

ModusToolbox 2.3

Quick Start

PWMによるLED点滅 プロジェクト作成手順

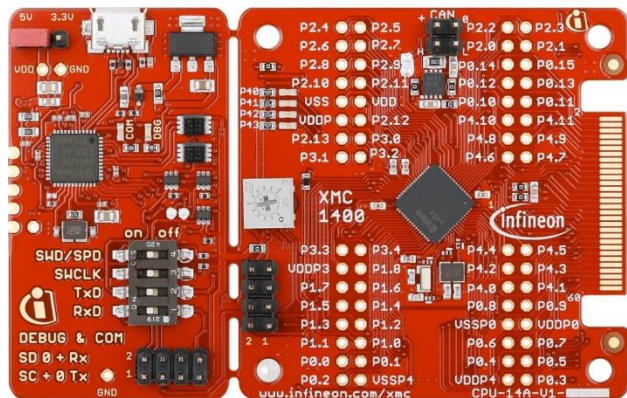


このドキュメントについて

- › このドキュメントでは、統合開発環境ModusToolboxを使用して XMC™ マイクロコントローラでPWMによるLEDの点滅プロジェクトを作成する手順について説明するものです。
- › このドキュメントではPCにModusToolbox 2.3がインストールされている事を前提としています。

使用するハードウェア(Kit)について

- このドキュメントで作成するプロジェクトを動作させるターゲットハードウェアとして KIT_XMC14_BOOT_001を前提としています。
- 他のKitを使用する場合、プロジェクト生成時のKit又はマイコンの型格を合わせて設定してください。また、出力先のLEDのPin番号も合わせて設定してください。

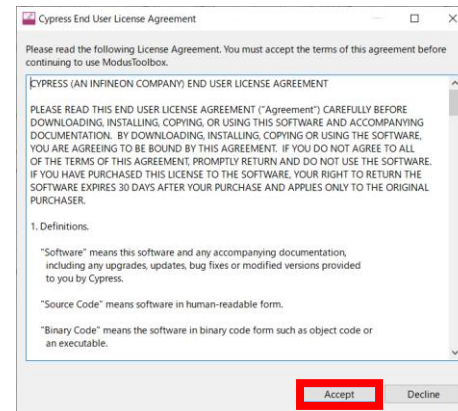
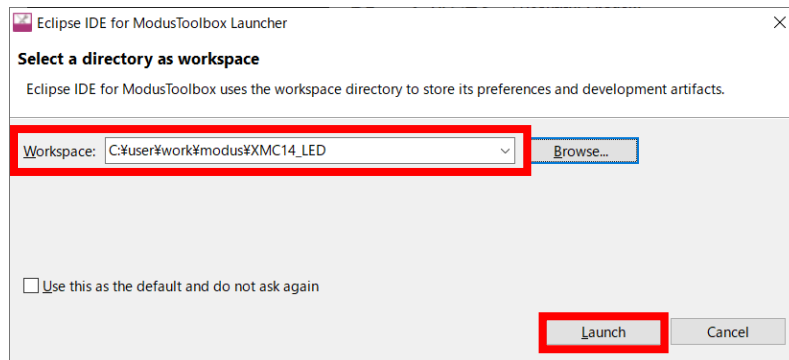
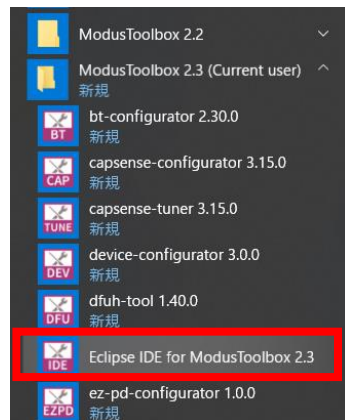


- KIT_XMC14_BOOT_001
- https://www.infineon.com/cms/jp/product/evaluation-boards/kit_xmc14_boot_001/

プロジェクト作成手順

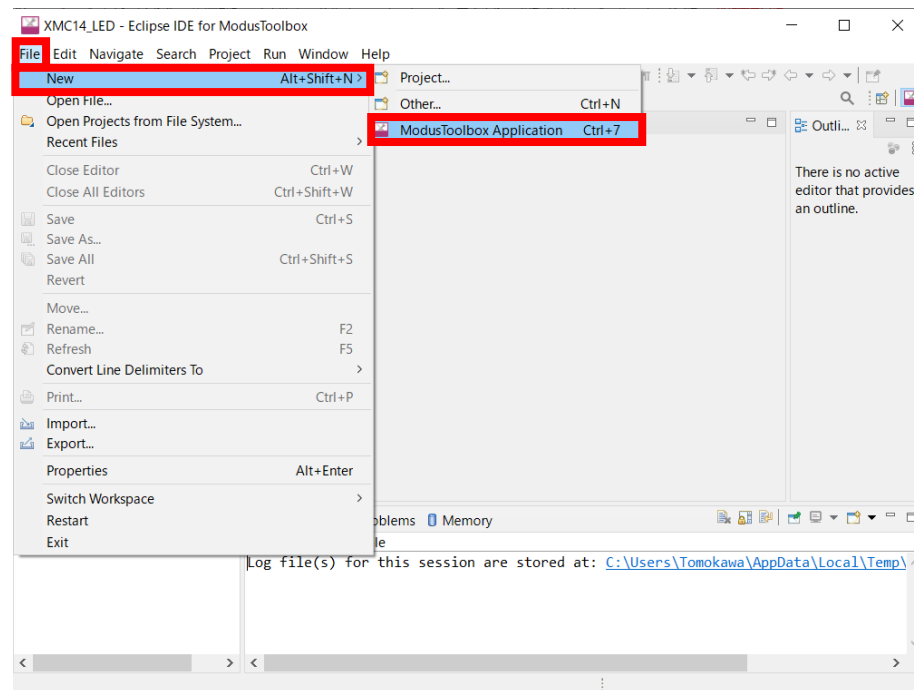
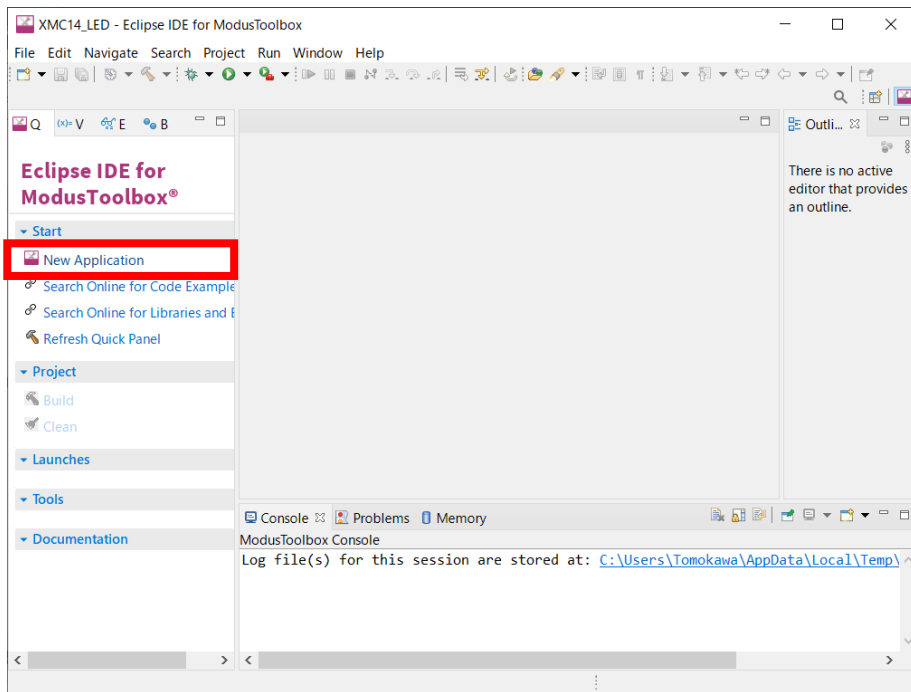
ModusToolboxの起動

1. スタートメニューよりEclipse IDE for ModusToolbox 2.3をクリックしてModusToolboxを起動します。
2. プロジェクトを格納するワークスペース・フォルダを指定します。
Note: フォルダ名、Pathには、マルチバイト文字を使用しないでください。
3. Launchをクリックします。
Note: End User License Agreementのダイアログが表示された場合はAcceptをクリックしてください。



