AN201701

STM32CubeMX と µC3/Compact の共存方法

Rev.1.0

September 29, 2017

改訂記録

Rev.	発行日	改訂内容	
		ページ	内容
1.0	2017.09.29	―	初版

目次

1. 概要

本ドキュメントは STM32CubeMX と μC3/Compact の共存方法を示すものです。ST マイクロエレクトロニクスの提供する STM32CubeMX は、ST マイクロエレクトロニクス製マイコンのデバイス周辺機能の設定を簡単に行うことができるツールです。また μC3/Compact を用いると RTOS の機能を各種マイコンに実装することができます。

STM32CubeMX と μC3/Compact を共存させることにより以下のメリットが得られます。

- μITRON 仕様のシステムコールが使えるようになります。これによりタスク管理、時間管理、周期ハンドラ等の機能が実装され、CPU 負荷低減、間欠的な処理の作成、処理の並列化等が行えるようになります。
- コンフィグレータを用いることで複雑なリアルタイム OS の機能をグラフィカルかつ容易に設定することができます。

本ドキュメントではまず、STM32CubeMX が生成したファイルと μC3/Compact のコンフィグレータが生成したファイルからプロジェクトを構築する方法を示します。その後、具体的な各サンプルプログラムの作成手順を説明します。

使用した動作確認環境

使用した動作確認環境を以下に示します。

表 1.1 使用した動作確認環境

項目	バージョン
μC3 パッケージ	μC3/Compact Cortex-M4 STM32F4 シリーズ EWARM 版 Release 2.1.9
STM32CubeMX	Version 4.21.0
コンパイラ	IAR Embedded Workbench for ARM 8.1.0.12863
評価ボード	STM32F4 Discovery kit for STM32F469 MCU

作成するサンプルプログラム

本ドキュメントで作成するサンプルプログラムを以下に示します。

表 1.2 作成するサンプルプログラム

章	内容
2 章	LD1 が消灯時に USER ボタンを押すと点灯し、点灯時にボタンを押すと消灯するプログラム。
3 章	USER ボタンを押すと PC 上のマウスポインタが移動するプログラム。 ※サンプルの一部に、ST マイクロエレクトロニクスの提供するソフトウェア STM32CubeF4 の USB Device のサンプルを使用しました。 ダウンロード URL: http://www.st.com/ja/embedded-software/stm32cubef4.html 参照元: STM32Cube_FW_F4_V1.16.0¥Projects¥STM32469I-Discovery¥Applications¥USB_Device¥HID_Standalone

---

TRON は"The Real-time Operation system Nucleus"の略称です。
μITRON は"Micro Industrial TRON"の略称です。
μC3 はイー・フォース株式会社の登録商標です。
STM32 は、ST マイクロエレクトロニクスの登録商標です。
その他、本資料中の製品名やサービス名は全てそれぞれの所有者に属する商標または登録商標です。
本書で記載されている内容は予告無く変更する場合があります。
STM32CubeMX の設定の詳細については STM32CubeMX のマニュアル等をご確認ください。

---

2. STM32CubeMX と μC3/Compact 共存プロジェクト構築方法

STM32CubeMX と μC3/Compact を共存させたプロジェクトの構築フローは次のようになります。

① STM32CubeMX を用いてデバイスのハードウェアに関する設定を行いスケルトンコードを生成。
② μC3/Compact のコンフィグレータを用いて RTOS に関する設定を行いスケルトンコードを生成。
③ 統合開発環境のプロジェクトを作成し、コンパイル、リンク、デバッグに関するオプション設定を行う。
④ STM32CubeMX 生成ソースコードと μC3/Compact 生成ソースコードの競合部分の調整のためソースコードの修正を行う。

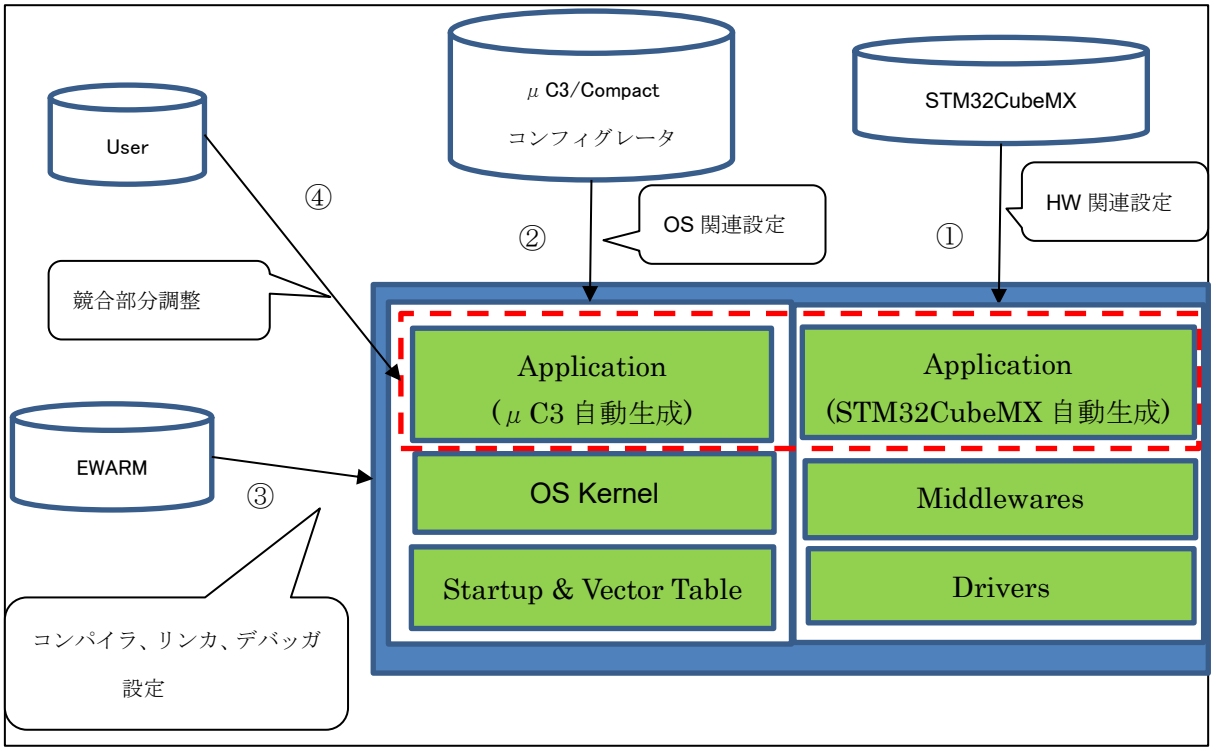


図 2. 1 STM32CubeMX と μC3/Compact を共存させたプロジェクトの構築フロー

①において STM32CubeMX 上で割り込みサービスルーチンの設定を行う際の注意点として Preemption Priority の 0（最も高い優先レベル）は、μC3/Compact のカーネルが占有することになるため、その他の割り込みに対しては Preemption Priority を 1 より大きい値とする必要があります。

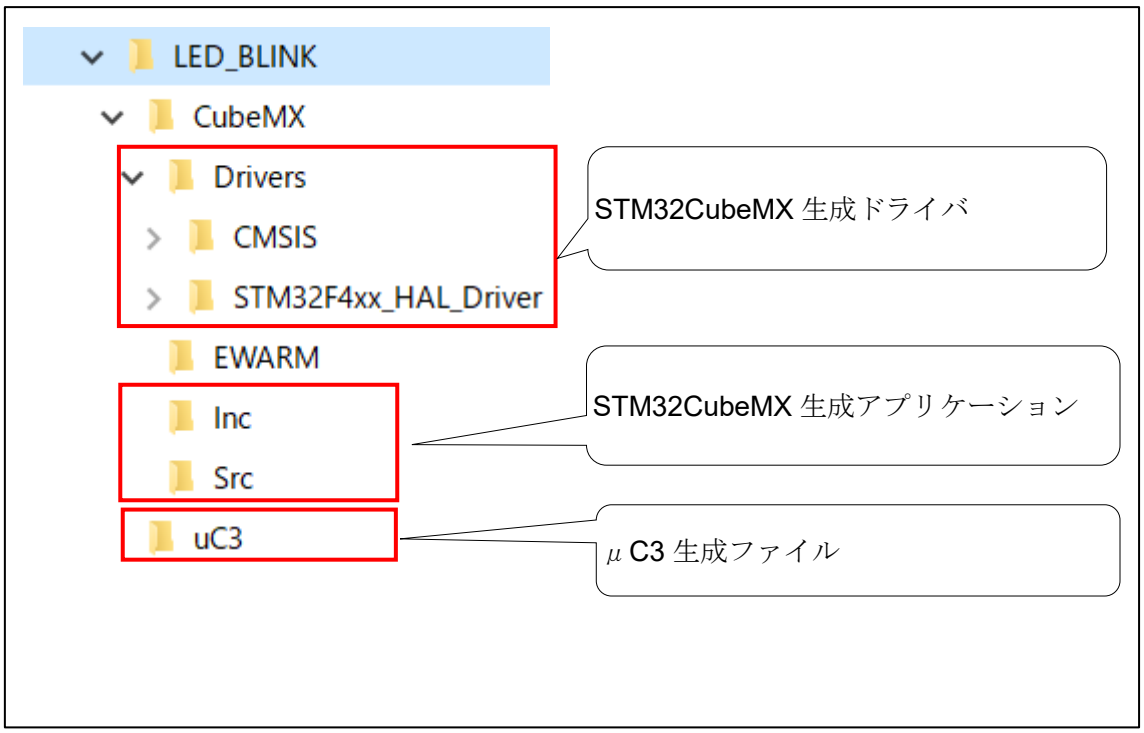
④でユーザーが自動生成されたコードを、エディタで修正する箇所は
・スタートアップルーチン(μC3 が生成する prst.s79)
・割り込みサービスルーチンとタスク(μC3 が生成する main.c)
・割り込み優先度初期化処理(μC3 が生成する hw_init.c)
等です。

