Mi-V RV32 簡易チュートリアル

PolarFire SoC Discovery Kit 版

MACNICA

目次

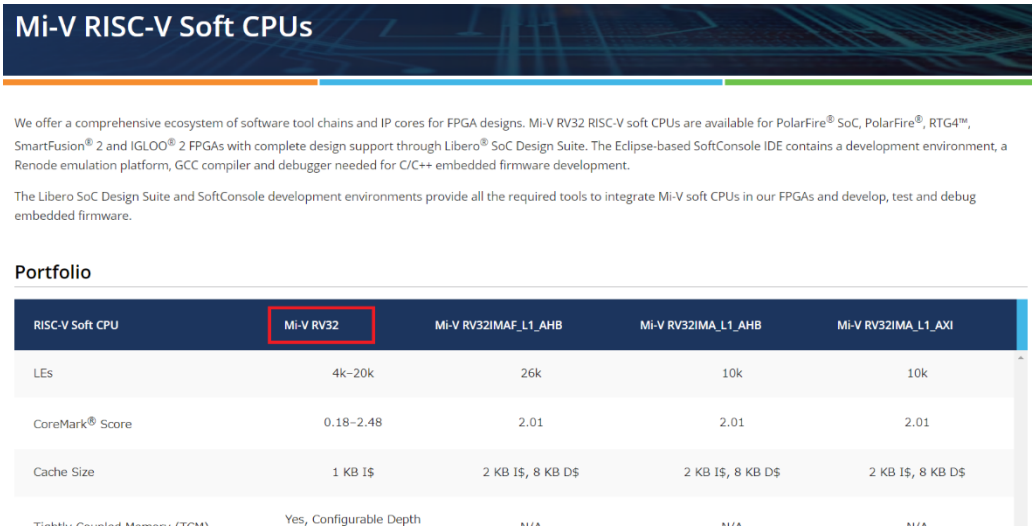
1. 概要

1-1. 本資料について

Mi-V(Microchip 社のソフト CPU)をはじめて使用される方を対象とした
簡易チュートリアルです。主に Libero SoC での新規プロジェクト作成から
Mi-V でプログラムを作成し LED を点滅させるまでの手順を記載します。
Libero SoC の使い方の説明は割愛しています。

1-2. Mi-V コア

Mi-V のコアには種類があります。本資料では最新コアである Mi-V RV32 を
使用します。



Mi-V RISC-V Soft CPUs

We offer a comprehensive ecosystem of software tool chains and IP cores for FPGA designs. Mi-V RV32 RISC-V soft CPUs are available for PolarFire® SoC, PolarFire®, RTG4™, SmartFusion® 2 and IGLOO® 2 FPGAs with complete design support through Libero® SoC Design Suite. The Eclipse-based SoftConsole IDE contains a development environment, a Renode emulation platform, GCC compiler and debugger needed for C/C++ embedded firmware development.

The Libero SoC Design Suite and SoftConsole development environments provide all the required tools to integrate Mi-V soft CPUs in our FPGAs and develop, test and debug embedded firmware.

Portfolio

RISC-V Soft CPU	Mi-V RV32	Mi-V RV32IMAF_L1_AHB	Mi-V RV32IMA_L1_AHB	Mi-V RV32IMA_L1_AXI
LEs	4k–20k	26k	10k	10k
CoreMark® Score	0.18–2.48	2.01	2.01	2.01
Cache Size	1 KB I$	2 KB I$, 8 KB D$	2 KB I$, 8 KB D$	2 KB I$, 8 KB D$
Tightly Coupled Memory (TCM)	Yes, Configurable Depth	N/A	N/A	N/A

1-3. 使用開発キット

PolarFire SoC Discovery Kit
※ペリフェラルは少ないですが比較的小さくて持ち運びやすい
　お手頃な開発キットです。
※今回ハード CPU の部分である MSS(Microprocessor Sub-System)は使用しません。

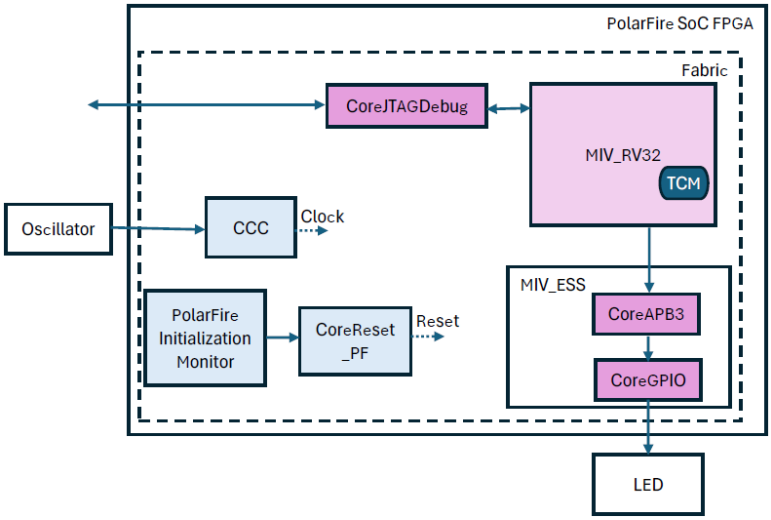
1-4. 動作確認バージョン

Libero SoC v2024.2
SoftConsole v2022.2

1-5. 動作させるデザイン

PolarFire SoC Discovery Kit の LED1、LED2、LED3、LED4 を
0101、1010 と交互に点灯、消灯させます。

構成:



2. SoftConsole のインストール（未インストールの場合）

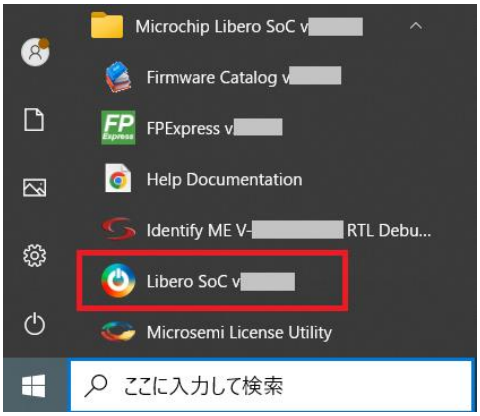
・Mi-V のソフトウェアは SoftConsole にて開発します。
・未インストールの場合は事前にダウンロード、インストールします。
・SoftConsole のバージョンは LiberoSoC のバージョンと連動していません。
2025 年 2 月時点では v2022.2 が最新です。
・SoftConsole はライセンス不要で使用可能です。

ダウンロードリンク:
https://www.microchip.com/en-us/products/fpgas-and-plds/fpga-and-soc-design-tools/soc-fpga/softconsole

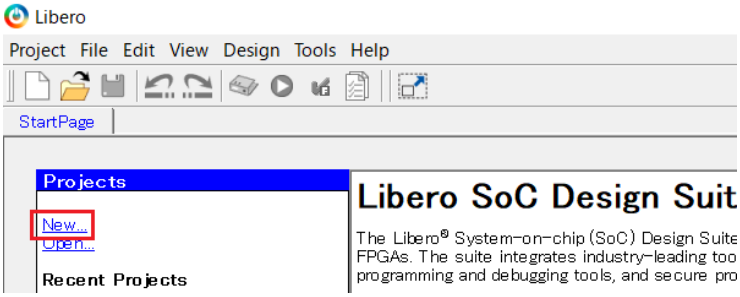
3. ハードウェア

3-1. Libero SoC プロジェクト作成

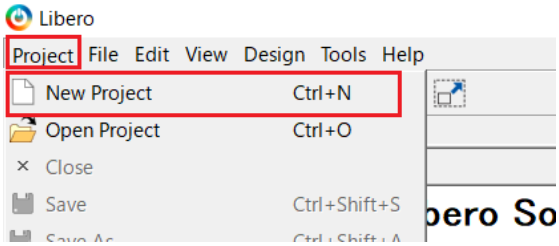
① Libero SoC を起動します。



② スタート・ページ ＞ “New”



もしくは Project > New Project



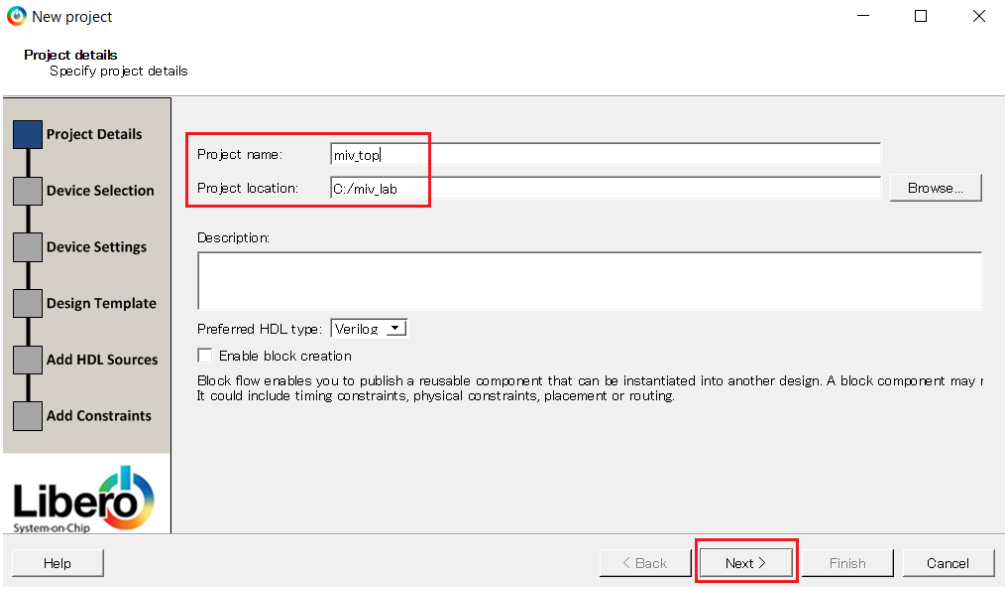
をクリックし、New Project ウィザードを開きます。

③ Project nam、Project location を設定し Next > を押します。

お手元の PC のご都合の良いフォルダへ設定して下さい。

Project name: miv_top

Project location: C:¥miv_lab

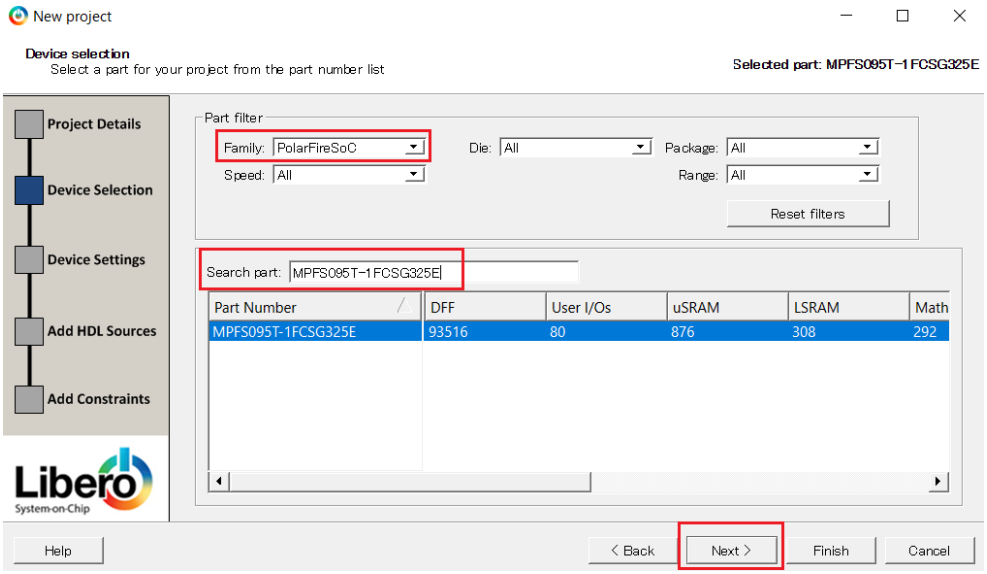


④ Device selection にて

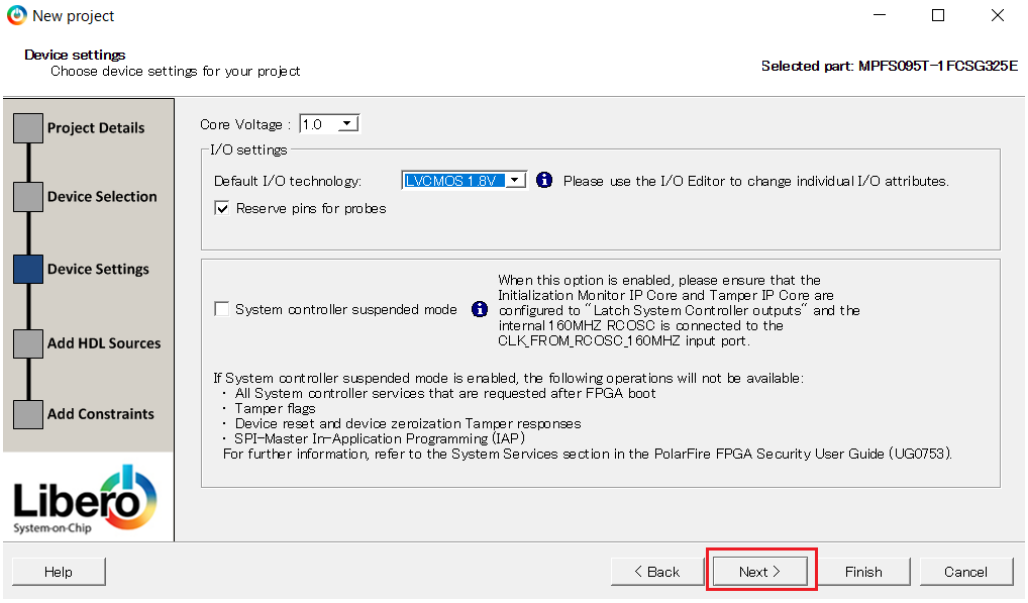
Family : PolarFireSoC を選択

Search part: MPFS095T-1FCSG325E を入力

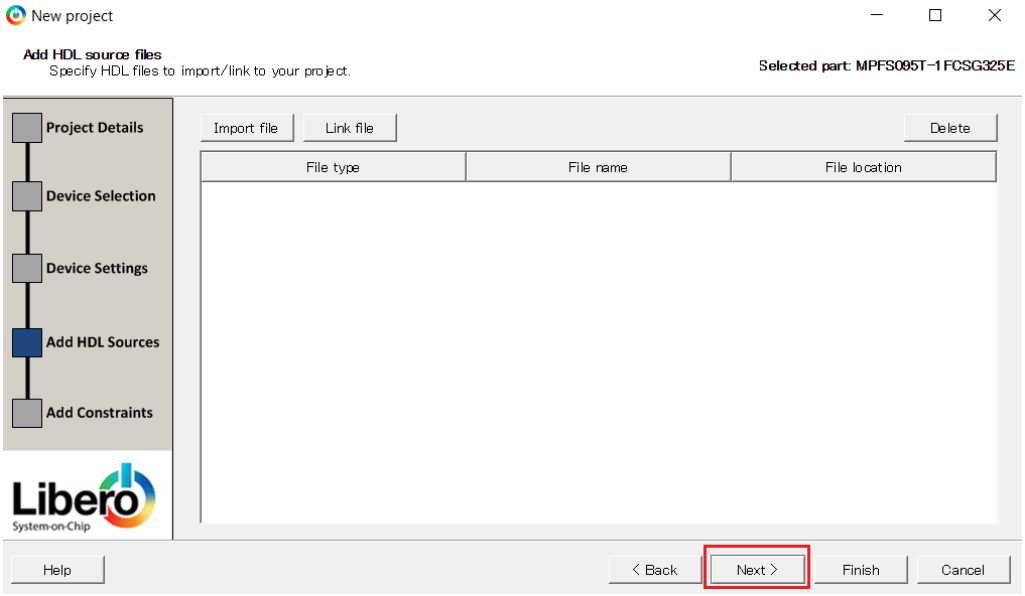
MPFS095T-1FCSG325E を選択し Next> を押します。



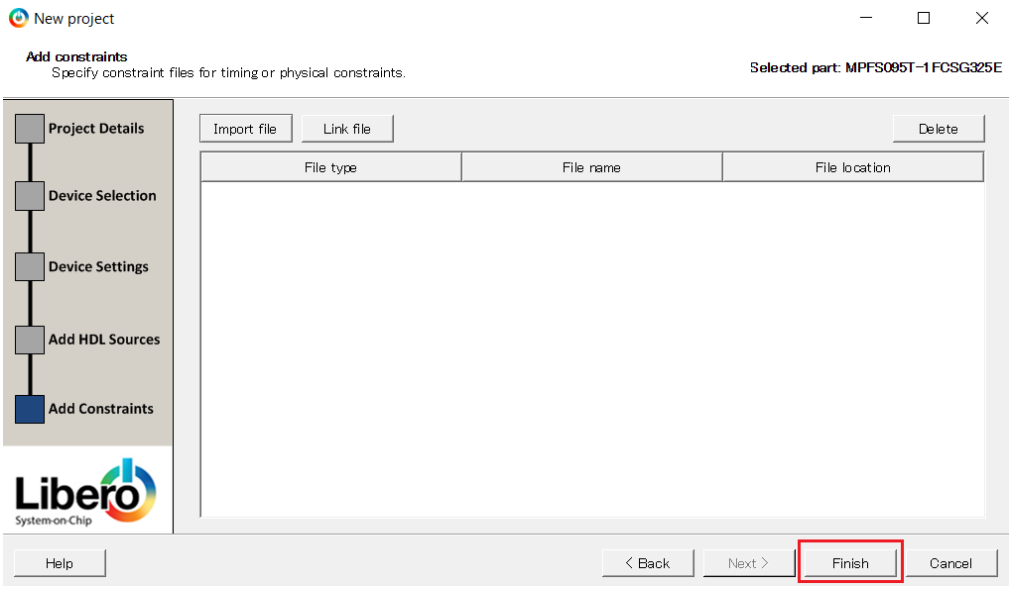
⑤ Device settings ページにてデフォルトのまま Next>を押します。



⑥ Add HDL source files にてデフォルトのまま Next> を押します。

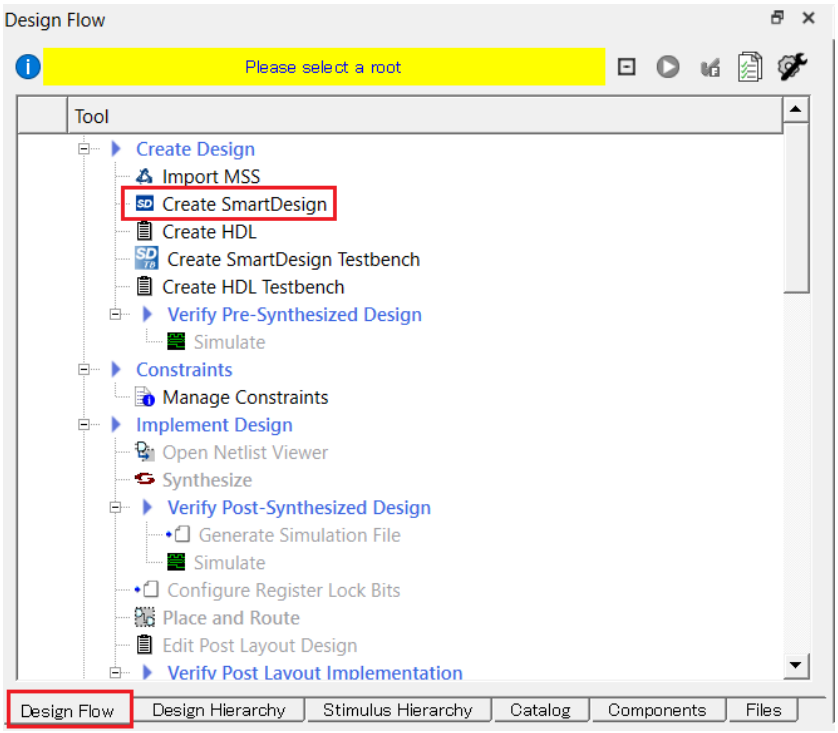


⑦ Add constraints にてデフォルトのまま Finish を押します。



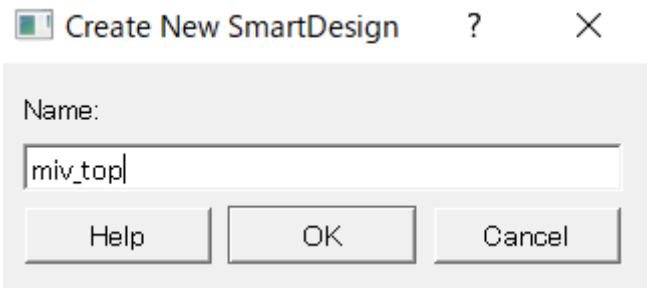
3-2. SmartDesign の新規作成

① 本トレーニングでは回路図エディタにてデザインを作成します。
Design Flow 内にて Create SmartDesign をダブルクリック。

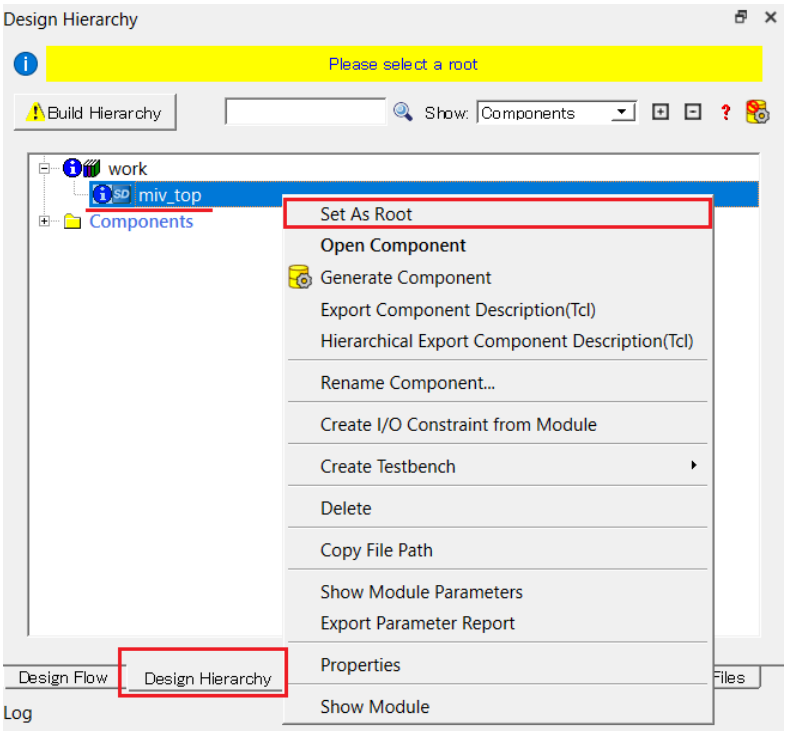


② 名前を入力し OK を押します。

Name: miv_top

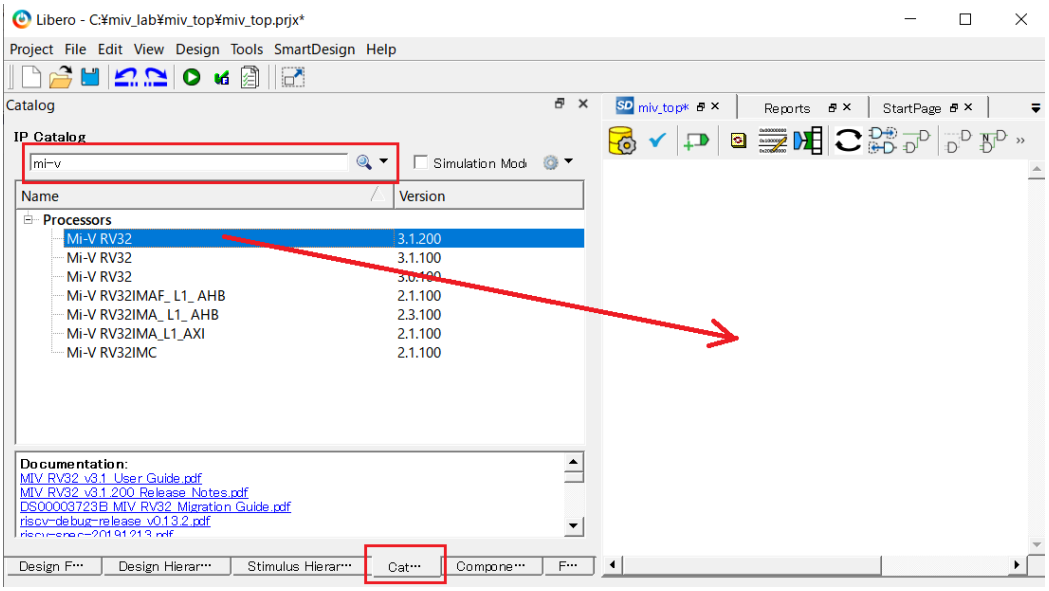


③ Design Hierarchy タブを開き、作成した SmartDesign を右クリック、Set As Root を選選択します。

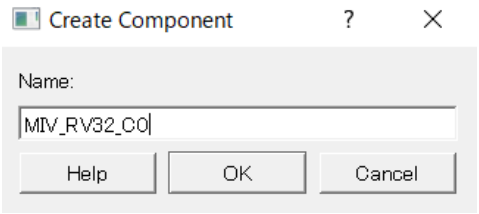


3-3. Mi-V RV32

① まずソフト CPU Mi-V のプロセッサーコアを置きます。
Catalog タブを開き Processors 内の Mi-V RV32 を
Smart Design 上へドラッグ&ドロップします。



② Create Component ウィンドウが立ち上がります。
そのまま OK を押します。



③ Mi-V のプロセッサーコアを設定します。今回は下記のように設定します。

Interface Options

AHB Initiator : None

APB Initiator : APB3、 AHB Mirrored I/F へチェック

AXI Initiator : None

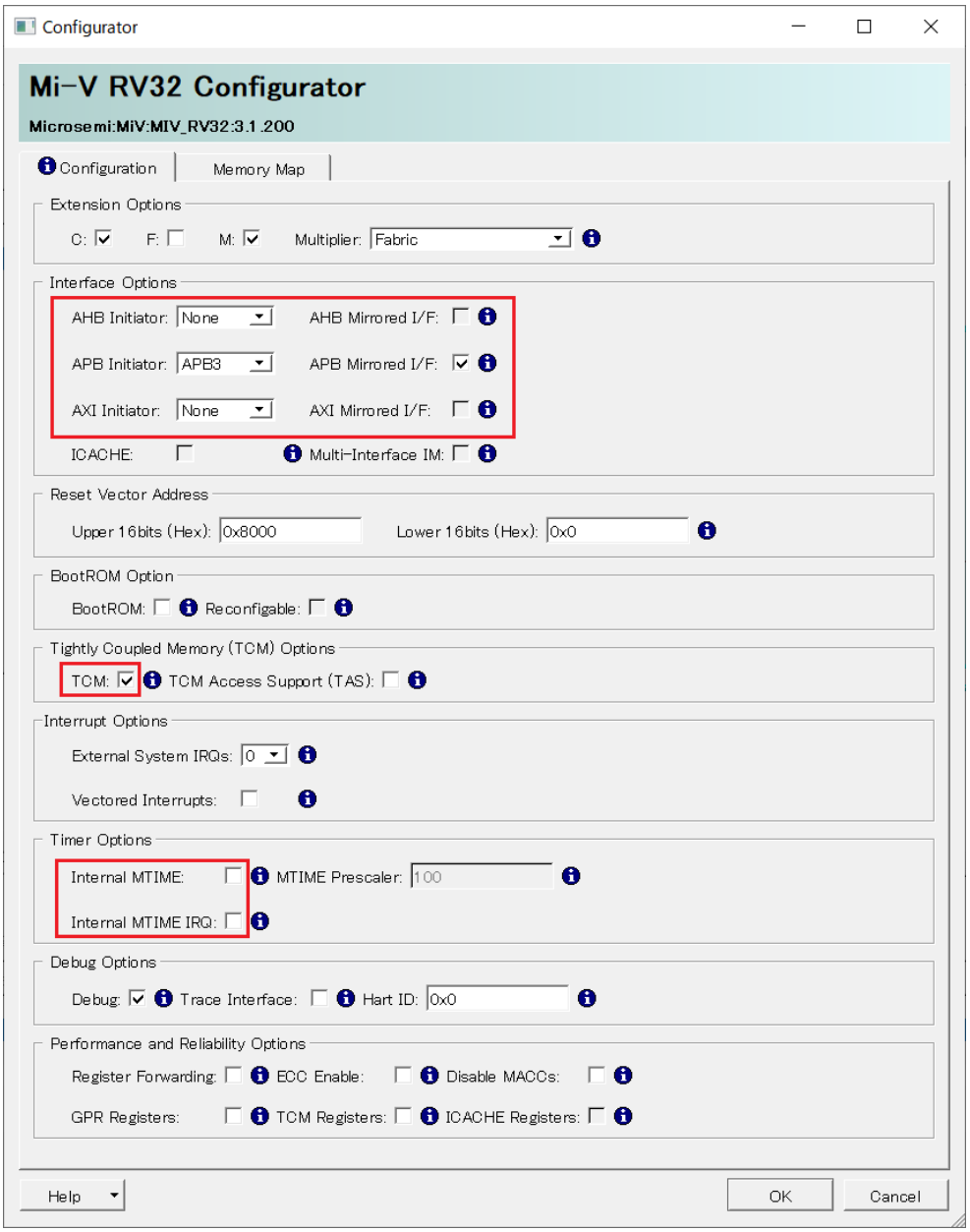
Tightly-Coupled Memory (TCM) Options

TCM : チェックを入れる

Timer Options

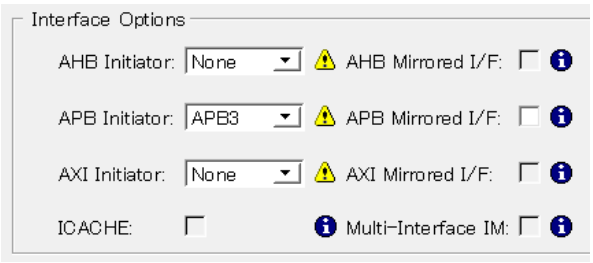
Internal MTIME: チェックを外す

Internal MTIME IRQ: チェックを外す



補足: Interface Options:

Mi-V RV32 プロセッサと GPIO を繋ぐため AMBA APB(Advanced Peripheral Bus)を有効にしています。

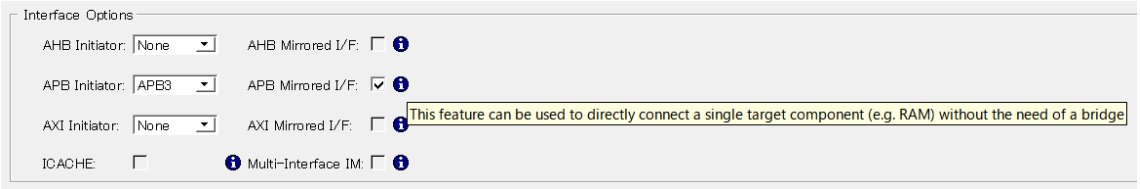


補足: APB Mirrored I/F について:

今回チェックを入れている APB Mirrored I/F についての説明は、

青い i マークにカーソルをあてると確認可能です。

"This feature can bw used to directly connect a single target component (e.g. RAM) without the need of a bridge"



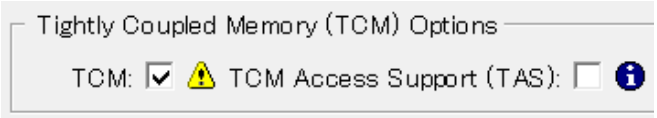
補足: Tightly-Coupled Memory (TCM) Options:

TCM は Tightly-Coupled Memory(密接合メモリ)の略になります。

本チュートリアルでは一例として TCM のアドレスを Mi-V の RESET_VECTOR_ADDRESS に設定します。

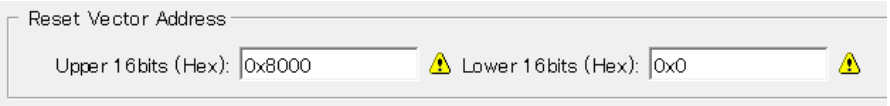
引用「The processor can be booted from this memory region by setting the RESET_VECTOR_ADDRESS to the address of the TCM.」

https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/UserGuides/ip_cores/directcores/MIVRV32v31UserGuide.pdf#page=15

