BLE/RFID Access Control System

Pratham Jain
Piyush Pahuja

Dec 12th, 2023

ECE 395: Advanced Digital Projects Lab
University of Illinois Urbana-Champaign

Abstract

We sought to create a novel keyless technology by integrating Bluetooth Low Energy (BLE) access control into existing RFID-based access systems. Our objective was to design a device capable of seamlessly attaching to doors equipped with RFID scanners across campus. This device establishes a connection with a student's smartphone via Bluetooth as soon as it enters proximity. Upon connection, the smartphone transmits the relevant data from the student's i-card to the device. Subsequently, the device verifies the student's presence in front of the door before emulating the i-card's information to the RFID scanner, effectively unlocking the door without the need for the physical i-card. This approach streamlines access for students on campus, eliminating the necessity to carry traditional access cards.

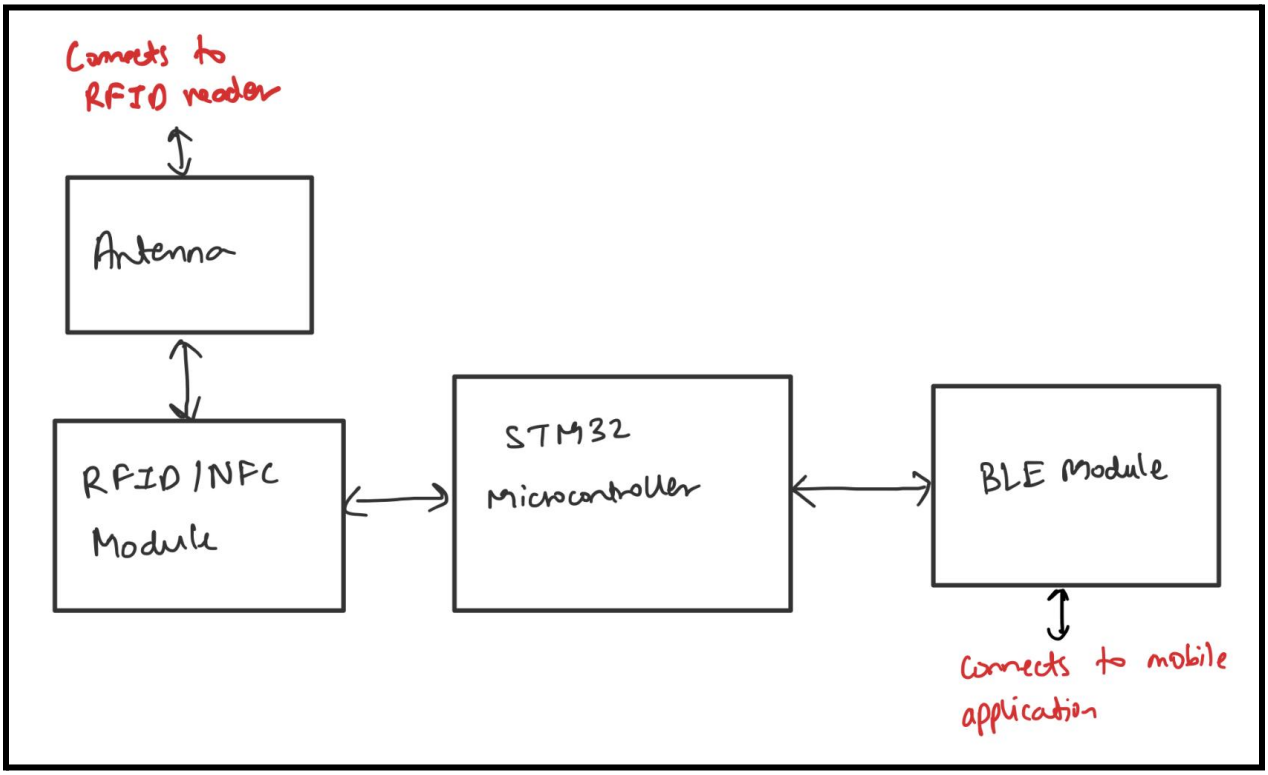

Figure 1: Project Proposal Block Diagram

Table of Contents

Terms and Keywords

ATT: Attribute Profile

BLE: Bluetooth Low Energy

DRC: Design Rules Checker

GAP: Generic Attribute Profile

GATT: Generic Attribute Profile

IC: Integrated Circuit

MOSFET: Metal-Oxide-Semiconductor Field-Effect Transistor

NDEF: NFC Data Exchange Format

NFC: Near-Field Communication

PCB: Printed Circuit Board

RFID: Radio Frequency Identification

SPI: Serial Peripheral Interface

Bluetooth Low Energy

Bluetooth Low Energy (or BLE, for short) is a relatively new technology that was developed over the Bluetooth protocol. It uses the same 2.4 GHz radio frequencies as classic Bluetooth but is designed to work with significantly lower power consumption. The ideal or common use cases for BLE are transferring small amounts of data between devices or interacting with proximity sensors to enable customized experiences. The primary advantage of using BLE is, unsurprisingly, power consumption. Wireless communication modules in devices like phones or computers are usually not limited by power consumption but devices like sensors, fitness trackers, and locks have stricter power requirements because they rely on a smaller power source. For this reason, we opted to use BLE since our device will attach itself to a door lock with a small power source like a CR2023 or 9V battery.

Due to our limited understanding of the working of Bluetooth and BLE, we spent some time researching this and learning more about how BLE works, with our project in mind.

In the protocol stack for Bluetooth, there are two layers of interest to us: the Generic Attribute Profile (GATT) and Generic Access Profile (GAP).

The Generic Attribute Profile defines how data and control is treated in the client-server communication in BLE. Usually, there is a client (a device that connects to other bluetooth devices) and a server device (a

device that accepts connections from other bluetooth devices). When a connection exists between a client and a server, the server exposes to the client certain data that it can control (read, write, etc.). These are in the form of Services and Characteristics. A service is a group of Attributes that are related to or control one piece of functionality. A characteristic can be thought of as an Attribute. It is a piece of data that the server can expose to the client. The GATT defines which services and characteristics to expose to the client and also what permissions the client has, with respect to the attributes.

For example, in our project, the door device was the server and the student's phone was the client. There was one main service, **ICARD**, and this service had one characteristic, **ICARD_NUMBER**. The client only had permission to write to ICARD_NUMBER. When the client (student's phone) writes their i-card number to the characteristic, the firmware would execute the relevant code to perform the desired operation (emulate the card data on the RFID antenna).

For debugging purposes, we had another service called HR_long.

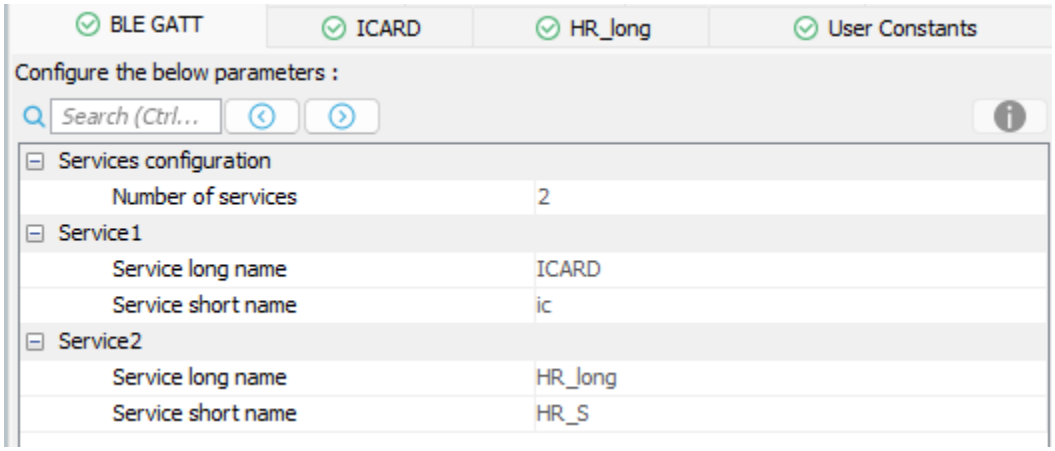

Figure 2: GATT summary.

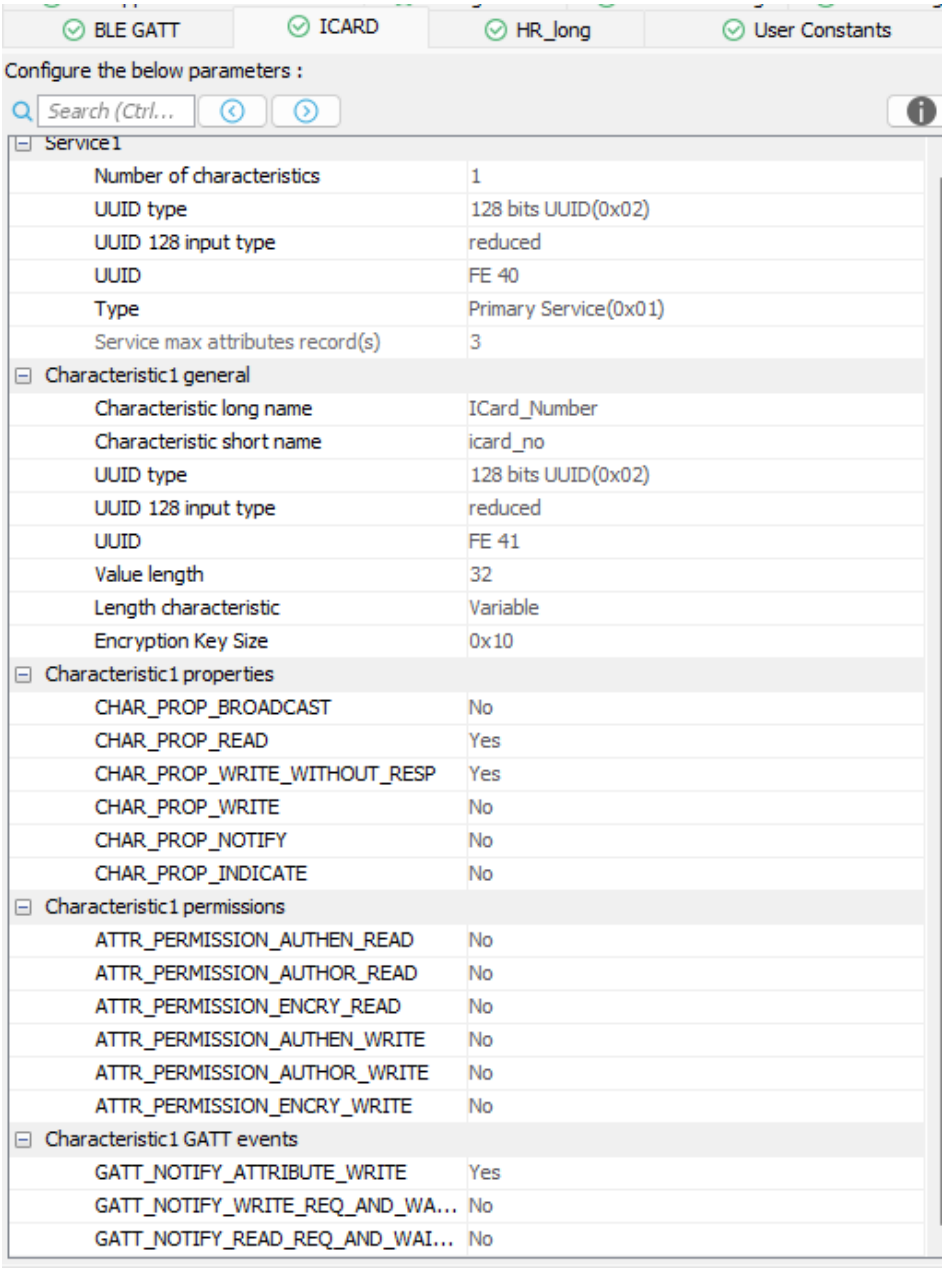

Figure 3: ICARD Service summary.

The Generic Access Profile defines how a server and client connect. This layer provides a framework that includes information like roles of each BLE device (client/server), advertisements, connection establishment, and security. For example, this layer is where we define the name of our device, who can see this device as discoverable, who is allowed to connect the device, etc.

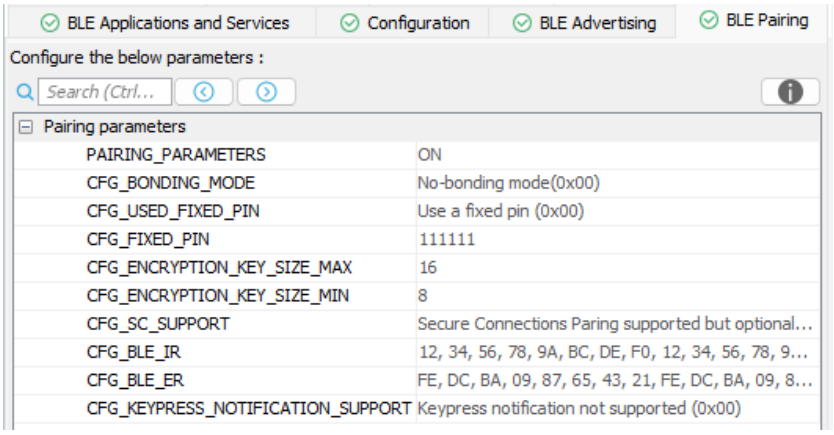

Figure 4: Pairing Configuration

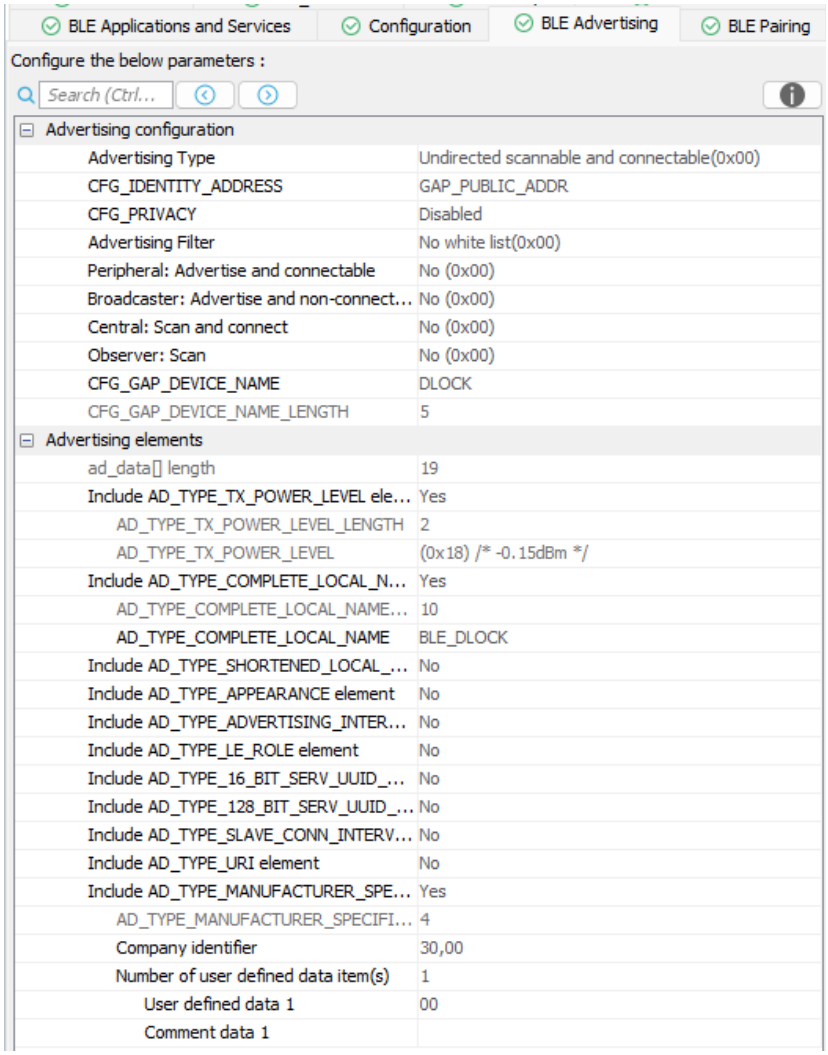

Figure 5: Advertisement configuration.

