

# Microcontrolador Atmega328 que realiza contagem de valores entre 00 até 29

Lustoza. Á. A, Silva.C. J. P, Magano. H. Jr, Barros. H. S, Silva Neto. J. C. N, Soares. J. T. M

**Resumo**— Este artigo tem por objetivo transparecer uma experimentação, prática, de um microcontrolador (Atmega328) na realização de contagem de valores entre 00 até 29 para fins de ensino didáticos de disciplinas que envolvam compreensão do funcionamento de arquiteturas de microcontroladores. Bem como o uso de ferramentas de desenvolvimentos de programação e depuração de microcontroladores.

No decorrer do trabalho, foram investigadas operações do Atmega328 no processamento das informações que lhe são entregues. Tais processo englobam a decodificação dos dados, a execução de comandos específicos e a reação a eventos predefinidos. O ressaltar a importância primordial dessa funcionalidade em aplicações práticas, como sistemas embarcados, onde uma comunicação eficiente figura como elemento essencial para o funcionamento adequado do dispositivo.

**Palavras-Chave**— Sistemas embarcados, microcontrolador, Atmega328, linguagem de programação C.

## I. INTRODUÇÃO

Diversamente do senso comum, a utilização da linguagem assembly em microcontroladores permanece amplamente empregada nos tempos atuais. Neste momento, os microcontroladores encontram-se inseridos em praticamente todos os dispositivos eletrônicos que operam de forma digital, como nas casas, em máquinas de lavar, fornos de micro-ondas, televisores, aparelhos de som e imagem, ar condicionado e telefones; em sistemas de injeções eletrônicas de veículos automotores, controladores de estabilidade, freios ABS (Anti-lock Braking System), computadores de bordo e GPS (Global Positioning System); também podemos citar em eletrônicos portáteis, em telefones celulares, tocadores de mídia eletrônica, vídeo games e relógios; já na indústria, temos suas aplicações em controladores lógico programáveis, em motores e fontes de alimentação. Em resumo, os microcontroladores encontram-se em todos os segmentos eletrônicos, desde simples equipamentos domésticos até sistemas industriais complexos. [1].

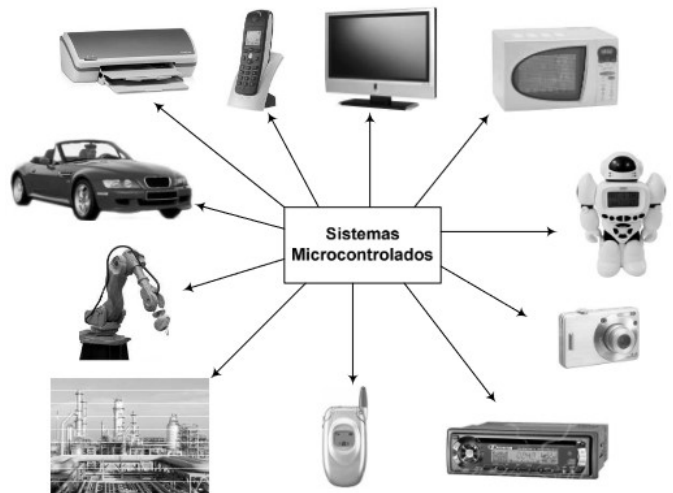


Fig. 1

SISTEMAS MICROCONTROLADOS [1].

Juntamente aos microcontroladores, existem diversas linguagens de programação, não tão de baixo nível como *assembly*, a serem utilizadas como formas de acessibilidade e portabilidade de uso, facilitando, assim, a programação sobre os microcontroladores. Dentre as diversas linguagens, podemos citar a *linguagem de programação em C*, que é uma das principais formas de programação no uso de microcontroladores [1]. Vale ressaltar que esta não é a única linguagem há ser utilizada, além dessa, podemos citar *C++*, *Python*, *Java*, entre outras. A escolha da linguagem depende do microcontrolador bem como dos requisitos do projeto. Linguagens como *C* e *Assembly*, são mais utilizadas para otimização de recursos, enquanto que *Python* é mais voltado para aplicações de prototipagem.

