UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA E COMPUTAÇÃO

Processamento embarcado aplicado a um sistema de detecção de vazamentos

Álvaro Medeiros Avelino

Orientador: Prof. Dr. Andres Ortiz Salazar

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Computação da UFRN (área de concentração: Automação e Sistemas) como parte dos requisitos para obtenção do título de Mestre em Ciências.

Número de ordem PPgEEC: M255
Natal, RN, dezembro de 2009

Divisão de Serviços Técnicos

Catalogação da publicação na fonte. UFRN / Biblioteca Central Zila Mamede

Avelino, Álvaro Medeiros.

Processamento embarcado aplicado a um sistema de detecção de vazamentos / Álvaro Medeiros Avelino - Natal, RN, 2009

64 f.

Orientador: Andres Ortiz Salazar

Dissertação (Mestrado) - Universidade Federal do Rio Grande do Norte. Centro de Tecnologia. Programa de Pós-Graduação em Engenharia Elétrica e Computação.

1. Sistema de detecção de vazamentos - Dissertação. 2. DSP - Dissertação. 3. GPS - Dissertação. 4 - Sistemas de tempo real - Dissertação. I. Salazar, Andres Ortiz. II. Universidade Federal do Rio Grande do Norte. III. Título.

RN/UF/BCZM CDU 004.03(043.3)

Processamento embarcado aplicado a um sistema de detecção de vazamentos

Álvaro Medeiros Avelino

Dissertação de Mestrado aprovada em 23 de dezembro de 2009 pela banca examinadora composta pelos seguintes membros:

Prof. Dr. Andres Ortiz Salazar (Orientador) . . . . . DCA/UFRN

Prof. Dr. André Laurindo Maitelli (Examinador interno) . . . . . DCA/UFRN

Prof. Dr. José Bezerra de Menezes Filho (Examinador externo) . . . . . IFPB

Prof. Dr. José Álvaro de Paiva (Examinador externo) . . . . . IFRN

Dedico esta Dissertação aos meus pais, Alva e Avelino, cujo exemplo de honestidade e trabalho têm sido um norteador para a minha vida, e para meus familiares e amigos, que têm me dado apoio nos momentos mais difíceis e mostrado a simplicidade de se ter determinação e esperança.

Agradecimentos

A Deus pela saúde e força de vontade que sempre me manteve firme nesta caminhada.

Aos meus pais, avó, tias, primos e primas que não mediram esforços no apoio e colaboração.

Aos amigos Gustavo Queiroz, Leonardo Dantas, Leonardo Gomes, e Leonardo José pela inestimável amizade construída ao longo dos anos.

Ao meu orientador, professor Andres Ortiz Salazar, sou grato pela orientação, apoio, críticas e sugestões.

Aos professores do Departamento de Computação e Automação da UFRN pela contribuição acadêmica.

Aos colegas do LAUT e do LAMP, especialmente aos do projeto SONIC. Meu muito obrigado.

E a todos que contribuíram de alguma forma para a concretização deste trabalho.

Resumo

Os sistemas embarcados estão amplamente difundidos atualmente. Um exemplo é o *Digital Signal Processor* (DSP), que é um dispositivo com alto poder de processamento. A contribuição deste trabalho consiste na implementação em DSP da lógica de um sistema de detecção de vazamentos em tempo real. Dentre os vários métodos de detecção de vazamentos existentes atualmente este trabalho se desenvolve utilizando uma técnica baseada na análise da pressão no duto e que utiliza Transformada *Wavelet* e Redes Neurais. Nesse contexto o DSP, além de realizar o processamento digital do sinal de pressão, também comunica-se com um *Global Positioning System* (GPS), que auxilia na localização do vazamento e com um sistema supervisório, disponibilizando informações para este. Para garantir robustez e confiabilidade na comunicação entre DSP e sistema supervisório é utilizado o protocolo Modbus. Como trata-se de uma aplicação de tempo real, uma atenção especial é dada ao tempo de resposta de cada uma das tarefas realizadas pelo DSP. Os testes e simulações de vazamentos foram realizados utilizando a estrutura do Laboratório de Avaliação de Medição em Petróleo (LAMP), da Universidade Federal do Rio Grande do Norte (UFRN).

Palavras-chave: Sistemas de detecção de vazamentos, DSP, GPS, sistemas de tempo real.

Abstract

Embedded systems are widely spread nowadays. An example is the Digital Signal Processor (DSP), which is a high processing power device. This work's contribution consist of exposing DSP implementation of the system logic for detecting leaks in real time. Among the various methods of leak detection available today this work uses a technique based on the pipe pressure analysis and uses Wavelet Transform and Neural Networks. In this context, the DSP, in addition to do the pressure signal digital processing, also communicates to a Global Positioning System (GPS), which helps in situating the leak, and to a SCADA, sharing information. To ensure robustness and reliability in communication between DSP and SCADA the Modbus protocol is used. As it is a real time application, special attention is given to the response time of each of the tasks performed by the DSP. Tests and leak simulations were performed using the structure of Laboratory of Evaluation of Measurement in Oil (LAMP), at Federal University of Rio Grande do Norte (UFRN).

Keywords: Leak detection systems, DSP, GPS, real time systems.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Símbolos e Abreviaturas

L:	Latência
S:	Sensibilidade
V_{ain}:	Tensão Analógica de Entrada
V_{max}:	Limite Superior de Tensão
V_{min}:	Limite Inferior de Tensão
Δt_p:	Intervalo de tempo entre o fim do processamento atual e o início do próximo
t_t:	Tempo total de processamento
t_{id}:	tempo com interrupções desabilitadas
t_{iisr}:	tempo para iniciar a execução da primeira instrução da ISR
t_{rc}:	tempo para restaurar o contexto
t_{risr}:	tempo para retornar da ISR
t_{rsi}:	tempo de resposta da interrupção
t_{rti}:	tempo de retorno da interrupção
t_{sc}:	tempo para salvar o contexto
ADC:	Analog-to-Digital Converter
ASIC:	Application-Specific Integrated Circuit
bps:	bits per second
BRR:	Baud Rate Register
CA:	Coeficiente de análise

CAN:	Controller Area Network
CCS:	Code Composer Studio
CD:	Coeficiente de detalhamento
CI:	Circuito Integrado
CPU:	Central Processing Unit
CR:	Carriage Return
DDE:	Dynamic Data Exchange
DMA:	Direct Memory Access
DOP:	Dilution Of Position
DSP:	Digital Signal Processor
EEPROM:	Electrically-Erasable Programmable Read-Only Memory
EIA:	Electronic Industries Alliance
ETA:	Estreito A
ETB:	Estreito B
FE:	Fundo de Escala
FIFO:	First-In First-Out
FPGA:	Field-Programmable Gate Array
FPU:	Floating-Point Unit
GGA:	Global Positioning System Fixed Data
GNSS:	Global Navigation Satellite System
GPIO:	General-Purpose Input Output
GPS:	Global Positioning System
GSA:	GNSS DOP and Active Satellites

GSV:	GNSS Satellites in View
I2C:	Inter-Integrated Circuit
IDE:	Integrated Development Environment
IEEE:	Institute of Electrical and Eletronics Engineers
ISR:	Interrupt Service Routine
LAMP:	Laboratório de Avaliação de Medição em Petróleo
LF:	Line Feed
LSPCLK:	Low Speed Peripheral Clock
McBSP:	Multichannel Buffered Serial Port
MIT:	Massachussets Institute of Techonology
NEMA:	National Engineering Manufacturers Association
NMEA:	National Marine Electronic Association
OTP:	One Time Programmable
PIE:	Peripheral Interrupt Expansion
PWM:	Pulse-Width Modulation
RAM:	Random Access Memory
RMC:	Recommended Minimum Specific GNSS Data
ROM:	Read-Only Memory
RTU:	Remote Terminal Unit
SAR:	Successive Approximation Register
SCI:	Serial Communications Interface
SCIRXINTA:	SCI RX Interrupt A
SPI:	Serial Peripheral Interface

TINT**0**: Timer **0** Interrupt

TTL: Transistor-Transistor Logic

UART: Universal Asynchronous Receiver Transmitter

UFRN: Universidade Federal do Rio Grande do Norte

UHF: Ultra High Frequency

USB: Universal Serial Bus

UTC: Universal Time Coordinate

VD: Valor Digital

XINT2: External Interrupt 2

Capítulo 1

Introdução

Em 2006 a malha dutoviária mundial era de aproximadamente 2 milhões de km de dutos, dos quais 65% estavam localizados na América do Norte, Europa Oriental e Rússia (Central Intelligence Agency, 2006). A Figura 1 mostra a distribuição da malha dutoviária mundial.

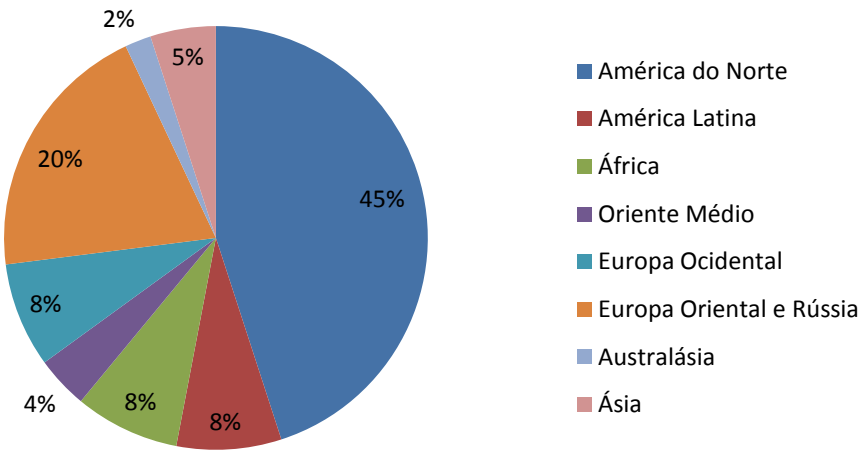

Figura 1: Distribuição da malha dutoviária mundial

No ano de 2007, a infraestrutura dutoviária nacional era composta de 526 dutos destinados à movimentação de petróleo, derivados, gás natural e outros produtos. Esses dutos somaram aproximadamente 16.000km de extensão, divididos em 10.700km para transporte e 5.200km para transferência. Com extensão de 8.100km, 87 dutos destinavam-se à movimentação de gás natural; 372 dutos, com extensão de 5.800km, à movimentação de derivados; 30 dutos, com extensão de 2.000km, à movimentação de petróleo; e os 76km restantes, compostos por 37 dutos, destinaram-se à movimentação dos demais produtos, tais como álcool, solventes e outros de menor importância (Agência Nacional do Petróleo, Gás Natural e Biocombustíveis, 2009).

Algumas razões para a expansão da malha dutoviária brasileira são (RENNÓ, 2007):

- Aumento da demanda de gás natural
- Crescimento da produção nacional de gás natural e óleo cru
- Aumento da demanda de derivados de óleo
- Exportação do etanol brasileiro
- Necessidade de melhorias de segurança

Um dos principais problemas pertinentes ao transporte de petróleo e derivados é a ocorrência de vazamentos, que implicam não apenas em perdas econômicas mas também em danos ao meio ambiente. Ainda em relação a este assunto, uma área de extrema importância de conservação é a Floresta Amazônica. Adicione os dois temas acima referidos à existência de campos de produção de petróleo na floresta mencionada, e é fácil entender a importância de evitar acidentes com derramamento de óleo (MOURA et al., 2002). Em países asiáticos a ocorrência de vazamentos está ligada também ao furto de petróleo.

Além do mais, oleodutos de produção nem sempre apresentam comportamento previsível, já que boa parte deles transportam fluido com escoamento multifásico. O escoamento multifásico dificulta a detecção de vazamentos por várias razões:

- Apresentam duas ou mais fases independentes
- O volume de cada fase varia ao longo do duto
- Cada fase apresenta uma velocidade de escoamento

Um dos fatores que determina os níveis de robustez e confiabilidade de um sistema de detecção de vazamentos é a quantidade de alarmes falsos emitidos pelo mesmo. Um desligamento de bomba, por exemplo, pode ser confundido com um vazamento, levando o sistema a emitir um alarme falso.

1.1 Histórico da pesquisa na UFRN

A pesquisa sobre detecção de vazamentos na UFRN foi iniciada em 2003 e atualmente encontra-se na segunda fase. Abaixo são descritas as duas etapas deste processo.

- 1ª etapa: 2003-2006
  - Sensor de pressão capacitivo, com tempo de resposta de 20ms
  - Controlador Lógico Programável (CLP) utilizado para adquirir o sinal do sensor

  - Comunicação sem fio utilizando *Ultra High Frequency* (UHF) entre o CLP e o sistema supervisório
  - Processamento dos dados feito no supervisório

- 2ª etapa: 2007-atual
  - Sensor de pressão piezoresistivo, com tempo de resposta de 1ms
  - Processamento local realizado por um DSP
  - Um GPS auxilia na localização do vazamento
  - Pesquisadores:
    * Eng. Álvaro Avelino - DSP
    * Graduando Gabriell John - Algoritmos
    * Prof. José Álvaro de Paiva - Algoritmos
    * Eng. Rodrigo Eduardo - Supervisório

1.2 Motivação

Baseado na dimensão atual e na projeção de expansão futura da malha dutoviária brasileira, pode-se inferir que é necessário investir em sistemas de detecção de vazamentos, tendo em vista que trata-se de um meio de minimizar possíveis danos causados ao meio ambiente.

Outra fonte de incentivo à pesquisa nesta área é a falta de credibilidade de alguns sistemas de detecção de vazamentos instalados na indústria atualmente, devido à grande quantidade de alarmes falsos emitidos por esses sistemas (MOURA et al., 2002).

1.3 Objetivo

Este trabalho tem como objetivo implementar em DSP um sistema de tempo real aplicado a um sistema de detecção de vazamentos que emprega tecnologia sônica. Os objetivos específicos são:

- Estabelecer a comunicação entre DSP e GPS, a fim de que o primeiro obtenha o horário fornecido pelo último
- Testar e validar em DSP um conjunto de algoritmos capaz de detectar vazamentos a partir da análise da pressão em um oleoduto

- Estabelecer a comunicação entre DSP e o sistema supervisório, com a finalidade de disponibilizar ao operador informações de vazamentos e a sua respectiva localização no intervalo do duto monitorado

1.4 Organização do texto

O Capítulo 2 aborda os sistemas embarcados. Discorre sucintamente sobre o histórico dos sistemas embarcados e sobre a utilização de DSP. Também trata de alguns aspectos de sistemas de tempo real. O Capítulo 3 explica o objetivo do projeto SONIC. Em seguida descreve os componentes utilizados e, por fim, mostra a arquitetura do projeto em campo. O Capítulo 4 aborda as características do sistema proposto neste trabalho e mostra os resultados obtidos a partir de testes em laboratório. Por último, o Capítulo 5 apresenta as considerações finais e algumas sugestões para trabalhos futuros.

Capítulo 2

Sistemas embarcados

Um sistema embarcado é uma combinação de *hardware* e *software*, projetado para realizar uma função específica (BARR, 1999). Diferente de computadores de propósito geral, como o computador pessoal, um sistema embarcado realiza um conjunto de tarefas predefinidas, geralmente com requisitos específicos.

