CLUG-Stammtisch

ESP32 - Wie kann ich die WLAN-Briefmarke programmieren?

Andreas Heik

TU Chemnitz, Universitätsrechenzentrum

24. Mai 2019

TECHNISCHE UNIVERSITÄT
CHEMNITZ

Motivation

Suche nach einem Mikrocontrollerprojekt mit WLAN.

→ ESP8266-Modul und ESP32-Modul der Firma Espressif[1]

- Anbindung über verschiedene Schnittstellen möglich, *aber*
- *Module enthalten eigenen Mikrocontroller mit ausreichend Platz für eigenen Projekte, GPIO Pins, . . .*
- *frei verfügbares ESP-IDF (Espressif IoT Development Framework)*
- *perfekte Hardware für IoT-Projekte*

→ CLT-Junior Workshop mit Fokus auf Hardware und Programmierung

[1] `https://www.espressif.com/`

Die Hardware

verschiedene Ausführungen:

- Module
  - Einsatz in spezifischen Projekten
  - für Programmierung und Betrieb ist Zusatzbeschaltung erforderlich
- Entwicklerboards enthalten meist
  - Stromversorgung via USB
  - USB-Serial Wandler
    (mit Reset-Schaltung zum Programmieren)
  - Pinheader für Steckbrett
  - Anscluß für LiPo-Akku und Laderegler
  - …

Die Hardware

	ESP8266	ESP32
MCU	Xtensa Single-core 32-bit L106	Xtensa Dual-Core 32-bit LX6
Typical Frequency	80 MHz	160 MHz
SRAM	(ja)	520 KiB
SPI-Flash[2]	2 MB	4 MB
Co-Prozessor	-	ULP
802.11 b/g/n Wi-Fi	HT20 (20MHz Kanalbreite, bis 72,2 Mbit/s)	HT40 (40 MHz Kanalbreite, bis 150 Mbit/s)
Bluetooth	-	Bluetooth 4.2 BR/EDR und BLE
GPIO	16	36
PWM (Software)	8 channels	16 channels
SPI/I2C/I2S/UART	2/1/2/2	4/2/2/3
ADC	10-bit	12-bit
CAN	-	ja
Ethernet MAC Interface	-	ja
Touch Sensor	-	ja
Temperature Sensor	-	ja
Hall effect sensor	-	ja
Preis	3,58 EUR	6,27 EUR

[2] je nach Module

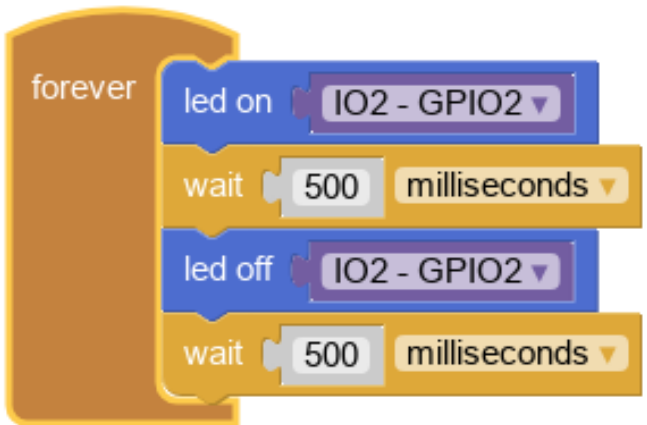
Let's get started

- Zugriff auf Serial-Port (USB)
  - idealerweise über eine `udev`-Regel
- whitecat-console (`wcc`)
  - Flash/Erase der Firmware auf dem Board
  - Transfer von Dateien
- Lua-RTOS Firmware (wird mittels `wcc` installiert)
- Whitecat IDE
  - Programmieroberfläche auf Basis von Blöcken (vgl. Scratch)
  - im Browser: `https://ide.whitecatboard.org/`
  - als Webkit-App (siehe Downloads vom CLT-Workshop[3])
- whitecat-create-agent
  - Verbindung zwischen serieller Schnittstelle und Whitecat IDE

[3] `https://tuc.cloud/index.php/s/mtw8brmSNJBqzGr`

Hello World

- blaue LED auf dem Board ist an `GPIO2` angeschlossen



- die Whitecat IDE erzeugt aus den Programmblöcken Lua-Code (Script)
- lädt diesen in das Filesystem des ESP32
- und führt das Script aus (`dofile("_run.lua")`)

