Embedded DSP

Ranim Tom

November 9, 2023

Contents

Todo list

Chapter 1

Introduction

1.1 Goal of the Reader

The purpose of this pseudo book is to document embedded DSP with STM32 microcontroller family.

Adapting Introduction:to adapt the introduction later when finishing the reports, like adding structure of each chapters for example,$\cdots$.

Adapting In troduction

1.2 Some Prerequisite: Boards and 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 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,$\cdots$).

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 for example which LED is connected to which pin,$\cdots$

1.2.1 CMSIS

The `CMSIS` package (stand for Common Microcontroller Software Interface Standard) contains a library which will help us to accelerate our development. As a concrete example related to our simulations, the `CMSIS` contains all the memory mapped for the GPIO, and clock also, so we won't implement them from scratch.

To download this package,

1. go to stm website
2. At the search bar: type stm32F4 (the microcontroller we are using)
3. At the tab, go to Tools and Software
    - Select STM32CubeF4 pacakge and dowload it

After we download the package, we will use only the necessary folders required in this development.
In the project we created (named `DSP-target-stm32F411`), we make a new folder call `chip-header`, which will include the necessary folders from the download package (since we will not use all the package)
We navigate in `\STM32Cube_FW_F4_V1.27.0\Drivers\CMSIS`, and inside `CMSIS` we copy the `include` and `device` folders in `chip-header` folder.
Also, we delete inside `device` directory the following folders: `Template` (located in `Chip_Header\CMSIS\Device\ST\STM32F4xx\Source`)

1.3 Test Board

In project `Board-Test` we have some simple script to blink the LED using the `CMSIS`.

In order to use the `Include` folder from the `CMSIS`, we need to link it to the project. To do this:

- Right click on the project name (after we create the project)
- Go to `Propoerties --> C/C++ General --> Path and Sumbols`
- Under the `include` tab, we add the path of our `Include` folder (which is in my case `E:\1_Simulations_Programming\Embedded Programming\DSP_target_stm32F411 Board_Test\Chip_Header\CMSIS\Device\ST\STM32F4xx\Include`)

The Generic path:

Another solution is include the following path instead of the previous one (because some other computer will not have the same path) `$(ProjDirPath)\Chip_Header\CMSIS \Device\ST\STM32F4xx\Include`.
The `$(ProjDirPath)` is a generic command, so when we use the project in some computer which have some different absolute path, it will be generic.

Note on Include files:

Also, when writing the statement `#include"stm32f4xx.h"`, we need to add the symbol in order for IDE to identify which target board we are using. In order to do this , we open the header file `stm32f4xx.h`.

Symbol to identify target: *To write it later, and explain the 2 solutions..*

Symbol to identify target

1.4 Summary Introduction

- Board used in this dev: stm32F411 nucleo board, which is based on ARM cortex M4 architecture
- Embedded Documents:

  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***.
- Clock in microcontroller: modern day microcontroller architecture use a concept called clock gating, meaning no direct access to the clock by default

Chapter 2

Basic of signals in stmcube IDE

2.1 Signal Drawing

First we will see how to draw a signal using ***logic analyzer*** which is part of our debugger in stm32 cube IDE. Later we will develop a UART driver to send signal to an external plotter to be plotted.

Note: when we say logic analyzer, we mean the internal one, we are not using an external one.

logic analyz

logic analyzer: To see later how to use external logic analyzer in other course in Arm.

2.1.1 Signal used

In this part, we will draw a signal stored in `.txt` file. Later we will see how to acquire signal from ADC and draw them.

