

Lauren Rolan, Leonardo Dorneles e Thomas Fontanari

Projeto de Acionamento e Leitura dos Sensores do Braço Mecânico Quanser 2DSFJE

Porto Alegre, Brasil
04 de Julho de 2018

Lauren Rolan, Leonardo Dorneles e Thomas Fontanari

Projeto de Acionamento e Leitura dos Sensores do Braço Mecânico Quanser 2DSFJE

O objetivo deste trabalho é relatar o desenvolvimento de uma *shield* para a Galileo Gen 2 capaz de controlar o braço robótico Quanser 2DSFJE. Assim, desenvolvemos no fim uma PCI e uma biblioteca que, juntas, devem ser capazes de fazer o controle do robô.

Universidade Federal do Rio Grande do Sul

Engenharia de Computação

Porto Alegre, Brasil

04 de Julho de 2018

Resumo

Este trabalho tem como objetivo relatar o desenvolvimento do projeto de controle do braço robótico Quanser 2DSFJE. Para isto, desenvolvemos uma biblioteca, *libquanser*, capaz de controlar um hardware que também foi desenvolvido aqui. Ao longo do projeto, fizemos também a prototipação de circuitos de teste para validar o funcionamento dos esquemáticos e da biblioteca. Ao final do projeto, possuímos a PCI e a biblioteca funcionando no contexto em que foram testadas. No entanto, como não tivemos acesso ao robô durante o projeto, não podemos afirmar que este será efetivamente controlado.

Palavras-chaves: Q2DSFJE. Ponte H. decoder em quadratura. LS7366R. L6203.

Listas de ilustrações

Figura 1 – Esquemático corresponde apenas ao circuito do LS7366R. Os conectores podem ser vistos no PDF em anexo.	7
Figura 2 – Esquemático correspondente apenas ao circuito do L6203. Os conectores podem ser vistos no PDF em anexo.	9
Figura 3 – Esquemático correspondente apenas ao inversor do sinal de PWM. Os conectores podem ser vistos no PDF em anexo.	10
Figura 4 – Leiaute da placa de teste para a ponte H.	11
Figura 5 – Teste do PWM e ponte H.	12
Figura 6 – Teste do esquemático correspondente ao LS7366R	13
Figura 7 – Execução do programa de teste do circuito correspondente ao LS7366R.	13
Figura 8 – Estrutura de dados da biblioteca <i>libquanser</i>	14
Figura 9 – Leiaute da PCI desenvolvida.	15
Figura 10 – PCI após a corrosão em percloreto de ferro.	16
Figura 11 – PCI da <i>Shield</i> em seu estado final.	17

Sumário

1	Introdução	5
2	Projeto de Controle do Braço Robótico	6
2.1	Leitura dos Encoders	6
2.1.1	Hardware	6
2.1.2	Software	7
2.2	Acionamento do Motor	8
2.2.1	Hardware	9
2.2.2	Software	10
2.3	Leitura dos Sensores de Fim de Curso	11
2.4	Testes dos esquemáticos	11
2.4.1	Círculo de Açãoamento do Motor	11
2.4.2	Círculo de Leitura dos <i>Encoders</i>	12
3	Biblioteca <i>libquanser</i> e <i>Shield</i>	14
3.1	<i>libquanser</i>	14
3.1.1	Código em C	14
3.1.2	<i>Script</i> de Inicialização	15
3.2	Desenvolvimento da PCI	15
4	Conclusão	18
	Referências	19

1 Introdução

O Quanser 2DSFJ trata-se de um modelo de braço robótico. Possui, entre outras partes, dois motores Dc – um para cada junta – que podem ser usados para realizar os movimentos do braço em dois graus de liberdade. Ainda, a posição de cada parte do braço pode ser obtida através dos *encoders* e dos sensores de fim de curso. Neste trabalho, portanto, desenvolvemos uma *shield* com o objetivo de controlar o movimento de uma das juntas do robô. Para tanto, dividimos o trabalho em três partes: o acionamento do motor, a leitura dos *encoders* e a leitura dos sensores de fim de curso - parte mais simples do projeto. O acionamento do motor, como especificado, foi desenvolvido baseado em PWM, de tal forma que um ciclo de trabalho de 100% representa movimento em velocidade máxima em uma direção e um ciclo de 0% representa movimento em velocidade máxima na outra direção. O motor funciona com uma fonte de 27 V e 3 A, que é fornecida na disciplina. A decodificação dos *encoders* foi feita em quadratura e em hardware, de tal forma que não há perdas. Para tanto, usamos como base o circuito integrado LS7366R ([LSI COMPUTER SYSTEMS, INC., 2015](#)), que fornecia uma interface SPI para leitura dos contadores. Os sensores de fim de curso, finalmente, por serem simplesmente sinais digitais, foram lidos através de portas GPIO. A correção dos possíveis erros de medição e leitura foi feita por meio de um algoritmo de PID, cujos valores de P, I e D foram calculados de maneira semi-automática, num processo de *tunning* específico e semi-automático.

2 Projeto de Controle do Braço Robótico

Nesta seção do trabalho, descrevemos mais a fundo cada uma das etapas citadas anteriormente, aprofundando-nos sobre o *software* e *hardware*. Neste relatório, no entanto, não colocamos todo o código desenvolvido e nem os esquemáticos completos, já que estes podem ser acessados mais facilmente pelos arquivos em anexo.