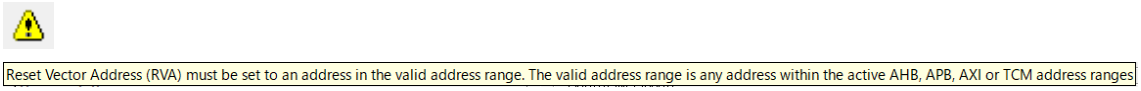


④ アドレスを変更します。

Reset vector はデフォルトでは 0x8000_0000 となっています。

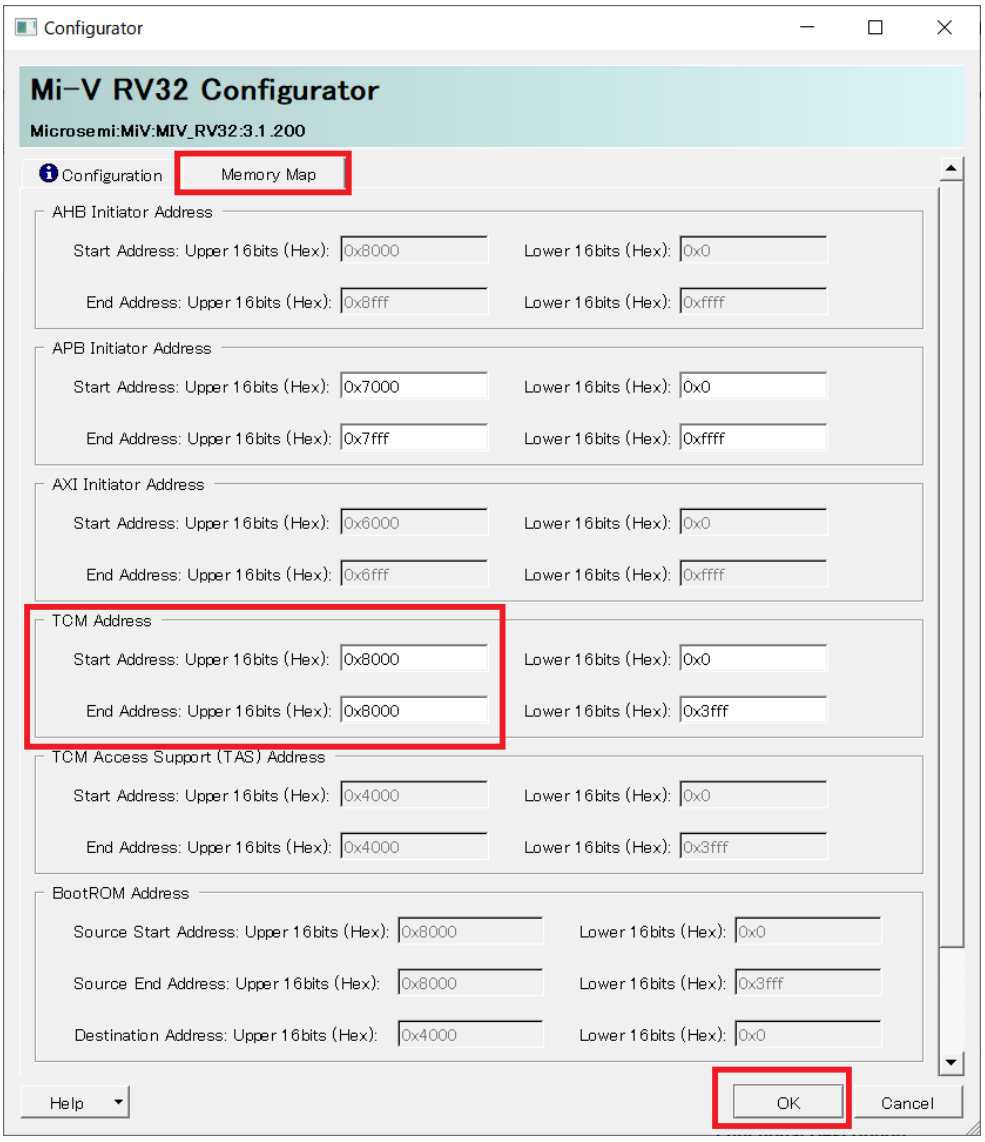


現在 0x8000_0000 がアドレス範囲外との警告が表示されているため変更対応します。

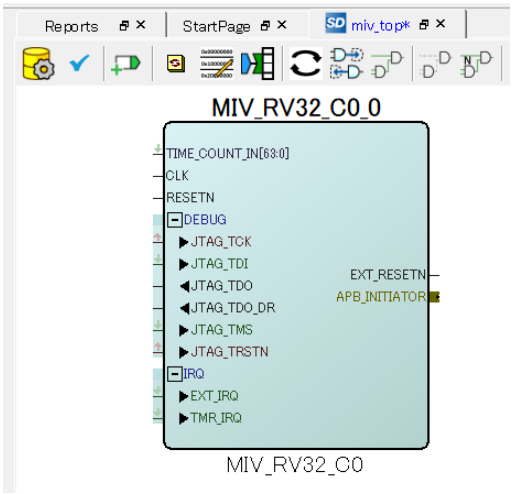


Memory Map タブを開き、

TCM Address の Upper 16bits (Hex)を 0x4000 から 0x8000 へ変更し OK を押します。

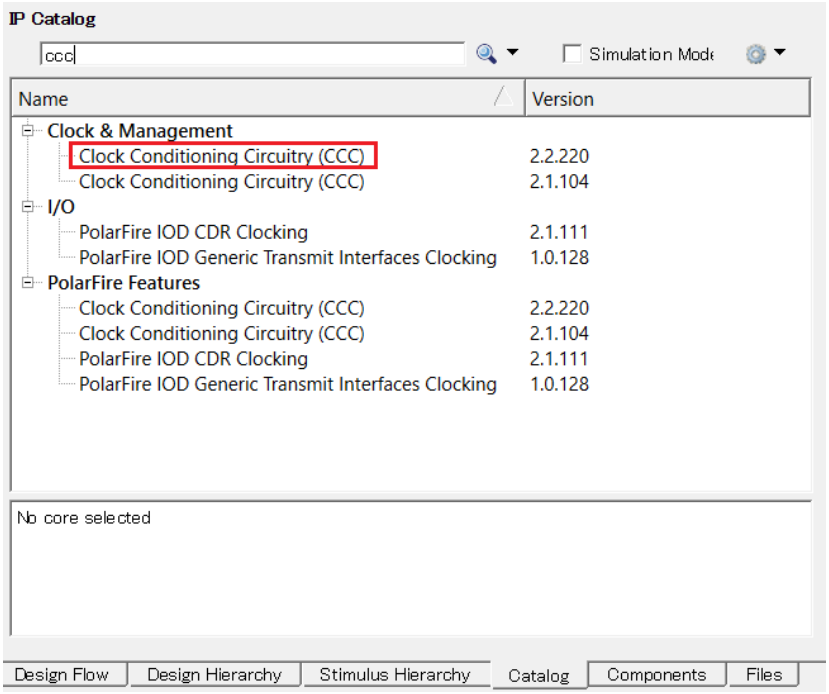


OK 押下後、Smart DesignMi-V RV32 コアのブロックが Smart Design 上に表示されます。

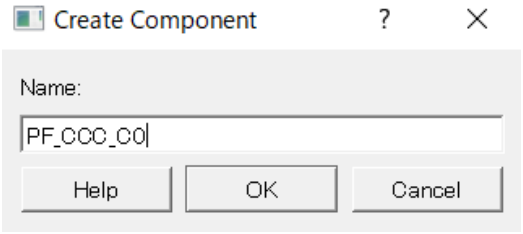


3-4. Clock Conditioning Circuitry (CCC)

① Catalog タブより CCC を選択し、SmartDesign 上へドラッグ&ドロップします。いわゆる PLL の IP になります。



② Create Component ウィンドウにてデフォルトのまま OK を押します。

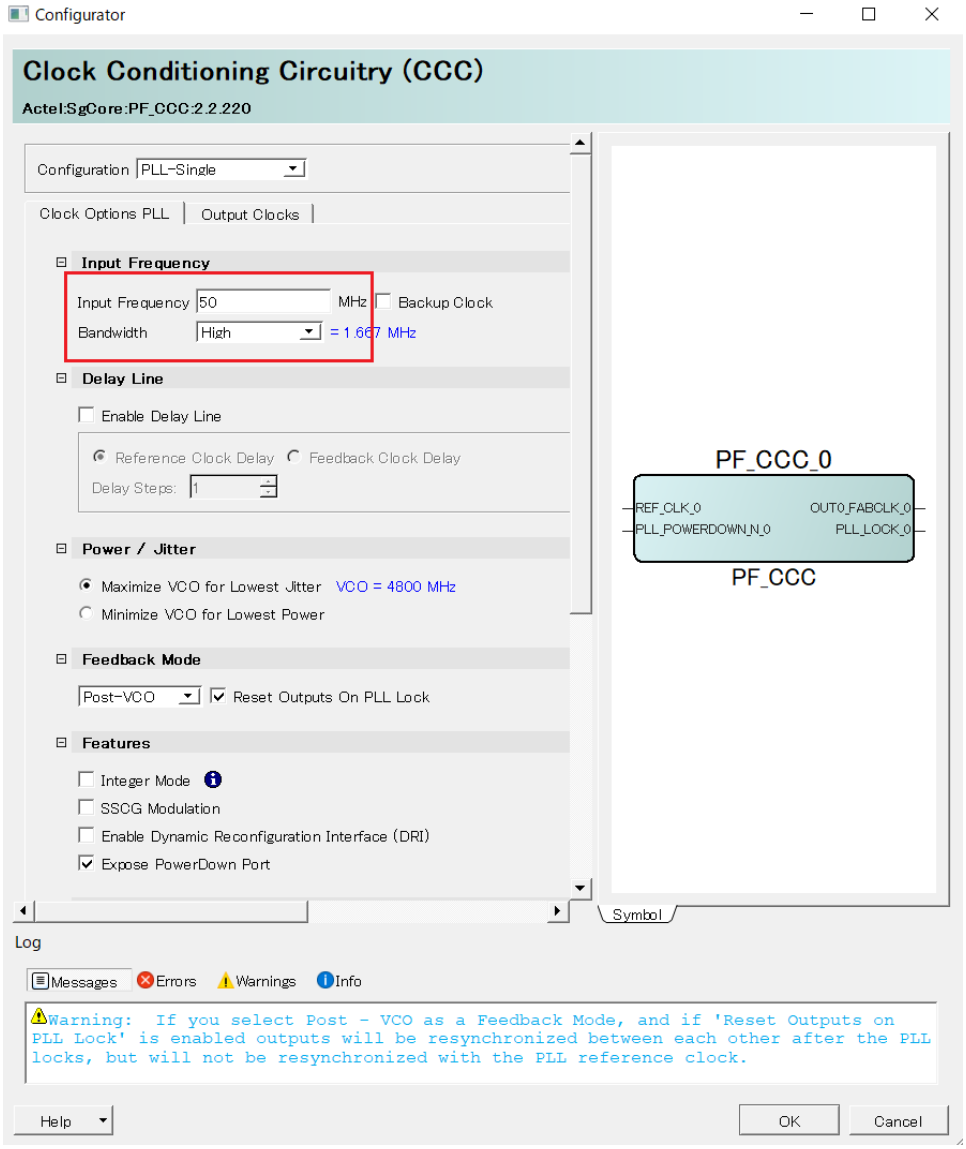


③ Input Frequency にて

Input Frequency : 50MHz

Bandwidth : High

に設定します。

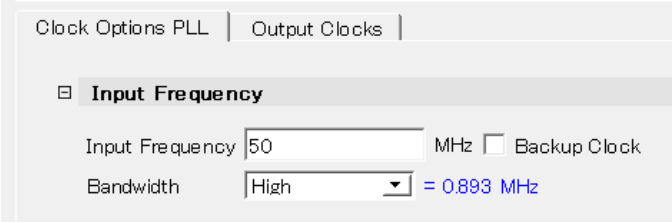


補足: CCC の Bandwidth 設定:

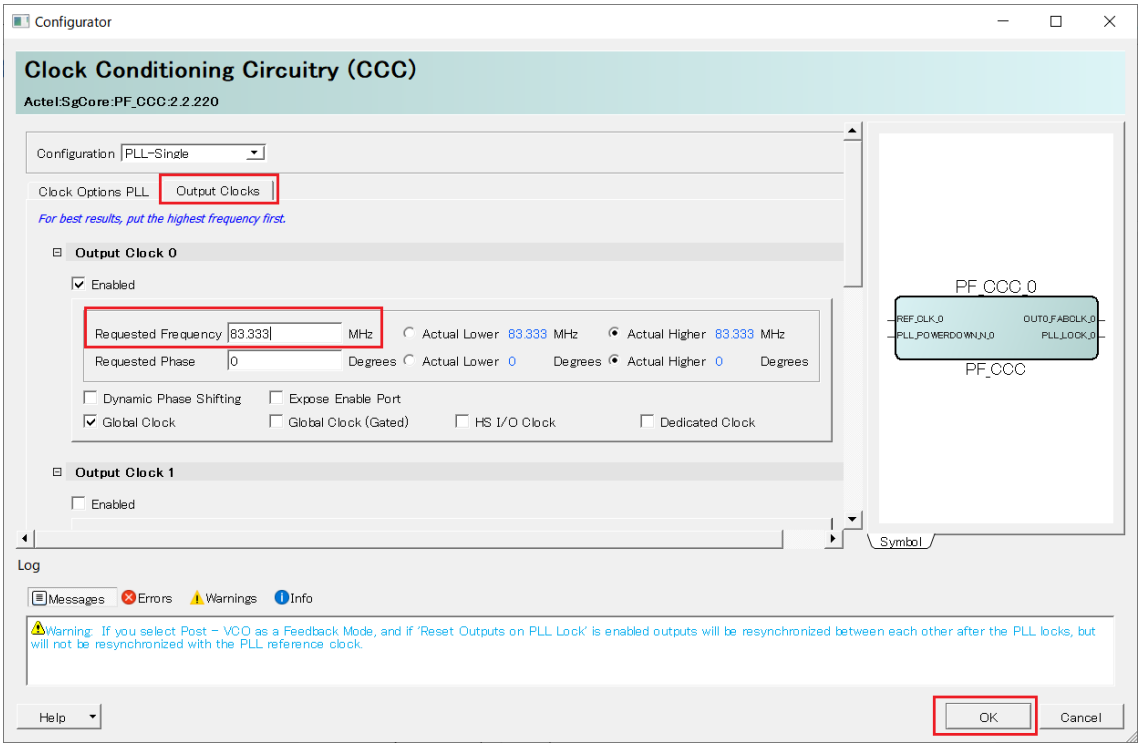
Bandwidth はリファレンスクロックの品質に応じて設定します。

引用「If the reference clock has a significant amount of jitter, use lower bandwidth to filter the noise. If a higher quality reference clock is used, fast lock time is achieved by using a higher bandwidth value.」

https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/UserGuides/Microchip_PolarFire_FPGA_and_PolarFire_SoC_FPGA_Clocking_Resources_User_Guide_VB.pdf#page=33



④ Output Clocks タブを開き、Output Clock 0 の Requested Frequency を 83.333MHz に設定し、OK を押します。



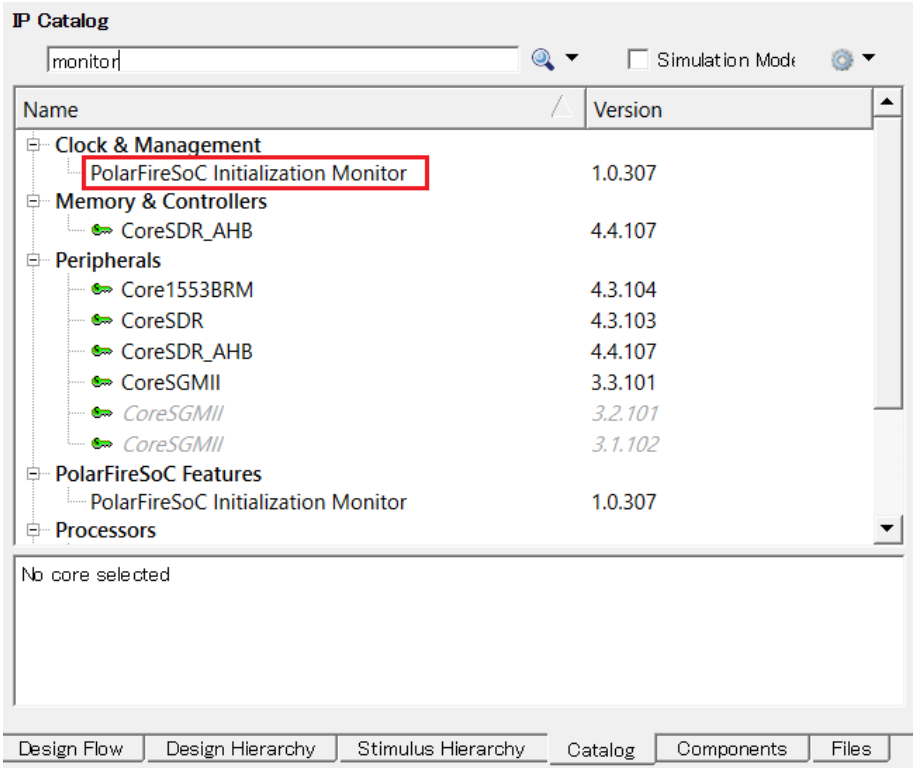
⑤ Warning ウィンドウが出るため OK を押します。

⑥ 同様に Information ウィンドウにて OK を押します。

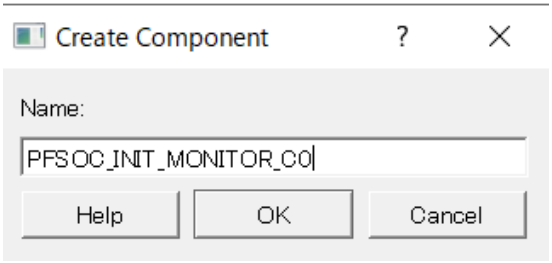
3-5. PolarFire Initialization Monitor

① PolarFireSoC Initialization Monitor を追加します。

IO キャリブレーションやバンクの電圧供給などをモニタリングできる IP です。



② Create Component ウィンドウにて OK を押します。



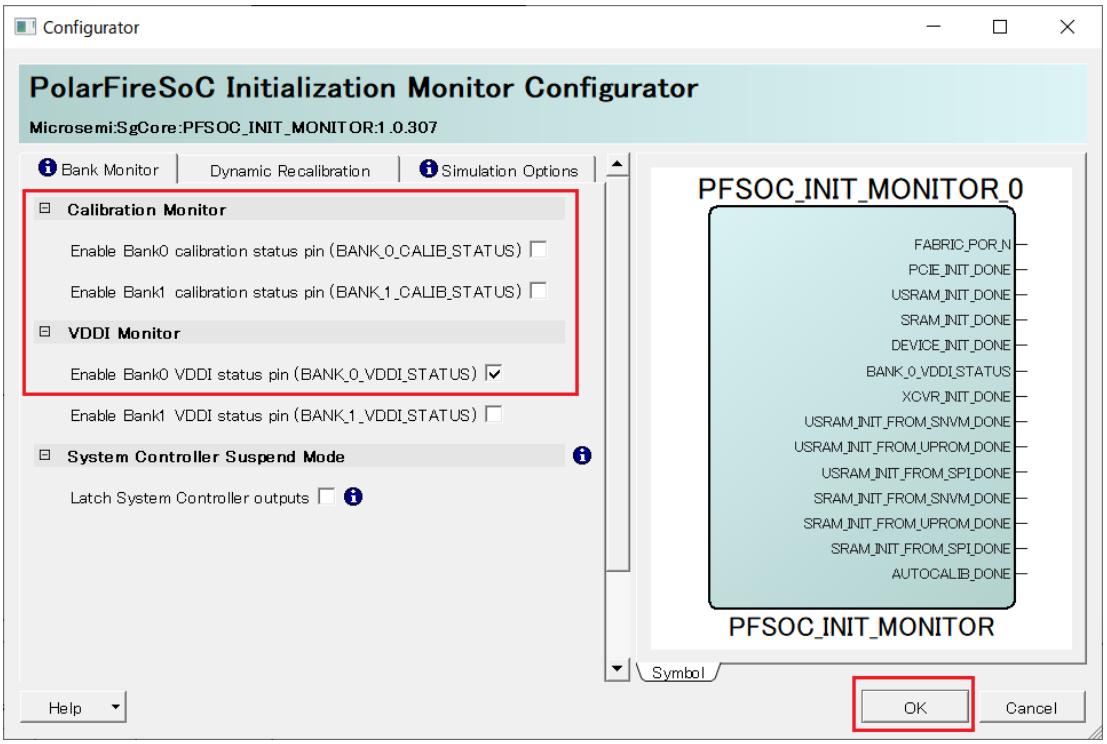
③ 下記のように設定し OK を押します。

Enable Bank0 calibration status pin (BANK_0_CALIB_STATUS) : チェックを外す

Enable Bank1 calibration status pin (BANK_1_CALIB_STATUS) : チェックを外す

Enable Bank0 VDDI status pin (BANK_0_VDDI_STATUS) : チェックを入れる

※ 今回は Bank 0 に繋がっている LED を使用します。



補足: DDR メモリを使用する場合

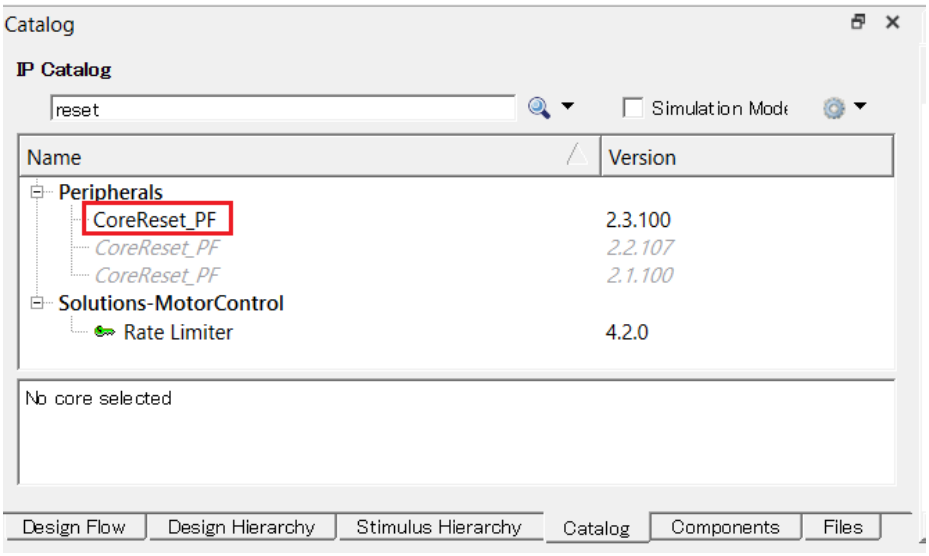
DDR メモリを使用する場合 BANK_#_CALIB_STATUS 信号を有効にします。

引用「Note:　IOs must be calibrated before initiating the training logic of the DDR controller. This requires generating a reset signal by ANDing the DEVICE_INIT_DONE and BANK_#_CALIB_STATUS signals of the PFSOC_INIT_MONITOR IP. BANK_# refers to the BANK where DDR subsystem is placed.」

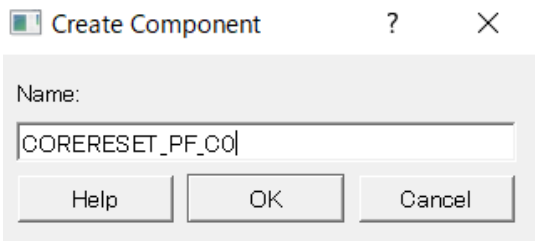
https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/UserGuides/microchip_polarfire_fpga_and_polarfire_soc_fpga_power_up_and_reset_user_guide_vb.pdf#page=14

3-6. CoreReset_PF

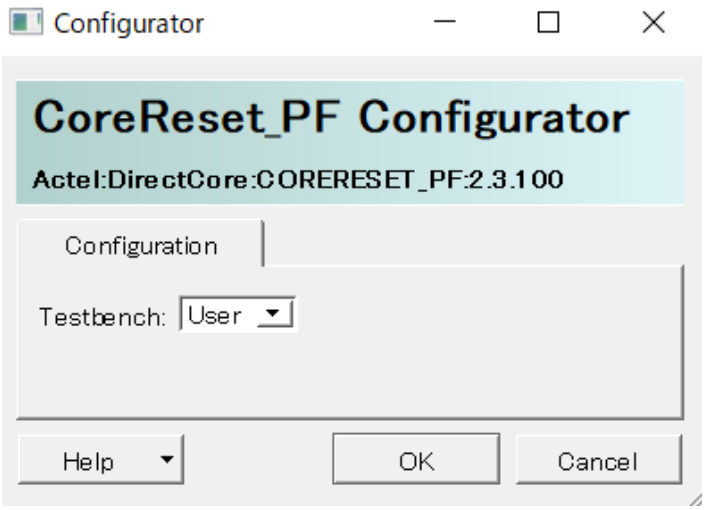
① CoreReset_PF を SmartDesign 上へドラッグ&ドロップします。



② Create Component ウィンドウにて OK を押します。



③ CoreReset_PF IP の設定画面にてデフォルトのまま OK を押します。



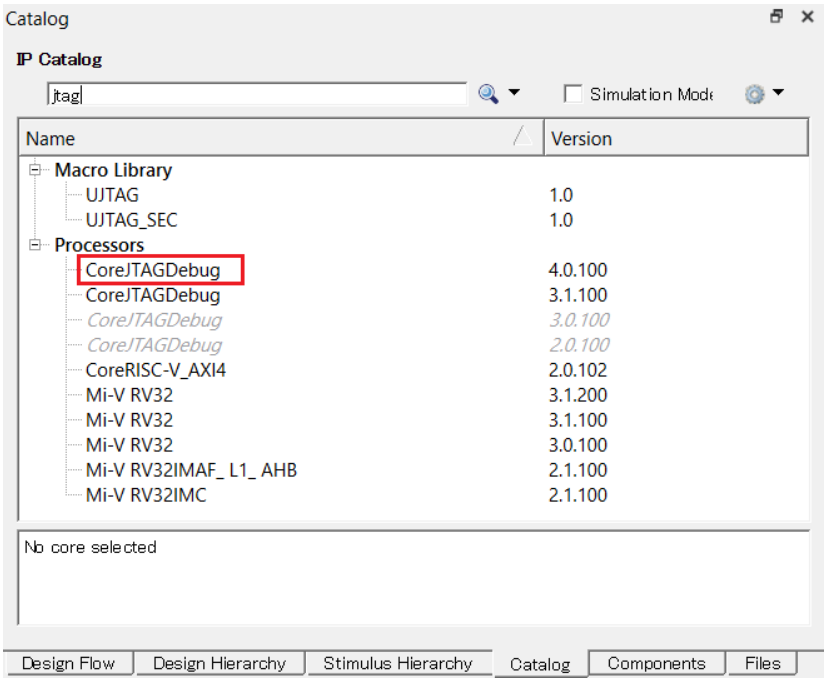
補足: PolarFire 以外のデバイスを使用する場合

CoreReset_PF は PolarFire 用の IP です。
他のデバイスを使用する場合は reset_synchronizer を HDL で用意し、
SmartDesign 上へドラッグ&ドロップします。
サンプルソースは MIV_RV32 User Guide をご参照ください。

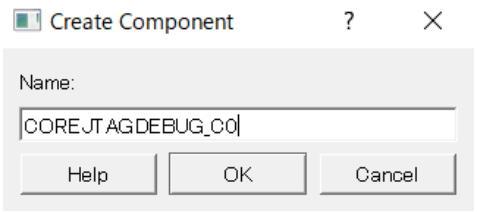
引用「Note: The difference between the PolarFire and the other design is the use of the PolarFire Initialization Monitor and CoreReset_PF. The HDL reset synchronizer acts as the CoreReset_PF, as this IP is only available on PolarFire. For other families, the reset synchronizer is used. The HDL code for the reset synchronizer is available in section 6.3.2 RTG4 (IG2/SF2) RESETN.」
https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/SupportingCollateral/MIV_RV32+v3.1+User+Guide.pdf

3-7. CoreJTAGDebug

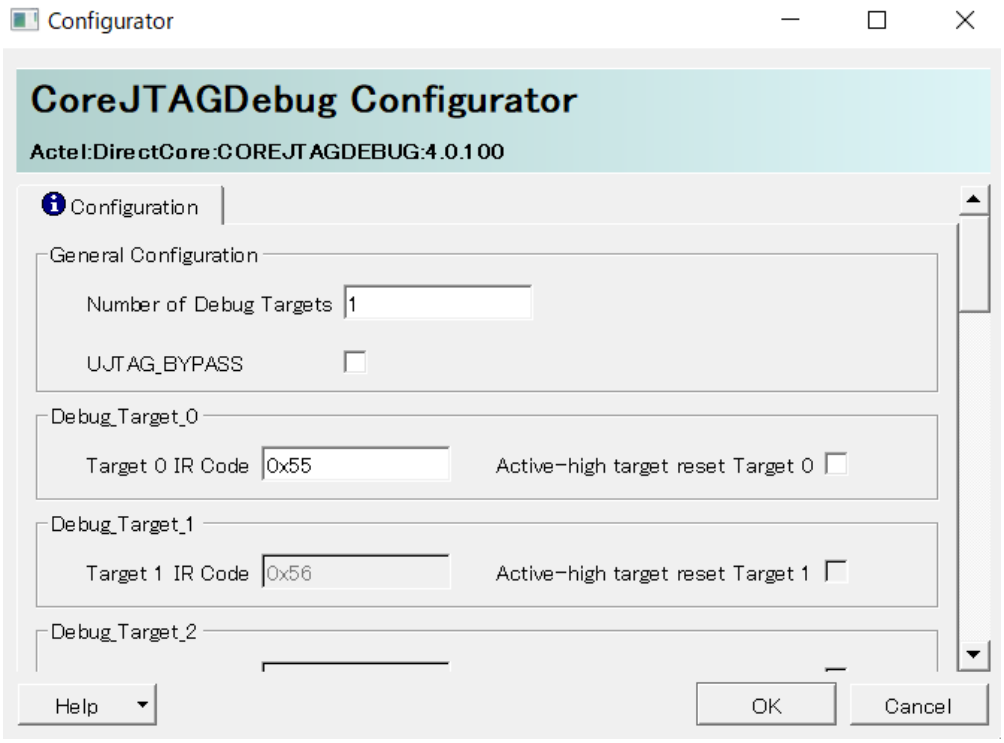
① CoreJTAGDebug を SmartDesign 上にドラッグ&ドロップします。



② Create Component ウインドウにて OK を押します。



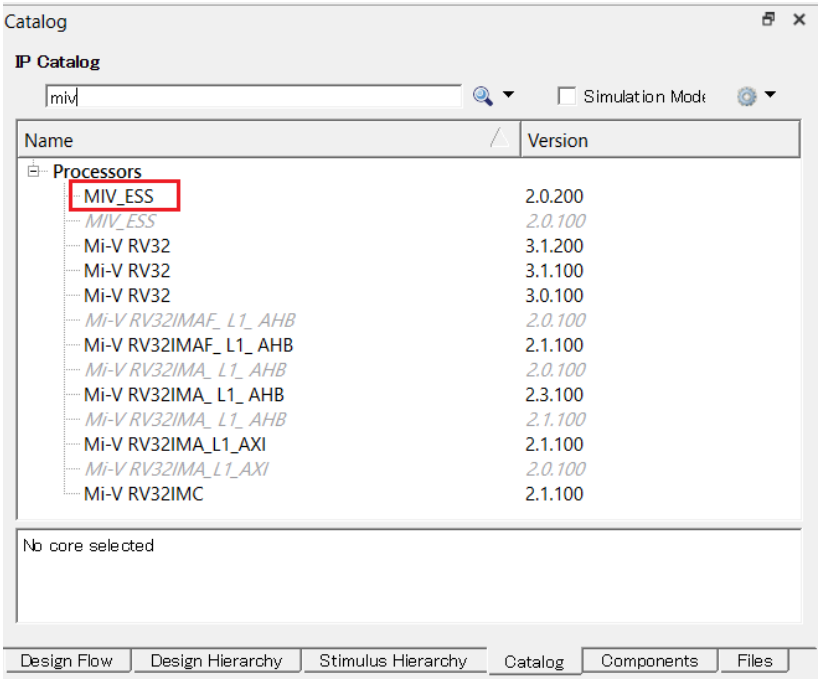
③ CoreJTAGDebug IP の設定画面にて、デフォルトのまま OK を押します。



