Task 4

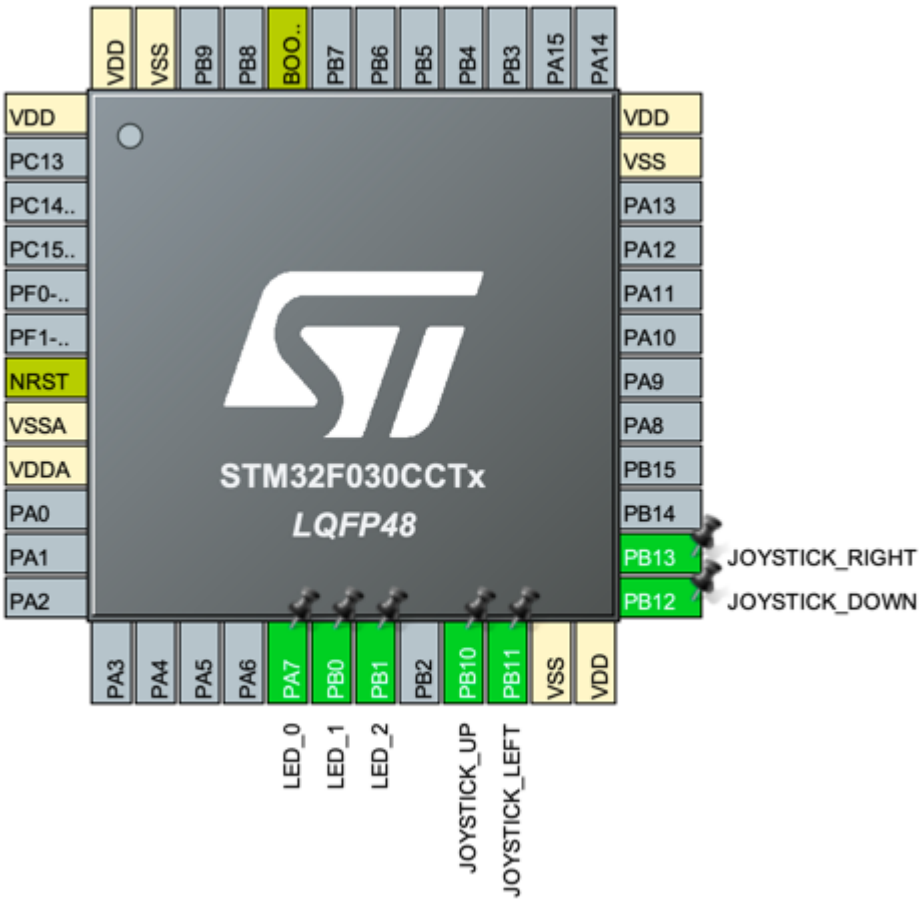
Further files of this task: See `Task_4/`.

