Bare Metal Programming

Ranim Tom

May 20, 2025

Contents

Todo list

Chapter 1

Introduction

1.1 Goal of the Reader

In this document, we dive into the fundamental of embedded system in a bare metal fashion, that is writing down code using register manipulation and no library at all.
This will give us understanding how microcontrollers work, manipulate registers, understanding data sheet and reference manuals, and schematics.

1.2 Development Env

For the development environment, that is writing code and uploading the code to the microcontroller, we use stm32cube IDE.

1.2.1 Stmcube Mx

There is also another tool from stm called stmcubmx. For now we don't use this tool. But for the purpose of information, it is a tool in which we can graphically configure our board, and generate the C code.

StmcubeMx

StmcubeMx:

- *To see later the role of this tool along with stmcube IDE, maybe in stmcube mx tutorial*

1.3 Board and Documentation

1.3.1 Board

In this course, we will use the ***stm32F411***. This board is part of the ***nucleo board family***. Another family is the ***discovery board***. These 2 boards may use same microcontroller, but they are 2 different development board (different pin configuration,$\cdots$).

1.3.2 Documents

First we need to download the appropriate resources.

Whenever we want to do bare metal programming, we need to access registers, know their appropriate addresses, their bit location,$\cdots$, that's why we need the reference manual.
In other words, reference manual contains all the information about the registers, their addresses, the fuction of each bit.
Note: When we open the reference manual, we see the acronym `RM0308`, where `RM` stands for reference manual.

The 2nd document is the data sheet, which tells us about the anatomy about our microcontroller: block diagram, pin configurations,$\cdots$. Also, inside the block diagram, we find the different path which link the microcontroller to the CPU.

The 3rd is the user manual. In this document, we can find for example which LED is connected to which pin,$\cdots$
Note: when we open the user guide, we find that there is allot of other stm32 nucleo board family also, that is because all the nucleo board are fabricated in the same way (same anotomy), but the microcontroller is different.

1.4 Hello LED example

As any introductory example, we will try to turn the LED on to test our set up and our board.
Since we want to trun a LED on, we need to know where is the LED is connected to which pin.

1.4.1 Port and Pins

Pins are usually grouped into group of ports, that is for example PA5, which stand port A, pin 5.

To find out where our LED is connected, we open the *user manual* and search for LED (section 7.4 page 24).

Some info about LED:

- there is what we call a power LED, that is once we connect the microcontroller to the pc
- the st-link led
- What we are interested in is the ***user LED***, the led which we can program, and it is connected to PA5.

So from the user manual we found out that the LED in stm32f411 is connected to Port A, pin 5

1.4.2 Ports and Addresses

Now we need to get to port A and pin 5. This can be done using the addresses. That is because portA is a ***peripheral***, and each peripheral in the microcontroller is mapped into some address range.

To get an idea about this mapping (between peripheral or any module and the addresses), we open the ***data sheet*** and go to the ***memory mapping section*** (section 5 page 51). We can see that the memory is disected into different regions, what we are interested in is the ***peripheral section***, which starts from 0x4000 0000 $\rightarrow$ 0xFFFF FFFF.

In the same section, we see also a table that includes bus, peripherals, and the correpondant address.
Now why is that? that is because to turn on each peripheral or module, we need to activate the clock of this peripheral, and to do that, we need buses which connect a peripheral or a module to the bus.

1.4.3 Buses

To understand more, we can go to the functional block diagram in the data sheet, page 16 figure 3. This block represents the internal connections inside the board.
We can see that for example that GPIOA is connected to AHB1 bus. To know that, we see what arrow is touching the GPIOA, and where the other head (other side) is going. We see that it is going to AHB1 bus.
So in other words, if we want to activate the clock for GPIOA, we need to go through AHB1 bus.

Note:

- APB stands for advance peripheral bus $\leftrightarrow$ minimum of 2 clock cycle to access peripherals
- AHB stands for advance high performance bus, which have more speed $\leftrightarrow$ 1 clock cycle to access to perhipherals

1.4.4 Code:

See `1_b_LED_ON_f411` in stm32-dev workspace.

