aPCmeter Firmware

s1

Generated by Doxygen 1.8.11

Thu Jul 14 2016 21:05:04

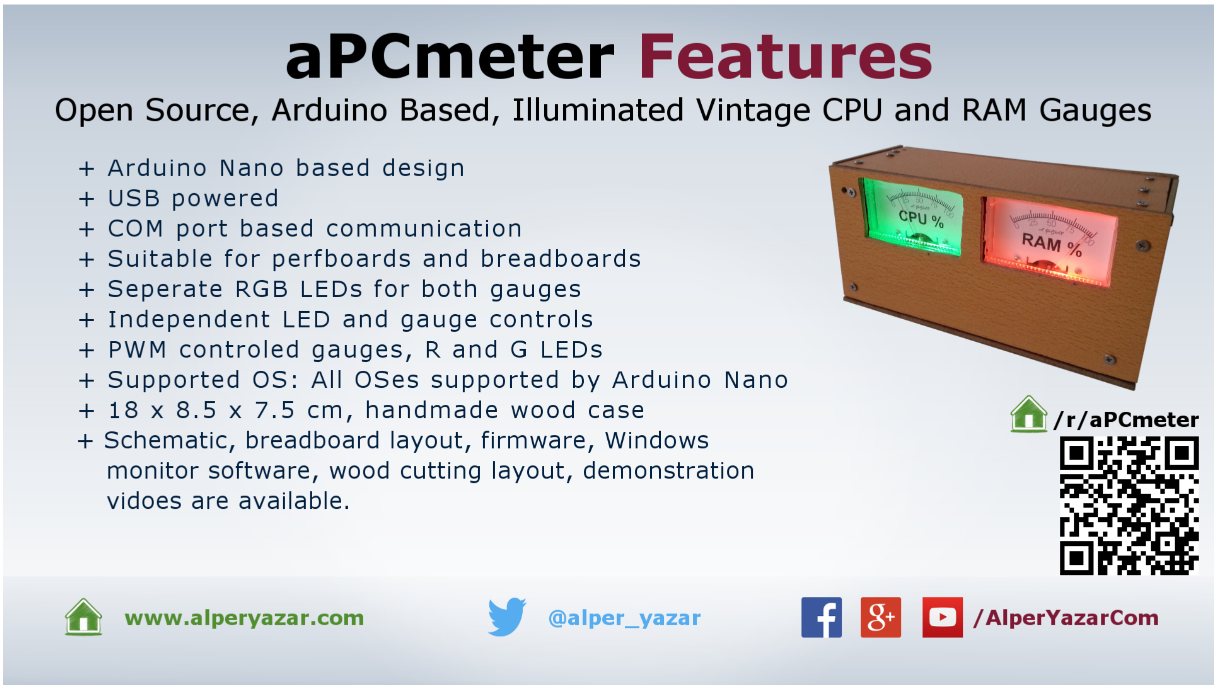
Contents

1 What is aPCmeter?

aPCmeter is Arduino Nano 3.0 based device which shows CPU and RAM usage of PC in a vintage looking way. Here there are some features of the board. Although I included some photos of the hardware, this document is dedicated to the Arduino firmware.

Note

For the most up-to-date information, please check the project webpage `http://www.alperyazar.↩com/r/aPCmeter`



aPCmeter Features

Open Source, Arduino Based, Illuminated Vintage CPU and RAM Gauges

+ Arduino Nano based design
+ USB powered
+ COM port based communication
+ Suitable for perfboards and breadboards
+ Seperate RGB LEDs for both gauges
+ Independent LED and gauge controls
+ PWM controled gauges, R and G LEDs
+ Supported OS: All OSes supported by Arduino Nano
+ 18 x 8.5 x 7.5 cm, handmade wood case
+ Schematic, breadboard layout, firmware, Windows monitor software, wood cutting layout, demonstration videoes are available.

/r/aPCmeter

www.alperyazar.com

@alper_yazar

/AlperYazarCom

Since my wood working ability is very limited, it has a crude looking wood case.

