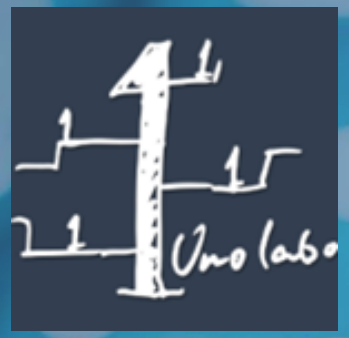


株式会社ウーノラボ

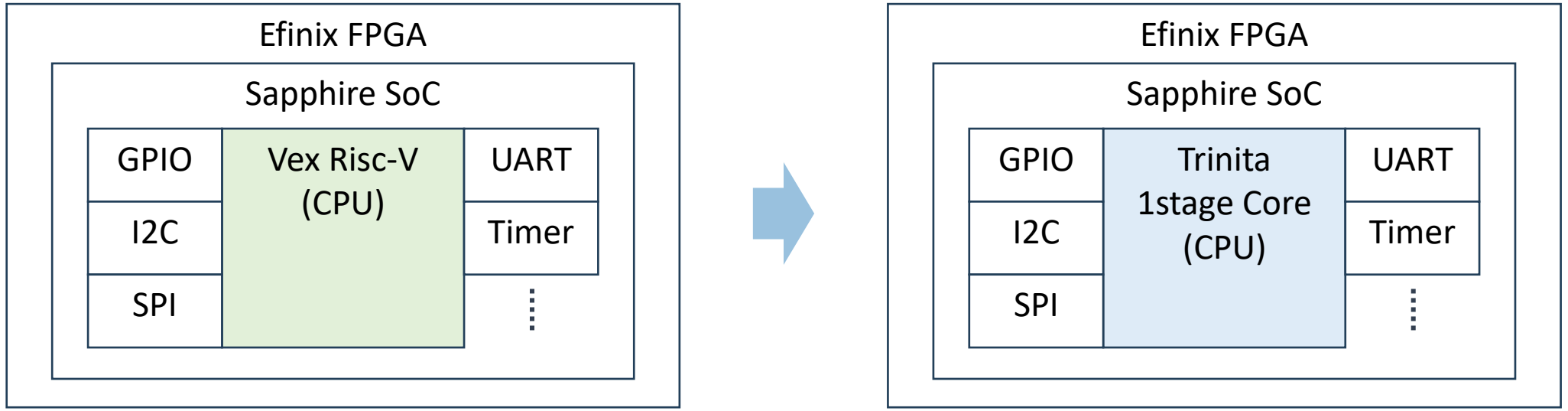
https://www.unolabo.co.jp

Efinix FPGA への Trinita 1stage Core の実装手順

Rev.4 2025.04.09

概要

Efinix FPGA に実装されている Sapphire SoC 内部の Vex RISC-V を Trinita 1stage Core に置き換える手順を説明します。Sapphire SoC の Peripheral はそのまま使えるため、置き換え後も Sapphire SoC 用のソフトウェアコードをそのまま流用できます。



置き換え作業の流れ

テンプレートコピー

Efinity で Sapphire SoC プロジェクトを作る

Sapphire SoC の VexRISC-V を 1stage Core に置換 (スクリプト)

1stage Core で使うクロックを PLL で生成・タイミング制約追加

1stage Core のパラメータを定義 (.vh ファイル)

トップデザインのポート宣言にクロックを追加し、.vh をインクルード

ソフトウェアをビルドし、RAM 初期化ファイル生成 (RISC-V IDE とスクリプト)

FPGA デザインをコンパイル (Efinity)