BLE/RFID Control Access System

Over the course of the semester, we conceptualized different implementations for this project, each one slightly more convenient for the user, but more complicated to develop than the previous one. However, we encountered various challenges and obstacles that encouraged us to brainstorm alternative solutions.

Our first implementation was the most straightforward and an almost brute force one. The student would approach the door and connect to the device using a BLE connection helper application on their phone. After a successful connection, the helper application would send the i-card code to the device. The device would receive the code and emulate that code to the reader. The advantage of this approach was that the device need not store any information related to the student since the RFID reader on the door would accept or reject the i-card just like if the student was tapping their i-card. The obvious disadvantage was that the student was required to do more work than we wished. However, we decided that this would be a very good starting point and would pave the way for further improvements and implementations. Since we were not familiar with the very technical aspects of BLE or RFID, figuring out this approach could have taken a long time.

To acclimate ourselves with working with BLE and RFID using the STM32 microcontroller, we decided to use two development boards that would allow us to develop the firmware and act as a good guide for designing a PCB and selecting required components.

For the BLE aspect, we decided to use the **NUCLEO-WB15CC** development board from ST. This board is based on the **STM32WB15CC6** microcontroller which is specifically designed for BLE applications. The board featured an antenna, transceiver, LEDs, and a Morpho connector, all of which were essential in learning how the microcontroller works.

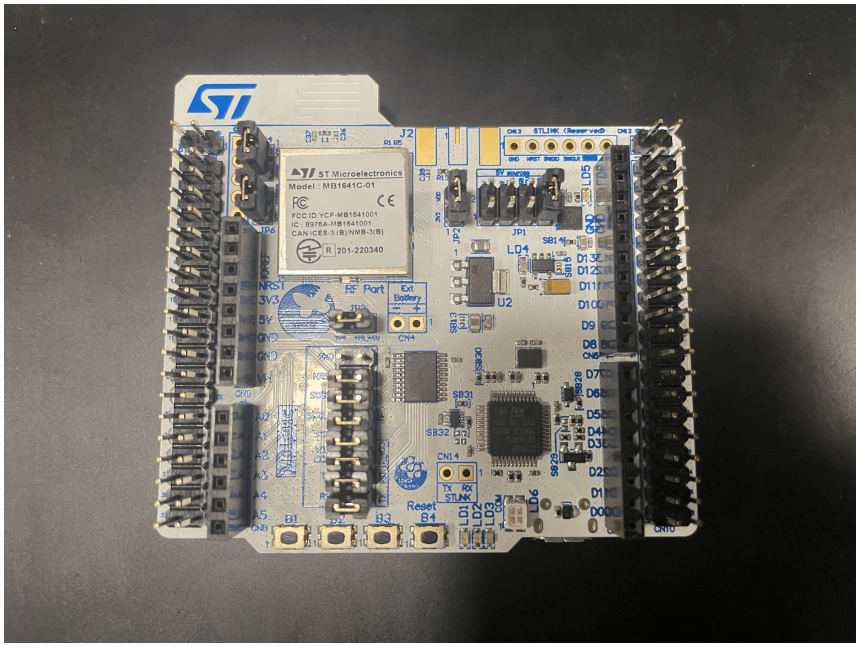

Figure 6: NUCLEO-WB15CC

For the RFID/NFC aspect, we decided to use the **X-NUCLEO-NFC08A1** expansion board. This board can easily connect to the Nucleo board using the Morpho connector and also has essential features like an antenna, matching circuit, and LEDs. This board uses the **ST25R3916B** IC which has card reader and card emulation modes. Specifically, this chip supports the NFC-A / ISO14443A protocol which the school's i-card uses.

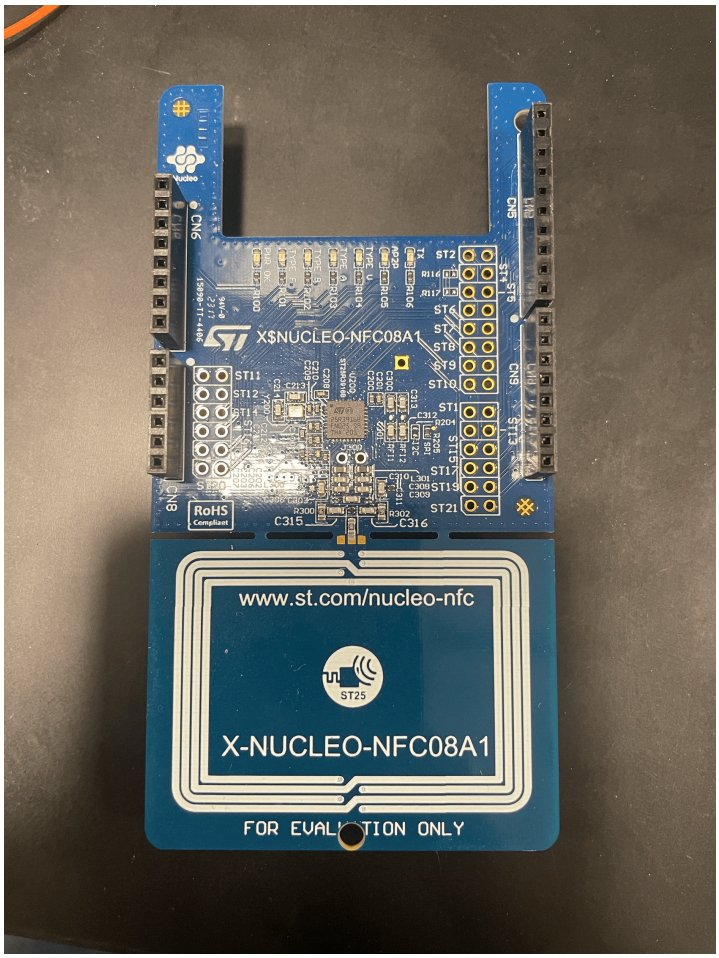

Figure 7: X-NUCLEO-NFC08A1

For both development boards, and consequently their chips, ST provides middleware that significantly helps in the development of firmware. The two middlewares are **STM32 WPAN** and **X-CUBE-NFC6**.

STM32 WPAN

First, we focused on developing the BLE firmware using the STM32 WPAN middleware. Using the WPAN middleware, we were able to create a very bare GATT and GAP for our device. After this, we added code to cater to functionality that we wanted and defined specific advertisement data which would make our device more compatible with the specific BLE helper applications that would be installed on the students' phones. There were many small issues that arose during this process which would stall our development. To debug, we initially used low power UART and then transitioned to UART. Although the

STM32WB15CC MCU has ST-LINK functionality, we were unable to take advantage of this for LPUART or UART debugging. However, we were able to connect a serial to USB adapter (FT232R) to the morpho connectors on the board to add debug logging functionality.

After fixing several small bugs and researching BLE-specific issues, we had developed firmware that allowed a student's phone to connect the device over BLE. After connecting, the phone could send an i-card number to the device to which the device could execute specified handlers. During the mid-semester check-in, we demonstrated this functionality where the device logged the received data from the client to the UART serial port.

X-CUBE-NFC6

After the mid-semester demonstrations, we started designing the schematics for the PCB using datasheets and board schematics as a reference. At the same time, we focused on the RFID aspect of the firmware. Using the provided middleware, we generated the bare code, and using all available resources from ST, we wrote some code to perform basic operations using the X-NUCLEO board. However, the project would refuse to build. At times, the compiler would provide a message that would indicate where the issue could be located but most of the time the compiler would provide random or ambiguous messages like the one below.

```
rivers/STM32F4xx_HAL_Driver/Inc -I../Drivers/STM32F4xx_HAL_Driver/Inc/Legacy -I../Drivers/CMSIS/Device/ST/STM32F4xx/Include -I../Drivers/CMSIS/Inclu
ons -static --specs=nano.specs -mfpu=fpv4-sp-d16 -mfloat-abi=hard -mthumb -Wl,--start-group -lc -lm -Wl,--end-group
 cannot open linker script file \AD.UILLINOIS.EDU\engr-ews\ppahuja2\Downloads\otherboard\nfcother\STM32F407VGTX_FLASH.ld: No such file or directory
```

Figure 8: Error message while building project. Named file exists and is in the specified directory.

Since we were unable to build the project with our project-specific code, we decided to try building very simple projects like tag detection or NDEF logging which simply log any and all data about the tags near the antenna. For the tags, we used multiple students' i-cards, other NFC-enabled cards (like transit cards from Chicago and Hong Kong), and blank NFC tags (like the NTAG203 NFC tag from Adafruit). This did not work either. Any project would refuse to build and would usually spiral into bug fixes and confusing error messages. We suspected that there might be a compatibility issue with the STM32WB15CC MCU so we decided to try redeveloping the simple projects for different microcontrollers and boards including the **STM32L031K6T7** and the **STM32F407G-DISC1**/**MB997D**. This did not work either.

Many of the error messages from the compiler revolved around the files generated by the middleware which we had assumed to be correct or require minimal intervention. Therefore, we were also focused on finding bugs in other parts of the project.

When researching a specific bug, we stumbled upon a forum thread where an ST employee said that they identified an issue stemming from an important header file (**rfal_utils.h**) in the middleware. This particular header file and its .c complement were used in several other files for functionality. Since this was discovered quite late into the semester (posted on Nov 23), we determined that trying to debug and fix this issue ourselves would not be feasible and might not yield a working final project. Although we

have a version of the code for RFID aspect, we were unable to ensure its functionality or debug it properly. Once ST releases an update that fixes this issue, we would be able to use STM32CubeMX to regenerate the relevant middleware files which will overwrite the erroneous code.

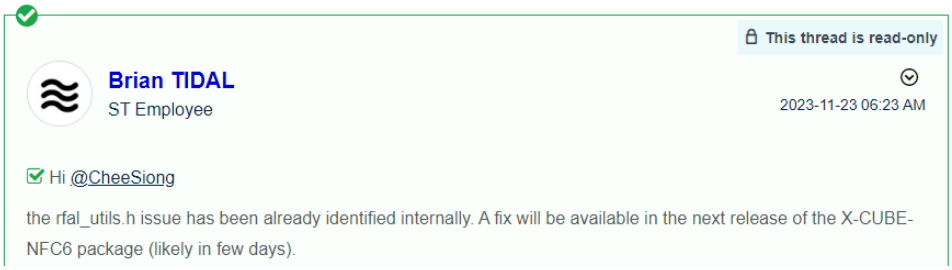

Figure 9: Issue identified by ST.

Pivot to Motor-Controlled Access

Due to the above mentioned problems, we had to pivot to another project. We kept the BLE module however, instead of emulating an RFID signal to open the door, we opened the door using a motor. When a BLE signal is sent to the microcontroller, it subsequently enables power to the motor. The motor unlocks the door and after 5 seconds spins the other way to lock it. We used a MOSFET to control the motor operation. We send a high voltage to the Gate Pin, which allows current to flow between the Source and Drain pins. A MOSFET was required because during initial testing, adding a motor between the GPIO pin and GND would cause the voltage to drop to 0.3V. Therefore, we added a MOSFET between the motor and power from the MCU, itself. This would not require us to obtain another power supply.

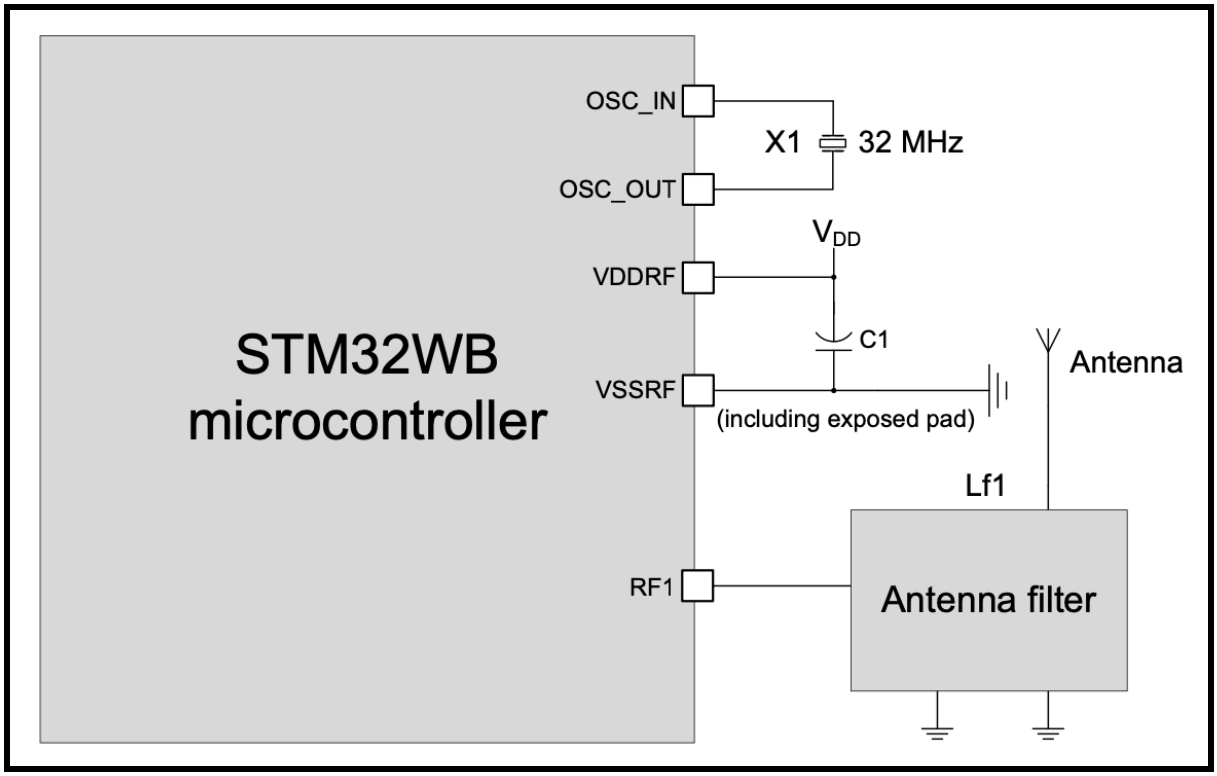

Figure 10: External Circuit to Microcontroller

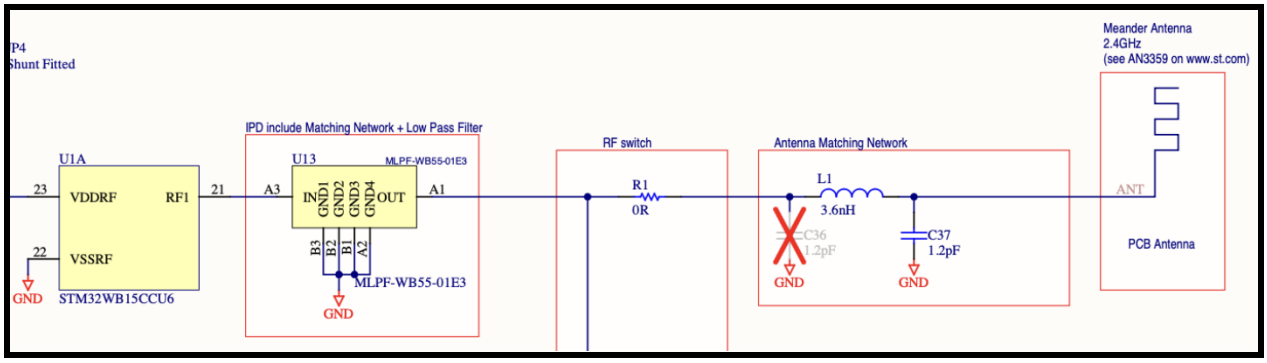

Figure 11: Antenna Filter (Matching Network)