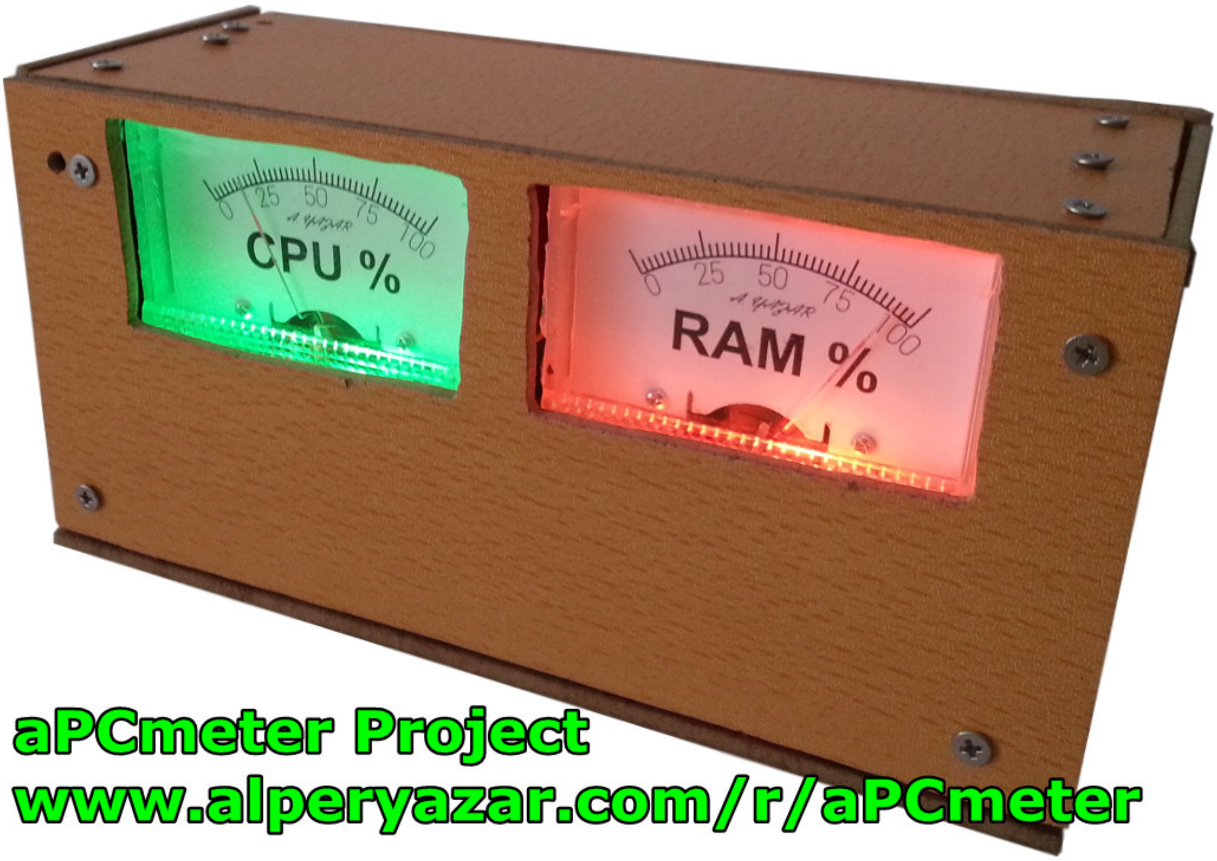
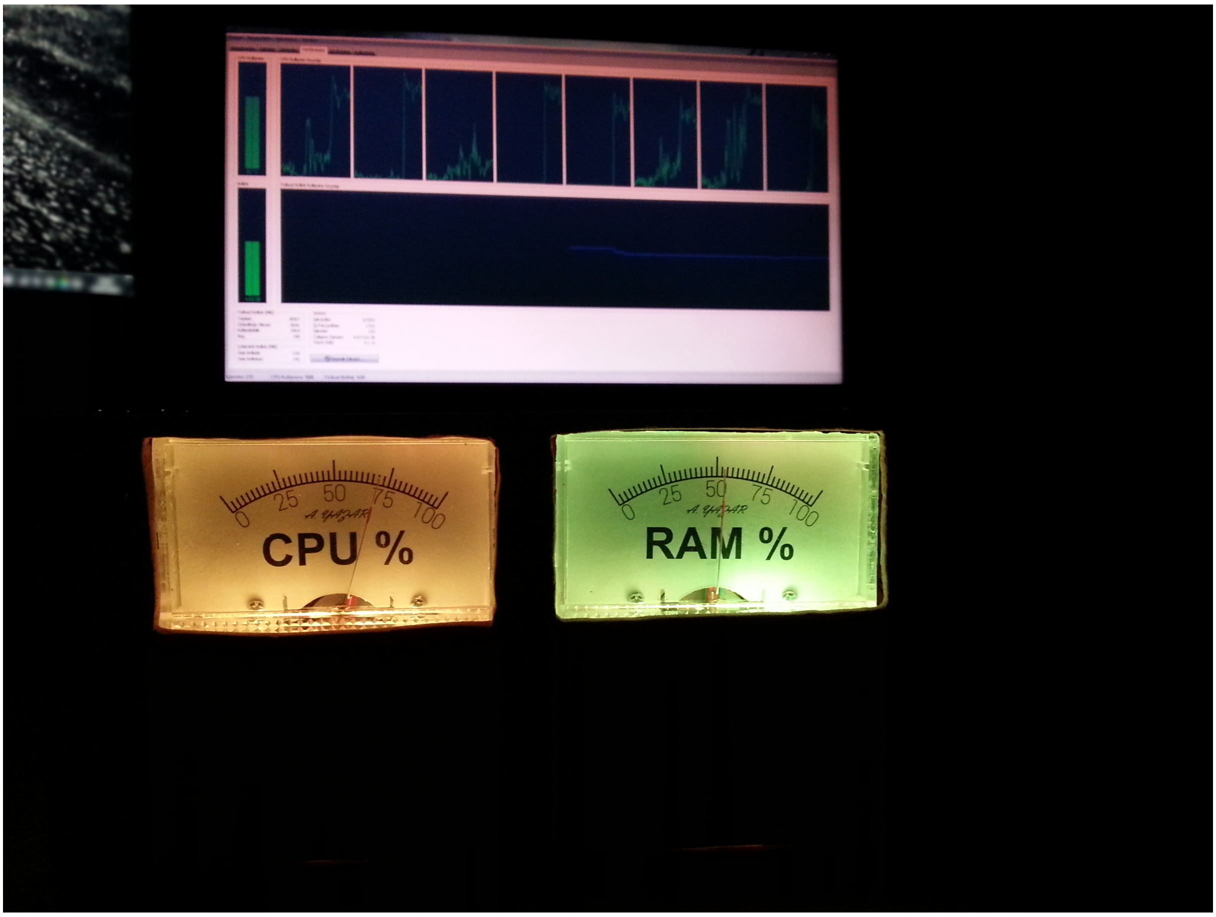
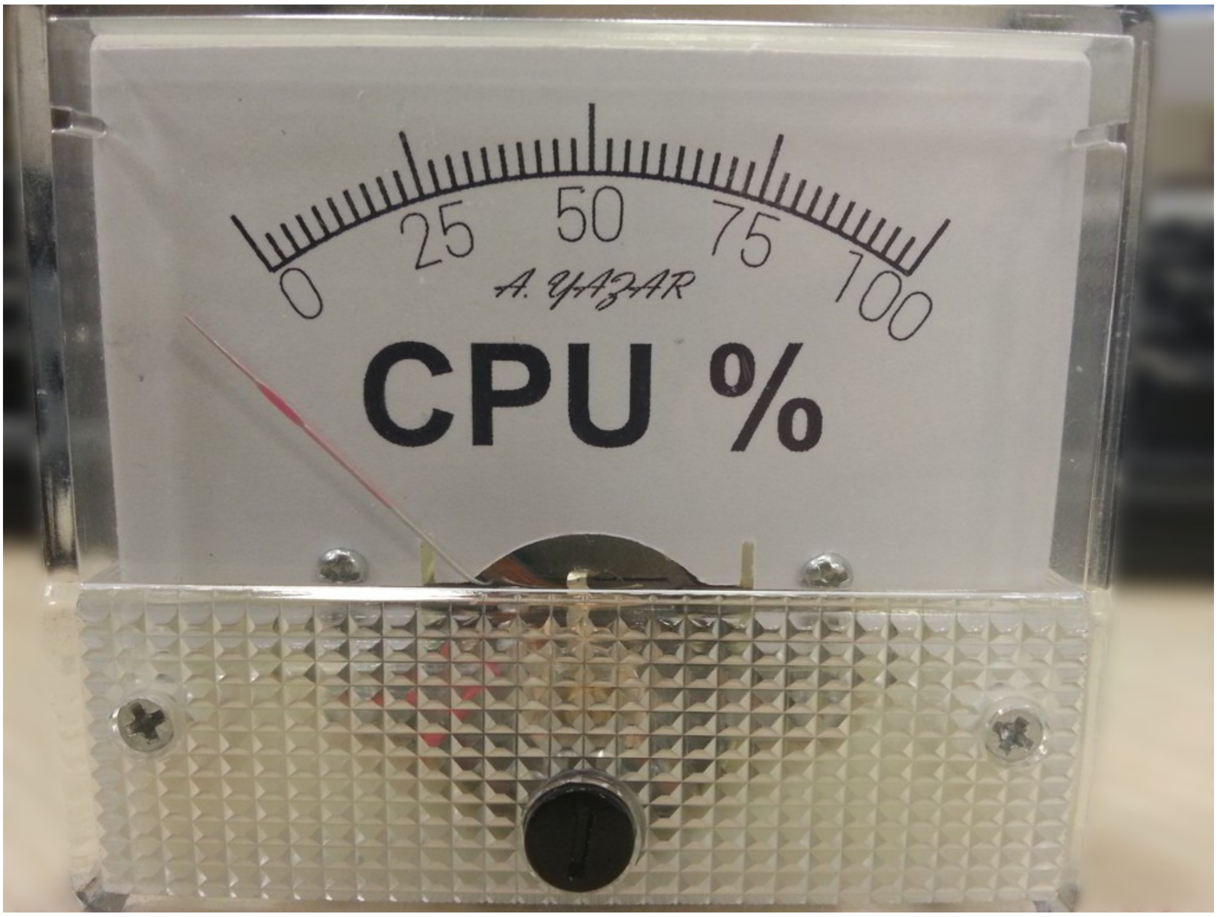
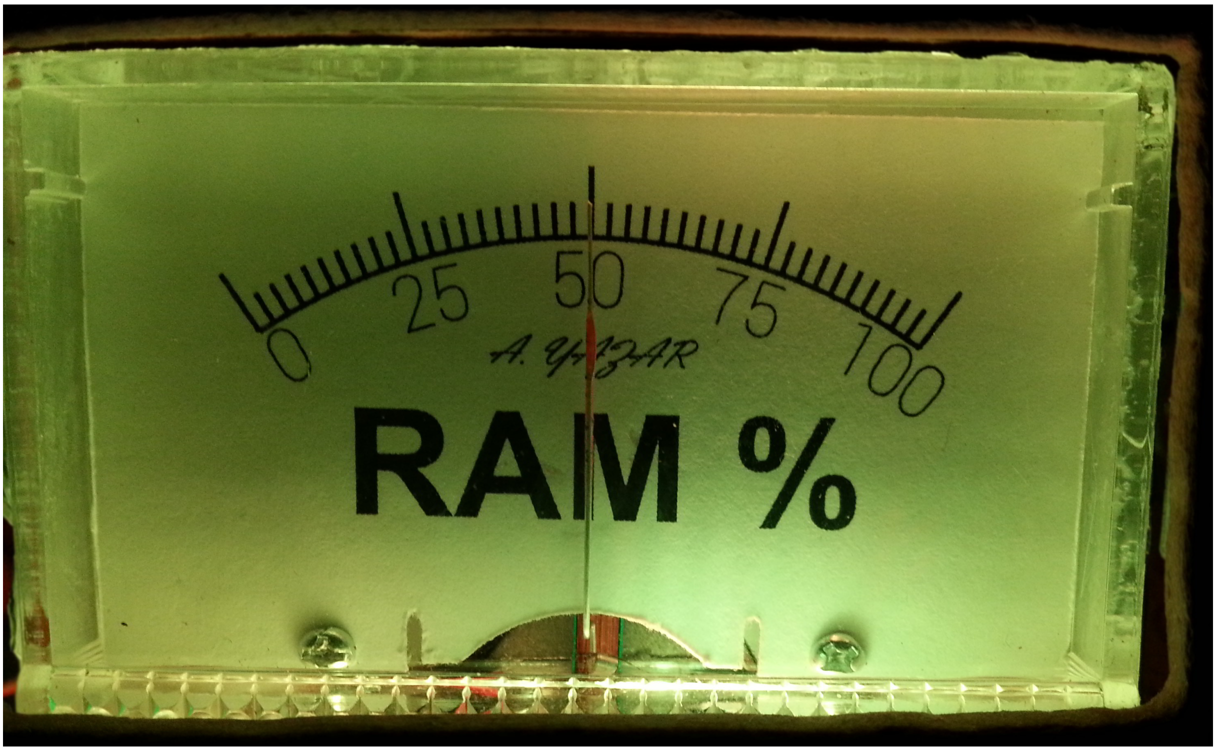


