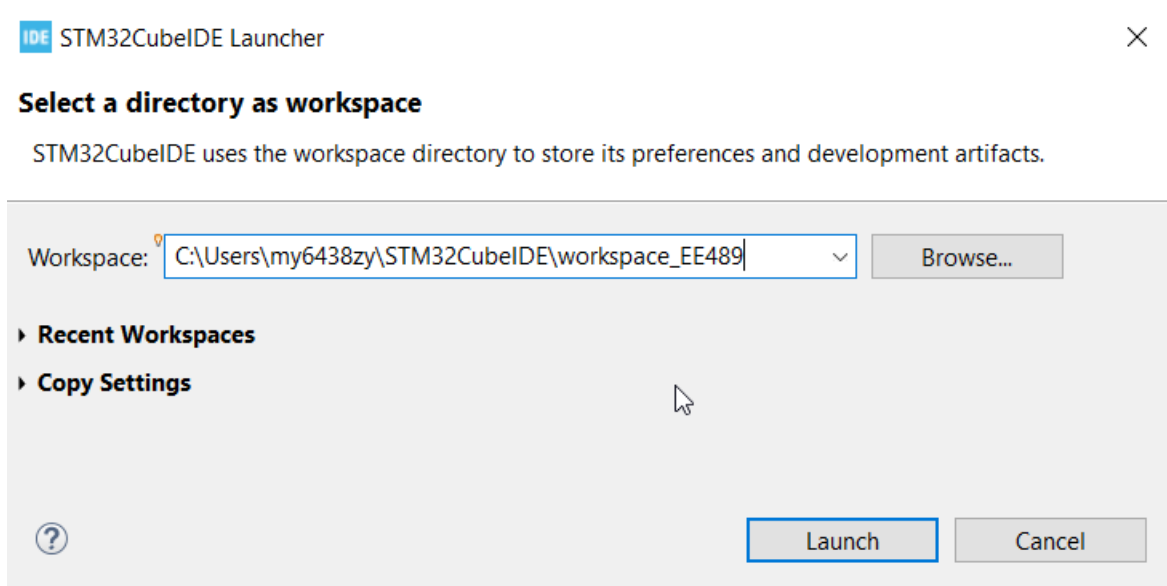


Real-time Embedded Systems Lab Manual

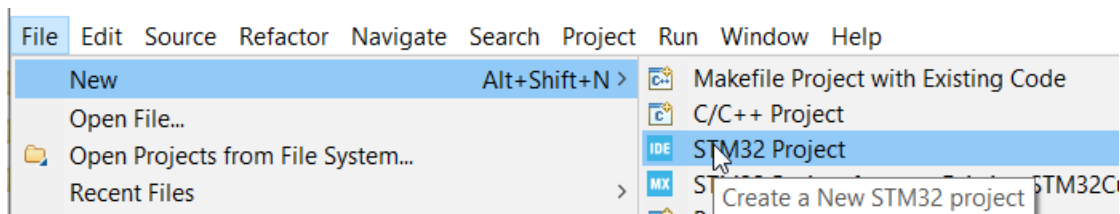
Thread Management in FreeRTOS – Part 1

1. Creating a new STM32 IDE project

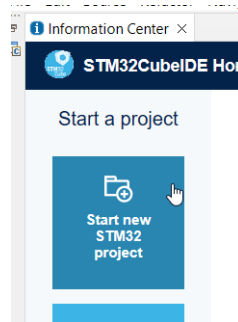
- a. Starting STM32Cube IDE. This IDE first displays the **STM32CubeIDE Launcher** dialog with workspace selection. The first time the IDE is started, it presents a default location and workspace name. Any newly created project is stored in this workspace. The workspace is created if it does not yet exist. For example, the workspace is set as `C:\Users\xxxx\STM32CubeIDE\workspace_EE489` (xxxx could be your starID). Then, click **Launch**.



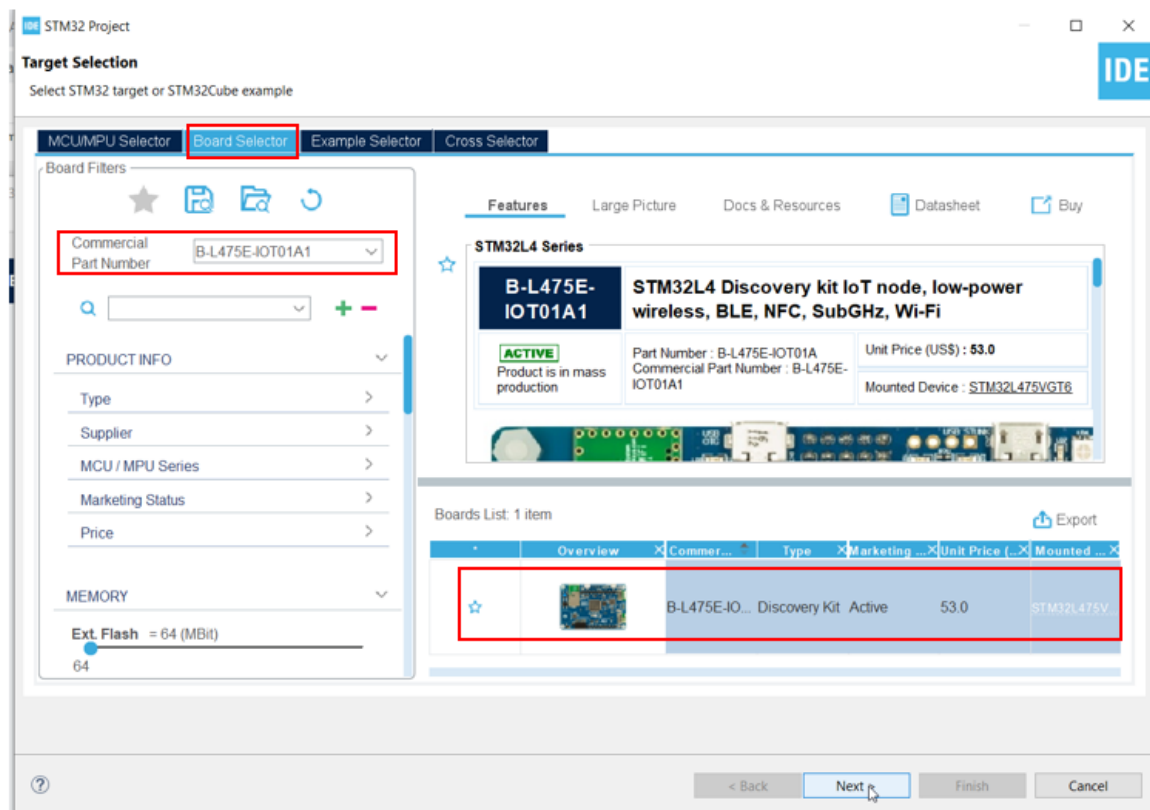
- b. Select **File >> New >> STM32 Project**.