Normalmente os sistemas embarcados são otimizados de modo a apresentarem dimensões e confiabilidade para o fim a que é empregado.

2.1 Histórico

Na década de 1940 os computadores eram dedicados a uma única tarefa. Porém, eram muito grandes para serem considerados embarcados e caros para desempenhar a maioria das tarefas executadas por sistemas embarcados atualmente.

O primeiro sistema embarcado reconhecido foi o *Apollo Guidance Computer* (ver Figura 2), desenvolvido por Charles Stark Draper no *Massachussets Institute of Technology* (MIT). Foi considerado um projeto ousado e arriscado, uma vez que utilizava o recém-desenvolvido circuito integrado para reduzir o tamanho e o peso do produto final (Wikipedia - Embedded Systems, 2009).

O primeiro sistema embarcado de produção em massa foi o computador guia (chamado Autonetics D-17) do míssil nuclear LGM-30 Míssil Minuteman, lançado em 1961. Ele utilizava transistores e possuía um disco rígido como memória principal. Quando a segunda versão do míssil entrou em produção no ano de 1966, o computador guia foi substituído por um novo, que constituiu o primeiro uso em grande volume de circuitos integrados.

À medida que o custo de microprocessadores e microcontroladores vem caindo, torna-se viável a substituição de componentes analógicos (tais como potenciômetros e capacitores variáveis) por um processador embarcado. A partir da década de 1980 vários compo-

nentes externos foram integrados em um mesmo chip, resultando em uma difusão ainda maior dos sistemas embarcados.

Em 1978 foi lançada pela *National Engineering Manufacturers Association* (NEMA) a norma para microcontroladores programáveis.

Figura 2: Primeiro sistema embarcado

2.2 Características

Um sistema embarcado genérico é composto basicamente por quatro componentes: entrada(s), memória, processador e saída(s).

As entradas e saídas podem ser analógicas e/ou digitais.

A memória pode ser interna ou externa. No caso das externas existem modelos que comunicam-se com o processador de forma paralela e outros modelos que comunicam-se de forma serial. A última geralmente comunica-se com o processador utilizando o protocolo *Inter-Integrated Circuit* (I2C) ou o protocolo *Serial Peripheral Interface* (SPI). Já as internas possuem um tempo de acesso mais curto que as externas, porém apresentam um custo mais elevado. Um exemplo é a memória *cache*.

Quanto à forma de armazenamento da informação, as memórias mais comuns são:

- *Random Access Memory* (RAM): é volátil e permite tanto leitura quanto escrita
- *Read-Only Memory* (ROM): não-volátil e permite apenas leitura
- *Electrically-Erasable Programmable Read-Only Memory* (EEPROM): não-volátil, permite tanto leitura quanto escrita

O processador pode ter:

- Um barramento para dados e outro para instruções (arquitetura *Harvard*)
- Um barramento para dados e instruções (arquitetura *Von Neumann*)
- Suporte a processamento de dados do tipo ponto-flutuante
- Memória *cache*
- *Central Processing Unit* (CPU) capaz de processar dados de 4, 8, 16, 32 ou 64 bits
- Um ou mais núcleos de processamento

A Tabela 1 cita alguns requisitos que devem ser levados em conta no projeto de um sistema embarcado. De acordo com a mesma Tabela, pode-se inferir que um sistema embarcado de baixo custo trabalha com um processador de 4 ou 8 bits, utiliza pouca memória e possui uma vida útil relativamente curta, enquanto um sistema de alto desempenho (e consequentemente um custo mais elevado) utiliza processadores de 32 bits, que por sua vez exige uma quantidade maior de memória e possui vida útil relativamente longa.

Tabela 1: Principais requisitos de projeto de sistemas embarcados

Critério	**Baixo**	**Médio**	**Alto**
Processador	4 ou 8 bits	16 bits	32 bits
Memória	< 16KB	64KB a 1MB	> 1MB
Custo de desenvolvimento ($)	< 100.000	100.000 a 1.000.000	> 1.000.000
Custo de produção ($)	< 10	10 a 1.000	> 1.000
Unidades	< 100	100 a 10.000	> 10.000
Vida útil	dias, semanas ou meses	anos	décadas
Confiabilidade	ocasionalmente pode falhar	deve funcionar confiavelmente	deve ser à prova de falhas

2.3 DSP

Processadores de uso geral, popularmente conhecidos como microprocessadores, são projetados para ter ampla funcionalidade. Estes processadores são projetados para serem usados em uma extensa variedade de aplicações. A meta de desempenho de processadores de uso geral é apresentar um desempenho máximo em torno de uma longa gama de aplicações.

Processadores especializados (também chamados de dedicados), por outro lado, são projetados para tirar proveito de uma funcionalidade limitada exigida pela aplicação para

alcançar objetivos específicos. DSP é um tipo de processador especializado. Este processador é projetado com foco em aplicações de processamento de sinais. Consequentemente, há pouco ou nenhum suporte a características tais como gerenciamento de memória virtual, proteção de memória e certos tipos de exceções. A Figura 3 destaca algumas aplicações que utilizam DSP.

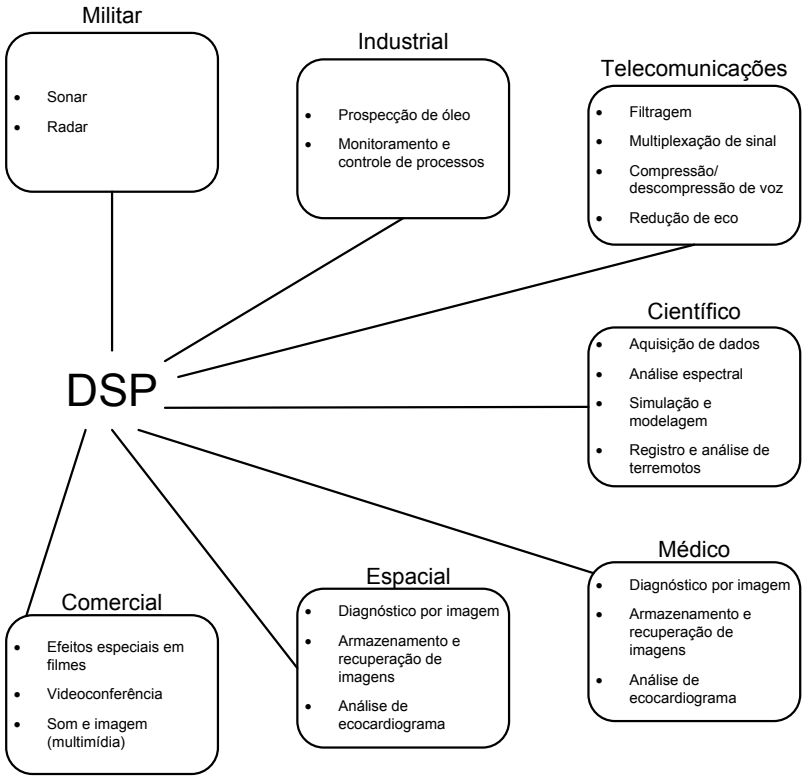

Figura 3: Aplicações de DSP

Os fabricantes de DSP mais conhecidos são Analog Devices®, Freescale® e Texas Instruments®. A Tabela 2 mostra as principais famílias de DSP (KUO, 2006) dos fabricantes anteriormente citados. A Texas Instruments®, por exemplo, tem três famílias de DSP (Texas Instruments, 2009*a*):

1. **C2000**: voltada para aplicações de controle. Divide-se em duas subfamílias - C24x e C28x. A primeira é de 16 bits, ponto-fixo, com frequência de até 40MHz. A última é de 32 bits, pode ser ponto-fixo ou ponto-flutuante e possui frequência de até 300 MHz.
2. **C5000**: família de baixo consumo de energia voltada para aplicações de vídeo e telecomunicações. Todos os modelos são de 16 bits e ponto-fixo. Divide-se em duas subfamílias - C54x e C55x. A primeira apresenta frequências entre 50 e 160MHz. A

última é composta por modelos de ultra baixo consumo de energia, com frequências entre 108 e 300MHz.

3. **C6000**: composta por processadores de alto desempenho, de 32 bits. Divide-se em três subfamílias - alta performance (C645x e C647x), *performance value*[1] (C62x e C642x) e ponto-flutuante (C67x, C672x e C674x). A primeira possui modelos com frequências de até 3 GHz, sendo composto por três núcleos de 1GHz cada. A segunda apresenta alto desempenho aliado a um baixo consumo de energia, com frequências entre 150 e 720MHz. A última possui dispositivos ponto-flutuante, com frequências entre 150 e 350MHz, e alguns modelos podendo processar até oito instruções por ciclo de *clock*.

A Tabela 3 mostra as três famílias de DSP da Texas Instruments®, divididas por áreas de aplicação.

Tabela 2: DSPs mais comuns atualmente

Fabricante	Família	Tipo de dado	Velocidade máxima
Texas Instruments®	TMS320C24x	Ponto-fixo	40MHz
	TMS320C28x	Ponto-fixo/flutuante	300MHz
	TMS320C54x	Ponto-fixo	160MHz
	TMS320C55x	Ponto-fixo	600MHz
	TMS320C62x	Ponto-fixo	300MHz
	TMS320C64x	Ponto-fixo	1,2GHz
	TMS320C67x	Ponto-flutuante	300MHz
Analog Devices®	ADSP218x	Ponto fixo	80MHz
	ADSP21xx	Ponto-fixo	160MHz
	ADSP213xx	Ponto-flutuante	450MHz
	ADSP-BF5xx	Ponto-fixo	750MHz
	ADSP-TS20x	Ponto-fixo/flutuante	600MHz
Freescale®	DSP56300	Ponto fixo	275MHz
	DSP568xx	Ponto-fixo	40MHz
	DSP5685x	Ponto-fixo	120MHz
	MSC71xx	Ponto-fixo	200MHz
	MSC81xx	Ponto-fixo	400MHz

2.3.1 Razões para utilizar um DSP

DSP é um tipo particular de microprocessador, já que o primeiro contém algumas características e componentes comuns ao segundo, tais como CPU, memória, conjunto de instruções e barramento. Entretanto, cada um destes componentes (no caso do DSP)

[1]Termo criado pelo fabricante. Não foi encontrada tradução adequada à língua portuguesa.

Tabela 3: Aplicações das famílias de DSP da Texas Instruments

	C6000	**C5000**	**C2000**
Áudio	x	x	
Automotiva			x
Comunicação	x	x	x
Industrial			x
Médica	x	x	x
Vídeo	x	x	
Wireless	x	x	

é projetado para desempenhar determinadas funções de forma mais eficiente. Um DSP tem *hardware* e conjunto de instruções otimizados para processamento numérico de alta velocidade e alto poder de processamento em tempo real de sinais analógicos (OSHANA, 2006).

2.3.2 Vantagens do processamento usando DSP

Há várias vantagens em realizar o processamento utilizando um DSP. Vejamos algumas:

1. **Mutabilidade**: É fácil reprogramar sistemas digitais para outras aplicações ou realizar uma sintonia em sistemas já existentes. Um DSP permite realizar mudanças e atualizações de forma rápida e prática em um sistema.
2. **Repetibilidade**: Componentes analógicos têm características que podem variar com o tempo ou até mesmo com a variação de temperatura. Uma solução digital programável é muito mais repetível devido à natureza programável do sistema. Sistemas com múltiplos DSPs, por exemplo, podem também executar exatamente o mesmo programa e serem bastante repetíveis. Com o processamento analógico do sinal cada DSP no sistema teria que ser individualmente sintonizado.
3. **Tamanho**, **peso** e **potência**: Uma solução com DSP implica em uma potência dissipada menor do que se fossem utilizados apenas componentes de *hardware*.
4. **Confiança**: Sistemas analógicos são confiáveis apenas à medida em que os dispositivos de *hardware* funcionem apropriadamente. Se algum destes dispositivos falhar devido às condições físicas, o sistema inteiro irá se degradar ou falhar. Uma solução que utilize *software* implementado em DSP funcionará adequadamente desde que o *software* seja implementado corretamente.
5. **Expansibilidade**: Para adicionar mais funcionalidade ao sistema, o engenheiro precisa adicionar mais *hardware*. Isso nem sempre é possível. Para adicionar a mesma

funcionalidade a um DSP basta adicionar *software*, que é bem mais simples.

DSPs são úteis em sistemas embarcados principalmente por uma razão: processamento do sinal. A habilidade de realizar funções de processamento de sinal complexas em tempo real dá vantagem ao DSP em relação a outras formas de processamento embarcado. DSPs têm que dar uma resposta em tempo real a sinais analógicos de entrada, convertê-los em sinais digitais, executar processamento adicional sobre esses sinais (se necessário), converter o sinal processado novamente a analógico, para então ser enviado à saída.

Cinco plataformas são amplamente utilizadas em sistemas que exigem processamento digital (KUO, 2006):

1. Chips com propósito especial, tais como *Application-Specific Integrated Circuits* (ASICs);
2. *Field-Programmable Gate Arrays* (FPGAs);
3. Microprocessadores ou microcontroladores de uso geral (μP/μC);
4. Processadores digitais de sinais de uso geral;
5. DSPs com aceleradores de *hardware* para aplicações específicas.

A Tabela 4 mostra os critérios que devem ser levados em conta na escolha de uma plataforma digital.

Tabela 4: Resumo de implementações de DSP em *hardware*

	ASIC	FPGA	μP/μC	DSP	DSP com acelerador de *hardware*
Flexibilidade	Nenhuma	Limitada	Alta	Alta	Média
Tempo de desenvolvimento	Longo	Médio	Curto	Curto	Curto
Consumo (potência)	Baixo	De baixo a médio	De médio a alto	De baixo a médio	De baixo a médio
Performance	Alta	Alta	De baixa a média	De média a alta	Alta
Custo de desenvolvimento	Alto	Médio	Baixo	Baixo	Baixo
Custo de produção	Baixo	De baixo a médio	De médio a alto	De baixo a médio	Médio

Comparado a soluções utilizando ASIC e FPGA, os DSPs apresentam como vantagens a facilidade no desenvolvimento do projeto e também a reprogramação, permitindo

uma futura atualização ou correção de erros. Geralmente os DSPs têm um melhor custo benefício em relação a *hardware* personalizado, como ASIC e FPGA, especialmente para produção de poucas unidades. Em comparação a microcontroladores e microprocessadores, os DSPs apresentam maior velocidade, melhor gerenciamento de energia e custo mais baixo.

2.4 Sistemas embarcados de tempo real

Um sistema de tempo real é um sistema que reage a um estímulo do ambiente (incluindo a passagem do tempo) dentro de intervalos de tempo bem definidos. Existem dois tipos de sistemas de tempo real: *soft* e *hard*. No *soft*, as tarefas são realizadas o mais rápido possível, porém, as tarefas não têm que ser concluídas em um intervalo de tempo específico. Já no *hard*, as tarefas têm não apenas que serem executadas corretamente mas também a tempo. A maioria dos sistemas consiste em uma combinação de requisitos *soft* e *hard*. As aplicações dos sistemas de tempo real são bastante vastas, mas a maioria dos sistemas de tempo real é embarcado (LABROSSE, 2002).

