

Arduino goes TPS

Bei meinen Leuchtturmmodell (was ich mittlerweile veräußert habe) brauchte ich eine Schaltung, die das Licht alle 5 Sekunden 3x blitzen läßt. Da das analog nicht gerade einfach ist, habe ich aus meiner Wühlkiste einen ATTiny24 rausgekramt (hat ich gerade noch da), mein AVR Studio angeschmissen und mal eben schnell die Steuerung geschrieben. Alles kein Problem.

Bei diversen Ausstellungen wurde ich dann immer mal wieder gefragt, wie man das nachbaut. Klar kann man, wenn man... Und natürlich sollten dabei andere Blitzfolgen einstellbar sein. Möglichst auch noch per Fernsteuerung wählbar inkl. Nebenlicht... Dabei viel mir auf, daß unter den Modellbauern die MCU Affinen doch recht rar gesäht sind. Schade eigentlich.

Dann bin ich auf die TPS von Burghart Kainka (<http://www.elektronik-labor.de/Lernpakete/TPS/TPS0.html>) aufmerksam geworden. Diese ist für viele Dinge im Modellbaubereich genau richtig. Einfach zu programmieren, auch „im Feld“, reduziert auf das nötigste. Das einzige, was der TPS fehlt, ist die Anbindung an eine Fernsteuerung, also das Auswerten von Fernsteuersignalen. Und auch die Möglichkeit einen (oder auch 2) Servo zu steuern, kann die originale TPS nicht. Noch nicht, dachte ich mir.

Da ich mich sehr gut mit den Atmel Prozessoren auskenne und auch schon viel Erfahrung mit dem Arduino-System hatte, war es nur ein kleiner Schritt die TPS auf den Arduino zu portieren. Noch dazu ist der Arduino sehr weit verbreitet.

Zunächst dacht ich, ich könnte die ATmega8 BASCOM Variante einfach anpassen, aber leider war das nicht so einfach möglich. Also hab ich die gesamte Steuerung neu programmiert und dabei auch gleich ein paar Befehls Erweiterungen für den Modellbau eingebaut. Hier stell ich nun die Version kostenfrei und zur allgemeinen Verwendung zur Verfügung. Wenn ihr einen Fehler findet, dann schreibt mir einfach. Ich werde versuchen, die gefundenen Bugs so schnell wie möglich zu entfernen.

Die Arduino Implementierung kann auf 2 verschiedenen MCU's laufen. Einerseits auf dem Arduino selber, dann sogar mit der Möglichkeit aus dem Emulator direkt programmiert zu werden. Die 2. Variante basiert auf einem ATTiny84. Diese läßt sich leider nur mit einem „normalen“ ISP Programmer programmieren. Und auch die TPS Befehle lassen sich dann wie bei der HOLTEK Version nur per LED und Taster einprogrammieren. (Diese Möglichkeit besteht bei der Arduino natürlich zus. auch noch, wenn man mal am Wasser was umproggen muss.)

Features

Neben dem was die TPS bietet, hat diese Version noch zus. folgende Features

- 2 RC Kanäle auslesbar. Hier kann an die beiden RC Kanäle jeweils 1 Kanal eines RC Empfängers angeschlossen werden.
- 2 Servokanäle. Es gibt Anschlussmöglichkeit für 2 Servos. (Leider fallen dadurch die beiden PWM Kanäle automatisch weg.
- Erweiterungen im Befehlssatz, einmal gibt es 1 Byte Befehle. Dadurch wird das Auslesen und das Setzen von ADC, RC, PWM, Servo deutlich feinfühlicher.
- Dazu passend gibt es eine neue Berechnung. $A = B * 16 + A$ um 1 Byte zu berechnen.
- Weiterhin gibt es 2 zus. Register zum Zwischenspeichern von Werten, wie auch einen 16

Einheiten großen Stack mit den 2 Zugriffsmethoden Push und Pop.

- Beim Skip gibt's noch die A=0 Bedingung
- Und last but not least kann man 6 Unterroutinen definieren und anspringen. Diese dürfen dann auch ausserhalb des 256 Byte großen Adressraumes liegen. Beim Arduino ist das EEPROM 1KB (ATMega328), beim ATtiny84 immerhin 512 Bytes groß.

