

# aPCmeter Firmware

s1

Generated by Doxygen 1.8.11

Thu Jul 14 2016 21:05:04

## Contents

<b>1</b>	<b>What is aPCmeter?</b>	<b>2</b>
1.1	The Overall System Structure . . . . .	5
<b>2</b>	<b>State Diagram</b>	<b>6</b>
<b>3</b>	<b>UART Protocol</b>	<b>7</b>
<b>4</b>	<b>Bug List</b>	<b>10</b>
<b>5</b>	<b>File Index</b>	<b>10</b>
5.1	File List . . . . .	10
<b>6</b>	<b>File Documentation</b>	<b>10</b>
6.1	aPCmeter.ino File Reference . . . . .	10
6.1.1	Detailed Description . . . . .	11
6.1.2	Macro Definition Documentation . . . . .	12
6.1.3	Function Documentation . . . . .	14
6.1.4	Variable Documentation . . . . .	17
6.2	aPCmeter.ino . . . . .	20
6.3	main.md File Reference . . . . .	23
6.4	main.md . . . . .	23
6.5	state_diagram.md File Reference . . . . .	23
6.6	state_diagram.md . . . . .	23
6.7	uart_protocol.md File Reference . . . . .	24
6.8	uart_protocol.md . . . . .	24
	<b>Index</b>	<b>27</b>

## 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 vidoes are available.



 /r/aPCmeter

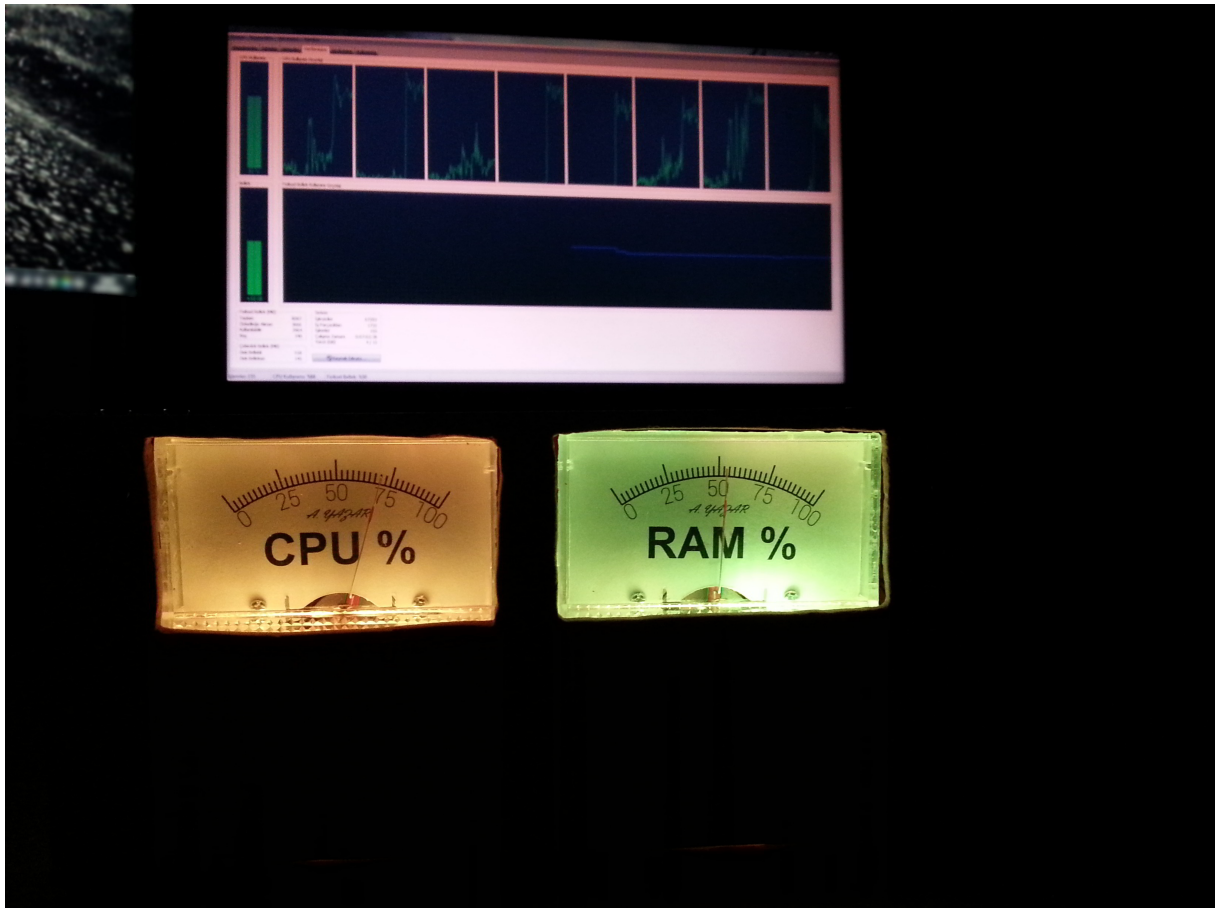
 [www.alperyazar.com](http://www.alperyazar.com)  [@alper\\_yazar](https://twitter.com/alper_yazar)    [/AlperYazarCom](https://www.youtube.com/AlperYazarCom)

