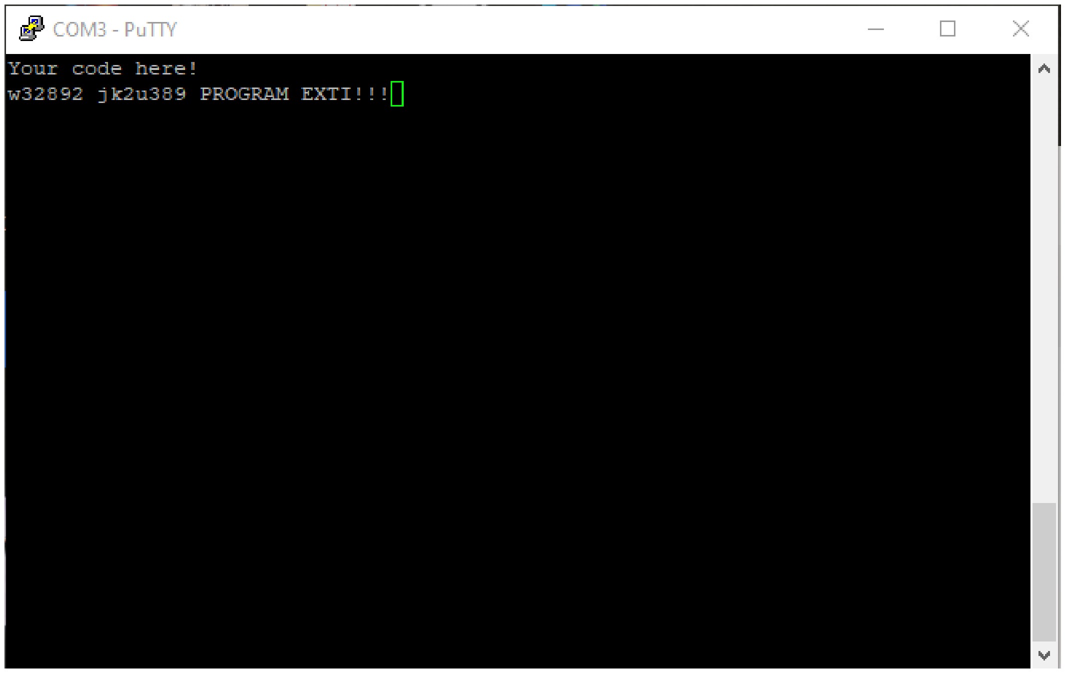
Task 1: Two-Terminal UART (Polling)

- Introduction and High-Level Description
    - The purpose of task 1 is to let the program monitor two serial ports continuously. If there is any character input from either of the onboard serial ports, it will be echoed back to both ports. And whenever “ESC” is pressed, an exit message will be shown on both terminals and the program will be halted.
    - USART6 on each DISCO board will be connected.
    - Wire connection:

Table 1: Task 1 schematic

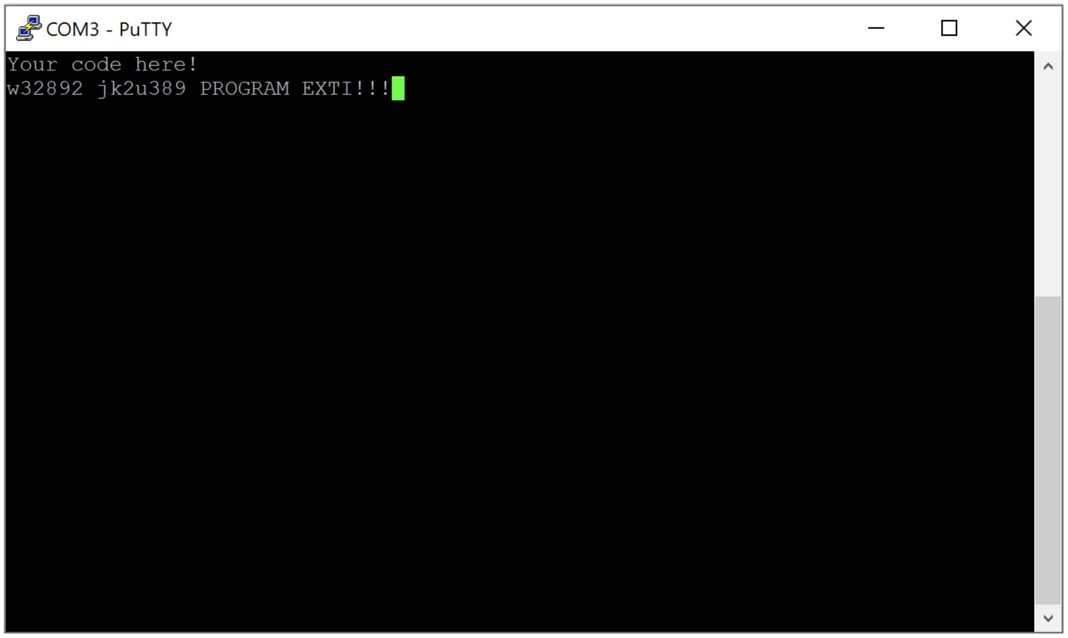
	Connected to	Connected to
TX to RX	Board 1 D1	Board 2 D2
RX to TX	Board 1 D2	Board 2 D1

    - Flowchart in appendix
- Low Level Description
    - Use `usb = HAL_UART_Receive(&USB_UART, (uint8_t*) input, 1, 10);` and `uart6 = HAL_UART_Receive(&huart6, (uint8_t*) output, 1, 20);` to let the program to wait for the corresponding event flag to be set.
    - `HAL_UART_Transmit(&huart6, (uint8_t*) input, 1, 10);` and `HAL_UART_Transmit(&USB_UART, (uint8_t*) input, 1, 10);` are used to transmit characters between two ports[1].
    - `HAL_OK` is the return value of the transmit function to indicate the success of sending or receiving data.
    - When “ESC” is pressed, `HAL_UART_Transmit(&USB_UART, (uint8_t*) "PROGRAM EXTI!!!", 16, 10);` will be used to show the stop message and halt the program[1].
    - `HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);` and `HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);` are used respectively to set up TX Config and RX Config
    - The GPIO settings used in previous labs are also used in this task to set the corresponding pins and modes.
- Results and Analysis
    - The program worked just as described in the Lab instruction. When there was a character typed by either of the keyboard, it showed on both of the terminals.
    - What’s new in this lab is how to let the pings use usb or uart6.
    - Here are the results.



```
Your code here!
w32892 jk2u389 PROGRAM EXTI!!!
```