Die PWM Signale werden mit 500Hz erzeugt, die Servosignale sind PPM kodiert. (50Hz Frequenz kleinster Impuls 1ms, größter Impuls 2ms, Mitte bei 1,5ms.) In der Arduino Variante kann man zusätzlich auch Töne erzeugen. Diese werden auf dem PWM2 Ausgang ausgegeben. Zur Ausgabe muss im A Register (ähnlich wie bei den anderen Byte Befehlen) die Tonehöhe angegeben werden. Dabei kommt die bei Midi übliche Kodierung der Töne zu tragen. Erlaubt sind Miditöne zwischen 36 (low C2) und 109 (C8). Ein Wert von 0 schaltet den Ton wieder ab.

Leider muss man sich in der Tiny Version entscheiden, ob man Servos oder den Tone Befehl nutzen möchte. Beides gleichzeitig geht Aufgrund der Doppelnutzung des Timers nicht. In den Quellen kann man über entsprechende #defines bestimmen, was man in seiner TPS Variante haben möchte.

Die Programmierung des Arduino kann über die Tasten, wie von der TPS Variante her bekannt erfolgen. Es gibt aber auch die Möglichkeit, Programme mit meinem TPS/SPS Emulator zu schreiben und direkt in den Arduino hoch zu laden. Theoretisch geht das auch mit der ATtiny Variante, dazu muss aber ein ISP Programmer vorhanden sein. Beim Arduino ist der ja quasi mit an Board.

Ganz neu ist in beiden Varianten ab der Version 0.10 eine neue Möglichkeit Programme mit einem einfachen seriellen Protokoll in die TPS zu laden. Das funktioniert sowohl mit der Arduino wie mit der Tiny Variante. Dazu wird das IntelHEX Format verwendet. Näheres dazu im Bereich [TPS Assembler](#)

Optionen im Quelltext

Alle Features können in den Programmquellen ein/ausgeschaltet werden. Dazu werden sog. Compilerschalter verwendet. Da alle möglichen Version über einen Satz an Arduino Quelldateien erzeugt werden können, wird zunächst mit Hilfe von bedingter Compilierung überprüft, für welche Zielplattform der Code erzeugt werden soll. Dafür sind mehrere #ifdef Anweisungen in der 1. Quelldatei (TPS.ino) eingebaut.

Folgende Tabelle soll etwas helfen den richtigen Abschnitt für seinen Controller zu finden.

Hardware	Controller	#ifdef Block
Arduino Uno	ATMega328P	__AVR_ATmega328P__
Arduino SPS Board	ATMega328P	__AVR_ATmega328P__
Tiny SPS Board	ATtiny84	__AVR_ATtiny84__
ATtiny84	ATtiny84	__AVR_ATtiny84__
ATtiny861	ATtiny861	__AVR_ATtiny861__
ATtiny4313	ATtiny4313	__AVR_ATtiny4313__

In jedem dieser Blöcken stehen nun die entsprechend der Hardware möglichen Optionen. Jede Option ist mit einem "#define" gekennzeichnet. Aktiviert wird sie, wenn man die Zeile ein kommentiert, d.h. die beiden führenden // entfernt. Will man die Option wieder abschalten, muss man die Zeile entweder entfernen oder mit // am Zeilenanfang kommentieren.

Beispiel: Tonausgabe aktivieren

```
#define TPS_TONE
```

Tonausgabe deaktivieren

```
//#define TPS_TONE
```

z.B. beim Uno wären möglich:

Beschreibung	#define aktiviert
#define TPS_USE_DISPLAY	Will man für die Erleichterung der Programmierung ein TM1637 Display benutzen, muss man diese Option aktivieren
#define TPS_RCRCRECEIVER	Hiermit aktiviert man die Möglichkeit einen Kanal eines RC-Fernsteuerempfänger auszulesen.
#define TPS_ENHANCEMENT	Diese Option aktiviert den erweiterten Befehlssatz
#define TPS_SERIAL_PRG	Ermöglicht Programme über die serielle Schnittstelle zu senden. Z.B. per Handy oder Emulator.
#define TPS_SERVO	Ansteuerung eines oder 2er Servos.
#define TPS_TONE	Ansteuerung eines Lautsprechers zur Tonausgabe.

Nicht alle Optionen stehen für alle Controller zur Verfügung. Der aktuelle implementierungs Stand ist in den jeweiligen Abschnitten zu sehen. Fehlt dort eine Option, kann diese derzeit nicht für den Chip verwendet werden.

