République Tunisienne
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université de Tunis El Manar
École Nationale d'Ingénieurs de Tunis

Final Year Project II

Progress Report

Start Date: 01/01/2023

Students Names:

Iheb Kesraoui & Jaafar Hosni

Project Subject:

Thread_Metric benchmark tool on CMSIS RTOS v2 API

Name of hosting company:

STMicroelectronics

University Supervisor:

Mr. Tahar Ezzedine

ST Supervisor:

Mr. Haithem Rahmani

Du lundi 06 /02/2023 Au Dimanche 12/02/2023

1 **Researched and documented information about Real-Time Operating Systems (RTOS) and their importance in embedded systems. The research included various types of RTOS and their features and applications [1].(University Supervisor)**

2 **Learned about the concept of threads and how they can be used to achieve multitasking in RTOS. This involved understanding the basics of threads, their creation, and management in RTOS [2]. (Student)**

Exemple of creation of thread on FreeRtos :

```
void my_task(void *pvParameters)
{
    // Task code here
}

void create_task(void)
{
    xTaskCreate(my_task, "MyTask", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORIT
}
```

Figure 1: creation on freeRtos

Task Management:

```
void task_management(void)
{
    TaskHandle_t xHandle;
    xHandle = xTaskCreate(my_task, "MyTask", configMINIMAL_STACK_SIZE, NULL, tskIDLE_PRIORITY, NULL);

    if (xHandle == NULL) {
        // Error handling here
    }

    vTaskDelete(xHandle);
}
```

Figure 2: Task Management on FreeRtos

creation of thread on ThreadX :

```c
#define THREAD_STACK_SIZE 1024

TX_THREAD my_thread;

void my_thread_entry(ULONG thread_input)
{
    // Thread code here
}

void create_thread(void)
{
    tx_thread_create(&my_thread, "MyThread", my_thread_entry, 0,
        my_thread_stack, THREAD_STACK_SIZE, 1, 1, TX_NO_TIME_SLICE, TX_AUTO_START);
}
```

Figure 3: creation on threadX

Task Management:

```c
void thread_management(void)
{
    tx_thread_create(&my_thread, "MyThread", my_thread_entry, 0,
        my_thread_stack, THREAD_STACK_SIZE, 1, 1, TX_NO_TIME_SLICE, TX_AUTO_START);

    tx_thread_terminate(&my_thread);
}
```

Figure 4: Task Management on threadX

3 **Studied the basics of FreeRTOS and its features. This included understanding the architecture, tasks, and scheduling of FreeRTOS, and how to use it to develop real-time embedded applications [3].** **(Student)**

4 **Explored the work environment of STM32 microcontrollers, including their architecture, memory management, and peripherals [4].(Company Supervisor)**

5 **Learned about the different tools provided by STMicroelectronics for developing applications on STM32, such as STM32CubeIDE [5]and STM32CubeMX [6]. This involved understanding their features, functionality, and how to use them for developing applications on STM32 microcontrollers.(Student)**

1. Download and install the required tools:
   - STM32CubeIDE: An integrated development environment (IDE) that includes the STM32CubeMX software tool, as well as a C/C++ compiler, debugger, and other development tools.
   - STM32CubeMX: A graphical software configuration tool that allows you to generate initialization code for STM32 microcontrollers, configure the peripherals, and generate project files for various IDEs.
2. Create a new project in STM32CubeMX:
   - Open STM32CubeMX and create a new project.
   - Select the microcontroller you will be using, and configure the peripherals and other settings for your application.
   - Generate the initialization code and project files for your selected IDE, such as STM32CubeIDE.
3. Open the project in STM32CubeIDE:
   - Import the project files generated by STM32CubeMX into STM32CubeIDE.
   - Compile the code and check for any errors or warnings.
   - Debug the code by setting breakpoints, stepping through the code, and viewing variables and memory.
4. Write and test your code:
   - Write your code in the source files generated by STM32CubeMX or create new files.
   - Use the various development tools provided by STM32CubeIDE, such as the C/C++ compiler, debugger, and libraries, to develop and test your application.
5. Flash the firmware onto the microcontroller:
   - Connect your STM32 microcontroller to your computer using a programmer or debugger.
   - Build the project in STM32CubeIDE to generate a binary file.
   - Use a flashing tool, such as ST-Link, to flash the binary file onto the microcontroller.