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

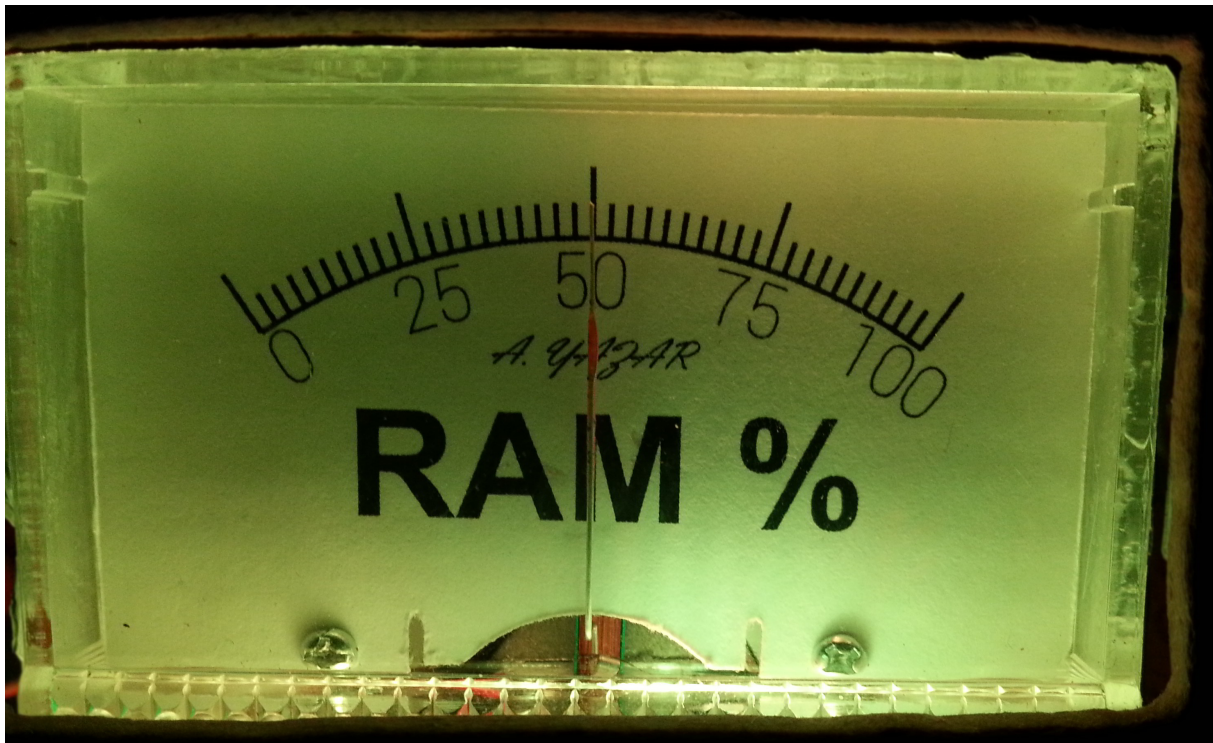
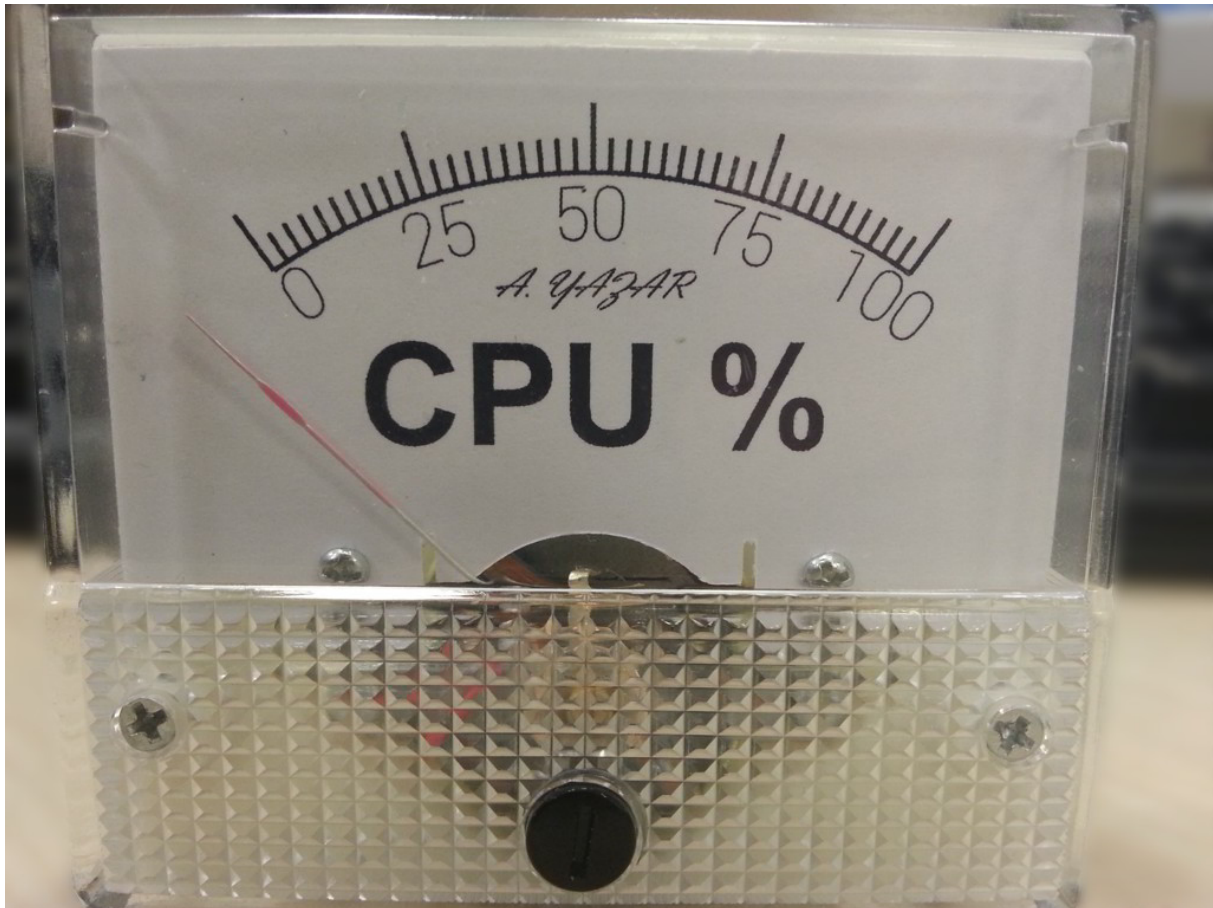