Trabalho como de Ezeugonna(et. all), 2022 propôs um sistema controlador de tráfego baseado em microcontrolador. O sistema, projetado em Proteus e programado com Arduino IDE, utiliza chip Atmega 328, sensores infravermelhos e LEDs. Ele gerencia efetivamente o congestionamento, mudando as faixas afetadas para verde e outras para vermelha, com duração de 9 segundos em situações de emergência e congestionamento. O modelo inclui recursos como campainha, luzes azuis e vermelhas acionadas por sensores infravermelhos durante emergências em estradas de três pistas, visando melhorar o fluxo e a segurança do tráfego [4].

## II. METODOLOGIA

A atividade consiste em criar um programa em C para o microcontrolador ATmega328p, com objetivo de o mesmo realizar uma contagem de, valores decimais de, **00 a 29**. A contagem deve ser mostrada em dois displays de sete segmentos, seguindo as premissas abaixo:

- 1) Deverá haver dois botões (push button) A e B.
- 2) Cada vez que o botão A for pressionado a contagem deverá ser acrescida de um. Ao passar de 29, deverá retornar para 0.
- 3) Cada vez que o botão B for pressionado a contagem deverá ser decrescida de um. Ao passar de 00, deverá retornar para 29.
- 4) Os botões A e B deverão utilizar as interrupções PCINT20 e PCINT21, respectivamente.
- 5) Ao ligar o circuito, a contagem deverá iniciar em 00.
- 6) Com objetivo de reduzir a quantidade de portas digitais do microcontrolador a serem empregadas, utilizou-se dois circuitos integrados registradores de deslocamento modelo **74HC595** ligados em série, sendo um para cada display.
- 7) O programa desenvolvido controla o **74HC595**.
- 8) O código foi escrito na linguagem de programação C, sendo indentado e comentado.

