



株式会社ウーノラボ

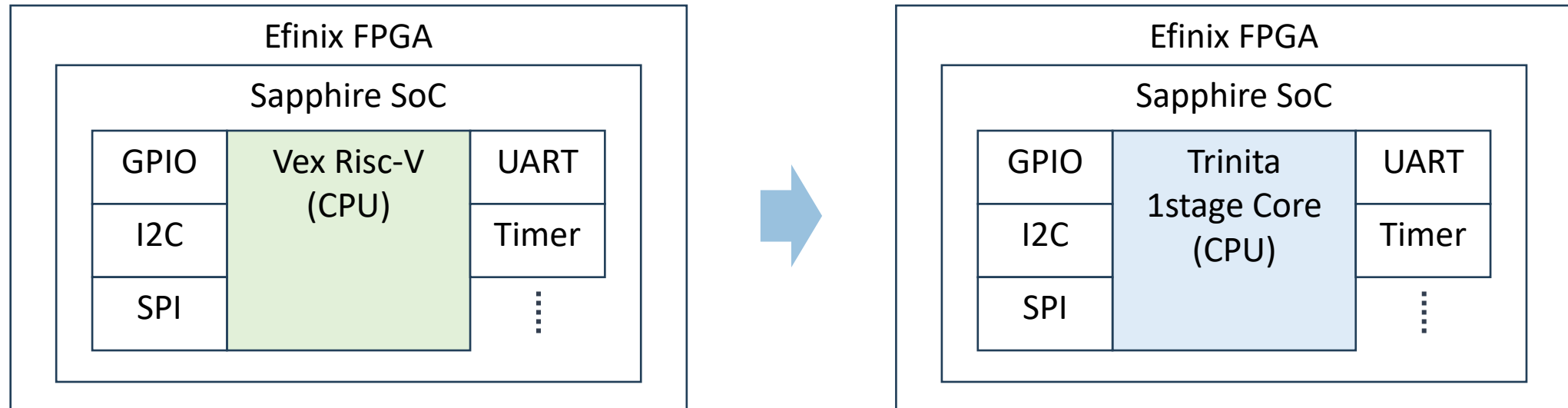
<https://www.unolabo.co.jp>

Efinix FPGA への Trinita 1stage Core の実装手順

Rev.2 2025.02.18



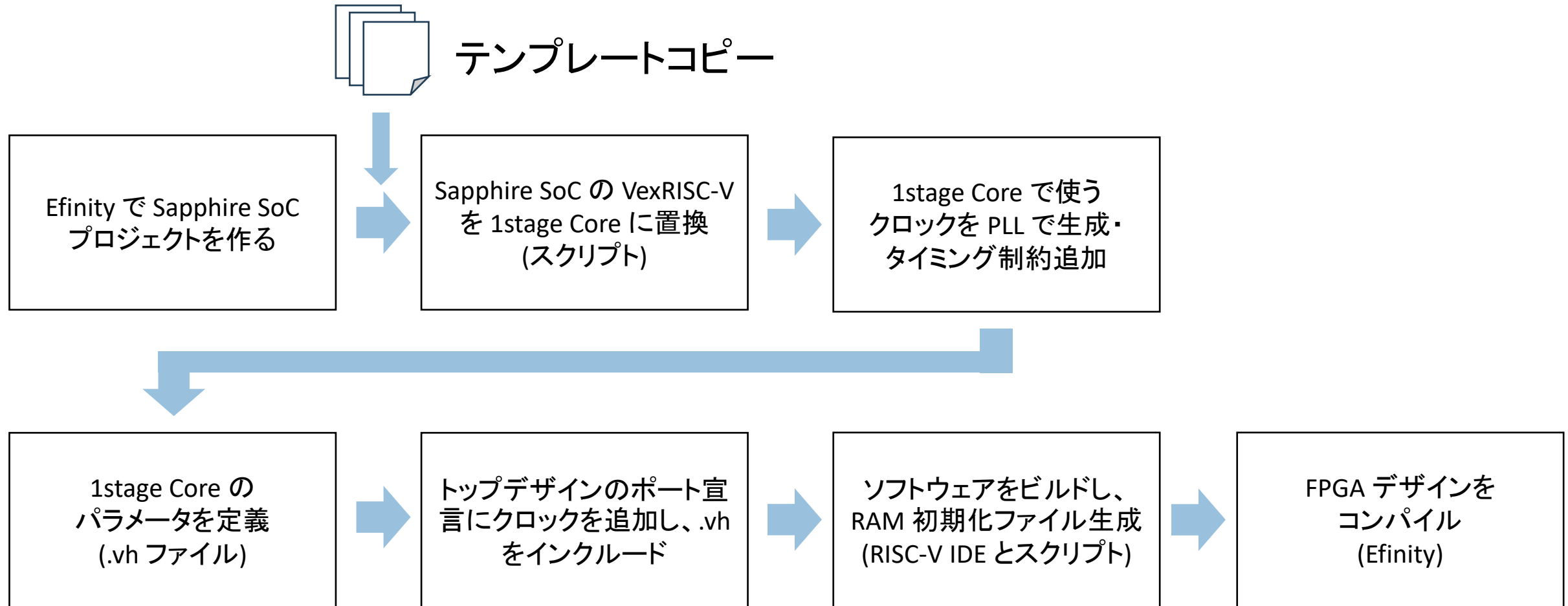
Efinix FPGA に実装されている Sapphire SoC 内部の Vex RISC-V を Trinita 1stage Core に置き換える手順を説明します。Sapphire SoC の Peripheral はそのまま使えるため、置き換え後も Sapphire SoC 用のソフトウェアコードをそのまま流用できます。