3. サンプルプログラム(ボタン押下→LED 点滅)作成手順

この章では、「LD1 が消灯時に USER ボタンを押すと点灯し、点灯時にボタンを押すと消灯する」プログラムの作成手順を説明します。

3.1. フォルダ構成

フォルダ構成を下記に示します。



次節から具体的な手順を示していきます。

3.2. STM32CubeMX を用いたプロジェクト作成

3.2.1. 新規プロジェクトの作成

STM32CubeMX を起動して、New Project を選択します。

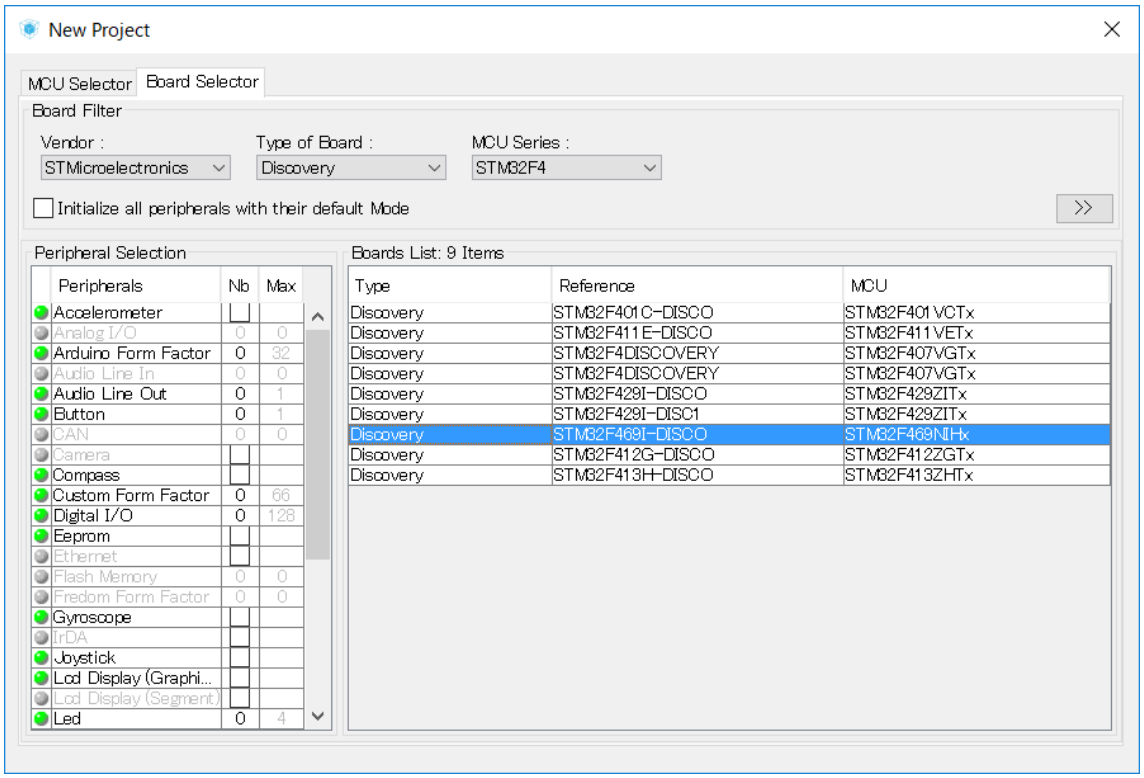
Board Selector タブを選択し、

Type of Board->Discovery

MCU Series->STM32F4

Reference->STM32F469I-DISCO

を選択し、ダブルクリックします。

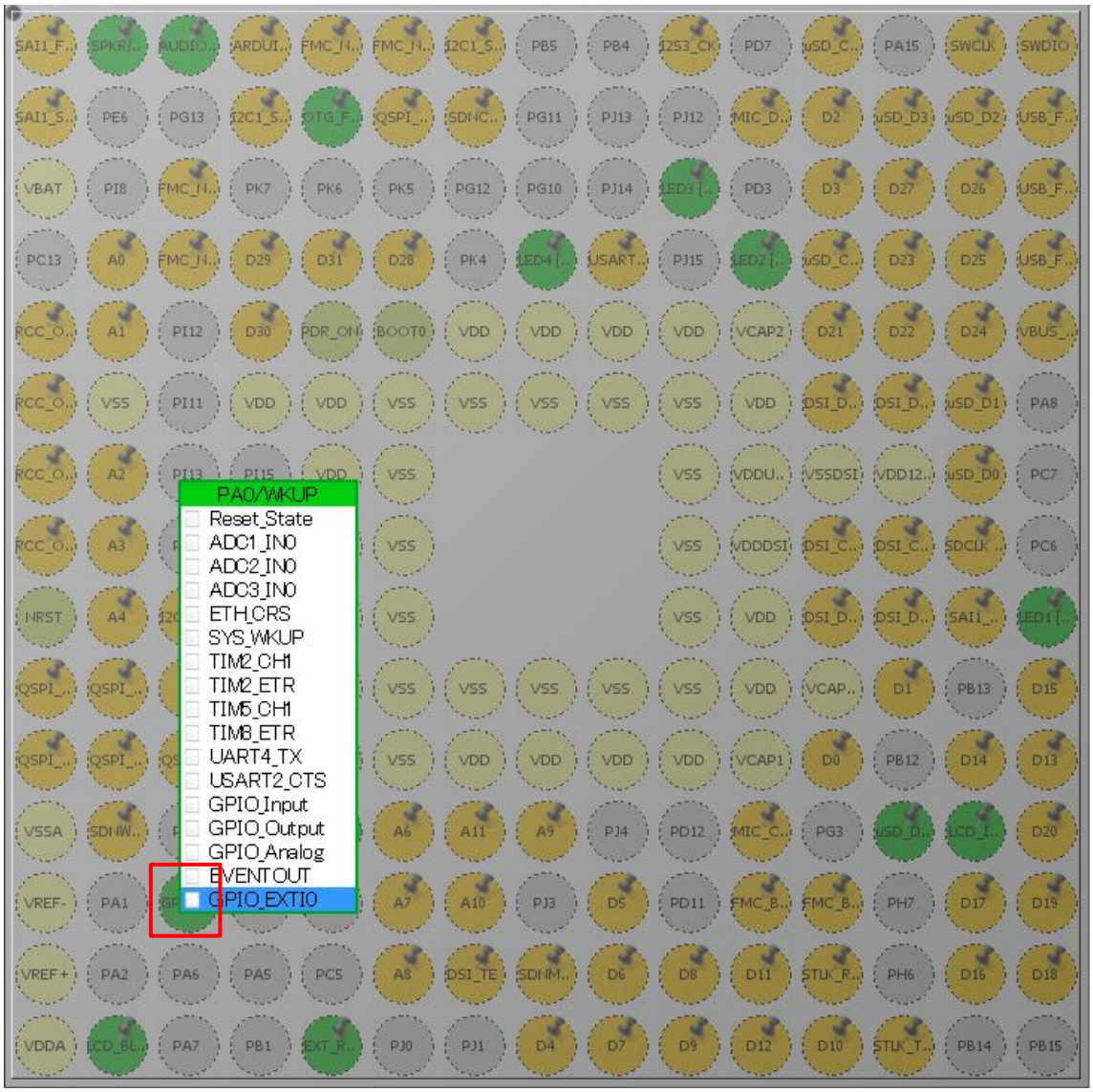


3.2.2. **Pinout 設定**

Pinout タブを選択し以下のように設定します。

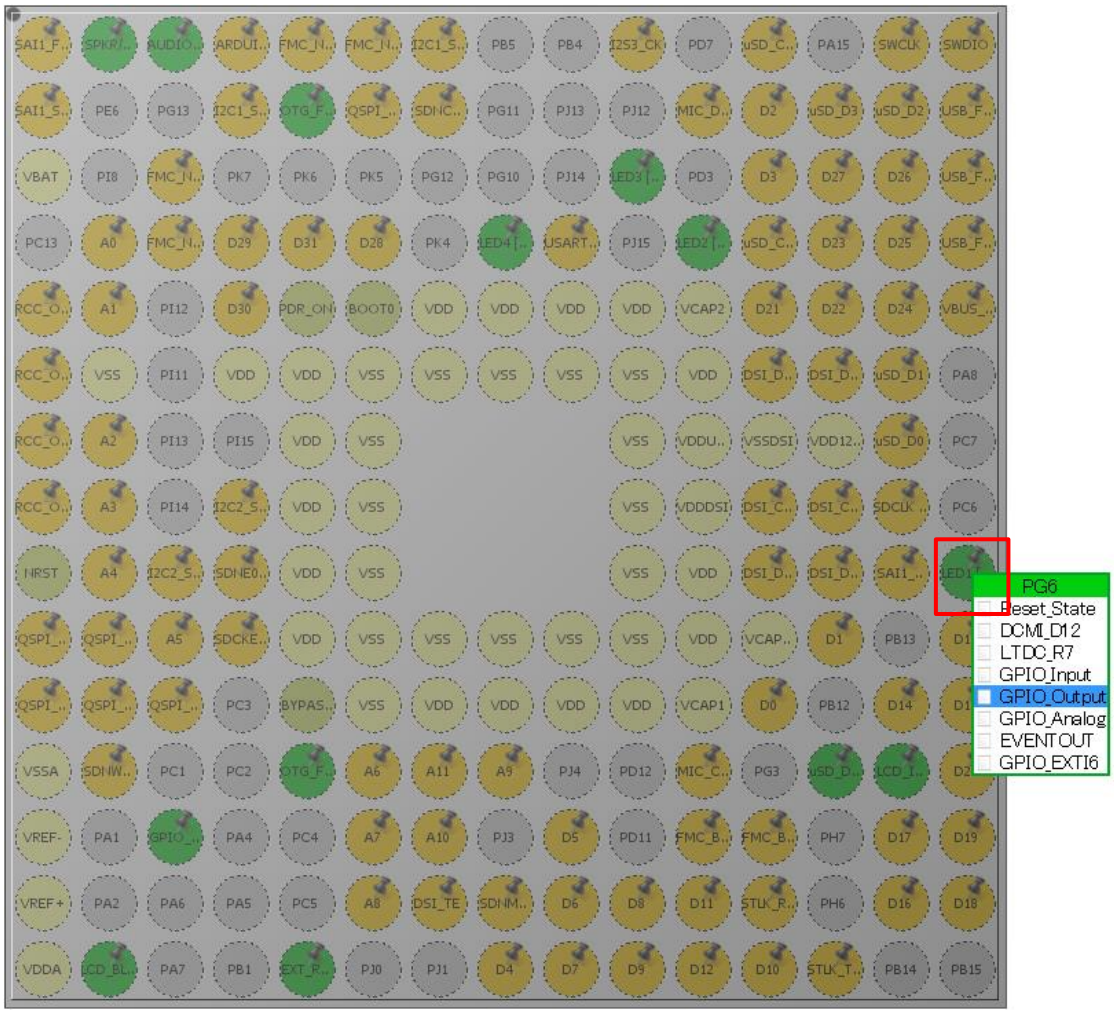
USER ボタン押下を検知する割り込みの設定

PA0/WKUP を GPIO_EXTI0 に設定します。



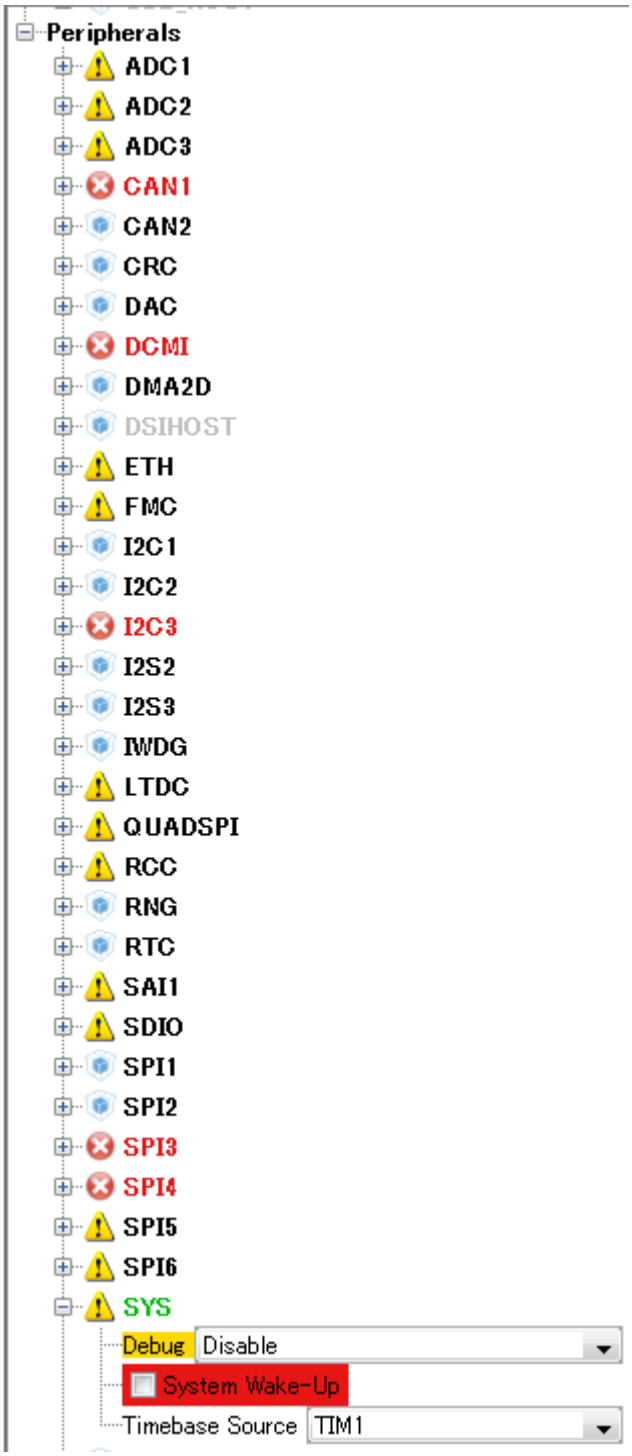
LD1 を点灯させるための設定

PG6 の設定が GPIO_Output となっていることを確認します。



RTOS を動かすための設定

左ペインの Configuration->Peripherals->SYS の設定で TimeBase Source を TIM1 と設定します。



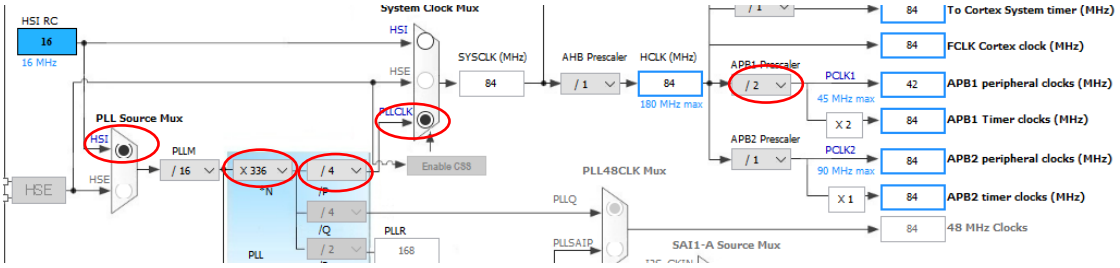
3.2.3. クロック設定

Clock Configuration タブを選択し以下のように設定します。

表 3.1 クロック設定

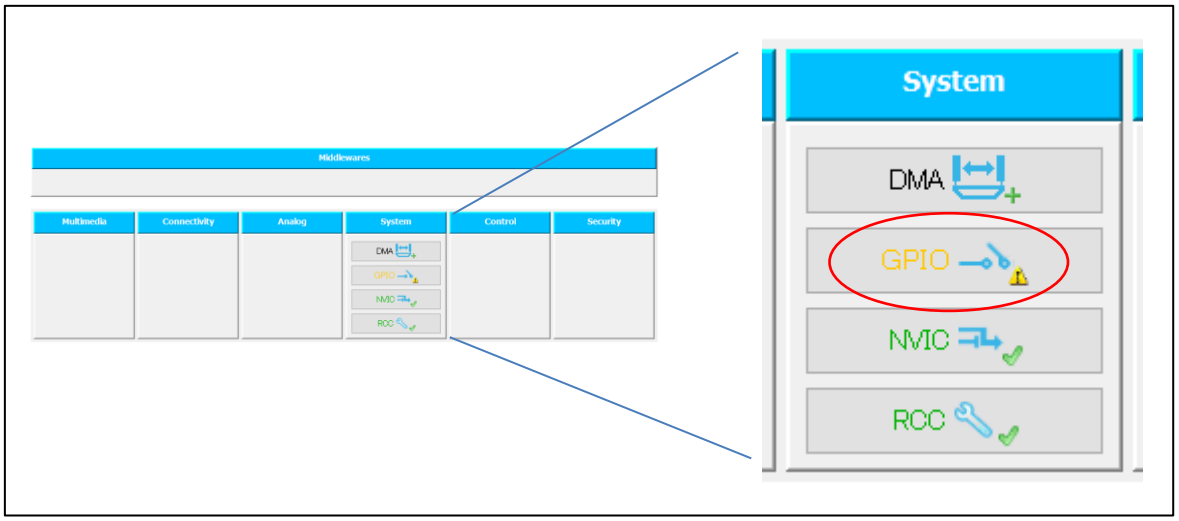
項目	設定
System CLK Mux	PLLCLK
PLL source	HSI
PLL N	336
PLL P	4
APB1 Prescaler	2

※STM32CubeMX 上で背景赤色表示されている項目は、設定変更が必要な項目です。



3.2.4. **GPIO 設定**

次に、Configuration タブの GPIO ボタンを選択します。



PA0/WKUP と PG6 の設定を確認します(PG6 は GPIO mode と Maximaum output speed を既定値から変更しています)。

表 3. 2 PA0/WKUP 設定

項目	設定
GPIO mode	External Interrupt Mode with Rising edge trigger detection
GPIO Pull-up/Pull-down	No pull-up and no pull-down

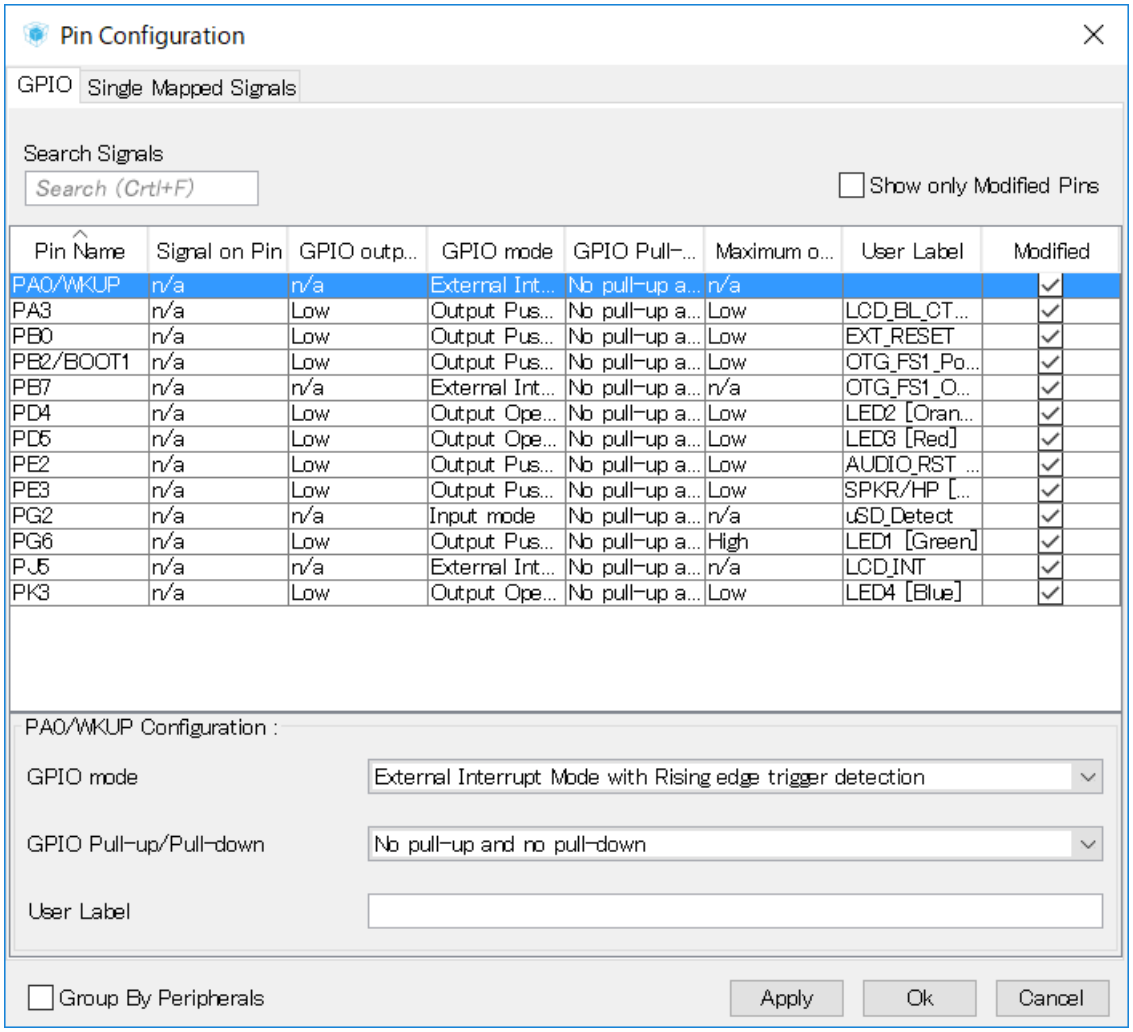
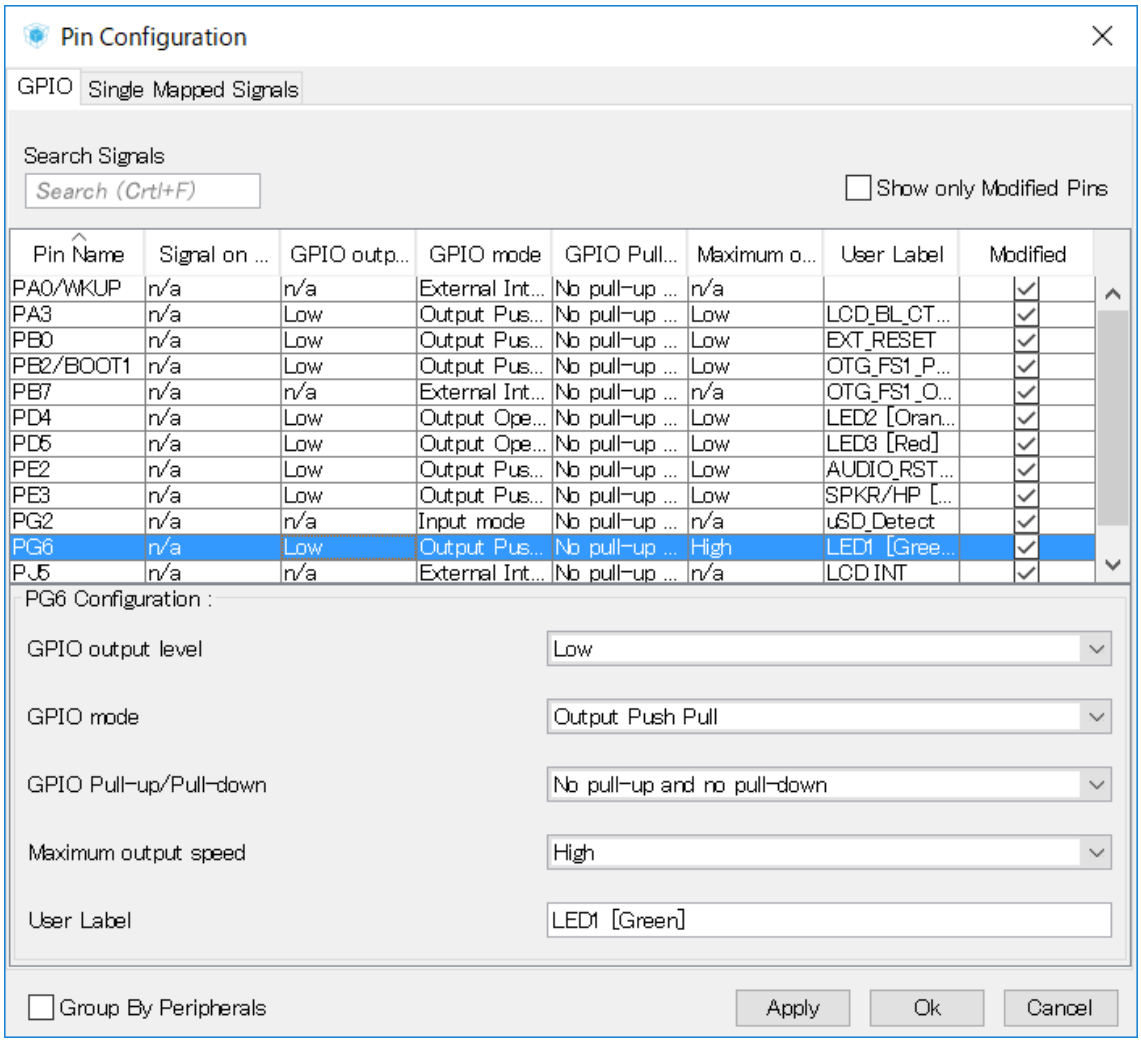


