

# Trabalho 2: Sistema de software do Uóli

## Avisos

Nesta seção serão apresentados os avisos importantes referentes ao trabalho 2.

1. Os Makefiles do trabalho foram modificados para aumentar a área disponível para a seção de dados do sistema operacional. Agora, em vez do código do usuário começar no endereço 0x77802000, ele começará no endereço 0x77812000. Lembrem-se de ajustar o código do sistema operacional para que o fluxo de execução seja transferido para o endereço novo ao invocar o código do usuário.
2. Prazos de entrega
  - 26-11-2017, até às 23:59 hs - Fator multiplicativo 1.0
  - 27-11-2017, até às 8:00 hs (8 AM) - Fator multiplicativo 0.8
  - 27-11-2017, até às 14:00 hs - Fator multiplicativo 0.6
3. Utilizem o grupo da disciplina para dúvidas e esclarecimentos! (<https://groups.google.com/forum/#!forum/unicamp-mc404-2s2017>) (<https://groups.google.com/forum/#!forum/unicamp-mc404-2s2017>)
4. Parte do trabalho pode ser feito em dupla.: leia com \*muita atenção\* a seção de Entrega e Avaliação, no final do texto.
5. Makefiles disponibilizados:
  - Makefile-ronda (files/Makefile-ronda).
  - Makefile-segue-parede (files/Makefile-segue-parede).

Dica: utilize a flag `-f` para selecionar o Makefile apropriado. Ex:

```
$> make -f Makefile-ronda
```

## Introdução

Neste segundo trabalho da disciplina, você vai desenvolver todas as camadas de *software* responsáveis pelo controle do robô Uóli. Essas camadas, ilustradas na Figura 1, são divididas em três subcamadas: (a) Sistema Operacional UóLi (SOUL), (b) Biblioteca de Controle (BiCo) e (c) Lógica de Controle (LoCo).

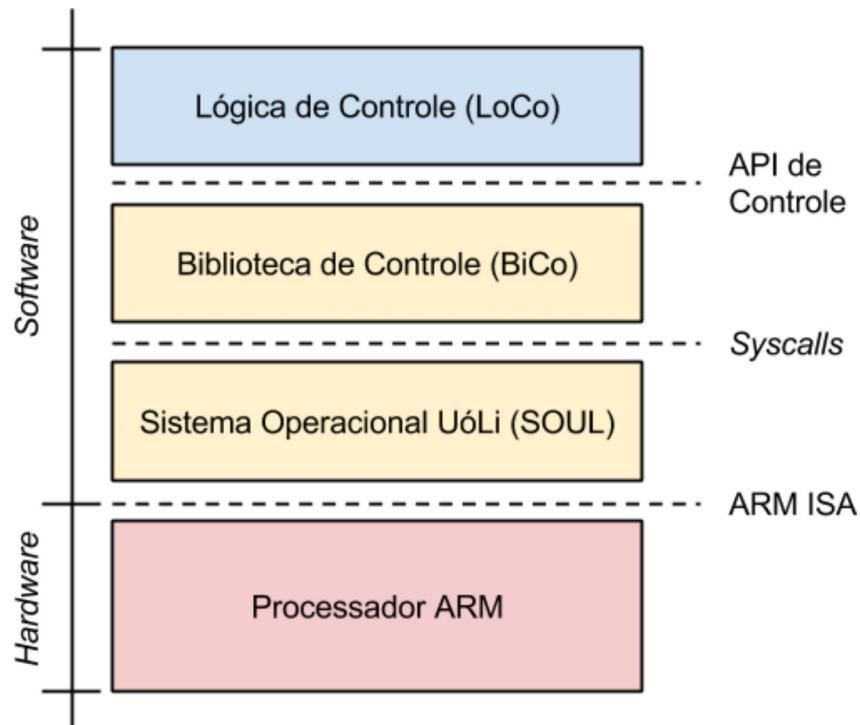


Figura 1: Pilha de software para controle do robô Uóli.

