SUZAKU
スターターキットガイド
（FPGA 開発編）

Version 2.0.2

2006 年 8 月 23 日

株式会社アットマークテクノ
http://www.atmark-techno.com/

SUZAKU 公式サイト
http://suzaku.atmark-techno.com/

目次

表目次

図目次

例目次

1.はじめに

この度は、『SUZAKU スターターキット』をお買い上げいただきありがとうございます。

本スターターキットは、FPGA 搭載ボード『SUZAKU』を初めて手に取る方にもお使いいただけるよう、第一歩を踏み出すために必要な機材をセットにした学習用キットです。

『SUZAKU』は FPGA【Field Programmable Gate Array】を搭載した組み込み機器開発ボードです。FPGA とは簡単にいうとプログラミングすることができる LSI のことで、さまざまな設計データを送り込んで再構築させることが可能なデバイスです。

この FPGA は近年、より大規模化・低価格化してきています。現在では容易に入手できる FPGA ひとつで、内部にプロセッサと複数の必要な周辺回路を同時に構成するといったことが可能となっています。例えば UART がいくつも欲しい、GPIO ポートが大量に必要だ、画像処理を高速に行うための回路を投入したい…さらに、プロセッサを 2 つ持ちたいといった場合ですら、回路規模が許す限り自由に構成させることが可能なのです。

『SUZAKU』は、この FPGA の利点を最大限に生かすべく誕生した小型 FPGA ボードです。
『SUZAKU』の特徴を以下に挙げます。

- ○ 固定された外部インターフェースとして、Ethernet と RS-232C を持っています。
- ○ マイクロコンピュータボードとして動作するために必要な要素であるクロック、DRAM、Flash、Ethernet MAC&Phy、RS-232C ドライバ/レシーバが、基板上に実装されています。
- ○ 電源は+3.3V 単一入力です。内部に FPGA 用の電源である 2.5V&1.2V を作る回路が組み込まれています。また、SUZAKU 自身で FPGA を再コンフィギュレーション可能にするための回路が組み込まれています。
- ○ 基板の外周に沿って 86 個（SZ310 は 70 個）の空きピンが備えられています。これらはすべて FPGA の I/O ピンに結線されており、外部デバイスや装置との接続のため自由に使用することができます。
- ○ FPGA の中ではソフトプロセッサ[MicroBlaze]もしくはハードプロセッサ[PowerPC405]が動いています。
- ○ Flash の中には、OS[Linux]、Ethernet などのデバイスドライバ、アプリケーション群が書き込まれており、電源を入れるだけでこれらを利用することができるようになっています。
- ○ 高機能である Linux を使用しながら、同時にリアルタイム処理を行うような用途向けに構成することも可能です。基板上には SDRAM が 2 枚実装されており、これらを FPGA 内に構成した 2 つの CPU から独立して使用させることができるため、片方で Linux を、他方でリアルタイム OS を動作させる、といった使い方ができます。（※SZ130-U00 のみ）

以上のように『SUZAKU』は、FPGA が持つ柔軟性と、Linux が持つ高機能性、豊富なソフトウェア資産…これらの利点を同時に享受することができるプラットフォームです。これらの特徴を利用することにより、旧来の開発手法に比べて開発期間を短縮し、コストダウンを実現することが可能になるでしょう。

『SUZAKU』上での開発作業の流れは、
① FPGA 開発
② ソフトウェア開発
の 2 段階に大きく分けることができます。本書ではこのうち①FPGA 開発について、実際に『SUZAKU スターターキット』を使用しながら解説していきます。

第 1 段階として取り上げる題材は、単色 LED を 1 つだけ点灯する簡単な回路作成です。その後カウンタ回路、単色 LED 順次点灯回路、7 セグメント LED デコーダ回路、ダイナミック点灯回路…といった流れで、だんだんとステップアップした回路を作成していきます。最終的にはこれらの回路を一まとめにして自作の IP コアとし、MicroBlaze から制御させて「スロットマシン」を実現することが目標です。

「スロットマシン」を実現させるまでには、FPGA 開発する上で必要な基礎知識、ISE や EDK といった専用開発ツールの使い方、VHDL 言語の記述方法、FPGA 上に搭載される「Microblaze」の使用方法…といった様々な内容が登場します。これらの内容から FPGA と『SUZAKU』の効果的な使い方を学んでいただけたるのではないかと思います。

解説の最後では、「スロットマシン」実現のため最低限必要なソフトウェアプログラミングまでを取り上げます。ソフトウェア開発のより詳しい内容については、本書と対となる「SUZAKU スターターキットガイド（ソフトウェア開発編）」を発行予定です。

本書を足掛かりとして、SUZAKU 開発者のスペシャリストを目指していただければ幸いです。

2. 注意事項

2.1. 安全に関する注意事項

SUZAKU スターターキットを安全にご使用いただくために、特に以下の点にご注意くださいますようお願いいたします。

本製品には一般電子機器用（OA機器・通信機器・計測機器・工作機械等）に製造された半導体部品を使用していますので、その誤作動や故障が直接生命を脅かしたり、身体・財産等に危害を及ぼす恐れのある装置（医療機器・交通機器・燃焼制御・安全装置等）に組み込んで使用したりしないでください。また、半導体部品を使用した製品は、外来ノイズやサージにより誤作動したり故障したりする可能性があります。ご使用になる場合は万一誤作動、故障した場合においても生命・身体・財産等が侵害されることのないよう、装置としての安全設計（リミットスイッチやヒューズ・ブレーカ等の保護回路の設置、装置の多重化等）に万全を期されますようお願い申しあげます。

2.2. 取り扱い上の注意事項

劣化、破損、誤動作、発煙、発火の原因となることがあります。取り扱い時には以下のような点にご注意ください。

- **入力電源**
  5V+5%以上の電圧を入力しないでください。極性を間違わないでください。SUZAKU の＋3.3V 外部入力(CON6)に電源を供給しないでください。

- **インターフェース**
  各インターフェース(外部 I/O、RS-232C、Ethernet、JTAG)には規定以外の信号を接続しないでください。また、信号の極性を間違う、信号の入出力方向を間違う等しないでください。

- **改造**
  コネクタ等を増設する以外の改造は行わないでください。

- **FPGA プログラム**
  周辺回路(ボード上の部品も含む)と信号の衝突(同じ信号に 2 つのデバイスから出力する)を起こすようなFPGA プログラムを行わないでください。FPGA のプログラムを間違わないでください。

- **電源の投入**
  本ボードや周辺回路に電源が入っている状態では絶対に FPGA I/O、JTAG 用コネクタの着脱を行わないでください。

- **静電気**
  本ボードには C-MOS デバイスを使用していますので、ご使用になるまでは帯電防止対策のされている、出荷時のパッケージ等にて保管してください。

- **ラッチアップ**
  電源および入出力からの過大なノイズやサージ、電源電圧の急激な変動等で、使用している C-MOS デバイスがラッチアップを起こす可能性があります。いったんラッチアップ状態となると、電源を切断しないかぎりこの状態が維持されるため、デバイスの破損につながることがあります。ノイズの影響を受けやすい入出力ラインには保護回路を入れる、ノイズ源となる装置と共通の電源を使用しない等の対策をとることをお勧めします。

● 衝撃、振動

落下や衝突などの強い衝撃を与えないでください。振動部や回転部などへの搭載はしないでください。強い振動や遠心力を与えないでください。

● 高温低温、多湿

極度に高温や低温になる環境や、湿度が高い環境では使用しないでください。

● 塵埃

塵埃の多い環境では使用しないでください。

2.3. FPGA 使用に関しての注意事項

● 本製品に含まれる FPGA プロジェクトについて

本製品に含まれる FPGA プロジェクト(付属のドキュメント等も含む)は、現状のまま(AS IS)提供されるものであり、特定の目的に適合することや、その信頼性、正確性を保証するものではありません。また、本製品の使用による結果について、なんら保証するものではありません。

本製品は、ベンダのツール(Xilinx 製 EDK、ISE やその他ベンダツール)やベンダの IP コアを利用し、FPGA プロジェクトの構築、コンパイル、コンフィギュレーションデータの生成を行っておりますが、これらツールに関しての販売、サポート、保証等は行っておりません。

2.4. ソフトウェア使用に関しての注意事項

● 本製品に含まれるソフトウェアについて

本製品に含まれるソフトウェア(付属のドキュメント等も含みます)は、現状のまま(AS IS)提供されるものであり、特定の目的に適合することや、その信頼性、正確性を保証するものではありません。また、本製品の使用による結果について、なんら保証するものではありません。

2.5. 本書に関しての注意事項

本書は SUZAKU スターターキット(SZ130-U00＋LED/SW ボード)を対象に書かれています。よってほとんどの図、例、記述等がSUZAKUスターターキット用となっております。SZ010-U00、SZ030-U00、SZ310-U00をお使いの場合は適宜それぞれに置き換えてお読みください。

3. 作業の前に

3.1. 必要なもの

SUZAKU スターターキットの場合は、箱に以下のものが収められています。ご確認ください。
それ以外の場合は以下のものが必要となるので、足りないものをそろえて下さい。

① SUZAKU※
② LED/SW ボード
③ CD-ROM
④ AC アダプタ 5V
⑤ D-sub9 ピン-10 ピン変換ケーブル
⑥ D-sub9 ピンクロスケーブル
⑦ スペーサ×4
⑧ ネジ×4
⑨ ジャンパプラグ×2

※本書対応の SUZAKU は SUZAKU-S（SZ010-U00、SZ030-U00、SZ130-U00）、SUZAKU-V（SZ310-U00）です。これ以降それぞれ型式の"-U00"を省略して表記します。

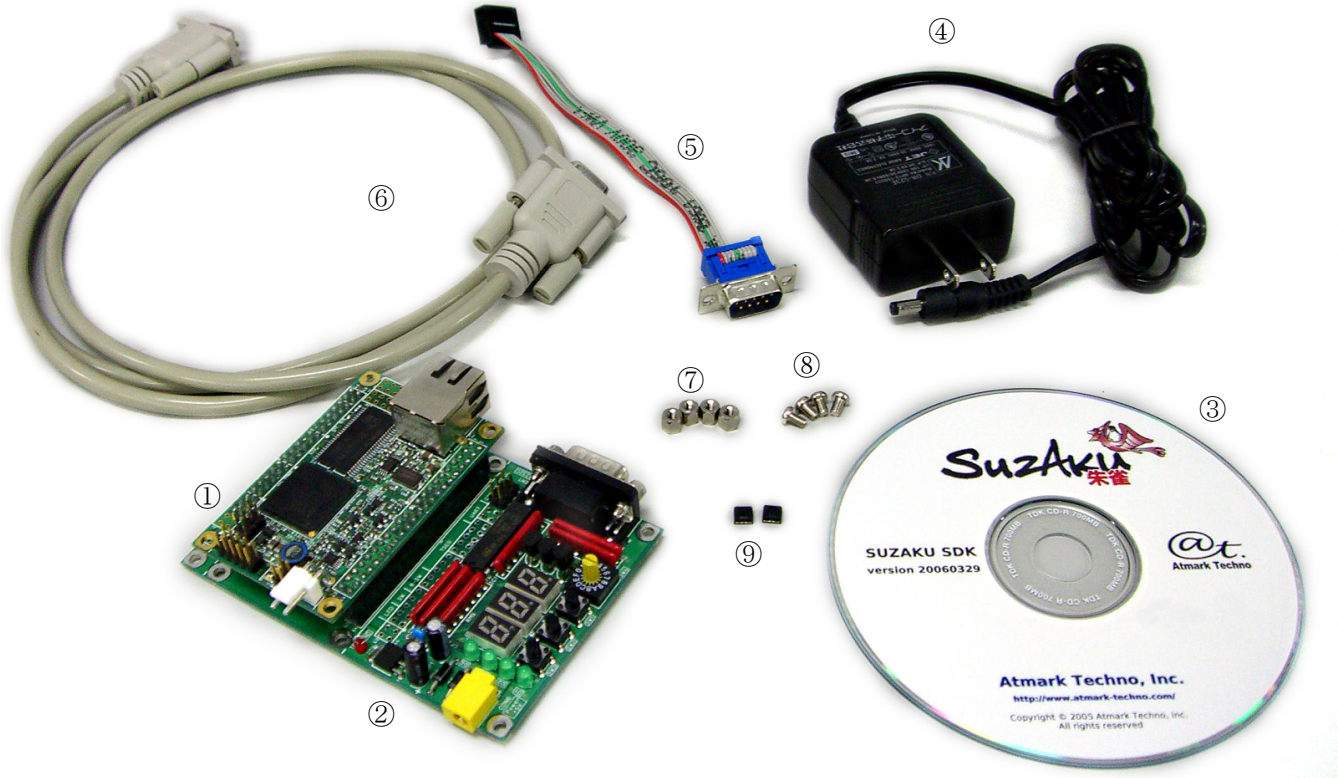


図 3-1 SUZAKU スターターキット

3.1.1. CD-ROM の内容

CD-ROM の内容を以下に示します。***は型式、x.x.x にはバージョンが入ります。

```
suzaku/
  ＋bootloader/
  ＋colinux/
  ＋cross-dev/
  ＋dist/
  －doc/
      sz010-u00_sz030-u00_hardware_manual_ja-x.x.x.pdf  -->SZ010 SZ030 ハードウェアマニュアル
      sz130-u00_hardware_manual_ja-x.x.x.pdf  -->SZ130 ハードウェアマニュアル
      sz310-u00_hardware_manual_ja-x.x.x.pdf  -->SZ310 ハードウェアマニュアル
      suzaku_software_manual_ja-x.x.x.pdf  -->SUZAKU ソフトウェアマニュアル
      uclinux-dist-developers-guide-x.x.pdf  -->uCLinux-dist Developers Guide
  －fpga_proj/
    －8.1i/
      ＋sz***/  -->SZ***のデフォルト
  ＋image/
  ＋sample/
  －tools/
      spi_writer.zip  -->SPI Writer
      LBPlay2_Release104  -->LBPlay2
suzaku-starter-kit/
      suzaku_starter_guide_fpga-x.x.x.pdf  -->SUZAKU スターターキットガイド（FPGA 開発編）
  －fpga/
      dynamic_ctrl.zip  -->ダイナミック点灯回路 ソース等
      le_seq_blink.zip  -->単色 LED 順次点灯回路 ソース等
      seg7_decorder.zip  -->7セグメント LED デコーダ回路 ソース等
      slot_counter.zip  -->カウンタ回路 ソース等
      slot_le.zip  --> not and or 回路 ソース等
      slot_c_source.zip  -->スロットマシン用 C 言語ソースコード
      opb_sil00h_v1_00_a.zip  -->スロットマシン IP コア（ハード版）
      opb_sil00u_v1_00_a.zip  -->スロットマシン自作 IP コア（ソフト版）
    －sz***-sil/  -->SZ010 のプロジェクトファイル
        sz***-add_slot-xxxxxxxx.zip  -->スロットマシン機能を追加したプロジェクトファイル
        sz***-add_uart_gpio-xxxxxxxx.zip  -->GPIO、UART を追加したプロジェクトファイル
  －doc/
      LED_SW_Board_parts.pdf  -->LED/SW 部品表
      LED_SW_Schematic.pdf  -->LED/SW 回路図
      sil00_hardware_manual_x.x.x.pdf  -->LED/SW ハードウェアマニュアル
      sil00_software_manual_x.x.x.pdf  -->LED/SW ソフトウェアマニュアル
  －image/
      fpga-sz***-sil.bin  -->スロット機能を追加した出荷時の fpga ファイル
      image-sz***-sil.bin  -->スロットの CGI を追加した出荷時の image ファイル
```

suzaku フォルダには SUZAKU に関するファイルが収められています。この中の fpga_proj フォルダには SUZAKU のデフォルトのプロジェクトが収められています。doc フォルダには SUZAKU の各種マニュアルが収められています。

suzaku-starter-kit フォルダには SUZAKU スターターキットに関するファイルが収められています。この中の doc フォルダには LED/SW ボードの各種資料、使用 IP コアのデータシートが収められ、fpga フォルダには本書で作成する VHDL ソースコード、ucf ファイル、プロジェクト、C 言語ソースコード等が収められています。

3.2. 開発環境

開発環境としては以下のソフトウェア、ハードウェアが必要です。

- **作業用 PC**
  Windows2000 または、WindowsXP が動作し、シリアルポート(1 ポート)、及びパラレルポート(1 ポート)を持つ PC を準備してください。
- **シリアル通信用ソフト**
  Tera Term(Pro)等のシリアル通信用ソフトを準備してください。Tera Term(Pro)はフリーソフトウェアのターミナルエミュレータで、シリアル通信等を行うことができます。Tera Term(Pro)には UTF-8 に対応しているバージョンもあります。
- **Xilinx ISE**
  Xilinx の ISE8.1i 以降(Foundation(有償版)、WebPACK(無償版)どちらでも可)を準備し、インストールしてください。無償版の ISE WebPACK は、Xilinx のホームページ(http://www.xilinx.co.jp/)からダウンロードできます。どちらでも本書内の開発は可能です。お好きな方をインストールしてください。インストール後ソフトウェアアップデートをしてください。※
- **Xilinx EDK**
  Xilinx EDK8.1i 以降(Embedded Development Kit)を準備し、インストールしてください。インストール後ソフトウェアアップデートをしてください。※
- **Xilinx Parallel CableⅢ、Ⅳまたはそれ相当**
  Parallel CableⅢ、Ⅳまたはそれ相当のものを準備してください。※

※ Xilinx 製品の詳細については、Xilinx のホームページ(http://www.xilinx.co.jp/)をご覧になられるか、Xilinx 代理店にお問い合わせください。

3.3. 組み立て

LED/SW ボードに SUZAKU を接続します。(SUZAKU スターターキットの場合は出荷時に接続されています)
接続する際、方向、位置に十分ご注意ください。間違った方向、位置に接続して電源を投入した場合、電源がショートして壊れる可能性があります。

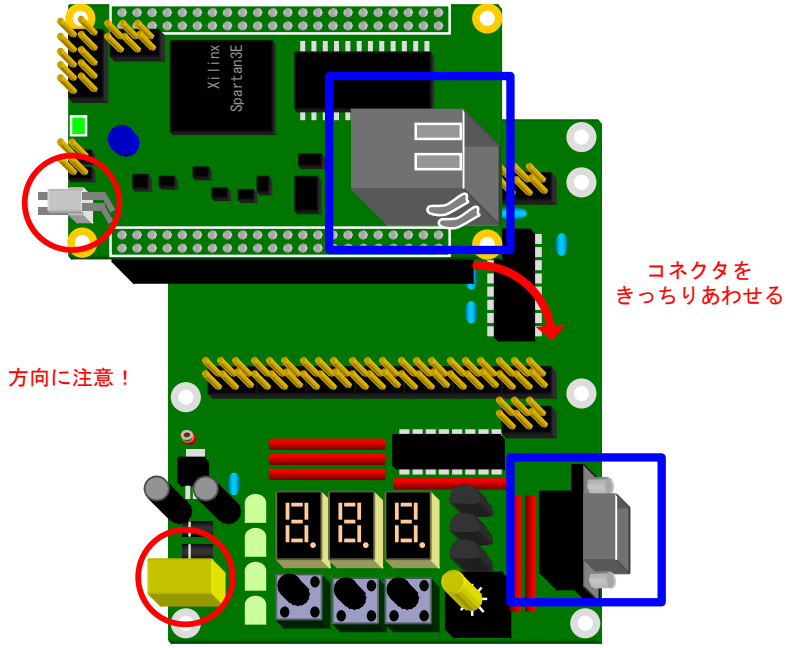


図 3-2　組み立て工程 1

もし、CON2、CON3 にコネクタが接続されていない場合、取り付け面と位置に注意し、コネクタを半田付けしてください。

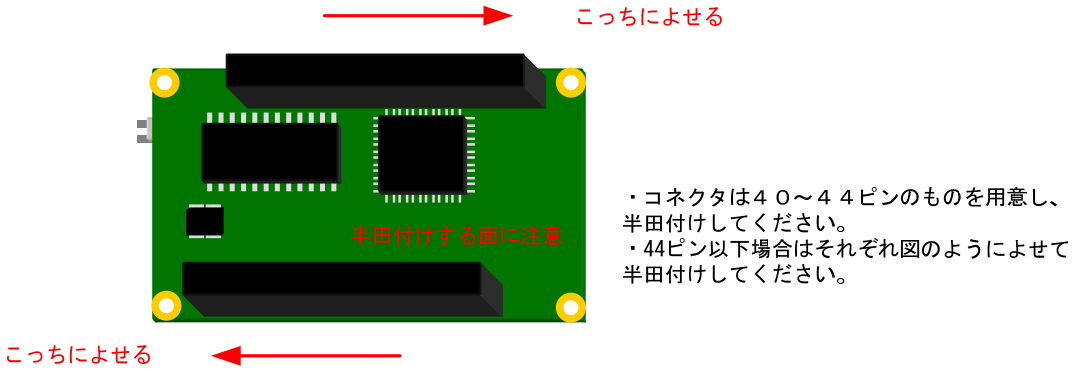


図 3-3　コネクタの半田付け

足を取り付けます。4ヶ所にスペーサを取り付けネジ締めしてください。

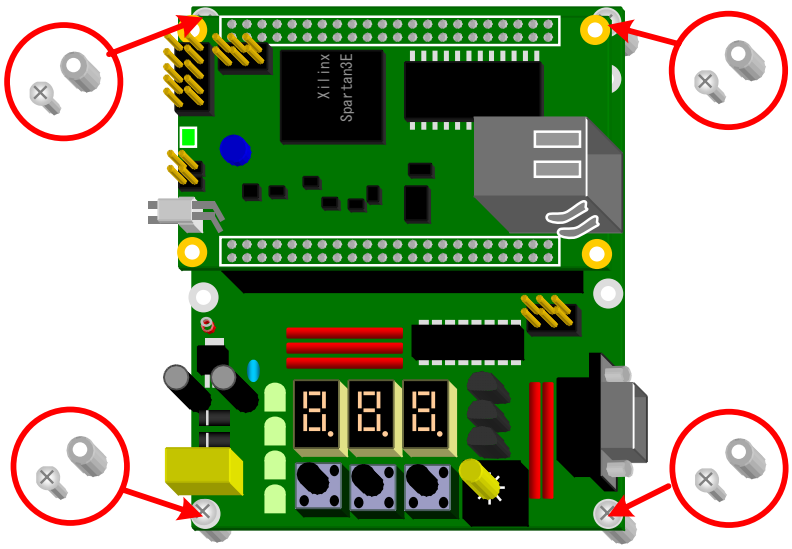


図 3-4　組み立て工程 2

4.SUZAKU＋LED/SW ボードの構成

4.1.各種インターフェースの配置

SUZAKU には FPGA やメモリ、Ethernet コントローラ等が実装され、Linux が動作します。LED/SW ボードには LED やスイッチ、RS-232C レシーバ/ドライバが実装されています。電源は LED/SW ボード側から供給します。(SUZAKU 側の電源コネクタは使用しません)各種インターフェースの配置は下図のようになっています。

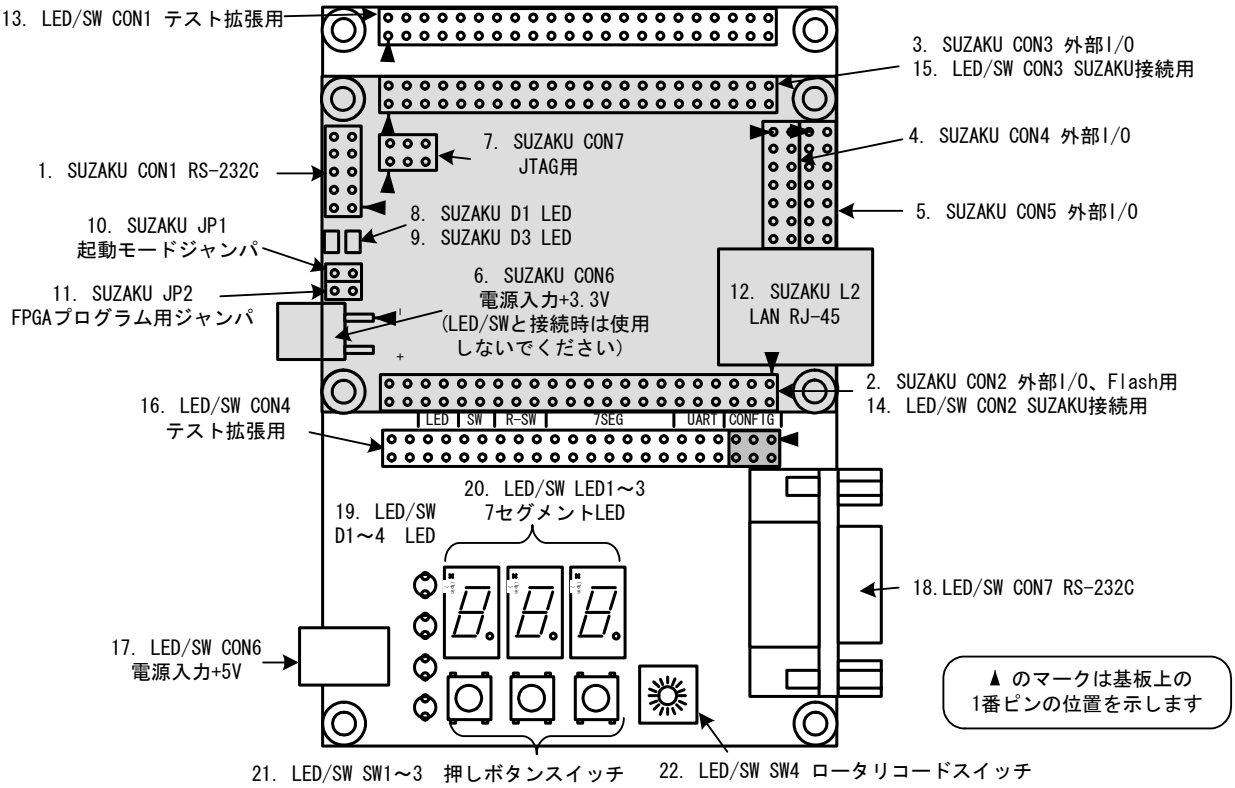


図 4-1 各種インターフェースの配置

表 4-1 SUZAKU のコネクタ配置

		部品番号	説明
SUZAKU	1	CON1	RS-232C コネクタ
	2	CON2	外部 I/O、Flash 用コネクタ(LED/SW CON2 と接続)
	3	CON3	外部 I/O コネクタ(LED/SW CON3 と接続)
	4	CON4	外部 I/O コネクタ
	5	CON5	外部 I/O コネクタ
	6	CON6	電源入力＋3.3V(LED/SW 接続時は絶対に使用しないでください)
	7	CON7	FPGA JTAG 用コネクタ
	8	D1	ユーザーコントロール LED(赤)
	9	D3	パワーON LED(緑)
	10	JP1	起動モードジャンパ
	11	JP2	FPGA プログラム用ジャンパ
	12	L2	Ethernet 10Base-T/100Base-Tx コネクタ

表 4-2　LED/SW のコネクタ配置

		部品番号	説明
LED/SW	13	CON1	テスト拡張用コネクタ（CON3 と同じピンアサインで配線接続されています）
	14	CON2	SUZAKU 接続コネクタ（SUZAKU CON2 と接続）
	15	CON3	SUZAKU 接続コネクタ（SUZAKU CON3 と接続）
	16	CON4	テスト拡張用コネクタ（CON2 と同じピンアサインで配線接続されています）
	17	CON6	＋5V 入力コネクタ
	18	CON7	RS-232C コネクタ
	19	LED1～3	7 セグメント LED　“High”レベルで点灯
	20	D1～4	単色 LED(緑)　“Low”レベルで点灯
	21	SW1～3	押しボタンスイッチ　押下で”Low”レベル
	22	SW4	ロータリコードスイッチ　選択時”Low”レベル

4.2. SUZAKU のインターフェース

4.2.1. SUZAKU CON1 RS-232C

RS-232C コネクタです。レベルバッファを介して、FPGA と接続しています。ボード側で使用しているコネクタは、型式：A1-10PA-2.54DSA、メーカ：ヒロセ（相当品）です。

表 4-3　シリアルコンソールの設定

項目	設定
転送レート	115.2kbps
データ	8bit
パリティ	なし
ストップ bit	1bit
フロー制御	なし

表 4-4　SUZAKU CON1 RS-232C

番号	信号名	I/O	機能	FPGA 接続ピン番号		
				SZ010 SZ030	SZ130	SZ310
1			空き			
2			空き			
3	RXD	I		E2	C12	C10
4	RTS	O		F4	B13	D9
5	TXD	O		E4	A13	C9
6	CTS	I		E1	D12	D10
7			空き			
8			空き			
9	GND		グランド			
10	+3.3VOUT		内部ロジック用電源出力+3.3V			

4.2.2. SUZAKU CON2 外部 I/O、Flash 用コネクタ

外部 I/O 及び Flash 用コネクタです。LED/SW ボードの CON2 とコネクタ接続しています。

表 4-5 SUZAKU CON2 外部 I/O、Flash 用コネクタ

番号	I/O	機能	FPGA 接続ピン番号		
			SZ010 SZ030	SZ130	SZ310
1		グランド			
2	O	内部ロジック用電源出力+3.3V			
3	I	FPGA プログラム用	TCK	CLK	TCK
4	I	FPGA プログラム用	TDI	D	TDI
5	O	FPGA プログラム用	TDO	DO	TDO
6	I	FPGA プログラム用	TMS	nCS	TMS
7	I/O	外部 I/O	A5	N5	E14
8	I/O	外部 I/O	A7	N4	E15
9	I/O	外部 I/O	A3	M6	E13
10	I/O	外部 I/O	D5	M5	F12
11	I/O	外部 I/O	B4	M3	F13
12	I/O	外部 I/O	A4	M4	F14
13	I/O	外部 I/O	C5	L5	F15
14	I/O	外部 I/O	B5	L6	F16
15	I/O	外部 I/O	E6	L4	G13
16	I/O	外部 I/O	D6	L3	G14
17	I/O	外部 I/O	C6	L2	G15
18	I/O	外部 I/O	B6	L1	G16
19		グランド or 誤挿入防止用			
20	I/O	外部 I/O	A8	C9	N9
21		グランド			
22	I/O	外部 I/O	B8	D9	P9
23	I/O	外部 I/O	E7	K5	G12
24	I/O	外部 I/O	D7	K6	H13
25	I/O	外部 I/O	C7	K4	H14
26	I/O	外部 I/O	B7	K3	H15
27	I/O	外部 I/O	D8	J2	H16
28	I/O	外部 I/O	C8	J1	J16
29	I/O	外部 I/O	A9	F9	J15
30	I/O	外部 I/O	A12	E9	J14
31	I/O	外部 I/O	C10	A10	J13
32	I/O	外部 I/O	D12	B10	K12
33	I/O	外部 I/O	A14	D11	K16
34	I/O	外部 I/O	B14	C11	K15
35	I/O	外部 I/O	A13	F11	K14
36	I/O	外部 I/O	B13	E11	K13
37	I/O	外部 I/O	B12	E12	L16
38	I/O	外部 I/O	C12	F12	L15
39	I/O	外部 I/O	D11	B11	L14
40	I/O	外部 I/O	E11	A11	L13
41		グランド			
42		グランド			
43	I	電源入力+3.3V			
44	I	電源入力+3.3V			

4.2.3. SUZAKU CON3 外部 I/O コネクタ

外部 I/O コネクタです。LED/SW ボードの CON3 とコネクタ接続しています。

表 4-6 SUZAKU CON3 外部 I/O コネクタ

番号	I/O	機能	FPGA 接続ピン番号		
			SZ010 SZ030	SZ130	SZ310
1	I	電源入力+3.3V			
2	I	電源入力+3.3V			
3		グランド			
4		グランド			
5	I/O	外部 I/O	B11	B14	L12
6	I/O	外部 I/O	C11	A14	M13
7	I/O	外部 I/O	D10	D14	M16
8	I/O	外部 I/O	E10	C14	N16
9	I/O	外部 I/O	A10	B16	M15
10	I/O	外部 I/O	B10	A16	M14
11	I/O	外部 I/O	B16	C18	P15
12	I/O	外部 I/O	C16	C17	P13
13	I/O	外部 I/O	C15	D17	R14
14	I/O	外部 I/O	D14	D16	P14
15	I/O	外部 I/O	D15	F15	T15
16	I/O	外部 I/O	D16	F14	T14
17	I/O	外部 I/O	E13	G14	N12
18	I/O	外部 I/O	E14	G13	P12
19	I/O	外部 I/O	E15	F18	N11
20	I/O	外部 I/O	E16	F17	M11
21	I/O	外部 I/O	F12	G15	M10
22	I/O	外部 I/O	F13	G16	N10
23	I/O	外部 I/O	C9	E10	R9
24		グランド			
25	I/O	外部 I/O	D9	D10	T9
26		グランド			
27	I/O	外部 I/O	F14	H14	P10
28	I/O	外部 I/O	F15	H15	T8
29	I/O	外部 I/O	G12	H16	R8
30	I/O	外部 I/O	G13	H17	P8
31	I/O	外部 I/O	G14	J12	N8
32	I/O	外部 I/O	G15	J13	P7
33	I/O	外部 I/O	H13	J15	N7
34	I/O	外部 I/O	H14	J14	M7
35	I/O	外部 I/O	H15	J17	M6
36	I/O	外部 I/O	H16	J16	N6
37	I/O	外部 I/O	P16	K15	P5
38	I/O	外部 I/O	R16	K14	N5
39	I/O	外部 I/O	K15	K13	T3
40	I/O	外部 I/O	G16	K12	T2
41					
42		未接続			
43	O	内部ロジック用電源出力+3.3V			
44		グランド			

4.2.4. SUZAKU CON4 外部 I/O コネクタ

外部 I/O コネクタです。コネクタは実装されていません。

表 4-7 SUZAKU CON4 外部 I/O コネクタ

番号	I/O	機能	FPGA 接続ピン番号		
			SZ010 SZ030	SZ130	SZ310
1		空き			
2		空き			
3	I/O	外部 I/O	L15	L18	N8
4	I/O	外部 I/O	L14	L17	P7
5	I/O	外部 I/O	K12	L16	N7
6	I/O	外部 I/O	L12	L15	M7
7	I/O	外部 I/O	K14	N18	M6
8	I/O	外部 I/O	K13	M18	N6
9	I/O	外部 I/O	J14	M16	P5
10	I/O	外部 I/O	J13	M15	N5
11	I/O	外部 I/O	J16	P17	T3
12	I/O	外部 I/O	K16	P18	T2

4.2.5. SUZAKU CON5 外部 I/O コネクタ

外部 I/O コネクタです。コネクタは実装されていません。

表 4-8 SUZAKU CON5 外部 I/O コネクタ

番号	I/O	機能	FPGA 接続ピン番号		
			SZ010 SZ030	SZ130	SZ310
1		グランド			
2	O	内部ロジック用電源出力 +3.3V			
3	I/O	外部 I/O	P15	M14	P4
4	I/O	外部 I/O	P14	M13	R3
5	I/O	外部 I/O	N16	R15	P3
6	I/O	外部 I/O	N15	R16	P2
7	I/O	外部 I/O	M14	R18	M10
8	I/O	外部 I/O	N14	T18	N10
9	I/O	外部 I/O	M16	U18	P10
10	I/O	外部 I/O	M15	T17	T8
11	I/O	外部 I/O	L13	T15	R8
12	I/O	外部 I/O	M13	R14	P8

4.2.6. SUZAKU CON6 電源入力+3.3V

SUZAKU と LED/SW ボードを接続時は使用しないでください。詳細は"7.3.1.電源について"を参照してください。

4.2.7. SUZAKU CON7 FPGA JTAG 用コネクタ

FPGA JTAG 用コネクタです。JTAG の I/O の電圧は+2.5V です。+2.5V に対応した JTAG ダウンロードケーブルを使用してください。

表 4-9 SUZAKU CON7 FPGA JTAG 用コネクタ

番号	信号名	I/O	機能
1	GND		グランド
2	+2.5VOUT		内部ロジック用電源出力 +2.5V
3	TCK	I	JTAG
4	TDI	I	JTAG
5	TDO	O	JTAG
6	TMS	I	JTAG

4.2.8. SUZAKU D1,D3 LED

ユーザーコントロール LED(赤)とパワーON LED(緑)です。

表 4-10 SUZAKU D1、D3 LED

信号名	I/O	機能	FPGA 接続ピン番号		
			SZ010 SZ030	SZ130	SZ310
D1	O	ユーザーコントロール LED	G5	T3	A9
D3	O	SUZAKU ボードに＋3.3V が供給されると点灯			

4.2.9. SUZAKU JP1,JP2 設定用ジャンパ

起動モード設定用ジャンパと FPGA プログラム用ジャンパです。

表 4-11 SUZAKU JP1、JP2 設定用ジャンパ

信号名	I/O	機能
JP1	I	起動モード設定用ジャンパです。オープンでオートブート(SUZAKU 起動時に Linux が自動的に起動)します。ショートでブートローダモード(ブートローダのみを起動した状態)になります。
JP2		FPGA に JTAG からコンフィギュレーションする時と、SPI Flash にコンフィギュレーションデータをダウンロードする時に使用するジャンパです。(本ジャンパをショートすると、電源再投入時 FPGA に対し、コンフィギュレーションを停止することができます)

4.2.10. SUZAKU L2 Ethernet 10Base-T/100Base-Tx

ボード側で使用しているコネクタ型式/メーカは、J0026D21B/PULSE です。

表 4-12 SUZAKU L2 Ethernet 10Base-T/100Base-Tx

番号	信号名	I/O	機能
1	TX+		差動ツイストペア出力+
2	TX-		差動ツイストペア出力-
3	RX+		差動ツイストペア入力+
4			75Ω終端(4 番ピンと 5 番ピンはショートしています)
5			75Ω終端(4 番ピンと 5 番ピンはショートしています)
6	RX-		差動ツイストペア入力-
7			75Ω終端(7 番ピンと 8 番ピンはショートしています)
8			75Ω終端(7 番ピンと 8 番ピンはショートしています)

4.3. LED/SW ボードのインターフェース

4.3.1. LED/SW CON1 テスト拡張用コネクタ

CON3 と同じピンアサインで信号が配線接続されています。詳しくは CON3 を参照してください。

4.3.2. LED/SW CON2 SUZAKU 接続コネクタ

SUZAKU CON2 と接続しています。

表 4-13 LED/SW CON2 SUZAKU 接続コネクタ

番号	信号名	I/O	機能	FPGA 接続ピン番号		
				SZ010 SZ030	SZ130	SZ310
1	GND		グランド			
2	+3.3V	I	+3.3V SUZAKU 側から供給			
3	CONF_C			TCK	CLK	TCK
4	CONF_I			TDI	D	TDI
5	CONF_O			TDO	DO	TDO
6	CONF_S			TMS	nCS	TMS
7	NC			A5	N5	E14
8	UART3	I	RTS	A7	N4	E15
9	UART2	O	TXD	A3	M6	E13
10	UART1	O	CTS	D5	M5	F12
11	UART0	I	RXD	B4	M3	F13
12	NC			A4	M4	F14
13	SEG7	O	セグメント DP "High"で点灯	C5	L5	F15
14	SEG6	O	セグメント G "High"で点灯	B5	L6	F16
15	SEG5	O	セグメント F "High"で点灯	E6	L4	G13
16	SEG4	O	セグメント E "High"で点灯	D6	L3	G14
17	SEG3	O	セグメント D "High"で点灯	C6	L2	G15
18	SEG2	O	セグメント C "High"で点灯	B6	L1	G16
19			誤挿入防止用			
20	SEG1	O	セグメント B "High"で点灯	A8	C9	N9
21	GND		グランド			
22	SEG0	O	セグメント A "High"で点灯	B8	D9	P9
23	NC			E7	K5	G12
24	nSEL2	O	7 セグメント LED3 "Low"でコモン選択	D7	K6	H13
25	nSEL1	O	7 セグメント LED2 "Low"でコモン選択	C7	K4	H14
26	nSEL0	O	7 セグメント LED1 "Low"でコモン選択	B7	K3	H15
27	NC			D8	J2	H16
28	nCODE3	I	ロータリスイッチ 4 ビット目 選択時"Low"	C8	J1	J16
29	nCODE2	I	ロータリスイッチ 3 ビット目 選択時"Low"	A9	F9	J15
30	nCODE1	I	ロータリスイッチ 2 ビット目 選択時"Low"	A12	E9	J14
31	nCODE0	I	ロータリスイッチ 1 ビット目 選択時"Low"	C10	A10	J13
32	NC			D12	B10	K12
33	nSW2	I	押しボタンスイッチ SW3 押下で"Low"	A14	D11	K16
34	nSW1	I	押しボタンスイッチ SW2 押下で"Low"	B14	C11	K15
35	nSW0	I	押しボタンスイッチ SW1 押下で"Low"	A13	F11	K14
36	NC			B13	E11	K13
37	nLE0	O	単色 LED(緑) D1 "Low"で点灯	B12	E12	L16
38	nLE1	O	単色 LED(緑) D2 "Low"で点灯	C12	F12	L15
39	nLE2	O	単色 LED(緑) D3 "Low"で点灯	D11	B11	L14
40	nLE3	O	単色 LED(緑) D4 "Low"で点灯	E11	A11	L13
41	GND		グランド			
42	GND		グランド			
43	+3.3V	O	電源出力 +3.3V SUZAKU 側に供給			
44	+3.3V	O	電源出力 +3.3V SUZAKU 側に供給			

4.3.3. LED/SW CON3 SUZAKU 接続コネクタ

SUZAKU CON3 と接続しています。

表 4-14 LED/SW CON3 SUZAKU 接続コネクタ

番号	信号名	I/O	機能
1	+3.3V	O	+3.3V SUZAKU 側に供給
2	+3.3V	O	+3.3V SUZAKU 側に供給
3	GND		グランド
4	GND		グランド
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24	GND		グランド
25			
26	GND		グランド
27			
28			
29			
30			
31			
32			
33			
34			
35			
36			
37			
38			
39			
40			
41			
42			
43	+3.3V	I	+3.3V SUZAKU 側から供給
44	GND		グランド

4.3.4. LED/SW CON4 テスト拡張用コネクタ

CON2 と同じピンアサインで信号が配線接続されています。
詳細は CON2 を参照してください。

4.3.5. LED/SW CON6 ＋5V 入力コネクタ

＋5V±5%の電源を入力してください。AC アダプタ 5V は添付品を使用してください。(＋5V 出力 EIAJ＃2)

表 4-15 LED/SW CON6 ＋5V 入力コネクタ

番号	信号名	I/O	機能
1	＋5V	I	＋5V センター＋ピン
2	GND		グランド

図 4-2 ＋5V センター＋ピン

4.3.6. LED/SW CON7 RS-232C コネクタ

D-sub9 ピンが実装されています。

表 4-16 LED/SW CON7 RS-232C コネクタ

番号	信号名	I/O	機能	FPGA 接続先		
				SZ010 SZ030	SZ130	SZ310
1						
2	UART0	I	RXD	B4	M3	F13
3	UART2	O	TXD	A3	M6	E13
4						
5	GND		グランド			
6						
7	UART3	O	RTS	A7	N4	E14
8	UART1	I	CTS	D5	M5	F12
9						

4.3.7. LED/SW 7 セグメント LED セレクタ

7 セグメント LED 選択用 PNP トランジスタが実装されています。"Low"を入力すると、それぞれに対応する 7 セグメント LED を選択することができます。

表 4-17 LED/SW 7セグメント LED セレクタ

番号	信号名	I/O	機能	FPGA 接続先		
				SZ010 SZ030	SZ130	SZ310
LED1	nSEL0	O	LED1 コモン "Low"で選択	B7	K3	H15
LED2	nSEL1	O	LED2 コモン "Low"で選択	C7	K4	H14
LED3	nSEL2	O	LED3 コモン "Low"で選択	D7	K6	H13

4.3.8. LED/SW LED1～3　7 セグメント LED

7 セグメント LED が 3 つ実装されています。”High”を入力すると、それぞれに対応するセグメントを点灯させることができます。

表 4-18　LED/SW LED1~3　7 セグメント LED

番号	信号名	I/O	機能	FPGA 接続先		
				SZ010 SZ030	SZ130	SZ310
A	SEG0	O	セグメント A "High"で点灯	B8	D9	P9
B	SEG1	O	セグメント B " High"で点灯	A8	C9	N9
C	SEG2	O	セグメント C " High"で点灯	B6	L1	G16
D	SEG3	O	セグメント D " High"で点灯	C6	L2	G15
E	SEG4	O	セグメント E " High "で点灯	D6	L3	G14
F	SEG5	O	セグメント F "High"で点灯	E6	L4	G13
G	SEG6	O	セグメント G "High"で点灯	B5	L6	F16
DP	SEG7	O	セグメント DP "High"で点灯	C5	L5	F15

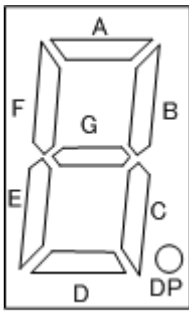


図 4-3　7 セグメント LED

4.3.9. LED/SW D1～4 単色 LED(緑)

単色 LED（緑）が4つ実装されています。”Low”を入力すると点灯します。

表 4-19　LED/SW D1～4 単色 LED(緑)

番号	信号名	I/O	機能	FPGA 接続先		
				SZ010 SZ030	SZ130	SZ310
D1	nLE0	O	単色 LED(緑) D1　"Low"で点灯	B12	E12	L16
D2	nLE1	O	単色 LED(緑) D2　"Low"で点灯	C12	F12	L15
D3	nLE2	O	単色 LED(緑) D3　"Low"で点灯	D11	B11	L14
D4	nLE3	O	単色 LED(緑) D4　"Low"で点灯	E11	A11	L13

