Fábio Dassan dos Santos

Utilização de DSPs e FPGAs em Unidades Eletrônicas Automotivas

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

Curso de Engenharia de Computação com
ênfase em Sistemas Embarcados

ORIENTADOR: Prof. Dr. Carlos Dias Maciel

São Carlos
2008

Aos meus pais.

Agradecimentos

A Deus, por me permitir chegar até esta etapa tão importante da minha vida.

A meus pais, grandes companheiros e exemplos, e que nunca deixaram que eu desanimasse.

Aos meus irmãos, que mesmo não me entendendo às vezes, sempre me acolhiam com carinho.

Aos meus companheiros de turma, aos mais próximos e aos mais distantes – todos tiveram sua contribuição na formação de futuro engenheiro que tenho.

Aos amigos com quem morei na faculdade, Elias Arbex e Breno Pelizer, Anderson Maia e Guilherme Buzo, por serem tão humanamente fortes.

Aos tantos outros amigos da moradia estudantil e de minha cidade natal, que sempre estiveram comigo quando precisei.

Ao meu orientador, professor Dr. Carlos Maciel, por ser mais que apenas um docente, mas um amigo presente.

Sumário

Lista de figuras

Resumo

A demanda por sistemas embarcados tem crescido significativamente nos últimos anos. Conseqüentemente, aumentou o interesse por dispositivos que ofereçam estabilidade, segurança e confiabilidade no desenvolvimento destes projetos. Especificamente na área automotiva, a rigorosidade das leis ambientais e a busca por sistemas que associem consumo baixo com desempenho satisfatório implicaram na evolução dos sistemas de controle. O ambiente agressivo no qual estes sistemas de controle são inseridos sugere uma necessidade grande de pesquisas no sentido de buscar uma alternativa eficiente e de baixo custo. Esta possível solução deve ser capaz de atuar de maneira tolerante a falha dentro do sistema automotivo, oferecendo um tempo de resposta bastante baixo, característica dos sistemas chamados de tempo real. Este trabalho tem como objetivo implementar um circuito de detecção de ciclos de motor, analisando a forma de onda gerada por este, de maneira a atuar através de um sinal de controle enviado a uma unidade injetora de combustível a partir da constatação do estado de funcionamento do motor. Para a análise do ciclo de rotação do motor, foram feitas duas implementações, em DSP e FPGA, com o intuito de verificar quais os pontos favoráveis e desfavoráveis de cada dispositivo dentro desta aplicação específica.

Abstract

Demand for embedded systems has grown significantly in recent years. Therefore, the interest in devices that provide stability, security and reliability in the development of these projects increased. Specifically in the automotive area, the environmental laws and the search for systems involving consumption down with satisfactory performance involved in the evolution of systems of control. The aggressive environment in which these control systems are inserted suggests a great need for research to seek an efficient and low cost alternative. This possible solution must be able to be fault-tolerant system in the automotive, offering a very low time response, so called real time. This work intend to implement a circuit for detecting the engine cycles, examining the waveform generated by this in order to send a signal to a control unit fuel injector from the observation of the state of the engine. For the analysis of the engine cycle of rotation, two deployments were made in DSP and FPGA in order to find the favorable and unfavorable points of each device within this specific application.

1. Introdução

1.1. Motivação

A demanda por sistemas embarcados tem crescido significativamente nos últimos anos. Conseqüentemente, aumentou o interesse por dispositivos que ofereçam estabilidade, segurança e confiabilidade no desenvolvimento destes projetos. Especificamente na área automotiva, a rigorosidade das leis ambientais e a busca por sistemas que associem consumo baixo com desempenho satisfatório implicaram na evolução dos sistemas de controle.

Além disso, o ambiente agressivo no qual estes sistemas de controle são inseridos (sujeitos a interferências de temperatura e vibração, entre outras) sugere uma necessidade grande de pesquisas no sentido de buscar uma alternativa eficiente e de baixo custo. Esta possível solução deve ser capaz de atuar de maneira tolerante a falha dentro do sistema automotivo, oferecendo um tempo de resposta bastante baixo, característica dos sistemas chamados de tempo real.

Inicialmente mecânicos, atualmente estes sistemas são implementados eletronicamente, utilizando dispositivos como microprocessadores e microcontroladores. Em meados dos anos 80, uma unidade de controle híbrida utilizava técnicas de análise analógica e digital de sinais. Estas técnicas eram utilizadas para medir parâmetros de entrada do motor, compará-los com informações armazenadas em tabelas digitais para então gerarem saídas pré-determinadas.

Mais tarde, os sistemas passaram a computar as saídas dinamicamente. Atualmente, os controles são tão sofisticados a ponto de receberem informações de várias partes do motor, tomar decisões e atuar sobre eles. Por exemplo, tanto o controle da aceleração de um veículo quanto do antitravamento das rodas numa frenagem podem ser gerenciados por unidades de controle.

As aplicações deste tipo de sistema em outras áreas que não a automotiva também servem como motivação para pesquisa. Alguns equipamentos agrícolas de fertilização e controle de agrotóxicos, por exemplo, utilizam o mesmo princípio de funcionamento de uma unidade de controle automotiva. Estas máquinas devem controlar a quantidade de produto químico que deve ser pulverizado por área, utilizando como referência a rotação do motor que as impulsiona, e o tempo que levam para realizar uma volta completa do eixo de transmissão.

1.2. Objetivos

A partir da constatação da crescente demanda por sistemas embarcados de controle de motores (especialmente os sistemas automotivos), este trabalho tem como objetivo implementar um circuito de detecção de ciclos de motor, analisando a forma de onda gerada por este, de maneira a atuar através de um sinal de controle enviado a uma unidade injetora de combustível a partir da constatação do estado de funcionamento do motor.

Este ambiente pode ser encontrado no contexto das Unidades de Controle Automotivas, também conhecidas como ECUs (do inglês *Engine Control Unit*). Por isso, além da análise deste sinal de rotação do motor, foi também implementado um pequeno circuito que descreve de maneira não-linear a aceleração de um veículo. A partir desta, pode-se obter um valor de rotação.

Para a implementação destes sistemas, realizou-se uma revisão bibliográfica sobre Unidades de Controle Automotivas, desde sua concepção até os dias atuais. Foi estudado seu princípio de funcionamento, e suas interações com outros subsistemas que existem em um veículo.

Para a análise do ciclo de rotação do motor, foram feitas duas implementações, em DSP e FPGA, com o intuito de verificar quais os pontos favoráveis e desfavoráveis de cada dispositivo dentro desta aplicação específica. Esta constatação leva em conta principalmente as facilidades e dificuldades encontradas pelo aluno durante a execução do projeto de conclusão de curso.

1.3. Organização da monografia

O segundo capítulo apresenta uma explicação dos conceitos teóricos relacionados aos sistemas de controle automotivos, além do histórico e dos princípios de funcionamento dos dispositivos considerados neste trabalho.

O terceiro capítulo apresenta os *softwares* utilizados para a construção do sistema proposto, assim como sua modelagem propriamente dita. Além disso, relata os pontos implementados sobre cada tecnologia, apresentando gráficos com demonstrações das atividades.

O quarto capítulo é reservado para a discussão sobre os resultados obtidos, além de sugestões de trabalhos futuros. No quinto capítulo tecem-se as conclusões deste projeto, seguidas pelas referências bibliográficas e pela bibliografia consultada.

No fim, encontra-se o anexo com parte dos códigos utilizados na elaboração deste projeto.

2. Fundamentação Teórica

2.1. Controle Eletrônico Automotivo

2.1.1. Evolução histórica

A aplicação dos conceitos de eletrônica em sistemas automotivos era um processo natural, e até mesmo inevitável. O constante aumento do preço dos combustíveis associado às práticas empregadas para a diminuição da poluição gerada pelos veículos certamente aceleraram o processo. Entretanto, antes mesmo da produção dos sistemas de controle modernos, já existiam dispositivos digitais que atuavam nos motores dos automóveis [1].

O primeiro controle eletrônico aplicado a um motor automotivo surgiu em 1978, e foi chamado de carburador de "ciclo fechado" (tradução livre de *closed loop*) [1]. Foi uma resposta à crise mundial do petróleo, além de representar a primeira ação direta no combate à emissão de gases poluentes na atmosfera.

Não demorou para que este dispositivo evoluísse. No ano seguinte houve a produção em série do primeiro dispositivo de controle automotivo de natureza puramente eletrônica [2]. Antes dele, outros dispositivos já atuavam no motor com o intuito de controlá-lo, porém eram mistos – parte mecânicos, parte eletrônicos [3].

Já era de conhecimento da ciência que o resultado da combustão (tanto a energia quanto a produção de gases) era diretamente influenciado pela precisão na mistura ar-combustível. Para se obter o máximo do processo, deveria obedecer-se a razão estequiométrica de 14:1. Além disso, a faísca gerada para causar a explosão deveria ocorrer no instante exato desta proporção. Obviamente, estes fatores dependiam de outras variáveis tais como a velocidade, a carga de trabalho, temperatura [1].

A partir de então, começou-se a avaliar quais eram as informações mais importantes que uma unidade de controle deveria ter para que pudesse executar um trabalho eficiente. Basicamente, era importante saber a rotação do motor, as posições do *crankshaft* (conhecido como virabrequim) e *camshaft* (eixo responsável por acionar o movimento de subida e descida do pistão), e a massa de ar admitida. A posição do acelerador e a razão de aceleração (para o sistema de transmissão) também foram consideradas.

Até então, eram utilizados microprocessadores de 8 bits para o controle das funções básicas do motor (determinação da razão ar-combustível, temporização da ignição) [4] feito a partir das informações acima citadas.

Com o tempo, vários sensores e atuadores foram inseridos nos motores, em subsistemas específicos, tais como o de injeção de combustível e o de controle do tempo de explosão do motor (controle da faísca). Curiosamente, chegou-se à conclusão de que a maior parte dos sinais adquiridos do motor necessita apenas de quatro amostras a cada 360º. Ou seja, a cada giro do motor são realizadas quatro medições simétricas sendo que, a partir delas, pode-se calcular o comportamento do sinal durante todo o período matematicamente, prevendo não somente a posição do motor, mas também se este está acelerando ou desacelerando [1].

Conforme a necessidade surgia, apareciam também os equipamentos que possibilitavam o avanço na construção das unidades eletrônicas de controle. Entre os anos de 1980 e 1982 surgiram elementos muito importantes para a aquisição de sinais do motor: sensores de rotação fotoelétricos, sensores de massa de ar, sensores de medição de rotação, e o controle de regime de trabalho livre do motor [2].

Posteriormente, conforme os estudos avançavam, outras variáveis foram identificadas e a obtenção de seus valores se fez necessária. A temperatura da água (para identificar a temperatura do motor), a quantidade de combustível injetado proporcionalmente à velocidade desenvolvida, a densidade do ar (para o cálculo da razão estequiométrica) foram algumas delas [1].

No início da década de 90, começaram a surgir os primeiros dispositivos de controle com microprocessadores de 16 bits, para atender à demanda de tratamento dos novos sinais que eram agregados ao controle automotivo. Durante os anos seguintes, foram desenvolvidos modelos cada vez mais complexos e eficientes até que, graças ao avanço na área de *hardware*, foi possível utilizar processadores de 32 bits. Estes têm uma capacidade muito grande de processamento, permitindo executar algoritmos sofisticados de tratamento de informações e controle de sistemas [4].

Atualmente, as unidades eletrônicas de controle automotivo estão bastante avançadas, e são capazes de executarem funções consideradas complexas para um dispositivo embarcado, como interface com o usuário, por exemplo [1]. Além disso, são capazes de identificar configurações pessoais de cada usuário, como a utilização de transmissão de marchas automática ou manual, ou mesmo a função de pilotagem automática (na realidade, apenas o controle automático da aceleração).

2.1.2. Princípios básicos

Basicamente, uma unidade eletrônica de controle é constituída de alguns elementos principais. O mais importante deles é o controlador, responsável pela tomada de decisões sobre os estados de funcionamento do motor. Ele trabalha com sinais digitais. Sensores de dados (tensão, temperatura, velocidade angular e linear, entre outros) devem ser convertidos de informação contínua ou analógica para o formato digital exigido por microcontroladores. Esta conversão pode ou não ser realizada pelo próprio controlador, dependendo das funcionalidades implementadas nele.

Outra característica crítica é a capacidade de captura e tratamento de eventos assim que eles acontecem. Nesse ponto, faz diferença a velocidade na qual o controlador consegue trabalhar com os sinais de entrada e saída de informação. Com todas as informações disponíveis, é possível identificar o estado de funcionamento do motor e, a partir dele, tomar uma decisão sobre como atuar nos subsistemas [1].

O sistema de controle da maioria dos motores utilizados atualmente é dividido em um conjunto de subsistemas. Estes interagem entre si e com os sensores e atuadores, auxiliando a unidade de controle a tomar a melhor decisão sobre a operação do motor em um dado instante de análise [5].

O principal destes é o de injeção de combustível, responsável por controlar a quantidade ideal de combustível para determinada condição de operação do motor. Além deste, existem os sistemas de controle da ignição, recirculação de gases e outros que variam de acordo com cada fabricante.

A precisão no controle destes sistemas visa atingir o ponto ideal de funcionamento do motor. Neste, o consumo de combustível é minimizado, assim como a emissão de poluentes. Estas duas metas têm sido fortemente buscadas pelos projetistas, pressionados principalmente por três fatores: a natural busca por minimização de custos; a alta nos preços dos combustíveis nos últimos 30 anos; e pelas rigorosas leis ambientais relacionadas à emissão de gases na atmosfera por veículos automotores.

O sistema de controle interpreta os sinais recebidos e, através de valores contidos em tabelas, conclui sobre uma determinada ação a ser desempenhada pelo motor. Estas tabelas são obtidas em laboratório através de um processo chamado calibração. A partir de medições obtêm-se curvas de torque do motor, potência, consumo específico e níveis de emissões, e são construídas tabelas (como carga *versus* rotação *versus* ponto de ignição, carga *versus* rotação *versus* tempo de ignição, temperatura do motor *versus* tempo de injeção, entre outras). Estas são armazenadas na memória interna da unidade de controle, e são recuperadas ponto a ponto de acordo com a condição de operação do motor [6].

Para cada condição do motor, define-se um modo de controle. Cada modo corresponde a uma rotina realizada pelo programa, e que é ativada a partir dos sinais recebidos dos sensores. Apesar do sistema de gerenciamento atuar de maneira integrada, os módulos de controle são representados separadamente para fins didáticos, seguindo a literatura da área, conforme indicado na Figura 1.

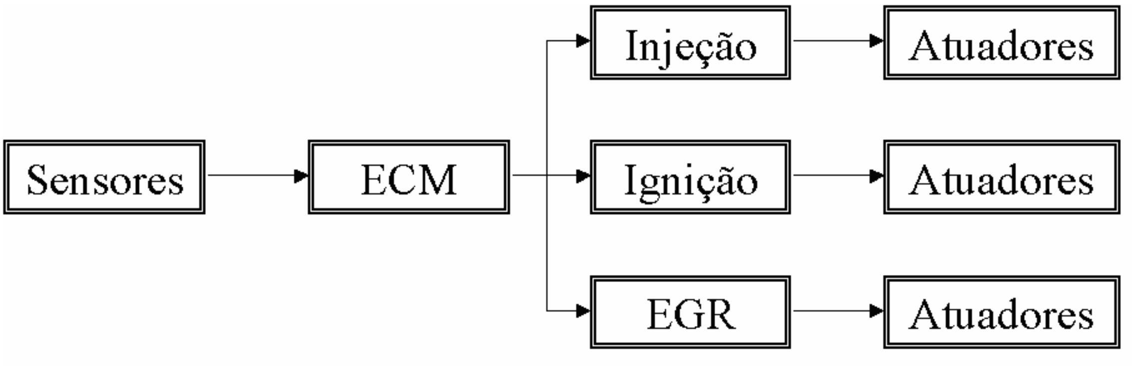

Figura 1: Divisão didática de um sistema de controle automotivo.

ECM = Engine Control Module, ou Módulo de Controle do Motor;
EGR = Exhaust Gas Recirculation, ou Controle de Recirculação de Gases.

A ECU deve possuir uma interface de comunicação com os sensores e atuadores, incluindo aí todas as questões relacionadas a protocolos e *drivers*. Um dos protocolos de comunicação mais utilizados na indústria automobilística é o *Controller Area Network* (CAN). Ele foi desenvolvido pela Bosch em 1986 para resolver problemas de comunicação entre dispositivos eletrônicos em automóveis [7].

Quase a totalidade das empresas de automóveis utiliza este padrão para comunicação entre os sensores e atuadores localizados no veículo e as unidades eletrônicas de controle. Também é utilizado na comunicação entre ECUs. Entre as razões para tal fato estão sua segurança e o custo baixo de implementação.

2.1.3. Descrição dos subsistemas de controle do motor

2.1.3.1. Controle eletrônico da ignição

O controle eletrônico de ignição trabalha a partir do mapa de avanço da ignição do motor. Uma vez detectada a condição de operação, as informações armazenadas em tabelas na memória da unidade de controle são recuperadas, para corrigir o ponto de ignição em função de alguns fatores. Dentre eles estão a rotação do motor, a pressão no coletor de admissão e a temperatura do motor.

Além de corrigir o ponto de ignição, a unidade eletrônica controla a ocorrência de *knocking*, de modo a atrasar o ponto de ignição quando o *knock* aparece. *Knocking* é o termo utilizado para descrever a combustão com características muito próximas à combustão detonante, quando comparado com o processo normal de combustão. Este fenômeno pode causar danos ao motor dependendo de sua intensidade e ocorrência.

2.1.3.2. Controle da recirculação dos gases de escape (EGR)

O sistema de recirculação de gases de escape tem por função desviar uma parte dos gases queimados da tubulação de exaustão de volta para a admissão do motor. O principal intuito desta medida é diminuir a emissão de gases na atmosfera, especialmente os que possuem nitrogênio em sua composição.

A quantidade de gás recirculado para a admissão varia em função da rotação do motor, pressão no coletor de admissão e temperatura do motor [8]. Estas informações são avaliadas pela unidade de controle, que atua de acordo com as condições momentâneas de funcionamento.

