

# AE-T8F81-DIP40 Efinity チュートリアル

---

対応バージョン : Efinity 2024.2.294+patch 2024.2.294.1.19  
Unified Netlist Flow対応

# 概要

## ■Efinityの概要

## ■仕様

## ■コンパイル

## ■シミュレーション(Verilogのみ)

## ■Interface Designer(ピン定義)

## ■タイミング解析

## ■書き込み

## ■デバッガ

## ■Appendix

- タイミング制約その2
- PLLを使用する
- リプルカウンタ
- リセット回路
- その他情報

# 必要機材のリスト

## ■T8F81 DIP化モジュールキット

- AE-T8F81-DIP40
- <https://akizukidenshi.com/catalog/g/g129595/>

## ■FT232HL USBシリアル変換モジュール

- FT232HL DIP化基板
- <https://akizukidenshi.com/catalog/g/g106503/>

## ■USBケーブル

- USB A オス to miniB オスケーブル
- <https://akizukidenshi.com/catalog/g/g117014/>

## ■ブレッドボード

- ブレッドボード BB-801
- <https://akizukidenshi.com/catalog/g/g105294/>

## ■LED

- 3mm赤色LED x 4個
- <https://akizukidenshi.com/catalog/g/g111577/>

## ■タクトスイッチ

- タクトスイッチ(黒) x 2個
- <https://akizukidenshi.com/catalog/g/g108073/>

## ■抵抗

- 1k $\Omega$  1/4W x 4個
- <https://akizukidenshi.com/catalog/g/g125102/>

## ■ジャンパーワイヤ

- オスーオス
- <https://akizukidenshi.com/catalog/g/g115869/>
- オスーメス
- <https://akizukidenshi.com/catalog/g/g117228/>



# 演習を始める前に

## ■ピンの設定は2つの方法があります

### – Interface Designer Flow (現在主流)

- **Interface Designer Flow**のページを実行してください
- 双方向バッファや3-Stateバッファをソースコードに記述しない
  - ソースコードにはI/Oブロックを含まない
- GUIによりI/Oブロックの配置を設定
- タイミング解析はCore内部のみでI/Oを含まない

### – Unified Netlist Flow (2024.2で始めて実装)

- **Unified Netlist Flow**のページを実行してください
- Interface Designerの設定を自動生成
  - 双方向バッファや3-Stateバッファをソースコードに記述
  - PLL、OSCのインスタンスをソースコードに記述
  - .isfファイルによりI/Oブロックの配置やPLL/OSCのリソース割り当てを記述
- SDCはInterface Designerデザインフローと同じ
  - タイミング解析はCore内部のみ

## ■Unified Netlist Flowは今後のアップデートが楽しみな機能です

- Unified Netlist FlowとInterface DesignerデザインフローはRTLのTOP階層が異なります
- Unified Netlist Flow は2024.2+patch 2024.2.294.1.19では制限が多い為、参考程度としてください
- 新機能ですので、今後のバージョンアップに期待しましょう



# Efinityの概要

# 推奨動作条件

## ■Efinity推奨動作条件

- サポートOS
  - Windows11 64bit or Windows10 64bit
  - Ubuntu v18.04 or later
  - Red Hat Enterprise x86-64 v8.0 or later
- メモリーサイズ

FPGA種別	8GB	16GB
Trion	T4, T8, T13, T20, T35	T55, T85, T120
Titanium	Ti35, Ti60	Ti90, Ti120, Ti180

## ■ダウンロード要件

- Webで「ユーザ登録」と「フリーライセンスのリクエスト」の両方を行ってください
- 資料のダウンロードにも必要となります
  - <https://www.efinixinc.com/support/>

# インストールするソフトウェア

## ■ Efinity Version: 2024.2.294

- <https://www.efinixinc.com/support/efinity.php>

## ■ Microsoft Visual C++ 2019 x64 runtime library

- <https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170>
- vc\_redist.x64.exe

## ■ Java Run time

- <https://www.oracle.com/java/technologies/javase/jdk18-archive-downloads.html>
- jdk-18.0.2.1\_windows-x64\_bin.exe

## ■ Zadig

- <https://zadig.akeo.ie/>
- zadig-2.8.exe
- インストール不要。ダウンロードしたexeがアプリ本体
- ドライバインストール手順は下記参照
- <https://www.efinixinc.com/docs/an050-managing-windows-drivers-v1.1.pdf>

## ■ gtkwave

- <https://sourceforge.net/projects/gtkwave/files/gtkwave-3.3.100-bin-win64/>
- gtkwave-3.3.100-bin-win64.zip
- インストールフォルダー : C:\¥Efinity¥gtkwave64

## ■ Icarus Verilog

- <http://bleyer.org/icarus/>
- iverilog-v12-20220611-x64\_setup.exe
- インストールフォルダー(展開したファイルを移動) : C:\¥Efinity¥iverilog



# 環境変数の設定

## ■ユーザーの環境変数に以下追加

- C:¥Efinity¥gtkwave64¥bin
- C:¥Efinity¥iverilog¥bin
- C:¥Program Files¥Java¥jdk-18.0.2.1¥bin

## ■その他注意点

- プロジェクトフォルダーは英数字のみ。スペースや倍角文字を含まないこと



# Efinityメイン画面

Explorer

ログをフィルタリング表示

タイミング解析結果の表示

配置の確認

ログ

タイミング  
メッセージ

フロアプラン

dashboard

コンソール

Interface Designer

デバッガの設定

デバッガ

プログラマー

I/OとハードIPコアの設定

ロジックアナライザ機能

コンフィギュレーションの書き込み

実行フロー  
ボタン

プロジェクト  
ファイル  
一覧

プロジェクト  
プロパ  
ティ

ビットファイル生成までの  
一連の実行を行うパネル

一連／個別  
実行切り替え

論理合成  
配置  
配線

実行停止  
ビットストリーム  
生成

# Netlist タブ と Result タブ

Elaboration ネットリスト (論理合成前)

Netlist

論理合成 ネットリスト

Hierarchy Elaborated Synthesized

Leaf Cells

- Cell: i1 (VERIFIC\_GND)
- Cell: i2 (VERIFIC\_PWR)
- Cell: add\_5 (add\_64u\_64u)
- Cell: mux\_6 (mux\_64)
- Cell: inv\_9 (inv\_4)
- Cell: mux\_10 (mux\_4)
- Cell: dff\_7 (wide\_dffrs\_64)

Nets

- NetBus: led(4)
- NetBus: count1sec(64)
- NetBus: n7(64)
- NetBus: n72(64)
- NetBus: n203(4)
- Net: n1
- Net: n2

Property Value

Instance	add_5
Parameter Count	0

Project Netlist Result

Interface Simulation

Synthesis

- count16sec.map.v
- count16sec.map.rpt
- count16sec.map.out
- count16sec.res.csv

Placement

Routing

Periphery Resource

GPIO	7 / 55
JTAG User TAP	0 / 2
Oscillator	0 / 1
PLL	0 / 1

Core Resources

Inputs	3 / 96
Outputs	4 / 113
Clocks	1 / 16

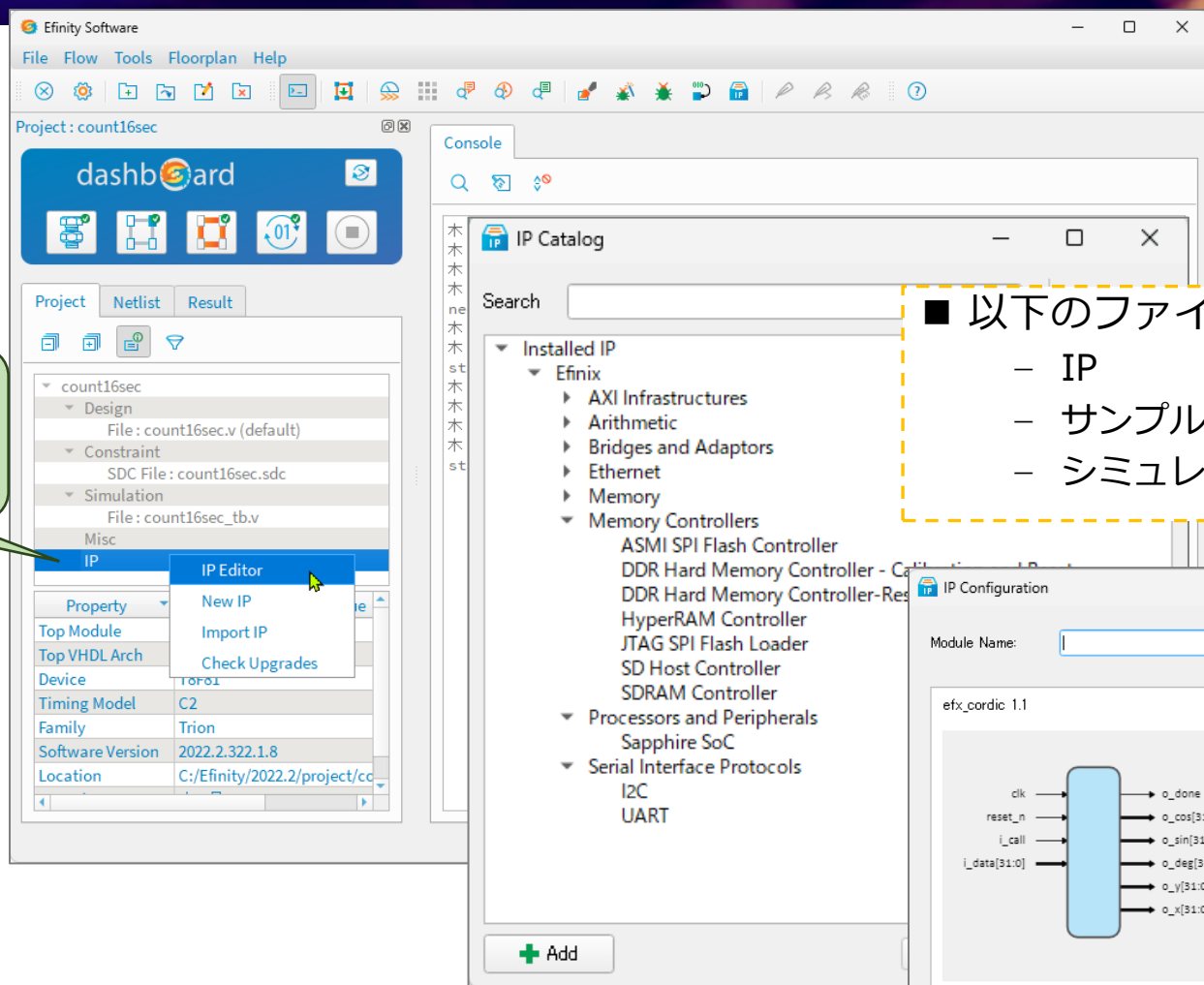
レポートファイル

結果サマリー

使用リソースと  
エラーある/無しをこ  
のパネルで確認

# IPエディタ

Explorer

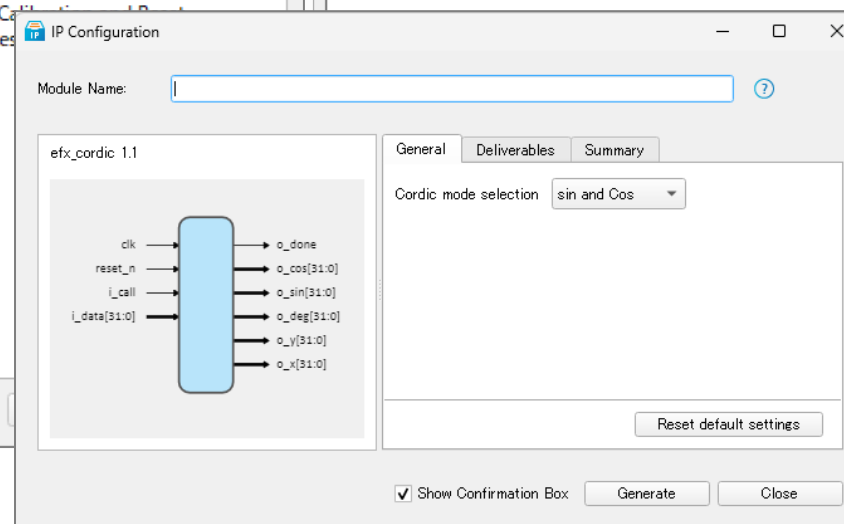


右クリックでIPコア  
の生成/修正

## ■ 以下のファイルをipフォルダーに生成

- IP
- サンプルデザイン
- シミュレーションのテストベンチ

IPコアを生成すると  
IPのフォルダーにサ  
ンプルデザインも生  
成される



# 論理合成のリソースレポート

## ■Resultタブ→Synthesis →<プロジェクト名>.res.csv

リソース使用比率はこの  
ファイルで確認。  
配置配線でリソース使用量  
が変わる為、最終結果は  
Resultタブで確認すること

The screenshot shows the Explorer software interface with the 'Result' tab selected. The left sidebar shows a tree view with 'Synthesis' expanded, and 'xyloni\_sapphire\_2022\_2.res.csv' selected. The main area displays the 'Resource Utilization' table in 'Text' view.

