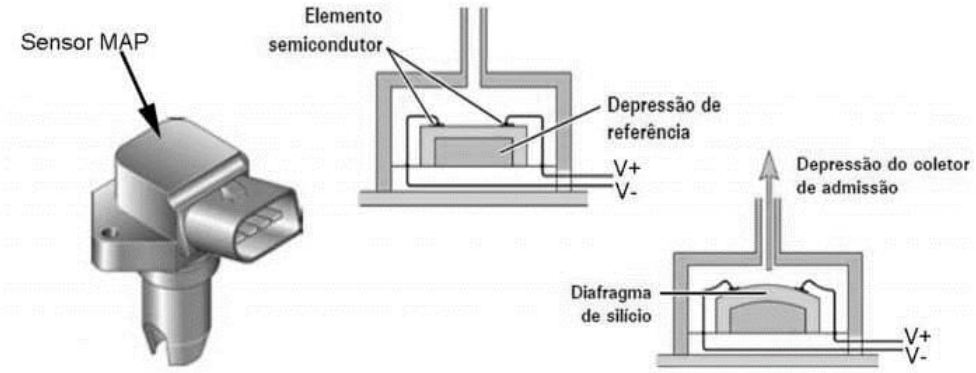
Figura 14 - Ponte de Wheatstone



Fonte: https://www.infoescola.com (2019)

Assim que o condutor liga chave de ignição, o sensor mede a pressão atmosférica do local em que o veículo se encontra, pressão essa que varia de acordo com a altitude. Já com o motor em funcionamento, o sensor mede a depressão do coletor de admissão, como mostra a ilustração da figura 15, determinando assim a carga do motor. (MANAVELLA, 2003)

Figura 15 - Sensor de Pressão Absoluta

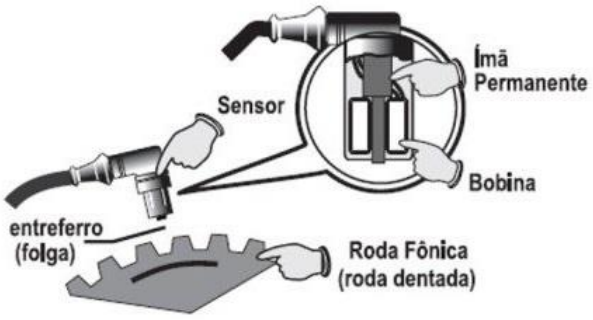


Fonte: Adaptado de Alcântara Dias, 2015

2.3.3.5. Sensor de rotação

Através do princípio de relutância magnética, este sensor é composto por um núcleo polar, imã permanente, enrolado por um fio condutor isolado. O campo magnético do imã permeia tanto o enrolamento ao redor do imã quanto a roda dentada metálica instalada no virabrequim. Quando o campo magnético do sensor está próximo a um dente da roda dentada o fluxo magnético é máximo. Já ao se aproximar da ausência do dente, o fluxo magnético é mínimo. Na figura 16 observa-se a instalação do sensor próximo a roda fônica (GLEHN, 2001 p. 25)

Figura 16 – Sensor de Rotação



Fonte: Adaptado de https://cursosonline.mte-thomson.com.br (2019)

A variação de campo magnético causada pela rotação da roda dentada, ora no dente ora na ausência do dente, provoca uma força eletromotriz no enrolamento do sensor. Quanto maior a rotação maior a tensão induzida, desta forma, é de suma importância o ajuste ideal da distância entre o sensor e o dente da roda dentada. (GLEHN, 2001 p. 25)

2.3.3.6. Sensor de fase

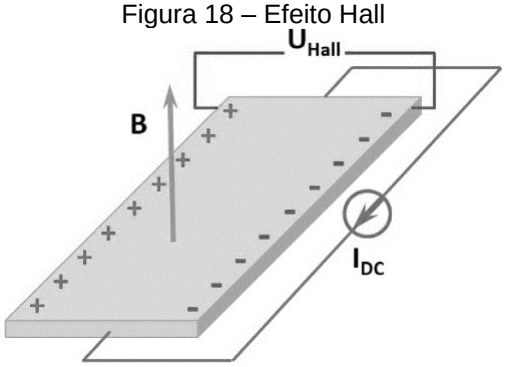
Este sensor tem a função de identificar qual fase cada cilindro se encontra, como ilustra afigura 17, permitindo assim que a central eletrônica tenha o exato momento de injeção e ignição nos cilindros, tornando assim possível tanto a injeção quanto a ignição independente e sequencial.

Figura 17 – Sensor de Fase

Fonte: Adaptado de https://product.tdk.com (2019)

Através do princípio Hall de funcionamento, composto por uma placa semicondutora por onde passa uma corrente e atravessada por um campo magnético perpendicular à respectiva corrente. Devido ao campo magnético, é gerado uma diferença de potencial nas extremidades da placa chamada tensão Hall, ilustrada na figura 18.

Figura 18 – Efeito Hall



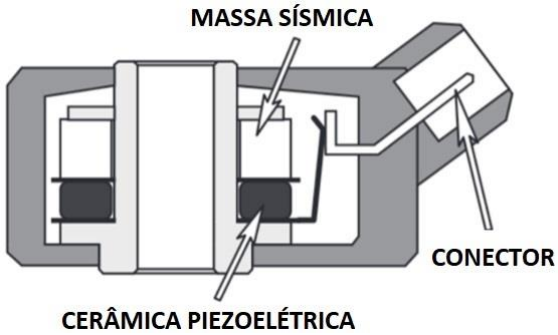
Fonte: Adaptado de https://www.osapublishing.org (2019)

Esta tensão varia com a aproximação de um material ferromagnético junto ao sensor. A frequência do sinal gerado no sensor aumenta de acordo com o aumento da rotação do motor. (CAPELLI, 2010 p. 280)

2.3.3.7. Sensor de detonação

O sensor de detonação tem por função identificar se houve o fenômeno da detonação, que nada mais é que uma combustão indesejada da mistura causada pelo aumento da pressão no cilindro e ocorre antes mesmo da ignição. O principal fator deste fenômeno é o combustível adulterado e ou de má qualidade. Instalado na parede externa do bloco do motor próximo aos cilindros para uma melhor leitura das vibrações provindas da detonação. O material piezoeléctrico composto no sensor visto na figura 19, gera uma tensão proporcional a deformação mecânica sofrida por este. (GLEHN, 2001 p. 56)

Figura 19 – Sensor de Detonação



Fonte: Adaptado de http://www.micro-tronik.com (2019)

Com o sinal gerado pelo sensor a central eletrônica é capaz de identificar a detonação, com frequências da ordem de 5Khz a 15Khz, e então atrasar o avanço do ponto de ignição, renunciando à ignição no momento exato para ter o controle da combustão. Para garantir que não haja uma leitura de ruídos provenientes de outras vibrações no motor, a aquisição do sinal do sensor é feita em janelas de tempos determinadas. Sem o auxílio do sensor de detonação, além da perda de rendimento ocorre também dano nos componentes do motor. (GLEHN, 2001 p. 56)

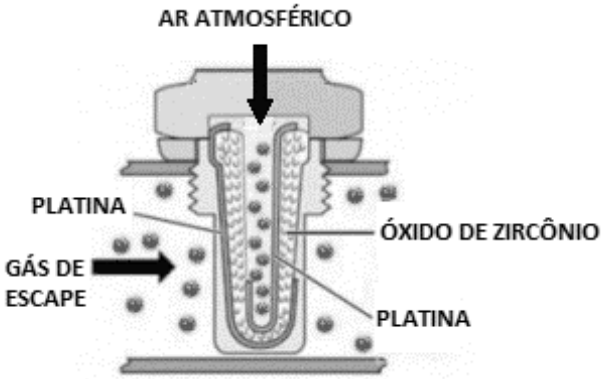
2.3.3.8. Sensor de oxigênio

O sensor de oxigênio está localizado no coletor de escape ou no escapamento e indica a estequiometria imediata da combustão, seja ela rica, estequiométrica ou pobre.

O funcionamento é similar ao de uma pilha que gera uma corrente através da troca de elétrons de um material para outro. Neste caso o que determina a corrente elétrica é a concentração de oxigênio nos gases de escape. Formado por um substrato

cerâmico de zircônio, um eletrólito sólido em temperaturas acima de 300°C, e coberta por dois eletrodos porosos de platina. O eletrodo externo do sensor está em contato com os gases do escape e o eletrodo interno com o ar atmosférico usado como referência, como mostra a figura 20. (MANAVELLA, 2003)

Figura 20 - Sensor de Oxigênio



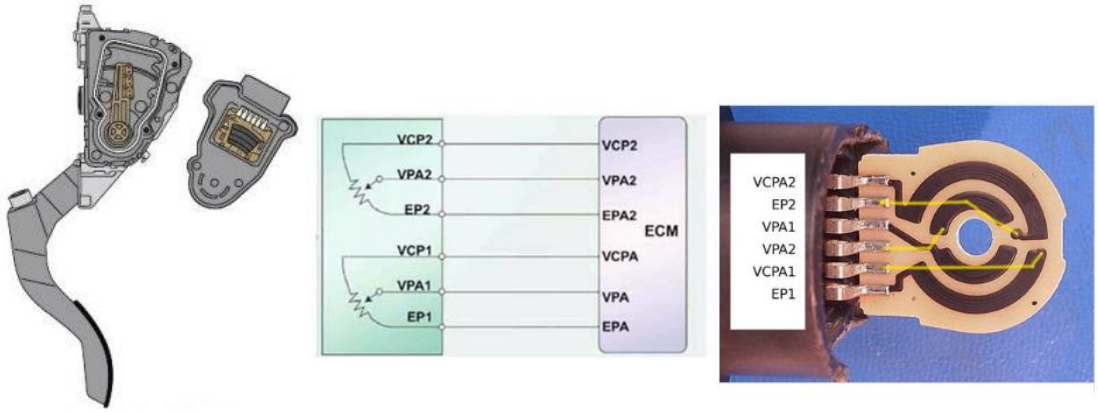
Fonte: Adaptado de Alcântara Dias, 2015

Quando o zircônio está acima de 300°C, quanto maior a diferença de concentração de oxigênio entre os eletrodos do sensor, maior a corrente de elétrons e consequentemente maior a tensão gerada. Como a concentração do eletrodo de referência é consideravelmente maior, o eletrodo exposto ao gás de escape tem um potencial elétrico positivo. Quando a mistura é rica a tensão gerada é de aproximadamente 800mV, já em uma mistura pobre a tensão é de aproximadamente 100 mV. (MANAVELLA, 2003)

2.3.3.9. Acelerador eletrônico

O pedal do acelerador possui dois potenciômetros. Na figura 21 é possível verificar sua vista explodida, sendo que a tensão de saída do segundo é sempre metade da tensão de saída do primeiro potenciômetro, essa redundância aumenta a confiabilidade e precisão do sistema.

Figura 21 – a) Pedal Acelerador b) Detalhe construtivo do potenciômetro



Fonte: a) https://www.troublecodes.net (2019 b) *Technical Support to the National Highway Traffic Safety Administration* (2020)

Com o sinal do pedal eletrônico e mais informações obtidas pelos demais sensores instalados no motor a central eletrônica controla o servo motor de abertura da válvula borboleta para atender à solicitação do motorista.

2.3.4. Atuadores

Nos sistemas de IE foram adotados atuadores com a finalidade de obter um controle mais preciso do sincronismo do motor em seus diversos modos e condições de operação.

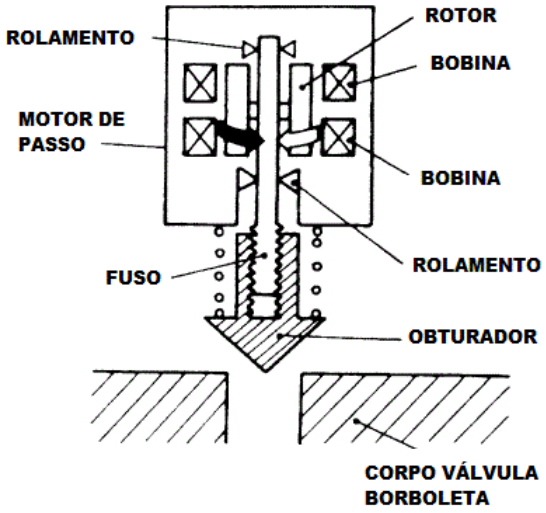
2.3.4.1. Atuador de marcha lenta

A fim de garantir o funcionamento do motor em perfeito sincronismo em marcha lenta, utiliza-se motores elétricos com a finalidade de controlar o fluxo de ar admitido.

2.3.4.1.1. Motor de passo

O motor de passo é composto por duas bobinas, onde dois terminais controlam a abertura e outros dois o fechamento da passagem do fluxo de ar. Esse motor é de corrente contínua em que cada pulso corresponde a um passo equivalente a um ângulo específico do eixo, que ao girar aumenta ou diminui a passagem do fluxo de ar como ilustrado na figura 22. (MANAVELLA, 2003)

Figura 22 - Motor de Passo



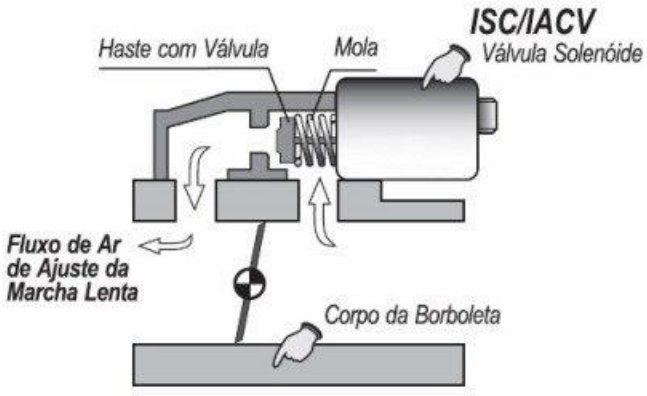
Fonte: Adaptado de https://www.autozone.com (2019)

A central eletrônica envia pulsos ao atuador de marcha lenta de acordo com a necessidade, para manter o motor estável em rotação de marcha lenta. (MANAVELLA, 2003)

2.3.4.1.2. Válvula solenoide

Montada no corpo de borboleta, está válvula controla o fluxo de ar através da modulação por frequência do sinal de acionamento da bobina da mesma, que varia o posicionamento de abertura do embolo de 0% a 100% de acordo com a necessidade, como mostra a figura 23. (MANAVELLA, 2003)

Figura 23 - Válvula Solenoide

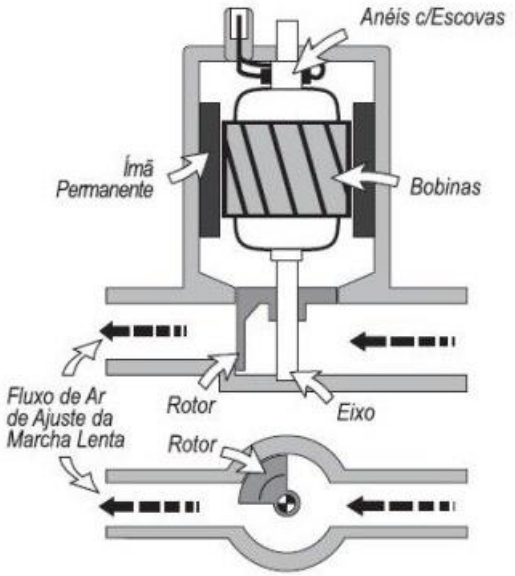


Fonte: Adaptado de https://cursosonline.mte-thomson.com.br (2019)

2.3.4.1.3. Válvula rotativa

O atuador rotativo nada mais é do que um motor de corrente contínua solidário a uma válvula (quarto de círculo) e constituído por duas bobinas e um imã permanente, de acordo com a figura 24. (MANAVELLA, 2003)

Figura 24 - Válvula Rotativa



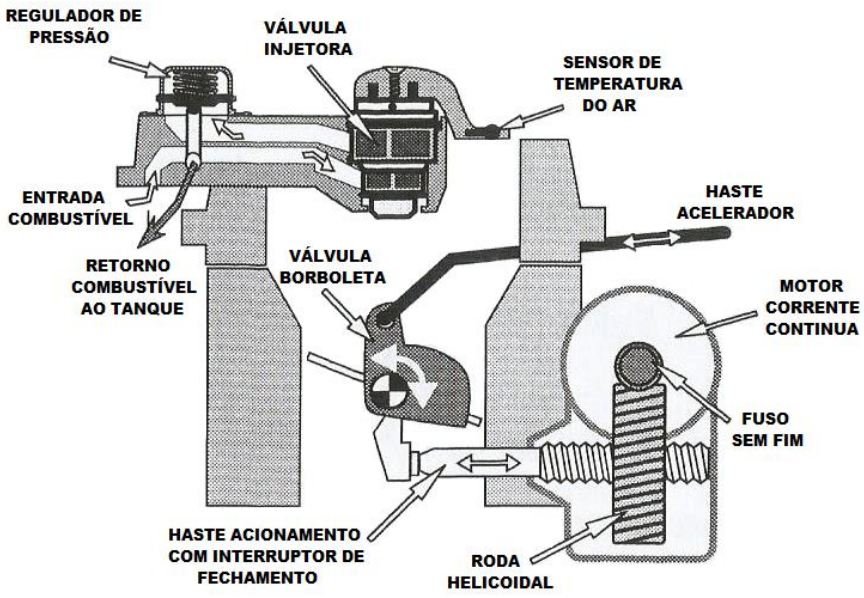
Fonte: Adaptado de https://cursosonline.mte-thomson.com.br (2019)

Quando energizada, as bobinas tendem a se alinhar com os imãs permanentes do estator fazendo com que o eixo gire em torno de seu eixo que é delimitado a um ângulo de 90°. O controle da posição de abertura da válvula se dá pela modulação de pulsos alternada entre as bobinas, que devido as forças opostas dos campos eletromagnéticos que se anulam, permitem o controle exato de abertura. Existem válvulas rotativas que utilizam apenas uma bobina e o retorno da válvula é feito através de uma mola. (MANAVELLA, 2003)

