ECE 153B – Winter 2023
Sensor and Peripheral Interface Design

Lab 4 – UART, I^{2}C, and SPI

Deadline: Feb 21, 2023, 7:00 PM

Objectives

1. Understand UART
   - Communicate with a computer to print debug messages
   - Interface with the HC-05 bluetooth module
2. Understand the I^{2}C communication protocol
   - Interface with the TC-74 temperature sensor
3. Understand SPI
   - Simulate SPI communication using loopback

Grading

Part	Weight
Part A	35 %
Part B	35 %
Part C	30 %

You must submit your code as zipped file and answers to the questions as PDF to the submission link on Gradescope by the specified deadline. In the week following the submission on Gradescope, you will demo your lab to the TA.

Please note that we take the Honor Code very seriously, do not copy the code from others.

Necessary Supplies

- STM32L4 Nucleo Board
- Type A Male to Mini B USB Cable
- HC-05 Bluetooth Module
  - USB Bluetooth Receiver (compatible with Windows only) for those of you who do not have a laptop/computer with Bluetooth functionality. We tested various devices and found that:
    - iPhones will not connect
    - Androids and Windows laptops (with Bluetooth capabilities) will connect
- TC-74 Temperature Sensor
- Breadboard & Jumper Wires
- 1 kΩ Resistor (x2)

1 Lab Overview

A. Use UART to control LEDs on the STM32L4 Nucleo board by sending commands from a terminal on your PC (using Termite). Use UART to interface with a Bluetooth module to control LEDs over a Bluetooth connection.

B. Use I^{2}C to receive temperature measurements from a temperature sensing module and display the information.

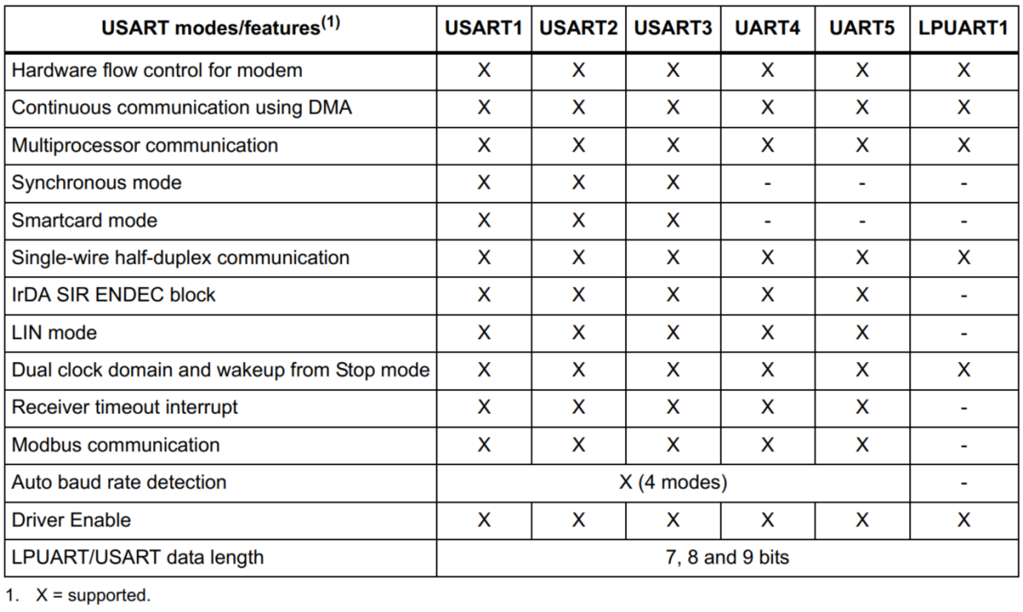
C. Use SPI loopback to simulate SPI communication.

2 Part A – UART

In this part of the lab, you will learn how to set up and use UART (Universal Asynchronous Receiver/Transmitter) to exchange data between the microprocessor and a serial port.

2.1 Introduction

The STM32L4 Nucleo board has 6 embedded UARTs. Table 1 describes the features of each UART that is available on the board. For this lab, we will be using USART2 (for communication through the USB cable) and USART1 (to be used with Bluetooth).



USART modes/features[1]	USART1	USART2	USART3	UART4	UART5	LPUART1
Hardware flow control for modem	X	X	X	X	X	X
Continuous communication using DMA	X	X	X	X	X	X
Multiprocessor communication	X	X	X	X	X	X
Synchronous mode	X	X	X	-	-	-
Smartcard mode	X	X	X	-	-	-
Single-wire half-duplex communication	X	X	X	X	X	X
IrDA SIR ENDEC block	X	X	X	X	X	-
LIN mode	X	X	X	X	X	-
Dual clock domain and wakeup from Stop mode	X	X	X	X	X	X
Receiver timeout interrupt	X	X	X	X	X	-
Modbus communication	X	X	X	X	X	-
Auto baud rate detection	X (4 modes)					-
Driver Enable	X	X	X	X	X	X
LPUART/USART data length	7, 8 and 9 bits					

1. X = supported.

Table 1: STM32L4x6 USART/UART/LPUART Features

Bidirectional communication with UART requires a minimum of two pins: **RX** (for receiving data) and **TX** (for transmitting data). On the STM32L4 Nucleo board, the TX and RX pins for USART1 are available through the alternative functions of **PB6** and **PB7**, respectively. The TX and RX pins for USART2 are available through the alternative functions of **PA2** and **PA3**, respectively.