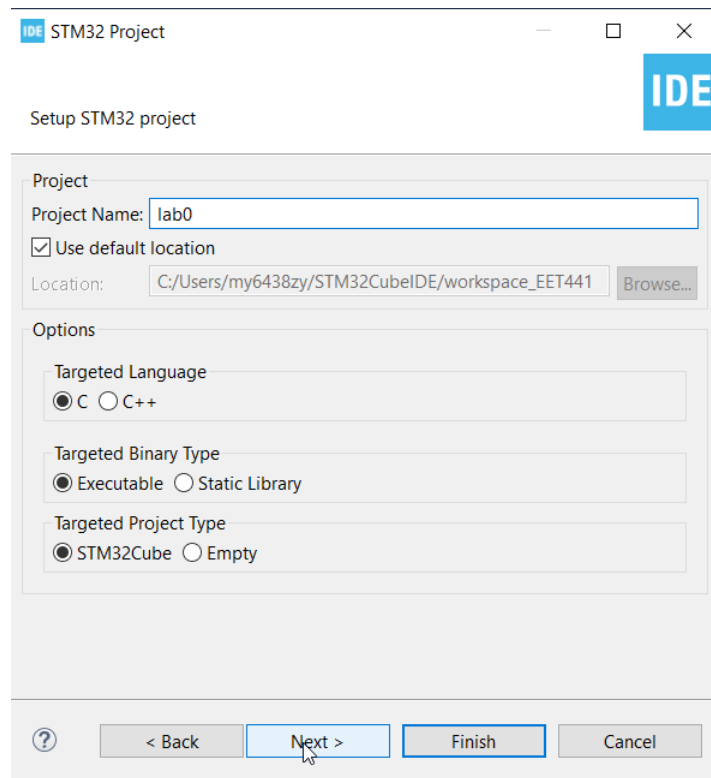
Alternatively, you could also start a new project by clicking the **“Start New STM32 project”** icon on the **“Information Center”** page, as shown below.



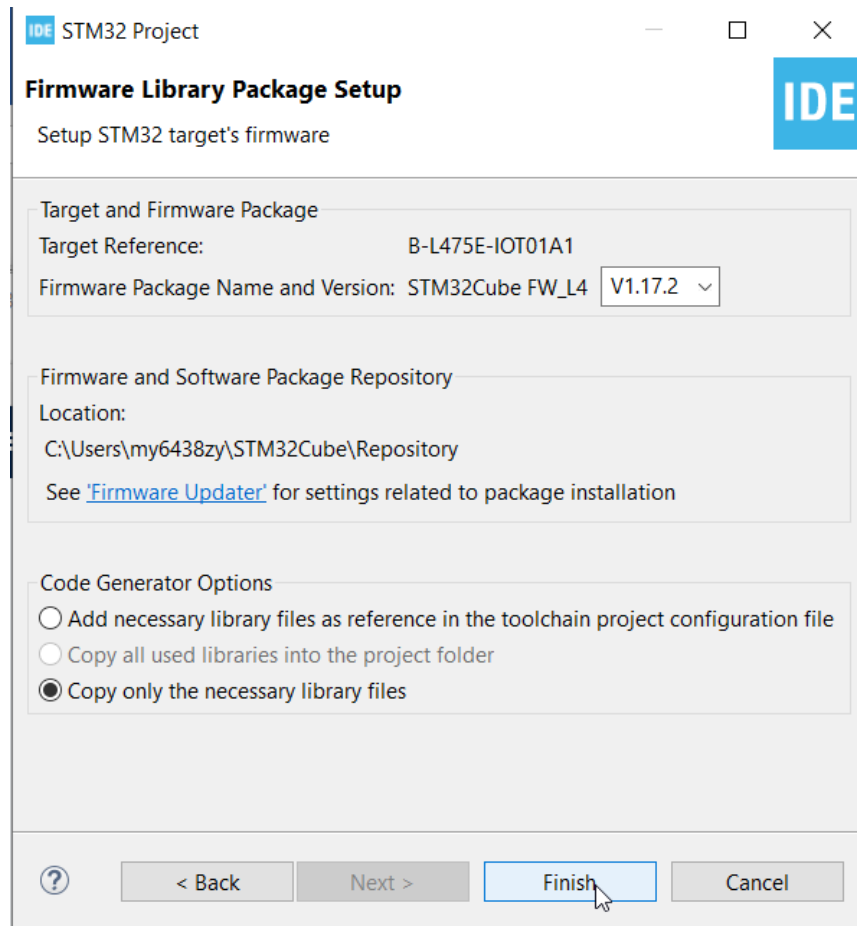
- c. On the **Target Selection** window, select the **Board Selector** tab. Then, select B-L475E-IOT01A from the drop-down list under the *Commercial Part Number* search panel. On the right-bottom of this window, Select B-L475E-IOT01A among the boards list that meet the search requirements specified before (only one board is listed here), and Click the **Next >** button at the bottom.



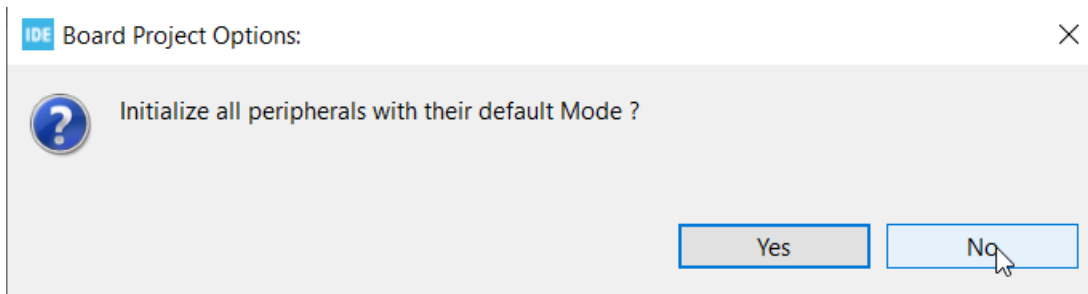
- d. The *Project setup* page opens. Enter a project name, i.e. lab0, and leave the settings for the project in the dialog boxes as default. Then click Next >.



- e. The *Firmware Library Package Setup* page opens. In this page, it is possible to select the STM32 Cube firmware package to use when creating the project. For this lab, the default settings are used. Press **Finish** to create the project.



- f. Answer **No** when prompted to **initialize all peripherals in their default mode** as we do not want to initialize all peripherals, but only the ones that we use.



2. Opening the Project Manager view

- a. Select the “Project” tab on the left side. Leave all the settings as default.

