Embedded Meetup
07.11.2023

Software Bootloader – What to do when dedicated MCU pins forgotten?

by K. Przygoda

Konrad Przygoda

Senior Software Engineer

Work background

Education: MSc & PhD in Electronics by Lodz University of Technology, Poland
Postdoc position in Deutches Elektronen-Synchrotron in Hamburg, Germany

Areas of expertise: RF, analog and digital hardware designs, VHDL & C Embedded

Work experience at Sii: since 4th February 2022, Aptiv – VSG carryback & ST400 (+ SW Test Lead), currently TC4x Aurix evaluation

Work interests: Piezo actuators & sensors, Laser Synchronization, Power Amplifiers, FPGAs, STM32/IFX MCUs, BLE5.0, FreeRTOS, Automotive & Autosar

About me

After hours: Basketball, swimming, jogging, bike riding

Agenda:

1. Introduction
2. Motivation
3. How to start with Software bootloader?
4. FLASH Memory organization for the Bootloader and Application
5. Program Start, Memory Mapping and Interrupt Vector Table
6. Bootloader Template
7. Bootloader Api
8. Programming script
9. Live demo

https://source.android.com/docs/core/architecture/bootloader?hl=pl

Real Example

Medical Application

- 30x PCBs already manufactured
- Clinic tests where scheduled
- The main functionality confirmed
- Patients were waiting for stimulators
- Doctors were waiting for the results
- Assembly company were waiting for the PCBs
- A single PCB mistake has been discovered: one of the dedicated pins for embedded bootloader missing (mainly due to easier PCB layout) -> No possible to cut the traces and route with extra wire (back drilling will cost more than PCB)

https://www.dreamstime.com

- ***What to do?***
  - Correct a mistake and manufacture new PCBs (additional 3-4 weeks, unexpected costs)
  - Write a software bootloader and make use of currently connected interface pins for firmware update

Motivation

Embedded bootloader – advantages and disadvantages

What is this?

- Stored in internal boot ROM memory of MCU
- Programmed by Manufacturer during production
- The main task is to download the application program to the internal FLASH memory through one of the available serial peripherals
- It transfers and update the FLASH memory code, the data, and the vector table sections
- A communication protocol is defined for each interface, with compatible command set and sequences

https://www.st.com/en/development-tools/stm32cubeprog.html

Advantages:

- Ready to use with activation pattern i.e. BOOT (pins)
- Manufactures provides a free dedicated programmers i.e. STM32 Programmer, IFX memtool

Disadvantages:

- Only dedicated pins for each serial interface can be used

i.e. stm32wb55xx

  - USART1 on pins PA9 and PA10,
- Activation pattern pins has to be controlled by the Host
- No Cyber Security support

https://www.infineon.com/cms/en/tools/aurix-tools/free-tools/infineon/

How to Start for the Software Bootloader?

Usecases

- Bootloader code should be as minimized as
- Bootloader code should be initialized only with necessary peripherals
- A communication interface for the bootloader should be carefully chosen:
  - If the speed is not crucial a low cost USART/SPI/I2C interfaces are a good option
  - If a high speed and high data throughput are needed the USB/CAN/Ethernet are better options
- MCU datasheet, reference and programmer manuals study

MCU FLASH foreseen for the Bootloader

STM32WB55RG

Lets put the Bootloader at the beginning of the FLASH, this allows starting before the Application, making firmware update if available and than running the main program.

STM32WB55RG MCU has 1 MB of FLASH, 512 kB for Application Core (Cortex M4) and 512 kB for Radio Core (Cortex M0)

Lets start with linker script for the Bootloader:

```
/* Specify the memory areas */
MEMORY{
        FLASH (rx)          : ORIGIN = 0x08000000, LENGTH = 32K
        RAM1 (xrw)          : ORIGIN = 0x20000004, LENGTH = 0x2FFFC
        RAM_SHARED (xrw)    : ORIGIN = 0x20030000, LENGTH = 10K
}
```

Lets arrange for the Bootloader 32 kB (first full 8 pages).

