

Lattice Propel 2024.1

チュートリアル

macnica

Oct. 2024 (Rev.1.1)

免責事項

本ドキュメントの内容はマクニカにて確認した参考情報としての扱いであり、マクニカがその内容を保証するものではありません。また、本ドキュメントの内容は全て現状有姿にて提供され、これに対する改版や技術サポートのご依頼に関しては理由の如何を問わずお控え頂くようお願いしております。お客様ご用途における使用可否の判断、使用の際の動作確認、お客様製品への実装における適合性や安全性の確認、法的要件の確認はお客様にて実施頂きますようお願いいたします。これらに対してもマクニカは一切の責任を負うことが難しく、いかなる保証もいたしかねます。また、本ドキュメントはマクニカの所有物であり、予告なしに変更を加えることがございますので予めご了承ください。

Lattice Propel チュートリアル もくじ

- はじめに p. 4
- RISC-V / Propel 実装手順の概要 p. 5
- Propel Builder の起動、IP のインストール、設定 p. 6
- Propel SDK の起動、初期画面 p. 9
- SoC プロジェクトの作成、確認 p. 11
- ソフトウェア・プロジェクトの作成、ビルド p. 23
- ソフトウェア・プロジェクトの更新 p. 27
- プログラムメモリーの設定 p. 28
- Radiant フィッティング p. 31
- SDK デバッグ設定、デバッグ p. 33
- 補足 p. 44
 - 1. 無償ライセンス・ファイルの生成・入手
 - 2. What's New in Propel 2024.1
 - 3. ECO Editor
 - 4. GPIO コンポーネントの多ビットポート
 - 5. ユーザーモジュールの組み込み
 - 6. 内部バス引き出しでユーザー回路とインターフェイス
 - 7. SoC/C プロジェクトのインポート
 - 8. SoC プラットフォームのインポート
 - 9. C/C++ プロジェクトのインポート
 - 10. SDK ビルド設定
 - 11. Propel 起動後のワークスペース切り替え

1. はじめに

- **Lattice Propel** での作業にはライセンス・ファイルが必要です (P.45 を参照)
 - **Propel SDK** はライセンスが無くても起動しますが、**Propel Builder** は起動しません
 - インストール後、“license” フォルダに入手したライセンス・ファイルを保存します
 - Windows 環境変数 “LM_LICENSE_FILE” が適切に設定されていることを確認します
- 本 RISC-V チュートリアルは Windows 環境で主に “Hello World” テンプレート・プロジェクトを例として実装する手順の紹介です
 - **MachXO2/3** ターゲットの場合は『Propel 2022.1 Tutorial』をご参照ください
- RISC-V の実装には **Lattice Propel** および **Radiant** を使用します
 - **Nexus** シリーズ、**Avant** ファミリーが対象の場合 2024.1 を予めインストールしておきます
 - **Radiant** の無償版で設計できます (SERDES 搭載品は有償版のみ)
 - **MachXO2/3** ターゲットの場合は **Radiant** ではなく **Diamond** を使用します
- **Lattice Propel** をインストール後 Eclipse IDE ベースの統合フレームワークで開発作業を行います
 - **Propel SDK** C/C++ プロジェクトの作成とビルド、およびデバッグ (“Debug Perspective”)
 - **Propel Builder** FPGA に構成する RISC-V を含む “SoC プラットフォーム” の設計と生成 (SoC プロジェクト)

RISC-V / Propel 実装手順の概要

実装ステップは以下の通りです：

(事前準備) **Propel Builder** を起動し、各 IP モジュールをインストールしておきます (p.7)

✓ 一度更新しておけば、それ以降は必要に応じてインストールするのみです

1. **Lattice Propel** を起動します

✓ 作業フォルダ (workspace) を事前に作成しておきます

2. SoC プラットフォーム (FPGA 実装回路) を作成します

✓ **Propel Builder** で "SoC プロジェクト" を作成し、プラットフォーム (回路設計と構成) を生成します

➢ ステップ 3 で必要な "sys_env.xml" などが生成されますので、必ず SoC 作成が先になります

3. C/C++ プログラム (ユーザ・アプリケーション) を作成します

3-1. **Propel SDK** でソフトウェア・プロジェクトを作成します

➢ C/C++ ソースコードは別途作成してインポートするか、この環境で作成・編集します

3-2. C/C++ ソースコード一式をビルドします

➢ アプリケーション実行ファイル *.elf と、これを FPGA 用に変換した *.mem ファイルが生成されます

4. **Propel Builder** に戻り SoC プラットフォームを更新します

✓ ビルドで生成した *.mem ファイルを RISC-V 用プログラムメモリー systemem の初期化に指定します

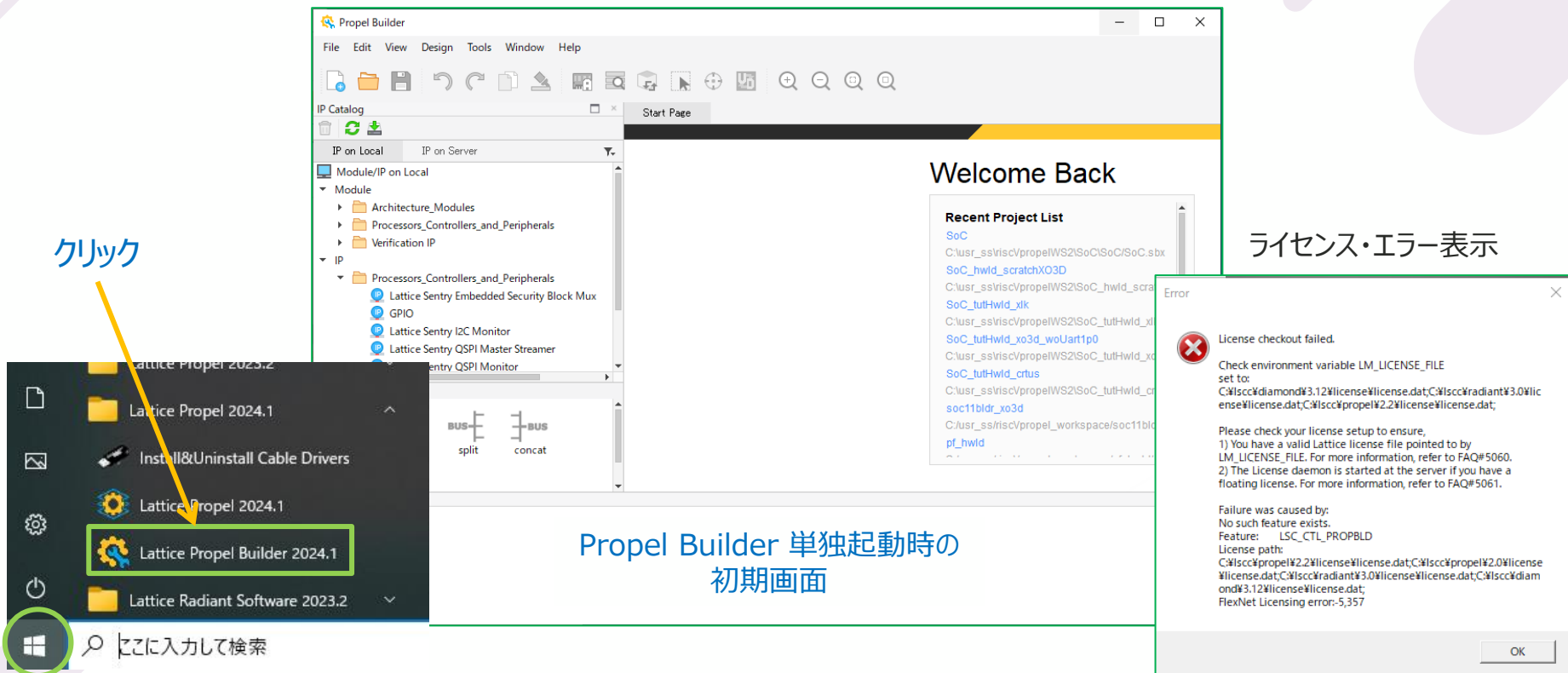
➢ systemem コンポーネント初期化指定しないでデバッグ作業を繰り返す方法もあります (p.30 参照)

5. (**Propel** から) **Radiant** を起動してフィッティングします

6. プログラマーでデバイスをプログラミングした後に動作確認 (HW/SW デバッグ) します


Propel Builder の起動

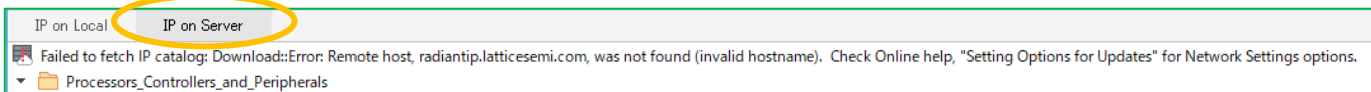
- Windows スタートメニューで Lattice Propel Builder 2024.1 をクリックして起動します
 - ✓ ライセンス・ファイルが見つからないと“License checkout failed” のメッセージが出ます (p.45 参照)





Propel Builder : IP のインストール

■ “IP on Server” タブを選択し、各コンポーネントをインストールします

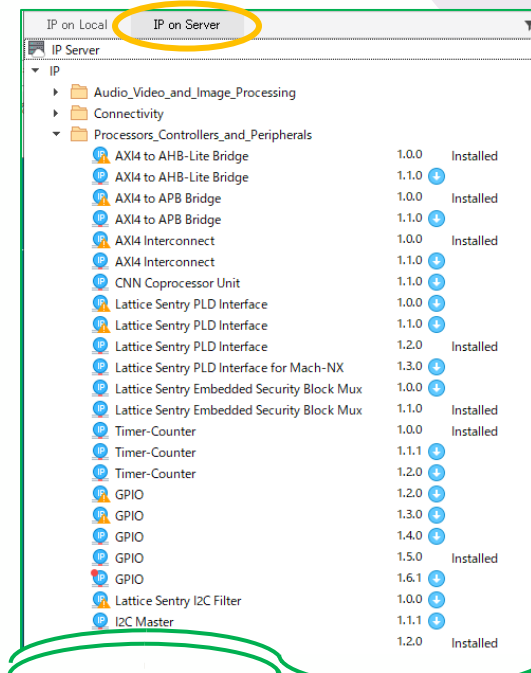
- まず  アイコンをクリックして Lattice Server にアクセスできるかどうかをチェックします
- アクセスできないと下のようなメッセージが表示されますので、次ページのネットワーク設定を確認・変更します



- ✓ 問題なければ右図のように入手可能なコンポーネントとそのバージョン、インストール済かどうかが表示されます
- ✓ インストールするコンポーネントを選択し、  アイコンのどちらかをクリックします
- 👉 “License Agreement” を求める表示が出ますので、“Accept” して作業を進めます
- 👉 RISC-V コアは SM と MC、および RX の三つがありますが、全てインストールしておきます
- 👉 各コンポーネントは最新バージョンのインストールを推奨します（古いバージョンが必要な場合にのみインストールします）

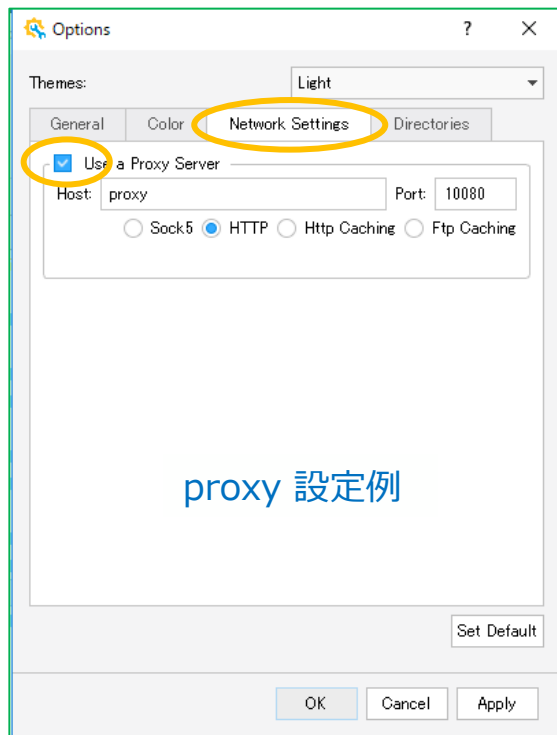
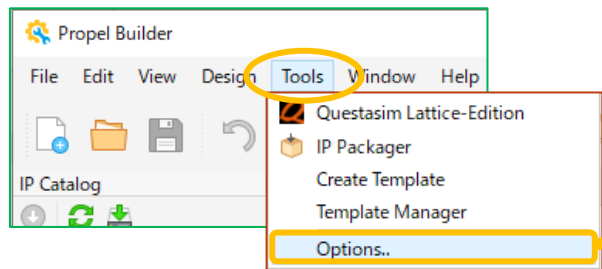
■ インストールが完了したら、いったん Propel Builder は終了します

- ✓ その後は IP が更新され次第、単独起動することなく、逐次インストールできます
- ✓ Propel Builder で作成した SoC プロジェクト（プラットフォーム）は、GUI 単独起動ではなくても、SDK からオープンできます（p.22、p.28、p.31）



Propel Builder : ネットワーク設定

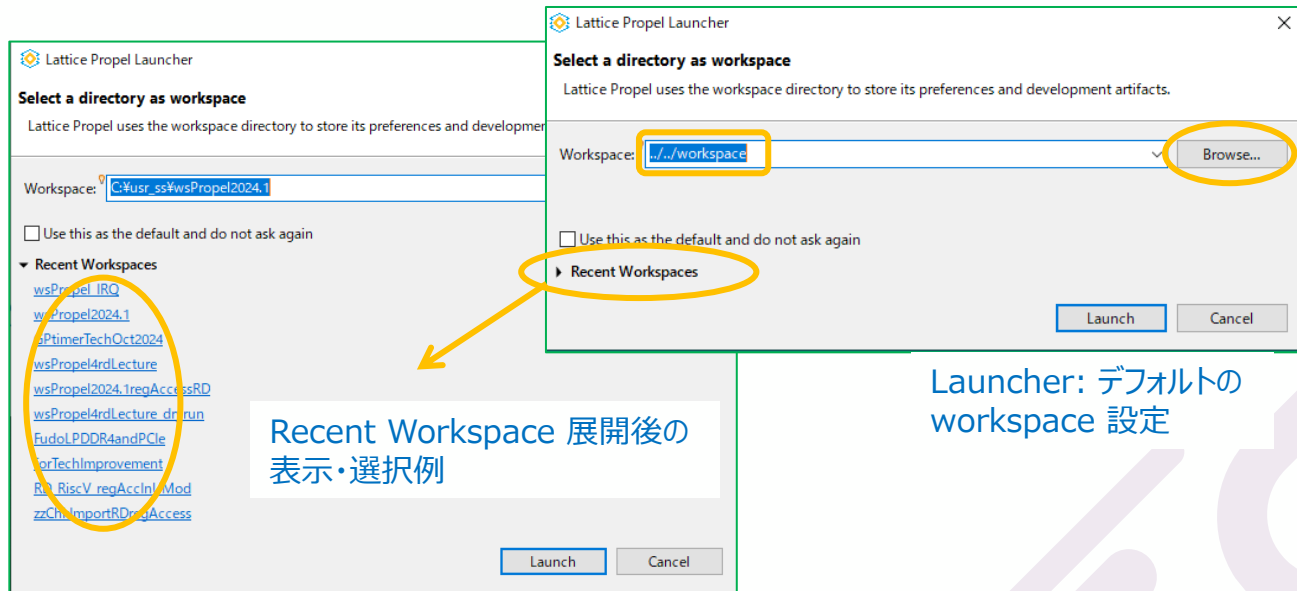
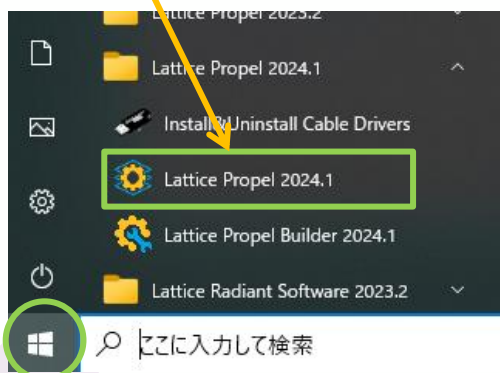
- Lattice Server アクセスに問題がある場合、ネットワーク設定を確認します
 - メニューバーの Tools → Options.. を選択します
 - 社内ネットワーク接続環境などでプロキシ・サーバーを介する場合は“Use a Proxy Server” にチェックしてホスト名、ポート名などを適切に設定します
 - Apply → OK をクリックします



Propel SDK の起動

- Propel の起動は Windows スタートメニューで Lattice Propel 2024.1 をクリックします
 - ✓ Launcher 窓が立ち上がり workspace を訊いてきます
 - ✓ デフォルトのディレクトリは後々の作業でやや煩雑なので、**予め専用の作業ディレクトリを作成しておき、ブラウズして指定先にします**
 - ☞ 以降ステップの SoC プロジェクト、C/C++ プロジェクトのフォルダは全てこのディレクトリ下に生成されます
 - ☞ 開発案件ごとに作成して使い分けるなど、プロジェクト管理が容易になります
 - ☞ 『Browse...』 ボタンで指定、または下部 “Recent Workspaces” を展開し作成済フォルダをクリックすることで起動時の指定を変更できます
 - ☞ P.62 に関連記述

クリック

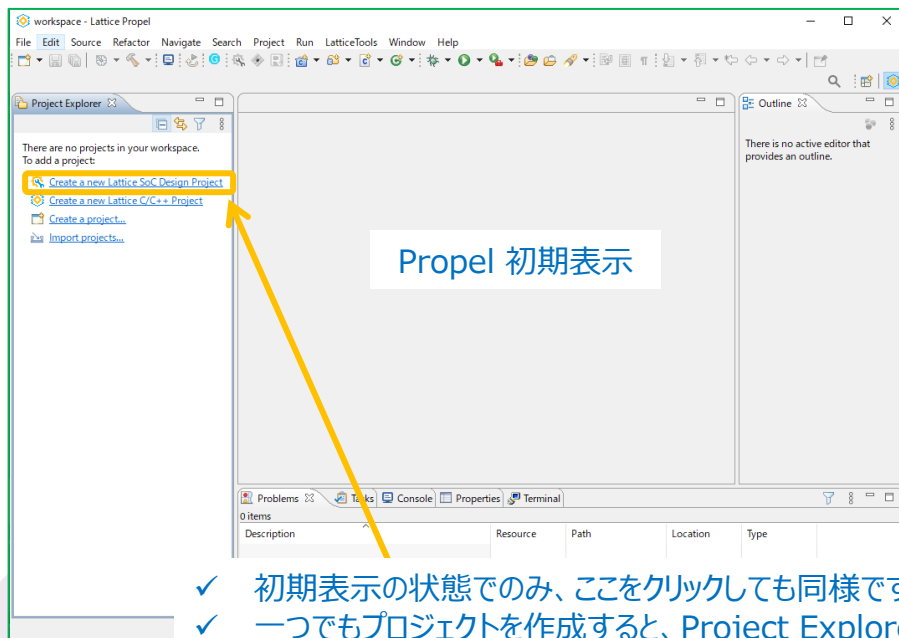


Launcher: デフォルトの workspace 設定

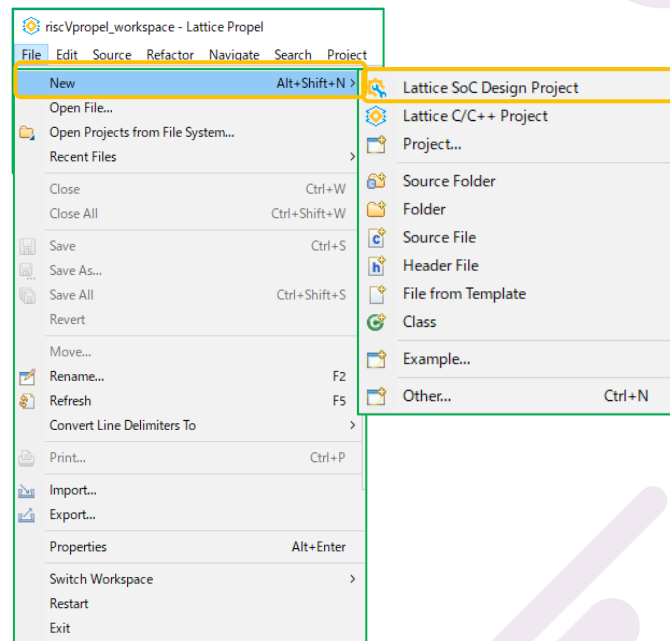
Recent Workspace 展開後の表示・選択例

Propel SDK の初期画面

- Propel SDK が起動したら（下）、File → New → Lattice SoC Design Project を選択して SoC ハードウェア・プラットフォームを作成します（右）
 - ☞ p.5 に明記した通り SoC 作成後の生成ファイル“sys_env.xml”がないと、C/C++ プロジェクトは作成できません
 - ☞ Propel Builder GUI を起動して作成しても、Project Explorer に表示（登録）されませんので、関連する C/C++ プロジェクトと連携したプロジェクト管理ができません。必ず SDK から作成するようにします（別途インポートする場合は pp.54-57）