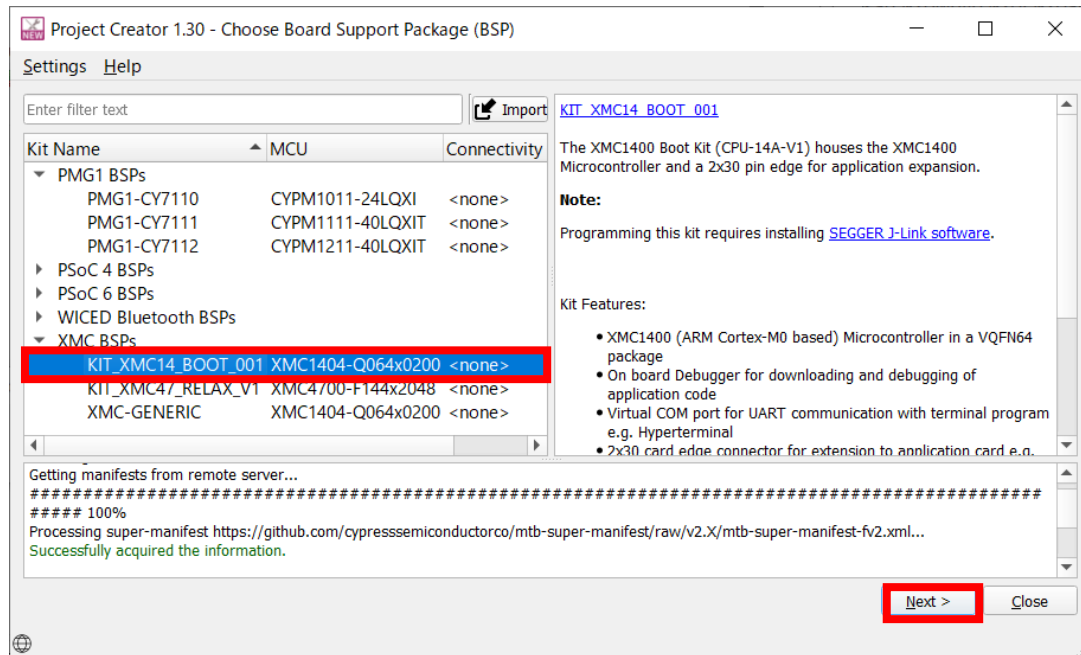
プロジェクトの新規作成(1/3)

1. Quick panelのNew Application又は、File → New → ModusToolbox Applicationを選択します。



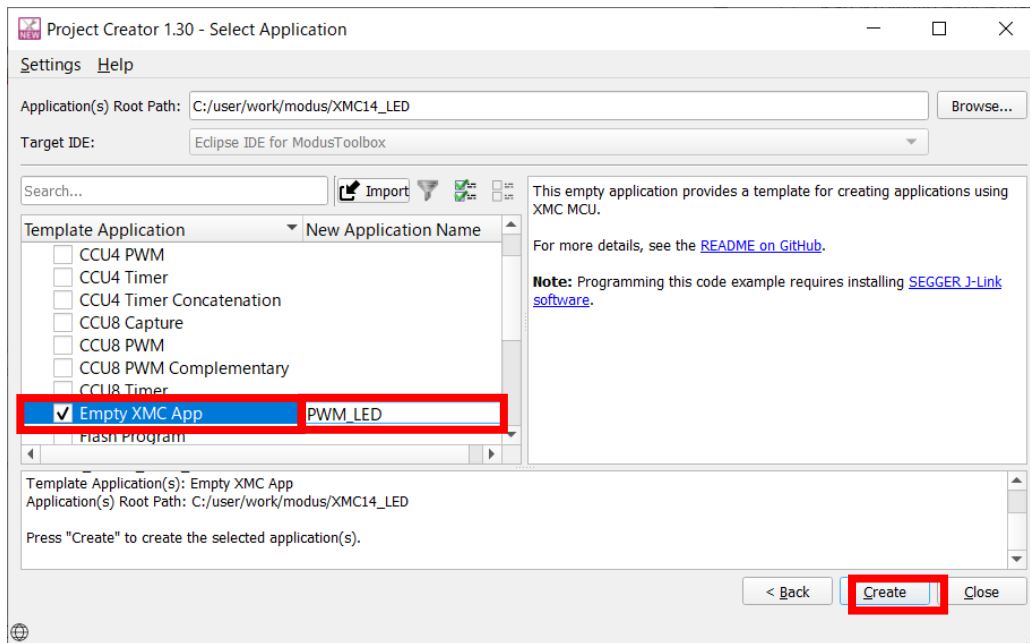
プロジェクトの新規作成(2/3)

2. XMC BSPs(Board Support Package)のKIT_XMC14_BOOT_001を選択しNextをクリックします。



プロジェクトの新規作成(3/3)

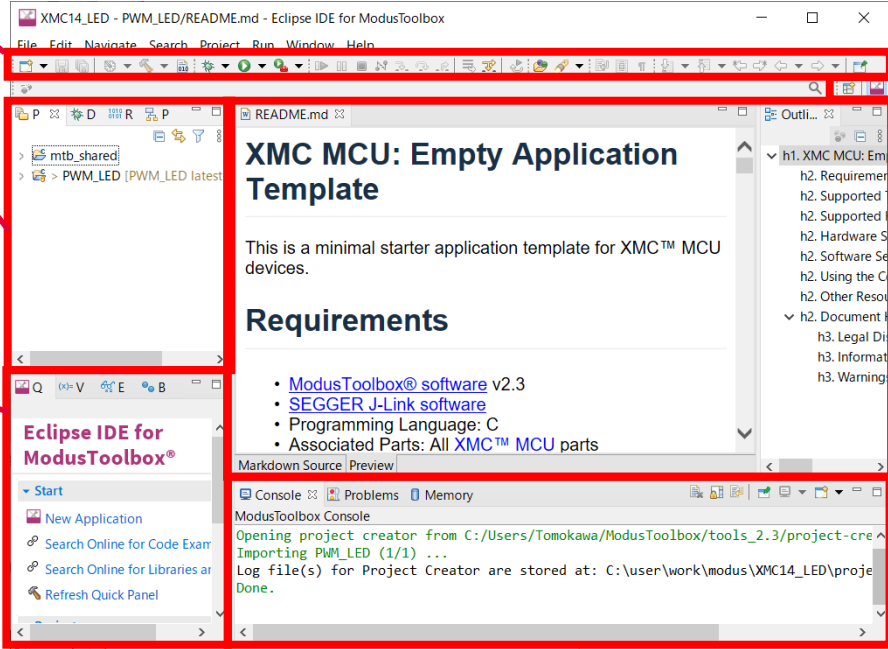
3. Template ApplicationでEmpty XMC Appをチェックします。
4. New Application NameにPWM_LEDを入力します。
5. Createをクリックします。



Note: 使用するペリフェラルに応じたテンプレートが用意されており、CCU4 PWMを選択することで、編集することなくPWMによるLED点滅が可能です。このドキュメントでは空のテンプレートを指定して作業を行います。

ModusToolBoxパースペクティブ

- プロジェクトを生成するとModusToolboxパースペクティブ(画面レイアウト)が表示されます。

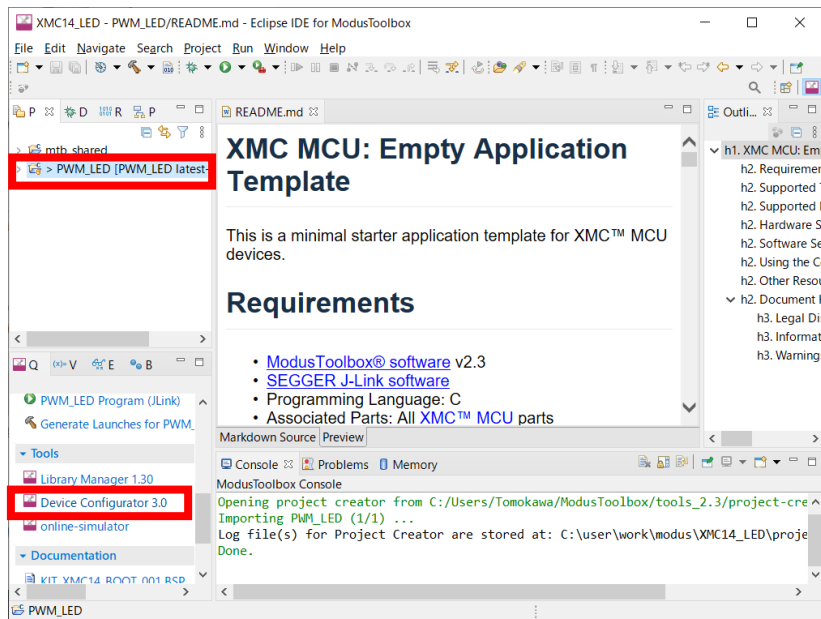


The screenshot shows the Eclipse IDE for ModusToolbox with a project named 'XMC14_LED'. The interface is divided into several panels, each highlighted by a red box and a callout:

- Tool Panel**: Located at the top left, it contains icons for various development tools.
- Project Explorer**: Located on the left side, it shows the project structure with 'mtb_shared' and 'PWM_LED [PWM_LED latest]'.
- Quick Panel**: Located at the bottom left, it provides quick access to various actions like 'New Application', 'Search Online for Code Examples', and 'Search Online for Libraries and Components'.
- Perspective (画面レイアウト)の切替**: Located at the top right, it shows icons for switching between different workspace layouts.
- Code editor**: The central area, labeled 'XMC MCU: Empty Application Template', where the source code is edited.
- Console / Memory**: Located at the bottom right, it displays the console output and memory view.

