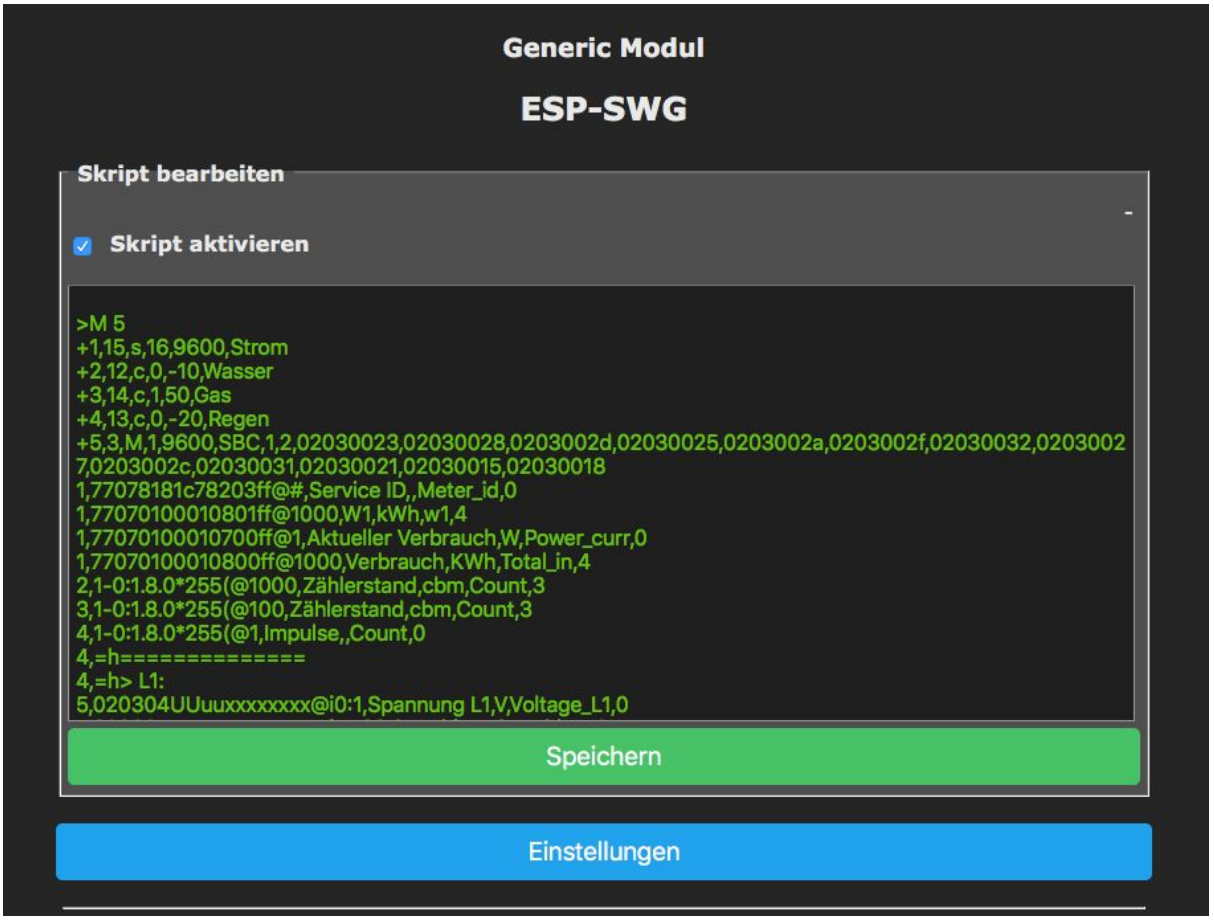
HOW TO: TASMOTA SML



Stand: 17.03.2020
Tasmota: Dev Branch 16.03.2020

Inhaltsverzeichnis

Vorwort / Hinweise

Diese Beschreibung soll dazu dienen Einsteigern die Konfiguration des SML Tasmota Treibers zu erleichtern.
Der Treiber wurde von Gemu2015 entwickelt und ist ein Teil von Tasmota. Er dient zum auslesen verschiedenster Stromzähler via Infrarot Fototransistor / Modbus oder Impulsverfahren.
Des Weiteren unterstützt er das auslesen von Heizungen via eBus.
Der Treiber bringt auch einen Counter mit, der zum auslesen einer Gaszählers oder eines Wasserzählers dienen kann.

Dieses Dokument soll lediglich als Leitfaden dienen. Für eventuelle Fehler in der Beschreibung oder den Schaltbildern übernehme ich keine Haftung und gebe keine Gewähr.
Für euer Unterfangen gibt es ist keine 0 8 15 Komplettlösung! Ihr werdet löten und rechnen müssen und viel Zeit und Geduld benötigen! Plant ausreichend Zeit ein.
Nur durch lernen, informationen sammeln, nachdenken und produktiv sein, kann eine gutes Produkt entstehen. Wer dumme Fragen stellt, bekommt dumme Antworten. Benutzt Google oder die Forum Suchfunktion. Viele anfängliche Fragen wurden bereits unzählige male beantwortet.

Deshalb:
Bevor ihr beim kleinsten Problem direkt einen Beitrag im Forum verfasst, bitte ich euch zuvor für mindestens 20 Minuten nach einer Lösung zu suchen.
Am Ende dieser Beschreibung befindet sich eine Linksammlung mit vielen hilfreichen Links. Das durcharbeiten dieser Links sehe ich als Voraussetzung an.

Des Weiteren appelliere ich hiermit an den gesunden Menschenverstand, die nötigen Sicherheitsvorkehrungen zu treffen. Keine Plomben zu entfernen oder irgendwelche Messgeräte/Zähler zu manipulieren.
Es sollte überflüssig sein dies zu erwähnen, dennoch: Manipulation ist strafbar!

Zu guter Letzt möchte ich mich vor allem Gerhard (Gemu2015) für seine Arbeit, für sein Produkt und den netten Kontakt mit Ihm bedanken. Ebenso dem ganzen Tasmota Team welches Tag für Tag dafür sorgt das ein geniales Produkt noch mehr Funktionien erhält. Den aktiven (forum.creationx.de) Forummitgliedern ohne die viele Informationen die in diesem Dokument enthalten sind, nicht entstanden wären.
Hierbei möchte ich “meierchen006” besonders hervorheben, der vor einiger Zeit bereits ein ähnliches Dokument verfasst hat, welches eine Grundlage zu diesem Dokument ist.

Viel Erfolg!

Hardware

Geeignete Hardware für Tasmota:

Alle auf dem ESP8266 Chip basierenden Entwicklerboards z.B.:

- Wemos D1 mini
- NodeMCU
- ESP01

Sonoff Hardware Relays sind auch kompatibel.

Die verschiedenen Boards unterscheiden sich in Ihrer Anzahl an GPIOs und der größe Ihres Flash Speichers als auch der Art wie sie zu flashen sind. Bitte informiert euch im vorraus.

OBIS/SML via IR-DIODE
Zum Auslesen eines Zählers der über eine IR-LED verfügt wird ein Fototransistor benötigt.

- BPW78A
- TEKT5400S
- Volkszähler Sende/Lesekopf

Einige Zähler müssen zum Senden ihrer Daten erst angeregt werden. Dazu ist ein weiteres Bauteil nötig. Eine IR-LED, die eine bestimmte Zeichenfolge an den Stromzähler sendet. Dieser Antwortet daraufhin mit seinen aktuellen Daten.

- IR333c Diode
- Volkszähler Sende/Lesekopf

Modbus RTU

Um Energiezähler mit Modbus Schnittstelle auslesen zu können, ist ein Modbus2TTL Adapter nötig.

- MAX485 RS485 zu TTL Adapter o. Ä.
- Kompatibler Modbus Zwischenzähler 1P/3P
- Modbusregisterübersicht!

Wasseruhr

Hierzu gibt es verschiedene Optionen. Leider gibt es eine vielzahl verschiedener Wasseruhren die sich nicht alle ohne weiteres auslesen lassen. Ich nenne hier lediglich ein paar Sensoren die es Möglich machen könnten.

- TCRT5000
- KY033
- Induktiver Näherungsschalter LJ12A3-4-Z o.Ä. ein größerer Durchmesser ist Ratsam.
- Laser Lichtschranke

Gasuhr

Eine Weitverbreitete Gasuhr ist der Gaszähler BK G4.
Viele dieser Modelle haben einen Magneten der sich mit einem einfachen Reedrelais auslesen lässt.
Alternativ haben auch einige der Zähler eine spiegelnde Fläche im Innenkreis der Zahl 6.

- Reedrelais
    - Meder M04 o.Ä.
    - Glaskolben Reed Kontakt
- TCRT5000
- KY033
- Laser Lichtschranke
-

Diverses

Es kann sein das folgende Dinge nötig sind:

- div. Widerstände 120Ohm - 470kOhm
- Patchkabel
- Geschirmtes mehradriges Kabel.
- Lötkolben + Lötzinn + Lötpumpe + Lötlitze
- Isolierklebeband
- Gaffa
- Magnet
- Heißkleber
- Geduld
- Zeit

Tasmota Firmware kompilieren

In der Standard Tasmota Firmware ist der SML Treiber nicht vorhanden. Deshalb muss eine eigene Version der Firmware kompiliert werden.
Dazu gibt es verschiedene Möglichkeiten. Ich möchte hier zwei Möglichkeiten vorstellen:

1. PlatformIO via MS Visual Studio / Atom
2. GitPod für Google Chrome

PlatformIO

PlatformIO lässt sich sowohl auf Windows als auch OSX installieren. Dazu einfach die Seite https://platformio.org aufrufen und der Installationsanweisung für PlatformIO for VSCode folgen. Es besteht auch die Möglichkeit PlatformIO for ATOM zu wählen, jedoch habe ich bisher, selbst unter OSX, mit MS VStudio die besseren Erfahrungen gemacht.