置き換え作業の流れ

© UNO Laboratories, Ltd.





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。

README License

Trinita 1stage Core 実装 <https://github.com/unolabo/efx-trinita-exa>

※ この手順は、下記の前提条件で説明します。

- 既存デザインに Sapphire SoC が
- Sapphire SoC のインスタンス名が sap である

※ 置き換え手順は YouTube でも公開しています。合わせて参照下さい。



1. テンプレートをコピーする

- template フォルダに格納されているファイル・フォルダを、プロジェクトフォルダにコピーします。
- Sapphire SoC のインスタンス名が sap 以外である場合、template/embedded_sw/sap 配下のフォルダを、お使いのインスタンス名のフォルダ配下にコピーしてください。

2. Sapphire SoC ソースコードの VexRiscV コアの Trinita コアに置き換える

- template
- ./ip/sap フォルダの sap.v を ./convtrinita フォルダにコピーします。
- コマンドプロンプトを開き ./convtrinita フォルダに移動します。
- 下記コマンドを実行します。このコマンドによって、sap.v の VexRiscV コアが Trinita コアに置き換わりま

Premium JP

検索 <https://www.youtube.com/watch?v=6zQXcA6hQyI>

GitHub efx-trinita-exa リポジトリ



Trinita 1stage Core

注意

- 本ソースコードの使用にあたり **ライセンス** に関わる必要があります。
- 本リポジトリの一部または全部を Clone / Download した時点で、ライセンス条項に同意したものとみなします。
- 本ドキュメントの pdf 版は [こちら](#) です。
- ソフトウェアを Flash Memory に格納し、Bootloader を使用して起動する方法は下記ドキュメントを参照してください。
[Bootloader によるソフトウェアの起動](#)

概要

このコアは Efinx FPGA Sapphire SoC 向けの 1 ステージ RISC-V コアです。※2024年3月現在 1 ステージコアの性能向上にともない、2 ステージコアは提供終了となりました。

Efinx Sapphire SoC の Vex RISC-V コアを差し替えています。Efinx のコンパイラ/ライブラリ (Verilog/ELC マクロ) 定義によって、下記のオプションを使用できます。

- CPU レジスタ実装方法: Block RAM または FF の選択
- Compressed 命令セット: Embedded 命令セットへの対応 (7B のみ)