A subcamada SOUL é responsável pelo gerenciamento do *hardware*, incluindo a configuração do *hardware* e o tratamento de interrupções de *hardware* e de *software*. Além disso, o SOUL deve prover um conjunto de serviços para a subcamada BiCo através de chamadas de sistemas, ou *syscalls*. A subcamada SOUL contém código que será executado em modo supervisor e deve ser implementada em linguagem de montagem.

A subcamada BiCo é responsável por prover uma interface de programação amigável para a Lógica de Controle, a API de Controle. A subcamada BiCo também deve ser implementada em linguagem de montagem, mas seu código será executado no modo usuário e ligado com

o código da subcamada LoCo com o auxílio do ligador (*linker*).

A subcamada LoCo é responsável pela lógica de controle do robô e deve invocar as funções definidas pela API de Controle e implementadas pela BiCo. A subcamada LoCo deve ser implementada em código na linguagem C e seu código será executado no modo usuário. Como informado acima, o código da LoCo deverá ser ligado ao código da BiCo com o ligador.

## Subcamada LoCo

Como descrito acima, o código da subcamada LoCo deve ser implementado em linguagem C e deve fazer uso das rotinas disponíveis na API de Controle para enviar comandos para o robô. A API está descrita no arquivo "api\_robot2.h (files/api\_robot2.h)".

Você desenvolverá dois programas na linguagem C para a camada LoCo: o `segue-parede.c` e o `ronda.c`.

### Lógica de controle do programa `segue-parede.c`

A lógica de controle do programa `segue-parede.c` deve ter dois modos de operação: (a) `busca-parede` e (b) `segue-parede`.

- a) A lógica `busca-parede` é iniciada assim que o Uóli é ligado. Esta lógica deve fazer o Uóli andar em linha reta até se aproximar de uma parede (o Uóli não deve colidir com a parede). Após encontrar a parede, o Uóli deve ajustar sua posição de forma que a parede fique do lado esquerdo dele.
- b) Uma vez que a posição foi ajustada, o modo `segue-parede` deve ser ativado. Neste modo, o Uóli deve andar para frente acompanhando a parede, ou seja, sempre mantendo a parede à sua esquerda, ajustando o traçado à medida que a parede se distanciar ou ficar muito próxima do robô. Novamente, é importante que o Uóli não colida com as paredes do ambiente.

### Lógica de controle do programa `ronda.c`

A lógica de controle de seu robô deve fazer o robô realizar uma ronda no ambiente. Para realizar a ronda, seu programa deve:

- Fazer o robô andar em uma "espiral quadrada". Para isso, o robô deve andar um pouco para a frente, fazer uma curva de aproximadamente 90 graus para a direita, andar mais um pouco para a frente, fazer outra curva para a direita, e assim por diante. É importante que, após cada curva, a distância percorrida para frente seja um pouco maior. A distância deve ser ajustada em unidades de tempo do sistema (veja abaixo). Para isso, você deve utilizar as rotinas de `alarm` para temporizar os movimentos. Seu robô deve ser configurado para andar para frente por uma unidade de tempo até a primeira curva à direita, depois andar por 2 unidades de tempo para frente antes de realizar a próxima curva, e assim por diante, até atingir 50 unidades de tempo. A partir daí o robô deve iniciar uma nova ronda.
- Sua lógica deve verificar os sensores para garantir que o robô não colida com as paredes. Caso haja uma parede no traçado do robô, você deve ajustar o curso do robô girando-o para a direita, certificando-se de que ele não colida com a parede.

## Subcamada BiCo

O código da subcamada BiCo deve implementar as rotinas da API de Controle em linguagem de montagem do ARM. A API está descrita no arquivo "api\_robot2.h (files/api\_robot2.h)". Para controlar o *hardware*, o código deve realizar chamadas ao sistema, ou *syscalls*. As *syscalls* são definidas abaixo.

## Subcamada SOUL

A subcamada SOUL deve gerenciar o *hardware* do sistema e prover serviços para a subcamada BiCo através das chamadas de sistemas.

