Universidade Federal do Rio Grande do Norte

Departamento de Engenharia de Computação e Automação
DCA0119 - SISTEMAS DIGITAIS

RELATÓRIO DO 1º PROJETO DE UNIDADE

Sistema embarcado em MCU para o controle de um secador de grãos

Integrantes:
Felipe Oliveira Lins E Silva - 20180154889
Emerson Wendlingger Dantas Sales - 20180154851
Luís Gabriel Pereira Condados - 2015093091
Luís Henrique Matias Viana - 20180155026
Professor orientador: Sérgio Natan

Natal-RN
2018

Universidade Federal do Rio Grande do Norte

Departamento de Engenharia de Computação e Automação

DCA0119 - Sistemas Digitais

RELATÓRIO DO 1º PROJETO DE UNIDADE

Relatório apresentado à disciplina de Sistemas Digitais, correspondente a 1º unidade do semestre 2018.2 do 7º período do curso de Engenharia de Computação e Automação da Universidade Federal do Rio Grande do Norte, sob orientação do **Prof. Sérgio Natan Silva**.

Natal-RN

2018

Conteúdo

1 INTRODUÇÃO **1**

2 ESPECIFICAÇÕES **2**

3 ESQUEMÁTICO E MONTAGEM DO CIRCUITO **3**

4 Implementação **4**
4.1 Módulo das definições de contantes/variáveis globais 4
4.2 Módulo PWM 6
4.3 Módulo USART 7
4.4 Módulo de leitura dos sensores 8
4.5 Módulo das interrupções 10
4.6 Módulo do programa principal 11

5 Resultados **14**

6 Link para o vídeo **16**

1 INTRODUÇÃO

Este projeto consiste na implementação de um secador de grãos utilizando um microcontrolador Atmega328p[1]. Este, recebe dois sinais analógicos de entrada - respectivamente de temperatura e umidade -, em sequência, assim que a chave CH é apertada, inicia o processo de secagem. LEDS serão acesos, conforme as especificações, durante o processo. O microcontrolador emite um sinal $x(t)$ PWM, que vai para um optoacoplador - o qual conecta o ATMEGA 328p ao circuito do motor, controlando-o.

O sinal $x(t)$ obedece a esta formula:

$$x(t) = \alpha \cdot z(t) \cdot U(t) + \beta \cdot V(t)$$

Onde, α e β são constantes; $z(t)$ corresponde ao formato da curva a qual queremos que modele o comportamento do motor; $U(t)$ e $V(t)$ são os valores obtidos, respectivamente, pelos sensores de umidade e temperatura.

2 ESPECIFICAÇÕES

1. O sistema deverá ter dois sinais analógicos de entrada: um que tem como origem o sensor de umidade ($U(t)$), e outro que vem do sensor de temperatura ($V(t)$).

2. O sistema deve possuir uma chave CH1 (HiZ e terra), a qual, quando em terra, dá início ao processo de secagem.

3. O sistema deve ter um LED (LED1) que tenha sua luminosidade proporcional a $x(t)$ - o qual é o sinal de saída, que controlará o motor.

4. O sistema deve ter um LED (LED2) o qual a luminosidade será proporcional a $V(t)$ ou $U(t)$ - situação do secador.

5. O sinal emitido pelo o μC deverá está em uma frequência entre 100 Hz e 100 kHz.

3 ESQUEMÁTICO E MONTAGEM DO CIRCUITO

Segue abaixo o esquemático do projeto e a montagem:

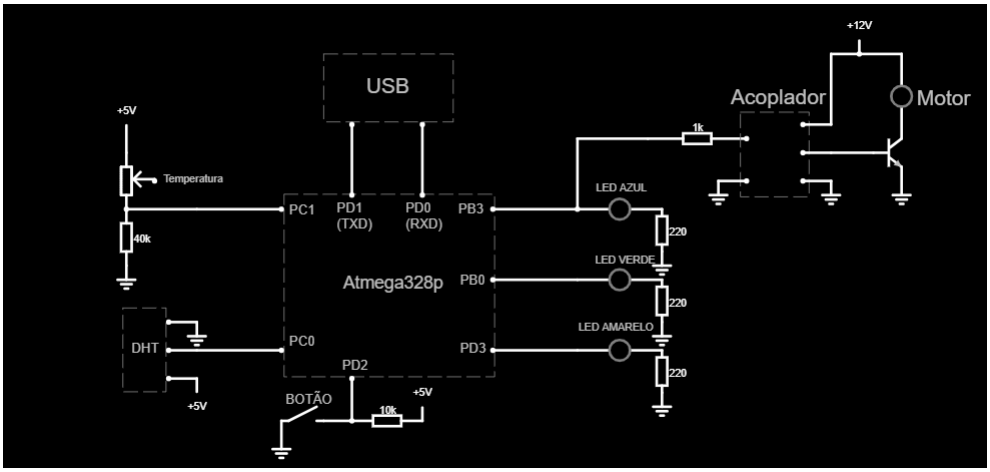

Figura 1: Esquemático do projeto

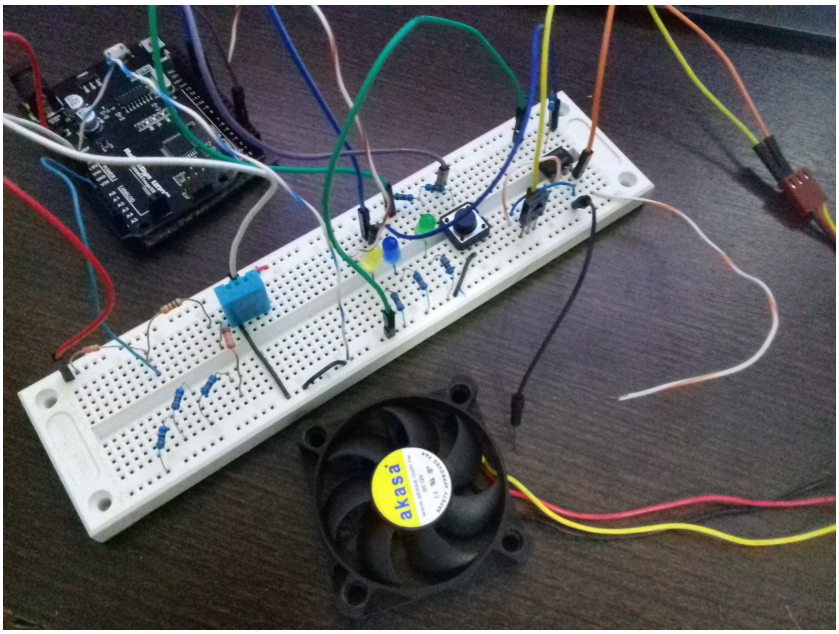

Figura 2: Montagem do circuito

4 Implementação

Na elaboração do código, optamos por dividi-lo em módulos - para que o desenvolvimento colaborativo seja o mais proveitoso possível. Os módulos são: definição de constantes/variáveis globais; PWM; USART; leitura de sensores; interrupções; programa principal.

4.1 Módulo das definições de contantes/variáveis globais

Este módulo tem como o objetivo inicializar variáveis voláteis, para utilizar nas interrupções, e defines de labels para facilitar a comprensão/desenvolvimento do código.

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

#define F_CPU 16000000UL // 16 MHz, clock_ms externo, define
    necessario para usar delay corretamente
#define OVF_T0_1_ms (F_CPU/(1000UL*256))
#define BAUDRATE 9600         //taxa de transmissao em bps
#define BAUD_PRESCALLER (((F_CPU / (BAUDRATE * 16UL))) - 1)
// DHT no pino PC0        => A0
#define DHT_PIN PC0
#define THERMISTOR PC1    //A1

#define LED_YELLOW_D PD3
#define LED_BLUE_B PB3
#define BUTTON_D PD2
#define LED_GREEN_B PB0
#define MOTOR LED_BLUE_B //A saida para do LED e do MOTOR eh a
    mesma

#define BETA 1/50.0
#define ALFA 1/90.0
// Motor no pino  PB3 (pino 11 do arduino)
// Led com PWM no pino PD3 (pino 3 do arduino)
enum {
  INIT = 0,
  CTRL = 1,
  STOP = 2
};
// Variaveis globais
volatile const uint8_t ovf_T0_1ms = OVF_T0_1_ms;
volatile uint8_t count_ovf_T0 = 0;
volatile uint64_t timer_ms = 0;
```

```
32 volatile uint8_t state = INIT;
33 volatile bool bounce_aux = false;
34 uint64_t t_first_edge = 0;
35 //#############################################
36 enum PWM_CHANNEL{
37   //Canal por nome
38   CHAN_MOTOR = 0b00000001,
39   CHAN_LED   = 0b00000010,
40 };
```

4.2 Módulo PWM

Este módulo foca em fazer funções para controlar as entradas PWM, desde a sua inicialização até a operação.