4.3.10. LED/SW SW1～3 押しボタンスイッチ

押しボタンスイッチが 3 つ実装されています。押すと"Low"を出力します。

表 4-20 LED/SW SW1～3

番号	信号名	I/O	機能	FPGA 接続先		
				SZ010 SZ030	SZ130	SZ310
SW1	nSW0	I	押しボタンスイッチ SW1 押下で"Low"	A13	F11	K14
SW2	nSW1	I	押しボタンスイッチ SW2 押下で"Low"	B14	C11	K15
SW3	nSW2	I	押しボタンスイッチ SW3 押下で"Low"	A14	D11	K16

4.3.11. LED/SW SW4 ロータリコードスイッチ

ロータリコードスイッチが実装されています。選択時"Low"を出力します。

表 4-21 LED/SW SW4

番号	信号名	I/O	機能	FPGA 接続先		
				SZ010 SZ030	SZ130	SZ310
SW4	nCODE0	I	ロータリコードスイッチ 2^0 選択で"Low"	C10	A10	J13
	nCODE1	I	ロータリコードスイッチ 2^1 選択で"Low"	A12	E9	J14
	nCODE2	I	ロータリコードスイッチ 2^2 選択で"Low"	A9	F9	J15
	nCODE3	I	ロータリコードスイッチ 2^3 選択で"Low"	C8	J1	J16

5.SUZAKU について

5.1.SUZAKU の特徴

SUZAKU は FPGA をベースとしたボードコンピュータです。
FPGA 上にプロセッサと周辺ペリフェラルコアを構成し、オペレーティングシステムとして Linux を採用しています。

● **FPGA**
Xilinx の FPGA を採用し、大規模で柔軟な拡張をすることが出来ます。SZ010、SZ030 は Spartan-3、SZ130 は Spartan-3E、SZ310 は Virtex-Ⅱ Pro を搭載しています。

● **プロセッサ**
SZ010、SZ030、SZ130 はソフトプロセッサの MicroBlaze を採用し、SZ310 はハードプロセッサの PowerPC405 を採用しています。

● **カスタマイズ**
FPGA の内部はユーザによってカスタマイズが可能です。また、基板外周に SZ010、SZ030、SZ130 は 86 ピン、SZ310 は 70 ピンのユーザが自由に使える外部 I/O を実装しています。例えば、GPIO や UART の数を増やし、外部 I/O ピンに割り当てるなどのカスタマイズが簡単に行えます。

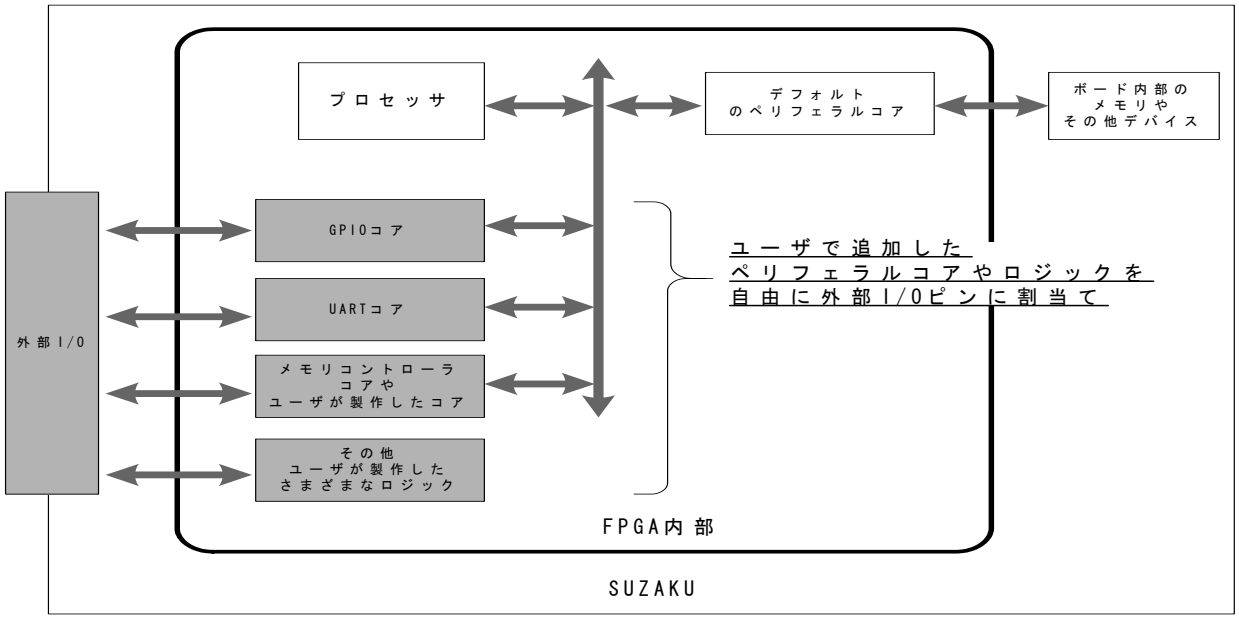


図 5-1 SUZAKU とは

● **Linux**
Linux を標準のオペレーティングシステムとして採用しているので、アプリケーションソフトウェアの開発には GNU のアセンブラや C コンパイラ等を使用することができます。SZ010、SZ030、SZ130 は MMU 不要の μCLinux、SZ310 は標準的な Linux に対応しています。

● **ネットワーク**
ボードには LAN(10Base-T/100Base-Tx)が実装されています。LAN コントローラデバイスドライバ、各種プロトコルが最初から用意されているので、簡単にネットワークに接続できます。

5.2. 仕様

SUZAKU の主な仕様を以下に示します。

表 5-1 SUZAKU の仕様

	SUZAKU-S			SUZAKU-V
モデル	SZ010	SZ030	SZ130	SZ310
FPGA デバイス	Xilinx Spartan-3 (XC3S400)	Xilinx Spartan-3 (XC3S1000)	Xilinx Spartan-3E (XC3S1200E)	Xilinx Virtex-II Pro (XC2VP4)
CPU コア	MicroBlaze			PowerPC405
CPU クロック	51.6096MHz			265.4208MHz
水晶発振器周波数	3.6864MHz (FPGA 内部の DCM により逓倍して使用)			
DRAM	16MB		16MB × 2	32MB
フラッシュメモリ	4MB	8MB	8MB (SPI)	8MB
Ethernet	10BASE-T/100BASE-TX			
拡張 I/O ピン	86			70
シリアル	1ch(UART : 115.2kbps)			
タイマ	2ch(OPB Timer : 1ch は OS で使用)			PowerPC 内蔵タイマ
コンフィギュレーション	TE7720		SPI Flash	TE7720
基板サイズ	72×47mm			
電源	電圧:+3.3V±3%			
リセット機能	ソフトウェアリセット			
標準 OS	μCLinux			Linux

5.3.メモリマップ

5.3.1. SZ010、SZ030

SZ010 および SZ030 のメモリマップは以下のとおりです。

表 5-2　SZ010、SZ030 のメモリマップ

Start Address	End Address	ペリフェラル	デバイス
0x0000 0000	0x0000 1FFF	BRAM	
0x0000 1000	0x7FFF FFFF	Reserved	
0x8000 0000	0x80FF FFFF	OPB-SDRAM Controller	SDRAM 16Mbyte
0x8100 0000	0xFEFF FFFF	Free	
0xFF00 0000	0xFF7F FFFF	OPB-EMC	Flash 4Mbyte or 8Mbyte
0xFF80 0000	0xFFCF FFFF	Free	
0xFFE0 0000	0xFFEF FFFF	OPB-EMC	LAN コントローラ
0xFFF0 0000	0xFFFF 0FFF	Free	
0xFFFF 1000	0xFFFF 10FF	OPB-Timer	
0xFFFF 1100	0xFFFF 1FFF	Free	
0xFFFF 2000	0xFFFF 20FF	OPB-UART Lite	RS-232C
0xFFFF 2100	0xFFFF 2FFF	Free	
0xFFFF 3000	0xFFFF 30FF	OPB-Interrupt Controller	
0xFFFF 3100	0xFFFF 9FFF	Free	
0xFFFF A000	0xFFFF A1FF	OPB-GPIO	ブートモードジャンパ ソフトウェアリセット
0xFFFF A200	0xFFFF A3FF	OPB-LED	
0xFFFF A400	0xFFFF FFFF	Free	

5.3.2. SZ130

SZ130 のメモリマップは以下のとおりです。

表 5-3　SZ130 のメモリマップ

Start Address	End Address	ペリフェラル	デバイス
0x0000 0000	0x0000 1FFF	BRAM	
0x0000 1000	0x7FFF FFFF	Reserved	
0x8000 0000	0x81FF FFFF	OPB-SDRAM Controller	SDRAM 32Mbyte
0x8200 0000	0xFEFF FFFF	Free	
0xFF00 0000	0xFF7F FFFF	OPB-EMC	SPI Flash 8Mbyte
0xFF80 0000	0xFFCF FFFF	Free	
0xFFE0 0000	0xFFEF FFFF	OPB-EMC	LAN コントローラ
0xFFF0 0000	0xFFFF 0FFF	Free	
0xFFFF 1000	0xFFFF 10FF	OPB-Timer	
0xFFFF 1100	0xFFFF 1FFF	Free	
0xFFFF 2000	0xFFFF 20FF	OPB-UART Lite	RS-232C
0xFFFF 2100	0xFFFF 2FFF	Free	
0xFFFF 3000	0xFFFF 30FF	OPB-Interrupt Controller	
0xFFFF 3100	0xFFFF 9FFF	Free	
0xFFFF A000	0xFFFF A1FF	OPB-GPIO	ブートモードジャンパ ソフトウェアリセット
0xFFFF A200	0xFFFF A3FF	OPB-LED	
0xFFFF A400	0xFFFF FFFF	Free	

5.3.3. SZ310

SZ310 のメモリマップは以下のとおりです。

表 5-4　SZ310 のメモリマップ

Start Address	End Address	ペリフェラル	デバイス
0x0000 0000	0x01FF FFFF	PLB-SDRAM Controller	SDRAM 32Mbyte
0x0200 0000	0xEFFF FFFF	Free	
0xF000 0000	0xF07F FFFF	PLB-EMC	Flash 8Mbyte
0xF080 0000	0xF0CF FFFF	Free	
0xF0E0 0000	0xF0EF FFFF	PLB-EMC	LAN コントローラ
0xF0F0 0000	0xF0FF 1FFF	Free	
0xF0FF 2000	0xF0FF 20FF	OPB-UART Lite	RS-232C
0xF0FF 2100	0xF0FF 2FFF	Free	
0xF0FF 3000	0xF0FF 30FF	OPB-Interrupt Controller	
0xF0FF 3100	0xF0FF 9FFF	Free	
0xF0FF A000	0xF0FF A1FF	OPB-GPIO	ブートモードジャンパ ソフトウェアリセット
0xF0FF A200	0xF0FF A3FF	OPB-LED	
0xF0FF A400	0xFFFF BFFF	Free	
0xFFFF C000	0xFFFF FFFF	BRAM	

6.LED/SW ボードについて

6.1.回路説明

ここでは LED/SW ボードの回路図(LED_SW_Schematic.pdf)の概要を説明します。本回路図及び部品表は付属 CD-ROM の"¥suzaku-starter-kit¥fpga¥doc"に収録されているので詳細はそちらを参照してください。

LED/SW には単色 LED が 4 つ(D1、D2、D3、D4)、押しボタンスイッチが 3 つ(SW1、SW2、SW3)、ロータリコードスイッチが 1 つ(SW4)、7 セグメント LED が 3 つ(LED1、LED2、LED3)、シリアルポートが 1 つ実装されており、それぞれ CON2 から SUZAKU と接続するようになっています。安定した+3.3V を得るため AC アダプタ 5V から 3 端子レギュレータで+3.3V を作っています。この+3.3V は CON2、CON3から SUZAKU 側へ供給されます。

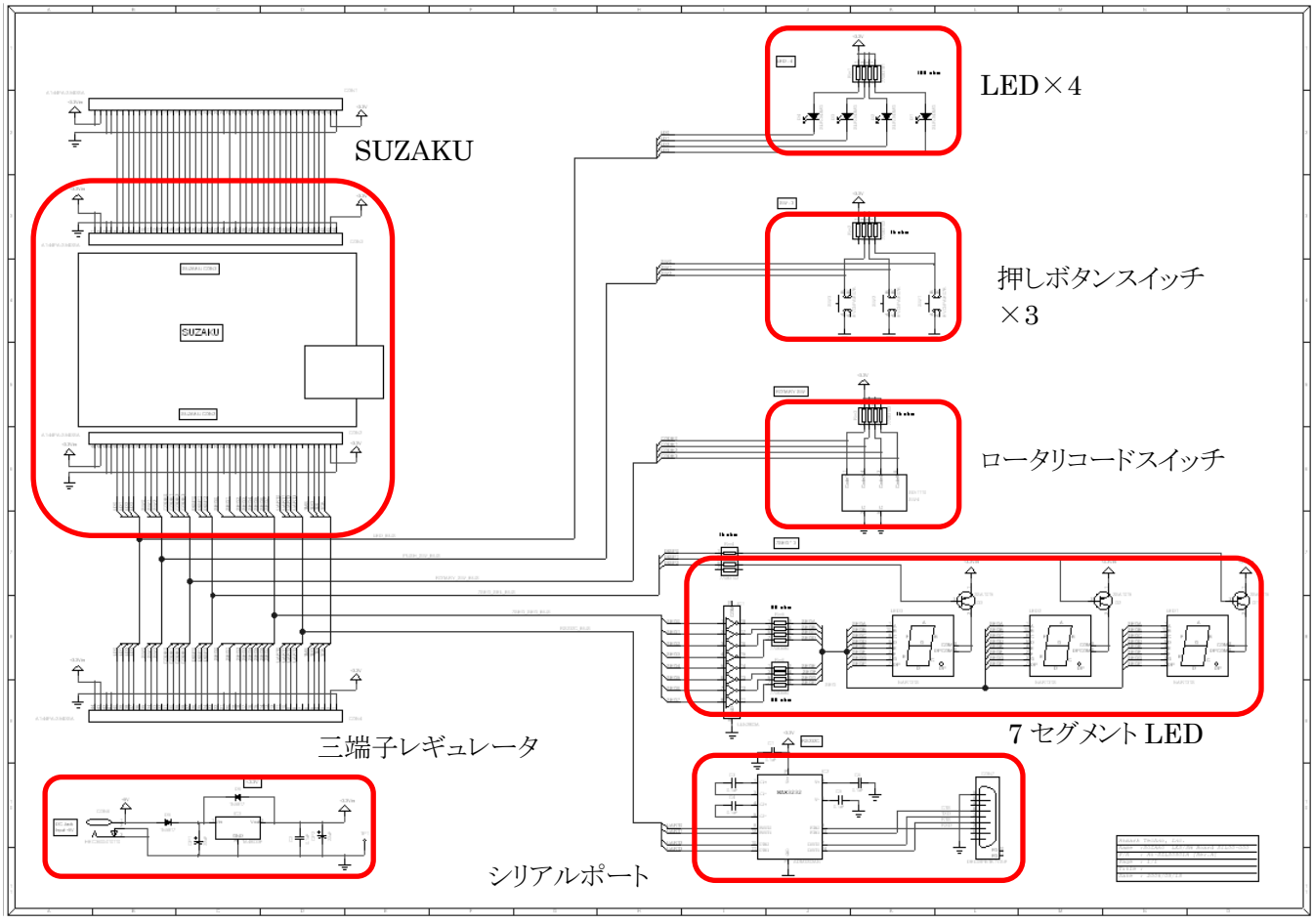


図 6-1 LED/SW 回路図(縮小版)

6.2. ピンアサイン

LED/SW ボードを使用する際に必要となるピンアサインを以下に示します。

表 6-1　クロック、リセット信号 ピンアサイン

番号	信号名	I/O	機能	FPGA 接続先		
				SZ010 SZ030	SZ130	SZ310
	SYS_CLK	I	クロック信号	T9	U10	C8
	SYS_RST	I	リセット信号	F5	D3	A8

表 6-2　機能用ピンアサイン

番号	信号名	I/O	機能	FPGA 接続先		
				SZ010 SZ030	SZ130	SZ310
8	UART3	I	RTS	A7	N4	E14
9	UART2	O	TXD	A3	M6	E13
10	UART1	O	CTS	D5	M5	F12
11	UART0	I	RXD	B4	M3	F13
13	SEG7	O	セグメント DP	C5	L5	F15
14	SEG6	O	セグメント G	B5	L6	F16
15	SEG5	O	セグメント F	E6	L4	G13
16	SEG4	O	セグメント E	D6	L3	G14
17	SEG3	O	セグメント D	C6	L2	G15
18	SEG2	O	セグメント C	B6	L1	G16
20	SEG1	O	セグメント B	A8	C9	N9
22	SEG0	O	セグメント A	B8	D9	P9
24	nSEL2	O	7 セグメント LED3 選択	D7	K6	H13
25	nSEL1	O	7 セグメント LED2 選択	C7	K4	H14
26	nSEL0	O	7 セグメント LED1選択	B7	K3	H15
28	nCODE3	I	ロータリコードスイッチ 2^3	C8	J1	J16
29	nCODE2	I	ロータリコードスイッチ 2^2	A9	F9	J15
30	nCODE1	I	ロータリコードスイッチ 2^1	A12	E9	J14
31	nCODE0	I	ロータリコードスイッチ 2^0	C10	A10	J13
33	nSW2	I	押しボタンスイッチ SW3	A14	D11	K16
34	nSW1	I	押しボタンスイッチ SW2	B14	C11	K15
35	nSW0	I	押しボタンスイッチ SW1	A13	F11	K14
37	nLE0	O	単色 LED(緑) D1	B12	E12	L16
38	nLE1	O	単色 LED(緑) D2	C12	F12	L15
39	nLE2	O	単色 LED(緑) D3	D11	B11	L14
40	nLE3	O	単色 LED(緑) D4	E11	A11	L13

6.2.1. CoreConnect

MicroBlaze、PowerPC405 はバスアーキテクチャとして IBM の CoreConnect を採用しています。CoreConnect のバスおよびレジスタビットの命名規則で MSB 側が 0 ビット目に定義されています。このため EDK より自動生成するバスはすべて MSB 側が0ビット目で定義（例：std_logic_vector(0 to 7)）されます。しかし、本スターターキットの LED/SW ボードを含め、これにつなぐほとんどの外部デバイスが、LSB 側が 0 ビット目で定義されており、このまま接続しても動作しません。

本書内の VHDL ソースの信号は、IBM の CoreConnect にあわせてバスを MSB 側が 0 ビット目で定義していますが、外部デバイスと接続するために、FPGA のピンアサイン設定ですべて信号をひっくり返しています。

7.SUZAKU＋LED/SW ボードを動かす

まず出荷状態の SUZAKU スターターキット(SZ130-U00＋LED/SW ボード)を動かします。出荷状態では Flash に Linux が OS として入り、FPGA に今回最終目標とするスロットマシンが入っています。SUZAKU スターターキット以外の場合は Flash の image を書き直して下さい。image は付属 CD-ROM の"¥suzaku-starter-kit¥image"の中の image-sz***-sil.bin を使ってください。書き換える方法については、「SUZAKU Software Manual」を参照してください。

SUZAKU にはオートブートモード(JP1:オープン)とブートローダモード(JP1:ショート)があり、オートブートモードでは Linux が自動的に起動し、ブートローダモードではブートローダのみが起動します。起動モード設定用ジャンパ JP1 によりこの 2 つを切り替え、動作を確認します。

7.1.接続方法

D-Sub9 ピン-10 ピン変換ケーブル、LAN ケーブルを適切なコネクタに接続してください。

RS-232C コネクタ(CON1)に D-Sub9 ピン-10 ピン変換ケーブルを接続する時、コネクタの白い三角マークと SUZAKU 基板上の白い三角マークを合わせるように接続します。コネクタの向きを反対に接続すると、機器を破損する恐れがありますので十分にご注意ください。

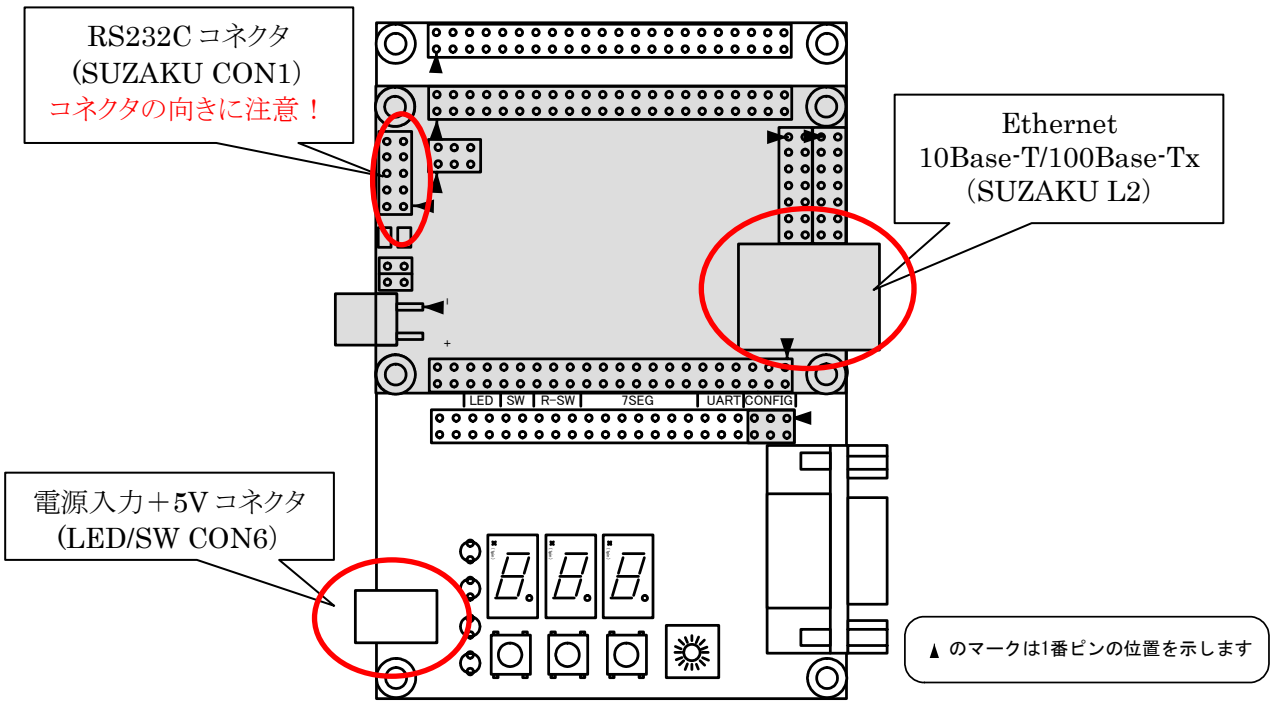


図 7-1　SUZAKU＋LED/SW ボードコネクタ配置図

7.2. シリアル通信ソフトウェア

SUZAKU はシリアルポートをコンソールとして使用します。SUZAKU のコンソールから出力される情報を読み取ったり、SUZAKU のコンソールに情報を送ったりするには、シリアル通信用ソフトウェアが必要です。ここでは Tera Term を使用した例を示します。

シリアル通信用ソフトウェアを立ち上げ、シリアル通信の設定を行ってください。

(“表 4-3 シリアルコンソールの設定” 参照)

・ Baud rate	115200
・ Data	8bit
・ Parity	none
・ Stop	1bit
・ Flow control	none

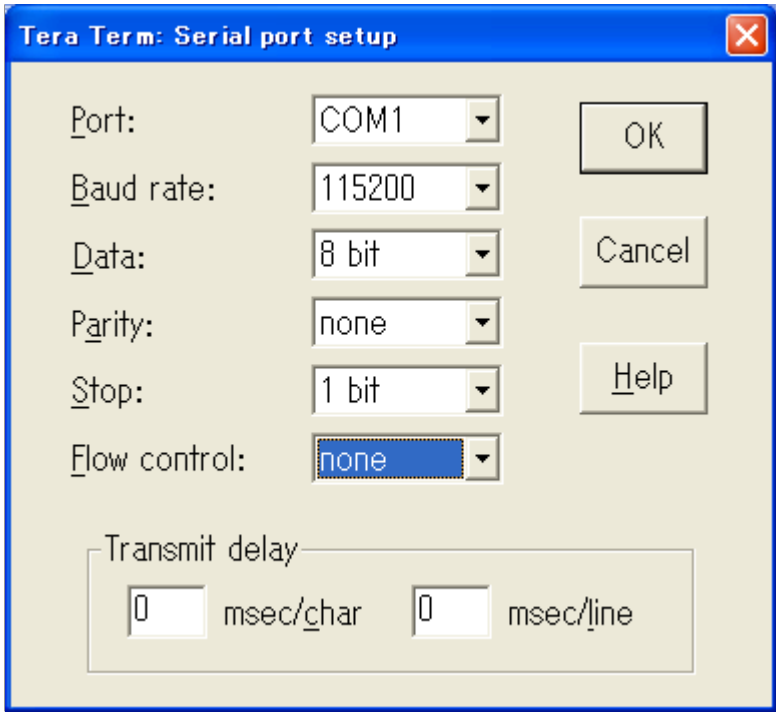


図 7-2 Tera Term の設定

7.3.オートブートモードで Linux を動かす

オートブートモードで Linux を動かします。
JP1、JP2 がオープンになっていることを確認してください。

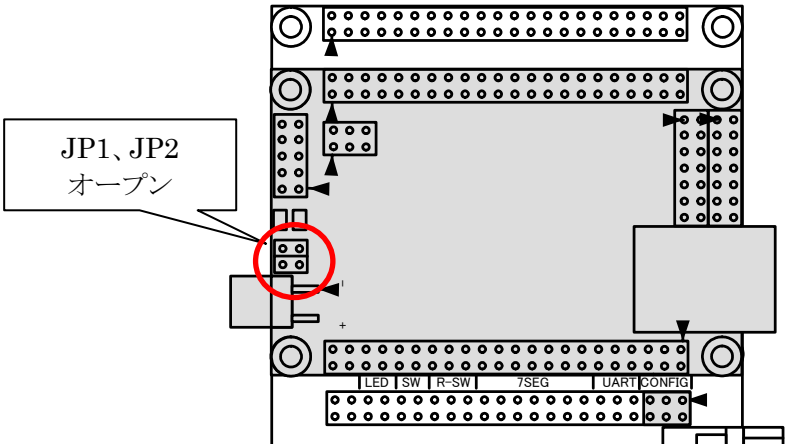


図 7-3　オートブートモード ジャンパの設定

7.3.1. 電源について

LED/SW CON6 から AC アダプタ 5V で電源を供給します。
SUZAKU CON6 からは絶対に電源を供給しないでください。電源がショートし、機器を破損する可能性があります。また、改造等により電源を外部から供給等行わないでください。SUZAKU と LED/SW ボードは、電源シーケンスの関係から、お互いに電源を供給し合うような形になっているので、機器を破損する可能性があります。

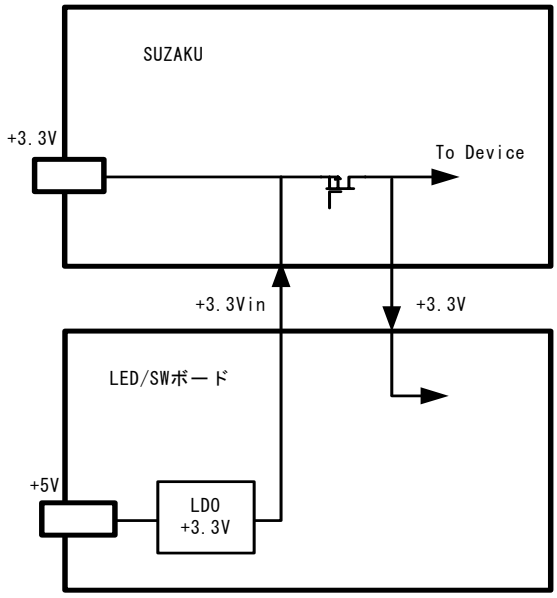


図 7-4　電源系統

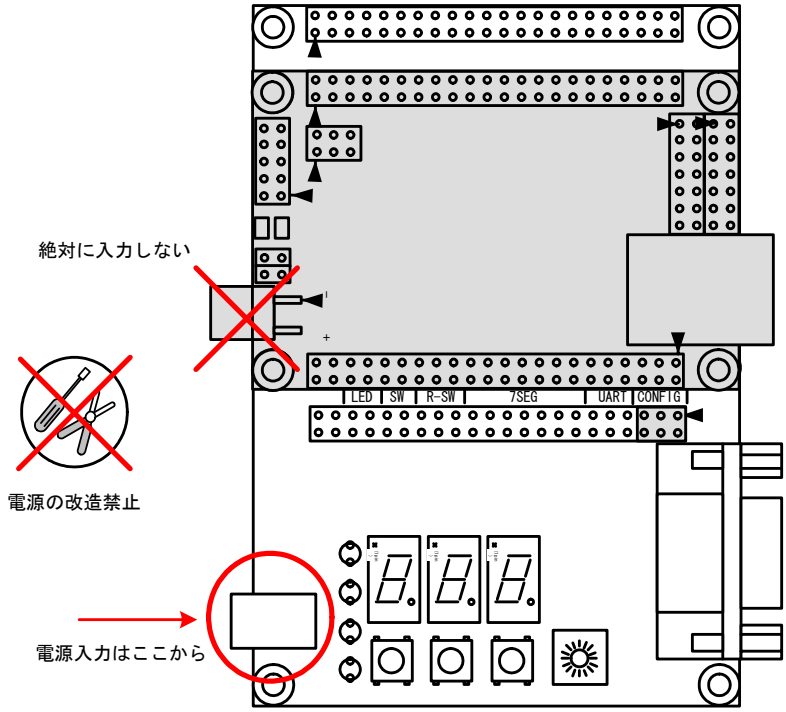


図 7-5　電源ケーブル接続の諸注意

7.3.2. Linux の起動

電源が供給されると、シリアル通信用ソフトウェアの画面に Linux の起動ログが表示されます。

例 7-1　SUZAKU の起動ログ(SZ130 の場合)

```
Linux version 2.4.32-uc0 (atmark@pc-build) (gcc version 3.4.1 ( Xilinx EDK 8.1 Build EDK_I.17 090206 )) #1 2006
年 7月 13日 木曜日 01:19:35 JST
On node 0 totalpages: 8192
zone(0): 8192 pages.
zone(1): 0 pages.
zone(2): 0 pages.
CPU: MICROBLAZE
Kernel command line:
Console: xmbserial on UARTLite
Calibrating delay loop... 25.60 BogoMIPS
Memory: 32MB = 32MB total
Memory: 29744KB available (957K code, 1703K data, 44K init)
Dentry cache hash table entries: 4096 (order: 3, 32768 bytes)
Inode cache hash table entries: 2048 (order: 2, 16384 bytes)
Mount cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 8192 (order: 3, 32768 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
Microblaze UARTlite serial driver version 1.00
ttyS0 at 0xffff2000 (irq = 1) is a Microblaze UARTlite
Starting kswapd
xgpio #0 at 0xFFFFA000 mapped to 0xFFFFA000
Xilinx GPIO registered
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
```

```
eth0: LAN9115 (rev 1150001) at ffe00000 IRQ 2
uclinux[mtd]: RAM probe address=0x80125a30 size=0x174000
uclinux[mtd]: root filesystem index=0
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 2048 bind 4096)
VFS: Mounted root (romfs filesystem) readonly.
Freeing init memory: 44K
Mounting proc:
Mounting var:
Populating /var:
Running local start scripts.
Setting hostname:
Setting up interface lo:
Mounting /etc/dhcpc:
Starting DHCP client:
Starting inetd:
Starting thttpd:

SUZAKU.STARTER-KIT login:
```

7.3.3. ログイン

表示されているSUZAKUのログインプロンプトからrootユーザでログインします。パスワードの初期設定は「root」です。

表 7-1　SUZAKU 初期設定時のユーザとパスワード

ユーザ名	パスワード
root	root

7.3.4. ネットワークの設定

出荷状態の SUZAKU は DHCP で IP を取得するように設定されています。お使いの環境に DHCP サーバがない場合は固定 IP を割り当てる必要があります。以下のコマンドを入力し、固定 IP を割り当ててください。”例 7-2 固定 IP アドレスの割り当て”の 192.168.11.234 の部分には適当な IP アドレスを入力してください。固定 IP を割り当てる時は SUZAKU 上の特権ユーザで実行してください。

例 7-2　固定 IP アドレスの割り当て

```
#ifconfig eth0 down
#ifconfig eth0 192.168.11.234
```

ネットワークの設定は以下のコマンドで表示されます。

例 7-3　ネットワークの設定の表示

```
#ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:11:0C:12:34:56
          inet addr:192.168.11.234  Bcast:192.168.10.255  Mask:255.255.255.0
          UP BROADCAST NOTRAILERS RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:114 errors:0 dropped:0 overruns:0 frame:0
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
```

7.3.5. ウェブ

出荷状態の SUZAKU では、thttpd という小さな HTTP サーバが起動しています。”例 7-3 ネットワーク設定の表示”で表示された IP アドレス(例では 192.168.11.234)にお使いのウェブブラウザでアクセスすることで、動作確認ができます。http://IP アドレス にアクセスしてください。

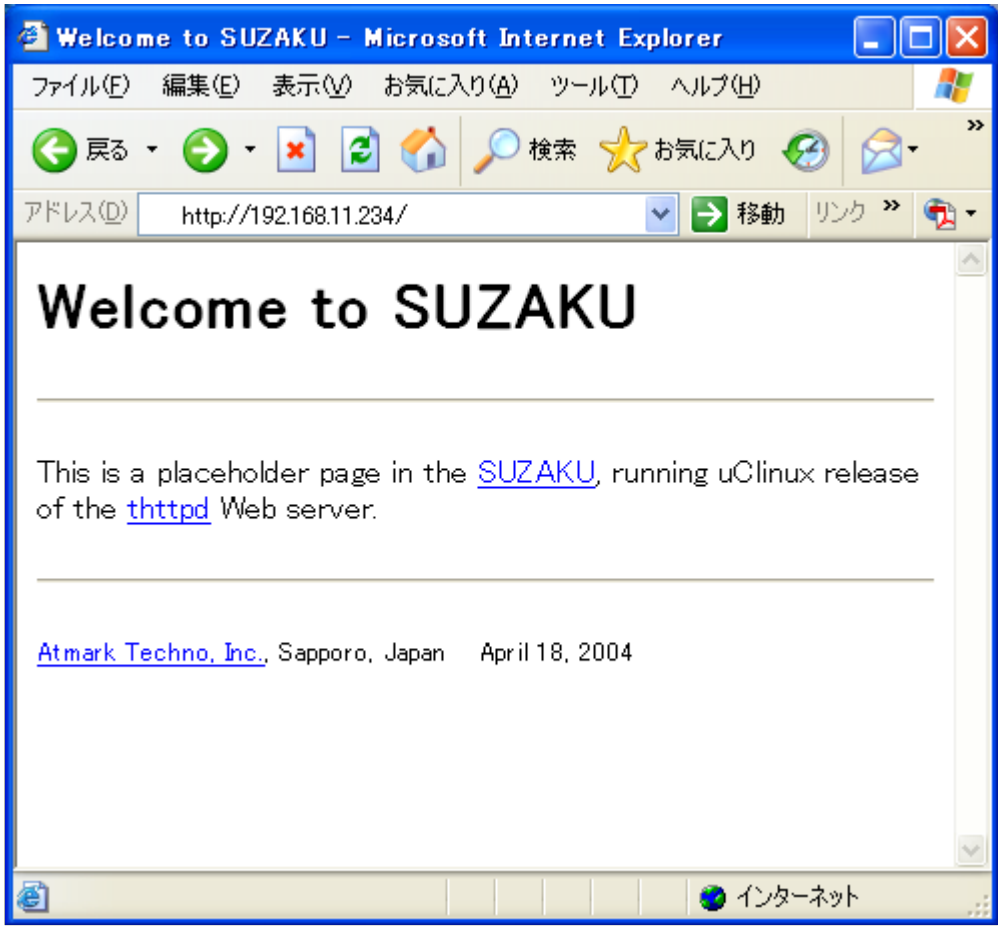


図 7-6 SUZAKU Web Page

さらに 7 セグメント LED を制御できる CGI が入っています。”http://IP アドレス/7seg-led-control.cgi”にアクセスしてください。

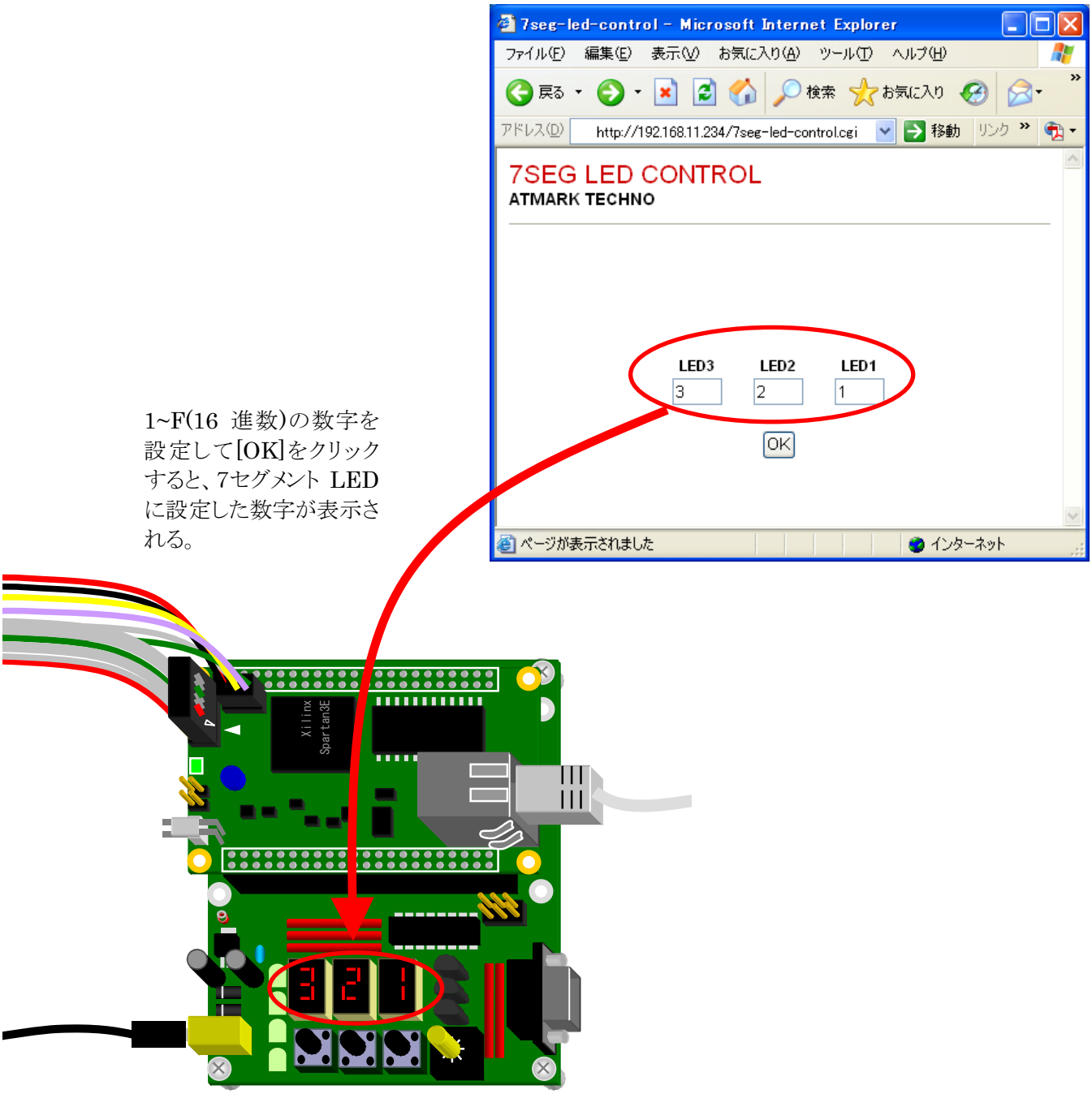


図 7-7 CGI を動かしてみる

7.3.6. 終了方法

SUZAKU ではルートファイルシステムを読み込み専用にすることで、電源即断時のルートファイルシステムの破損を回避しています。このため通常 SUZAKU を終了する場合は電源を切るだけで終了することができます。ただし、SUZAKU が設定ファイルを保存するために採用している Flat Filesystem では、情報を書き出している間の電源断には対応していません。電源を切ることによって Flat Filesystem 上のデータを失う可能性があります。また、SUZAKU をカスタマイズすることでルートファイルシステムを出荷時の ROMFS から JFFS2 に変更することが可能ですが、この場合電源を切ることによって保存したはずのデータを失う可能性があります。

詳しくは Flat Filesystem や Flat Filesystem Daemon のマニュアル、JFFS2 のマニュアルをご覧ください。

7.4. ブートローダモードでスロットマシンを動かす

ブートローダモードでスロットマシンを動かします。
JP1にジャンパプラグをさしてショートさせてください。

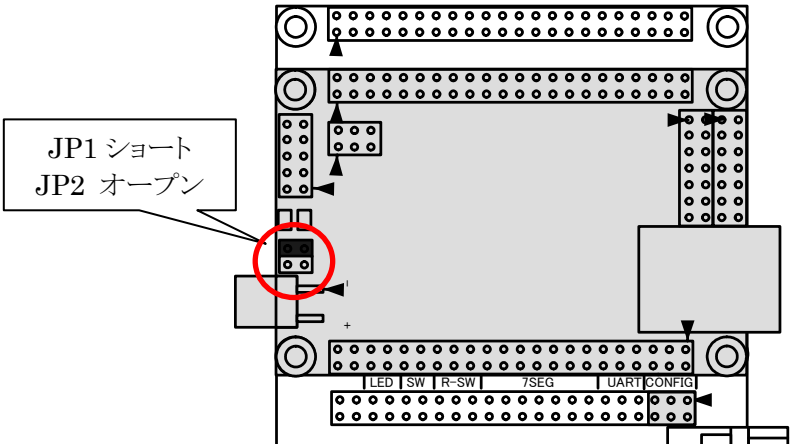


図 7-8　ブートローダモード ジャンパの設定

7.4.1. スロットマシン起動

シリアル通信用ソフトウェアが起動されていることを確認してから AC アダプタ 5V を接続し、電源を供給してください。シリアル通信用ソフトウェアの画面に以下が表示され、スロットマシンが動きます。

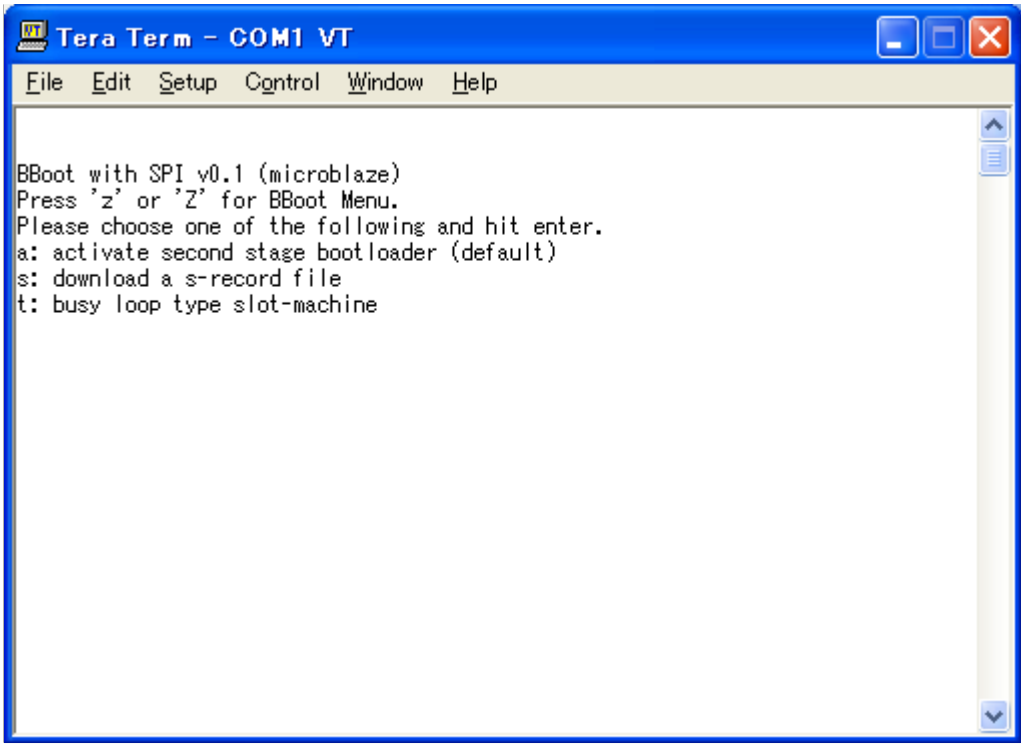


図 7-9　スロットマシンの起動(SZ130 の場合)

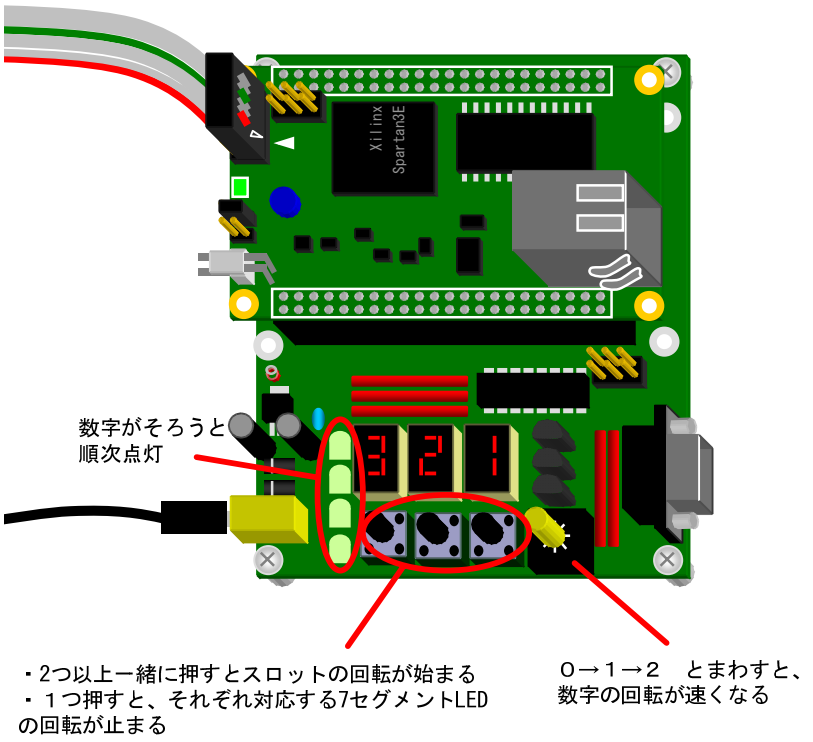


図 7-10　スロットマシンを動かしてみよう

8.ISE の使い方

FPGA 側から SUZAKU の開発をするためには、ISE の使い方を知ることが必要不可欠です。ISE は Xilinx が提供する FPGA の統合型設計環境です。GUI 統合ツール Project Navigator で、FPGA に必要な論理合成、配置配線、bit ファイルの書き込みのツールなど、トータルな開発環境を提供しています。

