**UNIVERSIDADE FEDERAL DE SANTA MARIA**
**CENTRO DE TECNOLOGIA**
**CURSO DE ENGENHARIA DE COMPUTAÇÃO**

# SENSOREAMENTO E CONTROLE DE CONVERSORES ESTÁTICOS COM A UTILIZAÇÃO DE FPGA

**TRABALHO DE CONCLUSÃO DE CURSO**

**Leonardo Lima Carvalho**

**Santa Maria, RS, Brasil**
**2016**

# SENSOREAMENTO E CONTROLE DE CONVERSORES ESTÁTICOS COM A ULTILIZAÇÃO DE FPGA

**Leonardo Lima Carvalho**

Trabalho de Conclusão de Curso apresentado ao Curso de Engenharia de Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de
**Engenheiro de Computação**

**Orientador: Prof. Dr. José Eduardo Baggio**

**Santa Maria, RS, Brasil**
**2016**

**Universidade Federal de Santa Maria**
**Centro de Tecnologia**
**Graduação no Curso de Engenharia de Computação**

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

**SENSOREAMENTO E CONTROLE DE CONVERSORES ESTÁTICOS COM A ULTILIZAÇÃO DE FPGA**

elaborado por
**Leonardo Lima Carvalho**

como requisito parcial para obtenção do grau de
**Engenheiro de Computação**

**COMISSÃO EXAMINADORA:**

_____
**José Eduardo Baggio, Dr.**
(Orientador)

_____
**Cesar Augusto Prior, Dr.** (UFSM)

_____
**Carlos Henrique Barriquello, Dr.** (UFSM)

Santa Maria 22, Dezembro de 2016

# AGRADECIMENTOS

Aos meus pais, que sempre apoiaram e incentivaram ao estudo dos seus dois filhos.

Ao professor José Eduardo Baggio pela persistência, desempenho, conhecimento transmitido e orientação durante a trajetória acadêmica e desenvolvimento do projeto.

A todos os professores e principalmente o coordenador do curso, que foi um grande transmissor de aprendizagem. Meu muito obrigado a todos.

Ao Fernando Carvalho, meu irmão, amigo, companheiro que também é estudante, incentivador de aprendizagem e cultura.

A Kaciusse Lourenço, minha namorada que esteve desde o início do curso ao meu lado e sempre teve paciência e me apoiou nos momentos difíceis.

A todos os colegas de jornada acadêmica, grandes amigos que levarei para toda a vida.

“*Se a vida fosse fácil eu não nascia chorando”*
(Hungria)

# RESUMO


Trabalho de Conclusão de Curso
Curso de Graduação em Engenharia de Computação
Universidade Federal de Santa Maria

**SENSOREAMENTO E CONTROLE DE CONVERSORES ESTÁTICOS COM A ULTILIZAÇÃO DE FPGA**

AUTOR: LEONARDO LIMA CARVALHO
ORIENTADOR: JOSÉ EDUARDO BAGGIO

Local e data da defesa: Santa Maria, 22 de Dezembro de 2016.

O trabalho apresentado a seguir busca implementar medições de tensão e corrente em conversores estáticos de potência e implementar uma lei de controle utilizando VHDL e implementando-a em uma FPGA. Para estudo de caso, foi considerado um conversor elevador (boost).

A utilização desse projeto é uma alternativa ao uso de DSPs (Digital Signal Processor) e microcontroladores, que possuem menor velocidade de operação, e consequentemente menor resposta dinâmica.

Um transdutor de corrente é acoplado no conversor Boost, convertendo a corrente lida em um sinal equivalente em tensão, a qual é direcionada a um conversor A/D (analógico/digital), para que eles possam ser lidos pela plataforma embarcada de circuito digital programável FPGA (Field Programmable Gate Array). Foi escolhido o uso de FPGA, por possuir grande versatilidade de implementação de sistemas digitais complexos e um baixo consumo de energia com alta velocidade de processamento e rápidos tempos de atualização.

O sistema baseia-se no uso de dois controladores PI (Proporcional Integral) implementados no FPGA para o ajuste de corrente média e tensão na saída do Boost, sendo os valores corrigidos e atualizados conforme o *clock* do sistema.

Detalhes sobre o projeto, o sistema, o desenvolvimento com o FPGA e os testes realizados são apresentados ao longo do desenvolvimento. Os resultados obtidos são apresentados através dos ambientes de desenvolvimento MATLAB/PID Tuner® e ISE®.



**Palavras-chave:** Sensor; Conversor Estático; FPGA; Controlador PI; DSPs..

# ABSTRACT

Thesis - Final Course Project
Computer Engineering Course
Federal University of Santa Maria

**SENSOREAMENTO E CONTROLE DE CONVERSORES ESTÁTICOS COM A ULTILIZAÇÃO DE FPGA**

AUTHOR: LEONARDO LIMA CARVALHO
SUPERVISOR: JOSÉ EDUARDO BAGGIO

Defense place and date: Santa Maria, December 22nd, 2016.

The work introduced next looks to implement voltage and current measures on static power converters and implement a control law using VHDL and implementing it using FPGA. For case study, it was considered a elevator converter (boost).

The use of this project is an alternative to the use of DSPs (Digital Signal Processor) and microcontrollers, which have lower speed of operation, and consequently less dynamic response.

A current transducer is linked in the Boost converter, converting the current read in an voltage equivalent signal, which is directed to an A/D converter (analog / digital), for them can be read by the boarded platform of programmable digital circuit FPGA (Field Progammable Gate Array ). It was chosen the use of FPGA, because it has great versatility of implementation in complex digital systems and a low power consumption with high processing speed and fast update times.

The system is based on the use of two PI controllers (Proportional Integral) implemented in the FPGA for the adjustment of average current and voltage in the Boost output, the values being corrected and updated according the *clock* of the system.

Details about the project, system, development with the FPGA and the tests performed are presented throughout the development. The results obtained are presented through the MATLAB / PID Tuner® and ISE® development environments.

**Keywords:** Sensor; Static Converter; FPGA; PI Controller; DSPs.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ANEXOS

## LISTA DE ABREVIATURAS E SIGLAS

| | |
|---|---|
| A/D | Analógico Digital |
| CI | Circuito Impresso |
| CLB | Blocos Lógicos Configuráveis (*Configurable Logic Blocks*) |
| CPLD | Dispositivos Lógicos Programáveis Complexos (*Complex Programmable Logic Device*) |
| D | Razão Cíclica |
| DC | Corrente Contínua (*Direct Current*) |
| DSP | Processador digital de sinais (*Digital Signal Processor*) |
| FPGA | (F*ield Programmable Gate Array*) |
| $I_D$ | Corrente no diodo |
| $I_L$ | Corrente no indutor |
| IOB | Blocos de entrada e saída (*Input Output Blocks)* |
| $I_S$ | Corrente na chave |
| $K_1$ | Ganho proporcional |
| $K_2$ | Ganho integral |
| KHz | Kilohertz - Frequência |
| LTK | Lei das Tensão de kirchhoff |
| PI | Proporcional Integral |
| PLD | Dispositivos Lógicos Programáveis (*Programmable Logic Device*) |
| PWM | Modulação por largura de pulso (*Pulse Width Modulation*) |
| S | Chave do conversor - Transistor |
| SPLD | Dispositivos Lógicos Programáveis Simples (*Simple Programmable Logic Device*) |
| $T_{OFF}$ | Período da chave aberta |
| $T_{ON}$ | Período da chave fechada |
| $T_S$ | Período de chaveamento |

# SUMÁRIO

# 1 INTRODUÇÃO

O uso de conversores estáticos para se fazer adequações de tensões e correntes está presente em praticamente todos os equipamentos eletrônicos da atualidade. Dentre todos tipos de conversores, o conversor elevador de tensão, comumente chamado de conversor *Boost* é um dos mais utilizados devido à sua simplicidade, e por permitir um alto fator de potência.

Levando-se em conta a popularidade do conversor *Boost*, resolveu-se adotá-lo para estudo e desenvolvimento do seu controle discreto implementado em FPGA, também desenvolvido em outros trabalhos [BEZERRA, 2010] .

Um dos objetivos desse projeto é modelar um controle eficaz para garantir a regulação da tensão de saída do conversor. Nos dispositivos eletroeletrônicos, a possibilidade da tensão ultrapassar os valores especificados causa perturbações durante a operação ou até mesmo avarias de elementos eletrônicos específicos e como consequência o mau funcionamento do equipamento.

Uma técnica bastante utilizada na indústria para o controle do conversor *Boost* é o projeto de controladores digitais PI (Proporcional Integral) que permite um erro de regime permanente nulo, conhecido na literatura por [DORF, BISHOP 2001].

Para este estudo foram especificados parâmetro de entrada e saída desejados, e foi considerado um conversor *Boost* previamente projetado, no qual se deseja desenvolver um controlador digital de alta velocidade.

Os parâmetros calculados para o controlador foram validados em simulação no software MATLAB / PID *Tuner* e implementados no FPGA.

A fim de acelerar dados de processamento, a vantagem da computação utilizando FPGA Nexys 2 permite a implementação de algoritmos de controle em paralelo, que também ocorre em paralelo com os processos de medição, geração do PWM (*Pulse Width Modulation*), filtragem de tensão e corrente.

A topologia implementada baseia-se em um conversor com uma tensão de entrada de 300V e tensão contínua de saída DC de 400V, corrente média de 1,67A e uma frequência de comutação de carga nominal de 100 kHz.

Assim utilizamos os parâmetros citados para adequação do sistema, sendo possível executar algoritmos de controle de tempo discreto em períodos de tempo abaixo de 10us (um período de chaveamento).

O projeto é composto por cinco seções que mostram como foi feito o seu desenvolvimento.

Na seção 2 é feito o estudo do modelo básico do conversor e suas características.

Na seção 3 é apresentado o projeto de cada um dos controladores com referência em outros autores que projetaram compensadores de corrente e tensão na saída DC-DC.

Na seção 4 será mostrada a estrutura montada para realizar a aquisição dos dados no FPGA, onde é visto os componentes como: placa de circuito impresso, o sensor de corrente e o conversor A/D (analógico/digital).

Na seção 5 os resultados obtidos na simulação do conversor em *Matlab* e *Ise Xilinx/ ISim* serão apresentados com a atuação dos controladores.

Por fim, é feita a conclusão do projeto e sugerida uma visão futura para um possível aperfeiçoamento do sistema.

# 2 REVISÃO BIBLIOGRÁFICA

## 2.1 Modelo de Referência do Conversor Boost

O Boost é conhecido na literatura [BARBI, MARTINS 2000] como conversor CC-CC (corrente contínua) elevador de tensão, uma das suas caraterísticas é ter a forma de onda da tensão de saída igual a da corrente de entrada. Na Figura 1 é apresentado a configuração básica do conversor.

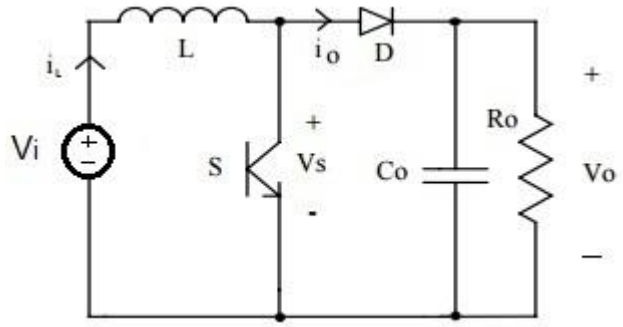



Figura 1 - Conversor elevador de tensão com entrada CC

O conversor opera com uma frequência de comutação que tem relação com o período de duração $T_s$ na qual o componente *S* (transistor) faz o chaveamento, assim ele fica alternando entre os estados de chave fechada $T_{on}$ e chave aberta $T_{off}$, ou seja, a razão cíclica *D* da chave, apresentando a típica forma de onda na tensão de saída conforme a Figura 2.

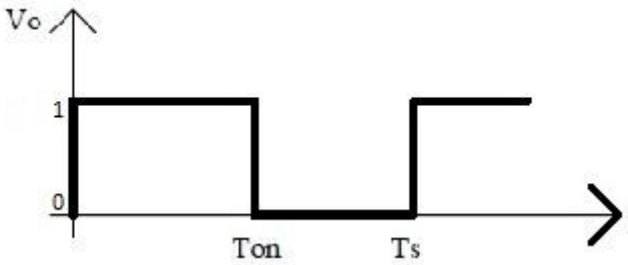



Figura 2 - Forma de onda de tensão de saída

A corrente que passa pelo indutor $I_L$ descreve o modo de operação do conversor, podendo ser condução contínua ou descontínua. Para fins de simulação

a escolha do modo de operação será em corrente contínua, pois apresenta distorção mínima tornando o sistema mais estável.

