Mateus Cappellari Vieira
Prícilla Karen Suzano
Rita Alamino
Thales de Oliveira
Thayllor Peres Devos dos Santos
Wanderson de Oliveira Paes

# Ecobatímetro – Desenvolvimento e Simulação de um Sistema Digital

Brasil

2019

Mateus Cappellari Vieira
Prícilla Karen Suzano
Rita Alamino
Thales de Oliveira
Thayllor Peres Devos dos Santos
Wanderson de Oliveira Paes

# Ecobatímetro – Desenvolvimento e Simulação de um Sistema Digital

Trabalho apresentado à disciplina de Sistemas Digitais do curso de Engenharia de Computação, constituindo parte integrante da avaliação do terceiro bimestre de 2019.

Universidade Federal do Rio Grande – FURG
Centro de Ciências Computacionais – C3
Engenharia de Computação

Sistemas Digitais – Turma U

Prof. Vitor Irigon Gervini

Brasil
2019

# Sumário

# 1 Introdução

Ao longo da história, diversas soluções foram propostas para medir a profundidade de corpos d'água, desde imensos medidores estáticos fisicamente implantados no solo aquático em posições estratégicas até a utilização de sensores ultrassônicos a fim de realizar a medição em diversos pontos, inclusive, em movimento. Para esse propósito, um ecobatímetro consiste em um dispositivo sonar, amplamente utilizado, que usa a diferença temporal entre a emissão de um pulso sonoro e a recepção do mesmo após a reflexão pelo fundo do leito de água para realizar o cálculo de profundidade.

Em seu funcionamento básico, um ecobatímetro usa a transmissão de ondas sonoras de alta frequência (15-200 kHz). Fisicamente falando, quando um pulso ultrassônico é gerado em uma direção específica, se houver um objeto no caminho desse pulso, parte ou todo o pulso será refletido de volta ao transmissor como um eco e poderá ser detectado através do caminho do receptor. Para o propósito desta aplicação, desprezamos qualquer angulação ou distorção produzida pelo movimento do ecobatímetro durante a emissão e recepção da onda.

No objeto deste trabalho, propomos uma solução na componente digital de um circuito elétrico que simula o funcionamento de um ecobatímetro que analisa profundidades de água de no máximo 50 metros, utilizando programação em baixo nível em um dispositivo microcontrolador acoplado a um circuito integrado 555, por uso de ferramentas como interrupções de sistema e armazenamento em registradores a fim de obter valores de tempo e distância a partir de pulsos elétricos.

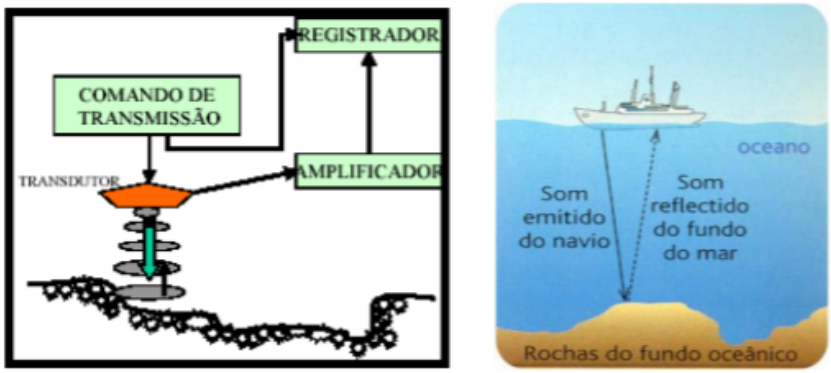


Figura 1 – Princípios de um ecobatímetro.

# 2 Objetivos

Este trabalho visa desenvolver um simulador da componente digital de um ecobatímetro utilizando principalmente o microcontrolador ATmega328/P, parte integrante da placa Arduino UNO, juntamente a um circuito integrado 555, na ferramenta de simulação de circuitos e sistemas elétricos SimulIDE. Isto foi pensado a fim de compreendermos o desenvolvimento de aplicações reais utilizando microcontroladores.

Nem todo o circuito do ecobatímetro foi desenvolvido, sendo que os componentes analógicos foram julgados como fora do pretexto deste trabalho. A proposta desse trabalho é que seja feito o desenvolvimento da parte digital do sistema com o uso de contadores intrínsecos ao microcontrolador em conjunto com a utilização de interrupções, para que a simulação seja feita de forma confiável em baixo nível de abstração em relação ao hardware.

# 3 Metodologia

Nesta seção serão apresentados os equipamentos e procedimentos utilizados para alcançar os resultados obtidos neste projeto.