2.1 Leitura dos Encoders

Como pode ser identificado na descrição do projeto, os *encoders* possuem duas saídas principais, *Channel A* e *Channel B*. A partir destes sinais, pode-se identificar se o braço está se movimentando no sentido horário ou anti-horário e com que velocidade. Isso pode ser interpretado através de um contador, que incrementará quando o movimento é feito em um sentido e decrementará quando em outro sentido. Essa contagem, no entanto, não pode ser feita razoavelmente por *software*. Isso acontece porque a frequência de operação do programa não seria suficientemente rápida para que todas as variações nos sinais fossem identificadas, resultando em perda na contagem. Portanto, a contagem deve ser feita em *hardware*.

Felizmente, existem circuitos integrados prontos que fazem essa decodificação em quadratura e contagem com base nos sinais de saída (*Channel A* e *Channel B*) dos *encoders*. Escolhemos, portanto, basear a leitura dos *encoders* no circuito integrado LS7366R, sugerido no Moodle da disciplina. A escolha foi feito porque a interface por SPI facilitava o processo, além do fato de que tínhamos acesso fácil a esse CI.

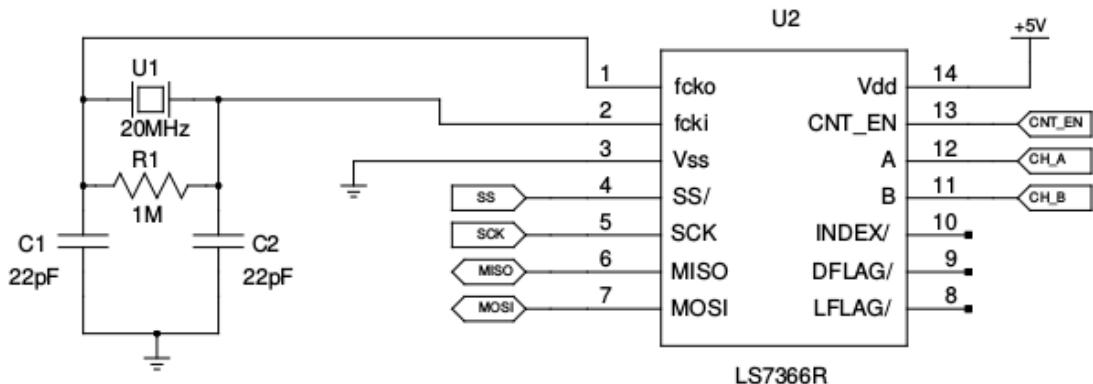
O LS7366R é um contador de 32 bits em quadratura com interface serial. Possui diversas configurações disponíveis, que podem ser configuradas através da escrita em registradores. Por exemplo, pode ser definido que a contagem será feita em quadratura através de um determinado bit do registrador MDR0. A configuração é também muito facilitada pelo cabeçalho para código em C disponibilizado pelo próprio *datasheet*. Através dele, é possível configurar os dados a serem escritos nos registradores a partir de ORs entre as diferentes definições do cabeçalho.

2.1.1 Hardware

A parte do esquemático correspondente ao circuito do decodificador é mostrada abaixo na Figura 2.1.1 (o esquemático completo pode ser visto em anexo). Para o desenvolvimento desse esquemático foi necessário decidir valores para os capacitores, resistor e frequência do cristal oscilador. Ao longo do semestre tentamos encontrar dois dos cir-

circuitos integrados LS7366R. Como dito anteriormente, tínhamos acesso fácil a um deles. No entanto, não conseguimos encontrar o segundo. Portanto, desenvolvemos de tal forma que apenas um dos *encoders* é decodificado. Acreditamos que isso não apresente grande problemas, já que as juntas do robô podem ser configuradas de tal forma que a leitura dos dois *encoders* deve ser muito parecida.

Figura 1: Esquemático corresponde apenas ao circuito do LS7366R. Os conectores podem ser vistos no PDF em anexo.



O *datasheet* explica como os valores dos capacitores podem ser calculados. No entanto, para esse cálculo, era necessário saber a capacitância do cristal oscilador e, também, a capacitância parasita observada pelo cristal. Por isso, optamos por, primeiro, testar valores que encontrássemos em exemplos de esquemáticos disponíveis pela internet. Encontramos o valor de 22 pF, mostrado na imagem, e que funcionou bem. O valor do resistor foi decidido também baseado no *datasheet*, que mostra um diagrama de blocos em que o valor de 1 M é usado. Finalmente, o valor de 20 MHz para o cristal oscilador também foi baseado em exemplos de uso do LS7366R disponibilizados na internet.

2.1.2 Software

A comunicação com o LS7366R é feita através de SPI e não é completamente trivial. Portanto, desenvolvemos um módulo em nossa biblioteca de controle do Quanser com base no código em C que o *datasheet* disponibiliza como exemplo para uso em um PIC. O mais importante deste código, na verdade, é o cabeçalho fornecido, que permite a construção fácil dos comandos.

Em nosso programa, fizemos as versões para Galileo do que se encontra no *datasheet*. Além disso, construímos funções para configurar o LS7366R da maneira apropriada para seu uso com o *encoder*. Mais detalhes sobre a biblioteca podem ser encontrados na documentação do código.

Além da comunicação, a biblioteca desenvolvida também conta com funções para o cálculo do PID baseado nas leituras pré-processadas do encoder. Optamos por usar

como pontos de referência (*Set Point* e *Process Variable*) as leituras em radianos deste dispositivo. A adaptação é então feita sobre estas medidas e seus valores esperados.