Some notes to be kept in mind:

- Each peripheral (as in GPIOA for example) is identified using its base address, found in the memory map table
  - the peripheral is like a house, and inside the house there are several rooms, these rooms are the registers
  - Each register is responsible for some task, such as setting input or output,$\cdots$
  - The information about the registers are found in the reference manual

- Each peripheral is configured via a clock register, connected using some bus

1.5 Summary

- type of documents:
  - reference manual $\leftrightarrow$ register info, bit functions, memory addresses,
  - data sheet $\leftrightarrow$ anatomy of microcontroller
  - user manual $\leftrightarrow$ LED connections, push buttons, $\cdots$
- the user LED $\rightarrow$ LED we can program

Each peripheral (GPIO,timer,$\cdots$) is like a house identified by its base address (found in data sheet or reference manual in memory map table)

Programming project:

- 2 programming project for turning led ON
- the 1^{st} one via pure register approach: finding each register, and adding the corresponding offset via the base register
- using structure approach, where this approach eliminate the need to add offset, and its added directly via the offset in the structure member

Chapter 2

GPIO

2.1 Introduction

GPIO stands for general purpose input output.

In MCU, pins are grouped into ports, such as port A, port B,$\cdots$.
We have also what we call special purpose or alternate functions of the pin, that is the pin is configured internally to handle something other then general points.
An example is the UART, where we need to configure the pin to receive and send data, or ADC, where we need to configure the pin to read data from analog pin. We understand more about this when reaching these part later.

2.1.1 Type of Register in GPIO Peripheral

We now present different types of registers in GPIO module. These registers are general and not only related to arm microcontroller.

- Direction register: set pin input or output
- Data register: Write data or read data

2.2 BSRR, Push Button

For this chapter for the GPIO, we implemented several exercice:

- `2_a_gpio`: we add a pacakage from stm website, which contains header files implementing all registers and their address range in the memory map, so we can access directly these registers and be able to manipulate them
- `2_b_gpio_bsrr`: we introduce a new register, the BSRR, which can directly manipulate pin input or output (the register contains both)

  BSRR usage

  BSRR usage:

  To see later what is the advantage of using this approach compared to older approach (using ODR register to write data)

- `2_c_gpio_pushbuttonb`: we introduce concept of push button

  Push button logic:

  To see later the point of active low and active high

  the blue button in our board is active low

GPIO:

- *To adapt later this chapter when reaching the GPIO driver also in fastbit academy course*
- *Also to read later chapter GPIO from book of MCU and see the tutorial he make*

Chapter 3

UART

3.1 Introduction

In this chapter we present the 1st communication protocol, UART, which stands for *universal asynchronous transmitter receiver*.

UART has several parameters and concepts. In this section we take an overview before diving into the data sheet and doing some code.

1. Communication Type:

   UART is know to be a ***serial communication protocol***.

   In **??**, we have the difference between serial and parallel communication.

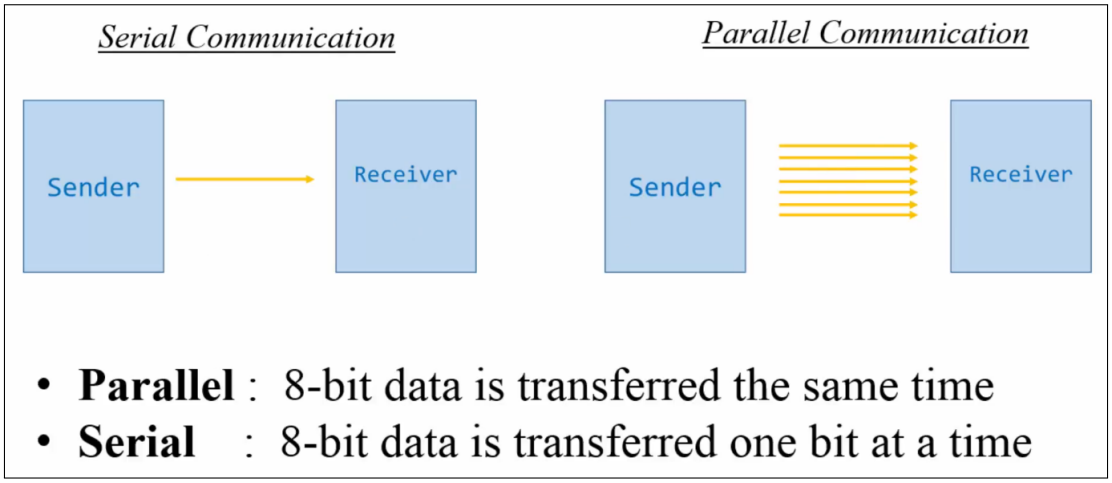

Figure 3.1: Serial vs parallel communication

   - Serial: 1 bit at a time ↔ hence 1 arrow
   - Parallel: multiple bits at a time ↔ hence multiple arrow