2.2 Basic Communication with the Terminal

First, you will set up UART communication between the STM32L4 Nucleo board and a terminal on your computer. The goal is to control the LED by sending commands from the terminal. In addition, the board will send back debug messages indicating the LED status.

Use the following steps to help you set up UART and the terminal on your computer.

1. Set up the clock for USART2. You will be modifying the `UART2_Init()` function in `UART.c`. You will be modifying the RCC registers. (Refer to Section 8.4 in the Reference Manual.)
   (a) Enable the USART2 clock in the peripheral clock register.
   (b) Select the system clock as the USART2 clock source in the peripheral independent clock configuration register.
2. Configure **PA2** and **PA3** to operate as UART transmitters and receivers. You will be modifying the `UART2_GPIO_Init()` function in `UART.c`. (Refer to Section 9.4 in the Reference Manual.)
   (a) Both GPIO pins should operate at very high speed.
   (b) Both GPIO pins should have a push-pull output type.
   (c) Configure both GPIO pins to use pull-up resistors for I/O.
3. Configure the settings of USART. You will be modifying the `USART_Init()` function in `UART.c`. You will be modifying the USART registers. Note that the argument is `USARt_TypeDef* USARTx`, which allows us to define the same configuration that should be applied to the specific USART that we want. Your lines of code should start with `USARTx->[register]`. We will use both USART2 and USART1 in this portion of the lab, so writing this general code will allow us to configure both in the same way without writing the same code twice – we can simply call `USART_Init(USART2)` or `USART_Init(USART1)`.

   Note that USART must be disabled to modify the settings – ensure that USART is disabled before you start to modify the registers. (Refer to Section 36.8 in the Reference Manual.)
   (a) In the control registers, set word length to 8 bits, oversampling mode to oversample by 16, and number of stop bits to 1.
   (b) Set the baud rate to 9600. To generate the baud rate, you will have to write a value into `USARTx_BRR`.

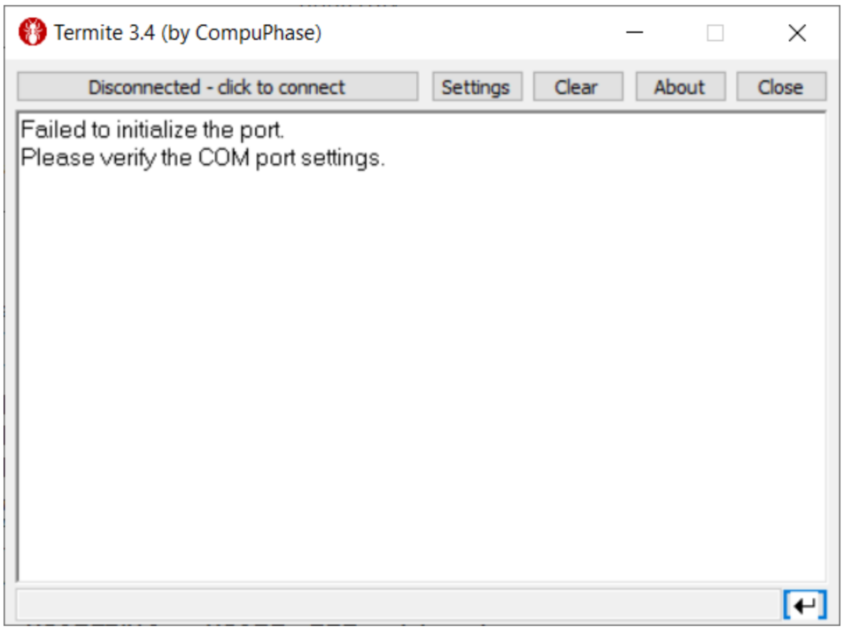
   Note the word "generate". You should not be writing "9600" into this register. Instead, you should write the value that will generate this desired baud rate. The baud rate is computed as follows

$$\text{TX/RX Baud Rate} = \frac{f_{CLK}}{USARTDIV}$$

   in the case of oversampling by 16. f_{CLK} is the frequency of the system clock and the value $USARTDIV$ is a constant that determines what the baud rate will be. Then, $USARTDIV$ is the value you have to solve for and write into `USARTx_BRR`.
   (c) In the control registers, enable both the transmitter and receiver.
   (d) Now that everything has been set up, enable USART in the control registers.

4. We will be using a program called Termite to communicate with the STM32L4 Nucleo board. Download the portable version of Termite from this link – this will allow us to run Termite without admin privileges.

