

Efinity Debugger

(Logic Analyzer編)

株式会社レスターエレクトロニクス



EfinityのDebugger

Efinityにはハードウェアデバッガの機能が実装されており、FPGA内部の信号の状態を確認できます。

モニタの手法としてLogic AnalyzerとVirtual IOがあります。

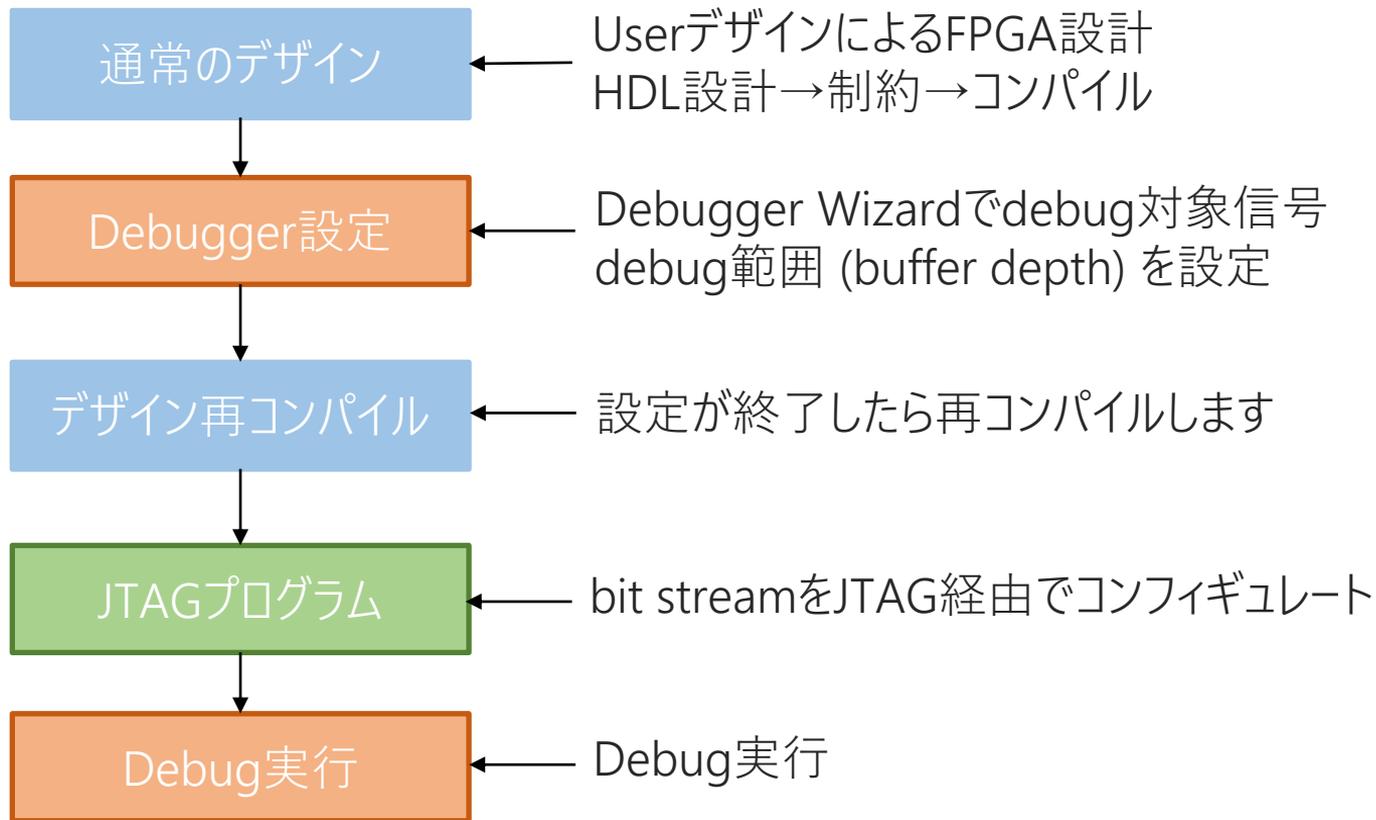
実現できること

- JTAG I/Fを用いたハードウェアデバッグ
- Virtual IOを用い信号のモニタとドライブ
- Logic Analyzerでトリガ条件を指定しての信号キャプチャ

