Assignment 3B: Firmware Dump

Objectives for this assignment:

- Create an environment to debug STM32 targets.
- Get familiar with a standard USB-to-TTL serial adapter
- Learn how to dump the firmware in different ways.

Bill of Materials

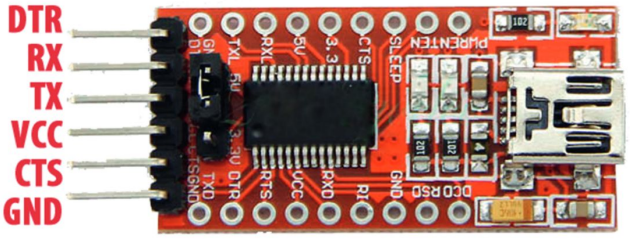
To complete this lab you will need:

- STM32 NUCLEOL152RE evaluation board
- 2x USB cable Type-A to type-B mini
- 1x FDTI adapter USB to Serial
- Jumper Wires

Background: USB-to-TTL serial adapter

Up to now, you interacted with the nucleo via a serial interface exposed on the USB connection. However, quite often (including in this exercise), you will find serial interfaces directly exposed on header pins, which directly map to UART/USART ports of the MCU.

In assignment 3A, you read data from such a PIN using the logic analyzer, however, this approach does not allow you to send data to your target.
Hence, we included the following piece of hardware:



In this exercise, you will only need the RX (receive), TX (transmit), and GND (ground) pins. RX of the adapter connects to TX of the target UART port, and TX of the adapter to RX of the UART port, while GND connects to GND.

Important:

The board has a jumper to switch VCC and logic levels between 5V and 3.3V.
By default, the jumper is set to 5V (as shown in the image). While the nucleo supports 5V operation for the UART ports, its analog ports do not. Hence, to be on the safe side, please move the jumper from pin 2-3 to pin 1-2 to enable usage of 3.3V levels.

Part 1: Debug environment (1pts)

Creating a debug environment is an essential step for this and future assignments.
The Nucleo evaluation board has a built-in ST-LINK debugger/programmer (top part of the board). In this assignment, you will learn how to interface with this debugger.

1. Install debugging software

While ST provides its own debugging tool (stutil), we recommend using openocd, as it is a versatile and powerful tool, which can connect to various boards and debuggers (and not just to ST products).

To obtain openocd, run the following command:

- `sudo apt install openocd`

Alternatively, you can also build it from source (http://openocd.org/getting-openocd/), however, the names of the files below may have changed.

2. Connect OpenOCD

When running openocd, you need to specify configuration files. The suggested paths are the files on a standard Ubuntu 20.04 distribution. To start OpenOCD, run the following command:

```
openocd -f /usr/share/openocd/scripts/interface/stlink-v2-1.cfg -f
/usr/share/openocd/scripts/target/stm32l1.cfg
```

If everything worked out, OpenOCD now opened a telnet server on port **4444**. This server allows various interactions with the target, feel free to play around!
Furthermore, OpenOCD also provides a GDB server, available at port **3333**.

2. Connecting GDB

You already should be familiar with GDB, and it's the bread and butter tool for dynamic debugging. However, the Nucleo board is based on the ARM architecture, so your standard x86-GDB won't work. We recommend using gdb-multiarch instead:

```
sudo apt install gdb-multiarch
```

(Note that arm-none-eabi-gdb may work as well, but requires additional setup effort.)

Now, start gdb-multiarch and execute the following commands to specify the architecture and connect to the remote target:

```
set architecture arm
target remote :3333
```

3. Read the flag!

The flag is located at address 0x20000180. Use GDB to extract it.

Deliverable:

- A text file containing the flag.

Hints:

- The flag is decrypted during runtime. If you cannot identify the flag, run the target for some additional time, then break, and try again.