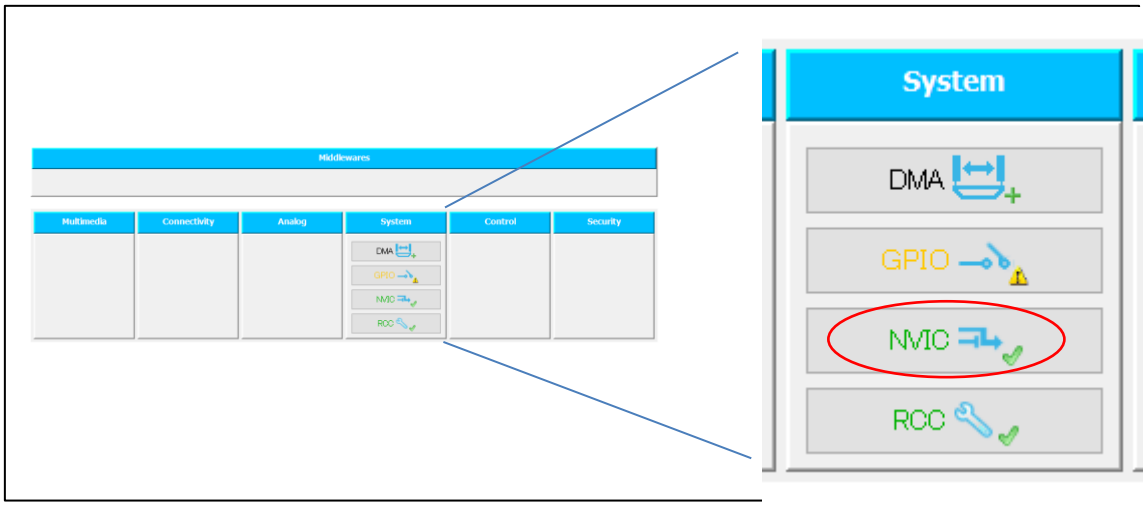
表 3.3 PG6 設定

項目	設定
GPIO output level	Low
GPIO mode	Output push-pull
GPIO Pull-up/Pull-down	No pull-up and no pull-down
Maximum output speed	High



3.2.5. 割り込み設定

次に Configuration タブの NVIC ボタンを押します。



以下のように設定します。

表 3.4 割り込み優先度設定

項目	設定
Priority Group	4bits for pre-emption priority 0 bits for subpriority
System tick timer	Enabled: Checked Preemption Priority: 15 Sub Priority: 0
EXTI line0 interrupts	Enabled: Checked Preemption Priority: 15 Sub Priority: 0
Time base: TIM1 update interrupt and TIM10 global interrupt	Enabled: Checked Preemption Priority: 1 Sub Priority: 0

2 章で述べたように Preemption Priority には 0 を設定しないようにしてください。

3.2.6. ソースコード生成

C:¥ LED_BLINK というフォルダを作成します。STM32CubeMX のプルダウンメニュー Project->Setting を選択し以下のように設定します。

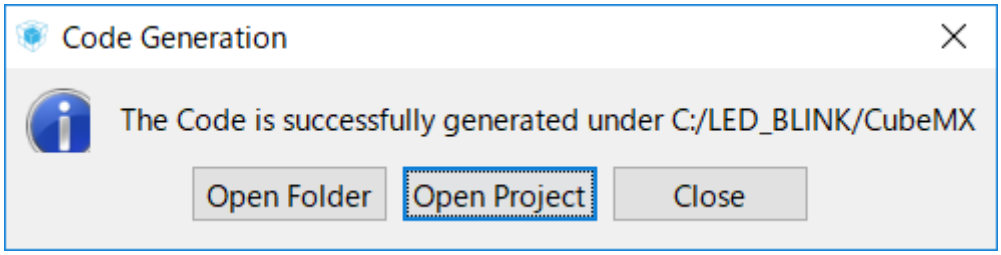
表 3.5 STM32CubeMX プロジェクト設定

タブ	設定
Project タブ	Project Name -> CubeMX
	Project Location -> C:¥ LED_BLINK
	Tool chain -> EWARM
Code Generator タブ	Copy only the necessary library files
	Keep User Code when re-generating
	Delete previously generated files when not re-generated

OK ボタンを押下すると、設定状態が「C:¥ LED_BLINK¥CubeMX¥CubeMX.ioc」として保存されます。

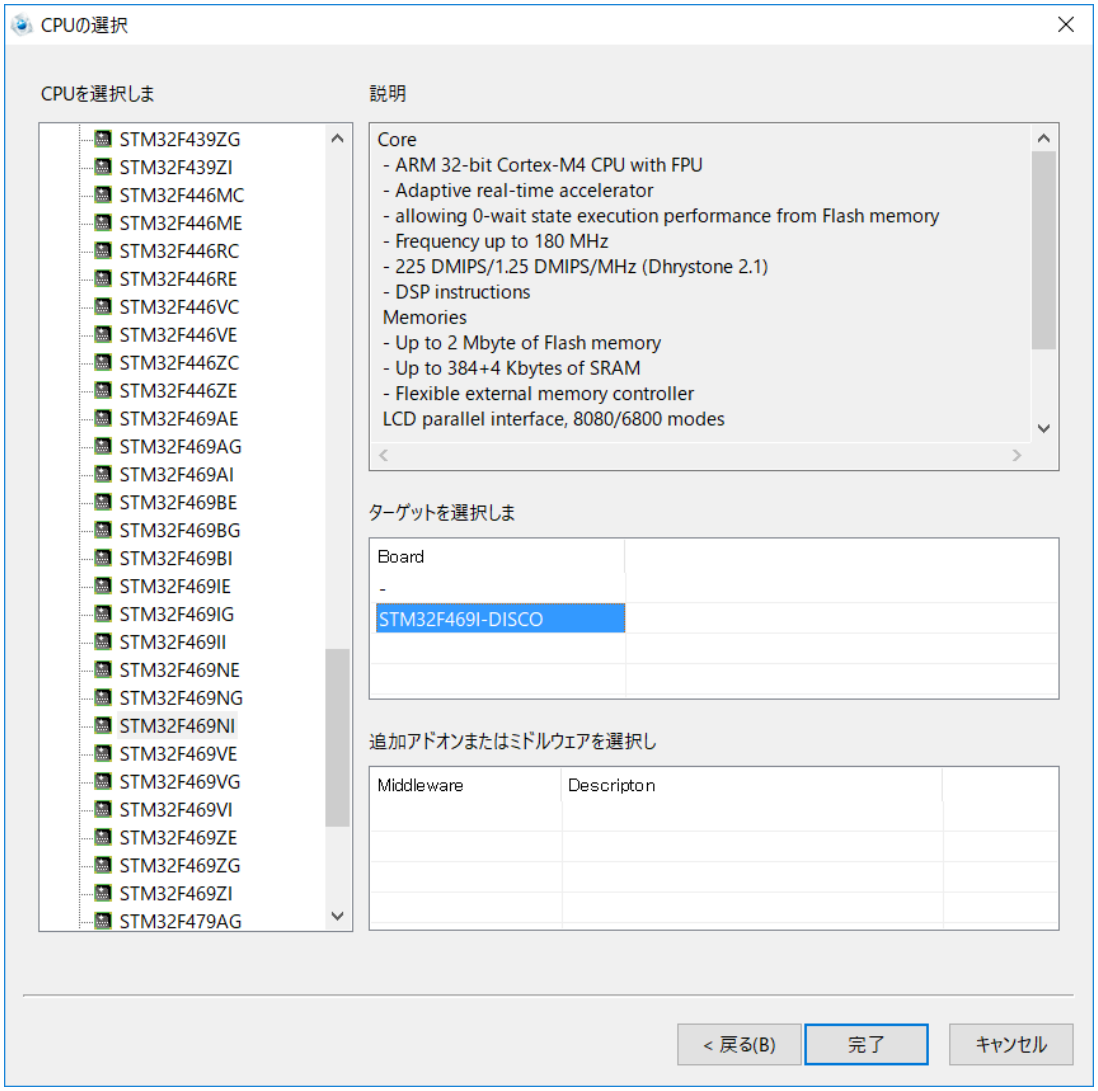
Project->Generate Code を選択すると、EWARM のプロジェクトとして、ソースコードが「C:¥ LED_BLINK ¥CubeMX¥」に生成されます。

以下のメッセージが表示されるので、「Close」で閉じます。

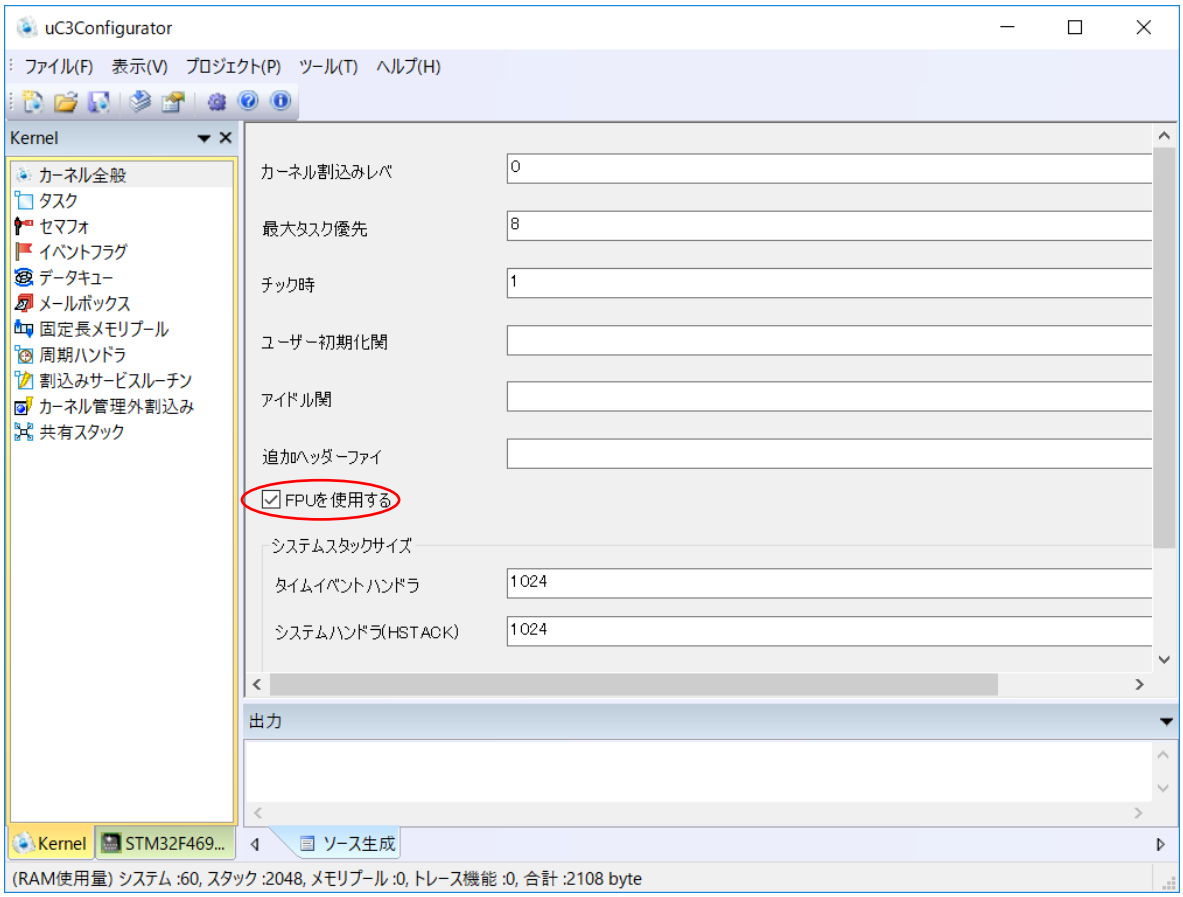


3.3. μC3/Compact の設定

μC3/Compact のコンフィグレータを起動し、STM32F469NI を選択、ターゲットは、「STM32F469I-DISCO」とし、完了をクリックします。