- ✓ 初期表示の状態でのみ、ここをクリックしても同様です
- ✓ 一つでもプロジェクトを作成すると、Project Explorer 枠内の表示は変わります



SoC プロジェクトの作成 (1)

- “Create SoC project” が表示されますので、プロジェクト名を入力し、その他を指定します
 - ✓ “Template Design” はチュートリアルでは Hello World Project を選択します (Finish で終了)
 - 👉 “Project name” は SoC プロジェクトであること、およびターゲットデバイスが容易に識別できる名称にします
 - 👉 “Project name” 入力と同じ名称のフォルダが workspace (p.9) の下に生成されます
 - 👉 RISC-V RX は RTOS 用です

SoC プロジェクト名を入力

言語を選択

Verilog
VHDL

RISC-V コアを選択

RISC-V MC
RISC-V RX
RISC-V SM

(Family 選択後に指定)

チュートリアルは
テンプレートから選択
(RISC-V タイプで異なる)

Project name: SoC1k1nxChkAHBLFT

Use default location:

Location: C:\usr_ss\wsp\Propel2024.1\SoC1k1nxChkAHBLFT

Choose file system: default

Language: Verilog

Board:

Device Select

Processor: RISC-V MC Family: LIFCL (CrossLink-NX)

Device: LIFCL-40 Speed: 8_High-Performance_1.0V

Package: CSBGA289 Condition: Commercial

Template Design

Empty Project

RISC-V MC SoC Project

Components included:

- a) Processor - RISC-V MC w/ PIC/TIMER
- b) GPIO
- c) ASRAM - Asynchronous SRAM
- d) UART - Serial port
- e) PLL
- f) Clock

重要 : Device Select 部各指定は、この後に変更することができません。特に Family は生成 RTL に作用しますので注意します。Device / Package / Speed の設定は厳密でなくても構いません (p.19 参照)

Family

LAV-AT (Avant)
LFCPNX (CertusPro-NX)
LFD2NX (Certus-NX)
LFMX05 (MachXO5-NX)
LIFCL (CrossLink-NX)
ECP5U
ECP5UM
ECP5UM5G
LFMNX
MachXO2
MachXO3D
MachXO3L
MachXO3LF

Speed Grade、温度グレード

テンプレート・プロジェクトの内容

SoC プロジェクトの作成 (2)

- Project Explorer 枠に生成された SoC プロジェクトとファイルが表示されます
- 同時に Propel Builder が立ち上がり、テンプレートで選択したプロジェクトの回路が表示されます

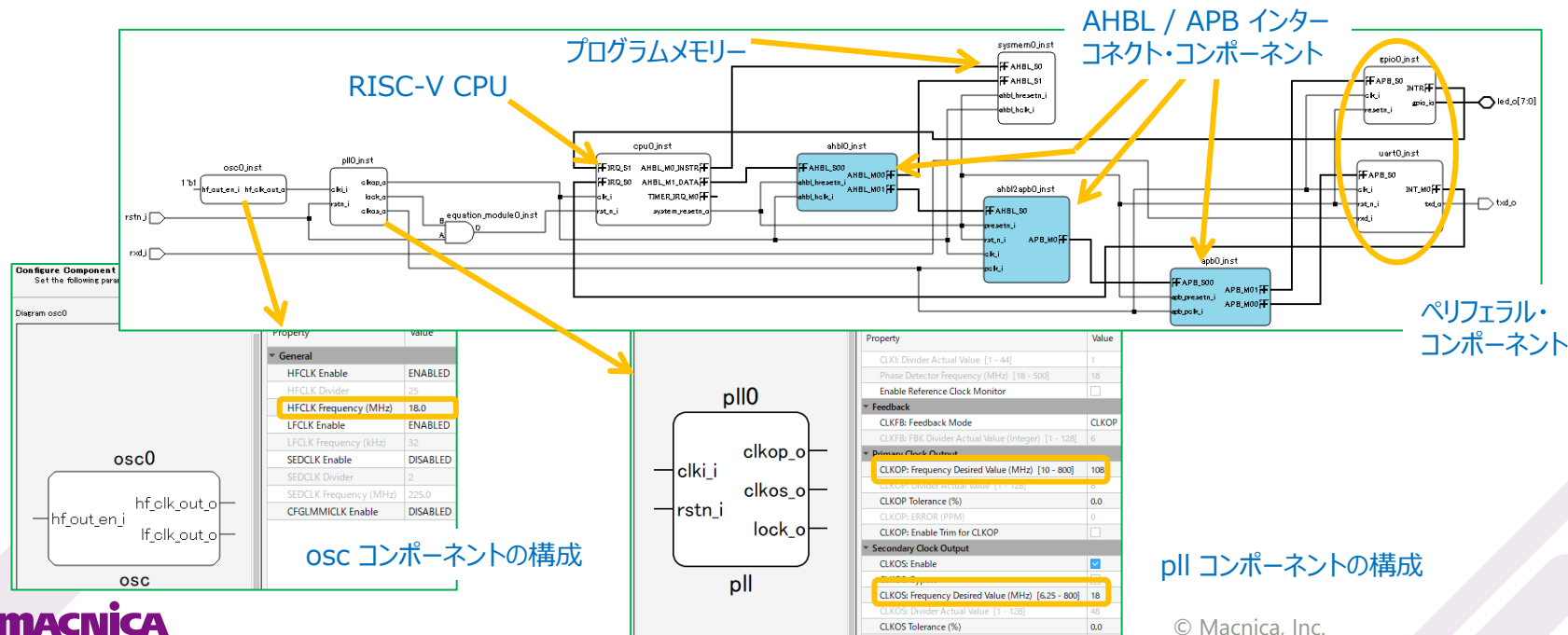
The screenshot displays the Propel Builder application window. On the left, the IP Catalog is open, showing a list of modules and IP blocks. A yellow circle highlights the 'Timer-Counter' module, and a yellow arrow points from it to the schematic diagram. The schematic diagram shows a complex circuit with various IP blocks connected together. A text box in the upper right of the schematic area reads 'Propel Builder 起動後のテンプレート・デザインの表示例'. At the bottom left, the 'IP Catalog' tab is circled in red, with a text box below it stating: '[IP Catalog] タブでは PC にインストール済みで、かつ選択デバイスで使用できるコアとペリフェラル名が表示されます'.

Propel Builder 起動後のテンプレート・デザインの表示例

[IP Catalog] タブでは PC にインストール済みで、かつ選択デバイスで使用できるコアとペリフェラル名が表示されます

SoC プロジェクトの作成 (3)

- テンプレートで選択した Crosslink-NX 用 Hello World の回路図トップが表示されます
 - ✓ RISC-V コアは、ここではインスタンス “cpu0_inst” です
 - ✓ “osc” と “pll” コンポーネントがクロックソースとしてインスタンスされています
 - ✓ 実行プログラムを格納するオンチップメモリ・コンポーネントは “system0_inst” です
 - ✓ ペリフェラル・インスタンスとして GPIO (gpio0_inst)、UART (uart0_inst) が APB で接続されています
 - 通常どのようなハードウェアにも存在するペリフェラルです。割り込み信号が直接コアに与えられます

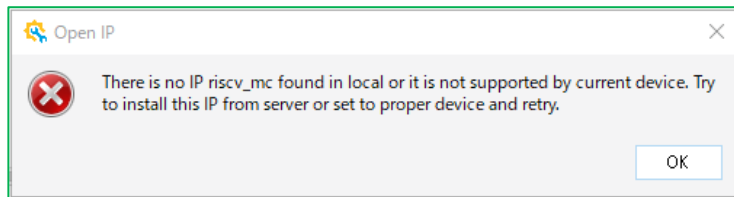


SoC プロジェクトの作成 (4-1)

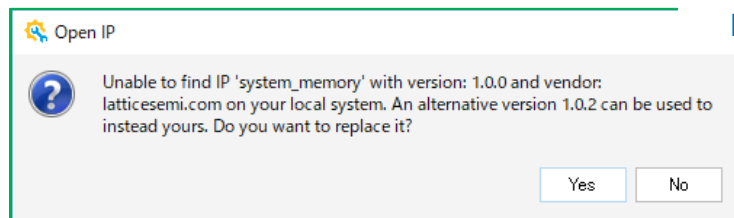
■ 各コンポーネントのバージョン確認・更新

- ✓ テンプレートや既存の Schematic が表示される際に、含まれるコンポーネント (IP) がインストールされていないと左下メッセージ 1 のような表示が出る場合があります
 - Propel Builder が立ち上がった状態で “IP Catalog” タブを選択し、当該 IP をインストールします (p.7)
- ✓ Schematic 内の各コンポーネントをダブルクリックして、左下メッセージ 2 のような表示が出ないことを確認します
 - “replace” の必要がなければ pp.15~17 のような “Configure IP” ウィンドウが立ち上がりますので、“Generate” をクリックします
 - メッセージ 2 の場合に “replace” に問題なければ “Yes” をクリックして先に進みます

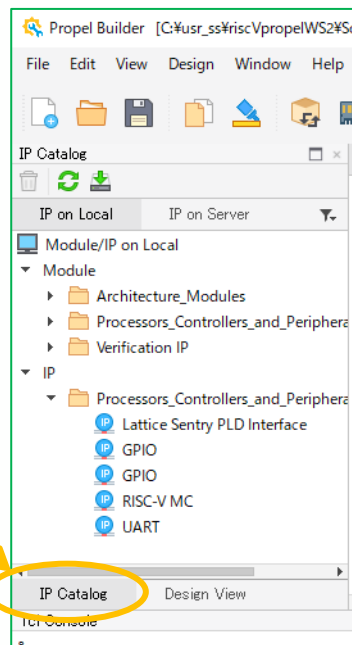
“Open IP” メッセージ 1



“Open IP” メッセージ 2

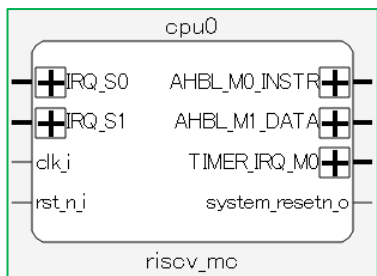


“IP Catalog” タブを選択後
p.7 のようにインストール



SoC プロジェクトの作成 (4-2)

- 各インスタンスをダブルクリックするとその構成 (Configuration) ができます
 - ✓ pp.15-17 の図・バージョンは Propel 2024.1 を元にしています
 - ✓ 下図は RISC-V MC ですが、構成を変更したら必ず Generate をクリックします (他も同様)
 - ✓ P.14 に示した通りバージョンが古いとメッセージが出ますので更新します



RISC-V MC/SM
のシンボル例

RISC-V MC 2.6.0
のデフォルト Property

Property	Value
CFU Configuration	
Enable CFU Ports	<input type="checkbox"/>
General	
Debug Enable	<input checked="" type="checkbox"/>
Soft JTAG	<input type="checkbox"/>
Cache Enable	<input type="checkbox"/>
Instruction Cache Enable	<input type="checkbox"/>
Data Cache Enable	<input type="checkbox"/>
Instruction Cache Cacheable Address Range Lower Limit (32'h00000000 ~ 32'hFFFFFFFF)	32'hFFFFFFFF
Instruction Cache Cacheable Address Range Higher Limit (32'h00000000 ~ 32'hFFFFFFFF)	32'h00000000
Data Cache Cacheable Address Range Lower Limit (32'h00000000 ~ 32'hFFFFFFFF)	32'hFFFFFFFF
Data Cache Cacheable Address Range Higher Limit (32'h00000000 ~ 32'hFFFFFFFF)	32'h00000000
C Extension for Compressed Instructions	<input checked="" type="checkbox"/>
M Extension for Integer Mult and Div	<input checked="" type="checkbox"/>
CFU Configuration	
Number of CFU [1 - 2]	1
General	
PIC Enable	<input checked="" type="checkbox"/>
Timer Enable	<input checked="" type="checkbox"/>
PIC and Timer Base Address (32'h00000000 ~ 32'hFFFFFFFF)	32'hFFF0000
Number of Interrupt Requests [2 - 8]	2
JTAG Channel Selection for Certain Devices [14 - 16]	14

SoC プロジェクトの作成 (4-3)

- systemem と AHBL/APB インターコネクト・コンポーネントです

Configure Component from Module ahb_lite_interconnect Version 1.3.1
Set the following parameters to configure this component.

Diagram ahbl0

Configure IP

Property	Value
General	
Total AHB-Lite Managers [1 - 32]	1
Total AHB-Lite Subordinates [1 - 32]	2
Manager Address Width(bits)	32
Full Address Decoding up to 1kB	<input checked="" type="checkbox"/>
Data Bus Width(bits)	32
Registered Output	<input type="checkbox"/>
Manager 0 Connection Setting	
Manager 0 Subordinate 0 Connect Enable	<input checked="" type="checkbox"/>
Manager 0 Subordinate 1 Connect Enable	<input checked="" type="checkbox"/>

Configure Component from Module ahb_lite_to_apb_bridge Version 1.1.1
Set the following parameters to configure this component.

Diagram ahbl2apb0

Configure IP

Property	Value
General	
Address Width(bits)	32
Data Bus Width(bits)	32
APB Clock Enable	<input checked="" type="checkbox"/>

Configure Component from Module apb_interconnect Version 1.2.1
Set the following parameters to configure this component.

Diagram apb0

Configure IP

Property	Value
General	
Total APB Requestors [1 - 32]	1
Total APB Completers [1 - 32]	2
Requestor Address Width(bits)	32
Full Address Decoding up to 1kB	<input checked="" type="checkbox"/>
Data Bus Width(bits)	32
Registered Output	<input type="checkbox"/>

プログラムサイズによって変更します
(デバイスに搭載されている EBR 数が許容限度です。
8kW (8,192) で EBR の場合 16 個を消費します)

Configure Component from Module system_memory Version 2.2.0
Set the following parameters to configure this component.

Diagram systemem0

Configure IP

Property	Value
General	
Interface	AHBL
Memory Address Depth [1 - 43008]	8192
Data Bus Width(bits)	32
Memory Type	EBR
Port Count	2
ECC Enable	<input type="checkbox"/>
Enable Arbiter	<input type="checkbox"/>
Data Streamer	
Enable Data Streamer	<input type="checkbox"/>
Data Streamer Interface	Generic FIFO
Data Streamer Clock Bypass	<input type="checkbox"/>
Data Streamer Write Start Address [0 - 8191]	0
Initialization	
Initialize Memory	<input type="checkbox"/>
Initialization File Format	hex
Initialization File	none

pp.29-30、p.47 に関連説明

SoC プロジェクトの作成 (4-4)

■ GPIO と UART コンポーネントです

Configure Component from IP gpio Version 1.6.2
Set the following parameters to configure this component.

Diagram gpio0

Configure IP

Property	Value
General	
Number of I/O Lines [1 - 32]	8
Remove Tri-State Buffer	<input type="checkbox"/>
Initial Output Value (hex) [0 - FFFFFFFF]	0
Initial Output Binary Value	0000 0000
IO Direction (hex) [0 - FFFFFFFF]	FF
IO Direction Encoded Value	0000 0000
Interface	APB

I/O ポートの本数

双方向バッファをなしにするオプション

出力の初期値 (各ビットごとに設定)

方向 : 1=出力、0=入力 (各ビットごとに設定)

Configure Component from IP uart Version 1.3.0
Set the following parameters to configure this component.

Diagram uart0

Configure IP

Property	Value
General	
Enable APB	<input checked="" type="checkbox"/>
System Clock Frequency (MHz) [2 - 200]	18
Serial Data Width	8
Stop Bits	1
Parity Enable	<input type="checkbox"/>
ODD Parity	<input type="checkbox"/>
Enable Stick Parity	<input type="checkbox"/>
Baud Rate	
Baud Rate Type	Standard
UART Standard Baud Rate	115200
UART Custom Baud Rate [2400 - 1000000]	115200
UART Feature Enables	
FIFO Enable	<input type="checkbox"/>
MODEM Enable	<input type="checkbox"/>
Rx Ready Enable	<input type="checkbox"/>
Tx Ready Enable	<input type="checkbox"/>


ボーレートの精度が +/-1% 以内* になる除数 (整数) が存在し、かつ PCB / Device が生成できる周波数にします

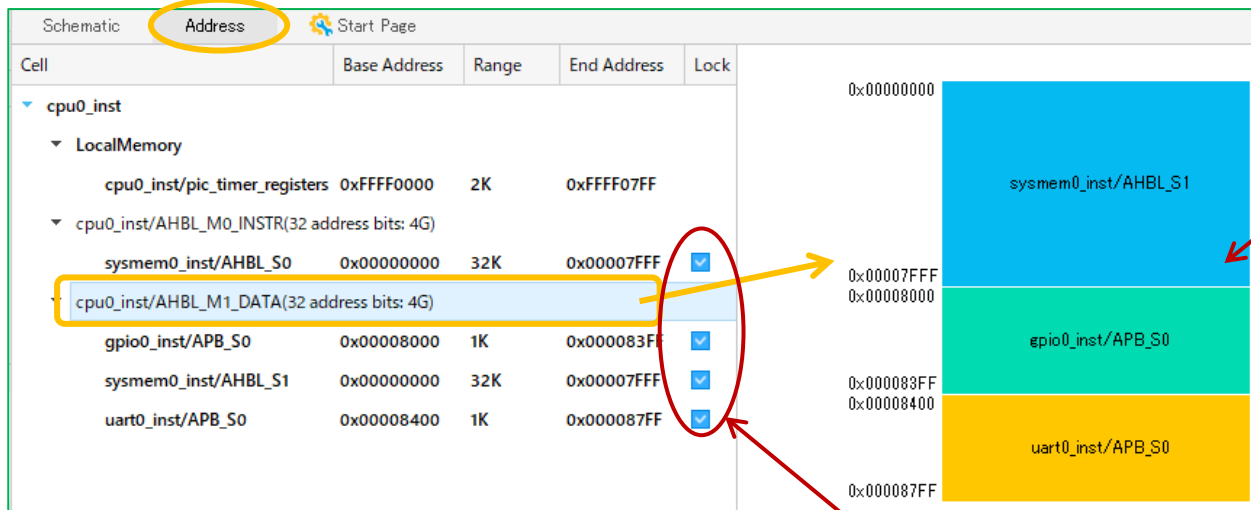
(* : 一般的には <1.5~2% 程度のデバイスが多い)

SoC プロジェクトの作成 (4-5)

■ Propel Builder の Address タブでアドレスマップが確認できます

✓ 下図はデータ（ペリフェラル）バス選択時の表示例です

- ① systemem など、アドレスを固定したいコンポーネントの "Base Address" を入力後に "Lock" をチェックし、 をクリックして自動アサインする、または
- ② 各コンポーネントの "Base Address" を適宜編集します (Lock をチェックしておきます)