Beispieloptionen: - erweiterter Befehlssatz - Servounterstützung - Tone

```
#ifndef __AVR_ATmega328P__
//#define TPS_USE_DISPLAY
//#define TPS_RCRCRECEIVER
#define TPS_ENHANCEMENT
//#define TPS_SERIAL_PRG
#define TPS_SERVO
#define TPS_TONE
#endif
```

Einschränkungen

Ein Mischbetrieb von Servo und PWM (z.B. PWM.1 und Servo.2) ist leider nicht möglich, da sich die beiden gegenseitig beeinflussen. Also entweder Servounterstützung oder PWM. Auch die Ton-Ausgabe ist hiervon betroffen, da der Ton auf dem Servo 2 (PWM 2) Ausgang liegt.

Hardwareentsprechungen

Hier eine Übersicht über die Ein/Ausgabepins der TPS und deren Entsprechung auf dem Arduino. Anschlüsse:

TPS Anschlüsse	Arduino Pins
Din1..4	D0..D3

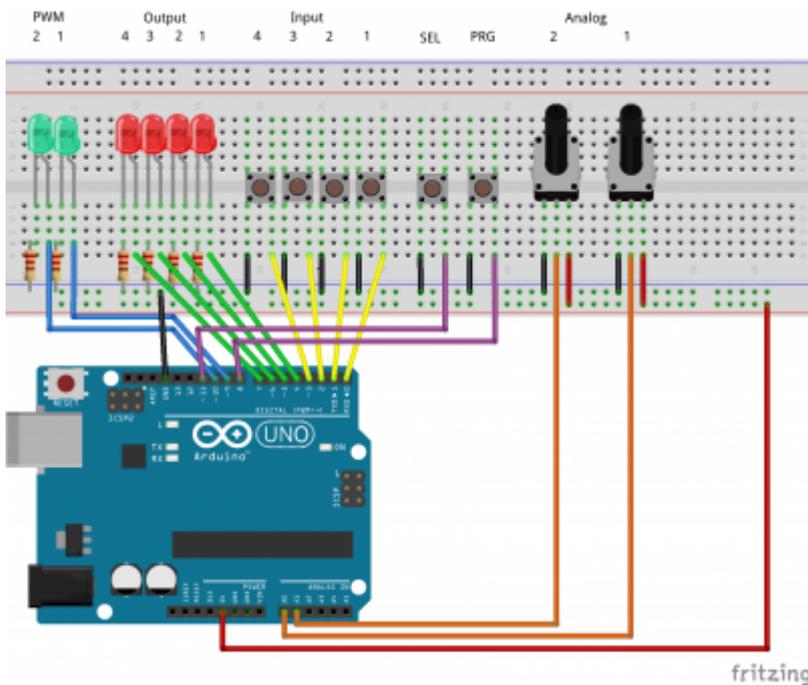
TPS Anschlüsse	Arduino Pins
Dout1..4	D4..D7
PWM1,2	D9,10
ADC1,2	A1,2
SW_PRG (S2)	D8
SW_SEL (S1)	D11

die zusätzlichen Ein/Ausgänge findet ihr hier:

Bezeichnung	Arduino Pins
RC1,2	D18, D19* (A4,A5)
Servo1,2	D9, D10
Tone	D10
TM1637 Data, CLK	D12, D13

*neben den üblichen 13 Pins des Arduinos können auch die analogen Eingänge (A0..5) als digitale Pins verwendung finden. Diesen werden einfach der Reihe nach weiter gezählt. D.h. A0 = D14, A1 = D15 ... A5 = D19

Und ein kleines Bild dazu:



Befehlssatz

Die gelb unterlegten Bereiche sind die Erweiterungen meiner ATTiny_TPS und der Arduino_TPS Version. In eckigen Klammern stehen jeweils die entsprechenden Mnemonics des [SPS Assemblers](#).