2.3.4.1.4. Motor de corrente contínua

Neste sistema o motor de corrente contínua tem seu eixo acoplado a um fuso sem fim que ao avançar empurra uma haste contra a válvula borboleta, como ilustra a figura 25, efetuando assim a abertura e controlando o fluxo de ar. (MANAVELLA, 2003)

Figura 25 - Atuador de Motor de Corrente Contínua



Fonte: Adaptado de (MANAVELLA, 2003)

A central eletrônica controla a abertura da válvula borboleta de 0° a 22° e identifica o total fechamento dela através de um interruptor. O controle pode ser feito tanto com acionamento contínuo quanto com pulsos, evitando assim o superaquecimento do motor. (MANAVELLA, 2003)

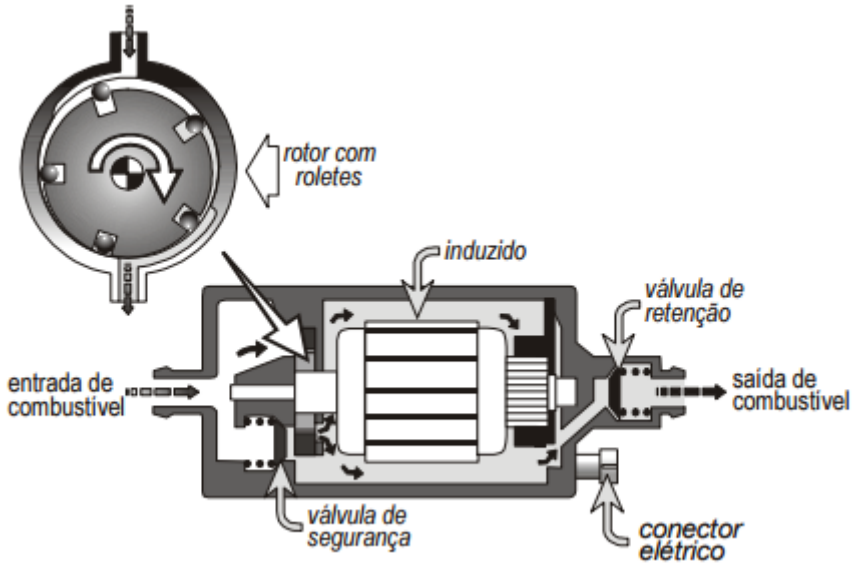
2.3.4.2. Bomba de combustível

Para que o combustível chegue até as válvulas injetoras com pressão ideal de trabalho, a bomba de combustível eletrônica é controlada pela central eletrônica e pode ser externa ou interna ao reservatório de combustível.

2.3.4.2.1. Bomba de roletes

Nesta configuração a bomba é externa ao reservatório de combustível e, o que impulsiona combustível são os roletes, que ao ser movido pelo induzido comprime o combustível contra a parede da bomba, como mostra a figura 26, fazendo com que esse atinja determinada pressão criando um fluxo que ao passar pelo induzido tem como função refrigerá-lo seguindo em direção ao filtro de combustível. (MANAVELLA, 2003)

Figura 26 - Bomba de Combustível de Roletes



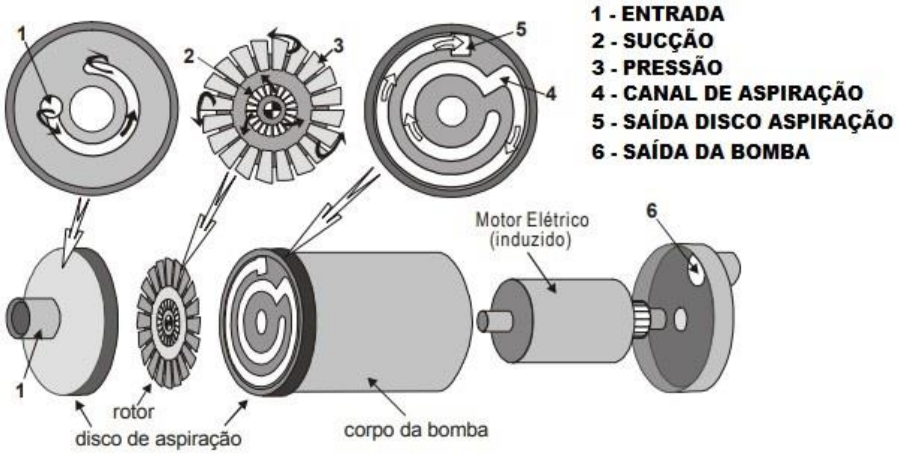
Fonte: Adaptado de http://www.hmautotron.eng.br (2019)

Esta bomba possui uma válvula de retenção unidirecional que faz com que o fluxo de combustível não retorne ao tanque e mantenha uma determinada pressão no circuito. Possui ainda uma válvula de proteção contra alta pressão denominada válvula reguladora de pressão. (MANAVELLA, 2003)

2.3.4.2.2. Bomba centrifuga

Esta bomba é interna ao reservatório de combustível e comumente possui dois estágios, sendo um de sucção e outro de pressurização. O estágio de sucção fica ao centro do rotor, que possui palhetas com perfil lateral responsáveis pela sucção. Na parte externa do rotor existem palhetas com perfil periférico que pressurizam o combustível. A figura 27 ilustra uma vista explodida da bomba com suas respectivas funções. Assim como na bomba de combustível de roletes, a bomba centrífuga faz uso do próprio combustível para arrefecer o induzido do motor e ainda possui uma válvula de retenção para evitar o retorno do combustível. (MANAVELLA, 2003)

Figura 27 - Bomba de Combustível Centrífuga



Fonte: Adaptado de http://www.hmautotron.eng.br (2019)

2.3.4.3. Válvulas injetoras de combustível

As válvulas injetoras são componentes responsáveis pela pulverização do combustível no fluxo de ar admitido. Constituem basicamente de uma válvula solenoide cujo embolo possui uma agulha, que faz com que o injetor permaneça fechado, com retorno através de uma mola. Ao ser acionada, a válvula injetor tem a bobina magnetizada fazendo com que a força do campo eletromagnético vença a força da mola e abra a agulha em aproximadamente 0,1mm. Através dessa abertura da agulha, o combustível que vem pressurizado da galeria passa por orifícios calibrados formando um *spray* em forma de cone de 30° de abertura que pode variar de acordo com a necessidade do fabricante. Os injetores possuem arruelas de borracha que têm tanto a função de amortecimento contra as altas vibrações que são

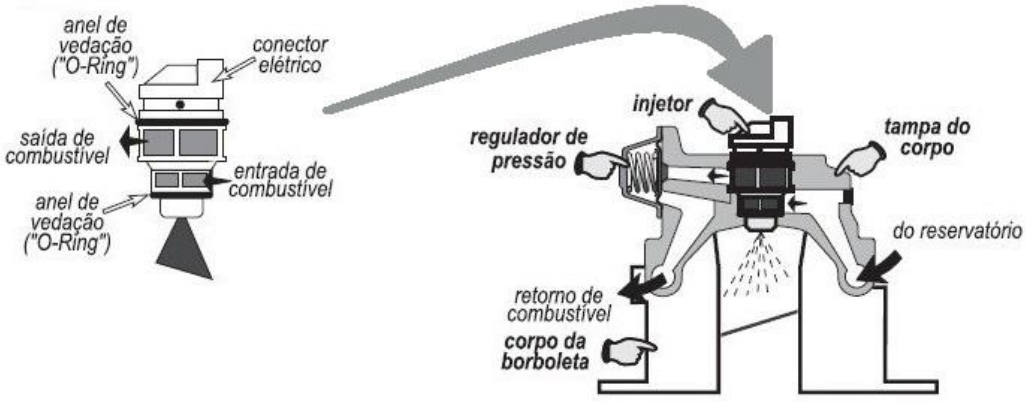
geradas no funcionamento do motor, quanto o isolamento térmico contra as altas temperaturas que podem provocar bolhas de vapor no combustível dificultando a partida a quente. (MANAVELLA, 2003)

Os injetores pode ser classificados como baixa resistência, com valores de 1,5 ohms a 3,5 ohms onde o tempo de carregamento é mais rápido reduzindo o tempo de abertura, e alta resistência, com valores de 12Ω a 20Ω . (MANAVELLA, 2003)

2.3.4.3.1. Injetores *bottom feed*

Normalmente usadas nos sistemas monoponto, este tipo de válvula injetora tem sua alimentação feita através de janelas que ficam localizadas nas laterais inferiores do injetor, como ilustrado na figura 28. (MANAVELLA, 2003)

Figura 28 - Injetor *Bottom Feed*



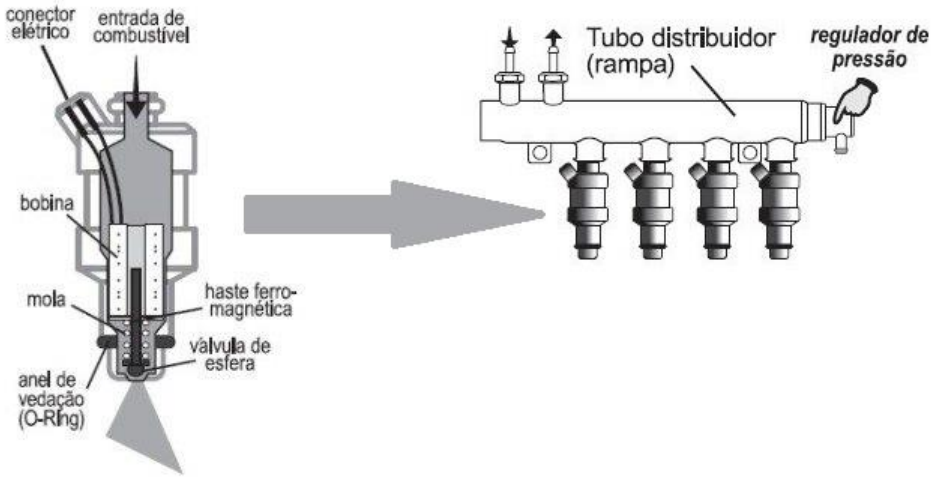
Fonte: Adaptado de https://cursosonline.mte-thomson.com.br (2019)

O fluxo de combustível sai por janelas nas laterais superiores do injetor até chegar a uma válvula reguladora de pressão. (MANAVELLA, 2003)

2.3.4.3.2. Injetores alimentação superior

O mais convencional a ser usado, esta válvula injetora é utilizada nos veículos multiponto em geral, onde os injetores são instalados entre o coletor de admissão e o tudo de distribuidor onde também fica instalada a válvula reguladora de pressão de combustível, como mostra a figura 29. (MANAVELLA, 2003)

Figura 29 - Injetor Alimentação Superior

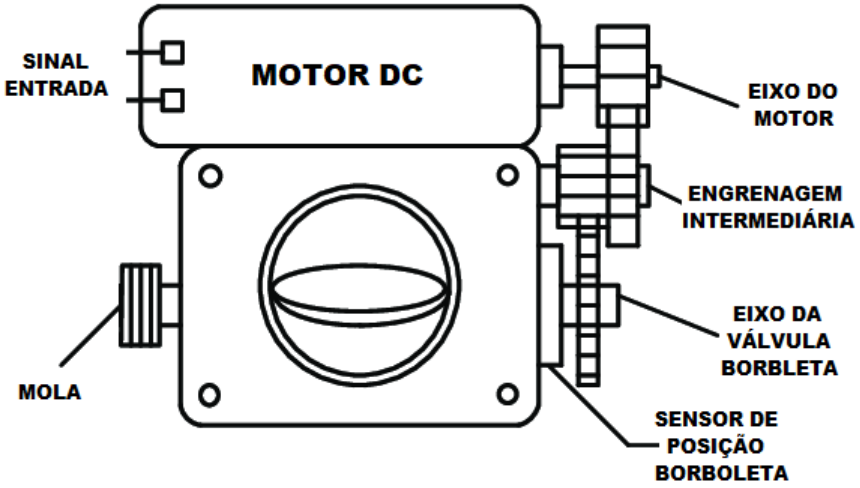


Fonte: Adaptado de https://cursosonline.mte-thomson.com.br (2019)

2.3.4.4. Válvula borboleta eletrônica

Esta válvula eletrônica que junto ao pedal de acelerador eletrônico substitui completamente o sistema de acionamento por cabos. Sua função é regular o fluxo de ar admitido pelo motor através de um motor elétrico, que tem seu eixo acoplado a válvula borboleta e o retorno feito por uma mola, como visto na figura 30, que é controlado por um sinal modulado. (DELPHI, 2003)

Figura 30 - Válvula Borboleta Eletrônica



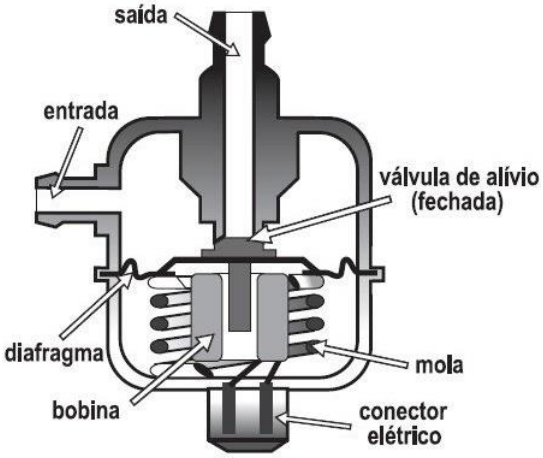
Fonte: Adaptada de https://www.researchgate.net (2019)

De acordo com a posição que se encontra o pedal do acelerador e outros parâmetros, a central eletrônica faz o controle necessário para abrir ou fechar a válvula. A válvula eletrônica elimina o uso de atuadores de marcha lenta e sistemas auxiliares de aceleração, já que esse controle agora é feito por meio da válvula de borboleta eletrônica. A eficiência da válvula borboleta eletrônica se dá devido ao fato de os motores trabalharem em regimes transitórios de trabalho, já na válvula borboleta mecânica não é possível fazer o controle de abertura e fechamento em transições rápidas como nas válvulas eletrônicas. (DELPHI, 2003)

2.3.4.5. Válvula canister

Para evitar que os gases provindos da evaporação do combustível do reservatório entre em contato com a atmosfera, se deu o sistema de controle evaporativo de combustível. O principal responsável por esse controle, além do filtro de carvão ativado que armazena os gases, é a válvula de purga do *canister*, uma válvula solenoide que controla o fluxo dos gases, ilustrada na figura 31, vindos do carvão ativado em direção ao coletor de admissão, devido a depressão gerada pelo embolo. (MANAVELLA, 2003)

Figura 31 - Válvula Canister



Fonte: Adaptado de https://cursosonline.mte-thomson.com.br (2019)

O acionamento da válvula *canister* pode ser feito por sinal contínuo ou através de sinal pulsado que regulam a abertura de 0% a 100% controlando o fluxo dos gases de acordo com o regime do motor. (MANAVELLA, 2003)

2.3.4.6. Bobina de ignição

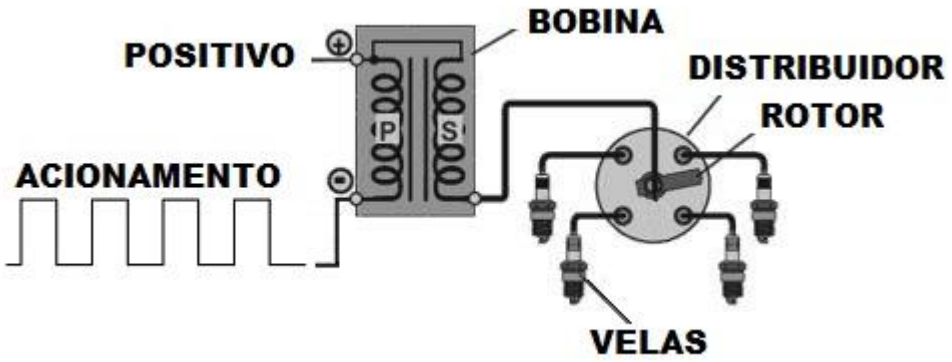
A bobina de ignição transforma a baixa tensão da bateria em alta tensão em um determinado instante por um determinado tempo, a fim de romper o dielétrico existente entre a abertura dos eletrodos da vela de ignição, gerando uma centelha que é imprescindível para a combustão do motor de ciclo Otto. A conversão de baixa para alta tensão é feita pelo acionamento do enrolamento primário da bobina por um tempo específico, que varia de acordo com as características da aplicação. Ao fim da carga da bobina, com o desligamento do terminal do primário da bobina, há um colapso do campo magnético que transfere a energia armazenada do enrolamento primário para o secundário que multiplica a tensão proporcionalmente a quantidade de espiras do

enrolamento secundário. A tensão se eleva até ser suficiente para romper o dielétrico dos eletrodos da vela de ignição. (DELPHI, 2005)

2.3.4.6.1. Chaveamento de alta tensão

Uma única bobina é responsável pela centelha em todos os cilindros do motor, realizando a distribuição através do chaveamento de alta tensão que é realizado pelo rotor, que fecha o contato com o plugue da tampa do distribuidor que corresponde a um determinado cilindro de destino da centelha, como na ilustração da figura 32. O acionamento da bobina pode ser feito por meio de platinado ou transistores. (DELPHI, 2005)