A seguir veremos as etapas de funcionamento e como é feita a transferência da tensão de entrada para a saída.

### 2.1.1 Primeira etapa $T_{on}$

No primeiro momento a chave *S* está conduzindo corrente e o indutor *L* recebe a energia fornecida pela fonte de entrada $V_i$, mostrada na Figura 3.

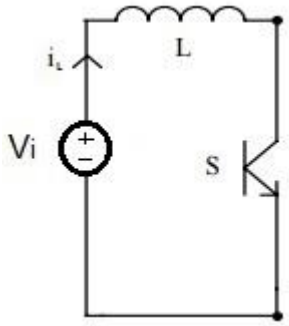



Figura 3 - Chave S conduzindo Ton

O restante do sistema vê a chave como curto circuito, por isso a corrente passa pelo caminho livre e o diodo da Figura 1 encontra-se reversamente polarizado. O tempo que a chave fica aberta ou fechada irá ser implementado através do cálculo da razão cíclica *D* que controla a tensão da saída na seção *4.2.2.2*.

### 2.1.2 Segunda etapa $T_{off}$

A chave S do conversor não está conduzindo corrente, então a energia que antes o indutor *L* havia armazenado é transferida para o capacitor $C_o$ mostrado na Figura 4.

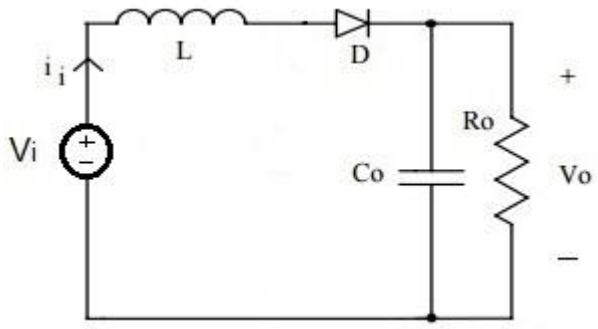



Figura 4 - Chave S não está conduzindo Toff

Para que o capacitor receba a energia da fonte $V_i$ e do indutor, o diodo $D$ precisa estar polarizado fazendo com que a carga $R_o$ seja alimentada pelo capacitor, assim ele mantém a energia na saída $V_o$ até que aconteça o ciclo, retornando para a primeira etapa.

## 2.2 Principais formas de onda

As principais formas de onda para a operação ideal do conversor no modo de condução contínua são mostradas na Figura 5.

A corrente no indutor $I_L$ cresce e decresce linearmente conforme o período de chaveamento, mas flui continuamente.

A corrente na chave $I_s$ e no diodo $I_D$ não são contínuas, pois quando um está conduzindo o outro não conduz repetindo essas características conforme o ciclo de chaveamento.

A tensão de saída do capacitor $VC_o$ é praticamente constante, visto que a tensão máxima no interruptor é igual a tensão de saída $V_o$ do conversor.

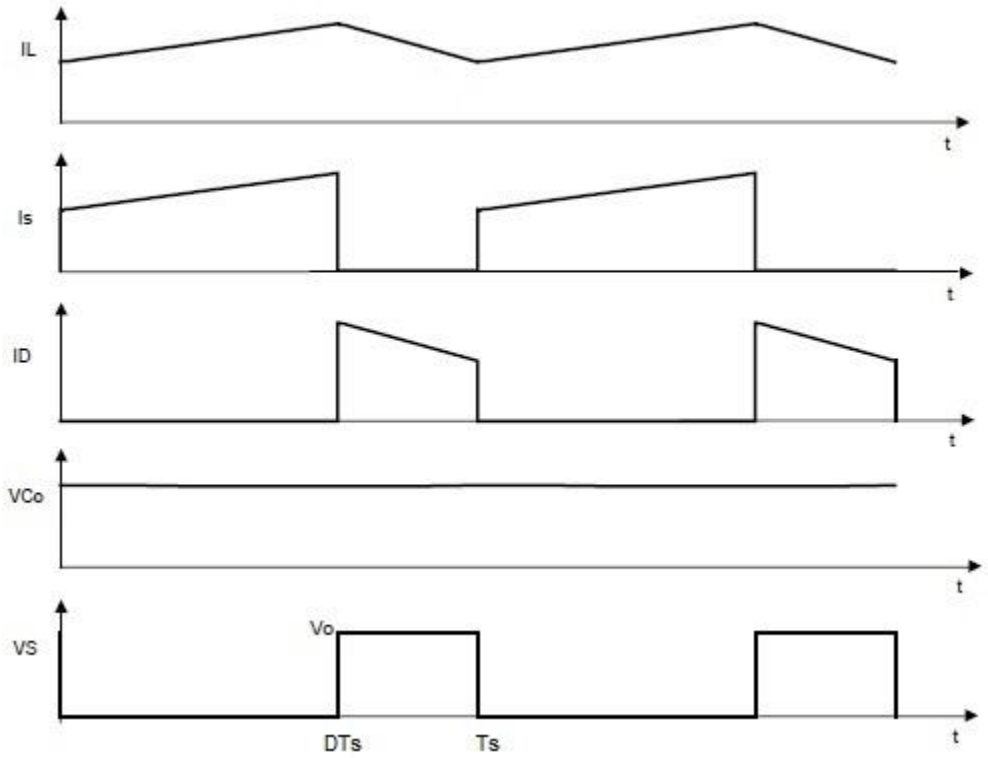



Figura 5 - Principais formas de onda do conversor Boost
Fonte: Adaptado de (Rashid, 1999)

## 2.3 Modelo de Referência para o Controle do conversor

O controle do conversor boost tem como objetivo regular a tensão contínua de saída, ou seja, garantir a estabilidade do sistema. Para que essa característica seja válida, a corrente de entrada deve ter uma forma de onda proporcional à tensão de entrada. O projeto utilizado para fazer o controle consiste em uma realimentação que comparada com uma referência, gera um erro que será a entrada do compensador o qual representado irá atuar para minimizar o erro de entrada. O resultado do compensador é apresentado pela variação da razão cíclica do PWM.

Na Figura 6 é apresentado o diagrama de blocos do controle do conversor onde $G_c(s)$ é o compensador que age na planta $G(s)$ e $H(s)$ representa a realimentação do sistema.

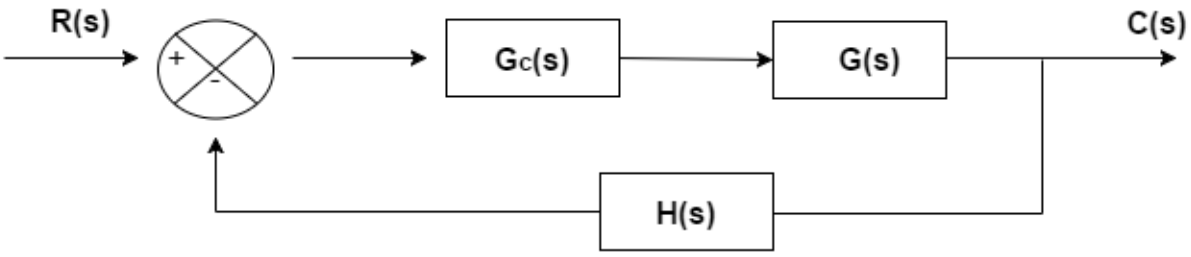



Figura 6 – Compensador em sistema de malha fechada
Fonte: (Ogata, 2000)

Esse modelo de controlador pode ser configurado na forma digital ou analógica, o método analógico escolhido é o proporcional integral (PI), que irá ser discretizado pelo software Matlab e implementado na forma digital no FPGA.

O método consiste na análise de duas malhas para o controle do conversor: a malha interna de corrente que controla a forma de onda da corrente que passa pelo indutor e a malha externa de tensão que regula a tensão de saída.

### 2.3.1 Compensador proporcional integral (PI)

O compensador proporcional integral atua a partir da combinação de ações: proporcional e integral intrínseco ao seu nome, atuando na redução do erro estacionário. Na Figura 7 é mostrado o gráfico da resposta característica do PI.

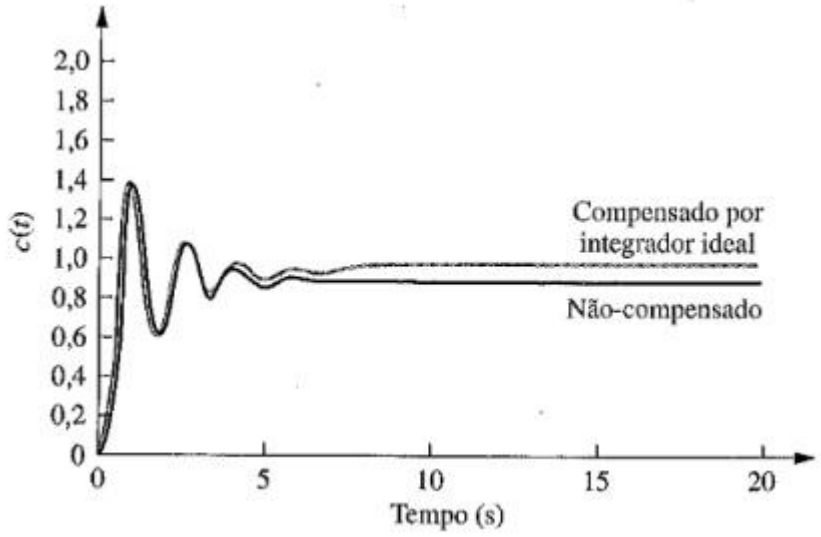



Figura 7 - Resposta característica do compensador PI
Fonte: Nise (2002)

Segundo [NISE, 2002] a Figura 7 mostra o diagrama do compensador clássico em malha fechada relacionando em um conjunto os dois elementos proporcional e integral, a fim de analisar o comportamento transitório e em regime permanente do sistema a ser controlado.

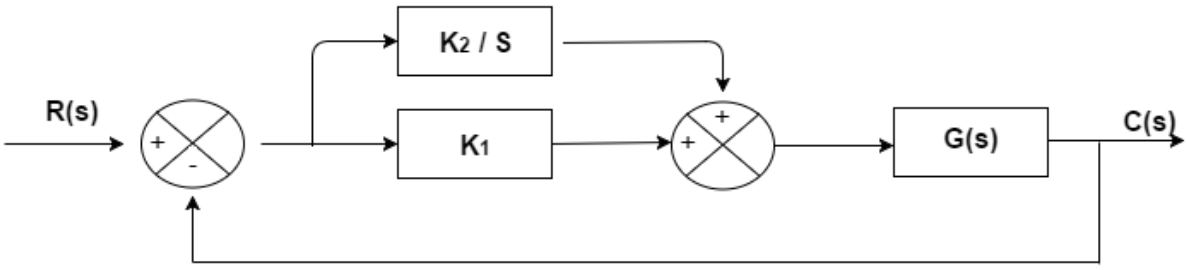



Figura 8 - Diagrama de blocos do compensador PI
Fonte: Nise (2002)

Os dois parâmetros relacionados do controlador são: ganho proporcional $K_1$ que elimina os ruídos do sistema, e termo integral $K_2$ que é responsável por eliminar o erro de *off-set*. O equacionamento do ganho do compensador com seus parâmetros é apresentado na Equação 1 [NISE, 2002].

$$G_C(s) = K_1 + \frac{K_2}{s} = \frac{K_1\left(s + \frac{K_2}{K_1}\right)}{s} \quad (1)$$

### 2.3.2 Discretização do controlador

O controle do conversor apresenta-se na forma analógica até o momento, e para poder ser implementado em sistemas digitais é preciso discretizar o sistema. O método de integração de Euler em atraso resolve a transformação. A Equação 2 mostra a transformação de *S* para *Z* conforme o período de chaveamento *Ts* método usual para discretização [BUSO, 2006].

$$S = \frac{z - 1}{z.T_s} \quad (2)$$

A Equação (1) pode ser apresentada da forma:

$$G_c(s) = K_2 \frac{\left(1 + \frac{K_1}{K_2}s\right)}{s} \quad (3)$$

Substituindo a Equação (2) na (3) chegamos ao seguinte resultado [BUSO, 2006]:

$$G_c(z) = K_2 \frac{\left(1 + \frac{K_1}{K_2}\frac{z-1}{zT_s}\right)}{\frac{z-1}{zT_s}} = \frac{(K_1 + K_2T_s)z - K_1}{z - 1} = K_1 + K_2T_s\frac{z}{z-1} \quad (4)$$

O diagrama de blocos do controlador PI é apresentado na Figura 9 e a partir deste é apresentado a equação das diferenças que representa o controle do boost.

