M Acionamento de Motores DC

Versão 1.0 (06/03/2020)

É frequente a necessidade de se acionar motores com o MSP430 e, não raro, encontram-se dificuldades e surpresas desagradáveis tais como dano no MSP ou queima do motor. Esta é a razão deste apêndice: auxiliar o leitor a controlar seu motor, sem sobressaltos (esperamos). Evidentemente, o acionamento de motores é um assunto vasto e não pretendemos estudá-lo de forma detalhada. Faremos apenas uma abordagem pragmática, tentando indicar ao leitor pontos importantes e oferecendo algumas soluções que talvez se adaptem ao seu problema.

M.1. Os Cuidados com Cargas Indutivas

O motor, não importa se AC, DC ou de passo, é uma carga indutiva, assim como transformadores e relés eletromecânicos. De forma simplista, todo equipamento que tiver enrolamentos (bobinas) é uma carga indutiva. Como vamos trabalhar exclusivamente com o acionamento em corrente contínua (DC), não nos preocuparemos com o atraso que a corrente sofre em relação à tensão, que é típico de cargas indutivas.

Porém, teremos uma grande preocupação com a Força Contra Eletromotriz provocada pelo colapso do campo magnético que acontece quando a alimentação de uma carga indutiva é desligada. Conferir se está correto. A equação abaixo é derivada da Lei da Indução Magnética, onde *v(t)* é a tensão nos terminais da bobina, Φ_B é o fluxo magnético, *i* é a corrente, *L* a indutância, *t* é o tempo e *d* é o operador derivada. Essa equação mostra que uma rápida variação no fluxo magnético (Φ_B) pode gerar altas tensões e isso acontece quando se chaveia os enrolamentos dos motores.

$$v(t) = \frac{d\Phi_B}{dt} = L\frac{di}{dt}$$

Para tornar a ideia bem clara, vamos a um experimento simples. A Figura M.1 apresenta uma bateria de 5V, uma chave *S*, um indutor *L* e um resistor *R*. O indutor representa um enrolamento de um motor qualquer e *R* sua resistência interna, que é muito baixa (alguns Ohms ou fração de Ohm). Na porção da esquerda desta figura, quando fechamos a chave *S*, passa uma corrente pelo indutor e cria um campo magnético ao seu redor. Quando a chave é aberta, esse campo magnético entre em colapso e provoca sobre a chave *S* uma

tensão elevada, digamos algumas centenas de Volts, como mostrado no lado direito da figura. O problema é que no acionamento de motores, a chave *S* é, na verdade, um transistor e essa alta tensão pode vir a danificá-lo. Precisamos evitar isso.

Curiosidade: a maioria desses aparelhinhos que, de brincadeira, dão choque nas pessoas, como uma caneta ou um livro ao ser aberto, geram sua alta tensão a partir do colapso do campo de um indutor.

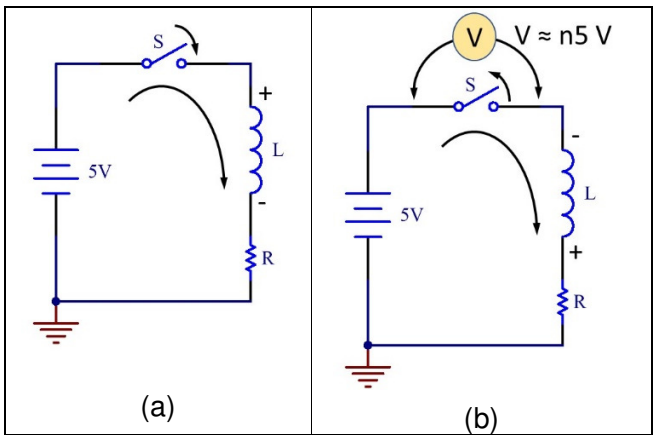

Figura F.1. O problema da Força Contra Eletromotriz gerado pelo colapso do campo magnético de um indutor.

A forma mais simples de se eliminar essa alta tensão da Força Contra Eletromotriz é usar um diodo em paralelo com a carga indutiva, como mostrado na Figura M.2. Note que, enquanto a chave estiver fechada, o diodo D está polarizado reversamente. Porém, quando a chave é aberta, o indutor se descarrega com tensão reversa, o que polariza diretamente o diodo e "fecha um curto" sobre o indutor. Em inglês esse diodo recebe o nome de "*frewheeling diode*" ou "*flyback diode*".

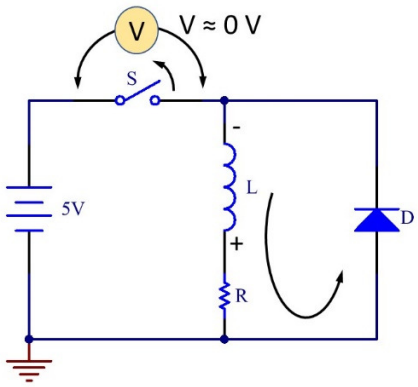

Figura M.2. Emprego de um diodo para "desviar" a Força Contra Eletromotriz gerada pelo colapso do campo magnético de um indutor.

Curiosidade: as antigas TVs com tubo de raios catódicos tinham um diodo de *flyback* que queimavam com certa frequência e os técnicos, de maneira jocosa, ao explicar o conserto para o cliente diziam que o problema era o Frei Beque.

É relativamente simples dimensionar esse diodo de proteção, mas não faremos isso. Para o leito que estiver interessado, existe farta documentação na Internet. Como vamos trabalhar com motores pequenos, sugerimos o diodo 1N4148. Para o caso de motores maiores, deve ser usado um diodo retificador, por exemplo um da família 1N400X (1N4001, 1N4002, ...). É muito comum o 1N4007.

Como o leitor deve saber, motores geram muito ruído eletromagnético. Uma forma de limitar um pouco este ruído é colocar um filtro RC em paralelo com o motor. Esse filtro é também conhecido como "*snubber*". Para nosso estudo, vamos simplesmente colocar um capacitor de 100 ηF em paralelo com o motor.

M.2. Como Usar Saída Digital para Acionar Motor

Vamos agora abordar o problema que nos interessa. Temos um motor e precisamos acioná-lo sob controle de um PWM gerado pelo MSP. Para o caso de *led s*, como fizemos no capítulo sobre os contadores/temporizadores, usamos a solução mostrada na Figura M.3.a. Não podemos fazer isso com motores. Pode ser é óbvio, mas precisamos orientar que os pinos do MSP nunca devem ser usados para acionar diretamente um motor, como mostrado na Figura M.3.b.

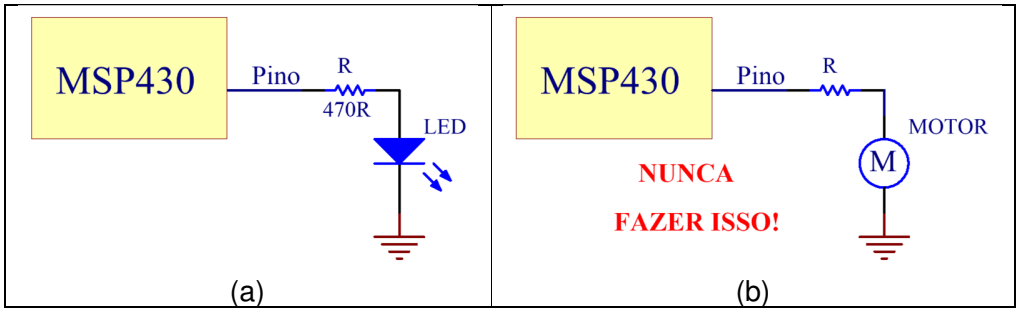

Figura M.3. Aqui é indicado o óbvio: o acionamento de um motor nunca pode ser análogo ao acionamento de um led .

A forma correta de se acionar um motor é através de um dispositivo externo, como mostrado na Figura M.4. Assim, usando o sinal digital do MSP controlamos o *dispositivo x*, e este é quem vai acionar a corrente que passa pelo motor. Este dispositivo deve ser compatível com as características do motor, em especial, com a corrente drenada por ele. Esta solução oferece uma grande vantagem que é a de permitir que a alimentação do motor, marca na figura com VM, seja diferente da alimentação do MSP430 (3,3 V). Este dispositivo x pode ser um transistor bipolar, um transistor MSOFET, um par Darlington,

etc. Como pode ser visto na Figura M4, esse tal dispositivo x pode estar entre a alimentação VM e o motor, ou entre o motor e a terra.

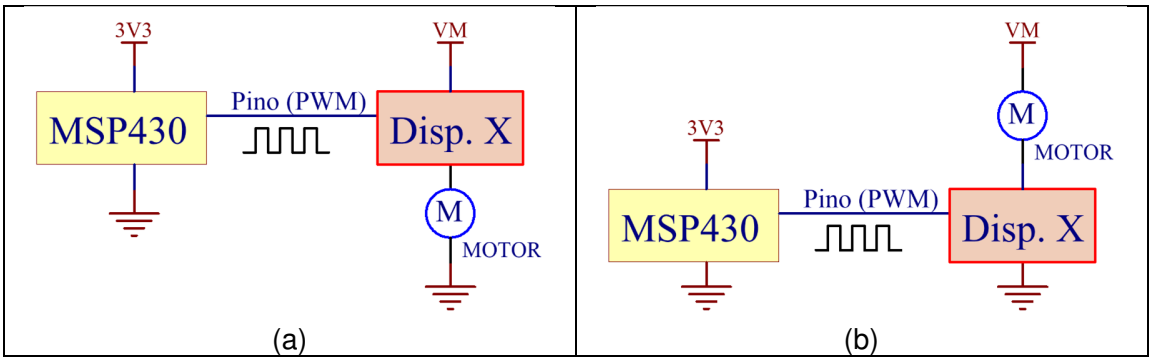

*Figura M.4. MSP acionando um motor através do **Dispositivo x**. Este dispositivo pode estar entre a alimentação (VM) e o motor ou entre o motor e a terra.*

M.2.1. Acionamento com Componentes Discretos

A Figura M.5 apresenta seis sugestões para acionamento de motor usando transistor MOSFET, transistor bipolar e transistores na configuração Darlington. Os transistores indicados são apenas sugestões. Eles não são críticos e podem ser substituídos por outros equivalentes. Entretanto, é muito importante conferir a capacidade de corrente do dispositivo. Como cuidado mínimo, é recomendado que o encapsulamento seja o TO-220, pois transistores colocados neste encapsulamento têm boa capacidade de corrente, ou seja, vão conseguir acionar a grande maioria dos motores que usaremos. É claro que, para os nossos propósitos, a tensão de alimentação do motor, indicada por "VM", não deve ser muito alta. Digamos que o limite seja 24 V. Acima disso, o leitor deve buscar informações em outra referência.

Regra do dedo queimado: com motor "rodando", toque o transistor com o dedo. Enquanto o leitor não se queimar, é porque (mesmo quente) está tudo bem. Se o transistor ficar quente a ponto de o leitor não conseguir mais tocá-lo, então é porque se faz necessário um dissipador de calor ou usar um motor menor.

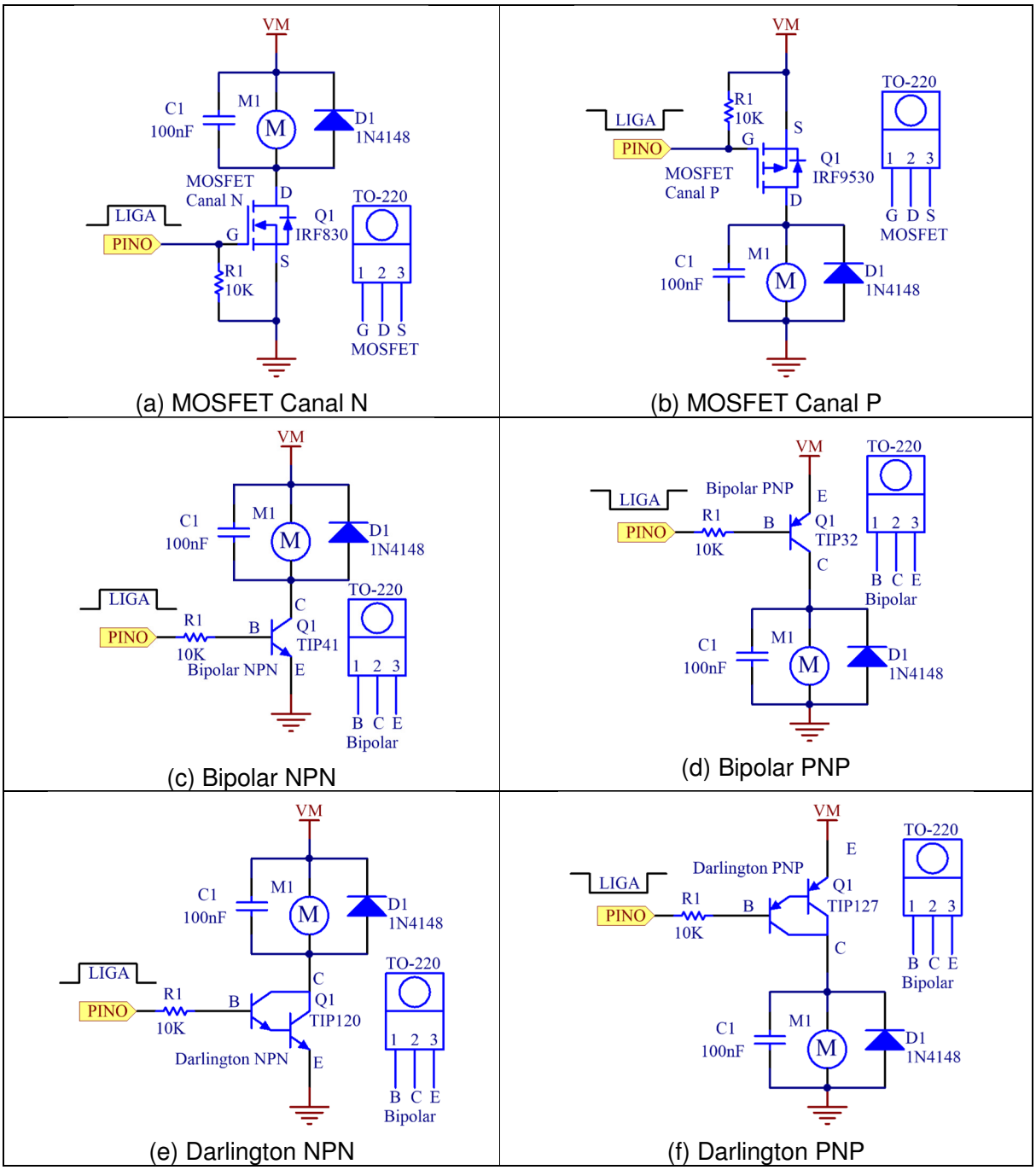

Figura M.5. Seis sugestões para acionamento de motores com o MSP. A alimentação do motor está sinalizada como "VM".