Figure 1: terminal 1 outpu

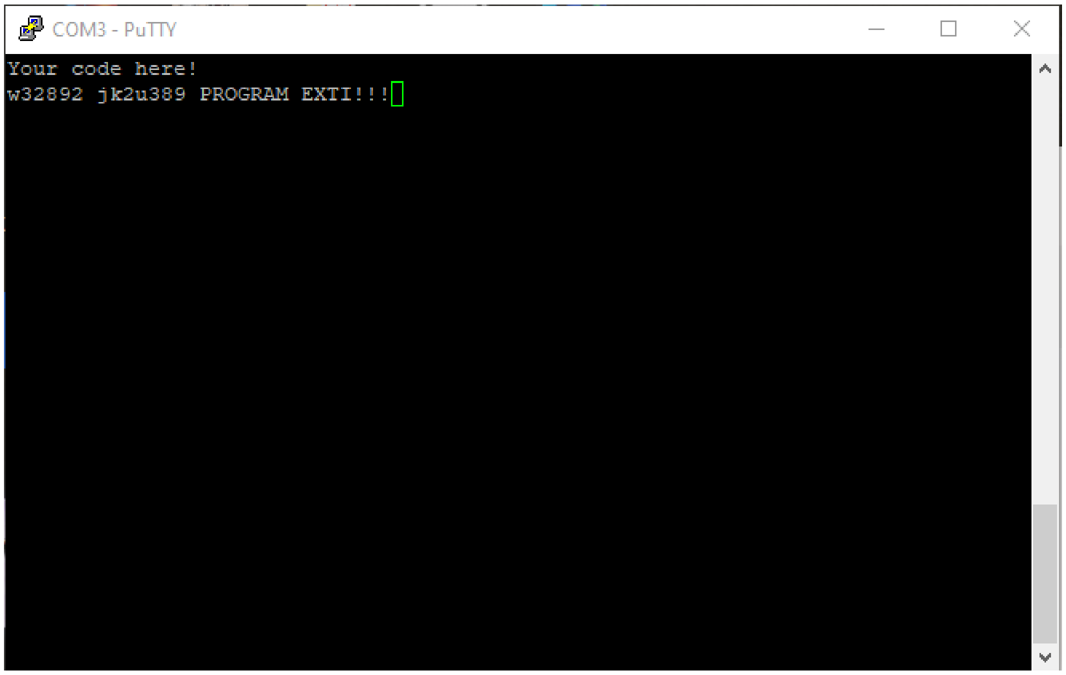


```
Your code here!
w32892 jk2u389 PROGRAM EXTI!!!
```

Figure 2: terminal 2 output

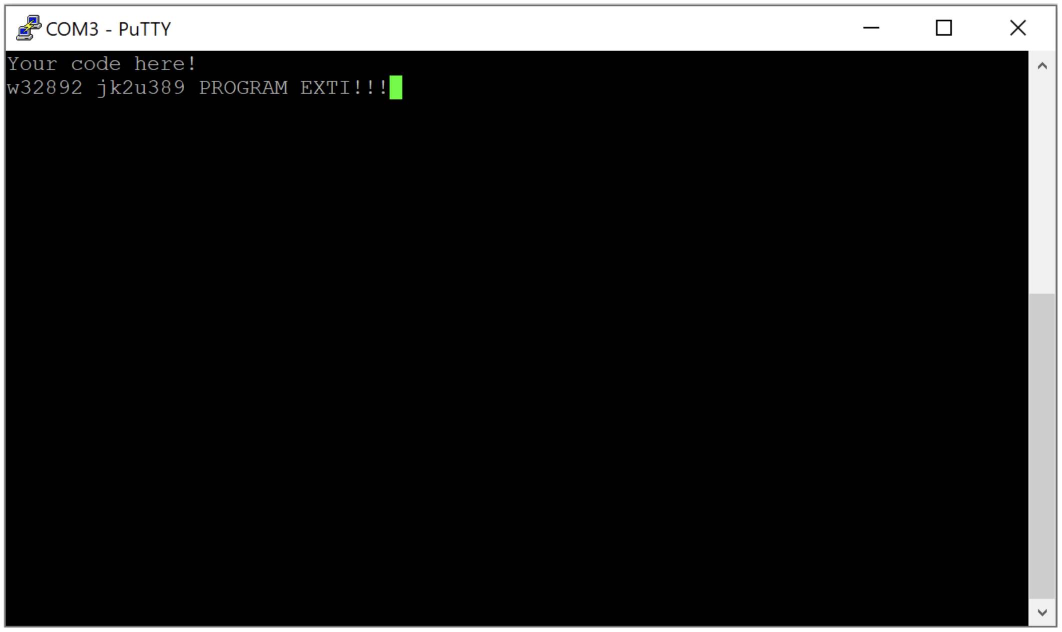
Task 2: Two-Terminal UART (Interrupt)

- Introduction and High-Level Description
  - The purpose of this task is quite like the purpose of the first task. But this time, instead of using a polling method, we are going to use a non-blocking interrupt structure.
  - The wire connection is the same as task 1
  - Flowchart in appendix
- Low-Level Description
  - The UART_Receive_IT function for USB_UART and WIRE_UART from task 1 are still used in this task.
  - Use `NVIC ->ISER [37/32] = (uint32_t) 1 << (37 % 32);` and `NVIC ->ISER [71/32] = (uint32_t) 1 << (71 % 32);` to enable the interrupts for USART1 and USART6[3].
  - Setting IRQHandler for USART1 and USART6 by using `HAL_UART_IRQHandler(&USB_UART);` and `HAL_UART_IRQHandler(&WIRE_UART);`.[1]
  - Additionally, we will have to use `HAL_UART_RxCpltCallback(UART_HandleTypeDef* uart);` function to let the program work properly. This function will be called when `HAL_UART_Receive_IT();` is called[1].
- Results and Analysis
  - The program worked just as described in the Lab instruction. When there was a character typed by either of the keyboard, it showed on both of the terminals.
  - Have to be aware that this time we have to write a call back function.
  - Have to clear the flag after triggering the interrupts.
  - Here are the results.



```
Your code here!
w32892 jk2u389 PROGRAM EXTI!!!
```

Figure 3: terminal 1 output



```
Your code here!
w32892 jk2u389 PROGRAM EXTI!!!
```

Figure 4: Terminal 2 output

Task 3: SPI Loopback Interface

- Introduction and High-Level Description
    - In this task, we will set SPI2 in 8-bit mode. SPI bus will be used in this task. More specifically, the MISO and MOSI will be connected together, which will create a loopback condition. The SPI port should be set to operate at 1MHz and the terminal should be separated into two parts. The top half will show the chapters typed in through keyboard and the bottom half will show the characters received from the SPI.
    - Wire connection