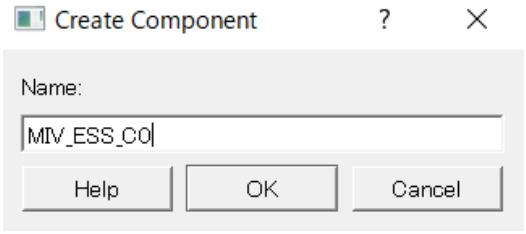
3-8. MIV_ESS

① MIV_ESS を SmartDesign 上にドラッグ&ドロップします。

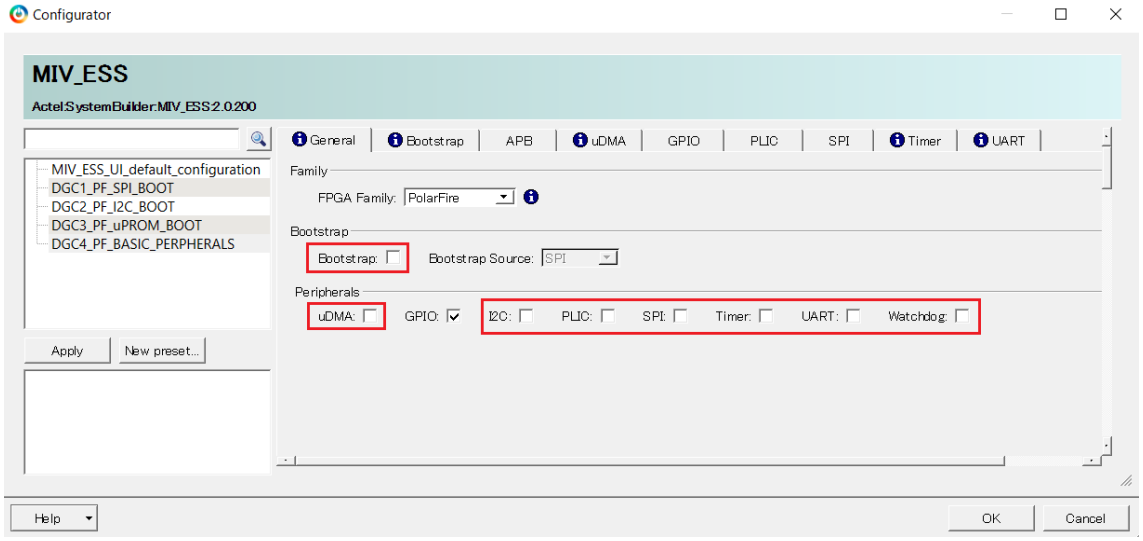
注意: Mi-V ではなく 横棒(-)なしの MIV です。



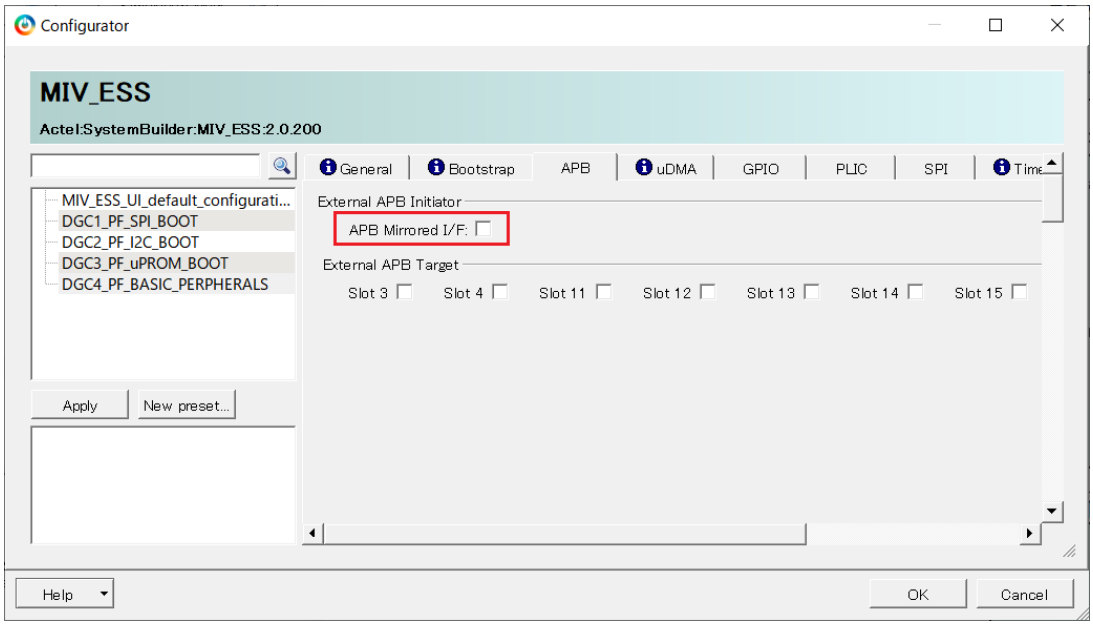
② Create Component ウィンドウにて OK を押します。



③ GPIO 以外のチェックを外します

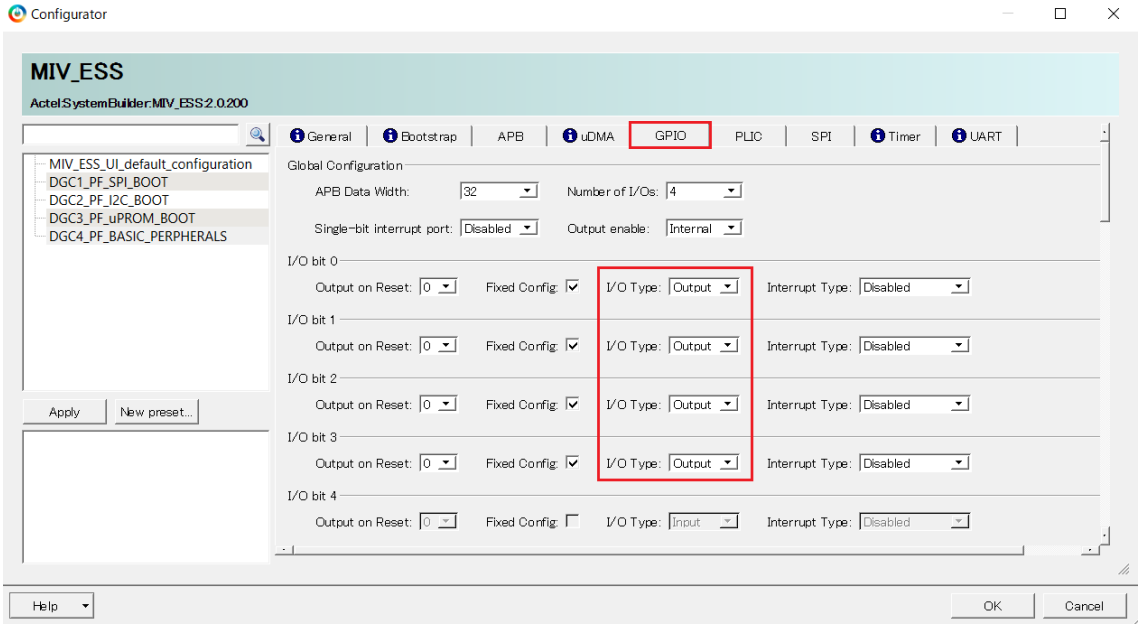


④ APB タブを開き、APB Mirrored I/F のチェックを外します。



⑤ GPIO タブを開きます。

LED 用として、I/O bit 0～3 の I/O Type を”Output”に設定し、OK を押します。



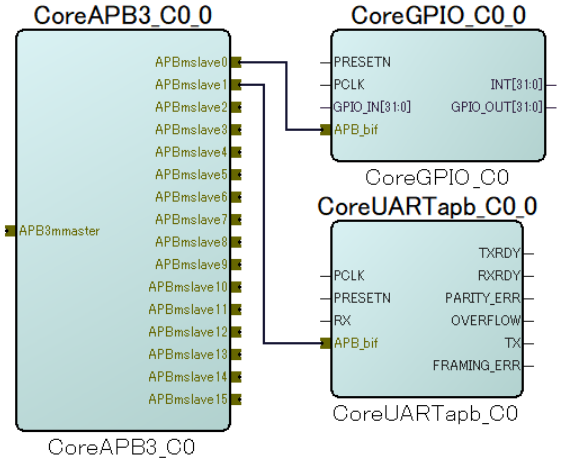
備考: MIV_ESS について

MIV_ESS は中に複数のペリフェラルが入っている IP になります。

MIV_ESS User Guide > Figure 2-1. MIV_ESS Block Diagram

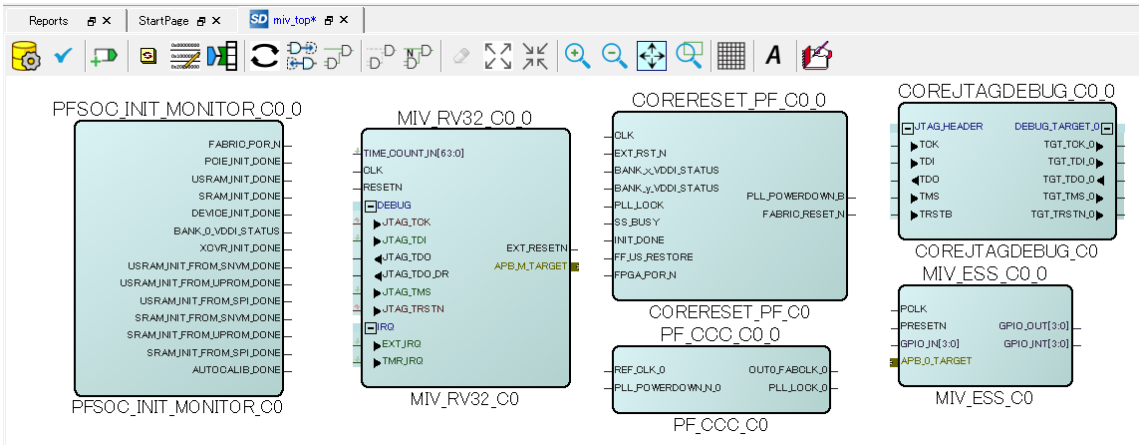
https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/UserGuides/ip_cores/directcores/MIVESS.pdf#page=9

今回は一例として MIV_ESS を使用しましたが、Core GPIO、CoreUARTapb、CoreAPB3、等使いたいペリフェラルに応じた IP を 1 個ずつ SmartDesign 上へドラッグ&ドロップ、接続しても等価回路を用意できます。



3-9. ブロックの接続

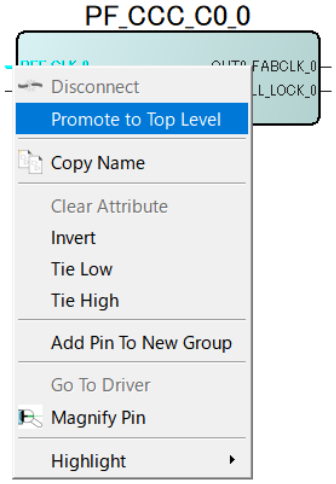
SmartDesign 上のブロックを接続します。



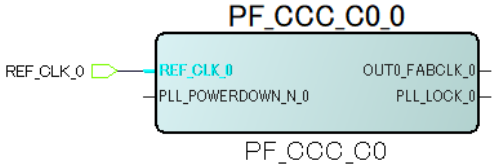
SmartDesign(回路図エディタ)について

【ポートを出す方法】

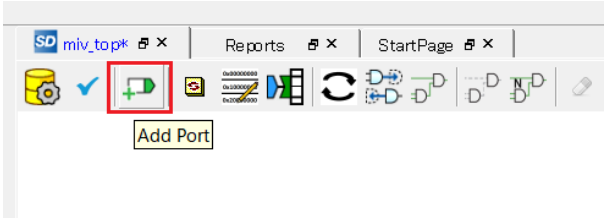
信号名を選び、右クリックするとコンテキストメニューが出ます。



Promote to Top Level を選択するとポートを出すことができます。

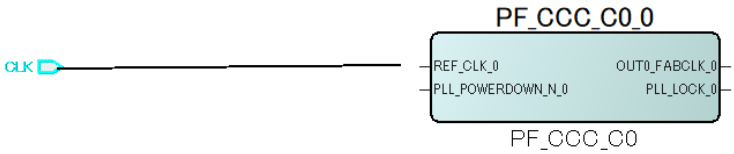


上部の Add Port からポートを追加してから、ブロックと接続することも可能です。



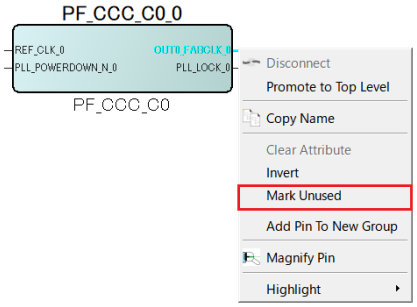
【接続方法】

接続したいポートやピンを選択して左クリックした状態のまま、接続先へドラッグします。

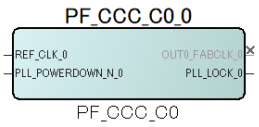


【ピンを未使用に設定する方法】

ピンを選択、右クリックし Mark Unused を選択します。

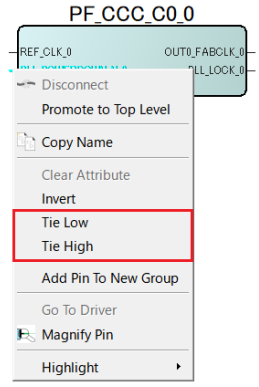


設定後 x マークが表示され、信号名がグレーアウトします。

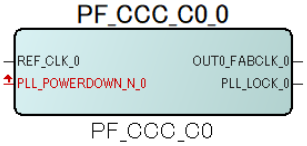


【ピンを High 固定、Low 固定にする方法】

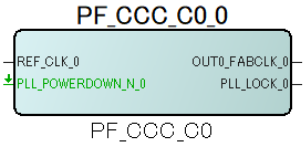
ピンを選択、右クリックし Tie Low や Tie High を選択します。



Tie High

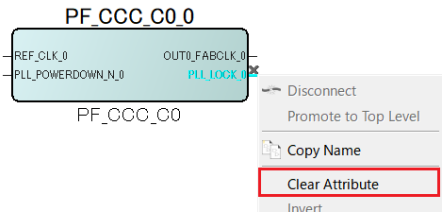


Tie Low



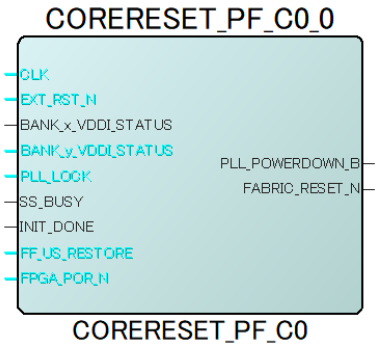
【Tie Low や Tie High、Mask Unused をクリアする方法】

ピンを選択、右クリックし Clear Attribute を選択します。



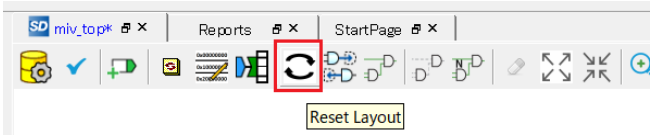
【一括設定】

Ctrl キーや Shift キーの活用でピンの複数選択が可能です。



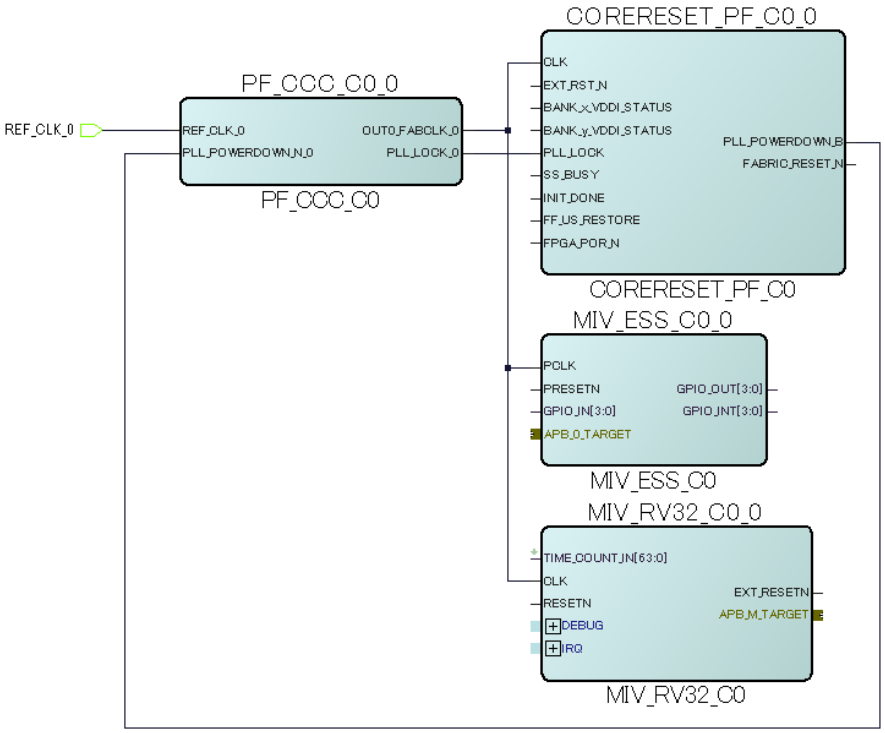
【回路図の整形】

視認性が悪くなった場合 SmartDesign 上部の Reset Layout ボタンをクリックすると回路図表示を整えることができます。



① PF_CCC_C0_0 ブロック

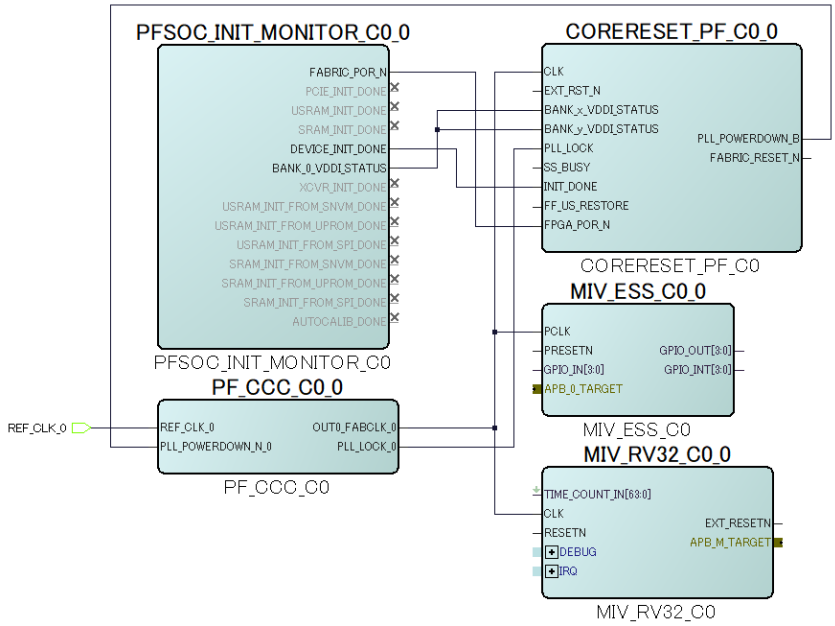
PF_CCC_C0_0	接続先
REF_CLK_0	ポートを出す(Promote to Top Level)
PLL_POWERDOWN_N_0	CORERESET_PF_C0_0 : PLL_POWERDOWN_B
PLL_LOCK_0	CORERESET_PF_C0_0 : PLL_LOCK
OUT0_FABCLK_0	MIV_RV32_C0_0 : CLK CORERESET_PF_C0_0 : CLK MIV_ESS_C0_0 : PCLK



② PFSOC_INIT_MONITOR_ C0_0 ブロック

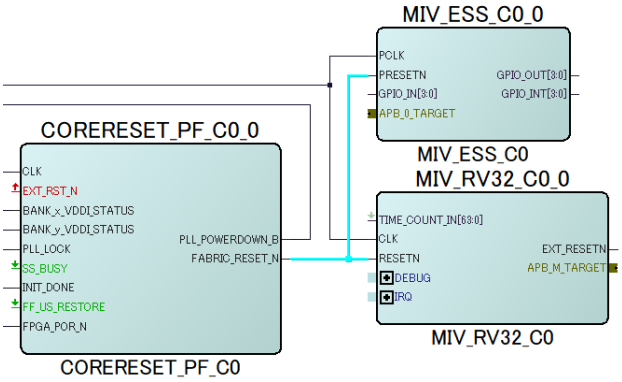
PFSOC_INIT_MONITOR_ C0_0	接続先
FABRIC_POR_N	CORERESET_PF_C0_0 : FPGA_POR_N
PCIE_INIT_DONE	Mask Unused
USRAM_INIT_DONE	Mask Unused
SRAM_INIT_DONE	Mask Unused
DEVICE_INIT_DONE	CORERESET_PF_C0_0 : INIT_DONE
BANK_0_VDDI_STATUS	CORERESET_PF_C0_0 : BANK_x_VDDI_STATUS CORERESET_PF_C0_0 : BANK_y_VDDI_STATUS
XCVR_INIT_DONE	Mask Unused

USRAM_INIT_FROM_SNVM_DONE	Mask Unused
USRAM_INIT_FROM_UPROM_DONE	Mask Unused
USRAM_INIT_FROM_SPI_DONE	Mask Unused
SRAM_INIT_FROM_SNVM_DONE	Mask Unused
SRAM_INIT_FROM_UPROM_DONE	Mask Unused
SRAM_INIT_FROM_SPI_DONE	Mask Unused
AUTOCALIB_DONE	Mask Unused

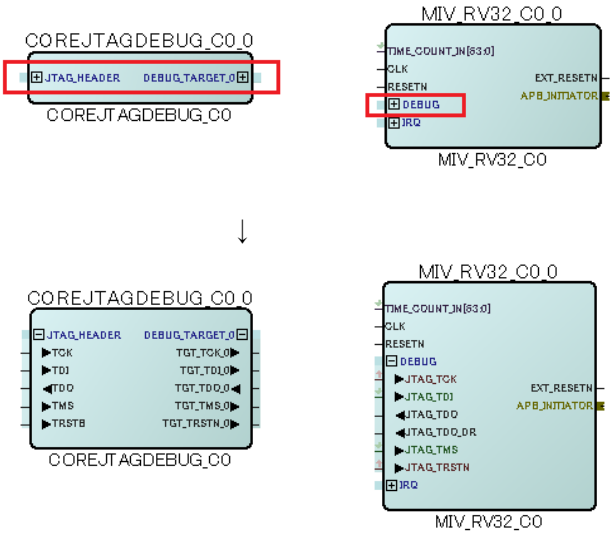


③ CORERESET_PF _C0_0 ブロック

PF_CCC_C0_0	接続先
EXT_RST_N	Tie High
SS_BUSY	Tie Low
FF_US_RESTORE	Tie Low
FABRIC_RESET_N	MIV_RV32_C0_0 : RESETN MIV_ESS_C0_0 : PRESETN

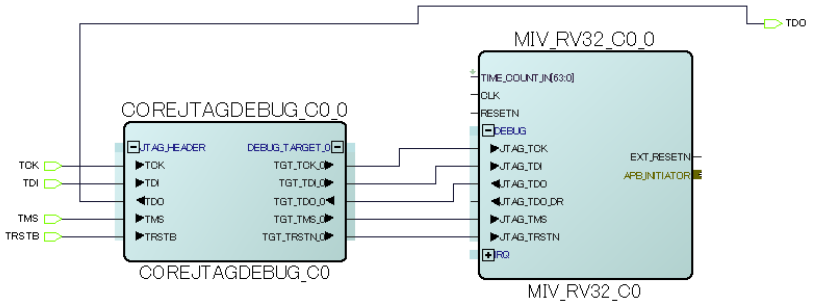


④ COREJTAGDEBUG_C0_0 の JTAG_HEADER、DEBUG_TARGET_0、MIV_RV32_C0_0 の DEBUG の+ボタンをクリックし展開します。



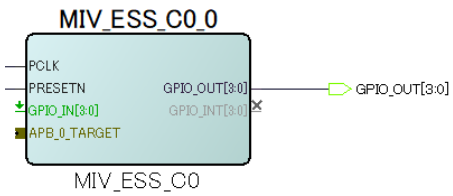
⑤ COREJTAGDEBUG_C0_0 ブロック

COREJTAGDEBUG_C0_0	接続先
TCK	ポートを出す(Promote to Top Level)
TDI	ポートを出す(Promote to Top Level)
TDO	ポートを出す(Promote to Top Level)
TMS	ポートを出す(Promote to Top Level)
TRSTB	ポートを出す(Promote to Top Level)
TGT_TCK_0	MIV_RV32_C0_0 : JTAG_TCK
TGT_TDI_0	MIV_RV32_C0_0 : JTAG_TDI
TGT_TDO_0	MIV_RV32_C0_0 : JTAG_TDO
TGT_TMS_0	MIV_RV32_C0_0 : JTAG_TMS
TGT_TRSTN_0	MIV_RV32_C0_0 : JTAG_TRSTN



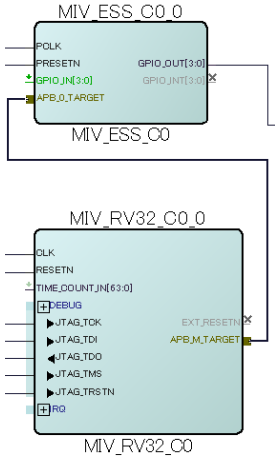
⑥ MIV_ESS_C0_0 ブロック

MIV_ESS_C0_0	接続先
GPIO_IN[3:0]	Tie Low
APB_0_mINITIATOR	MIV_RV32_C0_0 : APB_INITIATOR
GPIO_OUT[3:0]	ポートを出す(Promote to Top Level)
GPIO_INT[3:0]	Mark Unused

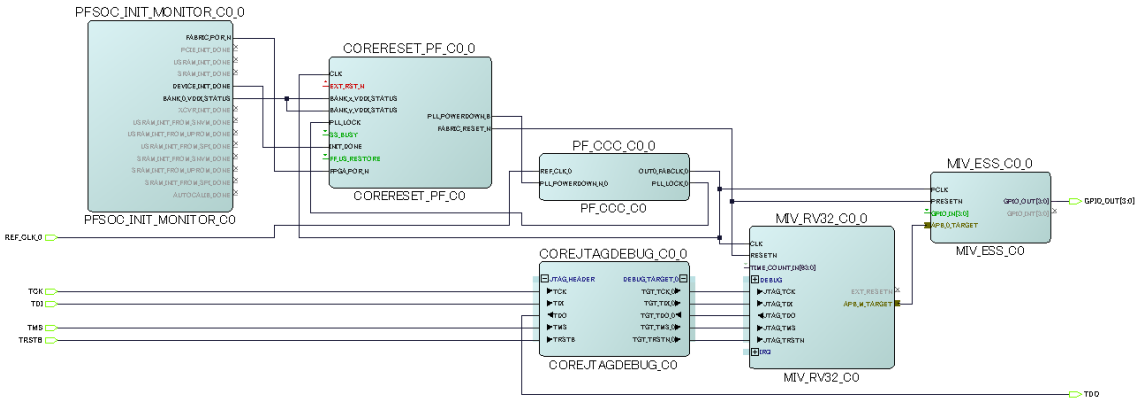


⑦ MIV_RV32_C0_0 ブロック

MIV_RV32_C0_0	接続先
EXT_RESETN	Mark Unused
APB_M_TARGET	MIV_ESS_C0_0 : APB_0_TARGET

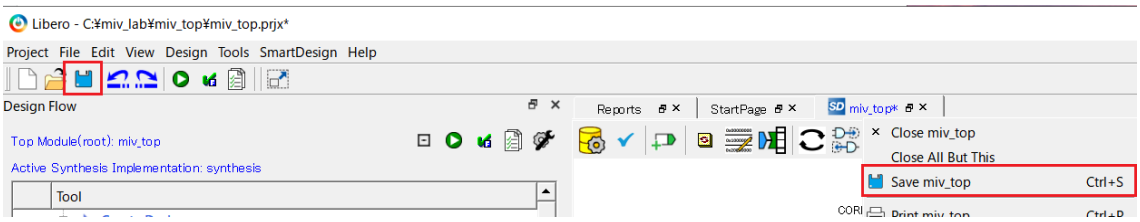


接続後：

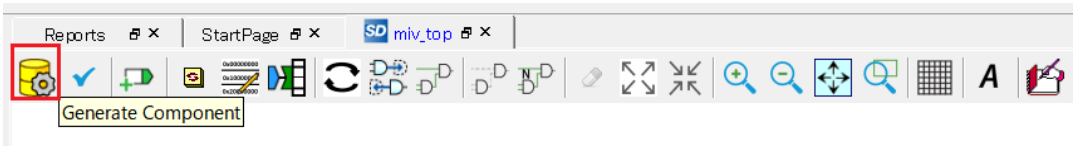


⑧ Save します。

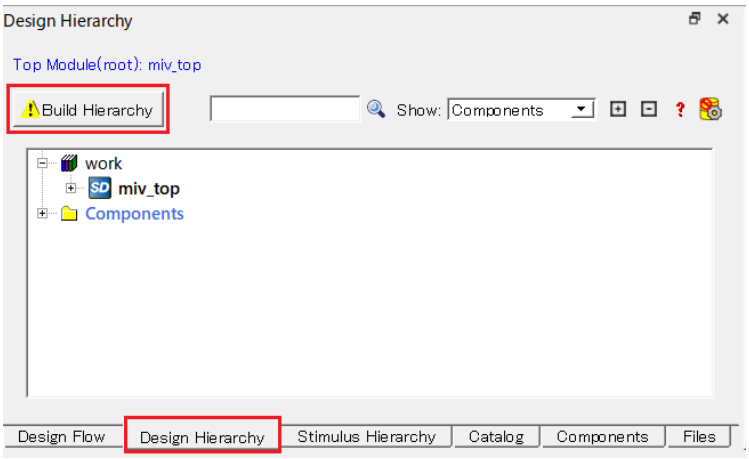
(Libero SoC プロジェクトを Save、もしくは回路図エディタタブを右クリックし Save)



⑨ Generate Component をクリックします。

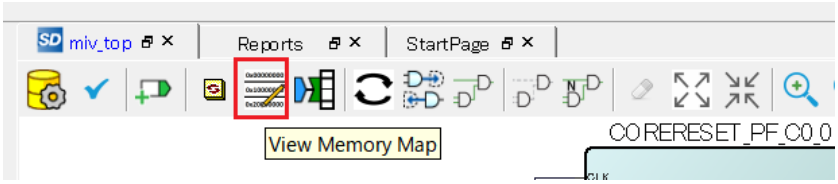


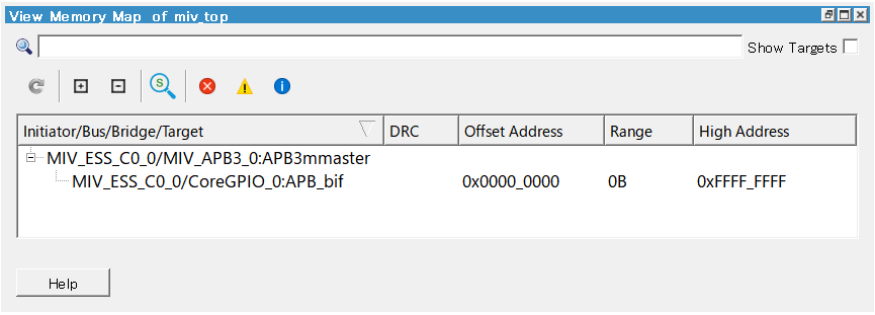
⑩ Design Hierarchy タブにて Build Hierarchy をクリックします。



備考: Address map

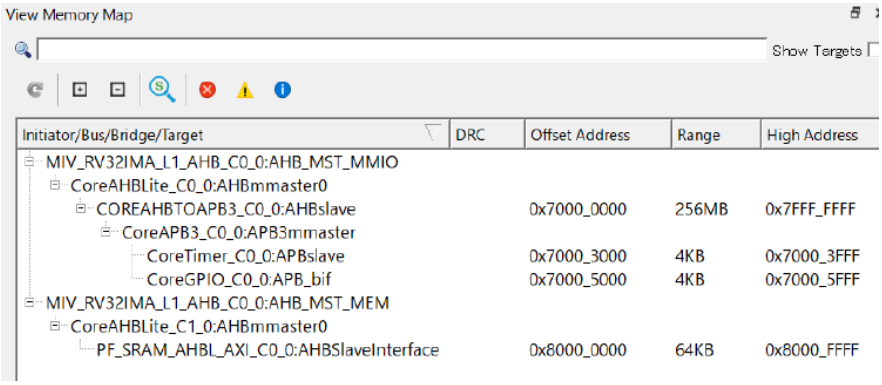
SmartDesign 上部の View Memory Map ボタンより、Address map を確認可能です。





表示は構成によって変わります。

一例:



