UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO

DESENVOLVIMENTO DE UM SISTEMA DE MEDIÇÃO DE VIBRAÇÕES PARA UMA TORRE DE UM AEROGERADOR

TRABALHO DE CONCLUSÃO DE CURSO

Matheus Felipe Schöne

Santa Maria, RS, Brasil

2016

DESENVOLVIMENTO DE UM SISTEMA DE MEDIÇÃO DE VIBRAÇÕES PARA UMA TORRE DE UM AEROGERADOR

Matheus Felipe Schöne

Monografia apresentada ao Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Engenharia de Controle e Automação.

Orientador: Prof. Dr. Claiton Moro Franchi

Santa Maria, RS, Brasil

2016

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Engenharia de Controle e Automação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Conclusão de Curso

DESENVOLVIMENTO DE UM SISTEMA DE MEDIÇÃO DE VIBRAÇÕES PARA UMA TORRE DE UM AEROGERADOR

elaborado por
Matheus Felipe Schöne

como requisito parcial para obtenção do grau de
Bacharel em Engenharia de Controle e Automação

COMISSÃO EXAMINADORA:

_____
Claiton Moro Franchi, Dr.
(Presidente/Orientador)
(UFSM)

_____
Humberto Pinheiro, Ph.Dr.
(UFSM)

_____
Carlos Eduardo Souza, Dr.
(UFSM)

Santa Maria, 11 de fevereiro de 2016.

RESUMO

Trabalho de Conclusão de Curso
Curso de Engenharia de Controle e Automação
Universidade Federal de Santa Maria

DESENVOLVIMENTO DE UM SISTEMA DE MEDIÇÃO DE VIBRAÇÕES PARA UMA TORRE DE UM AEROGERADOR

AUTOR: MATHEUS FELIPE SCHÖNE
ORIENTADOR: CLAITON MORO FRANCHI
Data e Local da Defesa: Santa Maria, 11 de fevereiro de 2016.

O presente trabalho tem como objetivo desenvolver um sistema de medição de vibrações, isolado da rede de energia elétrica, disponibilizando em um cartão de memória micro SD as medições realizadas. O sistema é composto de um microcontrolador *concerto*, uma placa de interface, dois painéis fotovoltaicos, duas baterias, um controlador de carga e quatro acelerômetros ICP®. O microcontrolador é o cérebro do sistema de aquisição, responsável pelo processamento das informações e gravação na memória externa. A placa de interface foi desenvolvida para interligar o microcontrolador aos acelerômetros e ao controlador de carga. Os painéis fotovoltaicos, as baterias e o controlador de carga são o sistema de geração e armazenamento de energia, fornecendo energia para o funcionamento do sistema. Com a intenção de validar o sistema desenvolvido, foram realizados testes utilizando vibrações com intensidade e frequência conhecidas.

Palavras-chave: Sistema de medição, Vibrações, Energias renováveis, *Concerto.*

ABSTRACT

Bachelor Final Project
Bachelor of Control and Automation Engineering
Federal University of Santa Maria

DEVELOPMENT OF A VIBRATION MEASUREMENT SYSTEM FOR A WIND TURBINE TOWER

AUTHOR: MATHEUS FELIPE SCHÖNE
ADVISER: CLAITON MORO FRANCHI
Defense Place and Date: Santa Maria, February 11st 2016.

This work aims to development a vibration measurement system providing the measurements performed on a microSD memory card. This system is isolated from the electricity network. The system consists of a *concerto* microcontroller, an interface printed circuit board (PCB), two photovoltaic panels, two batteries, a charge controller and four ICP® accelerometers. The microcontroller is the brain of the acquisition system, responsible for processing the information and write to the external memory. The interface PCB is designed to connect the microcontroller to the accelerometers and the charge controller. The photovoltaic panels, the battery and the load controller are the generation and storage system, supplying energy for the system. In order to validate the developed system, tests were performed using vibrations of known frequency and intensity.

Key words: Measurement system, Vibrations, Renewable energy, *Concerto*.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Com aumento do consumo de energia elétrica nos últimos anos, a necessidade de inclusão de outras fontes de energia na matriz energética é muito importante. Nesse cenário, a energia eólica aparece como uma das fontes renováveis mais interessantes em termos de segurança de fornecimento, produção e sustentabilidade ambiental (GWEC, 2008).

O Brasil possui um grande potencial para geração de energia eólica, com destaque para as regiões litorâneas Nordeste e Sul. Com isso, nos últimos anos existiram grandes investimentos nessa área. Nesse sentido, justifica-se o investimento em pesquisa e desenvolvimento em energia eólica.

Em um sistema eólico, a torre é o suporte estrutural do aerogerador e as cargas alternadas suportadas por ela são complexas. Nos últimos anos, inúmeros acidentes causados por falhas nas torres de aerogeradores foram registrados. Para garantir que a torre é adequada para o aerogerador, é necessária a realização de uma análise da torre. Através da análise modal, é possível encontrar a frequência natural da torre, que deve ser diferente da frequência nominal de funcionamento do aerogerador, a fim de evitar o fenômeno de ressonância (CHEN; JIANG, 2010).

Logo, o objetivo deste trabalho é desenvolver um sistema de medição de vibração isolado para uma torre de um aerogerador. Esse sistema servirá de instrumento para a obtenção das frequências naturais e das formas modais da torre a partir do espectro de frequência obtido pela utilização da Transformada Rápida de Fourier (FFT).

Este trabalho está organizado da seguinte maneira: na primeira seção é realizada uma breve revisão bibliográfica sobre vibrações e os componentes de um sistema solar fotovoltaico; na segunda seção são descritos os procedimentos adotados durante o desenvolvimento do trabalho, além disso, são apresentados os materiais que compõem o trabalho; a terceira seção apresenta o protótipo construído e a validação do sistema. Por fim, conclui-se o trabalho analisando os resultados obtidos, além da apresentação de algumas sugestões para a continuidade do projeto.

2 REFERÊNCIAL TEÓRICO

Nesta seção são apresentados os principais conceitos envolvidos neste trabalho. Inicialmente, uma introdução sobre o sistema a ser medido, após um conteúdo sobre vibrações, bem como o sensor utilizado para medi-las. Além disso, são apresentados os componentes presentes em um sistema solar fotovoltaico.

2.1 Sistema torre-aerogerador

O conjunto torre e aerogerador compõem um sistema mecânico sujeito a vibrações devido à ação do vento. Caso essas vibrações aproximem-se das frequências naturais da torre, existe a possibilidade de ressonância. Sendo assim, os materiais do conjunto podem sofrer fadiga devido à vibração excessiva (PAULA, 2012).

Para monitorar e evitar esse desgaste, é possível medir e mapear as vibrações que ocorrem no sistema através da análise modal.

2.2 Vibrações

Qualquer movimento oscilatório, que se repita após um período de tempo, é denominado como vibração. Normalmente, um sistema vibratório é composto de três partes: um meio elástico, responsável pelo armazenamento de energia potencial; um meio inercial, o qual armazena energia cinética; e um meio amortecedor, com a finalidade de dissipar energia (RAO, 2009).

2.2.1 Análise modal experimental

Análise modal experimental é o processo de determinação das características dinâmicas inerentes de um sistema, sendo uma aplicação dos conceitos de vibrações. Consiste em extrair, através da função resposta em frequência, os parâmetros modais do sistema, sendo eles frequência natural, fator de amortecimento e forma modal (SILVA, 2009).

Através da utilização dos parâmetros modais, é possível resolver problemas de ressonância devido à interação com a frequência natural do sistema. Além disso, verificar a forma de vibração da estrutura, chamada de modo de vibração (BOLINA, 2014). A Figura 1 apresenta o primeiro e o terceiro modo de vibração de uma viga engastada.

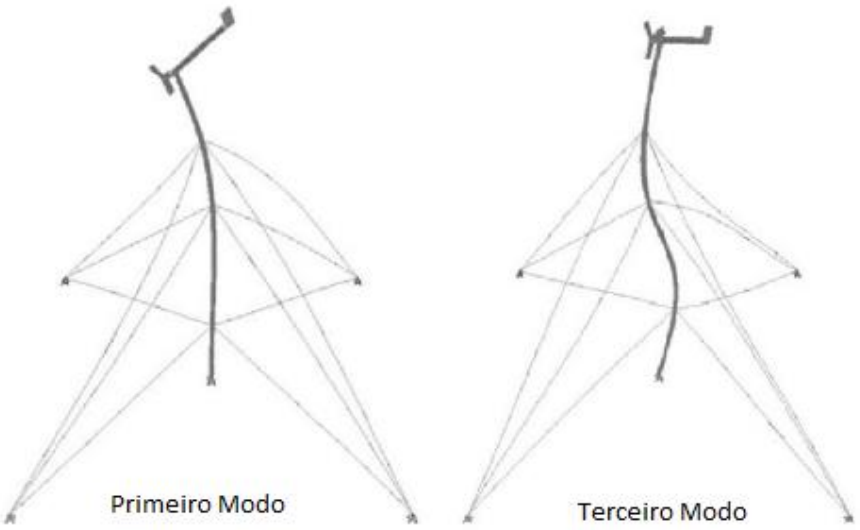

Figura 1 - 1º e 3º modos de vibração de uma viga engastada. [SILVA, 2009]

2.2.2 Acelerômetros

Acelerômetros são os dispositivos utilizados para a medição de vibrações. Os acelerômetros piezoelétricos contêm uma massa fixada sobre um material piezoelétrico, de quartzo ou cristais cerâmicos, que converte a aceleração aplicada em um sinal elétrico proporcional (PCB, 2016). Estes dispositivos, porém, podem ser utilizados somente para medir vibração senoidal, transitória ou randômica. Existem sensores com captação de sinal uniaxial e triaxial, dessa forma, a medição em vários eixos é realizada utilizando dois ou mais acelerômetros uniaxiais em um mesmo ponto ou um único sensor triaxial.

Outra restrição desses sensores, é a captação do sinal ser realizada em apenas um eixo, assim para medições em mais eixos, são necessários mais acelerômetros em um mesmo ponto de medição.

O efeito piezoelétrico produz uma acumulação oposta de partículas carregadas sobre o cristal. Essa carga gerada é proporcional à força aplicada. As partículas carregadas são coletadas por eletrodos e transmitidas por fios para um condicionador eletrônico externo, que produz um sinal de tensão proporcional à carga (NSK, 2016).

O principio de funcionamento de um acelerômetro piezoelétrico é apresentado na Figura 2.

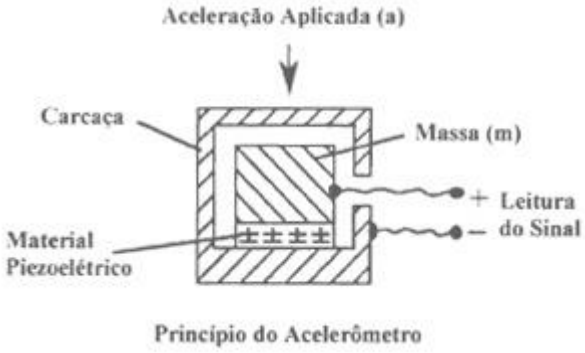

Figura 2 - Principio de Funcionamento do Acelerômetro. [NSK, 2016]

Os cabos que conectam o sensor e o condicionador necessitam de uma baixa capacitância para que não ocorra perda de carga, além disso, elevadas vibrações nos cabos podem causar elevado ruído no sinal (NSK, 2016).

Para resolver essas limitações, acelerômetros ICP® (*Integrated Circuit Piezoeletric*), apresentado na Figura 3, trazem internamente embarcados no sensor, microamplificadores eletrônicos que condicionam os sinais de carga para sinais de tensão. Com isso, evita-se o uso de condicionadores externos e cabos especiais (PCB, 2015).

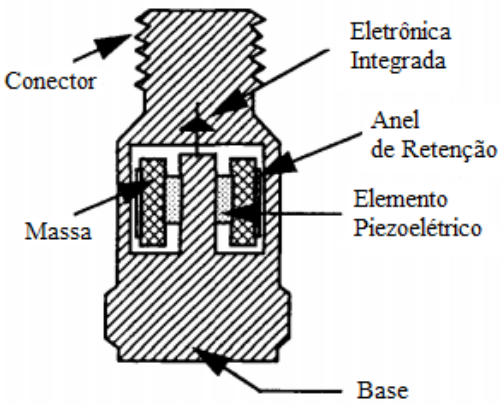

Figura 3 - Típico Acelerômetro ICP® Industrial. [PCB, 2015]

A saída do sensor é um sinal de corrente alternada (CA) proporcional à vibração recebida pelo sensor. O sinal CA é transmitido sobre uma tensão de polarização de corrente contínua (CC), permitindo-o oscilar entre a tensão de alimentação (Vcc) e a referência 0 V (SKF, 1999). A Figura 4 apresenta o comportamento do sinal CA sobre a tensão de polarização CC.

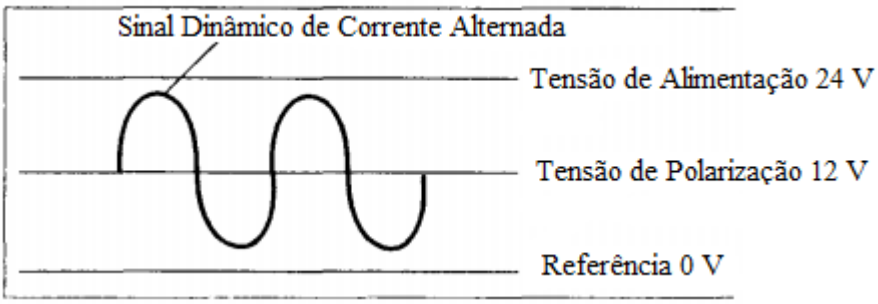

Figura 4 - Sinal de Saída CA sobre o Sinal de Polarização CC.