Table 2: Task 3 schematic

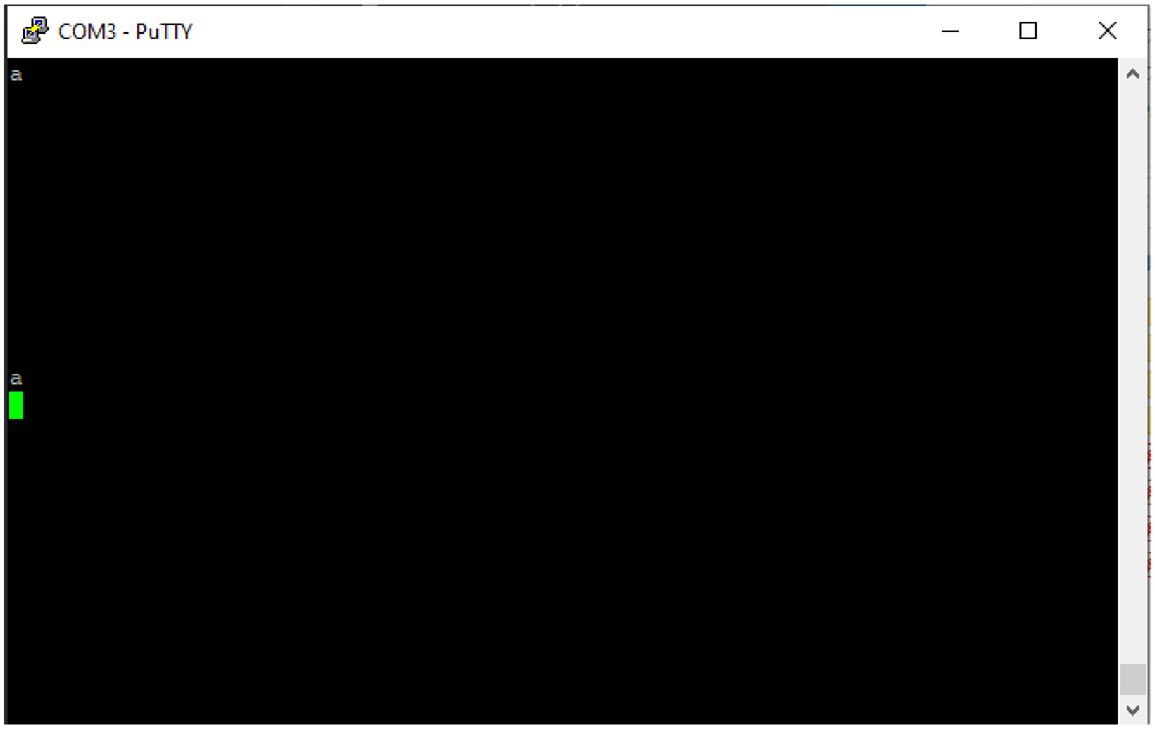
	Connected to	Connected to
MISO to MOSI	D11 DISCO board	D12 DISCO board

    -
    - Flowchart
- Low -Level Description
    - First, enable SPI2 by `__SPI2_CLK_ENABLE();` . Then, use `SPI_handle_type.Instance = SPI2;` to indicate that the rest modifications are made specifically for SPI2 [2].

```
SPI_handle_type.Init.Mode = SPI_MODE_MASTER;
SPI_handle_type.Init.TIMode = SPI_TIMODE_DISABLE;
SPI_handle_type.Init.Direction =  SPI_DIRECTION_2LINES;
SPI_handle_type.Init.DataSize = SPI_DATASIZE_8BIT;
SPI_handle_type.Init.CLKPolarity = SPI_POLARITY_HIGH;
SPI_handle_type.Init.CLKPhase = SPI_POLARITY_HIGH;
SPI_handle_type.Init.NSS = SPI_NSS_HARD_OUTPUT;
SPI_handle_type.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_256;
SPI_handle_type.Init.FirstBit = SPI_FIRSTBIT_LSB;
SPI_handle_type.Init.TIMode = SPI_TIMODE_DISABLE;
SPI_handle_type.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
```

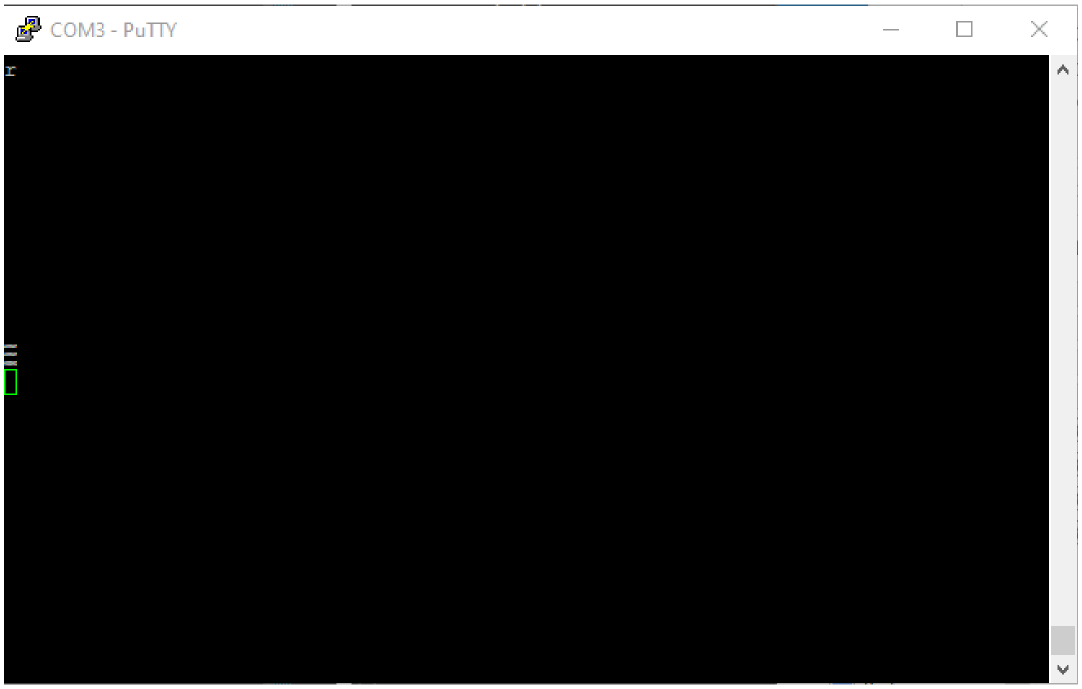
    These codes modify the master mode, Motorola mode, and etc.
    - Use `void HAL_SPI_MspInit(SPI_HandleTypeDef *hspi)` to modify the pins and modes like in Lab 1 and Lab2.
    - In this case, `HAL_SPI_TransmitReceive(&SPI_handle_type, (uint8_t*) input, (uint8_t*) output,1,10);` is used to transmit and receive characters [2].
- Results and Analysis
    - The program worked as expected.
    - SPI2 should be set as introduced on page 462, “Mastering-stm32”.

- Here are the results.


COM3 - PuTTY

```
a









a
```

Figure 5: Task 3 correct result


COM3 - PuTTY

```
r









≡
```

Figure 6: Task 3 no connection result

Task 4: Connect to STM32-Based SPI Device

- *(a) Introduction and High-Level Description*
    - In this task, students need to use SPI to communicate. A STaTS device, Nucleo-F303RE board, is provided as a peripheral in this task. The DISCO board is the controller for SPI.
    - Since the firmware is given, only the program on the DISCO board needs to be written by a student.
    - There are multiple small tasks in this task, the following is the method to fulfill each task:
        - Send terminal characters from DISCO to peripheral: send each input character to peripheral unless ESC for calling a menu.
        - Receive terminal characters from peripheral device: a function in the menu opens the receive mode, each time 1 character can be received.
        - Read the firmware version upon start-up. This function is called after initialization, receive data from the peripheral and print it on the terminal
        - Read temperature: a function in the menu, open receive temperature mode. Print the temperature data on the terminal when reading is ready.
        - Clear peripheral terminal: function in the send data to a certain register to clear the terminal screen.
        - Change Device ID: a function in the menu. Get the new ID from keyboard and send it to peripheral

- Flowchart in appendix
- Wire connection:

Table 3: Task 4 schematic

	Connected to	Connected to
SPI_MISO	D11 DISCO board	D11 peripheral
SPI_MOSI	D12 DISCO board	D12 peripheral
SPI_CLK	D13 DISCO board	D13 peripheral
SPI_CS	D10 DISCO board	A2 peripheral

- *(b) Low Level Description*
  - All the data used in this task is from the data sheet, STaTS device Datasheet.
  - The basic idea of SPI is to set the CS pin to reset, set a delay of 10us, then transmit to locate the required bit. Set a delay, transmit or receive data from the peripheral. Set a delay and set the CS pin to set status.
  - For each transmission requires a delay.