### A. Imagem do circuito ATmega328p no software Proteus

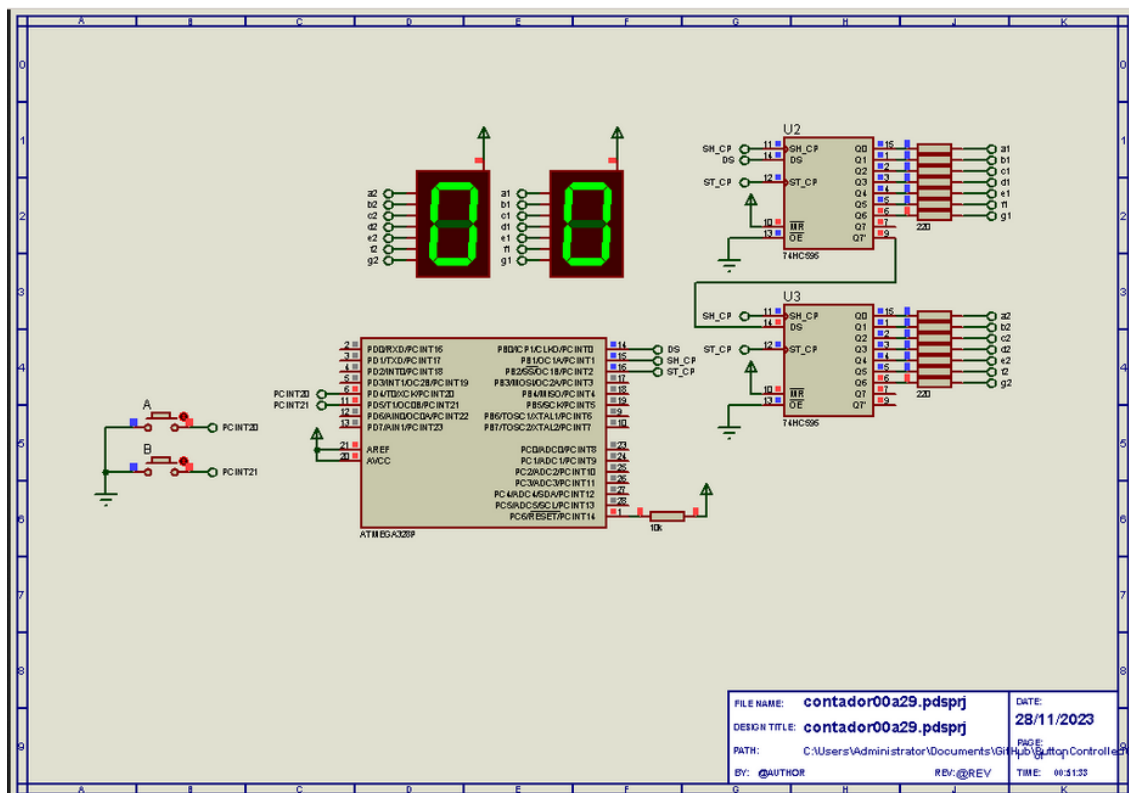


Fig. 2

ATMEGA328P. CLIQUE AQUI PARA VER O GIF

*B. Tabela de decodificação para displays de 7 segmentos*

Dígito	Anodo comum		Catodo comum	
	gfedcba		gfedcba	
<b>0</b>	0b1000000	0x40	0b0111111	0x3F
<b>1</b>	0b1111001	0x79	0b0000110	0x06
<b>2</b>	0b0100100	0x24	0b1011011	0x5B
<b>3</b>	0b0110000	0x30	0b1001111	0x4F
<b>4</b>	0b0011001	0x19	0b1100110	0x66
<b>5</b>	0b0010010	0x12	0b1101101	0x6D
<b>6</b>	0b0000010	0x02	0b1111101	0x7D
<b>7</b>	0b1111000	0x78	0b0000111	0x07
<b>8</b>	0b0000000	0x00	0b1111111	0x7F
<b>9</b>	0b0011000	0x18	0b1100111	0x67
<b>A</b>	0b0001000	0x08	0b1110111	0x77
<b>B</b>	0b0000011	0x03	0b1111100	0x7C
<b>C</b>	0b1000110	0x46	0b0111001	0x39
<b>D</b>	0b0100001	0x21	0b1011110	0x5E
<b>E</b>	0b0000110	0x06	0b1111001	0x79
<b>F</b>	0b0001110	0x0E	0b1110001	0x71

Fig. 3

VALORES PARA A DECODIFICAÇÃO DE DISPLAYS DE 7 SEGMENTOS[1]

### III. CONCLUSÕES

Este estudo mostrou a significância de agregar o microcontrolador Atmega328 com a linguagem de programação C. O destaque incidiu na utilização do microcontrolador, esta pesquisa proporcionou uma visão compreensível acerca da relevância da integração do microcontrolador Atmega328, realçando sua aplicabilidade e contribuições para aplicações práticas em sistemas embarcados [2]ref4).

### REFERÊNCIAS

- [1] Charles Borges de Lima M, **AVR e Arduino. Técnicas de Projeto**. Clube de Autores, 2012.
- [2] SOARES, Gabriel Gutierrez Pereira et al. **Sistema de proteção microcontrolado para motores elétricos de indução trifásicos**. Revista Principia-Divulgação Científica e Tecnológica do IFPB, n. 50, p. 34-46, 2020.
- [3] CARUSO, M. et al. Gerenciamento inteligente de energia de baixo custo baseado no microcontrolador ATmega 328P-PU. In: 6ª Conferência Internacional IEEE 2017 sobre Pesquisa e Aplicações em Energia Renovável (ICRERA) . IEEE, 2017. p. 1204-1209.
- [4] IFEANYI, Ezeugonna Franklyn; SAMUEL, Okoro Ugochukwu; PRO-MISE, Okere. DESIGN OF A MICRO CONTROLLER BASED TRAFFIC LIGHT CONTROLLER SYSTEM FOR CONGESTION AND EMERGENCY SITUATION.

