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, pratica, 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 funcionamentos 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 ressalte 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 nas casas, em máquinas de lavar, fornos de micro-ondas, televisores, aparelhos de som e imagem, ar condicionado e telefones; em sistemas de injeções eltrônicas de veículos automotores, controladores de estabilidade, freios ABS (Anti-lock Braking System), computadores de bordo e GPS (Global Positioning System); também podemos sitar 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].

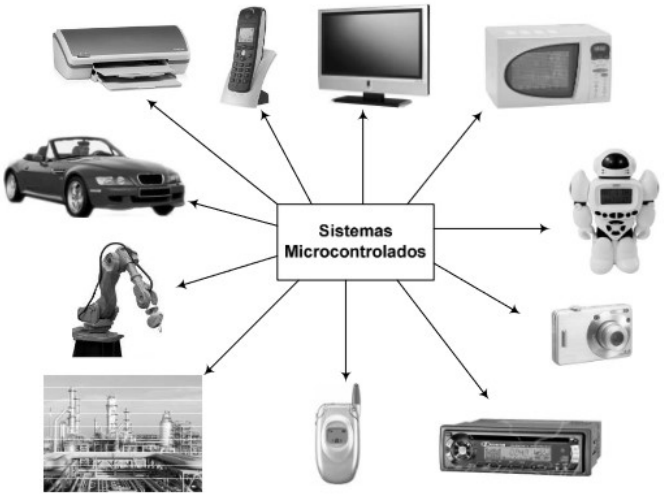

Fig. 1
Sistemas Microcontrolados [1].

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 está não é a unica 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, enguanto que $Python$ é mais voltado para aplicações de prototipagem.

Trabalho como de Ezeugonna(*et. all), 2022* propos 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 abaixos:

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 edentado e comentado.