Figure 5: steps for using the enviroment

6 **Studied the basics of using STM32CubeIDE and STM32CubeMX for developing applications on STM32 microcontrollers [Link to the first Project that we've started] . This involved understanding the workflow, configuration, and code generation using these tools.(Company Supervisor)**

Du lundi 13 /02/2023 Au Dimanche 19/02/2023

1. **Started implementing ThreadX in our project using STM32cubeMX.**
   - **Generate code: STM32CubeMX can generate code for the ThreadX RTOS that is compatible with STM32 microcontrollers, saving significant time compared to writing the code from scratch.**
   - **Configure ThreadX: The intuitive graphical user interface (GUI) in STM32CubeMX makes it easy to configure ThreadX, including setting the number of threads, priorities, and stack sizes.**
   - **Integrate with STM32CubeIDE: STM32CubeMX is fully integrated with STM32CubeIDE, making it easy to import the generated code into the IDE and start programming the application, with access to debugging and profiling tools to optimize performance.**
2. **Successfully tested all ThreadX functions and verified their output**

   some code example that we successfully run :

```
**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 30
Time Period Total:  715464

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 60
Time Period Total:  715457

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 90
Time Period Total:  715455

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 120
Time Period Total:  715456

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 150
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 180
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 210
Time Period Total:  715454

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 240
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 270
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 300
Time Period Total:  715454

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 330
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 360
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 390
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 420
Time Period Total:  715454
```

Figure 6: $\mathrm{tm}_{b}asic_{p}rocessing_{t}est$

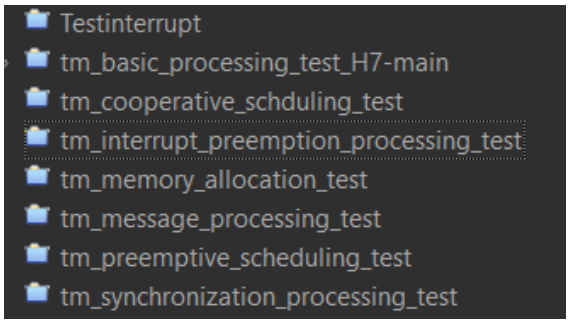

Figure 7: all the function that we tested

3. **Worked on printing the results of the ThreadX functions to the console for better visibility and debugging on STM32CubeIDE**

- **Set up the debug environment in STM32CubeIDE: This involves configuring the debug settings and connecting the microcontroller to the IDE. You will need to specify the communication protocol (e.g., ST-LINK, J-Link, etc.) and select the correct target device.**

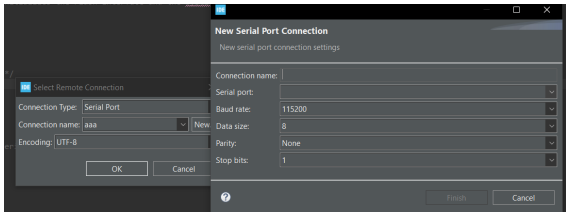

Figure 8: UART

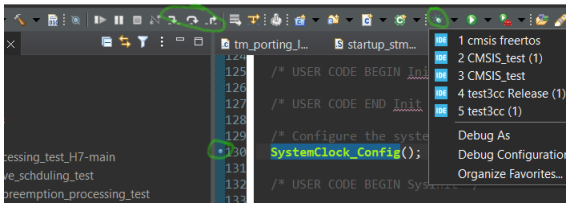

Figure 9: Debug session

- **Modify the ThreadX source code: Add code to the ThreadX functions to print the results to the console. This can be done using printf statements or other debugging tools provided by the IDE.**
- **Build and run the application: After modifying the code, build the application and run it on the microcontroller. Use the console window in STM32CubeIDE to view the output of the ThreadX functions and debug any issues that arise.**

4. **Explained the importance of each function and how they contribute to the functionality of the system [Link to the drive Link]**

5. **Explored the different features of Github, such as creating branches, merging changes, and resolving conflicts (could you share with us your github account cause the repositorie is private)[Link to the github repositories]**

   - **Clone the repository : "git clone [Link to the github repositories]"**
   - **Once the repository is cloned, navigate to the project directory: cd repository**
   - **Create a new project in your preferred development environment, such as Visual Studio Code or Eclipse**
   - **Make your changes to the project and save your changes.**

- **Stage your changes for commit using the git add command**
- **Commit your changes using the git commit command, along with a message describing your changes : git commit -m "Added new feature to project"**
- **Push your changes back to GitHub using the git push command : git push origin main**