Figura 32 - Chaveamento de Alta Tensão

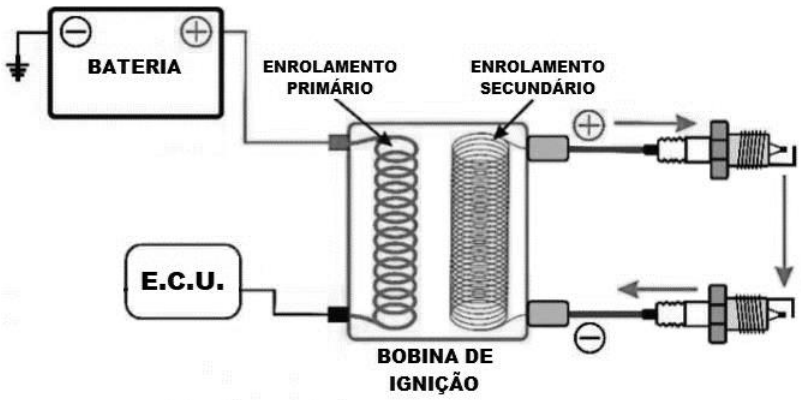


Fonte: Adaptado de https://cursosonline.mte-thomson.com.br (2019)

2.3.4.6.2. Centelha perdida

Neste sistema cada extremidade do enrolamento secundário da bobina é ligado a uma vela de ignição do cilindro gêmeo, assim, quando a bobina transforma a baixa tensão em alta existe a quebra do dielétrico em dois cilindros simultaneamente, um está em fase de compressão e o outro em fase de escape, como se vê na ilustração da figura 33, originando o nome centelha perdida já que na fase de escape não haverá nenhum ganho de energia no motor. O dielétrico do cilindro em fase de escape é bem menor do que o cilindro que está na fase de compressão. A vantagem de se usar o recurso de centelha perdida é o fato de que a central não precisa saber qual cilindro ocorrerá a combustão, apenas carrega a bobina correspondente ao cilindro gêmeo. (DELPHI, 2005)

Figura 33 - Centelha Perdida

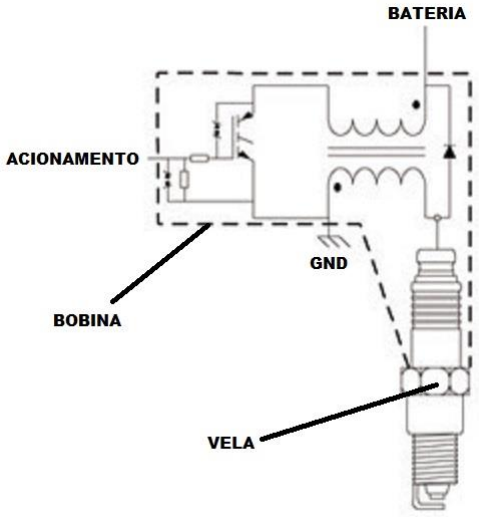


Fonte: Adaptado de https://www.hondashadow.net (2019)

2.3.4.6.3. Bobina de faísca única

Este sistema equipa uma bobina por cilindro ligada diretamente sobre a vela, ilustrado na figura 34, possibilitando que o motor trabalhe em altas rotações já que cada bobina tem um período de 720° da árvore de manivelas entre cada acionamento. Esta bobina consegue fornecer até três vezes mais energia garantindo uma combustão mais eficiente com o mesmo consumo de energia. (DELPHI, 2005)

Figura 34 - Bobina de Faísca Única



Fonte: Adaptado de http://ttec-4848sensorsbytung.blogspot.com (2019)

2.3.4.7. Linguagens de programação

Dado a necessidade de integração do *hardware* (conjunto de sensores e atuadores) com o *software*, o *software* sendo responsável por orquestrar o funcionamento do MCI. Dado o conjunto de tarefas que devem ser desempenhadas, o nível de complexidade do programa está diretamente ligado a arquitetura do computador a ser programado e a tarefa a ser executado pelo programa. (NAHAS, et al., 2012)

Para programar qualquer sistema é necessário o uso de uma linguagem de programação. Esta linguagem é o meio de comunicação entre o homem e a máquina, sendo que o programa criado pelo homem deve ser interpretado e executado pela máquina. (NAHAS, et al., 2012)

Existem inúmeras maneiras de definir linguagem de programação, algumas delas são:

- Linguagem usada para preparar programas de computador, *American Standard Vocabulary for Information Processing* (ASVIP, 1970);
- Linguagem artificial para expressar programas, (ISO,2001);
- Notação auto consistente para a descrição precisa de programas de computador, (Wizzit, 2001).

Segundo (Sammit, 2009) as definições no geral seguem o mesmo padrão, pois, não refletem o uso do idioma, não requer o conhecimento do código de máquina pelo programador, devem ser independente da máquina e cada instrução de um programa escrito em determinada linguagem ao ser convertido em código de máquina gera um conjunto de instrução de máquina. (NAHAS, et al., 2012)

2.3.4.7.1. Classificação das linguagens de programação

As linguagens de programação podem ser divididas em paradigmas de programação e então ser classificadas de acordo com o domínio utilizado, isto porque uma linguagem pode conter mais de um paradigma. (NAHAS, et al., 2012) A programação pode ser:

- Processual: este paradigma é baseado na decomposição do programa em um conjunto de procedimentos;
- Orientado a objetos: neste paradigma o programa é organizado em conjunto de dados, também conhecidos como objetos, que interagem entre si. A programação orientada a objeto começou a ser mais utilizada do início da década de 1990 e ainda hoje é um dos principais paradigmas utilizados.
- Funcional: utilizando funções matemáticas para processar os dados, no programa uma função pode tomar uma segunda função como parâmetro e o resultado ainda ser uma função.
- Lógica: assim como sugere o nome, esse paradigma utiliza a lógica matemática como processamento.

Saber o objetivo do uso da linguagem é importante, pois dificilmente uma única linguagem atenderá toda as áreas de programação. (NAHAS, et al., 2012) (NAHAS, et al., 2012) Desta forma, as linguagens dividem-se ainda de acordo com sua finalidade que são:

- Linguagem de programação de uso geral: pode ser amplamente empregada em diversas áreas da programação em uma infinidade de aplicações.
- Linguagem de programação de sistemas: utilizada exclusivamente para desenvolvimento de *softwares* presente nos *hardwares* de computadores e eletrônicos em geral. A linguagem *Assembly* é a mais conhecida.
- Linguagem de programação de *script*: este tipo de linguagem é aplicado em situações em que não há compilação, ou seja, os comandos são interpretados e imediatamente executados, como nas páginas da *internet.*
- Linguagem de programação de domínio específico: destinadas para fins exclusivos, ou seja, não servem para áreas que não sejam especificadas. O Csound, por exemplo, só serve para tratamento de áudio.
- Linguagem de programação concorrente: utilizados para desenvolvimento de programas que executam tarefas simultaneamente de um mesmo programa ou de programas diferentes.

2.3.4.7.2. História das linguagens de programação

Assim como em qualquer área do conhecimento, na linguagem de programação não é diferente, onde o estudo da história e conceitos da programação evita erros que já fora cometido anteriormente por desenvolvedores. De acordo com (Pont, 2003), a evolução da linguagem de programação pode ser classificada em gerações com mostra a tabela 1. (NAHAS, et al., 2012)

Tabela 1 - Gerações das linguagens de programação

Geração da linguagem	Exemplo de linguagem
Primeira geração	Código de máquina *Assembly*
Segunda geração	COBOL, FORTRAN
Terceira geração - Orientada ao processo	C, Pascal
Quarta geração - Orientada a objeto	C++, Java

Fonte: (NAHAS, et al., 2012)

Com a invenção do computador na década de 1940 até o início da década de 1950, os desenvolvedores utilizavam apenas o código de máquina como linguagem de programação. Ainda no início desta década, a linguagem *Assembly* foi desenvolvida e substituiu o código de máquina. Apesar de ser menos complexa do que o código de máquina, a linguagem *Assembly* ainda requeria um alto nível de

conhecimento e era propensa a erros o que levou a sua rápida substituição. (NAHAS, et al., 2012)

Ao longo da década de 1950, surgiram diversas linguagens de programação de alto nível amplamente utilizadas atualmente e muitas que tiveram influência direta em boa parte das linguagens de programação atuais. Na década de 1960 surgiu a Simula, a primeira linguagem a suportar orientação a objetos. (NAHAS, et al., 2012)

Entre o final das décadas de 1960 e 1970, houve uma enorme ascensão das linguagens de programação. A linguagem C, por exemplo, uma linguagem de programação de sistemas desenvolvida entre 1969 e 1973 ainda hoje é muito empregada em programas. (NAHAS, et al., 2012)

Na década de 1980 foi desenvolvido o C++, uma linguagem combinada de sistemas operacionais. Nesta década houve um grande foco no desenvolvimento de linguagem de programação de sistemas em módulos. (NAHAS, et al., 2012)

Com a popularização da *internet* na década de 1990 veio à tona as linguagens de programação *script*, anteriormente utilizadas para outras finalidades. Essas linguagens eram basicamente versões modificadas de linguagens existentes e paradigmas baseados em linguagem C. (NAHAS, et al., 2012)

2.3.4.7.3. Linguagens de programação para sistemas de tempo real

Em sistemas de tempo real existem diferentes situações que determinam a precisão de tempo na execução das tarefas do programa. O nível de gravidade de acidentes e/ou consequências causadas ao ser humano ou ao meio ambiente em geral, caso determinada tarefa tenha um erro ou atraso de frações de segundos. Num sistema de freio, por exemplo, o tempo de leitura dos sensores de rotação das rodas e o acionamento das válvulas controladoras de pressão do fluído de freio, responsáveis pelo controle de frenagem individual de cada roda, tem um determinado tempo de execução para evitar acidentes ocasionados por perda de controle do veículos por falha no sistema de freio. Já num sistema de monitoramento, em que câmeras programadas para desligar durante os fins de semanas, não teria graves consequências caso haja um atraso no desligamento do sistema em questão. (NAHAS, et al., 2012)

Devido a estas diferentes características, os sistemas de tempo real podem ser classificados em:

- Rígido: sistemas em que quando o prazo de execução de determinada tarefa não é cumprido ocorre uma falha grave no sistema. Muito utilizado em sistemas de acionamento automáticos e sistemas de emergência, onde a tarefa deve ser executada imediatamente quando solicitada ou no exato momento planejado como em aviões, usinas nucleares e marca passos.
- Firme: este sistema permite o atraso em determinadas situações, em que este atraso não gere anomalias ou falhas no sistema. Por exemplo, quando há um atraso na comunicação, onde, mesmo que esta informação seja de suma importância para o sistema o atraso não gera nenhum acidente ou falha significativa. Num sistema de vigilância, quando ocorre um erro ou atraso em um *frame* do vídeo não afeta os demais frames, somente o frame corrompido.
- Suave: neste sistema, mesmo em situações de atraso na comunicação os dados são aproveitados com nas estações meteorológicas, em que mesmo com atraso de informações de temperatura, umidade e velocidade do vento, ainda é possível realizar as previsões do clima.

Para o desenvolvimento de sistemas de tempo real são necessários ferramentas de simulação e testes que garantem uma melhor eficácia do sistema que não eram possíveis serem feitas em *Assembly*. Pensando nisto, desenvolvedores da década de 1960 decidiram desenvolver linguagens de alto nível de fácil aprendizagem, depuração, manutenção, documentação e portabilidade para programação de sistemas de tempo real. (NAHAS, et al., 2012)

Segundo Boulton e Reid (1969) os principais requisitos para o desenvolvimento de sistemas de tempo real são os métodos de tratamento de sinais, interrupções e agendamento de tarefas.

Para atender tais requisitos, era possível através de extensões e modificações das linguagens de programação já existentes. Também era possível criar linguagens específicas para o desenvolvimento de sistemas de tempo real que, ainda assim, teriam muito em comum com as linguagens conhecidas na época. (NAHAS, et al., 2012)

Na década seguinte, 1970, os desenvolvedores buscavam um meio de programação de tempo real simultânea, já que mesmo com as extensões e modificações das linguagens de uso geral não se obtinha uma boa simultaneidade. (NAHAS, et al., 2012)

Em 1983 surgiu a ADA, uma linguagem de programação de alto nível orientada a objetos utilizada pelo departamento de defesa dos Estados Unidos, ganhou destaque devido a facilidade de obtenção de segurança, confiabilidade e previsibilidade na execução do programa sendo uma das mais utilizadas. (NAHAS, et al., 2012)

2.3.4.7.4. Escolha da linguagem de programação

Para a melhor escolha da linguagem de programação para o desenvolvimento de sistemas de tempo real é preciso levar e consideração o tempo de desenvolvimento, conhecimento da linguagem em questão por parte dos programadores, as características do *hardware* utilizado e a portabilidade da linguagem em relação a tarefa a ser executada pelo programa. (NAHAS, et al., 2012)

Pont (2003) observou os seguintes fatores a serem considerados em sistemas embarcados:

- A linguagem deve ser eficiente o suficiente para suprir as restrições do *hardware;*
- Requer um acesso de baixo nível para leitura e gravação de endereços específicos da memória através do uso de ponteiros;
- Suportar criação de bibliotecas flexíveis para serem reutilizadas em projetos distintos com mínimas alterações;
- Linguagem popular facilita o aprendizado e recrutamento de programadores devido a vasta documentação disponível para consultas.

Enfim, não existe e nem será criada uma linguagem ideal para cada caso, assim, a linguagem escolhida deve ser a mais apropriada o possível.

3. DESENVOLVIMENTO

Neste capítulo será apresentado o desenvolvimento deste projeto, mostrando como foi realizado o desenvolvimento do *software* para o sistema de sincronismo do motor.

3.1. Hardware de simulação dos sinais de sincronismo do motor

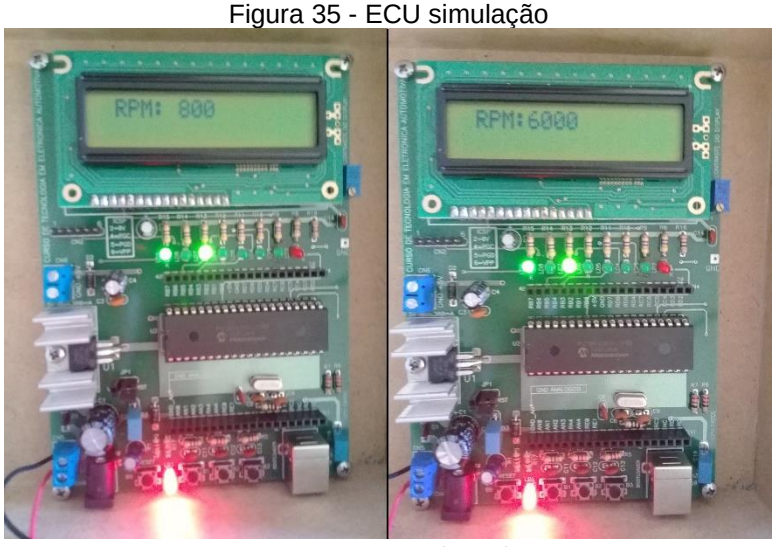
Com o intuito de gerar os sinais dos sensores de rotação e fase do MCI, foi empregado a placa de desenvolvimento FATEC e construído um *software* e dessa forma simular o funcionamento do MCI para o desenvolvimento do *software* de sincronismo do motor.

O *software* desenvolvido fornece a possibilidade de selecionar diferentes funcionalidades:

- Incremento/ decremento de valores de rotação;
- MCI em marcha lenta.

Na figura 35 vemos a placa de desenvolvimento FATEC com o programa de simulação nos dois extremos de rotação possíveis.

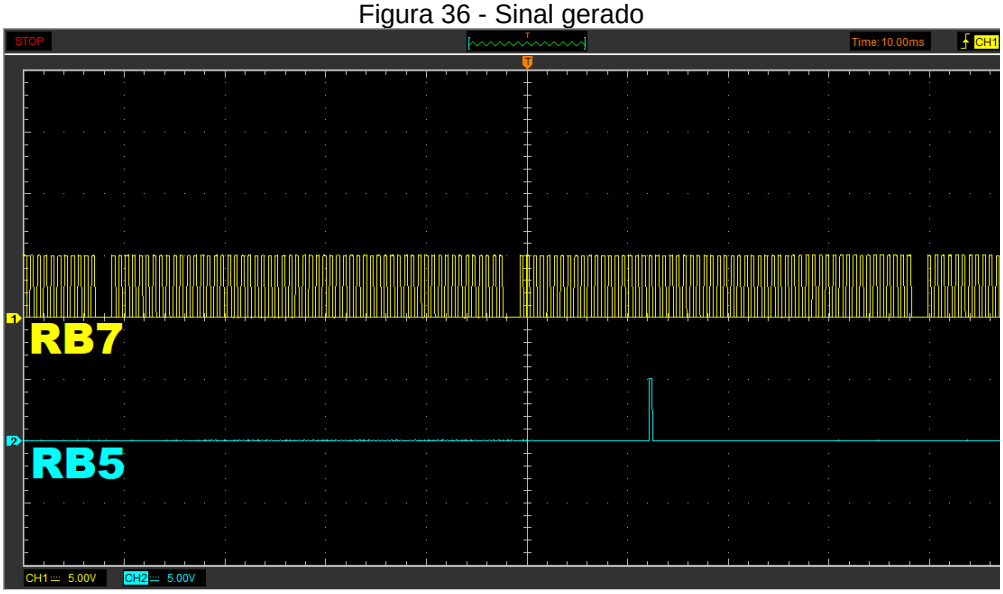
Figura 35 - ECU simulação



Fonte: o autor (2020)

O sinal de onda quadrada da roda fônica 60-2 é obtido no pino RB7 do microcontrolador PIC18F4550 e o sinal do sensor de fase da árvore de comando de válvulas é obtido no pino RB5. Na figura 36 é ilustrado o sinal dos respectivos pinos obtidos no osciloscópio.

