به نام خدا

دانشگاه تهران

پردیس دانشکده های فنی

دانشکده برق و کامپیوتر

پروژه نهایی درس

طراحی بر اساس ریزپردازنده

عنوان پروژه

سیستم کنترل حرارت موتور

استاد درس

دکتر فاطمی

اعضای گروه

فهرست

مقدمه :

خواسته این پروژه به ترتیب به شرح زیر است:

۱. دریافت دما از طریق سنسور NTC تحت پریفرال ADC متصل شده به پین ۲۵ پورت صفر

۲. اتصال به ماژول وای فای esp8266 تحت پروتکل UART1

۳. ارسال دما از طریق وای فای به سرور thinkspeak

۴. آنالیز روی دماهای ارسال شده از سمت میکرو توسط افزونه متلب سرور

۵. ارسال پاسخ متناسب به عنوان وضعیت روشن/خاموش موتور به سمت میکرو

۶. دریافت پاسخ سرور توسط میکرو

۷. نشان دادن وضعیت موتور روی hexseg

(۸.) جریان اطلاعات و دستورات در تمامی مراحل توسط دیباگر متصل به یوآرت ۲ قابل مشاهده باشد.

تحلیل کد:

در ابتدای کد کتابخانه های لازم آورده شده‌اند.

```
#include "LPC17xx.h"
#include <stdio.h>
#include <string.h>
#include <stdbool.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "lpc17xx_gpio.h"
#include "lpc17xx_adc.h"
#include "lpc17xx_pinsel.h"
```

در این پروژه قسمت UART به صورت رجیستری و بدون کتابخانه کمکی زده شده است ، دلیل استفاده نکردن از کتابخانه به دلیل نکات و جزییاتی بود که باید به صورت دقیقی در سطح رجیسترها تعیین می گردید. اما راه اندازی ADC و HEXSEG با کمک کتابخانه پریفرال های ADC و GPIO شرکت کمسیس انجام شده است.

توابع:

در ادامه توابع طراحی شده در کد آورده شده‌اند که به توضیح هر یک از آنها خواهیم پرداخت.

```
int string_analyze(void);
void hexseg(int);
void stringer(void);
void stringer2(void);
int delay (int);  // Delay function
unsigned char getchar1 (bool);  // Get Char from UART1
unsigned char getchar2 (void);  // Get Char from UART2
void sendchar1 (unsigned char ); // Send Char from UART1
void sendchar2 (unsigned char ); //  Send Char from UART2
void sendstring1 (char *);  //  Send string from UART1
void sendstring2 (char *);   // Send string from UART2
```

ابتدا به توابع sendchar1 , sendchar2 پرداخته می‌شود که به ترتیب مربوط به ارسال character به UART1 و UART2 می‌باشند.

THR رجیستری است که دیتایی که قرار است از طریق UART منتقل شود در آن قرار میگیرد. بیت پنجم LSR نشان می دهد که دیتا در رجیستر THR قرار گرفته یا خیر. اگر THR خالی باشد این بیت صفر و اگر دیتا در آن آماده ارسال باشد این بیت برابر یک می‌شود. به همین دلیل شرط عبور از حلقه while صفر بودن بیت پنجم LSR می‌باشد. یعنی برای ارسال character تابع صبر می‌کند تا رجیستر THR خالی شود. بعد از آن به خط بعد می رود و character که قرار است ارسال شود روی رجیستر THR قرار می‌گیرد.

```
238  void sendchar1 (unsigned char ch) // Send Char from UART1
239  {
240      while (!(LPC_UART1->LSR & 1<<5));
241      LPC_UART1->THR=ch;
242      LPC_UART1->TER |= (1 << 7);
243  }
244
245  void sendchar2 (unsigned char ch) //  Send Char from UART2
246  {
247      while (!(LPC_UART2->LSR & 1<<5));
248      LPC_UART2->THR=ch;
249      LPC_UART2->TER |= (1 << 7);
250  }
```

شکل ۱ - کد توابع ارسال کاراکتر UART ها

در ادامه از این توابع برای ارسال یک رشته (string) به پورت های سریال استفاده شده‌است. بدین منظور توابع sendstring1 , sendstring2 تعریف شده‌اند.

همانطور که در تعریف این توابع مشخص است با استفاده از توابع sendchar یک رشته از کلمات را به صورت character به character ارسال می‌کنند. به تعداد کارکتر های یک رشته ، تابع sendchar برای هر کارکتر فراخوانی می‌شود.

```
252  void sendstring1 (char *str)  //  Send string from UART1
253  {
254      for (int i=0;str[i]!='\0';i++)
255      sendchar1(str[i]);
256
257  }
258
259  void sendstring2 (char *str)   // Send string from UART2
260  {
261      for (int i=0; str[i] ;i++)
262      sendchar2 (str[i]);
263
264  }
```

شکل ۲ - توابع ارسال رشته از طریق UART

برای دریافت character از پورت سریال توابع getchar1 , getchar2 به ترتیب برای UART1 و UART2 تعریف شده‌اند.

آخرین داده ای که UART دریافت می‌شود در رجیستر RBR قرار می‌گیرد. حال اگر داده دریافت شده در RBR قرار بگیرد بیت صفر LSR برابر یک و زمانی که RBR خالی شود به طور خودکار صفر می‌شود.

بنابراین تابع به نحوی نوشته شده‌است که منتظر می‌ماند تا بیت صفر LSR برابر صفر شود. سپس داده‌ای که در RBR است را به عنوان خروجی داده برمی‌گرداند.

```
unsigned char getchar1 (void)  // Get Char from UART1
{
    while (!(LPC_UART1->LSR & 1<<0));
    return LPC_UART1->RBR;
}

unsigned char getchar2 (void)  // Get Char from UART2
{
    while (!(LPC_UART2->LSR & 1<<0));
    return LPC_UART2->RBR;
}
```