Muitas vezes os esquemas apresentados não funcionam bem. Podem interferir no funcionamento a tensão de alimentação do motor (VM), o tipo de transistor usado e o nível de tensão no pino que aciona o transistor. Por isso, a Figura M.6 apresenta mais algumas alternativas, um pouco mais complexas, para o acionamento. Para o opto TRIAC sugerimos MOC3020, 3021, 3022 ou 3023. Esta figura apenas apresenta sugestões usando MOSFET, entretanto, podem ser aplicadas para o caso de transistores bipolares.

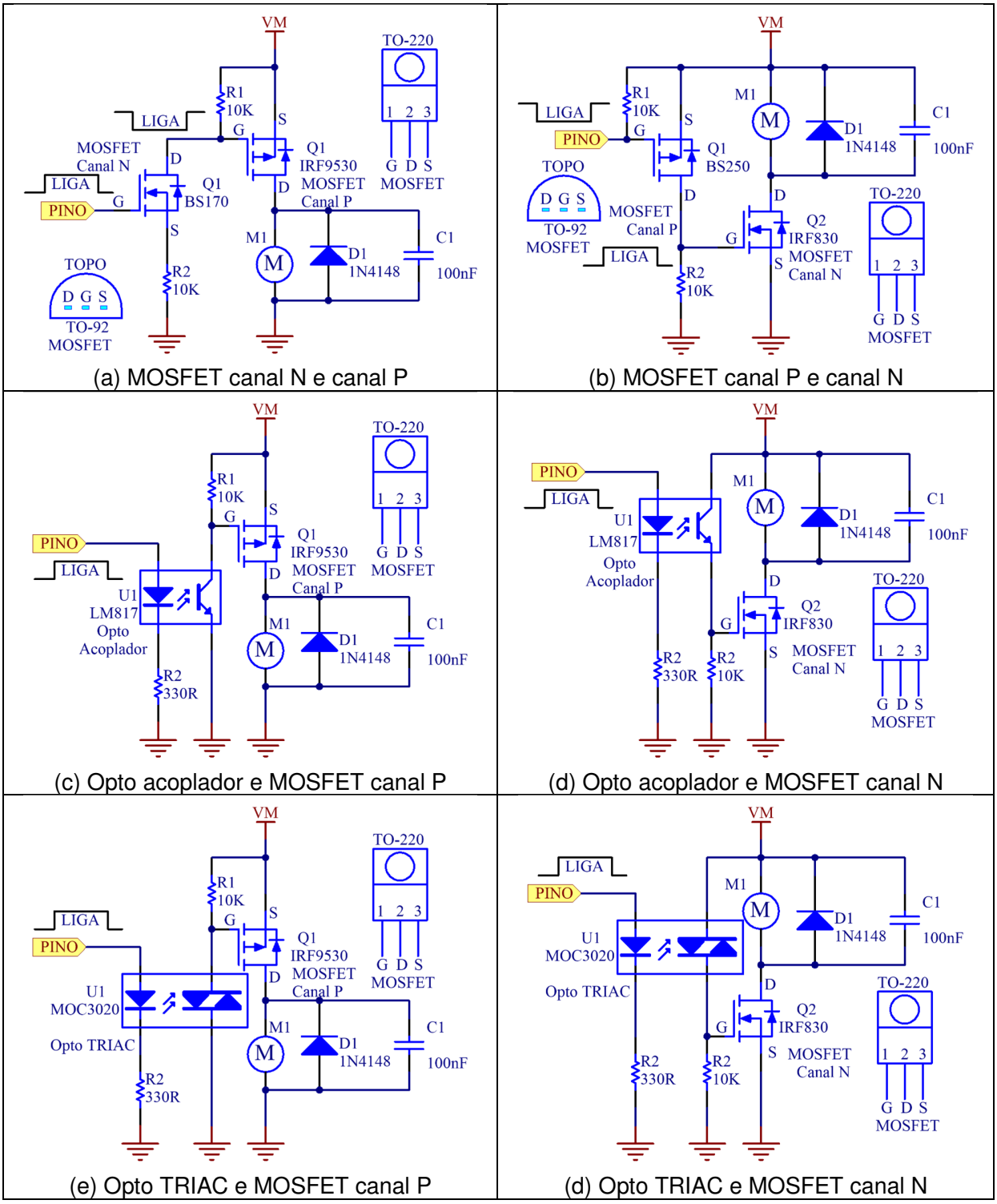

Figura M.6. Sugestões para acionamento de motores com o MSP, agora usando um componente auxiliar (outro transistor, opto acoplador ou opto TRIAC). A alimentação do motor está sinalizada como "VM".

M.2.1. Acionamento com Ponte H

O que vimos até agora, nos permite acionar um motor, porém ele sempre vai girar no mesmo sentido. Para trocar o sentido de rotação do motor, será necessário inverter seus terminais, para assim inverter o sentido da corrente. Vamos agora avançar um pouco e propor uma solução para acionar o motor e ainda selecionar seu sentido de rotação.

A solução é simples e recebe o nome de Ponte H. Ela está apresentada de forma hipotética na Figura M.7, onde usamos chaves para facilitar a explicação. Na próxima figura cada chave será substituída por um transistor. Note que o nome vem da maneira que motor está ligado às chaves, que lembra a letra "H". Se acionarmos as chaves S1 e S4, como mostrado no centro da figura, a corrente passa pelo motor da esquerda para direita. Porém, se fecharmos as chaves S2 e S3, a corrente passa pelo motor no sentido inverso, da direita para a esquerda. Assim, apenas acionando as chaves, podemos girar o motor em cada um dos dois sentidos possíveis.

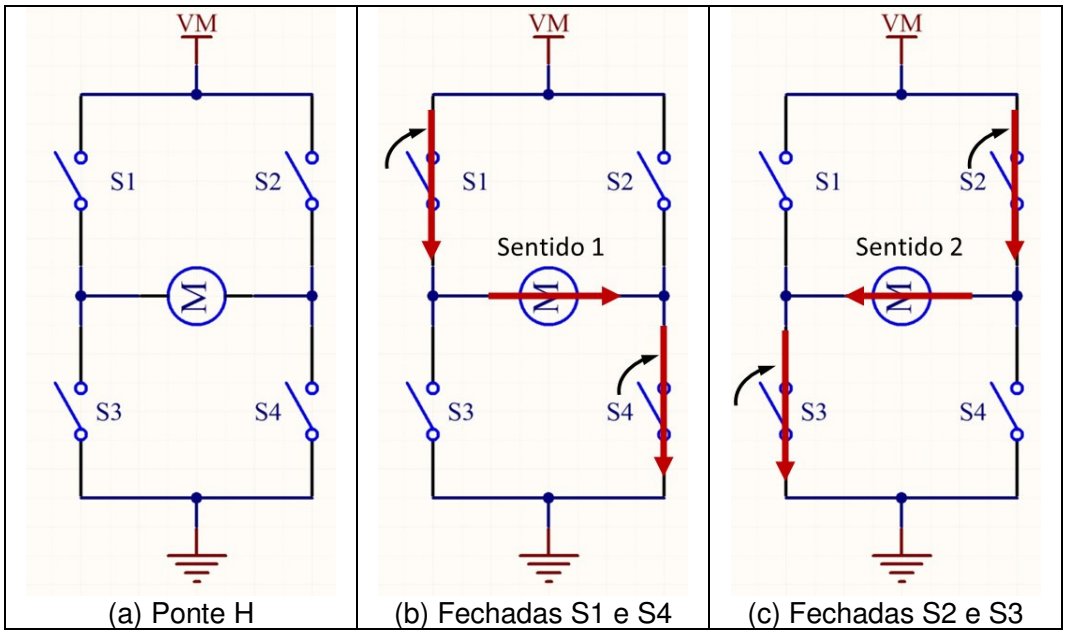

Figura M.7. Ponte H e as duas opções de rotação do motor.

A Figura M.8 apresenta uma sugestão para uma Ponte H usando transistores MOSFET. Note que na Porta (Gate) de cada transistor foi colocada a letra "L" (baixo) ou a letra "H" (alto) para indicar o nível lógico que aciona cada transistor. É fácil de ver que os níveis lógicos dos controles "PINO 1" e "PINO 2" devem estar sempre invertidos. Não foram colocados os diodos de proteção porque os transistores MOSFET usados já o trazem incorporado. Entretanto, em caso de dúvida, adicione esses 4 diodos.

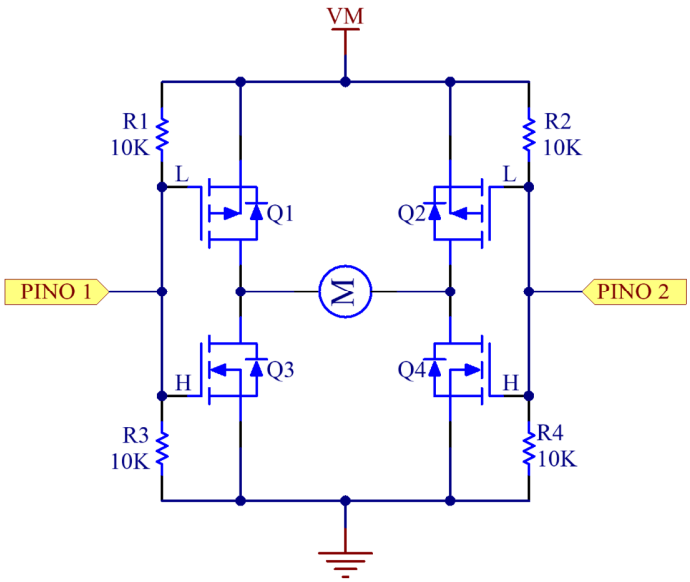

Figura M.8. Sugestão para a construção de uma Ponte H usando transistores MOSFET.

A Ponte H é uma excelente solução para acionamento de motores e um componente comercial muito comum é o CI L298. Esse chip, cujo esquema está apresentado na Figura M.9 oferece duas pontes H em um único encapsulamento. O controle está indicado na Tabela M.1, logo abaixo.

- Ponte A: motor em OUT 1 e OUT2 e os controles são En A, In1 e In2;
- Ponte B: motor em OUT 3 e OUT4 e os controles são En B, In3 e In4;
- VS é a alimentação do motor e
- VSS é a alimentação da lógica digital, em geral 5 V.

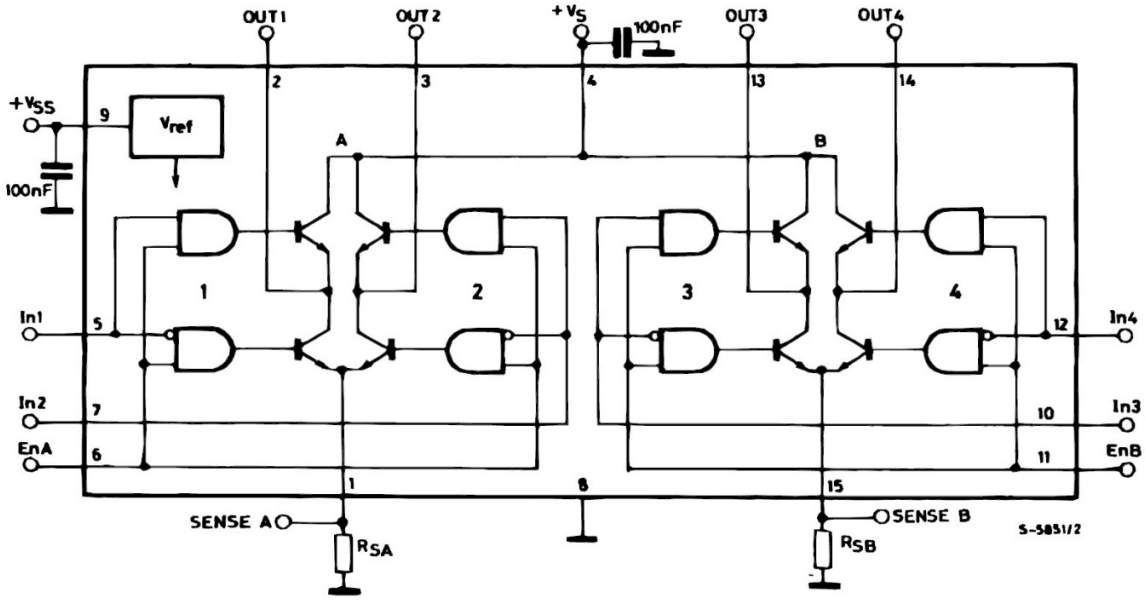

Figura M.9. Detalhe do circuito interno do L298 que oferece duas Pontes H independentes.