Cell	Base Address	Range	End Address	Lock
cpu0_inst				
LocalMemory				
cpu0_inst/pic_timer_registers	0xFFFF0000	2K	0xFFFF07FF	
cpu0_inst/AHBL_M0_INSTR(32 address bits: 4G)				
systemem0_inst/AHBL_S0	0x00000000	32K	0x00007FFF	<input checked="" type="checkbox"/>
cpu0_inst/AHBL_M1_DATA(32 address bits: 4G)				
gpio0_inst/APB_S0	0x00008000	1K	0x000083FF	<input checked="" type="checkbox"/>
systemem0_inst/AHBL_S1	0x00000000	32K	0x00007FFF	<input checked="" type="checkbox"/>
uart0_inst/APB_S0	0x00008400	1K	0x000087FF	<input checked="" type="checkbox"/>

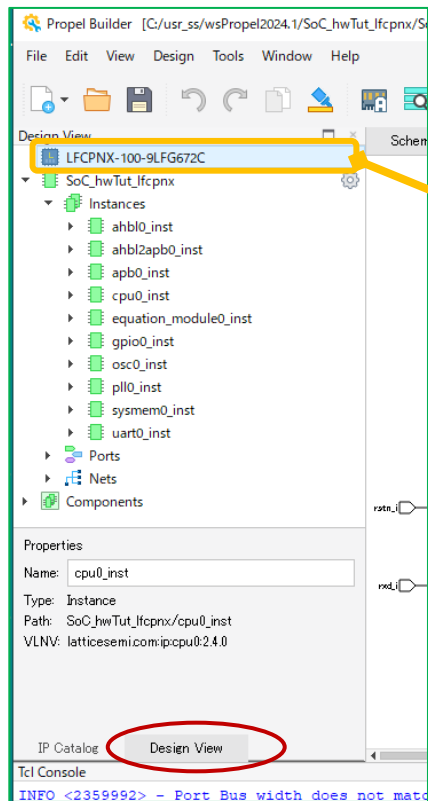
アドレス範囲がオーバーラップしていると赤色表示になりますの (DRC エラー) で、問題の有無が容易に判別できます

データバスのメモリマップ表示例

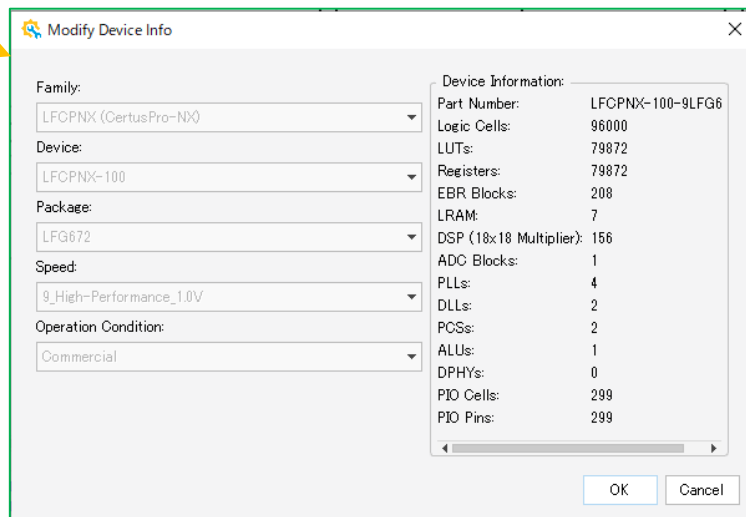
アドレスを固定 (ロック) する際にチェックします

SoC プロジェクトの作成 (4-6)

- Propel Builder の Design View タブから、ターゲット・デバイスの属性が確認できます
 - ☞ Propel 2024.1 からは Family のみでなく、Device/Package/Speed/Operating... も全て変更できなくなりました



(CertusPro-NX の場合の例)






全てグレイアウトになり、変更ができません

SoC プロジェクトの作成 (5)

■ 新規にコンポーネントやポートを追加した場合など、それらを結線する方法について記述します

✓ ワイヤ (1bit 幅の信号やポート) : 下

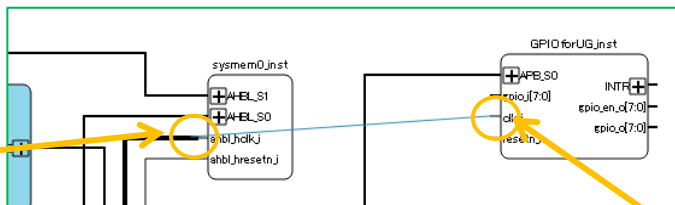
- マウスをピン上に移動して  アイコンが表示されたらクリックし、クリックした状態で、結線先のピンにドラッグ (移動) します
- 結線可能だと信号線がボードに変化し、また緑色のマーク  が現れますのでクリックします。結線されると赤色になります
- 連続してこの状態で他にも結線できるように  アイコンと配線が表示されたままになっていますので、全て結線するまで続けます
- 結線する作業を終了する場合は、この状態で何もいないところでマウス右クリックするか『Esc』キーをヒットします

✓ バス信号の接続 : 右

- ワイヤの a ~ b は同じ操作です
- 結線されると、配線が赤色になり、これで結線操作は終了します

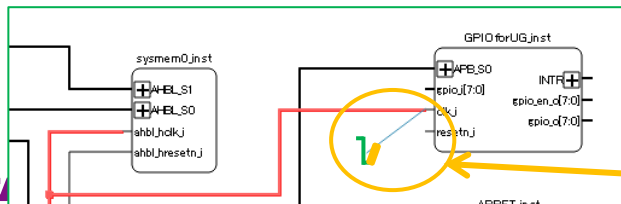
a & b. ピンでクリックしたままドラッグして再クリックすると
信号線がボードになり、緑のマークが出る

有効な結線先
(クリック前)



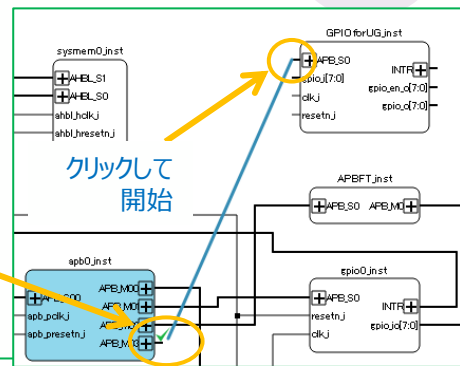
クリックして開始

c. クリックが有効だと赤色になります。アイコンと配線のマーク
は表示されたままで、連続して他と結線できる



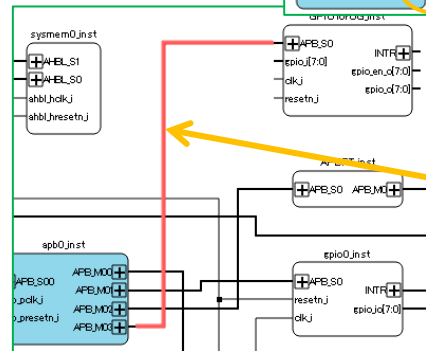
結線後も表示された
まま (Esc で終了)

1. バスの場合、始めはワイヤ
の a & b と操作は同じ



クリックして
開始

有効な結線先には
緑マークが表示される



2. 結線先でクリックする
と赤色になり、これで
自動終了

SoC プロジェクトの作成 (6)

- 各コンポーネントの配置と接続、構成 (パラメータ)、およびアドレス配置を完了します

- ✓ アドレス配置は "Auto Assign" しても良いです

- メニューバーかアイコンで次を行います

1. DRC (Validate Design) ← 必須です!

2. Generate

- Tcl Console はエラーがなければ右下のようになります

- ✓ 保存後 Propel SDK GUI に戻ります (次ページ)

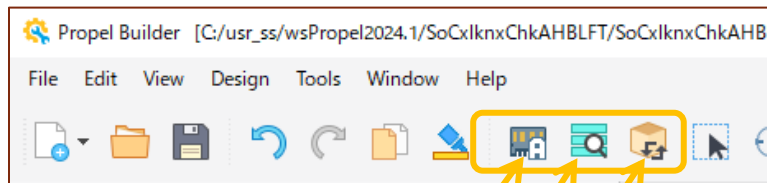
- ☞ SoC の属性を全て含む二つのファイル "sys_platform.h" と "sys_env.xml"、
および各コンポーネントのドライバーが C/C++ プロジェクトにコピーされます

- ✓ Warning は極力解消しておくようにします

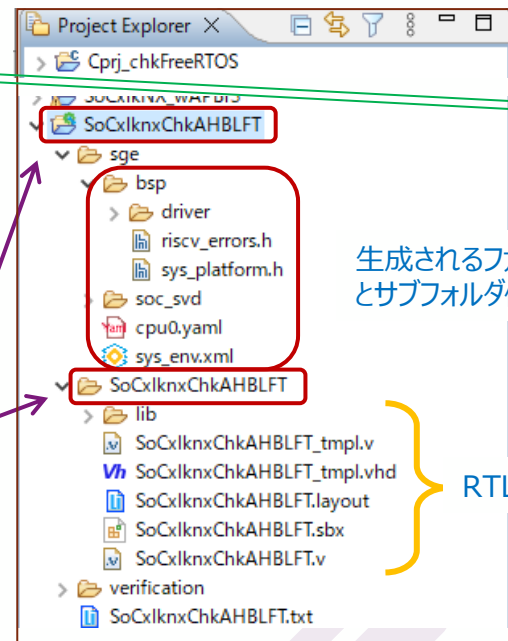
- ☞ 本例 "cpu0_inst_TIMER_IRQ_M0 remains unconnected..." はそのまま残っていても Generate できます

"Generate" 後の
コンソール表示例

```
Tcl Console
INFO - Start: sbp_design generate.
INFO - Finished: sbp_design generate.
% sbp_design save
% sbp_design pge sge -i {C:\usr_ss\wsPropel2022.1\SoC2022pi_LIFCLtut\SoC
Warning[SoCDesign]: cpu0_inst_TIMER_IRQ_M0 remains unconnected
%
```



(Address) Auto Assign
"Validate" (DRC)
Generate




生成されるファイル
とサブフォルダ例

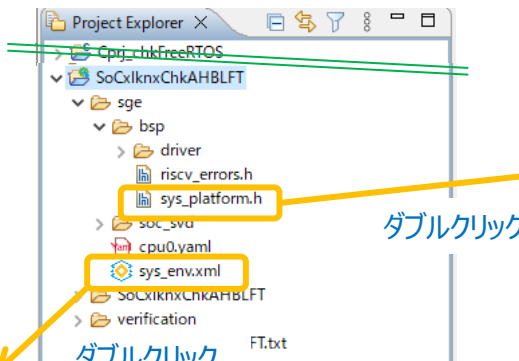
SoC プロジェクト名

"Generate" 後の Project
Explorer ウィンドウ表示例

SoC プロジェクトの確認

- **Propel Builder** で問題なく **Generate** が実行されると “sge” サブフォルダと各ファイルが生成されています
 - ✓ Project Explorer 枠で確認できます
 - ✓ ダブルクリックして、各ファイルの内容を確認することができます

SoC プロジェクトを選択してメニューバー下部のアイコン列から  をクリックすれば **Propel Builder** が起動し、回路図が表示されます
(右クリックメニュー **Open Design In** → **Propel Builder** でも同様)



```
sys_platform.h X
52 #ifndef SYS_PLATFORM_H
53 #define SYS_PLATFORM_H
54
55 /* cpu mode setting */
56 #define INST_C_EXT
57 #define INST_M_EXT
58 #include <riscv_errors.h>
59
60 /* general info */
61 #define DEVICE_FAMILY "LIFCL"
62
63 /* ip instance base address */
64
65 #define CPU0_INST_NAME "cpu0_inst"
66 #define CPU0_INST_BASE_ADDR 0xffff0000
67
68 #define GPIO0_INST_NAME "gpio0_inst"
69 #define GPIO0_INST_NAME GPIO0_INST_NAME
70 #define GPIO0_INST_BASE_ADDR 0x8400
71 #define GPIO_INST_BASE_ADDR GPIO0_INST_BAS
72
73 #define SYSTEM0_INST_AHBL_SLV0_MODEL_MEM_M
74 #define SYSTEM0_INST_AHBL_SLV0_MODEL_MEM_M
75
76 #define SYSTEM0_INST_AHBL_SLV1_MODEL_MEM_M
77 #define SYSTEM0_INST_AHBL_SLV1_MODEL_MEM_M
78
79 #define UART0_INST_NAME "uart0_inst"
80 #define UART0_INST_BASE_ADDR 0x8000
81 #define UART_INST_BASE_ADDR UART0_INST_BAS
82
83
84 /* cpu0_inst parameters */
85 #define CPU0_INST_CACHE_ENABLE False
86 #define CPU0_INST_CACHE_ENV True
87 #define CPU0_INST_CFU_EN False
88 #define CPU0_INST_CFU_N_CFU5 1
89 #define CPU0_INST_C_EXT True
90 #define CPU0_INST_C_STANDALONE True
91 #define CPU0_INST_DCACHE_ENABLE False
```

各モジュール / コンポーネントのドライバーが参照する属性情報の定義ファイル (例)

SoC プロジェクトの構成を定義する情報 (例)

component	ipName	ipVersion	driverVersion	vendor	baseAddress	range	irq
cpu0_inst	riscv_mc	2.6.0	2.5.0	latticesemi.com			
pic_timer_mem_map	PIC_Timer_Registers				0xFFFF0000	0x00008000	
system0_inst	system_memory	2.2.0		latticesemi.com	0x00000000	0x00008000	
gpio0_inst	gpio	1.6.2	1.6.2	latticesemi.com	0x00008400	0x00000400	IRQ_S1
uart0_inst	uart	1.3.0		latticesemi.com	0x00008000	0x00000400	IRQ_S0

ソフトウェア・プロジェクトの作成 (1)

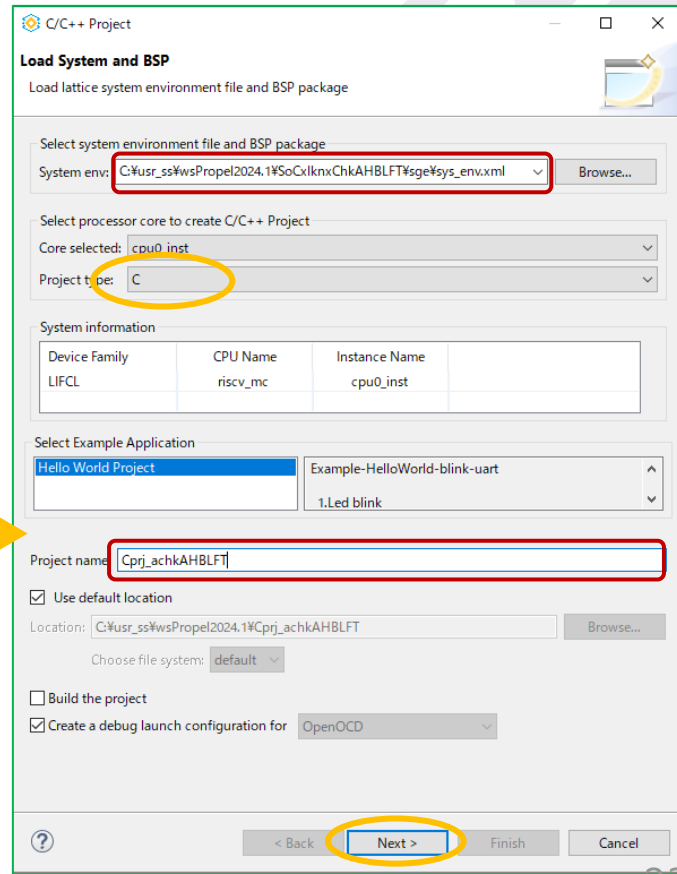
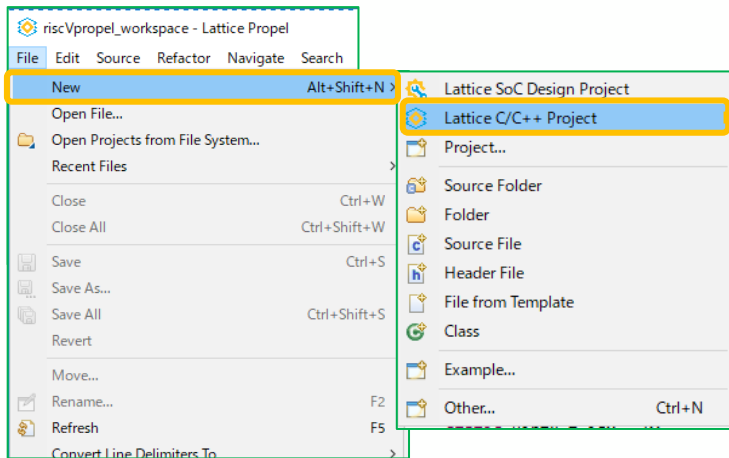
■ SoC プロジェクトの次に C/C++ ソフトウェア・プロジェクトを生成します

1. File → New → Lattice C/C++ Project を選択
2. 立ち上がる“C/C++ Project” 設定 GUI で :

① System env: セルには SoC プラットフォーム作成時に生成された
“sys_env.xml” を指定します (確認します)
(或いはブラウザ・ボタンかプルダウンから意図するものを指定)

② ソフトウェア・プロジェクトの名称を入力し、Project Type (C/C++) を
確認し、Next をクリックします

- 👉 プロジェクト名を入力しないと『Next』ボタンはアクティブになりません
- 👉 ソフトウェア・プロジェクトであることが容易に識別できる名称にします

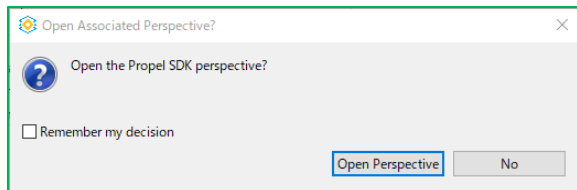
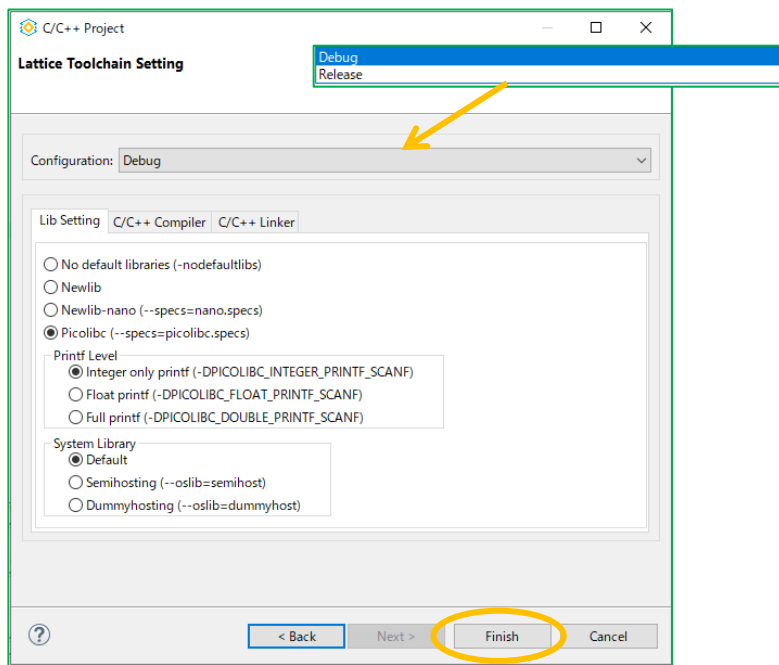


ソフトウェア・プロジェクトの作成 (2)

■ “C/C++ Project” 設定 GUI (つづき)

③ “Configuration:” で Debug / Release を選択して “Finish” します

- ☞ この後、右に示すような確認を促すウィンドウが表示されることがありますが、適宜クリックして進みます
- ☞ [Lib Setting]、[C/C++ Compiler]、[C/C++ Linker] 各タブがありますが、ここでは全てデフォルトのままにします
- ☞ 入力したソフトウェア・プロジェクト名のフォルダが自動生成されます（事前にフォルダを作成しておく必要はありません）
- ☞ プロジェクト作成後の Debug / Release 切り替え方法は p.61 をご参照ください



