

Béuren Felipe Bechlin, Gabriel Ammes Pinho, Thiago Martins

Acionamento das juntas do robô Quanser 2DSFJE utilizando a Intel Galileo Gen 2

Porto Alegre

2017

Béuren Felipe Bechlin, Gabriel Ammes Pinho, Thiago Martins

Acionamento das juntas do robô Quanser 2DSFJE utilizando a Intel Galileo Gen 2

Relatório apresentado como requisito parcial
para obtenção de conceito na disciplina de
Microcontroladores (ENG10032).

Universidade Federal do Rio Grande do Sul
Escola de Engenharia
Departamento de Sistemas Elétricos de Automação e Energia
MICROCONTROLADORES - ENG10032

Porto Alegre
2017

Sumário

Introdução	3
1 HARDWARE	4
1.1 Driver de motor DC	4
1.2 Shield Galileo	7
1.3 Imagens das PCIs	10
2 SOFTWARE	11
2.1 Q2DSFJ.h	11
2.2 LS7366R.h	14
2.3 ioutils.h	17
2.4 pid.h	19
2.5 Script de inicialização	21
2.6 Programa de teste	29
3 ANEXOS	32

Introdução

Este projeto consiste no desenvolvimento de um *shield* para a Intel Galileo Gen 2, construído em uma placa de circuito impresso, capaz de acionar as juntas do robô Quanser 2DSFJE. O motor deve ser acionado por PWM, de forma que um ciclo de trabalho de 0% corresponda ao motor girando em velocidade máxima em um sentido, um ciclo de trabalho de 50% corresponda ao motor parado, e um ciclo de trabalho de 100% corresponda ao motor girando em velocidade máxima no sentido contrário. Os *encoders* devem ser decodificados em quadratura, e a contagem deve ser feita em *hardware*, para evitar perdas.

Juntamente com o *hardware*, deve ser desenvolvido um *software* que implemente um controlador PID. O software deve incluir uma biblioteca com uma API a ser utilizada pelo controlador, possibilitando o comando do motor em Volts, a leitura dos *encoders* em radianos, e a leitura do sensor de fim de curso. A biblioteca deve executar no sistema Linux como um programa no espaço de usuário, sem privilégios de superusuário. Os detalhes a respeito do funcionamento do *hardware* e do *software* desenvolvidos serão apresentados nos capítulos seguintes.

1 Hardware

Com o intuito de realizar o controle do motor DC, e de facilitar e organizar as conexões com a Intel Galileo Gen 2, foram construídas duas placas de circuito impresso. Uma delas possui circuito com uma ponte H optacoplada para eliminar e isolar o circuito lógico do circuito de potência. Para a leitura dos *encoders* de quadratura foi escolhido o circuito integrado LSR7366R, que possui interface SPI, diminuindo o número de conexões físicas necessárias para transmissão de dados.

1.1 Driver de motor DC

O chaveamento do motor é realizado por meio de uma ponte H com o uso de MOSFETs canal N. Por sua vez, para chavear os transistores, é utilizado um circuito integrado HIP4081A, que por meio de circuito de *bootstrap* tem tensão suficiente para comutar os transistores de cima da ponte. O módulo com a ponte H pode ser visualizado na Figura 1.

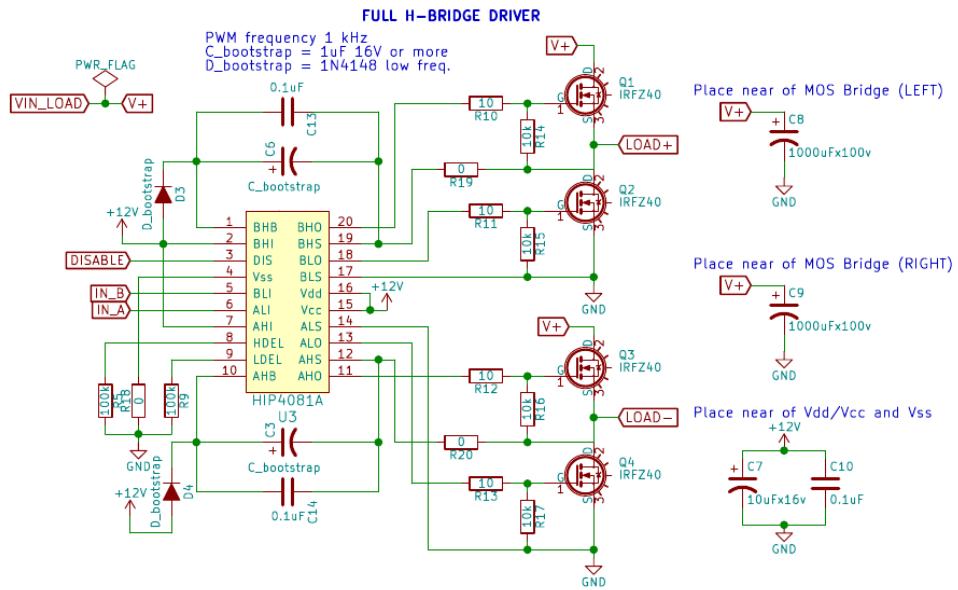


Figura 1 – Esquemático da ponte H comutada pelo circuito integrado HIP4081A.

No entanto, para o correto chaveamento da ponte H, é necessário ter o sinal PWM e o seu sinal negado para comutar os transistores de cada lado da ponte. Para isso, foi utilizado o circuito integrado 74HC14, que conta com 5 inversores com *schmitt trigger* que ajudam na reconstituição do sinal digital. Nessa fase também foram adicionados optoacopladores para isolar os sinais de entrada de todo o circuito de *driver* com o uso

do circuito integrado 4N25. O sinal de ativação da ponte também está optoacoplado e naturalmente em nível baixo caso não tenha sido forçado a outro valor, assim é pré-condicionado o estado desabilitado. O esquemático do módulo pode ser visualizado na Figura 2.

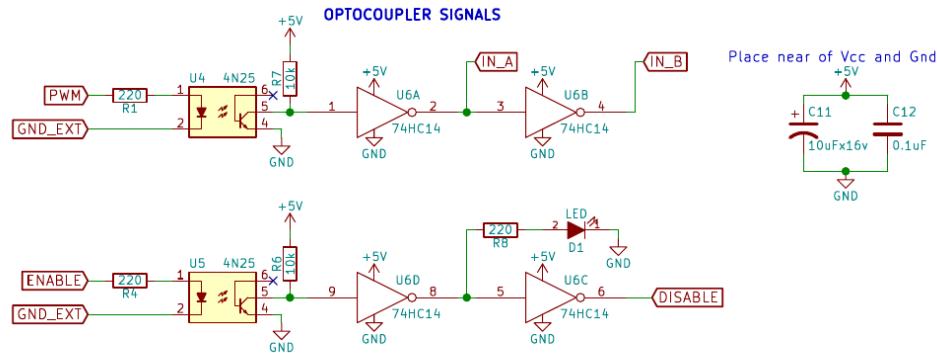


Figura 2 – Esquemático de lógica de inversão do PWM e enable com optoacopladores.

Para a alimentação dos circuitos integrados foram utilizados reguladores de tensão fixa de 12V e 5V. Os reguladores não são necessariamente alimentados pela mesma tensão de entrada que alimenta a ponte H, e isso pode ser alterado por um *jumper* disponível na placa.