**aPCmeter Project**  
**[www.alperiyazar.com/r/aPCmeter](http://www.alperiyazar.com/r/aPCmeter)**

Figure 1 My Ugly aPCmeter







## 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

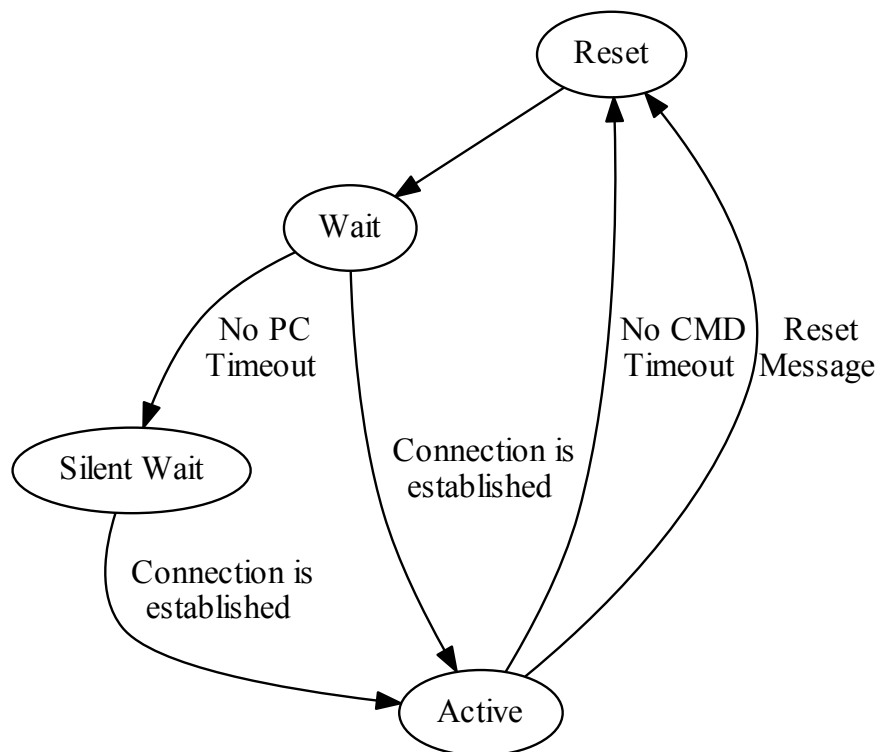
low, its gauge is illuminated by only green LED. As CPU usage increases, it is illuminated by both red and green LED resulting a yellow light. In addition to CPU and RAM usage information, PC software calculates lighting. Gauge position and lighting information are independent for Arduino software. PC 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.alperiyazar.com/r/aPCmeter> Without 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	<p>This is the state after power up. Also setting the program counter (PC) to zero will reset the CPU.</p> <p>See also</p> <p><code>resetFunc()</code></p>