Die Lua-RTOS Firmware

- Real-Time Betriebssystem mit minimalen Ressourcenbedarf
- Layerkonzept
  - Lua Interpreter mit Modulen für Hardwarezugriff und Dienste
  - Real-Time Microkernel
  - Hardware Abstraction Layer
- Filesystem (auf Flash-Speicher)
- Shell (interaktiver Lua Interpreter)
- Editor
- Startskripte und Konfigurationsfile
- Firmwarevariante mit OTA Update-Funktion
  (Update über Netzwerk aus laufender Firmware heraus)
- Firmeware selbst bauen (Hinweise auf github-Projektseite)
  - ESP32-Tools (xtensa-esp32-gcc)
  - ESP-IDF (Development Framework)
  - Lua RTOS

Die Shell und das Filesystem

- interaktiver Zugang
  - `minicom -D /dev/ttyUSB0 -b 115200 -o`
    (Hardware-Flußkontrolle deaktivieren)
- Kommandos in der Shell
  - `ls`, `cd`, `cat`, `cp`, `mv`, `rm`, `mkdir`, `pwd`
  - `edit <file>`
    Hilfe mittels `STRG + Y`
  - `netstat`, `ping`
- Dateitransfer mittels `wcc`
  (`minicom` vorher beenden!)
  - `wcc -p /dev/ttyUSB0 -ls /`
  - `wcc -p /dev/ttyUSB0 -down /config.lua .`
- Startscripte
  (Ausführung verhindern mittels `STRG + D` beim booten)
  - `config.lua` - Konfigurationsvariablen
  - `system.lua` - Netzwerkkonfig, Dienstestart, ...
  - `autorun.lua` - Start der Anwendung

Whitecat IDE

- gut strukturierte, aufgeräumte Oberfläche
- zahlreichen grafische Bausteine (Blöcke)
- intuitive Verwendung der Bausteine
- Umschalten zur Ansicht des Lua Codes
- Entwicklermodus (Exception handling, Fehlerausgabe)
- automatischer Upload und Ausführen des Lua Codes
- integrierte Konsole (Ausgabe von `print`)
- Speichern des Blockprogrammes (xml) auch auf dem Filesystem des ESP32

Tipp: Bei Kommunikationsproblemen mit dem Board den wccagent in einem Terminal starten `wccagent -ui -lc`

Sensoren

- ESP32 enthält interessante Hardwaremodule:
  - ADC, GPIO, PWM, I2C, SPI, TMR, ...
- Lua-RTOS Firmware unterstützt die Kommunikation mit vielen **Sensoren**:
  - DS1820[4], DHT11, 10k Thermistor, BME280, ...
- und **Modulen**:
  - Drehimpulsgeber, graf. Displays, Neopixel (WS2812), Servos, ...
- Details liefert die Funktion `sensor.list()`
- Sensor-Konzept:

```
s = sensor.attach("BME280", i2c.I2C0)
s:read(pressure)
s:detach()
```

[3] funktioniert nicht, Auflösung falsch → `conversion time` zu kurz

Lua Tipps

- Delay und Timer
  - `tmr.delay()` verursacht hohe Last da, das Warten durch zählen leerer Prozessorzyklen erreicht wird
  - eine bessere Alternative bietet `tmr.attach()` auf Basis von Hard- und Softwaretimern
  - Auslastung kann mit `thread.list()` angezeigt werden
- Sleep
  - verschiedene Sleep-Modi um Stromaufnahme zu reduzieren
  - *deep sleep* normalerweise im Mikroamperbereich, nicht mit dem Entwicklerboard zu erreichen (rote LED, Spannungsregler, USB-Serial-Wandler)
  - Aufwachen nach `os.sleep(10)` ist mit Neustart vergleichbar
- NTP
  - Firmware enthält NTP-Client, aktivierbar beim Start mittels `config.sntp=true`
  - Uhrzeit anzeigen: `os.date()`, Laufzeit: `os.uptime()`