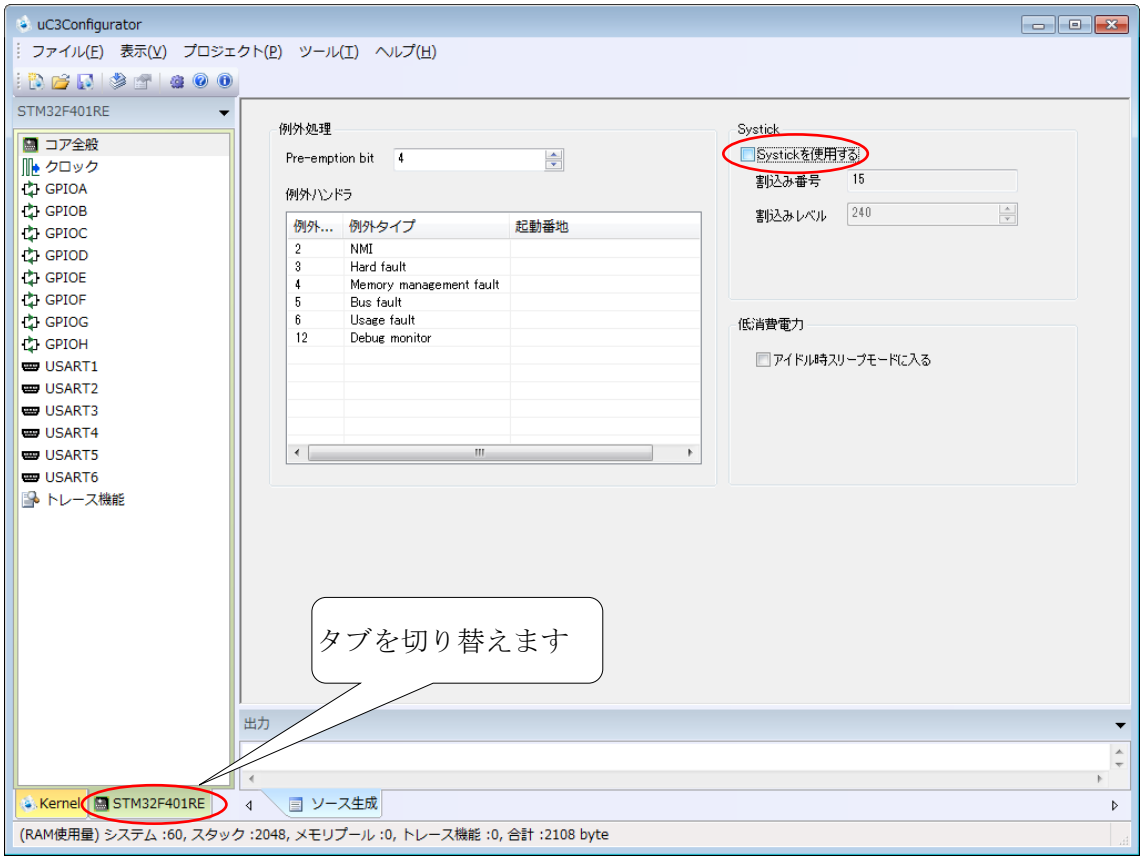


次の画面が開いた後、カーネル全般タブの FPU を使用するにチェックを入れます。



またチック時が 1 であることを確認します。
※チック時を変更した場合、自動生成コードで修正する箇所が発生します。その箇所については 3.5 で記述致します。

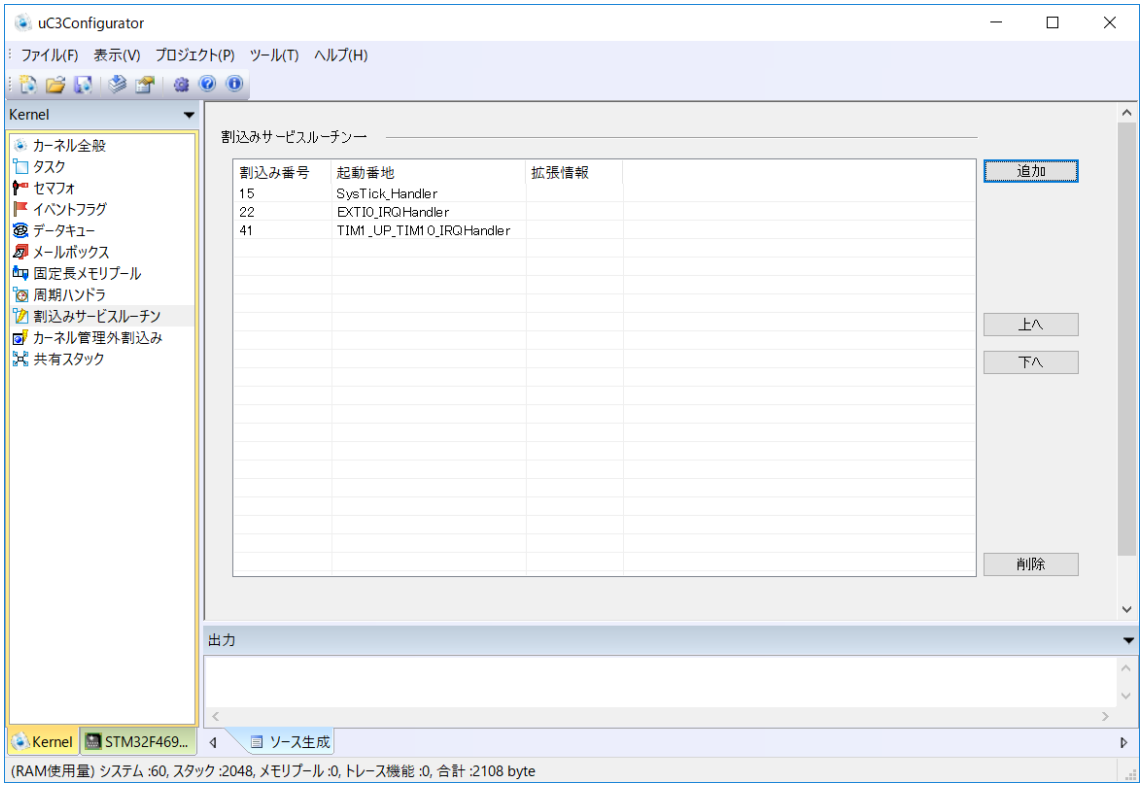
次に、μC3/Compact が生成するソースコードから、Cortex-M3,M4 の周辺機能である、システムタイマ（SysTick）を利用しなくするための設定を実施します。コア全般の「Systick を使用する」のチェックを外します。



次に、μC3/Compact の割込みサービスルーチンを登録します。STM32CubeMX が生成した「C:¥ LED_BLINK ¥CubeMX¥Src¥stm32f4xx_it」に下記割込みハンドラの関数が登録されていますので、これをμC3/Compact のコンフィグレータ上で登録します。これは STM32CubeMX が生成した割込み処理をμC3/Compact の RTOS が管理する割込みサービスルーチンとするための設定です。

表 3.6 割り込みサービスルーチン設定

割り込み番号	起動番地
15	SysTick_Handler
22	EXTI0_IRQHandler
41	TIM1_UP_TIM10_IRQHandler



次は、タスク部分です。以下のタスクを登録します。

表 3.7 タスク設定

ID	関数名	タスク属性(RTOS 起動時のタスク状態)	スタックサイズ(Byte)
ID_TASKMAIN	TaskMain	TA_ACT(実行可能状態)	256
ID_TASKLED	TaskLed	休止状態	256

それぞれのタスクは以下の役割を持ちます。

ID_TASKMAIN

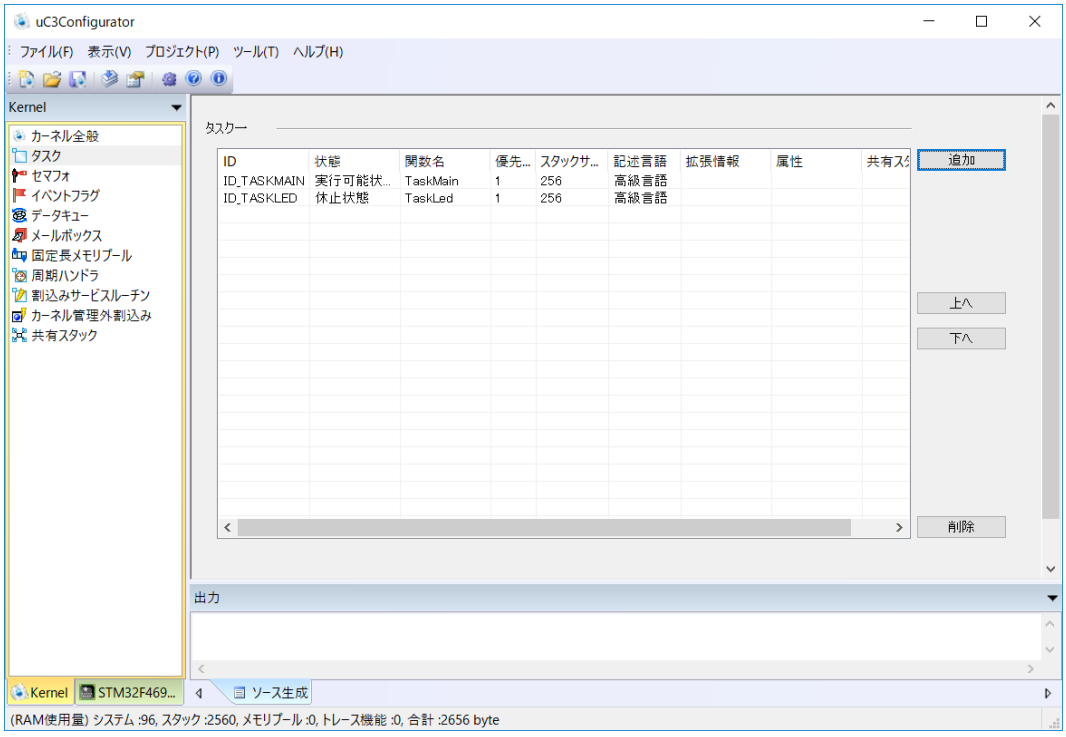
STM32CubeMX の自動生成する main()関数内で行っていた各種ペリフェラルの初期化。

ID_TASKLED

USER ボタンの押下待ち状態を slp_tsk()システムコールで行い、起床後 LED の点滅制御を行う。

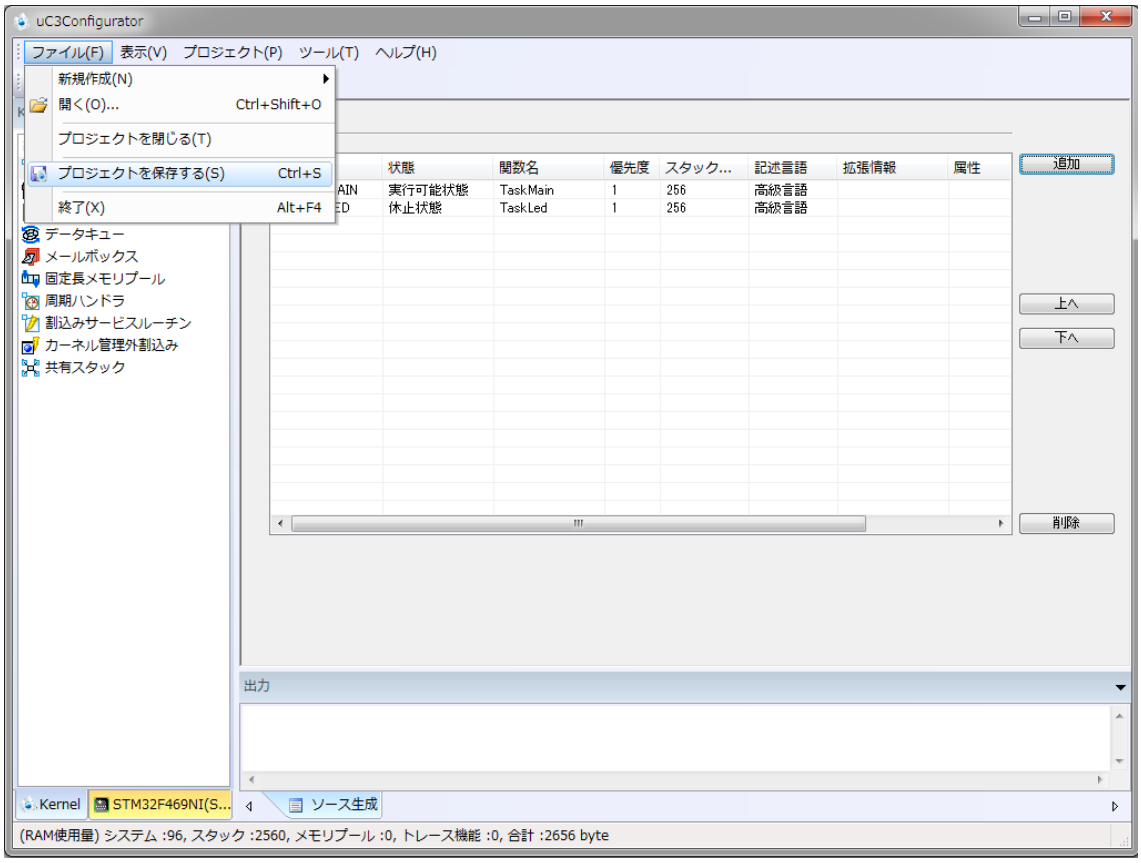
※ここではスタックサイズをデフォルトの 256 としてありますが、アプリケーションのよってはスタックサイズを増やす必要があります。

コンフィグレータでのタスクの実装例を以下に示します。

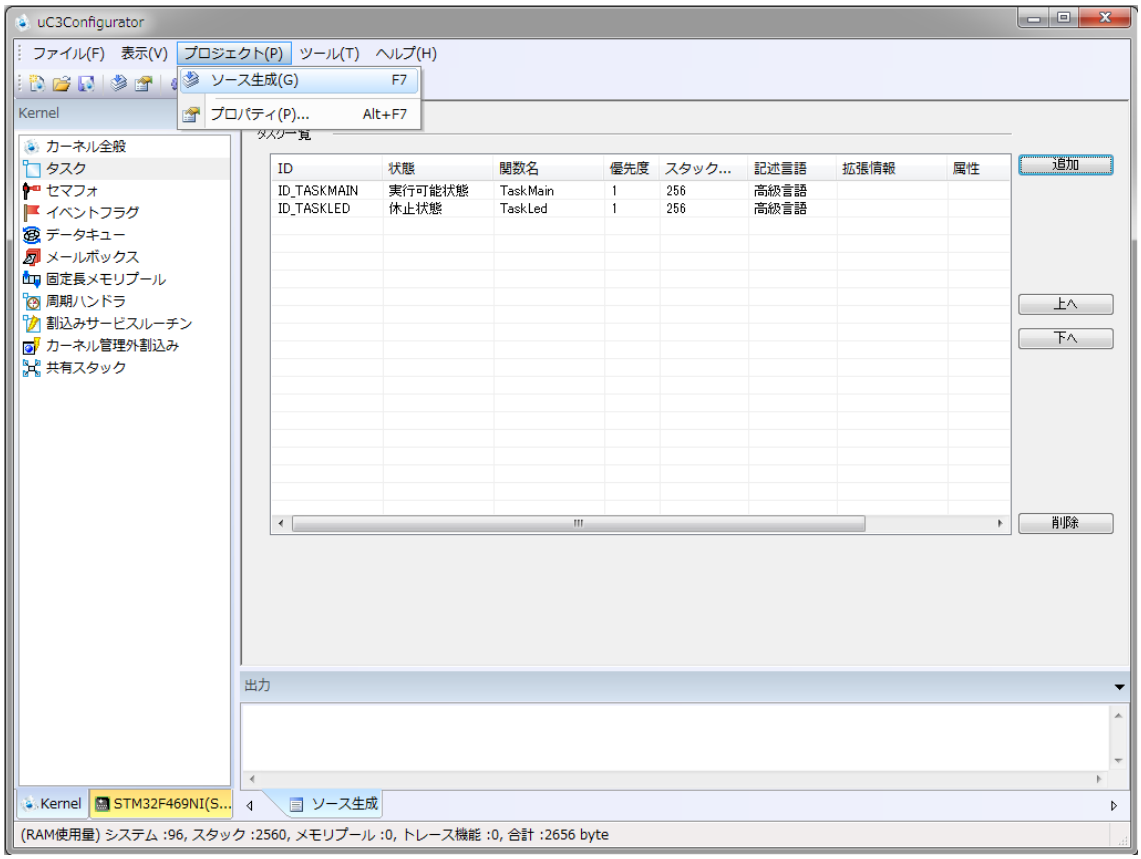


クロック、GPIO については、STM32CubeMX で設定するため μC3 コンフィグレータ上では設定しません。

ここで C:¥LED_BLINK¥uC3 を作成した後、「ファイル」->「プロジェクトを保存する」を選択してコンフィグレータの設定内容 uC3Project.3cf を C:¥LED_BLINK¥uC3 に保存します。