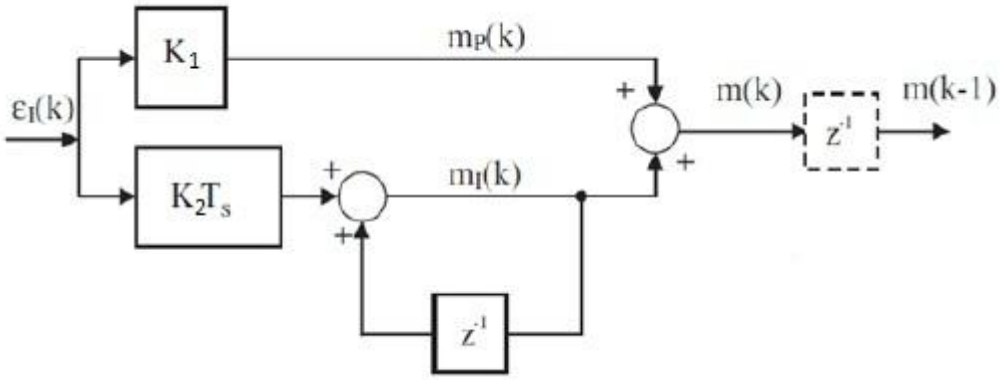



Figura 9 - Modelo discretizado do compensador PI
Fonte: Adaptada Buso (2006)

Fazendo a análise da Figura 9 chagamos na Equação 5, a qual será implementada no FPGA para fazer o controle digital do conversor.

$$U[n] = K_p.e[n] + K_I T_s.e[n] + K_I T_s.e[n-1] \quad (5)$$

Com a função discretizada a partir da análise da equação das diferenças, podemos então implementá-la no dispositivo lógico programável . A seguir veremos a escolha do dispositivo que irá fazer o controle digital do conversor.

## 2.4 DISPOSITIVOS LÓGICOS PROGRAMÁVEIS

Os Dispositivos Lógicos Programáveis (Programmable Logic Device - PLD) são circuitos integrados utilizados para implementar funções lógicas. Sua principal característica é a grande flexibilidade de configuração e reconfiguração que o usuário pode fazer.

Os PLDs utilizam na sua arquitetura microcontroladores e circuitos integrados de aplicações específicas, podendo ser divididos conforme sua arquitetura em: CPLD (*Complex Programmabel Logic Devices*), SPLD (*Simple Programmable Logic Device*) e FPGA (*Field Programmable Gate Array*).

Neste trabalho é escolhido o FPGA para desenvolvimento da programação, e na seção seguinte é apresentada sua arquitetura.

### 2.4.1 FPGA

O FPGA foi desenvolvido pela Xilinx em 1984 para fins militares [7], e hoje em dia é bastante utilizado na indústria automotiva por ser um dispositivo de manipulação de hardware [8]. Sua arquitetura é formada por um grande arranjo de blocos lógicos ou células configuráveis de entrada e saída contidos em um único circuito integrado, o qual é apresentado na Figura 10.

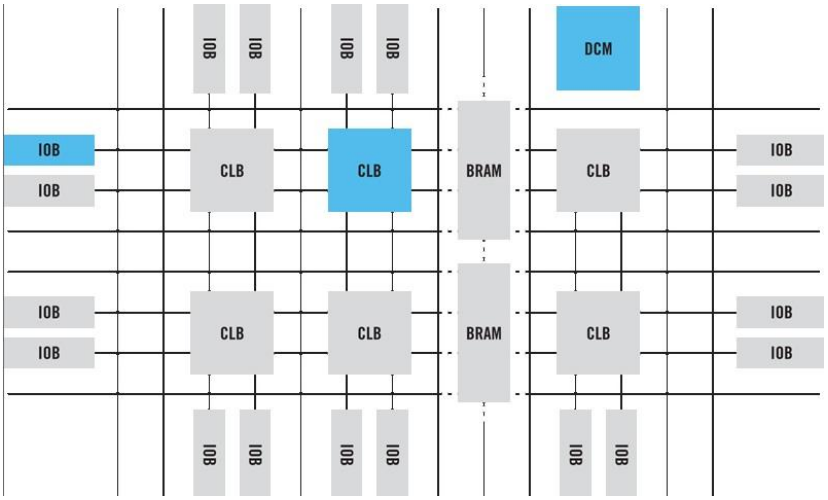



Figura 10 - Arquitetura de um FPGA
Fonte: https://agetechnology.wordpress.com/category/altera

Cada parte que compõe o FPGA segundo [8] tem uma função específica que é descrita a seguir.

*IOB* (*Input Output Blocks*) células ou blocos que podem ser configurados como entrada ou saída de dados.

*CLB* (*Configurable Logic Blocks*) blocos lógicos ligados aos recursos de conexões que em conjunto fazem a comunicação de todo o circuito integrado.

Os FPGAs contém na sua arquitetura básica blocos para armazenamento de programação, memória e também DCM (*Digital Clock Management*) que são ciclos de relógio.

A implementação neste tipo de dispositivo se faz pelas linguagens de programação VHDL 2008 (utilizada no trabalho) ou Verilog, elas descrevem comportamento do projeto físico do hardware. Conhecido o dispositivo que irá manipular os dados, na próxima seção será apresentada a estrutura física que é acoplada no conversor para obter a aquisição dos dados de corrente e tensão.

Também é apresentado os códigos do Matlab e Xilinx/Ise que descrevem o comportamento do conversor com a atuação dos controladores.

# 3 PROJETO E SIMULAÇÃO

Para fazer o controle digital da chave no conversor apresentaremos a implementação e a simulação do código VHDL, estruturado através dos resultados gerados no Matlab.

Será mostrado como foi montada a estrutura para aquisição de valores e análise do comportamento do conversor Boost, o resultado da simulação do controlador PI com valores obtidos através do software *Matlab/Simulink* atuando na malha de corrente e tensão.

## 3.1 Análise e simulação do controle do conversor boost

Para obtermos os valores do controle PI, primeiramente temos que fazer a simulação de um conversor boost, o qual temos seus parâmetros previamente estabelecidos mostrados na Tabela 1. O software escolhido que simula a atuação do controle é o Matlab/Simulink e após obtermos a função de controle com os valores discretizados será implementado no dispositivo FPGA, para que este atue supervisionando e ajustando a tensão na saída do conversor conforme o valor escolhido pelo usuário.

Tabela 1 - Valores dos parâmetros do conversor boost

| Símbolo | Significado | Valor |
|---|---|---|
| $V_i$ | Tensão de entrada | 300V |
| $V_o$ | Tensão de saída | 400V |
| $P_o$ | Potência de saída | 500W |
| $f_s$ | Frequência de comutação | 100KHz |
| $L$ | Indutor | 4000uH |
| $C$ | Capacitor | 200uF |
| $R$ | Resistência | 320Ω |

O *boost* tem seus parâmetros de funcionamentos já definidos, ou seja, não faz-se necessário calcular os valores de cada elemento que o compõe.

Para o projeto do controle do *boost*, são usadas duas malhas de controle: uma malha externa para a supervisão da tensão e uma malha interna para o controle da corrente, onde cada uma tem seu compensador projetado no Matlab/PID *Tuner*. Na Figura 11 é apresentado o controle completo da malha de corrente e malha de tensão atuando juntos.

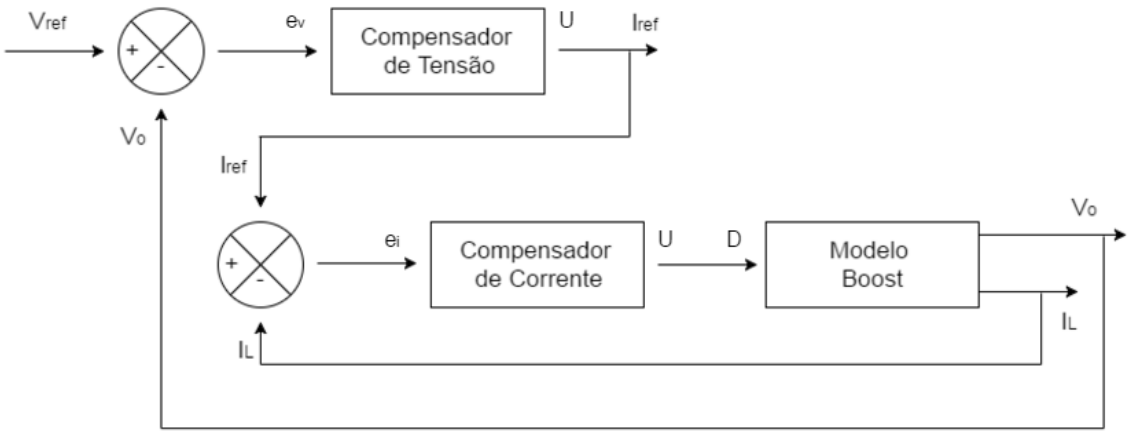



Figura 11 - Controle completo do boost com a atuação dos compensadores

O típico controle discreto em malha fechada mostrado na Figura 11 tem parâmetros definidos como:

$V_{ref}$: Tensão de referência

$I_{ref}$: Corrente de referência

$e_v$: Erro de tensão

$e_i$: Erro de corrente

U: Saída do controlador

$I_L$: Corrente no indutor

$V_o$: Tensão de saída

D: Razão cíclica

Agora identificado os elementos de controle e atuação no conversor boost, iremos definir as funções de transferências de corrente e tensão que irão gerar os valores analógicos de controle.

### 3.1.1 Modelo para controle analógico do boost

#### *3.1.1.1 Malha de controle da corrente no indutor*

Considerando que a tensão de entrada não influencie na função de transferência em malha fechada na corrente do conversor, podemos ter o circuito equivalente do boost mostrado na Figura 12.

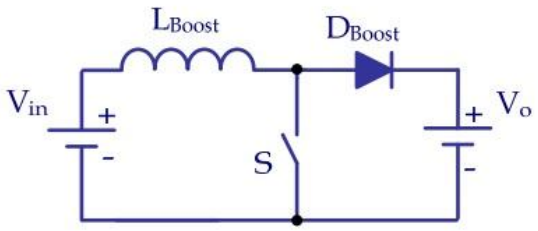



Figura 12 - Modelo equivalente do conversor boost
Fonte: (Jappe, 2009)

A corrente que circula pelo indutor $I_L$ tem relação com a razão cíclica da chave *S*, tendo em vista estudos que já fazem essa relação, a Equação 6 [JAPPE, 2009] mostra a função de transferência que determina a malha de controle de corrente do boost.

$$\frac{I_L(s)}{D(s)} = G_{ID}(s) = \frac{V_o}{s.L} \tag{6}$$

Podemos ressaltar que esse modelo de função de transferência quando usada em altas frequências não compromete a operação do conversor [JAPPE, 2009].

Definido o modelo da função de transferência da planta de corrente contínua, a Equação 9 será aplicada no código do Matlab.

#### *3.1.1.2 Malha de controle da tensão de saída*

Já visto o modelo que relaciona a corrente do indutor com a razão cíclica, agora para controlar a tensão de saída $V_o$ temos que relacionar $I_L$ com $V_o$, pois é a corrente que passa pelo indutor que realizada a transferência de energia da fonte $V_{in}$

para a carga $R_o$, controlando o valor médio de $I_L$ é possível controlar a tensão de saída. A Figura 13 mostra o circuito equivalente do boost.

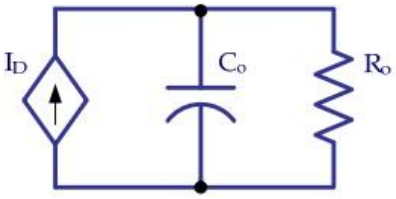



Figura 13 - Circuito equivalente para determinar a função de transferência da malha de tensão
Fonte: (Jappe, 2009)

No circuito da Figura 13 a fonte de corrente $I_D$ é a representação da corrente média que passa pelo diodo. A função de transferência da Equação 7 é o resultado final da análise da relação da tensão de saída com a corrente no indutor [JAPPE, 2009].

$$\frac{V_o(s)}{I_L(s)} = G_{VI}(s) = \frac{R_o}{R_o C_o s + 1} \tag{7}$$

Também apresentamos na Equação 7 o modelo da função de transferência da planta de tensão contínua no tempo para aplicar no código do Matlab.

Na próxima seção é apresentada a interface do software Matlab para a escolha do controlador de acordo com os requisitos do usuário.

### 3.1.2 Projeto dos compensadores

O método escolhido para fazer o controle da corrente no indutor e da tensão na saída do boost é o controle PI. A partir das funções de transferência da malha de corrente apresentada na Equação 6 e da malha de tensão na Equação 7, a simulação é feita através do software Matlab/PID Tuner. Na Figura 14 podemos ver a interface do PID Tuner. Importando as funções de transferência podemos escolher a curva manipulando o tempo de resposta e o comportamento do transiente de acordo com os critérios do usuário.