Figura 36 - Sinal gerado

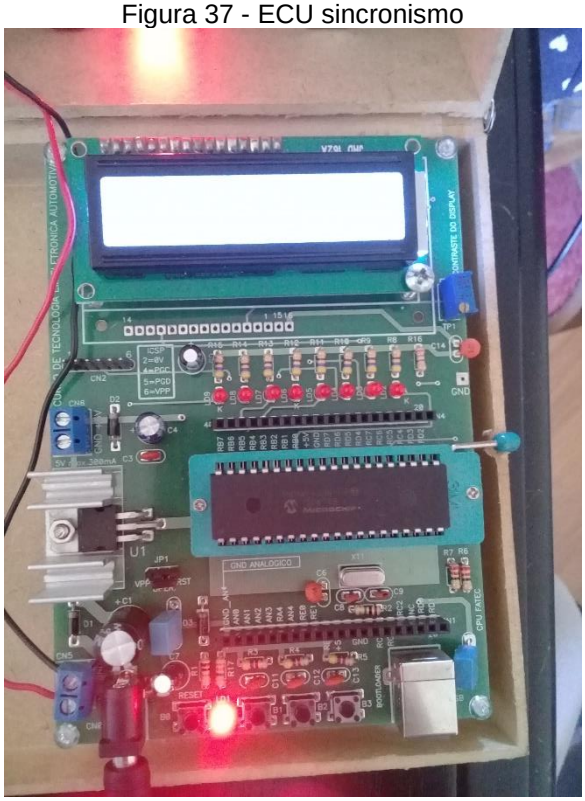


Fonte: O autor (2020)

3.2. Hardware de sincronismo do motor

Para o desenvolvimento do sistema de sincronismo, foi utilizada uma outra placa de desenvolvimento FATEC, figura 37.

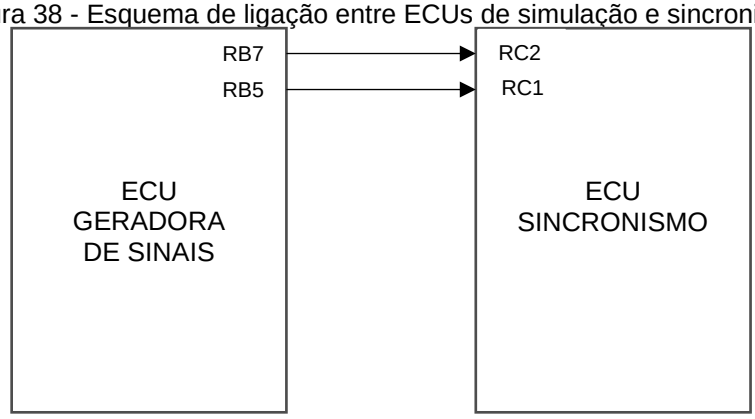
Figura 37 - ECU sincronismo



Fonte: O autor (2020)

O *hardware* de sincronismo ligado ao *hardware* gerador de sinais, como ilustrado na figura 38, realizará a leitura das informações geradas dos sinais de rotação e fase.

Figura 38 - Esquema de ligação entre ECUs de simulação e sincronismo.



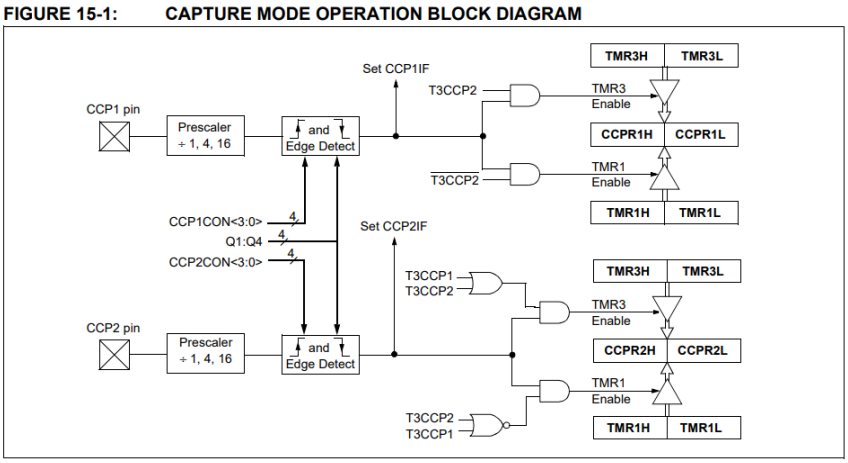
Fonte: O autor (2020)

Devido ao sinal de rotação possuir a característica de variação de tempos entre 83,3µs e 625µs, a implementação da leitura desse sinal pode ser feita de duas formas:

1. *Pooling:* Neste sistema o microcontrolador espera a troca de estado de uma determinada porta através da leitura contínua, impedindo que este realize qualquer outra tarefa.
2. Interrupção: Este método permite que o microcontrolador execute outras tarefas e, quando ocorrer mudança de estado da porta, o programa interrompe o que estiver fazendo e executa a rotina de interrupção. Ao fim da rotina de interrupção o programa volta ao mesmo ponto de onde foi interrompido.

No *hardware* de sincronismo, os pinos RC1 e RC2 foram configurados como CCP2 e CCP1 respectivamente no modo de captura, como ilustra a figura 39.

Figura 39 - Esquema de funcionamento da CCP1 e CCP2.



Fonte: Adaptado de PIC18F2455/2550/4455/4550 Data Sheet (2006)

Para o projeto foi utilizado o pino RC2, configurado como interrupção.

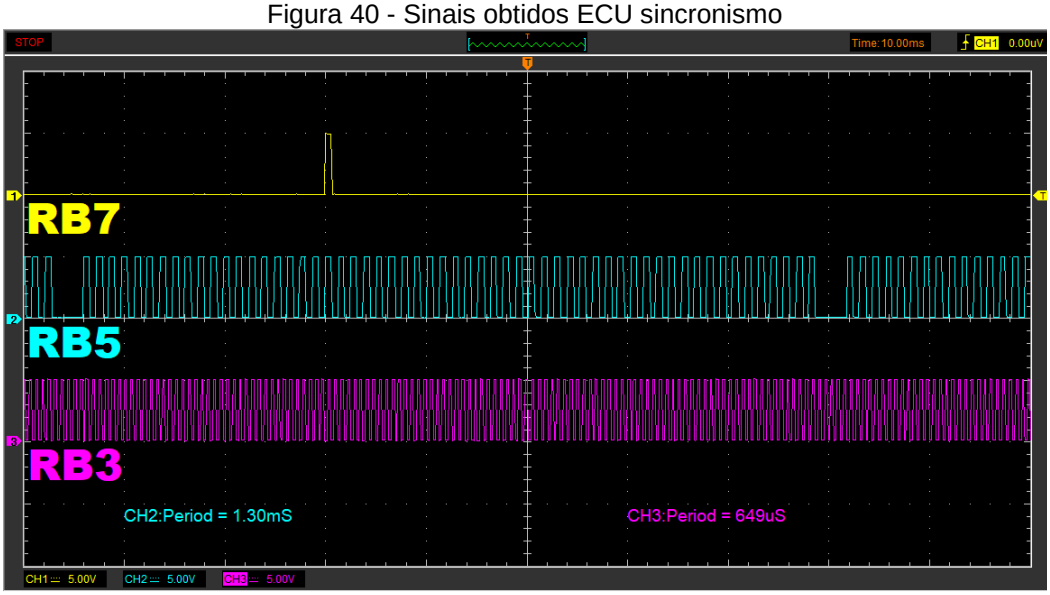
Dessa forma o sinal recebido de rotação da árvore de manivelas causa uma interrupção, também foi adicionado um *led* ao pino RB7, reproduzindo um sinal similar ao de entrada no RC2, que será empregado para a depuração funcional do *software*.

O pino RC1 recebe o sinal do sensor de fase e, a cada interrupção gerada por esse pino inverte o estado do *led* ligado ao RB5, reproduzindo um sinal similar ao de entrada no pino RC1.

O sinal de rotação é tratado pelo *software* e então gera um sinal com metade do período de onda do sinal em entrada que é transmitido ao pino RB3 e tem um *led* a ele ligado.

Com um osciloscópio ligado aos respectivos pinos temos os sinais ilustrados na figura 40.

Figura 40 - Sinais obtidos ECU sincronismo



Fonte: O autor (2020)

Com o sinal gerado em RB3 obtém-se uma precisão de leitura de 1,5°.

3.3. Software de simulação do motor

Utilizando os periféricos do microcontrolador PIC18F4550, as interrupções por base de tempo, o programa foi desenvolvido com base na característica dos sinais dos sensores de rotação e fase, respeitando as suas respectivas características de tempo e nível lógico. Na tabela 2 encontram-se os valores de tempo das respectivas rotações.

Tabela 2 - Valores de tempo da ECU de simulação

Rotação [rpm]	Rotação [rps]	Tempo de volta [ms]	Tempo de dente [µs]
800	13,333	75	625
1000	16,667	60	500
1500	25	40	333,333
2000	33,333	30	250
2500	41,667	24	200
3000	50	20	166,667
3500	58,333	17,143	142,857
4000	66,667	15	125
4500	75	13,333	111,111
5000	83,333	12	100
5500	91,667	10,909	90,909
6000	100	10	83,333

Fonte: O autor (2020)

Com o valor de tempo equivalente a três graus, referente ao intervalo da extremidade do dente ou do intervalo entre um dente e outro, determina-se o valor inicial do registrador de tempo, *timer* 0, para que a interrupção seja gerada no tempo exato para cada valor de rotação. Para determinar o valor inicial de contagem utiliza-se a fórmula 1.

Fórmula 1 - Determinar valor inicial da contagem do timer

$$Valor\ inicial = tempo\ máximo - \frac{tempo\ desejado}{tempo\ de\ incremento}$$

Fonte: O autor (2020)

Com os valores de tempos desejados aplicados a fórmula 1, obteve-se os valores iniciais apresentados na tabela 3, com os respectivos valores hexadecimais a serem inseridos no programa.

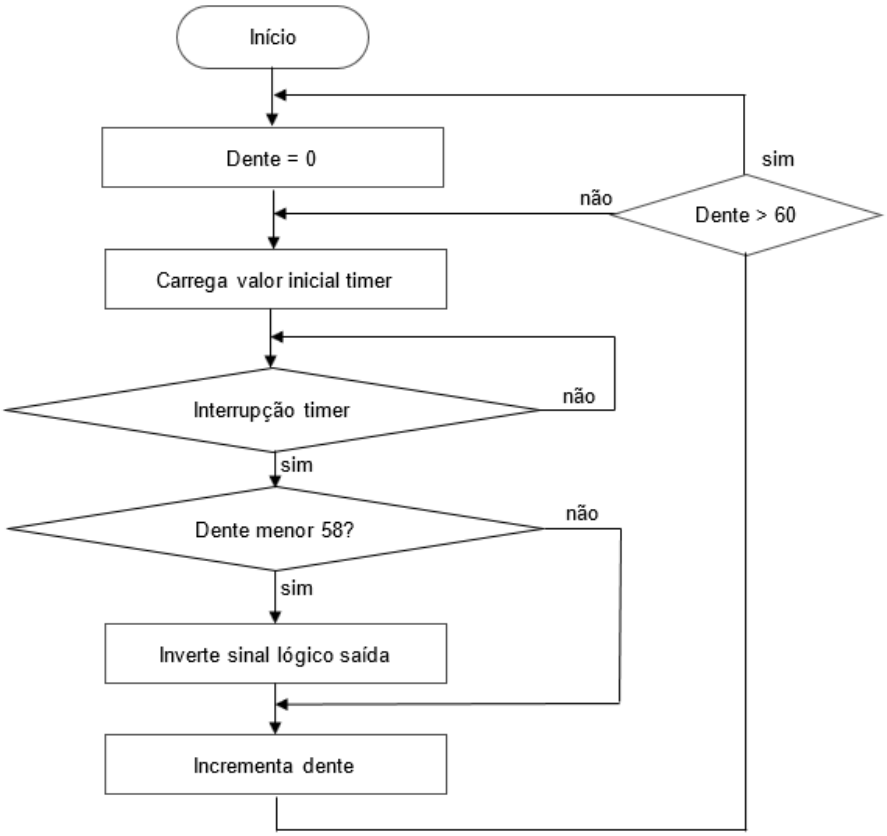
Tabela 3 - Tabela valor inicial Timer 0

Rotação [rpm]	Tempo desejado [µs]	Pré-escala	Valor inicial decimal	Valor inicial hexadecimal	Valor inicial calibrado hexadecimal
800	625	32	22	15	14
1000	500	32	69	44	43
1500	333,333	32	131	83	82
2000	250	32	162	A2	A1
2500	200	32	181	B5	B4
3000	166,667	8	06	06	05
3500	142,857	8	42	29	28
4000	125	8	69	44	43
4500	111,111	8	89	59	58
5000	100	8	106	6A	69
5500	90,909	8	120	77	76
6000	83,333	8	131	83	82

Fonte: O autor (2020)

Determinado o valor inicial, e calibrando o valor devido a latência, aplica-se no programa de acordo com o gráfico 2.

Gráfico 2 - Fluxograma programa simulação



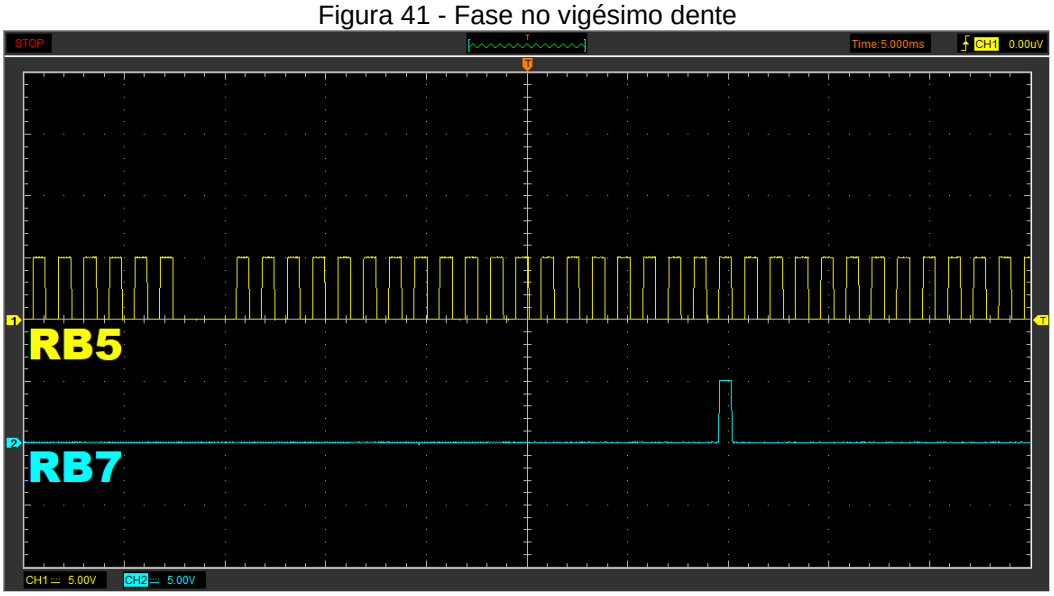
Fonte: O autor (2020)

3.4. Software de sincronismo

O programa de sincronismo consiste basicamente em espelhar o sinal recebidos do motor para que assim possa fazer a aquisição do tempo de rotação e garantir o sincronismo da injeção e ignição.

Inicialmente, o programa habilita somente o CCP2 no modo captura para que toda borda de subida, ligado ao sensor de fase. Assim que ocorrer a interrupção, o registrador de posição atual da roda fônica tem seu valor carregado com o valor referente ao PMS do primeiro e quarto cilindro, como se vê na figura 41 o PMS está no vigésimo dente, e habilita uma flag indicando que o motor está na primeira volta do ciclo Otto.

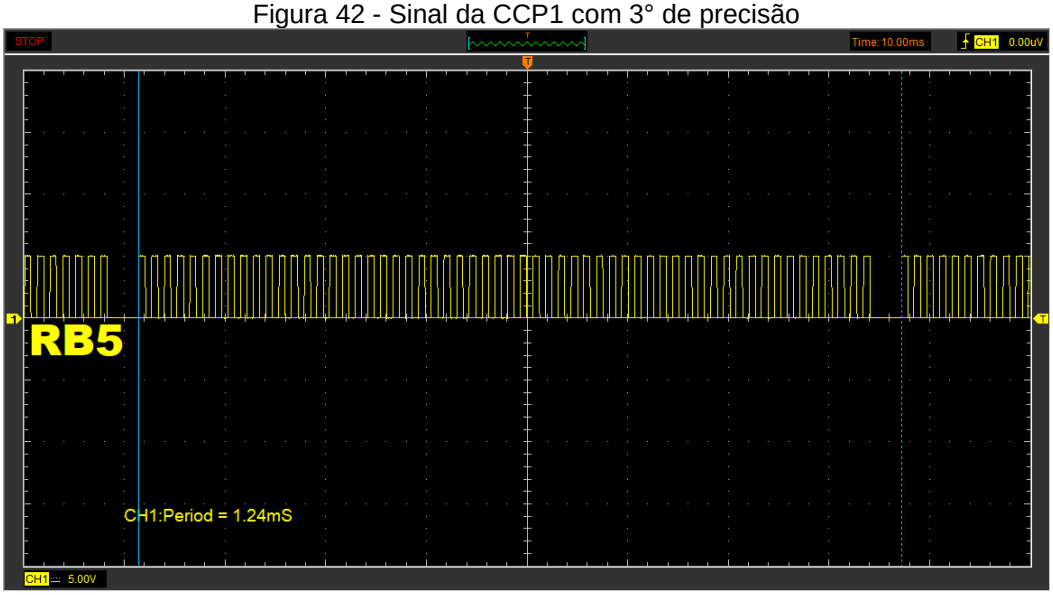
Figura 41 - Fase no vigésimo dente



Fonte: O autor (2020)

Após o sincronismo de fase do motor, é habilitado o CCP1 em modo de captura, e configurada inicialmente para detectar todas as bordas de subida do sinal. Quando esta condição é satisfeita, o CCP1 é reconfigurado para atuar em todas as bordas de descida, garantindo uma precisão de três graus, como ilustrado na figura 42.