A Tabela M.1 resume a operação do L298. Note que se **En** = 0, o motor está livre. Se **En** = 1, as entradas **In** controlam o sentido de rotação, aqui denominados de direto e reverso. Existe uma opção de frenagem do motor, que é conseguida com **En** = 1 e as duas entradas **In** no mesmo nível lógico. Para ser usado com PWM controlando a velocidade de rotação, o sinal PWM deve ser ligado à entrada **En**.

Tabela M.1. Tabela verdade para uso do L298

En A (En B)	In 1 (In 3)	In 2 (In 4)	Função
0	X	X	Livre
1	1	0	Direto
1	0	1	Reverso
1	0	0	Freio
1	1	1	Freio

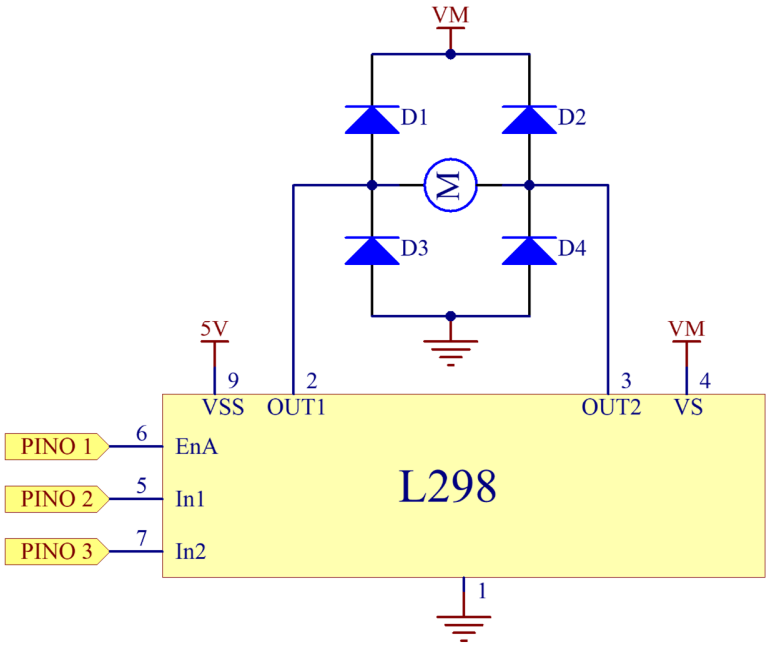

Figura M.10. Sugestão para emprego de uma das pontes H do CI L298. Note os diodos de proteção.

A Figura M.10 apresenta uma sugestão para emprego de uma das pontes H. Note que há a necessidade de se adicionar externamente os 4 diodos de proteção (*free wheeling*). Ainda pode ser adicionado um capacitor de 100 ηF em paralelo com o motor para reduzir

o ruído. A tensão VM (denominada VS pelo manual do L298) é a que alimenta o motor e seu valor máximo é de 45 V. O valor mínimo para VM é igual a V_{IH} + 2,5 V, onde V_{IH} é o valor mínimo de tensão para uma entrada digital do L298 ser interpretada como nível alto. O manual do L298 indica V_{IH} = 2,3 V (mínimo), assim precisamos de VM (VS) > 4,8 V (2,3 V + 2,5 V). A parte lógica do L298 é alimentada por VSS cujo valor típico é 5 V (faixa de 4,5 V até 7,0 V).

Para o MSP430 operando em 3,3 V, não deve haver problemas na sinalização digital para com o L298, como mostrado na Figura M.11. Temos:

- $V_{OL} < V_{IL}$, pois 0,25 V < 1,5 V e
- $V_{OH} > V_{IH}$, pois 3,05 V > 2,3 V.

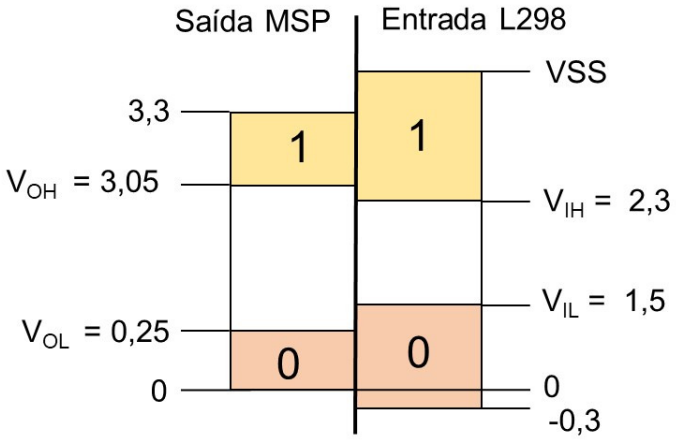

Figura M.11. Comparação da sinalização digital gerada pelo MPS operando em 3,3 V, com as exigências do L298, que é alimentado com 5,0 V. A unidade usada é o Volt.

O mercado oferece uma grande disponibilidade de placas que facilitam o uso do L298, como a apresentada na Figura M.12. Pelo detalhamento feito nesta figura, é fácil de ver que todos os recursos deste *chip* já foram disponibilizados nos pinos da placa e que também inclui os 4 diodos de proteção para cada motor. A alimentação desta placa pode parecer um pouco confusa. Como não existe um manual disponível, as orientações são conseguidas na Internet e são algumas vezes, conflitantes.

Veja na Figura M.12 que existe uma entrada para VS (alimentação do motor), rotulada com a frase “5V até 35 V” e uma outra entrada para VSS rotulada “5V (in/out)” e um jumper marcado com 12V. Após verificar as conexões com um multímetro, foi possível levantar o circuito apresentado na Figura M.13, o que permite entender um pouco melhor essas conexões.

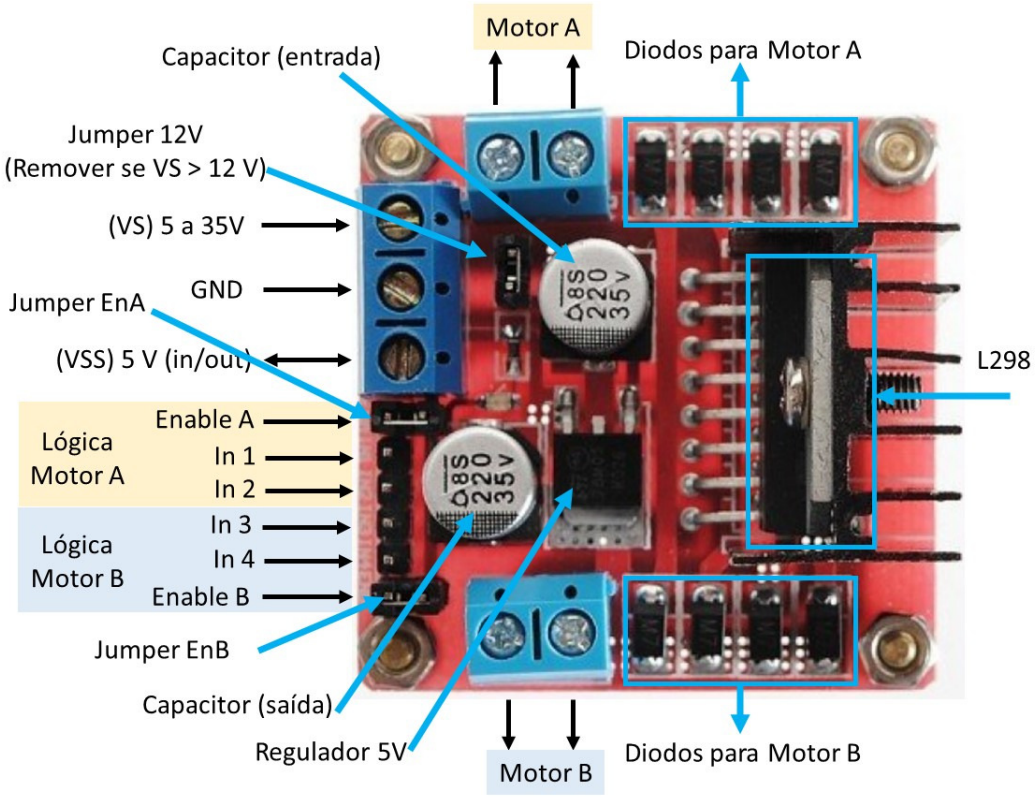

Figura M.12. Duas pontes H do L298 já montado em uma placa para fácil emprego.

Note, no esquema da Figura M.13, que essa placa possui um regulador linear 78M05 que é um regulador de 5V e capacidade de até 500 mA. Sobre esse regulador, o manual do fabricante afirma que ele possui limitação de corrente (300 mA quando em 35 V) para o caso de curto circuito e ainda que tem recurso para desligamento por aquecimento. De forma orgulhosa, esse manual diz que o produto é virtualmente indestrutível. Note ainda que essa placa usa dois capacitores eletrolíticos de 35 V, daí a limitação em 35 V, abaixo dos 45 V que o manual do L298 especifica como limite.

Quando o usuário alimenta VS com uma tensão de 5 V até 12 V e mantém o jumper J1 fechado, essa tensão também passa pelo regulador 78M05 e fornece os 5 V para o pino 4 (VS) do L298. Além disso, no terminal VSS da placa, esses 5 V ficam disponíveis e o usuário pode usá-los para alimentar seu circuito externo. Quando o usuário trabalha com VS > 12 V, ele deve remover o jumper J1 e com isso o regulador 78M05 fica inoperante. Neste caso, além de VS o usuário deve também fornecer os 5 V para o VSS.

Um outro detalhe, que está ilustrado na Figura M.13, é com relação às entradas EnA e EnB. Elas estão em um *jumper*. Se o usuário quiser que a tensão VSS (5,0 V) habilite o EnA ou EnB, basta fechar esse *jumper*. Para controlar externamente esses sinais, é preciso deixar o *jumper* aberto. Em cada *jumper*, o pino a ser usado para o controle externo é o mais próximo da borda da placa.

Dúvidas:

Ricardo Zelenovsky, Daniel Café e Eduardo Peixoto

1) O regulador 78M05 tem um *dropout* de 2 V, isto significa que precisamos de VS > 7,0 V e não apenas 5bV como se encontra indicado na Internet.
2) A tensão máxima de entrada no regulador é 35 V, então por que pedem para remover o jumper quando VS > 12 V?
3) Se a limitação de corrente é em 300 mA, então por que o manual afirma que se pode chegar acima de 500 mA com o uso de dissipador? Será que a limitação de corrente se faz baseada no aquecimento do CI?

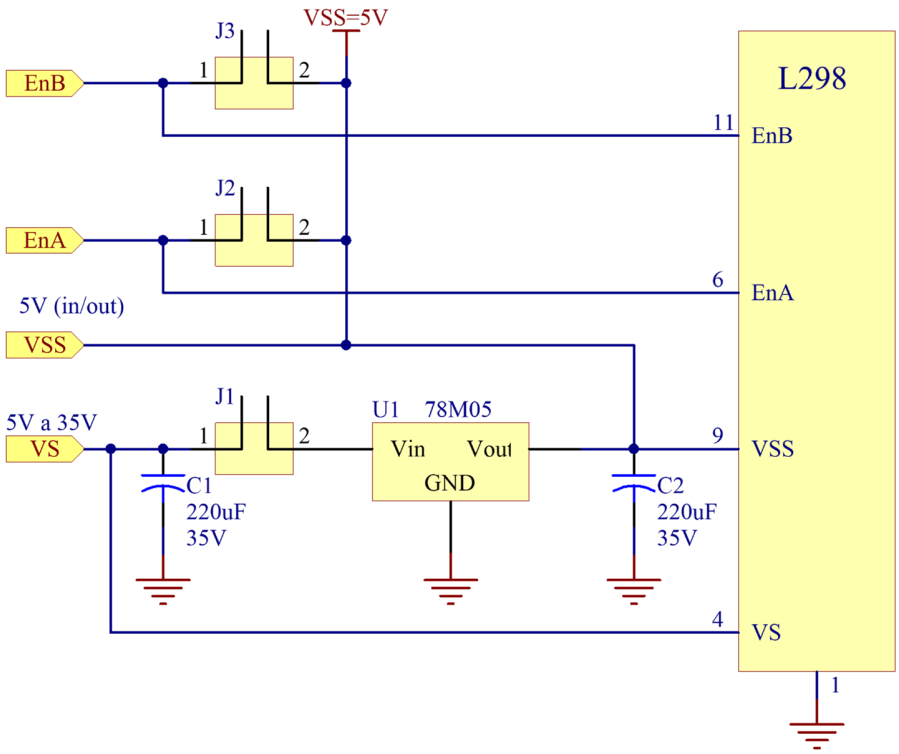

Figura M.13. Esquema do circuito de alimentação da placa com a Ponte H L298.

