EA075 - Material Suplementar ao Projeto Final

Segundo Semestre de 2019

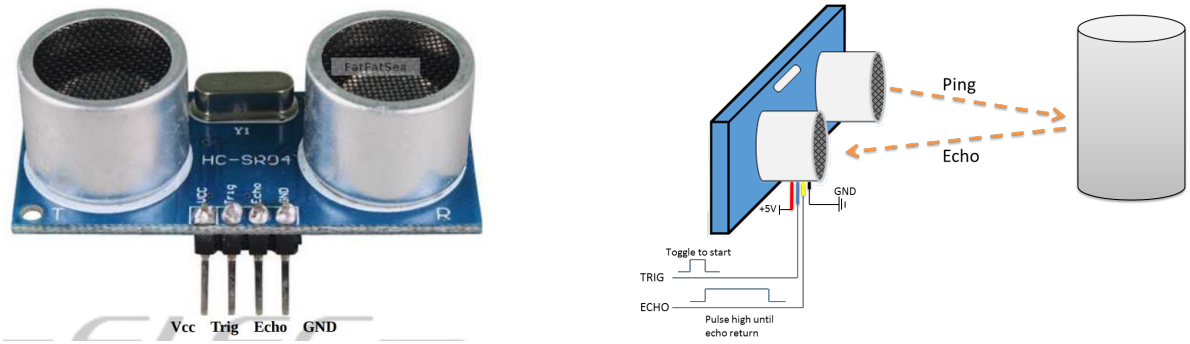
Implementação dos Sinais de Controle do Sensor Ultrassônico usando os mecanismos de interrupção do TC1 (Temporizador e Contador de 16 *bits*)

Autora: Wu Shin-Ting

Devido à sua faixa de operação e ao seu baixo custo, sensores ultrassônicos são muito aplicados hoje em dia nos projetos de sistemas embarcados para medir distâncias de superfícies (de preferência planas) em relação a um ponto de referência. O princípio de operação destes sensores, como o sensor HC-SR04 [1], é baseado na medição do tempo de eco, detectado pelo pino Echo mostrado na Figura 1.(a), de um sinal ultrassônico emitido pelo próprio sensor após um sinal de gatilho no pino Trig mostrado na Figura 1.(a). Medindo o intervalo de tempo *t* entre o instante do envio do sinal ultrassônico (Ping) e o instante do retorno deste sinal (Echo), como ilustra a Figura 1.(b), pode-se obter indiretamente a distância *d* entre o sensor e o anteparo pela equação

$$2*d = v * t \rightarrow d = (v * t) / 2,$$

onde v é a velocidade de som.



(a) Pinagem (b) Princípio de operação

Figura 1: Sensor HC-SR04 [1]

Como é explicado nas suas folhas de dados, é somente necessário aplicar um pulso curto de 10 uS no pino Trig para engatilhar o procedimento de uma medição (Figura 2). Ele começa com o envio de um trem de pulsos de 8 ciclos de sinais ultrassônicos de 40kHz e elevar o nível do sinal no pino Echo, Echo Pulse, em 1. Este sinal vai para o nível 0 somente quando o eco dos sinais ultrassônicos é detectado pelo sensor. Este instante corresponde à borda de descida do sinal Echo Pulse (Figura 2).

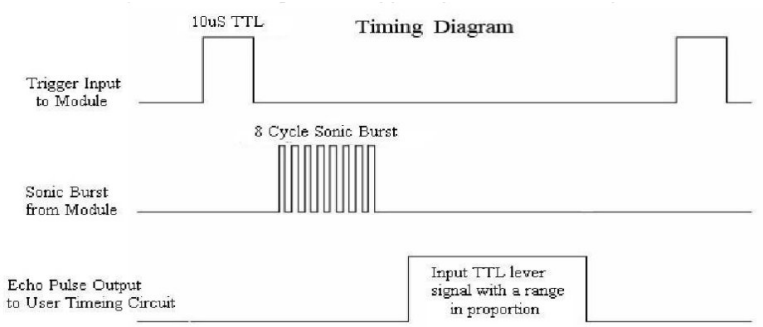


Figura 2: Diagrama de tempo do sensor HC-SR04 [1].