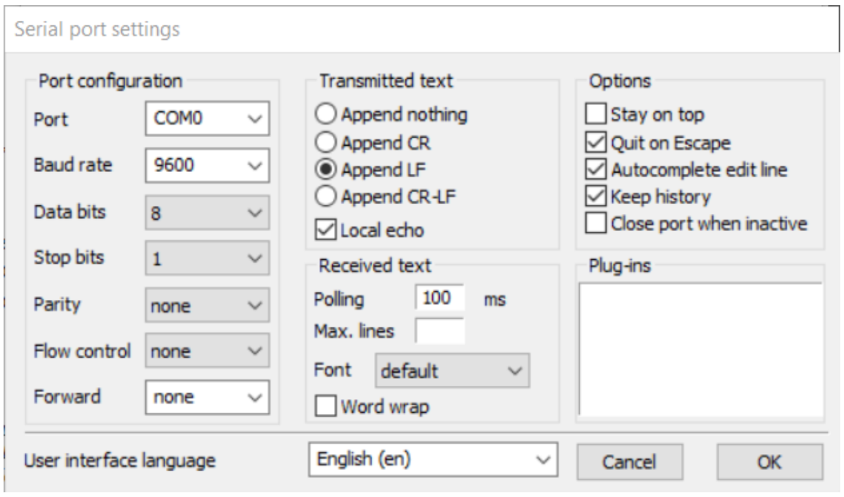
   Run `Termite.exe` and you will see a window as shown in Figure 1.



```
Failed to initialize the port.
Please verify the COM port settings.
```

Figure 1

   Click on the **Settings** button and ensure that the **Baud rate**, **Data bits**, and **Stop bits** fields match with the communication parameters that we set in the USART2 registers. (See Figure 2).



Port: COM0
Baud rate: 9600
Data bits: 8
Stop bits: 1
Parity: none
Flow control: none
Forward: none

Transmitted text: Append nothing [ ], Append CR [ ], [✓ Append LF], Append CR-LF [ ]; Local echo [✓]

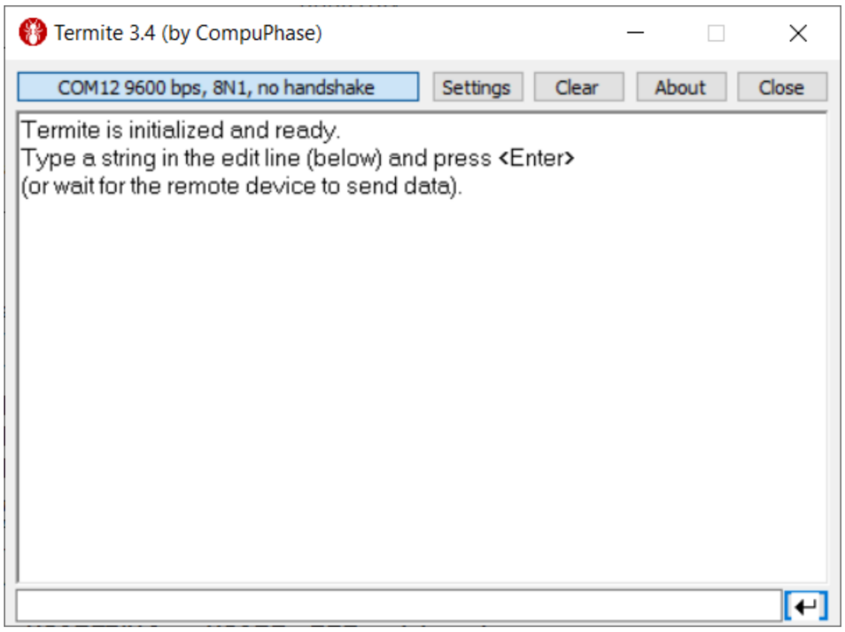
Received text: Polling 100 ms; Max. lines; Font default; Word wrap [ ]

Options: Stay on top [ ]; Quit on Escape [✓]; Autocomplete edit line [✓]; Keep history [✓]; Close port when inactive [ ]

User interface language: English (en)

Figure 2

   Exit out of the **Settings** menu and click on the **Disconnected – click to connect** button. When a successful connection is made, you will see the window with a message saying that "Termite is initialized and ready". Note that your COM port may be different. Usually, connecting your STM32L4 Nucleo board before pressing the **Connect** button will automatically find the necessary COM port. (See Figure 3).


Termite 3.4 (by CompuPhase)
COM12 9600 bps, 8N1, no handshake
Settings
Clear
About
Close

```
Termite is initialized and ready.
Type a string in the edit line (below) and press <Enter>
(or wait for the remote device to send data).
```

Figure 3

At this point, everything should be set up: UART is configured with the desired communication parameters and Termite is set up as a terminal that will communicate with the STM32L4 Nucleo board via the UART transmitters/receivers. In the file `UART_printf.c` you will see that we have retargeted the functions `printf()` and `scanf()`. This allows you to:

- Use `printf()` to transmit data to Termite. The data that you send will be printed on the Termite window.
- Use `scanf()` to receive data from Termite. Data can be sent from Termite by typing a message into the terminal and pressing enter.

Using these two functions, your task is to implement the following behavior in the main() function. You only need to handle sending one character messages from Termite.

- Send data to the terminal that prompts the user to enter a command. The valid commands are “Y”, “y”, “N”, and “n”.
- Receive the response from the terminal.
  - If the response is “Y” or “y”, turn the green LED on. In addition, send a message (that may be used for debugging purposes) to the terminal saying that the green LED has been turned on.
  - If the response is “N” or “n”, turn the green LED off. In addition, send a message to the terminal saying that the green LED has been turned off.
  - If an unrecognized command is received, send a message to the terminal prompting the user to try again with a valid command.