State	Explanation
Wait	<p>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, <i>no PC timeout</i> is occurred and the state is changed from Wait to Silent Wait.</p> <p>See also</p> <p><a href="#">_NO_PC_TIMEOUT</a>  <a href="#">_UART_TIMEOUT_ms</a>  <a href="#">wait_for_connection()</a>.</p>
Silent Wait	<p>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 powered 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.</p> <p>See also</p> <p><a href="#">wait_for_connection()</a></p>
Active	<p>This is the state where aPCmeter does its job. After connection is established, aPCmeter works in active state until power is removed. The serial communication 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.</p> <p>See also</p> <p><a href="#">loop()</a>  <a href="#">_NO_CMD_TIMEOUT</a>  <a href="#">_UART_TIMEOUT_ms</a></p>

### 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 Message (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 separate command for blue LEDs just to turn them on or off separately.

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/alperiyazar/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:

#### [aPCmeter.ino](#)

APCmeter Project, source code of Arduino firmware

10

## 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 indicate 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...

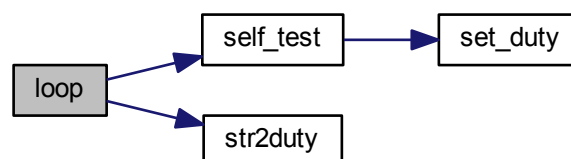
#### See also

<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

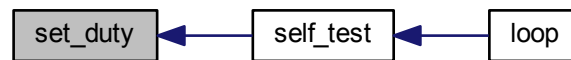
in	<i>channel</i>	0 means CPU gauge, 1 means RAM gauge
in	<i>duty</i>	It should be between 0 and 255 (including both). Otherwise, it will be truncated into this range. It is the PWM value for <code>analogWrite()</code> function. 0 corresponds to 0% for gauge, 100% for green and 0% for red.

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](#), [gaugePinRAM](#), [greenPinCPU](#), [greenPinRAM](#), [redPinCPU](#), [redPinRAM](#), and [wait\\_for\\_connection\(\)](#).

Here is the call graph for this function:



#### 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

<i>Integer</i>	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 ( )

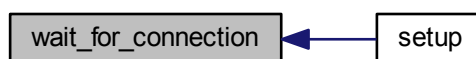
Wait proper connection initialization from PC.

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:



### 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

```

00001
00022 #define _NO_CMD_TIMEOUT 29
00023
00032 #define _NO_PC_TIMEOUT 29
00033
00040 #define _UART_TIMEOUT_ms 2000
00041
00046 #define _UART_BAUD_RATE 9600
00047
00052 const int redPinCPU = 3;
00053
00058 const int greenPinCPU = 5;
00059
00064 const int bluePinCPU = 2;
00065
00070 const int gaugePinCPU = 6;
00071
00076 const int redPinRAM = 9;
00077
00082 const int greenPinRAM = 10;
00083
00088 const int bluePinRAM = 19;
00089
00094 const int gaugePinRAM = 11;
00095
00100 char RX_buf[64];
00101
00105 void(* resetFunc) (void) = 0;
00106
00111 void setup()
00112 {
00113     pinMode(gaugePinCPU, INPUT);
00114     pinMode(gaugePinRAM, INPUT);
00115
00116     digitalWrite(redPinCPU, HIGH);
00117     digitalWrite(greenPinCPU, HIGH);
00118     digitalWrite(bluePinCPU, HIGH);
00119
00120     digitalWrite(redPinRAM, HIGH);
00121     digitalWrite(greenPinRAM, HIGH);
00122     digitalWrite(bluePinRAM, HIGH);
00123
00124     pinMode(redPinCPU, OUTPUT);
00125     pinMode(greenPinCPU, OUTPUT);
00126     pinMode(bluePinCPU, OUTPUT);
00127
00128     pinMode(redPinRAM, OUTPUT);
00129     pinMode(greenPinRAM, OUTPUT);
00130     pinMode(bluePinRAM, OUTPUT);
00131
00132     //All LEDs are off. Gauges are off.
00133
00134     // Set properties of Serial's.
00135     Serial.setTimeout(_UART_TIMEOUT_ms);
00136     Serial.begin(_UART_BAUD_RATE);
00137     while (!Serial) {
00138         ; // wait for serial port to connect. Needed for native USB
00139     }
00140     wait_for_connection();
00141 }
00142
00146 void wait_for_connection()
00147 {
00148     int timeout = 0;
00149
00150     // Try to randomize pseudo-random sequences used by random() funtion.
00151     randomSeed(analogRead(0));
00152     while (1)
00153     {
00154         Serial.println("aPCmeter");
00155         if ( Serial.readBytesUntil('E', RX_buf, 64) == 0)
00156         {
00157             analogWrite(redPinCPU, random(256));
00158             analogWrite(redPinRAM, random(256));
00159             analogWrite(greenPinRAM, random(256));
00160             analogWrite(greenPinCPU, random(256));
00161             digitalWrite(bluePinCPU, random(2));
00162             digitalWrite(bluePinRAM, random(2));
00163             timeout++;
00164             if (timeout > _NO_PC_TIMEOUT)
00165             {
00166                 timeout--;
00167                 digitalWrite(greenPinCPU, HIGH);
00168                 digitalWrite(bluePinCPU, HIGH);

```

```

00169     digitalWrite(greenPinRAM, HIGH);
00170     digitalWrite(bluePinRAM, HIGH);
00171     analogWrite(redPinRAM, 128 + random(128));
00172     analogWrite(redPinCPU, 128 + random(128));
00173 }
00174 }
00175 else
00176 {
00177     if ((RX_buf[0] == 'S') && (RX_buf[1] == 's'))
00178     {
00179         timeout = 0;
00180         digitalWrite(redPinCPU, HIGH);
00181         digitalWrite(redPinRAM, HIGH);
00182         digitalWrite(bluePinCPU, HIGH);
00183         digitalWrite(bluePinRAM, HIGH);
00184         digitalWrite(greenPinCPU, LOW);
00185         digitalWrite(greenPinRAM, LOW);
00186         // Both gauges are fully green.
00187         Serial.println("OK");
00188         return;
00189     }
00190     memset(RX_buf, ' ', 64); //Clear all buffer
00191 }
00192 }
00193 }
00194 }
00199 void loop()
00200 {
00201     int timeout_ctr = 0;
00202     while (1)
00203     {
00204         if (Serial.readBytesUntil('E', RX_buf, 64) == 0)
00205         {
00206             timeout_ctr++;
00207             if (timeout_ctr > _NO_CMD_TIMEOUT)
00208             {
00209                 resetFunc();
00210             }
00211         }
00212         else
00213         {
00214             switch (RX_buf[0])
00215             {
00216                 case 'P':
00217                     if (RX_buf[19] == 'p')
00218                     {
00219                         analogWrite(redPinCPU, str2duty(&(RX_buf[1])));
00220                         analogWrite(greenPinCPU, str2duty(&(RX_buf[4])));
00221                         analogWrite(gaugePinCPU, str2duty(&(RX_buf[7])));
00222                         analogWrite(redPinRAM, str2duty(&(RX_buf[10])));
00223                         analogWrite(greenPinRAM, str2duty(&(RX_buf[13])));
00224                         analogWrite(gaugePinRAM, str2duty(&(RX_buf[16])));
00225                         Serial.println("OK");
00226                         timeout_ctr = 0;
00227                     }
00228                     break;
00229                 case 'T':
00230                     if (RX_buf[1] == 't')
00231                     {
00232                         self_test();
00233                         Serial.println("OK");
00234                         timeout_ctr = 0;
00235                     }
00236                 case 'B':
00237                     if (RX_buf[3] == 'b')
00238                     {
00239                         if (RX_buf[1] == '1')
00240                         {
00241                             digitalWrite(bluePinCPU, LOW);
00242                         }
00243                         if (RX_buf[1] == '0')
00244                         {
00245                             digitalWrite(bluePinCPU, HIGH);
00246                         }
00247                         if (RX_buf[2] == '1')
00248                         {
00249                             digitalWrite(bluePinRAM, LOW);
00250                         }
00251                         if (RX_buf[2] == '0')
00252                         {
00253                             digitalWrite(bluePinRAM, HIGH);
00254                         }
00255                         Serial.println("OK");
00256                         timeout_ctr = 0;
00257                     }
00258                 case 'R':
00259                     if (RX_buf[1] == 'r')

