Documentation of Task 02

Further files of this task: See `Task_2/`.

Prerequisites:

- Install `gcc-arm-none-eabi` on the Raspberry Pi so that we are able to cross-compile to the Cortex-M0 from the Pi itself:

  ```
  sudo apt install gcc-arm-none-eabi
  ```

- Note: Ensure to clone submodules too, s.t. the CMSIS and HAL drivers are available for compilation:

  ```
  git submodule update --init --recursive
  ```

Compilation and Flashing:

In the Makefile, `better-blinky.c` (Task 2) is set up as the default source code to compile and execute on the extension board. To compile or flash `blinky.c` (Task 1) instead, run

```
make CSRC=blinky.c
```

and

```
make flash CSRC=blinky.c
```

respectively.

Makefile

The following additions were made:

- Line 6: the path to the CMSIS library. In our repository, STM32CubeF0 is located within /utils.
- Line 44: the full command to download the compiled ELF file to the Cortex-M0. Therefore, the `stm32f0raspberry.cfg` file is needed, which we created in Task 1. Information on how the program command works can be found in the OpenOCD Documentation at page 126.

CMSIS

The CMSIS library for our M0 processor is located at `STM32CubeF0/Drivers/CMSIS/Device/ST/STM32F0xx/Include/stm32f030xc.h`.

Note: There is also a device-independant header file in the same folder `stm32f0xx.h` which by itself includes the actual device-specific header file based on a predefined macro.

Some Remarks About `better-blinky`

- We think of "LED toggle *period*" as the time a whole blinking cycle takes, i.e. the time between the LED being turned on and the time it is being turned on next.
  E.g., if the period is 1000 ms, then the LED gets turned on for 500 ms, turned off for 500 ms and then back on again.
- We decided to limit the period setting to 50 to 10000 ms.
- The joystick pins are configured s.t. the movements seem natural when J2 is facing upwards. This is different to the STICK_UP, STICK_LEFT, ... signals on the extension board.

Important registers

- `MODER` mode register (p. 134)
- `PUPDR` pull-up/pull-down register (p. 135)
- `SYSCFG->EXTICR` external interrupt configuration register 1 (p. 144)
- `IMR` Interrupt mask register (p. 177)
- `RTSR` Rising trigger selection register (p. 177)
- `FTSR` Falling trigger selection register (p. 178)
- `ODR` output data register (p. 136)
- `CR1`: 'Control Register 1' – this register is used to enable and disable the peripheral.
- `CNT`: 'Counter Register' – this register hold's the timer's current counter value. It counts up from 0 once the timer is started.
- `PSC`: 'Prescaler Register' – this register will hold the timer's prescaler. A prescaler value of `N` will tick the timer's counter register up by one every `N+1` clock cycles.
- `ARR`: 'Autoreload Register' – the autoreload value is the 'period' of the timer. An autoreload value of `N` will cause the timer to trigger an update event every time the `CNT` register counts up to `N`.
- `EGR`: 'Event Generation Register' – Setting the `UG` bit in this registers resets all of the timer's counters and tells it to use the currently-set prescaler/autoreload values.
- `DIER`: 'DMA/Interrupt Enable Register' – Setting the `UIE` bit in this register sets the timer to trigger a hardware interrupt when an update event occurs. Usually, that happens when the 'Counter' register matches the 'Autoreload' value.

- `SR`: ‘Status Register’ – The `UIF` flag is set in this register when a timer’s hardware interrupt triggers, and must be cleared before another one can occur.

source: Reference manual and “Bare Metal” STM32 Programming (Part 5): Timer Peripherals and the System Clock – Vivonomicon's Blog (23.5.24)