Figure 1 My Ugly aPCmeter


0
25
50
75
100
A. YAZAR
CPU %
0
25
50
75
100
A. YAZAR
RAM %





1.1 The Overall System Structure

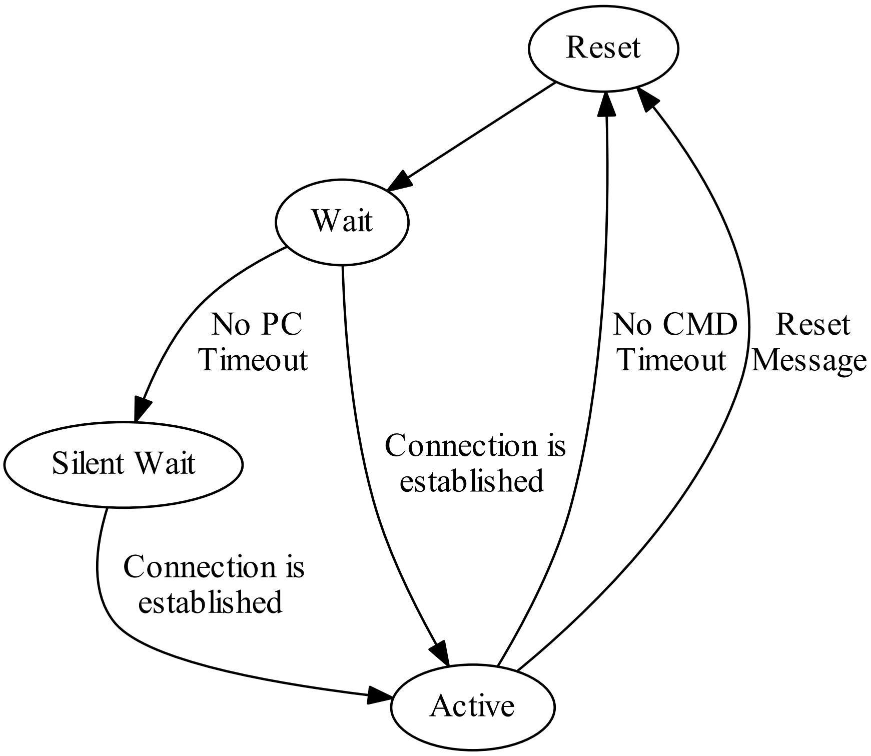
The overall structure is very simple. A software running on PC computes CPU and RAM usage percentages. Notice that both CPU and RAM gauges have lighting. Each gauges has its own RGB LED. For example if CPU usage is low, its gauge is illuminated by only green LED. As CPU usage increases, it is illuminated by both red and green LED resulting a yellow light. In addition to CPU and RAM usage information, PC software calculates lighting. Gauge position and lighting information are independent for Arduino software. PC software sends data over USB based virtual COM port and Arduino displays them. This is how Arduino code works basically.

Note

For available PC software, please check the project webpage `http://www.alperyazar.com/r/aPCmeter` Wtihout a proper PC software, aPCmeter hardware can't work properly.

2 State Diagram

State diagram is shown in the following figure.



Explanations of the states are given as below.