Device Configuratorの起動

- ▶ PWM信号は、XMC™ マイクロコントローラのCCU4(Capture/Compare Unit 4) または CCU8ペリフェラルを使用して生成します。
- ▶ ペリフェラルの設定はModusToolboxのDevice Configuratorを使用していきます。

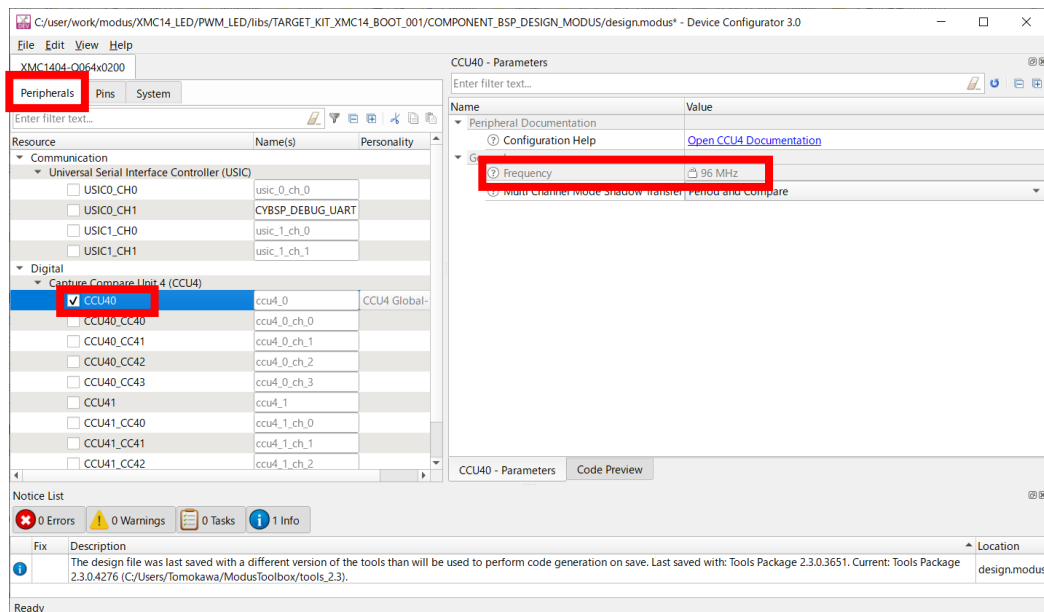


1. Project ExplorerでPWM_LEDプロジェクトを選択します。
2. Quick PanelでDevice Configurator をクリックします。

Note: Project ExplorerでPWM_LEDプロジェクトを選択することでQuick PanelにDevice Configuratorが現れます。

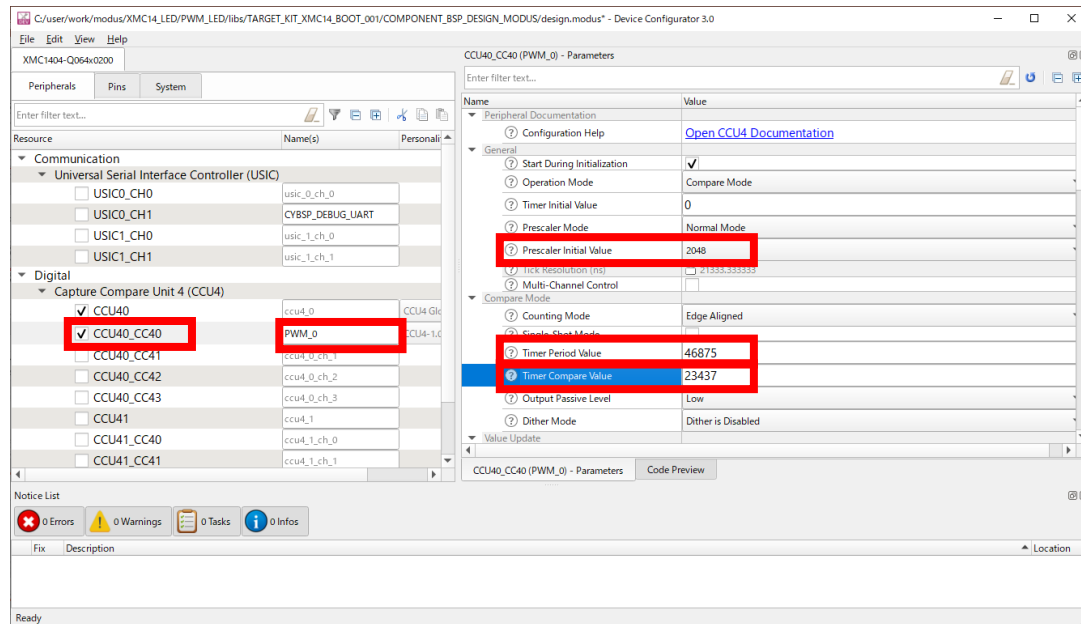
PWMの設定(1/2)

- ▶ XMC1400には、2つのCCU4が搭載(インスタンス)されており、CCU4は、CC0～3の4つのスライスを持っています。
- ▶ このプロジェクトではCCU4のインスタンス0, スライス0を使用します(CCU40.CC0)



3. Device Configurator の Peripheralsタブをクリックします。
4. CCU40をチェックします。
5. 右側のCCU40 – Parameters で、Frequencyが96MHzである事を確認してください。その他特に設定するものではありません。

PWMの設定(2/2)



6. CCU40_CC40をチェックし、Name(s)にPWM_0を設定します。
7. CCU40_CC40 – Parameters で、以下の項目を設定します。

Name	Value
Prescaler Initial Value	2048
Timer Period Value	46875
Timer Compare Value	23437

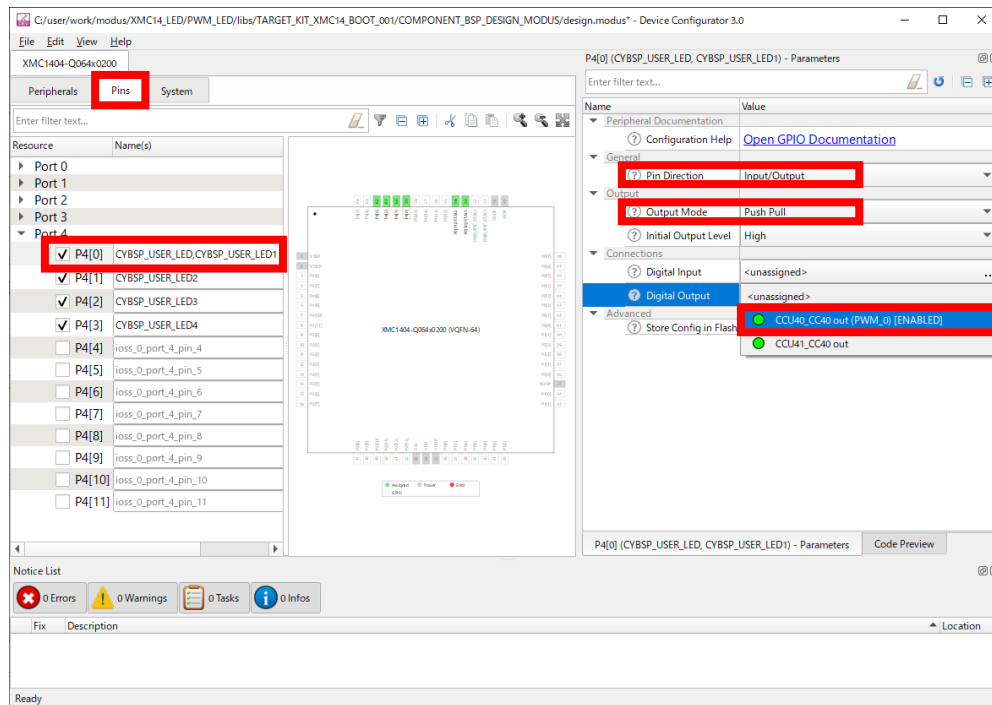
Note: 1秒間隔でLEDを点滅させるため、96MHzで駆動しているCCU40をPrescalerで2048分周した46875が1秒の期間となります。