確認のウィンドウ表示

ソフトウェア・プロジェクトのビルド (1)

- 生成されたプロジェクトとファイル一式は下図例のようになります
 - ☞ Project Explorer 枠の表示は workspace 下に自動作成される C/C++ プロジェクト・フォルダです
 - ☞ 右ソース・ウィンドウは main.c を示しています

生成チュートリアル・プロジェクト
下のファイルとサブフォルダを
展開した状態

自動的に取り込まれた
ドライバー群

本チュートリアル用の
ソースファイル

```
41
42 #include "uart.h"
43 #include "gpio.h"
44 #include "pic.h"
45 #include "utils.h"
46 #include <stdio.h>
47
48 struct uart_instance uart_core_uart;
49 struct gpio_instance gpio_inst;
50
51 int main(void) {
52     static uint8_t idx = 0;
53     static uint8_t pin_state = 0xFF;
54
55     //initialize GPIO
56     gpio_inst.instance_name = GPIO0_INST_NAME;
57     gpio_init(&gpio_inst, GPIO0_INST_BASE_ADDR, GPIO0_INST_LINES_NUM, GPIO0_
58
59 #if _UART_ENABLE_INTERRUPTS_
60     //setup uart IRQ
61     pic_init(CPU0_INST_PICTIMER_START_ADDR);
62     uart_core_uart.intrLevel = UART0_INST_IRQ;
63     pic_isr_register(UART0_INST_IRQ, uart_isr, (void *)&uart_core_uart);
64 #endif
65
66     //initialize UART
67     uart_init(&uart_core_uart, UART0_INST_BASE_ADDR, CPU_FREQUENCY, UART0_INS
68
69 #ifdef LSCC_STDIO_UART_APB
70     extern struct uart_instance *g_stdio_uart;
71     g_stdio_uart = &uart_core_uart;
72 #
73
74     printf("Hello RISC-V world!\r\n");
75
76     while (true) {
77         gpio_output_write(&gpio_inst, idx, pin_state);
78
79         if (++idx == LED_COUNT) {
```

ダブルクリック

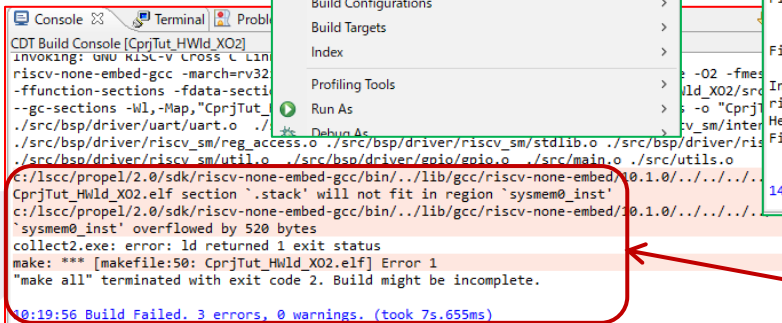
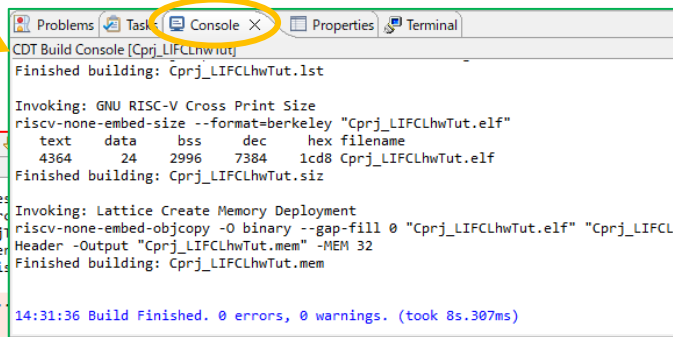
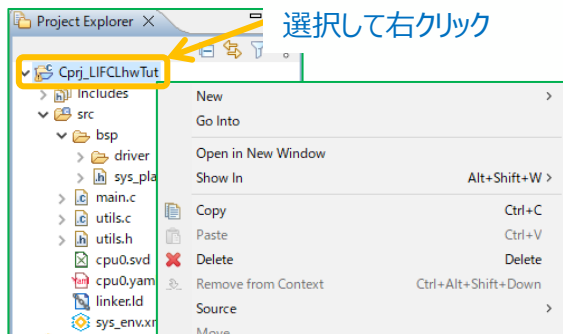
ソフトウェア・プロジェクトのビルド (2)

■ ソフトウェア・プロジェクトをビルドします

1. プロジェクトを選択して右クリックし、“Build Project” を選択します
2. 問題がなければコンソールに <project_name>.mem がビルドされて“Build Finished” というメッセージが表示されます

- 👉 エラーが無くなるまでソースを編集など、デバッグします
- 👉 Warning も（極力）解消するようにします
- 👉 左下は “system” サイズが小さすぎるための Fail 例です

3. 次に Propel Builder で作業して（次ページから）、再度 SDK GUI に戻ります



Builder で設定した system サイズがオーバーフローした（メモリサイズが小さすぎる）ためのエラー例

ソフトウェア・プロジェクトの更新

- Propel Builder で SoC プラットフォームに何らかの変更を行った場合の操作です

右クリック

方法 1

方法 2

確認

← 適宜チェック

Update

Update System and BSP

Select system environment file and BSP package

Current System env: C:/usr_ss/wsPropel2024.1/Cprj_achkAHBLFT/src/sys_env.xml

New System env: C:/usr_ss/wsPropel2024.1/SoCxlknxChkAHBLFT/sgc/sys_env.xml

Re-generate toolchain parameters and linker script

Update BSP package

Update BSP Driver Information


Driver Name	Current IP Version (Driver Version)	New IP Version (Driver Version)
riscv_mc	2.6.0 (2.5.0)	2.6.0 (2.5.0)
gpio	1.6.2 (1.6.2)	1.6.2 (1.6.2)
uart	1.3.0	1.3.0

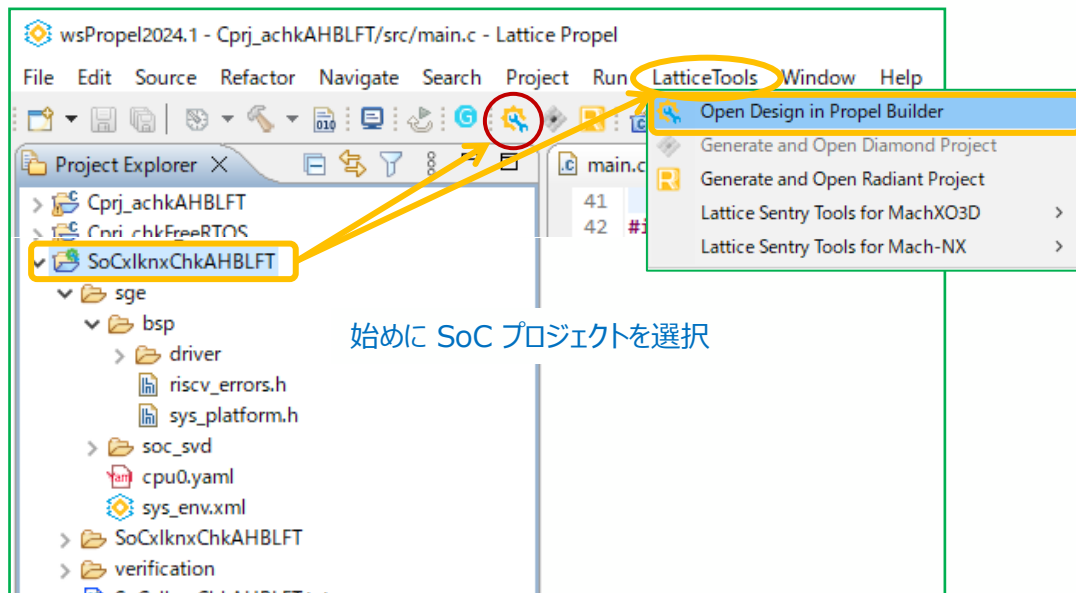
どちらの方法でも、
最初に C/C++
プロジェクトを選択します

- ✓ アドレスマップなどのマイナーな変更の場合は手動で C/C++ プロジェクト下の `sys_env.xml`、`sys_platform.h` を SoC プロジェクトの最新の同名ファイルに差し替え（コピー＆ペースト）しても更新できます
- ✓ 周辺コンポーネントの追加・削除などの変更では左図のように “Update Lattice C/C++ Project...” を選択後の表示ウィンドウで『Update BSP package』ボックスをチェックし『Update』ボタンをクリックすることで一括して自動更新できます
- ✓ CPU の構成やメモリーに変更がある場合などは、『Re-generate toolchain parameters and linker script』ボックスをチェックしてから『Update』します
- ✓ その後どの方法でも “Clean Project” を実行してから “Build Project” を行います

(Propel 2023.2 以前に作成された SoC に対する 2024.1 での本操作はできませんのでご注意ください)

Propel Builder : プログラムメモリーの設定 (1)

- ソフトウェア・プロジェクトのビルドで生成した mem ファイルをプログラムメモリーの初期化に使用するための設定を行います
 1. SoCプロジェクトを選択します (ソフトウェア・プロジェクトではありません)
 2. 次のいずれかの方法で Propel Builder を立ち上げます (次ページ)
 - ✓ アイコン  をクリック (下図赤丸)、または
 - ✓ メニューから [LatticeTools] → [Open Design in Propel Builder] を選択、または
 - ✓ SoCプロジェクト行を右クリックして表示されるメニューから [Open Design In] →[Propel Builder] を選択



Propel Builder : プログラムメモリーの設定 (2)

① systemem インスタンスをダブルクリック

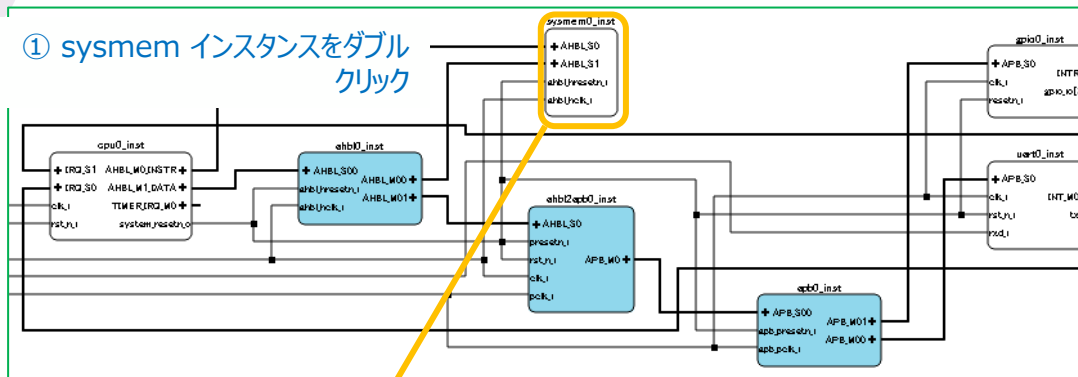


Diagram systemem0

Configure IP

Property	Value
General	
Interface	AHBL
Memory Address Depth [1 - 32768]	8192
Data Bus Width(bits)	32
Memory Type	EBR
Port Count	2
ECC Enable	<input type="checkbox"/>
Enable Arbiter	<input type="checkbox"/>
Data Streamer	
Enable Data Streamer	<input type="checkbox"/>
Data Streamer Interface	Generic FIFO
Data Streamer Clock Bypass	<input type="checkbox"/>
Data Streamer Write Start Address [0 - 8191]	0
Initialization	
Initialize Memory	<input checked="" type="checkbox"/>
Initialization File Format	hex
Initialization File	none

systemem0

- +AHBL_S0
- +AHBL_S1
- ahbl_hclk_i
- ahbl_hresetn_i

system_memory

② チェック

④ mem ファイル指定後、Generate

Initialization	
Initialize Memory	<input checked="" type="checkbox"/>
Initialization File Format	hex
Initialization File	Cpri_LIFCLhwTut/Debug/Cpri_LIFCLhwTutmem

③ セルをクリックするとブラウズボタンがアクティブになるので、ブラウズして *.mem ファイルを指定

■ Propel Builder で mem ファイルをシステムメモリ (プログラムメモリ) の初期化ファイルに指定します

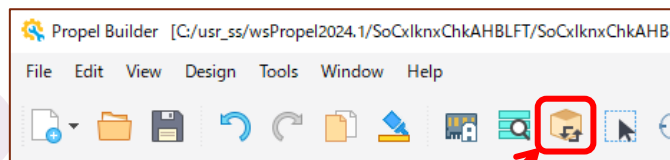
1. systemem インスタンスをダブルクリック
2. "Initialize Memory" のボックスをチェック
3. "Initialize File" セルの右端のブラウズ・ボタンをクリックし、mem ファイルをブラウズして選択 (ソフトウェア・プロジェクト・フォルダ下 Debug/Release サブフォルダの下に生成されています)
4. 当該 mem であることを確認して Generate
5. (次ページ)

Propel Builder : プログラムメモリーの設定 (3)

5. 最後にメニューバーかアイコンで Generate し、保存後 **Propel SDK GUI** に戻ります

- Generate 時に Tcl Console でエラーがないことを確認します
- 必ず保存します

- ☞ この .mem 指定ステップに限り、Generate 後にソフトウェア・プロジェクトを再ビルドする必要はありません
- ☞ システムデバッグ段階でソフトウェア・プロジェクトを再ビルドして *.mem を更新する度に、system コンポーネントをダブルクリックして再度 Generate した後に、SoC プラットフォームを保存するか再 Generate する必要があります
- ☞ system の Generate のみでは、更新された *.mem が反映されません
- ☞ この場合、Radiant / Diamond では論理合成からのやり直しになります
- ☞ 処理時間を短縮する方法が ECO Editor で (pp.47-48)、両方の Generate を省くことができます
- ☞ なお、system の初期化ファイルを指定せずにブランクのままにしておくことで、p.35 の OpenOCD 設定 GUI から起動するデバッガーでのデバッグが可能です。この場合はフィッティングも ECO Editor も不要ですので、C/C++ 記述が確定できずに OpenOCD ベースのデバッグを繰り返す場合の手法の一つとして有効な選択肢です
- ☞ いずれかのコンポーネントの何らかの構成 (パラメーター) を変更した場合は Generate 後に P/F ビルダー下の sys_env.xml、sys_platform.h を SoC プロジェクト下に手動でコピー＆ペーストしてから再ビルドする必要があります (Clean Project → Build Project 操作のみでは自動更新されません。P.27 関連記述)



Generate

```
Tcl Console
INFO - Start: sbp_design generate.
INFO - Finished: sbp_design generate.
% sbp_design save
% sbp_design pge sge -i {C:/usr_ss/wsPropel2022.1/SoC2022pl_LIFCLtut/SoC
Warning[SoCDesign]: cpu0_inst_TIMER_IRQ_M0 remains unconnected
%
```

Radiant フィットティング (1)

- ターゲットデバイスが Nexus シリーズの場合は Radiant を選択します

1. SoC プロジェクトを選択して右クリック (ソフトウェア・プロジェクトではありません)

2. Open Design In → Radiant を選択

(Radiant 2024.1 が立ち上がります : 次ページ)

☞ Diamond はグレイアウトします

☞ SoC プロジェクトを選択した後、メニューバー下部のアイコン列から  をクリックしても同等です

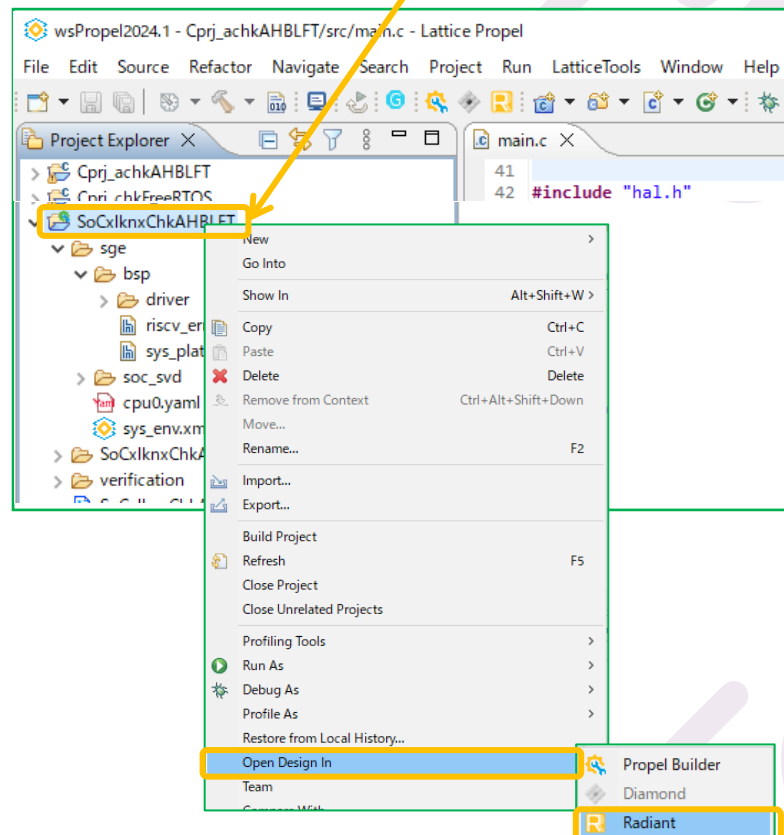
☞ SoC プロジェクトを選択してメニューバー下部のアイコン列から



をクリックすれば Propel Builder が起動し、回路図が表示されます

(右クリックメニュー Open Design In → Propel Builder でも同様)

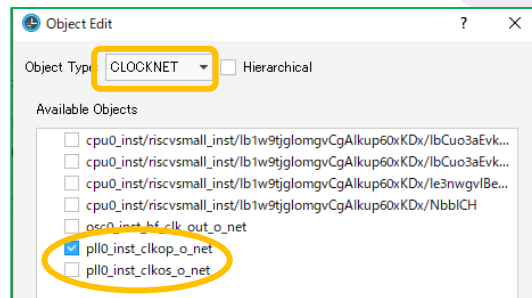
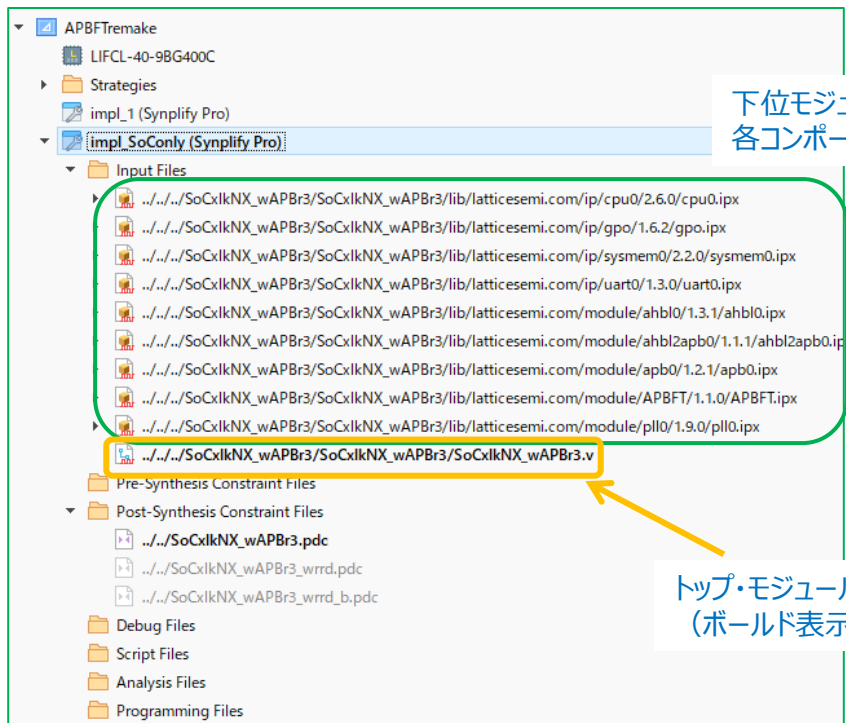
選択して右クリック