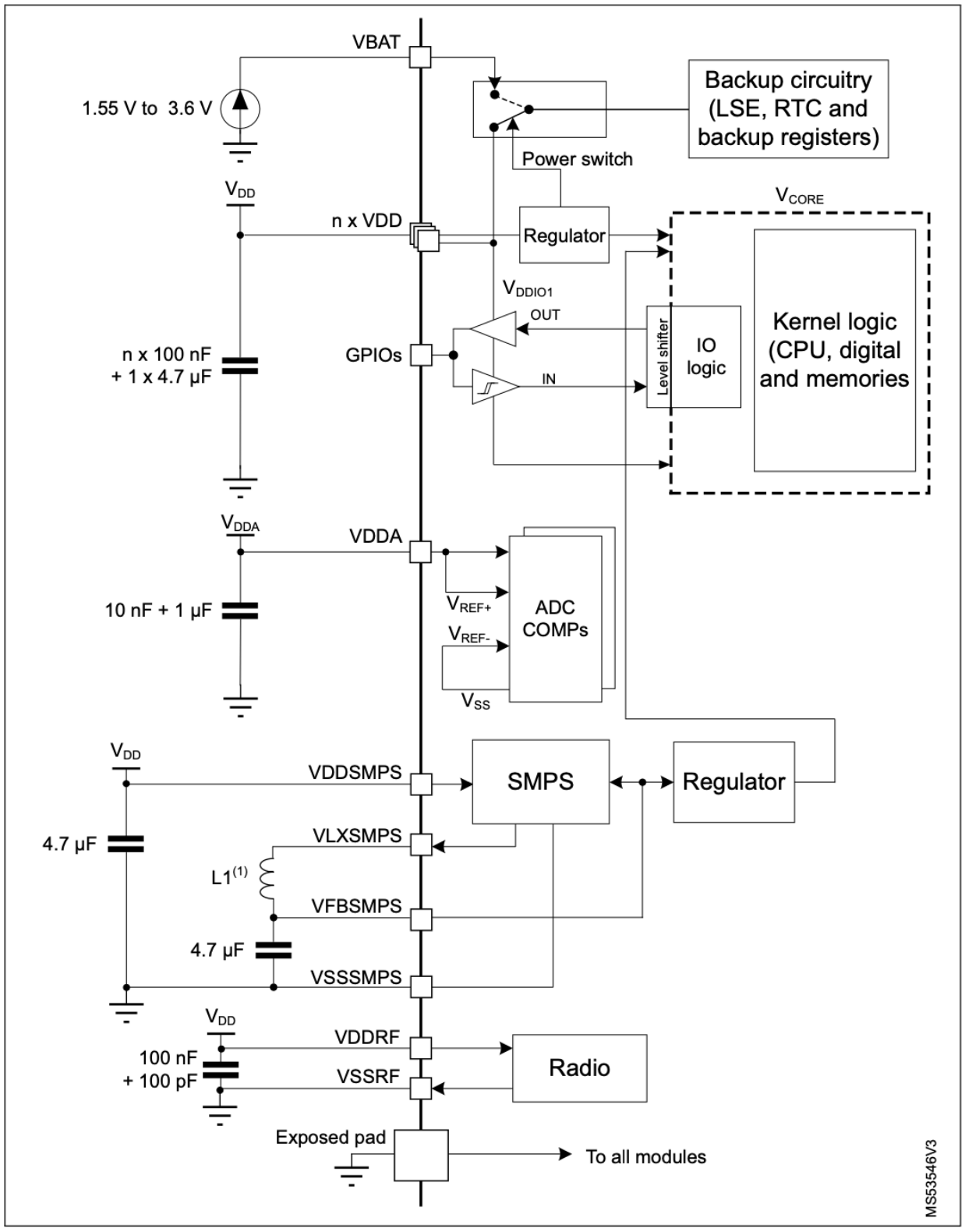

Figure 12: Power Supply Scheme for Microcontroller

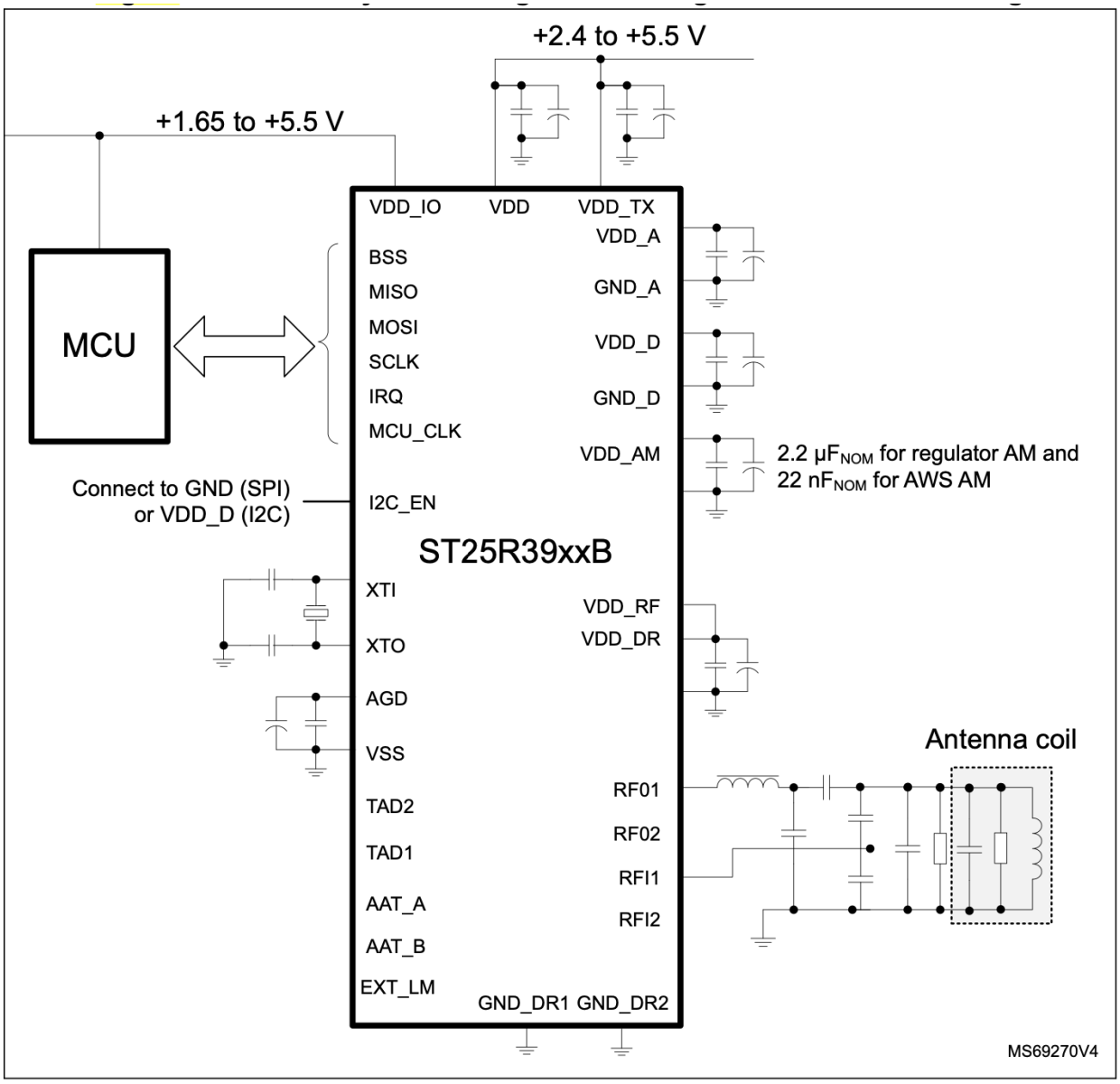

Figure 13: External Connections to the NFC Chip

SPI

SPI is a synchronous serial communication protocol commonly used for connecting microcontrollers, sensors, and other peripheral devices within electronic systems. SPI enables data exchange between a master device and one or more slave devices through a four-wire bus, comprising a serial clock (SCK), a master-out-slave-in (MOSI) line, a master-in-slave-out (MISO) line, and a chip select (CS) line. The master device initiates communication by generating clock pulses, synchronizing data transmission with the connected slaves. Each slave device has a unique chip select line, allowing the master to select the specific device it wishes to communicate with. SPI offers advantages such as high data transfer rates, simplicity in hardware implementation, and support for full-duplex communication. We use SPI to enable communication between the two ICs we use on our board: STM32WB15CC and ST25R3916B.

Schematic

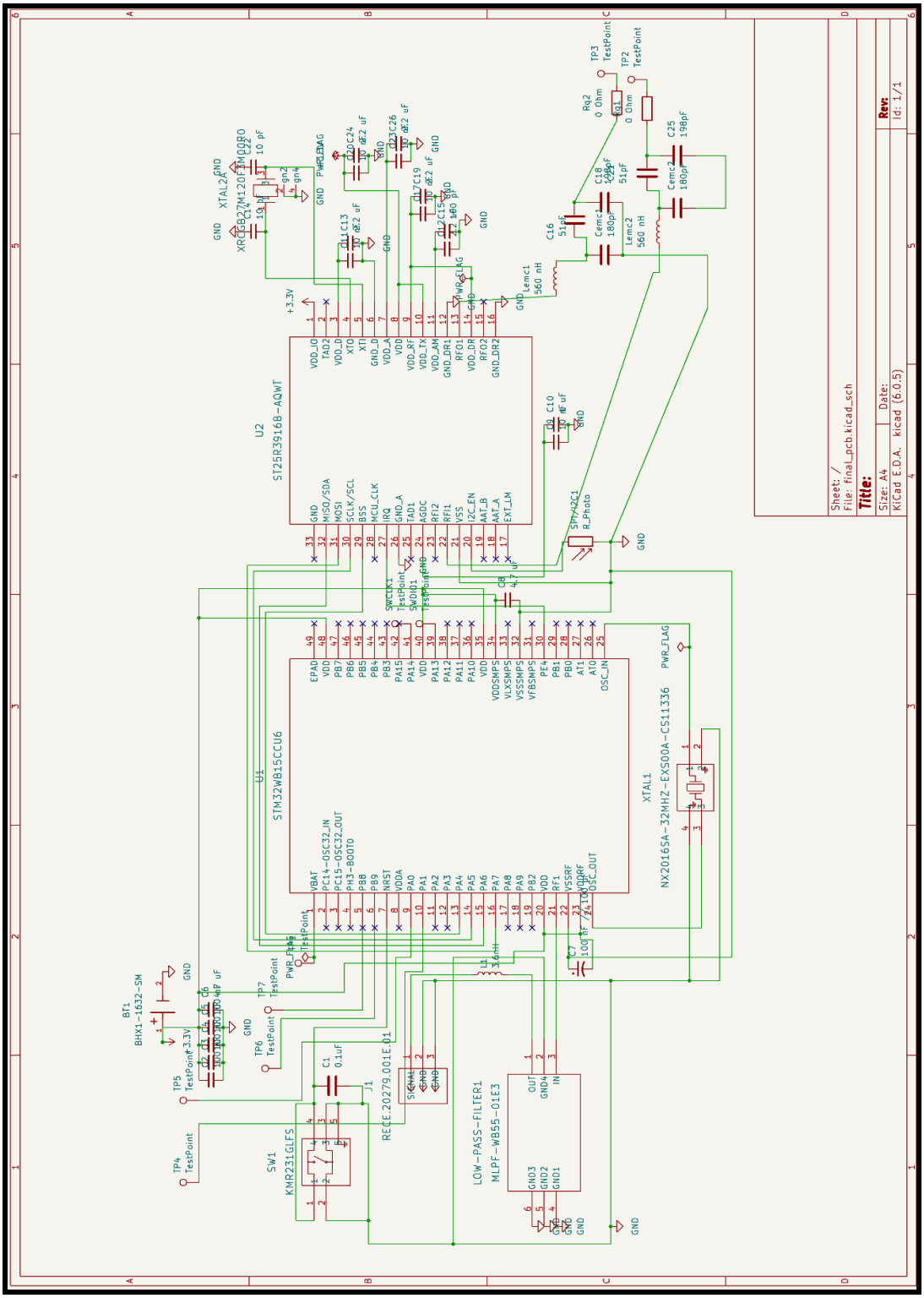

Figure 14: PCB Schematic

We integrated the development kit schematics alongside the schematics sourced from the relevant IC datasheets to discern and optimize the suitable components and values. Subsequently, utilizing KiCad 6.0, we orchestrated the arrangement of the components to formulate our schematic. After subjecting it to scrutiny with the Electrical Rules Checker, we imported our schematic into the PCB editor to articulate the precise layout. Post-validation for Design Rule Check (DRC) errors, we proceeded to export the Gerber files.