Sapphire SoC の RISC-V コアを Trinita コアに置き換える手順 2024年3月版

株式会社ウーノラボ チャンネル登録

高評価 共有



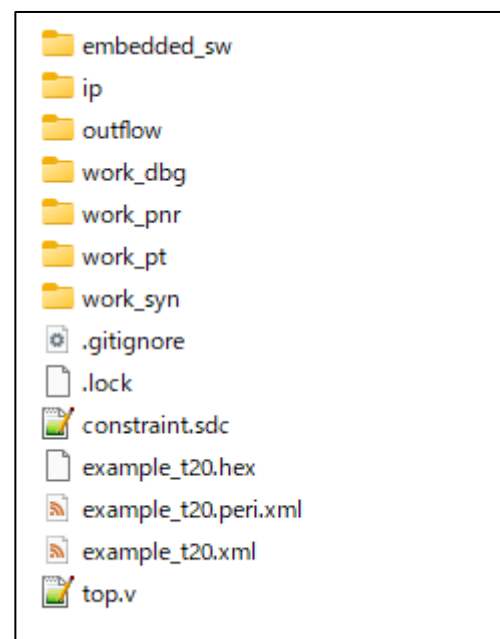
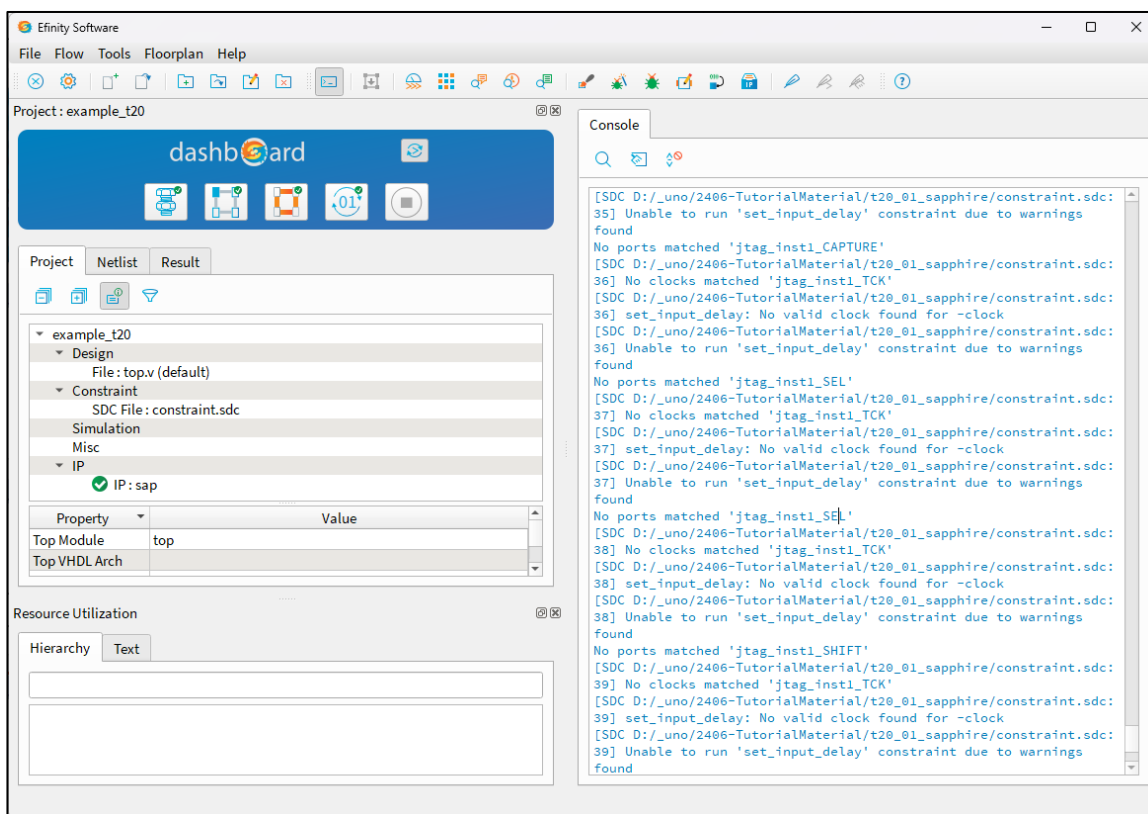
置き換え手順



Efinity プロジェクトの作成

© UNO Laboratories, Ltd.