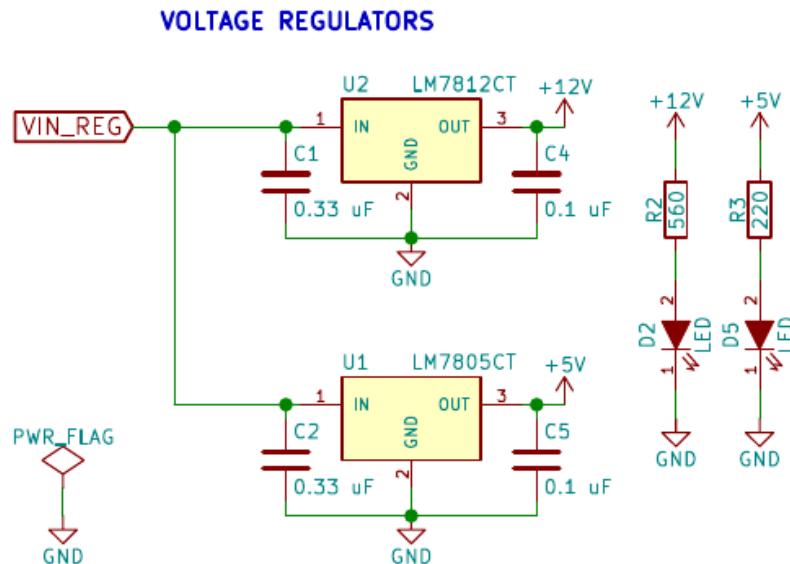


Figura 3 – Esquemático do módulo de alimentação dos circuitos integrados.

Esses módulos estão em uma placa de circuito impresso de fenólite de face simples. O *layout* da camada de cobre de baixo e a camada de serigrafia estão representados nas Figuras 4 e 5, respectivamente.

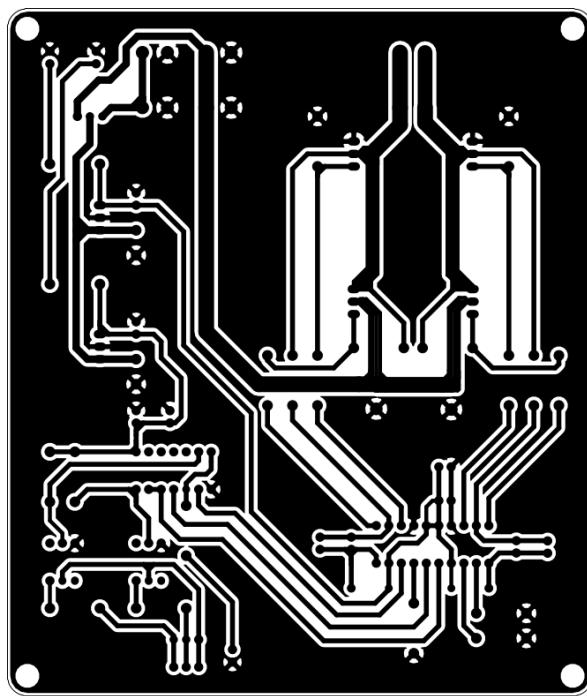


Figura 4 – Camada de cobre da PCI.

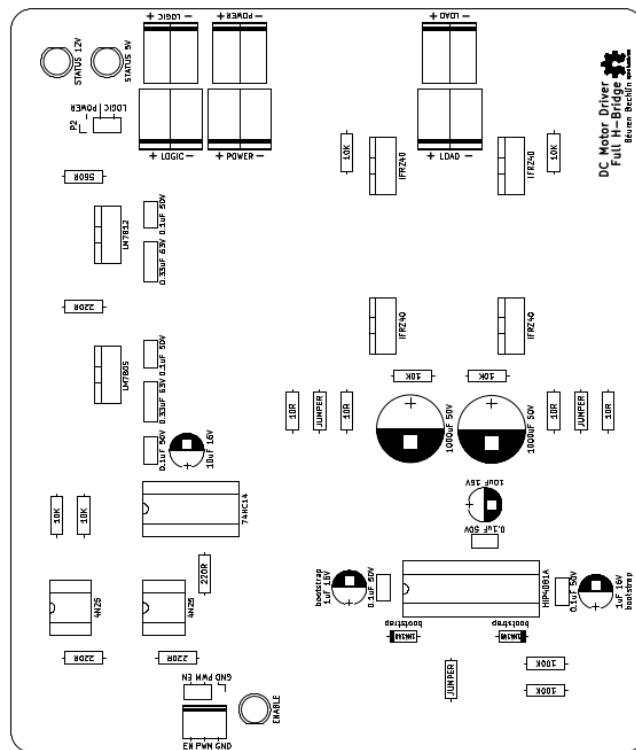


Figura 5 – Camada de serigrafia da PCI.

1.2 Shield Galileo

Na placa com o *shield* para a Intel Galileo foram adicionados os contadores de *encoder* de quadratura, circuitos integrados LS7366R, ligados no mesmo barramento SPI somente com os seletores diferentes para poder selecionar um por vez no barramento. Além disso foram usados conectores do tipo molex para organizar e facilitar o uso das GPIOs que foram utilizadas pelo projeto. Abaixo se encontra a tabela de GPIOs que foram utilizadas no projeto.

<i>IO3</i>	PWM	Controle de chaveamento da ponte H
<i>IO4</i>	SAÍDA	Sinal de habilitação da ponte H
<i>IO5</i>	ENTRADA	Leitura dos sensor 1 de fim de curso
<i>IO6</i>	ENTRADA	Leitura dos sensor 2 de fim de curso
<i>IO7</i>	SAÍDA	Sinal de habilitação de contagem do contador 1
<i>IO8</i>	SAÍDA	Sinal de <i>chip select</i> do contador 1
<i>IO9</i>	SAÍDA	Sinal de habilitação de contagem do contador 2
<i>IO10</i>	SAÍDA	Sinal de <i>chip select</i> do contador 1

Tabela 1 – Especificação das GPIOs usadas.

Essa descrição também está presente no esquemático do *shield*, como pode ser visto na Figura 6.

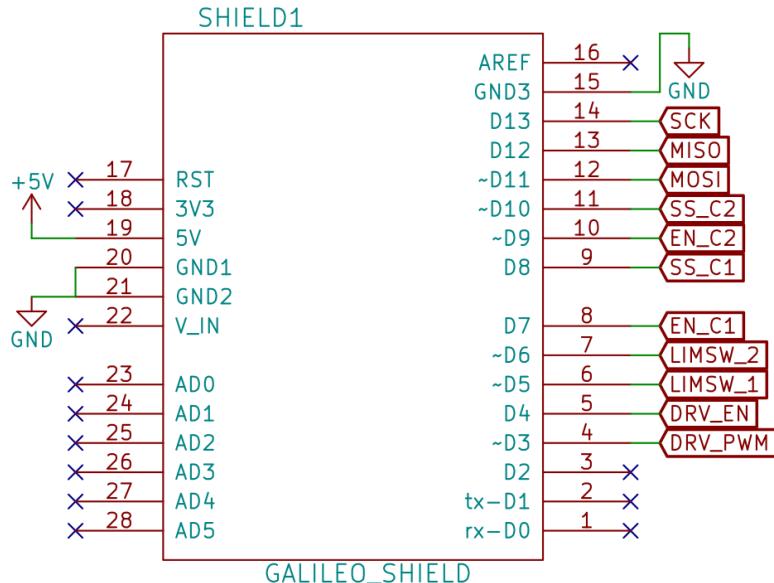


Figura 6 – Esquemático com a definição da interface.

Os leitores/decodificadores de *encoder* estão conectados por soquetes sobre a placa, mas devido a uma falha na aquisição dos componentes, foi necessário usar um adaptador, já

que os circuitos integrados são do formato SMD. Esse circuito integrado realiza contagens até 32 bits, mas seus registradores de contagem podem ser configurados para 8, 16, 24 e 32 bits. Devido a nossa necessidade, utilizamos 16 bits. Toda vez que o LS7366R é *resetado* seus registradores de configuração também são; logo, sempre no início de operações é interessante reenviar a configuração para cada registrador. Na Figura 7 pode ser visto que ambos estão sobre o mesmo barramento e com osciladores independentes para sua conexão. Cada contador de *encoder* está ligado a um molex que possui o sinal A, B e GND.