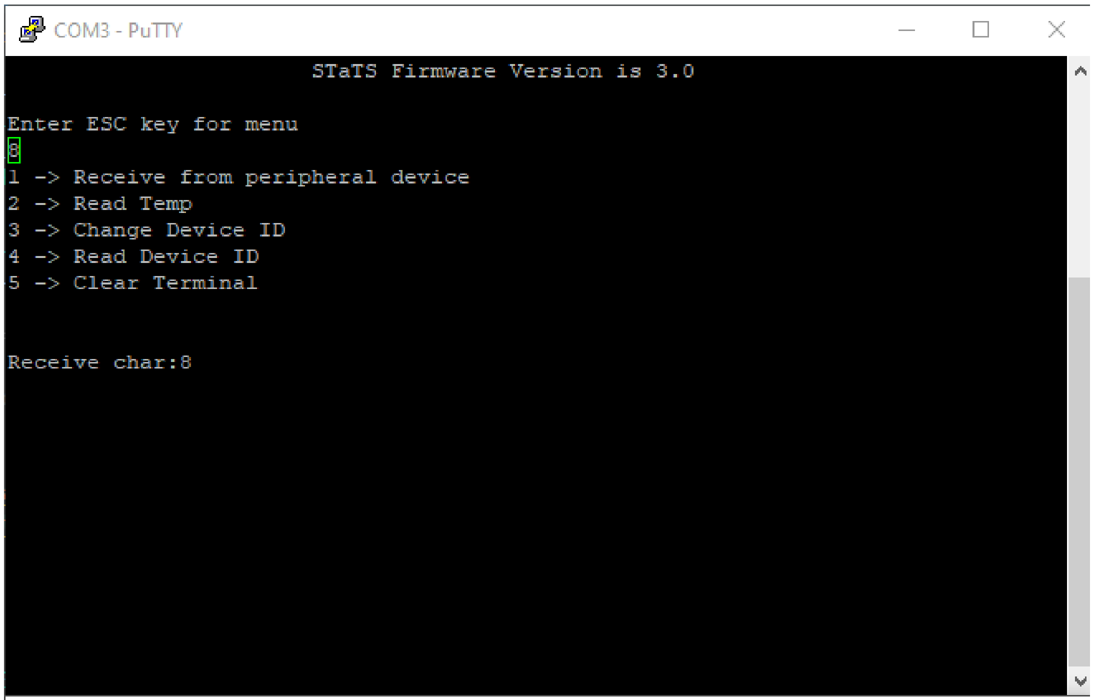
- The GPIO setting for the pin to operate SPI is important Pin D11, 12, and 13 are for SPI_SIMO, SPI_MISO, and SPI_CLK, these pins need to be set to alternate function with push-pull. Pin D10, SPI_CS, needs to be set to output push-pull mode.
- The SPI transfer setting needs to be set correctly to transform data. Polarity low, phase 2 edge, NSS soft, baud prescaler 128, data size 8 bit, first bit most significant bit [4].
- Here students use the `HAL_GPIO_WritePin` to change the status of CS pin, the part is the same as it used in LAB 1.
- `HAL_SPI_Transmit`, `HAL_SPI_Receive`, and `HAL_SPI_TransmitReceive` [2] are used for transmitting or receiving data. The first parameter is the SPI configuration, the second is the bit of register is the data to transfer, it can be a register number or the data to be sent to the peripheral
- For sending and receiving characters, register 5, CH_BUF, is used. Sending data uses receive, receiving data uses TransmitRecevie function by sending 0x00 to prevent the peripheral receiving data. Then receive the 3 bit data from it. To check if data is in the peripheral data, the keep checking status register bit 6 and 5 until one of them is not 0. Then request the data in the buffer and print it in the terminal.
- For the firmware, register 7 and 8 are used.

- For the temperature, first send data to control register bit 1 to request a new temperature read. Then keep checking status bit 3 until it is 1 when the temperature is ready. Then ready the register 3 and 4 to get the 12 bit temperature number. The Celsius degree of temperature need to be changed:

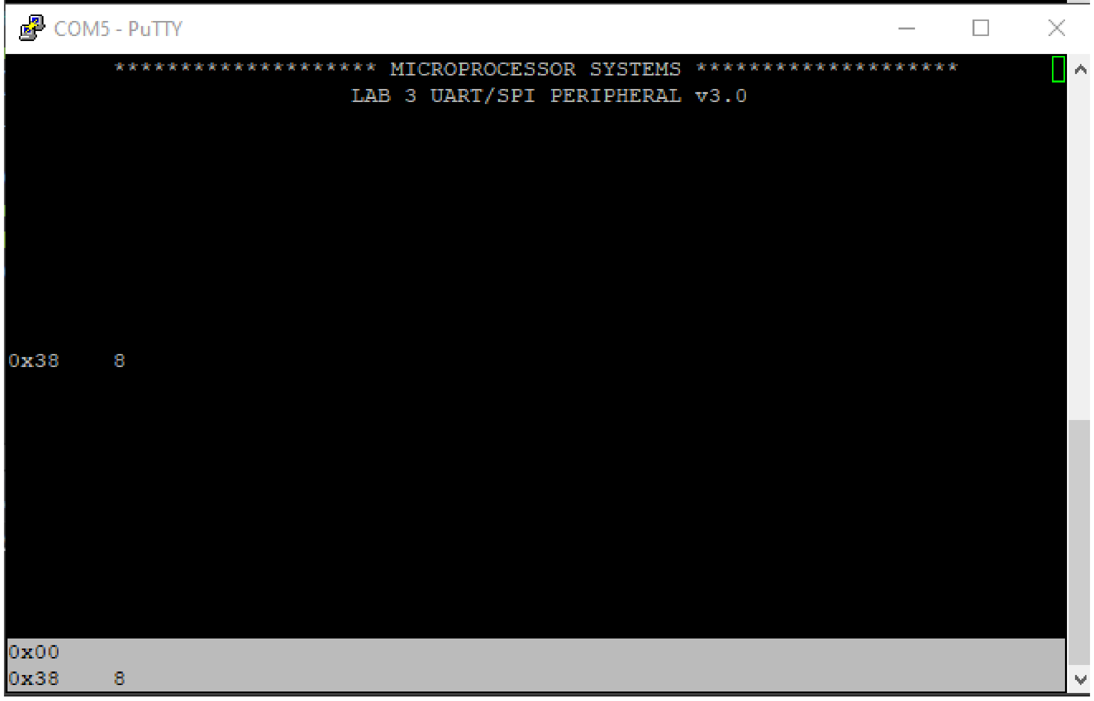
  Temperature = 357.6 – 0.187 * value

- To change the device ID of the peripheral, read register 9 DEVID to get the old ID. First set bit 7 of the control register to 1 to enable the device ID change. With this step, the controller will be able to change ID. Get the ID from the keyboard, change the input char to into by minus it to "0", otherwise the device ID will be AscII number. Send the data to the register 9 DEVID.