**Resource Utilization Table:**

Module	FFs	ADDs	LUTs	RAMs	DSP/MULTs
top_soc:top_soc	3023(0)	655(0)	3649(3)	16(0)	4(0)
soc_inst:SapphireSoC	3023(0)	655(0)	3646(0)	16(0)	4(0)
u_EfxSapphireSoc:EfxSapphireSoc_a3dbd00444f549bc8dd9c13f1f17876b	3023(521)	655(19)	3646(325)	16(0)	4(0)
bufferCC_7:BufferCC_4_a3dbd00444f549bc8dd9c13f1f17876b	2(2)	0(0)	0(0)	0(0)	0(0)
bufferCC_8:BufferCC_5_a3dbd00444f549bc8dd9c13f1f17876b	2(2)	0(0)	0(0)	0(0)	0(0)
system_cores_0_logic_cpu:VexRiscv_a3dbd00444f549bc8dd9c13f1f17876b	1311(1244)	335(335)	1998(1956)	4(4)	4(4)
lBusSimplePlugin_rspJoin_rspBuffer_c:StreamFifoLowLatency_a3dbd00444f549bc8dd9c13f1f17876b	67(67)	0(0)	42(42)	0(0)	0(0)
system_hardJtag_debug_logic_jtagBridge:JtagBridgeNoTap_a3dbd00444f549bc8dd9c13f1f17876b	83(74)	0(0)	47(45)	0(0)	0(0)
flowCCByToggle_1:FlowCCByToggle_a3dbd00444f549bc8dd9c13f1f17876b	9(7)	0(0)	2(2)	0(0)	0(0)
inputArea_target_buffercc:BufferCC_1_a3dbd00444f549bc8dd9c13f1f17876b	2(2)	0(0)	0(0)	0(0)	0(0)
system_hardJtag_debug_logic_debugger:SystemDebugger_a3dbd00444f549bc8dd9c13f1f17876b	78(78)	0(0)	24(24)	0(0)	0(0)
bufferCC_9:BufferCC_6_a3dbd00444f549bc8dd9c13f1f17876b	2(2)	0(0)	0(0)	0(0)	0(0)
system_bridge_bmb_arbiter:BmbArbiter_a3dbd00444f549bc8dd9c13f1f17876b	3(0)	0(0)	72(0)	0(0)	0(0)
memory_arbiter:StreamArbiter_a3dbd00444f549bc8dd9c13f1f17876b	3(3)	0(0)	72(72)	0(0)	0(0)





Interface Designerのデザイン  
チェックで出るメッセージは  
「Design Check」に解説がある

## Efinity Help

- Efinity Help
  - Efinity Help Overview
    - New in v2022.2
    - Upgrading from Older Versions
    - Hardware and Software Requirements
    - Efinity Quick Start
    - Icon Reference
    - Revision History
  - Projects and Preferences
    - Setting General Tool Preferences
    - Auto-Load Place-and-Route Data
    - Efinity Main Window
    - Project Editor
    - Project Pane
    - Migrating a Project to another FPGA
    - Using VHDL Libraries
  - Running the Tool Flow
    - Run the Flow with the Dashboard Controls
    - Run the Flow from the Command Line
    - Netlist Pane
    - Viewing Messages and Logs
    - Result Pane
    - Viewing Place-and-Route Results
  - Using the IP Manager
    - Supported IP Cores

## Efinity Help Overview

The Efinity® software provides a complete tool flow for designing a visual way for you to set up projects, run the software flow, view portion of your design. You use the command-line to perform sim

Figure 1. Design Flow Overview

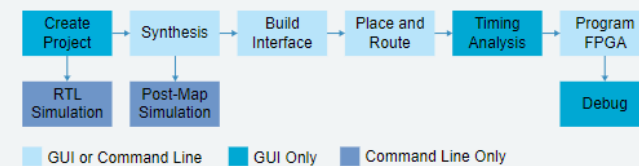














Table 1. Titanium FPGAs Supported in Efinity® Software v2022

FPGA	Package
Ti35	F100S3F2, F225
Ti60	W64
	F100S3F2, F225
Ti90	M361, M484
	F529, J361, J484, G529
Ti120	M361 M484

# ファイル・フォルダの種類

 ip	IP生成フォルダ
 outflow	実行結果
 work_dbg	ツールのワークフォルダ
 work_pnr	
 work_pt	
 work_syn	
 count16sec.peri.xml	
 count16sec.sdc	タイミング制約
 count16sec.v	Verilogソースコード
 count16sec.vdb	生成されたネットリスト
 count16sec.xml	プロジェクト設定
 count16sec_tb.v	Verilogテストベンチ

count16sec.bit : JTAG 書き込みファイル  
count16sec.hex : SPI 書き込みファイル  
count16sec.map.out : 論理合成のエラー  
count16sec.place.out:ピンアサイン  
count16sec.timing.rpt: タイミング

HelpのAppendix: Efinity Project Filesに  
ファイルの解説がある



# 仕様

# 仕様書

## ■機能仕様

- 4ビットのカウンターを約 1 秒間隔でカウントアップさせる
- SWは以下のように動作する
  - rst\_n : 1:リセットネゲート、0（押下） : リセットアサート
  - reverse\_n : 1:カウントアップ、0（押下） : カウントダウン
- クロックソース : 25MHz、外部から入力

## ■その他事項

- リセットはリムーバルタイムを考慮する
- VerilogファイルフォーマットはUTF-8、コメントは日本語可
- ディレクトリ名は英数字と"\_"のみ、スペース使用不可
- 全てのエラー、ワーニングを除去する
- タイミング制約はプロジェクトの初期段階で設定する