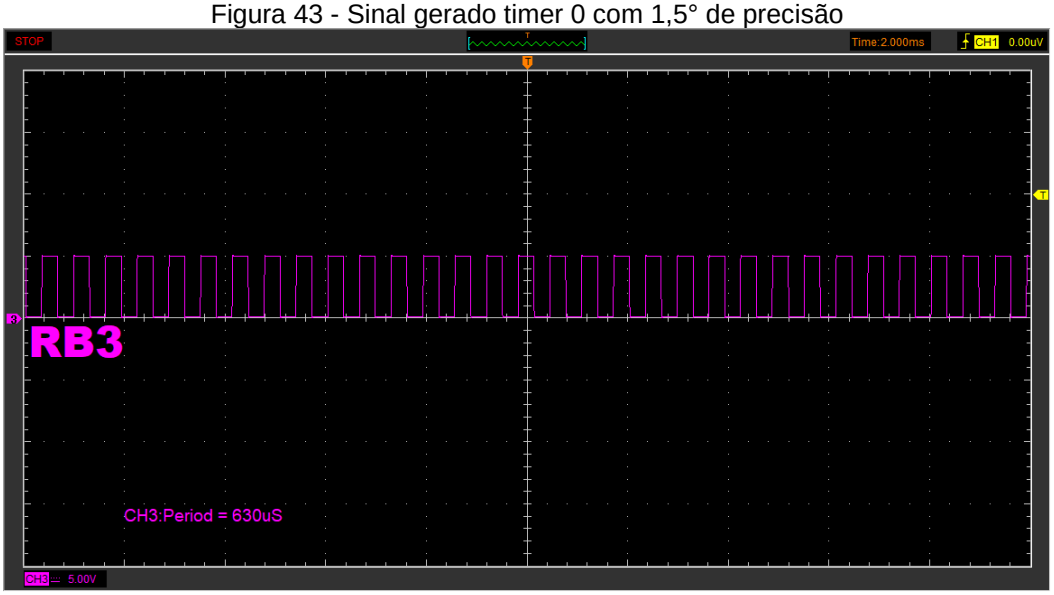
Figura 42 - Sinal da CCP1 com 3° de precisão



Fonte: O autor (2020)

A cada interrupção da CCP1 o valor do registrador de posição atual da roda fônica é incrementado. Com metade do valor de *timer* 1 definido como tempo de interrupção do *timer* 0, obtém-se uma precisão de um grau e meio da posição da roda fônica. O *led* ligado ao RB3 é aceso pela interrupção do *timer* 0 e apagado pela interrupção gerada pela CCP1, gerando o sinal da roda fônica digital ilustrado na figura 43.

Figura 43 - Sinal gerado timer 0 com 1,5° de precisão



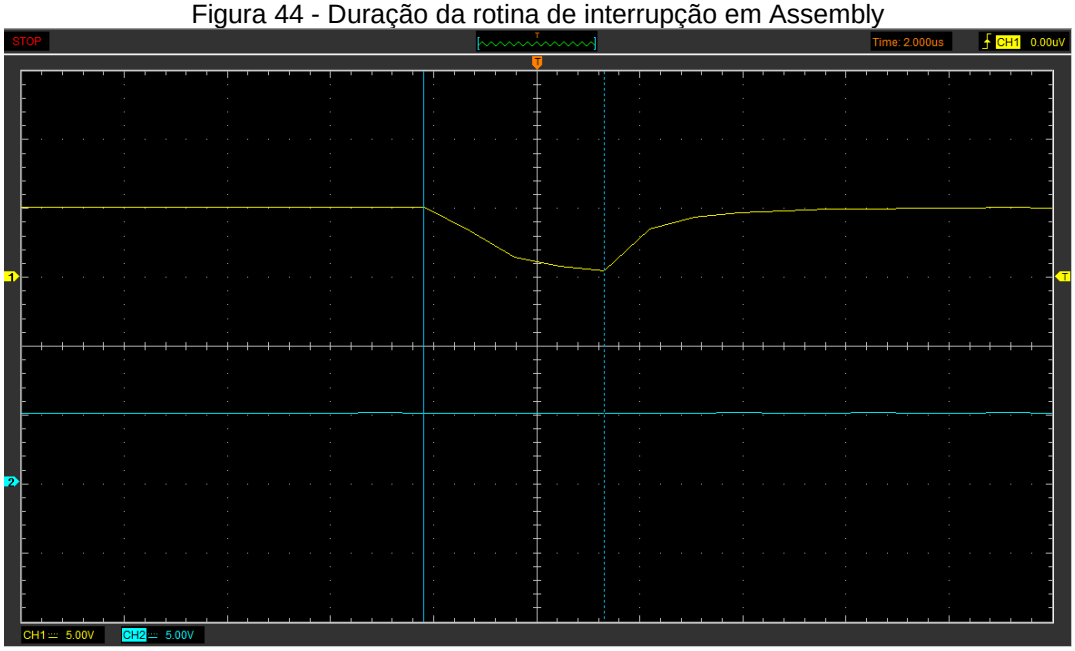
Fonte: O autor (2020)

4. TESTES E RESULTADOS

Após a implementação do *software* em *Assembly*, iniciou-se o processo de teste de desempenho entre o sistema desenvolvido em *Assembly* e o existente em linguagem C.

Na figura 44, é demonstrado o tempo de detecção de cada dente da roda fônica com o programa implementado em *Assembly*, o tempo é de 3,21*µs.*

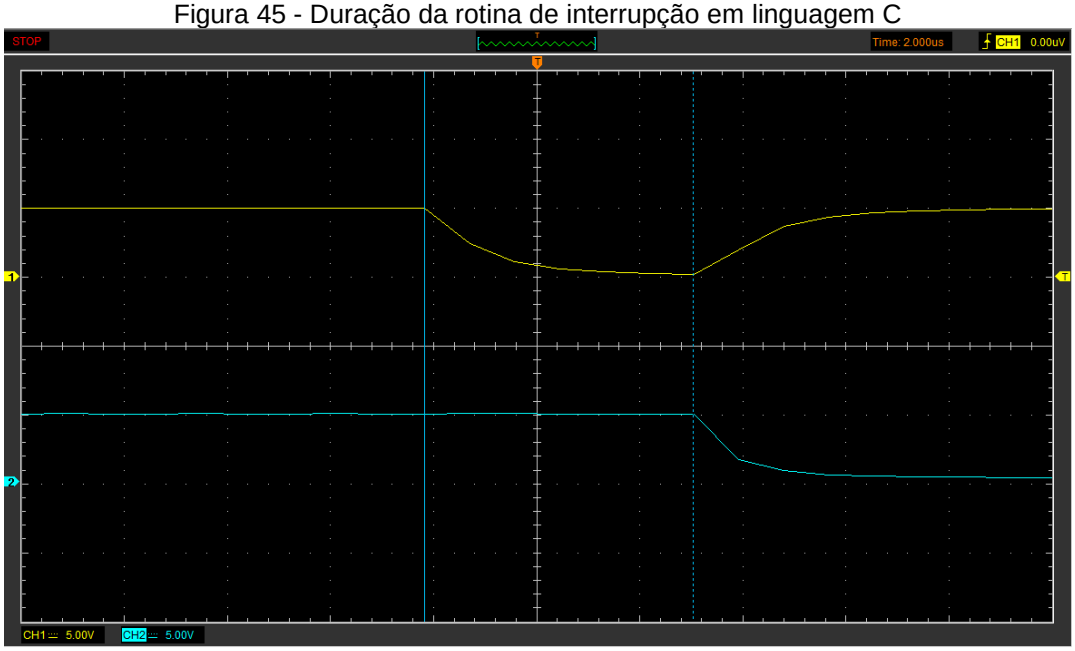
Figura 44 - Duração da rotina de interrupção em Assembly



Fonte: O autor (2020)

A implementação do mesmo algoritmo de leitura dos dentes da roda fônica em linguagem C, figura 45, levou aproximadamente 5*µs.*

Figura 45 - Duração da rotina de interrupção em linguagem C



Fonte: O autor (2020)

Posteriormente, foram realizados testes com diferentes valores de rotação, na tabela 4 estão os resultados dos testes referentes ao sincronismo do primeiro dente da roda virtual, equivalente à metade do primeiro dente da roda fônica.

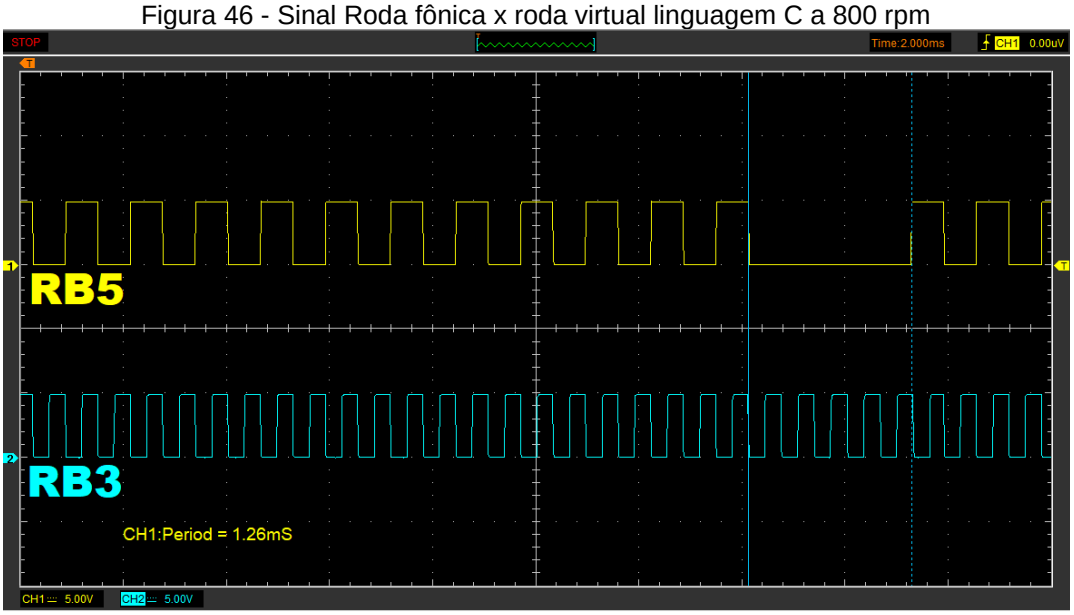
Tabela 4 - Sincronismo do primeiro dente da roda virtual

Rotação [Rpm]	Assembly	Linguagem C
800	Ok	Ok
1000	Ok	Ok
1500	Ok	Falha
2000	Ok	Falha
2500	Ok	Falha
3000	Ok	Falha
3500	Ok	Falha
4000	Ok	Falha
4500	Ok	Falha
5000	Falha	Falha
5500	Falha	Falha
6000	Falha	Falha

Fonte: O autor (2020)

É possível ver nos resultados obtidos que, no *software* desenvolvido em *Assembly* o sincronismo tem uma melhor precisão, apresentando falhas somente em rotações mais elevadas. Nas rotações de 800 e 1000 rpm, o primeiro dente da roda virtual apresentou o mesmo resultado, conforme ilustrado na figura 46.

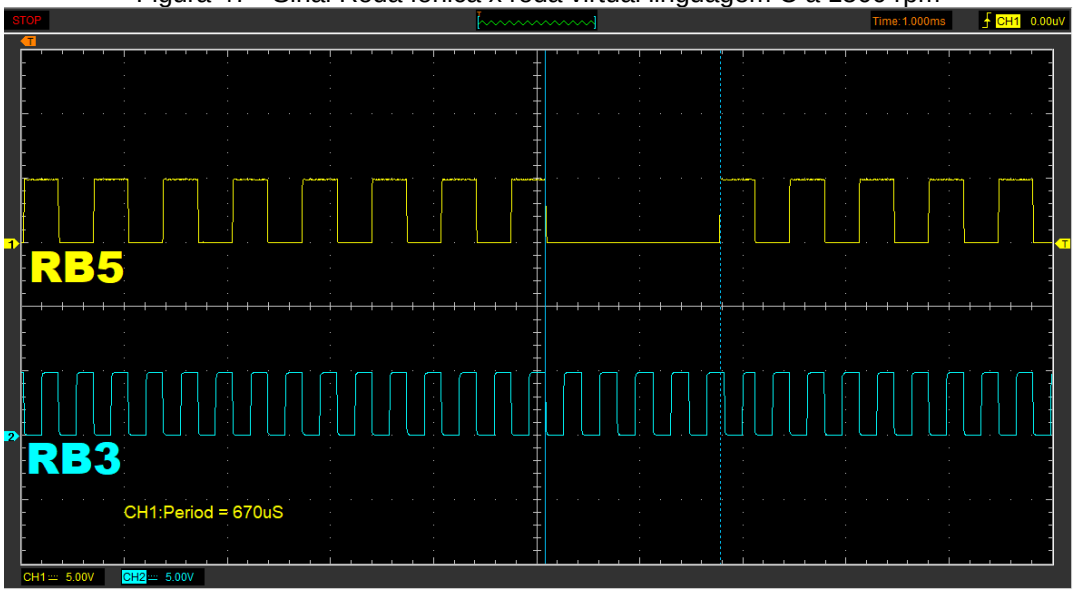
Figura 46 - Sinal Roda fônica x roda virtual linguagem C a 800 rpm



Fonte: O autor (2020)

Com o incremento da rotação, a partir de 1500 rpm, o programa desenvolvido em linguagem C apresentou falhas de sincronismo, como visto na figura 47.

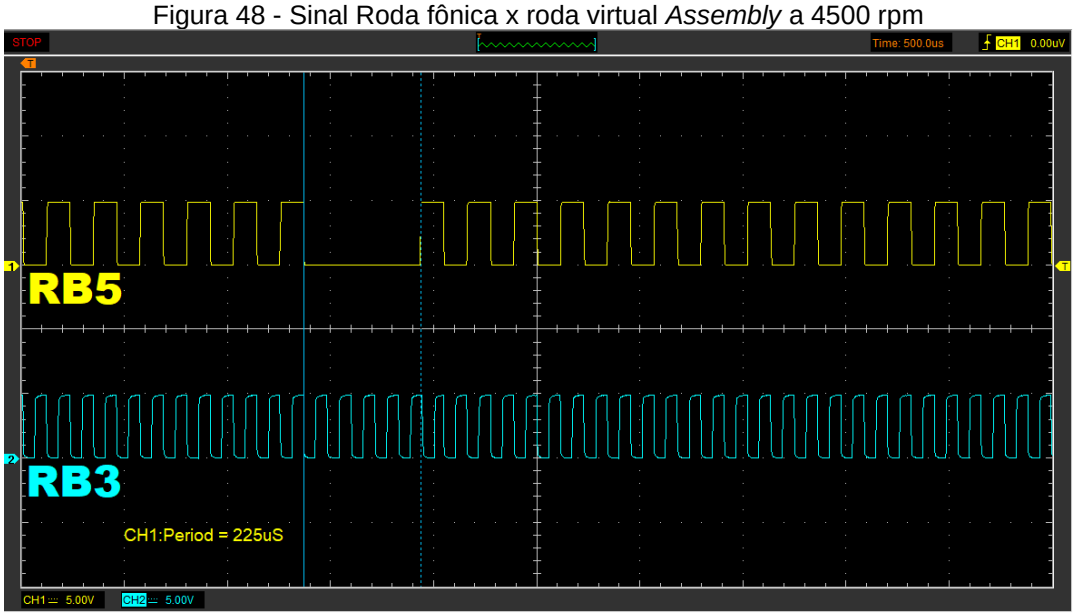
Figura 47 - Sinal Roda fônica x roda virtual linguagem C a 1500 rpm



Fonte: O autor (2020)

Enquanto, no programa *Assembly,* o sincronismo permaneceu íntegro quanto ao sincronismo até a rotação de 4500 rpm como ilustra a figura 48.

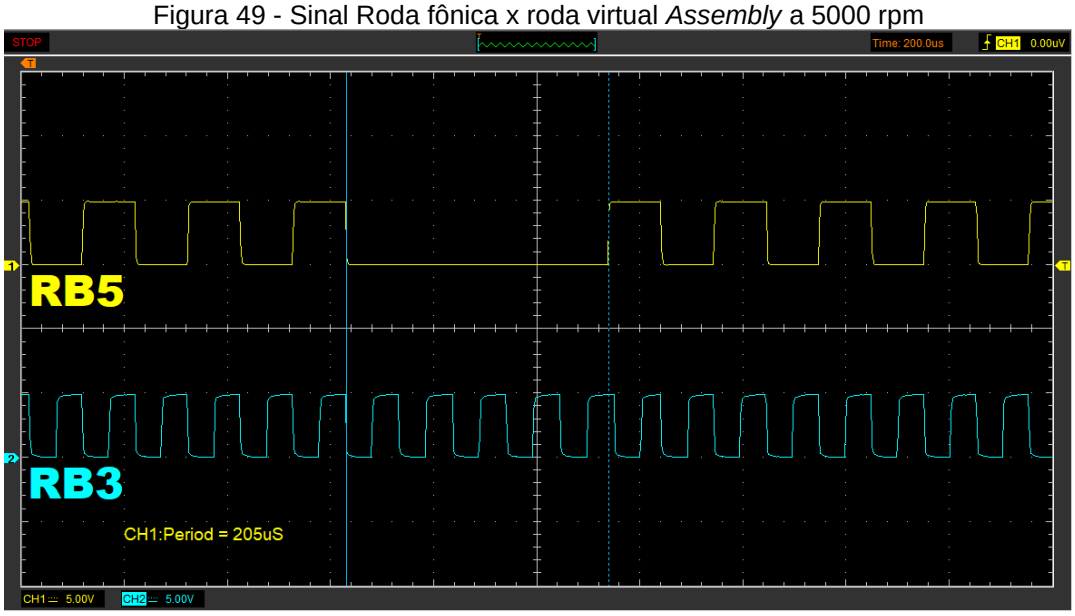
Figura 48 - Sinal Roda fônica x roda virtual *Assembly* a 4500 rpm



Fonte: O autor (2020)

A partir da rotação de 5000 rpm, o programa desenvolvido em *Assembly* começa a apresentar falha no sincronismo do primeiro dente, como pode se ver na figura 49. Foi denominado falha, pois o processamento do dente tende a se atrasar.