شکل ۳ - توابع دریافت کاراکتر از UART

در ادامه و در توابع stringer از این دوتابع برای دریافت یک رشته به صورت سریال استفاده شده‌است. با استفاده از این توابع رشته به صورت character به character دریافت می‌شود.

```
void stringer()
    {
    int t=0;
    char temp[]="HI"; //
    char close[]="OK";  //

    while( strcmp(close,temp)!=0 )  //t<count
    {
        string_in[t]=getchar1();
        t++;
        delay(10);
        if(t>1) //
            sprintf(temp,"%c%c",string_in[t-2],string_in[t-1]); //
    }

}

void stringer2() {
    int t=0;
    char temp[]="HICHII";
    char close[]="CLOSED";

    while( strcmp(close,temp)!=0  )
    {
        string_in[t]=getchar1();
        t++;
        delay(10);
        if(t>5)
            sprintf(temp,"%c%c%c%c%c%c",string_in[t-6],string_in[t-5],string_in[t-4],string_in[t-3],string_in[t-2],string_in[t-1]);
    }
}
```

شکل ۴ - توابع دریافت رشته از UART

نحوه عملکرد تابع stringer به این صورت است که تا زمانی که رشته OK دریافت نشود به دریافت رشته ها از UART ادامه می‌دهد. این تابع برای دریافت پاسخ wifi تعریف شده است که می دانیم در انتهای پاسخ‌های wifi رشته OK به طور ثابت قراردارد.

تابع stringer2 برای دریافت پاسخ یک دستور خاص از سرور است. زمانی که از سرور داده‌ها ارسال می‌شوند در انتهای پاسخ رشته CLOSED قرار دارد. بنابراین stringer2 تا زمانی که رشته CLOSED را دریافت کند به دریافت رشته‌ها ادامه می‌دهد.

تابع string_analyze برای تحلیل و تشخیص داده اصلی موجود در رشته پاسخ سرور نسبت به درخواست داده شده می‌باشد. از آنجا که هنگام نوشتن در field1 داده نامعتبر در field2 قرار می‌گیرد، برای خواندن داده درخواست نشان دادن ۶ داده آخر به سرور فرستاده می‌شود. برای تحلیل پاسخ سرور از انتهای پاسخ شروع به تحلیل شده‌است. ۴ کاراکتر جلوتر از هر رشته field2 یافته شده در پاسخ سرور که دیتا قرار می‌گیرد

بررسی شده‌است. اگر ۱ باشد ۱ و اگر ۰ باشد ۰ برگردانده می‌شود. طبیعتا اگر داده نامعتبر یا null به عنوان داده قرار گرفته شده‌باشد داده‌ای برگردانده نمی‌شود.

```
int string_analyze()
{
    char f[]="field2";
    char f2[]="hichii";

    int t=599;
    while(t>=0)
    {

        sprintf(f2,"%c%c%c%c%c%c",string_in[t-5],string_in[t-4],string_in[t-3],string_in[t-2],string_in[t-1],string_in[t]);
        if( strcmp(f,f2)==0 )
        {   if( string_in[t+4]=='0' )
                return 0;
            else if( string_in[t+4]=='1' )
                return 1;
        }
        t--;

    }
    return 2;
}
```

شکل ۵ - تابع آنالیز و استخراج دیتای موردنظر از رشته پاسخ سرور

همچنین یک تابع delay برای ایجاد تاخیر های لازم در اجرای کد اصلی تعریف شده‌است که در ادامه قابل مشاهده است.

```
int delay(int count)  // Delay function
{
  int j=0,i=0;
  for(j=0;j<count;j++)
  {
    // At 100Mhz, the below loop introduces  DELAY of 1 us
    for(i=0;i<23;i++)
        {
            __asm__("nop\n\t");
  }
}
    return 10;
}
```

شکل ۶ - تابع ایجاد تاخیر سخت افزاری

برای ایجاد تاخیر یک دستور سخت افزاری داخل حلقه قرار گرفته که این حلقه ۱ میکروثانیه تاخیر ایجاد می‌کند.

برای نشان دادن وضعیت روشن بودن یا خاموش بودن موتور، ما از نمایشگر hexseg استفاده می کنیم، تابع hexseg آن را کنترل می کند:

```
253  void hexseg(int a)
254  {
255      GPIO_SetDir(0,0x000000FF, 1);
256      GPIO_SetDir(2,0x000000FF, 1);
257      GPIO_ClearValue(0, 0x000000FF);
258      GPIO_ClearValue(2, 0x000000FF);
259      if(a==1)
260      {
261          GPIO_SetValue(0, 0x000000C7);  //R
262          GPIO_SetValue(2, 0x00000098);
263      }
264      else if(a==0)
265      {
266          GPIO_SetValue(0, 0x000000BB);  //S
267          GPIO_SetValue(2, 0x00000088);
268      }
269      else
270      {
271      GPIO_ClearValue(0, 0x000000FF);
272      GPIO_ClearValue(2, 0x000000FF);
273      }
```

شکل ۷ - تابع ایجاد حروف S و R روی hexseg

در تابع hexseg حرف R به منزله روشن و حرف S به معنی خاموش است. ، باتوجه به ورودی که می گیرد ؛ اگر عدد یک باشد حرف R را روشن می کند و اگر عدد صفر باشد، حرف S را روشن خواهد کرد. در مابقی حالات به طور پیشفرض نمایشگر خاموش خواهدبود.

بخش main:

حال به تحیل بخش main کد پرداخته می‌شود.

راه اندازی UART ها :

از uart1 به عنوان واسط بین برد و ماژول وای فای esp8266 استفاده شده است. هم چنین uart2 ، ابزار دیباگ و مشاهده نتایج دستورات در این پروژه هست. تمامی دستورات و داده ها و پاسخ هایی که بین برد ، وای فای و سرور رد و بدل می شود توسط uart2 به کامپیوتر ارسال شده و در برنامه hecules مشاهده می شود.

```
33      //init UART1
34      LPC_SC->PCLKSEL0|=0X0;     //SET CLOCK OF UART0    CPUCLK/4=24MHZ
35      LPC_UART1->LCR=0X83;      //SET 8bit data & enable dlab
36      LPC_UART1->DLL=13;        // SET BAUD RATE = 115200
37      LPC_UART1->DLM=0;
38      LPC_UART1->LCR=0X3;      // DESABLE DLAB
39      LPC_UART1->FCR=0X7;      // SET FIFO AND CLAER
40      LPC_PINCON->PINSEL0 |= (1 << 30);             // Pin P0.10 used as TXD2 (Com2)
41    LPC_PINCON->PINSEL1 |= (1 << 0); // SET PIN FOR UART0
42      LPC_UART1->IER=0X01;
43
44      // init UART2
45      LPC_SC->PCLKSEL0|=0X0;     //SET CLOCK OF UART0    CPUCLK/4=24MHZ
46      LPC_PINCON->PINSEL0 |= (1 << 20);             // Pin P0.10 used as TXD2 (Com2)
47    LPC_PINCON->PINSEL0 |= (1 << 22);            // Pin P0.11 used as RXD2 (Com2)
48      LPC_SC->PCONP = LPC_SC->PCONP|(1<<24);        //Open UART2 power control bit
49      LPC_UART2->LCR=0x83;      //SET 8bit data & enable dlab
50      LPC_UART2->DLL=13;        // SET BAUD RATE = 115200
51      LPC_UART2->DLM=0;
52      LPC_UART2->LCR=0X3;      // DESABLE DLAB
53      LPC_UART2->FCR=0X7;      // SET FIFO AND CLAER
54      LPC_UART2->IER=0X01;
```