Radiant フィットティング (2)


■ Radiant がプロジェクトをオープンした状態で起動します

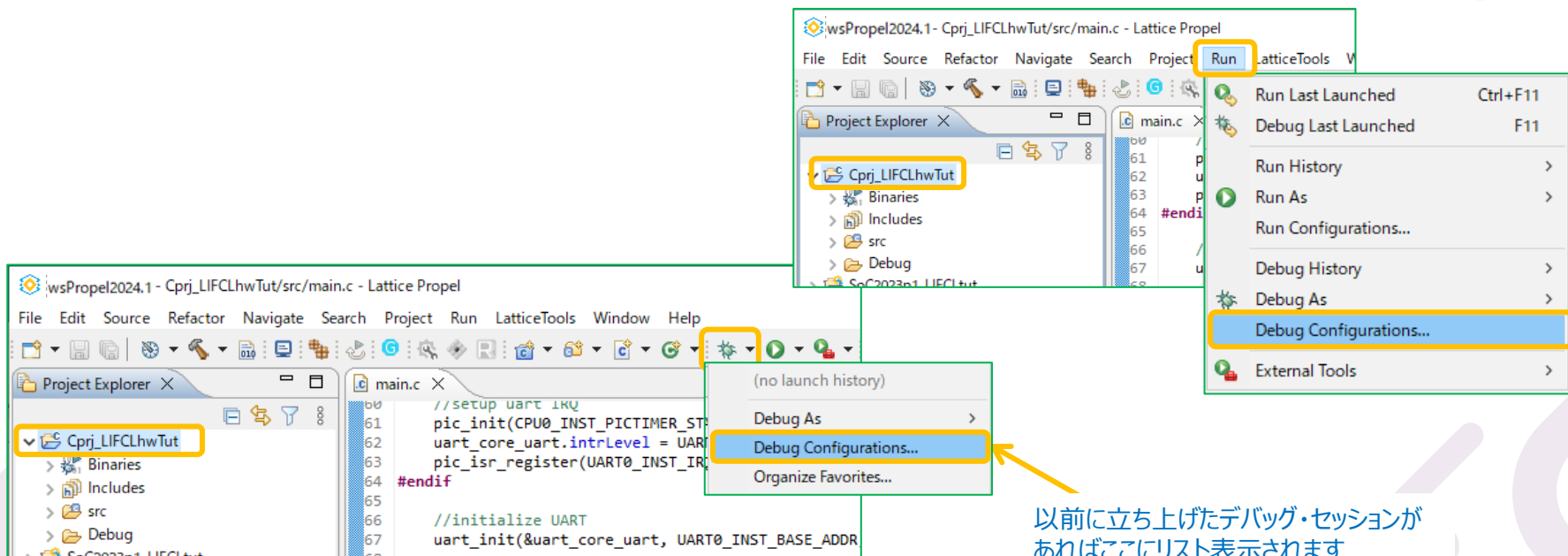
- ☞ インポートされるのはRTL トップ (*.v) と下位コンポーネントの *.ipx です
- ☞ オンチップ・オシレータの周波数は自動認識されますが、仮に認識されないネットがあれば、Post-Synthesis Timing Constraint Editor (または *.pdc ファイルを編集すること) で手動で与えます (下右図、一部)
- ☞ ポート配置指定も基本的には手動で与えます



“Post-Synthesis Timing Constraint Editor”
でクロックネットを指定するウィンドウでの選択例
(p.13 に示す回路図における PLL 出力相当)

SDK デバッグ設定 (1)

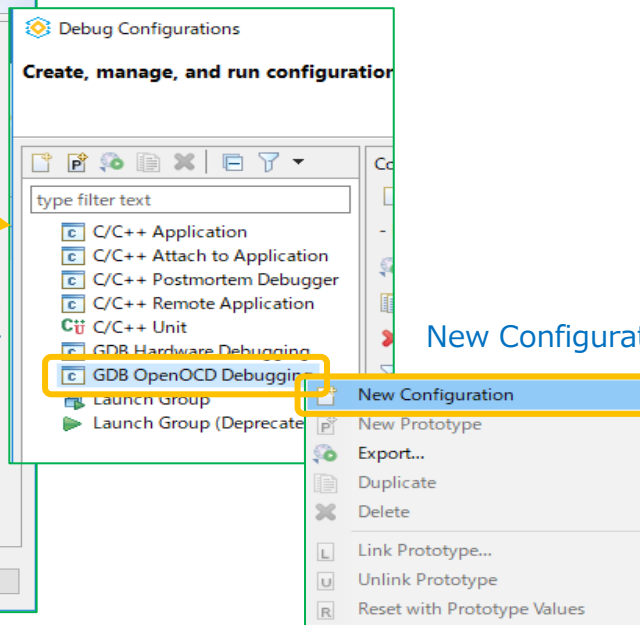
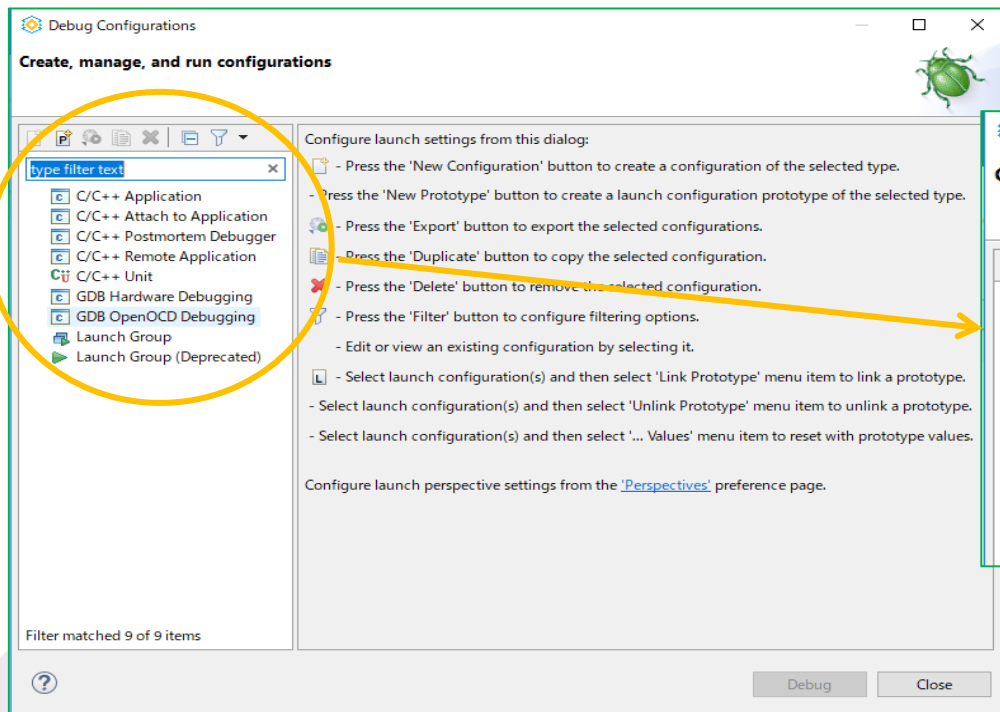
- Programmer でデバイスをプログラミングしておきます
- デバッガー設定を起動しますが、これには二通りの方法があります
 1. デバッグ・アイコン  をクリックします
(下図：アイコン右側の▼部をクリックすると表示される“Debug Configurations...”を選択しても同じです)
 2. メニューバーで Run → Debug Configurations... を選択します (右図)



以前に立ち上げたデバッグ・セッションがあればここにリスト表示されます

SDK デバッグ設定 (2)

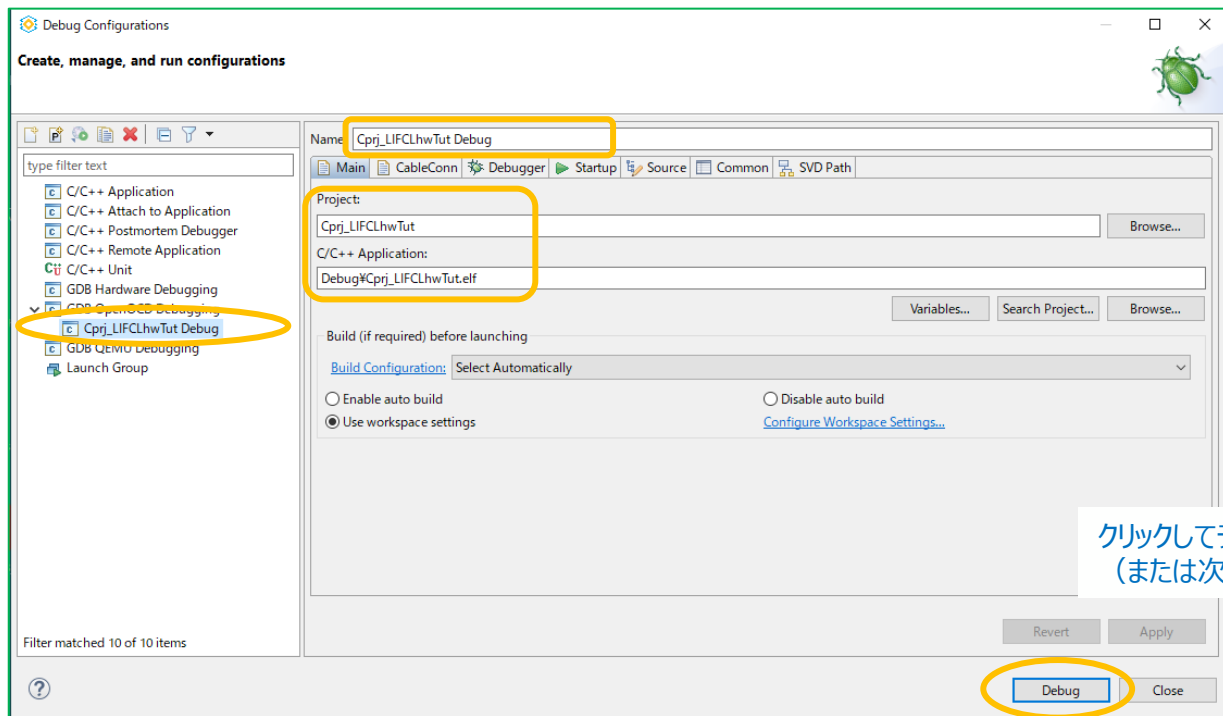
- デバッグ設定用のウィンドウ (Debug Configurations) が表示されます
- “GDB OpenOCD Debugging” 行をダブルクリックするか、本行を選択して右クリックすると表示されるメニューから “New Configuration” を選択します



New Configuration を選択

SDK デバッグ設定 (3)

- “Name” は作業プロジェクトに対して “<Project-name> Debug” になります
- “Project:” セルと “C/C++ Application:” セルが意図するものであることを確認し、“Name セル”のデバッグセッション名をチェックします



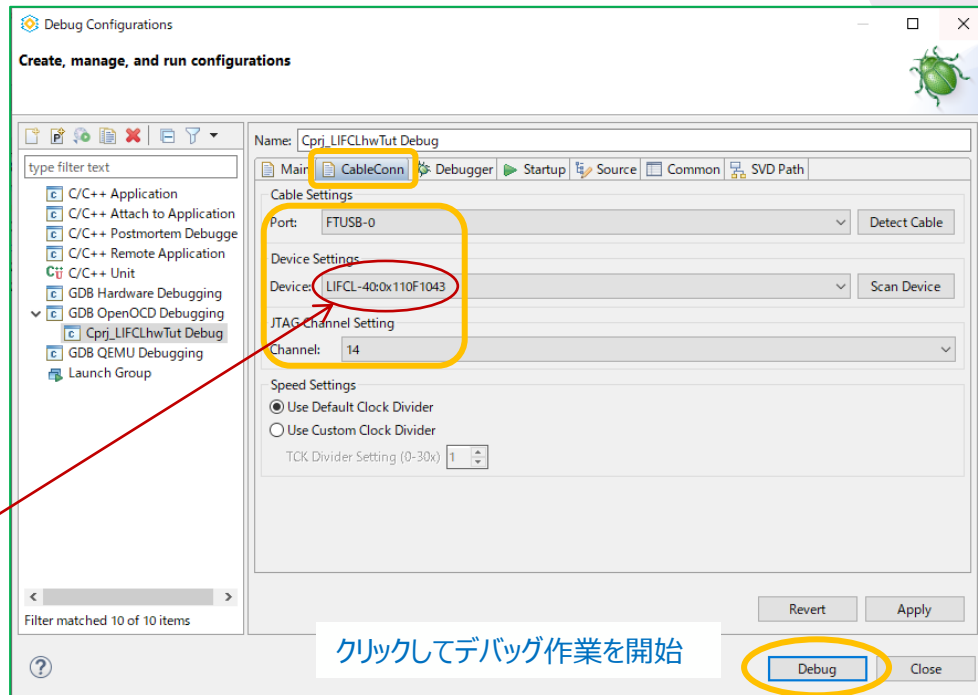
SDK デバッグ設定 (4)

■ 前ページで『Debug』 ボタンをクリックする前の、作業が確実に開始できるかどうか分かるステップです

- ✓ [CableConn] タブを選択し、『Detect Cable』、『Scan Device』 ボタンを順にクリックします
- ✓ “Port” セルが本タブを選択時に “FTUSB-0” のように表示されていれば『Detect Cable』 は省略可です
- ✓ 『Scan Device』 後、“Device” セルが下図例のようになっていればデバッグ開始できます

☞ “Device” セルがブランクのままや
“+” 表示などの場合は何か問題があり
ますので、解消する必要があります

PC の C ドライブ直下に “Data” という名称の
フォルダがあると、『Scan Device』 後にこの
セルが “+” 表示になることが報告されています



SDK デバッグ設定 (5)

- まれにデバッガー動作を開始後、左下のようなエラーがコンソールに表示される場合があります
 - ✓ 少なくとも MachXO3 Starter ボードが該当します
- この場合の対処方法の一つを示します：
 - ✓ [Debugger] タブを選択し、Config options セル内のオプション “set channel” と “set cmdlength” の値を下図のように編集します
 - ✓ 以下 FAQ に記述があります

<https://www.latticesemi.com/support/answerdatabase/7/2/6/7268>

コンソールのメッセージ例

```
Info : clock speed 4000 kHz
Error: JTAG scan chain interrogation failed: all zeroes
Error: Check JTAG interface, timings, target power, etc.
Error: Trying to use configured scan chain anyway...
Error: fpga_spinal.bridge: IR capture error; saw 0x00 not 0x01
Warn : Bypassing JTAG setup events due to errors
Info : Listening on port 3333 for gdb connections
Started by GNU MCU Eclipse
Info : Listening on port 6666 for tcl connections
Info : Listening on port 4444 for telnet connections
Info : accepting 'gdb' connection on tcp/3333
Info : Halt timed out, wake up GDB.
Error: timed out while waiting for target halted
```

Error: JTAG scan chain interrogation failed: all zeroes

The screenshot shows the Eclipse IDE's configuration for OpenOCD. The 'Debugger' tab is active, and the 'Config options' field contains the following command line:

```
-c 'echo "DEBUG_ENABLE=${DEBUGENABLE}"' -c 'set target ${DEVICE}' -c 'set tck ${TCKDIV}' -c 'set port ${PORT}' -c 'set channel 0' -c 'set cmdlength 21' -c 'set loc ${LOCATION}' -f interface/lattice-cable.cfg -c 'set RISCVC SMALL_YAML ${ProjDirPath}/src/cpu0.yaml' -f target/riscv-small.cfg
```

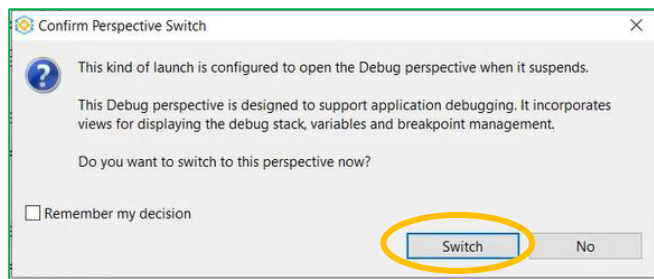
FAQ ページから引用

SDK デバッグ (1)

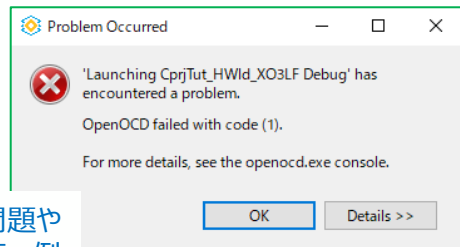
- ① ファイアーウォールのウォーニングが出ますが、許可して進みます
- ② Perspective 切り替えのメッセージが表示されますので、“Switch” をクリックします
- ③ 起動に問題があるとメッセージが出ますので解消してやり直します
- ④ 起動には比較的長い時間がかかりますが、GUI 右下のステータスバーが表示されていれば正常です
- ⑤ デバッガーが起動すると Console にログが表示されます（下に一部のみを例示）

👉 赤字は GDB のログメッセージであり、必ずしもエラーを意味する訳ではありません

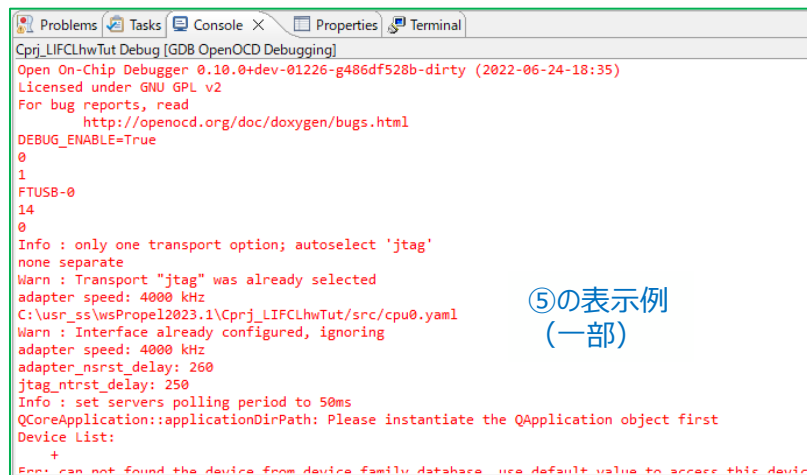
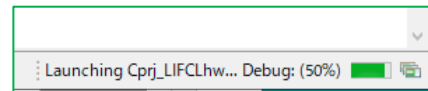
②の表示