A. Imagem do circuito ATmega328p no software Proteus

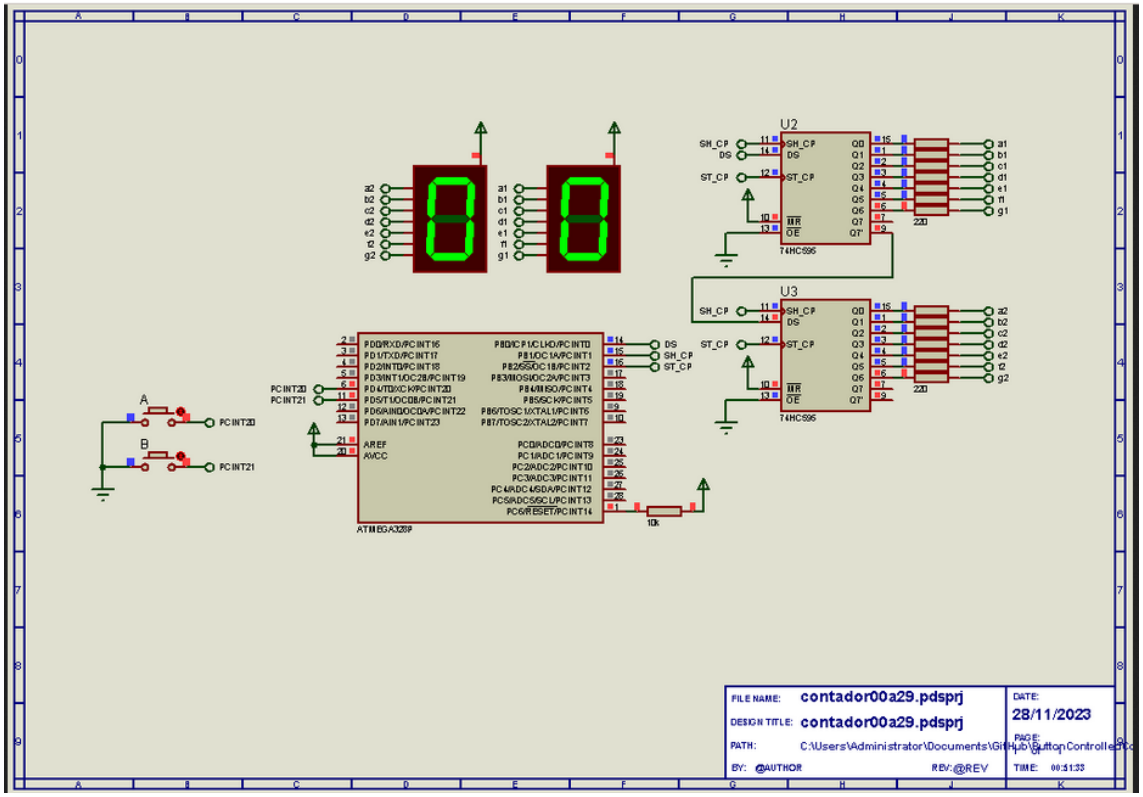

Fig. 2
ATmega328p. clique aqui para ver o gif

B. Tabela de decodificação para displays de 7 segmentos

Dígito	Anodo comum		Catodo comum	
	gfedcba		gfedcba	
0	0b1000000	0x40	0b0111111	0x3F
1	0b1111001	0x79	0b0000110	0x06
2	0b0100100	0x24	0b1011011	0x5B
3	0b0110000	0x30	0b1001111	0x4F
4	0b0011001	0x19	0b1100110	0x66
5	0b0010010	0x12	0b1101101	0x6D
6	0b0000010	0x02	0b1111101	0x7D
7	0b1111000	0x78	0b0000111	0x07
8	0b0000000	0x00	0b1111111	0x7F
9	0b0011000	0x18	0b1100111	0x67
A	0b0001000	0x08	0b1110111	0x77
B	0b0000011	0x03	0b1111100	0x7C
C	0b1000110	0x46	0b0111001	0x39
D	0b0100001	0x21	0b1011110	0x5E
E	0b0000110	0x06	0b1111001	0x79
F	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 micro controlador Atmega328 com a linguagem de programção C. O destaque incidiu na utilização do mircrocontrolador, esta pesquisa proporcionou uma visão compreensivel acerca da relevância da integração dO microcontrolador Atmega328, realçando sua aplicabilidade e contribuições para aplicações práticas em sistemas embarcados [**?**]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; PROMISE, Okere. DESIGN OF A MICRO CONTROLLER BASED TRAFFIC LIGHT CONTROLLER SYSTEM FOR CONGESTION AND EMERGENCY SITUATION.