Table 3. Flash memory - Single bank organization

Area	Addresses	Size (bytes)	Name
Main memory(1)	0x0800 0000 - 0x0800 0FFF	4 K	Page 0
	0x0800 1000 - 0x0800 1FFF	4 K	Page 1
	0x0800 2000 - 0x0800 2FFF	4 K	Page 2
	. . .	. . .	. . .
	0x0807 E000 - 0x0807 EFFF	4 K	Page 126
	0x0807 F000 - 0x0807 FFFF	4 K	Page 127
	0x0808 0000 - 0x0808 0FFF	4 K	Page 128
	. . .	. . .	. . .
	0x080F E000 - 0x080F EFFF	4 K	Page 254
	0x080F F000 - 0x080F FFFF	4 K	Page 255
Information block	0x1FFF 0000 - 0x1FFF 6FFF	28 K	System memory
	0x1FFF 7000 - 0x1FFF 73FF	1 K	OTP area
	0x1FFF 8000 - 0x1FFF 807F	128	Option bytes

Source: rm0434-multiprotocol-wireless-32bit-mcu-armbased-cortexm4-with-fpu-bluetooth-lowenergy-and-802154-radio-solution-stmicroelectronics.pdf

MCU FLASH foreseen for the Application

STM32WB55RG

Since we decided for the Bootloader 32 kB (32 * 1024 = 32768 = 0x8000) and our memory is 512 kB, for the Application we have 512 kB – 32 kB = 480 kB. The FLASH memory ORIGIN should be added in this case with 0x8000 and its LENGTH should be decreased to 480 kB.

```
MEMORY
{
	FLASH (rx) : ORIGIN = 0x08008000, LENGTH = 480K
	RAM1 (xrw) : ORIGIN = 0x20000004, LENGTH = 0x2FFFC
	RAM_SHARED (xrw) : ORIGIN = 0x20030000, LENGTH = 10K
}
```

P_NUCLEO_WB55_APPLICATION.elf - /P_NUCLEO_WB55_APPLICATION/Debug - Oct 6, 2023, 5:46:54 PM

Memory Regions | Memory Details

Region	Start address	End address	Size	Free	Used	Usage (%)
FLASH	0x08008000	0x0807ffff	480 KB	467,16 KB	12,84 KB	2.68%
RAM1	0x20000004	0x2002ffff	192 KB	189,89 KB	2,11 KB	1.10%
RAM_SHAR...	0x20030000	0x200327ff	10 KB	10 KB	0 B	0.00%

BOOT Mode Selection

STM32WB55RG

When MCU starts it is choosing the boot mode. The BOOTn pins are commonly used for that purpose. The BOOTn pins values are sampled after powering MCU or exiting Standby mode and a proper booting is activating.

Table 2. Boot modes

nBOOT1 FLASH_OPTR[23]	nBOOT0 FLASH_OPTR[27]	BOOT0 pin PH3	nSWBOOT0 FLASH_OPTR[26]	Main flash empty(1)	Boot memory space alias
x	x	0	1	0	Main flash memory is selected as boot area
x	x	0	1	1	System memory is selected as boot area
x	1	x	0	x	Main flash memory is selected as boot area
0	x	1	1	x	Embedded SRAM1 is selected as boot area
0	0	x	0	x	
1	x	1	1	x	System memory is selected as boot area
1	0	x	0	x	

Source: rm0434-multiprotocol-wireless-32bit-mcu-armbased-cortexm4-with-fpu-bluetooth-lowenergy-and-802154-radio-solution-stmicroelectronics.pdf

Memory Mapping vs. Boot mode/Physical remap

STM32WB55RG

- Boot from main flash memory: the main flash memory is aliased in the CPU1 boot memory space at address 0x0000 0000, and is accessible even from its physical address 0x0800 0000. In other words, the flash memory content can be accessed starting from address 0x0000 0000 or 0x0800 0000.
- Boot from system flash memory: the system flash memory is aliased in the CPU1 boot memory space at address 0x0000 0000, and is also still accessible from its physical address 0x1FFF 0000.
- Boot from SRAM: the memory is aliased in the CPU1 boot memory space at address 0x0000 0000, and is accessible even from its physical address 0x2000 0000.

Source: rm0434-multiprotocol-wireless-32bit-mcu-armbased-cortexm4-with-fpu-bluetooth-lowenergy-and-802154-radio-solution-stmicroelectronics.pdf

The above mapping and boot remap shows that starting memory for chosen boot configuration is always at address 0x00000000 (It can be always accessed from its original address).