	0	1	2	3	4	5	6	7
	n.n.	Port [DOUT]	Delay [WAIT]	Jump back relative [RJMP]	A=# [LDA]	=A	A=	A=Ausdruck
0	NOP [NOP]	aus	1ms	0	0	A<->B [SWAP]		
1		1	2ms	1	1	B=A [MOV]	A=B [MOV]	A=A + 1 [INC]
2		2	5ms	2	2	C=A [MOV]	A=C [MOV]	A=A - 1 [DEC]
3		3	10ms	3	3	D=A [MOV]	A=D [MOV]	A=A + B [ADD]
4		4	20ms	4	4	Dout=A [STA]	Din [LDA]	A=A - B [SUB]
5		5	50ms	5	5	Dout.1=A.1 [STA]	Din.1 [LDA]	A=A * B [MUL]
6		6	100ms	6	6	Dout.2=A.1 [STA]	Din.2 [LDA]	A=A / B [DIV]
7		7	200ms	7	7	Dout.3=A.1 [STA]	Din.3 [LDA]	A=A and B [AND]
8		8	500ms	8	8	Dout.4=A.1 [STA]	Din.4 [LDA]	A=A or B [OR]
9		9	1s	9	9	PWM.1=A [STA]	ADC.1 [LDA]	A=A xor B [XOR]
a		10	2s	10	10	PWM.2=A [STA]	ADC.2 [LDA]	A= not A [NOT]
b		11	5s	11	11	Servo.1=A [STA]	RCin.1 [LDA]	A= A % B (Rest) [MOD]
c		12	10s	12	12	Servo.2=A [STA]	RCin.2 [LDA]	A= A + 16 * B [BYTE]
d		13	20s	13	13	E=A [MOV]	A=E [MOV]	A= B - A [BSUBA]
e		14	30s	14	14	F=A [MOV]	A=F [MOV]	A=A SHR 1 [SHR] ab V 11
f		15	60s	15	15	Push A [PUSH]	Pop A [POP]	A=A SHL 1 [SHL] ab V 11

Zus. Features in der Arduino_TPS Version:

- Es gibt 2 zus. Register (E und F)
- und es gibt einen Kellerstapel mit den 2 Methoden push (auflegen) und pop (runter nehmen). In dem Stapel haben 16 Werte Platz.
- Weiterhin gibt es 2 neue Berechnungen, einmal den Rest einer Division ($A = A \% B$) und einmal eine 8-Bit umwandlung. $A = A + 16 * B$
- Ab Version 0.6 ist auch noch der Swap Befehl hinzugekommen, der A und B Register vertauscht.
- Und eine neue Berechnung $A = B - A$. Gerade wenn man sich im 8-Bit Raum aufhält ist es manchmal recht umständlich, diese Operation auszuführen.
- ab Version 11 gibt es nun auch Shift Operationen. Diese verschieben den Inhalt des Registers A entweder nach links SHL (was eine Multiplikation mit 2 entspricht) oder nach rechts (SHR). Das wäre dann eine Division durch 2. **Achtung: zwar befinden wir uns hier im 4Bit Raum, diese Operationen sind aber 8 Bit fähig.**

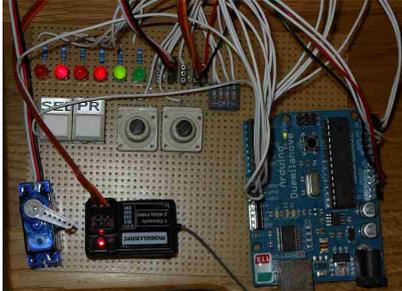
	8	9	a	b	c	d	e	f
	Page [PAGE]	Jump absolut (#+16*page) [JMP]	C* C>0: C=C-1; Jump # + (16*page) [LOOPC]	D* D>0:D=D-1; Jump # + (16*page) [LOOPC]	Skip if	Call # + (16*Page) [Call]	Callsub/Ret	Byte Befehle
0	0	0	0	0	A==0 [SKIPO]	0	ret [RTR]	A=ADC.1 [BLDA]
1	1	1	1	1	A>B [AGTB]	1	Call 1 [CASB]	A=ADC.2 [BLDA]
2	2	2	2	2	A<B [ALTB]	2	2 [CASB]	A=RCin.1 [BLDA]
3	3	3	3	3	A==B [AEQB]	3	3 [CASB]	A=RCin.2 [BLDA]
4	4	4	4	4	Din.1==1 [DEQ1 1]	4	4 [CASB]	PWM.1=A [BSTA]
5	5	5	5	5	Din.2==1 [DEQ1 2]	5	5 [CASB]	PWM.2=A [BSTA]
6	6	6	6	6	Din.3==1 [DEQ1 3]	6	6 [CASB]	Servo.1=A [BSTA]
7	7	7	7	7	Din.4==1 [DEQ1 4]	7		Servo.2=A [BSTA]
8	8	8	8	8	Din.1==0 [DEQ0 1]	8	Def 1 [DFSB]	Tone=A [TONE]
9	9	9	9	9	Din.2==0 [DEQ0 2]	9	2 [DFSB]	
a	10	10	10	10	Din.3==0 [DEQ0 3]	10	3 [DFSB]	
b	11	11	11	11	Din.4==0 [DEQ0 4]	11	4 [DFSB]	
c	12	12	12	12	S_PRG==0 [PRG0]	12	5 [DFSB]	
d	13	13	13	13	S_SEL==0 [SEL0]	13	6 [DFSB]	int. LED on
e	14	14	14	14	S_PRG==1 [PRG1]	14		int. LED off
f	15	15	15	15	S_SEL==1 [SEL1]	15	restart [REST]	PrgEnd [PEND]