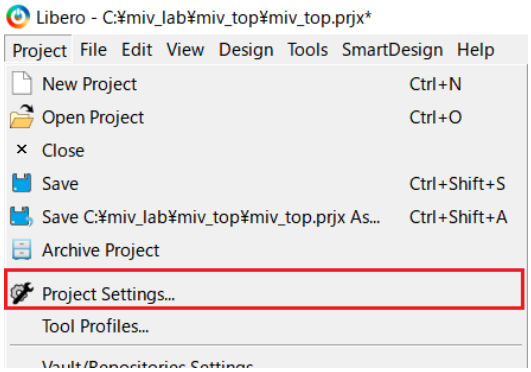
Address map のエラーが消えない場合 ：

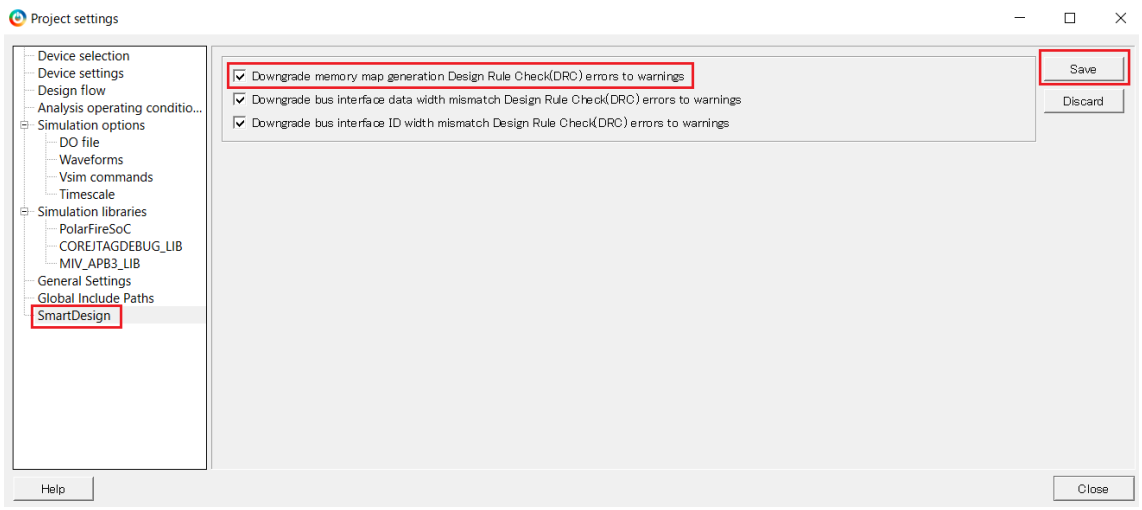
Libero SoC の Project Settings にて"Downgrade memory map generation DRC errors to warnings"へチェックを入れます。

現在 Libero SoC にて Issue があり、将来のバージョンにて修正される予定です。

引用「Important: In case any error is observed related to address space access issue, ensure to perform the following step: Navigate to Project > Project Settings > SmartDesign, and then Enable "Downgrade memory map generation DRC errors to warnings" as shown in the following figure. 」

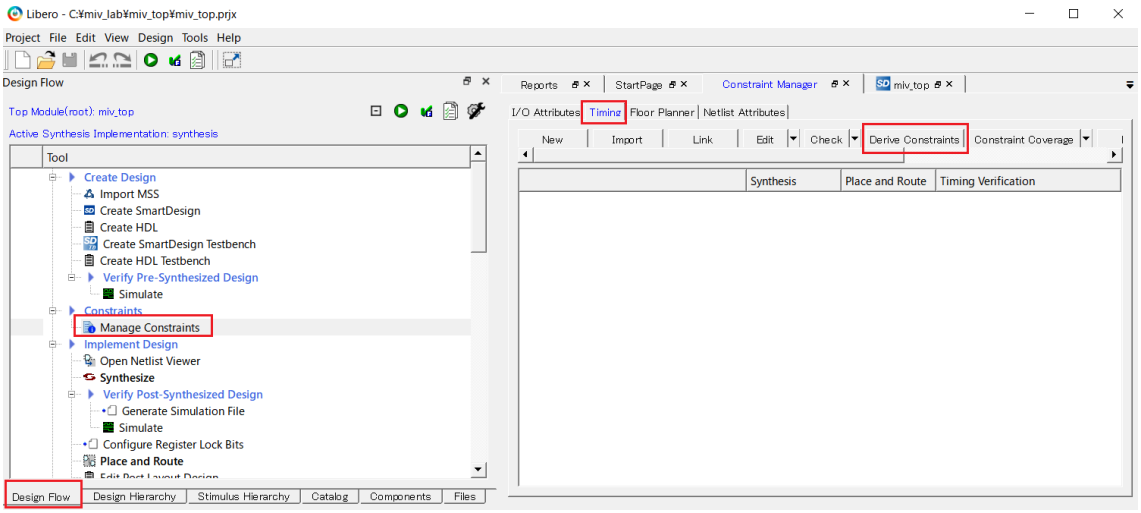
https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ApplicationNotes/ApplicationNotes/PolarFire_FPGA_Building_MIV_Subsystem_AN4997.pdf#page=21



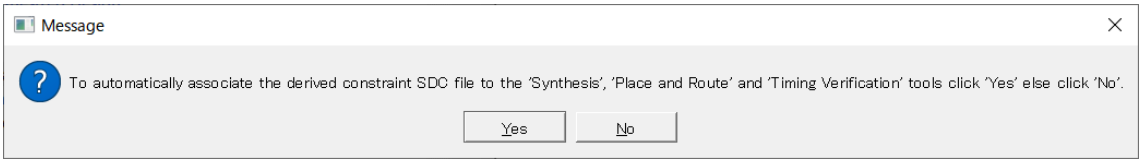


3-10. Drive Constraints

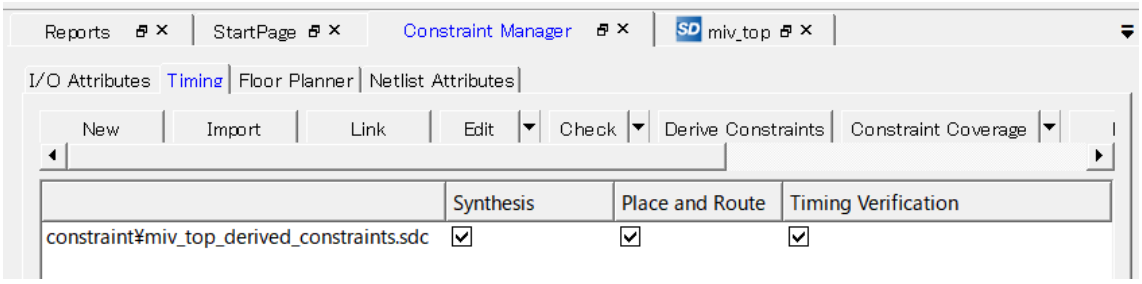
① Design Flow タブから Manage Constraints を開き、
Timing タブにて Drive Constraints を実施します。



② Message ウィンドウにて Yes をクリックします。

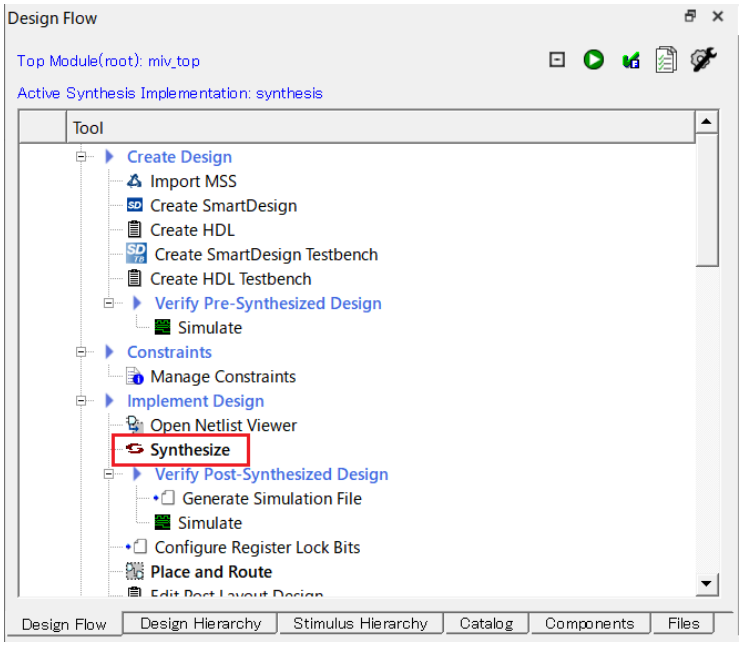


③ sdc ファイルが自動生成されたこと、Synthesis、Place and Route、
Timing Verification にチェックが入っていることを確認します。



3-11. 論理合成

Synthesize にて論理合成を実施します。



3-12. ピンアサイン

クロック、LED をピンアサインします。

ピン番号は、PolarFire SoC FPGA Discovery Kit User Guide の

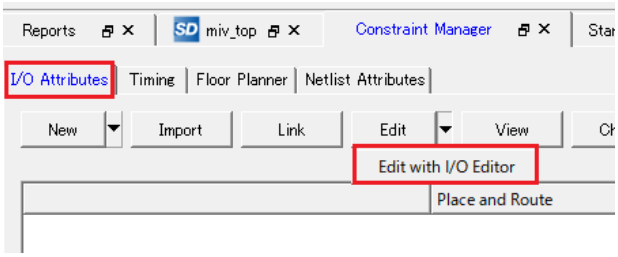
3.5 Debug Circuitry

3.9 50 MHz Oscillator (DSC1001DL5-050.0000)

から確認可能です。

https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/UserGuides/PolarFire_SoC_FPGA_Discovery_Kit_User_Guide.pdf

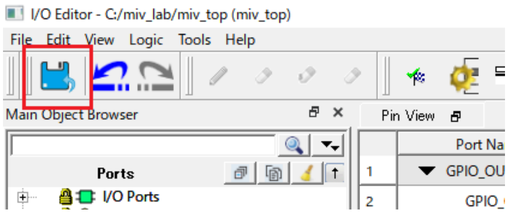
① Constraint Manager の I/O Attributes タブより I/O Editor を開きます。



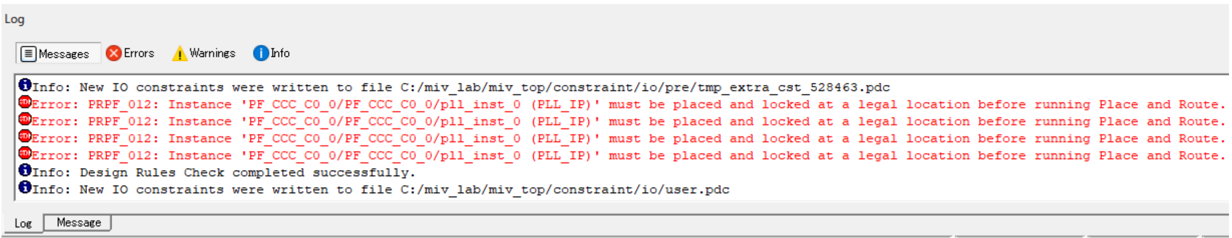
② 下記の通りアサインします。

信号名	ピン番号
GPIO_OUT[0]	T18
GPIO_OUT[1]	V17
GPIO_OUT[2]	U20
GPIO_OUT[3]	U21
REF_CLK_0	R18

③ Save して、I/O Editor を閉じます。

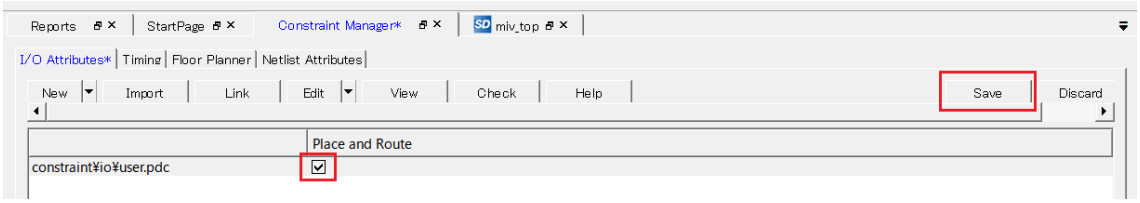


Log ウィンドウにて
Error: PRPF_012: Instance 'PF_CCC_C0_0/PF_CCC_C0_0/pll_inst_0 (PLL_IP)' must be placed and locked at a legal location before running Place and Route.
が表示された場合無視します。(後ほど配置配線にて自動で PLL をアサインします。)

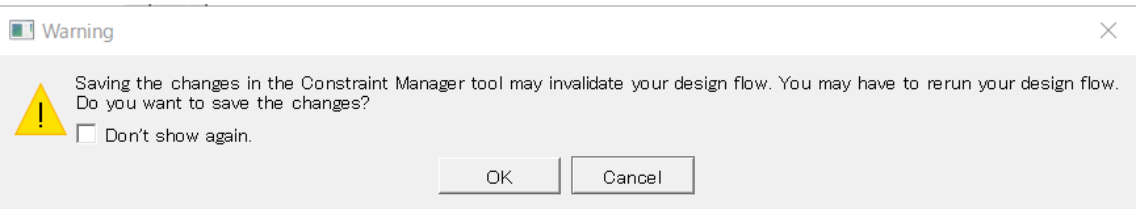


④（もしチェックが入っていない場合）

Place and Route へチェック入れて Save します。

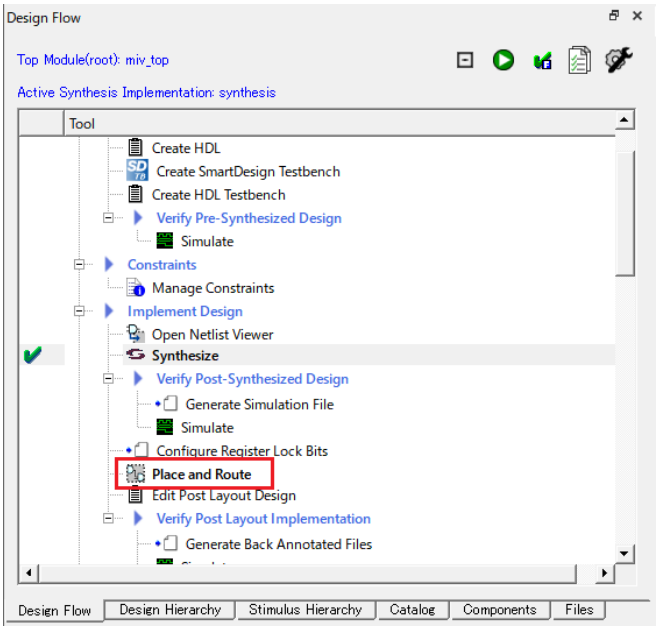


Warning ウィンドウが出た場合は OK をクリックします。



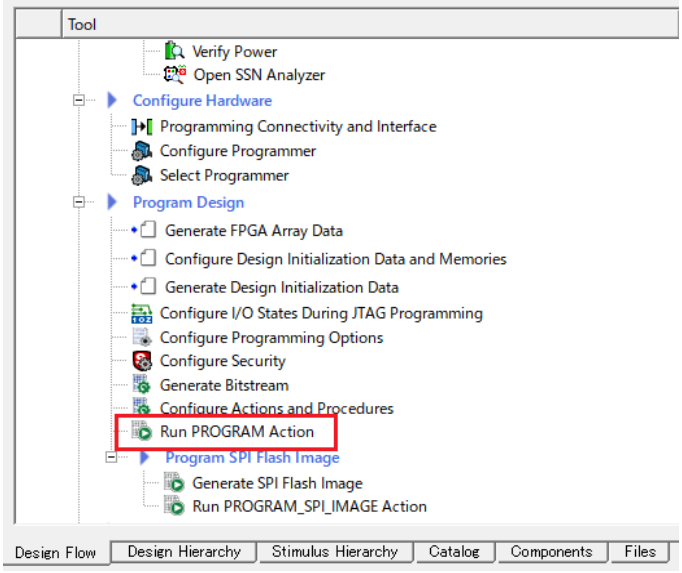
3-13. 配置配線

Place and Route をクリックし、配置配線します。



3-14. 書き込み

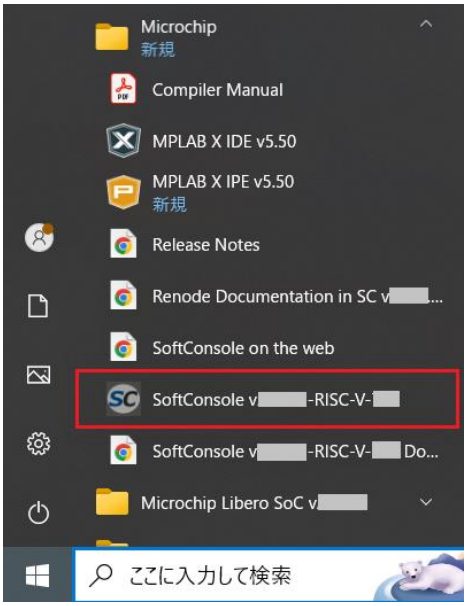
Discovery Kit を PC へ接続し、Run PROGRAM Action をダブルクリック、
作成したデザインを書き込みます。



4. ソフトウェア

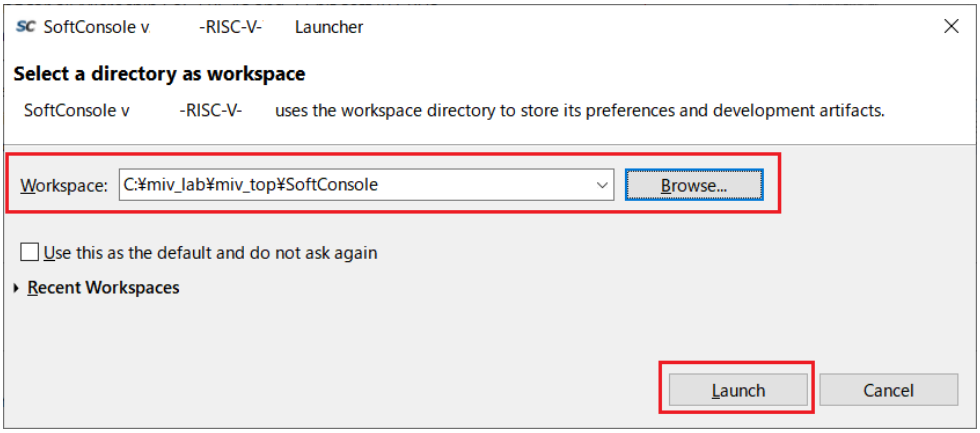
4-1. SoftConsole 起動

SoftConsole を起動します。

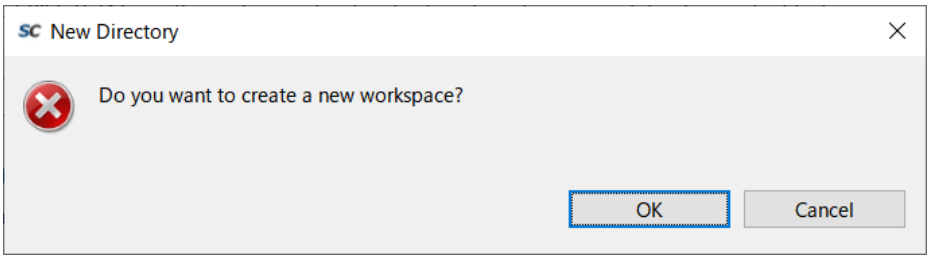


4-2. workspace の作成

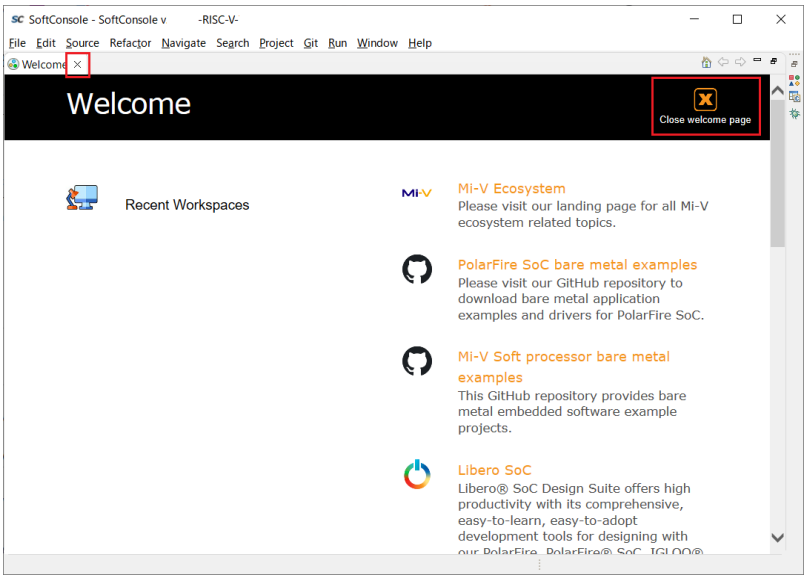
① SoftConsole のプロジェクトを管理する Workspace として任意のフォルダを指定、Launch をクリックします。



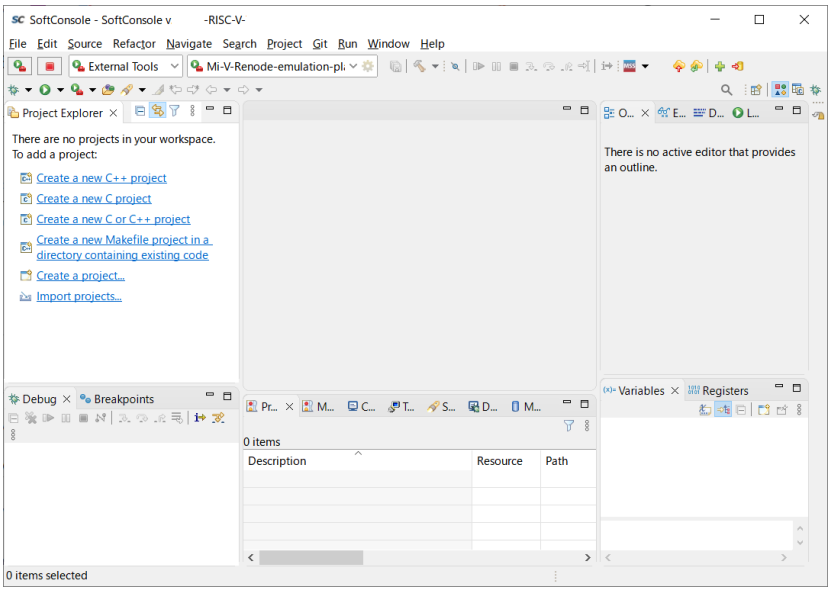
② New Directory ウィンドウにて、新しく workspace を作成するかどうか聞かれるため OK をクリックします。



③ Welcome ページを閉じます。

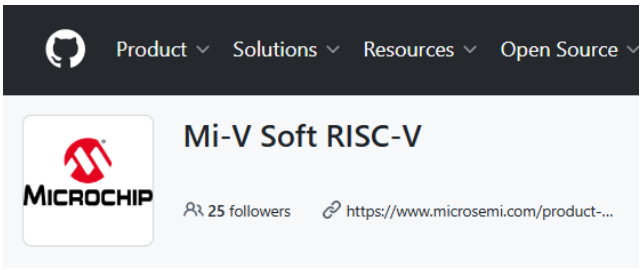


Welcome ページを閉じた後:

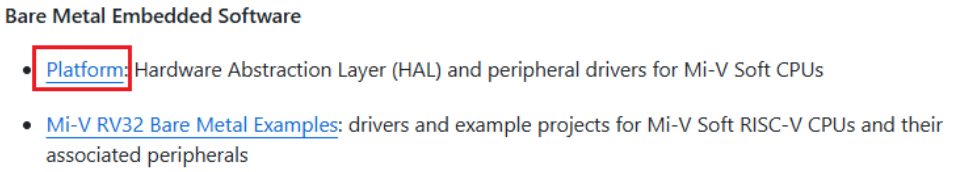


4-2. Driver の入手

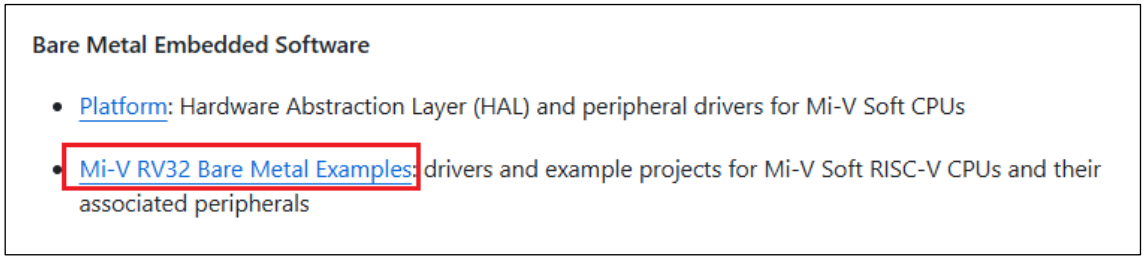
① GitHub にて ”Mi V Soft RISC V” ページを開きます。
https://github.com/Mi-V-Soft-RISC-V



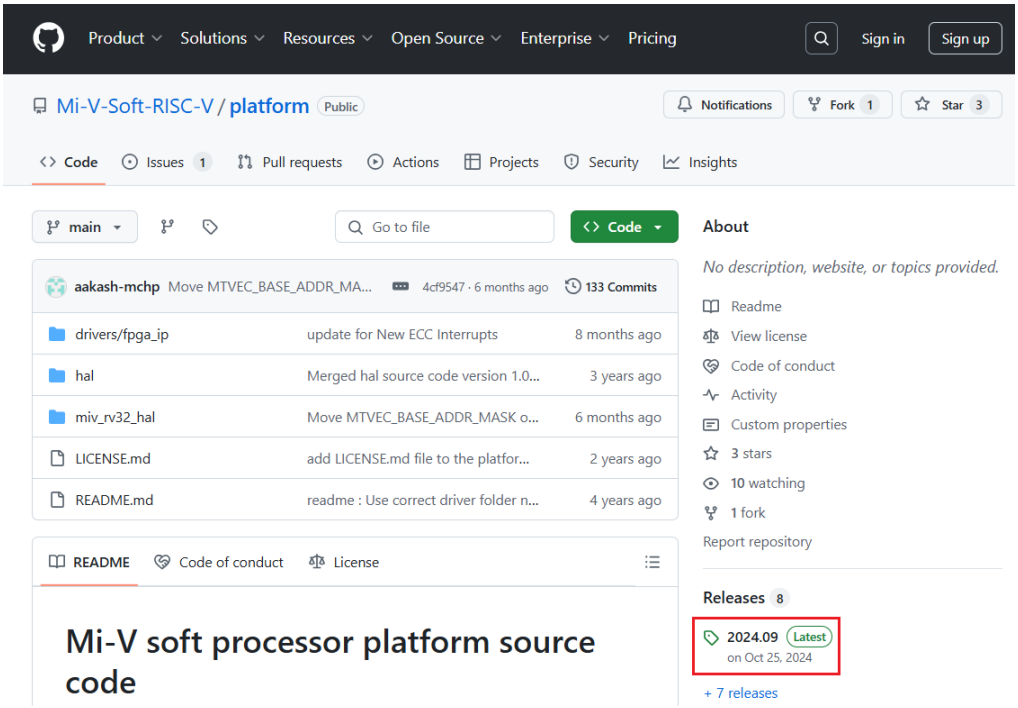
② ページを下にスクロールします。
Mi V CPU やペリフェラルの Driver を入手するため Platform のリンクを開きます。

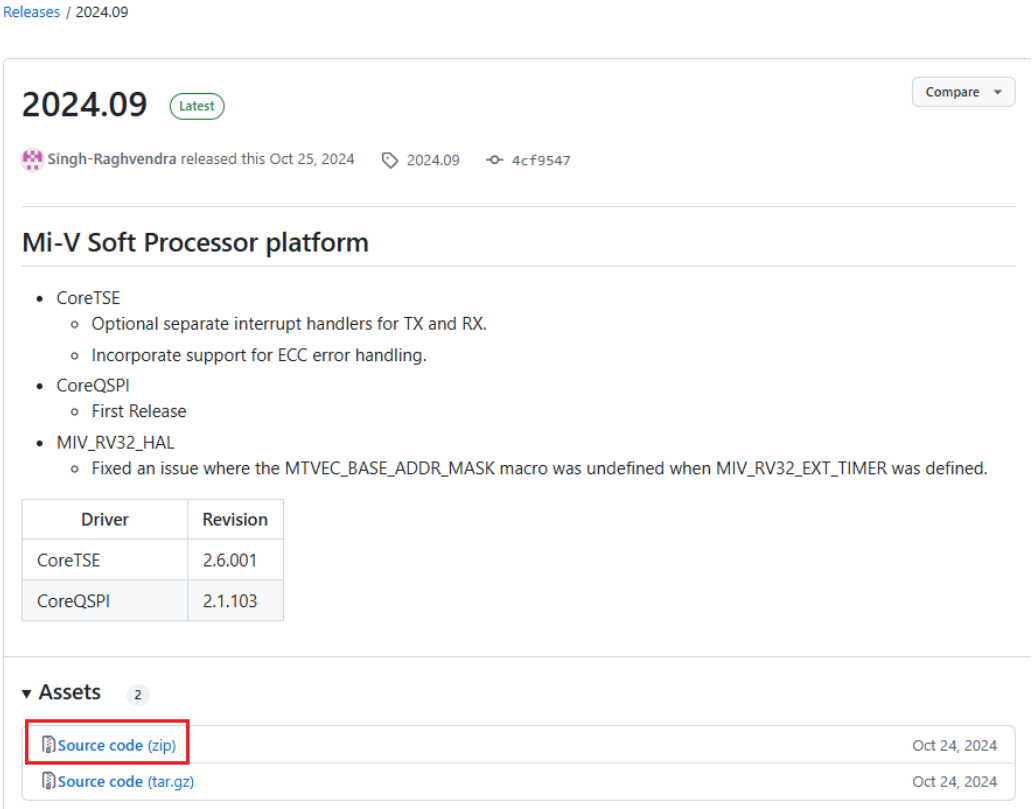


備考: Mi-V RV32 Bare Metal Examples
今回は 0 から C ソースを作成しますが、Mi V RV32 Bare Metal Examples を
ダウンロードするとペリフェラルに応じたサンプルデザインが含まれています。
必要に応じて適宜ご参考ください。

