


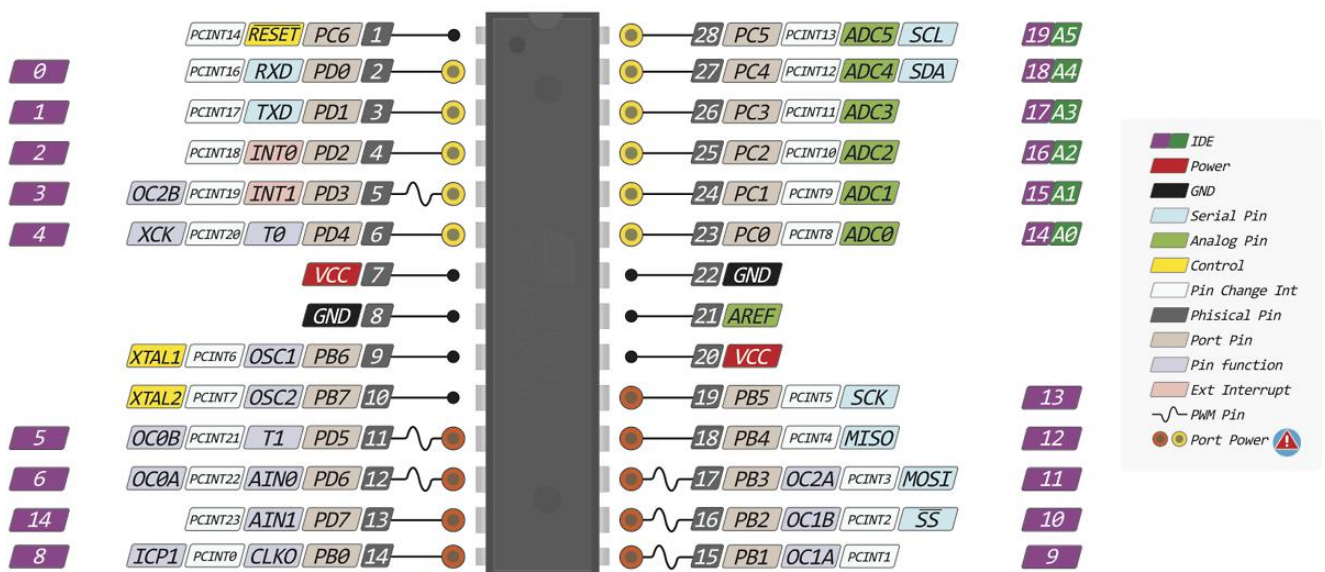
<div> <i>Instituto Nacional de Telecomunicações</i></div>	RELATÓRIO 4		Data: / /	
	Disciplina: E209			
	Prof: João Pedro Magalhães de Paula Paiva			
	Monitores: Thalita Domingos, João Henrique Delfino, Pedro Fraga			
Conteúdo: Microcontroladores AVR				
Tema: GPIO ATmega 328P				
Nome:			Matrícula:	Curso:

Objetivos

- Apresentar os conceitos da arquitetura do microcontrolador Atmega328P.
- Interpretar as funcionalidades dos registros dos pinos GPIO do microcontrolador.
- Utilizar ferramentas para aplicar os firmwares na prática para resolução dos problemas.
- Aplicar na prática a lógica booleana em conjuntos com os operadores booleanos para filtragem de bits.

Parte Teórica:

O microcontrolador Atmega328P



O Atmega328P apresenta conjuntos de “portas” ou “portais” identificados por **PB**, **PC** e **PD**. Cada pino do conjunto possui funcionalidades básicas de I/O (**entradas e saída**). Alguns destes pinos possuem funções especiais, as mesmas serão abordadas em relatórios futuros.

