Recap from Wednesday

- The MQTT protocol was discussed:
  - Publisher Subscriber model with a Broker as central entity
  - Messages are called “topics” which have a name and an arbitrary payload
  - Topic names are organized in a tree hierarchy like directories. (They are also written with the same syntax as Linux directories)
  - Messages can have three different QOS
  - Clients can connect or disconnect from the broker at any time.
  - There are some interesting options during connect or publishing:
    - last Will, cleanSession, retain

- In the exercises we extended the Sensor readout program to publish on a MQTT network
  - All groups succeeded within the time of the exercise because:

    A) You are very good of course :-)

    B) MQTT is really relatively simple to handle and very user-friendly

Microcontrollers

Part VII

Going further with microcontrollers

… some appetizers...

(Christoph.Schwick@cern.ch)

Going further with microcontrollers

- We programmed the microcontroller in Python
  - This is perfectly fine when only moderate performance is required
  - IoT applications are a perfect use-case for this kind of programming
  - However, you will never be able to unleash the full power of your microcontroller with micropython…
    - it is just way too slow,
    - it has no real-time capabilities (or very limited ones only)
      - especially there is no way to handle multiple threads with clear scheduling priorities
    - it has not way to use both cores of the microcontroller
  - This is not a criticism of micropython but just states that micropython is not the correct solution for all possible problems
  - If you CAN solve a problem in micropython you are usually MUCH quicker to implement the solution in micropython than doing the same thing is ‘C’

Going further

- If you want to see what is possible to do with the microcontroller you will need to master two topics
- Development system for “C” code:
  - You need some experience in C programming (or need to be willing to learn C)
  - Develop code of the esp32 with a “C” (or “C++”) development system.
  - For the ESP32 this is provided by Expressif (the company building the ESP32 chip)
  - It is well documented.
  - It is relatively straight forward to use. (You will have to learn about the cmake build system)
  - There are tons of running examples which come with the development system.
  - No fancy GUIs are used. You can use your favorite editors if you want.
  - The Arduino system uses the Expressif development system under the hood.

Going Further

- You need to learn FreeRTOS
  - RTOS stands for Real Time Operating System
  - FreeRTOS is a very mature RTOS used widely in industry (or research)
    - It is a good investment to learn this system.

- Why do we need an RTOS on a microcontroller
  - Very often a microcontroller application can be split into different “tasks” which have to be done.
  - Each of the task can be programmed separately (like a subroutine)
    - Of course you will often need some data exchange and communication between the tasks
  - Usually tasks have different priorities and “real time requirements”
    - If a user clicks on a button on a screen to open a new menu on a screen the response time can be as slow as the typical reaction time of human beings (some 100 ms)
    - If a sensor detects some mal-functioning in a dangerous machinery the reaction time must be guaranteed to better than some well defined maximum.
  - An RTOS does exactly this: it provides tools to prioritize many different tasks of the same application.
  - Also tools for communication between the tasks are provided.