# T8F81の接続

rst\_n スイッチ  
reverse\_n スイッチ  
led0, 1, 2, 3

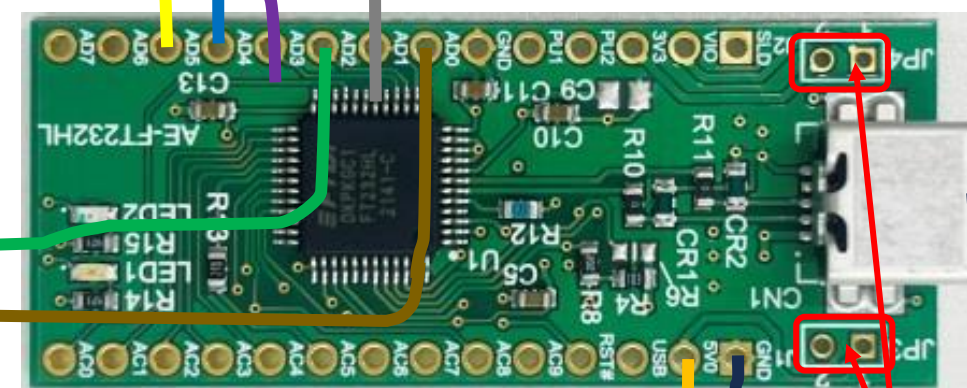
1K $\Omega$

ピン名	用途
AD0	TCK
AD1	TDI
AD2	TDO
AD3	TMS
AD4	CRST
AD5	SS

USBケーブルで  
PCに接続



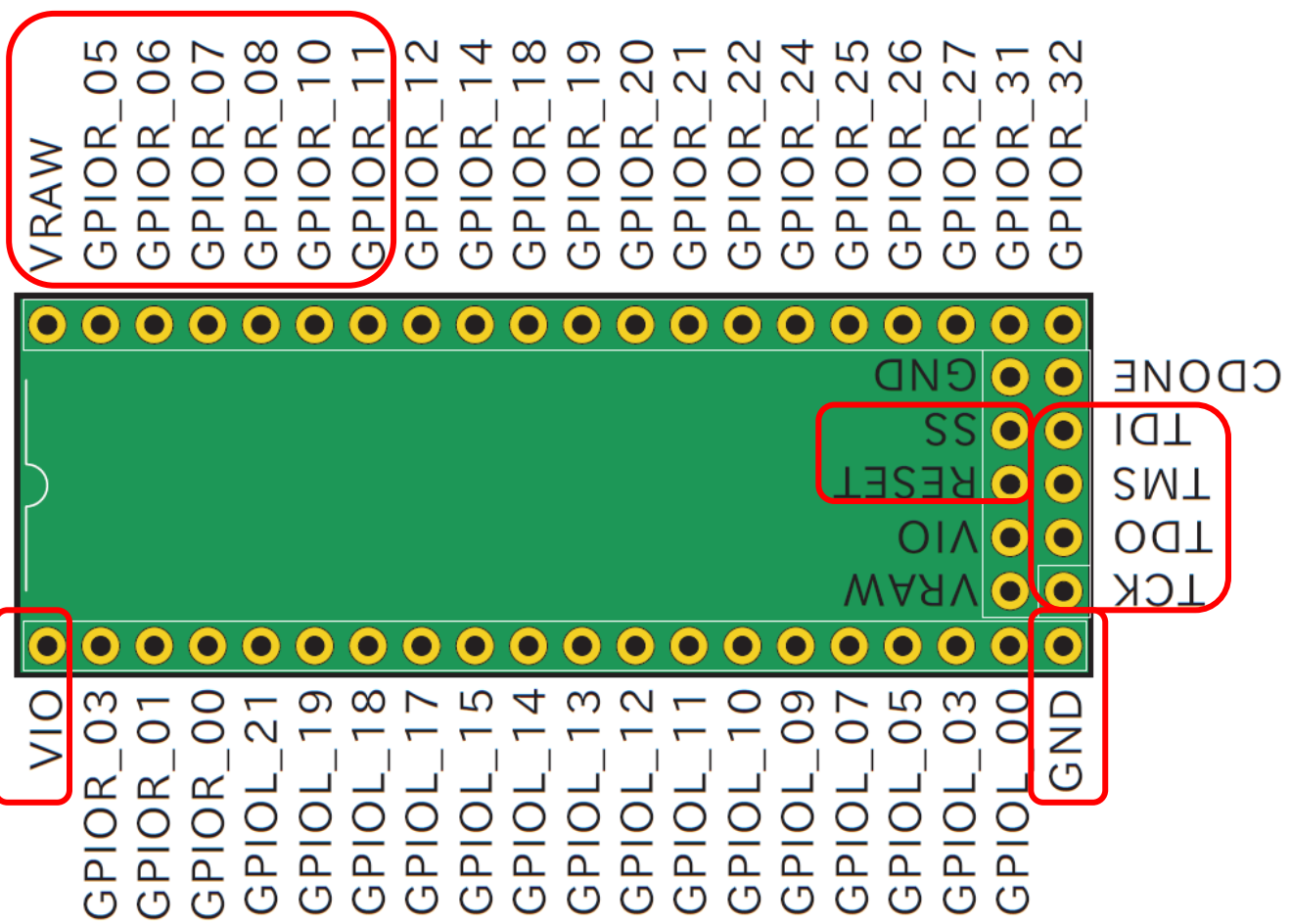
AE-T8F81-DIP40



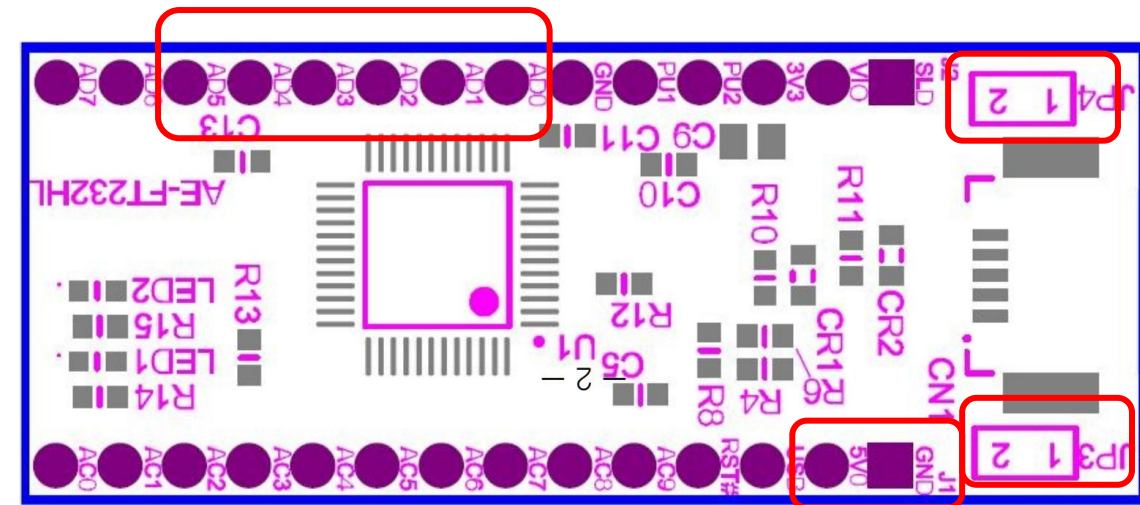
AE-FT232HL

ジャンパー 2 か所  
接続

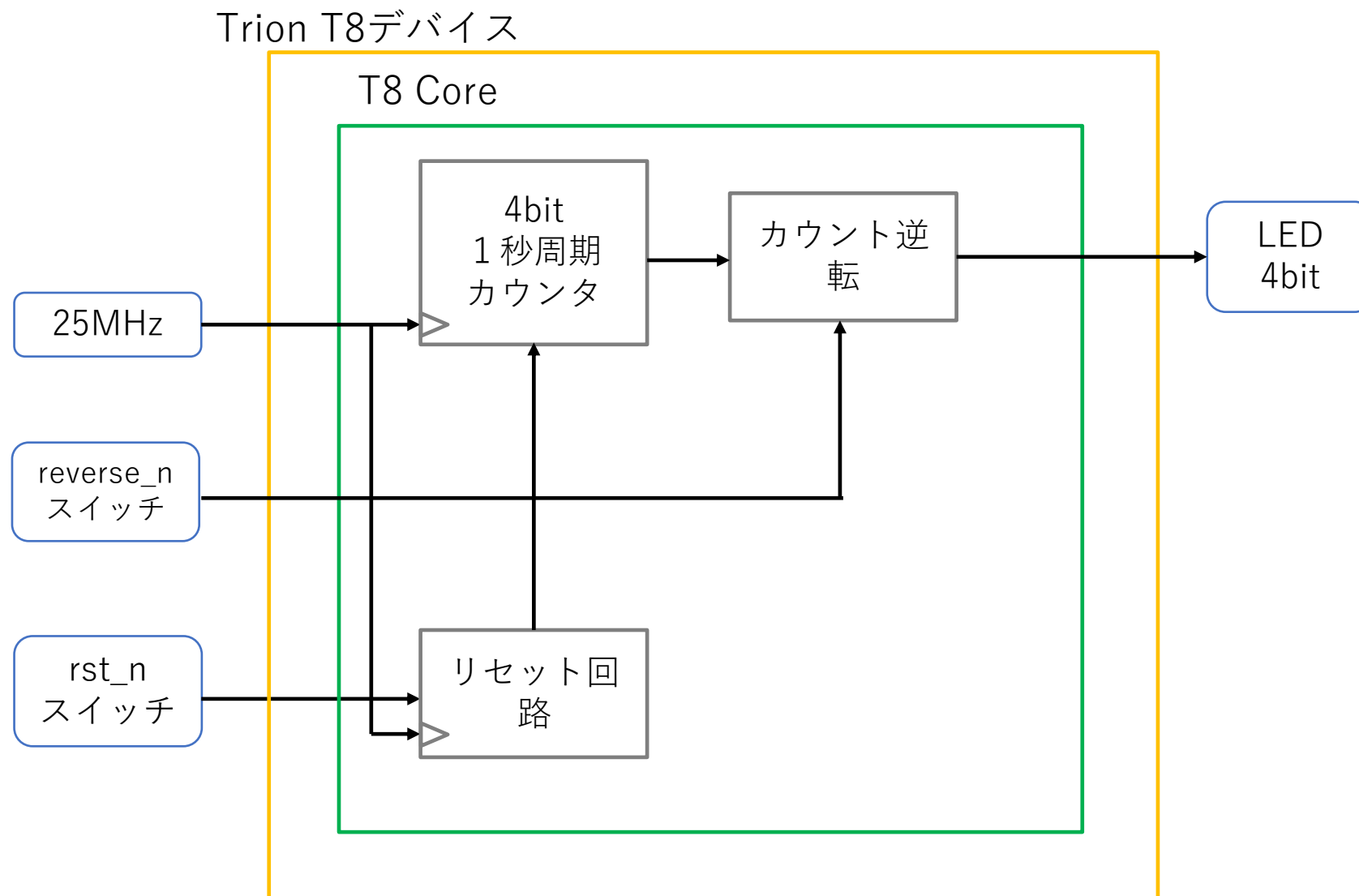
# 参考資料：ピン配置



ピン名	用途
AD0	TCK
AD1	TDI
AD2	TDO
AD3	TMS
AD4	RESET
AD5	SS



# ブロック図



# I/O、リセット、クロック系統

## ■I/O、リセット、クロック系統は仕様の段階で確認

– I/Oブロック、PLL、IPコアのみを実装した回路を作成しコンパイル

– クロック、PLLのリファレンスは専用ピンを使用

- クロックとPLLの専用ピンは共通ではない

– クロックの注意点

- FFのクロック入力に入っている信号は全てクロックとして扱う
- クロックが多すぎるとか、分周クロックを※リップルカンターで作っているなどの場合にグローバルバッファを過剰に使用するケースが発生する
- クロックセレクトをLUTで構成してはいけない

クロック、リセット、IPコアがFPGAにマッピング可能か確認するための作業。ユーザー回路はダミー回路で良い。ピン配置やIPコアの設定によってマッピングできないことがあるので、どのFPGAメーカーも同様の推奨がある。

基板設計時にクロック専用ピンとPLL専用ピンの両方にクロックを入れておくと良い

※ リプルカウンターはAppendixを参照



# コンパイル

# Efinityを起動

## ■ Windows操作

- スタートメニュー → Efinity 2024.2をクリック

## ■ Efinity初期設定

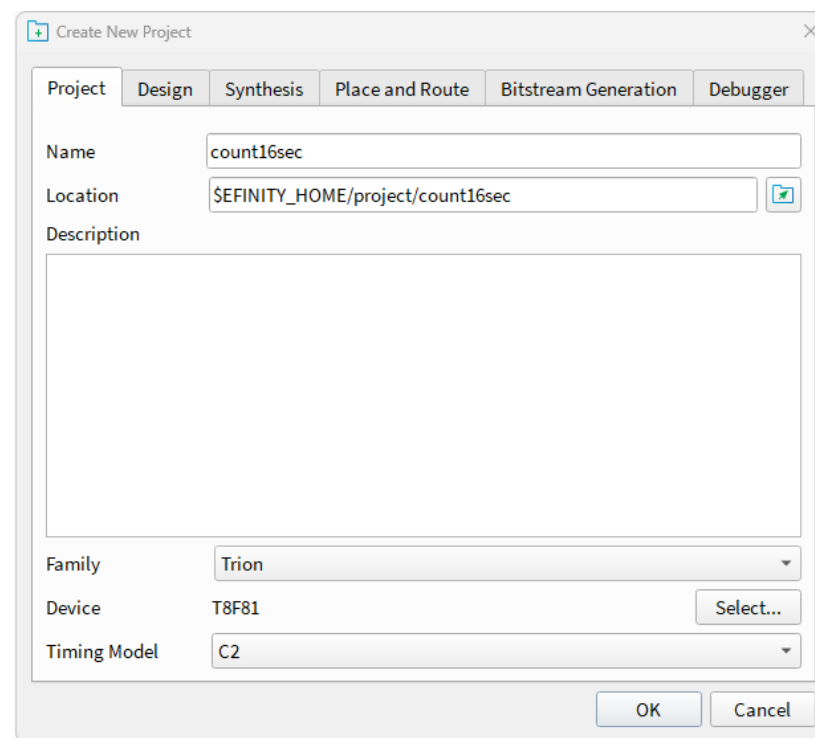
- Fileメニュー → Preferences → Auto-load Place and Route data : チェックOff ※1

※1 起動時間を短縮するための設定

## ■ Efinityでプロジェクト作成

- Fileメニュー → Create Project
- Create New Project窓
  - Projectタブ
    - Name : count16sec
    - Location : デフォルト
    - Device : T8F81
  - Designタブ
    - Top Module/Entity : count16sec
  - [OK]をクリック
  - Projectタブ以外はデフォルトとする ※2

※2 File → Edit projectで後から設定変更可能

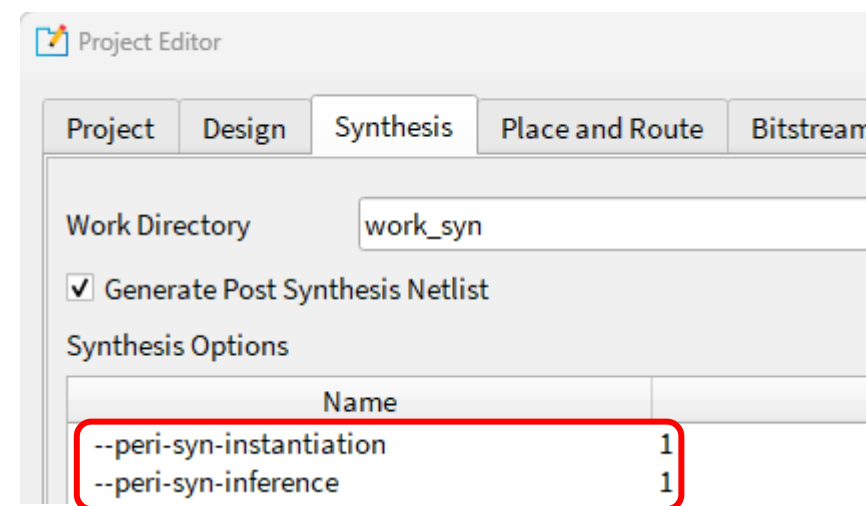


# Unified Netlist Flowの追加設定

演習

## ■Unified Netlist Flowの設定

- File→Edit Project→Synthesisタブ→Synthesis Options
  - --peri-syn-instantiation : 1
  - --peri-syn-inference : 1
- [OK]、[OK] でProject Editorをクローズ



# コンパイル

## ■Efinityでコンパイル

### – Interface Designer Flow



- Projectタブ → Design右クリック → Add
  - Open窓
    - Copy to Project : チェックOn
    - File name : count16sec.v※
    - [Open]クリック

※Designにテストベンチを入れない  
テストベンチはSimulationに入れる  
シミュレーションの章で解説

### – Unified Netlist Flow

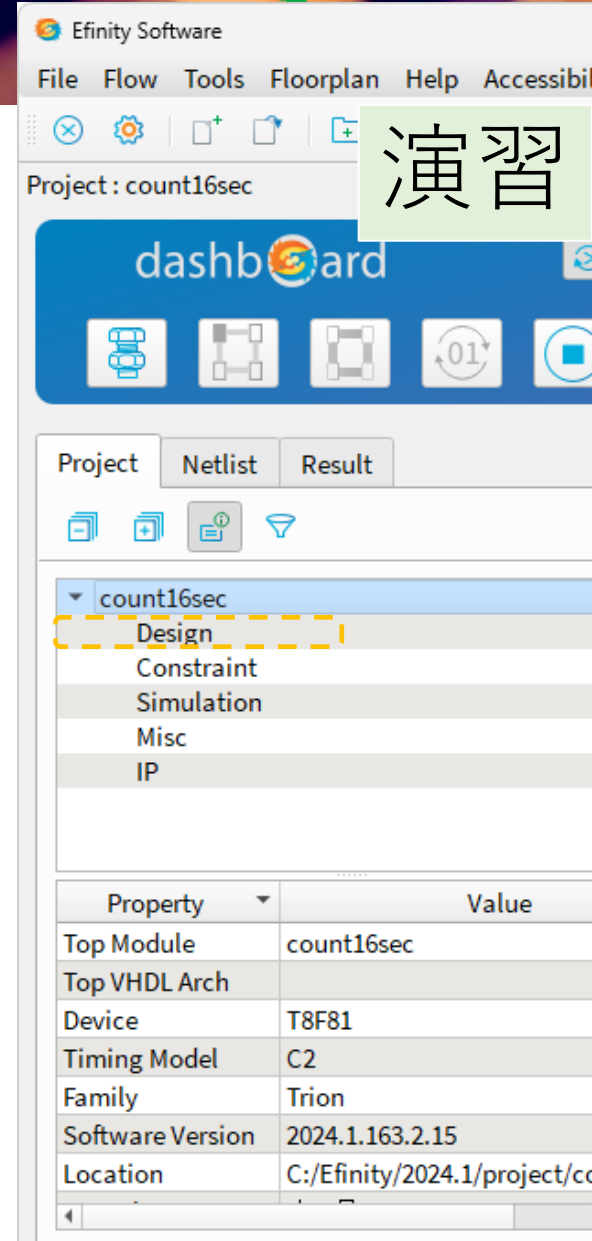
- count16sec.peri.xmlがあれば削除してから上の手順を実施

## ■SystemVerilog2009を設定

- Synthesis  をクリック（エラーが出る）
- Fileメニュー → Edit Project
  - Project Editor窓 → Designタブ
    - Verilog : SystemVerilog2009
    - [OK]
- Synthesis  をクリック（Consoleにエラーなし）

Explorer

演習





# コンパイル結果の確認


## ■Resultタブでエラーを確認 ※

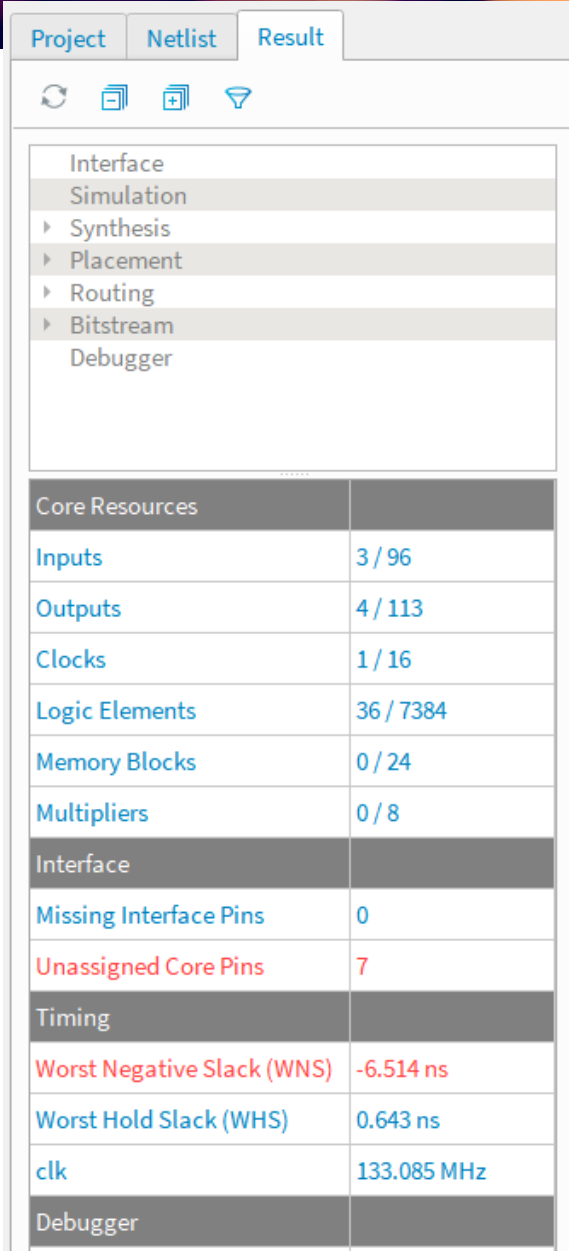
- RTLは正常
  - 論理シミュレーション可能
  - ネットリストシミュレーション可能
- Unassigned Core Pins
  - ピンの定義がされていない
- Worst Negative Slack
  - タイミング設定がされていない