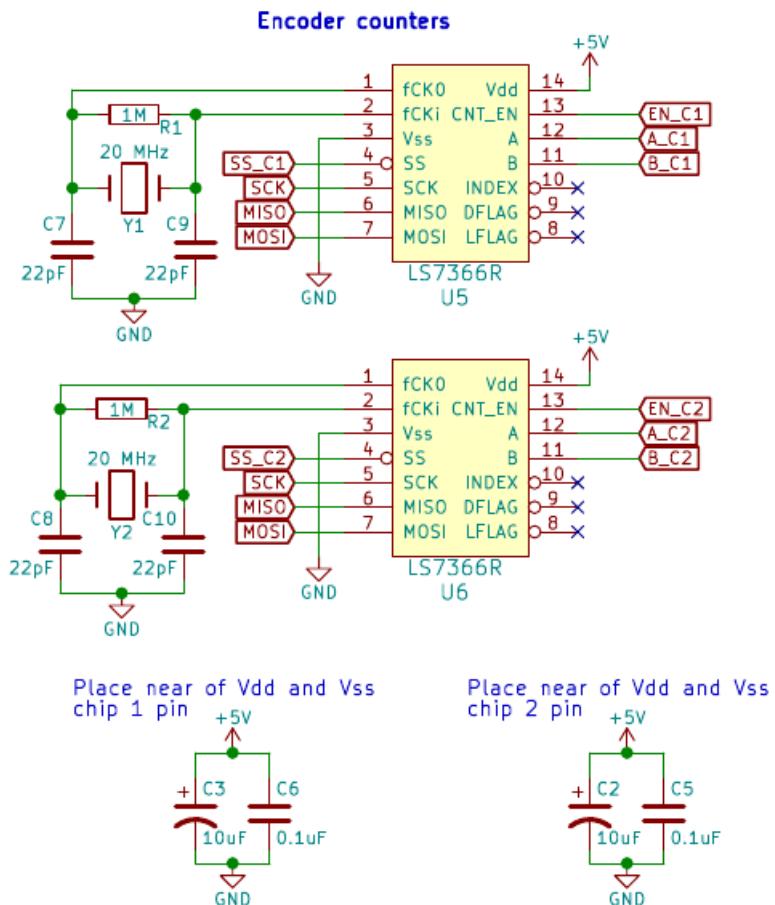


Figura 7 – Esquemático com os decodificadores de encoder.

Essa PCI também foi desenvolvida utilizando placa de fenolite virgem de face simples e ambas as placas foram fresadas com o auxílio da fresadora de PCI "João de Barro" do laboratório CTA-IF-UFRGS. O *layout* da camada de cobre de trás e também a camada de serigrafia estão nas figuras abaixo.

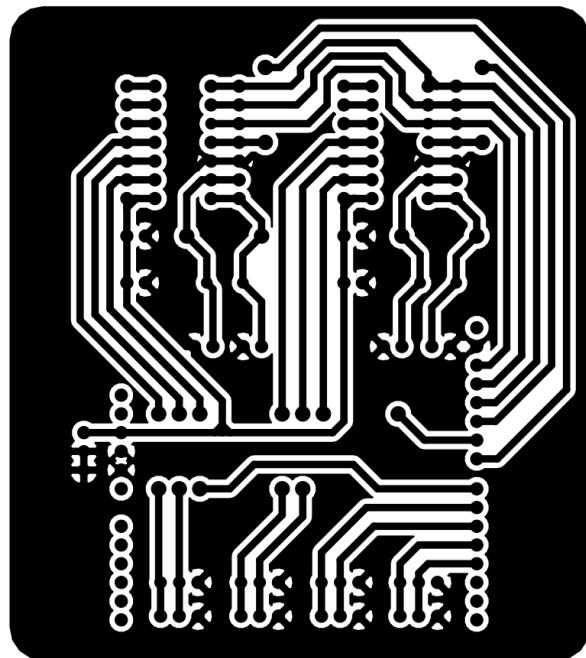


Figura 8 – Camada de cobre da PCI.

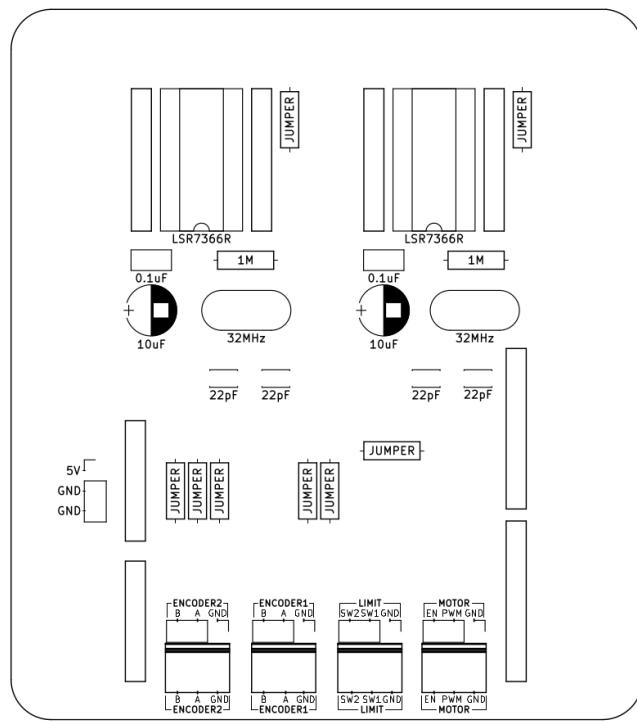


Figura 9 – Camada de serigrafia da PCI.

1.3 Imagens das PCIs

Abaixo estão as fotos das placas de circuito impresso depois de soldados os componentes.

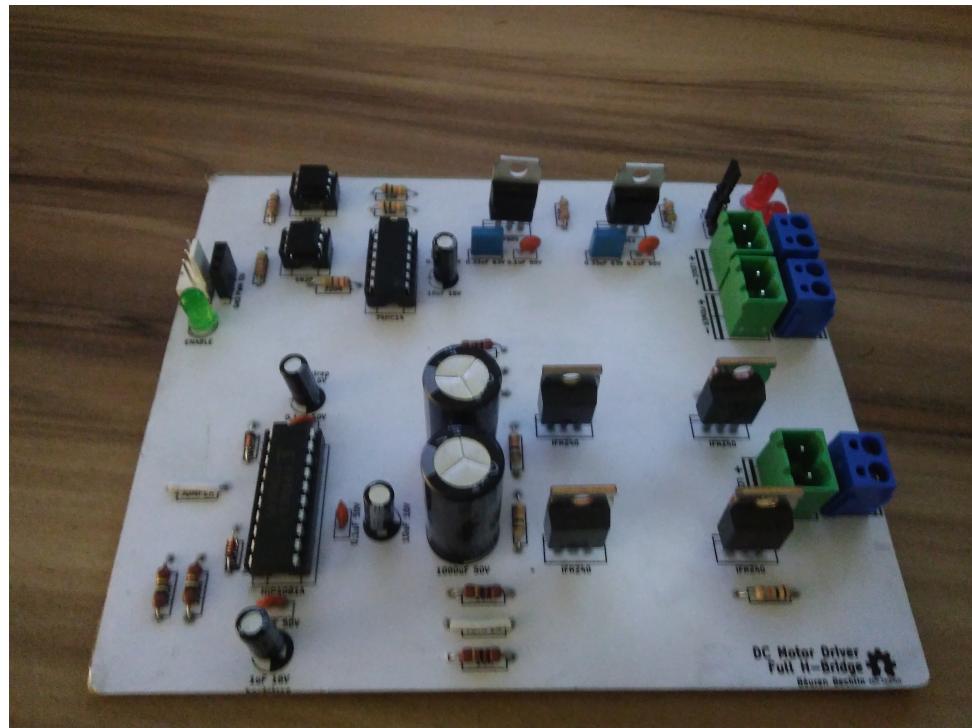


Figura 10 – Camada de cobre da PCI.

