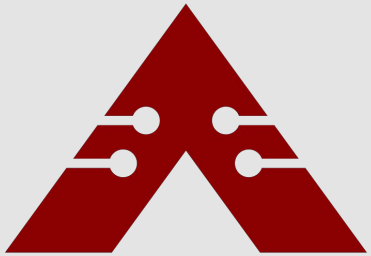


ADA

PROJETOS EM ENGENHARIA
DE COMPUTAÇÃO

Microcontroladores

Uma abordagem do Arduino ao ATmega - Parte 2

Motivação

- Maior controle
- Escalabilidade
- Empreendedorismo
- Baixo custo
- Abre um leque de possibilidades

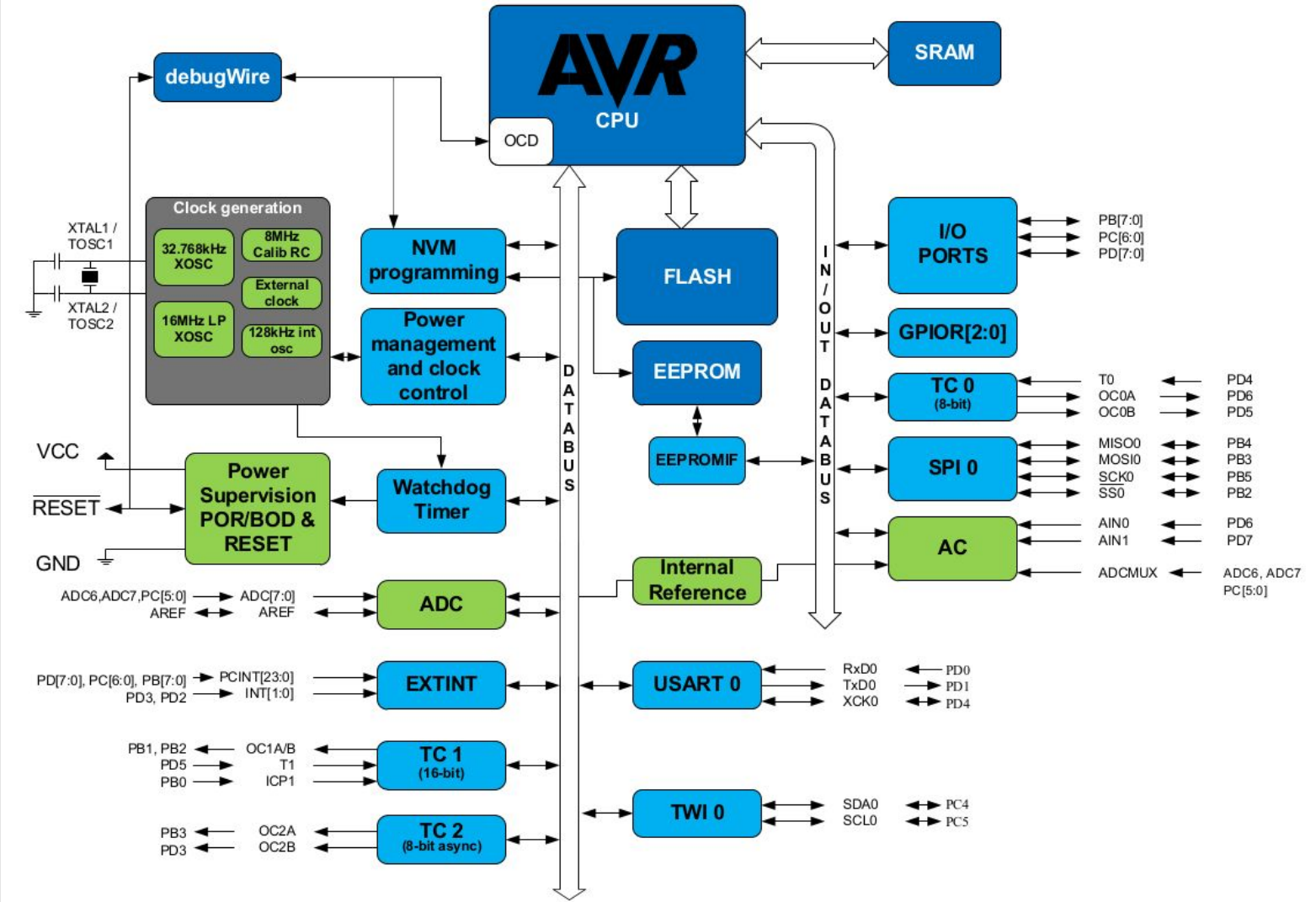
Atmega328

- Microcontrolador de 8 bits
- Arquitetura Harvard RISC (131 instruções)
- 32 x 8 registradores de uso geral
- Até 20MHz
- 2 multiplicadores de dois ciclos