その後「プロジェクト」->「ソース生成」を選択し C:¥ LED_BLINK ¥uC3 にソースコード生成を実施します。

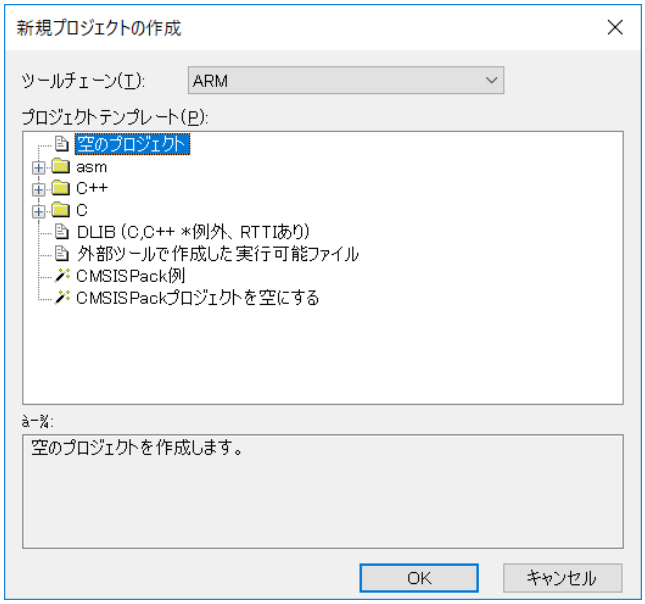


生成されたファイルの「C:¥LED_BLINK¥uC3¥main.c」を main_0.c にリネームします。

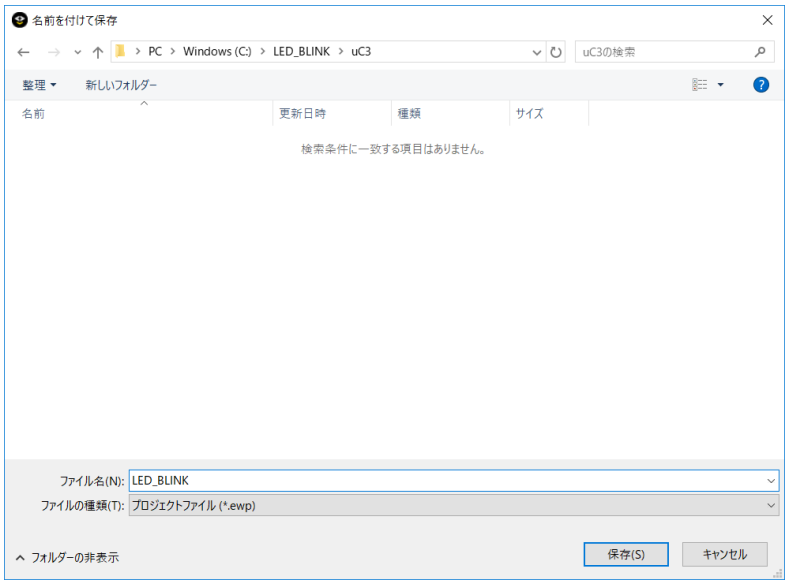
3.4. EWARM のプロジェクト設定

この節では、STM32CubeMX が生成したソースコードと、μC3/Compact が生成したソースコードを用いて EWARM のプロジェクトを作成する手順を示します。

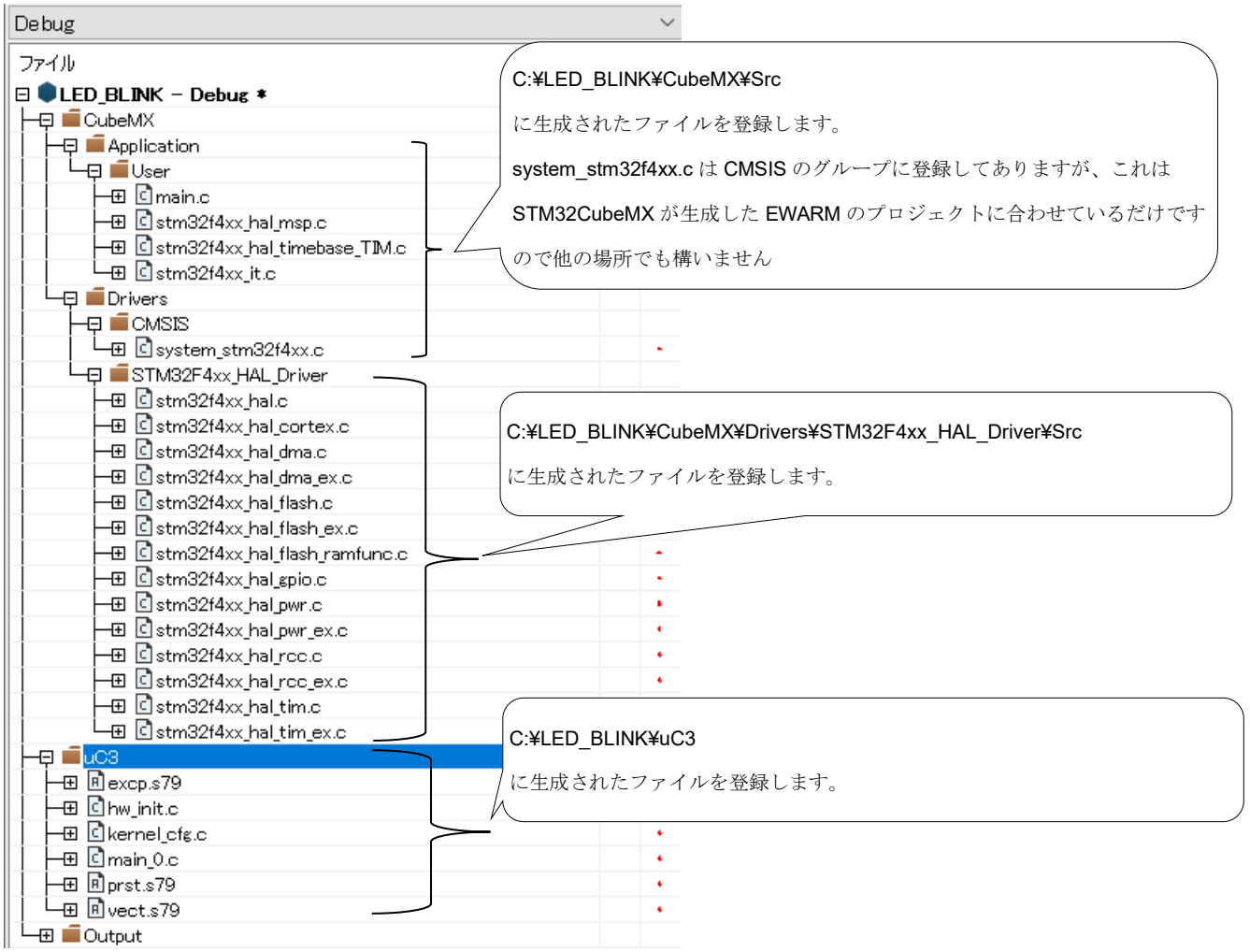
「IAR Embedded Workbench」を起動します。最初に、「プロジェクト」→「新規プロジェクトの作成」によりプロジェクトを作成します。この時、「空のプロジェクト」を選択し、「OK」をクリックして空のプロジェクトを作成します。



プロジェクトには、任意の名前を付け保存します。ここでは、μC3 コンフィグレータが生成したファイルのあるフォルダに、プロジェクト名を「LED_BLINK」にし「保存」をクリックします。

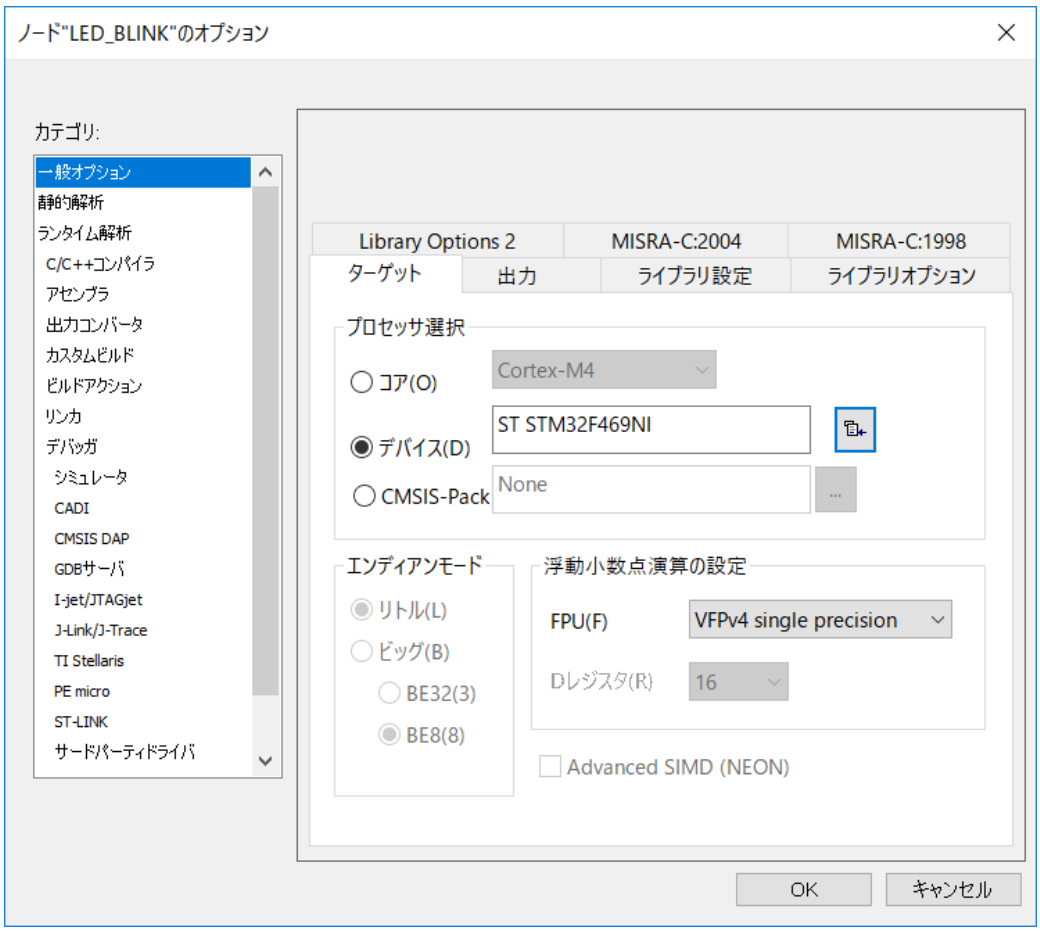


プロジェクトへのファイルの登録は以下のようにします。STM32CubeMX が生成したベクターテーブル startup_stm32f469xx.s は登録しないことに注意してください。

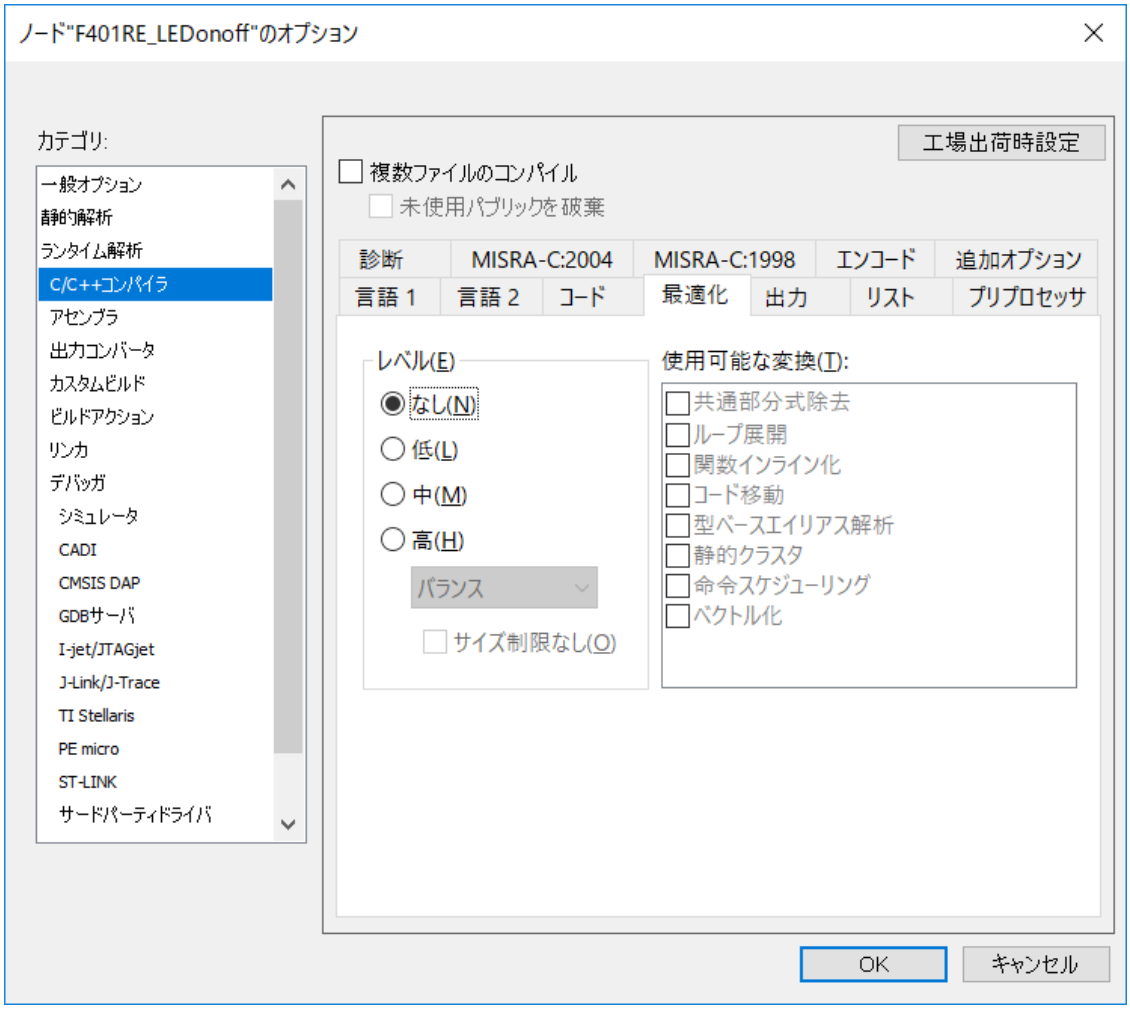


オプションの設定は以下のように行います。

① カテゴリ「一般オプション」の「ターゲット」タブでは、「デバイス」をクリックし、さらにその右にあるボタンをクリックしてデバイスリストを表示させます。該当するデバイスを選択します。



② カテゴリ「C/C++コンパイラ」の「最適化」タブでは、レベルをなしにします。
※この設定は必須ではありません。



③　　「プリプロセッサ」タブで追加インクルードディレクトリに

$PROJ_DIR$¥..¥CubeMX¥Inc

$PROJ_DIR$¥..¥CubeMX¥Drivers¥STM32F4xx_HAL_Driver¥Inc

$PROJ_DIR$¥..¥CubeMX¥Drivers¥STM32F4xx_HAL_Driver¥Inc¥Legacy

$PROJ_DIR$¥..¥CubeMX¥Drivers¥CMSIS¥Device¥ST¥STM32F4xx¥Include

$PROJ_DIR$¥..¥CubeMX¥Drivers¥CMSIS¥Include

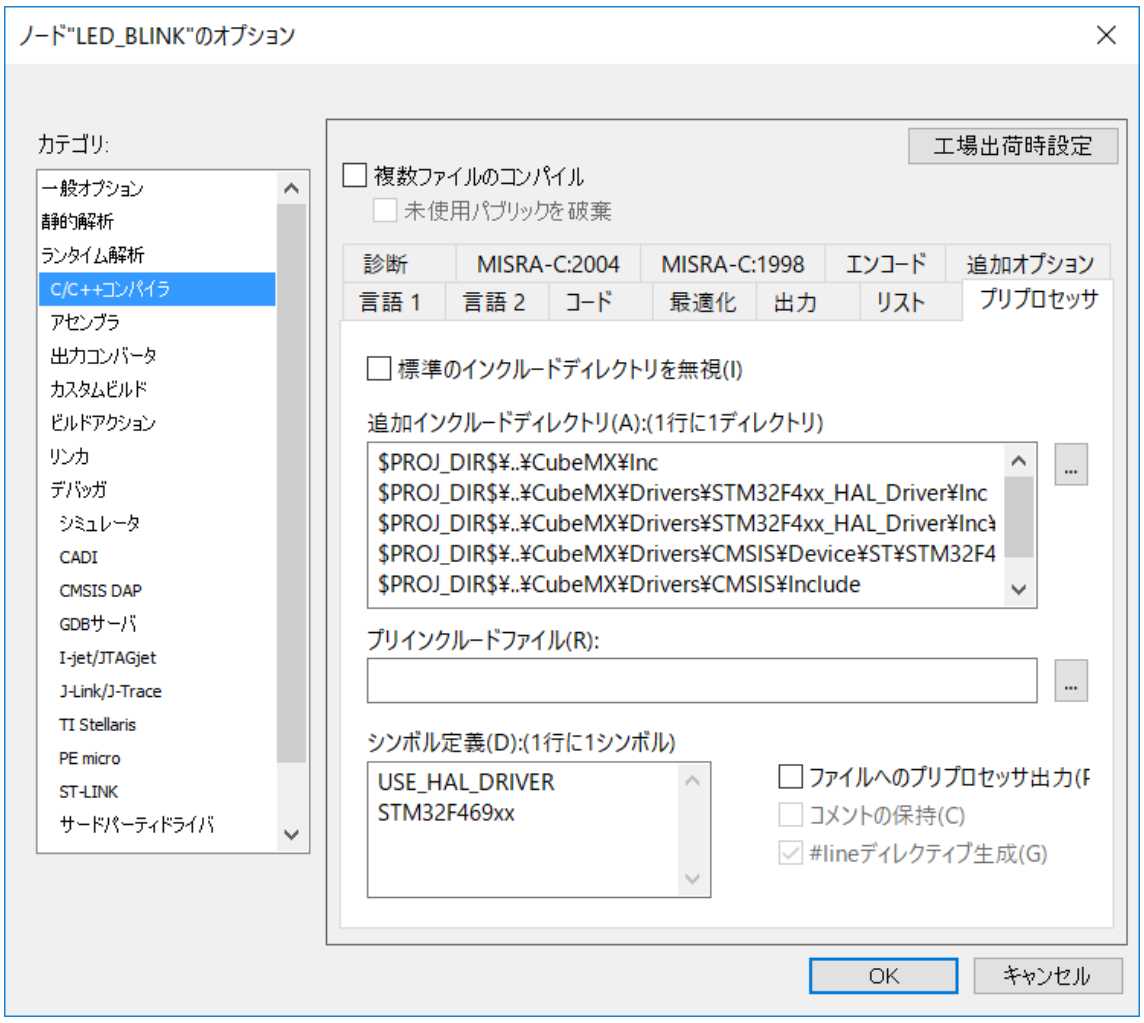
$PROJ_DIR$¥.

