"… um profissional de qualquer campo da computação não deveria considerar o computador como uma simples caixa preta que executa programas por mágica..." (IEEE/ACM Computer Science Curriculum, 2008)

Timers

Paulo Ricardo Lisboa de Almeida

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

Configurando clock interno

- Seu ATMega328p-pu provavelmente veio configurado para operar com um clock interno de 1MHz
- Submeta o comando no avrdude para modificar o clock para 8MHz
  - avrdude -b 9600 -p m328p -c usbasp -e -U lfuse:w:0xe2:m -U hfuse:w:0xd9:m -U efuse:w:0xff:m
- Para voltar a operar em 1MHz
  - avrdude -b 9600 -p m328p -c usbasp -e -U lfuse:w:0x62:m -U hfuse:w:0xd9:m -U efuse:w:0xff:m
- Os comandos são um oferecimento de **Lucas Litter Mentz**
  - Detalhes na Seção 9.2.1 - *Default Clock Source* do manual
  - www.engbedded.com/fusecalc

Gerando delays

- Muitas vezes precisamos sincronizar as coisas via delays
- Aguardar um evento, enviar um sinal por determinado período para garantir que o periférico capturou o sinal, …
- Como gerar esses delays?

Gerando delays

- Opção 1
  - Podemos criar um loop de espera ocupada
    - Fizemos isso com nossas rotinas de *delay_omic* e *delay_debounce*
- Vantagens? Desvantagens?

Gerando delays

- Opção 1
  - Podemos criar um loop de espera ocupada
    - Fizemos isso com nossas rotinas de delay_omic e delay_debounce

- Vantagens? Desvantagens?
  + Precisão: montamos os delays, e sabemos exatamente quantos ciclos de clock cada instrução precisa para ser executada (veja no manual)
  + Simplicidade: é relativamente simples criar esses loops

Gerando delays

- Opção 1
  - Podemos criar um loop de espera ocupada
    - Fizemos isso com nossas rotinas de delay_omic e delay_debounce

- Vantagens? Desvantagens?

  \+ Precisão: montamos os delays, e sabemos exatamente quantos ciclos de clock cada instrução precisa para ser executada (veja no manual)

  \+ Simplicidade: é relativamente simples criar esses loops

  \- Trabalho jogado fora: o microcontrolador fica preso em um loop que não faz nada útil
    - Gasta energia e processamento

Gerando delays

- Opção 1
  - Podemos criar um loop de espera ocupada
    - Fizemos isso com nossas rotinas de delay_omic e delay_debounce

- Vantagens? Desvantagens?

  \+ Precisão: montamos os delays, e sabemos exatamente quantos ciclos de clock cada instrução precisa para ser executada (veja no manual)

  \+ Simplicidade: é relativamente simples criar esses loops

  \- Trabalho jogado fora: o microcontrolador fica preso em um loop que não faz nada útil

  - **É exatamente essa estratégia que os compiladores geralmente usam com suas funções de delay**

Gerando delays

- Utilização de *timers*
  - Quando disponíveis em seu hardware, os *timers* podem gerar eventos de tempos em tempos
  - Os eventos podem incluir
    - Setar uma flag em algum registrador
    - Gerar uma interrupção
  - Os timers contam o tempo de acordo com algum sinal de clock
  - Se configurados corretamente, podem gerar contagens precisas e sem gastar tempo da CPU com espera ocupada

Timers do ATMega328p

- O ATMega328p possui 3 timers distintos
  - Timer0 → Registrador de 8 bits
  - Timer1 → Registrador de 16 bits
  - Timer2 → Registrador de 8 bits
- Vamos usar o Timer1

Timer1

- Os detalhes estão na Seção 12 do manual

Timer1 – Modo de contagem

- Bits WGM13 a WGM10
  - Veja as configurações completas na Tabela 16-4

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP
0	0	0	0	0	Normal	0xFFFF
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF
4	0	1	0	0	CTC	OCR1A

Vamos usar essa configuração. O timer conta até atingir o valor de referência armazenado em OCR1A

UDESC

Timer1 – Modo de contagem

- Bits WGM13 a WGM10

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP
0	0	0	0	0	Normal	0xFFFF
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF
4	0	1	0	0	CTC	OCR1A

16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

UDESC

TCCR1A

- Contém os dois primeiros bits de WGM
  - Demais bits podem ser zero para criarmos um contador simples

16.11.1 TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1B

- Contém os dois últimos bits de WGM
  - ICNC1 → habilita o cancelador de ruído. Especialmente útil para clocks externos. Vamos deixar **0**.
  - ICES1 → A contagem é feita quando o sinal de clock sobe (1) ou desce (0). Vamos deixar **1**.
  - Os bits CS servem para definir o **Prescaler**

16.11.2 TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

UDESC

Prescaler

- O Prescaler (comum em muitas CPUS) serve para definir quantos ciclos de clock são necessários para somar 1 em no Timer
  - Geralmente são definidos em potência de 2

A cada subida de clock, contar um no Timer

A cada 1024 subidas de clock, contar um no Timer

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}$/1 (No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

UDESC

Prescaler

- Quais as vantagen/desvantagens de se usar o prescaler?