## 3.1 Programas e materiais utilizados

Para a simulação digital do ecobatímetro, foi utilizado o software de simulação SimulIDE, que permite a construção e manipulação de sistemas elétricos e eletrônicos. No simulador, utilizamos o microcontrolador ATmega328/P e o circuito integrado 555, interligados em seus pinos via fiação. O software consegue representar o sistema com aproximadamente 80% de sua velocidade real. O código foi feito em C++, prezando o baixo nível.

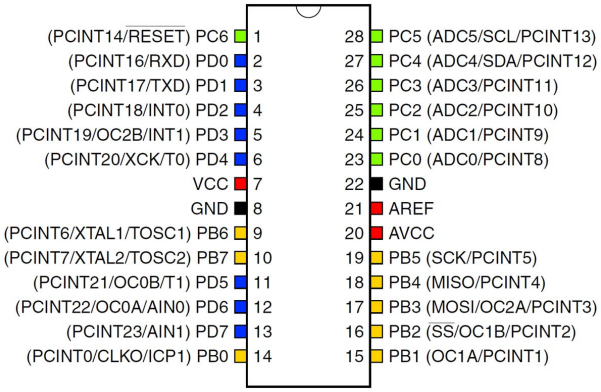


Figura 2 – Esquema *pinout* do ATmega328/P utilizado.

As especificações técnicas relativas ao funcionamento do microcontrolador são oriundas do *datasheet* original do microcontrolador ATmega328/P, disponibilizado pela Atmel. Tais especificações consistem, por exemplo, no conteúdo dos registradores presentes no microcontrolador, no mapeamento dos pinos disponíveis, entre outras informações técnicas relacionadas ao ATmega328/P.

## 3.2 Métodos utilizados

Utilizamos dois dos contadores internos do microcontrolador ATmega328/P, TIMER1 e TIMER2 (TCNT1 e TCNT2, respectivamente), para obtermos controle sobre o sistema. A onda de emissão que simula a onda real analógica que seria emitida pelo ecobatímetro é uma onda quadrada PWM de 40kHz, gerada pela constante inversão de bits no pino de saída, regulada em relação ao período da mesma.

O PWM (Pulse Width Modulation ou Modulação de Largura de Pulso) refere-se ao conceito de pulsar rapidamente um sinal digital em um condutor. É a técnica usada para gerar sinais analógicos de um dispositivo digital como um microcontrolador. Pode ser visto na figura 3 exemplos de ondas PWM.

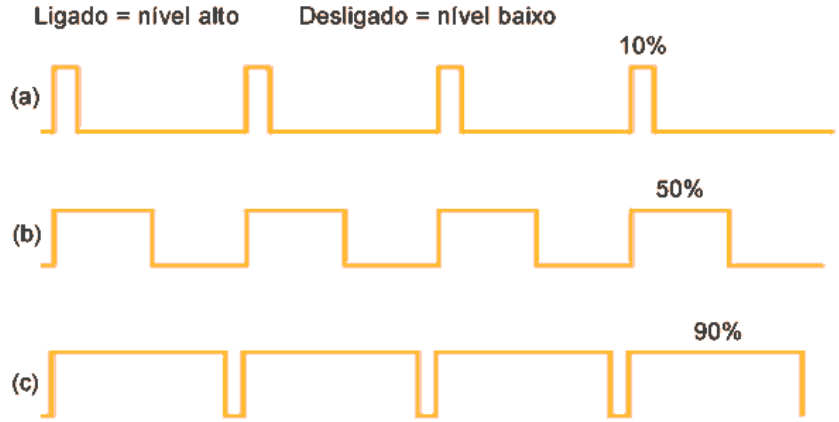


Figura 3 – Exemplos de ondas PWM.

O controle de fluxo do código foi construído através de interrupções, disparadas de imediato quando ocorre algum evento ativador destas. Utilizamos tanto interrupções nos pinos quanto as interrupções intrínsecas aos timers do microcontrolador. Tais interrupções consistem em mecanismos que "quebram" o processamento atualmente em execução no ATmega328/P, a fim de executar uma determinada sequência de comandos.

Por fim, é interessante ressaltar a utilização do CI555 para a finalidade de simular a distância que estaria presente entre o ecobatímetro e o leito do corpo d'água. Este circuito integrado tem um "delay"interno construído eletronicamente, regulável através da resistência e capacitância a ele atreladas.

## 3.3 Macros

No código da aplicação, utilizamos algumas *macros*, isto é, definições de constantes que, quando chamadas, apontam para uma nova chamada, ou uma operação de baixo nível nos registradores do ATmega328/P. Essas operações geralmente consistem em manipulações lógicas ou algébricas do(s) bit(s) presente(s) no registrador desejado. As macros na figura

4 consistem nas que são chamadas durante a execução do código, acompanhadas de uma breve explicação de seu funcionamento.