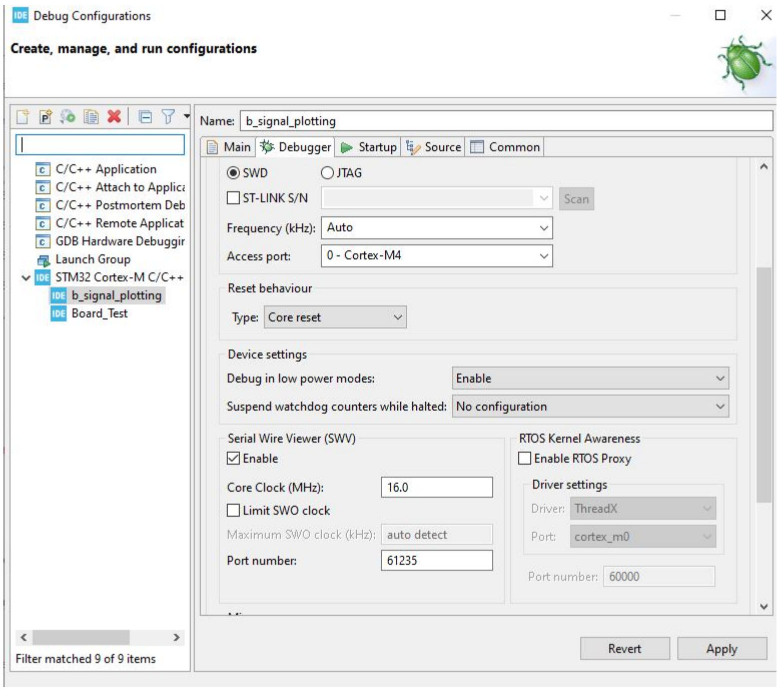
2.1.2 Enabling FPU

The next important step is to enable the FPU (stands for floating point unit).
key concept: every microcontroller comes with 2 type of peripheral: a manufacturer peripheral (like USB,I2C,$\cdots$), and core peripheral (which are implemented by arm in our case). The FPU is one of these core peripheral, and to enable it, we nee to refer to ***arm Cortex M4 generic user guide***.

2.1.3 Internal view using SWV

After we run the code:

- Right click on the project and select `Debug as Stm32 cortex C/C++ Application` as shown in Figure 2.1.



Debug Configurations

Create, manage, and run configurations

C/C++ Application
C/C++ Attach to Applic
C/C++ Postmortem Deb
C/C++ Remote Applicat
GDB Hardware Debuggir
Launch Group
STM32 Cortex-M C/C++
b_signal_plotting
Board_Test

Filter matched 9 of 9 items

Name: b_signal_plotting

Main | Debugger | Startup | Source | Common

(●) SWD ( ) JTAG
☐ ST-LINK S/N Scan
Frequency (kHz): Auto
Access port: 0 - Cortex-M4

Reset behaviour
Type: Core reset

Device settings
Debug in low power modes: Enable
Suspend watchdog counters while halted: No configuration

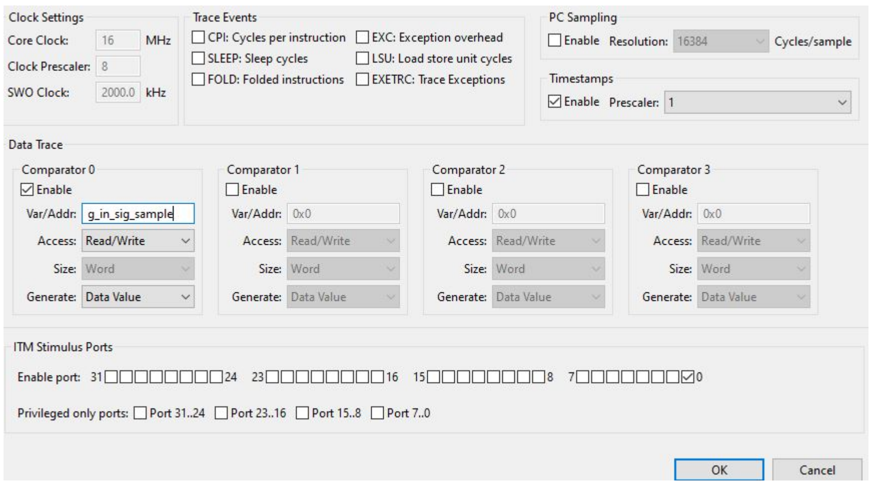
Serial Wire Viewer (SWV)
☑ Enable
Core Clock (MHz): 16.0
☐ Limit SWO clock
Maximum SWO clock (kHz): auto detect
Port number: 61235