オンライン資料

この資料で紹介する手順は GitHub で公開されています。
チュートリアル動画も YouTube で公開していますので、合わせて参照下さい。

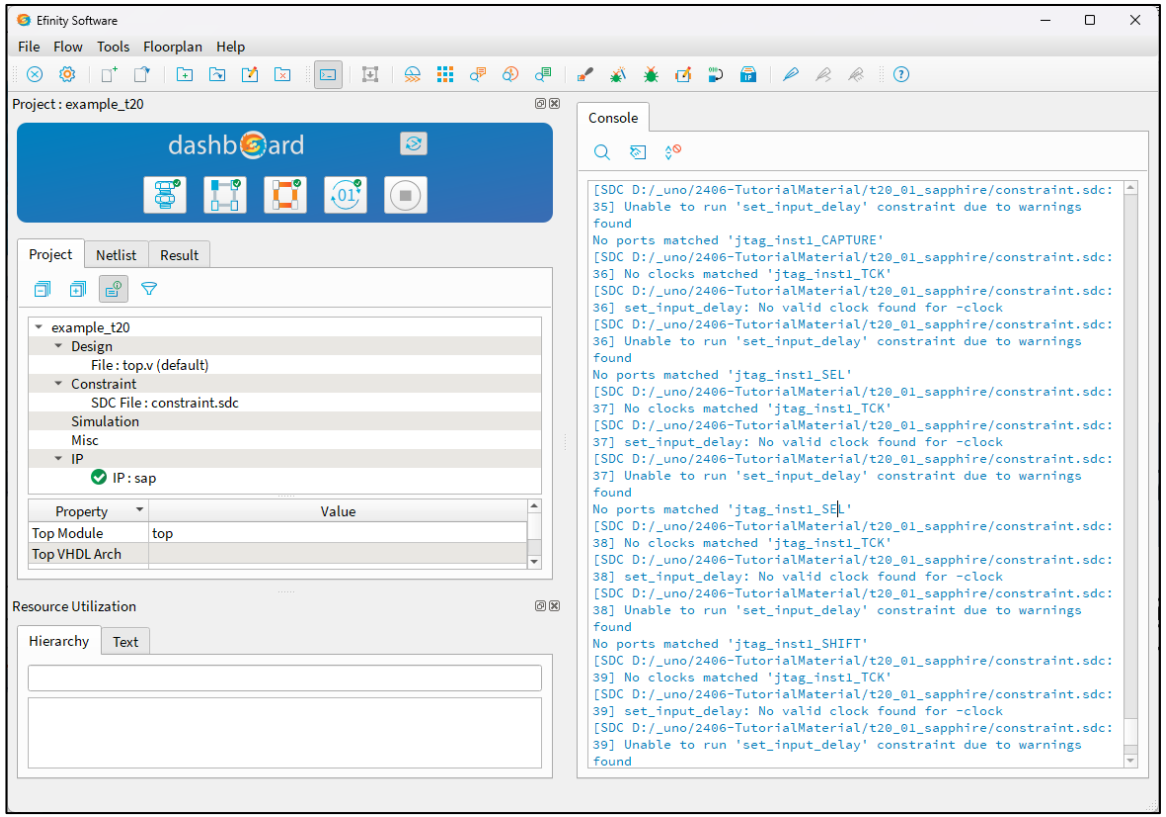
https://github.com/unolabo/efx-trinita-exa

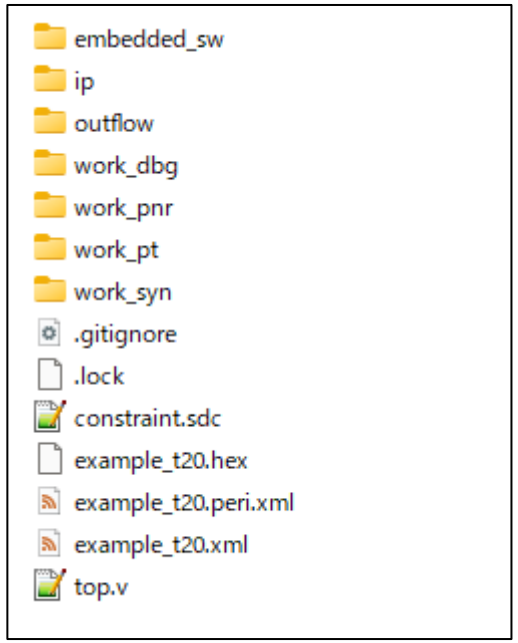
https://www.youtube.com/watch?v=nwmNSFLWMqQ