```

```

00260         {
00261             Serial.println("OK");
00262             timeout_ctr = 0;
00263             Serial.flush();
00264             resetFunc();
00265         }
00266         case 'K':
00267             if (RX_buf[1] == 'k')
00268             {
00269                 Serial.println("OK");
00270                 timeout_ctr = 0;
00271             }
00272         default:
00273             timeout_ctr++;
00274             break;
00275     }
00276     memset(RX_buf, ' ', 64);
00277 }
00278 }
00279 }
00280 }
00281
00289 int str2duty(const char *str)
00290 {
00291     int temp = 0;
00292
00293     temp = 100 * (str[0] - 0x30);
00294     temp += 10 * (str[1] - 0x30);
00295     temp += 1 * (str[2] - 0x30);
00296
00297     return (constrain(temp, 0, 255));
00298 }
00299 }
00300
00307 void set_duty(int channel, int duty)
00308 {
00309     // channel 0-> CPU, 1-> RAM
00310
00311     duty = constrain(duty, 0, 255);
00312     if (channel == 0)
00313     {
00314         analogWrite(gaugePinCPU, duty);
00315         analogWrite(greenPinCPU, duty);
00316         analogWrite(redPinCPU, 255 - duty);
00317     }
00318     else if (channel == 1)
00319     {
00320         analogWrite(gaugePinRAM, duty);
00321         analogWrite(greenPinRAM, duty);
00322         analogWrite(redPinRAM, 255 - duty);
00323     }
00324     else
00325     {
00326     }
00327 }
00328 }
00329
00334 void self_test()
00335 {
00336     int duty = 0;
00337     pinMode(gaugePinCPU, INPUT);
00338     pinMode(gaugePinRAM, INPUT);
00339     delay(3000); //Gauges will relax in 3 seconds.
00340     //Just sweep
00341     for (duty = 0; duty < 255; duty++)
00342     {
00343         analogWrite(gaugePinCPU, duty);
00344         analogWrite(gaugePinRAM, duty);
00345         if ( (0 <= duty) && (duty < 85))
00346         {
00347             digitalWrite(redPinCPU, HIGH);
00348             digitalWrite(greenPinCPU, HIGH);
00349             digitalWrite(bluePinCPU, LOW);
00350
00351             digitalWrite(redPinRAM, HIGH);
00352             digitalWrite(greenPinRAM, HIGH);
00353             digitalWrite(bluePinRAM, LOW);
00354         }
00355         if ( (85 <= duty) && (duty < 170))
00356         {
00357             digitalWrite(redPinCPU, HIGH);
00358             digitalWrite(bluePinCPU, HIGH);
00359             digitalWrite(greenPinCPU, LOW);
00360
00361             digitalWrite(redPinRAM, HIGH);
00362             digitalWrite(bluePinRAM, HIGH);
00363             digitalWrite(greenPinRAM, LOW);

```

```

00364
00365     }
00366     if ( (170 <= duty) && (duty < 255))
00367     {
00368         digitalWrite(greenPinCPU, HIGH);
00369         digitalWrite(bluePinCPU, HIGH);
00370         digitalWrite(redPinCPU, LOW);
00371
00372         digitalWrite(bluePinRAM, HIGH);
00373         digitalWrite(greenPinRAM, HIGH);
00374         digitalWrite(redPinRAM, LOW);
00375
00376     }
00377     delay(30);
00378 }
00379 for (duty = 255; duty > 0; duty--)
00380 {
00381     set_duty(0, duty);
00382     set_duty(1, duty);
00383     delay(10);
00384 }
00385 pinMode(gaugePinCPU, INPUT);
00386 pinMode(gaugePinRAM, INPUT);
00387 delay(1000);
00388 }
00389