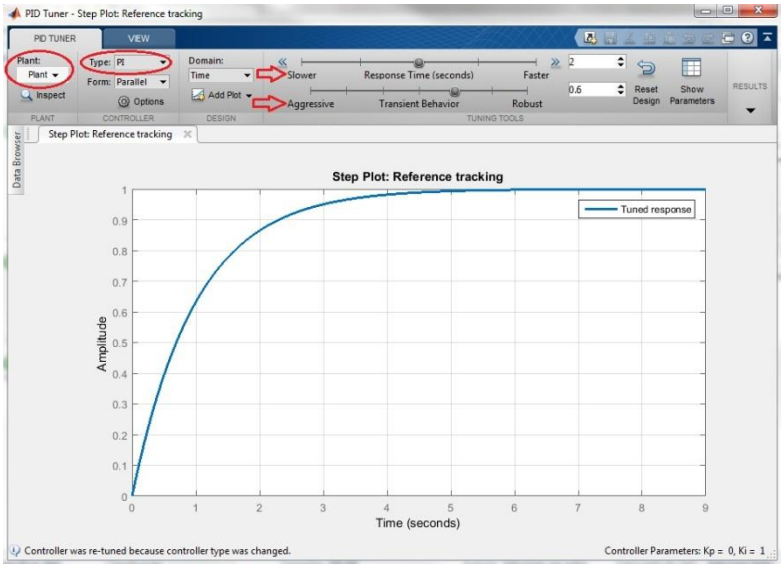



Figura 14 - Interface do PID Tuner

A curva característica do controle PI no software de simulação é apresentada na Figura 14. Após vermos as funções de transferências contínuas na *seção 4.2.1* iremos aplicar a Equação 6 e a Equação 7 para encontrar o melhor ajuste do controle da tensão de saída e da corrente do sistema respectivamente.

### *3.1.2.1 Compensador de tensão*

O resultado do gráfico escolhido através da função de transferência do compensador de tensão é apresentado na Figura 15.

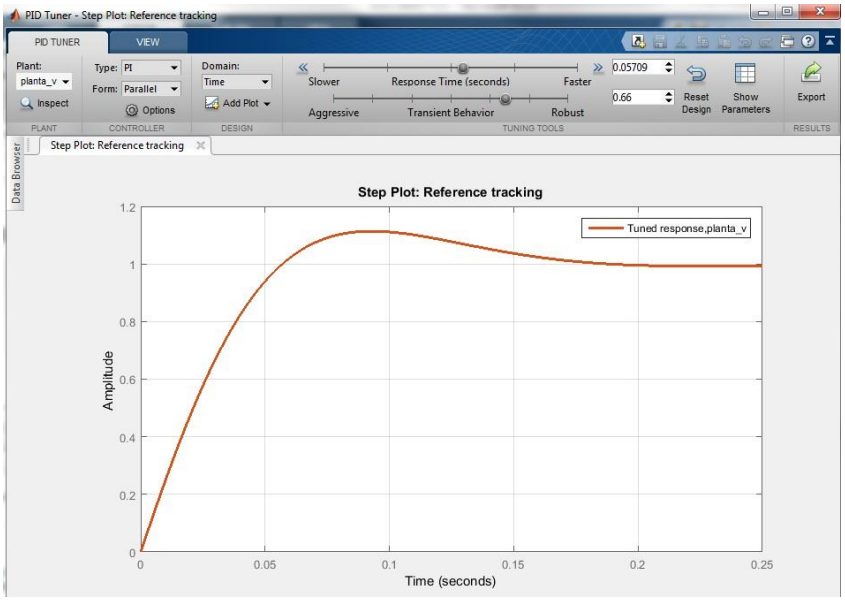



Figura 15 - Gráfico escolhido para o compensador de tensão

Uma vez definida a resposta desejada, a ferramenta *PID Tuner* exporta a função de transferência do compensador para a área de trabalho do MatLab®. Para se obter a função de transferência discreta, utiliza-se a função *c2d* no Matlab, que transforma o resultado no tempo contínuo para discreto. Efetuando-se estes procedimentos, obteve-se a seguinte função de transferência discretizada:

$$\frac{U(s)}{e_v(s)} = \frac{0{,}03506z - 0{,}03505}{z - 1} \quad (8)$$

Manipulando a Equação 14 de acordo com a Equação 5 que representa a equação a diferenças do controlador PI temos a seguinte resposta:

$$U[k] = U[k-1] + 0{,}03506.e[k] - 0{,}03505.e[k-1] \quad (9)$$

Analisando a Figura 11 na malha do controle da tensão, é a saída *U(s)* que irá gerar a corrente de referência $I_{ref}$ para o comparador do controle da corrente. O valor da entrada do compensador $e_v$ é calculado pela diferença da tensão de leitura do conversor $V_o$ e da tensão de referência $V_{ref}$. A Equação 10 mostra o resultado da corrente de referência.

$$Iref[k] = Iref[k-1] + 0{,}03506.e_v[k] - 0{,}03505.e_v[k-1] \quad (10)$$

Apresentamos a lei de controle que gera a corrente de referência. A Equação 10 discretizada será implementada no dispositivo FPGA para o cálculo do erro de corrente $e_i$ utilizado na próxima etapa do compensador de corrente.

### *3.1.2.2 Compensador de corrente*

Do mesmo modo que obteve-se o gráfico do compensador de tensão e a função de transferência discretizada, foi feito a análise do controle da corrente para atuar no conversor.

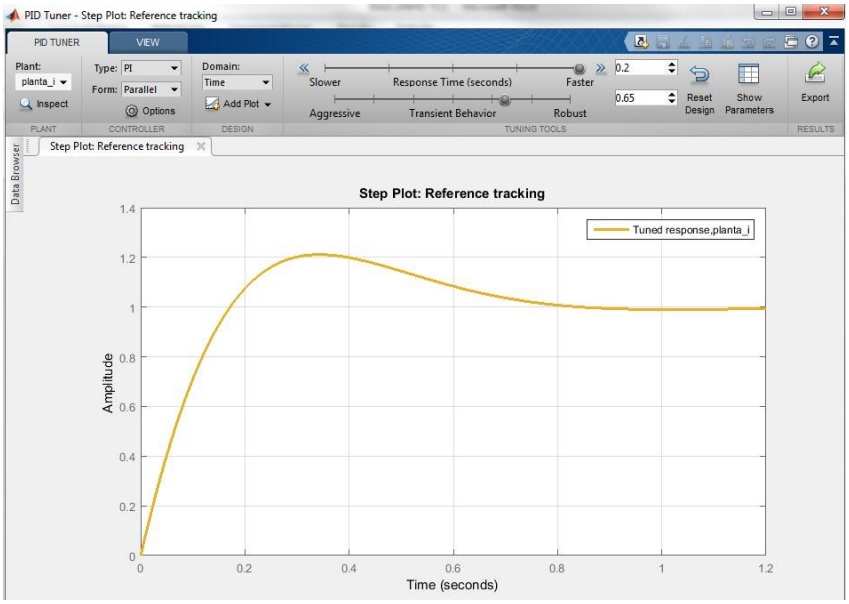



Figura 16 - Gráfico escolhido para o compensador de corrente

O resultado discretizado da função de transferência do compensador de corrente é apresentado na Equação 11.

$$\frac{U(s)}{e(s)} = \frac{0{,}7942z - 0{,}7832}{z - 1} \tag{11}$$

Na equação a diferenças do controlador PI temos seguinte resultado:

(12)

$$U[k] = U[k-1] + 0{,}7942.e[k] - 0{,}7832.e[k-1]$$

Analisando a Figura 11 na malha do controle da corrente, é a saída *U(s)* que nada mais é que a razão cíclica *D*, sinal para atuar no *boost* fazendo o controle da tensão. O valor da entrada do compensador $e_i$ é o erro de corrente, calculado pela diferença da corrente de leitura que passa pelo indutor $I_L$ e da corrente de referência $I_{ref}$ gerada pelo compensador da malha de tensão, calculado na Equação 10. A Equação 13 mostra o resultado da razão cíclica completa que atua no conversor *boost*.

$$D[k] = D[k-1] + 0{,}7942.e_i[k] - 0{,}7832.e_i[k-1] \tag{13}$$

Apresentamos a lei de controle discreta que gera a razão cíclica *D* da chave *S* do conversor. A Equação 13 será implementada no dispositivo FPGA para fazer o controle digital do *boost*.

## 3.2 Estrutura de aquisição de valores

### 3.2.1 Placa de circuito impresso - PCI

Foi projetada através do software *Eagle* (*free version*) a placa de circuito impresso a qual interliga componentes eletrônicos como resistores, capacitores, sensor de corrente e o conversor analógico digital (A/D), os quais são necessários para coletar os valores de corrente e tensão que o conversor boost gera. Na Figura 17 é apresentado o esquemático do projeto elétrico da placa de circuito impresso.

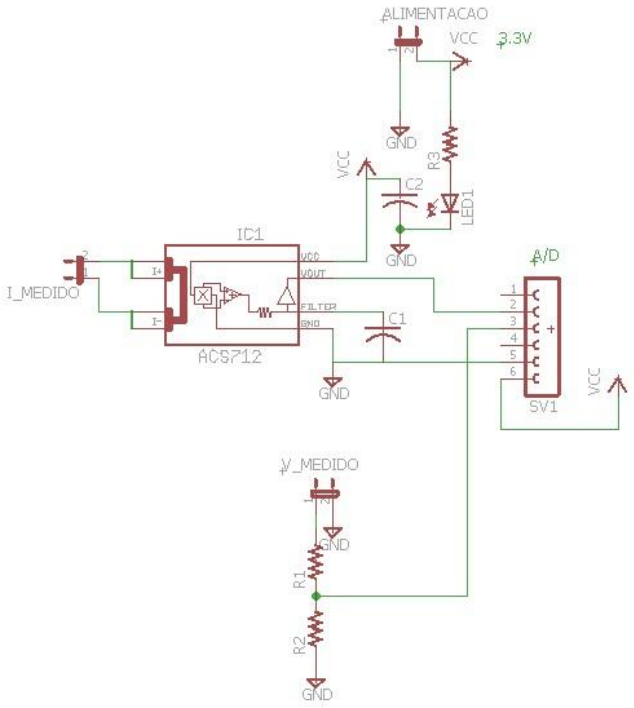



Figura 17 - Esquemático elétrico da placa de circuito impresso

Para a aquisição do valor de corrente do *Boost* foi escolhido o sensor de corrente do modelo ACS712 5A e para a aquisição de tensão foi feito um divisor resistivo que tem na sua saída o valor máximo de 3,3V, pelo fato de ser a tensão máxima que o conversor A/D suporta para leitura.

Os valores adquiridos de corrente e tensão estão na forma contínua e precisam ser discretizados, então é utilizado um conversor A/D de duas entradas de dados, assim os valores obtidos poderão ser manipulados pelo FPGA.

A tensão de saída do Boost é de 400V então é calculado na Equação 14, seguindo a *LTK* (Lei das Tensões de Kirchhoff), o valor dos resistores $R_1$ e $R_2$ para obter o valor $V_S$ igual à 3,3V na saída do divisor resistivo.

(14)

$$V_s = V_{in}\,\frac{R_2}{R_1 + R_2}$$

Substituindo os valores que queremos adquirir temos:

$$3.3 = 400\,\frac{R_2}{R_2 + R_1} \tag{15}$$

isolando os termos, o resultado é:

$$R_1 = 120{,}21\,R_2 \tag{16}$$

Substituindo (16) em (15), obtemos o valor dos resistores. Para que o experimento seja real, escolhemos os valores mais próximos de resistores comerciais, os quais são: $R_1$ = 120KΩ e $R_2$ = 1KΩ.

Os demais componentes que estão na Figura 17 como o resistor $R_3$, é somente para limitar a corrente que passa pelo LED que identifica quando a placa está energizada. Os capacitores $C_1$ e $C_2$ são usuais do sensor de corrente e seus valores estão especificados no *datasheet* [15] do mesmo.

Na Figura 18 é apresentado o circuito integrado que fará a aquisição do valor de corrente e tensão do *Boost*.

