PX5 [RTOS]

Porting Guide – IAR STM32 Cortex M Series

Version 5.1.0

Published November 15, 2023

Table of Contents

Chapter

1

PX5 RTOS - IAR - porting guide for STM32 Evaluation Kits - Overview

PX5 RTOS samples are available for several STM32 evaluation kits, but if you need to port it to a different one for which there is no sample, this document describes the process to achieve that using IAR EWARM and resources which can be obtained directly from the PX5 RTOS website and GitHub.

While this document shows the steps for the porting process, if you need more details you can refer to the PX5 RTOS Binding User Guide.

Note that this porting guide refers to pre-built object code versions of px5.c and px5_binding.s (px5.o and px5_binding.o). Furthermore, the evaluation is limited to a maximum of 10 threads. Once the limit is reached, an EINVAL error code is returned from the pthread_create API. If a full source code evaluation is required, please contact PX5 at sales@px5rtos.com.

Chapter 2

Porting Steps

This chapter describes the process of porting PX5 RTOS to an STM32 Cortex M microcontroller using the IAR EWARM development tool.

An official sample published by STMicro on their website will be the base for the porting exercise. Although you can also start by leveraging STM32CubeIDE, this approach will require several additional steps modifying files to work with IAR.

Step 1: Get a basic IAR working sample from STMicro GitHub page

For this guide, we'll be using the STM32C031C6 NUCLEO, but the process is similar to other Cortex M MCUs.

Visit STMicro GitHub page and clone this repository or download the files following instructions found here:

https://github.com/STMicroelectronics/STM32CubeC0/tree/main#how-to-use

It is crucial to follow the steps required, specifically the one related to submodules needed for this project.

After obtaining the files, open the IAR project (*project.eww*) file from the path:

Projects/NUCLEO-C031C6/Examples/CORTEX/CORTEXM_SysTick/EWARM.

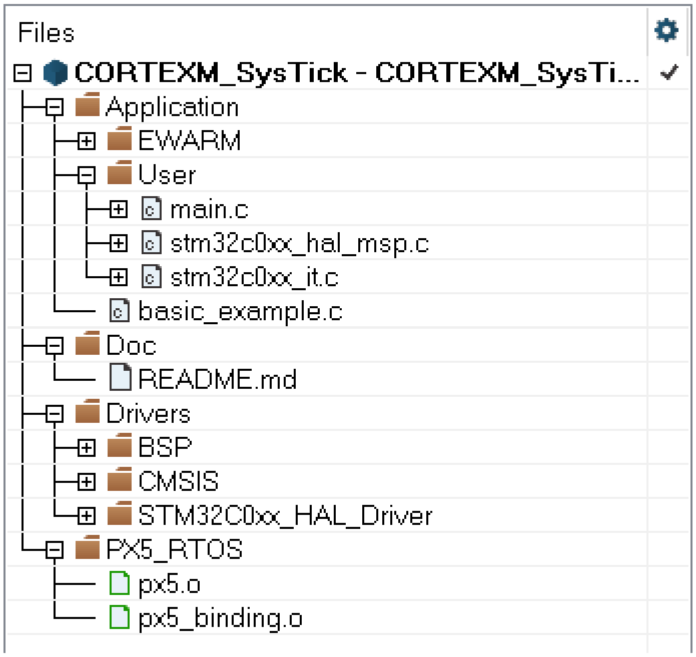
Step 2: Add PX5 RTOS source code

For this step you'll need to have access to PX5 RTOS files, which you can obtain from any sample on the PX5 RTOS website with a compatible architecture (like Cortex M0 sample if you're using a Cortex M0 MCU).

Add a group named PX5_RTOS to your project and then *add files*. Select both px5.o and px5_bindings.o files.

Add a sample file to the project, under *Application*. In this guide we'll refer to the basic sample, but feel free to use any other sample you prefer. Select the *basic_example.c* file.

