

# Trabalho 2: Sistema de *software* do Uóli

## Avisos

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

### 1. Prazos de entrega

- 26-11-2015, até as 23:59
  - Fator de multiplicação 1.0x; e
- 27-11-2015, até as 23:59
  - Fator de multiplicação 0.9x; e
- 28-11-2015, até as 23:59
  - Fator de multiplicação 0.8x; e
- 39-11-2015, até as 23:59
  - Fator de multiplicação 0.6x; e

2. Utilizem o grupo da disciplina para dúvidas e esclarecimentos! O endereço é:  
<https://groups.google.com/forum/#forum/unicamp-mc404-2s2015>

3. Parte do trabalho pode ser feito em dupla.: leia com \*muita atenção\* a seção de Entrega e Avaliação, no final do texto.

4. Makefile disponibilizado: [Makefile](#).

## Introdução

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

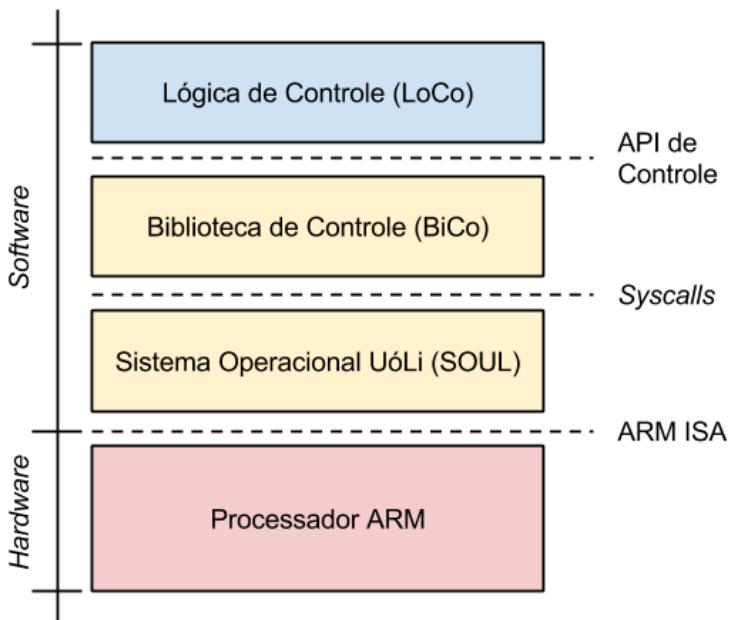


Figura 1: Pilha de *software* para controle do robô Uóli.

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

A sub-camada BiCo é responsável por prover uma interface de programação amigável para a Lógica de Controle, a API de Controle. Esta sub-camada também deve ser implementada em linguagem de montagem, mas seu código será executado no modo usuário e ligado com o código da sub-camada LoCo com o auxílio do ligador (*linker*).

A sub-camada LoCo é responsável pela lógica de controle do robô e deve invocar as funções definidas pela API de Controle e implementadas pela BiCo. Esta sub-camada 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.

## Sub-camada LoCo

Como descrito acima, o código da sub-camada 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](#)".

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 com que o Uóli ande 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 do mesmo.
- b) Uma vez que a posição foi ajustada, o modo segue-parede deve ser ativado. Neste modo, o Uóli deve andar para frente seguindo a traçado da 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 mesmo. 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 com que o robô realize uma ronda no ambiente. Para realizar a ronda, seu programa deve:

- Fazer com que o robô ande 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 que ele ande para frente por uma unidade de tempo até a primeira curva à direita, depois ande 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 que ele não colida com a parede.

## Sub-camada BiCo

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

## Sub-camada SOUL

A Sub-camada SOUL deve gerenciar o *hardware* do sistema e prover serviços para a sub-camada 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 mesmo deve ser incrementado de 1 unidade a cada TIME\_SZ ciclos do relógio (*clock*) de periféricos (similar ao que foi desenvolvido na atividade de [laboratório 8](#)). 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 8](#) você implementou um pequeno programa em linguagem de montagem do ARM para atender às interrupções de *hardware* do tipo IRQ. Nesse trabalho, você deve 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, você deve colocar 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. Para realizar a chamada de sistema, o código de usuário utiliza a instrução svc 0x0. Esta instrução irá gerar uma exceção e fará com que o registrador PC aponte 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 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:

Syccall	Parâmetros	Retorno
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 8](#) você implementou 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, ou GPIO, do sistema tem como função prover uma interface para conexão de componentes externos, como periféricos de um robô, ao processador.