本書ではハードウェアデバッガのLogic Analyzerに関して説明いたします



Debuggerの準備



Debugger Wizard - Logic Analyzer 準備



Open debug wizardアイコンをクリック

Logic Analyzerではdebugを行なう前にdebug wizardを用いて事前の準備ができます
debug wizardでは

- debug対象信号の指定
 - 観測領域（buffer depth）の定義
- などを設定できます

Debugger Wizard画面

Efinity Debug Wizard

① **Trigger and Storage Settings:**
Buffer Depth: 2048 Input Pipeline Stage: 2 Capture Control

② **Connection Settings**
JTAG USER TAP: USER1

③ **Select signals:**
Signals from: Elaborated Netlist
Filter:

Regular Expression Case Sensitive Filter Auto-Generated Nets Hierachy View

④

Name	Width	Clock Domain
knight_led		
└ STOP	1	Undefined
└ state	4	CLK
└ skip_cnt	27	CLK
└ RST_X	1	Undefined
└ LED	8	CLK
└ CLK	1	Undefined

>>

Name	Width	Clock Domain	Probe Type
state	4	CLK	DATA AND TRIGGER
skip_cnt	27	CLK	DATA AND TRIGGER
LED	8	CLK	DATA AND TRIGGER

⑤

<<

Next Cancel

Debugger Wizard ①項目設定

➤ ①赤枠

- Buffer Depth - 観測量（バッファの深さ）をプルダウンリストから選択

※ 内蔵RAMを消費します

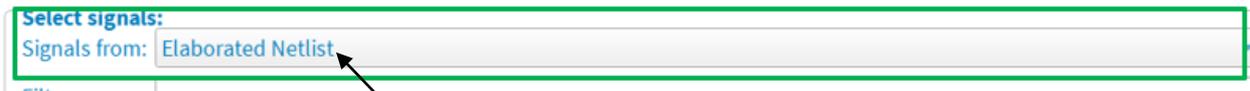


- Capture Control - checkすればcapture条件を設定可能（常にチェック！）

Debugger Wizard

➤ ②青枠

- JTAG USER TAP - USERデザインに合わせプルダウンリストから選択 (USER1,2,3,4)



➤ ③緑枠

- Signals from - Post-Map, elaborated (Pre-Map) Netlistのいずれかを選択

Debugger Wizard

The screenshot shows the Debugger Wizard interface with two panels. The left panel, titled '候補信号' (Candidate Signals), lists signals from a 'knight_led' component: STOP (width 1, clock domain Undefined), state (width 4, clock domain CLK), skip_cnt (width 27, clock domain CLK), RST_X (width 1, clock domain Undefined), LED (width 8, clock domain CLK), and CLK (width 1, clock domain Undefined). An orange box highlights the '>>' button between the panels. The right panel, titled '選択信号' (Selected Signals), lists the selected signals: state (width 4, clock domain CLK), skip_cnt (width 27, clock domain CLK), and LED (width 8, clock domain CLK). A black box highlights the 'CLK' clock domain for the selected signals, with a circled '5' next to it. A blue arrow points from the Japanese text on the right to this black box.

Name	Width	Clock Domain
▼ knight_led		
└ STOP	1	Undefined
▶ = state	4	CLK
▶ = skip_cnt	27	CLK
└ RST_X	1	Undefined
▶ = LED	8	CLK
└ CLK	1	Undefined

候補信号

Name	Width	Clock Domain	Probe Type
= state	4	CLK	DATA AND TRIGGER
= skip_cnt	27	CLK	DATA AND TRIGGER
= LED	8	CLK	DATA AND TRIGGER

選択信号

サンプリングクロック
設定選択可能
(デフォルトは
駆動クロック)

➤ ④ 橙枠

- 信号の選択 - Debugで観測する信号を選択
左のウィンドウからクリック >> ボタンで選択

➤ ⑤ 黒枠

- Clock Domain – サンプリングするClock sourceを選択
Undefinedの信号もClock sourceを選択することでDebug対象にできます

