Embedded Systems Hands-On 1: Design and Implementation of Hardware/Software Systems

Task 3: Cortex-M0 Bare-metal Programming

Carsten Heinz

June 5, 2020

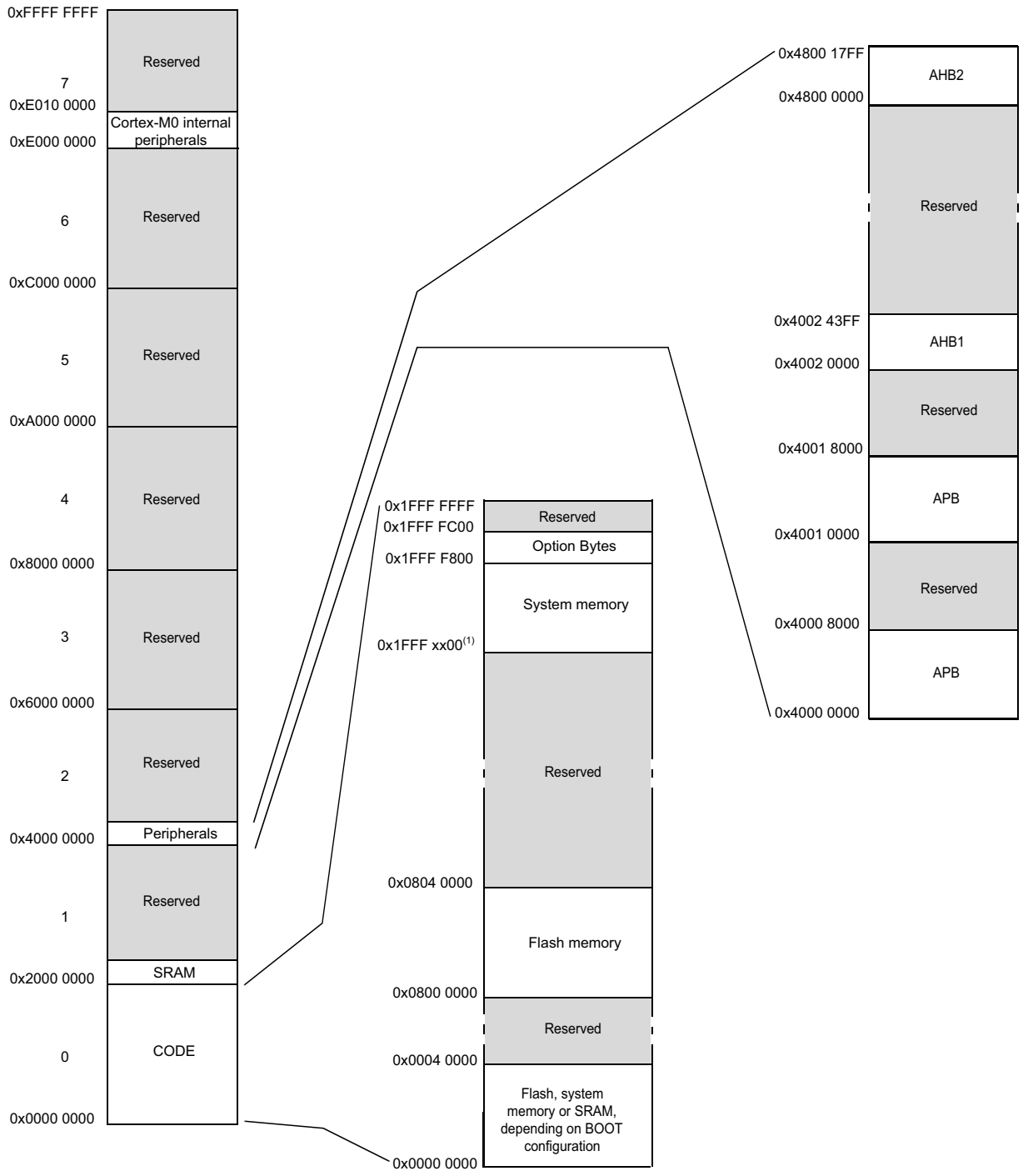
TECHNISCHE UNIVERSITÄT DARMSTADT

This task handles the basic programming of the Cortex-M0. It is scheduled for one week.

Summary

- Controlling GPIO and timers via CMSIS
- Event-based control with polling and interrupts

The communication with peripheral components (e.g., GPIO, timers, bus controllers) of ARM processors is realized as memory mapped IO, i.e., by accessing certain memory addresses. The memory map of the Cortex-M0 looks as follows:



To simplify the programming of ARM processors from different vendors, many common features such as the sys-tick counter or GPIO blocks are accessible via vendor-independent addresses and functions. These are summarized in the CMSIS libraries provided by the vendors. The standardized naming scheme and header files simplify the source code porting between different ARM vendors.