2.1.3.3. Sensores

Os sensores são responsáveis por obter as condições de funcionamento do motor em um determinado instante, e enviá-las à unidade de controle. Existem diversos sensores espalhados pelo motor, com o intuito de levantar o máximo de informações possível. Estas auxiliam nas decisões sobre qual ação tomar com relação a determinado estado de funcionamento. Entre todos, pode-se destacar como principais:

- **Sensores de pressão no coletor de admissão** - têm a função de informar as variações de pressão no coletor de admissão. Em alguns casos, esta pressão é utilizada para determinar qual a carga de trabalho na qual o motor se encontra, definindo o avanço da ignição;
- **Sensores mássicos** - são responsáveis pela medida da massa de ar admitida pelo motor. Outra maneira de fazer isso é a utilização de sensores volumétricos, que medem o fluxo volumétrico de ar;

- **Sensores de posição da borboleta de aceleração** - informam a posição angular da borboleta de aceleração à unidade eletrônica. Isto permite adotar estratégias de controle de liberação de combustível e momento de detonação da centelha de acordo com as tabelas armazenadas em sua memória;
- **Sensores de temperatura** - responsáveis por informar a temperatura do ar aspirado pelo motor e da água do sistema de arrefecimento. A temperatura do ar é necessária para se determinar sua densidade, utilizada para o cálculo da massa de ar que está sendo admitida pelo motor. A temperatura da água é utilizada como indicativo da temperatura do motor, servindo como parâmetro para que estratégias específicas possam ser realizadas, tais como:
    - Enriquecimento da mistura ar-combustível no momento da partida, quando o motor ainda está frio;
    - *Cut-off* com o motor frio (diminuição ou corte da injeção de combustível quando o carro não está acelerado);
    - Substituição do sensor de temperatura do ar, caso este não seja empregado;
    - Sensor de rotação do motor/PMS - tem por finalidade gerar o sinal de rotação do motor, e a posição da árvore de manivelas;
    - Sensor de fase - combinado com o sinal de rotação, permite que a unidade de controle identifique o cilindro em ignição.

2.1.3.4. Atuadores

Atuadores são todos os componentes do sistema de controle responsáveis por gerar uma ação sobre a planta – no caso, motores de combustão interna –, a partir de um sinal de controle. Nos sistemas de injeção eletrônica, este sinal é de natureza elétrica, resultado do processamento realizado pela unidade de controle.

Dentre os principais atuadores, pode-se destacar:

- **Válvulas injetoras de combustível** - dispositivos dosadores de combustível. Outros componentes podem realizar sua função (antigamente, essa função era exercida por carburadores mecânicos);
- **Bobina de ignição** - responsável por gerar a alta tensão requerida para provocar o centelhamento da vela de ignição. A centelha inicia o processo de combustão da mistura ar-combustível;

- **Corretor da marcha lenta** – tem como objetivo manter a rotação do motor o mais estável possível, quando o pedal do acelerador não está acionado e a rotação do motor é baixa.

2.1.3.5. Sensor lambda

A realimentação da malha fechada do sistema é feita pela sonda *lambda*, trabalhando em parceria com o conversor catalítico (dispositivo usado para reduzir a toxicidade das emissões dos gases de escape de um motor de combustão interna). Hoje, este conversor é o método mais eficiente de purificação dos gases de exaustão dos motores de combustão interna. Operando juntos, os sistemas de ignição e injeção permitem obter níveis muito baixos de emissão de gases poluentes. Com a utilização de um catalisador, estes níveis podem ser realmente bastante baixos, pois um catalisador (de três níveis) tem o poder de reduzir os índices de emissão dos gases prejudiciais em até 90%.

Este número só pode ser alcançado se o motor operar muito próximo da proporção estequiométrica ideal de funcionamento ($\lambda = 1 \pm 0.05$). Este pequeno desvio só pode ser mantido com o auxílio de sistemas de injeção de combustível controlados eletronicamente. Por essa razão, utiliza-se o controle em malha fechada com sonda *lambda*, ou seja, a composição da mistura ar-combustível é mantida dentro da faixa ótima através de ações de controle.

Em outras palavras, a sonda *lambda* funciona como um sensor de realimentação que indica se a mistura está acima ou abaixo da proporção estequiométrica [6].

Este e todos os outros subsistemas descritos podem ser vistos esquematicamente na figura 2.

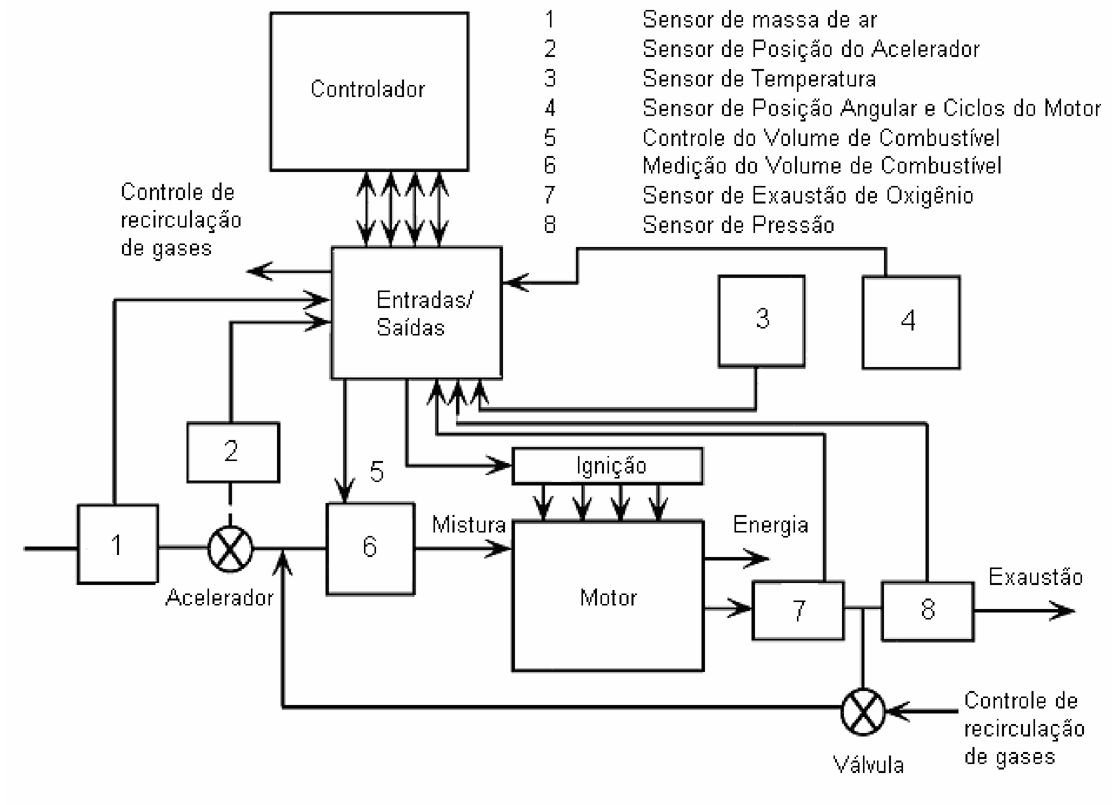

Figura 2: Diagrama de Blocos de uma ECU [6]

2.1.4. Sistema de Injeção de Combustível

O propósito do sistema de injeção de combustível é controlar a quantidade exata de combustível no tempo exato para a obtenção da razão estequiométrica [9]. Baseado nos sinais de entrada, a unidade de controle o instante em que cada bico injetor é ativado ou não. Este sistema será descrito com mais detalhes, devido à sua caracterização pela aplicação prática deste trabalho.

Para que o motor tenha um funcionamento suave, econômico e não contamine o ambiente, ele necessita receber a perfeita mistura ar/combustível em todas as faixas de rotação. Um carburador, por melhor que seja e por melhor que esteja sua regulagem, não consegue alimentar o motor na proporção ideal de mistura em qualquer regime de funcionamento. Os sistemas de injeção eletrônica têm essa característica de permitir que o motor receba somente o volume de combustível que ele necessita.

Mais do que isto, os conversores catalíticos - ou simplesmente catalisadores - tiveram papel decisivo no desenvolvimento de sistemas de injeção eletrônicos. Para que sua eficiência fosse plena, seria necessário medir a quantidade de oxigênio presente no sistema de exaustão e alimentar o sistema com esta informação para corrigir a proporção da mistura. O primeiro passo neste sentido foram os carburadores eletrônicos, mas cuja difícil regulagem e problemas que apresentaram, levaram ao seu pouco uso.

Surgiram então os primeiros sistemas de injeção monoponto (ou *single-point*), consistindo de uma válvula injetora ou bico, que fazia a pulverização do combustível junto ao corpo da borboleta do acelerador. Toda vez que o pedal do acelerador é acionado, esta válvula (borboleta) se abre, admitindo mais ar. Um sensor no eixo da borboleta indica o quanto de ar é admitido. Esta informação é reconhecida pela central de gerenciamento, que fornece o combustível proporcionalmente.

Para que o sistema possa suprir o motor com maiores quantidades de combustível de acordo com a necessidade, a linha de alimentação dos bicos injetores é pressurizada e alimentada por uma bomba de combustível elétrica, a qual envia doses maiores que as necessárias para que sempre o sistema possa alimentar adequadamente o motor em qualquer regime em que ele funcione. O excedente retorna ao tanque. Nos sistemas *single point* a alimentação é direta ao bico único. No sistema *multi-point*, em que existe um bico para cada cilindro, existe uma linha de alimentação única para fornecer combustível para todos os injetores, localizada antes da válvula de admissão.

Seja no caso de sistemas *single-point* ou *multi-point*, os bicos injetores dosam a quantidade de combustível liberada para o motor pelo tempo em que permanecem abertos. As válvulas de injeção são acionadas eletromagneticamente, abrindo e fechando através de impulsos elétricos provenientes da unidade de comando. Estes tipos de injeção estão ilustrados nas figuras 3 e 4.

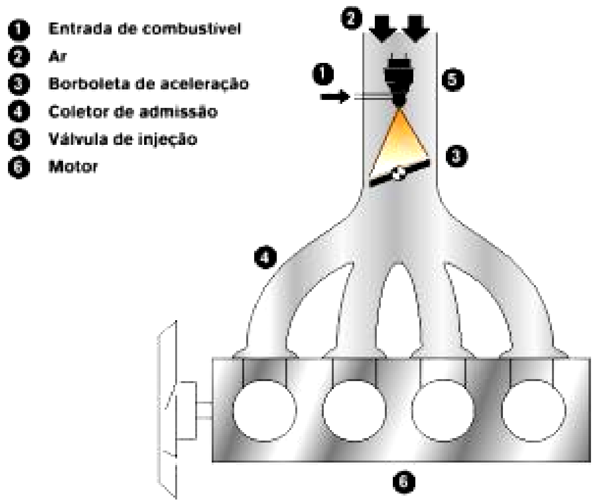

Figura 3: Sistema de Injeção de Combustível *Single-Point*

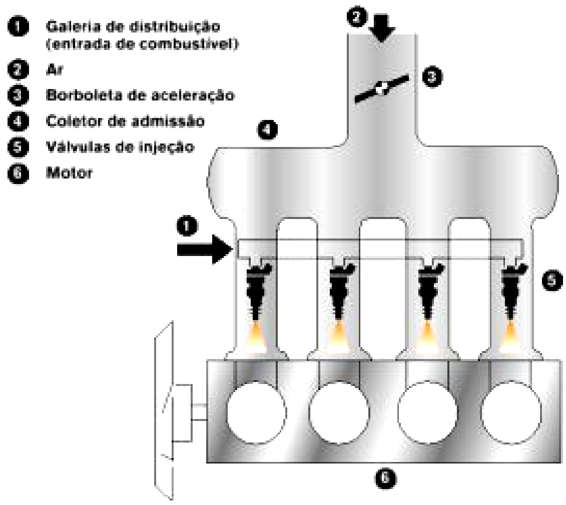

Figura 4: Sistema de Injeção de Combustível *Multipoint*

2.1.4.1. Esquema de funcionamento

O instante no qual o injetor atuará no sistema é determinado pela unidade eletrônica de controle. Esta informação é obtida através de um sensor de fundamental importância chamado **sensor Hall**. Seu princípio de funcionamento consiste em gerar diferenças de potencial de voltagem a partir de efeitos eletromagnéticos [10].

O sensor Hall é constituído basicamente por uma pastilha semicondutora alimentada eletricamente. Esta pastilha fica associada ao eixo de rotação do motor, que transmite seu movimento a um disco giratório com quatro janelas (Figura 5). Quando a abertura do disco giratório está posicionada entre o sensor e o imã permanente, o primeiro fica imerso no campo magnético do imã. Esta situação gera no interior da Unidade de Comando uma tensão de aproximadamente 12 Volts.

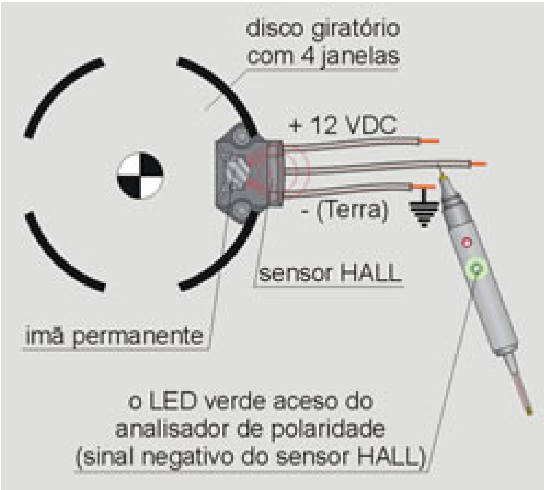

Figura 5: Sensor Hall com abertura entre o sensor e o imã

Quando o disco está posicionado entre o imã e o sensor, não há contato do sensor HALL com o campo magnético e a tensão gerada é de zero Volt (Figura 6).

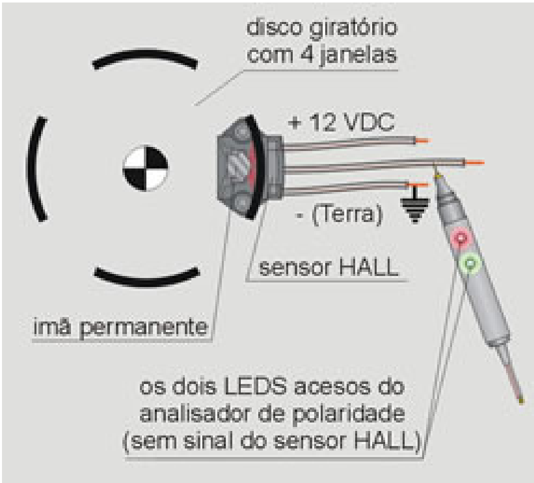

Figura 6: Sensor Hall posicionado entre o imã e o sensor

O disco giratório pode ter 4 janelas igualmente espaçadas, ou 3 janelas igualmente espaçadas entre si e uma maior (dependendo do sistema em questão). No disco de 4 janelas simétricas, o início das janelas indica quantos graus estão 2 dos cilindros do ponto morto superior (esta angulação varia de acordo com o sistema de injeção – Figura 7). No disco de 3 janelas iguais e uma maior, o início da janela maior indica quantos graus está o 1º cilindro do ponto morto superior.

Especificamente neste segundo caso, o sensor acaba gerando uma onda quadrada com um período constante, mas com o *duty cycle*[1] da janela maior diferente de 50% (Figura 8). Esta diferença, associada à análise do sinal de rotação do motor, permite inferir o momento exato que a seqüência de injeção de combustível nos cilindros deve ser iniciada.

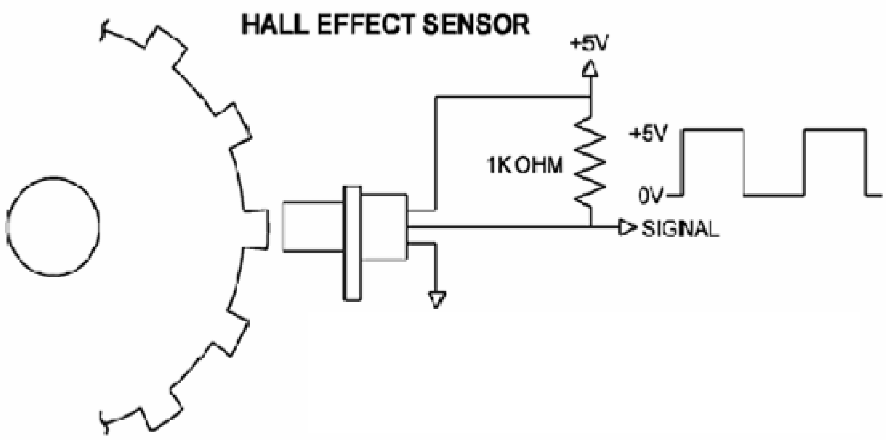

Figura 7: Representação do sensor Hall e da onda gerada

[1] Razão entre o período do pico e o período total de uma onda quadrada.

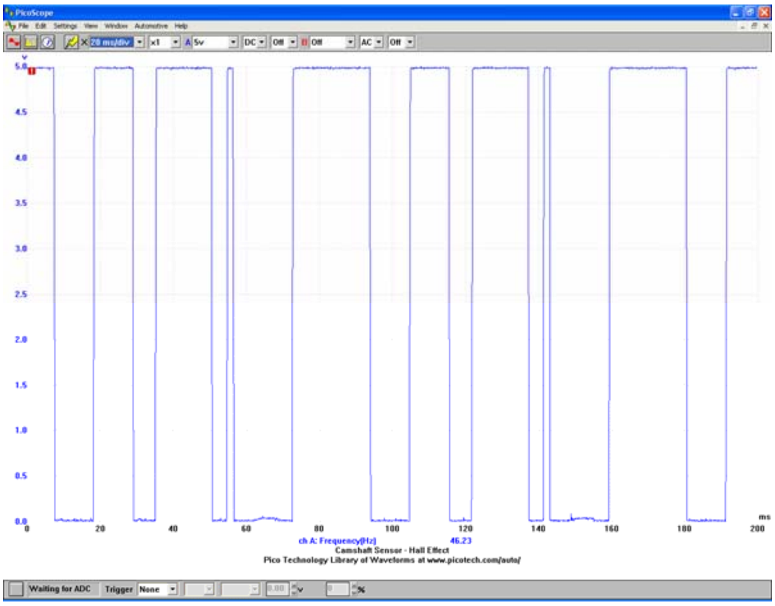

Figura 8: Exemplo de três janelas iguais e uma maior, no sensor Hall

Existem algumas estratégias que a ECU pode adotar para realizar a injeção de combustível. Como a injeção é determinada pela posição angular do virabrequim, é considerada uma injeção síncrona. Dependendo da aplicação, os três principais tipos de injeção síncrona são: simultânea, por grupo ou seqüencial [11].

Na injeção simultânea, todos os bicos são acionados ao mesmo tempo por um circuito em comum. A admissão de combustível é feita uma vez por ciclo do motor. Já na injeção por grupo, os bicos são divididos em partes, sendo que a entrada de combustível ocorre de maneira alternada entre estes grupos (em um motor com quatro cilindros, geralmente se divide em dois grupos de dois cilindros, sendo o primeiro e o terceiro cilindros acionados simultaneamente, e a seguir o segundo e o quarto). A injeção seqüencial, como o próprio nome diz, ocorre de maneira a ativar um bico por vez, ordenadamente, de maneira que todos os bicos injetem combustível pelo menos uma vez durante um ciclo do motor [11].

