Design Description

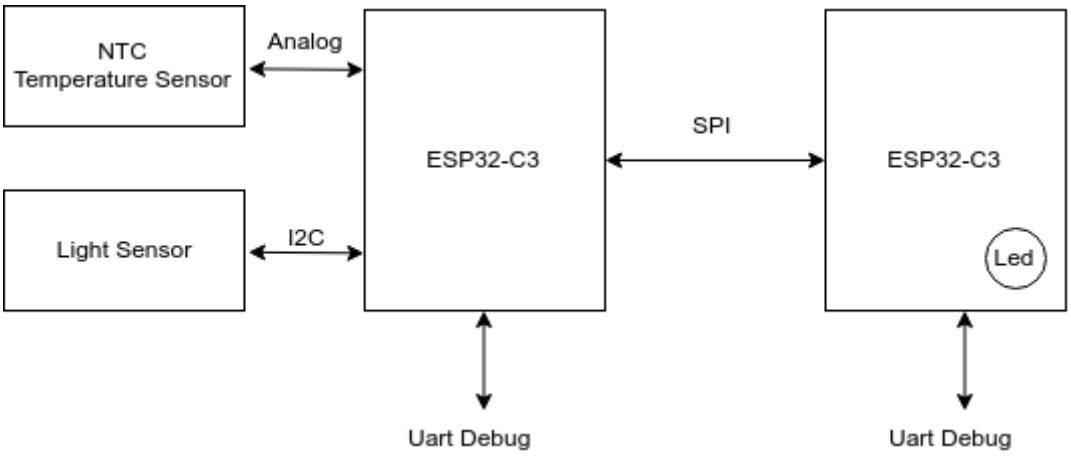
Title: UniBz Laboratory
Name: Jana Karas

Content

General System Description

Description of the scope of the document.

Block Diagram



General Structure

The system is composed of two connected microcontrollers. Both microcontrollers are of type “ESP32-C3”, a single-core microcontroller System-on-a-Chip (SoC). To uniquely define the parts of the design, we will call one SoC “ESP32-A” and the other one “ESP32-B”. ESP32-A is connected to a temperature sensor and a light sensor, ESP32-B is connected to an external LED. Both microcontrollers are additionally connected to an external PC for debugging purposes.

The system’s function is to control the LED depending on the temperature and the light intensity. In Detail:

- The LED current is set proportional to the temperature
- If the light is over 500 lx or under 50 lx, the LED current is zero.

Communication Protocols

Connection ESP32-A to Temperature Sensor

ESP32-A is connected to a negative-temperature-coefficient (NTC) thermistor. With NTC thermistors, resistance decreases as the temperature rises. We use this feature for sensing the ambient temperature.

The microcontroller is connected to the NTC at GPIO 3 via an Analog-to-Digital Converter (ADC). Additionally, the circuit is connected to Voltage and Ground. The reference ratio for the NTC is 100 kiloohm at 25 ° C, as can be extracted from the datasheet of the NTC.

Additionally, a second resistance of 82 kiloohm is connected in series, which has to be considered when calculating the temperature from the measured voltage.

Connection ESP32-A to Light Sensor

Due to issues in procurement, the system does not have a light sensor.

Connection ESP32-B to LED

ESP32-B is connected to an LED at GPIO 9. Additionally, the circuit is connected to the 3,3 V Voltage and Ground. For regulating the light intensity, a pulse-width-modulator (pwm) is used. The pwm controls how long the high and low phases of the data line are and like that regulates the light intensity.

Connection ESP32-A to ESP32-B

ESP32-A and ESP32-B communicate through an SPI bus.

We will use ESP32-A as a master and ESP32-B as a slave. Like that, ESP32-A can constantly send the measured temperature data without having to make a request and ESP32-B can read the data at the same speed.

The pins that will be used at ESP32-A are the following:

Chip Select (CS)	GPIO 10
Serial Clock (SCLK)	GPIO 6
Master Input Slave Output (MISO)	GPIO 2
Master Output Slave Input (MOSI)	GPIO 7
Additional Line for Handshaking	GPIO 8

The pins that will be used at ESP32-B are the same:

Chip Select (CS)	GPIO 10
Serial Clock (SCLK)	GPIO 6
Master Input Slave Output (MISO)	GPIO 2
Master Output Slave Input (MOSI)	GPIO 7
Additional Line for Handshaking	GPIO 8

Connection ESP32-A to PC

For debugging purposes, ESP32-A is connected to an external PC through the UART to USB interface. No configuration is needed.