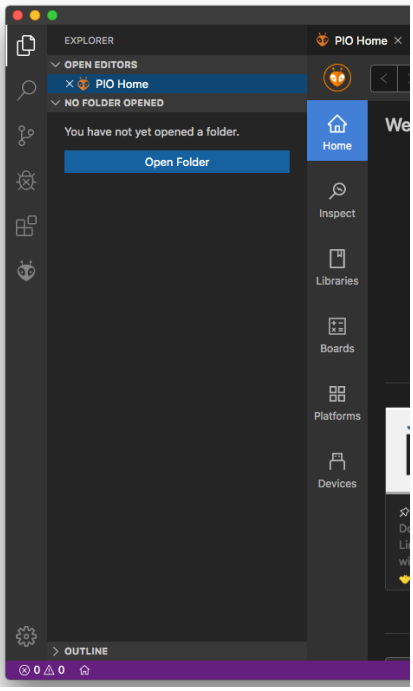
Ist dies geschehen kann entweder von dem Tasmota Fork von Gemu2015 heruntergeladen werden oder alternativ direkt von dem Tasmota Github.
Ich kann beide empfehlen, neuen Nutzern möchte ich zum Original Tasmota raten..
Der Fork von Gemu2015 dient der Entwicklung. Diese Beschreibung bezieht sich auf das Original Tasmota.

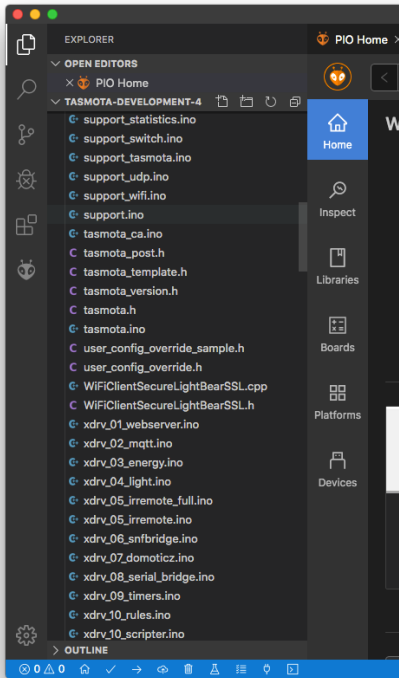
Tasmota Fork Gemu2015:
https://github.com/gemu2015/Sonoff-Tasmota

Original Tasmota:
https://github.com/arendst/Tasmota

Nach dem Download der Dateien wird der entpackte Ordner mit VSCode -> PlatformIO geöffnet.

In dem linken Bereich des VSCodes sieht man nun die Dateien des GithubTasmota Ordners.





Auch hier gibt es viele Wege die nach Rom führen, ich beschreibe hier einen:
Fangen wir an mit der Datei: platformio_override_sample.ini.

Diese Datei muss umbenannt werden in: platformio_override.ini.
Das gleiche gilt für die Datei user_config_override_sample.h die in user_config_override.h umbenannt werden soll.
Die zuletzt genannte Datei befindet sich im Unterordner “tasmota”.

In der Datei platformio_override.ini wird folgendes ergänzt bzw. durch löschen des Semicolon aktiviert:

Auswahl des Enviroments. Die Menüsprache definiert ihr in der “user_config_override.h” Datei. Alternativ, falls ihr es nicht in der user_config_override.h - Datei machen wollt, könnt ihr hier an dieser Stelle “tasmota” mit tasmota-DE” ersetzen.

```
default_envs =
; *** Uncomment the line(s) below to select version(s)
               tasmota
```

Aktivieren der user_config_override.h Datei. Ohne dies, sind alle Änderungen in der “user_config_override.h” Datei umsonst.

```
; *** Use settings from file user_config_override.h
                         -DUSE_CONFIG_OVERRIDE
```

Auswählen des USB Ports zum Upload der FW.

```
; *** Upload Serial reset method for Wemos and NodeMCU
upload_port               = /dev/cu.wchusbserialfd120
```

Der upload_port muss entsprechend dem Port angepasst werden, an dem der ESP8266 angeschlossen ist. Das kann bei Windows z.B. COM5 oder bei OSX /dev/cu.usbserialfd120 (Beispiel) sein.

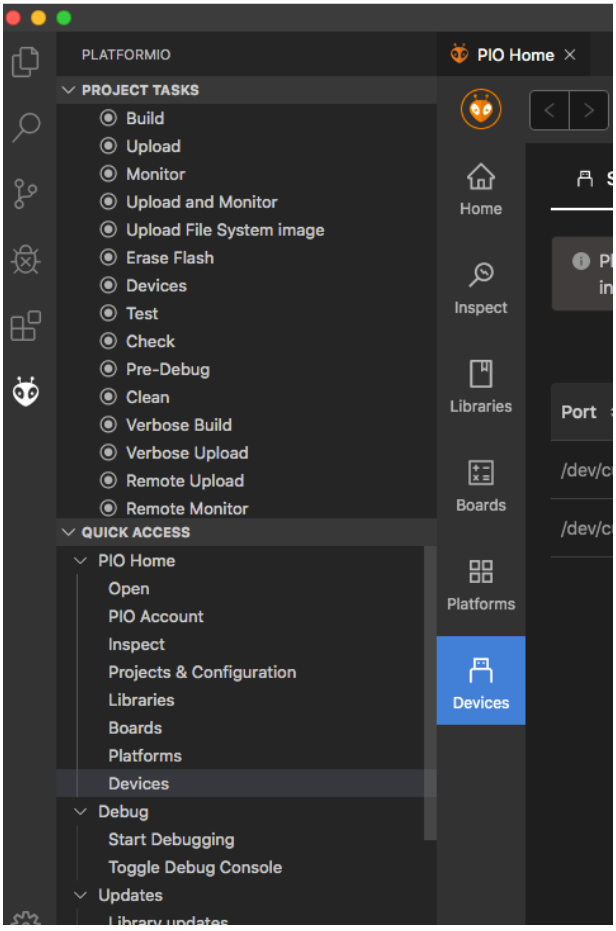
OPTIONAL: Wenn Ihr ein Entwicklerboard mit 4MB Flash Speicher verwendet, könnt ihr durch hinzufügen folgender Zeile in der Sektion “common”, den Flash Speicher auf 4MB statt 1MB definieren. Falls ihr keine Ahnung habt wovon ich schreibe, lasst einfach alles wie es war.

```
[common]
board_build.ldscript      = eagle.flash.4m.ld
```

Es gibt hier ein paar Dinge die man wissen muss.
Die nachträgliche Änderung des Flashspeichers eines bereits mit Tasmota geflashtem Gerät ist zwar Möglich, führt aber zu vielen Problemen. U.A. MQTT und WIFI Fehlern. Es muss IMMER auch bei späteren Updates via OTA die gleiche Flash größe genommen werden.
Das ganze funktioniert auf dieser Art nur, wenn ihr zuvor den Flash über “Erase Flash” gelöscht habt und die Software via USB2TTL auf euer Entwicklerboard flasht.
Des Weiteren muss hierzu eine aktuelle Version der Original Tasmota FW. verwendet werden, die mind. Version. 8.1.0.11 ist.

Es lässt sich herausfinden indem man im PlatformIO Menü auf Devices klickt während der ESP8266 angeschlossen ist. In der Serial im unteren rechten Fenster wird nun eine

Auflistung der angeschlossenen Geräte angezeigt.



Geht danach im linken Menü wieder auf das Symbol mit den zwei übereinanderliegenden Blättern und sucht im Ordner “tasmota” die von euch umbenannte Datei: user_config_override.h.

In der Datei user_config_override.h wird folgendes angepasst/hinzugefügt:

Fügt die Dinge, die ihr benötigt bitte direkt in eine freie Zeile vor dieser Zeile:

```
#endif // _USER_CONFIG_OVERRIDE_H_
```

ein.

Menüsprache:

```
#define MY_LANGUAGE de-DE // German in Germany
```

Richtige Timezone: (Deutschland inkl. Sommer/Winterzeit)

```
#define APP_TIMEZONE 99
```

Es kann nur entweder Rules oder Script verwendet werden. Wir benötigen für den SML Treiber: “Script”.

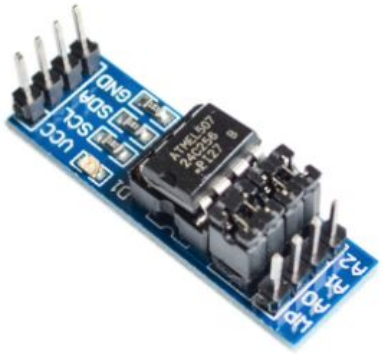
```
#ifndef USE_SCRIPT
#define USE_SCRIPT //# adds about 17k flash size, variable ram size
#endif
#ifdef USE_RULES
#undef USE_RULES
#endif
```

Das Wichtigste, der SML Treiber:

```
#define USE_SML_M
```

OPTIONAL: Um verschiedene Funktionen des Scripters freischalten zu können, müssen einige weitere #defines hinzugefügt werden. Wozu diese dienen, lässt sich am besten im Tasmota Wiki - Abschnitt Scripting, welches ich in der Linksammlung am Ende des Dokuments hinterlegt habe, nachlesen. Beispielsweise: Sendmail, Buttons, Alexa HUE emulation, SDCARD, EEPROM, uvm.

```
#define USE_SCRIPT_WEB_DISPLAY // Modify webUI
#define USE_SCRIPT_JSON_EXPORT // Add support for >J Section. Publish JSON payload
on TelePeriod.
#define USE_EXPRESSION       // Add support for expression evaluation in rules
(+3k2 code, +64 bytes mem)
#define SUPPORT_IF_STATEMENT // Add support for IF statement in rules (+4k2 code,
-332 bytes mem)
```



Optional gibt die Möglichkeit die Anzahl der zur Verfügung stehenden Zeichen im Tasmota Scripteditor mit einem externen I2C EEPROM zu erweitern. (**24C256**)

Sollte dieses Eeprom vorhanden sein, muss es via I2C an dem ESP8266 angeschlossen werden. Die entsprechenden Pins (SDA/SCL) müssen später in den Geräteeinstellungen in der WEBGUI der Tasmotafirmware eingestellt werden.