2.2. Dispositivos de Processamento

2.2.1. Processador de Sinais Digitais (DSP)

Sinais na vida real são analógicos por natureza. Entretanto, para que se possa trabalhar computacionalmente com eles, é preciso que estes sejam representados de maneira digital. Este é o conceito do processamento digital de sinais, no qual o Processador Digital de Sinais (DSPs, do inglês *Digital Signal Processor*) está inserido.

Os DSPs são microprocessadores especializados em processamento digital de sinais. A utilização destes dispositivos tem crescido significativamente nos últimos anos, tendo no mercado de dispositivos portáteis (celulares, *handhelds*) o principal destaque [12].

Os Processadores de Sinais Digitais são dispositivos especializados em processamento digital de sinais das mais diversas naturezas (áudio, vídeo, dados), quer em tempo real quer *off-line*. Possui uma alta velocidade de processamento, se comparado com a maioria dos microcontroladores disponíveis no mercado, medida em MIPS (*Million Instruction Per Second*) [13].

Recentemente, estes microprocessadores têm sido utilizados em projetos presentes no mercado envolvendo controle digital. São capazes de prover de maneira rápida e eficaz soluções para diversos problemas deste tipo de sistema (processamento em tempo real). Podem ser usados tanto sozinhos quanto em união com outros elementos computacionais (periféricos, microcontroladores, FPGAs).

Os DSPs podem ser divididos em duas categorias principais, baseadas na maneira que representam valores numéricos e operações numéricas. Estes dois formatos principais são ponto fixo e ponto flutuante. As diferenças entre processadores de ponto fixo e flutuante são tão significativas que requerem implementações (internas e de algoritmos) distintas, além de um conjunto específico de instruções [14].

Os processadores de ponto fixo representam e manipulam números como inteiros. Os processadores de ponto flutuante representam primeiramente números no formato do ponto flutuante, embora possam também suportar a representação e os cálculos de números inteiros. O ponto flutuante é representado como uma combinação da mantissa (ou parte fracionária) com um expoente.

Os processadores de ponto flutuante podem executar as operações tanto de ponto flutuante como de inteiros, tornando-os mais flexíveis. A potencialidade deste tipo de DSP é apropriada nos sistemas onde os coeficientes do ganho mudam com tempo, ou os coeficientes têm escalas dinâmicas grandes. Em contrapartida, apresentam custos mais elevados.

Por estas razões, geralmente os DSPs de ponto fixo são mais baratos, e costumam realizar tarefas mais simples com maior velocidade. Todavia, os DSPs de ponto flutuante permitem maior precisão na representação numérica, além de um ciclo de desenvolvimento mais rápido (os programadores não precisam se preocupar com problemas como *overflow* ou *underflow* de variáveis, por exemplo) [12].

As principais características que diferem um DSP de um microprocessador comum são [15]:

- Paralelismo na execução das instruções;
- Otimização da operação produto/acumulação (endereçamento circular);
- Otimização da arquitetura para operações matemáticas repetitivas (ciclos);
- Comutação consciente de contexto nas interrupções;
- Separação entre dados, dados secundários e instruções do programa;

2.2.2. Field Programmable Gate Array (FPGA)

O *Field Programmable Gate Array* (FPGA) é um dos dispositivos semicondutores mais utilizados para o processamento de informações digitais. Foi criado pela Xilinx Inc., e teve o seu lançamento no ano de 1985 como um dispositivo que poderia ser configurado de acordo com as aplicações do usuário (programador) [16].

Esta, por sinal, é uma das suas principais vantagens. Apesar de outros dispositivos também serem maleáveis, nenhum deles permite que se faça uma reconfiguração completa do sistema. A esta característica dá-se o nome de reconfigurabilidade. Através das ferramentas de desenvolvimento, é possível especificar o dispositivo para funcionar conforme os interesses do seu projeto.

Os FPGAs têm sido bastante utilizados para o controle de sistemas digitais. Uma das áreas de atuação é a de controle de motores, controle de dispositivos eletrônicos voltados ao controle elétrico, e controle de movimento, conforme pode ser observado na figura 9. A parte de controle de motores se refere à manipulação direta de motores AC (corrente alternada) e DC (corrente contínua) para se obter velocidade, posição ou torque específico [17].

O controle elétrico está relacionado diretamente com estratégias para a conversão de sistemas DC-AC, AC-DC ou DC-DC. Já a parte de controle de movimento se refere à implementação de algoritmos para desvio de obstáculos, controle do perfil de aceleração e rota, usualmente relacionados com a área de robótica ou das chamadas máquinas CNCs (*Computerized Numerically Control*). Outras aplicações que se pode citar são implementações de conceitos de lógica *fuzzy* para controle de temperatura, e construção de sistemas de controle eletrônicos automotivos [17].

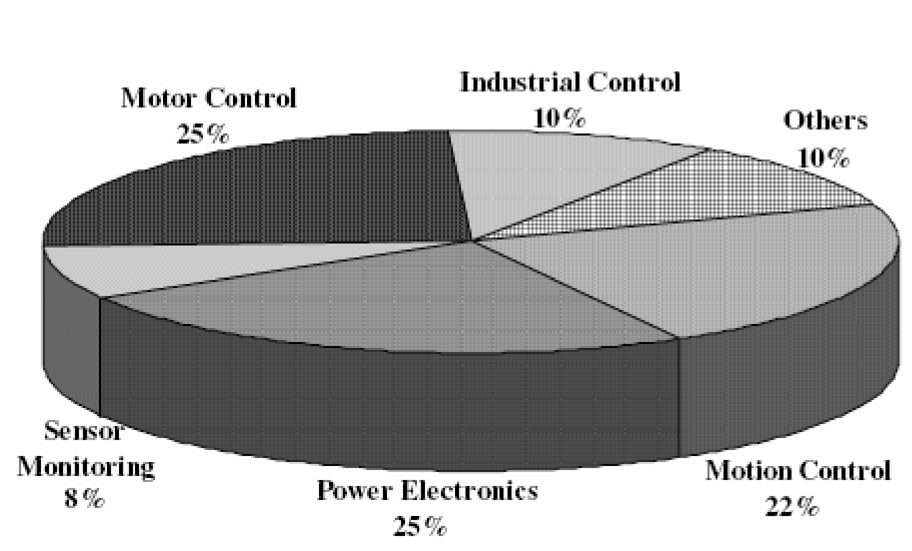

Figura 9: Distribuição das principais aplicações que utilizam FPGAs [17]

Basicamente, são constituídos por blocos lógicos, blocos de entrada e saída, e chaves de interconexão. Os blocos lógicos formam uma matriz bidimensional, e as chaves de interconexão são organizadas como canais de roteamento horizontal e vertical entre as linhas e colunas dos blocos lógicos. Os canais de roteamento possuem chaves de interligação programáveis, que permitem conectar os blocos lógicos de maneira conveniente em função das necessidades de cada projeto [16].

No interior de cada bloco lógico do FPGA existem vários modos possíveis para implementação de funções lógicas. O mais utilizado pelos fabricantes de FPGA é o bloco de memória LUT (*Look-Up Table*). Esse tipo de bloco lógico contém células de armazenamento que são utilizadas para implementar pequenas funções lógicas. Quando um circuito lógico é implementado em um FPGA, os blocos lógicos são programados para realizar as funções necessárias. Os canais de roteamento são estruturados de forma a realizar a interconexão necessária entre os blocos lógicos.

A arquitetura de roteamento de um FPGA é a forma pela qual seus barramentos e as chaves de comutação são posicionados para permitir a interconexão entre as células lógicas. Essa arquitetura deve permitir que se obtenha um roteamento completo e, ao mesmo tempo, uma alta densidade de portas lógicas [17].

As chaves programáveis de roteamento apresentam algumas propriedades que afetam principalmente a velocidade e o tempo de propagação dos sinais. Tais características (tamanho, resistência, capacitância e tecnologia de fabricação) definem itens como volatilidade e capacidade de reprogramação. Na escolha de um dispositivo reconfigurável, esses fatores devem ser considerados diante do sistema proposto.

3. Materiais e Métodos

3.1. Descrição do Sistema

A partir da revisão bibliográfica sobre o princípio de funcionamento das unidades eletrônicas de controle automotivo, e das principais características que as compõem, foi proposta a construção de um pequeno sistema que pudesse simular de maneira simplificada algumas funcionalidades de um motor. Este sistema está representado através do diagrama de blocos da figura 10.

Entenda-se por funcionalidades algumas informações presentes em um motor real, tais como aceleração e rotação. Para este projeto, foi estipulado que a rotação pudesse variar entre 240 rpm (rotações por minuto) e 1200 rpm. Este valor é apenas uma representação de um possível sinal de rotação encontrado em sistemas automotivos reais.

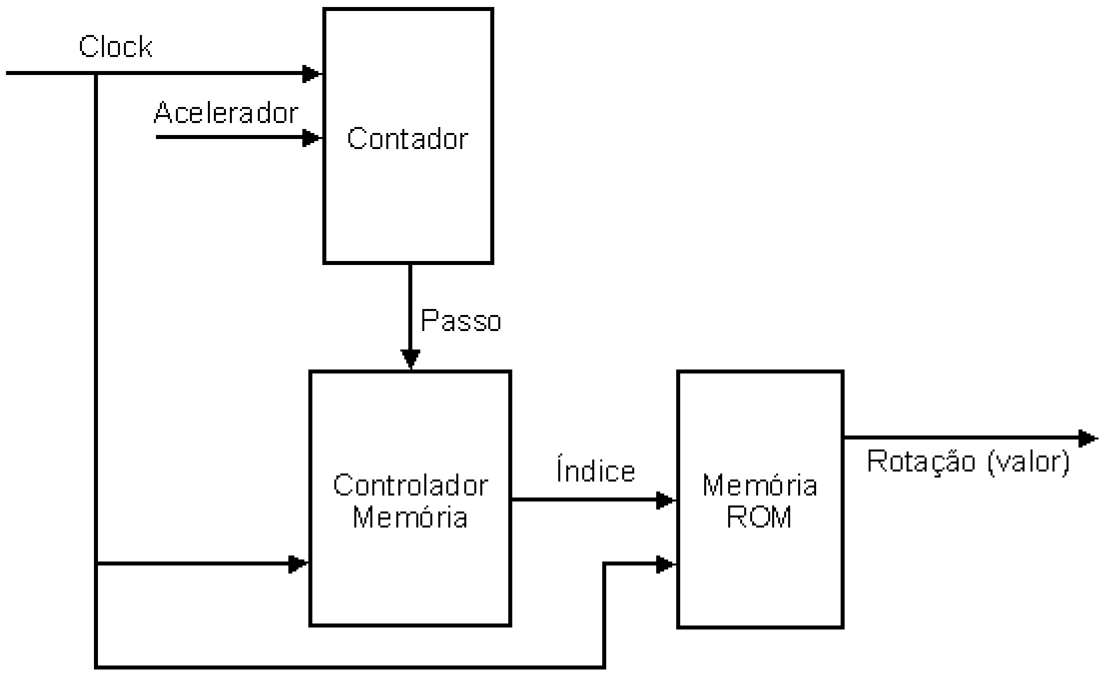

Figura 10: Diagrama de blocos do simulador de motor

Na outra ponta do sistema, foi construído um módulo de controle destes sinais gerados a partir do bloco anteriormente descrito. Este módulo pode ser considerado como uma unidade de controle simplificada, de acordo com as especificações do motor simulado. Esta unidade de controle é responsável por detectar, no sinal de rotação, um período que contenha um *duty cycle* diferente de 50%. Esta condição é necessária para ativar o sistema de injeção de combustível nos cilindros (considerou-se um motor de quatro cilindros, com estratégia de injeção em grupo, ou seja, os injetores ímpares são acionados simultaneamente, assim como os pares na seqüência).

Além disso, esta unidade de controle é responsável por atuar no sistema de admissão de ar. Ele faz isso a partir do mapeamento dos estados de funcionamento do motor. Uma vez identificado o valor de rotação, este é utilizado para calcular o ângulo de abertura da borboleta de admissão de ar.

Um esquemático desta unidade simplificada de controle pode ser visto na figura 11.

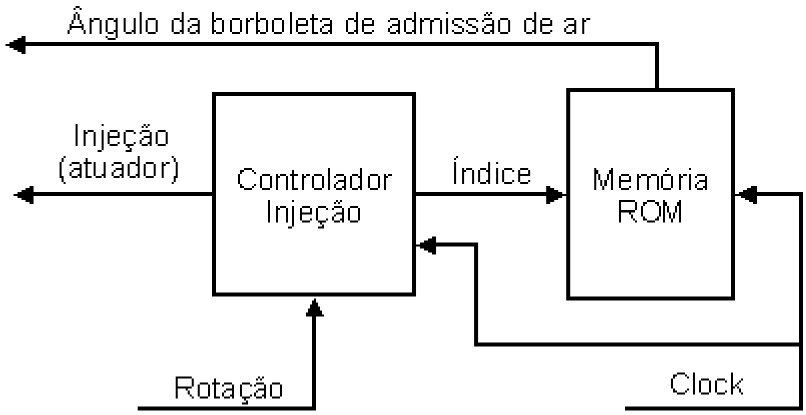

Figura 11: Diagrama de blocos do sistema de controle eletrônico

3.2 Materiais e *softwares* utilizados

Neste trabalho, algumas ferramentas e dispositivos foram utilizados para a elaboração do sistema. Basicamente, foram utilizados *softwares* de programação para DSPs e FPGAs. No caso dos primeiros, o programa utilizado foi o CodeComposer Studio, em sua versão 3.1; já para os últimos, o ambiente de desenvolvimento foi o Xilinx ISE 9.2i.

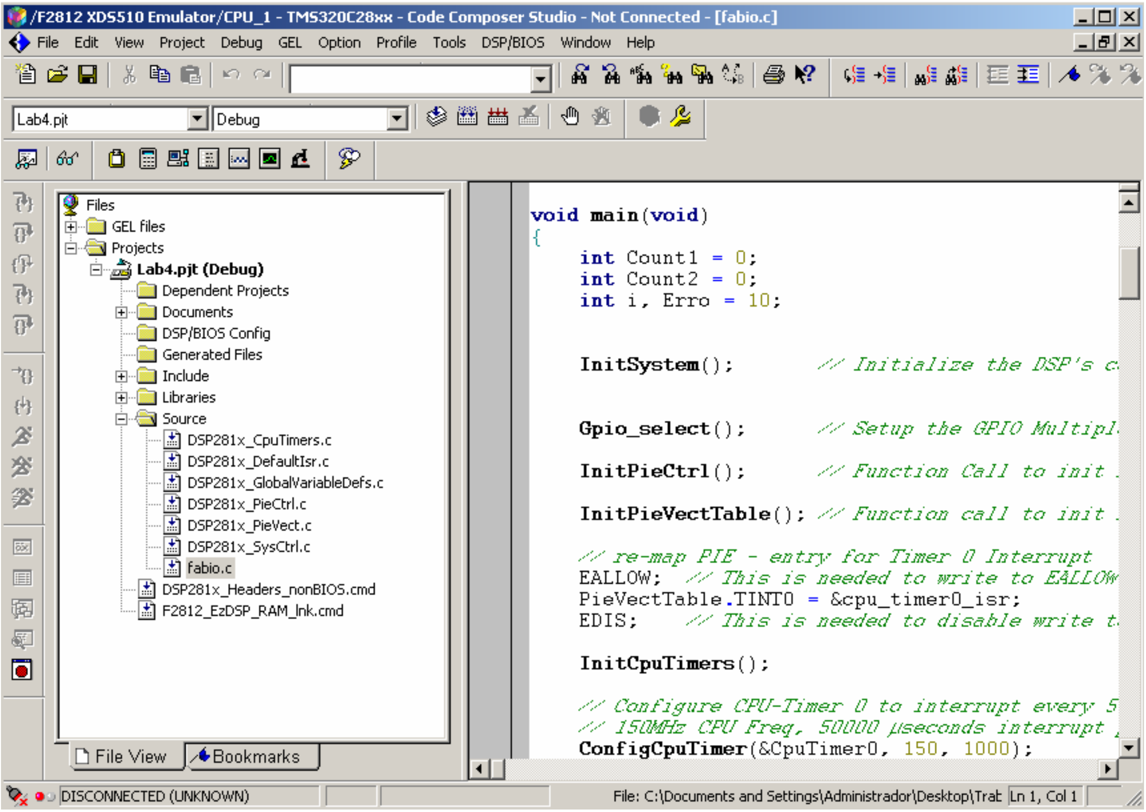

Figura 12: Tela do programa CodeComposer

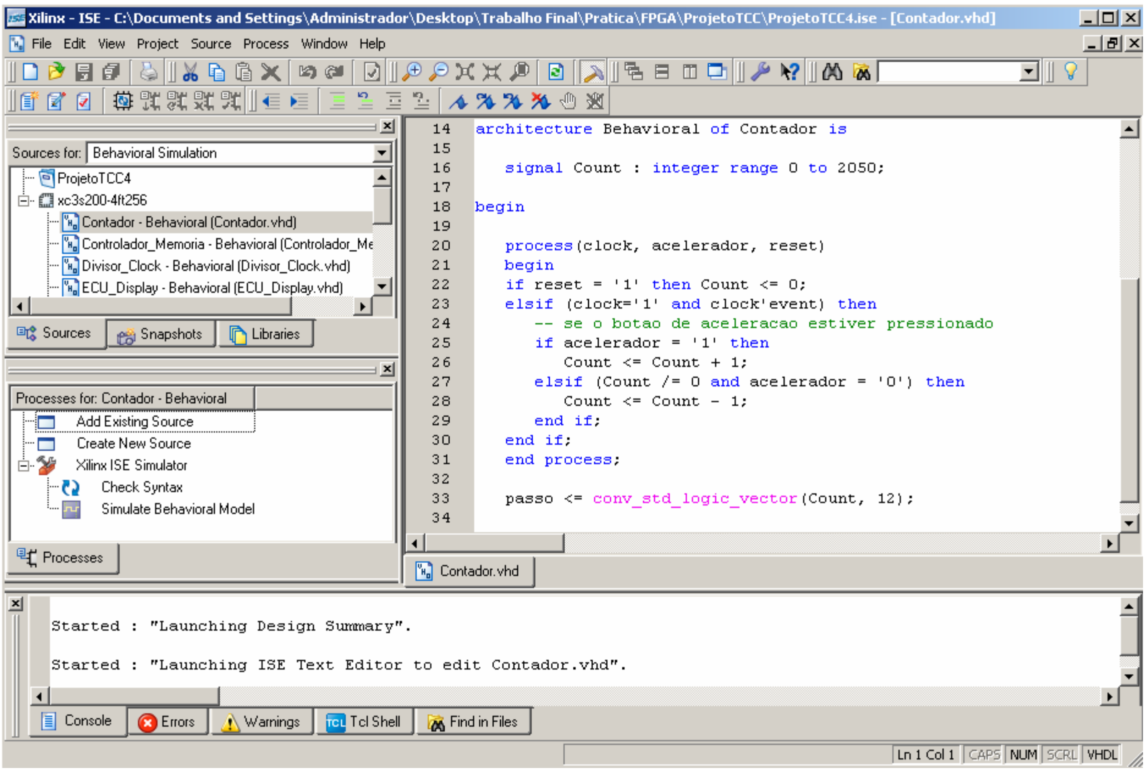

Figura 13: Tela do programa ISE

Sobre os dispositivos utilizados, o DSP escolhido para os trabalhos foi o modelo TMS320F2812 da Texas Instruments. Este modelo é conhecido por combinar a facilidade de uso de um microcontrolador com o poder de processamento de um DSP, além de ter a facilidade de programação e eficiência da linguagem C. Segundo o fabricante, é recomendado para aplicações embarcadas em ambientes industriais, tais como controle digital de motores, por exemplo.