### Tempo do sistema (*system time*)

O SOUL possui um relógio interno que mantém o tempo do sistema, ou *system time*. O tempo do sistema deve ser iniciado com 0 sempre que o sistema for (re)iniciado e o tempo deve ser incrementado de 1 unidade a cada `TIME_SZ` ciclos do relógio (*clock*) de periféricos (similar ao que foi, ou será, desenvolvido na atividade de laboratório 10 ([..//lab10/lab10.html](#))). O símbolo `TIME_SZ` deve ser definido como uma constante utilizando-se a diretiva `.set` no arquivo que gerencia as interrupções do GPT. Utilize um valor razoável para que o tempo do sistema não passe muito rápido nem muito devagar.

### Atendendo chamadas de sistemas (*syscalls*)

Na atividade de laboratório 10 ([..//lab10/lab10.html](#)) você implementou um pequeno programa em linguagem de montagem do ARM para atender às interrupções de *hardware* do tipo IRQ. Nesse trabalho, você deve modificar e expandir a implementação anterior para atender também às **interrupções de software (*syscalls*)**, disparadas pela instrução `SVC`.

Na convenção do ARMv7 (ABI - *Application Binary Interface*), para realizar uma chamada ao sistema, coloca-se o número da *syscall* no registrador `R7`, e os parâmetros seguem a mesma convenção de uma chamada de função comum (devem estar nos registradores `R0` a `R3`); o valor de retorno é passado via registrador `R0`. Esta será a convenção adotada neste trabalho.

Para realizar a chamada de sistema, o código de usuário utiliza a instrução `svc 0x0`. Esta instrução irá gerar uma exceção e fará o registrador `PC` apontar para a posição `base_vet + 0x08`, em que `base_vet` é a base do vetor de interrupções que é definido na seção `.iv`. Nesse ponto,

então, o processador troca o modo para **SUPERVISOR** - a implementação de uma chamada de sistema, portanto, é similar à implementação de interrupções de *hardware*. No entanto, você recebe em R7 um valor que corresponde ao número da *syscall* que se deseja chamar. O seu tratador de chamadas de sistema deve, portanto, analisar o valor contido nesse registrador e selecionar a rotina de tratamento adequada (`set_speed_motor`, `set_speed_motors` ou outra). Lembre-se de que, para retornar do tratador de chamadas de sistema para o código do usuário que invocou a *syscall*, você deve utilizar a seguinte instrução especial:

```
movs pc, lr
```

Essa instrução, além de retornar ao código que estava sendo executado antes da interrupção, recupera o registrador CPSR original, modificando o modo do processador para o modo corrente antes da ocorrência da interrupção.

## Descrição das *Syccalls*

A tabela abaixo apresenta os dados das *syscalls* do sistema. Caso ocorra mais de um erro na execução da chamada de sistema, o código retornado deve ser o do erro de maior valor.

<b>Syccall</b>	<b>Parâmetros</b>	<b>Retorno</b>
read_sonar Código: 16	R0: Identificador do sonar (valores válidos: 0 a 15).	R0: Valor obtido na leitura dos sonares; -1 caso o identificador do sonar seja inválido.
register_proximity_callback Código: 17	R0: Identificador do sonar (valores válidos: 0 a 15). R1: Limiar de distância (veja descrição em api_robot2.h). R2: ponteiro para função a ser chamada na ocorrência do alarme.	R0: -1 caso o número de callbacks máximo ativo no sistema seja maior do que MAX_CALLBACKS. -2 caso o identificador do sonar seja inválido. Caso contrário retorna 0.
set_motor_speed Código: 18	R0: Identificador do motor (valores válidos 0 ou 1). R1: Velocidade.	R0: -1 caso o identificador do motor seja inválido, -2 caso a velocidade seja inválida, 0 caso Ok.
set_motors_speed Código: 19	R0: Velocidade para o motor 0. R1: Velocidade para o motor 1.	R0: -1 caso a velocidade para o motor 0 seja inválida, -2 caso a velocidade para o motor 1 seja inválida, 0 caso Ok.
get_time Código: 20	-	R0: retorna o tempo do sistema.
set_time Código: 21	R0: tempo do sistema	-
set_alarm Código: 22	R0: ponteiro para função a ser chamada na ocorrência do alarme. R1: tempo do sistema.	R0: -1 caso o número de alarmes máximo ativo no sistema seja maior do que MAX_ALARMS. -2 caso o tempo seja menor do que o tempo atual do sistema. Caso contrário retorna 0.