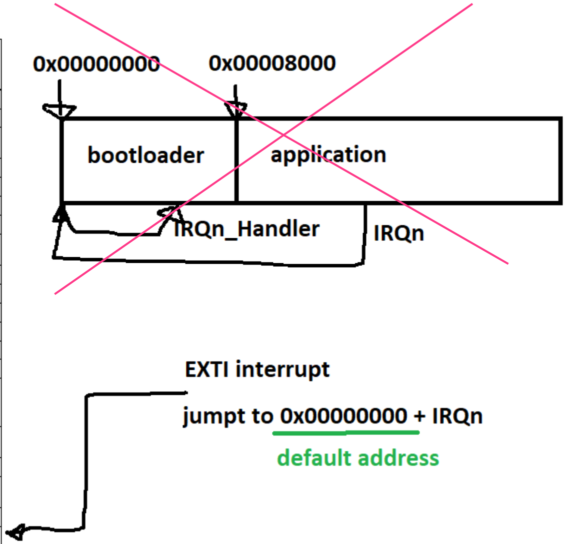
Interrupt Vector Table offset

STM32WB55RG

If we forgot to shift our vector table then our shifted application code when interrupt, it will jump to empty region and Hardware fault will be generated

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000 0000
-	-3	Fixed	Reset	Reset	0x0000 0004
-	-2	Fixed	NMI	Non maskable interrupt HSE CSS, Flash ECC, and SRAM2 parity	0x0000 0008
-	-1	Fixed	HardFault	All classes of fault	0x0000 000C
-	0	Settable	MemManager	Memory manager	0x0000 0010
-	1	Settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
-	2	Settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
-	-	-	-	Reserved	0x0000 001C 0x0000 0028
-	3	Settable	SVCall	System service call via SWI instruction	0x0000 002C
-	4	Settable	Debug	Debug monitor	0x0000 0030
-	-	-	-	Reserved	0x0000 0034
-	5	Settable	PendSV	Pendable request for system service	0x0000 0038
-	6	Settable	Systick	System tick timer	0x0000 003C
0	7	Settable	WWDG	Window watchdog early wakeup	0x0000 0040
1	8	Settable	PVD, PVM1, PVM3	PVD through EXTI[16] (C1IMR2[20]) PVM1 through EXTI[31] (C1IMR2[16]) PVM3 through EXTI[33] (C1IMR2[18])	0x0000 0044
2	9	Settable	TAMP, RTC_STAMP, LSE_CSS	Tamper, TimeStamp, LSECSS interrupt through EXTI[18]	0x0000 0048
3	10	Settable	RTC_WKUP	RTC wakeup interrupt through EXTI[19]	0x0000 004C
4	11	Settable	Flash	Flash memory global interrupt and Flash memory ECC single error interrupt	0x0000 0050
5	12	Settable	RCC	RCC global interrupt	0x0000 0054
6	13	Settable	EXTI0	EXTI line 0 interrupt through EXTI[0]	0x0000 0058
7	14	Settable	EXTI1	EXTI line 1 interrupt through EXTI[1]	0x0000 005C
8	15	Settable	EXTI2	EXTI line 2 interrupt through EXTI[2]	0x0000 0060
9	16	Settable	EXTI3	EXTI line 3 interrupt through EXTI[3]	0x0000 0064
10	17	Settable	EXTI4	EXTI line 4 interrupt through EXTI[4]	0x0000 0068
11	18	Settable	DMA1_CH1	DMA1 channel 1 interrupt	0x0000 006C

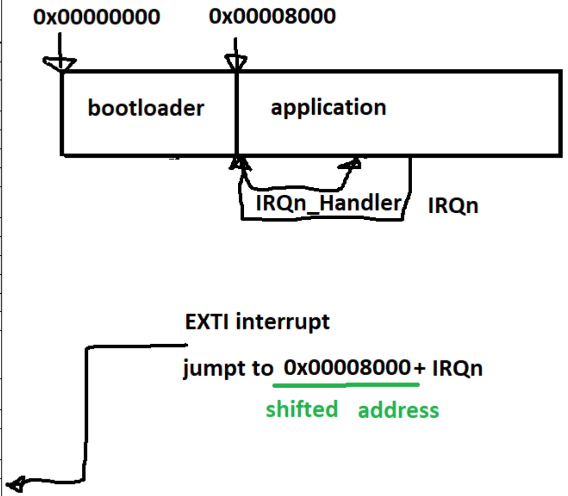
Interrupt Vector Table



Source: rm0434-multiprotocol-wireless-32bit-mcu-armbased-cortexm4-with-fpu-bluetooth-lowenergy-and-802154-radio-solution-stmicroelectronics.pdf