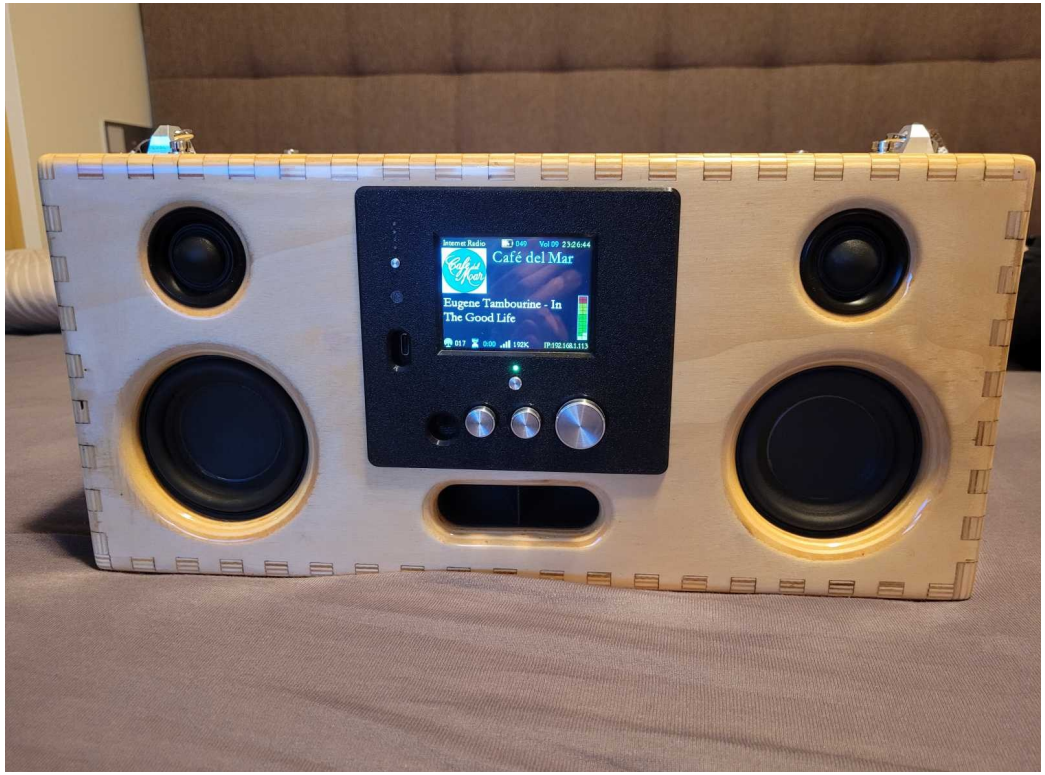
Going Further

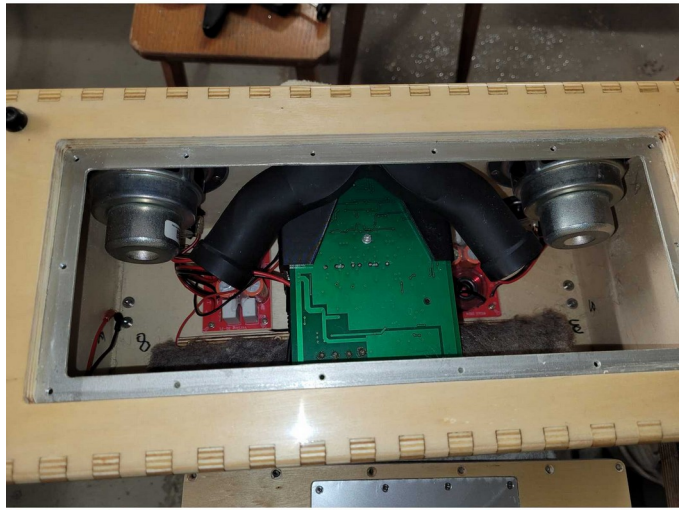
- The web site provides a section with a discussion of task scheduling in FreeRTOS.
  - If you are interested to go further read that section.
  - If afterwards you are still interested you can dive into the FreeRTOS tutorial which is an excellent documentation of the FreeRTOS system.

- Links to the FreeRTOS documentation are contained on the web page

- Btw, FreeRTOS you can also run on your computer (simulators for Windows or Linux) so that you can do tests.
  - It is “free”
  - It is “small” (the kernel is only ~60kB large !!! It is made for microcontrollers!)