ここでは LED/SW ボードの単色 LED(D1)を点灯させると共に ISE の使い方を簡単に説明します。本書では ISE8.1i を使用しています。ISE の使い方の詳細は ISE のヘルプ、マニュアル等を参照してください。ISE には日本語のヘルプ、マニュアル等も用意されています。

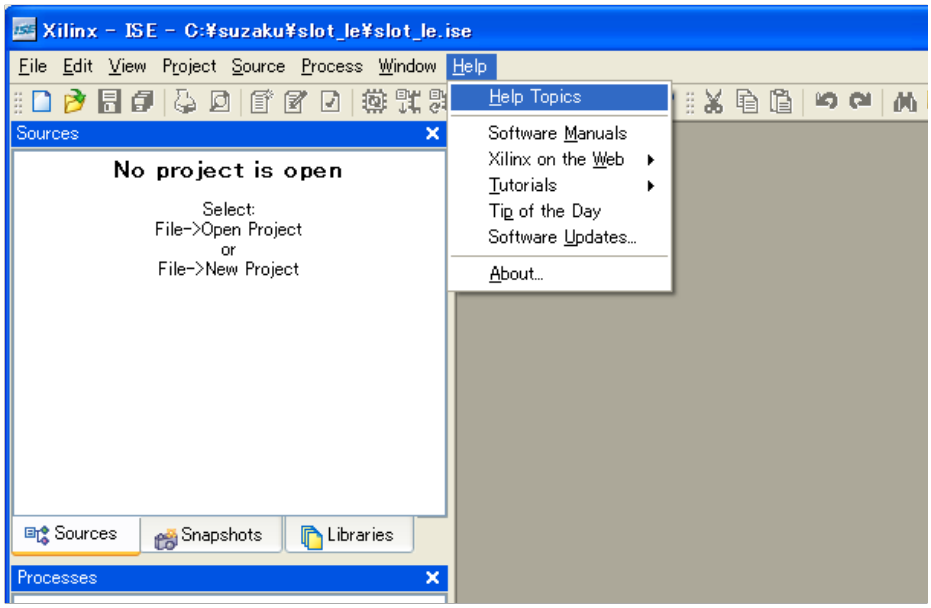


図 8-1 ISE のヘルプ、マニュアル等

8.1. 単色 LED 周辺回路

単色 LED 周辺回路は下図のようになっています。それぞれ 180Ωの抵抗で 3.3V にプルアップされています。FPGA から"Low"を出力すると、単色 LED が点灯し、"High"を出力すると、単色 LED が消灯します。

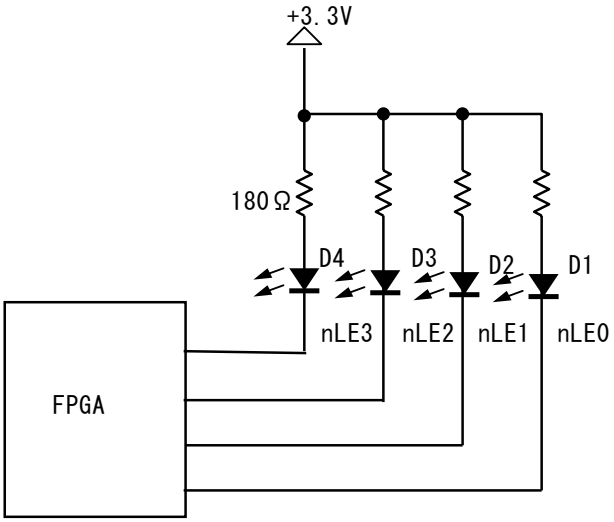


図 8-2 単色 LED 周辺回路

● FPGA の入力、出力について

SUZAKU の FPGA の I/O ピンは C-MOS＋3.3V に設定されています。FPGA からは"Low"で 0.4V 以下、"High"で 2.9V 以上が出力されます。FPGA へは 0.8V 以下で"Low"、2.0V 以上で"High"が入力されます。ただし、ディジタル入力定格は-0.3V～3.6V なので、それを超えて入力しないでください。

表 8-1 FPGA 入力、出力

	Low(V)	High(V)
出力	OUT＜0.4	2.9＜OUT
入力	-0.3＜IN＜0.8	2.0＜IN＜3.6

8.2. プロジェクトの新規作成

Project Navigator を起動してください。[File]→[New Project]をクリックしてください。

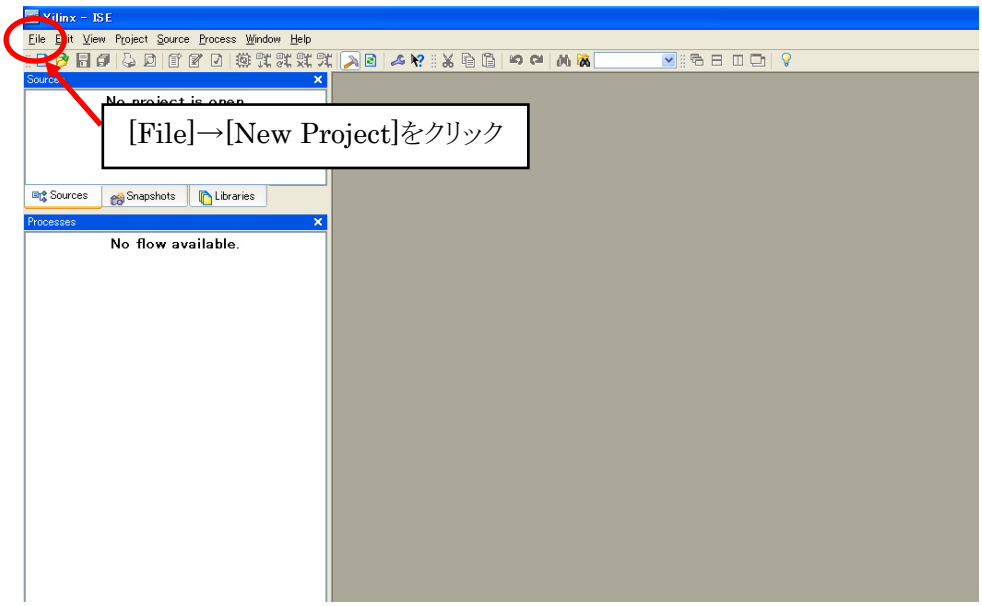


図 8-3　Project Navigator 起動

New Project Wizard が表示されます。[Project Location]の[…]をクリックし、プロジェクトのディレクトリパスを指定します。ここでは C¥suzaku とします。[Project Name]に プロジェクト名を入力します。 slot_le と入力し、[Top-Level Source Type]が[HDL]となっていることを確認し、[Next]をクリックしてください。

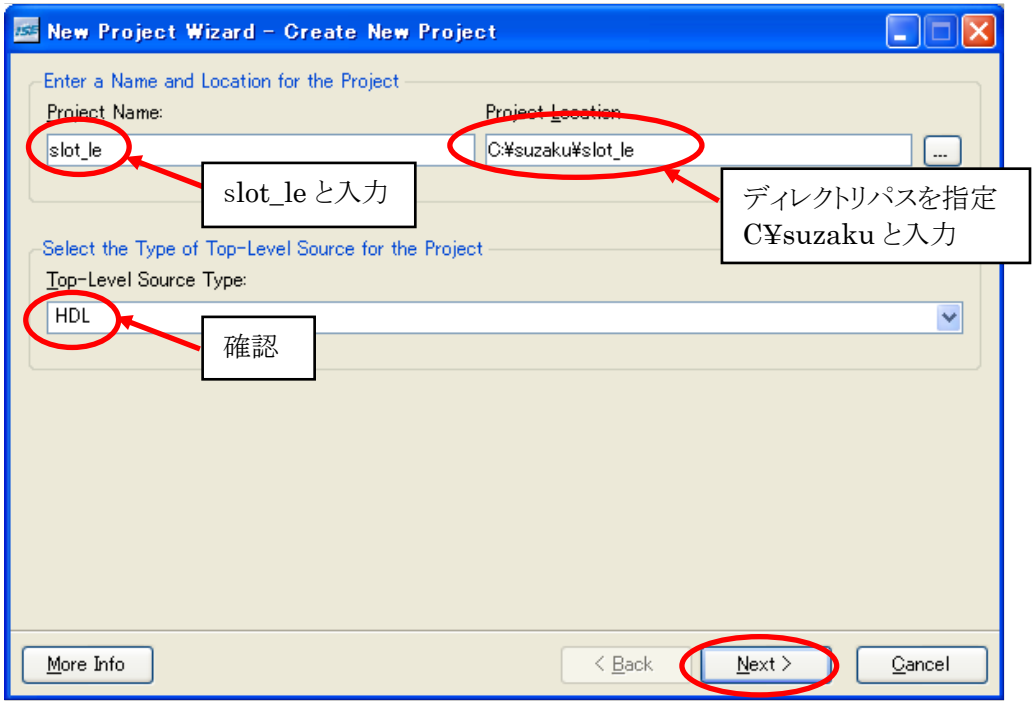


図 8-4　プロジェクトの新規作成

8.3. デバイスの選択

SUZAKU に実装されている FPGA デバイスを選択します。お使いの SUZAKU の型式の設定にし、[Next]をクリックしてください。

型式	SZ010	SZ030	SZ130	SZ310
Product Category	All			
Family	Spartan3		Spartan3E	Virtex2P
Device	XC3S400	XC3S1000	XC3S1200E	XC2VP4
Package	FT256		FG320	FG256
Speed	－4			－5
Synthesis Tool	XST(VHDL/Verilog)			
Simulator	ISE Simulator(VHDL/Verilog)			

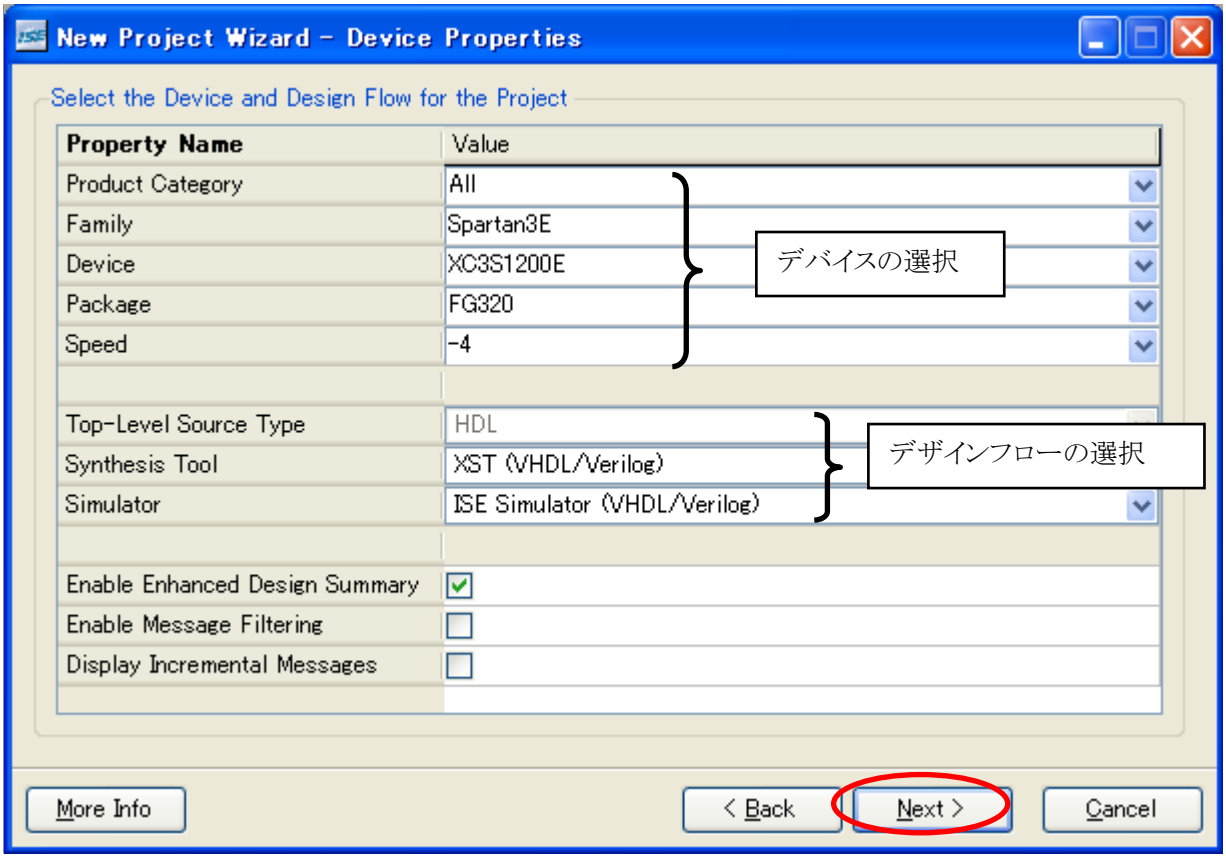


図 8-5　デバイスの選択(SZ130 の場合)

8.4. ソースファイルの作成

[New Source]をクリックしてください。

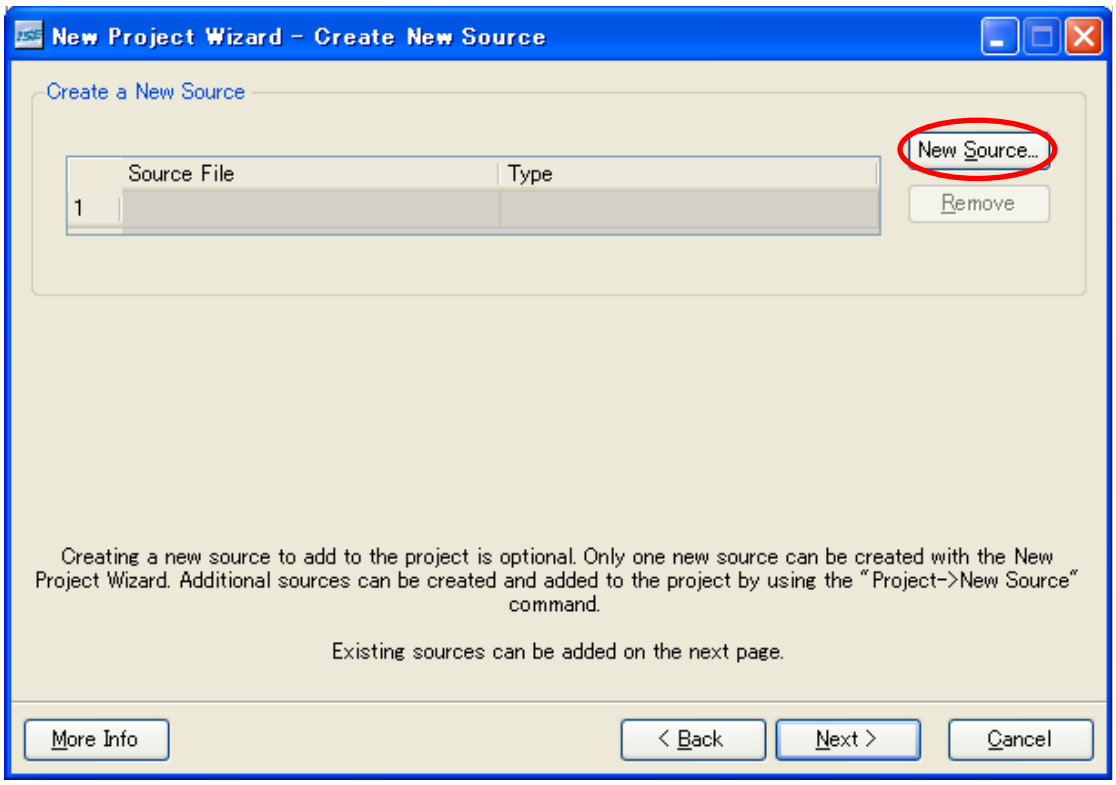


図 8-6 New Source 作成

[VHDL Module]を選択し、[File name]に top と入力し、[Next]をクリックしてください。VHDL ソースファイルが作成されます。

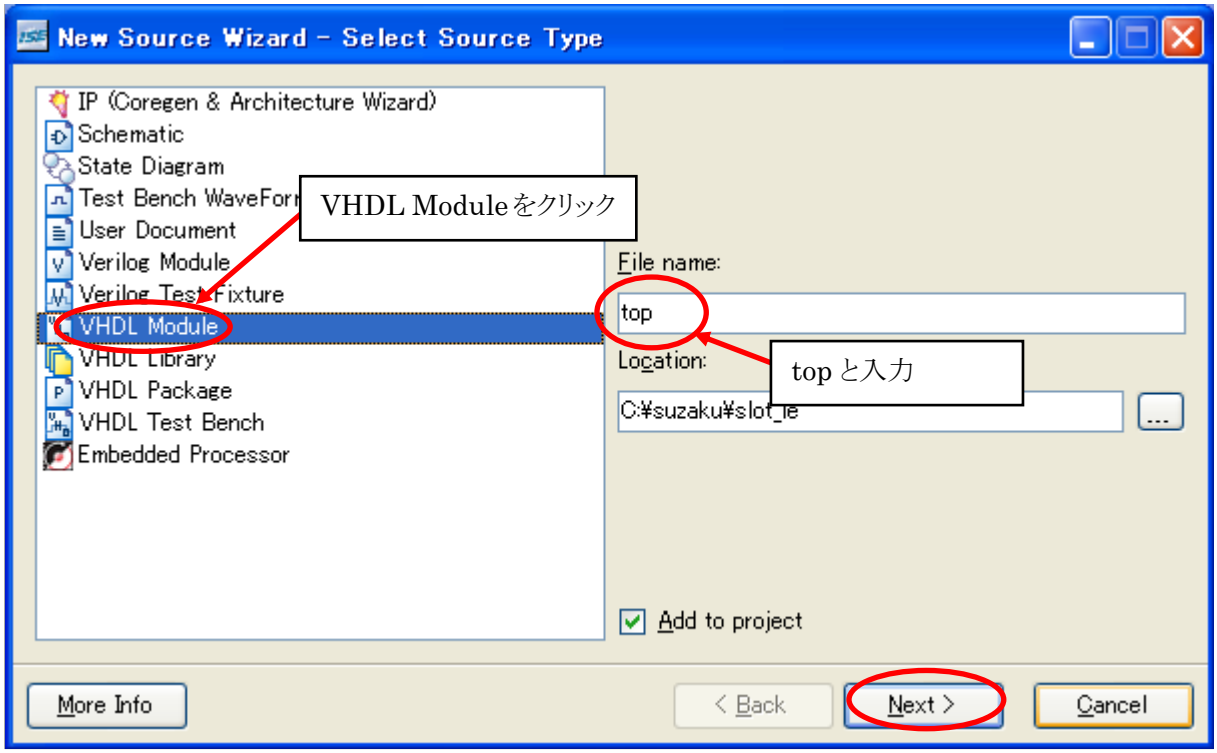


図 8-7 VHDL ソースファイル作成

[Architecture Name]を IMP に変更し、[Next]をクリックしてください。

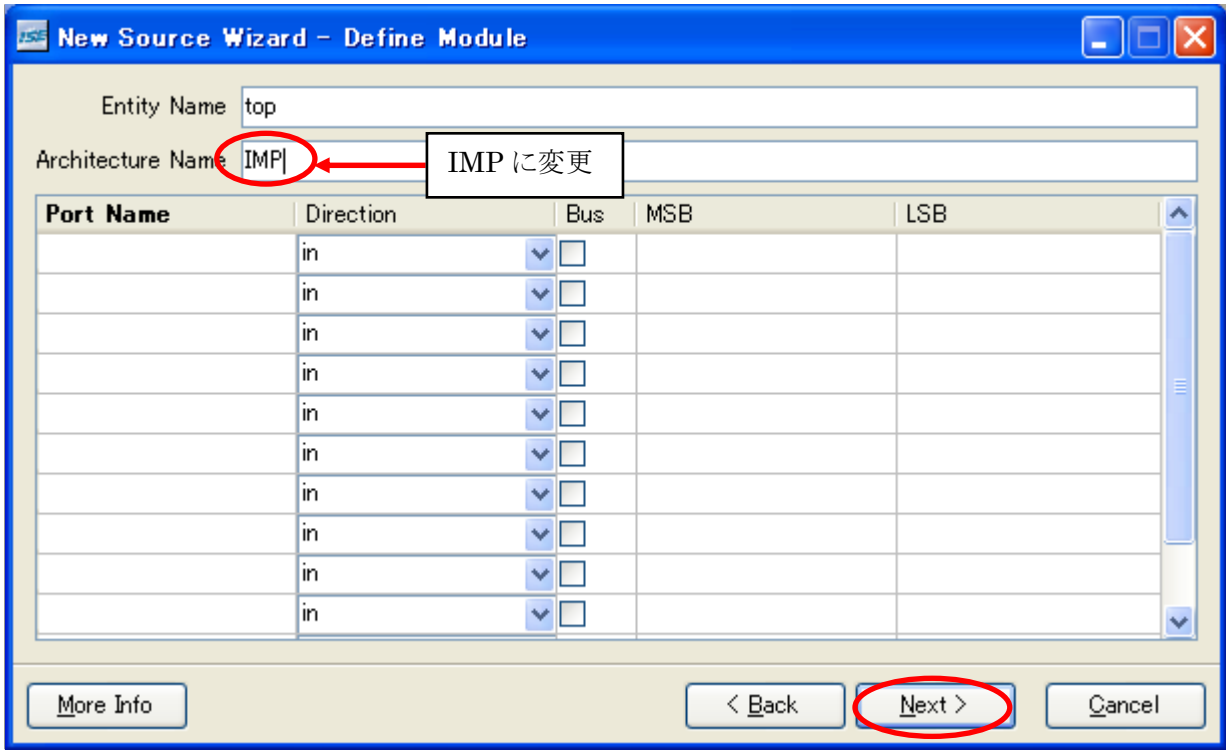


図 8-8　アーキテクチャ名定義

今作った VHDL ソースファイルの設定が表示されます。間違いがなければ[Finish]をクリックしてください。

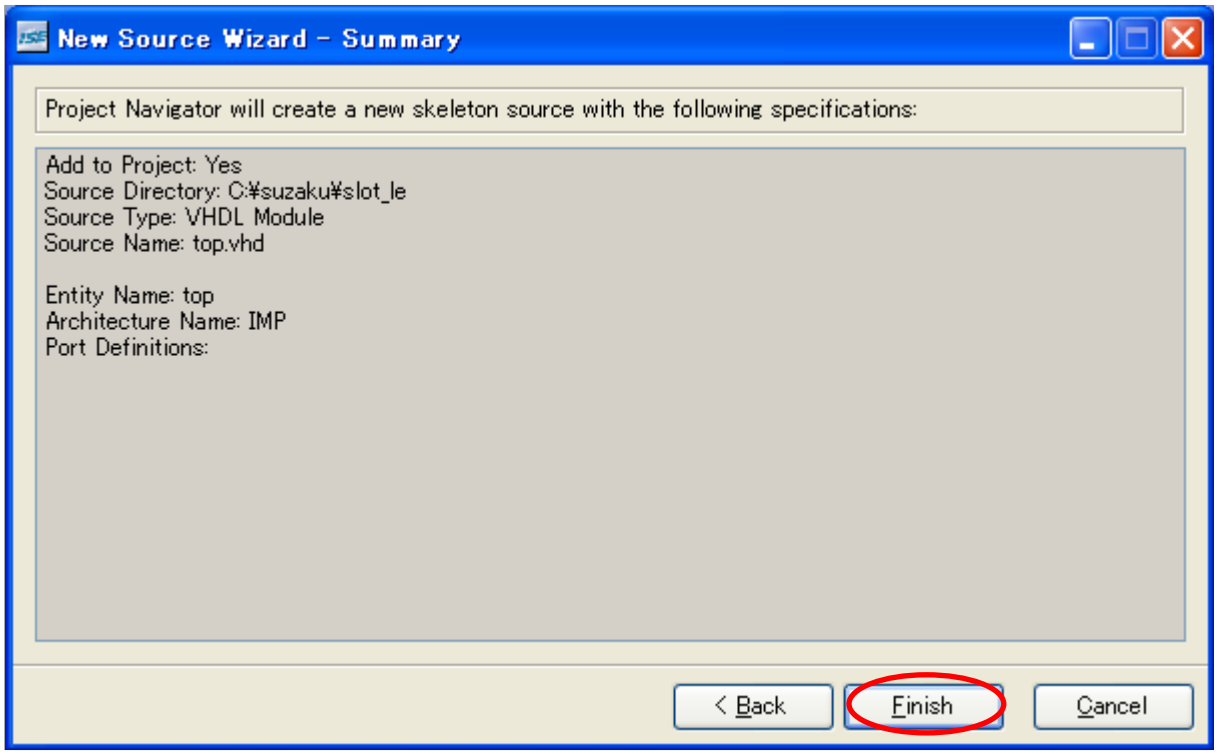


図 8-9　ソースファイル作成確認画面

以下の画面が出るまで[Next]をクリックし、最後に[Finish]をクリックしてください。

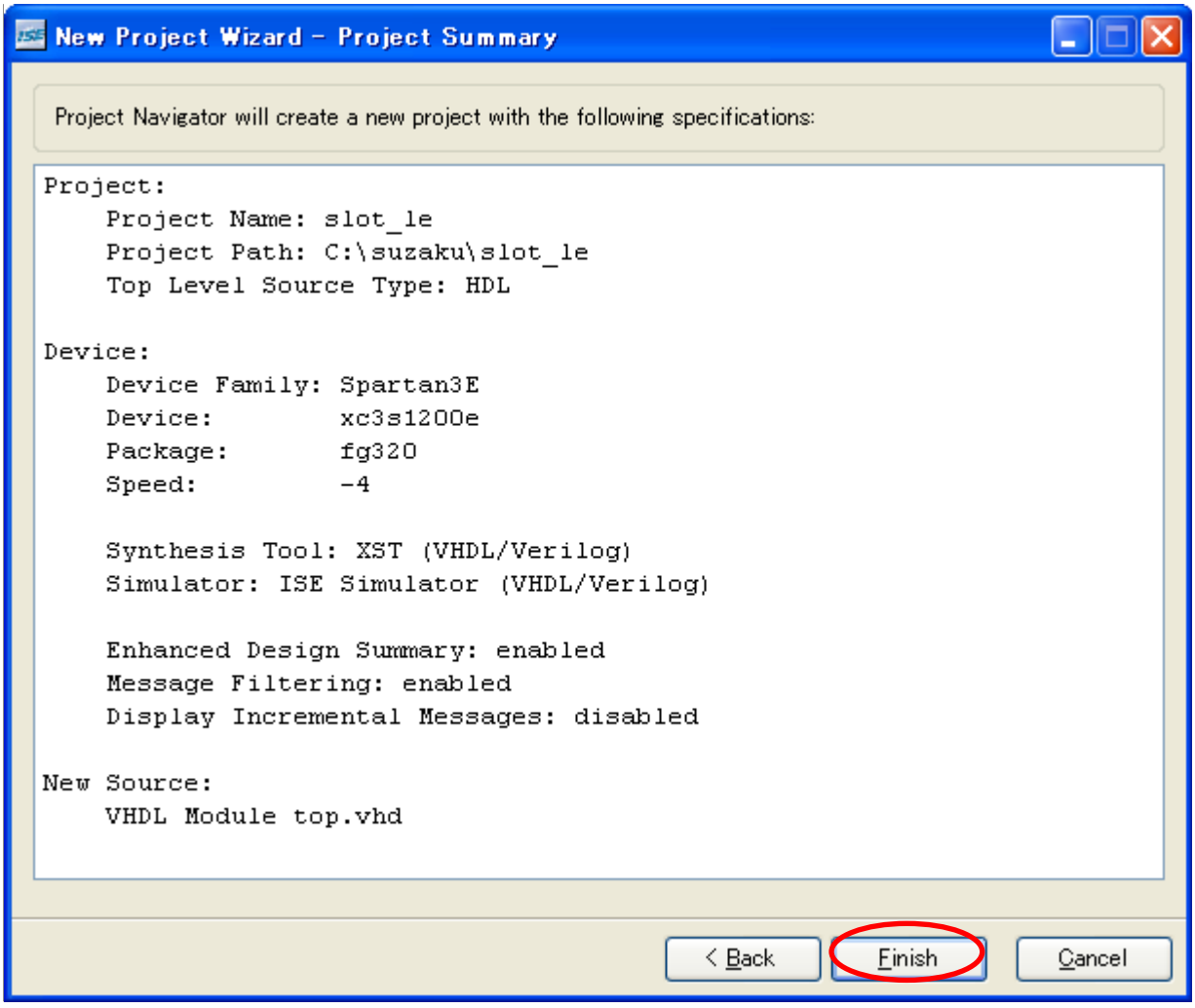


図 8-10 最終確認画面(SZ130 の場合)

以上で新規プロジェクトおよび VHDL ソースファイルが出来上がります。
top-IMP(top.vhd)をダブルクリックしてください。top.vhd が開きます。

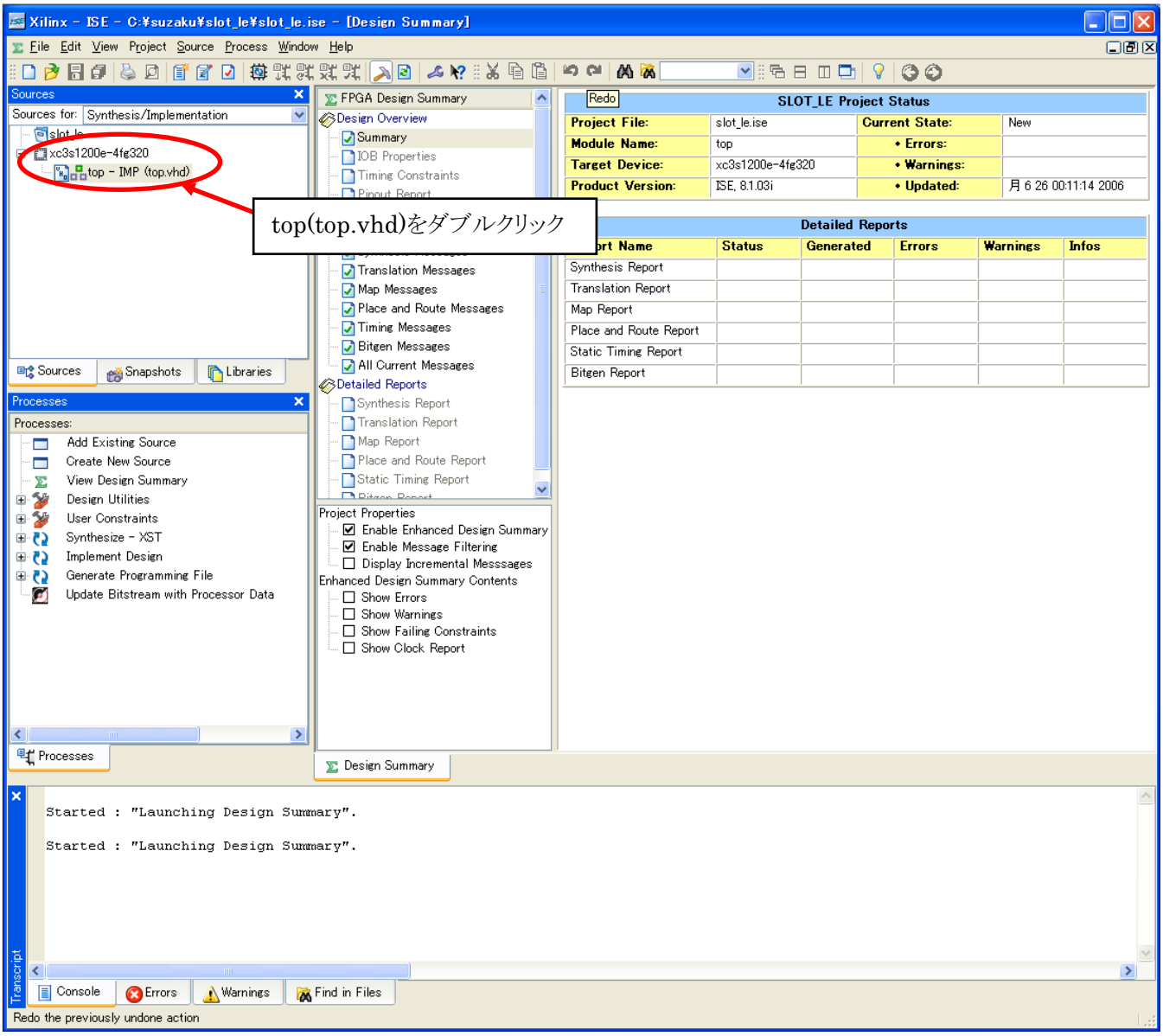
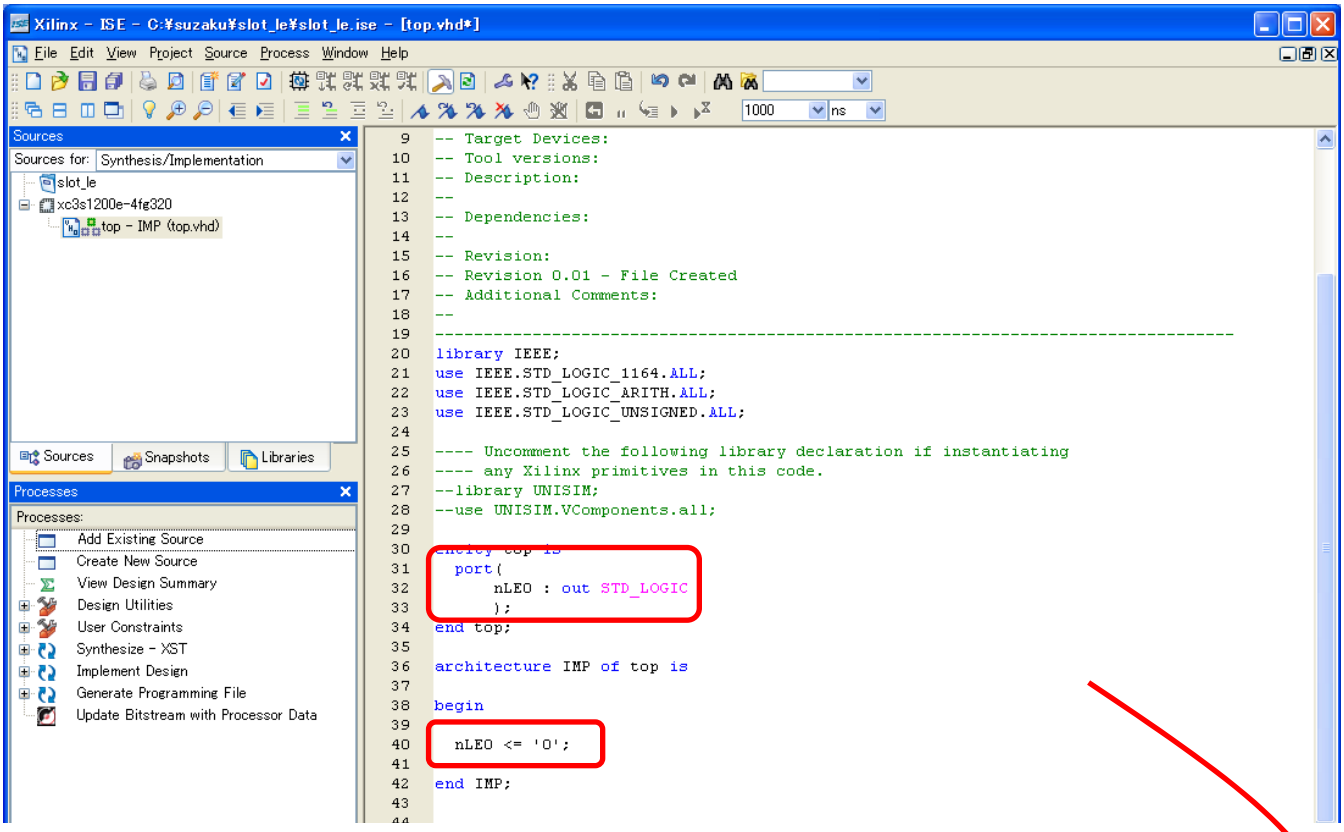


図 8-11 新規プロジェクト、ソースファイル作成完了

8.5. ソースコードの入力

テンプレートが自動生成されています。以下のように単色 LED への出力信号の定義と単色 LED を点灯させる文を追加してください。




```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;
entity top is
  port (
        nLED : out  STD_LOGIC
        );
end top;

architecture IMP of top is

begin
        nLED <= '0';
end IMP;
```

出力信号の定義

単色 LED を点灯させる

図 8-12 ソースコード入力

追加できたら、[File]→[Save]をクリックして保存してください。

8.6. 文法チェック

トップモジュールtop – IMP(top.vhd)を選択し、Synthesize をダブルクリックしてください。Synthesize をダブルクリックすると文法チェックが始まります。ソースコードに間違いがなければ Synthesize の横にチェックマークもしくは警告マークが付きます。もしエラーマークになった場合はログをチェックし、ソースコードを見直してください。ソースコードを修正して保存するとマークが疑問符に変わるので、再び Synthesize をダブルクリックし、エラーマークがなくなるまで繰り返してください。

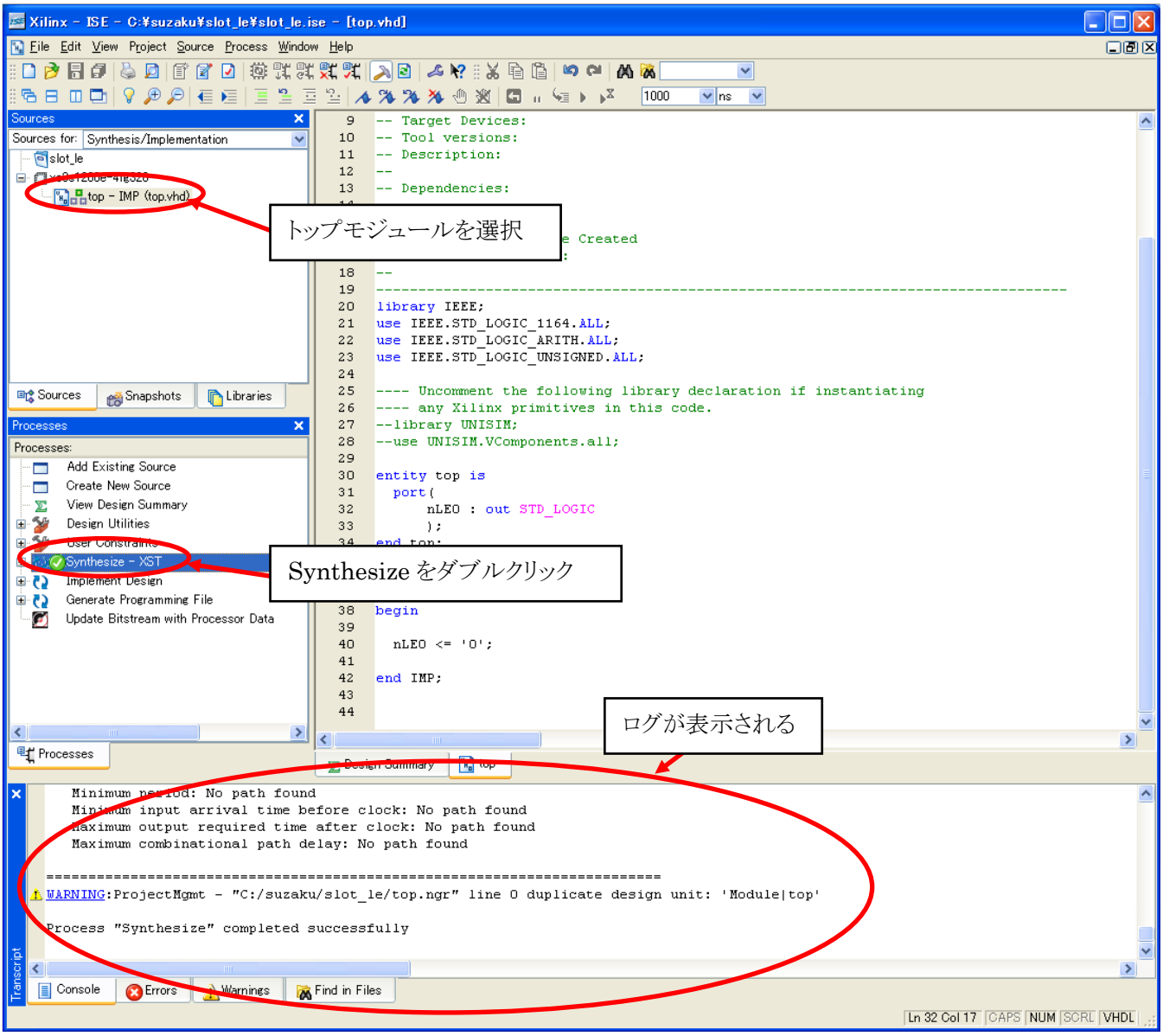


図 8-13 文法チェック

8.7.インプリメント

Implement Design の横の⊞、Translate の横の⊞をクリックして開いてください。Assign Package Pins Post-Translate をダブルクリックしてください。

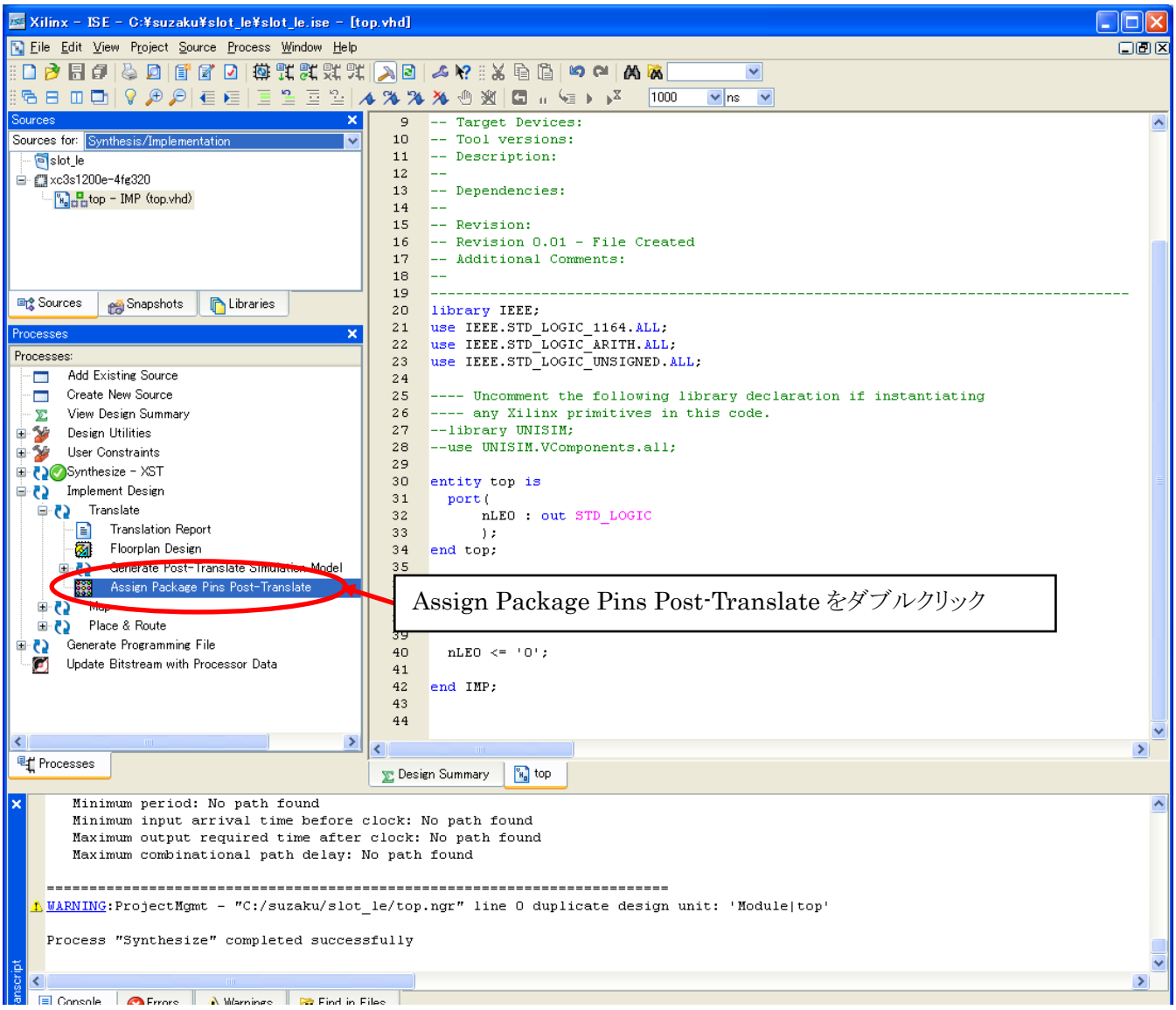


図 8-14 PACE を立ち上げる

ucf ファイルを追加してもいいかという質問をされるので[Yes]をクリックしてください。

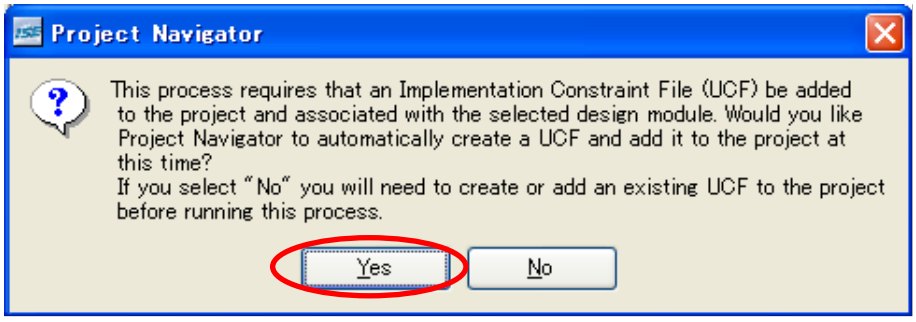


図 8-15 ucf ファイル作成確認

PACEというピンアサインを設定できるソフトが立ち上がります。”表 6-2 機能用ピンアサイン” を参照し、D1 の単色 LED とつながっている FPGA のピンを割り当てます。