置き換え手順

Efinity プロジェクトの作成

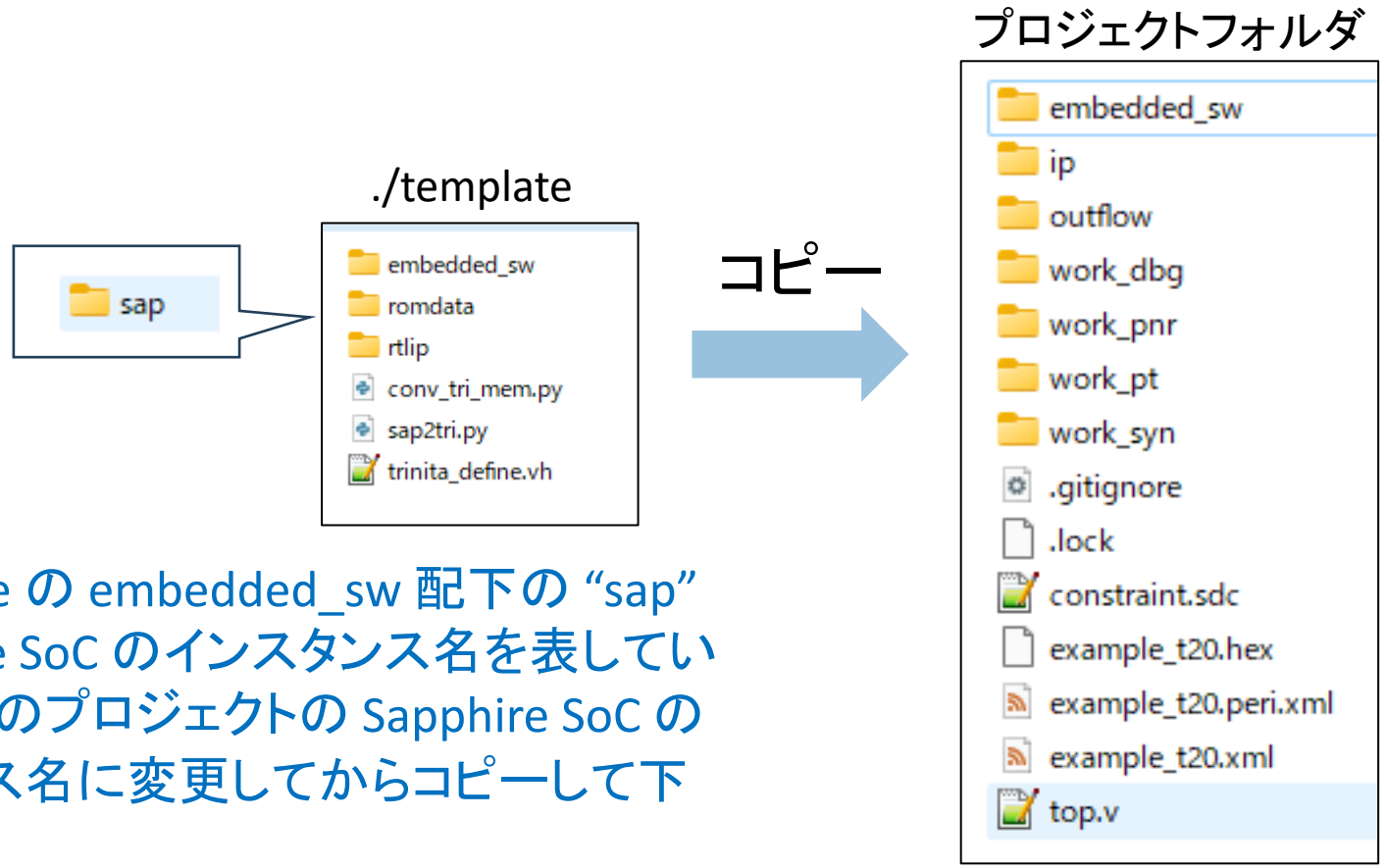
置き換え作業を始める前に、Sapphire SoC を実装済みのプロジェクトを用意します。
※SoC の Peripheral 構成をこの時点で決定することを推奨します。