* *(c) Results and Analysis:*
  - The program worked as expected
  - The delay is significant in SPI. Without delays, the reading and sending may not work properly.
  - The SPI connection may be influenced by the wire connection, the SPI may not work if the wire had a bad connection. So the SPI has a chance not to work.

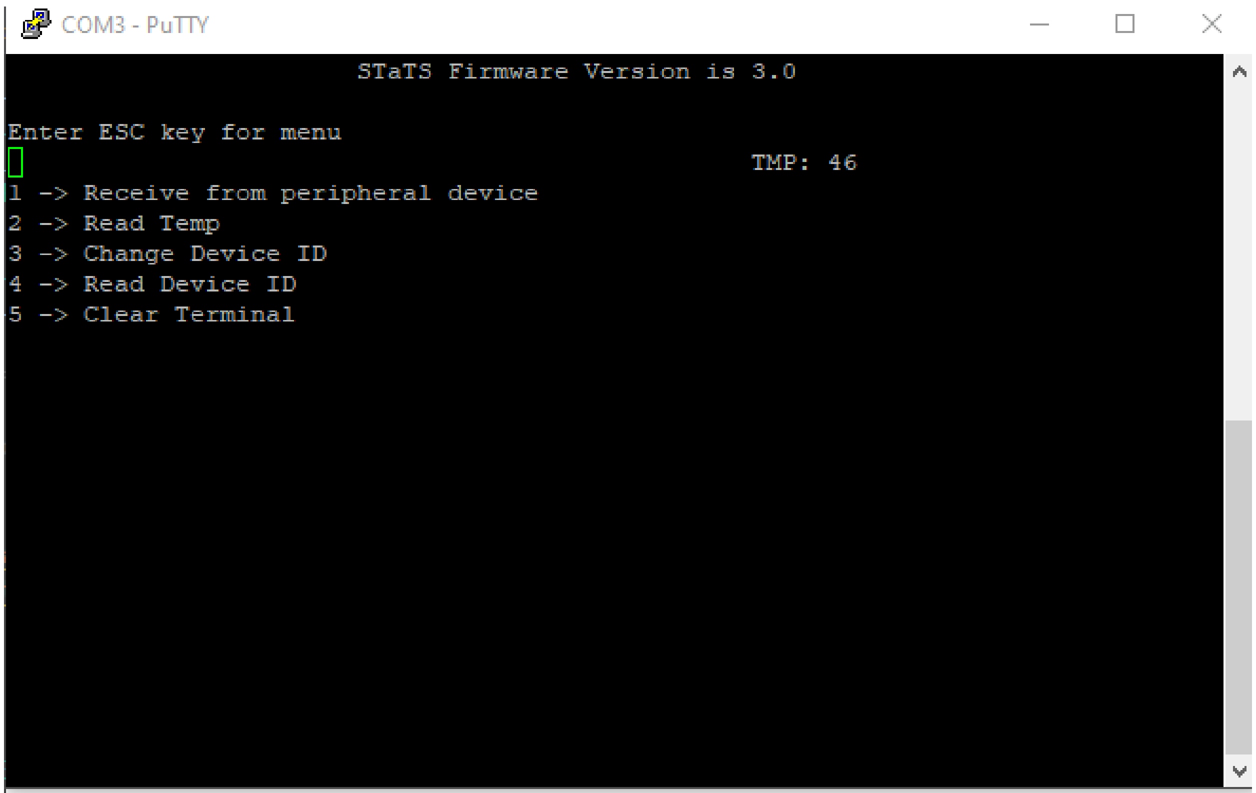


```
                    STaTS Firmware Version is 3.0

Enter ESC key for menu
8
1 -> Receive from peripheral device
2 -> Read Temp
3 -> Change Device ID
4 -> Read Device ID
5 -> Clear Terminal


Receive char:8
```

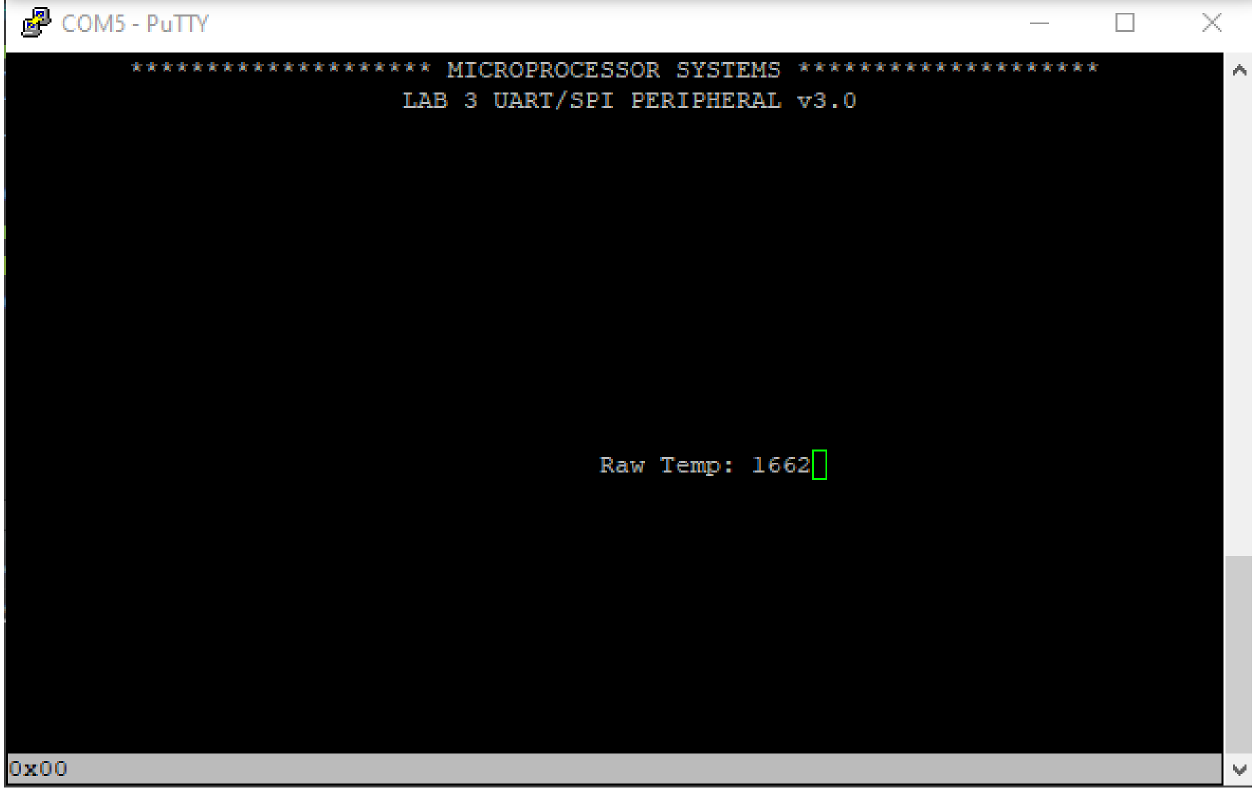


```
        ******************** MICROPROCESSOR SYSTEMS ********************
                          LAB 3 UART/SPI PERIPHERAL v3.0








0x38    8











0x00
0x38    8
```