を登録します。またシンボル定義には

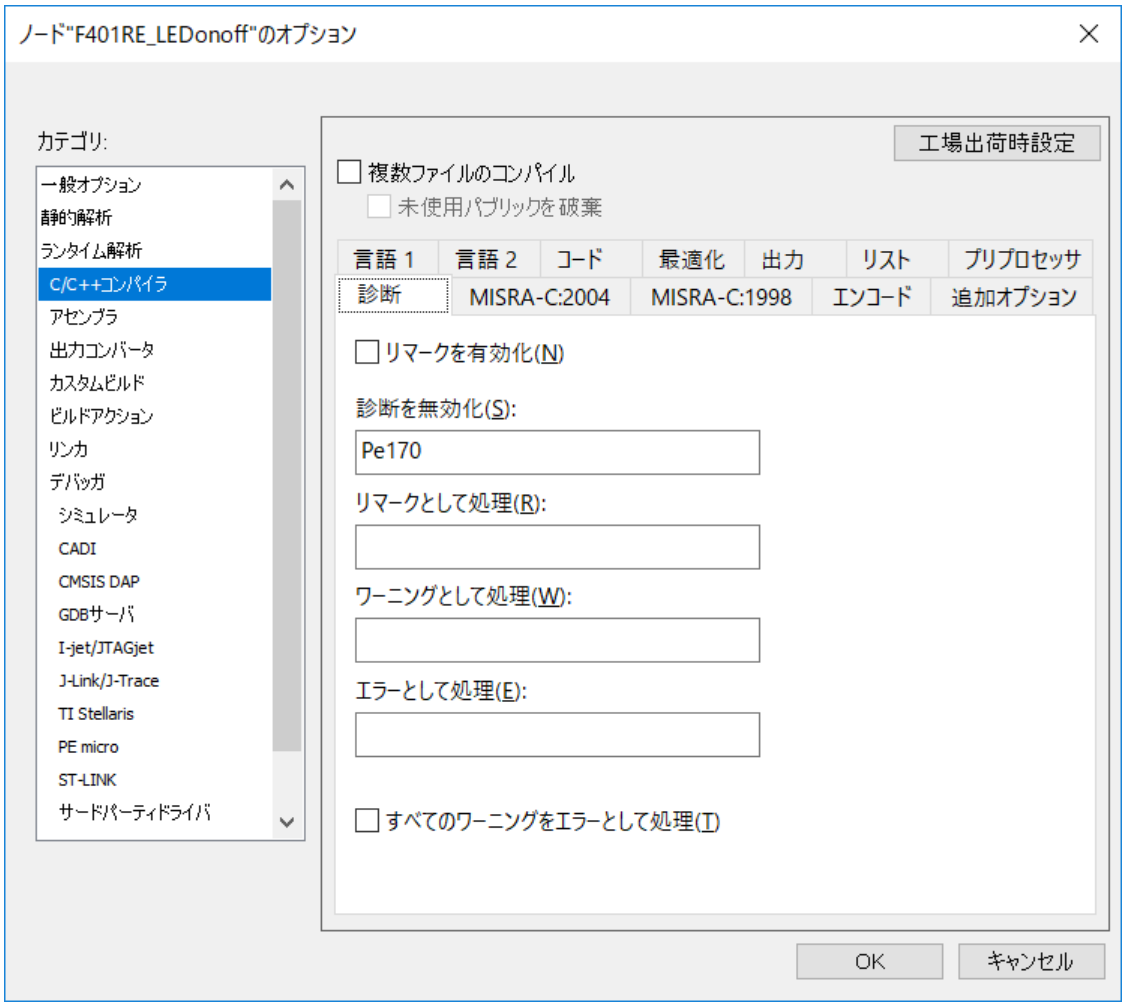
USE_HAL_DRIVER

STM32F469xx

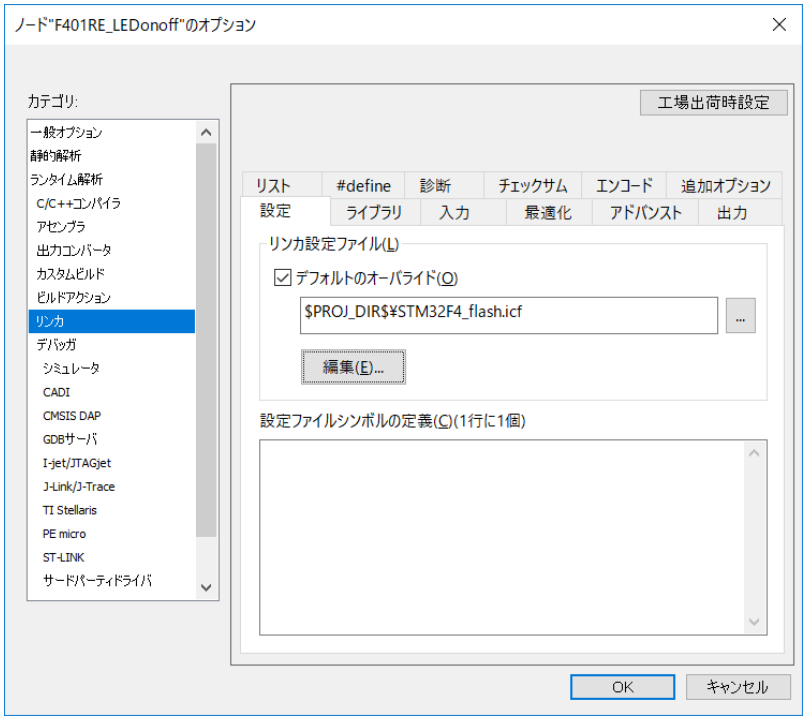
を登録します。



④　　「診断」タブで診断を無効化に Pe170 を登録します。



⑤　　カテゴリ「リンカ」の「設定」タブでは、リンカ設定ファイルを μC3 が生成した $PROJ_DIR$¥STM32F4_flash.icf に変更します。

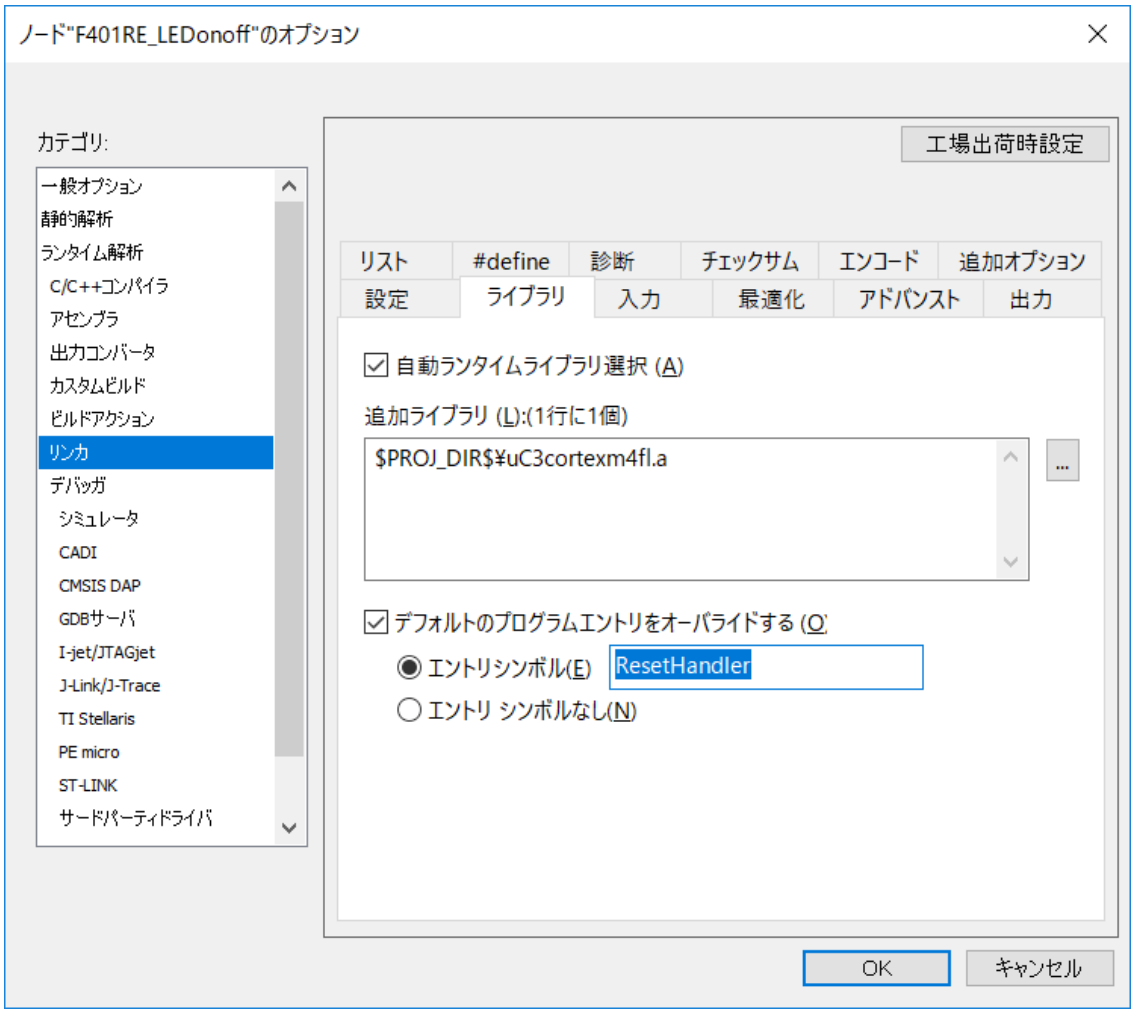


またリンカスクリプトをテキストエディタで開き HEAP 領域を確保します。

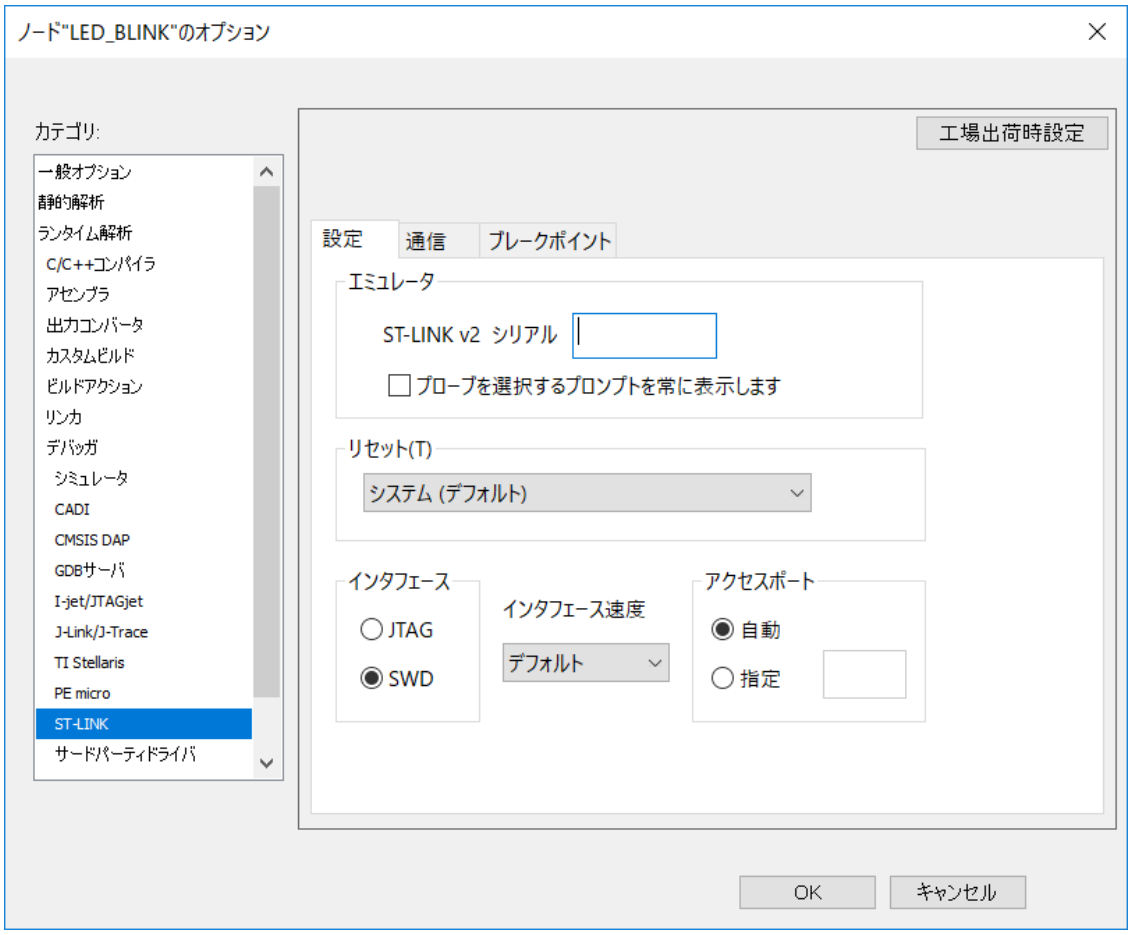
```
/*###ICF### Section handled by ICF editor, don't touch! ****/
…
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__   = 0x200;
/**** End of ICF editor section. ###ICF###*/
…
define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };

…
place at end of RAM_region { block … order {block HEAP, block HSTACK, block CSTACK }};
```

⑥　　「ライブラリ」タブでは、追加ライブラリに μC3 が生成した $PROJ_DIR$¥uC3cortexm4fl.a を登録します(FPU を使用する場合のカーネル)。またエントリシンボルを ResetHandler とします。



⑦　　「デバッガ」カテゴリは、使用状況によって変わりますが今回の環境では「設定」タブのドライバを「ST-LINK」に設定しました。それに合わせて「ST-LINK」カテゴリの「設定」タブでインターフェースを SWD にします。