表 8-2 nLE0 ピンアサイン

	SZ010 SZ030	SZ130	SZ310
nLE0	B12	E12	L16

[File]→[Save]をクリックして保存し、PACE を閉じてください。

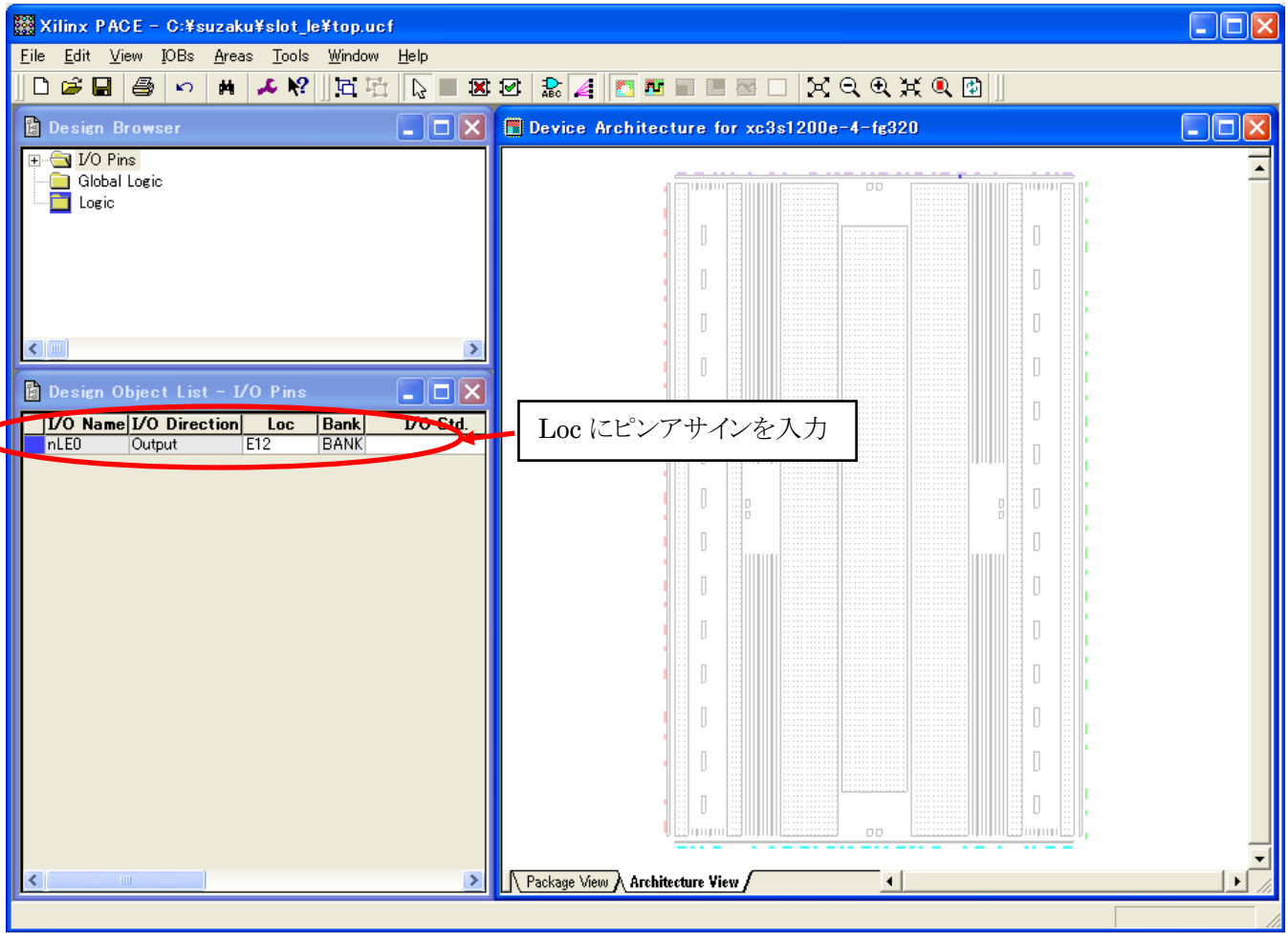


図 8-16 PACE によるピンアサイン(SZ130 の場合)

Project Navigator に戻って top-IMP(top.vhd)の横の⊞をクリックして開いてください。ピンアサインのファイル”top.ucf”が出来上がっています。今回は PACE でピンアサインしましたが、他の方法でピンアサインすることもできます。各々のやり易い方法を探してみてください。

PACE で設定したピンアサインを Text で編集することもできます。top.ucf をクリックすると、Processes のウィンドウに top.ucf のプロセスが表示されるので Edit Constraints(Text)をダブルクリックしてください。ピンアサインが Text で表示されます。

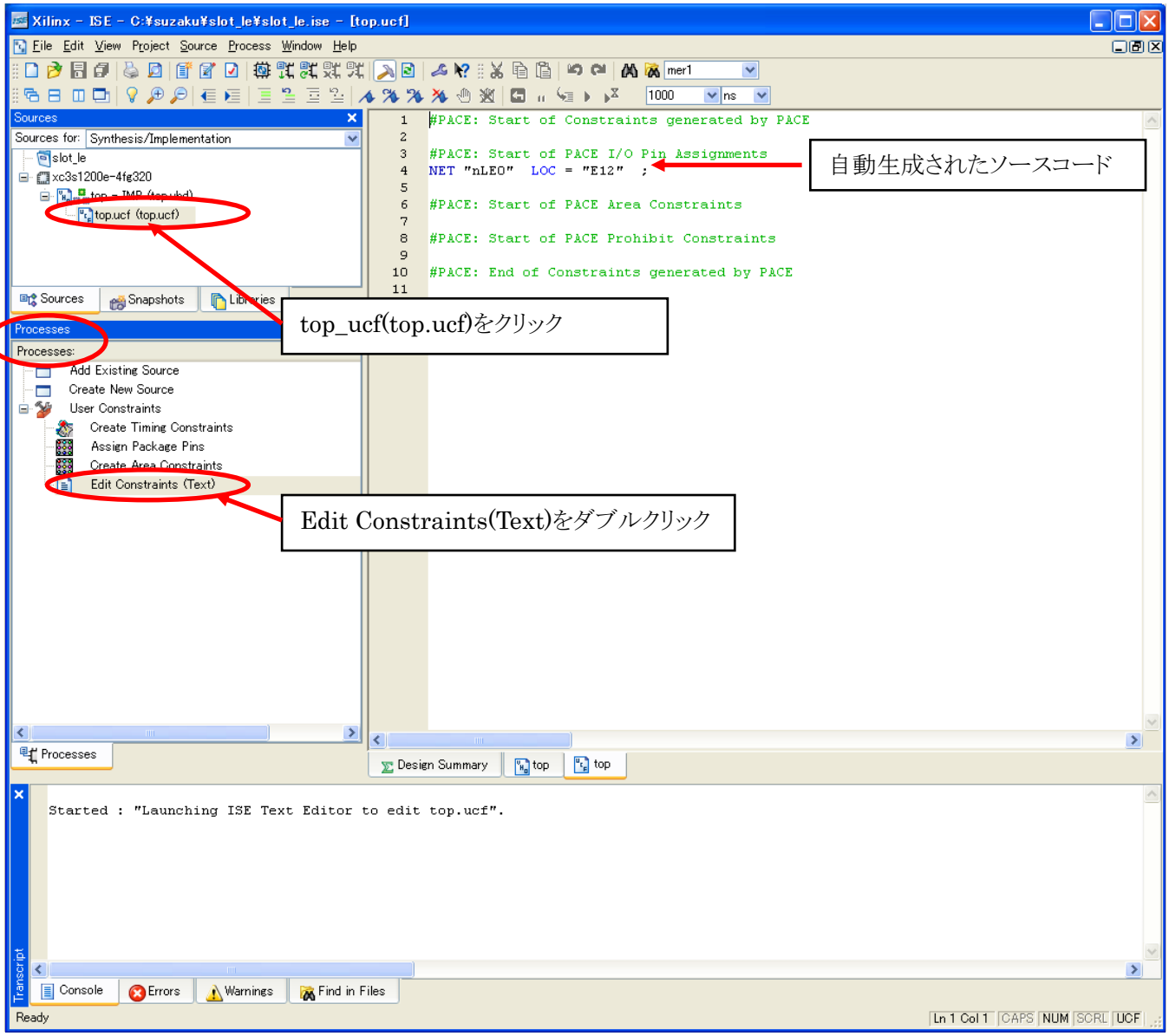


図 8-17 ピンアサインのソースコード(SZ130 の場合)

Implement Design をダブルクリックしてください。残りのインプリメントが始まります。

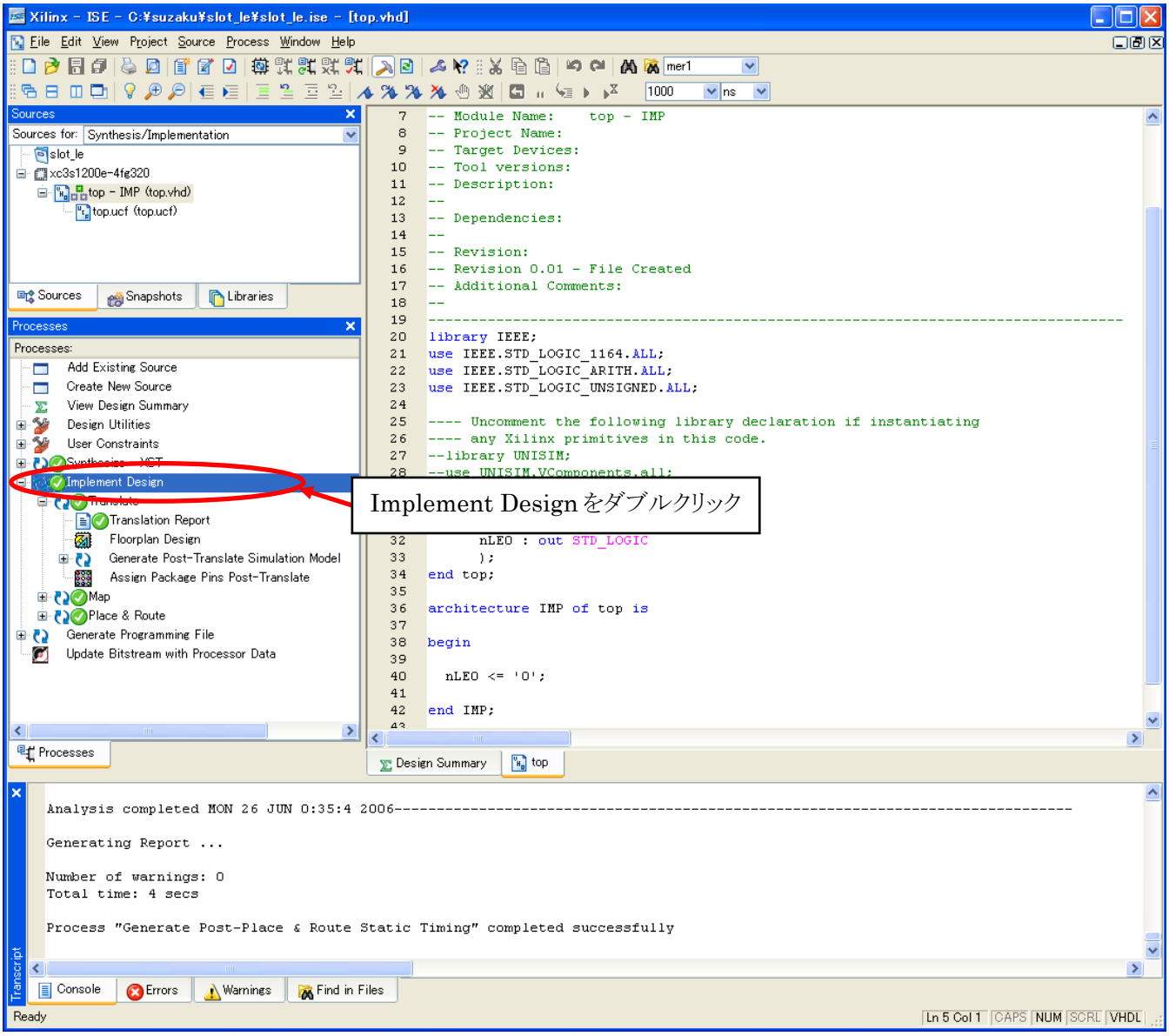


図 8-18 インプリメント

Generate Programming File をダブルクリックします。エラーがなければ、top.bit という FPGA コンフィギュレーション用の bit ファイルが作成されます。

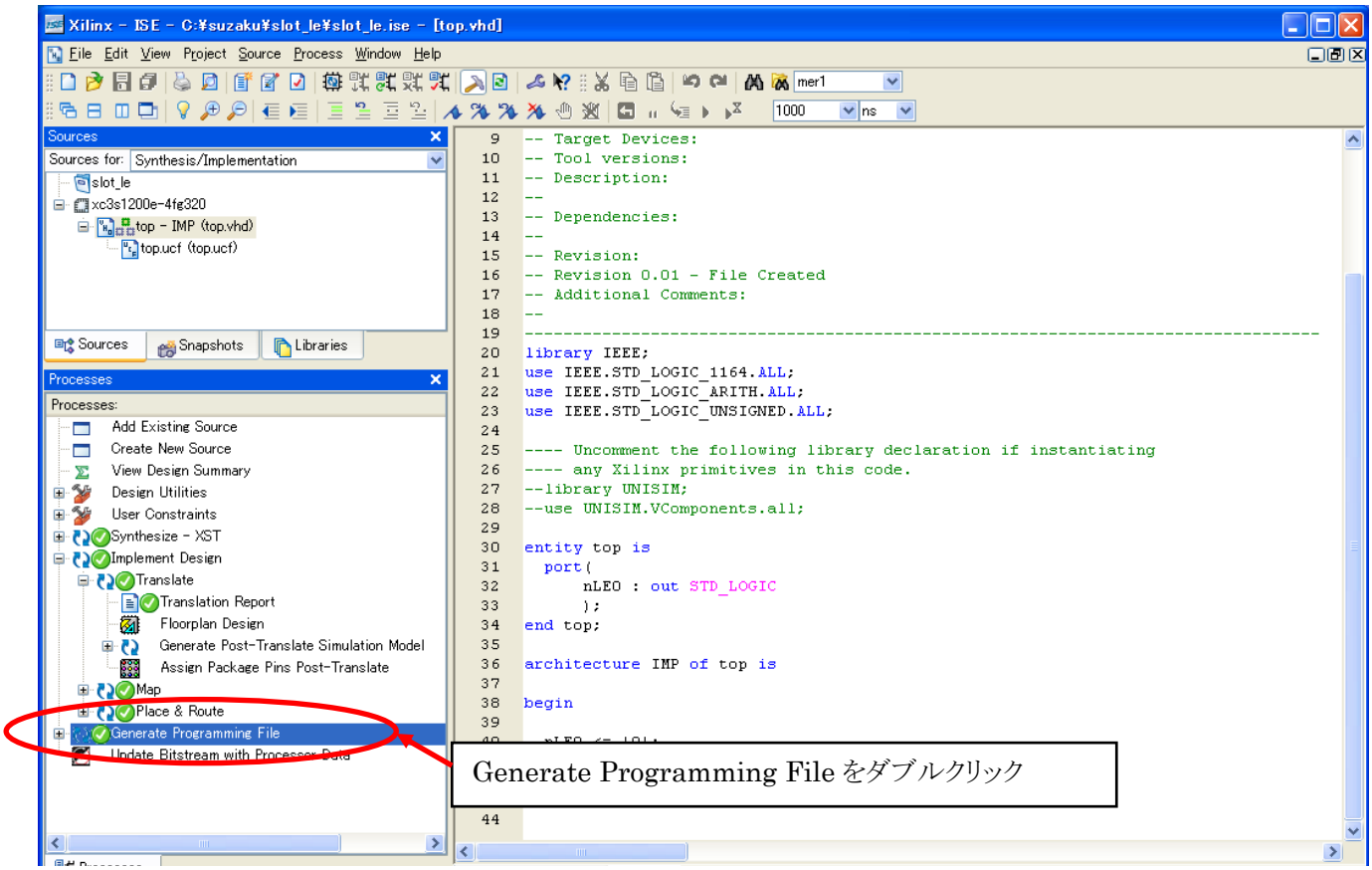


図 8-19 bit ファイル作成

8.8. コンフィギュレーション

SUZAKU の FPGA へのコンフィギュレーションには JTAG でコンフィギュレーションする方法と Flash に保存してコンフィギュレーションする方法の 2 通りがあります。

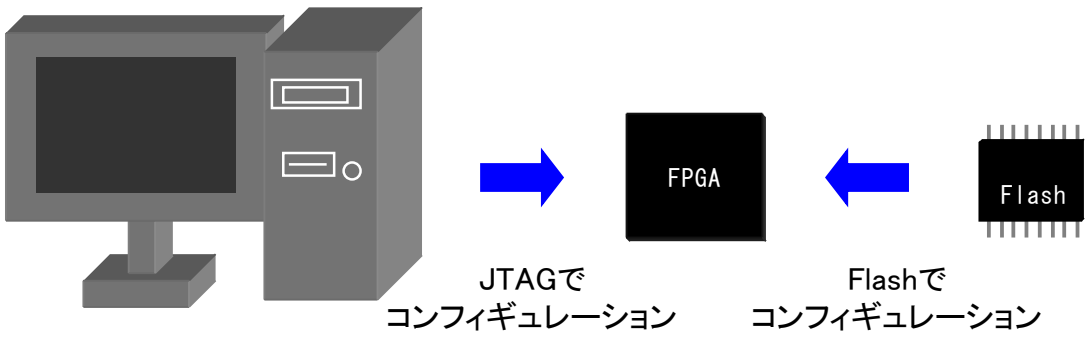


図 8-20 コンフィギュレーション

8.8.1. JTAG でコンフィギュレーション

今回は JTAG から FPGA にコンフィギュレーションします。この方法だと Xilinx の FPGA が SRAM ベースのためコンフィギュレーションは速いですが、電源を切るたびにコンフィギュレーションし直さなければなりません。しかし、デバッグ時は速さ重視でこちらの方法でコンフィギュレーションします。

まず、SUZAKU JP2 にジャンパプラグをさし、ショートさせてください。JP2 をショートさせると、電源投入時 FPGA に対し、Flash からのコンフィギュレーションを停止させることができます。

SUZAKU CON7に JTAG のダウンロードケーブル(Xilinx Parallel CableⅢまたはⅣ)を接続し、LED/SW CON6 に AC アダプタ 5V を接続し、電源を投入してください。SUZAKU D3 のパワーON LED(緑)が点灯しているか確認してください。

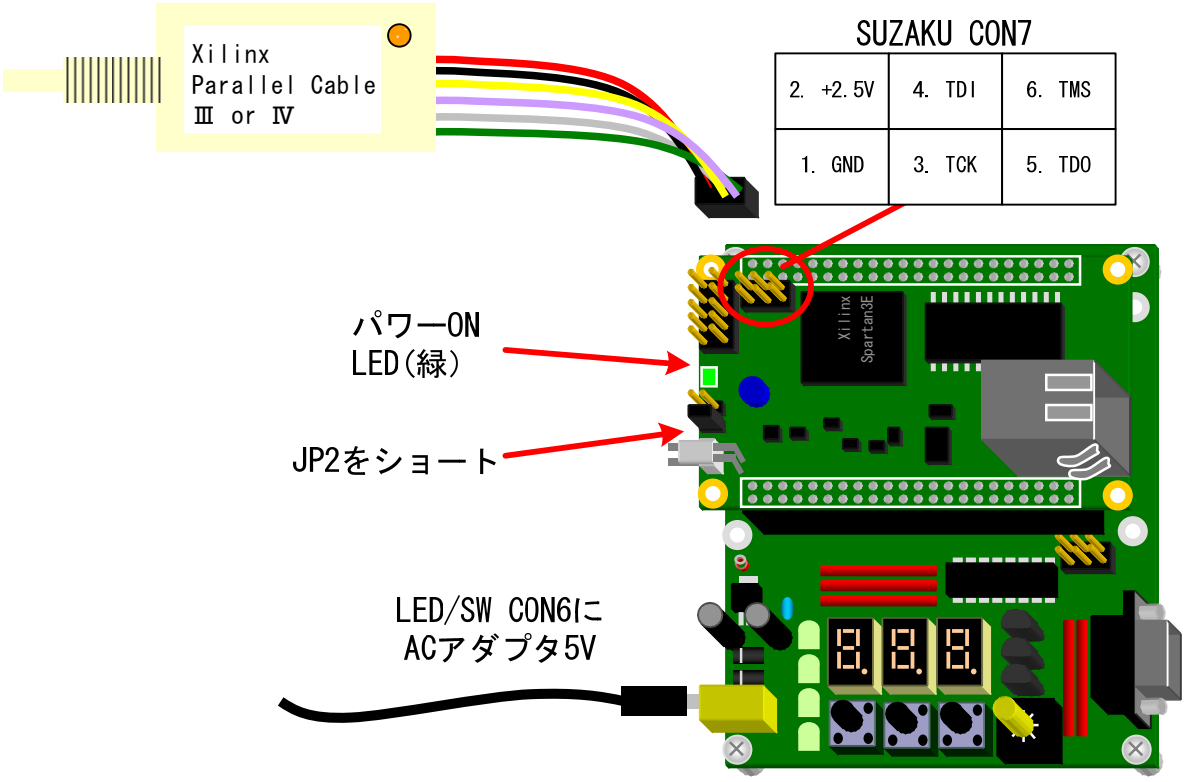


図 8-21 JTAG 書き込み

Project Navigator の Configure Device (iMPACT)をダブルクリックしてください。

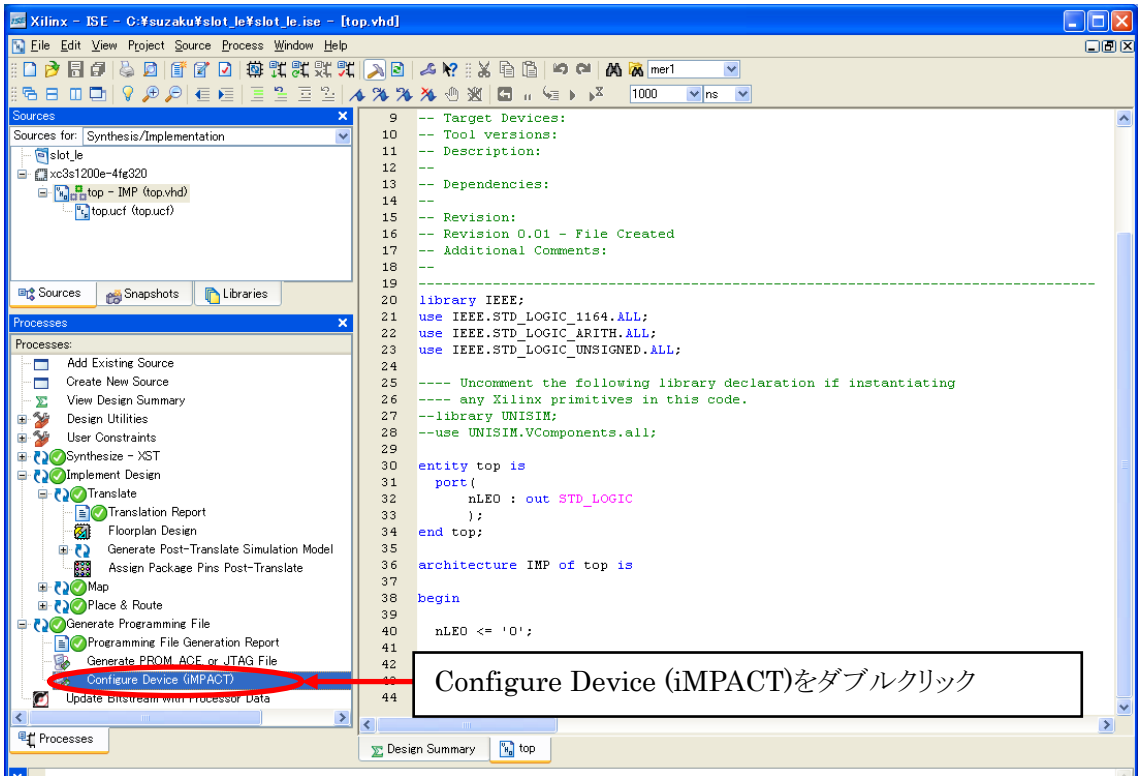


図 8-22 iMPACT 立ち上げ

下図のように iMPACT というソフトが立ち上がります。[Configure device using Boundary-Scan(JTAG)]にチェックを入れ、[Finish]をクリックしてください。

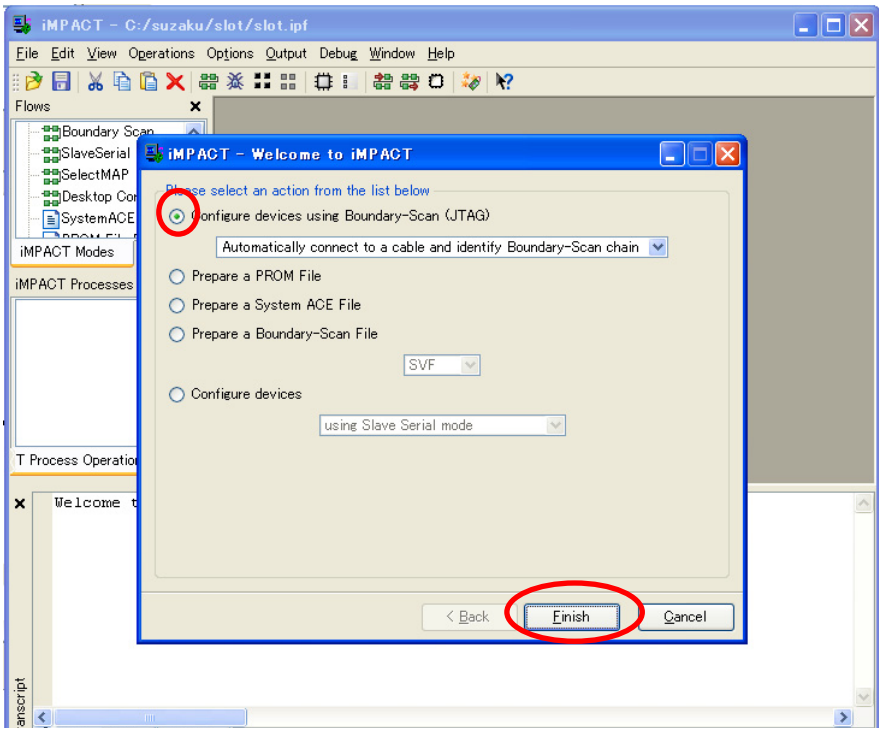


図 8-23 iMPACT 設定画面

接続ミスがなく、SUZAKU の電源が入っていればデバイスが発見されます。

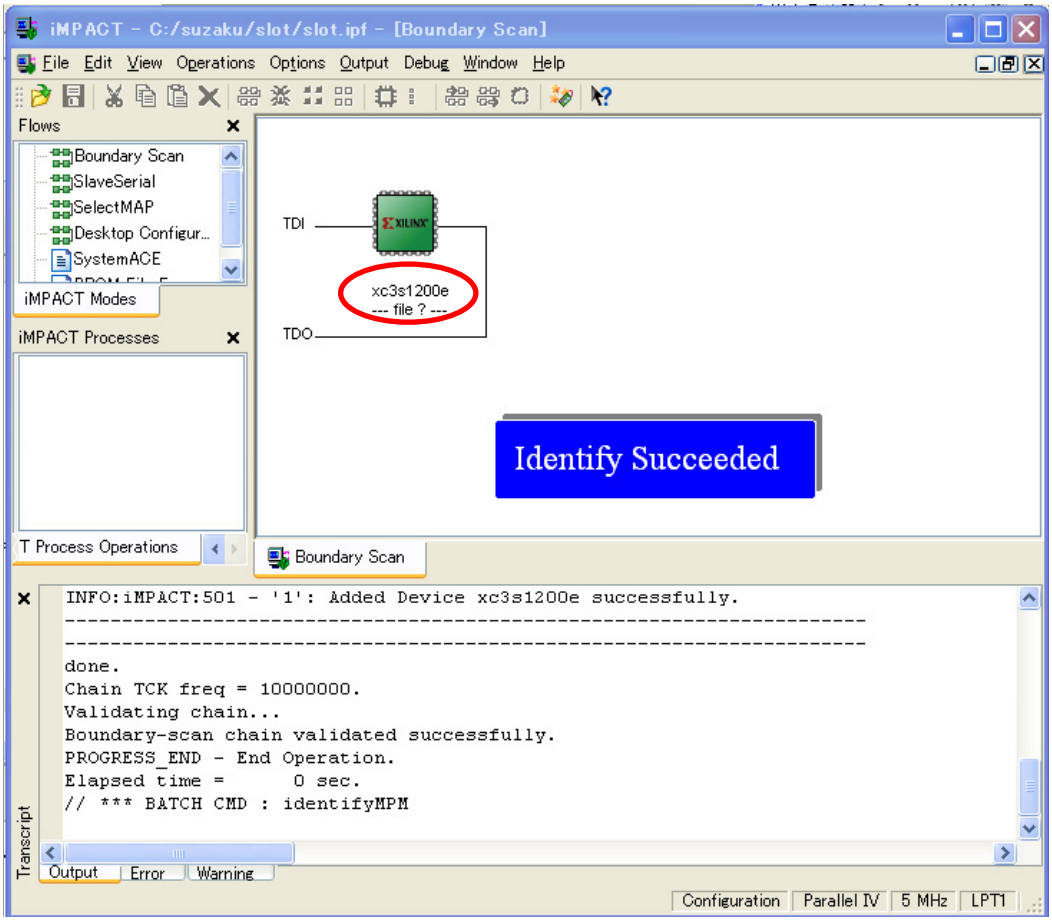


図 8-24 FPGA デバイス発見(SZ130 の場合)

先ほど作成したコンフィギュレーション用のファイル top.bit を選んで[Open]をクリックしてください。

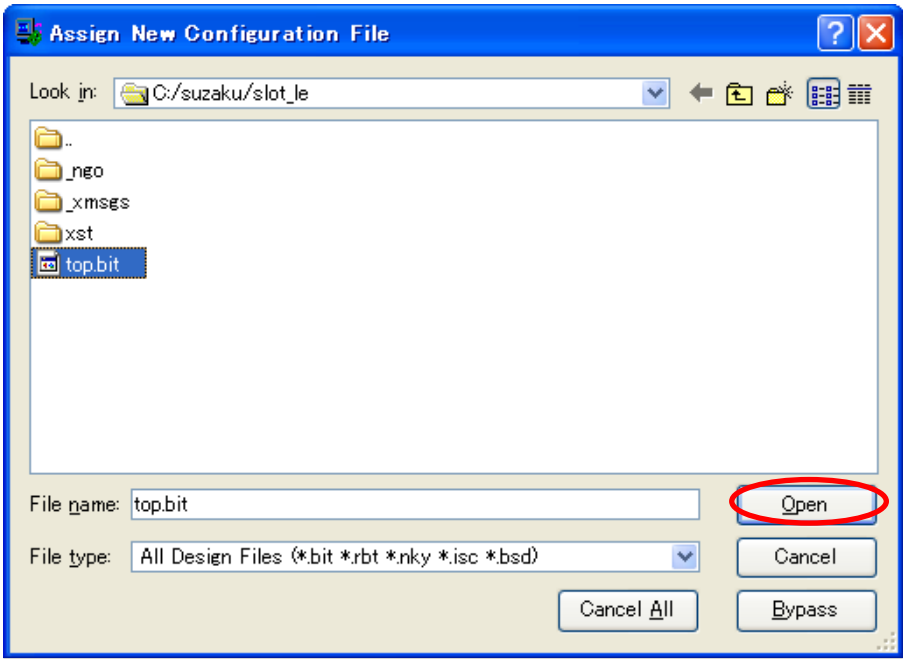


図 8-25 bit ファイル選択

この時、Warning がでますが、[OK]をクリックして下さい。

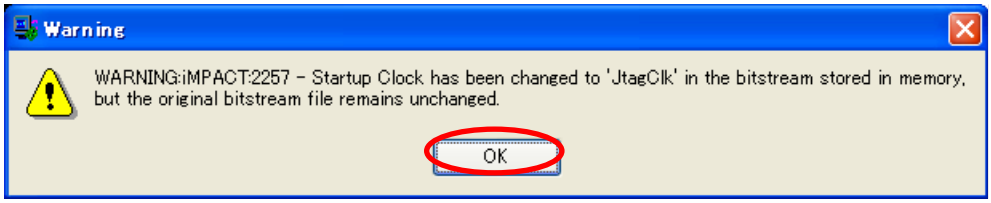


図 8-26 WARNING

デバイスをクリックし、緑色になったことを確認し、Program をダブルクリックしてください。

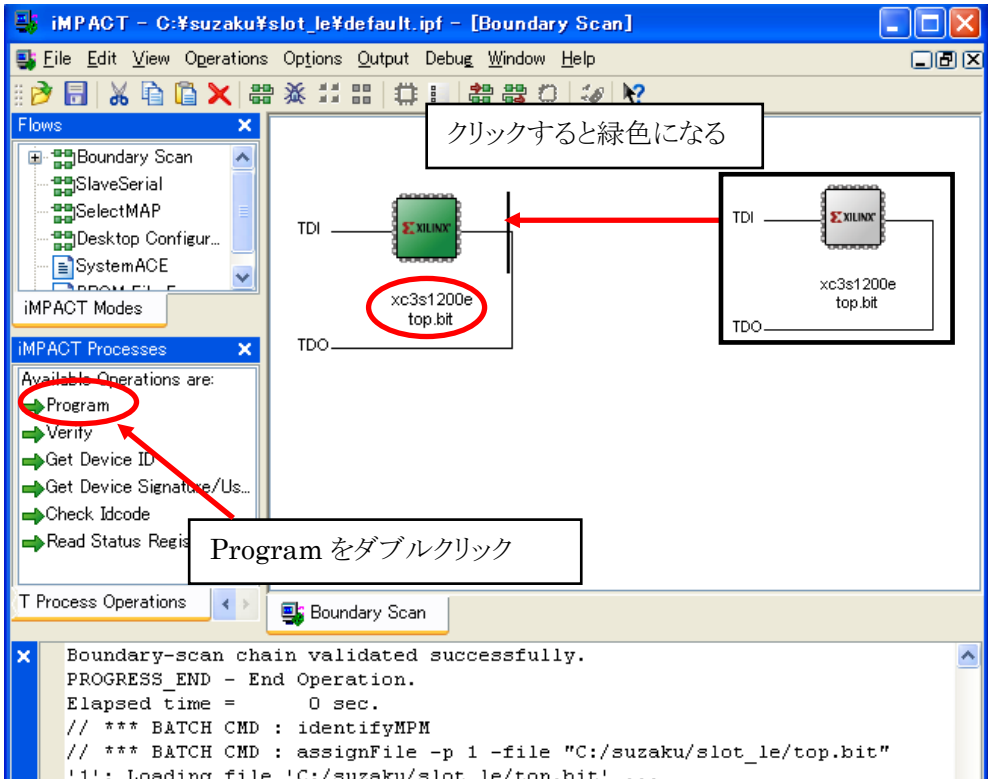


図 8-27 デバイス選択

Verify のチェックボタンをはずし、[OK]をクリックしてください。コンフィギュレーションが始まります。

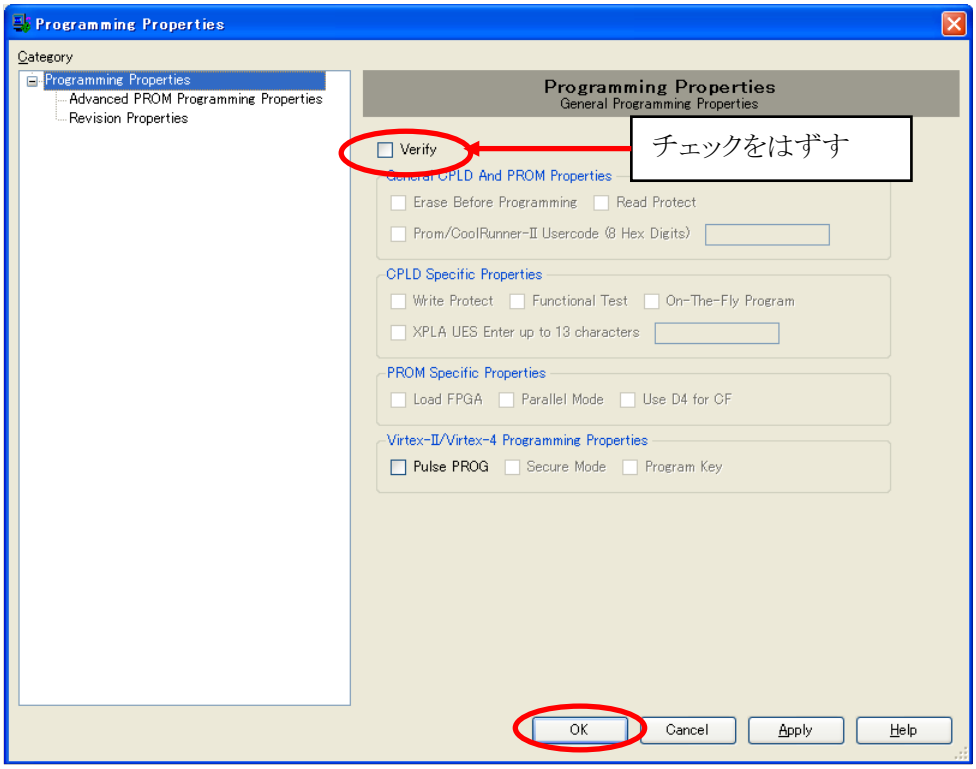


図 8-28 Program 設定

Program Succeeded と表示されれば、コンフィギュレーション成功です。

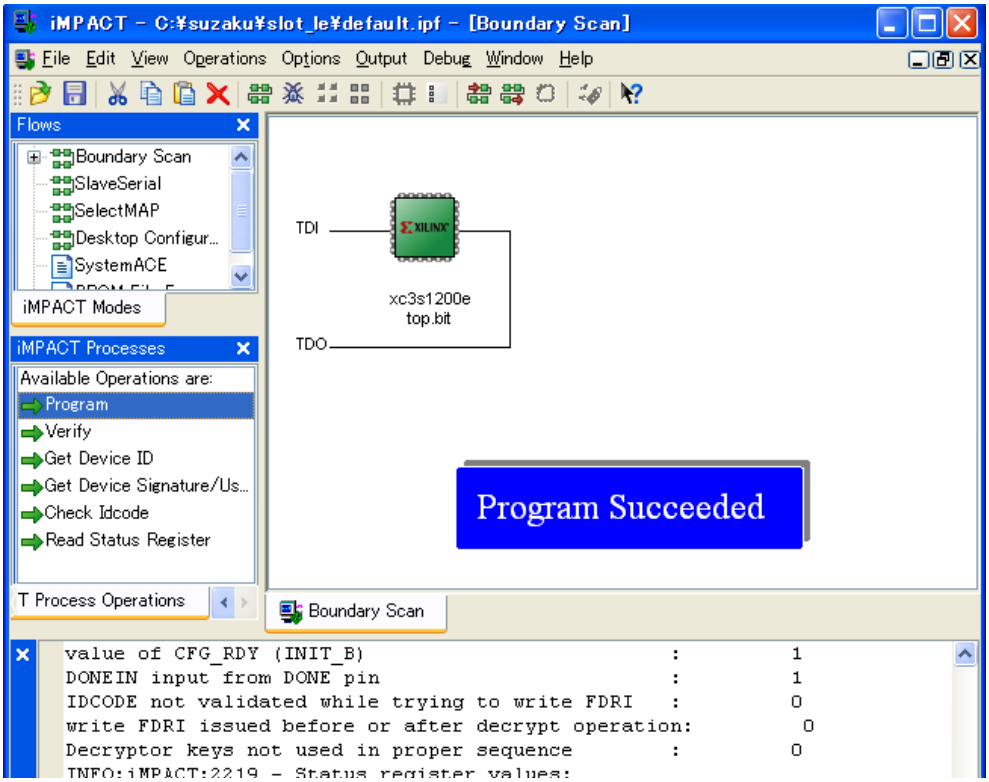


図 8-29 コンフィギュレーション成功

単色 LED(D1)が光ったでしょうか？

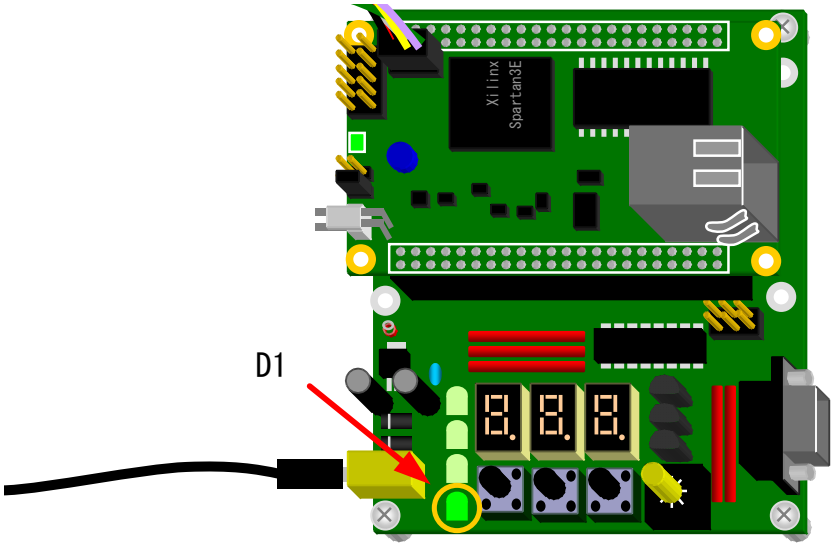


図 8-28 単色 LED(D1)点灯

8.8.2. JTAG でコンフィギュレーション手順まとめ

1. SUZAKU JP2 にジャンパプラグをさしてショートさせる
2. SUZAKU CON7 に JTAG ダウンロードケーブルを接続する
3. LED/SW CON6 に AC アダプタ 5V を接続し、電源投入
4. SUZAKU D3 のパワーON LED(緑)が点灯していることを確認
5. Project Navigator の Configure Device(iMPACT)をダブルクリックして iMPACT を立ち上げ、コンフィギュレーション
6. 動作確認

※電源を切ると、コンフィギュレーション内容は失われます。

8.8.3. Flash に保存してコンフィギュレーション

一回電源を切ってもう一度電源を入れてみてください。Flash に保存されているデータがコンフィギュレーションされるので、スロットマシンの状態に戻ったと思います。これは先ほども述べましたが、Xilinx の FPGA が SRAM ベースのためです。内部の回路内容を保持させるには Flash にコンフィギュレーションデータを書き込む必要があります。

まず、SUZAKU JP2 にジャンパプラグをさし、ショートさせてください。JP2 をショートさせると、電源投入時 FPGA に対し、Flash からのコンフィギュレーションを停止させることができます。コンフィギュレーションを停止させないと書き込み不良等を起こしてしまいます。

LED/SW CON4 に JTAG のダウンロードケーブル（Xilinx Parallel CableⅢまたは Ⅳ）を接続し、LED/SW CON6 に AC アダプタ5V を接続し、電源を投入してください。SUZAKU D3 のパワーON LED(緑)が点灯しているか確認してください。