شکل 8 - راه اندازی تنظیمات اولیه uart ها

بخش ابتدایی مربوط به مقداردهی‌های اولیه و راه اندازی uart1 ,2 می باشد. baudrate هردو ۱۱۵۲۰۰ می باشد. این نرخ داده توسط رجیسترهای DLM و DLL و با کمک مقسم کلاک PCLKSEL0 تولید می شود. ارسال و دریافت داده بدون parity و در قالب ۸ بیت و با یک stopbit خواهد بود، این مقادیر توسط رجیستر LCR تعیین می شوند. هم چنین باید پین یوآرت ها و نوع function آن ها تعیین شود، برای این کار ما در رجیسترهای pinsel0,1 عملکرد پین های rx و tx متناظر را تعیین کردیم.

به طور خلاصه رجیسترهای به کار رفته در جدول زیر نشان داده شده اند:

Register	Description
RBR	Contains the recently received Data
THR	Contains the data to be transmitted
FCR	FIFO Control Register
LCR	Controls the UART frame formatting(Number of Data Bits, Stop bits)
DLL	Least Significant Byte of the UART baud rate generator value.
DLM	Most Significant Byte of the UART baud rate generator value.

شکل ۹- رجیسترهای راه اندازی uart

راه اندازی ADC :

برای استفاده از ADC از کتابخانه های lpc17xx_adc.h , lpc17xx_pinsel.h که خود شرکت NXP آن را ارائه نموده است، استفاده شد. برای مهیا کردن ADC (initialization) همانطور در **Error! Reference source not found.** زیر آمده است، لازم است موارد زیر انجام شود :

```
PINSEL_CFG_Type adcpinsel;
adcpinsel.Funcnum=1;
adcpinsel.OpenDrain=PINSEL_PINMODE_NORMAL;
adcpinsel.Pinmode=PINSEL_PINMODE_PULLUP;
adcpinsel.Pinnum=25;
adcpinsel.Portnum=0;

PINSEL_ConfigPin(&adcpinsel);

ADC_Init(LPC_ADC,100000);
ADC_ChannelCmd(LPC_ADC,ADC_CHANNEL_2,ENABLE);
```

شکل ۱۰- تنظیمات اولیه ADC

(۱) با استفاده از کلاس PINSEL_CFG_Type یک متغیر تعریف می‌کنیم که این نشان دهنده پین ADC ما و تنظیمات مربوط به آن است.

(۲) با توجه به اینکه برای استفاده از P0.25 به عنوان ADC باید نحوه عملکرد آن را روی ADC قرار دهیم، Funcnum آن را برابر با یک قرار می‌دهیم.

(۳) وضعیت اوپن-درین بودن را نبودن را مشخص می‌کنیم.

(۴) مشخص می‌کنیم که از مقاومت پول-آپ یا پول-دوان استفاده خواهیم کرد یا پین را به صورت Tristate استفاده می‌کنیم.

(۵) سپس پورت و پین مربوطه مشخص می‌کنیم و با فراخوانی تابع PINSEL_ConfigPin این تنظیماتی که تعیین شد را اعمال می‌کنیم.

(۶) با فراخوانی تابع ADC_Init نرخ نمونه برداری را تعیین می‌کنیم (ماکسیمم ۲۰۰۰۰۰).

(۷) و در آخر کانال ADC مربوط به این پین (p0.25) را فعال می‌کنیم که اینجا کانال ۲ می‌باشد.

شروع ارسال دستورات و داده به wifi :

در خط ۷۲ کد یک آرایه رشته با نام str تعریف شده‌است. این درواقع یک بافر نگهدارنده دستورات ارسالی به وای فای است. ما دستورات خود را به صورت رشته در آن ذخیره می کنیم.

```
72      char str[75];
```

شکل ۱۱ - بافر نگهدارنده دستورات ارسالی

در بخش بعدی به کمک توابعی که تعریف شده‌اند و پیش از این توضیح داده شد، رشته‌هایی شامل دستورات به ترتیب برای wifi ارسال می‌شود. ما برای ارتباط با ماژول esp8266 باید از طریق AT commands با آن صحبت کنیم. نکته مهم در این جا این هست که به ازای هر دستور ارسالی، وای فای یک پاسخ متناسب

ارائه خواهد داد. اگر دستور مورد نظر با موفق اجرا شده باشد ، در انتهای پاسخ حتما یک OK خواهیم داد. بنابراین ضروری هست تا پاسخ ها نیز دریافت و نمایش داده شود.

```
84      sprintf (str,"AT%c%c",0x0d,0x0a);
85      sendstring1(str);
86      stringer();
87      sendstring2(string_in);
88
89
90      sprintf (str,"AT+CWMODE=1%c%c",0x0d,0x0a);
91      sendstring1(str);
92      stringer();
93      sendstring2(string_in);
94
95
96      sprintf (str,"AT+CWJAP=\"MPLab\",\"MpProject1400\"%c%c",0x0d,0x0a);
97      sendstring1(str);
98      stringer();
99      sendstring2(string_in);
```

شکل ۱۲ - دستورات ابتدایی اتصال به وای فای

۱. ابتدا رشته AT ارسال می‌شود تا تنها از ارتباط صحیح با ماژول wifi اطمینان حاصل شود. در صورت صحت عملکرد انتظار دریافت پاسخ OK از wifi می‌رود. همانطور که در کد مشاهده می‌شود توسط تابع sendstring1 یعنی از طریق UART1 رشته به wifi ارسال و پاسخ آن به کمک تابع stringer دریافت می‌شود و این پاسخ در رشته string_in قرارمی‌گیرد. برای مشاهده، پاسخ آن از طریق UART2 به کامپیوتر ارسال می‌شود.