Principais características elétricas do Atmega328P:

- Tensão absolutamente mínima e máxima de operação $2,3V \leq (V_{CC}) \leq 6V$.
- Tensão de saída no pino I/O em nível lógico 1 (V_{OH}) $> 4.1V \{V_{CC} = 5V\}$.
- Tensão de saída no pino I/O em nível lógico 0 (V_{OL}) $< 0.8V \{V_{CC} = 5V\}$.
- Tensão de entrada no pino I/O para obter nível lógico 1 (V_{IH}) $> 70\% * V_{CC}$.
- Tensão de entrada no pino I/O para obter nível lógico 0 (V_{IL}) $< 10\% * V_{CC}$.
- Corrente máxima total por todos os pinos = **200mA**.
- Corrente máxima em um conjunto = **100mA**.
- Corrente máxima por pino = **40mA**.

Referências: Pg. 258 - 259 do datasheet

Input low voltage, XTAL1 pin	$V_{CC} = 2.7V \text{ to } 5.5V$	V_{IL1}	-0.5		$0.1V_{CC}^{(1)}$	V
Input high voltage, XTAL1 pin	$V_{CC} = 2.7V \text{ to } 5.5V$	V_{IH1}	$0.7V_{CC}^{(2)}$		$V_{CC} + 0.5$	V
Output low voltage ⁽³⁾	$I_{OL} = 20mA, V_{CC} = 5V$ $I_{OL} = 5mA, V_{CC} = 3V$	V_{OL}			0.8 0.5	V
Output high voltage ⁽⁴⁾	$I_{OH} = -20mA, V_{CC} = 5V$ $I_{OH} = -10mA, V_{CC} = 3V$	V_{OH}	4.1 2.3			V

O uso da linguagem C nos microcontroladores:

A linguagem C, inicialmente criada para desenvolvimento de programas de computador, foi aos poucos sendo substituída por outras linguagens que facilitavam o desenvolvimento, mas ganhou uma sobrevida devido ao seu uso nos sistemas embarcados, ou seja, nos microcontroladores.

A sintaxe da linguagem é a mesma, seja para PC ou para MCU, mas cabe salientar algumas observações:

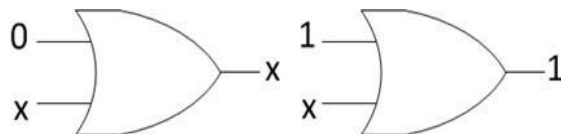
- Deve-se prestar atenção nos tipos de variáveis utilizadas (**char, short, int, long**) devido a limitação de espaço de memória de dados (RAM).
- Normalmente, os firmwares não possuem fim. Dessa forma, utiliza-se estruturas de repetição infinita (**loop-infinito**) no programa: **for(;;);** ou **while(1);**.
- Quando os programas são de baixa complexidade e apresentam lógica simples, pode-se utilizar uma execução sequencial, que possibilita a implementação prática da máquina de estados:
 - o realiza a leitura das entradas e armazena em variáveis,
 - o interpreta os valores das variáveis e executa a lógica desejada,
 - o atualiza as saídas (método denominado **super-loop**).
- É boa prática utilizar recursos que facilitam a alteração do uso dos pinos de GPIO/portais. Normalmente isso é feito utilizando a diretiva **"#define"**. Dessa forma, caso um periférico tenha que ser trocado de pino, fica simples adaptar o programa. Exemplo:

```
#define P7 0b10000000  
#define P4 0b00010000
```

Técnica de mascaramento:

Durante o curso, será muito comum utilizar bits para manipulação dos registros. Porém, a arquitetura do Atmega328P é de 8-bits, ou seja, as variáveis mínimas são de 8-bits (Byte). Para manipular bits, utiliza-se a aritmética binária com a lógica “OU” e “E” da seguinte forma:

- **Lógica OU:** possível fazer com que uma informação **X** seja “1”. Se fizermos a lógica **OU** entre “bit qualquer” e “1”, o resultado sempre será “1”. Se fizermos a lógica **OU** entre “bit qualquer” e “0”, o resultado será o valor do “bit qualquer”.