4.1 Better blinky with HAL

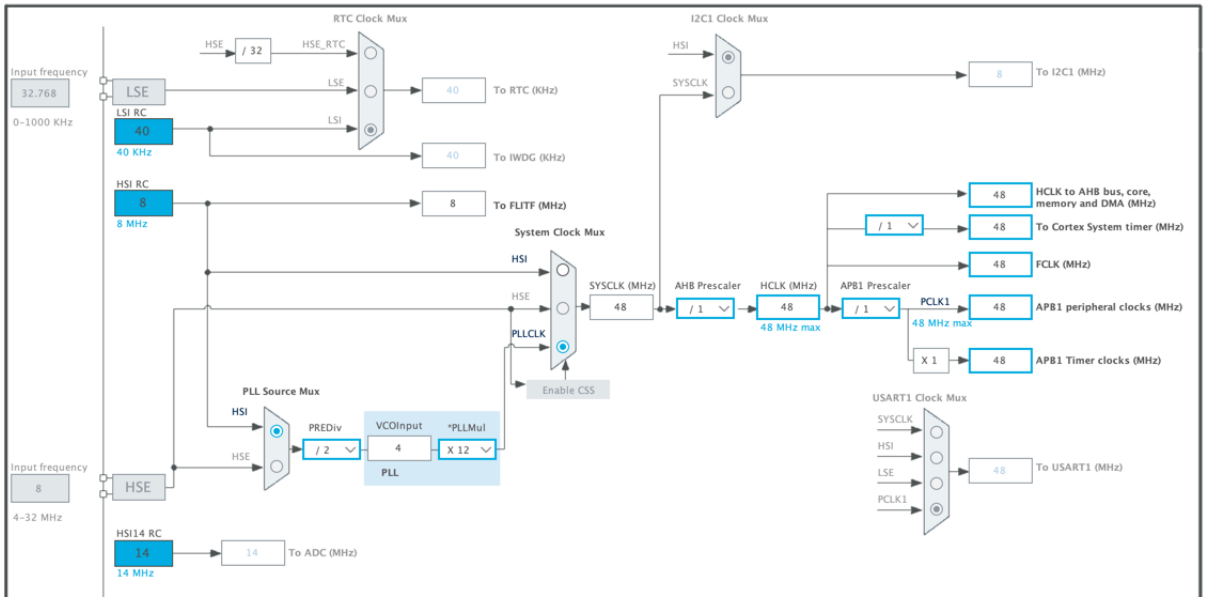
In this and subsequent HAL tasks, we used STM32CubeMX. It generates a project structure and code that initialises the M0 features needed for each task.

In this task, we used STM32CubeMX to

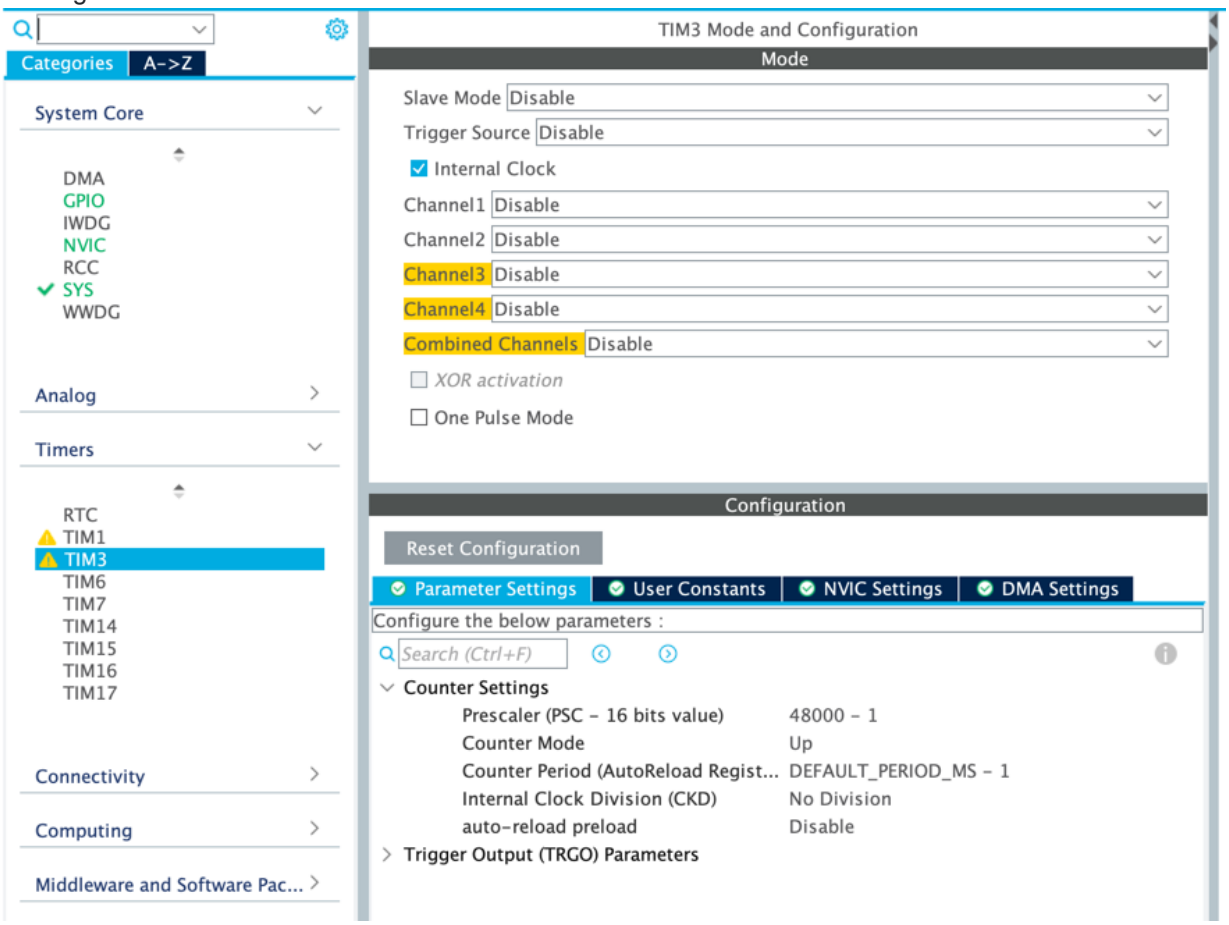
- set the GPIO Input/Output pins on the chip,