Atmega328

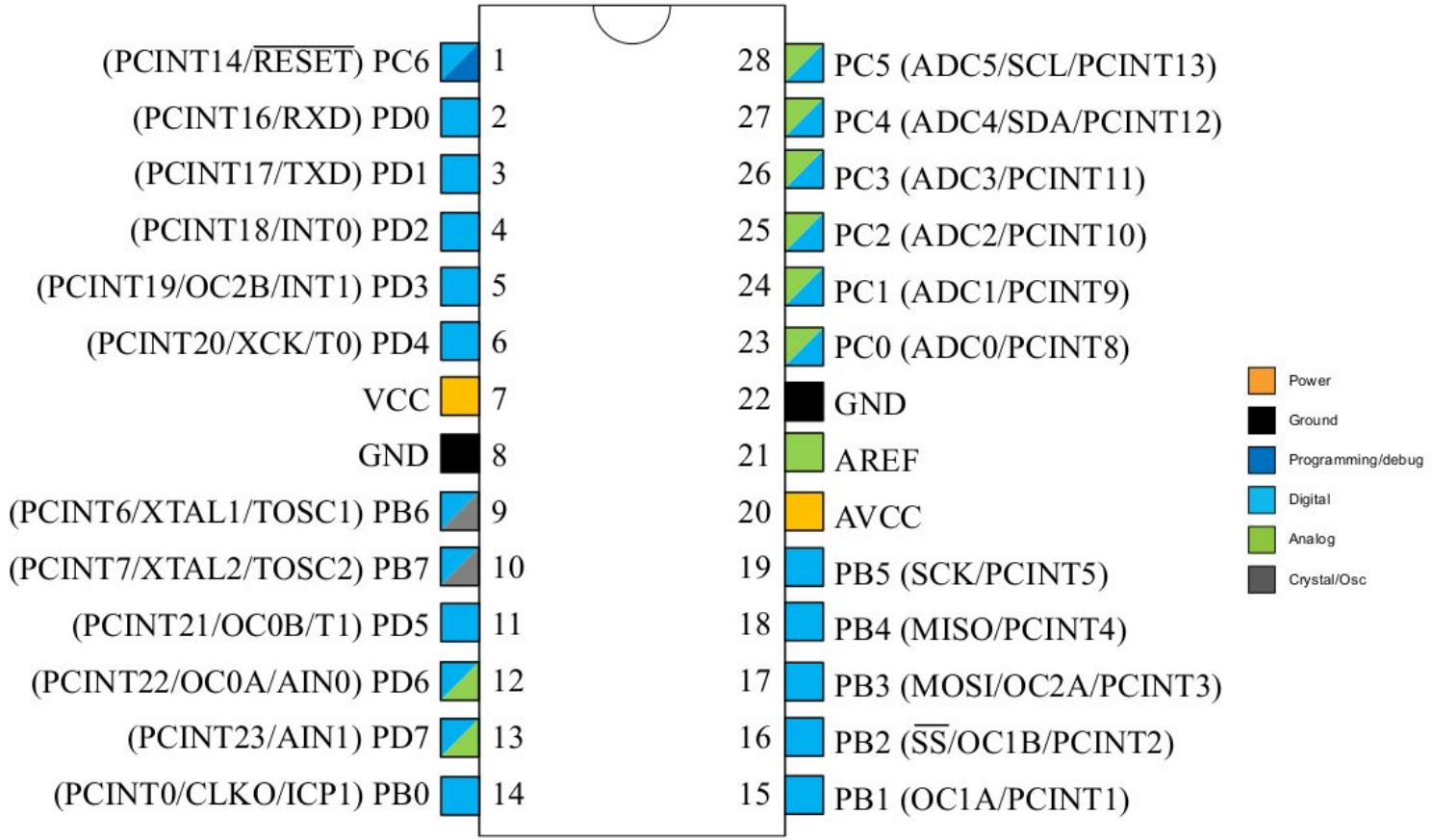
- 32KB de memória de programa Flash
- 1KB de EEPROM
- 2KB de memória SRAM
- 2 temporizadores/contadores de 8 bits
- 1 temporizador/contador de 16 bits

Aproximadamente 2 dólares.
No Brasil em torno de 10 reais.
Na China é possível achar por 5 reais!!