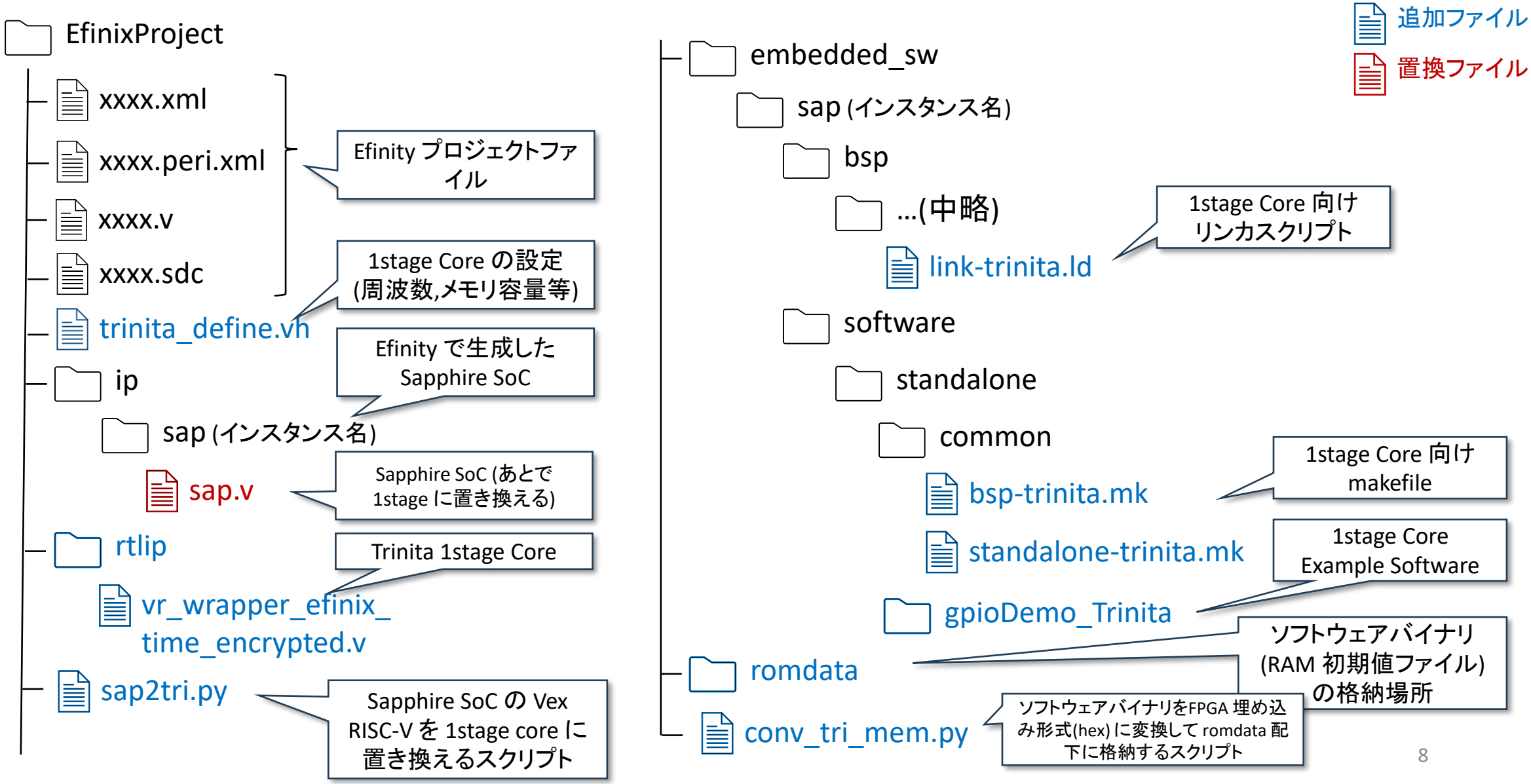
テンプレートをプロジェクトフォルダにコピー

./template フォルダの内容をプロジェクトフォルダにコピーします。



※ template の embedded_sw 配下の “sap” は Sapphire SoC のインスタンス名を表しています。所望のプロジェクトの Sapphire SoC のインスタンス名に変更してからコピーして下さい。