Uma aplicação de tempo real é tipicamente mais complexa de se projetar, se comparada a uma aplicação que não seja de tempo real.

2.4.1 Tipos de eventos em sistemas de tempo real

Eventos de tempo real dividem-se em três categorias: assíncronos, síncronos ou isócronos.

Eventos assíncronos são totalmente imprevisíveis. Um exemplo é uma ligação recebida em um telefone celular.

Eventos síncronos são previsíveis e ocorrem com regularidade. Por exemplo, o áudio e o vídeo em uma câmera.

Eventos isócronos ocorrem com regularidade dentro de uma janela de tempo. Por exemplo, áudio em uma aplicação multimídia via rede deve aparecer dentro de uma janela de tempo correspondente à chegada do fluxo de vídeo.

2.4.2 Abordagens de *software* em tempo real

Existem três abordagens de *software* em tempo real: *foreground/background*, *kernel* preemptivo e *kernel* não-preemptivo.

Na abordagem *foreground/background* uma aplicação consiste em um *looping* infinito que chama módulos (ou funções) para realizar as devidas operações (*background*). *Interrupt Service Routines* (ISRs) gerenciam eventos assíncronos (*foreground*). O *foreground* também é chamado nível de interrupção; o *background* também é conhecido como nível de tarefas.

Operações críticas devem ser executadas por ISRs para garantir que serão atendidas a tempo. A informação disponibilizada por uma ISR para uma rotina em *background* só é processada quando a rotina em *background* é executada. Isso é chamado resposta do nível de tarefas. O pior caso da resposta do nível de tarefa depende de quanto tempo é necessário para que a rotina em *background* seja executada. A Figura 4 exemplifica a execução de um trecho de código em um sistema *foreground/background*.

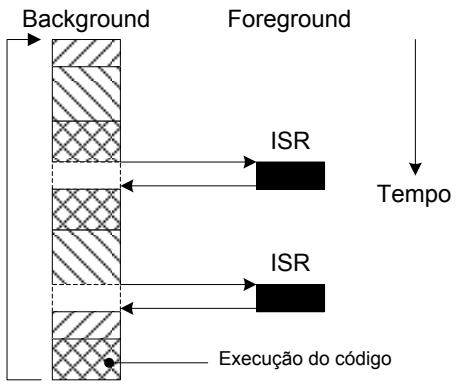

Figura 4: Exemplo de execução em um sistema *foreground/background*

Muitas aplicações embarcadas (forno de micro-ondas, telefones e brinquedos, por exemplo) utilizam sistemas *foreground/background* (LABROSSE, 2002).

Os *kernels* não-preemptivos permitem que cada tarefa seja executada até que esta voluntariamente ceda o controle da CPU. Uma interrupção suspende a execução da tarefa atual. Uma vez concluída a ISR, a CPU volta a executar a tarefa suspensa.

Já os *kernels* preemptivos sempre executam a tarefa de mais alta prioridade que está pronta para execução. A ocorrência de uma interrupção suspende a tarefa que está sendo executada. Uma vez finalizada a ISR, o *kernel* retoma a execução da tarefa de mais alta prioridade que está pronta para ser executada (não obrigatoriamente a tarefa que foi suspensa pela ISR).

2.4.3 Interrupção

Interrupção é um mecanismo utilizado para informar à CPU que um evento ocorreu. A interrupção pode ser de *hardware* ou de *software* e o evento pode ser assíncrono ou

síncrono. Pode ainda ser mascarável ou não mascarável. A primeira é um tipo de interrupção cujo evento de disparo nem sempre é importante, então o programador pode decidir habilitá-la (por *software*) ou não. Já a última (como o botão de *reset*, por exemplo) é tão importante que nunca deve ser ignorada (não pode ser desabilitada por *software*) (VALENZUELA, 2005).

Interrupções permitem que a CPU processe eventos no instante em que ocorrem, o que previne a CPU continuamente de realizar *polling* para observar se um determinado evento mesmo ocorreu. Microcontroladores e DSPs permitem que as interrupções mascaráveis sejam habilitadas ou desabilitadas por *software*.

Alguns *kernels* preemptivos permitem interrupções aninhadas, o que significa que enquanto uma interrupção está sendo processada, a CPU pode reconhecer e executar outra interrupção (desde que a última seja de maior prioridade).

As ISRs devem conter o mínimo de código possível. ISRs longas implicam em manter interrupções desabilitadas (no caso das abordagens *foreground/background* ou *kernel* não-preemptivo) por mais tempo, impedindo a CPU de atender a outras possíveis ISRs (GANSSLE, 2004).

2.4.4 Latência de uma interrupção

Todos os sistemas de tempo real desabilitam interrupções para manipular seções críticas de código e reabilitam as interrupções uma vez que o código crítico tenha sido executado. Quanto mais tempo as interrupções permanecerem desabilitadas, maior será a latência. A latência em um sistema *foreground/background* é dada pela seguinte Equação:

$$L = max(t_{id}) + t_{iisr} \tag{2.1}$$

onde L é a latência, t_{id} é o tempo com interrupções desabilitadas e t_{iisr} é o tempo para iniciar a execução da primeira instrução da ISR.

2.4.5 Tempo de resposta de uma interrupção

O tempo de resposta de uma interrupção é definido como sendo o tempo entre a recepção da interrupção e o início do código do usuário que manipula a interrupção. Este tempo leva em conta toda a sobrecarga envolvida na manipulação da interrupção. O tempo de resposta de uma interrupção em um sistema *foreground/background* é dado por:

$$t_{rsi} = L + t_{sc} \tag{2.2}$$

onde t_{rsi} é o tempo de resposta da interrupção, L é a latência da interrupção e t_{sc} é o tempo para salvar o contexto.

2.4.6 Retorno da interrupção

O retorno de uma interrupção é definido como o tempo exigido pela CPU para retornar do código da interrupção ou para executar uma tarefa de mais alta prioridade (caso o sistema suporte esse tipo de situação). O retorno da interrupção em um sistema *foreground/background* é dado por:

$$t_{rti} = t_{rc} + t_{risr} \tag{2.3}$$

onde t_{rti} é o tempo de retorno da interrupção, t_{rc} é o tempo para restaurar o contexto e t_{risr} é o tempo para retornar da ISR.

A Figura 5 mostra a latência, o retorno e o tempo de resposta de uma interrupção em um sistema *foreground/background*.

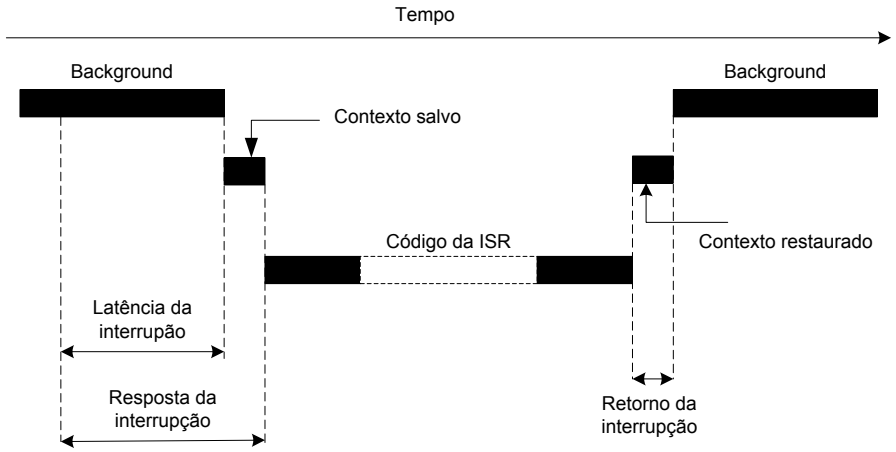

Figura 5: Latência, resposta e retorno de uma interrupção em um sistema *foreground/background*

2.4.7 Seções críticas de código

Uma seção crítica de código (ou simplesmente região crítica) é um trecho de código que precisa ser executado indissociavelmente. Uma vez iniciada a execução, a mesma não pode ser interrompida. Para garantir essa condição as interrupções devem ser desabilitadas antes do início do código e reabilitadas uma vez finalizada a execução do código.

Em um ambiente de tempo real interrupções devem ser desabilitadas o mínimo possível. Desabilitar interrupções afeta a latência das interrupções e pode causar a perda delas.

2.4.8 Troca de contexto

Quando a CPU recebe um pedido de interrupção, caso ela esteja realizando algum processamento é necessário salvar alguns registradores antes de atender à interrupção. Caso contrário poderá haver perda de informações. Essa operação é chamada salvar contexto. Da mesma forma, após ter atendido uma interrupção, a CPU retorna ao processamento anterior. Esta última operação é chamada restaurar contexto. Ambas as operações são comuns a sistemas *foreground/background* e sistemas baseados em *kernel* (ZHOU; PETROV, 2006).

2.5 Conclusão

De acordo com o conteúdo exposto neste Capítulo, foi possível constatar a grande potencialidade e aplicabilidade dos sistemas embarcados em diversas áreas de conhecimento.

O Capítulo apresentou as principais características dos sistemas embarcados e os diversos tipos de dispositivos que são capazes de realizar processamento digital de sinais. Foram também mostradas algumas vantagens que o DSP apresenta em relação aos demais dispositivos. Exemplos de famílias de DSPs da Texas Instruments® foram mostrados, bem como a área de aplicação de cada uma das famílias.

Por último foram abordados os sistemas embarcados de tempo real, em que os principais conceitos relativos à abordagem *foreground/background* foram expostos, com atenção especial às interrupções.

Capítulo 3

Projeto SONIC

O projeto SONIC tem por objetivo detectar vazamentos através do monitoramento da pressão do duto. O sistema irá monitorar a região entre as estações Estreito A (ETA) e Estreito B (ETB), da PETROBRAS®, localizadas em Alto do Rodrigues, RN. A região monitorada compreende 9,8km de extensão. Para isso, serão instalados quatro sensores de pressão, sendo dois em cada estação. Esta técnica é conhecida como filtragem direcional e permite investigar a origem do vazamento: caso este seja detectado pelos sensores mais externos, isso indica que o vazamento ocorreu fora da região de monitoramento; caso contrário, o referido vazamento ocorreu entre ETA e ETB.

Em cada estação há um DSP coletando e processando os dados fornecidos pelos dois sensores de pressão. Ao final do processamento, caso seja detectado vazamento, o DSP envia, juntamente com o horário fornecido pelo GPS, um quadro ao supervisório indicando vazamento. A comunicação entre o DSP e o supervisório utiliza o protocolo Modbus RTU e o paradigma mestre/escravo, sendo o primeiro, o escravo e o último, o mestre. A interface física que o DSP utiliza para comunicar-se com o supervisório é a EIA232. Entretanto, este padrão tem uma limitação física de 30m entre os dispositivos (PARK; MACKAY, 1997). Na estação ETB a distância entre o painel do projeto SONIC e a sala de controle é de 220m. Então a saída do DSP é convertida ao padrão EIA485, que suporta distâncias de até 1.200m (PARK; MACKAY, 1997). No abrigo de automação o sinal do padrão RS485 é convertido em Ethernet. Desta forma os dados podem ser acessados pela rede PETROBRAS®.

3.1 Sensor

O transmissor de pressão utilizado é o PTX7800, da Druck®. Este transmissor é manométrico e o elemento sensor é piezoresistivo (captura a estática do sinal). Este tipo de instrumento tem o funcionamento de seu transdutor baseado na variação de comprimento

e diâmetro, e, portanto, na variação da resistência, que ocorre quando um fio de resistência sofre uma deformação elástica proveniente de uma tensão mecânica gerada por uma pressão (BEGA et al., 2006). A Tabela 5 mostra as principais características do sensor.

Tabela 5: Características do sensor

Alimentação	9-30Vdc
Exatidão	0,25% FE, combinado com não-linearidade, histerese e repetibilidade
Sobrepressão	3 × FE
Temperatura de operação (ambiente e processo)	-40 a 100^{o}C
Tempo de resposta	1ms
Carga máxima	50 × (tensão aplicada - 9)
Saída	4-20mA

A isolação de *Hastelloy* do diafragma e a solda ao corpo de aço inoxidável 316L e o encapsulamento eletrônico fornecem uma montagem livre de manutenção, adequada para utilização em uma vasta gama de aplicações, como instalações industriais (Druck, 2003). A Figura 6 mostra o sensor de pressão utilizado.

Figura 6: Sensor de pressão PTX7800

3.2 DSP

O DSP escolhido foi o TMS320F28335, da Texas Instruments®. Duas características foram decisivas para a escolha deste modelo de DSP: o conversor analógico-digital interno e o suporte a processamento em ponto-flutuante. Além destas duas características, alguns grupos de pesquisa da UFRN já utilizavam um modelo parecido (e da mesma família), o TMS320F2812. A Tabela 6 mostra as principais características do DSP utilizado no projeto SONIC.

Tabela 6: Características do DSP

Frequência	150MHz
Tipo de instrução	Ponto-flutuante e ponto-fixo
Arquitetura	Harvard
Memória RAM	68KB
Memória FLASH	512KB
Timers	3 de 32 bits
Interfaces seriais	2 CAN, 1 I2C, 2 McBSP, 3 SCI, 1 SPI

3.2.1 Arquitetura do DSP

Conforme citado na Tabela 6 o DSP suporta instruções em ponto-flutuante com precisão simples (Texas Instruments, 2008*b*), de acordo com o padrão IEEE-754. Esta é uma característica bastante atraente, tendo em vista que a aplicação exige bastante cálculo. O bloco responsável pelo processamento das instruções em ponto-flutuante é o *Floating-Point Unit* (FPU). O DSP possui ainda seis canais de *Direct Memory Access* (DMA). A Figura 7 mostra a representação de um número de 32 bits em ponto-flutuante utilizada pelo DSP.

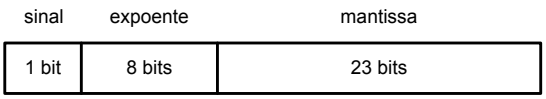

Figura 7: Padrão IEEE-754

A Figura 8 (fonte: Texas Instruments) mostra o diagrama de blocos do DSP. A Figura destaca em amarelo os blocos de memória RAM (à esquerda) e Flash (à direita). Se necessário, blocos contíguos de memória RAM podem ser mesclados.

O DSP é *in socket*, o que significa que se houver algum problema com o processador, é possível realizar a troca apenas do processador, não comprometendo o restante do kit de desenvolvimento. Outra vantagem deste DSP é o *Analog-to-Digital Converter* (ADC) interno (Texas Instruments, 2007*c*), dispensando o uso de um Circuito Integrado (CI) externo específico para esta tarefa. O kit conta ainda com uma interface RS232, através de um conector DB9, que neste trabalho será utilizada para comunicação do DSP com o supervisório.

