



INTRODUCTION TO MURASAKI

An STM32 Peripheral Class Library

STAY HOME



CONTENTS



INTRODUCTION



WHAT IS
MURASAKI?

Photo by Daniele Levi Pelati on Unsplash



IMPORT AND
RUN

Photo by Daniele Andy Li on Unsplash



DOCUMENTATION

Photo by Debby Hudson on Unsplash



HOW TO
CREATE YOUR
OWN
APPLICATION

Photo by Sneaky Elbow on Unsplash



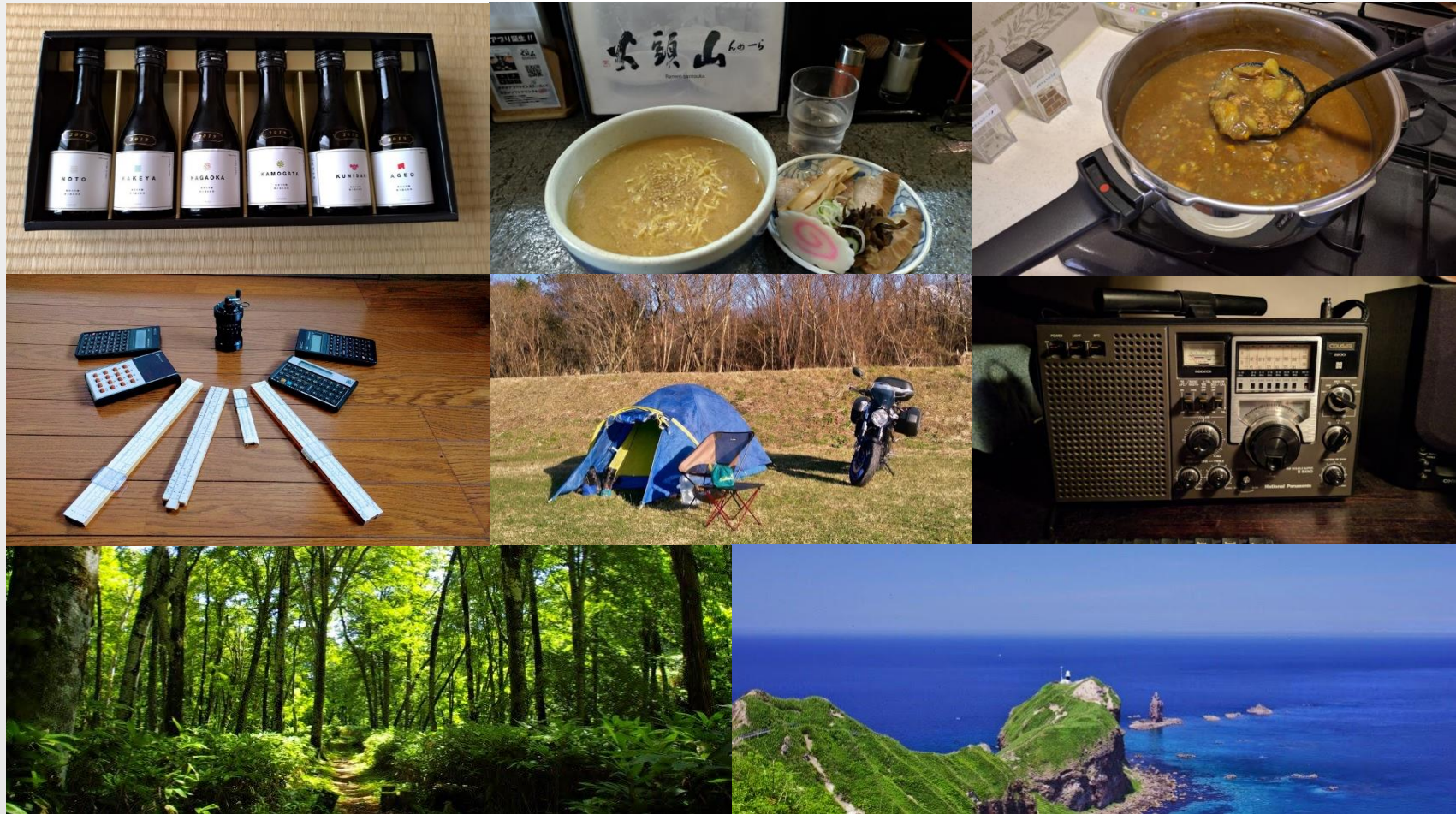
SUMMARY

Photo by Aaron Burden on Unsplash



INTRODUCTION

WHO AM I?





WHAT IS MURASAKI?

IN THIS SECTION

- What is Murasaki?
- What is the target processor?
- What is the development environment?
- What is the overview of the classes?

OVERVIEW

- A collection of the peripheral wrapper class
 - UART, SPI, I2C, ADC, Encoder, SAI, I2S, EXTI, GPIO
- RTOS aware classes are available
 - Task, Synchronizer, CriticalSection
- IO APIs are :
 - Wrapped by Class.
 - Synchronized : IO call returns when IO function is complete.
 - Blocking : If IO is occupied, wait until it becomes vacant.
- Hosted by GitHub
 - <https://github.com/suikan4github/murasaki>

TARGET

- STM32 microcomputer series.
- Core processor is not matter.
- Providing one unified API through the STM32 variants
- Tested Target
 - STM32F0 : CORTEX-M0
 - STM32G0 : CORTEX-M0+
 - STM32L1 : CORTEX-M3
 - STM32F4 : CORTEX-M4
 - STM32G4 : CORTEX-M4
 - STM32F7 : CORTEX-M7
 - STM32H7 : CORTEX-M7

REQUIRED ENVIRONMENT

- Linux or Windows
 - Developed on Ubuntu 16.04 LTS
 - Windows 10 + WSL is confirmed
 - MacOS is not tested
- CubeIDE 1.3
 - Device Configuration Tool is essential
 - Makefile build must be acceptable. But not tested.
- Other tools
 - Doxygen
 - pdflatex
 - Terminal emanator

MOTIVATION

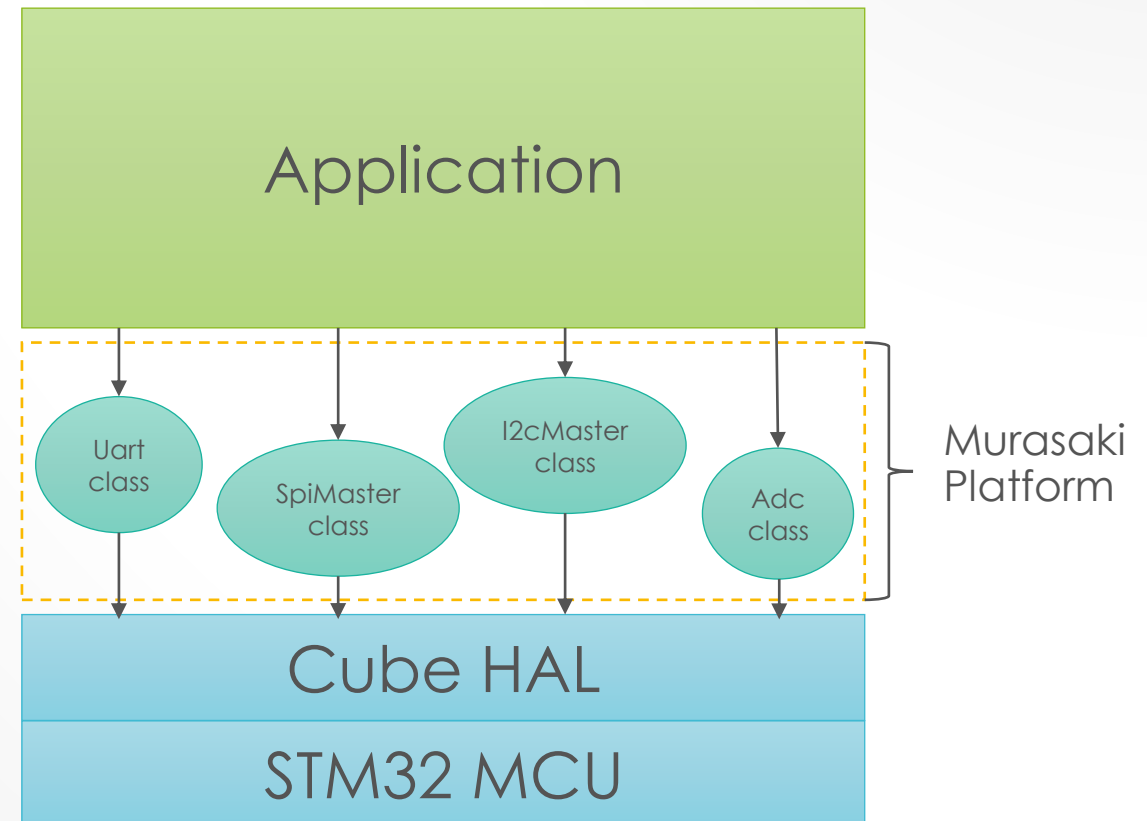
- A class library which doesn't bother the development.
 - Strict name space
 - Strict parameter typing
 - Synchronous and blocking IO API
 - Multi-task aware
 - Enough speed to support audio DMA data
- Strict parameter checking
- Context free printf()

OUT OF SCOPE

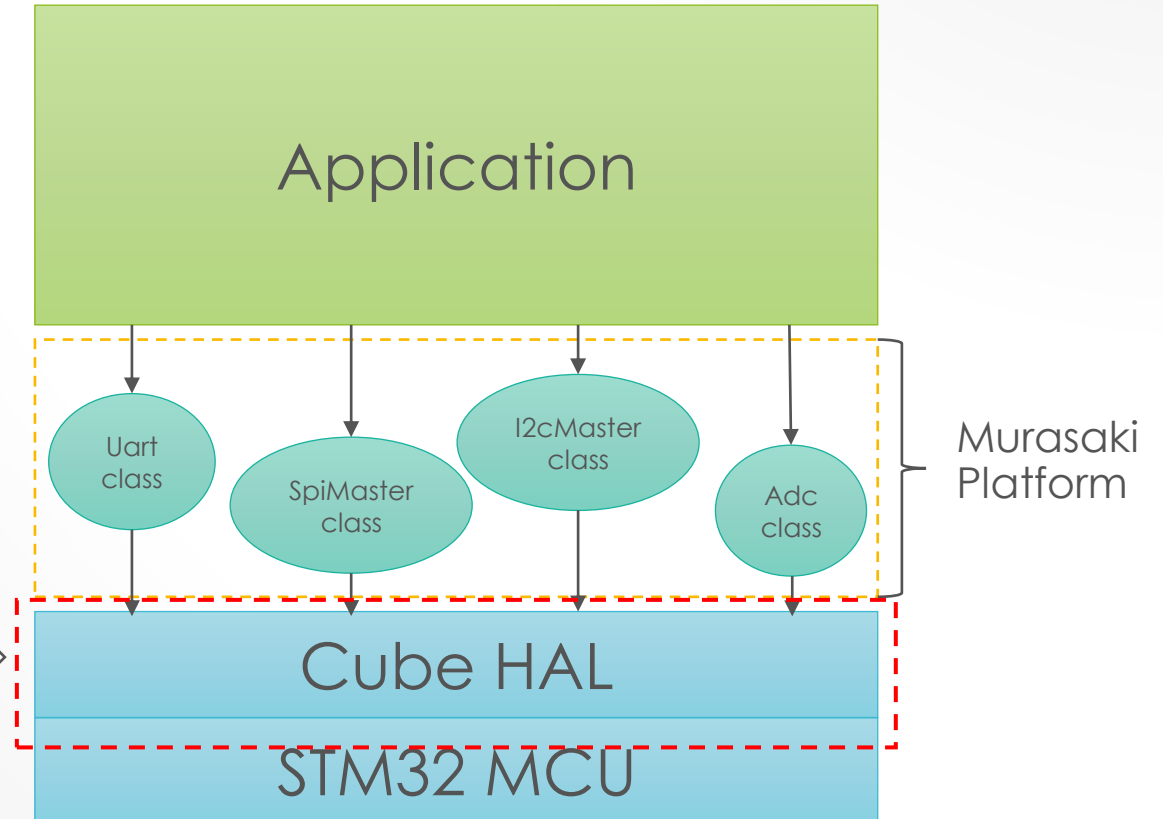
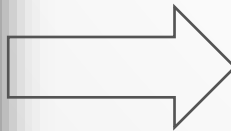
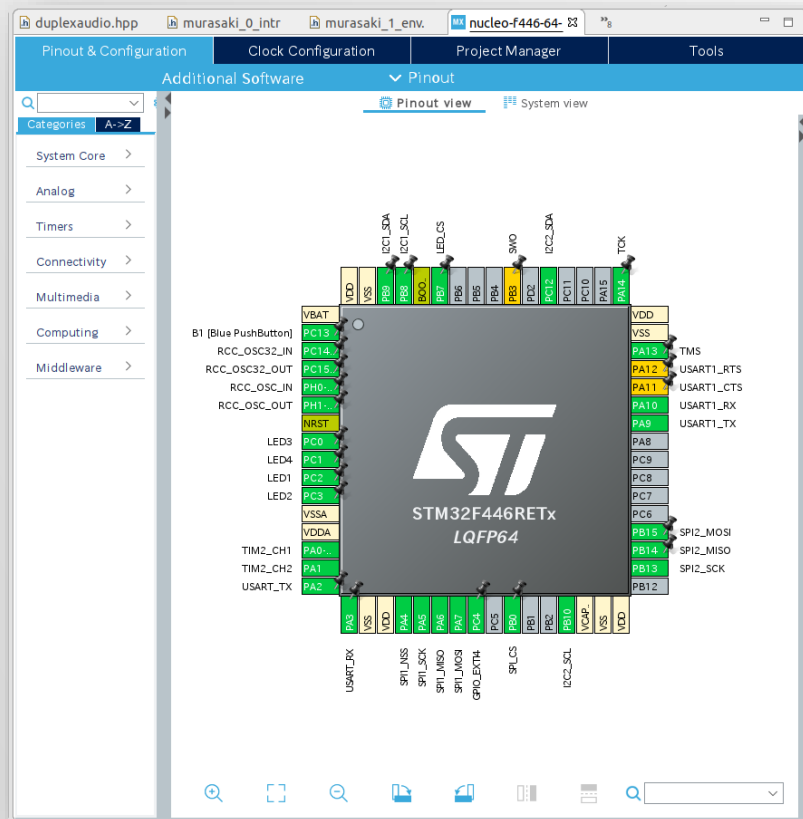
- Very quick response processing
 - Motor control
 - Power supply control
- Very low power operation
 - Mouse
 - TV remote controller
- Very small memory footprint
- High reliability application

PLATFORM AS ABSTRACT LAYER

- Murasaki provides dedicated class for each peripheral
- Application will use the peripheral through these class variable.
- Programmer can create these class variable as he/she wants.



CUBE IDE AND MURASAKI



PRINTF() DEBUG SUPPORT

- A dedicated Printf() function is available.
- Thread safe
- Bi-context
 - Can use from both task and interrupt context
- Buffered
 - As soon as text is stored in the buffer, Print() returns.

```
116 void ExecPlatform()
117 {
118
119     murasaki::debugger->Printf("\n");
120     murasaki::debugger->Printf("Mandelbrot set by NUCLEO-G431RB\n");
121
122     // Render the set.
123     for (int y = 0; y < YPICSIZE; y++) {
124         for (int x = 0; x < XPICSIZE; x++)
125             // print a message with counter value to the console.
126             murasaki::debugger->Printf("%c",
127                                     mapper[mandelbrot(XPOS(x),
128                                                         YPOS(y),
129                                                         sizeof(mapper) - 1)]);
130         murasaki::debugger->Printf("\n");
131     }
132     // Loop forever
133     while (true) {
134         // Blink LED
135         murasaki::platform.led->Toggle();
136         // wait for a while
137         murasaki::Sleep(500);
138     }
139 }
140
141
```

UART

- UART is packed in the Uart class.
- Transmit() / Receive () member functions are :
 - Synchronous
 - Blocking

```
154
155 void Test101Master(void) {
156     // Start UART simple test.
157     murasaki::platform.test_state = cmd_101;
158     murasaki::platform.test_success = false;
159
160     murasaki::platform.sync_command->Release();    // tell slave tas
161
162     murasaki::platform.uart->Transmit(
163         tx_data_uart,
164         sizeof(tx_data_uart)
165     );
166
167     murasaki::platform.sync_ack->Wait();
168     murasaki::debugger->Printf(
```


I2C

- I2C is packed in the I2cMaster and I2cSlave class.
- Transmit () / Receive() member functions are :
 - Synchronous
 - Blocking

```
208   murasaki::platform.sync_command->Release();    // tell slave task next
209
210   murasaki::platform.i2c_master->Transmit(
211       i2c_device,
212       tx_data_i2c,
213       sizeof(tx_data_i2c),
214       &transferred_count);
215
```

SPI

- SPI is packed in the SpiMaster and SpiSlave class.
- TransmitAndReceive() member function is :
 - Synchronous
 - Blocking

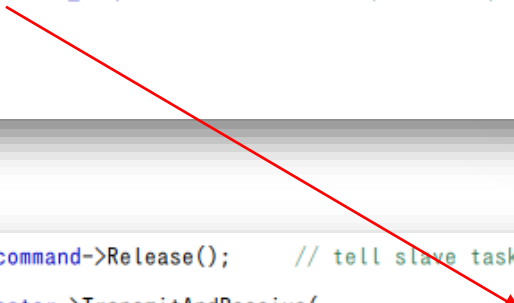
```
murasaki::platform.sync_command->Release(); // tell slave task next test
murasaki::platform.spi_master->TransmitAndReceive(
    murasaki::platform.slave_adapter,
    tx_data_spi_master,
    rx_data_spi_master,
    sizeof(tx_data_spi_master));
```

BY THE WAY, SPI IS CONSIDERED HARMFUL

- The clock polarity and the clock phase is up to the slave.
 - For each time to access different slave, the master have to be re-configured.
- Muarsaki uses the SpiSlaveAdapter class to specify these configuration.

```
129 // CPOL and CPHA follows pin configuration of SPI 1.  
130 murasaki::platform.slave_adapter = new murasaki::SpiSlaveAdapter(  
131     0,  
132     0,  
133     SPI_CS_GPIO_Port,  
134     SPI_CS_Pin);  
135
```

```
murasaki::platform.sync_command->Release(); // tell slave task next test  
murasaki::platform.spi_master->TransmitAndReceive(  
    murasaki::platform.slave_adapter,  
    tx_data_spi_master,  
    rx_data_spi_master,  
    sizeof(tx_data_spi_master));
```



AUDIO

- Audio is packed in the DuplexAudio class.
- TransmitAndReceive() member functions is :
 - Synchronous
- DuplexAudio is not blocking IO.
 - This IO doesn't assume multiple task access the IO randomly
- Both I2S and SAI port are supported

```
220 while (true) // Talk Through
221 {
222     // Wait the end of current audio transmission & receive.
223     // Then, copy the tx buffer to tx DMA buffer.
224     // And then copy the rx DMA buffer to rx buffer.
225     murasaki::platform.audio->TransmitAndReceive(
226                                     tx_left,
227                                     tx_right,
228                                     rx_left,
229                                     rx_right);
230     // Copy RX to TX : talk through
231     for (int i = 0; i < AUDIO_CHANNEL_LEN; i++) {
232         tx_left[i] = rx_left[i];
233         tx_right[i] = rx_right[i];
234     }
235
236     // Blink status.
237     murasaki::platform.led_st0->Toggle();
238     murasaki::platform.led_st1->Toggle();
239 }
240 }
241 }
```

GPIO

- GPIO class
 - GpIn
 - GpOut
- Simple bit operations

```
212
213 // Initiate the LED on the AKSAHI 02 board
214 murasaki::platform.led_st0->Clear();
215 murasaki::platform.led_st1->Set();
216
217
218
219
220 while (true) // Talk Through
221 {
222     // Wait the end of current audio transmission & receive.
223     // Then, copy the tx buffer to tx DMA buffer.
224     // And then copy the rx DMA buffer to rx buffer.
225     murasaki::platform.audio->TransmitAndReceive(
226                                     tx_left,
227                                     tx_right,
228                                     rx_left,
229                                     rx_right);
230
231     // Copy RX to TX : talk through
232     for (int i = 0; i < AUDIO_CHANNEL_LEN; i++) {
233         tx_left[i] = rx_left[i];
234         tx_right[i] = rx_right[i];
235     }
236
237     // Blink status.
238     murasaki::platform.led_st0->Toggle();
239     murasaki::platform.led_st1->Toggle();
240 }
```

OTHER PERIPHERALS

- ADC is packed in the Adc class.
- Convert member function is :
 - Synchronous
 - Blocking
- EXTI is packed in the Exti class.
- Wait member function is :
 - Synchronous
- Exti is not blocking.
 - Only one task wait for specific interrupt.