```
void pwm_initialize(){
  // Definir o modo de opera  o para FastPWM
  TCCR2A &= ~(1 << WGM22);
  TCCR2A |= (1 << WGM21 | 1 << WGM20);

  // Definir a fonte do clk (clk_i/256) => 16e6/256 = 62.5KHz
  TCCR2B &= ~(1 << CS20)
  TCCR2B |= (1 << CS21) | (1 << CS22);

  // Definido para 0% do ciclo de trabalho
  OCR2A = 0x00;
  OCR2B = 0x00;

  //Configura os pinos do PWM como saida
  DDRB |= 1 << LED_BLUE_B; // OC2A
  DDRD |= 1 << LED_YELLOW_D; // OC2B
}
// Ativar canais PWM
void pwm_enable (enum PWM_CHANNEL channel) {
  if (channel & CHAN_MOTOR) TCCR2A |= 1 << COM2A1;
  if (channel & CHAN_LED) TCCR2A |= 1 << COM2B1;
}
// Desativar canais PWM
void pwm_disable (enum PWM_CHANNEL channel) {
  if (channel & CHAN_OC2A) TCCR2A &= ~(1 << COM2A1);
  if (channel & CHAN_OC2B) TCCR2A &= ~(1 << COM2B1);
}
// Define o ciclo de trabalho no canal PWM_CHANNEL
void pwm_dutycycle (enum PWM_CHANNEL channel, uint8_t dutycycle)
   {
  if (channel & CHAN_OC2A) OCR2A = dutycycle;
  if (channel & CHAN_OC2B) OCR2B = dutycycle;
}
```

4.3 Módulo USART

Este módulo implementa a comunicação serial USART, para podermos transmitir os dados, por comunicação serial, para o computador.

```
void usart_initialize(){
  //habilita a comunicacao USART
  //habilita entrada (Rx) e saida(Tx)
  UCSR0B = (1<<RXEN0)|(1<<TXEN0);
  UCSR0C = (3<<UCSZ00);

  // definir a taxa de transmissao
  UBRR0H = (uint8_t)(BAUD_PRESCALLER>>8);
  UBRR0L = (uint8_t)(BAUD_PRESCALLER);
  //formato do frame 8bits de dado e 1stop bits
  UCSR0B = (1<<RXEN0)|(1<<TXEN0);//ativa o receptor e o
    transmissor
  UCSR0C = (3<<UCSZ00);
  //Habilita interrupcoes locais
  UCSR0B |= (1 << RXCIE0) | (1 << TXCIE0);
}
void usart_transmisionString(uint8_t *data){
  while(*data != 0x00){
    usart_transmision(*data);
    data++;
  }
}
//Coloca um byte no buffer de saida de dado
void usart_transmision(uint8_t data){
  //Aguarda a ultima transmissao ser concluida
  while( !(UCSR0A & (1<<UDRE0)));
  //coloca o byte no buffer
  UDR0 = data;
}
//Aguarda a recepcao ser concluida
uint8_t usart_reception(){
  //aguardar buffer a finalizacao da recepcao
  while ( !(UCSR0A & (1 << RXC0)) );
  return UDR0;
}
//Limpa o buffer de entrada de dados
void usart_flush(){
  uint8_t trash;
  while ( (UCSR0A & (1<<RXC0))) trash = UDR0;
}
```

4.4 Módulo de leitura dos sensores

Este módulo implementa funções para a leitura de dados dos sensores.

