Universidade Federal do Rio Grande do Norte

Centro de Tecnologia - CT

Departamento de Engenharia Elétrica - DEE

Laboratório de Automação, Controle e Instrumentação - LACI

Relatório Técnico 01

ELE3717 - Relatório 01

Hélio Valério Lima

Natal, 11 de maio de 2025

Como citar.

Dias, S. M. Material do curso de sistemas microcontrolados (ELE3717): **Modelo de Relatório Técnico**, UFRN, Natal, RN, 2025, 12 p.

Resumo

Segundo a ABNT, o resumo deve ressaltar o objetivo, o método, os resultados e as conclusões do documento. A ordem e a extensão destes itens dependem do tipo de resumo (informativo ou indicativo) e do tratamento que cada item recebe no documento original.

Palavras-chaves: Relatório; ABNT; Sistemas digitais; Circuitos Digitais.

Sumário

1 Introdução

Este relatório técnico aborda o desenvolvimento de firmwares em linguagem Assembly voltados para sistemas microcontrolados utilizando o microprocessador ATmega328P. O objetivo é explorar as práticas de programação de baixo nível, enfatizando controle direto de hardware e otimização de recursos. Serão descritas a estrutura dos firmwares, os procedimentos de desenvolvimento e os testes realizados. As informações utilizadas no desenvolvimento dessas soluções se encontram disponíveis no datasheet oficial do ATmega328P. Os projetos mencionados neste relatório foram desenvolvidos com o objetivo de serem utilizados no microcontrolador Arduino UNO, de modo que a sua funcionalidade possa ser validada.

2 Fundamentação Teórica

Para o projeto e implementação das atividades desenvolvidas e descritas nesse relatório foram consultados extensivamente os datasheets de arquitetura e instruções do microprocessador ATMega328P, o qual é utilizado no Arduino UNO. As atividades foram desenvolvidas utilizando a linguagem Assembly.
A linguagem Assembly é uma linguagem de baixo nível usada para programação de microcontroladores e microprocessadores. Ao contrário das linguagens de alto nível (como C ou Python), Assembly permite o controle direto do hardware, oferecendo maior precisão, desempenho e economia de recursos, aspectos fundamentais em sistemas embarcados com restrições de memória e processamento.

Assembly é composta por **mnemônicos** que representam instruções de máquina específicas da arquitetura do processador. No caso deste projeto, utilizamos o microcontrolador **AVR ATMega328P**, que possui uma arquitetura **RISC (Reduced Instruction Set Computing)**, com um conjunto de instruções eficiente e otimizado para operações em tempo real.

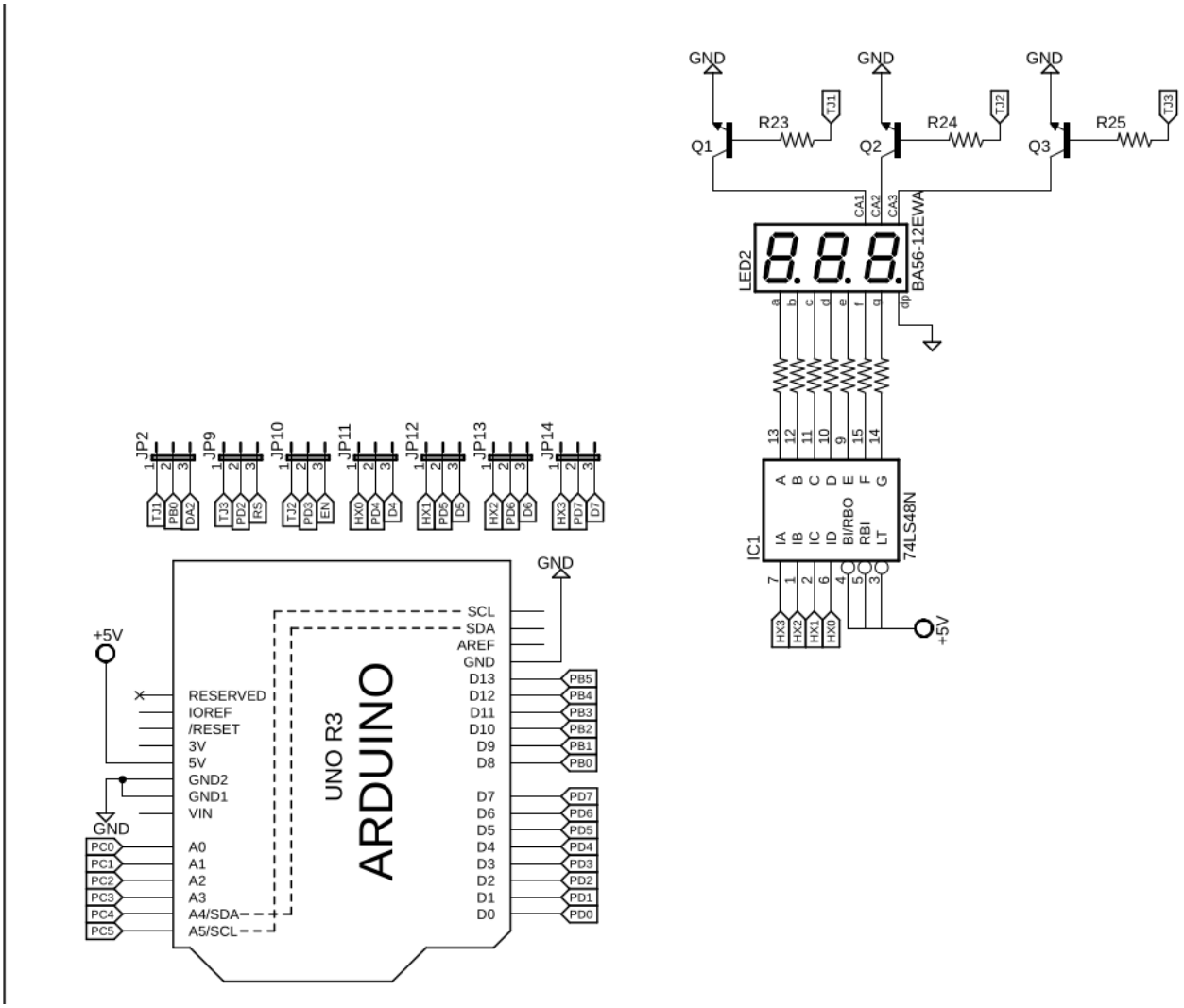
O ATMega328P possui 32 registradores de propósito geral de 8 bits, além de periféricos integrados como **timers**, **contadores**, **portas de I/O digitais**, **conversores analógico-digital (ADC)**, e **interrupções**, todos acessíveis via Assembly.