Zus. Features in der Arduino_TPS Version:

- Weil wir genug EEPROM haben ist der Page Bereich auf 16 Seiten ausgedehnt. Macht also ins. 256 Bytes
- Beim Skip gibt's noch den A=0 Befehl.
- Über die E Befehle können nun auch 6 echte Unterprogramme angelegt werden. Diese werden über den Befehl Def# angelegt. Mit Call# wird die Routine angesprungen. Mit Return kommt man zurück. Die Def# dürfen auch oberhalb der 256 Bytes im EEPROM liegen. Also auch ausserhalb der Reichweite der Sprungbefehle.
- Neu ist auch der Restartbefehl, mit dem der ganze Controller neu gestartet wird.
- Im Bereich F sind Befehle hinterlegt, die mit 8 Bit Auflösung arbeiten.
- FF bedeutet Programmende. → automatischer Sprung nach 0.

- **NEU** Tone Befehl: gibt einen Ton aus. Basis sind die Midinoten. Es werden die Midi Noten von 36 (C2) bis 108 (C8) unterstützt. Ausgang ist der PWM 2. Basis ist das A-Register als 8 Bit Wert.

Und nun viel Spass mit dem Arduino.
Achja, wie immer alles ohne Gewähr...
Und nun noch ein kleiner Testaufbau:



InField Programmierung

Eines der Kernfeatures der TPS ist die Möglichkeit, die TPS nur mit den Tastern und den LEDs zu programmieren. D.h. bedeutet, daß man zunächst keinerlei Programmierwerkzeuge benötigt. Allerdings ist eine ausgedruckte oder zumindest auf dem Handy angezeigte Befehlstabelle sehr hilfreich.

Natürlich muss man auch beachten, daß man u.U. angehängte Hardware vorher von der TPS abkoppelt. Sonst kann es zu unschönen Effekten kommen.

Mir war der originale Programmiermodus nicht ganz einleuchtend. Deswegen habe ich die händische Programmierschnittstelle überarbeitet. So erschien Sie mir etwas logischer. Der Ablauf ist dabei immer der gleiche und gliedert sich in 3 Bereiche.

1. Anzeige der aktuellen Adresse.
2. Programmierung des Kommandos. (das ist das obere Nibble (4-Bit) des Befehls, in der o.g. Tabelle also die Spalte)
3. Programmierung der Daten zu dem Kommando. (das ist das untere Nibble des Befehls, in der o.g. Tabelle also die Zeile)

Die Programmierung startet immer bei Adresse 0, programmiert wird mit der PRG Taste, SEL wählt dann den Wert aus.

Wichtig, genau wie bei dem originalen Programmiermodus, ist hier, das man das komplette Programm mit der PRG Taste „durchsteppen“ kann.

Der Programmiermodus wird mit einem Reset bei gedrückter PRG Taste gestartet. Damit man weiß, daß man sich im Programiermodus befindet, blinken nun einmal alle 4 LEDs.

Danach startet die o.g. Proggmierschleife.