2.2 Acionamento do Motor

O motor precisa ser acionado por PWM, de tal forma que o braço robótico deve mover em um sentido quando o ciclo de trabalho é de 100% e no outro sentido quando o ciclo de trabalho é de 0%. Como a variação de velocidade deve ser linear entre os dois extremos citados e possuímos apenas uma fonte de 27 V, a solução natural é usar uma ponte H em modo *locked anti-phase*.

Consideramos diferentes soluções para a construção da ponte H. Em um primeiro momento, tentamos nos basear exclusivamente no que era disponibilizado no Moodle, já que os esquemáticos dados como exemplo provavelmente funcionariam. Assim, nosso primeiro esquemático envolvia o uso do *driver* de ponte-H LT1162. A decisão de usar este circuito integrado, no entanto, apresentava algumas dificuldades. A primeira delas é o fato de que não conseguimos encontrá-lo em lojas locais. Além disso, em sites online confiáveis encontramos apenas sua versão SMD, por um preço razoavelmente alto. Sabemos que componentes SMD são bastante difíceis de soldar com os equipamentos convencionais e isso nos motivou contra seu uso. Finalmente, como o dispositivo era apenas um *driver* de ponte H (e não uma ponte H completa), precisaríamos ainda comprar os transistores e soldá-los na placa de circuito impresso. Isso aumentaria o custo do projeto e, também, a dificuldade para construir a placa de circuito impresso.

Nossa segunda opção ainda foi baseada no material disponibilizado no Moodle. O circuito integrado 33886 é uma ponte H completa que suportava até 3 A, o suficiente para o projeto desenvolvido. Em relação ao custo, seu uso representava melhorias em relação à escolha anterior. O problema neste caso foi, novamente, que só conseguímos encontrá-lo à venda em sua versão SMD. Por isso, continuamos procurando outros circuitos integrados.

Encontramos finalmente duas possíveis soluções que poderiam ser feitas apenas com compras em lojas locais. A primeira delas era baseada no *driver* de meia ponte IR2111. O maior problema, neste caso, é que são necessários dois deles, já que é um *driver* de meia ponte apenas. Isso aumenta a complexidade da placa de circuito impresso. Além disso, o circuito não poderia ser controlado diretamente pela Galileo, já que a tensão mínima suportada para o sinal de entrada lógico *HIGH*, segundo o *datasheet*, era de 6,4 V. Por causa disso, precisaríamos de mais uma etapa no circuito para amplificar a tensão de saída do PWM da Galileo.

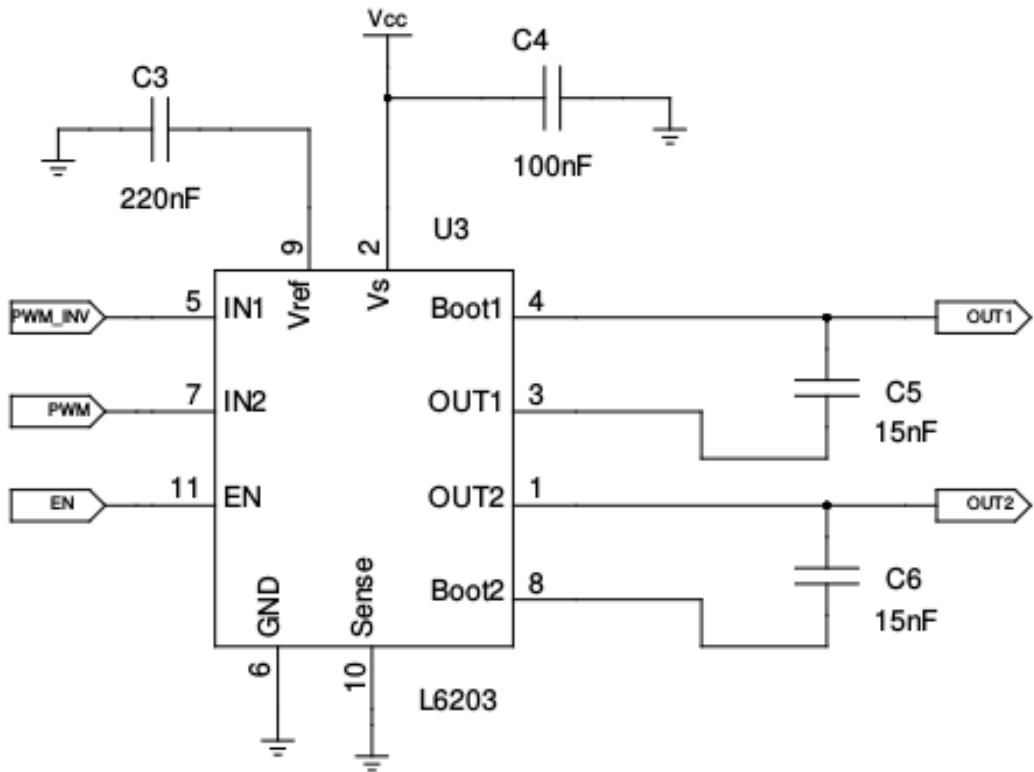
A segunda opção, e que decidimos usar enfim, é baseada no circuito integrado L6203 (ST MICROELETROONICS, 2003). Semelhante ao 33886, trata-se de uma ponte H completa. Este circuito integrado suporta até 5 A e tensão de alimentação entre 12 V

e 48 V, mais do que o suficiente para o controle do braço robótico. Ainda, por se tratar de ponte H completa, era uma solução barata. Conseguimos encontrá-lo em lojas locais por apenas 23 reais. Os detalhes do circuito associado a ele são mostrados nas próximas seções.