3.5. 自動生成コードの修正

① 初期化に関する修正

C:¥ LED_BLINK¥CubeMX¥Src¥main.c を開き以下の 3 カ所に修正を加えます。

int main(void)関数内の while ループをコメントアウト。

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
//while (1)
//{
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */

//}
/* USER CODE END 3 */
```

int main(void)を void TaskMain(VP_INT exinf)にリネーム。

```
void TaskMain(VP_INT exinf)
```

インクルードの追加。

```
/* USER CODE BEGIN Includes */
/* {{UC3_INCLUDE */
#include "kernel.h"
#include "kernel_id.h"
#include "hw_dep.h"
/* }}UC3_INCLUDE */
/* USER CODE END Includes */
```

またμC3 側が自動生成したスタートアップ処理 (prst.s79)でSTM32CubeMXの生成コードを利用する部分は修正します。具体的にはクロックの設定と GPIO の設定を行う「ldr r0, =REG_RCC」から「; Enter the iar code」の手前の部分を以下のように修正します。

```
        EXTERN  SystemInit
        EXTERN  __iar_program_start

        PUBLIC  ResetHandler
ResetHandler:
        mrs     r0,  control
        ldr     r1,  =SFE(CSTACK)
        orr     r0,  r0,  #0x02
        ldr     r2,  =SFE(HSTACK)
        bic     r0,  r0,  #0x01
        msr     psp, r1
        msr     msp, r2
        msr     control,  r0

        ldr     r0, =SystemInit
        blx     r0

; Enter the iar code
        ldr     r0, =__iar_program_start
        bx      r0
```

EXTERN SystemInit を追加。

SystemInit を用いて初期化を行います。

② 割り込みサービスルーチンとタスク

C:¥ LED_BLINK¥uC3¥main_0.c 内の割り込みサービスルーチンとタスクのスケルトンコードをコメントアウトします。

```
/*******************************
        SysTick_Handler
 *******************************/
/* {{UC3_ISR(SysTick_Handler) */
//void SysTick_Handler(VP_INT exinf)
//{
//}
/* }}UC3_ISR */
(中略)
/*******************************
        TaskMain
 *******************************/
/* {{UC3_TASK(TaskMain) */
//void TaskMain(VP_INT exinf)
//{
//}
/* }}UC3_TASK */
```

STM32CubeMX 側のものを用いるのでコメントアウトします

また C:¥LED_BLINK¥CubeMX¥Inc¥stm32f4xx_it.h 内に以下のインクルード文を追加します。

```
/* Includes ------------------------------------------------------------------*/

/* {{UC3_INCLUDE */

#include "kernel.h"

#include "kernel_id.h"

#include "hw_dep.h"

/* }}UC3_INCLUDE */
```

さらに割り込みサービスルーチンの引数を void から VP_INT exinf に変更します。
C:¥LED_BLINK¥CubeMX¥Inc¥stm32f4xx_it.h だけでなく
C:¥LED_BLINK¥CubeMX¥Src¥stm32f4xx_it.c に対しても行います。

```
void SysTick_Handler(VP_INT exinf);

void EXTI0_IRQHandler(VP_INT exinf);

void TIM1_UP_TIM10_IRQHandler(VP_INT exinf);
```

これは μC3/Compact の割込みサービスルーチンの関数とするための修正です。

③　　C:¥ LED_BLINK¥CubeMX¥Src¥main.c に割り込みサービスルーチンとタスクの実装を追加します。

TaskMain() 内の /* USER CODE BEGIN 2 */~/* USER CODE END 2 */ の間で act_tsk(ID_TASKLED)をコールして LED タスクを起床させる処理を追加。

```
/* USER CODE BEGIN 2 */
act_tsk(ID_TASKLED);
/* USER CODE END 2 */
```

LED タスクを起床させる

/* USER CODE BEGIN 4 */~/* USER CODE END 4 */の間に LED タスクの関数を記述します。

```
void TaskLed(VP_INT exinf)
{
  while(1){
      slp_tsk();
      HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
  }
}
```

LED1 をトグルしその後休止します

/* USER CODE BEGIN 4 */~/* USER CODE END 4 */の間にボタン押下を検出する処理を追加します。

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  if(GPIO_Pin == A0_Pin){
    iwup_tsk(ID_TASKLED);
  }
}
```

USER ボタンが押されると LED タスクを起床させます

STM32CubeMX の SYSTICK の更新をμC3 側に反映させるため処理を追加します。

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
/* USER CODE BEGIN Callback 0 */

/* USER CODE END Callback 0 */
  if (htim->Instance == TIM1) {
    HAL_IncTick();
    isig_tim();
  }
/* USER CODE BEGIN Callback 1 */

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

isig_tim()で SYSTICK の更新を μC3 側に反映します

※μC3/Compact のコンフィグレータで「Systick を使用する」のチェックを外した場合タイマの割り込みサービスルーチンから isig_tim()を呼ぶ必要があります。またμC3/Compact のコンフィグレータでチック時を 1 から変更した場合、タイマの割り込みサービスルーチンの周期を変更する必要があります。具体的には
C:¥ LED_BLINK¥CubeMX¥Src¥stm32f4xx_hal_timebase_TIM.c 内の関数 HAL_InitTick()において行っている以下の処理を修正します。

```
  htim1.Init.Period = (1000000 / 1000) - 1;
```

例えばチック時を 2 とした場合は

```
  htim1.Init.Period = (1000000 * 2 / 1000) - 1;
```

とします。

④　　C:¥ LED_BLINK¥CubeMX¥Src¥main.c の void SystemClock_Config(void)の先頭に HAL_RCC_DeInit()を追加します。

```
void SystemClock_Config(void)
{
  HAL_RCC_DeInit();
```

HAL_RCC_DeInit()を追加します

これにより SystemClock_Config()が複数回コールされる度にクロックの再設定が行えるようになります。

ビルドして実行すると、USER ボタンを押下する度に LED が点滅します。

また以下のようにプログラムを修正すると 10s ごとに LED が点滅するようになり TICK の更新周期を確認することができます。

```
void TaskLed(VP_INT exinf)
{
  while(1){
      //slp_tsk();
      dly_tsk(10000);
      HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
  }
}
```

3.6. 動作確認環境

動作確認にはボード上の以下のコンポーネントを使用します。

ボード上のコンポーネント	備考
CN1 ST-LINK/V2-1	電源供給及びデバッグのため、USB2.0 ケーブル(mini-B タイプ)の mini-B 側を接続します(A 側は PC に接続します)。
B2 Wake-up button	外部割込みのため使用します。
LD1	緑色 LED

記号については ST マイクロエレクトロニクス提供 User manual UM1932: Discovery kit with STM32F469NI MCU(http://www.st.com/resource/en/user_manual/dm00218846.pdf)を参照してください。

4. サンプルプログラム(USB-HID)作成手順

この章では、「USER ボタンを押すと PC 上のマウスポインタが移動する」プログラムの作成手順を説明します。3 章の説明と重複する箇所については簡略的な説明にしてあります。

4.1. フォルダ構成

最上位のフォルダ名が LED_BLINK→USB-HID となる以外は 3.1 の説明と同じになります。

4.2. STM32CubeMX を用いたプロジェクト作成

4.2.1. 新規プロジェクトの作成

3.2.1 で説明した手順と同じになります。

4.2.2. Pinout 設定

Pinout タブを選択し以下のように設定します。

USER ボタン押下を検知する割り込みの設定。

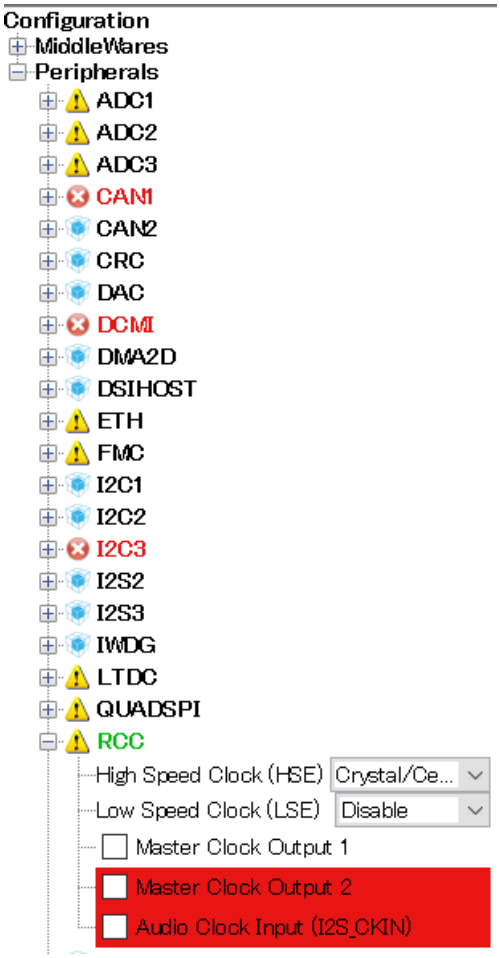
3.2.2 と同じく PA0/WKUP を GPIO_EXTI0 に設定します。

RTOS を動かすための設定。

3.2.2 と同じく SYS の設定で TimeBase Source を TIM1 と設定します。

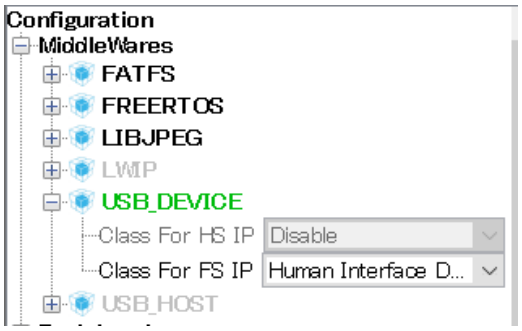
USB を使用するための設定。

左ペインから Configuration->Peripherals->RCC を選択し、HSE を Crystal/Ceramic に設定します。



次に Configuration->Peripherals->USB_OTG_FS を選択し、Mode を Device Only に設定し、Activate_VBUS にチェックを入れます。

そして Configuration->MiddleWares->USB_DEVICE を選択し、Class For FS IP を Human Interface Device に設定します。

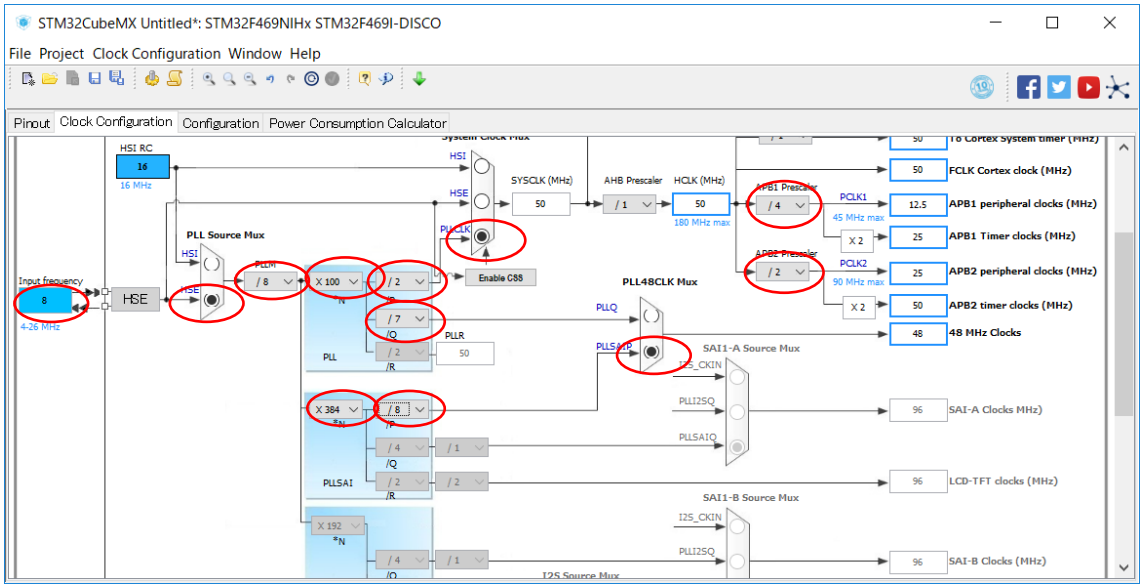


4.2.3. クロック設定

Clock Configration タブを選択し以下のように設定します。

表 4.1 クロック設定

項目	設定
System CLK Mux	PLLCLK
PLL source	HSE
Input frequency	8
PLL M	8
PLL N	100
PLL P	2
PLL Q	7
APB1 Prescaler	4
APB2 Prescaler	2
PLL48CLK Mux	PLLSAIP
PLLSAI N	384
PLLSAI P	8



4.2.4. GPIO 設定

次に、Configuration タブの GPIO ボタンを選択します。

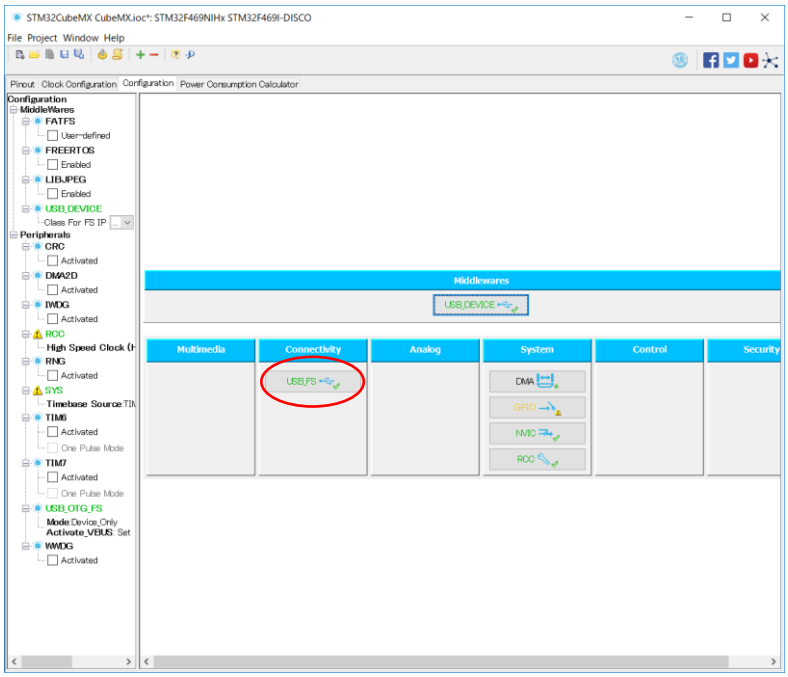
PA0/WKUP の設定を確認します。ここは 3.2.4 と同じ設定になります。

表 4.2 PA0/WKUP 設定

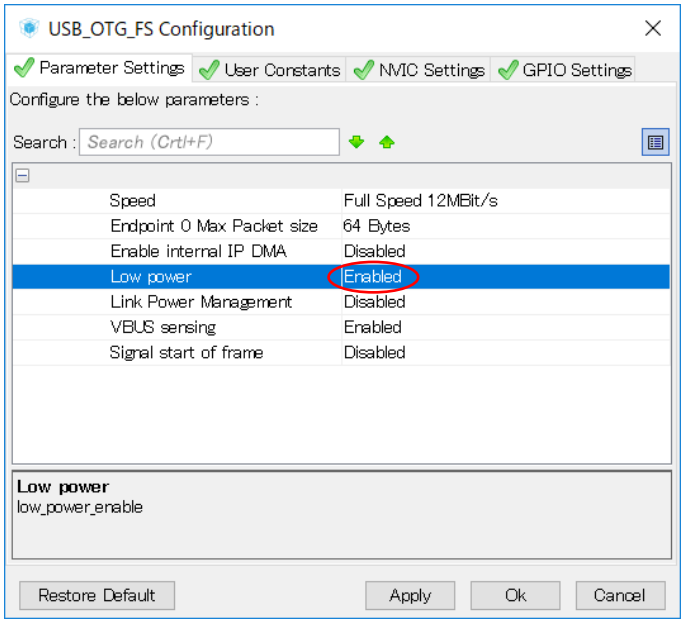
項目	設定
GPIO mode	External Interrupt Mode with Rising edge trigger detection
GPIO Pull-up/Pull-down	No pull-up and no pull-down