M.3. Exercícios Resolvidos

ER M.1. Usando um transistor, construa um ambiente para testar o acionamento de um motor DC. Este ambiente deve permitir variar a frequência do PWM, o ciclo de carga e ainda medir a rotação do motor. A descrição do pedido deste problema é um pouco genérica para possibilitar a diversidade nas soluções.

Solução: A proposta aqui apresentada é bem versátil, mas ainda simples. Vamos escolher o *timer* TB0 para gerar o PWM, pois ele tem dupla buferização. As chaves S1 e S2 do *Launchpad* servirão para aumentar ou diminuir a frequência do PWM. Como usamos as duas chaves para alterar a frequência, precisamos de alguma solução para variar o ciclo de carga. Bem, podemos usar um potenciômetro e o ADC para ler a tensão

sobre esse potenciômetro e fazer a correspondência com o ciclo de carga. Para medir a velocidade de rotação do motor sugerimos o uso de um pedaço de canudo de plástico, desses que estão proibidos, preso ao eixo do motor, que ao girar faz sombra sobre um sensor infravermelho. Cada vez que o sensor for sombreado, significa que o motor girou meia volta. Medindo o intervalo entre esses sombreamentos, podemos calcular a velocidade de rotação do motor.

Vamos reservar o *led* vermelho para indicar a frequência do PWM e o *led* verde para piscar sinalizando o funcionamento do sensor infravermelho. A exibição do ciclo de carga do PWM e da velocidade de rotação do motor é mais complicada, pois são numéricas. Deveríamos, então, usar um LCD, mas vamos propor uma solução mais simples. Hoje em dia, é fácil conseguir um multímetro que ofereça a opção de frequencímetro. Usando dois pinos do MSP, vamos gerar duas ondas quadradas, uma com frequência numericamente idêntica ao ciclo de carga (PWM) e a outra com frequência numericamente idêntica à velocidade em RPM. Basta ligar o frequencímetro a um desses pinos para saber o ciclo de carga ou a velocidade de rotação. O diagrama da Figura M.14 sumariza a proposta.

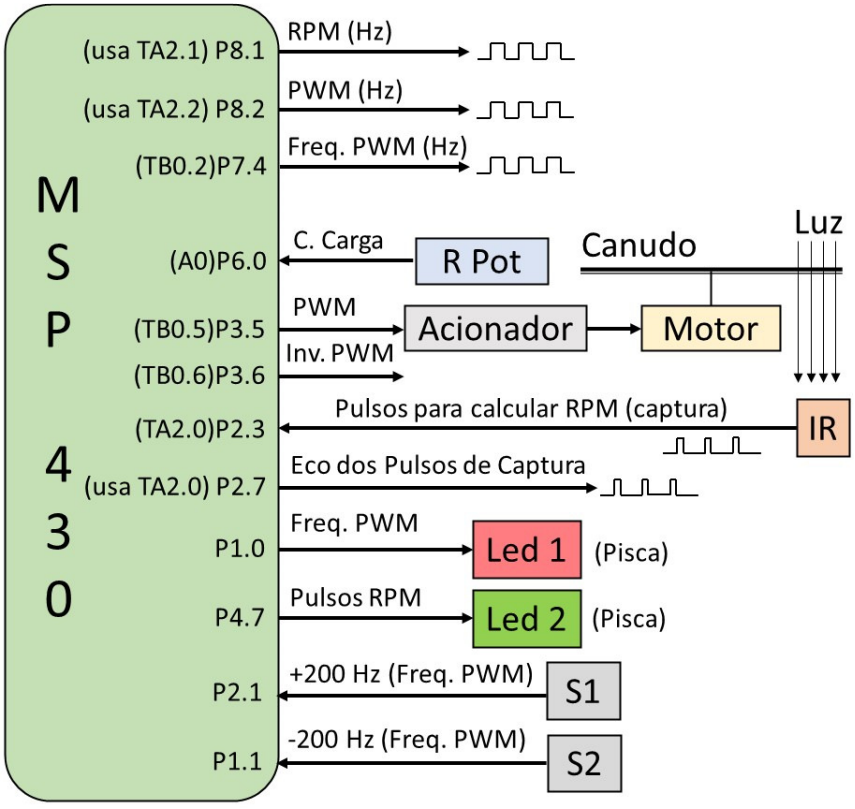

Figura M.14. Diagrama indicando a sugestão para ensaiar acionamento de motores DC.

As duas tabelas apresentadas a seguir resumem toda a interface. A Tabela M.2 apresenta as três entradas que permitem ao leitor controlar os parâmetros do experimento e ainda a

recepção do sinal gerado pelo sensor infravermelho que monitora a velocidade de rotação do motor. Já a Tabela M.3 apresenta todas as saídas disponíveis. Note que são geradas duas saídas PWM, sendo uma o inverso da outra, assim, o leitor pode selecionar uma delas, de acordo com a solução empregada no acionamento do seu motor. O ciclo de carga (PWM) é apresentado multiplicado por 10. Isso foi feito porque o frequencímetro apresentou dificuldades para medir frequências baixas. Assim, ciclo de carga de 10% é apresentado com uma onda quadrada na frequência de 100 Hz. É claro que 1.000 Hz correspondem a 100% de ciclo de carga.

Tabela M.2 Entradas de controle do ambiente de experimentação com motor DC.

Controle do Experimento (sinalização de entrada)		
Recurso	**Pino**	**Descrição**
ADC	(A0) P6.0	**Ciclo de Carga**: ADC converte a tensão sobre o potenciômetro. Código = 0 → 0% e código = 4095 → 100%.
TA0	(TA0.2) P2.3	**Velocidade de rotação**: Usa o recurso de captura de TA0. O intervalo entre duas capturas corresponde a meia volta.
GPIO	(S1) P2.1	**Freq + 200 Hz**: Aumenta a frequência do PWM em 200 Hz.
GPIO	(S2) P1.1	**Freq - 200 Hz**: Aumenta a frequência do PWM em 200 Hz.

Tabela M.3. Saídas de sinalização do ambiente de experimentação com motor DC.

Resultados do Experimento (sinalização de saída)		
Recurso	**Pino**	**Descrição**
TB0	(TB0.5) P3.5	**Saída PWM**: Gera PWM na frequência selecionada e com o ciclo de carga indicado no potenciômetro. Configurado com OUTMODE = 6.
TB0	(TB0.6) P3.6	**Saída PWM Invertida**: Idêntico ao anterior. Configurado com OUTMODE = 2.
TB0	(TB0.2) P7.4	**Frequência do PWM**: Gera onda quadrada na frequência usada gerar o PWM.
GPIO	P2.7	**Eco dos pulsos de Captura**: Se captura flanco de subida, faz P2.7 = 1 e se captura flanco de descida, faz P2.7 = 0. Com osciloscópio se pode confirmar se os pulsos do sensor IR estão sendo recebidos.
GPIO	P8.1	**Velocidade RPM**: Onda quadrada numa frequência numericamente idêntica à velocidade em RPM. Usa TA2.1.
GPIO	P8.2	**Ciclo de Carga**: Onda quadrada numa frequência numericamente idêntica a 10 vezes o ciclo de carga. Usa TA2.2.
GPIO	P4.7 (*Led* 2)	**Captura RPM**: Indica a captura dos pulsos gerados pelo sensor IR. Inverte o estado do *led* verde a cada "n" capturas.

GPIO	P1.0 (*Led* 1)	**Frequência do PWM**: Indica de forma simples a frequência do PWM. Sua sinalização é explicada na próxima tabela.

A Tabela M.4 resume a sinalização apresentada pelo *led* vermelho (*Led* 1), que indica a frequência usada para gerar o PWM, que é controlada pelas chaves S1 (+200 Hz) e S2 (-200 Hz). Com essa indicação, fica fácil para o leitor selecionar a frequência desejada, pois se caminha dentro de cada faixa com passos de 200 Hz.

Tabela M.4 Sinalização realizada pelo led vermelho (P1.0).

***Led* vermelho**	**Frequência do PWM**
Aceso	0 Hz
Apagado	0 Hz < frequência < 1 kHz
Pisca 1 vez	1 kHz ≤ frequência < 2 kHz
Pisca 2 vezes	2 kHz ≤ frequência < 3 kHz
Pisca 3 vezes	3 kHz ≤ frequência < 4 kHz
Pisca 4 vezes	4 kHz ≤ frequência < 5 kHz
Aceso	frequência = 5 kHz

A Figura M.15 apresenta duas fotos da montagem que foi realizada sobre uma *protoboard*. Como o motor ensaiado era pequeno, ele foi preso na própria *protoboard*. Com fita isolante, prendemos 4 *leads* (terminal de resistor), um em cada lado do corpo do motor e depois os encaixamos nos orifícios da placa. O sensor infravermelho (IR) foi também preso ao corpo do motor, de forma a ficar logo abaixo do plano de rotação do canudo. É interessante isolar um dos terminais do sensor IR.

No caso desta montagem, o canudo tem 6 cm e, na metade, foi feito um pequeno orifício de forma que nele encaixasse sob pressão o eixo do motor. Não foi necessário usar cola. Na figura, é possível ver que foi usada fita adesiva azul para garantir uma boa sombra. Como o canudo era translúcido, havia o temor de que o sombreamento não fosse bom. Porém, não se conseguiu melhora apreciável e essa fita pode ser dispensada. Finalmente, para iluminar o motor foi usada uma dessas luminárias de mesa. O esquema também funciona bem com uma lanterna e, em alguns casos, com o *led* branco do celular. É preciso ter em mente que a iluminação ambiente interfere com o experimento. Provavelmente, o sensor IR não funcionará bem em um ambiente aberto e exposto ao sol.

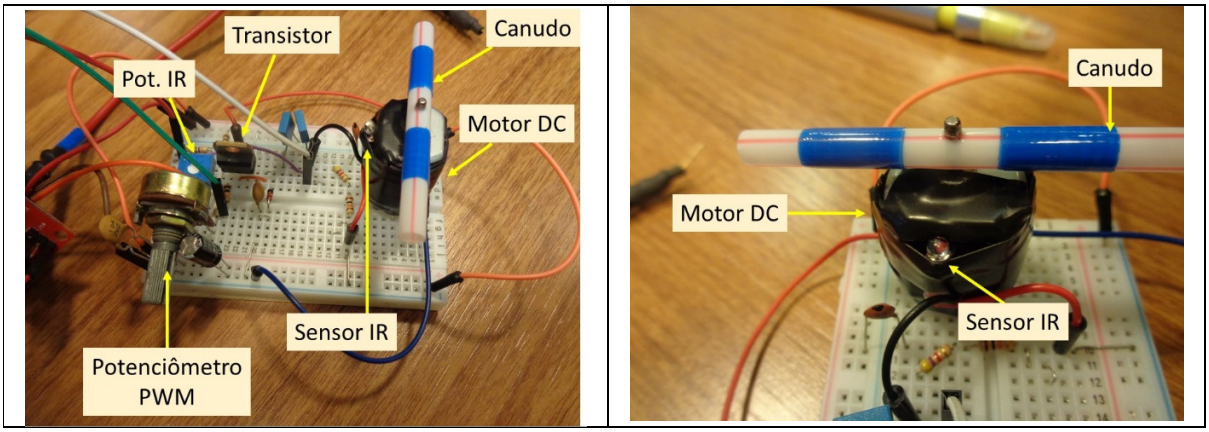

Figura M.15. Detalhes da montagem do experimento, em especial indicamos o motor com o canudo acoplado ao eixo e o sensor de infravermelho.

A Figura M.16 apresenta o esquema elétrico do ambiente proposto. Todas as entradas e saídas estão sinalizadas com setas vermelhas. A listagem apresentada mais adiante, traz o programa solução. Este programa ficou um pouco longo porque se quis beneficiar a clareza. Há o uso excessivo de variáveis globais. A seguir vamos comentar os principais itens deste ambiente de experimentação com motores.