Damit das EEPROM funktioniert muss auch dieser Treiber geladen werden und dem Scripteditor verfügbare Größe des Speichers definiert werden:

```
#define USE_I2C
#define USE_24C256
#define EEP_SCRIPT_SIZE 4095
//#define EEP_SCRIPT_SIZE 5120
//#define EEP_SCRIPT_SIZE 6143
//#define EEP_SCRIPT_SIZE 7168
//#define EEP_SCRIPT_SIZE 8191
```

Es gibt noch einige Sachen die etwas spezieller sind und die nur eine Eintragung in die user_config_override.h Datei benötigen, falls es hiermit **Probleme** gibt. Ansonsten überspringt es einfach.

```
#define SML_MAX_VARS 30
```

Hiermit wird bestimmt wieviel Variablen für den Descriptor zur Verfügung stehen.
Dazu später mehr.

Optional Wifi/MQTTSetup: Um Zeit zu sparen, ist es Möglich direkt seine Wifi und MQTT Daten einzugeben. würde dann so aussehen:

```
#undef STA_SSID1
#define STA_SSID1 "HIERDERNAMEDESWIFINETZWERKS" // [Ssid1] Wifi SSID

#undef STA_PASS1
#define STA_PASS1 "HIERDASWIFIPASSWORT" // [Password1] Wifi password

#undef MQTT_HOST
#define MQTT_HOST "IPDESMQTTHOSTS" // [MqttHost]

#undef MQTT_PORT
#define MQTT_PORT 1883 // [MqttPort] MQTT port (10123 on CloudMQTT)

#undef MQTT_USER
#define MQTT_USER "MQTTUSER" // [MqttUser] Optional user

#undef MQTT_PASS
#define MQTT_PASS "MQTTPASSWORT" // [MqttPassword] Optional password
```

Optional Um evtl. Speicher zu sparen, lässt sich alles, was nicht benötigt wird auch abschalten:
Dazu könnt ihr folgende Liste vor den von euch erstellten Änderungen hinzufügen:
Falls ihr bestimmte Treiber aus dieser Liste benötigt, kommentiert sie mit einem "//" aus, damit sie weiterhin geladen werden. HomeAssistant könnte hier für einige ein Stichwort sein welches auskommentiert werden muss.

```
#undef USE_DOMOTICZ
#undef USE_HOME_ASSISTANT
#undef USE_EMULATION_HUE
#undef USE_EMULATION_WEMO
#undef USE_TIMERS
#undef USE_SONOFF_RF
#undef USE_SONOFF_SC
#undef USE_TUYA_MCU
#undef USE_ARMTRONIX_DIMMERS
#undef USE_PS_16_DZ
#undef USE_SONOFF_IFAN
#undef USE_BUZZER
#undef USE_ARILUX_RF
#undef USE_DEEPSLEEP
#undef USE_WS2812
#undef USE_MY92X1
#undef USE_SM16716
#undef USE_SM2135
#undef USE_SONOFF_L1
#undef USE_COUNTER
#undef USE_DS18x20
#undef USE_SHT
#undef USE_HTU
#undef USE_BH1750
#undef USE_SHT3X
#undef USE_LM75AD
#undef USE_ADE7953
#undef USE_MHZ19
#undef USE_SENSEAIR
#undef USE_PMS5003
#undef USE_NOVA_SDS
#undef USE_SERIAL_BRIDGE
#undef USE_ENERGY_MARGIN_DETECTION
#undef USE_PZEM004T
#undef USE_PZEM_AC
#undef USE_PZEM_DC
```

```
#undef USE_MCP39F501
#undef USE_DHT
#undef USE_IR_REMOTE
#undef USE_IR_RECEIVE
#undef USE_SR04
#undef USE_HX711
#undef USE_BMP
#undef USE_WS2812
#undef USE_MY92X1
#undef USE_SM16716
#undef USE_SM2135
#undef USE_SONOFF_L1
#undef USE_PZEM004T
#undef USE_PZEM_AC
#undef USE_PZEM_DC
#undef USE_MCP39F501
#undef USE_SHUTTER
#undef USE_PWM_DIMMER
#undef USE_SONOFF_D1
```

Kompilieren/Upload

Es gibt zwei Möglichkeiten die Firmware auf euer ESP8266 zu bringen.

Falls ihr schon eine recht aktuelle Tasmota Version auf eurem Board habt, könnt ihr dies via OTA (OverTheAir) tun.
Ist dies nicht der Fall, müsst ihr euren ESP8266 an euer PC/MAC anschließen.

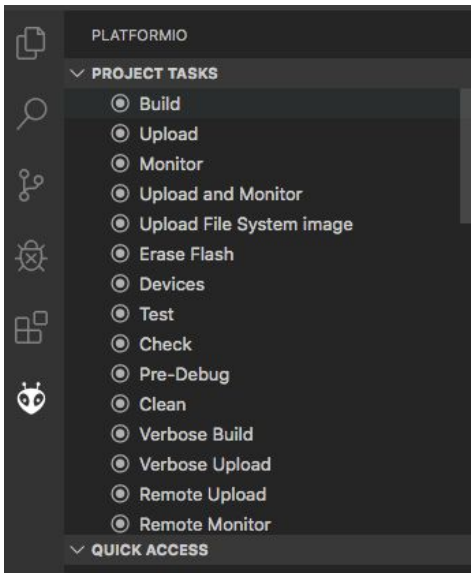
Upload via Kabel

Der Upload via Kabel funktioniert meistens nur, wenn ihr zu diesem Zeitpunkt nichts an den Pins 3 & 1 also RX/TX am Entwicklerboard angeschlossen habt. Ein Phototransistor würde z.B. die Kommunikation via USB stören.

Geht im PlatformIO Menü auf Upload&Monitor. Sofern ihr euren ESP8266 richtig angeschlossen habt und auch der entsprechende Port in der patformio_override.ini korrekt ist, kompliliert PlatformIO nun den Code und erstellt daraus eine “.bin” - Datei. Diese wird direkt nach dem erstellen auf euer Board geladen. Wenn das alles korrekt abgelaufen ist, sollte eine grüne Schrift am ende des Vorgangs zu sehen sein, kurz bevor die

Monitorkonsole (unten rechts) geöffnet wird. Alternativ ist es auch an den Daten zu erkennen, die in der Monitorkonsole unten rechts zu sehen sind.
Falls ihr bereits eure WiFi-Daten in die user_config_override.h Datei eingegeben habt, sollte sich der ESP8266 innerhalb kürzester Zeit mit eurem WLAN verbinden. Dies sollte auch im Serialmonitor-Fenster unten rechts zu sehen sein. Dort müsste die aktuelle IP Adresse eures ESP8266 stehen. Ist dies nicht der Fall, schaut auch gerne noch einmal in eurem Router nach ob ihr Ihn dort finden könnt.

Habt ihr eure WiFi-Daten nicht eingegeben, sollte die Tasmotafirmware einen eigenen AccessPoint erstellen, auf den ihr euch via WLAN einwählen könnt um dort eure WiFi-Daten einzutragen.



Upload via OTA:

Geht im PlatformIO Menü auf Build. Nun wird der Code kompliliert. Es wird eine ".bin"-Datei erzeugt. Diese findet ihr im versteckten Unterordner eures Projektverzeichnises (also des entpackten Tasmota Verzeichnisses). Der Name des versteckten Ordners lautet ".pioenvs". Dort findet ihr einen weiteren Ordner der tasmota-DE lauten sollte. Darin befindet sich eine Datei: "firmware.bin".

Nachtrag: Seit dem 08.01.2020 erstellt PlatformIO anscheinend einen nicht versteckten Ordner im Tasmota Hauptverzeichens mit dem Namen: "build_Output" in dem die Firmware Datei zu finden ist.

In der WEBGUI, die ihr über die IP - Adresse eures ESP8266 erreicht klickt ihr auf Firmware Update um dort die besagte "firmware.bin" - Datei auszuwählen und den Upload zu beginnen.

Diese Methode eignet sich gut, falls ihr im Programmcode noch Änderungen angepasst habt, oder Updates installieren wollt.

Hardware vorbereiten

Startet euren ESP8266 und ruft die WEBGUI mit der Eingabe der IP des ESPs auf. Seht ihr die GUI? Top!
Nehmt euren ESP8266 wieder vom Strom und bereitet die Hardware vor.

OBIS/SML Stromzähler via IR

Grundlagen:

Informiert euch bei Google, Energieunternehmen oder im Handbuch eures Stromzählers über selbigen.
IST ES ÜBERHAUPT MÖGLICH IHN OPTISCH AUSZULESEN?
Versucht so viele Informationen wie möglich zusammenzutragen. In welchem Format sendet euer Zähler? SML oder OBIS? Mit welcher Baudrate? Wo befindet sich die IR-LED? Was ist SML oder OBIS überhaupt? Gibt es auch eine Empfangs-Diode am Zähler? Benötigt mein Zähler eine bestimmte Zeichenfolge bei einer bestimmten Baudrate damit er seine Daten sendet oder sendet er sowieso alle 3 Sekunden seine Daten? Ist mein Zähler überhaupt freigeschaltet für diese Dinge?