Figure 7: Receive data from peripheral

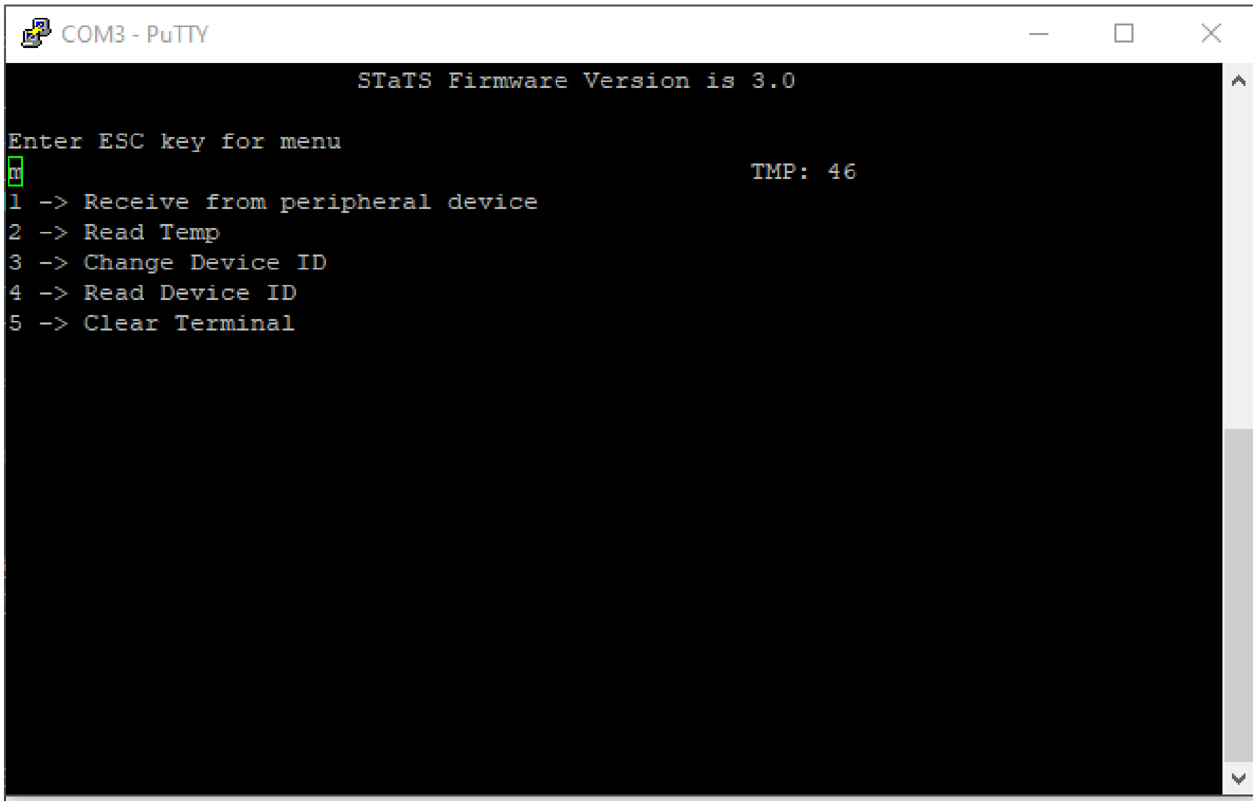


```
                      STaTS Firmware Version is 3.0

Enter ESC key for menu
                                                 TMP: 46
1 -> Receive from peripheral device
2 -> Read Temp
3 -> Change Device ID
4 -> Read Device ID
5 -> Clear Terminal
```

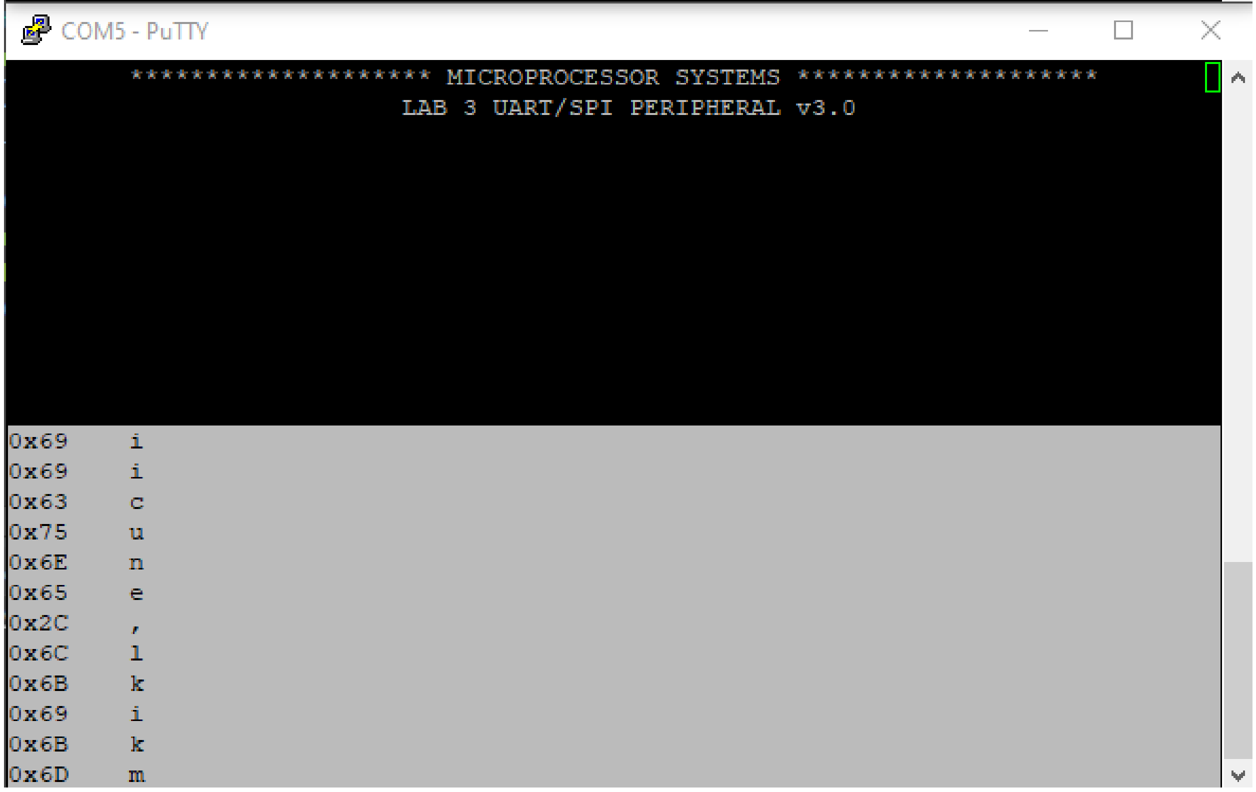


```
        ******************** MICROPROCESSOR SYSTEMS ********************
                         LAB 3 UART/SPI PERIPHERAL v3.0

                                       Raw Temp: 1662

0x00
```