Examples



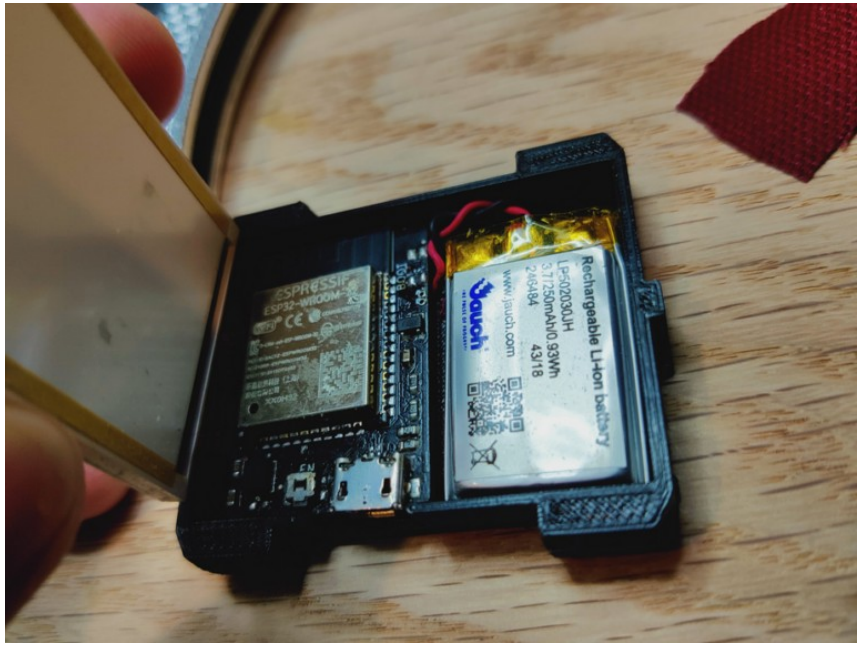


https://github.com/schreibfaul1/ESP32-MiniWebRadio

Examples

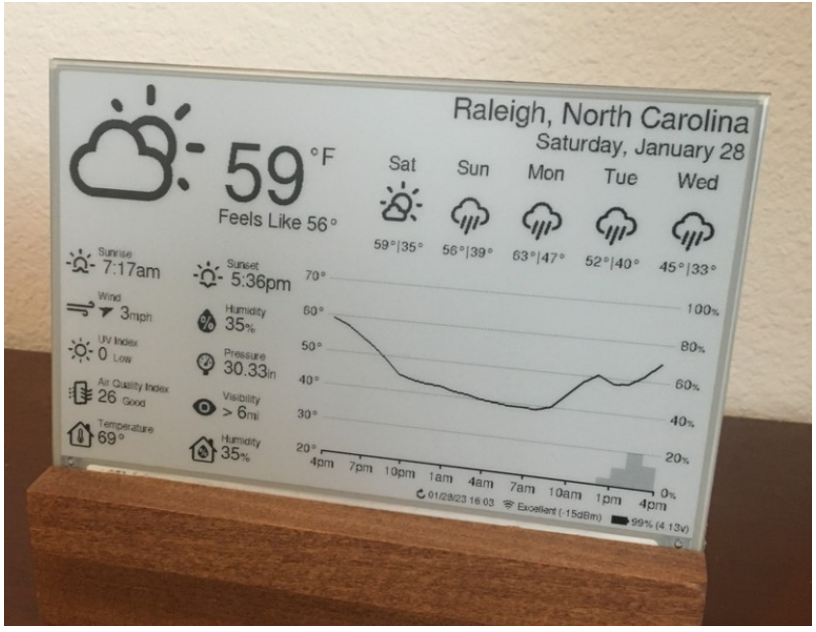
https://picockpit.com/raspberry-pi/paragon-project-a-portable-music-player/