2.3 Interfacing with the Bluetooth Module

In this part of the lab, you will set up the HC-05 Bluetooth module and set up UART for communicating with the module. Then, you will connect to the HC-05 Bluetooth module with your phone/laptop to control the LEDs on the STM32L4 Nucleo board as you did in the previous part. Refer to the *HC-05 Bluetooth Module Datasheet* for details about the module. Figure 4 shows the connection diagram for the HC-05 Bluetooth module. (Note that the `TX` pin of one device should be connected to the `RX` pin of the other device.)

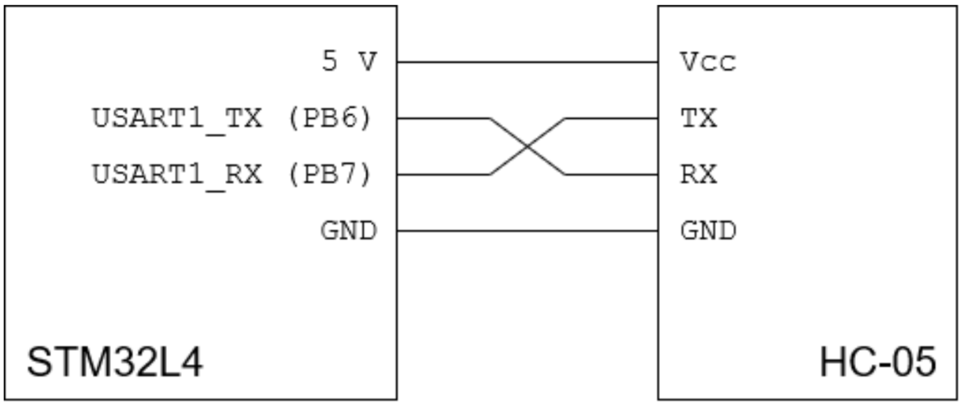


Figure 4: HC-05 Connection Diagram

For this part of the lab, you will be using **PB6** (`USART1_TX`) and **PB7** (`USART1_RX`) for UART communication because we need to access pins that we can connect wires to. Use the following steps to help you set up USART1 and communicate over Bluetooth.

Refer to the steps in the previous part to set up USART1. That is,

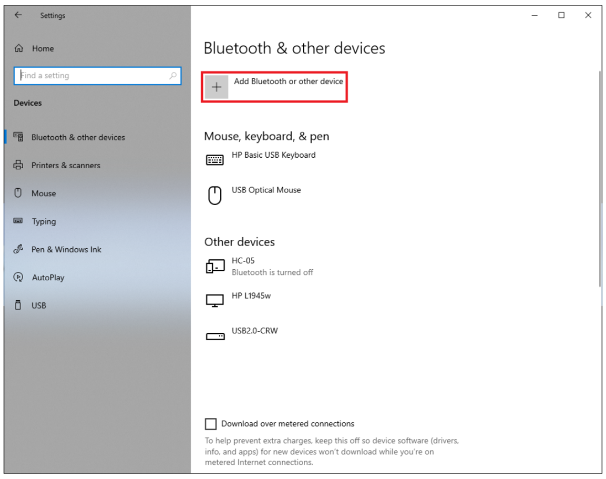
1. Modify `UART1_Init()` to set up the USART1 clock.
2. Modify `UART1_GPIO_Init()` to configure **PB6** and **PB7** to operate as UART transmitters and receivers.
3. To send/receive messages using USART1, be sure to go into `UART_printf.c`, comment out the lines in `fputc()` and `fgetc()` corresponding to USART2, and uncomment the lines in `fputc()` and `fgetc()` corresponding to USART1.

Once you have your code running, connect to it using your phone/laptop. The default pairing code is `1234` (which can be changed if you desire by sending the module `AT` commands – refer to the datasheet for more information). The HC-05 also has a red status LED that can help you determine what state it is in. If the LED is blinking rapidly, it is on and is ready to be paired. If the LED is blinking twice approximately every 5 seconds, it is paired to a device.

2.4 Establishing Bluetooth Connection through USB Bluetooth Receiver

Note: This section is for those of you who have a laptop without Bluetooth functionality.

First, insert the USB Bluetooth receiver into the USB port (it should be recognized as "dongle"). The receiver should be plug-and-play, so there is no additional setup that has to be done on the lab computers. Run your code so that your HC-05 is ready to be paired with another device. In the lab computers, go into **Settings** → **Devices** → **Bluetooth & other devices** and click on the **Add Bluetooth or other device** button (see Figure 5).