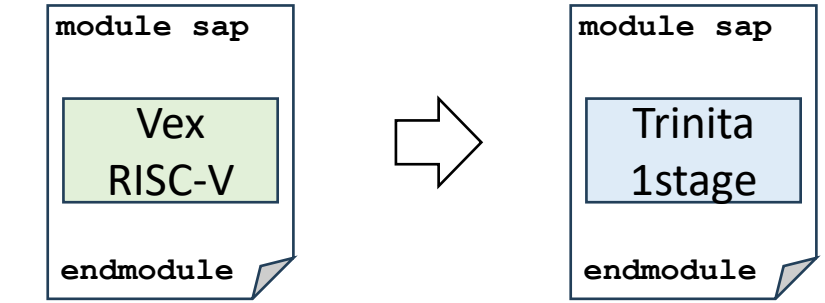
追加されるファイル (テンプレートからコピー)



Vex RISC-V を Trinita 1stage Core に置き換える

コマンドプロンプトで Efinity プロジェクトのフォルダに移動し、Python スクリプトを実行します。

これで Trinita 1stage Core が sap.v に実装されます。



置き換え前の sap.v　　置き換え後の sap.v

```
>python sap2tri.py ./ip/sap/sap.v
```

PLL の設定 T20 / Ti60 の場合

Trinita 1stage Core では位相をずらしたクロックを必要とします。
Efinity の Interface Designer で下記のように PLL を設定します。

- Trion T20 最大 25 MHz
  - io_systemClk : 0 deg
  - io_systemClk2 : 270 deg (io_systemClk3 の反転クロック)
  - io_systemClk3 : 90 deg

- Titanium Ti60 最大 75MHz
  - io_systemClk : 0 deg
  - io_systemClk2 : 225 deg
  - io_systemClk3 : 90 deg

(参考) PLL の設定 T8 の場合

T8 の PLL では、位相をずらした 12.5 MHz のクロックを生成できないため、ロジック領域で分周・位相設定してクロックを生成します。

- Trion T8 最大 12.5 MHz
  - io_systemClk : 0 deg
  - io_systemClk2 : 252 deg
  - io_systemClk3 : 108 deg

タイミング制約

```
create_clock -period 8 CLK125M
create_clock -name io_systemClk  -period 80 -waveform {  0 40} io_systemClk
create_clock -name io_systemClk2 -period 80 -waveform { 56 16} io_systemClk2
create_clock -name io_systemClk3 -period 80 -waveform { 24 64} io_systemClk3
```

```
114
115     reg  io_systemClk;
116     reg  io_systemClk2;
117     reg  io_systemClk3;
118     reg [9:0] cntdiv;
119
120     (* syn_preserve = "true" *) reg tri_clk_1;
121     (* syn_preserve = "true" *) reg tri_clk_2;
122     (* syn_preserve = "true" *) reg tri_clk_3;
123
124     always@(posedge CLK125M)
125     begin
126       if (~io_pllResetn)
127         cntdiv <= 10'b0000011111;
128       else
129         cntdiv <= {cntdiv[8:0], cntdiv[9]};
130     end
131
132     always@(posedge CLK125M)
133     begin
134         io_systemClk <= cntdiv[0];
135         io_systemClk2 <= cntdiv[7];
136         io_systemClk3 <= cntdiv[3];
137     end
138
```

10分周

位相設定

タイミング制約の設定

PLL で追加したクロックを .sdc で定義します。

T20 では io_systemClk3 を LogicBlock で not 反転して io_systemClk2 を生成しています。
そのため、下記の例では io_systemClk2 を定義していません。
※Efinity では not 反転クロックは .sdc で定義しなくても自動的にタイミング解析されるためです。

```
# PLL Constraints
##################
create_clock -period 40.0000 io_systemClk
create_clock -waveform {10.0000 30.0000} -period 40.0000 io_systemClk3
```

