life.augmented

STM32H5 - I3C Interface

ST Restricted

Agenda

1 I3C overview

2 I3C new features

3 Compatibility with I^{2}C

4 Why to use I3C and what might be challenging?

5 Hands-On

life.augmented

Quick introduction to I3C

life.augmented

I3C overview

- I3C is a MIPI standardized protocol designed to overcome I^{2}C limitations
  (limited speed, external signals needed for interrupts, no automatic detection of the devices connected to the bus...) while improving power-efficiency
- I3C = “Improved I^{2}C”

I3C overview

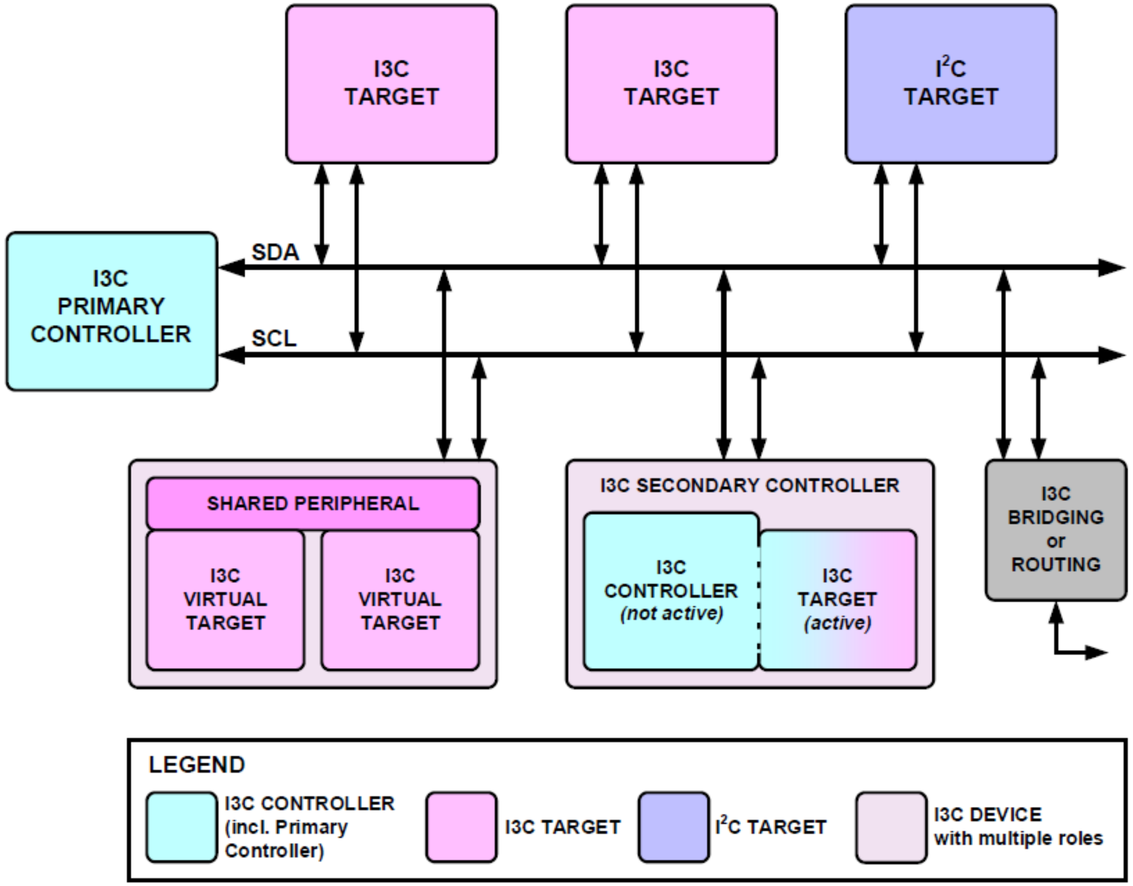


Figure 10 I3C Bus with I²C Devices and I3C Devices

- Two wire interface (SDA/SCL)
- New naming:
  - Master => Controller
  - Slave => Target
- “Compatible with I^{2}C”
- Defined by MIPI specification
  - I3C basic specification available for free
  - Might be more accessible than I^{2}C spec
- Used by JEDEC for DDR5 sideband

I3C new features (1/4)
Switching to push-pull drive

- I2C speed is limited by open-drain bus
  - You can reduce pull-up value and increase GPIO drive capability,
    but this starts to be limiting, FM+ (1 MHz) is supported only on selected pins
- I3C uses open-drain only during arbitration header
  - Then it switches to push-pull
  - Maximum according to spec is around 4.16 MHz
    - Depends on bus capacitance
- This allows:
  - Increase communication speed up to 12.5 MHz (faster than full-speed USB)
  - Lower power consumption
- Pull-up on SDA controlled by I3C primary controller
  - Cca. 1.2 kΩ pull-up integrated in STM32H5
- SCL always driver by controller in push-pull
  - No clock stretching

I3C new features (2/4)
Dynamic address assignment

- Each I3C target must be assigned a dynamic address to be able to communicate
- 7-bit address
- Allows configuring priority between devices
  - Addresses might be changed on-the-fly
  - Lower address => higher priority (similar for CAN)
  - Priority is used for In-band interrupts (IBI)
- Only certain addresses can be assigned to targets
  - Also depends on compatibility with some I^{2}C features
- The process is based on 64-bit unique ID
  - This might not always solve address collision, when having multiple parts of same chip
- Address can be also assigned based on static/I^{2}C legacy address

I3C new features (3/4)
CCC and in band interrupts

- Common command codes (CCC)
  - One type of I3C transfer
  - Standardize bus initialization and some basic commands
  - Separate command code for Broadcast (all devices) and Direct command (specific device)
- In-band interrupt (IBI)
  - Allows any I3C device to send notification to I3C controller
  - Device can generate interrupts without requiring an external signal
  - The interrupt can contain small payload, up to 4 bytes
- Timing control
  - Allows to timestamp data received from sensors
  - Sampling data sent as part of IBI

I3C new features (4/4)
Advanced features

- Group address
  - Allows to target multiple I3C Targets using single address (multicast)
  - Optional feature
    - Multiple group addresses might be assigned to single target
- Hot-join
  - Mechanism that allows target devices to join the bus on-the-fly
  - E.g.: when exiting low power mode and activating additional HW
- Secondary controller
  - Additional devices on the bus can request to be the controller
  - This might be useful in systems where “main” controller can enter low-power mode
    - The bus is maintained by secondary controller (sensor hub) and can wake-up main controller

I^2C compatibility

- The specification declares compatibility with I^2C, but there are several limitations
- Not all I^2C slaves follow I^2C specification
  - E.g.: some of our competitors use two-wire interface
- High-speed I3C communication relies on 50ns spike filter to be implemented in I^2C slave
- Clock stretching is not allowed
- Certain addresses are reserved for I3C (e.g. 0x7E)
- I3C targets might still have I^2C mode => I3C might require specific configuration
- Compatibility with I^2C might not be easy to achieve and might cause significant speed decrease

I^{2}C compatibility

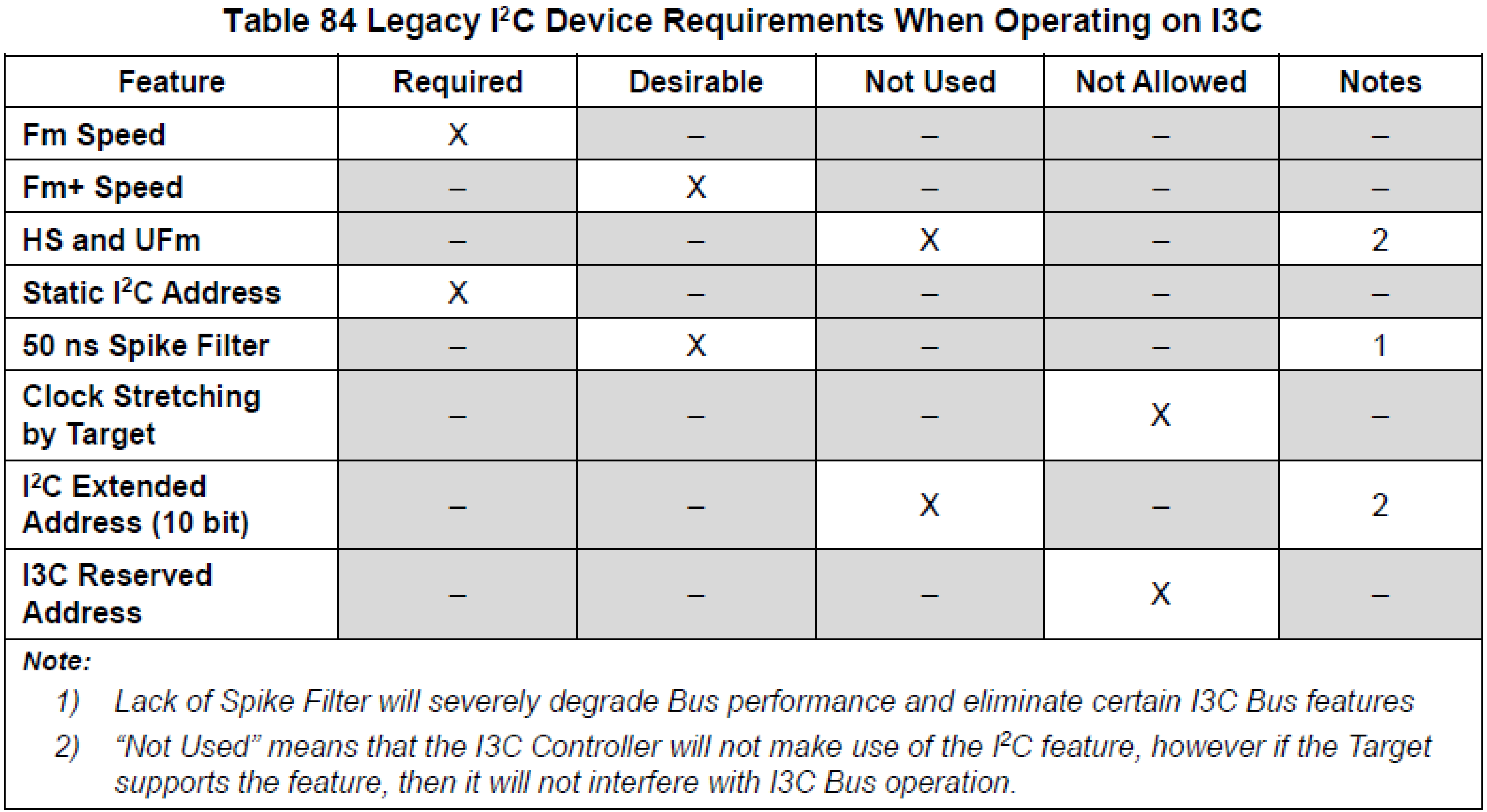


Table 84 Legacy I^{2}C Device Requirements When Operating on I3C

Feature	Required	Desirable	Not Used	Not Allowed	Notes
Fm Speed	X	–	–	–	–
Fm+ Speed	–	X	–	–	–
HS and UFm	–	–	X	–	2
Static I^{2}C Address	X	–	–	–	–
50 ns Spike Filter	–	X	–	–	1
Clock Stretching by Target	–	–	–	X	–
I^{2}C Extended Address (10 bit)	–	–	X	–	2
I3C Reserved Address	–	–	–	X	–

Note:

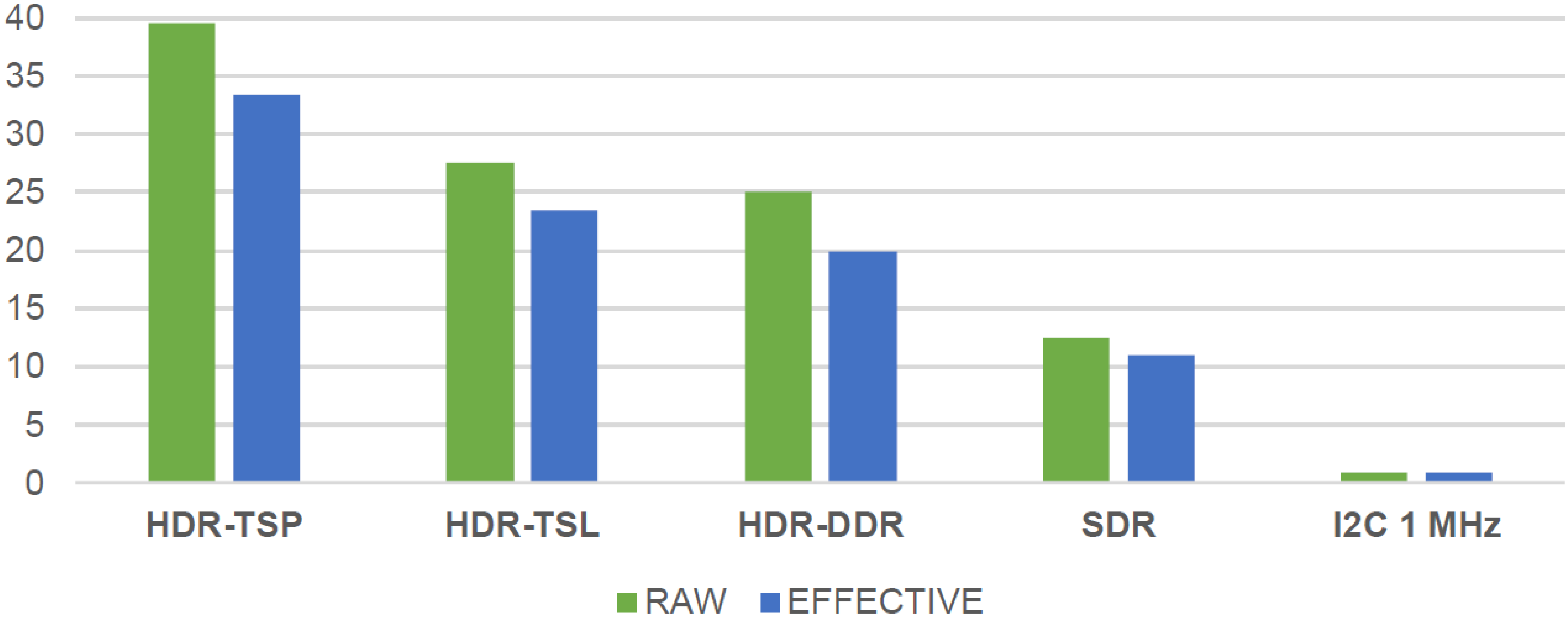
1) *Lack of Spike Filter will severely degrade Bus performance and eliminate certain I3C Bus features*
2) *“Not Used” means that the I3C Controller will not make use of the I^{2}C feature, however if the Target supports the feature, then it will not interfere with I3C Bus operation.*

Source: MIPI I3C Basic specification

Why use I3C?