### APÊNDICE

#### A. CÓDIGO EM C

```
1  /**
2  * @file contador00a29.c
3  * @brief Contador com controle de
4  *   ↳ botões utilizando registradores de
5  *   ↳ deslocamento.
6  *
7  * Este programa implementa um contador
8  * com controle de botões utilizando
9  * registradores de deslocamento.
10 * Os botões A e B são utilizados para
11 * incrementar e decrementar o
12 * contador, respectivamente.
13 * O valor do contador é exibido em
14 * dois displays de sete segmentos.
15 * O programa utiliza interrupções para
16 * detectar as ações dos botões e
17 * atualizar o valor do contador.
18 *
19 * @date 24/11/2023
20 * @version 1.0
21 */
22
23 #define F_CPU 8000000UL
24 #include <avr/io.h>
25 #include <util/delay.h>
26 #include <avr/interrupt.h>
27 #define cpl_bit(Y,bit_x)
28   ↳ (Y^=(1<<bit_x)) //troca o estado do
29   ↳ bit x
30 #define tst_bit(y,bit)
31   ↳ (y&(1<<bit))//retorna 0 ou 1
32   ↳ conforme leitura do bit
```

```
33 // Defina os pinos para os botões A e B
34 #define BOTAO_A_PINO PCINT20
35 #define BOTAO_B_PINO PCINT21
36 // Defina os pinos para os
37   ↳ registradores de deslocamento
38 #define DS_PINO PB0 // Data
39 #define SH_CP_PINO PB1 // Clock
40 #define ST_CP_PINO PB2 // Latch
41 ISR(PCINT2_vect); //declara uso de
42   ↳ PCINT2 (pinos PCINT20:21)
43
44 // Variáveis globais para a contagem e
45   ↳ estado dos botões
46 volatile uint8_t contador = 0;
47 volatile uint8_t estadoBotaoA = 0;
48 volatile uint8_t estadoBotaoB = 0;
49
50 // Função para inicializar os
51   ↳ registradores de deslocamento
52 void inicializarShiftRegisters() {
53   // Configurar os pinos como saída
54   DDRB |= (1 << DS_PINO) | (1 <<
55     ↳ SH_CP_PINO) | (1 << ST_CP_PINO);
56 }
57
58 // Função para enviar um byte para os
59   ↳ registradores de deslocamento
60 void enviarByte(uint8_t dado) {
61   for (int i = 7; i >= 0; i--) {
62     // Configurar o pino de dados
63     ↳ (DS)
64     if (dado & (1 << i)) {
65       PORTB |= (1 << DS_PINO);
66     } else {
67       PORTB &= ~(1 << DS_PINO);
68     }
69
70     // Pulsar o pino de clock
71     ↳ (SH_CP)
72     PORTB |= (1 << SH_CP_PINO);
73     PORTB &= ~(1 << SH_CP_PINO);
74   }
75
76   // Pulsar o pino de latch (ST_CP)
77   PORTB |= (1 << ST_CP_PINO);
78   PORTB &= ~(1 << ST_CP_PINO);
79 }
80
81 // Mapeamento de dígitos para códigos
82   ↳ de sete segmentos
83 uint8_t mapaSeteSegmentos[10] = {
84   0b11000000, // 0
85   0b11111001, // 1
86   0b10100100, // 2
87   0b10110000, // 3
88   0b10011001, // 4
89   0b10010010, // 5
90   0b10000010, // 6
```