۲. سپس دستور CWMODE=1 به wifi ارسال شده تا تعیین شود که به چه شکل قراراست از ماژول wifi استفاده شود. عدد یک به معنی مود station و عدد ۲ به معنی مود server هست.

۳. و درنهایت در رشته بعدی SSID و پسورد wifi آزمایشگاه وارد می‌شود تا اتصال انجام شود.

در ادامه بخش پایانی و اصلی که حلقه تکرارشونده کد می‌باشد آورده شده است که به توضیح آن پرداخته می‌شود.

حلقه اصلی برنامه:

```
102  while(1) {
103      k++;
104      if(k==7) k=1;
105
106
107      ADC_StartCmd(LPC_ADC,ADC_START_NOW);
108      while(ADC_ChannelGetStatus(LPC_ADC,ADC_CHANNEL_2,ADC_DATA_DONE)==0);
109      analog=ADC_ChannelGetData(LPC_ADC,ADC_CHANNEL_2);
110      int temp=(analog*4)-7800;
111
112      memset(string_in,0,sizeof(string_in));
113      sprintf (str,"AT+CIPMUX=0%c%c",0x0d,0x0a);
114      sendstring1(str);
115      stringer();
116      sendstring2(string_in);
117
118      memset(string_in,0,sizeof(string_in));
119      sprintf (str,"AT+CIPSTART=\"TCP\",\"api.thingspeak.com\",80%c%c",0x0d,0x0a);
120      sendstring1(str);
121      stringer();
122      sendstring2(string_in);
123
124      memset(string_in,0,sizeof(string_in));
125      int size_inst;
126      if(k%6!=0) size_inst=46+(int)(ceil(log10(temp))); else size_inst=47;
127      sprintf (str,"AT+CIPSEND=%d%c%c",size_inst,0x0d,0x0a);  //49or48 for writing /
128      sendstring1(str);
129      stringer();
130      sendstring2(string_in);
```

شکل ۱۳- دستورات ارسال و دریافت داده از سرور

این حلقه تا ابد ادامه خواهد داشت. در ۵ بار نخست حلقه ، ما عملیات نوشتن دما در سرور را انجام می دهیم. در بار ششم از سرور داده وضعیت روشن/خاموش بودن موتور دریافت می شود. این عملیات ۶ گانه بارها تکرار خواهدشد.

دریافت ADC و تبدیل آنالوگ به دما:

در خط ۱۰۷ و ۱۰۸ با فراخوانی تابع ADC_StartCmd، نمونه برداری و محاسبه توسط ADC را شروع می‌کنیم و پس از اینکه کار ADC تمام شد (بند دوم) داده را درون متغیری با عنوان analog می‌ریزیم.

در خطوط ۱۰۹ و ۱۱۰پس از اینکه مقدار آنالوگ بدست آمد، با توجه به تقسیم مقاومتی که صورت گرفته است مقدار مقاومت سنسور NTC بدست می‌آید. در سنسورهای NTC مقدار مقاومت، متناظر با یک دمای یکتاست که این در دیتاشیت های این سنسورها می‌آید. بنابراین با درون‌یابی از روی مقاومت به دست آمده می‌توان به دمای محیط رسید.

متاسفانه با توجه به معلوم نشدن تقسیم مقاومتی صورت گرفته، مجبور شدیم مقدار آنالوگ را با یک معادله فرضی به دما تبدیل کنیم و آن را گزارش نماییم.

اتصال به سرور و ارسال/دریافت اطلاعات:

۱. با ارسال دستور CIPMUX=0 وضعیت سینگل یا دوبل بودن کانال ارسال اطلاعات را مشخص می کنیم باتوجه به اینکه فقط با یک پورت ارتباط داریم، سینگل مود کافیست.

۲. با ارسال دستور CIPSTART ، آدرس سرور سایت thinkspeak و پورت متناظر یعنی ۸۰ را معین کنیم. بلافاصله پس از این ، ماژول وای یک کانال ارتباطی TCP بین برد و سرور ایجاد خواهدکرد

۳. در خط ۱۲۶ با توجه به api سایت thingspeak و همینطور با توجه به داده‌ای که قرار است به سرور ارسال شود، طول رشته ارسالی به سرور از طریق wifi مشخص می‌گردد.

```
132     memset(string_in,0,sizeof(string_in));
133     char write_inst[]="GET /update?api_key=K801QQ0UAIUVHN0W&field1=";
134     char read_inst[]="GET /channels/1405439/fields/2.json?results=6";
135     if(k%6==0)
136     sprintf (str,"%s%c%c",read_inst,0x0d,0x0a);      //GET /channels/1405439/fields
137     else sprintf (str,"%s%d%c%c",write_inst,temp,0x0d,0x0a);
138
139     sendstring1(str);
140     sprintf (str,"AT+CIPCLOSE%c%c",0x0d,0x0a);
141     sendstring1(str);
142     //stringer(1026,0);
143     stringer2();
144     sendstring2(string_in);
145
146     if(k%6==0)
147     {
148         int cond=string_analyze();
149         sendchar2(cond +'0');
150         hexseg(cond);
151     }
152
153 delay(60000000);
154 }
```

شکل ۱۴ - دستورات ارسال و دریافت داده از سرور

توجه کنید که حالت نوشتن یا خواندن از سرور توسط یک شرط بررسی می شود که آیا شمارنده حلقه در مضارب ۶ است یا خیر؟ زیرا به ازای هر ۵ دفعه نوشتن ، یک دفعه می خوانیم.