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 base mais um certo deslocamento indicado na tabela abaixo.

Registrador	Deslocamento
DR	0x00
GDIR	0x04
PSR	0x08

A seguir serão descritos cada um dos registradores:

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

Este registrador armazena os dados que serão direcionados às linhas de saída ou os dados que vieram das linhas de entrada para o GPIO, dependendo de como a linha esteja programada no registrador GDIR (consulte a seguir).

O resultado de uma leitura deste registrador depende de como a linha (*bit*) está configurada.

- 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

- próprio conteúdo deste registrador, que será escrito na linha 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 da linha de entrada correspondente.

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

Este registrador controla as direções de cada uma das linhas de conexão do GPIO. Cada bit especifica a direção de um sinal de uma das linhas do GPIO. O mapeamento de cada linha para um dispositivo externo depende de como esse dispositivo foi conectado ao GPIO.

- Se o GDIR[n] possuir valor lógico igual a '0', a n-ésima linha está configurada como entrada.
- Se o GDIR[n] possuir valor lógico igual a '1', a n-ésima linha 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 da linha de entrada correspondente.

- PSR[n] retorna o valor lógico do sinal da n-ésima linha.

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

## 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 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

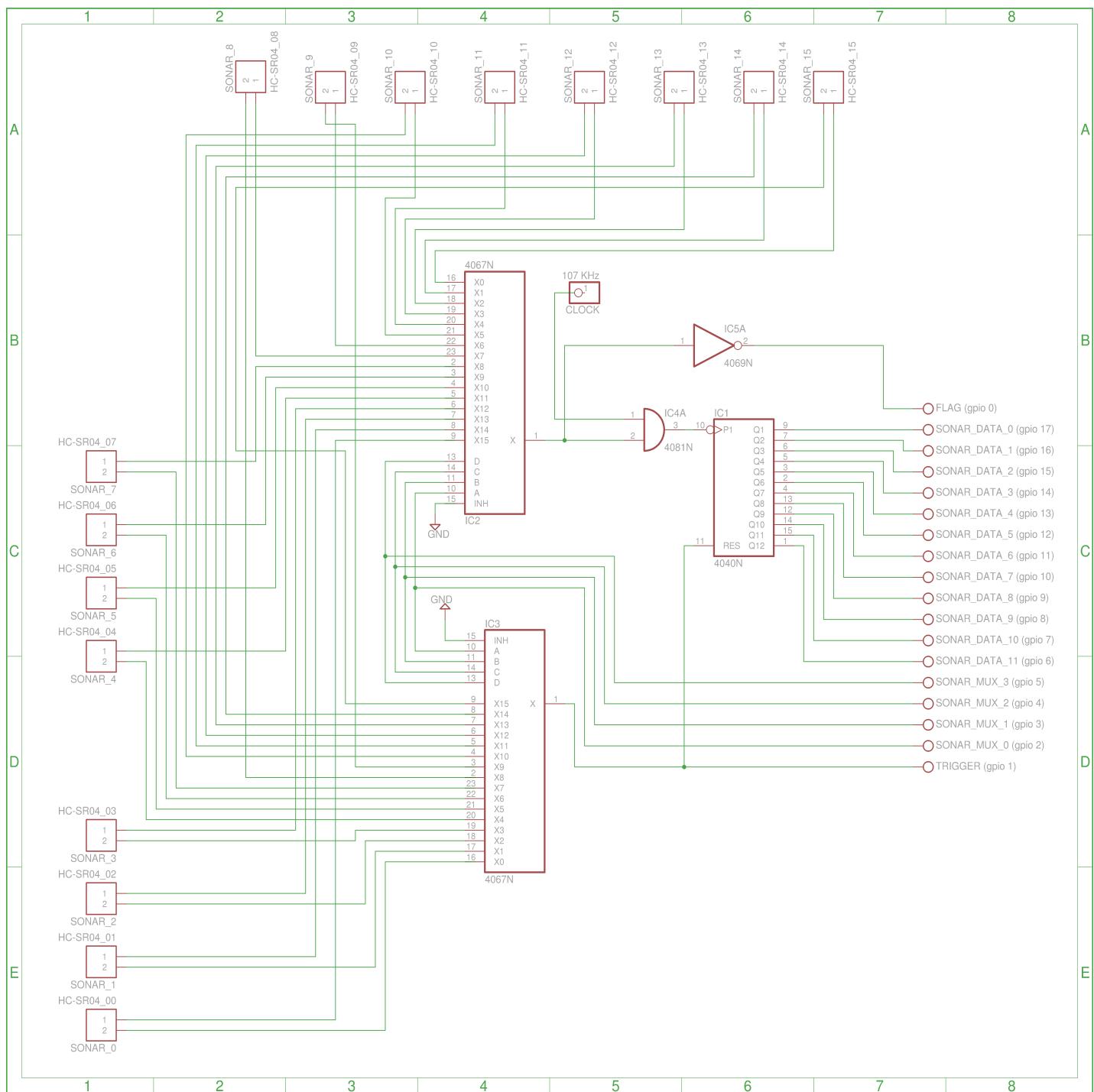
Nesta seção vamos apresentar o funcionamento básico do *hardware* criado para o robô, tanto a parte eletrônica, quanto os aspectos necessários para o controle e programação destes periféricos.