AVR
CPU
OCD
debugWire
SRAM
Clock generation
32.768kHz XOSC
8MHz Calib RC
External clock
16MHz LP XOSC
128kHz int osc
XTAL1 / TOSC1
XTAL2 / TOSC2
NVM programming
Power management and clock control
FLASH
EEPROM
EEPROMIF
DATABUS
I/O PORTS
PB[7:0]
PC[6:0]
PD[7:0]
GPIOR[2:0]
TC 0 (8-bit)
T0
OC0A
OC0B
PD4
PD6
PD5
SPI 0
MISO0
MOSI0
SCK0
SS0
PB4
PB3
PB5
PB2
AC
AIN0
AIN1
PD6
PD7
ADCMUX
ADC6, ADC7
PC[5:0]
VCC
RESET
GND
Power Supervision POR/BOD & RESET
Watchdog Timer
Internal Reference
ADC6,ADC7,PC[5:0]
AREF
ADC[7:0]
AREF
ADC
PD[7:0], PC[6:0], PB[7:0]
PD3, PD2
PCINT[23:0]
INT[1:0]
EXTINT
USART 0
RxD0
TxD0
XCK0
PD0
PD1
PD4
PB1, PB2
PD5
PB0
OC1A/B
T1
ICP1
TC 1 (16-bit)
TWI 0
SDA0
SCL0
PC4
PC5
PB3
PD3
OC2A
OC2B
TC 2 (8-bit async)

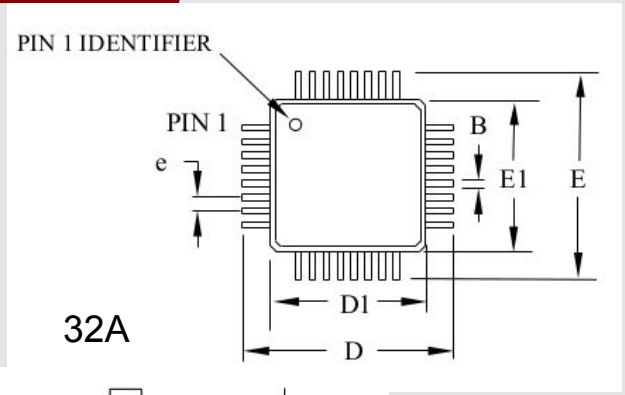
Pin-out

Figure 5-1. 28-pin PDIP

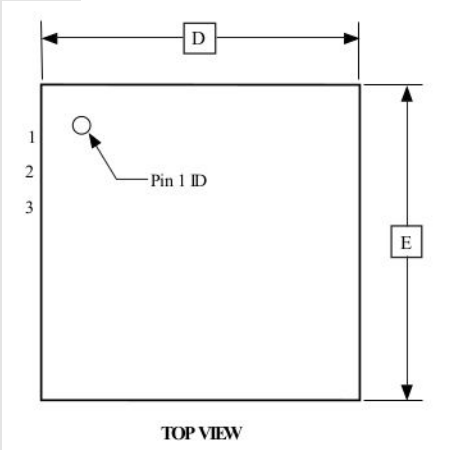


Atmega 328

- Outros encapsulamentos
  - 32A
    - E = 9mm
    - D = 9mm
  - 28M1
    - E = 4mm
    - D = 4mm