Exemplos:

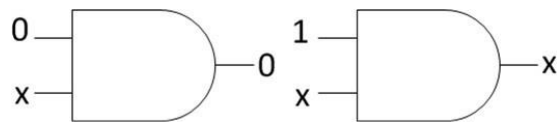
Escrever “1” no bit 0: $PORTx = PORTx \mid 0b00000001$;

PORTx - bits	7	6	5	4	3	2	1	0
PORTx antes	X	X	X	X	X	X	X	X
Máscara a ser aplicada	0	0	0	0	0	0	0	1
PORTx depois	X	X	X	X	X	X	X	1

Escrever “1” no bit 6: $PORTx = PORTx \mid 0b01000000$;

PORTx - bits	7	6	5	4	3	2	1	0
PORTx antes	X	X	X	X	X	X	X	X
Máscara a ser aplicada	0	1	0	0	0	0	0	0
PORTx depois	X	1	X	X	X	X	X	X

- **Lógica E:** possível fazer com que uma informação **X** seja “0” ou mascarar(filtrar) uma informação **X** desejada para ser lida. Se fizermos a lógica **E** entre “bit qualquer” e “0”, o resultado sempre será “0”. Se fizermos a lógica **E** entre “bit qualquer” e o valor “1”, o resultado será o valor do “bit qualquer”.



Exemplos:

Escrever “0” no bit 0: $PORTx = PORTx \& \sim(0b00000001);$

PORTx - bits	7	6	5	4	3	2	1	0
PORTx antes	X	X	X	X	X	X	X	X
Máscara a ser aplicada	1	1	1	1	1	1	1	0
PORTx depois	X	X	X	X	X	X	X	0

Escrever “0” no bit 6: $PORTx = PORTx \& \sim(0b01000000);$

PORTx - bits	7	6	5	4	3	2	1	0
PORTx antes	X	X	X	X	X	X	X	X
Máscara a ser aplicada	1	0	1	1	1	1	1	1
PORTx depois	X	0	X	X	X	X	X	X

Ler a informação contida no bit 3: $var = PINx \& 0b00001000;$

PINx - bits	7	6	5	4	3	2	1	0
PINx antes	X	X	X	X	X	X	X	X
Máscara a ser aplicada	0	0	0	0	1	0	0	0
VAR	0	0	0	0	X	0	0	0

Parte prática:

1. Crie um circuito seguindo o esquemático elétrico apresentado, e a partir do programa base anexado, construa a lógica para resolução dos seguintes problemas.
 - a. Execute o programa base e desenhe um **diagrama em blocos** que represente o circuito criado.
 - b. **Modifique o programa** para que o **LED2** seja acionado somente quando o **botão S2** for pressionado.
 - c. **Modifique o programa** para que o **LED1** e **LED2** sejam acionados **alternadamente**, com intervalos de **1s** enquanto o **botão S2** for pressionado, e com intervalos de **100ms** quando o **botão S1** for pressionado.
2. **Crie uma máquina de estados** de sua preferência, e utilizando o circuito base **programe um firmware** para executar a lógica imaginada. A lógica deverá fazer uso dos portais para entrada e saída de dados.

Programa base:

```
int main(void)
{
    DDRD = DDRD | 0b10000000; // Pino PD7 definido como saída
    PORTD = PORTD | 0b00100000; // Habilitar PULL-UP no PD5
    PORTD = PORTD & ~(0b10000000); // Desliga a saída PD7

    for (;;) // Super Loop
    {
        int botao = PIND & 0b00100000; // Lê o estado do PD5

        if (botao == 0) // Botão está pressionado ?
        {
            PORTD = PORTD | 0b10000000; // PD7 -> HIGH
            _delay_ms(1000); // Espera 5s
            PORTD = PORTD & ~(0b10000000); // PD7 -> LOW
        }
    }
}
```

Circuito base: <https://www.tinkercad.com/things/0UmXJi9DrdS-e209-rel-4-programa-base>