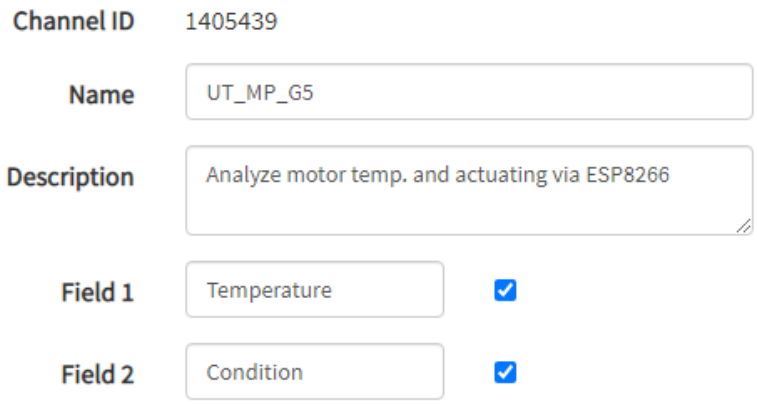
در ادامه با توجه به آن که قصد ارسال داده به سرور و یا دریافت داده از آن توسط ماژول wifi وجود داشته باشد، رشته مربوطه به wifi ارسال می‌گردد و پس از آن با ارسال رشته ای CIPCLOSE بسته می‌شود. همانطور که مشاهده می‌شود برای مشاهده پاسخ سرور از تابع stringer2 استفاده شده‌است که دلیل آن در توضیح تابع گفته شده‌است. پس از آن حلقه پایان می‌یابد و با اعمال تاخیری مجددا حلقه آغاز می‌شود.

در خط ۱۴۶ حالت مود خواندن را بررسی می کنیم. در وضعیت خواندن از سرور وارد این حلقه می شویم و پاسخ ارسالی از سرور را با استفاده از تابع analyaze تحلیل کرده و داده مورد نظرمان که همان وضعیت روشن/خاموش است را بیرون میکشیم. در ادامه آن را به hexseg خواهیم داد.

توضیح سرور:

در لبه سرور ما از سایت thinkspeak استفاده کرده ایم. در ادامه توضیحات آن خواهد آمد.

ما در این سایت کانال ایجاد کردیم:

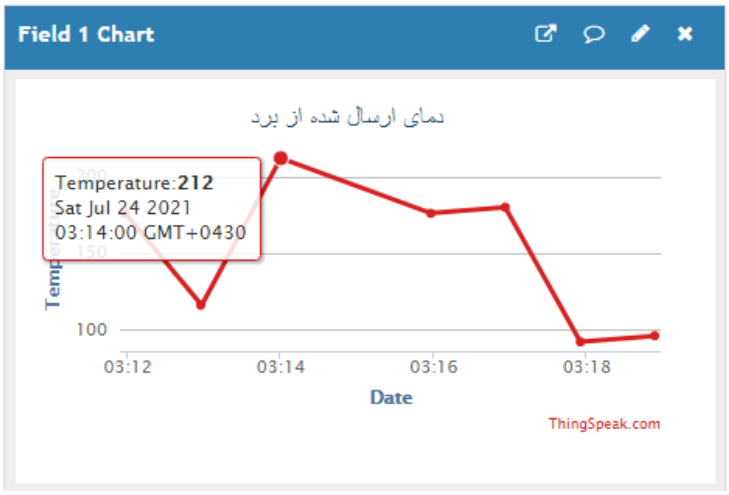


شکل ۱۵ - مشخصات کانال

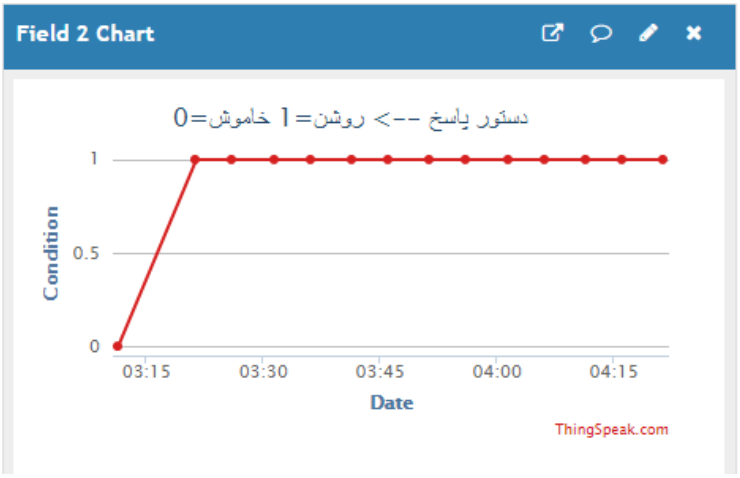
آیدی کانال و اسم آن درشکل بالا آمده است. هم چنین برای مشاهده عمومی وضعیت آن می توان از لینک زیر استفاده کرد:

https://thingspeak.com/channels/1405439

ما دو فیلد ایجاد کرده ایم. فیلد ۱ به نام Temprature ، دماهای ارسالی از برد در این فیلد ذخیره خواهندشد. فیلد ۲ Condition هست. این فیلد دربردارنده وضعیت روشن/خاموش موتور خواهد بود که توسط آنالیز متلب همین سایت محتوای فیلد ۲ آپدیت می شود. در واقع فیلد ۲ اطلاعاتی را دارد که ما از سمت برد آن ها را می خوانیم.



شکل ۱۶- فیلد ۱ - نوشتن دما



شکل ۱۷ - فیلد ۲ - خواندن وضعیت

پس در مجموع از دید میکرو ، فیلد ۱ نوشتنی و فیلد ۲ خواندنی هست. اطلاعات فیلد ۲ چگونه تولید می شوند؟

ما توسط افزونه سایت کد متلب زیر نوشته ایم:

MATLAB Code

```
1
2  channel_id = 1405439;
3  writeAPIKey ='K801QQ0UAIUVHN0W';
4
5  temp = thingSpeakRead(channel_id,'Fields',1,'NumMinutes',3);
6
7  avg_temp = mean(temp);
8  display(avg_temp,'Average temp');
9
10 if(avg_temp > 100 ) cond=0; else cond=1; end
11
12 thingSpeakWrite(channel_id,cond,'Fields',2,'WriteKey',writeAPIKey);
```

شکل ۱۸ - کد متلب تولید داده فیلد ۲

در این کد ، ما داده های ورودی فیلد ۱ یعنی همان دماهای ۳ دقیق قبل را می خوانیم. سپس از آن ها یک میانگین می گیریم، اگر میانگین دماها بزرگتر از صد بود ، مقدار صفر را در فیلد دوم می نویسیم در غیر این صوزت مقدار یک را خواهیم نوشت. پس داده های فیلد دو توسط این کد متلب تولید می شوند، همان طور که در شکل های قبل نیز دیدید ، داده های فیلد ۲ تنها صفر و یک هستند.