MULTI-TASKING

- Task is easy to create.
- Synchronizer class for
 - Wait
 - Release
- CriticalSection class for
 - Inter-task exclusive resource access.

```
82 // For demonstration of FreeRTOS task.
83 murasaki::platform.task1 = new murasaki::SimpleTask(
84                                     "task1",
85                                     256,
86                                     murasaki::ktpNormal,
87                                     nullptr,
88                                     &TaskBodyFunction
89                                     );
90 MURASAKI_ASSERT(nullptr != murasaki::platform.task1)
91
```

```
131 /* ----- User Functions ----- */
132 /**
133  * @brief Demonstration task.
134  * @param ptr Pointer to the parameter block
135  * @details
136  * Task body function as demonstration of the @ref murasaki::SimpleTask.
137  *
138  * You can delete this function if you don't use.
139  */
140 void TaskBodyFunction(const void *ptr) {
141
142     while (true) // dummy loop
143     {
144         murasaki::platform.led->Toggle(); // toggling LED
145         murasaki::Sleep(700);
146     }
147 }
148
```

AUTOMATIC INTERRUPT HANDLING

- Peripheral interrupts are handled internally.
- Programmer doesn't need to care.

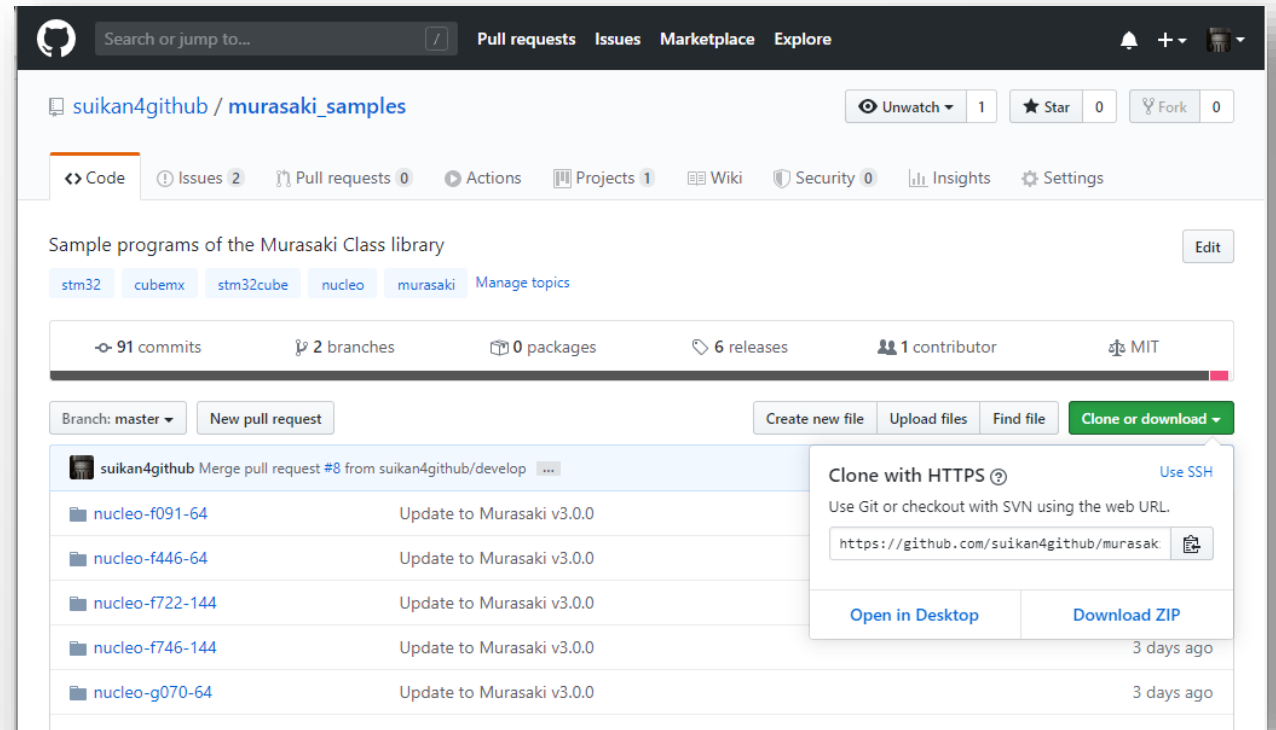


IMPORT AND RUN

Photo by Daniele Andy Li on Unsplash

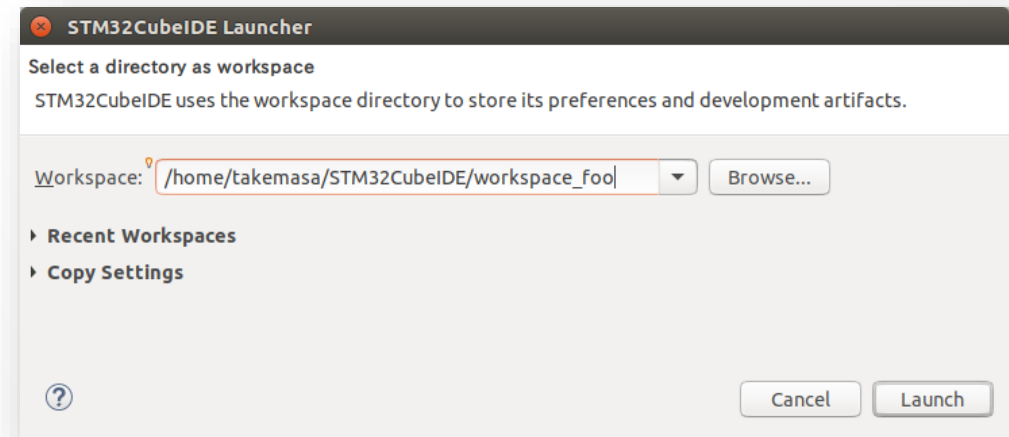
IN THIS SECTION

- We fetch the sample programs from github, and import
- Run the program
- Walkthrough the program
- See the debug functionality

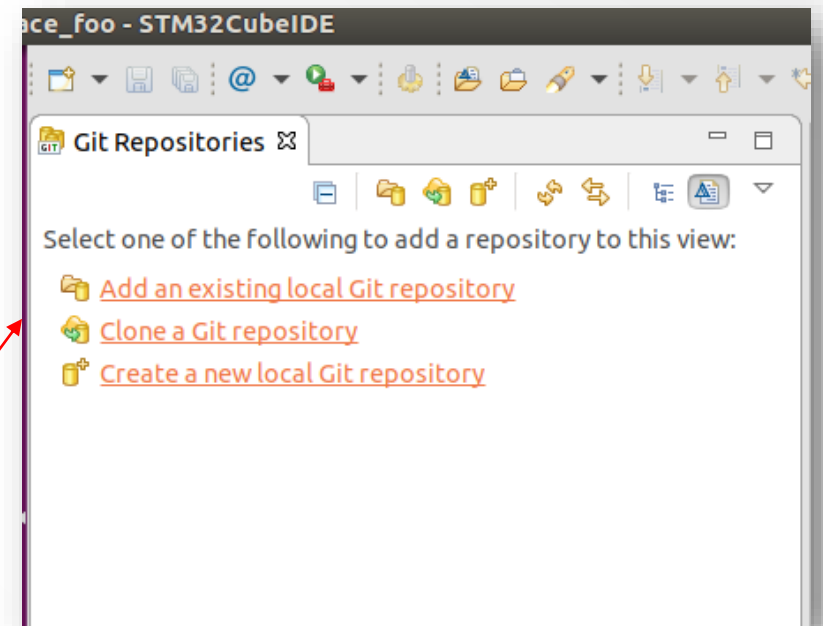
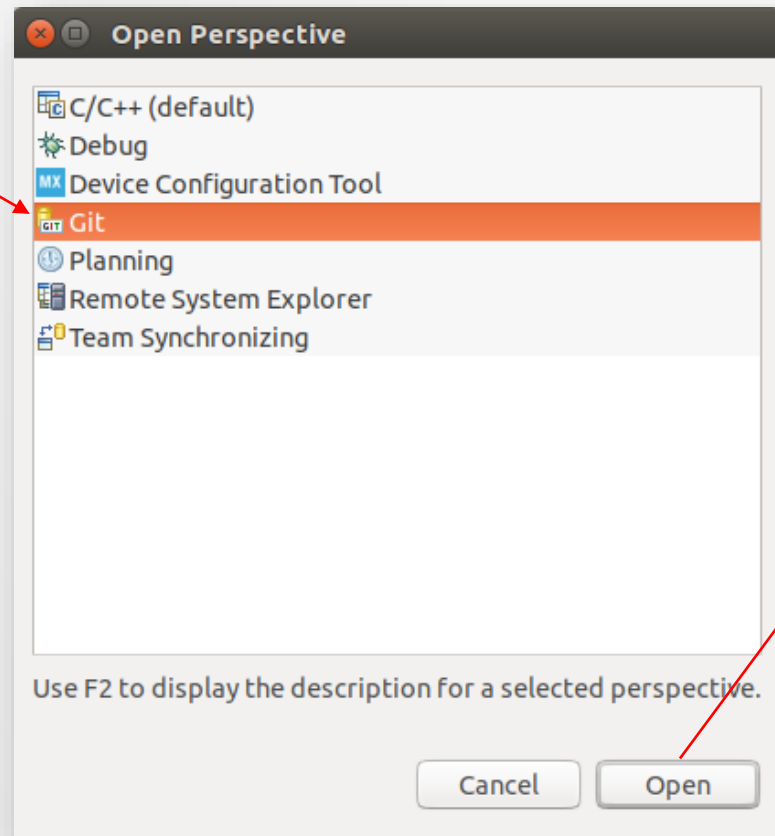
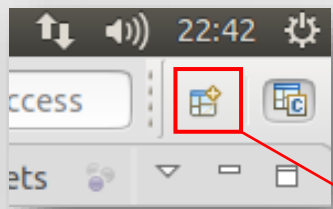


1. OPNE A NEW WORKSPACE

- Create a new workspace



2.OPEN THE GIT PERSPECTIVE



3. COPY AN URL OF THE REPOSITORY

The screenshot shows the GitHub interface for the repository 'suikan4github/murasaki_samples'. The repository name is highlighted with a red dashed box. The 'Clone or download' button is also highlighted with a red dashed box, and its dropdown menu is open. The 'Clone with HTTPS' option is selected and highlighted with a red dashed box, and its corresponding URL is also highlighted with a red dashed box. A red arrow points from the 'Clone or download' button to the 'Clone with HTTPS' option. The dropdown menu also shows 'Use SSH' and 'Download ZIP' options.

suikan4github / **murasaki_samples**

Unwatch 1 Star 0 Fork 0

Code Issues 2 Pull requests 0 Actions Projects 1 Wiki Security 0 Insights Settings

Sample programs of the Murasaki Class library Edit

stm32 cubemx stm32cube nucleo **murasaki** Manage topics

91 commits 2 branches 0 packages 6 releases 1 contributor MIT

Branch: master New pull request

Create new file Upload files Find file **Clone or download**

suikan4github Merge pull request #8 from suikan4github/develop ...

Clone with HTTPS Use SSH

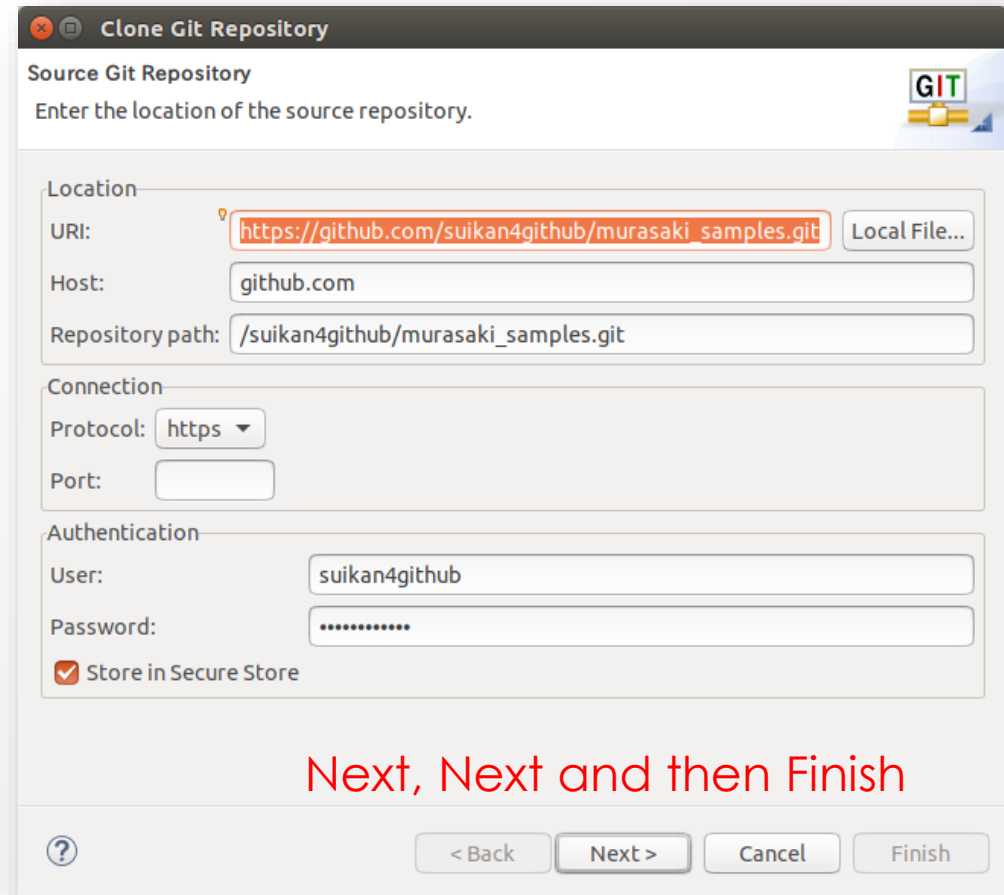
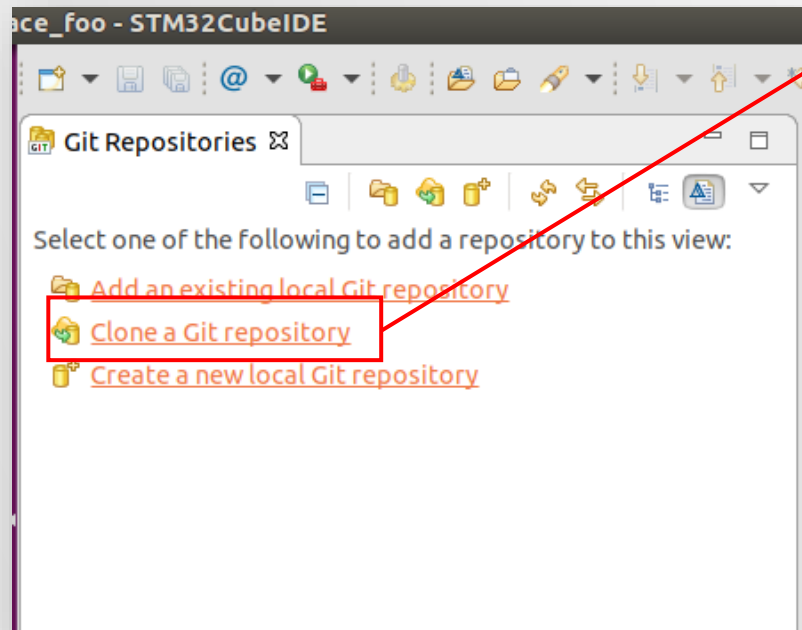
Use Git or checkout with SVN using the web URL.

`https://github.com/suikan4github/murasaki`

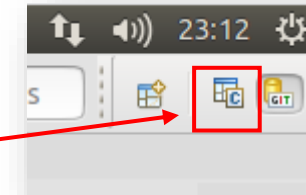
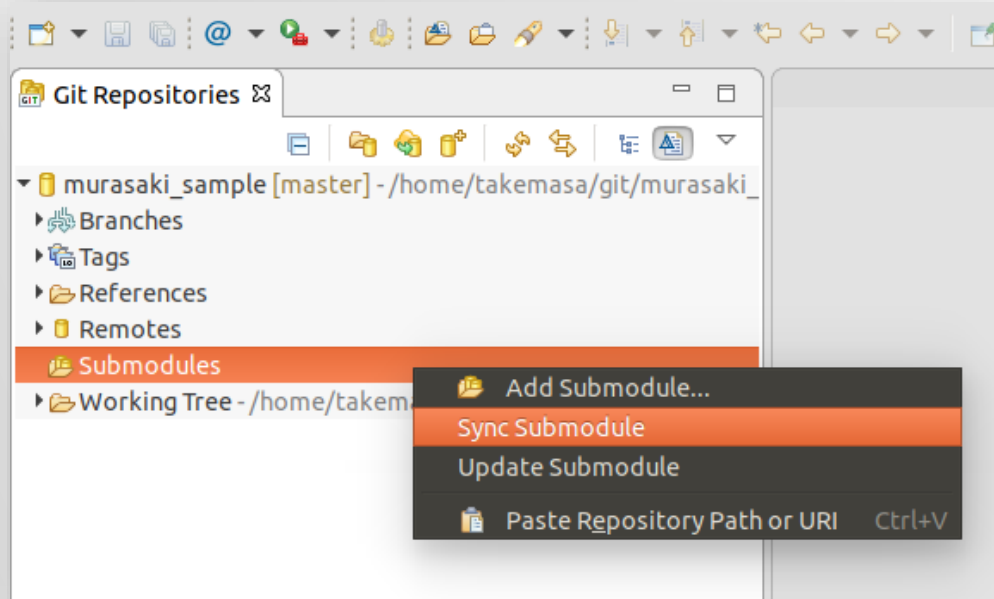
Open in Desktop Download ZIP

| | | |
|-----------------|---------------------------|------------|
| nucleo-f091-64 | Update to Murasaki v3.0.0 | 3 days ago |
| nucleo-f446-64 | Update to Murasaki v3.0.0 | 3 days ago |
| nucleo-f722-144 | Update to Murasaki v3.0.0 | 3 days ago |
| nucleo-f746-144 | Update to Murasaki v3.0.0 | 3 days ago |
| nucleo-g070-64 | Update to Murasaki v3.0.0 | 3 days ago |

4. CLONE THE REPOSITORY

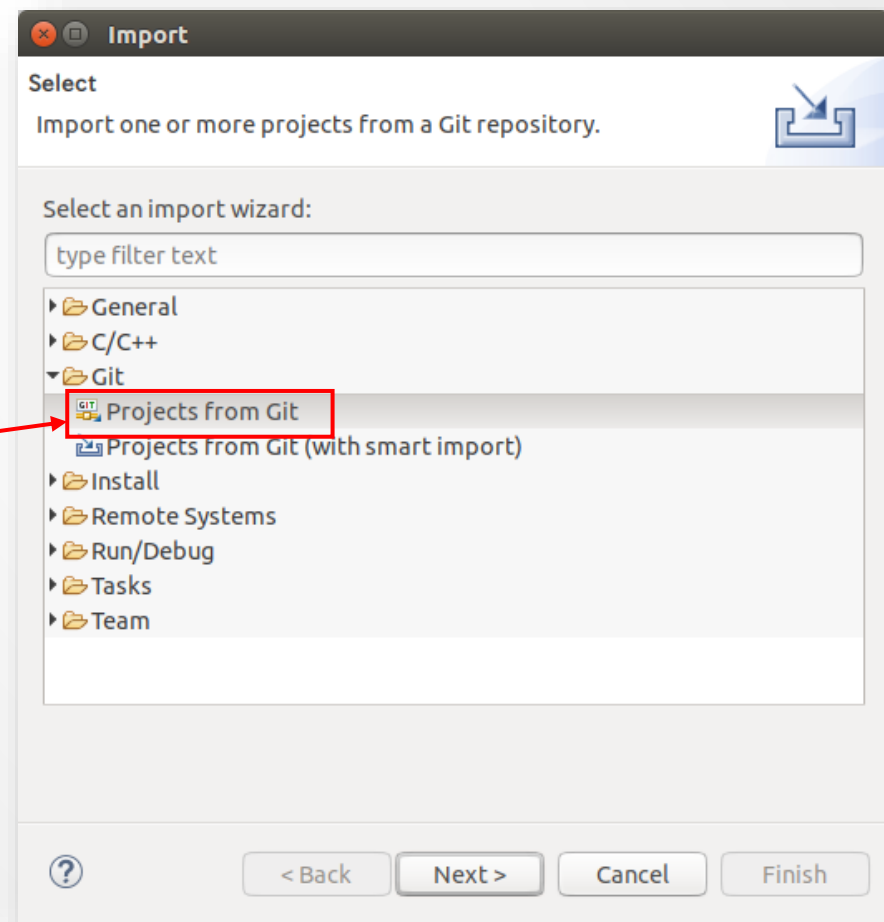
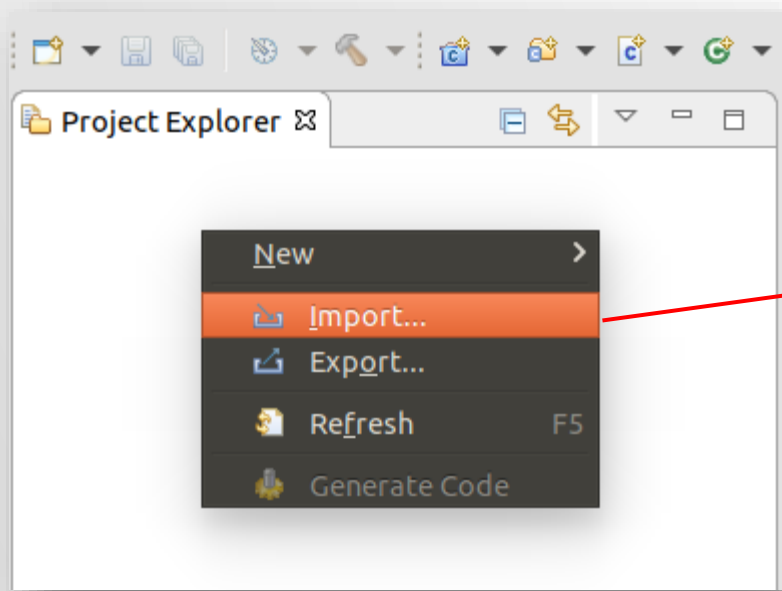