PCB Layout

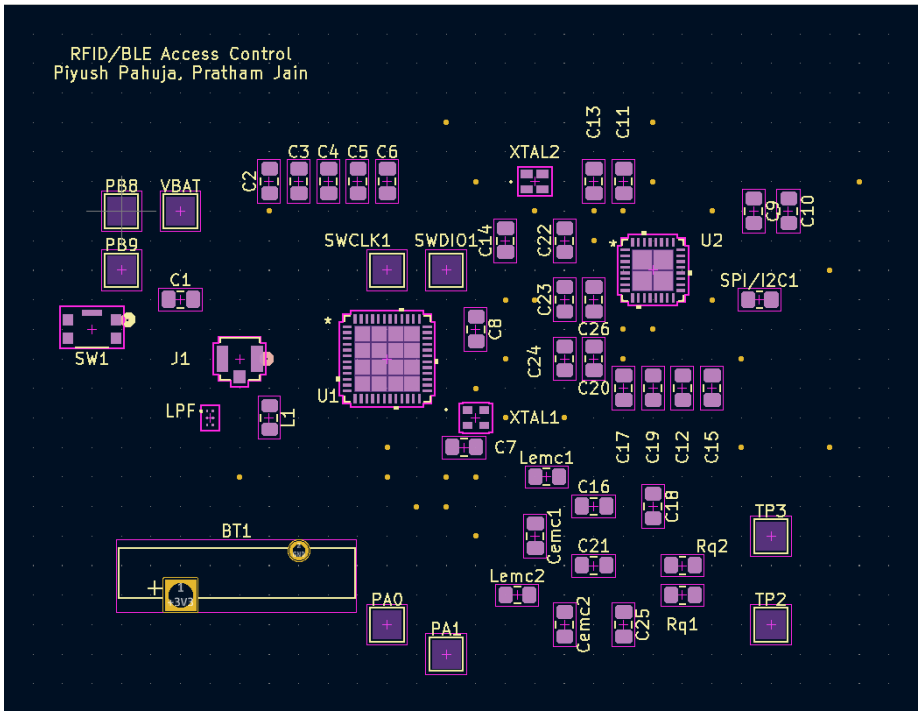

Figure 15: No Copper Layers Selected

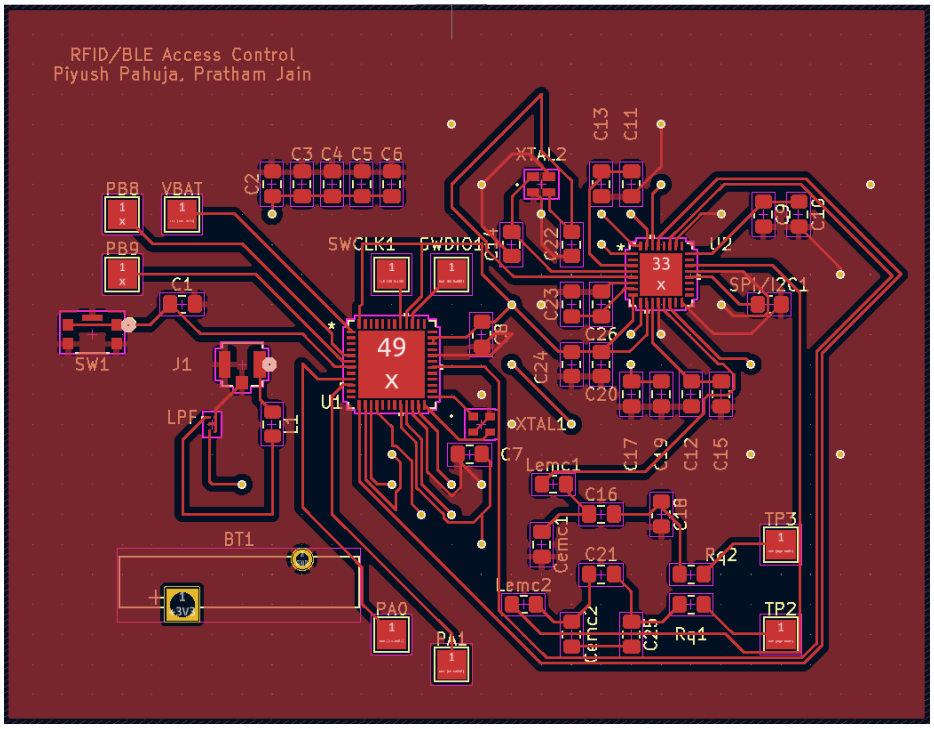

Figure 16: F.Cu

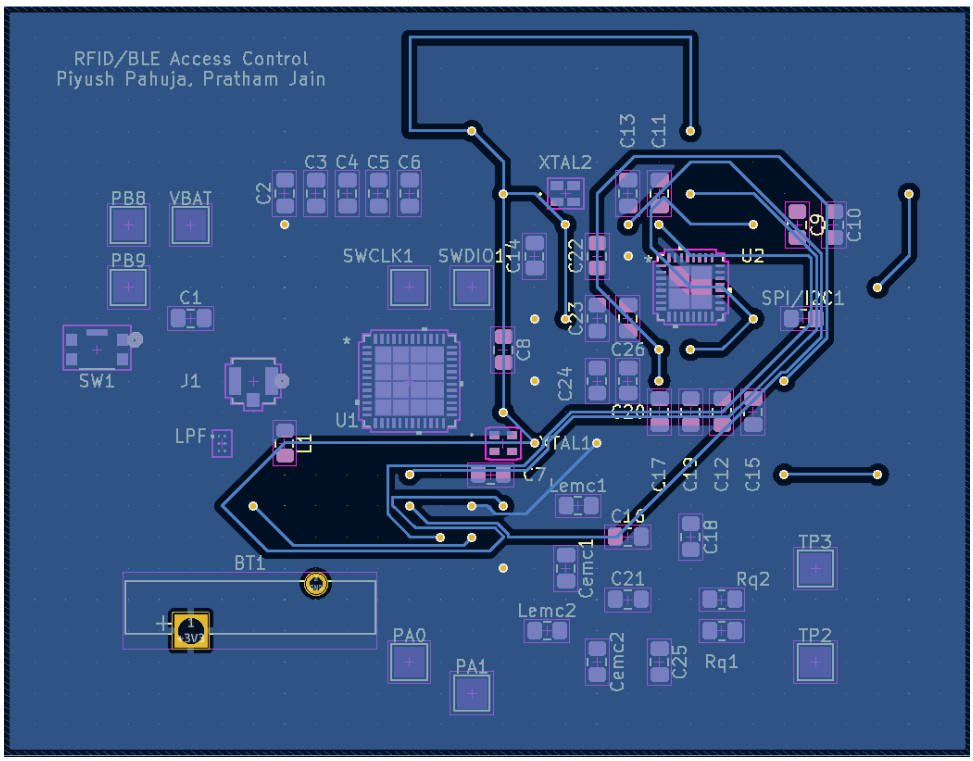

Figure 17: B.Cu

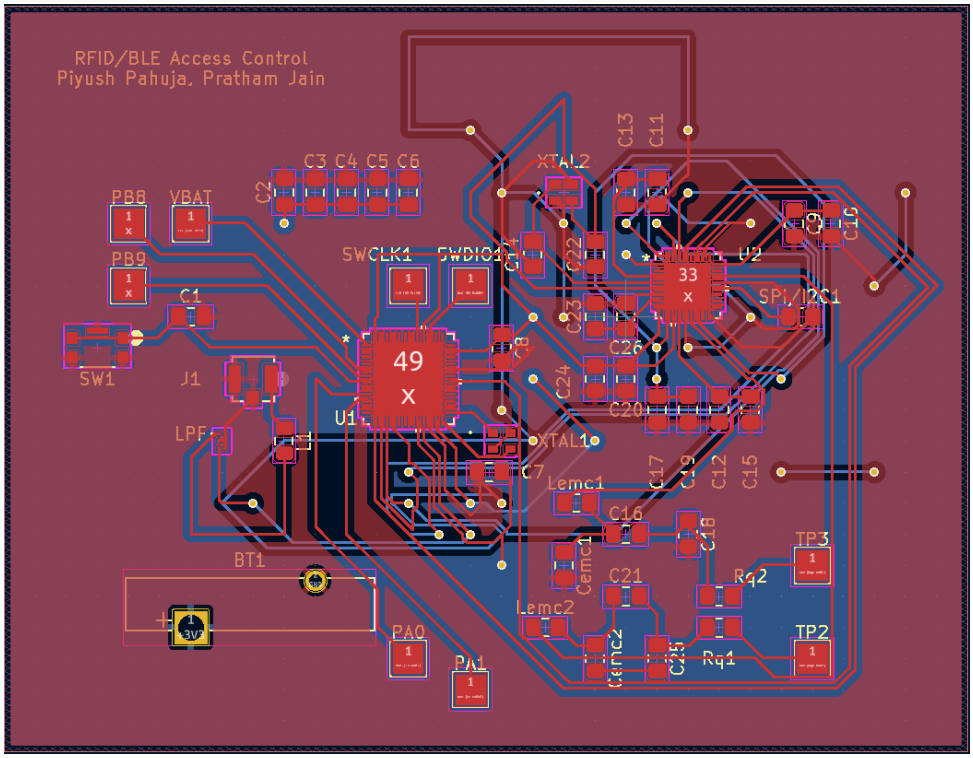

Figure 18: F.Cu & B.Cu Selected

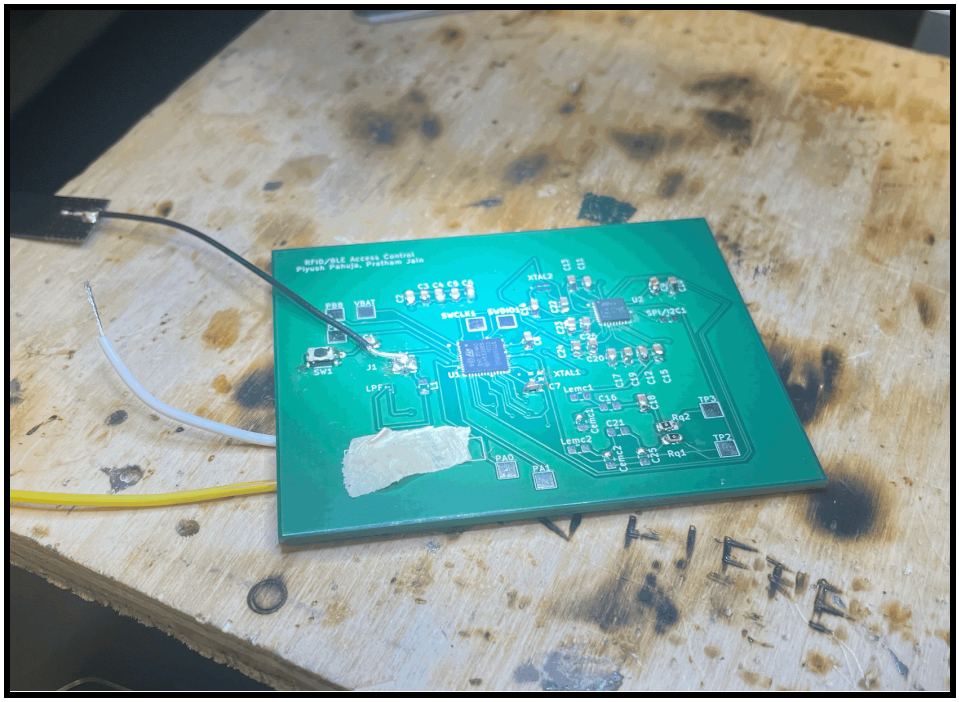

Figure 19: PCB

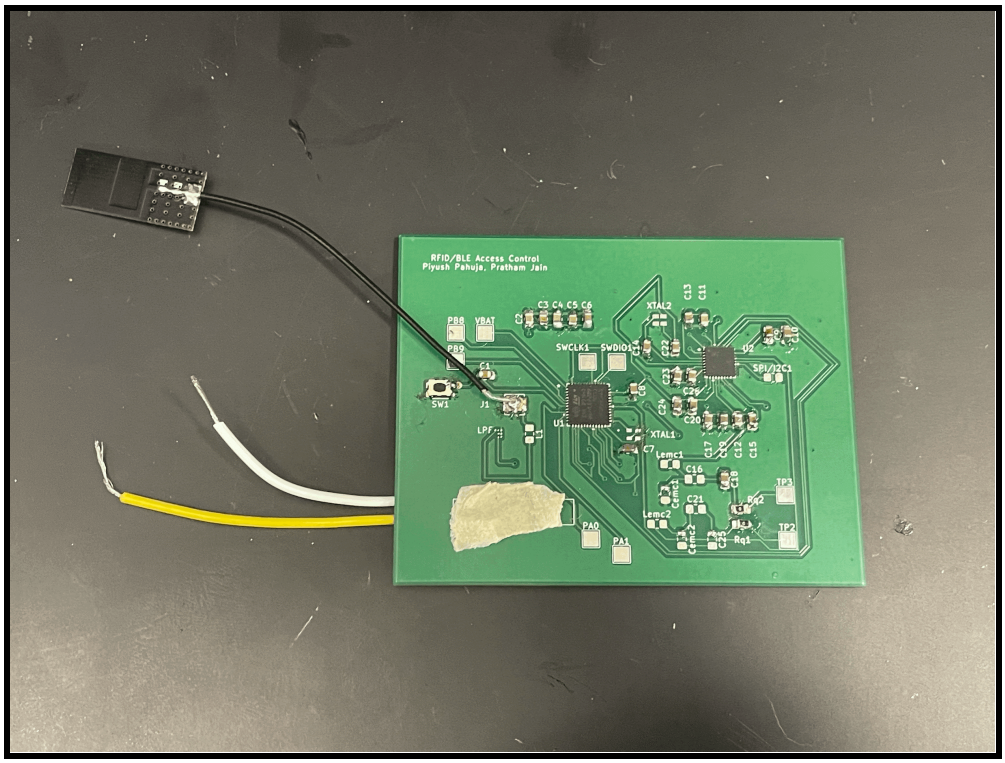

Figure 20: PCB

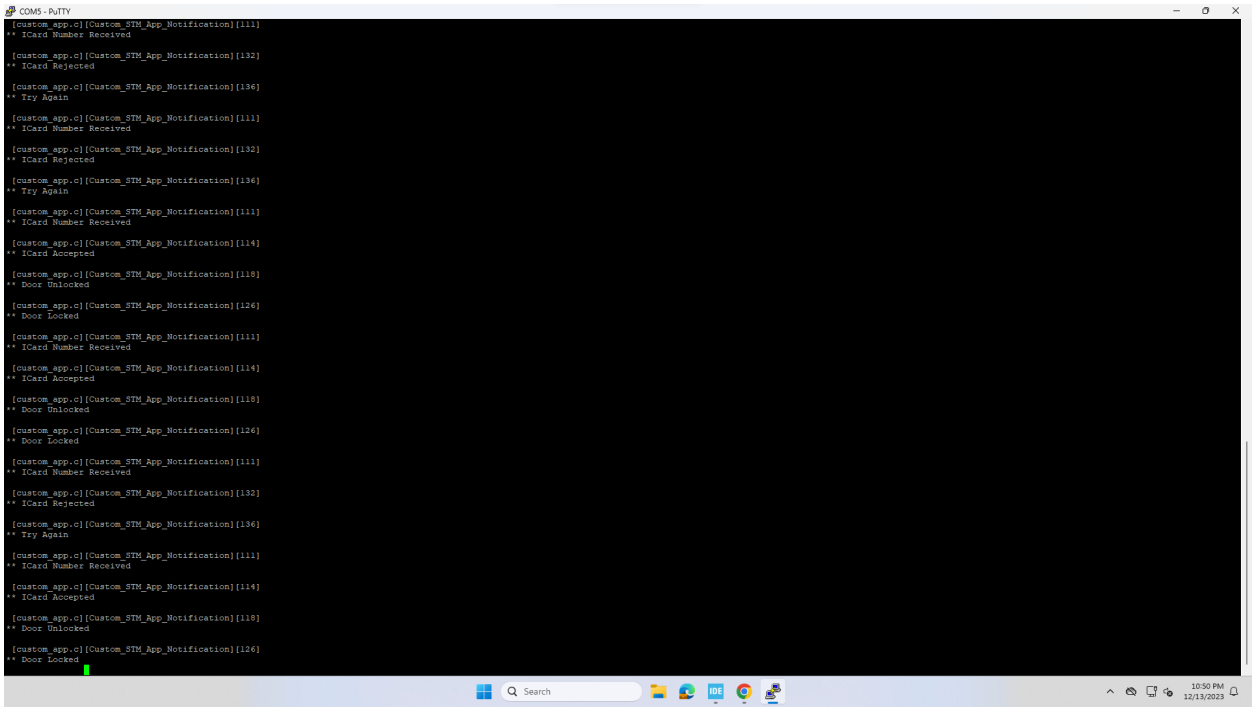

Figure 21: Output to Screen while running project

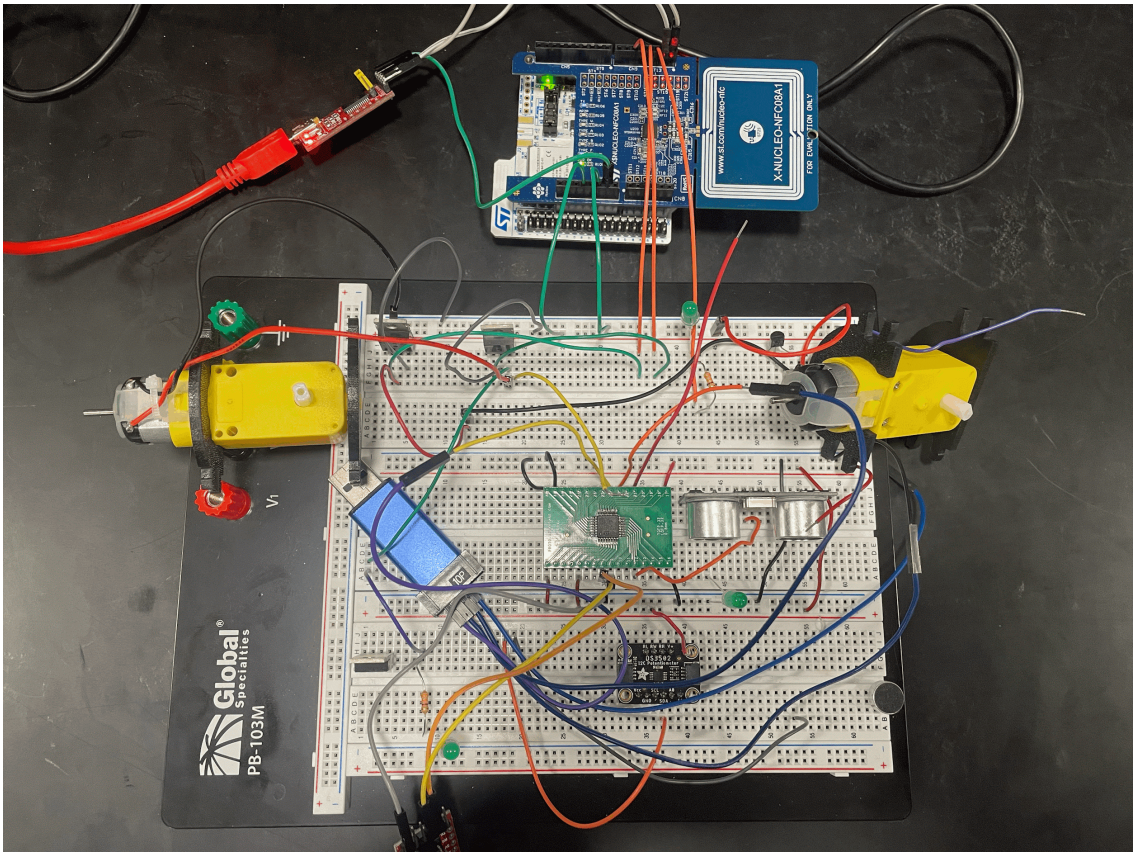

Figure 22: Setup of Final Demonstration using development boards

References

https://www.st.com/en/microcontrollers-microprocessors/stm32wb15cc.html

https://www.st.com/resource/en/datasheet/stm32wb15cc.pdf

https://www.st.com/en/evaluation-tools/nucleo-wb15cc.html

https://www.st.com/resource/en/schematic_pack/mb1641-wb15cc-c01_schematic.pdf

https://community.st.com/t5/st25-nfc-rfid-tags-and-readers/how-to-use-nucleo-l4r5zi-with-x-nucleo-nfc08a1/td-p/612752

https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group1/ed/64/c3/6d/04/fc/47/5b/mb1641_schematic/files/MB1641-WB15CC-C01_Schematic.pdf/jcr:content/translations/en.MB1641-WB15CC-C01_Schematic.pdf

https://www.st.com/resource/en/user_manual/um2823-stm32wb-nucleo64-board-mb1641-stmicroelectronics.pdf

https://www.st.com/en/ecosystems/x-nucleo-nfc08a1.html

https://www.st.com/content/ccc/resource/technical/layouts_and_diagrams/schematic_pack/group2/d1/0e/47/cf/0f/26/4c/fb/X-NUCLEO-NFC08A1_SCHEMATIC/files/x-nucleo-nfc08a1-schematic.pdf/jcr:content/translations/en.x-nucleo-nfc08a1-schematic.pdf

https://www.st.com/resource/en/data_brief/x-nucleo-nfc08a1.pdf

https://www.st.com/resource/en/user_manual/um3007-getting-started-with-the-nfc-card-reader-expansion-board-based-on-st25r3916b-for-stm32-and-stm8-nucleos-stmicroelectronics.pdf

https://wiki.st.com/stm32mcu/wiki/Connectivity:STM32WB_BLE_STM32CubeMX#ST_BLE_toolbox-_P2P_server_notification_and_write

https://novelbits.io/bluetooth-low-energy-ble-complete-guide/

https://developer.android.com/develop/connectivity/bluetooth/ble/ble-overview

Appendix

Code explicitly written by us is placed between USER … BEGIN and USER … END. In addition, parts of the code generated by STM32CubeMX were modified to work according to our project.

main.c

```
int main(void)
{
 /* USER CODE BEGIN 1 */
 /* USER CODE END 1 */
 /* MCU Configuration--------------------------------------------------------*/
 /* Reset of all peripherals, Initializes the Flash interface and the Systick.
*/
 HAL_Init();
 /* Config code for STM32_WPAN (HSE Tuning must be done before system clock
configuration) */
 MX_APPE_Config();
 /* USER CODE BEGIN Init */
 /* USER CODE END Init */
 /* Configure the system clock */
 SystemClock_Config();
/* Configure the peripherals common clocks */
 PeriphCommonClock_Config();
 /* IPCC initialisation */
 MX_IPCC_Init();
 /* Initialize all configured peripherals */
 MX_GPIO_Init();
 MX_DMA_Init();
 MX_RTC_Init();
 MX_RF_Init();
 /* USER CODE BEGIN 2 */
 HAL_GPIO_TogglePin (GPIOA, GPIO_PIN_11);
 /* USER CODE END 2 */
 /* Init code for STM32_WPAN */
 MX_APPE_Init();
 /* Infinite loop */
 /* USER CODE BEGIN WHILE */
 while (1)
 {
   /* USER CODE END WHILE */
   MX_APPE_Process();
   /* USER CODE BEGIN 3 */
 }
 /* USER CODE END 3 */
}
```

app_entry.c

```
void MX_APPE_Config(void)
{
 /**
  * The OPTVERR flag is wrongly set at power on
  * It shall be cleared before using any HAL_FLASH_xxx() api
  */
 __HAL_FLASH_CLEAR_FLAG(FLASH_FLAG_OPTVERR);
 /**
  * Reset some configurations so that the system behave in the same way
  * when either out of nReset or Power On
  */
 Reset_Device();
 /* Configure HSE Tuning */
 Config_HSE();
 return;
}

void MX_APPE_Init(void)
{
 System_Init();       /**< System initialization */
 SystemPower_Config(); /**< Configure the system Power Mode */
 HW_TS_Init(hw_ts_InitMode_Full, &hrtc); /**< Initialize the TimerServer */
/* USER CODE BEGIN APPE_Init_1 */
 APPD_Init();
/* USER CODE END APPE_Init_1 */
 appe_Tl_Init();  /* Initialize all transport layers */
 /**
  * From now, the application is waiting for the ready event (VS_HCI_C2_Ready)
  * received on the system channel before starting the Stack
  * This system event is received with APPE_SysUserEvtRx()
  */
/* USER CODE BEGIN APPE_Init_2 */
/* USER CODE END APPE_Init_2 */
  return;
}
 void MX_APPE_Process(void)

{
 /* USER CODE BEGIN MX_APPE_Process_1 */
 /* USER CODE END MX_APPE_Process_1 */
 UTIL_SEQ_Run(UTIL_SEQ_DEFAULT);
 /* USER CODE BEGIN MX_APPE_Process_2 */
 /* USER CODE END MX_APPE_Process_2 */
}
```

app_debug.c

```
/* Private function prototypes
-----------------------------------------------*/
/* USER CODE BEGIN PFP */
static void APPD_SetCPU2GpioConfig( void );
static void APPD_BleDtbCfg( void );
/* USER CODE END PFP */