State	Explanation
Reset	This is the state after power up. Also setting the program counter (PC) to zero will reset the CPU. **See also** resetFunc()

State	Explanation
Wait	aPCmeter enters this state after Reset immediately. In this state, "aPCmeter\r\n" message is sent over serial channel. After the message is sent, aPCmeter waits a response from PC. If no response is received within 2 seconds, 4 pseudo random numbers are generated for PWM values of red L↵ ED of CPU gauge, green LED of CPU gauge, red LED of RAM gauge and green LED of RAM gauge. Similarly, 2 pseudo random binary numbers are generated for blue LED of CPU gauge and blue LED of RAM gauge. Using this generated numbers, gauges are illuminated in a pseudo random manner. Gauges stay unpowered completely. Then, the message is sent again. This loop continues for 60 seconds at most. If a valid message is received from the PC, connection is considered to be established. If no valid message is received within 60 seconds, *no PC timeout* is occured and the state is changed from Wait to Silent Wait. **See also** _NO_PC_TIMEOUT _UART_TIMEOUT_ms wait_for_connection().
Silent Wait	This state is very similar to Wait state. The differences are as in follows. In Silent Wait state, green and blue LEDs of both gauges become off. Brightness of both red LEDs are changed at every 2 seconds pseudo randomly. As in Wait state, "aPCmeter\r\n" message is sent over serial channel and a valid response is expected. However, Silent Wait state continues until a valid response is received. Notice that Wait and Silent Wait states are almost the same. Purpose of the Wait state is to indicate that aPCmeter is running and is waiting a connection response from PC. Also all LEDs are illuminated in pseudo random manner to test all of them. If aPCmeter is powerd up but no PC software gives a response, changing colors of gauges continuously may disturb people around aPCmeter. Therefore aPCmeter enters in Silen Wait mode after 60 seconds. If you look at directly, you may see that it still tries to connect to PC. **See also** wait_for_connection()
Active	This is the state where aPCmeter does its job. After connection is established, aPCmeter works in active state until power is removed. The serial communciation protocol between Arduino and PC is given in the following chapters. In this state, maximum duration between two consecutive successful command should be less than 2 seconds. If aPCmeter doesn't receive a successful command within 60 seconds, it resets itself. **See also** loop() _NO_CMD_TIMEOUT _UART_TIMEOUT_ms

3 UART Protocol

UART protocol works on 9600/8-N-1.

aPCmeter is a command driven device. In other words, PC software sends commands to aPCmeter, it does its job and acknowledges the PC software. Next command should send after the previous one is acknowledged. Each command starts with a capital letter followed by data of command and ends with small case of the starting letter. If it is desired after the last small case letter 'E' character can be send. E stands for end. When aPCmeter receives the 'E' character, it starts to parse the command immediately. If no 'E' is sent after the command, aPCmeter may wait at most 2 seconds before start to process the sent command. It is advised to end all comands with 'E' to trigger the command processing.

Here is the list of all possible messages:

Hello Mesage (Dir: aPCmeter -> PC)

A hello message is sent from aPCmeter when it is in Wait or Silent Wait state periodically. Here is the message: "aPCmeter\r\n", total of 10 bytes. If PC software ready, it should respond with Start Message.

Start Message (Dir: PC -> aPCmeter)

This message is meaningful when aPCmeter is in Wait or Silent Wait state. It should be send from the PC when aPCmeter sends the Hello Message. It consists of 2 (+1 optional ending) bytes

Byte 1	Byte 2	Byte 3
'S'	's'	'E' (Optional)

Response from the aPCmeter:

Byte 1	Byte 2	Byte 3	Byte 4
'O'	'K'	'\r'	'\n'

After the response, LEDs of both gauges stay at green. aPCmeter enters to the Active state and the all following commands may be send without any particular order. In Active state, the PC software should send the following commands with period no less than 2 seconds.

PWM Message (Dir: PC -> aPCmeter)

This is the main message to use aPCmeter. It is used to change PWM values of gauges and LEDs. Here is the structure

Byte Number	Message
1	'P'
2-4	PWM value of red LED of CPU gauge
5-7	PWM value of green LED of CPU gauge
8-10	PWM value of CPU gauge
11-13	PWM value of red LED of RAM gauge
14-16	PWM value of green LED of RAM gauge
17-19	PWM value of RAM gauge
20	'p'
21	'E' (Optional)

Each PWM value is 3 byte length string without null termination like "100", "001", "245". Values should between 0 and 255 (including the both limits) otherwise they are truncated. All PWM values are updated with this command. This is the way to change color and position of gauges.

For LEDs, "000" corresponds to full bright LED and "255" corresponds to completely off LED. For gauges, it is the opposite. "000" corresponds to no movement whereas "255" corresponds to full deflection.

Response from the aPCmeter:

Byte 1	Byte 2	Byte 3	Byte 4
'O'	'K'	'\r'	'\n'

Test Message(Dir: PC -> aPCmeter)

It is used to start built-in self-test routine. In this routine both gauges are moved slowly and color of gauges are changed slowly. This may be used to check functional operation of aPCmeter.

Byte 1	Byte 2	Byte 3
'T'	't'	'E' (Optional)

Response from the aPCmeter after self-test:

Byte 1	Byte 2	Byte 3	Byte 4
'O'	'K'	'\r'	'\n'

Blue Message(Dir: PC -> aPCmeter)

Blue LEDs are not PWM driven and they are not controlled by the PWM command. There are seperate command for blue LEDs just to turn them on or off seperately.

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
'B'	'B'	CPU Blue LED	RAM Blue LED	'E' (Optional)

x Blue LED: Controls the blue LED of the x gauge. If the corresponding byte is '1', the corresponding blue LED becomes on. Similarly, if the byte is '0' the corresponding LED becomes off.

Response from the aPCmeter:

Byte 1	Byte 2	Byte 3	Byte 4
'O'	'K'	'\r'	'\n'

Reset Message(Dir: PC -> aPCmeter)

It is used to reset the aPCmeter. State is changed from Active to Reset after this command.

Byte 1	Byte 2	Byte 3
'R'	'r'	'E' (Optional)

Response from the aPCmeter:

Byte 1	Byte 2	Byte 3	Byte 4
'O'	'K'	'\r'	'\n'

Keep Alive Message(Dir: PC -> aPCmeter)

It is used to avoid *No CMD Timouet*. As stated previously, PC software should send messages to aPCmeter periodically to avoid timeout. If PC software has nothing to say, it can just send Keep Alive Message. It does nothing except avoiding timeout.

Byte 1	Byte 2	Byte 3
'K'	'k'	'E' (Optional)

Response from the aPCmeter:

Byte 1	Byte 2	Byte 3	Byte 4
'O'	'K'	'\r'	'\n'

4 Bug List

File aPCmeter.ino

Visit: https://github.com/alperyazar/aPCmeter/issues to check existing bugs or to report a new one.

5 File Index

5.1 File List

Here is a list of all files with brief descriptions:

6 File Documentation

6.1 aPCmeter.ino File Reference

aPCmeter Project, source code of Arduino firmware.

Macros

- #define _NO_CMD_TIMEOUT 29

  Determines the maximum trials between two consecutive command reception in active state before reset.
- #define _NO_PC_TIMEOUT 29

  Determines the duration of wait state after reset.
- #define _UART_TIMEOUT_ms 2000

  Receive timeout value in msec for serial channel receive function.
- #define _UART_BAUD_RATE 9600

  Baud rate for serial communication in bps.

