UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
DEPARTAMENTO DE SISTEMAS ELÉTRICOS DE AUTOMAÇÁO E ENERGIA
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO

GABRIEL STEFANIAK NIEMIEC - 262503
LUIS MIGUEL SANTOS BATISTA - 265037
NICOLAS EYMAEL DA SILVA - 262506

RELATÓRIO FINAL

Relatório apresentado como requisito parcial para a aprovação na disciplina de Microcontroladores.

Professor: Walter Fetter Lages

Porto Alegre
2019

SUMÁRIO

1 INTRODUÇÃO

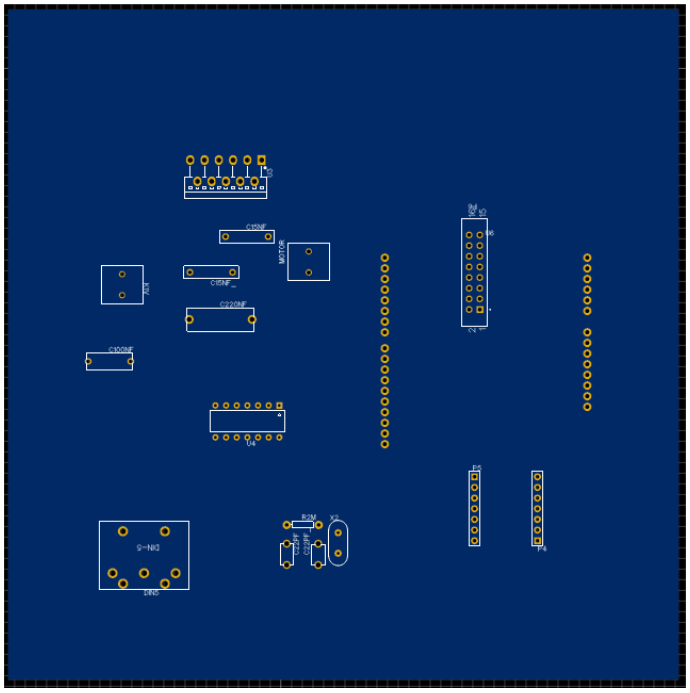
Neste relatório serão apresentadas todas as etapas envolvidas para o projeto de um *shield* para a Intel Galileo Gen2. Tal dispositivo deve ser capaz de controlar o funcionamento do robô Quanser 2DSFJ, mais especificamente uma de suas juntas. Para isso, foram projetados tanto o hardware quanto o software a fim de atender às especificações estabelecidas.

Inicialmente, com o auxílio do software *EasyEDA* foi projetado o esquemático elétrico e layout do projeto. Em seguida, foi montada a placa de circuito impresso capaz de ler os encoders, sensores de fim de curso e acionar o seu motor por PWM. Quanto ao software foi implementada uma biblioteca para controlar o Quanser, a API possibilita o acionamento dos motores em Volts, a leitura dos encoders em radianos e a leitura dos sensores de fim do curso. O cálculo da tensão a ser aplicada no motor num determinado instante segue a técnica do controlador proporcional integral derivativo (ou simplesmente PID), de modo a minimizar o sinal de erro.

2 HARDWARE

O projeto foi desenvolvido no sistema CAD *EasyEDA*, uma ferramenta *web* que permite a criação e edição de esquemáticos e *layouts* de PCI (vide imagens 2.1 e 2.2).

Figura 2.1: Camada de cima da PCI.



O resultado final da placa pode ser visto na Figura 2.3. A PCI foi feita por uma fresa de PCIs localizada no Laboratório de Automação da UFRGS e os componentes foram soldados pelo grupo. As entradas e saídas do circuito são conectadas diretamente na Galileo (vide Tabela 2.1) ou atráves de conectores KRE.

O hardware pode ser dividido em três módulos independentes de acordo com suas funções: o acionamento do motor, a leitura do encoder e a leitura dos sensores de fim de curso.

Tabela 2.1: Pinos utilizados no *shield*.

IO2	OUT	Bridge Enable	Ativa/desativa a ponte H
IO6	OUT	PWM	Controla a rotação do motor
IO9	OUT	Decoder Enable	Ativa/desativa a contagem do decoder
IO4	IN	Limit Switch 0	Detecta se o braço está no limite da esquerda
IO5	IN	Limit Switch 1	Detecta se o braço está no limite da direita
IO10	OUT	Decoder SS	“Slave Select”, controla a transferência de dados
IO11	OUT	SPI MOSI	“Master Out Slave In” da comunicação SPI
IO12	IN	SPI MISO	“Master In Slave Out” da comunicação SPI
IO13	OUT	SPI SCK	Sinal de Clock pro decoder

Figura 2.2: Camada de baixo da PCI.