3.2.2 Conversor analógico-digital

O módulo ADC do DSP possui 16 canais, divididos em dois módulos independentes de oito canais cada. Opcionalmente, estes módulos podem ser cascateados para formar

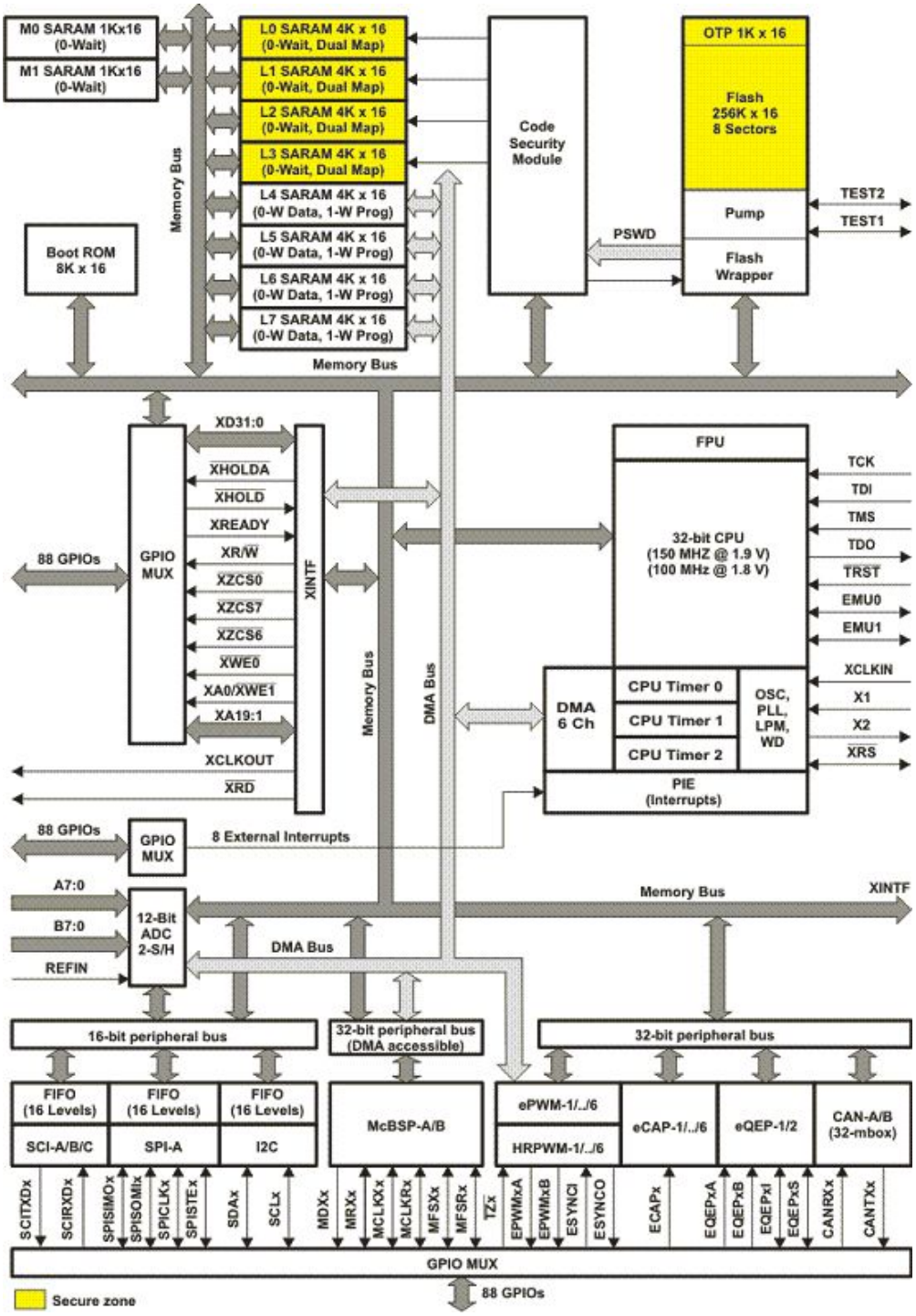

Figura 8: Diagrama de blocos do DSP

um módulo de 16 canais. O módulo ADC dispõe de 16 registradores para armazenar o resultado das conversões. Uma conversão pode ser iniciada por *software*, pelo módulo *Pulse-Width Modulation* (PWM) ou pela *External Interrupt 2* (XINT2).

A rotina ADC_cal() é programada pelo fabricante na memória *One Time Programmable* (OTP). O *boot* através da memória ROM automaticamente chama a rotina ADC_cal() para inicializar o módulo ADC com dados específicos de calibração. Em operação normal, este processo ocorre automaticamente sendo desnecessário qualquer tipo de intervenção do usuário.

O valor digital é mostrado na Equação 3.1, desde que a entrada respeite os limites citados na última linha da Tabela 7. Como o conversor possui 12 bits de resolução, o valor digital está na faixa entre 0 e 4.095.

Tabela 7: Características do ADC

Resolução	12 bits
Quantidade de canais	16
Tempo de conversão	80ns
Arquitetura	SAR
Tensão analógica	0 a 3V

$$VD = 4095 \times \frac{V_{ain}}{3} \tag{3.1}$$

onde VD é o Valor Digital e V_{ain} é a Tensão Analógica de Entrada.

Outro aspecto importante do ADC é a sensibilidade, que consiste no quociente entre a variação de uma indicação de um sistema de medição e a variação correspondente do valor da grandeza medida (*Vocabulario Internacional de Metrologia*, 2008). A sensibilidade do ADC do DSP 28335 é mostrada na Equação 3.2.

$$S = \frac{V_{max} - V_{min}}{4095} = \frac{(3V - 0V)}{4095} = 0,7324mV \tag{3.2}$$

onde V_{max} é o Limite Superior de Tensão e V_{max} é o Limite Inferior de Tensão.

3.2.3 Interrupções

O bloco *Peripheral Interrupt Enable* (PIE), mostrado na Figura 8, do DSP é responsável pelo gerenciamento das interrupções. O DSP possui 96 interrupções que são agrupadas em 12 grupos de oito. Das 96 possíveis interrupções, apenas 43 são utilizadas, sendo o restante reservado para dispositivos futuros (Texas Instruments, 2009*b*). Cada grupo alimenta uma das 12 linhas de interrupção de núcleo (INT1 a INT12). Cada uma das 96

interrupções possui um vetor armazenado em um bloco dedicado da memória RAM, que pode ser modificado pelo usuário.

Cada uma das interrupções pode ser habilitada ou desabilitada através do bloco PIE. A Figura 9 mostra em diagrama de blocos a requisição de uma interrupção multiplexada.

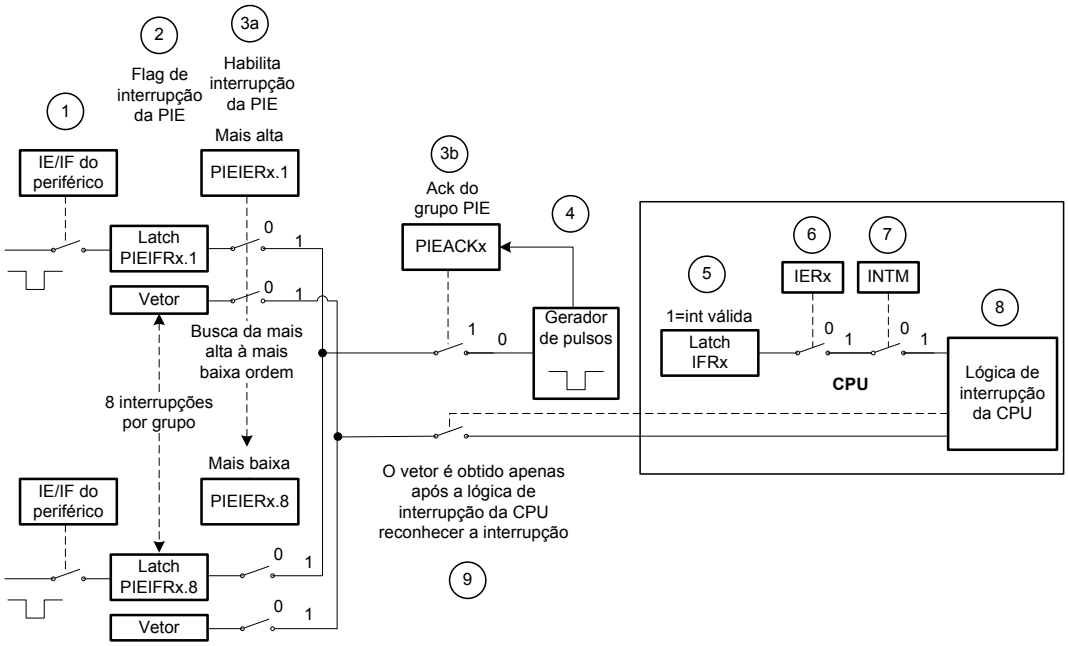

Figura 9: Fluxograma de requisição de uma interrupção multiplexada

A sequência de passos abaixo explica como a CPU reconhece e atende a um pedido de interrupção do bloco PIE:

1. Qualquer periférico ou interrupção ligado(a) ao PIE gera uma interrupção. Se as interrupções estiverem habilitadas no nível do periférico então a solicitação de interrupção é enviada ao módulo PIE.
2. O módulo PIE reconhece que a interrupção *y* do grupo *x* do PIE (INTx.y) afirmou uma interrupção e o *bit* de *flag* de interrupção do PIE apropriado é setado: PIEIFRx.y = 1.
3. Para que a solicitação de interrupção seja enviada do PIE à CPU, ambas as condições expressas abaixo têm que ser verdadeiras:
   (a) O *bit* apropriado de habilitação tem que estar setado (PIEIERx.y = 1) e
   (b) O *bit* PIEACKx do grupo tem que ser igual a zero.
4. Se ambas as condições acima forem verdadeiras, então a requisição de interrupção é enviada à CPU e o *bit* de *ack* é novamente setado (PIEACKx = 1). O *bit* PIE-

ACKx permancerá setado até que seja manualmente zerado para indicar que outras interrupções do mesmo grupo possam ser enviadas do PIE à CPU.

5. O *flag* de interrupção da CPU é setado (CPU IFRx = 1) para indicar que a interrupção *x* está pendente a nível de CPU.
6. Se a interrupção da CPU estiver habilitada (CPU IER *bit* x = 1 ou DBGIER *bit* x = 1) e a máscara de interrupção global estiver zerada (INTM = 0) então a CPU servirá a INTx.
7. A CPU reconhece a interrupção e salva o contexto automaticamente, zera o *bit* IER, seta INTM e zera EALLOW.
8. A CPU então requisita o vetor apropriado do PIE.
9. Para interrupções multiplexadas, o módulo PIE utiliza o valor atual dos registradores PIEIERx e PIEIFRx para decodificar qual endereço do vetor deve ser usado. Há duas possíveis situações:
   (a) O vetor com a interrupção de maior prioridade do grupo que está habilitado no registrador PIEIERx, e a *flag* setada como pendente no PIEIFRx é buscada e usada como endereço de desvio. Desta maneira mesmo que uma interrupção de mais alta prioridade fosse setada após o passo 7, esta será servida primeiro.
   (b) Se nenhuma interrupção com *flag* setada no grupo esteja habilitada, então o PIE responderá com o vetor para a interrupção de mais alta prioridade do grupo. Ou seja, o endereço de desvio usado para INTx.1. Este comportamento corresponde às instruções de TRAP ou INT nos processadores da família C2000.

Neste ponto, o *bit* PIEIFRx.y é zerado e a CPU desvia para o vetor da interrupção buscado pelo PIE.

A Tabela 8 mostra a representação dos vetores de interrupção. A prioridade de interrupções é decrescente da esquerda para a direita e de cima para baixo. O vetor da *Timer0 Interrupt* (TINT0), por exemplo, está localizado na posição 0xD4C da memória RAM.

3.2.4 Interface SCI

Conforme citado na Tabela 6, o DSP possui três módulos *Serial Communications Interface* (SCI): SCI-A, SCI-B e SCI-C. Apenas a SCI-A e a SCI-B podem ser configurados para funcionar no modo *Universal Asynchronous Receiver Transmitter* (UART). Utilizando o modo UART e um *transceiver* UART-EIA232 é possível, por exemplo, estabelecer uma comunicação EIA232 entre o DSP e um computador. Tanto a SCI-A quanto

Tabela 8: Tabela de vetores de interrupção

	INTx.8	**INTx.7**	**INTx.6**	**INTx.5**	**INTx.4**	**INTx.3**	**INTx.2**	**INTx.1**
INT1.y	WAKEINT (LPM/WD) 0xD4E	TINT0 (TIMER 0) 0xD4C	ADCINT (ADC) 0xD4A	XINT2 Ext. int. 2 0xD48	XINT1 Ext. int. 1 0xD46	Reservado - 0xD44	SEQ2INT (ADC) 0xD42	SEQ1INT (ADC) 0xD40
INT2.y	Reserved - 0xD5E	Reserved - 0xD5C	EPWM6_TZINT (ePWM6) 0xD5A	EPWM5_TZINT (ePWM5) 0xD58	EPWM4_TZINT (ePWM4) 0xD56	EPWM3_TZINT (ePWM3) 0xD54	EPWM2_TZINT (ePWM2) 0xD52	EPWM1_TZINT (ePWM1) 0xD50
INT3.y	Reserved - 0xD6E	Reserved - 0xD6C	EPWM6_INT (ePWM6) 0xD6A	EPWM5_INT (ePWM5) 0xD68	EPWM4_INT (ePWM4) 0xD66	EPWM3_INT (ePWM3) 0xD64	EPWM2_INT (ePWM2) 0xD62	EPWM1_INT (ePWM1) 0xD60
INT4.y	Reserved - 0xD7E	Reserved - 0xD7C	ECAP6_INT (eCAP6) 0xD7A	ECAP5_INT (eCAP5) 0xD78	ECAP4_INT (eCAP4) 0xD76	ECAP3_INT (eCAP3) 0xD74	ECAP2_INT (eCAP2) 0xD72	ECAP1_INT (eCAP1) 0xD70
INT5.y	Reserved - 0xD8E	Reserved - 0xD8C	Reserved - 0xD8A	Reserved - 0xD88	Reserved - 0xD86	Reserved - 0xD84	EQEP2_INT (eQEP2) 0xD82	EQEP1_INT (eQEP1) 0xD80
INT6.y	Reserved - 0xD9E	Reserved - 0xD9C	MXINTA (McBSP-A) 0xD9A	MRINTA (McBSP-A) 0xD98	MXINTB (McBSP-B) 0xD96	MRINTB (McBSP-B) 0xD94	SPITXINTA (SPI-A) 0xD92	SPIRXINTA (SPI-A) 0xD90
INT7.y	Reserved - 0xDAE	Reserved - 0xDAC	DINTCH6 (DMA6) 0xDAA	DINTCH5 (DMA5) 0xDA8	DINTCH4 (DMA4) 0xDA6	DINTCH3 (DMA3) 0xDA4	DINTCH2 (DMA2) 0xDA2	DINTCH1 (DMA1) 0xDA0
INT8.y	Reserved - 0xDBE	Reserved - 0xDBC	SCITXINTC (SCI-C) 0xDBA	SCIRXINTC (SCI-C) 0xDB8	Reserved - 0xDB6	Reserved - 0xDB4	I2CINT2A (I2C-A) 0xDB2	I2CINT1A (I2C-A) 0xDB0
INT9.y	ECAN1INTB (CAN-B) 0xDCE	ECAN0INTB (CAN-B) 0xDCC	ECAN1INTA (CAN-A) 0xDCA	ECAN0INTA (CAN-A) 0xDC8	SCITXINTB (SCI-B) 0xDC6	SCIRXINTB (SCI-B) 0xDC4	SCITXINTA (SCI-A) 0xDC2	SCIRXINTA (SCI-A) 0xDC0
INT10.y	Reserved - 0xDDE	Reserved - 0xDDC	Reserved - 0xDDA	Reserved - 0xDD8	Reserved - 0xDD6	Reserved - 0xDD4	Reserved - 0xDD2	Reserved - 0xDD0
INT11.y	Reserved -	Reserved -	Reserved -	Reserved -	Reserved -	Reserved -	Reserved -	-
INT12.y	LUF (FPU) 0xDFE	LVF (FPU) 0xDFC	Reserved - 0xDFA	XINT7 Ext. Int. 7 0xDF8	XINT6 Ext. Int. 6 0xDF6	XINT5 Ext. Int. 5 0xDF4	XINT4 Ext. Int. 4 0xDF2	XINT3 Ext. Int. 3 0xDF0

a SCI-B possuem este *transceiver*, mas também é possível utilizar cada um dos módulos sem o *transceiver* (via conector de expansão, disponível na placa de desenvolvimento). O módulo SCI-A pode ser utilizado também através de um conector DB9, disponível na placa de desenvolvimento.