$$96000000 / 2048 = 46875$$

$$46875 * 0.5 = 23437.5 \quad // \text{ Duty比50\%}$$

Pinマッピング

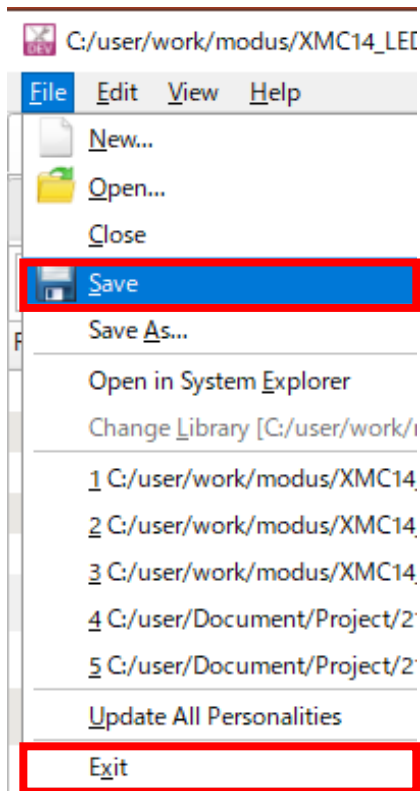
- PWM出力をLEDに接続されたPinに接続します。
- KIT_XMC14_BOOT_001の場合、P4[0]がLED101となるのでP4[0]に割り当てます



1. Device ConfiguratorのPinsタブをクリックします
2. P4[0]をチェックし、Name(s)にCYBSP_USER_LED,CYBSP_USER_LED1を設定します。
3. 右側のP4[0] – Parametersで以下の項目を設定します。

Name	Value
Pin Direction	Input/Output
Output Mode	Push Pull
Digital Output	CCU40_CC40 out

Device Configuratorの終了



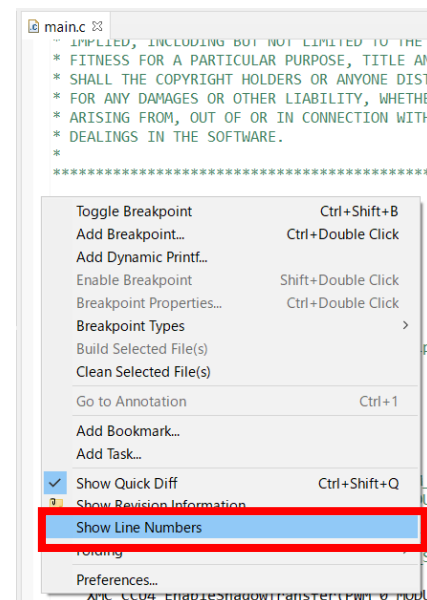
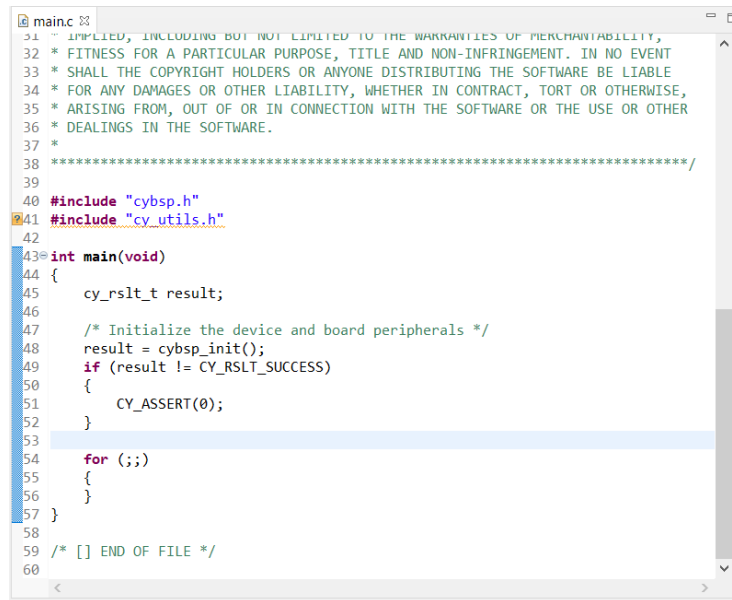
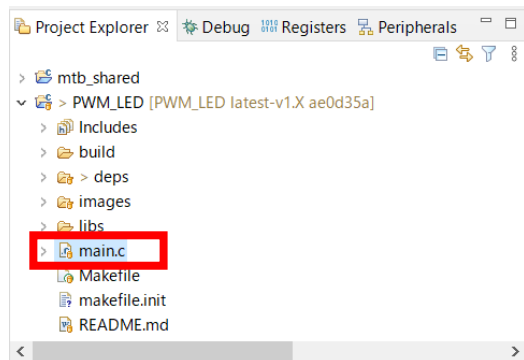
1. File → Saveを選択します。
2. File → Exitを選択してDevice Configuratorを終了します

Note: Device Configuratorの起動中はプロジェクトファイルにLockがかかりますので、ModusToolboxの作業に戻る際には Device Configuratorを必ず終了させてください。

コード編集(1/2)

1. Project explorer 内のmain.cをダブルクリックしてコードを表示します。

Note: 行番号を表示するには、コードエディタViewの左端でマウスを右クリックしてメニューを表示し、Show Line Numbersを選択します。



コード編集(2/2)

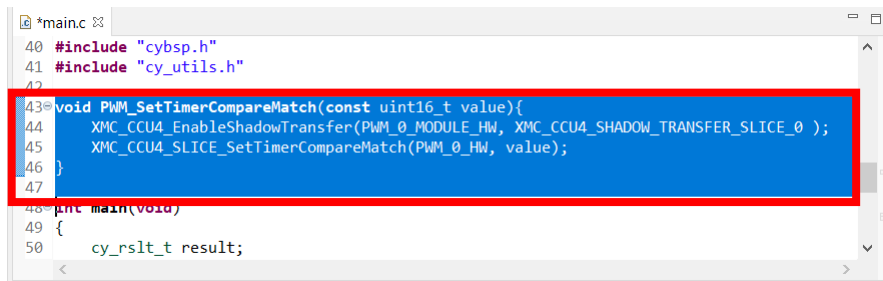
- main.c内の43-47行にDuty cycleを設定する関数を記述します。

```
void PWM_SetTimerCompareMatch(const uint16_t value){
    XMC_CCU4_EnableShadowTransfer(PWM_0_MODULE_HW, XMC_CCU4_SHADOW_TRANSFER_SLICE_0);
    XMC_CCU4_SLICE_SetTimerCompareMatch(PWM_0_HW, value);
}
```

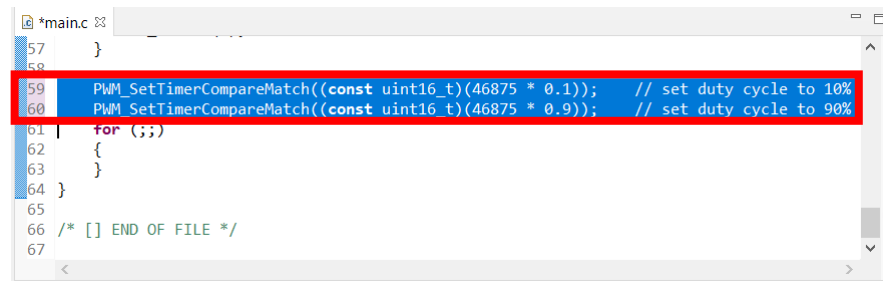
- main.c内の59-60行に関数を呼び出すステートメントを記述します。

```
PWM_SetTimerCompareMatch((const uint16_t)(46875 * 0.1));           // set duty cycle to 10%
PWM_SetTimerCompareMatch((const uint16_t)(46875 * 0.9));           // set duty cycle to 90%
```