## Iniciando o Sistema

Ao iniciar o sistema, o SOUL deve realizar duas atividades: (a) configurar o *hardware* e (b) transferir a execução para a aplicação de controle no modo usuário.

### Configurando o *Hardware*

Na atividade de laboratório Laboratório 10 (..lab10/lab10.html) você implementou(tará) trechos de código para configurar o GPT e o TZIC. Nesse trabalho, você deverá expandir seu código para configurar o GPIO. Essas alterações vão permitir que o SOUL execute os serviços oferecidos através das *Syccalls*.

O dispositivo de propósito geral de entradas e saídas do sistema, ou GPIO, tem como função prover uma interface para conexão de componentes externos, como periféricos de um robô, ao processador.

Ao todo, o GPIO do simulador fornece 32 pinos que foram conectados a dispositivos periféricos do robô. As conexões foram realizadas de acordo com a tabela abaixo:

Pino	Conexão	Configuração
0	FLAG	Entrada
1	TRIGGER	Saída
2	SONAR_MUX[0]	Saída
3	SONAR_MUX[1]	Saída
4	SONAR_MUX[2]	Saída
5	SONAR_MUX[3]	Saída
6	SONAR_DATA[0]	Entrada

7	SONAR_DATA[1]	Entrada
8	SONAR_DATA[2]	Entrada
9	SONAR_DATA[3]	Entrada
10	SONAR_DATA[4]	Entrada
11	SONAR_DATA[5]	Entrada
12	SONAR_DATA[6]	Entrada
13	SONAR_DATA[7]	Entrada
14	SONAR_DATA[8]	Entrada
15	SONAR_DATA[9]	Entrada
16	SONAR_DATA[10]	Entrada
17	SONAR_DATA[11]	Entrada
18	MOTOR0_WRITE	Saída
19	MOTOR0_SPEED[0]	Saída
20	MOTOR0_SPEED[1]	Saída
21	MOTOR0_SPEED[2]	Saída
22	MOTOR0_SPEED[3]	Saída
23	MOTOR0_SPEED[4]	Saída
24	MOTOR0_SPEED[5]	Saída
25	MOTOR1_WRITE	Saída
26	MOTOR1_SPEED[0]	Saída
27	MOTOR1_SPEED[1]	Saída
28	MOTOR1_SPEED[2]	Saída
29	MOTOR1_SPEED[3]	Saída
30	MOTOR1_SPEED[4]	Saída
31	MOTOR1_SPEED[5]	Saída

As portas do GPIO estão mapeadas no espaço de endereçamento físico do sistema. O dispositivo está conectado ao barramento do sistema no endereço base 0x53F84000. Ele possui três registradores de 32 bits. Cada registrador possui um endereço que é igual ao endereço indicado na tabela abaixo.

Registrador	Deslocamento
DR	0x53F84000
GDIR	0x53F84004
PSR	0x53F84008

A seguir são descritos cada um dos registradores:

#### 1. DR: Registrador de dados (*Data Register*)

Este registrador armazena os dados que serão direcionados aos pinos de saída ou os dados que vieram dos pinos de entrada para o GPIO, dependendo de como o pino esteja programado no registrador GDIR (consulte a seguir).

O resultado de uma leitura deste registrador depende de como o pino (*bit*) está configurado.

- Se o n-ésimo *bit* do GDIR (GDIR[n]) possuir o valor lógico igual a '1', então o retorno da leitura do *bit* DR[n] será o conteúdo a ser escrito no pino de saída correspondente.
- Se o *bit* GDIR[n] possuir valor lógico igual a '0', então o retorno da leitura do *bit* DR[n] será o valor do pino de entrada correspondente.

#### 2. GDIR: Registrador de direções (*Direction Register*)

Este registrador controla as direções de cada um dos pinos de conexão do GPIO.