The screenshot shows the STM32CubeIDE Project Manager window. The left sidebar has four tabs: Project, Code Generator, Advanced Settings, and Tools. The 'Project' tab is selected. The main area is divided into sections: Project Settings, Code Generator, and Advanced Settings. The Project Settings section includes fields for Project Name (lab0), Project Location (C:\Users\my6438zy\STM32CubeIDE\workspace_EET441), Application Structure (Advanced), Toolchain Folder Location (C:\Users\my6438zy\STM32CubeIDE\workspace_EET441\lab0), and Toolchain / IDE (STM32CubeIDE). The Code Generator section has a checkbox for 'Generate Under Root' which is checked. The Advanced Settings section includes Linker Settings (Minimum Heap Size: 0x200, Minimum Stack Size: 0x400), Thread-safe Settings (Cortex-M4NS, Enable multi-threaded support: unchecked, Thread-safe Locking Strategy: Default - Mapping suitable strategy depending on RTOS selection), Mcu and Firmware Package (Mcu Reference: STM32L475VGTx, Firmware Package Name and Version: STM32Cube FW_L4 V1.17.2, Use latest available version: checked), and Firmware Relative Path.

- b. Select the “Code Generator” tab on the left side. Optionally, under **HAL settings**, check **set all free pins as analog (to optimize the power consumption)**.

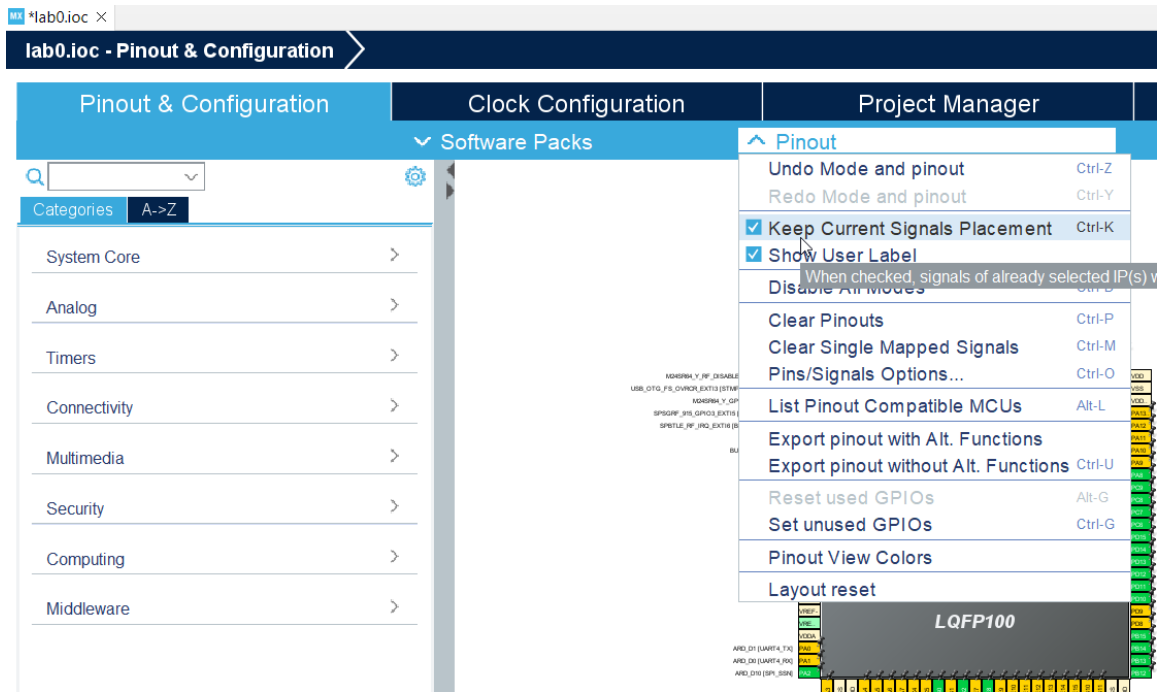
The screenshot shows the STM32CubeIDE Project Manager window with the 'Code Generator' tab selected. The left sidebar has four tabs: Project, Code Generator, Advanced Settings, and Tools. The 'Code Generator' tab is selected. The main area is divided into sections: Project, Code Generator, and Advanced Settings. The Project section has three radio buttons: 'Copy all used libraries into the project folder' (unselected), 'Copy only the necessary library files' (selected), and 'Add necessary library files as reference in the toolchain project configuration file' (unselected). The Code Generator section has a section for 'Generated files' with three checkboxes: 'Generate peripheral initialization as a pair of '.c/.h' files per peripheral' (unchecked), 'Backup previously generated files when re-generating' (unchecked), and 'Keep User Code when re-generating' (checked). The Advanced Settings section has a section for 'HAL Settings' with two checkboxes: 'Set all free pins as analog (to optimize the power consumption)' (checked) and 'Enable Full Assert' (unchecked). The Template Settings section has a text field for 'Select a template to generate customized code' and a 'Settings...' button.

3. Opening the Clock Configuration view

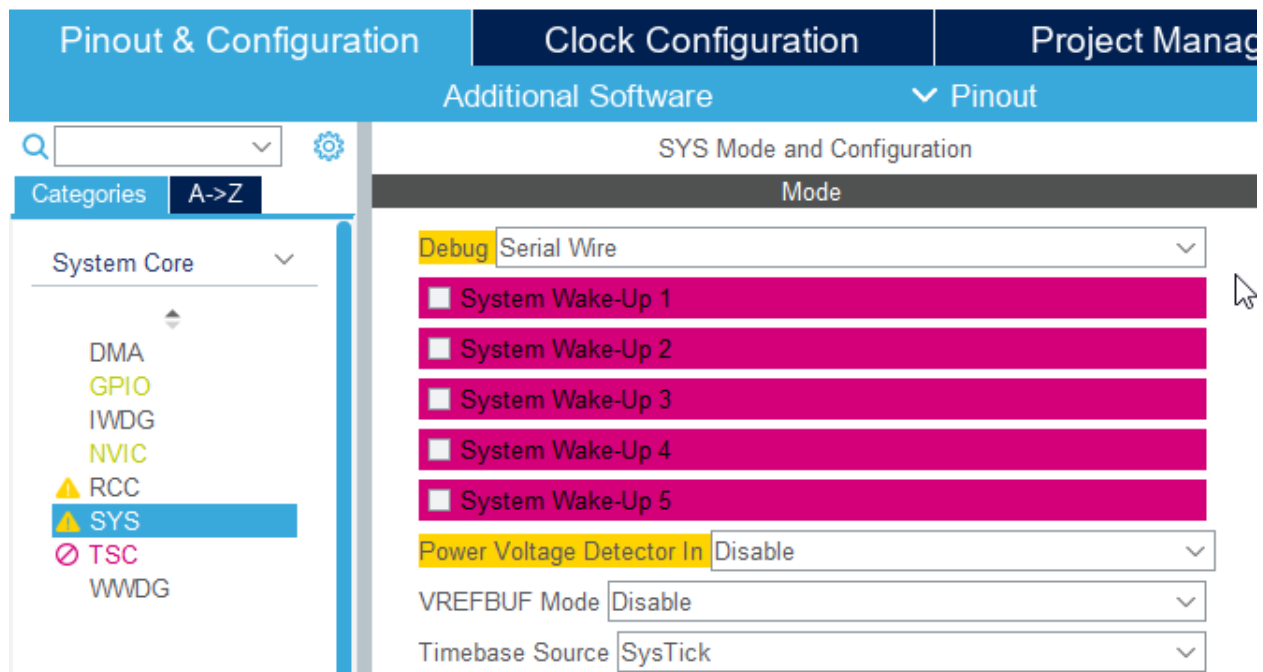
In this view, keep the default settings of clock. No change is needed.