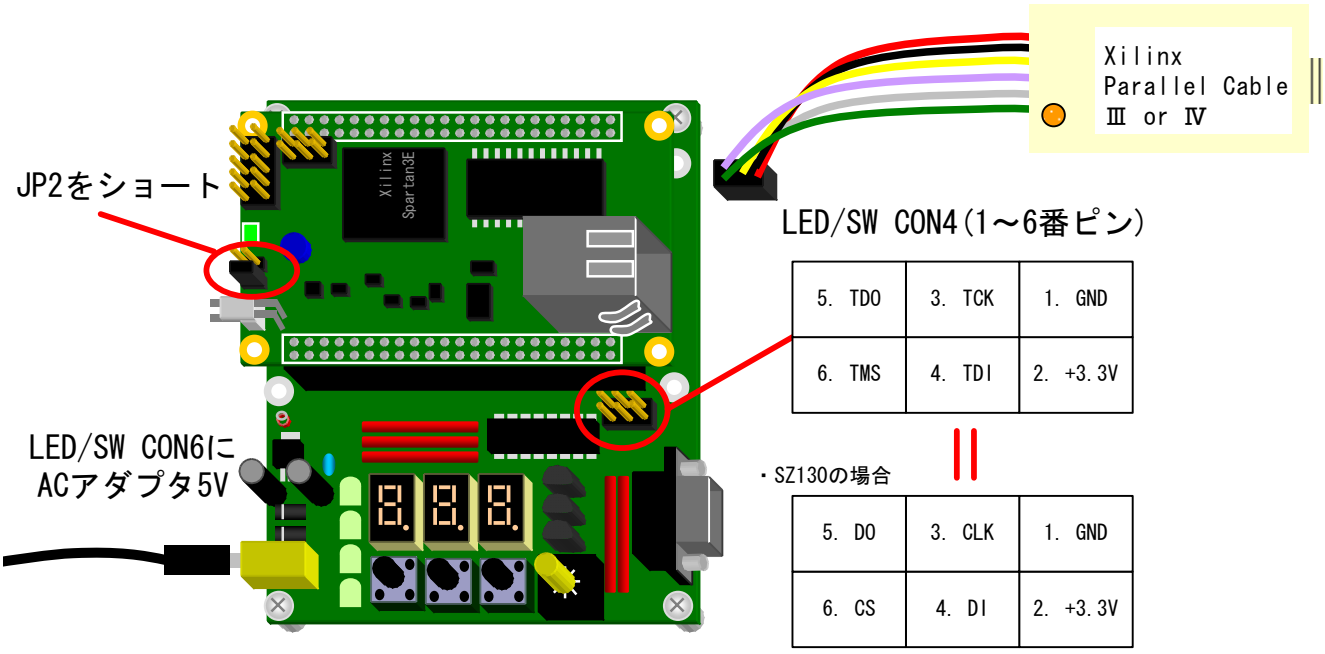


図 8-30 Flash 書き込み

● SZ010、SZ030、SZ310 の場合

SZ010、SZ030、SZ310 の場合 FPGA コンフィグレーション IC に TE7720(メーカ：東京エレクトロンデバイス)を実装しています。

TE7720 は、JTAG(CON2)から送られてくるデータを Flash にプログラムし、再起動時に Flash からデータを読み込み、FPGA をコンフィグレーションする IC です。TE7720 については東京エレクトロンデバイスのホームページから詳細資料をダウンロードできます。

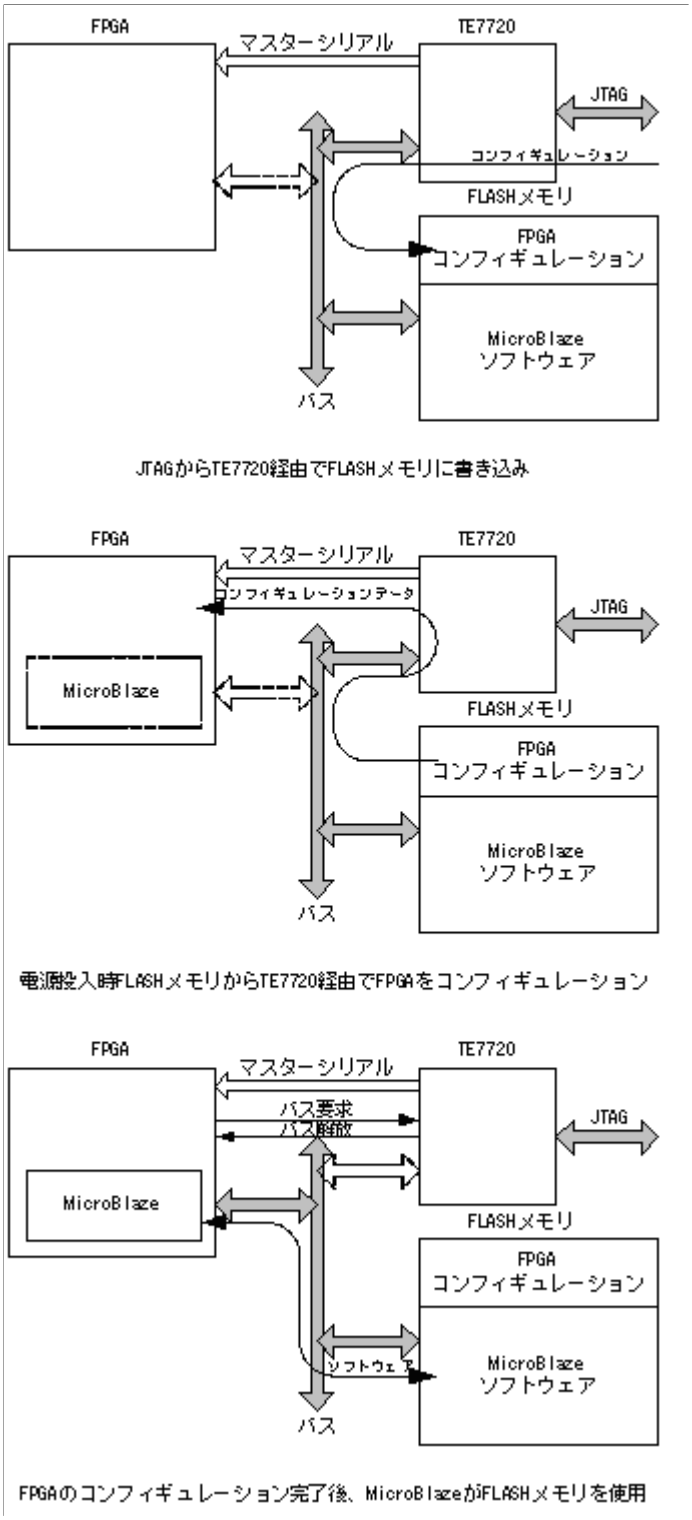


図 8-31 TE7720 コンフィギュレーション

まず、bit ファイルを mcs ファイルに変換します。
iMPACT を立ち上げ、[Prepare a PROM File]を選択し、[Next]をクリックして下さい。

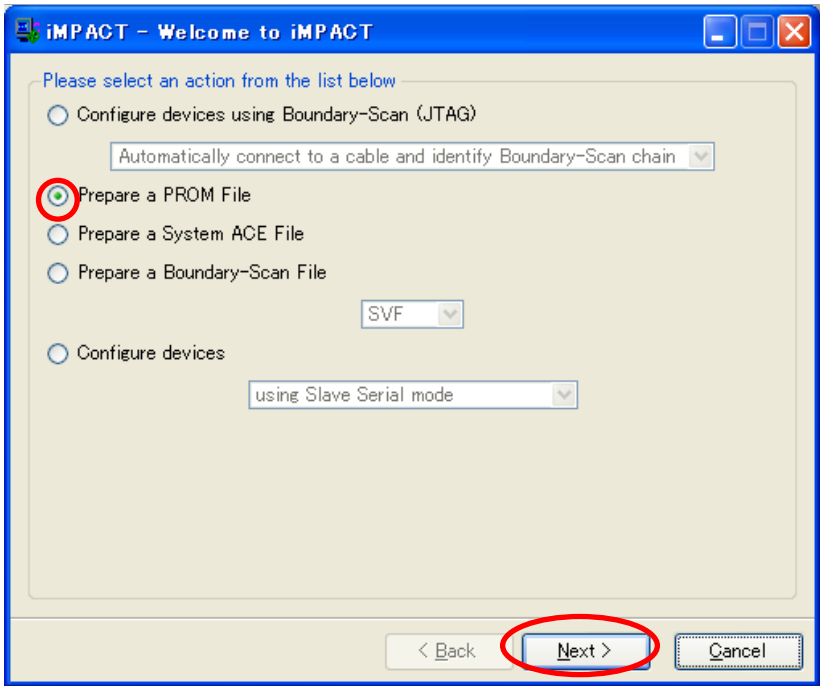


図 8-32 TE7720 iMPACT 立ち上げ

[Xilinx PROM]、[MCS]をチェックし、[PROM File Name]に top と入力し、[Location]に mcs ファイルの保存先を設定し、[Next]をクリックして下さい。

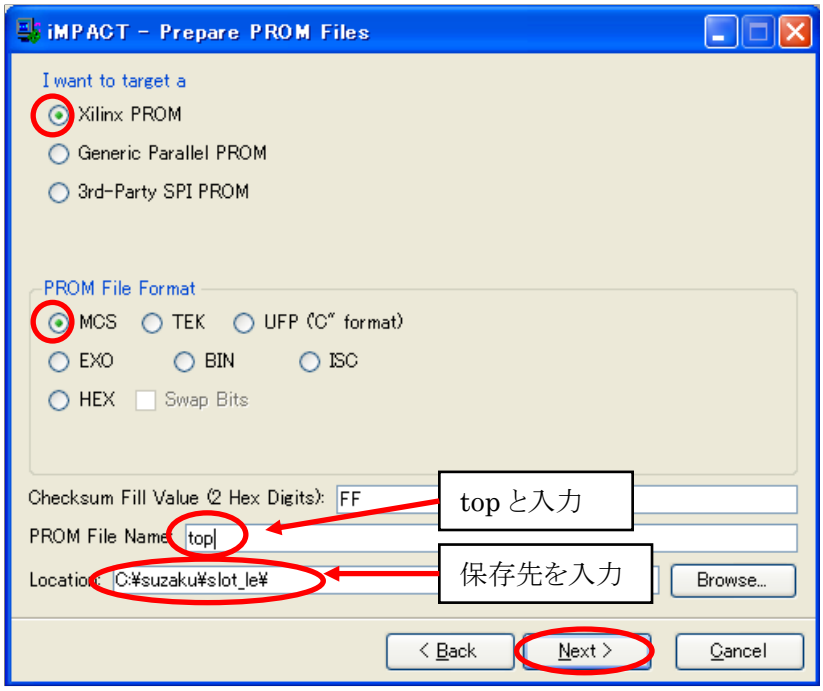


図 8-33 TE7720 iMPACT 立ち上げ

[Select a PROM]で vc18v→xc18v04 を選択し、[Add]をクリックして下さい。

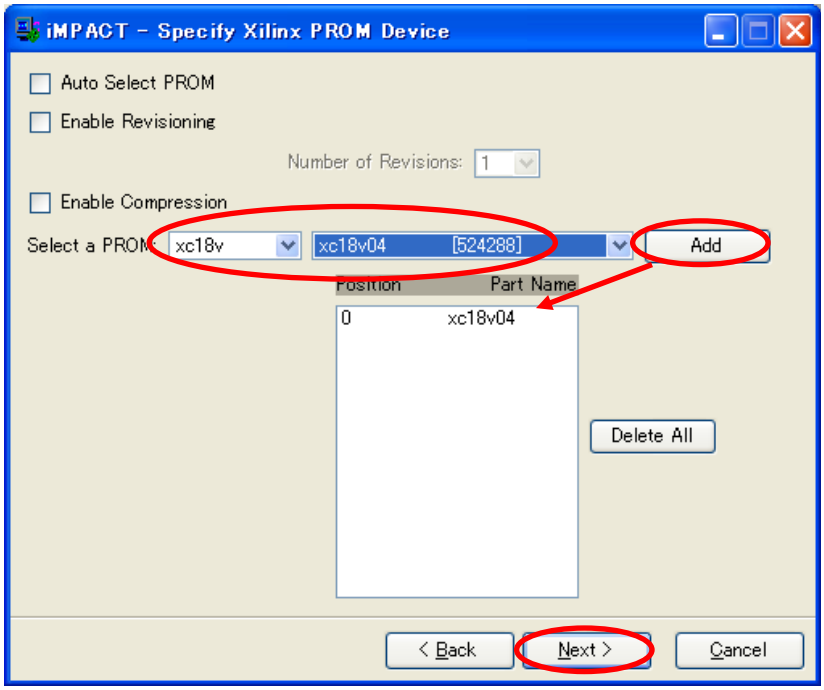


図 8-34 PROM の選択

確認画面が表示されます。間違いがなければ[Finish]をクリックして下さい。

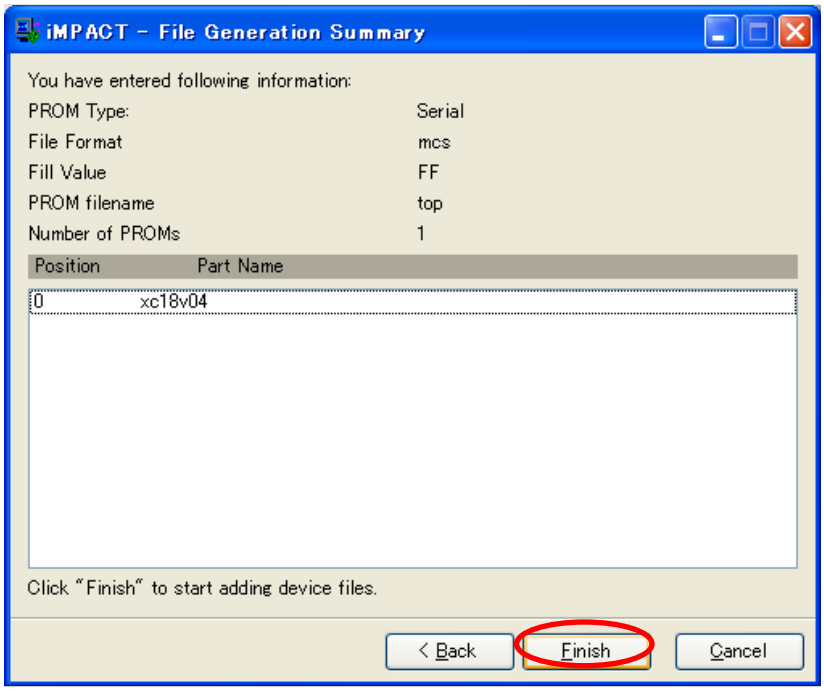


図 8-35 TE7720 確認画面

次の画面が表示されるので、[OK]をクリックして下さい。

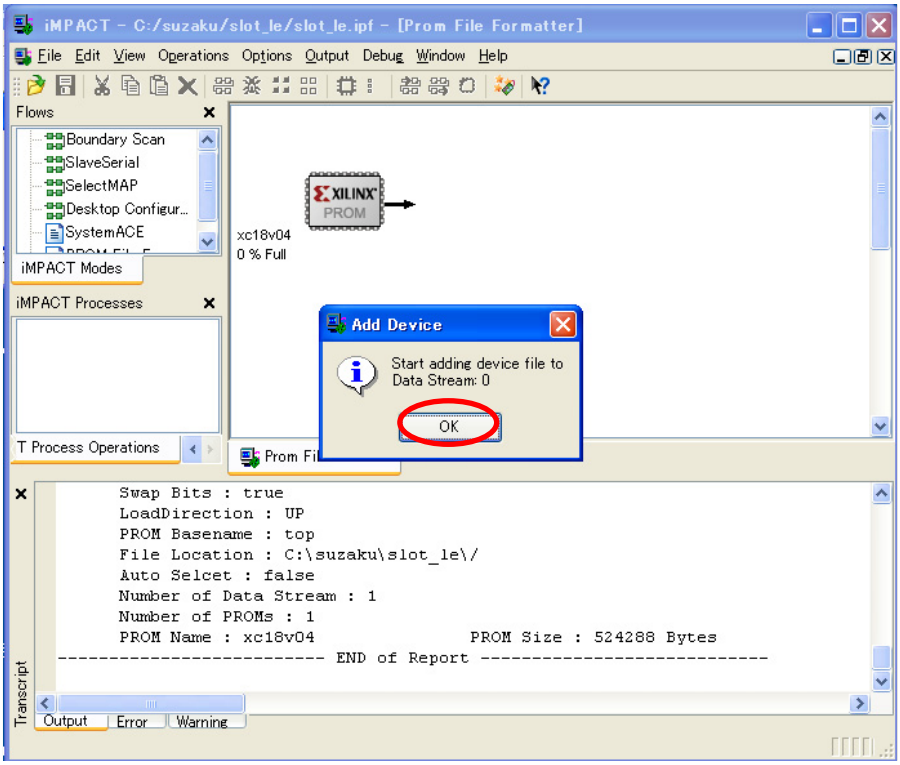


図 8-36 TE7720 デバイスファイル追加

bit ファイルを選択し、[開く]をクリックして下さい。

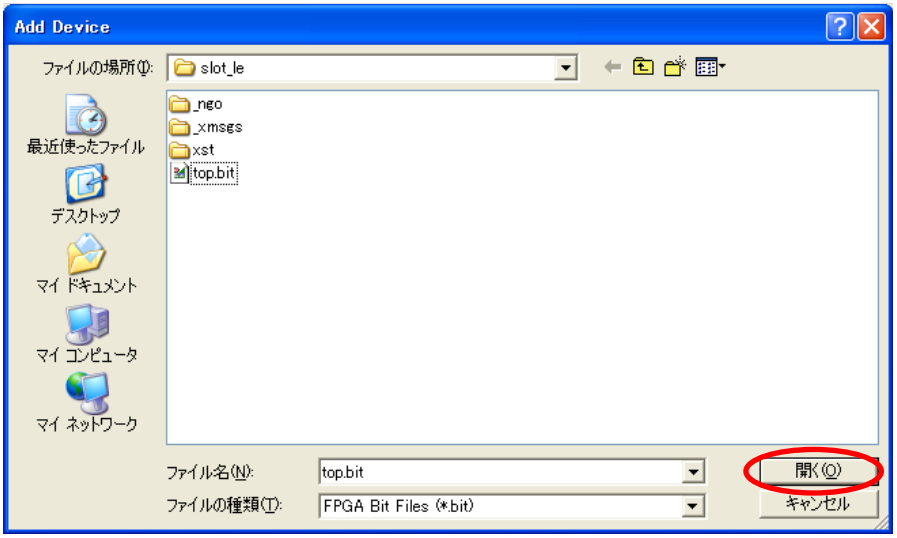


図 8-37 TE7720 bit ファイルを開く

他のデバイスを追加するか聞かれるので、[No]をクリックして下さい。

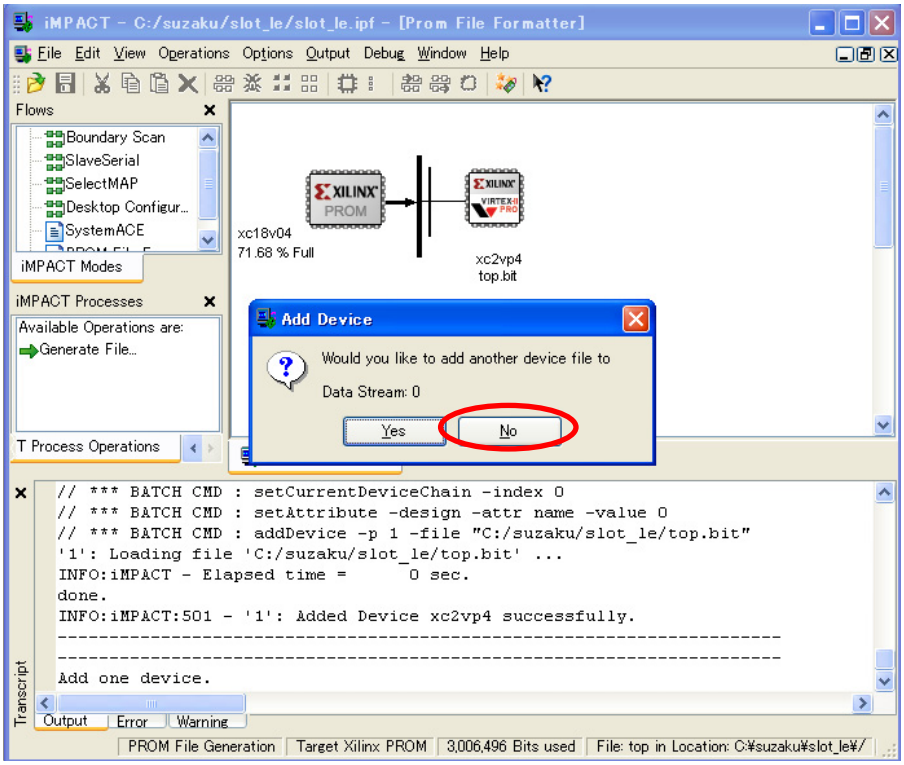


図 8-38 TE7720 デバイスファイルさらに追加

次の画面が表示されるので[OK]をクリックして下さい。

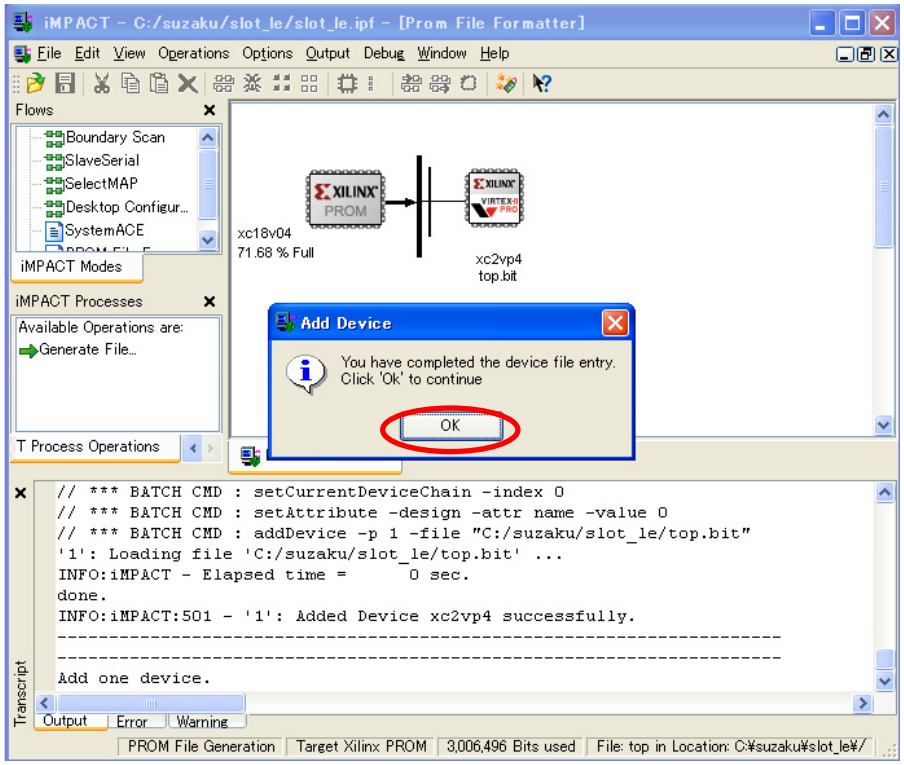


図 8-39 TE7720 準備完了

[Generate File...]をダブルクリックして下さい。PROM File Generation Succeeded と表示されたら、mcs ファイル作成完了です。

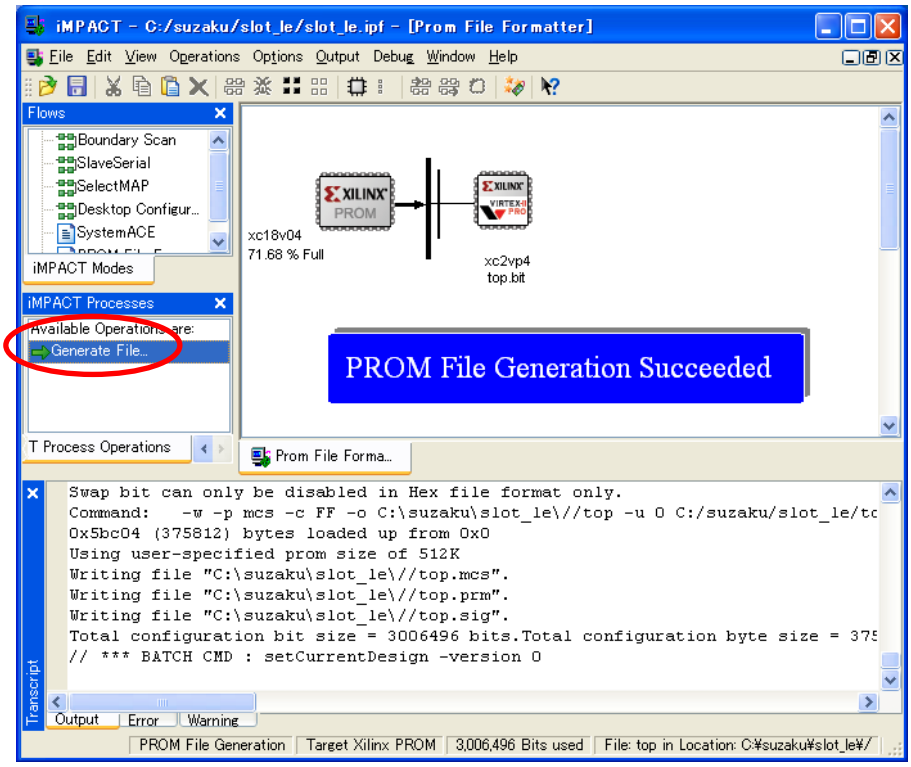


図 8-40 mcs ファイル出来上がり

付属 CD-ROM の"¥suzku¥tools"の中の圧縮ファイル"LBPlay2_Release108.zip"をハードウェアに展開してください。展開後のフォルダの中に"Lbplay2.lzh"が入っているので、これをさらに展開してください。このフォルダの中に TE7720 にデータを転送するためのソフトウェア LBPlay2 が入っています。(LBPlay2 は、東京エレクトロンデバイスのホームページから最新版をダウンロードすることが出来ます)

先ほど作った mcs ファイルを、展開したフォルダの中にに"device.def"と"lbplay2.exe"の 2 つのファイルがあることを確認して、コピーしてください。

コマンドプロンプトを開き、mcsファイル等があるフォルダに移動してください。
lbplay2 –deb top.mcs
と実行してください。

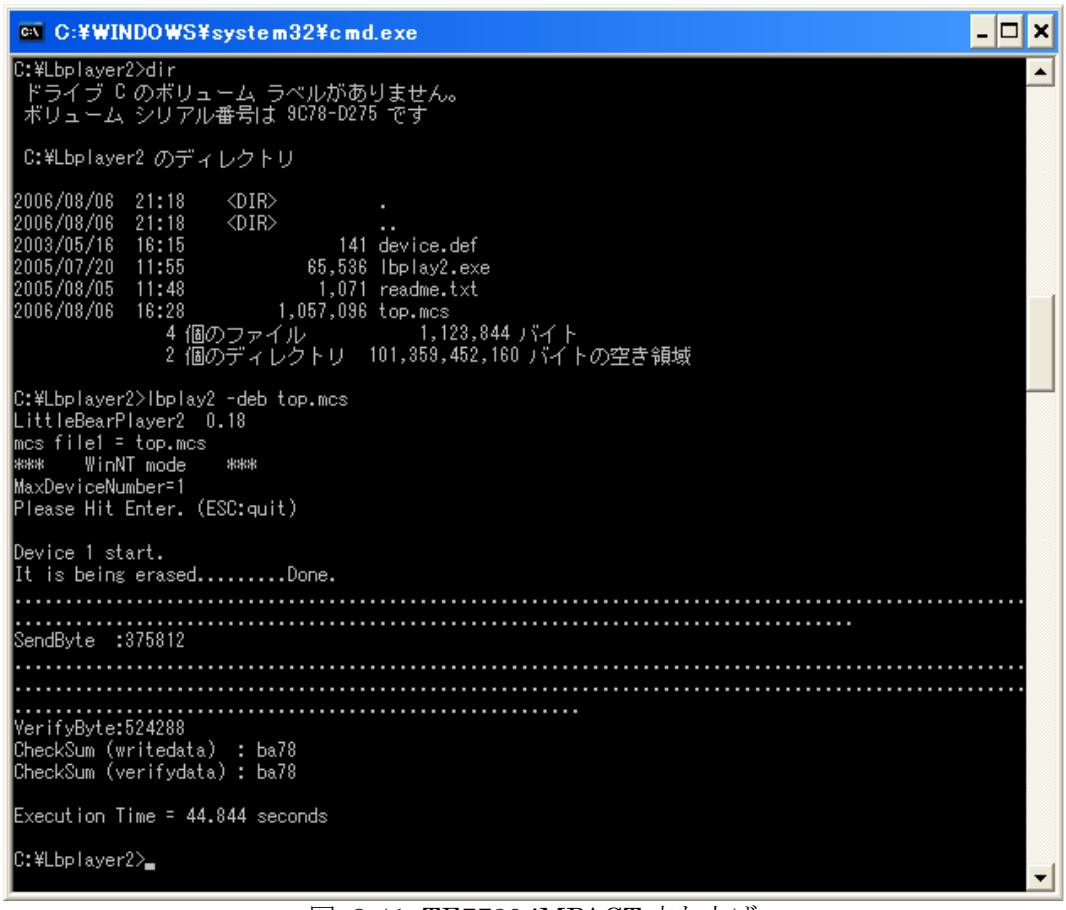


図 8-41 TE7720 iMPACT 立ち上げ

エラーが出なければ、プログラム完了です。
何らかの原因でエラーを起こした場合は、SUZAKU を動作させず、再プログラミングを行ってください。

LED/SW CON6 から AC アダプタ 5V を抜いて電源を切り、JP2 のジャンパプラグと LED/SW CON4 のダウンロードケーブルをはずしてください。再び LED/SW CON6 に AC アダプタ 5V を接続し、電源を再投入してください。

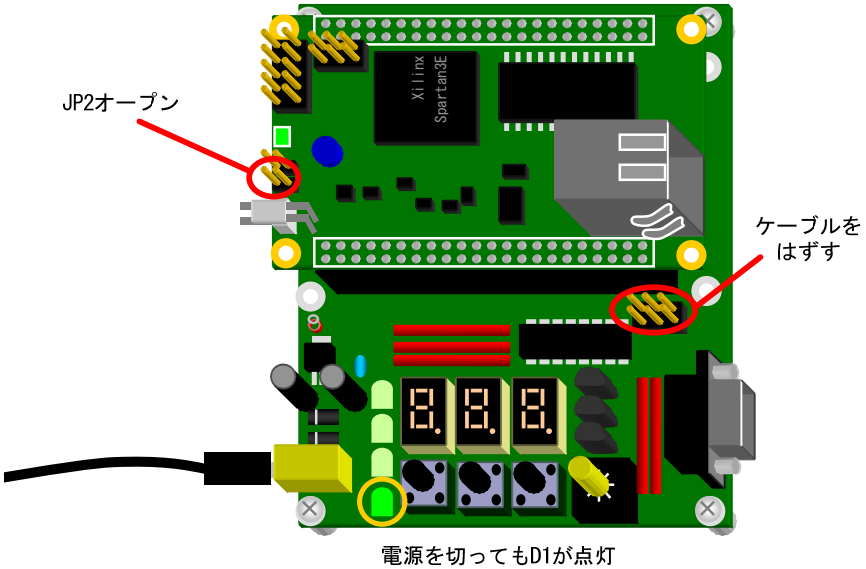


図 8-42 Flash 書き込み成功

単色 LED(D1)が光ったでしょうか？今回は電源を切ってもコンフィギュレーションデータは失われません。

● ERROR: Please check %windir%¥system32¥drivers¥windrvr6.sys.が発生した場合

付属 CD-ROM の"¥suzku¥tools"の中の圧縮ファイル"LBPlay2_Release108.zip"をハードウェアに展開してください。展開後のフォルダの中に"Release205.zip"が入っているので、これをさらに展開してください。

展開後のフォルダの中にある"windrvr6.inf"、"windrvr6.sys を同じ名前のファイルがないことを確認し、Administrator 権限ユーザで以下のフォルダにコピーしてください。

- WindowsNT/2000 の場合　　C:¥WINNT¥system32¥drivers
- WindowsXP の場合　　C:¥WINDOWS¥system32¥drivers

コマンドプロンプトを立ち上げ、wdreg.exe のあるフォルダに移動し、

wdreg -inf [Windows インストールディレクトリ]¥system32¥drivers¥windrvr6.inf install

を実行してください。

install: completed successfully

と表示されます。これでドライバがインストールされ、エラーが出なくなります。

■ SZ130 の場合

SPI Flash にプログラムします。SPI Flash は SZ130 の裏面に実装されています。

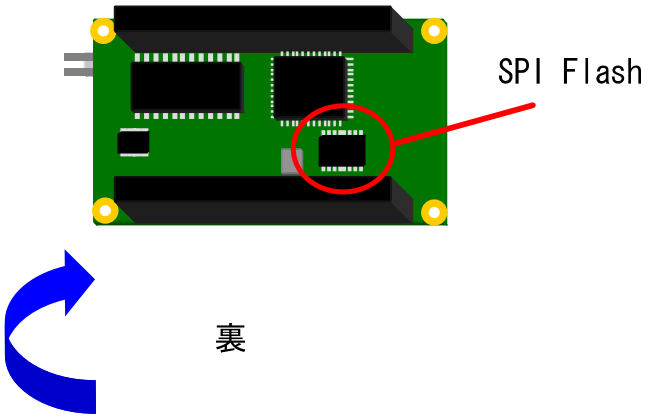


図 8-43 SPI Flash の所在

付属 CD-ROM の"¥suzaku¥tools"の中の圧縮ファイル"spi_writer.zip"をハードディスクに展開してください。展開後のフォルダの中の"Spi_Writer.exe"を実行してください。下図が立ち上がります。[...]をクリックしてください。

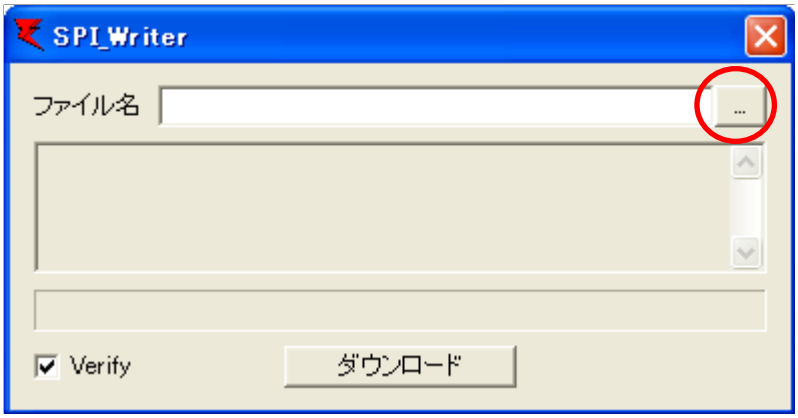


図 8-44 SPI_Writer

ファイル選択画面が立ち上がるので、書き込みたい bit ファイルを選択し、[開く]をクリックしてください。

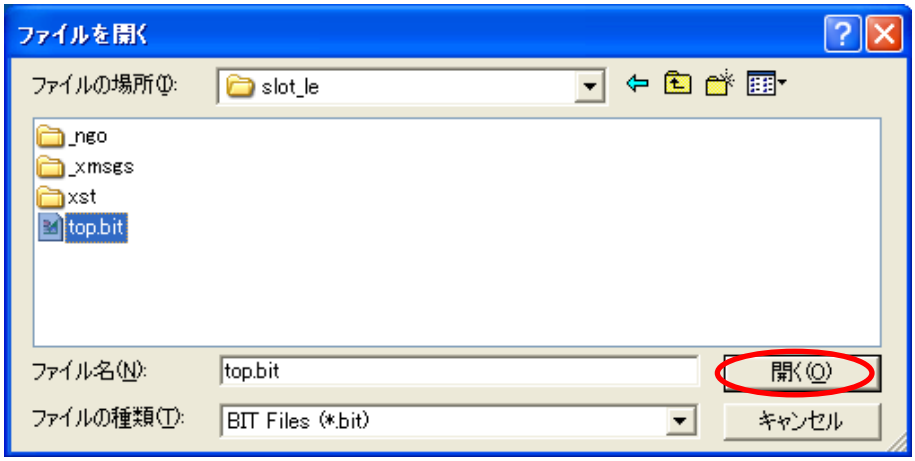


図 8-45 bit ファイル選択

※SPI Writer は書き込みたい bit ファイルをドラッグ&ドロップで選択することもできます。

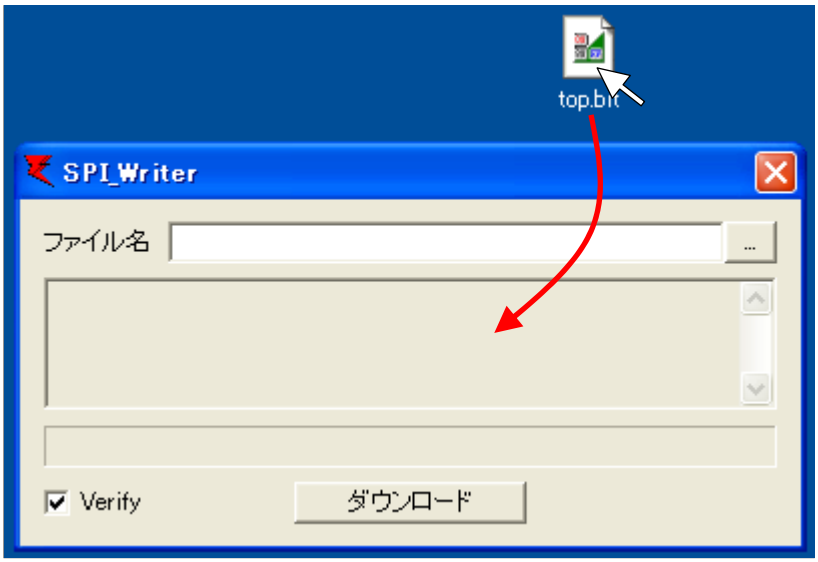


図 8-46 ドラッグ&ドロップ

これで書き込み準備完了です。ダウンロードをクリックしてください。Verify を必要としない場合は、チェックボタンをはずしてください。

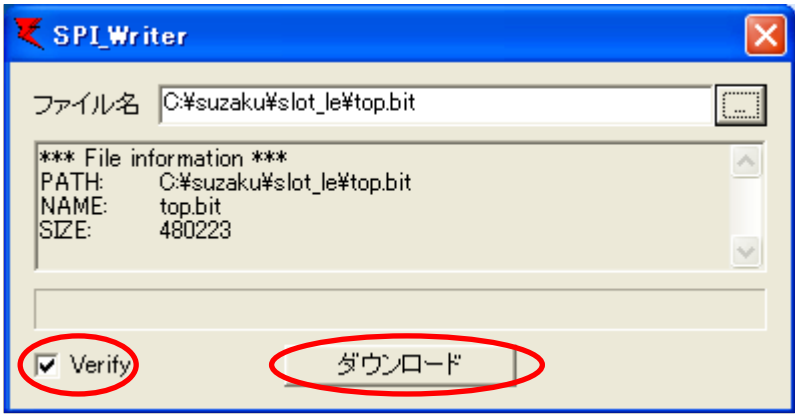


図 8-47 書き込み準備完了

書き込みを開始してもいいか確認画面が表示されるので[OK]をクリックしてください。

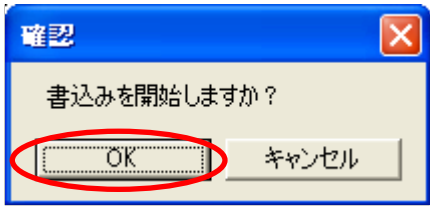


図 8-48 書き込み確認画面

コンフィギュレーションデータが SPI Flash に書き込まれます。ここで"Please check windrvr.sys"というエラーが発生した場合は"8.8.5 Please check windrvr.sys が発生した場合"を参照してください。

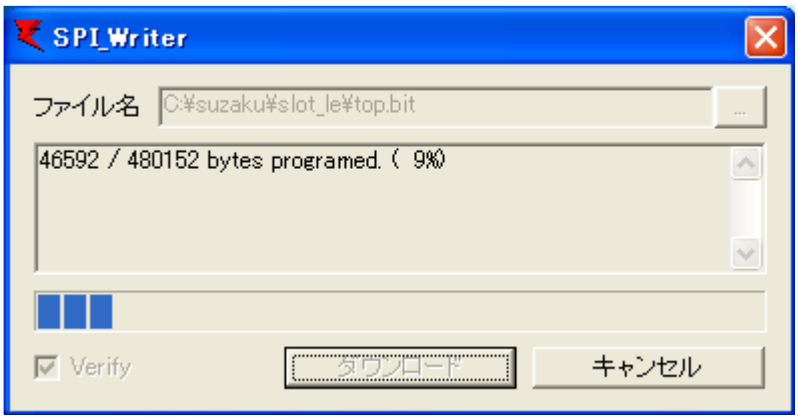


図 8-49 書き込み中

以下の画面のように"Download has been completed!"と表示されたら書き込み終了です。

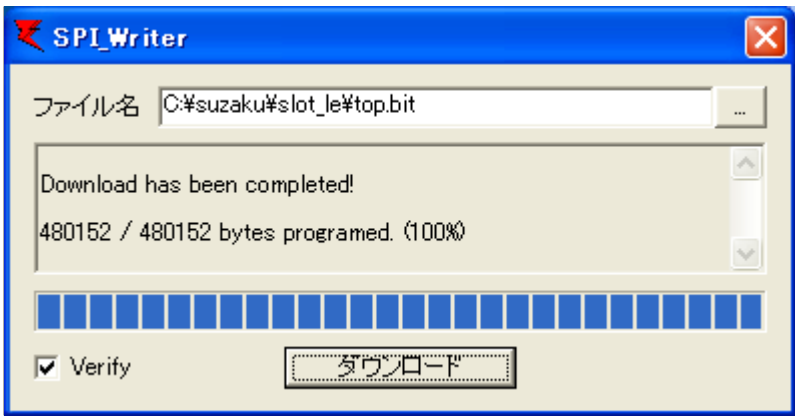


図 8-50 書き込み終了

LED/SW CON6 から AC アダプタ 5V を抜いて電源を切り、JP2 のジャンパプラグと LED/SW CON4 のダウンロードケーブルをはずしてください。再び LED/SW CON6 に AC アダプタ 5V を接続し、電源を再投入してください。

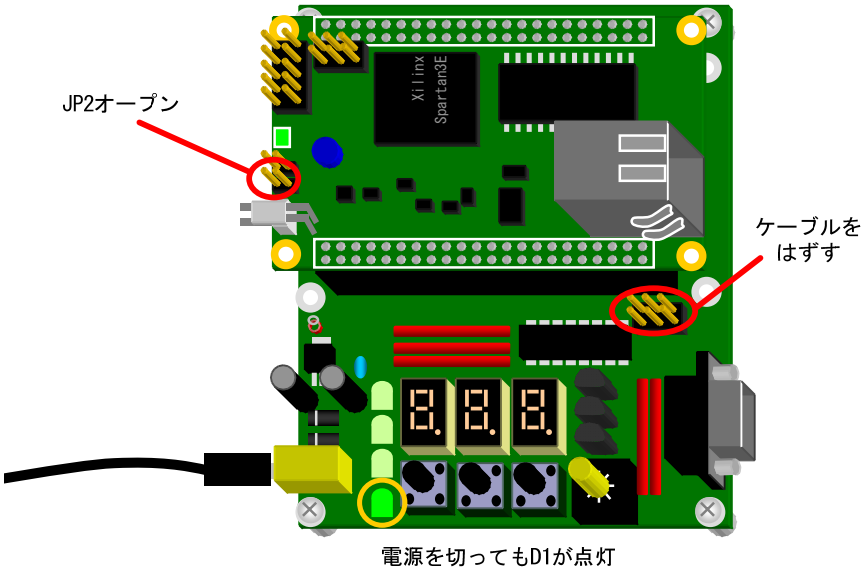


図 8-51 SPI Flash 書き込み成功

単色 LED(D1)が光ったでしょうか？今回は電源を切ってもコンフィギュレーションデータは失われません。

■ Please check windrvr.sys が発生した場合

ISE をインストールした PC では書き込みが始まらず、以下のエラーが出ることがあります。

図 8-52 エラー表示

付属 CD-ROM の"¥suzaku¥tools"の中の圧縮ファイル"spi_writer.zip"をハードディスクに展開してください。展開後のフォルダの中にある WINDRVR.SYS を同じ名前のファイルがないことを確認し、Administrator 権限ユーザで以下のフォルダにコピーしてください。

- WindowsNT/2000 の場合　　C:¥WINNT¥system32¥drivers
- WindowsXP の場合　　C:¥WINDOWS¥system32¥drivers

コマンドプロンプトを立ち上げ、wdreg.exe のあるフォルダに移動し、

wdreg install

を実行してください。

Creating driver entry...OK
Starting driver entry...OK

と表示されます。これでドライバがインストールされ、エラーが出なくなります。

■ SPI Writer について

SPI Writer は SPI Flash の先頭から 1Mbyte まで消去し、コンフィギュレーションデータを書き込む SUZAKU の SPI Flash 専用の書き込みツールです。

SUZAKU は SPI Flash にソフトウェアのデータやその他データを保存しており、これらのデータを壊さないために専用ツールで書き込みます。

SPI Flash の書き込みツールとしては Xilinx 提供の xspi というツールもありますが、xspi は SPI Flash のデータを全消去して、コンフィギュレーションデータを書き込むツールであるため、SUZAKU の SPI Flash 書き込み用として使うには、注意が必要となります。

8.8.4. Flash に保存してコンフィギュレーション手順まとめ

● SZ010、SZ030、SZ310 の場合

1. SUZAKU JP2 にジャンパプラグをさしてショートさせる
2. LED/SW CON4 にダウンロードケーブルを接続する
3. LED/SW CON6 に AC アダプタ 5V を接続し、電源投入
4. SUZAKU D3 のパワーON LED(緑)が点灯していることを確認
5. iMPACT を立ち上げ、mcs ファイルを作成
6. コマンドプロンプトを立ち上げ、LBPlay2 でコンフィギュレーションデータを書き込む
7. LED/SW CON6 の AC アダプタ 5V をはずし、電源を切る
8. LED/SW CON4 のダウンロードケーブルをはずす
9. SUZAKU JP2 のジャンパプラグをはずす
10. LED/SW CON6 に AC アダプタ 5V を接続し、電源再投入
11. 動作確認

※電源を切っても、コンフィギュレーション内容は失われません。

■ SZ130 の場合

1. SUZAKU JP2 にジャンパプラグをさしてショートさせる
2. LED/SW CON4 にダウンロードケーブルを接続する
3. LED/SW CON6 に AC アダプタ 5V を接続し、電源投入
4. SUZAKU D3 のパワーON LED(緑)が点灯していることを確認
5. SPI_Writer を立ち上げ、SPI Flash にコンフィギュレーションデータを書き込む
6. LED/SW CON6 の AC アダプタ 5V をはずし、電源を切る
7. LED/SW CON4 のダウンロードケーブルをはずす
8. SUZAKU JP2 のジャンパプラグをはずす
9. LED/SW CON6 に AC アダプタ 5V を接続し、電源再投入
10. 動作確認