Position	Priority	Type of priority	Acronym	Description	Address
-	-	-	-	Reserved	0x0000 0000
-	-3	Fixed	Reset	Reset	0x0000 0004
-	-2	Fixed	NMI	Non maskable interrupt HSE CSS, Flash ECC, and SRAM2 parity	0x0000 0008
-	-1	Fixed	HardFault	All classes of fault	0x0000 000C
-	0	Settable	MemManager	Memory manager	0x0000 0010
-	1	Settable	BusFault	Pre-fetch fault, memory access fault	0x0000 0014
-	2	Settable	UsageFault	Undefined instruction or illegal state	0x0000 0018
-	-	-	-	Reserved	0x0000 001C 0x0000 0028
-	3	Settable	SVCall	System service call via SWI instruction	0x0000 002C
-	4	Settable	Debug	Debug monitor	0x0000 0030
-	-	-	-	Reserved	0x0000 0034
-	5	Settable	PendSV	Pendable request for system service	0x0000 0038
-	6	Settable	Systick	System tick timer	0x0000 003C
0	7	Settable	WWDG	Window watchdog early wakeup	0x0000 0040
1	8	Settable	PVD, PVM1, PVM3	PVD through EXTI[16] (C1IMR2[20]) PVM1 through EXTI[31] (C1IMR2[16]) PVM3 through EXTI[33] (C1IMR2[18])	0x0000 0044
2	9	Settable	TAMP, RTC_STAMP, LSE_CSS	Tamper, TimeStamp, LSECSS interrupt through EXTI[18]	0x0000 0048
3	10	Settable	RTC_WKUP	RTC wakeup interrupt through EXTI[19]	0x0000 004C
4	11	Settable	Flash	Flash memory global interrupt and Flash memory ECC single error interrupt	0x0000 0050
5	12	Settable	RCC	RCC global interrupt	0x0000 0054
6	13	Settable	EXTI0	EXTI line 0 interrupt through EXTI[0]	0x0000 0058
7	14	Settable	EXTI1	EXTI line 1 interrupt through EXTI[1]	0x0000 005C
8	15	Settable	EXTI2	EXTI line 2 interrupt through EXTI[2]	0x0000 0060
9	16	Settable	EXTI3	EXTI line 3 interrupt through EXTI[3]	0x0000 0064
10	17	Settable	EXTI4	EXTI line 4 interrupt through EXTI[4]	0x0000 0068
11	18	Settable	DMA1_CH1	DMA1 channel 1 interrupt	0x0000 006C

Interrupt Vector Table



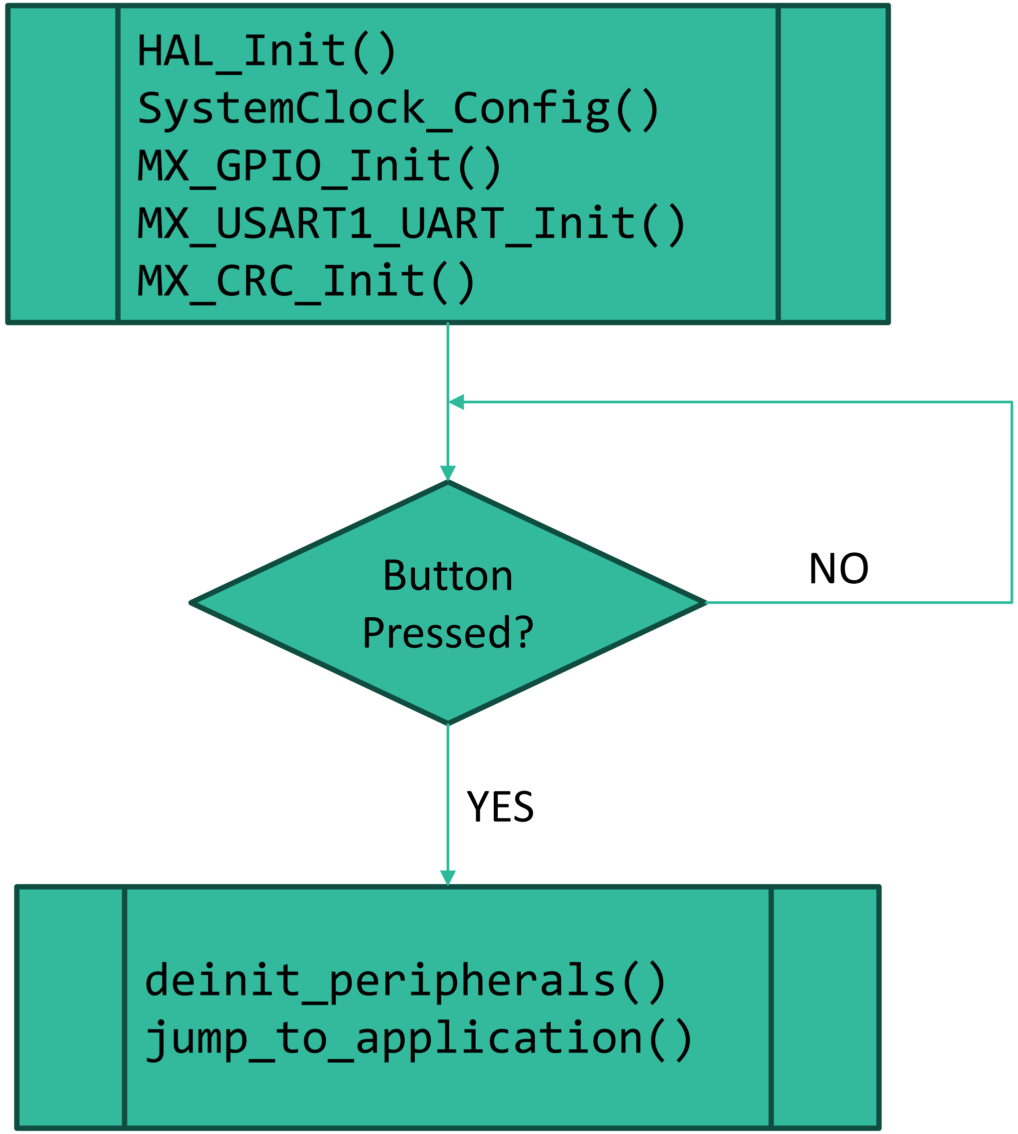

```
#define VECT_TAB_OFFSET 0x00008000U /*!< Vector
Table base offset field.
This value must be a multiple of 0x200. */
```

```
#if defined(USER_VECT_TAB_ADDRESS)
/* Configure the Vector Table location add
offset address ------------------*/
SCB->VTOR = VECT_TAB_BASE_ADDRESS |
VECT_TAB_OFFSET;
#endif
```