Este chip faz parte de um kit de desenvolvimento fornecido pela Spectrum Digital Incorporated Inc., e tem como características principais:

- *Clock* de 150MHz;
- 18 Kb de memória RAM;
- 128Kb de memória ROM (embutida no chip);
- 64Kb de memória RAM (embutida no chip);

Especificamente, o microcontrolador possui características importantes para o controle automotivo, como suporte ao protocolo de comunicação CAN, *timers* e conversores analógico-digitais [19].

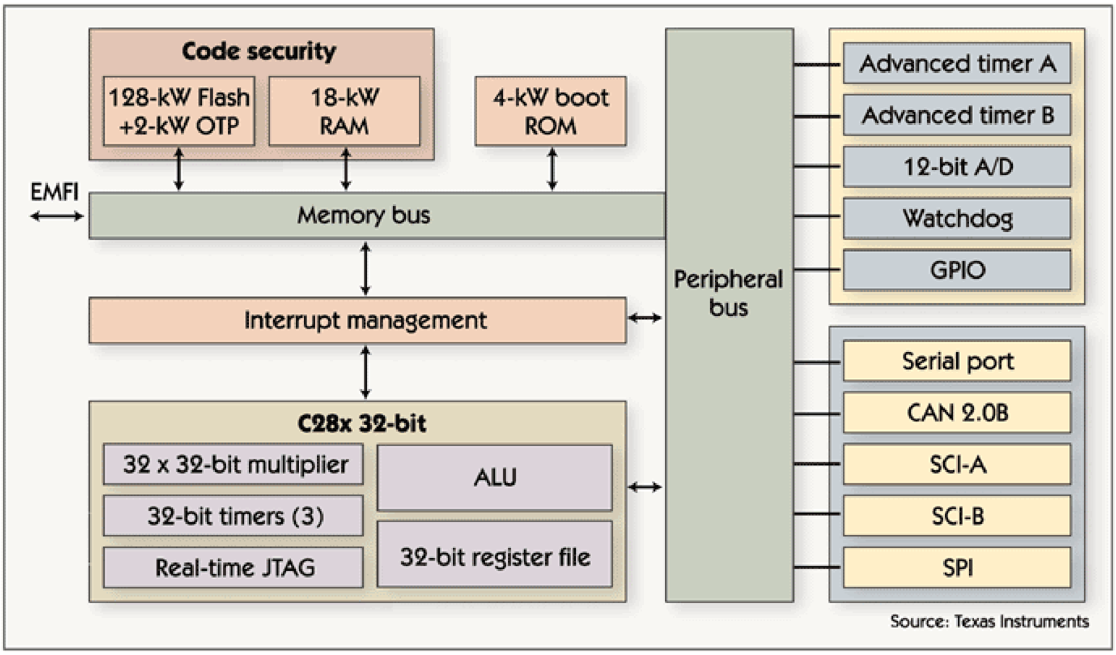

Figura 14: Diagrama de blocos do TMS320F2812, da Texas Instruments (fonte: Texas Instruments)

No caso do FPGA, este trabalho utilizou o kit de desenvolvimento fornecido pela Xilinx Inc. denominado Spartan3, revisão E. Este kit contém o dispositivo XC3S200, que contém as seguintes características principais [20]:

- 50 MHz de *clock* (embutido na placa);
- 200K de portas programáveis;
- 12 multiplicadores dedicados, que auxiliam na velocidade do processamento;
- 4 DCMs (*Digital Clock Managers*);
- 173 portas de entrada e saída (máximo);
- Interface serial e VGA (no kit);
- 4 Displays de 7 segmentos, e expansão para inclusão de display de LCD 16x2.

3.3. Características de implementação

3.3.1. Motor Simulado

Basicamente, o sistema é constituído de um módulo que verifica se o usuário está acelerando ou não – neste caso, representado como um botão que pode ser apertado ou não. Caso este sinal de aceleração seja positivo, o bloco fica responsável por contar quanto tempo este sinal fica ativo no sistema (acelerando), através de um contador. Assim que este sinal deixa de ser positivo (volta para zero), o contador é decrementado unitariamente até que atinja o valor inicial.

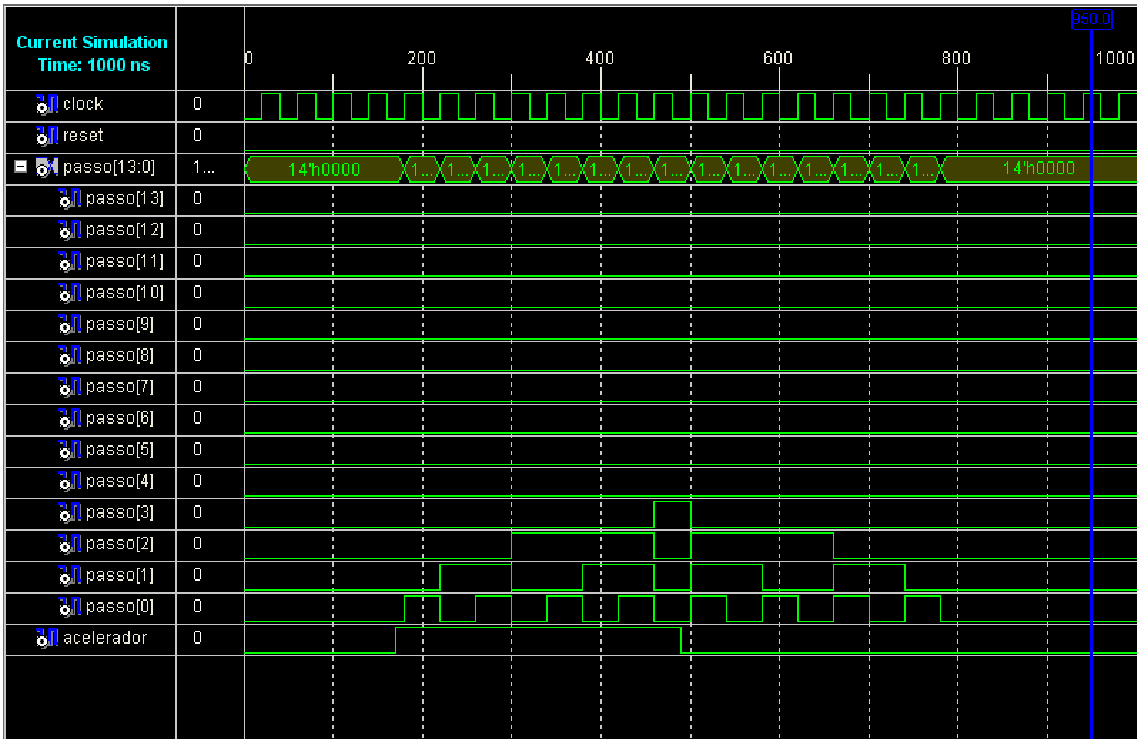

Figura 15: Simulação do contador de aceleração feita no ISE Simulator

Este valor do contador é transmitido para outro bloco, responsável por acessar uma tabela de dados que contém o mapeamento de todos os valores possíveis de rotação dentro dos valores máximo e mínimo estipulados no projeto. Em outras palavras, o resultado do contador é utilizado como índice para acessar cada posição desta tabela gravada em uma memória ROM.

A relação entre o valor obtido da aceleração e o valor da rotação encontrado na tabela não é linear, mesmo porque em sistemas reais essa prerrogativa é verdadeira. Assim, foi elaborada uma regra simples para discretização de uma curva de aceleração. Esta regra consiste nas seguintes regras:

Paramétrico = Número – MaxJanela

Passo = Paramétrico / FatorDiscretização

*Índice = Passo + (n*10)*

Onde *Número* é o valor recebido do bloco contador; *Paramétrico* é o valor parametrizado do Número, ou seja, o valor entre zero e o máximo da janela de discretização para aquele intervalo; *FatorDiscretização* é o número pelo qual este valor parametrizado será dividido, ou seja, quantos passos o contador precisa dar para que o valor do índice na tabela de dados seja incrementado; *Índice* é o valor propriamente dito que será utilizado para referenciar as posições na memória ROM; e *n* é o número do intervalo de discretização – no caso deste sistema, foram considerados 16 intervalos.

Um exemplo prático auxilia no entendimento destas regras:

```
if (Numero >= 0 and Numero <= 160) then
      Parametrico <= Numero;
      IndiceParcial <= Divisao (Parametrico, 16);
      Indice <= IndiceParcial;
elsif (Numero > 160 and Numero <= 310) then
      Parametrico <= Numero – 160;
      IndiceParcial <= Divisao (Parametrico, 15);
      Índice <= IndiceParcial + 10;
```

Considere, por exemplo, o valor 35 como resultado do contador de aceleração. Este número cai no primeiro intervalo de condição. Ele será simplesmente dividido por 16, e o resultado (2) será considerado como o índice da tabela ROM. Este cálculo tem o efeito prático de forçar que, a cada 16 passos do contador de aceleração, o índice da tabela que contém valor de rotação seja incrementado uma vez.

Agora, considere o número 210. Este valor satisfaz a segunda condição de parametrização. Por isso, é retirado deste valor o máximo do intervalo de discretização anterior (neste caso, 160) – resultando sempre em um valor entre 0 e 150. Daí, este valor parametrizado é dividido por 15, para se obter o mesmo efeito do caso anterior – a cada 15 passos do contador, um passo do índice da tabela é dado. Assim, sucessivamente são feitas parametrizações até que, em determinado valor, a proporção entre o valor lido do contador e o utilizado como índice da tabela é de 1:1. Esta conta acaba por criar o gráfico da figura 16:

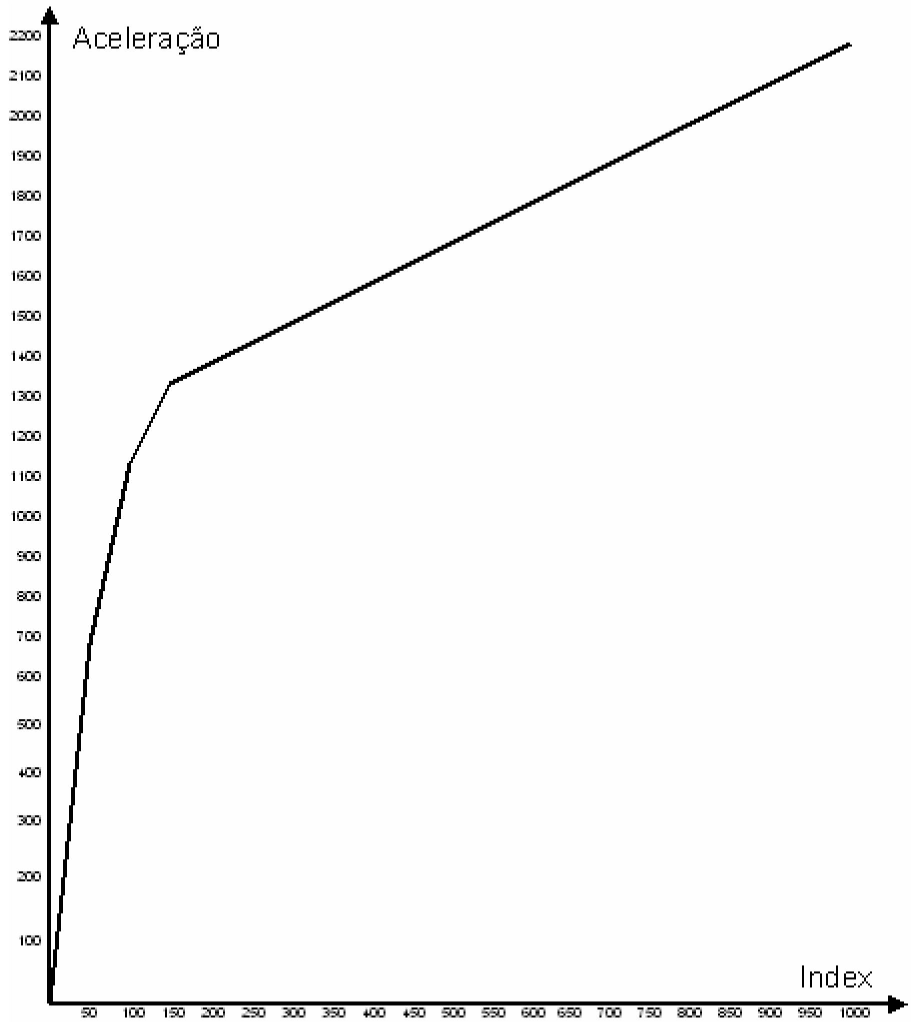

Figura 16: Curva que relaciona a aceleração com os índices da tabela de rotação

O efeito prático desta medida é a de que, nos primeiros instantes da aceleração, o motor simulado responda com mais suavidade ao estímulo dado, oferecendo uma resposta mais rápida em termos de rotação a partir de valores maiores de aceleração.

Pela facilidade de implementar sistemas modelados em blocos, esta parte do trabalho foi feita em VHDL para FPGA, utilizando o dispositivo Spartan3, da Xilinx Inc.

3.4. Unidade Eletrônica de Controle

3.4.1. Implementação do Sistema em FPGA

Com as especificações do projeto disponíveis, a implementação para FPGA ocorreu sem grandes problemas. Devido à maneira natural de arquitetar um sistema através de blocos, este tipo de representação facilita deveras a execução do projeto em VHDL.

Mesmo depois da divisão em blocos, o sistema em FPGA pôde ser mais descentralizado ainda, já que o dispositivo permite processamento paralelo nativo, através dos chamados *process* dentro da descrição do comportamento das arquiteturas de cada bloco. Por isso, é exigida por parte do projetista uma visão paralela do comportamento do sistema, já que várias tarefas podem ser executadas simultaneamente.

Assim, a implementação em VHDL foi muito semelhante à já mencionada estrutura de blocos do sistema. Ao todo, quatro blocos foram implementados e interconectados entre si, a saber:

- ***Divisor_Clock* =** bloco responsável por gerar freqüências de operação mais baixas para o sistema. Segundo a especificação, a rotação mais rápida encontrada na tabela é de 1200 rpm. Isso implica em uma onda com período igual a 50 ms, ou seja, freqüência de 20Hz, conforme indica as equações (1) e (2).

$$f = 1200 / 60 = 20\ Hz \qquad (1)$$

$$T = 1 / 20 = 0{,}05\ s = 50\ ms \qquad (2)$$

Para o processo de detecção do *duty cycle*, é necessário realizar amostragens nesse sinal de entrada. Pelo Critério de *Nyquist*, o dobro da freqüência já era suficiente para se ter uma boa precisão sobre o sistema – o que torna o *clock* da placa (de 50MHz) muito alto para realizar tal tarefa. Assim, através deste bloco é possível obter freqüências de 2MHz, 2KHz e 500Hz.

- ***ECU_Display*** = bloco que recebe o valor da rotação como entrada, e fica encarregado de formatá-lo e exibi-lo nos displays de 7 segmentos do kit de desenvolvimento.

- ***Control_Injecao*** = módulo responsável pela detecção do *duty cycle* diferente de 50% no sinal de rotação. Para cada período do sinal, são realizadas amostragens seqüenciais e armazenadas quantas destas são em nível lógico alto, e quantas têm nível lógico baixo. Então, ao fim do período o controlador compara quantas amostras existem de cada tipo, e se a diferença entre elas for maior que um determinado erro estipulado (por definição de projeto, o erro de amostragem é de 10%), o sinal de injeção é enviado aos cilindros, alternadamente.