2.3 Componentes de um sistema de geração de energia fotovoltaica

2.3.1 Painéis solares fotovoltaicos

Em um sistema de geração de energia fotovoltaica, o painel solar é responsável por transformar a energia oriunda da radiação solar em energia elétrica. Essa transformação ocorre devido ao efeito fotovoltaico, que consiste na propriedade apresentada por alguns materiais de fornecer uma diferença de potencial graças à ação de raios luminosos.

Cada painel é formado por um arranjo de células fotovoltaicas conectadas. O arranjo de células é necessário, pois uma única célula apresenta uma tensão muito baixa, na ordem de 0,5 V. Assim, a tensão do painel é determinada pela quantidade de células utilizadas (IMHOFF, 2007).

Dentre as diversas tecnologias de células fotovoltaicas disponíveis, as células fabricadas utilizando silício cristalino são as mais utilizadas.

2.3.2 Controlador de carga

Controladores de carga são dispositivos eletrônicos que tem como objetivo principal controlar o fluxo de potência entre os componentes do sistema solar fotovoltaico. Esses dispositivos são incluídos na maioria dos sistemas com a função de proteger a bateria em condições extremas de carga e descarga (GUIMARÃES, 2004).

O princípio de funcionamento desses dispositivos é o seguinte: quando a tensão da bateria atingir carga plena, o painel fotovoltaico é desconectado; de maneira análoga, quando

a tensão da bateria atinge o estado crítico definido pela fabricante, o fornecimento de energia à carga é interrompido.

2.3.3 Baterias

Um sistema de armazenamento de energia é imprescindível para sistemas fotovoltaicos sem conexão com a rede elétrica. Para esses sistemas, são utilizadas baterias com a finalidade de armazenar o excedente produzido e não consumido durante os períodos de maior insolação. Dessa forma, em períodos em que a geração de energia é nula ou insuficiente para manter o sistema, a carga armazenada garante um fornecimento ininterrupto de energia elétrica (GUIMARÃES, 2004).

As baterias utilizadas em sistemas isolados devem apresentar como principais características a alta eficiência, baixa manutenção e alta taxa de confiabilidade. Devido a isso, as baterias chumbo-ácida são as mais utilizadas pelo mercado (IMHOFF, 2007).

3 MATERIAIS E MÉTODOS

Nesta seção são apresentadas a arquitetura do sistema, as ferramentas e os métodos utilizados para o desenvolvimento do sistema de medição isolado.

3.1 Arquitetura do sistema

Considerando que o sistema de medição foi construído para a utilização em uma torre de um aerogerador, o mesmo funciona de maneira independente ao funcionamento do aerogerador. Desta forma, o sistema é composto de dois segmentos. Um responsável pela geração e armazenamento de energia para o funcionamento e outro associado à aquisição, condicionamento e tratamento das medições de vibrações realizadas.

A Figura 5 apresenta a arquitetura do sistema de medição, onde são mostrados todos os componentes do sistema.

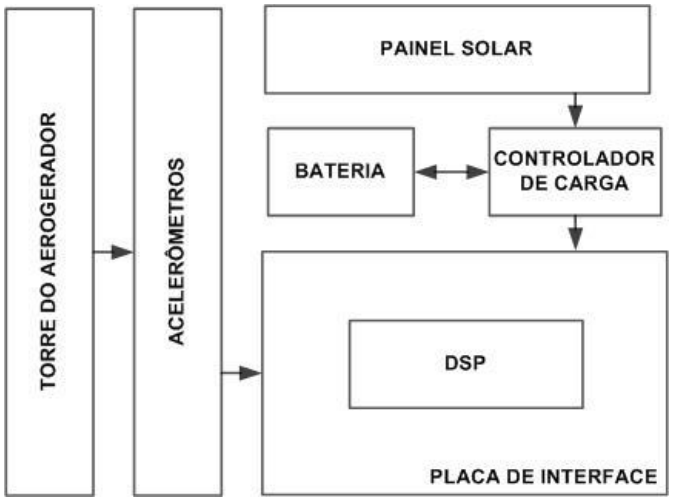

Figura 5 - Arquitetura do sistema de medição.

Conforme apresentado pela Figura 5, o sistema é composto pelos seguintes elementos:

- Torre do Aerogerador: sistema estrutural que pretende-se analisar.
- Acelerômetros ICP®: realizam as medições de vibrações.
- Placa de interface: responsável pelo condicionamento dos sinais e alimentação do microcontrolador.

- Microcontrolador *Concerto* F28M36x: encarregado pelo processamento de dados e armazenamento em memória externa.
- Sistema de geração e armazenamento de energia:
  - Painel solar: gera energia elétrica.
  - Controlador de carga: responsável pelo carregamento da bateria.
  - Bateria: armazena energia para o funcionamento do sistema durante períodos sem incidência solar.

Uma representação de todo o sistema de medição acoplado à torre do aerogerador, demonstrando o posicionamento do sistema na torre é mostrada na Figura 6.

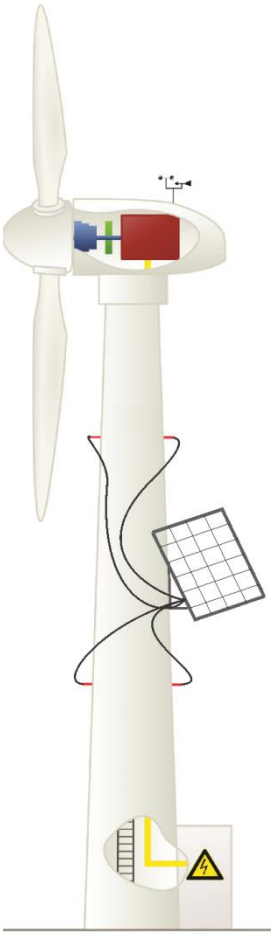

Figura 6 - Esquema do sistema montado na torre do aerogerador.

3.2 Acelerômetro ICP®

Os sensores ICP® necessitam de uma fonte de alimentação de corrente constante para garantir o funcionamento adequado. Esses sensores requerem um circuito de excitação, mostrado pela Figura 7, que consiste de uma fonte de tensão regulada de 18 a 30 Vcc, um diodo de corrente constante (DCC) de 2 a 10 mA e um capacitor de desacoplamento de 10 a 30 µF responsável por retirar o nível CC do sinal (IMI SENSORS, 2016).

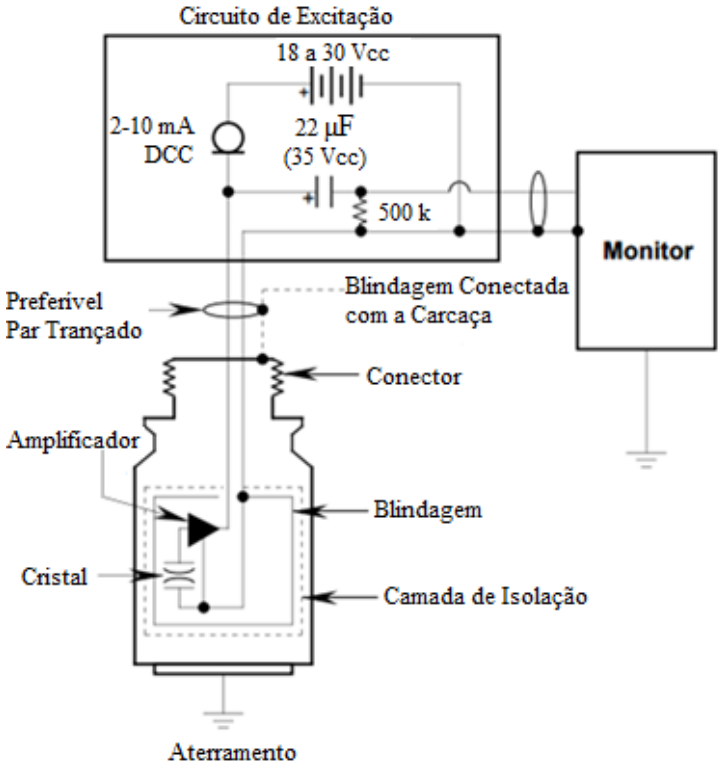

Figura 7 - Circuito de excitação. [IMI SENSORS, 2016]

Para a realização deste trabalho foram utilizados quatro acelerômetros ICP® industriais de baixo custo modelo 603C01 com as seguintes especificações apresentadas na Tabela 1. Para a obtenção das formas modais de uma estrutura, são necessários, no mínimo, dois sensores acoplados a estrutura. Assim, foram utilizados dois pontos de medição com dois acelerômetros em cada ponto, a fim de obter medidas em dois eixos para cada ponto.

Tabela 1 - Especificações dos Acelerômetros 603C01.

Parâmetro	Valor
Sensitividade (± 10 %)	100 mV/g
Faixa de Medição	± 50 g
Faixa de Frequência (± 3 dB)	0,5 a 10000 Hz
Tensão de Excitação	18 a 28 Vcc
Corrente Constante de Excitação	2 a 20 mA
Tensão de Polarização de Saída	8 a 12 Vcc

3.3 Placa de interface

Conforme apresentado na seção 2.2, os acelerômetros necessitam de um circuito de alimentação com corrente constante, além disso, é necessário um circuito de tratamento do sinal emitido pelo acelerômetro. Esse circuito de tratamento é responsável pela transformação do sinal recebido para os níveis de tensão utilizados pelo microcontrolador.

Outro requisito é a alimentação do microcontrolador e dos demais componentes utilizados no circuito de tratamento de sinal. Assim, são utilizados quatro circuitos de potência responsáveis pela síntese das tensões necessárias.

3.3.1 Circuito de condicionamento de sinal

A partir dos dados apresentados na Tabela 1, a tensão de excitação dos acelerômetros foi definida para 24 V por permitir a utilização de um conjunto de baterias para alimentar o sistema. Já a corrente de excitação foi definida para 4,7 mA devido a escolha do diodo de corrente constante 1N5314. A conexão dos acelerômetros com a placa de interface foi realizada utilizando conectores do tipo BNC.

O circuito de excitação do acelerômetro, juntamente com o capacitor de desacoplamento e o resistor de filtro foram colocados antes da entrada do circuito de tratamento de sinal. Isso porque, após o resistor de filtro, existe apenas o sinal CA proporcional à vibração medida. Além disso, para evitar ruído de alta frequência foram utilizados *beads* de ferrite na entrada do sinal.

Conforme mostrado na Tabela 1, os acelerômetros apresentam uma sensitividade de 100 mV/g. Assim, para melhorar a resolução em pequenas acelerações, um ganho de duas vezes foi definido.

A implementação do ganho, assim como o *offset* necessário para retirar a parte negativa do sinal CA foi obtido através da utilização do amplificador de instrumentação AD620. A Figura 8 apresenta o diagrama de conexão do dispositivo.

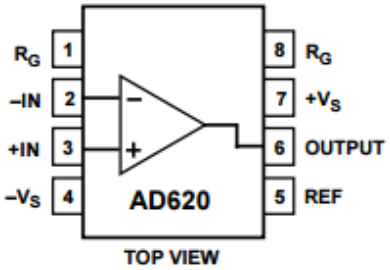

Figura 8 - Diagrama de conexão AD620.

O ganho deste amplificador é definido por qualquer impedância entre os pinos 1 e 8 e é calculado pela equação abaixo, onde G é o ganho desejado:

$$R_g = \frac{49{,}4\ k\Omega}{G - 1} \quad (1)$$

Dessa forma, para o ganho de duas vezes definido anteriormente, é necessária uma impedância de 49,4 kΩ.

Já o pino 5 deste amplificador determina a tensão zero de saída, o que permite injetar um *offset* diretamente no sinal de saída. Devido ao conversor analógica-digital do microcontrolador trabalhar na faixa de 0 a 3,3 V, foi adicionado um *offset* de 1,5 V ao sinal amplificado. Antes de conectar o sinal de 1,5 V ao pino 5 do amplificador diferencial, um estágio seguidor de tensão foi utilizado para realizar casamento de impedâncias.

Da mesma forma, também foi empregado um seguidor de tensão na saída do amplificador diferencial, contudo, com outra finalidade. Esse estágio, serve como proteção para o microcontrolador, limitando o sinal em 3 V devido ao amplificador operacional ser alimentado com 0 e 3 V. Os amplificadores operacionais utilizados nos seguidores de tensão estão encapsulados no circuito integrado LM6132.

A Figura 9 exibe o circuito de condicionamento descrito.

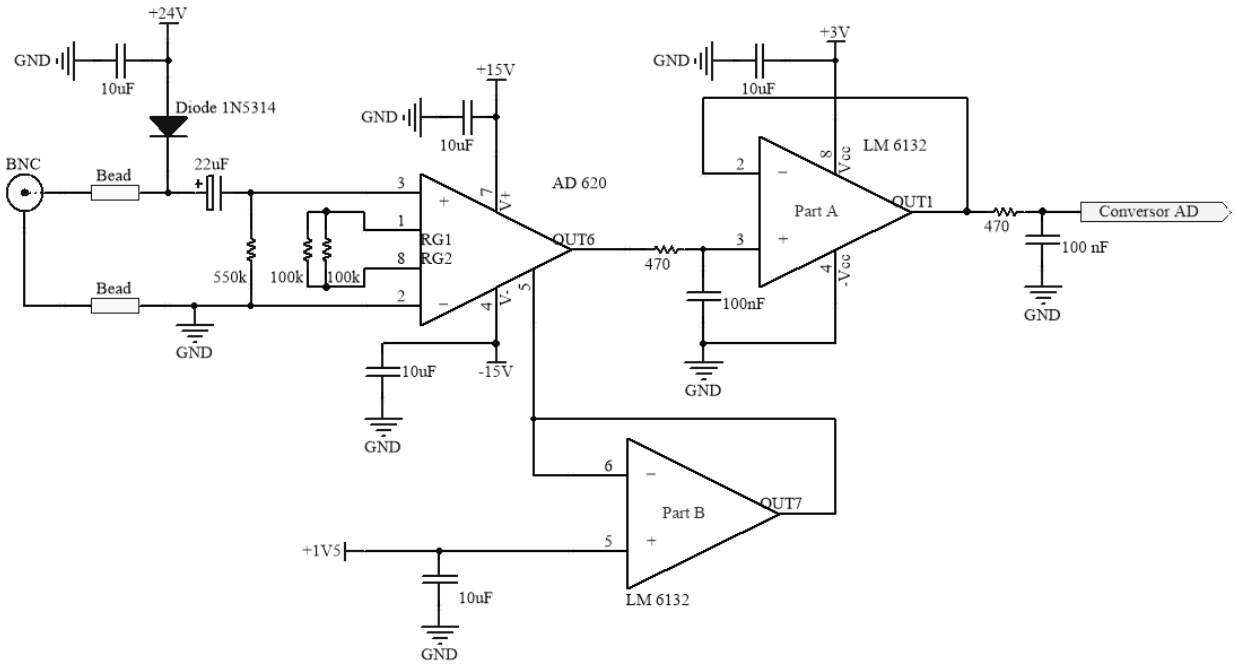

Figura 9 - Circuito de condicionamento.