5.UPDATE SUBMODULES

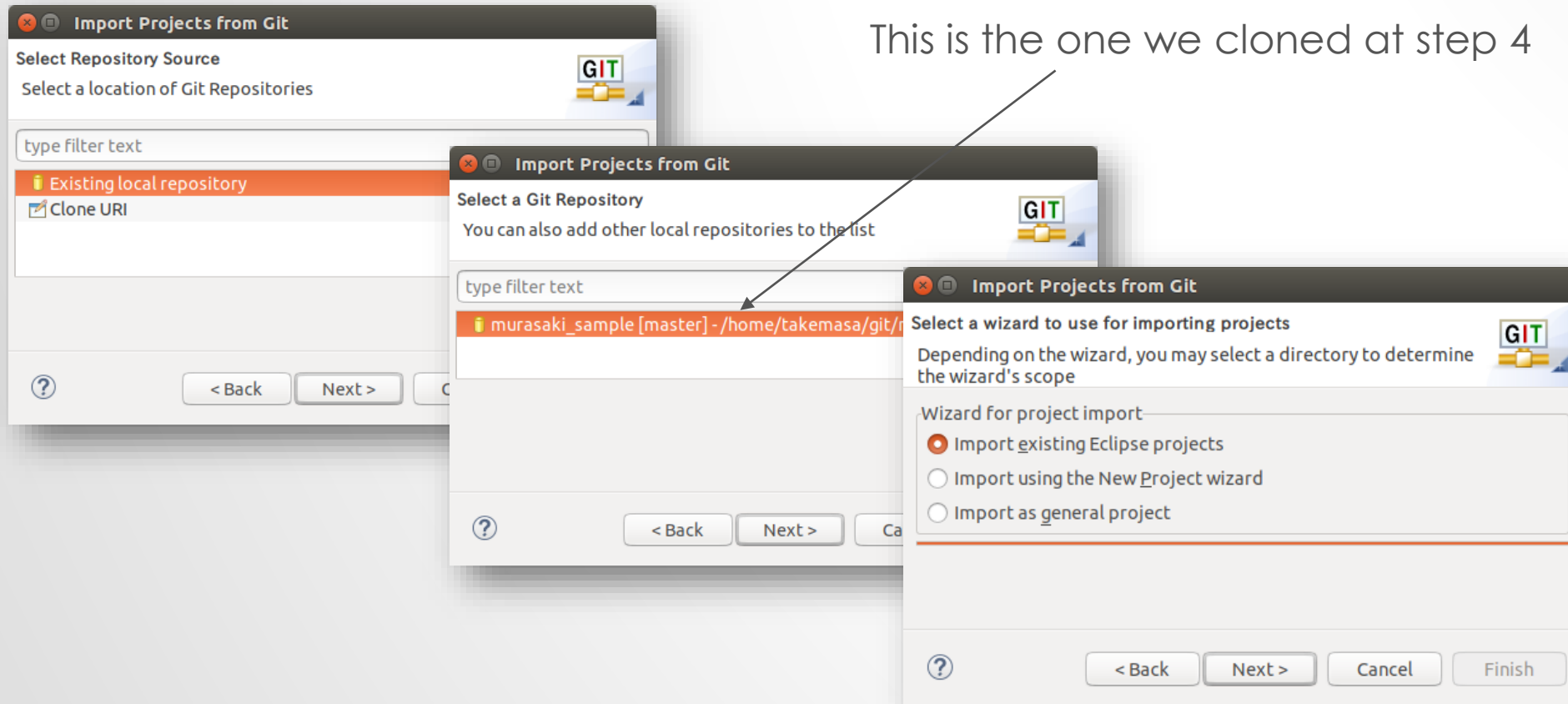


Go to C perspective

6.START TO IMPORT

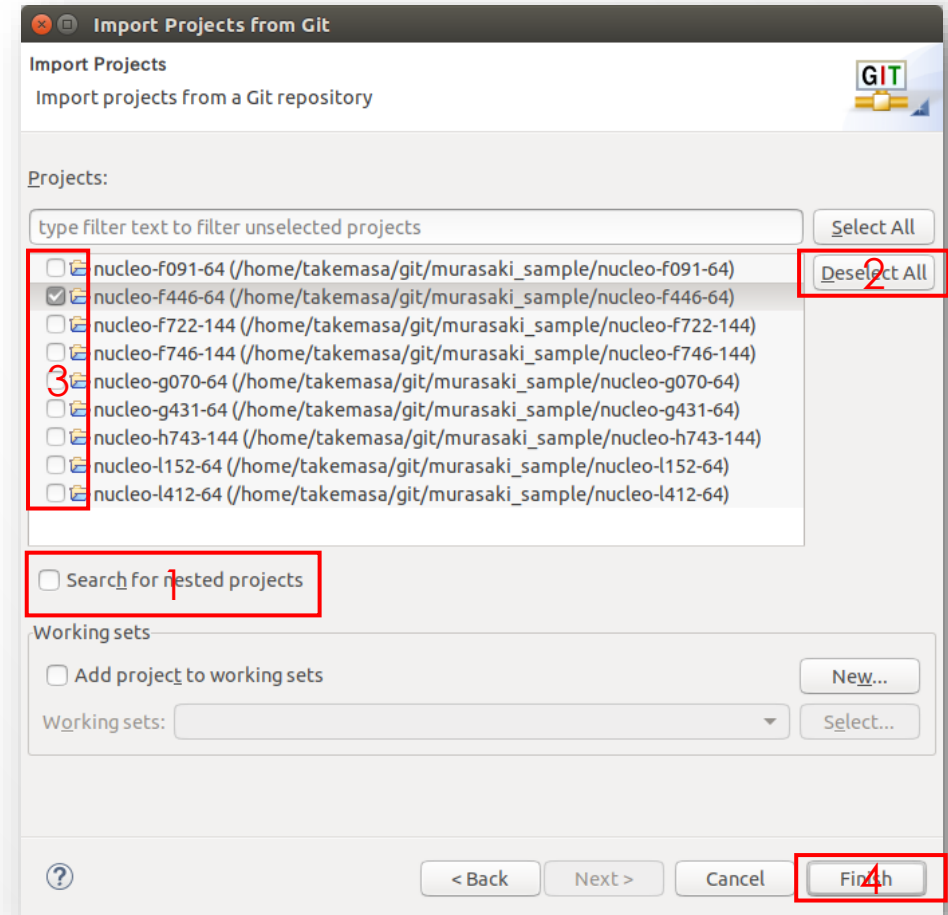


7. SPECIFY THE REPOSITORY



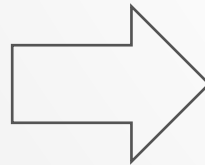
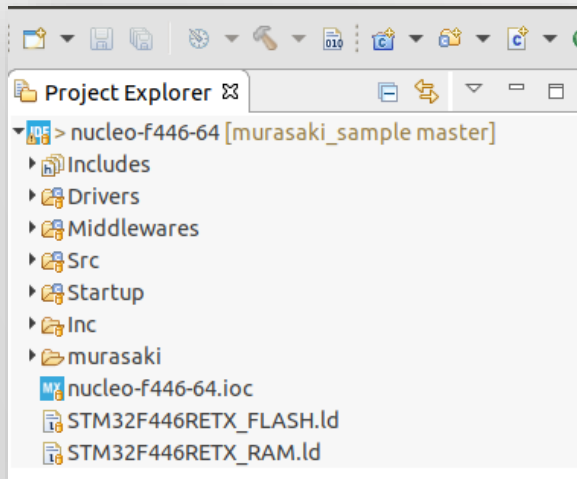
8. SPECIFY A PROJECT TO IMPORT

1. Uncheck "Search for nested project"
2. Click "Deselect All"
3. Check desired project.
4. Click "Finish"



9. TRIAL BUIDL

- Now, we have a sample project in the workspace
- Ctrl-B to build



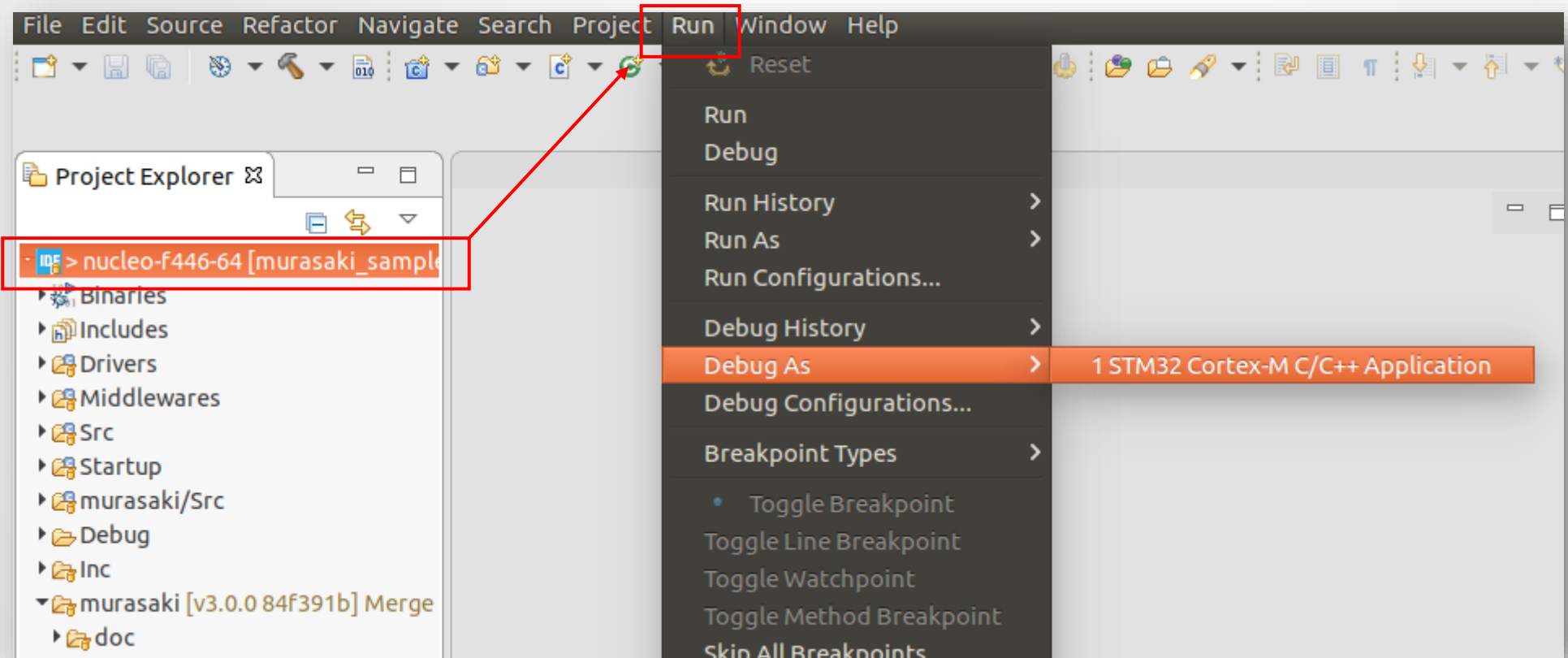
```
CDT Build Console [nucleo-f446-64]
arm-none-eabi-gcc ../Drivers/SIM32F4xx_HAL_Driver/Src/stm32f4xx_hal_pwr.c -mcpu=cortex-m4 -std
arm-none-eabi-gcc ../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_pwr_ex.c -mcpu=cortex-m4 -
arm-none-eabi-gcc ../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_rcc.c -mcpu=cortex-m4 -std
arm-none-eabi-gcc ../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_rcc_ex.c -mcpu=cortex-m4 -
arm-none-eabi-gcc ../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_tim.c -mcpu=cortex-m4 -std
arm-none-eabi-gcc ../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_tim_ex.c -mcpu=cortex-m4 -
arm-none-eabi-gcc ../Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_uart.c -mcpu=cortex-m4 -st
arm-none-eabi-g++ -o "nucleo-f446-64.elf" @"objects.list" -mcpu=cortex-m4 -T"/home/takemasa/gi
Finished building target: nucleo-f446-64.elf

arm-none-eabi-size nucleo-f446-64.elf
arm-none-eabi-objdump -h -S nucleo-f446-64.elf > "nucleo-f446-64.list"
arm-none-eabi-objcopy -O binary nucleo-f446-64.elf "nucleo-f446-64.bin"
  text  data  bss  dec  hex filename
69700  136  36376  106212  19ee4 nucleo-f446-64.elf
Finished building: default.size.stdout

Finished building: nucleo-f446-64.bin

Finished building: nucleo-f446-64.list
```

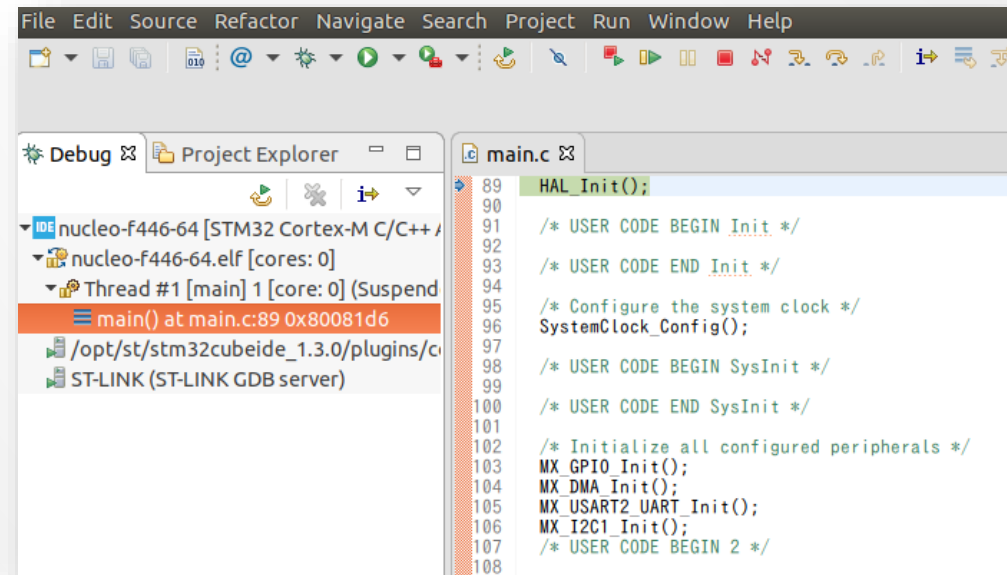
10. START THE DEBUGGER



Make sure the Nucleo is connected through USB

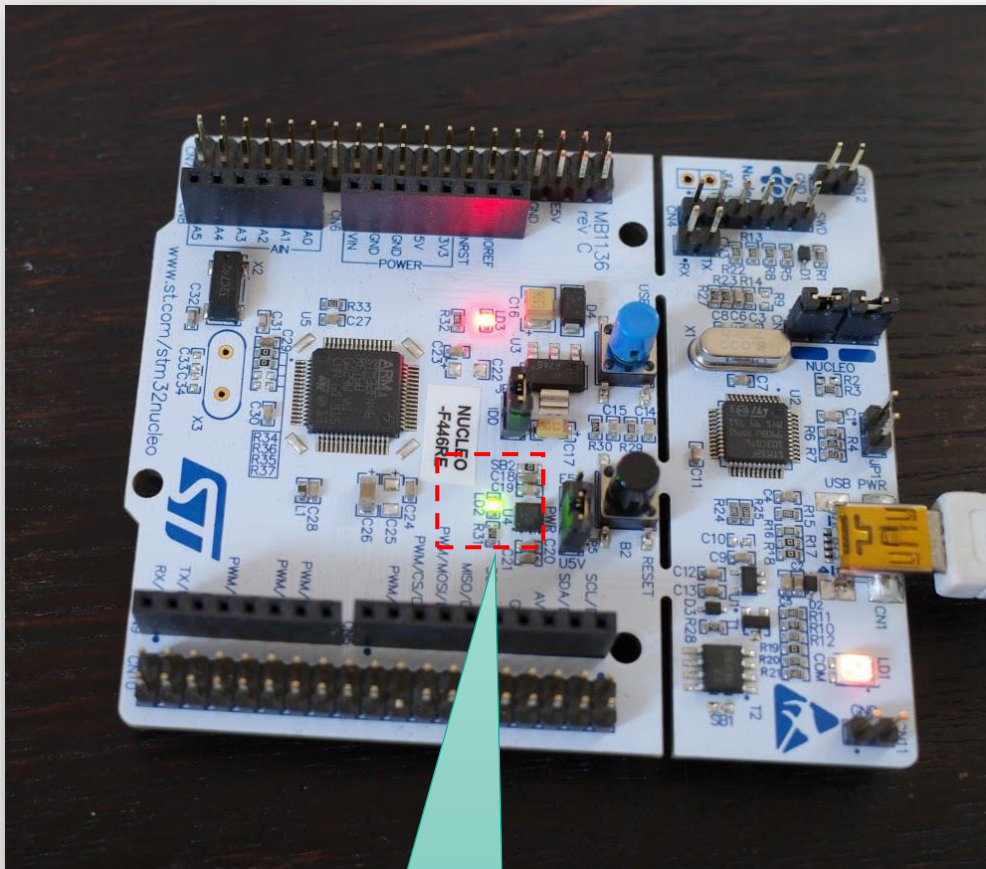
11.READY TO RUN

- Make sure a terminal emulator is waiting the serial communication



```
File Edit Source Refactor Navigate Search Project Run Window Help
nucleo-f446-64 [STM32 Cortex-M C/C++/
nucleo-f446-64.elf [cores: 0]
Thread #1 [main] 1 [core: 0] (Suspend
main() at main.c:89 0x80081d6
/opt/st/stm32cubeide_1.3.0/plugins/c
ST-LINK (ST-LINK GDB server)
89 HAL_Init();
90
91 /* USER CODE BEGIN Init */
92
93 /* USER CODE END Init */
94
95 /* Configure the system clock */
96 SystemClock_Config();
97
98 /* USER CODE BEGIN SysInit */
99
100 /* USER CODE END SysInit */
101
102 /* Initialize all configured peripherals */
103 MX_GPIO_Init();
104 MX_DMA_Init();
105 MX_USART2_UART_Init();
106 MX_I2C1_Init();
107 /* USER CODE BEGIN 2 */
108
```

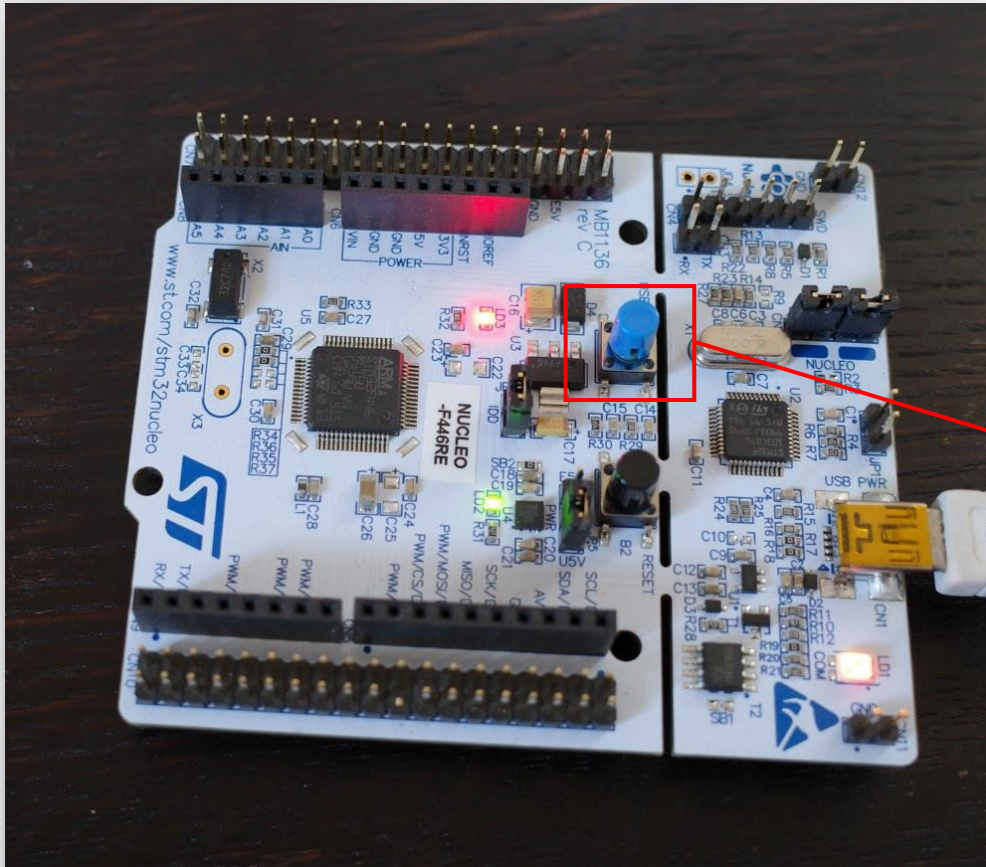
12.RUN THE PROGRAM



Blinking

```
takemasa@vm: ~/git/shared_foo/script
takemasa@vm:~/git$ cd shared_foo/
takemasa@vm:~/git/shared_foo$ ls
book-introducing-python  halide_study      kica
6
cpp-opencv               jupyter-notebook  mana
takemasa@vm:~/git/shared_foo$ cd script/
takemasa@vm:~/git/shared_foo/script$ ls
configure-github-global  configure-github-local
v2  terminal
takemasa@vm:~/git/shared_foo/script$ ./terminal
C-Kermit 9.0.302 OPEN SOURCE:, 20 Aug 2011, for
Copyright (C) 1985, 2011,
Trustees of Columbia University in the City of
Type ? or HELP for help.
(/home/takemasa/git/shared_foo/script/) C-Kermit
Connecting to /dev/stlink_console, speed 115200
Escape character: Ctrl-\ (ASCII 28, FS): enable
Type the escape character followed by C to get
or followed by ? to see other options.
-----
W!!! Push blue button to start the demo
```