Wenn ihr das geklärt habt:

Verbindet eure BPW78A wie auf dem Schaltbild gezeigt mit dem ESP8266 Board.

Wenn ihr die Hardware Serielle Schnittstelle des ESP8266s verwenden wollt (meine Empfehlung), schließt sie an RX(GPIO3) an. Leider gibt es einige Boards bei denen es offensichtlich Probleme mit dem Pullup Widerstand des RX Eingangs gibt. (Wemos D1 Mini v3.x) Ist dies der Fall wählt einen anderen GPIO. Alternativ könnt ihr also die SoftwareSerial benutzen indem ihr z.B. GPIO12 nehmt.

Optional:
Falls euer Stromzähler erst ein Signal benötigt, das ihn dazu nötigt seine Daten Preis zu geben, schließt eine IR-LED über einen Transistor an einen anderen GPIO an.

Platziert die beiden optischen Bauteile so nah wie möglich an euren Stromzähler. Wichtig ist, eure beiden Bauteile dürfen sich nicht untereinander “sehen”. Dazu müsst ihr also evtl. ein schwarzes Stück plastik oder etwas Holz zwischen beide Bauteile bringen. Dies ist natürlich nur nötig falls ihr sowohl sendet als auch empfangt. Habt ihr einen Zähler, der sowieso ständig seine Daten sendet, benötigt ihr natürlich keine Sende Diode am ESP8266 und dementsprechend auch keinen optischen Schutz zwischen euren beiden Bauteilen.

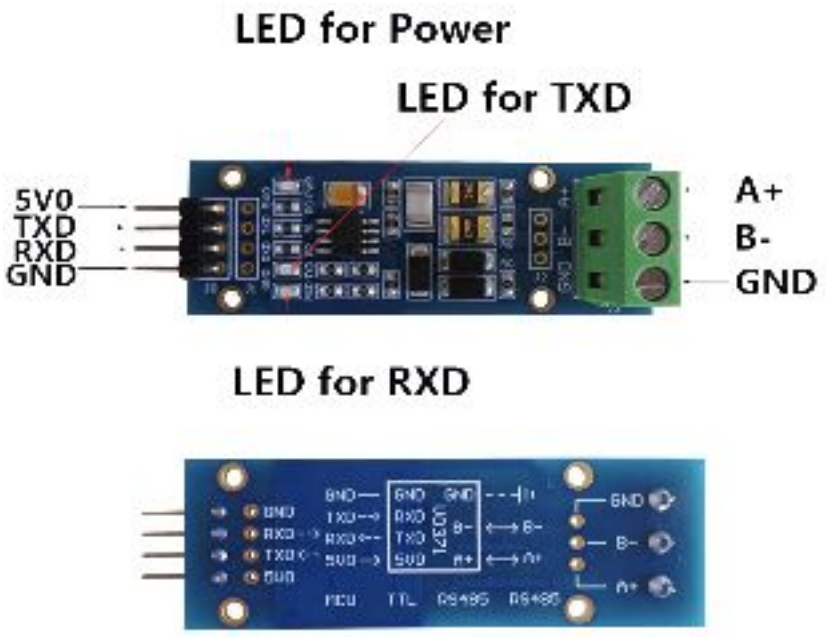
Dann lest nun weiter bei “Software Einrichten / SML/OBIS Stromzähler”.

Stromzähler MODBUS RTU

Ich möchte noch einmal darauf hinweisen, dass jegliche Arbeiten an elektrischen Anlagen durch eine Elektrofachkraft auszuführen sind. Es besteht Lebensgefahr!

Zum auslesen eine Modbus-fähigen Zählers, oder auch Zwischenstromzählers benötigt man einen Adapter (RS485->TTL).

Den Modbuszähler verbindet ihr mit einem geschirmten zwei-adrigen Kabel mit dem RS485 Adapter. Für gewöhnlich Haben die Adapter einen A+ und einen B- Anschluss. Diesen müsst ihr mit eurem Modbuszähler verbinden. Es kann vorkommen, dass die Beschriftung der Zähler von dieser A/B Variante abweicht. Da ist ein wenig rumprobieren von nöten. Der Adapter muss nun via RX/TX 5V oder 3V und GND an euren ESP angeschlossen werden. Ich rate dazu, den Adapter an GPIO3 und GPIO1 also RX und TX des ESP's anzuschließen. Dies funktionierte bei mir am besten.
Man muss jedoch beachten, dass bei gesteckter Verbindung, das Flashen via USB nicht möglich ist.
Des Weiteren betreibe ich meinen Adapter mit 5V. Das führt dazu, dass das Signal welches via RX/TX vom Adapter kommt einen HIGH -Pegel hat von nahezu 5V. Eigentlich sollen die GPIOs vom Wemos D1 Mini bzw. vom ESP8266 nicht vertragen. Allerdings funktioniert dies bei mir dennoch (Ein Artikel dazu: https://github.com/tttapa/ESP8266/issues/61).



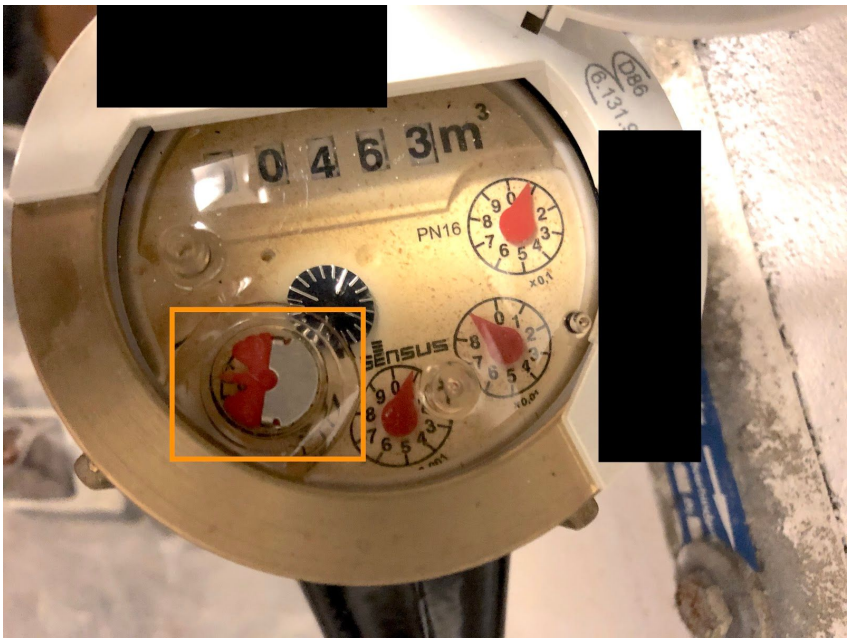
[1]Leider ist auch die Beschriftung der Adapter dies es überall zu kaufen gibt, oft nicht eindeutig was ihre RX/TX Beschriftung angeht. Einige Adapter haben auf ihren PCBs die Eingänge schon vertauscht, andere nicht. Auch hier gilt leider, wie so oft: Ausprobieren.

Am Modbuszähler selbst muss eine Adresse/ID ausgewählt werden und u.U. auch die Baudrate bestimmt werden. Hier kann man getrost 9600Baud wählen. Als Adresse kann man jede X-beliebige nehmen, ich empfehle allerdings die 1 falls nicht bereits weitere Zähler am gleichen Modbusdatenstrang hängen.

[1] DSD Tech SH-U12 Bildquelle Amazon.de

Wasserzähler

Einige Wasserzähler besitzen bereits eine werksseitige Vorbereitung um bestimmte Impulse zählen zu können.
Diese Zähler sind oft daran zu erkennen, dass sie ein Metallplättchen auf einen Ihrer Zeiger haben:



(Oranger Kasten)

Es gibt inzw. einige Zähler die so ein Plättchen aufweisen. Leider ist dies nicht bei allen Zählern der Fall.

Diese Art der Wasserzähler lässt sich zum Beispiel mit einem Induktiven Näherungssensor “auslesen”.
Man kann hier nicht direkt von “auslesen” sprechen, weil dieser Näherungssensor natürlich keine Werte wie etwa ein SML/OBIS Stromzähler liefert sondern lediglich einen Impuls.
Das heißt, dass der aktuelle Verbrauch bei der Montage des Sensors dem SML Treiber mitgeteilt werden muss. Natürlich muss dieser Wasserzähler vorher als “Counter” im Skript angelegt werden. Auch muss darauf geachtet werden, welche Größe ein Impuls liefert. Der Zähler auf diesem Bild gibt 1L/Impuls aus.

Zur Montage des Näherungssensors, eignet sich am besten eine große Schelle und ein Winkel mit passender Bohrung. Am besten ist es, wenn der Winkel eine verlängerte Bohrung (ein Schlitzes) besitzt um den Sensor bei der Montage besser justieren zu können. Mit etwas geschick und Werkzeug lässt sich dies also einfach realisieren. Trotzdem der Näherungssensor eigentlich eine Spannung von 6-12 V verlangt, habe ich tatsächlich die besten Erfahrungen bei einem Betrieb mit 5V gemacht.