4. Opening the Pinout&Configuration view

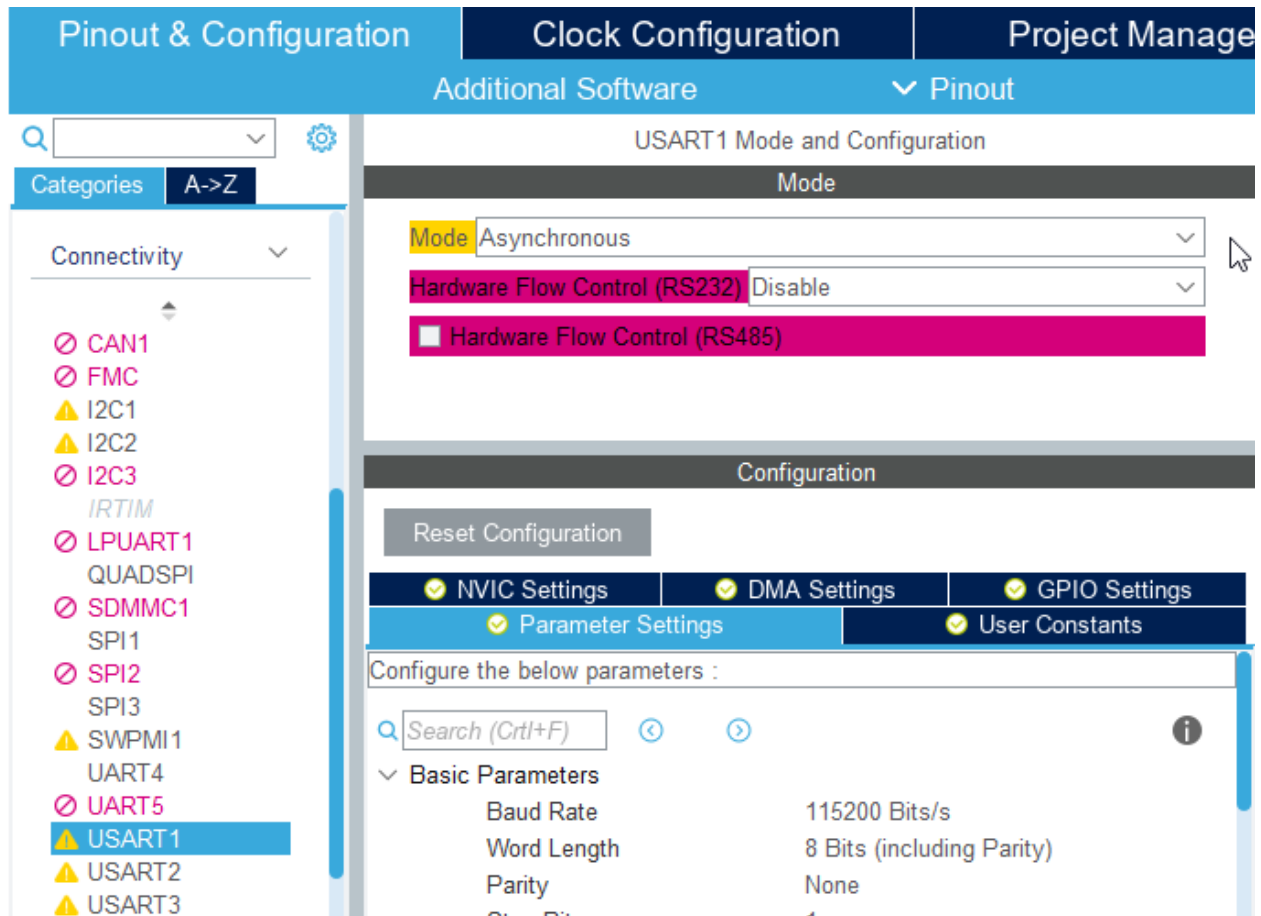
- a. By default, the **Pinout** view shows. Since the MCU pin configurations must match the B-L475E-IOT01A board, enable *Pinout>>Keep Current Signals Placement* to maintain the peripheral function allocation to a given pin. **This step is optional.**



- b. Select the category **System Core** >> **SYS**, configure its **Debug** mode as *Serial Wire*; keep the default setting of **Timebase Source** as SysTick.