- Higher data rate
  - 12.5MHz, that is faster than full-speed USB
- Lower pin required
  - In-band interrupt support
  - Possibility to increase bandwidth with multi-lane support for selected targets
- Lower power consumption
  - Thanks to switching to push-pull mode
  - Especially compared to e.g. Fast-mode+ where low pull-ups need to be used
- Standardized commands for common operations
  - Entering low-power modes, device reset etc.
- Higher data rates
  - Multiple lanes and HDR allows increasing bandwidth in the future

I3C speed comparison



Source: MIPI I3C Basic specification

What can be challenging?

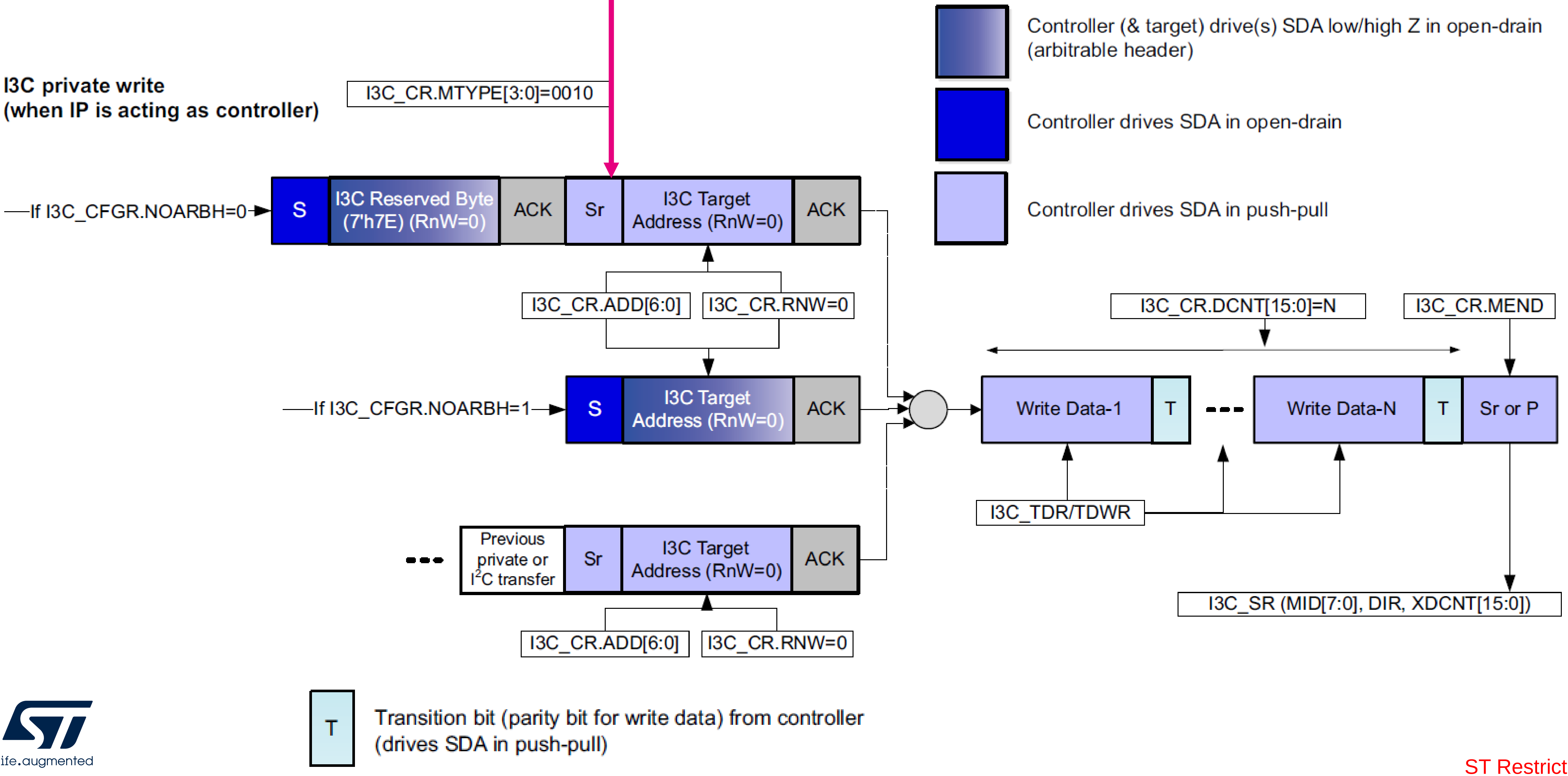
- I^{2}C compatibility depends heavily on the used devices
  - It might be difficult to read for datasheet if the device is compatible
- Higher frequency can lead to issues with signal integrity
  - Higher frequency might work only on shorter connections
  - Bus capacitance can become limiting
  - This might also be an issue if level shifter should be used
- Requires bus initialization during startup
  - This might take some time
- Is more complex compared to I3C
- Low availability at the moment

I3C transfers

- Private read / write commands
  - E.g. reading / writing sensor registers
  - Similar to I^{2}C, but using I3C timing
- Specific I3C CCC commands
  - Dynamic address assignment (ENTDAA)
  - Target reset (RSTACT)
- I3C CCC commands
  - Broadcast write
  - Direct read / write
- In-band interrupt (IBI)
  - Initiated by I3C Target
- I^{2}C legacy transfer
  - Same as private read / write, but with I^{2}C timing
- Hot-join request
  - Specific In-band interrupt
- Controller role request

Private write as Controller

Repeated start
This prevents collision between CCC and target address


I3C private write
(when IP is acting as controller)
I3C_CR.MTYPE[3:0]=0010
Controller (& target) drive(s) SDA low/high Z in open-drain (arbitrable header)
Controller drives SDA in open-drain
Controller drives SDA in push-pull
If I3C_CFGR.NOARBH=0
S
I3C Reserved Byte (7'h7E) (RnW=0)
ACK
Sr
I3C Target Address (RnW=0)
ACK
I3C_CR.ADD[6:0]
I3C_CR.RNW=0
I3C_CR.DCNT[15:0]=N
I3C_CR.MEND
If I3C_CFGR.NOARBH=1
S
I3C Target Address (RnW=0)
ACK
Write Data-1
T
Write Data-N
T
Sr or P
I3C_TDR/TDWR
Previous private or I²C transfer
Sr
I3C Target Address (RnW=0)
ACK
I3C_SR (MID[7:0], DIR, XDCNT[15:0])
I3C_CR.ADD[6:0]
I3C_CR.RNW=0
T
Transition bit (parity bit for write data) from controller (drives SDA in push-pull)

Hands on

Goal

- I3C communication between 2 boards
  - First board will act as Controller
  - Second board will act as Target
- Controller will perform dynamic address assignment
- Target board will send In-band interrupt (IBI) when button pressed
- **Please work in pairs to test the hands-on!**
- Based on I3C_Controller_InBandInterrupt_IT and I3C_Target_InBandInterrupt_IT examples for Nucleo-H503RB

I3C
PRIMARY
CONTROLLER

Part 1: I3C Controller Firmware sequence

- Initialize I3C (generate by MX)
- Start dynamic address assignment
  - Assign address to each new device
- (optional) Send SETMRL command to limit IBI payload to 4-bytes
  - This is limitation of STM32H5 I3C Controller
- Configure / Enable reception of IBI from selected address
  - Otherwise the IBI is not acknowledged by Controller
  - This is just internal operation, no communication on the I3C bus
  - Real sensor will probably require some command or register write to enable
- Receive IBI via I3C interrupt
- Read IBI source address and payload in main loop
  - Toggle LED when IBI received

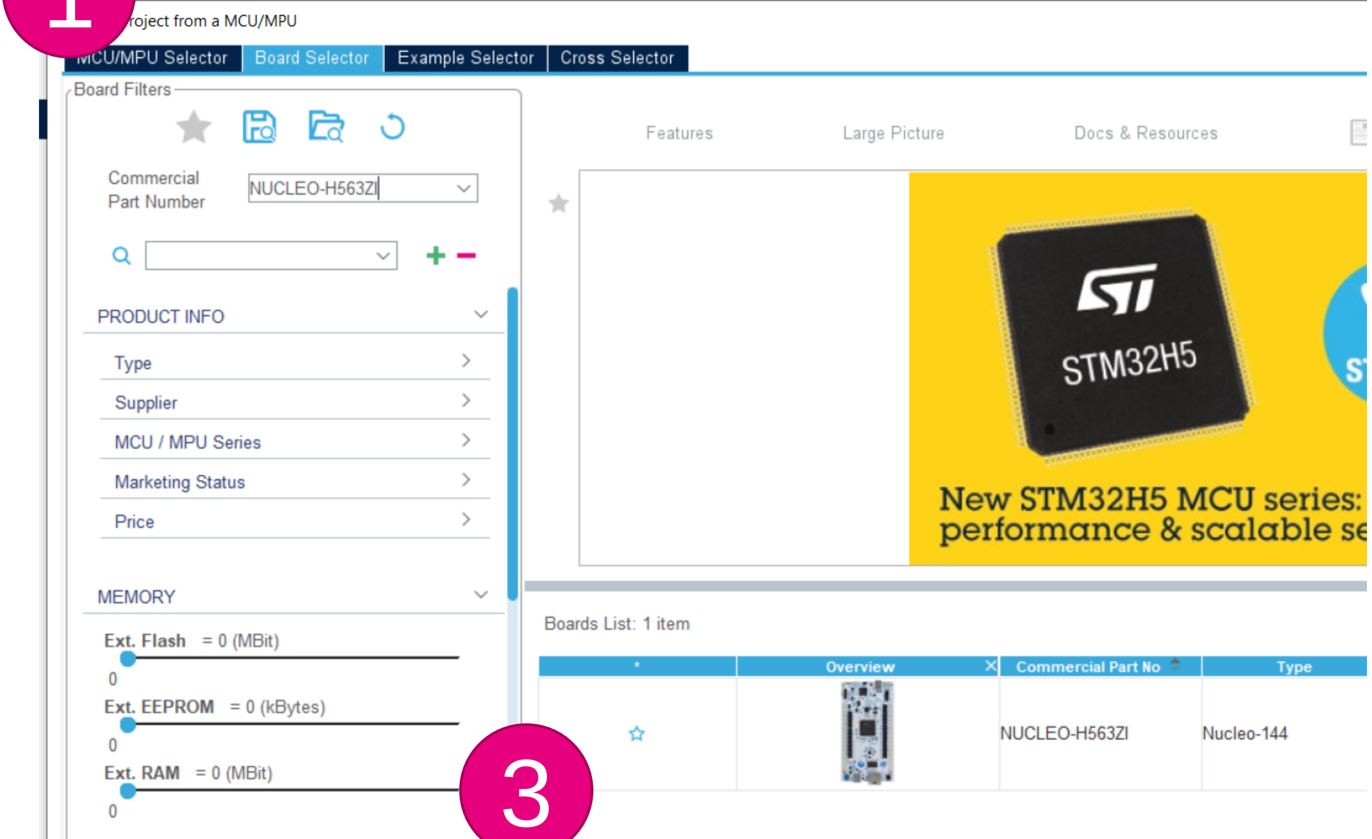
Loop

I3C PRIMARY CONTROLLER

Part 1: I3C Controller Creating CubeMX project

- Start by selecting **NUCLEO-H563ZI board**
  - This will initialize the LEDs and push buttons automatically
- When prompted “Initialize all peripherals in default mode”, select No
- Select project **without TrustZone**