/* Functions Definition
------------------------------------------------------*/
void APPD_EnableCPU2( void )
{
/* USER CODE BEGIN APPD_EnableCPU2 */
 APPD_GeneralConfig.STBY_DebugGpioaPinList = STBY_DebugGpioaPinList;
 APPD_GeneralConfig.STBY_DebugGpiobPinList = STBY_DebugGpiobPinList;
 APPD_GeneralConfig.STBY_DebugGpiocPinList = STBY_DebugGpiocPinList;
 APPD_GeneralConfig.STBY_DtbGpioaPinList = STBY_DtbGpioaPinList;
 APPD_GeneralConfig.STBY_DtbGpiobPinList = STBY_DtbGpiobPinList;
 SHCI_C2_DEBUG_Init_Cmd_Packet_t DebugCmdPacket =
 {
   {{0,0,0}},                            /**< Does not need to be initialized
*/
   {(uint8_t *)aGpioConfigList,
   (uint8_t *)&APPD_TracesConfig,
   (uint8_t *)&APPD_GeneralConfig,
   GPIO_CFG_NBR_OF_FEATURES,
   NBR_OF_TRACES_CONFIG_PARAMETERS,
   NBR_OF_GENERAL_CONFIG_PARAMETERS}
 };
 /**< Traces channel initialization */
 TL_TRACES_Init( );
 /** GPIO DEBUG Initialization */
 SHCI_C2_DEBUG_Init( &DebugCmdPacket  );
/* USER CODE END APPD_EnableCPU2 */
 return;
}
/**********************************************************
*
* LOCAL FUNCTIONS
*
***********************************************************/
static void APPD_BleDtbCfg( void )
{
/* USER CODE BEGIN APPD_BleDtbCfg */
 STBY_DtbGpioaPinList = 0;
 STBY_DtbGpiobPinList = 0;
/* USER CODE END APPD_BleDtbCfg */
 return;
```

```
}
/**************************************************************
*
* WRAP FUNCTIONS
*
**************************************************************/
#if(CFG_DEBUG_TRACE != 0)
void DbgOutputInit( void )
{
/* USER CODE BEGIN DbgOutputInit */
#ifdef CFG_DEBUG_TRACE_UART
if (CFG_DEBUG_TRACE_UART == hw_lpuart1)
{
#if(CFG_HW_LPUART1_ENABLED == 1)
   MX_LPUART1_UART_Init();
#endif
}
else if (CFG_DEBUG_TRACE_UART == hw_uart1)
{
#if(CFG_HW_USART1_ENABLED == 1)
   MX_USART1_UART_Init();
#endif
}
#endif
/* USER CODE END DbgOutputInit */
 return;
}
void DbgOutputTraces(  uint8_t *p_data, uint16_t size, void (*cb)(void) )
{
/* USER CODE END DbgOutputTraces */
 HW_UART_Transmit_DMA(CFG_DEBUG_TRACE_UART, p_data, size, cb);
/* USER CODE END DbgOutputTraces */
 return;
}
#endif
```

app_ble.c

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file    App/app_ble.c
 * @author  MCD Application Team
 * @brief   BLE Application
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2023 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */
/* USER CODE END Header */
/* Includes
------------------------------------------------------------*/
#include "main.h"
#include "app_common.h"
#include "dbg_trace.h"
#include "ble.h"
#include "tl.h"
#include "app_ble.h"
#include "stm32_seq.h"
#include "shci.h"
#include "stm32_lpm.h"
#include "otp.h"
#include "custom_app.h"

/* Private defines
------------------------------------------------------------*/
#define FAST_ADV_TIMEOUT               (30*1000*1000/CFG_TS_TICK_VAL) /**< 30s
*/
#define INITIAL_ADV_TIMEOUT            (60*1000*1000/CFG_TS_TICK_VAL) /**< 60s
*/
#define BD_ADDR_SIZE_LOCAL    6
/* USER CODE BEGIN PD */
/* USER CODE END PD */
/* Private macro
------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
```

```
/* Private variables
-----------------------------------------------------------*/
/**
* Advertising Data
*/
uint8_t a_AdvData[19] =
{
 2, AD_TYPE_TX_POWER_LEVEL, 0 /* -0.15dBm */, /* Transmission Power */
 10, AD_TYPE_COMPLETE_LOCAL_NAME, 'B', 'L', 'E', '_', 'D', 'L', 'O', 'C', 'K',
/* Complete name */
 4, AD_TYPE_MANUFACTURER_SPECIFIC_DATA, 0x30, 0x00, 0x00 /*  */,
};
/* USER CODE BEGIN PV */
uint8_t prev_addr_ble = 0x00;
/* USER CODE END PV */
/* Private function prototypes
-----------------------------------------------*/
static void BLE_UserEvtRx(void *p_Payload);
static void BLE_StatusNot(HCI_TL_CmdStatus_t Status);
static void Ble_Tl_Init(void);
static void Ble_Hci_Gap_Gatt_Init(void);
static const uint8_t* BleGetBdAddress(void);
static void Adv_Request(APP_BLE_ConnStatus_t NewStatus);
static void Adv_Cancel(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* External variables
-----------------------------------------------------------*/
/* USER CODE BEGIN EV */
/* USER CODE END EV */
/* Functions Definition
-----------------------------------------------------------*/
void APP_BLE_Init(void)
{
 SHCI_CmdStatus_t status;
#if (RADIO_ACTIVITY_EVENT != 0)
 tBleStatus ret = BLE_STATUS_INVALID_PARAMS;
#endif /* RADIO_ACTIVITY_EVENT != 0 */
 /* USER CODE BEGIN APP_BLE_Init_1 */
 /* USER CODE END APP_BLE_Init_1 */
 SHCI_C2_Ble_Init_Cmd_Packet_t ble_init_cmd_packet =
 {
   {{0,0,0}},                          /**< Header unused */
   {0,                                 /** pBleBufferAddress not used */
    0,                                 /** BleBufferSize not used */
    CFG_BLE_NUM_GATT_ATTRIBUTES,
    CFG_BLE_NUM_GATT_SERVICES,
    CFG_BLE_ATT_VALUE_ARRAY_SIZE,
```