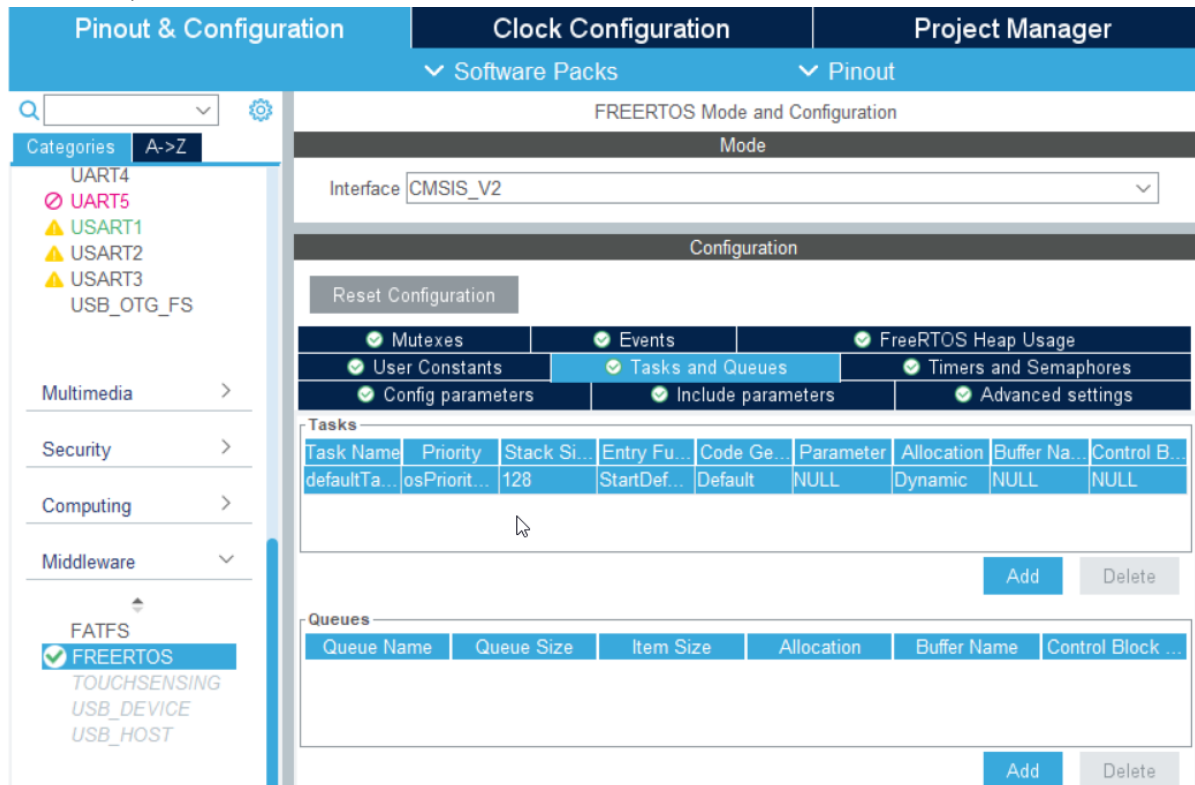


- c. Select the category **Connectivity** >> **USART1**, configure its **Mode** to *Asynchronous*. (USART1 shares the same pins as ST-LINK). Keep the rest mode and configuration parameters as default. This enables sending and receiving messages between PC and the board via USART1.

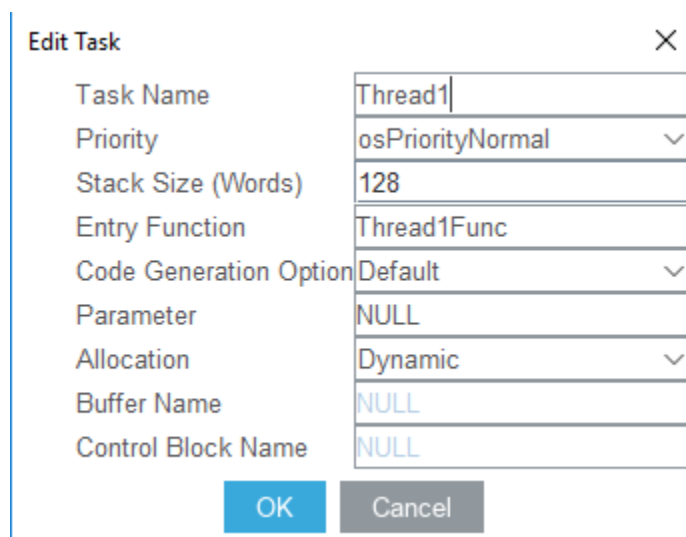


5. Configuring the Middleware FreeRTOS

- a. Select *Middleware>>FREERTOS*, On the **Mode** panel, select **CMSIS_V2 Interface** to enable *FreeRTOS*;



- b. On the **Configuration** panel at the bottom, select the tab **Tasks and Queues**. A default task is already under Tasks tab, which could be edited, but not deleted. Double click the default task to open the **Edit Task** window. Change **Task Name** and **Entry Function** as below.



- c. Then, click the **Add** button to add a new task; Change **Task Name** and **Entry Function** as *Thread2* and *Thread2Func* respectively, and set its **Priority** also as **osPriorityNormal**.

New Task

| | |
|------------------------|------------------|
| Task Name | Thread2 |
| Priority | osPriorityNormal |
| Stack Size (Words) | 128 |
| Entry Function | Thread2Func |
| Code Generation Option | Default |
| Parameter | NULL |
| Allocation | Dynamic |
| Buffer Name | NULL |
| Control Block Name | NULL |

OK Cancel

Create a constant variable by selecting the “User Constants” tab, click “add” to enter a constant name “mainDELAY_LOOP_COUNT”, and constant Value “0xffffffff”.

Search Constants

add remove

| Constant Name | Constant Value |
|---------------|----------------|
|---------------|----------------|

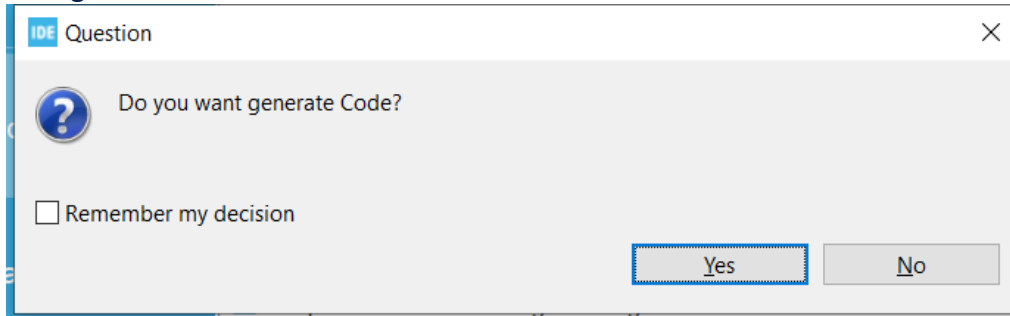
User Constants

| | |
|----------------|----------------------|
| constant Name | mainDELAY_LOOP_COUNT |
| constant Value | 0xffffffff |

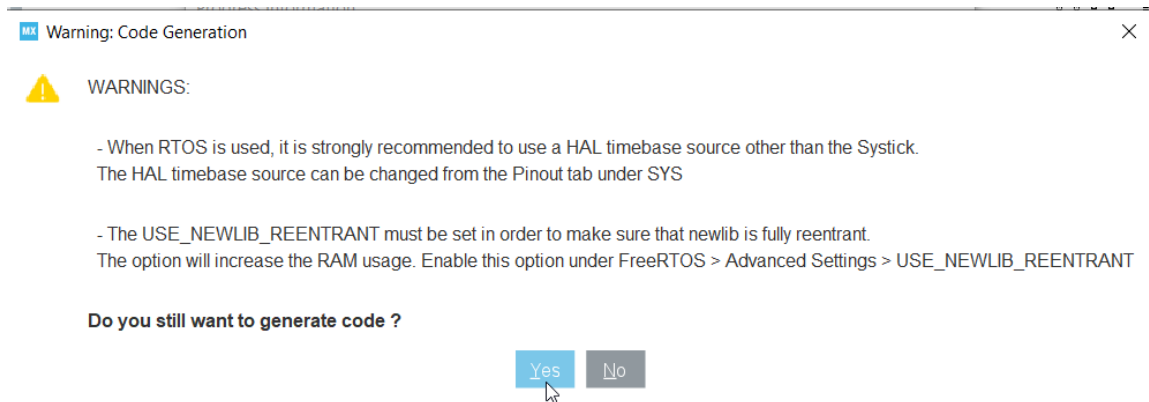
OK Cancel

6. Generating Code

- a. When changes are made in the STM32CubeMX editor, the .ioc file in the tab is marked as changed. When the file is saved, a dialog opens asking “Do you want to generate Code?”, making it easy to generate new code in the project that supports the new device configuration. Click Yes.