2. Serial communication comes in 2 methods:

   (a) Synchronous: communication is via a clock

(b) Asynchronous: no clock is used $\leftrightarrow$ Rx and Tx make convention about the baude rate

3. Transmission mode:

   In **??**, we have all the transmission mode associated with some block diagram to illustrate the particular mode

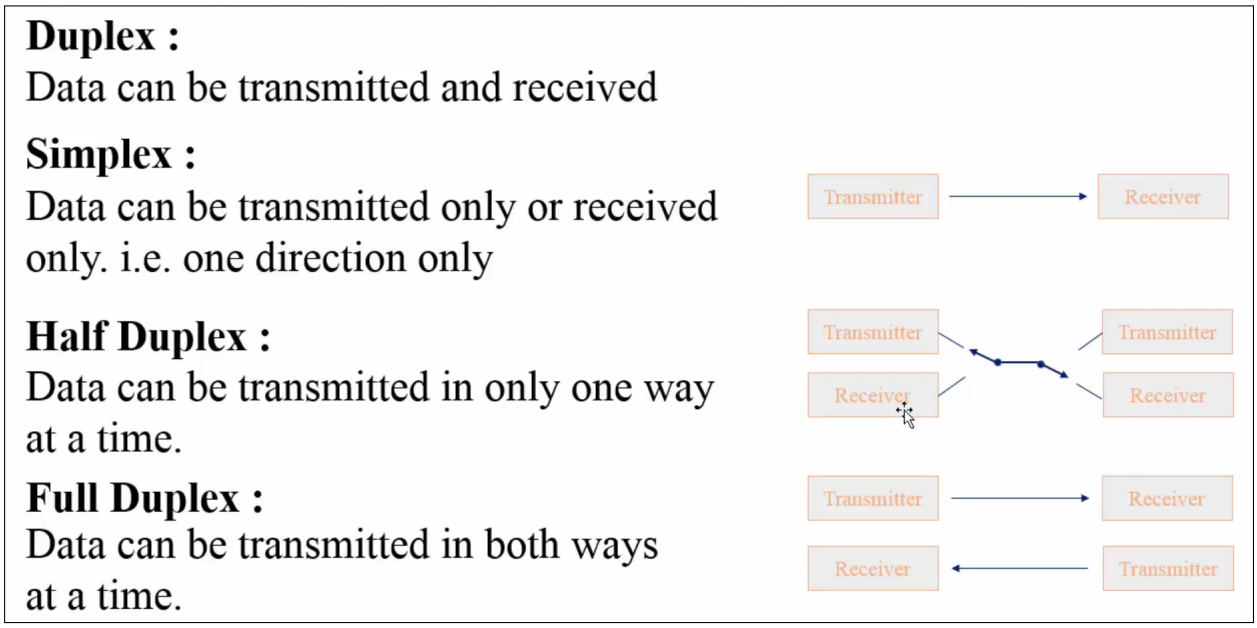

Figure 3.2: Different transmission mode between a Tx and a Rx

- the duplex is the general mode, where we have 2 possibility: either half duplex (data can be sent or received in only way), or full duplex (sent and receive in the same time)
- simplex is only in 1 direction: either transmit or receive

4. Protocol aspect:

   ?? presents an example for sending the character `A`, where beside the actual data, we use ***start bit*** and ***stop bit***