## ■ここでEfinityを終了

- Fileメニュー → Exit

- 保存済みプロジェクトを開く場合は
  - File → Open Project
    - ファイル名: count16sec.xml
    - [開く]

※ ログ  を使用すると検索やフィルタが可能



Core Resources	
Inputs	3 / 96
Outputs	4 / 113
Clocks	1 / 16
Logic Elements	36 / 7384
Memory Blocks	0 / 24
Multipliers	0 / 8
Interface	
Missing Interface Pins	0
Unassigned Core Pins	7
Timing	
Worst Negative Slack (WNS)	-6.514 ns
Worst Hold Slack (WHS)	0.643 ns
clk	133.085 MHz
Debugger	

演習

# コマンドラインでの操作

## 演習

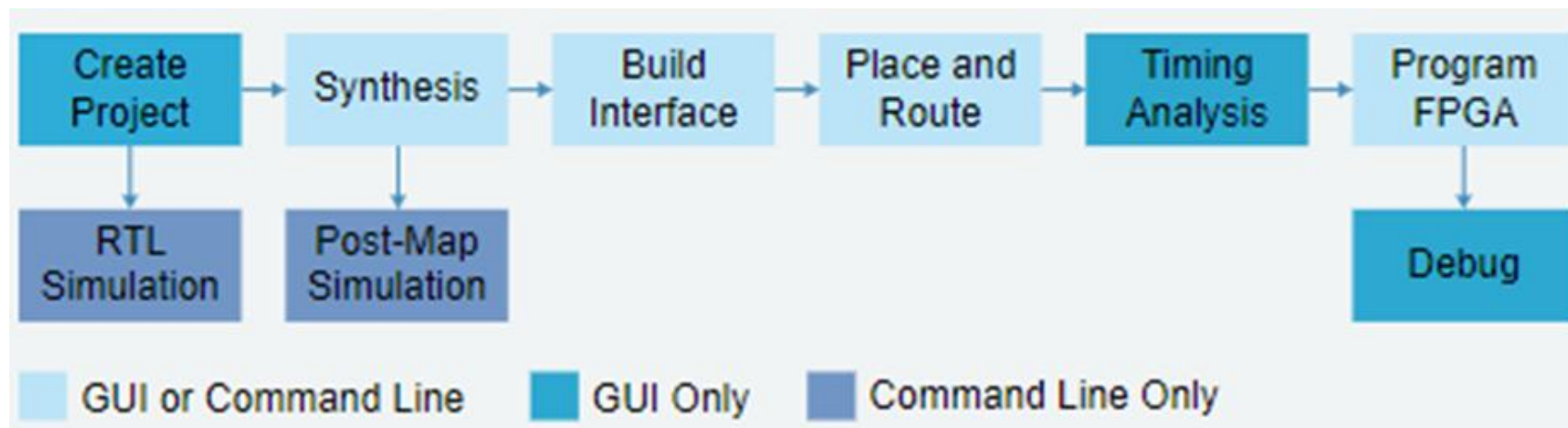
### ■ コマンド プロンプトで操作

- 環境変数の設定
  - `cd C:¥Efinity¥2024.2¥project¥count16sec`
  - `C:¥Efinity¥2024.2¥bin¥setup.bat`
- 論理合成、配置配線、ビットストリーム生成
  - **Interface Designer Flow**
    - `efx_run.bat count16sec.xml --flow compile`
  - **Unified Netlist Flow**
    - `del count16sec.peri.xml`
    - `efx_run.bat count16sec.xml --flow compile --un_flow`

Unified Netlist Flowの場合のみ、  
count16sec.peri.xmlが既にあるとエラー  
となるのでコンパイル前に削除する

### ■ コマンドライン操作可否

- プロジェクト作成、タイミング解析、デバッグはコマンドライン非サポート



# ピン定義

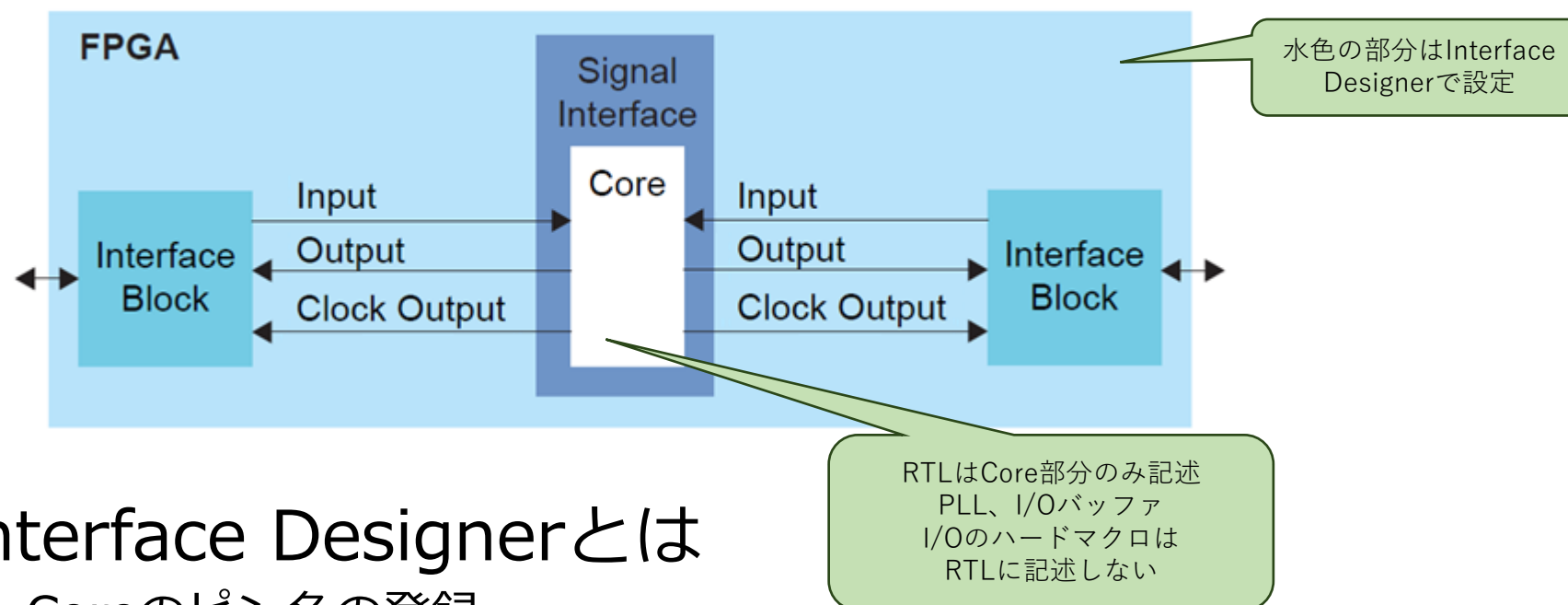


# Interface Designerを 使用するピン定義

**Interface Designer Flow**



# Interface Designer



## ■Interface Designerとは

- Coreのピン名の登録
- Interface Blockとの関連づけ
- 汎用I/Oのパッケージピン位置 (Resource名) の指定

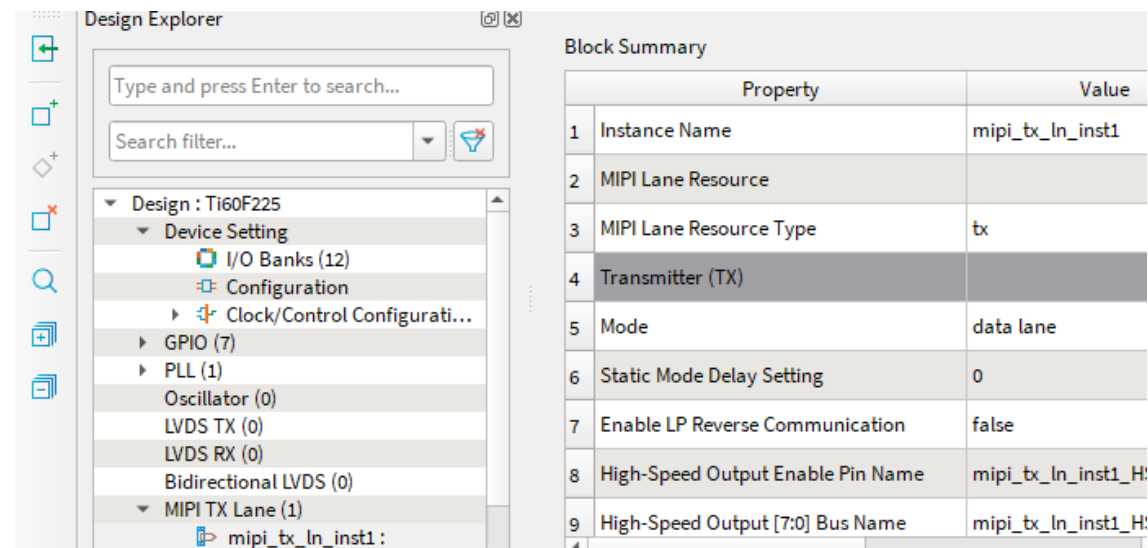
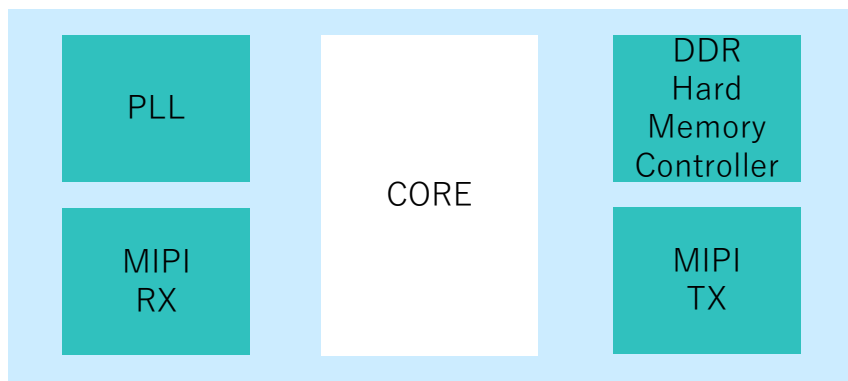
# Interface Designer — ハード IP コア

## ■ハードIPコアの登録

- 使用したハードIPコアのResource名によりピン位置が決まる

## ■IPコアのピン名の設定

- COREに接続するピン名の設定
- ピン名はRTLのTOP階層の信号名と一致させる
- Interface Designer内のみで接続されるクロックはRTLに記述しない



# Interface Designer

The screenshot displays the Interface Designer software interface, showing the design flow and various configuration windows. The main window is titled "Interface Designer" and features a menu bar (File, Design, Help) and a toolbar. The "Design Explorer" panel on the left shows the project structure, including "Design : T8F81", "Device Setting", "I/O Banks (5)", and "GPIO (7)". The "Design Summary" panel on the right provides details about the design, including the device (T8F81), package (81-ball FBGA), and last change date (Mon Jan 30 16:25:33 2023).

Key components and annotations include:

- 保存** (Save): Annotation pointing to the "Save" icon in the toolbar.
- エラーチェック** (Error Check): Annotation pointing to the "Error Check" icon in the toolbar.
- Instance View の On/Off**: Annotation pointing to the "Instance View" toggle switch in the Design Explorer.
- ピン配置の設定** (Pin Configuration Setting): Annotation pointing to the "Resource Assigner" window.
- Instance View でパッケージピンの設定** (Pin Configuration in Instance View): Annotation pointing to the "GPIO : Instance View" table.
- ピン名の設定** (Pin Name Setting): Annotation pointing to the "GPIO (7)" list in the Design Explorer.
- 信号名とIN/OUTの設定** (Signal Name and IN/OUT Setting): Annotation pointing to the "GPIO (7)" list in the Design Explorer.

The "GPIO : Instance View" table shows the following data:

Instance	Package Pin	Resource	I/O Bank	Alt Conn
clk	C3	GPIO_L_20	1B	PLL_CLKIN
led[0]	B3	GPIO_L_21	1B	None
led[1]	J6	GPIO_R_37	2B	None
led[2]	D7	GPIO_R_16	2A	GCTRL

The "Design Summary" table shows the following data:

Property	Value
1 Name	count16sec
2 Device	T8F81
3 Package	81-ball FBGA
7 Last Change Date	Mon Jan 30 16:25:33 2023
8 Database Version	20222999

# Package Planner

## ■パッケージ上のピンの位置を確認

File Design Help

Design Explorer

Type and press Enter to search...

Search filter...