Intuitivamente, a solução que vem à mente é o seguinte pseudo-código:

1. Colocar o sinal digital Trig em 0.
2. Aguardar alguns microsegundos.
3. Gerar um pulso, colocando o sinal digital Trig em 1.
2. Aguardar 10us.
3. Colocar o sinal digital Trig em 0.
4. Aguardar que Echo fique em 1.
5. Iniciar a contagem de tempo até que o sinal Echo volte para 0.

De fato, é uma solução que se encontra facilmente quando fizermos uma busca pela internet, como a seguinte função [2]:

```
long Ultrasonic::Timing()
{
        digitalWrite(Trig_pin, LOW);
        delayMicroseconds(2);
        digitalWrite(Trig_pin, HIGH);
        delayMicroseconds(10);
        digitalWrite(Trig_pin, LOW);
        duration = pulseIn(Echo_pin,HIGH,Time_out);
        if ( duration == 0 ) {
        duration = Time_out; }
        return duration;
}
```

Uma implementação da função pulseIn é encontrada no site oficial de ArduinoCore-avr em [3], que utiliza a função countPulseASM implementada em assembly. Assumindo que o tempo de cada iteração seja conhecido, o programa consiste em ficar preso num laço até que o sinal Echo volte a 0 e contar a quantidade de iterações, maxloops, dentro deste laço.

```
unsigned long pulseIn(uint8_t pin, uint8_t state, unsigned long timeout)
{
        uint8_t bit = digitalPinToBitMask(pin);
        uint8_t port = digitalPinToPort(pin);
        uint8_t stateMask = (state ? Bit : 0);
        unsigned long maxloops = microsecondsToClockCycles(timeout)/16;
        unsigned long width = countPulseASM(portInputRegister(port), bit, stateMask, maxloops);
        if (width)
                return clockCyclesToMicroseconds(width * 16 + 16);
        else
                return 0;
}
```

Na apresentação do projeto da equipe 3, Pedro comentou sobre o problema que ele deparou com a "interferência" desta forma de medição no controle do seguimento de linha durante o movimento do veículo da sua equipe. Pois, o veículo pode estar fazendo uma curva e os sinais capturados pelos sensores infravermelhos serem ignorados, se o processador estiver ocupado com o processamento do sensor ultrasônico.

O que acham explorarmos os temporizadores e o mecanismo de interrupção disponíveis nos microcontroladores para tratarmos de forma mais eficiente estes eventos assíncronos? Como comentei nas aulas, além de gerar os sinais PWM, os contadores/temporizadores nos permitem criar outras funções importantes para controles de um sistema digital que envolvem tempos. Duas delas são suportadas pelo Timer1 de 16 *bits* do Atmega328 [5]. Conforme a folha técnica deste microcoontrolador (Seção 15 em [6]), este temporizador suporta 1 unidade de Input Capture (IC) e 2 unidades independentes de Output Compare (OC). O temporizador Timer0 do Atmega328 também suporta 2 unidades independentes de Output Compare (seção 14.5 em [6]). Uma unidade Input Capture de um temporizador é aquela que, quando captura um evento externo previamente configurado (borda ou nível), não só gera uma interrupção como copia o valor do contador do temporizador num registrador (Seção 15.6 em [6]). E uma unidade Output Compare é aquela em que se escreve um valor no seu registrador e quando este valor se iguala com o valor do contador do temporizador, o temporizador coloca no seu pino de saída um nível de sinal pré-configurado (Seção 15.7 em [6]). Em [7] é mostrado como se implementa o modo de operação PWM usando uma unidade Output Compare do temporizador de 8 *bits* Timer0 do ATmega328. Em [8] pode-se encontrar uma síntese sobre configurações do Timer1 para operar em IC e OC.