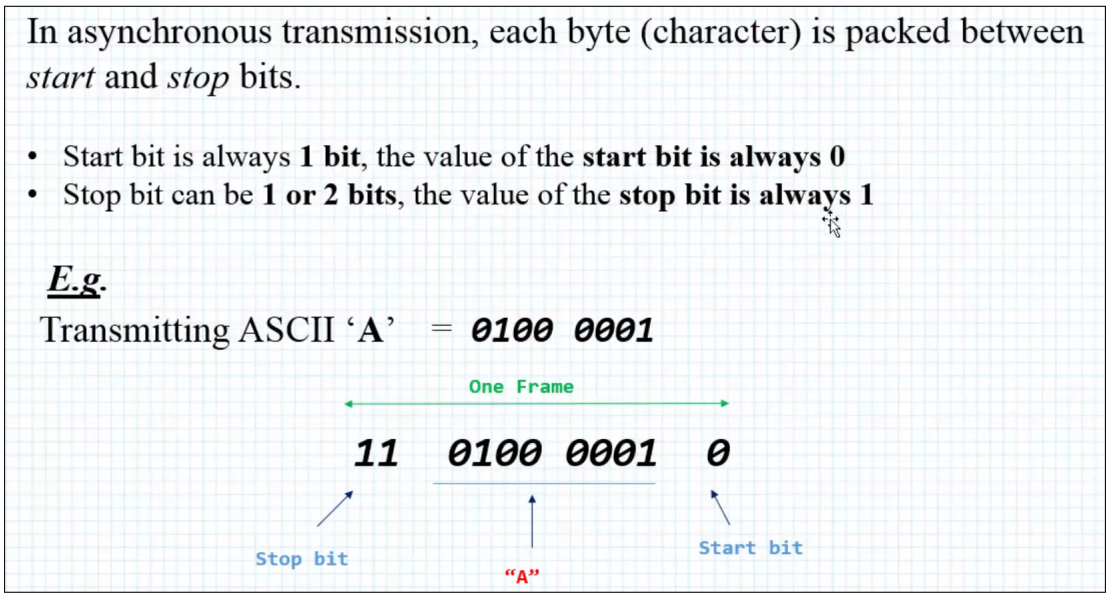

Figure 3.3: Protocol Example: start and stop bit

5. Some other configurations:

   There are also some other miscellaneous configuration

   - Word length: the number of data bits that can be transmitted or received. Can be 8 or 9 bits
   - Mode: Specify whether the Rx or Tx mode is enabled or disabled
   - Parity: for error checking, it comes with even or odd mode.

3.2 Coding Example

1st in order to begin codning with UART, we need to see how many UART module we have in our board. To see that, we open the data sheet on the general block diagram (Figure 3 page 16).

We can see that we have 2 USART module: 1 connected to APB1 bus (named `USART2`), and the others are connected to APB2 bus (named `USART1` and `USART6`).

We are going to use the `USART2` module, because it is connected to a USB port which we can use to communicated to our computer.

Note for hardware connection:

- In stm32F411 nucleo board, it happens that the `USART2` module is connected to the USB port in which we use to flash the firmware of the microcontroller.
- In discovery board for example, the `USART2` module is not connected to the USB port, so we need some converter device to make the USART → to USB conversion.

  UART and Connection: To explore this point later when re-working later with the discovery board

RT and
nnection

3.2.1 UART Clock Registers Configuration

`USART2` module is connected to APB1 bus, which is handled by the `RCC_APB1ENR` register.

In the reference manual (section 6.3.3 page 118), the bit number 17 is responsible for `USART2` module.

3.2.2 Pins Configuration

`USART2` module has a Tx and Rx pins. To know to which GPIO these pins correspond, we open the data sheet to Table 9, the alternate function mapping page 48.

Description and Reading of the table:

- The 1st column contains all ports and pins
- The element of the table contains some modules, such as USART-Tx, USAR-Rx,...
- When we scan for USART2-Tx, USAR2-Rx pins, we see that USART2-Tx corresponds to PA2 and USAR2-Rx corresponds to PA3.
- Also, USART2-Tx and USART2-Rx are in the AF07 column, and we will use that later.

3.2.3 Alternate function register

For Tx application for USART-2 module, we can see that until now we have:

- Tx pin corresponds to PA2, and under AF7

To set the alternate function value, there are 2 register responsible for that: GPIOx-AFRL, which handles the 1^{st} 8 I/O pins (pins $0 \rightarrow 7$), and GPIOx-AFRH which handles the rest (pins $8 \rightarrow 15$)
PA2 is the 2^{nd} pin under GPIO-A, so we need to configure GPIOx-AFRL.

Structure of GPIOx-AFRL:
In the reference manual page 161, we see that GPIOx-AFRL is numerated AFRL0 $\rightarrow$ AFRL7. This means, for example, AFRL0 is for pin 0, AFRL1 is for pin 1, and so on.

In our case, we need to set AFRL2 (because we use PA2) to 0111 (which corresponds to Af7).

Programming Note:

- In the header files provided by stm, we don't have GPIOA$\rightarrow$AFRL. Instead, we have GPIAO$\rightarrow$AFR[], where AFR is an array of size 2.
- GPIAO$\rightarrow$AFR[0] is for AFRL and GPIAO$\rightarrow$AFR[1] is for AFRH

3.3 Summary UART

- UART Tx and Rx pins mapping to GPIO: we use the alternate function mapping table, in page 48 in the data sheet (see **??**).