<pre> 71     0b11111000, // 7 72     0b10000000, // 8 73     0b10010000 // 9 74 }; 75 76 // Função para atualizar os displays de 77   ↳ sete segmentos 78 void atualizarDisplay() { 79     // Mapear os dígitos e enviar para 80     ↳ os registradores de deslocamento 81     uint8_t digitos[2] = 82     ↳ {mapaSeteSegmentos[contador / 83     ↳ 10], mapaSeteSegmentos[contador 84     ↳ % 10]}; 85 86     for (int i = 0; i &lt; 2; i++) { 87         // Enviar o código do dígito 88         ↳ para os displays de sete 89         ↳ segmentos 90         enviarByte(digitos[i]); 91     } 92 } 93 94 // Configurar e inicializar as 95   ↳ interrupções 96 void configurarInterrupcoes() { 97     // Configurar os pinos dos botões 98     ↳ como entrada 99     DDRD &amp;= ~(1 &lt;&lt; BOTAO_A_PINO)   (1 100     ↳ &lt;&lt; BOTAO_B_PINO)); 101     // Ativar resistores de pull-up 102     ↳ para os botões 103     PORTD  = (1 &lt;&lt; BOTAO_A_PINO)   (1 104     ↳ &lt;&lt; BOTAO_B_PINO); 105 106     // Configurar a interrupção para o 107     ↳ botão A (PCINT20) 108     PCICR  = (1 &lt;&lt; PCIE2); 109     PCMSK2  = (1 &lt;&lt; PCINT20); 110 111     // Configurar a interrupção para o 112     ↳ botão B (PCINT21) 113     PCMSK2  = (1 &lt;&lt; PCINT21); 114 115     // Ativar interrupções globais 116     sei(); 117 } 118 119 int main(void) { 120     inicializarShiftRegisters(); 121     configurarInterrupcoes(); 122 123     while (1) { 124         // Atualizar os displays de 125         ↳ sete segmentos 126         atualizarDisplay(); 127         _delay_ms(100); // Adicionando 128         ↳ um atraso de 100 ms </pre>	<pre> 113     } 114 115     return 0; 116 } 117 118 // Rotina de interrupção para os botões 119   ↳ A e B (PCINT20 e PCINT21) 120 ISR(PCINT2_vect) { 121     _delay_ms(10); // Debounce 122     if (!(PIND &amp; (1 &lt;&lt; BOTAO_A_PINO))) { 123         // Incrementar o contador 124         contador = (contador + 1) % 30; 125     } 126     if (!(PIND &amp; (1 &lt;&lt; BOTAO_B_PINO))) { 127         // Decrementar o contador 128         contador = (contador == 0) ? 29 129         ↳ : contador - 1; 130     } 131 } </pre>
---	--

## B. Equações Matemáticas

1 **Appendix** [Código]

2 %Inserir as informações referentes aos  
3 ↳ apêndices aqui.

4 **begin**{minted}[frame=single,  
5 ↳ breaklines=true,  
6 ↳ breakanywhere=true, linenos]{c}

7 /\*\*

8 \* @file contador00a29.c

9 \* @brief Contador com controle de

10 ↳ botões utilizando registradores de  
11 ↳ deslocamento.

12 \*

13 \* Este programa implementa um contador  
14 ↳ com controle de botões utilizando  
15 ↳ registradores de deslocamento.

16 \* Os botões A e B são utilizados para  
17 ↳ incrementar e decrementar o  
18 ↳ contador, respectivamente.

19 \* O valor do contador é exibido em  
20 ↳ dois displays de sete segmentos.

21 \* O programa utiliza interrupções para  
22 ↳ detectar as ações dos botões e  
23 ↳ atualizar o valor do contador.

24 \*

25 \* @date 24/11/2023

26 \* @version 1.0

27 \*

28 \*/

29 #define F\_CPU 8000000UL

30 #include <avr/io.h>

31 #include <util/delay.h>

32 #include <avr/interrupt.h>

33 #define cpl\_bit(Y,bit\_x)

34 ↳ (Y^=(1<<bit\_x)) //troca o estado do  
35 ↳ bit x

36 #define tst\_bit(y,bit)

37 ↳ (y&(1<<bit))//retorna 0 ou 1  
38 ↳ conforme leitura do bit

39 // Defina os pinos para os botões A e B

40 #define BOTAO\_A\_PINO PCINT20

41 #define BOTAO\_B\_PINO PCINT21

42 // Defina os pinos para os

43 ↳ registradores de deslocamento

44 #define DS\_PINO PB0 // Data

45 #define SH\_CP\_PINO PB1 // Clock

46 #define ST\_CP\_PINO PB2 // Latch

47 ISR(PCINT2\_vect); //declara uso de  
48 ↳ PCINT2 (pinos PCINT20:21)

49 // Variáveis globais para a contagem e  
50 ↳ estado dos botões

51 volatile uint8\_t contador = 0;

52 volatile uint8\_t estadoBotaoA = 0;