2.2.1 Hardware

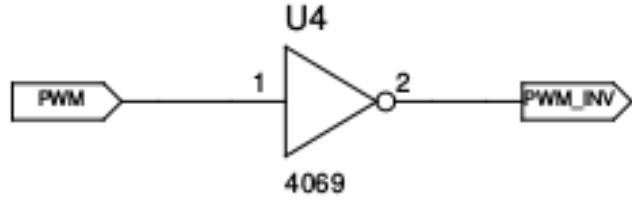
A parte do esquemático que corresponde à ponte H e ao inversor do sinal de PWM são mostradas nas Figuras 2 e 3 abaixo - lembramos que o esquemático completo está em outro arquivo. Neste caso, o *datasheet* do circuito integrado foi suficiente para que pudéssemos decidir os valores dos componentes a serem usados. As questões mais importantes dizem respeito à escolha do valor para os capacitores de *bootstrap* e como fornecer o sinal de PWM invertido.

Figura 2: Esquemático correspondente apenas ao circuito do L6203. Os conectores podem ser vistos no PDF em anexo.



O *datasheet* do L6203 estabelece um capacitor de no mínimo 10 nF. Esse mínimo é para garantir que os transistores de potência fiquem completamente ligados, minimizando a queda de tensão sobre eles. Além disso, o *datasheet* sugere que um valor muito grande pode levar a picos de corrente na saída de Sense. Visto isso e levando em conta outros

Figura 3: Esquemático correspondente apenas ao inversor do sinal de PWM. Os conectores podem ser vistos no PDF em anexo.



circuitos de exemplo que encontramos na internet, optamos por usar capacitores de 15 nF e capacidade de até 63 V.

Para obter o sinal invertido, consideramos o tempo de propagação do inversor lógico e o *dead time* do L6203. Usamos o inversor 4069, que apresenta um tempo de propagação típico de 30 ns quando operando a 10 V. O *dead time* típico do L6203 é de 100 ns e, portanto, não deve ocorrer condução simultânea nos dois braços da ponte H. Não obstante, o L6203 possui um circuito de proteção térmica, que impede que a temperatura de junção atinja 150 °C, desabilitando o dispositivo se for necessário. Assim, consideramos que o uso do inversor lógico 4069 (TEXAS INSTRUMENTS, 2016) era suficiente.

Outro ponto relevante é que, para operar nas condições requeridas de 1 A continuamente e 3 A de pico, o L6203 precisa de um dissipador. Nesse caso, já possuímos um dissipador relativamente grande, de tal forma que decidimos usá-lo.

2.2.2 Software

Para fazer o controle do L6203 foram necessários dois sinais de saída provenientes da Galileo: o sinal de *enable*, que tem como função habilitar ou desabilitar a ponte H, e o sinal de PWM. O L6203 possui duas entradas de PWM, mas, como explicado no item anterior, uma das entradas é o mesmo sinal invertido.

Para facilitar o controle destes sinais, desenvolvemos um módulo em nossa biblioteca que é responsável pelo controle de GPIOs e outro módulo que é responsável pelo controle dos PWMs. Estes módulos são chamados, respectivamente, de GPIO e GPWM. Mais informações a respeito deles podem ser lidas na documentação do código.

A respeito do PID, o PWM foi usado na função *bumpTest*, onde uma onda correspondente ao objetivo era enviada ao motor e, com base nas leituras do *encoder*, os parâmetros eram ajustados seguindo a lógica explicada em (THE..., 2015).

2.3 Leitura dos Sensores de Fim de Curso

A leitura dos sensores de fim de curso corresponde à parte mais simples do trabalho. Como pretendemos controlar apenas uma das juntas do motor, são necessários apenas duas entradas de GPIO. Para tanto, usamos a biblioteca de GPIO que havíamos desenvolvido e já citada anteriormente.

No que diz respeito ao esquemático do hardware, apenas o conector IDPC 16 já é suficiente.

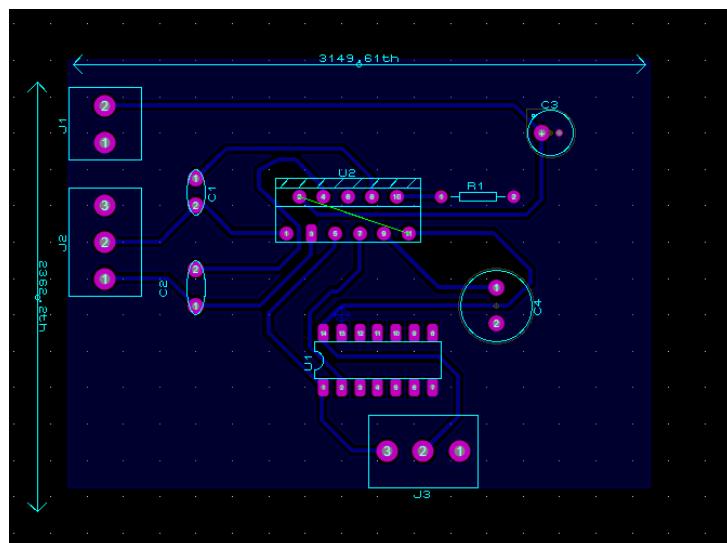
2.4 Testes dos esquemáticos

Antes da construção da *shield*, optamos por testar os circuitos de decodificação e de acionamento do motor. Para tanto, montamos o circuito correspondente ao LS7366R em uma *protoboard* e, também, montamos em uma PCI uma versão simplificada da *shield*, contendo apenas o circuito correspondente ao L6203, para que pudéssemos testá-lo com mais facilidade.