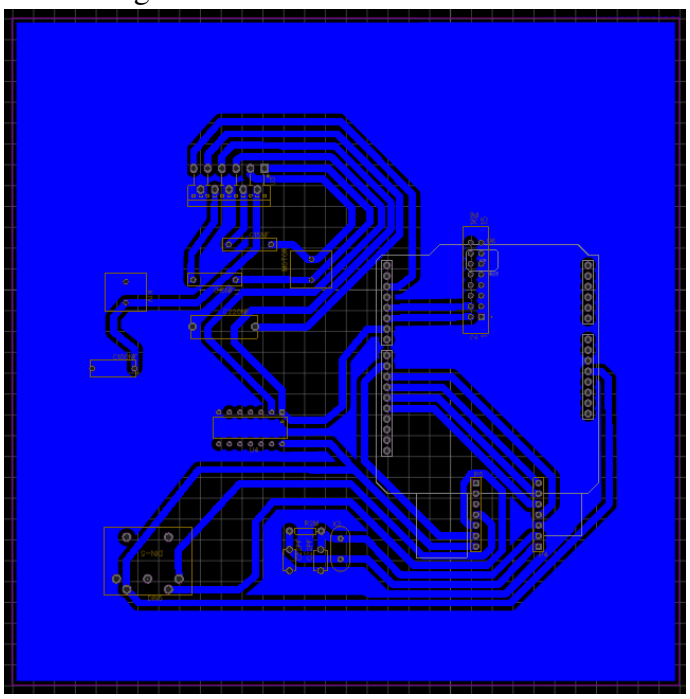
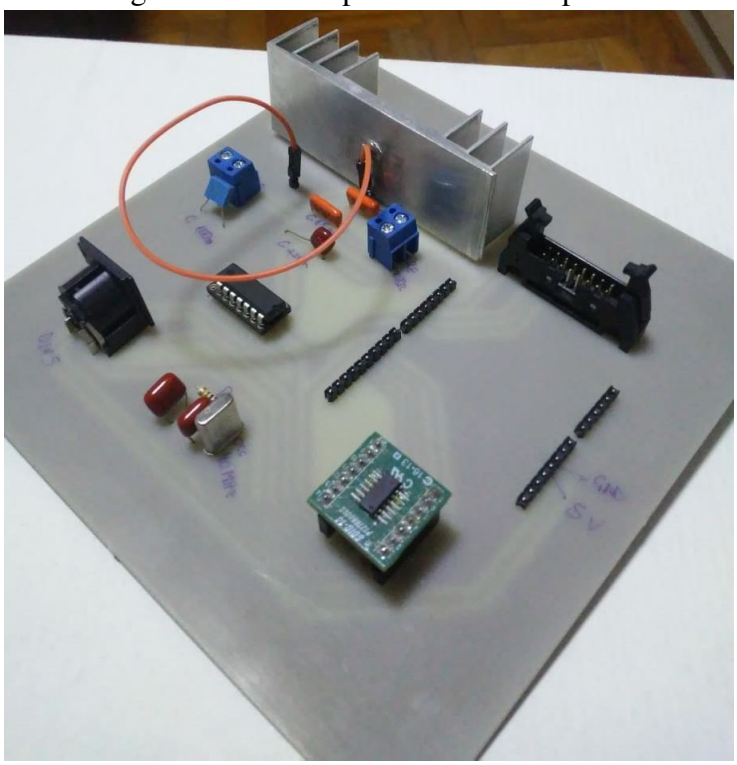


Figura 2.3: *Shield* para Galileo completo.



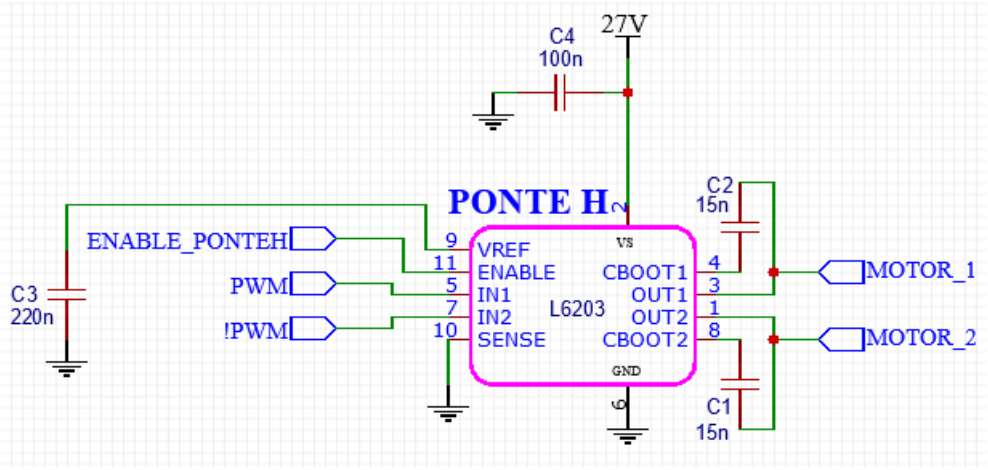
2.1 Módulo de acionamento do motor

O motor é acionado por PWM (com resolução de pelo menos 8 bits) tal que um ciclo de trabalho de 0% corresponde ao motor girando em velocidade máxima em um sentido e ciclo de trabalho de 100% corresponde ao motor girando em velocidade máxima