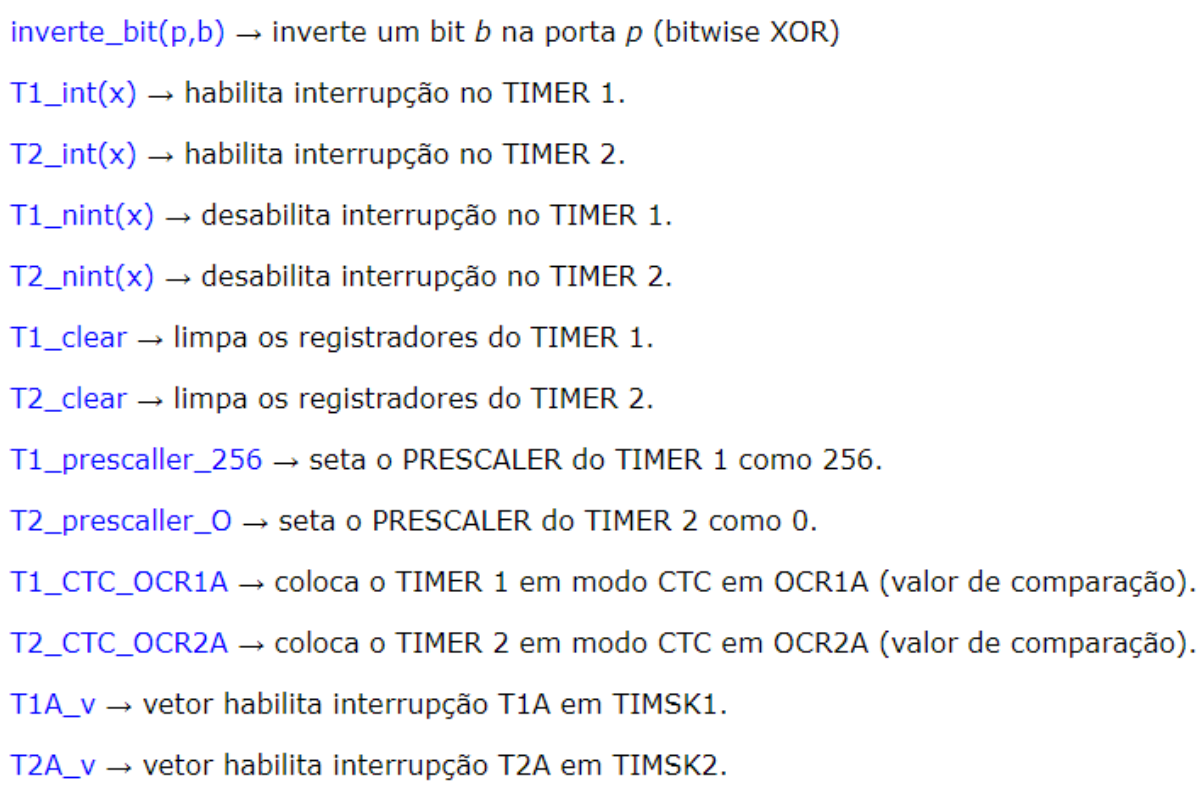


Figura 4 – Macros utilizadas ativamente no código.

Existem algumas outras definições de constantes ativamente utilizadas no arquivo do código, sendo elas uma constante "SPEED" com o valor fixo em 1500, determinando a velocidade do som no meio aquoso, e duas outras, T1A e T2A, setadas respectivamente em 1 e 2, que consistem nos valores de habilitação para as interrupções em ambos os TIMERs.

# 4 Resultados

Desenvolvemos o código e circuito que constroem um cenário de simulação, no qual temos a parte digital de um ecobatímetro capaz de medir profundidades de até 50 metros. Para este cenário, escolhemos uma profundidade exemplo de 20 metros. Com a escolha desse valor, foi então desenvolvida uma onda esperada de 40 kHz para obtermos a profundidade de 20 metros como resposta.

A profundidade do leito d'água foi calculada pela fórmula abaixo, derivada da equação do deslocamento retilíneo uniforme:

$$P = V_s \cdot \frac{(t_{TIMER1} + 1)}{2} \cdot \frac{256}{16000000}, \tag{4.1}$$

na qual $P$ denota a profundidade, $Vs$ é a velocidade do som no meio aquoso, no valor de 1500 m/s, $t_{TIMER1}$ é o valor final no contador TIMER1, que nada mais é que o tempo de viagem da onda sob a ação do prescaler, e o fator mais à direita é o inverso do prescaler aplicado ao TIMER1, aplicado a fim de obtermos um valor temporal em segundos.