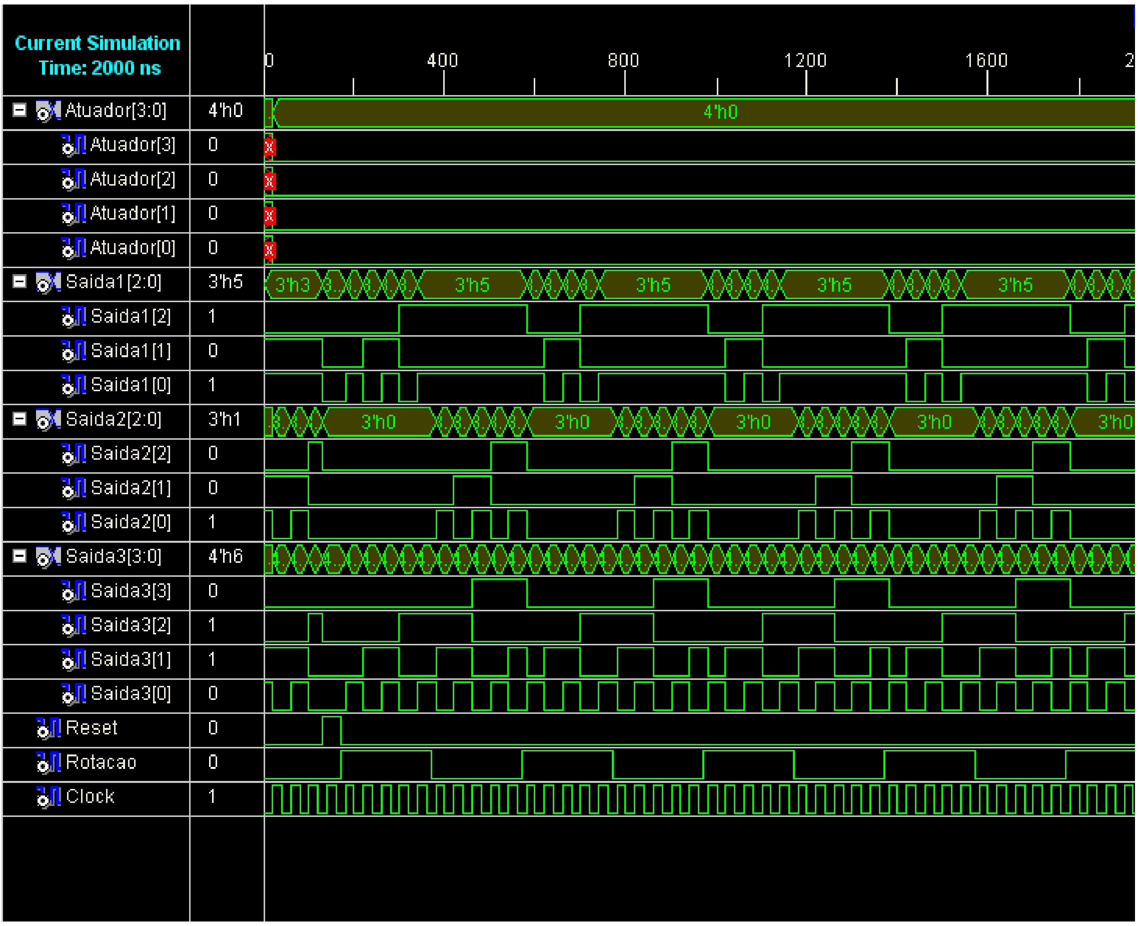

Figura 17: Simulação do controle de injeção a partir da detecção do *duty cycle*

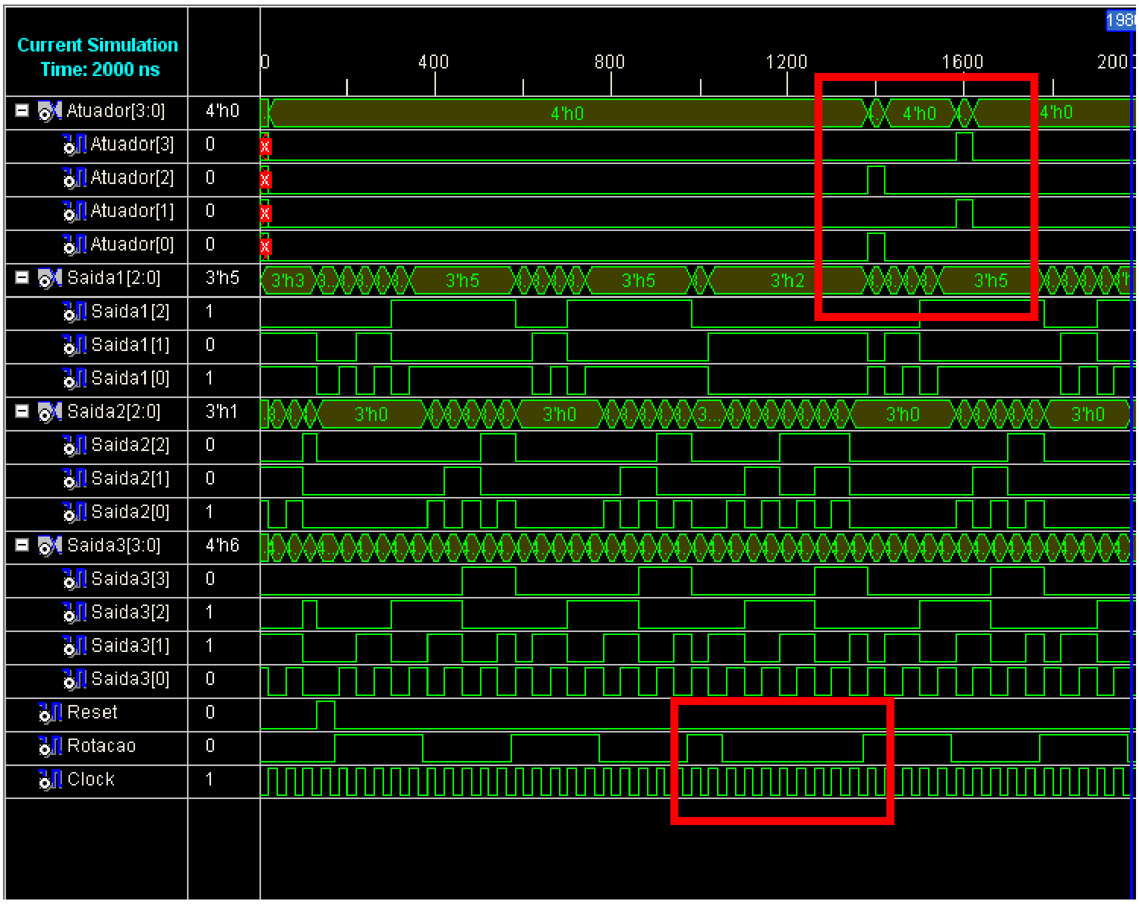

Figura 18: Identificação do *duty cycle* diferente de 50%, e conseqüente acionamento dos sinais de injeção

Por uma questão de espaço, a unidade de controle foi implementada no mesmo dispositivo que o simulador do motor.

3.4.2. Implementação do Sistema em DSP

A programação do DSP foi realizada em linguagem C, que por um lado facilita muito na implementação propriamente dita do sistema, mas prejudica a noção de processamento paralelo. Assim como no caso da FPGA, no controlador foram implementadas rotinas que permitiam identificar um período que tivesse um *duty cycle* diferente de 50%.

O programa em si, em linguagem C, é relativamente simples. Basicamente, dentro de um laço do tamanho do período do sinal a ser analisado (no caso, da rotação do motor), e a cada intervalo de tempo definido realizar uma amostragem deste sinal, verificando se está em nível lógico alto ou baixo. A seguir, comparar o número de amostras de cada tipo: se a diferença entre elas for maior que um erro estipulado, o sinal de ativação da injeção é enviado.

Neste caso, assim como na implementação com FPGA, foi necessário realizar uma diminuição do *clock* para efetuar as amostragens. Entretanto, devido à presença de *timers* e suporte a interrupção, esse processo foi bem mais simples (a cada medição, esperava-se um tempo determinado, que era contado pelo timer).

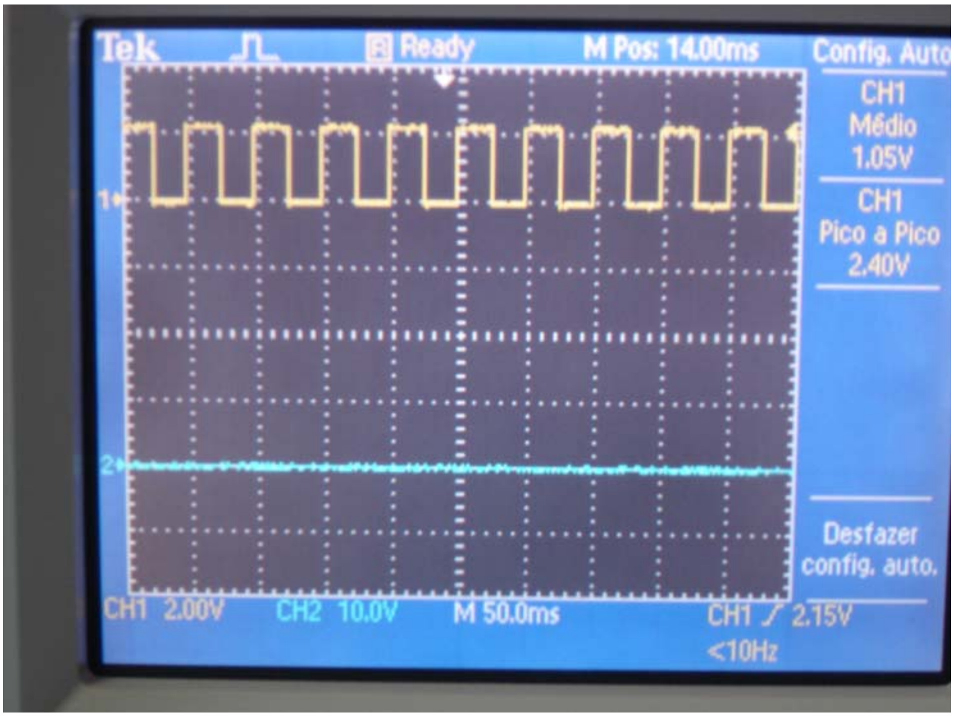

Figura 19: Osciloscópio indicando os sinais de rotação e acionamento da injeção

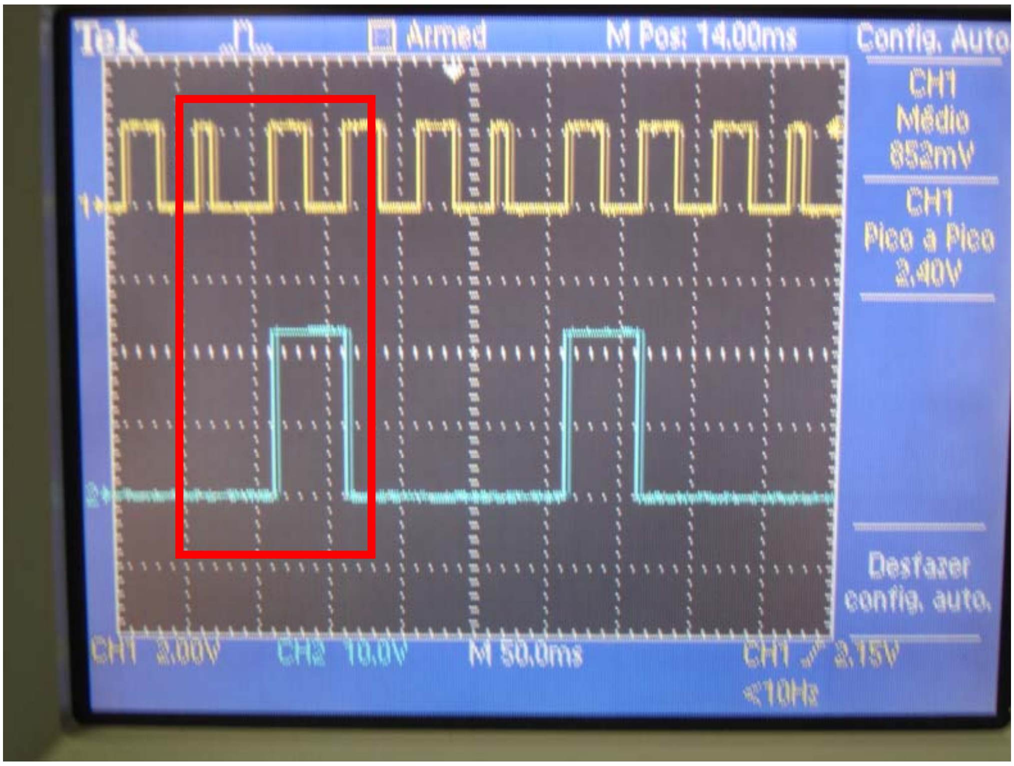

Figura 20: Detecção do *duty cycle* diferente de 50% no osciloscópio, conectado ao DSP

4. Resultados e discussões

4.1. Vantagens e Desvantagens de cada dispositivo (neste projeto)

Tanto a utilização de DSPs quanto FPGAs para a construção de sistemas de controle automotivo é viável e eficiente, desde que implementada da maneira correta. No caso específico do projeto deste trabalho, ambos os dispositivos tiveram um comportamento muito satisfatório no que diz respeito à velocidade de processamento e capacidade de controle do sistema proposto.

Todavia, de acordo com os objetivos estabelecidos inicialmente para a execução deste trabalho de conclusão de curso, foi possível perceber de maneira sutil algumas diferenças entre as duas abordagens propostas, que não implicam necessariamente nos resultados finais de cada dispositivo, mas que podem influenciar na fase de projeto envolvendo cada um deles.

A partir da revisão bibliográfica, uma arquitetura de sistema foi criada e representada através de blocos, que executavam tarefas simultaneamente. Esse tipo de pensamento facilita bastante o trabalho com FPGAs, já que a tarefa da implementação acaba se tornando apenas uma codificação em VHDL das idéias que já se teve sobre o sistema. Essa facilidade é um pouco menor com um projeto seqüencial, como é a programação em C para DSPs – ainda que isso não represente uma dificuldade, apenas uma diferença.

Outro ponto interessante entre as duas abordagens é a diferença de preocupação que se tem com relação a cada um deles. Enquanto no FPGA o projetista tem controle total sobre o sistema, e sabe o que existe ou não existe nele, com o DSP é necessário um pouco de atenção com as configurações necessárias para que ele funcione perfeitamente. Em outras palavras, antes de se programar propriamente dito o sistema, é necessário voltar um pouco de atenção às configurações de uma série de registradores, que interferem no comportamento do sistema, tais como tipos de porta (entrada ou saída), habilitação ou desabilitação de alguns controles, tais como *watchdog* e interrupções, entre outros.

Por outro lado, esses mesmos dispositivos que não estão presentes em FPGA ajudam bastante o programador de DSP na hora de realizar algumas tarefas, como atraso na medição do sinal de entrada, por exemplo. Enquanto que no primeiro foi necessário construir um bloco que dividia o sinal de *clock* em sinais de freqüência mais baixa, no DSP esta operação foi feita apenas com a configuração relativamente simples de uma interrupção e de um *timer*, que realizava a contagem do tempo e retornava assim que este findava.

As ferramentas de desenvolvimento também apresentaram algumas diferenças importantes entre si, além das já esperadas – afinal, se tratam de ferramentas distintas para linguagens de programação distintas. Porém, uma sensação que ficou bastante acentuada é a de que o CodeComposer (para trabalhar com DSP) é bem mais robusto em termos de problemas (travamento) do que o ISE, embora este último seja um pouco mais intuitivo (comandos mais acessíveis ao usuário, interface mais clara e organizada) que o primeiro. Ainda assim, ambientação com a ferramenta de desenvolvimento pode ser resolvida com treinamentos e cursos; problemas como simulações que travam ou estouros de memória por parte do ambiente de desenvolvimento já são questões mais complexas.

Um ponto a se destacar é o tempo total de desenvolvimento do projeto tanto para DSP quanto para FPGA – incluindo aqui projeto do sistema, documentação e codificação propriamente dita. Destacando apenas a parte de codificação, entre as primeiras versões elaboradas e as finais de cada dispositivo, houve uma diferença perceptível em dias de trabalho. Para o projeto em DSP, foram gastos aproximadamente 15 dias (de trabalho para codificação) enquanto que para FPGA foram 40 dias (somente codificação).

Isso pode se explicado por duas razões: primeiro, porque a linguagem de programação C é bem mais conhecida e de uso cotidiano do aluno do que VHDL. Durante a graduação, foram desenvolvidos muito mais aplicativos em C para várias disciplinas do que trabalhos em VHDL. Mesmo no estágio do aluno, o trabalho com C foi rotineiro, enquanto que VHDL ficou apenas durante a graduação. Segundo, a quantidade de disciplinas voltadas para o desenvolvimento com microcontroladores é bem maior na grade disciplinar do curso do que as que envolvem VHDL. Isso implica em um maior condicionamento ao trabalho com registradores e instruções previamente programadas (como nos microcontroladores) do que no desenvolvimento de processos que trabalham em paralelo no sistema (FPGAs).

Ainda sobre as características particulares de cada tecnologia, experimentalmente tentou-se quantificar a portabilidade do código produzido para ambos os dispositivos. Observou-se que, no caso do DSP, haveria certo trabalho para adaptar o código produzido para o *chip* de outro fabricante, devido a todo o mapeamento de portas de entrada e saída e funções internas, como *timers*, por exemplo. Já no caso de FPGA, o código poderia ser recompilado tranquilamente para outro fabricante, com adaptações mínimas (inclusive este teste foi feito, quando o projeto desenvolvido para um dispositivo Xilinx no *software* ISE foi recompilado no ambiente de desenvolvimento Quartus II, da Altera Corporation, outra grande fabricante de FPGAs). Assim, notou-se que a portabilidade de um código em VHDL é maior do que a de um código em C para um DSP específico.

Sobre o sistema proposto, este procurou ser o mais fiel possível à realidade de um sistema automotivo, e a unidade de controle foi desenvolvida pensando justamente em uma aplicação de natureza real, e não somente simulada. Obviamente, este modelo precisa ser aperfeiçoado para que testes em ambientes reais possam ser feitos. Entretanto, acredita-se que um primeiro passo foi dado na direção de construir um sistema que possa atender às necessidades atuais de controle de motores de combustão, e outros nos quais esta tecnologia possa ser aplicada.

4.2. Sugestões de próximos trabalhos

Como foi dito, o primeiro passo foi dado ao construir este sistema de detecção de ciclos de motor, que pode ser aplicado diretamente no controle de sistemas automotivos e outros que utilizem o mesmo conceito. Entretanto, muito pode ser feito para que este sistema se transforme de fato em algo concreto, e que possa ser testado em uma bancada de motores, por exemplo.

Algumas das funcionalidades deste trabalho, por exemplo, como a exibição da rotação ou mesmo o sinal de controle da borboleta de admissão de ar, não foram implementadas em DSP, devido a pouca disponibilidade do kit de desenvolvimento. Por isso, um dos primeiros passos é a aquisição de mais materiais para teste dos sistemas que podem ser implementados tanto em DSP quando em FPGA – e não apenas um kit de cada dispositivo. Até mesmo para que se testem outras abordagens dentro da mesma tecnologia.

Assim, é possível dar prosseguimento à pesquisa iniciada com este trabalho de conclusão de curso, com o intuito de aprimorar este sistema, através de um mestrado ou especialização.

5. Conclusão

O mapeamento das funcionalidades de um controlador eletrônico automotivo permitiu elaborar um sistema que simulasse de maneira simplificada seu funcionamento, assim como um conjunto de blocos que trabalharam como um motor virtual.

Através deste modelo, verificou-se que tanto a utilização de um processador digital de sinais quanto de um FPGA é possível para a aplicação em questão, desde que sejam consideradas as características particulares do projeto. Obviamente, cada dispositivo possui fatores que facilitam ou dificultam a construção do sistema. Entretanto, mesmo os pontos negativos podem ser contornados sem que haja grandes problemas.

Além disso, foi possível perceber que é bem possível aplicar a mesma arquitetura de controle estudada neste trabalho em outras áreas que, aparentemente, não têm muita relação com a automotiva. Como no exemplo citado no início do trabalho, agricultura de precisão e controle de máquinas de usinagem, por exemplo, podem ser realizados com o emprego dos mesmos princípios das unidades eletrônicas de controle automotivo.

Sendo assim, conclui-se que o trabalho foi bem aproveitado por parte do aluno, absorvendo os conselhos dados por seu orientador, e colocando em prática os conceitos absorvidos durante toda a graduação.

6. Referências Bibliográficas

[1] Internet. Intel.com **Engine Control Overview**. http://www.intel.com/design/auto/engback.htm Acessado em Janeiro de 2008.

[2] Internet. Hitachi **Automotive Systems History** http://www.hitachi.co.jp/Div/apd/en/vision/vision_002.html Acessado em Janeiro de 2008.

[3] Internet. Wikipedia. Engine Control Unit. http://en.wikipedia.org/wiki/Engine_Control_Unit#History Acessado em Janeiro de 2008.