Uma simulação do funcionamento do circuito de condicionamento é apresentada pela Figura 10. Através da imagem, é possível verificar o estágio de ganho e adição de *offset*, além do estágio de proteção.

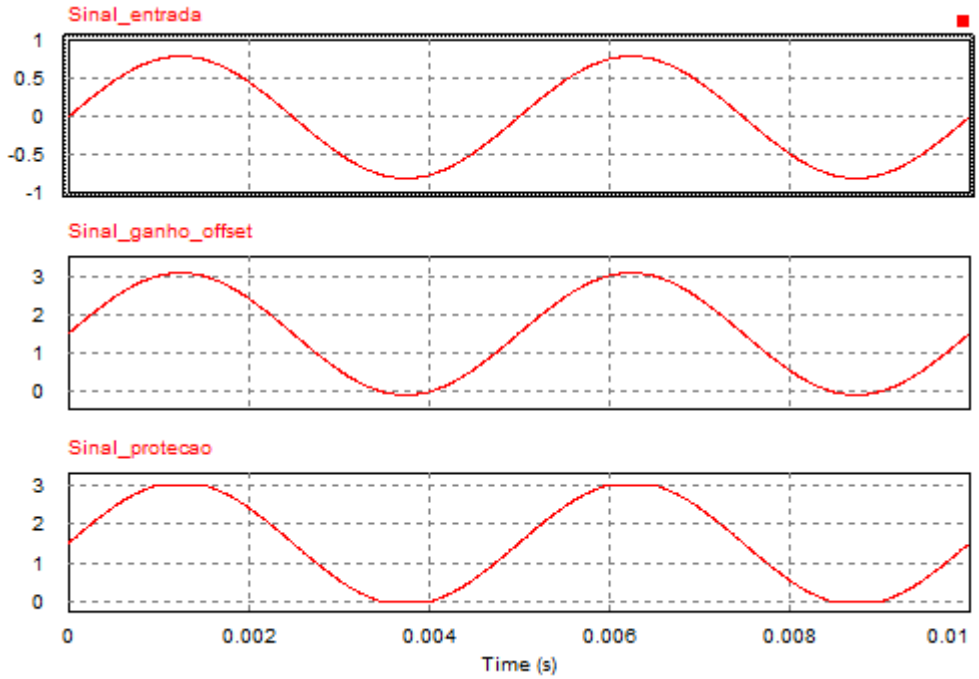

Figura 10 - Sinal em cada estágio do circuito de condicionamento.

3.3.2 Circuitos de potência

A partir do circuito de condicionamento apresentado anteriormente, foram definidos os circuitos de potência necessários para a placa. Assim, ficou estabelecida a necessidade de cinco diferentes níveis de tensão, além do sinal de alimentação de 24 V.

Em virtude do amplificador diferencial AD620 receber valores positivos e negativos no sinal de entrada, é necessário alimentá-lo com tensão positiva e negativa. Logo, as tensões de alimentação foram definidas como 15 V e – 15 V.

Os demais níveis de tensão são utilizados para alimentação do circuito integrado LM6132, alimentação do microcontrolador e sinal de *offset*. Devido a função de proteção das entradas analógicas do microcontrolador, o circuito integrado LM6132 é alimentado com 3 V. Já o microcontrolador utiliza 5 V para sua alimentação. O sinal de *offset*, conforme apresentado anteriormente, é de 1,5 V.

Para a síntese das tensões de 15 V, -15 V e 5 V foram utilizados três conversores CC-CC não isolados. As tensões de 15 V e -15 V foram disponibilizadas através de dois modelos do conversor PTN78000. Já para a tensão de 5 V foi utilizado o conversor PTH08080.

A síntese da tensão de 15 V foi realizada utilizando o modelo PTN78000H, apresentado na Figura 11. Esse conversor possibilita a escolha da tensão de saída e apresenta como principais características:

- Corrente de saída: 1,5 A;
- Tensão de entrada: 15 - 36 V;
- Tensão de saída ajustável: 11,85 – 22 V;
- Alta eficiência: > 95 %.

Figura 11 - Conversor CC-CC PTN78000H.

No caso da tensão de -15 V foi aplicado o modelo PTN78000A (Figura 12), o qual apresenta as seguintes características:

- Corrente de saída: 1,5 A;
- Tensão de entrada: 7 - 29 V;

- Tensão de saída ajustável: -15 V a -3 V;
- Eficiência: > 84 %.

Figura 12 - Conversor CC-CC PTN78000A.

Empregando o conversor PTH08080W (Figura 13), a tensão de 5 V foi obtida. As características desse conversor são mostradas a seguir:

- Corrente de saída: 2,25 A;
- Tensão de entrada: 4,5 - 18 V;
- Tensão de saída ajustável: 0,9 - 5,5 V;
- Alta eficiência: > 93 %.

Figura 13 - Conversor CC-CC PTH08080W.

Os conversores PTN78000 e seus respectivos circuitos responsáveis pela síntese das tensões são demonstrados na Figura 14. Já a Figura 15 apresenta os circuitos do conversor PTH08080W, do regulador de tensão LP2950-3 responsável pela tensão de 3 V e do divisor resistivo de 1,5 V.

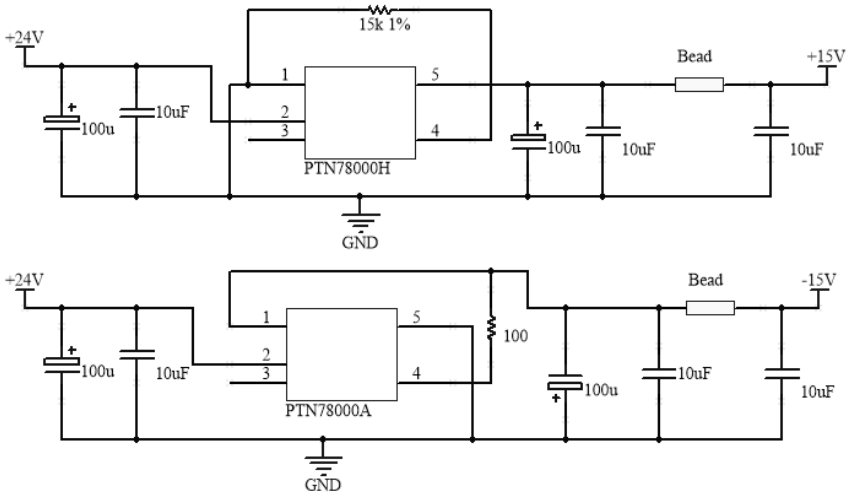

Figura 14 - Circuitos dos conversores PTN78000.

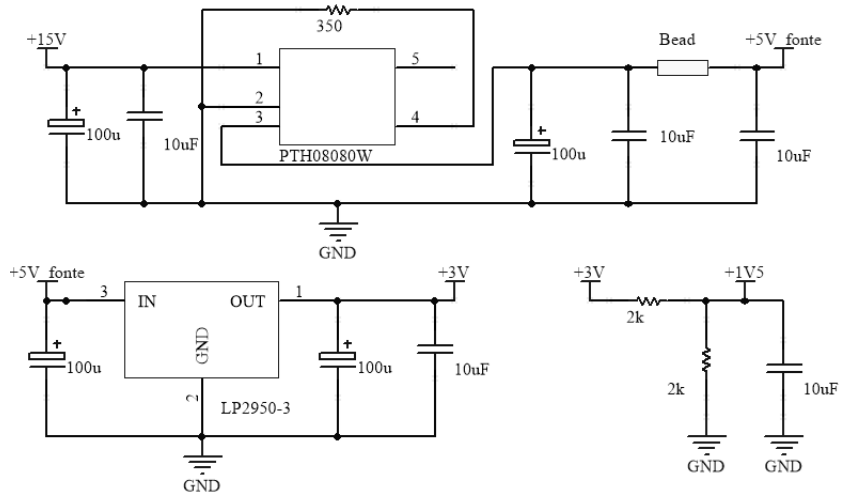

Figura 15 - Circuitos do conversor PTH08080W, LP2950-3 e divisor resistivo.

3.4 Microcontrolador *Concerto* F28M36x

O microcontrolador é a peça-chave do sistema de medição. Ele é responsável pela aquisição dos sinais provenientes dos acelerômetros, pelos cálculos matemáticos e também realiza a gravação dos dados processados em uma memória externa.

O *concerto* (Figura 16) é um microcontrolador com dois processadores encapsulados em um mesmo *chip*, com subsistemas de comunicação e controle independentes. O núcleo dedicado ao sistema de comunicação é baseado na CPU 32-bit ARM Cortex-M3 que oferece inúmeros periféricos de comunicação. Já o núcleo de controle é baseado na CPU de ponto flutuante de 32-bit C28x TMSF28335 que fornece os periféricos de controle, como *Enhanced*

Pulse-Width Modulator (ePWM) com proteção de falhas. Este microcontrolador apresenta como principais características:

- C28x TMSF28335 @ 150 MHz;
- ARM Cortex-M3 @ 125 MHz;
- Memória flash de 1,5MB;
- Memória RAM de 232KB;
- Conversor Analógico-Digital de 12 bits;
- Sistema tempo real;
- Porta *Ethernet*;
- Conector MicroSD;

Figura 16 - *Concerto* F28M36x.

Devido à existência de dois processadores no microprocessador, é necessário realizar a comunicação entre eles. Para isso, existe um componente denominado *Inter-Processor Comunication* (IPC) que contém diversos módulos de comunicação.

3.4.1 Módulo *SharedMemory*

O módulo *SharedMemory* foi desenvolvido para ser utilizado em um ambiente multi-processador onde existem regiões de memória que são compartilhadas e acessadas por diferentes processadores (Texas Instruments, 2014).

Esta região pode ser acessada por ambos processadores, contudo, cada bloco de memória pertence a um dos processadores, baseado na configuração realizada. Dessa forma, quando um bloco da memória pertence ao M3, por exemplo, esse tem acesso total ao bloco, enquanto o C28x tem acesso apenas à leitura daquele bloco.

O módulo *SharedMemory* foi utilizado neste trabalho por conter oito blocos de memória de 8KB cada, totalizando 64 KB para comunicação entre os processadores. A Figura 17 apresenta uma representação do módulo, além dos blocos e seus respectivos endereços em cada processador.

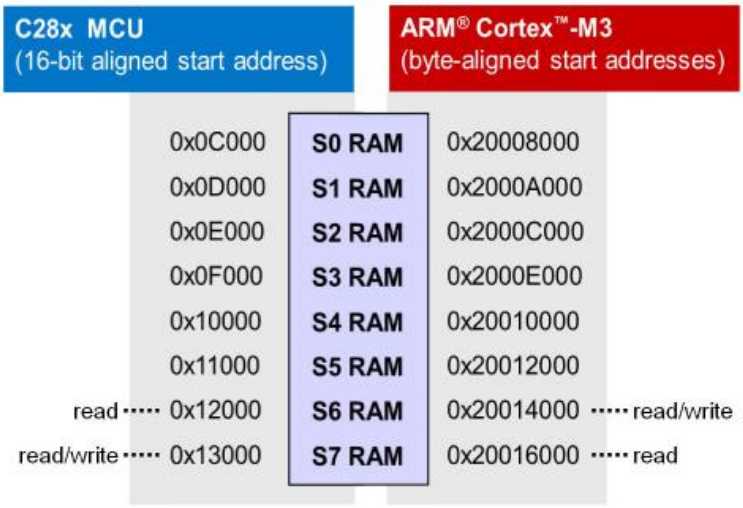

Figura 17 - Módulo SharedMemory.

3.4.2 Programação do C28x

Conforme apresentado anteriormente, o C28x é responsável pelas aplicações de controle. Como a aquisição de dados é um processo crítico para o projeto, esse processador foi designado a essa tarefa.

Para realizar a leitura de sinais através do conversor analógico-digital (A/D), é necessário configurar os módulos dos A/Ds e também do modulador de largura de pulso (PWM) responsável pela frequência de disparo da interrupção de conversão A/D.

A frequência do módulo PWM depende do modo de contagem e do período do *clock* de base de tempo. Assim, utilizando o modo *Up and Down Count*, apresentado na Figura 18,

a frequência do ePWM é calculada pela equação, onde F_{PWM} é a frequência do ePWM, TBPRD é o registrador de contagem e T_{TBCLK} é o período do *clock* de base de tempo:

$$F_{PWM} = \frac{1}{2 * TBPRD * T_{TBCLK}} \tag{2}$$

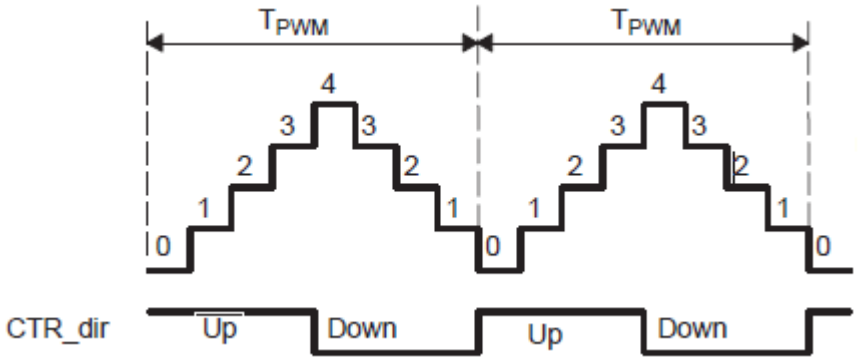

Figura 18 - Modo Up and Down Count.