```
    CFG_BLE_NUM_LINK,
    CFG_BLE_DATA_LENGTH_EXTENSION,
    CFG_BLE_PREPARE_WRITE_LIST_SIZE,
    CFG_BLE_MBLOCK_COUNT,
    CFG_BLE_MAX_ATT_MTU,
    CFG_BLE_PERIPHERAL_SCA,
    CFG_BLE_CENTRAL_SCA,
    CFG_BLE_LS_SOURCE,
    CFG_BLE_MAX_CONN_EVENT_LENGTH,
    CFG_BLE_HSE_STARTUP_TIME,
    CFG_BLE_VITERBI_MODE,
    CFG_BLE_OPTIONS,
    0,
    CFG_BLE_MAX_COC_INITIATOR_NBR,
    CFG_BLE_MIN_TX_POWER,
    CFG_BLE_MAX_TX_POWER,
    CFG_BLE_RX_MODEL_CONFIG,
    CFG_BLE_MAX_ADV_SET_NBR,
    CFG_BLE_MAX_ADV_DATA_LEN,
    CFG_BLE_TX_PATH_COMPENS,
    CFG_BLE_RX_PATH_COMPENS,
    CFG_BLE_CORE_VERSION,
    CFG_BLE_OPTIONS_EXT
   }
 };
 /**
  * Initialize Ble Transport Layer
  */
 Ble_Tl_Init();
#if (CFG_LPM_STANDBY_SUPPORTED == 0)
 UTIL_LPM_SetOffMode(1U << CFG_LPM_APP_BLE, UTIL_LPM_DISABLE);
#endif /* CFG_LPM_STANDBY_SUPPORTED == 0 */
 /**
  * Register the hci transport layer to handle BLE User Asynchronous Events
  */
 UTIL_SEQ_RegTask(1<<CFG_TASK_HCI_ASYNCH_EVT_ID, UTIL_SEQ_RFU,
hci_user_evt_proc);
 /**
  * Starts the BLE Stack on CPU2
  */
 status = SHCI_C2_BLE_Init(&ble_init_cmd_packet);
 if (status != SHCI_Success)
 {
   APP_DBG_MSG("  Fail   : SHCI_C2_BLE_Init command, result: 0x%02x\n\r",
status);
   /* if you are here, maybe CPU2 doesn't contain
STM32WB_Copro_Wireless_Binaries, see Release_Notes.html */
   Error_Handler();
```

```
 }
 else
 {
   APP_DBG_MSG("  Success: SHCI_C2_BLE_Init command\n\r");
 }
 /**
  * Initialization of HCI & GATT & GAP layer
  */
 Ble_Hci_Gap_Gatt_Init();
 /**
  * Initialization of the BLE Services
  */
 SVCCTL_Init();
 /**
  * Initialization of the BLE App Context
  */
 BleApplicationContext.Device_Connection_Status = APP_BLE_IDLE;
 BleApplicationContext.BleApplicationContext_legacy.connectionHandle = 0xFFFF;
 /**
  * From here, all initialization are BLE application specific
  */
 UTIL_SEQ_RegTask(1<<CFG_TASK_ADV_CANCEL_ID, UTIL_SEQ_RFU, Adv_Cancel);
 /* USER CODE BEGIN APP_BLE_Init_4 */
 /* USER CODE END APP_BLE_Init_4 */
 /**
  * Initialization of ADV - Ad Manufacturer Element - Support OTA Bit Mask
  */
#if (RADIO_ACTIVITY_EVENT != 0)
 ret = aci_hal_set_radio_activity_mask(0x0006);
 if (ret != BLE_STATUS_SUCCESS)
 {
   APP_DBG_MSG("  Fail   : aci_hal_set_radio_activity_mask command, result:
0x%x \n\r", ret);
 }
 else
 {
   APP_DBG_MSG("  Success: aci_hal_set_radio_activity_mask command\n\r");
 }
#endif /* RADIO_ACTIVITY_EVENT != 0 */
 /**
  * Initialize Custom Template Application
  */
 Custom_APP_Init();
 /* USER CODE BEGIN APP_BLE_Init_3 */
 /* USER CODE END APP_BLE_Init_3 */
 /**
  * Make device discoverable
  */
```

```
 BleApplicationContext.BleApplicationContext_legacy.advtServUUID[0] = NULL;
 BleApplicationContext.BleApplicationContext_legacy.advtServUUIDlen = 0;
 /**
  * Start to Advertise to be connected by a Client
  */
 Adv_Request(APP_BLE_FAST_ADV);
 /* USER CODE BEGIN APP_BLE_Init_2 */
 /* USER CODE END APP_BLE_Init_2 */
 return;
}
SVCCTL_UserEvtFlowStatus_t SVCCTL_App_Notification(void *p_Pckt)
{
 hci_event_pckt    *p_event_pckt;
 evt_le_meta_event *p_meta_evt;
 evt_blecore_aci   *p_blecore_evt;
 tBleStatus        ret = BLE_STATUS_INVALID_PARAMS;
 hci_le_connection_complete_event_rp0        *p_connection_complete_event;
 hci_disconnection_complete_event_rp0        *p_disconnection_complete_event;
#if (CFG_DEBUG_APP_TRACE != 0)
 hci_le_connection_update_complete_event_rp0
*p_connection_update_complete_event;
#endif /* CFG_DEBUG_APP_TRACE != 0 */
 /* PAIRING */
 aci_gap_pairing_complete_event_rp0          *p_pairing_complete;
 /* PAIRING */
 /* USER CODE BEGIN SVCCTL_App_Notification */
 /* USER CODE END SVCCTL_App_Notification */
 p_event_pckt = (hci_event_pckt*) ((hci_uart_pckt *) p_Pckt)->data;
 switch (p_event_pckt->evt)
 {
   case HCI_DISCONNECTION_COMPLETE_EVT_CODE:
   {
     p_disconnection_complete_event = (hci_disconnection_complete_event_rp0 *)
p_event_pckt->data;
     if (p_disconnection_complete_event->Connection_Handle ==
BleApplicationContext.BleApplicationContext_legacy.connectionHandle)
     {
       BleApplicationContext.BleApplicationContext_legacy.connectionHandle = 0;
       BleApplicationContext.Device_Connection_Status = APP_BLE_IDLE;
       APP_DBG_MSG(">>== HCI_DISCONNECTION_COMPLETE_EVT_CODE\n");
       APP_DBG_MSG("     - Connection Handle:   0x%x\n     - Reason:
0x%x\n\r",
                   p_disconnection_complete_event->Connection_Handle,
                   p_disconnection_complete_event->Reason);
       /* USER CODE BEGIN EVT_DISCONN_COMPLETE_2 */
       /* USER CODE END EVT_DISCONN_COMPLETE_2 */
     }
     /* USER CODE BEGIN EVT_DISCONN_COMPLETE_1 */
```

```
      /* USER CODE END EVT_DISCONN_COMPLETE_1 */
      /* restart advertising */
      Adv_Request(APP_BLE_FAST_ADV);
      /**
       * SPECIFIC to Custom Template APP
       */
      HandleNotification.Custom_Evt_Opcode = CUSTOM_DISCON_HANDLE_EVT;
      HandleNotification.ConnectionHandle =
BleApplicationContext.BleApplicationContext_legacy.connectionHandle;
      Custom_APP_Notification(&HandleNotification);
      /* USER CODE BEGIN EVT_DISCONN_COMPLETE */
      /* USER CODE END EVT_DISCONN_COMPLETE */
      break; /* HCI_DISCONNECTION_COMPLETE_EVT_CODE */
    }
    case HCI_LE_META_EVT_CODE:
    {
      p_meta_evt = (evt_le_meta_event*) p_event_pckt->data;
      /* USER CODE BEGIN EVT_LE_META_EVENT */
      /* USER CODE END EVT_LE_META_EVENT */
      switch (p_meta_evt->subevent)
      {
        case HCI_LE_CONNECTION_UPDATE_COMPLETE_SUBEVT_CODE:
#if (CFG_DEBUG_APP_TRACE != 0)
          p_connection_update_complete_event =
(hci_le_connection_update_complete_event_rp0 *) p_meta_evt->data;
          APP_DBG_MSG(">>== HCI_LE_CONNECTION_UPDATE_COMPLETE_SUBEVT_CODE\n");
          APP_DBG_MSG("     - Connection Interval:   %.2f ms\n     - Connection
latency:    %d\n     - Supervision Timeout: %d ms\n\r",
                      p_connection_update_complete_event->Conn_Interval*1.25,
                      p_connection_update_complete_event->Conn_Latency,

p_connection_update_complete_event->Supervision_Timeout*10);
#endif /* CFG_DEBUG_APP_TRACE != 0 */
          /* USER CODE BEGIN EVT_LE_CONN_UPDATE_COMPLETE */
          /* USER CODE END EVT_LE_CONN_UPDATE_COMPLETE */
          break;
        case HCI_LE_CONNECTION_COMPLETE_SUBEVT_CODE:
        {
          p_connection_complete_event = (hci_le_connection_complete_event_rp0 *)
p_meta_evt->data;
          /**
           * The connection is done, there is no need anymore to schedule the LP
ADV
           */
          APP_DBG_MSG(">>== HCI_LE_CONNECTION_COMPLETE_SUBEVT_CODE - Connection
handle: 0x%x\n", p_connection_complete_event->Connection_Handle);
          APP_DBG_MSG("     - Connection established with Central:
@:%02x:%02x:%02x:%02x:%02x:%02x\n",
```

```
                    p_connection_complete_event->Peer_Address[5],
                    p_connection_complete_event->Peer_Address[4],
                    p_connection_complete_event->Peer_Address[3],
                    p_connection_complete_event->Peer_Address[2],
                    p_connection_complete_event->Peer_Address[1],
                    p_connection_complete_event->Peer_Address[0]);
          APP_DBG_MSG("     - Connection Interval:   %.2f ms\n     - Connection
latency:    %d\n     - Supervision Timeout: %d ms\n\r",
                    p_connection_complete_event->Conn_Interval*1.25,
                    p_connection_complete_event->Conn_Latency,
                    p_connection_complete_event->Supervision_Timeout*10
                   );
          if (BleApplicationContext.Device_Connection_Status ==
APP_BLE_LP_CONNECTING)
          {
            /* Connection as client */
            BleApplicationContext.Device_Connection_Status =
APP_BLE_CONNECTED_CLIENT;
          }
          else
          {
            /* Connection as server */
            BleApplicationContext.Device_Connection_Status =
APP_BLE_CONNECTED_SERVER;
          }
          BleApplicationContext.BleApplicationContext_legacy.connectionHandle =
p_connection_complete_event->Connection_Handle;
          /**
           * SPECIFIC to Custom Template APP
           */
          HandleNotification.Custom_Evt_Opcode = CUSTOM_CONN_HANDLE_EVT;
          HandleNotification.ConnectionHandle =
BleApplicationContext.BleApplicationContext_legacy.connectionHandle;
          Custom_APP_Notification(&HandleNotification);
          /* USER CODE BEGIN HCI_EVT_LE_CONN_COMPLETE */
          if(p_connection_complete_event->Peer_Address[5] == prev_addr_ble &&
unlocked_before == 1){
                APP_DBG_MSG("\r\n\r** ICard Access Found \n");
                                    HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_0);
                                    HAL_GPIO_TogglePin (GPIOA, GPIO_PIN_12);
                                    HAL_Delay (1000);   /* Insert delay 2000 ms
*/
                                    APP_DBG_MSG("\r\n\r** Door Unlocked \n");
                                    HAL_GPIO_TogglePin (GPIOA, GPIO_PIN_12);
                    //              HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_0);
                                    HAL_Delay(3000);
                                    HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_0);
                                    HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_1);
```

```
                                    HAL_GPIO_TogglePin (GPIOA, GPIO_PIN_11);
                                    HAL_Delay (2000);   /* Insert delay 2000 ms
*/
                                    APP_DBG_MSG("\r\n\r** Door Locked \n");
                                    HAL_GPIO_TogglePin (GPIOA, GPIO_PIN_11);
                                    HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_1);
        }else unlocked_before = 0;
        prev_addr_ble = p_connection_complete_event->Peer_Address[5];
        /* USER CODE END HCI_EVT_LE_CONN_COMPLETE */
        break; /* HCI_LE_CONNECTION_COMPLETE_SUBEVT_CODE */
      }
      default:
        /* USER CODE BEGIN SUBEVENT_DEFAULT */
        /* USER CODE END SUBEVENT_DEFAULT */
        break;
    }
    /* USER CODE BEGIN META_EVT */
    /* USER CODE END META_EVT */
    break; /* HCI_LE_META_EVT_CODE */
  }
  case HCI_VENDOR_SPECIFIC_DEBUG_EVT_CODE:
    p_blecore_evt = (evt_blecore_aci*) p_event_pckt->data;
    /* USER CODE BEGIN EVT_VENDOR */
    /* USER CODE END EVT_VENDOR */
    switch (p_blecore_evt->ecode)
    {
      /* USER CODE BEGIN ecode */
      /* USER CODE END ecode */
      /**
       * SPECIFIC to Custom Template APP
       */
      case ACI_L2CAP_CONNECTION_UPDATE_RESP_VSEVT_CODE:
        /* USER CODE BEGIN EVT_BLUE_L2CAP_CONNECTION_UPDATE_RESP */
        /* USER CODE END EVT_BLUE_L2CAP_CONNECTION_UPDATE_RESP */
        break;
      case ACI_GAP_PROC_COMPLETE_VSEVT_CODE:
        APP_DBG_MSG(">>== ACI_GAP_PROC_COMPLETE_VSEVT_CODE \r");
        /* USER CODE BEGIN EVT_BLUE_GAP_PROCEDURE_COMPLETE */
        /* USER CODE END EVT_BLUE_GAP_PROCEDURE_COMPLETE */
        break; /* ACI_GAP_PROC_COMPLETE_VSEVT_CODE */
#if (RADIO_ACTIVITY_EVENT != 0)
      case ACI_HAL_END_OF_RADIO_ACTIVITY_VSEVT_CODE:
        /* USER CODE BEGIN RADIO_ACTIVITY_EVENT*/
        /* USER CODE END RADIO_ACTIVITY_EVENT*/
        break; /* ACI_HAL_END_OF_RADIO_ACTIVITY_VSEVT_CODE */
#endif /* RADIO_ACTIVITY_EVENT != 0 */
      /* PAIRING */
      case (ACI_GAP_KEYPRESS_NOTIFICATION_VSEVT_CODE):
```