Design : T8F81

Package Planner - count16sec

File View Help

Package View

81-ball FBGA Pinout Diagram

Top View

Device Name: T8F81

Pinout Diagram (Grid):

	1	2	3	4	5	6	7	8	9
A	VCCA_PLL	GND	TDR	TDO	GPIO R_00	GPIO R_03	VCCO2_A	GPIO R_08	GPIO R_10
B	GND_A_PLL	VCCO1_B	GPIO L_21_HSTATU	TMS	GPIO R_01	GPIO R_05	GND	GPIO R_11	GPIO R_14
C	VCC	GPIO L_16_CLK2	GPIO L_20_PLLIN	TCK	GPIO R_02_RESERVED_0	GPIO R_06	GPIO R_07	GPIO R_12	GPIO R_15_CBUS0
D	GND	GPIO L_17_CLK3	GPIO L_19_CTRL3	VCC	GND	GPIO R_13	GPIO R_18_CTRL7_CBUS1	GPIO R_17_CTRL8_CBUS2	GPIO R_18_CLK7
E	GPIO L_15_CLK1	GPIO L_14_CLK0	GPIO L_18_CTRL2	GND	VCC	GPIO R_23_CTRL4	GPIO R_21_CLK4_CBUS1	GPIO R_19_CLK6	GND

View Configuration

Report Pin information

Pin Type: GPIO

Pin name: A9

Pad name: GPIOR\_10

I/O Bank: 2A

I/O Bank Voltage: 3.3 V

I/O Standard: N/A

Block Type: GPIO

Block Instance: N/A

Resource Name: GPIOR\_10

Function: User IO

View Configuration

Report Pin information

☐ Display I/O bank group

☐ Display PLL Reference Clock

☐ Display Global Clock

☐ Display Global Control

☐ Display Assigned Resource

I/OバンクやI/Oの種類でフィルターできる



# ピン仕様

## 演習

T8F81C2のResource名	Coreピン名
GPIOR_07	led[0]
GPIOR_08	led[1]
GPIOR_10	led[2]
GPIOR_11	led[3]
GPIOR_05	rst_n
GPIOR_06	reverse_n
GPIOL_16 (PLL不使用の場合) または、GPIOL_20 (PLL使用の場合)	clk

## ■Efinityでプロジェクトを開く

- 手順はコンパイルの章を参照

## ■エラーを確認

- Resultタブ → Placement → count16sec.place.rpt を開き Unassigned Core Pins を確認

# Coreピンの登録

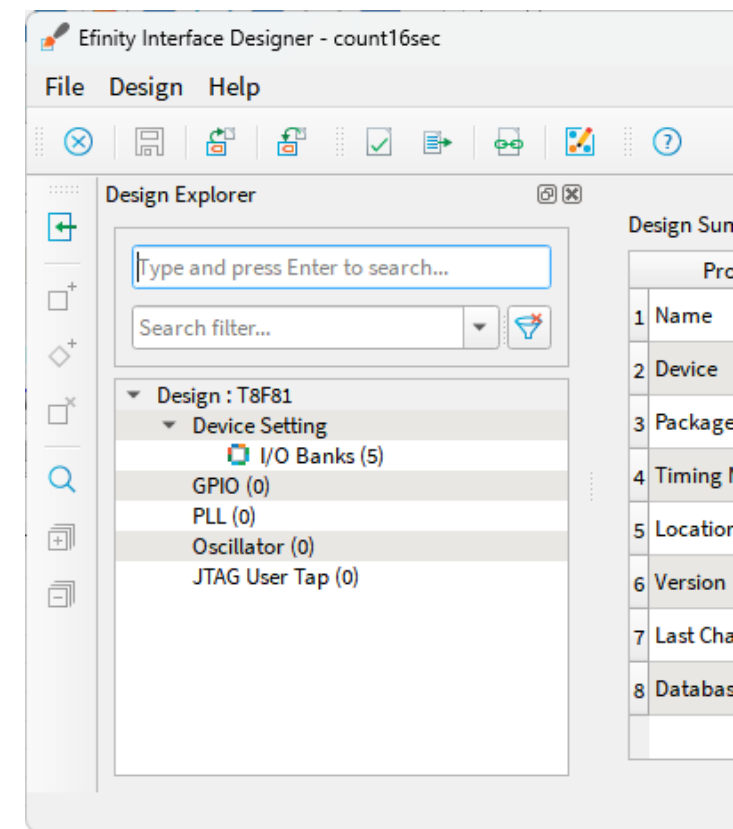
## 演習

### ■ Open Interface Designer をクリック

– Design Explorer窓の


- GPIOの右クリック→Create Bus
  - Name : led
  - MSB : 3, LSB : 0
  - Mode : output
  - [Next]、[Next]、[Finish]
- GPIOの右クリック→Create Block
  - Instance Name : rst\_n
  - Pull Option : weak pullup
  - Enable Schmitt Trigger : On
- GPIOの右クリック→Create Block
  - Instance Name : reverse\_n
  - Pull Option : weak pullup
  - Enable Schmitt Trigger : On
- GPIOの右クリック→Create Block
  - Instance Name : clk
  - Connection Type : gclk

注 : クロックはクロックピンから入力し、gclkに設定。  
またはPLLを使用する場合はPLLピンから入力。  
PLLピンでPLL未使用の場合、通常ピンから入力したのと同じ状態となり非推奨。ジッター増加などの原因となる。










# パッケージピンの定義

演習

- Interface Designerで設定
  - GPIO Resource Assigner  クリック
    - GPIO:Interface Viewタブで右表を入力

Instance	Resource
clk	GPIOL_16
led[0]	GPIOR_07
led[1]	GPIOR_08
led[2]	GPIOR_10
led[3]	GPIOR_11
reverse_n	GPIOR_06
rst_n	GPIOR_05

ここに入力

Resource Assigner							
GPIO : Instance View							
			all	all	all	all	
Instance	Package Pin	Resource	I/O Bank	Alt Conn	Features	Clock Region	Pad
 clk	C2	GPIOL_16	1B	GCLK	None	L1	GPIOL_16_CLK2
 led[0]	C7	GPIOR_07	2A	None	None	R1	GPIOR_07
 led[1]	A8	GPIOR_08	2A	None	None	R1	GPIOR_08
 led[2]	A9	GPIOR_10	2A	None	None	R1	GPIOR_10
 led[3]	B8	GPIOR_11	2A	None	None	R1	GPIOR_11
 reverse_n	C6	GPIOR_06	2A	None	None	R1	GPIOR_06
 rst_n	B6	GPIOR_05	2A	None	None	R1	GPIOR_05

# パッケージピンの定義のつづき

## 演習

### ■Interface Designerの設定の続き

- Check Design  クリック
  - エラーが無いことを確認

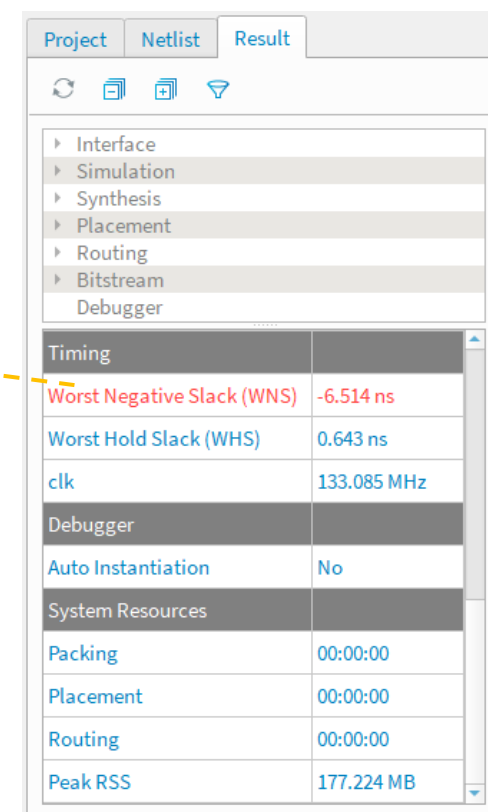
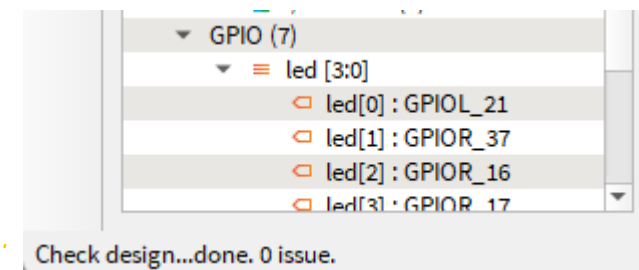
Check Designで自動

- Save  をクリック Saveされている
- Interface Designer をクローズ

### ■Synthesis をクリック

- エラーはResultタブのTimingエラーのみとなる

### ■Efinityをクローズ





# Interface Designerを 使用しないピン定義 Unified Netlist Flow

**Unified Netlist Flow**

# Unified Netlist Flowとは

## ■Unified Netlist Flow

- Interface Designerの設定を自動で生成
  - 汎用I/Oのみサポート
  - PLL、OSCも対応

# ピン仕様

演習

T8F81C2のResource名	Coreピン名	追加の設定
GPIOR_07	led[0]	--
GPIOR_08	led[1]	--
GPIOR_10	led[2]	--
GPIOR_11	led[3]	--
GPIOR_05	rst_n	Schmitt Trigger, Weak Pullup
GPIOR_06	reverse_n	Schmitt Trigger, Weak Pullup
GPIOL_16 (PLL不使用の場合) または、GPIOL_20 (PLL使用の場合)	clk	--

## ■Efinityでプロジェクトを開く

- 手順はコンパイルの章を参照

## ■エラーを確認

- Resultタブ → Placement → count16sec.place.rpt を開きUnassigned Core Pinsを確認

# ピン設定

## 演習

### ■I/Oピンをファイルで指定する場合の設定

- File→Edit Project→Synthesisタブ→Synthesis Options
  - peri-syn-instantiation : 1
  - peri-syn-inference : 1
- [OK]、[OK] でProject Editorをクローズ

### ■I/Oピンの設定ファイルを作成

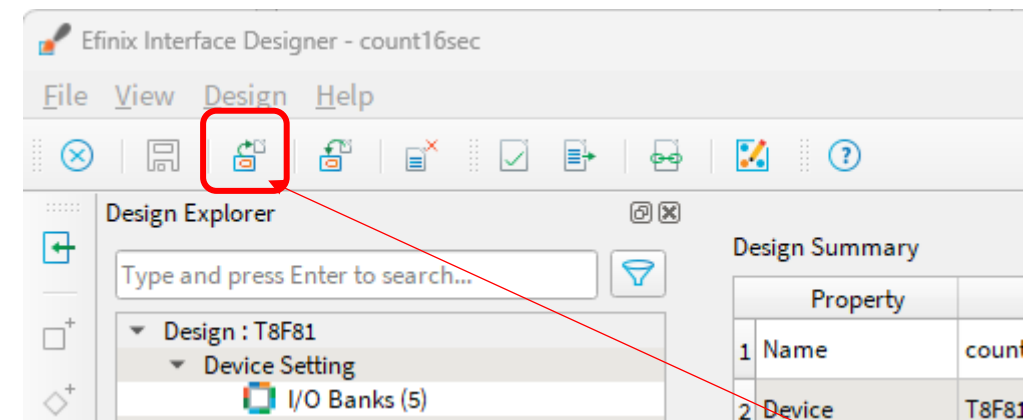
- Projectタブ→ISF Infoの右クリック→Create
  - Filename : count16sec
- [OK]をクリック
- count16sec.isfに以下の内容を記述してCtrl-Sで保存

# Set resource assignment **ピン番号**

```
design.assign_pkg_pin("led[0]","C7")
design.assign_pkg_pin("led[1]","A8")
design.assign_pkg_pin("led[2]","A9")
design.assign_pkg_pin("led[3]","B8")
design.assign_pkg_pin("clk","C2")
design.assign_pkg_pin("reverse_n","C6")
design.assign_pkg_pin("rst_n","B6")
```

# Set property, non-defaults **ピンの追加の設定**

```
design.set_property("reverse_n","SCHMITT_TRIGGER","1")
design.set_property("reverse_n","PULL_OPTION","WEAK_PULLUP")
design.set_property("rst_n","SCHMITT_TRIGGER","1")
design.set_property("rst_n","PULL_OPTION","WEAK_PULLUP")
```



記述は前章のInterface DesignerのExport Designで生成される.isfファイルを参考にする



# ピン設定のつづき

## ■Synthesis をクリック

- エラーはResultタブのTimingエラーのみとなる


## ■Efinityをクローズ


演習


Project


Netlist

Result









▶ Interface

▶ Simulation

▶ Synthesis

▶ Placement

▶ Routing

▶ Bitstream

Debugger

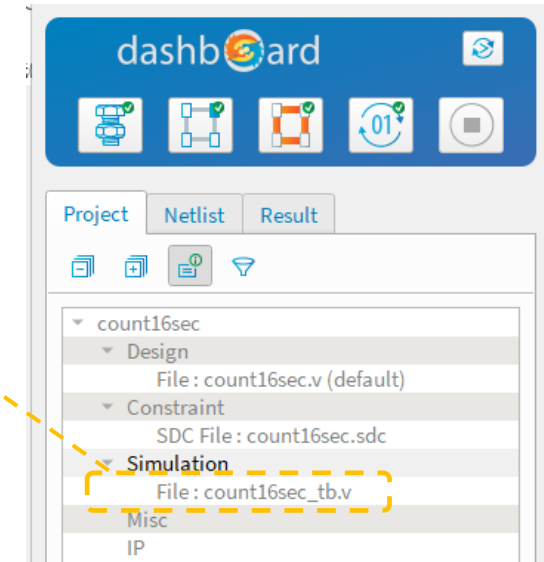
Timing	
Worst Negative Slack (WNS)	-6.514 ns
Worst Hold Slack (WHS)	0.643 ns
clk	133.085 MHz
Debugger	
Auto Instantiation	No
System Resources	
Packing	00:00:00
Placement	00:00:00
Routing	00:00:00
Peak RSS	177.224 MB