Configuração da UCS:
Como são usadas várias interrupções e algumas ações são executadas dentro das rotinas dessas interrupções, foi considerado conveniente elevar o relógio da CPU para 20 MHz. Também era necessário um relógio um pouco mais rápido para o *timer*. Assim, a função `void mclk_20MHz(void)` faz a seguinte configuração na UCS:

- MCLK = 20 MHz (usando o XT2 para acionar DCO com FLLN=5 e FLLD=1);
- SMCLK = 4 MHz (usando o cristal 2 = XT2 = 4 MHz) e
- ACLK = 32.768 Hz (usando o cristal 1 = XT1 = 32.768 Hz)

Frequência do PWM:
Para este experimento se considerou razoável que a frequência do PWM ficasse na faixa de 0 Hz até 5 kHz. A variável global `int freq`, inicializada com zero, é responsável por registrar a frequência atual. Assim, ao ser ligado nada acontece (`freq=0`), apenas o *led* vermelho fica aceso para indicar 0 Hz. Acionamentos de S1 e S2, somam ou subtraem 200 à essa variável. A cada alteração, os limites de 0 Hz e 5 kHz são verificados e depois o valor de TB0CCR0 é atualizado de acordo com a equação abaixo. Vale lembrar que TB0 está contando no modo Ascendente/Descendente e por isso surge o número 2 no denominador.

$$TB0CCR0 = \frac{SMCLK}{2 * freq} - 1$$

Ciclo de Carga e Geração do PWM:
O programa armazena o ciclo de carga do PWM na variável global `float pwm` (cuidado para não confundir esse nome). O valor desta variável é atualizado a partir da tensão sobre o potenciômetro de 20 kΩ que é lida pelo ADC. Na Figura M.16 note que o

potenciômetro (R1 = 20 kΩ) está ligado aos 3,3 V. Quando o potenciômetro está no mínimo, a tensão lida é 0 V. Porém, com o potenciômetro no máximo, a tensão será de 3,3 V. O potenciômetro tem valor alto para minimizar o dreno de corrente, que é de apenas 165 µA (3,3 / 20k).

O ADC está programado para 8 conversões repetidas (*auto scan*). As conversões são disparadas por TA0.1, na taxa de 1 kHz. Quando acontece a oitava conversão, os dados são transferidos por DMA para o vetor `int adc_vet[]`. Assim, temos um novo vetor a cada 8 ms, o que é rápido o suficiente para ler as variações no potenciômetro. Cada repetição do laço do programa principal calcula a média dessas 8 leituras e atualiza a variável `int adc`, assim nessa variável temos sempre a média das últimas 8 leituras. É importante usar a média das medidas para minimizar a influência do ruído. O ciclo de carga é calculado com a equação seguinte, que usa a precisão de "float".

$$Ciclo\ de\ Carga = pwm = 1.000\% \frac{adc}{4.095{,}0}$$

Usa-se o valor 1.000% e não 100% porque, como já dissemos, o valor do pwm é sempre armazenado multiplicado por 10. Isso facilita a geração de uma onda quadrada com frequência numericamente igual ao Ciclo de Carga.

Com a frequência e o ciclo de carga especificados, usamos os comparadores TB0.5 e TB0.6 para gerar o PWM. O comparador TB0.5 opera no modo de saída 6 e o TB0.6 opera no modo de saída 2. Esses dois modos são complementares, assim, o leitor pode escolher aquele que for mais adequado para o seu circuito de acionamento do motor. Lembramos que o contador TB0R está no Modo 3 (ascendente/descente), que é o mais adequado para o controle da velocidade de motores. O valor a ser carregado nos comparadores é dado pela conta abaixo, que usa que usa a precisão de "float" (lembre-se de que o ciclo de carga – variável `pwm` – está multiplicada por 10). Já vimos que o valor de TB0CCR0 é calculado em função da frequência (variável `freq`) do PWM.

$$TB0CCR5 = TB0CCR6 = TB0CCR0\left(\frac{1.000{,}0 - pwm}{1.000{,}0}\right)$$

Captura do Sinal do Sensor Infravermelho:
A rotação do motor é monitorada por Q1 (TIL 78) que é um fototransistor infravermelho (IR). O funcionamento de um fototransistor é bastante simples. Ele é fabricado em um encapsulamento transparente de forma que se possa projetar luz sobre o cristal de sua base. Quando há luz, a incidência de fótons na base libera portadores na estrutura cristalina e o transistor entra em condução. Uma boa quantidade de luz pode provocar a saturação. Por outro lado, quando há pouca luz incidente, não há portadores suficientes e o transistor fica cortado. Pela conexão apresentada na Figura M.16, se pode ver que quando o transistor está em condução o pino P2.3 vai para nível baixo. Porém, quando não há luz, o transistor corta e a tensão em P2.3 vai para 3,3 V.

O resistor R2 e o potenciômetro R3 controlam a sensibilidade do sensor. Quanto maior for a soma destes dois resistores, menor é quantidade de luz necessária para saturar o fototransistor. Não vamos precisar de uma fonte de luz específica. As luminárias que usamos no dia a dia geram uma grande quantidade de energia na faixa infravermelha. No caso deste experimento, foi usada uma luminária comum de mesa jogando luz

diretamente sobre o fototransistor. Com essa abundância de luz, foi fácil conseguir a saturação. Também pode-se usar uma lanterna comum e até a lanterna do celular.

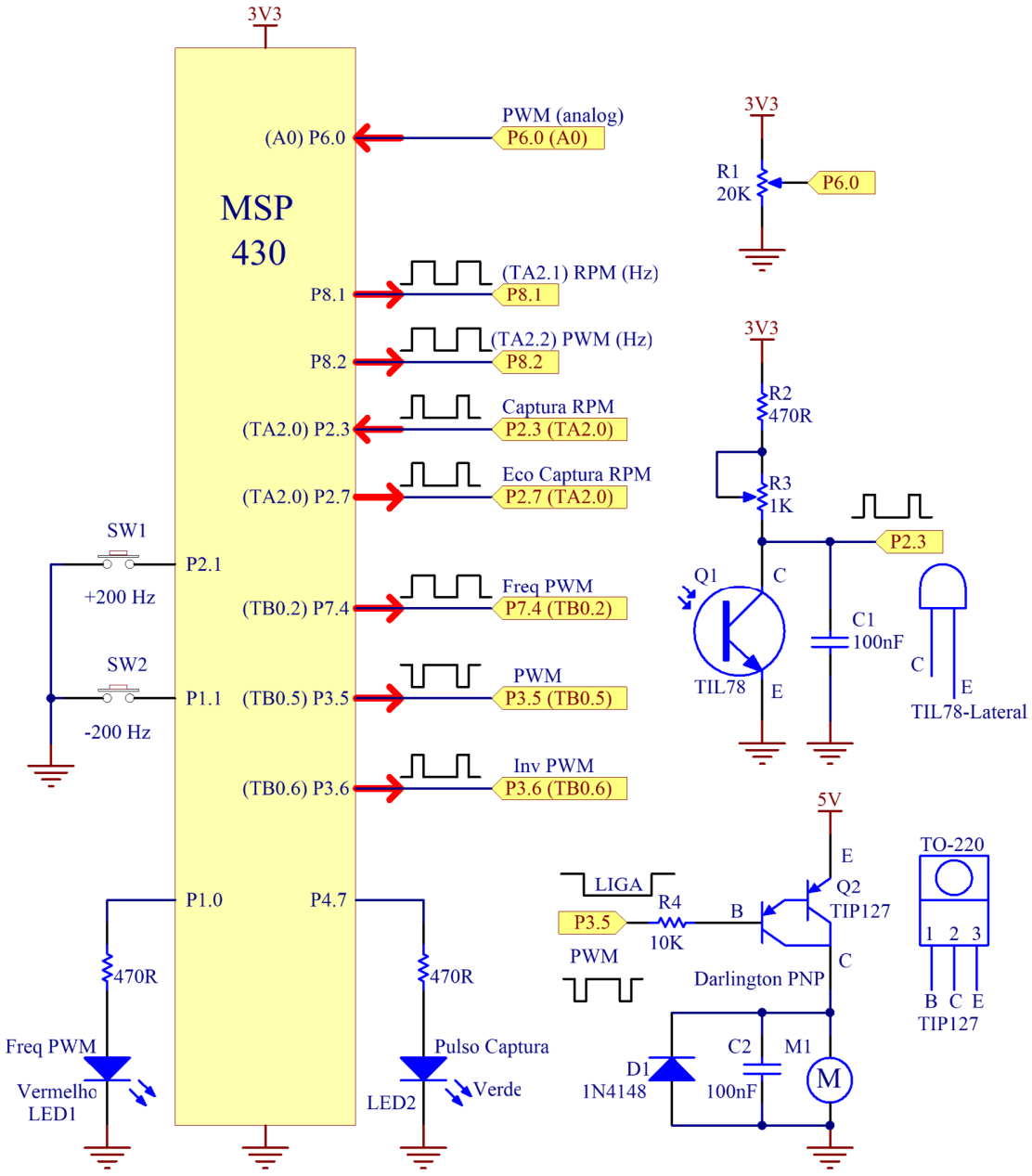

Figura M.16. Esquema elétrico da sugestão para ensaiar o acionamento de motores DC.

Como já foi exposto, ao girar, o eixo do motor movimenta o canudo que faz sombra sobre o sensor quando passa sobre ele. O ponto importante é conseguir que o fototransistor entre em corte quando o canudo o estiver sombreando. Por esse motivo o fototransistor foi montado no corpo do motor, bem próximo ao plano de giro do canudo. Com este

sensor funcionando, basta usar o modo de captura de um *timer* para pedir o intervalo entre dois pulsos consecutivos e calcular a velocidade de rotação. Porém, nem tudo é assim tão simples.

Como ponto de partida, era preciso saber o que exatamente o *hardware* de captura do *timer* está "vendo". Por isso, o recurso de captura, além de medir o intervalo de tempo, também ecoa em P2.7 o que o MSP estava recebendo. Assim, o programa faz P2.7 = 0, ao capturar um flanco de descida e P2.7 = 1 ao capturar um flanco de subida. Com um osciloscópio é possível ver o que realmente estava acontecendo.

As Figuras M.17 e M.18 apresentam as medições realizadas com o osciloscópio mostrando em azul a saída do sensor IR, que é entregue ao pino P2.3 e em vermelho o eco deste sinal, no pino P2.7, gerado pelo sistema de captura do MSP. Como se pode observar, o resultado foi muito bom: o sinal em vermelho está bem coerente com o azul. O leitor pode notar que o sinal em azul apresenta alguns pulsos de baixa tensão, como se fosse modulado por uma onda quadrada. Esses pulsos são interferência do PWM que está acionando o motor. Era de se esperar que sua amplitude fosse muito mais baixa, mas eles não interferiram com a medição. Uma possível explicação para esses pulsos é a quantidade de corrente consumida pelo motor cada vez que o seu transistor é acionado e que acaba por perturbar a fonte de 5 V (que vem do USB).

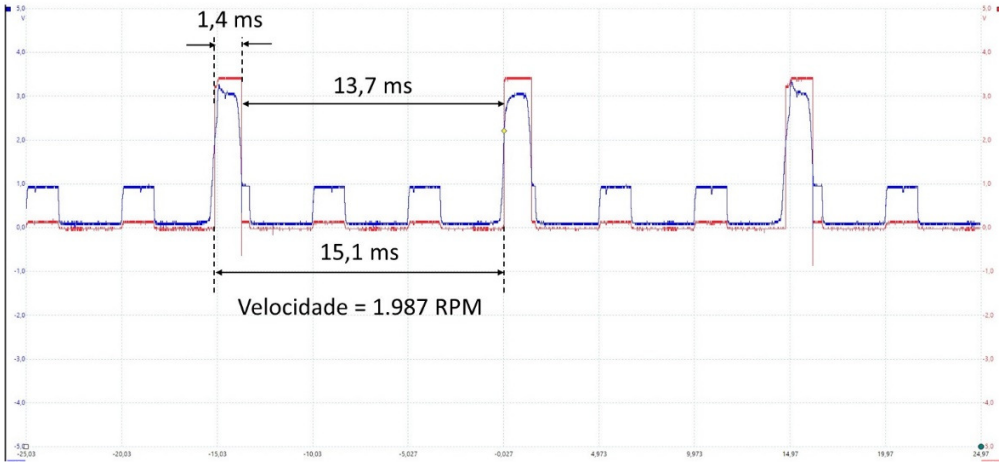

Figura M.17. Imagem do osciloscópio mostrando em azul a saída do sensor IR e em vermelho as capturas realizadas pelo timer do MSP, com o motor em 1.987 RPM.

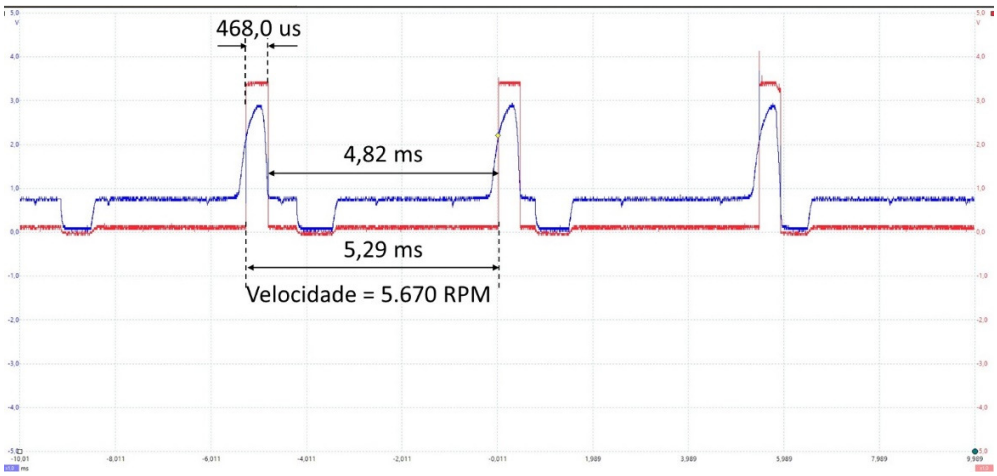

Figura M.18. Imagem do osciloscópio mostrando em azul a saída do sensor IR e em vermelho as capturas realizadas pelo timer do MSP, com o motor em 5.670 RPM.