```
          APP_DBG_MSG(">>== ACI_GAP_KEYPRESS_NOTIFICATION_VSEVT_CODE\n");
          /* USER CODE BEGIN ACI_GAP_KEYPRESS_NOTIFICATION_VSEVT_CODE*/
          /* USER CODE END ACI_GAP_KEYPRESS_NOTIFICATION_VSEVT_CODE*/
          break;
        case ACI_GAP_PASS_KEY_REQ_VSEVT_CODE:
          APP_DBG_MSG(">>== ACI_GAP_PASS_KEY_REQ_VSEVT_CODE \n");
          ret =
aci_gap_pass_key_resp(BleApplicationContext.BleApplicationContext_legacy.connec
tionHandle, CFG_FIXED_PIN);
          if (ret != BLE_STATUS_SUCCESS)
          {
            APP_DBG_MSG("==>> aci_gap_pass_key_resp : Fail, reason: 0x%x\n",
ret);
          }
          else
          {
            APP_DBG_MSG("==>> aci_gap_pass_key_resp : Success \n");
          }
          /* USER CODE BEGIN ACI_GAP_PASS_KEY_REQ_VSEVT_CODE*/
          /* USER CODE END ACI_GAP_PASS_KEY_REQ_VSEVT_CODE*/
          break;
        case ACI_GAP_NUMERIC_COMPARISON_VALUE_VSEVT_CODE:
          APP_DBG_MSG(">>== ACI_GAP_NUMERIC_COMPARISON_VALUE_VSEVT_CODE\n");
          APP_DBG_MSG("     - numeric_value = %ld\n",
                      ((aci_gap_numeric_comparison_value_event_rp0
*)(p_blecore_evt->data))->Numeric_Value);
          APP_DBG_MSG("     - Hex_value = %lx\n",
                      ((aci_gap_numeric_comparison_value_event_rp0
*)(p_blecore_evt->data))->Numeric_Value);
          ret =
aci_gap_numeric_comparison_value_confirm_yesno(BleApplicationContext.BleApplica
tionContext_legacy.connectionHandle, YES);
          if (ret != BLE_STATUS_SUCCESS)
          {
            APP_DBG_MSG("==>>
aci_gap_numeric_comparison_value_confirm_yesno-->YES : Fail, reason: 0x%x\n",
ret);
          }
          else
          {
            APP_DBG_MSG("==>>
aci_gap_numeric_comparison_value_confirm_yesno-->YES : Success \n");
          }
          /* USER CODE BEGIN ACI_GAP_NUMERIC_COMPARISON_VALUE_VSEVT_CODE*/
          /* USER CODE END ACI_GAP_NUMERIC_COMPARISON_VALUE_VSEVT_CODE*/
          break;
        case ACI_GAP_PAIRING_COMPLETE_VSEVT_CODE:
```

```
          p_pairing_complete =
(aci_gap_pairing_complete_event_rp0*)p_blecore_evt->data;
          APP_DBG_MSG(">>== ACI_GAP_PAIRING_COMPLETE_VSEVT_CODE\n");
          if (p_pairing_complete->Status != 0)
          {
            APP_DBG_MSG("     - Pairing KO \n     - Status: 0x%x\n     - Reason:
0x%x\n", p_pairing_complete->Status, p_pairing_complete->Reason);
          }
          else
          {
            APP_DBG_MSG("     - Pairing Success\n");
          }
          APP_DBG_MSG("\n");
          /* USER CODE BEGIN ACI_GAP_PAIRING_COMPLETE_VSEVT_CODE*/
          /* USER CODE END ACI_GAP_PAIRING_COMPLETE_VSEVT_CODE*/
          break;
        /* PAIRING */
        /* USER CODE BEGIN BLUE_EVT */
        /* USER CODE END BLUE_EVT */
      }
      break; /* HCI_VENDOR_SPECIFIC_DEBUG_EVT_CODE */
      /* USER CODE BEGIN EVENT_PCKT */
      /* USER CODE END EVENT_PCKT */
    default:
      /* USER CODE BEGIN ECODE_DEFAULT*/
      /* USER CODE END ECODE_DEFAULT*/
      break;
  }
  return (SVCCTL_UserEvtFlowEnable);
}
APP_BLE_ConnStatus_t APP_BLE_Get_Server_Connection_Status(void)
{
  return BleApplicationContext.Device_Connection_Status;
}
/* USER CODE BEGIN FD*/
/* USER CODE END FD*/
/*************************************************************
*
* LOCAL FUNCTIONS
*
*************************************************************/
static void Ble_Tl_Init(void)
{
  HCI_TL_HciInitConf_t Hci_Tl_Init_Conf;
  Hci_Tl_Init_Conf.p_cmdbuffer = (uint8_t*)&BleCmdBuffer;
  Hci_Tl_Init_Conf.StatusNotCallBack = BLE_StatusNot;
  hci_init(BLE_UserEvtRx, (void*) &Hci_Tl_Init_Conf);
  return;
```

```
}
static void Ble_Hci_Gap_Gatt_Init(void)
{
 uint8_t role;
 uint16_t gap_service_handle, gap_dev_name_char_handle,
gap_appearance_char_handle;
 const uint8_t *p_bd_addr;
 uint16_t a_appearance[1] = {BLE_CFG_GAP_APPEARANCE};
 tBleStatus ret = BLE_STATUS_INVALID_PARAMS;
 /* USER CODE BEGIN Ble_Hci_Gap_Gatt_Init*/
 /* USER CODE END Ble_Hci_Gap_Gatt_Init*/
 APP_DBG_MSG("==>> Start Ble_Hci_Gap_Gatt_Init function\n");
 /**
  * Initialize HCI layer
  */
 /*HCI Reset to synchronise BLE Stack*/
 ret = hci_reset();
 if (ret != BLE_STATUS_SUCCESS)
 {
   APP_DBG_MSG("  Fail   : hci_reset command, result: 0x%x \n", ret);
 }
 else
 {
   APP_DBG_MSG("  Success: hci_reset command\n");
 }
 /**
  * Write the BD Address
  */
 p_bd_addr = BleGetBdAddress();
 ret = aci_hal_write_config_data(CONFIG_DATA_PUBADDR_OFFSET,
CONFIG_DATA_PUBADDR_LEN, (uint8_t*) p_bd_addr);
 if (ret != BLE_STATUS_SUCCESS)
 {
   APP_DBG_MSG("  Fail   : aci_hal_write_config_data command -
CONFIG_DATA_PUBADDR_OFFSET, result: 0x%x \n", ret);
 }
 else
 {
   APP_DBG_MSG("  Success: aci_hal_write_config_data command -
CONFIG_DATA_PUBADDR_OFFSET\n");
   APP_DBG_MSG("  Public Bluetooth Address:
%02x:%02x:%02x:%02x:%02x:%02x\n",p_bd_addr[5],p_bd_addr[4],p_bd_addr[3],p_bd_ad
dr[2],p_bd_addr[1],p_bd_addr[0]);
 }
  /**
  * Write Identity root key used to derive IRK and DHK(Legacy)
  */
```

```
 ret = aci_hal_write_config_data(CONFIG_DATA_IR_OFFSET, CONFIG_DATA_IR_LEN,
(uint8_t*)a_BLE_CfgIrValue);
 if (ret != BLE_STATUS_SUCCESS)
 {
   APP_DBG_MSG("  Fail   : aci_hal_write_config_data command -
CONFIG_DATA_IR_OFFSET, result: 0x%x \n", ret);
 }
 else
 {
   APP_DBG_MSG("  Success: aci_hal_write_config_data command -
CONFIG_DATA_IR_OFFSET\n");
 }
 /**
  * Write Encryption root key used to derive LTK and CSRK
  */
 ret = aci_hal_write_config_data(CONFIG_DATA_ER_OFFSET, CONFIG_DATA_ER_LEN,
(uint8_t*)a_BLE_CfgErValue);
 if (ret != BLE_STATUS_SUCCESS)
 {
   APP_DBG_MSG("  Fail   : aci_hal_write_config_data command -
CONFIG_DATA_ER_OFFSET, result: 0x%x \n", ret);
 }
 else
 {
   APP_DBG_MSG("  Success: aci_hal_write_config_data command -
CONFIG_DATA_ER_OFFSET\n");
 }
 /**
  * Set TX Power.
  */
 ret = aci_hal_set_tx_power_level(1, CFG_TX_POWER);
 if (ret != BLE_STATUS_SUCCESS)
 {
   APP_DBG_MSG("  Fail   : aci_hal_set_tx_power_level command, result: 0x%x
\n", ret);
 }
 else
 {
   APP_DBG_MSG("  Success: aci_hal_set_tx_power_level command\n");
 }
 /**
  * Initialize GATT interface
  */
 ret = aci_gatt_init();
 if (ret != BLE_STATUS_SUCCESS)
 {
   APP_DBG_MSG("  Fail   : aci_gatt_init command, result: 0x%x \n", ret);
 }
```

```
 else
 {
   APP_DBG_MSG("  Success: aci_gatt_init command\n");
 }
 /**
  * Initialize GAP interface
  */
 role = 0;
#if (BLE_CFG_PERIPHERAL == 1)
 role |= GAP_PERIPHERAL_ROLE;
#endif /* BLE_CFG_PERIPHERAL == 1 */
/* USER CODE BEGIN Role_Mngt*/
/* USER CODE END Role_Mngt */
 if (role > 0)
 {
   const char *name = CFG_GAP_DEVICE_NAME;
   ret = aci_gap_init(role,
                      CFG_PRIVACY,
                      CFG_GAP_DEVICE_NAME_LENGTH,
                      &gap_service_handle,
                      &gap_dev_name_char_handle,
                      &gap_appearance_char_handle);
   if (ret != BLE_STATUS_SUCCESS)
   {
     APP_DBG_MSG("  Fail   : aci_gap_init command, result: 0x%x \n", ret);
   }
   else
   {
     APP_DBG_MSG("  Success: aci_gap_init command\n");
   }
   ret = aci_gatt_update_char_value(gap_service_handle,
gap_dev_name_char_handle, 0, strlen(name), (uint8_t *) name);
   if (ret != BLE_STATUS_SUCCESS)
   {
     BLE_DBG_SVCCTL_MSG("  Fail   : aci_gatt_update_char_value - Device
Name\n");
   }
   else
   {
     BLE_DBG_SVCCTL_MSG("  Success: aci_gatt_update_char_value - Device
Name\n");
   }
 }
 ret = aci_gatt_update_char_value(gap_service_handle,
                                  gap_appearance_char_handle,
                                  0,
                                  2,
                                  (uint8_t *)&a_appearance);
```

```
 if (ret != BLE_STATUS_SUCCESS)
 {
   BLE_DBG_SVCCTL_MSG("  Fail   : aci_gatt_update_char_value - Appearance\n");
 }
 else
 {
   BLE_DBG_SVCCTL_MSG("  Success: aci_gatt_update_char_value - Appearance\n");
 }
 /**
  * Initialize Default PHY
  */
 ret =
hci_le_set_default_phy(ALL_PHYS_PREFERENCE,TX_2M_PREFERRED,RX_2M_PREFERRED);
 if (ret != BLE_STATUS_SUCCESS)
 {
   APP_DBG_MSG("  Fail   : hci_le_set_default_phy command, result: 0x%x \n",
ret);
 }
 else
 {
   APP_DBG_MSG("  Success: hci_le_set_default_phy command\n");
 }
 /**
  * Initialize IO capability
  */

BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.ioCapabilit
y = CFG_IO_CAPABILITY;
 ret =
aci_gap_set_io_capability(BleApplicationContext.BleApplicationContext_legacy.bl
eSecurityParam.ioCapability);
 if (ret != BLE_STATUS_SUCCESS)
 {
   APP_DBG_MSG("  Fail   : aci_gap_set_io_capability command, result: 0x%x \n",
ret);
 }
 else
 {
   APP_DBG_MSG("  Success: aci_gap_set_io_capability command\n");
 }
 /**
  * Initialize authentication
  */
 BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.mitm_mode
= CFG_MITM_PROTECTION;

BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.encryptionK
eySizeMin = CFG_ENCRYPTION_KEY_SIZE_MIN;
```

```
BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.encryptionK
eySizeMax = CFG_ENCRYPTION_KEY_SIZE_MAX;

BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.Use_Fixed_P
in = CFG_USED_FIXED_PIN;
 BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.Fixed_Pin
= CFG_FIXED_PIN;

BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.bonding_mod
e = CFG_BONDING_MODE;
 /* USER CODE BEGIN Ble_Hci_Gap_Gatt_Init_1*/
 /* USER CODE END Ble_Hci_Gap_Gatt_Init_1*/
 ret =
aci_gap_set_authentication_requirement(BleApplicationContext.BleApplicationCont
ext_legacy.bleSecurityParam.bonding_mode,

BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.mitm_mode,
                                            CFG_SC_SUPPORT,

CFG_KEYPRESS_NOTIFICATION_SUPPORT,

BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.encryptionK
eySizeMin,

BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.encryptionK
eySizeMax,

BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.Use_Fixed_P
in,

BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.Fixed_Pin,
                                            CFG_IDENTITY_ADDRESS);
 if (ret != BLE_STATUS_SUCCESS)
 {
   APP_DBG_MSG("  Fail   : aci_gap_set_authentication_requirement command,
result: 0x%x \n", ret);
 }
 else
 {
   APP_DBG_MSG("  Success: aci_gap_set_authentication_requirement command\n");
 }
 /**
  * Initialize whitelist
  */
 if
(BleApplicationContext.BleApplicationContext_legacy.bleSecurityParam.bonding_mo
de)
```

```
 {
   ret = aci_gap_configure_whitelist();
   if (ret != BLE_STATUS_SUCCESS)
   {
     APP_DBG_MSG("  Fail   : aci_gap_configure_whitelist command, result: 0x%x
\n", ret);
   }
   else
   {
     APP_DBG_MSG("  Success: aci_gap_configure_whitelist command\n");
   }
 }
 APP_DBG_MSG("==>> End Ble_Hci_Gap_Gatt_Init function\n\r");
}
static void Adv_Request(APP_BLE_ConnStatus_t NewStatus)
{
 tBleStatus ret = BLE_STATUS_INVALID_PARAMS;
 BleApplicationContext.Device_Connection_Status = NewStatus;
 /* Start Fast or Low Power Advertising */
 ret = aci_gap_set_discoverable(ADV_TYPE,
                                CFG_FAST_CONN_ADV_INTERVAL_MIN,
                                CFG_FAST_CONN_ADV_INTERVAL_MAX,
                                CFG_BLE_ADDRESS_TYPE,
                                ADV_FILTER,
                                0,
                                0,
                                0,
                                0,
                                0,
                                0);
 if (ret != BLE_STATUS_SUCCESS)
 {
   APP_DBG_MSG("==>> aci_gap_set_discoverable - fail, result: 0x%x \n", ret);
 }
 else
 {
   APP_DBG_MSG("==>> aci_gap_set_discoverable - Success\n");
 }
/* USER CODE BEGIN Adv_Request_1*/
/* USER CODE END Adv_Request_1*/
 /* Update Advertising data */
 ret = aci_gap_update_adv_data(sizeof(a_AdvData), (uint8_t*) a_AdvData);
 if (ret != BLE_STATUS_SUCCESS)
 {
     APP_DBG_MSG("==>> Start Fast Advertising Failed , result: %d \n\r", ret);
 }
 else
 {
```