Contudo, o registrador de contagem, TBPRD, é do tipo inteiro sem sinal (16 bit), e possui valor máximo de 65535. Assim, 1,114 kHz é a menor frequência de amostragem obtida quando o processador está operando em 150 MHz.

No entanto, é possível obter menores frequências de amostragem com o processador rodando em 150 MHz, isso acontece através da utilização do *prescaler* do processador. Esse dispositivo permite dividir a frequência do TBCLK sem alterar a frequência de operação do processador. Alterando o valor dos registradores HSPCLKDIV e CLKDIV, de acordo com os valores da Tabela 2, é possível alterar a frequência do TBCLK através da equação, onde SYSCLKOUT é a frequência de operação do processador:

$$TBCLK = \frac{\text{SYSCLKOUT}}{(\text{HSPCLKDIV} * \text{CLKDIV})} \tag{3}$$

A frequência de amostragem é facilmente alterada de acordo com as informações apresentadas anteriormenteAssim, foi definida uma frequência de 1024 Hz de forma a obter 1024 amostras em um período de um segundo.

Tabela 2 - Valores aceitos pelos registradores.

HSPCLKDIV	CLKDIV
1	1
2	2
4	4
6	8
8	16
10	32
12	64
14	128

Com os valores dos registradores e da frequência definidos, foi realizada a configuração completa do módulo PWM e também das variáveis da memória compartilhada utilizadas para o envio dos dados ao processador M3. A configuração e o código completo desenvolvido para este processador é apresentado no apêndice A.

3.4.3 Programação do Cortex-M3

O microcontrolador Cortex-M3 é responsável por realizar o cálculo da FFT a partir dos dados recebidos do processador C28x e gravar o resultado no cartão de memória SD. Para realizar essa atividade, uma tarefa (*thread*) foi criada no sistema operacional tempo real (TI-RTOS).

O cálculo da FFT das medidas de vibrações, foi realizado através do algoritmo *four1* (PRESS, 2002), implementado em uma função externa a tarefa principal.

O resultado obtido pelo sistema é gravado em um arquivo *comma-separated values* (CSV), o qual armazena os dados de forma tabelada permitindo tratar esses dados em diversos *softwares*. Esse processo de gravação é realizado a cada vez em que o cálculo da FFT é finalizado, isto é, ocorre de acordo com a frequência de amostragem escolhida.

O código que implementa a gravação dos dados no cartão microSD e realiza o cálculo da FFT no Cortex M3 está disponível no apêndice B.

Um fluxograma demonstrando a lógica de funcionamento do código implementado é apresentado na Figura 19.

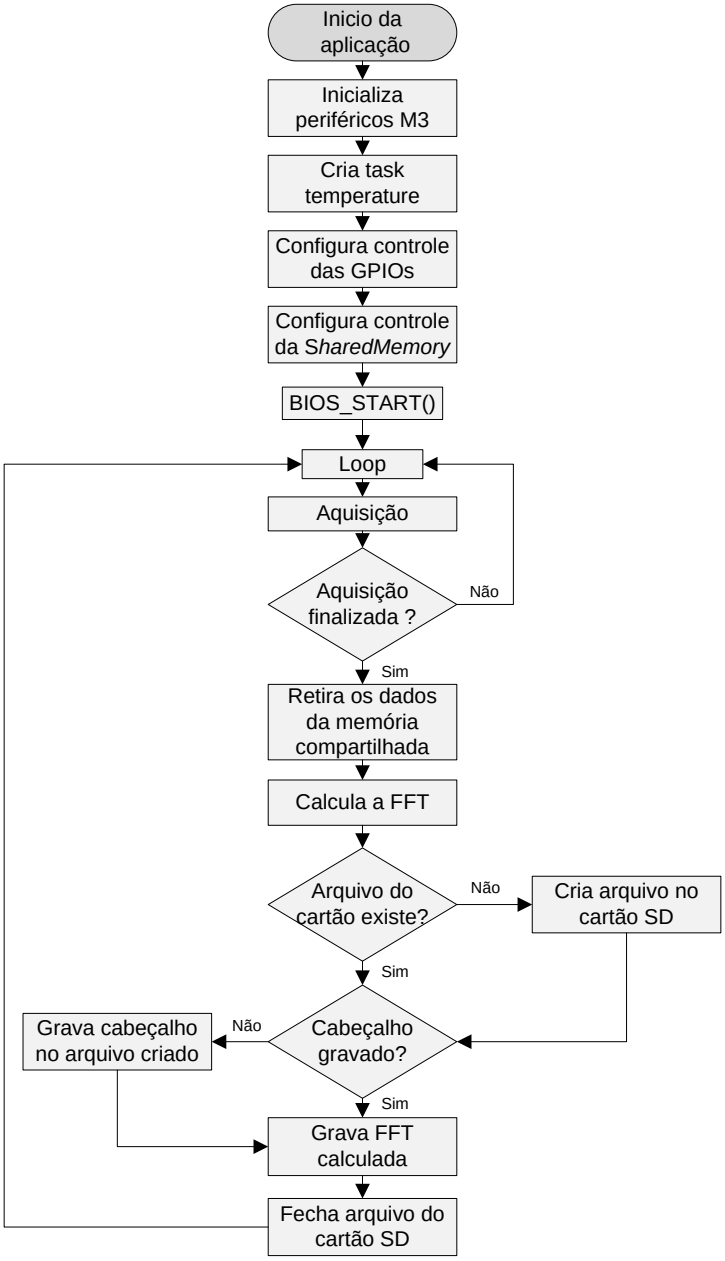

Figura 19 - Fluxograma de funcionamento do Cortex-M3.

3.5 Sistema de geração e armazenamento de energia

Um sistema solar fotovoltaico composto por um elemento gerador de energia, um elemento armazenador e outro elemento responsável pelo controle do armazenamento foi utilizado para o projeto.

3.5.1 Painel solar fotovoltaico

Conforme apresentado anteriormente, o sistema de medição necessita de 24 V para garantir seu funcionamento de maneira adequada. Assim, o painel deve fornecer tensão de máxima potência em torno de 30 V.

Outro requisito analisado foi a quantidade de potência necessária para o funcionamento do sistema de medição. Através da utilização de uma fonte de tensão de 24 V conectada a entrada da placa, foi verificado que o sistema consome uma corrente de 150 mA. Assim, a potência consumida pelo sistema é 3,6 W.

Devido ao fator de capacidade médio de sistemas fotovoltaicos no Brasil ser em torno de 15 % (EPE, 2012), foi utilizado um fator de capacidade de 10 % considerando o pior cenário possível. Com isso, a potência mínima para o painel suprir a demanda do sistema é de 36 W.

Como não foram encontrados painéis solares fotovoltaicos de baixa potência (< 100 W) de 30 V no mercado nacional, foram utilizados dois painéis de 17 V ligados em série para fornecer a tensão necessária para o sistema. A Tabela 3 apresenta as principais especificações dos painéis.

Tabela 3 - Especificações dos painéis fotovoltaicos.

Parâmetro	Valor
Potência Máxima	20 W
Tolerância da potência	± 3%
Tensão de potência máxima	17,6 V
Corrente de potência máxima	1,14 A
Dimensões	480 x 340 x 30 mm
Peso	1,85 Kg

3.5.2 Controlador de carga

A escolha do controlador de carga ocorre de maneira simples, onde a tensão de funcionamento do controlador deve ser compatível com o sistema e a corrente máxima suportada deve ser superior a corrente do sistema.

Com isso, foi utilizado um controlador de carga com tensão de 24 V e corrente de 10 A. Um modelo de menor corrente poderia ter sido utilizado, contudo, não foi encontrado um modelo comercial para essa faixa de tensão. As principais especificações do controlador são apresentadas na Tabela 4.

Tabela 4 - Especificações do controlador de carga.

Parâmetro	Valor
Tensão de operação	12 - 24 V (automático)
Corrente máxima de operação	10 A
Máxima tensão dos painéis	50 V
Dimensões	118 x 76,8 x 33 mm
Peso	200 g

3.5.3 Bateria

Para o sistema de armazenamento de energia, optou-se pela utilização de baterias chumbo-ácida reguladas por válvula (VRLA) por apresentarem uma boa relação custo/benefício. Novamente a tensão de 24 V foi um requisito para o dimensionamento, pois as baterias VRLA apresentam tensão nominal de 12 V. Assim, para armazenar 24 V foram utilizadas duas baterias ligadas em série.

Outro ponto analisado foi a capacidade da bateria em ampère-hora (Ah). Uma bateria VRLA de 100 Ah C20, por exemplo, permite uma corrente de descarga de 5 A durante vinte horas. Dessa forma, como o sistema de medição consume uma corrente de 150 mA, baterias VRLA de 5 Ah C20 são suficientes para manter o sistema em funcionamento por até trinta horas. As especificações das baterias utilizadas são mostradas na Tabela 5.

Tabela 5 - Especificações da bateria.

Parâmetro	Valor
Tensão nominal	12 V
Capacidade nominal	5 Ah
Tensão de flutuação	13,5 – 13,8 V
Corrente de curto circuito	100 A
Dimensões	90 x 70 x 101 mm
Peso	1,7 Kg

Uma representação geral do sistema de geração e armazenamento de energia é apresentado na Figura 20.

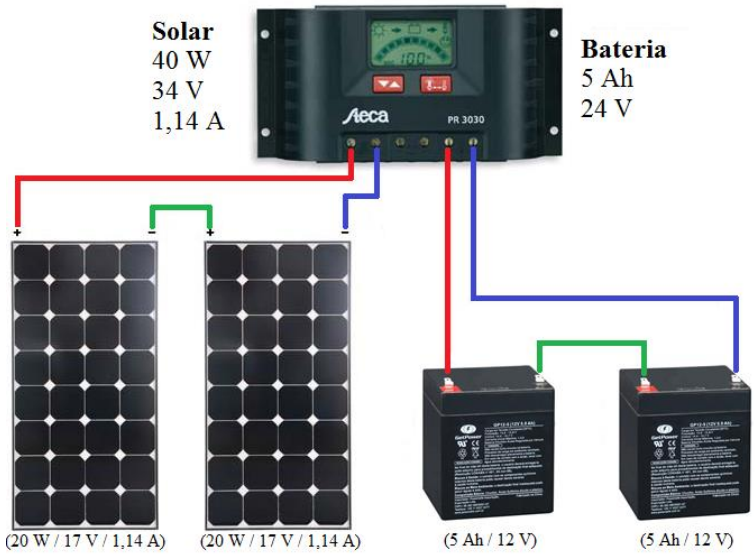

Figura 20 - Sistema de geração e armazenamento de energia.

4 RESULTADOS EXPERIMENTAIS

Neste capítulo são apresentados a placa de circuito impresso desenvolvida e os testes de validação realizados a fim de comprovar o funcionamento do sistema.

4.1 Placa de circuito impresso

Para a execução deste trabalho, foi desenvolvido o *layout* de uma placa de circuito impresso utilizando o *software Altium*®.

A placa elaborada contempla os circuitos de potência, o circuito de condicionamento de sinal, quatro conectores padrão BNC para conexão com os acelerômetros, um conector *borne* para conexão com o controlador de carga e um conector para o microcontrolador *Concerto*.

Assim, a Figura 21 apresenta o *layout* no *software* e a configuração final da placa montada.

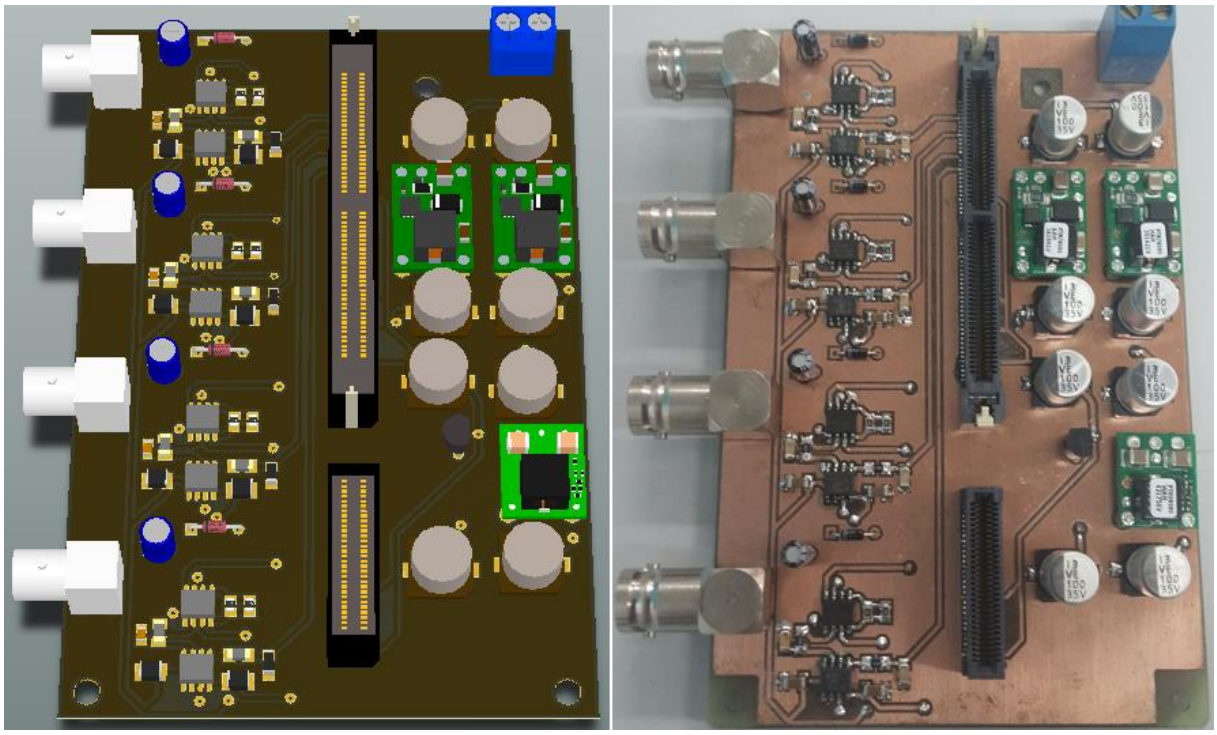

Figura 21 - Layout e placa de interface concluída.