```
1  //Realiza a conversao ADC e retorna o valor em Graus Celcius
2  uint8_t readTherm(){
3    ADCSRA |= 0b10000111; //divide o clock_ms por 128 (o clock_ms
       de conversao sera 12Mhz/128)
4
5    ADMUX  |= 0b01000000;//usa o Vcc como ref
6    ADMUX  |= 0b00000001; //converte do pino A1
7    ADCSRA |= 0b01000000; // inicia a conversao A/D
8    while(!(ADCSRA & 0b00010000)); //Aguarda a conversao ser
       concluida
9    return (ADC*27)/512;
10 }
11 uint8_t readDHT_byte(){
12   uint8_t data = 0;
13   for(uint8_t i = 0; i < 8; i++){
14     while( !(PINC & (1<<DHT_PIN)));//espera sair da zona LOW
15     _delay_us(30);
16     if( PINC & (1<<DHT_PIN) )//se ler nivel Alto nessa linha,
       entao o bit eh 1
17       data |= 1 << (7-i);
18     while( PINC & (1<<DHT_PIN) );//espera receber o nivel LOW
       indicando o proximo bit
19   }
20   return data;
21 }
22 bool readDHT(uint8_t data[]){
23   uint8_t dht_in = 0;
24   //Pedido de transmissao de dados
25   DDRC |= 1 << DHT_PIN;
26   PORTC&= !(1<<DHT_PIN);
27   _delay_ms(18); //ate 18ms
28   PORTC|= (1<<DHT_PIN);
29   _delay_us(1);
30
31   //Confirmacao do DHT
32   DDRC &= !(1<<DHT_PIN);
33   _delay_us(80); //ate 80us
34   dht_in = PINC & (1<<DHT_PIN);
35   if(dht_in)
36     return false;
37
38   _delay_us(80);//ate 80us
39   dht_in = PINC & (1<<DHT_PIN);
40   if(!dht_in)
41     return false;
42
```

```
43    //inicio da transmissao dos dados
44    _delay_us(80);
45    for(uint8_t i = 0; i < 5; i++){
46      data[i] = readDHT_byte();
47    }
48
49    uint8_t checksum = data[0] + data[2];
50
51    if(checksum != data[4])
52      return false;
53
54    return true;
55  }
```

4.5 Módulo das interrupções

Este módulo implementa o tratamento das interrupções. No programa, foram utilizadas interrupções para tratar o bounce gerado pelo o botão - com o objetivo de melhorar a máquina de estados implementada no módulo do programa principal. Também, foram implementados um timer em ms; bem como uma interrupção para recepção de dados via USART.

```
1  //Interrupcao responsavel por atualizar o estado da maquina de
       estados
2  //esta associada a interrupcao gerada pelo botao e eh realizado
3  //tratamento do bounce.
4  ISR(INT0_vect){
5    if(bounce_aux){//primeira chamada da interrupcao
6     t_first_edge = clock_ms();
7     bounce_aux = false;
8    }
9    else{
10     if((clock_ms() - t_first_edge) > 100){//Teste se houve 100ms
       de diferenca entre interrupcao atual e a ultima
11       bounce_aux = true;
12       state = (state == INIT)?CTRL:INIT;
13     }
14   }
15 }
16 ISR(TIMER1_COMPA_vect){
17   TCNT1H = 0;
18   TCNT1L = 0;
19   timer_ms++;
20 }
21 ISR(USART_RX_vect){
22   uint8_t data = usart_reception();
23   usart_flush();
24   if(data == '1')state = CTRL;
25   else if(data == '0')state = INIT;
26   else {
27     PORTB |=  (1 << LED_GREEN_B);
28     _delay_ms(500);
29     PORTB &= ~(1 << LED_GREEN_B);
30     _delay_ms(500);
31   }
32 }
```

4.6 Módulo do programa principal

Neste módulo fica o programa principal, nele foram utilizados uma função de setup - para facilitar a leitura da main-, e uma máquina de estados para melhorar a legibilidade e desenvolvimento do código.

```
void setup(){
  DDRB |= 1 << LED_GREEN_B; //configura LED_GREEN_B como saida
  DDRD &= !(1 << BUTTON_D); //configura BUTTON_D como entrada
  DDRC |= 1 << DHT_PIN;


  PORTC|= !(1 << DHT_PIN);       //coloca DHT_PIN para nivel alto
  PORTB&= !(1 << LED_GREEN_B);   //pino LED_GREEN_B para nivel
    baixo


  //configura interrupcao para borda de descida no BUTTON_D
  EICRA &= !(1 << ISC00);
  EICRA |=  (1 << ISC01);
  EIMSK |=  (1 << INT0); //acionar apenas INT0