Settings

Home

Find a setting

Devices

Bluetooth & other devices

Printers & scanners

Mouse

Typing

Pen & Windows Ink

AutoPlay

USB

Bluetooth & other devices

Add Bluetooth or other device

Mouse, keyboard, & pen

HP Basic USB Keyboard

USB Optical Mouse

Other devices

HC-05
Bluetooth is turned off

HP L1945w

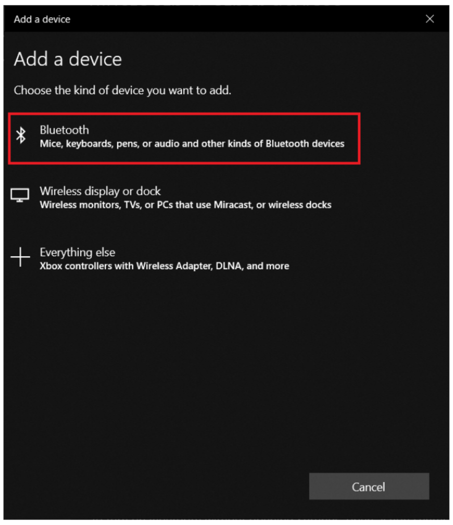
USB2.0-CRW

Download over metered connections

To help prevent extra charges, keep this off so device software (drivers, info, and apps) for new devices won't download while you're on metered Internet connections.

Figure 5

Click on the button for adding Bluetooth devices (see Figure 6) and connect to the HC-05.


Add a device

Add a device

Choose the kind of device you want to add.

Bluetooth
Mice, keyboards, pens, or audio and other kinds of Bluetooth devices

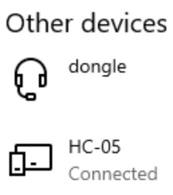
Wireless display or dock
Wireless monitors, TVs, or PCs that use Miracast, or wireless docks

Everything else
Xbox controllers with Wireless Adapter, DLNA, and more

Cancel

Figure 6

After pairing, you should be able to use Termite as usual for communicating with the STM32L4 Nucleo board using Bluetooth and UART. You might have to try different COM ports until a connection is established between the lab computer and the HC-05. To know whether you have a connection established, you can check **Settings** → **Devices** → **Bluetooth & other devices** and you will see "Connected" next to HC-05 instead of "Paired" (see Figure 7).


Other devices

dongle

HC-05
Connected

Figure 7

2.5 Establishing Bluetooth Connection with Android Phones

Install an app such as *Bluetooth Terminal HC-05*. This app provides an easy interface for finding nearby Bluetooth devices, connecting with it, and sending messages to the connected Bluetooth device.

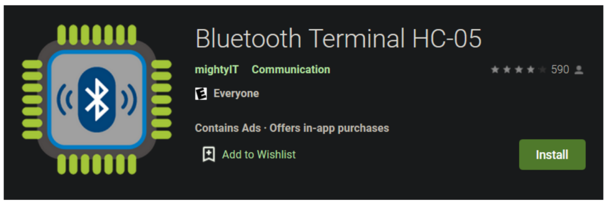


Figure 8

If you don't have an Andriod phone, please come to office hour and ask TAs for the test phone.

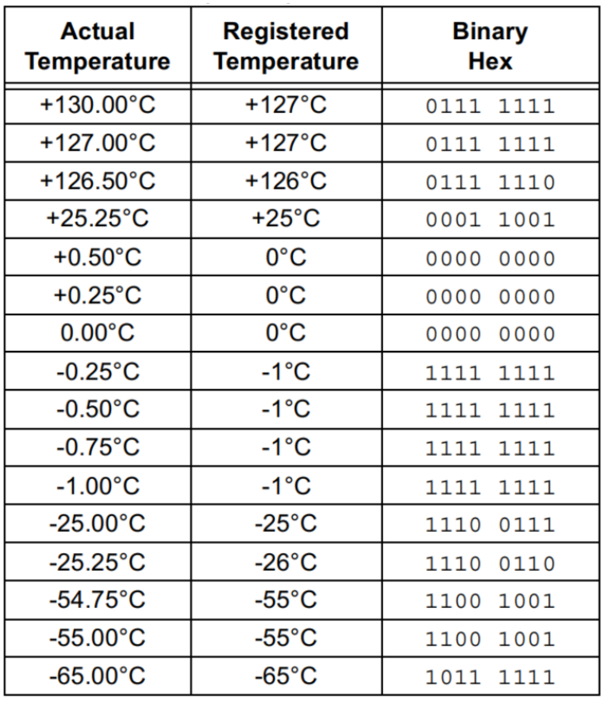
2.6 Testing with Bluetooth Terminal

You need to set the UART delay sufficiently large and use debug mode to read and write data losslessly. You may notice that if you run the program outside of debug mode, the program can only recognize the first character you enter.

3 Part B – I^2C

In this part of the lab, you will be interfacing with the TC-74 temperature sensor to get and display the temperature data on Termite. You are to interface with the temperature sensor using the I^2C serial protocol. I^2C enables communication between devices using two wires: the **serial data line** (SDA) and the **serial clock line** (SCL).

The TC-74 temperature sensor converts the measured temperature to 8 bits of digital data which can be transmitted out via the I^2C interface. As shown in Table 2, the representable temperature ranges from −65 °C to 127 °C.