Das auf dem Bild zu sehende Modell ist ein: “LJ12A3-4-Z” NPN 3-adrig.
Es gibt dieses Modell auch dicker mit einer höheren Distanz. Würde ich einen neuen benötigen, würde ich heute zu einem größeren Model greifen. Die Distanz zwischen Plättchen und Sensor ist scharf an der Grenze obwohl der Sensor schon auf dem Glas aufliegt.
Alternativ, sollte es auch möglich sein, mit einer Reflexionslichtschranke das glänzende Plättchen zu erkennen und hieraus einen Impuls zu generieren. Probiert habe ich dies allerdings nie.

Wie bereits eingangs erwähnt, gibt es auch Wasserzähler ohne dieses Plättchen. Eine Zählung bei diesen Geräten umzusetzen erweist sich als deutlich schwieriger.
Es gibt hierzu viele verschiedene Ansätze im Internet zu finden.

Erwähnenswert sind hierbei die Beispiele mit einer Lichtschranke inkl. Linse vor der LED oder auch Lichtschranken mit Laserlicht. Ich persönlich befinde mich hierbei noch in der Testphase.
Ich finde es jedoch wichtig zu erwähnen, dass die Möglichkeit besteht, dass sich Algen am Uhrenglas bilden können. Dies sei wohl zurückzuführen auf die dauerhafte beleuchtung durch die LED. In wie weit

Gaszähler

Infos folgen…

Modbus Zwischenzähler

Infos folgen...

Software Einrichten

SML/OBIS Stromzähler:

Ihr seid euch sicher, dass ihr alles korrekt angeschlossen habt? Keinen Draht vertauscht? Gut! Schließt euren ESP wieder an einer Stromversorgung an und überprüft ob irgendetwas heiß wird oder verbrannt riecht. Falls ja! STROM SOFORT ABZIEHEN und überprüfen was falsch läuft.
Wenn ihr alles korrekt angeschlossen habt sollte der ESP nicht viel wärmer als Handwarm werden. Der Fototransistor und sein Pullupwiderstand dürften keine wärme absondern.

Ruft nun also die WEBGUI auf und geht dort auf Einstellungen / Skript.

Setzt einen Haken um das Skript zu aktivieren.

Öffnet einen neuen Tab und ruft:
https://tasmota.github.io/docs/#/peripherals/Smart-Meter-Interface
auf.

Lest es euch durch!
Ihr werdet vielleicht einige Dinge die wir bereits durchgeführt haben, wiedererkennen.
Im unteren Teil der Seite findet ihr Beispielskripte von verschiedenen Zählern. Schaut mal ob euer dabei ist.

Ist er es? Top!
Kopiert den Code in euer Script Editor, speichert das Skript und ruft das Hauptmenü auf. Sollte alles korrekt funktionieren, müsstet ihr jetzt schon eure Daten sehen.

Ist er nicht dabei? Schade!

Es gibt nun unzählige Möglichkeiten dieses Problem anzugehen.
Wenn ihr am Anfang einen guten “research” betrieben habt, solltet ihr von Google oder der Betriebsanleitung vielleicht schon wissen, welche OBIS oder SML Strings euer Zähler sendet.
Lest bitte noch einmal den vorangegangenen Link durch. Ihr solltet verstehen wie der Code funktioniert.
Das nun hier auszuführen würde nur doppelte Arbeit bedeuten, deshalb gehe ich ab hier davon aus, dass gewisse Grundkenntnisse über den nötigen Syntax vorhanden sind.

Wenn ihr im Internet schon die passende entschlüsselung für euren Zähler gefunden habt ist das super, dann könnt ihr mit etwas geschick die entsprechenden Daten für euch einfach Eintragen. Wie das geht könnt ihr von der folgenden Beschreibung ableiten.
Auch hier heißt es wieder, viele Wege führen nach Rom;

Ich beschreibe hier einen Weg:
Da ihr ja inzw. recht viel über euren Zähler wisst, wisst ihr nun auch ob es sich um einen OBIS oder SML Zähler handelt.
Je nachdem um was es sich handelt, nehmt einen der auf der verlinkten Seite bereitgestellten Codes für OBIS oder SML und kopiert diesen in euren Skript Editor.

Beispiel OBIS:

```
>D

>B

=>sensor53 r

>M 1

+1,3,o,0,9600,OBIS

1,1-0:1.8.1*255(@1,Total consumption,KWh,Total_in,4

1,1-0:2.8.1*255(@1,Total Feed,KWh,Total_out,4

1,=d 2 10 @1,Current consumption,W,Power_curr,0

1,1-0:0.0.0*255(@#),Meter Nr,, Meter_number,0

#
```

Beispiel SML:

```
>D

>B

=>sensor53 r

>M 1

+1,3,s,0,9600,SML

1,77070100010800ff@1000,Total consumption,KWh,Total_in,4

1,77070100020800ff@1000,Total Feed,KWh,Total_out,4

1,77070100100700ff@1,Current consumption,W,Power_curr,0

1,77070100000009ff@#,Meter Nr,,Meter_number,0

#
```

Wie ihr aus dem Tasmota Wiki wisst, bedeutet die Zahl hinter dem “>M” dass nur ein Meter (ein Gerät) nachfolgend definiert wird.

Deshalb findet ihr in der nächsten Zeile:

```
+1,3,o,0,9600,OBIS
```

+1 definiert den ersten Zähler
3 gibt den GPIO an an dem euer Fototransistor angeschlossen ist (in diesem Falle RX).
o steht für OBIS (weitere Alternativen bitte dem Wiki entnehmen)
0 hat bei einem Stromzähler keine Bedeutung. Ersetzt man die 0 mit einer 16 so wird aus den gelesenen Daten ein Mittelwert gebildet.
9600 ist die Baudrate
OBIS ist nur der Name des Zählers. Der wird später im Hauptmenü angezeigt. Dort kann also auch Heinzwurst stehen.

Das gleiche Schema gilt für die definition eines SML Zählers. Der einzige Unterschied ist, das statt eines “**o**” ein “**s**” in der Zeile vorhanden ist.

Es folgt der Descriptor:

```
1,1-0:1.8.1*255(@1,Total consumption,KWh,Total_in,4
```

1 gibt an, dass es sich um einen Wert handelt der zum definierten Stromzähler 1 gehört.
1-0:1.8.1.*255(Ist der Vergleichsstring, den der Stromzähler sendet. Hierzu gleich mehr.
@1 gibt an das der Vergleichsstring hier zuende ist und 1 definiert die Wertigkeit des eingelesenen Wertes. (Siehe Wiki)

Total consumption Ist der Name der später hinter dem Namen des Zählers (Hanswurst) steht. Auch dieser kann frei gewählt werden.

KWh gibt die Einheit an.

Total_in So lautet die Bezeichnung dieses Wertes für das MQTT Protokoll. Auch dieser darf frei gewählt werden, jedoch eine bestimmte Länge nicht überschreiten.

4 definiert die Dezimalstellen. 4 Nachkommastellen.

Ganz ähnlich sieht es auch bei einem SML Zähler aus.
Lediglich der Vergleichsstring unterscheidet sich. Sendet ein OBIS Zähler seine Daten in eine für uns leserliche Art, ist dies nicht der Fall beim SML Zähler.

```
1,77070100010800ff@1000,Total consumption,KWh,Total_in,4
```

Soviel zur Theorie. Nun wollen wir aber wissen ob das, was wir gebaut haben(Hardware), auch funktioniert.
Wir haben also einen Code genommen (auch wenn er nicht für unseren Zähler ist) der unserem Protokoll OBIS/SML entspricht.
Das Skript wird gespeichert und wir gehen zurück ins Hauptmenü. Falls wir änderungen vorgenommen haben, sollten diese nun auch im Hauptmenü zu sehen sein. Das heißt, wenn wir unseren Zähler “Hanswurst” genannt haben, sollte dort nun auch Hanswurst stehen.
Ist das nicht der Fall. Ist etwas schief gelaufen. Evtl. wurde der Haken zum aktivieren des Skriptes nicht gesetzt.

Optional Sende Diode:
Für den Fall, dass eine Sendediode von nöten ist, muss diese deklariert werden.
+1,3,o,0,9600,OBIS,1,10,2F3F210D0A
Nach dem Zählernamen “OBIS” wird der TX GPIO angegeben (Der Pin an dem die Sendediode hängt)(1).
10 gibt die Verzögerung der Sendesequenz an. Sie wird mit 100ms multipliziert.
2F3F210D0A ist die Sendesequenz ASCII: “/?!” als Hexadezimal.

Der nächste Schritt wäre also ein Klick auf Konsole (in der WEBGUI) um dort folgendes einzutippen:

Sensor53 d1

Warum “d**1**”? Weil wir den Zähler als in unserem Skript als **ersten** Zähler definiert haben. Hätten wir zwei Zähler zum auslesen und würden wir erfahren wollen was der **zweite** Zähler so empfängt, dann würde es “d**2**” sein. Zum **deaktivieren** der Konsolenausgabe und zur **aktivierung** der WEBGUI-Ausgabe, muss “d**0**” gewählt werden.
Übrigens kann “Sensor53 d1” **nichts** ausgeben, wenn im Skript nicht die richtigen Eingänge/Ausgänge (GPIOs), wie im vorherigen Absatz beschrieben, definiert wurden.
Ist alles korrekt angegeben, führt es dazu, dass der descriptor deaktiviert wird und das was unser Fototransistor liest direkt in die Konsole geschrieben wird.
Was bedeutet das? Das sich ab jetzt im Hauptmenü keine Daten ändern werden sondern alles empfangene in die Konsole geschrieben wird.