## 4.1 Circuito

Para descrever o circuito, primeiramente precisamos encontrar o valor do resistor e capacitor acoplados ao CI555, necessários para a contagem do tempo de viagem da onda. Para isso iremos manter a capacitância no valor de 100 µF e encontrar o valor da resistência. Como foi escolhido o valor de 20 metros de profundidade, calculamos primeiramente o tempo que uma onda a 1500 m/s, a velocidade aproximada do som na água, percorre os 20 metros. O valor encontrado é de aproximadamente 0,013 segundos. Como esse tempo é para percorrer 20 metros e sabemos que no tempo de viagem a onda percorre 40 metros, 20 para ir e 20 para voltar, é necessário multiplicar 0,013 por dois para substituirmos na fórmula e encontrarmos o valor da resistência.

A equação utilizada para a calibração da resistência do CI555 é a disposta abaixo:

$$R = \frac{tempo_{viagem}}{C \cdot ln(3)} \tag{4.2}$$

$$R = 242,42\,\Omega \tag{4.3}$$

Após a aplicação dessa fórmula, obtemos para a resistência o valor de 242,42 Ω, não variando a capacitância. Desta forma, o circuito integrado 555 simulará com eficácia o tempo em que a onda estaria viajando da origem até a profundidade do leito.

Com esse componente pronto, fazemos sua conexão com o ATmega328/P. O pino B5 foi escolhido como o pino de saída do microcontrolador que realiza o *trigger* do CI555, enviando a onda PWM para o mesmo. A saída de pulso do CI555, que determina o final da simulação de distância, foi conectada ao pino D2 do ATmega328/P.

O microcontrolador em si tem seu sinal alto gerado por uma fonte ativa de 5V conectada a um resistor de 10 kΩ, que, juntamente com um botão de *switch* que abre/fecha circuito com o *ground*, está conectada no pino D3. Dessa forma, quando o circuito é fechado, o microcontrolador recebe uma rápida entrada em nível baixo, ativando a primeira interrupção, e assim sucessivamente.

A figura 5 demonstra o circuito em seu estado inerte, com a fonte que gera o sinal de controle desligada.

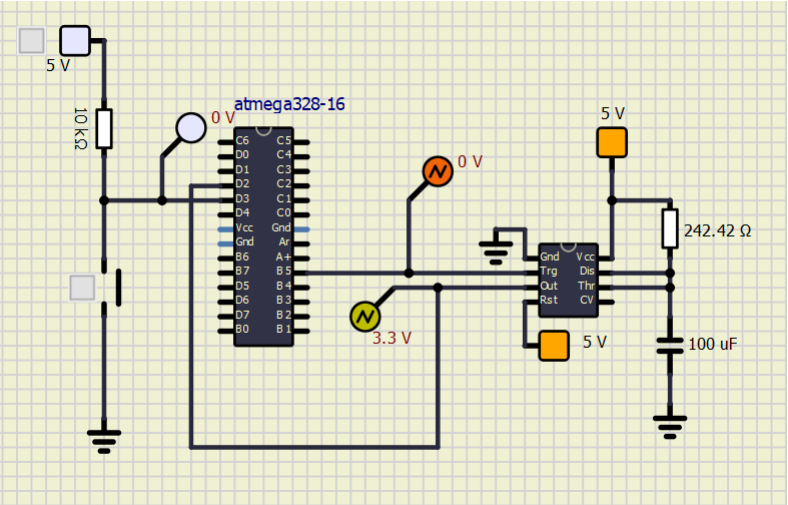


Figura 5 – Circuito inerte. Note a disposição do microcontrolador e do CI555.

Quando a fonte é ligada, o pino D3 recebe um sinal alto, como mostra a figura 6.