Figura 49 - Sinal Roda fônica x roda virtual *Assembly* a 5000 rpm



Fonte: O autor (2020)

5. CONCLUSÃO

Após os resultados obtidos, concluiu-se que, apesar da linguagem C ter preferência na programação, devido a sua facilidade e rapidez na programação, apresenta uma certa latência no processamento. Quando o compilador converte da linguagem C para linguagem de baixo nível nem sempre faz da maneira mais eficiente, como se pode ver nos resultados obtidos neste estudo, que compara dois programas iguais em linguagens diferentes.

Sabendo que em um MCI o sincronismo interfere diretamente em questões ambientais como emissão de poluentes, consumo excessivo de combustível e desgaste prematuro de componentes e também em questões de satisfação ao usuário como conforto, potência e durabilidade, utilizando a linguagem *Assembly* ao menos no controle do sincronismo, já obteríamos um ganho significativo em todos esses aspectos

5.1. Trabalhos futuros

Integrar o desenvolvimento com um *software* de injeção eletrônica completo;

Utilizar o *software* desenvolvido como uma biblioteca em linguagens de alto nível, permitindo a utilização em *softwares* de gerenciamento já existentes;

Realizar testes funcionais com o *software* desenvolvido e comparar com o existente.

6. REFERÊNCIAS

BRUNETTI, FRANCO. 2012. *Motores de Combustão Interna.* São Paulo : Blucher, 2012.

CAPELLI, ALEXANDRE. 2010. *Eletroeletrônica Automotiva Injeção Eletrônica, Arquitetura do Motor e Sistemas Embarcados.* São Paulo : Érica, 2010.

DELPHI. 2003. ETC Air Control Valve (ETC-ACV). *Application Manual.* Rochester, Nova Iorque, Estados Unidos da América : Delphi Corporation, Junho de 2003.

—. 2005. Ignition System Application Manual. *Energy and chassis.* Brighton, Michigan, Estados Unidos da América : Delphi Automotive Systems, Janeiro de 2005.

GLEHN, FABIO RIBEIRO von. 2001. *Curso de Injeção Eletrônica.* Goiânia : Ciclo Engenharia, 2001.

MANAVELLA, HUMBERTO JOSÉ. 2003. *Controle Integrado do Motor Sistemas de Injeção- Ignição Eletrônica.* São Pauo : HM Autotrônica, 2003.

MICROCHIP. 2006. PIC18F2455/2550/4455/4550 Data sheet 28/40/44-Pin, High-Performance. Chandler and Tempe, Arizona and Mountain View, California : s.n., 2006.

MOUAAZ NAHAS, ADI MAAITA. 2012. Choosing Appropriate Programming Language to Implement Software to Real-Time Resource-Constrained Embedded Systems. 2012.

7. APÊNDICES

7.1. Apêndice A - Programa simulador de sinais de rotação e fase

```
;=====================================================================
;Arquivo:          rodadigital.asm
;Autor:            Felipe Marotti Mochiuti
;=====================================================================

;=====================================================================
;                                    CONFIGURAÇÕES
;=====================================================================
LIST         p=18f4550
#include     "p18f4550.inc"
  CONFIG  PLLDIV = 5            ; PLL: Divide por 5 para cristal de 20
MHz (A frequência de entrada deve ser de 4Mhz)
  CONFIG    CPUDIV   =   OSC1_PLL2         ;  Configura  frequência  do
microcontrolador para 48Mhz
  CONFIG  FOSC = HSPLL_HS       ; Configura cristal oscilador como High
Speed e habilita multiplicador de frequência PLL
  CONFIG  PWRT = ON              ; Habilita Power-up, aguarda em média
65,5 ms até que VCC se estabilize
  CONFIG  BOR = OFF             ; Desabilita Brown-out Reset
  CONFIG  WDT = OFF             ; Desabilita Watchdog Timer
  CONFIG  PBADEN = OFF          ; PORTB configurado como portas digitais
  CONFIG  MCLRE = ON            ; Pino RE3 habilitado como MCLR
;=====================================================================

;=====================================================================
;                               ENTRADAS E SAÍDAS
;=====================================================================
;                                    SAÍDAS
;---------------------------------------------------------------------
#define      sinal            LATB,7           ; Saída do sinal gerado
da roda fônica
#define      fase             LATB,5           ; Saída do sinal gerado
da fase do comando
;---------------------------------------------------------------------
;                                    ENTRADAS
;---------------------------------------------------------------------
#define      B1                    PORTE,0          ;   Botão   para
abaixar a rotação
#define      B2                    PORTE,1          ;   Botão   para
rotação de marcha lenta 800 Rpm
#define      B3                    PORTE,2          ;   Botão   para
aumentar a rotação
;=====================================================================

;=====================================================================
;                               DEFINIÇÕES LCD

;=====================================================================
;                                    ESCRITA LCD
;---------------------------------------------------------------------
#define      S_RS       bsf         LATD,1      ; Comando setar o bit
Register Select do display ligado ao RD1
#define      S_EN       bsf         LATD,0      ; Comando setar o bit
```

```
Enable do display ligado ao RD0
     #define      S_RW        bsf         LATD,2      ; Comando setar o bit
Read/Write do display ligado ao RD2
     #define      C_RS        bcf         LATD,1      ; Comando zerar o bit
Register Select do display ligado ao RD1
     #define      C_EN        bcf         LATD,0      ; Comando zerar o bit
Enable do display ligado ao RD0
     #define      C_RW        bcf         LATD,2      ; Comando zerar o bit
Read/Write do display ligado ao RD2
     ;-----------------------------------------------------------------------
     ;                                    PINOS LCD
     ;-----------------------------------------------------------------------
     #define      S_D4        bsf         LATD,4      ; Comando setar o bit
Data Pin 4 do display ligado ao RD4
     #define      S_D5        bsf         LATD,5      ; Comando setar o bit
Data Pin 5 do display ligado ao RD5
     #define      S_D6        bsf         LATD,6      ; Comando setar o bit
Data Pin 6 do display ligado ao RD6
     #define      S_D7        bsf         LATD,7      ; Comando setar o bit
Data Pin 7 do display ligado ao RD7
     #define      C_D4        bcf         LATD,4      ; Comando zerar o bit
Data Pin 4 do display ligado ao RD4
     #define      C_D5        bcf         LATD,5      ; Comando zerar o bit
Data Pin 5 do display ligado ao RD5
     #define      C_D6        bcf         LATD,6      ; Comando zerar o bit
Data Pin 6 do display ligado ao RD6
     #define      C_D7        bcf         LATD,7      ; Comando zerar o bit
Data Pin 7 do display ligado ao RD7
     ;-----------------------------------------------------------------------
     ;                              FLAGS RODA VIRTUAL
     ;-----------------------------------------------------------------------
     #define      f_falha                 FLAG,7           ; Flag indicadora
da falha da roda fônica
     #define      f_fase                  FLAG,6           ; Flag indicadora
da fase do comando de válvulas
     ;-----------------------------------------------------------------------
     ;                              FLAGS DEBOUNCE
     ;-----------------------------------------------------------------------
     #define      f_press                      DEBOUNCE_FLAG,0   ;      Flag
indicadora de botão pressionado
     #define      f_debounce              DEBOUNCE_FLAG,1   ; Flag indicadora
de debounce realizado
     #define      f_b1                    DEBOUNCE_FLAG,2 ;  Flag  indicadora
da primeira passagem pela rotina do botão b1
     #define      f_b2                    DEBOUNCE_FLAG,3   ; Flag indicadora
da primeira passagem pela rotina do botão b2
     #define      f_b3                    DEBOUNCE_FLAG,4   ; Flag indicadora
da primeira passagem pela rotina do botão b3
     ;-----------------------------------------------------------------------

     ;=======================================================================
     ;PAGINAÇÃO DE MEMÓRIA
     ;=======================================================================
     cblock       0x0020
     ;-----------------------------------------------------------------------
     ;  variáveis  para  salvar  contexto  antes  de  realizar  a  rotina  de
interrupção
          W_TEMP
          STATUS_TEMP
```

```
        BSR_TEMP
    ;-----------------------------------------------------------------------
     DENTE                  ; Variável armazena valor do dente atual
        AUX_RPM                     ; Variável armazena valor referente ao
tempo de interrupção do RPM selecionado
     FLAG                   ; Armazena valor das Flags de falha e fase
     RPM_VTR                ; Registrador de posição de RPM
    ;-----------------------------------------------------------------------
    ; variáveis do display
     COMANDO                        ; Variável utilizada para comando do
display
     CARACTER               ; Variável utilizada para escrita de caracter
no display
     AUX_DELAY              ; Variável utilizada para delay do display e
botões
     ENDERECO               ; Variável utilizada para endereço de escrita
no display
    ;-----------------------------------------------------------------------
    ; variáveis para debounce dos botões
     AUX_DEBOUNCE           ; Variável de tempo do debounce incrementada no
interrupção de tempo
     DEBOUNCE_FLAG          ; Registrador que armazena Flags auxiliares do
debounce
     VALOR_DEBOUNCE         ; Valor de tempo do debounce

    endc

    ;=======================================================================
    ;VARIAVEIS
    ;=======================================================================
    ; valores utilizados para base de tempo de cada dente da roda fônica com
uma pré-escala de 1:32
    #define      RPM_800                0x14  ; Tempo de dente de 625us
    #define      RPM_1000         0x43  ; Tempo de dente de 500us
    #define      RPM_1500         0x82  ; Tempo de dente de 333us
    #define      RPM_2000         0xA1  ; Tempo de dente de 250us
    #define      RPM_2500         0xB4  ; Tempo de dente de 200us
    ;-----------------------------------------------------------------------
    ; valores utilizados para base de tempo de cada dente da roda fônica com
uma pré-escala de 1:8
    #define      RPM_3000         0x05  ; Tempo de dente de 166us
    #define      RPM_3500         0x28  ; Tempo de dente de 142us
    #define      RPM_4000         0x43  ; Tempo de dente de 125us
    #define      RPM_4500         0x58  ; Tempo de dente de 111us
    #define      RPM_5000         0x69  ; Tempo de dente de 100us
    #define      RPM_5500         0x76  ; Tempo de dente de 90us
    #define      RPM_6000         0x82  ; Tempo de dente de 83us
    ;-----------------------------------------------------------------------
    ; valor das posições da roda fônica e fase do comando
    #define      valor_inicial    0x88  ; zera após 120 incrementos
    #define      valor_falha      0xFB  ; valor do intervalo da falha
    #define      valor_fase       0xAE  ; sinal de fase no 20° dente
    #define      t_debounce       0x9B  ; debounce de 100ms
    ;=======================================================================
    ;VETOR RESET
    ;=======================================================================
         org          0x0000
         goto    inicio
```

```
;=====================================================================
;INTERRUPÇÕES
;=====================================================================
       org         0x08
     movwf   W_TEMP                              ; salva w em W_TEMP
     movff   STATUS, STATUS_TEMP               ; salva status em STATUS_TEMP
     movff   BSR, BSR_TEMP                     ; salva bsr em BSR_TMEP
;---------------------------------------------------------------------
;                              TIMER 0
;---------------------------------------------------------------------
       btfsc INTCON,TMR0IE
       btfss INTCON,TMR0IF
       bra         saida_tmr0
       movff AUX_RPM,TMR0L
       incfsz      DENTE
       bra         testa_dente
       movlw valor_inicial
       movwf DENTE
       bcf         sinal
testa_dente:
       bcf         fase
       movlw valor_fase
       cpfseq      DENTE
       bra         testa_falha
       btg         f_fase
       btfsc f_fase
       bsf         fase
testa_falha:
       movlw valor_falha
       cpfsgt      DENTE
       btg         sinal
       bcf         INTCON,TMR0IF
saida_tmr0:
;---------------------------------------------------------------------
;                              TIMER 1
;---------------------------------------------------------------------
       btfss PIR1,TMR1IF
       goto  saida_tmr1
       movlw 0xD1
       movwf TMR1H
       movlw 0x1F
       movwf TMR1L
       decf  AUX_DELAY
       btfsc f_press
       incfsz      AUX_DEBOUNCE
       goto  saida_timer_1_ok
       bsf         f_debounce
saida_timer_1_ok:
       bcf         PIR1,TMR1IF
saida_tmr1:

;---------------------------------------------------------------------
;                              SAIDA INTERRUPÇÕES
;---------------------------------------------------------------------
saida_int:
     movff   BSR_TEMP, BSR                     ; recupera BSR
     movf    W_TEMP, W                         ; recupera WREG
     movff   STATUS_TEMP, STATUS               ; recupera STATUS
```

```
     retfie
;=====================================================================

;=====================================================================
;PROGRAMA PRINCIPAL
;=====================================================================
inicio:
     call    config_hw
       movlw t_debounce
       movff WREG,VALOR_DEBOUNCE
       clrf  DEBOUNCE_FLAG
       movlw valor_inicial
       movwf DENTE
       clrf  RPM_VTR
       clrf  FLAG
       setf  LATB

       call  lcd_init
       movlw 0x11
       call  endereco_lcd
       movlw 'R'
       call  caracter_lcd
       movlw 'P'
       call  caracter_lcd
       movlw 'M'
       call  caracter_lcd
       movlw ':'
       call  caracter_lcd
       movlw ' '
       call  caracter_lcd
       movlw '8'
       call  caracter_lcd
       movlw '0'
       call  caracter_lcd
       movlw '0'
       call  caracter_lcd
       bsf         INTCON,TMR0IE

loop:
       btfss f_b1
       btfss B1
       call  trata_b1

       btfss f_b2
       btfss B2
       call  trata_b2

       btfss f_b3
       btfss B3
       call  trata_b3
       goto  loop

config_hw:                                  ;configurações gerais
;---          in/out              ---
       movlw 0x0F
       movwf ADCON1
       movlw 0x00                  ;
       movwf TRISA                 ;
       clrf  PORTA
```

```
            movlw 0x00              ;
            movwf TRISB             ;
            clrf  PORTB
            movlw 0x80              ;
            movwf TRISC             ;
            clrf  PORTC
            movlw 0x00              ;
            movwf TRISD             ;
            clrf  PORTD
            movlw 0x07              ;
            movwf TRISE             ;
            clrf  PORTE

    ;---          interrupções      ---                *********
            bcf         RCON,IPEN         ;Desabilita      prioridade      de
interrupções
            movlw 0xC0              ;Habilita interrupções e periféricos
          movwf   INTCON

    ;---          timer0                  ---
            movlw 0xC4
            movwf T0CON
            movlw RPM_800
            movwf AUX_RPM
            movff AUX_RPM,TMR0L

    ;---          timer1                  ---
            movlw 0x81
            movwf T1CON
            clrf  TMR1L
            bsf         PIE1,TMR1IE

    ;---          ---
         return

    rpm_select:
           tstfsz       RPM_VTR
           goto  maior_0
           bsf          T0CON,T0PS2
           bcf          T0CON,T0PS1
           bcf          T0CON,T0PS0
           movlw RPM_800
           movwf AUX_RPM
           movlw 0x15
           call  endereco_lcd
           movlw ' '
           call  caracter_lcd
           movlw '8'
           call  caracter_lcd
           goto  saida_rpm_select

    maior_0:
           clrf  WREG
           addlw 0x01
           cpfsgt       RPM_VTR
           goto  rpm_1

           addlw 0x01
           cpfsgt       RPM_VTR
```

```
        goto  rpm_2

        addlw 0x01
        cpfsgt      RPM_VTR
        goto  rpm_3

        addlw 0x01
        cpfsgt      RPM_VTR
        goto  rpm_4

        addlw 0x01
        cpfsgt      RPM_VTR
        goto  rpm_5

        addlw 0x01
        cpfsgt      RPM_VTR
        goto  rpm_6

        addlw 0x01
        cpfsgt      RPM_VTR
        goto  rpm_7

        addlw 0x01
        cpfsgt      RPM_VTR
        goto  rpm_8

        addlw 0x01
        cpfsgt      RPM_VTR
        goto  rpm_9

        addlw 0x01
        cpfsgt      RPM_VTR
        goto  rpm_10
        goto  rpm_11

rpm_1:
        movlw RPM_1000
        movwf AUX_RPM
;       movlw 0xFE
;       movwf LATB
        movlw 0x15
        call  endereco_lcd
        movlw '1'
        call  caracter_lcd
        movlw '0'
        call  caracter_lcd
        goto  saida_rpm_select

rpm_2:
        movlw RPM_1500
        movwf AUX_RPM
;       movlw 0xFD
;       movwf LATB
        movlw 0x15
        call  endereco_lcd
        movlw '1'
        call  caracter_lcd
        movlw '5'
        call  caracter_lcd
```

```
        goto  saida_rpm_select

rpm_3:
        movlw RPM_2000
        movwf AUX_RPM
;       movlw 0xFC
;       movwf LATB
        movlw 0x15
        call  endereco_lcd
        movlw '2'
        call  caracter_lcd
        movlw '0'
        call  caracter_lcd
        goto  saida_rpm_select

rpm_4:
        bsf         T0CON,T0PS2
        bcf         T0CON,T0PS1
        bcf         T0CON,T0PS0
        movlw RPM_2500
        movwf AUX_RPM
;       movlw 0xFB
;       movwf LATB
        movlw 0x15
        call  endereco_lcd
        movlw '2'
        call  caracter_lcd
        movlw '5'
        call  caracter_lcd
        goto  saida_rpm_select

rpm_5:
        bcf         T0CON,T0PS2
        bsf         T0CON,T0PS1
        bcf         T0CON,T0PS0
        movlw RPM_3000
        movwf AUX_RPM
;       movlw 0xFA
;       movwf LATB
        movlw 0x15
        call  endereco_lcd
        movlw '3'
        call  caracter_lcd
        movlw '0'
        call  caracter_lcd
        goto  saida_rpm_select

rpm_6:
        movlw RPM_3500
        movwf AUX_RPM
;       movlw 0xF9
;       movwf LATB
        movlw 0x15
        call  endereco_lcd
        movlw '3'
        call  caracter_lcd
        movlw '5'
        call  caracter_lcd
        goto  saida_rpm_select
```

```
rpm_7:
        movlw RPM_4000
        movwf AUX_RPM
;       movlw 0xF8
;       movwf LATB
        movlw 0x15
        call  endereco_lcd
        movlw '4'
        call  caracter_lcd
        movlw '0'
        call  caracter_lcd
        goto  saida_rpm_select

rpm_8:
        movlw RPM_4500
        movwf AUX_RPM
;       movlw 0xF7
;       movwf LATB
        movlw 0x15
        call  endereco_lcd
        movlw '4'
        call  caracter_lcd
        movlw '5'
        call  caracter_lcd
        goto  saida_rpm_select

rpm_9:
        movlw RPM_5000
        movwf AUX_RPM
;       movlw 0xF6
;       movwf LATB
        movlw 0x15
        call  endereco_lcd
        movlw '5'
        call  caracter_lcd
        movlw '0'
        call  caracter_lcd
        goto  saida_rpm_select

rpm_10:
        movlw RPM_5500
        movwf AUX_RPM
;       movlw 0xF5
;       movwf LATB
        movlw 0x15
        call  endereco_lcd
        movlw '5'
        call  caracter_lcd
        movlw '5'
        call  caracter_lcd
        goto  saida_rpm_select

rpm_11:
        movlw RPM_6000
        movwf AUX_RPM
;       movlw 0xF4
;       movwf LATB
        movlw 0x15
```

```
        call  endereco_lcd
        movlw '6'
        call  caracter_lcd
        movlw '0'
        call  caracter_lcd
        goto  saida_rpm_select
saida_rpm_select:
        return


;---          LCD          ---
lcd_init:
;       ----  inicialização do lcd    ---

        C_RS
        C_EN
        C_RW

;1      ---          coando inicialização    ---

        S_D4
        S_D5
        C_D6
        C_D7

        movlw 0x0F          ;delay 15ms
        call  delay

        S_EN
        movlw 0x01
        call  delay
        C_EN
        movlw 0x01
        call  delay

;2      ---
        movlw 0x05
        call  delay

        S_EN
        movlw 0x01
        call  delay
        C_EN
        movlw 0x01
        call  delay

;3      ---

        movlw 0x05
        call  delay

        S_EN
        movlw 0x01
        call  delay
        C_EN
        movlw 0x01
        call  delay

;4      ---   envio 0010  nibble high
```

```
        C_D7
        C_D6
        S_D5
        C_D4

        S_EN
        movlw 0x01
        call  delay
        C_EN
        movlw 0x01
        call  delay

;5      ---
        movlw 0x28              ;4 bits, 2 linhas, 5x7
        call  comando_lcd

        movlw 0x06              ;incrementa, shift off
        call  comando_lcd

        movlw 0x0C              ;display on, cursor off, blink off
        call  comando_lcd

        movlw 0x01              ;     limpa display
        call  comando_lcd

        retlw 0x00

;---          COMANDO     LCD          ---
comando_lcd:
        movwf COMANDO

        C_EN
        C_RW
        C_RS

        movlw 0x01
        call  delay

        movf  COMANDO,W
        andlw 0x80
        btfss STATUS,Z
        S_D7
        btfsc STATUS,Z
        C_D7

        movf  COMANDO,W
        andlw 0x40
        btfss STATUS,Z
        S_D6
        btfsc STATUS,Z
        C_D6

        movf  COMANDO,W
        andlw 0x20
        btfss STATUS,Z
        S_D5
        btfsc STATUS,Z
        C_D5
```

```
        movf  COMANDO,W
        andlw 0x10
        btfss STATUS,Z
        S_D4
        btfsc STATUS,Z
        C_D4

        S_EN
        movlw 0x01
        call  delay
        C_EN
        movlw 0x01
        call  delay

        swapf COMANDO
        movf  COMANDO,W
        andlw 0x80
        btfss STATUS,Z
        S_D7
        btfsc STATUS,Z
        C_D7

        movf  COMANDO,W
        andlw 0x40
        btfss STATUS,Z
        S_D6
        btfsc STATUS,Z
        C_D6

        movf  COMANDO,W
        andlw 0x20
        btfss STATUS,Z
        S_D5
        btfsc STATUS,Z
        C_D5

        movf  COMANDO,W
        andlw 0x10
        btfss STATUS,Z
        S_D4
        btfsc STATUS,Z
        C_D4

        S_EN
        movlw 0x01
        call  delay
        C_EN
        movlw 0x01
        call  delay

        movf  COMANDO,w
        andlw 0x30
        btfsc STATUS,Z
        goto  saida_cmd
        movlw 0x01
        call  delay

saida_cmd:
```

```
        retlw 0x00

;---         CARACTERE   LCD         ---
caracter_lcd:
        movwf CARACTER

        C_RW
        S_RS
        C_EN

        movlw 0x01
        call  delay

        movf  CARACTER,W
        andlw 0x80
        btfss STATUS,Z
        S_D7
        btfsc STATUS,Z
        C_D7

        movf  CARACTER,W
        andlw 0x40
        btfss STATUS,Z
        S_D6
        btfsc STATUS,Z
        C_D6

        movf  CARACTER,W
        andlw 0x20
        btfss STATUS,Z
        S_D5
        btfsc STATUS,Z
        C_D5

        movf  CARACTER,W
        andlw 0x10
        btfss STATUS,Z
        S_D4
        btfsc STATUS,Z
        C_D4

        S_EN
        movlw 0x01
        call  delay
        C_EN
        movlw 0x01
        call  delay

        swapf CARACTER
        movf  CARACTER,W
        andlw 0x80
        btfss STATUS,Z
        S_D7
        btfsc STATUS,Z
        C_D7

        movf  CARACTER,W
        andlw 0x40
        btfss STATUS,Z
```

```
        S_D6
        btfsc STATUS,Z
        C_D6

        movf  CARACTER,W
        andlw 0x20
        btfss STATUS,Z
        S_D5
        btfsc STATUS,Z
        C_D5

        movf  CARACTER,W
        andlw 0x10
        btfss STATUS,Z
        S_D4
        btfsc STATUS,Z
        C_D4

        S_EN
        movlw 0x01
        call  delay
        C_EN
        movlw 0x01
        call  delay

        retlw 0x00
;---          POSIÇÃO     LCD          ---
endereco_lcd:
        movwf ENDERECO
        btfsc ENDERECO,4
        goto  linha_1
        goto  linha_2

linha_1:
        bcf         ENDERECO,7
        bcf         ENDERECO,6
        bcf         ENDERECO,5
        bcf         ENDERECO,4
        goto  sel_coluna

linha_2:
        bcf         ENDERECO,7
        bsf         ENDERECO,6
        bcf         ENDERECO,5
        bcf         ENDERECO,4

sel_coluna:
        C_RW
        C_RS
        C_EN

        movlw 0x01
        call  delay

        S_D7

        movf  ENDERECO,W
        andlw 0x40
        btfss STATUS,Z
```

```
S_D6
btfsc STATUS,Z
C_D6

movf  ENDERECO,W
andlw 0x20
btfss STATUS,Z
S_D5
btfsc STATUS,Z
C_D5

movf  ENDERECO,W
andlw 0x10
btfss STATUS,Z
S_D4
btfsc STATUS,Z
C_D4

S_EN
movlw 0x01
call  delay
C_EN
movlw 0x01
call  delay

swapf ENDERECO
movf  ENDERECO,W
andlw 0x80
btfss STATUS,Z
S_D7
btfsc STATUS,Z
C_D7

movf  ENDERECO,W
andlw 0x40
btfss STATUS,Z
S_D6
btfsc STATUS,Z
C_D6

movf  ENDERECO,W
andlw 0x20
btfss STATUS,Z
S_D5
btfsc STATUS,Z
C_D5

movf  ENDERECO,W
andlw 0x10
btfss STATUS,Z
S_D4
btfsc STATUS,Z
C_D4

S_EN
movlw 0x01
call  delay
C_EN
movlw 0x01
```

```
        call  delay

        retlw 0x00

;---         delay       ---
delay:
        movwf AUX_DELAY
delay_1:
        tstfsz      AUX_DELAY
        goto  delay_1
        retlw 0x00

;-------------------------------------------------------------------
-----
;                                DEBOUNCE BOTÕES
;-------------------------------------------------------------------
-----
trata_b1:
 btfss f_b1
 movff VALOR_DEBOUNCE,AUX_DEBOUNCE
 bsf         f_b1
 bsf         f_press
 btfss f_debounce
 goto  saida_trata_b1
 bcf         f_b1
 bcf         f_press
 bcf         f_debounce
 btfsc B1
 goto  saida_trata_b1
 tstfsz      RPM_VTR
 goto  dec_ok
 goto  saida_trata_b1
dec_ok:
 decf  RPM_VTR
 call  rpm_select
saida_trata_b1:
 return

trata_b2:
 btfss f_b2
 movff VALOR_DEBOUNCE,AUX_DEBOUNCE
 bsf         f_b2
 bsf         f_press
 btfss f_debounce
 goto  saida_trata_b2
 bcf         f_b2
 bcf         f_press
 bcf         f_debounce
 btfsc B2
 goto  saida_trata_b2
 movlw 0x00
 cpfsgt      RPM_VTR
 goto  saida_trata_b2
 clrf  RPM_VTR
 call  rpm_select
saida_trata_b2:
 return

trata_b3:
```

```
 btfss f_b3
 movff VALOR_DEBOUNCE,AUX_DEBOUNCE
 bsf         f_b3
 bsf         f_press
 btfss f_debounce
 goto  saida_trata_b3
 bcf         f_b3
 bcf         f_press
 bcf         f_debounce
 btfsc B3
 goto  saida_trata_b3
 movlw 0x0B
 cpfslt      RPM_VTR
 goto  saida_trata_b3
 incf  RPM_VTR
 call  rpm_select
saida_trata_b3:
 return

;==============================================================================
end
```