4.2 Validação do sistema

Para a validação do sistema foram realizados testes a fim de verificar a autenticidade dos resultados obtidos. O processo de validação se dividiu em três diferentes etapas:

- 1ª Etapa: Validação da aquisição de dados e do cálculo da FFT;
- 2ª Etapa: Calibração da leitura dos acelerômetros;
- 3ª Etapa: Obtenção das frequências naturais de uma estrutura.

4.2.1 Validação da aquisição de dados e do cálculo da FFT

Esta etapa da validação visou verificar se não existiam erros no processo de aquisição de dados, na transferência desses dados entre os processadores e também no cálculo da FFT.

A realização do teste consistiu na emulação de um sinal, formado pela composição de senos e cossenos, com amplitude e frequência conhecidas para cada canal dentro da rotina de interrupção responsável pela leitura do conversor A/D. Este teste também validou a gravação de dados no cartão micro SD, visto que os resultados foram obtidos através do arquivo gravado no cartão. Um dos sinais introduzidos no microcontrolador é apresentado abaixo, enquanto na Figura 22 é apresentado o resultado obtido.

$$canal\ 1 = 700 * sen(30 * t) + 200 * sen(10 * t) \quad (4)$$

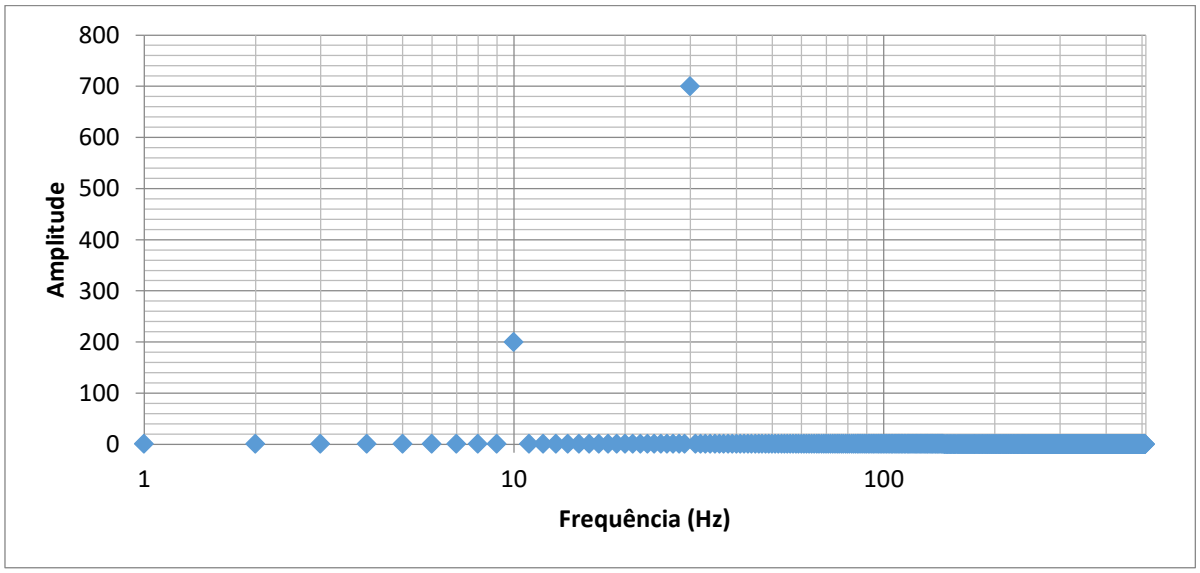

Figura 22 - Validação do software.

Analisando a Figura 22, é possível visualizar, através do espectro de frequência, que a amplitude e a frequência são iguais ao sinal aplicado. Assim, fica evidente que o funcionamento do sistema de aquisição, a troca de dados e o cálculo da FFT são válidos.

4.2.2 Calibração da leitura dos acelerômetros

Com a primeira etapa da validação concluída, o segundo passo consistiu em conectar os acelerômetros ao sistema de aquisição e realizar a calibração da leitura realizada. Essa calibração foi necessária devido à existência dos ganhos do circuito de instrumentação e a correção do *offset* realizada no microcontrolador.

Para realizar a calibração, foi empregado o excitador de calibração *type* 4294 da B&K do Laboratório de Acústica do Centro de Tecnologia da Universidade Federal de Santa Maria. Este equipamento permite um ajuste preciso da instrumentação de medição a partir de uma aceleração padrão de 10 m/s² (1,0197 g). Além disso, embarca um excitador eletrônico controlado por um oscilador de cristal com uma frequência de 159,15 Hz.

Todos os canais foram confeccionados com os mesmos componentes, contudo, pequenas diferenças podiam existir devido ao grau de tolerância dos componentes. Assim, o ensaio de calibração, apresentado na Figura 23, foi realizado em todos os canais.

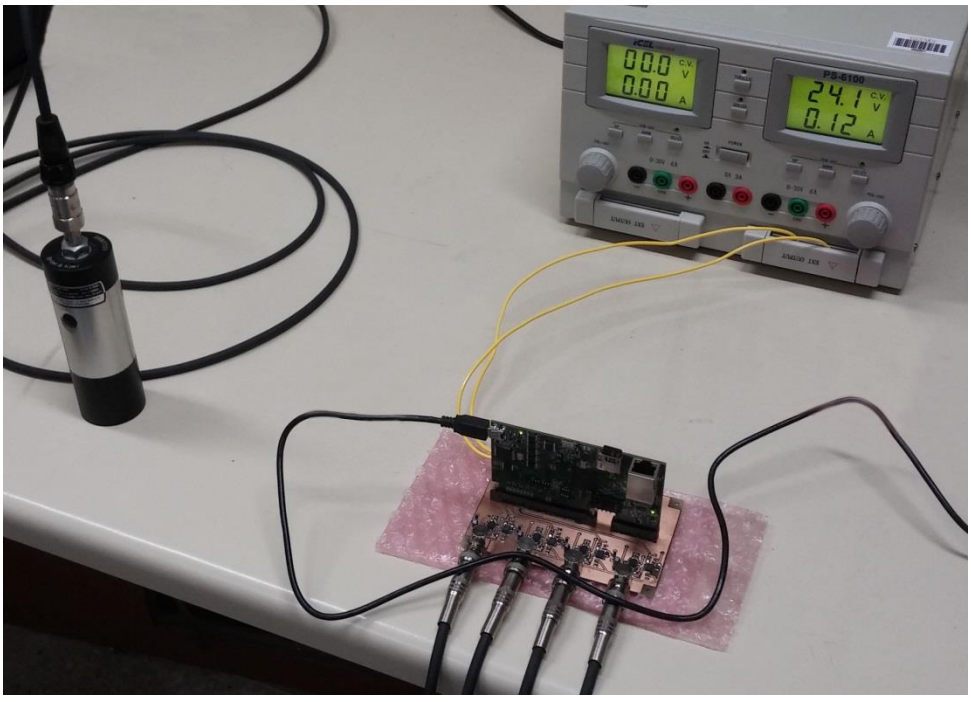

Figura 23 - Ensaio de calibração.

Realizado o procedimento de calibração para todos os canais, foram obtidos os dados apresentados nas Figura 24 e Figura 25.

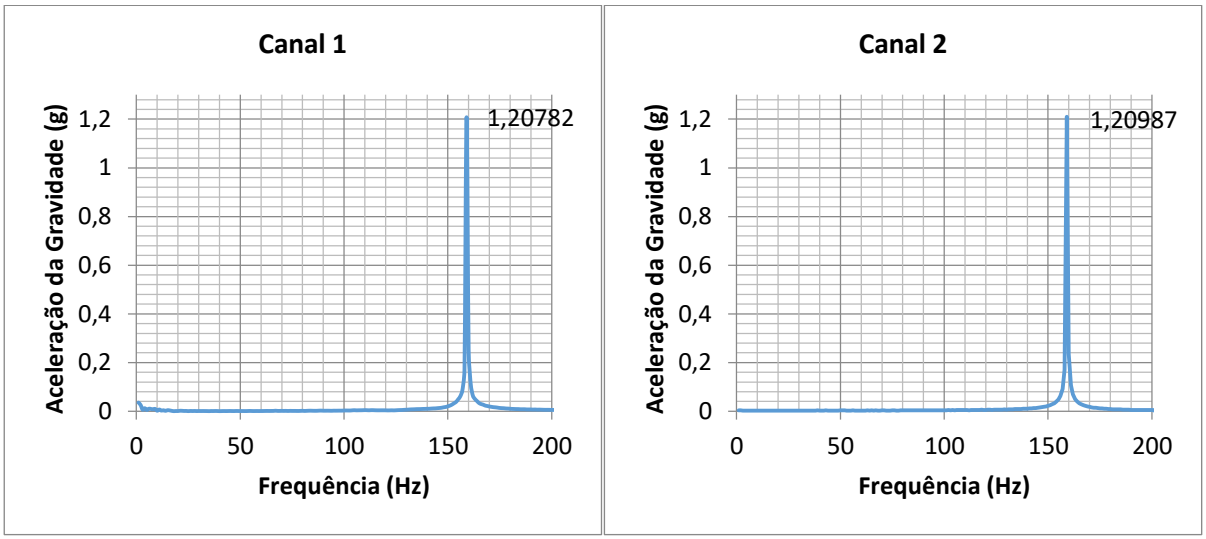

Figura 24 - Resultado do ensaio de calibração para o canal 1 e 2.

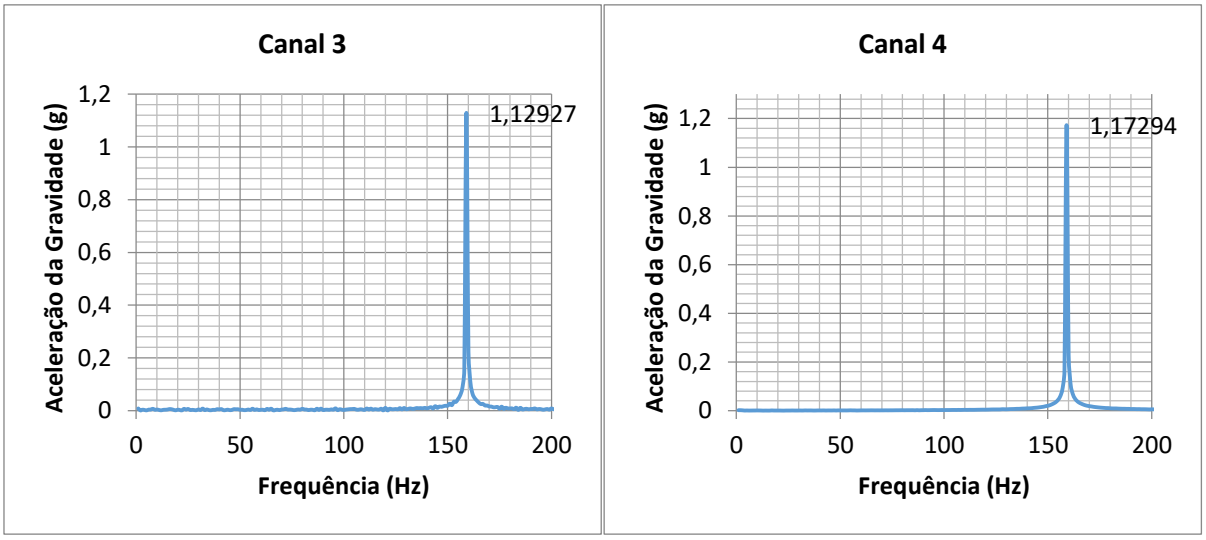

Figura 25 - Resultado do ensaio de calibração para o canal 3 e 4.

Através dos resultados encontrados, foi verificada a existência de erro de medição em todos os canais devido ao *offset* necessário para tornar a medida positiva. Para resolver essa diferença, foi necessária a aplicação de um fator de correção no microcontrolador. O fator de correção foi calculado comparando o valor medido com o valor aplicado pelo excitador através da seguinte equação:

$$Fator\ de\ correção = \frac{1{,}01971162129}{valor\ medido} \quad (4)$$

Utilizando a equação, os fatores de correção para cada canal do sistema foram calculados e implementados no código. A Tabela 6 apresenta os fatores encontrados.

Tabela 6 - Fatores de correção resultantes do ensaio de calibração.

Fator de Correção	Valor
Canal 1	0,842827166846445
Canal 2	0,844260469110394
Canal 3	0,902983913043400
Canal 4	0,869361930193567

4.2.3 Obtenção das frequências naturais de uma estrutura

Para finalizar a validação do sistema, um ensaio em uma viga livre-livre foi realizado a fim de encontrar as suas frequências naturais de vibração. Com este ensaio, foi possível verificar o sistema desenvolvido para a condição em que será utilizado na torre.

O ensaio foi realizado posicionando um acelerômetro sobre a viga, utilizando um imã para fixá-lo, e com o auxílio de um martelo de impacto foi produzido um impacto com a viga fazendo a mesma vibrar. Na Figura 26 é possível observar o conjunto viga-acelerômetro.