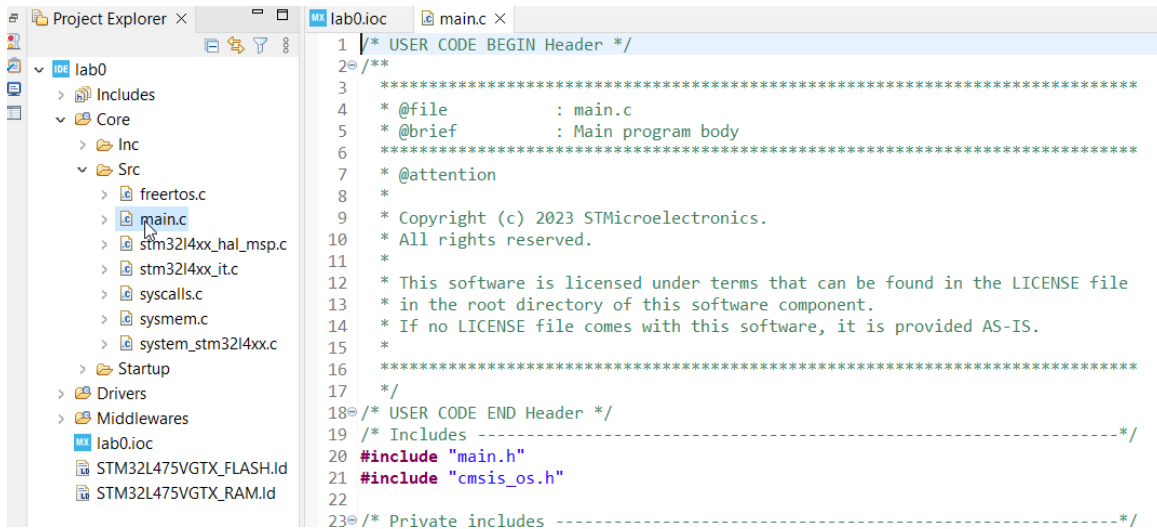


- b. You may get a warning pop-up window about using sysclk as time base. You could just click “yes” to ignore it.



7. Editing the code in CubeIDE.

- a. Open main.c file in the editor window.



- b. Between `/* USER CODE BEGIN 0 */` and `/* USER CODE END 0 */`, define the functions: `putchUSART1()` and `putsUSART1()`.

```
void putchUSART1 (char ch)
{
    /* Place your implementation of fputc here */
    /* e.g. write a character to the serial port and Loop until the end of
    transmission */
    while (HAL_OK != HAL_UART_Transmit(&huart1, (uint8_t *) &ch, 1, 30000))
    {
        ;
    }
}

void putsUSART1 (char* ptr)
{
    while(*ptr)
    {
        putchUSART1(*ptr++);
    }
}
```

```

75 /* USER CODE BEGIN 0 */
76 void putchUSART1 (char ch)
77 {
78     /* Place your implementation of fputc here */
79     /* e.g. write a character to the serial port and loop until the end of transmission */
80     while (HAL_OK != HAL_UART_Transmit(&huart1, (uint8_t *) &ch, 1, 30000))
81     {
82     };
83     }
84 }
85
86 void putsUSART1 (char* ptr)
87 {
88     while(*ptr)
89     {
90         putchUSART1(*ptr++);
91     }
92 }
93 /* USER CODE END 0 */
~.

```

- c. Between `/* USER CODE BEGIN 2 */` and `/* USER CODE END 2 */` in the `main()` function, Add the statement as below

```
putsUSART1("\n\rFreeRTOS Lab 0\n\r");
```

```

121 /* Initialize all configured peripherals */
122 MX_GPIO_Init();
123 MX_USART1_UART_Init();
124 /* USER CODE BEGIN 2 */
125 putsUSART1("\n\rFreeRTOS Lab 0\n\r");
126 /* USER CODE END 2 */

```

- d. Find the generated `Thread1Func()` function and edit it as below. Comment out the line **`osDelay(1);`** for this lab as we will define a continuous task (always ready to run), not a periodic task.

```

515 /* USER CODE END Header_Thread1Func */
516 void Thread1Func(void *argument)
517 {
518     /* USER CODE BEGIN 5 */
519     volatile unsigned long ul;
520     /* Infinite loop */
521     for(;;)
522     {
523         putsUSART1("\n\rThread 1 is running.\n\r");
524
525         for(ul = 0; ul < mainDELAY_LOOP_COUNT; ul++)
526         {}
527
528         // osDelay(1);
529     }
530     /* USER CODE END 5 */
531 }
~.

```

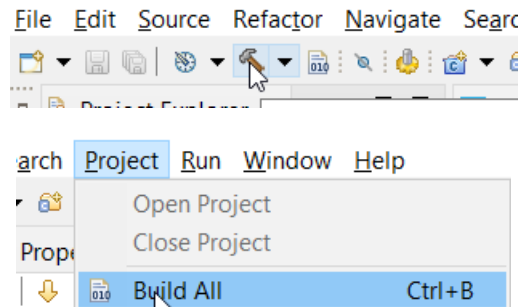
- e. Edit the Thread2Func() function in the similar way. The only difference is the string for display.

```
putsUSART1("\n\rThread 2 is running.\n\r");

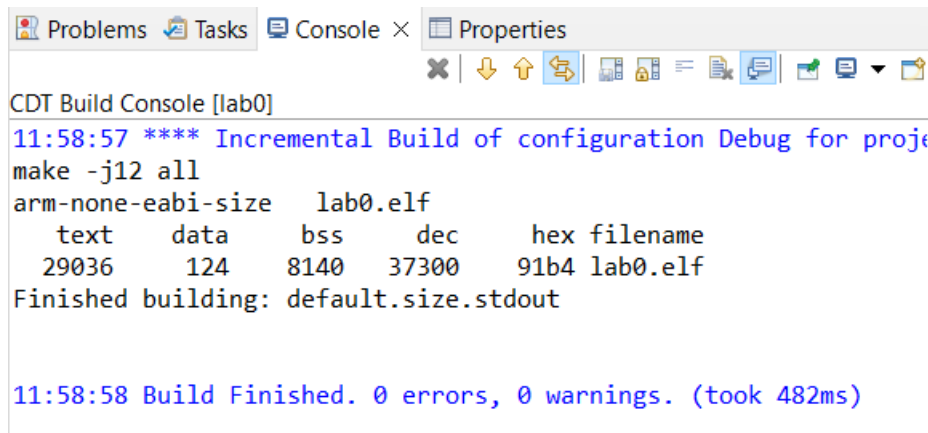
539 /* USER CODE END Header_Thread2Func */
540 void Thread2Func(void *argument)
541 {
542     /* USER CODE BEGIN Thread2Func */
543     volatile unsigned long ul;
544     /* Infinite loop */
545     for(;;)
546     {
547         putsUSART1("\n\rThread 2 is running.\n\r");
548
549         for(ul = 0; ul < mainDELAY_LOOP_COUNT; ul++)
550             {}
551         // osDelay(1);
552     }
553     /* USER CODE END Thread2Func */
554 }
```