Tabela 9: Valores do registrador de taxa de transmissão da SCI

Taxa ideal	**BRR (hex)**	**Taxa atual**	**Erro (%)**
2.400	7A0	2.400	0
4.800	3D0	4.798	-0,04
9.600	1E7	9.606	0,06
19.200	F3	19.211	0,06
38.400	79	38.422	0,0005
57.600	50	57.870	0,004
115.200	28	114.329	-0,007

Os módulos SCI podem ser configurados com 65.536 diferentes taxas de transferência. A Tabela 9 exemplifica a relação entre a taxa desejada e o valor do *Baud Rate Register* (BRR), considerando o *Low Speed Peripheral Clock* (LSPCLK) como sendo 37,5MHz.

Podem também ser disparados por interrupção de recepção e/ou transmissão. Por padrão, quando as duas interrupções de uma SCI estão habilitadas, a de recepção tem prioridade sobre a de transmissão, reduzindo a probabilidade de perda de dados na recepção. Podem ainda funcionar em modo *half-duplex* ou *full-duplex*.

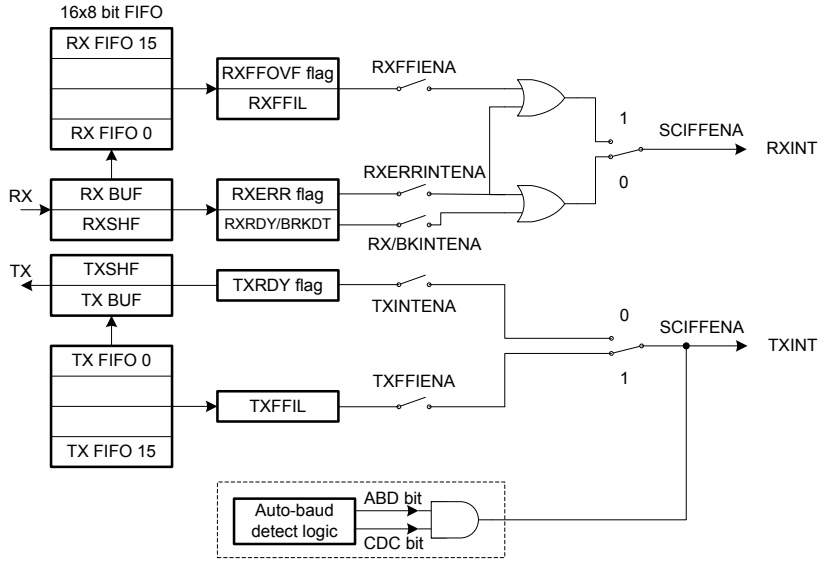

Figura 10: Flags de interrupção e lógica de habilitação da SCI FIFO

Além disso, cada módulo possui uma *First-In First-Out* (FIFO) de uso opcional que pode ser configurada de 1 a 16 *bytes*. Por exemplo: a utilização de uma FIFO com 16 *bytes* aliada à interrupção de recepção permite que a última seja disparada apenas quando 16 *bytes* forem recebidos (e, consequentemente, armazenados na FIFO).

Esse recurso diminui a sobrecarga da CPU, uma vez que apenas uma interrupção é disparada, ao invés de 16. Consequentemente, as operações de salvar e restaurar contexto são realizadas apenas uma vez, ao invés de 16. Cada interface SCI possui uma FIFO para recepção e outra para transmissão. A Figura 10 exibe a lógica de utilização da SCI com FIFO e interrupções.

3.2.5 Ambiente de programação

O *Integrated Development Environment* (IDE) utilizado para programação do DSP é o *Code Composer Studio* (CCS), que permite ao usuário escrever o código em *Assembly*, C ou C++. O CCS possui um recurso chamado *watch*, que permite monitorar variáveis localizadas na memória RAM em tempo de execução. A Figura 11 mostra a interface do CCS. Nesta Figura pode-se observar no canto inferior direito a janela *watch* monitorando uma variável (Texas Instruments, 2006).

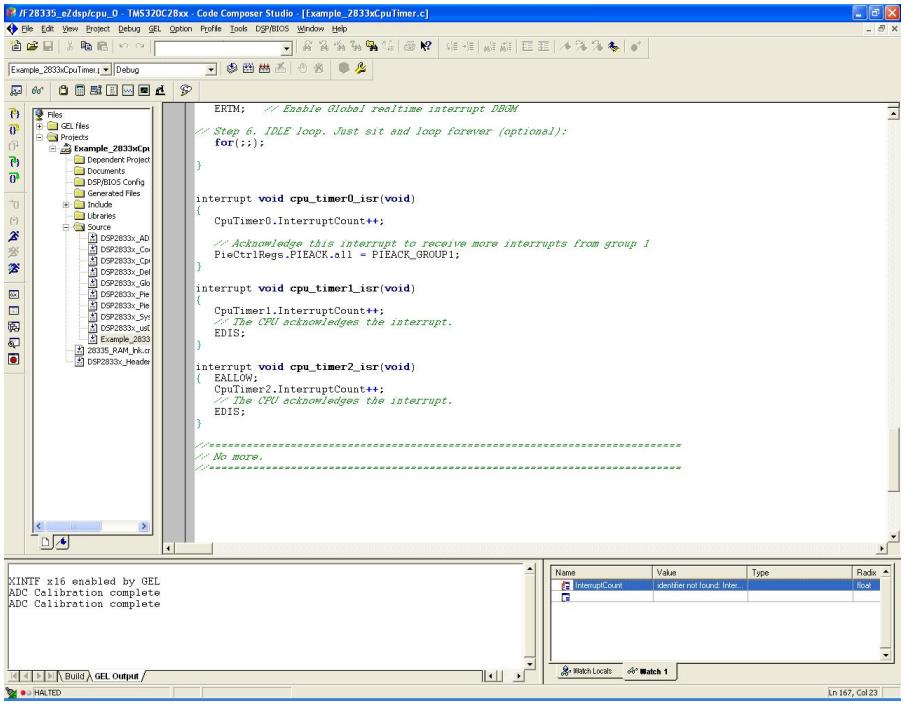

Figura 11: Ambiente de desenvolvimento CCS

O CCS comunica-se com o DSP através de uma conexão *Universal Serial Bus* (USB)

ou através de um emulador. A Figura 12 mostra um modelo de emulador de baixo custo para DSPs da família C2000 da Texas Instruments (Spectrum Digital Inc, 2008).

Figura 12: Emulador XDS510LC

O CCS também permite a utilização de *breakpoints*, que são bastante úteis na fase de *debug* do programa. *Breakpoints* introduzem uma pausa na execução do programa e atualizam a janela *watch*. O gerenciador de *breakpoints* (mostrado na Figura 13) permite configurar *breakpoints* individualmente, e também disponibiliza várias ações a serem associadas aos *breakpoints*.

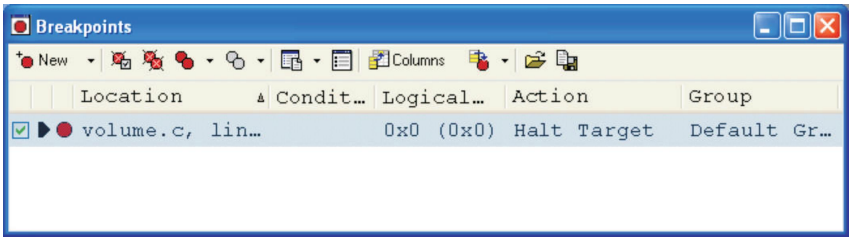

Figura 13: Gerenciador de *breakpoints*

As possíveis ações a serem associadas a um *breakpoint* são:

- Transferir dados de um arquivo gravado em um PC para um vetor a ser utilizado pelo algoritmo gravado em DSP
- Transferir dados de um vetor no DSP para um arquivo em um PC
- Atualizar uma janela (*watch*, por exemplo) com novos dados

A Figura 14 mostra a janela de configuração de parâmetros que surge quando a ação escolhida para o *breakpoint* é ler dados de um arquivo gravado em PC. Abaixo segue uma descrição detalhada dos campos:

- *File*: Caminho (no PC) onde o arquivo está salvo
- *Wrap around*: se marcada esta opção, o arquivo é tratado como um *buffer* circular, ou seja, quando o fim do arquivo é atingido, a leitura recomeça do início
- *Start address*: especifica em que variável (ou vetor) os dados do arquivo serão salvos
- *Length*: determina quantos *bytes* serão lidos do arquivo cada vez que o *breakpoint* é atingido
- Se, ao invés de ser escolhida a ação de ler de um arquivo, fosse escolhida a ação escrever em um arquivo, a janela de configuração de parâmetros teria o campo formato. Neste campo o usuário especifica em que formato o arquivo de saída será gravado. As opções são: *Hex*, *Integer*, *Long*, *Float*, *Coff* ou *Addressable Unit*.

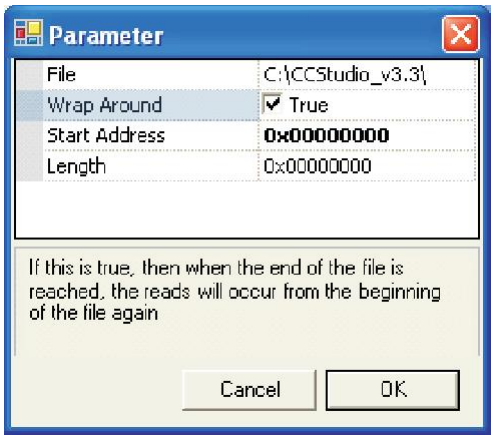

Figura 14: Parâmetros de leitura de um arquivo usando *breakpoint*

O fabricante do DSP disponibiliza um conjunto de exemplos de código em linguagem C que utilizam os diversos módulos do DSP (Texas Instruments, 2008*a*).

3.3 Condicionamento do sinal do sensor

O conversor corrente tensão é necessário para adequar a saída do sensor (4-20mA) à entrada do ADC do DSP (0-3V). Para isso, foi colocada uma carga de 150Ω em série com a saída do sensor e a tensão medida entre os terminais da carga. Entretanto, o sinal de

saída do sensor está sujeito a ruído. Então foi utilizado um amplificador de instrumentação com ganho unitário.

O amplificador de instrumentação é um circuito analógico que fornece uma saída baseada na diferença entre duas entradas, podendo ser multiplicada por um fator de escala (BOYLESTAD; NASHELSKY, 2004). O amplificador de instrumentação garante ainda uma alta impedância de entrada e uma baixa impedância de saída, ou seja, o circuito funciona também como um isolador. A isolação é importante na proteção do DSP, já que este modelo não possui as entradas do ADC isoladas.

Na saída do circuito foi colocado um diodo zener para limitar a tensão de saída em 3V. A Figura 15 mostra o esquema do circuito utilizado. Nota-se que o circuito está duplicado, pois são utilizados dois sensores.

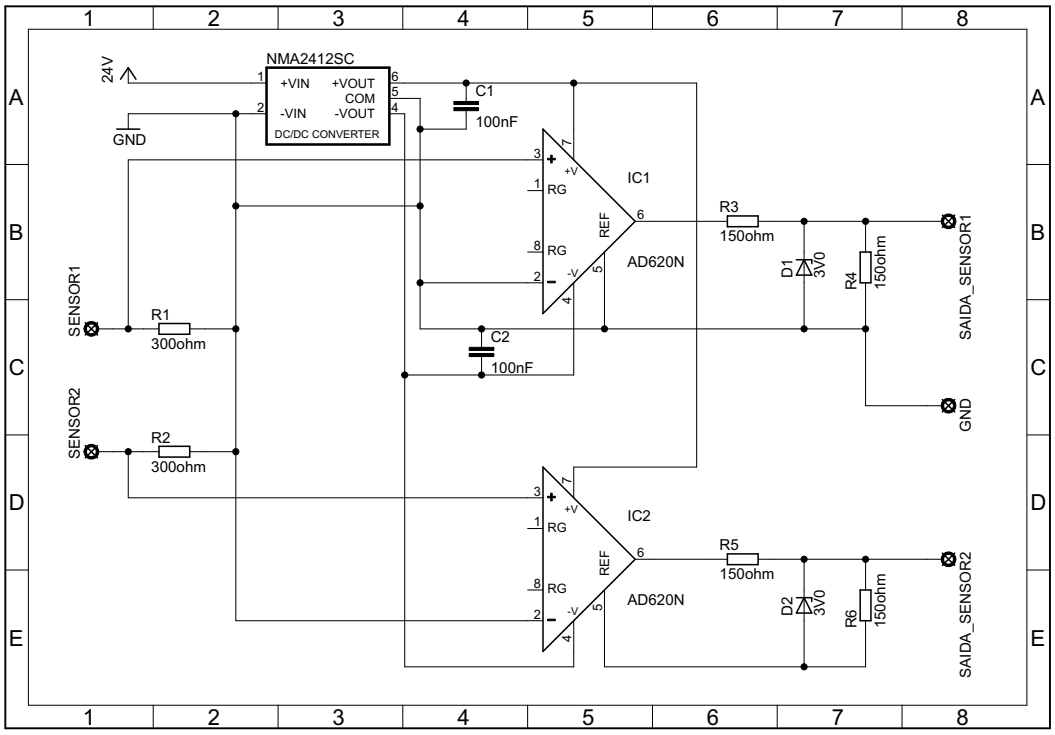

Figura 15: Circuito condicionador de sinais

Finalmente, foi confeccionada uma placa de circuito impresso contendo dois amplificadores de instrumentação, sendo um para cada sensor.

3.4 GPS

O GPS utilizado é o ET332, da Globalsat®. A Figura 16 mostra o GPS.