Figura 26 - Acelerômetro fixado à viga.

Como resultados desse ensaio, as frequências naturais de vibração da viga foram obtidas e são apresentados na Figura 27.

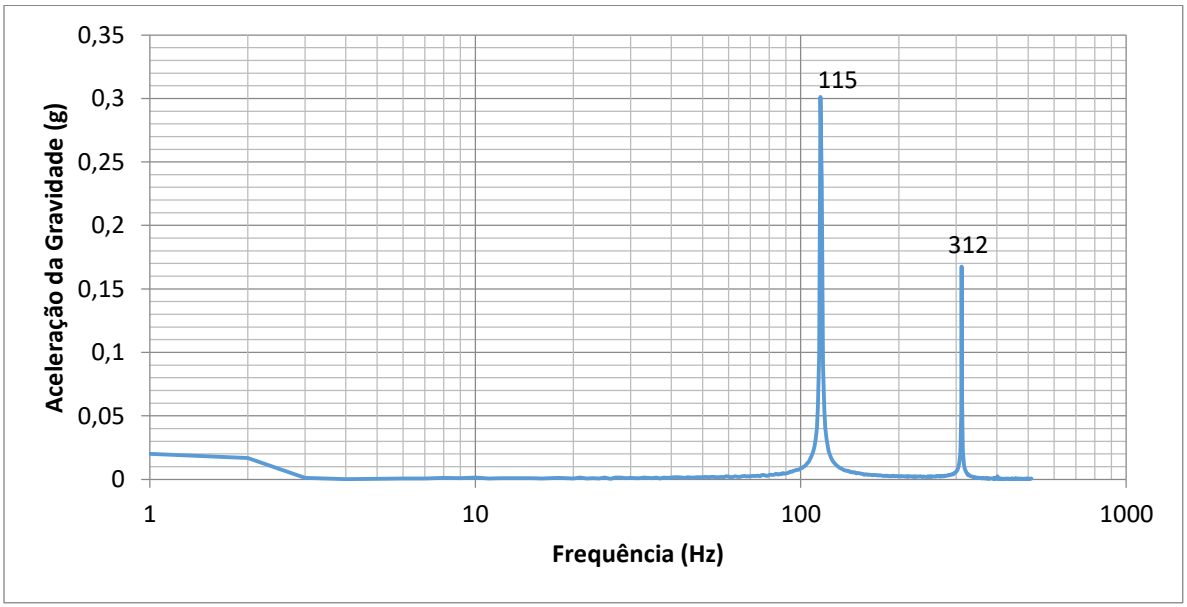

Figura 27 - Frequências naturais de vibração da viga.

Vale ressaltar que devido à taxa de amostragem de 1024 Hz utilizada, apenas as frequências de 115 Hz e 312 Hz foram encontradas. Esses valores correspondem as duas primeiras frequências naturais do sistema.

Segundo (OLIVEIRA, 2015), utilizando vinte elementos de discretização, as duas primeiras frequências são 115,59 Hz e 316,46 Hz, dessa forma, o sistema desenvolvido apresentou erro de 0,5% para a menor frequência e 1,41% para a segunda frequência. Com a utilização de cem elementos, a primeiras frequências são 116,59 Hz e 320,07 Hz, assim, o erro é de 1,36% e 2,52% para a primeira e segunda frequência, respectivamente.

CONCLUSÃO

O trabalho atingiu os objetivos propostos inicialmente, sendo construído um sistema de medição de vibrações, que permite alteração da taxa de amostragem, leitura das medidas através de um cartão microSD, além de possuir funcionamento autônomo.

A construção do sistema baseou-se na necessidade de desenvolvimento de uma placa de interface entre os acelerômetros e o microcontrolador. Através dessa placa, a conexão entre esses dispositivo foi estabelecida. Além disso, integrou em dimensões reduzidas, todos os equipamentos necessários para o funcionamento do *Concerto*.

A comprovação de que as medições realizadas pelo sistema são corretas, ocorreu a partir dos ensaios de validação. Através desses ensaios, foi observado o correto funcionamento do sistema de medição, necessitando apenas de um fator de correção no valor de aceleração medido. A comparação com a medição de um sistema profissional, comprovou o funcionamento do sistema com pequeno erro para a menor frequência, foco principal do sistema.

O próximo passo do trabalho é a aplicação do sistema ao conjunto estrutural, dessa forma, é possível verificar realmente a eficiência do sistema. Para trabalhos futuros, algumas funcionalidades devem ser adicionadas ao sistema a fim de melhorar seu desempenho. Uma funcionalidade é a utilização de um relógio, o qual proporcionaria uma melhora no processo de gravação das medidas. Como o sistema não utiliza todos os recursos do microcontrolador, dois circuitos poderiam ser adicionados à placa de interface, sendo responsáveis pela medição da corrente e tensão do conjunto de baterias.

REFERÊNCIAS BIBLIOGRÁFICAS

BOLINA, C. C.; PALECHOR, E.U. L.; VÁSQUEZ, M. P. R; NICARIO, W. G.; GUTIERRES, M. P. D. e LOPEZ, A. A. O. **Vibrações: As Frequências Naturais estimada e Experimental de uma Estrutura.** CNMAI, 2014. Disponível em: < http://www.proceedings.blucher.com.br/article-details/vibraes-as-frequncias-naturais-estimada-e-experimental-de-uma-estrutura-11900> Acesso em: 11 out. 2016

CHEN, J; JIANG D. **Modal Analysis of Wind Turbine Tower.** WNWEC, 2010.

EMPRESA DE PESQUISA ENERGÉTICA - EPE. **Análise da Inserção da Geração Solar na Matriz Elétrica Brasileira.** 2012. Disponível em: <http://www.epe.gov.br/geracao/documents/estudos_23/nt_energiasolar_2012.pdf> Acesso em: 05 jan. 2016.

GLOBAL WIND ENERGY COUNCIL – GWEC. **Global wind energy outlook 2008**. Brussels: GWEC, 2008.

GUIMARÃES, P. C.; RIBEIRO, C. M.; BASTOS, L. E. G.;VALENTE, L. C. G.;SILVA, P. C. d. e OLIVEIRA, R. X. d. **Manual de Engenharia para Sistemas Fotovoltaicos**. CEPEL - CRESESB, 2004.

IMHOFF, J. **Desenvolvimento de Conversores Estáticos para Sistemas Fotovoltaicos Autônomos.** 2007. Dissertação de Mestrado – Universidade Federal de Santa Maria.

IMI SENSORS. **Platinum Low-cost Industrial ICP® AccelerometerInstallation and Operating Manual.** Disponível em: <https://www.pcb.com/contentstore/docs/PCB_Corporate/IMI/Products/Manuals/603C01.pdf > Acesso em: 11 set. 2015

NSK. **Sensores de Vibração.** Disponível em:<http://www1.br.nsk.com/etreinar/sm/sm4.asp> Acesso em: 17 dez. 2015.

OLIVEIRA, T. A. **Análise Modal de Estruturas: Abordagem Numérica e Experimental em um Sistema Livre-Livre.** Trabalho de Conclusão de Curso – Universidade Federal de Santa Maria.

PAULA, R. de L. **Modelo acoplado torre-aerogerador de eixo horizontal.** 2012 Dissertação de Mestrado – Universidade Federal do Rio de Janeiro.

PCB PIEZOTRONICS. **Signal Conditioning Basics for ICP® & Charge Output Sensors.** 2015. Disponível em: <https://www.pcb.com/techsupport/tech_signal>

PCB PIEZOTRONICS. **Introduction to Piezoelectric Accelerometers.** 2016. Disponível em: <https://www.pcb.com/techsupport/tech_accel.aspx> Acesso: 06 jan. 2016.

PRESS, W. H.; TEUKOLSKY, S. A.; VETTERLING, W. T.; FLANNERY, B. P. **Numerical Recipes in C.** Cambridge University Press, 2002.

RAO, S. **Vibrações mecânicas.** Pearson Prentice Hall, 2009.

SILVA, S. **Vibrações Mecânicas.** UNIOESTE, Foz do Iguaçu, 2009.

SKF. **Vibration Sensors.** 1999. Disponível em: <http://www.exvalos.cz/soubory/File/SKF/SNIMACE_VIBRACI.pdf > Acesso em: 18 dez. 2015