- Se o GDIR[n] possuir valor lógico igual a '0', o n-ésimo pino está configurada como entrada.
- Se o GDIR[n] possuir valor lógico igual a '1', o n-ésima pino está configurada como saída.

#### 3. PSR: Registrador de plataforma (*pad status register*)

Este é um registrador disponível apenas para leitura. Cada *bit* armazena o valor do sinal do pino de entrada correspondente.

- PSR[n] retorna o valor lógico do sinal do n-ésimo pino.

### Transferindo a execução para a aplicação de controle

Como visto acima, após a configuração do *hardware*, o SOUL deve transferir o fluxo de execução para a aplicação de controle. Lembre-se de que a aplicação de controle deve ser executada no modo usuário, dessa forma, o SOUL deve modificar o modo de operação para **USER**.

Além de ajustar o modo de operação, o SOUL deve configurar uma pilha para o processo da aplicação de controle.

## Apêndice - Hardware

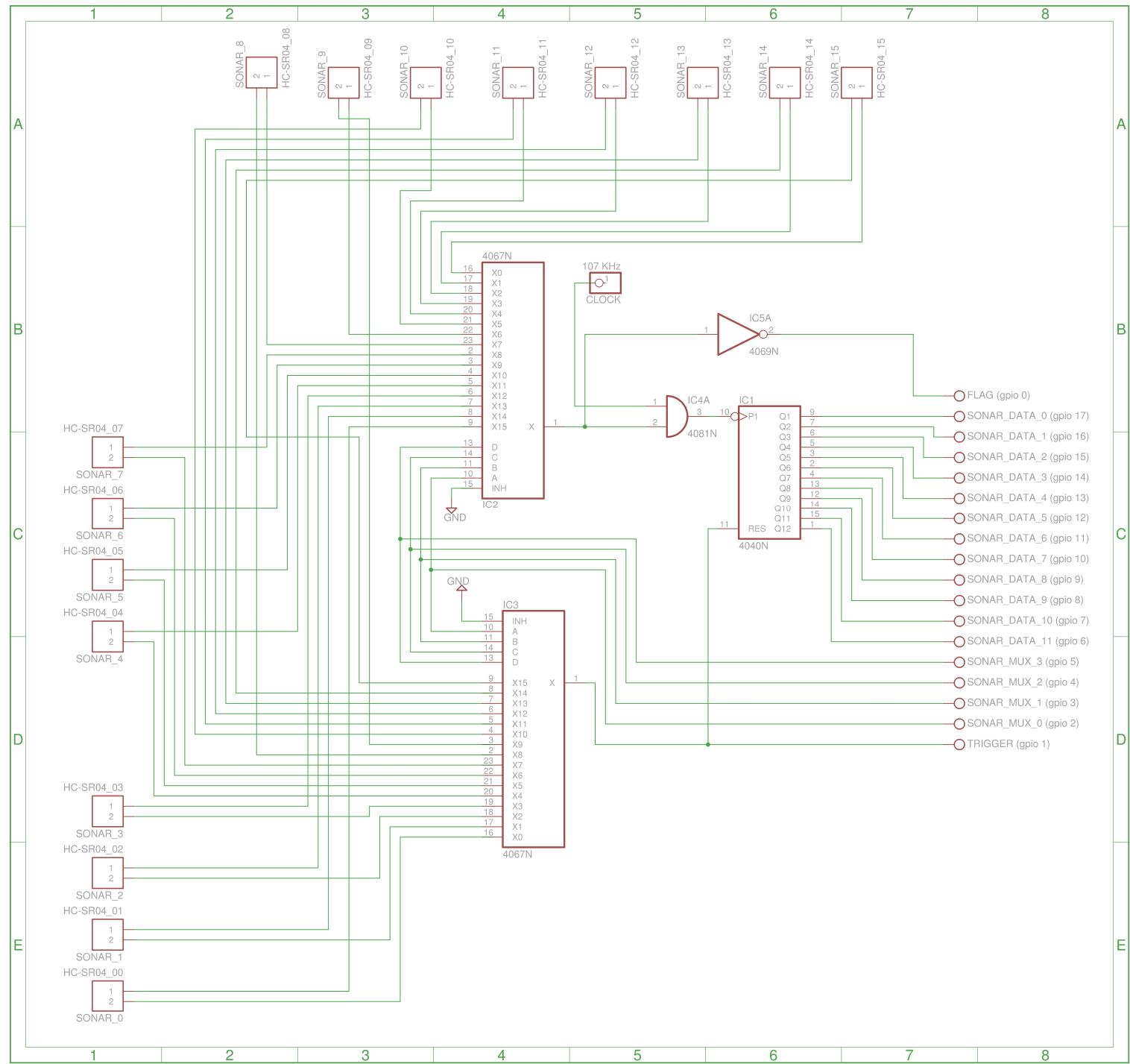
Esta seção apresenta o funcionamento básico do *hardware* criado para o robô, tanto a parte eletrônica, quanto os aspectos necessários para o controle e programação destes periféricos.