2.4.1 Circuito de Acionamento do Motor

O leiaute da placa de circuito impresso para teste do L6203, baseada nos esquemáticos discutidos acima, é mostrada na Figura 4. Construímos a PCI para facilitar o teste, porque o circuito não encaixava diretamente em uma *protoboard*. Além disso, tínhamos interesse em conseguir mais experiência com o desenvolvimento de placas de circuito impresso antes de construir o *shield*, que era mais complicado. A PCI foi desenvolvida através de transferência térmica e corrosão em percloro de ferro.

Figura 4: Leiaute da placa de teste para a ponte H.



Na Figura 5 abaixo, mostramos as saídas da placa quando usada uma tensão de 24 V para alimentar o circuito. O sinal de PWM, aqui, foi fornecido por um Arduino, já que a única fonte com tensão maior do que 12 V (requisito do L6203) a que possuímos acesso ficava no Laboratório de Graduação e não tínhamos acesso a Galileo naquele local.

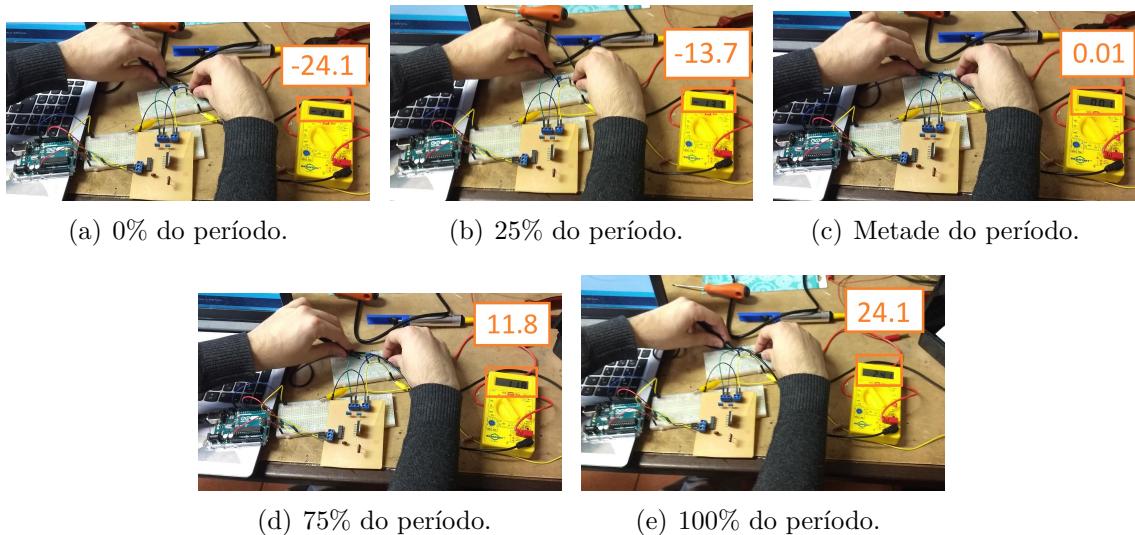


Figura 5: Teste do PWM e ponte H.

2.4.2 Circuito de Leitura dos *Encoders*

Para testar o circuito correspondente ao LS7366R, montamos o esquemático discutido anteriormente em uma *protoboard*. Como não tínhamos acesso ao motor, geramos os sinais do *encoder* artificialmente, através do Arduino. Para isso, bastou reproduzir os sinais de *Channel A* e *Channel B* de maneira a representar um incremento ou decremento. A figura 6 mostra o circuito montado, juntamente da Galileo e do Arduino.

A Figura 7 mostra o resultado da execução de um programa que faz a leitura do contador no LS7366R. O programa, primeiramente, configura os registradores MDR0 e MDR1 para uso com os *encoders*. Os valores são então lidos do próprio LS7366R, para garantir que foram escritos corretamente, e são então impressos na tela. As próximas linhas mostram o contador incrementando de acordo com os sinais produzidos pelo Arduino.