APÊNDICE A

O código que realiza a amostragem das medidas, empregado no processador C28x, é apresentado a seguir.

```
//###########################################################################
// UNIVERSIDADE FEDERAL DE SANTA MARIA - UFSM
// GRUPO DE ELETRONICA DE POTENCIA E CONTROLE - GEPOC
// PROJETO EÓLICA 2015
//###########################################################################
// TCC - DESENVOLVIMENTO DE UM SISTEMA DE MEDIÇÃO DE VIBRAÇÕES PARA UMA TORRE DE
// UM AEROGERADOR
// Código referente ao MCU F28335 do DSP Concerto F28M36x
// Código base: Juliano Grigulo
// AUTOR: Matheus Felipe Schöne E-MAIL: schone.matheus@gmail.com
//###########################################################################
// DESCRIÇÃO:
// Este código realiza:
// > Parametrização dos módulos de conversão analógica/digital do F28335 e do
módulo PWM1, disparando a conversão AD por meio do módulo ePWM.
// > Aquisição de dados dos acelerômetro da torre e envio para o Cortex-M3
// > Pisca um LED de um GPIO dentro da rotina de PWM.
//###########################################################################
/* Arquivos de cabeçalho XDCtools */
#include<xdc/std.h>
#include<xdc/runtime/Log.h>
#include<xdc/runtime/IHeap.h>
#include<xdc/runtime/Memory.h>
#include<xdc/runtime/Error.h>
#include<xdc/runtime/System.h>
#include<xdc/cfg/global.h>
/* Arquivos de cabeçalho IPC */
#include<ti/ipc/MessageQ.h>
#include<ti/ipc/MultiProc.h>
/* Arquivos de cabeçalho BIOS  */
#include<ti/sysbios/BIOS.h>
#include<ti/sysbios/heaps/HeapBuf.h>
#include<ti/sysbios/knl/Task.h>
#include<ti/sysbios/knl/Semaphore.h>
#include<string.h>
#include"demo.h"
#include"math.h"
#include"DSP28x_Project.h"// Device Headerfile and Examples Include File
// Configure which ePWM timer interrupts are enabled at the PIE level:
// 1 = enabled,  0 = disabled
#define PWM1_INT_ENABLE  1
//---------------------------------------------------------------------------
// Declaração dos buffers e alocação dos mesmos na memória compartilhada
//---------------------------------------------------------------------------
double escreves0[2048];
double escreves1[2048];
double escreves2[2048];
double escreves3[2048];
longint escreves4[2048];
```

```
longint escreves5[2048];
longint flag_C28[2048];
longint le_flag_M3[2048];
#pragma DATA_SECTION(escreves0, "SHARERAMS0");
#pragma DATA_SECTION(escreves1, "SHARERAMS1");
#pragma DATA_SECTION(escreves2, "SHARERAMS2");
#pragma DATA_SECTION(escreves3, "SHARERAMS3");
#pragma DATA_SECTION(escreves4, "SHARERAMS4");
#pragma DATA_SECTION(escreves5, "SHARERAMS5");
#pragma DATA_SECTION(flag_C28, "SHARERAMS6");
#pragma DATA_SECTION(le_flag_M3, "SHARERAMS7");
Void adc_fxn(UArg arg);
// Variáveis dos canais de AD's
int     offset_ch01=270, offset_ch02=265, offset_ch03=265, offset_ch04=270;
int     offset = 2048;
int     canal_01 = 0, canal_02 = 0, canal_03 = 0, canal_04 = 0;
float    medida_01 = 0, medida_02 = 0, medida_03 = 0, medida_04 = 0;
// Variáveis medidas em seu valor real
double acel_01, acel_02, acel_03, acel_04;
// Variáveis auxiliares
float flag=0.;
int BUFFER_SIZE1=1024;
int buffer01[1024], buffer02[1024], buffer03[1024], buffer04[1024],
buffer05[1024], buffer06[1024], buffer07[1024], buffer08[1024]; // Não retirar,
causa problema na memória compartilhada
//Outras Variaveis
int k=0;     // Incremento usado para salvar os dados adquiridos na memória
compartilhada
float count_led=0.;       // temporizador de acionamento do LED

//---------------------------------------------------------------------------
// Rotina principal, responsável pela definição da frequência de acionamento do
led
//---------------------------------------------------------------------------

Void tsk0_func(UArg arg0, UArg arg1)
{
while (1) {
if(flag>=1.){
        flag=0.;
        count_led++;
       if (count_led < 1000) GpioG1DataRegs.GPADAT.bit.GPIO31 = 0;
       if (count_led > 1000 && count_led < 2000) GpioG1DataRegs.GPADAT.bit.GPIO31 =
1;
       if (count_led > 2000 ) count_led = 0;
}//flag
  }//while
}//tsk0

//---------------------------------------------------------------------------
// Função MAIN que executa inicialização do DSP, parametrização do ADC e PWM e
aquisição de dados
//---------------------------------------------------------------------------

Void main()
{
// Initialize System Control for Control and Analog Subsytems
// Enable Peripheral Clocks
```

```
// This example function is found in the F28M35x_SysCtrl.c file.
InitSysCtrl();

// Step 2. Initialize GPIO:
// This example function is found in the F28M36x_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
InitGpio();

InitEPwmGpio();


    EALLOW;

//---------------------------------------------------------------------------
//Configuração das GPIO's como saída
//---------------------------------------------------------------------------

GpioG1CtrlRegs.GPADIR.bit.GPIO31  = 1;  //Set as output  //LEDs
        GpioG1CtrlRegs.GPAMUX2.bit.GPIO31 = 0;
        GpioG1CtrlRegs.GPADIR.bit.GPIO29  = 1;  //Set as output  //LEDs
        GpioG1CtrlRegs.GPAMUX2.bit.GPIO29 = 0;    // GPIO functionality GPIO29
        GpioG1CtrlRegs.GPBDIR.bit.GPIO40  = 1;  //Set as output  //PWM
        GpioG1CtrlRegs.GPBMUX1.bit.GPIO40 = 0;    // GPIO functionality GPIO40
        GpioG1CtrlRegs.GPBDIR.bit.GPIO41  = 1;  //Set as output  //PWM
        GpioG1CtrlRegs.GPBMUX1.bit.GPIO41 = 0;    // GPIO functionality GPIO40
        GpioG1CtrlRegs.GPADIR.bit.GPIO17  = 1;  //Set as output  //PWM
        GpioG1CtrlRegs.GPAMUX2.bit.GPIO17 = 0;    // GPIO functionality GPIO17
EDIS;
    GpioG1DataRegs.GPADAT.bit.GPIO31 = 1;
GpioG1DataRegs.GPADAT.bit.GPIO29 = 0;

// Copy time critical code and Flash setup code to RAM
// The  RamfuncsLoadStart, RamfuncsLoadEnd, and RamfuncsRunStart
// symbols are created by the linker. Refer to the F28M35H52C1.cmd file.
    memcpy(&RamfuncsRunStart, &RamfuncsLoadStart, (size_t)&RamfuncsLoadSize);

// Call Flash Initialization to setup flash waitstates
// This function must reside in RAM
InitFlash();

// Initialize all the Device Peripherals:
// This function is found in F28M35x_InitPeripherals.c
InitAdc1();

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;      // Stop all the TB clocks
    EDIS;

    EALLOW;

    SysCtrlRegs.PCLKCR1.bit.EPWM1ENCLK = 1;  // ePWM1
    SysCtrlRegs.PCLKCR1.bit.EPWM2ENCLK = 1;  // ePWM2
    SysCtrlRegs.PCLKCR1.bit.EPWM3ENCLK = 1;  // ePWM3
    SysCtrlRegs.PCLKCR1.bit.EPWM4ENCLK = 1;  // ePWM4
    SysCtrlRegs.PCLKCR1.bit.EPWM5ENCLK = 1;  // ePWM5
    SysCtrlRegs.PCLKCR1.bit.EPWM6ENCLK = 1;  // ePWM6

    EDIS;
```

```
    EALLOW;
//Assumes ePWM1 clock is already enabled in InitSysCtrl();
//Set event triggers (SOCA) for ADC SOC1
    EPwm1Regs.ETSEL.bit.SOCAEN  = 1;          // Enable SOC on A group
    EPwm1Regs.ETSEL.bit.SOCASEL = ET_CTR_ZERO;    // Select SOC from CMPA on
upcount
    EPwm1Regs.ETPS.bit.SOCAPRD  = ET_1ST;        // Generate pulse on every 1st
event

//-----------------------------------------------------------------------------
//Configuração dos PWM's
//-----------------------------------------------------------------------------

           EPwm1Regs.TBPRD = PWM_PERIOD;   // Set timer period -- PWM_PERIOD é
definido no arquivo F28M36x_Examples.h
EPwm1Regs.TBPHS.half.TBPHS = 0; // Set Phase register to zero
           EPwm1Regs.TBCTR = 0; // clear TB counter
           EPwm1Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Symmetric
           EPwm1Regs.TBCTL.bit.PHSEN = TB_DISABLE; // Phase loading disabled
           EPwm1Regs.TBCTL.bit.PRDLD = TB_SHADOW;
           EPwm1Regs.TBCTL.bit.SYNCOSEL = TB_SYNC_DISABLE;
           EPwm1Regs.TBCTL.bit.HSPCLKDIV = TB_DIV2; // TBCLK = SYSCLKOUT/2
           EPwm1Regs.TBCTL.bit.CLKDIV = TB_DIV1;
           EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
           EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
           EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // load on CTR = Zero
           EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO; // load on CTR = Zero
           EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;
           EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;
           EPwm1Regs.AQCTLB.bit.CBU = AQ_SET;
           EPwm1Regs.AQCTLB.bit.CBD = AQ_CLEAR;

        EDIS;

    EALLOW;
    SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;         // Start all the timers synced
    EDIS;

    EALLOW;

// Configure ADC

    Adc1Regs.ADCCTL2.bit.ADCNONOVERLAP = 1;     // Enable non-overlap mode i.e.
conversion and future sampling events dont overlap
    Adc1Regs.ADCCTL1.bit.INTPULSEPOS   = 1;     // ADCINT1 trips after AdcResults
latch
    Adc1Regs.INTSEL1N2.bit.INT1E       = 1;     // Enabled ADCINT1
    Adc1Regs.INTSEL1N2.bit.INT1CONT    = 0;     // Disable ADCINT1 Continuous mode
    Adc1Regs.INTSEL1N2.bit.INT1SEL     = 0;     // setup EOC0 to trigger ADCINT1
to fire

    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN0 = 1;    // Simultaneous sampling enable
for SOC0/SOC1
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN2 = 1;    // Simultaneous sampling enable
for SOC2/SOC3
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN4 = 1;    // Simultaneous sampling enable
for SOC4/SOC5
```

```
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN6 = 1;    // Simultaneous sampling enable
for SOC6/SOC7
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN8 = 1;    // Simultaneous sampling enable
for SOC8/SOC9
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN10 = 1;   // Simultaneous sampling enable
for SOC10/SOC11
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN12 = 1;
    Adc1Regs.ADCSAMPLEMODE.bit.SIMULEN14 = 1;

//Setting up the trigger source
    AnalogSysctrlRegs.TRIG1SEL.all  = 5;      // Assigning EPWM1SOCA to ADC
TRIGGER 1 of the ADC module
    Adc1Regs.ADCSOC0CTL.bit.CHSEL = 0;      // set SOC0 channel select to
ADCINA0/ADCINB0 pair
    Adc1Regs.ADCSOC0CTL.bit.TRIGSEL = 5;      // Set SOC0 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC0CTL.bit.ACQPS = 6;       // set SOC0 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
    Adc1Regs.ADCSOC2CTL.bit.CHSEL = 2;      // set SOC2 channel select to
ADCINA2/ADCINB2 pair
    Adc1Regs.ADCSOC2CTL.bit.TRIGSEL = 5;      // set SOC2 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC2CTL.bit.ACQPS = 6;       // set SOC2 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
    Adc1Regs.ADCSOC4CTL.bit.CHSEL = 3;    // set SOC4 channel select to
ADCINA3/ADCINB3 pair
    Adc1Regs.ADCSOC4CTL.bit.TRIGSEL = 5;      // set SOC4 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC4CTL.bit.ACQPS = 6;       // set SOC4 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
    Adc1Regs.ADCSOC6CTL.bit.CHSEL = 4;    // set SOC6 channel select to
ADCINA4/ADCINB4 pair
    Adc1Regs.ADCSOC6CTL.bit.TRIGSEL = 5;      // set SOC6 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC6CTL.bit.ACQPS = 6;       // set SOC6 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
    Adc1Regs.ADCSOC9CTL.bit.CHSEL = 6;    // set SOC8 channel select to
ADCINA6/ADCINB6 pair
    Adc1Regs.ADCSOC9CTL.bit.TRIGSEL = 5;      // set SOC8 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC9CTL.bit.ACQPS = 6;       // set SOC8 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
    Adc1Regs.ADCSOC11CTL.bit.CHSEL = 7;    // set SOC10 channel select to
ADCINA6/ADCINB6 pair
    Adc1Regs.ADCSOC11CTL.bit.TRIGSEL = 5;      // set SOC10 start trigger to ADC
Trigger 1(EPWM1 SOCA) of the adc
    Adc1Regs.ADCSOC11CTL.bit.ACQPS = 6;       // set SOC10 S/H Window to 7 ADC
Clock Cycles, (6 ACQPS plus 1)

EDIS;

    flag_C28[0] = 0; // Inicia a flag de sinalização de termino de aquisição como
0
BIOS_start();    // does not return
}
//---------------------------------------------------------------------------
// FUNÇÃO disparada pelo módulo PWM1 que realiza a leitura dos ADs e acionamento
do LED
```

```
//---------------------------------------------------------------------------

Void adc_fxn(UArg arg)
{
	flag_C28[0] = 0; // Zera a flag, e possibilita a aquisição do AD

//---------------------------------------------------------------------------
// Aquisição das grandezas medidas pelos A/D's
//---------------------------------------------------------------------------

//-- Conversão dos canais A/D

	canal_01 = Adc1Result.ADCRESULT0;    // Recebe a aquisição da Porta 9 do AD
	canal_02 = Adc1Result.ADCRESULT4;    // Recebe a aquisição da Porta 17 do AD
	canal_03 = Adc1Result.ADCRESULT11;   // Recebe a aquisição da Porta 27 do AD
	canal_04 = Adc1Result.ADCRESULT5;    // Recebe a aquisição da Porta 33 do AD

//-- Normalização das medidas - offset zero

	medida_01 = canal_01 - offset + offset_ch01;
	medida_02 = canal_02 - offset + offset_ch02;
	medida_03 = canal_03 - offset + offset_ch03;
	medida_04 = canal_04 - offset + offset_ch04;

//-- Aquisições em valores reais (aceleração)

	acel_01 = corr_ganh1*ganho_grav*medida_01;  // Variáveis corr_ganh e
ganho_grav são definidas no arquivo F28M36x_Examples.h
	acel_02 = corr_ganh2*ganho_grav*medida_02;
	acel_03 = corr_ganh3*ganho_grav*medida_03;
	acel_04 = corr_ganh4*ganho_grav*medida_04;

	flag=flag+1;

//---------------------------------------------------------------------------
// Escreve os valores medidos na memória compartilhada
//---------------------------------------------------------------------------

	escreves0[k] = acel_01;
	escreves1[k] = acel_02;
	escreves2[k] = acel_03;
	escreves3[k] = acel_04;

//-- Contador usado para encerrar a aquisição de dados e sinalizar ao ARM para
realizar o cálculo da FFT

	if(k == BUFFER_SIZE1 - 1)
	{
		k = 0;
		flag_C28[0] = 1; // Seta a flag para 1, informando ao ARM que a
aquisição terminou
	}
	else k++;

	Adc1Regs.ADCINTFLGCLR.bit.ADCINT1 = 1;  //Clear ADCINT1 flag reinitialize
		// for next SOC
	//}
}
```

```
//TODO
```