- **1a**: zunächst blinkt einmal kurz die 1. LED. Danach wird das obere Nibble der Adresse angezeigt. (0.5 sec)
- **1b**: jetzt blinkt einmal die 2. LED. Danach wird das untere Nibble der Adresse angezeigt. (0.5 sec)
- **2**: danach blinkt die LED 3 und der Wert des Kommandos (oberen Nibbles des Befehls) wird angezeigt. Mit SEL kann man nun diesen Wert verändern, PRG bestätigt dann die Einstellung.
- **3**: jetzt blinkt die LED 4 und der Wert der Daten (unteres Nibbles des Befehls) wird angezeigt. Mit SEL kann man nun diesen Wert verändern, PRG bestätigt dann die Einstellung. Wurde der Befehl geändert, blinken nun alle LEDs einmal und der neue Befehl wird gespeichert. Dann wird

die Adresse um 1 hochgezählt und die Schleife startet bei 1a. Ist man mit der Programmierung fertig oder möchte man abbrechen, genügt ein Reset.

Hier mal das ganze tabellarisch aufgestellt: Button programming the Arduino_TPS/Tiny_TPS

Zum Starter der PProgrammieruung, RESET und PRG gleichzeitig drücken, dann RESET los lassen. PRG gedrückt halten bis alle LEDs aufleuchten.

LEDs	Kommentar	nächster Schritt
1111	warten bis PRG nicht gedrückt	PRG los lassen
*0001	Indikator untere Nibble der Adresse	nicht weiter zu tun, Verzögerung von 1/4 Sekunde
xxxx	unteres Nibble der Adress wird angezeigt, ist die Adresse 0x00, dann leuchtet keine LED	nicht weiter zu tun, Verzögerung von 1/2 Sekunde
0010	Indikator oberes Nibble der Adresse	nicht weiter zu tun, Verzögerung von 1/4 Sekunde
xxxx	oberes Nibble der Adress wird angezeigt, ist die Adresse 0x00, dann leuchtet keine LED	nicht weiter zu tun, Verzögerung von 1/2 Sekunde
0100	Indikator für Kommando	nicht weiter zu tun, Verzögerung von 1/4 Sekunde
xxxx	Anzeige des aktuellen Kommandos	mit SEL kannst du nun den Wert des Kommandos ändern (Spalte in der Programmier-tabelle), mit PRG geht's weiter zum nächsten Schritt
1000	Indikator für Daten	nicht weiter zu tun, Verzögerung von 1/4 Sekunde
xxxx	Anzeige der aktuellen Daten	mit SEL kannst du nun den Wert der Daten ändern (Zeile in der Programmier-tabelle), mit PRG wird dann gespeichert (einmal blinken alles LEDs) und es geht's weiter mit der nächsten Adresse (weiter bei *)

In der Arduino_TPS ist ein weiterer spezieller Programmiermodus enthalten. Da der verwendete ATmega328 mehr Pins hat, kann man dort ein externes Display anschliessen. Verwendung findet dort ein 4 stelliges 7-Segmentdisplay mit TM1637



Das Display wird, neben den üblichen +5V und GND für die Spannungsversorgung, an die Arduino Pins 12 (Data) und 13 (Clock) angeschlossen. Das sind beim ATmega die Pins PB4 (18) für Data und PB5 (19) für Clock.

Jetzt hat man die Möglichkeit gleichzeitig Adresse (auf den ersten beiden Stellen und den Befehl (auf den Stellen 3 und 4) anzuzeigen.

In den „advanced programming mode“ gelangt man, wenn man PRG und SEL beim Reset gleichzeitig drückt.

Der Ablauf der Programmierung ist der o.G. sehr ähnlich, nur das die Adresse und der Befehlswert auf dem Display ständig angezeigt wird. Editiert werden kann aber immer nur Kommando oder Daten. Ein weiterer Vorteil ist, daß die beiden Pins nicht für TPS Ein/Ausgabepins verwendet werden. Somit muss keine externe Hardware von der eingebauten TPS Platine entfernt werden.

Download

Hier der direkte Link auf die letzte Version: | [last Release als ZIP](#)

Die aktuelle Version liegt auf github: [Arduino_TPS Source](#)

Historie: Die alten Versionen verwalte ich mit meinem Git Versions System. Gogs.
[Arduino SPS Source](#)

From:

<https://wkla.no-ip.biz/ArduinoWiki/> - **Arduino im Modellbau**

Permanent link:

<https://wkla.no-ip.biz/ArduinoWiki/doku.php?id=arduino:arduinosp>

Last update: **2022/05/22 10:02**