※電源を切っても、コンフィギュレーション内容は失われません。

8.9. 空きピン処理

D1 を点灯させたとき、D2、D3、D4 が少し光っているのに気がついたでしょうか？
これは空きピンの処理の仕方によります。
Generate Programming File を右クリックしてメニューを出し、Properties を選択してください。

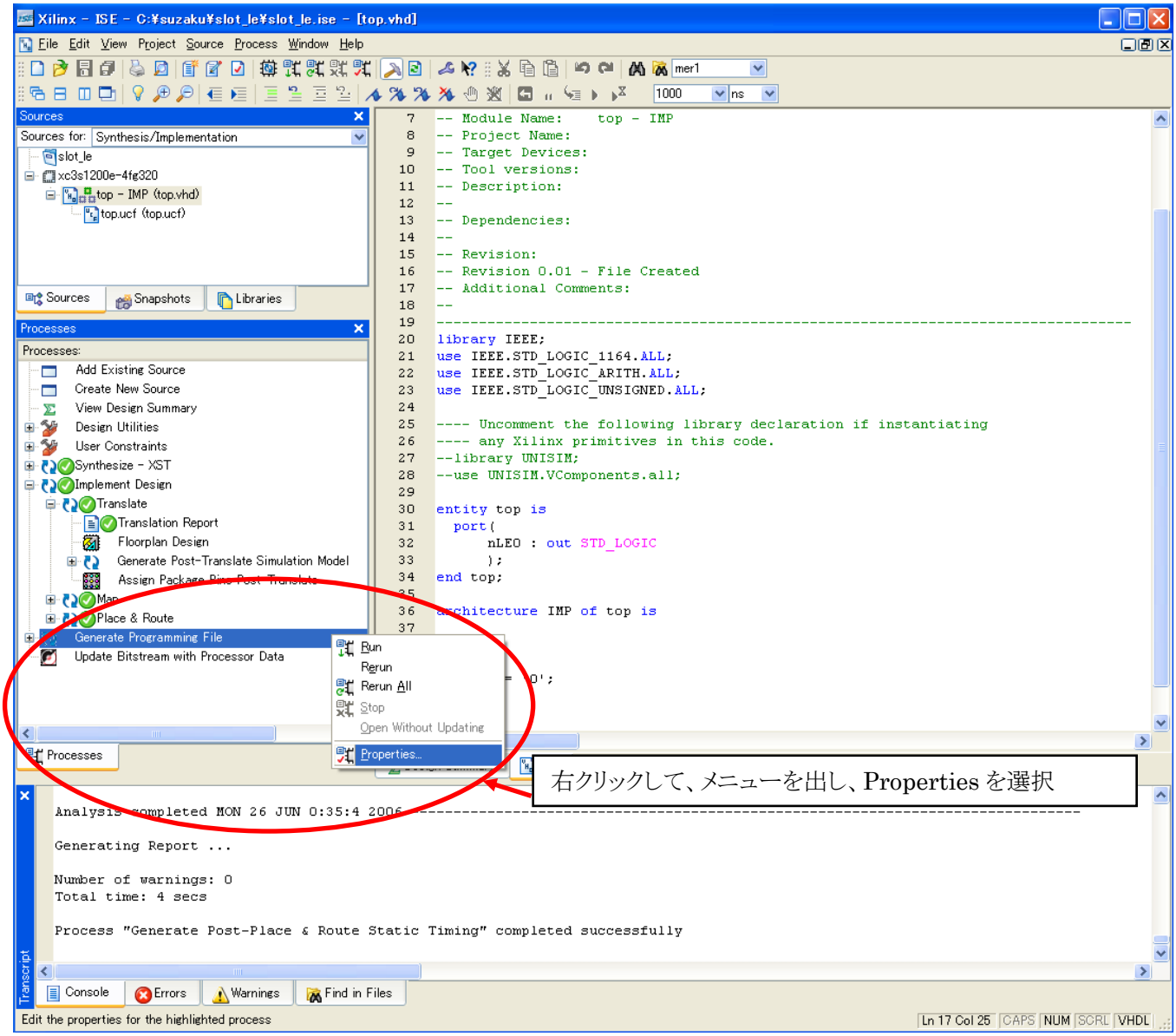


図 8-53 空きピン処理の設定画面の出し方

[Configuration Options]を選ぶと次の画面が出てきます。ここで終端処理を決めることができます。この中に[Unused IOB Pins]というのがありますが、これが空きピン処理の設定になります。初期設定で、[Pull Down]になっています。これを[Pull Up]にすると、D2、D3、D4 は光らなくなりますが、SUZAKU では空きピンを初期設定の Pull Down のまま使ってください。Pull Up や Float に変更した場合、動作しなくなる可能性があります。少し光るのが嫌な場合は空きピンの終端処理の設定を変更するのではなく、ピンに何か信号を割り当ててください。

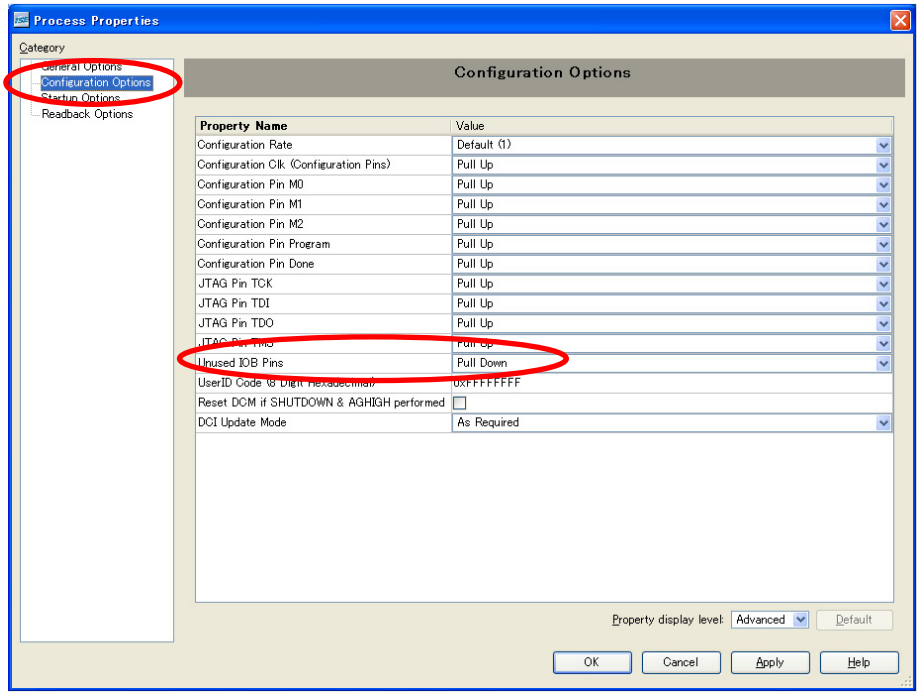


図 8-54 空きピン処理設定

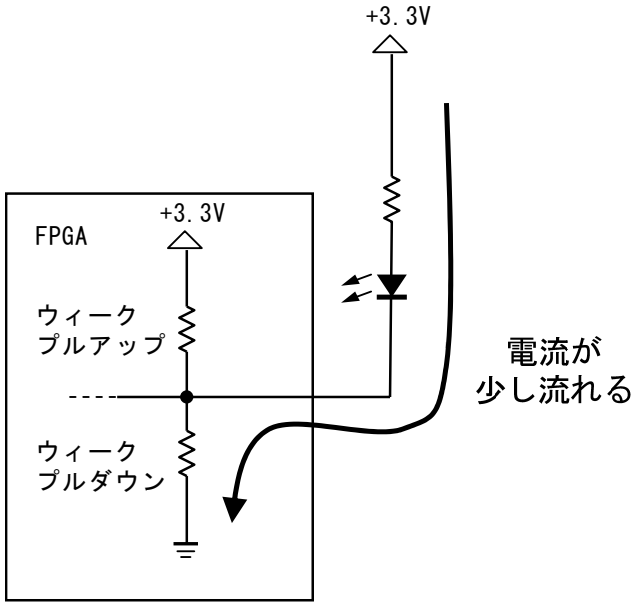


図 8-55 少し光る理由

9.VHDL によるロジック設計

ここでは VHDL の記述方法とロジック設計について説明します。

9.1.VHDL の基本構造

まずは VHDL の記述方法を説明します。
VHDL の基本構造は

- ライブラリ宣言とパッケージ呼び出し
- エンティティ(entity)
- アーキテクチャ(architecture)

からなります。
ライブラリ宣言とパッケージ呼び出しで、各種演算子や関数などを定義したパッケージを呼び出し、エンティティに外部とのインターフェースを記述し、アーキテクチャに内部回路の構造や動作を記述します。
VHDL では予約語も含めて文字の大小の区別をしません。例えば **Port** は **port** とかいても **PORT** とかいても同じに扱われます。何が予約語にあたるのかは各自調べてみてください。もし、ISE 付属のテキストエディタを使っている場合は、青やピンクなど色が変わって表示された場合予約語となります。
--で始めるとその行末までがコメントになります。

例 9-1　VHDL 基本構造

```
--ライブラリ宣言とパッケージ呼び出し
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--エンティティ(入出力の宣言)
entity slot is
    Port (
--ここに入出力ピンの宣言を書く
        );
end slot;

--アーキテクチャ(回路本体)
architecture IMP of slot is

--内部信号等の各種宣言を記述する

begin
    --ここに回路を記述する
end IMP;
```

9.2. ライブラリ宣言とパッケージ呼び出し

ISE で VHDL ソースコードを自動生成すると、パッケージが3つ呼び出されます。それぞれの用途は下表の通りです。他にも様々なパッケージがあるので、必要に応じて呼び出してください。

表 9-1 ライブラリとパッケージ

ライブラリ	パッケージ	用途
IEEE	std_logic_1164	基本関数
	std_logic_arith	算術演算
	std_logic_unsigned	符号なし演算

9.3. エンティティ(entity)

エンティティ内ではポートの宣言を行います。外部とのインターフェースについて定義する部分がエンティティになります。ISE で VHDL ソースコードを自動生成すると、エンティティ名はファイル名と同じ名前になります。

例 9-2 entity 記述

```
entity slot is --entity エンティティ名 is
    Port (
    SYS_CLK : in std_logic;
nLE : out std_logic_vector(0 to 2);
nSW : in  std_logic_vector(0 to 2) --最後に ; は不要
       );
end slot; --end エンティティ名;
```

● 信号の定義

信号は以下の形式で宣言します。

例 9-3 信号の定義

```
ポート信号名:入出力方向 データタイプ名;
```

● 入出力方向

入出力方向には in、out、inout 等を記述します。

表 9-2 入出力方向

in	入力であることを指定
out	出力であることを指定(内部で再利用できない)
inout	入出力であることを指定

VHDL では、出力ポート信号を内部に参照できません。内部で参照したいときは、内部参照用に内部信号を用います。内部信号の宣言については"9.4 アーキテクチャ(architecture)"の内部信号の定義の項を参照してください。

● データタイプ

データタイプには色々ありますが、よく使うのは std_logic と std_logic_vector です。std_logic で1ビットの信号を定義し、std_logic_vector(0 to n) で n＋1 ビット幅の信号を定義します。

nLE : out std_logic_vector(0 to 2) とすると 3 ビットの幅を持った出力信号を定義することができます。

nLE(0)、nLE(1)、nLE(2)とすることで、それぞれのビットを切り出すことができます。

to を使って定義すると、MSB 側が 0 ビット目になります。(downto とすると LSB 側が 0 ビット目になります。本書では IBM の CoreConnect にあわせてバスを定義するため、to を使います。)

表 9-3 データタイプ

std_logic	IEEE ライブラリで定義
std_logic_vector	std_logic のベクタ・タイプ
integer	整数型(32 ビット)

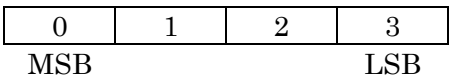


図 9-1 to を使って定義

9.4. アーキテクチャ(architecture)

回路の構造や動作などをここに記述をします。アーキテクチャ名は任意ですが、本書では IMP としています。

例 9-4 architecture 記述

```
architecture IMP of slot is --architecture アーキテクチャ名 of エンティティ名 is
--内部信号の定義
signal count : STD_LOGIC_VECTOR(0 to C_CNT_WIDTH-1);
   signal count_led : STD_LOGIC;
begin
--ここに同時処理文を記述する
   nLE <= not le; --信号代入文
   process(SYS_CLK) --プロセス文
   begin
   if SYS_CLK'event and SYS_CLK = '1' then
   if SYS_RST = '1' then
          count <= (others => '0');
   else
          count <= count + 1;
   end if;
   end if;
   end process;
end IMP; --end アーキテクチャ名;
```

同時に処理される

● 内部信号の定義

内部で使用する信号は architecture と begin の間に記述します。信号の宣言には signal を用い、以下の形式で宣言します。データタイプ名はエンティティの信号宣言と同じですので、”9.3. エンティティ(entity)”のデータタイプの項を参照してください。

例 9-5 内部信号定義

```
signal 信号名 : データタイプ名;
```

● 同時処理文

begin～end の間に直接記述された回路を同時処理文といいます。同時処理文ではそれぞれが他の同時処理文と関係なく動作し、並列に処理されます。信号代入文、プロセス文などの回路を記述します。

● 信号代入文

A<=B;とすると、A に B が代入されます。

● プロセス文

プロセス文は以下の形で記述します。

例 9-6　プロセス文

```
process(センシティビティリスト)
  begin
   .
   .
   .
end process;
```

センシティビティリストに記述した値のどれかが変化すると、中に記述した文が上から実行されていきます。最終行まで実行すると上に戻り、次にこれらの信号が変化するまで動作を停止します。

9.5. 組み合わせ回路(not、and、or)

ここからは少しロジック設計について説明します。

”not”、”and”、”or”などの基本論理ゲートを組み合わせて作られるものを組み合わせ回路といい、クロックを必要とせずに現在の入力だけで出力が決まります。押しボタンスイッチと単色 LED を使って基本論理ゲートの動作を確認します。

9.5.1. 押しボタンスイッチ周辺回路

押しボタンスイッチ周辺回路は、下図のようになっており、ボタンを押していないと”High”が FPGA に入力され、ボタンを押していると”Low”が FPGA に入力されます。

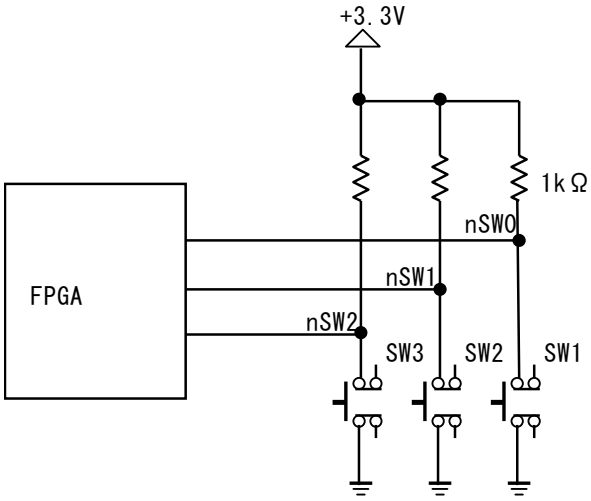


図 9-2　押しボタンスイッチ周辺回路

9.5.2. not、and、or を使う

信号には正論理、負論理の 2 種類の表現があります。例えば、単色 LED を LE という信号名で定義し、”High”(”1”)で点灯した場合は正論理、nLE という信号名で定義し、”Low”(“0”)で点灯した場合は、負論理となります。(nLE の n は負論理だということを明言するために使います)

LED/SW ボード正論理、負論理の信号が混在しているので、分かりやすくするために負論理の信号は FPGA 内部で反転させて正論理として扱うようにしています。

● not

負論理から正論理(負論理から正論理)は次の一文で記述できます。

例 9-7 not 記述

```
nLE0 <= not le0;
```


le0	nLE0
0	1
1	0

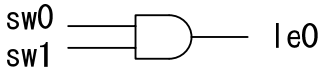
図 9-3 not 回路と真理値表

● and

SW1(信号名:sw0)と SW2(信号名:sw1)を両方押したら D1(信号名:le0)が点灯するというのは以下の一文で記述できます。

例 9-8 and 記述

```
le0 <= sw0 and sw1;
```



sw0	sw1	le0
0	0	0
0	1	0
1	0	0
1	1	1

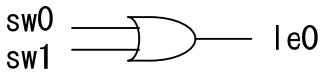
図 9-4 and 回路と真理値表

● or

SW1(信号名:sw0)か SW2(信号名:sw1)のどちらか一方でも押されたら D1(信号名:le0)が点灯するというのは以下の一文で記述できます。

例 9-9 or 記述

```
le0 <= sw0 or sw1;
```



sw0	sw1	le0
0	0	0
0	1	1
1	0	1
1	1	1

図 9-5　or 回路と真理値表

■ top.vhd

先ほど単色 LED(D1)を光らせたプロジェクトを変更して試してみてください。

例 9-10　not、and、or(top.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--エンティティ(入出力の宣言)
entity top is
    Port (
nLE0 : out STD_LOGIC; --単色 LED(D1)への出力信号(負論理)
nSW0 : in STD_LOGIC; --スイッチ(SW1)からの入力信号(負論理)
nSW1 : in STD_LOGIC --スイッチ(SW2)からの入力信号(負論理)
       );
end top;

--アーキテクチャ(回路本体)
architecture IMP of top is

    signal le0 : STD_LOGIC; --単色 LED 内部信号(正論理)
    signal sw0 : STD_LOGIC; --スイッチ(SW1)内部信号(正論理)
    signal sw1 : STD_LOGIC; --スイッチ(SW2)内部信号(正論理)

begin

    sw0 <= not nSW0; --not 回路で入力前に正論理にする
    sw1 <= not nSW1; --not 回路で入力前に正論理にする

le0 <= sw0 and sw1; -- and 回路(両方押したら LED が光る)
--le0 <= sw0 or sw1; -- or 回路(どちらか一方でも押したら LED が光る)

nLE0 <= not le0; --not 回路で出力前に負論理にする

end IMP;
```

■ ピンアサイン
ピンアサインは以下になります。

表 9-4 not、and、or ピンアサイン

	SZ010 SZ030	SZ130	SZ310
nLE0	B12	E12	L16
nSW0	A13	F11	K14
nSW1	B14	C11	K15

9.6. 順序回路

その時点の入力だけでなく、過去の入力信号にも依存する回路を順序回路といいます。値を保持する、そのまま出力する、といったことができます。

順序回路は基本的に同期設計により成り立ちます。非同期設計は現在の状況に応じて物事が動き、次に何が起こるか分からなくなるので、順序回路には向きません。もし非同期信号を使いたい場合は、通常 1 回クロックに同期させてから使います。

ISE Simulator を使って、最も基本の順序回路であるカウンタの動きを確認します。

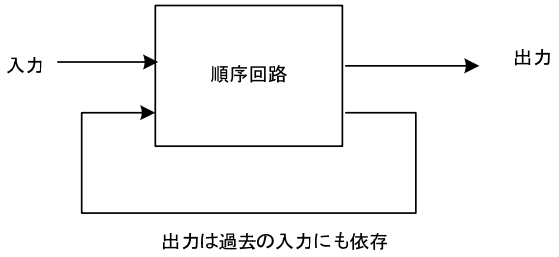


図 9-6 順序回路の概念図

9.6.1. D-FF(D 型フリップフロップ)

順序回路で重要なのは D-FF(Delay FF)です。

クロックの立ち上がりでデータを保持し、次のクロックで保持したデータを出力します。クロックの立ち上がり以外でデータが変化しても出力は変化しません。

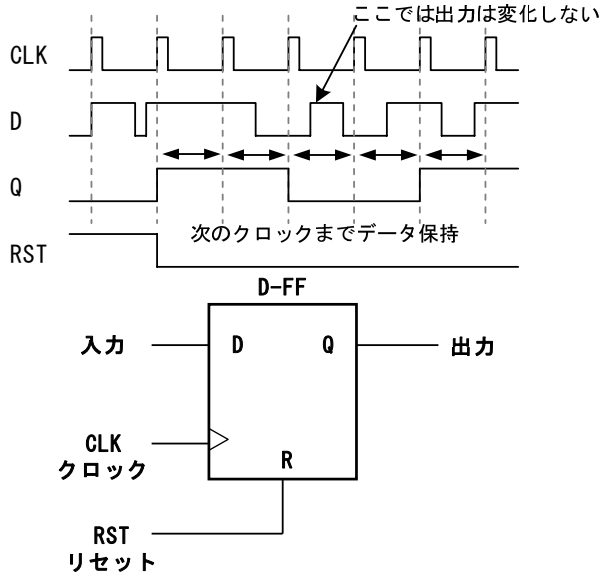


図 9-7 D-FF の動作

9.6.2. 同期設計

回路は入力信号の時間差によって動作が決まります。非同期設計では ns 単位で時間差を作ってしまうことがあり、タイミング設計が非常に困難です。論理合成、配置配線による信号の遅延はツールの種類やバージョンに依存します。また、温度やデバイスの個体差によっても信号が遅延します。これらのすべての遅延を非同期設計で押さえ込むのは至難の業です。押さえ込むのに失敗すると、タイミング不良を起こし、次に何が起こるのか分からなくなってしまいます。

それにひきかえ同期設計はタイミング設計が簡単になります。同期設計では、同期用クロックの周期時間よりもゲートや配線による遅延やセットアップなどの積算時間ほうが短ければ、回路が設計通りに動作することが保障されています。FPGA は内部にクロック専用線を複数もっていて、これらのクロック専用線は他の線に比べて Delay が少なく、信号が速く到達することが保障されています。

よって、一般的に FPGA ではこのクロック専用線を用い、同期設計を行います。

同期回路は組み合わせ回路と D-FF とで成り立っています。組み合わせ回路の規模を小さくすることで、遅延は少なくなり速い回路を作ることができます。どこに D-FF を入れ、組み合わせ回路の規模どう小さくするかで、全体の最高動作周波数が決まってきます。

9.6.3. カウンタ

カウンタとはクロックにあわせて数値をインクリメント(デクリメント)していく回路です。カウンタの回路は以下のように記述できます。

例 9-11 カウンタ記述

```
process(SYS_CLK) --クロック信号に変化があると実行
begin
    if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
    if SYS_RST = '1' then --リセットされたら(同期リセット)
            count <= (others => '0'); --カウンタ初期化
    else --その他は
            count <= count + 1; --カウント値をインクリメント
    end if;
    end if;
end process;
```

● クロックの記述

クロックの立ち上がりエッジに同期させたい場合以下のように記述します。 SYS_CLK = '0'とすると立下りエッジに同期させることができます。

例 9-12 クロックの立ち上がりエッジに同期

```
if SYS_CLK'event and SYS_CLK='1' then
```

● リセットの記述

クロックの記述の下にリセットを記述すると同期リセット、上に記述すると非同期リセットになります。
SUZAKU には電源監視 IC が実装されており、電源投入時にリセットがかかるようになっています。このリセット信号を用いて、信号の初期化を行います。VHDL ではこの様に外部からのリセットで初期化する方法の他に内部信号定義の時に初期化する方法もあります。

例 9-13 同期リセット

```
if SYS_CLK'event and SYS_CLK = '1' then
    if SYS_RST = '1' then
```

● if 文
if 文は以下の形式で記述します。

例 9-14 if 文

```
if 条件 then
順次処理文
elsif 条件 then
順次処理文
else
順次処理文
end if;
```

● 初期値
others は残りすべてという意味で、others=>'0'とすると、残っているビットすべてに 0 が代入されます。

例 9-15 other で初期化

```
count <= (others => '0');
```

9.7. ISE Simulator の使い方

今回は、4 ビットカウンタのシミュレーションを行います。4 ビットカウンタでは 0 から 15 まで数えることができます。

9.7.1. カウンタ VHDL

プロジェクトを新規作成してください。
プロジェクト名は slot_counter とし、new Source で slot_counter.vhd とし、新規ソースコードを作ってください。

■ slot_counter.vhd
カウンタ回路を記述してください。記述できたら Synthesize をダブルクリックして文法に間違いがないかチェックしてください。

例 9-16 カウンタのシミュレーション（slot_counter.vhd）

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity slot_counter is
    generic (
    C_CNT_WIDTH : integer := 4 --カウンタのビット幅
    );
    Port (
    SYS_CLK : in STD_LOGIC; --クロック信号
    SYS_RST : in STD_LOGIC; --リセット信号
    count : out STD_LOGIC_VECTOR(0 to C_CNT_WIDTH-1)  --カウンタ値
    );
end slot_counter;

architecture IMP of slot_counter is
--内部信号の定義
    signal count_w : STD_LOGIC_VECTOR(0 to C_CNT_WIDTH-1); --カウンタ値内部用
begin
```

```
    process(SYS_CLK) --クロック信号に変化があると実行
    begin
        if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
        if SYS_RST = '1' then --リセットされたら(同期リセット)
                count_w <= (others => '0'); --カウンタ初期化
        else  --その他は
                count_w <= count_w + 1; --カウント値をインクリメント
        end if;
        end if;
    end process;

    count <= count_w; --カウンタ値を外部に出力

    end IMP;
```

● ジェネリック文

バスの幅などのパラメータを渡す時などに使います。記述形式はポート文とほぼ同じですが、情報を渡すだけなので、”in”や”out”などの方向の指定はありません。

例 9-17 generic 文

```
    generic (
        信号名 : データタイプ名 := 初期値
        );
```

9.7.2. テストベンチの新規作成

カウンタの動作をシミュレーションで確認します。

[Project]→[New Source]をクリックしてください。

[Test Bench WaveForm]を選択し、[File name]にファイル名を入力し、[Next]をクリックしてください。ここではファイル名を slot_counter_tb とします。

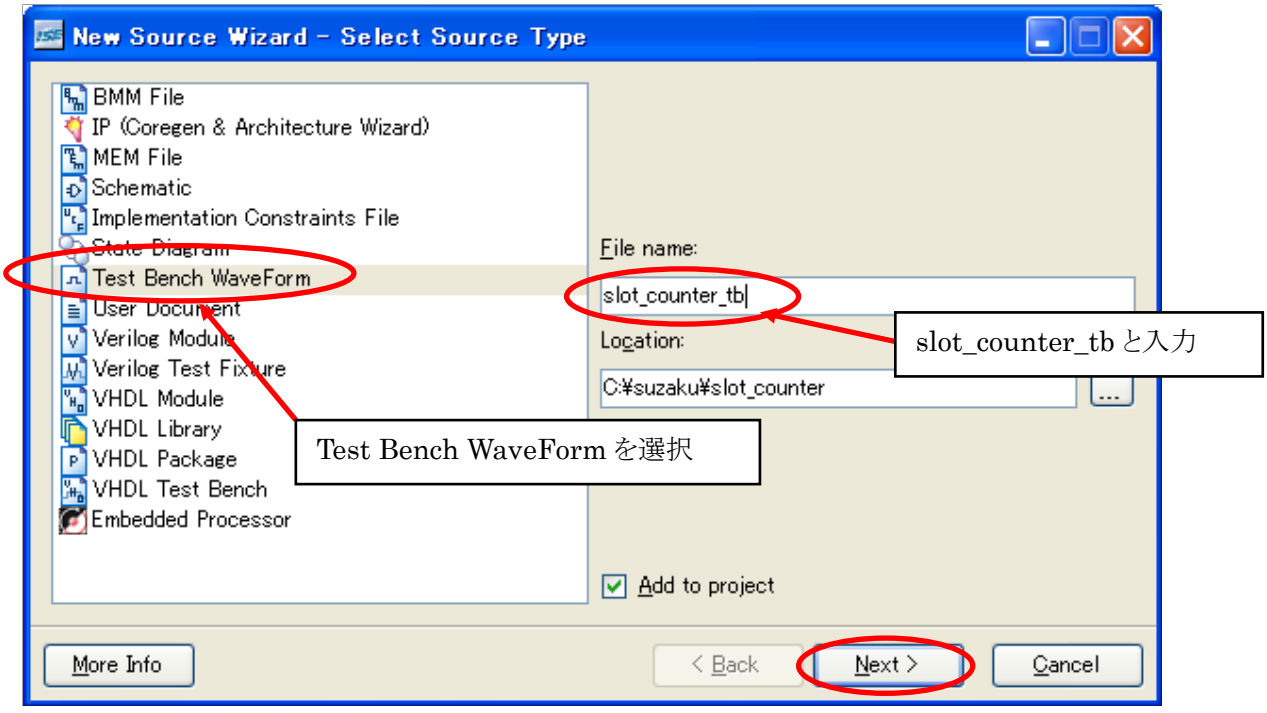


図 9-8 テストベンチ作成

次の画面が出るまで[Next]および[Finish]をクリックしてください。クロック波形を作成します。[Initial Length of Test Bench] を 10000 に変更して[Finish]をクリックしてください。[Initial Length of Test Bench]を変更すると、シミュレーション時間を変更することができます。他にも色々設定を変えることができますが、今回はカウンタの動きを見たいだけなので変更しません。

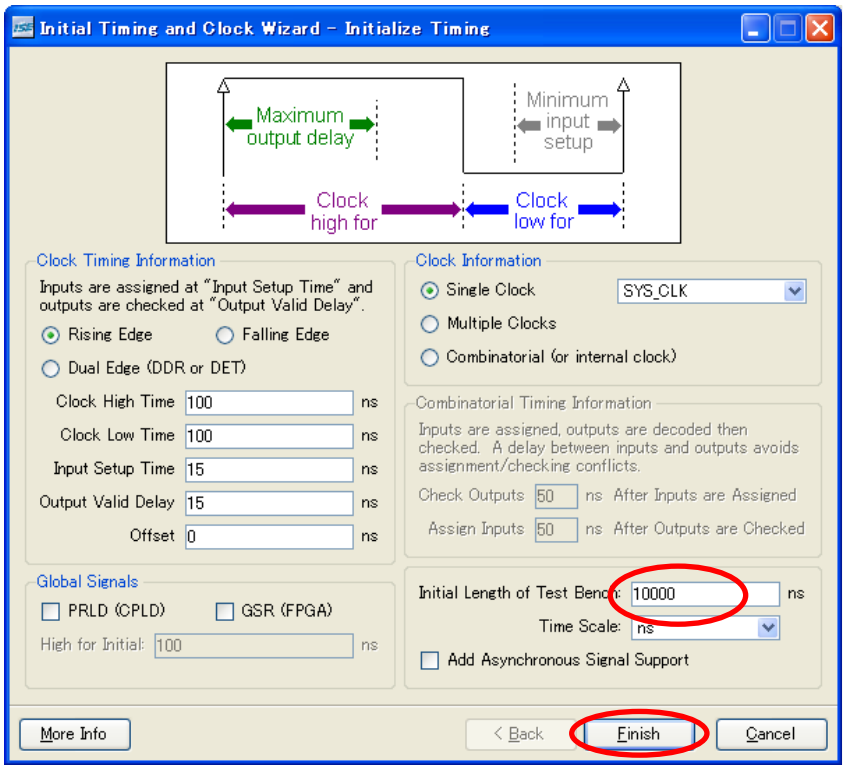


図 9-9　クロック波形作成

次の画面が表示されます。
Source ウィンドウの Source for:を[Behavioral Simulation]に変更してください。
リセット信号を入力しないと信号が初期化されないので、リセット信号を入力します。クロックが細かくて見にくいので  キーを押して適当な大きさに拡大してください。水色のセルをクリックすると信号の"High"、"Low"を切り替えることができます。図のように SYS_RST の信号を生成してください。100ns で信号を立ち上げ、500ns で信号を立ち下げています。
[File]→[Save]をクリックし保存してください。

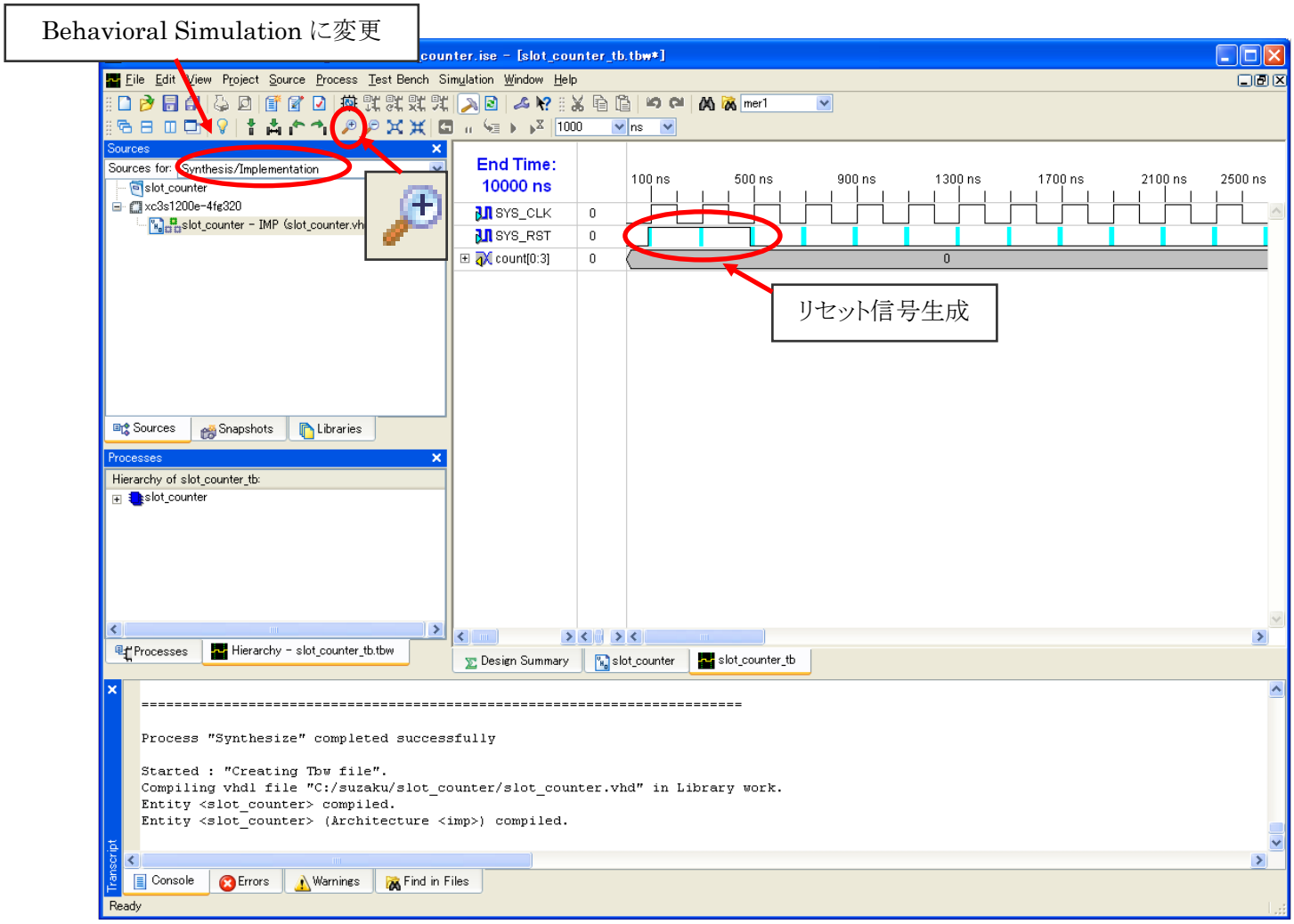


図 9-10 リセット波形生成

9.7.3. シミュレーション結果

Processes タブをクリックしてください。Xilinx ISE Simulator の⊞をクリックして開き、Simulate Behavioral Model をダブルクリックしてください。シミュレーション結果が表示されます。

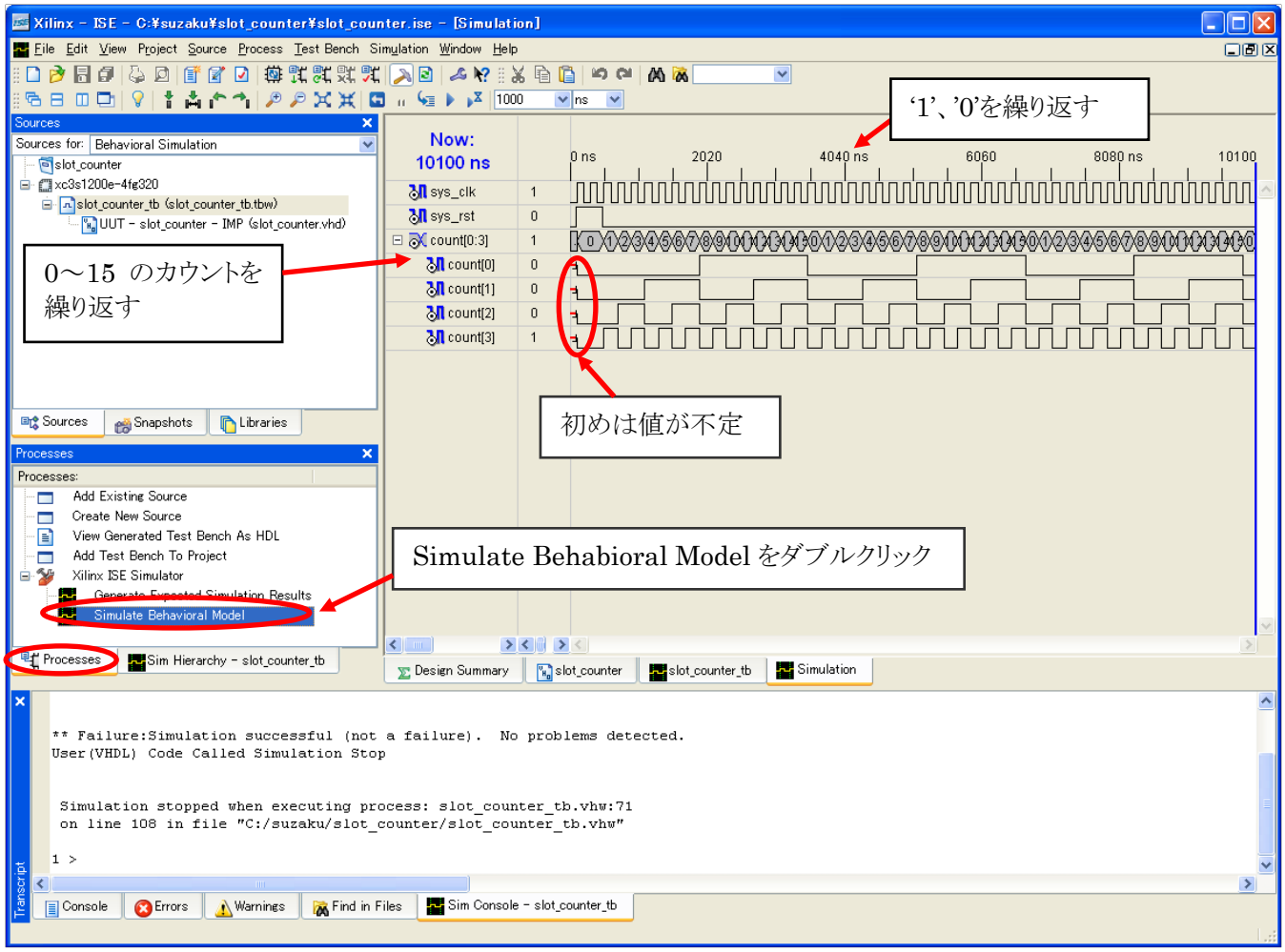


図 9-11 シミュレーション結果

● クロックについて

sys_clk の波形を見てください。クロックは”1”、”0”を繰り返します。

● リセットについて

count の波形の一番初めは赤い線で u と書かれています。これは値が不定という意味です。sys_rst が”High”になると、初期化されて値が決定します。

● カウンタの波形チェック

sys_clk の立ち上がりのタイミングごとにカウントアップしているのが分かります。

count[3]の波形の周期は sys_clk の倍、count[2]の波形の周期は count[3]の倍、count[1]の波形の周期は count[2]の倍・・・となっています。これを分周といいます。

VHDL による回路設計について、この後は必要に応じて説明していきます。

10. FPGA 入門 スロットマシン製作

“7. 4. ブートローダモードでスロットマシンを動かす”で動かしたスロットマシンは下図の構成で作られています。このスロットマシンの回路を製作していきます。

スロットマシンの機能を実現するには色々な方法が考えられるのですが、今回は SUZAKU のデフォルトに自分で作成した IP コアを接続し、ソフトウェアで制御します。

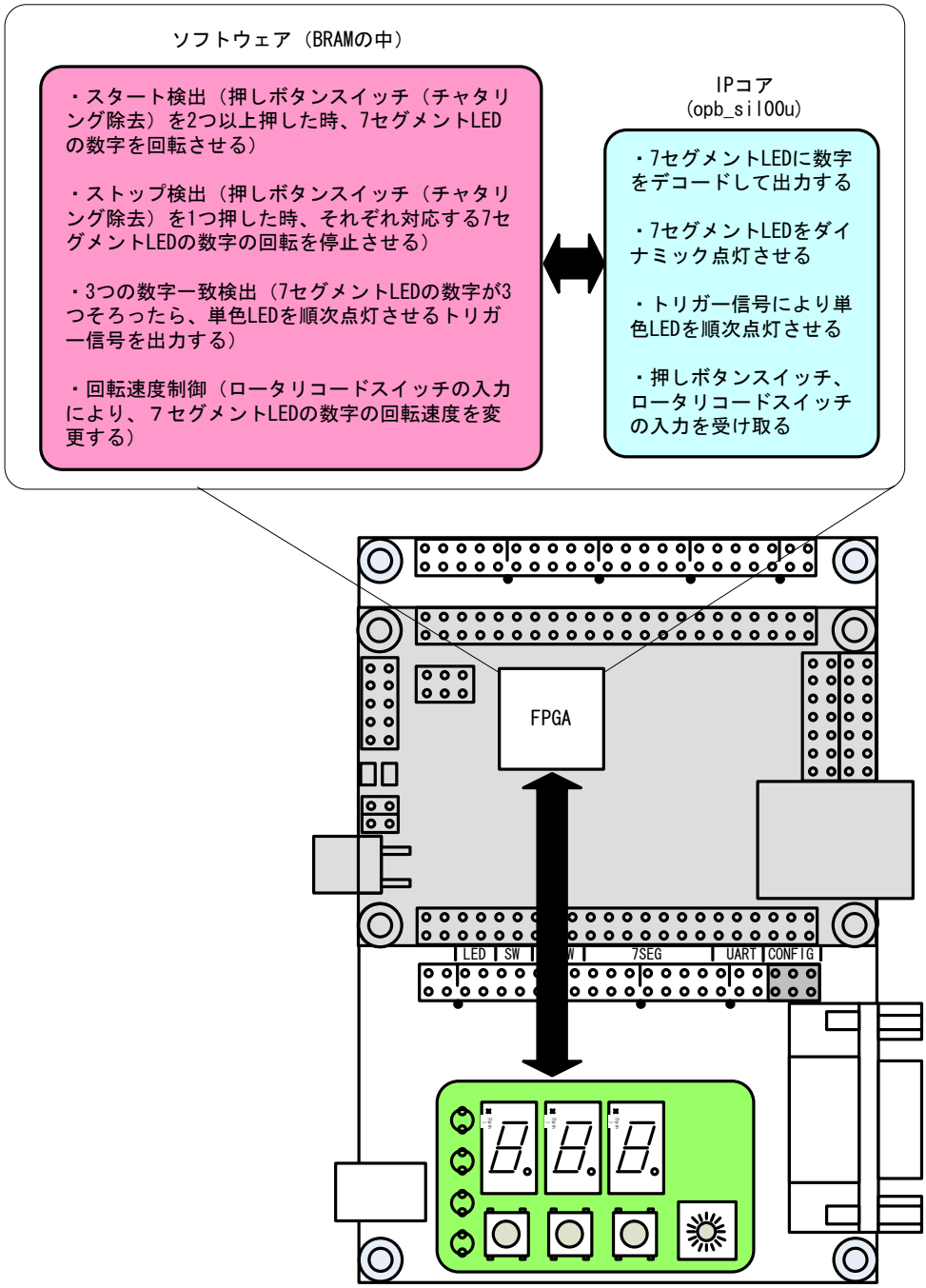


図 10-1 スロットマシンの構成

10.1. 単色 LED の順次点灯

スロットが当たった時に、当たった！という感じを出すために、単色 LED を順次点灯（D1→D2→D3→D4→D1→・・・・）させる回路を製作します。

SUZAKU のクロックは 3.6864MHz で、シリアル通信の 115.2kHz でちょうど割り切れる値になっています。このクロックをタイミング信号として単色 LED を順次点灯させると速すぎるので、目に見えるくらいの速さのタイミング信号をカウンタで作ります。カウンタは先ほどシミュレーションの時に作った回路をそのまま使いますが、今回はカウンタのビット数を 19bit とします。カウンタの最上位ビット count(0)の値は 2^{19}=524288 カウントごとに(約 7Hz)、”0”、”1”を繰り返します。

単色 LED を順次点灯させるのに、シフトレジスタを用います。シフトレジスタをシフトさせる一番簡単な条件は、タイミング信号が”0”または”1”の時、常にシフトすることです。カウンタの最上位ビットから出力される”0”、”1”はデューティー比 50:50 になっていて、このままだと、一番簡単な条件でシフトレジスタを作った場合、同じレベルの間は常にシフトし続けてしまうので使えません。このため count(0)の値が”0”から”1”になる時のエッジを検出し、1 クロックだけ”1”を出力するタイミング信号を作ります。

10.1.1. 単色 LED 周辺回路

単色 LED 周辺回路は”図 8-2 単色 LED 周辺回路”を参照してください。

10.1.2. 単色 LED 順次点灯 VHDL

プロジェクトを新規作成してください。
プロジェクト名は le_seq_blink とし、new Source で top.vhd を作ってください。
top – IMP(top.vhd)を右クリックしてメニューを出し、[New Source...]を選択してください。