Actual Temperature	Registered Temperature	Binary Hex
+130.00°C	+127°C	0111 1111
+127.00°C	+127°C	0111 1111
+126.50°C	+126°C	0111 1110
+25.25°C	+25°C	0001 1001
+0.50°C	0°C	0000 0000
+0.25°C	0°C	0000 0000
0.00°C	0°C	0000 0000
-0.25°C	-1°C	1111 1111
-0.50°C	-1°C	1111 1111
-0.75°C	-1°C	1111 1111
-1.00°C	-1°C	1111 1111
-25.00°C	-25°C	1110 0111
-25.25°C	-26°C	1110 0110
-54.75°C	-55°C	1100 1001
-55.00°C	-55°C	1100 1001
-65.00°C	-65°C	1011 1111

Table 2: Temperature to Digital Value Conversion

The temperature resolution is 1 °C and the conversion rate is 8 samples per second. Multiple sensors can be connected to the same I^2C bus and can communicate with each other as long as they all have unique addresses. For this lab, you will only have to interface with one temperature sensor. The temperature sensors can have one of eight different addresses from `1001000` to `1001111`. Table 3 shows the address of the temperature sensor that corresponds to the part number TC74Ax.

Part Number	Default Address
A0	1001000
A1	1001001
A2	1001010
A3	1001011
A4	1001100
A5	1001101
A6	1001110
A7	1001111

Table 3: Address Corresponding to Part Number

Figure 9 shows the connection diagram for the temperature sensor.

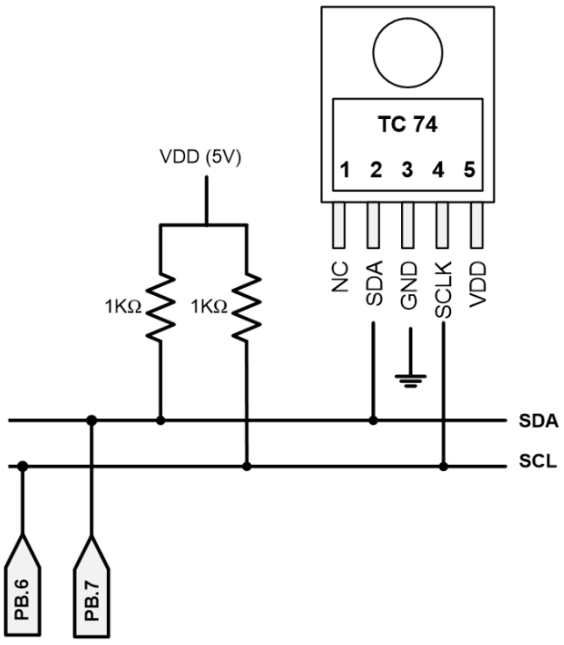


Figure 9: Temperature Sensor Connection Diagram

You will use **PB6** and **PB7** for `I2C1_SCL` and `I2C1_SDA`, respectively. Configure **PB6** and **PB7** with the following settings. Write your code in the `I2C_GPIO_Init()` function in `I2C.c`.

Pin	**Mode**	**Output Type**	**Output Speed**	**Pull-Up/Down**
PB6	Alt. Function - `I2C1_SCL`	Open-Drain	Very High	Pull-Up
PB7	Alt. Function - `I2C1_SDA`	Open-Drain	Very High	Pull-Up

Use the following steps to configure I^2C settings. Write your code in the `I2C_Initialization()` function in `I2C.c`.

1. Set up the clock for I^2C in the RCC registers. Refer to Section 8.4 of the *STM32L4x6 Reference Manual* for detailed information about the RCC registers and their bits.
   (a) Enable the clock for `I2C1` in the peripheral clock enable register.
   (b) Set the system clock as the clock source for `I2C1` in the peripherals independent clock configuration register.
   (c) Reset `I2C1` by setting bits in the peripheral reset register. After doing so, clear the bits so that `I2C1` does not remain in a reset state.

2. Configure the I^2C registers for communication with the temperature sensor. Similar to when we configured USART settings, ensure that I^2C is disabled before modifying the registers. Refer to Section 35.7 of the *STM32L4x6 Reference Manual* for more information about the I^2C registers and their bits.
   (a) Enable the analog noise filter, disable the digital noise filter, enable error interrupts, and enable clock stretching. Set the master to operate in 7-bit addressing mode. Enable automatic end mode and `NACK` generation. (These settings are all in the control registers.)

(b) Set the values in the timing register. This guarantees correct data hold and setup times that are used in master/peripheral modes. The timing register stores several values:

- `PRESC` – This value is the timing prescaler. It is used to generate the clock period that will be used for the data setup, data hold, and SCL high/low counters.

$$f_{\text{PRESC}} = \frac{f_{\text{I2CCLK}}}{\texttt{PRESC} + 1} \qquad \iff \qquad t_{\text{PRESC}} = (\texttt{PRESC} + 1) \times t_{\text{I2CCLK}}$$

- `SCLDEL` – This value determines the minimum data setup time, which is a delay between an SDA edge and the next SCL rising edge in transmission mode. The generated delay time is

$$t_{\text{SCLDEL}} = (\texttt{SCLDEL} + 1) \times t_{\text{PRESC}}$$

- `SDADEL` – This value determines the minimum data hold time, which is a delay between the SCL falling edge and the next SDA edge in transmission mode. The generated delay time is

$$t_{\text{SDADEL}} = (\texttt{SDADEL} + 1) \times t_{\text{PRESC}}$$

- `SCLH` – This value determines the minimum period for the high phase of the clock signal. The generated high period is

$$t_{\text{SCLH}} = (\texttt{SCLH} + 1) \times t_{\text{PRESC}}$$

- `SCLL` – This value determines the minimum period for the low phase of the clock signal. The generated low period is

$$t_{\text{SCLL}} = (\texttt{SCLL} + 1) \times t_{\text{PRESC}}$$

Figure 10 shows an illustration of the data hold and setup times.