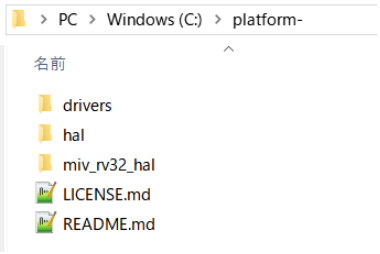


③ 最新版をダウンロードします。



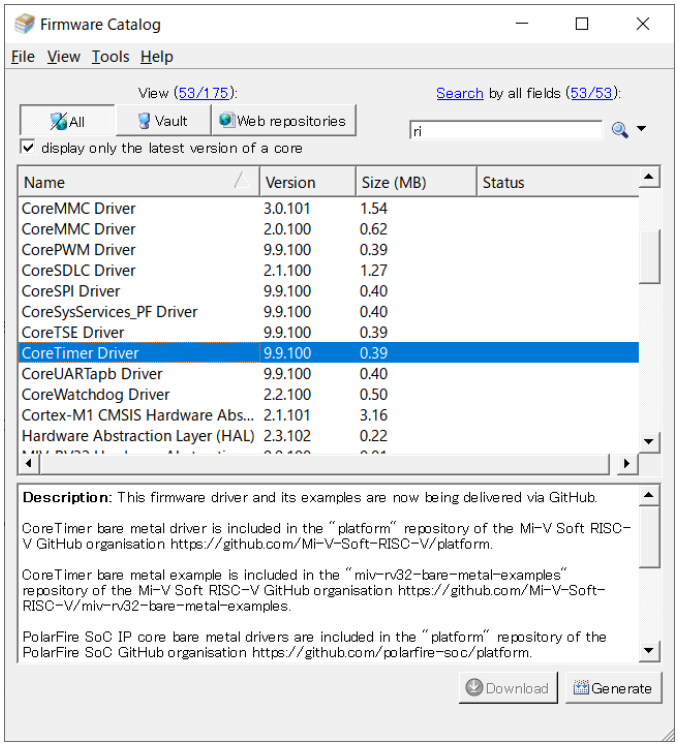


④ ダウンロードした zip ファイルを任意の場所に解凍します。



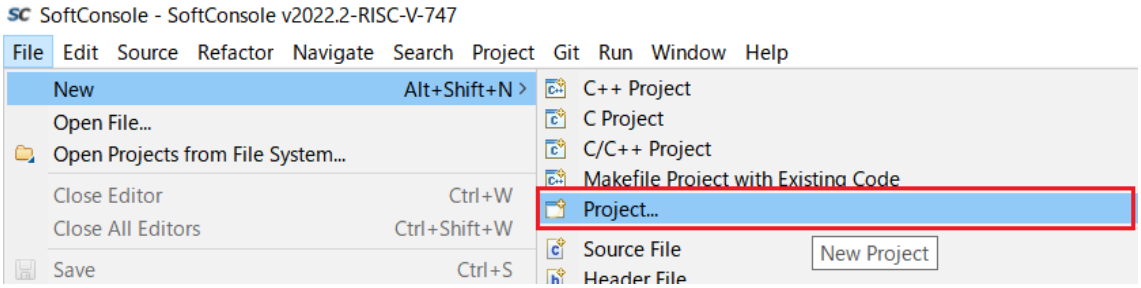
備考: Firmware Catalog について

以前は Firmware Catalog から Driver を入手していましたが
現在は GitHub から入手頂くよう変更になりました。

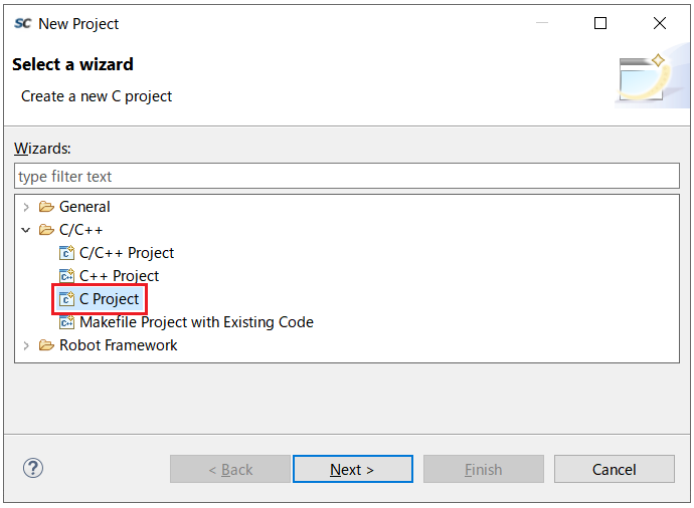


4-3. 新規プロジェクト作成

① File > New > Project... をクリックします。



② C/C++を展開し、 C Project を選択して Next をクリックします。

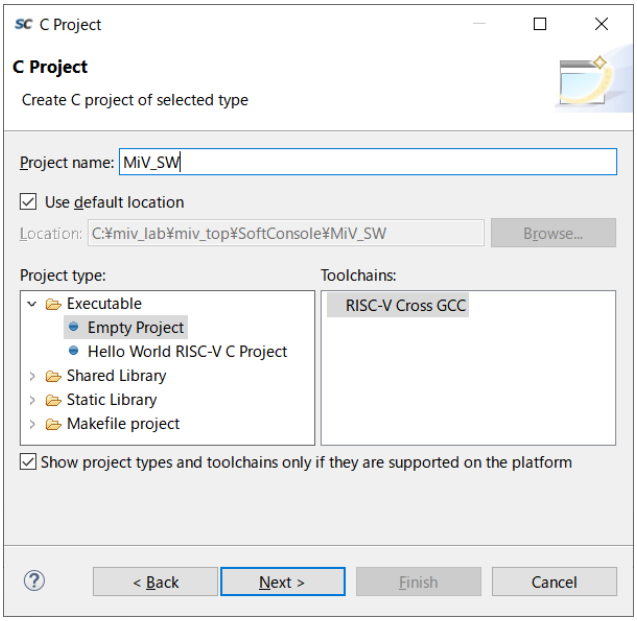


③ 下記の通り設定し Next をクリックします。

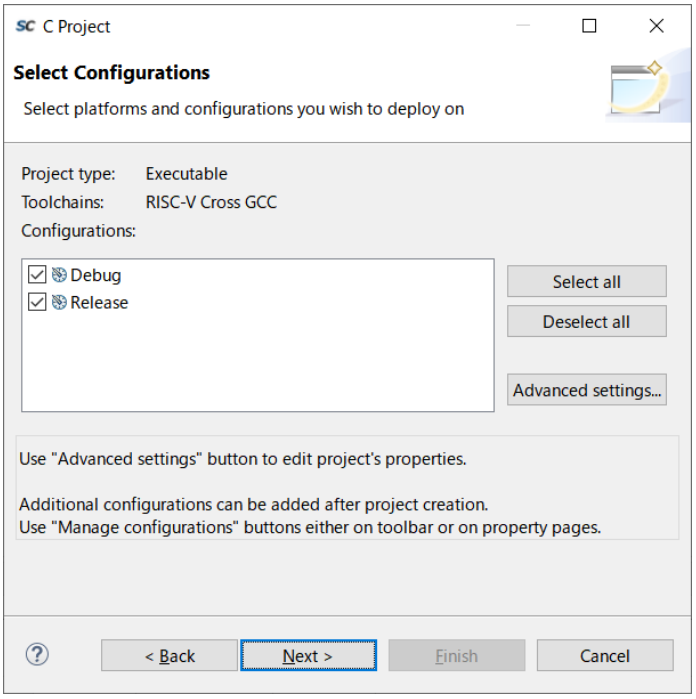
Project name : MiV_SW

Project type : Empty Project を選択

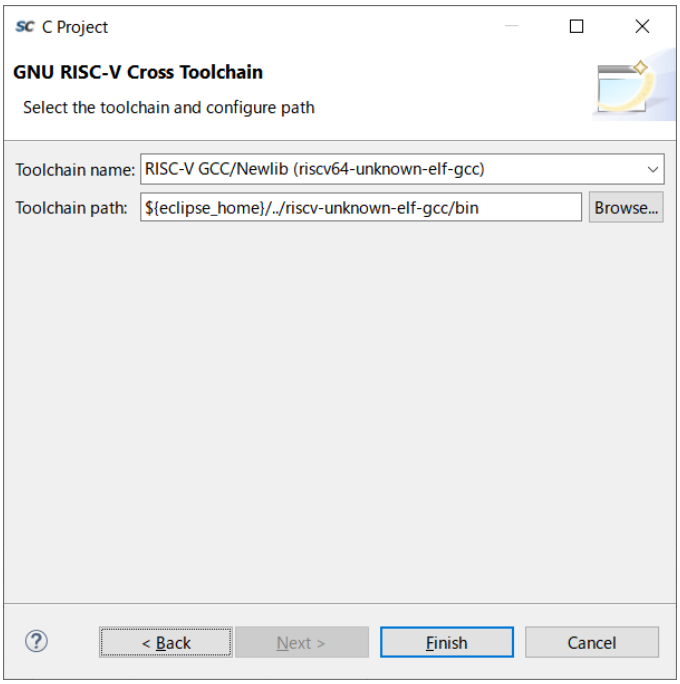
Toolchains : RISC V Cross GCC を選択



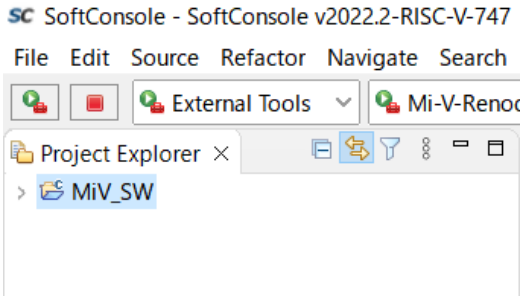
④ デフォルトのまま Next ボタンをクリックします。



⑤ デフォルトのまま Finish ボタンをクリックします。

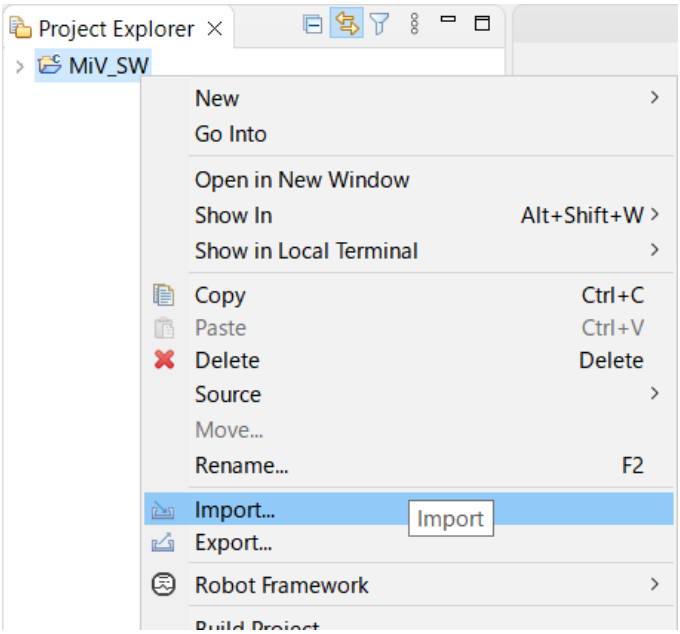


⑥ プロジェクトが作成されたことを確認します。

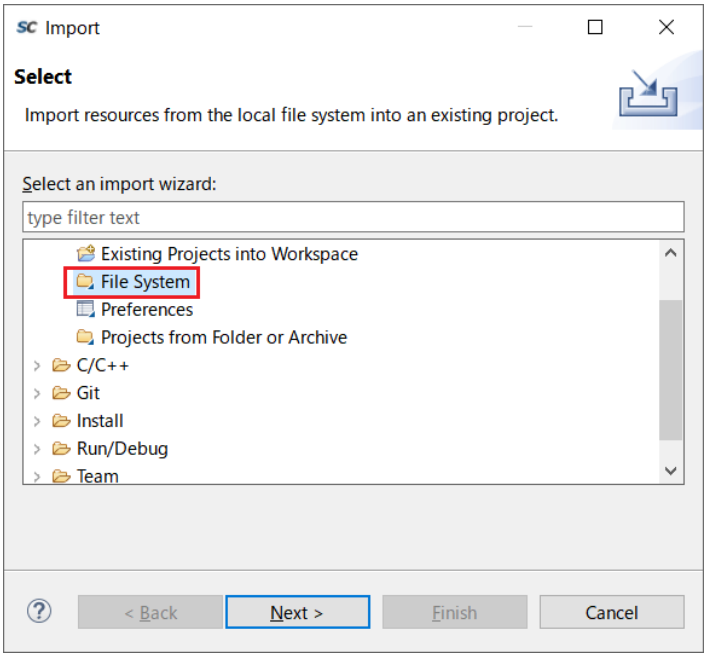


4-4. Driver のインポート

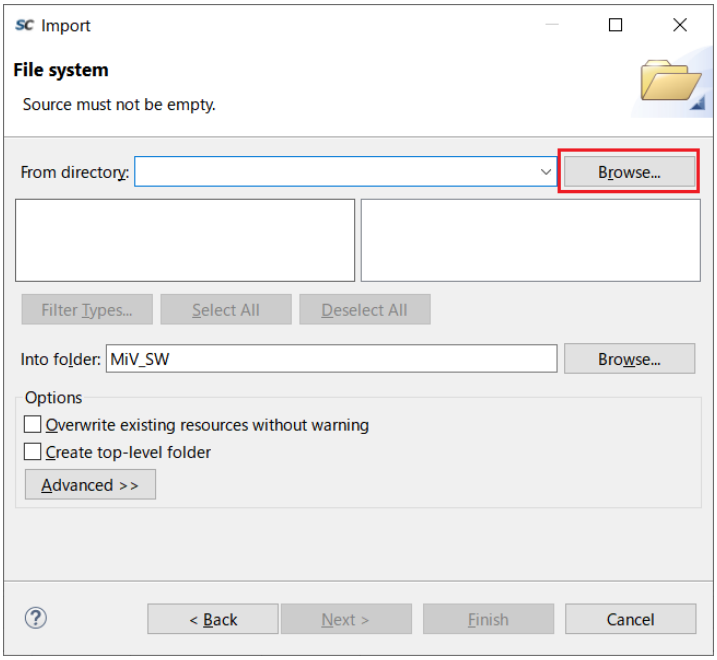
① SoftConsole にて MiV_SW を右クリックし Import... をクリックします。



② File System を選択し Next をクリックします。



③ Browse...ボタンから、GitHub からダウンロードし解凍した platform-xxxx フォルダを選択します。

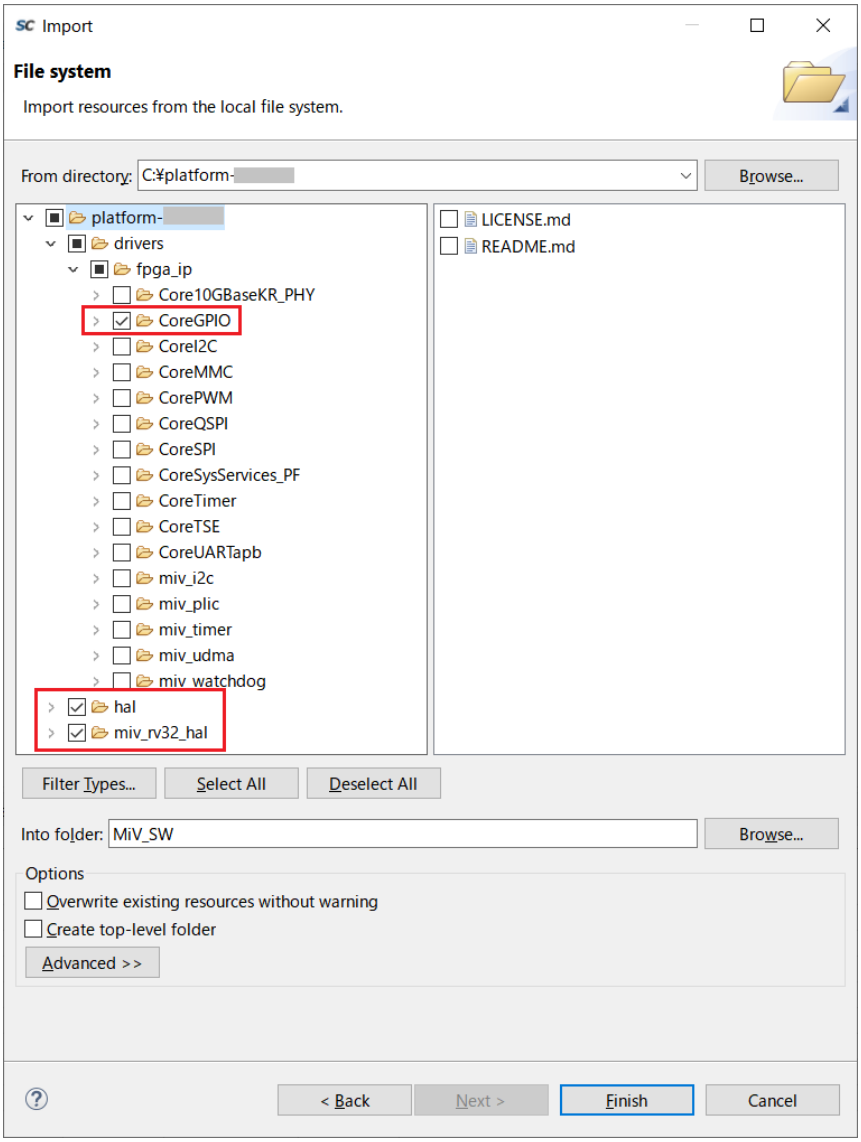


④ 下記へチェックを入れ、 Finish をクリックします。

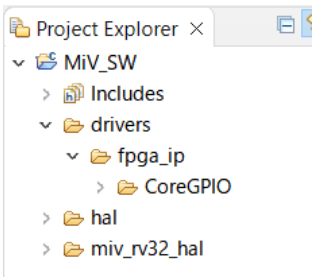
drivers/ fpga_ip/CoreGPIO

hal

miv_rv32_hal

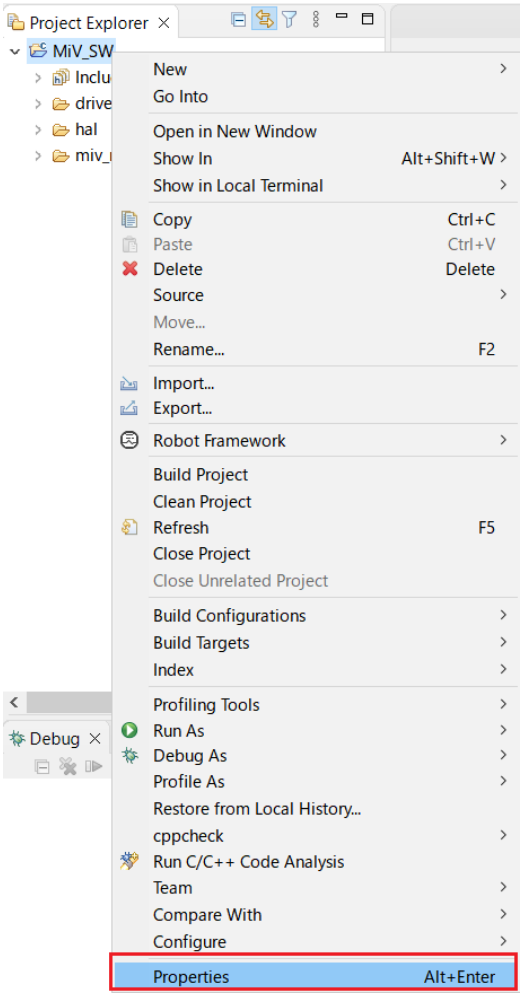


インポート後：

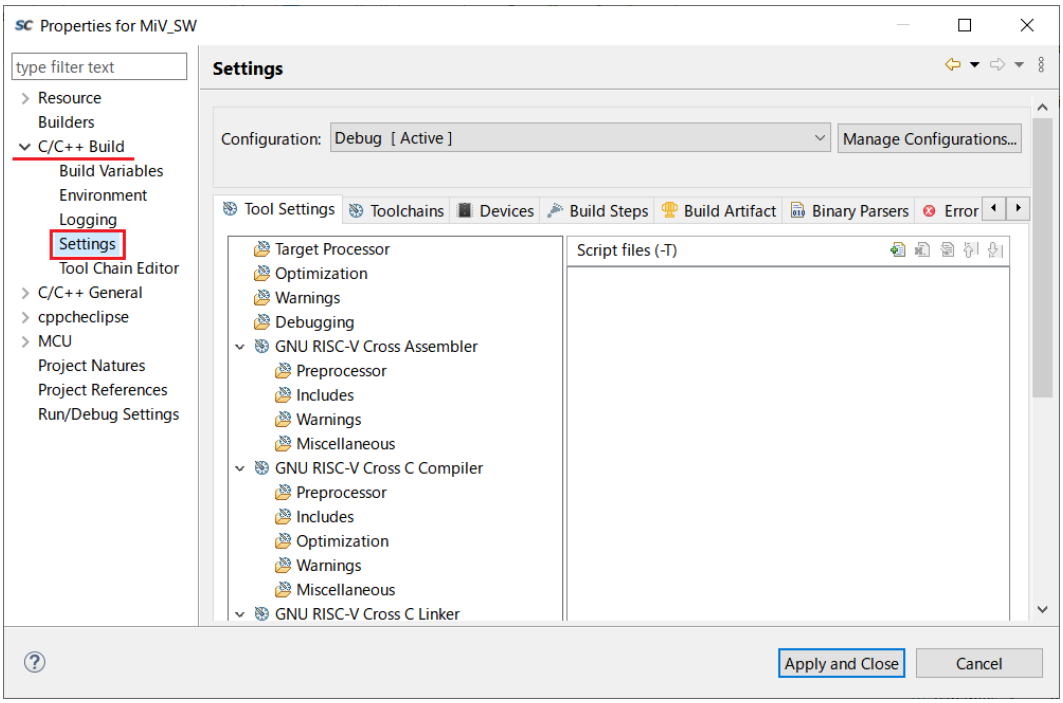


4-5. プロパティ設定

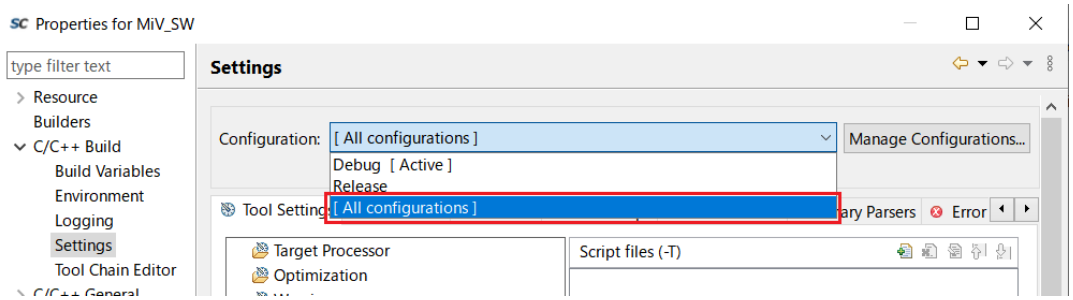
① プロジェクト名を右クリックし、Properties をクリックします。



② C/C++ Build > Settings を選択します。



③ Configuration にて［ All configurations ］を選択します。

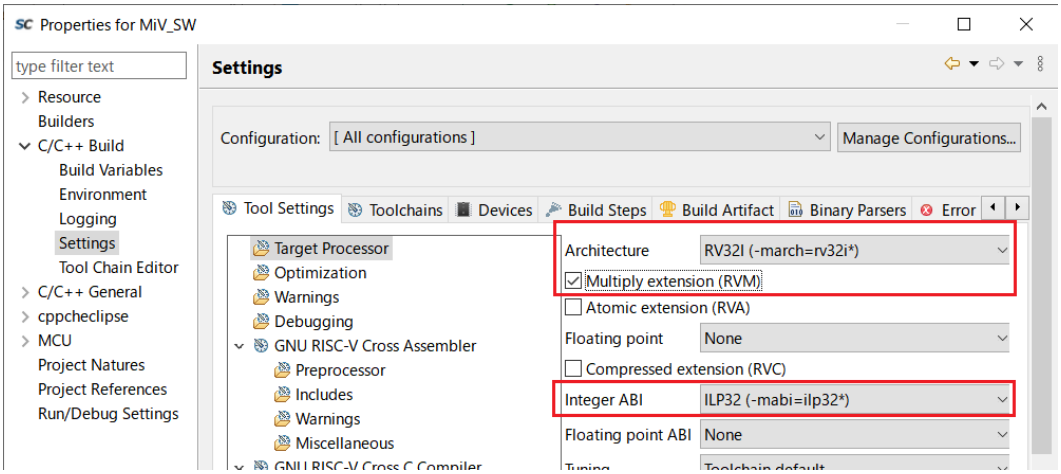


④ Target Processor にて下記の通り設定します。

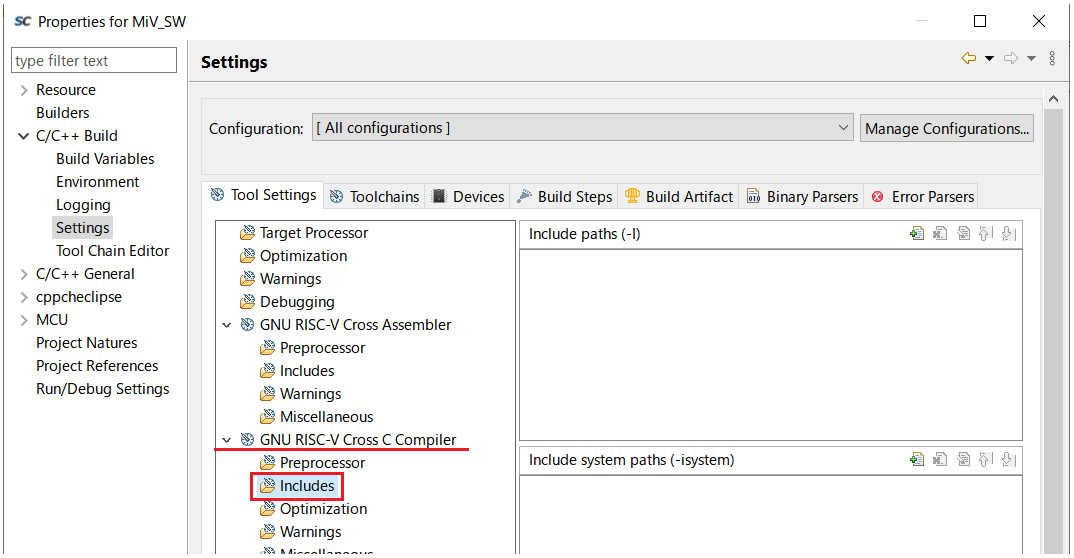
Architecture: RV32I (march=rv32i*)

Multiply extension: チェックを入れる

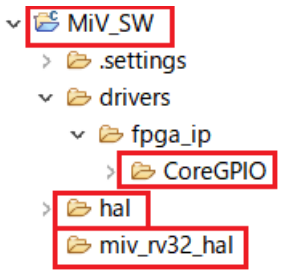
Integer ABI: ILP32 (-mabi=ilp32*)

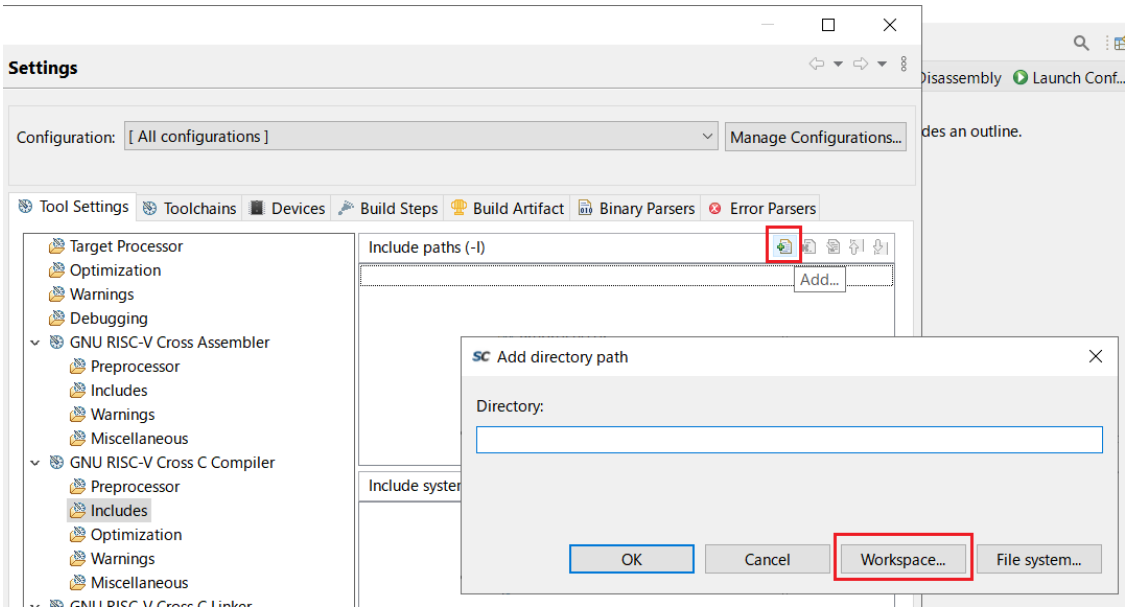


⑤ GNU RISC-V Cross C Compiler を展開し Includes を選択します。



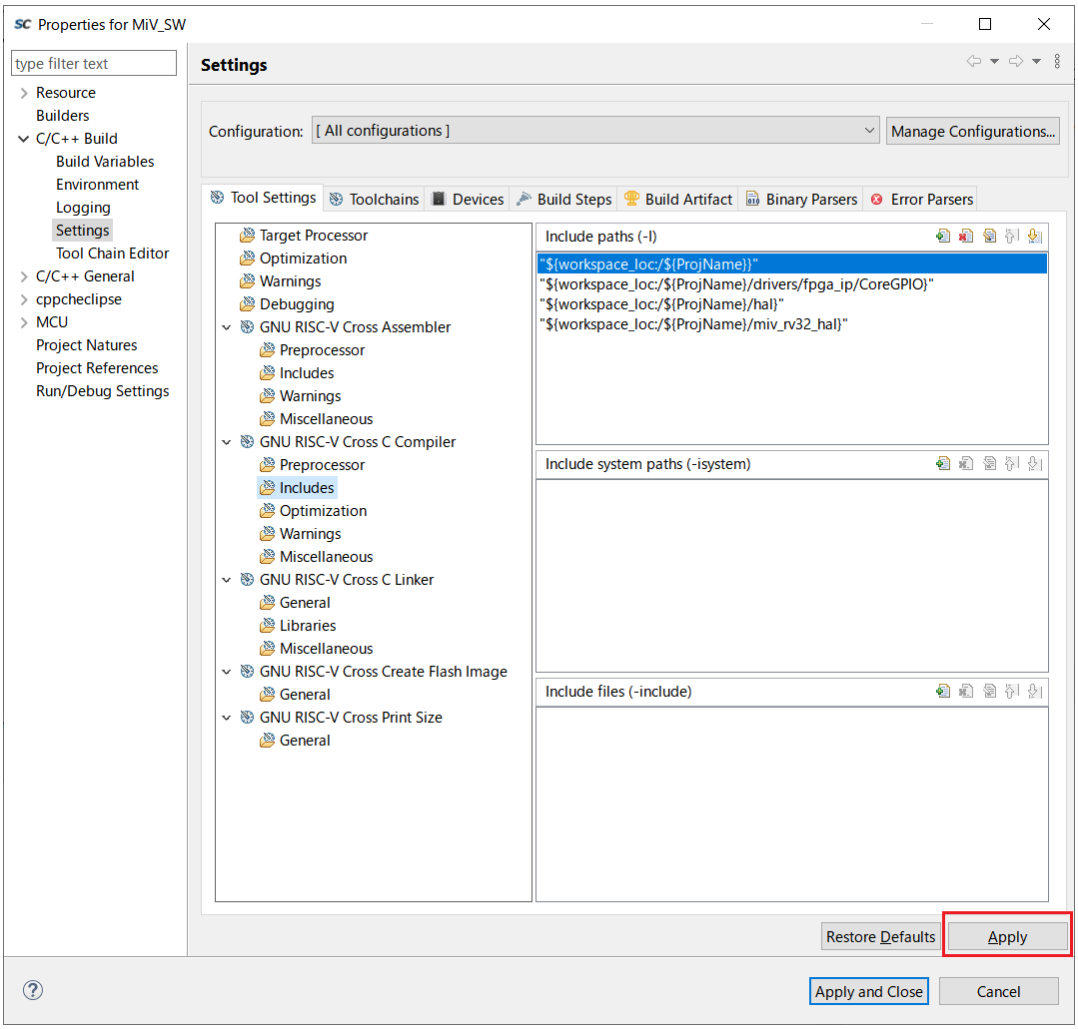
⑥ Include paths (-I) へ下記フォルダを追加します



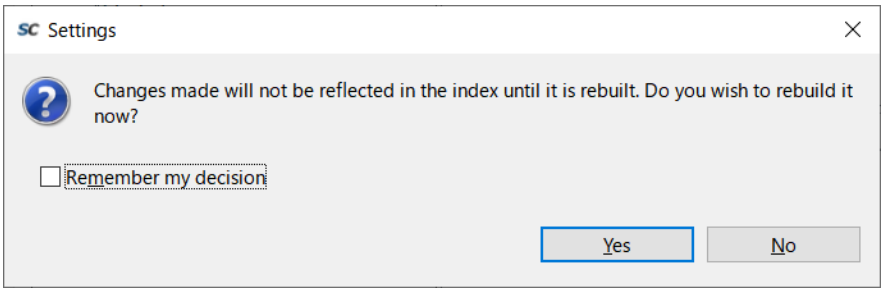


※ Ctrl キーで複数フォルダを一括選択、追加可能です。

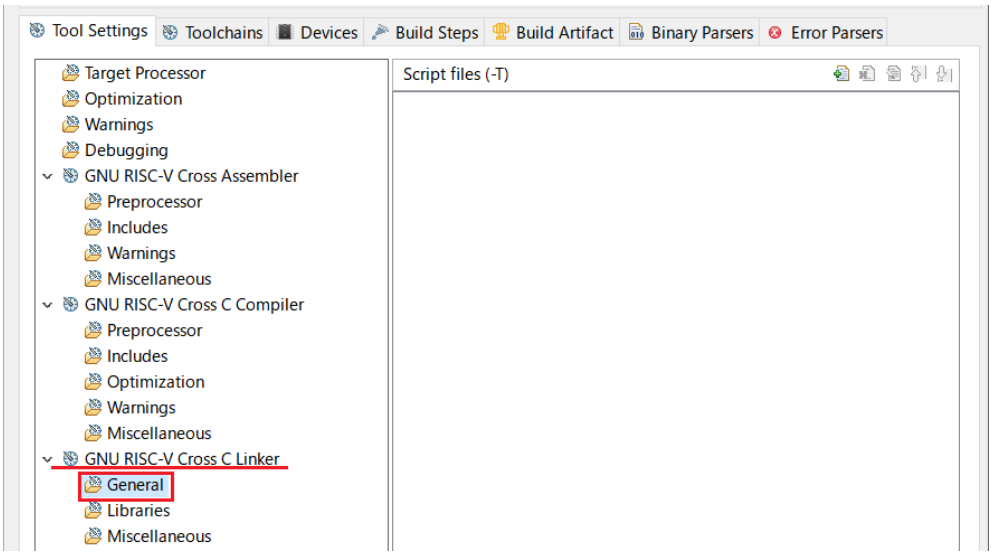
⑦ Apply をクリックします。



⑧ Yes をクリックします。

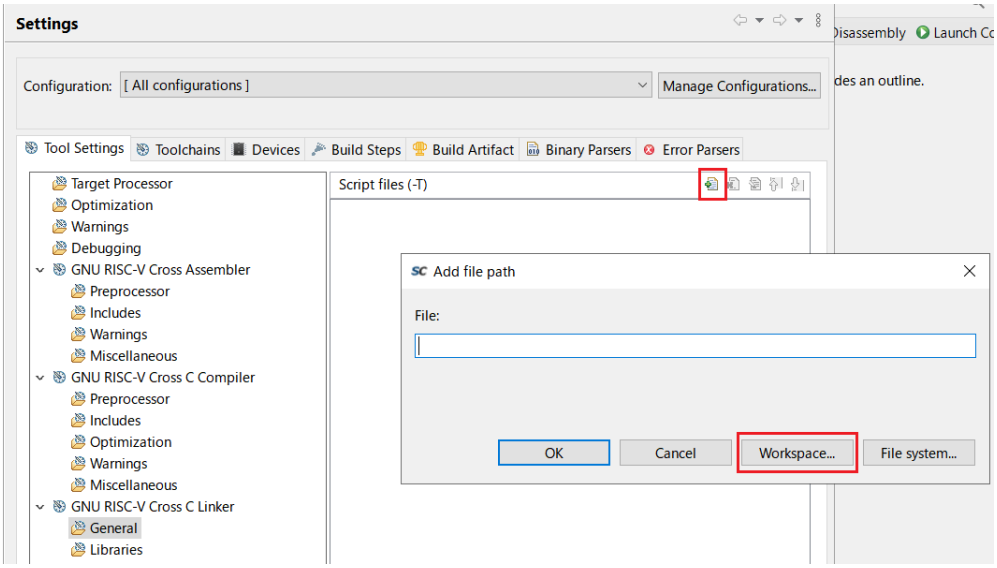


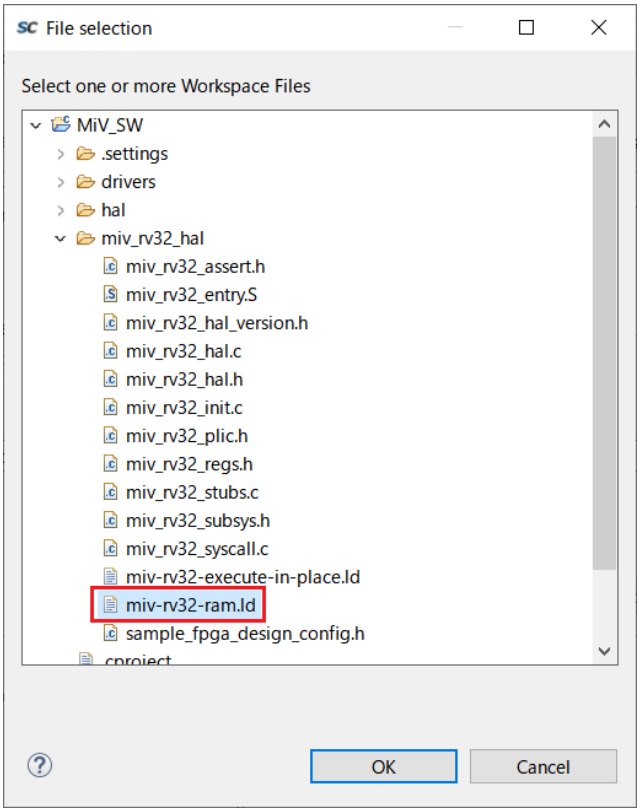
⑨ GNU RISC V Cross C Linker > General を選択します。