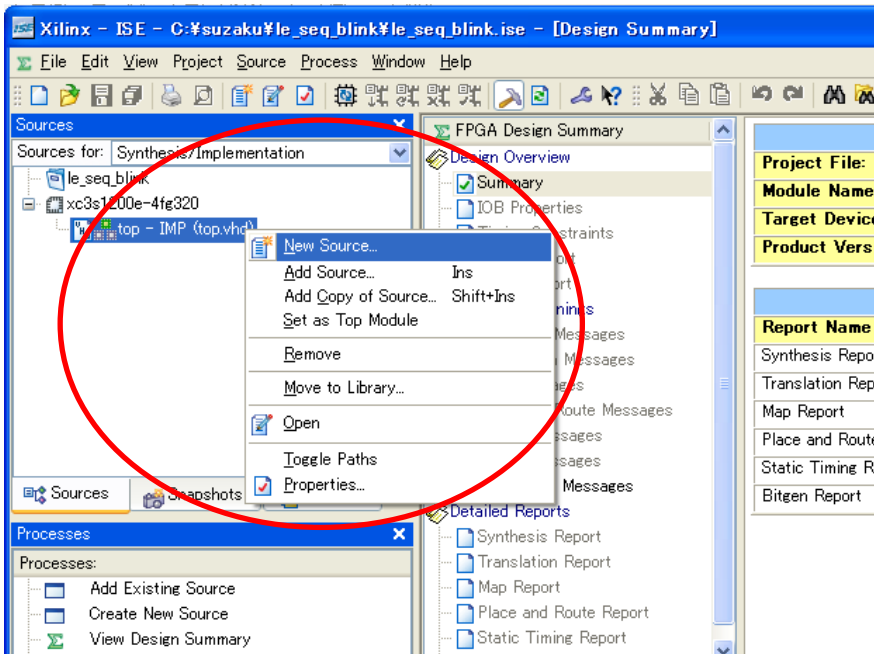


図 10-2 New Source の追加

New Source Wizard が立ち上がるので、[VHDL Module]を選択し、[File name]に le_seq_blink と入力し、新しいソースファイルを作ってください。

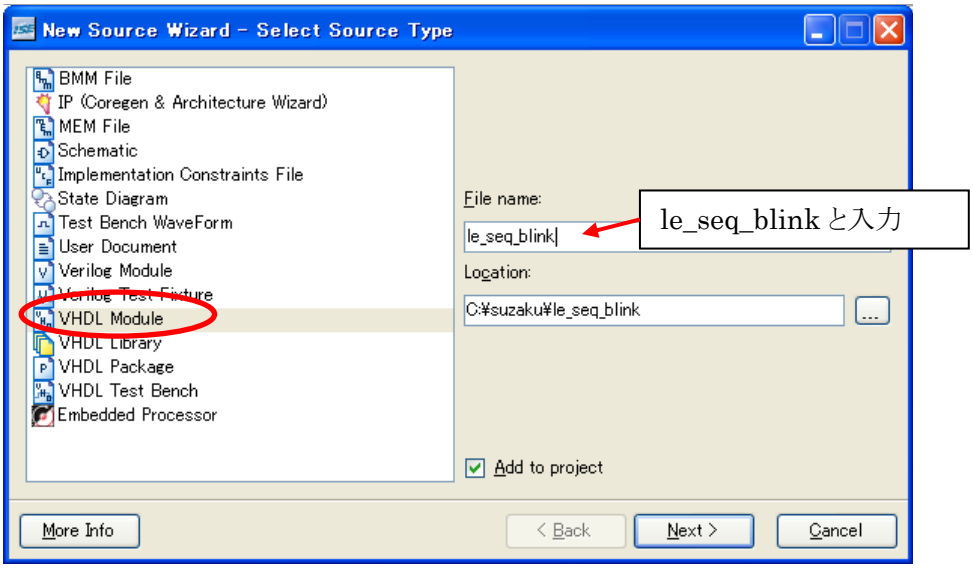


図 10-3 New Source 名前入力

top – IMP(top.vhd)を右クリックしてメニューを出し、[Add Copy of Source...]を選択してください。

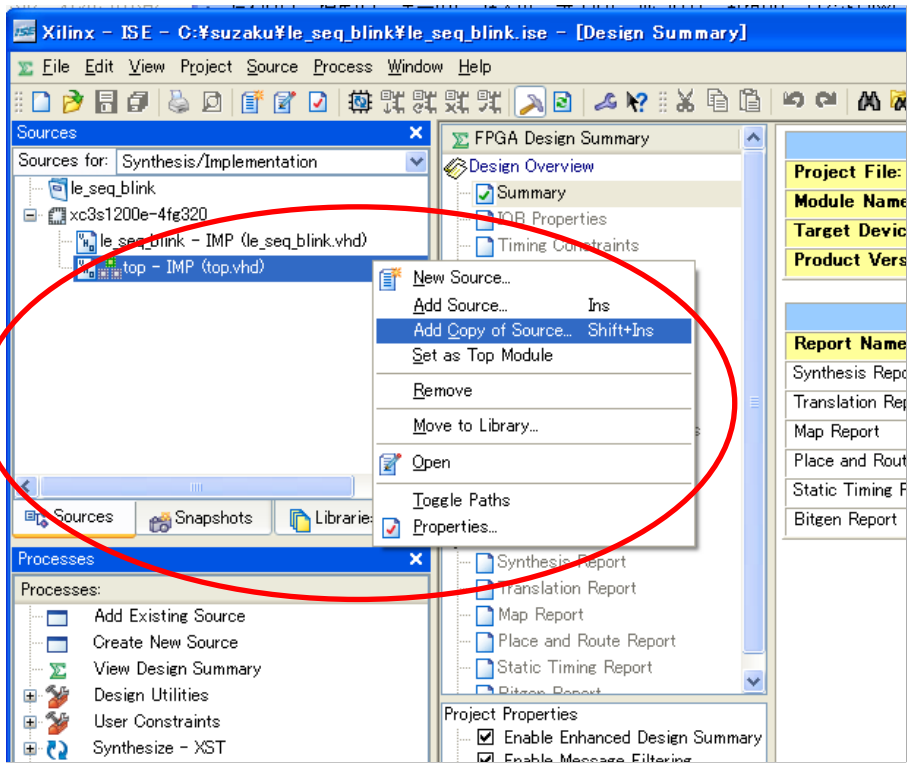


図 10-4 既存のソースファイル追加

先ほどシミュレーションで作った slot_counter.vhd を選択してください。
下図が表示されるので、[OK]をクリックしてください。プロジェクトに slot_counter.vhd が追加されます。

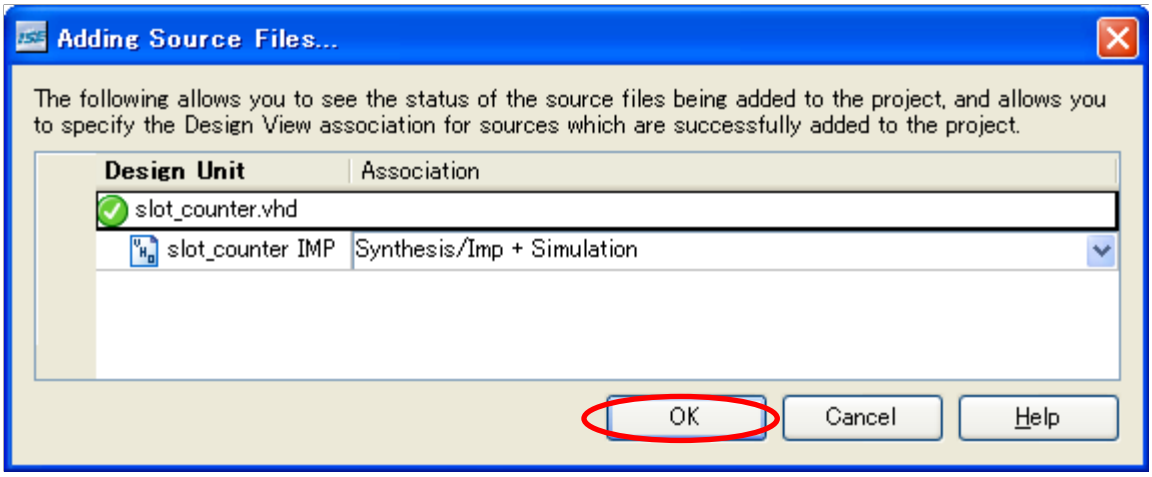


図 10-5 既存のソースファイル追加時の確認

top.vhd を上位階層とします。top-IMP(top.vhd)の上で右クリックしメニューを出し、[Set as Top Module]を選択してください。のマークがついているのが上位階層になります。

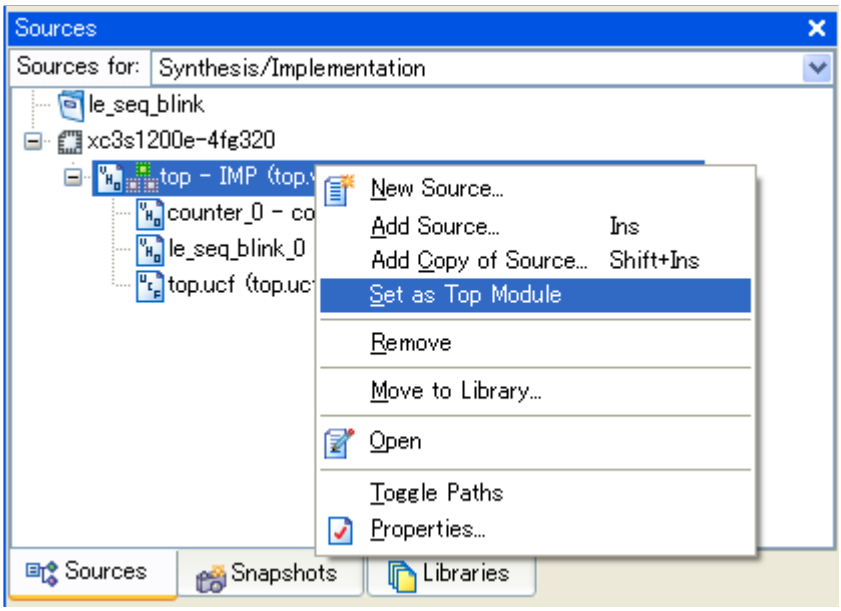


図 10-6 上位階層に設定

■ le_seq_blink.vhd

単色 LED を順次点灯させる回路を記述します。記述できたら保存して、le_seq_blink-IMP(le_seq_blink.vhd)を選択し、Check Syntax をダブルクリックして、文法チェックをしてください。

例 10-1 単色 LED 順次点灯(le_seq_blink.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity le_seq_blink is
    Port (
        SYS_CLK: in STD_LOGIC; --クロック信号
        SYS_RST : in STD_LOGIC; --リセット信号
        le_timing : in  STD_LOGIC; --単色 LED 順次点灯のタイミング信号
        le : out  STD_LOGIC_VECTOR(0 to 3) --単色 LED 出力信号
        );
end le_seq_blink;

architecture IMP of le_seq_blink is
--内部信号の定義
        signal le_w : STD_LOGIC_VECTOR(0 to 3); --単色 LED 内部信号
        signal le_tim : STD_LOGIC; --単色 LED 順次点灯のタイミング内部信号
        signal le_tim_reg : STD_LOGIC; --単色 LED 順次点灯タイミング信号の 1 クロック前の値
begin

  process(SYS_CLK) --クロック信号に変化があると実行
  begin
        if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
                if SYS_RST = '1' then --リセットされたら(同期リセット)
                        le_tim_reg <= '0'; --初期化
                else
                        le_tim_reg <= le_timing; --1 クロック前の値を保持
                end if;
        end if;
  end process;

  process(SYS_CLK) --クロック信号に変化があると実行
  begin
        if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
                if SYS_RST = '1' then --リセットされたら(同期リセット)
                        le_tim <= '0'; --初期化
                else
                        le_tim <= le_timing and (not le_tim_reg); --エッジ検出
                end if;
        end if;
  end process;

  process(SYS_CLK) --クロック信号に変化があると実行
  begin
        if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
                if SYS_RST = '1' then --リセットされたら(同期リセット)
                        le_w <= "0001"; --はじめに D1 を光らせる
                else
                        if le_tim = '1' then --タイミング信号の値が'1'になったら
                                le_w <= le_w(1 to 3) & le_w(0); --1bit 左にシフト
```

```
                    end if;
                end if;
            end if;
    end process;

    le <= le_w; --外部に出力

end IMP;
```

■ top.vhd

top.vhd を上位階層として slot_counter と led_seq_blink の回路を呼び出します。記述できたら top-IMP(top.vhd)を選択し、Synthesize をダブルクリックして、文法チェックをしてください。

例 10-2 単色 LED 順次点灯(top.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top is
    generic (
        C_CNT_WIDTH : integer := 19 --カウンタのビット幅
    );
    Port (
        SYS_CLK: in STD_LOGIC; --クロック信号
        SYS_RST : in STD_LOGIC;      --リセット信号
        nLE : out  STD_LOGIC_VECTOR(0 to 3) --単色 LED 出力信号(負論理)
        );
end top;

architecture IMP of top is
        signal count : STD_LOGIC_VECTOR(0 to C_CNT_WIDTH-1);
        signal le : STD_LOGIC_VECTOR(0 to 3); --単色 LED 内部信号

  component slot_counter
    generic (
        C_CNT_WIDTH : integer := C_CNT_WIDTH --カウンタのビット幅
    );
    Port (
        SYS_CLK : in STD_LOGIC; --クロック信号
        SYS_RST : in STD_LOGIC; --リセット信号
        count : out STD_LOGIC_VECTOR(0 to C_CNT_WIDTH-1) --カウンタ値
        );
  end component;

  component le_seq_blink
    Port (
        SYS_CLK: in STD_LOGIC; --クロック信号
        SYS_RST : in STD_LOGIC; --リセット信号
        le_timing : in  STD_LOGIC; --単色 LED 順次点灯のタイミング信号
        le : out  STD_LOGIC_VECTOR(0 to 3) --単色 LED 出力信号
        );
  end component;

begin
```

```
  slot_counter_0 : slot_counter
    Port map(
      SYS_CLK => SYS_CLK,
      SYS_RST => SYS_RST,
      count => count
      );

  le_seq_blink_0 : le_seq_blink
    Port map(
      SYS_CLK => SYS_CLK,
      SYS_RST => SYS_RST,
      le_timing => count(0), --カウンタの最上位ビットを接続
      le => le
      );

nLE <= not le; --外部に出力

end IMP;
```

● タイミング信号生成(エッジ検出)

カウンタの最上位ビットの前回の値を保持し、その値と今回の最上位ビットの値が違ったならば信号を出力します。

例 10-3　　エッジ検出

```
process(SYS_CLK) --クロック信号に変化があると実行
begin
        if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
                if SYS_RST = '1' then --リセットされたら(同期リセット)
                        le_tim_reg <= '0'; --初期化
                else
                        le_tim_reg <= le_timing; --1 クロック前の値を保持
                end if;
        end if;
end process;

process(SYS_CLK)
begin
        if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
                if SYS_RST = '1' then --リセットされたら(同期リセット)
                        le_tim <= '0'; --初期化
                else
                le_tim <= le_timing and (not le_tim_reg); --エッジ検出
                end if;
        end if;
end process;
```

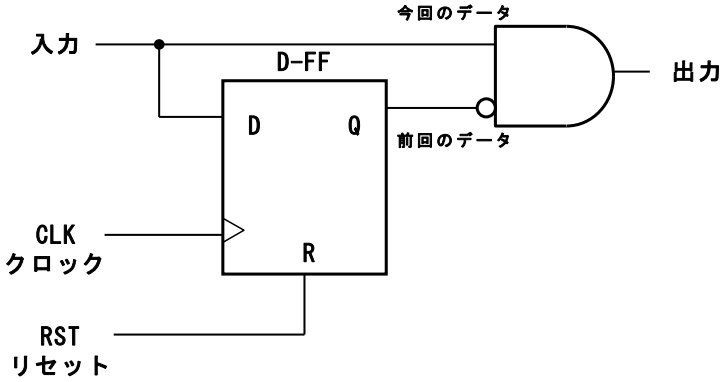
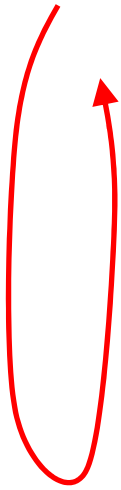


図 10-7 エッジ検出回路

count(0)	count(1)	count(2)	count(3)
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

最大値までカウントしたら 0 にもどってカウントし続ける



最上位ビットに注目

図 10-8 最上位ビットの動作(4 ビットカウンタの場合)

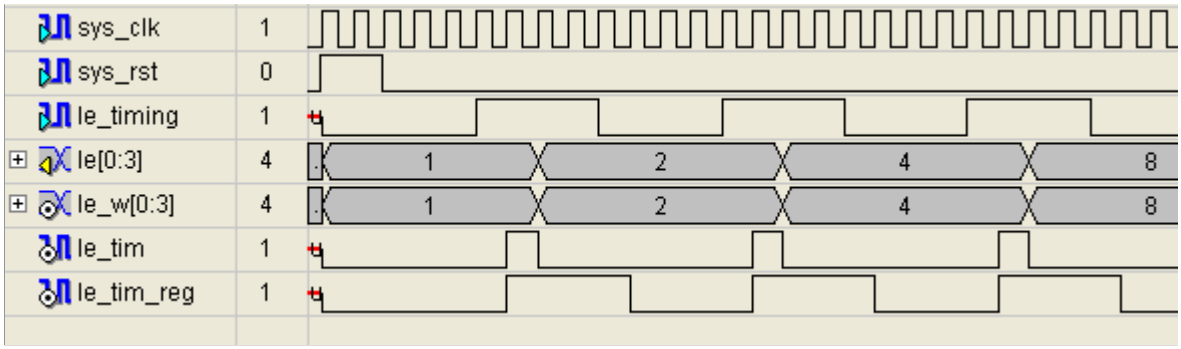


図 10-9 エッジ検出の波形(4 ビットカウンタの場合)

● シフトレジスタ

シフトレジスタは、記憶しているデータの桁を左右にシフトさせることができるレジスタです。左にシフトするには以下の記述をします。

例 10-4 シフトレジスタ

```
process(SYS_CLK) --クロック信号に変化があると実行
begin
	if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
		if SYS_RST = '1' then --リセットされたら(同期リセット)
			le <= "0001";
		else
			if le_tim = '1' then --タイミング信号の値が'1'になったら
				le <= le(1 to 3) & le(0); --1bit 左にシフト
			end if;
		end if;
	end if;
end process;
```

● &について

&を使うと bit を連結することができます。

例 10-5 bit 連結

```
le <= le(1 to 3) & le(0);
```

(1 to 3) で、1ビット目から 3 ビット目までを切り出すことができます。(to で幅を設定している場合は to、downto で幅を設定している場合は downto で切り出す)イベントが発生するたびに最上位ビットを最下位に連結させることにより、"1"の値を順に左にシフトさせます。

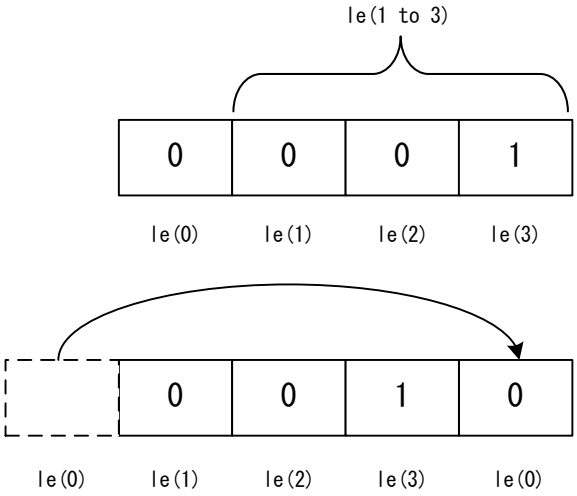


図 10-10 bit 連結

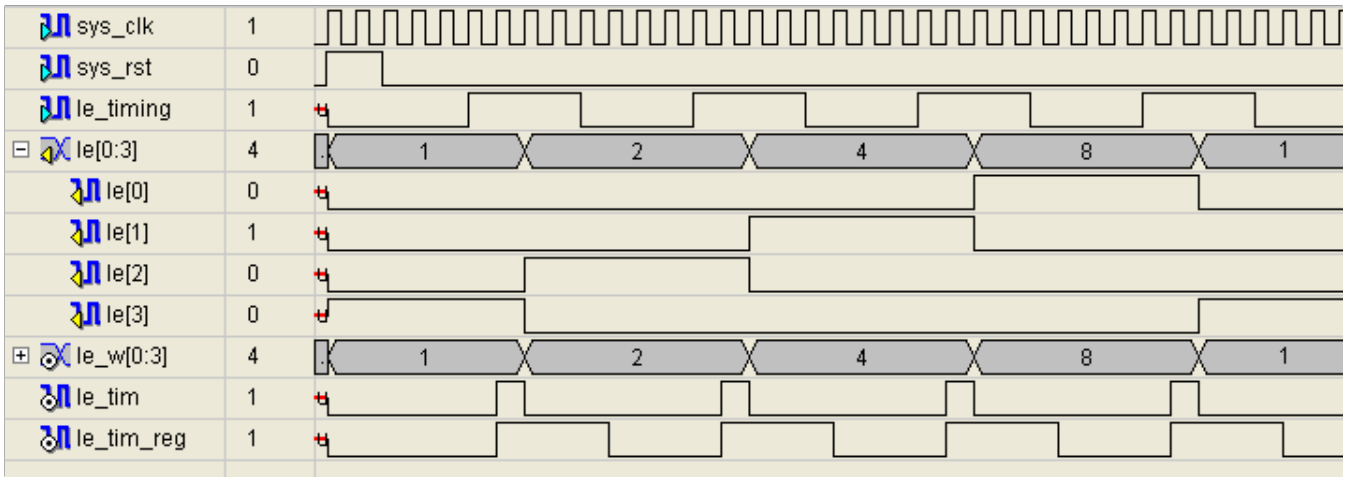


図 10-11 シフトレジスタの波形(4 ビットカウンタの場合)

● コンポーネント

上位のメイン回路から下位回路を呼び出すためには、エンティティ文の中でコンポーネントとして定義します。

例 10-6 component 文

```
component コンポーネント名
  Port (
      信号名 : 入出力方向 データタイプ
      );
end component;
```

コンポーネントの定義が終わったら、アーキテクチャ文の begin の下で呼び出します。

下位回路のポートと信号は port map で結合します。ラベル名は、このコンポーネントにつけられる名前で、そのアーキテクチャ内でユニークな名前でなければいけません。

例 10-7 port map 文

```
ラベル名 : コンポーネント名
Port map (
        ポート名=>信号名
          );
```

■ ピンアサイン

単色 LED への出力信号を STD_LOGIC_VECTOR(0 to n)で定義しました。to で定義すると MSB 側が 0 ビット目になります。この場合信号を入出力する前に、信号の MSB と LSB をひっくり返さなければいけません。VHDL のソースでひっくり返してもいいのですが、最後にピンアサインでひっくり返します。例えば nLE0 は nLE<3>、nLE1 は nLE<2>、nLE2 は nLE<1>、nLE3 は nLE<0>にピンアサインします。(”6.2. ピンアサイン” 参照)

ピンアサインができたら、Implement Design をダブルクリックしてください。

図 10-12 ピンアサインでひっくり返す

表 10-1 単色 LED 順次点灯ピンアサイン

	SZ010 SZ030	SZ130	SZ310
SYS_CLK	T9	U10	C8
SYS_RST	F5	D3	A8
nLE<0>	E11	A11	L13
nLE<1>	D11	B11	L14
nLE<2>	C12	F12	L15
nLE<3>	B12	E12	L16

■ コンフィギュレーション

Generate Programming File をダブルクリックし、bit ファイルを作ってください。
Configure Device(iMPACT)をダブルクリックし、iMPACT を立ち上げ、コンフィギュレーションしてください。
単色 LED が D1→D2→D3→D4→D1→・・・・と順次点灯するのを確認してください。

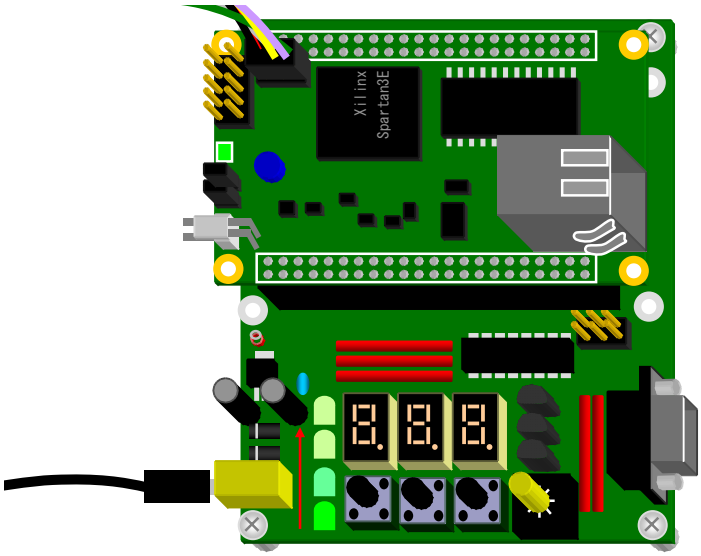


図 10-13 単色 LED 順次点灯

10.2. 7 セグメント LED デコーダ

7 セグメント LED に数字を表示させて回転させます。まずは数字を表示するために 7 セグメント LED のデコーダを作ります。デコーダを作っただけでは数字が表示できているかどうか分からないので、ここではロータリコードスイッチからの入力を 7 セグメント LED に表示する回路を作ります。

10.2.1. ロータリコードスイッチ周辺回路

LED/SW ボードに実装されているロータリコードスイッチは 4bit で 0～F までの数字を表現できます。それぞれ 1kΩの抵抗で 3.3V にプルアップされています。負論理なので内部で正論理にして使います。正論理にした場合のそれぞれの”High”(”1”)、”Low”(“0”)は”表 10-1 ロータリコードスイッチ（正論理）”のようになります。

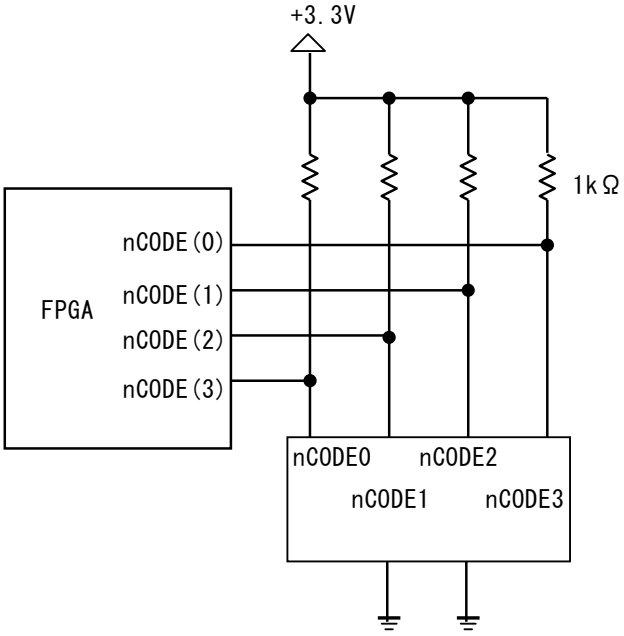


図 10-14 ロータリコードスイッチ周辺回路とピンアサイン

表 10-2 ロータリコードスイッチ(正論理)

数字	CODE3	CODE2	CODE1	CODE0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
b	1	0	1	1
C	1	1	0	0
d	1	1	0	1
E	1	1	1	0
F	1	1	1	1

10.2.2. 7 セグメント LED 周辺回路

7 セグメント LED のセグメントは下図のように配置されていて、A~G までの各発光ダイオードの適当なものだけを光らすと数字を表示することができます。“表 10-3 7 セグメント LED デコーダ(正論理)”と照らし合わせてどう光らせれば数字になるか確認してみてください。

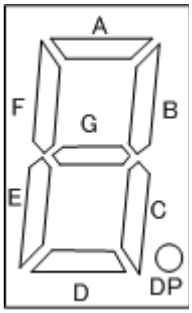


図 10-15 セグメントの配置

表 10-3 7 セグメント LED デコーダ(正論理)

数字	SEG7(DP)	SEG6(G)	SEG5(F)	SEG4(E)	SEG3(D)	SEG2(C)	SEG1(B)	SEG0(A)
0	0	0	1	1	1	1	1	1
1	0	0	0	0	0	1	1	0
2	0	1	0	1	1	0	1	1
3	0	1	0	0	1	1	1	1
4	0	1	1	0	0	1	1	0
5	0	1	1	0	1	1	0	1
6	0	1	1	1	1	1	0	1
7	0	0	1	0	0	1	1	1
8	0	1	1	1	1	1	1	1
9	0	1	1	0	1	1	1	1
A	0	1	1	1	0	1	1	1
B	0	1	1	1	1	1	0	0
C	0	0	1	1	1	0	0	1
D	0	1	0	1	1	1	1	0
E	0	1	1	1	1	0	0	1
F	0	1	1	1	0	0	0	1

LED/SW ボードには 7 セグメント LED が 3 つ実装されていて、Q1 に”Low”を入力すると LED1、Q2 に“Low”を入力すると LED2、Q3 に”Low”を入力すると LED3 を扱うことができます。Q1、Q2、Q3 を同時に”Low”にすることで、全部を光らすこともできますが、同じ数字しか表示することはできません。異なる数字を表示したいときはダイナミック点灯という手法を用います。("10.3. ダイナミック点灯” 参照)

Q1～Q3 のセレクト信号は負論理、7 セグメント LED は正論理です。7 セグメント LED が正論理なのは電流を増やすために、バッファとしてインバータが 7 セグメント LED の前に実装されているためです。

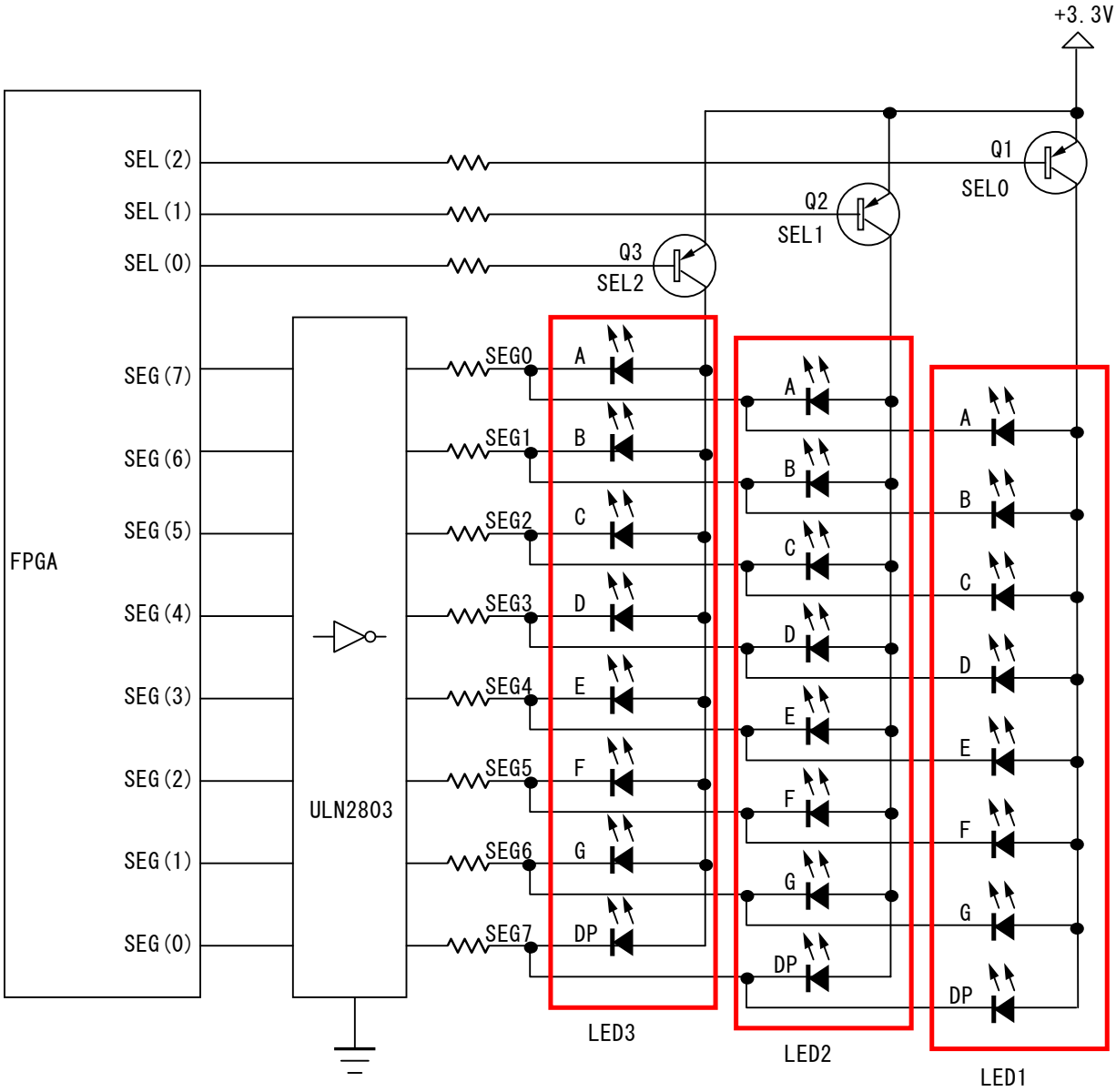


図 10-16 7 セグメント LED 周辺回路

10.2.3. 7 セグメント LED デコーダ VHDL

プロジェクトを新規作成してください。
プロジェクト名は seg7_decorder とし、new Source で top.vhd を作ってください。
top – IMP(top.vhd)を右クリックしてメニューを出し、[New Source...]を選択し、seg7_decorder.vhd を新規作成してください。
top-IMP(top.vhd)の上で右クリックしメニューを出し、[Set as Top Module]を選択し、top.vhd を上位階層としてください。

■ seg7_decorder.vhd
7 セグメント LED のデコーダ回路を記述してください。記述できたら、seg7_decirder-IMP(seg7_decorder.vhd)を選択し、Check Syntax をダブルクリックして、文法チェックをしてください。

例 10-8 7 セグメント LED デコーダ(seg7_decorder.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity seg7_decorder is
    Port (
          SEG : out STD_LOGIC_VECTOR(0 to 7); --7 セグメント LED への出力信号
          seg_data : in STD_LOGIC_VECTOR(0 to 3) --4bit バイナリコード
          );
end seg7_decorder;
architecture IMP of seg7_decorder is
begin
        --デコーダ記述
        process(seg_data)
                begin
                        case seg_data is
                                when "0000" => SEG <= "00111111"; --0
                                when "0001" => SEG <= "00000110"; --1
                                when "0010" => SEG <= "01011011"; --2
                                when "0011" => SEG <= "01001111"; --3
                                when "0100" => SEG <= "01100110"; --4
                                when "0101" => SEG <= "01101101"; --5
                                when "0110" => SEG <= "01111101"; --6
                                when "0111" => SEG <= "00100111"; --7
                                when "1000" => SEG <= "01111111"; --8
                                when "1001" => SEG <= "01101111"; --9
                                when "1010" => SEG <= "01110111"; --A
                                when "1011" => SEG <= "01111100"; --b
                                when "1100" => SEG <= "00111001"; --C
                                when "1101" => SEG <= "01011110"; --d
                                when "1110" => SEG <= "01111001"; --E
                                when others => SEG <= "01110001"; --F
                        end case;
        end process;
end IMP;
```

■ top.vhd

top.vhd を上位階層として seg7_decorder の回路を呼び出します。記述できたら top-IMP(top.vhd)を選択し、Synthesize をダブルクリックして、文法チェックをしてください。

例 10-9 7 セグメント LED デコーダ(top.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top is
    Port (
          nCODE : in STD_LOGIC_VECTOR(0 to 3); --ロータリコードスイッチからの入力信号(負論理)
          SEG : out STD_LOGIC_VECTOR(0 to 7); --7 セグメント LED への出力信号(正論理)
          nSEL : out STD_LOGIC_VECTOR(0 to 2) --7 セグメント LED セレクト信号(負論理)
          );
end top;

architecture IMP of top is
        signal code : STD_LOGIC_VECTOR(0 to 3); --ロータリコードスイッチ内部信号(正論理)
        signal sel : STD_LOGIC_VECTOR(0 to 2); --7 セグメント LED セレクト内部信号(正論理)

  component seg7_decorder
    Port (
          SEG : out STD_LOGIC_VECTOR(0 to 7); --7 セグメント LED への出力信号
          seg_data : in STD_LOGIC_VECTOR(0 to 3) --4bit バイナリコード
          );
  end component;

begin
  seg7_decorder_0 : seg7_decorder
    Port map(
      SEG => SEG,
      seg_data => code
      );
sel <= "001"; --7 セグメント LED1 を点灯させる
nSEL <= not sel; --負論理にして出力
code <= not nCODE; --正論理にして入力

end IMP;
```

● case 文

case 文は次の書式で記述します。

例 10-10 case 文

```
  case 式 is
  when 値 => 順次処理文
  when others => 順次処理文
  end case;
```

■ ピンアサイン

信号を STD_LOGIC_VECTOR(0 to n)で定義しているので、MSB 側が 0 ビット目になります。信号は最後にピンアサインでひっくり返しています。例えば nCODE0 は nCODE<3>、nCODE1 は nCODE<2>、nCODE2 は nCODE<1>、nCODE3 は nCODE<0>にピンアサインします。ピンアサインができたら、Implement Design をダブルクリックしてください。

表 10-4 7セグメント LED デコーダ ピンアサイン

	SZ010 SZ030	SZ130	SZ310
nCODE<0>	C8	J1	J16
nCODE<1>	A9	F9	J15
nCODE<2>	A12	E9	J14
nCODE<3>	C10	A10	J13
SEG<0>	C5	L5	F15
SEG<1>	B5	L6	F16
SEG<2>	E6	L4	G13
SEG<3>	D6	L3	G14
SEG<4>	C6	L2	G15
SEG<5>	B6	L1	G16
SEG<6>	A8	C9	N9
SEG<7>	B8	D9	P9
nSEL<0>	D7	K6	H13
nSEL<1>	C7	K4	H14
nSEL<2>	B7	K3	H15

■ コンフィギュレーション

Generate Programming File をダブルクリックし、bit ファイルを作ってください。
Configure Device(iMPACT)をダブルクリックし、iMPACT を立ち上げ、コンフィギュレーションしてください。
ロータリコードスイッチをまわすと、対応する数字が 7 セグメント LED(LED1)に表示されます。

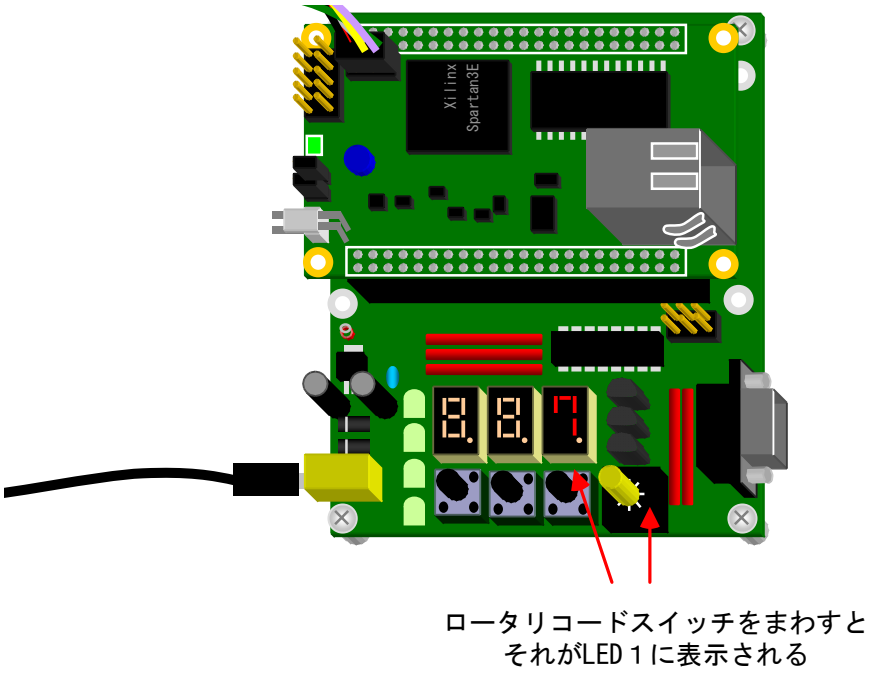


図 10-17 7セグメント LED デコーダ

10.3. ダイナミック点灯

3 つの 7 セグメント LED をダイナミック点灯させます。

ダイナミック点灯とは配線の本数を減らすための手法です。複数の 7 セグメント LED に同じデータ線を接続しています。7 セグメント LED を順次点灯することにより、複数の 7 セグメント LED が同時に点灯しているように見せます。

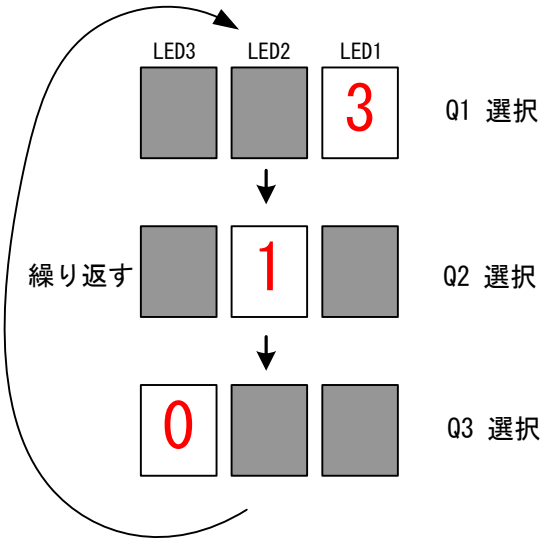


図 10-18 7 セグメント LED ダイナミック点灯

10.3.1. 7 セグメント LED 周辺回路

ダイナミック点灯に必要となる 7 セグメント LED 周辺回路は”図 10-16 7 セグメント LED 周辺回路”を参照してください。

10.3.2. ダイナミック点灯 VHDL

プロジェクトを新規作成してください。

プロジェクト名は dynamic_ctrl とし、new Source で top.vhd を作ってください。

top – IMP(top.vhd)を右クリックしてメニューを出し、[New Source...]を選択し、dynamic_ctrl.vhd を新規作成してください。

top-IMP(top.vhd)を右クリックしてメニューを出し、[Add Copy of Source...]を選択し、slot_counter.vhd、seg7_decorder.vhd を追加してください。

top-IMP(top.vhd)の上で右クリックしメニューを出し、[Set as Top Module]を選択し、top.vhd を上位階層としてください。

■ dynamic_ctrl.vhd

ダイナミック点灯させる回路を記述してください。記述できたら、dynmic_ctrl-IMP(dynamic_ctrl.vhd)を選択し、Check Syntax をダブルクリックして、文法チェックをしてください。

例 10-11 ダイナミック点灯(dynamic_ctrl.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity dynamic_ctrl is
    Port (
          SYS_CLK : in STD_LOGIC; --クロック信号
```

```
          SYS_RST : in STD_LOGIC;     --リセット信号
          nSEL : out STD_LOGIC_VECTOR(0 to 2); --7 セグメント LED セレクト信号(負論理)
          seg7_timing : in  STD_LOGIC; --ダイナミック点灯タイミング信号
          seg_in1 : in  STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED1 の値
          seg_in2 : in  STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED2 の値
          seg_in3 : in  STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED3 の値
          seg_data : out  STD_LOGIC_VECTOR(0 to 3) --4bit バイナリコード
          );
end dynamic_ctrl;

architecture IMP of dynamic_ctrl is

        signal sel : STD_LOGIC_VECTOR(0 to 2); --7 セグメント LED セレクト信号(正論理)
        signal seg_data_w : STD_LOGIC_VECTOR(0 to 3); --4bit バイナリコード内部信号
        signal seg7_tim : STD_LOGIC; --ダイナミック点灯タイミング信号
        signal seg7_tim_reg : STD_LOGIC; --1 クロック前の値

begin
process(SYS_CLK)
begin
        if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
                if SYS_RST = '1' then --リセットされたら(同期リセット)
                        seg7_tim_reg <= '0'; --初期化
                else
                        seg7_tim_reg <= seg7_timing; --値を保持
                end if;
        end if;
end process;
process(SYS_CLK)
begin
        if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
                if SYS_RST = '1' then --リセットされたら(同期リセット)
                        seg7_tim <= '0'; --初期化
                else
                        seg7_tim <= seg7_timing and (not seg7_tim_reg); --エッジ検出
                end if;
        end if;
end process;

process(SYS_CLK) --クロック信号に変化があると実行
begin
        if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
                if SYS_RST = '1' then --リセットされたら(同期リセット)
                        sel <= "001"; --はじめに7セグメント LED1を光らせる
                else
                        if seg7_tim = '1' then --7 セグ用タイミング信号の値が'1'になったら
                                sel <= sel(1 to 2) & sel(0); --1bit 左にシフト
                        end if;
                end if;
        end if;
end process;

process(SYS_CLK)
begin
        if SYS_CLK'event and SYS_CLK = '1' then --クロックの立ち上がりに同期
                if SYS_RST = '1' then --同期リセット
```