These values must meet the requirements of the TC-74 temperature sensor. Keep in mind that the system clock (which we configured to be the source of the I^{2}C clock) has a frequency of 80 MHz. First, write down the prescaler value that you will use. Then, look in *Digital Thermal Sensor - TC74 - Data Sheet* to find the specifications for these timings. Using this information, you can calculate what values need to be written to the timing register such that the timing guarantees meet the temperature sensor's specifications. Macros for the values' position in the timing register have been defined for you.

`PRESC` = ____________________

Min. Low Clock Period = ____________________

Min. High Clock Period = ____________________

Min. Data Setup Time = ____________________

Min. Data Hold Time = ____________________

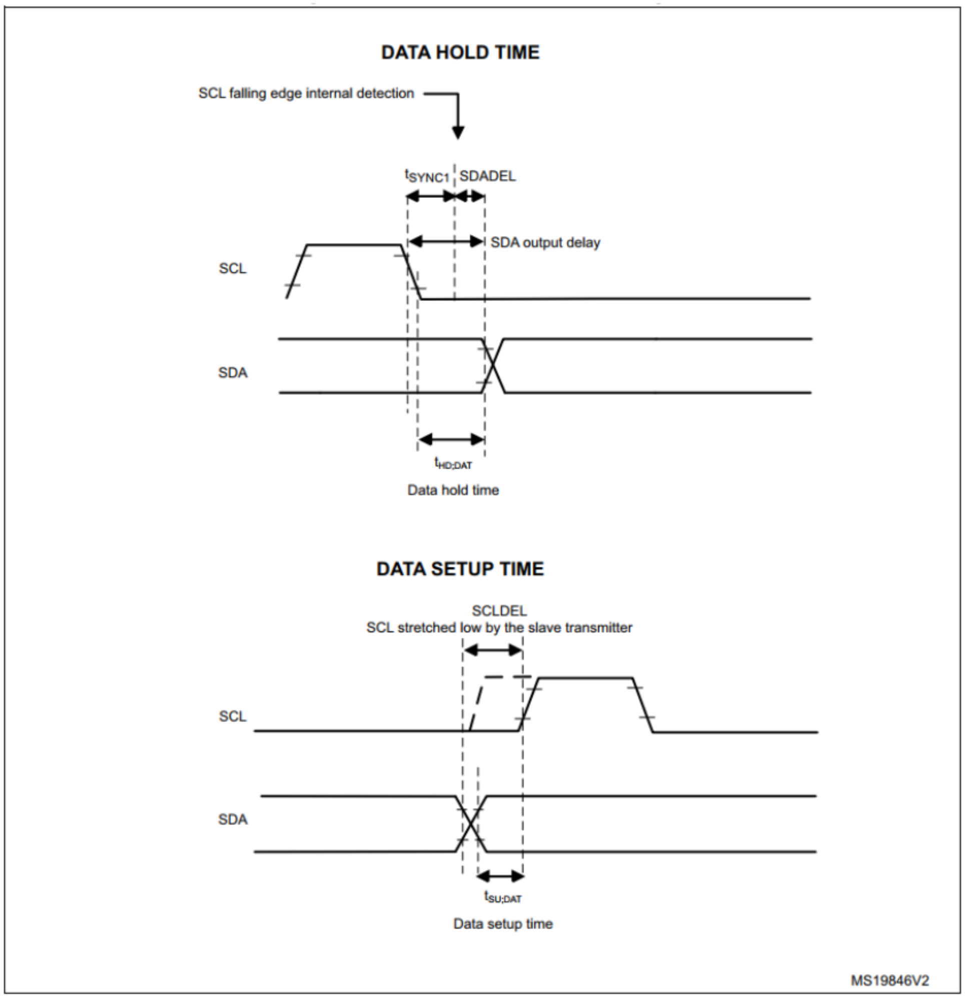


Figure 10: Data Hold and Setup Timing

3. Set your own address in the own address registers. To modify the address, you must first disable the own address. Do this for only Own Address 1 – we do not need Own Address 2 (ensure that it remains disabled).
   (a) Set own address to 7-bit mode.
   (b) Write the own address that you want to use – you are free to use `OwnAddr = 0x52` that is provided in the code.
   (c) Enable own address.
4. Enable I^{2}C in the control register.

In `I2C.c`, study the functions `I2C_SendData()` and `I2C_ReceiveData()`. These are the two functions that you will use to communicate with the temperature sensor. In *Digital Thermal Sensor - TC74 - Data Sheet*, find the command byte that you must send to the temperature sensor to get a temperature measurement.