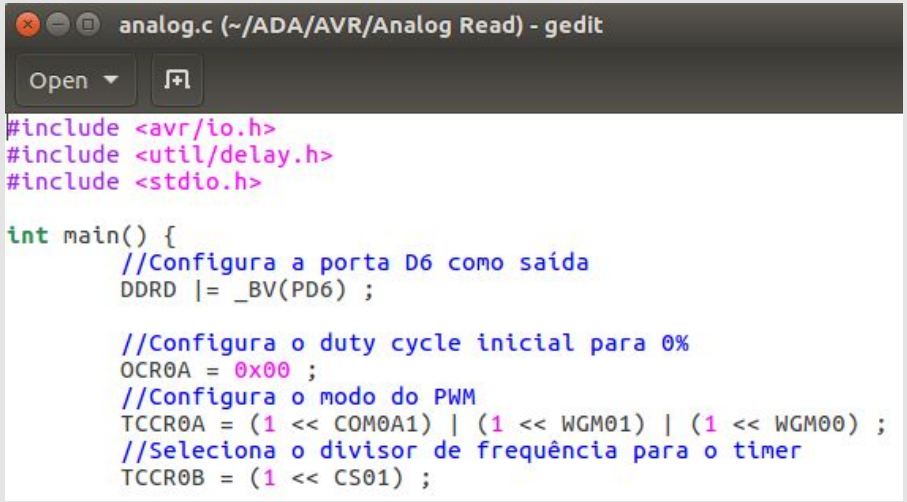
32A

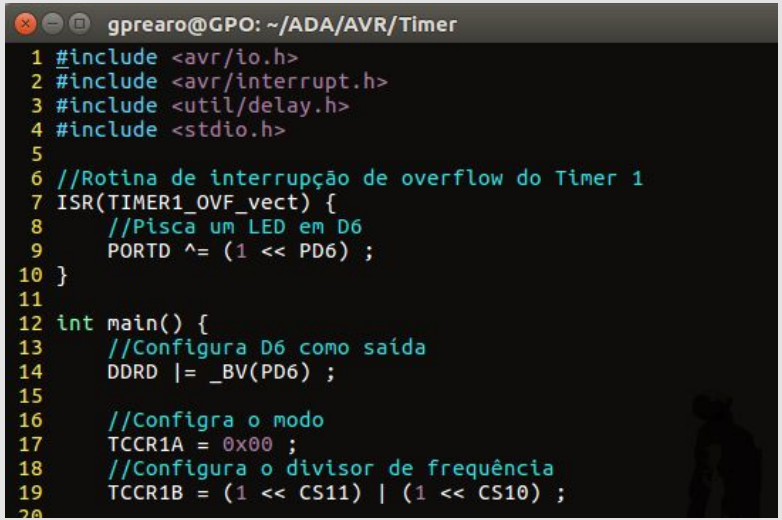


Ferramentas

Editor de texto

- Utilizado para criar nossos programas em C
- Podemos utilizar o `gedit` ou o `sublime`, por exemplo.





Instalação do Necessário

- É necessário ter instalado o compilador e o gravador
- Para distribuições baseadas em Debian (como Ubuntu):
  - `sudo apt-get install gcc-avr gdb-avr  binutils-avr avr-libc avrdude avrdude-doc`
- Para distribuições baseadas em Arch:
  - `sudo pacman -S avr-gcc avr-gdb  avr-binutils avr-libc avrdude`

Compilador

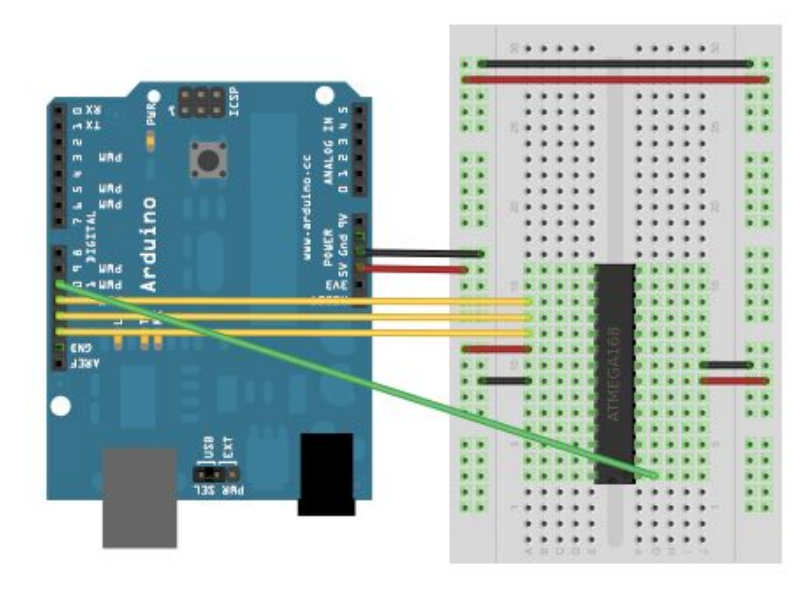
- Leva nosso código de C para o arquivo binário que será gravado no Atmega
- Iremos utilizar o gcc-avr
  - `avr-gcc -Os -DF_CPU=$(CPU_F) -mmcu=atmega328p -c -o $(NAME).o $(NAME).c`
  - `avr-gcc -mmcu=atmega328p $(NAME).o -o $(NAME)`
  - `avr-objcopy -O ihex -R .eeprom $(NAME) $(NAME).hex`

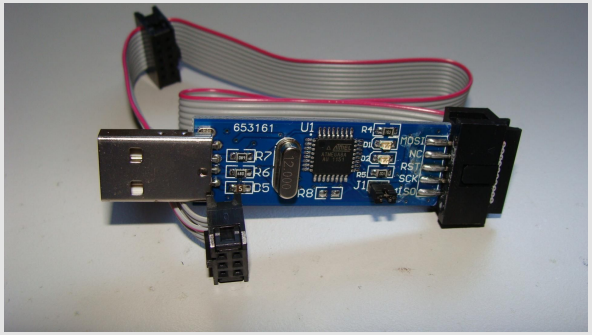
Gravador

- Utilizado para gravar o programa no Atmega
- Iremos utilizar o avrdude
  - `avrdude -c arduino -p m328p -P /dev/ttyACM0 -b 115200`
  - `avrdude -c arduino -p m328p -P /dev/ttyACM0 -b 115200 -U flash:w:$(NAME).hex`

Uso de um Arduino como ISP

- Faz a comunicação entre o computador e o Atmega
- Podemos usar um arduino como gravador ISP





https://www.arduino.cc/en/Tutorial/ArduinoISP

Faremos isso?