Clock Domain - 注意点

➤ 指定できるClock Sourceは1種類のみ

- Clock Sourceを複数指定するとDebuggerは動作しません
→ 駆動クロックが異なる信号がある場合、Clock sourceの変更必要

Name	Width	Clock Domain
▼ JTAG_SPI_Falsh		
└─ clkln	1	Undefined
└─ hold_n	1	Undefined
└─ jtag_inst1_CAPTURE	1	Undefined
└─ jtag_inst1_DRCK	1	Undefined
└─ jtag_inst1_RESET	1	Undefined
└─ jtag_inst1_RUNTEST	1	Undefined
└─ jtag_inst1_SEL	1	Undefined
└─ jtag_inst1_SHIFT	1	Undefined
└─ jtag_inst1_TCK	1	Undefined
└─ jtag_inst1_TDI	1	Undefined
└─ jtag_inst1_TDO	1	u_efx_spi_loader_top/jt...
└─ jtag_inst1_TMS	1	Undefined
└─ jtag_inst1_UPDATE	1	Undefined
└─ miso	1	Undefined
└─ mosi	1	clkln
└─ nss	1	clkln



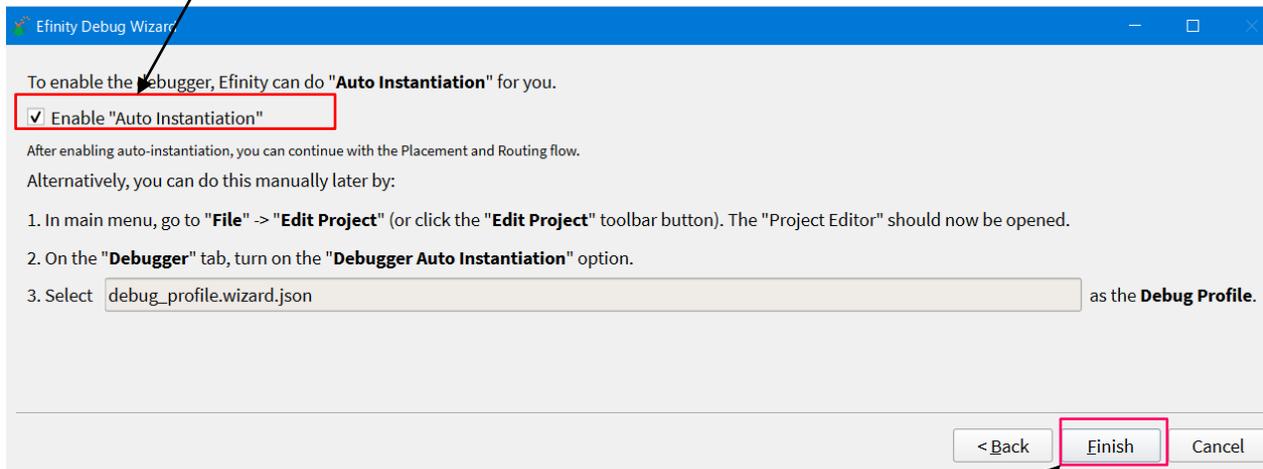
Name	Width	Clock Domain	Probe Type
└─ jtag_inst1_TDO	1	clkln	DATA AND TRIGGER
└─ mosi	1	clkln	DATA AND TRIGGER
└─ nss	1	clkln	DATA AND TRIGGER

異なるClock Domainの欄をクリック
→ サンプルングClockを変更し1種類のみとする

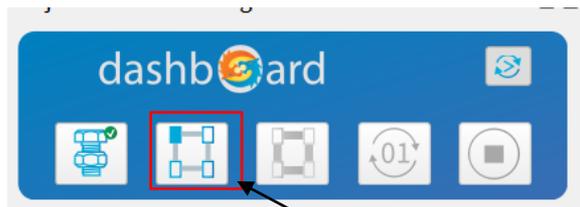
異なるClock Domainの信号を選択

Debugger Wizard

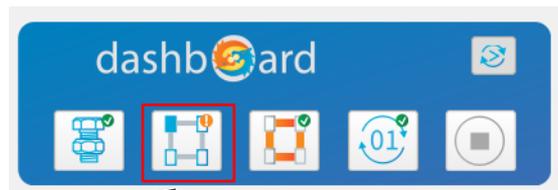
'Enable Auto Instantiation'がチェックされていることを確認！