https://gitlab.com/pawelbanas/sii-mentor-mentee/-/tree/main/WB55_bootloader?ref_type=heads

Bootloader Template

Skeleton



Bootloader Template

Peripherals Deinitialization

```
void deinit_peripherals() {
HAL_CRC_DeInit(&hcrc);
HAL_UART_DeInit(&huart1);
HAL_NVIC_DisableIRQ(B1_EXTI_IRQn);
HAL_GPIO_DeInit(LD2_GPIO_Port, LD2_Pin);
HAL_GPIO_DeInit(B1_GPIO_Port, B1_Pin);
HAL_RCC_DeInit(); // We're using LL RCC, so
we'll use this function
HAL_DeInit();


// SysTick Reset
SysTick->CTRL = 0;
SysTick->LOAD = 0;
SysTick->VAL = 0;
}
```

https://gitlab.com/pawelbanas/sii-mentor-mentee/-/tree/main/WB55_bootloader?ref_type=heads

Bootloader

Jump to Application

```
typedef void (*jumpFunction)(); // helper-typedef
uint32_t const jumpAddress = *(__IO uint32_t*) (cmd->app_flash_start_addr + 4); //
Address of application's Reset Handler
jumpFunction runApplication = (jumpFunction) jumpAddress; // Function we'll use to jump
to application


deinit_peripherals(); // Deinitialization of peripherals and systick
__set_MSP(*((__IO uint32_t*) cmd->app_flash_start_addr)); // Stack pointer setup
runApplication(); // Jump to application
```

https://gitlab.com/pawelbanas/sii-mentor-mentee/-/tree/main/WB55_bootloader?ref_type=heads

Bootloader Api

Structure

Bootloader.c

```
get_command()
responsd_ok()
respond_error()
erase_application()
flash_and_verify()
receive_and_flash_firmware()
verify_firmware()
```

Bootloader.h

```
BootloaderInit()
BootloaderTask()
```

Bootloader Api

Commands

Operation Code	Description	Data Timeout
0x01	Echo – bootloader should responde with ok!	None
0x02	Binary file size setup	Binary file size
0x03	Firmware update mode entry	None
0x04	Check CRC of Application	CRC for comparison
0x05	Jump to application	None

```
typedef enum BootloaderOpcode_t
{
        BOOTLOADER_CMD_INVALID = 0x00,
        BOOTLOADER_CMD_ECHO = 0x01,
        BOOTLOADER_CMD_SETSIZE = 0x02,
        BOOTLOADER_CMD_UPDATE = 0x03,
        BOOTLOADER_CMD_CHECK = 0x04,
        BOOTLOADER_CMD_JUMP = 0x05
} BootloaderOpcode;
```

https://gitlab.com/pawelbanas/sii-mentor-mentee/-/tree/main/WB55_bootloader?ref_type=heads