- File → Saveを選択又はCtrl-SでファイルをSaveします。



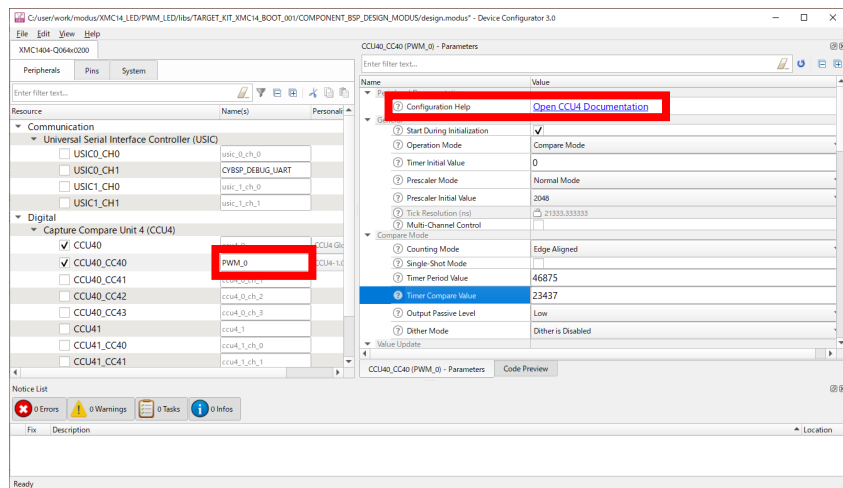
```
*main.c
40 #include "cybsp.h"
41 #include "cy_utils.h"
42
43 void PWM_SetTimerCompareMatch(const uint16_t value){
44     XMC_CCU4_EnableShadowTransfer(PWM_0_MODULE_HW, XMC_CCU4_SHADOW_TRANSFER_SLICE_0);
45     XMC_CCU4_SLICE_SetTimerCompareMatch(PWM_0_HW, value);
46 }
47
48 int main(void)
49 {
50     cy_rslt_t result;
```



```
*main.c
57 }
58
59 PWM_SetTimerCompareMatch((const uint16_t)(46875 * 0.1)); // set duty cycle to 10%
60 PWM_SetTimerCompareMatch((const uint16_t)(46875 * 0.9)); // set duty cycle to 90%
61 for (;;)
62 {
63 }
64 }
65
66 /* [] END OF FILE */
67
```

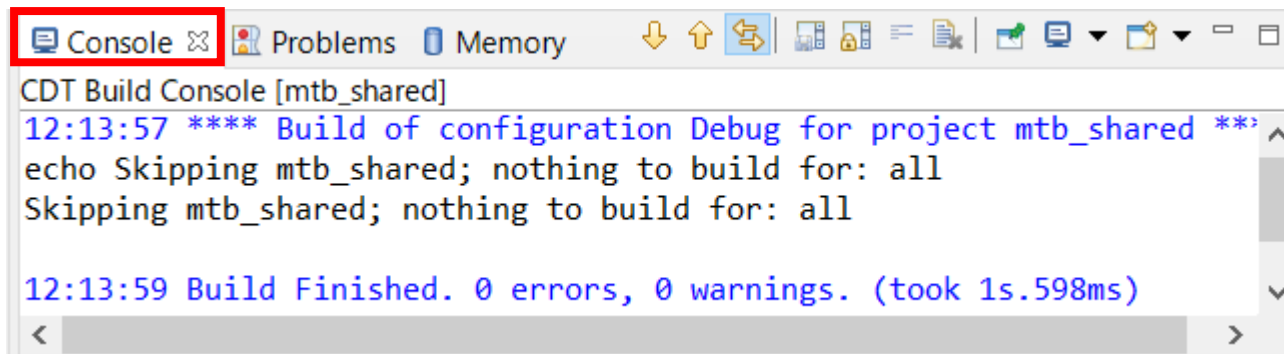

コード編集(補足)

- APIを使用してPWM(CCU40.CC0)にアクセスするには、Device ConfiguratorのName(s)で設定したPWM_0に_MODULE_HW又は_HWを付加したハンドラを指定することで行います。
- API及び定数や構造体についてはDevice ConfiguratorのConfiguration Helpに示されたLinkによりWebサイトを参照する事ができます。



プロジェクトのビルド

- Tool PanelのBuild Allアイコンをクリックしてプロジェクトをビルドします 
- ビルド結果はConsole / MemoryのConsoleタブで確認できます。



The screenshot shows the IDE's interface with the 'Console' tab selected and highlighted by a red box. The console output displays the following text:


```
CDT Build Console [mtb_shared]
12:13:57 **** Build of configuration Debug for project mtb_shared ****
echo Skipping mtb_shared; nothing to build for: all
Skipping mtb_shared; nothing to build for: all

12:13:59 Build Finished. 0 errors, 0 warnings. (took 1s.598ms)
```

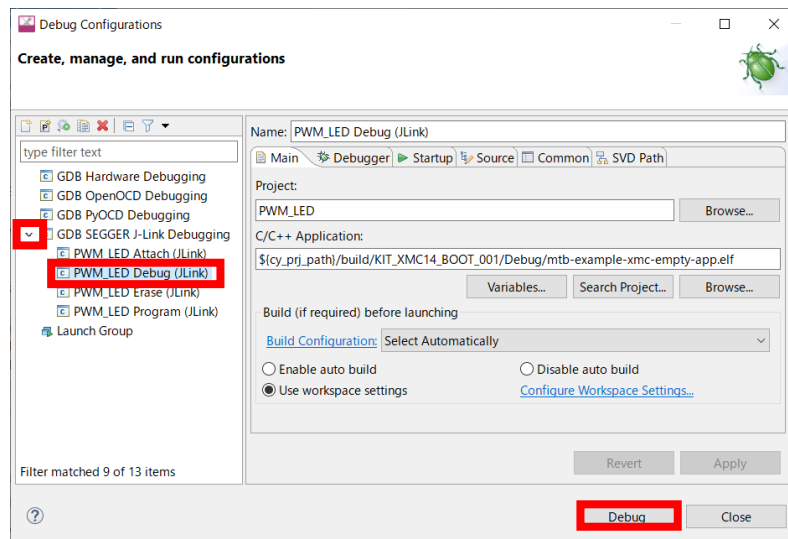
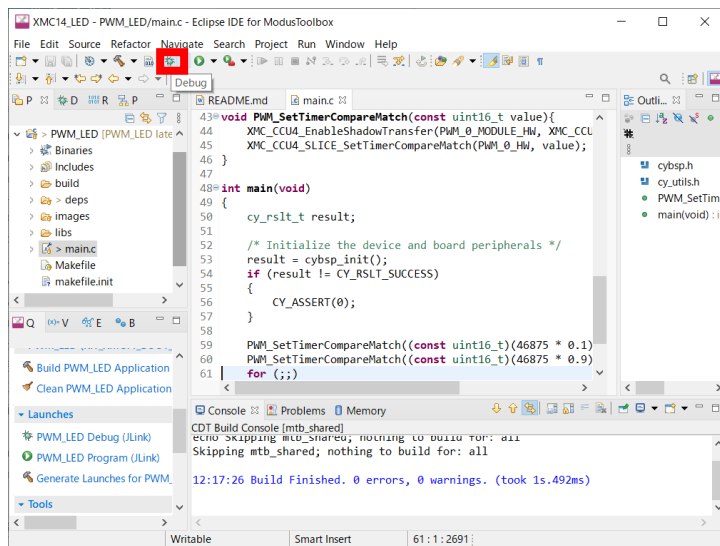
プログラムの書き込み 及びDebug