# シミュレーション

# RTLシミュレーション

## 演習

- Windowsでテストベンチファイルをコピー
  - count16sec\_tb.v を C:¥Efinity¥2024.2¥project¥count16sec¥ にコピーする
- Efinityにテストベンチを登録
  - Efinityでプロジェクトを開く (手順はコンパイルの章を参照)
  - Projectタブ → Simulation を右クリック → Add → count16sec\_tb.v を選択
  - EfinityをExitする
- Windowsの検索で
  - cmd.exe と入力しコマンドプロンプトを開く
- コマンドプロンプトで入力
  - プロジェクトフォルダーへ移動
    - > cd C:¥Efinity¥2024.2¥project¥count16sec
  - 環境変数を設定
    - > C:¥Efinity¥2024.2¥bin¥setup.bat
  - RTLシミュレーションの実行
    - **Interface Designer Flow**
      - > efx\_run.bat count16sec.xml --flow rtlsim
    - **Unified Netlist Flow**
      - > efx\_run.bat count16sec.xml --flow ptsimrtl --un\_flow



```
C:¥Efinity¥2022.2¥project¥count16sec>efx_run.bat count16sec.xml --flow rtlsim
Running: efx_run_sim.py count16sec --sim rtl --family Trion --device T8F81 -v count16sec.v,t:default --output_dir outflow
simrtl : PASS
```

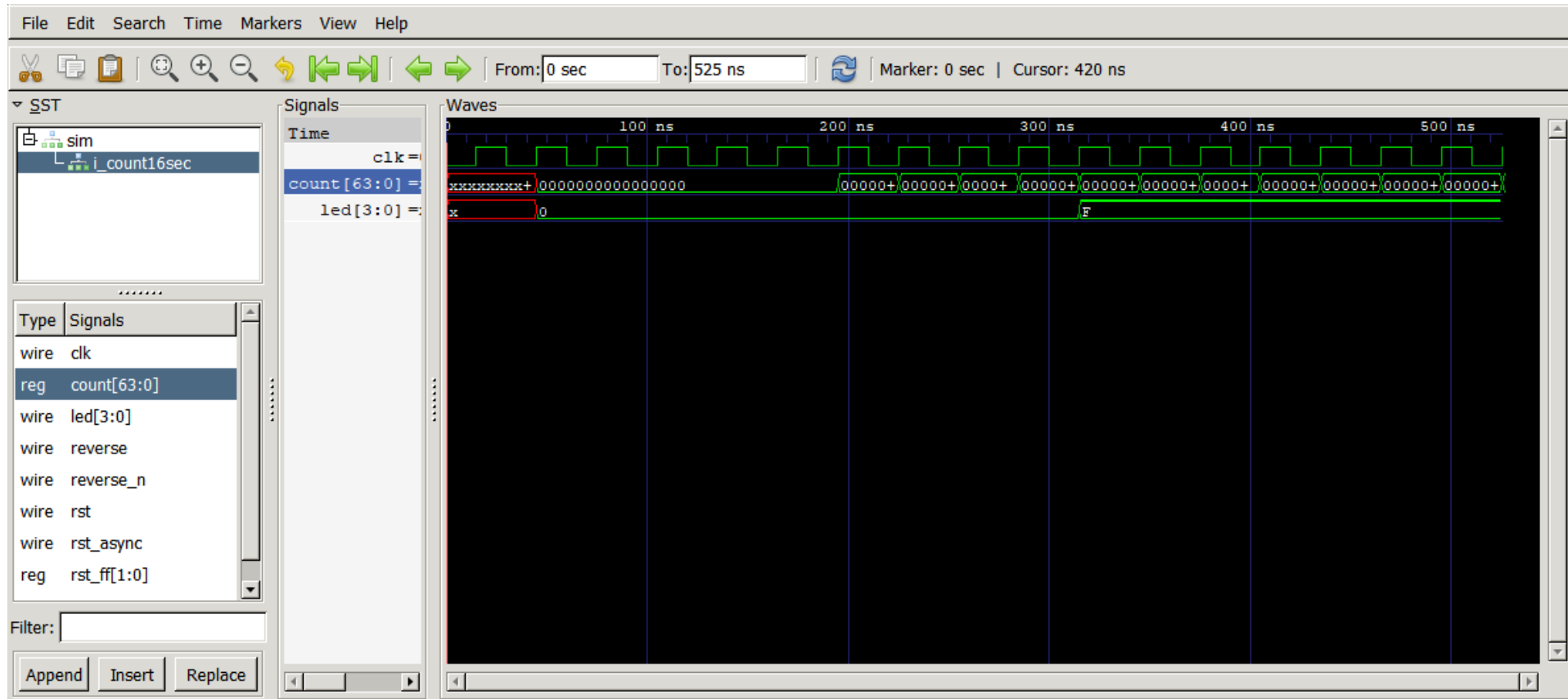
- 実行ログを確認
  - outflow¥count16sec.log

# シミュレーション

## ■ コマンドプロンプトで入力

### – 波形の表示

- > `gtkwave outflow¥count16sec.vcd`





# ネットリストレベルシミュレーション

## 演習

### ■ ネットレベルシミュレーション

- ネットリストレベルシミュレーションの実行（コマンドプロンプト）
  - **Interface Designer Flow**
    - > efx\_run.bat count16sec.xml --flow mapsim
    - > gtkwave outflow¥count16sec.vcd
  - **Unified Netlist Flow**
    - count16sec\_tb.vの内容をcount16sec\_tb\_ptsimfc.vで上書きする
    - > efx\_run.bat count16sec.xml --flow ptsimfc --un\_flow
    - > gtkwave outflow¥count16sec.vcd

### ■ 注意点

- シミュレーションはVerilogのみ対応
- iVerilogのSystemVerilogの対応は"logic"の信号記述のみ

iVerilogの制限  
Efinityの論理合成はSystemVerilogに対応

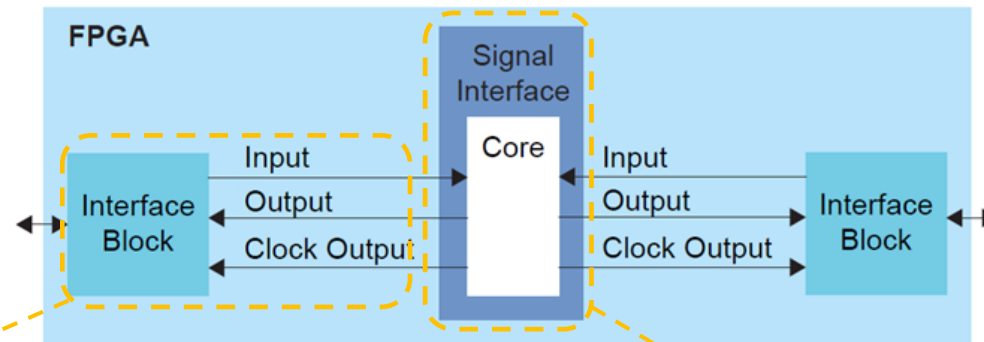
### ■ ModelSimを使用する場合

- ModelSimのwin32ディレクトリをWindowsのPathに追加する
- --modelsimオプションを使用するとModelsimでシミュレーション可能
  - **Interface Designer Flow**
    - efx\_run.bat count16sec.xml --flow rtlsim --modelsim
  - **Unified Netlist Flow**
    - efx\_run.bat count16sec.xml --flow ptsimrtl --un\_flow --modelsim
- <Modelsimインストールフォルダ>\modelsim.ini を修正
  - [vlog]の次の行にsvlog =1を追加(SystemVerilog使用の場合)

自動生成されたスクリプトを実行している  
<Project>¥work\_sim¥RUNMSIM\_count16sec

# タイミング解析

# タイミングレポート



## ■ インタフェースブロックの遅延

– count16sec.pt\_timing.rpt

Non-registered GPIO Configuration:

Instance Name	Pin Name	Parameter	Max (ns)	Min (ns)
clk	clk	GPIO_IN	1.954	0.698
reverse_n	reverse_n	GPIO_IN	1.954	0.698
rst_n	rst_n	GPIO_IN	1.954	0.698
led[0]	led[0]	GPIO_OUT	5.361	1.915
led[1]	led[1]	GPIO_OUT	5.361	1.915
led[2]	led[2]	GPIO_OUT	5.361	1.915
led[3]	led[3]	GPIO_OUT	5.361	1.915

インターフェースブロックのタイミングは別ファイルにレポートされる

## ■ Coreの遅延

– count16sec.timing.rpt

Launch Clock Path name	model name	delay (ns)	cumulative delay (ns)
clk	inpad	0.000	0.000
clk	inpad	0.150	0.150
clk	net	1.167	1.317
CLKBUF__1 1	gbuf	1.185	2.502
CLKBUF__1 0	gbuf	0.000	2.502
clk~0	net	0.000	2.502
edb_top_inst/.../din~FF CLK	ff	0.000	2.502

SDCで解析するのはCore部分の遅延

# タイミング制約の基本

## ■制約はプロジェクトの初期段階から入れる

- モジュールレベル合成で十分なマージンを確保
- 全体の合成や配置配線にタイミング制約が無い場合は実行時間が長くなる

## ■基本のタイミング制約は4つ

- クロックの周期(全てのクロックに設定)
  - `create_clock -period 100 TCK`
- 異なるクロック間の非同期設定
  - `set_clock_groups -exclusive -group {CLK} -group {TCK}`
  - 異なるクロック間を同期設計する場合は要注意
- 入力ピンの遅延、出力ピンの遅延
  - `set_output_delay -clock CLK -max 0.111 [get_ports {TDO}]`
  - `set_output_delay -clock CLK -min 0.053 [get_ports {TDO}]`
  - `set_input_delay -clock CLK -max 0.321 [get_ports {UPDATE}]`
  - `set_input_delay -clock CLK -min 0.161 [get_ports {UPDATE}]`
- フォルスパスの指定(max delayより優先される)
  - `set_false_path -from rst_n`



# タイミング制約

## ■ 自動生成されるサンプルスクリプトを参考にする

- <プロジェクトフォルダー>%outflow%count16sec.pt.sdc
- コメントアウトされている設定は参考として使用
- クロックの設定は必須修正

注：サンプルSDCは配置配線または、Interface Designer の Generate Efinity Constraint Files ボタンで生成される

```
# set_input_delay -clock <CLOCK> -max <MAX CALCULATION> [get_ports {clk}]
# set_input_delay -clock <CLOCK> -min <MIN CALCULATION> [get_ports {clk}]
# set_input_delay -clock <CLOCK> -max <MAX CALCULATION> [get_ports {reverse_n}]
# set_input_delay -clock <CLOCK> -min <MIN CALCULATION> [get_ports {reverse_n}]
```

- JTAGなどのハードマクロの設定に関してはmin/max値はそのまま使用可能

```
# JTAG Constraints
#####
# create_clock -period <USER_PERIOD> [get_ports {jtag_inst1_TCK}]
# create_clock -period <USER_PERIOD> [get_ports {jtag_inst1_DRCK}]
set_output_delay -clock jtag_inst1_TCK -max 0.155 [get_ports {jtag_inst1_TDO}]
set_output_delay -clock jtag_inst1_TCK -min -0.053 [get_ports {jtag_inst1_TDO}]
```