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。

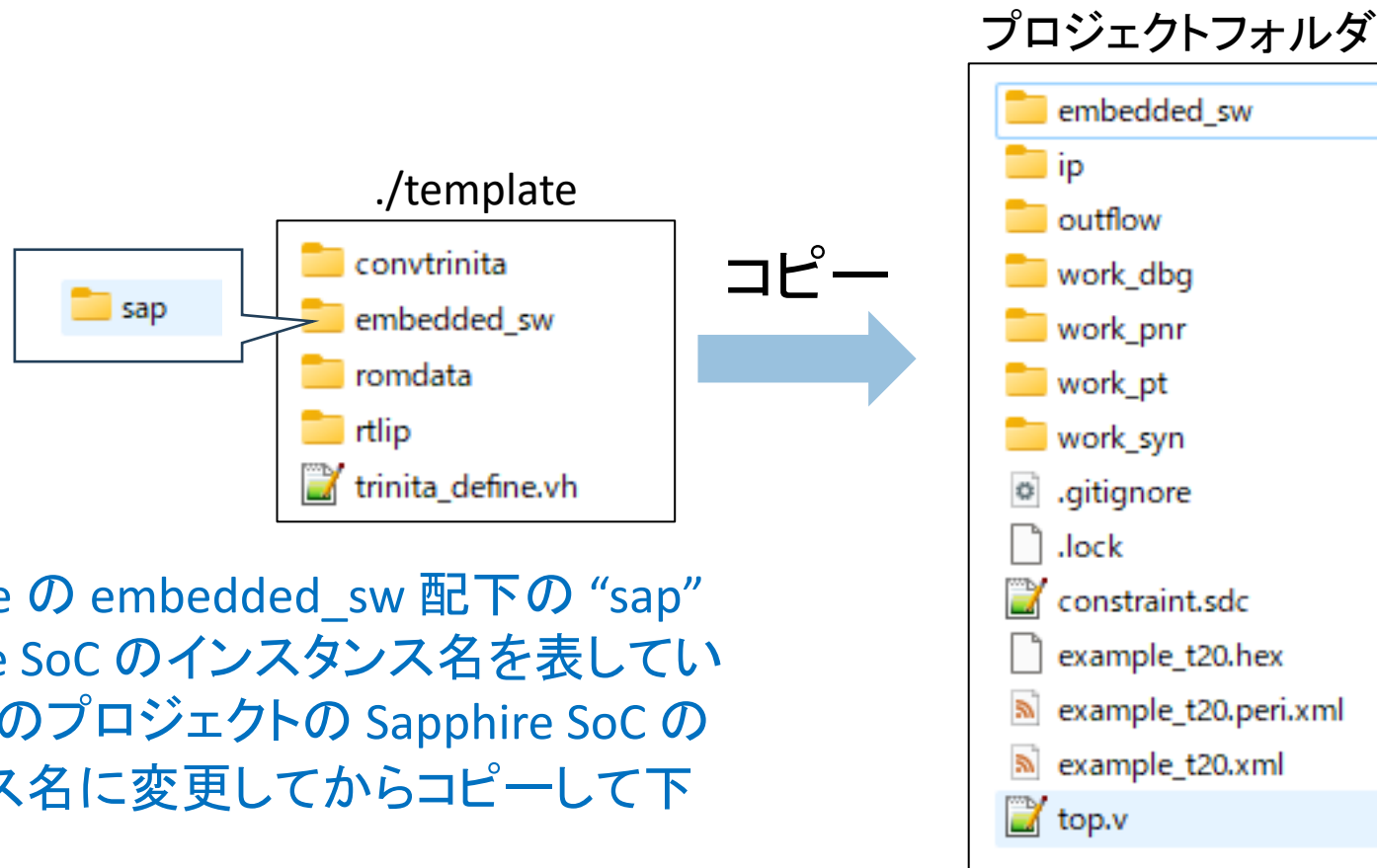




テンプレートをプロジェクトフォルダにコピー

© UNO Laboratories, Ltd.