-
- LED101
- XMC1400
Microcontroller
- Debugger

Debuggerの起動

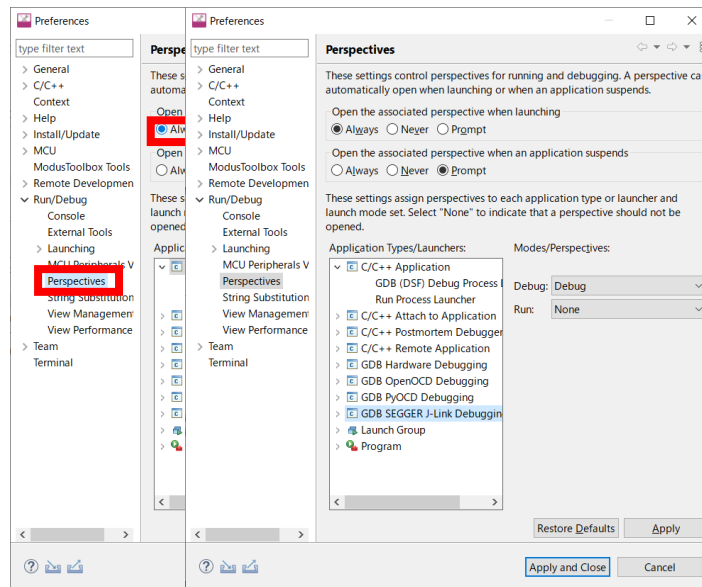
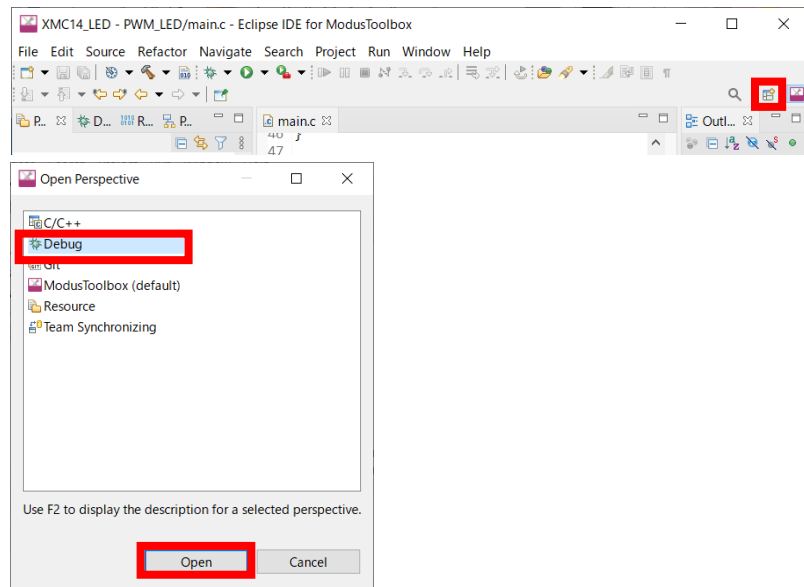
- ▶ Tool PanelのDebugアイコンをクリックしてDebuggerを起動します 

 1. プロジェクトで初めてDebugを行う場合、Debug Configuration生成画面が表示されるのでGDB SEGGER J-Link Debuggingの左端をクリックして配下のメニューを広げます。
 2. PWM_LED Debug(JLink)をクリックして選択し、Debugをクリックします。



Debugパースペクティブへの切り替え

- ▶ ModusToolboxパースペクティブのままDebugを行う事も可能ですが、Debugger使用時に適したDebugパースペクティブへの切り替えを行います。
 1. PerspectiveのOpen Perspectiveアイコンをクリックします。
 2. Open Perspectiveダイアログが表示されますのでDebugを選択してOpenをクリックします。

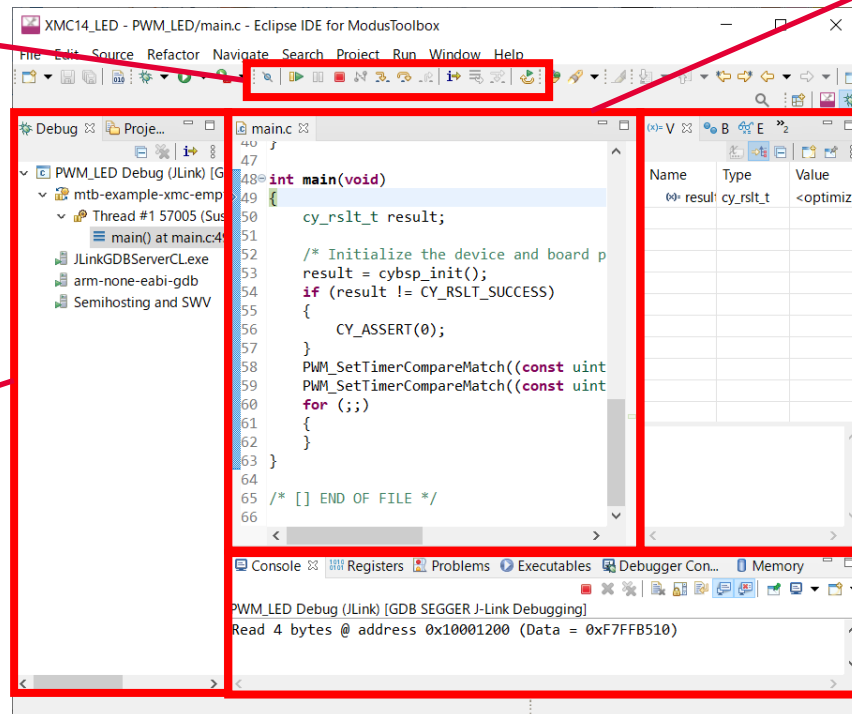


Window →
Perspectivesを選択して
ダイアログを表示し、
Debugの
ViewにあるOpen
Associated
をalwaysに設
定とで自動切り
替定されます。

Debugパースペクティブ各部説明

Debugger Action
プログラムの実行/停止
操作アイコン

Debug View
Debugのスレッド情報
表示



Editor View
ソースコードレベルで
の実行位置表示及びブ
レークポイントの設定/
解除

**Variable/Breakpoint
View**
変数/ブレークポイント
の監視及び設定

**Memory/Register
View**
メモリー/レジスタの監
視及び変更

Debug(1/9)

- Debugセッションが開始されるとプログラムが書き込まれ、main()関数先頭のステートメントまで自動的に実行されます。

Note: main()関数の先頭に自動的にブレークポイントが設定されます。このブレークポイントはDebugger ConfigurationのStartupタブにあるSet Breakpoint at:の項目で指定されています。

```

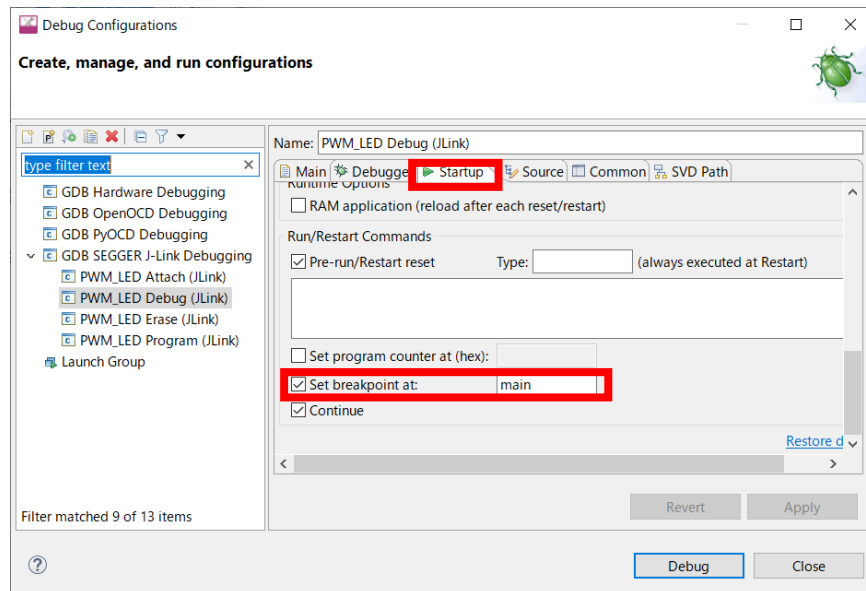
48 int main(void)
49 {
50     cy_rslt_t result;

```

```

main.c
47
48 int main(void)
49 {
50     cy_rslt_t result;
51
52     /* Initialize the device and board peripherals */
53     result = cybsp_init();
54     if (result != CY_RSLT_SUCCESS)
55     {

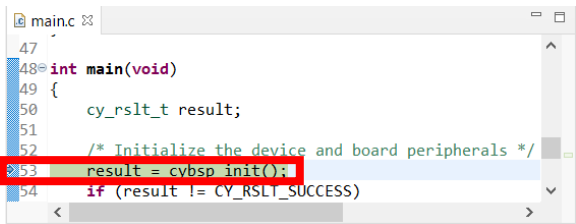
```



Debug(2/9)