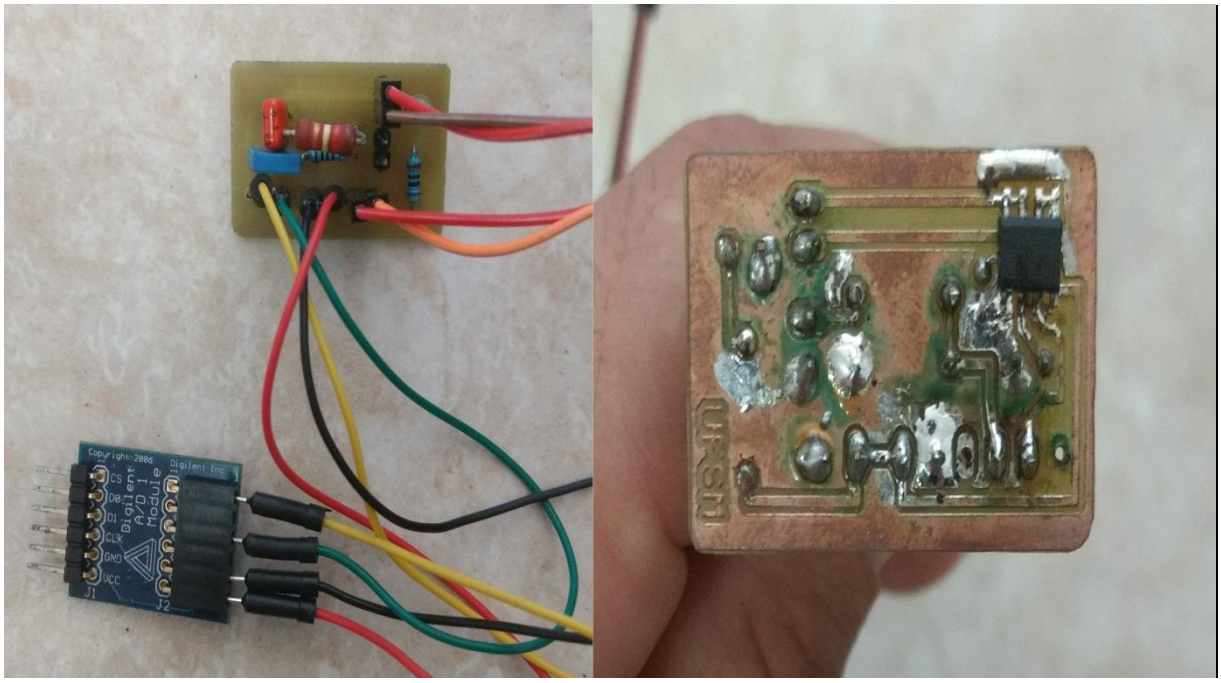



Figura 18 - Placa de circuito impresso com os componentes soldados

Na seção seguinte é apresentado o hardware integrado na placa CI encarregado de fazer a leitura da corrente que passa pelo indutor do conversor.

### 3.2.2 Sensor de corrente

O sensor utilizado no projeto é o ACS712-05B fabricado pela Allegro®, e é bastante usado em sistemas que necessitam de supervisão de corrente, tanto contínua quanto alternada. O componente eletrônico é de fácil utilização pelo usuário, por ser preciso, compacto, robusto e tendo deficiência no sensoriamento, mantendo um baixo consumo de energia, tornando-se a escolha ideal para o desenvolvimento do projeto.

Segundo o *datasheet* do fabricante [15], a corrente que passa pelo sensor gera um campo magnético que o circuito integrado interno transforma em uma tensão proporcional de saída. Assim cada vez que a corrente que passa pelos pinos de entrada 1 e 2 para os pinos 3 e 4 sofre alteração, esta será vista na tensão de saída, já que a sensibilidade do componente é de 185 mV/A. Na Figura 19 é mostrada a estrutura típica do sensor ACS712.

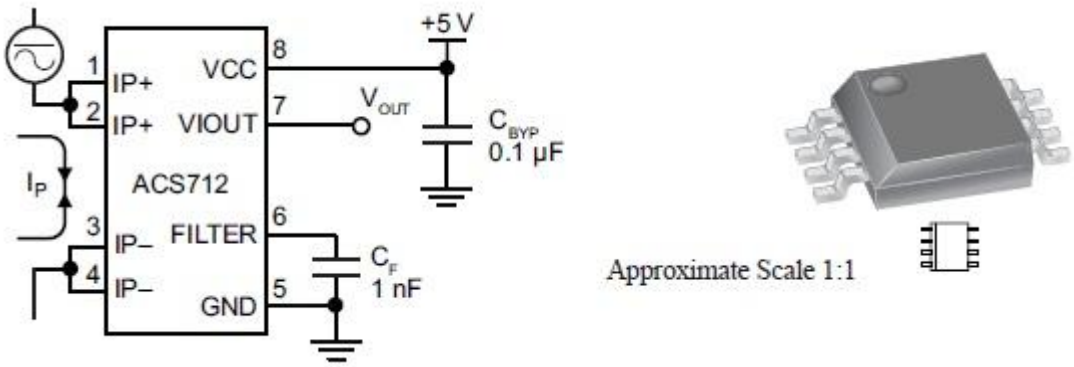



Figura 19 - Estrutura típica do AC712
Fonte: http://www.allegromicro.com/ACS712

Características de operação do sensor:

- Baixo nível de ruído do sinal analógico
- A largura de banda do dispositivo é configurado através do pino (6)
- 5us de tempo de resposta da saída
- Largura de banda de 80 kHz
- Erro total de saída de 1,5% (op. Em 25ºC).
- Resistência interna de 1.2mΩ
- Tensão de operação de 5.0V
- 66 a 185mV/A de sensibilidade na saída
- Tensão de saída proporcional em operação em DC ou AC

Na Figura 20 é apresentado o gráfico da tensão na saída do sensor de acordo com a leitura da corrente de entrada.

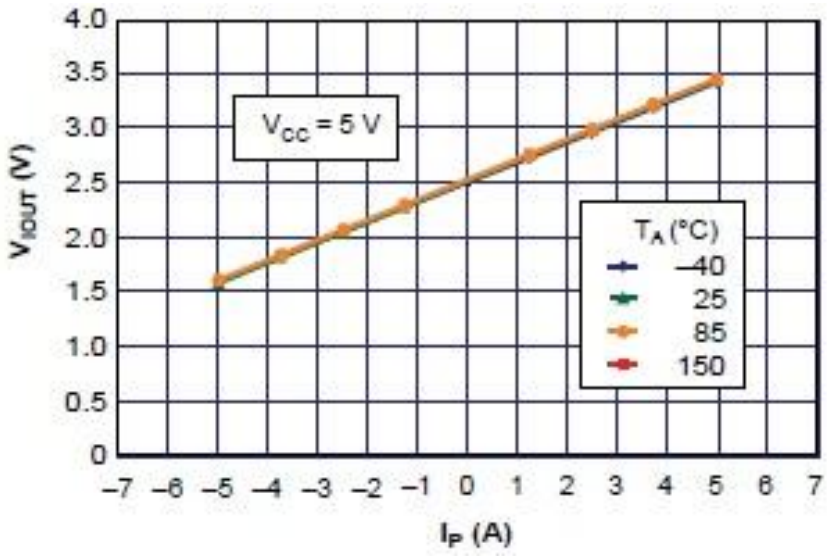



Figura 20 - Gráfico tensão de saída x corrente de entrada
Fonte: http://www.allegromicro.com/ACS712

O projeto do conversor tem que ser preciso quando falamos da corrente que circula por ele, pois é a partir dela que se faz o controle da tensão na saída. É necessário a utilização deste tipo de sensor porque a sua sensibilidade é representada por alguns milivolt/amper variados na saída.

O último componente que faz parte do hardware de aquisição de dados do boost é o conversor do sinal analógico lido, que transformado em dado digital pode ser manipulado pelo dispositivo FPGA.

### 3.2.3 Conversor A/D

O conversor analógico digital utilizado é o PMODAD1 fabricado pela Digilent®, que tem capacidade de comunicação com qualquer dispositivo que utilize a Interface Periférica Serial (SPI). Segundo o *datasheet* do fabricante [3] o conversor tem a capacidade de converter simultaneamente 2 sinais de entrada analógica para digital, variando de 0 até 3,3V com largura de 12 bits, ou seja tem um intervalo de valor digital de 0 a 4095. Na Figura 21 é mostrado seu diagrama de circuito.

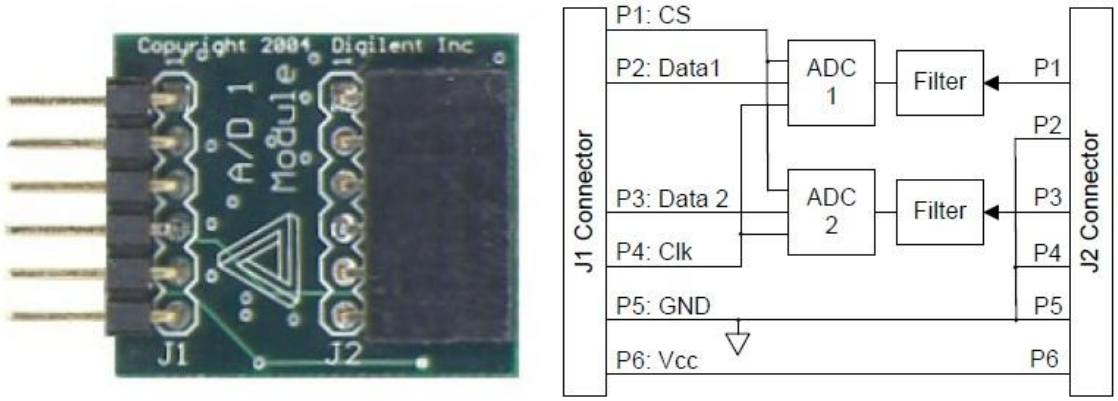



Figura 21 - Diagrama do circuito PMODAD1
Fonte: https://reference.digilentinc.com/pmod/pmod/ad1/ref_manual

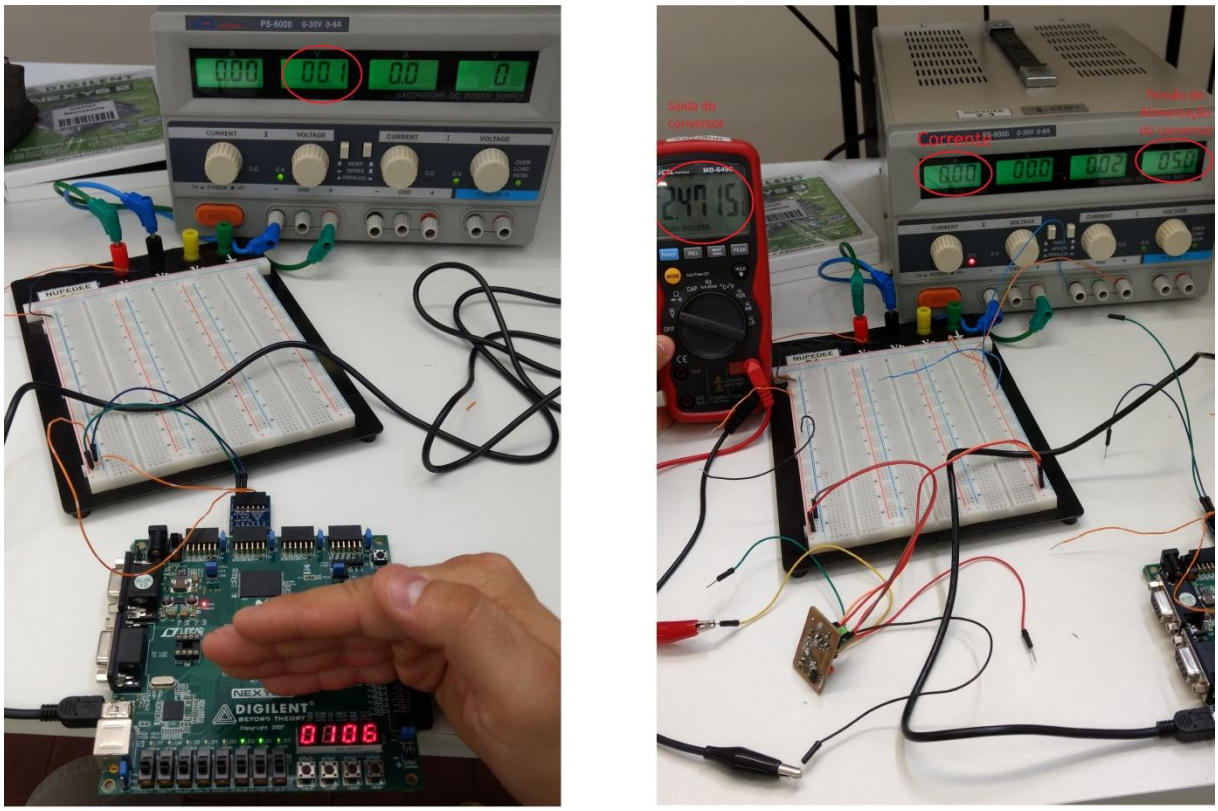
Esse dispositivo pode operar entre 2,35V até 5,25V de alimentação, então foi estabelecida a mesma alimentação que o sensor de corrente atua Vcc = 5V.