Apêndice

A. Código em C

```c

/**
 * @file contador00a29.c
 * @brief Contador com controle de botões utilizando registradores de deslocamento.
 *
 * Este programa implementa um contador com controle de botões utilizando registradores de deslocamento.
 * Os botões A e B são utilizados para incrementar e decrementar o contador, respectivamente.
 * O valor do contador é exibido em dois displays de sete segmentos.
 * O programa utiliza interrupções para detectar as ações dos botões e atualizar o valor do contador.
 *
 * @date 24/11/2023
 * @version 1.0
 *
 */

#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#define cpl_bit(Y,bit_x) (Y^=(1<<bit_x)) //troca o estado do bit x
#define tst_bit(y,bit) (y&(1<<bit))//retorna 0 ou 1 conforme leitura do bit

// Defina os pinos para os botões A e B
#define BOTAO_A_PINO   PCINT20
#define BOTAO_B_PINO   PCINT21
// Defina os pinos para os registradores de deslocamento
#define DS_PINO    PB0   // Data
#define SH_CP_PINO PB1   // Clock
#define ST_CP_PINO PB2   // Latch
ISR(PCINT2_vect);//declara uso de PCINT2 (pinos PCINT20:21)

// Variáveis globais para a contagem e estado dos botões
volatile uint8_t contador = 0;
volatile uint8_t estadoBotaoA = 0;
volatile uint8_t estadoBotaoB = 0;

// Função para inicializar os registradores de deslocamento
void inicializarShiftRegisters() {
    // Configurar os pinos como saída
    DDRB |= (1 << DS_PINO) | (1 << SH_CP_PINO) | (1 << ST_CP_PINO);
}

// Função para enviar um byte para os registradores de deslocamento
void enviarByte(uint8_t dado) {
    for (int i = 7; i >= 0; i--) {
        // Configurar o pino de dados (DS)
        if (dado & (1 << i)) {
            PORTB |= (1 << DS_PINO);
        } else {
            PORTB &= ~(1 << DS_PINO);
        }

        // Pulsar o pino de clock (SH_CP)
        PORTB |= (1 << SH_CP_PINO);
        PORTB &= ~(1 << SH_CP_PINO);
    }

    // Pulsar o pino de latch (ST_CP)
    PORTB |= (1 << ST_CP_PINO);
    PORTB &= ~(1 << ST_CP_PINO);
}

// Mapeamento de dígitos para códigos de sete segmentos
uint8_t mapaSeteSegmentos[10] = {
    0b11000000, // 0
    0b11111001, // 1
    0b10100100, // 2
    0b10110000, // 3
    0b10011001, // 4
    0b10010010, // 5
    0b10000010, // 6
```

```
    0b11111000, // 7
    0b10000000, // 8
    0b10010000  // 9
};

// Função para atualizar os displays de sete segmentos
void atualizarDisplay() {
    // Mapear os dígitos e enviar para os registradores de deslocamento
    uint8_t digitos[2] = {mapaSeteSegmentos[contador / 10], mapaSeteSegmentos[contador % 10]};

    for (int i = 0; i < 2; i++) {
        // Enviar o código do dígito para os displays de sete segmentos
        enviarByte(digitos[i]);
    }
}

// Configurar e inicializar as interrupções
void configurarInterrupcoes() {
    // Configurar os pinos dos botões como entrada
    DDRD &= ~((1 << BOTAO_A_PINO) | (1 << BOTAO_B_PINO));
    // Ativar resistores de pull-up para os botões
    PORTD |= (1 << BOTAO_A_PINO) | (1 << BOTAO_B_PINO);

    // Configurar a interrupção para o botão A (PCINT20)
    PCICR |= (1 << PCIE2);
    PCMSK2 |= (1 << PCINT20);

    // Configurar a interrupção para o botão B (PCINT21)
    PCMSK2 |= (1 << PCINT21);

    // Ativar interrupções globais
    sei();
}

int main(void) {
    inicializarShiftRegisters();
    configurarInterrupcoes();

    while (1) {
        // Atualizar os displays de sete segmentos
        atualizarDisplay();
        _delay_ms(100);  // Adicionando um atraso de 100 ms
    }

    return 0;
}

// Rotina de interrupção para os botões A e B (PCINT20 e PCINT21)
ISR(PCINT2_vect) {
    _delay_ms(10); // Debounce
    if (!(PIND & (1 << BOTAO_A_PINO))) {
        // Incrementar o contador
        contador = (contador + 1) % 30;
    }
    if (!(PIND & (1 << BOTAO_B_PINO))) {
        // Decrementar o contador
        contador = (contador == 0) ? 29 : contador - 1;
    }
}
```