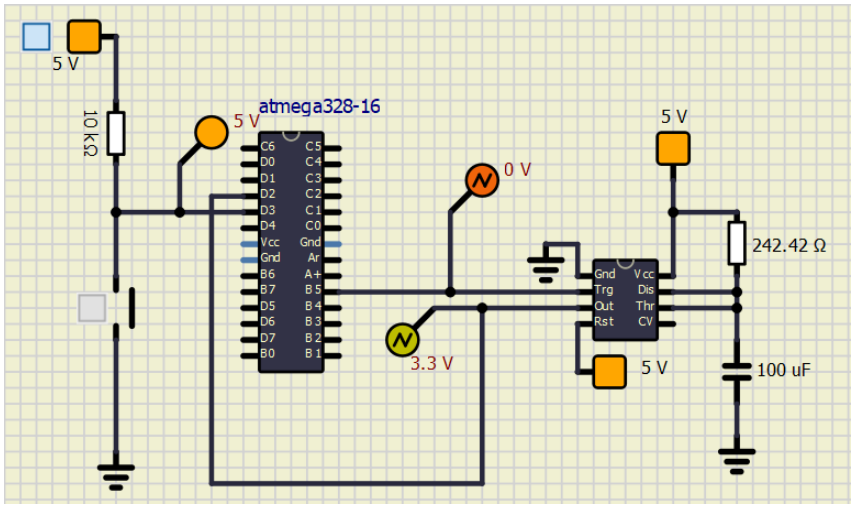


Figura 6 – Circuito ligado. ATmega328/P recebe o sinal alto.

Enfim, quando ocorre o pressionamento do botão que fecha o circuito com o *ground*, um sinal baixo é recebido no pino D3 e a cadeia de interrupções é acionada. A figura 7 explicita o circuito neste estado.

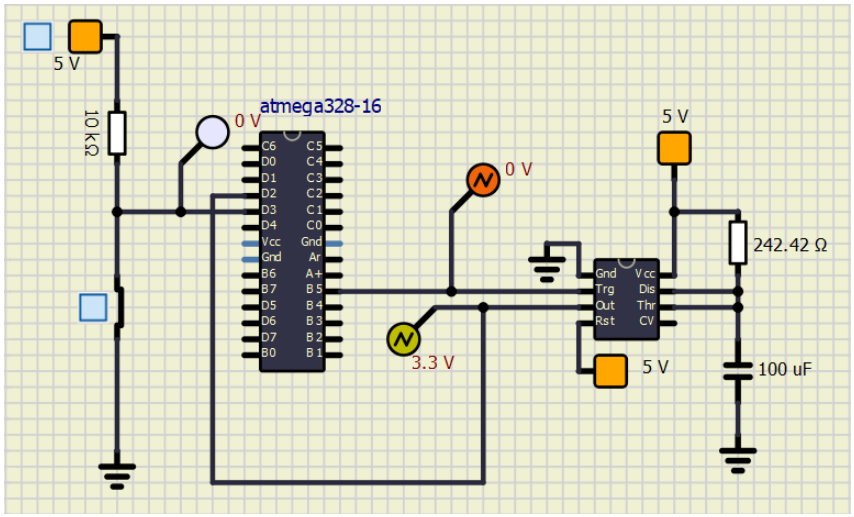


Figura 7 – Circuito fechado. ATmega328/P recebe o sinal baixo, iniciam-se as interrupções.

Por fim, após a sequência lógica do código ser finalizada, o sistema utilizará a porta serial aberta em software para mostrar o resultado final da profundidade calculada para cada vez que o botão foi pressionado.

## 4.2 Código

Nesta seção será apresentado o código escrito para realizar a simulação do ecobatímetro e a explicação de cada função apresentada. O código foi desenvolvido em C++, linguagem padrão para aplicações em Arduino e, por consequência, no ATmega328/P.

Algumas definições importantes para a compreensão:

- PRESCALER
  É um divisor de frequência que divide o clock de cada timer (16 MHz) para uma frequência específica desejada, por exemplo: Se o prescaler de um timer for setado em 8, a operação deste timer será em 16MHz/8.
- MODO CTC_OCR*A
  Um timer neste modo "Clear on Compare" conta até um valor armazenado em um determinado registrador (OCR*A), e, ao chegar neste valor, zeramos o TIMER e inicializamos uma interrupção.
- EICRA e EIMSK
  São dois registradores que controlam interrupções externas.
- SREG
  Registrador que guarda a flag de interrupção global.

O código está dividido em funções, dentre funções básicas de *setup* e *loop*, funções auxiliares e as interrupções, que ocorrem no estouro de TIMER1 e TIMER2 e nos pinos D2 e D3, mapeados internamente como INT0 e INT1, respectivamente. As subseções abaixo apresentam o código de maneira modular, respeitando a ordem cronológica dos eventos de simulação na medida do possível.

### 4.2.1 Função setup

```
void setup(){
  Serial.begin(9600);
  // configura o timer 1 para contar ate o tempo de retorno maximo
  T1_clear;                 // limpa registradores de configuracao do TIMER1
  T1_CTC_OCR1A;             // modo: CTC - overflow em OCR1A
  T1_prescaller_256;        // prescaller TIMER1 256
  OCR1A = 4124;             // (tempo_s*clock)/prescaller-1 = (66ms*16Mhz)/(256)-1 = 4124

  // configura o timer 2
  T2_clear;
  T2_prescaller_O;
  T2_CTC_OCR2A;
  OCR2A = 199; // 12.5 microsegundos (periodo/2 de 40 kHz)


  EICRA = 1<<ISC01|1<<ISC11; // Seta o modo de interrupcao pra quando tiver uma descida.
  EIMSK = 1<<INT0|1<<INT1;   // habilitacao para a chamada da interrupcao em INT0 e INT1.
}
```

Figura 8 – Função setup.