- set the system clock to 48 MHz and



- to configure TIM3



using the same settings we set manually in Task 2.

Note: We needed to adapt the resulting code and the `CMakeLists.txt` file to our needs and to our project structure, so regeneration of the code from STM32CubeMX will arise problems.

To compile the HAL tasks, use CMake:

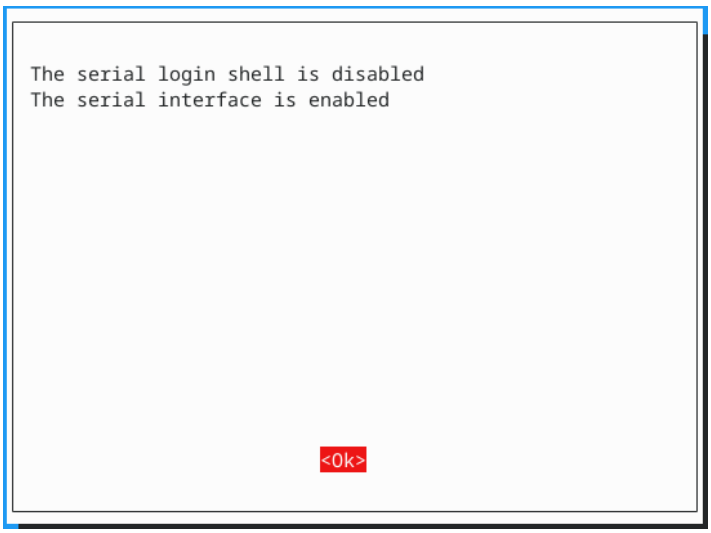
```
mkdir build
cd build

cmake ..
make
```

4.2 UART

To use UART, we need to enable it first

- open the raspberry pi config with: `sudo raspi-config`
- select `3. Interace options`
- select `Serial Port`
- It will ask you two questions.
  1. `Would you like a login shell to be accessible over serial?` -> select `No`
  2. `Would you like the serial port hardware to be enabled?` -> select `Yes`
- You should see the following message:



- select `OK`
- select `finish and reboot`

The involved GPIO pins on the raspberry pi for the UART communication are

- 15 (rx = receive)
- 14 (tx = transmit)

On our extension board, they are connected to USART 5 on the M0

On the raspberry pi we can now select `/dev/serial0` as UART port.

Note: `/dev/serial0` is a symbolic link to `/dev/ttyS0`

For more information, see the official documentation (Configuration - Raspberry Pi Documentation)

Python Software Preparations on the Raspberry Pi

We use the `pyserial` python library, which can be installed via `pip` in the following way:

```
# Change to your home folder
cd

# Install python3 and pip
sudo apt install python3-full pip

# Create and activate a virtual environment
python3 -m venv .venv
source .venv/bin/activate

# Install pyserial
pip3 install pyserial
```

Execute Benchmark

1. Compile/Run the M0 Code normally
2. Start the Python Code for the Raspberry Pi (`Code-raspi/main.py`). The script `raspi-run.sh` can be used to simplify this when you are on your host machine:

   ```
   ./raspi-run.sh <Raspberry Pi's Hostname/IP/SSH-Host-Name> <Raspberry Pi's username>
   ```

   Note: Within CLion, the Build Target "Shell Script" can be used to call the script. Be sure to set the working directory to 2_Universal_Asynchronous_Receiver_Transmitter.