Your project should now look like the following:



```
Files
⊟ CORTEXM_SysTick - CORTEXM_SysTi...                ✓
├⊟ Application
│ ├⊞ EWARM
│ ├⊟ User
│ │ ├⊞ main.c
│ │ ├⊞ stm32c0xx_hal_msp.c
│ │ └⊞ stm32c0xx_it.c
│ └── basic_example.c
├⊟ Doc
│ └── README.md
├⊟ Drivers
│ ├⊞ BSP
│ ├⊞ CMSIS
│ └⊞ STM32C0xx_HAL_Driver
└⊟ PX5_RTOS
  ├── px5.o
  └── px5_binding.o
```

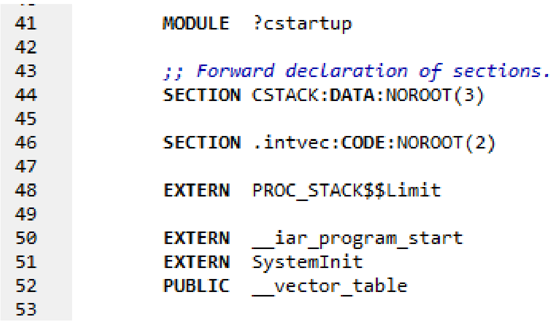
Step 3: Modify startup file

The start-up file requires a modification so the application will use the process stack. Open the startup file under **Application/EWARM/startup_stm32c031xx.s**

Find the definition for *EXTERN __iar_program_start,* which is likely on line 48, and add the following before it:

```
EXTERN   PROC_STACK$$Limit
```

This part of the startup file will look like this after the change:

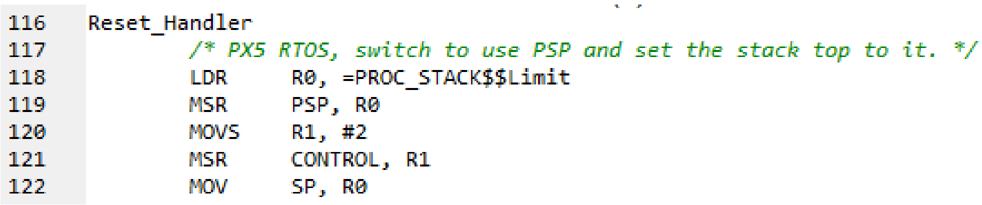


```
41          MODULE  ?cstartup
42
43          ;; Forward declaration of sections.
44          SECTION CSTACK:DATA:NOROOT(3)
45
46          SECTION .intvec:CODE:NOROOT(2)
47
48          EXTERN  PROC_STACK$$Limit
49
50          EXTERN  __iar_program_start
51          EXTERN  SystemInit
52          PUBLIC  __vector_table
53
```

Still on this file, find the Reset_handler and add the following to it:

```
  /* PX5 RTOS, switch to use PSP and set the stack top
to it. */

     LDR      R0, =PROC_STACK$$Limit

     MSR      PSP, R0

     MOVS     R1, #2

     MSR      CONTROL, R1

     MOV      SP, R0
```

After the change this section of the file should look like the following:



```
Reset_Handler
        /* PX5 RTOS, switch to use PSP and set the stack top to it. */
        LDR     R0, =PROC_STACK$$Limit
        MSR     PSP, R0
        MOVS    R1, #2
        MSR     CONTROL, R1
        MOV     SP, R0
```

Step 4: Modify linker file

The linker file requires modifications so the application will use the process stack. First, we need to add it to the project - ***stm32c031xx_flash.icf,*** which is in the root of the sample folder. Then open it.