Functions

- void setup ()

  This function is called after startup. Like a well known main() function in C.
- void wait_for_connection ()

  Wait proper connection initialization from PC.
- void loop ()

  This function is automatically after setup(). Loops forever...
- int str2duty (const char ∗str)

  Converted string to integer. (Ex: "105" -> 105)
- void set_duty (int channel, int duty)

  Set duty cycle for red, green LED and gauge for a channel.
- void self_test ()

  BIST (built-in self-test)

Variables

- const int redPinCPU = 3

  Pin number of red LED of CPU gauge.
- const int greenPinCPU = 5

  Pin number of green LED of CPU gauge.
- const int bluePinCPU = 2

  Pin number of blue LED of CPU gauge.
- const int gaugePinCPU = 6

  Pin number of CPU gauge.
- const int redPinRAM = 9

  Pin number of red LED of RAM gauge.
- const int greenPinRAM = 10

  Pin number of green LED of RAM gauge.
- const int bluePinRAM = 19

  Pin number of blue LED of RAM gauge.
- const int gaugePinRAM = 11

  Pin number of RAM gauge.
- char RX_buf [64]

  Receive buffer for serial channel.
- void(∗ resetFunc)(void)=0

  Tricky reset function. Just set PC to 0.

6.1.1 Detailed Description

aPCmeter Project, source code of Arduino firmware.

This single file contains all source code of aPCmeter Arduino Nano firmware. It is tested by using Arduino IDE v1.6.7 on Windows 7 x64.

Author

Alper Yazar

Version

s1

Date

2016-05-08

Copyright

CC BY-NC-SA 4.0 (http://creativecommons.org/licenses/by-nc-sa/4.0/)

Bug Visit: https://github.com/alperyazar/aPCmeter/issues to check existing bugs or to report a new one.

Warning

NO WARRANTY! AS IS!

See also

The Official Project Page: http://www.alperyazar.com/r/aPCmeter

Definition in file aPCmeter.ino.

6.1.2 Macro Definition Documentation

6.1.2.1 #define _NO_CMD_TIMEOUT 29

Determines the maximum trials between two consecutive command reception in active state before reset.

When aPCmeter is in active state, it waits commands from PC. PC software should send commands with period less than _UART_TIMEOUT_ms to avoid time out. If no valid message is received within _UART_TIMEOUT_ms msec after the last successfully received command, a timeout counter is incremented by one. If value of this counter exceeds _NO_CMD_TIMEOUT, aPCmeter resets itself.

See also

loop()

```
Maximum duration between two successful message before reset in msec =
(#_NO_CMD_TIMEOUT + 1) x #_UART_TIMEOUT_ms
```

Warning

It should be a non-negative integer.

Definition at line 22 of file aPCmeter.ino.

Referenced by loop().

6.1.2.2 #define _NO_PC_TIMEOUT 29

Determines the duration of wait state after reset.

After reset, aPCmeter sends "aPCmeter" message periodically determined by _UART_TIMEOUT_ms. After each message it waits a valid message from PC for the first connection. At each time "aPCmeter" message is sent, a pseudo random color is selected for both CPU and RAM gauges and both gauges are illuminated with the selected color. This is used to indiciate that aPCmeter is looking for a PC connection and to provide pseudo random test pattern sequence for all three channels (R, G, B) of both LEDs. If the number of trials exceeds the threshold value determined by _NO_PC_TIMEOUT, aPCmeter enters to the silent wait state. In that state, G and B LEDs of both gauges become off. Only brightness of R LEDs of both gauges is changed in a pseudo random manner. This is done to avoid disturbing color changes when all R, G, B LEDs are illuminated pseudo randomly. This may be the case when aPCmeter is left connected to the PC or another USB host device. However brightness of red LEDs continues to change and aPCmeter looks a PC connection for ever. One may still be able to notice that aPCmeter is active since brightness of red LEDs is changed periodically.

Note

Duration of wait state can be calculated as follow

```
Duration of wait state in msec =  (#_NO_PC_TIMEOUT + 1) x #_UART_TIMEOUT_ms
```

Warning

It should be a non-negative integer.

See also

wait_for_connection()

Definition at line 32 of file aPCmeter.ino.

Referenced by wait_for_connection().

6.1.2.3 #define _UART_BAUD_RATE 9600

Baud rate for serial communication in bps.

See also

setup()

Definition at line 46 of file aPCmeter.ino.

Referenced by setup().

6.1.2.4 #define _UART_TIMEOUT_ms 2000

Receive timeout value in msec for serial channel receive function.

aPCmeter waits at most _UART_TIMEOUT_ms msec when serial read functions are called. If no message is received from PC in _UART_TIMEOUT_ms msec, serial receive functions are timed out and code continues. Actually this is basic "tic" source for this code.

Warning

It should be a non-negative integer.

See also

setup()

Definition at line 40 of file aPCmeter.ino.

Referenced by setup().

6.1.3 Function Documentation

6.1.3.1 void loop ( )

This function is automatically after setup(). Loops forever...

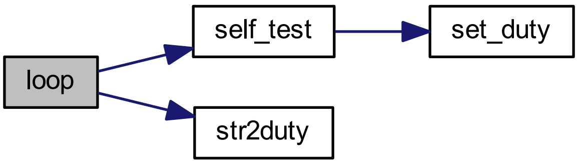
See also

`https://www.arduino.cc/en/Reference/Loop`

Definition at line 199 of file aPCmeter.ino.

References _NO_CMD_TIMEOUT, bluePinCPU, bluePinRAM, gaugePinCPU, gaugePinRAM, greenPinCPU, greenPinRAM, redPinCPU, redPinRAM, resetFunc, RX_buf, self_test(), and str2duty().

Here is the call graph for this function:



6.1.3.2 void self_test ( )

BIST (built-in self-test)

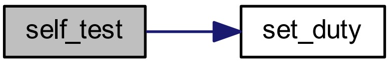
Sweep all LEDs and gauges to make sure that everything is working well.

Definition at line 334 of file aPCmeter.ino.

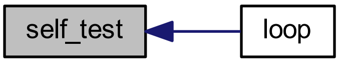
References bluePinCPU, bluePinRAM, gaugePinCPU, gaugePinRAM, greenPinCPU, greenPinRAM, redPinCPU, redPinRAM, and set_duty().

Referenced by loop().

Here is the call graph for this function:



Here is the caller graph for this function:



6.1.3.3 void set_duty (int *channel*, int *duty*)

Set duty cycle for red, green LED and gauge for a channel.

It is used by BIST (built-in self-test)

Parameters

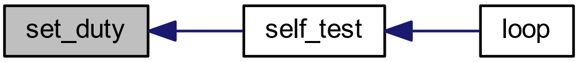
in	*channel*	0 means CPU gauge, 1 means RAM gauge
in	*duty*	It should be between 0 and 255 (including both). Otherwise, it will be truncated into this range. It is the PWM value for analogWrite() function. 0 corresponds to 0% for gauge, 100% for green and 0% for red.

Definition at line 307 of file aPCmeter.ino.

References gaugePinCPU, gaugePinRAM, greenPinCPU, greenPinRAM, redPinCPU, and redPinRAM.

Referenced by self_test().

Here is the caller graph for this function:



6.1.3.4 void setup ( )

This function is called after startup. Like a well known main() function in C.

See also

`https://www.arduino.cc/en/Reference/Setup`

Definition at line 111 of file aPCmeter.ino.