自動生成されたタイミング  
は推奨動作条件の値。この  
まま使用する

# タイミング制約

## 演習


### ■count16sec.sdc をエディターで作成

- 「clk」 クロックの周期は「40」 ns (つまり25MHz)
- I/Oの入出力遅延は5ns(この値は仕様ではなく演習として設定してみる)

```
# 40ns@25MHz
create_clock -period 40 clk
# 5ns for all input/output pins
set_output_delay -clock clk 5 [ all_outputs ]
set_input_delay -clock clk 5 [ get_ports {reverse_n} ]
# Set false path to asynchronous reset
set_false_path -from rst_n
```

delay設定値はインターフェースブロックまでのタイミング。  
SDC作成時にインターフェースブロックの遅延を加味して値を決める

### ■Efinityでプロジェクトを読み込む

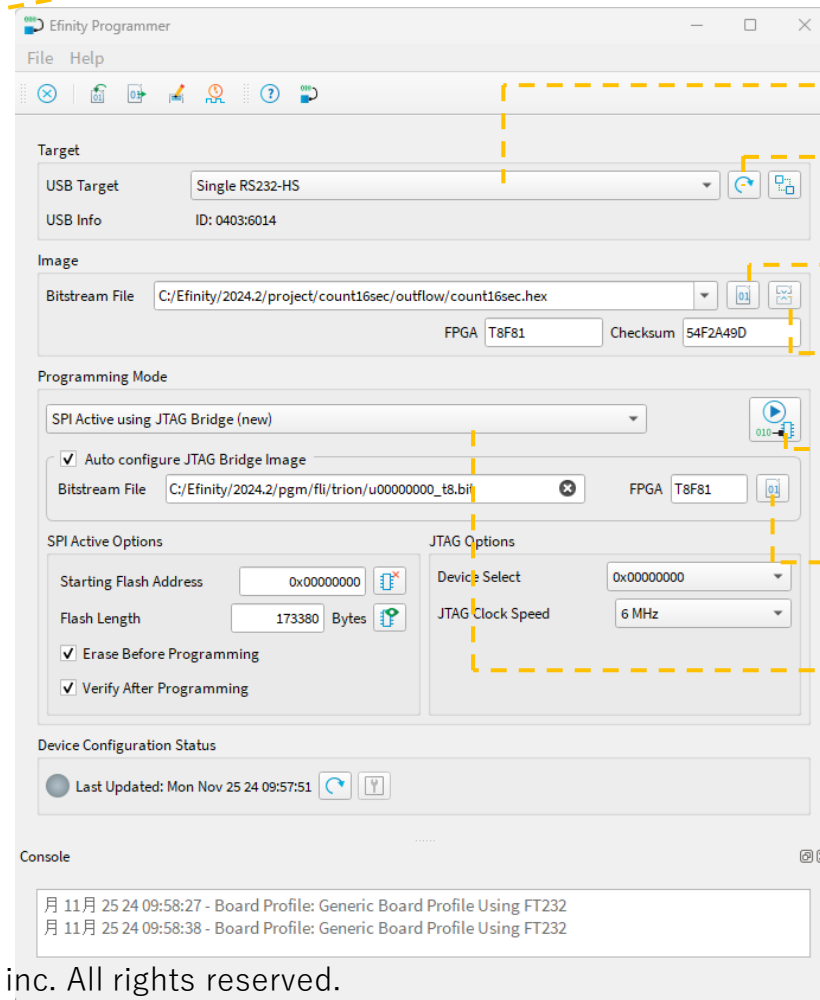
- Projectタブ → Constraint右クリック → Add
  - File Name : count16sec.sdc
- Synthesis  をクリック
  - Resultタブのエラーなしを確認する
- Efinityをクローズ

#### Unified Netlist Flow

Unified Netlist Flowの場合のみ、  
count16sec.peri.xmlが既にあるとエラー  
となるのでコンパイル前に削除する

# 書き込み

# Programmer



数字の順番に操作する

② ターゲットの選択

① ターゲットの検出

⑤ ファイルの選択 SPI Active : .hexファイル  
JTAG : .bitファイル

複数イメージの結合 (オプション)

⑥ 書き込み 最大4つのイメージをCBSEL[1:0]ピンで  
切り替えて使用可能

④ JTAG Bridgeファイルの選択

③ 書き込みモード ROM書込 : SPI Active using JTAG Bridge(new)  
FPGA直接 : JTAG





注2 : SPI Active x2 x4の設定はプロジェクトの設定で行う



# Programmer - JTAG

## 演習

## ■JTAGで書き込みを行う





- Efinityでプロジェクトを読み込む
  - Programmer  をクリック
- Refresh USB Target  をクリック
  - USB Target にSingle RS232-HSが検出されていること
- Programming Mode : JTAG
- Select Image File  をクリック
  - ファイル名 : outflow/count16sec.bit
- Start Program  をクリック
  - Device is in user mode! と表示され、動作を開始する
- USBを抜く→刺すで電源をトグルする
  - プログラムする前の状態に戻る

注1 : Zadigでドライバーをインストールしておくこと

# Programmer - SPI Active using JTAG Bridge(new)

## 演習

## ■SPI Active using JTAG Bridge(new)で書き込みを行う

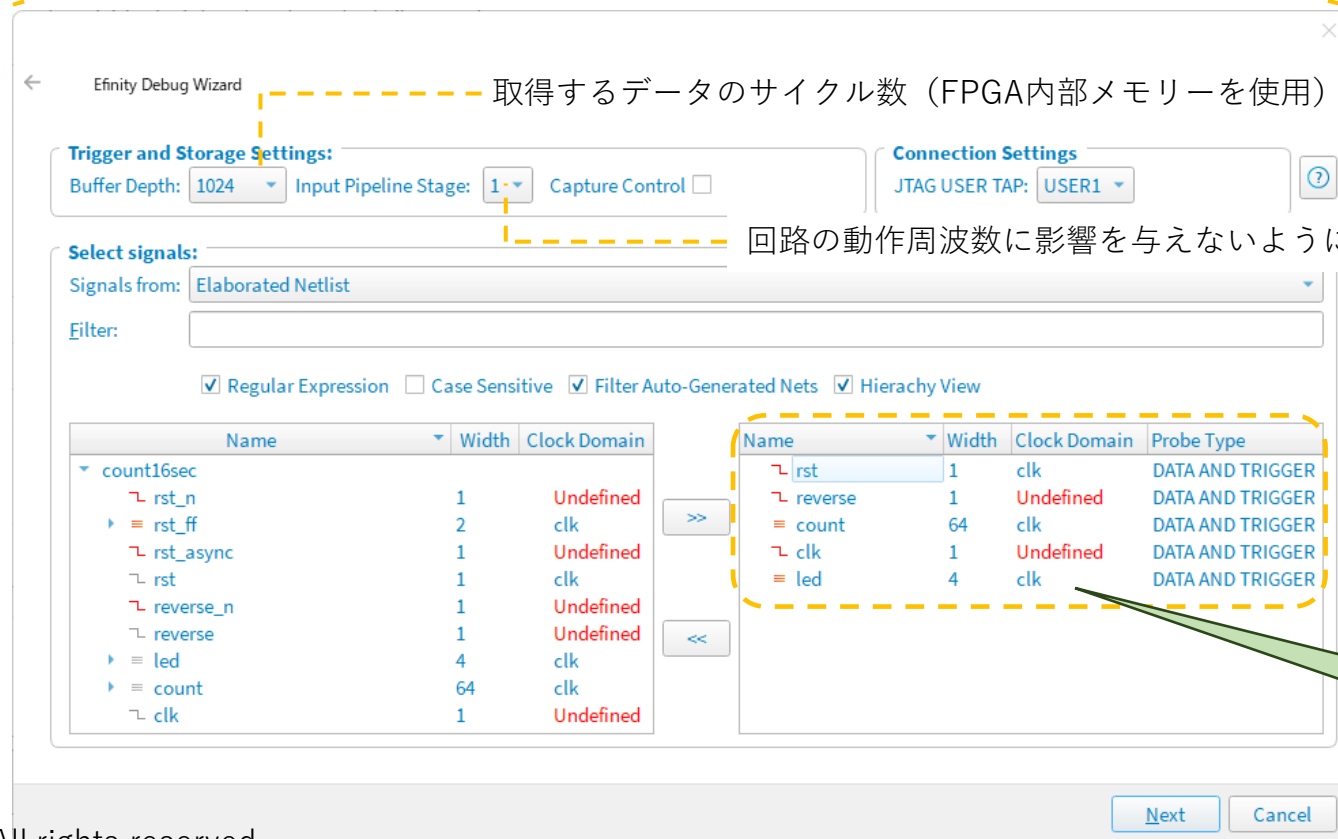
- Refresh USB Target  をクリック
  - USB Target にSingle RS232-HSが検出されていることを確認
- Programming Mode : SPI Active using JTAG Bridge(new)
- Image → Select Image File  をクリック
  - ファイル名 : outflow/count16sec.hex
- Auto configure JTAG Bridge Image → Select Image File  をクリック
  - ファイル名 : C:¥Efinity¥2024.2¥pgm¥fli¥trion¥u00000000 t8.bit
- Start Program  をクリック
  - メッセージに以下のように表示される
    - 木 12月 21 23 10:13:51 - JTAG2SPI programming...done
- USBを抜く→刺すで電源をトグルする
  - 正常動作を開始する