13.PUSH THE BLUE BUTTON

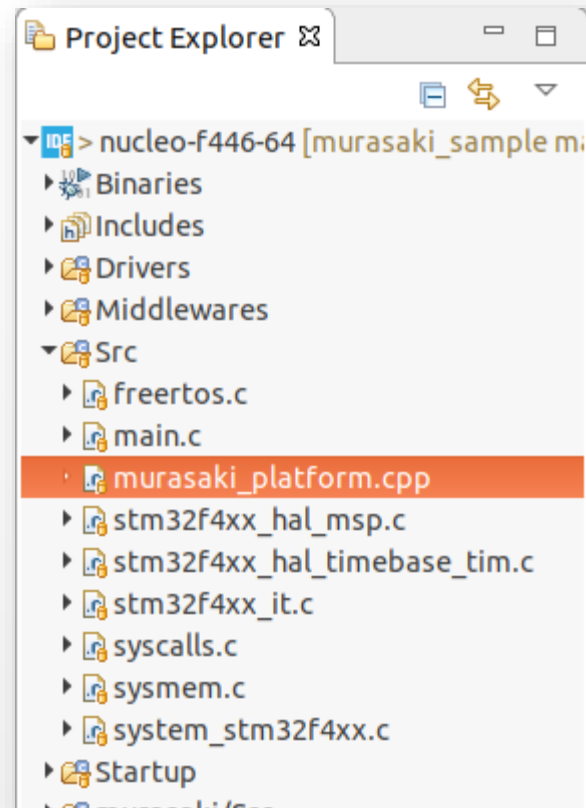


```
takemasa@vm: ~/git/shared_foo/script
(/home/takemasa/git/shared_foo/script/) C-Kermit>c
Connecting to /dev/stlink_console, speed 115200
Escape character: Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
-----
[0] W!!! Push blue button to start the demo

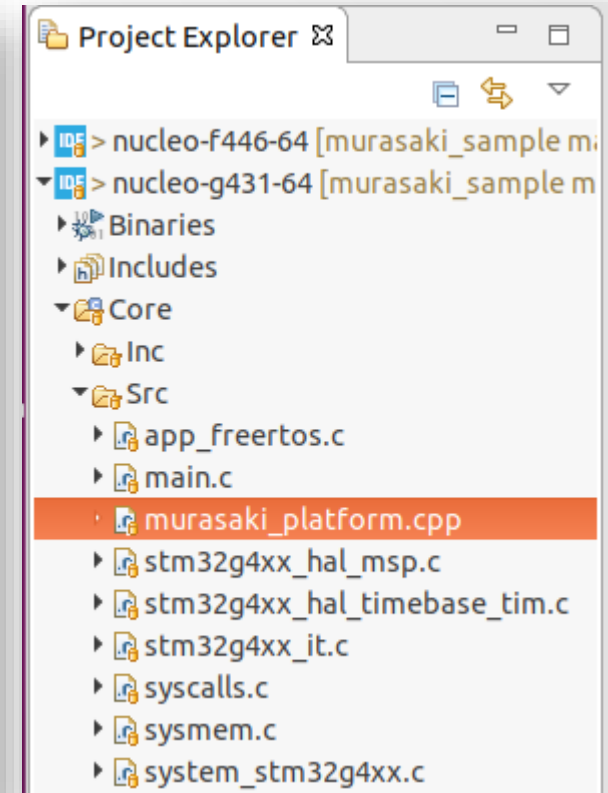
          Probing I2C devices
          | 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----+-----
0         |
10        |
20        |
30        |
40        |
50        |
60        |
70        |
Hello 0
Hello 1
Hello 2
```

TWO STRUCTURES OF PROJECT

- The CubeIDE introduce “Advanced Project structure”
- In the “Advanced” structure, the Src directory is under the Core directory.
- Let’s open the murasaki_platform.cpp file



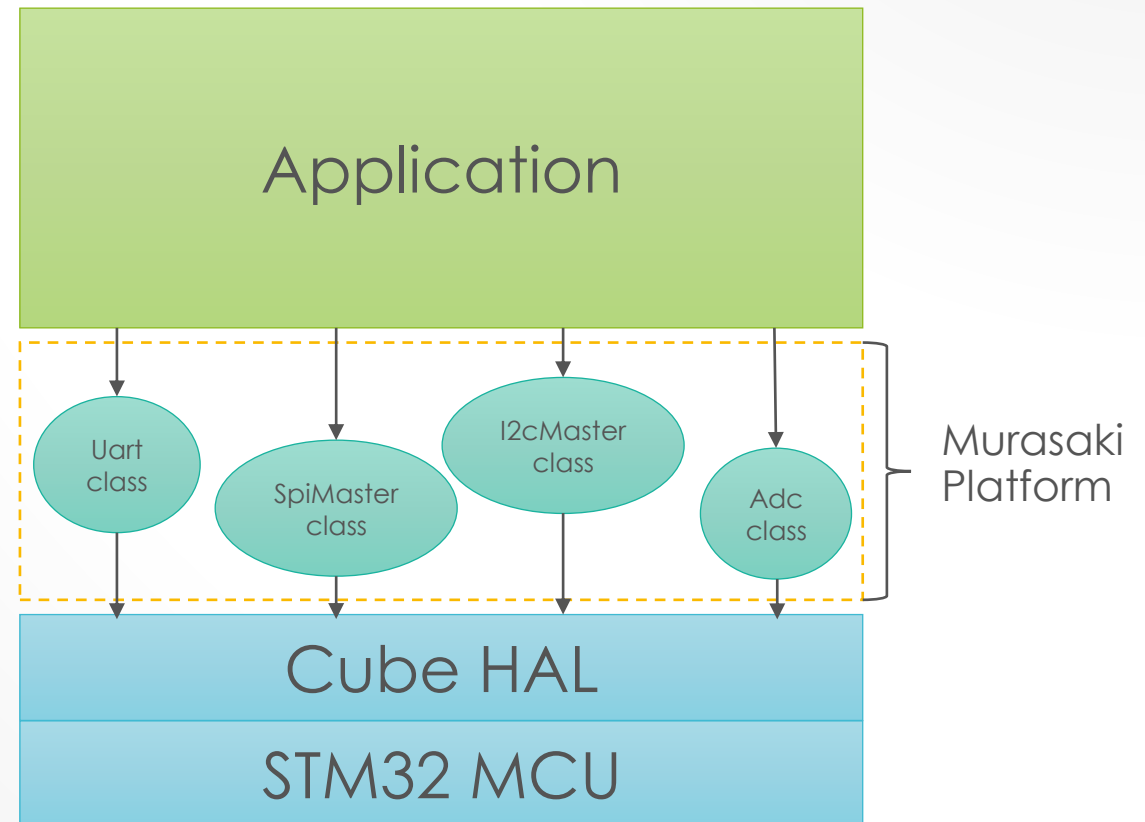
Old structure



Advanced structure

INSIDE MURASAKI_PLATFORM.CPP

- InitPlatform()
 - Initialization of Murasaki
 - Programmer must edit this function to initialize his/her platform
- ExecPlatform()
 - Execution body of application.



INSIDE EXECPLATFORM()

1. Start a new task
2. Print a message
3. Then, wait for the blue button
 1. This task halt here and wait for the interrupt from blue button

```
Type the escape character followed by C to get  
or followed by ? to see other options.
```

```
W!!! Push blue button to start the demo
```

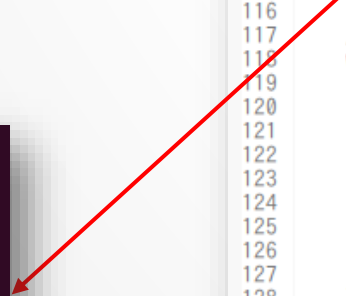
```
102 void ExecPlatform()  
103 {  
104     // counter for the demonstration.  
105     int count = 0;  
106  
107     // Start LED blink  
108     murasaki::platform.task1->Start();  
109  
110     // waiting for the Button push.  
111     murasaki::debugger->Printf("!!! Push blue button to start the demo %n");  
112     murasaki::platform.bl->Wait();  
113  
114     // List up connected I2C device to the console.  
115     I2cSearch(murasaki::platform.i2c_master);  
116  
117     // Loop forever  
118     while (true) {  
119  
120         // print a message with counter value to the console.  
121         murasaki::debugger->Printf("Hello %d %n", count);  
122  
123         // update the counter value.  
124         count++;  
125  
126         // wait for a while  
127         murasaki::Sleep(500);  
128     }  
129 }  
130
```

AFTER BUTTON PUSHED

- An utility function I2cSerch() is called.

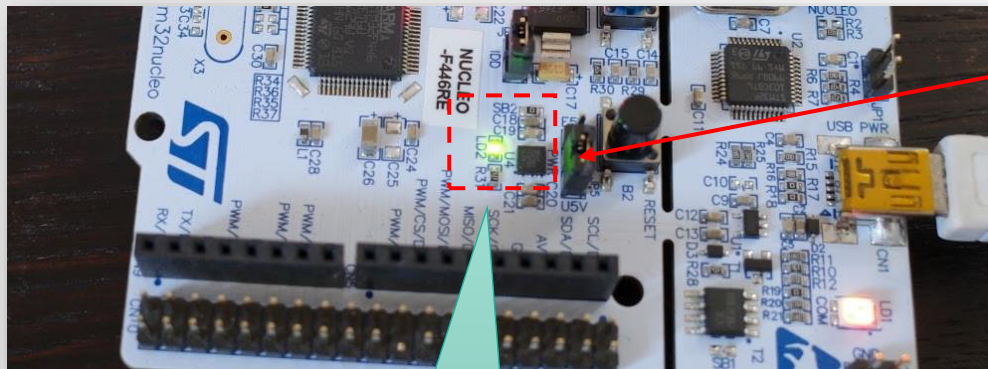
```
Push blue button to start the demo
Probing I2C devices
| 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
-----
0 | - - - - - - - - - - - - - - - - - - - - - -
10| - - - - - - - - - - - - - - - - - - - - - -
20| - - - - - - - - - - - - - - - - - - - - - -
30| - - - - - - - - - - - - - - - - - - - - - -
40| - - - - - - - - - - - - - - - - - - - - - -
50| - - - - - - - - - - - - - - - - - - - - - -
60| - - - - - - - - - - - - - - - - - - - - - -
70| - - - - - - - - - - - - - - - - - - - - - -
Hello 0
```

```
102 void ExecPlatform()
103 {
104     // counter for the demonstration.
105     int count = 0;
106
107     // Start LED blink
108     murasaki::platform.task1->Start();
109
110     // waiting for the Button push.
111     murasaki::debugger->Printf("!!! Push blue button to start the demo %n");
112     murasaki::platform.bl->Wait();
113
114     // List up connected I2C device to the console.
115     I2cSearch(murasaki::platform.i2c_master);
116
117     // Loop forever
118     while (true) {
119
120         // print a message with counter value to the console.
121         murasaki::debugger->Printf("Hello %d %n", count);
122
123         // update the counter value.
124         count++;
125
126         // wait for a while
127         murasaki::Sleep(500);
128     }
129 }
130
```



INSIDE STARTED TASK

- The started task just blinks LED.



```
140 void TaskBodyFunction(const void *ptr) {  
141     while (true) // dummy loop  
142     {  
143         murasaki::platform.led->Toggle(); // toggling LED  
144         murasaki::Sleep(700);  
145     }  
146 }  
147 }  
148 }
```

Blinking

POWER OF NAME SPACE

- Alt-/ shows the candidate of the keywords/identifiers.
- This makes programming easy
- Strict namespace of Murasaki narrowing down the candidate by minimum timing.

```
140 void TaskBodyFunction(const void *ptr) {
141
142     while (true) // dummy loop
143     {
144         mu
145         mu
146     }
147 }
148 # MURASAKI ASSERT(COND)
```



```
140 void TaskBodyFunction(const void *ptr) {
141
142     while (true) // dummy loop
143     {
144         murasaki::
145         murasaki::
146     }
147 }
148
```



```
140 void TaskBodyFunction(const void *ptr) {
141
142     while (true) // dummy loop
143     {
144         murasaki::platform.
145         murasaki::Sleep(700)
146     }
147 }
148
```



```
140 void TaskBodyFunction(const void *ptr) {
141
142     while (true) // dummy loop
143     {
144         murasaki::platform.led->
145         murasaki::Sleep(700);
146     }
147 }
148
149
150
151
152
153
```

DEBUG : PRINTF

Let's add Murasaki::debugger->Print() and run

```
102 void ExecPlatform()
103 {
104     // counter for the demonstration.
105     int count = 0;
106
107     // Start LED blink
108     murasaki::platform.task1->Start();
109
110     // waiting for the Button push.
111     murasaki::debugger->Printf("!!! Push blue button to start the demo");
112     murasaki::platform.b1->Wait();
113
114     murasaki::debugger->Printf("Hello, %d %n", 2020);
115
116     // List up connected I2C device to the console.
117     I2cSearch(murasaki::platform.i2c_master);
118
119     // Loop forever
120     while (true) {
121
122         // print a message with counter value to the console.
123         murasaki::debugger->Printf("Hello %d %n", count);
124
125         // update the counter value
```

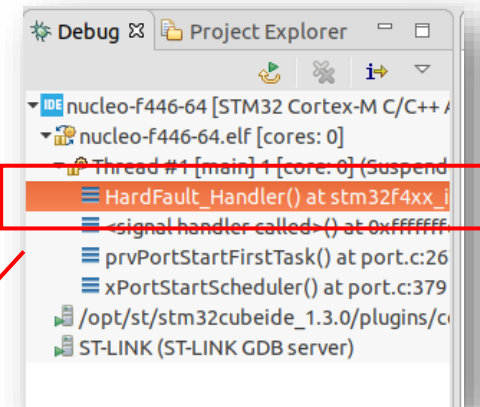
```
Escape character: Ctrl-\ (ASCII 28, FS): enable
Type the escape character followed by C to get
or followed by ? to see other options.
-----
!!! Push blue button to start the demo
Hello, 2020
█
```

DEBUG : ASSERTION

Let's add Murasaki::debugger->Print() and run

```
102 void ExecPlatform()
103 {
104     // counter for the demonstration.
105     int count = 0;
106
107     // Start LED blink
108     murasaki::platform.task1->Start();
109
110     // waiting for the Button push.
111     murasaki::debugger->Printf("!!! Push blue button to start the demo %n");
112     murasaki::platform.b1->Wait();
113
114     MURASAKI_ASSERT(1 == 2);
115
116     // List up connected I2C device to the console.
117     I2cSearch(murasaki::platform.i2c_master);
118 }
```

Push button



Resume again

```
takemasa@vm: ~/git/shared_foo/script
!!! Push blue button to start the demo
-----
!! Assertion failure in function ExecPlatform(), at line 114 of file murasaki_platform.cpp !!
Fail expression : 1 == 2

Spurious exception or hardfault occured.
Stacked R0 : 0x00000000
Stacked R1 : 0xFFFFFFFF
Stacked R2 : 0x00000000
```

DEBUG : SYSLOG

Let's add Murasaki::SetSyslogFacilityMask() and SetSyslogSeverityThreshold()

```
102 void ExecPlatform()  
103 {  
104     // counter for the demonstration.  
105     int count = 0;  
106  
107     // Start LED blink  
108     murasaki::platform.task1->Start();  
109  
110     murasaki::SetSyslogFacilityMask(murasaki::kfaExti);  
111     murasaki::SetSyslogSererityThreshold(murasaki::kseDebug);  
112  
113     // waiting for the Button push.  
114     murasaki::debugger->Printf("!!! Push blue button to start the dem  
115     murasaki::platform.bl->Wait();  
116  
117     // List up connected I2C device to the console.  
118     I2cSearch(murasaki::platform.i2c master);
```

```
takemasa@vm: ~/git/shared_foo/script  
  
!!! Push blue button to start the demo  
66164, 0x200023e8, kfaExti, kseDebug: exti.cpp, line 199, Wait(): Enter  
557825001, 0, kfaExti, kseDebug: exti.cpp, line 256, isReady(): Enter  
557837878, 0, kfaExti, kseDebug: exti.cpp, line 264, isReady(): Exit with true  
557851576, 0x200023e8, kfaExti, kseDebug: exti.cpp, line 238, Match(): Enter  
557864315, 0x200023e8, kfaExti, kseDebug: exti.cpp, line 241, Match(): Matched. Exit with true  
557878601, 0x200023e8, kfaExti, kseDebug: exti.cpp, line 218, Release(): Enter  
557891519, 0x200023e8, kfaExti, kseDebug: exti.cpp, line 238, Match(): Enter  
557904255, 0x200023e8, kfaExti, kseDebug: exti.cpp, line 241, Match(): Matched. Exit with true  
557918800, 0x200023e8, kfaExti, kseDebug: exti.cpp, line 221, Release(): Matched and release  
557933484, 0x200023e8, kfaExti, kseDebug: exti.cpp, line 225, Release(): Exit with true.  
557951601, 0x200023e8, kfaExti, kseDebug: exti.cpp, line 211, Wait(): Exit with 0
```


DEBUG : TASK STACK HEADROOM

Let's add member functions of TaskStrategy class

```
102 void ExecPlatform()
103 {
104     // counter for the demonstration.
105     int count = 0;
106
107     // Start LED blink
108     murasaki::platform.task1->Start();
109
110     // waiting for the Button push.
111     murasaki::debugger->Printf("!!! Push blue button to start the demo %n");
112     murasaki::platform.b1->Wait();
113
114     int depth = murasaki::platform.task1->getStackDepth();
115     int head_room = murasaki::platform.task1->getStackMinHeadroom();
116     murasaki::debugger->Printf("Stack headroom is %d/%d %n",
117                               head_room,
118                               depth);
119 }
```

```
takemasa@vm: ~/git/shared_foo/script
!!! Push blue button to start the demo
Stack headroom is -1/256
```

-1 ?