یک نکته مهم روند اجرای این کد هست که هر چند وقت یک بار تولید شود؟

توسط افزونه ای به نام time control ما تناوب های آپدیت فیلد ۲ را مشخص می کنیم:

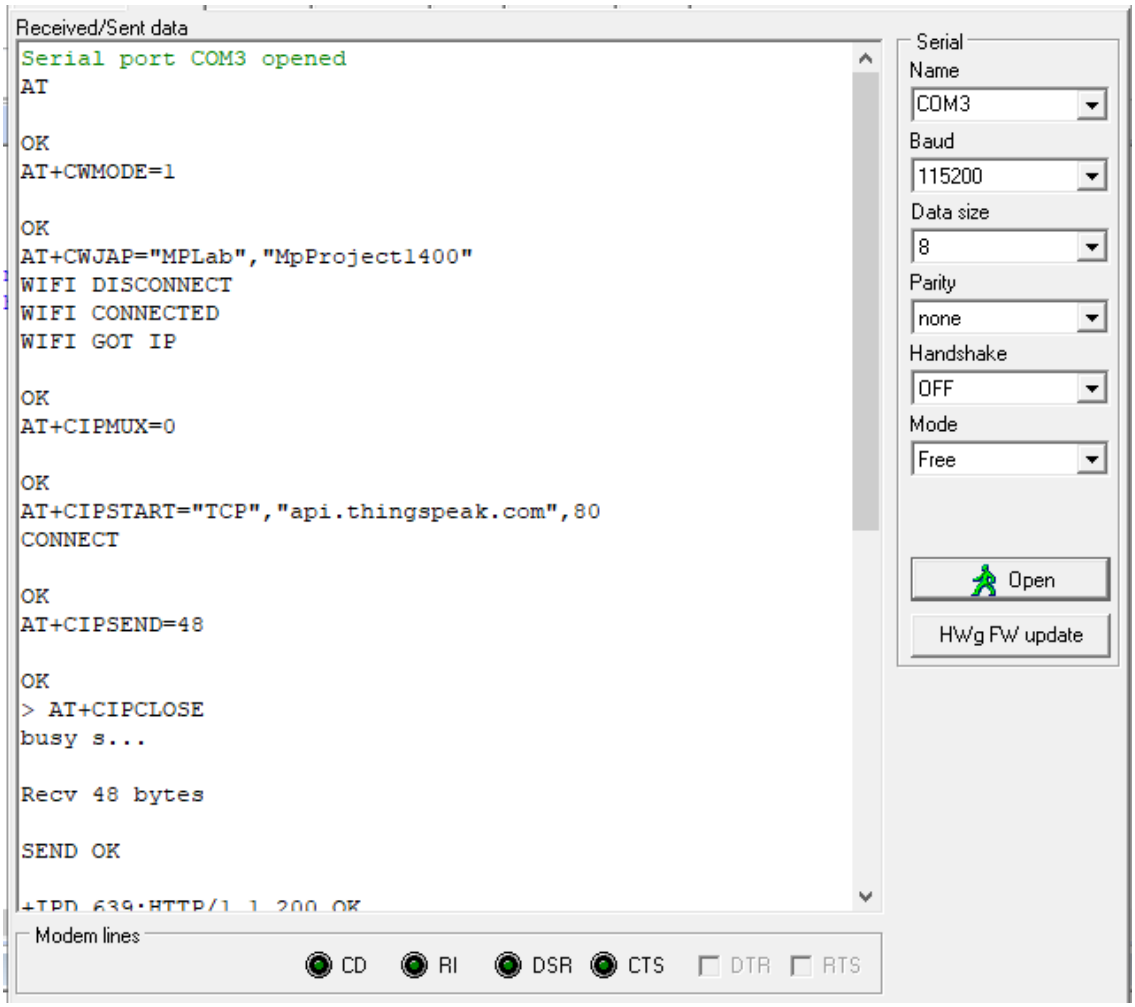
Name:	Time control
Frequency:	Every 5 minutes
Time Zone:	Tehran (edit)
Last Ran:	2021-07-24 4:26 am
Run At:	2021-07-24 4:31 am
Fuzzy Time:	± 0 minutes
MATLAB Analysis:	check condition

شکل ۱۹- کنترل زمانی اجرای کد فیلد ۲

در شکل بالا در هر ۵ دقیقه ، کد متلب condition اجرا خواهد شد. پس در نتیجه می توان گفت، داده های فیلد دوم از نظر زمانی اختلاف ۵ دقیقه ای خواهندداشت. به همین دلیل هم هست که در کد نوشته شده در برد ، ما هر ۵ دقیقه یک بار دستور خواندن را به سرور میفرستادیم.

نتایج اجرا :