Prescaler

- Quais as vantagen/desvantagens de se usar o prescaler?
  - Ao colocar uma divisão alta, por exemplo
    + O timer demora mais antes de gerar um *overflow*
      - Podemos “esperar mais tempo”
    - Perdemos precisão

Prescaler

- Como exemplo, vamos usar um prescaler de 1/256

OCR1AH e OCR1AL

- OCR1AH e OCR1AL representam um **número de 16 bits**. Em **nossa configuração**
  - O timer vai contar até atingir o valor armazenado nesses 16 bits
  - Quando isso ocorrer, uma interrupção será gerada
  - O contador é zerado, e começará a contar mais uma vez

16.11.5 OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Configurando

- Constantes extras necessárias

```
.equ TCCR1A,0x80
.equ TCCR1B,0x81
.equ OCR1AH,0x89
.equ OCR1AL,0x88
.equ TIMSK1,0x6F
.equ TIFR1, 0x16
```

Configuração do Timer1

```
;configurando TIMERS e interrupção de timer
ldi r16, 0b00000000
sts TCCR1A, r16
ldi r16, 0b01001100
;borda de subida. Compara com OCR1A. Prescaller de 1/256
sts TCCR1B, r16
ldi r16, 0xFF
sts OCR1AH, r16
sts OCR1AL, r16 ; contar até 0xFFFF
ldi r16,0b00000010 ; comparar com OCR1A e gerar interrupção
sts TIMSK1, r16
ldi r16, 0xFF
out TIFR1, r16 ;mandando 1 em todos os flags para apagar os flags
sei
;fim de configuração de TIMER
```

Configuração do Timer1

O Timer vai contar até 0xFFFF
antes de gerar uma interrupção

```
;configurando TIMERS e interrupção de timer
ldi r16, 0b00000000
sts TCCR1A, r16
ldi r16, 0b01001100
;borda de subida. Compara com OCR1A. Prescaller de 1/256
sts TCCR1B, r16
ldi r16, 0xFF
sts OCR1AH, r16
sts OCR1AL, r16 ; contar até 0xFFFF
ldi r16,0b00000010 ; comparar com OCR1A e gerar interrupção
sts TIMSK1, r16
ldi r16, 0xFF
out TIFR1, r16 ;mandando 1 em todos os flags para apagar os flags
sei
;fim de configuração de TIMER
```

UDESC

Tratador de interrupção

Somar 1 em r31 (valor exibido no display)

```
intTimer1:
        push r16
        in r16, SREG
        push r16

        inc r31

        pop r16
        out SREG, r16
        pop r16
        reti
```

UDESC

Vetor de interrupção

- Qual o endereço do vetor de interrupção para Timer1 quando comparando com OCR1A?

Vetor de interrupção

- Qual o endereço do vetor de interrupção para Timer1 quando comparando com OCR1A?
  - **Palavra** 0x16
    - Endereço **0x32** efetivo de memória

```
.org 0x0
   rjmp main
.org 0x32
   rjmp intTimer1

;...
```

10	0x0012	TIMER2 OVF	Timer/Counter2 Overflow
11	0x0014	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x0016	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x0018	TIMER1 COMPB	Timer/Coutner1 Compare Match B

No exemplo

- Estamos utilizando o clock interno de 8Mhz do microcontrolador para alimentar o Timer1
- Prescaler de 1/256
- Contagem até 0xFFFF antes de gerar uma interrupção
- **A cada quantos segundos o valor em r31 será incrementado?**

No exemplo

- Estamos utilizando o clock interno de 8Mhz do microcontrolador para alimentar o Timer1
- Prescaler de 1/256
- Contagem até 0xFFFF antes de gerar uma interrupção
- **A cada quantos segundos o valor em r31 será incrementado?**
  - **$(2^{16} \times 256) \div (8 \times 10^6)$ = 2,097152 segundos**
  - Número de ticks do clock necessários para a contagem chegar até 0xFFFF
  - Número de ticks de clock por segundo gerados pelo oscilador do microcontrolador

1 Segundo!

- Como fazer para que o contador incremente a cada exatamente 1 segundo?