8. Build and debug the project in CubeIDE.

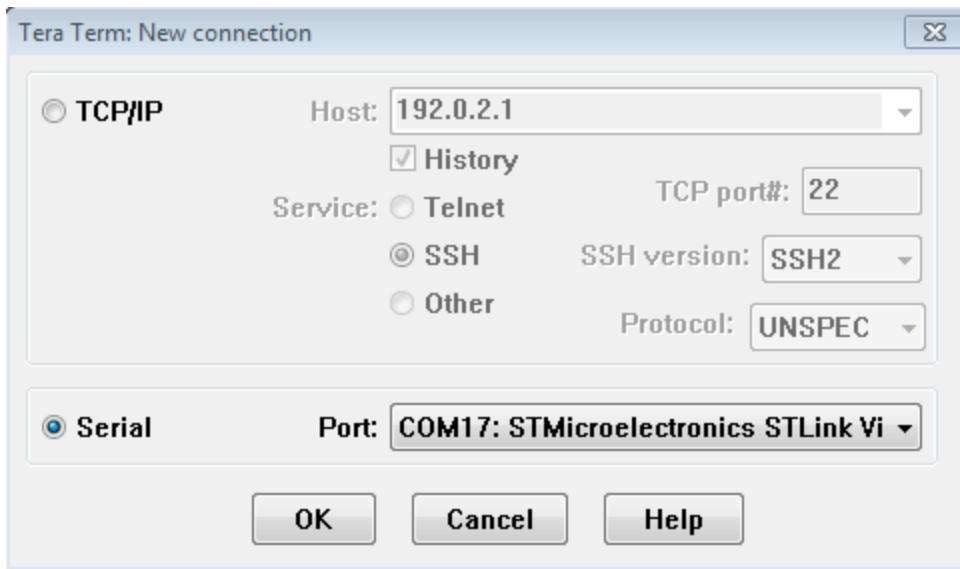
- a. Click the Build button on the menu bar, or Select Project >> Build all target files as below.



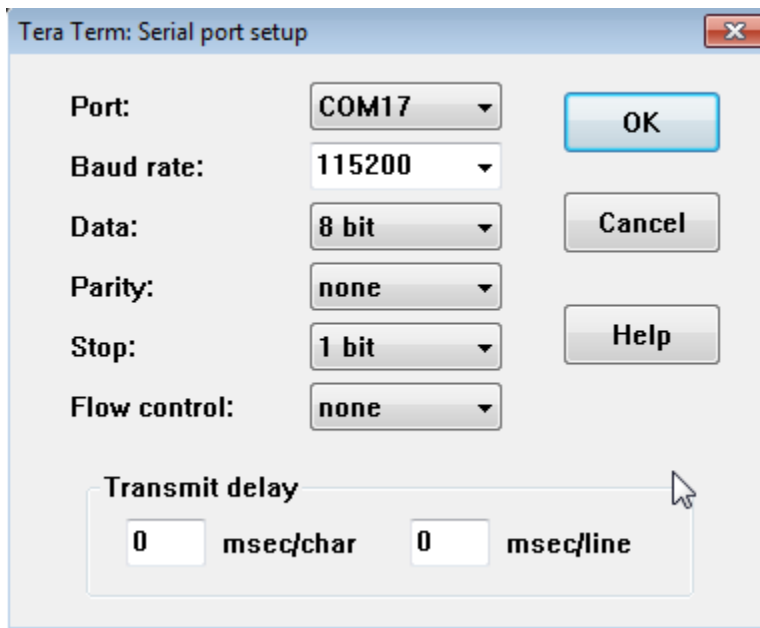
- b. You should get the following outputs without errors from the **Console** window.



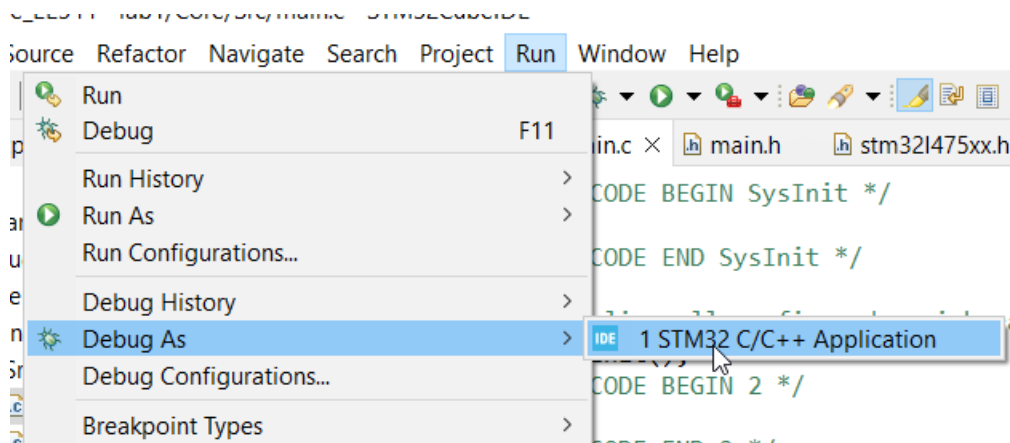
- c. Open the Tera Term window. On the popup *New Connection* window, Select *Serial >> COMXX: STMicroelectronics STLink Virtual COM port* from the drop down *Port* list.



