Smart Home: Infodisplay selber bauen

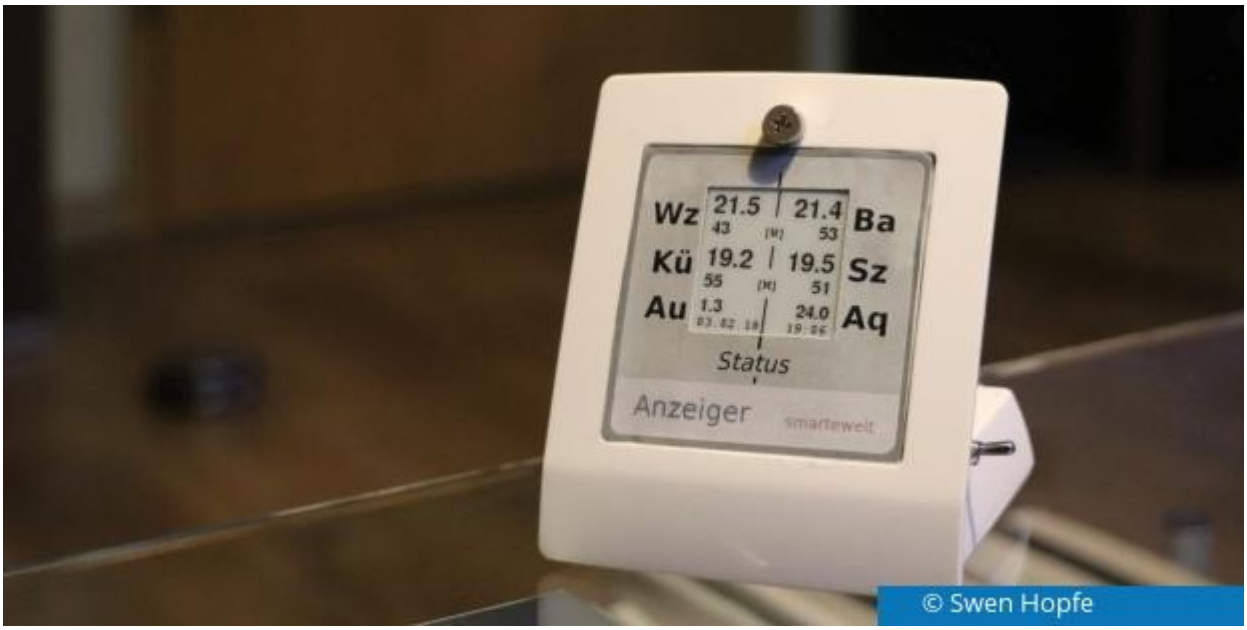
17.05.2018 | 10:46 Uhr | Swen Hopfe ()

Swen Hopfe

Swen Hopfe arbeitet bei einem mittelständischen Unternehmen aus der Chipkarten-Branche und ist Experte für Smart Cards, RFID, das IoT, Raspberry Pi und Arduino. Mehr (/autoren/swen_hopfe_1523011113498823183)