[4] ALONSO, GCMB. **Simulador de Ambiente Automotivo para Injeções Eletrônicas**. 2004, Dissertação de Mestrado. Faculdade de Engenharia Elétrica e Computação. Universidade de Campinas, Campinas, SP.

[5] DESARKAR, M. S. et al. **Case Study of Design of an Engine Control Unit.** Department of Computer Science and Engineering, Indian Institute of Technology Kanpur, 2004.

[6] MILHOR, C.E. **Sistema de Desenvolvimento para Controle Eletrônico dos Motores de Combustão Interna Ciclo Otto**. 2002, 72 p, Dissertação (Mestrado) SP – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos (SP).

[7] BOSCH, R. GmbH. **Gasoline-Engine Management**. 1st Edition, Stuttgart, Germany, 1999.

[8] RIBBENS, W.B. **Understanding Automotive Electronics**. 5th Edition, Sams Publishing, 1995, 434p.

[9] TOYOTA, **Fuel Systems Overview**. 2001, Toyota Motor Sales, USA, Inc.

[10] Internet. Wikipedia, **Hall Effect**, http://en.wikipedia.org/wiki/Hall_effect Acessado em janeiro de 2008.

[11] TOYOTA, **Fuel Systems: Injection Duration Controls,** 2001, Toyota Motor Sales, USA, Inc.

[12] SMITH, S.W. **The Scientist and Engineer's Guide to Digital Signal Processing**, 2001, Chapter 28.

[13] EYRE, J.; BIER, J. **The Evolution of DSP Processors**. Berkeley Design Technology Inc. White Paper, 2000.

[14] LERNER, B. **Fixed vs. Floating Point: A Surprisingly Hard Choice**. Analog Devices, 2007.

[15] Internet, Wikipedia, http://en.wikipedia.org/wiki/Digital_signal_processor Acessado em janeiro de 2008.

[16] MENDONÇA, A., ZELENOVSKY, R. **Projetos com FPGA: Famílias Modernas**. Internet. Disponível em <http://mzeditora.com.br/artigos/ fpga_fam.htm> Acessado em junho de 2007.

[17] PORRMANN, M; PAIZ, C. **The Utilization of Reconfigurable Hardware to Implement Digital Controllers: a Review**, 2007, Heinz Nixdorf Institute, University of Paderborn.

[18] FPGA. Internet, Wikipedia. <http://pt.wikipedia.org/wiki/FPGA> Acessado em janeiro de 2007.

[19] TEXAS Instruments. TMS320F2812 Tutorial, 2004.

[20] XILINX, **Spartan-3 FPGA Family: Complete Data Sheet**, 2007, Xilinx Inc.

7. Bibliografia Consultada

CHUJO, N **Fail-Safe ECU System Using Dynamic Reconfiguration of FPGA**. R&D Review of Toyota CRLD, Vol. 37, Nº 2, 2002, p. 54-60.

GAMBIER, A. **Real-Time Control Systems: A Tutorial**. Automation Laboratory, University of Mannhein, Alemanha, 2004.

OLIVEIRA, R. S. **Sistemas de Tempo Real. Departamento de Automação e Sistemas**, Universidade Federal de Santa Catarina, Florianópolis, 2000.

SIMON, D.E. **An Embedded Software Primer**, Addison Wesley, 1999, 448 p.

ALTERA Corporation. **FPGA vs. DSP Design Reliability and Maintenance**. White Paper, 2007.

ALTERA. Internet. Disponível em <http://www.altera.com/end-markets/auto/network/aut-network.html> Acessado em junho de 2007.

ANALOG Devices. Internet. Disponível em <http://www.analog.com/processors/index.html> Acessado em junho de 2007.

BALLUCHI, A. et al. **Automotive Engine Control and Hybrid Systems: Challenges and Opportunities**. Proceedings of the IEEE, vol. 88, "Special Issue on Hybrid Systems" (invited paper), no. 7, July 2000, p. 888-912.

COSTA, C. **Projetando Controladores Digitais com FPGA**. Editora Novatec, 2006, 159 p.

CUATTO, T. et al. **A Case Study in Embedded System Design: an Engine Control Unit**. Dep. of Electrical Engineering and Computer Science, University of California at Berkeley, San Francisco, USA, 1998, p. 804-807.

DASE, C. et al. **Motorcycle Control Prototyping Using an FPGA-Based Embedded Control System**. IEEE Control Systems Magazine, October 2006.

DSP. Internet (Wikipedia). Disponível em <http://pt.wikipedia.org/wiki/DSP> Acessado em janeiro de 2007.

HANSELMANN, H. **DSP in Control: The Total Development Environment**. IEEE IECON 22nd International Conference on Industrial Electronics, Control, and Instrumentation, DSPACE GmbH, Paderborn, Germany, 1996, p. 1647-1654.

HUNT Engineering. **Choosing DSP or FPGA for your Application**. Press Release, 2002.

KÄLLSTRÖN, F.J. **Embedded Software for New Engine Controller**. Department of Electrical Engineering, Linköping, 2005.

MARTINO, J.P. **Technological Forecasting for Decision Making**. 3rd Edition, McGraw-Hill Engineering and Technology Management Series, 1993, 462 p.

NUNES, R.A.A et al. **Introdução a Processadores de Sinais Digitais - DSP**. Nota Técnica CBPF-NT-001/2006.

PAPAIOANNOU, I.N.; MOSCATO, L. **Automotive Electronics in Brazil: Facts, Trends and Proposals**. 18th International Congress of Mechanical Engineering, Ouro Preto, MG, 2005.

8. Anexo

8.1 Códigos de implementação de alguns blocos em FPGA

Control_Injecao.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Control_Injecao is
   Port ( Clock            : in  STD_LOGIC;
         -- clock de 2KHz
                              Reset : in  STD_LOGIC;
         Rotacao : in  STD_LOGIC;
         Atuador : out STD_LOGIC_VECTOR(3 downto 0);
                              Saida3          : out std_logic_vector(3 downto 0);
                              Saida1          : out std_logic_vector(2 downto 0);
                              Saida2          : out std_logic_vector(2 downto 0));
end Control_Injecao;

architecture Behavioral of Control_Injecao is

         signal Numero                  : integer;
         signal Count1           : integer;
         signal Count2           : integer;
         signal CountClock       : integer;
         signal Periodo          : integer;
         signal Threshold        : integer;
         signal Inj                      : std_logic;

         -- funcao de divisao de dois numeros
         function Div (Num1 : integer) return integer is
                  variable quoc, rest : integer := 0;
         begin
                  rest := Num1;
                  while (rest >= 10) loop
                           rest := rest - 10;
                           quoc := quoc + 1;
                  end loop;

                  return quoc;
         end Div;

begin

         Periodo <= 10;                              -- para 12000 rpm, quando clock for 50Mhz

         process(Clock, Reset)
   begin
                  -- 10% de tolerancia de erro
```

```
		Threshold <= Div(Periodo);

		if (Reset = '1') then
			Count1 <= 0;
			Count2 <= 0;
			CountClock <= 0;
			Inj <= '0';
		elsif (Clock'event and Clock = '1') then
			if (CountClock < Periodo) then
				if (Rotacao = '1') then Count1 <= Count1 + 1; end if;
				if (Rotacao = '0') then Count2 <= Count2 + 1; end if;
				CountClock <= CountClock + 1;
				Atuador <= "0000";
			end if;
			if (CountClock = Periodo) then
				CountClock <= 1;
				Count1 <= 1;
				Count2 <= 0;
				if (abs(Count1 - Count2) > 2 * Threshold) then
					Atuador <= "0101";
					Inj <= '1';
				end if;
			end if;
			if ((Inj = '1') and (CountClock = Periodo - CountClock)) then
				Atuador <= "1010";
				Inj <= '0';
			end if;
		end if;
	end process;

	Saida1 <= conv_std_logic_vector(Count1, 3);
	Saida2 <= conv_std_logic_vector(Count2, 3);
	Saida3 <= conv_std_logic_vector(CountClock, 4);

end Behavioral;
```

ECU_Display.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ECU_Display is
	port (CLKIN	: in std_logic;
       Rotacao : in std_logic_vector(10 downto 0);
			AN3		: inout std_logic;
       AN2		: inout std_logic;
       AN1		: inout std_logic;
       AN0		: inout std_logic;
       LED		: out std_logic_vector(6 downto 0));
end ECU_Display;

architecture Behavioral of ECU_Display is
```

```
	signal CTR			: STD_LOGIC_VECTOR(12 downto 0);
	signal Valor		: integer;
	signal quoc		: integer;
	signal rest			: integer;
	signal unid, dez, cent, mil : integer;

	-- funcao que converte um numero decimal para BCD, para mostrar no display
	function Conv_BCD (Num1 : integer) return std_logic_vector is
		variable LED : std_logic_vector(6 downto 0);
	begin
		case Num1 is
			when 1 => LED := "1111001";
			when 2 => LED := "0100100";
			when 3 => LED := "0110000";
			when 4 => LED := "0011001";
			when 5 => LED := "0010010";
			when 6 => LED := "0000010";
			when 7 => LED := "1111000";
			when 8 => LED := "0000000";
			when 9 => LED := "0010000";
			when OTHERS => LED := "1000000";
		end case;
		return LED;
	end Conv_BCD;


begin

	Valor <= conv_integer(Rotacao);

	-- encontra os numeros que serao exibidos
	process (CLKIN)
	begin

		rest <= Valor;
		while (rest >= 10) loop			-- captura a unidade
			rest <= rest - 10;
			quoc <= quoc + 1;
		end loop;
		unid <= rest;				-- seta a unidade do numero

		if (quoc > 0) then
			rest <= quoc;
			quoc <= 0;
			while (rest >= 10) loop		-- captura a dezena
				rest <= rest - 10;
				quoc <= quoc + 1;
			end loop;
			dez <= rest;			-- seta a dezena do numero
		end if;

		if (quoc > 0) then
			rest <= quoc;
			quoc <= 0;
			while (rest >= 10) loop		-- captura a centena
```

```
                        rest <= rest - 10;
                        quoc <= quoc + 1;
                end loop;
                cent <= rest;                   -- seta a centena do numero
        end if;

        if (quoc > 0) then                      -- se ainda tiver numero para ser
dividido
                rest <= quoc;
                quoc <= 0;
                while (rest >= 10) loop         -- captura o milhar
                        rest <= rest - 10;
                        quoc <= quoc + 1;
                end loop;
                mil <= rest;                    -- seta o milhar do numero
        else mil <= 0;                          -- senao o milhar eh zero
        end if;

    end process;

    -- exibe os numeros no display
    Process (CLKIN)
    begin
        if CLKIN'event and CLKIN = '1' then
                if (CTR = "0000000000000") then
                        if (AN0='0') then
                                AN0 <= '1';
                                LED <= Conv_BCD(unid);    -- unidade
                                AN1 <= '0';
                        elsif (AN1='0') then
                                AN1 <= '1';
                                LED <= Conv_BCD(dez);     -- dezena
                                AN2 <= '0';
                        elsif (AN2='0') then
                                AN2 <= '1';
                                LED <= Conv_BCD(cent);    -- centena
                                AN3 <= '0';
                        elsif (AN3='0') then
                                AN3 <= '1';
                                LED <= Conv_BCD(mil);     -- milhar
                                AN0 <= '0';
                        end if;
                end if;

                CTR <= CTR + "0000000000001";
                if (CTR > "1000000000000") then
                        CTR <= "0000000000000";
                end if;

        end if;
    End Process;

End Behavioral;
```

Divisor_Clock.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all ;

entity Divisor_Clock is port (
                                        clock               : in  std_logic;        --
50Mhz
                                        clock_2M            : out std_logic;
        -- 02Mhz
                                        clock_2K            : out std_logic;
        -- 02Khz
                                        clock_500      : out std_logic);            --
500Hz
end Divisor_Clock;

architecture Behavioral of Divisor_Clock is

        signal Aux1  : std_logic;
        signal Count : integer range 0 to 24999;
        signal Count2: integer range 0 to 999;
        signal Count3: integer range 0 to 3999;

begin

        -- generates a 2 Mhz signal from a 50 Mhz signal
        process (clock)
        begin
        if clock'event and clock = '1' then
                Count <= Count + 1;
                if Count < 12500 then
                        clock_2M <= '1';
                        Aux1 <= '1';
                else
                        clock_2M <= '0';
                        Aux1 <= '0';
                end if ;
                if Count = 24999 then Count <= 0;
                end if;
        end if;
        end process;

        -- generates a 2 Khz signal from a 2 Mhz signal
        process (Aux1)
        begin
        if Aux1'event and Aux1 = '1' then
                Count2 <= Count2 + 1;
                if Count2 < 500 then
                        clock_2K <= '1';
                else
                        clock_2K <= '0';
                end if ;
                if Count2 = 999 then Count2 <= 0;
                end if;
```

```
        end if;
        end process;

        -- generates a 500Hz signal from a 2 Mhz signal
        process (Aux1)
        begin
        if Aux1'event and Aux1 = '1' then
                Count3 <= Count3 + 1;
                if Count3 < 2000 then
                        clock_500 <= '1';
                else
                        clock_500 <= '0';
                end if ;
                if Count3 = 3999 then Count3 <= 0;
                end if;
        end if;
        end process;

end architecture;
```

Controlador_Memoria.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Controlador_Memoria is
   Port ( Clock : in  STD_LOGIC;
        Passo : in  STD_LOGIC_VECTOR (11 downto 0);
        Index : out  STD_LOGIC_VECTOR (9 downto 0));
end Controlador_Memoria;

architecture Behavioral of Controlador_Memoria is

        signal Numero : integer;
        signal Indice : integer;
        signal IndiceParcial : integer;
        signal Parametrico : integer;

        -- funcao de divisao de dois numeros
        function Divisao (Num1, Num2 : integer) return integer is
                variable quoc, rest : integer := 0;
        begin
                rest := Num1;
                while (rest >= Num2) loop
                        rest := rest - Num2;
                        quoc := quoc + 1;
                end loop;

                return quoc;
        end Divisao;


begin
```

```
	Numero <= conv_integer(Passo);

	process(Clock)
begin
		if (Clock'event and Clock = '1') then

			if (Numero >= 0 and Numero <= 160) then
			Parametrico <= Numero;
				IndiceParcial <= Divisao (Parametrico, 16);
				Indice <= IndiceParcial;
			elsif (Numero > 160 and Numero <= 310) then
			Parametrico <= Numero - 160;
				IndiceParcial <= Divisao (Parametrico, 15);
				Indice <= IndiceParcial + 10;
			elsif (Numero > 310 and Numero <= 450) then
			Parametrico <= Numero - 310;
				IndiceParcial <= Divisao (Parametrico, 14);
				Indice <= IndiceParcial + 20;
			elsif (Numero > 450 and Numero <= 580) then
			Parametrico <= Numero - 450;
				IndiceParcial <= Divisao (Parametrico, 13);
				Indice <= IndiceParcial + 30;
			elsif (Numero > 580 and Numero <= 700) then
			Parametrico <= Numero - 580;
				IndiceParcial <= Divisao (Parametrico, 12);
				Indice <= IndiceParcial + 40;
			elsif (Numero > 700 and Numero <= 810) then
			Parametrico <= Numero - 700;
				IndiceParcial <= Divisao (Parametrico, 11);
				Indice <= IndiceParcial + 50;
			elsif (Numero > 810 and Numero <= 910) then
			Parametrico <= Numero - 810;
				IndiceParcial <= Divisao (Parametrico, 10);
				Indice <= IndiceParcial + 60;
			elsif (Numero > 910 and Numero <= 1000) then
			Parametrico <= Numero - 910;
				IndiceParcial <= Divisao (Parametrico, 9);
				Indice <= IndiceParcial + 70;
			elsif (Numero > 1000 and Numero <= 1080) then
			Parametrico <= Numero - 1000;
				IndiceParcial <= Divisao (Parametrico, 8);
				Indice <= IndiceParcial + 80;
			elsif (Numero > 1080 and Numero <= 1150) then
		Parametrico <= Numero - 1080;
				IndiceParcial <= Divisao (Parametrico, 7);
				Indice <= IndiceParcial + 90;
			elsif (Numero > 1150 and Numero <= 1210) then
		Parametrico <= Numero - 1150;
				IndiceParcial <= Divisao (Parametrico, 6);
				Indice <= IndiceParcial + 100;
			elsif (Numero > 1210 and Numero <= 1260) then
		Parametrico <= Numero - 1210;
				IndiceParcial <= Divisao (Parametrico, 5);
				Indice <= IndiceParcial + 110;
```

```
                elsif (Numero > 1260 and Numero <= 1300) then
            Parametrico <= Numero - 1260;
                    IndiceParcial <= Divisao (Parametrico, 4);
                    Indice <= IndiceParcial + 120;
                elsif (Numero > 1300 and Numero <= 1330) then
            Parametrico <= Numero - 1300;
                    IndiceParcial <= Divisao (Parametrico, 3);
                    Indice <= IndiceParcial + 130;
                elsif (Numero > 1330 and Numero <= 1350) then
            Parametrico <= Numero - 1330;
                    IndiceParcial <= Divisao (Parametrico, 2);
                    Indice <= IndiceParcial + 140;
                elsif (Numero > 1350 and Numero < 2200) then
                    Indice <= Numero - 1200;
                end if;

            end if;
        end process;

        Index <= conv_std_logic_vector(Indice, 10);

end Behavioral;
```

Contador.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use ieee.numeric_std.ALL;

entity Contador is
        Port ( clock            : in  STD_LOGIC;
        -- clock de 500Hz
                        reset           : in  STD_LOGIC;
       passo            : out STD_LOGIC_VECTOR (11 downto 0);
       acelerador : in  STD_LOGIC);
end Contador;

architecture Behavioral of Contador is

        signal Count : integer range 0 to 2050;

begin

        process(clock, acelerador, reset)
  begin
        if reset = '1' then Count <= 0;
        elsif (clock='1' and clock'event) then
            -- se o botao de aceleracao estiver pressionado
            if acelerador = '1' then
                Count <= Count + 1;
            elsif (Count /= 0 and acelerador = '0') then
                Count <= Count - 1;
            end if;
```