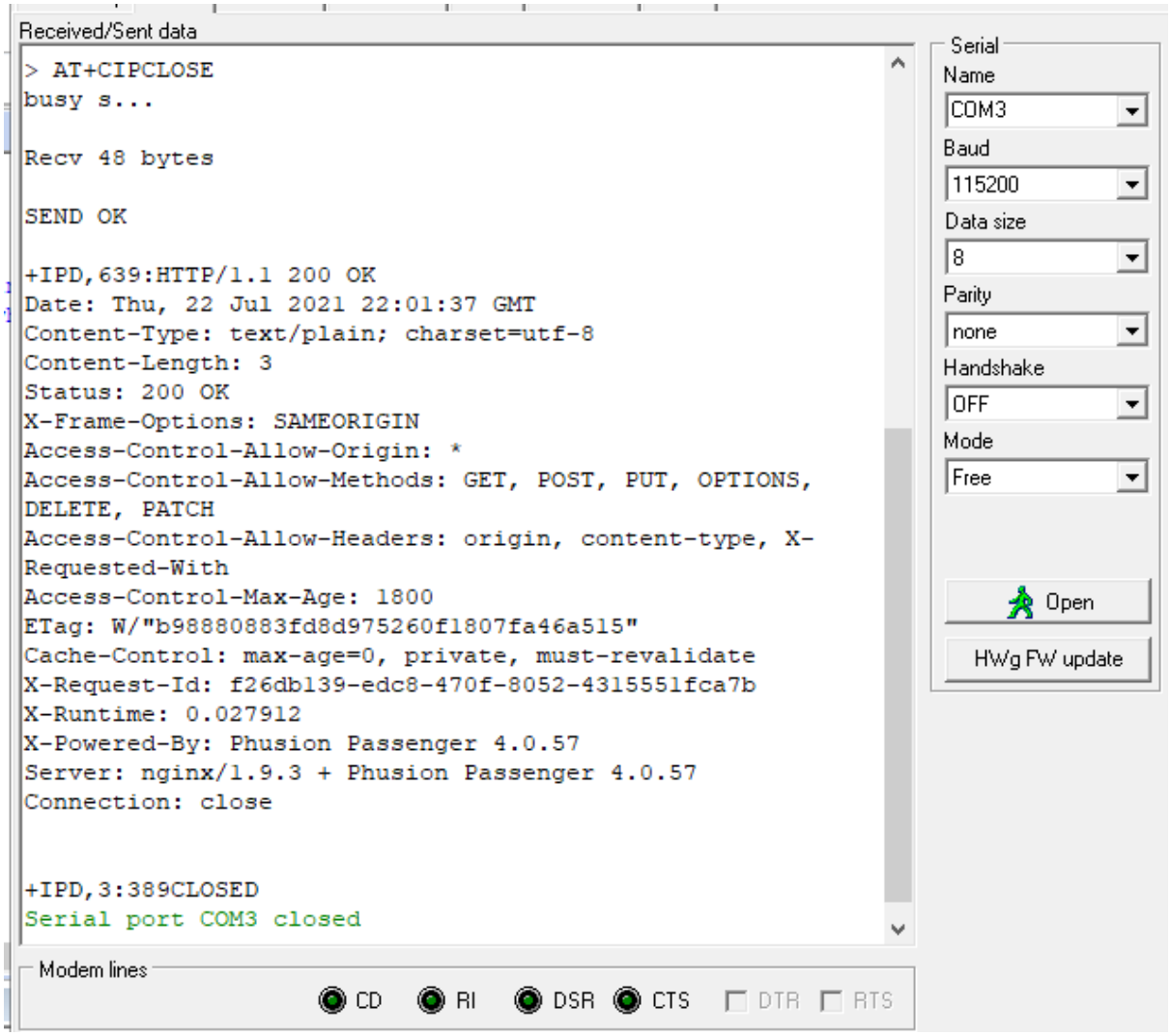
اول آنکه توصیه ما اینست که فیلم ضمیمه شده در مستندات را حتما مشاهده کنید، توضیحات و اجرای برنامه به صورت زنده انجام شده است که مطلوب تر می باشد.



شکل ۲۰- مشاهده دستورات و پاسخ های وای فای

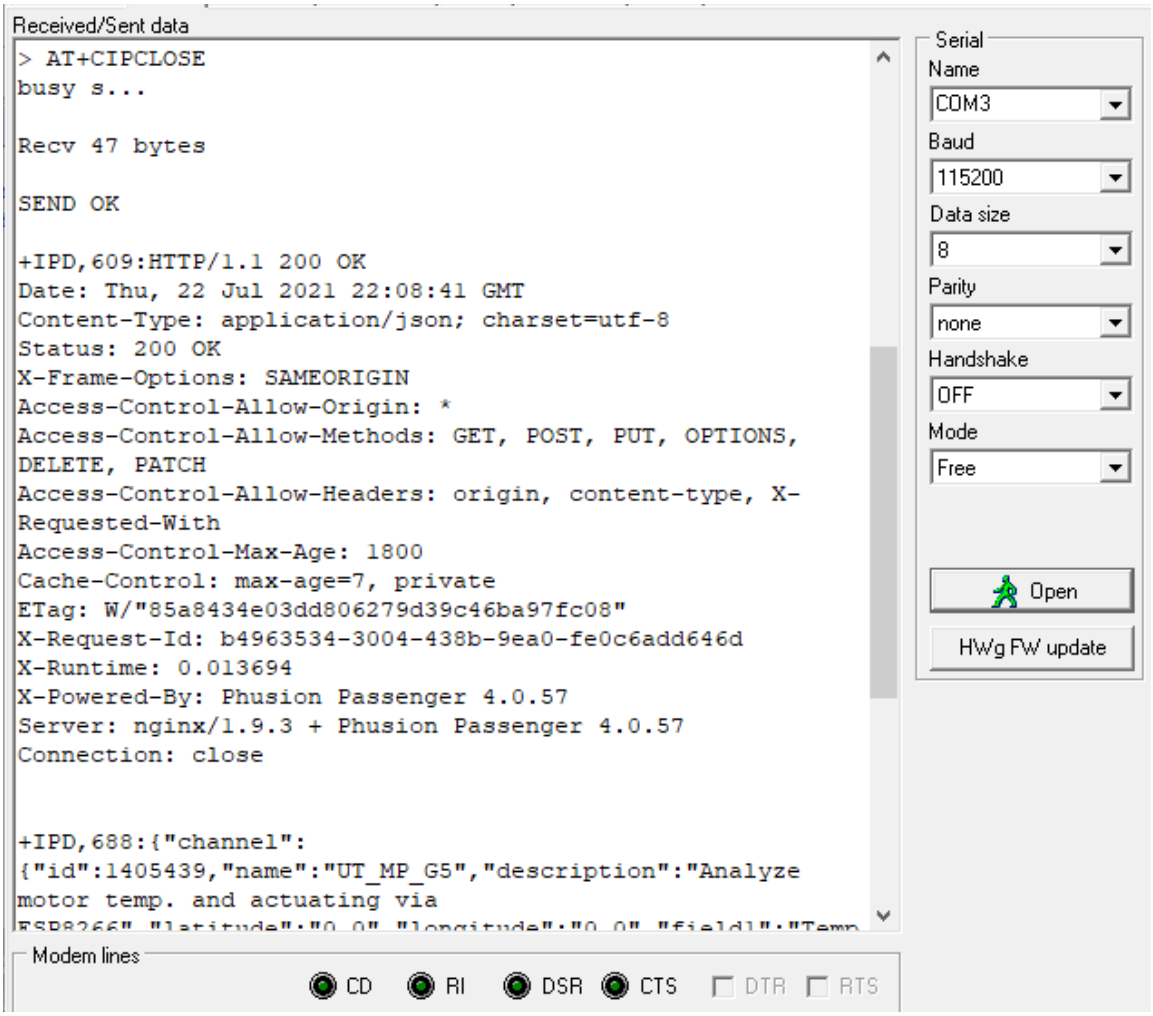
عکس بالا دستورات و پاسخ هایی هست که بین برد و ماژول وای فای رد و بدل شده است. رویه و منطق این دستورات در قسمت تحلیل کد توضیح داده شده است.