Figura 6: Teste do esquemático correspondente ao LS7366R

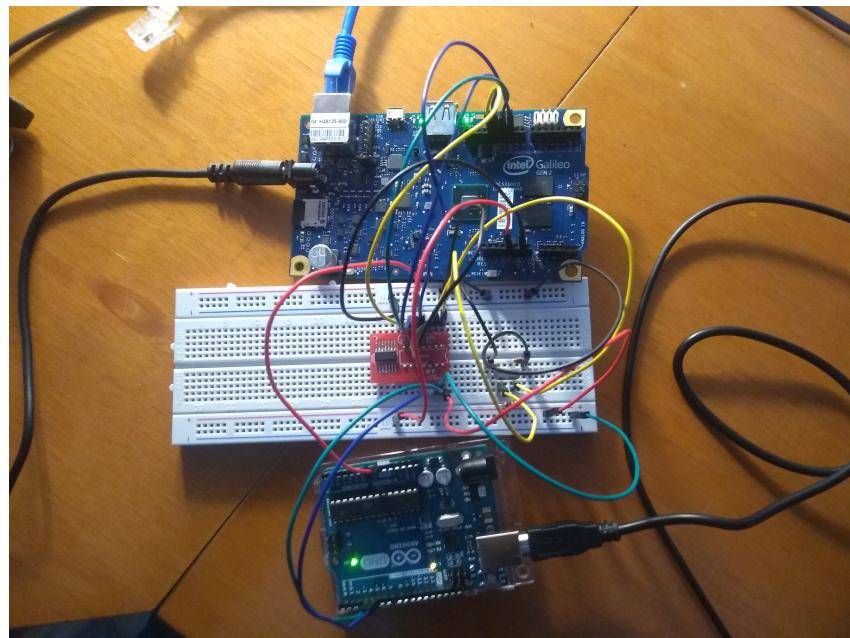


Figura 7: Execução do programa de teste do circuito correspondente ao LS7366R.

```
galileo:~$ ./test_gpio
mode1r: 131 mode2r: 2
Counter: 0
Counter: 1
Counter: 2
Counter: 3
Counter: 4
Counter: 5
Counter: 6
Counter: 7
Counter: 8
Counter: 9
```

3 Biblioteca *libquanser* e *Shield*

Com base nos módulos desenvolvidos citados nas seções anteriores e com os esquemáticos em anexo desenvolvemos a *shield* e a biblioteca para seu controle específico. Abaixo, fornecemos uma descrição breve da biblioteca, já que informações detalhadas podem ser encontradas na documentação, e relatamos a construção da PCI.

3.1 *libquanser*

3.1.1 Código em C

A biblioteca *libquanser* possui os módulos para uso de GPIOs, PWM e do LS7366R. Além disso, o arquivo *quanser.c* possui as funções específicas para inicialização e controle dos pinos de IO usados pela *shield*. A estrutura principal usada pela biblioteca é mostrada na Figura 8. A partir dela podemos entender quais pinos são usados e para que.

Figura 8: Estrutura de dados da biblioteca *libquanser*.

```
typedef struct {
    GPIO decoder_en;      // IO8 gpio40
    DECODER decoder;     // SPI bus
    GPIO motor_en;        // IO6 gpio1
    PWM motor;            // IO5 pwm3
    GPIO switch1;         // IO3 gpio14
    GPIO switch2;         // IO4 gpio6
} QUANSER;
```

Os atributos *decoder_en* e *motor_en* são GPIOs nos pinos de IO 8 e 6, respectivamente. Como sugerido pelos seus nomes, eles fazem o controle dos sinais de *enable* dos circuitos integrados LS7366R e L6203.

O atributo *decoder* é uma estrutura responsável por abstrair o uso do LS7366R. Podemos aplicar sobre ela as diversas funções de comunicação desenvolvidas para o LS7366R. No caso específico da biblioteca *libquanser*, a inicialização da biblioteca configura o decodificador com os parâmetros adequados para a leitura dos *encoders*.

O atributo *motor*, do tipo PWM, corresponde ao pino de PWM3, no IO5, disponível na Galileo. A biblioteca, através das funções desenvolvidas no módulo do PWM, implementa funções para acionar o motor em Volts.

Finalmente, os atributos *switch1* e *switch2*, do tipo GPIO, servem para fazer a leitura dos sensores de fim de curso da junta controlada.

Informações mais detalhadas a respeito da biblioteca podem ser encontradas na documentação, produzida através da ferramenta *Doxxygen*.

3.1.2 Script de Inicialização

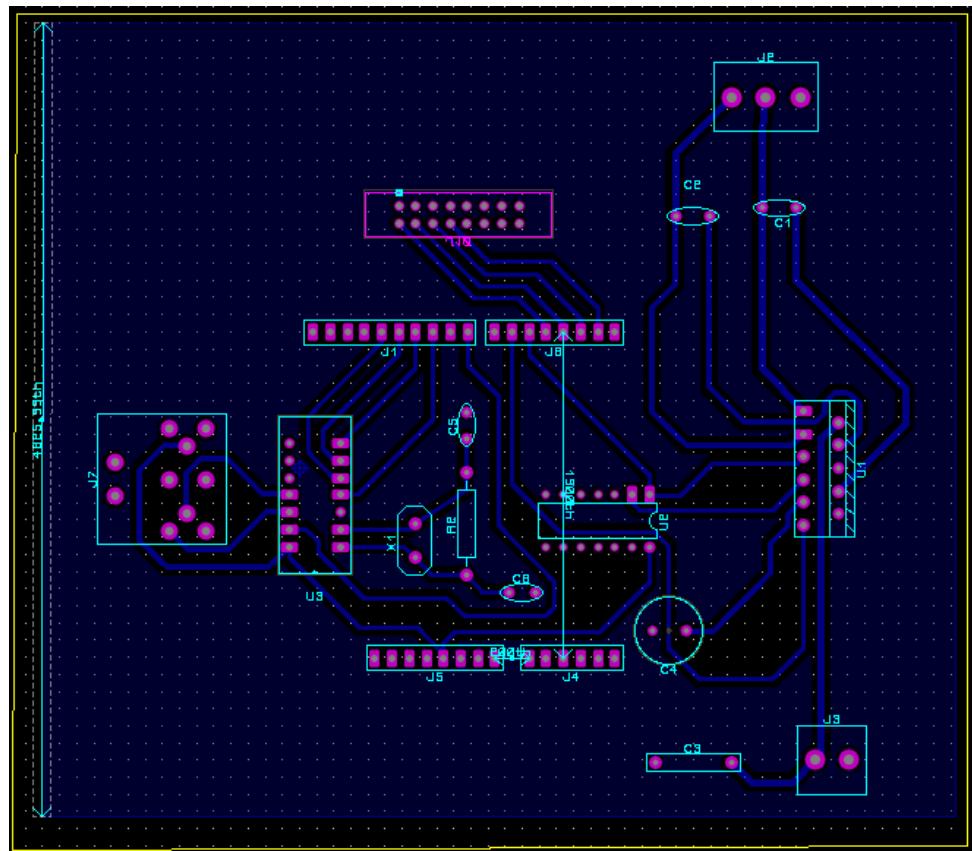
O *script* de inicialização pode ser encontrado no diretório *init.d*. Nele, são inicializados os pinos necessários citados na seção anterior. Além disso, o *script* fornece permissão de acesso aos arquivos necessários para uso da biblioteca a todos os usuários que forem membros do grupo *qnsr*.