Web- und SSH-Server

- Webserver
  - Dokumentroot unter `/wwww`
  - Lua-Scripte als CGI möglich
    (nach Scriptupdates `file.luap` löschen)
  - Zugriff auf globale Lua-Variablen möglich (Datenaustausch zwischen Anwendungsthread und Web-Script)
  - Webserver mit `net.service.http.start()` starten
- SSH-Server
  - in der aktuellen Firmware standardmäßig deaktiviert, Firmware neu erstellen
  - root-Passwort mit `os.passwd()` festlegen
  - SSH-Server mit `net.service.ssh.start()` starten

weitere Programmiermöglichkeiten

- Grenzen der Lua-RTOS-Firmware
  - Nutzung spezieller Hardwarefunktionen
    z.B. Touchsensoren, Bluetooth, . . .
  - Implementation komplexer Funktionalität
- Firmware mit Interpreter für höhere Programmiersprachen
  - NodeMCU, (Lua basierte Firmware)
  - MicroPython
  - Espruino (JavaScript), webbasierter IDE und graf. Editor
- native Firmwareentwicklung
  - auf Basis des Espressif IoT Development Framework (ESP-IDF)
  - komfortabler mit **Arduino IDE**
  - PlatformIO, . . .

weitere Programmiermöglichkeiten

- Grenzen der Lua-RTOS-Firmware
  - Nutzung spezieller Hardwarefunktionen
    z.B. Touchsensoren, Bluetooth, ...
  - Implementation komplexer Funktionalität
- Firmware mit Interpreter für höhere Programmiersprachen
  - NodeMCU, (Lua basierte Firmware)
  - MicroPython
  - Espruino (JavaScript), webbasierter IDE und graf. Editor
- native Firmwareentwicklung
  - auf Basis des Espressif IoT Development Framework (ESP-IDF)
  - komfortabler mit **Arduino IDE**
  - PlatformIO, ...

Programmieren mit der Arduino IDE

- Plattformpaket: **Arduino core for ESP32 WiFi chip**[5]
  - Boardbeschreibung (Parameter, Pinlayout, . . .)
  - Teile des SDK
  - Downloadtool für Compilersuite
  - Bibliotheken und Beispiele
  - Installationsanleitung auf der GitHub-Seite
- große Anzahl Arduino-Bibliotheken verfügbar
- Arduino und ESP32:
  - passendes Board auswählen
  - Port für Upload auswählen
  - Beispiel öffnen, anpassen, kompilieren, hochladen
  - Seriellen Monitor öffnen (`Serial.print()`-Ausgaben)

Hinweis: Upload des Binärcodes überschreibt vorhandene Firmware

[5] `https://github.com/espressif/arduino-esp32`

Beispiel: Touch Sensor

- elektrischer und mechanischer Aufbau eine kapazitiven Touch-Sensors
  `https://github.com/espressif/esp-iot-solution/blob/master/documents/touch_pad_solution/touch_sensor_design_en.md`
- Änderung der Kapazität am Touch-GPIO erfasst der ADC
- Auswertung durch Vergleich mit Schwellwert

```
int pinTouch = 4;
int pinLED = 2;
int CapacityMax = 20;

void loop() {
    int val = 0;
    for(int i=0; i< 100; i++)  {
      val += touchRead(pinTouch);
    }
    val = val / 100;
    if (val < CapacityMax) {
      digitalWrite(pinLED, !digitalRead(pinLED));
    }
}
```

Beispiel: Bluetooth LE

- Bluetooth Low Energy (BLE) auch als Bluetooth Smart bekannt
- Generic Access Profile (GAP) - Advertising und Verbindungskontrolle
  - periphere Geräte sendet regelmäßig Advertising Data
  - zentrales Gerät kann einen Scan Response Request senden
- Generic Attribute Profile (GATT)
  - Service(s)
    - logische Gruppierung von Daten
    - Unterscheidung durch UUID[6]
    - enthält Characteristic(s)
  - Characteristic(s)
    - kleinste Einheit enthält Daten (z.B. Temperaturwert)
    - Unterscheidung durch UUID[7]
    - Lesen und Schreiben von Daten (read/write)
    - Notify bei Wertänderungen (vom Server initiiert)