3 Materiais e Métodos

3.1 Atividade 1

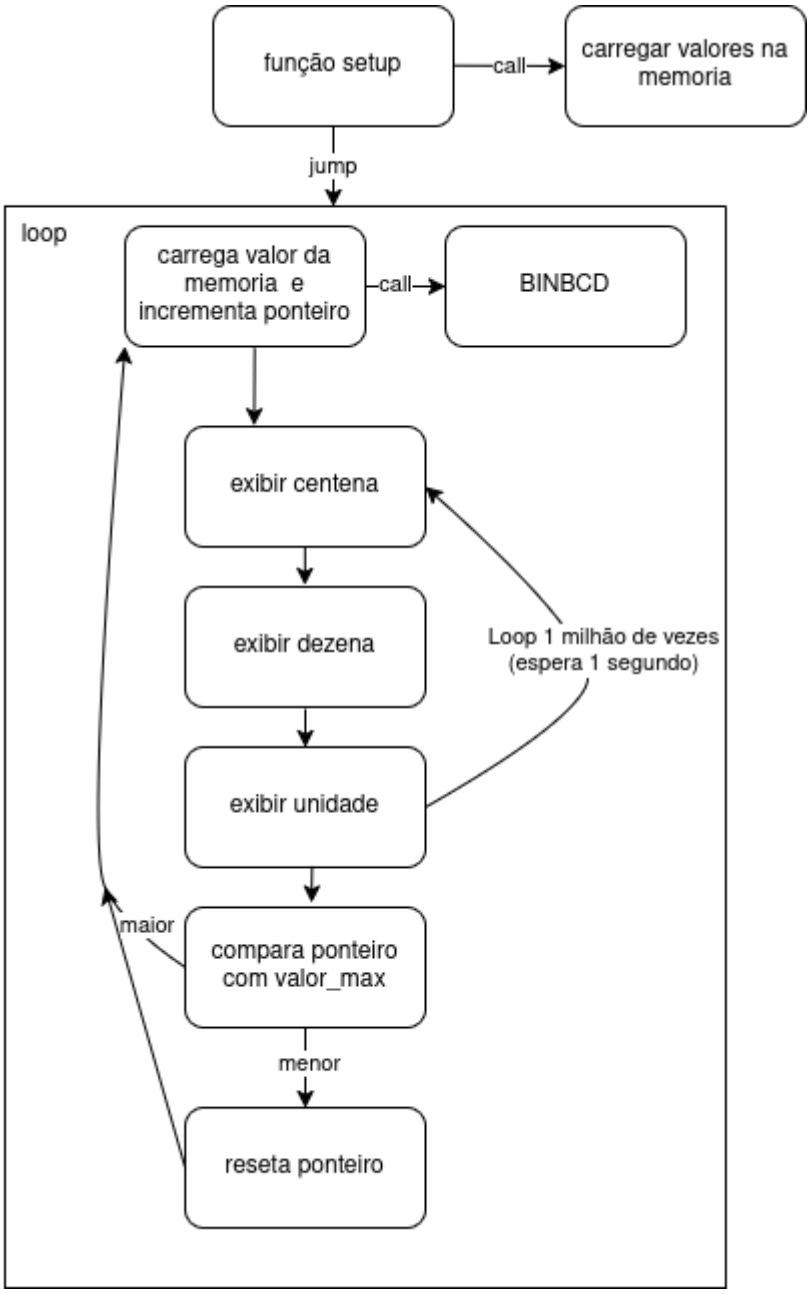
Atividade 1 consiste em desenvolver firmware para exibição de 10 valores aleatórios em 3 displays de 7 segmentos utilizando o circuito integrado 74LS48. O circuito funciona de acordo com o esquemático apresentado na figura 1, e deve ser implementado em assembly para o microprocessador ATMEGA328P. A solução desenvolvida é representada no fluxograma exibido na figura 2, cujo código-fonte e funções internas serão detalhados posteriormente.

Figura 1 - Esquemático do circuito da atividade 01.



Fonte: https: //sigaa.ufrn.br

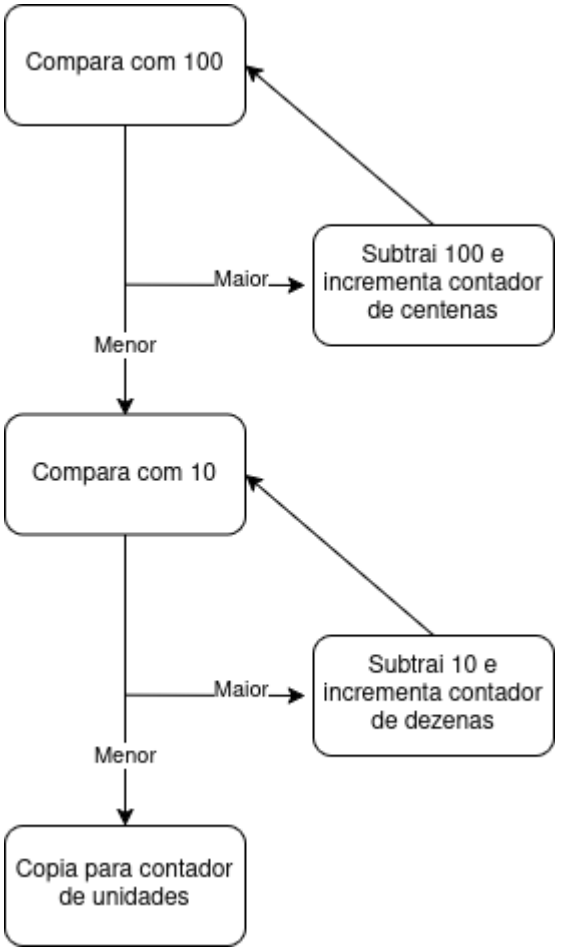
Figura 2 - Fluxograma da solução desenvolvida



Fonte: Os autores

A função de BIN-BCD funciona utilizando 5 registradores, onde 3 deles serão utilizados para armazenar os valores de centena, dezena e unidade. O funcionamento do BIN-BCD é de acordo com o fluxograma presente na figura 3, e seu código-fonte está disponível na figura 4.

Figura 3 - Fluxograma do BIN-BCD



Fonte: Os autores

Figura 4 - Código-fonte do BIN-BCD

```
;;----binbcd-----
binbcd:
mov aux, vholder
clr centena
clr dezena
clr unidade

bcdcentena:
cpi aux, 100
brlt bcddezena
```

```
subi aux, 100
inc centena
rjmp bcdcentena
bcddezena:
cpi aux, 10
brlt bcdunidade
subi aux, 10
inc dezena
rjmp bcddezena
bcdunidade:
mov unidade, aux
clr aux
clr vholder
ret
;;----binbcd-end-------
```

Fonte: O autor.

Função exibir funciona de acordo com o fluxograma apresentado na figura 5. O estado de espera é necessário para evitar sobreposição nas exibições. É necessário converter o valor BCD de entrada devido à configuração do circuito que segue uma lógica incompreensível.

Figura 5 - Fluxograma exibir

Fonte: O autor.