./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。





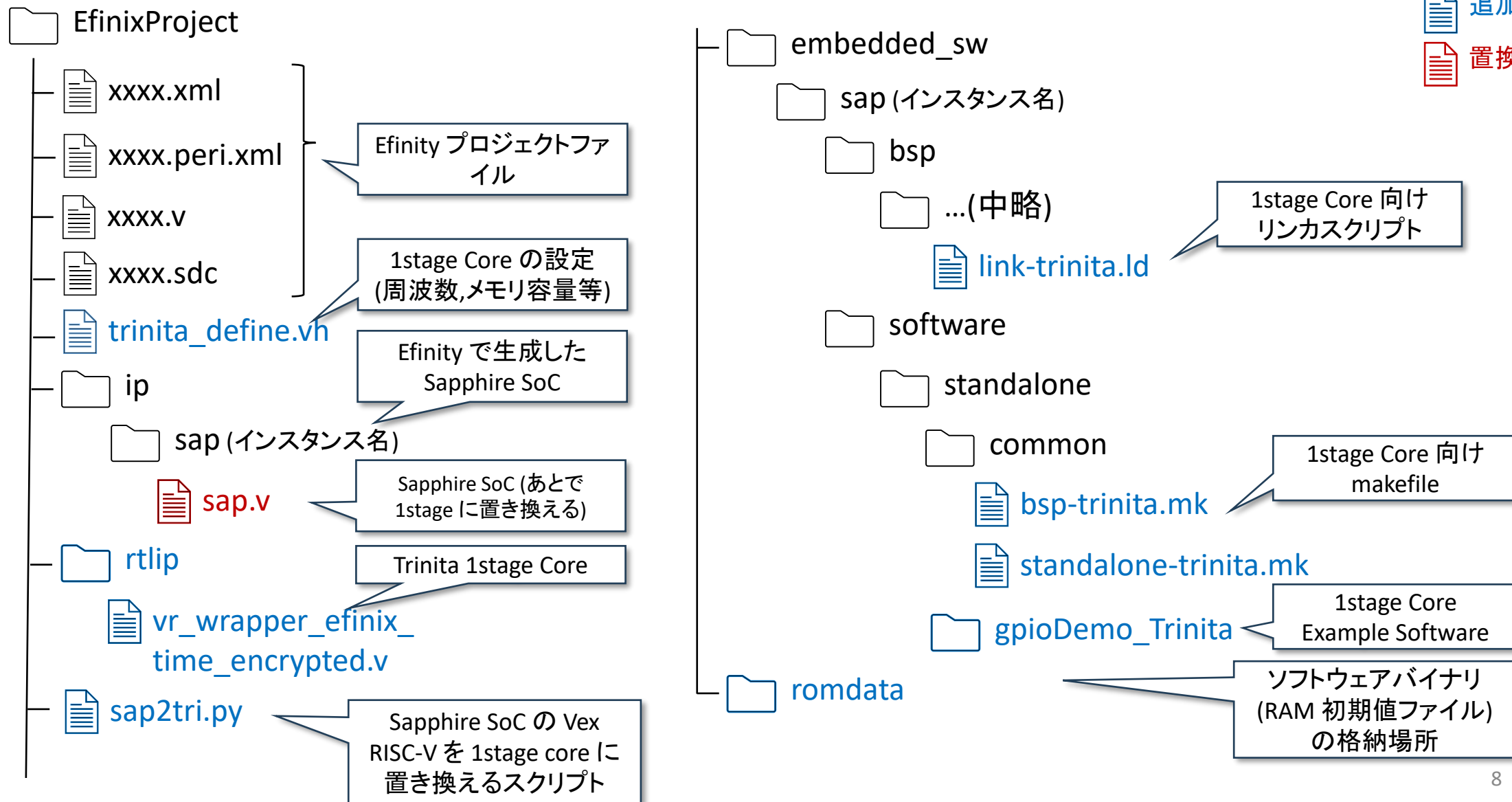
※ template の `embedded_sw` 配下の “sap” は Sapphire SoC のインスタンス名を表しています。所望のプロジェクトの Sapphire SoC のインスタンス名に変更してからコピーして下さい。



追加されるファイル (テンプレートからコピー)

© UNO Laboratories, Ltd.