O *script* deve ser instalado para execução automática no *boot* da Galileo, o que deve ser feito no *root* com permissões de super usuário. O uso da biblioteca, depois de instalado o *script*, pode ser feito no espaço de usuário, contanto que o usuário faça parte do grupo *qnsr*.

3.2 Desenvolvimento da PCI

O esquemático da *shield* é omitido do relatório, já que pode ser mais facilmente acessado no arquivo de PDF em anexo. Na Figura 9 é mostrado o leiaute da placa correspondente ao esquemático em anexo. O leiaute foi projetado no *software* Proteus, pois este possuía algumas *footprints* já disponíveis, que não encontramos no *gEDA*. Além disso, e mais importante, pudemos realizar a simulação do circuito correspondente ao L6203 através do Proteus.

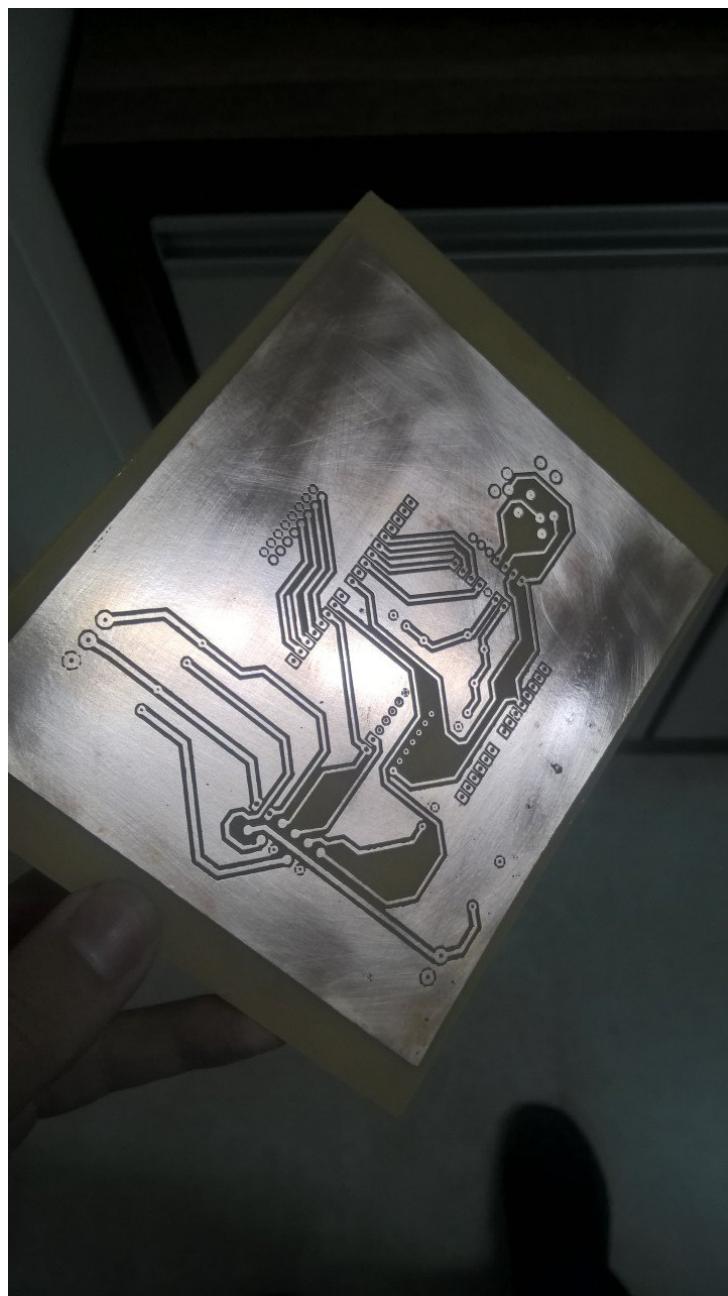
Figura 9: Leiaute da PCI desenvolvida.



Alguns fatores que não foram discutidos anteriormente interessantes de serem citados aqui. Como não encontramos um conector fêmea DIN 5, tivemos que usar um DIN 8 que encontramos em uma placa antiga doada no Laboratório de Graduação. Outro ponto interessante é visto nos conectores J1, J6, J5 e J4. A distância entre cada um deles e a quantidade de pinos foi pensada de tal forma que a placa pudesse ser encaixada diretamente sobre a Galileo.