APÊNDICE B

O código, empregado no processador Cortex-M3, é apresentado a seguir.

```
//////////////////////////////////////////////////////////////////////////////
// UNIVERSIDADE FEDERAL DE SANTA MARIA - UFSM
// GRUPO DE ELETRONICA DE POTENCIA E CONTROLE - GEPOC
// PROJETO EÓLICA 2015
//////////////////////////////////////////////////////////////////////////////
// TCC - DESENVOLVIMENTO DE UM SISTEMA DE MEDIÇÃO DE VIBRAÇÕES PARA UMA TORRE DE
// UM AEROGERADOR
// Código referente ao MCU Cortex-M3 do DSP Concerto F28M36x
// Código base: Juliano Grigulo
// AUTOR: Matheus Felipe Schöne E-MAIL: schone.matheus@gmail.com
//////////////////////////////////////////////////////////////////////////////
// DESCRIÇÃO:
// Este código realiza:
// > Comunicação com MCU F28335 via SHAREDMEMORY
// > Cálculo da FFT dos dados adquiridos pelo MCU F28335
// > Armazenamento da FFT calculada no cartão de memória externa
//////////////////////////////////////////////////////////////////////////////
/* XDCtools Header files */
#include<xdc/std.h>
#include<xdc/cfg/global.h>
#include<xdc/runtime/Diags.h>
#include<xdc/runtime/Error.h>
#include<xdc/runtime/IHeap.h>
#include<xdc/runtime/Log.h>
#include<xdc/runtime/Memory.h>
#include<xdc/runtime/System.h>
/* IPC Header files */
#include<ti/ipc/MessageQ.h>
#include<ti/ipc/MultiProc.h>
/* BIOS Header files */
#include<ti/sysbios/BIOS.h>
#include<ti/sysbios/heaps/HeapBuf.h>
#include<ti/sysbios/knl/Task.h>
#include<ti/sysbios/knl/Clock.h>
/* TI-RTOS Header files */
#include<ti/drivers/SDSPI.h>
#include<ti/drivers/I2C.h>
#include<ti/drivers/GPIO.h>
/* NDK Header files */
#include<ti/ndk/inc/netmain.h>
#include<ti/ndk/inc/_stack.h>
/* Example/Board Header file */
#include"Board.h"
#include<stdlib.h>
#include"TMDXDOCK28M36.h"
#include"demo.h"
#include"USBCDCD_LoggerIdle.h"
#include<math.h>
#include<stdint.h>
#include<stdio.h>
#include<string.h>
```

```
#include<file.h>
//setup M3
#include"inc/hw_sysctl.h"
#include"inc/hw_ints.h"
#include"inc/hw_memmap.h"
#include"inc/hw_nvic.h"
#include"inc/hw_types.h"
#include"inc/hw_gpio.h"
#include"board_drivers/set_pinout_f28m36x.h"
#include"driverlib/ipc.h"
#include"driverlib/sysctl.h"
#include"driverlib/interrupt.h"
#include"driverlib/debug.h"
#include"driverlib/cpu.c"
#include"driverlib/gpio.h"
//Memória compartilhada
#include"driverlib/ram.h"

//---------------------------------------------------------------------------
// Declaração dos buffers e alocação dos mesmos na memória compartilhada
//---------------------------------------------------------------------------

float les0[2048];
float les1[2048];
float les2[2048];
float les3[2048];
int les4[2048];
int les5[2048];
int le_flag_C28[2048];
int flag_M3[2048];
#pragma DATA_SECTION(les0, "SHARERAMS0");
#pragma DATA_SECTION(les1, "SHARERAMS1");
#pragma DATA_SECTION(les2, "SHARERAMS2");
#pragma DATA_SECTION(les3, "SHARERAMS3");
#pragma DATA_SECTION(les4, "SHARERAMS4");
#pragma DATA_SECTION(les5, "SHARERAMS5");
#pragma DATA_SECTION(le_flag_C28, "SHARERAMS6");
#pragma DATA_SECTION(flag_M3, "SHARERAMS7");

// Definição das constantes utilizadas pela função four1 (FFT)

#define PI 3.14159265358979323846
#define TWOPI (2.0*PI)
externvoidSerialInit();
externvoidSerialMain();

// Buffers utilizados para o cálculo da FFT

      float medida_acel_01[2049], medida_acel_02[2049], medida_acel_03[2049],
medida_acel_04[2049];

// Variáveis Auxiliares

      int k = 0;  // Contador utilizado no laço for que recebe os dados da memória
compartilhada

      int kk = 0; // Contador utilizado no laço for que realiza a gravação dos
dados no cartão
```

```
	int flag_cartao = 0; // Flag utilizada para gravar o cabeçalho somente uma
vez
	int flag_gravacao = 0;

	longint timer = 1; // Contador utilizado como uma espécie de relógio

	float cont = 0;

int a = 1;
// Buffer size used for the file copy process
//#ifndef CPY_BUFF_SIZE
//#define CPY_BUFF_SIZE       2048
//#endif

// String conversion macro
#define STR_(n)             #n
#define STR(n)              STR_(n)

// Drive number used for FatFs
#define DRIVE_NUM           0

constchar  inputfile[] = "fat:"STR(DRIVE_NUM)":input.csv";
constchar outputfile[] = "fat:"STR(DRIVE_NUM)":output.csv";
char flobuf[100] = "";

//unsigned char cpy_buff[CPY_BUFF_SIZE + 1];

SDSPI_Handle sdspiHandle;
SDSPI_Params sdspiParams;

// Variables for the CIO functions
FILE *src, *dst;

// Variables to keep track of the file copy progress
unsignedint bytesRead = 0;
unsignedint bytesWritten = 0;
unsignedint filesize;
unsignedint totalBytesCopied = 0;
int recordingEnabled = RECORDING_CLOSED, l =0;
char statusDSP = 'N', comms = 'D', FLAG_BUF0;

voidfour1(float data[], int nn, int isign)
{
int n, mmax, m, j, istep, i;
double wtemp, wr, wpr, wpi, wi, theta;
float tempr, tempi;

    n = nn << 1;
    j = 1;
for (i = 1; i < n; i += 2) {
            if (j > i) {
                  tempr = data[j];     data[j] = data[i];     data[i] = tempr;
                  tempr = data[j+1]; data[j+1] = data[i+1]; data[i+1] = tempr;
            }
            m = n >> 1;
            while (m >= 2 && j > m) {
                  j -= m;
```

```
                    m >>= 1;
                }
                j += m;
    }
    mmax = 2;
while (n > mmax) {
       istep = 2*mmax;
       theta = TWOPI/(isign*mmax);
       wtemp = sin(0.5*theta);
       wpr = -2.0*wtemp*wtemp;
       wpi = sin(theta);
       wr = 1.0;
       wi = 0.0;
       for (m = 1; m < mmax; m += 2) {
       for (i = m; i <= n; i += istep) {
                    j =i + mmax;
                    tempr = wr*data[j]   - wi*data[j+1];
                    tempi = wr*data[j+1] + wi*data[j];
                    data[j]   = data[i]   - tempr;
                    data[j+1] = data[i+1] - tempi;
                    data[i] += tempr;
                    data[i+1] += tempi;
       }
           wr = (wtemp = wr)*wpr - wi*wpi + wr;
           wi = wi*wpr + wtemp*wpi + wi;
       }
       mmax = istep;
    }

}

//---------------------------------------------------------------------------
// Função que realiza comunicação com MCU F28335 via SHAREDMEMORY
// > Recebe os dados escritos pelo F28335 na memória compartilhada, salva os
valores na forma complexa em um vetor auxiliar
// > Realiza o cálculo da FFT e sobreescreve os valores do vetor auxiliar
// > Salva a FFT cálculada no cartão de memória
//---------------------------------------------------------------------------

Void record_func(UArg arg0, UArg arg1)
{
       System_printf("Start the main loop\n");
while (true) {

       if(le_flag_C28[0] == 1)// Verifica se o C28 sinalizou o termino da aquisição
       {
       for (k = 0;k < 1024; k++) // Transfere os dados da memória compartilhada
para outro vetor, salvando na forma complexa utilizada pela função four1 (FFT)
       {
              medida_acel_01[2*k+1] = les0[k]; // Salva a parte real nas posições
impares do vetor
       medida_acel_01[2*k+2] = 0.0;     // Salva a parte imaginária nas posições
pares do vetor (0 pois é adquirido um valor real)

       medida_acel_02[2*k+1] = les1[k];
       medida_acel_02[2*k+2] = 0.0;

       medida_acel_03[2*k+1] = les2[k];
```

```
	medida_acel_03[2*k+2] = 0.0;

	medida_acel_04[2*k+1] = les3[k];
	medida_acel_04[2*k+2] = 0.0;
	}

	four1(medida_acel_01, 1024, 1); // Calcula a FFT utilizando o vetor auxiliar
já na forma complexa, sobrescreve o resultado no mesmo vetor
	four1(medida_acel_02, 1024, 1);
	four1(medida_acel_03, 1024, 1);
	four1(medida_acel_04, 1024, 1);

////////////////////////////////////////////////// SALVA DADOS NO CARTÃO MICROSD
                    System_printf("Iniciou operação\n");
                    System_flush();

              // Mount and register the SD Card
                    SDSPI_Params_init(&sdspiParams);
                    sdspiHandle = SDSPI_open(Board_SDSPI0, DRIVE_NUM,
&sdspiParams);
                    if (sdspiHandle == NULL)
                    {
                          System_abort("Error starting the SD card\n");
                    }
                    else {
                          System_printf("Drive %u is mounted\n", DRIVE_NUM);
                    }

                    // Try to open the source file
                    src = fopen(inputfile, "a");
                    if (!src)
                    {
                          System_printf("Creating a new file \"%s\"...",
inputfile);
                          System_flush();

                          // Open file for both reading and writing
                          src = fopen(inputfile, "a+");
                          if (!src)
                          {
                                System_printf("Error: \"%s\" could not be
created\n",
                                              inputfile);
                                System_abort("Aborting...\n");
                          }
                          System_sprintf(flobuf, "REALTÓRIO\n");
                          fwrite(flobuf, 1, strlen(flobuf), src);
                          fflush(src);

                          // Reset the internal file pointer
                          //rewind(src);

                    }

                    else
                    {
                          System_printf("Using existing copy of \"%s\"\n",
inputfile);
```

```
                    //Get the filesize of the source file
                    //fseek(dst, 0, SEEK_END);
                    //filesize = ftell(dst);
                    //rewind(dst);
                    }

                    // Create a new file object for the file copy
                    dst = fopen(outputfile, "a"); // "a" = comando append
                    if (!dst)
                    {
                          System_printf("Error opening \"%s\"\n", outputfile);
                          System_abort("Aborting...\n");
                    }
                    else
                    {
                          System_printf("Starting file copy\n");
                    }
                    if (flag_cartao == 0) // Verifica se já realizou a gravação do
cabeçalho
                    {
                          System_sprintf(flobuf, "%Acelerômetro 1:;;%Acelerômetro
2:;;%Acelerômetro 3:;;%Acelerômetro 4:\n"); // Escreve os dados do cabeçalho em 4
colunas
                          bytesWritten = fwrite(flobuf, 1, strlen(flobuf), dst);
                          fflush(dst);
                          System_sprintf(flobuf,
"%Real;%Imaginário;%Real;%Imaginário;%Real;%Imaginário;%Real;%Imaginário\n");
                          bytesWritten = fwrite(flobuf, 1, strlen(flobuf), dst);
                          fflush(dst);
                          flag_cartao = 1; // Seta a flag para 1
                    }
                    if (flag_gravacao == 1)
                    {
                          for (kk = 1; kk < 512; kk++) // Grava a metade dos dados
da fft, pois a fft apresenta a propriedade do espelhamento
                          {
                                System_sprintf(flobuf, "%f;%f;%f;%f;%f;%f;%f;%f
\n", medida_acel_01[2*kk+1], medida_acel_01[2*kk+2], medida_acel_02[2*kk+1],
medida_acel_02[2*kk+2], medida_acel_03[2*kk+1], medida_acel_03[2*kk+2],
medida_acel_04[2*kk+1], medida_acel_04[2*kk+2]); // Grava os dados em 8 colunas,
sendo a primeira para a parte real e a segunda para a parte imaginária e assim
sucessivamente
                                bytesWritten = fwrite(flobuf, 1, strlen(flobuf),
dst);
                                fflush(dst);
                          }
                          System_sprintf(flobuf, "%d;%d;%d;%d;%d;%d;%d;%d \n",
timer, timer, timer, timer, timer, timer, timer, timer); // Grava o contador de
tempo ao final de cada janela
                          bytesWritten = fwrite(flobuf, 1, strlen(flobuf), dst);
                          fflush(dst);
                    }
              // Close the file
                    fclose(src);
                    fclose(dst);
                    // Stopping the SDCard
                    SDSPI_close(sdspiHandle);
                    System_printf("Drive %u unmounted\n", DRIVE_NUM);
```

```
                    System_flush();
                    // statusDSP = 'O'; // forcei o statusDSP = 'O'
                    // comms = 'U'; // forcei o comms = 'U'

                    timer++;
                    flag_gravacao = 1;
////////////////////////////////////////////////////////////////////////////
       }
    }
Log_print0(Diags_USER1, "sleep");
        Task_sleep(200); // SLEEP por 200 clock ticks
//"ESTE FOI ALTERADO PARA NÃO INUNDAR A CPU E ASSIM NÃO EXTOURANDO OS
REGISTRADORES"->(não esta fazendo diferença)
Log_print0(Diags_USER1, "awake");
}//Task

//-----------------------------------------------------------------------------
// Código MAIN, incializa módulos GPIO, UART, ETHERNET-MAC e LIBERA PORTAS GPIO
//-----------------------------------------------------------------------------
intmain(void)
{
    Task_Handle taskHandle;
    Task_Params taskParams;
    Error_Block eb;
/* Set up the board specific items */
    TMDXDOCK28M36_initGeneral();
    TMDXDOCK28M36_initUART();
    TMDXDOCK28M36_initGPIO();
    TMDXDOCK28M36_initSDSPI();
    TMDXDOCK28M36_initUSB(TMDXDOCK28M36_USBDEVICE);
    TMDXDOCK28M36_initEMAC();
    System_printf("Demo with HTTP, I2C, and SD\nSystem provider is set to SysMin,
halt the target and use ROV to view output.\n");
/* SysMin will only print to the console when you call flush or exit */
    System_flush();
    USBCDCD_init();
    SerialInit();
    Error_init(&eb);
/* Create the Task that communicates to I2C temperature sensor. */
    Task_Params_init(&taskParams);
    taskParams.stackSize = 5096; //1280
    taskParams.priority = 10;
    taskParams.instance->name = "Record";
    taskHandle = Task_create(record_func, &taskParams, &eb);
if (taskHandle == NULL) {
        System_printf("Task_create() failed!\n");
        BIOS_exit(0);
    }
/*=============================================================================*/
GPIO_write(Board_LED0, Board_LED_ON);

    System_printf("Starting the FatSD example\n");
System_flush();
/*=============================================================================*/
// Cortex -M3 CPU tem domínio sobre GPIOs do Concerto, portanto devemos designar
para o DSP quais
// serão usadas pelo Cortex-M3 e quais serão usadas pelo F28335.
```

```
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
//SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK); Precidicial ao Ehternet
//SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL); Prejudicial ao Ethernet
//SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM); Prejudicial ao Ethernet
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOQ);
//SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOR); Prejudicial ao cartão SD
SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOS);
//não sabemos o que faz mas a princípio não influencia
//GPIOPadConfigSet(GPIO_PORTA_BASE, 0xFF, GPIO_PIN_TYPE_STD_WPU);
//GPIOPadConfigSet(GPIO_PORTB_BASE, 0xFF, GPIO_PIN_TYPE_STD_WPU);
// Give C28 control of all GPIOs
GPIOPinConfigureCoreSelect(GPIO_PORTA_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTB_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTC_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTD_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTE_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTF_BASE, 0xFF, GPIO_PIN_M_CORE_SELECT);
//Nucleo M3 controla GPIO's da porta F
GPIOPinConfigureCoreSelect(GPIO_PORTG_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTH_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTJ_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
//GPIOPinConfigureCoreSelect(GPIO_PORTK_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
Prejudicial ao Ethernet
//GPIOPinConfigureCoreSelect(GPIO_PORTL_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
Prejudicial ao Ethernet
//GPIOPinConfigureCoreSelect(GPIO_PORTM_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
Prejudicial ao Ethernet
GPIOPinConfigureCoreSelect(GPIO_PORTN_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTP_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
GPIOPinConfigureCoreSelect(GPIO_PORTQ_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
//GPIOPinConfigureCoreSelect(GPIO_PORTR_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);
Prejudicial ao cartão SD
GPIOPinConfigureCoreSelect(GPIO_PORTS_BASE, 0xFF, GPIO_PIN_C_CORE_SELECT);

//O ARM tem acesso a escrita como padrão. Esse comando transfere o acesso de
escrita para o C28 para as seguintes sessões da memória compartilhada
RAMMReqSharedMemAccess((S0_ACCESS | S1_ACCESS | S2_ACCESS | S3_ACCESS | S4_ACCESS
| S5_ACCESS | S6_ACCESS),1);

/* Start BIOS. Will not return from this call. */

    statusDSP = 'O';
    BIOS_start();
return (0);
}
```