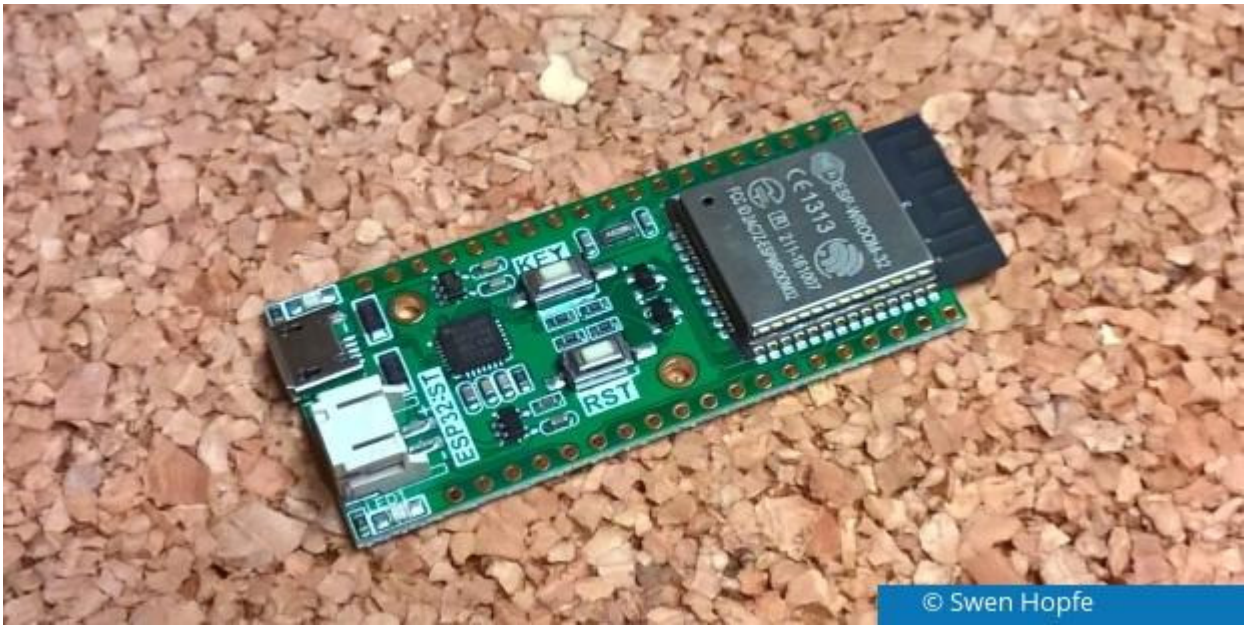
Autorenprofil schließen ()

Daten aus dem Wohnbereich (Temperatur, Luftfeuchte) sollen vom FHEM-Hausautomationsserver (Raspi) auf den ESP32, einen Ein-Chip-Rechner, übertragen und auf einem Infodisplay angezeigt werden. Die Datenübertragung erfolgt via MQTT-Protokoll.



Der ESP32 von Espressif (https://www.espressif.com/) ist eine kleine Entwickler-Platine, ähnlich dem Arduino Nano, und kompatibel, um Arduino-Entwicklungen darauf laufen zu lassen.

Er ist leistungsfähiger als sein Vorgängermodell ESP8266, und integriert WLAN und Bluetooth. Als Arduino-kompatibler Baustein lässt er sich auch genauso programmieren, ist aber auch um einiges leistungsfähiger als viele Arduino-Boards.



Außerdem haben mich seine Stromspar-Eigenschaften interessiert, und so sollte eine batterieversorgte Anzeige entstehen, die einige wichtige Werte (Temperatur, Luftfeuchtigkeit) aus dem Wohnbereich anzeigt. Dazu habe ich ein E-Ink-Display verbaut, das im Ruhezustand den Anzeigeinhalt auf dem Display beibehält und dabei keinen Strom aufnimmt.

Unsere anzuzeigenden Werte sollen von den Sensoren geliefert werden, die schon in unsere Smart-Home-Lösung eingebunden sind.

Dort läuft auf einem Raspberry Pi ein FHEM-Hausautomationsserver, mehrere Räume werden überwacht. Diese Konfiguration ergibt die Möglichkeit, mit MQTT – Message Queue Telemetrie Transport (https://de.wikipedia.org/wiki/MQTT) - zu arbeiten, um diverse Werte aus FHEM auf den ESP32 zu übertragen.