Bootloader Api

Receive and Flash

FLASH memory is blocked by default against any write. In order to unlock access we have to write to FLASH control registers a dedicated value.

```
static bool erase_application(unsigned firstPage, unsigned lastPage)
{
…
            if (HAL_FLASHEx_Erase(&eraseConfig, &PageError) != HAL_OK)
            {
                        …
            }
…
…
```

```
static bool flash_and_verify(BootloaderCommand* cmd, uint8_t const*
const bytes, size_t const amount, uint32_t const offset)
{
…
            if (HAL_FLASH_Program(FLASH_TYPEPROGRAM_DOUBLEWORD,
            programmingAddress, programmingData) != HAL_OK)
            {
                        …
            }
```

```
static bool receive_and_flash_firmware(BootloaderCommand* cmd, uint32_t const firmwareSize)
{
…
            if (HAL_FLASH_Unlock() != HAL_OK)
            {
                        …
            }
            if (!erase_application(GetPage(cmd->app_flash_start_addr), GetPage(cmd->app_flash_stop_addr)))
            {
                        …
            }
            if (!flash_and_verify(cmd, cmd->bootloaderBuffer, bytesToReceive, bytesProgrammed))
            {
                        …
            }
```

https://gitlab.com/pawelbanas/sii-mentor-mentee/-/tree/main/WB55_bootloader?ref_type=heads

Bootloader Api

CRC calculation

In order to calculate CRC we will use built-in block CRC. The hardware CRC block is calculating CRC from 32-bit word and operate only little endian.

```
static bool verify_firmware(BootloaderCommand* cmd, uint32_t const firmwareSize)
{
        // sanity check - fail loudly if no application size is set
        if (firmwareSize == 0) {
        return false;
}

uint32_t const calculatedChecksum = HAL_CRC_Calculate(&hcrc, (uint32_t*) cmd->app_flash_start_addr,
firmwareSize / 4);


        return (calculatedChecksum == cmd->data);
}
```

https://gitlab.com/pawelbanas/sii-mentor-mentee/-/tree/main/WB55_bootloader?ref_type=heads

Programming Script

Python

We will write our programming script using python3. We will use standard packages such serial, argparse, os, math as well as our custom crc32 calculation implementation script

```
import serial
import argparse
import os
import math
import stm32_crc
from enum import IntEnum
```

```
def __init__(self, poly_value=0x4C11DB7,
init_value=0xFFFFFFFF):
```

```
def _calc32(self, previous, word):
   crc = previous ^ word
   for i in range(32):
       if crc & 0x80000000:
           crc = ((crc << 1) & 0xFFFFFFFF) ^ self._poly
       else:
           crc = (crc << 1) & 0xFFFFFFFF
   return crc
```

https://gitlab.com/pawelbanas/sii-mentor-mentee/-/tree/main/WB55_bootloader?ref_type=heads

Programming Script

File Selection

We will create a simple interface for input arguments like USART port nr or Programming file name

```
parser = argparse.ArgumentParser(
        description='Update STM32 microcontroller firmware via UART')
parser.add_argument('update_file', type=str,
                        help='Path to compiled firmware file (*.bin)')
parser.add_argument('com_port', type=str,
                        help='COM port of the microcontroller')
args = parser.parse_args()
```

```
if not os.path.isfile(args.update_file):
        print(f'File {args.update_file} does not exist, exiting...')
        exit(1)

    print(f'Updating from {args.update_file}')
```

https://gitlab.com/pawelbanas/sii-mentor-mentee/-/tree/main/WB55_bootloader?ref_type=heads

Programming Script

Communication with device

```
try:
        stm_uart = serial.Serial(
            port=args.com_port, baudrate=921600, timeout=20)
    except serial.SerialException as ex:
        print(
            f'Cannot open STM32 port: {ex}. Make sure it\'s not opened by another
application!')
        exit(2)

    if not stm_uart.isOpen():
        print('Couldn\'t open STM32 UART port! Exiting...')
        exit(2)
```

https://gitlab.com/pawelbanas/sii-mentor-mentee/-/tree/main/WB55_bootloader?ref_type=heads

Programming Script

API Implementation

```
class BootloaderCommand(IntEnum):
    INVALID = 0x00,
    ECHO = 0x01,
    SETSIZE = 0x02,
    UPDATE = 0x03,
    CHECK = 0x04,
    JUMP = 0x05
def create_command(command: BootloaderCommand, data: int = 0) -> bytes:
    return bytes([command]) + data.to_bytes(4, 'big')

def send_command(command: bytes, port: serial.Serial) -> bool:
    port.write(command)
    stm_response = port.read(3)

    return stm_response == BOOTLOADER_OK

def is_bootloader_running(port: serial.Serial) -> bool:
    return send_command(create_command(BootloaderCommand.ECHO), port)

def send_firmware_size(port: serial.Serial, size: int) -> bool:
    return send_command(create_command(BootloaderCommand.SETSIZE, size), port)
```