References _UART_BAUD_RATE, _UART_TIMEOUT_ms, bluePinCPU, bluePinRAM, gaugePinCPU, gaugePin↩ RAM, greenPinCPU, greenPinRAM, redPinCPU, redPinRAM, and wait_for_connection().

Here is the call graph for this function:

setup
wait_for_connection

6.1.3.5 int str2duty (const char ∗ *str*)

Converted string to integer. (Ex: "105" -> 105)

PWM information for LEDs and gauges are sent as a text formatted as 3 digit decimal number like "000", "104", "205". This function is used to 3 digit decimal number to its corresponding integere value.

Parameters

`in`	*str*	It is a pointer to the string that is converted to integer. It should be 3 digit unsigned decimal value. Null termination doesn't affect the behaviour.

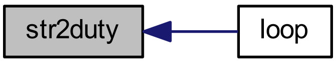
Return values

Integer	value is returned. Value is truncated in between 0 and 255.

Definition at line 289 of file aPCmeter.ino.

Referenced by loop().

Here is the caller graph for this function:



6.1.3.6 void wait_for_connection ( )

Wait proper connection initialization from PC.

Definition at line 146 of file aPCmeter.ino.

References _NO_PC_TIMEOUT, bluePinCPU, bluePinRAM, greenPinCPU, greenPinRAM, redPinCPU, redPinRAM, and RX_buf.

Referenced by setup().

Here is the caller graph for this function:

wait_for_connection
setup

6.1.4 Variable Documentation

6.1.4.1 const int bluePinCPU = 2

Pin number of blue LED of CPU gauge.

See also

`http://playground.arduino.cc/Learning/Pins`

Definition at line 64 of file aPCmeter.ino.

Referenced by loop(), self_test(), setup(), and wait_for_connection().

6.1.4.2 const int bluePinRAM = 19

Pin number of blue LED of RAM gauge.

See also

http://playground.arduino.cc/Learning/Pins

Definition at line 88 of file aPCmeter.ino.

Referenced by loop(), self_test(), setup(), and wait_for_connection().

6.1.4.3 const int gaugePinCPU = 6

Pin number of CPU gauge.

See also

http://playground.arduino.cc/Learning/Pins

Definition at line 70 of file aPCmeter.ino.

Referenced by loop(), self_test(), set_duty(), and setup().

6.1.4.4 const int gaugePinRAM = 11

Pin number of RAM gauge.

See also

http://playground.arduino.cc/Learning/Pins

Definition at line 94 of file aPCmeter.ino.

Referenced by loop(), self_test(), set_duty(), and setup().

6.1.4.5 const int greenPinCPU = 5

Pin number of green LED of CPU gauge.

See also

http://playground.arduino.cc/Learning/Pins

Definition at line 58 of file aPCmeter.ino.

Referenced by loop(), self_test(), set_duty(), setup(), and wait_for_connection().

6.1.4.6 const int greenPinRAM = 10

Pin number of green LED of RAM gauge.

See also

http://playground.arduino.cc/Learning/Pins

Definition at line 82 of file aPCmeter.ino.

Referenced by loop(), self_test(), set_duty(), setup(), and wait_for_connection().

6.1.4.7 const int redPinCPU = 3

Pin number of red LED of CPU gauge.

See also

http://playground.arduino.cc/Learning/Pins

Definition at line 52 of file aPCmeter.ino.

Referenced by loop(), self_test(), set_duty(), setup(), and wait_for_connection().

6.1.4.8 const int redPinRAM = 9

Pin number of red LED of RAM gauge.

See also

http://playground.arduino.cc/Learning/Pins

Definition at line 76 of file aPCmeter.ino.

Referenced by loop(), self_test(), set_duty(), setup(), and wait_for_connection().

6.1.4.9 void(∗ resetFunc) (void)=0

Tricky reset function. Just set PC to 0.

Definition at line 105 of file aPCmeter.ino.

Referenced by loop().

6.1.4.10 char RX_buf[64]

Receive buffer for serial channel.

More than enough actually...

Definition at line 100 of file aPCmeter.ino.

Referenced by loop(), and wait_for_connection().