Du lundi 20 /02/2023 Au Dimanche 26/02/2023

1. **Create a new project using STM32CubeMX and STM32CubeIDE and intialize ThreadX on it and prepare CMSIS API on top of it :**

   - **Configure our project to use the CMSIS API by adding the necessary header files and linking to the appropriate libraries.**

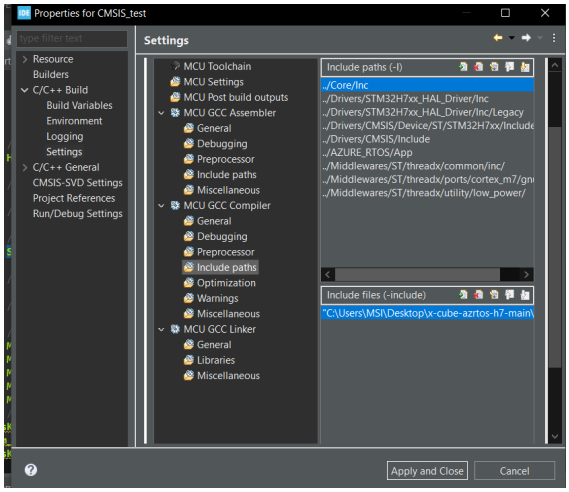

Figure 10: add the libraries

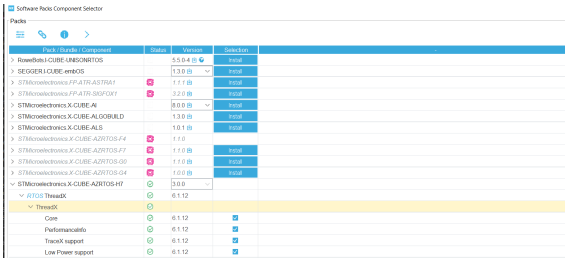

Figure 11: add the threadX in the project

   - **Add the necessary drivers from the github repository to our project.**

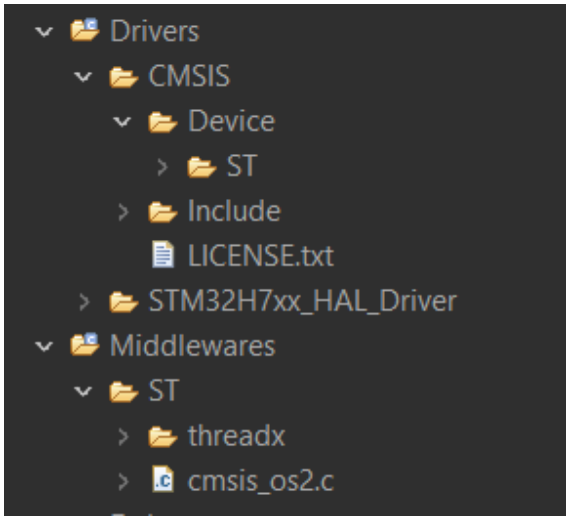

Figure 12: add the drivers of CMSIS

2. **Implement the porting layer of the thread metric tests by using the ThreadX API to map its functions to the corresponding functions in the CMSIS V2 API**