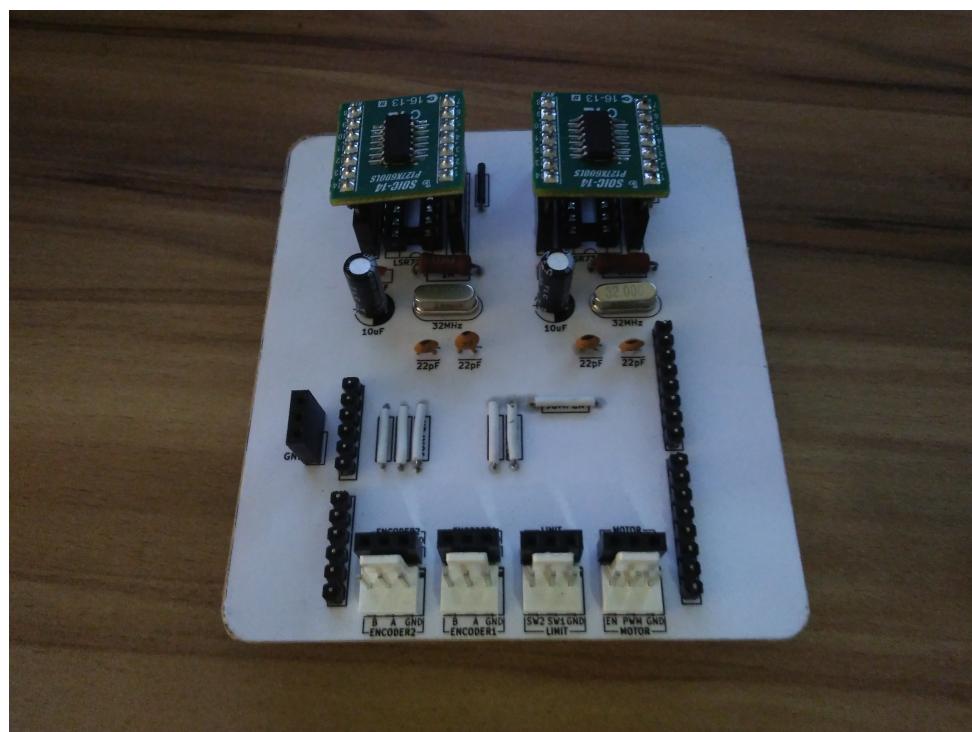


Figura 11 – Camada de serigrafia da PCI.

2 Software

Foi desenvolvida uma biblioteca a nível de usuário para utilização do projeto. Essa biblioteca consiste basicamente em três módulos:

- **Q2DSFJ:** encapsula todas as funcionalidades necessárias para controle e acionamento de motor.
- **7366R:** biblioteca para comunicação com os leitores/decodificadores de quadratura segundo o *datasheet*. Possui todos os códigos de configurações e padronização da comunicação do circuito integrado.
- **ioutils:** funcionalidades para facilitar as operações com GPIOS.

2.1 Q2DSFJ.h

Abaixo segue algumas definições do módulo.

```
#define COUNTS_PER_REV 4096
#define MATH_PI      3.14159265358979323846264338327950288
#define PWM_FREQ     1000
#define HALF_DUTY_CYCLE 500000

#define Q2DSFJ_ENC_1  0
#define Q2DSFJ_ENC_2  1

#define Q2DSFJ_LIMSW_1 0
#define Q2DSFJ_LIMSW_2 1

#define TRUE         1
#define FALSE        0

struct _Q2DSFJ {
    /* File handle for motor enable pin */
    int mot_en;
    /* File handle for motor pwm pin */
    int mot_pwm;
    /* File handle for enable galileo pwm */
    int pwm_en;
    /* File handles for limit switch pins */
}
```

```

int lim_sw[2];
/* File handles for encoder counter enable */
int enc_en[2];
/* File handles for encoder counter chip select */
int enc_cs[2];
/* File handle for spi communication */
int spi;
};

typedef struct _Q2DSFJ Q2DSFJ_T;

```

- struct _Q2DSFJ; typedef struct _Q2DSFJ Q2DSFJ_T;
 - Descrição: estrutura de dados internos que salva os *handles* dos arquivos necessários para leitura. Também conta com uma definição de tipo para facilitar em sua declaração.
- int Q2DSFJ_initialize(Q2DSFJ_T* this);
 - Descrição: Inicializa os *handles* dos arquivos das GPIOs e comunicação SPI, configura o modo de operação do LS7366R que realiza a contagem dos *encoders*, zerando seus registradores de contagem; habilita e configura o PWM, entranto desabilita o motor.
 - Retorno: 1, caso a inicialização tenha ocorrido com sucesso, ou 0, em caso de erro.
- int Q2DSFJ_reset_enc_counter(Q2DSFJ_T* this, int enc_number);
 - Descrição: Reinicia os registradores utilizados nos contadores de um *encoder*.
 - Argumentos:
 - * *enc_number*: especifica o número do *encoder*.
 - Retorno: 1, caso a reinicialização tenha ocorrido com sucesso, ou 0, em caso de erro.
- int Q2DSFJ_enable_enc_counter(Q2DSFJ_T* this, int enc_number, int enable);
 - Descrição: Habilita ou desabilita os registradores utilizados nos contadores de um *encoder*.
 - Argumentos:
 - * *enc_number*: especifica o número do *encoder*.

- * `enable`: indica se os registradores devem ser habilitados, enable igual a 1, ou desabilitados, enable igual a 0.
 - Retorno: 1, em caso de sucesso, ou 0, em caso de erro.
- `int Q2DSFJ_read_enc_counter_raw(Q2DSFJ_T* this, int enc_number);`
 - Descrição: Faz a leitura do contador de um determinado *encoder*.
 - Argumentos:
 - * `enc_number`: especifica o número do *encoder*.
 - Retorno: Valor contido no registrador, sem qualquer conversão. Em caso de erro, a função retorna 0.
- `int Q2DSFJ_read_enc_counter(Q2DSFJ_T* this, int enc_number);`
 - Retorno: Faz a leitura do contador de um determinado *encoder*.
 - Argumentos:
 - * `enc_number`: especifica o número do *encoder*.
 - Retorno: Valor contido no registrador, convertido para radianos. Em caso de erro, a função retorna 0.
- `int Q2DSFJ_read_lim_switch(Q2DSFJ_T* this, int limsw);`
 - Descrição: Realiza a leitura digital dos sensores de fim de curso. Importante lembrar que eles estão normalmente em nível alto e só quando o sensor chega no fim de curso o nível lógico se torna zero.
 - Argumentos:
 - * `limsw`: especifica o número do *limit switch*.
 - Retorno: 1, caso o motor esteja no fim de curso, e 0, caso não esteja.
- `int Q2DSFJ_enable_motor(Q2DSFJ_T* this, int enable);`
 - Descrição: Habilita ou desabilita o motor.
 - Argumentos:
 - * `new_voltage`: especifica a tensão desejada, podendo variar entre -27V e 27V.
 - * `enable`: indica se o motor deve ser habilitado, enable igual a 1, ou desabilitado, enable igual a 0.
 - Retorno: 1, em caso de sucesso, ou 0, em caso de erro.
- `int Q2DSFJ_set_motor_voltage(Q2DSFJ_T* this, float new_voltage);`
 - Descrição: Define o valor da tensão do motor.