6.2 aPCmeter.ino

```

#define _NO_CMD_TIMEOUT 29

#define _NO_PC_TIMEOUT 29

#define _UART_TIMEOUT_ms 2000

#define _UART_BAUD_RATE 9600

const int redPinCPU = 3;

const int greenPinCPU = 5;

const int bluePinCPU = 2;

const int gaugePinCPU = 6;

const int redPinRAM = 9;

const int greenPinRAM = 10;

const int bluePinRAM = 19;

const int gaugePinRAM = 11;

char RX_buf[64];

void(* resetFunc) (void) = 0;

void setup()
{
  pinMode(gaugePinCPU, INPUT);
  pinMode(gaugePinRAM, INPUT);

  digitalWrite(redPinCPU, HIGH);
  digitalWrite(greenPinCPU, HIGH);
  digitalWrite(bluePinCPU, HIGH);

  digitalWrite(redPinRAM, HIGH);
  digitalWrite(greenPinRAM, HIGH);
  digitalWrite(bluePinRAM, HIGH);

  pinMode(redPinCPU, OUTPUT);
  pinMode(greenPinCPU, OUTPUT);
  pinMode(bluePinCPU, OUTPUT);

  pinMode(redPinRAM, OUTPUT);
  pinMode(greenPinRAM, OUTPUT);
  pinMode(bluePinRAM, OUTPUT);

  //All LEDs are off. Gauges are off.

  // Set properties of Serial's.
  Serial.setTimeout(_UART_TIMEOUT_ms);
  Serial.begin(_UART_BAUD_RATE);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for native USB
  }
  wait_for_connection();
}

void wait_for_connection()
{
  int timeout = 0;

  // Try to randomize pseudo-random sequences used by random() funtion.
  randomSeed(analogRead(0));
  while (1)
  {
    Serial.println("aPCmeter");
    if ( Serial.readBytesUntil('E', RX_buf, 64) == 0)
    {
      analogWrite(redPinCPU, random(256));
      analogWrite(redPinRAM, random(256));
      analogWrite(greenPinRAM, random(256));
      analogWrite(greenPinCPU, random(256));
      digitalWrite(bluePinCPU, random(2));
      digitalWrite(bluePinRAM, random(2));
      timeout++;
      if (timeout > _NO_PC_TIMEOUT)
      {
        timeout--;
        digitalWrite(greenPinCPU, HIGH);
        digitalWrite(bluePinCPU, HIGH);
        digitalWrite(greenPinRAM, HIGH);
        digitalWrite(bluePinRAM, HIGH);
        analogWrite(redPinRAM, 128 + random(128));
        analogWrite(redPinCPU, 128 + random(128));
      }
    }
    else
    {
      if ((RX_buf[0] == ’S’) && (RX_buf[1] == ’s’))
      {
        timeout = 0;
        digitalWrite(redPinCPU, HIGH);
        digitalWrite(redPinRAM, HIGH);
        digitalWrite(bluePinCPU, HIGH);
        digitalWrite(bluePinRAM, HIGH);
        digitalWrite(greenPinCPU, LOW);
        digitalWrite(greenPinRAM, LOW);
        // Both gauges are fully green.
        Serial.println("OK");
        return;
      }
      memset(RX_buf, ’ ’, 64); //Clear all buffer
    }
  }
}

void loop()
{
  int timeout_ctr = 0;
  while (1)
  {
    if ( Serial.readBytesUntil(’E’, RX_buf, 64) == 0)
    {
      timeout_ctr++;
      if (timeout_ctr > _NO_CMD_TIMEOUT)
      {
        resetFunc();
      }
    }
    else
    {
      switch (RX_buf[0])
      {
        case ’P’:
          if (RX_buf[19] == ’p’)
          {
            analogWrite(redPinCPU, str2duty(&(RX_buf[1])));
            analogWrite(greenPinCPU, str2duty(&(RX_buf[4])));
            analogWrite(gaugePinCPU, str2duty(&(RX_buf[7])));
            analogWrite(redPinRAM, str2duty(&(RX_buf[10])));
            analogWrite(greenPinRAM, str2duty(&(RX_buf[13])));
            analogWrite(gaugePinRAM, str2duty(&(RX_buf[16])));
            Serial.println("OK");
            timeout_ctr = 0;
          }
          break;
        case ’T’:
          if (RX_buf[1] == ’t’)
          {
            self_test();
            Serial.println("OK");
            timeout_ctr = 0;
          }
        case ’B’:
          if (RX_buf[3] == ’b’)
          {
            if (RX_buf[1] == ’1’)
            {
              digitalWrite(bluePinCPU, LOW);
            }
            if (RX_buf[1] == ’0’)
            {
              digitalWrite(bluePinCPU, HIGH);
            }
            if (RX_buf[2] == ’1’)
            {
              digitalWrite(bluePinRAM, LOW);
            }
            if (RX_buf[2] == ’0’)
            {
              digitalWrite(bluePinRAM, HIGH);
            }
            Serial.println("OK");
            timeout_ctr = 0;
          }
        case ’R’:
          if (RX_buf[1] == ’r’)
          {
            Serial.println("OK");
            timeout_ctr = 0;
            Serial.flush();
            resetFunc();
          }
        case ’K’:
          if (RX_buf[1] == ’k’)
          {
            Serial.println("OK");
            timeout_ctr = 0;
          }
        default:
            timeout_ctr++;
          break;
      }
      memset(RX_buf, ’ ’, 64);
    }

  }
}

int str2duty(const char *str)
{
  int temp = 0;

  temp = 100 * (str[0] - 0x30);
  temp += 10 * (str[1] - 0x30);
  temp += 1 * (str[2] - 0x30);

  return (constrain(temp, 0, 255));

}

void set_duty(int channel, int duty)
{
  // channel 0-> CPU, 1-> RAM

  duty = constrain(duty, 0, 255);
  if (channel == 0)
  {
    analogWrite(gaugePinCPU, duty);
    analogWrite(greenPinCPU, duty);
    analogWrite(redPinCPU, 255 - duty);
  }
  else if (channel == 1)
  {
    analogWrite(gaugePinRAM, duty);
    analogWrite(greenPinRAM, duty);
    analogWrite(redPinRAM, 255 - duty);
  }
  else
  {
  }

}

void self_test()
{
  int duty = 0;
  pinMode(gaugePinCPU, INPUT);
  pinMode(gaugePinRAM, INPUT);
  delay(3000); //Gauges will relax in 3 seconds.
  //Just sweep
  for (duty = 0; duty < 255; duty++)
  {
    analogWrite(gaugePinCPU, duty);
    analogWrite(gaugePinRAM, duty);
    if ( (0 <= duty) && (duty < 85))
    {
      digitalWrite(redPinCPU, HIGH);
      digitalWrite(greenPinCPU, HIGH);
      digitalWrite(bluePinCPU, LOW);

      digitalWrite(redPinRAM, HIGH);
      digitalWrite(greenPinRAM, HIGH);
      digitalWrite(bluePinRAM, LOW);
    }
    if ( (85 <= duty) && (duty < 170))
    {
      digitalWrite(redPinCPU, HIGH);
      digitalWrite(bluePinCPU, HIGH);
      digitalWrite(greenPinCPU, LOW);

      digitalWrite(redPinRAM, HIGH);
      digitalWrite(bluePinRAM, HIGH);
      digitalWrite(greenPinRAM, LOW);

    }
    if ( (170 <= duty) && (duty < 255))
    {
      digitalWrite(greenPinCPU, HIGH);
      digitalWrite(bluePinCPU, HIGH);
      digitalWrite(redPinCPU, LOW);

      digitalWrite(bluePinRAM, HIGH);
      digitalWrite(greenPinRAM, HIGH);
      digitalWrite(redPinRAM, LOW);

    }
    delay(30);
  }
  for (duty = 255; duty > 0; duty--)
  {
    set_duty(0, duty);
    set_duty(1, duty);
    delay(10);
  }
  pinMode(gaugePinCPU, INPUT);
  pinMode(gaugePinRAM, INPUT);
  delay(1000);
}

```

6.3 main.md File Reference

6.4 main.md

```
# What is aPCmeter? # {#mainpage}

aPCmeter is Arduino Nano 3.0 based device which shows CPU and RAM usage of PC in a vintage looking way. Here there are some features of the board. Although I included some photos of the hardware, this document is dedicated to the Arduino firmware.

@note For the most up-to-date information, please check the project webpage http://www.alperyazar.com/r/aPCmeter

@image latex aPCmeter_Promo.png

Since my wood working ability is very limited, it has a crude looking wood case.

@image latex aPCmeter_white_annotated_1024_721.jpg My Ugly aPCmeter
@image latex  aPCmeter_1.jpg
@image latex  aPCmeter_2.jpg
@image latex  aPCmeter_6.jpg

# The Overall System Structure # {#the_overall_system_structure}

The overall structure is very simple. A software running on PC computes CPU and RAM usage percentages. Notice that both CPU and RAM gauges have lighting. Each gauges has its own RGB LED. For example if CPU usage is low, its gauge is illuminated by only green LED. As CPU usage increases, it is illuminated by both red and green LED resulting a yellow light. In addition to CPU and RAM usage information, PC software calculates lighting. Gauge position and lighting information are independent for Arduino software. PC software sends data over USB based virtual COM port and Arduino displays them. This is how Arduino code works basically.

@note For available PC software, please check the project webpage http://www.alperyazar.com/r/aPCmeter Wtihout a proper PC software, aPCmeter hardware can't work properly.
```