Figure 8: Read temperature

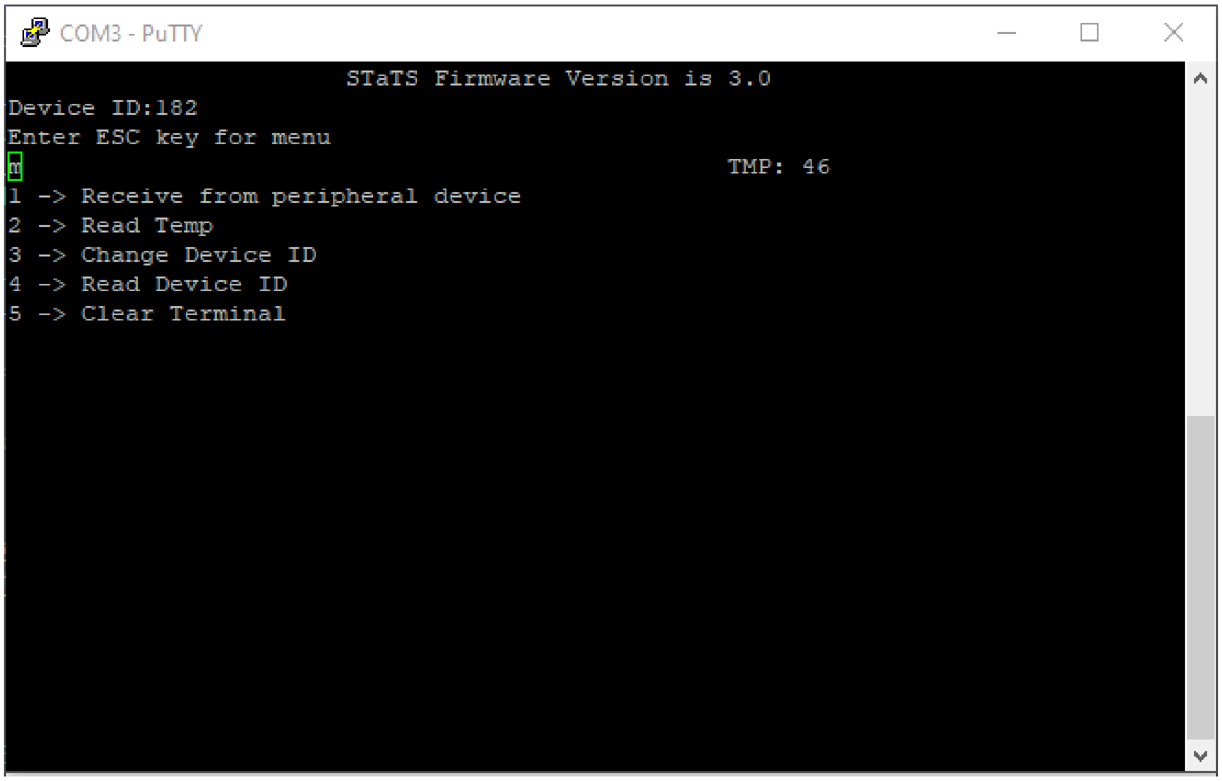


```
                       STaTS Firmware Version is 3.0

Enter ESC key for menu
m                                                TMP: 46
1 -> Receive from peripheral device
2 -> Read Temp
3 -> Change Device ID
4 -> Read Device ID
5 -> Clear Terminal
```



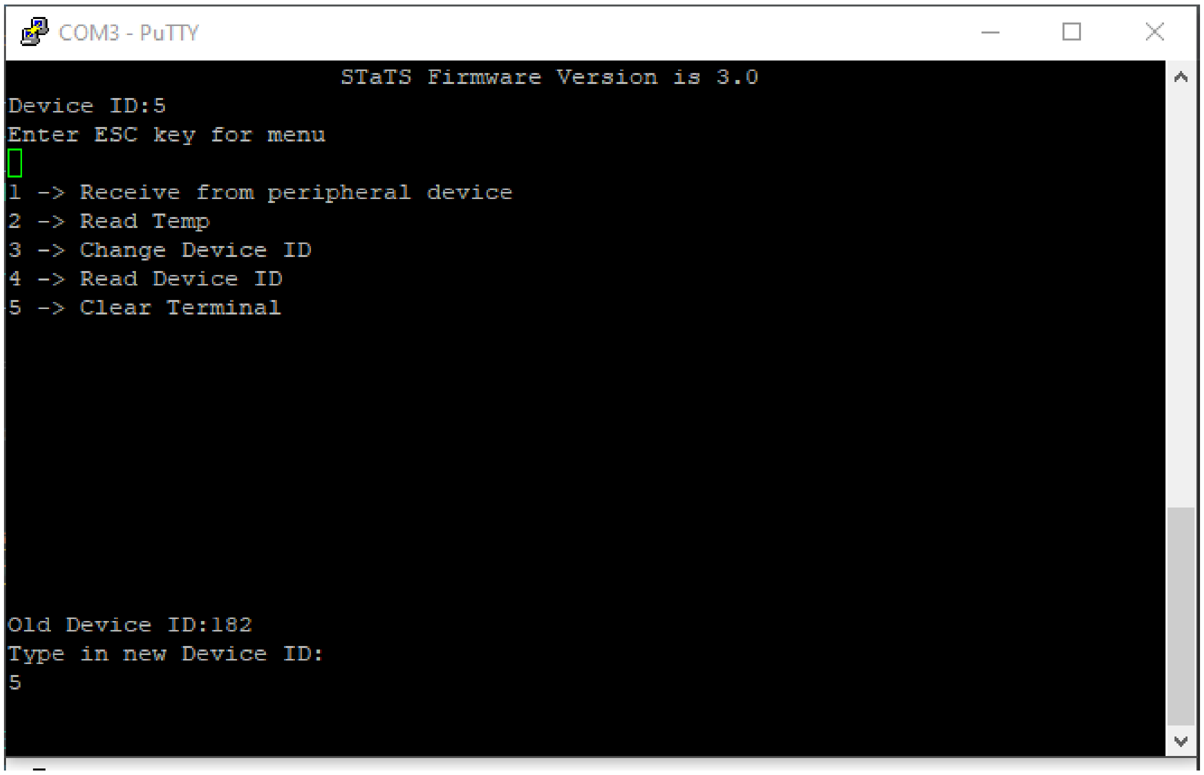
```
        ******************** MICROPROCESSOR SYSTEMS ********************
                          LAB 3 UART/SPI PERIPHERAL v3.0

0x69    i
0x69    i
0x63    c
0x75    u
0x6E    n
0x65    e
0x2C    ,
0x6C    l
0x6B    k
0x69    i
0x6B    k
0x6D    m
```

Figure 9: Send character to peripheral



```
                              STaTS Firmware Version is 3.0
Device ID:182
Enter ESC key for menu
m                                                               TMP: 46
1 -> Receive from peripheral device
2 -> Read Temp
3 -> Change Device ID
4 -> Read Device ID
5 -> Clear Terminal
```

Figure 10: Read peripheral ID address



```
                              STaTS Firmware Version is 3.0
Device ID:5
Enter ESC key for menu

1 -> Receive from peripheral device
2 -> Read Temp
3 -> Change Device ID
4 -> Read Device ID
5 -> Clear Terminal







Old Device ID:182
Type in new Device ID:
5
```

FIgure 11: Change the peripheral ID

Conclusion:

Through Task 1, we got an idea about how to transfer characters between 2 DISCO boards using Polling mode. One port is set to use USQRT1 over USB and the other port is set to use USART6. We learned how to use HAL_UART_Receive properly and how to set a desired pin to a desired mode.