- Argumentos:
 - * `new_voltage`: especifica a tensão desejada, podendo variar entre -27V e 27V.
 - Retorno: 1, em caso de sucesso, ou 0, em caso de erro.
- `int Q2DSFJ_motor_home(Q2DSFJ_T* this);`
 - Descrição: Aciona o motor para que vá lentamente até o limite esquerdo, e quando o limite for atingido, reinicia os contadores dos encoders.

2.2 LS7366R.h

Esse *header* conta com um conjunto de definições para comunicação e operação com o LS7366R. Como mencionado anteriormente, esse circuito integrado possui comunicação SPI e um pequeno conjunto de registradores visíveis externamente. São eles:

- DTR: usado para guardar os dados do MOSI, útil para transferir algum valor para o registrador de CNTR. É configurável com 8, 16, 24 e 32 bits.
- CNTR registrador de contagem, onde é salvo a contagem do encoder decodificado em quadratura. É configurável com 8, 16, 24 e 32 bits.
- STR registrador de condição do circuito integrado, ou seja, *carry flag*, *borrow flag* entre outros. Registrador de 8 bits.
- MDR0 Registrador que guarda os dados de configuração do contador.
- MDR1 Segundo registrador que guarda dados de configuração.

Lista de códigos de configuração e instruções suportadas pelo LS7366R

```

/* *** MDR0 configuration data ***

/* Count modes */
#define NQUAD      0x00 //non-quadrature mode
#define QUADRX1    0x01 //X1 quadrature mode
#define QUADRX2    0x02 //X2 quadrature mode
#define QUADRX4    0x03 //X4 quadrature mode
#define COUNT_MODE  0x03 //bits location in register
#define COUNT_SHIFT 0    //bit shift in register

/* Running modes */
#define FREE_RUN   0x00
#define SINGE_CYCLE 0x04

```

```
#define RANGE_LIMIT 0x08
#define MODULO_N 0x0C
#define RUNNING_MODE 0x0C //bits location in register
#define RUNNING_SHIFT 2 //bit shift in register

/* Index modes */
#define DISABLE_INDX 0x00 //index_disabled
#define INDX_LOADC 0x10 //index_load_CNTR
#define INDX_RESETC 0x20 //index_rest_CNTR
#define INDX_LOADO 0x30 //index_load_OL
#define ASYNCH_INDX 0x00 //asynchronous index
#define SYNCH_INDX 0x40 //synchronous index
#define INDEX_MODE 0x70 //bits location in register
#define INDEX_SHIFT 4 //bit shift in register

/* Clock filter modes */
#define FILTER_1 0x00 //filter clock frequency division factor 1
#define FILTER_2 0x80 //filter clock frequency division factor 2
#define FILTER_MODE 0x80 //bits location in register
#define FILTER_SHIFT 7 //bit shift in register

/* **MDR1 configuration data*** */

/* Flag modes */
#define NO_FLAGS 0x00 //all flags disabled
#define IDX_FLAG 0x10 //IDX flag
#define CMP_FLAG 0x20 //CMP flag
#define BW_FLAG 0x40 //BW flag
#define CY_FLAG 0x80 //CY flag
#define FLAG_MODE 0xF0 //bits location in register
#define FLAG_SHIFT 4 //bit shift in register

/* 1 to 4 bytes data-width */
#define BYTE_4 0x00 //four byte mode
#define BYTE_3 0x01 //three byte mode
#define BYTE_2 0x02 //two byte mode
#define BYTE_1 0x03 //one byte mode
#define DATA_WID_MODE 0x03 //bits location in register
#define DATA_WID_SHIFT 0 //bit shift in register

/* Enable/disable counter */
#define EN_CNTR 0x00 //counting enabled
```

```

#define DIS_CNTR    0x04 //counting disabled
#define COUNT_EN_MODE 0x04 //bits location in register
#define COUNT_EN_SHIFT 2 //bit shift in register

/* LS7366R op-code list */
#define CLR_MDR0    0x08
#define CLR_MDR1    0x10
#define CLR_CNTR    0x20
#define CLR_STR     0x30
#define READ_MDR0   0x48
#define READ_MDR1   0x50
#define READ_CNTR   0x60
#define READ_OTR    0x68
#define READ_STR    0x70
#define WRITE_MDR1  0x90
#define WRITE_MDR0  0x88
#define WRITE_DTR   0x98
#define LOAD_CNTR   0xE0
#define LOAD_OTR    0xE4

```

- int LS7366R_init(int spi_fd);
 - Descrição: Inicializa os parâmetros da comunicação SPI suportados pelo LS7366R.
 - Argumentos:
 - * *spi_fd*: *handle* do arquivo de SPI.
 - Retorno: 1, caso a inicialização tenha ocorrido com sucesso, ou 0, em caso de erro.
- int LS7366R_clear(int spi_fd, int cs_fd, unsigned char op_code);
 - Descrição: instruções sem operandos realizadas sobre os registradores do LS7366R.
 - Argumentos:
 - * *spi_fd*: *handle* do arquivo de SPI.
 - * *cs_fd*: *handle* do arquivo de *chip select*.
 - * *op_code*: código da instrução.
 - Retorno: 1, em caso de sucesso, ou 0, em caso de erro.
- int LS7366R_write(int spi_fd, int cs_fd, unsigned char op_code, unsigned char data);
 - Descrição: Escreve um byte em uma instrução de chamada de escrita para um registrador.

- Argumentos:
 - * `spi_fd`: *handle* do arquivo de SPI.
 - * `cs_fd`: *handle* do arquivo de *chip select*.
 - * `op_code`: código da instrução.
 - * `data`: dado que deseja ser escrito no registrador.
- Retorno: 1, em caso de sucesso, ou 0, em caso de erro.
- `int LS7366R_read(int spi_fd, int cs_fd, unsigned char op_code, int bytes);`
 - Descrição: Lê determinado número de bytes em uma instrução de chamada de leitura para um registrador.
 - Argumentos:
 - * `spi_fd`: *handle* do arquivo de SPI.
 - * `cs_fd`: *handle* do arquivo de *chip select*.
 - * `op_code`: código da instrução.
 - * `bytes`: número de bytes que deseja ser lido.
 - Retorno: número lido transformado para int.