Figura 6 - Código-fonte exibir

```
exibircentena:
mov vholder, centena
rcall oconverter
cbr output, 0b00001100 ;;sbr se inverso
in opi, pinb
sbr opi, (1<<0);; cbr se inverso
out portb, opi
out portd, output
rcall waiting
rcall waiting
ret
```

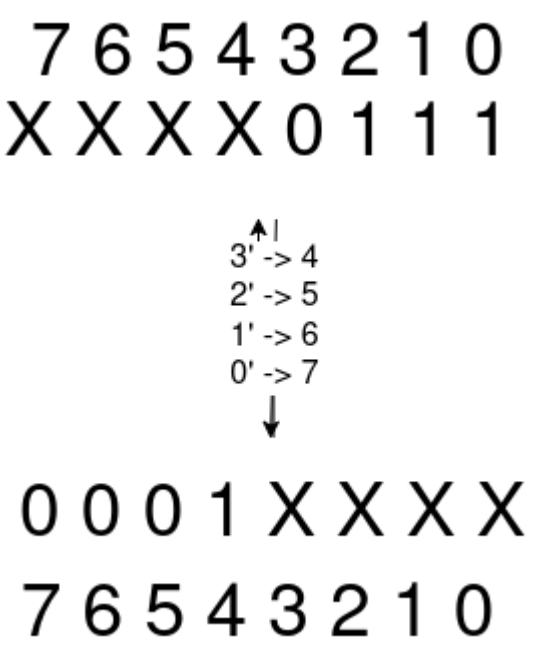
```
exibirdezena:
mov vholder,dezena
rcall oconverter
sbr output, 0b00001000 ;;cbr se inverso
cbr output, 0b00000100 ;; sbr se inverso
in opi, pinb
cbr opi, (1<<0);; sbr se inverso
out portb, opi
out portd, output
rcall waiting
rcall waiting
ret

exibirunidade:
mov vholder, unidade
rcall oconverter
cbr output, 0b00001000 ;;sbr se inverso
sbr output, 0b00000100 ;; cbr se inverso
in opi, pinb
cbr opi, (1<<0);; sbr se inverso
out portb, opi
out portd, output
rcall waiting
rcall waiting
ret
```

Fonte: O autor.

A função de conversão utilizada para inverte a ordem e o valor lógico de modo que fique de acordo com as entradas do 74LS48. O funcionamento é exemplificado na figura 7, e o código fonte utilizado para esse fim está presente na figura 8.

Figura 7 - Funcionamento do conversor



Fonte: O autor

Figura 8 - Código-fonte do conversor

```
oconverter:
;;conversao com logica inversa e ordem oposta
;; vholder[3:0] -> -aux[4:7]
;; volder[3] -> -aux[4]
;; vholder[2] -> -aux[5]
;; vholder [1] -> -aux[6]
;; vholder [0] -> -aux[7]
clr output
;; se não for logica inversa, usar sbrc
sbrc vholder, 3
```

```
sbr output, 0b00010000
sbrc vholder, 2
sbr output, 0b00100000
sbrc vholder, 1
sbr output, 0b01000000
sbrc vholder, 0
sbr output, 0b10000000
ret
;;;;----output-converter-end-------
```

Fonte: O autor.

A partir das funcionalidades desenvolvidas, é possível solucionar o problema, com base no fluxograma apresentado na figura 2. O código fonte implementado está presente na figura 9.

Figura 9 - Código-fonte da atividade 1

```
.include "m328pdef.inc"
.def opi =r16
.def centena = r21
.def dezena= r22
.def unidade= r23
.def mempointerH= r27
.def mempointerL= r26
.def vholder= r5
.def aux= r19
.def iter1= r6
.def iter2=r7
.def iter3= r8
.def output= r18
```

```
.org 0x0000
rjmp setup

loopteste1:
ser opi
out ddrd, opi
sts ddrb, opi
loopteste2:
ldi output, 0x80
ldi opi, 0x01
out portb, opi
clr opi
out portd, output
rjmp loopteste2

setup:
ser opi
out DDRD,opi
out DDRB, opi
ldi mempointerH, 0x01
ldi mempointerL, 0x01
rcall loadvalormemoria
rjmp loop

loadvalormemoria:
ldi opi, 120
sts 0x0101, opi
```

```
ldi opi, 9
sts 0x0102, opi
ldi opi, 22
sts 0x0103, opi
ldi opi, 33
sts 0x0104, opi
ldi opi, 0
sts 0x0105, opi
ldi opi,55
sts 0x0106, opi
ldi opi, 66
sts 0x0107, opi
ldi opi, 92
sts 0x0108, opi
ldi opi, 88
sts 0x0109, opi
ldi opi, 99
sts 0x010A, opi
clr opi
ret

;;----binbcd-----
binbcd:
mov aux, vholder
clr centena
clr dezena
clr unidade
```

```
bcdcentena:
cpi aux, 100
brlt bcddezena
subi aux, 100
inc centena
rjmp bcdcentena
bcddezena:
cpi aux, 10
brlt bcdunidade
subi aux, 10
inc dezena
rjmp bcddezena
bcdunidade:
mov unidade, aux
clr aux
clr vholder
ret
;;----binbcd-end-------

;;----output-converter----
oconverter:
;;conversao com logica inversa e ordem oposta
;; vholder[3:0] -> -aux[4:7]
;; volder[3] -> -aux[4]
;; vholder[2] -> -aux[5]
;; vholder [1] -> -aux[6]
```

```
;; vholder [0] -> -aux[7]
clr aux
;; se não for logica inversa, usar sbrc
sbrc vholder, 3
sbr aux, 0b00010000
sbrc vholder, 2
sbr aux, 0b00100000
sbrc vholder, 1
sbr aux, 0b01000000
sbrc vholder, 0
sbr aux, 0b10000000
mov output,aux
ret
;;;;----output-converter-end-------

;;;----MAIN-LOOP-------
loop:
ld opi, X+
mov vholder, opi
rcall binbcd
ldi opi, 10
mov iter1, opi
clr opi
switchloop1:
ldi opi, 100
mov iter2, opi
clr opi
```

```
switchloop2:
ldi opi, 100
mov iter3, opi
clr opi
switchloop3:

;;;exibir centena
exibircentena:
cpi centena, 0
breq exibirdezena
mov vholder, centena
rcall oconverter
cbr output, 0b00001100 ;;sbr se inverso
ldi opi, 0x01 ;; clr se inverso
out portb, opi
out portd, output
rcall waiting

;;exibir dezena
exibirdezena:
clr r30
add r30, centena
add r30, dezena
cpi r30, 0
breq exibirunidade
mov vholder, dezena
rcall oconverter
```

```
sbr output, 0b00001000 ;;cbr se inverso
cbr output, 0b00000100 ;; sbr se inverso
clr opi ;; ldi 1 se inverso
out portb, opi
out portd, output
rcall waiting

;;exibir
exibirunidade:
mov vholder, unidade
rcall oconverter
cbr output, 0b00001000 ;;sbr se inverso
sbr output, 0b00000100 ;; cbr se inverso
clr opi ;; ldi 1 se inverso
out portb, opi
out portd, output
rcall waiting

dec iter3
brne switchloop3
dec iter2
brne switchloop2
dec iter1
brne switchloop1

cpi mempointerL, 0x0B
brne loop
```

```
ldi mempointerL, 0x01
rjmp loop

;;------MAIN-LOOP-END----------

;;;----estado de espera--------
waiting:
ldi r17, 10
waitloop:
dec r17
breq endwait
rjmp waitloop
endwait:
ret
;;------fim estado de espera-----
```