## Sonares

Como visto em laboratórios anteriores, o robô dispõe de 16 sonares, sendo 8 frontais e 8 traseiros. Cada um destes sonares trata-se de um dispositivo HC SR04 (files/sonar/HCSR04.pdf), que é controlado via dois sinais, o *TRIGGER* e o *ECHO*. Basicamente, o *TRIGGER* é um sinal lógico que quando levado de um nível baixo para um nível alto, mantendo este por pelo menos 10 ms, e novamente colocando em um nível baixo, faz o sonar obter uma nova leitura.

A leitura retornada pelo sonar é dada pela quantidade de tempo durante a qual o sinal de *ECHO* fica no nível alto. Dessa forma, podemos obter um valor proporcional para a distância, baseado em uma fórmula disponibilizada pelo fabricante do dispositivo.

Para facilitar a comunicação com estes 16 sonares, um novo *hardware* foi proposto. O esquemático deste *hardware* é ilustrado na figura abaixo:



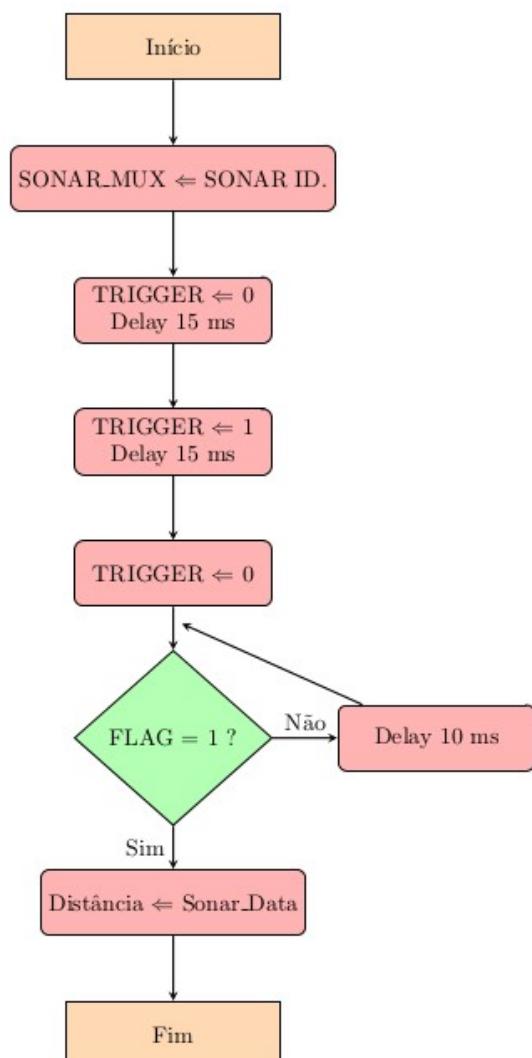
Como o acesso aos sonares é feito individualmente, o circuito de controle conta com um multiplexador e um demultiplexador, para ambos os casos utilizamos o circuito integrado (CI) 74HC4067 (files/sonar/74HC\_HCT4067.pdf). Estes dois CIs são identificados no esquemático como IC3 para o multiplexador e IC2 para o demultiplexador.