パラメータ(マクロ)の設定

Trinita 1stage Core のパラメータ(マクロ)を .vh ファイルで定義します。

```
`define EFINIX 1
`define FREQ 25
`define START_ADDRESS 32'hF9000000
`define IMEM_AWIDTH 15
`define DMEM_AWIDTH 15
`define FILE_IMEM  "./romdata/imem.hex"
`define FILE_IMEM0 "./romdata/imem0.hex"
`define FILE_IMEM1 "./romdata/imem1.hex"
`define FILE_IMEM2 "./romdata/imem2.hex"
`define FILE_IMEM3 "./romdata/imem3.hex"
`define FILE_DMEM  "./romdata/dmem.hex"
`define FILE_DMEM0 "./romdata/dmem0.hex"
`define FILE_DMEM1 "./romdata/dmem1.hex"
`define FILE_DMEM2 "./romdata/dmem2.hex"
`define FILE_DMEM3 "./romdata/dmem3.hex"
`define TRION 1
`define SAPPHIRE 1
```

動作周波数 [MHz]

IMEM の深さ 2^15=32KB

DMEM の深さ 2^15=32KB

Titanium FPGA の場合は、
`define TITANIUM 1 に変更する

トップデザインの変更(1)

先ほど PLL で追加した io_systemClk2, 3 をトップデザインの port 宣言に追加し、Sapphire SoC のインスタンスにも接続します。

```
module top
(
    input  io_asyncResetn,
    input  io_systemClk,
    //input  io_systemClk2,
    input  io_systemClk3,
    output io_pllResetn,
    input  io_pllLocked,
```

```
assign io_systemClk2 = ~io_systemClk3;
```

```
    sap u_sap(
    .io_systemClk ( io_systemClk ),
    .io_systemClk2 ( io_systemClk2 ),
    .io_systemClk3 ( io_systemClk3 ),
    .jtagCtrl_enable ( jtagCtrl_enable ),
    .jtagCtrl_tdi ( jtagCtrl_tdi ),
    .jtagCtrl_capture ( jtagCtrl_capture ),
    .jtagCtrl_shift ( jtagCtrl_shift ),
    .jtagCtrl_update ( jtagCtrl_update ),
    .jtagCtrl_reset ( jtagCtrl_reset ),
    .jtagCtrl_tdo ( jtagCtrl_tdo ),
    .jtagCtrl_tck ( jtagCtrl_tck ),
    .system_spi_0_io_data_0_read ( system_spi_0_io_dat
```

T20 では io_systemClk3 を LogicBlock で not 反転して io_systemClk2 を生成しています。
そのため、top のポート宣言で io_systemClk2 がコメントアウトされています。

トップデザインの変更(2)

先ほど定義したパラメータ (.vh ファイル) をプロジェクトのトップデザインにインクルードします。下記のように `include 文を追加して下さい。

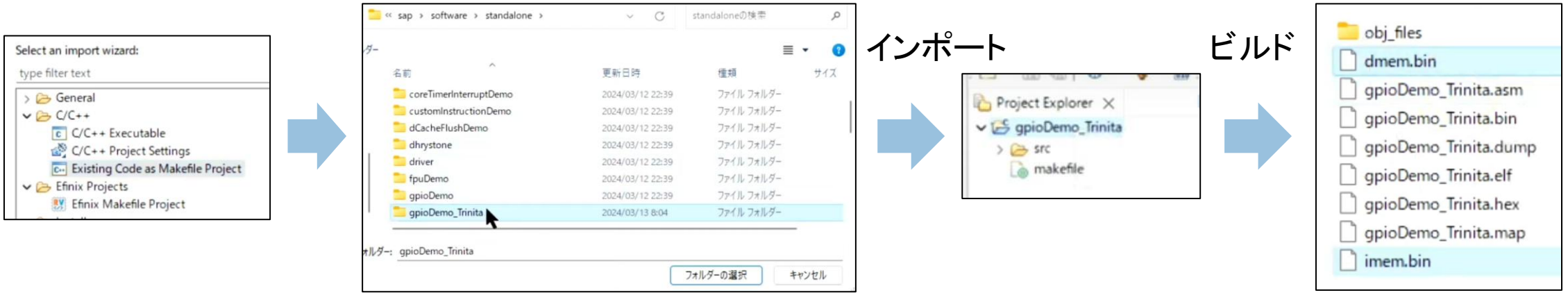
```
    output jtagCtrl_tdo,
    input  jtagCtrl_tck
);

    `include "trinita_define.vh"

    wire io_systemReset;
```