```
        end if;
        end process;

        passo <= conv_std_logic_vector(Count, 12);

end Behavioral;
```

ECU_ROM.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ECU_ROM is
   Port ( Clock : in  STD_LOGIC;
          Reset : in  STD_LOGIC;
          Enable : in  STD_LOGIC;
          Address : in  STD_LOGIC_VECTOR (4 downto 0);
          Frequencia : out  STD_LOGIC_VECTOR (5 downto 0));
end ECU_ROM;

architecture Behavioral of ECU_ROM is

        type ECU_ROM_Array is array (0 to 31) of std_logic_vector(5 downto 0);

        -- vetor que armazena os valores que sao utilizados pelo controlador da injecao
  constant Content: ECU_ROM_Array := (
                0 => "000010", 1 => "000100", 2 => "000110",
                3 => "001000", 4 => "001010", 5 => "001100",
                6 => "001110",          7 => "010000", 8 => "010010",
                9 => "010100",          10 => "010110", 11 => "011000",
                12 => "011010", 13 => "011100", 14 => "011110",
                15 => "100000", 16 => "100010", 17 => "100100",
                18 => "100110", 19 => "101000", 20 => "101010",
                21 => "101100", 22 => "101110", 23 => "110000",
                24 => "110010", 25 => "110100", 26 => "110110",
                27 => "111000", 28 => "111010", 29 => "111100",
                30 => "111110", OTHERS => "111111"
        );

begin

        process(Clock, Reset, Address)
  begin
                if (Reset = '1') then
                        Frequencia <= "000000";
    elsif (Clock'event and Clock = '1') then
                        if (Enable = '1') then
                                Frequencia <= Content(conv_integer(Address));
      else
        Frequencia <= "000000";
      end if;
                end if;
        end process;
```

```
end Behavioral;
```

Gera_Frequencia.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Gera_Frequencia is
	Port (Clock		: in	std_logic;
			Reset		: in	std_logic;
			Teste		: out std_logic_vector(3 downto 0);
			Periodo		: in	std_logic_vector(9 downto 0);
			Saida		: out	std_logic);
end Gera_Frequencia;

architecture Behavioral of Gera_Frequencia is

	signal Max		: integer;
	signal Duty		: integer;
	signal Count	: integer;
	signal Control : boolean;

	-- funcao de divisao de dois numeros
	function Divisao (Num1 : integer) return integer is
		variable quoc, rest : integer := 0;
	begin
		rest := Num1;
		while (rest >= 2) loop
			rest := rest - 2;
			quoc := quoc + 1;
		end loop;

		return quoc;
	end Divisao;

begin

	process(Clock)
	begin
		if (Reset = '1') then
			Count <= 0;
			Saida <= '0';
			Duty <= 0;
		elsif (Clock'event and Clock = '1') then
			-- converte o valor do periodo para inteiro
			Max <= conv_integer(Periodo);

			-- enquanto for menor que o Duty, parte alta do periodo
			if (Count < Periodo) then
				if (Count < Periodo - Count) then
					Saida <= '1';
				else Saida <= '0';
```

```
                end if;
                Count <= Count + 1;
            end if;
            -- zera o contador e permite que uma nova leitura de periodo
seja feita
            if (Count = Periodo) then
                Count <= 0;
            end if;

            Teste <= conv_std_logic_vector(Max, 4);

        end if;

    end process;

end Behavioral;
```

8.2 Códigos de implementação em C para o DSP

Main.c

```
#include "DSP281x_Device.h"
#include "square.h"


// Prototype statements for functions found within this file.

void Gpio_select(void);
void SpeedUpRevA(void);
void InitSystem(void);

interrupt void cpu_timer0_isr(void); // Prototype for Timer 0 Interrupt Service Routine

void main(void)
{
    int Count1 = 0;
    int Count2 = 0;
    int i, Erro = 10;


    InitSystem();       // Initialize the DSP's core Registers


    Gpio_select();      // Setup the GPIO Multiplex Registers

    InitPieCtrl();      // Function Call to init PIE-unit ( code :
DSP281x_PieCtrl.c)

    InitPieVectTable(); // Function call to init PIE vector table ( code :
DSP281x_PieVect.c )
```

```
	// re-map PIE - entry for Timer 0 Interrupt
	EALLOW;  // This is needed to write to EALLOW protected registers
	PieVectTable.TINT0 = &cpu_timer0_isr;
	EDIS;    // This is needed to disable write to EALLOW protected registers

	InitCpuTimers();

	// Configure CPU-Timer 0 to interrupt every 50 ms:
	// 150MHz CPU Freq, 50000 µseconds interrupt period
ConfigCpuTimer(&CpuTimer0, 150, 1000);

// Enable TINT0 in the PIE: Group 1 interrupt 7
	PieCtrlRegs.PIEIER1.bit.INTx7 = 1;

	// Enable CPU INT1 which is connected to CPU-Timer 0:
IER = 1;

	// Enable global Interrupts and higher priority real-time debug events:
	EINT;   // Enable Global interrupt INTM
	ERTM;   // Enable Global realtime interrupt DBGM

	CpuTimer0Regs.TCR.bit.TSS = 0;

	EALLOW;
	SysCtrlRegs.WDCR=0x0068;
	EDIS;

	while(1)
	{
		for(i=0;i<50;i++){

				if(duty60[i]==0)
					Count1++;
				if(duty60[i]==1)
					Count2++;
				GpioDataRegs.GPBDAT.bit.GPIOB1=duty60[i];
				while (CpuTimer0.InterruptCount<1);
				CpuTimer0.InterruptCount=0;
		}
		if (abs(Count2 - Count1) > Erro)
			GpioDataRegs.GPBDAT.bit.GPIOB0  =1;
		else GpioDataRegs.GPBDAT.bit.GPIOB0  =0;


		Count1 = 0;
		Count2 = 0;

		for(i=0;i<50;i++){

				if(duty60[i+50]==0)
					Count1++;
				if(duty60[i+50]==1)
					Count2++;
			GpioDataRegs.GPBDAT.bit.GPIOB1=duty60[i+50];
```

```
                        while (CpuTimer0.InterruptCount<1);
                        CpuTimer0.InterruptCount=0;
                }
                if (abs(Count2 - Count1) > Erro)
                        GpioDataRegs.GPBDAT.bit.GPIOB0  =1;
                else GpioDataRegs.GPBDAT.bit.GPIOB0  =0;

                Count1 = 0;
                Count2 = 0;

        for(i=0;i<50;i++){

                                if(duty60[i+100]==0)
                                        Count1++;
                                if(duty60[i+100]==1)
                                        Count2++;
                        GpioDataRegs.GPBDAT.bit.GPIOB1=duty60[i+100];
                        while (CpuTimer0.InterruptCount<1);
                        CpuTimer0.InterruptCount=0;
                }
                if (abs(Count2 - Count1) > Erro)
                        GpioDataRegs.GPBDAT.bit.GPIOB0  =1;
                else GpioDataRegs.GPBDAT.bit.GPIOB0  =0;

                Count1 = 0;
                Count2 = 0;

        for(i=0;i<50;i++){

                                if(duty60[i+150]==0)
                                        Count1++;
                                if(duty60[i+150]==1)
                                        Count2++;
                        GpioDataRegs.GPBDAT.bit.GPIOB1=duty60[i+150];
                        while (CpuTimer0.InterruptCount<1);
                        CpuTimer0.InterruptCount=0;
                }
                if (abs(Count2 - Count1) > Erro)
                        GpioDataRegs.GPBDAT.bit.GPIOB0  =1;
                else GpioDataRegs.GPBDAT.bit.GPIOB0  =0;

                Count1 = 0;
                Count2 = 0;
                //EALLOW;
        //      SysCtrlRegs.WDKEY = 0xAA;               // and serve watchdog #2

        //EDIS;
    }
}

void Gpio_select(void)
{
        EALLOW;
        GpioMuxRegs.GPAMUX.all = 0x0;   // all GPIO port Pin's to I/O
    GpioMuxRegs.GPBMUX.all = 0x0;
```

```
   GpioMuxRegs.GPDMUX.all = 0x0;
   GpioMuxRegs.GPFMUX.all = 0x0;
   GpioMuxRegs.GPEMUX.all = 0x0;
   GpioMuxRegs.GPGMUX.all = 0x0;

   GpioMuxRegs.GPADIR.all = 0x0; // GPIO PORT  as input
   GpioMuxRegs.GPBDIR.all = 0x00FF;      // GPIO Port B15-B8 input , B7-B0 output
   GpioMuxRegs.GPDDIR.all = 0x0; // GPIO PORT  as input
   GpioMuxRegs.GPEDIR.all = 0x0; // GPIO PORT  as input
   GpioMuxRegs.GPFDIR.all = 0x0; // GPIO PORT  as input
   GpioMuxRegs.GPGDIR.all = 0x0; // GPIO PORT  as input

   GpioMuxRegs.GPAQUAL.all = 0x0;        // Set GPIO input qualifier values to zero
   GpioMuxRegs.GPBQUAL.all = 0x0;
   GpioMuxRegs.GPDQUAL.all = 0x0;
   GpioMuxRegs.GPEQUAL.all = 0x0;
   EDIS;
}

void InitSystem(void)
{
        EALLOW;
        SysCtrlRegs.WDCR= 0x00AF;                    // Setup the watchdog
                                                           // 0x00E8   to disable
the Watchdog , Prescaler = 1
                                                           //  0x00AF    to  NOT
disable the Watchdog, Prescaler = 64
        SysCtrlRegs.SCSR = 0;                        // Watchdog generates a RESET
        SysCtrlRegs.PLLCR.bit.DIV = 10;    // Setup the Clock PLL to multiply by 5

        SysCtrlRegs.HISPCP.all = 0x1; // Setup Highspeed Clock Prescaler to divide by
2
        SysCtrlRegs.LOSPCP.all = 0x2; // Setup Lowspeed CLock Prescaler to divide
by 4

        // Peripheral clock enables set for the selected peripherals.
        SysCtrlRegs.PCLKCR.bit.EVAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.EVBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIAENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SCIBENCLK=0;
        SysCtrlRegs.PCLKCR.bit.MCBSPENCLK=0;
        SysCtrlRegs.PCLKCR.bit.SPIENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ECANENCLK=0;
        SysCtrlRegs.PCLKCR.bit.ADCENCLK=0;
        EDIS;
}

interrupt void cpu_timer0_isr(void)
{
   CpuTimer0.InterruptCount++;
        // Serve the watchdog every Timer 0 interrupt
        //EALLOW;
//      SysCtrlRegs.WDKEY = 0x55;                    // Serve watchdog #1
        //EDIS;
```

```
  // Acknowledge this interrupt to receive more interrupts from group 1
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}
```

Square.c

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

int main() {
                                   //a variavel utilizada pelo programa do TURBOC
int duty[200],i;
double m=0.95;                          //o indice de modulacao em amplitude da onda PWM p/ modular em amplitude   0<m<1
int PWMmax=50;                          //resolucao maxima (numero de pontos por periodo em determinada frequencia) do timer2

PWMmax/=2;

FILE *ofp;

ofp=fopen("square.h","w");    //gera o arquivo


fprintf(ofp,"\n//duty p/ m = %d",m);             //imprime no arquivo
fprintf(ofp,"\nconst int duty60[200] = {");     //imprime no arquivo

for(i=0;i<50;i++){
   //imprime no arquivo
         if(i<=24)
                  fprintf(ofp,"1,\n",duty[i]); //imprime no arquivo
   if(i>24)
         fprintf(ofp,"0,\n",duty[i]); //imprime no arquivo
}

for(i=0;i<50;i++){
   //imprime no arquivo
         if(i<=24)
                  fprintf(ofp,"1,\n",duty[i+50]); //imprime no arquivo
   if(i>24)
         fprintf(ofp,"0,\n",duty[i+50]); //imprime no arquivo
}

for(i=0;i<50;i++){
   //imprime no arquivo
         if(i<=24)
                  fprintf(ofp,"1,\n",duty[i+100]); //imprime no arquivo
   if(i>24)
         fprintf(ofp,"0,\n",duty[i+100]); //imprime no arquivo
}

for(i=0;i<50;i++){
         if(i==49)
                  fprintf(ofp,"0};",duty[i+150]);    //imprime no arquivo
```

```
	if(i<=24)
		fprintf(ofp,"1,\n",duty[i+150]); //imprime no arquivo
   if(i>24 && i!=49)
	 fprintf(ofp,"0,\n",duty[i+150]); //imprime no arquivo
}

fprintf(ofp,"\nconst int duty50[200] = {");     //imprime no arquivo

for(i=0;i<200;i++){
	duty[i]=(int)(m*PWMmax*sin(0.0314159265*i)+PWMmax);
	if(i==199)
		fprintf(ofp,"%d};",duty[i]);    //imprime no arquivo
	else
		fprintf(ofp,"%d,\n",duty[i]);   //imprime no arquivo

}


fclose(ofp);

return 0;
}
```

ROM.vhd

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity ROM is
	Port ( Clock     : in  STD_LOGIC;
	     Reset     : in  STD_LOGIC;
	     Enable   : in  STD_LOGIC;
	     Address  : in  STD_LOGIC_VECTOR (9 downto 0);
	     Data_out : out STD_LOGIC_VECTOR (9 downto 0));
end ROM;

architecture Behavioral of ROM is

	type ROM_Array is array (0 to 999) of std_logic_vector(9 downto 0);

	-- vetor com 1000 posicoes, que armazena os valores discretos de rotacao do
motor
  constant Content: ROM_Array := (
	0 => "0000000000", 1 => "0000000001", 2 => "0000000010",
	3 => "0000000011", 4 => "0000000100", 5 => "0000000101",
	6 => "0000000110", 7 => "0000000111", 8 => "0000001000",
	9 => "0000001001", 10 => "0000001010", 11 => "0000001011",
	12 => "0000001100", 13 => "0000001101", 14 => "0000001110",
	15 => "0000001111", 16 => "0000010000", 17 => "0000010001",
	18 => "0000010010", 19 => "0000010011", 20 => "0000010100",
	21 => "0000010101", 22 => "0000010110", 23 => "0000010111",
	24 => "0000011000", 25 => "0000011001", 26 => "0000011010",
```

```
27 => "0000011011", 28 => "0000011100", 29 => "0000011101",
30 => "0000011110", 31 => "0000011111", 32 => "0000100000",
33 => "0000100001", 34 => "0000100010", 35 => "0000100011",
36 => "0000100100", 37 => "0000100101", 38 => "0000100110",
39 => "0000100111", 40 => "0000101000", 41 => "0000101001",
42 => "0000101010", 43 => "0000101011", 44 => "0000101100",
45 => "0000101101", 46 => "0000101110", 47 => "0000101111",
48 => "0000110000", 49 => "0000110001", 50 => "0000110010",
51 => "0000110011", 52 => "0000110100", 53 => "0000110101",
54 => "0000110110", 55 => "0000110111", 56 => "0000111000",
57 => "0000111001", 58 => "0000111010", 59 => "0000111011",
60 => "0000111100", 61 => "0000111101", 62 => "0000111110",
63 => "0000111111", 64 => "0001000000", 65 => "0001000001",
66 => "0001000010", 67 => "0001000011", 68 => "0001000100",
69 => "0001000101", 70 => "0001000110", 71 => "0001000111",
72 => "0001001000", 73 => "0001001001", 74 => "0001001010",
75 => "0001001011", 76 => "0001001100", 77 => "0001001101",
78 => "0001001110", 79 => "0001001111", 80 => "0001010000",
81 => "0001010001", 82 => "0001010010", 83 => "0001010011",
84 => "0001010100", 85 => "0001010101", 86 => "0001010110",
87 => "0001010111", 88 => "0001011000", 89 => "0001011001",
90 => "0001011010", 91 => "0001011011", 92 => "0001011100",
93 => "0001011101", 94 => "0001011110", 95 => "0001011111",
96 => "0001100000", 97 => "0001100001", 98 => "0001100010",
99 => "0001100011", 100 => "0001100100", 101 => "0001100101",
102 => "0001100110", 103 => "0001100111", 104 => "0001101000",
105 => "0001101001", 106 => "0001101010", 107 => "0001101011",
108 => "0001101100", 109 => "0001101101", 110 => "0001101110",
111 => "0001101111", 112 => "0001110000", 113 => "0001110001",
114 => "0001110010", 115 => "0001110011", 116 => "0001110100",
117 => "0001110101", 118 => "0001110110", 119 => "0001110111",
120 => "0001111000", 121 => "0001111001", 122 => "0001111010",
123 => "0001111011", 124 => "0001111100", 125 => "0001111101",
126 => "0001111110", 127 => "0001111111", 128 => "0010000000",
129 => "0010000001", 130 => "0010000010", 131 => "0010000011",
132 => "0010000100", 133 => "0010000101", 134 => "0010000110",
135 => "0010000111", 136 => "0010001000", 137 => "0010001001",
138 => "0010001010", 139 => "0010001011", 140 => "0010001100",
141 => "0010001101", 142 => "0010001110", 143 => "0010001111",
144 => "0010010000", 145 => "0010010001", 146 => "0010010010",
147 => "0010010011", 148 => "0010010100", 149 => "0010010101",
150 => "0010010110", 151 => "0010010111", 152 => "0010011000",
153 => "0010011001", 154 => "0010011010", 155 => "0010011011",
156 => "0010011100", 157 => "0010011101", 158 => "0010011110",
159 => "0010011111", 160 => "0010100000", 161 => "0010100001",
162 => "0010100010", 163 => "0010100011", 164 => "0010100100",
165 => "0010100101", 166 => "0010100110", 167 => "0010100111",
168 => "0010101000", 169 => "0010101001", 170 => "0010101010",
171 => "0010101011", 172 => "0010101100", 173 => "0010101101",
174 => "0010101110", 175 => "0010101111", 176 => "0010110000",
177 => "0010110001", 178 => "0010110010", 179 => "0010110011",
180 => "0010110100", 181 => "0010110101", 182 => "0010110110",
183 => "0010110111", 184 => "0010111000", 185 => "0010111001",
186 => "0010111010", 187 => "0010111011", 188 => "0010111100",
189 => "0010111101", 190 => "0010111110", 191 => "0010111111",
```