```

## 6.3 main.md File Reference

## 6.4 main.md

```

00001 # What is aPCmeter? # {#mainpage}
00002
00003 aPCmeter is Arduino Nano 3.0 based device which shows CPU and RAM usage of PC in a vintage looking
00004 way. Here there are some features of the board. Although I included some photos of the hardware, this document
00005 is dedicated to the Arduino firmware.
00006
00007 @note For the most up-to-date information, please check the project webpage
00008 http://www.alperiyazar.com/r/aPCmeter
00009
00010 @image latex aPCmeter_Promo.png
00011
00012 Since my wood working ability is very limited, it has a crude looking wood case.
00013
00014 @image latex aPCmeter_white_annotated_1024_721.jpg My Ugly aPCmeter
00015 @image latex aPCmeter_1.jpg
00016 @image latex aPCmeter_2.jpg
00017 @image latex aPCmeter_6.jpg
00018
00019 # The Overall System Structure # {#the_overall_system_structure}
00020
00021 The overall structure is very simple. A software running on PC computes CPU and RAM usage percentages.
00022 Notice that both CPU and RAM gauges have lighting. Each gauges has its own RGB LED. For example if CPU
00023 usage is low, its gauge is illuminated by only green LED. As CPU usage increases, it is illuminated by both red
00024 and green LED resulting a yellow light. In addition to CPU and RAM usage information, PC software calculates
00025 lighting. Gauge position and lighting information are independent for Arduino software. PC software sends
00026 data over USB based virtual COM port and Arduino displays them. This is how Arduino code works basically.
00027
00028 @note For available PC software, please check the project webpage http://www.alperiyazar.com/r/aPCmeter
00029 Wtihout a proper PC software, aPCmeter hardware can't work properly.

```

## 6.5 state\_diagram.md File Reference

## 6.6 state\_diagram.md

```

00001 # State Diagram # {#states}
00002
00003 State diagram is shown in the following figure.
00004
00005 @dotfile state_diagram.gv
00006
00007 Explanations of the states are given as below.
00008
00009 State | Explanation

```

```

00010 ----- | -----
00011 Reset | This is the state after power up. Also setting the program counter (PC) to zero will reset
the CPU. @see resetFunc()
00012 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 occurred and the
state is changed from Wait to Silent Wait. @see #_NO_PC_TIMEOUT @see #_UART_TIMEOUT_ms @see
wait_for_connection().
00013 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 powered up but no PC software gives a response, changing colors of gauges
continuously may disturb people around aPCmeter. Therefore aPCmeter enters in Silent 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()
00014 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 communication 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

```

00001 # UART Protocol # {#uart}
00002
00003 UART protocol works on 9600/8-N-1.
00004
00005 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.
00006
00007 Here is the list of all possible messages:
00008
00009 ## Hello Message (Dir: aPCmeter -> PC)
00010 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.
00011
00012 ## Start Message (Dir: PC -> aPCmeter)
00013 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
00014
00015 Byte 1 | Byte 2 | Byte 3
00016 ----- | ----- | -----
00017 'S' | 's' | 'E' (Optional)
00018
00019 Response from the aPCmeter:
00020
00021 Byte 1 | Byte 2 | Byte 3 | Byte 4
00022 ----- | ----- | ----- | -----
00023 'O' | 'K' | '\\r' | '\\n'
00024
00025 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.
00026
00027 ## PWM Message (Dir: PC -> aPCmeter)
00028 This is the main message to use aPCmeter. It is used to change PWM values of gauges and LEDs. Here is
the structure
00029
00030 Byte Number | Message |
00031 ----- | -----
00032 1 | 'P'
00033 2-4 | PWM value of red LED of CPU gauge
00034 5-7 | PWM value of green LED of CPU gauge
00035 8-10 | PWM value of CPU gauge
00036 11-13 | PWM value of red LED of RAM gauge
00037 14-16 | PWM value of green LED of RAM gauge
00038 17-19 | PWM value of RAM gauge

```