ソフトウェアのビルド

RISC-V IDE で ./embedded_sw/sap をワークスペースとして開きます。
サンプルソフトウェアである “gpioDemo_Trinita” プロジェクトをインポートします。
このプロジェクトをビルドすると gpioDemo_Trinita/build 配下に dmem.bin と imem.bin が生成されます。



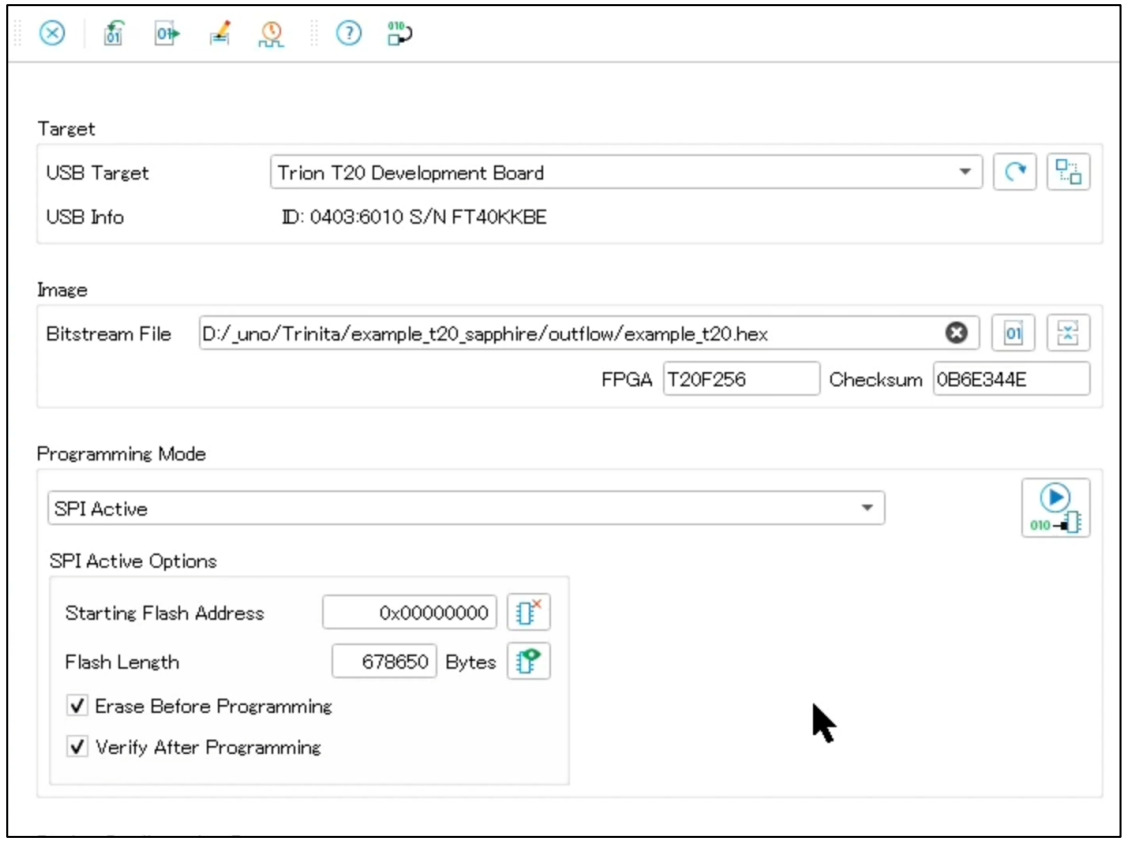
ソフトウェアの RAM 初期値データ化

コマンドプロンプトで Efinity プロジェクトのフォルダに移動し、Python スクリプトを実行します。
このスクリプトによって、先程生成した imem.bin と dmem.bin が FPGA の Block RAM に埋め込まれます。

```
>python conv_tri_mem.py ./embedded_sw/sap/software/standalone/gpioDemo_Trinita/build
```

Efinity でコンパイル

DMEM / IMEM の生成が終わったら Efinity でプロジェクトをコンパイルします。
ビットストリームファイル(.hex)が生成されたら、これを Efinity Programmer でフラッシュメモリに書き込みます。