3.2 Atividade 02

Este projeto consiste no desenvolvimento de um firmware, em linguagem Assembly, para um sistema embarcado baseado no microcontrolador AVR ATMega328P. O sistema implementa um contador inteligente com valores configuráveis pelo usuário, operando entre limites mínimo e máximo ajustáveis (de 0 a 999) e com passo de incremento/decremento variável de 1 a 15. A contagem é controlada por botões (S1 para contar, S3 para descontar) e os parâmetros são configurados por meio de um potenciômetro (P2) e confirmados por um LED RGB. A interface de saída utiliza três displays de sete segmentos com efeito POV para exibir valores de centenas, dezenas e unidades. O sistema inicia com contagem crescente a partir de 000 e pode ser alternado para modo de ajuste ou resetado a qualquer momento pelo usuário.

O funcionamento do sistema implementado é como segue:

Ao pressionar os botões S1 ou S3, o contador irá contar uma vez, crescente ou decrescente de acordo com o botão pressionado. Para que haja outra contagem, é preciso pressionar o botão novamente. Caso a tentativa de contagem crescente ultrapasse o valor máximo, o contador permanece travado no valor máximo, e partir desse ponto só é possível contar de modo decrescente até que a próxima contagem crescente esteja dentro do intervalo válido. O mesmo vale para uma tentativa de contagem decrescente que ultrapasse o valor mínimo.

Ao pressionar o botão S2 pela primeira vez, o sistema ativa o modo de configuração. Na próxima vez que o botão S2 for pressionado, o sistema entra em modo de configuração do valor mínimo. Enquanto o botão estiver pressionado, será exibido no display o valor lido na entrada analógica do potenciômetro, quando o botão for solto, esse valor será salvo na memória como o valor de limite mínimo. Apertar o botão S2 novamente leva o sistema para a configuração do valor máximo, cujo comportamento é idêntico ao do limite mínimo. Apertar o botão mais uma vez leva ao modo de configuração do step, o qual funciona de forma idêntica aos anteriores, no entanto, apenas os valores exibidos na dezena e unidade serão levados em consideração para a configuração do step. Se o valor exibido for maior que 15 no momento em que o botão for liberado, o valor do step é definido como 15. Apertar o botão S2 mais uma vez após isso retorna o sistema ao estado inicial, de modo que pressionar o botão novamente leva ao modo de configuração do valor mínimo.

O código-fonte relatado abaixo utiliza as definições presentes na figura 10 para fins de compreensão e legibilidade.

```
.include "m328pdef.inc"
;;;;defs
.def centena= r3
.def dezena= r4
.def unidade= r5
.def step= r6
.def vholder= r7
.def carry= r8

.def opi= r16
.def aux1= r17
.def aux2= r18
.def output= r19
.def controle= r20
.def loopbcd=r21
.def aux=r22
.def vholderbcd=r23
.def estado=r31
```

```
.equ addrmaxcent= 0x0120
.equ addrmaxdez=0x0121
.equ addrmaxuni= 0x0122
.equ addrmincent= 0x0123
.equ addrmindez= 0x0124
.equ addrminuni= 0x0125
.equ addrcurcent= 0x0110
.equ addrcurdez= 0x0111
.equ addrcuruni= 0x0112
.equ red=1
.equ green=2
.equ blue=3
.equ botaosetup=2
.equ botaoup=1
.equ botaodown=3
.equ PCINT1_vect=0x0008
;;;;end_defs
```

Para verificar se algum dos três botões foram pressionados, utiliza-se interrupção do tipo PCINT1, de modo que caso haja mudança de estado em algum dos botões, o sistema é interrompido. Caso não seja feita nenhuma configuração do usuário, o sistema funcionará no modo padrão contando de 0 a 999. A configuração das interrupções, do conversor analógico digital e do funcionamento padrão do sistema ocorre na função setup exibida na figura 11. Essa função é a primeira a ser executada quando o sistema é iniciado/resetado.

Figura 11 - Função Setup

```
setup:
;;configurar pinos
ldi opi, 0xFF
out DDRD, opi
```

```
out ddrb, opi
ldi opi, 0x00
out DDRC, opi

;;configurar ADC
ldi opi, (1<<REFS0)
sts ADMUX, opi
ldi opi, (1 << ADEN) | (1 << ADPS2) | (1 <<
ADPS1) | (1 << ADPS0)
sts ADCSRA, opi


;;configurar interrupts 1,2,3
ldi opi, (1<<PCIE1)
sts PCICR, opi
ldi opi, (1<<PCINT9) | (1<<PCINT10) |
(1<<PCINT11)
sts PCMSK1, opi
sei

;;configurar user_config
ldi controle, 0x01

;;configurar contador modo padrão
ldi opi, 0
mov centena, opi
mov dezena, opi
```

```
mov unidade, opi
sts addrminuni, opi
sts addrmindez, opi
sts addrmincent, opi
sts addrcurcent, opi
sts addrcurdez, opi
sts addrcuruni, opi
inc opi
mov step, opi
ldi opi, 9
sts addrmaxcent, opi
sts addrmaxdez, opi
sts addrmaxuni, opi
rjmp loop
```

Fonte: O autor

Após concluir a função setup, o sistema pula para a função loop, que consiste simplesmente em exibir o valor atual do contador enquanto aguarda por uma interrupção, conforme demonstrado na figura 12. As funções exibir são exatamente as mesmas da atividade anterior, cujo código-fonte esta presente na figura 6.

Figura 12 - Função Loop

```
loop:
lds opi, addrcurcent
mov centena, opi
lds opi, addrcurdez
mov dezena, opi
lds opi, addrcuruni
mov unidade, opi
rcall exibircentena
```

```
rcall exibirdezena
rcall exibirunidade
rjmp loop
```

Quando algum dos botões é pressionado, é acionada uma interrupção, e após verificar qual botão foi pressionado, o sistema salta para o trecho de código associado a essa funcionalidade. O tratamento da interrupção ocorre de acordo com a figura 13.

Figura 13 - Tratamento das interrupções

```
.org PCINT1_vect
rjmp interrupt_botoes

interrupt_botoes:
in opi, PINC
sbrs opi, botaodown
rjmp contardown
sbrs opi, botaoup
rjmp contarup
sbrs opi, botaosetup
rjmp int_config_cont
clr opi
reti
```

Caso pressionados os botões S1 ou S3, o sistema conta de forma crescente ou decrescente, de modo que a interrupção salta para contarup ou contardown, cujos códigos estão presentes nas figuras 14 e 15. O novo valor só é salvo se estiver contido dentro do intervalo [valor_min:valor_max], de outro modo é substituído por valor máximo ou mínimo.