Find the *define symbol __ICFEDIT_size_cstack__ = 0x400;* which is likely on line 13, and add the following right after it:

```
define symbol __ICFEDIT_size_proc_stack__ = 0x400;
```

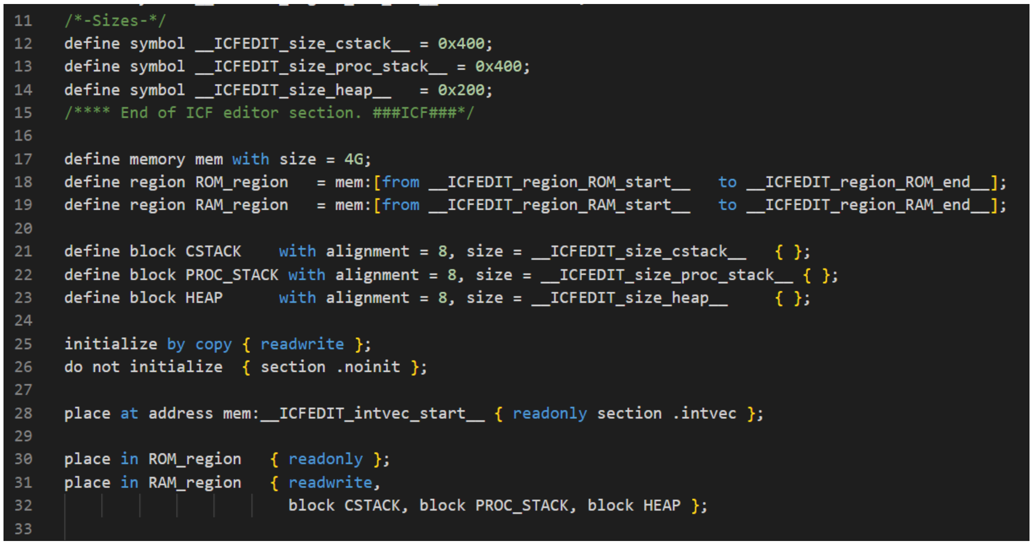
Then, add the following right after the *define block CSTACK*:

```
    define block PROC_STACK with alignment = 8, size =
    __ICFEDIT_size_proc_stack__ { };
```

Finally, modify the *place in RAM_region*, with the following:

```
place in RAM_region { readwrite,

       block CSTACK, block PROC_STACK, block HEAP };
```

After the change this section of the file should look like the following:



```
11   /*-Sizes-*/
12   define symbol __ICFEDIT_size_cstack__ = 0x400;
13   define symbol __ICFEDIT_size_proc_stack__ = 0x400;
14   define symbol __ICFEDIT_size_heap__   = 0x200;
15   /**** End of ICF editor section. ###ICF###*/
16
17   define memory mem with size = 4G;
18   define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
19   define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];
20
21   define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
22   define block PROC_STACK with alignment = 8, size = __ICFEDIT_size_proc_stack__ { };
23   define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };
24
25   initialize by copy { readwrite };
26   do not initialize  { section .noinit };
27
28   place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };
29
30   place in ROM_region   { readonly };
31   place in RAM_region   { readwrite,
32                           block CSTACK, block PROC_STACK, block HEAP };
33
```

Step 5: Modify main.c file

Since the example file already contains a main function, we need to rename the existing one. Open the *main.c* file under Application.

Rename the main function call from *int main(void)* to *void platform_setup(void)*

Then, remove the while loop from the same function.

Step 6: Modify the ISR file

Some changes are required in order to provide a single, periodic timer interrupt to drive all of PX5 RTOS time related services. Open the ISR file *stm32c0xx_it.c* under Application.

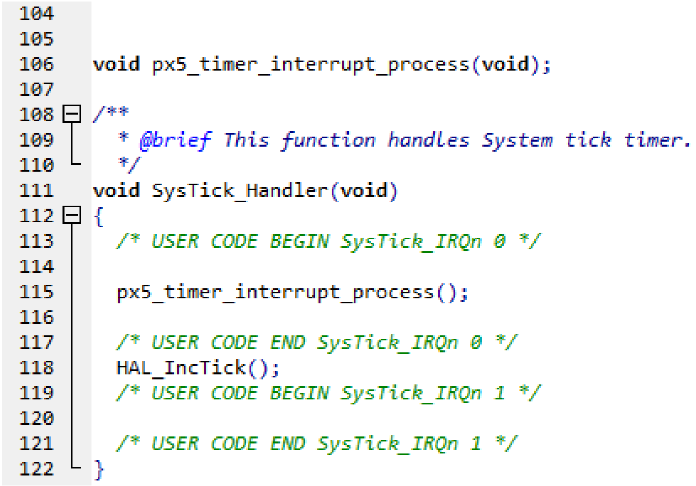
Look for the *SysTick_Handler* function and add the following function call:

```
px5_timer_interrupt_process();
```