4.2.5. USB 設定

Configuration タブの USB_OTG_FS ボタンを押します。



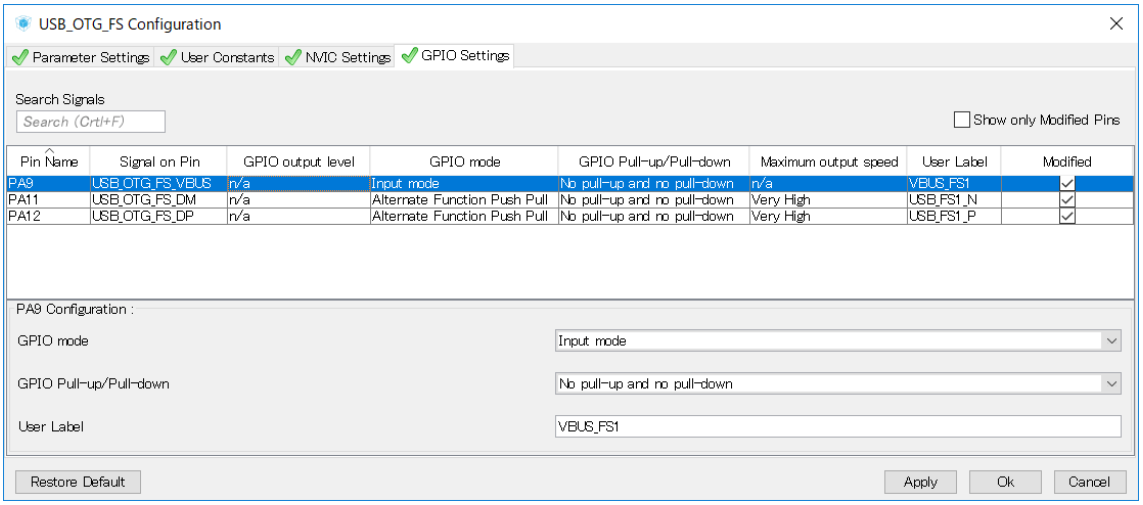
Low Power を Enabled に設定し、OK ボタンを押下します。



Configuration タブの GPIO ボタンを選択します。さらに USB_OTG_FS タブを選択し PA9 を以下のように設定します。

表 4.3 PA9 設定

項目	設定
GPIO mode	Input mode
GPIO Pull-up/Pull-down	No pull-up and no pull-down



4.2.6. 割り込み設定

Configuration タブの NVIC ボタンを押し以下のように設定します。

表 4. 4 割り込み優先度設定

項目	設定
Priority Group	4bits for pre-emption priority 0 bits for subpriority
Pendable request for system service	Enabled: Checked Preemption Priority: 15 Sub Priority: 0
System tick timer	Enabled: Checked Preemption Priority: 15 Sub Priority: 0
EXTI line0 interrupts	Enabled: Checked Preemption Priority: 15 Sub Priority: 0
Time base: TIM1 update interrupt and TIM10 global interrupt	Enabled: Checked Preemption Priority: 1 Sub Priority: 0
USB ON The Go FS wake-up interrupt though EXT line 18	Enabled: Checked Preemption Priority: 5 Sub Priority: 0
USB ON The Go FS global interrupt	Enabled: Checked Preemption Priority: 5 Sub Priority: 0

2 章で述べたように Preemption Priority には 0 を設定しないようにしてください。

4.2.7. ソースコード生成

フォルダ名とプロジェクト名を LED_BLINK→USB-HID と変更する以外は 3.2.6 で説明した内容と同じになります。

4.3. μC3/Compact の設定

μC3/Compact のコンフィグレータの起動、ターゲットの選択、FPU の使用、システムタイマ（SysTick）を利用しなくするための設定は、3.3 と同じ手順となります。

割込みサービスルーチンの登録については STM32CubeMX が生成した以下の割込みハンドラの関数をμC3/Compact のコンフィグレータ上で登録します。

表 4.5 割り込みサービスルーチン設定

割り込み番号	起動番地
15	SysTick_Handler
22	EXTI0_IRQHandler
41	TIM1_UP_TIM10_IRQHandler
58	OTG_FS_WKUP_IRQHandler
83	OTG_FS_IRQHandler

タスクは以下を登録します。

表 4.6 タスク設定

ID	関数名	タスク属性 (RTOS 起動時のタスク状態)	スタックサイズ (Byte)
ID_TASKMAIN	TaskMain	TA_ACT(実行可能状態)	512
ID_TASKUSBMOUSE	TaskUsbMouse	休止状態	256

それぞれのタスクは以下の役割を持ちます。

ID_TASKMAIN

STM32CubeMX の自動生成する main()関数内で行っていた各種ペリフェラルの初期化。

ID_TASKUSBMOUSE

USER ボタンの押下待ち状態を slp_tsk()システムコールで行い、起床後マウスポインタを動かす。

※ここでは TaskMain のスタックサイズを 512 としてありますが、アプリケーションのよってはさらにスタックサイズを増やす必要があります。

クロック、GPIO については、STM32CubeMX で設定するためμC3 コンフィグレータ上では設定しません。

コンフィグレータの設定内容の保存方法、ソースコード生成についてはフォルダ名を USB-HID とする以外は３章と同じ内容となります。そして生成されたファイルの「C:¥USB-HID¥uC3¥main.c」を main_0.c にリネームすることも３章と同じになります。

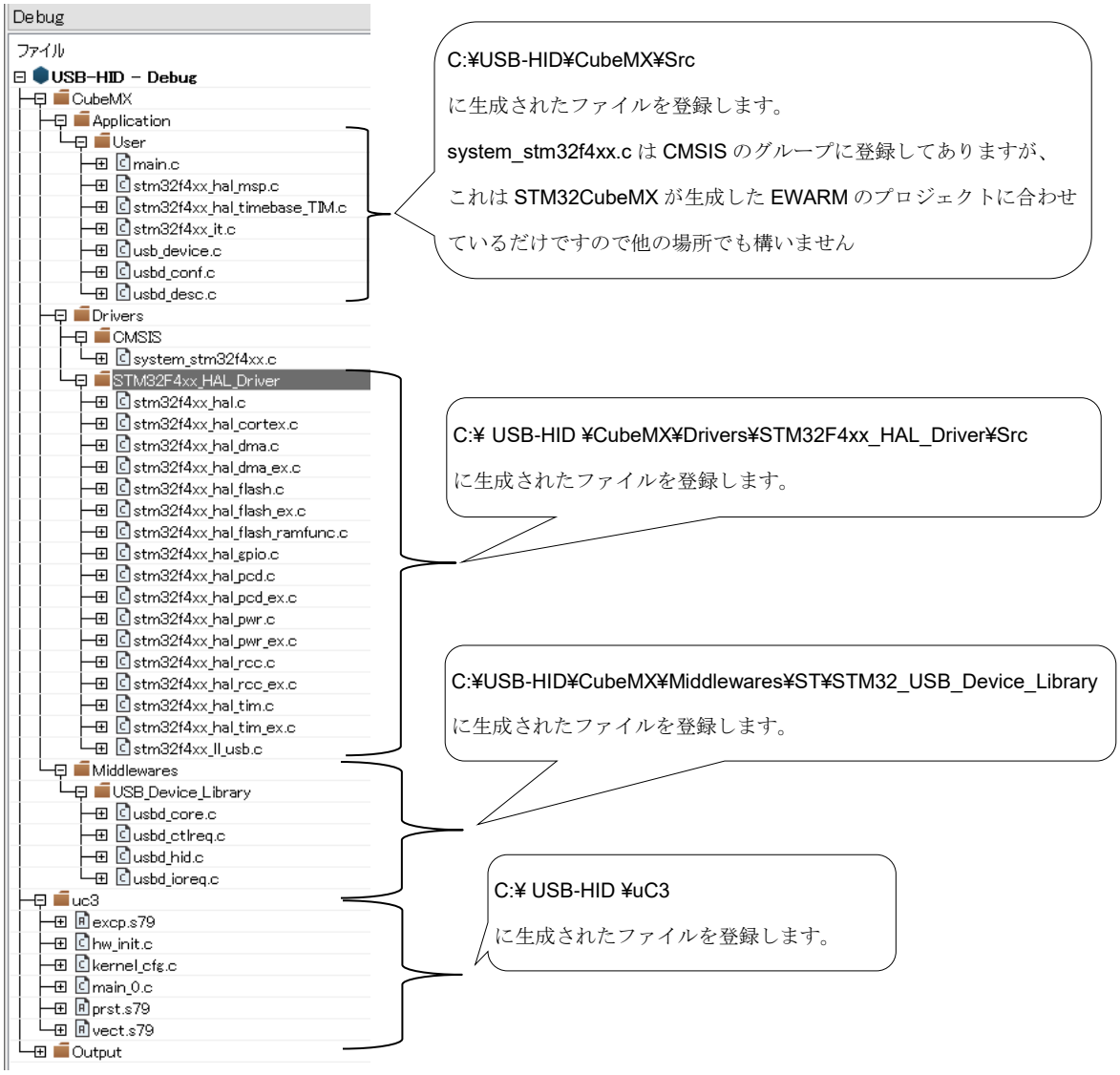
4.4. EWARM のプロジェクト設定

この節では、STM32CubeMX が生成したソースコードと、μC3/Compact が生成したソースコードを用いて EWARM のプロジェクトを作成する手順を示します。

「IAR Embedded Workbench」を起動します。最初に、「プロジェクト」→「新規プロジェクトの作成」によりプロジェクトを作成します。この時、「空のプロジェクト」を選択し、「OK」をクリックして空のプロジェクトを作成します。

プロジェクトには、任意の名前を付け保存します。ここでは、μC3 コンフィグレータが生成したファイルのあるフォルダに、プロジェクト名を「USB-HID」にし「保存」をクリックします。

プロジェクトへのファイルの登録は以下のようにします。



オプションの設定内容で 3.4 と異なる箇所は、「プリプロセッサ」タブで追加インクルードディレクトリに

$PROJ_DIR$¥..¥CubeMX¥Inc
$PROJ_DIR$¥..¥CubeMX¥Drivers/STM32F4xx_HAL_Driver/Inc
$PROJ_DIR$¥..¥CubeMX¥Drivers/STM32F4xx_HAL_Driver/Inc/Legacy
$PROJ_DIR$¥..¥CubeMX¥Middlewares/ST/STM32_USB_Device_Library/Core/Inc
$PROJ_DIR$¥..¥CubeMX¥Middlewares/ST/STM32_USB_Device_Library/Class/HID/Inc
$PROJ_DIR$¥..¥CubeMX¥Drivers/CMSIS/Device/ST/STM32F4xx/Include
$PROJ_DIR$¥..¥CubeMX¥Drivers/CMSIS/Include
$PROJ_DIR$/.

を登録する箇所です。その他は 3.4 の設定と同じになります。

4.5. 自動生成コードの修正

① 初期化に関する修正

C:¥ USB-HID¥CubeMX¥Src¥main.c を開き以下の修正を加えます。

int main(void)関数内の MX_USB_DEVICE_Init()呼び出しをコメントアウト。

USB の初期化関数 MX_USB_DEVICE_Init()は USB タスク TaskUsbMouse()から呼び出すのでここでの呼び出しはコメントアウトします。

int main(void)関数内の while ループをコメントアウト。

int main(void)を void TaskMain(VP_INT exinf)にリネーム。

これらは 3.5 と同じ修正です。

インクルードの追加。

```
/* USER CODE BEGIN Includes */

/* {{UC3_INCLUDE */

#include "kernel.h"

#include "kernel_id.h"

#include "hw_dep.h"

/* }}UC3_INCLUDE */

#include "usbd_hid.h"

/* USER CODE END Includes */
```

3.5 で記載したファイルの他、"usbd_hid.h"が追加しています。

またμC3 側が自動生成したスタートアップ処理 (prst.s79)で修正する箇所も 3.5 と同じになります。

② 割り込みサービスルーチンとタスク

C:¥ USB-HID ¥uC3¥main_0.c 内の割り込みサービスルーチンとタスクのスケルトンコードをコメントアウトすること、C:¥ USB-HID ¥CubeMX¥Inc¥stm32f4xx_it.h 内に以下のインクルード文を追加することも 3.5 と同じです。

```
/* Includes ------------------------------------------------------------------*/

/* {{UC3_INCLUDE */

#include "kernel.h"

#include "kernel_id.h"

#include "hw_dep.h"

/* }}UC3_INCLUDE */
```

割り込みサービスルーチンの引数を void から VP_INT exinf に変更することも 3.5 と同じです。

```
void SysTick_Handler(VP_INT exinf);

void EXTI0_IRQHandler(VP_INT exinf);

void TIM1_UP_TIM10_IRQHandler(VP_INT exinf);

void OTG_FS_WKUP_IRQHandler(VP_INT exinf);

void OTG_FS_IRQHandler(VP_INT exinf);
```

③　　C:¥ USB-HID¥CubeMX¥Src¥main.c に割り込みサービスルーチンとタスクの実装を追加します。

TaskMain() 内の /* USER CODE BEGIN 2 */~/* USER CODE END 2 */ の間で act_tsk(ID_TASKUSBMOUSE)をコールして USB MOUSE タスクを起床させる処理を追加。

```
/* USER CODE BEGIN 2 */
act_tsk(ID_TASKUSBMOUSE);
/* USER CODE END 2 */
```

USB MOUSE タスクを起床させる

/* USER CODE BEGIN 4 */~/* USER CODE END 4 */の間に USB MOUSE タスクの関数を記述します。

```
void TaskUsbMouse(VP_INT exinf)
{
   MX_USB_DEVICE_Init();
   /* Infinite loop */
   for(;;)
   {
         slp_tsk();
         static int8_t cnt = 0;
         int8_t    x = 0, y = 0 ;
         uint8_t HID_Buffer[4];

         if(cnt++ > 0)
         {
            x = 5;
         }
         else
         {
            x = -5;
         }
         HID_Buffer[0] = 0;HID_Buffer[1] = x; HID_Buffer[2] = y; HID_Buffer[3] = 0;
         USBD_HID_SendReport(&hUsbDeviceFS, HID_Buffer, 4);
   }
}
```

USEMOUSE タスク

USE 初期化処理

/* USER CODE BEGIN 4 */~/* USER CODE END 4 */の間にボタン押下を検出する処理を追加します。

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
   if(GPIO_Pin == A0_Pin){
      iwup_tsk(ID_TASKUSBMOUSE);
   }
}
```

USER ボタンが押されると USB MOUSE タスクを起床します。

STM32CubeMX の SYSTICK の更新を μC3 側に反映させるため処理の追加、C:¥ USB-HID ¥CubeMX¥Src¥main.c の void SystemClock_Config(void)の先頭に HAL_RCC_DeInit()を追加することは 3.5 と同じです。

Sleep 設定を変更。

C:¥ USB-HID ¥CubeMX¥Src¥ usbd_conf.c の HAL_PCD_SuspendCallback()と HAL_PCDEx_LPM_Callback()で行っている Sleep 設定において SCB_SCR_SLEEPONEXIT_Msk を無効にします。

```
/* SCB->SCR |=(uint32_t)((uint32_t)(SCB_SCR_SLEEPDEEP_Msk |
                                    SCB_SCR_SLEEPONEXIT_Msk)); */

SCB->SCR |= (uint32_t)((uint32_t)(SCB_SCR_SLEEPDEEP_Msk));
```

SCB_SCR_SLEEPONEXIT_Msk を無効にします。

ビルドして実行すると、USER ボタンを押下する度に PC 上のマウスポインタが移動します。

4.6. 動作確認環境

動作確認にはボード上の以下のコンポーネントを使用します。

ボード上のコンポーネント	備考
CN1 ST-LINK/V2-1	電源供給及びデバッグのため、USB2.0 ケーブル(A mini-B タイプ)の mini-B 側を接続します(A 側は PC に接続します)。
B2 Wake-up button	外部割込みのため使用します。
CN13 USB OTG FS Connector	マウスポインタカーソル移動のため、USB2.0 ケーブル(A micro-B タイプ)の micro-B 側を接続します(A 側は PC に接続します)。

記号については ST マイクロエレクトロニクス提供 User manual UM1932: Discovery kit with STM32F469NI MCU(http://www.st.com/resource/en/user_manual/dm00218846.pdf)を参照してください。

AN201701　STM32CubeMX と μ C3/Compact の共存方法

2017 年 09 月 29 日　　Rev.1.0

イー・フォース株式会社　http://www.eforce.co.jp/
お問い合わせ support@eforce.co.jp