RTOS Kernel Awareness
☐ Enable RTOS Proxy
Driver settings
Driver: ThreadX
Port: cortex_m0
Port number: 60000

Revert Apply

Figure 2.1: Enabling port to see global variable

- When the window open, go to `Debugger` tab, and enable `SWV` option (stretch the window down if you don't see it)
- `Window --> Show View --> SWV --> SWV Data trace timeline graph`
- Now on the gear port, enable the global variable as shown in Figure 2.2



Clock Settings
Core Clock: 16 MHz
Clock Prescaler: 8
SWO Clock: 2000.0 kHz

Trace Events
☐ CPI: Cycles per instruction ☐ EXC: Exception overhead
☐ SLEEP: Sleep cycles ☐ LSU: Load store unit cycles
☐ FOLD: Folded instructions ☐ EXETRC: Trace Exceptions

PC Sampling
☐ Enable Resolution: 16384 Cycles/sample

Timestamps
☑ Enable Prescaler: 1

Data Trace

	Comparator 0	Comparator 1	Comparator 2	Comparator 3
Enable	☑	☐	☐	☐
Var/Addr:	g_in_sig_sample	0x0	0x0	0x0
Access:	Read/Write	Read/Write	Read/Write	Read/Write
Size:	Word	Word	Word	Word
Generate:	Data Value	Data Value	Data Value	Data Value

ITM Stimulus Ports
Enable port: 31☐☐☐☐☐☐☐☐24 23☐☐☐☐☐☐☐☐16 15☐☐☐☐☐☐☐☐8 7☐☐☐☐☐☐☐☑0
Privileged only ports: ☐ Port 31..24 ☐ Port 23..16 ☐ Port 15..8 ☐ Port 7..0

OK Cancel

Figure 2.2: Enabling port to see global variable

2.2 Developing UART driver

Now we will plot signal using UART protocol communication.

Important points:

- When we open the data sheet to the function block diagram, we can see that we have many UART modules (`USART1`,`USART2`,$\cdots$).
- We can see that at some `USART2` for example, we have `RX,TX` as `AF` (in alternate function mode)
  - Whenever we need to connect to some peripheral, we need to configure `GPIO` pin (here in `AF` mode to configure `RX,TX`)
- Because `UART` is a physical communication protocol, it needs some wire to communicate. Now our computer doesn't have wires, but it has `USB` port to communicate
  - Now internally, `USART2` is connected to the `USB` port used to flash in and debug our microcontroller, so we can use it.
  - This is only true for nucleo board, discovery board for example doesn't have such connection in `USART2`
- Now `TX,RX` are `GPIO` in `AF` mode, so we need to go to `AF` mapping table, which can be found in the data sheet (Table 9 page 47)
  - We can see by inspecting the table, that for `USART2`, `PA2` and `PA3` are the port to be configured
  - Also we need to get need to configure clock configuration for `PA`
- Also, we can see from the functional block diagram that `USART2` is connected to the `APB` bus, so we need to configure the clock of this bus

2.2.1 Developing Tx function

In this section we will develop the tx function.

- configure `GPIOA` clock: via `RCC AHB1 ENR` register (because GPIOA uses `AHB1` bus)

2.3 Summary Basic of Signals in stm32cube IDE

- Peripheral types: core and manufacture

  every microcontroller comes with 2 type of peripheral: a manufacturer peripheral (like USB,I2C,$\cdots$), and core peripheral (which are implemented by arm in our case). The FPU is one of these core peripheral, and to enable it, we nee to refer to ***arm Cortex M4 generic user guide***