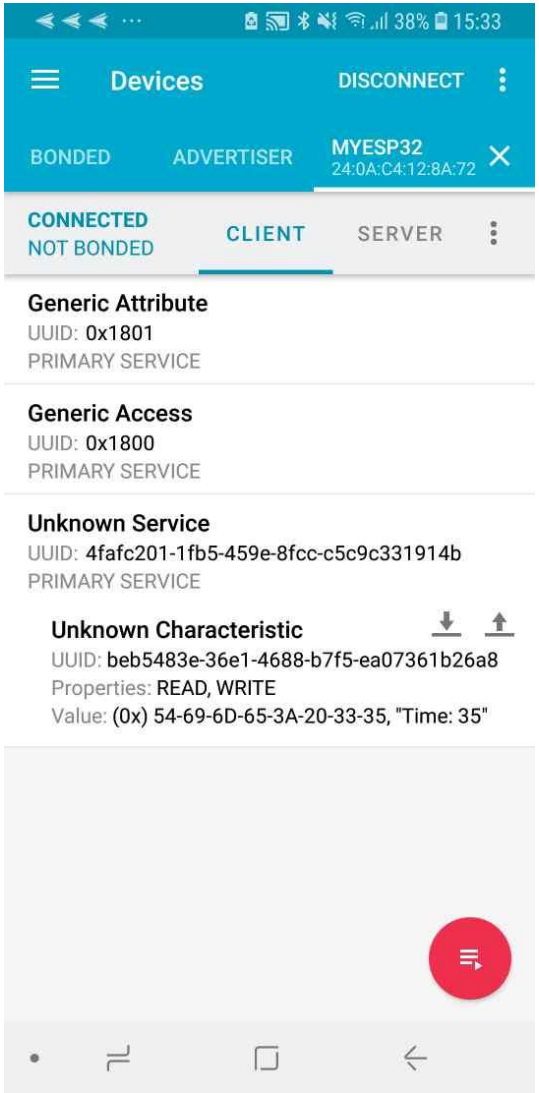
[6] https://www.bluetooth.com/specifications/gatt/services/
[7] https://www.bluetooth.com/specifications/gatt/characteristics/

Bluetooth Kommunikationspartner

- Smartphone, Tablet mit App (z.B. nRF Connect for Mobile)
- Laptop mit Bluetooth-Modul und `bluez`-Tools

```
hcitool lescan
 LE Scan ...
 24:0A:C4:12:8A:72 MyESP32

gatttool -b 24:0A:C4:12:8A:72 -I
 [24:0A:C4:12:8A:72][LE]> connect
 Attempting to connect to 24:0A:C4:12:8A:72
 Connection successful
 [24:0A:C4:12:8A:72][LE]> char-desc
 handle: 0x002a, uuid: beb5483e-36e1-4688-b7f5-
     ea07361b26a8
 [24:0A:C4:12:8A:72][LE]> char-read-hnd 0x002a
 Characteristic value/descriptor: 54 69 6d 65 3a
     20 35 33 35
```



Einsatz am Campus der TUC

- ESP32 in der Lehre, Professur Experimentelle Sensorik
  - Unterstützung durch Bereitstellung von Entwicklungsumgebungen
  - Beispielimplementation: WPA2-Enterprise (eduroam) für ESP32
  - neu: spezielles WPA2-Funknetz
  - Beratung und Unterstützung bei Betrieb: MQTT, Node-RED
- Meßpunkte im Datacenter des URZ
  - Sensoren: BME280, DS1820
  - auf Basis einer angepassten Lua-RTOS-Firmware
  - OTA-Updatefunktion
  - Datenübertragen mittel MQTT, Archivierung auf Basis RRD
- CLT-Junior Workshop
  - Fokus auf IoT-Anwendungen
  - Programmierung mittel Whitecat-IDE
  - Einsatz verschiedener Sensoren

Quellen und Links

- CLT-Junior, IoT-Workshop, Downloads
  `https://tuc.cloud/index.php/s/mtw8brmSNJBqzGr`
- Whitecat IDE und Lua-RTOS Firmware
  `https://github.com/whitecatboard/`
- Lua-RTOS Dokumentation
  `https://github.com/whitecatboard/Lua-RTOS-ESP32/wiki`
- Arduino core for ESP32 WiFi chip
  `https://github.com/espressif/arduino-esp32`
- NodeMCU, Lua basierte Firmware
  `https://nodemcu.readthedocs.io/en/dev-esp32/`
- MicroPython
  `http://micropython.org/download#esp32`
- Espruino (JavaScript), webbasierter IDE und graf. Editor
  `https://www.espruino.com/ESP32`
- TUNIOT FOR ESP32, blockbasierter Editor, C++
  `http://easycoding.tn/index.php/esp32/`

ESP32 Programmierung

Vielen Dank für Ihre Aufmerksamkeit!