Sollte eure Hardware funktionieren sieht das z.B. so aus:
OBIS:
*06:32:04 : 1.8.1(011324.383*kWh)*
06:32:04 :
*06:32:05 : 0.1.0*54(52)*
06:32:05 :
*06:32:06 : 1.8.1*54(014664.12)*
06:32:06 :
*06:32:06 : 1.8.2*54(0044444.220)*
06:32:06 :
*06:32:07 : 1.8.0*54(056512.216)*

SML:
9:45:22 : 1b 1b 1b 1b 01 01 01 01 76 05 aa 9d 0d
19:45:22 : 77 62 00 62 00 72 65 00 00 01 01 76 01 01 07 45 51 59 52 33 42 0b 09 01 45 53 59 11 03 98 0c 34 01 01 63 29 e0 00 76 05 aa 9d 0d 78 62 00 62 00 72 65 00 00 07 01
19:45:22 : 77 01 0b 09 01 45 53 59 11 62 01 64 07 9c 8d ff 76
19:45:22 : 77 07 81 81 c7 82 03 ff 01 01 01 01 04 45 53 59 01
19:46:05 : 77 07 01 00 01 08 01 ff 01 01 62 1e 52 01 65 00 18 14 aa 01

Bei Obis ist es nun relativ einfach herauszufinden was was ist, in dem ihr einfach auf euren Zähler schaut.

Nehmt nun also den in der Konsole angezeigten Vergleichsstring und ändert ihn in eurem Skript:

```
1,1-0:1.8.1*255(@1,Total consumption,KWh,Total_in,4
```

zu:

```
1,1.8.1(@1,Total consumption,KWh,Total_in,4
```

Bei SML ist es ähnlich.
Eine SML Nachricht fängt meines Wissens nach immer mit 1b 1b 1b an. Eine gute Erklärung hierzu findet ihr auf der Volkszähler Website:
https://wiki.volkszaehler.org/software/sml

In meinem Falle (Easymeter Q3B) könnte man z.B.:

```
1,77070100010800ff@1000,Total consumption,KWh,Total_in,4
```

zu:

```
1,77070100010801ff@1000,Total consumption,KWh,Total_in,4
```

ändern.

Speichert das Skript und gebt in der Konsole “Sensor53 d0” ein, damit das Hauptmenü wieder aktualisiert wird.

Nun solltet ihr also die korrekten Daten für diesen einen Datenpunkt im Hauptmenü sehen. Mit den anderen Daten geht ihr genauso vor.

Modbus RTU

Falls im Wiki oder im Forum keine Informationen zu dem von euch genutzten Modbuszähler zu finden sind, müsst ihr die Bedienungsanleitung eures Zählers hinzuziehen. Bei jeder ordentlichen Dokumentation eines Zählers, ist ein Auszug des Modbus Registers vorhanden. In der Bedienungsanleitung müsste es Informationen zur genutzen Parität und Baudrate geben. Sowie auch die nötigen Menüpunkte um am Zähler selbst die ID des Zählers zu wählen.
Das Register ist meistens am Ende einer Anleitung zu finden und ähnelt einer standard Exceltabelle mit vielen Zahlen und Spalten in denen die für das betreffende Register auszulesenden Daten beschrieben werden.

Ein Beispiel vom SBC ALE3:

R	Lesen	Schreiben	Beschreibung		Wert
32–33	X		Zähler T2 total Energiezähler total Tarif 2	0x1f 0x20	10^{-2} kWh (Multiplikator 0,01) Bsp.: Zähler T2 total Hoch = 13 Zähler T2 total Niedrig = 60'383 13 × 65'536 + 60'383 = 912'351 = 9123,51 kWh
34–35	X	X	Zähler T2 partial Energiezähler partial Tarif 2	0x21 0x22	10^{-2} kWh (Multiplikator 0,01) Bsp.: Zähler T2 partial Hoch = 13 Zähler T2 partial Niedrig = 60'383 13 × 65'536 + 60'383 = 912'351 = 9123,51 kWh
36	X		URMS Phase 1 Spannung Phase 1	0x23	V Bsp.: 230 = 230 V
37	X		IRMS Phase 1 Strom Phase 1	0x24	10^{-1} A (Multiplikator 0,1) Bsp.: 314 = 31,4 A
38	X		PRMS Phase 1 Wirkleistung Phase 1	0x25	10^{-2} kW (Multiplikator 0,01) Bsp.: 1545 = 15,45 kW
39	X		QRMS Phase 1 Blindleistung Phase 1	0x26	10^{-2} kvar (Multiplikator 0,01) Bsp.: 1545 = 15,45 kvar
40	X		Cos phi Phase 1	0x27	10^{-2} (Multiplikator 0,01) Bsp.: 67 = 0,67
41	X		URMS Phase 2 Spannung Phase 2	0x28	V Bsp.: 230 = 230 V
42	X		IRMS Phase 2 Strom Phase 2	0x29	10^{-1} A (Multiplikator 0,1) Bsp.: 314 = 31,4 A
43	X		PRMS Phase 2 Wirkleistung Phase 2	0x2a	10^{-2} kW (Multiplikator 0,01) Bsp.: 1545 = 15,45 kW
44	X		QRMS Phase 2		10^{-2} kvar (Multiplikator 0,01)

Es ist nicht genormt wie die Registerwerte angegeben werden. In manchen Anleitungen sind sie schon als Hexadezimal aufgeführt in anderen, so wie in dem Beispiel, leider nicht. Zu dem kann es noch weitere besonderheiten geben, zum Beispiel, dass die Werte

entsprechenden Register r-1 sind. Also statt in 32 - 33DEC in 31 - 32DEC. In dem Beispiel habe ich mir die entsprechenden Hexadezimalzahlen dazu geschrieben.

Ein weiterer wichtiger Wert ist der des “HoldingRegisters”.
Man kann es sich vielleicht wie ein Hochregallager vorstellen.
Die ZählerID benennt das Gebäude in dem sich das Lager befindet, das Holdingregister gibt das Lagerregal an und das Register bezeichnet in welchem Fach die Daten sind.

Zum Testen lässt es sich am besten mit einem leeren Skript beginnen:

>D
>B
=> sensor53 r

>M 1
+1,3,M,1,9600,SBC,1,2,01030023,01030028
1,010304UUuuxxxxxxxx@i0:1,Spannung L1,V,Voltage_L1,0
1,010304UUuuxxxxxxxx@i1:1,Spannung L2,V,Voltage_L2,0
#

Zählerdefinition:
1 = erster Zähler im Skript
3 = GPIO3 Also RX als Input
M = Modbus -> Parität 8E1. Alternativ wäre “m” für 8N1
1 = nur für Zählerdefinitionen vom Typ Counter wichtig. Hier kann auch eine 0 stehen.
9600 = Baudrate
SBC = Frei wählbarer Name (JSONPrefix)
1 = GPIO1 Also TX als Output
2 = TX Period. Die Zahl multipliziert mit 250ms gibt an wie oft eine Registeranfrage versendet wird.
01030023(HEX) = gibt Beispielhaft das erste abzufragende Register an: 01 ZählerID,03 Holdingregister,0023 Registernr.(Dem Bild entnommen).
01030028(HEX) = gibt Beispielhaft das zweite abzufragende Register an.

Descriptor:

1 = erster Zähler im Skript
010304(HEX) = Zähler Modbus ID 01 (Muss mit der im Zähler übereinstimmen), HoldingRegister 03, Anzahl Bytes 04 ?

UUuu = unsingned Word High order byte First. (Alternativen: Link)
xxxxxxxx = Die Zeichen der Antwort die vernachlässigt werden können. Die ersten 4 X sind das 2. Word (Registerauszug) und die letzten 4 X sind die CRC Checksumme.
@i0 = Die Zuordnung zum Anfrageregister in der Meterdef. (hier: 01030023) Man zählt von 0 an.
:1 = Scaling Faktor. Kann sowohl kleiner als auch größer 0 sein.

Spannung L1 = Label dieses Wertes in der WEBGUI.
V = Einheit (max. 7 Zeichen).
Voltage_L1 = MQTT Variablen Name.
0 = Anzahl Dezimalstellen

Soweit also zu der Theorie. Auch hier gilt: Recherche ist alles. Lest die Beschreibung gut durch. Ist euer Holdingregister vielleicht ein anderes? Habt ihr wirklich die korrekte ZählerID? Stimmt die Verkabelung? Seht ihr anhand der auf dem Adapter verbauten LEDs, dass der Zähler antwortet oder Ssendet nur der ESP etwas? Testhalber auch einmal die Kabel verdrehen. Mit und ohne Terminierung probieren.
Falls ihr eine Antwort vom Zähler bekommt jedoch keine korrekten Werte in der WEBGUI angezeigt bekommt, probiert es auch hier wieder einmal mit “sensor53 d1”.
Ihr solltet nun in der WEBGUI die entsprechenden Antworten des Zählers zu sehen bekommen. Macht es euch einfacher indem ihr im Skript (Meterdefinition) nur eine Anfrage stellt (z.B.:01030023). Dementsprechend dürftet ihr auch nur einen Wert als Antwort zurück bekommen.
Eine Antwort kann so aussehen:
LogUhrzeit | Wert
00:45:55 : 01 03 04 00 e9 00 02 99 06

Wie zu sehen ist, die ZählerID, das Holdingregister, 04 Anzahl Bytes, “ 00 e9 00 02” die zwei wichtigen Werte Paarungen! Die sind es die ausgelesen werden müssen. Die 00 02 Interessiert uns nicht wir brauchen die 00 e9 (HEX)=233(DEC). Also High order Byte first.
Es kann vorkommen das ihr die 00 02 dennoch benötigt, bzw. die anstelle der 00 02 ein anderer Wert steht. Also ein anderes Register. Dadurch lässt sich mit einer Registeranfrage so manches mal gleich zwei Werte auslesen.
So lässt sich mit der Entsprechenden Beispielanfrage (01030023) beim SBC ALE3 sowohl die Spannung L1 als auch der Strom L1 auslesen:

Empfangener Wert:
01 03 04 00 e9 00 02 99 06
Descriptor:
1,010304UUuuxxxxxxxx@i0:1,Spannung L1,V,Voltage_L1,0
1,010304xxxxUUuuxxxx@i0:10,Strom L1,A,Current_L1,2

In Zeile 1 fangen wir die ersten vier Zeichen ab, in der zweiten Zeile die hinteren vier Zeichen.
Die letzten vier XXXX stehen für den CRC Check und haben für uns keine Bedeutung und können deshalb durch die xx vernachlässigt werden.