Add the function declaration before the *SysTick_Handler* function:

```
void px5_timer_interrupt_process(void);
```

This part of the ISR file should look like the following after the modification:



```


void px5_timer_interrupt_process(void);

/**
  * @brief This function handles System tick timer.
  */
void SysTick_Handler(void)
{
  /* USER CODE BEGIN SysTick_IRQn 0 */

  px5_timer_interrupt_process();

  /* USER CODE END SysTick_IRQn 0 */
  HAL_IncTick();
  /* USER CODE BEGIN SysTick_IRQn 1 */

  /* USER CODE END SysTick_IRQn 1 */
}
```

Still on this file, comment out or remove these two functions to avoid duplicate declaration:

- `void PendSV_handler(void)`
- `void SVC_Handler(void)`

Step 7: Add header files to your IAR project

Right click the project name and select *options*.

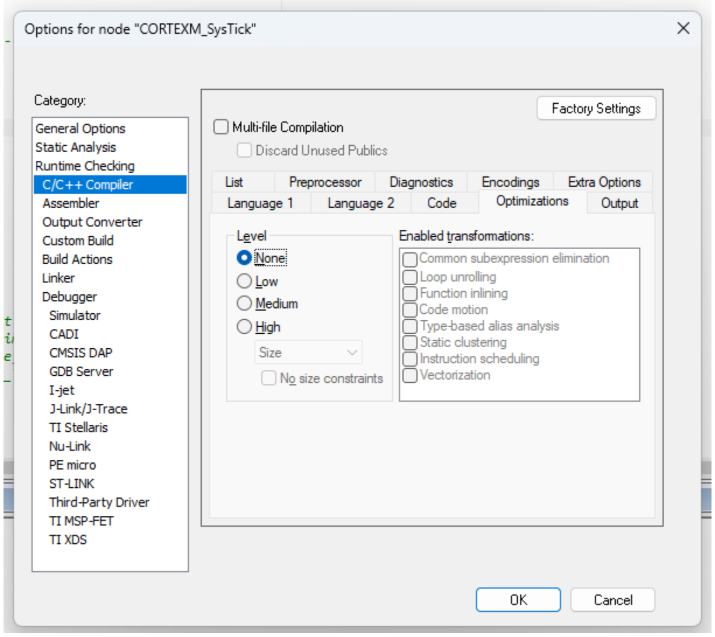
Under *C/C++ Compiler / Preprocessor / Additional include directories* add the following directories:

1. *[PX5_RTOS-Sample-Folder]\source*

After header files, update *Defined Symbols* with the information below:

PX5_EVALUATION

Attention: You might also need to change the option for compiler optimization which might remove variables like the thread_counter from your sample. To do so, right click the project name, select C/C++ Compiler, then Optimizations and click None under Level.



Options for node "CORTEXM_SysTick"

Category: C/C++ Compiler

□ Multi-file Compilation

□ Discard Unused Publics

Level:
● None
○ Low
○ Medium
○ High

Enabled transformations:
□ Common subexpression elimination
□ Loop unrolling
□ Function inlining
□ Code motion
□ Type-based alias analysis
□ Static clustering
□ Instruction scheduling
□ Vectorization

Your project is ready to be built and tested on your target device.

Enhance • Simplify • Unite

11440 West Bernardo Court • Suite 300
San Diego, CA 92127, USA

Phone: +1 (858) 753-1715
Website: px5rtos.com