③の表示例：デバイス接続の問題や
コンフィグされていない時などのエラー例




④の表示例

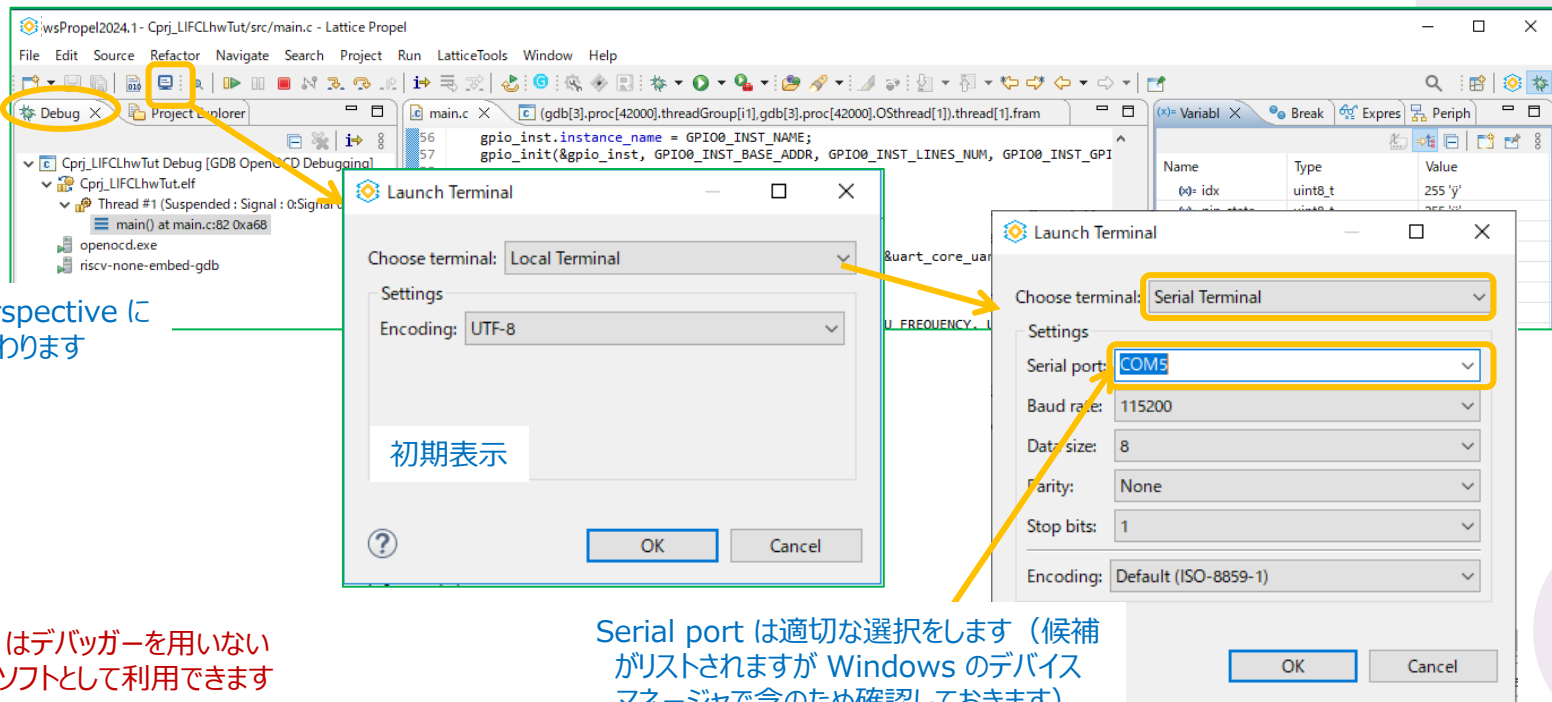


⑤の表示例
(一部)

SDK デバッグ (2)

- 起動に成功すると、表示が **Debug Perspective** に切り替わります
 - ✓ main() 先頭で自動的に停止します
 - ✓  アイコン (Open a terminal) をクリックし、ターミナル (UART) 設定をします
 - “Serial Terminal” にプルダウンを変更し、“Serial port” を候補から選択します

Debug Perspective に
切り替わります





初期表示

Serial port は適切な選択をします (候補がリストされますが Windows のデバイスマネージャで念のため確認しておきます)

Serial Terminal はデバッガーをいれない時でも、ターミナル・ソフトとして利用できます

SDK デバッグ (3)

- 前ページの操作でコンソール部に新たに Terminal タブが現れますので、これを選択します

- ✓  アイコン (resume) をクリックして、プログラムを実行します
 - "printf" 文でメッセージを出力する記述があれば Terminal に表示されます
- ✓ 実行停止は  アイコン (suspend) です
- ✓ ブレークポイント設定 (次ページ)、ステップ実行 (右) などでデバッグ作業を進めます
- ✓ メモリ内やレジスタ値を表示させる機能もあります

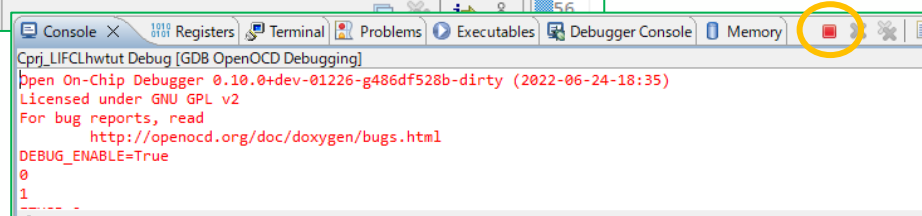
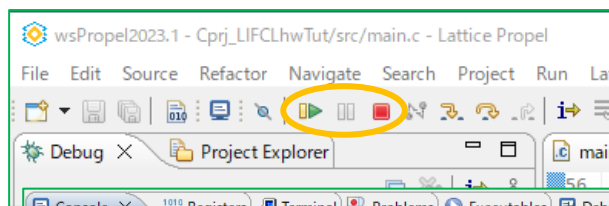
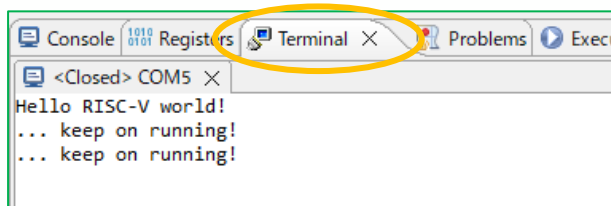
- デバッガ (GDB セッション) の終了は GUI 上部のアイコン列にある  か、右下 "Console" タブを選択すると右側にある  アイコン (terminate) をクリックします

→ 注意事項 : p.43



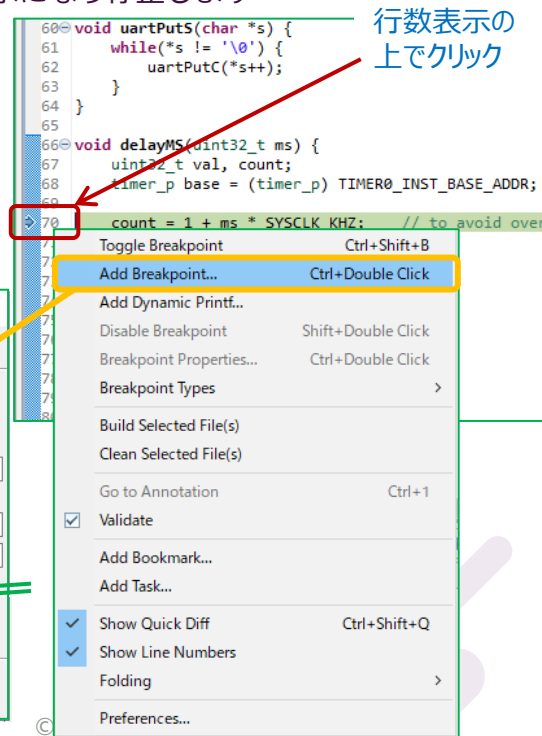
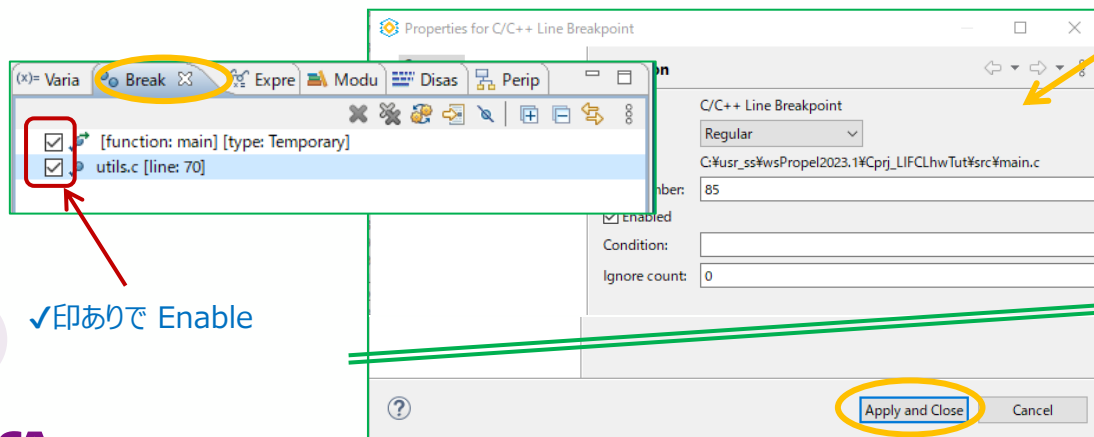
step return
step over
step into

step into 下位関数が呼ばれば、その行に進む
step over 下位関数が呼ばれても、そこに飛ばずに次の行に進む
step return 下位関数に飛んだ後、飛んだ元の次の行に進む
等々、...



SDK デバッグ (4)

- ブレークポイントの設定はデバッガー実行中に一旦停止して以下を操作します
 - ✓ ブレークするソース行を選択 → 行頭の行数表示箇所ですクリックして “Add Breakpoint...” をクリックします
 - ✓ プロパティ・ウィンドウで “Apply and Close” (“Ignore count” 0 で毎回ブレーク) します
 - ✓ 右上ウィンドウ枠の “Break” ビューで設定されているブレーク・ポイント一覧が表示されます
 - ✓ 実行 (Resume) してブレークポイントに達すると、そのソースコード行が緑色表示になり停止します
- 解除は同様にして “Toggle Breakpoint” するか、Break ビュー内リストのチェックボックスをクリック (disable) します
- 設定済ブレークポイントのプロパティ変更は Break ビュー内で選択して右クリック → “Breakpoint Properties...” をクリックして再設定します



SDK デバッグ (5)

■ デバッガー GUI デフォルトで右上には種々 View がタブで選択できます

- ✓ Variables / Breakpoints / Peripherals / Modules ...
 - [Variables] タブではグローバル / ローカル変数のその時点の値を確認および編集できます
 - [Breakpoints] タブではブレークポイントの確認やイネーブル / ディセーブル設定ができます

■ 次の操作をすることで逆アセンブル・コードを表示すると、ステップ実行の様子を確認することができます


- ✓ Window → Show View → Disassembly


The screenshot illustrates the steps to view assembly code in the Lattice Propel IDE. The 'Window' menu is open, and 'Show View' is selected. The 'Disassembly' view is highlighted in the list of views. A yellow arrow points from the 'Disassembly' option to the Disassembly window, which shows the assembly code for the 'main' function.

```
main:  
00000200:  addi  sp,sp,-32  
00000202:  sw    s1,20(sp)  
00000204:  lui   a3,0x1c  
00000206:  lui   s1,0x8  
00000208:  addi  a1,s1,1024 # 0x8400  
0000020c:  li    a5,8  
0000020e:  li    a4,1  
00000210:  addi  a3,a3,512 # 0x1c200  
00000214:  li    a2,38  
00000218:  addi  a0,gp,-1964  
0000021c:  sw    ra,28(sp)  
0000021e:  sw    s0,24(sp)  
00000220:  sw    s2,16(sp)  
00000222:  addi  s0,gp,-1964  
00000226:  sw    s3,12(sp)  
00000228:  sw    s4,8(sp)  
0000022a:  sw    s5,4(sp)  
0000022c:  jal   0x92 <uart_init>  
55      g_stdio_uart = &uart_core_uart;  
0000022e:  lui   a0,0x1  
00000230:  lui   a5,0x1  
00000232:  addi  a0,a0,-40 # 0xfd8  
00000236:  sw    s0,240(a5) # 0x10f0 <g_stdio_uart>  
58      printf("Hello RISC-V world!\r\n");  
0000023a:  jal   0x30c <puts>  
59      LED_SET(ALL_OFF);  
0000023c:  li    a5,255  
00000240:  sw    a5,4(<s1)
```

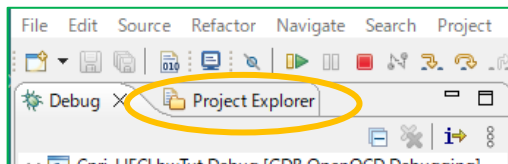
SDK デバッグ (6)

- デバッガーを Terminate しただけでは元の SDK Perspective 表示には戻りません

1. GUI 左上の『Project Explorer』タブをクリック、または
2. GUI 右上の  アイコン (Open Perspective) をクリック、または
3. メニューバーで次の操作をします : Window → Perspective → Open Perspective → Propel SDK

重要 : デバッガーを終了する前に Propel を終了したり、ボードを取り外すと Programmer が誤動作する状態になる可能性があります (PC を再起動しても復帰しない)。必ず p.40 に示すとおり  アイコンで通常終了してから次の作業をするようにします

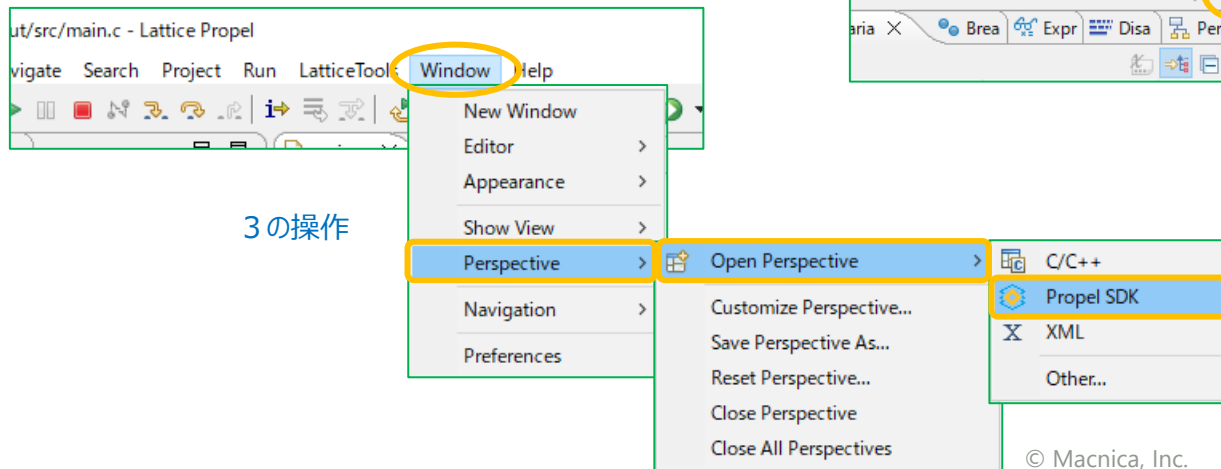
1の操作



2の操作



3の操作



補 足

- 補足 1 無償ライセンス・ファイルの生成・入手
- 補足 2 What's New in Propel 2024.1
- 補足 3 ECO Editor
- 補足 4 GPIO コンポーネントの多ビットポート
- 補足 5 ユーザーモジュールの組み込み
- 補足 6 内部バス引き出しでユーザー回路とインターフェイス
- 補足 7 SoC/C プロジェクトのインポート
- 補足 8 SoC プラットフォームのインポート
- 補足 9 C/C++ プロジェクトのインポート
- 補足 10 SDK ビルド設定
- 補足 11 Propel 起動後のワークスペースの切り替え

補足 1 : 無償ライセンス・ファイルの生成・入手

1. ラティスのホームページから Propel のページに移動し、“Licensing” をクリックします
2. 移動先で “Request a Free License” をクリックします
3. 移動先で “Name”, “Email” を確認し Host NIC セルに入力します
4. “I verify ...” の内容を確認し文頭をチェックします
5. “Generate Licensing” をクリックします

Please follow these steps to request your Lattice Propel Node-locked License:

1. Review your Web Account information below. [[Edit](#)]

Name: registered name

Email: registered email address

2. Fill in the Software License Request Form and Submit.

Finding the Host NIC:

For Windows, from an MS-DOS window, use the ipconfig /all command

For Linux, from the command prompt, use the ifconfig -a command

Software License Request Form

Note: The license file will be sent to the web account email address: **registered email address**

Host NIC (physical address) *

Please enter your 12-digit NIC without any separator or spaces.

I verify that I am not an employee of Cadence Design Systems, Mentor Graphics Corporation, or

(click once)

NOTE: This form requires JavaScript to be enabled in your web browser in order to process your request.

https://www.latticesemi.com/en/Products/DesignSoftwareAndIP/FPGAandLDS/LatticePropel

Getting Started

1. Download: Choose and download software from the [Software Downloads & Documentation](#) table below

2. Install: Follow the installation guide, found in [Software Downloads & Documentation](#) section below.

3. License: You will need a Lattice Propel [button](#) below to request a license.

(事前にユーザー登録している必要があります)

Licensing

Contact Sales

← → ↻ 🏠 https://www.latticesemi.com/en/Support/Licensing

Lattice Propel License

The Lattice Propel design environment for Lattice FPGA-based processor systems includes an IP catalog linked to Lattice's on-demand IP server, IP system integration tools, and a software development kit (SDK). The Propel License enables users to access this easy to use system integration environment.

To request a license you will need the following:

- Physical MAC address (12-digit hexadecimal)

2. 移動先で Propel のセクションにあるリンク “Request a Free License” をクリックします

補足 2 : Propel 2024.1

■ What's New in Propel 2024.1 (Rel.Notes) →

■ なお、本ドキュメントで参照した関連文書は以下の通りです：

- [FPGA-UG-02211-1-0-Lattice-Propel-2024-1-SDK-User-Guide](#)
- [fpga-ug-02212-1-0-lattice-propel-2024-1-builder-user-guide](#)

New Operating System (OS) Support

- Ubuntu 22.04 LTS

New Device Support

- Lattice ECP5U™
- Lattice ECP5UM™
- Lattice ECP5UM5G™

Tools and Enhancements

- Supports user custom application templates
- Supports TCL in IP Packager
- Supports ECP5 and ECP5-5G devices
- Supports Lattice Avant RISC-V MC/RX SoC templates
- Supports "Attach to running target" in debugger
- Supports GUI color customization options for schematic
- Supports code coverage and timing profiling on *RISC-V RX SoC Project*
- Supports an extension on C projects created for *RISC-V RX SoC Project*
- Supports a new entry to distinguish SoC creation from custom templates or built-in templates
- Supports QuestaSim instead of ModelSim
- Supports DRC of generating default value in top RTL file for AMBA4 dangling optional ports
- Supports DRC of cacheable address range on SoC including RISC-V RX processor
- Supports DRC of connection compatibility between RISC-V RX processor and TCM
- Supports VHDL for RTL module of glue logic
- Supports friendlier interface names in IP Packager GUI Display
- Supports QEMU Virtual Platform
- Supports creating application template *Hello World Project* for RISC-V MC/SM/RX minimum system
- Supports creating application template *RX Demo Project* for RISC-V RX minimum system
- Supports creating default debug launch configuration when creating a C project
- Supports enabling/disabling automatically build the project when creating a C project
- Hides glue logics from verification project view
- Supports read-only address map for verification projects
- Adds a new entry of importing Lattice C/C++ Projects into Workspace
- Adds application template *FreeRTOS-LTS-Minimal Project*
- Former *FreeRTOS Project* is renamed to *FreeRTOS-LTS PMP-Blinky Project*
- FreeRTOS Kernel update from v10.0.1 to v10.5.1 based on FreeRTOS 202210.01 LTS

補足3 : ECO Editor (Radiant)

- プログラムメモリー `system` の初期化ファイルのみを変更するツールが ECO エディターです
 - ✓ ユーザー・アプリケーションのデバッグ完了までは、C/C++ コードの記述を修正・編集して確認する、繰り返し作業になることが一般的です
 - ☞ その都度、実行コード `*.elf` を変換した `*.mem` ファイルが更新されます
 - ✓ この更新された `*.mem` を反映するためには `system` コンポーネントをダブルクリックして再度 Generate した後、SoC プラットフォームを保存して再 Generate する必要があります
 - ☞ `system` の Generate のみでは更新された `*.mem` が反映されません
 - ☞ この場合、Radiant / Diamond では論理合成からのやり直しになります
 - ☞ 全てのデバッグが完了した後、デザインのリリース段階では少なくとも再 Generate が必要です
 - ✓ Radiant / Diamond には `system` の初期化ファイル `*.mem` のみの変更が可能なツール“ECO Editor”があります。
 - ☞ これは PAR 完了後のネットリストに対する操作であり、SoC プラットフォーム自体が変更されていなければ、PAR までのフィッティング処理は一切不要になり、繰り返し処理の時間を大幅に節約できます
 - ☞ `system` コンポーネントや SoC プラットフォームの再 Generate は不要です
 - ✓ SoC プラットフォームに含まれる各コンポーネントの構成（パラメータ）や回路の変更があれば、フィッティングは論理合成から再実行する必要があります。`system` のサイズやアドレスマッピングの変更もこれに含まれます
 - ☞ OpenOCD ベースのデバッグを繰り返す場合、`system` の初期化ファイルを指定せずにブランクのままにしておく方法があります（フィッティングも ECO Editor も不要です）。ビルド後に P.33 のデバッグ設定から作業します