1. Debugger Action のStep Into アイコンをクリックします

最初のステートメントまで実行されたことを確認します。

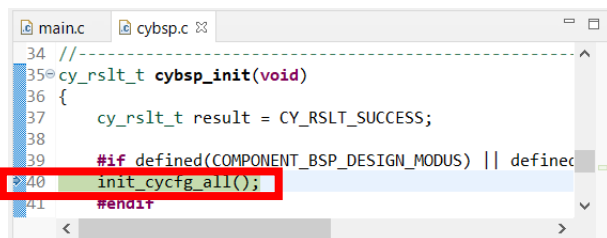


```

47
48 int main(void)
49 {
50     cy_rslt_t result;
51
52     /* Initialize the device and board peripherals */
53     result = cybsp_init();
54     if (result != CY_RSLT_SUCCESS)
  
```

2. 再度Debugger Action のStep Into アイコンをクリックします

Editor Viewにcybsp.cが表示され、cybsp_init()関数内の最初のステートメントまで実行されたことを確認します。



```

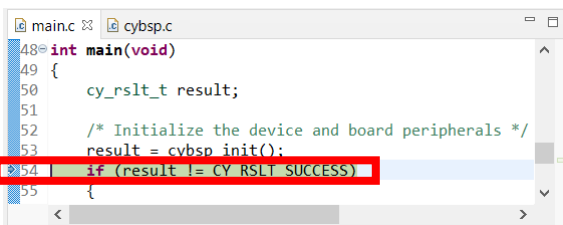
34 //-----
35 cy_rslt_t cybsp_init(void)
36 {
37     cy_rslt_t result = CY_RSLT_SUCCESS;
38
39     #if defined(COMPONENT_BSP_DESIGN_MODUS) || defined(COMPONENT_BSP_DESIGN_MODUS)
40     init_cycfg_all();
41     #endif
  
```

Note: Step Into アイコンはクリックする度に1ステートメントを実行しますが、実行するステートメントに関数が含まれている場合、その関数内部に移動します。

Debug(3/9)

3. Debugger Action のStep Return アイコンをクリックします

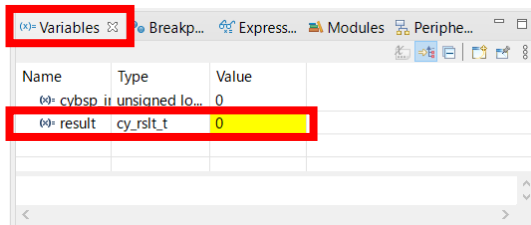
cybsp_init()関数の残りのステートメントが全て実行され、main()関数のcybsp_init()関数呼び出しステートメントの次まで戻っていることと、Variable Viewでresult変数のValueが0に設定されている事を確認します。また、ターゲットハードウェアのLED101が点滅していることを確認します。



```

48 int main(void)
49 {
50     cy_rslt_t result;
51
52     /* Initialize the device and board peripherals */
53     result = cybsp_init();
54     if (result != CY_RSLT_SUCCESS);
55     {

```

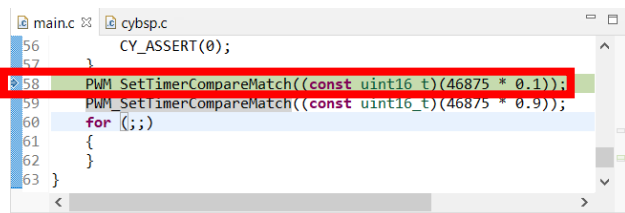


Name	Type	Value
cybsp	unsigned lo...	0
result	cy_rslt_t	0

Note: Step Return アイコンは現在実行中の関数の呼び出し元まで全てのステートメントを実行します。

4. Debugger Action のStep Into アイコンをクリックします

PWM_SetTimerCompareMatch()関数の呼び出しステートメントまで進んだことを確認します。



```

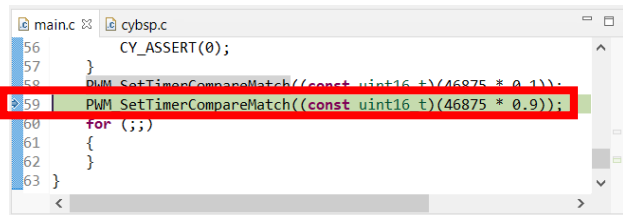
56     CY_ASSERT(0);
57 }
58 PWM_SetTimerCompareMatch((const uint16_t)(46875 * 0.1));
59 PWM_SetTimerCompareMatch((const uint16_t)(46875 * 0.9));
60 for (;;)
61 {
62 }
63 }

```

Debug(4/9)

5. Debugger Action のStep Over アイコンをクリックします

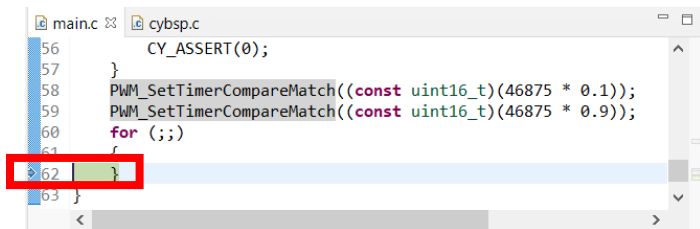
PWM_SetTimerCompareMatch()関数内に移動せず、次のステートメントまで実行されたことを確認します。
また、LED101の点灯期間が短くなっている事を確認します。



Note: Step Over アイコンは、実行するステートメントに関数が含まれている場合、その関数内部の全てのステートメントを実行します。

6. Debugger Action のStep Over アイコンをクリックします

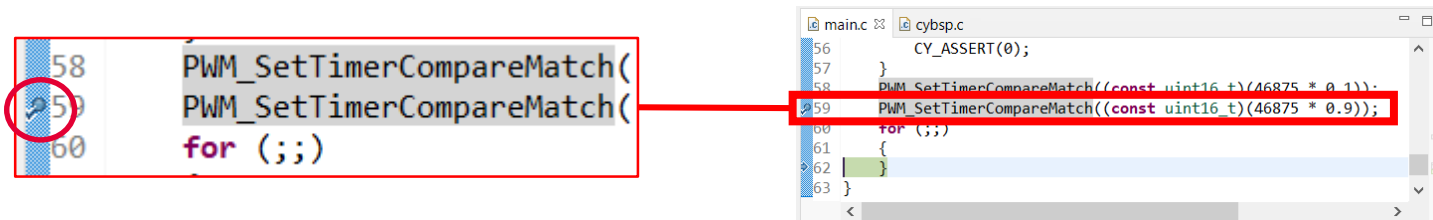
PWM_SetTimerCompareMatch()関数内に移動せず、次のステートメントまで実行されたことを確認します。
また、LED101の点灯期間が長くなっている事を確認します。



Debug(5/9)

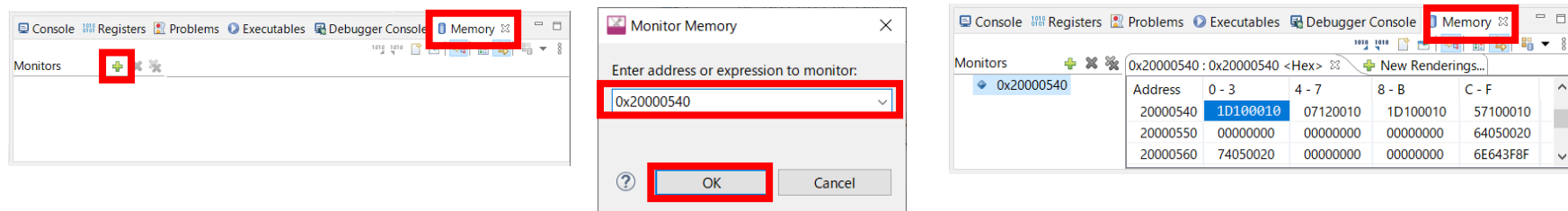
- Editor Viewの59行目の行番号の左側にある水色のライン上でダブルクリックしてブレークポイントを設定します。

Note: ブレークポイントを設定した行を再度ダブルクリックすることでブレークポイントを解除できます。



- Memory View のAdd Memory Monitorアイコンをクリックし、Monitor Memoryダイアログに0x20000540を入力してOKをクリックします +

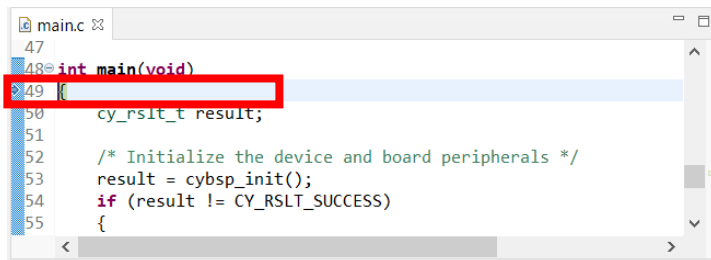
Note: Console表示が行われた場合、Memoryタブを選択して表示を切り替えます。