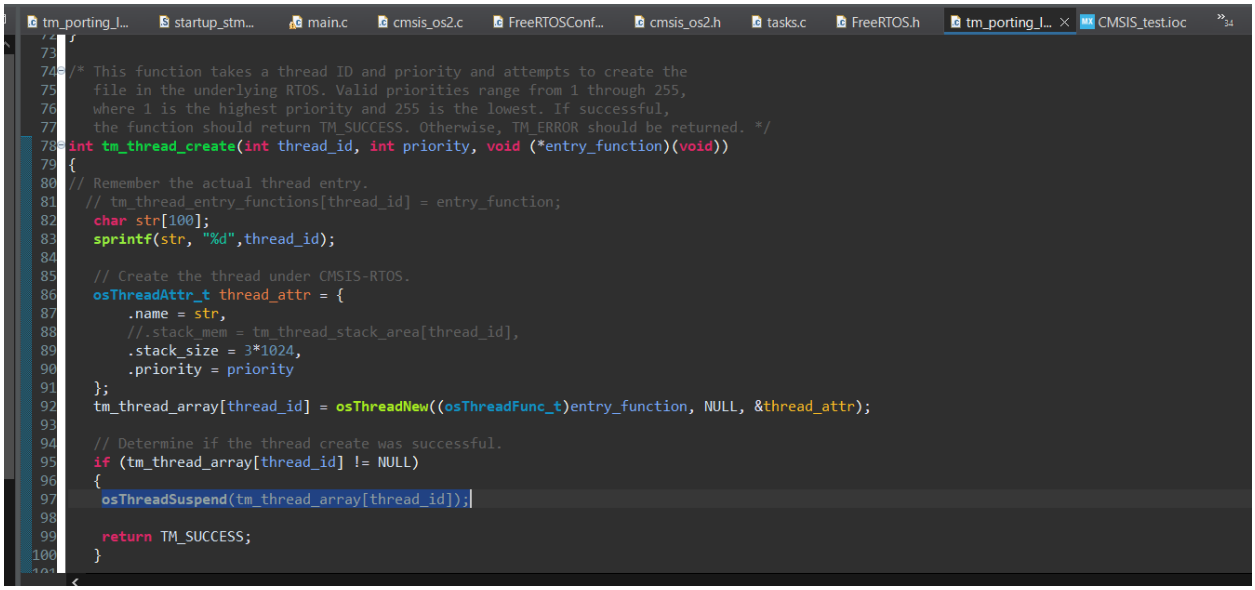

Figure 13: exemple of the thread creation on CMSIS

3. **Fail to run the tests (the problem is that the thread was always null)**

4. **In order to test the porting layer we initialize a new project using CMSIS V2 API on top of the FreeRtos that is supported by the ST environments**

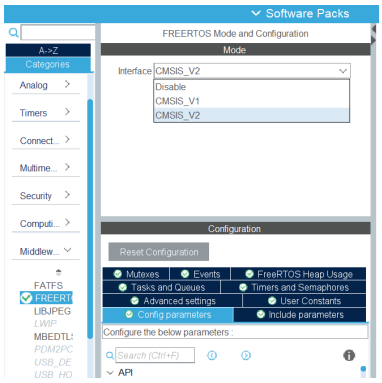

Figure 14: use STM32cubeMX to include the cmsis api on freertos

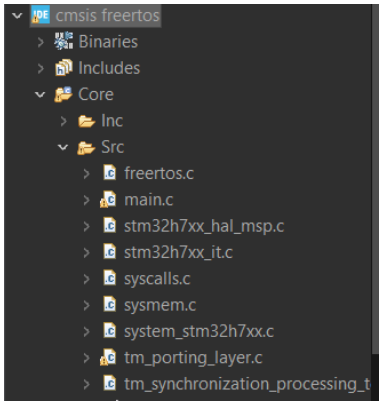

Figure 15: this is the project

5. successfully running all the test and proving the success of the porting layer

```
**** Thread-Metric Memory Allocation Test **** Relative Time: 30
Time Period Total:  15782788

**** Thread-Metric Memory Allocation Test **** Relative Time: 60
Time Period Total:  15782603

**** Thread-Metric Memory Allocation Test **** Relative Time: 90
Time Period Total:  15782562

**** Thread-Metric Memory Allocation Test **** Relative Time: 120
Time Period Total:  15782562

**** Thread-Metric Memory Allocation Test **** Relative Time: 150
Time Period Total:  15782516

**** Thread-Metric Memory Allocation Test **** Relative Time: 180
Time Period Total:  15782516

**** Thread-Metric Memory Allocation Test **** Relative Time: 210
Time Period Total:  15782516

**** Thread-Metric Memory Allocation Test **** Relative Time: 240
Time Period Total:  15782516

**** Thread-Metric Memory Allocation Test **** Relative Time: 270
Time Period Total:  15782516

**** Thread-Metric Memory Allocation Test **** Relative Time: 300
Time Period Total:  15782516

**** Thread-Metric Memory Allocation Test **** Relative Time: 330
Time Period Total:  15782517

**** Thread-Metric Memory Allocation Test **** Relative Time: 360
Time Period Total:  15782516

**** Thread-Metric Memory Allocation Test **** Relative Time: 390
Time Period Total:  15782516
```

Figure 16: an exemple of tm basic processing layer running on freertos

6. Planning a meeting with the company supervisor to discuss the result and figure out the problem with the CMSIS wrapper on top of threadX