Figura 14 - Código-fonte contarup

```
contarup:
ldi r29, 15
```

```
clr carry
lds opi, addrcurcent
mov centena, opi
lds opi, addrcurdez
mov dezena,opi
lds opi, addrcuruni
mov unidade, opi
ldi opi, 10
add unidade, step
compareunidade:
cp unidade, opi
brlt somardezena
inc carry
sub unidade, opi
rjmp compareunidade
somardezena:
add dezena, carry
clr carry
comparedezena:
cp dezena, opi
brlt somarcentena
inc carry
sub dezena, opi
rjmp comparedezena
somarcentena:
add centena, carry
cp centena, opi
```

```
brlt comparecommax
clr centena
comparecommax:
lds opi, addrmaxcent
cp centena, opi
brlt salvenovovalor
brne overflowcountup
lds opi, addrmaxdez
cp dezena, opi
brlt salvenovovalor
brne overflowcountup
lds opi, addrmaxuni
cp unidade, opi
brlt salvenovovalor
;;se chegou aqui estourou o valor do contador
;; vai para valor máximo
overflowcountup:
lds opi, addrmaxcent
mov centena, opi
lds opi, addrmaxdez
mov dezena, opi
lds opi, addrmaxuni
mov unidade, opi
clr opi
salvenovovalor:
mov opi, centena
sts addrcurcent, opi
```

```
mov opi, dezena
sts addrcurdez, opi
mov opi, unidade
sts addrcuruni, opi
clr opi
reti
```

Figura 15 - Código-fonte contardown

```
contardown:
clr carry
lds opi, addrcurcent
mov centena, opi
lds opi, addrcurdez
mov dezena,opi
lds opi, addrcuruni
mov unidade, opi
ldi opi, 0
cp centena, opi
brne continue_contar_down
cp dezena, opi
brne continue_contar_down
cp unidade, opi
brne continue_contar_down
rjmp comparecommin

continue_contar_down:
ldi opi, 10
```

```
compareunidadesubtracao:
cp unidade, step
brge subunidade
inc carry
add unidade, opi
rjmp compareunidadesubtracao
subunidade:
sub unidade,step
mov aux1, carry
clr carry
comparedezenasubtracao:
cp dezena, aux1
brge subdezena
inc carry
add dezena, opi
rjmp comparedezenasubtracao
subdezena:
sub dezena, aux1
clr aux1
comparecentenasubtracao:
cp centena,carry
brge subcentena
mov centena, carry
subcentena:
sub centena, carry

comparecommin:
```

```
lds opi, addrmincent
cp centena, opi
breq comparemindez
brge salvenovovalor
comparemindez:
lds opi, addrmindez
cp dezena, opi
breq compareminuni
brge salvenovovalor
compareminuni:
lds opi, addrminuni
cp unidade, opi
brge salvenovovalor
;;underflow sub
mov unidade, opi
lds opi,addrmindez
mov dezena, opi
lds opi,addrmincent
mov centena, opi
clr opi
rjmp salvenovovalor
```

Se pressionado o botão S2, o sistema salta para o modo de configuração. No modo de configuração é utilizado o registrador controle para determinar em que estado de configuração o sistema se encontra, e saltar para o modo respectivo. Ao final de cada configuração o bit de controle é deslocado para a esquerda passando o sistema para o próximo estado.

Figura 16 - Controle de estado de configuração do usuário

```
int_config_cont:
```

```
;;acende o led
;;controle: bit 1, ajuste minimo; bit 2,
ajuste máximo, bit 3, ajuste step

cpi controle, (1<<1)
breq setup_lower_bound
cpi controle, (1<<2)
breq setup_upper_bound
cpi controle, (1<<3)
breq setup_step_intermediario
cpi controle, (1<<0)
breq end_interrupt_setup
ldi controle, 0x01
rjmp end_interrupt_setup

end_interrupt_setup:
lsl controle
reti
```

Figura 17- Configuração limite inferior

```
setup_lower_bound:
;;muda cor do led
in opi, pinb
cbr opi, (1<<blue)
sbr opi, (1<<red)
cbr opi, (1<<green)
out portb, opi
```

```
;;pega valor do adc
rcall ler_adc
;;chama binbcd
rcall bin_bcd_10bits
;;carrega o valor nos regs
rcall exibircentena
rcall exibirdezena
rcall exibirunidade
;;checa se botão foi apertado
;;se sim pula para end_setup_lower_bound
in aux1, PINC
sbrc aux1, botaosetup ;; ou sbrs
rjmp end_setup_lower_bound
rjmp setup_lower_bound
```

```
end_setup_lower_bound:
clr aux1
mov opi, centena
sts addrmincent, opi
mov opi, dezena
sts addrmindez, opi
mov opi, unidade
sts addrminuni, opi
rjmp end_interrupt_setup
```

Figura 18 - Configuração limite superior

```
setup_upper_bound:
;;muda cor do led
in opi, pinb
cbr opi, (1<<blue)
cbr opi, (1<<red)
sbr opi, (1<<green)
out portb, opi
;;pega valor do adc
rcall ler_adc
;;chama binbcd
rcall bin_bcd_10bits
;;carrega o valor nos regs
rcall exibircentena
rcall exibirdezena
rcall exibirunidade
;;checa se botão foi apertado
;;se sim pula para end_setup_upper_bound
in aux1, PINC
sbrc aux1, botaosetup ;; ou sbrs
rjmp end_setup_upper_bound
rjmp setup_upper_bound

end_setup_upper_bound:
clr aux1
mov opi, centena
sts addrmaxcent, opi
mov opi, dezena
```

```
sts addrmaxdez, opi
mov opi, unidade
sts addrmaxuni, opi
rjmp end_interrupt_setup
```

Para o step foi necessário criar uma etiqueta intermediária devido a limitações de salto da instrução breq.

Figura 19 - Configuração step

```
setup_step_intermediario:
rjmp setup_step
```

```
setup_step:
;;muda cor do led
in opi, pinb
cbr opi, (1<<blue)
cbr opi, (1<<red)
sbr opi, (1<<green)
out portb, opi
;;pega valor do adc
rcall ler_adc
;;chama binbcd
rcall bin_bcd_10bits
;;carrega o valor nos regs
rcall exibircentena
rcall exibirdezena
rcall exibirunidade
;;checa se botão foi apertado
```

```
;;se sim pula para end_setup_step
in aux1, PINC
sbrc aux1, botaosetup ;; ou sbrs
rjmp end_setup_step_p1
rjmp setup_step

end_setup_step_p1:
mov opi, dezena
cpi opi, 0
breq skip_add_10_setup
ldi opi, 10
skip_add_10_setup:
add opi, unidade
cpi opi, 16
brlo end_setup_step_p2
ldi opi, 15
end_setup_step_p2:
mov step, opi
rjmp end_interrupt_setup
```

A função ler_adc ativa a conversão e espera até que ela esteja concluída, salvando os valores lidos no registrador X.