A função setup tem como objetivo preparar os contadores e habilitar o uso das interrupções para que a lógica desenvolvida flua normalmente. Essa função é a primeira

a ser executada pelo microcontrolador após ele ser ligado com o *firmware* deste código carregado, e executa apenas uma vez.

A itemização a seguir detalha cada um dos comandos utilizados na função, com exceção da primeira linha, que abre a porta 9600 para comunicação serial.

- T1_clear
  Limpa as configurações do TIMER1, deixando-o em modo *default*.
- T1_CTC_OCR1A
  Coloca o TIMER1 em modo CTC_OCR1A, para que resete quando sua conta chegar no valor igual ao que esteja armazenado em OCR1A.
- T1_prescaller_256
  Seta o prescaller do TIMER1 para 256, ou seja, ele operará em 16 MHz/256.
- OCR1A = 4124
  Guarda o valor correspondente ao tempo que a onda de 40 kHz demoraria para percorrer 50 metros (máximo suportado pelo sistema).
- T2_clear
  Limpa o TIMER2, analogamente ao que ocorreu com o TIMER1.
- T2_prescaller_0
  Seta o prescaller do TIMER2 para 0 (timer com o clock de 16MHz)
- OCR2A = 199
  Seta o valor de estouro para o TIMER2, que ativará a interrupção de OCR2A match toda vez que seu valor chegar a 199. Este valor foi obtido multiplicando o clock de 16MHz do TIMER2 por metade do período da onda de 40kHz, 12.5 microssegundos, e diminuindo uma unidade.

Em seguida, são configuradas as interrupções em dois registradores de controle de interrupções externas:

- EICRA = 1<<ISC01 | 1<<ISC11
  Seta o modo de interrupção externa INT0 e INT1 (pinos D2 e D3, respectivamente) para quando houver uma borda de descida. De extrema importância para o controle do sistema, pois a inicialização do cálculo ocorre com a passagem de um sinal baixo.
- EIMSK = 1<<INT0 | 1<<INT1
  Habilitação para a chamada da interrupção em INT0 e INT1.

### 4.2.2 Função loop

```
void loop(){
// Constantemente verifica se está na hora de printar o resultado
 if (toprint){
    printDistance();
    toprint = false;
 }
}
```

Figura 9 – Função loop.

Esta é a função main que roda em loop após o setup. O ATmega328/P faz esse teste continuamente para verificar se está na hora de mostrar os resultados.

Se a variável de controle “toprint” for verdadeira:

- Mostra a distância através da porta serial aberta;
- Coloca “toprint” em falso.

As interrupções são responsáveis por quebrar o laço de repetição "infinito" causado por essa função.

### 4.2.3 Interrupção pino D3

```
ISR(INT1_vect){ // pino D3
 if (!enable) {
   enable = true; // Há uma execução em andamento
   reset();
   T2_int(T2A); // Chama a interrupção do TIMER 2
 }
}
```

Figura 10 – Interrupção pino D3.

No momento em que o botão é pressionado, um sinal baixo é enviado para o ATmega328/P. Essa borda de descida é percebida e a interrupção do pino de entrada D3 é ativada.

Essa interrupção utiliza-se de um teste com a variável "enable", inicializada como falsa, para saber se há ou não um cálculo em andamento agora. Essa variável funciona como uma trava, que não permite que uma cadeia de interrupções seja iniciada se já há uma simulação em andamento.

O funcionamento, então, é basicamente o seguinte – se o pino está em baixo:

- Seta o enable;

- Aplica a função reset, que reseta os TIMERs do sistema e desabilita as duas interrupções dos mesmos;
- Habilita a interrupção no contador 2.

### 4.2.4 Interrupção OCR2A

```
ISR(T2A_v){
  // Interrupção OCR2A match faz o toggle da porta B5.
  // Para gerar a onda de 40 kHz
  // Toggle feito em 12.5 microsegundos
  times ++; // Soma cada inversão feita
  if (times <= 111){ // 111 -> Número de inversões máximas (40kHz) (t_init/(T/2)).
    inverte_bit(PORTB,5);
    return;
  } else {
     inverte_bit(PORTB,5);
     T2_nint(T2A); // Desativa a interrupção do TIMER 2.
     T1_int(T1A); // Ativa a interrupção do TIMER 1.
     TCNT1 = 84; // Correção de precisão: 84 = (1.344 ms * 16 MHz/256)
  }
}
```

Figura 11 – Interrupção OCR2A.