Porém, a experimentação mostrou que, em alguns casos, existem problemas. As Figuras M.19 e M.20 apresentam dois casos problemáticos. Na Figura M.19, as setas indicam que o MSP não capturou corretamente o sinal do sensor IR. Isso, entretanto, não seria um grande problema, já que, em todo o caso, o flanco de subida estava no instante correto. Já a Figura M.20 apresenta um caso mais preocupante. As setas em linha contínua mostram dois instantes em que aconteceram múltiplas capturas. Na figura não é possível perceber com clareza, mas um flanco na linha azul resultou em vários flancos na linha vermelha. A Figura M.21 apresenta este fato com mais detalhes. Essas capturas equivocadas ficam cada vez mais frequentes à medida que se aumenta a rotação do motor.

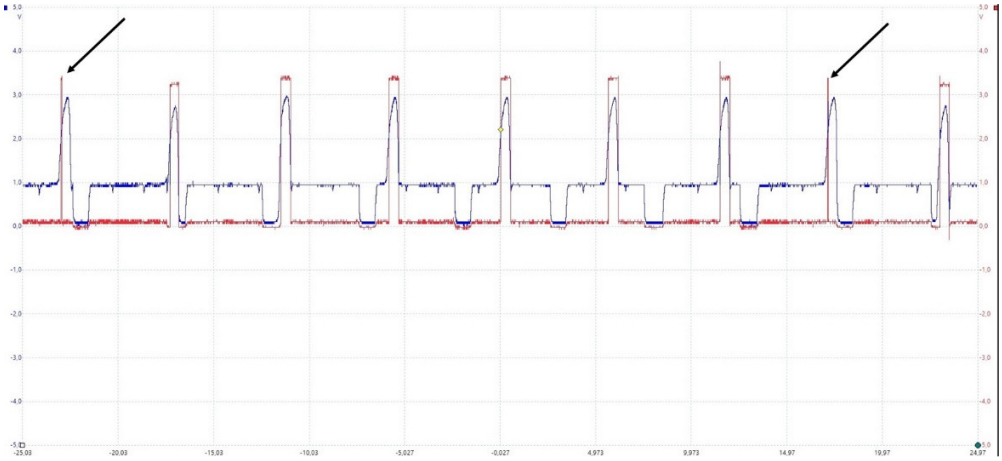

Figura M.19. Nesta imagem estão sinalizados dois problemas com a captura do sinal gerado pelo sensor IR.

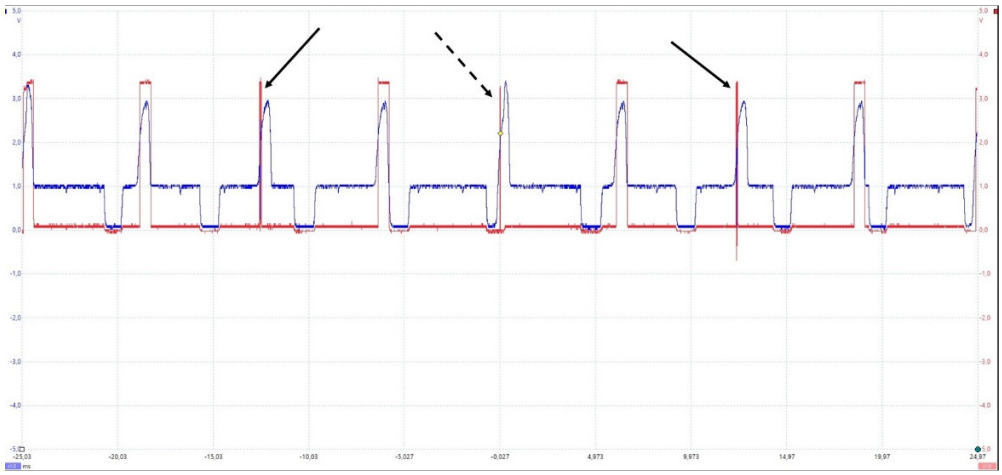

Figura M.20. Aqui é apresentado um problema ainda mais grave, as setas em linha contínua indicam instantes em que o MSP fez várias capturas (curtas) seguidas. Não é possível ver com detalhes, mas a linha vermelha subiu e desceu várias vezes em cada evento (note a linha mais grossa).

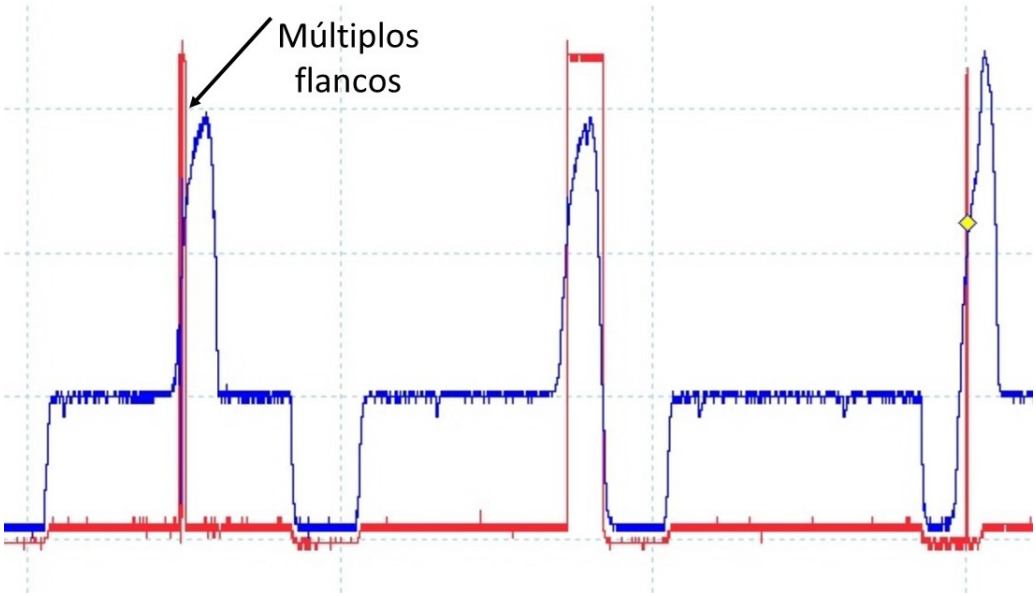

Figura M.21. Detalhe de uma captura com problemas.

Em termos práticos, esses pulsos problemáticos não podem ser levados em conta durante o cálculo da velocidade de rotação do motor. Era, portanto, necessário conseguir uma forma de se separar as capturas corretas das equivocadas. O critério usado foi o de comparar a largura do pulso em nível alto com a largura do pulso em nível baixo. Vamos usar os dados das Figuras M.17 e M.18, como mostrado a seguir.

Figura M.17 → $Relação = \frac{T_{alto}}{T_{baixo}} = \frac{13,7}{1,4} = 9,79$

Figura M.18 → $Relação = \frac{T_{alto}}{T_{baixo}} = \frac{4,82}{0,47} = 10,26$

Parece existir uma relação quase que constante. Na verdade, a experimentação mostrou que essa relação pode ficar bem abaixo de 9 e ir bem acima de 10. O posicionamento da luminária, a intensidade de luz e o tipo da lâmpada interferem. Por isso, o programa permite ao leitor definir os valores máximo e mínimo para que um par de pulsos (alto e baixo) possa ser considerado válido. Veja que na listagem, são definidas as duas constantes que estão copiadas abaixo. Esses valores funcionaram muito bem, entretanto, o usuário pode variar essas constantes para seu caso particular.

```
#define RMIN 5    //Mínima relação Low/High (Pulsos IR)
#define RMAX 20   //Máxima relação Low/High (Pulsos IR)
```

Para sinalizar ao leitor que a captura está funcionando, o *led* verde é invertido a cada 10 capturas. Assim, enquanto este *led* piscar, é porque a captura está acontecendo. Esse ritmo de pisca está definido com a constante `VD_FREQ` e pode ser alterado.

Para finalizar, é preciso comentar que a captura é realizada com o comparador 0 de TA2 (TA2.0). Para isso, o relógio de TA2 foi programado para 500 kHz (SMCLK/8) e o contador TA2 foi especificado para operar no modo Contínuo. Sob essas condições, o intervalo máximo entre duas capturas seria de 131 ms (2µs x 65.536) que, em alguns casos pode ser pequeno, pois corresponde a 457 RPM. Por isso foi usado um contador auxiliar em software, denominado `unsigned int ta2_aux`, que usa a interrupção TAIFG para contar as ultrapassagens de TA2. Assim, a captura concatena TA2CCR0 com `ta2_aux` o que resulta em 32 bits (variável long) e fornece uma excelente faixa de contagem.

Cálculo da Velocidade de Rotação do Motor:
A interrupção de captura, cada vez que identifica pulsos com perfil correto, calcula o intervalo entre os flancos de subida consecutivos, o armazena no vetor `rpm_vet[]` e depois sinaliza tornando verdadeiro o flag `rpm_flag`. Assim, o vetor `rpm_vet[]` tem sempre os últimos intervalos medidos. A quantidade de medições a ser armazenada é definida com a constante `RPM_QTD`, que para este experimento está em 16.

O laço principal, cada vez que percebe que o flag `rpm_flag` foi alterado para verdadeiro, calcula média das capturas e a usa para calcular a velocidade em RPM, com a seguinte conta, onde TA2CLK (500 kHz) é frequência de relógio que aciona o contador TA2.

$$RPM = 60\frac{TA2CLK}{2 \times média}$$

Sinalização de RPM e Ciclo de Carga (PWM):
Para indicar a velocidade e o ciclo de carga é preciso gerar ondas quadradas em frequências numericamente idênticas à RPM e ao Ciclo de Carga (PWM) pelos pinos P8.1 e P8.2. Os recursos do *timer* TA2 são assim usados:

RPM: TA2CCR1 → vai gerar uma onda quadrada em P8.1 e
PWM: TA2CCR2 → vai gerar uma onda quadrada em P8.2.

Como o contador TA2 opera em modo Contínuo, usaremos as interrupções para incrementar os valores de TA2CCR1 e TA2 CCR0, como foi feito no ER 7.4.

Para melhor compreensão, vamos a um exemplo. Para indicar 1.000 RPM precisamos gerar em P8.1 uma onda quadrada de 1 kHz. Isto significa que a cada 500 µs invertemos o estado deste pino. No intervalo de 500 µs, TA2 realiza 250 contagens. Precisamos então provocar uma interrupção a cada 250 contagens. A solução então é simples: começamos com TA2CCR1 = 0 e, a cada interrupção, invertemos P8.1 e somamos 250 a TA2CCR1. Neste caso, vamos dizer que o passo é de 250 contagens. Então, tudo se resume em calcular o valor do passo.

Foram criadas funções para calcular o passo para cada caso:
`unsigned int pwm_ta2p2(void)` → calcula o passo para TA2CCR2 e
`unsigned int rpm_ta2p1(void)` → calcula o passo para TA2CCR1 e

Se o leitor verificar essas duas funções na listagem a seguir, irá verificar que elas são quase idênticas (diferem apenas no limite máximo de RPM e PWM). Entretanto, elas foram escritas separadas para o caso de que, num experimento futuro, precisem ser diferentes. Os limites são verificados, para evitar resultados absurdos. No presente caso de considera o mínimo de 0,5% de ciclo de carga ou de 5 RPM. O máximo é de 100,0% do ciclo de carga e a rotação de 10.000 RPM. O cálculo do passo é simples, onde TA2CLK é de 500 kHz e o ciclo de carga é aqui representado pela variável PWM.

$$PASSO_{RPM} = \frac{TA2CLK}{2 \times RPM} \; e \; PASSO_{PWM} = \frac{TA2CLK}{2 \times PWM}$$

As variáveis globais `unsigned int ta2_rpm_passo` e `unsigned int ta2_pwm_passo` são as responsáveis por armazenar o valor do passo que deve ser adicionado ao TA2CCR1 ou TA2CCR2 a cada interrupção.

Sinalização da Frequência do PWM com o *Led* Vermelho:
É necessária uma forma simples e eficiente de sinalizar ao usuário a faixa de frequência em que ele está operando o PWM. Isto é feito com o *led* vermelho piscando em uma sequência especial. A Tabela M.4 (já apresentada), especifica essa sinalização.

A forma simples de fazer isso foi usando a interrupção por ultrapassagem (TAIFG) de TA2, que está operando no modo contínuo, na frequência de 500 kHz. Nesse caso, a interrupção acontece a cada 131 ms (2 µs x 65.536). Foi criada a variável global `unsigned int vm[10]`, que armazena o padrão para acender ou apagar esse *led*. Usando um contador, a cada interrupção, uma posição deste vetor é consultada. Se essa posição estiver em 1, o *led* vermelho é aceso e se estiver em 0, o *led* é apagado. Assim, basta escrever neste vetor o padrão que se quer apresentar.