# デバッグ

# Debug Wizard (ロジックアナライザ機能)



2024.2+patch 2024.2.294.1.19では  
Unified Design Flowでは異常終了する。  
修正パッチをお待ちください。





# Debug Wizard

## デバッグスタート ⑥

## デバッグスタート 即時キャプチャー

## 信号の追加⑤

## 信号の削除

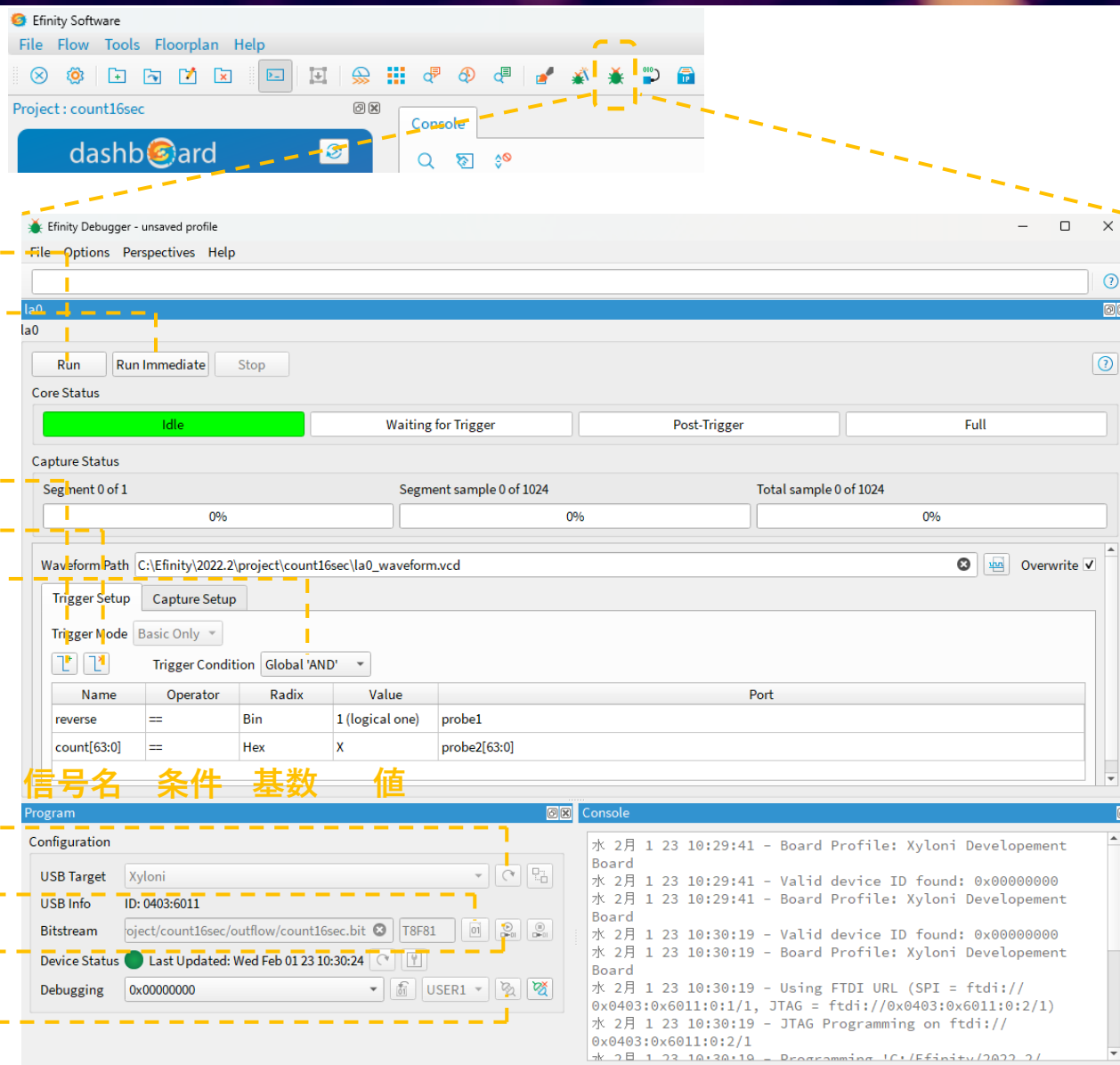
## 信号のAND/OR条件

## USB Target検出①

## .bitファイル選択 ②

## プログラミング ③

## デバッガ接続 ④



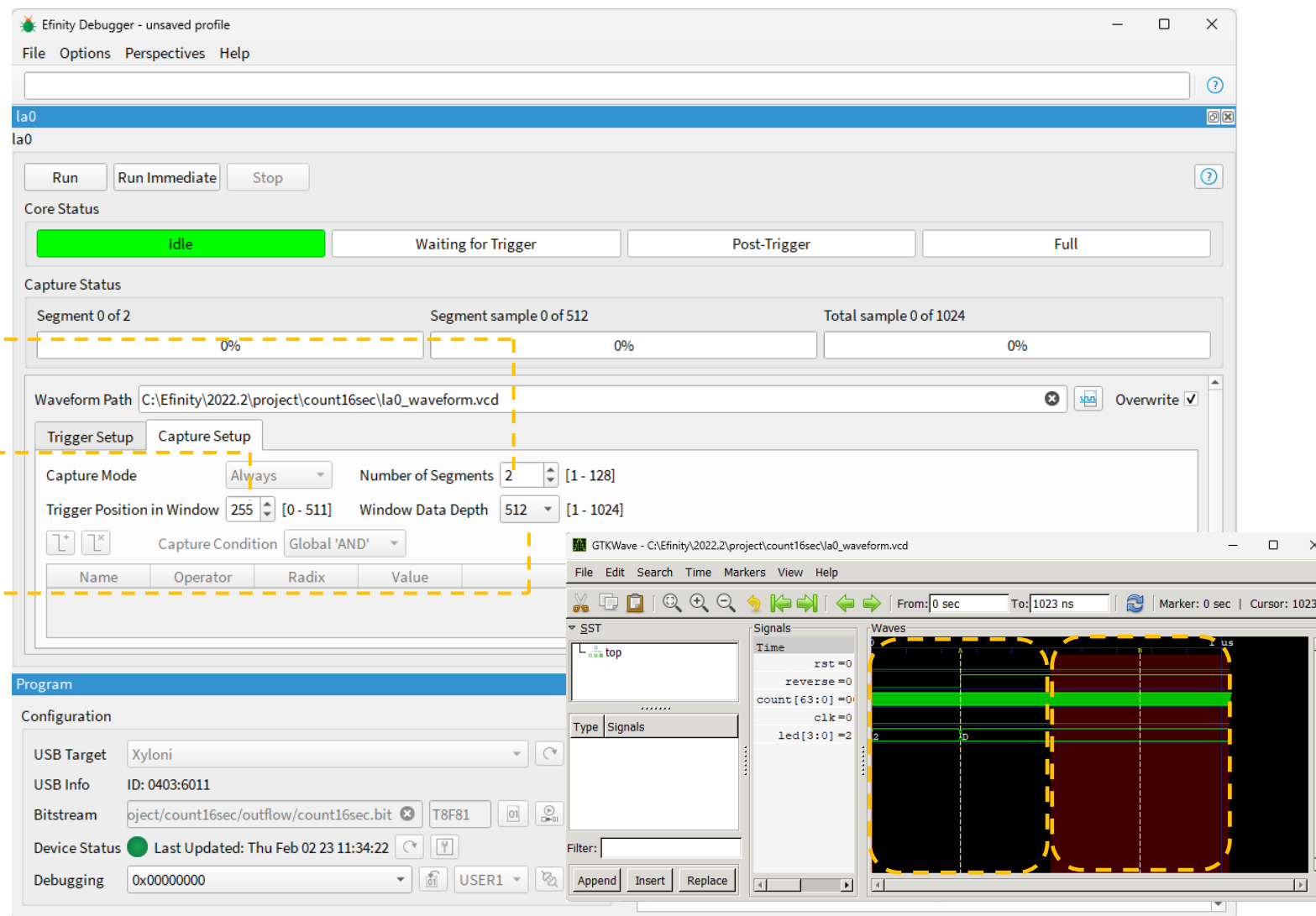
# Debug Wizard

取得回数を2以上にすると  
連続複数回の取得が可能

データ取得回数

トリガー前の表示  
サイクル数

1回で取得する  
データサイズ



# Debug Wizard


## ■デバッガを取り除く場合

- EfinityのFileメニュー → Edit Project
- Project Editor窓
  - Debuggerタブ → Debug Auto Instantiation チェックOff

注：デバッガを入れたままクロック系の修正を行うとエラーが発生する場合があります。不明なエラーが発生する場合はデバッガをOffにしてコンパイルをお試してください

# Debug






## ■ EfinityのDebug Wizard をクリック

- 左側の led、count、reverse\_n 信号を  ボタンで右側に追加
- Clock Domainの **Undefined** をダブルクリックして clk に変更
- [Next]、[Finish]

## ■ EfinityのSynthesis をクリック

- コンパイルの正常終了を確認

## ■ EfinityのDebugger をクリック

- Refresh USB Target  をクリック
  - USB TargetにSingle RS232-HSが検出される
- Select Image File  をクリック
  - outflow/count16sec.bit を選択
- Start Programming  をクリック
  - 書き込みの正常終了を確認
- Connect Debugger  をクリック
  - Core Statusが **Idle** に代わる
- Add trigger condition  をクリック
  - reverse\_n を選択し [OK] をクリック
- reverse の Value が 0 であることを確認
- [Run] クリック
- ブレッドボードの reverse\_n スイッチを押下
  - 波形が表示される

演習



# Appendix

# タイミング制約その2

# タイミング制約 (デバッガ追加後)

オプション

## ■サンプルファイルの「JTAG」の部分を設定に追加

- サンプルファイル : outflow¥count16sec.pt.sdc
- こちらに設定追加 : count16sec.sdc

## ■JTAGのタイミング制約を設定

- 「jtag\_inst1\_TCK」 クロックの周期を 「66」 ns (つまり15MHz)に設定
- タイミング制約の章を参考に設定しましょう

## ■False Path指定を追加してみましょう

- 「clk」と「jtag\_inst1\_TCK」は非同期パスになります

```
# False Path
#####
set_clock_groups -exclusive -group {clk} -group {jtag_inst1_TCK}
```



# PLLを使用する





# PLLを使用する - Interface Designerの場合

オプション

## ■ 25MHzのclkをPLLで周波数を12.5MHzに変えて使用する

## ■ Interface Designerの

- PLLを右クリック → Create Block
  - Instance Name : **pll** ユニークな名前
  - PLL\_Resource : PLL\_0 Reference Clock ResourceにGPIO\_L\_20と表示される
  - [Automated Clock Calculation]をクリック
  - PLL Clock Calculatorで
    - Input Pin : 25.0 MHz
    - Output Pin : 12.5 MHz, **clk** RTLのクロックと同じ名前
    - [Finish]
- Check Design  クリック
  - GPIO\_L\_20にエラーがある
- Interface DesignerのDesign Explorer → GPIO (7) → clk:GPIO\_L\_16をクリック
  - Block Editor
    - Instance name : **pll\_refclk** ユニークな名前
    - Connection Type : pll\_clk\_in
- Instance Viewで
  - pll\_refclkのResourceをGPIO\_L\_20に変更
- Check Design  クリック
  - 0 issueで正常終了
- Interface Designerを閉じる

基板配線はclk専用ピンとpll\_refclk専用ピンの両方に同じクロックが入っている

## ■ Efinityでコンパイルして書き込み

- LEDの点滅速度が1/2になる

# PLLを使用する - Unified Netlist Flowの場合

オプション

■ 25MHzのclkをPLLで周波数を12.5MHzに変えて使用する

■ ソースコードにPLLを記述

- count16sec.vを右のように修正

■ count16sec.isfファイルの該当行を修正

- 修正前 : design.assign\_pkg\_pin("clk","C2")

- 修正後 : design.assign\_pkg\_pin("pll\_refclk","C3")

■ Efinityでコンパイルして書き込み

- LEDの点滅速度が1/2になる

N,M,O, CLKOUT0\_DIVの設定値は  
前のページのInterface Designerで  
確認してコードに反映させる

チュートリアルではPLLにリセット  
を入れていませんがロック時間が  
データシートの規定以上となります  
のでPLLのリセットは必ず行ってく  
ださい。

```
module count16sec (
    // Coreのピンを記述(デバイスのピンではない)
    input pll_refclk, // クロック
    input rst_n,      // リセット(負論理)
    input reverse_n,  // カウント逆転(負論理)
    output [3:0] led   // LED
);

logic [63:0] count; // カウント値
logic [1:0] rst_n_ff; // リセット同期化
wire rst_n_sync;    // リセット同期後

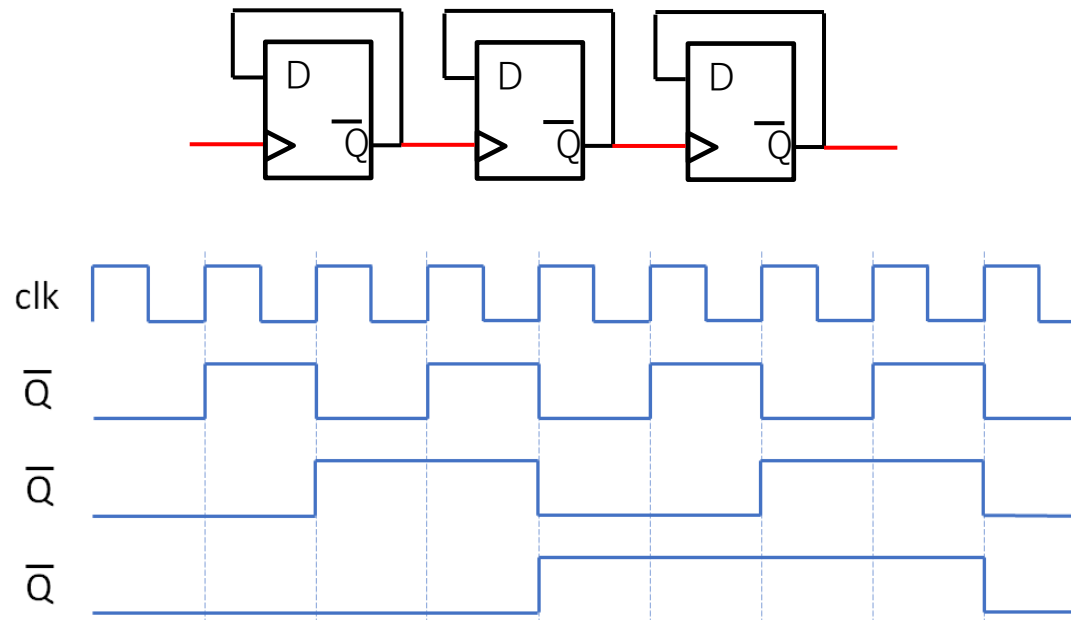
// Unified Netlist Flow: Instantiate EFX_PLL_V1
wire clk;
EFX_PLL_V1 #(
    .N(1),
    .M(32),
    .O(1),
    .CLKOUT0_DIV(64),
    .CLKOUT1_DIV(2),
    .CLKOUT2_DIV(2),
    .REFCLK_FREQ(25.00)
) pll_inst1 (
    .CLKIN(pll_refclk),
    .RSTN(1'b1),
    .CLKOUT0(clk)
);

// リセット同期化回路(非同期リセットはアサート非同期、ネゲート同期の原則)
always @(posedge clk or negedge rst_n) begin
    if(~rst_n) begin
```

# リプルカウンタ

# リプルカウンタ

- 1 / 2 分周カウンタの出力を 1 / 2 分周カウンタのクロックに接続して分周を行う方式。FPGAではクロック本数が増加するため非推奨回路となっている



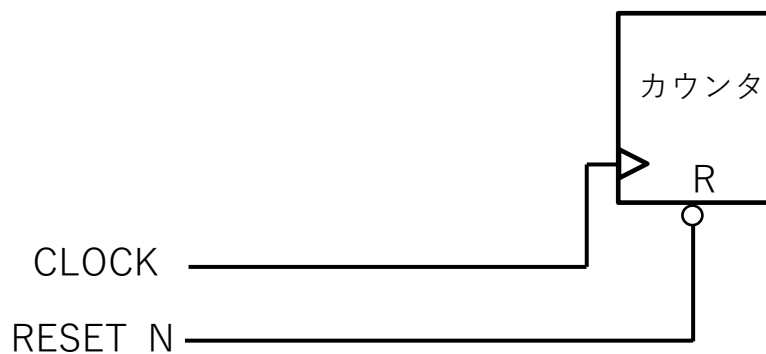


# リセット回路

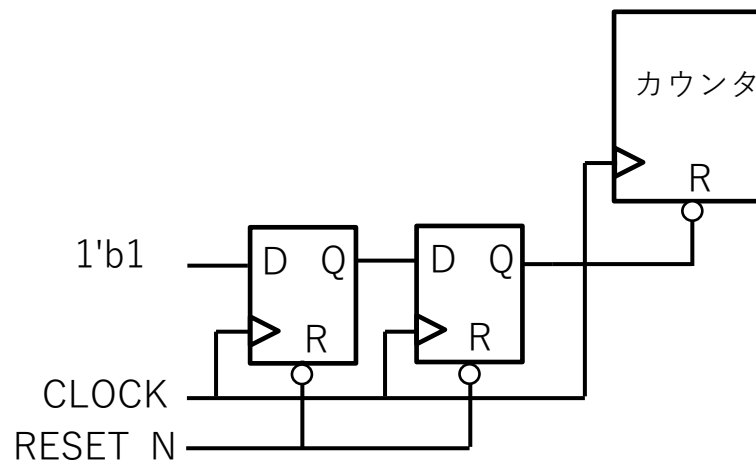
# リセット回路

## ■非同期リセット端子のルール

- アサートは非同期、ネゲートは同期



非同期にリセットを解除するとカウンタの各ビットがばらばらにスタートする



リセットの解除を同期にすると正常に動作する

# その他情報



# その他情報

■ <https://www.efinixinc.com/support/>

## ■データシート

– Support → Support Home → Documentation

## ■ハードウェア デザイン チェックリスト（基板出図に必須のチェック）

– Support → Support Home → Board Design

## ■パッケージ ユーザーガイド（パッケージの寸法図、ピンの説明）

– Support → Support Home → Documentation

## ■消費電力見積もり

– Support → Support Home → Board Design

## ■サンプルコード

– Support → Support Home → Knowledgebase

## ■サンプルデザイン

– Support → Support Home → Examples

デバイスとパッケージを設定すると  
プロジェクト専用のチェックリスト  
が生成される



# サポートに関して

- 本資料は秋月電子通商と(株)エクスプローラのコラボレーションで公開させていただいております
- サポート窓口はご購入された販売店となる旨をご了承お願いいたします
- Efinix社製品の商用でのご購入は(株)エクスプローラにお問い合わせいただけますと幸いです

おつかれさまでした

---