```
    APP_DBG_MSG("==>> Success: Start Fast Advertising \n\r");
  }
  return;
}
const uint8_t* BleGetBdAddress(void)
{
  uint8_t *p_otp_addr;
  const uint8_t *p_bd_addr;
  uint32_t udn;
  uint32_t company_id;
  uint32_t device_id;
  udn = LL_FLASH_GetUDN();
  if (udn != 0xFFFFFFFF)
  {
    company_id = LL_FLASH_GetSTCompanyID();
    device_id = LL_FLASH_GetDeviceID();
    /**
     * Public Address with the ST company ID
     * bit[47:24] : 24bits (OUI) equal to the company ID
     * bit[23:16] : Device ID.
     * bit[15:0] : The last 16bits from the UDN
     * Note: In order to use the Public Address in a final product, a dedicated
     * 24bits company ID (OUI) shall be bought.
     */
    a_BdAddrUdn[0] = (uint8_t)(udn & 0x000000FF);
    a_BdAddrUdn[1] = (uint8_t)((udn & 0x0000FF00) >> 8);
    a_BdAddrUdn[2] = (uint8_t)device_id;
    a_BdAddrUdn[3] = (uint8_t)(company_id & 0x000000FF);
    a_BdAddrUdn[4] = (uint8_t)((company_id & 0x0000FF00) >> 8);
    a_BdAddrUdn[5] = (uint8_t)((company_id & 0x00FF0000) >> 16);
    p_bd_addr = (const uint8_t *)a_BdAddrUdn;
  }
  else
  {
    p_otp_addr = OTP_Read(0);
    if (p_otp_addr)
    {
      p_bd_addr = ((OTP_ID0_t*)p_otp_addr)->bd_address;
    }
    else
    {
      p_bd_addr = a_MBdAddr;
    }
  }
  return p_bd_addr;
}
/* USER CODE BEGIN FD_LOCAL_FUNCTION */
/* USER CODE END FD_LOCAL_FUNCTION */
```

```
/*************************************************************
*
* SPECIFIC FUNCTIONS FOR CUSTOM
*
*************************************************************/
static void Adv_Cancel(void)
{
 /* USER CODE BEGIN Adv_Cancel_1 */
 /* USER CODE END Adv_Cancel_1 */
 if (BleApplicationContext.Device_Connection_Status !=
APP_BLE_CONNECTED_SERVER)
 {
   tBleStatus ret = BLE_STATUS_INVALID_PARAMS;
   ret = aci_gap_set_non_discoverable();
   BleApplicationContext.Device_Connection_Status = APP_BLE_IDLE;
   if (ret != BLE_STATUS_SUCCESS)
   {
     APP_DBG_MSG("** STOP ADVERTISING **  Failed \r\n\r");
   }
   else
   {
     APP_DBG_MSG("  \r\n\r");
     APP_DBG_MSG("** STOP ADVERTISING **  \r\n\r");
   }
 }
 /* USER CODE BEGIN Adv_Cancel_2 */
 /* USER CODE END Adv_Cancel_2 */
 return;
}
/* USER CODE BEGIN FD_SPECIFIC_FUNCTIONS */
/* USER CODE END FD_SPECIFIC_FUNCTIONS */
/*************************************************************
*
* WRAP FUNCTIONS
*
*************************************************************/
void hci_notify_asynch_evt(void* p_Data)
{
 UTIL_SEQ_SetTask(1 << CFG_TASK_HCI_ASYNCH_EVT_ID, CFG_SCH_PRIO_0);
 return;
}
void hci_cmd_resp_release(uint32_t Flag)
{
 UTIL_SEQ_SetEvt(1 << CFG_IDLEEVT_HCI_CMD_EVT_RSP_ID);
 return;
}
void hci_cmd_resp_wait(uint32_t Timeout)
{
```

```
 UTIL_SEQ_WaitEvt(1 << CFG_IDLEEVT_HCI_CMD_EVT_RSP_ID);
 return;
}
static void BLE_UserEvtRx(void *p_Payload)
{
 SVCCTL_UserEvtFlowStatus_t svctl_return_status;
 tHCI_UserEvtRxParam *p_param;
 p_param = (tHCI_UserEvtRxParam *)p_Payload;
 svctl_return_status = SVCCTL_UserEvtRx((void *)&(p_param->pckt->evtserial));
 if (svctl_return_status != SVCCTL_UserEvtFlowDisable)
 {
   p_param->status = HCI_TL_UserEventFlow_Enable;
 }
 else
 {
   p_param->status = HCI_TL_UserEventFlow_Disable;
 }
 return;
}
```

app_ble.h

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file    App/app_ble.h
 * @author  MCD Application Team
 * @brief   Header for ble application
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2023 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */
/* USER CODE END Header */
/* Define to prevent recursive inclusion
-------------------------------------*/
#ifndef APP_BLE_H
#define APP_BLE_H
#ifdef __cplusplus
extern "C" {
#endif
/* Includes
------------------------------------------------------------------------*/
#include "hci_tl.h"
/* Private includes
-----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
/* Exported types
-----------------------------------------------------------*/
typedef enum
{
 APP_BLE_IDLE,
 APP_BLE_FAST_ADV,
 APP_BLE_LP_ADV,
 APP_BLE_SCAN,
 APP_BLE_LP_CONNECTING,
 APP_BLE_CONNECTED_SERVER,
 APP_BLE_CONNECTED_CLIENT
} APP_BLE_ConnStatus_t;
/* USER CODE BEGIN ET */
```

```
/* USER CODE END ET */
/* Exported constants
-----------------------------------------------------------*/
/* USER CODE BEGIN EC */
/* USER CODE END EC */
/* External variables
-----------------------------------------------------------*/
/* USER CODE BEGIN EV */
/* USER CODE END EV */
/* Exported macro
-------------------------------------------------------------*/
/* USER CODE BEGIN EM */
/* USER CODE END EM */
/* Exported functions -------------------------------------------*/
void APP_BLE_Init(void);
APP_BLE_ConnStatus_t APP_BLE_Get_Server_Connection_Status(void);
/* USER CODE BEGIN EF */
/* USER CODE END EF */
#ifdef __cplusplus
}
#endif
#endif /*APP_BLE_H */
```

custom_app.c

```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file    App/custom_app.c
 * @author  MCD Application Team
 * @brief   Custom Example Application (Server)
 ******************************************************************************
 * @attention
 *
 * Copyright (c) 2023 STMicroelectronics.
 * All rights reserved.
 *
 * This software is licensed under terms that can be found in the LICENSE file
 * in the root directory of this software component.
 * If no LICENSE file comes with this software, it is provided AS-IS.
 *
 ******************************************************************************
 */
/* USER CODE END Header */
/* Includes
------------------------------------------------------------------*/
#include "main.h"
#include "app_common.h"
#include "dbg_trace.h"
#include "ble.h"
#include "custom_app.h"
#include "custom_stm.h"
#include "stm32_seq.h"
/* Private includes
-----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
/* USER CODE END Includes */
/* Private typedef
-----------------------------------------------------------*/
typedef struct
{
 /* P2P_Server */
 /* HR_long */
 uint8_t               Hr_m_Notification_Status;
 /* USER CODE BEGIN CUSTOM_APP_Context_t */
 /* USER CODE END CUSTOM_APP_Context_t */
 uint16_t              ConnectionHandle;
} Custom_App_Context_t;
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */
```

```
/* Private defines
------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */
/* Private macros
------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */
/* Private variables
------------------------------------------------------*/
/**
* START of Section BLE_APP_CONTEXT
*/
static Custom_App_Context_t Custom_App_Context;
/**
* END of Section BLE_APP_CONTEXT
*/
uint8_t UpdateCharData[247];
uint8_t NotifyCharData[247];
/* USER CODE BEGIN PV */
uint8_t unlocked_before = 0;
/* USER CODE END PV */
/* Private function prototypes
-----------------------------------------------*/
/* P2P_Server */
/* HR_long */
static void Custom_Hr_m_Update_Char(void);
static void Custom_Hr_m_Send_Notification(void);
/* USER CODE BEGIN PFP */
/* USER CODE END PFP */
/* Functions Definition
------------------------------------------------------------*/
void Custom_STM_App_Notification(Custom_STM_App_Notification_evt_t
*pNotification)
{
 /* USER CODE BEGIN CUSTOM_STM_App_Notification_1 */
 /* USER CODE END CUSTOM_STM_App_Notification_1 */
 switch (pNotification->Custom_Evt_Opcode)
 {
   /* USER CODE BEGIN CUSTOM_STM_App_Notification_Custom_Evt_Opcode */
   /* USER CODE END CUSTOM_STM_App_Notification_Custom_Evt_Opcode */
   /* P2P_Server */
   case CUSTOM_STM_ICARD_NO_READ_EVT:
     /* USER CODE BEGIN CUSTOM_STM_ICARD_NO_READ_EVT */
     /* USER CODE END CUSTOM_STM_ICARD_NO_READ_EVT */
     break;
   case CUSTOM_STM_ICARD_NO_WRITE_NO_RESP_EVT:
     /* USER CODE BEGIN CUSTOM_STM_ICARD_NO_WRITE_NO_RESP_EVT */
```

```
      APP_DBG_MSG("\r\n\r** ICard Number Received \n");
//      APP_DBG_MSG("\r\n\r** Write Data: 0x%02X %02X \n",
pNotification->DataTransfered.pPayload[0],
pNotification->DataTransfered.pPayload[1]);
      if (pNotification->DataTransfered.pPayload[0] == 0x42){
            APP_DBG_MSG("\r\n\r** ICard Accepted \n");
            HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_0);
            HAL_GPIO_TogglePin (GPIOA, GPIO_PIN_12);
            HAL_Delay (1000);   /* Insert delay 2000 ms */
            APP_DBG_MSG("\r\n\r** Door Unlocked \n");
            HAL_GPIO_TogglePin (GPIOA, GPIO_PIN_12);
//          HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_0);
            HAL_Delay(3000);
            HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_0);
            HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_1);
                  HAL_GPIO_TogglePin (GPIOA, GPIO_PIN_11);
                  HAL_Delay (1000);   /* Insert delay 2000 ms */
                  APP_DBG_MSG("\r\n\r** Door Locked \n");
                  HAL_GPIO_TogglePin (GPIOA, GPIO_PIN_11);
                  HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_1);
                  unlocked_before = 1;
//                APP_DBG_MSG("\r\n\r** Door Locked \n");
      } else{
            APP_DBG_MSG("\r\n\r** ICard Rejected \n");
            HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_1);
                  HAL_Delay (2000);   /* Insert delay 1000 ms */
                  HAL_GPIO_TogglePin (GPIOB, GPIO_PIN_1);
                  APP_DBG_MSG("\r\n\r** Try Again \n");
      }
    /* USER CODE END CUSTOM_STM_ICARD_NO_WRITE_NO_RESP_EVT */
     break;
   /* HR_long */
   case CUSTOM_STM_HR_M_NOTIFY_ENABLED_EVT:
     /* USER CODE BEGIN CUSTOM_STM_HR_M_NOTIFY_ENABLED_EVT */
     /* USER CODE END CUSTOM_STM_HR_M_NOTIFY_ENABLED_EVT */
     break;
   case CUSTOM_STM_HR_M_NOTIFY_DISABLED_EVT:
     /* USER CODE BEGIN CUSTOM_STM_HR_M_NOTIFY_DISABLED_EVT */
     /* USER CODE END CUSTOM_STM_HR_M_NOTIFY_DISABLED_EVT */
     break;
   default:
     /* USER CODE BEGIN CUSTOM_STM_App_Notification_default */
     /* USER CODE END CUSTOM_STM_App_Notification_default */
     break;
 }
 /* USER CODE BEGIN CUSTOM_STM_App_Notification_2 */
 /* USER CODE END CUSTOM_STM_App_Notification_2 */
 return;
```

```
}
void Custom_APP_Notification(Custom_App_ConnHandle_Not_evt_t *pNotification)
{
 /* USER CODE BEGIN CUSTOM_APP_Notification_1 */
 /* USER CODE END CUSTOM_APP_Notification_1 */
 switch (pNotification->Custom_Evt_Opcode)
 {
   /* USER CODE BEGIN CUSTOM_APP_Notification_Custom_Evt_Opcode */
   /* USER CODE END P2PS_CUSTOM_Notification_Custom_Evt_Opcode */
   case CUSTOM_CONN_HANDLE_EVT :
     /* USER CODE BEGIN CUSTOM_CONN_HANDLE_EVT */
     /* USER CODE END CUSTOM_CONN_HANDLE_EVT */
     break;
   case CUSTOM_DISCON_HANDLE_EVT :
     /* USER CODE BEGIN CUSTOM_DISCON_HANDLE_EVT */
     /* USER CODE END CUSTOM_DISCON_HANDLE_EVT */
     break;
   default:
     /* USER CODE BEGIN CUSTOM_APP_Notification_default */
     /* USER CODE END CUSTOM_APP_Notification_default */
     break;
 }
 /* USER CODE BEGIN CUSTOM_APP_Notification_2 */
 /* USER CODE END CUSTOM_APP_Notification_2 */
 return;
}
void Custom_APP_Init(void)
{
 /* USER CODE BEGIN CUSTOM_APP_Init */
 /* USER CODE END CUSTOM_APP_Init */
 return;
}
/* USER CODE BEGIN FD */
/* USER CODE END FD */
/*************************************************************
*
* LOCAL FUNCTIONS
*
*************************************************************/
/* P2P_Server */
/* HR_long */
void Custom_Hr_m_Update_Char(void) /* Property Read */
{
 uint8_t updateflag = 0;
 /* USER CODE BEGIN Hr_m_UC_1*/
 /* USER CODE END Hr_m_UC_1*/
 if (updateflag != 0)
 {
```

```
    Custom_STM_App_Update_Char(CUSTOM_STM_HR_M, (uint8_t *)UpdateCharData);
  }
  /* USER CODE BEGIN Hr_m_UC_Last*/
  /* USER CODE END Hr_m_UC_Last*/
  return;
}
void Custom_Hr_m_Send_Notification(void) /* Property Notification */
{
  uint8_t updateflag = 0;
  /* USER CODE BEGIN Hr_m_NS_1*/
  /* USER CODE END Hr_m_NS_1*/
  if (updateflag != 0)
  {
    Custom_STM_App_Update_Char(CUSTOM_STM_HR_M, (uint8_t *)NotifyCharData);
  }
  /* USER CODE BEGIN Hr_m_NS_Last*/
  /* USER CODE END Hr_m_NS_Last*/
  return;
}
/* USER CODE BEGIN FD_LOCAL_FUNCTIONS*/
/* USER CODE END FD_LOCAL_FUNCTIONS*/
```