7.2. Apêndice B - Programa de sincronismo *assembly*

```
;==============================================================================
;Arquivo:          sincronismo.asm
;Autor:            Felipe Marotti Mochiuti
;Modificação:      11/06/2020 - Adicionando comentários
;==============================================================================

;==============================================================================
;                  CONFIGURAÇÕES
;------------------------------------------------------------------------------
LIST         p=18f4550
#include     "p18f4550.inc"
  CONFIG  PLLDIV   = 5              ; PLL: Divide por 5 para cristal de
20 MHz (A frequência de entrada deve ser de 4Mhz)
       CONFIG  CPUDIV   = OSC1_PLL2       ;    Configura    frequência    do
microcontrolador para 48Mhz
       CONFIG  FOSC           = HSPLL_HS        ; Configura cristal oscilador
como High Speed e habilita multiplicador de frequência PLL
       CONFIG  FCMEN          = OFF             ;  Desabilita  o  monitor  de
falha de frequência
       CONFIG  IESO           = OFF             ; Desabilita a troca de clock
interno/externo
       CONFIG  PWRT           = ON              ; Habilita Power-up, aguarda
em média 65,5 ms até que VCC se estabilize
       CONFIG  BOR            = OFF             ; Desabilita Brown-out Reset
       CONFIG  WDT            = OFF             ; Desabilita Watchdog Timer
       CONFIG  PBADEN   = OFF             ;  PORTB  configurado  como  portas
digitais
       CONFIG  MCLRE    = ON              ; Pino RE3 habilitado como MCLR
       CONFIG  CCP2MX   = ON              ; Habilita pino RC1 como entrada da
CCP2
;==============================================================================

;==============================================================================
;                        PAGINAÇÃO DE MEMÓRIA
```

```
;-----------------------------------------------------------------------
cblock       0x00
;--Variáveis para salvar contexto quando ocorrer alguma interrupção---
    W_TEMP_LP
    STATUS_TEMP_LP
    BSR_TEMP_LP
 W_TEMP_HP
 STATUS_TEMP_HP
 BSR_TEMP_HP
;-----------------------------------------------------------------------
    DENTE_ATUAL                                     ;Variável de posição da
roda fônica em relação ao sensor de rotação
    FLAG                                            ;Variável    utilizada
para flags do programa

    AUX_TMR0L                                       ;Variável     auxiliar
Timer0 do bit 0 ao 7
    AUX_TMR0H                                       ;Variável     auxiliar
Timer0 do bit 8 ao 15
endc
;=======================================================================

;=======================================================================
;                   VARIAVEIS
;-----------------------------------------------------------------------
#define      falha_roda        0x73        ;Intervalo da falha no dente
58
#define      pms_14_roda       0x28        ;Ponto     Morto     Superior
cilindros 1 e 4
;=======================================================================

;=======================================================================
;                     FLAGS
;-----------------------------------------------------------------------
#define      f_volta_1              FLAG,0      ;Indicador de volta do
ciclo Otto: 1 = 1ªvolta; 0 = 2ªvolta.
#define      f_carga_tmr0           FLAG,1      ;Indicador de carga do
timer0: 1 = timer0 carregado.
#define      f_dente_1              FLAG,2      ;Indicador do primeiro
dente da roda fônica: 1 = 1°dente
;=======================================================================

;=======================================================================
;               ENTRADAS E SAÍDAS
;-----------------------------------------------------------------------
#define      sinal_fase             LATB,7      ;Sinal     da     fase
espelhado no pino RB7
#define      sinal_fonica           LATB,5      ;Sinal  da  roda  fônica
espelhado no pino RB5
#define      sinal_virtual          LATB,3      ;Sinal da roda virtual
espelhado no pino RB3
;=======================================================================

;=======================================================================
;                                   VETOR RESET
;-----------------------------------------------------------------------
 org         0x0000
 goto  config_hw
;=======================================================================
```

```
;=====================================================================
;                          VETOR INTERRUPÇÕES
;---------------------------------------------------------------------
 org         0x0008
 movwf W_TEMP_HP                        ; salva w em W_TEMP_HP
 movff STATUS, STATUS_TEMP_HP       ; salva status em STATUS_TEMP_HP
 movff BSR, BSR_TEMP_HP             ; salva bsr em BSR_TMEP_HP
 bra         isr_hp                                 ;    vai    para
interrupção de alta prioridade
;---------------------------------------------------------------------
 org         0x0018
 movwf W_TEMP_LP                        ; salva w em W_TEMP_LP
 movff STATUS, STATUS_TEMP_LP       ; salva status em STATUS_TEMP_LP
 movff BSR, BSR_TEMP_LP             ; salva bsr em BSR_TEMP_LP
 bra         isr_lp                                 ;    vai    para
interrupção de baixa prioridade
;---------------------------------------------------------------------
;                                  PRIORIDADE BAIXA
;---------------------------------------------------------------------
isr_lp:
;---------------------------------------------------------------------
;                                          CCP1
;---------------------------------------------------------------------
 btfsc PIE1,CCP1IE                      ;  Verifica  se  CCP1  esta
habilitada
 btfss PIR1,CCP1IF                      ;   Verifica   se   ocorreu
interrupção da CCP1
 bra         saida_ccp1                         ;    Se    não    houve
interrupção vai para saida_ccp1
 btg         CCP1CON,0                          ;  Inverte  leitura  de
borda de subida para descida e vice-versa
 btfss f_dente_1                        ; Verifica se roda fônica já
passou pelo primeiro dente
 bra         dente_1                                ; Se não passou
pelo primeiro dente vai para dente_1
 bcf         STATUS,C                           ; Limpa o bit de CARRY
do STATUS antes de rotacionar o byte alto do Timer1 para direita
 rrcf  TMR1H,W                                  ; Rotaciona o byte alto
de Timer1 e armazena o resultado em W
 comf  WREG                             ;  Salva  o  complemento  do
valor de W em W
 movwf TMR0H                            ; Transfere o valor de W para
byte alto de Timer0
    movwf    AUX_TMR0H                          ; Transfere o valor de
W para AUX_TMR0H
 rrcf  TMR1L,W                                  ;  Rotaciona  o  byte
baixo do Timer1 com o valor atual de CARRY e armazena em W
 comf  WREG                             ;  Salva  o  complemento  do
valor de W em W
 movwf TMR0L                            ; Transfere o valor de W para
byte baixo de Timer0
 movwf AUX_TMR0L                        ; Transfere o valor de W para
AUX_TMR0L
 bsf         f_carga_tmr0                       ;  Habilita  a  Flag  de
carga do Timer0
 bcf         sinal_virtual                      ; Zera o valor do sinal
da roda digital para que esta comece o período em nível baixo
```

```
      bcf         INTCON,TMR0IF                    ;  Limpa  a  Flag  de
ocorrência de interrupção do Timer0
      bsf         INTCON,TMR0IE                    ; Habilita interrupção
do Timer0
     dente_1:
      bsf         f_dente_1                        ;  Habilita  Flag  de
passagem do primeiro ente da roda fônica
      clrf  TMR1H                            ; Limpa registrador TMR1H
      clrf  TMR1L                            ; Limpa registrador TMR1L
      btg         sinal_fonica                     ; Inverte sinal lógico
da roda fônica
      incf  DENTE_ATUAL                      ;  Incrementa  variável  de
dente da roda fônica
      movlw falha_roda                       ; Transfere 115 decimal para
W
      cpfsgt      DENTE_ATUAL                      ;  Compara  valor  de
DENTE_ATUAL com 115 e pula próxima linha se DENTE_ATUAL for maior
      bra         saida_ccp1_ok                    ; Se não for maior vai
para saida_ccp1_ok
      btg         f_volta_1                        ; Troca status de volta
1 para 2 e vice-versa
      clrf  DENTE_ATUAL                      ; Zera a variável de dente
atual
      bcf         f_carga_tmr0                     ; Limpa a Flag de carga
de Timer0
      bcf         f_dente_1                        ;  Limpa  a  Flag  de
passagem pelo dente 1
     saida_ccp1_ok:
      bcf         PIR1,CCP1IF                      ;  Limpa  a  Flag  de
ocorrência de interrução da CCP1
     saida_ccp1:
     ;-------------------------------------------------------------------
     ;                                        CCP2
     ;-------------------------------------------------------------------
      btfss PIR2,CCP2IF                      ;   Verifica   se   ocorreu
interrupção da CCP2
      bra         saida_ccp2                       ; Se não houve vai para
saida_ccp2
      btg         CCP2CON,0                        ;  Inverte  leitura  de
borda de subida para descida e vice-versa
      btg         sinal_fase                       ; Inverte estado lógico
do sinal de fase
      bsf         f_volta_1                        ; Habilita a Flag da 1ª
volta
      bcf         PIR2,CCP2IF                      ;  Limpa  a  Flag  de
ocorrência de interrupção da CCP2
     saida_ccp2:
      bra         saida_int                        ; vai para saida_int
     ;-------------------------------------------------------------------
     ;                                  PRIORIDADE ALTA
     ;-------------------------------------------------------------------
     isr_hp:
     ;-------------------------------------------------------------------
     ;            TIMER 0
     ;-------------------------------------------------------------------
      btg         sinal_virtual                    ; Inverte estado lógico
da roda virtual
      bcf         INTCON,TMR0IE                    ;           Desabilita
interrupção do Timer0
```

```
      btfsc f_carga_tmr0                        ;  Testa  Flag  de  carga  do
Timer0 e pula próxima linha se igual a zero
      bra         saida_timer_0                     ; Se Flag igual a 1 vai
para saida_timer_0
     digital_falha:
         movf     AUX_TMR0H,W                    ;  Transfere  AUX_TMR0H
para W
      movwf TMR0H                               ;  Transfere  W  para  o  byte
alto do Timer0
      movf  AUX_TMR0L,W                         ; Transfere AUX_TMR0L para W
      movwf TMR0L                               ;  Transfere  W  para  o  byte
baixo do Timer0
      bsf         INTCON,TMR0IE                 ; Habilita interrupção
Timer0
     saida_timer_0:
      bcf         f_carga_tmr0                  ; Limpa Flag de carga
do Timer0
      bcf         INTCON,TMR0IF                 ;    Limpa    Flag    de
ocorrência de interrupção do Timer0
     ;-----------------------------------------------------------------------
     ;                              SAIDA INTERRUPÇÃO
     ;-----------------------------------------------------------------------
     saida_int_hp:
      movff BSR_TEMP_HP, BSR              ; Recupera BSR
      movf  W_TEMP_HP, W                        ; Recupera WREG
      movff STATUS_TEMP_HP, STATUS        ; Recupera STATUS
      retfie                                            ;    Retorna    da
interrupção de alta prioridade
     saida_int:
      movff BSR_TEMP_LP, BSR              ; Recupera BSR
      movf  W_TEMP_LP, W                        ; Recupera WREG
      movff STATUS_TEMP_LP, STATUS        ; Recupera STATUS
      retfie                                            ;    Retorna    da
interrupção de baixa prioridade

     ;=======================================================================
     ;                         CONFIGURAÇÃO HARDWARE
     ;-----------------------------------------------------------------------
     config_hw:
     ;-----------------------------------------------------------------------
     ;                         CONFIGURAÇÃO DAS PORTAS DE ENTRADA E SAÍDA
     ;-----------------------------------------------------------------------
      movlw 0x0F
      movwf ADCON1                                  ;  Configura  todas  as
portas como digital
      movlw 0x00
      movwf TRISA                               ;   PORTA   configurado   como
saída
      movlw 0x00
      movwf TRISB                               ;   PORTB   configurado   como
saída
      movlw 0x06
      movwf TRISC                               ;  Bits  1  e  2  do  PORTC
configurado como entrada e os demais como saída
      movlw 0x00
      movwf TRISD                               ;   PORTD   configurado   como
saída
      movlw 0x00
      movwf TRISE                               ;   PORTE   configurado   como
```

```
saída
      setf  LATB                                ; Apaga os leds do PORTB
     ;-----------------------------------------------------------------
     ;                              CONFIGURAÇÃO INTERRUPÇÕES
     ;-----------------------------------------------------------------
      clrf  INTCON                                    ; Limpa o registrador
INTCON
      clrf  INTCON2                                   ; Limpa o registrador
INTCON2
      clrf  INTCON3                                   ; Limpa o registrador
INTCON3
      clrf  PIE1                                ; Limpa o registrador PIE1
      clrf  PIE2                                ; Limpa o registrador PIE2
      clrf  PIR1                                ; Limpa o registrador PIR1
      clrf  PIR2                                ; Limpa o registrador PIR2
      clrf  IPR1                                ; Limpa o registrador IPR1
      clrf  IPR2                                ; Limpa o registrador IPR2
      movlw 0xC0
      movwf INTCON                                    ;             Habilita
interrupções e periféricos
      bsf         RCON,IPEN                           ; Habilita prioridade
nas interrupções
     ;-----------------------------------------------------------------
     ;                                  CONFIGURAÇÃO TIMER 0
     ;-----------------------------------------------------------------
      movlw   0x88
      movwf   T0CON                                   ; Timer0 habilitado e
configurado com 16 bits e pré-escala 1:1
      clrf    TMR0L                                   ;   Limpa   byte   baixo
Timer0
      clrf    TMR0H                                   ;   Limpa   byte   alto
Timer0
      bcf       INTCON,TMR0IE                 ;   Interrupção   do   Timer0
desabilitada
      bsf         INTCON2,TMR0IP                    ;     Habilita     alta
prioridade para o Timer0
     ;-----------------------------------------------------------------
     ;                              CONFIGURAÇÃO TIMER 1
     ;-----------------------------------------------------------------
      movlw 0x01
      movwf T1CON                               ;    Timer1    habilitado    e
configurado com 16 bits e pré-escala 1:1
      clrf  TMR1L                               ; Limpa byte baixo Timer1
      clrf  TMR1H                               ; Limpa byte alto Timer1
      bcf         IPR1,TMR1IP                         ;     Habilita     baixa
prioridade para o Timer1
      bcf         PIE1,TMR1IE                         ;            Desabilita
interrupção do Timer1
     ;-----------------------------------------------------------------
     ;                                  CONFIGURAÇÃO CCP1
     ;-----------------------------------------------------------------
      movlw 0x05
      movwf CCP1CON                                   ; Habilita captura de
toda borda de subida CCP1
      movlw 0x04
      movwf T3CON                               ; Configura Timer1 como clock
da CCP1
      bcf         IPR1,CCP1IP                         ;     Habilita     baixa
prioridade para o CCP1
```

```
      bcf         PIE1,CCP1IE                         ;            Desabilita
interrupção da CCP1
     ;-----------------------------------------------------------------------
     ;                                   CONFIGURAÇÃO CCP2
     ;-----------------------------------------------------------------------
     ;CAPTURA: CADA BORDA DE SUBIDA
     ;TIMER: SEM
      movlw 0x05
      movwf CCP2CON                                  ; Habilita captura de
toda borda de subida CCP2
      bcf         IPR2,CCP2IP                        ;    Habilita    baixa
prioridade para a CCP2
      bsf         PIE2,CCP2IE                        ; Habilita interrupção
da CCP2
     ;=======================================================================

     ;=======================================================================
     ;                              PROGRAMA PRINCIPAL
     ;-----------------------------------------------------------------------
     inicio:
      movlw pms_14_roda                          ; Transfere 40 decimal para W
      movwf DENTE_ATUAL                          ; Transfere valor de W para
variável DENTE_ATUAL
      clrf  FLAG                                 ; Limpa todas as Flags
      movlw 0x57
      movwf LATB                                 ; Acende Leds RB7, RB5 e RB3
     sincroniza:
      btfss   f_volta_1                          ; Testa Flag do PMS cilindro
1 e 2 pula próxima linha se igual a um
      bra         sincroniza                          ; Se Flag igual a zero
volta para sincroniza
      bcf         PIR1,CCP1IF                         ;    Limpa    Flag    de
ocorrência de interrupção da CCP1
      clrf  TMR1                                 ; Limpa Timer1
      bsf     PIE1,CCP1IE                             ;  Habilita  interrução
da CCP1
     loop:
      nop
      bra   loop                                      ; Loop infinito
     ;=======================================================================

     end
```