1



Commercial Part Number: NUCLEO-H563ZI

Ext. Flash = 0 (MBit)

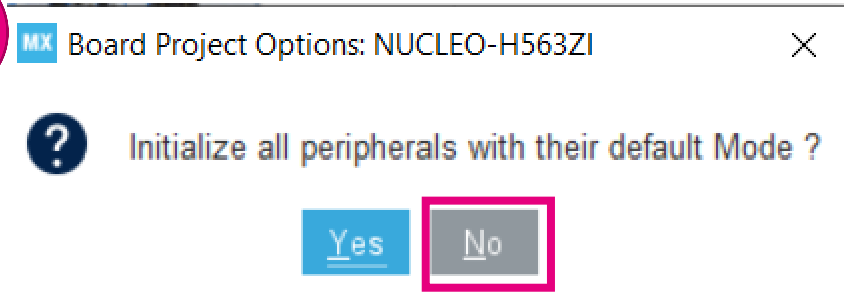
Ext. EEPROM = 0 (kBytes)

Ext. RAM = 0 (MBit)

Boards List: 1 item

*	Overview	Commercial Part No	Type
☆		NUCLEO-H563ZI	Nucleo-144

2

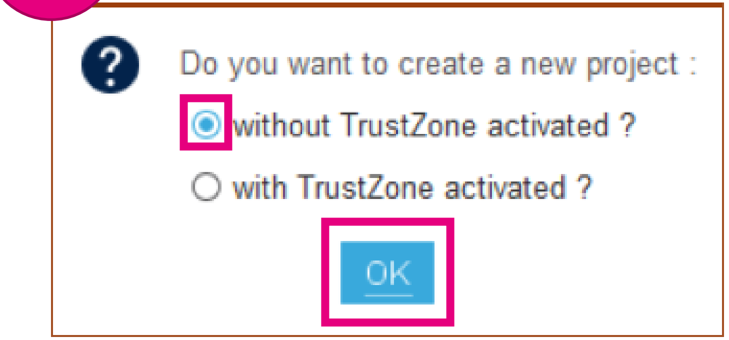


Board Project Options: NUCLEO-H563ZI

Initialize all peripherals with their default Mode ?

Yes [✓ No]

3



Do you want to create a new project :

● without TrustZone activated ?

○ with TrustZone activated ?

[✓ OK]

life.augmented

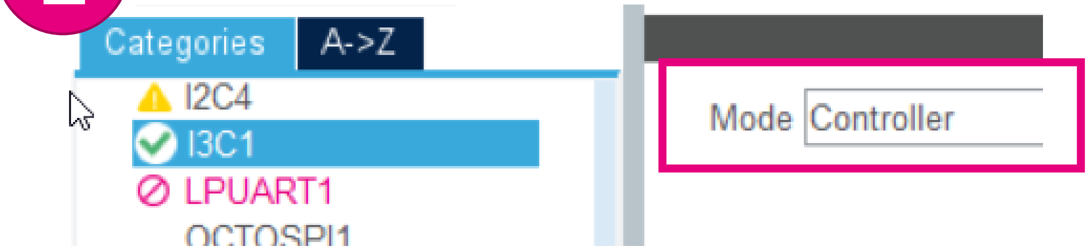
ST Restricted

I3C
PRIMARY
CONTROLLER

Part 1: I3C Controller Configure I3C peripheral

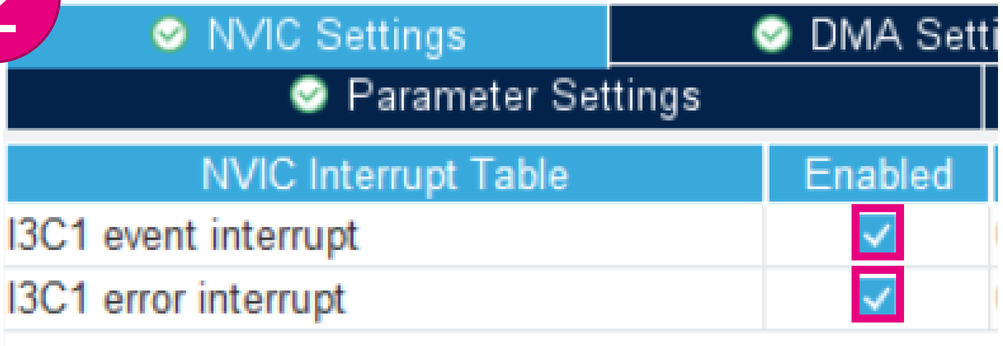
- Enable I3C1 in Controller mode
- Leave default config:
  - I3C pure bus (no I2C legacy devices)
  - Frequency I3C Controller to 12500kHz (12.5MHz)
- Enable event & error interrupts in NVIC
- The default GPIO configuration should be ok: Alternate function push-pull, very high speed, internal pull-up
  - Internal pull-up on GPIO (40kOhms typical) helps with startup issue

1

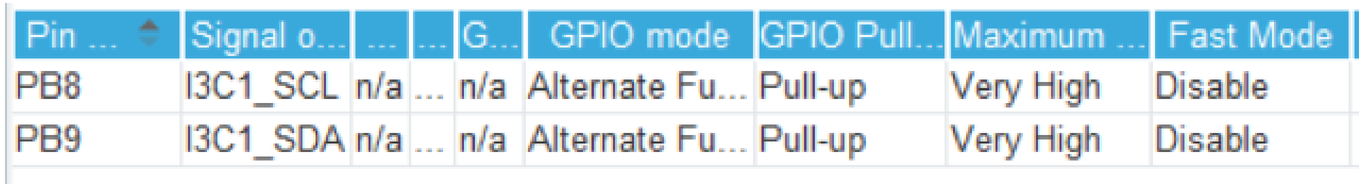


Mode Controller

2



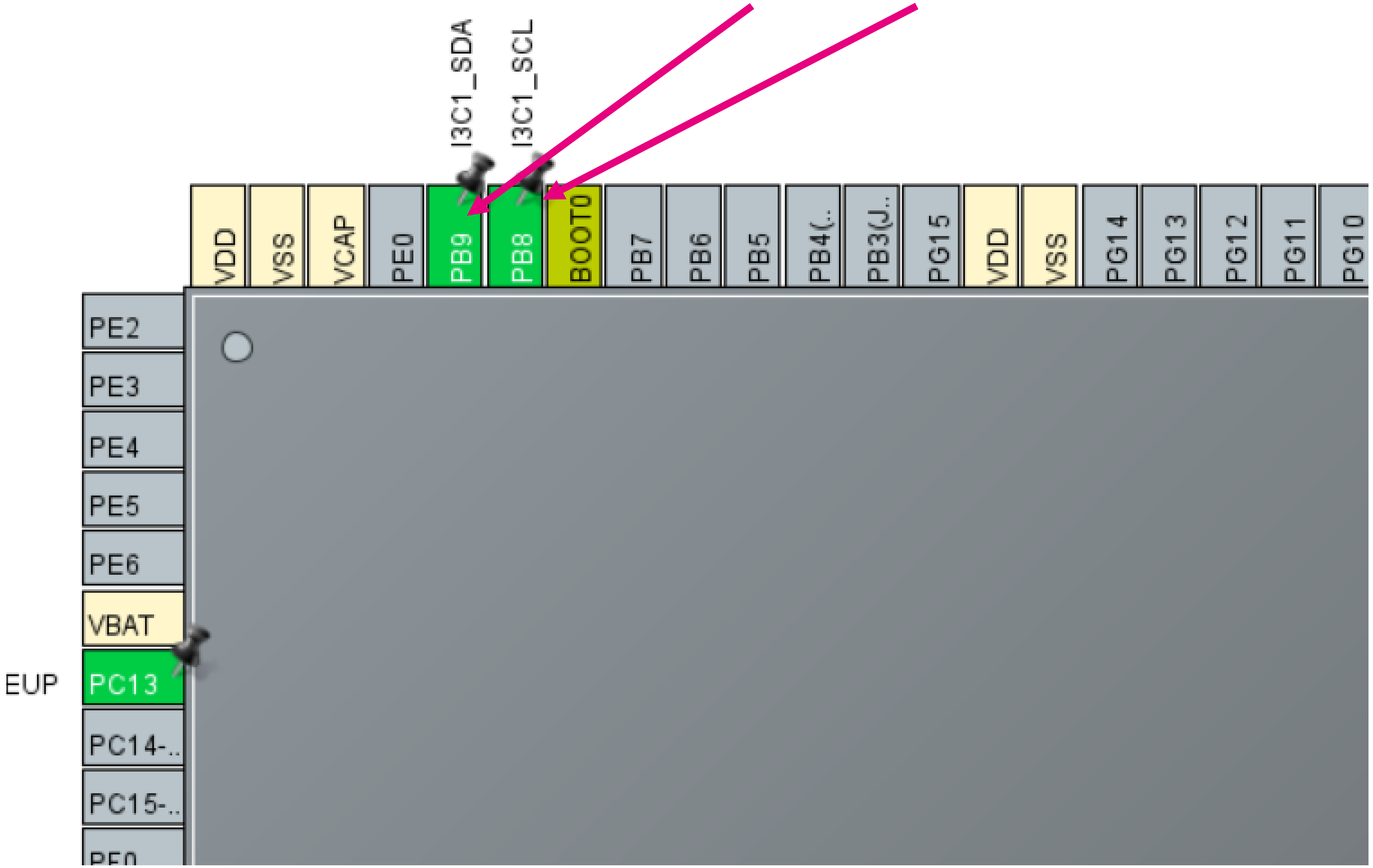
NVIC Interrupt Table	Enabled
I3C1 event interrupt	☑
I3C1 error interrupt	☑



Pin ...	Signal o...	...	...	G...	GPIO mode	GPIO Pull...	Maximum ...	Fast Mode
PB8	I3C1_SCL	n/a	...	n/a	Alternate Fu...	Pull-up	Very High	Disable
PB9	I3C1_SDA	n/a	...	n/a	Alternate Fu...	Pull-up	Very High	Disable

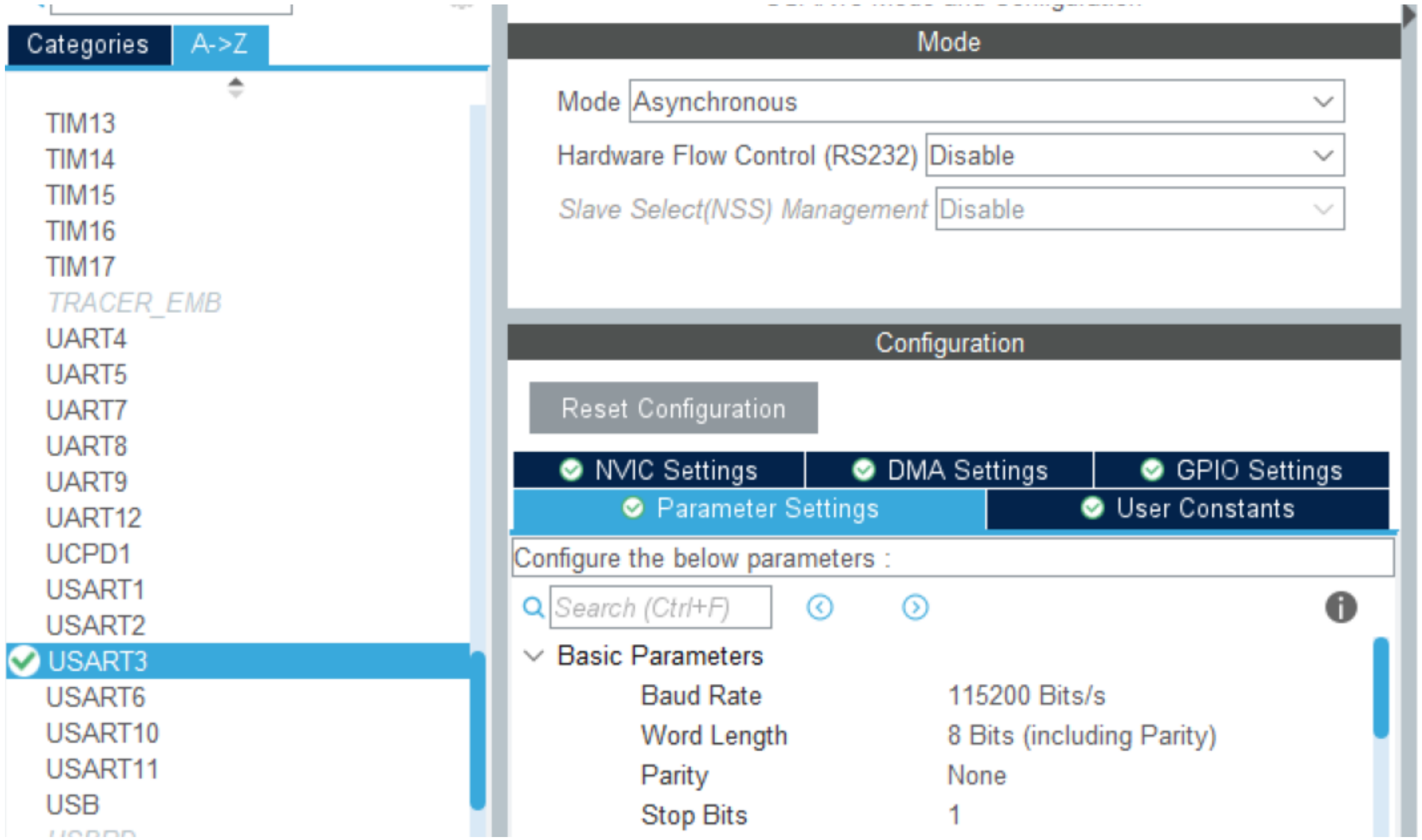
I3C
PRIMARY
CONTROLLER

Part 1: I3C Controller
Move pins to PB8/PB9 (Arduino header)