```

00039 20 | 'p'
00040 21 | 'E' (Optional)
00041
00042 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.
00043
00044 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.
00045
00046 Response from the aPCmeter:
00047
00048 Byte 1 | Byte 2 | Byte 3 | Byte 4
00049 ----- | ----- | ----- | -----
00050 'O'      | 'K'      | '\\r'    | '\\n'
00051
00052 ## Test Message(Dir: PC -> aPCmeter)
00053 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.
00054
00055 Byte 1 | Byte 2 | Byte 3
00056 ----- | ----- | -----
00057 'T'      | 't'      | 'E' (Optional)
00058
00059 Response from the aPCmeter after self-test:
00060
00061 Byte 1 | Byte 2 | Byte 3 | Byte 4
00062 ----- | ----- | ----- | -----
00063 'O'      | 'K'      | '\\r'    | '\\n'
00064
00065 ## Blue Message(Dir: PC -> aPCmeter)
00066
00067 Blue LEDs are not PWM driven and they are not controlled by the PWM command. There are separate
      command for blue LEDs just to turn them on or off separately.
00068
00069 Byte 1 | Byte 2 | Byte 3 | Byte 4 | Byte 5
00070 ----- | ----- | ----- | ----- | -----
00071 'B'      | 'B'      | CPU Blue LED | RAM Blue LED | 'E' (Optional)
00072
00073 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.
00074
00075 Response from the aPCmeter:
00076
00077 Byte 1 | Byte 2 | Byte 3 | Byte 4
00078 ----- | ----- | ----- | -----
00079 'O'      | 'K'      | '\\r'    | '\\n'
00080
00081 ## Reset Message(Dir: PC -> aPCmeter)
00082 It is used to reset the aPCmeter. State is changed from Active to Reset after this command.
00083
00084 Byte 1 | Byte 2 | Byte 3
00085 ----- | ----- | -----
00086 'R'      | 'r'      | 'E' (Optional)
00087
00088 Response from the aPCmeter:
00089
00090 Byte 1 | Byte 2 | Byte 3 | Byte 4
00091 ----- | ----- | ----- | -----
00092 'O'      | 'K'      | '\\r'    | '\\n'
00093
00094 ## Keep Alive Message(Dir: PC -> aPCmeter)
00095 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.
00096
00097 Byte 1 | Byte 2 | Byte 3
00098 ----- | ----- | -----
00099 'K'      | 'k'      | 'E' (Optional)
00100
00101 Response from the aPCmeter:
00102
00103 Byte 1 | Byte 2 | Byte 3 | Byte 4
00104 ----- | ----- | ----- | -----
00105 'O'      | 'K'      | '\\r'    | '\\n'

```





## Index

- [\\_NO\\_CMD\\_TIMEOUT](#)
    - [aPCmeter.ino, 12](#)
  - [\\_NO\\_PC\\_TIMEOUT](#)
    - [aPCmeter.ino, 12](#)
  - [\\_UART\\_BAUD\\_RATE](#)
    - [aPCmeter.ino, 13](#)
  - [\\_UART\\_TIMEOUT\\_ms](#)
    - [aPCmeter.ino, 13](#)
- [aPCmeter.ino, 10, 20](#)
  - [\\_NO\\_CMD\\_TIMEOUT, 12](#)
  - [\\_NO\\_PC\\_TIMEOUT, 12](#)
  - [\\_UART\\_BAUD\\_RATE, 13](#)
  - [\\_UART\\_TIMEOUT\\_ms, 13](#)
  - [bluePinCPU, 17](#)
  - [bluePinRAM, 17](#)
  - [gaugePinCPU, 18](#)
  - [gaugePinRAM, 18](#)
  - [greenPinCPU, 18](#)
  - [greenPinRAM, 18](#)
  - [loop, 14](#)
  - [RX\\_buf, 19](#)
  - [redPinCPU, 19](#)
  - [redPinRAM, 19](#)
  - [resetFunc, 19](#)
  - [self\\_test, 14](#)
  - [set\\_duty, 15](#)
  - [setup, 16](#)
  - [str2duty, 16](#)
  - [wait\\_for\\_connection, 17](#)
- [bluePinCPU](#)
  - [aPCmeter.ino, 17](#)
- [bluePinRAM](#)
  - [aPCmeter.ino, 17](#)
- [gaugePinCPU](#)
  - [aPCmeter.ino, 18](#)
- [gaugePinRAM](#)
  - [aPCmeter.ino, 18](#)
- [greenPinCPU](#)
  - [aPCmeter.ino, 18](#)
- [greenPinRAM](#)
  - [aPCmeter.ino, 18](#)
- [loop](#)
  - [aPCmeter.ino, 14](#)
- [main.md, 23](#)
- [RX\\_buf](#)
  - [aPCmeter.ino, 19](#)
- [redPinCPU](#)
  - [aPCmeter.ino, 19](#)
- [redPinRAM](#)
  - [aPCmeter.ino, 19](#)
- [resetFunc](#)
  - [aPCmeter.ino, 19](#)
- [self\\_test](#)
  - [aPCmeter.ino, 14](#)
- [set\\_duty](#)
  - [aPCmeter.ino, 15](#)
- [setup](#)
  - [aPCmeter.ino, 16](#)
- [state\\_diagram.md, 23](#)
- [str2duty](#)
  - [aPCmeter.ino, 16](#)
- [uart\\_protocol.md, 24](#)
- [wait\\_for\\_connection](#)
  - [aPCmeter.ino, 17](#)