همان طور که میبنید ، پاسخ مازول به دستورات ارسالی OK و موفقیت آمیز بوده است. شکل بالا مربوط به حالت نوشتن در سرور می باشد.



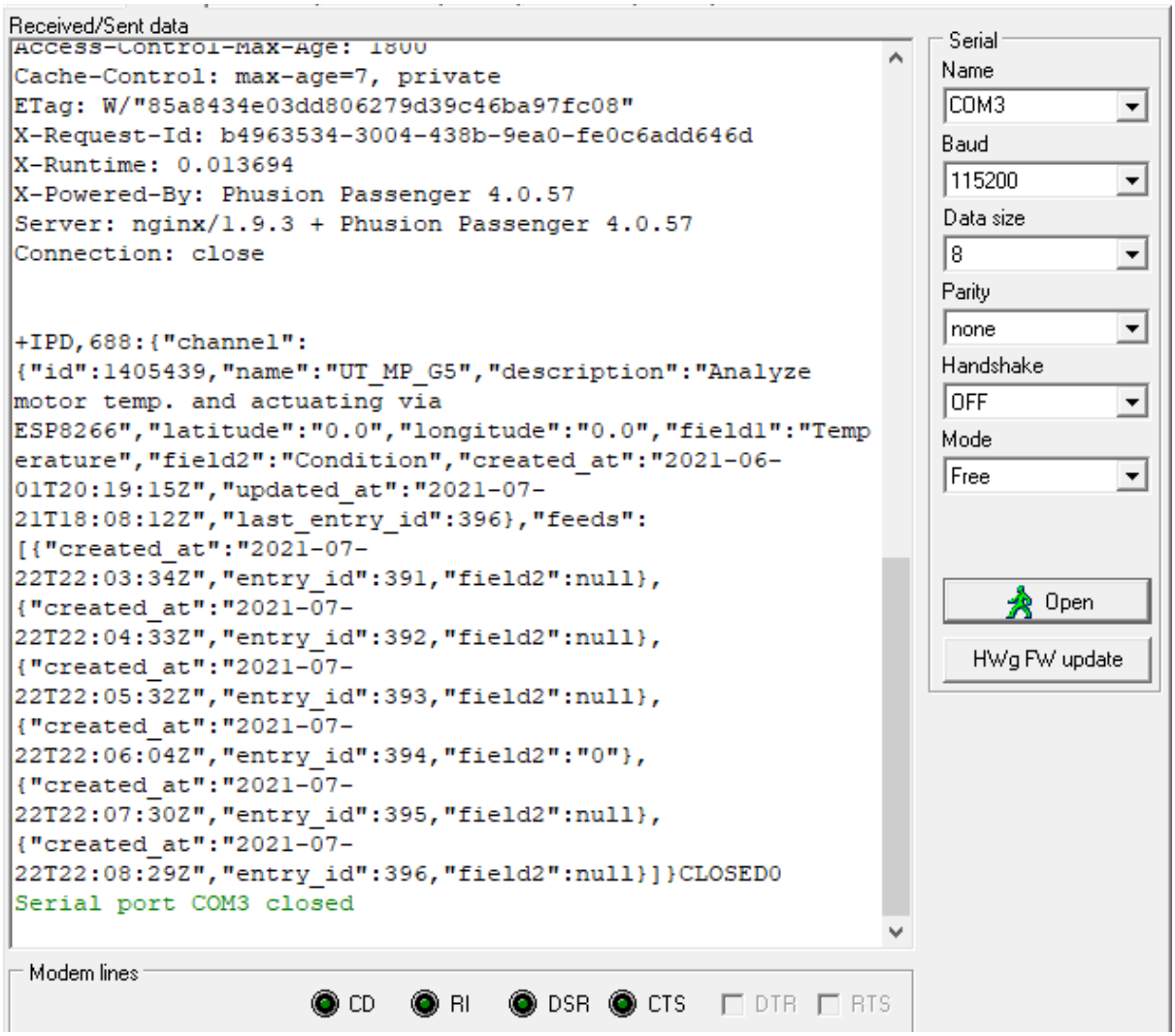
شکل ۲۱- پاسخ سرور به نوشتن دما

در عکس بالا پاسخ سرور به دستور نوشتن آمده است. همان طور که می بینید کد 200 OK از طرف سرور ارسال شده این بدین معنی هست که داده ما با موفقیت در فیلد ۱ سایت ، نوشته شده. هم چنین در انتهای پاسخ ، پیغام CLOSED جواب داده شده است.



شکل ۲۲ - بخش اول پاسخ سرور به خواندن وضعیت

هنگامی که از فیلد ۲ سرور داده ای را می خواهیم بخوانیم، پیغام پاسخ در دو بخش می آید. بخش اول که در شکل بالا مشخص هست ، حاوی اطلاعات عمومی پروتکل هست هم چنین پیغام 200 OK به منزله ارسال و دریافت موفق هست.



شکل ۲۳ - بخش دوم پاسخ سرور به خواندن وضعیت و داده های فیلد ۲

در بخش دوم پاسخ سرور به دستور خواندن از فیلد ۲ ، محتوای ۶ داده اخیر این فیلد فرستاده می شود. همان طور که در عکس بالا می بینید ، ۶ ردیف داده پشت هم آمده اند. داده های null نامعتبرهستند و ما فقط داده صفر یا یک را معتبر می دانیم. به همین دلیل توسط تابع string_analyze ، مقدار یک استخراج خواهدشد.

شکل ۲۴- نمایش وضعیت موتور

اگر وضعیت موتور خاموش باشد(که از فیلد ۲) خواندیم ، حرف S نمایش داده می شود ، در غیر این صورت حرف R نمایش داده خواهدشد.