Exemplos:
2 piscadas seguidas: `vm[10]={1,0,1,0,0,0,0,0,0,0}`
3 piscadas seguidas: `vm[10]={1,0,1,0,1,0,0,0,0,0}`
aceso: `vm[10]={1,1,1,1,1,1,1,1,1,1}`

Listagem do programa solução para o ER M.1
(As duas faixas coloridas são para referência no próximo exemplo)

```
// ER M.1
// Ambiente para ensaio de motor DC
// (TB0.2)  P7.4 --> Frequência do PWM
// (TB0.5)  P3.5 --> Saída PWM
// (TB0.6)  P3.6 --> Saída PWM invertida
// (A0-ADC) P6.0 <-- Pot para ciclo de carga
// (TA0.1)       --> Disparar ADC
// (TA0.2)  P2.3 <-- Pulsos do sensor IR (RPM)
// (TA2.1)  P8.1 --> 10*PWM em Hz (onda quadrada)
// (TA2.2)  P8.2 --> RPM em Hz (onda quadrada)
//  (S1)    P2.1 <-- +200 Hz na Freq PWM
//  (S2)    P1.1 <-- -200 Hz na Freq PWM
// (Led  1)  P1.0 --> Pisca para indicar faixa freq PWM
// (Led  2)  P4.7 --> Pisca para indicar sensor IR

#include <msp430.h>

#define TRUE    1
#define FALSE   0

#define SMCLK        4E6    //SMCLK = 4 MHz
#define TA2CLK (SMCLK/8)    //TA2CLK = 500 kHz
#define T1ms        3999    //TA1 --> Interrup ADC = (4E6/1000)-1

#define ADC_TOTAL      8    //Qtd de conv média ciclo carga
#define ADC12MEM0ADDR  (SFR_FARPTR) 0x0720 //ADCMEM0

#define RMIN           5    //Mín relação Low/High (Pulsos IR)
#define RMAX          20    //Máx relação Low/High (Pulsos IR)
#define RPM_QTD       16    //Tamanho vetor guardar rpm
#define RPM_SEM_MAX 5000    //Repetições para indicar RPM=0
#define VD_FREQ      10     //Ritmo led  verde pisca

#define ABT            1    //Constante representa SW Aberta
#define FEC            0    //Constante representa SW Fechada
#define DBC         2000    //Atraso para o debounce

// Protótipo das funções
void vm_freq(void);           //Pisca led  vermelho
unsigned int pwm_ta2p2(void); //Passo para RPM
unsigned int rpm_ta2p1(void); //Passo para 10*PWM
void tb0_config(void);        //TB0
void ta2_config(void);        //TA2
void ta0_config(void);        //TA0
void dma_config(void);        //Config DMA
void adc_config(void);        //Config ADC
void mclk_20MHz(void);        //Config UCS
```

```
int check_s1(void);           //Checar chave S1
int check_s2(void);           //Checar chave S2
void config_pinos(void);      //Config S1, S2, Led 1 e Led 2
void debounce(void);          //Delay para debounce

// Variáveis Globais
volatile int freq=0;                  //Freq do PWM
volatile int adc,adc_vet[ADC_TOTAL];  //Conversões ADC
volatile float pwm=500;               //10*PWM = Valor atual do PWM
volatile unsigned int rpm;            //Velocidade em RPM
volatile unsigned int ta2_rpm_passo=1000;   //Passo para gerar RPM Hz
volatile unsigned int ta2_pwm_passo=1000;   //Passo para gerar 10*PWM Hz
volatile unsigned int ta2_aux=0;      //Contador overflow TA2 para RPM
volatile unsigned int rpm_vet[RPM_QTD]; //Vetor com capturas em contagens
volatile unsigned int id;               //Indexador vetor captura
volatile unsigned long tp1,tp2,tp3,rel; //Instantes de captura
volatile unsigned long dif1,dif2;       //Largura pulsos capturados
volatile unsigned int rpm_flag;         //Marcar que RPM foi calculado

volatile int vd_cont;                       //Contar para piscar Verde
volatile unsigned int vm[10],vm_cont=0;     //Piscar vermelho

int ps1=ABT,ps2=ABT;        //Estado anterior das chaves

int main(void)
{
    unsigned int i,soma,rpm_cont=0;
    volatile unsigned long rpm_aux; //aux no cálculo RPM
    WDTCTL = WDTPW | WDTHOLD;       // stop watchdog timer
    mclk_20MHz();
    config_pinos();
    tb0_config();
    ta2_config();
    ta0_config();
    dma_config();
    adc_config();
    vm_freq();

    __enable_interrupt ( );
    while(1){
        if (check_s1()) freq+=200;
        if (check_s2()) freq-=200;
        if (freq<0)     freq=0;     //Limite 0
        if (freq>5000)  freq=5000;  //Limite 5000
        if (freq != 0)  TB0CCR0=(SMCLK/(2*freq))-1;
        else            TB0CCR0=0;
        vm_freq();      //Vermelho indica freq

        soma=0;
```

```
        for (i=0; i<ADC_TOTAL; i++)
            soma += adc_vet[i];
        adc=soma/ADC_TOTAL;          //Média das ADC_TOTAL conversões
        pwm=1000*(adc/4095.);        //Calcula ciclo de carga
        ta2_pwm_passo=pwm_ta2p2();   //Passo TA2.2 para 10*PWM Hz
        TB0CCR2=TB0CCR0/2;           //Freq do PWM
        TB0CCR5=TB0CCR0*((1000.-pwm)/1000); //Ciclo de carga TB0.5
        TB0CCR6=TB0CCR0*((1000.-pwm)/1000); //Ciclo de carga TB0.6

        if (rpm_flag==TRUE){
            rpm_flag=FALSE;
            rpm_aux=0;
            for (i=0; i<RPM_QTD; i++)   rpm_aux += rpm_vet[i];
            rpm_aux = rpm_aux/RPM_QTD;      //Média das capturas
            rpm = ((float)TA2CLK/(2*rpm_aux))*60;   //Calcular RPM
            ta2_rpm_passo=rpm_ta2p1();              //Programar RPM Hz
        }
        else{
            rpm_cont++;                     //Laço sem captura
            if (rpm_cont == RPM_SEM_MAX){   //Ver limite
                rpm_cont=0;                 //Zerar RPM
                rpm=0;
                ta2_rpm_passo=rpm_ta2p1();  //Programar RPM Hz
            }
        }
     }
     return 0;
}

// Led  Vermelho pisca para indicar freq
// Calcula qtd de piscadas
// 0 Hz ou 5000 Hz --> Aceso
// 0 < 999 Hz --> Apagado
// 1000 < 1999 --> 1 vez      2000 < 2999 --> 2 vezes
// 3000 < 3999 --> 3 vezes    4000 < 4999 --> 4 vezes
void vm_freq(void){
    unsigned int i;
    for (i=0; i<10; i++)    vm[i]=0;
    switch(freq/1000){
    case 1: vm[0]=1; break;                          //1X
    case 2: vm[0]=vm[2]=1; break;                    //2X
    case 3: vm[0]=vm[2]=vm[4]=1; break;              //3X
    case 4: vm[0]=vm[2]=vm[4]=vm[6]=1; break;        //4X
    case 5: vm[0]=vm[2]=vm[4]=vm[6]=vm[8]=1; break; //5X
    default:   break;
    }
    if (freq==0 || freq==5000)                       //Sempre aceso
        for (i=0; i<10; i++)    vm[i]=1;
}
```

```
// PWM, calcular passo para TA2 gerar Freq = 10*PWM
unsigned int pwm_ta2p2(void){
    unsigned int aux;
    if (pwm < 5)             aux=TA2CLK/(2*5);
    else if (pwm > 1000.)    aux=TA2CLK/(2*1000);
    else                     aux=TA2CLK/(2*pwm);
    return aux;
}

// RPM, calcular passo para TA2 gerar Freq = RPM
unsigned int rpm_ta2p1(void){
    unsigned int aux;
    if (rpm < 5)             aux=TA2CLK/(2*5);
    else if (pwm > 10000.)   aux=TA2CLK/(2*10000);
    else                     aux=TA2CLK/(2*rpm);
    return aux;
}

// TA2.1 e TA2.2 - Interrupção
// Gerar onda quadrada para RPM em Hz e 10*PWM em Hz
// Também controla pisca do led  vermelho (P1.0)
//#pragma vector = TIMER2_A1_VECTOR
#pragma vector = 43
__interrupt void ta2_ccifg(void){
    switch(TA2IV){
    case 2:     P8OUT^=BIT1;   //TA2.1 = RPM = P8.1
                TA2CCR1+=ta2_rpm_passo;
                break;
    case 4:     P8OUT^=BIT2;   //TA2.2 = 10*PWM = P8.2
                TA2CCR2+=ta2_pwm_passo;
                break;
    case 0XE:   ta2_aux++;
                P1OUT &= ~BIT0; //Piscar led  vermelho
                if (vm[vm_cont] == 1) P1OUT |= BIT0;
                if (++vm_cont == 10)  vm_cont=0;
                break;
    default: break;
    }
}

// TA2.0 Interrupt - Captura cálculo RPM
//#pragma vector = TIMER2_A0_VECTOR
#pragma vector = 44
__interrupt void ta2_ccr0(void){
    if ((TA2CCTL0&CM_3)==CM_1){           //Flanco subida
        tp2=((long)ta2_aux<<16)+TA2CCR0;  //Ler captura
        TA2CCTL0 = CM_2|SCS|CAP|CCIE;     //Mudar para flanco subida
        P2OUT |= BIT7;                    //P2.7 = 1
        if (++vd_cont == VD_FREQ){        //Contar verde
```

```
            P4OUT ^= BIT7;                  //Inverter led  verde
            vd_cont=0;                      //Zerar contador
        }
    }
    else{
        tp3=((long)ta2_aux<<16)+TA2CCR0;  //Flanco descida
        dif1=tp2-tp1;                     //Diferenca 1
        dif2=tp3-tp2;                     //Diferença 2
        if (dif1 != 0)  rel=dif1/dif2;    //Relaçao entre elas
        if ((rel>RMIN) && (rel<RMAX)){    //Dentro da faixa esperada?
            rpm_vet[id++]=tp3-tp1;        //Armazenar captura
            rpm_flag=TRUE;                //Avisar que houve captura
        }
        if (id==RPM_QTD)  id=0;           //Verificar limite do vetor
        tp1=tp3;                          //Guardar para prox captura
        TA2CCTL0 = CM_1|SCS|CAP|CCIE;     //Mudar para flanco descida
        P2OUT &= ~BIT7;                   //P2.7 = 0
    }
}

// TB0 config --> Gerar PWM
// (P7.4) TB0.2 --> Freq do PWM
// (P3.5) TB0.5 --> PWM Modo 6
// (P3.6) TB0.6 --> PWM Modo 3 (invertido)
void tb0_config(void){
    TB0CTL = TBSSEL_2 | MC_3;       //SMCLK e MC=3
    TB0CCTL0= 0;
    TB0CCR0=1000;
    TB0CCTL2= CLLD_2 | OUTMOD_6;    //P7.4 = Freq do PWM
    TB0CCR2=TB0CCR0/2;
    TB0CCTL5= CLLD_2 | OUTMOD_6;    //P3.5 = PWM acionar motor
    TB0CCR5=TB0CCR0/2;
    TB0CCTL6= CLLD_2 | OUTMOD_2;    //P3.6 = Inv PWM acionar motor
    TB0CCR6=TB0CCR0/2;
    ;
    P7DIR |= BIT4;                  //P7.4 = Freq PWM
    P7SEL |= BIT4;
    ;
    P3DIR |= BIT6|BIT5;             //P3.5 = PWM acionar motor
    P3SEL |= BIT6|BIT5;             //P3.6 = Inv PWM acionar motor
}

// TA2 config em 4 MHz Modo Contínuo
// TA2.0 --> (P2.3) Captura para calcular RPM
// TA2.1 --> RPM em P8.1
// TA2.2 --> 10*PWM em P8.2
void ta2_config(void){
    TA2CTL = TASSEL_2 | ID_3 | MC_2 | TAIE;   //SMCLK/8, MC=2, hab int
    TA2CCR1 = 0;                //P8.1 = RPM em Hz
    TA2CCTL1 = CCIE;            //Hab int CCR1 (RPM)
```

```
    TA2CCR2 = 0;                //P8.2 = 10*PWM em Hz
    TA2CCTL2 = CCIE;            //Hab int CCR2 (PWM)
    P8DIR |= BIT2 | BIT1;       //P8.1=P8.2=Saída
    P8OUT &= ~(BIT2 | BIT1);    //P8.1=P8.2=0
    ;
    TA2CCTL0 = CM_2 | SCS | CAP | CCIE; //Captura pulsos RPM
    P2DIR &= ~BIT3;     //P2.3 recebe pulsos IR
    P2REN |= BIT3;
    P2OUT |= BIT3;
    P2SEL |= BIT3;
    P2DIR |= BIT7;      //Eco pulsos de captura
    P2OUT &= ~BIT7;     //Eco = 0
}

// TA0.1 --> disparar ADC a cada 1 mseg
void ta0_config(void){
    TA0CTL = TASSEL_2 | MC_1;   //SMCLK e MC=1
    TA0CCR0 = T1ms;             //T=1ms --> 1 kHz
    TA0CCTL1 = OUTMOD_6;        //Hab int CCR0
    TA0CCR1= TA0CCR0/2;
}

// DMA transfer ADC para adc_vet[8]
void dma_config(void){
    DMACTL0 = DMA0TSEL_24;    //ADC dispara DMA
    DMA0CTL = DMASRCINCR_3 |  //Incr end fonte
              DMADSTINCR_3 |  //Incr end destino
              DMADT_5      |  //Modo Bloco Repetido
              DMAEN ;         //Habilitar DMA
    DMA0SA = ADC12MEM0ADDR;   //Endereço ADC MEM0
    DMA0DA = &adc_vet;        //Endereço destino
    DMA0SZ = ADC_TOTAL;       //Qtd. de transferências
}

// ADC para 8 conversões disparadas por TA0.1 (
void adc_config(void){
    ADC12CTL0  &= ~ADC12ENC;          //Desab p/ configurar
    ADC12CTL0 = ADC12ON;              //Ligar ADC
    ADC12CTL1 =  ADC12CSTARTADD_0 |
                 ADC12SHS_1       |   //TA0.1 Dispara ADC
                 ADC12SHP         |   // S/H usar timer
                 ADC12DIV_7       |   //Clock ADC Divisor = 8
                 ADC12SSEL_3      |   //Clock ADC = SMCLK
                 ADC12CONSEQ_3;       //Seq de canais
    ADC12CTL2  =  ADC12TCOFF |   // Desligar sensor temperatura
                  ADC12RES_2;    // Resolução 12-bit
    ADC12MCTL0 =            ADC12SREF_0 | ADC12INCH_0;
    ADC12MCTL1 =            ADC12SREF_0 | ADC12INCH_0;
    ADC12MCTL2 =            ADC12SREF_0 | ADC12INCH_0;
```

```
    ADC12MCTL3 =            ADC12SREF_0 | ADC12INCH_0;
    ADC12MCTL4 =            ADC12SREF_0 | ADC12INCH_0;
    ADC12MCTL5 =            ADC12SREF_0 | ADC12INCH_0;
    ADC12MCTL6 =            ADC12SREF_0 | ADC12INCH_0;
    ADC12MCTL7 = ADC12EOS | ADC12SREF_0 | ADC12INCH_0;
    P6SEL |= BIT0;              //Deslig parte digital de P6.0
    ADC12CTL0 |= ADC12ENC;      //Habilitar ADC12
}

// UCS: Programar relógios
// ACLK  = XT1 =>     32.768 Hz
// SMCLK = XT2 =>  4.000.000 Hz
// MCLK  = DCO => 20.000.000 Hz
// 20 MHz = 4 MHz (XT2) x 1 (FLLD) x 5 (FLLN)
void mclk_20MHz(void){

    // Configure crystal ports
    P5SEL |= BIT2 | BIT3 | BIT4 | BIT5; // Config P5 para cristais

    // This should make XT1 startup in 500ms and XT2 in less than 1ms
    UCSCTL6 = XT2DRIVE_3 |    // Turn up crystal drive to
              XT1DRIVE_3 |    // speed up oscillator startup
              XCAP_3     |    // Use maximum capacitance (12pF)
//            XT1OFF     |    // Make sure XT1 and XT2
//            XT2OFF     |    // oscillators are active
//            SMCLKOFF   |    // Leave SMCLK ON
//            XT1BYPASS  |    // Not using external clock source
//            XT2BYPASS  |    // Not using external clock source
//            XTS        |    // XT1 low-frequency mode, which
              0;              // means XCAP bits will be used

    UCSCTL0 = 0x00;           // Let FLL manage DCO and MOD

    UCSCTL1 = DCORSEL_6 |      // Select DCO range to around 20MHz
//            DISMOD    |      // Enable modulator
                        0;

    UCSCTL2 = FLLD_0 |         // (D=1) Set FLL dividers
              4;               // DCOCLK = 1 * (1+4) * FLLREF = 20 MHz

    UCSCTL3 = SELREF_5 |       // Use XT2 (4 MHz) Oscillator for FLLREF
              FLLREFDIV_0;     // divided by 1

    UCSCTL5 = DIVPA_0 |        // Output dividers to 1
              DIVA_0  |        // ACLK  divided by 1
              DIVS_0  |        // SMCLK divided by 1
              DIVM_0;          // MCLK  divided by 1

    UCSCTL7 = 0;               // Clear XT2,XT1,DCO fault flags
```

```
    UCSCTL8 = SMCLKREQEN |      // Enable conditional requests for
              MCLKREQEN  |      // SMCLK, MCLK and ACLK. In case one
              ACLKREQEN;        // fails, another takes over

    do {                        // Check if all clocks are oscillating
          UCSCTL7 &= ~( XT2OFFG  |   //Try clear XT2,XT1,DCO fault flags,
                        XT1LFOFFG |  // system fault flags and check if
                        DCOFFG );    // oscillators are still faulty
          SFRIFG1 &= ~OFIFG;         //
    } while (SFRIFG1 & OFIFG);       // Exit only when everything is ok

    UCSCTL6 &= ~(XT1DRIVE_3 |    // Xtal is now stable,
                 XT2DRIVE_3);   // reduce drive strength (to save power)

    UCSCTL4 = SELA__XT1CLK |   // ACLK  = XT1 =>     32.768 Hz
              SELS__XT2CLK |   // SMCLK = XT2 =>  4.000.000 Hz
              SELM__DCOCLK;    // MCLK  = DCO => 20.000.000 Hz
}

void config_pinos(void) {
    P2DIR &= ~BIT1;     P2REN |= BIT1;    P2OUT |= BIT1;    //S1
    P1DIR &= ~BIT1;     P1REN |= BIT1;    P1OUT |= BIT1;    //S2
    P1DIR |= BIT0;      P1OUT &= ~BIT0;                     //Led 1-VM
    P4DIR |= BIT7;      P4OUT &= ~BIT7;                     //Led 2-Verde
}

// Copiar essas funções do ER 7.8
int check_s1(void) { … }
int check_s2(void) { … }
void debounce(void){ … }
```