WE NEED ADDITIONAL SETTINGS

Pinout & Configuration | Clock Configuration | Project Manager

Additional Software | Pinout

FREERTOS Mode and Configuration

Configuration

Reset Configuration

| | |
|---|---|
| <input checked="" type="checkbox"/> Mutexes | <input checked="" type="checkbox"/> FreeRTOS Heap Usage |
| <input checked="" type="checkbox"/> Tasks and Queues | <input checked="" type="checkbox"/> Timers and Semaphores |
| <input checked="" type="checkbox"/> Advanced settings | <input checked="" type="checkbox"/> User Constants |
| <input checked="" type="checkbox"/> Config parameters | <input checked="" type="checkbox"/> Include parameters |

Configure the below parameters:

Search (Ctrl+F)

| | |
|---|----------|
| USE_MALLOC_FAILED_HOOK | Disabled |
| USE_DAEMON_TASK_STARTU... | Disabled |
| CHECK_FOR_STACK_OVERFLOW | Option1 |
| Run time and task stats gathering rela... | Disabled |
| GENERATE_RUN_TIME_STATS | Option1 |
| USE_TRACE_FACILITY | Option2 |
| USE_STATS_FORMATTING_FU... | Disabled |
| Co-routine related definitions | |
| USE_CO_ROUTINES | Disabled |
| MAX_CO_ROUTINE_PRIORITIES | ? |

FATFS
 FREERTOS
LIBJPEG
MBEDTLS

FREERTOS Mode and Configuration

Configuration

Reset Configuration

| | |
|---|---|
| <input checked="" type="checkbox"/> Mutexes | <input checked="" type="checkbox"/> FreeRTOS Heap Usage |
| <input checked="" type="checkbox"/> Tasks and Queues | <input checked="" type="checkbox"/> Timers and Semaphores |
| <input checked="" type="checkbox"/> Advanced settings | <input checked="" type="checkbox"/> User Constants |
| <input checked="" type="checkbox"/> Config parameters | <input checked="" type="checkbox"/> Include parameters |

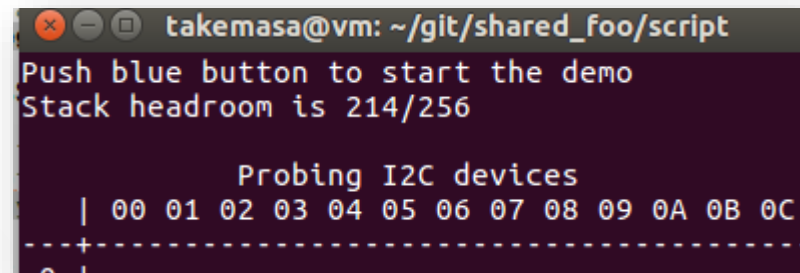
Configure the below parameters:

Search (Ctrl+F)

| | |
|-------------------------|----------|
| vTaskDelay | Enabled |
| xTaskGetSchedulerState | Enabled |
| xTaskResumeFromISR | Enabled |
| xQueueGetMutexHolder | Disabled |
| xSemaphoreGetMutex... | Disabled |
| pcTaskGetTaskName | Disabled |
| uxTaskGetStackHighW... | Disabled |
| xTaskGetCurrentTaskH... | Disabled |
| eTaskGetState | Enabled |
| xEventGroupSetBitFro... | Disabled |
| xTimerPendFunctionCall | Disabled |

RESULT OF TASK HEADROOM

- We can see the “rest of stack” for each task.
- The unit is byte.
- Check FreeRTOS Configuration for details :
 - configCHECK_FOR_STACK_OVERFLOW



```
takemasa@vm: ~/git/shared_foo/script
Push blue button to start the demo
Stack headroom is 214/256

      Probing I2C devices
| 00 01 02 03 04 05 06 07 08 09 0A 0B 0C
-----
0 1
```

DEBUG : HARD FAULT

```
102 void ExecPlatform()  
103 {  
104     // counter for the demonstration.  
105     int count = 0;  
106  
107     // Start LED blink  
108     murasaki::platform.task1->Start();  
109  
110     // waiting for the Button push.  
111     murasaki::debugger->Printf("!!! Push blue button to sta  
112     murasaki::platform.b1->Wait();  
113  
114     typedef void (*FUNCTYPE)(void);  
115  
116     FUNCTYPE funcPtr = reinterpret_cast<FUNCTYPE>(0);  
117     funcPtr();  
118  
119     // List up connected I2C device to the console.  
120     I2cSearch(murasaki::platform.i2c_master);  
121  
122     // Loop forever
```

```
takemasa@vm: ~/git/shared_foo/script  
!!! Push blue button to start the demo  
  
Spurious exception or hardfault occured.  
Stacked R0 : 0x00000000  
Stacked R1 : 0x00000000  
Stacked R2 : 0x00000007  
Stacked R3 : 0x00000000  
Stacked R12 : 0xA5A5A5A5  
Stacked LR : 0x080087F7  
Stacked PC : 0x00000000  
Stacked PSR : 0x40000000  
CFSR : 0x00020000  
HFSR : 0x40000000  
DFSR : 0x00000000  
AFSR : 0x00000000  
MMAR : 0xE000ED34  
BFAR : 0xE000ED38  
  
(Note : To avoid the stacking by C compiler, use rele  
!!! Push blue button to start the demo
```



DOCUMENTATION

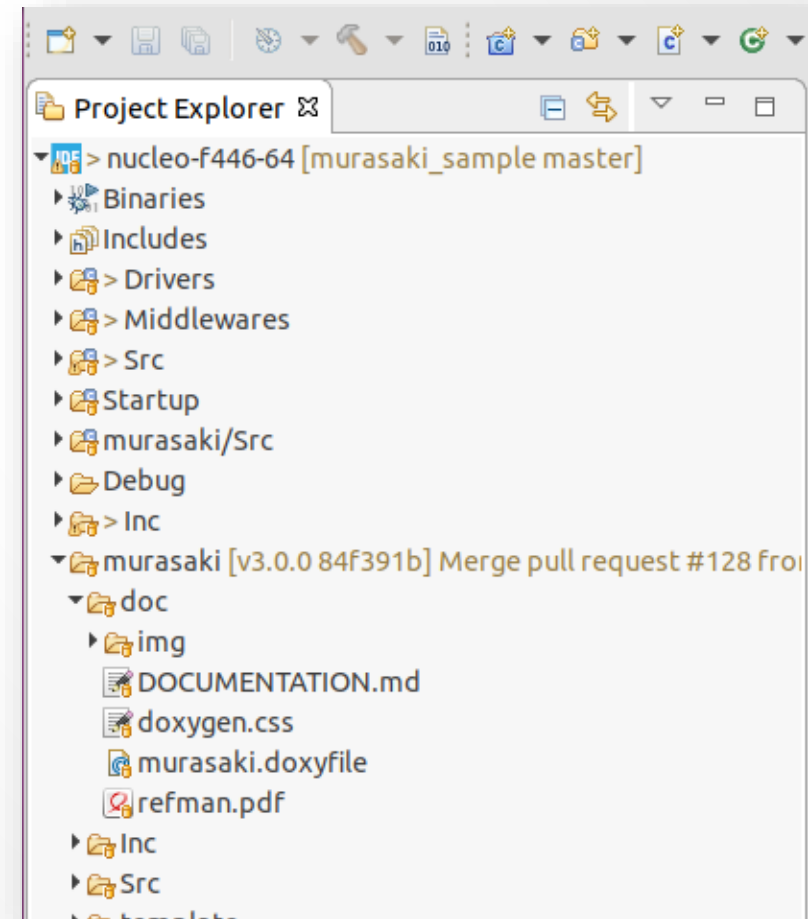
Photo by Debby Hudson on Unsplash

IN THIS SECTION

- Location of PDF document
- Making Doxygen document
- Tour of Doxygen document
- WIKI

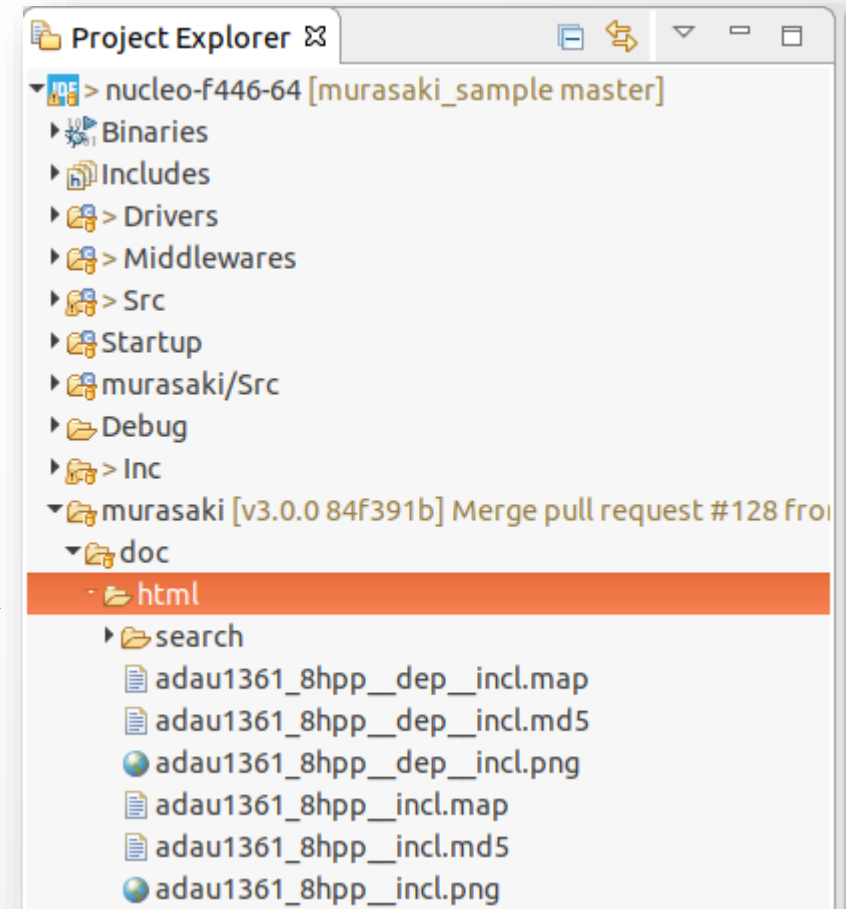
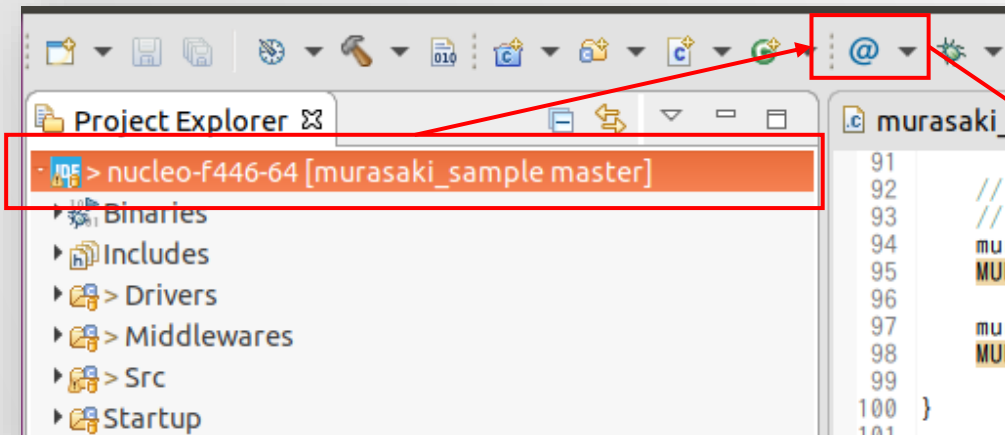
LOCATION OF PDF DOCUMENT

- murasaki/doc
- PDF document is same contents with HTML document
- Just a matter of format



MAKING DOXYGEN DOCUMENT

Install Eclox, to create the document



HTML DOCUMENTATION

- Classes and Functions are grouped by “Module”
- “Murasaki Class Collection” module is a list of the peripheral classes.
 - Usually, application programmer uses classes in this module.

Murasaki Class Library 3.0.0
A peripheral controll class collections for the STM32 microprocessor fa

[Main Page](#) [Related Pages](#) [Modules](#) [Namespaces](#) [Classes](#)

Modules

Here is a list of all modules:

| | |
|--------------------------------------|------------------------|
| ▼ Murasaki API reference | Murasaki API referen |
| Murasaki Class Collection | STM32 Class library |
| Third party classes | Classes for the thirdp |
| Definitions and Configuration | Definitions and confi |
| Application Specific Platform | Variables to control t |
| Abstract Classes | Generic classes as te |
| Synchronization and Exclusive access | Sync between the tas |
| Helper classes | Classes to support th |
| Utility functions | Collection of the use |
| ▼ CMSIS | |

MURASAKI CLASS COLLECTION

- Peripheral like UART, SPI, I2C, ADC, GPIO, SAI, I2S are controlled by these classes.
- Algorithm class like SimpleTask or DuplexAudio are also listed here.

Murasaki Class Library 3.0.0
A peripheral control class collections for the STM32 microprocessor family

Main Page | Related Pages | **Modules** | Namespaces | Classes | Files

Murasaki Class Collection

[Murasaki API reference](#)

STM32 Class library. [More...](#)

Collaboration diagram for Murasaki Class Collection:

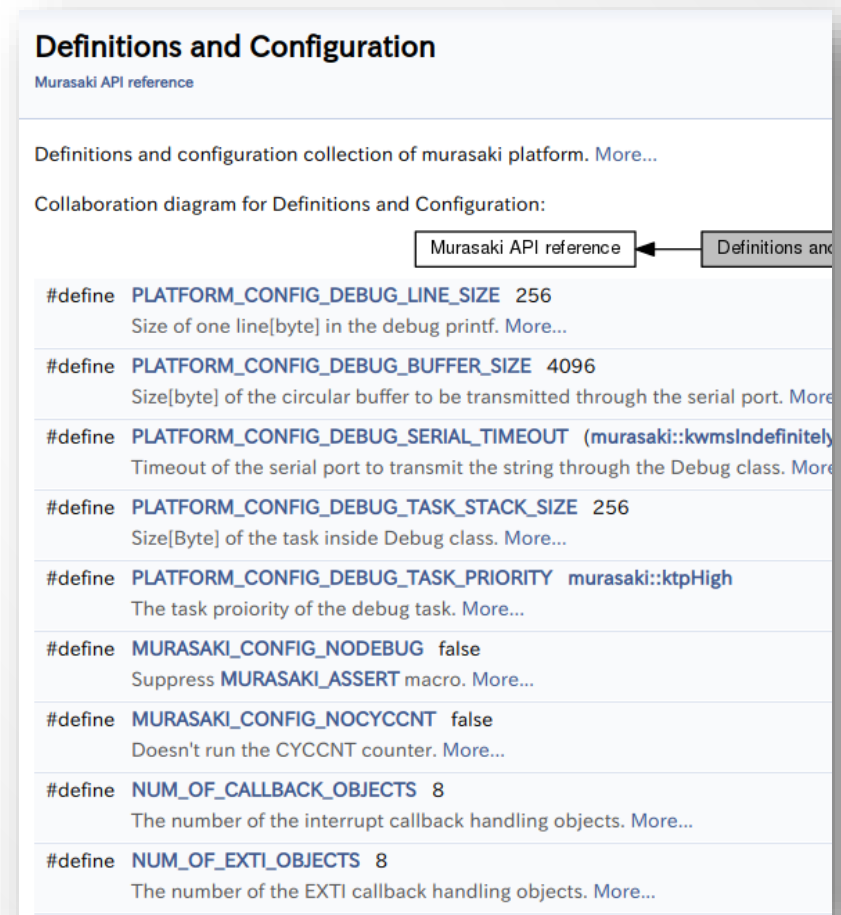
```
graph LR; MurasakiClass[Murasaki Class] --> MurasakiAPI[Murasaki API reference];
```

Classes

- class **murasaki::Adc**
STM32 dedicated ADC class. [More...](#)
- class **murasaki::BitIn**
General purpose bit input. [More...](#)
- class **murasaki::BitOut**
General purpose bit output. [More...](#)
- class **murasaki::Debugger**
Debug class. Provides printf() style output for both task and ISR context. [More...](#)
- class **murasaki::DuplexAudio**
Stereo Audio is served by this class. [More...](#)
- class **murasaki::Exti**
EXTI wrapper class. [More...](#)
- class **murasaki::I2cMaster**
Thread safe, synchronous, and blocking IO. Encapsulating I2C master. Based on ST

DEFINITIONS AND CONFIGURATION

- These macros configures Murasaki
 - Buffer size
 - Use of Debug
- To change the configuration, define these macros in the platform_config.hpp



The screenshot shows a web page titled "Definitions and Configuration" with a subtitle "Murasaki API reference". Below the title, there is a description: "Definitions and configuration collection of murasaki platform. More...". A collaboration diagram shows a box labeled "Murasaki API reference" with an arrow pointing to a box labeled "Definitions and Configuration". The main content is a list of macro definitions:

- #define PLATFORM_CONFIG_DEBUG_LINE_SIZE 256**
Size of one line[byte] in the debug printf. More...
- #define PLATFORM_CONFIG_DEBUG_BUFFER_SIZE 4096**
Size[byte] of the circular buffer to be transmitted through the serial port. More...
- #define PLATFORM_CONFIG_DEBUG_SERIAL_TIMEOUT (murasaki::kwmsIndefinitely)**
Timeout of the serial port to transmit the string through the Debug class. More...
- #define PLATFORM_CONFIG_DEBUG_TASK_STACK_SIZE 256**
Size[Byte] of the task inside Debug class. More...
- #define PLATFORM_CONFIG_DEBUG_TASK_PRIORITY murasaki::ktpHigh**
The task proiority of the debug task. More...
- #define MURASAKI_CONFIG_NODEBUG false**
Suppress **MURASAKI_ASSERT** macro. More...
- #define MURASAKI_CONFIG_NOCYCCNT false**
Doesn't run the CYCCNT counter. More...
- #define NUM_OF_CALLBACK_OBJECTS 8**
The number of the interrupt callback handling objects. More...
- #define NUM_OF_EXTI_OBJECTS 8**
The number of the EXTI callback handling objects. More...

SYNCHRONIZATION AND EXCLUSIVE ACCESS

- Inter-task synchronization
- Interrupt vs. task Synchronization
- Inter-task exclusive accesses.

Synchronization and Exclusive access

[Murasaki API reference](#)

Sync between the task and interrupt. Make the resources thread safe. [More...](#)

Collaboration diagram for Synchronization and Exclusive access:



Murasaki API reference

Classes

class [murasaki::CriticalSection](#)

A critical section for task context. [More...](#)

class [murasaki::Synchronizer](#)

Synchronization class between a task and interrupt. This class provides

UTILITY FUNCTIONS

- Cycle counter control
- I2C device search
- Other functionality may added

Utility functions

[Murasaki API reference](#)

Collection of the useful functions. [More...](#)

Collaboration diagram for Utility functions:

[Murasaki API reference](#) ←

Functions

void [murasaki::I2cSearch \(murasaki::I2cMasterStrategy *master\)](#)
I2C device serach function. [More...](#)

void [murasaki::InitCycleCounter \(\)](#)
Initialize and start the cycle counter. [More...](#)

unsigned int [murasaki::GetCycleCounter \(\)](#)
Obtain the current cycle count of CYCCNT register. [More...](#)

USAGE GUIDES

- Beside of APIs, Murasaki has different aspect of documents.
 - Usage Introduction
 - Program flow explanation
 - Porting Guide

| Main Page | Related Pages | Module |
|--|---------------|--------|
| <h2>Related Pages</h2> | | |
| Here is a list of all related documentation pages. | | |
| ▼ Preface | | |
| Simplified IO | | |
| Preemptive multi-task | | |
| Synchronous IO | | |
| Thread-safe IO | | |
| Versatile printf() logger | | |
| Guard by assertion | | |
| System Logging | | |
| Configurable | | |
| Target and Environment | | |
| ▼ Usage Introduction | | |
| Message output | | |
| Serial communication | | |
| Debugging with Murasaki. | | |
| Tasking | | |
| ▼ Other peripherals | | |
| I2C Master | | |
| I2C Slave | | |
| SPI Master | | |
| SPI Slave | | |
| GPIO | | |
| Duplex Audio | | |
| ▼ Program flow | | |

| |
|-------------------------------------|
| ▼ Program flow |
| Application flow |
| HAL Assertion flow |
| Spurious Interrupt flow |
| Assertion flow |
| General Interrupt flow |
| EXTI flow |
| ▼ Porting guide |
| Directory Structure |
| ▼ CubeIDE setting |
| Heap Size |
| Stack Size |
| Task stack size of the default task |
| UART peripheral |
| SPI Master peripheral |
| SPI Slave peripheral |
| I2C peripheral |
| EXTI |
| Configuration |
| Task Priority and Stack Size |
| Heap memory consideration |
| Platform variable |
| Routing interrupts |
| Error handling |
| Summary of the porting |

CONFIGURATION OF PERIPHERAL

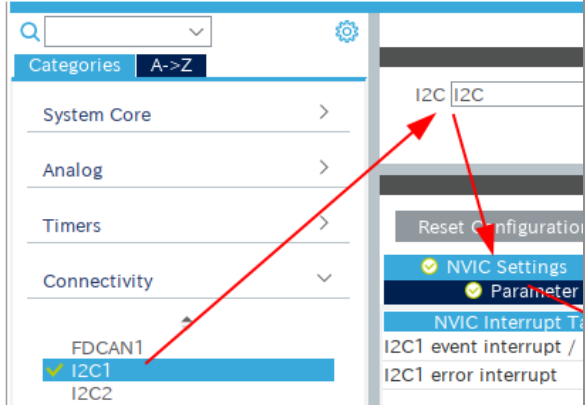
- Also, each peripheral class describes :
 - How to configure the device
 - Interrupt handling
 - IO operations

Detailed Description

The `I2cMaster` class is the wrapper of the I2C controller.

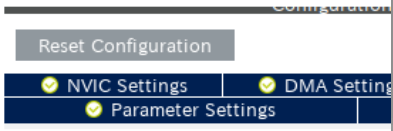
Configuration

To configure the I2C peripheral as master, chose I2C peripheral in the Device Configuration Manager and enable its NVIC interrupt.



The screenshot shows the STM32CubeMX configuration tool. On the left, the 'Connectivity' category is expanded, and 'I2C1' is selected. On the right, the 'NVIC Settings' and 'Parameter Settings' are visible. Red arrows point from the 'I2C1' selection to the 'NVIC Settings' and 'Parameter Settings' sections.