  //configuracoes do contador T1
  //Modo normal de operacao e sem prescaler.
  TCCR1A = 0x0;
  TCCR1B |= 1 << CS10; //no prescaler.
  //coloca um valor para Math A, que corresponda a 1ms.
  OCR1AH = (F_CPU/1000UL) >> 8;
  OCR1AL = (F_CPU/1000UL) << 8;
  TIMSK1 |= 1 << OCIE1A; //habilita interrupcao para Math A.


  usart_initialize();
  pwm_initialize();
  pwm_enable( CHAN_LED | CHAN_MOTOR );

  sei();//habilita interrupcoes, chave global
}

int main(){
  setup();
  // index 0 => Humidade %, parte inteira
  // index 1 => Humidade %, parte decimal
  uint8_t dht_data[5];
  uint64_t start_timer;
  uint64_t timer;
  uint8_t duty_MOTOR = 0, duty_LED = 255;
  uint8_t T;
  int z;
```

```
43  uint8_t U;
44
45  while(true){
46      switch (state) {
47        case INIT:
48          duty_MOTOR = 0;
49          duty_LED = 0;
50          PORTB &= ~(1 << LED_GREEN_B);
51          pwm_dutycycle(CHAN_LED,0);
52          pwm_dutycycle(CHAN_MOTOR,0);
53          start_timer = clock_ms();
54        break;
55        case CTRL:
56          PORTB |= (1 << LED_GREEN_B);
57          timer = clock_ms() - start_timer;
58          readDHT(dht_data);
59          U = dht_data[0];
60          T = readTherm();
61          duty_LED = (U/90.0)*256;
62
63          if( timer < 10000){
64            //comportamento 1
65            z = (3.5/100.0)*(timer/1000.0)*256;
66          }else if(timer > 10000 && timer < 15000){
67            //comportamento 2
68            z = (35/100.0)*256;
69          }else if(timer > 15000 && timer < 20000){
70            //comportamento 3
71            z = ((((65-32)/5.0)/100.0)*(timer/1000.0 - 15) +
    35/100.0)*256;
72          }else if(timer > 20000 && timer < 25000){
73            //comportamento 4
74            z = (65/100.0)*256;
75          }else if(timer > 25000 && timer <= 30000){
76            //comportamento 5
77            z = (((-65.0/5)/100.0)*(timer/1000.0 - 25) +
    65/100.0)*256;
78          }else{
79            //finish
80            state = INIT;
81          }
82
83          duty_MOTOR = z*U*ALFA + BETA*T;
84          pwm_dutycycle(CHAN_MOTOR, duty_MOTOR);
85          pwm_dutycycle(CHAN_LED, duty_LED);
86
87          usart_transmision(duty_MOTOR);
88
89        break;
```

```
90          default:
91
92          break;
93        }
94    }
95
96    return 0;
97  }
```

5 Resultados

Para observamos o funcionamento do sistema, utilizamos o módulo USART para enviar periodicamente os valores calculados para o PWM do motor, via serial para o computador, e plotar esses valores com um script em python, o algoritmo bem como a curva obtida são apresentados a seguir.

```
import serial
import matplotlib.pyplot as plt
from drawnow import *
import atexit

values = []

plt.ion()
cnt=0

serialArduino = serial.Serial('/dev/ttyUSB0', 9600)
# serialArduino.bytesize = 8
serialArduino.parity = serial.PARITY_NONE
serialArduino.stopbits = serial.STOPBITS_ONE
serialArduino.timeout = None


def plotValues():
    plt.title('PWM_MOTOR')
    plt.grid(True)
    plt.ylabel('PWM')
    plt.plot(values, 'rx-', label='values')
    plt.legend(loc='upper right')

def doAtExit():
    serialArduino.close()
    print("Close serial")
    tmp = str(serialArduino.isOpen())
    print(str("serialArduino.isOpen() = ") + tmp)

atexit.register(doAtExit)

print(str("serialArduino.isOpen() = ") + str(serialArduino.isOpen()))
```

```
#pre-load dummy data
for i in range(0,30):
    values.append(0)

serialArduino.write(b'0x01')
serialArduino.flushOutput()
while True:
    while (serialArduino.inWaiting()==0):
        pass
    valueRead = serialArduino.read()
    serialArduino.flushInput()


    valueInInt = int.from_bytes(valueRead,byteorder='little')
    if valueInInt <= 255:
        if valueInInt >= 0:
            values.append(valueInInt)
            values.pop(0)
            drawnow(plotValues)

        else:
            print("Invalid! negative number")
    else:
        print("Invalid! too large")