Dazu muss zuerst ein MQTT-Broker auf dem Raspberry Pi installiert werden. Unter Raspbian gibt es mosquitto:

```
$ sudo apt-get update
$ sudo apt-get install mosquitto mosquitto-clients libmodule-pluggable-perl
```

Jetzt kann man kontrollieren, ob der mosquitto-Service läuft:

```
$ sudo service --status-all 2>&1 | grep mosquitto
```

Oder ausführlichere Statusinformationen anzeigen lassen:

```
$ sudo service mosquitto status
```

Stoppen und Starten des Service geschieht folgendermaßen:

```
$ sudo service mosquitto stop
$ sudo service mosquitto start
```

Nun muss noch FHEM für MQTT vorbereitet werden. Dazu muss man zuerst die Perl MQTT-Module installieren (/user/local/man! vorher löschen):

```
$ sudo cpan install Net::MQTT:Simple
$ sudo cpan install Net::MQTT:Constants
```

Jetzt kann man ein entsprechendes Device in FHEM anlegen

```
define esp32001 MQTT_DEVICE
attr esp32001 IODev myBroker
```

```
attr esp32001 room MQTT
```

und eine Routine bauen, die Werte (hier exemplarisch vier) zum ESP (alle 1,5 min) sendet:

```
define sendmval at +*00:01:30 {\
my $rv01 = ReadingsVal('Wandthermostat_WZ_Weather','temperature','');;\
my $rv02 = ReadingsVal('Wandthermostat_Bad_Weather','measured-temp','');;\
my $rv03 = ReadingsVal('Thermostat_Kueche','measured-temp','');;\
my $rv04 = ReadingsVal('Thermostat_SZ','measured-temp','');;\
fhem("set myBroker publish fhemhome/flur/stat01 $rv01");;\
fhem("set myBroker publish fhemhome/flur/stat02 $rv02");;\
fhem("set myBroker publish fhemhome/flur/stat03 $rv03");;\
fhem("set myBroker publish fhemhome/flur/stat04 $rv04");;\
}
```

Entwickelt haben wir wieder mit der Platformio-IDE. Um Werte per MQTT zu empfangen, nutzen wir den „PubSubClient“ und die Funktion

void receivedCallback(char topic, byte* payload, unsigned int length)*

die ein Charakter-Array mit dem „Payload“, also dem gerade empfangenen Transportgut, füllt und auch noch die Länge des gefüllten Arrays zurückgibt.

In unserem Setup ist der ESP32 innerhalb 2 min Wachzeit bereit, Werte zu empfangen. Der Broker auf dem Pi übermittelt alle 1,5 min, neue Werte liegen im Wachzeitfenster damit sicher an. Danach gibt es einen Tiefschlaf (Sleep Time) für 8 min. Wachzeit und Schlafzeit ergeben somit eine Zykluszeit von 10 min.

Im Quelltext werden die 8 Minuten/480 Sekunden mit 1.000.000 multipliziert, da die Sleep Time in Mikrosekunden übergeben wird.

```
if (sleep_enabled == true) {

  long SleepTime = 480;

  esp_sleep_enable_timer_wakeup(SleepTime * 1000000);

  Serial.println("Gehe in Tiefschlaf...");

  esp_deep_sleep_start();

}
```

Vor dem Einbau haben wir alle Hardware-Komponenten inklusive LiPo-Akku auf dem Breadboard gesteckt und verkabelt und diverse Strommessungen vorgenommen.

Auf dem ESP32 läuft unsere Firmware, die insgesamt 10 Werte empfangen soll. Für den ersten Test beschränken wir uns auf nur einen Raum, auch die Testnachricht wird später nicht mehr angezeigt. [illegible]lessen wurde ein Zeitstempel eingeführt. So kann man auf dem Display gleich erkennen, wann die

Anzeige das letzte Mal aktualisiert wurde.

Daneben wird auf dem E-Ink-Display über zwei Kürzel „M“ und „W“ signalisiert, ob eine Verbindung zum MQTT-Broker und ins WLAN eingerichtet ist. Das Display ist über I2C an den ESP angebunden.