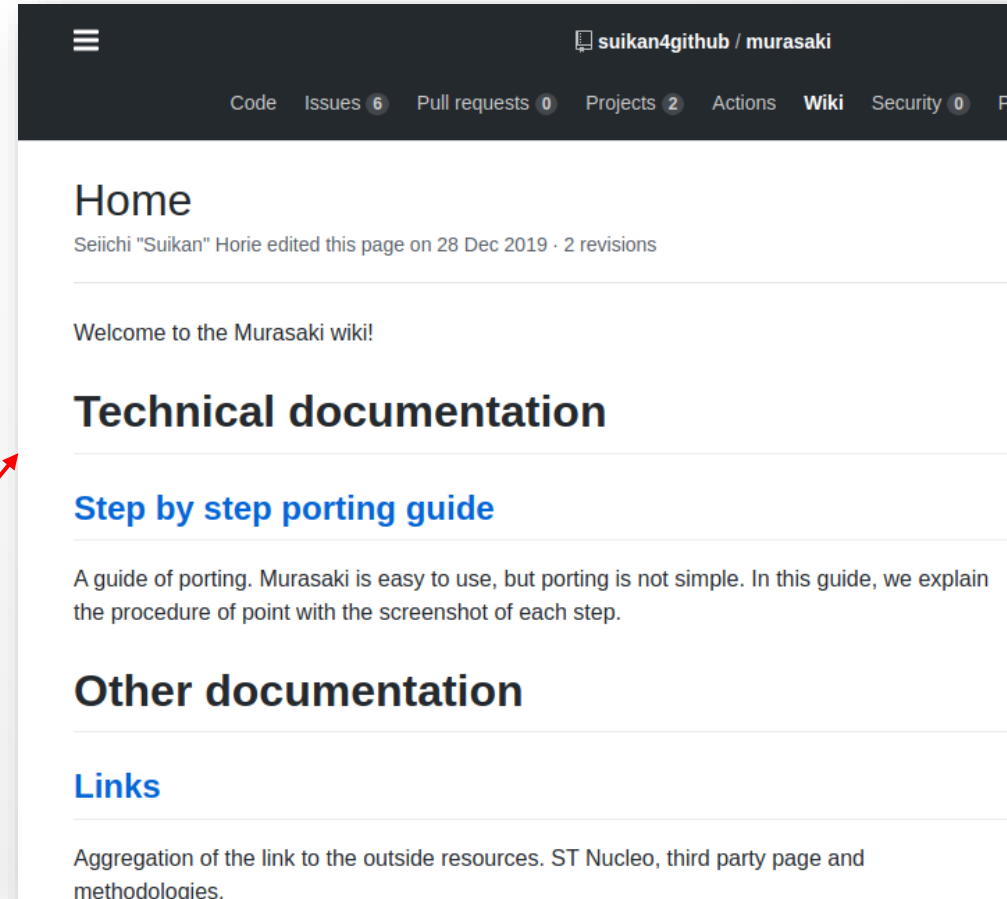
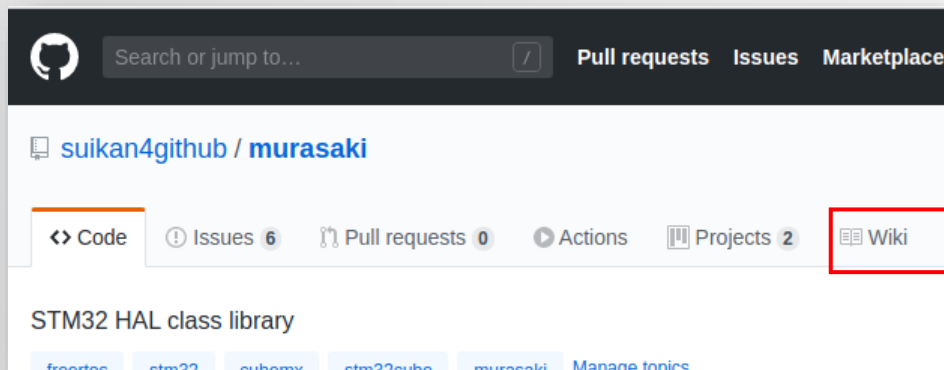
Also, pay attention to the I2C Maximum Output Speed. The default setting by the manufacturer is 100 kHz. You can verify this with an oscilloscope.



The screenshot shows the STM32CubeMX configuration tool. On the right, the 'NVIC Settings' and 'Parameter Settings' are visible. A 'Reset Configuration' button is also present.

WIKI

- Murasaki project has its own wiki.
- Supplemental information will be placed here





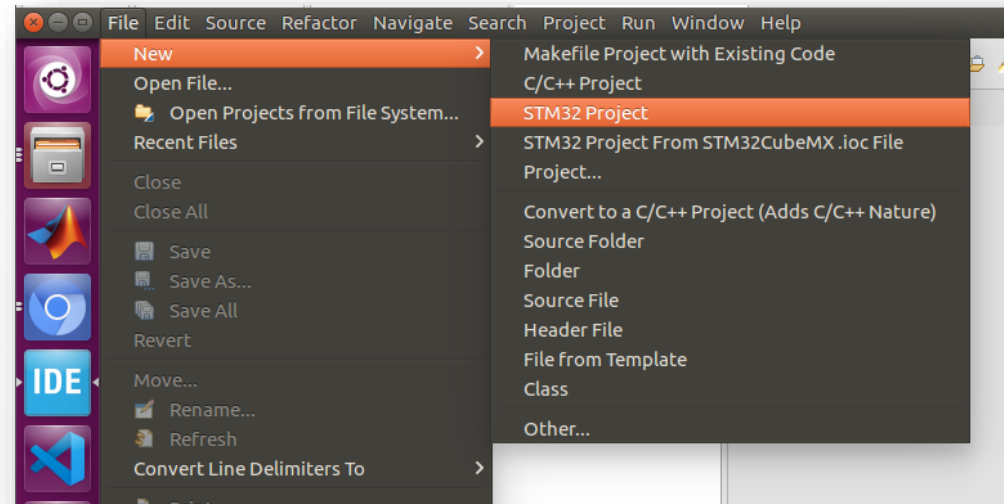
HOW TO CREATE YOUR OWN APPLICATION

IN THIS SECTION

- Create a new project to your Nucleo
- Configure the device by CubeIDE Device Configuration Tool
- Clone Murasaki into the project
- Install Murasaki
- Set up the project to use Murasaki
- Build
- Adjust to the target Nucleo

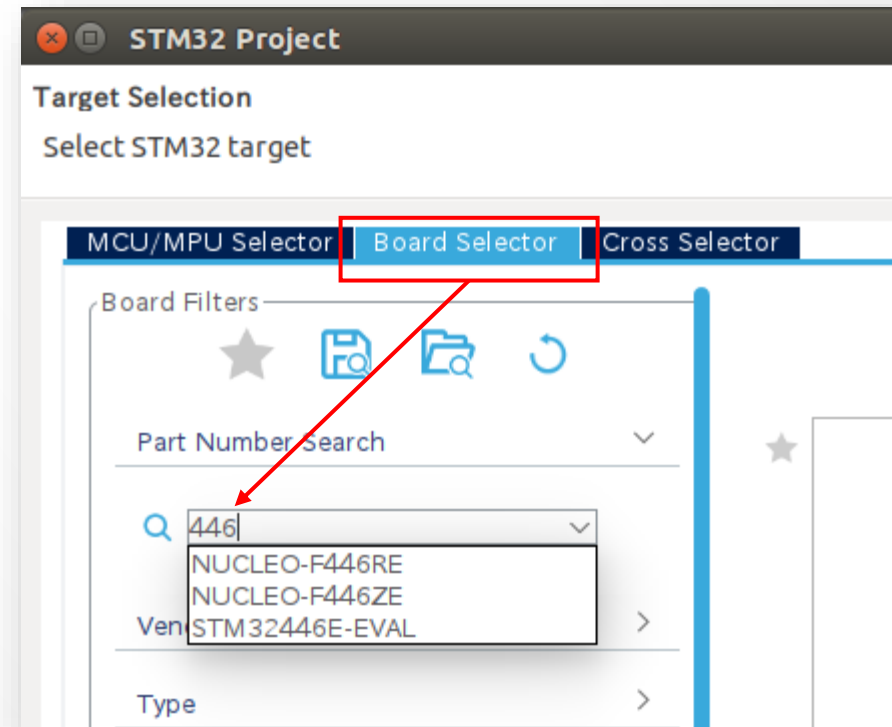
CREATE AN C++ PROJECT FOR YOUR NUCLEO

- First of all, create a new STM32 Project into your work space.
- File -> New -> STM32 Project



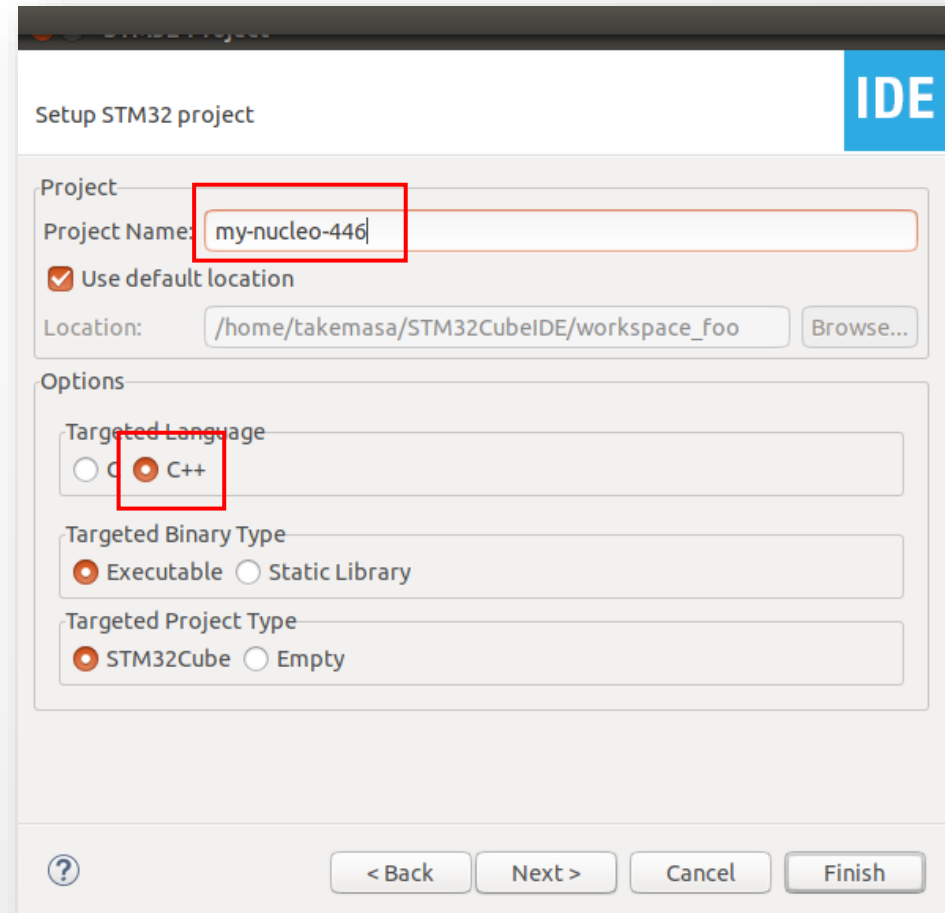
SPECIFY THE CORRECT NUCLEO NAME

- Follow the procedure :
 1. Select board selector
 2. Type the name of the Nucleo to the Number search
 3. Select correct Nucleo board
 4. Then, click "Next"



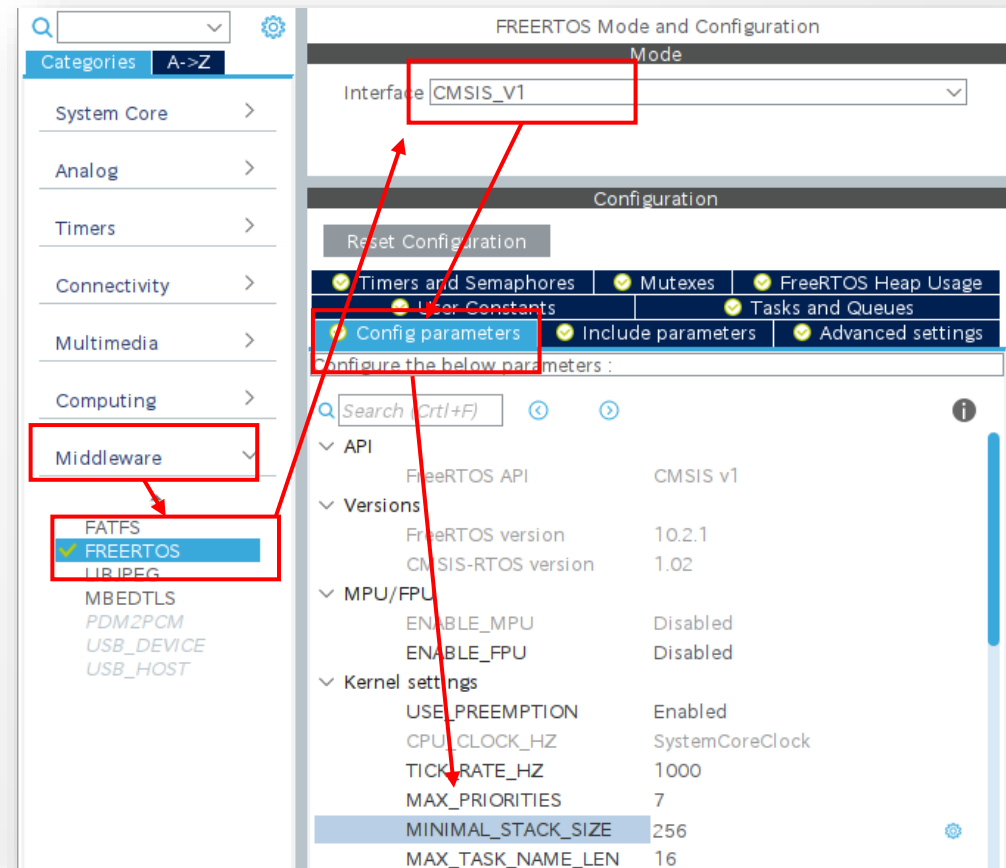
SPECIFY THE PROJECT NAME

- Follow the procedure :
 1. Specify the name in to "Project Name"
 2. Make sure to select the "C++" as the Target Language
 3. Then click "Finish"
 4. Regarding the default pin state and Eclipse perspective, click OK for a while
 5. Now, the Device Configuration Tool appears



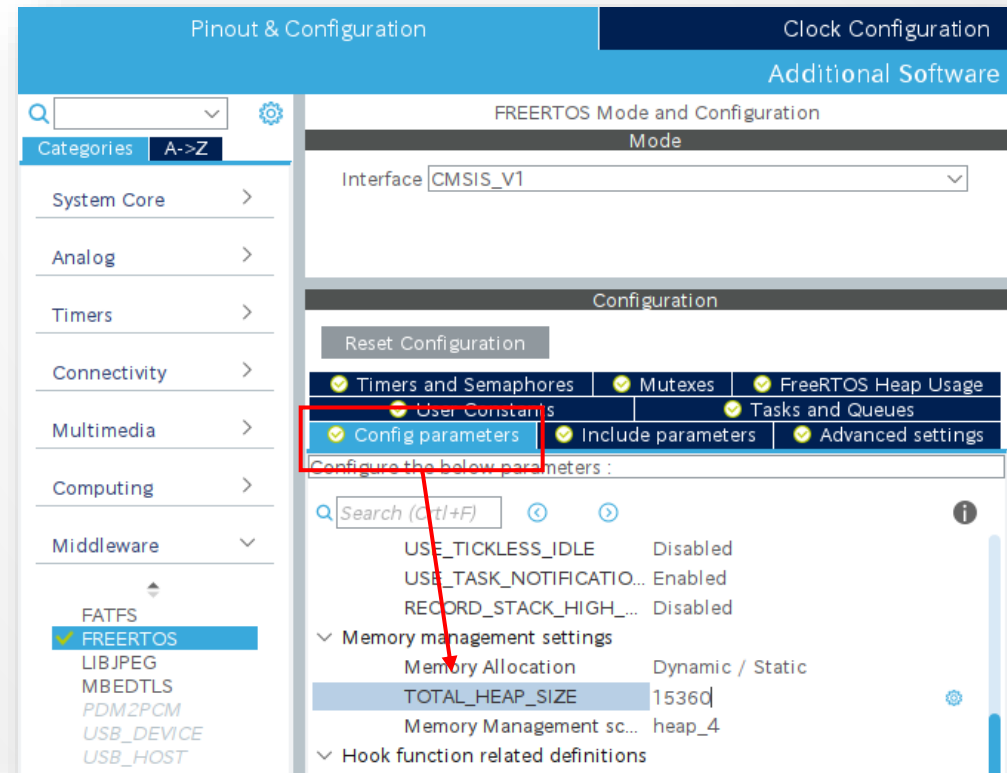
CONFIGURE THE FREERTOS

- Choose CMSIS_V1 as FreeRTOS interface.
- Set the Minimal_stack_size as 256
 - This will increase the stack size of the default task



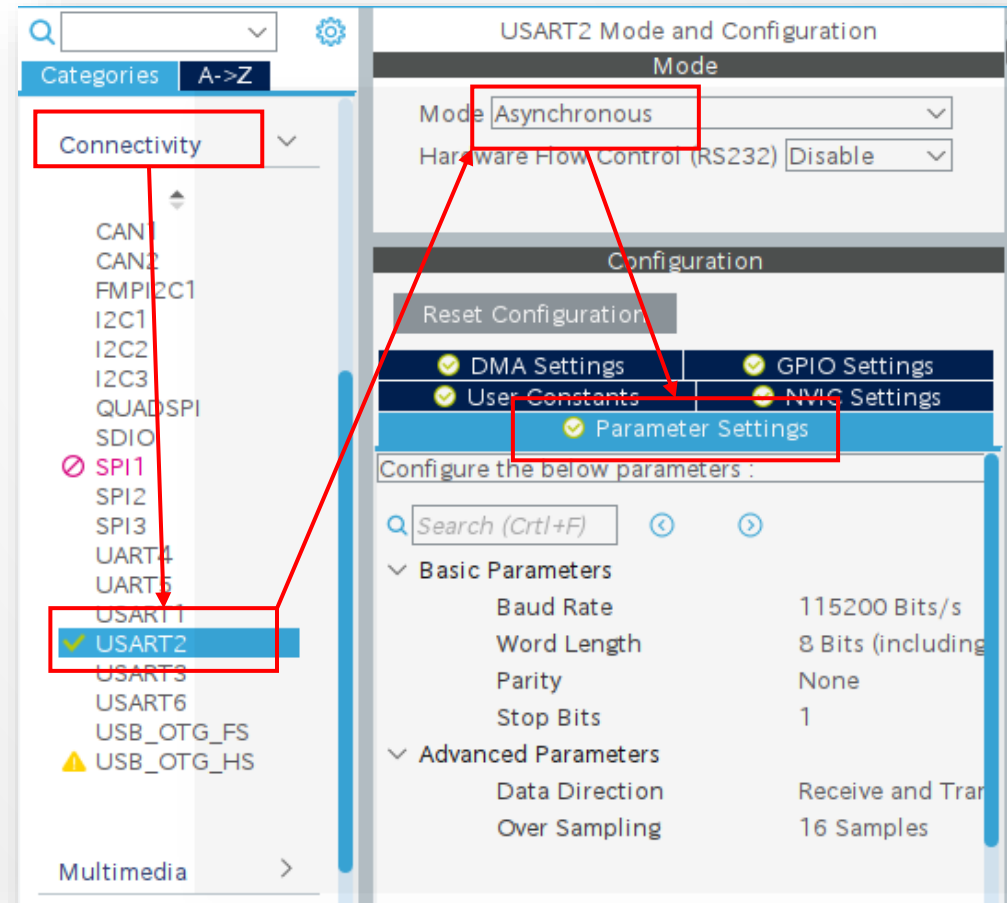
CONFIGURE THE HEAP SIZE

- Set TOTAL_HEAP_SIZE 12kB or more.
- Murasaki uses FreeRTOS heap for all activity.
- At least 12kB is require.