2.3 ioutils.h

- `int fd_open(int *fd, const char *path, int mode);`
 - Descrição: Abre um descritor de arquivo da GPIO.
 - Argumentos:
 - * `fd`: *handle* do arquivo.
 - * `path`: indica o caminho do arquivo.
 - * `mode`: indica se o arquivo será aberto para leitura, escrita, ou leitura/escrita.
 - Retorno: 1, em caso de sucesso, ou 0, em caso de erro.
- `int digital_write(int pin_fd, int data);`
 - Descrição: Define o valor lógico de uma saída GPIO.
 - Argumentos:
 - * `pin_fd`: *handle* do arquivo de IO.
 - * `data`: valor lógico a ser escrito.
 - Retorno: 1, em caso de sucesso, ou 0, em caso de erro.
- `int digital_read(int pin_fd);`

- Descrição: Lê o valor lógico de uma entrada GPIO.
- Argumentos:
 - * `pin_fd`: *handle* do arquivo de IO.
- Retorno: Valor contido no arquivo.
- `int analog_read_raw(int pin_fd);`
 - Descrição: Lê o valor de uma entrada GPIO analógica.
 - Argumentos:
 - * `pin_fd`: *handle* do arquivo de IO.
 - Retorno: Valor contido no arquivo.
- `int analog_read_scaled(int pin_fd, int scale_fd);`
 - Descrição: Lê o valor escalado de uma entrada GPIO analógica.
 - Argumentos:
 - * `pin_fd`: *handle* para o descritor do arquivo.
 - * `scale_fd`: *handle* do arquivo de escala do ADC.
 - Retorno: Valor contido no arquivo especificado em `pin_fd` multiplicado pelo valor lido em `scale_fd`.
- `float analog_read_scaled_5v(int pin_fd, int scale_fd);`
 - Descrição: Lê o valor escalado para 5V de uma entrada GPIO analógica.
 - Argumentos:
 - * `pin_fd`: *handle* para o descritor do arquivo.
 - * `scale_fd`: *handle* do arquivo de escala do ADC.
 - Retorno: Valor contido no arquivo especificado em `pin_fd`, adaptado em uma escala de 0 a 5V.
- `int pwm_set_frequency(float freq);`
 - Descrição: Define a frequência do PWM.
 - Argumentos:
 - * `freq`: especifica a frequência, em hertz.
 - Retorno: 1, em caso de sucesso, ou 0, em caso de erro.
- `int pwm_set_period(int period);`
 - Descrição: Define o período do PWM.

- Argumentos:
 - * **period**: especifica o período desejado.
- Retorno: 1, em caso de sucesso, ou 0, em caso de erro.
- **int pwm_set_duty(int duty_fd, int duty_cycle);**
 - Descrição: Define o ciclo de trabalho do PWM.
 - Argumentos:
 - * **duty_fd**: *handle* do arquivo de IO.
 - * **duty_cycle**: especifica o ciclo de trabalho.
 - Retorno: 1, em caso de sucesso, ou 0, em caso de erro.
- **int pwm_set_duty_percent(int duty_fd, int freq, float percent);**
 - Descrição: Define o ciclo de trabalho do PWM, em porcentagem.
 - Argumentos:
 - * **duty_fd**: *handle* do arquivo de IO.
 - * **freq**: especifica a frequência, em hertz.
 - * **percent**: percentual do ciclo de trabalho.
 - Retorno: 1, em caso de sucesso, ou 0, em caso de erro.

2.4 pid.h

```
#define MAX_BOUNDARY 0
#define MIN_BOUNDARY 1

struct _pid{
    /* */
    float set_point;

    /* Constants PID*/
    float kp;
    float ki;
    float kd;

    /* Saturation Control*/
    int max_flag;
    float max;
    int min_flag;
```

```

float min;

/* Internal state variables*/
float integrated;
float _last_sample;
};

typedef struct _pid PID_T;

```

- struct _pid

- Descrição: estrutura de dados internos que salva parâmetros e variáveis internas para implementação do algoritmo de controle PID. Também conta com uma definição de tipo para facilitar em sua declaração.

- int pid_init(PID_T* pid, float kp, float ki, float kd);

- Descrição: inicialização de parâmetros internos do algoritmo já definindo as constantes que são mostradas abaixo.

- Argumentos:

- * **pid**: referência das informações do PID.
- * **kp**: constante proporcional.
- * **ki**: constante integrativa.
- * **kd**: constante derivativa.

- Retorno: 1, em caso de sucesso, ou 0, em caso de erro.

- int pid_set_boundary(PID_T* pid, int mode, float value);

- Descrição: altera condições de fronteira do pid, ou seja, insere uma saturação em algum dos extremos.

- Argumentos:

- * **pid**: referência das informações do PID.
- * **mode**: qual extremo se deseja inserir uma saturação, onde *MAX_BOUNDARY* indica saturação de máximo valor e *MIN_BOUNDARY* indica saturação de mínimo valor.
- * **value**: valor da saturação.

- Retorno: 1, em caso de sucesso, ou 0, em caso de erro.

- int pid_setpoint(PID_T* pid, float new_set_point);

- Descrição: Altera o *setpoint* do PID, ou seja, o ponto em que se deseja alcançar.
 - Argumentos:
 - * **pid**: referência das informações do PID.
 - * **new_set_point**: novo ponto de chegada para o PID.
 - Retorno: 1, em caso de sucesso, ou 0, em caso de erro.
- **int pid_compute(PID_T* pid, float dt, float sample);**
- Descrição: computação do PID levando em consideração o tempo passado entre essa nova medida e a passada.
 - Argumentos:
 - * **pid**: referência das informações do PID.
 - * **dt**: variação de tempo entre a última computação e essa que será realizada.
 - * **sample**: medida de *feedback* para o algoritmo.
 - Retorno: sinal de

2.5 Script de inicialização

O script de inicialização é necessário para configurar os *muxes* internos da Intel Galileo para rotear o sinal corretamente dos pinos de entrada e saída. Importante lembrar que existe uma regra no *Makefile* para enviar diretamente esse arquivo para a pasta */etc/init.d/* onde ficam os *scripts* de inicialização de serviços do sistema operacional. Essa regra depende de dois parâmetros e pode ser visualizada no trecho abaixo, retirado do *Makefile*.

```
...
GALILEO=galileo1
USER=root
...
init_script: init_libQ2DSFJ
  scp $^ $(USER)@$GALILEO:/etc/init.d/
```