no sentido oposto, com uma variação linear entre os dois extremos. Para implementar esse comportamento linear é utilizando uma ponte H, um circuito capaz de determinar o sentido de corrente e valor de tensão no controle de um motor DC através do chaveamento de transistores controlado por um sinal PWM.

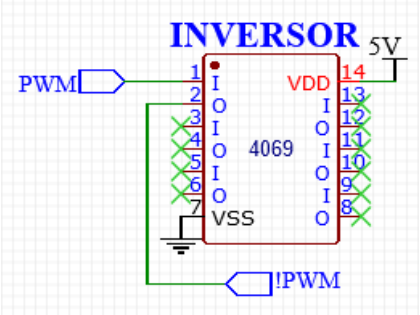
Como a fonte para o motor fornece uma corrente de pico de 3A, foi necessário escolher uma ponte H que conseguisse suportar tal corrente, então escolhemos o CI L6203 (vide Figura 2.4) que é um *driver* de ponte completa capaz de operar com uma corrente máxima de 5A e tensões de até 48V, juntamente com um dissipador térmico. Os capacitores foram dimensionados conforme os dados do *datasheet*.

Figura 2.4: Esquemático da ponte H L6203



O dispositivo é controlado por um sinal PWM em uma entrada e pelo seu sinal negado na outra entrada. O sinal foi negado utilizando o CI inversor 4069 (ver figura 2.5. O sinal PWM foi gerado pela Galileo com uma frequência de 50Hz e seu ciclo de trabalho pode ser alterado por *software* com as funções disponibilizadas na API.

Figura 2.5: Esquemático do inversor 4069



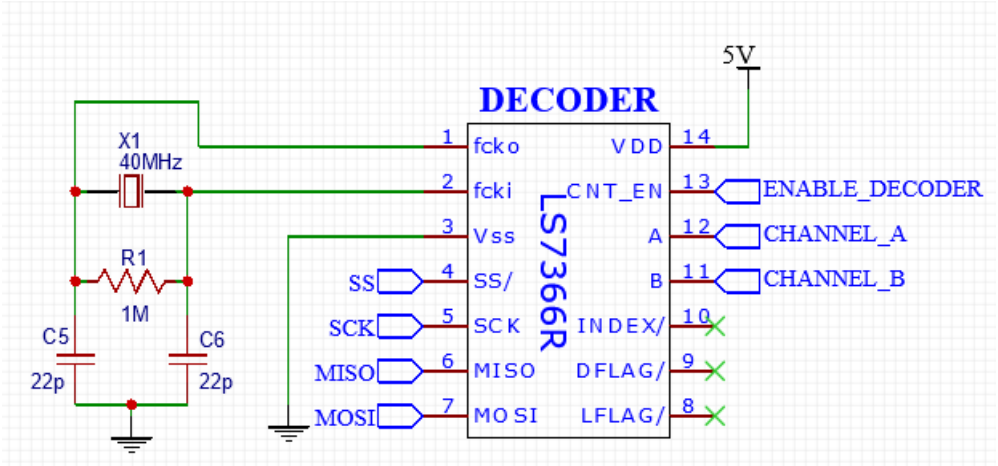
2.2 Módulo de leitura do *encoder*

O *Quanser* 2DSFJE possui quatro *encoders* óticos de quadratura, dispositivos eletromecânicos que utilizam sensores de luz para converter posição ou movimento angular em um sinal elétrico, cada um deles com uma precisão de 1024 linhas por revolução.

O *hardware* é capaz de realizar a decodificação e contagem de um dos *encoders* de uma das juntas do robô. Essa contagem não é feita totalmente por *software* por questão de precisão.

Para realizar a decodificação e contagem por *hardware*, optamos pelo CI *decoder* LS7366R (vide Figura 2.6), um contador em quadratura de 32-bits com interface serial. A comunicação com o *decoder* é feita pelo protocolo SPI através de uma arquitetura mestre-escravo com um único mestre. A configuração é feita através de escritas de *opcodes* em registradores específicos do *decoder* e os *opcodes* são disponibilizados no *datasheet*.