Outro CI importante para o funcionamento do circuito é o contador binário MC14040B (files/sonar/MC14040B-D.PDF), que no momento em que o TRIGGER é levado para o nível lógico '1', zera sua contagem. Quando o sinal de ECHO for para o nível lógico '1', ele inicia sua contagem, quando este sinal retorna para '0', a contagem é parada. Dessa forma temos na saída deste contador, pinos SONAR\_DATA[11:0], um valor de 12 bits proporcional à distância encontrada pelo sonar. A frequência de clock para este contador é calibrada para nunca zerar sua saída durante

o momento que o ECHO for '1', mesmo que o sonar selecionado não encontre obstáculo.

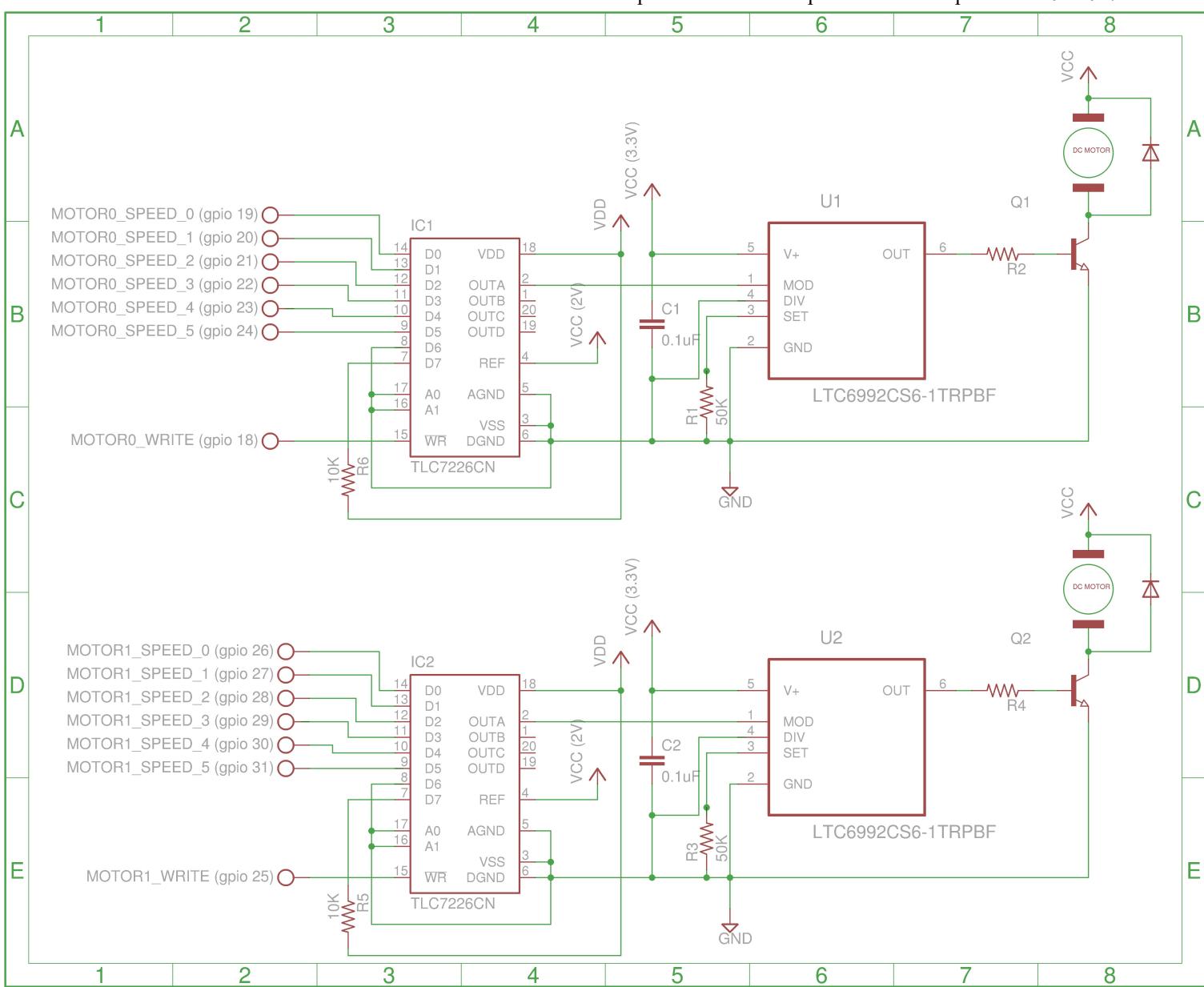
Para auxiliar na obtenção da informação, um sinal adicional é colocado, informando quando o contador terminou sua contagem, trata-se do sinal de FLAG. Sempre que uma leitura for solicitada, após o sinal de TRIGGER retornar para '0', basta esperar o momento que o sinal de FLAG vá para '1', garantindo que a leitura foi completada.