Abaixo se o *script* de inicialização desenvolvido.

```
#!/bin/sh
### BEGIN INIT INFO
# Provides:          Projeto Final
# Required-Start:
# Should-Start:
# Required-Stop:
```

```
# Default-Start:    S
# Default-Stop:
# Short-Description: Configura GPIO, PWM e SPI para o Projeto Final.
# Description:      Configura GPIO, PWM e SPI para o Projeto Final.
# Beuren Bechlin, Gabriel Ammes, Thiago Martins
### END INIT INFO

case "$1" in
    start|restart|force-reload)
        ######
        # IO3 (PWM)
        if [ ! -d /sys/class/pwm/pwmchip0/pwm1 ] ; then
            echo -n "1" > /sys/class/pwm/pwmchip0/export
        fi
        chgrp pwm /sys/class/pwm/pwmchip0/device/pwm_period
        chmod g+w /sys/class/pwm/pwmchip0/device/pwm_period
        chgrp pwm /sys/class/pwm/pwmchip0/pwm1/duty_cycle
        chmod g+w /sys/class/pwm/pwmchip0/pwm1/duty_cycle
        chgrp pwm /sys/class/pwm/pwmchip0/pwm1/enable
        chmod g+w /sys/class/pwm/pwmchip0/pwm1/enable

        # gpio16 = 0 = out
        if [ ! -d /sys/class/gpio/gpio16 ] ; then
            echo -n "16" > /sys/class/gpio/export
        fi
        echo -n "out" > /sys/class/gpio/gpio16/direction
        echo -n "0" > /sys/class/gpio/gpio16/value

        # gpio17 = in = no pull-up nor pull-down
        if [ ! -d /sys/class/gpio/gpio17 ] ; then
            echo -n "17" > /sys/class/gpio/export
        fi
        echo -n "in" > /sys/class/gpio/gpio17/direction

        # gpio76 = 0
        if [ ! -d /sys/class/gpio/gpio76 ] ; then
            echo -n "76" > /sys/class/gpio/export
        fi
        echo -n "0" > /sys/class/gpio/gpio76/value

        # gpio64 = 1 */
        if [ ! -d /sys/class/gpio/gpio64 ] ; then
```

```
        echo -n "64" > /sys/class/gpio/export
    fi
    echo -n "1" > /sys/class/gpio/gpio64/value

#####
# IO4 (SAIDA)
if [ ! -d /sys/class/gpio/gpio6 ] ; then
    echo -n "6" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio6/direction
chgrp gpio /sys/class/gpio/gpio6/value
chmod g+rwx /sys/class/gpio/gpio6/value

# Level Shifter GPIO
if [ ! -d /sys/class/gpio/gpio36 ] ; then
    echo -n "36" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio36/direction
echo -n "0" > /sys/class/gpio/gpio36/value

#####
# IO5 (ENTRADA)
if [ ! -d /sys/class/gpio/gpio0 ] ; then
    echo -n "0" > /sys/class/gpio/export
fi
echo -n "in" > /sys/class/gpio/gpio0/direction
chgrp gpio /sys/class/gpio/gpio0/value
chmod g+rwx /sys/class/gpio/gpio0/value

# Level Shifter GPIO
if [ ! -d /sys/class/gpio/gpio18 ] ; then
    echo -n "18" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio18/direction
echo -n "1" > /sys/class/gpio/gpio18/value

# Pin Mux 1 GPIO
if [ ! -d /sys/class/gpio/gpio66 ] ; then
    echo -n "66" > /sys/class/gpio/export
fi
echo -n "0" > /sys/class/gpio/gpio66/value
```

```
#####
# IO6 (ENTRADA)
if [ ! -d /sys/class/gpio/gpio1 ] ; then
    echo -n "1" > /sys/class/gpio/export
fi
echo -n "in" > /sys/class/gpio/gpio1/direction
chgrp gpio /sys/class/gpio/gpio1/value
chmod g+rws /sys/class/gpio/gpio1/value

# Level Shifter GPIO
if [ ! -d /sys/class/gpio/gpio20 ] ; then
    echo -n "20" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio20/direction
echo -n "1" > /sys/class/gpio/gpio20/value

# Pin Mux 1 GPIO
if [ ! -d /sys/class/gpio/gpio68 ] ; then
    echo -n "68" > /sys/class/gpio/export
fi
echo -n "0" > /sys/class/gpio/gpio68/value

#####
# IO7 (SAIDA)
if [ ! -d /sys/class/gpio/gpio38 ] ; then
    echo -n "38" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio38/direction
chgrp gpio /sys/class/gpio/gpio38/value
chmod g+rws /sys/class/gpio/gpio38/value

#####
# IO8 (SAIDA)
if [ ! -d /sys/class/gpio/gpio40 ] ; then
    echo -n "40" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio40/direction
chgrp gpio /sys/class/gpio/gpio40/value
chmod g+rws /sys/class/gpio/gpio40/value

#####
# IO9 (SAIDA)
```

```
if [ ! -d /sys/class/gpio/gpio4 ] ; then
    echo -n "4" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio4/direction
chgrp gpio /sys/class/gpio/gpio4/value
chmod g+rws /sys/class/gpio/gpio4/value

# Level Shifter GPIO
if [ ! -d /sys/class/gpio/gpio22 ] ; then
    echo -n "22" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio22/direction
echo -n "0" > /sys/class/gpio/gpio22/value

# Pin Mux 1 GPIO
if [ ! -d /sys/class/gpio/gpio70 ] ; then
    echo -n "70" > /sys/class/gpio/export
fi
echo -n "0" > /sys/class/gpio/gpio70/value

#####
# IO10 (SAIDA)
if [ ! -d /sys/class/gpio/gpio10 ] ; then
    echo -n "10" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio10/direction
chgrp gpio /sys/class/gpio/gpio10/value
chmod g+rws /sys/class/gpio/gpio10/value

# Level Shifter GPIO
if [ ! -d /sys/class/gpio/gpio26 ] ; then
    echo -n "26" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio26/direction
echo -n "0" > /sys/class/gpio/gpio26/value

# Pin Mux 1 GPIO
if [ ! -d /sys/class/gpio/gpio74 ] ; then
    echo -n "74" > /sys/class/gpio/export
fi
echo -n "0" > /sys/class/gpio/gpio74/value
```

```
#####
# IO11 (SPI)
# Level Shifter GPIO
if [ ! -d /sys/class/gpio/gpio24 ] ; then
    echo -n "24" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio24/direction
echo -n "0" > /sys/class/gpio/gpio24/value

# 22k Pull-Up GPIO
if [ ! -d /sys/class/gpio/gpio25 ] ; then
    echo -n "25" > /sys/class/gpio/export
fi
echo -n "in" > /sys/class/gpio/gpio25/direction

# Pin Mux 1 GPIO
if [ ! -d /sys/class/gpio/gpio44 ] ; then
    echo -n "44" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio44/direction
echo -n "1" > /sys/class/gpio/gpio44/value

# Pin Mux 2 GPIO
if [ ! -d /sys/class/gpio/gpio72 ] ; then
    echo -n "72" > /sys/class/gpio/export
fi
echo -n "0" > /sys/class/gpio/gpio72/value

#####
# IO12 (SPI)
# Level Shifter GPIO
if [ ! -d /sys/class/gpio/gpio42 ] ; then
    echo -n "42" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio42/direction
echo -n "1" > /sys/class/gpio/gpio42/value

# 22k Pull-Up GPIO
if [ ! -d /sys/class/gpio/gpio43 ] ; then
    echo -n "43" > /sys/class/gpio/export
fi
echo -n "in" > /sys/class/gpio/gpio43/direction
```

```
#####
# IO13 (SPI)
# Level Shifter GPIO
if [ ! -d /sys/class/gpio/gpio30 ] ; then
    echo -n "30" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio30/direction
echo -n "0" > /sys/class/gpio/gpio30/value

# 22k Pull-Up GPIO
if [ ! -d /sys/class/gpio/gpio31 ] ; then
    echo -n "31" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio31/direction
echo -n "0" > /sys/class/gpio/gpio31/value

# Pin Mux 1 GPIO
if [ ! -d /sys/class/gpio/gpio46 ] ; then
    echo -n "46" > /sys/class/gpio/export
fi
echo -n "out" > /sys/class/gpio/gpio46/direction
echo -n "1" > /sys/class/gpio/gpio46/value

chgrp spi /dev/spidev1.0
chmod g+rws /dev/spidev1.0
;;
stop)
#####

# IO3 = pwm1
echo -n "0" > /sys/class/pwm/pwmchip0/pwm1/enable
echo -n "76" > /sys/class/gpio/unexport
echo -n "64" > /sys/class/gpio/unexport
echo -n "17" > /sys/class/gpio/unexport
echo -n "1" > /sys/class/gpio/gpio16/value
echo -n "16" > /sys/class/gpio/unexport
echo -n "1" > /sys/class/pwm/pwmchip0/unexport

#####

# IO4
echo -n "6" > /sys/class/gpio/unexport
echo -n "in" > /sys/class/gpio/gpio36/direction
```

```
echo -n "36" > /sys/class/gpio/unexport

#####
# I05
echo -n "0" > /sys/class/gpio/unexport
echo -n "in" > /sys/class/gpio/gpio18/direction
echo -n "18" > /sys/class/gpio/unexport
echo -n "66" > /sys/class/gpio/unexport

#####
# I06
echo -n "1" > /sys/class/gpio/unexport
echo -n "in" > /sys/class/gpio/gpio20/direction
echo -n "20" > /sys/class/gpio/unexport
echo -n "68" > /sys/class/gpio/unexport

#####
# I07
echo -n "38" > /sys/class/gpio/unexport

#####
# I08
echo -n "40" > /sys/class/gpio/unexport

#####
# I09
echo -n "4" > /sys/class/gpio/unexport
echo -n "in" > /sys/class/gpio/gpio22/direction
echo -n "22" > /sys/class/gpio/unexport
echo -n "70" > /sys/class/gpio/unexport

#####
# I010
echo -n "10" > /sys/class/gpio/unexport
echo -n "in" > /sys/class/gpio/gpio26/direction
echo -n "26" > /sys/class/gpio/unexport
echo -n "74" > /sys/class/gpio/unexport

#####
# I011
echo -n "1" > /sys/class/gpio/gpio24/value
echo -n "24" > /sys/class/gpio/unexport
```

```

echo -n "25" > /sys/class/gpio/unexport
echo -n "0" > /sys/class/gpio/gpio44/value
echo -n "44" > /sys/class/gpio/unexport
echo -n "72" > /sys/class/gpio/unexport

#####
# IO12
echo -n "42" > /sys/class/gpio/unexport
echo -n "43" > /sys/class/gpio/unexport

#####
# IO13
echo -n "1" > /sys/class/gpio/gpio30/value
echo -n "30" > /sys/class/gpio/unexport
echo -n "in" > /sys/class/gpio/gpio31/direction
echo -n "31" > /sys/class/gpio/unexport
echo -n "0" > /sys/class/gpio/gpio46/value
echo -n "46" > /sys/class/gpio/unexport
;;
status)

echo -n Period:
cat /sys/class/pwm/pwmchip0/device/pwm_period
echo -n Duty Cicle:
cat /sys/class/pwm/pwmchip0/pwm1/duty_cycle
echo -n Enable:
cat /sys/class/pwm/pwmchip0/pwm1/enable
;;
*)
echo -n "Usage: $0 "
echo "{start|stop|restart|force-reload|status}"
exit 1
;;
esac

exit 0