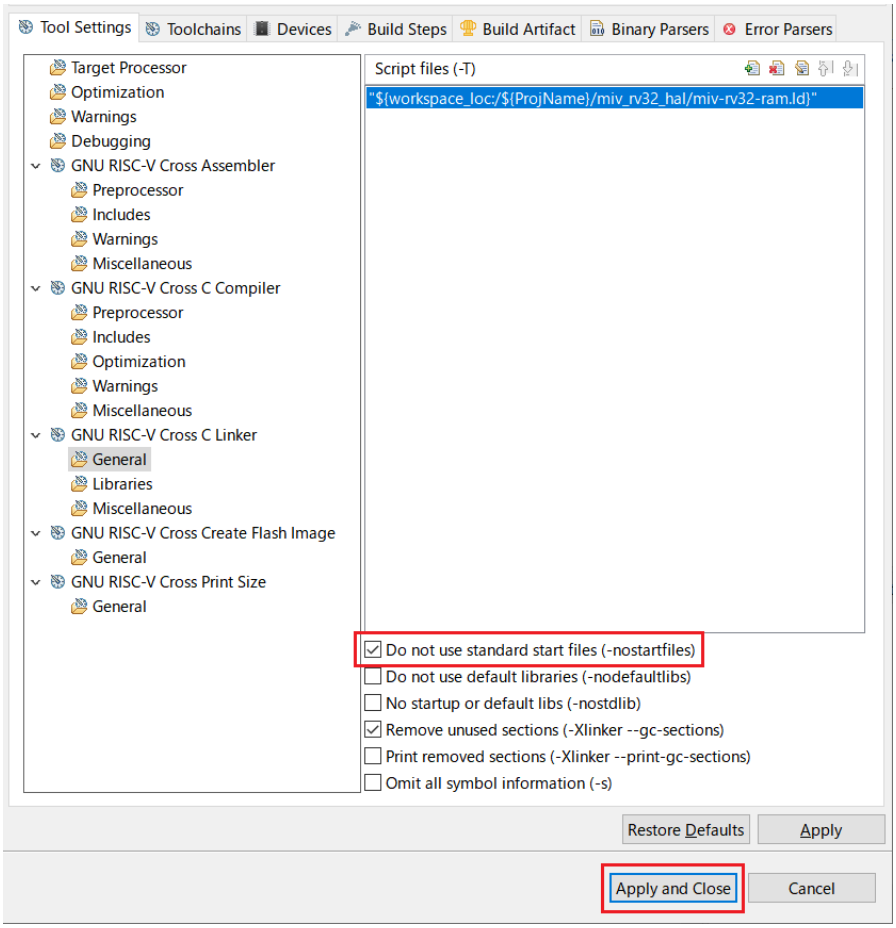
⑩ リンカースクリプトを追加します。

Script files (-T) 欄の Add...ボタンより、miv_rv32_hal 下の miv-rv32-ram.ld 下記ファイルを追加します。

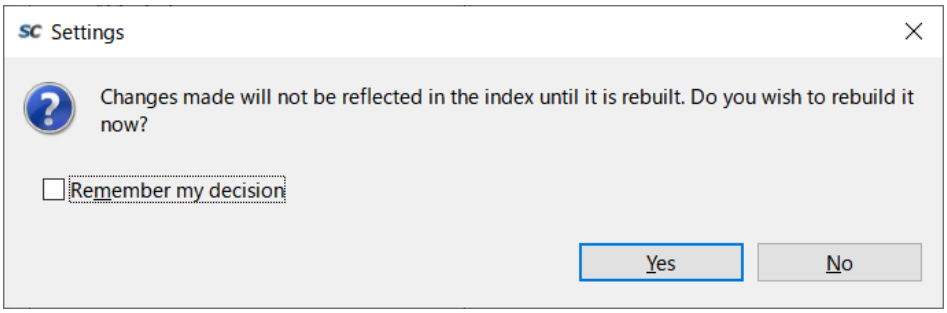




⑪ Do not use standard start files にチェックを入れ、Apply and Close をクリックします。

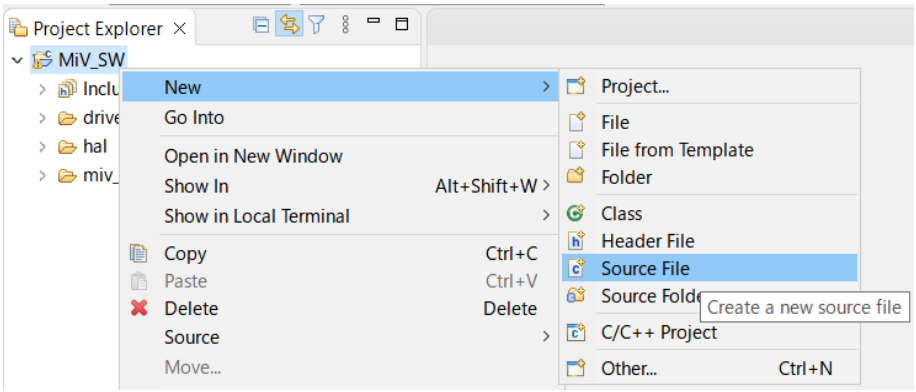


⑫ Yes をクリックします。

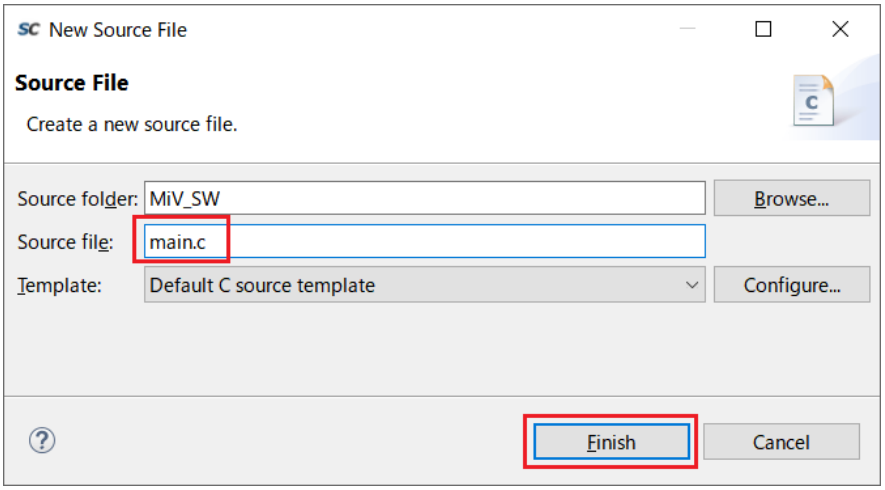


4-6. ソースコードの作成

① プロジェクトを右クリック ＞New＞Source File をクリックします。



② Source file にて main.c と入力し、Finish をクリックします。



③ 作成された main.c へ下記ソースコードを記入します。

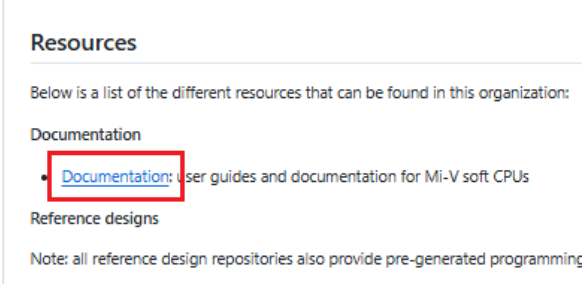
main.c

```
/*
 * main.c
 */

#include "drivers/fpga_ip/CoreGPIO/core_gpio.h" /* GPIOを使うため */
#include "miv_rv32_hal/fpga_design_config.h"
#include "miv_rv32_hal/miv_rv32_hal.h"
#include "hal/hw_reg_access.h"


/* g_gpio_outを宣言。
 * 構造体gpio_instance_tはcore_gpio.h内に定義されている。
 * ここでCoreGPIOに関するデータを保持する。
 */
gpio_instance_t g_gpio_out;


/* LED点滅用 delay() */
void delay(int count)
{
    volatile int i;
    for (i = 0; i < count; i++);
}


int main()
{
    /********************
     * GPIO_init()
     ********************/

    /*  GPIO_init()関数呼び出しによりCoreGPIO driverを初期化。
     *  他のGPIO  driver関数を呼び出す前にGPIO_init()関数呼び出しが必要。
```

```
     *  詳細はcore_gpio.hを参照。
     *
     *  void GPIO_init
     *  (
     *      gpio_instance_t *   this_gpio,
     *      addr_t              base_addr,
     *      gpio_apb_width_t    bus_width
     *  );
     *  第1引数: 構造体gpio_instance_tへのポインター。今回はg_gpio_outと宣言。
     *  第2引数: 初期化するGPIOのベースアドレス。
     *  第3引数: APBバス幅をdriverへ伝える。GPIO_APB_8_BITS_BUS or GPIO_APB_16_BITS_BUS or GPIO_APB_32_BITS_BUS
     *  戻り値: なし
     */

    GPIO_init(&g_gpio_out, COREGPIO_OUT_BASE_ADDR, GPIO_APB_32_BITS_BUS);


    /***************************************
     * GPIO_set_output(), GPIO_set_outputs()
     ***************************************/
    /*
     * GPIO port 1つの出力値を設定したい場合はGPIO_set_output()、複数GPIO port設定したい場合はGPIO_set_outputs()を使用
     * 詳細はcore_gpio.hを参照。
     *
     * void GPIO_set_outputs
     * (
     *    gpio_instance_t *   this_gpio,
     *    uint32_t            value
     * );
     *
     * void GPIO_set_output
     * (
```

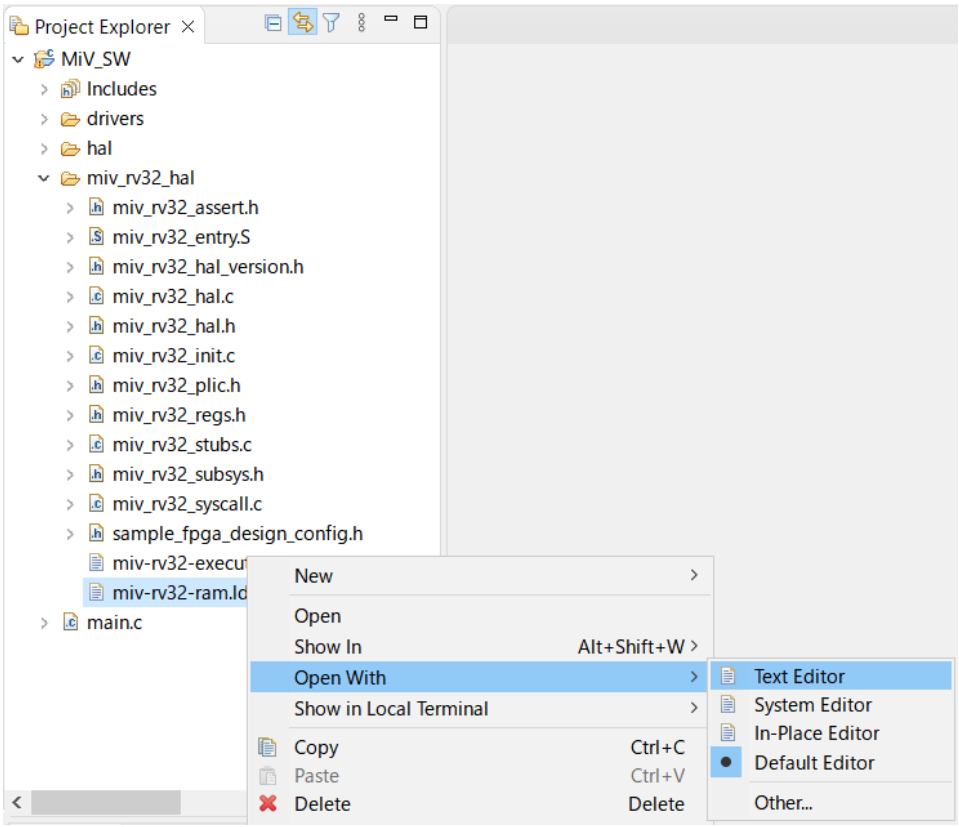
```
     *    gpio_instance_t *   this_gpio,
     *    gpio_id_t           port_id,
     *    uint8_t             value
     * );
     */

    /* 1 portずつ出力値を与える場合 */
    //GPIO_set_output( &g_gpio_out, GPIO_0,1 );
    //GPIO_set_output( &g_gpio_out, GPIO_1,0 );
    //GPIO_set_output( &g_gpio_out, GPIO_2,1 );
    //GPIO_set_output( &g_gpio_out, GPIO_3,0 );

    while (1)
    {
        /* GPIO_0～GPIO_3まで一括で出力値指定 */
        /* LED点灯 1010 */
        GPIO_set_outputs( &g_gpio_out, 0xA );
        delay(1000000);

        /* 点灯反転 0101 */
        GPIO_set_outputs( &g_gpio_out, 0x5 );
        delay(1000000);
    }

    return 0;
}
```

関数についての説明は、Driver のソースコード内、もしくは GitHub より確認可能です。https://github.com/Mi-V-Soft-RISC-V

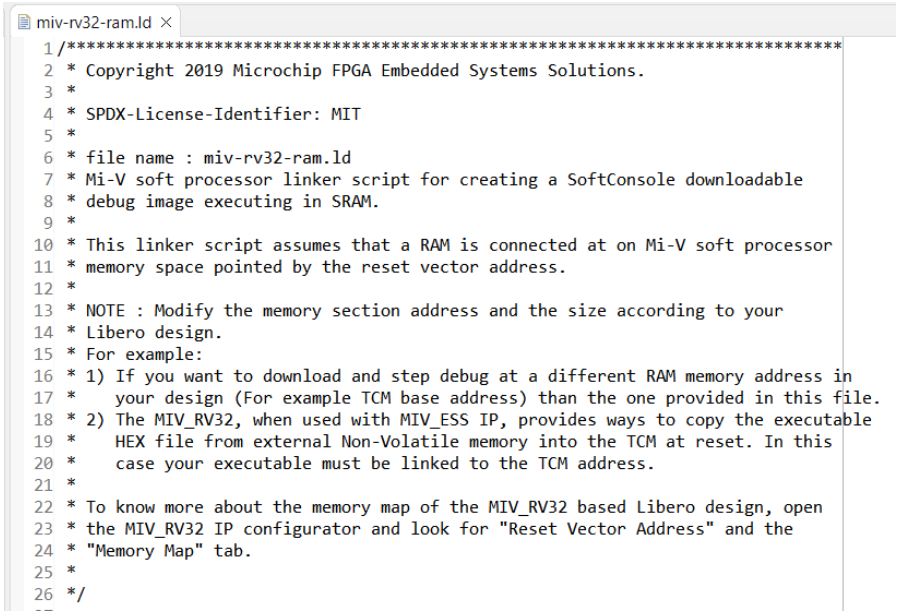


4-7. リンカスクリプトの編集

① miv-rv32-ram.ld ファイルを開きます。



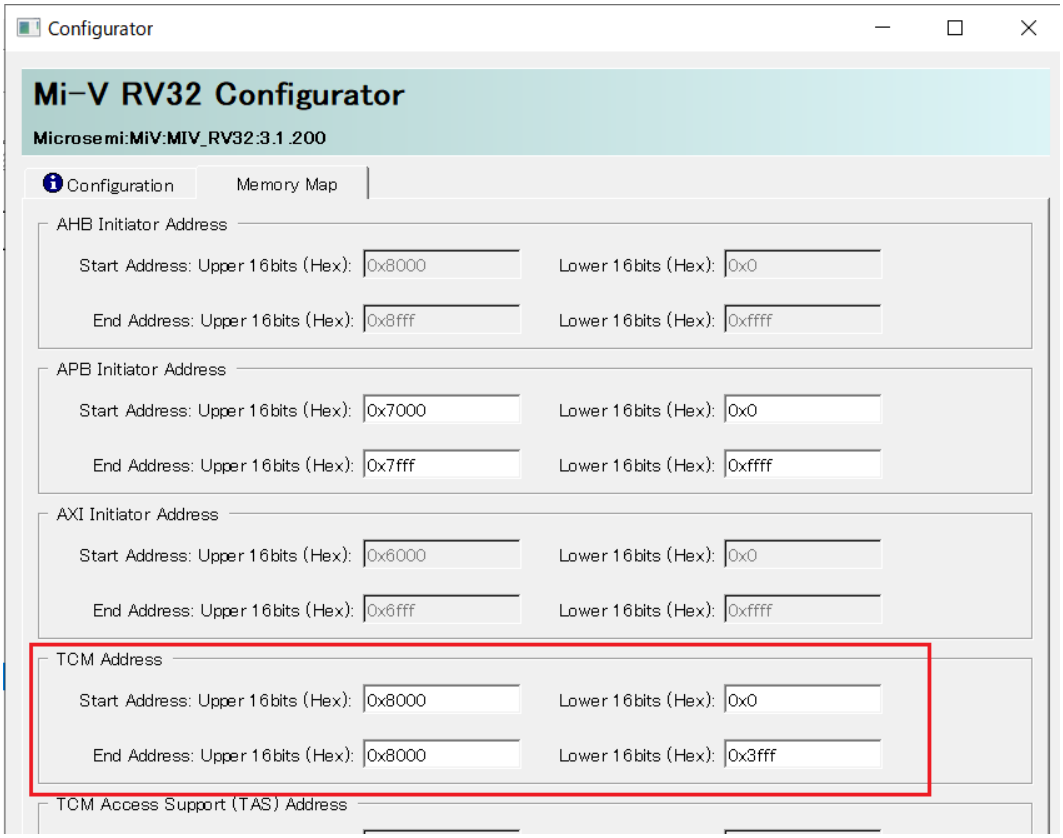
ファイルについての説明は上部に記載されています。



② 33 行目にて、
アドレスが 0x80000000 になっていることを確認、
LENGTH を 16k へ変更し保存します。

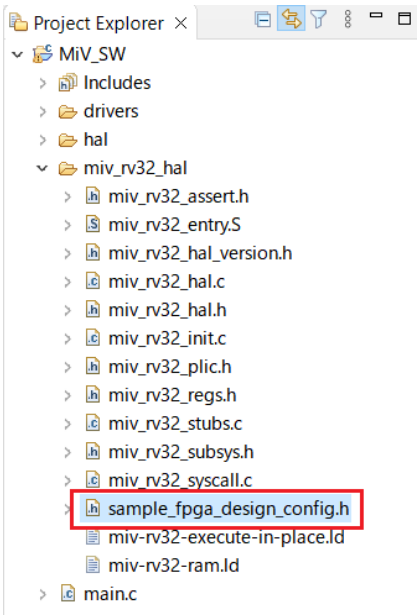
```
30
31 MEMORY
32 {
33     ram (rwx) : ORIGIN = 0x80000000, LENGTH = 16k
34 }
35
```

Libero SoC、MIV_RV32_C0_0 での設定 ：

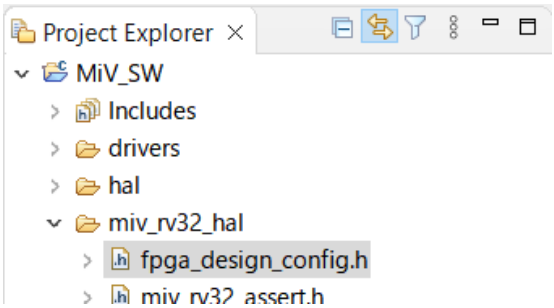


4-8. fpga_design_config.h の用意

miv_rv32_hal フォルダ下のサンプル、sample_fpga_design_config.h を流用します。



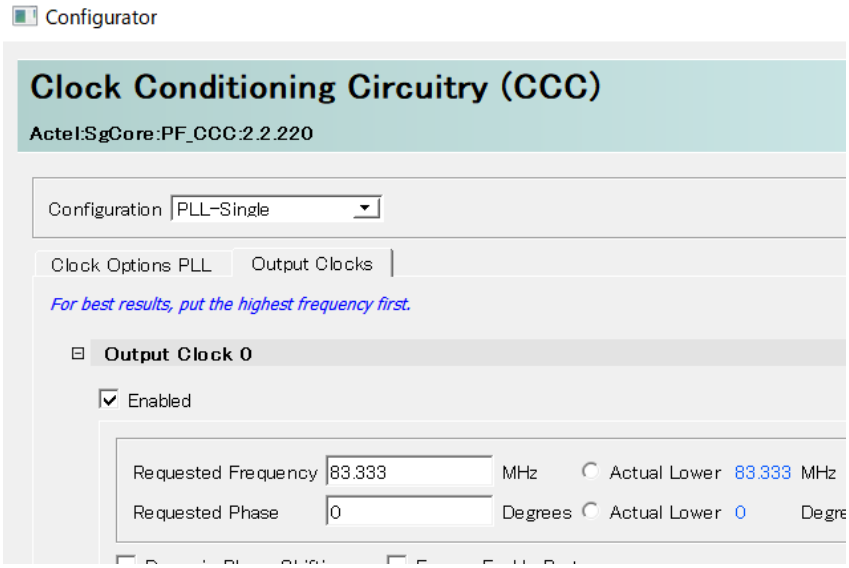
① 名前を sample_fpga_design_config.h から fpga_design_config.h へ変更し、
ファイルを開きます。



② #define SYS_CLK_FREQ を 83333000UL へ変更します。

```
49 /*****************************************************************************//**
50  * Soft-processor clock definition
51  * This is the only clock brought over from the Mi-V Libero design.
52  */
53 #ifndef SYS_CLK_FREQ
54 #define SYS_CLK_FREQ                    83333000UL
55 #endif
56
57 /*****************************************************************************//**
```

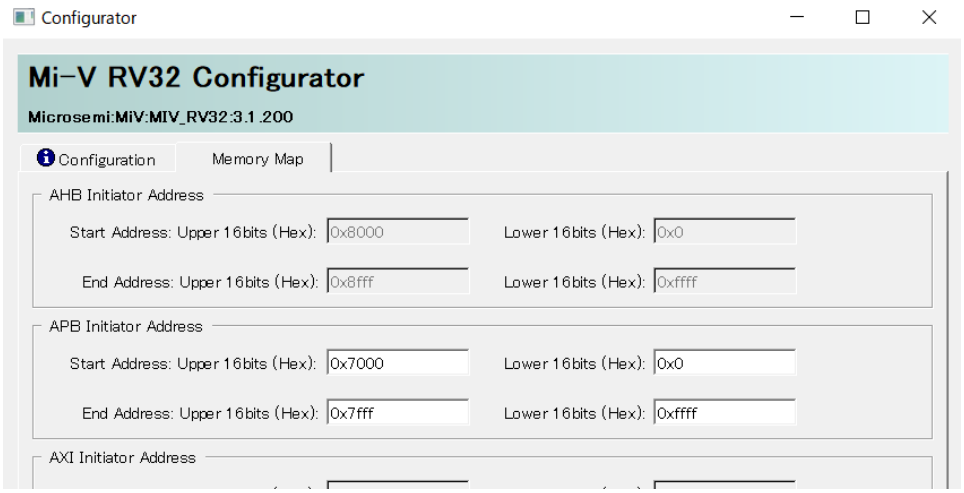
Libero SoC での CCC の Output Clock 設定 ：



③ GPIO のアドレスが 0x75000000UL となっていることを確認します。

```
57 /*****************************************************************************//**
58  * Peripheral base addresses.
59  * Format of define is:
60  * <corename>_<instance>_BASE_ADDR
61  * The <instance> field is optional if there is only one instance of the core
62  * in the design
63  * MIV_ESS is an extended peripheral subsystem IP core with peripherals
64  * connections as defined below.
65  * The system can be further extended by attaching APB peripherals to the
66  * empty APB slots.
67  */
68 #define MIV_ESS_PLIC_BASE_ADDR          0x70000000UL
69 #define COREUARTAPB0_BASE_ADDR          0x71000000UL
70 #define MIV_MTIMER_BASE_ADDR            0x72000000UL
71 #define MIV_ESS_APBSLOT3_BASE_ADDR      0x73000000UL
72 #define MIV_ESS_APBSLOT4_BASE_ADDR      0x74000000UL
73 #define COREGPIO_OUT_BASE_ADDR          0x75000000UL
74 #define CORESPI_BASE_ADDR               0x76000000UL
75 #define MIV_ESS_uDMA_BASE_ADDR          0x78000000UL
76 #define MIV_ESS_WDOG_BASE_ADDR          0x79000000UL
77 #define MIV_ESS_I2C_BASE_ADDR           0x7A000000UL
78 #define MIV_ESS_APBSLOTB_BASE_ADDR      0x7B000000UL
79 #define MIV_ESS_APBSLOTC_BASE_ADDR      0x7C000000UL
80 #define MIV_ESS_APBSLOTD_BASE_ADDR      0x7D000000UL
81 #define MIV_ESS_APBSLOTE_BASE_ADDR      0x7E000000UL
82 #define MIV_ESS_APBSLOTF_BASE_ADDR      0x7F000000UL
83
```

Libero SoC にて、今回は APB Initiator Address を 0x70000000～とし、MIV_ESS を接続しています。



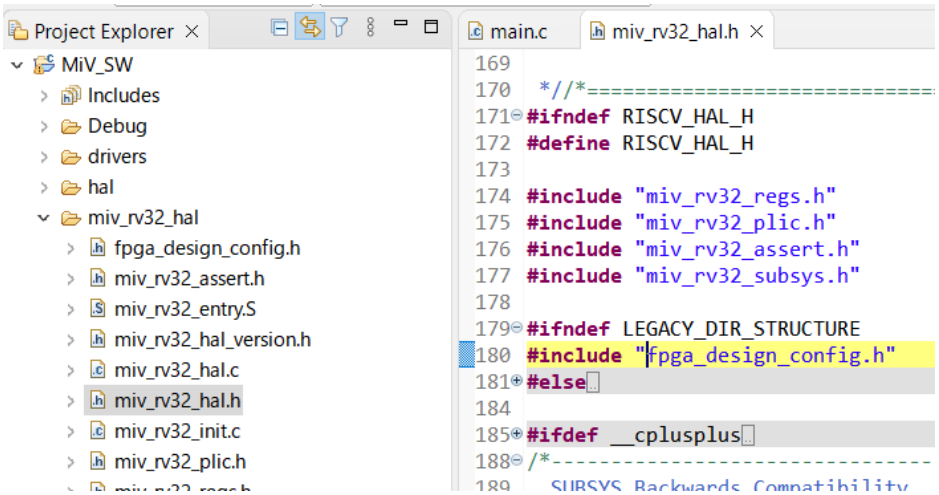
GPIO の offset address は
MIV_ESS User Guide > Table 2-2. Peripheral Module Address Offsets
から確認可能です。
https://ww1.microchip.com/downloads/aemDocuments/documents/FPGA/ProductDocuments/UserGuides/ip_cores/directcores/MIVESS.pdf#page=13

Base_Address_of ESS + Offset address of GPIO = CoreGPIO address
0x7000 0000 + 0x0500 0000 = 0x7500 0000

4-9. #include パスの修正

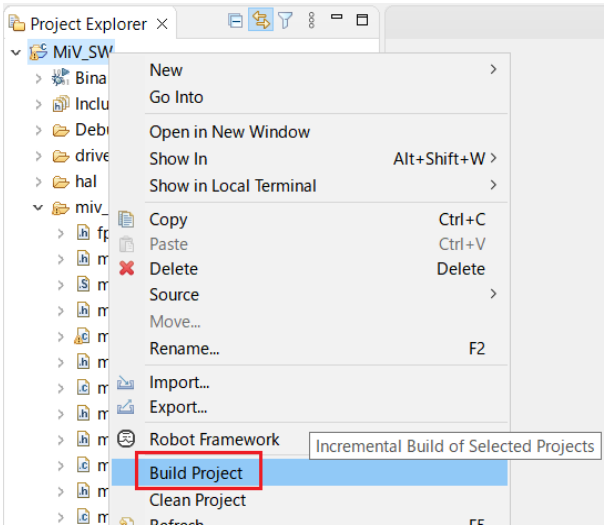
プロジェクトに Import した GitHub のファイル内には、fpga_design_config.h を呼び出しているものがあります。
現在の構成に合わせてパス表記を変更します。

miv_rv32_hal 内 miv_rv32_hal.h ファイルを開き
#include " fpga_design_config /fpga_design_config.h"を
#include "fpga_design_config.h"へ変更