ER M.2. Vamos aproveitar a estrutura do ER M.1, com a diferença de que agora vamos acionar o motor com a Ponte H com o L298. O sentido de rotação e o ciclo de carga serão indicados pelo potenciômetro de 20 kΩ, da seguinte forma:

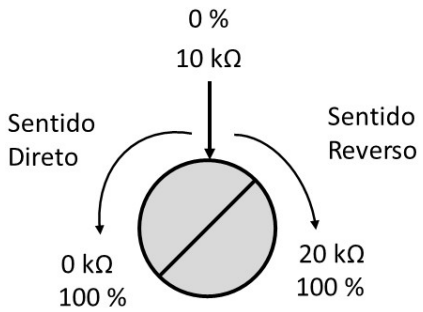

Figura M.22. Nova interpretação para o potenciômetro.

Solução: A Figura M.23 apresenta a solução para a conexão da placa com o L298 ao circuito anterior (Figura M.16). Note que houve poucas alterações. Surgiram dois pinos P4.1 e P4.2, que são usados para indicar ao L298 o sentido de rotação do motor e a saída PWM foi ligada ao *enable* (EnA).

Por isso, quase tudo o que foi feito para a solução do exercício anterior pode ser aproveitado para o exercício atual. De acordo com a Figura M.22, será necessário reinterpretar a leitura do ADC. Essa reinterpretação é simples e está apresentada na Tabela M.5.

Tabela M.5 Reinterpretação do valor entregue pelo ADC

Potenciômetro	Código ADC	Reinterpretado	PWM
0 Ω	0	2047	100% sentido direto
10 kΩ	2047	0	Parado
20 kΩ	4095	2047	100% sentido reverso

As duas linhas abaixo fazem essa reinterpretação.

```
if (adc>2047)     adc = adc-2048;          //Faixa de 2047 até 0 (direto)
else              adc = 2048-adc;          //Faixa de 0 até 2047 (reverso)
```

O programa solução é conseguido com apenas duas alterações na solução do exercício anterior. O trecho abaixo foi recortado do programa original e indica as duas áreas que deverão ser alteradas para o que está indicado na próxima listagem. É muito simples.

Listagem original

```
...
    __enable_interrupt ( );
...
        adc=soma/ADC_TOTAL;          //Média das ADC_TOTAL conversões
        pwm=1000*(adc/4095.);        //Calcula ciclo de carga
...
```

Listagem com as alterações para se conseguir o programa solução

```
...
    P4DIR |= BIT2|BIT1;              //P4.1 e P4.2 para sentido rotação
    P4OUT &= ~(BIT2|BIT1);           //P4.1=P4.2=0
    __enable_interrupt ( );
...
        adc=soma/ADC_TOTAL;          //Média das ADC_TOTAL conversões
        if (adc>2047){
            P4OUT |= BIT1;           //Sentido direto
            P4OUT &= ~BIT2;          //Sentido direto
            adc = adc-2048;          //Escala de 0 até 2047
```

```
        }
        else{
            P4OUT &= ~BIT1;          //Sentido reverso
            P4OUT |= BIT2;           //Sentido reverso
            adc = 2048-adc;          //Escala de 0 até 2047
        }
        pwm=1000*(adc/4095.);        //Calcula ciclo de carga
...
```

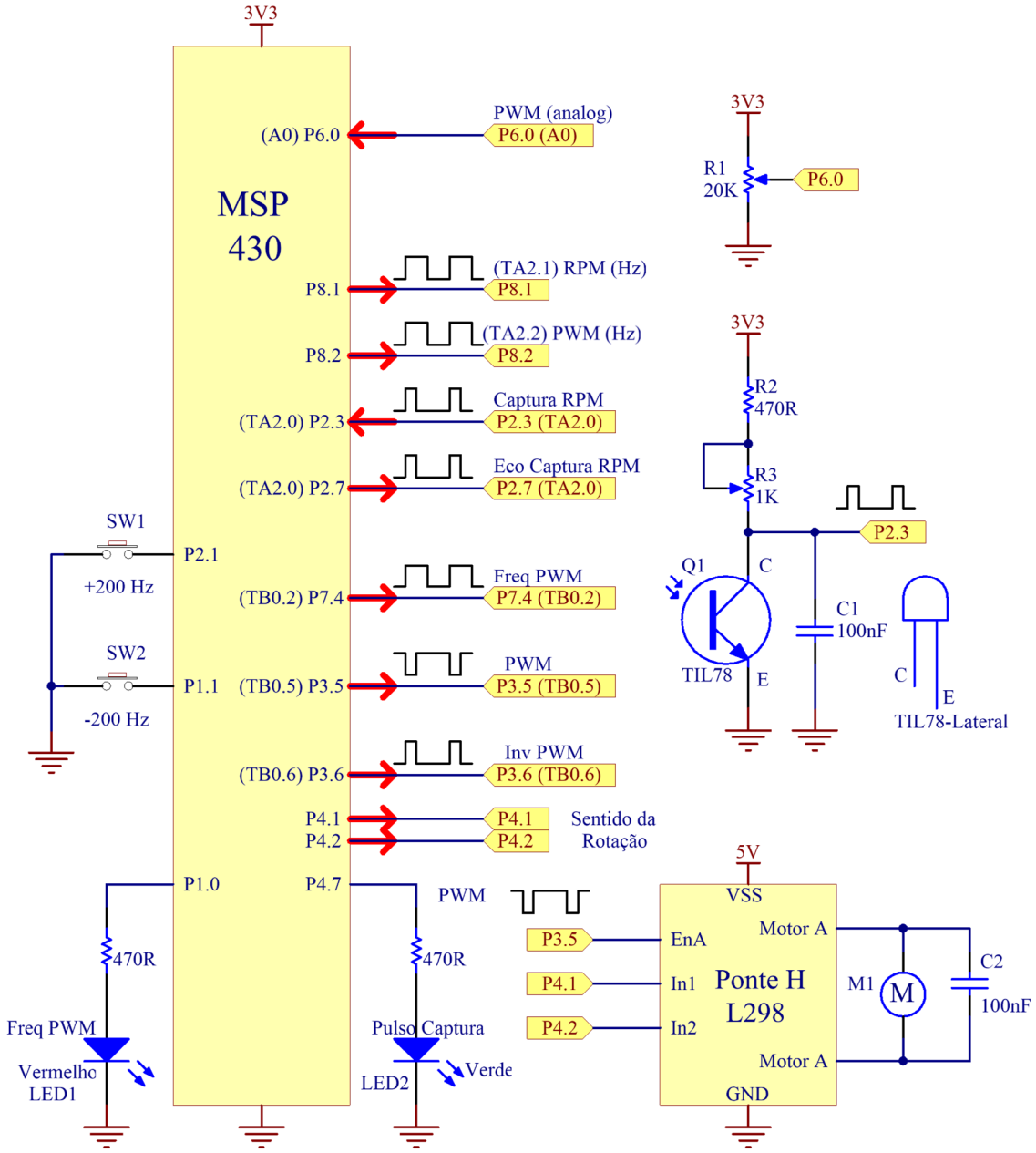

Figura M.23. Ambiente de experimentação de motor DC usando a placa com o L298.