Essa é a interrupção do TIMER2, responsável pela geração do PWM. É executada toda vez que o TIMER2 chega em um valor igual ao que está armazenado no registrador OCR2A. A cada 12.5 microssegundos, que é igual a metade do período da onda, inverte-se o valor na porta B5, a fim de gerar uma onda quadrada. A variável "times" conta quantas vezes foi feita a inversão (após 111 vezes se passou um tempo de emissão igual a 1.344 ms).

Se times <111

Então

Inverte o bit

Senão

Inverte o bit

T2_nint(T2A) -> Desabilita a interupção do TIMER2

T1_int(T1A) -> Habilita a interupção do TIMER1

TCNT1 = 84 -> Inicia o TIMER1 com tempo de transmissão: 1.344 ms * 16MHz/256 (tempo suficiente para medir aproximadamente 1m)

A partir deste ponto, ou seja, com a onda gerada e transmitida ao CI555, existem duas possibilidades distintas:

1. O tempo de espera no CI555 ditou que o chão está a uma distância de 50 m ou mais:
   Então, será inicializada a interrupção do TIMER1 (pois ocorre OCR1A match).

2. O tempo de espera no CI555 ditou que o leito está a uma distância de 49.99 m ou menos:
   Nesse caso, será executada a interrupção no pino D2, para que o cálculo da distância seja feito e apresentado.

### 4.2.5 Interrupção OCR1A

```
ISR(T1A_v) {
  // interrupcao OCR1A match
  // ocorre na distancia maxima "overflow" do timer 1.
  // 50m
  dist = 50;
  reset();
  toprint = true;
  enable = false;
}
```

Figura 12 – Interrupção OCR1A.

Essa interrupção relativa ao TIMER1 ocorre assim que este contador chega ao valor presente em OCR1A. Isso significa que a simulação do tempo no 555 acusa uma distância no ou acima do valor máximo de 50 metros. Os comandos presentes na função estão explanados a seguir:

- dist = 50
  Sendo a distância maior ou igual a 50 m, forçamos a mesma a ser 50 m, pois qualquer valor acima disto extrapola os limites de nosso sistema.

- reset()
  Resetamos o valor dois TIMERs e desabilitamos suas interrupções.

- toprint = true
  Deixando a variável "toprint"com conteúdo verdadeiro, habilitamos o print da distância na função loop.

- enable = false
  Permite um novo cálculo de distância quando o botão for novamente pressionado.

### 4.2.6 Interrupção pino D2

```
ISR(INT0_vect){ // pino D2
  // Calcula a distância
  unsigned int i= TIM16_ReadTCNT1(); // Pega tempo final do TIMER1
  time = (((double)(i+1)*256)/16000000.0)/2; // Desfaz o prescaler
  dist = time*(double)SPEED;
  reset(); // Reseta os timers
  toprint = true;
  enable = false;
}
```

Figura 13 – Interrupção pino D2.

A interrupção no pino D2 (INT0) é ativada quando este pino recebe um pulso vindouro da saída do CI555, indicando que o tempo calibrado no componente já passou, e o cálculo da distância pode em fim iniciar. Nesta interrupção, a profundidade (distância) é calculada a partir da equação 4.1, utilizando uma leitura do conteúdo do TIMER1, que nesse ponto nada mais será que o tempo total de viagem da onda (ida e volta), sob efeito do prescaler (256). Isto é resultado uma soma do tempo de emissão com o qual o contador foi inicializado com o valor contado enquanto o 555 tomava seu tempo.

Cada comando desta interrupção faz o descrito abaixo:

- unsigned int i = TIM16_ReadTCNT1()
  Armazena o valor final do TIMER1 na variável i, através da chamada da função TIM16_ReadTCNT1().

- time = (((double)(i+1)*256)/16000000.0)/2
  Inicia a aplicação da equação 4.1, calculando o tempo que a onda demorou para chegar no fundo, dividindo por dois e já aplicando o inverso do prescaler para que o tempo fique em segundos.

- dist = time*(double)SPEED
  Calcula a distância até o fundo, continuando a equação 4.1.

- reset()
  Reseta os valores dos TIMERS e desativa suas interrupções.

- toprint = true
  Permite o print da distância pela função loop.

- enable = false
  Permite um novo cálculo de distância quando o botão for novamente pressionado.