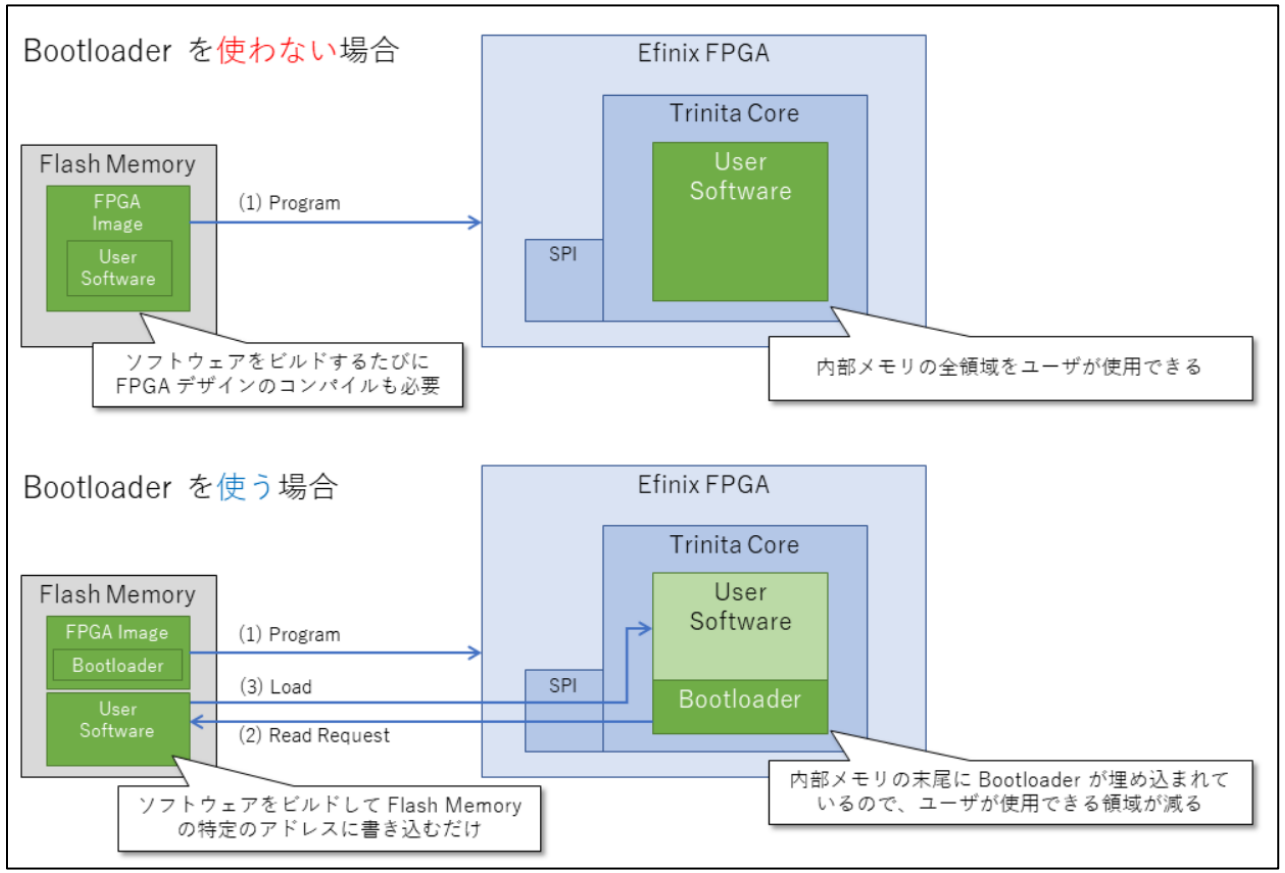
補足

Bootloader を使ったソフトウェアの起動

以上の手順で、ソフトウェアのバイナリデータを FPGA ビットストリームファイルに埋め込んだので、FPGA の電源起動の後、自動的にソフトウェアも起動します。

もしソフトウェアを外部フラッシュメモリに格納して、Bootloader でソフトウェアを起動させたい場合は、下記の GitHub のページを参照して下さい。

https://github.com/unolabo/efx-trinita-exa/blob/main/README_Bootloader.md



株式会社ウーノラボ

https://www.unolabo.co.jp

ありがとうございました