6.5 state_diagram.md File Reference

6.6 state_diagram.md

```
# State Diagram # {#states}

State diagram is shown in the following figure.

@dotfile state_diagram.gv

Explanations of the states are given as below.

State  | Explanation
```

```
------------- | -------------
Reset  | This is the state after power up. Also setting the program counter (PC) to zero will reset the CPU. @see resetFunc()
Wait  | aPCmeter enters this state after Reset immediately. In this state, "aPCmeter\r\n" message is sent over serial channel. After the message is sent, aPCmeter waits a response from PC. If no response is received within 2 seconds, 4 pseudo random numbers are generated for PWM values of red LED of CPU gauge, green LED of CPU gauge, red LED of RAM gauge and green LED of RAM gauge. Similarly, 2 pseudo random binary numbers are generated for blue LED of CPU gauge and blue LED of RAM gauge. Using this generated numbers, gauges are illuminated in a pseudo random manner. Gauges stay unpowered completely. Then, the message is sent again. This loop continues for 60 seconds at most. If a valid message is received from the PC, connection is considered to be established. If no valid message is received within 60 seconds, _no PC timeout_ is occured and the state is changed from Wait to Silent Wait. @see #_NO_PC_TIMEOUT @see #_UART_TIMEOUT_ms @see wait_for_connection().
Silent Wait | This state is very similar to Wait state. The differences are as in follows. In Silent Wait state, green and blue LEDs of both gauges become off. Brightness of both red LEDs are changed at every 2 seconds pseudo randomly. As in Wait state, "aPCmeter\r\n" message is sent over serial channel and a valid response is expected. However, Silent Wait state continues until a valid response is received. Notice that Wait and Silent Wait states are almost the same. Purpose of the Wait state is to indicate that aPCmeter is running and is waiting a connection response from PC. Also all LEDs are illimunated in pseudo random manner to test all of them. If aPCmeter is powerd up but no PC software gives a response, changing colors of gauges continuously may disturb people around aPCmeter. Therefore aPCmeter enters in Silen Wait mode after 60 seconds. If you look at directly, you may see that it still tries to connect to PC. @see wait_for_connection()
Active | This is the state where aPCmeter does its job. After connection is established, aPCmeter works in active state until power is removed. The serial communciation protocol between Arduino and PC is given in the following chapters. In this state, maximum duration between two consecutive successful command should be less than 2 seconds. If aPCmeter doesn't receive a successful command within 60 seconds, it resets itself. @see loop() @see #_NO_CMD_TIMEOUT @see #_UART_TIMEOUT_ms
```

6.7 uart_protocol.md File Reference

6.8 uart_protocol.md

```
# UART Protocol # {#uart}

UART protocol works on 9600/8-N-1.

aPCmeter is a command driven device. In other words, PC software sends commands to aPCmeter, it does its job and acknowledges the PC software. Next command should send after the previous one is acknowledged. Each command starts with a capital letter followed by data of command and ends with small case of the starting letter. If it is desired after the last small case letter 'E' character can be send. E stands for end. When aPCmeter receives the 'E' character, it starts to parse the command immediately. If no 'E' is sent after the command, aPCmeter may wait at most 2 seconds before start to process the sent command. It is advised to end all comands with 'E' to trigger the command processing.

Here is the list of all possible messages:

## Hello Mesage (Dir: aPCmeter -> PC)
A hello message is sent from aPCmeter when it is in Wait or Silent Wait state periodically. Here is the message: "aPCmeter\r\n", total of 10 bytes. If PC software ready, it should respond with Start Message.

## Start Message (Dir: PC -> aPCmeter)
This message is meaningful when aPCmeter is in Wait or Silent Wait state. It should be send from the PC when aPCmeter sends the Hello Message. It consists of 2 (+1 optional ending) bytes

Byte 1 | Byte 2 | Byte 3
------ | ------ | ----------------
'S'    | 's'    | 'E' (Optional)

Response from the aPCmeter:

Byte 1 | Byte 2 | Byte 3 | Byte 4
------ | ------ | ------ | ------
'O'    | 'K'    | '\\r'   | '\\n'

After the response, LEDs of both gauges stay at green. aPCmeter enters to the Active state and the all following commands may be send without any particular order. In Active state, the PC software should send the following commands with period no less than 2 seconds.

## PWM Message (Dir: PC -> aPCmeter)
This is the main message to use aPCmeter. It is used to change PWM values of gauges and LEDs. Here is the structure

Byte Number | Message |
------ | ------
1   | 'P'
2-4 | PWM value of red LED of CPU gauge
5-7 | PWM value of green LED of CPU gauge
8-10 | PWM value of CPU gauge
11-13 | PWM value of red LED of RAM gauge
14-16 | PWM value of green LED of RAM gauge
17-19 | PWM value of RAM gauge
```

```
20 | ’p’
21 | ’E’ (Optional)

Each PWM value is 3 byte length string without null termination like "100", "001", "245". Values should between 0 and 255 (including the both limits) otherwise they are truncated. All PWM values are updated with this command. This is the way to change color and position of gauges.

For LEDs, "000" corresponds to full bright LED and "255" corresponds to completely off LED. For gauges, it is the opposite. "000" corresponds to no movement whereas "255" corresponds to full deflection.

Response from the aPCmeter:

Byte 1 | Byte 2 | Byte 3 | Byte 4
------ | ------ | ------ | ------
’O’    | ’K’    | ’\\r’   | ’\\n’

## Test Message(Dir: PC -> aPCmeter)
It is used to start built-in self-test routine. In this routine both gauges are moved slowly and color of gauges are changed slowly. This may be used to check functional operation of aPCmeter.

Byte 1 | Byte 2 | Byte 3
------ | ------ | ----------------
’T’    | ’t’    | ’E’ (Optional)

Response from the aPCmeter after self-test:

Byte 1 | Byte 2 | Byte 3 | Byte 4
------ | ------ | ------ | ------
’O’    | ’K’    | ’\\r’   | ’\\n’

## Blue Message(Dir: PC -> aPCmeter)

Blue LEDs are not PWM driven and they are not controlled by the PWM command. There are seperate command for blue LEDs just to turn them on or off seperately.

Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5
------ | ------ | ------ | ------ | ------
’B’    | ’B’    | CPU Blue LED     | RAM Blue LED | ’E’ (Optional)

x Blue LED: Controls the blue LED of the x gauge. If the corresponding byte is ’1’, the corresponding blue LED becomes on. Similarly, if the byte is ’0’ the corresponding LED becomes off.

Response from the aPCmeter:

Byte 1 | Byte 2 | Byte 3 | Byte 4
------ | ------ | ------ | ------
’O’    | ’K’    | ’\\r’   | ’\\n’

## Reset Message(Dir: PC -> aPCmeter)
It is used to reset the aPCmeter. State is changed from Active to Reset after this command.

Byte 1 | Byte 2 | Byte 3
------ | ------ | ----------------
’R’    | ’r’    | ’E’ (Optional)

Response from the aPCmeter:

Byte 1 | Byte 2 | Byte 3 | Byte 4
------ | ------ | ------ | ------
’O’    | ’K’    | ’\\r’   | ’\\n’

## Keep Alive Message(Dir: PC -> aPCmeter)
It is used to avoid _No CMD Timouet_. As stated previously, PC software should send messages to aPCmeter periodically to avoid timeout. If PC software has nothing to say, it can just send Keep Alive Message. It does nothing except avoiding timeout.

Byte 1 | Byte 2 | Byte 3
------ | ------ | ----------------
’K’    | ’k’    | ’E’ (Optional)

Response from the aPCmeter:

Byte 1 | Byte 2 | Byte 3 | Byte 4
------ | ------ | ------ | ------
’O’    | ’K’    | ’\\r’   | ’\\n’
```

Index