O fluxograma abaixo nos dá uma visão mais clara de como podemos realizar uma leitura via algum dos sonares.



## Motores

O robô possui dois motores, um para cada roda. O controle de velocidade para estes motores é implementado usando PWM ([http://en.wikipedia.org/wiki/Pulse-width\\_modulation](http://en.wikipedia.org/wiki/Pulse-width_modulation)). Um *hardware* específico é proposto para realizar o controle do PWM, no qual para cada motor fornecemos 6 *bits* para quantificar a velocidade e mais um *bit* responsável por habilitar a escrita dessa nova velocidade. O esquemático pode ser encontrado na figura abaixo:



Para gerar o PWM, utilizamos o CI LTC6992 (files/pwm\_files/69921234fc.pdf), que converte um valor analógico, de 0 V até 1 V, para uma largura de pulso proporcional. Para obtermos o valor analógico correspondente, utilizamos o conversor digital-analógico TLC7226 (files/pwm\_files/tlc7226.pdf), no qual os 6 bits vão mapear o PWM e o bit de write, quando colocado em '0', habilita o novo valor.

## Entrega e Avaliação

- Utilize a plataforma SuSy para entrega: <https://susy.ic.unicamp.br:9999/mc404abef/T02> (<https://susy.ic.unicamp.br:9999/mc404abef/T02>).
- Os trabalhos podem ser individuais ou em dupla. Para os trabalhos feitos em dupla:
  - MUITO IMPORTANTE:** apenas a camada do SOUL pode ser realizada em conjunto pela dupla. O desenvolvimento e a submissão das outras camadas (BICO e LOCO) devem ser realizados individualmente!.
  - Ambos os componentes da dupla devem submeter o trabalho no SuSy, cada um enviando sua versão individual do código das camadas BICO e LOCO.
- Deve ser entregue APENAS um arquivo de nome raXXXXXX.tar.gz, que por sua vez deve conter um diretório raXXXXXX que inclua todos os arquivos de código do seu trabalho, um arquivo chamado grupo.txt, e os dois Makefiles: Makefile-ronda e Makefile-segue-parede
- Ambos os Makefiles devem gerar o arquivo disk.img como regra padrão.
- O arquivo **grupo.txt** deve conter o RA dos integrantes da dupla, ou apenas um RA, no caso de trabalho individual. Os valores devem ser da forma raZZZZZZ e devem ser separados por uma quebra de linha; nenhum outro dado deve ser colocado nesse arquivo.
- Seu código deve estar bem documentado, incluindo a descrição das rotinas e trechos de código não triviais.
- O arquivo **worlds\_mc404.zip** (files/worlds\_mc404.zip) contém os cenários nos quais os robôs serão testados.
- Na interface **api\_robot2.h**, a função **read\_sonars** recebe como parâmetro dois índices, indicando o sensor inicial e o sensor final do conjunto, e também o endereço de um vetor para armazenar os valores lidos. Este vetor deve ter tamanho final - inicial + 1, ou seja, os valores começam a ser armazenados na posição zero do vetor.
- Os símbolos **MAX\_ALARMS** e **MAX\_CALLBACKS** devem ser definidos como constantes utilizando-se a diretiva **.set**. Defina ambos com o valor padrão 8.
- Qualquer tentativa de fraude implicará média 0 na disciplina, para todos os envolvidos.**