Es ist nicht gesagt, dass euer Zähler seine Daten ebenso sendet wie dieser hier in dem Beispiel. Es kann sein, dass ihr die Daten als Float empfangt. Dann müsstet ihr statt UUuu oder uuUU vermutlich ffffffff oder FFffFFff verwenden. Eine Liste mit den möglichen Datentypen findet ihr hier: Link.
Vergesst nicht mit “Sensor53 d0” in der Webkonsole den Descriptor und damit die aktualisierung der Werte in der WEBGUI wieder einzuschalten.

Solltet ihr in eurem Descriptor so viele Werte abfragen, dass ihr über 20 Descriptor einträge kommt, müsst ihr in eurer user_config_override.h den Wert der möglichen Variablen erhöhen: `#define SML_MAX_VARS 30`

Wasserzähler

Infos folgen…

Gasuhr

Infos folgen…

Modbus Zwischenzähler

Infos folgen...

Tipps&Tricks oder auch: Gut zu wissen

1. **Sensor53 dX**
   “Fängt” sozusagen den ankommenden Datenverkehr ab und zeigt ihn in der Konsole an. Der Descriptor ist dadurch ausgeschaltet. Deshalb können und werden keine Daten aktualisiert im GUI Hauptmenü.
   X gibt den im SML Treiber deklarierten Zähler an. Also den ersten oder zweiten.. fünften Zähler.
   Zum deaktivieren des dumpen (abfangen) der Daten, wird in die Webconsole Sensor53 d0 eingegeben.
2. **#define SML_MAX_VARS X**
   X gibt die Anzahl der Variablen die dem Descriptor zur Verfügung stehen an. Hat man z.B. im Falle eines Modbus Zwischenzählers recht viele Descriptorzeilen, das sind die Zeilen unterhalb der Zählerdefinition, kann es vorkommen, dass nicht alle Werte abgespeichert werden können.
   Habt ihr dieses Problem, erhöht diese.
   Default = 20
3. **#define MAXVARS X**
   X gibt die Anzahl der nummerischen Variablen in der Script >D Sektion an. solltet ihr die Standard Anzahl überschreiten ist dies daran zu erkennen, dass in der Konsole nach dem abspeichern des Skripts, ein init -5 Fehler erscheint.
   Default = 50
4. **#define MAXSVARS X**
   X gibt die Anzahl der textlichen Variablen in der Script >D Sektion an.
   solltet ihr die Standard Anzahl überschreiten ist dies daran zu erkennen, dass in der Konsole nach dem abspeichern des Skripts, ein init -5 Fehler erscheint oder einige Variablen werden garnicht erst geladen.
   Default = 5
5. **>D X**
   In der >D Sektion können an der X-Position die maximale länge einer “String” (text) Variable angegeben werden.
   Dies gilt für alle zuvor definierten Text-Variablen, auch wenn sie kürzer sind.
   Wird Text den ihr in eine Variable abgespeichert habt, nicht vollständig angezeigt, könnte das eure Lösung sein.
   Obacht, Um so größer der Wert gewählt wird, um so mehr Speicher wird verbraucht, auch wenn die Variablen möglicherweise nicht komplett gefüllt sind.
6. **#define SCRIPT_DEBUG 1**
   Gibt mehr Informationen beim Speichern eines Skripts unabhängig vom SML Code.
   Zeigt beim Speichern folgende Werte an:
   i. nvXXX,tvXXX,vnsXXX,ramXXXX

ii. nv = Anzahl Number Variables
iii. tv = Anzahl Text Variables
iv. vns = Anzahl der verwendeten Zeichen für Variablen Variable Name Size. Max=256
v. ram = vom Scripter verwendeter RAM.

7. **Kurze Variablen Namen!**
   Haltet die Namen eurer Variablen extrem kurz, denn ihr habt nur einen begrenzten Speicher für diese. (256 Zeichen in Totale).
8. **Teleperiod X**
   Durch eingabe von Teleperiod X könnt ihr die Häufigkeit des MQTT Telegrams einstellen. Minimum: X = 10, Default = 300.

9. **"=>" ; Nutzen von Befehlen für die Konsole**
   a. mit => bzw -> können vom Script aus verschiedenste Befehle aufgerufen werden, die sich auch in der Webkonsole verwenden lassen.
   b. Der Unter zwischen =>,+>,-> ist:
      i. => Rekursionsausführung deaktiviert
      ii. +> Rekursionsausführung aktiviert
      iii. -> keine MQTT bzw. Protokollnachricht senden. "Stille ausführung"
   c. Ein Beispiel um alle 30 Sekunden den Befehl Publish (MQTT senden) auszuführen:

```
>S
;wird alle 30 Sek ausgeführt
if upsecs%30==0
then
=>Publish %s1%/%topic%/tele/Sensor-Gas %3v1%
=>Publish %s1%/%topic%/tele/Sensor-Strom %3v2%
=>Publish %s1%/%topic%/tele/Sensor-H2O %3v3%
=>Publish %s1%/%topic%/tele/Sensor-Power %3v4%
endif
```

10. **>J Sektion Eigene Variablen via MQTT zur Teleperiod senden**
    Damit dies funktioniert muss "#define USE_SCRIPT_JSON_EXPORT" in der user_config_override.h werden.
    In dem Webscript kann nun die Sektion >J hinzugefügt werden.
    Dinge die in dieser Sektion stehen, werden zu jeder Teleperiod via MQTT als JSON Payload gesendet.
    Beispiel:

```
>J
,"Regen_mm":%3r%
,"Strom_Heute":%3sd%
,"Strom_Vortag":%3vs%
```

11. **4MB Flash bei Wemos D1 Mini u.A.**
    Wer die nicht notwendigen 4MB Flashspeicher eines D1 Mini oder NodeMCU verwenden möchte, kann in der PlatformIO.ini oder noch besser in der PlatformIO_override.ini die Zeile: " -Wl,-Teagle.flash.1m.ld " suchen und diese

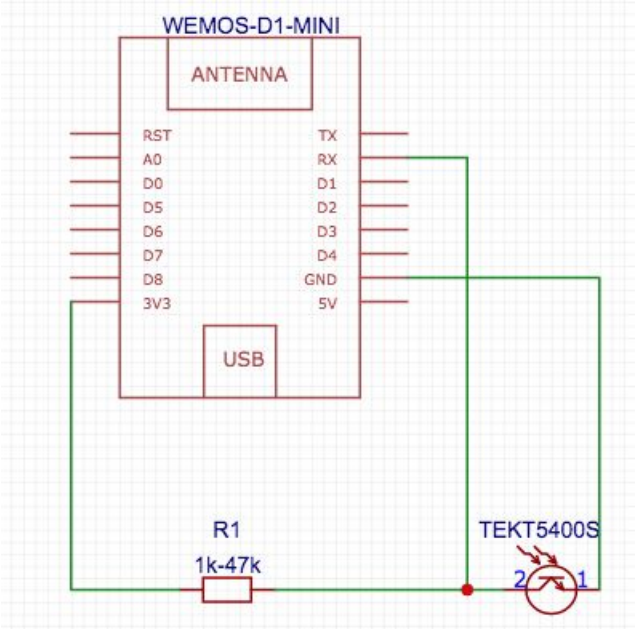
mit einem “;” auskommentieren und durch: “-Wl,-Tesp8266.flash.4m1m.ld ” ersetzen.
Die meisten Sonoff bzw. Shelly Hardware haben nur 1MB Flashspeicher.
Deshalb ist Tasmota so programmiert, das alles auch mit 1MB funktioniert.

12. 160Mhz statt 80Mhz

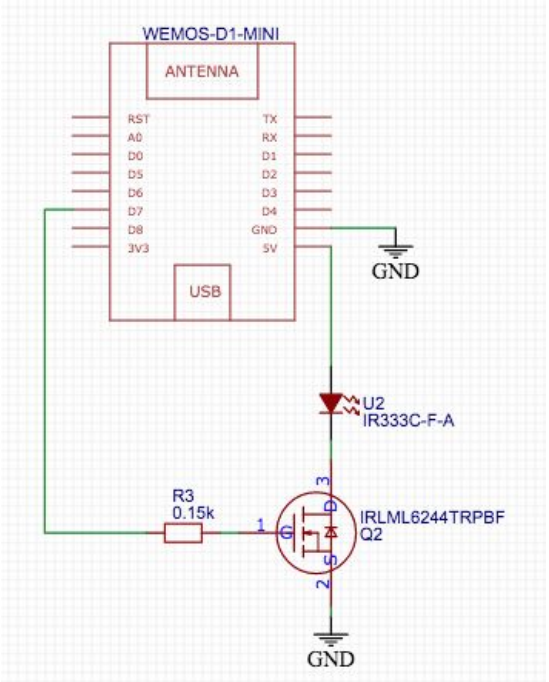
Alternativ kann man ebenfalls in der Platformio_override.ini die Zeile: “;board_build.f_cpu = 160000000L” suchen und das “;” am Anfang der Zeile entfernen um die Taktfrequenz des ESP8266 von 80Mhz auf 160Mhz zu erhöhen.
Konnte bisher keine Leistungsunterschiede erkennen.