O GPS fornece, dentre outras informações, o horário UTC no formato *hhmmss.sss*.

Figura 16: GPS

Zero UTC corresponde à meia-noite em *Greenwich England* situada na longitude zero. Como o Brasil está no terceiro fuso horário a oeste de *Greenwich*, é necessário subtrair 3 horas do horário fornecido pelo GPS. Além disso, o GPS garante um desvio do horário fornecido em relação ao horário dos satélites de, no máximo, 1μs (GlobalSat Technology Corporation, 2008).

Para o correto funcionamento do GPS é necessário que a antena do mesmo seja instalada em um local sem obstáculos, para permitir visada direta com os satélites.

O GPS utiliza o protocolo NMEA 0183 e fornece as seguintes instruções: *Global Positioning System Fixed Data* (GGA), *GNSS DOP and Active Satellites* (GSA), *GNSS Satellites in View* (GSV) e *Recommended Minimum Specific GNSS Data* (RMC). A taxa de atualização padrão das mensagens GGA, GSA e RMC é 1Hz e a mensagem GSV é 5Hz (SiRF Technology, Inc, 2008). A Tabela 10 mostra o tamanho de cada uma dessas mensagens.

Tabela 10: Tamanho das mensagens de saída do GPS

Mensagem	**Tamanho**
GGA	77 *bytes*
GSA	58 *bytes*
GSV	70 *bytes*
RMC	71 *bytes*
Total	276 *bytes*

Destas, as instruções GGA e RMC fornecem o tempo. Entretanto, a última apresenta duas vantagens: tamanho de quadro menor (facilitando a decodificação por parte do DSP) e o campo *status*, que informa se o dado fornecido pelo GPS é válido ou não. A Tabela 11 mostra um exemplo de mensagem RMC e descreve cada um dos seus campos.

Tabela 11: Formato da mensagem RMC

Nome	Exemplo	Unidade	Descrição
Identificador da mensagem	$GPRMC		Cabeçalho da mensagem RMC
Horário UTC	161229.487		hhmmss.sss
Campo de status	A		A=dado válido ou V=dado inválido
Latitude	3723.2475		ddmm.mmmm
Indicador N/S	N		N=norte or S=sul
Longitude	12158.3416		dddmm.mmm
Indicador E/W	W		E=leste or W=oeste
Velocidade sobre o solo	0.13	nós	
Rumo	309.62	graus	Verdadeiro
Data	120598		ddmmyy
Variação magnética		graus	E=leste or W=oeste
Checksum	*10		
<CR><LF>			Terminador da mensagem

Entretanto, é possível configurar o envio de mensagens do GPS. As opções são: desabilitar uma mensagem, modificar a taxa de atualização de uma mensagem e solicitar o envio de uma mensagem apenas uma vez.

O GPS, ao receber uma mensagem enviada pelo usuário, realiza o processamento da mesma e armazena possíveis alterações em memória RAM. Este GPS possui uma bateria de *backup*, que armazena as configurações quando o módulo GPS não está alimentado. O fabricante sugere que para o correto armazenamento das informações a tensão na bateria deve estar entre 2,5 e 3,6V. A Tabela 12 mostra todas as mensagens que podem ser enviadas ao GPS.

Quando ligado, o GPS leva um certo tempo para fornecer informações confiáveis. Este tempo é chamado *cold start* e, segundo o fabricante, varia entre 1s e 42s.

A saída do GPS é serial, utiliza o padrão *Transistor-Transistor Logic* (TTL) e uma taxa de 4.800bps para comunicação.

A alimentação do GPS vai de 3,8 a 6,5V e consome 75mA.

3.5 Conversor RS232/RS485

Este equipamento converte as informações do padrão EIA232 no padrão EIA485 e vice-versa. O modelo utilizado é o FM-DC.001.004 TX/RX da Flexmedia®, que suporta uma taxa de transferência de dados de até 115.200bps. A Figura 17 mostra o conversor.

Tabela 12: Mensagens de entrada do GPS

Mensagem	Nome	Descrição
100	SetSerialPort	Configura parâmetros do PORT A e protocolo
101	NavigationInitialization	Parâmetros necessários para iniciar o uso de X/Y/Z
102	SetDGPSPort	Configura parâmetros do PORT B para entrada DGPS
103	Query/Rate Control	Solicita mensagem padrão NMEA e/ou ajusta frequência de envio
104	LLA Navigation Initialization	Parâmetros necessários para usar Lat/Lon/Alt
105	Development Data On/Off	Liga ou desliga as mensagens de desenvolvimento
106	Select Datum	Escolha do *datum* usado para as transformações de coordenadas
107	Proprietary	Reservado
108	Proprietary	Reservado
110	Extended Ephemeris Debug	Debug estendido
200	Marketing Software Configuration	Ajusta os parâmetros do protocolo
MSK	MSK Receiver Interface	Mensagem de comando para um receptor rádio-beacon MSK

3.6 Conversor RS485/Ethernet

Este equipamento converte dados do padrão EIA485 no padrão Ethernet e vice-versa. O modelo utilizado no projeto SONIC é o ESC713 da HI tecnologia. Este conversor suporta uma taxa de transferência do padrão RS485 de até 115.200bps e a versão 10BaseT do padrão Ethernet. A Figura 18 mostra o conversor.

3.7 Supervisório

O propósito do *software* de supervisão é promover a interface homem/máquina, permitindo ao usuário inspecionar, executar comandos de operação e supervisão disponibilizados em telas configuradas, estas representando um processo, através de um microcomputador.

O supervisório está localizado na sala de controle da estação ETB. É empregado um servidor responsável por converter os dados do protocolo Modbus em protocolo DDE.

Figura 17: Conversor RS232/RS485

Figura 18: Conversor RS485/Ethernet

As telas de supervisão foram desenvolvidas utilizando o *software* InTouch10, responsável por requisitar dados ao servidor (SILVA, 2009).

3.8 Arquitetura do sistema em campo

A Figura 19 mostra o diagrama de blocos do projeto SONIC empregado para monitorar a região entre ETA e ETB.

3.9 Conclusão

Este capítulo apresentou o objetivo do projeto SONIC e também a função de cada um dos equipamentos utilizados neste. Uma atenção especial foi dada ao DSP, que é o elemento responsável por realizar o processamento do sinal enviado pelos sensores de pressão.

Foi também mostrada a metodologia do projeto em campo.

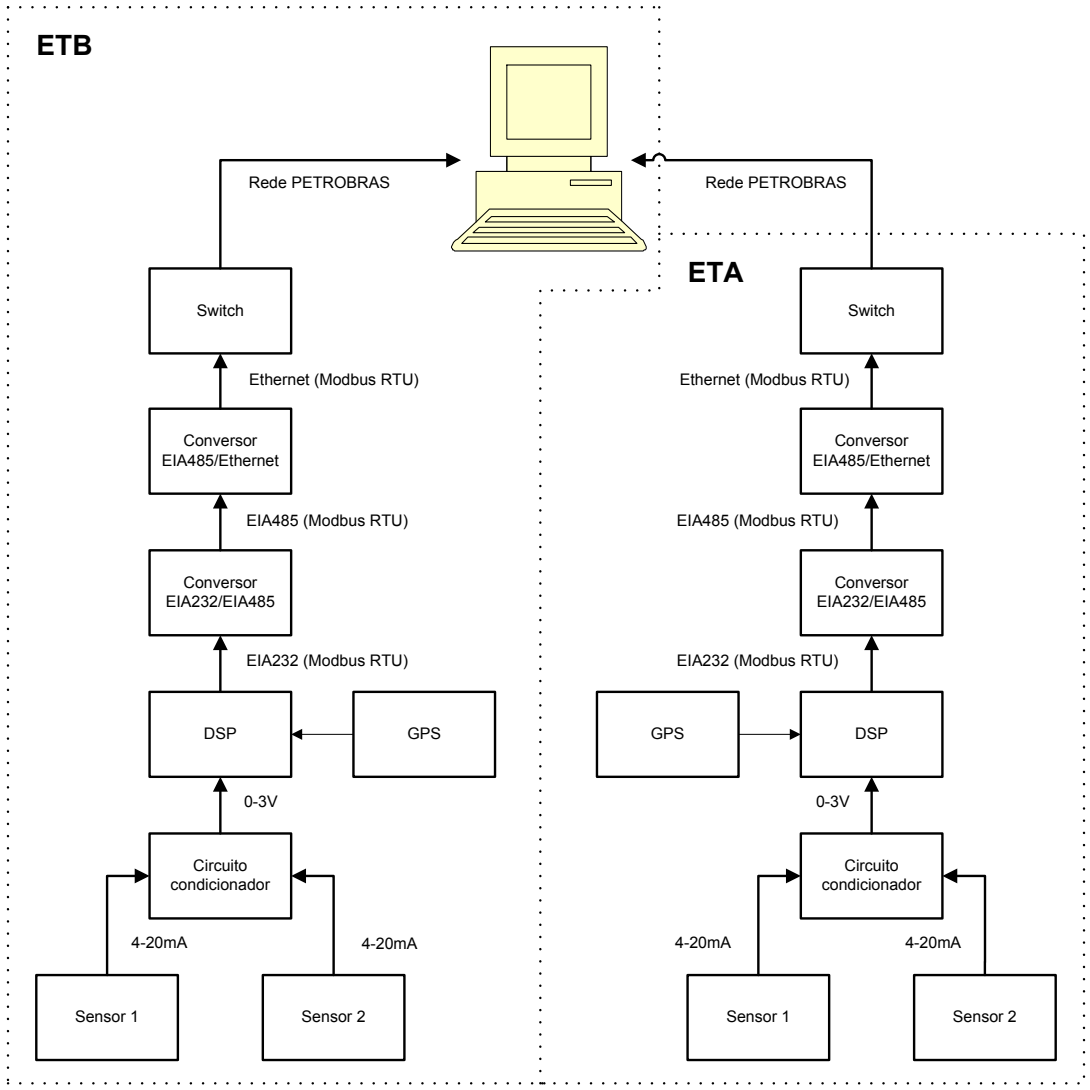

Figura 19: Arquitetura do sistema

Capítulo 4

Sistema proposto

Este Capítulo expõe as partes constituintes do programa gravado em DSP, que são:

1. Aquisição do sinal de pressão enviado pelos sensores;
2. Processamento das amostras;
3. Comunicação com o GPS;
4. Comunicação com o supervisório.

Inicialmente há uma explanação sobre cada uma das partes acima citadas separadamente. Em seguida são analisados os resultados obtidos pelo programa do DSP contendo todas as partes reunidas.

Como se trata de um sistema de tempo real, um aspecto de extrema importância é o tempo de execução de cada uma das funções. Para mensurar esta variável foi utilizado um pino de GPIO do DSP, configurado como saída. Desta forma, para mensurar quanto tempo é necessário para que um trecho de código seja executado, basta pôr a saída em nível alto antes do início do trecho de código e, imediatamente após o final do trecho de código, pôr a saída em nível baixo. Utilizando um osciloscópio digital para observar a duração do pulso é possível saber quanto tempo durou a execução do código.

4.1 Comunicação com o GPS

O DSP comunica-se com o GPS para adquirir o horário enviado pelos satélites. A Tabela 13 mostra os parâmetros utilizados para estabelecer a conexão entre o DSP e o GPS.

Conforme citado na Seção 3.4, o GPS envia as mensagens do protocolo NMEA0183 a cada 1s (exceto a mensagem GSV, que é enviada a cada 5s). A mensagem de interesse

Tabela 13: Configuração da SCI-B para comunicação com o GPS

Bits por segundo	4.800
Bits de dados	8
Paridade	Nenhum
Bits de parada	1
Controle de fluxo	Nenhum

é a RMC, que possui 68 caracteres[1].

No DSP é criado um vetor de 256 posições para armazenar todas as mensagens enviadas pelo GPS. Uma vez preenchido o vetor é feita uma busca exaustiva (também conhecida como força bruta) pelo identificador da mensagem RMC ($GPRMC). Uma vez encontrado o padrão, o *bit* de status é checado, para garantir que a comunicação do GPS com os satélites foi bem-sucedida e, consequentemente, a informação disponibilizada pelo GPS é válida.

Neste ponto as interrupções são desabilitadas, pois o armazenamento do horário é considerado uma seção crítica de código. Em seguida as interrupções são reabilitadas. Caso não seja encontrado o identificador padrão da mensagem, o processo é reiniciado. A Figura 20 mostra o fluxograma de aquisição do horário do GPS.

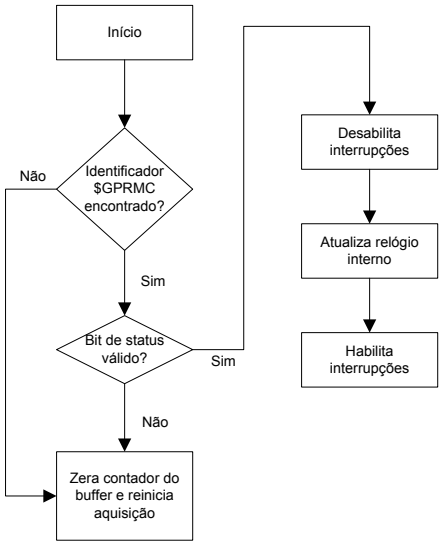

Figura 20: Fluxograma de aquisição do horário do GPS

[1]Se a comunicação entre o GPS e os satélites não for bem-sucedida, a mensagem pode ser incompleta e, portanto, terá menos de 68 caracteres

4.2 Aquisição do sinal de pressão

Esta é a tarefa de mais alta prioridade, tendo em vista que a detecção é baseada na análise da pressão. A aquisição deve ser contínua. Caso contrário o resultado do processamento pode não ser confiável.

Nesta etapa são adquiridos os dados dos dois sensores, sendo utilizado um canal de conversão para cada sensor. A aquisição é sequencial, sendo realizada a aquisição do sinal do sensor 1 pelo canal A0 do ADC e, em seguida, a aquisição do sinal do sensor 2 pelo canal A1 do ADC. A taxa de aquisição utilizada é 1ms. Para isso foi configurada a TINT**0** para ocorrer periodicamente a cada 1ms.

O resultado da conversão é armazenado em um vetor, chamado *vetor de aquisição*. São criados dois vetores de aquisição, já que são utilizados dois canais de conversão.

Quando os vetores de aquisição estão completos, o conteúdo dos mesmos é copiado para outros dois vetores, chamados *vetores de processamento*. Após a cópia, o *flag* de processamento é setado. A Figura 21 mostra o fluxograma do ADC.