Part 1: I3C Controller Configure USART3 for debug

- Enable USART3 in Asynchronous mode
- Leave default configuration (115200 baudrate, 8-bits without parity)



Mode: Asynchronous

Hardware Flow Control (RS232): Disable

Slave Select(NSS) Management: Disable

Configure the below parameters :

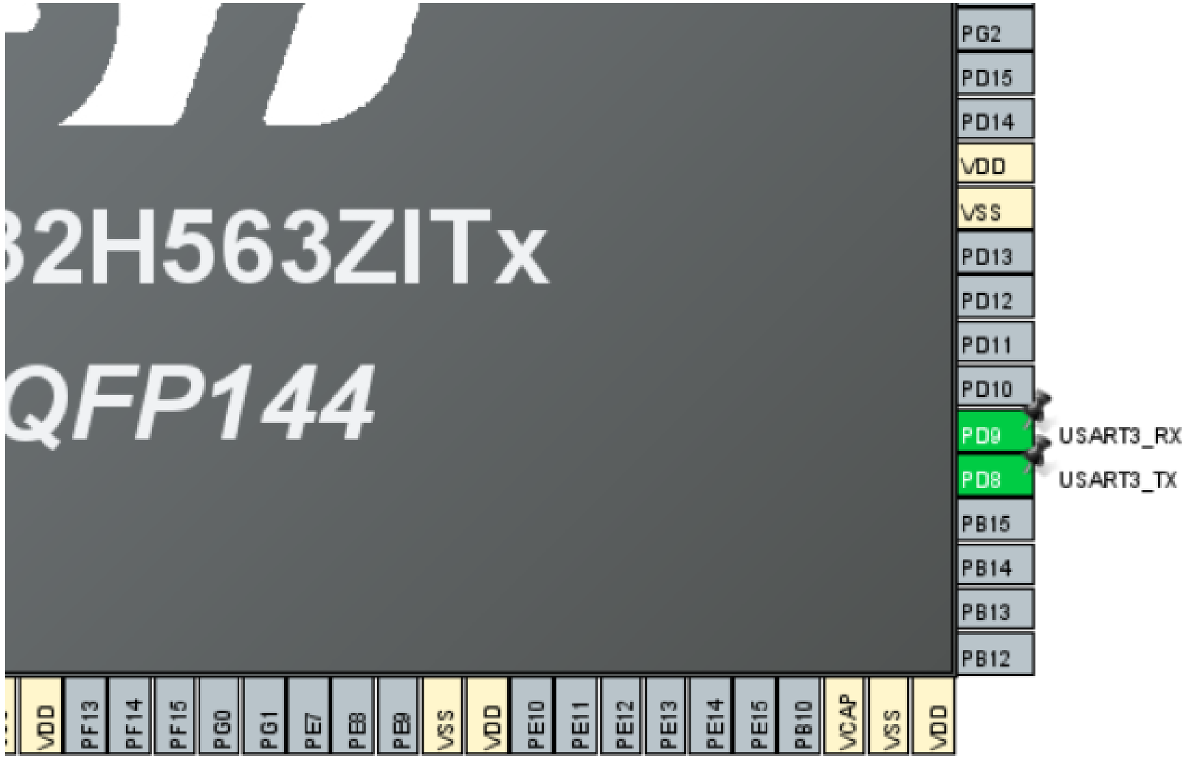
Basic Parameters

Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

I3C PRIMARY CONTROLLER

Part 1: I3C Controller
Configure USART3 for debug

- Enable USART3 in Asynchronous mode
- Leave default configuration (115200 baudrate, 8-bits without parity)

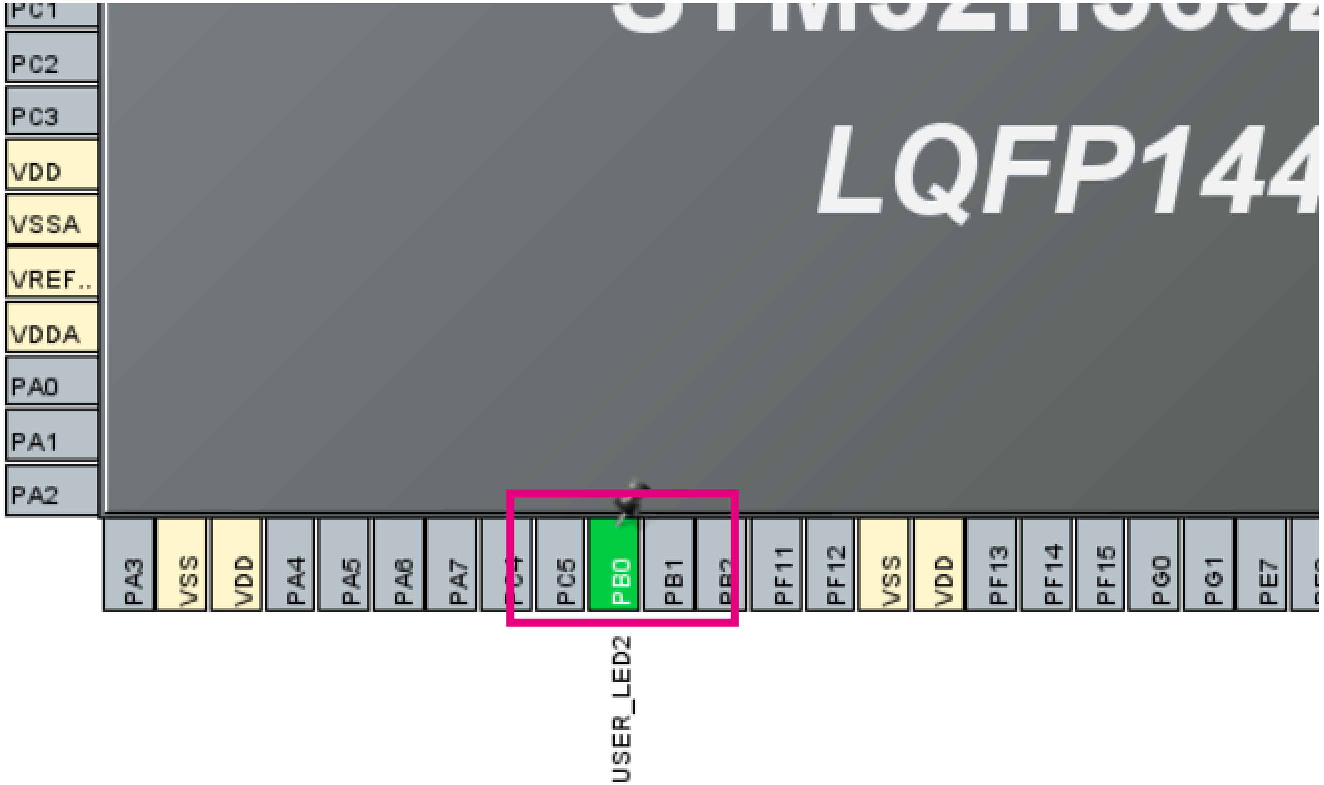


life.augmented

I3C
PRIMARY
CONTROLLER

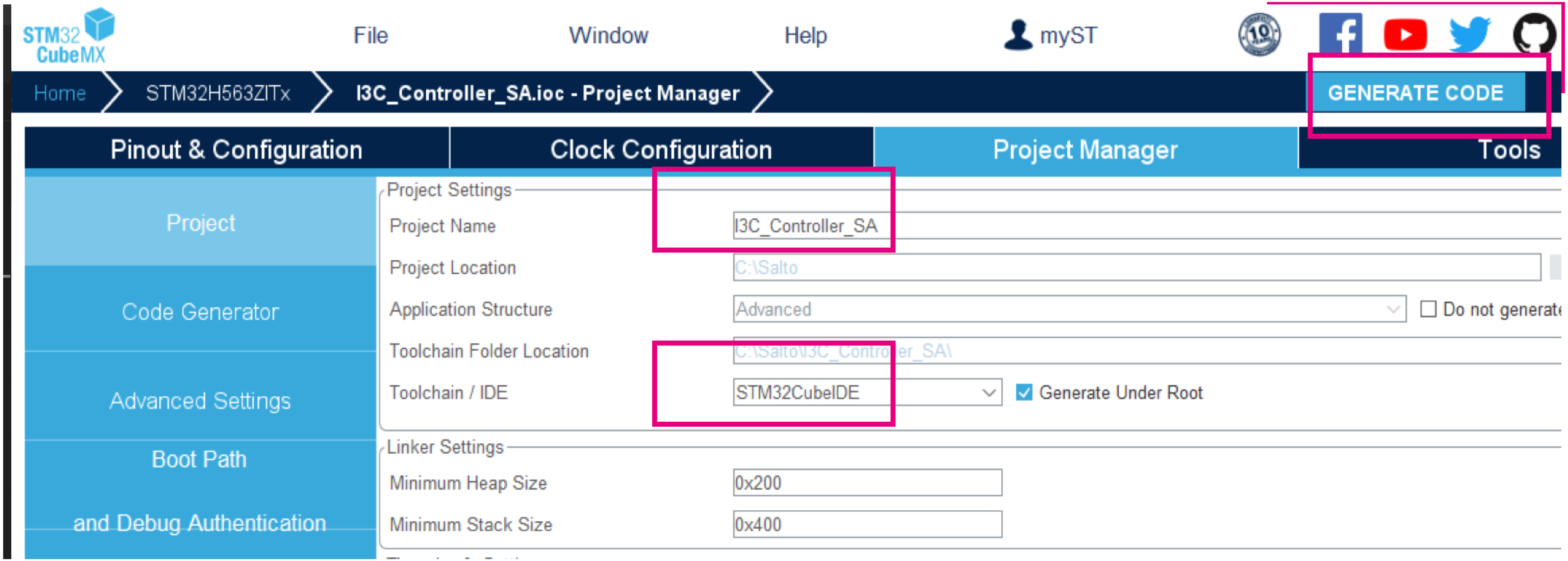
Part 2: I3C Target Configure LED

- Assign PB0 to GPIO output and name it «USER_LED2»



I3C PRIMARY CONTROLLER

Part 2: I3C Controller Generate Code



Project Settings

Project Name: I3C_Controller_SA

Project Location: C:\Salto

Application Structure: Advanced   ☐ Do not generat

Toolchain Folder Location: C:\Salto\I3C_Controller_SA\

Toolchain / IDE: STM32CubeIDE   ☑ Generate Under Root

Linker Settings

Minimum Heap Size: 0x200

Minimum Stack Size: 0x400

I3C
PRIMARY
CONTROLLER

Part 1: I3C Controller Open & Modify Project

- Open the project using STM32CubeIDE
- Modify the main.c file
  - By using cheat sheet
  - Or following next slides

I3C
PRIMARY
CONTROLLER

Part 1: I3C Controller Define variables

```
/* USER CODE BEGIN PV */
/* Used for dynamic address assignment (ENTDAA) enumeration */
volatile uint32_t uwTargetCount = 0;    /* Number of I3C targets enumerated */
uint8_t targetAddr = 0x32;              /* Address of I3C target */
volatile uint32_t uwEnumDone= 0;        /* Flag for signaling end of enumeration */
volatile uint64_t lastUid;              /* Last UID (incl. BCR, DCR) */

/* Used for IBI reception */
I3C_DeviceConfTypeDef DeviceConf;       /* Struct for configuring IBI reception */
volatile uint32_t uwIBIRequested = 0;   /* Flag signaling IBI received */
I3C_CCCInfoTypeDef CCCInfo;             /* Struct for reading IBI payload */

/* Used for sending SETMRL */
I3C_CCCTypeDef CCCDesc;                 /* Describes CCC direct/broadcast transfer */
I3C_XferTypeDef xferData;               /* More generic XFER sturcture used by HAL */
uint32_t xferData_conrol;               /* Buffer for C-FIFO (single word) */
uint8_t xferData_tx[3];                 /* Buffer for TX-FIFO (3-bytes) */
/* USER CODE END PV */
```