Du lundi 27 /02/2023 Au Dimanche 05/03/2023

1. Assembly code and memory allocation problem:

- **We encountered an issue with the assembly code and memory allocation.**
- **After analyzing the code and debugging, the problem was identified and solved.**
- **The solution involved updating the assembly code and making adjustments to the memory allocation.**

```
  /* User_heap_stack section, used to check that there is enough RAM left */
  ._user_heap_stack :
  {
    . = ALIGN(8);
    PROVIDE ( end = . );
    PROVIDE ( _end = . );
    . = . + _Min_Heap_Size;
    . = . + _Min_Stack_Size;
    . = ALIGN(8);
  } >DTCMRAM

  /* Remove information from the standard libraries */
  /DISCARD/ :
  {
    libc.a ( * )
    libm.a ( * )
    libgcc.a ( * )
  }

  .ARM.attributes 0 : { *(.ARM.attributes) }
}
```

Figure 17: assembly code

```
#define RTOS2_BYTE_POOL_STACK_SIZE              32 * 1024

/* define the CMSIS RTOS2 heap size */

#define RTOS2_BYTE_POOL_HEAP_SIZE               32 * 1024

/* USER CODE BEGIN 2 */
/* This define defines the type of memory allocation. */
#define USE_MEMORY_POOL_ALLOCATION
/*#define USE_MEMORY_POOL_ALLOCATION*/
/* USER CODE END 2 */

#endif
```

Figure 18: we change the memory so we can create the thread

2. Test execution using CMSIS ThreadX API:

- **All the tests are now running on the CMSIS ThreadX API.**

```
**** Thread-Metric Synchronization Processing Test **** Relative Time: 30
Time Period Total:  28780797

**** Thread-Metric Synchronization Processing Test **** Relative Time: 60
Time Period Total:  28780891

**** Thread-Metric Synchronization Processing Test **** Relative Time: 90
Time Period Total:  28780753

**** Thread-Metric Synchronization Processing Test **** Relative Time: 120
Time Period Total:  28780701

**** Thread-Metric Synchronization Processing Test **** Relative Time: 150
Time Period Total:  28780618

**** Thread-Metric Synchronization Processing Test **** Relative Time: 180
Time Period Total:  28780646

**** Thread-Metric Synchronization Processing Test **** Relative Time: 210
Time Period Total:  28780646

**** Thread-Metric Synchronization Processing Test **** Relative Time: 240
Time Period Total:  28780646

**** Thread-Metric Synchronization Processing Test **** Relative Time: 270
Time Period Total:  28780646

**** Thread-Metric Synchronization Processing Test **** Relative Time: 300
Time Period Total:  28780893

**** Thread-Metric Synchronization Processing Test **** Relative Time: 330
Time Period Total:  28780646

**** Thread-Metric Synchronization Processing Test **** Relative Time: 360
Time Period Total:  28780899
```

Figure 19: cooperative test

```
**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 30
Time Period Total:  715464

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 60
Time Period Total:  715457

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 90
Time Period Total:  715455

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 120
Time Period Total:  715456

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 150
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 180
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 210
Time Period Total:  715454

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 240
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 270
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 300
Time Period Total:  715454

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 330
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 360
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 390
Time Period Total:  715453

**** Thread-Metric Basic Single Thread Processing Test **** Relative Time: 420
Time Period Total:  715454
```

Figure 20: tm basi processing test

- **We successfully implemented the API to enable thread management and multi-tasking.**
- **This will improve the performance and efficiency of the system.**

3. Organizing test results:

- **We has started organizing the test results in an Excel file.**
- **This will allow for easy tracking and analysis of the results.**
- **The Excel file includes information such as operating system used, test type, and test outcome.**

Time Period	**Nucleo-H7**		
	CMSIS/FreeRtos	CMSIS/ThreadX	ThreadX Native
tm_basic_processing_test	715 457	748 693	153 729
tm_cooperative_schduling_test	65 227 095/Error	60 377 697	21 511 547
tm_interrupt_preemption_processing_test			
tm_interrupt_processing_test			
tm_memory_allocation_test	15 782 606		24 084 639
tm_message_processing_test		Error	11 870 716
tm_preemptive_scheduling_test	Error	Error	7 809 798
tm_synchronization_processing_test	28 780 780	120 398 283	33 956 054

Figure 21: the benchmark

Students:	Iheb Kesraoui & Jaafar Hosni
ST Supervisor:	Mr. Haithem Rahmani
University Supervisor:	Mr. Tahar Ezzedine

References

References

[1] RTOS

[2] thread

[3] FreeRTOS

[4] STMicroelectronics

[5] Link to download STM32cubeIDE

[6] Link to download STM32cubeMX

[7] Link to the first Project that we've started