Finishをクリック → Debug Instance生成



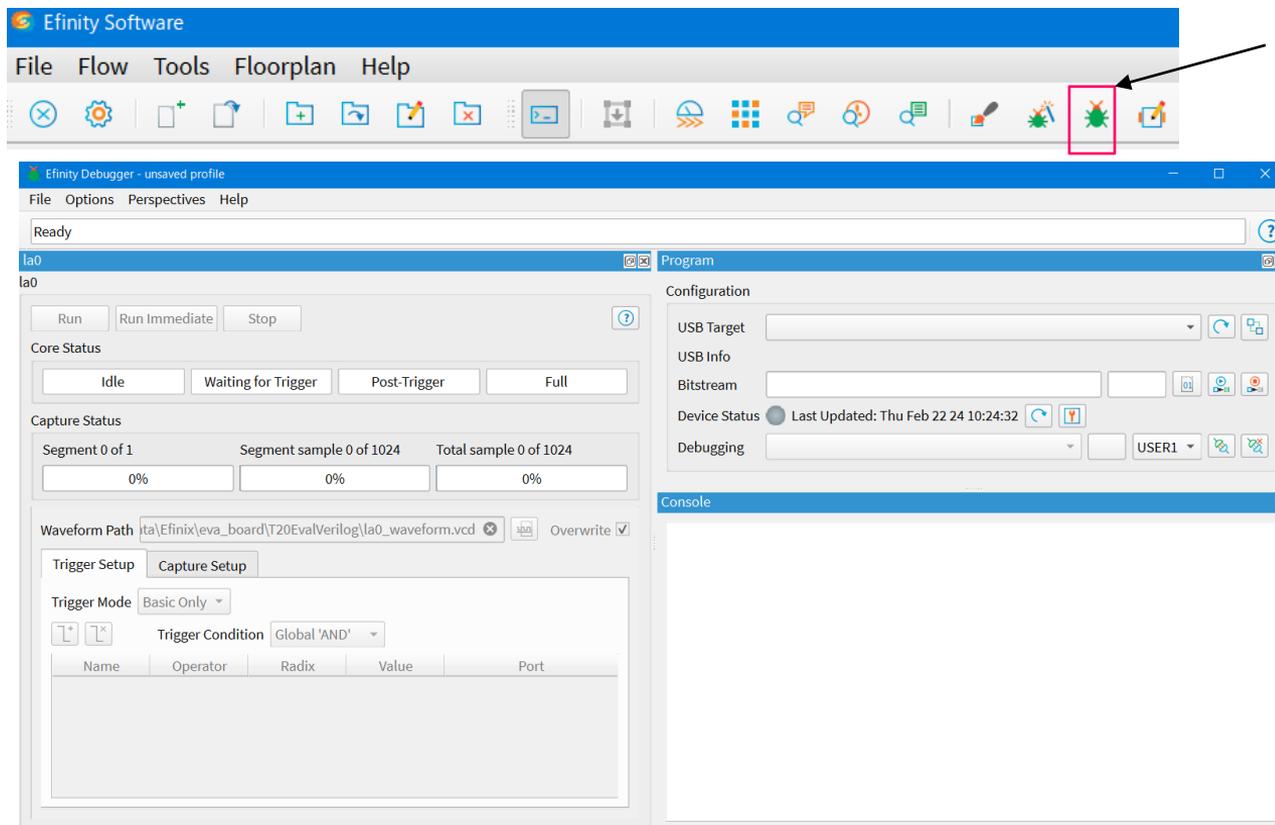
or



Wizardを終了すると'Place'ボタンがグレイアウト or !マーク
➤ Placeボタンを押し再コンパイル

コンパイルを終了すると Logic Analyzerによるデバッグが
可能な環境が準備できます

Debugger Logic Analyzer起動

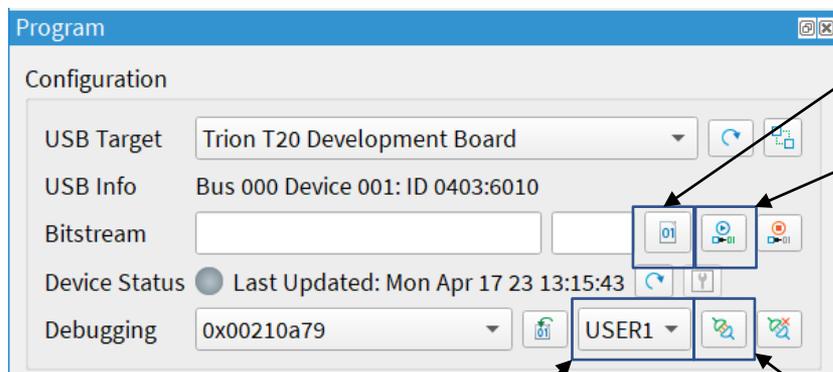


アイコンをクリック
→ Debuggerを起動

起動時のDebugger
画面



Debugger - Program



(1) Bit Streamファイル (.bit)を選択

(2) プログラム開始

(3) JTAG User TAP選択

(4) コネクトをクリックしデバッガ接続

Debuggerを起動したら → 'Program'のWindowにて

- ✓ (1) Bit Streamファイルを選択 → (2) プログラムを行なう
- ✓ (3) JTAG User TAPの選択が正しいことを確認 (P.7にて設定と一致しているか?)
 - (4) コネクトしDebuggerを開始できる状態となります。

Debugger - Trigger Setup -> トリガ条件設定

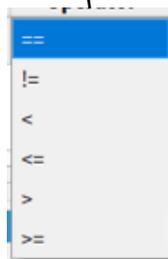
トリガー信号追加アイコン

Trigger Setup Capture Setup