```

2.6 Programa de teste

Para teste, foi implementado um algoritmo que procura chegar na posição, em graus, relativa ao sensor de fim de curso da esquerda. Para isso, inicialmente o motor é

acionado em baixa velocidade até que chegue ao fim de curso, e então os contadores de *encoder* de quadratura são *resetados*. Após, a posição determinada é atingida através do algoritmo de PID. É definida uma margem aceitável de erro para essa posição para que o algoritmo consiga convergir. Também são lidos os sensores de fim de curso para evitar que o motor passe deles por algum tipo de anomalia.

```
#include <stdio.h>
#include <stdlib.h>

#include <time.h>
#include <pid.h>
#include <Q2DSFJ.h>

#define THRESHOLD 0.00001

int main(int argc, char* argv[]){
    Q2DSFJ_T *lib = (Q2DSFJ_T*)malloc(sizeof(Q2DSFJ_T));

    PID_T *pid = (PID_T*)malloc(sizeof(PID_T));

    clock_t last, now;
    float angle, sample, new_voltage, delta, rad_angle;

    if(argc != 2)
        printf("Error\n\tUsage: %s <angle[0;180]>", argv[0]);

    /* Init library*/
    Q2DSFJ_initialize(lib);
    /* Init pid*/
    pid_init(pid, 0.1, 0.2, 0.0005);
    pid_set_boundary(pid, MAX_BOUNDARY, 27);
    pid_set_boundary(pid, MIN_BOUNDARY, -27);

    angle = atof(argv[1]);

    if(angle < 0 || angle > 180)
        printf("Error\n\t Angle MUST BE [0;180]");

    rad_angle = (angle*MATH_PI)/180;
    /* Enable motor*/
    Q2DSFJ_enable_motor(lib, 1);
    /* Return motor to left*/
```

```
Q2DSFJ_motor_home(lib);

pid_setpoint(pid, rad_angle);
last = clock();

while(1){
    /* Some limit switch arrived */
    if (!Q2DSFJ_read_lim_switch(lib, Q2DSFJ_LIMSW_1) ||
        !Q2DSFJ_read_lim_switch(lib, Q2DSFJ_LIMSW_2))
        break;

    /* Arith. avg of two encoders */
    sample = Q2DSFJ_read_enc_counter(lib, Q2DSFJ_ENC_1);
    sample += Q2DSFJ_read_enc_counter(lib, Q2DSFJ_ENC_2);
    sample /= 2;

    delta = sample - rad_angle;
    delta = delta < 0 ? -1*delta : delta;
    /* Arrived on position*/
    if(delta <= THRESHOLD)
        break;

    now = clock();
    /* Calculate the pid */
    new_voltage = pid_compute(pid, (now-last)/CLOCKS_PER_SEC, sample);
    Q2DSFJ_set_motor_voltage(lib, new_voltage);
    last = now;
}

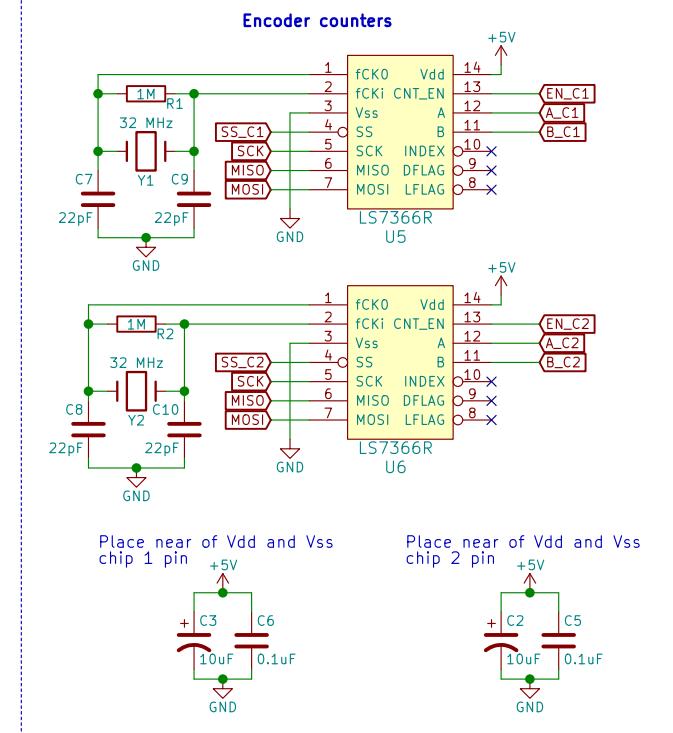
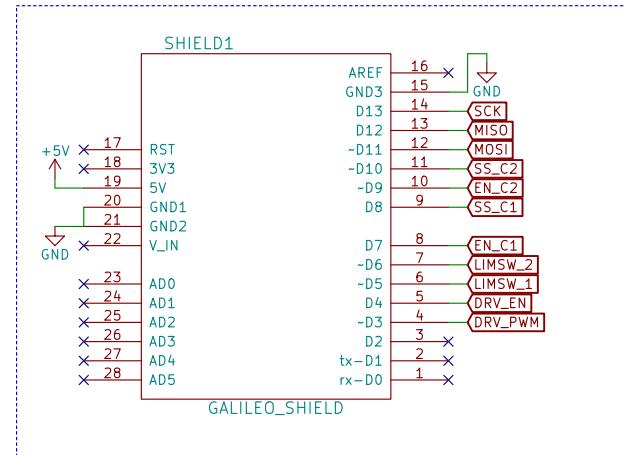
Q2DSFJ_set_motor_voltage(lib, 0);
Q2DSFJ_enable_motor(lib, 0);

free(lib);
free(pid);
return 0;
}
```

3 Anexos

Abaixo estão anexados os esquemáticos do *shield* da Intel Galileo com os contadores de *encoders* e também do *driver* do motor DC.

A



Sheet: /
File: encoder_counter.sch

Title: Shield Galileo with encoder counters connect by SPI bus

Size: A4 Date: 10/07/17
KiCad E.D.A. kicad 4.0.2+dfsg1-stable

Rev: 0.1
Id: 1/1