Assim como com a placa de teste desenvolvida para o L6203, a *shield* foi construída através de transferência térmica e por corrosão com percloreto de ferro. Uma foto da placa de circuito impresso após a corrosão por percloreto é mostrada na Figura 10

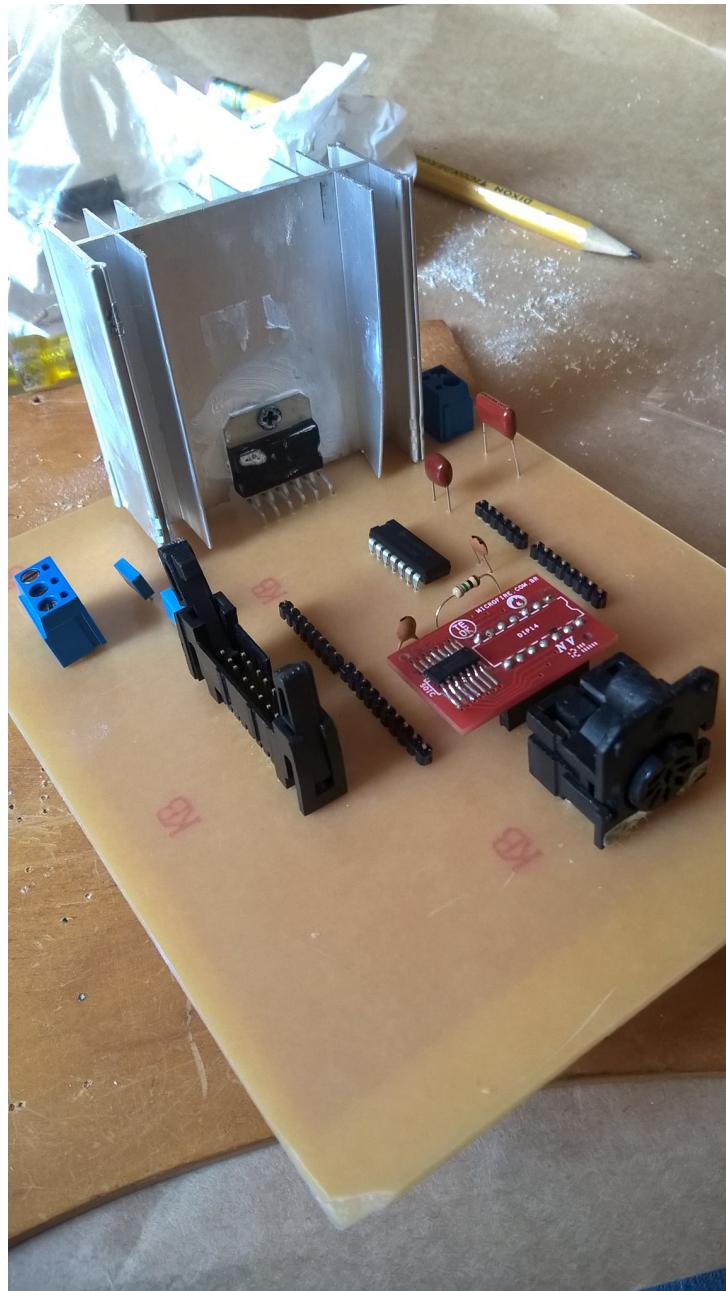
Figura 10: PCI após a corrosão em percloreto de ferro.



Depois de realizada a corrosão, fizemos os furos na placa e soldamos os compo-

tes. O resultado final é mostrado na Figura 11.

Figura 11: PCI da *Shield* em seu estado final.



Com a PCI completa, realizamos os mesmos testes feitos para os circuitos de teste. Obtemos os mesmos resultados, após alguns problemas com componentes em curto circuito, demonstrando o funcionamento da placa. No entanto, não pudemos testar a placa no braço robótico, de tal forma que a validação completa do circuito ainda não foi feita.

4 Conclusão

O objetivo deste trabalho era a aplicação dos conhecimentos adquiridos em sala de aula em um projeto único, unindo a estes também conceitos obtidos em cadeiras anteriores. A movimentação de uma junta do Quanser 2DSFJE incluiu principalmente o domínio sobre a configuração da comunicação via SPI, o uso correto do PWM e o acesso seguro dos pseudo-arquivos usados por estes dispositivos.

O uso do SPI foi de fato a parte mais complexa a ser desenvolvida neste trabalho, uma vez que não conseguimos fazê-la funcionar no laboratório correspondente (Laboratório 8) e por ser peça essencial no desenvolvimento do projeto como um todo. Assim sendo, este foi o que mais demandou tempo e testes.

O PWM e os sensores de fim-de-curso foram, por outro lado, mais simples de serem implementados e testados. Estes dispositivos consistem essencialmente em leitura e escrita de pseudo-arquivos, e a função que demandou maior esforço foi a de conversão de tensão média para *duty-cycle*.

Finalmente, o PID foi desenvolvido usando as funções que pré-processam os dados obtidos do *decoder* e PWM, transformando-os em informações mais fáceis de serem trabalhadas.

Referências

LSI COMPUTER SYSTEMS, INC. *32-BIT QUADRATURE COUNTER WITH SERIAL INTERFACE*. [S.l.], 2015. Citado na página 5.

ST MICROELETROONICS. *DMOS FULL BRIDGE DRIVER*. [S.l.], 2003. Citado na página 8.

TEXAS INSTRUMENTS. *CD4069UB CMOS Hex Inverter*. [S.l.], 2016. Citado na página 10.

THE Normal or Standard PID Algorithm. 2015. Disponível em: <<https://controlguru.com/the-normal-or-standard-pid-algorithm/>>. Citado na página 10.