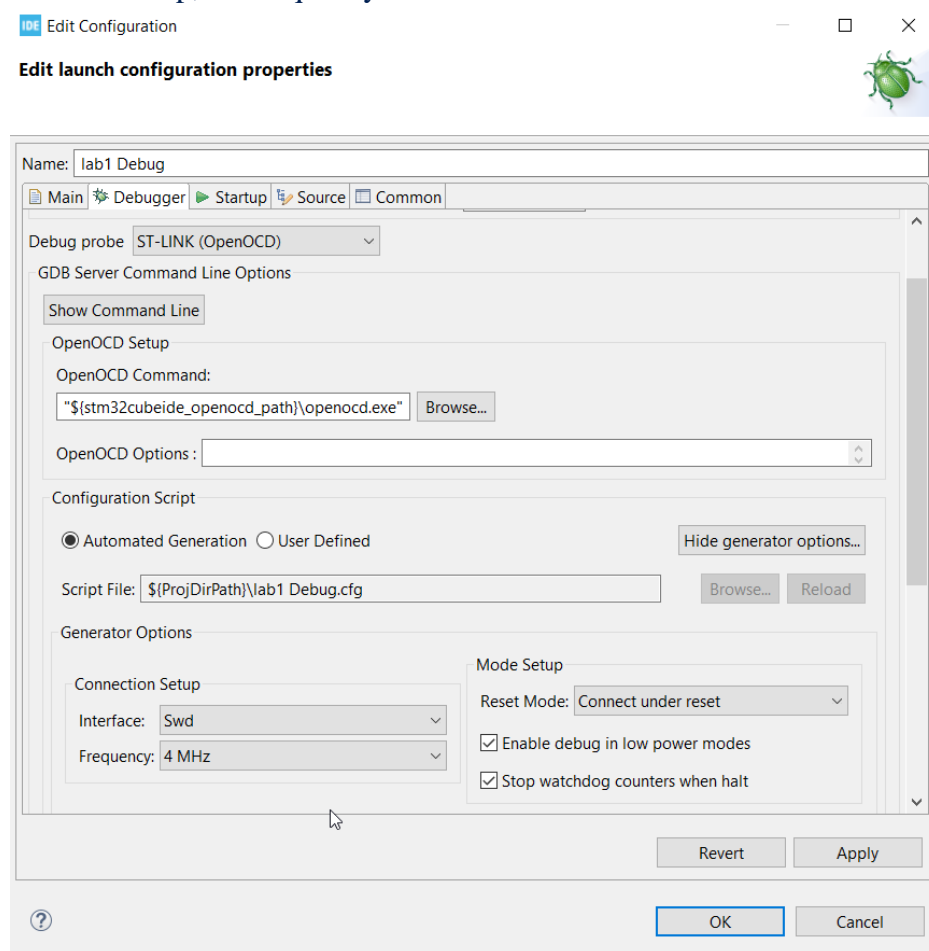
d. Configure *Setup>> Serial Port....* Select *Baud rate* as 115200, click OK.



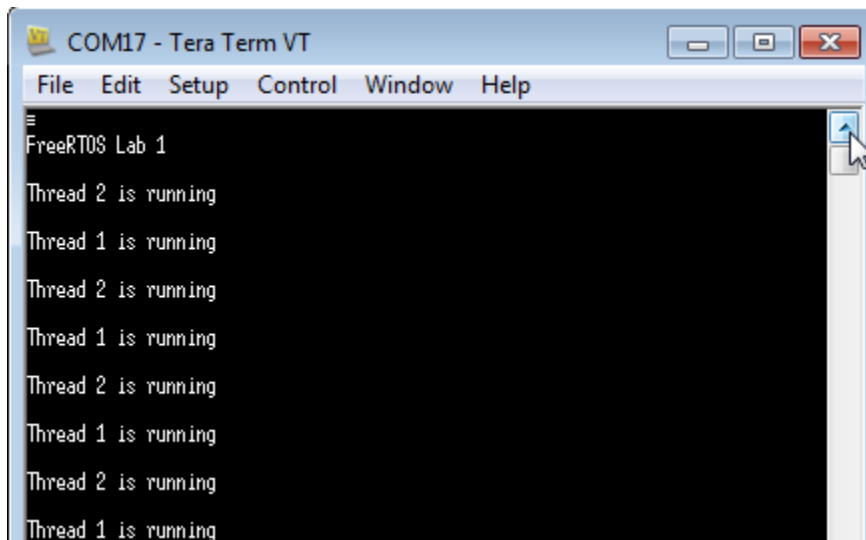
- e. Debug the project by selecting **Run >> Debug As >> STM32C/C++ Application**
The CubeIDE window enters the debug perspective.



- f. The Debug Configuration window pops up if the board is connected. Under *Debug probe*, choose **ST-LINK(OpenOCD)**.
Under Connection Setup, set frequency to 4MHz.



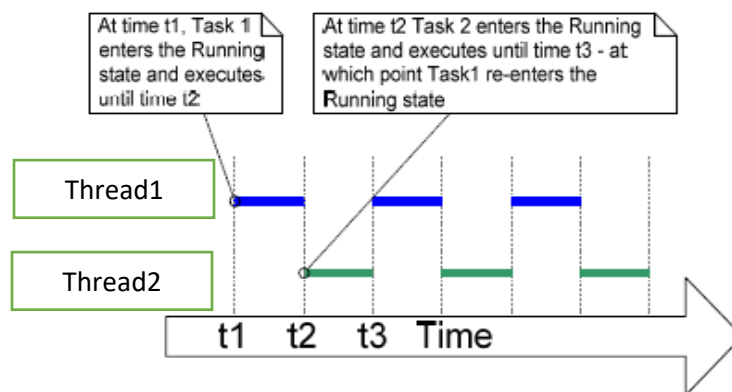
- g. Click OK to start debugging.
- h. The Tera Term window will display the following strings to show that thread 1 and 2 with the same priority are running in the round robin manner.



Summary

This demonstrates the steps of creating two simple threads and then starting threads executing. The threads simply print out a string periodically, using a null loop to create a period delay. Both threads functions are identical except for the string that they print.

Both Thread1 and Thread2 are running at the same priority, and so share CPU time on the single processor. Their actual execution pattern should be as the following.



The arrow along the bottom of the above figure shows the passing of time from time t1 onwards. The colored lines show which thread is running at each point in time.

Exercise: It is also possible to create a thread within another thread. So, let's try to create Thread1 from main(), and then create Thread2 from within Thread1.

- Move the statement of creating Thread2Func instance in main() into Thread2 immediately before the infinite *for(;;)* loop.
- Rebuild and execute the project.

```
142      /* creation of Thread2 */
143      Thread2Handle = osThreadNew(Thread2Func, NULL, &Thread2_attributes);
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256 /* USER CODE END Header_Thread1Func */
257 void Thread1Func(void *argument)
258 {
259     /* USER CODE BEGIN 5 */
260     volatile unsigned long ul;
261
262     /*
263      * Move the Thread2 instance creation to here.
264      * */
265
266     /* Infinite loop */
267     for(;;)
268     {
269         putsUSART1("\n\rThread 1 is running. \n\r");
270
271         for(ul = 0; ul < mainDELAY_LOOP_COUNT; ul++)
272         {}
273
274         //osDelay(1);
275     }
276     /* USER CODE END 5 */
277 }
```

Thus, Thread2 would not get created until the scheduler had been started. Please compare the output produced by the modified example with the one from the original example and check if they would be same.