Testing UART with 1000x10 Bytes:

```
UART_Benchmark started with 1000 times 10 bytes.
Test succeeded for baudrate   9600. Elapsed time: 20.95 s.
Test succeeded for baudrate  19200. Elapsed time: 10.53 s.
Test succeeded for baudrate  57600. Elapsed time: 3.56 s.
Test succeeded for baudrate 115200. Elapsed time: 1.82 s.
UART_Benchmark finished successfully.
```

We can observe that the time required for the communication approximately decreases linearly to the baudrate.

Side note about USART vs UART

UART stands for **U**niversal **A**synchronous **R**eceiver **T**ransmitter whereas the **USART** stands for **U**niversal **S**ynchronous **A**synchronous **R**eceiver **T**ransmitter. The term Synchronous enables the USART to send an additional clock signal to the receiving device. The data is then sampled at a predefined edge (Rising or Falling) of the clock. The USART mode uses **3 pins** (clock, Tx and Rx) compared to the **2 pins** (Tx and Rx) used in the UART. Source 26.6.24

Task 3

Addresses and datasheets

v3 Sensor Board:

Sensor	Datasheet	Slave Address (default) (7bit)	read address (8bit)	write address (8bit)
LPS331AP (barometer)	URL	1011101b	10111011b	10111010b
LSM404D (accelerometer and magnetometer)	URL	0011101b	00111011b	00111010b
L3GD20H (gyro)	URL	1101011b	11010111b	11010110b

Note: The L3GD20H sensor will not be used in this task, as reading gyroscope measurements is not required.

v5 Sensor Board:

Sensor	Datasheet	Slave Address (default) (7bit)	read address (8bit)	write address (8Bit)
LSM6DS33 (gyro and accelerometer)	URL	1101011b	11010111 (D7h)	11010110 (D6h)
LIS3MDL (magnetometer)	URL	0011110b	00111101 (3Dh)	00111100 (3Ch)
LPS25H (barometer)	URL	1011101b	10111011 (BBh)	10111010 (BAh)

The source code was partly inspired by the official Arduino libraries for the sensors.

- GitHub - pololu/lsm6-arduino: Arduino library for Pololu LSM6DS33 and LSM6DSO boards (under MIT License)
- GitHub - pololu/lps-arduino: Arduino library for Pololu LPS25H and LPS331AP boards (under MIT License)
- GitHub - pololu/lis3mdl-arduino: Pololu Arduino library for LIS3MDL magnetometer (under MIT License)
- GitHub - pololu/lsm303-arduino (Under MIT License)

Measurement Process

The M0 waits for two numbers to be sent via UART at first:

1. The number of measurements to take
2. Additional delay between each measurement in ms (Note: Without additional delay, expect to receive about one new measurement dataset per second with the v3 board. This is due to the M0 waiting for each sensor to have new data available.)

The measurements are then transmitted via UART in a CSV-like style.

To run this process:

1. Compile/Run the M0 Code normally
2. Start the Python Code for the Raspberry Pi (`python-uart/main.py`).
    - As in 4.2, `raspi-run.sh` may be used to simplify this when you are on your host machine.
    - When `main.py` is called without any arguments, it asks for the number of measurements to take and if you want an additional delay between each measurement
    - `main.py` may be called with arguments instead so that the output can be redirected into a csv file directly.

      ```
      python3 main.py <number of measurements> <delay> > file.csv
      ```

    - Note: Activate the virtual environment before calling `main.py` manually.
3. While the M0 is still running, `main.py` can be called again as often as needed.