Debug(6/9)

9. Debugger Action のRestart アイコンをクリックします

Resetが行われ、main()関数先頭のステートメントに戻ったことを確認します。



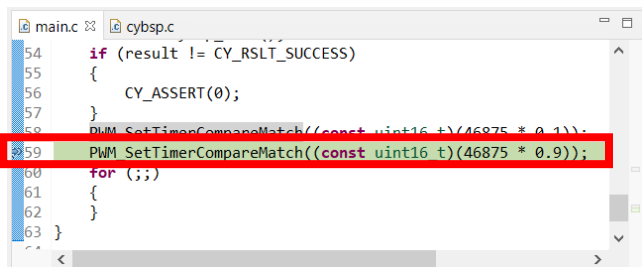
```

47
48: int main(void)
49: {
50:     cy_rslt_t result;
51
52:     /* Initialize the device and board peripherals */
53:     result = cybsp_init();
54:     if (result != CY_RSLT_SUCCESS)
55:     {

```

10. Debugger Action のResume アイコンをクリックします

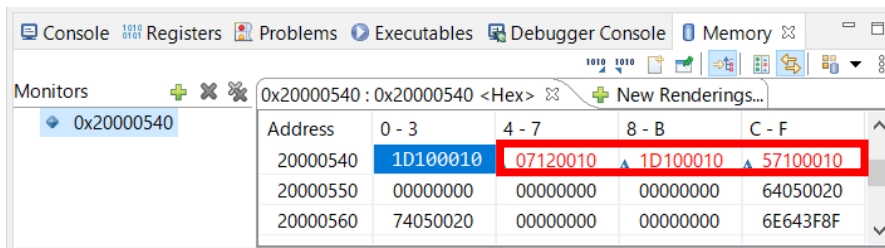
ブレークポイントを設定した行まで実行されたことと、LED101の点灯期間が短くなっている事を確認します。
また、実行中に変更されたメモリー内容が赤く表示されることをMemory Viewで確認します。



```

54: if (result != CY_RSLT_SUCCESS)
55: {
56:     CY_ASSERT(0);
57: }
58: PWM_SetTimerCompareMatch((const uint16_t)(46875 * 0.1));
59: PWM_SetTimerCompareMatch((const uint16_t)(46875 * 0.9));
60: for (;;)
61: {
62: }
63: }

```

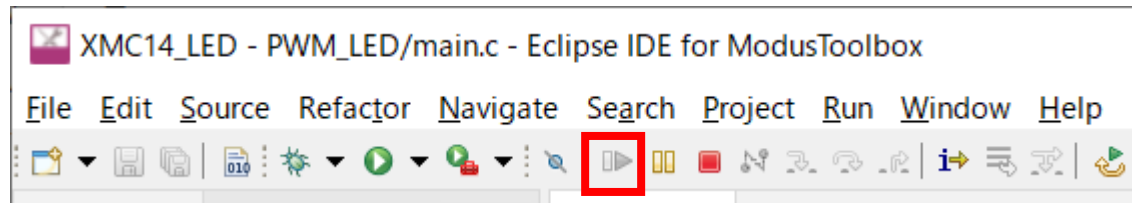


Address	0 - 3	4 - 7	8 - B	C - F
20000540	1D100010	07120010	1D100010	57100010
20000550	00000000	00000000	00000000	64050020
20000560	74050020	00000000	00000000	6E643F8F

Debug(7/9)

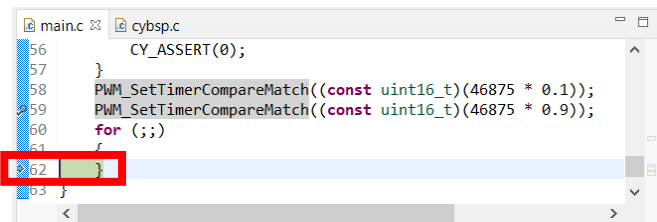
11. 再度Debugger Action のResume アイコンをクリックします

Resumeアイコンがグレーアウトし、連続実行されていることと、LED101の点灯期間が長くなっている事を確認します。



12. Debugger Action のSuspend アイコンをクリックします

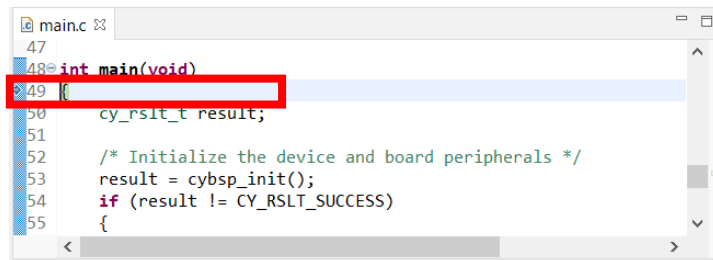
main()関数の最後まで実行されたことをEditor Viewで確認します。



Debug(8/9)

13. Debugger Action のRestart アイコンをクリックします

Resetが行われ、main()関数先頭のステートメントに戻ったことを確認します。



```

47
48: int main(void)
49 {
50     cy_rslt_t result;
51
52     /* Initialize the device and board peripherals */
53     result = cybsp_init();
54     if (result != CY_RSLT_SUCCESS)
55     {

```

14. Debugger Action のSkip All Breakpoints アイコンをクリックします

アイコンの色が変わりEnableになっていることを確認します。

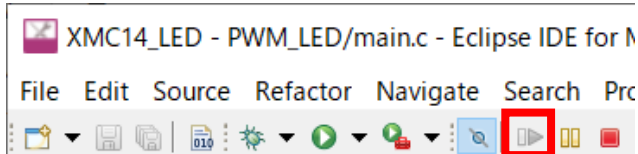
Note:再度クリックするとアイコンの色が戻りDisableとなります。



Debug(8/9)


15. Debugger Action のResume アイコンをクリックします

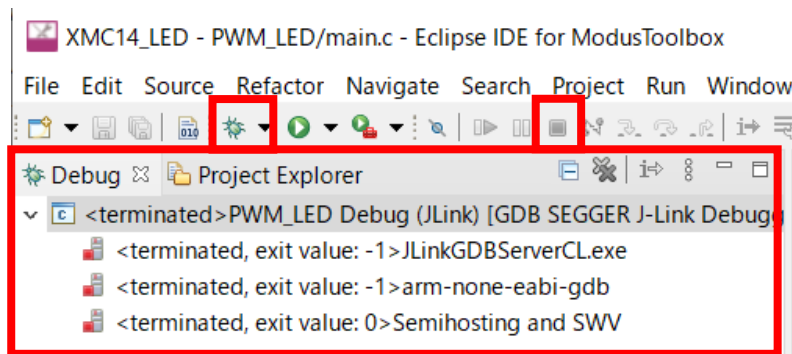
Resumeアイコンがグレイアウトし、ブレークポイントで停止せずに連続実行されていることを確認します。




16. Debugger Action のTerminate アイコンをクリックします

Debugger Actionのアイコンがグレイアウトし、Debug ViewのスレッドがTerminateされていることを確認します。

Note: Debugger ActionのDebugアイコンをクリックすることでデバッガーを再起動できます 












Debug(9/9)

17. PerspectiveのModusToolbox PerspectiveアイコンをクリックしてDebugセッションを終了します 

Note: Debugセッションを終了する際は必ずTerminateしてください。

Appendix

Debugger Actionアイコン一覧

アイコン	名称	説明
	Skip All Breakpoints	ブレークポイントを無効化します。ブレークポイントの削除は行いません。クリックする度にEnableとDisableをトグルします。
	Resume	プログラムを連続実行します。
	Suspend	実行中のプログラムを停止します。
	Terminate	デバッガーを停止します。 DAVE™ CEパースペクティブに移動する前に必ず停止してください。
	Step Into	クリックする度に1ステートメントを実行しますが、実行するステートメントに関数が含まれている場合、その関数内部に移動します。
	Step Over	実行するステートメントに関数が含まれている場合、その関数内部の全てのステートメントを実行します。
	Step Return	現在実行中の関数の呼び出し元まで全てのステートメントを実行します。
	Instruction Stepping Mode	ステップ実行をステートメント単位から機械語単位に切り替えます。クリックする度にEnableとDisableをトグルします。
	Restart	Resetが行われ、main()関数先頭のステートメントに戻ります。



Part of your life. Part of tomorrow.