補足3 : ECO Editor (Radiant、つづき)

■ プログラムメモリー sysmem の初期化ファイル更新手順は以下のとおりです

① PAR まで完了後  アイコンをクリックします

② ECO Editor 下部の “Memory Initialization” タブを選択します

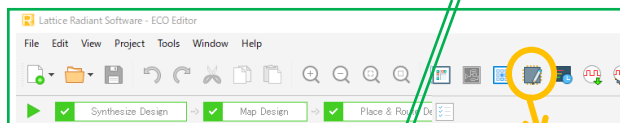
③ “Memory Instance” 名の中から “sysmem0” のようなインスタンス名に相当するものを特定します (デザインに依っては他にもメモリ・インスタンスがある場合がありますので、間違えないようにします)

④ Attribute セルにある “Choose File ...” をダブルクリックすると表示される “Update Initial Memory” GUI で “File Format” を “Hexadecimal” に変更後 “Memory file” セル右端のブラウズボタンをクリックして、意図する .mem ファイルを指定 (絶対パス表記) し、OK ボタンをクリックします

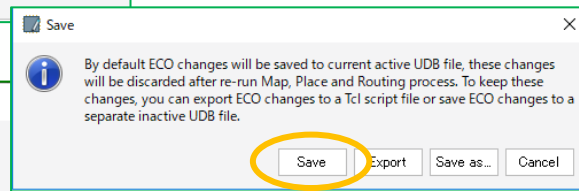
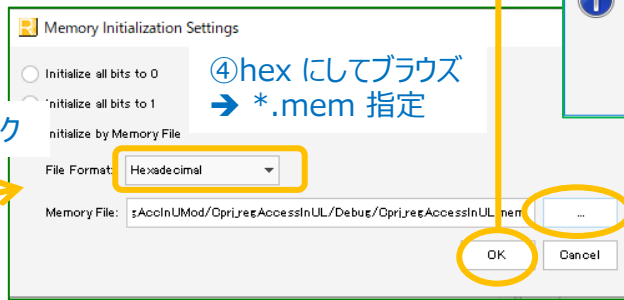
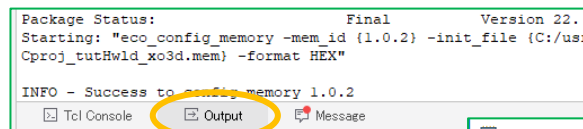
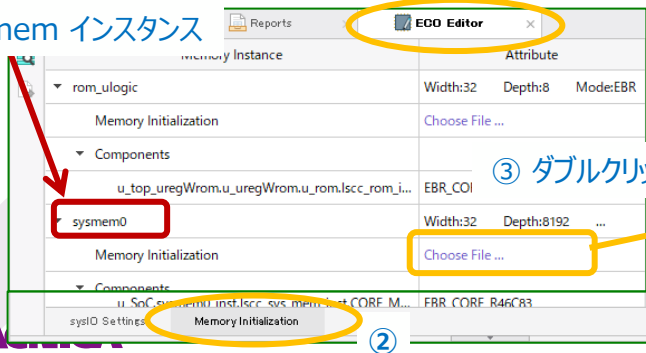
⑤ Output 枠に “Success to update memory 1.0.2” が出れば成功です

⑥ 保存アイコン  をクリックすると表示されるメッセージで “Save” をクリックして ECO Editor を終了します

⑦ Export Files プロセスを再実行します

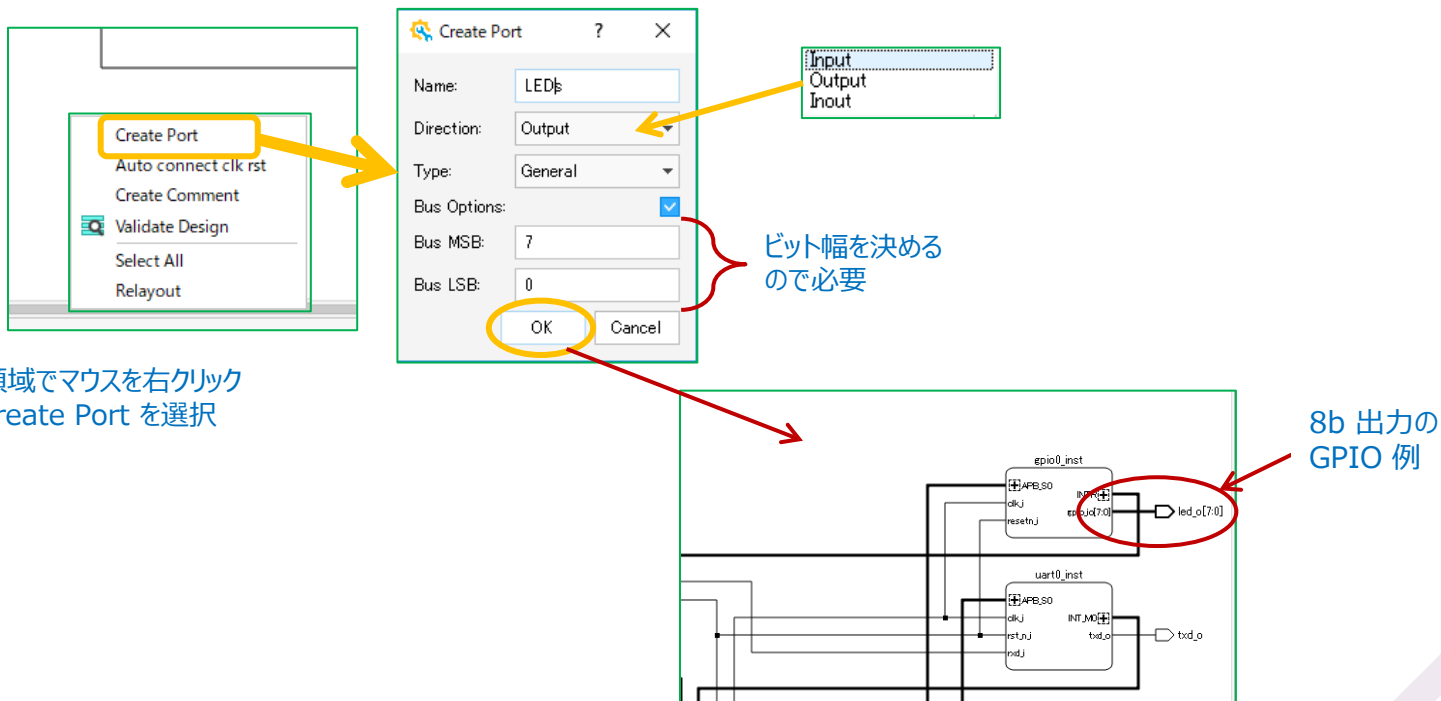


sysmem インスタンス



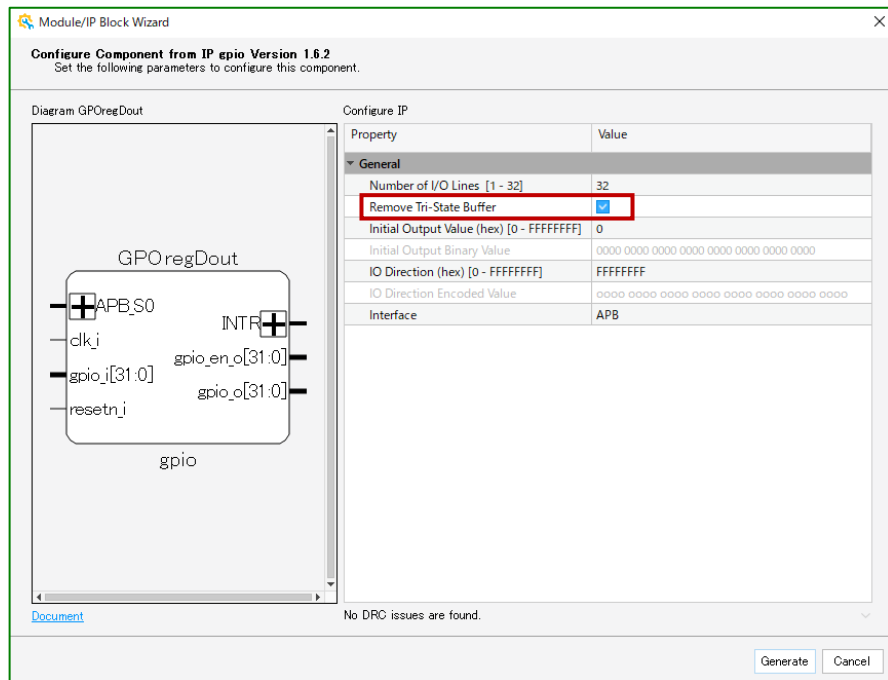
補足 4 : GPIO 外部ポートの多ビットポート

- SoC ビルダで回路図作成時に多ビット幅外部ポートを追加する際の例です
 - ✓ Bus Options のボックスをチェックし、MSB と LSB のビット位置を明示的に入力します
 - ✓ その後 Schematic で結線します



補足 5 : GPIO ポートをユーザー論理と接続する際の注意

- ユーザー論理は RTL トップで SoC と統合（編集）しますが、その際は以下に留意します
 - ✓ 下図に示す通り “Remove Tri-State Buffer” をチェックします（P.17 に関連記述）
 - 双方向バッファがないため、I/O ポートは “gpio_i”、“gpio_o”、“gpio_en_o” になります
 - 出力設定時 “gpio_en_o” はトライステート・バッファ制御用です。双方向バッファを制御しない場合はオープンでも構いません



The screenshot shows the 'Module/IP Block Wizard' for the 'GPIO' component. The 'Configure Component from IP gpio Version 1.6.2' dialog is open. The 'Diagram GPOregDout' shows a block diagram of the 'gpio' component with inputs 'APBS0', 'clk_i', 'gpio_i[31:0]', and 'resetn_i', and outputs 'INTR', 'gpio_en_o[31:0]', and 'gpio_o[31:0]'. The 'Configure IP' table on the right shows the following properties:

Property	Value
General	
Number of I/O Lines [1 - 32]	32
Remove Tri-State Buffer	<input checked="" type="checkbox"/>
Initial Output Value (hex) [0 - FFFFFFFF]	0
Initial Output Binary Value	0000 0000 0000 0000 0000 0000 0000 0000
IO Direction (hex) [0 - FFFFFFFF]	FFFFFFFF
IO Direction Encoded Value	0000 0000 0000 0000 0000 0000 0000 0000
Interface	APB

At the bottom of the dialog, it says 'No DRG issues are found.' and there are 'Generate' and 'Cancel' buttons.

補足 6 : 内部バス引き出しでユーザー回路と I/F (1)

■ APB バスを引き出す方法を示します

- ✓ RTL トップで APB バス I/F をもつユーザー回路と Propel 生成 SoC RTL とを統合する場合です
 1. “APB Interconnect” コンポーネントをダブルクリックし、マスター (M) ポート数を一つ増やしておきます
 2. Module → “Processors...” → “APB Feedthrough” コンポーネントを追加し、その S ポートを “APB Interconnect” コンポーネントの M ポートに接続します
 3. APB Feedthrough コンポーネントの設定は “Completer” を指定します (アドレスマッピングの対象になります)
 4. バス引き出しはコンポーネントを選択して右クリック後のメニューから Export を選択します
 5. 『DRC』 実行後 『Generate』 します (必要に応じてアドレスを編集したり “Lock” します)

バスを接続

バスが引き出される

APB Interconnect コンポーネント

APB Feedthrough コンポーネントの設定例

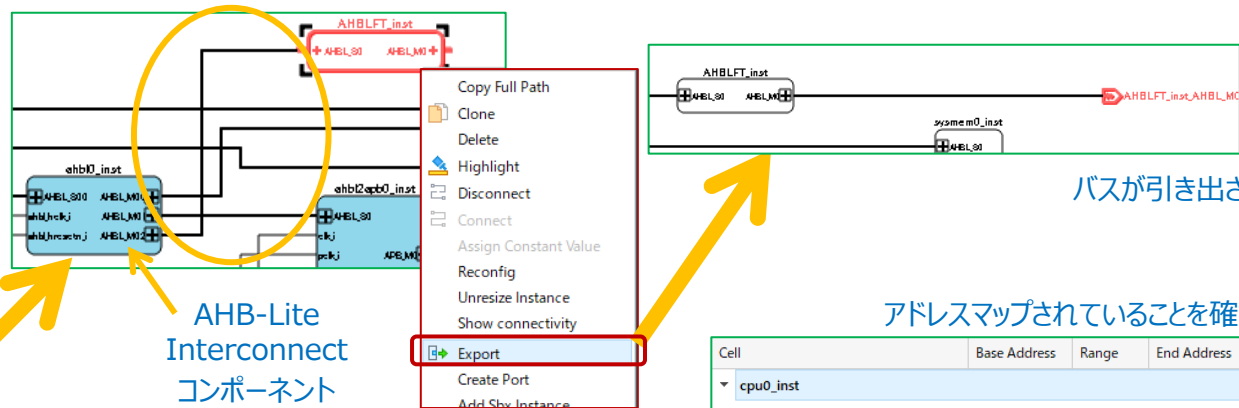
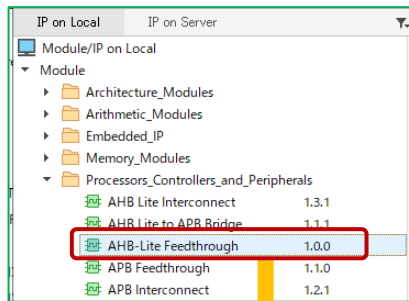
アドレスマップされていることを確認

Cell	Base Address	Range	End Address	Lock
▼ cpu0_inst				
▶ LocalMemory				
▶ SoC_APBFT_lfd2nx/cpu0_inst/riscv_ahbl_m_instr_Address_Space(32 address bits: 4G)				
▼ SoC_APBFT_lfd2nx/cpu0_inst/riscv_ahbl_m_data_Address_Space(32 address bits: 4G)				
APBFT_inst/APB_S0	0x00008800	1K	0x00008BFF	<input type="checkbox"/>
gpio0_inst/APB_S0	0x00008400	1K	0x000087FF	<input type="checkbox"/>
system0_inst/AHBL_S1	0x00000000	32K	0x00007FFF	<input checked="" type="checkbox"/>
uart0_inst/APB_S0	0x00008000	1K	0x000083FF	<input checked="" type="checkbox"/>

補足 6 : 内部バス引き出しでユーザー回路と I/F (2)

■ AHBL バスを引き出す方法を示します

- ✓ RTL トップで AHBL バス I/F をもつユーザー回路と Propel 生成 SoC RTL とを統合する場合は
 1. “AHBL Interconnect” コンポーネントをダブルクリックし、マスター (M) ポート数を一つ増やしておきます
 2. Module → “Processors...” → “AHB-Lite Feedthrough” コンポーネントを追加し、その S ポートを “AHBL Interconnect” コンポーネントの M ポートに接続します
 3. AHBL Feedthrough コンポーネントの設定は “Slave” を指定します (アドレスマッピングの対象になります)
 4. バス引き出しはコンポーネントを選択して右クリック後のメニューから Export を選択します
 5. 『DRC』 実行後 『Generate』 します (必要に応じてアドレスを編集したり “Lock” します)



Configure AHBLFT:

General

Property	Value
Address Width(bits)	10
Data Bus Width(bits)	32
Export Interface as	Slave
Memory Map Range (0x)	0x400

Cell	Base Address	Range	End Address	Lock
cpu0_inst				
LocalMemory				
SoC_AHBLFT_lfd2nx/cpu0_inst/riscv_ahbl_m_instr_Address_Space(32 address bits: 4G)				
SoC_AHBLFT_lfd2nx/cpu0_inst/riscv_ahbl_m_data_Address_Space(32 address bits: 4G)				
AHBLFT_inst/AHBL_S0	0x00008800	1K	0x00008BFF	<input type="checkbox"/>
gpio0_inst/APB_S0	0x00008400	1K	0x000087FF	<input checked="" type="checkbox"/>
system0_inst/AHBL_S1	0x00000000	32K	0x00007FFF	<input checked="" type="checkbox"/>
uart0_inst/APB_S0	0x00008000	1K	0x000083FF	<input checked="" type="checkbox"/>

補足 6 : 内部バス引き出しでユーザー回路と I/F (3)

注 : 現状 2024.1 でこの構成で DRC を実行すると Propel Builder が強制終了します。原因は不明です。

■ AXI4 バスを引き出す方法を示します

- ✓ RTL トップで AXI4 バス I/F をもつユーザー回路と Propel 生成 SoC RTL とを統合する場合です
 1. “AHBL Interconnect” コンポーネントをダブルクリックし、マスター (M) ポート数を一つ増やしておきます
 2. IP → “Processors...” → “AHB-Lite to AXI4 Bridge” コンポーネントを追加し、その S ポートを “AHBL Interconnect” コンポーネントの M ポートに接続します
 3. IP → “Processors...” → “AXI4 Feedthrough” コンポーネントを追加し、その S ポートを “AHB-Lite to AXI4 Bridge” コンポーネントの M ポートに接続します
 4. AXI4 Feedthrough コンポーネントの設定は “Subordinate” を指定します (アドレスマッピングの対象になります)
 5. バス引き出しはコンポーネントを選択して右クリック後のメニューから Export を選択します
 6. 『DRC』 実行後 『Generate』 します (必要に応じてアドレスを編集したり “Lock” します)

Configure AHBtoAXI4:

Property	Value
General	
Address Width(bits)	32
Data Bus Width(bits)	32
AXI ID Width(bits)	4
AXI Secure Access	<input type="checkbox"/>
AXI Timeout Counter	0
Read Data bus Pipeline Enable	<input type="checkbox"/>
Write Data bus Pipeline Enable	<input type="checkbox"/>

AHB-Lite to AXI4 Bridge の設定例

バスを接続

バスが引き出される

Configure AXI4FT:

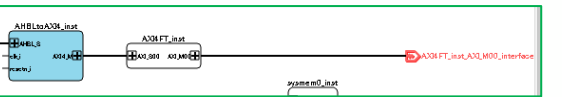
Property	Value
General	
AXI Address Width(bits)	32
AXI Data Bus Width(bits)	32
AXI ID Width (bits)	4
AXI User Width (bits)	32
Export Interface as	Subordinate
Memory Map Width(bits)	32
Memory Map Range (0x)	0x11

Manager (Subordinate)

AXI4 Feedthrough コンポーネントの設定例

- Copy Full Path
- Clone
- Delete
- Highlight
- Disconnect
- Connect
- Assign Constant Value
- Reconfig
- Unresize Instance
- Show connectivity
- Export**
- Create Port
- Add Sub Instance

タイプ以外に特に ID 幅やメモリマップ幅などの設定にも注意します



アドレスマップされていることを確認 (Mem.Map Width=16b の場合)

Cell	Address	Base Address	Range	End Address	Lock
cpu0_inst					
LocalMemory					
SoC_AXI4FT_ifd2nx/cpu0_inst/riscv_ahbl_m_instr_Address_Space(32 address bits: 4G)					
SoC_AXI4FT_ifd2nx/cpu0_inst/riscv_ahbl_m_data_Address_Space(32 address bits: 4G)					
AXI4FT_inst/AXI_S00	0x00010000	64K	0x0001FFFF	<input type="checkbox"/>	
gpio0_inst/APB_S0	0x00008400	1K	0x000087FF	<input checked="" type="checkbox"/>	
system0_inst/AHBL_S1	0x00000000	32K	0x00007FFF	<input checked="" type="checkbox"/>	
uart0_inst/APB_S0	0x00008000	1K	0x000083FF	<input checked="" type="checkbox"/>	

補足7 : SoC/Cプロジェクトのインポート

- SoC と C/C++ プロジェクトを複製しないでインポートする手順です
 1. 作業フォルダを作成し、その下にパッケージ zip ファイルを解凍しておきます
 - デスクトップや日本語名フォルダの下は避けるようにします
 2. Propel SDK を起動し、“workspace” の場所を当該作業フォルダにして立ち上げます（重要）
 3. SDK GUI で File メニューから“Open Projects from File System...”を選択
 4. 『Directory』をクリックしてインポート元のディレクトリを指定します
 - “.settings” フォルダの見えるディレクトリです
 - SoC or C/C++ プロジェクトそれぞれについて、この作業を行います
 5. Project Explorer 内にインポート元と同名のプロジェクトが表示されます（インポート先に同名の新規フォルダは作成されません）

SoC プロジェクトの場合の例

“.settings” フォルダの見えるディレクトリを選択

C/C++ プロジェクトの場合の例

© Macnica, Inc.