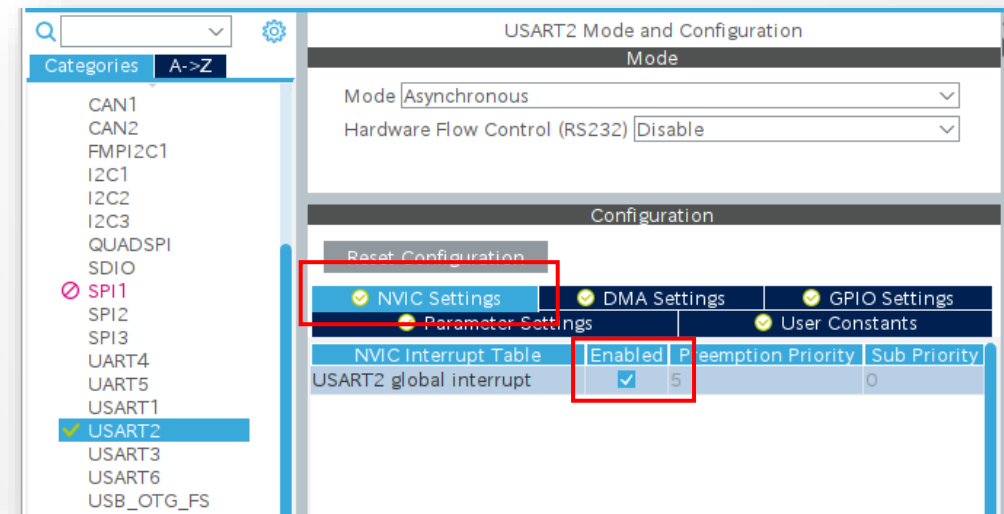
CONFIGURE THE UART

- Nucleo Uses one UART as USB serial port.
- The port is up to the Nucleo
 - UART
 - USART
 - LPUSART
- By default, appropriate port is asynchronous, by CubeIDE
- Make sure the parameters are:
 - 115200bps
 - 8bit
 - No parity
 - 1 stop bit



CONFIGURE THE UART NVIC

- Check the Global interrupt



CONFIGURE THE UART DMA

- Follow procedure :
 1. Select the DMA settings
 2. Add a DMA
 3. Configure it as TX
 4. Add one more DMA
 5. Configure it as RX
 6. Leave the Mode and Width as default
 - Normal
 - Byte

The screenshot displays the 'USART2 Mode and Configuration' window. The 'Mode' section includes a dropdown for 'Mode' set to 'Asynchronous' and a dropdown for 'Hardware Flow Control (RS232)' set to 'Disable'. The 'Configuration' section features a 'Reset Configuration' button and a row of tabs: 'Parameter Settings', 'User Constants', 'NVIC Settings', 'DMA Settings' (highlighted with a red box), and 'GPIO Settings'. Below the tabs is a table with columns 'DMA Request', 'Stream', 'Direction', and 'Priority'. The table contains two rows: 'USART2_RX' with 'DMA1 Stream 5', 'Peripheral To Memory', and 'Low'; and 'USART2_TX' with 'DMA1 Stream 6', 'Memory To Peripheral', and 'Low'. At the bottom left, there are 'Add' and 'Delete' buttons, with 'Add' highlighted by a red box. The 'DMA Request Settings' section at the bottom includes a 'Mode' dropdown set to 'Normal', an 'Increment Address' checkbox, a 'Peripheral' checkbox, a 'Memory' checkbox checked, a 'Use Fifo' checkbox, a 'Threshold' dropdown, and 'Data Width' dropdowns set to 'Byte'.

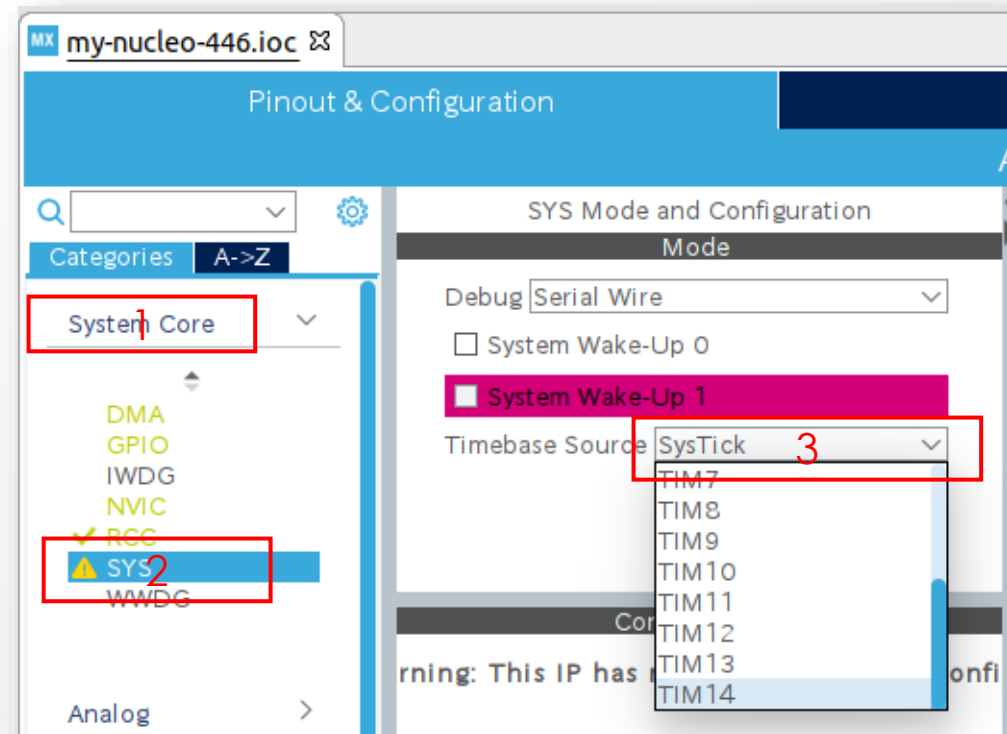
| DMA Request | Stream | Direction | Priority |
|-------------|---------------|----------------------|----------|
| USART2_RX | DMA1 Stream 5 | Peripheral To Memory | Low |
| USART2_TX | DMA1 Stream 6 | Memory To Peripheral | Low |

DMA Request Settings

| | | | | | | | |
|----------|--------------------------|-------------------|--------------------------|------------|--------------------------|--------|-------------------------------------|
| Mode | Normal | Increment Address | <input type="checkbox"/> | Peripheral | <input type="checkbox"/> | Memory | <input checked="" type="checkbox"/> |
| Use Fifo | <input type="checkbox"/> | Threshold | | Data Width | Byte | Byte | |

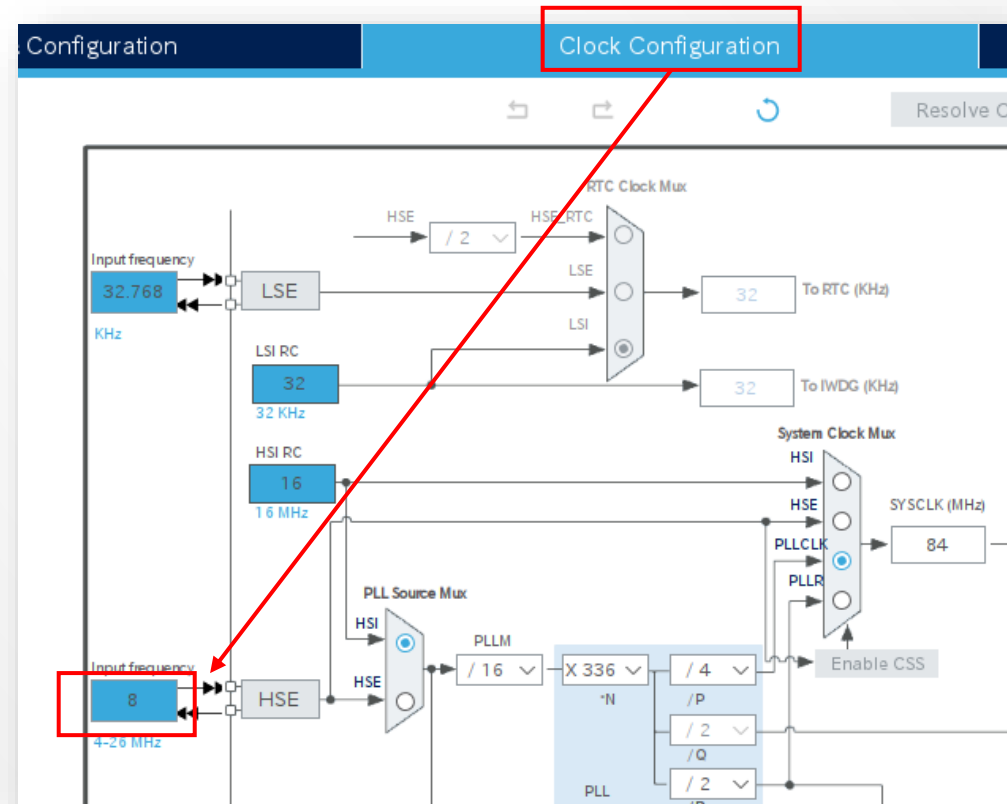
CHANGE THE TIME BASE

- QubeHAL uses its own time base
 - By default, this time base is SysTick
 - Also FreeRTOS uses it.
- Follow the procedure :
 1. Select "System Core"
 2. Select "SYS"
 3. Choose any Timebase Source except SysTick



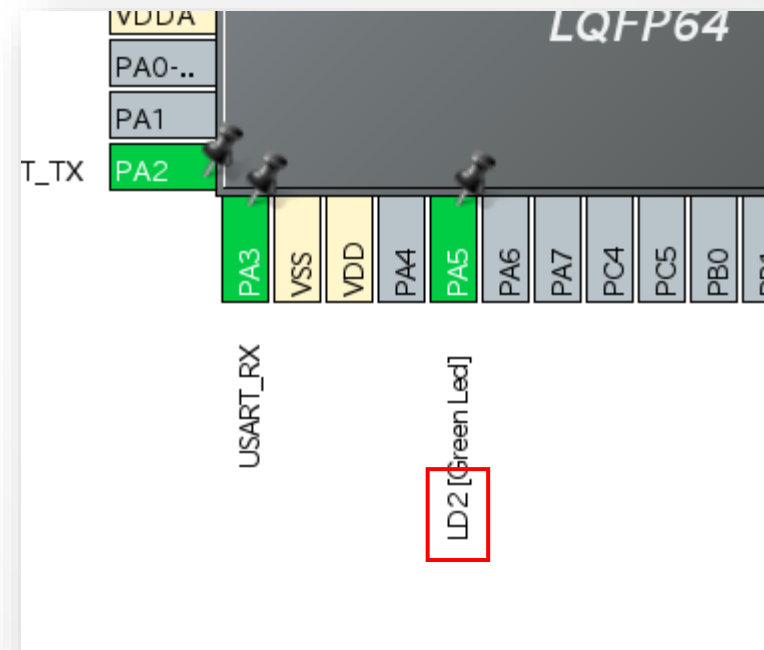
CONFIGURE THE CLOCK (F722 ONLY)

- CubeIDE has bug of Nucleo F722 clock configuration
- Follow the procedure to fix :
 1. Select "Clock Configuration"
 2. Change the "Input Frequency" to 8 MHz.
 - By default, it is 25MHz



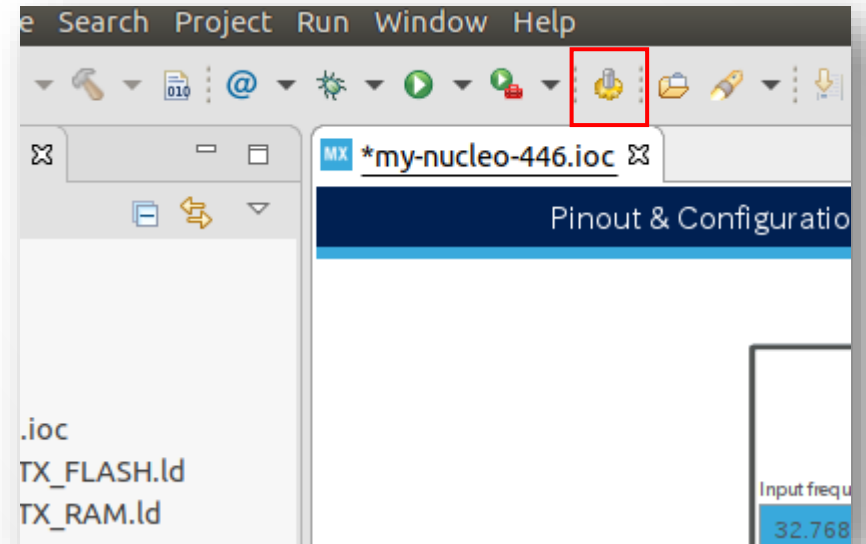
CHECK THE NAME OF LED PIN

- We use LED pin in the Skelton code
- Check the name of LED pin
 - It is board dependent



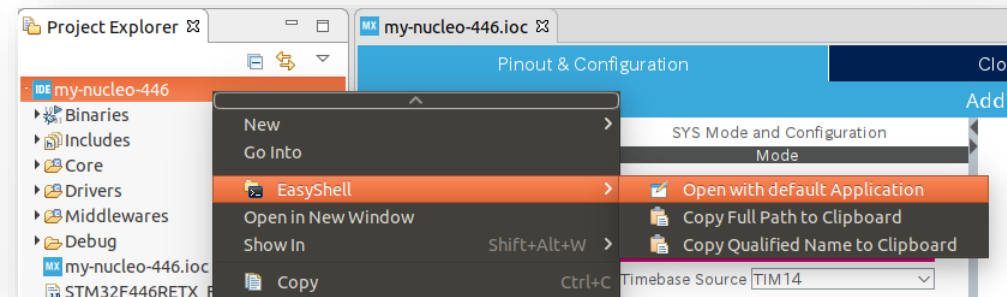
GENERATE A CODE

- Now, time to generate a code.
- Once generated, type Ctrl-B to build.
- Build must be OK.



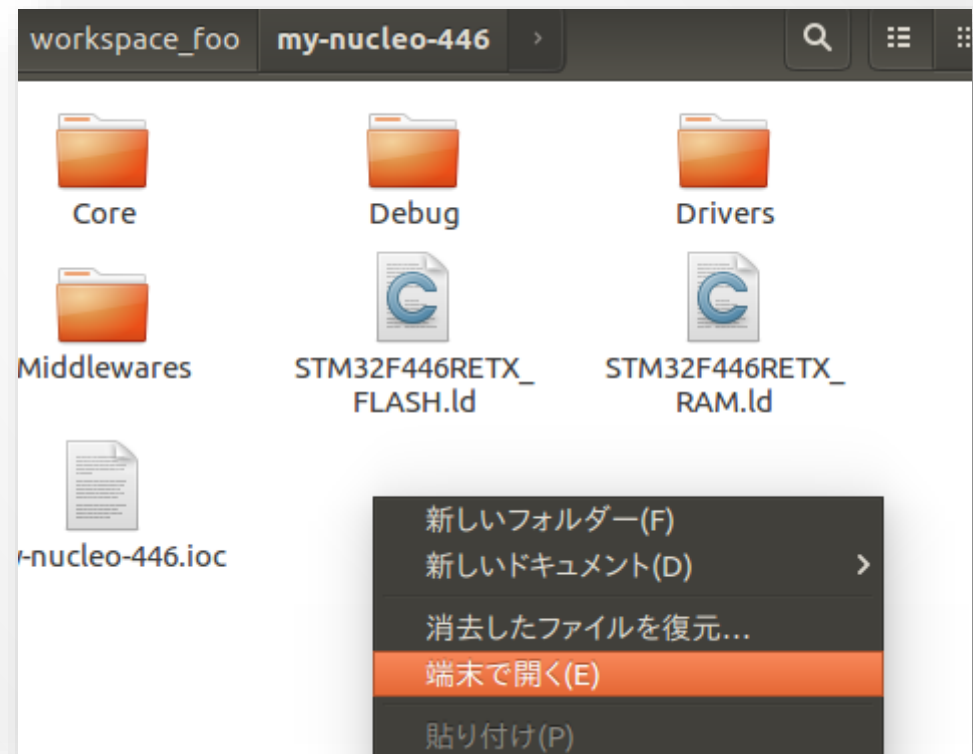
OPEN THE PROJECT LOCATION

- This is most tricky part.
- If you have installed EasyShell, execute “Open with default Application”
 - On Linux, project location is opened by Nautilus file browser
 - On Windows, project location is opened by command prompt window
- If you have not installed EasyShell, go to the project by yourself



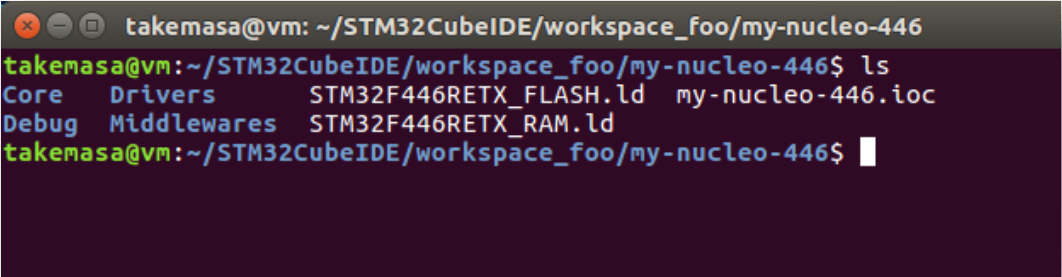
OPEN THE SHELL WINDOW

- Linux only
- From the context menu open the shell window



SHELL WINDOW IS OPEN

- Shell window is located at project
- Make sure the project contents exist
- Now, we are ready to clone the Murasaki repository



```
takemasa@vm: ~/STM32CubeIDE/workspace_foo/my-nucleo-446
takemasa@vm:~/STM32CubeIDE/workspace_foo/my-nucleo-446$ ls
Core Drivers      STM32F446RETX_FLASH.ld  my-nucleo-446.ioc
Debug Middlewares STM32F446RETX_RAM.ld
takemasa@vm:~/STM32CubeIDE/workspace_foo/my-nucleo-446$
```

COPY THE REPOSITORY URL

The screenshot shows the GitHub interface for the repository 'suikan4github/murasaki'. A red box highlights the repository name 'suikan4github/murasaki' in the top navigation bar. A red arrow points from this box to a green 'Clone or download' button in the repository's toolbar. A yellow callout box points to the 'Clone with HTTPS' section of the dropdown menu, containing the URL 'https://github.com/suikan4github/mura'. A second red arrow points from the callout box to the text 'Make sure the protocol is HTTPS'.

suikan4github / **murasaki** Unwatch 1 Star 5 Fork 2

Code Issues 6 Pull requests 0 Actions Projects 2 Wiki Security 0 Insights Settings

STM32 HAL class library Edit

freertos stm32 cubemx stm32cube **murasaki** Manage topics

295 commits 2 branches 0 packages 10 releases 2 contributors MIT

Branch: master New pull request Create new file Upload files Find file **Clone or download**

Clone with HTTPS Use SSH

Use Git or checkout with SVN using the web URL.

https://github.com/suikan4github/mura

Download ZIP

6 days ago

Make sure the protocol is HTTPS

CLONE THE REPOSITORY

```
takemasa@vm: ~/STM32CubeIDE/workspace_foo/my-nucleo-446
takemasa@vm:~/STM32CubeIDE/workspace_foo/my-nucleo-446$ ls
Core  Drivers      STM32F446RETX_FLASH.ld  my-nucleo-446.ioc
Debug Middlewares  STM32F446RETX_RAM.ld
takemasa@vm:~/STM32CubeIDE/workspace_foo/my-nucleo-446$ git clone https://github
.com/suikan4github/murasaki.git
Cloning into 'murasaki'...
remote: Enumerating objects: 197, done.
remote: Counting objects: 100% (197/197), done.
remote: Compressing objects: 100% (179/179), done.
remote: Total 1898 (delta 119), reused 91 (delta 15), pack-reused 17
Receiving objects: 100% (1898/1898), 12.06 MiB | 5.31 MiB/s, done
Resolving deltas: 100% (1341/1341), done.
Checking connectivity... done.
takemasa@vm:~/STM32CubeIDE/workspace_foo/my-nucleo-446$
```

`git clone https://github.com/suikan4github/murasaki.git`

INSTALL MURASKI TO PROJECT

```
takemasa@vm: ~/STM32CubeIDE/workspace_foo/my-nucleo-446/murasaki
# Linux
cd Murasaki
./install

REM Windows
cd Murasaki
wsl ./install

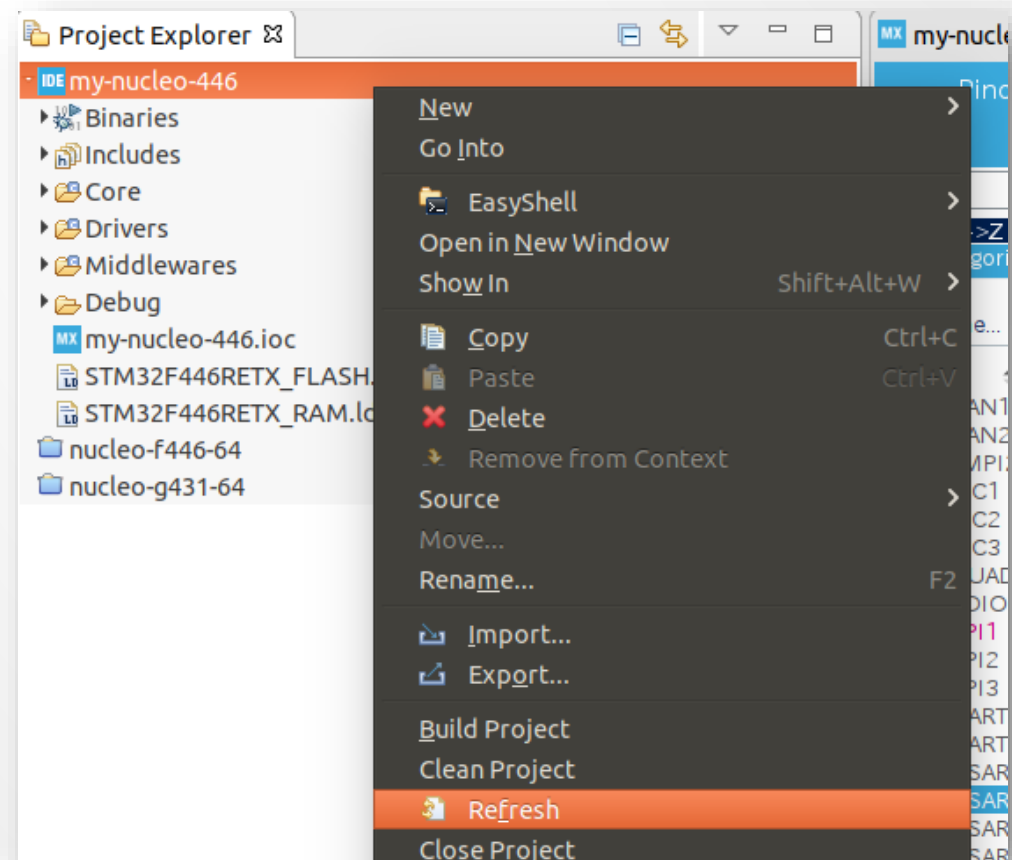
Cloning into 'murasaki'...
remote: Enumerating objects: 1898, done.
remote: Counting objects: 100% (1898/1898), done.
remote: Compressing objects: 100% (1701/1701), done.
remote: Total 1898 (delta 119), reused 91 (delta 15), pack-reused 1701
Receiving objects: 100% (1898/1898), 12.06 MiB | 5.31 MiB/s, done.
Resolving deltas: 100% (1341/1341), done.
Checking connectivity... done.
takemasa@vm:~/STM32CubeIDE/workspace_foo/my-nucleo-446$ cd murasaki/
takemasa@vm:~/STM32CubeIDE/workspace_foo/my-nucleo-446/murasaki$ ./install
Found ../Core/Src as the source directory.
Found ../Core/Inc as the include directory.
Found ../Core/Startup as the startup directory.
template files copied
../Core/Src/main.c modified.
../Core/Startup/startup_stm32f446retx.s modified.
../Core/Src/stm32f4xx_it.c modified.
../Core/Inc/murasaki_include_stub.h created.
takemasa@vm:~/STM32CubeIDE/workspace_foo/my-nucleo-446/murasaki$
```

WHY DO WE NEED INSTALLATION?