38 volatile uint8\_t estadoBotaoB = 0;

39 // Função para inicializar os

40 ↳ registradores de deslocamento

41 void inicializarShiftRegisters() {

42 // Configurar os pinos como saída

43 DDRB |= (1 << DS\_PINO) | (1 <<  
44 ↳ SH\_CP\_PINO) | (1 << ST\_CP\_PINO);  
45 }

46 // Função para enviar um byte para os

47 ↳ registradores de deslocamento

48 void enviarByte(uint8\_t dado) {

49 for (int i = 7; i >= 0; i--) {

50 // Configurar o pino de dados  
51 ↳ (DS)

52 if (dado & (1 << i)) {  
53 PORTB |= (1 << DS\_PINO);

54 } else {  
55 PORTB &= ~(1 << DS\_PINO);  
56 }

57 // Pulsar o pino de clock

58 ↳ (SH\_CP)

59 PORTB |= (1 << SH\_CP\_PINO);

60 PORTB &= ~(1 << SH\_CP\_PINO);

61 }

62 // Pulsar o pino de latch (ST\_CP)

63 PORTB |= (1 << ST\_CP\_PINO);

64 PORTB &= ~(1 << ST\_CP\_PINO);

65 }

66 // Mapeamento de dígitos para códigos

67 ↳ de sete segmentos

68 uint8\_t mapaSeteSegmentos[10] = {

69 0b11000000, // 0

70 0b11111001, // 1

71 0b10100100, // 2

72 0b10110000, // 3

73 0b10011001, // 4

74 0b10010010, // 5

75 0b10000010, // 6

76 0b11111000, // 7

77 0b10000000, // 8

78 0b10010000 // 9

79 };

80 // Função para atualizar os displays de

81 ↳ sete segmentos

82 void atualizarDisplay() {

83 // Mapear os dígitos e enviar para

84 ↳ os registradores de deslocamento

85 uint8\_t digitos[2] =

86 ↳ {mapaSeteSegmentos[contador /

87 ↳ 10], mapaSeteSegmentos[contador

88 ↳ % 10]};

```

85     for (int i = 0; i < 2; i++) {
86         // Enviar o código do dígito
87         ↪ para os displays de sete
88         ↪ segmentos
89         enviarByte(digitos[i]);
90     }
91 // Configurar e inicializar as
92 ↪ interrupções
93 void configurarInterrupcoes() {
94     // Configurar os pinos dos botões
95     ↪ como entrada
96     DDRD &= ~(1 << BOTAO_A_PINO) | (1
97     ↪ << BOTAO_B_PINO));
98     // Ativar resistores de pull-up
99     ↪ para os botões
100    PORTD |= (1 << BOTAO_A_PINO) | (1
101    ↪ << BOTAO_B_PINO);
102
103    // Configurar a interrupção para o
104    ↪ botão A (PCINT20)
105    PCICR |= (1 << PCIE2);
106    PCMSK2 |= (1 << PCINT20);
107
108    // Configurar a interrupção para o
109    ↪ botão B (PCINT21)
110    PCMSK2 |= (1 << PCINT21);
111
112    // Ativar interrupções globais
113    sei();
114 }
115
116 int main(void) {
117     inicializarShiftRegisters();
118     configurarInterrupcoes();
119
120     while (1) {
121         // Atualizar os displays de
122         ↪ sete segmentos
123         atualizarDisplay();
124         _delay_ms(100); // Adicionando
125         ↪ um atraso de 100 ms
126     }
127
128     return 0;
129 }
130
131 // Rotina de interrupção para os botões
132 ↪ A e B (PCINT20 e PCINT21)
133 ISR(PCINT2_vect) {
134     _delay_ms(10); // Debounce
135     if (!(PIND & (1 << BOTAO_A_PINO))) {
136         // Incrementar o contador
137         contador = (contador + 1) % 30;
138     }
139     if (!(PIND & (1 << BOTAO_B_PINO))) {
140         // Decrementar o contador

```

```

        contador = (contador == 0) ? 29
        ↪ : contador - 1;
    }
}

```