 追加ファイル
 置換ファイル



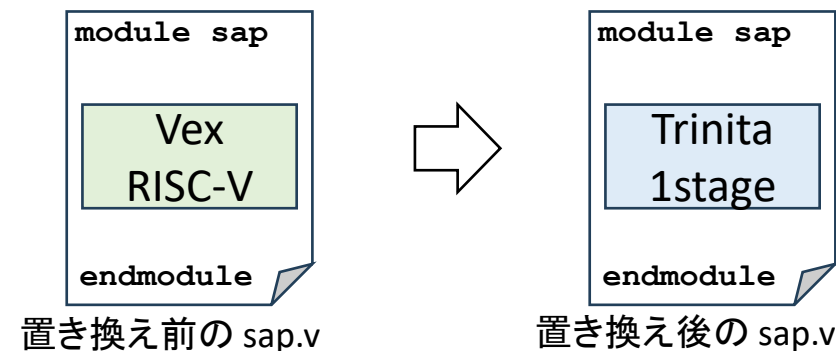


Vex RISC-V を Trinita 1stage Core に置き換える

© UNO Laboratories, Ltd.

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。



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



PLL の設定 T20 / Ti60 の場合

© UNO Laboratories, Ltd.

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

© UNO Laboratories, Ltd.

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

© UNO Laboratories, Ltd.

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 $2^{15}=32\text{KB}$

DMEM の深さ $2^{15}=32\text{KB}$

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)

© UNO Laboratories, Ltd.

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。

```
1 module top
2 (
3     input  io_asyncResetn,
4     input  io_systemClk,
5     //input io_systemClk2,
6     input  io_systemClk3,
7     output io_pllResetn,
8     input  io_pllLocked,
9 )
```

```
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_data_0_read )
)
```

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)

© UNO Laboratories, Ltd.

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。

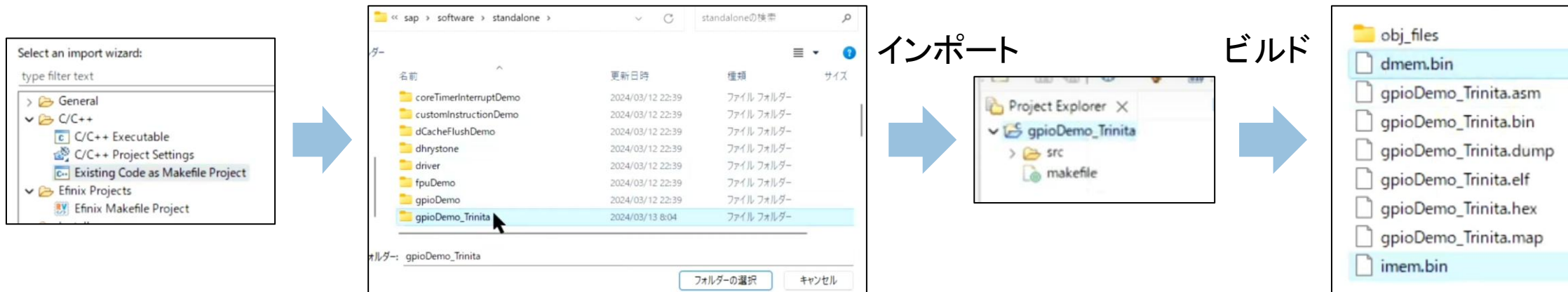
```
1      output jtagCtrl_tdo,  
2      input  jtagCtrl_tck  
3  );  
4  
5      `include "trinita_define.vh"  
6  
7      wire io_systemReset;
```



ソフトウェアのビルド

© UNO Laboratories, Ltd.

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

© UNO Laboratories, Ltd.