Read Temperature Command Byte = `0x`____________________

Write code in `main()` that will constantly get temperature measurements from the sensor. Furthermore, display the received measurement on Termite (just the number is sufficient).

4 Part C – SPI

In this part of the lab, you will setup SPI1 and SPI2 to emulate communicating with an SPI device. This is done by connecting the `MOSI1` with `MISO2` and `MOSI2` with `MISO1` pins together with jumper wires. Also, `SCK` of 2 SPI ports should be connected together. Data will be sent from the `MOSI` pin, and received by the `MISO` pin. You will verify that the receive data is equal to the transmitted data. On data match, you will toggle the LED. Each iteration should increment the transmit data. The transmit data should increment from 0 to 9.

You will use **PB3**, **PB4**, and **PB5**, for `SPI1_SCK`, `SPI1_MISO` `SPI1_MOSI`, respectively. And, **PB13**, **PB14**, and **PB15**, for `SPI2_SCK`, `SPI2_MISO` `SPI2_MOSI`, respectively. Configure the pins with the following settings. Write your code in the `SPI1_GPIO_Init()` and `SPI2_GPIO_Init()` function in `SPI.c`.

Pin	**Mode**	**Output Type**	**Output Speed**	**Pull-Up/Down**
PB3	Alt. Function - `SPI1_SCK`	Push-Pull	Very High	No Pull
PB4	Alt. Function - `SPI1_MISO`	Push-Pull	Very High	No Pull
PB5	Alt. Function - `SPI1_MOSI`	Push-Pull	Very High	No Pull
PB13	Alt. Function - `SPI2_SCK`	Push-Pull	Very High	No Pull
PB14	Alt. Function - `SPI2_MISO`	Push-Pull	Very High	No Pull
PB15	Alt. Function - `SPI2_MOSI`	Push-Pull	Very High	No Pull

To complete this section of the lab, you will need to set up SPI, the LED, and the SystemTick.

1. Modify `LED.c` to implement `LED_Init()`.
2. Modify `SysTick_Init()` in `SysTimer.c` to configure a 1ms SysTick interrupt.
3. Implement `SPI1_GPIO_Init()` and `SPI2_GPIO_Init()` in `SPI.c` according to the above table.
4. Implement `SPI1_Init()` and `SPI2_Init()` in `SPI.c`.
   (a) Enable the SPI clock.
   (b) Set the RCC SPI reset bit, then clear it to reset the SPI1 or SPI2 peripheral.
   (c) Disable the SPI enable bit. The peripheral must be configured while it is disabled.
   (d) Configure the peripheral for full-duplex communication.
   (e) Configure the peripheral for 2-line unidirectional data mode.
   (f) Disable output in bidirectional mode.
   (g) Configure the frame format as MSB first.
   (h) Configure the frame format to 8-bit mode.
   (i) Use Motorola SPI mode.
   (j) Configure the clock to low polarity.
   (k) Configure the clock to first clock transition.
   (l) Set the baud rate prescaler to 16.
   (m) Disable hardware CRC calculation.
   (n) Set SPI1 to master mode and SPI2 to slave mode.
   (o) Enable software SSM.
   (p) Enable NSS pulse generation.

(q) Configure the internal slave select bit, 1 for master and 0 for slave.

(r) Set the FIFO threshold to 1/4 (required for 8-bit mode).

(s) Enable the SPI peripheral.

5. Implement `SPI_Send_Byte` in `SPI.c`. When a transfer is performed, the peripheral will shift data in as the master shifts data out. In this exercise we simulate the peripheral with the master.
   (a) Wait for the `Transmit Buffer Empty` flag to become set.
   (b) Write data to the `SPIx->DR` register to begin transmission. Note that the register memory address must be cast to **(volatile uint8_t*)** and dereferenced when using the 8-bit format.
   (c) Wait for the `Busy` to become unset for the transmission to complete.

6. Implement `SPI_Receive_Byte` in `SPI.c`.
   (a) Wait for the `Receive Not Empty` flag to set for the data to be received.
   (b) Read received data from the `SPIx->DR` register. Again note that the register memory address must be cast to **(volatile uint8_t*)** and dereferenced when using the 8-bit format.

7. In the main loop, transfer a byte. Check the received value. If the values match, toggle the LED. Increment the value to transmit such that it cycles from 0 to 9, incrementing by 1 on each loop iteration. (This part is already done for you)

5 Checkoff Requirements

We will check for:

1. In part A with USB, termite can control the LED by sending y/n. If a wrong character is sent, the console should perperly notify users.

2. In part A with Bluetooth, the HC-05 terminal should achieve the same functionality as using termite.

3. in part B, it should read a reasonable temperature repeatedly. When heating it up, the number should go up.

4. In part C, send data using SPI1 and receive data using SPI2

6 References

[1] STM32L4x6 Advanced ARM-based 32-bit MCUs Reference Manual

[2] Yifeng Zhu, "Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C", ISBN: 0982692633

[3] Nucleo Kit with STM32L476VG MCU User Manual

[4] HC-05 Bluetooth Module Datasheet

[5] L3GD20 Gyroscope Datasheet

[6] TC-74 Temperature Sensor Datasheet