I3C
PRIMARY
CONTROLLER

Part 1: I3C Controller
Implement ENTDAA callbacks

```
/* USER CODE BEGIN 0 */
void HAL_I3C_TgtReqDynamicAddrCallback(I3C_HandleTypeDef *hi3c, uint64_t targetPayload)
{
  /* Send associated dynamic address */
  HAL_I3C_Ctrl_SetDynAddr(hi3c, targetAddr + uwTargetCount);
  /* Store target ID */
  uwTargetCount++;
  lastUid = targetPayload;
}

void HAL_I3C_CtrlDAACpltCallback(I3C_HandleTypeDef *hi3c)
{
  /* Enumeration completed */
  uwEnumDone = 1;
  /* Print debug info about device */
  uint8_t bcr = __HAL_I3C_GET_BCR(lastUid);
  uint8_t dcr = (uint8_t)(lastUid >> 56);
  printf("%d targets enumerated\n", uwTargetCount);
  printf("Last BCR: 0x%x, DCR: 0x%x, full UID: 0x%llx\n", bcr, dcr, lastUid);
}
```

Even if we expect only single Target
We should send address for each callback call
(or abort the procedure)

We printf Target UID once enumeration is done
(Printing in AddrCallback could disturb timing)

I3C
PRIMARY
CONTROLLER

Part 1: I3C Controller Implement IBI & error callback

```c
void HAL_I3C_NotifyCallback(I3C_HandleTypeDef *hi3c, uint32_t eventId)
{
  if ((eventId & EVENT_ID_IBI) == EVENT_ID_IBI)
  {
    uwIBIRequested = 1;
  }
}

void HAL_I3C_ErrorCallback(I3C_HandleTypeDef *hi3c){
  /* Broadcast address not acknowledged */
  if(hi3c->ErrorCode == HAL_I3C_ERROR_CE2){
    printf("No target enumerated\n");
    uwEnumDone = 1;
  }
}
```

life.augmented

I3C
PRIMARY
CONTROLLER

Part 1: I3C Controller
Implement printf via UART

```
PUTCHAR_PROTOTYPE
{
        HAL_UART_Transmit(&huart3, (uint8_t *)&ch, 1, 0xFFFF);
        return ch;
}
/* USER CODE END 0 */


/* USER CODE BEGIN Includes */
#include <stdio.h>
#include "string.h"
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)/* USER CODE END Includes */
```

I3C
PRIMARY
CONTROLLER

Part 1: I3C Controller
Execute dynamic address assignment

```
/* USER CODE BEGIN 2 */
 printf("\n\nHello STM32H5!\n");
 HAL_I3C_Ctrl_DynAddrAssign_IT(&hi3c1, I3C_RSTDAA_THEN_ENTDAA);

 while(uwEnumDone == 0);

 if(uwTargetCount > 0){
   DeviceConf.DeviceIndex = 1;
   DeviceConf.TargetDynamicAddr = 0x32;
   DeviceConf.CtrlRoleReqAck = DISABLE;
   DeviceConf.CtrlStopTransfer = DISABLE;
   DeviceConf.IBIAck = ENABLE;
   DeviceConf.IBIPayload = ENABLE;
   HAL_I3C_Ctrl_ConfigBusDevices(&hi3c1, &DeviceConf, 1);

   /* (Optional) send SETMRL request to limit IBI payload to 4-bytes */
 }

 HAL_I3C_ActivateNotification(&hi3c1, NULL, HAL_I3C_IT_IBIIE);
 printf("Waiting for event...\n");
 /* USER CODE END 2 */
```