Figura 20 - Ler_ADC

```
ler_adc:
lds aux2, ADCSRA
sbr aux2, (1<<ADSC)
sts ADCSRA, aux2
```

```
wait_conversao_ADC:
lds aux2, ADCSRA
sbrs aux2, ADIF
rjmp wait_conversao_ADC
lds XL, ADCL
lds XH, ADCH
lds aux2, ADCSRA
sbr aux2, (1<<ADIF)
sts ADCSRA, aux2
ret
```

Foi necessário um ajuste no BINBCD utilizado na atividade 1, de modo que ele pudesse comportar operações com 10 bits, que é a resolução do ADC. Binbcd_old é a mesma função exibida na figura 4, apenas ajustando o registrador de entrada para XL.

Figura 21 - BIN_BCD

```
bin_bcd_10bits:
clr centena
clr dezena
clr unidade
bin_bcd_reverse:
cpi XH, 0x00
breq binbcd_old
sbiw XH:XL, 50
sbiw XH:XL, 50
inc centena
rjmp bin_bcd_reverse
```

A partir do código desenvolvido é possível implementar um contador inteligente e configurável, que segue os requisitos da atividade. O código fonte completo do código desenvolvido está presente na figura 22.

Figura 22 - Código-fonte atividade 2

```
.include "m328pdef.inc"
;;;;defs
.def centena= r3
.def dezena= r4
.def unidade= r5
.def step= r6
.def vholder= r7
.def carry= r8

.def opi= r16
.def aux1= r17
.def aux2= r18
.def output= r19
.def controle= r20
.def loopbcd=r21
.def aux=r22
.def vholderbcd=r23
.def estado=r31
.equ addrmaxcent= 0x0120
.equ addrmaxdez=0x0121
.equ addrmaxuni= 0x0122
.equ addrmincent= 0x0123
.equ addrmindez= 0x0124
.equ addrminuni= 0x0125
```

```
.equ addrcurcent= 0x0110
.equ addrcurdez= 0x0111
.equ addrcuruni= 0x0112
.equ red=1
.equ green=2
.equ blue=3
.equ botaosetup=2
.equ botaoup=1
.equ botaodown=3
.equ PCINT1_vect=0x0008
;;;;end_defs

.org 0x0000
rjmp setup

.org PCINT1_vect
rjmp interrupt_botoes


setup:
;;configurar pinos
ldi opi, 0xFF
out DDRD, opi
out ddrb, opi
ldi opi, 0x00
out DDRC, opi
```

```
;;configurar ADC
ldi opi, (1<<REFS0)
sts ADMUX, opi
ldi opi, (1 << ADEN) | (1 << ADPS2) | (1 <<
ADPS1) | (1 << ADPS0)
sts ADCSRA, opi


;;configurar interrupts 1,2,3
ldi opi, (1<<PCIE1)
sts PCICR, opi
ldi opi, (1<<PCINT9) | (1<<PCINT10) |
(1<<PCINT11)
sts PCMSK1, opi
sei

;;configurar user_config
ldi controle, 0x01

;;configurar contador modo padrão
ldi opi, 0
mov centena, opi
mov dezena, opi
mov unidade, opi
sts addrminuni, opi
sts addrmindez, opi
sts addrmincent, opi
```

```
sts addrcurcent, opi
sts addrcurdez, opi
sts addrcuruni, opi
inc opi
mov step, opi
ldi opi, 9
sts addrmaxcent, opi
sts addrmaxdez, opi
sts addrmaxuni, opi


rjmp loop

interrupt_botoes:
in opi, PINC
sbrs opi, botaodown
rjmp contardown
sbrs opi, botaoup
rjmp contarup
sbrs opi, botaosetup
rjmp int_config_cont
clr opi
reti

contarup:
ldi r29, 15
clr carry
```

```
lds opi, addrcurcent
mov centena, opi
lds opi, addrcurdez
mov dezena,opi
lds opi, addrcuruni
mov unidade, opi
ldi opi, 10
add unidade, step
compareunidade:
cp unidade, opi
brlt somardezena
inc carry
sub unidade, opi
rjmp compareunidade
somardezena:
add dezena, carry
clr carry
comparedezena:
cp dezena, opi
brlt somarcentena
inc carry
sub dezena, opi
rjmp comparedezena
somarcentena:
add centena, carry
cp centena, opi
brlt comparecommax
```

```
clr centena
comparecommax:
lds opi, addrmaxcent
cp centena, opi
brlt salvenovovalor
brne overflowcountup
lds opi, addrmaxdez
cp dezena, opi
brlt salvenovovalor
brne overflowcountup
lds opi, addrmaxuni
cp unidade, opi
brlt salvenovovalor
;;se chegou aqui estourou o valor do contador
;; vai para valor máximo
overflowcountup:
lds opi, addrmaxcent
mov centena, opi
lds opi, addrmaxdez
mov dezena, opi
lds opi, addrmaxuni
mov unidade, opi
clr opi
salvenovovalor:
mov opi, centena
sts addrcurcent, opi
mov opi, dezena
```

```
sts addrcurdez, opi
mov opi, unidade
sts addrcuruni, opi
clr opi
reti


contardown:
clr carry
lds opi, addrcurcent
mov centena, opi
lds opi, addrcurdez
mov dezena,opi
lds opi, addrcuruni
mov unidade, opi
ldi opi, 0
cp centena, opi
brne continue_contar_down
cp dezena, opi
brne continue_contar_down
cp unidade, opi
brne continue_contar_down
rjmp comparecommin

continue_contar_down:
ldi opi, 10
compareunidadesubtracao:
```

```
cp unidade, step
brge subunidade
inc carry
add unidade, opi
rjmp compareunidadesubtracao
subunidade:
sub unidade,step
mov aux1, carry
clr carry
comparedezenasubtracao:
cp dezena, aux1
brge subdezena
inc carry
add dezena, opi
rjmp comparedezenasubtracao
subdezena:
sub dezena, aux1
clr aux1
comparecentenasubtracao:
cp centena,carry
brge subcentena
mov centena, carry
subcentena:
sub centena, carry

comparecommin:
lds opi, addrmincent
```

```
cp centena, opi
breq comparemindez
brge salvenovovalor
comparemindez:
lds opi, addrmindez
cp dezena, opi
breq compareminuni
brge salvenovovalor
compareminuni:
lds opi, addrminuni
cp unidade, opi
brge salvenovovalor
;;underflow sub
mov unidade, opi
lds opi,addrmindez
mov dezena, opi
lds opi,addrmincent
mov centena, opi
clr opi
rjmp salvenovovalor



loop:
lds opi, addrcurcent
mov centena, opi
```

```
lds opi, addrcurdez
mov dezena, opi
lds opi, addrcuruni
mov unidade, opi


rcall exibircentena
rcall exibirdezena
rcall exibirunidade
rjmp loop

waiting:
ldi aux2, 10
wait_loop:
dec aux2
breq end_wait