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。

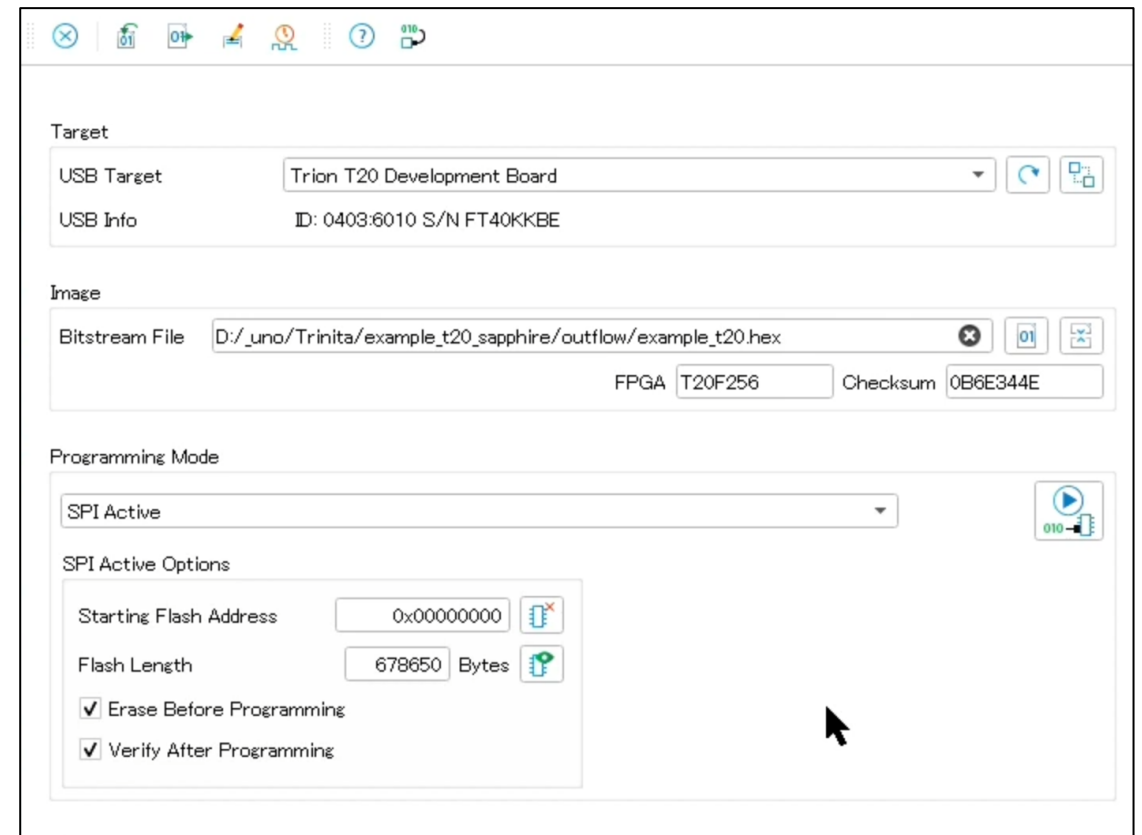
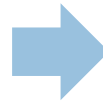
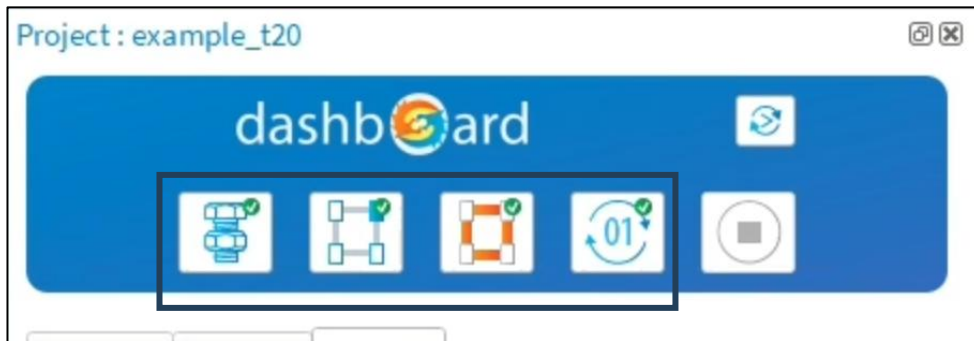
```
>python conv_tri_mem.py ./embedded_sw/sap/software/standalone/gpioDemo_Trinita/build
```



Efinity でコンパイル

© UNO Laboratories, Ltd.