A linguagem de programação que faz comunicação do conversor com o FPGA é fornecida pelo próprio fabricante e encontra-se em anexo no trabalho após ser testada. Como seu código não é o objetivo principal do projeto ele não é apresentado no desenvolvimento.

Observando a Figura 21 na entrada analógica, conector J2, os dados entram nos pinos P1 e P3, e na saída conector J1, Data1 e Data2 estão na forma sequencial de 12 bits em cada uma das leituras.

### 3.2.4 Teste do funcionamento da placa CI

Para certificarmos o funcionamento da placa CI na aquisição de dados, foram feito alguns testes validando os valores de leitura pelo FPGA.




(a) - entrada 100mV, saída 100mV (b) - entrada 0A, saída 2,47V

Figura 22 - Testes de bancada

Na Figura 22 (a) acoplamos no FPGA o conversor A/D, o qual colocamos em uma das suas entradas analógicas uma tensão de 100mV gerados por uma fonte chaveada, e como resultado apresentou no display do dispositivo digital o mesmo valor gerado pela fonte de tensão.

Na Figura 22 (b) foi feito o teste do funcionamento do sensor de corrente já acoplado na placa CI, na sua entrada de leitura foi coloca uma corrente de 0A e na sua saída apresentou uma tensão de aproximadamente 2,5V bem como mostra a Figura 20 segundo testes realizados pelo fabricante [14].

O hardware projetado para aquisição dos dados de corrente e tensão está funcionando corretamente, assim como a simulação do controle no *Matlab*, após estas etapas será apresentado a implementação do controle digital e seus resultados, os quais serão comparados com os valores obtidos através do *Matlab*.

# 4 RESULTADOS EXPERIMENTAIS

Serão apresentados nesta seção os resultados gráficos e numéricos do controle atuando no conversor boost simulado, também será feito o comparativo entre os valores da simulação do Matlab e do Ise Xilinx/ ISim.

## 4.1 Simulação no software matlab

A partir dos valores já definidos dos componentes para o conversor boost, *seção 3.1,* foi feita a escolha do compensador PI através das plantas de corrente e tensão de acordo com os critérios do usuário, *seção 3.1.1*. A lei de controle implementada na simulação do Matlab que atua no conversor é apresentada na Figura 23.

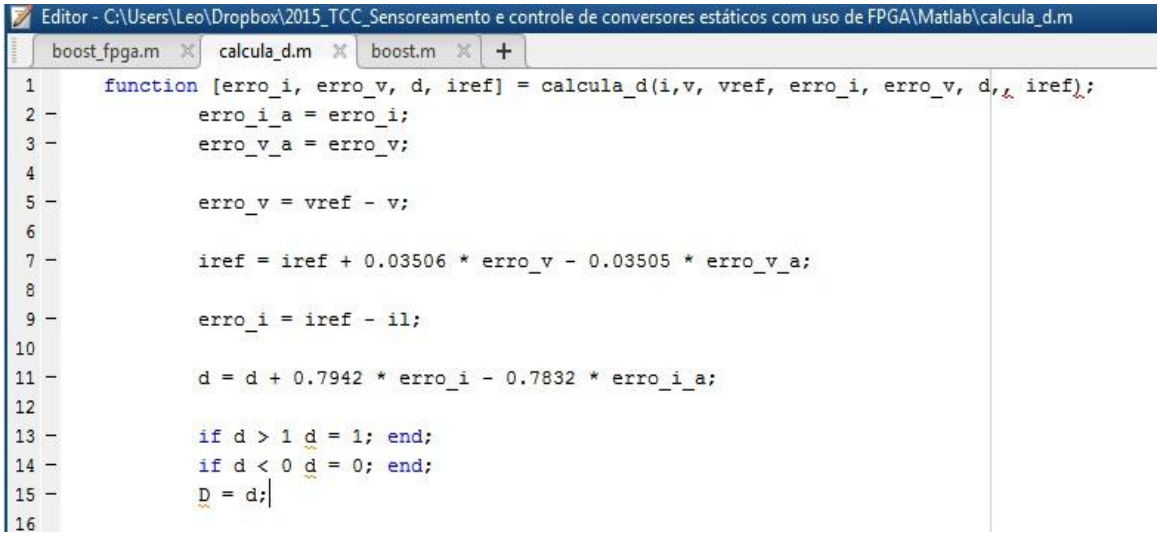



Figura 23 - Código da lei de controle aplicado no Matlab

A Figura 24 mostra o comportamento do controlador PI atuando no conversor boost.

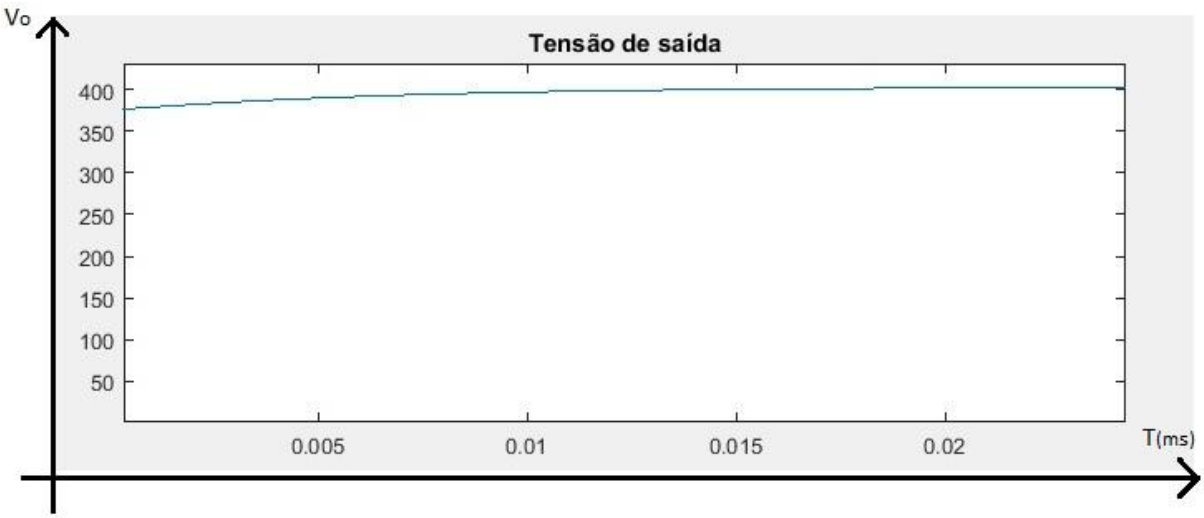



Figura 24 - Simulação do controlador PI atuando no boost

Esse é o resultado esperado na saída do boost de acordo com a escolha da velocidade de estabilização da curva característica da tensão de saída. Analisando a Figura 24 o tempo de acomodação para tensão de saída em 400V deu-se em torno de 15ms .

Na Figura 25 é apresentada a corrente que passa pelo indutor seguida pela corrente média do sistema.

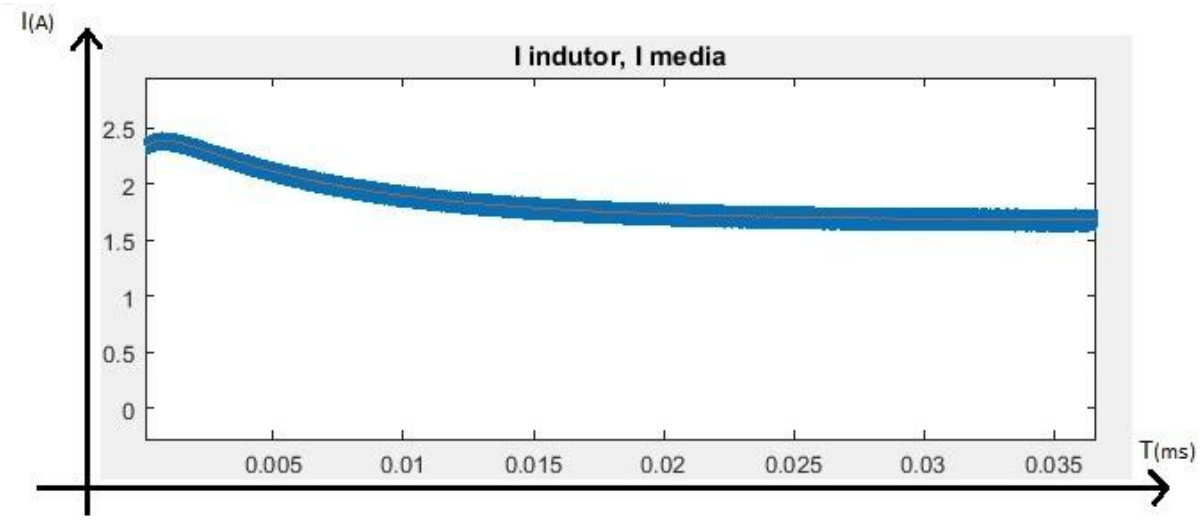



Figura 25 - Corrente média do sistema

A corrente de referência *Iref* que a saída do compensador da malha de tensão gerou é mostrada na Figura 26.

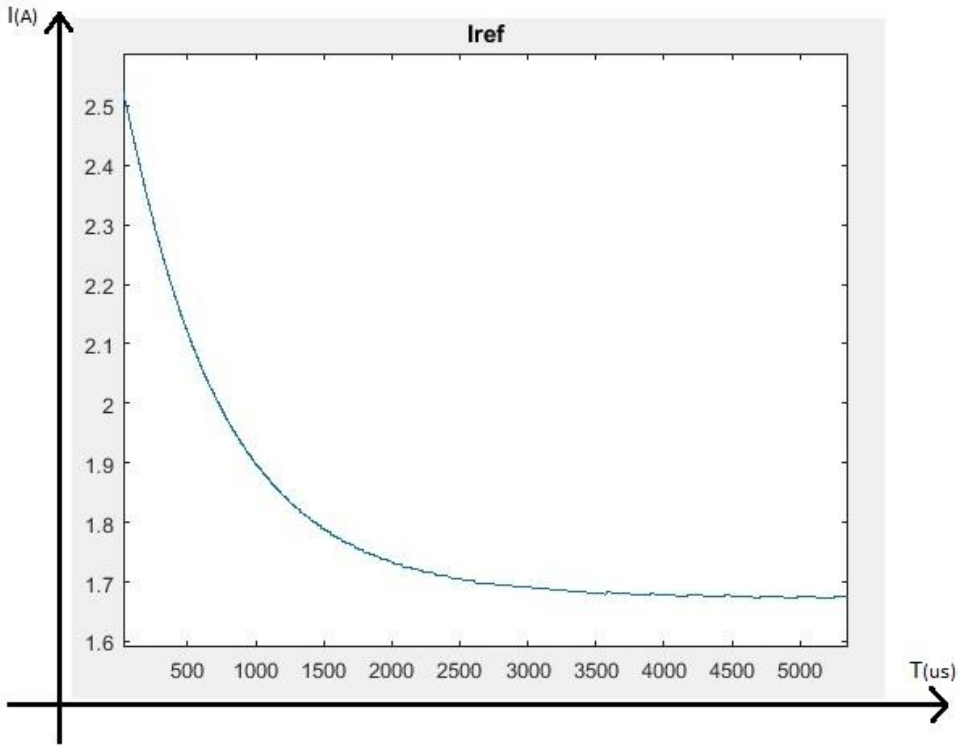



Figura 26 - Corrente de referência Iref

A corrente de referência da Figura 26 após a estabilização da mesma apresentou valores característicos da corrente média do sistema segundo a Figura 25, valor esse próximo de 1,67A. Assim os cálculos simulados no Matlab mostraram-se satisfatórios de acordo com o comportamento escolhido do controle do conversor.

A partir dos valores coletados na simulação do Matlab, iremos implementar o código VHDL para simular no Xilinx/Ise o controle digital do conversor.

## 4.2 Simulação no software Xilinx/ISE

Para fazer a implementação dos resultados dos compensadores no dispositivo FPGA em VHDL 2008, foi modificado os valores numéricos que foram obtidos no Matlab pois estavam na forma fracionária e para que se tornassem números inteiros foram multiplicados por valores em base binária. O código desenvolvido é para fins de teste de implementação em apenas um passo da simulação, ou seja, um valor discreto adquirido.

