BASIC MODUS OPERANDI

Emilian Manole

On

↓

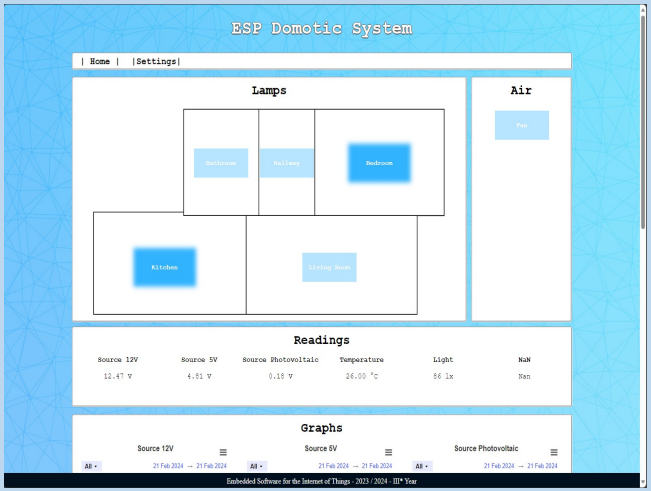
Connect to WiFi

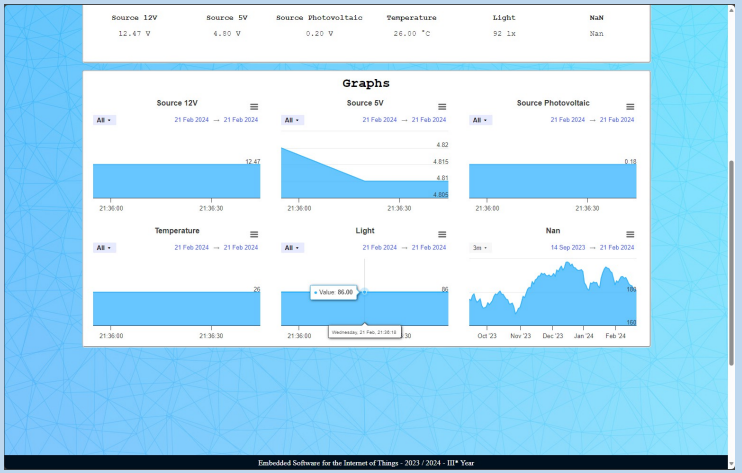
↓

Init Uart for MSP

↓

NTPClient start





↓

Read values/sensors

↓ to database .CSV

date/time read + append data → write on last reading

waiting for http get to switch relays

get data from MSP

MSP432 timer and interrupts

Leonardo Rigotti

- Continous mode timer
- ACLK at 32kHz, divided by 14 so ~2.2kHz
- Overflows when hits 0xFFFF, around every 28s and sends an interrupt

```
void TA0_N_IRQHandler(void){
    Timer_A_clearInterruptFlag(TIMER_A0_BASE);
    sendTemperature();
    sendLight();
}
```

in main.c

- MCU is in LPM0 (ACLK and SMCLK active)
- ISR sends data via UART to other device
- return to LPM0:
    - Interrupt_enableSleepOnIsrExit();

- interrupt on EUSCIA2 (pin 3.2, 3.3)
- executes when something is received via the UART receiver
- stores the data in a global variable and clear flags

MSP432-ESP32 UART communication

Leonardo Rigotti

- EUSCI UART configuration
- 24MHz (SMCLK source), 115200 baudrate, no parity bit, one stop bit, 8 bit long message
- Using eUSCI 2, i.e. pin 3.2 and 3.3

```
void sendInBlock(int value){

    if(value<255){
        sendData(value);
    } else {
        while(value>=255){
            sendData(255);
            value-=255;
        }
    }
    if(value>0){
        sendData(value);
    }
}
```

in uart.c

PROBLEM: only 1 byte via UART

SOLUTION: send data that is >= 255 (max value with a byte) in smaller blocks ...

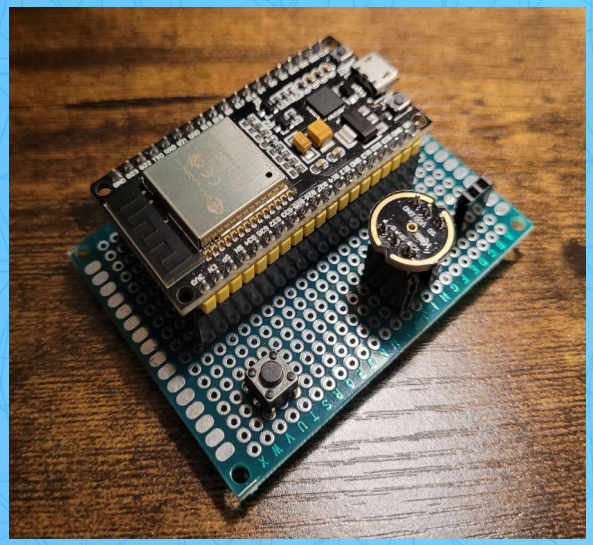
```
void readTemperature(){
  while(1){
    int data = getData();
    if(data==0) break;
    temperature+=data;
  }

  temperature=(float) (temperature)/10.0;
  updateFanStatus(temperature);
  Serial.print(temperature);

  temperature = 0;
}
```

in esp_uart.cpp

… that get sum up on the ESP side

Voice Command

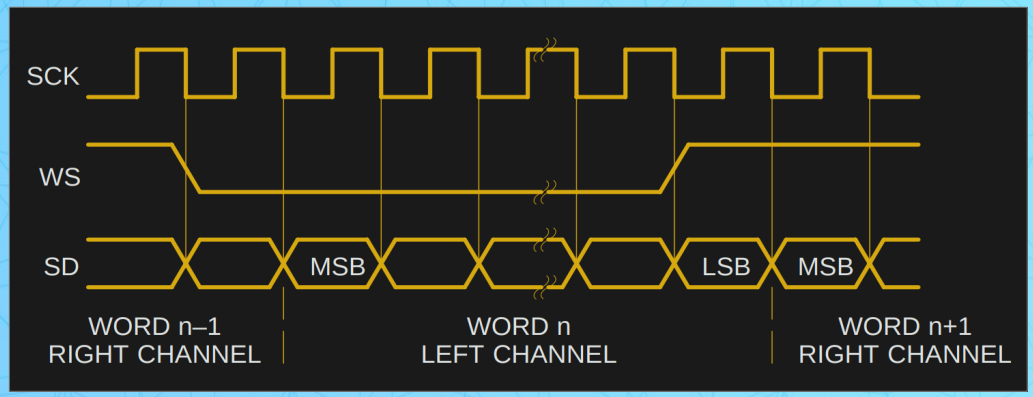
Riccardo Randon, Elia Avanzolini



We utilized an ESP32 board to capture audio signals from an INMP441 microphone using the I2S protocol.

Due to the high data rate, we employ DMA to store the incoming data in memory.

Afterwards, we utilize flash memory to store the file and produce the necessary WAV audio file format as per Google APIs.



Speech to Text API

Riccardo Randon, Elia Avanzolini

Once we obtain the WAV file, ESP32 sends it to the PC's Python server.
Here, with the assistance of the library provided by Google, we send a request to the Speech-to-Text APIs.

The response comes as a JSON file, which we extract the needed information from, then send an integer containing home automation configurations back to ESP32.