Connection ESP32-B to PC

For debugging purposes, ESP32-A is connected to an external PC through the UART to USB interface. No configuration is needed.

Power Management

Both boards are powered via the USB-UART interface. For the circuits, the power outlet pins of 5,5 Volts are used.

Software design description

ESP32-A

Block 1 – Reading Temperature via ADC

The code first configures the ADC: We are using ADC1 with CHANNEL 3 (the channel is equivalent to the GPIO pin, see above).

For calculating the temperature, we will use the raw voltage read by the ADC.

For calculating the resistance of the NTC we will use the following formula:

$$\mathrm{res}_{\mathrm{NTC}} = \frac{82\,000\ \Omega * (3.3\ \mathrm{V} - \mathrm{volt}_{\mathrm{raw}})}{\mathrm{volt}_{\mathrm{raw}}}$$

This is due to the parallel connection of the ADC and the NTC.

For calculating the temperature from the resistance of the NTC we need a few additional values:

- the reference ratio of 25 ° C (= 298 Kelvin) and 100 kiloohm
- a constant “B” with the value 4075 Kelvin (see datasheet).

Using these values, the temperature is calculated as follows:

$$\mathrm{temp}_{\mathrm{Kelvin}} = \frac{1}{\frac{1}{298\ \mathrm{K}} - \frac{\ln(100\,000\ \Omega) - \ln(\mathrm{res}_{\mathrm{ntc}})}{4075\ \mathrm{K}}}$$

For converting to Celsius:

$$\mathrm{temp}_{\mathrm{Celsius}} = \mathrm{temp}_{\mathrm{Kelvin}} - 273.15$$

Block 2 – SPI Master

The second component of firmware-A sends the temperature to microcontroller B. As described above, ESP32-A is the master in this case.

The code configures the GPIOs according to the table above. When using SPI with two processors, it is common to use an additional line for handshaking. In this case, we use GPIO 8. This line is set to “High” in idle time. When the slave is ready, it pulls the handshake line to “low”. After finishing the transmission, the slave pulls the line to “high” again.

The code block consists of two functions:

1) Initialize the SPI Master
2) A function to

- send the temperature through the SPI using the function “spi_device_transmit”
- receive the frequency of the LED to allow for a status check.

The temperature is converted to a string and sent exclusively. It is being read and sent once a second.

The advantage of using a string (or actually a character array) over an int or float value, is that it is more easily extendable. For instance, we could send some complementary information. In this case, we do not have strict constraints in the data transmission, so we can allow for the additional overhead.

ESP32-B

Block 1 – SPI Slave

The first component of firmware-B receives the temperature from microcontroller A. As described above, ESP32-B is the slave.

The code configures the GPIOs according to the table above, additionally the GPIO for the handshake is 8.

The code block consists of two functions:

1) Initialize the SPI slave
2) A function to

    - read data from the SPI “recvbuf” and
    - send the frequency of the LED to allow for checking the status.

The temperature is converted back into a float value and passed on to the next function.

Block 2 – Control LED

This code block calculates the desired duty value for the PWM from the temperature. The duty value determines the ratio between the highs and lows in the data line: A duty value of more than 8190 means that the LED is switched off, whereas a duty value of 0 leads to the highest intensity of the LED.

The temperature range that our system considers is from 0 ° Celsius to 40 ° Celsius. We want to map this temperature value to a blinking frequency. The two phases of the LED blinking are: very low light intensity (duty of 8000) and very high light intensity (duty value of 0). The temperature is being mapped to a range of the blinking duration of 2000 ms at 0 °C to 200 ms at 40 °C.

Therefore, the following simple mathematical function is used for mapping the temperature to the blinking duration:

$$\text{blinking\_duration} = 2000 - \text{temp}_{\text{Celsius}} * 18 * 2.5$$

The blinking is created using the “vTaskDelay” function.

The SPI receiver and the blinking are separated into two threads so that they do not block each other.