rjmp wait_loop
end_wait:
ret


oconverter:
;;conversao com logica inversa e ordem oposta
;; vholder[3:0] -> -aux[4:7]
;; volder[3] -> -aux[4]
;; vholder[2] -> -aux[5]
;; vholder [1] -> -aux[6]
```

```
;; vholder [0] -> -aux[7]
clr output
;; se não for logica inversa, usar sbrc
sbrc vholder, 3
sbr output, 0b00010000
sbrc vholder, 2
sbr output, 0b00100000
sbrc vholder, 1
sbr output, 0b01000000
sbrc vholder, 0
sbr output, 0b10000000
ret
;;;;----output-converter-end-------
exibircentena:
mov vholder, centena
rcall oconverter
cbr output, 0b00001100 ;;sbr se inverso
in opi, pinb
sbr opi, (1<<0);; cbr se inverso
out portb, opi
out portd, output
rcall waiting
rcall waiting
ret

exibirdezena:
mov vholder,dezena
```

```
rcall oconverter
sbr output, 0b00001000 ;;cbr se inverso
cbr output, 0b00000100 ;; sbr se inverso
in opi, pinb
cbr opi, (1<<0);; sbr se inverso
out portb, opi
out portd, output
rcall waiting
rcall waiting
ret

exibirunidade:
mov vholder, unidade
rcall oconverter
cbr output, 0b00001000 ;;sbr se inverso
sbr output, 0b00000100 ;; cbr se inverso
in opi, pinb
cbr opi, (1<<0);; sbr se inverso
out portb, opi
out portd, output
rcall waiting
rcall waiting
ret


int_config_cont:
;;acende o led
```

```
;;controle: bit 1, ajuste minimo; bit 2,
ajuste máximo, bit 3, ajuste step

cpi controle, (1<<1)
breq setup_lower_bound
cpi controle, (1<<2)
breq setup_upper_bound
cpi controle, (1<<3)
breq setup_step_intermediario
cpi controle, (1<<0)
breq end_interrupt_setup
ldi controle, 0x01
rjmp end_interrupt_setup

setup_step_intermediario:
rjmp setup_step
setup_lower_bound:
;;muda cor do led
in opi, pinb
cbr opi, (1<<blue)
sbr opi, (1<<red)
cbr opi, (1<<green)
out portb, opi
;;pega valor do adc
rcall ler_adc
;;chama binbcd
rcall bin_bcd_10bits
```

```
;;carrega o valor nos regs
rcall exibircentena
rcall exibirdezena
rcall exibirunidade
;;checa se botão foi apertado
;;se sim pula para end_setup_lower_bound
in aux1, PINC
sbrc aux1, botaosetup ;; ou sbrs
rjmp end_setup_lower_bound
rjmp setup_lower_bound

end_setup_lower_bound:
clr aux1
mov opi, centena
sts addrmincent, opi
mov opi, dezena
sts addrmindez, opi
mov opi, unidade
sts addrminuni, opi
rjmp end_interrupt_setup



end_interrupt_setup:
lsl controle
reti
```

```
setup_upper_bound:
;;muda cor do led
in opi, pinb
cbr opi, (1<<blue)
cbr opi, (1<<red)
sbr opi, (1<<green)
out portb, opi
;;pega valor do adc
rcall ler_adc
;;chama binbcd
rcall bin_bcd_10bits
;;carrega o valor nos regs
rcall exibircentena
rcall exibirdezena
rcall exibirunidade
;;checa se botão foi apertado
;;se sim pula para end_setup_upper_bound
in aux1, PINC
sbrc aux1, botaosetup ;; ou sbrs
rjmp end_setup_upper_bound
rjmp setup_upper_bound

end_setup_upper_bound:
clr aux1
mov opi, centena
sts addrmaxcent, opi
mov opi, dezena
```

```
sts addrmaxdez, opi
mov opi, unidade
sts addrmaxuni, opi
rjmp end_interrupt_setup

setup_step:
;;muda cor do led
in opi, pinb
cbr opi, (1<<blue)
cbr opi, (1<<red)
sbr opi, (1<<green)
out portb, opi
;;pega valor do adc
rcall ler_adc
;;chama binbcd
rcall bin_bcd_10bits
;;carrega o valor nos regs
rcall exibircentena
rcall exibirdezena
rcall exibirunidade
;;checa se botão foi apertado
;;se sim pula para end_setup_step
in aux1, PINC
sbrc aux1, botaosetup ;; ou sbrs
rjmp end_setup_step_p1
rjmp setup_step
```

```
end_setup_step_p1:
mov opi, dezena
cpi opi, 0
breq skip_add_10_setup
ldi opi, 10
skip_add_10_setup:
add opi, unidade
cpi opi, 16
brlo end_setup_step_p2
ldi opi, 15
end_setup_step_p2:
mov step, opi
rjmp end_interrupt_setup

;;----binbcd-----
bin_bcd_10bits:
clr centena
clr dezena
clr unidade
bin_bcd_reverse:
cpi XH, 0x00
breq binbcd_old
sbiw XH:XL, 50
sbiw XH:XL, 50
inc centena
rjmp bin_bcd_reverse
```

```
binbcd_old:
mov aux, XL


bcdcentena:
cpi aux, 100
brlo bcddezena
subi aux, 100
inc centena
rjmp bcdcentena
bcddezena:
cpi aux, 10
brlo bcdunidade
subi aux, 10
inc dezena
rjmp bcddezena
bcdunidade:
mov unidade, aux

ret
;;----binbcd-end-------


ler_adc:
lds aux2, ADCSRA
sbr aux2, (1<<ADSC)
sts ADCSRA, aux2
```

```
wait_conversao_ADC:
lds aux2, ADCSRA
sbrs aux2, ADIF
rjmp wait_conversao_ADC
lds XL, ADCL
lds XH, ADCH
lds aux2, ADCSRA
sbr aux2, (1<<ADIF)
sts ADCSRA, aux2
ret


superwaiting:
ldi r28, 0xFF
waitinglp1:
rcall waiting
dec r28
breq end_super_waiting
rjmp waitinglp1
end_super_waiting:
ret
```

3.3 Atividade 03

A atividade 3 consiste no desenvolvimento de um firmware para implementar um medidor de distâncias do tipo HC_SR04, e da capacidade de salvar esses valores medidos em memória não-volátil (EEPROM).

A solução projetada consiste em utilizar os contadores de 16 bits do ATMega328P para medir o tempo em que o pino referente à ECHO do sensor se mantém em nível alto, e depois processar o valor de modo a convertê-lo para distância em cm. O cálculo da distância em cm é feito a partir da determinação de quantos pulsos de clock são equivalentes a uma distância percorrida de 1 cm. A partir disso é possível determinar a distância a partir da divisão do valor medido pelo contador por esse número.