I3C
PRIMARY
CONTROLLER

Part 1: I3C Controller Main loop

```
/* USER CODE BEGIN WHILE */
while (1)
{
  while(uwIBIRequested == 0);

  HAL_I3C_GetCCCInfo(&hi3c1, EVENT_ID_IBI, &CCCInfo);
  HAL_GPIO_TogglePin(USER_LED1_GPIO_Port, USER_LED1_Pin);
  uwIBIRequested = 0;

  printf("Interrupt received from 0x%x, count = %d, data = 0x%x\n",
         CCCInfo.IBICRTgtAddr, CCCInfo.IBITgtNbPayload, CCCInfo.IBITgtPayload);
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

Compile and run the code
So far we don’t have any target connected

life.augmented

I3C
TARGET

Part 2: I3C Target Firmware sequence

- Initialize I3C (generate by MX)
- Wait for dynamic address assignment
- Wait for button press
- Send IBI when button press is detected
- Wait for IBI completion

Loop

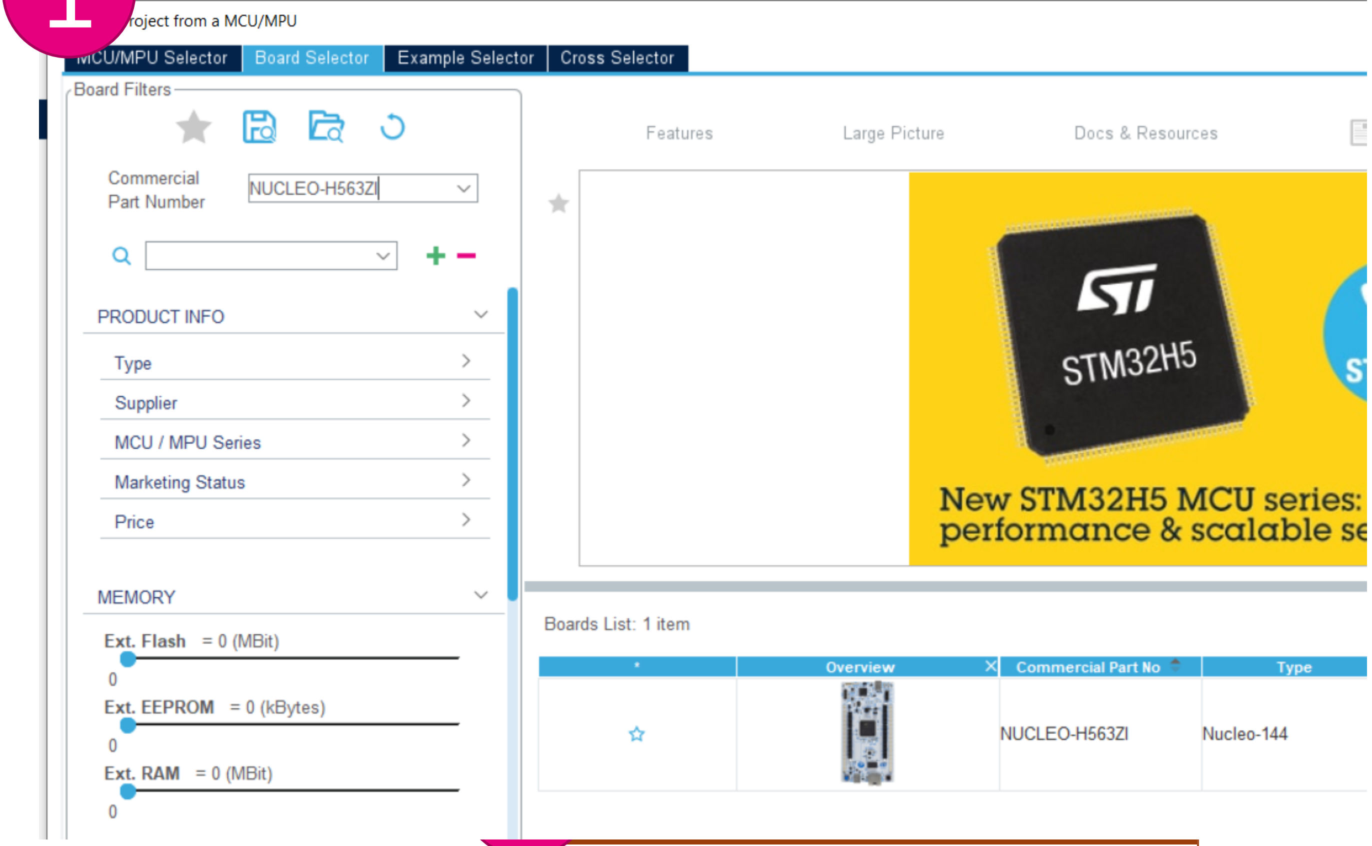
I3C TARGET

Same as I3C Controller

Part 2: I3C Target Creating CubeMX project

- Start by selecting **NUCLEO-H563ZI board**
  - This will initialize the LEDs and push buttons automatically
- When prompted “Initialize all peripherals in default mode”, select No
- Select project **without TrustZone**

1



Commercial Part Number: NUCLEO-H563ZI

Ext. Flash = 0 (MBit)

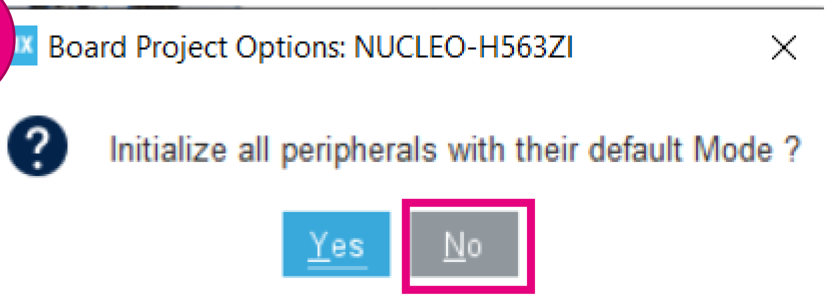
Ext. EEPROM = 0 (kBytes)

Ext. RAM = 0 (MBit)

Boards List: 1 item

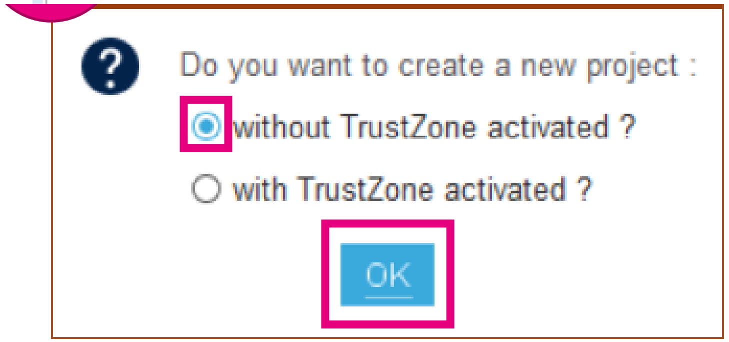
*	Overview	Commercial Part No	Type
☆		NUCLEO-H563ZI	Nucleo-144

2



Board Project Options: NUCLEO-H563ZI

Initialize all peripherals with their default Mode ?



Do you want to create a new project :

● without TrustZone activated ?

○ with TrustZone activated ?

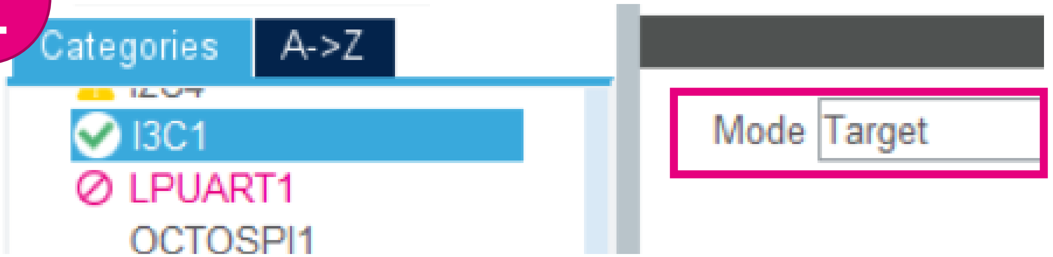
life.augmented

I3C TARGET

Part 2: I3C Target Configure I3C peripheral

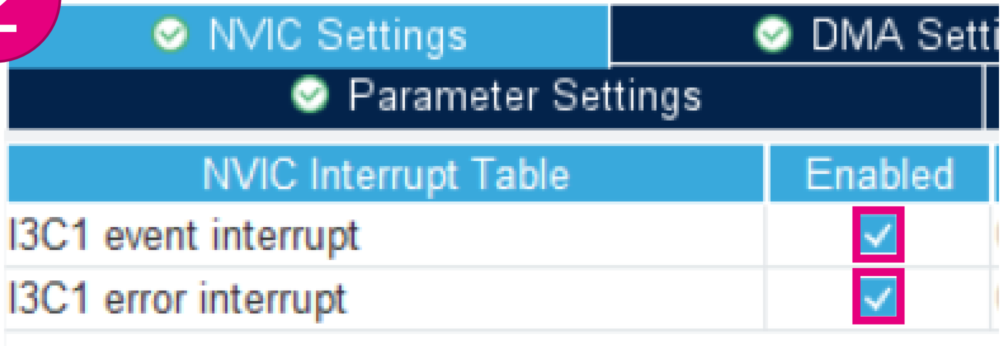
- Enable I3C1 in Target mode
- Leave default timing
- Enable event & error interrupts in NVIC
- The default GPIO configuration should be ok: Alternate function push-pull, very high speed, no pull-up

1



Mode Target

2



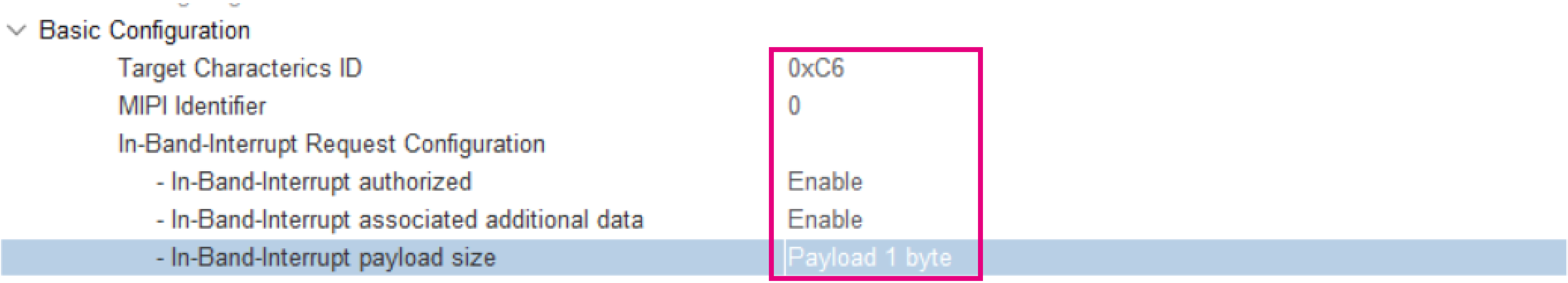
NVIC Interrupt Table	Enabled
I3C1 event interrupt	☑
I3C1 error interrupt	☑

life.augmented

I3C
TARGET

Part 2: I3C Target
Enable I3C IBI

- Set “Target Characteristics ID” to 0xC6
  - This is the DCR MIPI value for MCU
  - Leavin 0x00 (Generic device) also ok
- **Enable In-band-Interrupt authorized and associated additional data**
- Keep payload at 1 byte



Basic Configuration

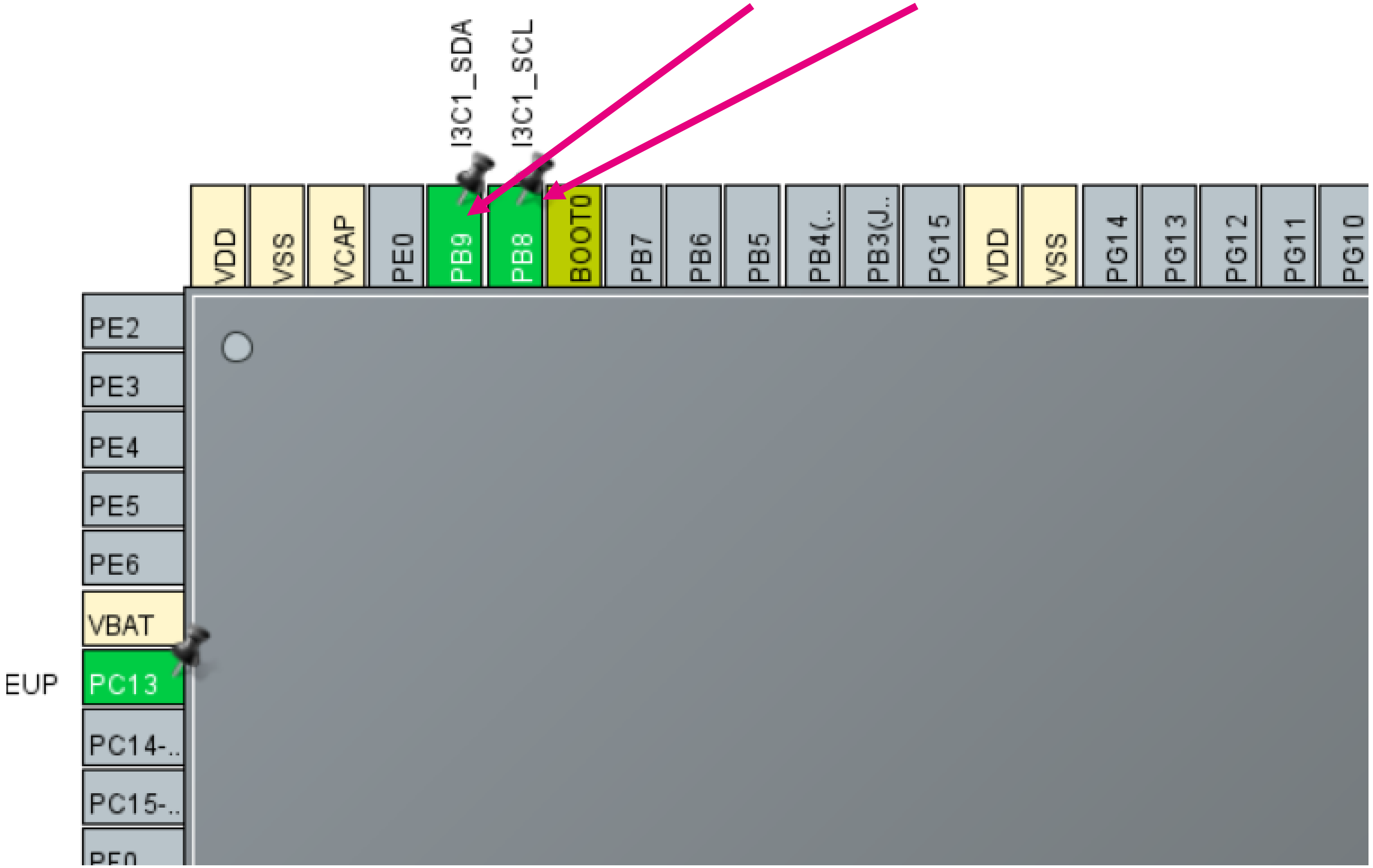
Target Characterics ID	0xC6
MIPI Identifier	0
In-Band-Interrupt Request Configuration	
- In-Band-Interrupt authorized	Enable
- In-Band-Interrupt associated additional data	Enable
- In-Band-Interrupt payload size	Payload 1 byte

I3C
TARGET