B. Equações Matemáticas

```
\appendix [Código]
%Inserir as informações referentes aos apêndices aqui.
\begin{minted}[frame=single, breaklines=true, breakanywhere=true, linenos]{c}

/**
 * @file contador00a29.c
 * @brief Contador com controle de botões utilizando registradores de deslocamento.
 *
 * Este programa implementa um contador com controle de botões utilizando registradores de deslocamento.
 * Os botões A e B são utilizados para incrementar e decrementar o contador, respectivamente.
 * O valor do contador é exibido em dois displays de sete segmentos.
 * O programa utiliza interrupções para detectar as ações dos botões e atualizar o valor do contador.
 *
 * @date 24/11/2023
 * @version 1.0
 *
 */

#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#define cpl_bit(Y,bit_x) (Y^=(1<<bit_x)) //troca o estado do bit x
#define tst_bit(y,bit) (y&(1<<bit))//retorna 0 ou 1 conforme leitura do bit
// Defina os pinos para os botões A e B
#define BOTAO_A_PINO   PCINT20
#define BOTAO_B_PINO   PCINT21
// Defina os pinos para os registradores de deslocamento
#define DS_PINO    PB0   // Data
#define SH_CP_PINO PB1   // Clock
#define ST_CP_PINO PB2   // Latch
ISR(PCINT2_vect);//declara uso de PCINT2 (pinos PCINT20:21)

// Variáveis globais para a contagem e estado dos botões
volatile uint8_t contador = 0;
volatile uint8_t estadoBotaoA = 0;
volatile uint8_t estadoBotaoB = 0;

// Função para inicializar os registradores de deslocamento
void inicializarShiftRegisters() {
    // Configurar os pinos como saída
    DDRB |= (1 << DS_PINO) | (1 << SH_CP_PINO) | (1 << ST_CP_PINO);
}

// Função para enviar um byte para os registradores de deslocamento
void enviarByte(uint8_t dado) {
    for (int i = 7; i >= 0; i--) {
        // Configurar o pino de dados (DS)
        if (dado & (1 << i)) {
            PORTB |= (1 << DS_PINO);
        } else {
            PORTB &= ~(1 << DS_PINO);
        }

        // Pulsar o pino de clock (SH_CP)
        PORTB |= (1 << SH_CP_PINO);
        PORTB &= ~(1 << SH_CP_PINO);
    }

    // Pulsar o pino de latch (ST_CP)
    PORTB |= (1 << ST_CP_PINO);
    PORTB &= ~(1 << ST_CP_PINO);
}

// Mapeamento de dígitos para códigos de sete segmentos
uint8_t mapaSeteSegmentos[10] = {
    0b11000000, // 0
    0b11111001, // 1
    0b10100100, // 2
    0b10110000, // 3
    0b10011001, // 4
    0b10010010, // 5
    0b10000010, // 6
    0b11111000, // 7
    0b10000000, // 8
    0b10010000  // 9
};

// Função para atualizar os displays de sete segmentos
void atualizarDisplay() {
    // Mapear os dígitos e enviar para os registradores de deslocamento
    uint8_t digitos[2] = {mapaSeteSegmentos[contador / 10], mapaSeteSegmentos[contador % 10]};

```

```
    for (int i = 0; i < 2; i++) {
        // Enviar o código do dígito para os displays de sete segmentos
        enviarByte(digitos[i]);
    }
}

// Configurar e inicializar as interrupções
void configurarInterrupcoes() {
    // Configurar os pinos dos botões como entrada
    DDRD &= ~((1 << BOTAO_A_PINO) | (1 << BOTAO_B_PINO));
    // Ativar resistores de pull-up para os botões
    PORTD |= (1 << BOTAO_A_PINO) | (1 << BOTAO_B_PINO);

    // Configurar a interrupção para o botão A (PCINT20)
    PCICR |= (1 << PCIE2);
    PCMSK2 |= (1 << PCINT20);

    // Configurar a interrupção para o botão B (PCINT21)
    PCMSK2 |= (1 << PCINT21);

    // Ativar interrupções globais
    sei();
}

int main(void) {
    inicializarShiftRegisters();
    configurarInterrupcoes();

    while (1) {
        // Atualizar os displays de sete segmentos
        atualizarDisplay();
        _delay_ms(100);  // Adicionando um atraso de 100 ms
    }

    return 0;
}

// Rotina de interrupção para os botões A e B (PCINT20 e PCINT21)
ISR(PCINT2_vect) {
    _delay_ms(10); // Debounce
    if (!(PIND & (1 << BOTAO_A_PINO))) {
        // Incrementar o contador
        contador = (contador + 1) % 30;
    }
    if (!(PIND & (1 << BOTAO_B_PINO))) {
        // Decrementar o contador
        contador = (contador == 0) ? 29 : contador - 1;
    }
}
```