Exemplary UART transfers

An example run of `main.py` :

```
pi-tobii@esa-pi:~ $ source .venv/bin/activate
(.venv) pi-tobii@esa-pi:~ $ python3 main.py 15 3000
>>>     Barometer ID: 0xbb
>>> Accelerometer ID: 0x49
>>>  Magnetometer ID: 0x49
 t [s], T [K],   p [Pa],a_x [m/s^2],a_y [m/s^2],a_z [m/s^2],  B_x [T],  B_y [T],  B_z [T]
  0.01,315.49, 89284.64,       0.10,       0.09,      -8.25, 0.000017, 0.000046, 0.000031
  3.02,315.49, 89281.03,      -0.80,      -9.00,      -0.14, 0.000029, 0.000060,-0.000022
  6.04,315.50, 89292.24,      -9.95,      -0.40,       2.09, 0.000042,-0.000006,-0.000034
  9.05,315.50, 89290.84,      -0.48,       9.50,       0.73, 0.000009,-0.000039,-0.000033
 12.06,315.49, 89295.65,       5.26,      -1.80,       4.51, 0.000034, 0.000055,-0.000005
 15.07,315.48, 89293.97,       6.10,       5.70,      -3.06,-0.000013,-0.000010, 0.000041
 18.09,315.44, 89290.36,       1.53,       9.19,      -1.08, 0.000004,-0.000023, 0.000034
 21.10,315.45, 89283.35,      10.25,      -0.67,       2.33,-0.000031, 0.000032, 0.000008
 24.11,315.43, 89301.20,       0.50,      -2.09,      11.09, 0.000013, 0.000047,-0.000038
 27.12,315.48, 89285.25,      14.62,       4.47,      10.50, 0.000039, 0.000040,-0.000025
 30.14,315.41, 89290.75,      19.60,      11.19,      18.38, 0.000038, 0.000037,-0.000027
 33.15,315.34, 89287.01,      -0.99,       9.69,       1.17, 0.000008,-0.000032,-0.000033
 36.16,315.34, 89290.94,     -10.15,       1.92,       1.50, 0.000032,-0.000015,-0.000029
 39.18,315.36, 89284.91,      -9.81,       0.58,       2.17, 0.000035,-0.000008,-0.000033
 42.19,315.34, 89283.89,      -9.70,      -0.13,       2.22, 0.000036,-0.000006,-0.000034
>>> End of measurements.
```

During measurement, the Raspberry Pi has been tilted and shaken, so a change in the acceleration values can be observed.
The pressure values are quite low as this task was done during holiday in the Alps.

Task 4

The measurement process is identical to Task 3.

Calibration

Before any measurement can be received, calibration has to happen.
Set the minimum illuminance by moving the joystick downwards, and the maximum illuminance by moving the joystick upwards.

Between measurements, the calibration values are kept. To delete the calibration, reset the M0. At any time, minimum or maximum illuminance may be recalibrated by moving the stick upwards or downwards again.

The Board LEDs show the status of the calibration:

- LED_1 (red): on while not fully calibrated
- LED_0 (green): steady while minimum illuminance calibrated, flashing after recalibration of minimum illuminance
- LED_2 (green): steady while maximum illuminance calibrated, flashing after recalibration of maximum illuminance

Exemplary UART transfers

```
pi-tobii@esa-pi:~ $ source .venv/bin/activate
(.venv) pi-tobii@esa-pi:~ $ python3 main.py
Number of measurements?               > 15
Additional delay between measurements? > 3000
>>> Ready to calibrate.
>>> Warning: Calibration failed. Min value has to be smaller than max value!
>>> Please try again.
>>> Calibration: min =  231.30, max = 3959.20
 t [s],E_V [%]
  0.00,  99.97
  3.00,  98.06
  6.00,  96.61
  9.01,  75.80
 12.01,  49.27
 15.01,  98.01
 18.01,  98.39
 21.01,  98.44
 24.01,  69.98
 27.01,  72.47
 30.01,  42.99
 33.01,   0.61
 36.01,  98.09
 39.02,  87.55
 42.02,  37.47
>>> End of measurements.
```

During measurement, a hand and different objects were moved across the sensor.