Troubleshooting:

- **OpenOCD is not able to connect to the target .**
  You either need to run OpenOCD as root user, or create an udev rule for the ST-Link:
  1) create a file **/etc/udev/rules.d/35-stlink-v2.rules** with the following content:
     ```
     #STLINK V2 and V2.1
     ATTRS{idProduct}=="3748", ATTRS{idVendor}=="0483", MODE="666",
     GROUP="plugdev"
     ```
  2) Verify your user is in the plugdev group
  3) reload udev rules: `sudo udevadm control --reload-rules`
  4) Unplug and replug the Nucleo

- **The instructions I disassemble with GDB don't make any sense?!**
  Since the STM32 runs Thumb code, it may be necessary to use the following command in GDB before disassembling:
  ```
  set arm force-mode thumb
  ```

Part 2: Dump the firmware (4pts)

Your objective for this part is to obtain an exact copy of the firmware present on the target. The firmware lives on the FLASH memory of the Nucleo board and can be dumped in multiple ways.

1. Dumping using GDB/OpenOCD

As you have debugging capabilities, you can just dump the firmware using them! If you finished Part 1 of this assignment, you should have all tools ready to do so!

2. Dumping via the bootloader

Unfortunately, you will rarely encounter enabled debug interfaces on real-world devices. Hence, this part demonstrates another common technique of firmware extraction: (ab-)using the bootloader!

More information about the bootloader can be found in the STM32L152RE datasheet:

Boot modes

At startup, boot pins are used to select one of three boot options:

- Boot from Flash memory
- Boot from System memory
- Boot from embedded RAM

The boot from Flash usually boots at the beginning of the Flash (bank 1). An additional boot mechanism is available through user option byte, to allow booting from bank 2 when bank 2 contains valid code. This dual boot capability can be used to easily implement a secure field software update mechanism.

The boot loader is located in System memory. It is used to reprogram the Flash memory by using USART1, USART2 or USB. See Application note “STM32 microcontroller system memory boot mode” (AN2606) for details.

You can enter Boot from System memory (which contains the bootloader) by setting the BOOT0 pin to high by connecting a 3.3V pin with BOOT0.
(Normally, BOOT0 is at 0V due to the R33 pull-down resistor mounted on the Nucleo board.)

Afterwards, you can interface the bootloader via the Serial1 interface (exposed on pins PA_9/PA_10).
The bootloader speaks a custom serial protocol over this UART interface. You can find it’s specification in **AN2606** and **AN3155.** Read these documents and learn how to communicate with the bootloader and dump the firmware!
We attached (some) relevant excerpts of the application notes in the Appendix.

Deliverable:

- The dumped firmware
- A picture of the wiring to interact with the bootloader
- A script carrying out the interaction with the bootloader to dump the firmware

Hints:

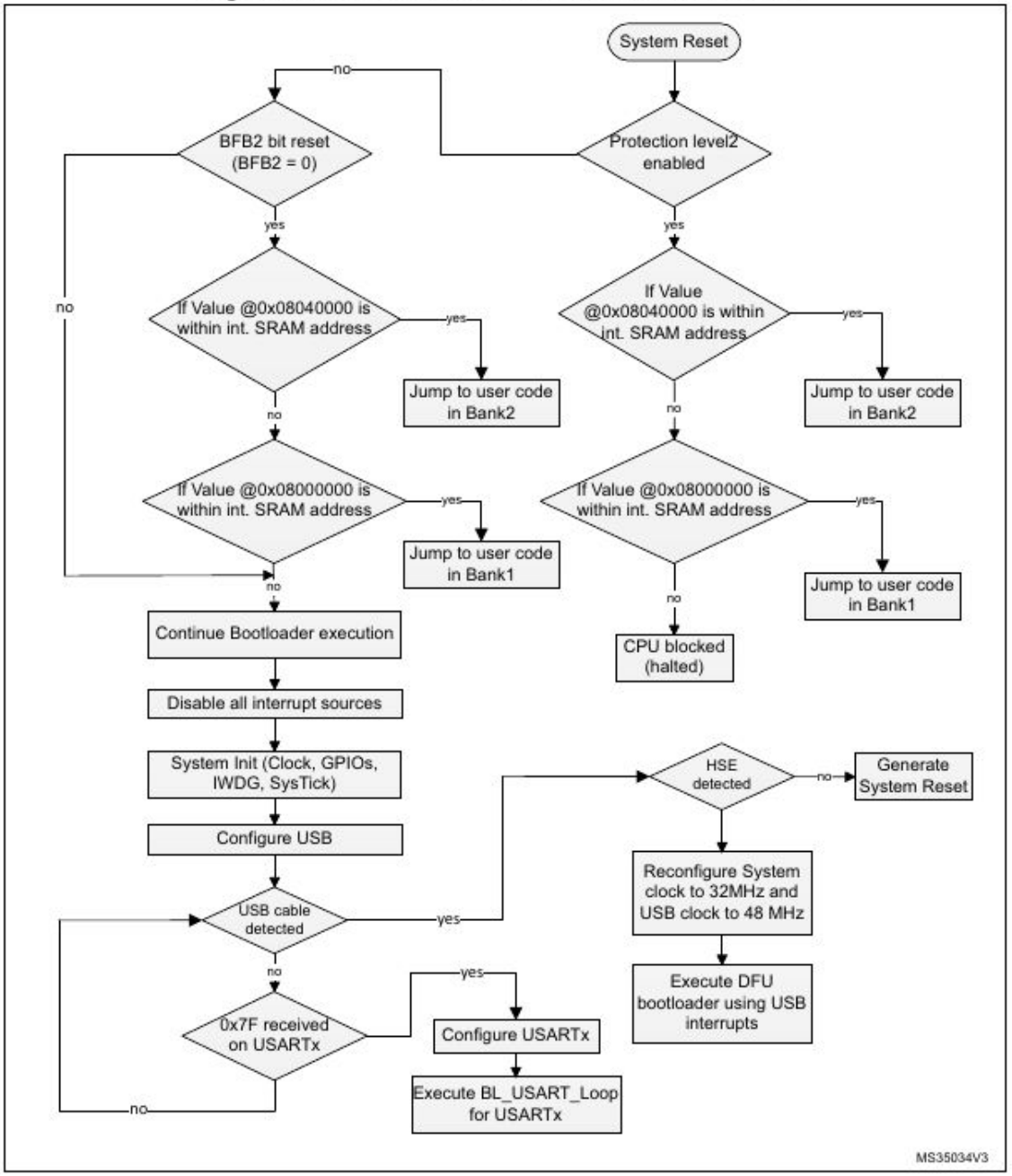
- Where is the firmware stored? Have a look at the **Memory Mapping** chapter of STM32L152RE datasheet.
- You most likely want to script the interaction with the bootloader. For this, we recommend the `pySerial` package.

Materials:

- STM32L152RE datasheet: http://www.st.com/resource/en/datasheet/stm32l152ze.pdf
- AN2606: https://www.st.com/resource/en/application_note/cd00167594-stm32-microcontroller-system-memory-boot-mode-stmicroelectronics.pdf
- AN3155: https://www.st.com/resource/en/application_note/cd00264342-usart-protocol-used-in-the-stm32-bootloader-stmicroelectronics.pdf

Appendix:

Figure 69. Bootloader selection for STM32L1xxxE devices



3.4 Read Memory command

The Read Memory command is used to read data from any valid memory address (refer to the product datasheets and to AN2606 for more details) in RAM, Flash memory and the information block (system memory or option byte areas).

When the bootloader receives the Read Memory command, it transmits the ACK byte to the application. After the transmission of the ACK byte, the bootloader waits for an address (four bytes, byte 1 is the address MSB and byte 4 is the LSB) and a checksum byte, then it checks the received address. If the address is valid and the checksum is correct, the bootloader transmits an ACK byte, otherwise it transmits a NACK byte and aborts the command.

When the address is valid and the checksum is correct, the bootloader waits for the number of bytes to be transmitted – 1 (N bytes) and for its complemented byte (checksum). If the checksum is correct it then transmits the needed data ((N + 1) bytes) to the application, starting from the received address. If the checksum is not correct, it sends a NACK before aborting the command.

The host sends bytes to the STM32 as follows:

Bytes 1-2:	0x11 + 0xEE
Wait for ACK	
Bytes 3 to 6	Start address byte 3: MSB, byte 6: LSB
Byte 7:	Checksum: XOR (byte 3, byte 4, byte 5, byte 6)
Wait for ACK	
Byte 8:	The number of bytes to be read – 1 ($0 < N \leq 255$);
Byte 9:	Checksum: XOR byte 8 (complement of byte 8)