補足 8 : SoC プロジェクトのインポート (1)

- 既存の SoC プロジェクトを複製して再利用する場合 “Propel SDK からインポート” します
 - ✓ ワークスペースの直下に新たにオリジナルと同名の SoC プロジェクトが複製されて、インポートされます
 - ✓ これによってその後の再利用が可能になります

ブラウザ時、SoC プロジェクトを作成時に自動生成された SoC フォルダを指定して『フォルダの選択』をクリックします

クリック

または、トップメニューから File → Import... を選択します

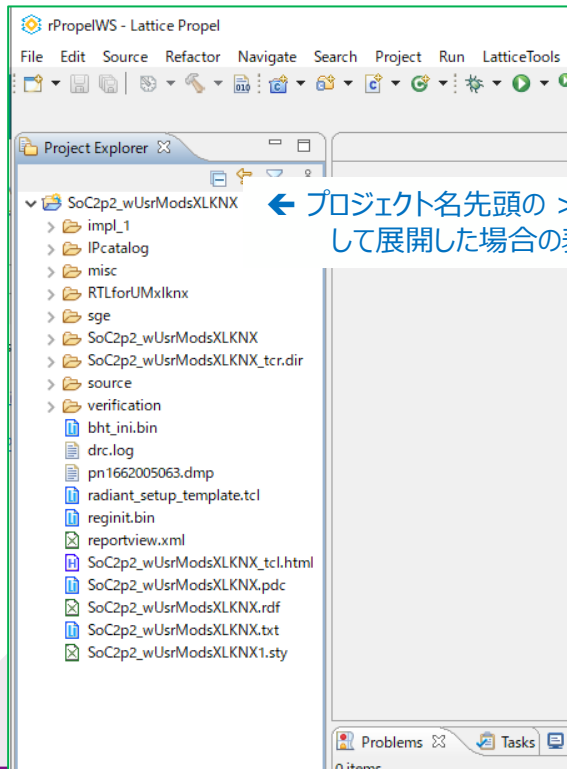
“Lattice Propel” を展開して選択、クリックします

有効なプロジェクトが認識されると、候補が表示されますので『Finish』をクリックします

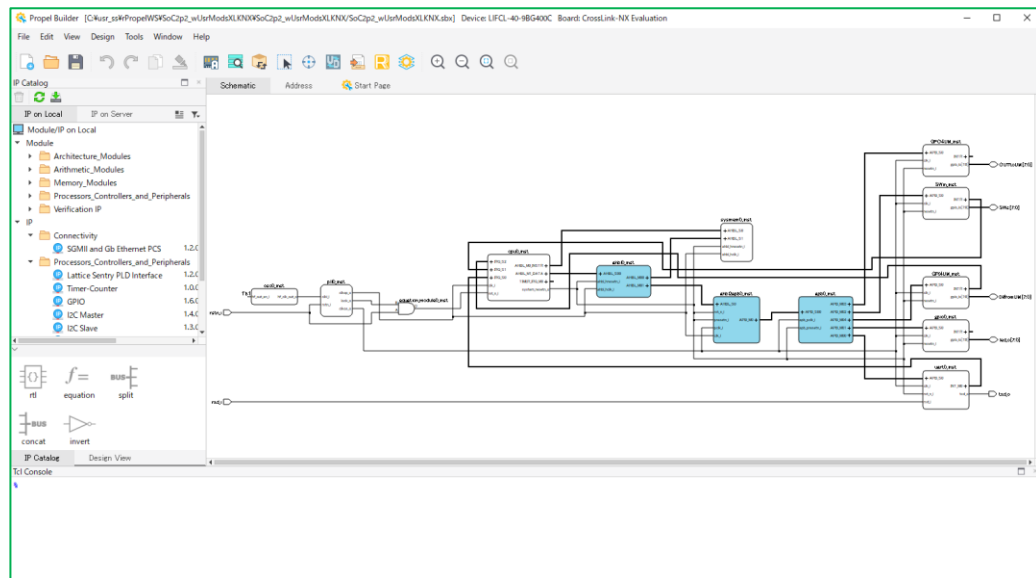
補足 8 : SoC プロジェクトのインポート (2)

- 問題なくインポートされると「Project Explorer」枠に SoC プロジェクトとサブフォルダー/ファイル一式が表示されます (下左)

☞ Propel SDK の Project Explorer にリストされないと、それ以降の一貫性のある作業ができません



これ以降はプロジェクト名を選択 → 右クリック → Open Design In → Propel Builder を選択すると、Propel Builder が立ち上がり回路図が表示され、確認・編集作業が可能になります



補足 8 : SoC プロジェクトのインポート (3)

- インポートした SoC プロジェクトを再現するためには以下のポイントを押さえておきます
 - ✓ 事前に各コンポーネントの最新バージョンをインストールしておきます (“IP on Local” タブで表示されるバージョンが最新かどうかを “IP on Server” タブでの表示で確認しておきます)
 - ✓ 回路図で各コンポーネントをダブルクリックすると、問題がなければコンフィグレーション・ウィンドウが表示されます。エラーメッセージが表示されないことを確認します (バージョンを更新するかどうかの確認メッセージの場合は任意です)
- 上の確認・設定が全て問題なければ『Generate』アイコンをクリックして SoC プラットフォームを再生成します
 - ✓ 複数ある GPIO コンポーネントの “アドレス重複” のエラーになる場合は、手動でアドレスを再配置し直してから生成します
 - ✓ オリジナルの SoC プロジェクトのターゲットデバイス・ファミリーを変更することはできません
- この後、この SoC プラットフォームをターゲットとして、C/C++ プロジェクトでユーザー・アプリケーションを作成・ビルドすることができます
 - ✓ プラットフォーム設定を書き出したファイルの一つ “sys_env.xml” が C/C++ プロジェクト作成時に必要です
 - ✓ ビルドに成功すると Debug / Release フォルダ下にプログラムメモリー初期化ファイル *.mem が生成されます
- ビルド後、再び SoC ビルダーにもどり、プログラムメモリー初期化ファイルを指定します
 - ✓ プログラムメモリー (system_inst0) コンポーネントをダブルクリックすると表示される設定ウィンドウで、「Initialize Memory」カラムのボックスをチェックし、「File Format」を “hex” にし、「Initialization File」セルにはビルドし直したソフトウェア・プロジェクト下 Debug/Release フォルダにブラウスし、当該 *.mem ファイルを指定します
 - ✓ その後、DRC → Generate を再度行います

補足9 : C/C++ プロジェクトのインポート

- 既存の C/C++ (SDK) プロジェクトを複製して再利用する場合は、インポート・メニューから行います
 1. File → Import... を選択
 2. 立ち上がる “Import” ウィンドウで General・Existing Projects into Workspace を選択
 3. 『Select root directory』 をブラウズしてインポートする C/C++ プロジェクト・フォルダを指定します
 4. Projects 枠に有効なプロジェクト名が表示されますので、チェックされていることを確認して 『Finish』
 5. SDK の Project Explorer 窓枠内にインポート元と同じ名称の C/C++ プロジェクトが複製されてインポートされます

この図は、Eclipse IDEで既存のC/C++プロジェクトをインポートする手順を示しています。

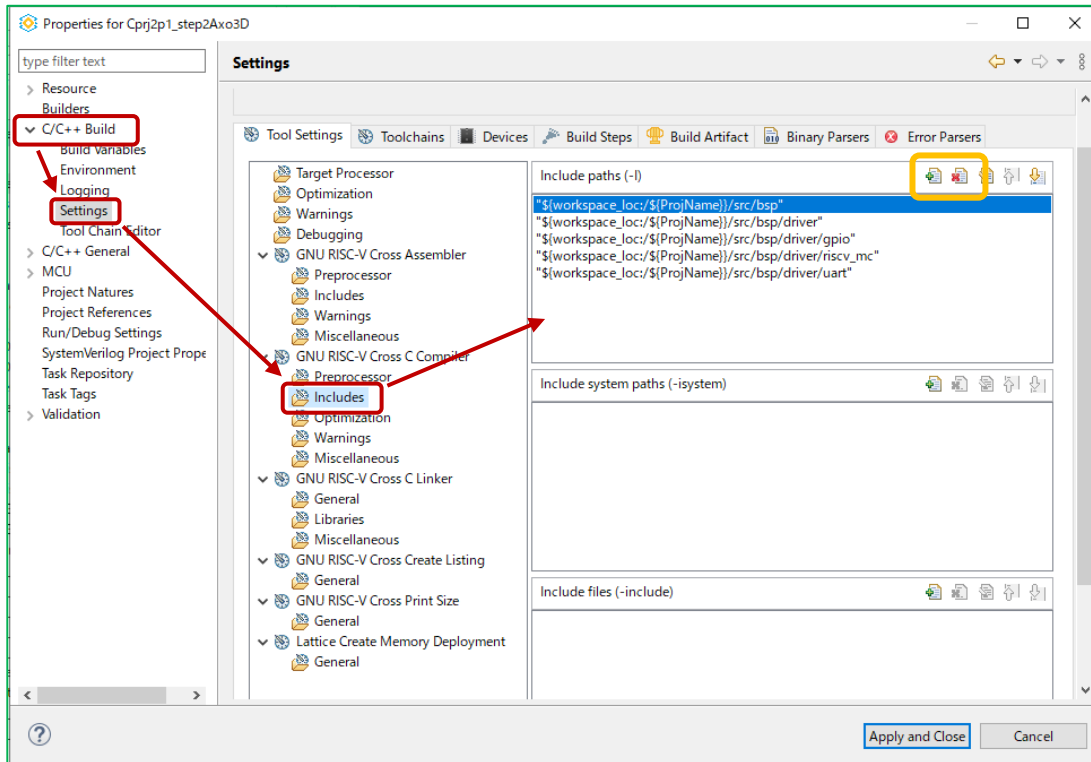
1. **Import** ウィンドウの **Select** タブで、**Existing Projects into Workspace** を選択します。**クリック**

2. **Import Projects** ウィンドウで、**Select root directory** を **ブラウズして指定** し、プロジェクト名を確認します。**確認**

3. **Project Explorer** で、インポートした C/C++ プロジェクトの表示例を確認します。**インポートした C/C++ プロジェクト表示例**

補足10 : SDK ビルド設定 ~ インクルード・ディレクトリ

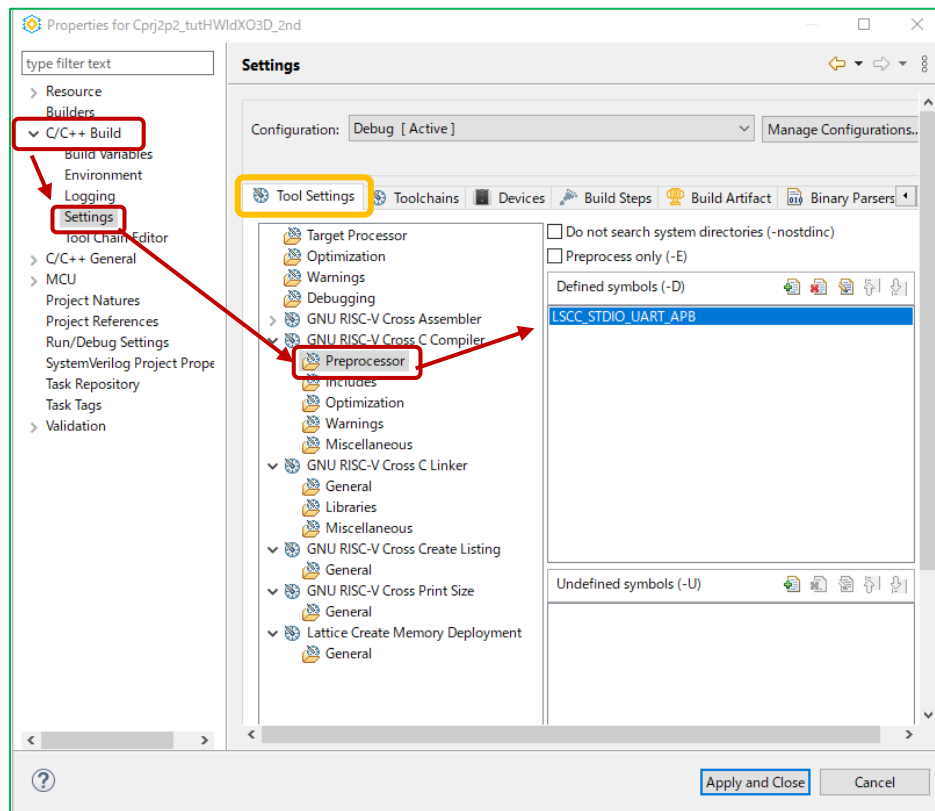
- C/C++ プロジェクトを選択して File・Properties... を選択します
- インクルード・ディレクトリの指定（既存ソースファイルを再利用する場合など）
 - ✓ C/C++ Build → Settings → GNU RISC-V Cross C Compiler → Includes



追加・削除時にクリックして
適宜操作・編集

補足10 : SDK ビルド設定 ~ プリプロセッサ

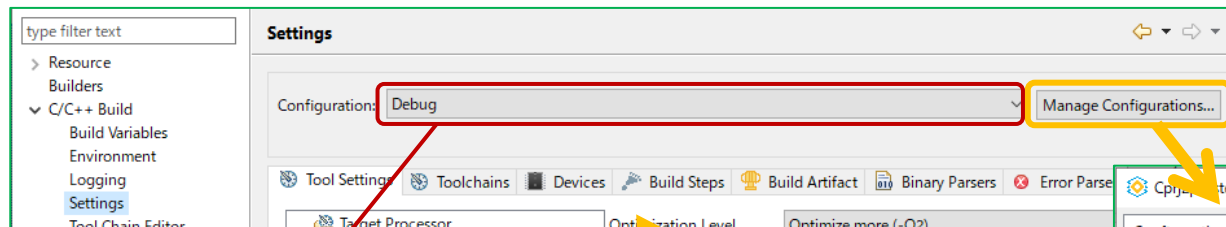
- 条件付きビルドなどの用途にユーザー定義のシンボルをここに記述しておきます
 - ✓ C/C++ Build → Settings → GNU RISC-V Cross C Compiler → Preprocessor



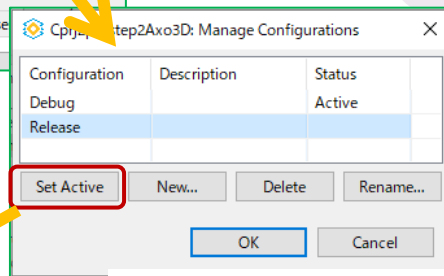
ビルド時に参照する
ユーザー定義のシンボル

補足10 : SDK ビルド設定 ~ Debug vs. Release

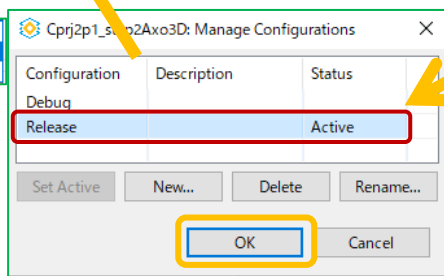
- ビルド・コンフィグレーションの変更ステップを示します (C/C++ Build → Settings)
 - ✓ Debug → Release を示す (逆も全く同様の操作。⑦~⑨ を忘れないようにします)



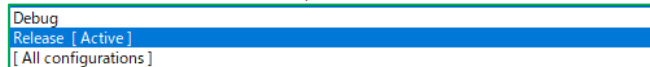
① クリック



- ② 非アクティブ側の行をクリック
- ③ 『Set Active』をクリック
- ④ 変更した側の Status が Active になるので『OK』をクリック

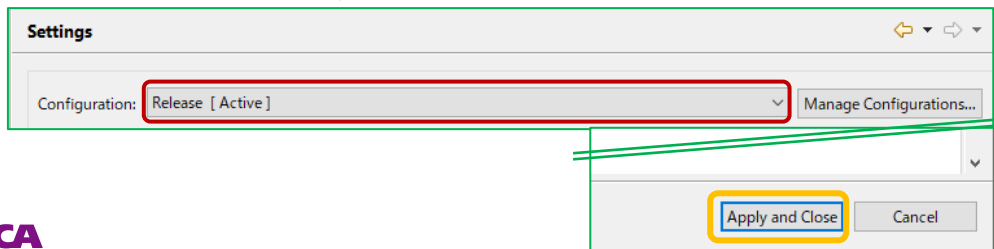


⑤ 『OK』で元に戻る



⑥ バーをクリック

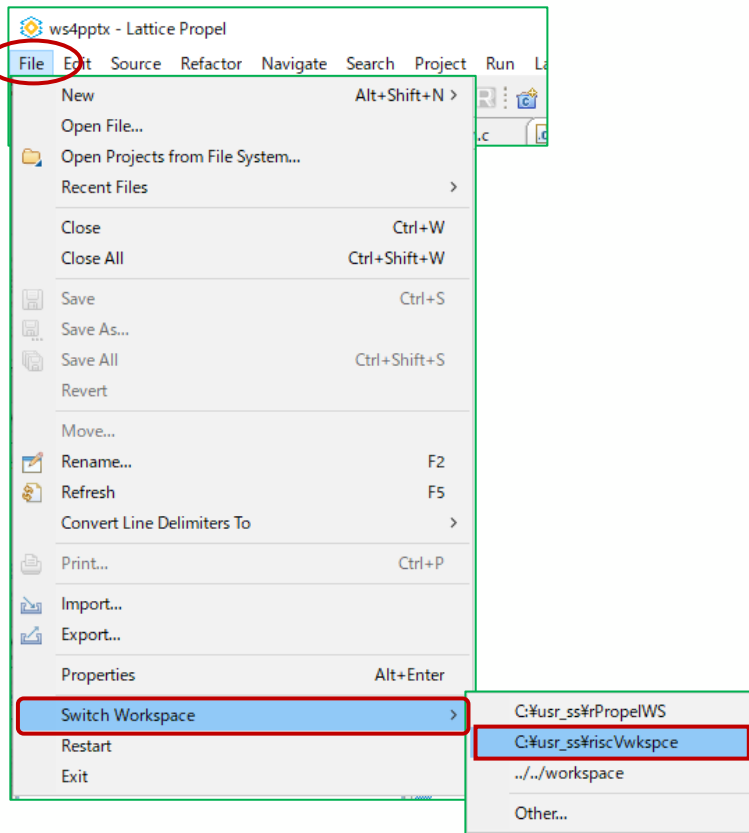
⑦ 変更した側が Active になっているので、それをクリック



- ⑧ バー表示が "[Active]" 表示で意図するビルドになっていることを確認
- ⑨ 右下の『Apply and Close』をクリック

補足11 : Propel 起動後のワークスペースの切り替え

- 起動時に選択する workspace (p.9) は起動後でも SDK GUI から切り替えができます
 - ✓ File → Switch Workspace → (選択)



切り替え操作の例

更新履歴

Date	Rev.	New Page#	Contents	By
2024/10/22	1.0	-- (p.20) (p.37) (pp.51-53) (p.54)	旧版用から Propel 2024.1 用にスクリーンショットなど全般的に更新 * Builder、配線時の操作方法について追加 * OpenOCD Debugger のエラー例と解消方法を追加 * 補足 “内部バス引き出しでユーザー回路と I/F” を追加 * 補足 “SoC/C プロジェクトのインポート” を追加	S.S.
2024/10/28	1.1	p.9, 11, 20, 29	フィードバックを反映してマイナーな更新	S.S.