### 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**, 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 em que 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:

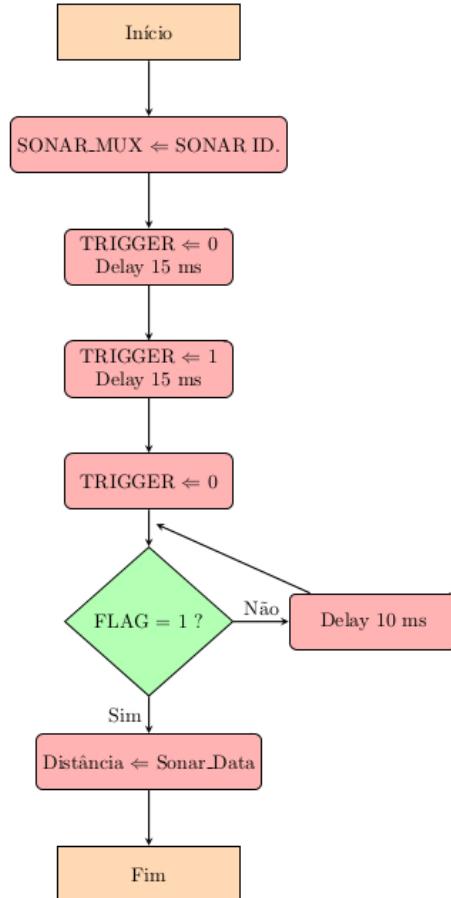


Como o acesso aos sonares é feito individualmente, o circuito de controle conta com um multiplexador e um demultiplexador, para ambos casos utilizamos o circuito integrado (CI) [74HC4067](#). Estes dois CIs são identificados no esquemático como IC3 para o multiplexador e IC2 para o demultiplexador.

Outro CI importante para o funcionamento do circuito, é o contador binário [MC14040B](#), 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 a 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.