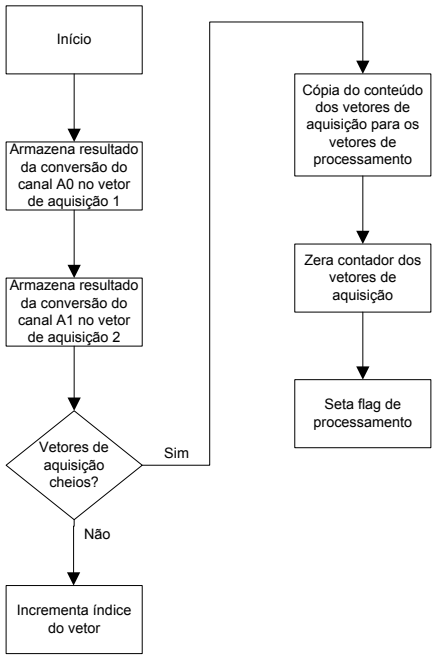

Figura 21: Fluxograma de utilização do ADC

4.3 Relógio interno

Conforme citado na Seção 3.4, o GPS utilizado fornece o horário UTC no formato *hhmmss.sss*, ou seja, com exatidão de um milissegundo. De posse deste horário, é possível

atualizar um relógio interno no DSP, para evitar a comunicação constante com o GPS, a fim de obter o horário. É importante ressaltar que a operação de atualizar o relógio interno com o horário fornecido pelo GPS é considerada uma seção crítica de código e, portanto, esta operação é feita com interrupções desabilitadas.

Conforme dito na Seção 4.2, foi configurada uma TINT**0** com período de 1ms, que coincide com a exatidão do horário fornecido pelo GPS. Além de realizar a conversão analógico-digital do sinal dos sensores de pressão, a TINT**0** também incrementa um milissegundo no horário interno do DSP. A Figura 22 mostra o fluxograma do relógio interno do DSP.

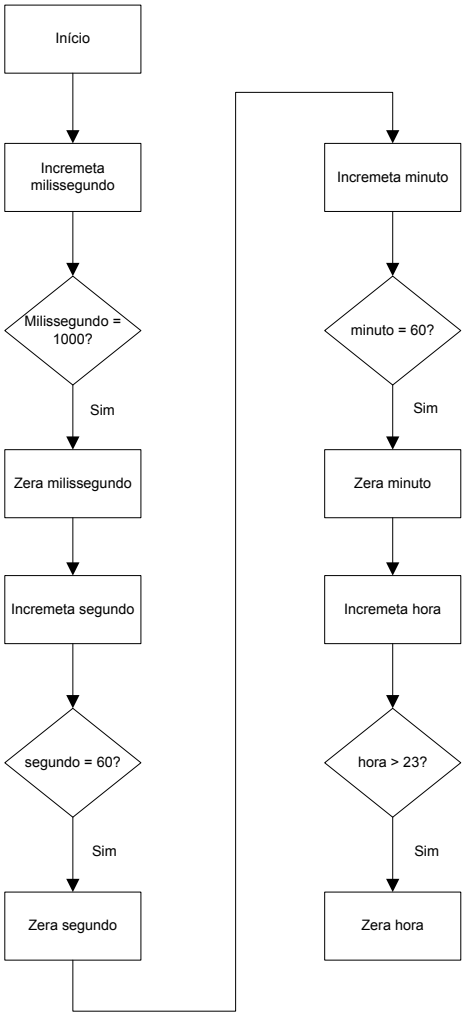

Figura 22: Fluxograma do relógio interno

Tabela 14: Configuração da SCI-A para comunicação com o supervisório

Bits por segundo	19.200
Bits de dados	8
Paridade	Nenhum
Bits de parada	2
Controle de fluxo	Nenhum

4.4 Comunicação com o supervisório

O DSP utiliza a interface SCI-A para a comunicação com o supervisório. A Tabela 14 mostra os parâmetros utilizados para estabelecer esta conexão.

Sendo assim, o supervisório envia um quadro de requisição de dados para o DSP. O último utiliza a *SCI RX Interrupt A* (SCIRXINTA) para armazenar a mensagem. O quadro enviado pelo supervisório contém oito *bytes* (SILVA et al., 2009). Desta forma foi configurada a interface SCI-A para utilizar a FIFO com oito *bytes*. Isso implica em disparar a interrupção de recepção da interface serial apenas quando os oito *bytes* enviados pelo supervisório chegarem ao DSP. Isso diminui o *overhead* da CPU, uma vez que cada chamada de interrupção possui, além do tempo de execução da interrupção, os tempos de resposta e de retorno da interrupção.

Após o armazenamento da mensagem, um *bit* de *flag* é setado para indicar que a decodificação da mensagem recebida deve ser iniciada.

Após a decodificação da mensagem, é montado no DSP um quadro de resposta, a ser enviado ao supervisório. Este quadro possui a pressão média verificada por cada sensor, a indicação da ocorrência de vazamento ou não e os tempos de término de cada um dos vetores de processamento. Este quadro possui 23 *bytes*.

O supervisório envia periodicamente um quadro ao DSP. A taxa com que esse envio ocorre é dependente do tempo de execução de todas as funções. Uma vez encontrado este intervalo de tempo para a situação de execução de pior caso é possível estabelecer o período adequado para que o supervisório requisite dados ao DSP.

4.5 Processamento

A técnica utilizada nos algoritmos é baseada na análise da queda do nível de pressão no duto. Esta técnica utiliza uma combinação da transformada *Wavelet* com Redes Neurais (PAIVA et al., 2009). A Figura 23 mostra o diagrama de blocos dos algoritmos de processamento.

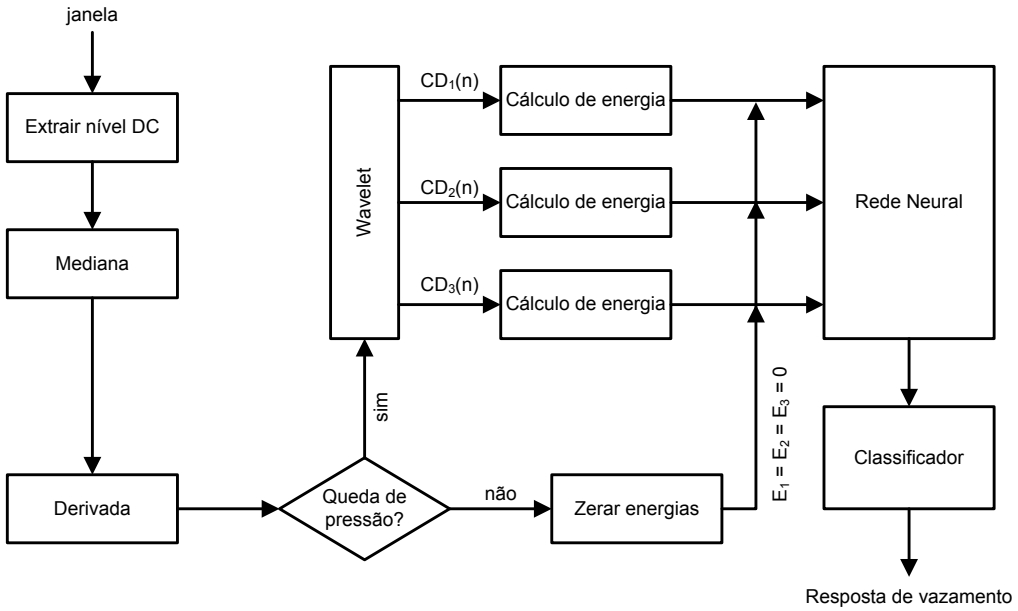

Figura 23: Metodologia de algoritmos

4.6 Resultados

A programação do DSP foi feita utilizando a versão 3.3.59.4 do CCS.

Os testes iniciais consistiram em avaliar a execução das partes do programa individualmente, ou seja, foi testado apenas o uso da TINT**0** para adquirir o sinal dos sensores de pressão, depois a comunicação com o supervisório, em seguida o processamento e a comunicação com o GPS. Uma vez que a comunicação com o GPS foi bem-sucedida, foi testado o relógio interno do DSP, tendo como base de incremento de tempo a TINT**0**.

Duas abordagens podem ser utilizadas para preencher o buffer de 256 posições: utilizando *polling* ou utilizando interrupção de recepção em conjunto com a FIFO. Neste trabalho as duas abordagens foram testadas. Do ponto de vista de *software*, a primeira abordagem é mais simples de ser implementada. Porém, utilizar *polling* significa manter o processador ocupado. Para preencher o vetor utilizando polling é necessário 1,1s.

Também foi testada a interrupção SCIRXINTB e a FIFO configurada para 16 bytes. Nesta abordagem, a cada 16 bytes recebidos, é disparada a interrupção, que neste caso apenas armazena os 16 bytes no vetor. Como o vetor possui 256 posições, são necessárias 16 interrupções para preencher o vetor. A cópia dos 16 bytes leva 2,04μs. Além disso, ao utilizar interrupção é necessário avaliar também os tempos de resposta e de retorno da interrupção. Como trata-se de uma aplicação de tempo real, deve-se levar em conta os tempos de pior caso. Para o DSP 28335 o tempo de resposta de uma interrupção no pior caso é de 586,96ns e o tempo de retorno de uma interrupção (também em pior

caso) é de 106,72ns (Texas Instruments, 2008*c*). Desta forma o tempo total necessário ao processamento de uma interrupção SCIRXINTB é de

$$t_{sci} = 2,04\mu s + 0,5869\mu s + 0,1067\mu s = 2,7336\mu s \tag{4.1}$$

onde t_{sci} é o tempo total necessário ao processamento de uma interrupção SCIRXINTB.

Como são necessárias 16 interrupções para preencher o vetor, o tempo total necessário é

$$t_{tsci} = 2,7336\mu s \times 16 = 43,7376\mu s \tag{4.2}$$

onde t_{tsci} é o tempo total necessário para preencher as 256 posições do vetor.

Entretanto, uma nova interrupção só é disparada quando a FIFO contém 16 bytes. Ou seja, há um intervalo de tempo entre interrupções. Apesar de a cópia dos 256 bytes ser realizada em 43,7376μs, o tempo entre a ocorrência de interrupções deve ser levado em conta. No total é necessário 1,1s para a cópia dos 256 bytes, porém, a CPU só fica ocupada durante 43,7376μs, ficando livre durante 1,099s.

Independente da abordagem utilizada no pior caso é necessário preencher o vetor duas vezes para encontrar a mensagem RMC. Isso ocorre porque no momento em que o vetor vai começar a ser preenchido pode ser que o GPS já tenha enviado a mensagem RMC e, portanto, só 1s após enviará a mensagem novamente.

Além disso, há uma outra situação a ser considerada. Uma vez preenchido completamente o vetor, ainda que o mesmo contenha a mensagem RMC pode ser que o *bit* de status seja inválido, indicando que a comunicação entre o GPS e os satélites não foi bem sucedida. Conforme citado na Seção 3.4 o GPS utilizado pode levar até 42s após a inicialização para fornecer a mensagem RMC com o *bit* de status válido.

Do ponto de vista do tempo necessário para adquirir o horário fornecido pelo GPS, a abordagem utilizando interrupção e FIFO mostrou-se mais eficiente e, portanto, foi a escolhida. Uma vez adquirido o horário do GPS, a SCIRXINTB é desabilitada, pois não é mais necessária a comunicação com o GPS.

Para os testes de aquisição do sinal de pressão e de simulação de vazamentos foi utilizada a planta do laboratório LAMP, da UFRN. A estrutura utilizada conta com dois tanques, uma bomba de transferência e um oleoduto de 3” e 14,36m de extensão.

Para testar a aquisição de dados, foram instalados dois sensores de pressão PTX7800. A pressão nominal no duto era 3bar. A pressão máxima suportada pelos sensores era 7bar (desconsiderando sobrepressão). Para simular vazamento foi instalada na tubulação do LAMP uma válvula de abertura manual tipo esfera com diâmetro de 3/4”. Para drenar o

Tabela 15: Configuração da SCI-A para comunicação com o *HyperTerminal*

Bits por segundo	115.200
Bits de dados	8
Paridade	Nenhum
Bits de parada	2
Controle de fluxo	Nenhum

volume de fluido vazado foi utilizado um recipiente plástico com capacidade de 200l. A Figura 24 mostra o esquema utilizado.

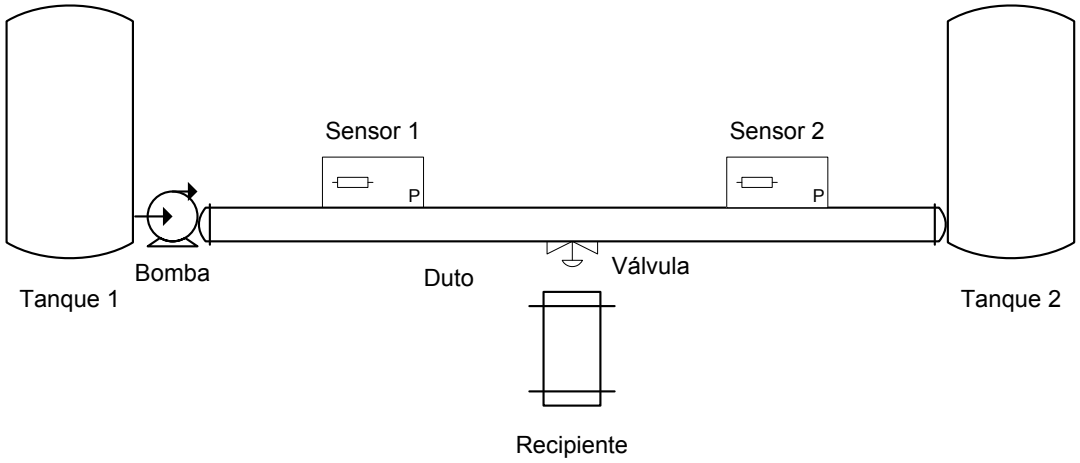

Figura 24: Planta utilizada durante as simulações

Para esse teste foi utilizada comunicação direta entre o DSP e um computador. Para adquirir os dados foi configurada a TINT**0** para ocorrer a cada 1ms. Em seguida, estes eram transmitidos pela SCI-A. No computador foi utilizado o *software HyperTerminal*, que gravava em arquivo os dados dos dois sensores. A Tabela 15 mostra os parâmetros utilizados para configurar a conexão entre o DSP e o *HyperTerminal*.

O sinal de pressão adquirido durante as simulações de vazamento no LAMP é mostrado na Figura 25.

Para testar os algoritmos, inicialmente estes foram escritos utilizando a linguagem C e testados em computador, usando o software DevCpp 4.9.

Em seguida, os algoritmos foram embarcados em DSP. Para testá-los em DSP foram utilizados dois *breakpoints*, sendo o primeiro responsável por carregar dados de um arquivo gravado no PC na memória do DSP para processamento e o último *breakpoint* responsável por gravar o resultado do processamento em um arquivo de saída (em formato *Integer*). A Figura 26 mostra o diagrama de blocos utilizado durante o teste dos algoritmos em DSP.