Figura 2.6: Esquemático do decoder LS7366R



Os pulsos de *clock* para o funcionamento do *decoder* são gerados por um cristal oscilador externo de 40MHz e os valores dos capacitores foram obtidos do *datasheet*. Como o *EasyEDA* não possuia o CI LS7366R em seu catálogo, optamos por incluir duas barras de 7 pinos espaçadas conforme as especificações técnicas do CI no *layout*.

2.3 Módulo dos sensores de fim de curso

O módulo mais simples entre os três, é responsável pela leitura de dois dos quatro sensores de fim de curso presentes no *Quanser*, pois pretendemos controlar apenas uma de suas juntas.

Essa parte consiste de duas entradas GPIO que são conectadas ao *Quanser* por meio de um conector *latch* de 16 pinos. O sensor gera um sinal de 5V quando o braço está em uma posição livre e 0V quando o braço está no fim de curso.

3 SOFTWARE

Quanto ao Software, será mostrado neste relatório uma visão geral do que foi implementado em termos de código. Além disso, será descrita a estrutura de diretórios e biblioteca utilizada.

3.1 Código

Em termos de software, dividimos as funcionalidades entre três bibliotecas. Isso permite encapsular os dados envolvidos, reutilizar o código em outras aplicações e torna a programação mais segura de erros humanos. Além do que é relatado aqui, foi enviado em anexo um conjunto de arquivos de documentação gerados pelo *doxygen*. Para mais informações, é encorajada veementemente a leitura da documentação.

3.1.1 Módulo pwm.c

É constituído de três funções para inicializar e habilitar o sinal PWM gerado pela Galileo, assim como modificar o *duty-cycle* (a fração do período do sinal que é mantida em valor lógico alto). Também constam os literais que regulam o próprio período do sinal. Utilizamos 20ms.

3.1.2 Módulo sensors.c

Este módulo contém as chamadas para ler e configurar os sensores de fim de curso e o *decoder*. O acesso ao *decoder* é bem complexo. Tentamos desenvolver a API de modo a tornar a leitura dele o mais simples e transparente o possível. Acreditamos ter conseguido esconder essa complexidade do usuário muito bem. Apesar da simplicidade dos sensores de fim de curso, também foi implementada uma função para a sua leitura.

3.1.3 Módulo quanser.c

O módulo *quanser* é a interface principal da API e contém chamadas para todas as funcionalidades exigidas na descrição do trabalho (leitura dos sensores e acionamento dos motores). Apesar de a aplicação PID ser solicitada apenas como modo de demonstrar a funcionalidade da biblioteca, decidimos incluí-la também na biblioteca em si visto que imaginamos ser uma aplicação bem interessante considerando o alvo da biblioteca. Se desejar, o usuário pode fazer sua própria implementação do controlador PID usando as outras funções deste módulo.

3.2 Estrutura de Diretórios

A pasta principal possui cinco subpastas. Elas estão organizadas da seguinte maneira:

- **include**: é reservada para os arquivos de cabeçalho;
- **init**: possui o script de inicialização da Galileo Gen 2 usado para a configuração das portas GPIO e do acesso aos arquivos correspondentes;
- **lib**: contém a biblioteca libgalileo.a que é utilizada como auxiliar no código da API do *shield*;
- **src**: responsável por organizar os arquivos que implementam as funcionalidades da API. Além disso contém o makefile que gera os arquivos objeto e compila os arquivos na pasta de teste;
- **tests**: agrupa os arquivos de teste utilizados para a validação do projeto;
- **html**: pasta que contém os arquivos HTML gerados pelo *doxygen* como documentação da nossa biblioteca.

4 CONCLUSÃO

Com os resultados deste relatório, foi possível colocar em prática diversos conhecimentos adquiridos no decorrer do curso. O projeto de um *shield* envolveu etapas de desenvolvimento de software e hardware, além de realizar a integração entre eles. Algumas dificuldades foram encontradas no que se diz respeito ao desenvolvimento de uma placa de circuito impresso, porém foi possível chegar em um resultado satisfatório.

Assim sendo, tais resultados explicitaram pontos importantes na elaboração de um hardware na prática, principalmente quanto ao dimensionamento e o roteamento dos componentes visto que há toda uma análise teórica do circuito para satisfazer a especificação. Além disso, o desenvolvimento do software é essencial e deve ser integrado com o hardware para que ambos funcionem em conjunto corretamente.

5 REFERÊNCIAS BIBLIOGRÁFICAS

https://lsicsi.com/datasheets/LS7366R.pdf

https://users.ece.utexas.edu/~˘valvano/Datasheets/L6203.pdf

https://www.ti.com/lit/ds/symlink/cd4069ub.pdf

https://www.youtube.com/watch?v=Rl50qI6e7HU

https://en.wikipedia.org/wiki/PIDcontroller

https://en.wikipedia.org/wiki/Rotaryencoder