4-10. ビルド

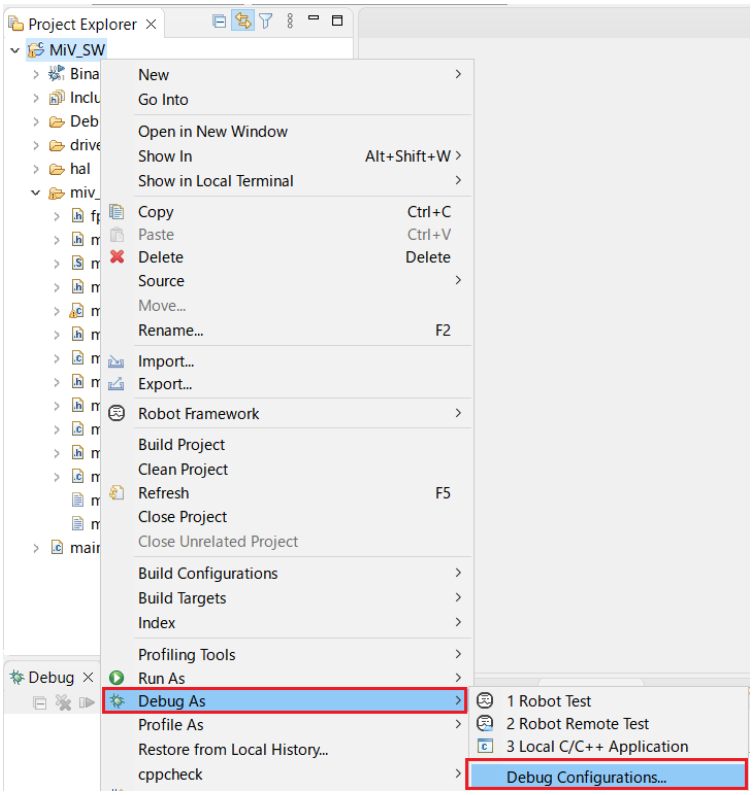
プロジェクトを右クリックし、Build Project からビルドします。



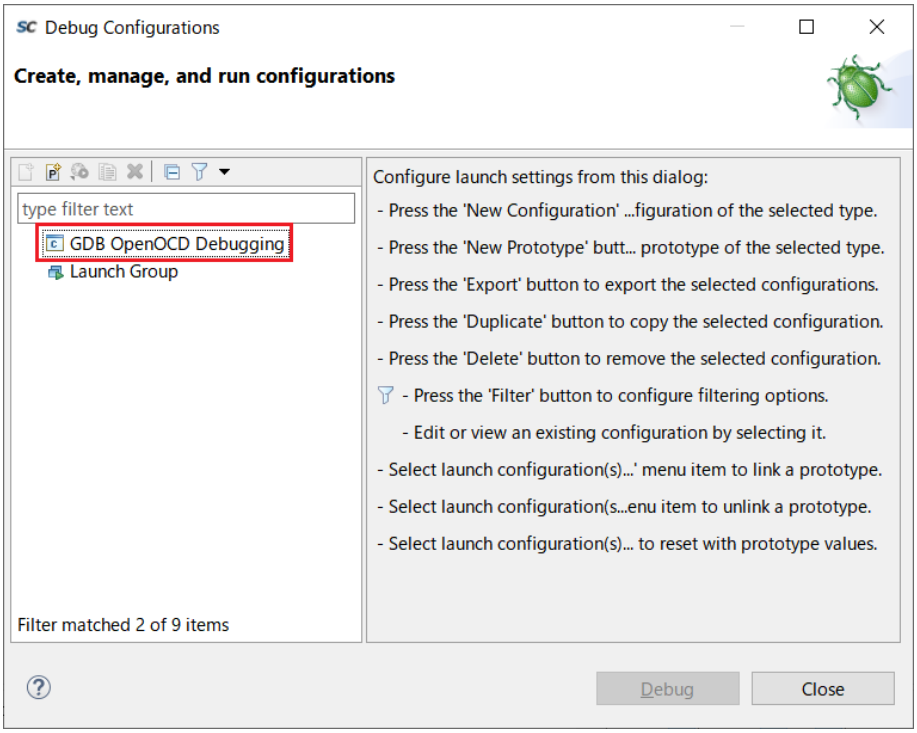
5. 実行

① 開発キットが PC に接続されていることを確認します。

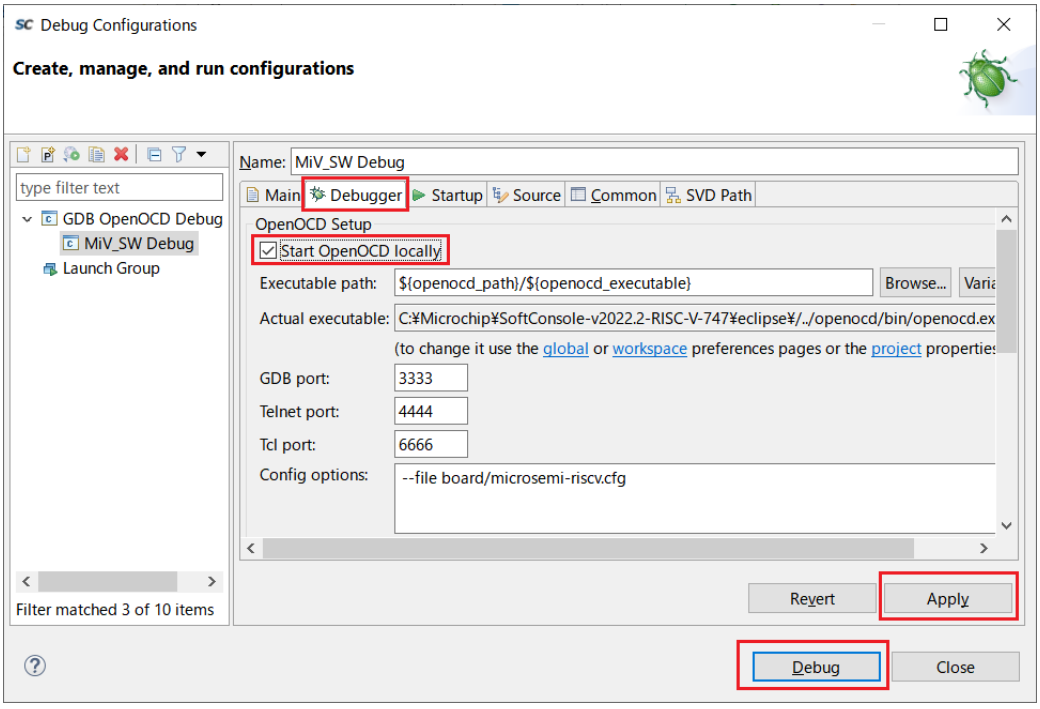
② プロジェクトを右クリックし、Debug As > Debug Configurations...をクリックします。



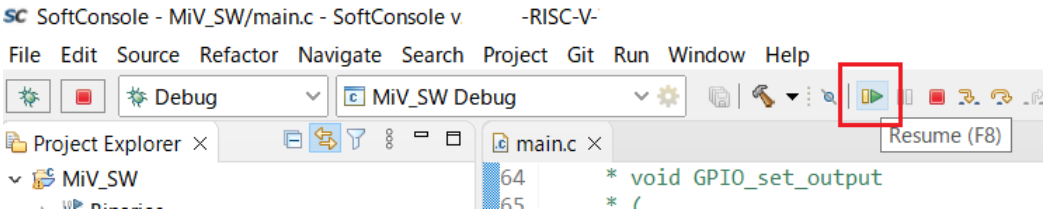
③ GDB OpenOCD Debugging をダブルクリックします。



④ Debugger タブを開き、Start OpenOCD locally へチェックを入れ
Apply 後、Debug をクリックします。



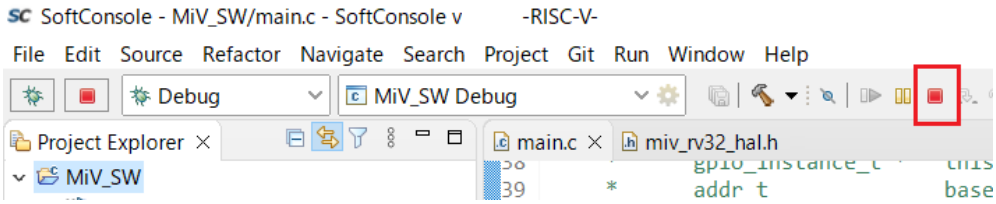
⑤ ▶の Resume ボタンをクリックします。



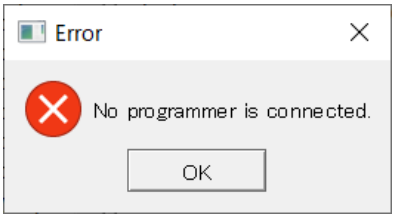
⑥ Discovery Kit にて LED の点滅を確認します。

⑦ SoftConsole にて Debug 実行を止めます。

※ SoftConsole からの Debug 実行と、Libero SoC からの書き込みは
同時にできません。
必要に応じて適時 Debug 実行は停止して下さい。



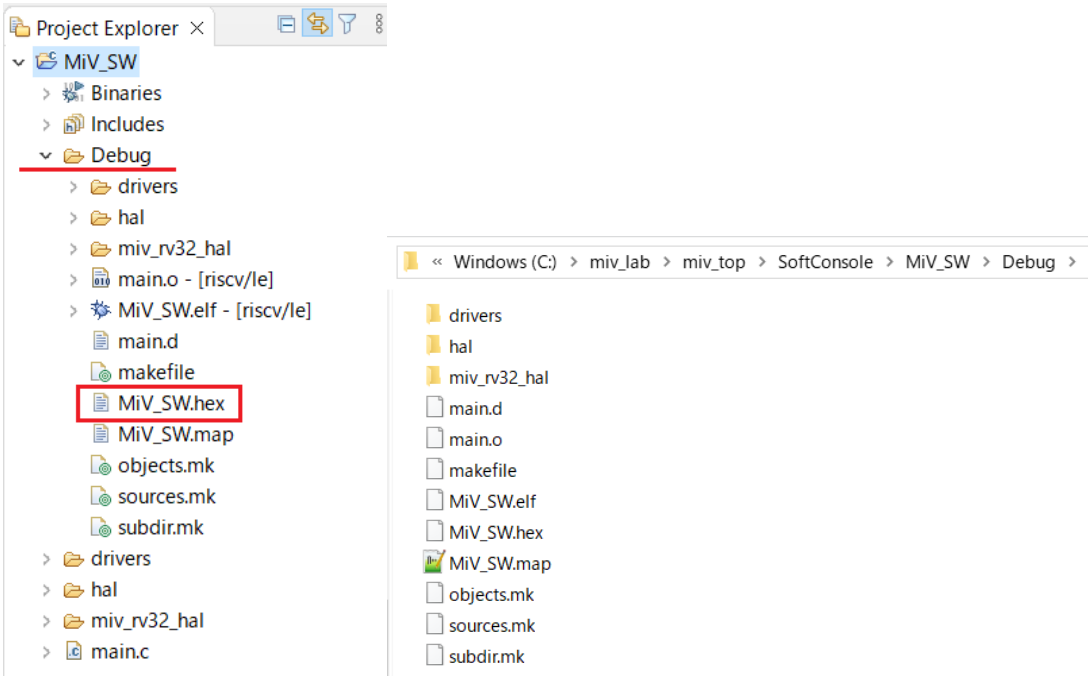
SoftConsole の Debug を止めずに、Libero SoC にて Run PROGRAM Action を行った場合:



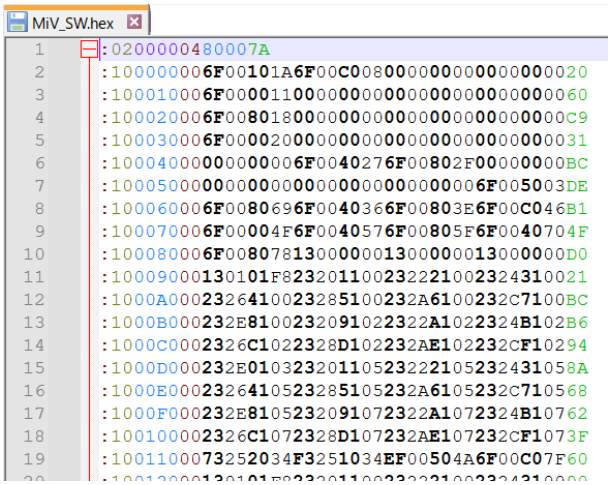
6. ソフトウェアをデバイスへ書き込んでみよう

※ ここでは一例として TCM の initialize ファイルとして割り当てます。
※ 例えば PolarFire SRAM (AHBLite and AXI)ブロックを使っている場合は
同様の方法で SRAM の initialize ファイルとして割り当てます。
※ Discovery Kit 上には SPI Flash が搭載されていないため、SW を SPI Flash へ
書き込むことはできません。

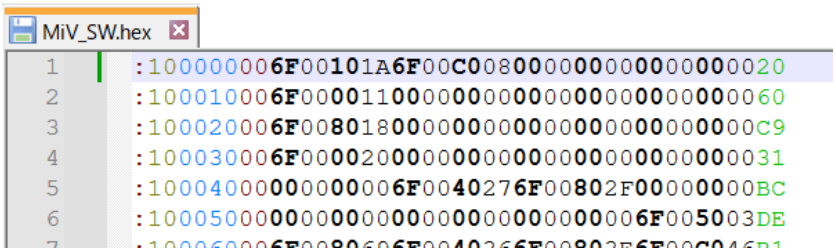
① SoftConsole にて、ビルド後 Debug フォルダ下に MiV_SW.hex が生成されていることを確認します。



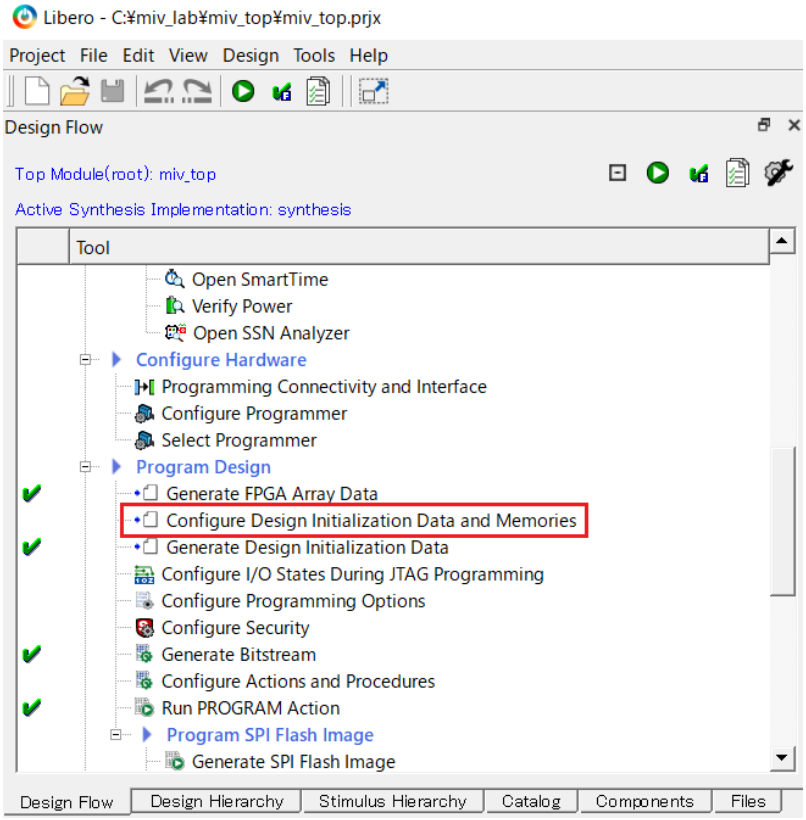
② 任意のテキストエディタで hex ファイルを開きます。



③ 1 行目を削除し、保存します。

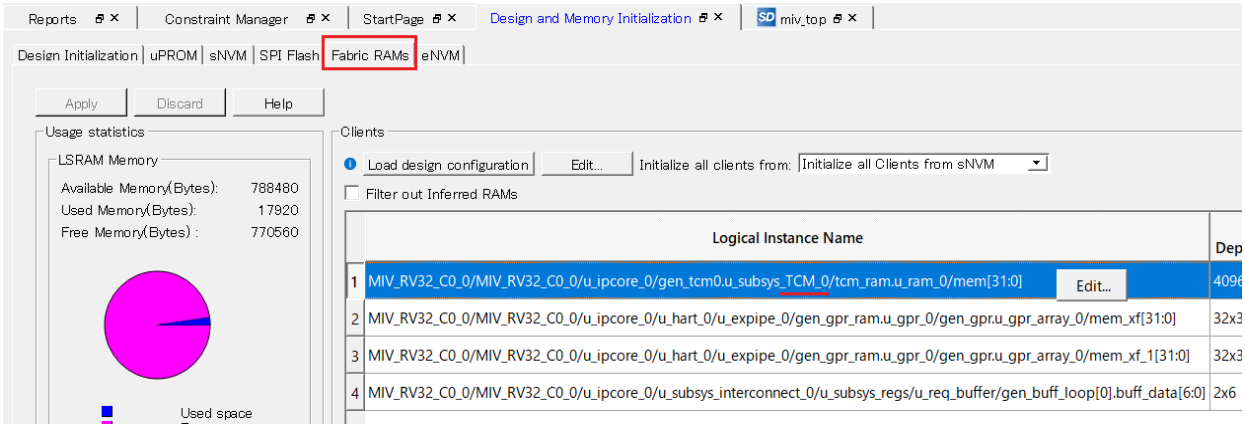


④ Libero SoC にて、Design Flow タブ内で
Configure Design Initialization Data and Memories をダブルクリックします。



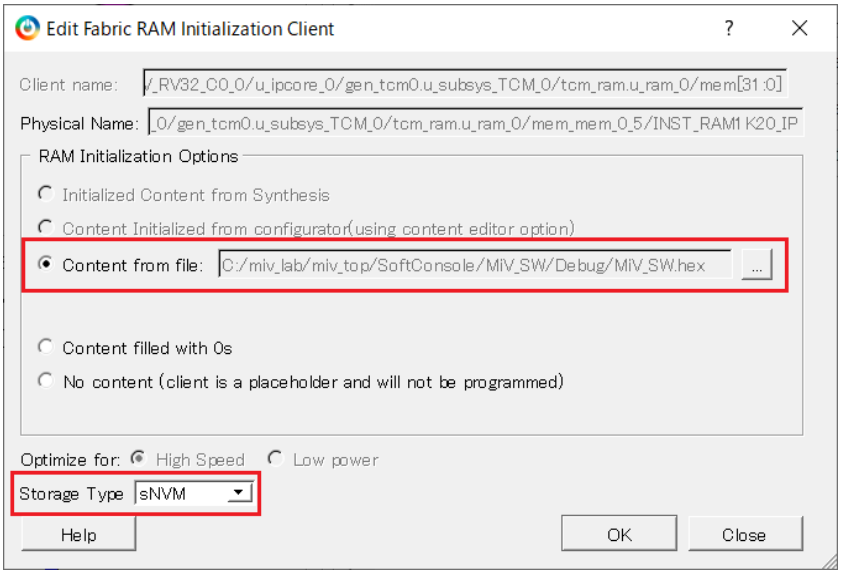
⑤ Fabric RAMs のタブを開きます。

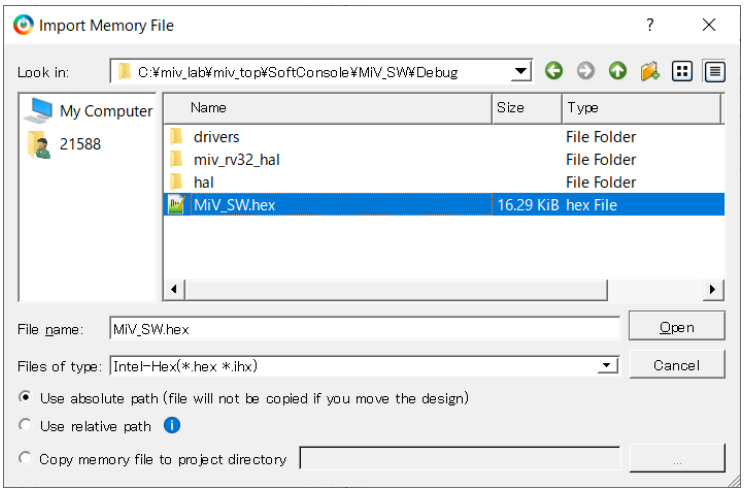
Logical Instance Name から TCM を見つけ、右クリック、Edit...をクリックします。



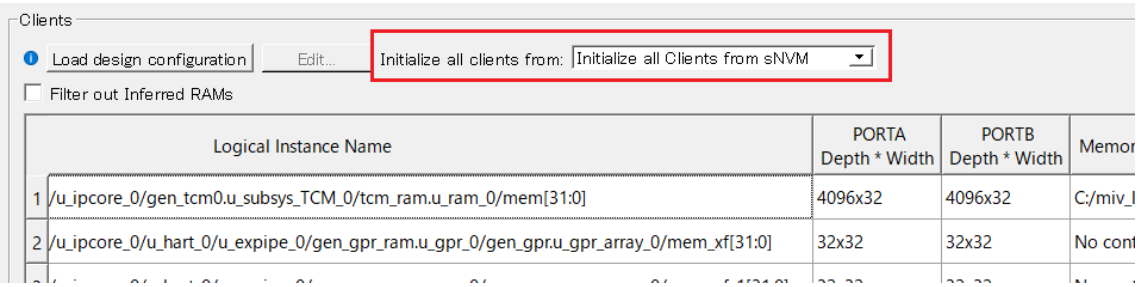
⑥ Content from file を選択し、...ボタンから MiV_SW.hex を選択します。

Storage Type はデフォルトの sNVM のままとします。

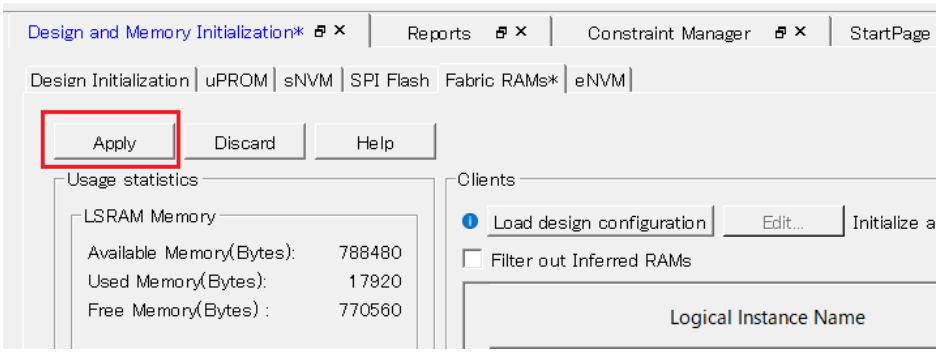




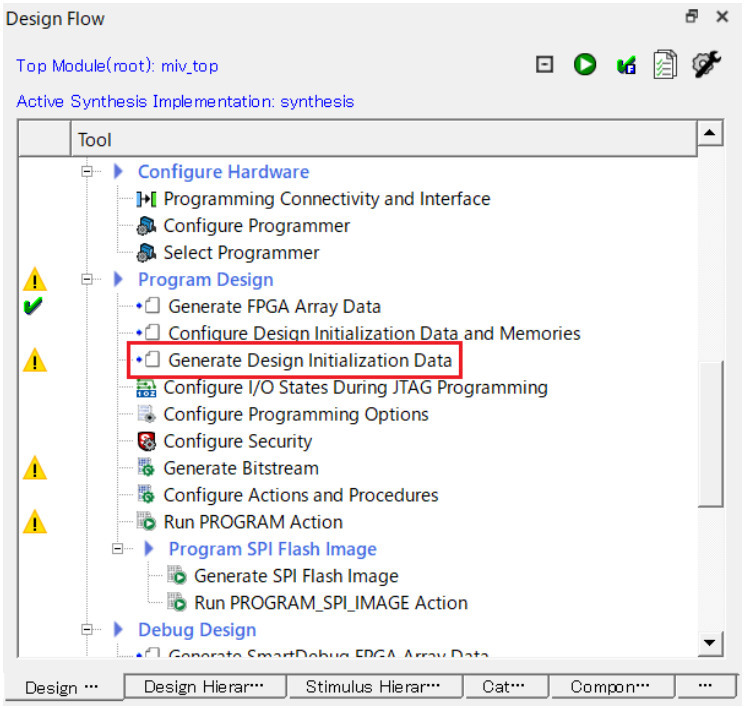
⑦ Initialize all clients from が Initialize all Clients from sNVM となっていることを確認します。



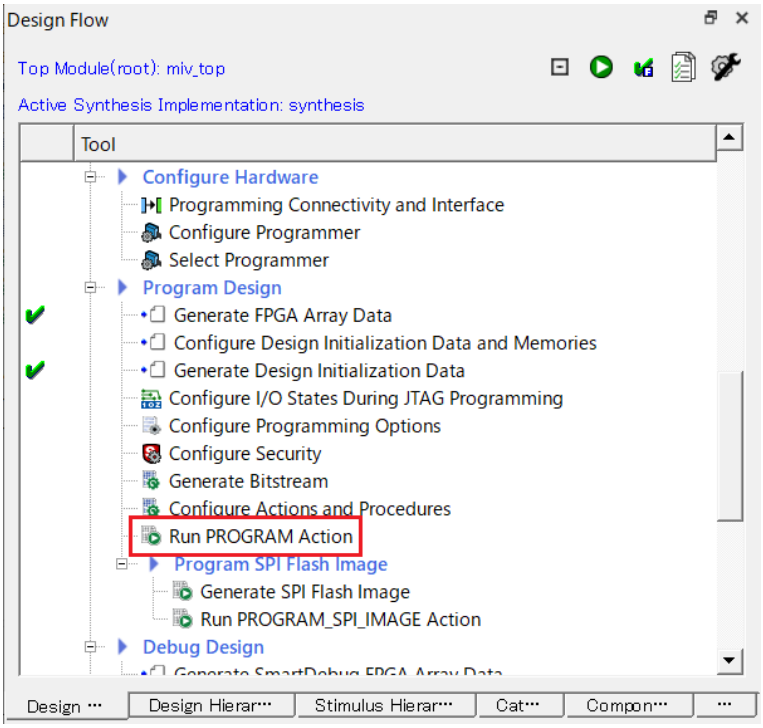
⑧ Apply します。



⑨ Generate Design Initialization Data をダブルクリックします。



⑩ Run PROGRAM Action をダブルクリックし書き込みます。



⑪ Discovery Kit のケーブルを抜き差しし SW が書き込まれていること
(SoftConsole からの Debug 実行なしで LED が点滅すること)を確認します。

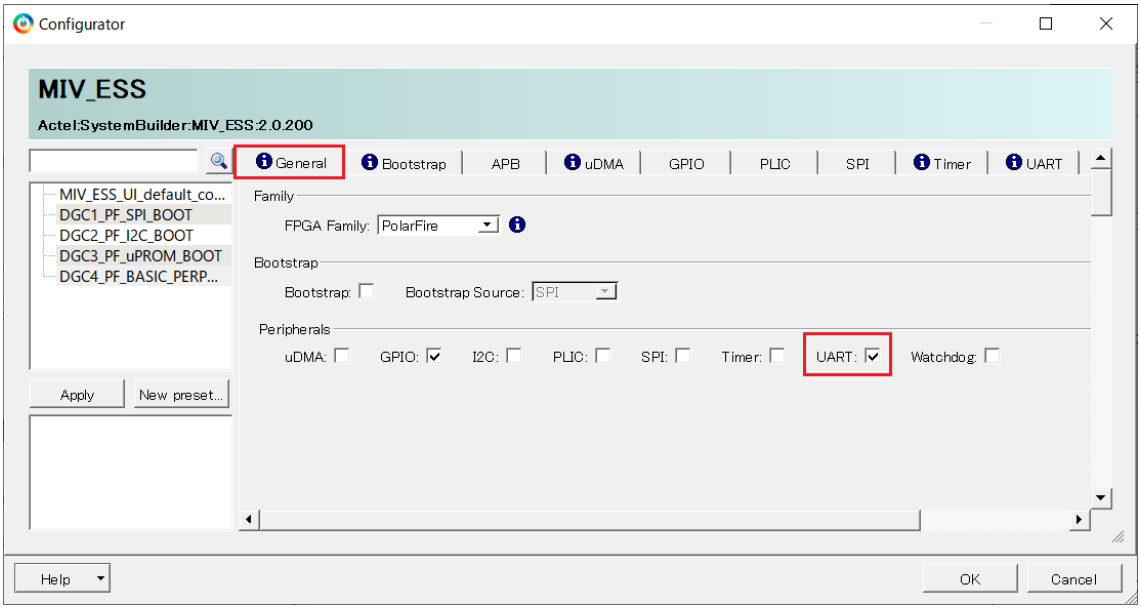
7. UART を動かしてみよう

既存の L チカのプロジェクトを流用し Hello World を表示させます。

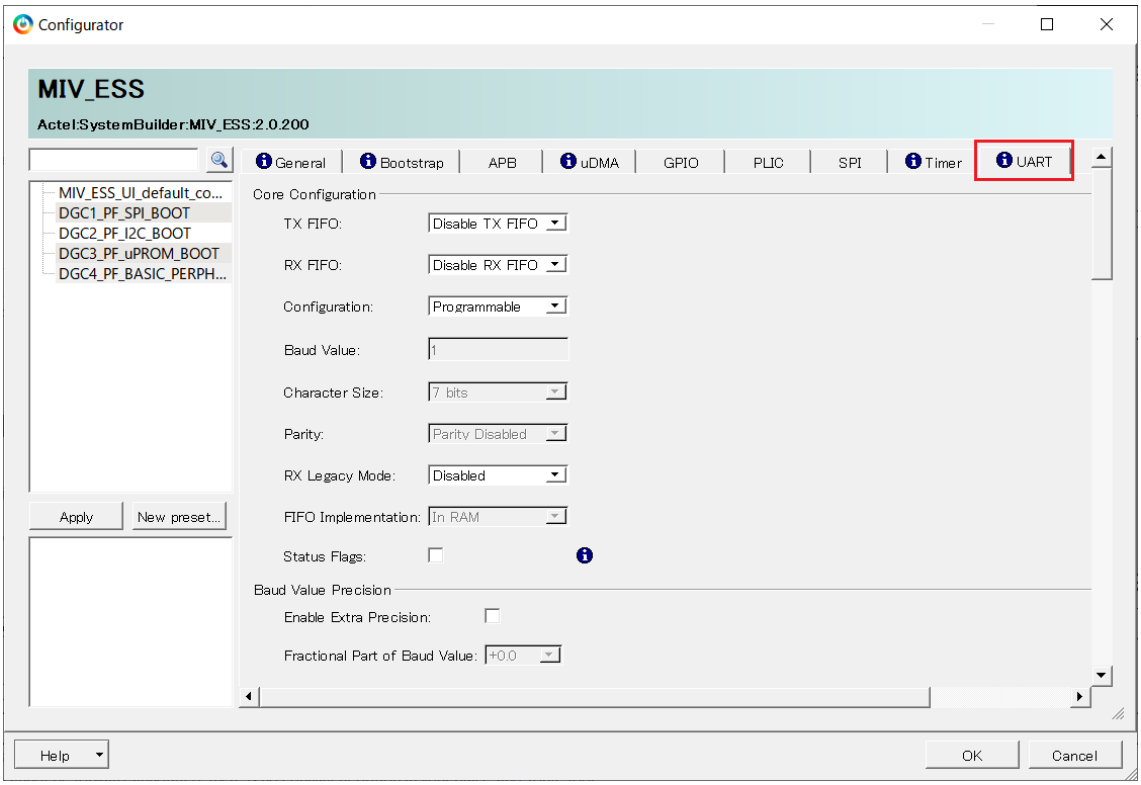
7-1. ハードウェア

① Libero SoC にて MIV_ESS_C0_0 ブロックを開きます。

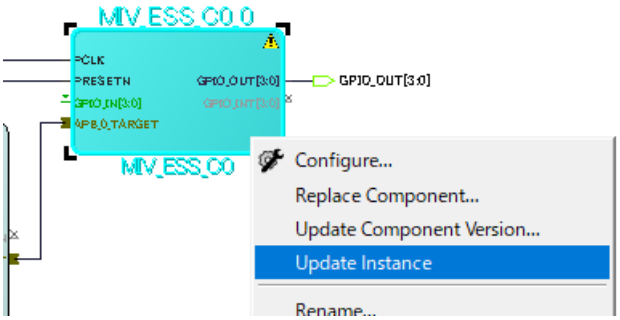
② General タブにて UART へチェックを入れます。



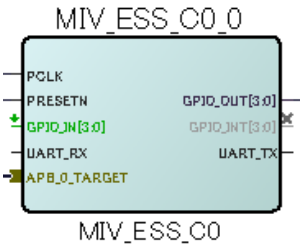
UART の詳細設定は UART タブにて可能です。
今回はデフォルトのままとします。



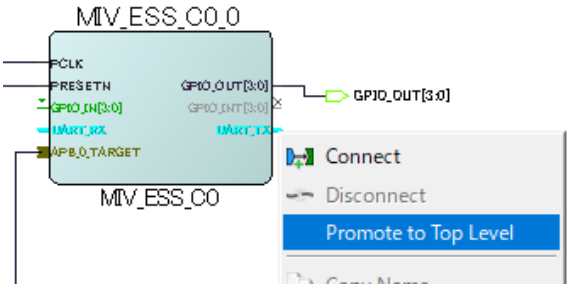
③ MIV_ESS_C0_0 ブロックを右クリック、Update Instance をクリックし
ブロックをアップデートします。