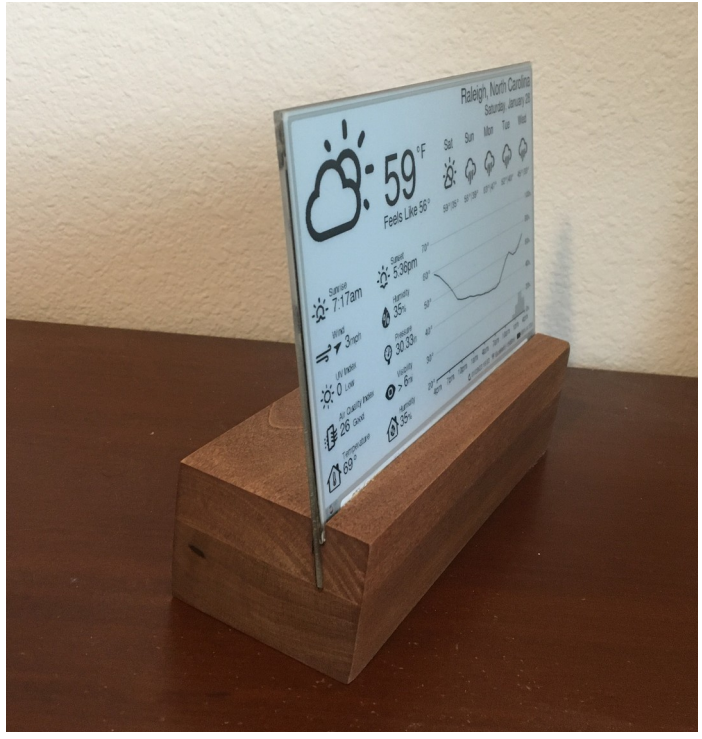
Examples



https://github.com/Bellafaire/ESP32-Smart-Watch

Examples





https://github.com/lmarzen/esp32-weather-epd


elexperiment.nl

In case you want to do a project with microcontrollers

The GOLDEN choice !!!

- Invent your own project
  - If you have an idea of something which interests you come to us and discuss. This is the best way to start !!!
    - You will learn most like this
    - You will have most fun like this
    - Of course you can always ask for advice if you get stuck (this is muuuuch better than copying something from the web!)
  - It seems that there is the possibility to purchase (...in some limits, of course…) also new hardware for the project in mind:
    - You need an ADC or DAQ with different specs than the internal ADCs or DACs?
    - A Sound input or output (I2S DAQ or I2S microphone)?
    - A Stepper motor or Servo (with driver electronics)?
    - A TFT (touch?) display
  - Tell us what you need and I will see with Prof Andrea Triossi if the Uni can purchase the required item.
  - If you want to purchase stuff privately since you want to keep it I am happy to share my personal experience with you. (I do not have a very wide overview and it will not reduce any kind of risk… but I can tell you about the problems I had so far…

Hardware you can have for your project

- Microcontrollers
  - If you need more than one this should be ok.
    - If you want to build a 1000 Core cluster we run into problems…
- Ditital microphone (I2S interface)
- High quality stereo audio output (I2S → headphone)
- Servos
- Step-motors
- TFT displays
- GPS Unit
- If you need something else it probably can be purchased (...in some limits...):
  - Accelerometer / gyroscope are very nice to play with...

In case you want some appetizers:

- An ALARM clock
  - Analogue display
  - Time setting via NTP (from NTP server over WIFI)
  - Show date
  - Touch pins to change brightness / disp on / off or more?
  - May be settings like alarm time/melodie/… over MQTT???
- Voltmeter or Oscilloscope (or could use digital microphone as input)
  - How fast can you go?
  - Do you need “C” here?
- Histogramming of sensor history
  - Touch pins to change which sensor to display
  - Auto-scale? Change scale manually?
- Games (Using a TFT display)
  - Snake, Tetris, other vintage games, or whatever comes to your mind
  - Requires some input buttons (or touch buttons) which you can get

Appetizers for projects

- Calculus trainer
  - Make exercises with proposed solution → answer is “right” or “wrong” (needs two touch pins)
  - Measure performance (answers per time: percentage correct/wrong… invent metrics)
  - Change type and difficulty of calculations
- Chat Client with MQTT
  - use last will (when members leave)
  - This requires a keyboard for entering text (can use computer for the project)
  - Invent some extra features, otherwise this project is too easy….
- A digital photo frame
  - Requires a TFT display (with slot for SD card)
  - Possibly decodes jpg photos (there are existing libraries for this, this lib you do not need to write yourself)
  - What about scaling pictures to the right format? (...not trivial...)
- Speed measurement device?
  - Need some kind of sensor or a pair of sensors.

Appetizers for projects

- Music generator
  - Play music
  - Generate nice sounds
  - Interpret (simple) midi files
  - Internet radio (C/C++)
  - Synthesizer (C/C++)
- Make a walky talky (C/C++ I guess)
  - I2S microphone and speaker
  - Communicate over WIFI: TCP/IP or UDP?
  - what is the latency?
- Remote controlled car/boat/plane ?
  - Needs servos and motors and mechanics…
  - May be somebody has an old car or boat at home which can be equipped with a ESP32 remote control?