Trigger Mode Basic Only

Trigger Condition Global 'AND'

Name	Operator	Radix	Value	Port
state[3:0]	==	Bin	0000	probe0[3:0]



プルダウンリストから
条件式を選択

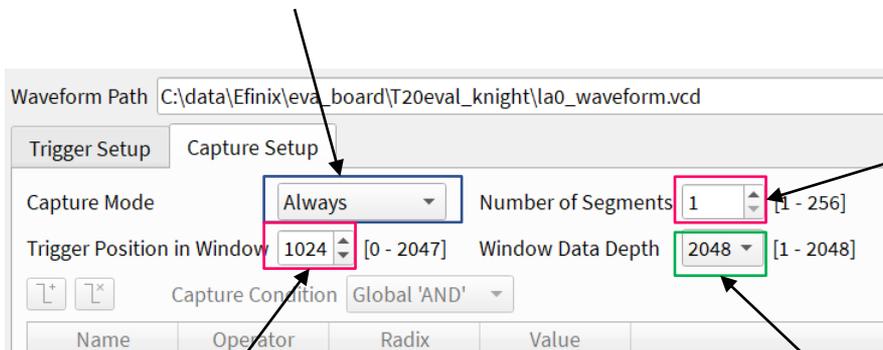
Bin
Hex
Dec

条件値を設定

上図のようにTrigger Setup画面で
Trigger条件を設定します。

Capture Setup

- Always - Trigger条件に関係なく波形データをキャプチャする
- Basic - Trigger条件と一致した時に波形をキャプチャ



キャプチャする回数を設定
= トリガ発生回数

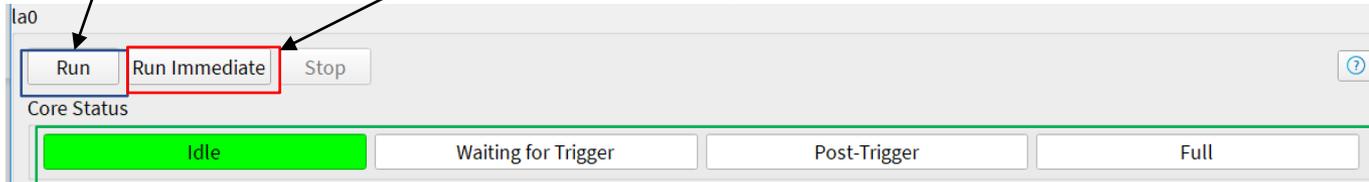
Trigger位置

キャプチャ1回分のバッファ容量
= Buffer Depth / キャプチャ回数

Debugger - Run

Debug開始
➤ Trigger条件満たすまで実行

Run Immediate
➤ Buffer Depth 1回分のRun実行



Debug実行中の状態を表示

- Runボタンを押してDebug開始 → トリガー条件を満たすとSTOPする
- Run Immediateを押すとBuffer Depth分'Run'→'STOP'（トリガ条件不要）
- Run -> Stopボタン押下でも波形ファイルは保存されます（トリガ条件不要）