For Task 2, we achieved the same result as the one in task 1 but using a different approach. This time, a non-blocking interrupt structure has been applied. We enabled the interrupts for both USART1 and USART6 so they can be triggered whenever there is an input through the keyboard. Also, in this task, we wrote a callback function. It will be called when HAL_UART_Receive_IT() is called.

In task 3, we successfully let the board receive the characters from the spi bus, and write the characters back to the terminal. We adjusted a lot of settings for the SPI handler. The spi port is configured about 1Mhz. And after we input characters from the keyboard, both the top half and the bottom half of the terminal show the characters.

Task 4 is much more complicated than the previous ones. We found that one of the issues is that we need to add delays for the SPI part. The program would not work properly if there were no delays between each transmission. Getting temperature is also quite complex. We need first to send a signal to request a temperature. Then we have to wait until the temperature is ready to be read. Finally, we have to convert the number we get to degree celsius.

From this lab, we learned several ways and modes about how to let two boards communicate with each other. These can be used in various ways. For instance the home accessories we used to build a smart-home, like light bulbs, sensors, and central hubs may contain part of what we learned. Checking devices' firmware versions and changing devices' IDs are particularly useful in these fields.

Citation

[1] *RM1905 User manual,* STMicroelectronics, NV, USA, 2017, pp. 1002-1018.

[2] *RM1905 User manual,* STMicroelectronics, NV, USA, 2017, pp. 900-916.

[3] *RM0410 Reference manual,* STMicroelectronics, NV, USA, 2018, pp. 313-318.

[4] *Mastering STM32,* Carmine Noviello, USA, 2018, pp. 461.

Appendix

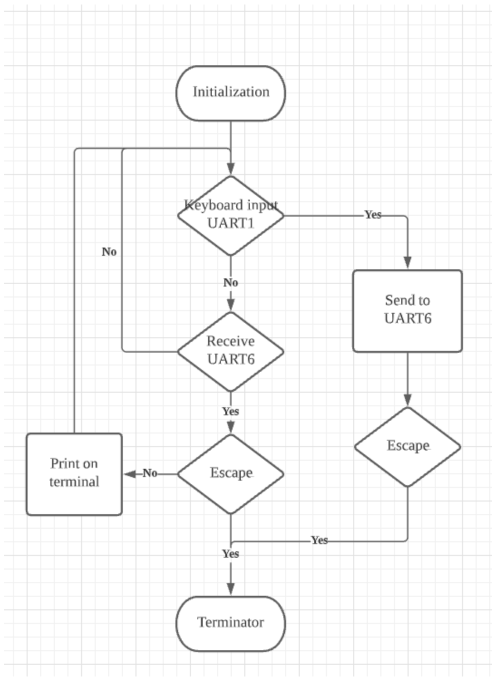


Figure 12: Task 1 Flowchart

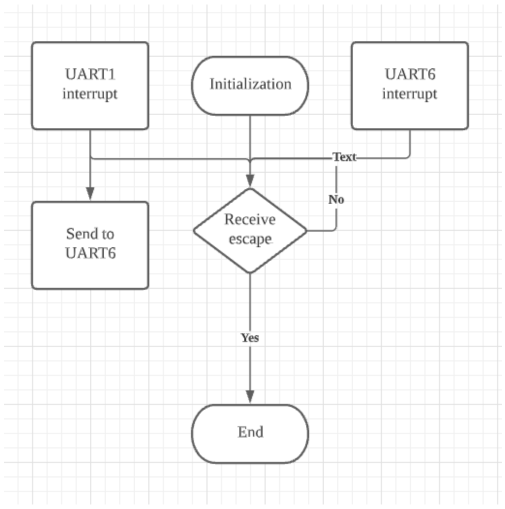


Figure 13: Task 1 Flowchart

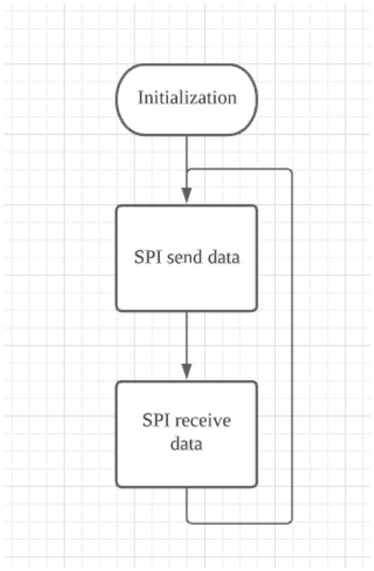


Figure 14: Task 1 Flowchart

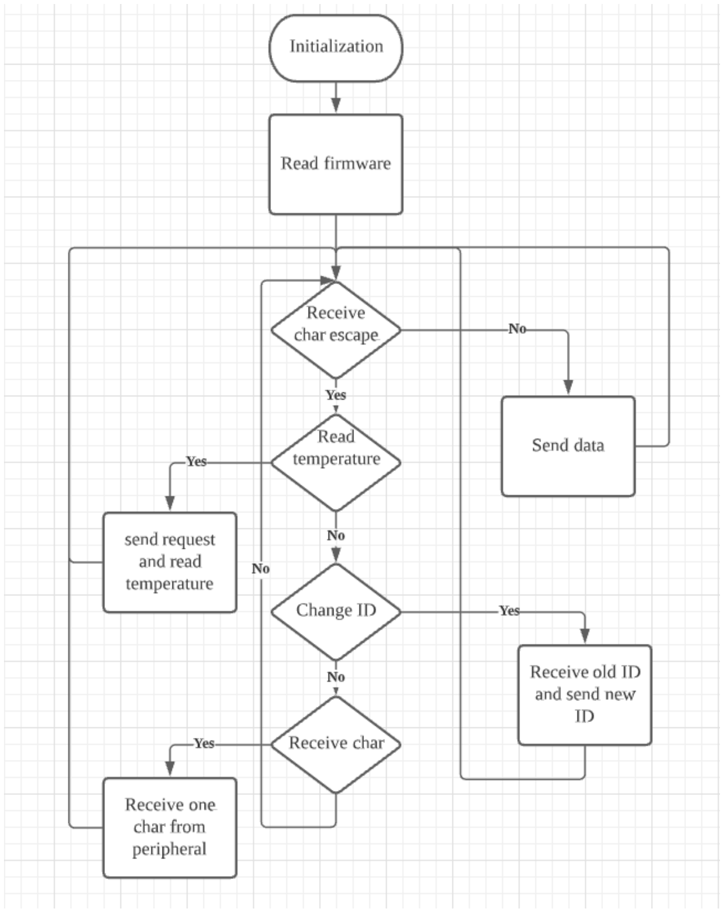


Figure 15: Task 1 Flowchart