EMBEDDED CONTROL

Dr. Luca De Cicco
luca.decicco@poliba.it
https://c3lab.poliba.it/LDC

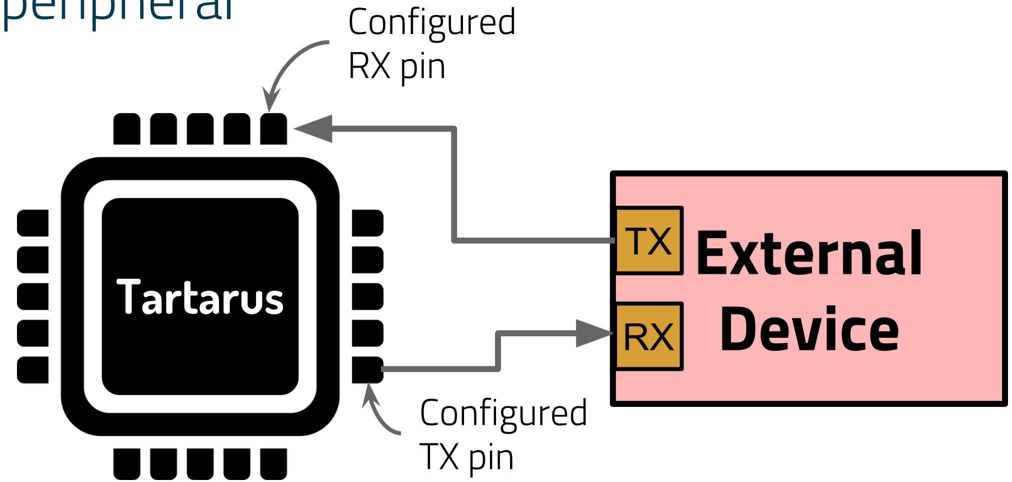
Let’s code!

The Tartarus

MCU assignment

The “Tartarus MCU” assignment

- The purpose of this assignment is to make you experiment with registers (without an HW) on a fictitious MCU called `Tartarus`
- You will have to write a firmware that allows communication between the MCU and an external (fictitious) device through a serial I/O peripheral



About the Tartarus MCU

- Tartarus MCU has **144 pins** and only one serial I/O peripheral
- The MCU pins are numbered from 0 to 143
- The aim is to program the serial I/O peripheral so that the external device can communicate with the MCU
- The peripheral is associated to two pins:
  - **RX pin**: used to receive data sent from the external device to the MCU
  - **TX pin**: used to send data from the MCU to the external device
- TX and RX pins can be mapped only to pins from **32 to 143** (the other pins are reserved for other functions)
- The **peripheral registers** are mapped to a **known memory region**

Tatarus Serial I/O Peripheral Registers

The peripheral can be programmed through three registers (`CTRL, RXDATA, TXDATA`):

CTRL (control register): offset 0x00

31	…	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RESERVED			Receive (RX) Pin								Transmission (TX) Pin								B	R	W

- Bit 0 (`W`): to request the transmission of data this bit has to be set to 1
- Bit 1 (`R`): to request data to be received this bit has to be set to 1
- Bit 2 (`B`): this is a status bit set by the MCU (**you are not allowed to change it!**) that indicates whether the peripheral is busy transmitting/receiving data (`B = 1`) or idle (`B = 0`)
- Bit 3:10 (`TX_PIN`): PIN number of the MCU where the TX line is connected
- Bit 11:18 (`RX_PIN`): PIN number of the MCU where the RX line is connected

RXDATA (offset 0x04): where data received is stored (32bit unsigned int)

TXDATA (offset 0x08): where data to be transmitted is stored (32 bit unsigned int)

Details on Peripheral

- TX/RX operations take some MCU ticks to be performed (the number of ticks is not precisely known)
- Thus, the peripheral can be busy either receiving or transmitting according to the **CTRL** register config
- If you try requesting an operation while the peripheral is busy, the **MCU will halt**
- If you try performing TX/RX operations with wrong TX or RX pins (see slide n.4) , the **MCU will halt**

What you have to do

- The assignment is divided into two steps:
  - You will first write a library (fancy way to refer to “a set of functions”) to access the peripheral
  - You will then use this library to write a “firmware” to implement a specific logic involving communication between the MCU and the external device
- You will have to demonstrate that the firmware is correct (write a short report)

The Library

- Define a `struct` named `FooPeriph_t` to access the three registers knowing the starting address of the first register (i.e., `FOO_PERIPH`)
- **Minimal** (can implement more) set of functions to implement:
  - `foo_periph_init`: initialize the peripheral by assigning the TX and RX pins and defaulting to `R=0`, `W=0`, `B=0` (no pending TX/RX, the peripheral is idle)
  - `foo_periph_tx_data`: send the specified 32bit variable to the peripheral (recall you have to set the `TXDATA` register)
  - `foo_periph_rx_data`: receive and return a 32bit variable from the peripheral (the peripheral will store the received data in the `RXDATA` register)

Firmware

Write a firmware that implements the following cycle of operations repeatedly:

1. Read 10 32-bit unsigned int readings from the peripheral
2. Compute the average value of the 10 received readings
3. Send the average value (32-bit unsigned int) to the peripheral
4. Go back to step 1

Code organization

Your code base will be structured this way (minimal setup):

- `mcu.c/mcu.h/mcu-api.h`: given **(DO NOT TOUCH!)**
- `lib-periph.c/lib-periph.h`: implementation of the library (you code it)
- `firmware.c`: implementation of the firmware
- if you need other modules you are welcome to implement them

Instructions to program the Tatarus MCU

- All the information you need to develop the library and firmware is in `mcu-api.h` that contains:
- **FOO_PERIPH**: the address in memory of the peripheral registers base address
- **MCU_STEP**: an instruction that is used to simulate MCU clock ticks (i.e., to simulate the time)
- Each time an operation is done you are required to call MCU_STEP
- For instance, if you want to wait for 2 MCU ticks, it is sufficient to execute MCU_STEP twice

Instructions to program the "MCU"

- Besides `mcu-api.h` you also have:
- `mcu.c`: an obfuscated C code that simulates the MCU along with the peripheral (so the MCU is a "black box")
- **You should not look at `mcu.c`** since you cannot open the MCU in the real life, you just read specs!
- The only way to interact with the MCU is through the registers mapped at the **FOO_PERIPH** address
- If you do something not permitted (f.i., reading or writing when the peripheral is still busy doing another read/write operation) an error will occur and the **MCU will halt** (you will have to CTRL-C to exit!)

Boilerplate firmware.c

```
#include <stdint.h>
#include <stdio.h>
#include “mcu-api.h”
/* other includes */
void main()
{
     while (1) {
          …
          /* one operation */
          MCU_STEP;
          …
     }
}
```

Happy Coding!