- There are several point which CubeIDE Skelton calls Murasaki.
 - InitPlatform()
 - ExecPlatform()
 - HAL's assertion failure hook
 - Spurious Interrupt
 - Hard fault
- These call must be coded by programmer
 - No weak binding routines
- The installer script is the best way to avoid the routine works

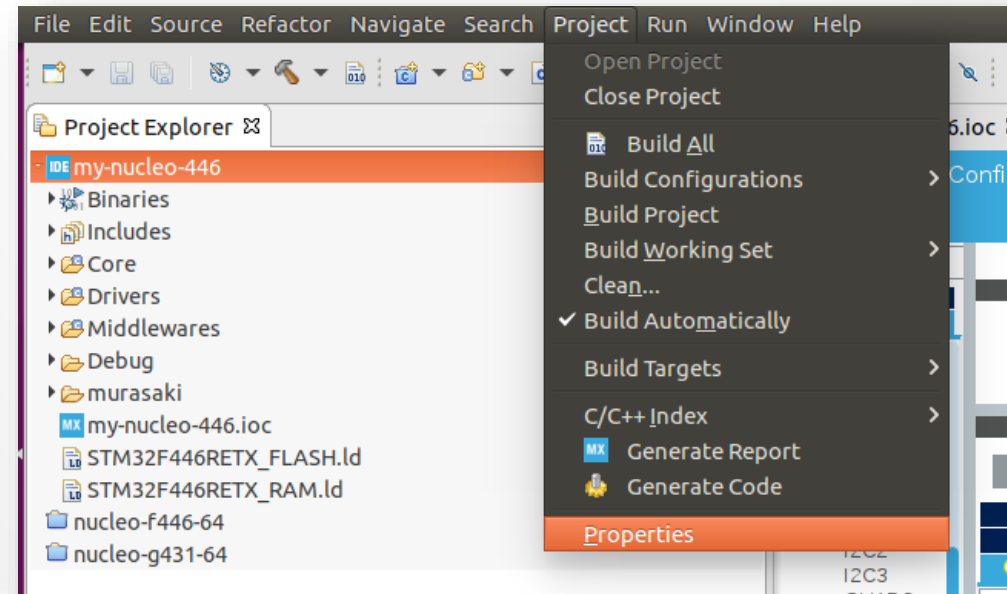
REFRESH THE PROJECT

- While we installed on the shell command, Eclipse doesn't know that
- Follow the procedure :
 1. Select project
 2. Open the context menu
 3. Execute "Refresh"
 4. Now, you can see "Murasaki" in the project



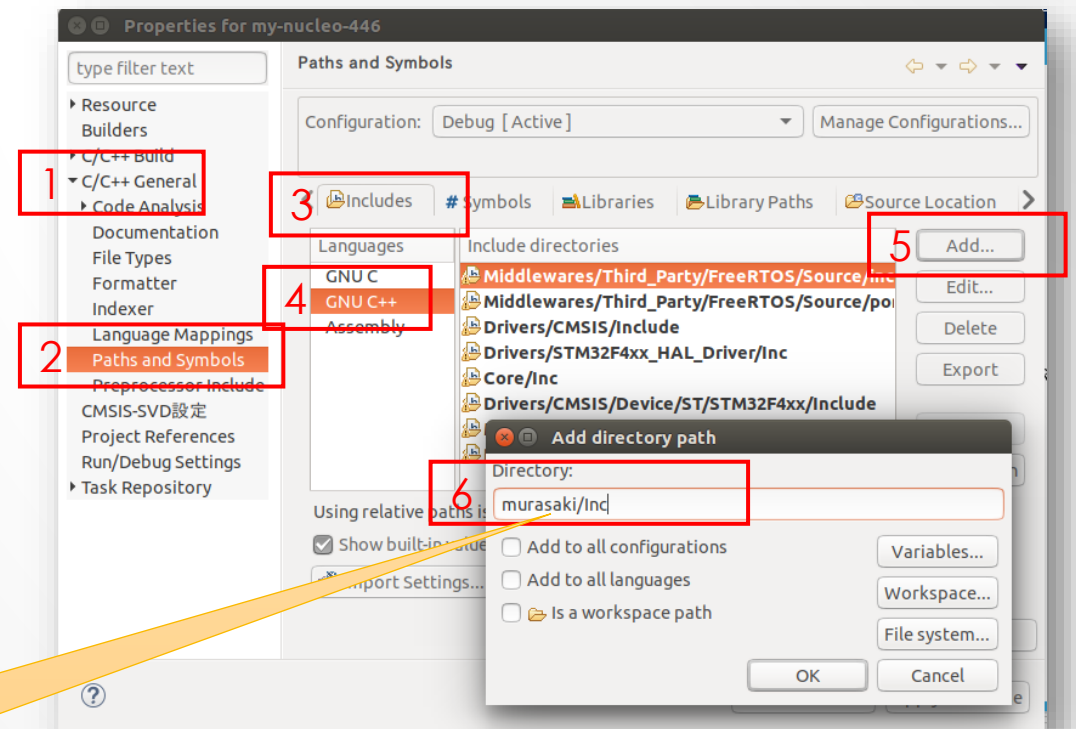
OPEN THE PROPERTY

- Now, we open the project to set :
 - Include directory
 - Source directory



ADD INCLUDE PATH

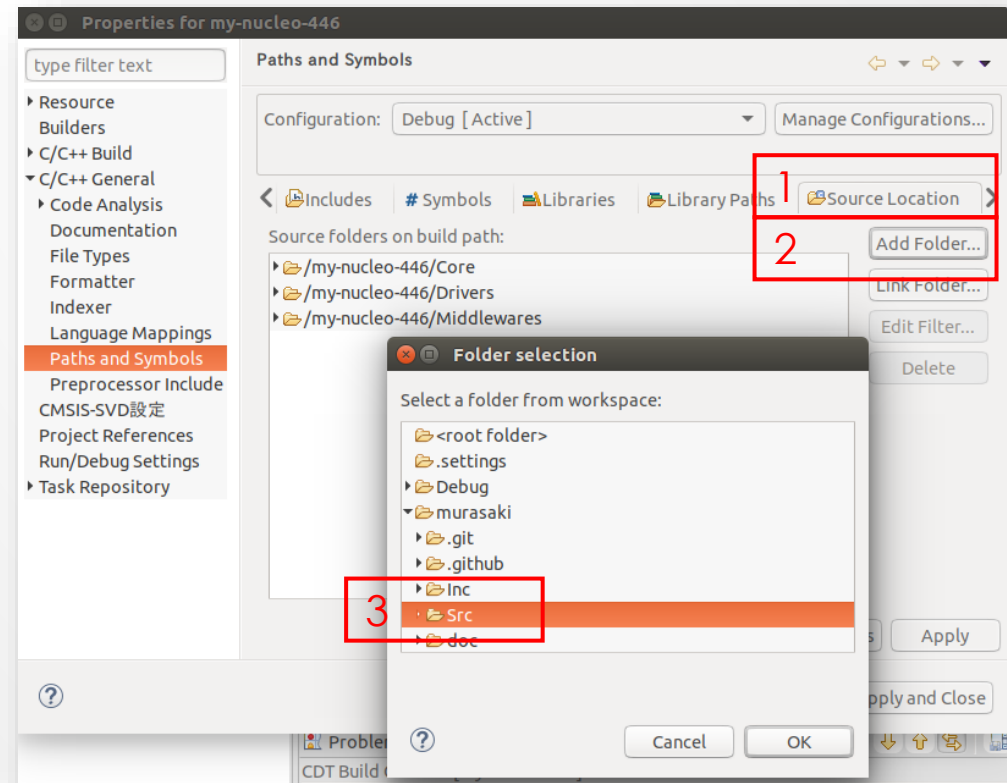
- Follow the procedure :
 1. Open "C/C++ General"
 2. Select "Paths and Symbols"
 3. Select "Includes tab"
 4. Select "GNU C++"
 5. Click "Add"
 6. Write "murasaki/Inc"
 7. Click "OK"



murasaki/Inc

ADD SOURCE LOCATION

- Follow the procedure :
 1. Select "Source Location"
 2. Click "Add Folder..."
 3. Select "murasaki/Src"
 4. Click "OK"



FIX THE UART PORT IDENTIFIER

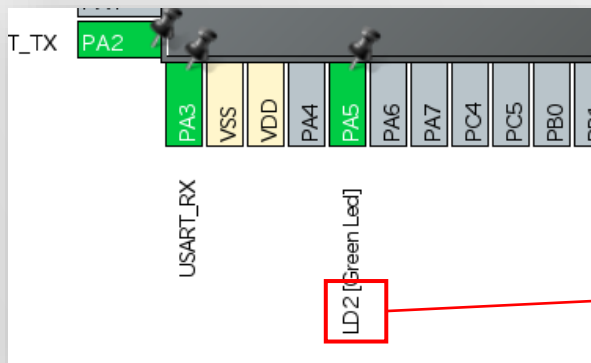
- The correct UART port is defined in main.c
- Apply this identifier to the murasaki_platform.cpp

```
main.c
43 /* USER CODE END PM */
44
45 /* Private variables -----
46 UART_HandleTypeDef huart2;
47 DMA_HandleTypeDef hdma_usart2_rx;
48 DMA_HandleTypeDef hdma_usart2_tx;
49
50 osThreadId defaultTaskHandle;
51 /* USER CODE BEGIN PV */
52
```

```
murasaki_platform.cpp
39 */
40 // Following block is just sample.
41 // Original declaration is in the top of main.c.
42 #if 0
43 extern I2C_HandleTypeDef hi2c1;
44 extern I2C_HandleTypeDef hi2c2;
45 extern SPI_HandleTypeDef hspi1;
46 extern SPI_HandleTypeDef hspi4;
47 extern UART_HandleTypeDef huart2;
48 #endif
49 extern UART_HandleTypeDef huart3;
50
51 /* ----- PLATFORM Prototypes ----- */
52
53 void TaskBodyFunction(const void *ptr);
54
55 /* ----- PLATFORM Implementation ----- */
56
57 // Initialization of the asystem.
58 void InitPlatform()
59 {
60 #if ! MURASAKI_CONFIG_NOCYCCNT
61 // Start the cycle counter to measure the cycle in MURASAKI_SYSLOG.
62 murasaki::InitCycleCounter();
63 #endif
64 // UART device setting for console interface.
65 // On Nucleo, the port connected to the USB port of ST-Link is
66 // referred here.
67 murasaki::platform_uart_console = new murasaki::DebuggerOutput(&huart3);
68 while (nullptr == murasaki::platform_uart_console)
69 ; // stop here on the memory allocation failure.
70
```

ADJUST THE LED IDENTIFIER

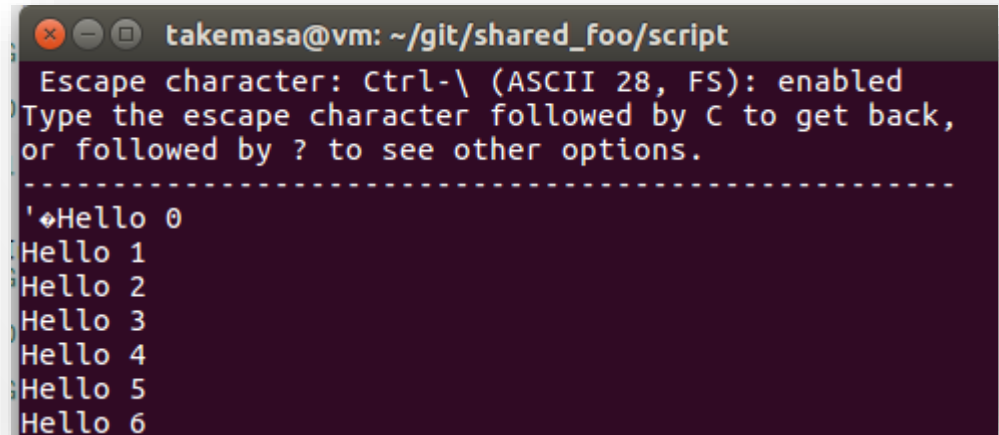
- The LED name is generated by the Device Configuration Tool
- Port name and Pin name have to be adjusted
 - LDx_GPIO_Port
 - LDx_Pin



```
murasaki_platform.cpp
53 void taskbodyfunction(const void *ptr);
54
55 /* ----- PLATFORM Implementation ----- */
56
57 // Initialization of the asystem.
58 void InitPlatform()
59 {
60 #if ! MURASAKI_CONFIG_NOCYCNCNT
61 // Start the cycle counter to measure the cycle in MURASAKI_SYSLOG.
62 murasaki::InitCycleCounter();
63 #endif
64 // UART device setting for console interface.
65 // On Nucleo, the port connected to the USB port of ST-Link is
66 // referred here.
67 murasaki::platform.uart_console = new murasaki::DebuggerUart(&uart3);
68 while (nullptr == murasaki::platform.uart_console)
69 ; // stop here on the memory allocation failure.
70
71 // UART is used for logging port.
72 // At least one logger is needed to run the debugger class.
73 murasaki::platform.logger = new murasaki::UartLogger(murasaki::platform.uart_console);
74 while (nullptr == murasaki::platform.logger)
75 ; // stop here on the memory allocation failure.
76
77 // Setting the debugger
78 murasaki::debugger = new murasaki::Debugger(murasaki::platform.logger);
79 while (nullptr == murasaki::debugger)
80 ; // stop here on the memory allocation failure.
81
82 // Set the debugger as AutoRePrint mode, for the easy operation.
83 murasaki::debugger->AutoRePrint(); // type any key to show history.
84
85 // For demonstration, one GPIO LED port is reserved.
86 // The port and pin names are fined by CubeIDE.
87 murasaki::platform.led = new murasaki::BIT0((LD2_GPIO_Port, LD2_Pin));
88 MURASAKI_ASSERT(nullptr != murasaki::platform.led)
89
```

FINALLY WE CAN RUN

- Build the target
- Debug the target
- And then, resume



```
takemasa@vm: ~/git/shared_foo/script
Escape character: Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
-----
'Hello 0
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
Hello 6
```

FURTHER PROGRAMING WITH MURASAKI

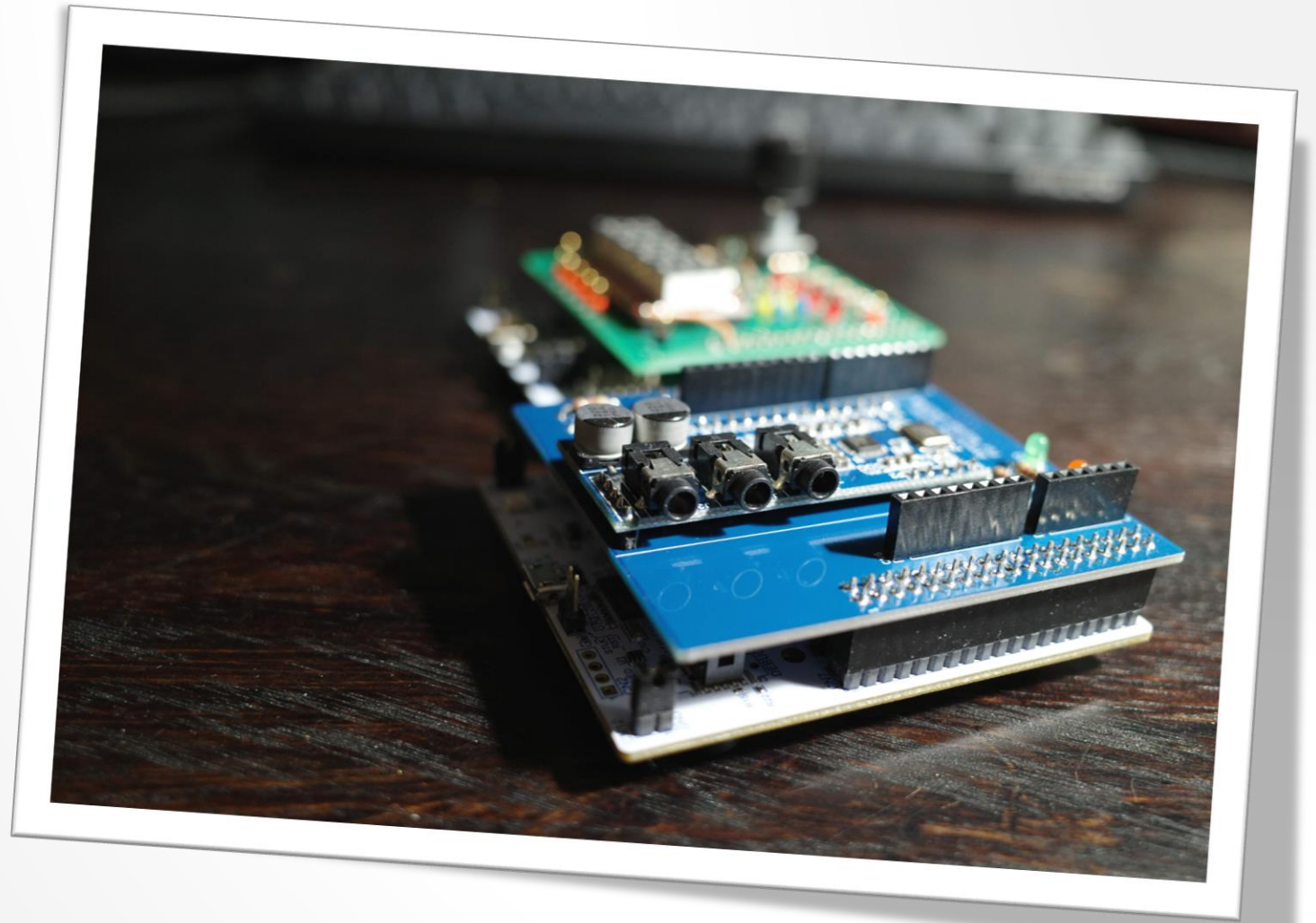
- Platform variable is defined in Inc/platform_def.hpp
 - Programmer must modify the Murasaki::Platform type
 - And must configure the device by Device Configuration Tool
 - And then Initialize them
- There is no Global Interrupt Mask
 - Murasaki assumes the peripheral IO control is in task context
 - Thus, inter-task exclusive access is enough
- If you need to control the IO which is not covered by Murasaki
 - You can control them through the HAL in a Murasaki task
 - You can use its HAL callback as you want
 - Murasaki doesn't interfere such the IO



SUMMARY

MURASAKI

- A class library for STM32 series
 - Multi-task native
 - Synchronous and blocking IO
 - Strict name space and IDE's name completion helps coding
 - Automatic interrupt handling
 - Rich debugging method
- Supporting multiple STM32 MCU series
- Hosted and managed by GitHub repository / tools
- Documentation by Doxygen



THANKS & STAY HOME