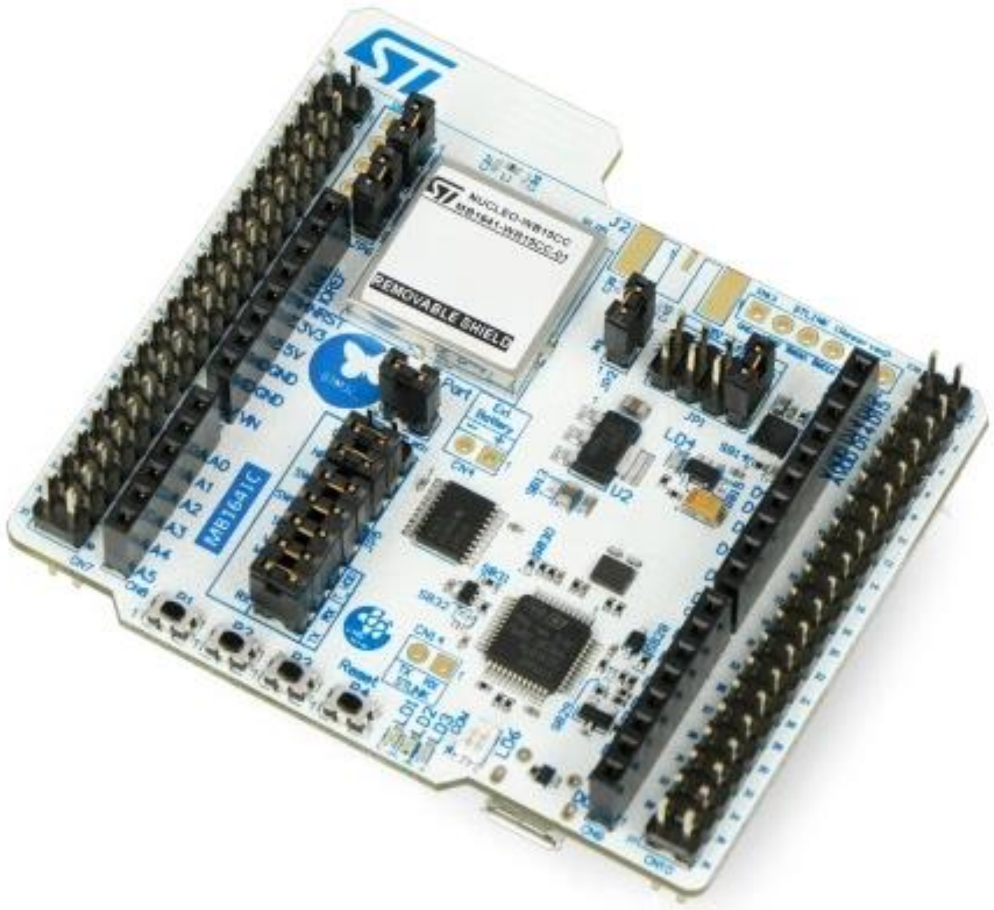

https://gitlab.com/pawelbanas/sii-mentor-mentee/-/tree/main/WB55_bootloader?ref_type=heads

Programming Script

Firmware flashing

```
def flash_firmware(port: serial.Serial, firmware: bytes, packet_size: int = 1024) -> bool:
    if not send_command(create_command(BootloaderCommand.UPDATE), port):
        print('Cannot enter into update mode!')
        return False

    bytes_sent = 0
    firmware_size = len(firmware)
    packets_amount = math.ceil(firmware_size / packet_size)
    packets_sent = 0

    while bytes_sent < firmware_size:
        # calculate next packet length
        bytes_left = firmware_size - bytes_sent
        next_packet_length = packet_size if bytes_left > packet_size else bytes_left
        firmware_slice = firmware[bytes_sent:(bytes_sent + next_packet_length)]

        print(f'Sending {len(firmware_slice)} bytes')
        port.write(firmware_slice)
        stm_response = port.read(3)

        if stm_response != BOOTLOADER_OK:
            print('Bootloader did not respond or returned an error while programming!')
            return False
        else:
            packets_sent += 1
            print(f'Progress: {packets_sent}/{packets_amount}')

        bytes_sent += next_packet_length

    final_response = port.read(3)
    return final_response == BOOTLOADER_OK
```