```
192 => "0011000000", 193 => "0011000001", 194 => "0011000010",
195 => "0011000011", 196 => "0011000100", 197 => "0011000101",
198 => "0011000110", 199 => "0011000111", 200 => "0011001000",
201 => "0011001001", 202 => "0011001010", 203 => "0011001011",
204 => "0011001100", 205 => "0011001101", 206 => "0011001110",
207 => "0011001111", 208 => "0011010000", 209 => "0011010001",
210 => "0011010010", 211 => "0011010011", 212 => "0011010100",
213 => "0011010101", 214 => "0011010110", 215 => "0011010111",
216 => "0011011000", 217 => "0011011001", 218 => "0011011010",
219 => "0011011011", 220 => "0011011100", 221 => "0011011101",
222 => "0011011110", 223 => "0011011111", 224 => "0011100000",
225 => "0011100001", 226 => "0011100010", 227 => "0011100011",
228 => "0011100100", 229 => "0011100101", 230 => "0011100110",
231 => "0011100111", 232 => "0011101000", 233 => "0011101001",
234 => "0011101010", 235 => "0011101011", 236 => "0011101100",
237 => "0011101101", 238 => "0011101110", 239 => "0011101111",
240 => "0011110000", 241 => "0011110001", 242 => "0011110010",
243 => "0011110011", 244 => "0011110100", 245 => "0011110101",
246 => "0011110110", 247 => "0011110111", 248 => "0011111000",
249 => "0011111001", 250 => "0011111010", 251 => "0011111011",
252 => "0011111100", 253 => "0011111101", 254 => "0011111110",
255 => "0011111111", 256 => "0100000000", 257 => "0100000001",
258 => "0100000010", 259 => "0100000011", 260 => "0100000100",
261 => "0100000101", 262 => "0100000110", 263 => "0100000111",
264 => "0100001000", 265 => "0100001001", 266 => "0100001010",
267 => "0100001011", 268 => "0100001100", 269 => "0100001101",
270 => "0100001110", 271 => "0100001111", 272 => "0100010000",
273 => "0100010001", 274 => "0100010010", 275 => "0100010011",
276 => "0100010100", 277 => "0100010101", 278 => "0100010110",
279 => "0100010111", 280 => "0100011000", 281 => "0100011001",
282 => "0100011010", 283 => "0100011011", 284 => "0100011100",
285 => "0100011101", 286 => "0100011110", 287 => "0100011111",
288 => "0100100000", 289 => "0100100001", 290 => "0100100010",
291 => "0100100011", 292 => "0100100100", 293 => "0100100101",
294 => "0100100110", 295 => "0100100111", 296 => "0100101000",
297 => "0100101001", 298 => "0100101010", 299 => "0100101011",
300 => "0100101100", 301 => "0100101101", 302 => "0100101110",
303 => "0100101111", 304 => "0100110000", 305 => "0100110001",
306 => "0100110010", 307 => "0100110011", 308 => "0100110100",
309 => "0100110101", 310 => "0100110110", 311 => "0100110111",
312 => "0100111000", 313 => "0100111001", 314 => "0100111010",
315 => "0100111011", 316 => "0100111100", 317 => "0100111101",
318 => "0100111110", 319 => "0100111111", 320 => "0101000000",
321 => "0101000001", 322 => "0101000010", 323 => "0101000011",
324 => "0101000100", 325 => "0101000101", 326 => "0101000110",
327 => "0101000111", 328 => "0101001000", 329 => "0101001001",
330 => "0101001010", 331 => "0101001011", 332 => "0101001100",
333 => "0101001101", 334 => "0101001110", 335 => "0101001111",
336 => "0101010000", 337 => "0101010001", 338 => "0101010010",
339 => "0101010011", 340 => "0101010100", 341 => "0101010101",
342 => "0101010110", 343 => "0101010111", 344 => "0101011000",
345 => "0101011001", 346 => "0101011010", 347 => "0101011011",
348 => "0101011100", 349 => "0101011101", 350 => "0101011110",
351 => "0101011111", 352 => "0101100000", 353 => "0101100001",
354 => "0101100010", 355 => "0101100011", 356 => "0101100100",
```

```
357 => "0101100101", 358 => "0101100110", 359 => "0101100111",
360 => "0101101000", 361 => "0101101001", 362 => "0101101010",
363 => "0101101011", 364 => "0101101100", 365 => "0101101101",
366 => "0101101110", 367 => "0101101111", 368 => "0101110000",
369 => "0101110001", 370 => "0101110010", 371 => "0101110011",
372 => "0101110100", 373 => "0101110101", 374 => "0101110110",
375 => "0101110111", 376 => "0101111000", 377 => "0101111001",
378 => "0101111010", 379 => "0101111011", 380 => "0101111100",
381 => "0101111101", 382 => "0101111110", 383 => "0101111111",
384 => "0110000000", 385 => "0110000001", 386 => "0110000010",
387 => "0110000011", 388 => "0110000100", 389 => "0110000101",
390 => "0110000110", 391 => "0110000111", 392 => "0110001000",
393 => "0110001001", 394 => "0110001010", 395 => "0110001011",
396 => "0110001100", 397 => "0110001101", 398 => "0110001110",
399 => "0110001111", 400 => "0110010000", 401 => "0110010001",
402 => "0110010010", 403 => "0110010011", 404 => "0110010100",
405 => "0110010101", 406 => "0110010110", 407 => "0110010111",
408 => "0110011000", 409 => "0110011001", 410 => "0110011010",
411 => "0110011011", 412 => "0110011100", 413 => "0110011101",
414 => "0110011110", 415 => "0110011111", 416 => "0110100000",
417 => "0110100001", 418 => "0110100010", 419 => "0110100011",
420 => "0110100100", 421 => "0110100101", 422 => "0110100110",
423 => "0110100111", 424 => "0110101000", 425 => "0110101001",
426 => "0110101010", 427 => "0110101011", 428 => "0110101100",
429 => "0110101101", 430 => "0110101110", 431 => "0110101111",
432 => "0110110000", 433 => "0110110001", 434 => "0110110010",
435 => "0110110011", 436 => "0110110100", 437 => "0110110101",
438 => "0110110110", 439 => "0110110111", 440 => "0110111000",
441 => "0110111001", 442 => "0110111010", 443 => "0110111011",
444 => "0110111100", 445 => "0110111101", 446 => "0110111110",
447 => "0110111111", 448 => "0111000000", 449 => "0111000001",
450 => "0111000010", 451 => "0111000011", 452 => "0111000100",
453 => "0111000101", 454 => "0111000110", 455 => "0111000111",
456 => "0111001000", 457 => "0111001001", 458 => "0111001010",
459 => "0111001011", 460 => "0111001100", 461 => "0111001101",
462 => "0111001110", 463 => "0111001111", 464 => "0111010000",
465 => "0111010001", 466 => "0111010010", 467 => "0111010011",
468 => "0111010100", 469 => "0111010101", 470 => "0111010110",
471 => "0111010111", 472 => "0111011000", 473 => "0111011001",
474 => "0111011010", 475 => "0111011011", 476 => "0111011100",
477 => "0111011101", 478 => "0111011110", 479 => "0111011111",
480 => "0111100000", 481 => "0111100001", 482 => "0111100010",
483 => "0111100011", 484 => "0111100100", 485 => "0111100101",
486 => "0111100110", 487 => "0111100111", 488 => "0111101000",
489 => "0111101001", 490 => "0111101010", 491 => "0111101011",
492 => "0111101100", 493 => "0111101101", 494 => "0111101110",
495 => "0111101111", 496 => "0111110000", 497 => "0111110001",
498 => "0111110010", 499 => "0111110011", 500 => "0111110100",
501 => "0111110101", 502 => "0111110110", 503 => "0111110111",
504 => "0111111000", 505 => "0111111001", 506 => "0111111010",
507 => "0111111011", 508 => "0111111100", 509 => "0111111101",
510 => "0111111110", 511 => "0111111111", 512 => "1000000000",
513 => "1000000001", 514 => "1000000010", 515 => "1000000011",
516 => "1000000100", 517 => "1000000101", 518 => "1000000110",
519 => "1000000111", 520 => "1000001000", 521 => "1000001001",
```

```
522 => "1000001010", 523 => "1000001011", 524 => "1000001100",
525 => "1000001101", 526 => "1000001110", 527 => "1000001111",
528 => "1000010000", 529 => "1000010001", 530 => "1000010010",
531 => "1000010011", 532 => "1000010100", 533 => "1000010101",
534 => "1000010110", 535 => "1000010111", 536 => "1000011000",
537 => "1000011001", 538 => "1000011010", 539 => "1000011011",
540 => "1000011100", 541 => "1000011101", 542 => "1000011110",
543 => "1000011111", 544 => "1000100000", 545 => "1000100001",
546 => "1000100010", 547 => "1000100011", 548 => "1000100100",
549 => "1000100101", 550 => "1000100110", 551 => "1000100111",
552 => "1000101000", 553 => "1000101001", 554 => "1000101010",
555 => "1000101011", 556 => "1000101100", 557 => "1000101101",
558 => "1000101110", 559 => "1000101111", 560 => "1000110000",
561 => "1000110001", 562 => "1000110010", 563 => "1000110011",
564 => "1000110100", 565 => "1000110101", 566 => "1000110110",
567 => "1000110111", 568 => "1000111000", 569 => "1000111001",
570 => "1000111010", 571 => "1000111011", 572 => "1000111100",
573 => "1000111101", 574 => "1000111110", 575 => "1000111111",
576 => "1001000000", 577 => "1001000001", 578 => "1001000010",
579 => "1001000011", 580 => "1001000100", 581 => "1001000101",
582 => "1001000110", 583 => "1001000111", 584 => "1001001000",
585 => "1001001001", 586 => "1001001010", 587 => "1001001011",
588 => "1001001100", 589 => "1001001101", 590 => "1001001110",
591 => "1001001111", 592 => "1001010000", 593 => "1001010001",
594 => "1001010010", 595 => "1001010011", 596 => "1001010100",
597 => "1001010101", 598 => "1001010110", 599 => "1001010111",
600 => "1001011000", 601 => "1001011001", 602 => "1001011010",
603 => "1001011011", 604 => "1001011100", 605 => "1001011101",
606 => "1001011110", 607 => "1001011111", 608 => "1001100000",
609 => "1001100001", 610 => "1001100010", 611 => "1001100011",
612 => "1001100100", 613 => "1001100101", 614 => "1001100110",
615 => "1001100111", 616 => "1001101000", 617 => "1001101001",
618 => "1001101010", 619 => "1001101011", 620 => "1001101100",
621 => "1001101101", 622 => "1001101110", 623 => "1001101111",
624 => "1001110000", 625 => "1001110001", 626 => "1001110010",
627 => "1001110011", 628 => "1001110100", 629 => "1001110101",
630 => "1001110110", 631 => "1001110111", 632 => "1001111000",
633 => "1001111001", 634 => "1001111010", 635 => "1001111011",
636 => "1001111100", 637 => "1001111101", 638 => "1001111110",
639 => "1001111111", 640 => "1010000000", 641 => "1010000001",
642 => "1010000010", 643 => "1010000011", 644 => "1010000100",
645 => "1010000101", 646 => "1010000110", 647 => "1010000111",
648 => "1010001000", 649 => "1010001001", 650 => "1010001010",
651 => "1010001011", 652 => "1010001100", 653 => "1010001101",
654 => "1010001110", 655 => "1010001111", 656 => "1010010000",
657 => "1010010001", 658 => "1010010010", 659 => "1010010011",
660 => "1010010100", 661 => "1010010101", 662 => "1010010110",
663 => "1010010111", 664 => "1010011000", 665 => "1010011001",
666 => "1010011010", 667 => "1010011011", 668 => "1010011100",
669 => "1010011101", 670 => "1010011110", 671 => "1010011111",
672 => "1010100000", 673 => "1010100001", 674 => "1010100010",
675 => "1010100011", 676 => "1010100100", 677 => "1010100101",
678 => "1010100110", 679 => "1010100111", 680 => "1010101000",
681 => "1010101001", 682 => "1010101010", 683 => "1010101011",
684 => "1010101100", 685 => "1010101101", 686 => "1010101110",
```

```
687 => "1010101111", 688 => "1010110000", 689 => "1010110001",
690 => "1010110010", 691 => "1010110011", 692 => "1010110100",
693 => "1010110101", 694 => "1010110110", 695 => "1010110111",
696 => "1010111000", 697 => "1010111001", 698 => "1010111010",
699 => "1010111011", 700 => "1010111100", 701 => "1010111101",
702 => "1010111110", 703 => "1010111111", 704 => "1011000000",
705 => "1011000001", 706 => "1011000010", 707 => "1011000011",
708 => "1011000100", 709 => "1011000101", 710 => "1011000110",
711 => "1011000111", 712 => "1011001000", 713 => "1011001001",
714 => "1011001010", 715 => "1011001011", 716 => "1011001100",
717 => "1011001101", 718 => "1011001110", 719 => "1011001111",
720 => "1011010000", 721 => "1011010001", 722 => "1011010010",
723 => "1011010011", 724 => "1011010100", 725 => "1011010101",
726 => "1011010110", 727 => "1011010111", 728 => "1011011000",
729 => "1011011001", 730 => "1011011010", 731 => "1011011011",
732 => "1011011100", 733 => "1011011101", 734 => "1011011110",
735 => "1011011111", 736 => "1011100000", 737 => "1011100001",
738 => "1011100010", 739 => "1011100011", 740 => "1011100100",
741 => "1011100101", 742 => "1011100110", 743 => "1011100111",
744 => "1011101000", 745 => "1011101001", 746 => "1011101010",
747 => "1011101011", 748 => "1011101100", 749 => "1011101101",
750 => "1011101110", 751 => "1011101111", 752 => "1011110000",
753 => "1011110001", 754 => "1011110010", 755 => "1011110011",
756 => "1011110100", 757 => "1011110101", 758 => "1011110110",
759 => "1011110111", 760 => "1011111000", 761 => "1011111001",
762 => "1011111010", 763 => "1011111011", 764 => "1011111100",
765 => "1011111101", 766 => "1011111110", 767 => "1011111111",
768 => "1100000000", 769 => "1100000001", 770 => "1100000010",
771 => "1100000011", 772 => "1100000100", 773 => "1100000101",
774 => "1100000110", 775 => "1100000111", 776 => "1100001000",
777 => "1100001001", 778 => "1100001010", 779 => "1100001011",
780 => "1100001100", 781 => "1100001101", 782 => "1100001110",
783 => "1100001111", 784 => "1100010000", 785 => "1100010001",
786 => "1100010010", 787 => "1100010011", 788 => "1100010100",
789 => "1100010101", 790 => "1100010110", 791 => "1100010111",
792 => "1100011000", 793 => "1100011001", 794 => "1100011010",
795 => "1100011011", 796 => "1100011100", 797 => "1100011101",
798 => "1100011110", 799 => "1100011111", 800 => "1100100000",
801 => "1100100001", 802 => "1100100010", 803 => "1100100011",
804 => "1100100100", 805 => "1100100101", 806 => "1100100110",
807 => "1100100111", 808 => "1100101000", 809 => "1100101001",
810 => "1100101010", 811 => "1100101011", 812 => "1100101100",
813 => "1100101101", 814 => "1100101110", 815 => "1100101111",
816 => "1100110000", 817 => "1100110001", 818 => "1100110010",
819 => "1100110011", 820 => "1100110100", 821 => "1100110101",
822 => "1100110110", 823 => "1100110111", 824 => "1100111000",
825 => "1100111001", 826 => "1100111010", 827 => "1100111011",
828 => "1100111100", 829 => "1100111101", 830 => "1100111110",
831 => "1100111111", 832 => "1101000000", 833 => "1101000001",
834 => "1101000010", 835 => "1101000011", 836 => "1101000100",
837 => "1101000101", 838 => "1101000110", 839 => "1101000111",
840 => "1101001000", 841 => "1101001001", 842 => "1101001010",
843 => "1101001011", 844 => "1101001100", 845 => "1101001101",
846 => "1101001110", 847 => "1101001111", 848 => "1101010000",
849 => "1101010001", 850 => "1101010010", 851 => "1101010011",
```

```
        852 => "1101010100", 853 => "1101010101", 854 => "1101010110",
        855 => "1101010111", 856 => "1101011000", 857 => "1101011001",
        858 => "1101011010", 859 => "1101011011", 860 => "1101011100",
        861 => "1101011101", 862 => "1101011110", 863 => "1101011111",
        864 => "1101100000", 865 => "1101100001", 866 => "1101100010",
        867 => "1101100011", 868 => "1101100100", 869 => "1101100101",
        870 => "1101100110", 871 => "1101100111", 872 => "1101101000",
        873 => "1101101001", 874 => "1101101010", 875 => "1101101011",
        876 => "1101101100", 877 => "1101101101", 878 => "1101101110",
        879 => "1101101111", 880 => "1101110000", 881 => "1101110001",
        882 => "1101110010", 883 => "1101110011", 884 => "1101110100",
        885 => "1101110101", 886 => "1101110110", 887 => "1101110111",
        888 => "1101111000", 889 => "1101111001", 890 => "1101111010",
        891 => "1101111011", 892 => "1101111100", 893 => "1101111101",
        894 => "1101111110", 895 => "1101111111", 896 => "1110000000",
        897 => "1110000001", 898 => "1110000010", 899 => "1110000011",
        900 => "1110000100", 901 => "1110000101", 902 => "1110000110",
        903 => "1110000111", 904 => "1110001000", 905 => "1110001001",
        906 => "1110001010", 907 => "1110001011", 908 => "1110001100",
        909 => "1110001101", 910 => "1110001110", 911 => "1110001111",
        912 => "1110010000", 913 => "1110010001", 914 => "1110010010",
        915 => "1110010011", 916 => "1110010100", 917 => "1110010101",
        918 => "1110010110", 919 => "1110010111", 920 => "1110011000",
        921 => "1110011001", 922 => "1110011010", 923 => "1110011011",
        924 => "1110011100", 925 => "1110011101", 926 => "1110011110",
        927 => "1110011111", 928 => "1110100000", 929 => "1110100001",
        930 => "1110100010", 931 => "1110100011", 932 => "1110100100",
        933 => "1110100101", 934 => "1110100110", 935 => "1110100111",
        936 => "1110101000", 937 => "1110101001", 938 => "1110101010",
        939 => "1110101011", 940 => "1110101100", 941 => "1110101101",
        942 => "1110101110", 943 => "1110101111", 944 => "1110110000",
        945 => "1110110001", 946 => "1110110010", 947 => "1110110011",
        948 => "1110110100", 949 => "1110110101", 950 => "1110110110",
        951 => "1110110111", 952 => "1110111000", 953 => "1110111001",
        954 => "1110111010", 955 => "1110111011", 956 => "1110111100",
        957 => "1110111101", 958 => "1110111110", 959 => "1110111111",
        960 => "1111000000", 961 => "1111000001", 962 => "1111000010",
        963 => "1111000011", 964 => "1111000100", 965 => "1111000101",
        966 => "1111000110", 967 => "1111000111", 968 => "1111001000",
        969 => "1111001001", 970 => "1111001010", 971 => "1111001011",
        972 => "1111001100", 973 => "1111001101", 974 => "1111001110",
        975 => "1111001111", 976 => "1111010000", 977 => "1111010001",
        978 => "1111010010", 979 => "1111010011", 980 => "1111010100",
        981 => "1111010101", 982 => "1111010110", 983 => "1111010111",
        984 => "1111011000", 985 => "1111011001", 986 => "1111011010",
        987 => "1111011011", 988 => "1111011100", 989 => "1111011101",
        990 => "1111011110", 991 => "1111011111", 992 => "1111100000",
        993 => "1111100001", 994 => "1111100010", 995 => "1111100011",
        996 => "1111100100", 997 => "1111100101", 998 => "1111100110",
        999 => "1111100111");

begin

        process(Clock, Reset, Address)
  begin
```

```
                    if (Reset = '1') then
                              Data_out <= "0000000000";
      elsif (Clock'event and Clock = '1') then
                              if (Enable = '1') then
                                        Data_out <= Content(conv_integer(Address));
        else
          Data_out <= "0000000000";
        end if;
      end if;
          end process;

end Behavioral;
```