Same as I3C Controller

Part 2: I3C Target

Move pins to PB8/PB9 (Arduino header)



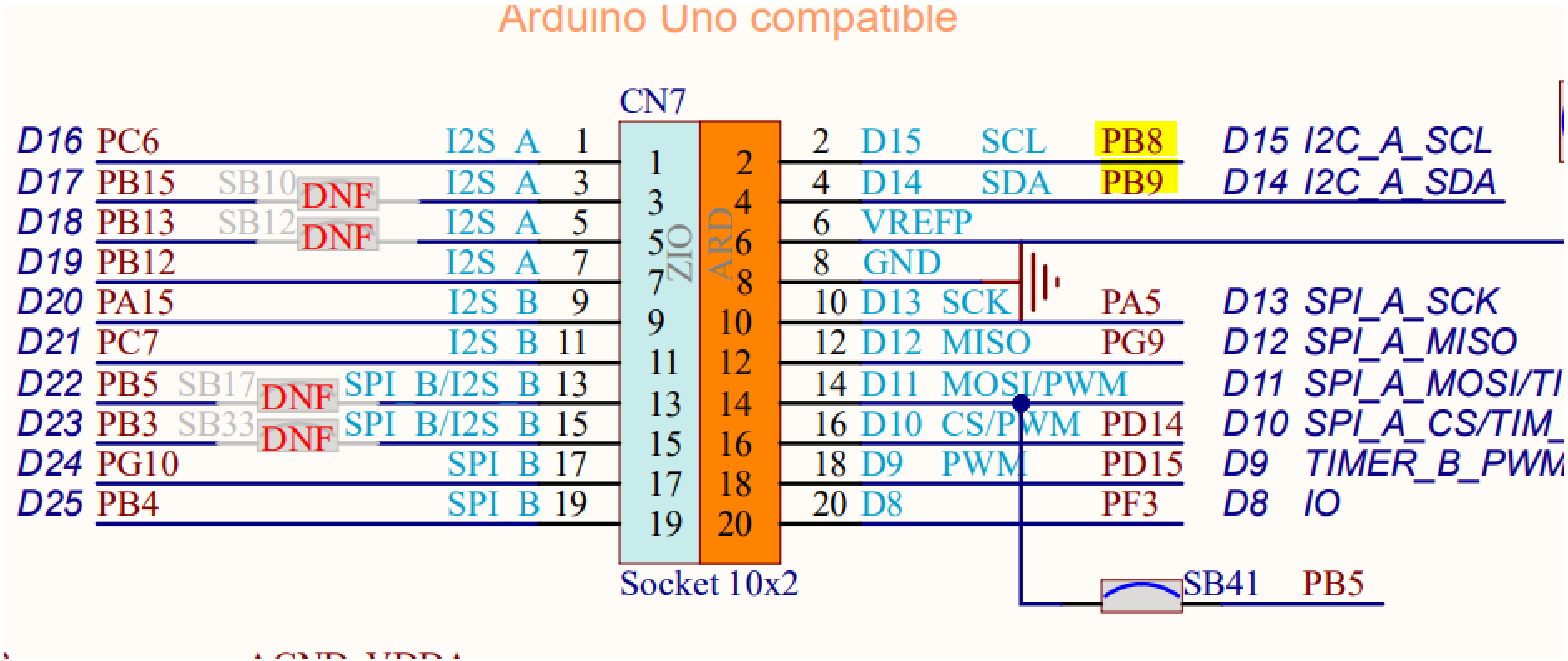
life.augmented

I3C TARGET

Same as I3C Controller

Part 2: I3C Target

Move pins to PB8/PB9 (Arduino header)

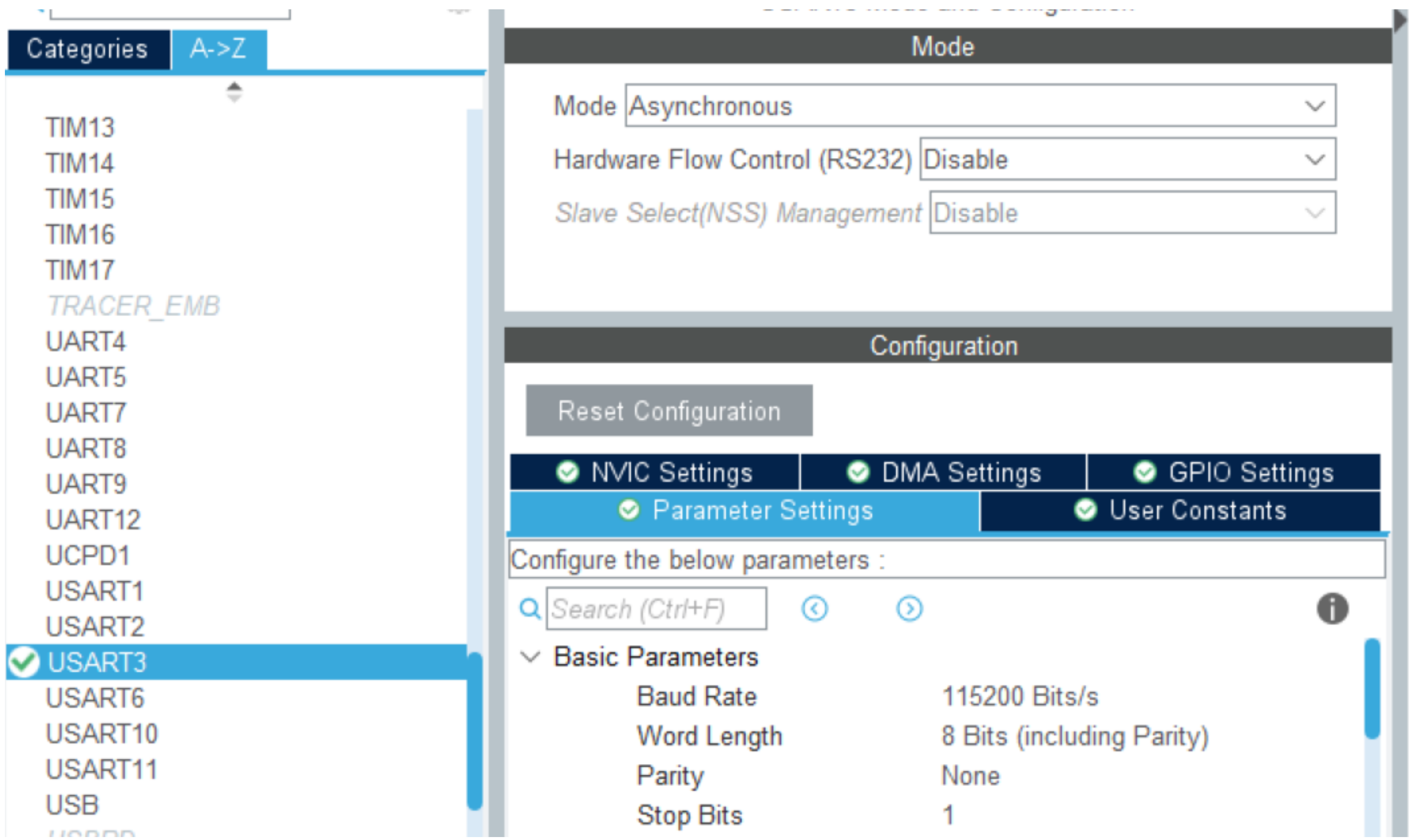

Arduino Uno compatible
CN7
D16 PC6 I2S_A 1 | 2 D15 SCL PB8 D15 I2C_A_SCL
D17 PB15 SB10 DNF I2S_A 3 | 4 D14 SDA PB9 D14 I2C_A_SDA
D18 PB13 SB12 DNF I2S_A 5 | 6 VREFP
D19 PB12 I2S_A 7 | 8 GND
D20 PA15 I2S_B 9 | 10 D13 SCK PA5 D13 SPI_A_SCK
D21 PC7 I2S_B 11 | 12 D12 MISO PG9 D12 SPI_A_MISO
D22 PB5 SB17 DNF SPI_B/I2S_B 13 | 14 D11 MOSI/PWM D11 SPI_A_MOSI/TI
D23 PB3 SB33 DNF SPI_B/I2S_B 15 | 16 D10 CS/PWM PD14 D10 SPI_A_CS/TIM_
D24 PG10 SPI_B 17 | 18 D9 PWM PD15 D9 TIMER_B_PWM
D25 PB4 SPI_B 19 | 20 D8 PF3 D8 IO
ZIO
ARD
Socket 10x2
SB41 PB5
AGND VDDA

I3C TARGET

Same as I3C Controller

Part 2: I3C Target
Configure USART3 for debug

- Enable USART3 in Asynchronous mode
- Leave default configuration (115200 baudrate, 8-bits without parity)



Mode: Asynchronous

Hardware Flow Control (RS232): Disable

Slave Select(NSS) Management: Disable

Configure the below parameters :

Basic Parameters

Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

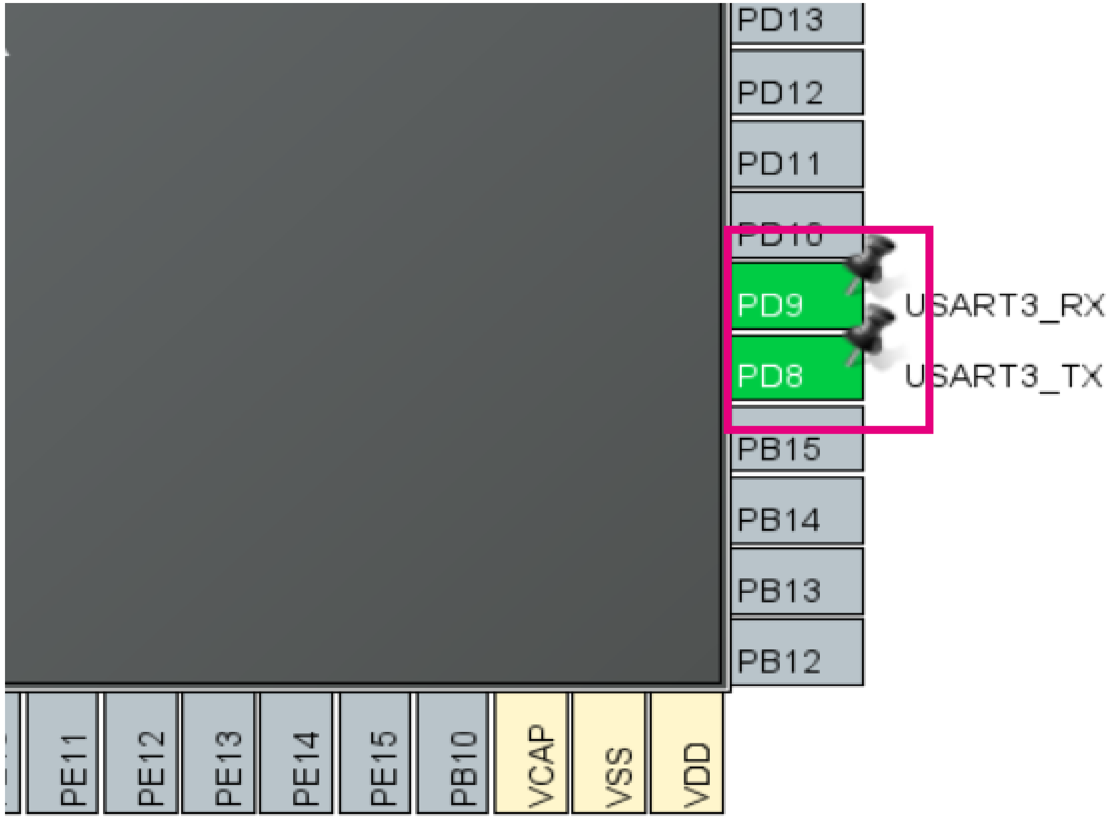
life.augmented

I3C TARGET

Same as I3C Controller

Part 2: I3C Target
Configure USART3 for debug

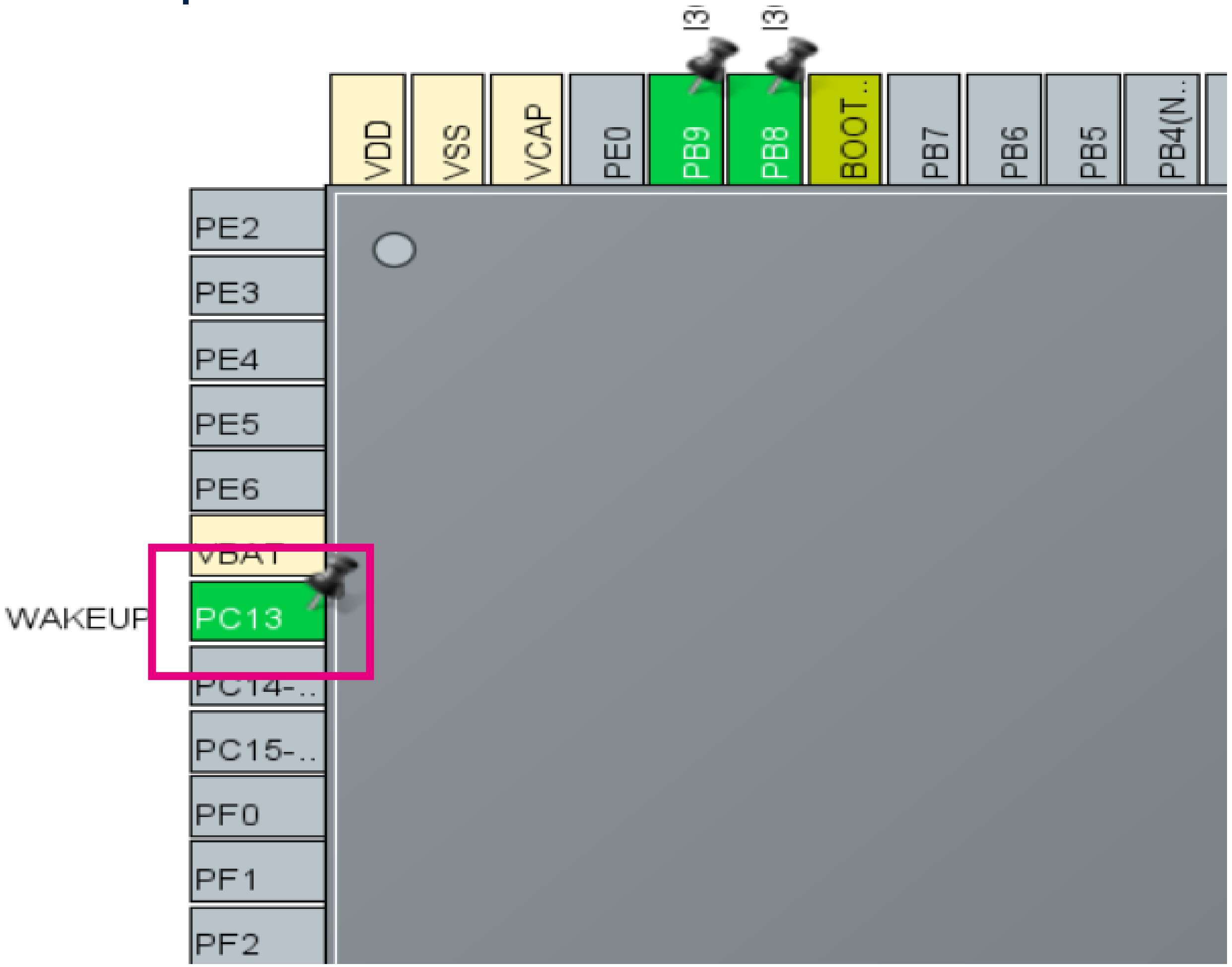
- Map USART3 to PD8 and PD9 which are connected to STLINK VCP



I3C
TARGET

Part 2: I3C Target
Configure Wakeup Button

- Assign PC13 to GPIO output and name it «WAKEUP»

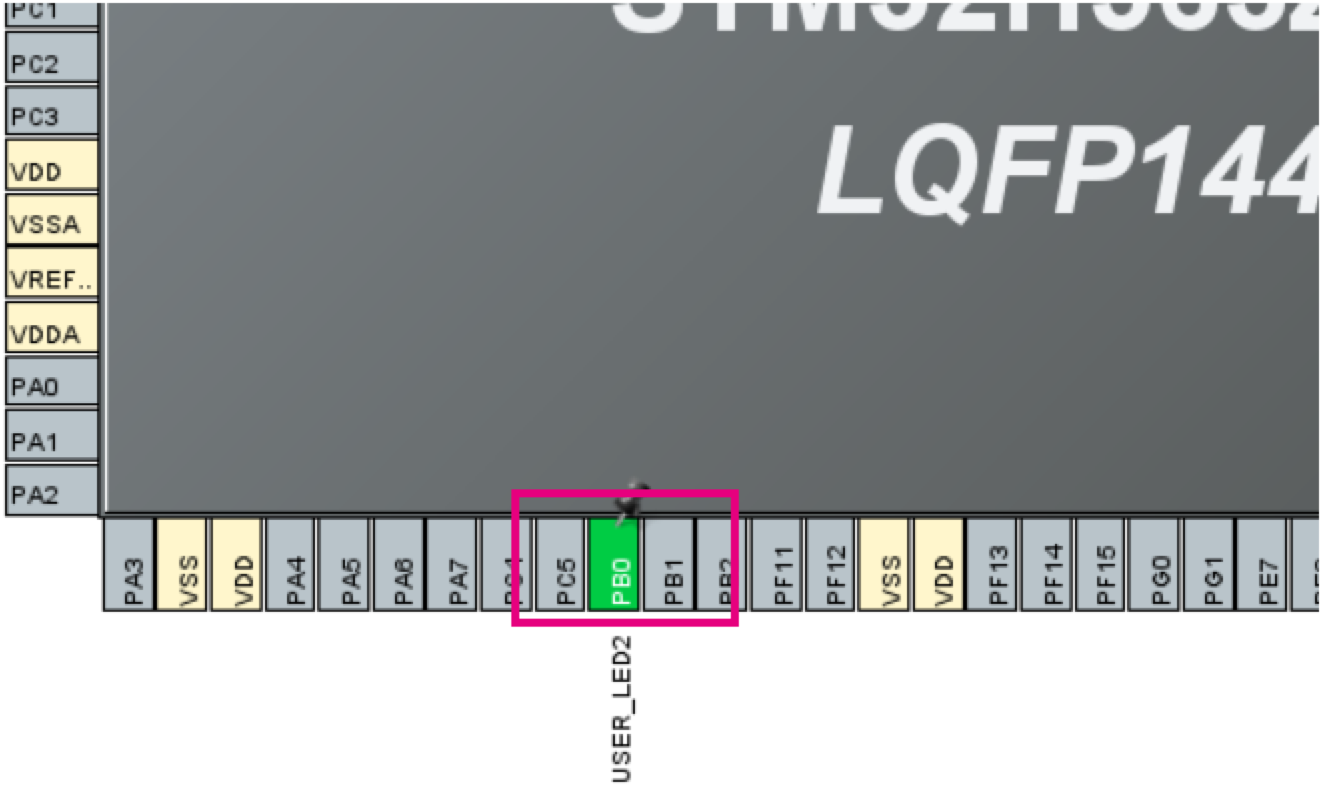


ST Restricted

I3C
TARGET

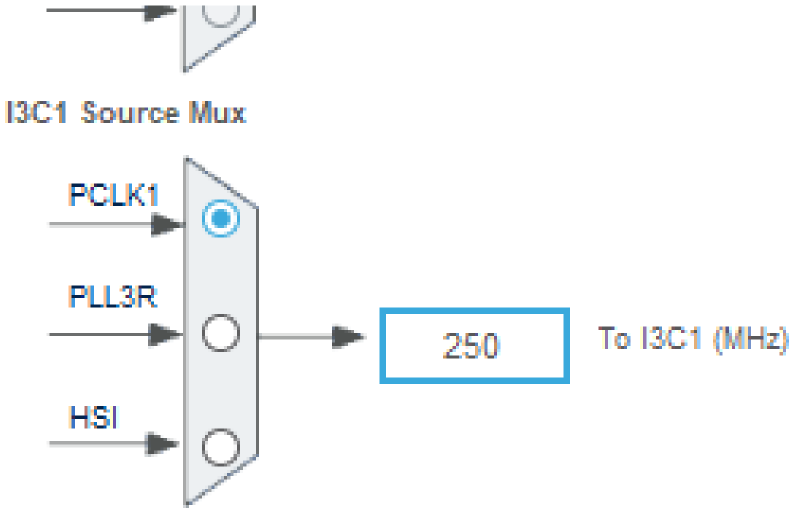
Part 2: I3C Target
Configure LED

- Assign PB0 to GPIO output and name it «USER_LED2»



life.augmented

Verify you select PLL clock 250 Mhz



I3C1 Source Mux

PCLK1 ●

PLL3R ○

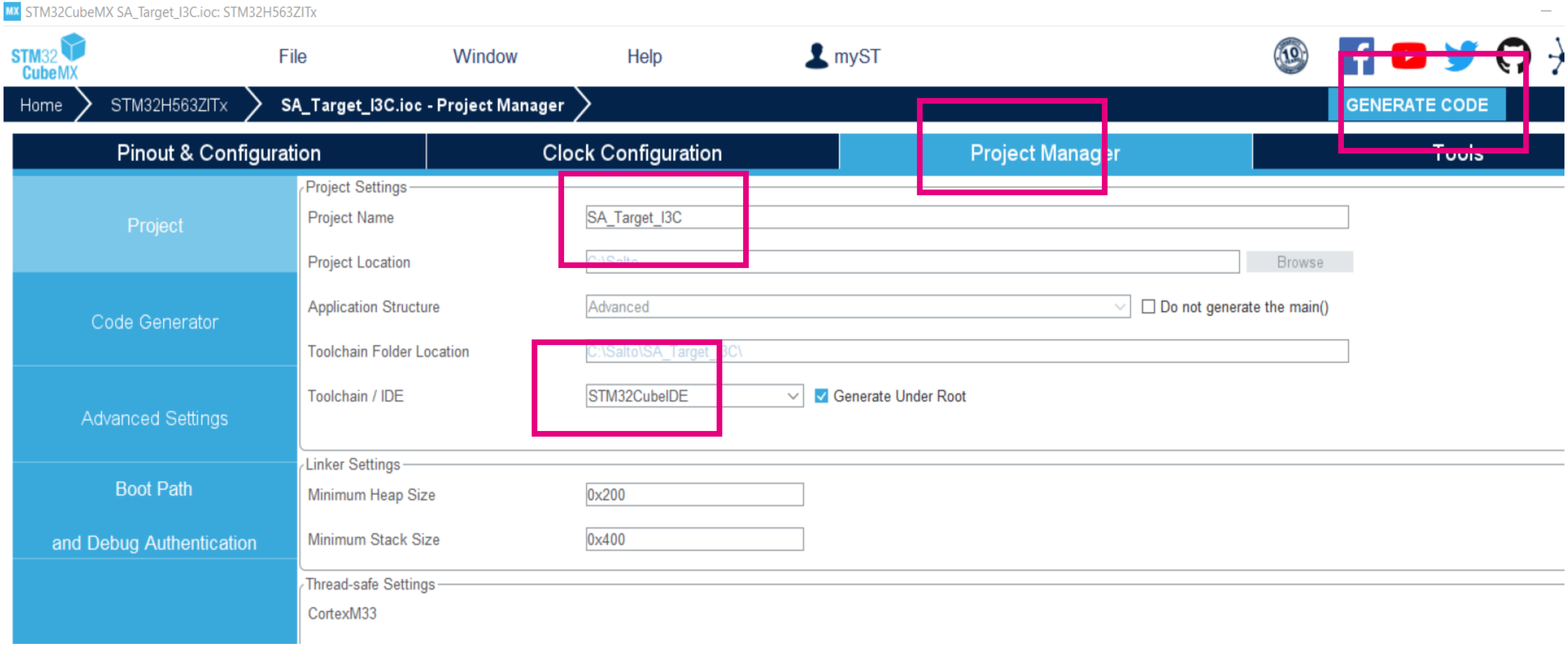
HSI ○

250 To I3C1 (MHz)

ST Restricted

I3C TARGET

Part 2: I3C Target Generate Code



STM32CubeMX SA_Target_I3C.ioc: STM32H563ZITx

Project Settings

Project Name: SA_Target_I3C

Project Location: C:\Salto

Application Structure: Advanced

☐ Do not generate the main()

Toolchain Folder Location: C:\Salto\SA_Target_I3C\

Toolchain / IDE: STM32CubeIDE

☑ Generate Under Root

Linker Settings

Minimum Heap Size: 0x200

Minimum Stack Size: 0x400

Thread-safe Settings

CortexM33

life.augmented

ST Restricted

I3C
TARGET

Part 2: I3C Target
Open & Modify Project

- Open the project
  - Make sure the right version of Cube IDE is used
- Modify the main.c file
  - By using dedicated copy-paste file
  - Or following next slides

I3C
TARGET

Part 2: I3C Target
Define variables