Zu Testzwecken kann man auch ein Monitoring der übertragenen Meldungen im Terminal auf dem Pi bzw. per *ssh* auf einem Netzwerkrechner vornehmen. Alle Nachrichten sieht man mittels

```
$ mosquitto_sub -v -h <IP des MQTT-Brokers/Raspberry Pi> -p 1883 -t '#'
```

Um eine Nachricht zu publizieren (hier an das sogenannte Topic "fhemhome/flur/stat", Anzeigedevice im Flur), gibt man folgendes ein:

```
$ mosquitto_pub -h <IP des MQTT-Brokers/Raspberry Pi> -t
fhemhome/flur/stat -m "Ich bin ein Test!"
```

Senden kann man auch von der FHEM-Befehlszeile oder aus der Standard-GUI, in dem man mittels Broker eine Message an das MQTT-Device sendet:

```
set myBroker publish fhemhome/flur/stat Ich bin ein Test!
```

Dabei wird der Message-String “Ich bin ein Test!” gerade nicht in Anführungszeichen gesetzt.

Untergebracht ist das Ganze in einem Fertiggehäuse aus dem Elektronikhandel. Um Platz zu sparen, haben wir nur eine Stiftleiste und in liegender Anordnung angelötet. Der Konnektor für den Akku ist nicht genutzt und hat eine Verkabelung nach innen bekommen. Der USB-Anschluss der Platine wiederum zeigt über eine Gehäuseöffnung direkt nach außen.

Die ständig grün leuchtende Power-LED unseres Dev-Boards haben wir entfernt. Dafür lassen wir die Board-LED an Pin2 bei Verbindungsaufnahme ins WLAN kurz aufleuchten. Somit kriegt man einen „Lebendig“-Status kurz nach dem Einschalten mit dem seitlichen Kippschalter. Danach kann man das Gerät abstellen.

Über dem ESP32 findet das Display Platz, über kurze Kabel mit der Steckleiste des ESP verbunden.

Die Ladeelektronik befindet sich auf der Platine selbst und ist verantwortlich dafür, dass man mittels USB-Kabel an einem 5V-Netzteil oder PC-Anschluss laden kann. Den LiPo-Akku kann man direkt anschließen.

Die Frontplatte spart in der Mitte das E-Ink-Display aus. Das wird nur für die variablen Daten benötigt und kann deshalb relativ klein ausfallen.

Alles zusammengebaut, ist unser „Anzeiger“ für das Smart Home nun fertig. Tests über die letzten Wochen haben ergeben, dass nach einigen Tagen nachgeladen werden muss. Da kann man die Parameter für Aktualisierung und Bereitschaft ändern, je nachdem, wie aktuell man seine Anzeige braucht, oder alternativ den Betrieb über USB-Kabel und kleines Netzteil machen.

Nicht zuletzt hat bei diesem Projekt auch die Optik für uns eine Rolle gespielt, damit das Ganze in die Wohnung passt und gut vorzeigbar ist. Da heißt es, sich bei der Konstruktion entsprechend Mühe zu geben.

Alles in allem bin ich mit dem ESP32 als Baustein für solche Aufgaben aber zufrieden und deshalb ist geplant, ihn auch noch in weiteren Vorhaben einzusetzen.

Smart Home - darum geht es.

Smart Home umschreibt einen der vermutlich am stärksten wachsenden Trends der Zukunft: Intelligente Geräte und Haus-Steuerungssysteme, die Ihnen Ihr Leben erleichtern sollen. Denkbar und vielfach bereits erhältlich sind dabei Lösungen wie Einbruchschutz oder Sets zum Stromsparen. Oder Intelligente Systeme zur Hausbelüftung oder Heizungssteuerung. Smart TVs bringen das Internet ins Wohnzimmer und Waschmaschinen lassen sich per App steuern. Über intelligente Stromzähler haben Sie mit einem Blick aufs Smartphone den aktuellen Stromverbrauch im Griff. Sie sehen, die möglichen Anwendungen sind vielfältig. In diesem Themenbereich widmen wir uns dem Thema Smart Home, erklären, wie die Technik funktioniert und wie intelligent vorhandene Lösungen am Markt wirklich sind.