- Não
- Gravaremos diretamente no arduino, e então tiraremos o ATmega que está nele.

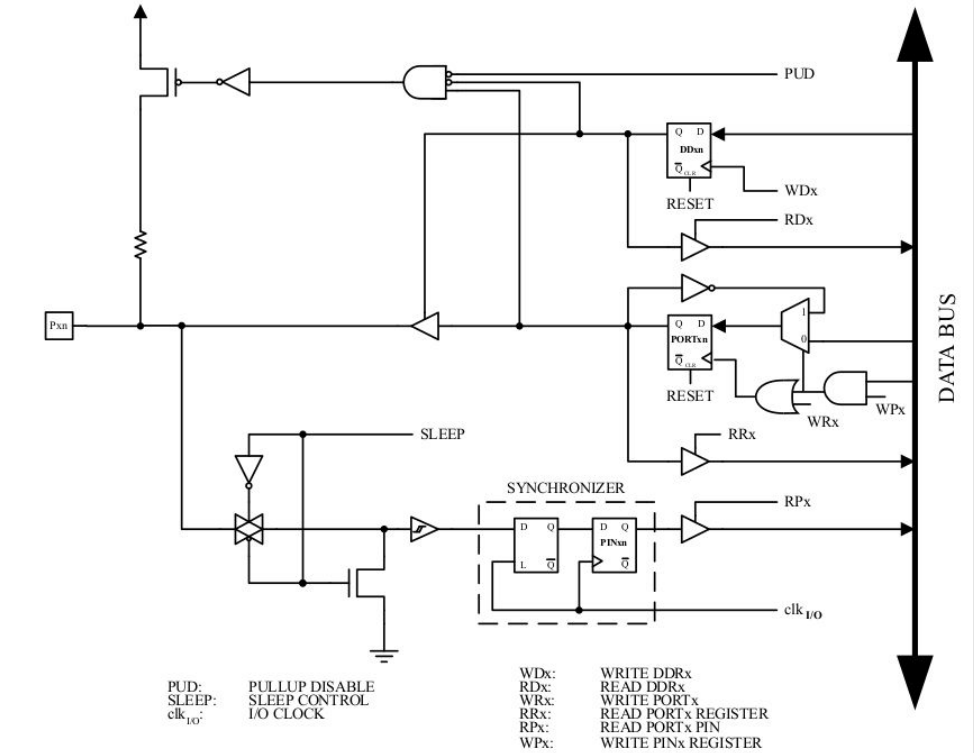
Mão na massa

Portas digitais

- Todas as portas são de entrada e saída
- Portas B (8 bits), C (6 ou 7 bits) e D (8 bits)
- Direção da porta configurada pelo `DDRx` (Data Direction Register)
  - `DDRB, DDRC, DDRD`
- Saída configurada pelos registradores `PORTx`
- Entrada lida pelos `PINx`

Portas digitais

Figure 18-2. General Digital I/O(1)



Portas digitais

Exemplo de código:

- Colocar nível lógico alto em PB5
  - `DDRB = (1<<DDB5) ;  //Configura a porta B5 como saída`
  - `PORTB = (1<<PB5) ;  //Define nível lógico alto na saída B5`
- Ler nível lógico da porta B2
  - `unsigned char i ;`
  - `i = (PINB & (1 << PINB2)) ;  //variável i recebe o conteúdo do pino B2`

Mas o que essas coisas significam?

Operação com bits em C

- unsigned char: tipo em C que representa um byte “puro”
- ~(NOT): inverte o valor lógico dos bits.
- &(AND): realiza a operação lógica E.
- |(OR): realiza a operação lógica OU.
- ^(XOR): realiza a operação lógica OU EXCLUSIVO.
- >>(RIGHT SHIFT): realiza shift à direita.
- <<(LEFT SHIFT): realiza shift à esquerda.

Operação com bits em C

- Seja X=11110000b, Y=11001100 e Z=10101010b.
- ~X=00001111b, ~Y=00110011b e ~Z=01010101b.
- X&Y=11000000b, X&Z=10100000b e Y&Z=10001000b.
- X|Y=11111100b, X|Z=11111010b e Y|Z=11101110b.
- X^Y=00111100b, X^Z=01011010b e Y^Z=01100110b.
- (X>>3)=00011110b e (Y>>5)=00000110b.
- (X<<3)=10000000b, (Y<<5)=10000000b e (1<<5)=00100000b. //1=00000001b

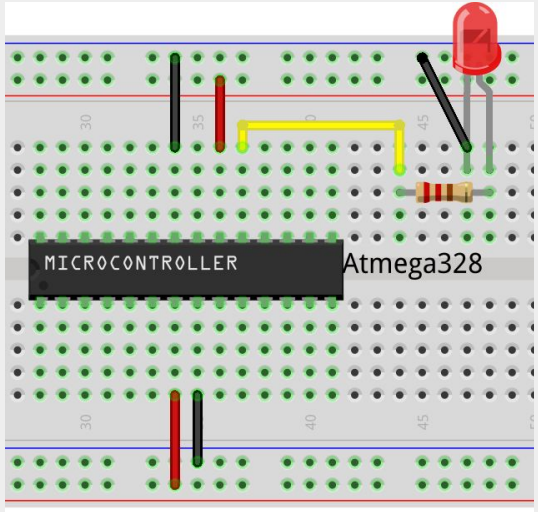
Portas digitais - Exemplos

- Colocar nível lógico alto em PB5
  - `DDRB = (1<<DDB5) ;  //Configura a porta B5 como saída`
  - `PORTB = (1<<PB5) ;  //Define nível lógico alto na saída B5`
- Ler nível lógico da porta B2
  - `unsigned char i ;`
  - `i = (PINB & (1 << PINB2)) ;  //variável i recebe o conteúdo do pino B2`

Exercício 01

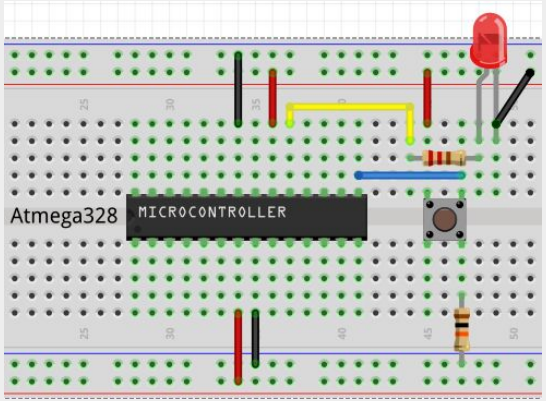
- Fazer um LED piscar usando um bit de uma das portas
- 1s ligado e 1s desligado

Dica: _delay_ms(1000) ;



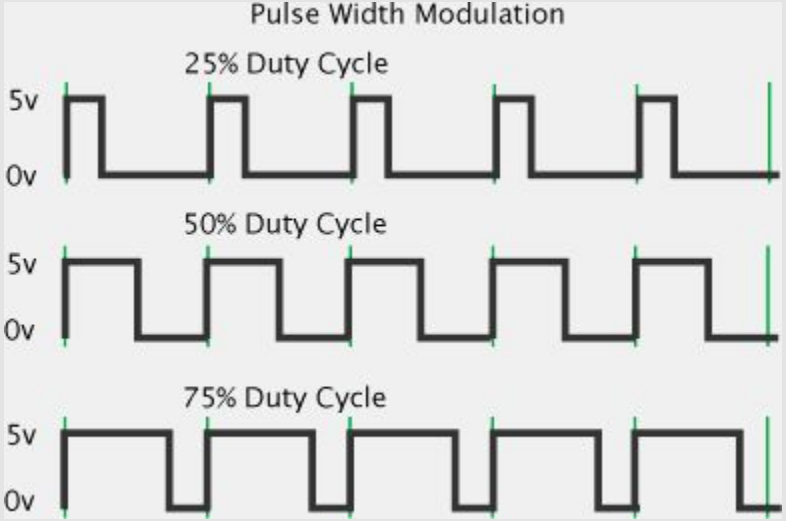
Desafio 01

- Ligar três LEDs e um botão em pinos do ATmega
- O primeiro LED começa aceso
- Ao pressionar o botão, apaga-se o LED aceso e acende o próximo



Saída analógica

- PWM
  - Saída digital pulsada
  - Controle através do *duty cycle*



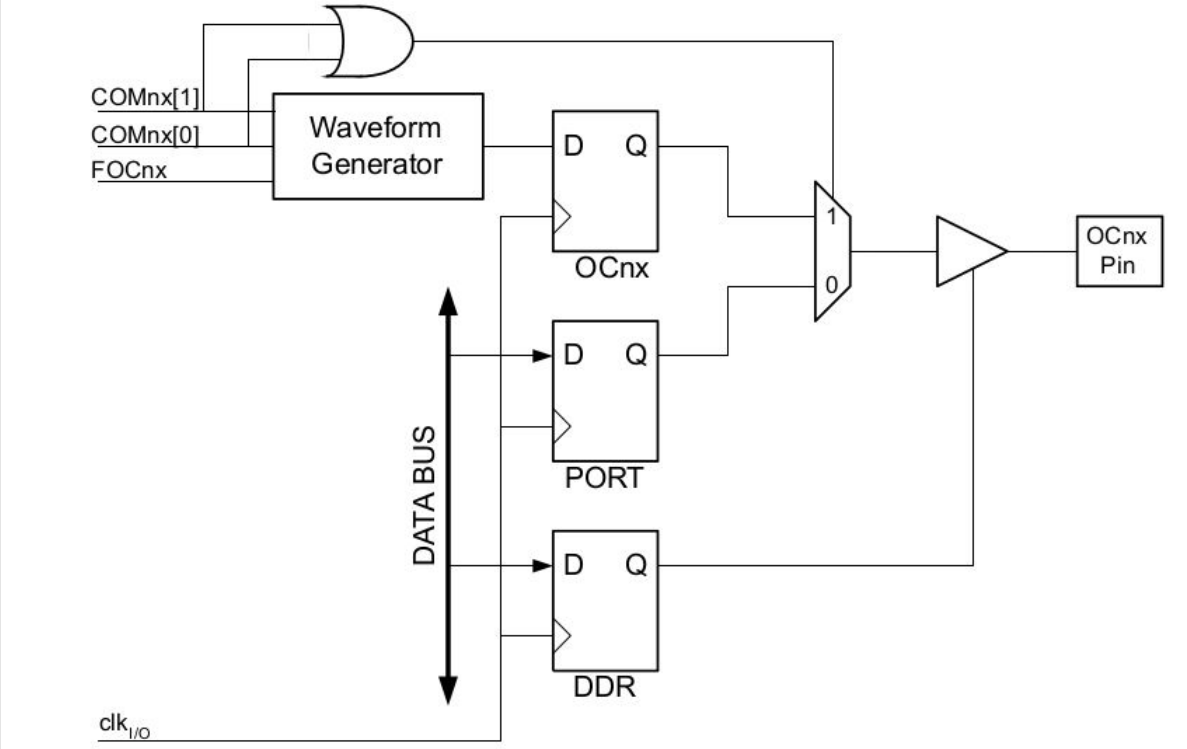
Saída analógica

- Através de PWM por temporizador
- Dois temporizadores de 8 bits (dois canais) e um de 16
- Usaremos o Timer 0, canal A (porta PD6)
- Registradores importantes:
  - `OCR0A (Output Compare Register)`
  - `TCCR0A (Timer/Counter 0 Control Register A)`
  - `TCCR0B (Timer/Counter 0 Control Register B)`

Saída analógica

- No Atmega:
  - Configura-se um temporizador para contar até certo número
    - Os bits 0 e 1 de `TCCR0A` definem disso
    - Usaremos ambos em 1, que equivale a contar até `0xff (Modo Fast PWM)`
  - Configura-se o que acontece quando há *match*
    - Bits 6 e 7 de `TCCR0A`, usaremos 0 e 1 respectivamente
  - Configura-se qual é a frequência de contagem
    - Os bits 0, 1 e 2 de `TCCR0B` definem isso
    - Usaremo 010, que equivale a ⅛ frequência do clock
  - Configura-se até qual número a saída deve ser 5V
    - Este número deve ser atribuído à `OCR0A`

Saída analógica



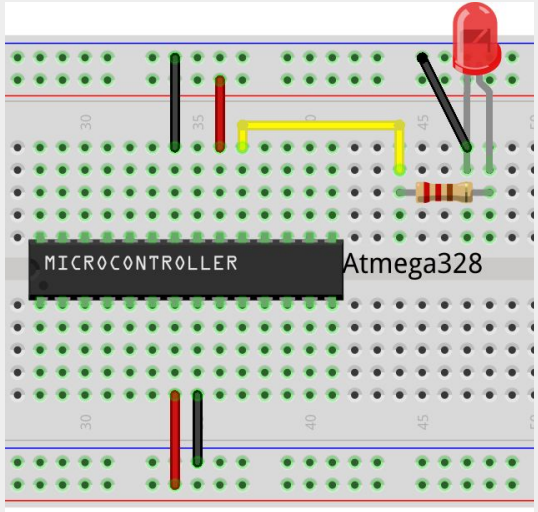
Saída Analógica - Exemplo

- Ativar PWM no pino D6 com duty cycle de 50%
  - `DDRD |= _BV(PD6) ;  //Configura a porta D6 como saída`
  - `OCR0A = 127 ;  //Seta o duty cycle para 50%`
  - `TCCR0A = (1 << COM0A1) | (1 << WGM01) | (1 << WGM00);`
    `//Configura o modo do PWM`
  - `TCCR0B = (1 << CS01) ;  //Configura a frequência do timer`

Exercício 02

- Fazer um LED aumentar gradualmente a intensidade de brilho até chegar ao máximo e então diminuir gradualmente (fazer isso cíclico).

Dica: _delay_ms(1000) ;



Entrada analógica

- Feito através do conversor Analógico/Digital
- Resolução de 10 bits
- Tensão de referência selecionável

$$d = \frac{V_{in} \cdot 1024}{V_{ref}}$$

Entrada analógica

- Registradores importantes
  - ADMUX
    - Os bits 6 e 7 selecionam a tensão de referência
    - O bit 5 ajusta a posição do resultado a conversão
    - Os bits de 0 a 3 selecionam qual canal deve-se realizar a leitura
  - ADCSRA
    - Bit 7 - Habilita a conversão Analógico-Digital
    - Bit 6 - Começa a conversão
    - Bit 4 - Sinaliza quando a conversão termina
    - Bits 0..2 - Frequência de conversão

Entrada analógica

- Registradores importantes
  - ADCH
    - Parte mais significativa do valor convertido
  - ADCL
    - Parte menos significativa
  - Os dois são alterados com ADLAR!!

Entrada Analógica - Exemplo

- Fazer leitura analógica do pino C0 (ADC0)
  - `ADMUX = (1 << REFS0) | (1 << ADLAR) ; //Vref = AVcc; Bits ajustados a esquerda; Canal 0`
  - `ADCSRA  = (1 << ADEN) ; //Habilita a conversão AD`
  - `ADCSRA |= (1 << ADSC) ; //Inicia a conversão AD`
  - `while (!(ADCSRA & (1 << ADIF))) ; //Espera a conversão terminar`
  - `variável = ADCH; //Lê resultado da entrada analógica`

Desafio 02

- Gerar uma entrada analógica usando potenciômetro
- Controlar a intensidade de brilho do LED a partir da leitura do potenciômetro

Interrupções

- Executa um determinado código quando determinado evento ocorre
- Outros códigos podem ser executados enquanto o evento não ocorre
- A interrupção pode ser externa ou interna
  - Exemplos
    - Externa: borda de subida de um sinal em uma das portas
    - Interna: o temporizador terminou a contagem

Interrupções Externas

- Dois tipos de interrupções externas
  - PCINT e INT
- PCINT
  - Registradores importantes
  - PCICR: Pin Change Interrupt Control Register
    - Bit 0 - Habilita interrupções PCINT0 ~ PCINT7
    - Bit 1 - Habilita interrupções PCINT8 ~ PCINT14
    - Bit 2 - Habilita interrupções PCINT16 ~ PCINT23
  - PCMSK: Pin Change Mask Register (de 0 a 2)
    - Habilita interrupções de cada pino

Interrupções

- Interrupção do tipo INT
- Registradores importantes
  - SREG: O bit 7 habilita interrupções
  - EICRA: External Interrupt Control Register A
    - Define os eventos de interrupção da INT0 (PD2) e INT1 (PD3)
  - EIMSK: External Interrupt Mask Register
    - Bit 0 habilita a interrupção INT0
    - Bit 1 habilita a interrupção INT1

Interrupções - Exemplo

- Configurar interrupções no pino D2 (INT0)
  - `DDRD &= ~(1 << PD2) ; //Configura a porta D2 como entrada`
  - `SREG |= 0x80 ; //Habilita interrupções`
  - `EICRA |= 0x03 ; //Configura INT0 para borda de subida do sinal`
  - `EIMSK = 0x01 ; //Habilita a interrupção INT0`

Interrupções - Exemplo

- Tratando interrupções geradas pelo pino D2 (INT0)

```
ISR(INT0_vect) {
    SREG &= ~(0x80) ; //Desabilita interrupções

    // Coloque aqui o que quer fazer na interrupção

    SREG |= 0x80 ; //Habilita novamente as interrupções
}
```

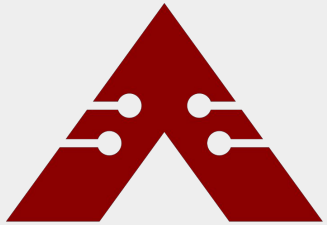
Desafio 03

- O mesmo que o Desafio 1, mas com interrupções
- Ligar três LEDs em pinos do ATmega
- Ligar um botão em um pino de interrupção externa
- O primeiro LED começa aceso
- Ao pressionar o botão, apaga-se o LED aceso e acende o próximo

Desafio Extra

- Implementar um jogo de Genius utilizando LEDs e botões e interrupções do ATmega.
- Deve gerar sequências crescentes de LEDs piscando para o jogador memorizar.
- O jogador deve reproduzir usando os botões.

Obrigado!



ADA

PROJETOS EM ENGENHARIA
DE COMPUTAÇÃO