serialArduino.flush()
serialArduino.close()
```

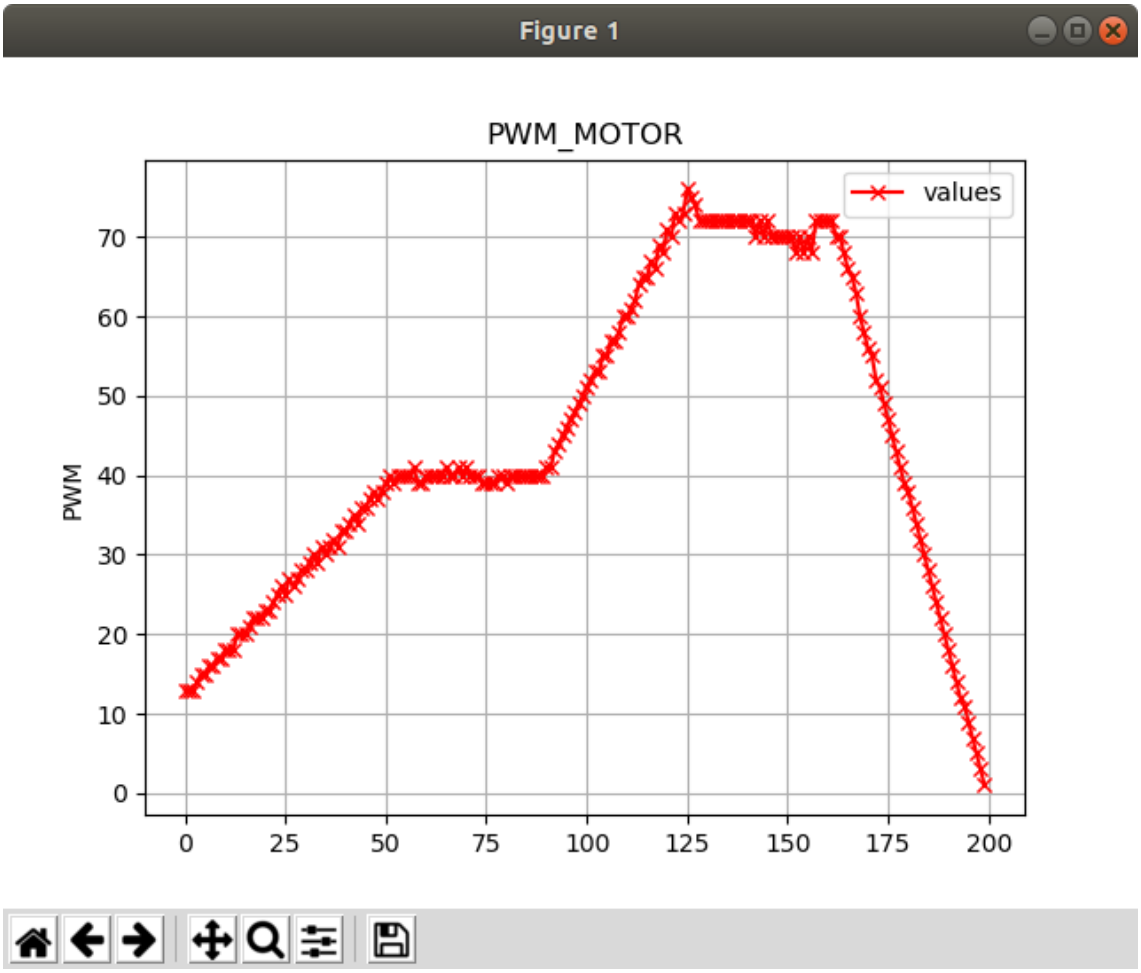

Figura 3: Curva do PWM do motor

A imagem 3 foi gerada com um algoritmo em python, o módulo de comunicação USART do microcontrolador envia os resultados do PWM pela serial do computador e o algoritmo python gera a curva em tempo real, o resultado também pode ser visto no vídeo de demostração do funcionamento do sistema.

6 Link para o vídeo

Segue aqui o link para o vídeo de apresentação dos resultados:

https://www.dropbox.com/s/r9jecjjedzhb3sm/video_resultadoPU1.mp4?dl=0

Referências

[1] *DATASHEET COMPLETE, ATmega328/P.* http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P$_D atasheet.pdf$.