### 4.2.7 Função TIM16_ReadTCNT1

```
unsigned int TIM16_ReadTCNT1( void )
{
  unsigned char sreg;
  unsigned int i;
  sreg = SREG;
  // Desativa interrupções
  cli();
  i = TCNT1;
  SREG = sreg;
  return i;
}
```

Figura 14 – Função TIM16_ReadTCNT1.

Essa função auxiliar serve para realizar a leitura do conteúdo do contador TIMER1. Nela, primeiramente salvamos o vetor global de interrupção em uma variável auxiliar, depois desativamos as interrupções com a função cli(). Feito isso, salvamos o valor do TIMER1 na variável i. Voltamos o vetor de interrupções para seu estado antes da desativação das interrupções. Por fim retornamos o valor da variável i.

### 4.2.8 Função printDistance

```
void printDistance( void )
{
  unsigned char sreg;
  double i;
  sreg = SREG;
  cli();
  Serial.println(dist, "%.6f");
  SREG = sreg;
}
```

Figura 15 – Função printDistance.

Essa função de auxílio mostra o resultado no monitor serial, e é chamada para este fim pela função loop, quando for o momento de fazê-lo, ou seja, quando o resultado da profundidade estiver pronto para exibição.

Nesta função, é feito um processo análogo ao feito na função TIM16_ReadTCNT1, salvando o vetor de interrupção global e as interrupções, a fim de mostrar o resultado da profundidade, armazenado na variável "dist", sem o comando ser interrompido. O comando Serial.println(dist, "%ḣf") escreve o valor na tela, e então a flag global de interrupção antes salva é armazenada novamente no registrador SREG.

## 4.3 Resultado final

Tendo o circuito montado e código finalizado, podemos visualizar através do monitor serial, a partir da segunda linha, o valor de aproximadamente 20 metros que era esperado, como descrito na figura 16.

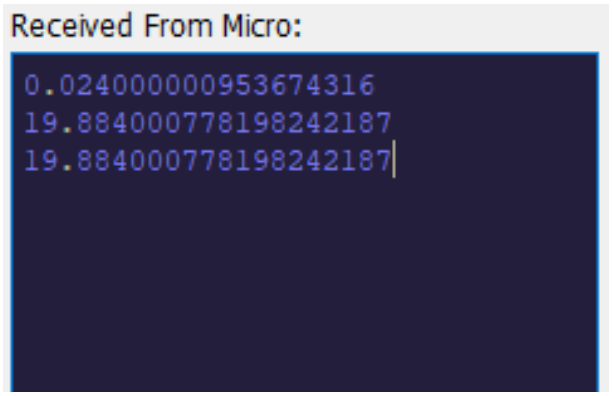


Figura 16 – Resultados da medição de profundidade sendo apresentados no monitor serial.

A primeira profundidade apresentada nessa figura é menor que as outras pois o trigger do CI555 é acionado imediatamente na ativação do circuito, ou seja, antes mesmo da função de setup. Dessa maneira, a partir da segunda vez que o botão é pressionado, a configuração e o cálculo seguem normalmente, e os resultados são factíveis.

# 5 Conclusão

O dispositivo ecobatímetro é a solução mais utilizada para medir profundidade, e seu valor de mercado não é barato. Contudo, utilizando ferramentas como Arduino (e portanto o ATmega328/P), que é de código aberto, podemos economizar significativamente no valor final. Com ferramentas como SimulIDE, temos a possibilidade de criar e simular um circuito, contribuindo muito, inclusive, na aprendizagem da matéria da disciplina de Sistemas Digitais.

O desenvolvimento do circuito que cria um cenário de simulação de um ecobatímetro, utilizando o ATmega328/P e o Circuito Integrado 555 auxiliou no entendimento do funcionamento das interrupções em sistemas digitais, além de computacionais como um todo, reforçando também conteúdos de circuitos resistivo-capacitivos.

# Referências

Atmel. *ATmega328/P DATASHEET COMPLETE*, 2016.

Honeywell ELAC. An Introduction to Echo Sounding. Technical report, Germany, 1982.

Gilberto Gagg. Levantamentos Hidrográficos: Noções Gerais. Technical report, UFRGS – UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, 2016.

Sabuj Das Gupta, Islam Md. Shahinur, Akond Anisul Haque, Amim Ruhul, and Sudip Majumder. Design and Implementation of Water Depth Measurement and Object Detection Model Using Ultrasonic Signal System. *International Journal of Engineering Research and Development*, 4, 2012.

Santiago González. SimulIDE <https://simulide.blogspot.com/>.

Ítalo Oliveira Ferreira. Coleta, Processamento e Análise de Dados Batimétricos Visando a Representação Computacional do Relevo Submerso Utilizando Interpoladores Determinísticos e Probabilísticos. Technical report, Universidade Federal de Viçosa, 2013.