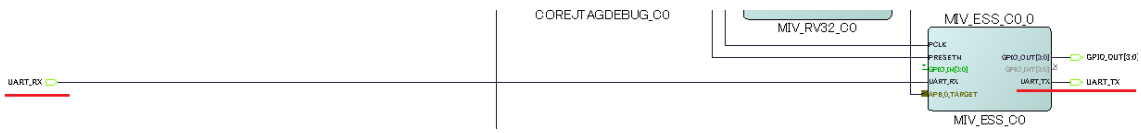


UART_RX、UART_TX のピンが表示されます。

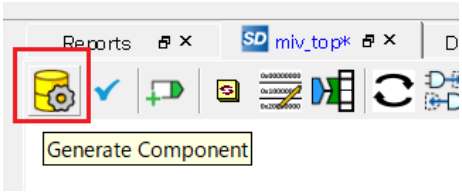


④ Promote to Top Level を使用し、UART_RX、UART_TX のポートを出します。

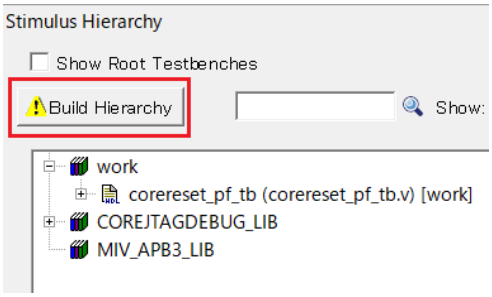




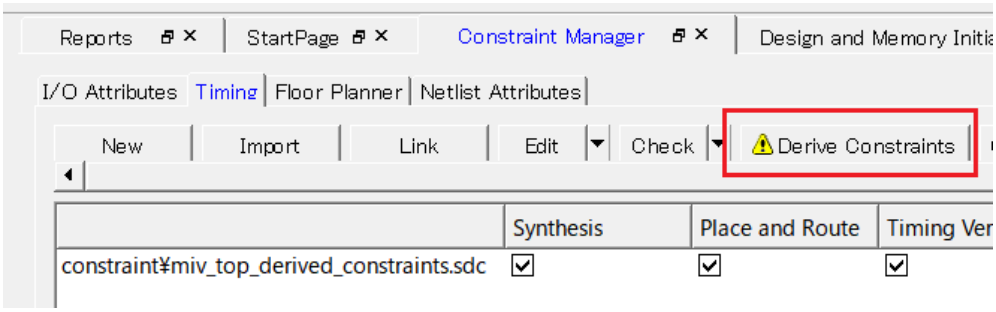
⑤ Smart Design を保存し、Generate Component をクリックします。



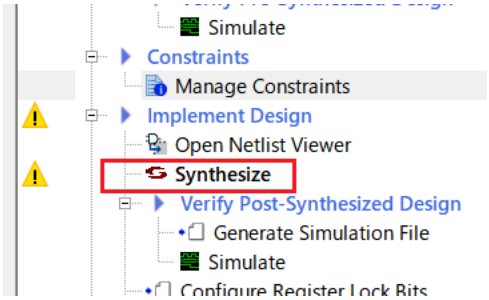
⑥ Build Hierarchy をクリックします。



⑦ Constraint Manager を開き、Timing タブにて Derive Constraints をクリックし、sdc ファイルを再生成します。



⑧ 論理合成を行います。



⑨ Constraint Manager より I/O editor を開き、UART_RX、UART_TX を
ピンアサインします。
UART_RX : W21
UART_TX : Y21

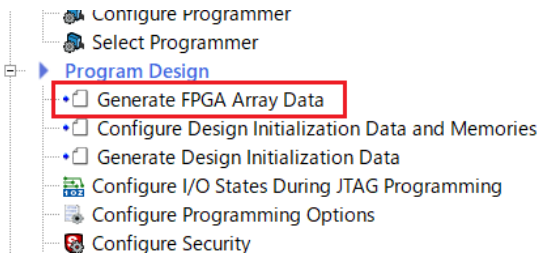
Pin View | Port View [active] | XCVR View | Memory View | IOD View | Package View | Floorplanr

	Port Name	Direction	I/O Standard	Pin Number	Locked	Macro Cell	Bank Name
1	GPIO_OUT				☐		
2	GPIO_OUT[0]	OUTPUT	LVCMOS18	T18	☑	OUTBUF	Bank0
3	GPIO_OUT[1]	OUTPUT	LVCMOS18	V17	☑	OUTBUF	Bank0
4	GPIO_OUT[2]	OUTPUT	LVCMOS18	U20	☑	OUTBUF	Bank0
5	GPIO_OUT[3]	OUTPUT	LVCMOS18	U21	☑	OUTBUF	Bank0
6	REF_CLK_0	INPUT	LVCMOS18	R18	☑	INBUF	Bank0
7	TCK	INPUT	--	A8	☑	UJTAG_SEC	--
8	TDI	INPUT	--	A9	☑	UJTAG_SEC	--
9	TDO	OUTPUT	--	A7	☑	UJTAG_SEC	--
10	TMS	INPUT	--	B7	☑	UJTAG_SEC	--
11	TRSTB	INPUT	--	B9	☑	UJTAG_SEC	--
12	UART_RX	INPUT	LVCMOS18	W21	☑	INBUF	Bank0
13	UART_TX	OUTPUT	LVCMOS18	Y21	☑	OUTBUF	Bank0

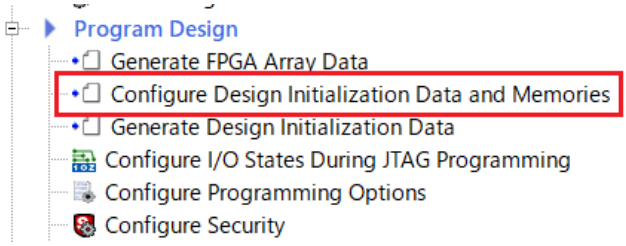
UART_RX、UART_TX のピン番号は、Discovery Kit の回路図より確認可能です。
https://www.microchip.com/en-us/development-tool/mpfs-disco-kit

⑩ Place and Route(配置配線)を行います。

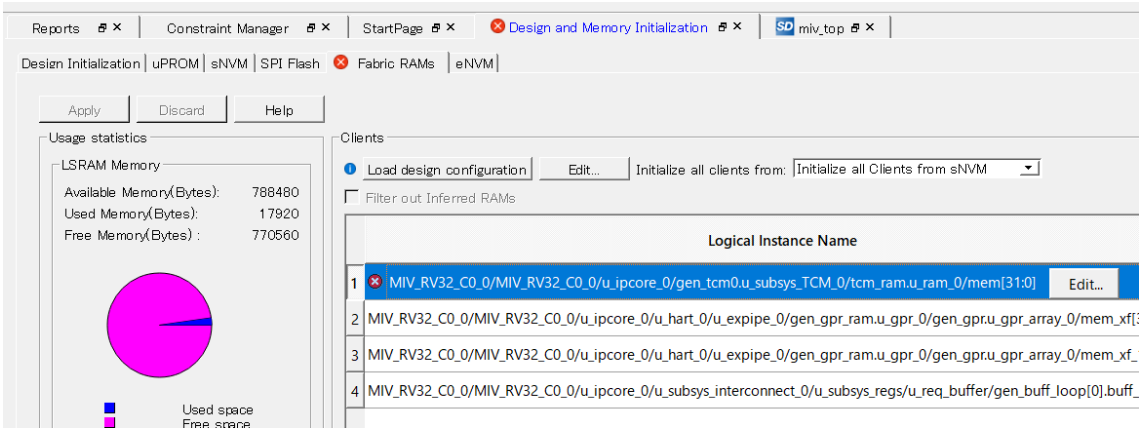
⑪ Generate FPGA Array Data をダブルクリックします。
(Configure Design Initialization Data and Memories を開くため事前に実行が必要。)

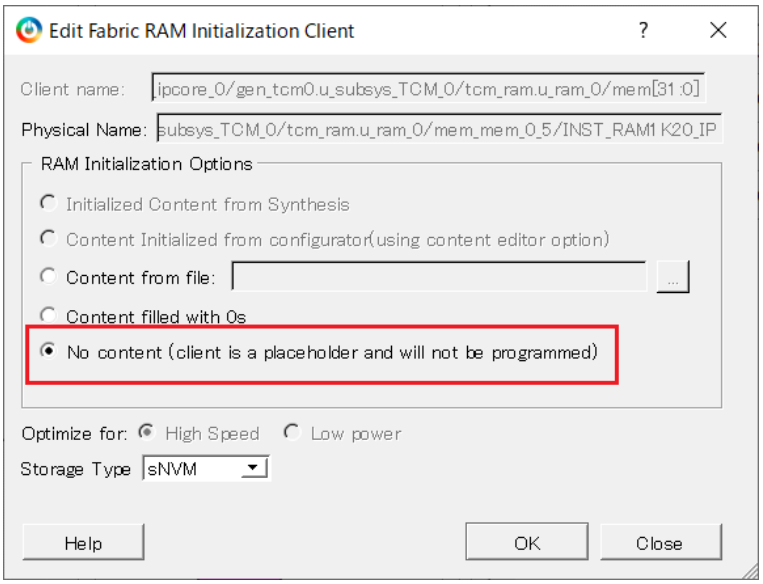


⑫ Configure Design Initialization Data and Memories を開きます。



⑬ L チカの hex ファイルが設定されているので削除します。
TCM を右クリック、Edit で開き、No content を選択します。

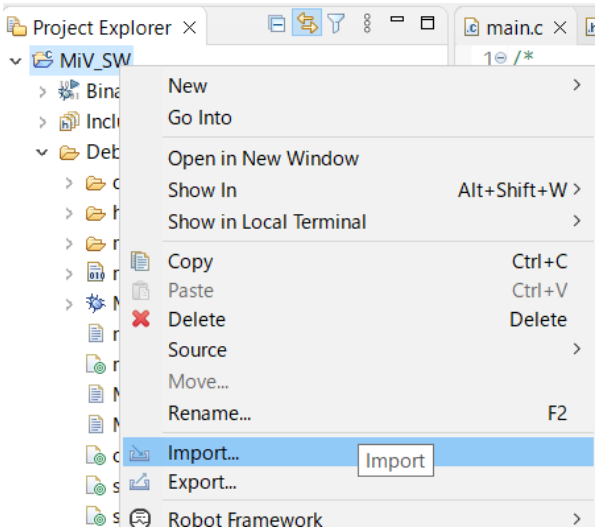


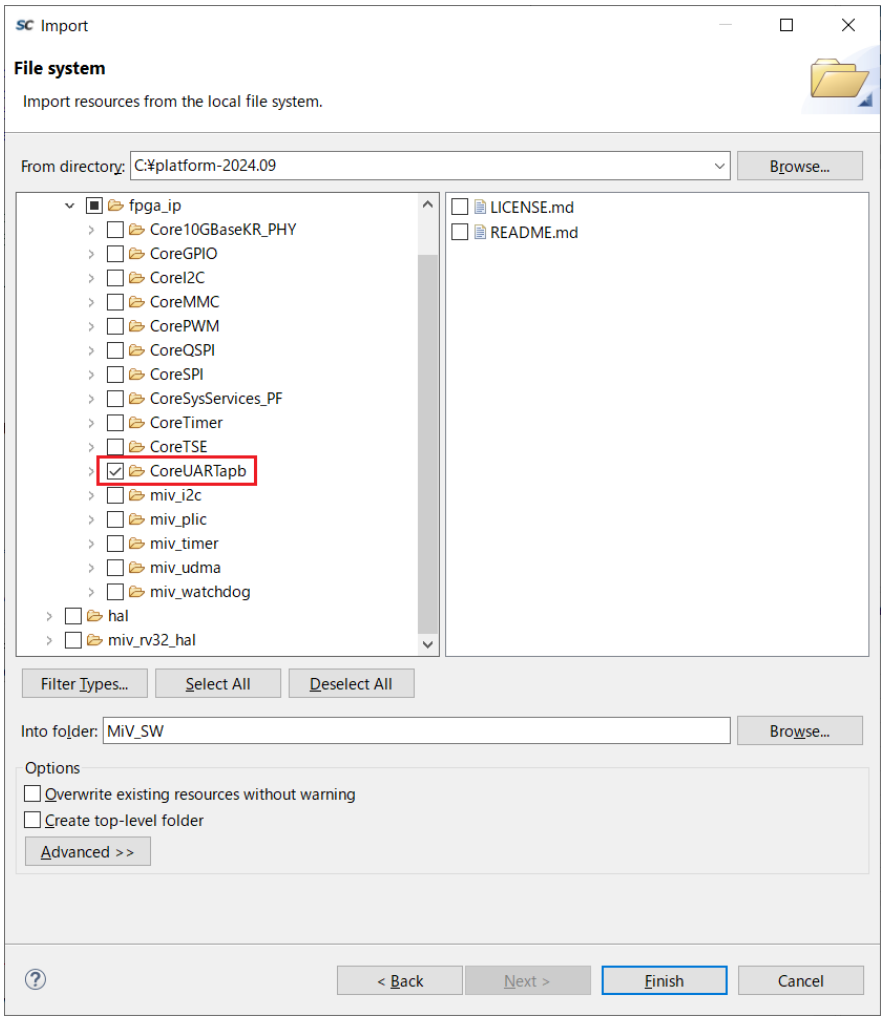


⑭ Run PROGRAM Action にて書き込みます。

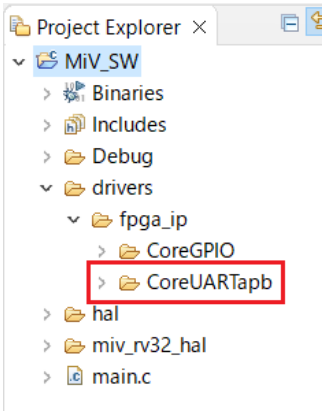
7-2. ソフトウェア

① SoftConsole にて、GitHub からダウンロード、解凍した Driver から CoreUARTapb を追加でインポートします。

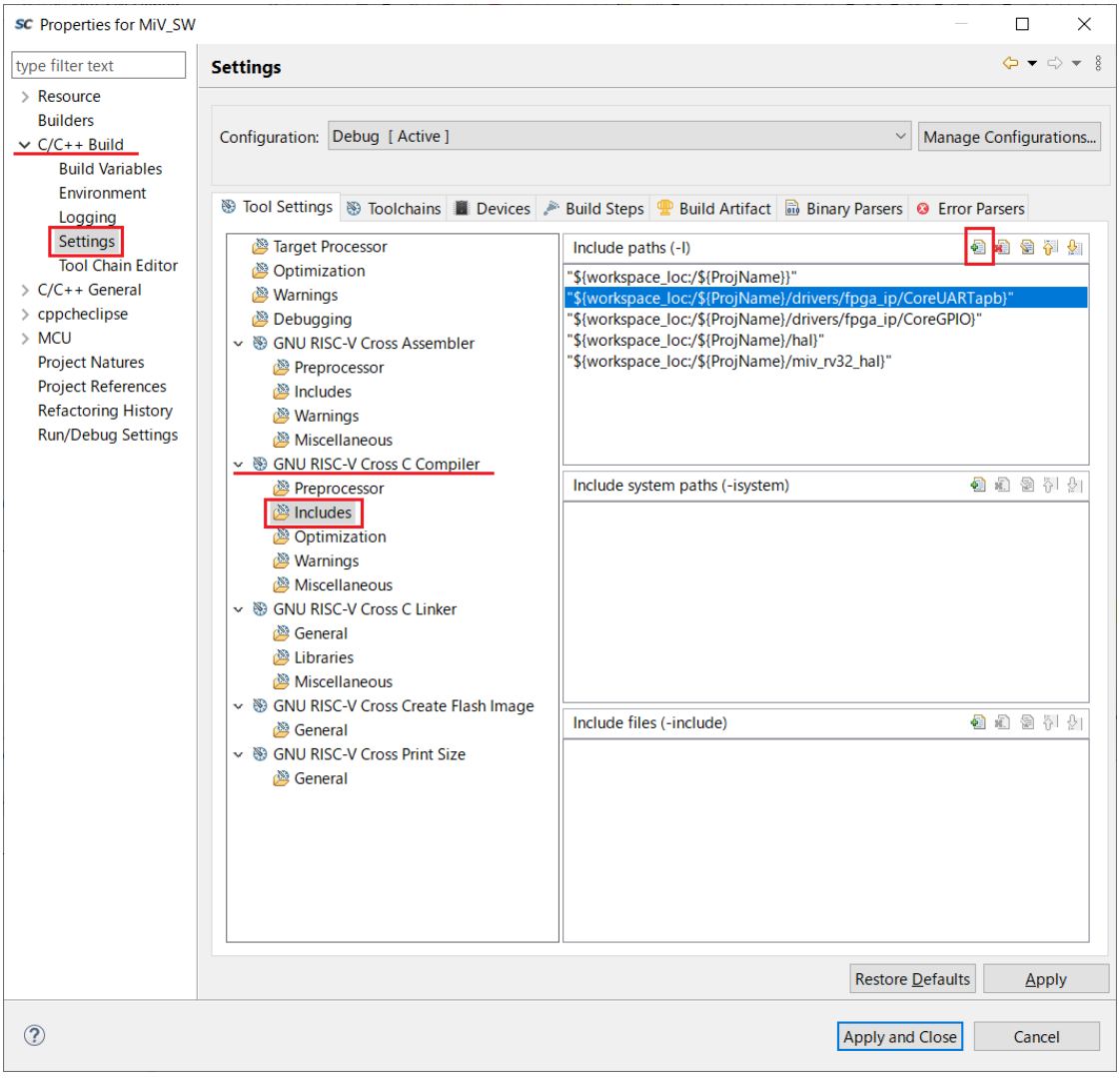


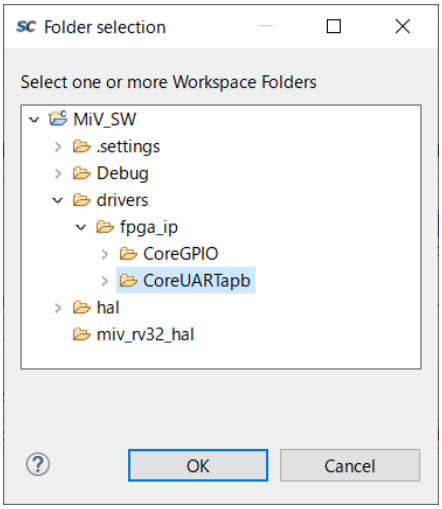


インポート後:



② プロジェクトのプロパティにて、Include Path へ CoreUARTapb を追加します。





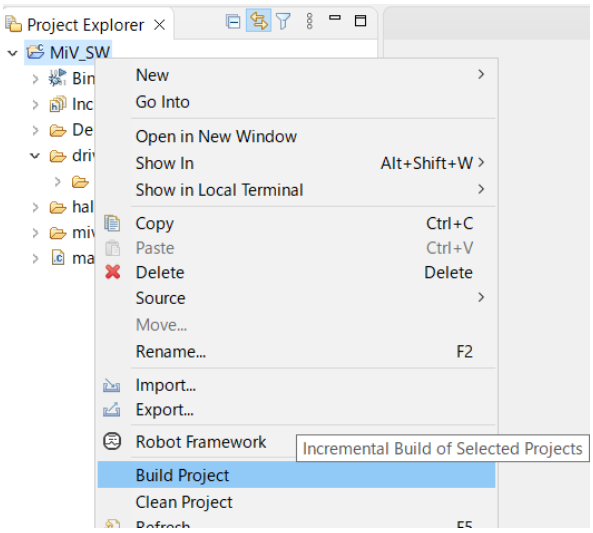
② main.c として下記ソースコードを記載します。

main.c

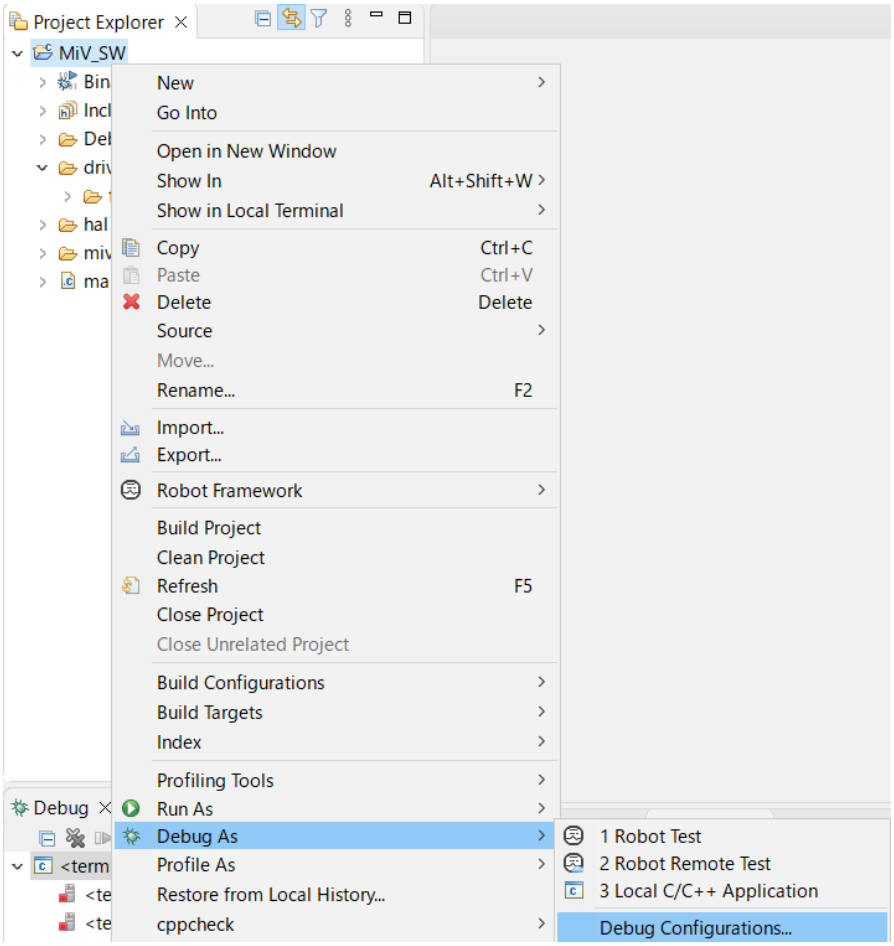
```
/*
 * main.c
 */

#include "drivers/fpga_ip/CoreGPIO/core_gpio.h"
#include "miv_rv32_hal/fpga_design_config.h"
#include "miv_rv32_hal/miv_rv32_hal.h"
#include "drivers/fpga_ip/CoreUARTapb/core_uart_apb.h" /* UARTを使うため */
#include "hal/hw_reg_access.h"

/*******************************************************************************
 * Instruction message. This message will be transmitted over the UART to
 * HyperTerminal when the program starts.
 ******************************************************************************/

uint8_t testmsg[] = {"Hello World!"};

/*-------------------------------------------------------------------------
 * UART instance data.
 */
UART_instance_t g_uart;
```

```c
int main()
{

    /*****************************************************************************
     * Initialize CoreUARTapb with its base address, baud value, and line
     * configuration.
     *****************************************************************************/
    UART_init(&g_uart,
              COREUARTAPB0_BASE_ADDR,
              BAUD_VALUE_115200,
              (DATA_8_BITS | NO_PARITY));

    /*****************************************************************************
     * Send the instructions message.
     *****************************************************************************/
    UART_polled_tx_string(&g_uart, (const uint8_t *)&testmsg);


    return 0;
}
```

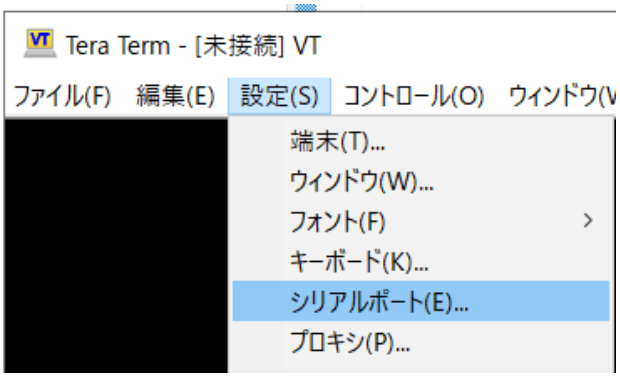
③ プロジェクトをビルドします。

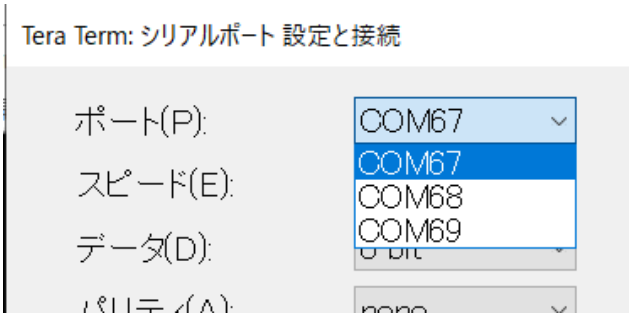


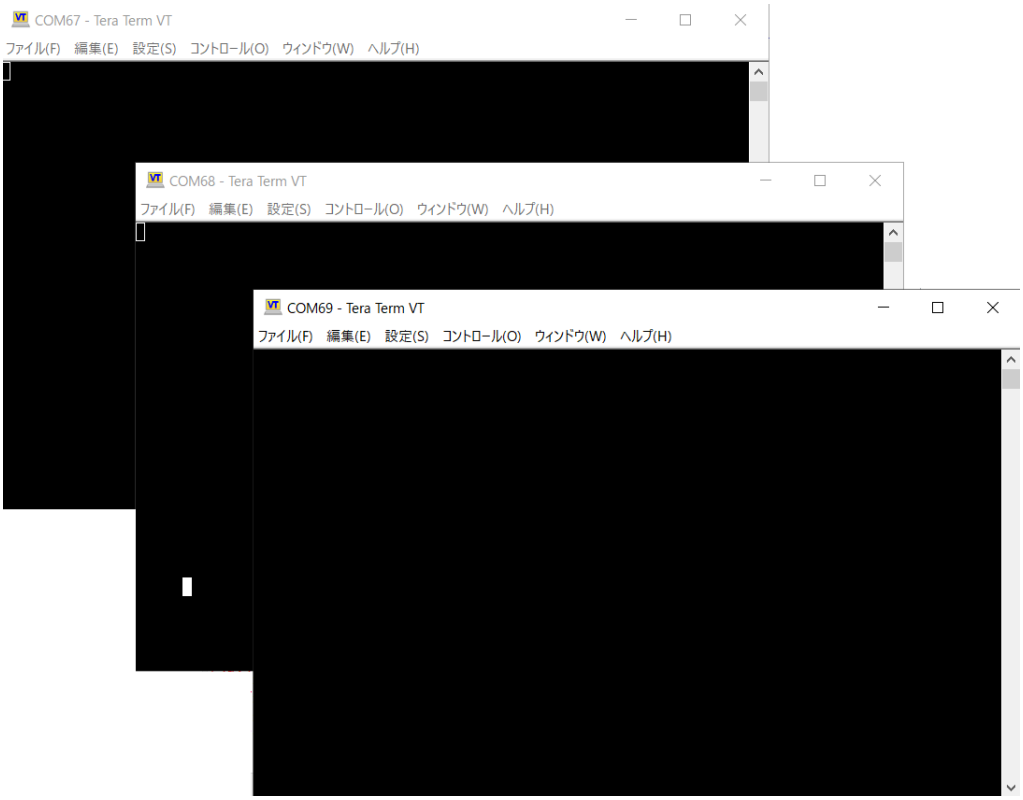
④ Debug Configurations...よりデバッグを開始します。



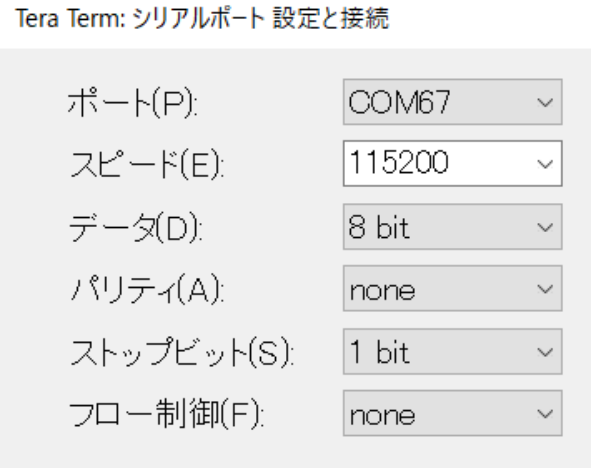
⑤ Tera Term 等、任意の Terminal を開きます。
ポートごとに複数 Terminal を起動します。
※ ポート番号は環境により異なります。



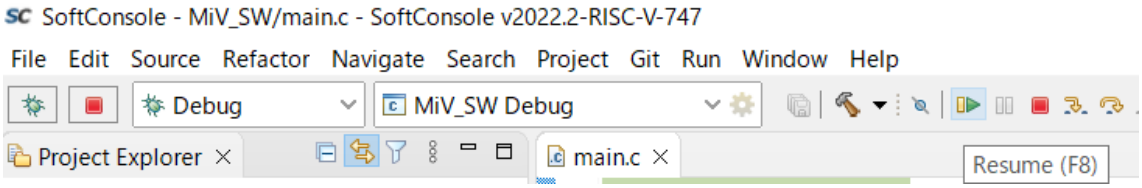




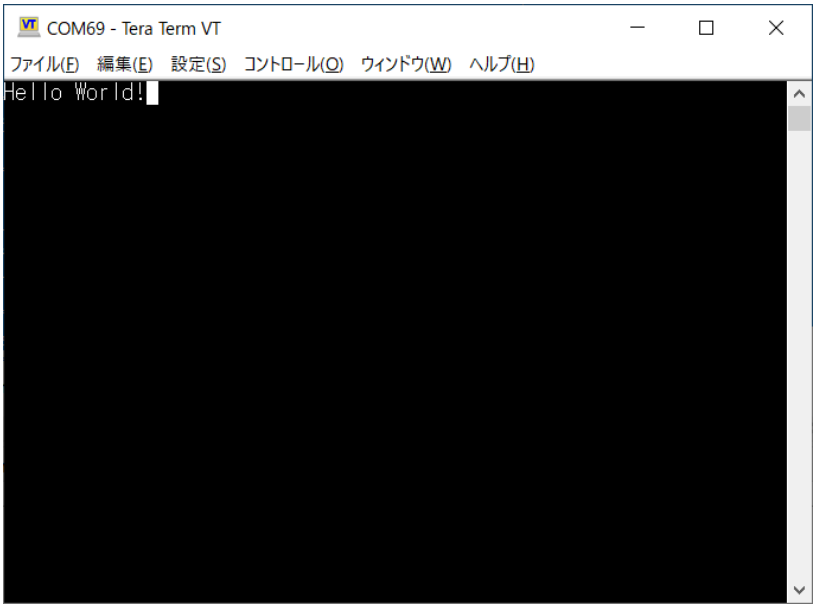
⑥ シリアル通信の設定を確認、適宜変更します。



⑦ Resume ボタンを押して、アプリケーションを実行します。



⑧ Terminal にて Hello World!が表示されることを確認します。



何も表示されない場合、UART の RX、TX のピン配置が逆になっていないか確認します。
表示されるものの文字化けする場合は Terminal のボーレート設定を変えることで、改善されるか確認します。ボーレート変更で改善する場合、CCC などクロック周りの設定を見直してみます。

8. 関連ドキュメント

詳細についてはメーカのドキュメントをご参照下さい。

・PolarFire® SoC Discovery Kit
https://www.microchip.com/en-us/development-tool/mpfs-disco-kit
・AN4997: PolarFire FPGA Building a Mi-V Processor Subsystem Application Note (Earlier TU0775)
https://www.microchip.com/en-us/application-notes/an4997
・GitHub Mi-V Soft RISC-V
https://github.com/Mi-V-Soft-RISC-V
・MIV RV32
https://www.microchip.com/en-us/products/fpgas-and-plds/ip-core-tools/miv-rv32
・MIV_ESS
https://www.microchip.com/en-us/products/fpgas-and-plds/ip-core-tools/miv-ess

以上

改版履歴

リビジョン	日付	概要
V1	2025 年 3 月	新規作成

免責およびご利用上の注意

1. 弊社より資料を入手されましたお客様におかれましては、下記の使用上の注意を一読いただいた上でご使用ください。
2. 本資料は予告なく変更することがあります。
3. 本資料の作成には万全を期していますが、万一ご不明な点や誤り、記載漏れなどお気づきの点がありましたら、本資料を入手されました下記代理店までご一報いただければ幸いです。
   株式会社マクニカ ホームページ https://www.macnica.co.jp/
4. 本資料で取り扱っている回路、技術、プログラムに関して運用した結果の影響については、責任を負いかねますのであらかじめご了承ください。
5. 本資料は製品を利用する際の補助的な資料です。製品をご使用になる際は、各メーカ発行の英語版の資料もあわせてご利用ください。