Schaltbilder

IR-Fototransistor



IR-Transmitter mit n-Mosfet

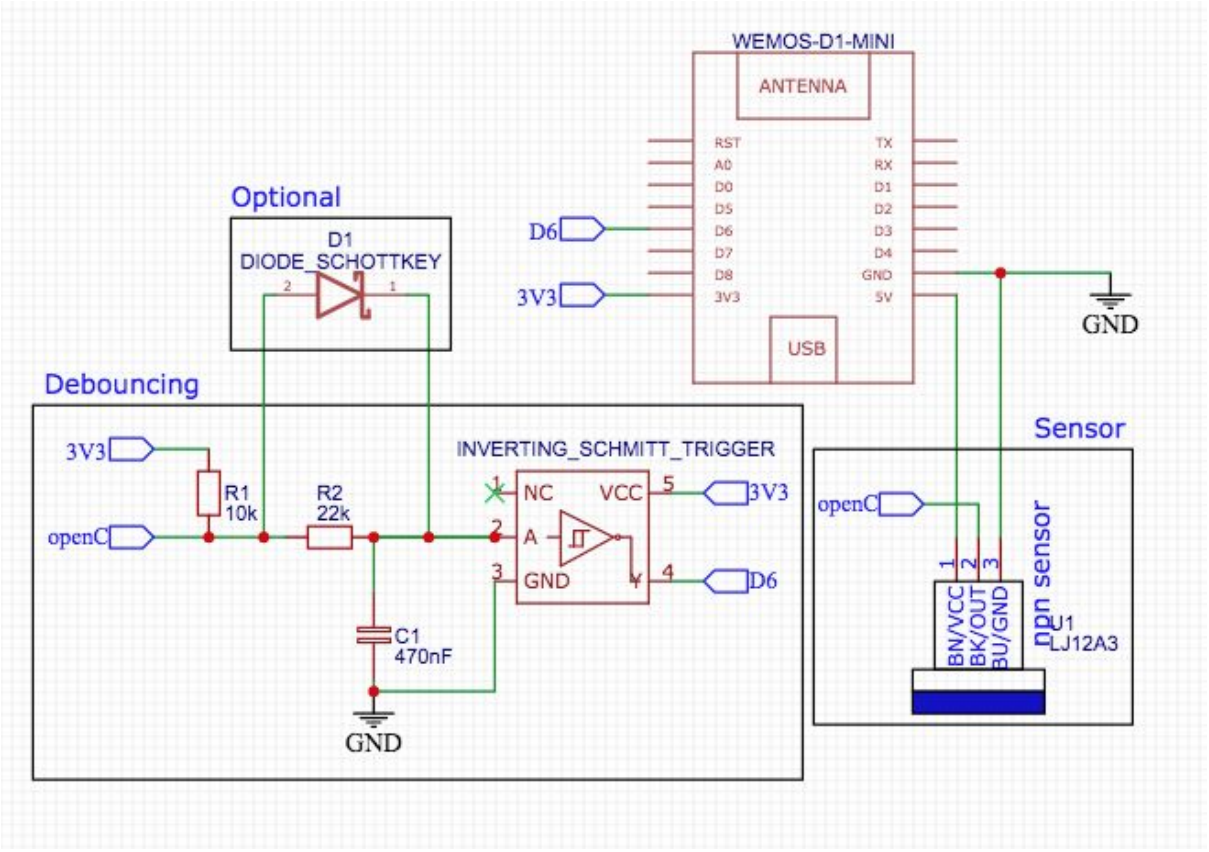


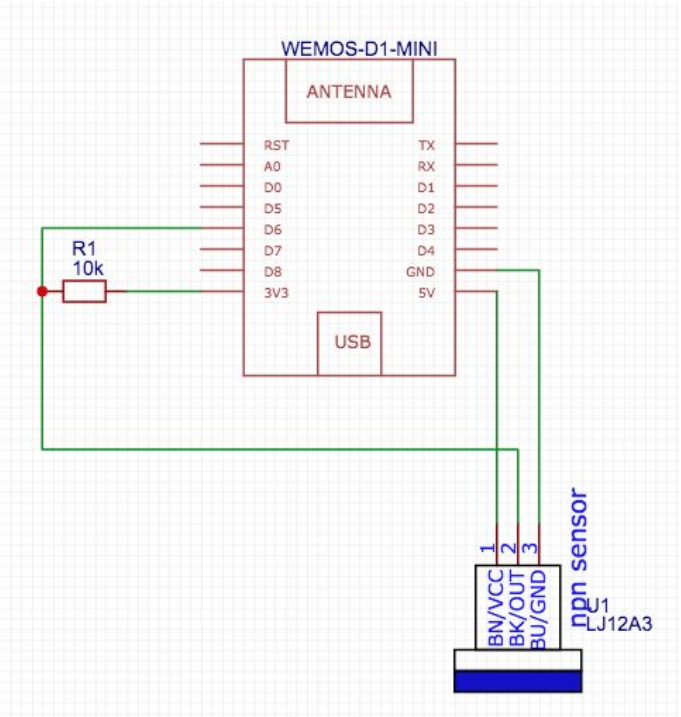
Induktiver Sensor für die Wasseruhr

Anmerkung:

Im Falle einer langen Zuleitung >1.2m ist es Ratsam ein geschirmtes Kabel zu verwenden.

Ich empfehle diese Schaltung. SchmittTrigger: 74HC14

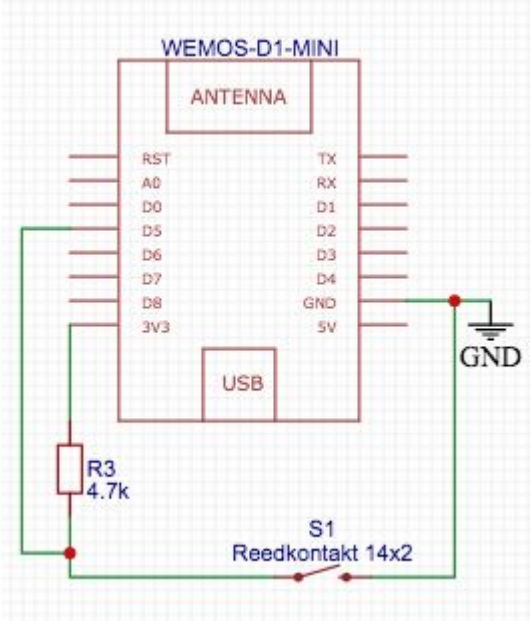




Alternative Möglichkeit bei kurzer Zuleitung und wenig Störungen.

Gaszähler

Anmerkung: Auch hier je nach Bedarf ein “Debouncing” hinzufügen.



Linksammlung

Github:
https://github.com/arendst/Tasmota
https://github.com/gemu2015/Sonoff-Tasmota

Informationen:
Forum Beitrag (Wichtig) vor allem die letzten 10 Seiten sind interessant. Die ersten Seiten sind nur interessant, falls man sich für die nicht mehr weiterentwickelte Version des Treibers interessiert. SuFu benutzen:
https://forum.creationx.de/forum/index.php?thread/1095-d0-zähler-sml-auslesen-mit-tasmota/&pageNo=48

Beschreibung von “meierchen006” aus dem oben genannten Forum, beinhaltet Erklärungen und Beispiele. Allerdings wird in dem Dokument auch der nicht mehr weiterentwickelte Weg über den Programmcode beschrieben (Offline). Achtet darauf, dass es sich um die WEBGUI/Scripter Beschreibung handelt. (Wichtig):
https://forum.creationx.de/index.php?attachment/4786-2019-10-11-0925-uhr-doku-sml-treiber-wgs-zähler-pdf/

Tasmota Wiki (wurde vor kurzem abgelöst durch ein neues Wiki allerdings gibt es im alten Wiki zum Teil noch mehr Informationen):
https://github.com/arendst/Tasmota/wiki

Neues Wiki:
https://tasmota.github.io/docs/#/Home

Treiber Wiki und Beispiele (Wichtig):
https://tasmota.github.io/docs/#/peripherals/Smart-Meter-Interface

Scripting (Wichtig):
https://tasmota.github.io/docs/#/Scripting-Language

Scripting Beispiele:
https://tasmota.github.io/docs/#/Scripting-Cookbook

Englische Beschreibung zum Thema PlatformIO:
https://tasmota.github.io/docs/#/Visual-Studio-Code

Volkszähler Projekt, Erklärung und Beispiele SML/OBIS:
https://wiki.volkszaehler.org/software/sml
https://wiki.volkszaehler.org/software/obis

Every Circuit
Ein “Freeware” Tool zum testen und erstellen von Schaltungen. Google Chrome ist eine Voraussetzung. Es ist keine gültige Emailadresse zur Anmeldung nötig.
http://everycircuit.com/

Changelog:

- 08.01.20
    - 1. Version publiziert
- 10.01.20
    - Schaltbilder hinzugefügt
    - Infos folgen hinzugefügt
- 13.01.20
    - Formatierung überarbeitet
    - Publish von eigenen Variablen via MQTT beschrieben
        - Beispiel von meierchen006 Forum Post #965
- 20.01.20
    - Modbus
        - Software
        - Hardware
- 17.03.20
    - kleinere Verbesserungen
    - update der Beschreibung für die aktuelle FW
        - damit einhergehend die Änderung der Flashgröße
    - update Flash via Kabel
    - update Liste mit den deaktivierten Funktionen und Sensoren.