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



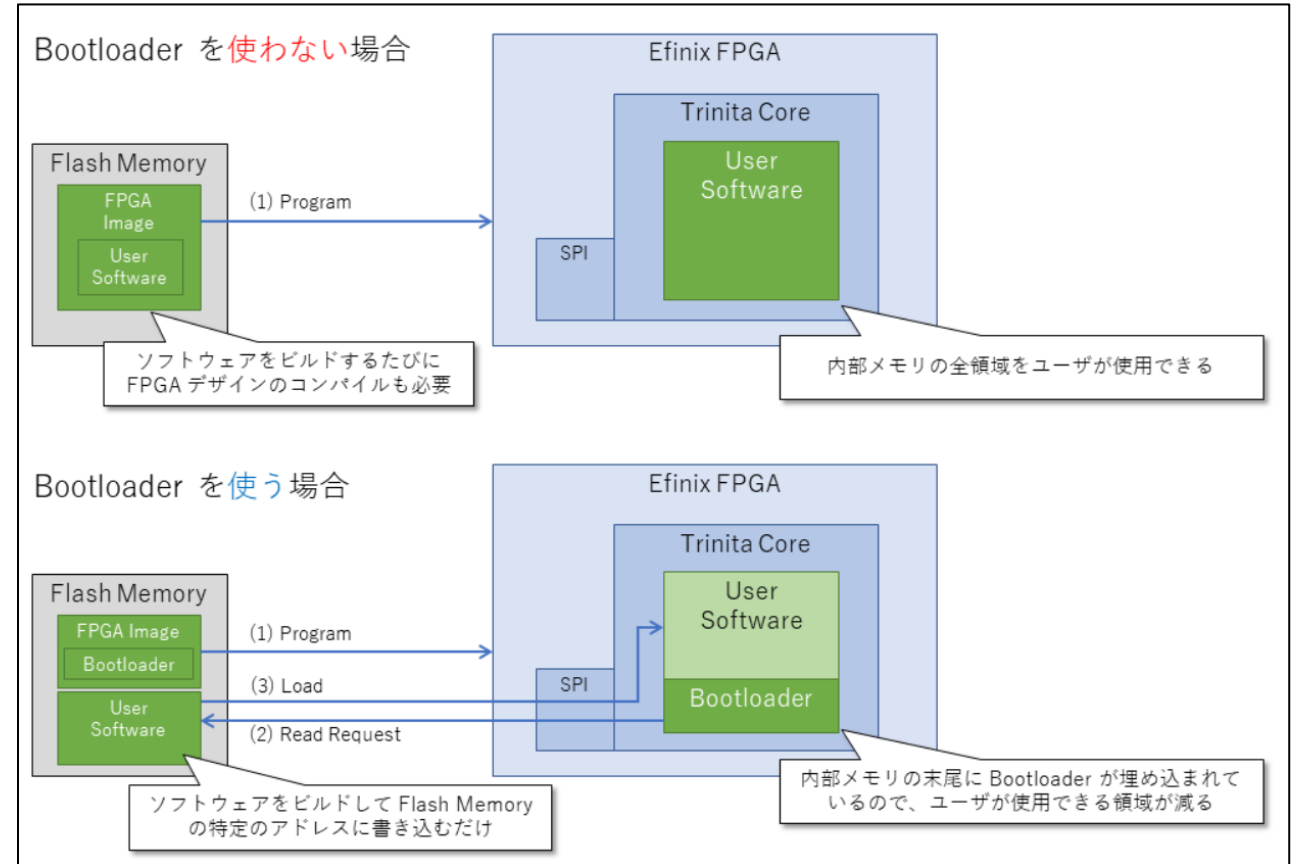
Bootloader を使ったソフトウェアの起動

© UNO Laboratories, Ltd.

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