O código VHDL que faz o controle do conversor é mostrado na Figura 27, em anexo encontra-se o código completo.

```
--LEI DE CONTROLE DO CONVERSOR BOOST

v_lida   := 3749; --(400*escala_v )
i_lida   := 1413; --(i_media*escala_i)
erro_v_a := 250;  --(valor adquirido no Matlab)
erro_v   := vref - v_lida;  --(4000-3749 = 251)
temp := (36763 * erro_v) - (36753 * erro_v_a);--(0.03506*2^20=36763)(2^20*0.03505=36753)(resultado=39 263)
iref_a  := 26667017; --(valor adquirido no Matlab)
iref  := iref_a + temp;--(resultado=26 706 280)
erro_i_a := 16181257;--(valor adquirido no Matlab)
erro_i   := (iref) - i_lida * k_erro_i; -- (k_erro_i=10240)

-- de := de + 813 * erro_i - 802 * erro_i_a;

s_erro_i := conv_std_logic_vector(erro_i,32);--(conversão de inteiro para vetor de 32 bits)
s_erro_i_a := conv_std_logic_vector(erro_i_a,32);--(conversão de inteiro para vetor de 32 bits)

s_erro_i(21 downto 0) := s_erro_i (31 downto 10);--(inversão de posições para rodar 2^10= dividir 1024)
s_erro_i(31 downto 22) := (others => '0');--(resultado=11950)

s_erro_i_a (21 downto 0) := s_erro_i_a (31 downto 10);--(inversão de posições para rodar 2^10= dividir 1024)
s_erro_i_a (31 downto 22) := (others => '0');--(resultado=15802)

int1 := 813 * conv_integer(s_erro_i);--(0.7942*2^10=813)
int2 := 802 * conv_integer(s_erro_i_a);--(0.7832*2^10=802)
int3 := int1 - int2; -- (resultado= -2 957 854)

-- como os erros estão divididos por 1024, tbm deve-se dividir o s_de_a por 1024 para todos ficarem na mesma potênc
s_de_a := 12847656; --(valor adquirido no Matlab)
s_de := conv_std_logic_vector( s_de_a, 32 ) + conv_std_logic_vector( int3,32 ); -- (resultado=9 889 802)

--Matlab -> s_de_p = s_de_p * 25 / 2^28
--precisa multiplicar o s_de *25 e dividir por 2^18 pois ele ja foi dividido por 2^10

s_de_p := 25 * conv_integer(s_de);

temp_sigs_de_p := conv_std_logic_vector( s_de_p, 32 );--(conversão de inteiro para vetor de 32 bits)
d (13 downto 0) := temp_sigs_de_p (31 downto 18) ; -- (inversão de posições para rodar 2^18= dividir 262144)
d (31 downto 14) := (others => '0');
d_signal <= d  ; --(razão cíclica está aumentada 1000X em razão da frequência de chaveamento)
```

Figura 27 - Código VHDL da lei de controle do conversor

O ajuste dos valores numéricos implementados em VHDL foram feitos da seguinte maneira.

*v_lida*: Tensão lida, valor real em torno de 300V à 400V, para ajuste da base o valor foi multiplicado pela *escala_v* (* 10), valor digital de 3000 à 4000.

*I_lida*: Corrente lida, valor real em torno de 1,5A à 2,5A para ajuste da base o valor foi multiplicado pela *escala_i* (* 2^10), valor digital de 1536 à 2560.

Os valores adquiridos no controle de tensão que gera a corrente de referência *Iref* Equação 10, foram ajustados pela escala (* 2^20), ou seja 0,03506*2^20=36763 e 0,03505*2^20=36753.

Os valores adquiridos no controle da corrente que gera a razão cíclica *D* Equação 13, foram ajustados pela escala (* 2^10), resultando nos valores 0,7942*2^10=813 e 0,7832*2^10=802.

Modificado os valores fracionários da Equação 10, o resultado digital implementado do *Iref* foi alterado para (* 10 * 2^20) vezes maior que o valor real do Matlab.

Para o cálculo da razão cíclica Equação 13 primeiramente precisou-se ajustar o valor do erro de corrente *erro_i*, linha 11 da Figura 23, para todos os parâmetros ficarem na mesma escala. Na linha 13 da Figura 27 o valor de *i_lida * k_erro_i* ((*escala_v * 2^20) / escala_i*), é ajustado para que todos os parâmetros do cálculo do erro de corrente *erro_i* (*iref - i_lida*) estivessem na mesma base. Agora o *erro_i* está ajustado pela escala (* 10 * 2^20).

Por fim, o cálculo da razão cíclica *D* encontra-se na escala aumentado (*2^20 * *escala_i * escala_v*)

O teste da equação a diferenças simulado no software Xilinx/ISim versão estudante, resultou os mesmos parâmetros que o Software Matlab apresentou, mostrado o resultado na Figura 28.

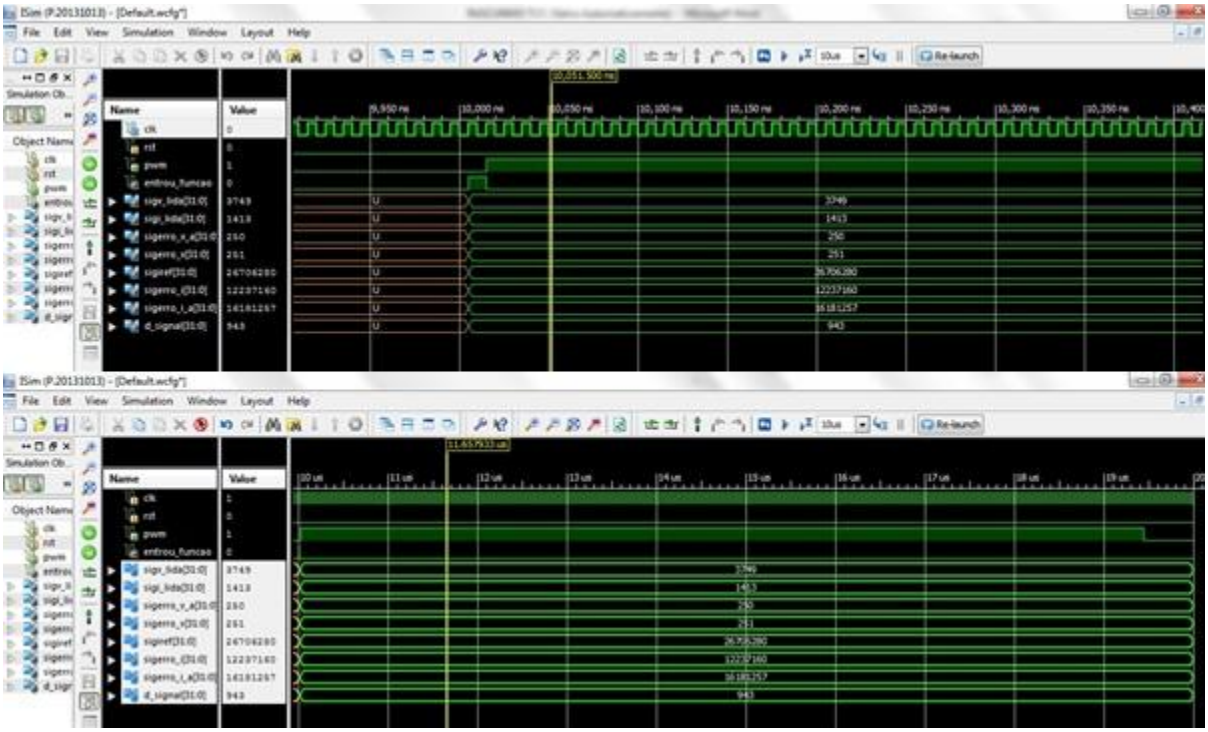


Figura 28 - Resposta da simulação do controle no Isim

O resultado da simulação na Figura 28 mostra o tempo resultante levado para calcular a razão cíclica (*d_signal*) do conversor que foi apresentado em 10ns. *d_signal* é variável de 0 à 1000 em função do divisor de clock do FPGA de 10ns

para 10us, o qual esta relacionado com a frequência de comutação do conversor escolhido em 100KHz originando o sinal *PWM*, gerado para fazer o controle da chave *S* no conversor.

A fins de teste o sinal *entrou_funcao* está em estado alto quando o contador de clock do FPGA atingiu o valor 1000, a partir deste ponto a lei de controle do boost está ocorrendo.

Na Tabela 2 é feito o comparativo dos valores da simulação no Matlab e ISim.

Tabela 2 - Comparativo dos valores de simulação

| **Váriavel** | **Matlab / Escala** | **VHDL / Escala** | **Valor real** |
|---|---|---|---|
| Corrente de aquisição | $1413/2^{10}$ | $1413/2^{10}$ | 1,379A |
| Tensão de aquisição | 3749/10 | 3749/10 | 374,9V |
| Erro de corrente | $12237160/10x2^{20}$ | $12237160/10x2^{20}$ | 1,167A |
| Erro de corrente anterior | $16181257/10x2^{20}$ | $16181257/10x2^{20}$ | 1,543A |
| Erro de tensão | 251/10 | 251/10 | 25,1V |
| Erro de tensão anterior | 250/10 | 250/10 | 25V |
| Corrente de referência | $26706280/10x2^{20}$ | $26706280/10x2^{20}$ | 2,546A |
| Razão cíclica | 0,943/1 | 943/1000 | 0,943 |

Os valores do Matlab na Tabela 2 foram adquiridos no segundo passo de simulação do conversor, em 33.33ns os quais foram fixados alguns parâmetros para serem implementdos no código VHDL, como os erros de tensão e corrente do passo anterior e razão cíclica anterior.

## 4.3 Conclusões

O comportamento obtido no sistema de malha fechada do controlador PI implementado em VHDL 2008 pode ser aplicado juntamente com a placa de circuito impresso desenvolvida para adquirir os sinais de corrente e tensão do conversor real fazendo a correção na tensão de saída em tempos de micro segundos.

# 5 CONCLUSÃO GERAL

O projeto e desenvolvimento para o controle do conversor é importante para monitoramento de sistemas que requerem estabilidade na tensão na saída. Foram apresentadas as ferramentas e a forma que foi escolhido o controle PI do conversor, a partir da simulação e dos valores dos compensadores adquiridos no *Matlab* foi feita a implementação do código no FPGA para que este fizesse a leitura, controle e atualização dos dados no boost em tempos de nano segundos.

O hardware projetado para aquisição de dados foi testado e apresentou os resultados conforme foram fixados os valores de corrente e tensão de leitura. podendo ser acoplado em um conversor real, o qual tenha as mesmas especificações do conversor que foi simulado no trabalho.

O controle digital da chave *S* do conversor foi realizado em alguns nano segundos, e esta é a razão de utilizar o FPGA para fins de realizar cálculos em paralelo, quanto menor tempo de resposta do controle maior é a eficiência do sistema proposto. A razão cíclica do chaveamento do conversor simulado foi realizada com precisão conforme a resolução de 12bits do conversor A/D no tempo de 10us, equivalente a frequência de 100KHz do boost.

A visão futura de projeto utilizando o FPGA é fazer o aperfeiçoamento da linguagem VHDL 2008, pois é possível utilizar uma biblioteca de código aberto que trabalha com números de ponto fixo, como Bishop (2012), assim não é necessário fazer a manipulação dos valores fracionários gerados pelo controle, tornando o sistema automatizado podendo ser até mesmo utilizado em outros projetos que tensões estáveis são prioridades.

# 6 REFERÊNCIAS BIBLIOGRÁFICAS

[1] BARBI, Ivo; MARTINS, Denizar Cruz. Conversores CC-CC básicos não isolados. **Edição dos autores. Florianópolis**, 2000.

[2] BEZERRA, MATHEUS SALES. Projeto, implementação e ensaios de um controlador PID utilizando FPGA. **Universidade Federal do Ceará–UFC**, 2010.

[3] BUSO, Simone; MATTAVELLI, Paolo. **Digital Control in Power Electronics**. San Rafael: Morgan & Claypool, 2006. 1ª Edição.

[4] Conversor A/D: https://reference.digilentinc.com/pmod/pmod/ad1/example_code. Acessado em 28 de abril de 2016.

[5] Conversor A/D: https://reference.digilentinc.com/pmod/pmod/ad1/ref_manual. Acessado em 28 de abril de 2016.

[6] - DORF, Richard C.; BISHOP, Robert H., Sistemas de controle moderno. **São Paulo: LTC**, 2001.

[7] FPGA - http://www.xilinx.com/esp/mil_aero/collateral/presentations/SEU_mitigation_technique.Pdf. Acessado em 05 de novembro de 2016

[8] FPGA - https://agetechnology.wordpress.com/category/altera. Acessado em 20 de outubro de 2016.

[9] HORTA, Edson Lemos. Dispositivos Lógicos Programáveis: Implementação de Sistemas Digitais em FPGAS. **São Paulo: Mackenzie**, 2013.

[10] JAPPE, Tiago Kommers et al. Análise do retificador boost monofásico sob interrupções instantâneas da tensão de alimentação. 2009.