1 Segundo!

- Como fazer para que o contador incremente a cada exatamente 1 segundo?
  - $(x * 256)/(8*10^6) = 1$
  - $X = 31250_{10} = 7A12_{16}$
  - Note que poderíamos ter ajustado também o prescaler (256)

Cuidado!

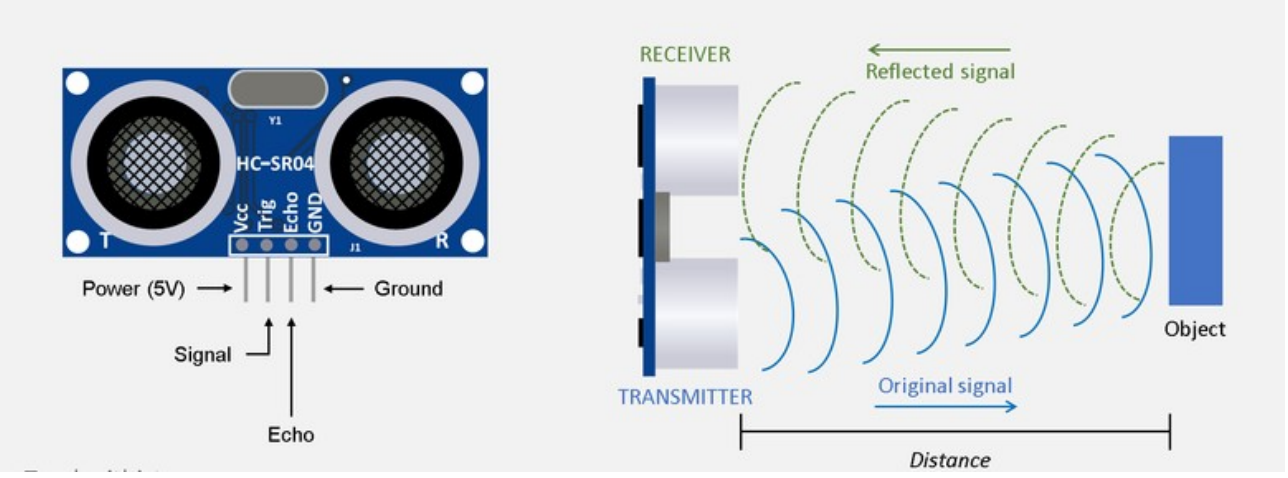
- O sinal de clock interno do ATMega (e PIC) são gerados por um circuito **Resistor-Capacitor**
  - Impreciso.
  - De acordo com o manual, o clock pode **variar até 10%** !!!
    - Depende da sua sorte, da temperatura, da tensão, ...
  - Ou seja, seu microcontrolador pode estar operando com qualquer clock entre 7,2 e 8,8 Mhz
- Em aplicações onde o tempo é crítico, utilize cristais de clock externos mais precisos

Exemplo de aplicação

- Sensor de estacionamento
  - Geralmente utilizados em carros
  - Comumente são ultrassônicos
    - Emitem um sinal sonoro em uma frequência acima de 20MHz (através de um alto-falante), e esperam o sinal refletir em algo (escutam o sinal por um microfone)
    - Calculando o tempo entre o sinal ser gerado, e escutado pelo microfone, e considerando a velocidade do som, determinamos a distância dos objetos

Exemplo de aplicação

- Podemos, por exemplo
  - Mandar o sinal sonoro e inicializar um *timer*
  - Quando o sinal sonoro for captado pelo microfone, uma interrupção é gerada
  - Paramos o *timer* ao receber a interrupção
  - O valor armazenado no *timer* é utilizado calcular a distância dos objetos
  - A medição é tão precisa quanto é precisa nossa medição do tempo



Exercício

- Ajuste o programa para que o contador (que deve ser exibido em um display de 7 segmentos) seja somado em 1, mas em intervalos de 0,5 e 2 segundos, de maneira intercalada (espera 0,5 segundos e soma 1, espera 2 segundos e soma 1, espera 0,5 segundos e soma 1, …). Utilize os timers para contar o tempo.
  - Submeta um vídeo e o código fonte no Moodle

Referências

- T. S. Margush. **Some Assembly Required: Assembly Language Programming with the AVR Microcontroller.** 2016.
- S. Naimi, S. Naimi, M. Mazidi. **The Avr Microcontroller and Embedded Systems Using Assembly and C.** 2010**.**
- **megaAVR® Data Sheet.** Microchip, 2018.
- **ATmega328P Automotive - Complete Datasheet**. Microchip.
- **AVR Instruction Set Manual**. Microchip, 2016.
- D. Patterson; J. Henessy. **Organização e Projeto de Computadores**: **a Interface Hardware/Software.** 5a Edição. Elsevier Brasil, 2017.