The information required to solve this task can be found in the Cortex [M0-Programming] manual, the Cortex [M0-Reference] manual, and the schematic of the extension board. Furthermore, a basic development environment for the Cortex-M0 can be found in the [GitLab] under `ESH01/Materials/Environment_baremetal`. You need [ARM-GCC] (`arm-none-eabi-gcc`) as a cross-compiler (available at [ARM-GCC]) and put the executable onto your path. After inserting your source code into `Environment_baremetal/main.c`, execute `make` to compile your code. Implement the Makefile target `make flash`, so that the compiled binary gets downloaded to the Cortex-M0. Please note that [ChibiOS] will be provided for a later task, *Task 3 has to be solved without the functions provided by this realtime operating system*.

1 Blinky: The embedded "Hello World"

As `printf` based debugging requires setting up a text output peripheral device first, the most basic functional test of a microcontroller toggles an LED.

Summary

- Toggle one of the LEDs of the extension board.
- Control a timer to realize 500 ms on/off time.

Find an appropriate Cortex-M0 GPIO pin connected to one of the LEDs of the extension board. The GPIO pins are organized in ports labeled with capital letters (e.g., A8 is pin 8 of port A). Find the peripheral registers controlling the pin direction (output) and the state (on/off) of your LED pin.

Use a timer to realize an accurate LED toggle period of 500 ms. Therefore, the appropriate peripheral registers have to be found to

- configure and activate the 48 MHz timer input clock (`CK_INT`),
- configure the timer prescaler to realize a 100 kHz timer counter clock (`CK_CNT`),
- configure the timer auto-reload value to realize an overflow each 500 ms,
- configure the timer to operate in upcounting mode,
- activate the timer,
- check the timer status in a polling ("busy waiting") loop and toggle the LED on each overflow.

Due to the CMSIS header files, the register names used in the Cortex-M0 manuals can be used to access the relevant registers. For example, `TIM3->CR1 |= TIM_CR1_CEN` will set the `CEN` bit of the `CR1` register of timer 3. *Do not use other libraries besides CMSIS to realize this task*. You may debug your code with [OpenOCD] and [Eclipse] as practiced in Task 2.

Minimum Expected Documentation

- The documented Blinky code.

2 Better Blinky

To improve the Blinky example, the LED toggle rate and the toggled LED should be controllable by the joystick of the extension board.

Summary

- Use an ISR to toggle the LED
- Capture the left (right) joystick movement to switch to the previous (next) LED.
- Capture the up (down) joystick movement to decrease (increase) the LED toggle period by 50 ms.

Capturing the joystick and the timer status in a single polling loop results in inaccurate timing, as the joystick status may be checked while the timer generates and overflow thus resulting in unpredictable jitter at the LED. The timer overflow event thus has to be handled with a higher priority than the joystick status. Configure the timer to generate an interrupt event on each overflow and implement a corresponding ISR to actually toggle the currently selected LED.

Define an appropriate data structure to represent the selected LED and toggle rate. Capture the joystick movement either by polling or by interrupts and adjust the LED toggle settings as described above. Make sure to limit the settings to a reasonable range. Find out how to actually modify the LED toggle rate.

Minimum Expected Documentation

- The documented Better Blinky code.

Bibliography

ARM-GCC ARM. *GCC ARM Embedded*. 2018. URL: https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads (visited on 2019-04-16).

ChibiOS *Free embedded RTOS*. URL: http://www.chibios.org (visited on 2018-03-08).

Eclipse *C/C++ Development Tooling*. 2016. URL: https://eclipse.org/cdt/ (visited on 2018-03-08).

GitLab *Code, test, and deploy together*. URL: https://about.gitlab.com/ (visited on 2018-03-08).

M0-Programming ST. *STM32F0xxx Cortex-M0 programming manual*. 2012. URL: http://www.st.com/resource/en/programming_manual/dm00051352.pdf (visited on 2018-03-08).

M0-Reference ST. *STM32F030x4/x6/x8/xC and STM32F070x6/xB advanced ARM-based 32-bit MCUs*. 2017. URL: http://www.st.com/resource/en/reference_manual/dm00091010.pdf (visited on 2018-03-08).

OpenOCD Dominic Rath. *Open On-Chip Debugger*. URL: http://openocd.org (visited on 2018-03-08).

Acronyms

CMSIS Cortex Microcontroller Software Interface Standard
GPIO General Purpose IO
IO Input/Output
ISR Interrupt Service Routine
LED Light-emitting Diode
RTOS Real-Time Operating System