```
/* USER CODE BEGIN PV */
/* Contain the IBI Payload (mandatory byte) */
uint8_t ubPayloadBuffer[] = {0xAB};

/* Variable to catch ENTDAA completion */
__IO uint8_t ubDynamicAddressCplt = 0;

/* Variable to catch IBI end of process */
__IO uint8_t ubIBIcplt = 0;
/* USER CODE END PV */
```

I3C
TARGET

Part 2: I3C Target
Implement notification callback

```c
/* USER CODE BEGIN 0 */
void HAL_I3C_NotifyCallback(I3C_HandleTypeDef *hi3c, uint32_t eventId)
{
  if ((eventId & EVENT_ID_DAU) == EVENT_ID_DAU)
  {
    /* Set Global variable to indicate the the event is well finished */
    ubDynamicAddressCplt  = 1;
  }

  if ((eventId & EVENT_ID_IBIEND) == EVENT_ID_IBIEND)
  {
    /* Set Global variable to indicate the the event is well finished */
    ubIBIcplt = 1;

    /* Toggle LED2: Transfer in transmission process is correct */
    HAL_GPIO_TogglePin(USER_LED2_GPIO_Port,USER_LED2_Pin);
  }
}
```

I3C
TARGET

Same as I3C Controller

Part 2: I3C Target
Implement printf via UART3

```
PUTCHAR_PROTOTYPE
{
        HAL_UART_Transmit(&huart3, (uint8_t *)&ch, 1, 0xFFFF);
        return ch;
}
/* USER CODE END 0 */


/* USER CODE BEGIN Includes */
#include <stdio.h>
#include "string.h"
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)/* USER CODE END
Includes */
```

I3C
TARGET

Part 2: I3C Target
I3C Initialization

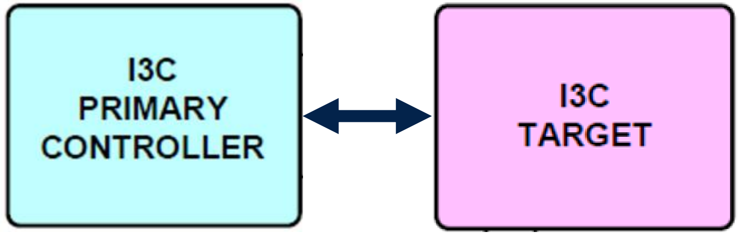
```
/* USER CODE BEGIN 2 */
printf("Hello STM32H5 I3C Target...\n");

HAL_I3C_ActivateNotification(&hi3c1, NULL, (HAL_I3C_IT_DAUPDIE | EVENT_ID_IBIEND));

while (ubDynamicAddressCplt != 1)
{
}

printf("Dynamic address received...\n");
/* USER CODE END 2 */
```

I3C
TARGET

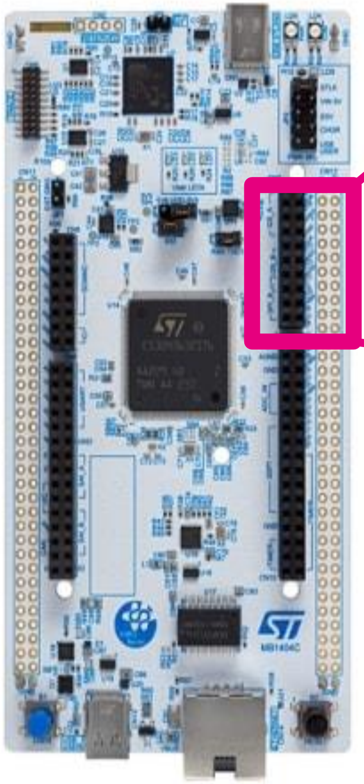
Part 2: I3C Target
Main loop

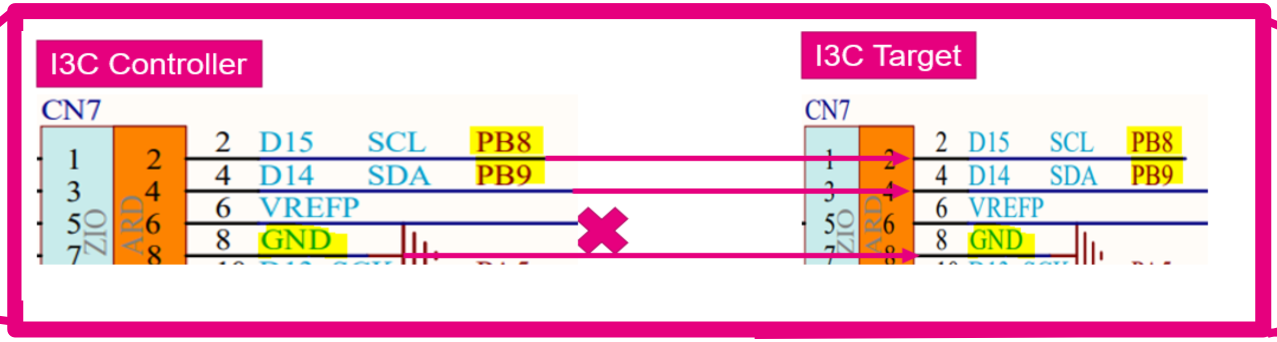
```
/* USER CODE BEGIN WHILE */
while (1)
{
  while(HAL_GPIO_ReadPin(WAKEUP_GPIO_Port, WAKEUP_Pin) == GPIO_PIN_SET);
  while(HAL_GPIO_ReadPin(WAKEUP_GPIO_Port, WAKEUP_Pin) == GPIO_PIN_RESET);

  HAL_I3C_Tgt_IBIReq_IT(&hi3c1, ubPayloadBuffer, 1);

  printf("Button pressed\n");

  while(ubIBIcplt == 0);

  ubIBIcplt = 0;

  printf("IBI sent\n");
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

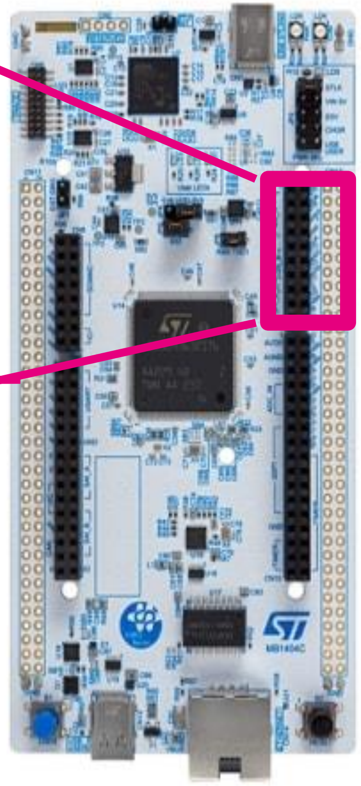
Compile and run the code
So far no Controller is connected
We will continue with connecting boards



Part 3: I3C connection
Connect the boards together







1- Power-off both boards

2- User the 3 provided jumper wires to connect signals between boards

Following signals from Arduino CN7 must be connected

- SCL
- SDA
- (skip VDDA)
- GND

Power the Target board first, then Controller board
Pressing User button on Target board should toggle LED on both boards

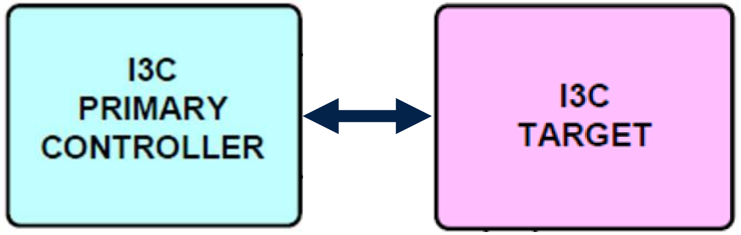
Expected output

I3C Controller

```
Hello STM32H5 Controller!

1 targets enumerated

Last BCR: 0x2e, DCR: 0xc6, full UID:
0xc62e000081130802

Waiting for event...

Interrupt received from 0x32, count = 1,
data = 0xab
```

I3C Target

```
Hello STM32H5 I3C Target...

Dynamic address received...

Button pressed

IBI sent
```



Part 3: I3C connection Conclusion

- This is just a basic example
- Errors are not handled properly
- Some additional features / command should be implemented to conform to I3C specification

Our technology starts with You

Find out more at www.st.com

life.augmented