Figura 23 - Código-fonte atividade 03

```
.include "m328pdef.inc"
;;defs
.def centena= r3
.def dezena= r4
.def unidade= r5
.def step= r6
.def vholder= r7
.def carry= r8
.def valor_lido= r9
.def opi= r16
.def aux1= r17
.def aux2= r18
.def output= r19
.def controle= r20

.equ pulsos1cm=50 ;; quantidade de pulsos de
clock equivalentes a 1cm
.equ PRE_SCALE_=2
.equ pinotrigger=4
.equ pinoecho=5 ;;PC5
.equ botaosalvar=2 ;;s2
.equ botaomedir=1 ;;s3
.equ botaoler=3  ;;s1
```

```
.equ PCINT1_vect=0x0008
.equ contadorH=TCNT1H
.equ contadorL=TCNT1L
.equ addrEEPROMH=0x00
.equ addrEEPROML=0x35
.equ addrcurcent = 0x0101
.equ addrcurdez = 0x0102
.equ addrcuruni = 0x0103

;defs

.org 0x0000
rjmp setup

.org PCINT1_vect
rjmp interrupt_botoes



setup:

;;configurar pinos
ldi opi, 0xFF
out DDRD, opi
out ddrb, opi
ldi opi, (1<<pinotrigger)
out DDRC, opi
```

```
;;configurar interrupts
ldi opi, (1<<PCIE1)
sts PCICR, opi
ldi opi, (1<<PCINT9) | (1<<PCINT10) |
(1<<PCINT11) | (1<<PCINT12)
sts PCMSK1, opi
sei

;;configurar contador
ldi opi, 0x00
sts TCCR1A, opi
sts TIMSK1, opi
sts TCCR1C, opi
ldi opi,(1<<PRE_SCALE_)
sts TCCR1B, opi

;;;configura EEPROM
ldi opi, addrEEPROMH
sts EEARH, opi
ldi opi, addrEEPROML
sts EEARL, opi
ldi opi, 0x04
sts EECR, opi

rjmp loop
```

```
interrupt_botoes:
in opi, PINC
sbrs opi, botaoler
rcall ler_eeprom
sbrs opi, botaosalvar
rcall escrever_eeprom
sbrs opi, botaomedir
rcall medir_valor
sbrs opi, botaoler
rjmp end_interrupt
sbrs opi, botaosalvar
rjmp end_interrupt
sbrs opi, botaomedeir
rjmp end_interrupt
rcall contar_xs
end_interrupt:
reti

escrever_eeprom:
mov opi, valor_lido
sts EEDR, opi
ldi opi, (1<<2)
sts EECR, opi
nop
ori opi, (1<<1)
sts EECR, opi
```

```
nop
clr opi
ret
ler_eeprom:
ldi opi, (1<<0)
sts EECR, opi
nop
lds opi, EEDR
mov valor_lido, opi
clr opi
ret

medir_valor:
ldi opi, (1<<pinotrigger)
out portc, opi
rcall wait_10
andi opi, ~(1<<pinotrigger)
ret

contar_xs:
;;primeira interrupção (borda de subida)
controle está 0
;; reseta o contador e termina
clr opi
sbrc controle, 0
rjmp skip_reset_counter
sts contadorH, opi
```

```
sts contadorL, opi
ldi controle, 1
rjmp end_contar_xs
skip_reset_counter: ;;segunda interrupção
(borda de descida) controle 1
;; armazena o valor do contador em X e reseta
controle
lds XH, contadorH
lds XL, contadorL
ldi controle, 0
rcall cvdist

end_contar_xs:
ret

cvdist:
clr valor_lido
cp valor_lido, 0
loop_cvdist:
brlt end_cvdist
sbiw XH:XL, pulsos1cm
inc valor_lido
rjmp loop_cvdist
end_cvdist: ;;valor lido ja convertido para
centimetros
ret
```

```
wait_10:
ldi r30, 10
loop_wait_10:
dec r30
breq end_wait_10
rjmp loop_wait_10
end_wait_10:
ret

loop:
lds opi, addrcurcent
mov centena, opi
lds opi, addrcurdez
mov dezena, opi
lds opi, addrcuruni
mov unidade, opi


rcall exibircentena
rcall exibirdezena
rcall exibirunidade
rjmp loop

;;----binbcd-----
bin_bcd_10bits:
clr centena
clr dezena
```

```
clr unidade
bin_bcd_reverse:
cpi XH, 0x00
breq binbcd_old
sbiw XH:XL, 50
sbiw XH:XL, 50
inc centena
rjmp bin_bcd_reverse

binbcd_old:
mov aux, XL


bcdcentena:
cpi aux, 100
brlo bcddezena
subi aux, 100
inc centena
rjmp bcdcentena
bcddezena:
cpi aux, 10
brlo bcdunidade
subi aux, 10
inc dezena
rjmp bcddezena
bcdunidade:
mov unidade, aux
```

```
ret
;;----binbcd-end-------
```

3.2 Resultados

Atividade 01: A atividade foi desenvolvida e entregue na monitoria da disciplina.

Atividade 02: A atividade foi desenvolvida, mas não foi entregue de acordo com as especificações desejadas.. No entanto, a sua funcionalidade é explicada em sua respectiva seção.

Atividade 03: Não foi possível concluir a atividade. A seção atividade 03, explica o projeto e o esforço de solução.

4 Conclusão

Os firmwares desenvolvidos para os três sistemas embarcados baseados no microcontrolador **ATMega328P** demonstram, de forma eficiente, a aplicação de conceitos fundamentais de programação em Assembly, controle de periféricos e manipulação de interfaces homem-máquina em sistemas embarcados.

No primeiro projeto, foi implementado um mecanismo de exibição rotativa de valores inteiros utilizando displays de sete segmentos e o efeito de *Persistence of Vision*, com atualização temporal precisa a cada segundo. No segundo sistema, um contador inteligente permitiu contagem crescente e decrescente com controle dinâmico dos limites e do passo de incremento, destacando a integração entre leitura analógica (potenciômetro), botões digitais e feedback visual por LED RGB. Já o terceiro projeto apresentou uma solução de medição de distância com o sensor ultrassônico HC-SR04, com armazenamento não volátil da medida em EEPROM e visualização em displays, reforçando o uso de temporização de precisão e interface com sensores externos.

Todos os sistemas foram projetados para operação estável, com suporte a reset em qualquer momento, comportamento intuitivo para o usuário e implementação eficiente da lógica de controle. As soluções atenderam integralmente aos requisitos funcionais propostos, evidenciando a viabilidade e robustez do uso do ATMega328P em aplicações embarcadas de baixo custo e alta confiabilidade.

Referências

TOCCI, R.; WIDMER, N.; MOSS, G. *Digital Systems: Principles and Applications*. [S.l.]: Pearson Education Limited, 2011. ISBN 9780130387936.

VAHID, F. *Sistemas Digitais*. [S.l.]: Grupo A - Bookman, 2009. ISBN 9788577802371.