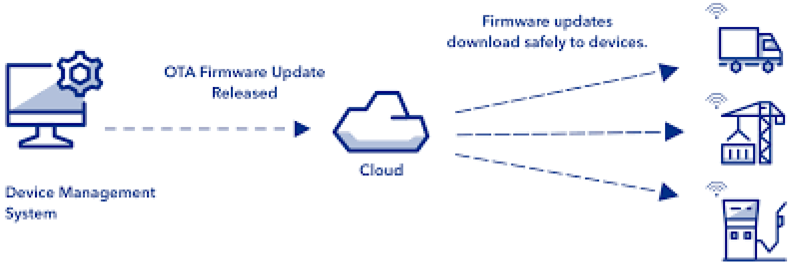
Live Demo



https://www.st.com/en/microcontrollers-microprocessors/stm32wb55rg.html

Summary & Conclusions

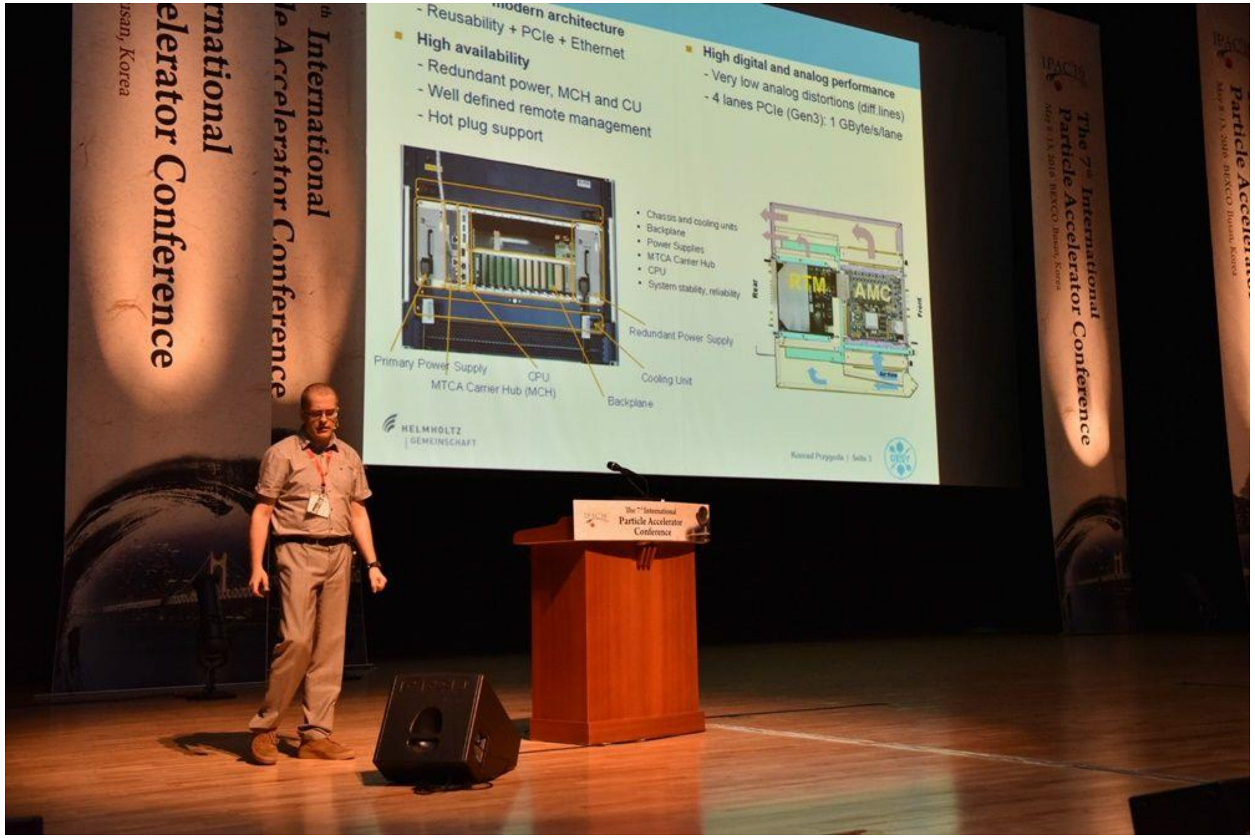
- Software bootloader can be an alternative option to embedded bootloaders
- Software bootloader has to be organized and build with steps
- Software bootloader can keep alive ready PCB projects with a single made mistake
- Software bootloader can be extended with dual boot
- Software bootloader can be a part of OTA/SOTA mechanisms
- Software bootloader can be a part of Cyber Security such
- Software bootloader can be easy integrated into operating systems



https://www.avsystem.com/blog/iot/sota-software-over-the-air/

Q&A

Thank You
for Your attention!



kprzygoda@sii.pl