Debug 結果 - 波形確認

Efinityには波形表示機能がありません。従ってサードパーティ製の波形表示ツールが必要となります。

Debugの結果は'vcd (Value Change Dump)'という形式のファイルで出力されます。この形式をサポートする波形表示ツールを使用すればdebug結果を確認できます。

'vcd'形式をサポートする無料の波形表示ツールとして'GTKWave'が存在します。

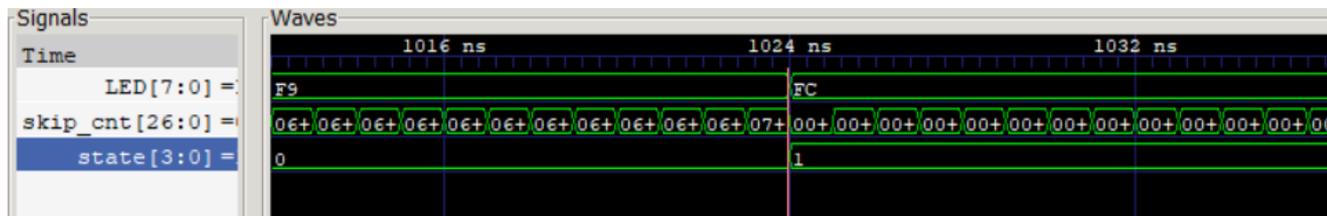
Efinix社でも波形表示ツールとしてGTKWaveを紹介しています。
下記URLからダウンロード可能です。

[gtkwave - Browse Files at SourceForge.net](#)



表示波形

Debug結果の表示波形例を以下に示します
(GTKWaveによる表示波形)



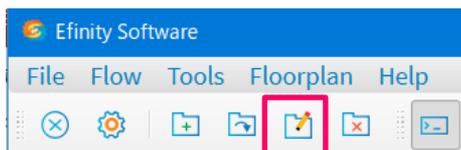
手順

1. GTK Waveを起動
2. 波形を確認したい対象の'vcd'ファイルをGTK Waveへドラッグ & ドロップ

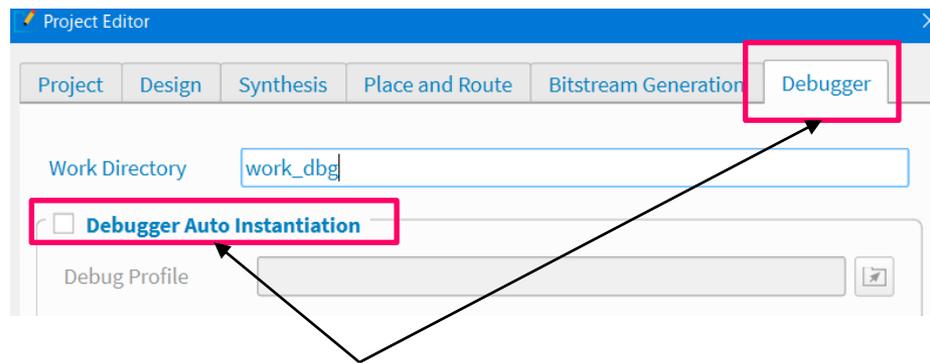
Auto Instantiationデバッグコア削除方法

Debug Wizardで作成したAuto Instantiationデバッグコア（Logic Analyzer）はデバッグ終了後は冗長回路となるので削除します。

削除の方法は以下の通り



Edit Project



Debuggerタブを選択 ->
Debugger Auto InstantiationをOFFにする



再コンパイル実行

ManualでのDebug信号定義



Open Debuggerアイコン

Debug Wizard以外にもProfile EditorにてDebug対象信号を定義することができます。

Open Debuggerアイコンをクリック

→ Debuggerを起動しProfile Editorで定義します

※ ただし、デバッグモジュール (HDL) 追加の他
トップモジュールの修正も必要です

→ ManualでのDebugは手間を要するので本書での説明は割愛します。

EOF