Aplicando IC e OC, podemos delegar para os temporizadores a tarefa de "contar" os tempos sem ocupar o processador e avisar o processador somente quando acontecer algum evento relevante, como a borda de

descida e de subida do sinal Echo. O seguinte pseudocódigo demonstra a ideia, considerando que o pino Echo do sensor esteja conectado ao pino ICP1 do Timer 1 (pino 8 do Arduino Uno) e o pino Trig num pino Ocx de um Timery:

Fluxo principal:

Setup:

1. Configurar a unidade IC do Timer1 para capturar as bordas de subida e de descida.

2. Configurar a unidade OC de um temporizador para setar em 1 o seu sinal de saída quando ocorre igualdade na comparação.

2. Inicializar o estado de uma nova medição (flag=0).

Laço principal:

1. Se flag == 0 (estado de nova medição), então

    1.a ler o valor do contador.

    1.b escrever no registrador do OC a soma deste valor e do equivalente a 10us.

    1.c setar em 1 o pino de saída da unidade OC

    1.d habilitar a interrupção da unidade OC.

    1.e flag = 1 (estado de medição).

2. Se flag == 2 (estado de medição completa), então

    2.a computar a distância a partir de tempo_inicial, tempo_final e qtde_overflows.

    2.b flag = 0 (estado de nova medição).

3. Fazer os outros processamentos, como o de monitoramento da linha guiadora.

Rotina de Serviço TIMER1_COMPA (transcorridos 10us configurados)

1. Resetar a quantidade de overflows do contador do Timer1 (qtde_overflows = 0).

2. Habilitar a interrupção de IC do Timer1.

3. Desabilitar a interrupção de OC.

Rotina de Serviço TIMER1_OVF

1. Incrementar a quantidade de overflows (qtde_overflows++).

Rotina de Serviço TIMER1_CAPT (quando ocorre a borda de descido/subida do Echo)

1. Se for a borda de subida, então tempo_inicial = valor do contador do Timer1.

2. Se for a borda de descida, então

    2.a tempo_final = valor do contador do Timer1.

    2.b flag = 2 (estado de medição completa).

3. Desabilitar a interrupção do IC do Timer1.

Note que, ao usar o mecanismo de interrupção, o processador não se ocupará mais com as contagens propriamente ditas, a de 10us do gatilho e a da largura do pulso Echo. Esta será feita pelos contadores em circuitos independentes embutidos nos temporizadores. O processador só executará as instruções das rotinas

de serviço, as de gravar os valores dos contadores do Timer1 nos momentos apropriados, e ficará mais livre para processar outors sinais capturados por outros sensores.

Referências:

[1] Elec Freaks. Ultrasonic Ranging Module HC – SR04. https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
[2] Github. Ultrasonic-HC-SR04. https://github.com/JRodrigoTech/Ultrasonic-HC-SR04/blob/master/Ultrasonic/Ultrasonic.cpp
[3] wiring_pulse.c. https://github.com/arduino/ArduinoCore-avr/blob/master/cores/arduino/wiring_pulse.c
[4] pulse_asm.S. https://github.com/arduino/ArduinoCore-samd/blob/master/cores/arduino/pulse_asm.S
[5] theenggproject. Introduction to Atmega328: A complete step by step tutorial on Introduction to Atmega328. https://www.theengineeringprojects.com/2017/08/introduction-to-atmega328.html
[6] Atmega328P Datasheet. ftp://ftp.dca.fee.unicamp.br/pub/docs/ea075/datasheet/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf
[7] Arnab Kumar Das. AVR Timer-0/Counter-0 Tutorial – Atmega328p – AVR – 8 bit – Arduino Uno. **https://www.arnabkumardas.com/online-courses/avr-timer-counter-programming-tutorial-atmega328p-avr-8-bit-arduino-uno/**
[8] Fábio Souza. Timers do ATmega328 no Arduino. **https://www.embarcados.com.br/timers-do-atmega328-no-arduino/**