[11] NISE, Norman S. **Engenharia de Sistemas de Controle.** Rio de Janeiro: LTC, 2002. 3ª Edição.

[12] OGATA, Katsuhiko. **Engenharia de controle moderno.** Rio de Janeiro: LTC, 2000, 3ª Edição.

[13] RASHID, Muhammad H. **Eletrônica de potência: circuitos, dispositivos e aplicações**. Makron, 1999.

[14] RIBEIRO, Rafael Vendrell. **SISTEMA FOTOVOLTAICO AUTÔNOMO BASEADO EM CONVERSORES CC-CC BOOST**. 2011. Tese de Doutorado. Universidade Federal do Rio de Janeiro.

[15] Sensor: http://www.allegromicro.com/~/media/files/datasheetts/acs712-datasheet. Acessado em 29 de junho de 2016.

# APÊNDICE A – CÓDIGO DE SIMULAÇÃO NO MATLAB

```
close all;
clear all;
n=1e-9;   u=1e-6;   m=1e-3;   k = 1e3;
vo_ref  = 400;
vi      = 300;
L       = 4000*u;
C       = 200*u;
p_out   = 500;           %% potencia de saida
R       = vo_ref^2 / p_out;
fs      = 100e3;         %% freq de chaveamento
T       = 1 / fs;        %% periodo de chav.
num_passo_periodo = 300;
passo   = T / num_passo_periodo;

num = vo_ref / L;%%PLANTA DE CORRENTE
den = [1 0];
planta_i = tf(num, den);

num = R;
den = [C*R 1];
planta_v = tf(num ,den);%%PLANTA DE TENSÃO

t_final = 80*m;
t  = 0 : passo : t_final;
tamanho = length(t);
vi = vi * ones(tamanho,1);
il = zeros(tamanho,1);   il(1) = 1;
vo = zeros(tamanho,1);     vo(1)=vo_ref - 25;
IM = zeros(tamanho,1);
I_MEDIA = zeros(round(tamanho / num_passo_periodo),1);
V_D = zeros(tamanho,1);
d  = (vo_ref - vi(1)) / vo_ref;
d=0;
cont_T = 0;
erro_i = 0;
i_media = il(1) * num_passo_periodo;
D(1) = d;
erro_v=0;
escala_i = 4096/4;
escala_v = 4096/409.6;
iref = round(p_out / vi(1)* 2^20  *  escala_v);


de=d * 2^10 * escala_v * escala_i;

for i = 2 : tamanho
    if t(i) > cont_T * T
        ti = t(i);
        cont_T = cont_T + 1;
        i_media = i_media / num_passo_periodo;
        i_aquisicao = round(i_media * escala_i);
        vo_aquisicao  = round(vo(i-1) * escala_v);

        erro_i_a = erro_i;
```

```
        erro_v_a = erro_v;
        erro_v = round(vo_ref*escala_v) - vo_aquisicao;
        ERRO_V(cont_T) = erro_v;

%        iref = iref + round(0.03506*2^20) * erro_v - round(2^20*0.03505) *
erro_v_a;
        iref = iref + round(36763 * erro_v) - round(36753 * erro_v_a);

        IREF(cont_T) = iref;
        if iref<0  iref = 0; end;

        erro_i = (iref) - i_aquisicao * round(escala_v * 2^20/escala_i); %
x ( escala_v *2^20)
        ERRO_I(cont_T) = erro_i;
       % de = de + 2^10*round(0.7942 * erro_i) - round(0.7832 *
erro_i_a)*2^10; %x(escala_v * 2^20 * 2^10)
        de = round(de + 813 * erro_i - 802 * erro_i_a);
        DE(cont_T) = de;

        d = de / 2^10 / escala_v /  2^20;
        if d > 1 d = 1; end;
        if d < 0 d = 0; end;
        D(cont_T) = d;

        I_MEDIA(cont_T) = i_media;

    end;
    V_D(i) = d;
    tp = t(i) - ti;
    if (tp >= d * T)  %% chave off
        il(i)   = il(i-1) + (vi(i-1) - vo(i-1))   * passo / L ;
        vo(i)   = vo(i-1) + (il(i-1) - vo(i-1)/R) * passo / C;
    else  %% chave on
        il(i)   = il(i-1) + vi(i-1) *  passo /L ;
        vo(i)   = vo(i-1) - vo(i-1) * passo / (R*C);
    end
    if il(i) < 0  il(i) = 0 ;  end;
    i_media = i_media + il(i);
    IM(i) = I_MEDIA(cont_T);
end
```

# APÊNDICE B - CÓDIGO DE SIMULAÇÃO NO XILINX/ISE

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

entity boost_teste is

                        port(

                        CLK     : in std_logic;
                        RST             : in std_logic;
                        PWM   : out std_logic

                        );


end boost_teste;

architecture Behavioral of boost_teste is

signal entrou_funcao : bit;

signal sigv_lida,sigi_lida, sigerro_v_a,sigerro_v,sigiref,sigerro_i, sigerro_i_a      :
std_logic_vector(31 downto 0);
signal d_signal : std_logic_vector(31 downto 0); -- varia de 0 a 1000

begin

        process(clk)

variable v_lida : integer;
variable i_lida : integer;
variable cont : integer range 0 to 1000;
variable erro_v, erro_v_a : integer;
variable vref : integer := 4000; --400v * escala_v
variable iref : integer := 26667017;
variable temp: integer range -1023 to 260000000;
variable k_erro_i :integer := 10240; -- round(escala_v * 2^20/escala_i)
variable erro_i : integer;
variable erro_i_a : integer;
variable de : std_logic_vector (34 downto 0);
variable temp_1, temp_2 : std_logic_vector (34 downto 0);
variable s_erro_i, s_erro_i_a  : std_logic_vector (31 downto 0);
variable d  : std_logic_vector (31 downto 0);
variable s_de, temp_sigs_de_p  : std_logic_vector (31 downto 0);
variable s_de_a  : integer := 12847656;
variable de_aux : std_logic_vector (24 downto 0);
variable d_temp : std_logic_vector (31 downto 0);
variable int1,int2, int3, s_de_p : integer;

  begin
    if rising_edge(clk)
            then if cont < 1000   then entrou_funcao <= '0';
            cont := cont + 1;
```

```
 if d > cont then PWM <= '1';
           else PWM <= '0';
end if;
else – começo da lei de controle
entrou_funcao <= '1';
cont := 0;

v_lida   := 3749;--conv_integer(DATA1) dado digital do A/D
sigv_lida <= conv_std_logic_vector( v_lida, 32 );
i_lida   := 1413;--conv_integer(DATA2); dado digital do A/D
sigi_lida <= conv_std_logic_vector( i_lida, 32 );
erro_v_a := 250;
sigerro_v_a <= conv_std_logic_vector( erro_v_a, 32 );

erro_v   := vref - v_lida;
sigerro_v <= conv_std_logic_vector( erro_v, 32 );

-- iref     := iref + (36763 * erro_v) - (36753 * erro_v_a);

temp := (36763 * erro_v) - (36753 * erro_v_a);
iref  := iref + temp;

sigiref <= conv_std_logic_vector( iref, 32 );

erro_i_a := erro_i;
erro_i   := (iref) - i_lida * k_erro_i;
sigerro_i <= conv_std_logic_vector( erro_i, 32 );

erro_i_a := 16181257;
sigerro_i_a <= conv_std_logic_vector( erro_i_a, 32 );

s_erro_i := conv_std_logic_vector(erro_i,32);
s_erro_i_a := conv_std_logic_vector(erro_i_a,32);

s_erro_i(21 downto 0) := s_erro_i (31 downto 10);--troca as posiçoes e divide por 1024
s_erro_i(31 downto 22) := (others => '0');--11950

s_erro_i_a (21 downto 0) := s_erro_i_a (31 downto 10);--troca as posiçoes e divide por 1024
s_erro_i_a (31 downto 22) := (others => '0');


int1 := 813 * conv_integer(s_erro_i);
int2 := 802 * conv_integer(s_erro_i_a);
int3 := int1 - int2;

-- de := de + 813 * erro_i - 802 * erro_i_a;

-- como os erros estão divididos por 1024, tbm deve-se dividir o s_de_a por 1024

s_de := conv_std_logic_vector( s_de_a, 32 ) + conv_std_logic_vector( int3,32 );
d (11 downto 0) := s_de (31 downto 20) ; -- desloca 20 bits para direita que 2^20
d (31 downto 12) := (others => '0');

d_temp := conv_std_logic_vector(conv_integer(d) * 1024 , 32);
d_signal <= d_temp  ;

end if;
end if;
end process;
```

# ANEXO A – CÓDIGO VHDL PARA COMUNICAÇÃO DO CONVERSOR A/D

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

	entity AD1_controller is
		Port	(
	    --General usage
				CLK	: in std_logic;	-- System Clock (50MHz)
				RST		: in std_logic;

		 --Pmod interface signals
				SDATA1	: in std_logic;
				SDATA2		: in std_logic;
				SCLK	: out std_logic;
				CS	: out std_logic;

		--User interface signals
				DATA1	: out std_logic_vector(11 downto 0);
				DATA2	: out std_logic_vector(11 downto 0);
				--START		: in std_logic;

				DONE	: out std_logic
				);
	end AD1_controller;

	architecture AD1 of AD1_controller is


	type states is (Idle,ShiftIn,SyncData);
			signal current_state : states;
			signal next_state : states;
			signal temp1	: std_logic_vector(15 downto 0) := (others => '0');
			signal temp2	: std_logic_vector(15 downto 0) := (others => '0');
			signal dat1	: std_logic_vector(11 downto 0):= x"000";
			signal dat2	: std_logic_vector(11 downto 0):= x"000";

			signal clk_div	: std_logic;
			signal clk_counter	: std_logic_vector(27 downto 0);
			signal shiftCounter	: std_logic_vector(3 downto 0) := x"0";
			signal enShiftCounter: std_logic;
			signal enParalelLoad : std_logic;
			signal START		: std_logic;


		begin



			clock_divide : process(rst,clk)

			begin
				if rst = '1' then
					clk_counter <= "0000000000000000000000000000";
```

```
                elsif (clk = '1' and clk'event) then
                        clk_counter <= clk_counter + '1';
                end if;
        end process;

        clk_div <= clk_counter(1);
        SCLK <=  not clk_counter(1);


counter : process(clk_div, enParalelLoad, enShiftCounter)
        begin
                if (clk_div = '1' and clk_div'event) then

                        if (enShiftCounter = '1') then
                           temp1 <= temp1(14 downto 0) & SDATA1;
                                temp2 <= temp2(14 downto 0) & SDATA2;
                                shiftCounter <= shiftCounter + '1';
                        elsif (enParalelLoad = '1') then
                          shiftCounter <= "0000";
                          dat1 <= temp1(11 downto 0);
                                dat2 <= temp2(11 downto 0);
                        end if;
                end if;
        end process;
  DATA1 <= dat1 ;
  DATA2 <= dat2 ;


SYNC_PROC: process (clk_div, rst)
   begin
     if (clk_div'event and clk_div = '1') then
       if (rst = '1') then
         current_state <= Idle;
       else
         current_state <= next_state;
       end if;
     end if;
   end process;


OUTPUT_DECODE: process (current_state)
   begin
     if current_state = Idle then
                        enShiftCounter <='0';
                        DONE <='1';
                        CS <='1';
                        enParalelLoad <= '0';
        elsif current_state = ShiftIn then
                        enShiftCounter <='1';
                        DONE <='0';
                        CS <='0';
                        enParalelLoad <= '0';
        else --if current_state = SyncData then
                        enShiftCounter <='0';
                        DONE <='0';
                        CS <='1';
                        enParalelLoad <= '1';
        end if;
   end process;
```

```
NEXT_STATE_DECODE: process (current_state, START, shiftCounter)
  begin

    next_state <= current_state;  --default is to stay in current state

    case (current_state) is
      when Idle =>
        if START = '1' then
          next_state <= ShiftIn;
        end if;
      when ShiftIn =>
        if shiftCounter = x"F" then
          next_state <= SyncData;
        end if;
      when SyncData =>
                        if START = '0' then
        next_state <= Idle;
                        end if;
      when others =>
        next_state <= Idle;
    end case;
  end process;

        process (CLK)
                        variable cont: integer range 0 to 50E3;

                        begin

                                if (CLK = '1' and CLK'EVENT) then
                                        if(cont < 50E3) then
                                                cont := cont + 1;
                                                else
                                                cont:= 0;
                                                end if;

                                                if (cont <100) then
                                                START <= '1';
                                                else
                                                START <= '0';
                                                end if;

                                end if;
                                end process;


end AD1;
```