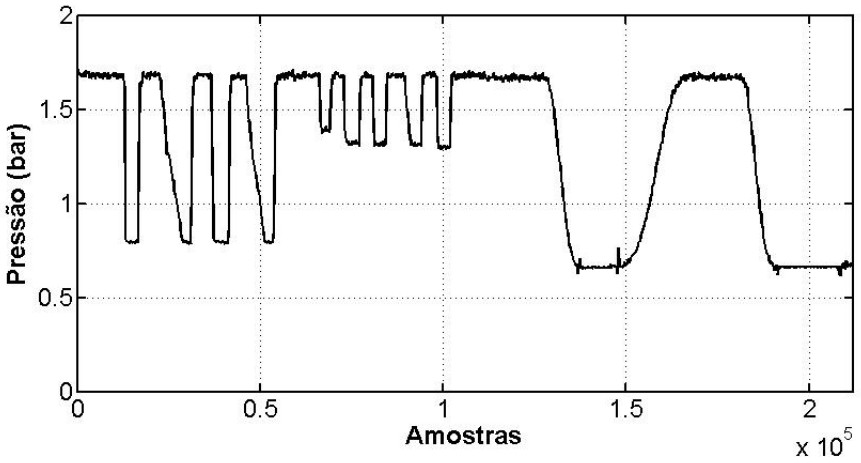

Figura 25: Sinal de pressão

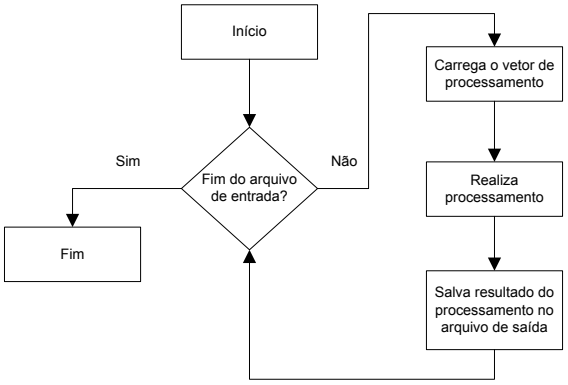

Figura 26: Fluxograma do teste dos algoritmos

A Tabela 16 mostra o tempo de execução de cada um dos algoritmos e, ao final, o tempo necessário ao processamento de todos os algoritmos.

Uma vez validadas cada uma das partes, foi criado um projeto no Code Composer Studio reunindo todas as tarefas citadas anteriormente.

A princípio, a compilação não foi bem-sucedida, pois o código gerado a partir da compilação era maior que a capacidade da memória interna do DSP. A solução encontrada foi ativar a otimização de nível 3 do compilador (Texas Instruments, 2007*b*). Então a compilação foi bem-sucedida.

A otimização do compilador é um recurso que dá liberdade ao programador, pois este pode criar a sua aplicação (a princípio) sem preocupar-se com otimização de código. Ao ativar a otimização, o compilador, durante a fase de geração do código de máquina, otimiza o código de alto nível especificamente para a arquitetura utilizada.

Ainda assim só foi possível compilar o código sem problemas com a alocação do

Tabela 16: Tempo de execução dos algoritmos

Algoritmo	**Tempo de execução (ms)**
Conversão de uma amostra	0,0023
Extrair nível DC	0,1760
Mediana	0,6001
Wavelet CD1	1,2360
Energia 1	0,0070
Wavelet CA1	1,2400
Wavelet CD2	0,5959
Energia 2	0,0035
Wavelet CA2	0,5999
Wavelet CD3	0,2798
Energia 3	0,0018
Zerar energias	0,0120
Rede neural	0,0465
Total	4,8008

código na memória do DSP utilizando o tamanho da janela de processamento igual a 256. Entretanto, as simulações dos algoritmos de processamento no DevCpp mostraram que a utilização do tamanho da janela igual a 512 traz melhores resultados (AVELINO et al., 2009). Então foi necessário modificar o arquivo que define a organização do mapa de memória do DSP. Como pode ser visto no canto superior esquerdo da Figura 8, a memória RAM do DSP é dividida em dez blocos (ou setores). Por padrão, apenas o bloco L0 é destinado à alocação do programa. Porém, é possível unir blocos contíguos a fim de obter uma região de memória com maior capacidade de armazenamento. Neste trabalho foram unidos os blocos L0 e L1, e desta maneira foi possível utilizar o tamanho da janela de processamento dos algoritmos de 512.

Uma vez avaliados os tempos de execução de cada uma das partes do projeto no Code Composer, é possível inferir a taxa mais adequada com que o supervisório deve periodicamente requisitar dados ao DSP. Utilizando a janela de processamento com tamanho 512, são necessários 512ms para preencher o vetor de aquisição. Já o processamento precisa de 4,8008ms para ser concluído, ou seja, quando o processamento termina há um longo período de tempo até que um novo vetor de processamento esteja disponível, que é definido por:

$$\Delta t_p = 512ms - 4,8008ms = 507,1992ms \tag{4.3}$$

onde Δt_p é o intervalo de tempo entre o fim do processamento atual e o início do próximo.

Já o tempo total de processamento, que inclui o tempo necessário ao preenchimento

do vetor de processamento e também o tempo de processamento é dado por:

$$t_t = 512ms + 4,8008ms = 516,8008ms \tag{4.4}$$

onde t_t é o tempo total de processamento.

Entretanto, trata-se de um sistema de tempo real. Não é recomendado tomar os valores calculados como absolutos, sendo uma boa prática estabelecer um tempo adicional extra, como uma espécie de fator de segurança. O tempo de preenchimento do vetor de processamento é fixo, já que o preenchimento ocorre dentro de uma interrupção períodica (TINT**0**). Entretanto o tempo de processamento pode variar para mais ou para menos. Desta forma, considerar-se-á o tempo de processamento no pior caso como sendo três vezes o tempo total mostrado na Tabela 16. Então o novo t_t é dado por:

$$t_t = 512ms + (3 \times 4,8008ms) = 526,4024ms \tag{4.5}$$

Sendo assim, uma taxa razoável para que o supervisório requisite dados ao DSP é 600ms.

A única parte do programa que pode ser classificada como *hard* (tópico exposto na Seção 2.4) é o código executado dentro da TINT**0**, ou seja as funções relativas à aquisição do sinal e o relógio interno. Essas não podem, em hipótese alguma, sofrer atraso.

4.7 Conclusão

Este capítulo abordou o funcionamento de cada uma das partes do código em DSP separadamente. A análise de tempo de resposta de cada uma dessas partes permitiu avaliar o tempo de resposta total do sistema. Ainda assim essa análise não pode ser considerada absoluta. Por esse motivo foi estimada uma tolerância (de tempo) adicional, em relação ao tempo de resposta total do sistema.

Os testes tomaram por base uma metodologia de detecção de vazamentos baseada em Transformada *Wavelet* e Redes Neurais. Nada impede que outra estratégia seja utilizada, como por exemplo a análise frequencial (AZEVEDO, 2009). O que de fato importa é com quanto tempo a técnica escolhida é capaz de fornecer uma resposta, a partir de uma entrada.

Além disso, algumas limitações da abordagem *foreground/background* foram citadas e foi estimada uma taxa para que o supervisório requisite dados ao DSP.

Por último, foram validados o DSP e o GPS escolhidos.

Capítulo 5

Considerações finais e perspectivas

Neste capítulo são apresentadas algumas considerações finais acerca dos resultados obtidos ao longo deste trabalho, bem como algumas sugestões de trabalhos de pesquisa futuros que podem tomar este como base.

O objetivo principal desta dissertação era propor um sistema de tempo real aplicado à detecção de vazamentos implementado em DSP, que possibilitasse não só a detecção do vazamento, mas também a localização do mesmo.

Como objetivo secundário, foi validado o modelo de DSP escolhido. Conforme exposto no Capítulo 2, atualmente há inúmeros modelos de DSPs disponíveis no mercado, o que acaba dificultando na hora de escolher o mais adequado à aplicação. Neste trabalho foi feito o inverso: foi especificado um modelo de DSP e a aplicação foi projetada para o modelo escolhido.

Como vantagem deste DSP pode-se citar o suporte técnico via *e-mail*, que foi um recurso bastante utilizado durante o desenvolvimento deste trabalho. Como desvantagem, pode-se citar a documentação que apesar de abrangente, não apresenta uma sequência lógica e muitas vezes é confusa (a linguagem em que é escrita não facilita a compreensão do texto).

Como não era objetivo deste trabalho avaliar as diversas metodologias que podem ser empregadas na detecção de vazamentos, apenas uma foi utilizada para validar a sua implementação em DSP.

5.1 Trabalhos futuros

A abordagem *foreground/background* utilizada pelos DSPs da família C2000 da Texas apresenta algumas limitações. Quando o código de uma ISR está sendo executado pela CPU automaticamente as demais interrupções são desabilitadas (Texas Instruments, 2008*c*). No caso deste projeto temos a interrupção TINT**0** como sendo de mais alta prio-

ridade que a SCIRXINTA (ver Tabela 8). Mesmo assim, caso a CPU esteja executando o código da SCIRXINTA, se neste momento ocorrer uma interrupção TINT**0**, a mesma não será servida. A recíproca também é verdadeira.

Para superar essa limitação pode ser utilizado o DSP/BIOS, que é um *kernel* preemptivo desenvolvido pela Texas Instruments e disponibilizado para utilização em conjunto com o Code Composer Studio.

Outra melhoria que pode ser empregada é a utilização da interface CAN (disponível na placa de desenvolvimento do DSP), ao invés da SCI, para comunicação com o supervisório. Os conversores RS232/RS485 e RS485/Ethernet seriam substituídos por um conversor CAN/Ethernet. O protocolo CAN suporta taxas de até 1Mbps e distâncias de até 1km (BOSCH, 1991).

Além disso, quando o intuito for apenas adquirir o sinal de pressão, como foi o caso deste trabalho ao testar a aquisição utilizando a infraestrutura do LAMP, pode ser utilizado um cartão de memória do tipo SD (Secure Digital). O tempo de escrita utilizando este tipo de cartão é bem inferior ao empregado pela interface SCI (Texas Instruments, 2007*a*), possibilitando realizar testes com uma taxa de aquisição mais elevada.

Uma vez que o sistema esteja totalmente implementado em DSP, incluindo todas as funções pertinentes à comunicação com dispositivos (GPS e supervisório) e ao processamento, pode-se avaliar os trechos em que o processador fica ocioso. Nesses trechos é preferível colocar o processador em modo *sleep*, contribuindo desta maneira para um menor consumo de energia.

Referências

Agência Nacional do Petróleo, Gás Natural e Biocombustíveis (2009), *Anuário Estatístico Brasileiro do Petróleo, Gás Natural e Biocombustíveis*.

AVELINO, Á. M.; R. E. F. DA SILVA; G. J. M. DE ARAUJO; J. Á. DE PAIVA; F. DE O. QUINTAES; A. L. MAITELLI; A. D. D. NETO; A. O. SALAZAR (2009), Real time leak detection system applied to oil pipelines using sonic technology and neural networks, *em* 'The 35th Annual Conference of the IEEE Industrial Electronics Society'.

AZEVEDO, F. M. DE (2009), Proposta de algoritmo para detecção de vazamentos em oleodutos utilizando análise frequencial de sinais de pressão em tubulações de petróleo, Dissertação de mestrado, Programa de Pós-Graduação em Engenharia Elétrica, UFRN, Natal, RN.

BARR, M. (1999), *Programming Embedded Systems in C and C++*, O'Reilly.

BEGA, E. A.; G. J. DELMÉE; P. E. COHN; R. BULGARELLI; R. KOCH; V. S. FINKEL (2006), *Instrumentação Industrial*, 1ª edição, Editora Interciência.

BOSCH (1991), *CAN Specification 2.0*.

BOYLESTAD, R. L.; L. NASHELSKY (2004), *Dispositivos eletrônicos e teoria dos circuitos*, Prentice Hall.

Central Intelligence Agency (2006), *The World Factbook*.

Druck (2003), *PTX 7800 Series Pressure Transmitters*.

GANSSLE, J. (2004), *The Firmware Handbook*, Elsevier.

GlobalSat Technology Corporation (2008), *GPS Engine Board ET-332 User Guide*.

KUO, S. M. (2006), *Real Time Digital Signal Processing Implementations and Applications*, 2ª edição, Wiley.

LABROSSE, J. J. (2002), *MicroC/os-II The Real Time Kernel*, 2ª edição, CMPBooks.

MOURA, C. H. W.; R. M. BAPTISTA; J. T. M. CAVALCANTE (2002), Leakage detection on multiphase flow pipelines, *em* 'International ISA Conference'.

OSHANA, R. (2006), *DSP Software Development Techniques for Embedded and Real-Time Systems*, Elsevier.

PAIVA, J. Á. DE; Á. M. AVELINO; R. E. F. DA SILVA; G. J. M. DE ARAUJO; A. L. MAITELLI; A. O. SALAZAR; A. D. D. NETO (2009), Implementação em DSP de um sistema de detecção de vazamentos utilizando técnicas sônicas e redes neurais artificiais, *em* 'V Congresso Rio Automação'.

PARK, J.; S. MACKAY (1997), *Practical Data Acquisition for Instrumentation and Control Systems*, 1ª edição, Newnes.

RENNÓ, M. (2007), Análise da malha regional de dutos e a expansão dos trechos brasileiros: gasodutos, alcoodutos e oleodutos, *em* '4th Energy Integration Congress'.

SILVA, R. E. F. DA (2009), Implementação de um módulo de supervisão para um sistema de detecção de vazamentos em dutos de petróleo, Dissertação de mestrado, Programa de Pós-Graduação em Ciência e Engenharia de Petróleo, UFRN, Natal, RN.

SILVA, R. E. F. DA; Á. M. AVELINO; G. J. M. DE ARAUJO; J. Á. DE PAIVA; A. L. MAITELLI; A. D. D. NETO; A. O. SALAZAR (2009), Supervisão em tempo real de um sistema de detecção de vazamentos que utiliza a tecnologia sônica, *em* '5º Congresso Brasileiro de P&D em Petróleo e Gás'.

SiRF Technology, Inc (2008), *NMEA Reference Manual*.

Spectrum Digital Inc (2008), *C2000 Series XDS510LC JTAG Emulator*.

Texas Instruments (2006), *Code Composer Studio Development Tools v3.3 Getting Started Guide*.

Texas Instruments (2007*a*), *Interfacing SD/MMC Cards With TMS320F28xxx DSCs*.

Texas Instruments (2007*b*), *TMS320C28x Optimizing C/C++ Compiler v5.0.0 User's Guide*.

Texas Instruments (2007*c*), *TMS320x2833x Analog-to-Digital Converter (ADC) Module*.

Texas Instruments (2008*a*), *C2833x/C2823x C/C++ Header Files and Peripheral Examples*.

Texas Instruments (2008*b*), *TMS320C28x Floating Point Unit and Instruction Set Reference Guide*.

Texas Instruments (2008*c*), *Using DSP/BIOS in C2800 Applications with High Interrupt Rates*.

Texas Instruments (2009*a*), *Embedded Processing Guide*.

Texas Instruments (2009*b*), *TMS320x2833x, 2823x System Control and Interrupts Reference Guide*.

VALENZUELA, A. (2005), 'Interrupts', Acessado em 10/10/2009. `http://www.cnx.org/content/m12321/1.2/`.

Vocabulario Internacional de Metrologia (2008).

Wikipedia - Embedded Systems (2009), Acessado em 10/11/2009. `http://www.en.wikipedia.org/wiki/Embedded_system`.

ZHOU, X.; P. PETROV (2006), Rapid and low-cost context-switch through embedded processor customization for real-time and control applications, *em* '43rd ACM/IEEE Design Automation Conference'.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo