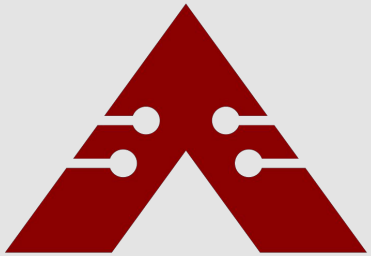


ADA

PROJETOS EM ENGENHARIA
DE COMPUTAÇÃO

Microcontroladores

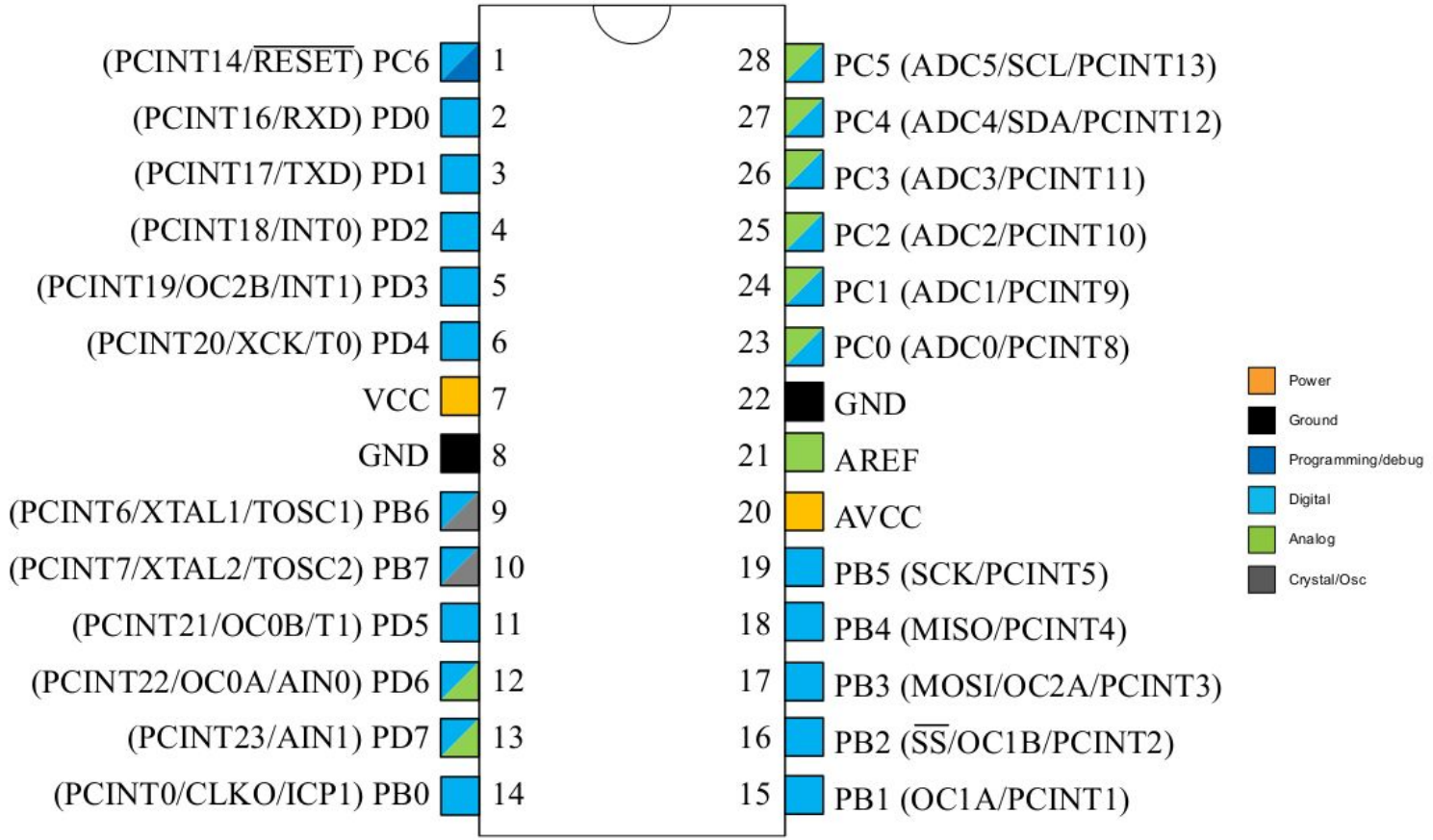
Uma abordagem do Arduino ao ATmega - Parte 2

Motivação

- Maior controle
- Escalabilidade
- Empreendedorismo
- Baixo custo
  - Arduino Uno a patir de R$36
  - ATmega328 a partir de R$15
  - Cotação feita dia 15/10 sem considerar frete

Pin-out

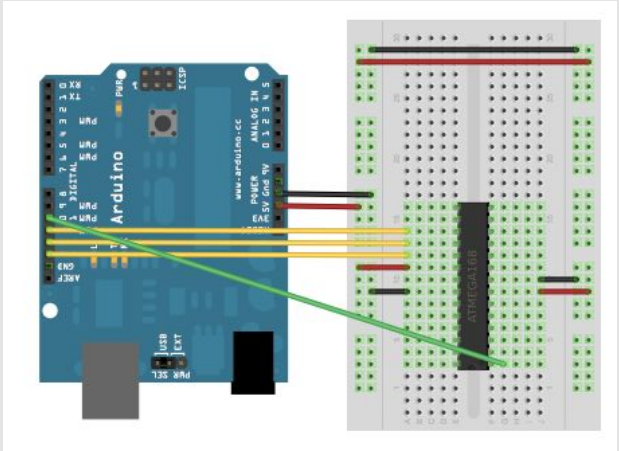
Figure 5-1. 28-pin PDIP



Como gravar?

Gravação ATmega

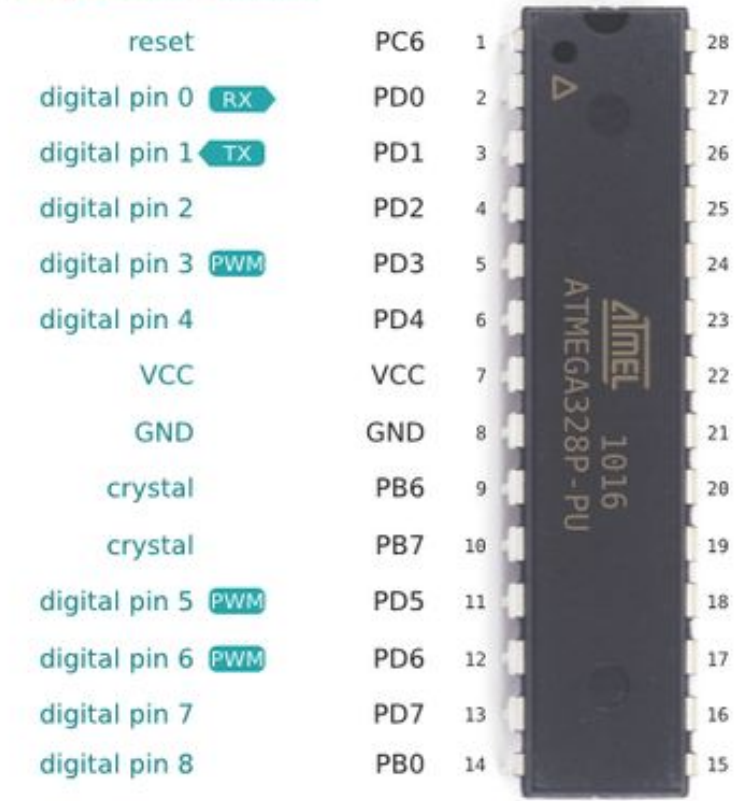
- Faz a comunicação entre o computador e o Atmega
- Podemos gravar o ATmega de duas maneiras:
  - Gravar diretamente o ATmega que fica no Arduino
  - Usar o Arduino como ISP



Como faremos?

- Gravaremos diretamente no arduino
- Usaremos o mapeamento de pinos de ATmega para Arduino para sabermos qual pino será usado

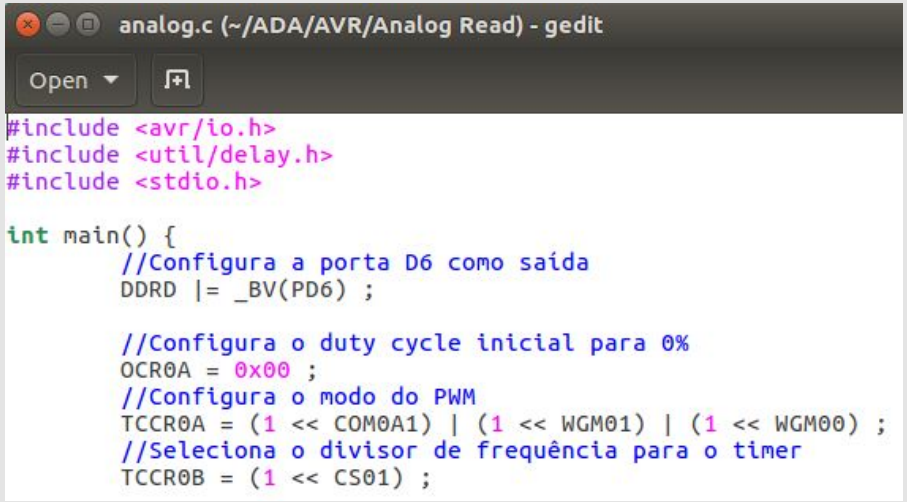
ATmega328P pin mapping

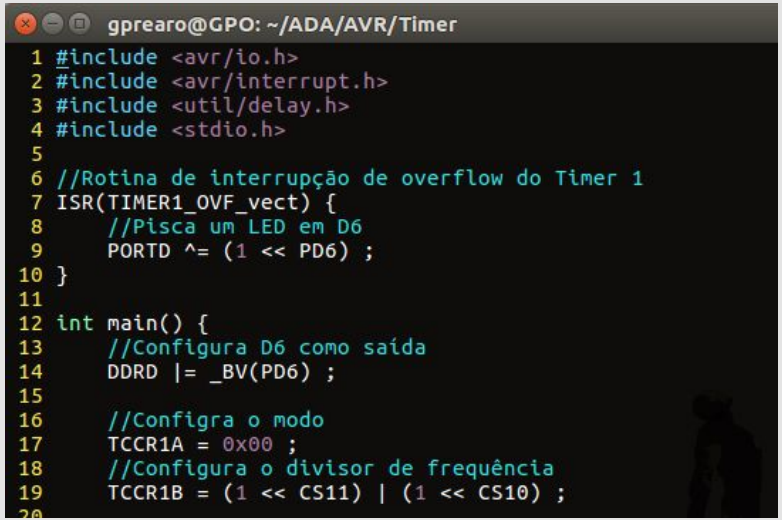


Ferramentas

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`

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)
- Todas as operações envolvem portas e não pinos (ou seja, envolvem bytes e não bits)

Direção do pino

- Define se o pino será de entrada ou saída(trabalho feito por `pinMode()` em Arduino)
- Feita pela configuração dos registradores `DDRx` (Data Direction Register)
  - `DDRB, DDRC, DDRD`
- ‘1’:saída e ‘0’:entrada

Direção do pino

- Para definir a direção de um pino, é preciso modificar o byte inteiro da porta que ele pertence
- Exemplo: configurar o pino B5 como saída
  - Será necessário modificar DDRB como um todo
  - `DDRB |= (1<<DDB5) ;`
  - `DDRB = DDRB | (1<<DDB5) ;`

Mas o que isso significa?

Operação com bits em C

- `DDRB |= (1<<DDB5)`
- `DDRB = DDRB | (1<<DDB5)`

- `|`(OR): realiza a operação lógica OU.
- `<<`(LEFT SHIFT): realiza shift à esquerda.

Operação com bits em C

- (1<<DDB5) = (00000001<<5) = (00100000)
- DDRB = (B7 B6 B5 B4 B3 B2 B1 B0)
- DDRB | (1<<DDB5) = DDRB | (00100000)
- DDRB | (1<<DDB5) = (B7 B6 1 B4 B3 B2 B1 B0)

Operação com bits em C

- Supondo DDRB = 10001100
- DDRB | (1<<DDB5) = (10001100) | (00100000)
- DDRB | (1<<DDB5) = (10101100)

Direção do pino

- E para configurar um pino como entrada?
- Será preciso configurar 0 no DDR correspondente
- Exemplo: configurar o pino D2 como entrada
  - Será necessário modificar DDRD como um todo
  - `DDRD &= ~(1<<DDD2);`
  - `DDRD = DDRD & ~(1<<DDD2);`

E isso, o que significa?

Operação com bits em C

- `DDRD &= ~(1<<DDD2)`
- `DDRD = DDRD & ~(1<<DDD2)`

- ~(NOT): inverte o valor lógico dos bits.
- &(AND): realiza a operação lógica E.

Operação com bits em C

- (1<<DDD2) = (00000001<<2) = (00000100)
- ~(1<<DDD2) = (11111011)
- DDRD = (D7 D6 D5 D4 D3 D2 D1 D0)
- DDRD & ~(1<<DDD2) = DDRD & (11111011)
- DDRD & ~(1<<DDD2) = (D7 D6 D5 D4 D3 0 D1 D0)

Operação com bits em C

- Supondo DDRD = 10001100
- DDRD & ~(1<<DDD2) = (10001100) & (11111011)
- DDRD & ~(1<<DDD2) = (10101000)

Saída e Entrada Digital

- Realizadas após configuração do DDR
- Também realizada ao trabalhar com a porta inteira ao invés de trabalhar com os pinos (byte ao invés de bit)
- Saída configurada pelos registradores `PORTx`
- Entrada lida pelos `PINx`

Portas digitais

Exemplo de Saída:

- Colocar nível lógico alto em PB5
  - `DDRB |= (1<<DDB5) ;  //Configura o pino B5 como saída`
  - `PORTB |= (1<<PB5) ;  //Define nível lógico alto na saída B5`

Vamos começar

- Programa blink - o Hello World dos microcontroladores
  - No pino B5 (Pino 13 do Arduino) já tem um LED

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>

int main() {
    DDRB |= (1 << DDB5); //Configura a porta B5 como output
    while (1) {
        PORTB |= (1 << DDB5); //Define a saída da porta B5 como nível lógico alto
        _delay_ms(500); //Espera por 500ms
        PORTB &= ~(1 << PB5); //Define a saída da porta B5 como nível lógico baixo
        _delay_ms(500); //Espera mais 500ms
    }
    return 0 ;
}
```

Portas digitais

Exemplo 2:

- Ler nível lógico do pino D2
  - `unsigned char i ;`
  - `DDRD &= ~(1<<DDD2);  //Configura o pino D2 como entrada`
  - `i = (PIND & (1 << PIND2)) ;  //variável i recebe o conteúdo do pino D2`

OBS: unsigned char é o tipo em C que representa um byte “puro”

Operador XOR

- ^(XOR): realiza a operação lógica OU EXCLUSIVO.
- Funciona como inversor lógico
  - `PORTB ^= (1<<PB5);  //Inverte o nível lógico na saída B5`
- Supondo PORTB = 10101100
- PORTB ^ (1<<PB5) = (10101100) | (00100000)
- PORTB ^ (1<<PB5) = (10001100)

Usar a entrada

- Mudar estado do LED ao apertar o botão
  - LED no pino D6 e botão no pino C5

```
#include <avr/io.h>
#include <util/delay.h>
#include <stdio.h>

int main() {
    DDRD |= (1 << DDD6); //Configura a porta D6 como saída
    DDRC &= ~(1 << DDC5); //Configura a porta C5 como entrada
    unsigned char entrada = 0; //entrada armazenará o valor lido na porta C
    unsigned char entrada_ant = 1; //entrada_ant armazenará o valor lido anteriormente na porta C
    while (1) {
        //Lógica de Borda
        entrada_ant = entrada; //Atualiza a entrada_ant
        entrada = (PINC & (1 << PINC5)); //Lê o valor novo da entrada
        //Se a entrada entá em '1' e é diferente da anterior:
        //  bordade subida! Então pisca o LED
        if (entrada > 0 && entrada_ant != entrada)
            PORTD ^= (1 << PD6);
        _delay_ms(10); //Espera um tempinho antes de efetuar a próxima leitura
    }
    return 0 ;
}
```

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

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
}
```

Exemplo de Interrupção

- Mudar estado do LED ao apertar o botão
  - LED no pino D6 e botão no pino D2

```
//Rotina da interrupção INT0
ISR(INT0_vect) {
    SREG &= ~(0x80); //Desabilita interrupções
    PORTD ^= (1 << PD6); //Muda o estado da porta D6
    _delay_ms(300); //Debouncing
    SREG |= 0x80; //Habilita novamente as interrupções
}
int main() {
    DDRD &= ~(1 << DDD2); //Configura a porta D2 como entrada
    DDRD |= (1 << DDD6); //Configura a porta D6 como saída
    SREG |= 0x80; //Habilita interrupções
    EICRA |= 0x03; //Configura INT0 para a borda de subida do sinal
    EIMSK = 0x01; //Habilita a interrupção INT0
    while (1) {
    }
    return 0 ;
}
```

Desafio 02

- 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

Projeto Final

- 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