7.3. Apêndice C - Programa de sincronismo linguagem C

```
     #include "config.h"
     /*****************************DEFINIÇÕES*****************************/
     #define      falha_roda          0x73            // Posição da falha da
roda fônica
     #define      pms_14_roda         0x28             // Posição da fase do
comando de válvulas
     #define     fase                 LATBbits.LB7     // Sinal espelhado do
sensor de fase
     #define      fonica              LATBbits.LB5     // Sinal espelhado do
sensor de rotação
     #define      digital              LATBbits.LB3     // Sinal gerado pela
roda virtual
     /*********************************************************************/
```

```
    /*****************************VARIÁVEIS*******************************/
    unsigned char   DENTE_ATUAL         =   0;      // Variável de posição
do dente da roda fônica
    unsigned int    AUX_TMR0            =   0;      // Variável auxiliar do
Timer0
    unsigned char   f_volta_1           =   0;      // Flag indicadora de
volta
    unsigned char   f_dente_1           =   0;      // Flag indicadora do
dente um da roda fônica
    unsigned char   f_carga_tmr0        =   0;      // Flag indicadora de
carga do Timer0
    /*********************************************************************/

    /***********************PROTÓTIPO DE FUNÇÕES**************************/
    void config_HW(void);       // Protótipo de função de configuração do
microcontrolador
    /*********************************************************************/

    /*************************INTERRUPÇÕES********************************/
    void high_priority interrupt ISR_High(void)             // Rotina de
interrupção de alta prioridade
    {
        if((INTCONbits.TMR0IE)&&(INTCONbits.TMR0IF))      // Verifica se o
Timer0 está habilitado e se ocorreu interrupção
        {
            digital                 =   ~digital;       // Inverte o nível
lógico do sinal de saída da roda virtual
            INTCONbits.TMR0IE       =   0;                  // Desabilita
interrupção do Timer0
            if(f_carga_tmr0 == 0)                           // Verifica se a
Flag da carga do Timer0 está carregada
            {                                               // Se a Flag não
estiver carregada executa entre chaves
                TMR0                =   AUX_TMR0;       // Zera registrador
do Timer0
                INTCONbits.TMR0IE   =   0x01;         // Habilita interrupção
do Timer0
            }
            f_carga_tmr0            =   0x00;           // Limpa Flag de
carga do Timer0
            INTCONbits.TMR0IF       =   0x00;           // Limpa Flag de
ocorrência de interrupção do Timer0
        }
    }
    void low_priority interrupt ISR_Low(void)               // Rotina de
interrupção de baixa prioridade
    {
        if((PIE1bits.CCP1IE)&&(PIR1bits.CCP1IF))          // Verifica se a
CCP1 está habilitada e se ocorreu interrupção
        {
            LATBbits.LATB0  =   0x00;
            CCP1CONbits.CCP1M0      =   ~CCP1CONbits.CCP1M0;    // Inverte
leitura de borda de subida para descida ou vice versa
            if(f_dente_1    ==  1)                      // Verifica Flag de
primeira interrupção após a falha da roda fônica
            {                                               // Se não for a
primeira interrupção após a falha executa entre chaves
                AUX_TMR0            =   ~(TMR1/2);      // Divide por dois
```

```
o valor de Timer1 e salva o complememto desse valor na variável auxiliar de
Timer0
                TMR0                =   AUX_TMR0;       // Transfere o valor
da variável auxiliar de Timer0 para Timer0
                f_carga_tmr0         =    0x01;             // Carrega Flag
indicadora de carga do Timer0
                digital              =    0x00;            // Coloca sinal da
roda virtual em nível lógico baixo
                INTCONbits.TMR0IF    =    0x00;             // Limpa Flag de
ocorrência de interrupção do Timer0
                INTCONbits.TMR0IE   =   0x01;         // Habilita interrupção
do Timer0
            }
            f_dente_1               =    0x01;           // Carrega Flag de
primeira interrupção após a falha da roda fônica
            TMR1                    =    0x00;           // Zera registrador
do Timer1
            fonica                   =    ~fonica;         // Inverte estado
lógico do sinal da Roda fônica
            DENTE_ATUAL++;                                  // Incrementa
variável de posição da roda fônica
            if(DENTE_ATUAL >  falha_roda)               // Compara variável
de posição da roda fônica com valor de referência da falha
            {                                           // Se variável maior
que valor de referência executa entre chaves
                f_volta_1           =   ~f_volta_1;     // Inverte indicador
de volta 1 para volta 2 e vice versa
                DENTE_ATUAL         =   0x00;           // Zera variável de
posição da roda fônica
                f_carga_tmr0         =    0x00;             // Limpa Flag
indicadora de carga do Timer0
                f_dente_1            =    0x00;             // Limpa Flag
indicadora de primeira interrupção após falha da roda fônica
            }
            PIR1bits.CCP1IF          =    0x00;            // Limpa Flag de
ocorrência de interrupção da CCP1
        }
        if((PIE2bits.CCP2IE)&&(PIR2bits.CCP2IF))          // Verifica se a
CCP@ está habilitada e se ocorreu interrupção
        {
            CCP2CONbits.CCP2M0      =    ~CCP2CONbits.CCP2M0;    // Inverte
leitura de borda de subida para descida ou vice versa
            fase                    =    ~fase;                  // Inverte
estado lógico do sinal de fase do comando de válvulas
            f_volta_1               =   0x01;                 // Habilita
Flag indicadora de primeira volta d ciclo Otto
            PIR2bits.CCP2IF          =    0x00;                  // Limpa
Flag de ocorrência de interrupção da CCP2
        }
        LATBbits.LATB0  =   0x01;
    }
    ///*****************************************************************/

    /*********************CONFIGURAÇÃO DO HARDWARE************************/
    void  config_HW(void)                   //  Função  de  configuração  do
microcontroldor
        {
        //                  ENTRADAS E SAÍDAS
     ADCON1   =   0x0F;   // Configura todas portas de entrada como digitais
```

```
      TRISA   =   0x00;   // Configura PORTA como saída
      TRISB =   0x00;   // Configura PORTB como saída
      TRISC =   0x06;   // Configura RC1 e RC2 como entrada e as demais portas
do PORTC como saída
      TRISD =   0x00;   // Configura PORTD como saída
      TRISE =   0x00;   // Configura PORTE como saída
      LATB    =   0xFF;   // Apaga Leds do PORTB
      //                      CONFIGURAÇÃO INTERRUPÇÕES
      INTCON      =   0xC0;   // Habilita interrupções e periféricos
      INTCON2     =   0x00;   // Zera registrador INTCON2
      INTCON3     =   0x00;   // Zera registrador INTCON3
      PIE1        =   0x00;   // Zera registrador PIE1
      PIE2        =   0x00;   // Zera registrador PIE2
      PIR1        =   0x00;   // Zera registrador PIR1
      PIR2        =   0x00;   // Zera registrador PIR2
      IPR1        =   0x00;   // Zera registrador IPR1
      IPR2        =   0x00;   // Zera registrador IPR2
      IPEN        =   0x01;  // Habilita prioridade de interrupções
         //                                  CONFIGURAÇÃO TIMER 0

      T0CON   =   0x88;       // Configura Timer0 com 16 bits e pré-escala
de 1:1
      TMR0    =   0x00;       // Zera registrador do Timer0
      TMR0IE  =   0x00;       // Limpa Flag de ocorrência do Timer0
      TMR0IP  =   0x01;       // Habilita prioridade alta para interrupção
do Timer0
         //                                CONFIGURAÇÃO TIMER 1
      T1CON   =   0x01;       // Configura Timer1 com 16 bits e pré-escala
de 1:1
      TMR1L   =   0x00;       // Zera registrador do Timer1
      TMR1IP  =   0x00;       // Habilita prioridade baixa para interrupção
do Timer1
      TMR1IE  =   0x00;       // Desabilita Interrupção do Timer1
         //                                  CONFIGURAÇÃO CCP1

      CCP1CON     =   0x05;   // Configura modo de captura de toda borda de
subida da CCP1
      T3CON       =   0x08;   // Configura Timer1 como base de tempo da CCP1
      CCP1IP      =   0x00;   // Habilita prioridade baixa para interrupção
do CCP1
      CCP1IE      =   0x00;   // Desabilita interrupção da CCP1
         //                                  CONFIGURAÇÃO CCP2
      CCP2CON     =   0x05;   // Configura modo de captura de toda borda de
subida da CCP2
      CCP2IP      =   0x00;   // Habilita prioridade baixa para interrupção
do CCP2
      CCP2IE      =   0x01;   // Habilita interrupção da CCP2
     }
     /*****************************************************************/

     /**********************PROGRAMA PRINCIPAL*****************************/
     void main(void)
     {
         config_HW();                                // Executa função para
configurar microcontrolador
         DENTE_ATUAL          =   pms_14_roda;     // Transfere valor de PMS
para dente atual
         LATB                 =   0x57;           // Acende leds do RB7, RB5
e RB3
```

```
        while(f_volta_1 == 0){}                    // Aguarda primeiro sinal
de fase para sincronismo
        PIR1bits.CCP1IF     =    0x00;            // Limpa Flag de ocorrência
de interrupção da CCP1
        PIE1bits.CCP1IE     =    0x01;            // Habilita interrupção da
CCP1
        TMR1                =    0x00;           // Zera registrador do Timer1
        while(1)                                  // Loop infinito
        {

        }
    }
    /*****************************************************************/
```