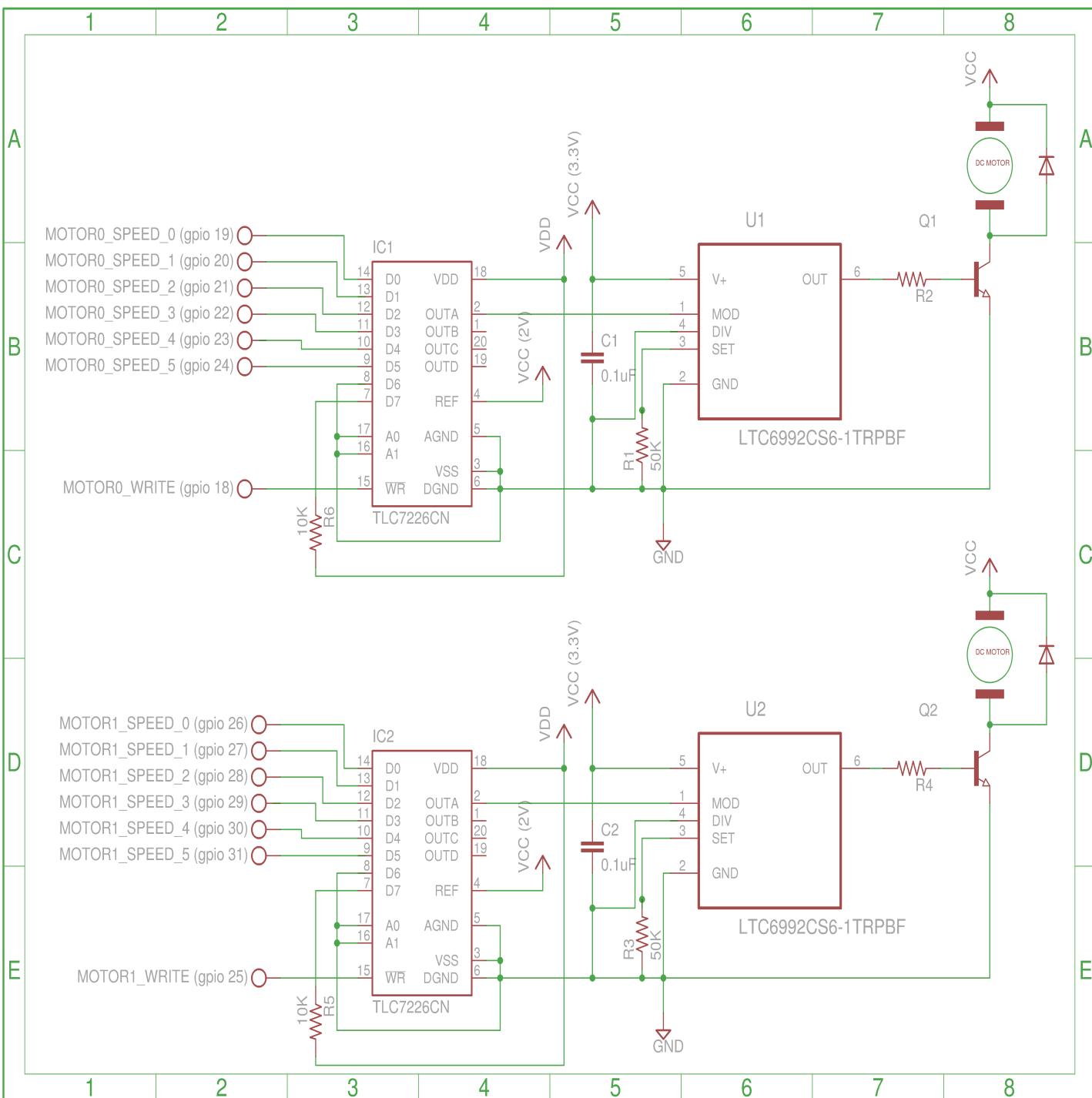
Para auxiliar na obtenção da informação, um sinal adicional é colocado, informando quando o contador terminou sua contagem, trata-se do sinal de FLAG. Sempre que uma leitura for solicitada, após o sinal de TRIGGER retornar para '0', basta esperar o momento que o sinal de FLAG for para '1', garantindo 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](#). Um hardware específico é proposto para realizar o controle do PWM, onde 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 C1 [LTC6992](#), que converte um valor analógico, de 0V até 1V, para uma largura de pulso proporcional. Pra obtermos o valor analógico correspondente, utilizamos o conversor digital-analógico [TLC7226](#), onde 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/mc404ab/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 raXXXXXXX.tar.gz, que por sua vez deve conter um diretório raXXXXXX que inclua todos os arquivos de código do seu trabalho, um Makefile e um arquivo chamado grupo.txt
- O Makefile deve 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 raZZZZZZZ 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](#) contém os cenários nos quais os robôs serão testados.
- 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.