```
                        seg_data_w <= "0000"; --初期化
                else
                        --セレクト信号により代入する数字を変える
                        if sel = "001" then
                                seg_data_w <= seg_in1; --7 セグメント LED1 用の数字
                        elsif sel = "010" then
                                seg_data_w <= seg_in2; --7 セグメント LED2 用の数字
                        elsif sel = "100" then
                                seg_data_w <= seg_in3; --7 セグメント LED3 用の数字
                        end if;
                end if;
        end if;
end process;

nSEL <= not sel; --負論理に直して出力
seg_data <= seg_data_w; --外部に出力

end IMP;
```

■ top.vhd

今回は約 1kHz で表示する 7 セグメント LED を切り替えます。そのためにカウンタの 8 ビット目のエッジを取ります。top.vhd を上位階層として slot_counter、seg7_decorder、dynamic_ctrl の回路を呼び出します。記述できたら top-IMP(top.vhd)を選択し、Synthesize をダブルクリックして、文法チェックをしてください。

例 10-12 ダイナミック点灯（top.vhd）

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity top is
    generic (
        C_CNT_WIDTH : integer:= 19 --カウンタのビット幅
    );
    Port (
        SYS_CLK: in STD_LOGIC; --クロック信号
        SYS_RST : in STD_LOGIC; --リセット信号
        nSEL : out  STD_LOGIC_VECTOR(0 to 2); --7 セグメント LED セレクト信号（負論理）
        SEG : out  STD_LOGIC_VECTOR(0 to 7) --7 セグメント LED への出力信号（正論理）
        );
end top;

architecture IMP of top is
        signal seg_in1 : STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED1 の値
        signal seg_in2 : STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED2 の値
        signal seg_in3 : STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED3 の値
        signal seg_data : STD_LOGIC_VECTOR(0 to 3); --4bit バイナリコード
        signal count : STD_LOGIC_VECTOR(0 to C_CNT_WIDTH-1); --カウンタ値

  component slot_counter
    generic (
        C_CNT_WIDTH : integer := C_CNT_WIDTH --カウンタのビット幅
    );
    Port (
          SYS_CLK : in STD_LOGIC; --クロック信号
          SYS_RST : in STD_LOGIC; --リセット信号
```

```
        count : out STD_LOGIC_VECTOR(0 to C_CNT_WIDTH-1) --カウンタ値
        );
  end component;

component dynamic_ctrl
    Port (
        SYS_CLK : in STD_LOGIC; --クロック信号
        SYS_RST : in STD_LOGIC; --リセット信号
        nSEL : out STD_LOGIC_VECTOR(0 to 2); --7 セグメント LED セレクト信号(負論理)
        seg7_timing : in  STD_LOGIC; --ダイナミック点灯タイミング信号
        seg_in1 : in  STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED1 の値
        seg_in2 : in  STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED2 の値
        seg_in3 : in  STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED3 の値
        seg_data : out  STD_LOGIC_VECTOR(0 to 3) --4bit バイナリコード
        );
end component;

component seg7_decorder
    Port (
        SEG : out STD_LOGIC_VECTOR(0 to 7); --7 セグメント LED への出力信号
        seg_data : in STD_LOGIC_VECTOR(0 to 3) --4bit バイナリコード
        );
  end component;

begin
  slot_counter_0 : slot_counter
    Port map(
      SYS_CLK => SYS_CLK,
      SYS_RST => SYS_RST,
      count => count
      );

  dynamic_ctrl_0 : dynamic_ctrl
    Port map(
      SYS_CLK => SYS_CLK,
      SYS_RST => SYS_RST,
      nSEL => nSEL,
      seg7_timing => count(8), --8 ビット目
      seg_in1 => seg_in1,
      seg_in2 => seg_in2,
      seg_in3 => seg_in3,
      seg_data => seg_data
      );

  seg7_decorder_0 : seg7_decorder
    Port map(
      SEG => SEG,
      seg_data => seg_data
      );

seg_in1 <= "0000"; --0
seg_in2 <= "0001"; --1
seg_in3 <= "0011"; --3

end IMP;
```

■ ピンアサイン

“表 6-1 クロック、リセット信号ピンアサイン”、”表 6-2 機能用ピンアサイン”を参照し、ピンアサインしてください。

今回ここにはピンアサインを載せないので、各自考えてください。どうしても分からない場合は付属 CD-ROM の”¥suzaku-starter-kit¥fpga”の中の”dynamic_ctrl.zip”を展開し、その中にある”top.ucf”を参照してください。

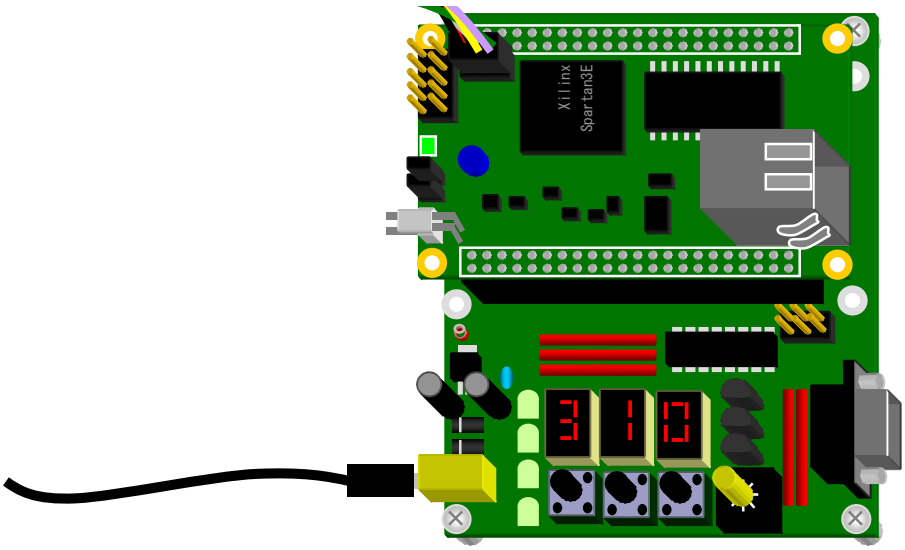
ピンアサインができたら、Implement Design をダブルクリックしてください。

■ コンフィギュレーション

Generate Programming File をダブルクリックし、bit ファイルを作ってください。

Configure Device(iMPACT)をダブルクリックし、iMPACT を立ち上げ、コンフィギュレーションしてください。

7 セグメント LED に”3”、”1”、”0”と表示されます。他の数字も表示してみてください。



ダイナミック点灯で
数字が表示されます

図 10-19 ダイナミック点灯

11. EDK の使い方

EDK はプロセッサや周辺ペリフェラルのソースコードやライブラリが登録されており、それらを GUI 環境下で構築、設定できるツールです。ユーザで作った IP コアも登録することができます。また、ソフトウェアのコンパイラやライブラリが登録されており、C 言語による開発も行うことができます。ISE の機能を取り込んでいるので、論理合成、マッピングを行うことができ、生成されたバイナリファイルを任意の BRAM にいれて、コンフィギュレーションファイルの生成を行うこともできます。

SUZAKU のデフォルトは EDK で構築されており、SUZAKU を使いこなすには EDK を使えなければいけません。ここでは一旦スロットマシンと離れ、SUZAKU のデフォルトに GPIO、UART の追加をすると共に EDK の使い方を説明します。本書では EDK8.1i を使用しています。EDK の使い方の詳細は EDK のヘルプ等を参照してください。EDK には日本語のマニュアルも用意されています。

11.1. SUZAKU のデフォルト

付属 CD-ROM の”¥suzaku¥fpga_proj¥8.1i¥sz***”(***は型式)の中の圧縮ファイル”sz***-xxxxxxxx.zip”(x は更新日)をハードディスクに展開してください。ここでは展開後のフォルダを”C:¥suzaku”の下にコピーし、フォルダの名前を”sz***-add_uart_gpio”に変更して作業を進めます。

”C:¥suzaku¥sz***-add_uart_gpio”の中の”xps_proj.xmp”をダブルクリックして開いてください。

Xilinx Platform Studio が起動し、下図が立ち上がります。これが SUZAKU のデフォルトになります。SUZAKU のデフォルトは Linux が動く最小構成になっています。

この SUZAKU のデフォルトに変更を加えていきます。

11.1.1. SZ010、SZ030 のデフォルト

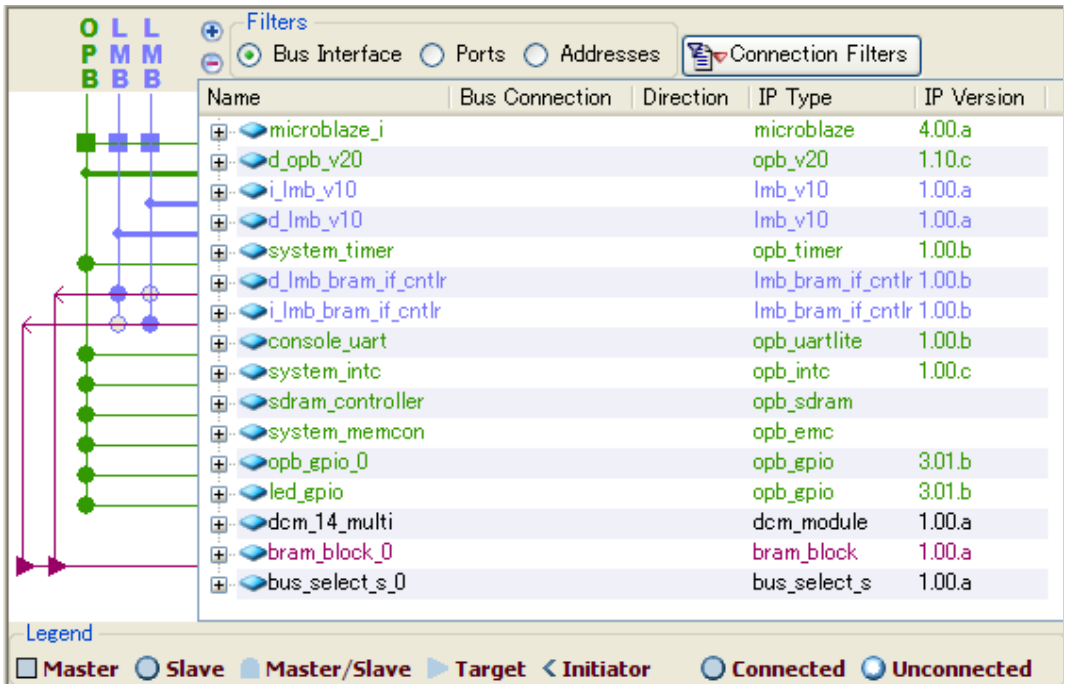


図 11-1 SZ010、SZ030 のデフォルト(EDK)

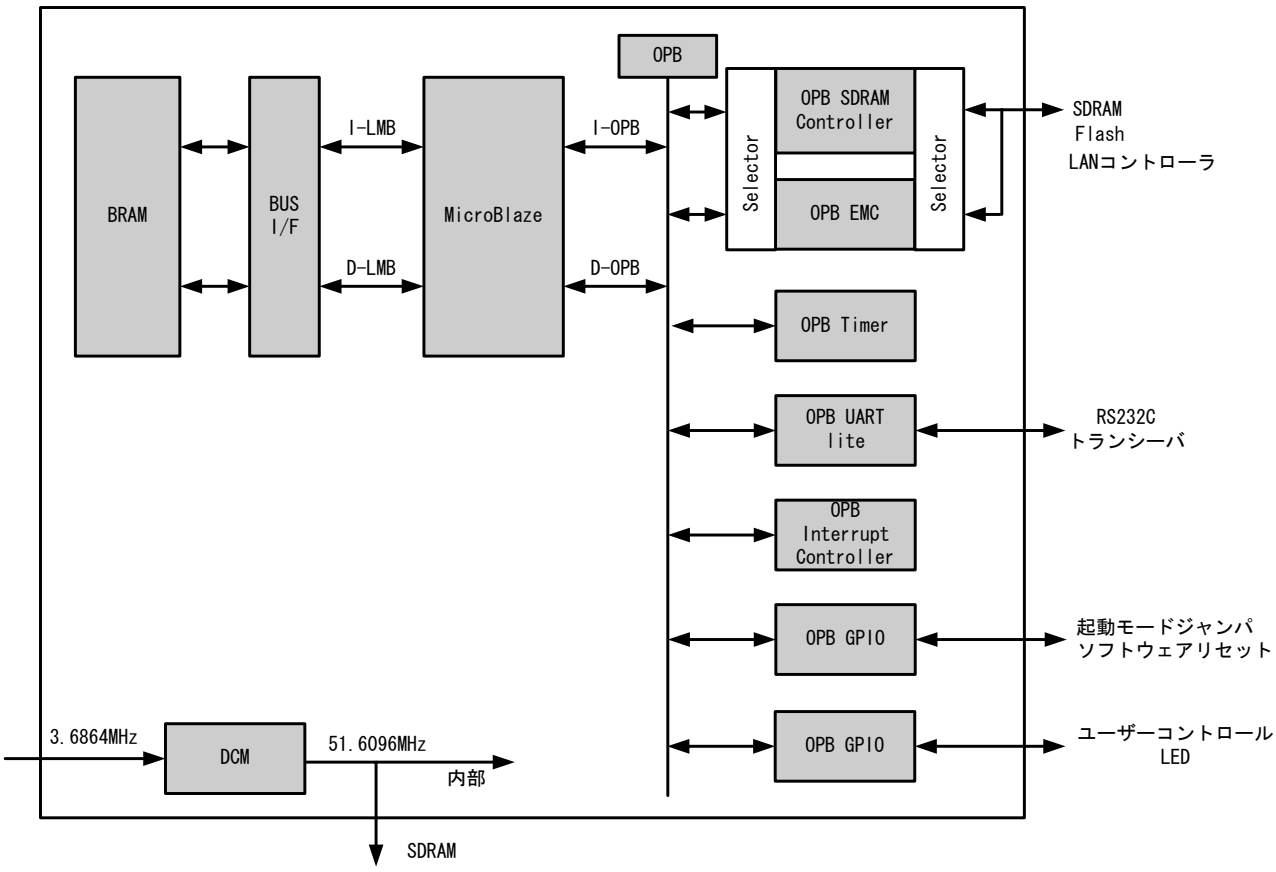


図 11-2 SZ010、SZ030 デフォルトのブロック図

11.1.2. SZ130 のデフォルト

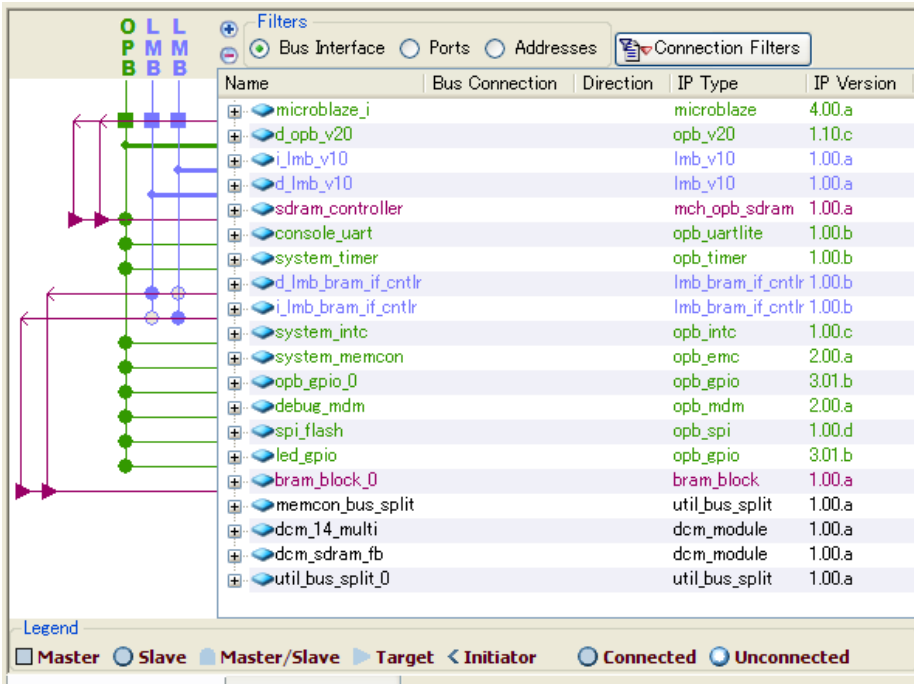


図 11-3 SZ130 のデフォルト(EDK)

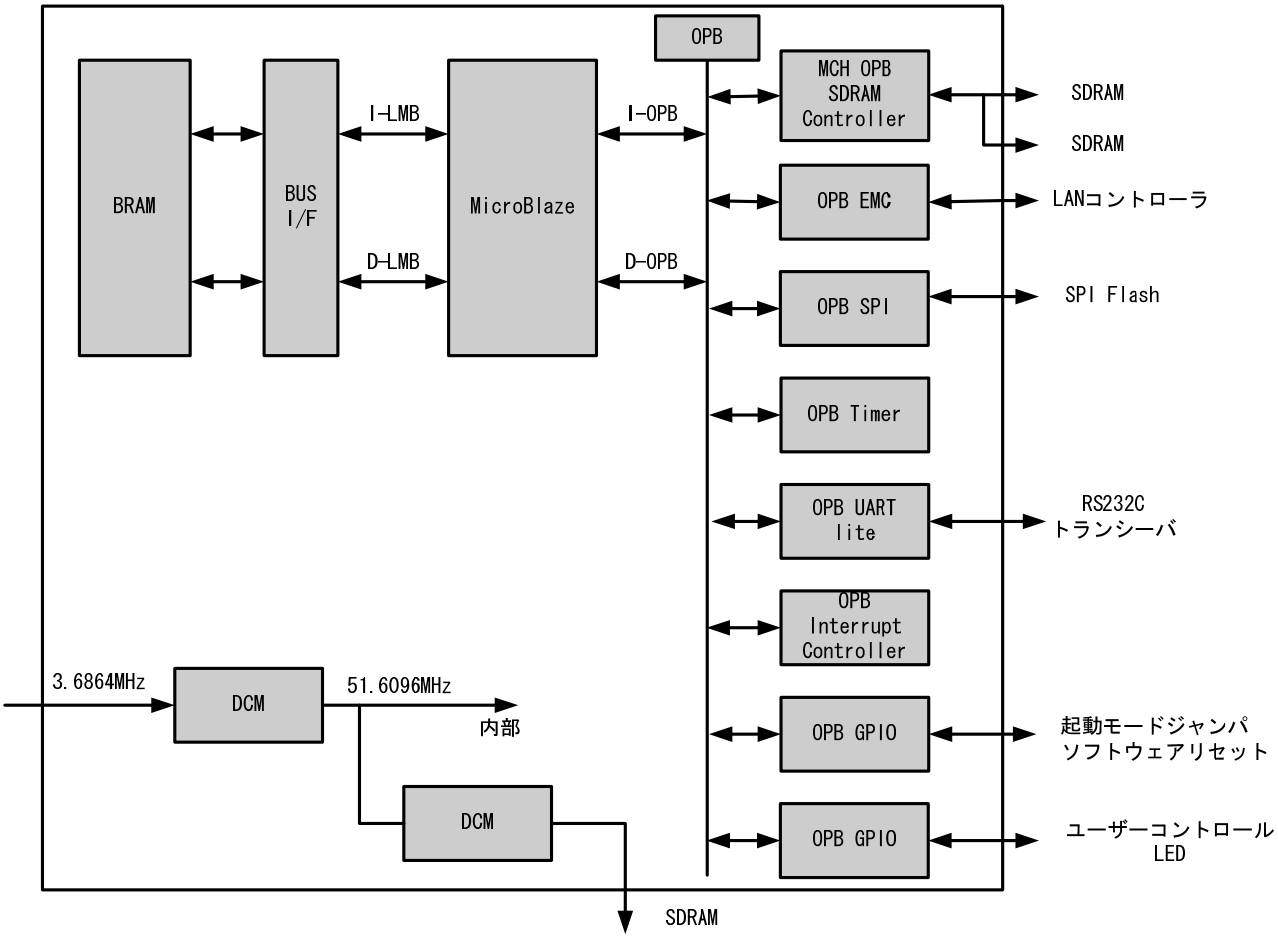


図 11-4 SZ130 デフォルトのブロック図

11.1.3. SZ310 のデフォルト

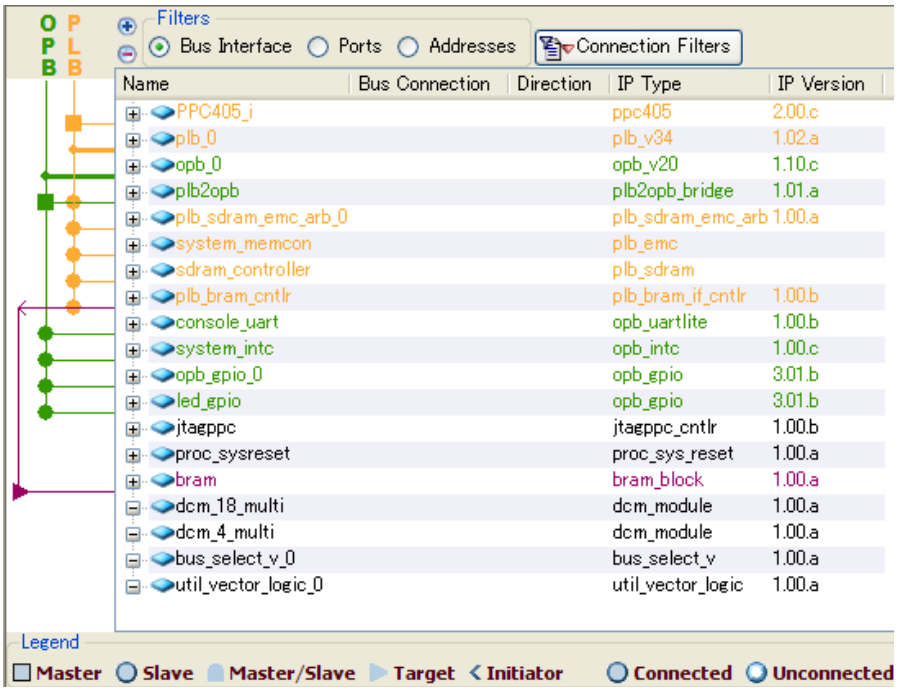


図 11-5 SZ310 のデフォルト(EDK)

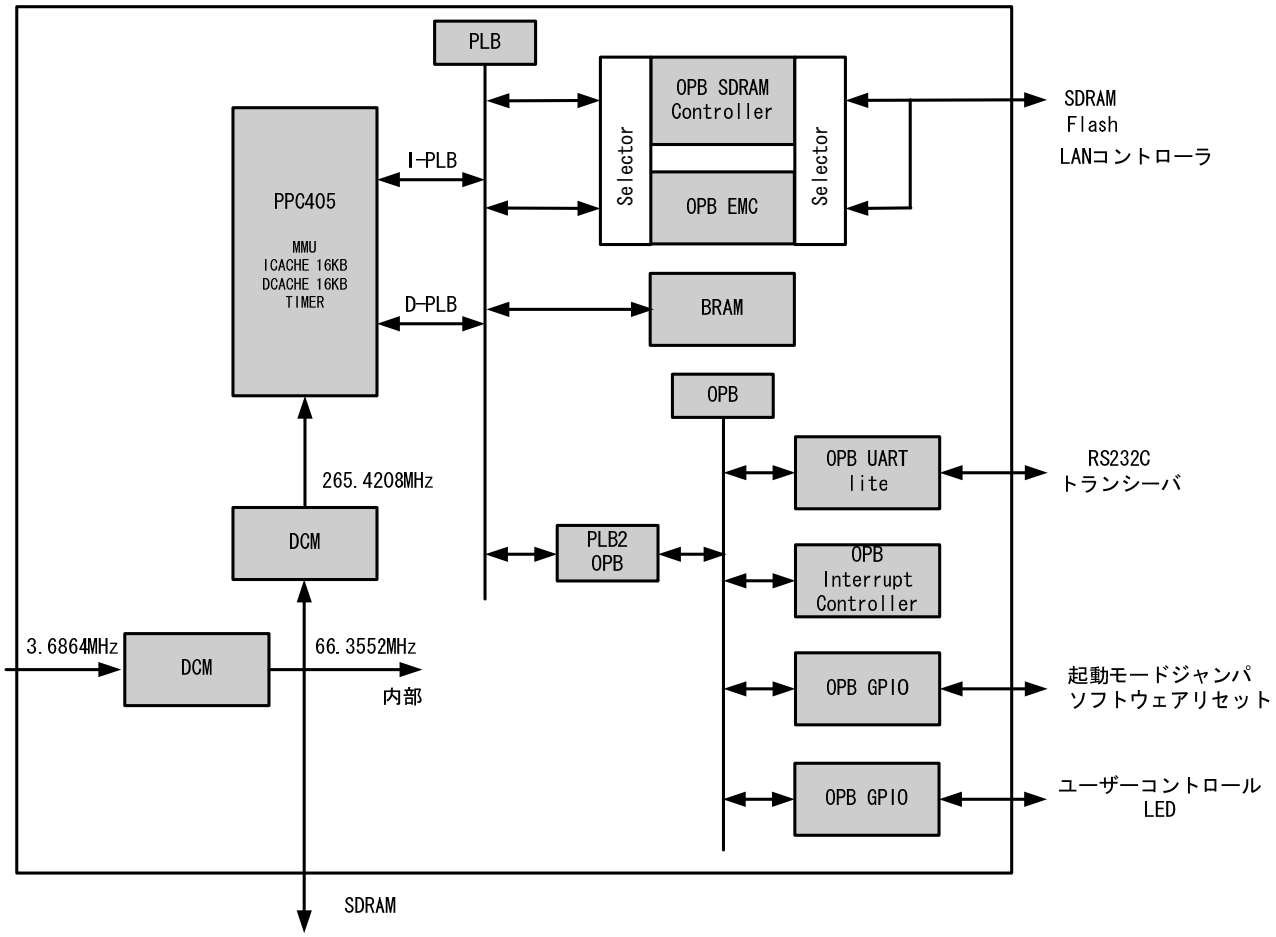


図 11-6 SZ310 デフォルトのブロック図

11.2. GPIO の追加

SUZAKU のデフォルトに GPIO を追加して、BRAM の中のソフトウェアに単色 LED（D1）を点灯させる一文を追加します。

11.2.1. IP コアの追加

IP コアを追加します。IP Catalog のタブをクリックしてください。IP Catalog には EDK に登録されている IP コアやユーザが登録した IP コアの一覧が表示されます。ここから使いたい IP コアを選択し、追加することができます。

General Purpose IO の中にある opb_gpio を右クリックしてメニューを出し、Add IP を選択してください。opb_gpio が追加されます。

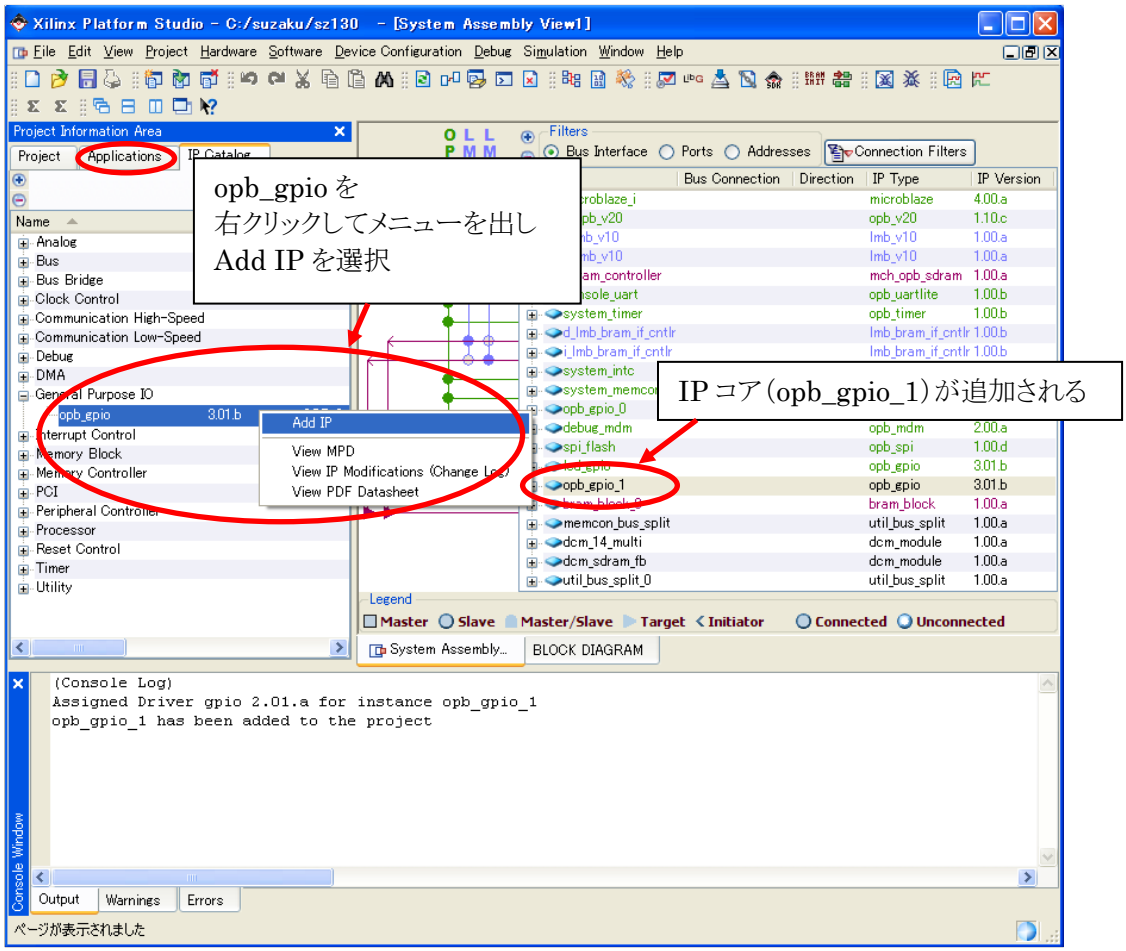


図 11-7 opb_gpio の追加

11.2.2. OPB バスに接続

OPB に GPIO を接続します。OPB は MicroBlaze やペリフェラルを接続するためのバスです。
Bus Interface を選択し、opb_gpio_1 の横の丸をクリックしてください。○ → ●
これで OPB バスに GPIO が接続されます。

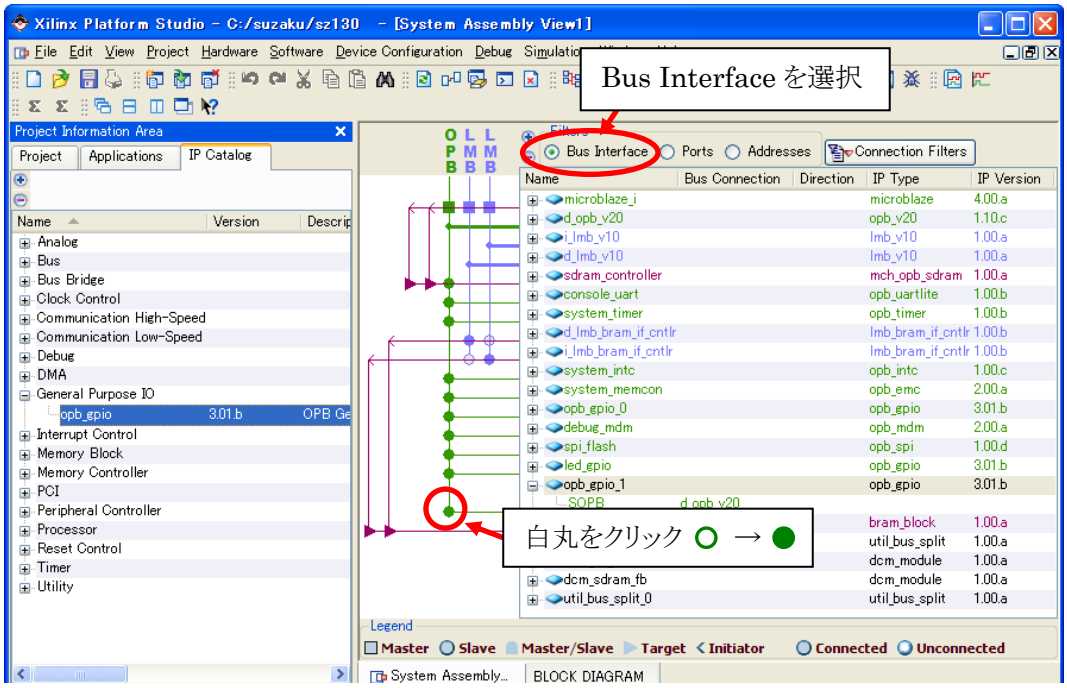


図 11-8 OPB バスに接続

11.2.3. IP コアの設定

IP コアはさまざまな設定をすることができます。今回追加した GPIO では GPIO の本数、出力の属性、プロセッサから制御する際の BaseAddress などを設定することができます。
opb_gpio_1 を右クリックしてメニューを出し、[Configure IP]を選択してください。

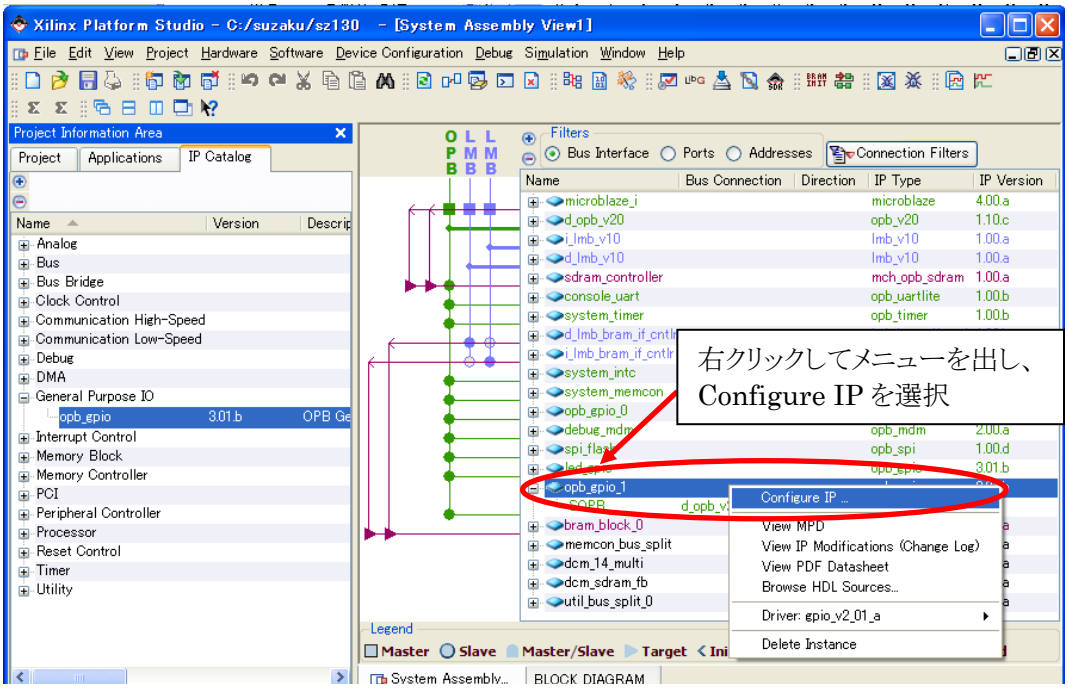


図 11-9 Configure IP

単色 LED を 1 つだけ光らすので、バスの幅は 1 ビットにします。

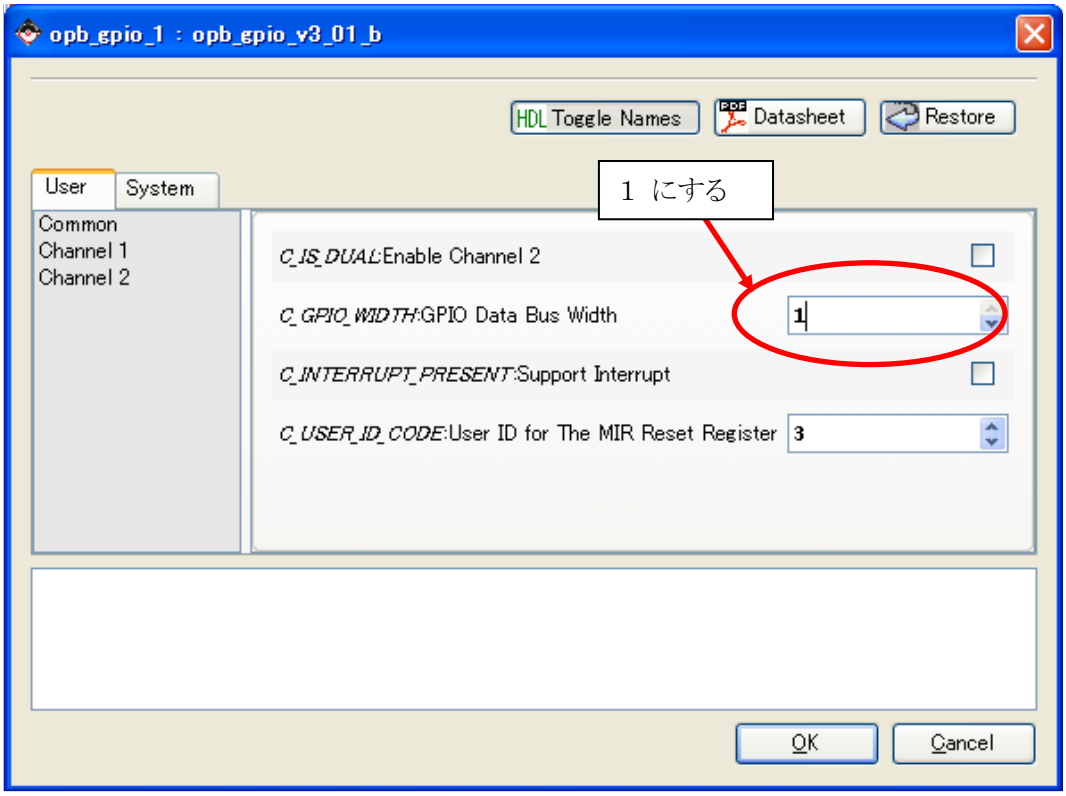


図 11-10 バス幅の設定

Channel1 を選択し、Bi-directional を FALSE にしてください。

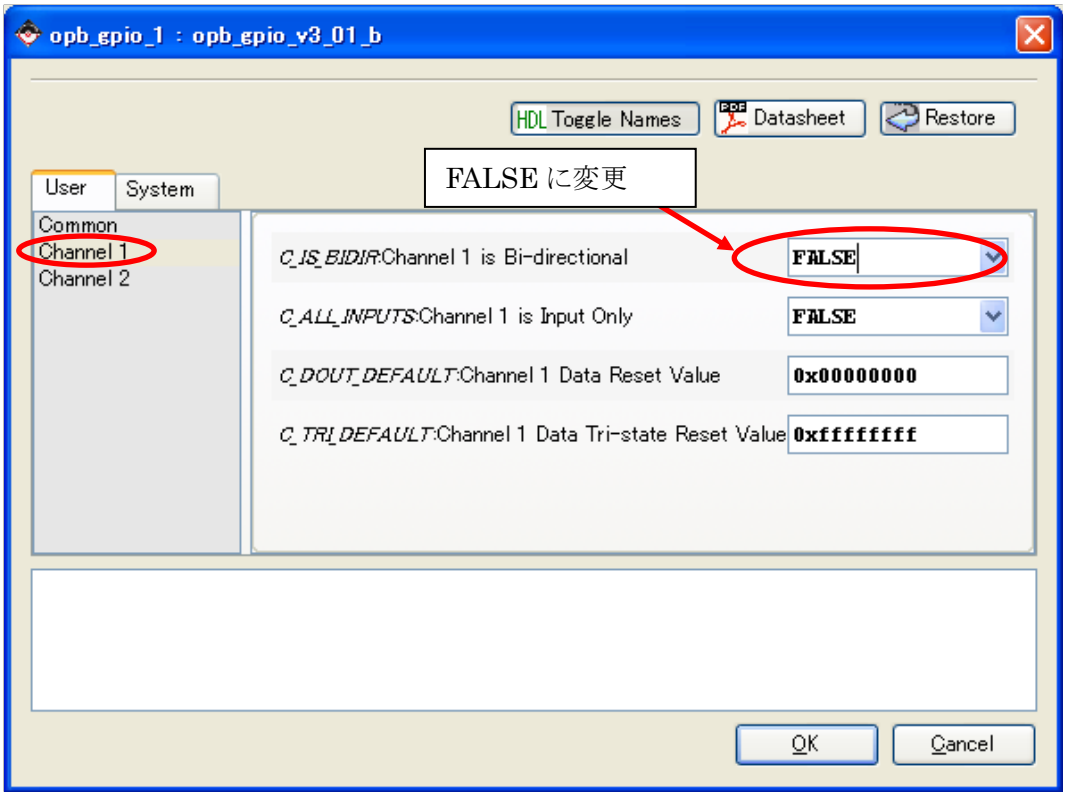


図 11-11 その他設定変更

System タブをクリックし、[Base Address]、[High Address]にそれぞれメモリアドレスを入力して、[OK]をクリックしてください。メモリアドレスは SUZAKU のメモリマップで Free と書いてあるところに割り当てます。
(”5.3 SUZAKU メモリマップ”参照)

表 11-1 GPIO メモリアドレス

	SZ010、SZ030 SZ130	SZ310
Base Address	0xFFFFA400	0xF0FFA400
High Address	0xFFFFA5FF	0xF0FFA5FF

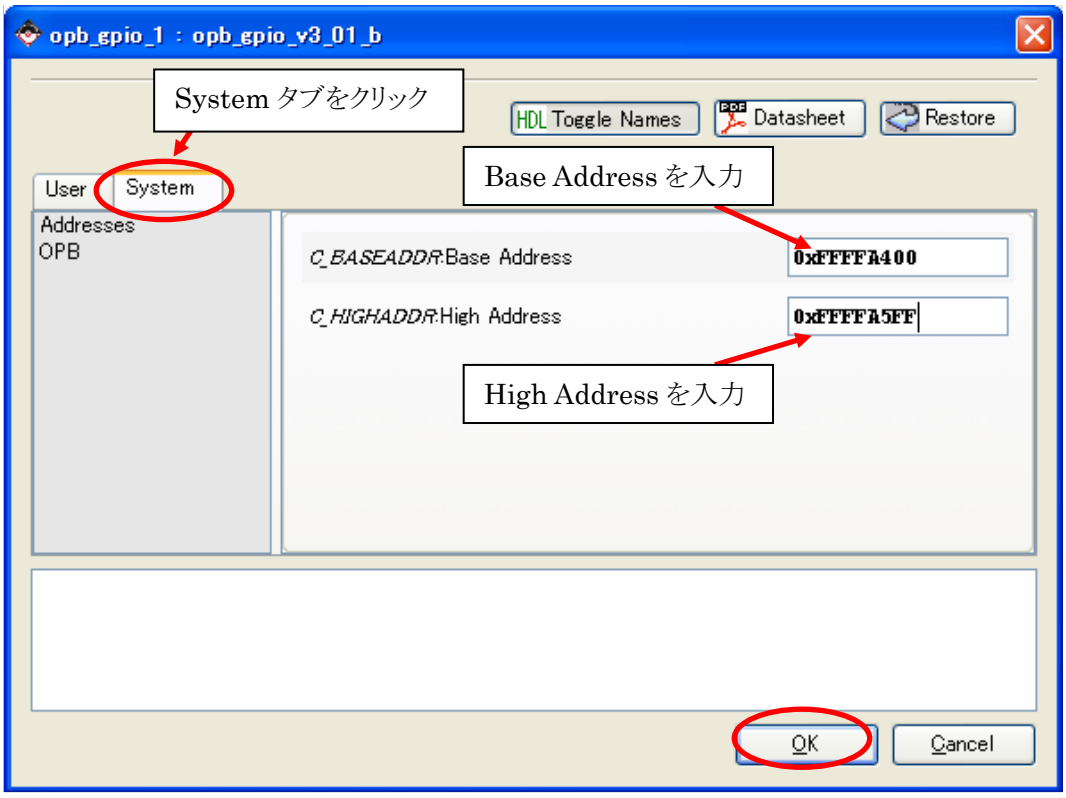


図 11-12 メモリアドレス設定

IP コアの詳細を知りたい時は、[Datasheet]をクリックしてください。データシートが表示されます。

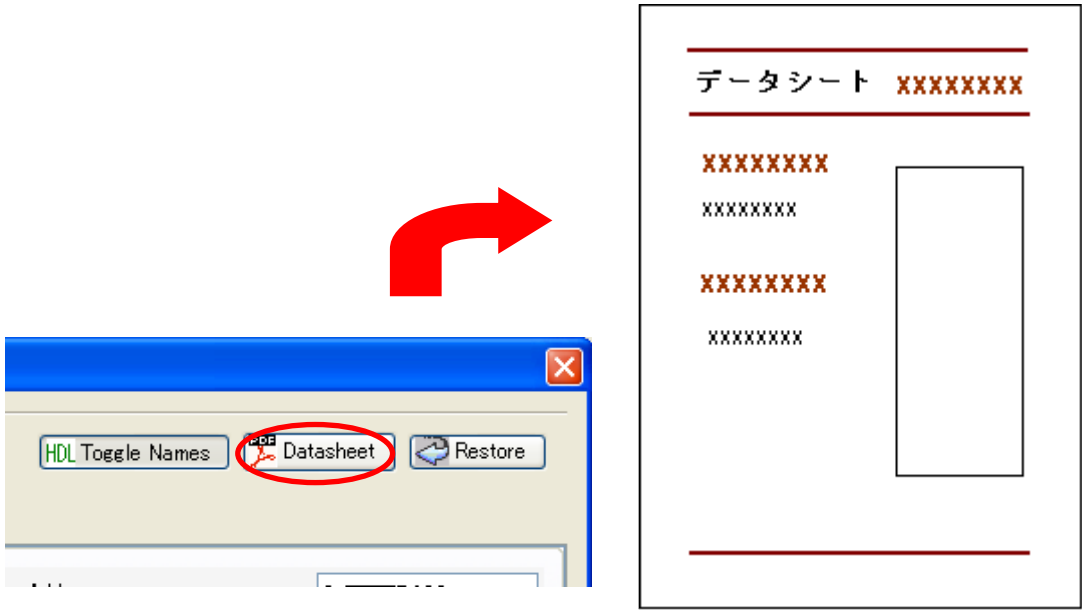


図 11-13 データシートの出し方

11.2.4. メモリマップ確認

Addresses を選択し、opb_gpio_1 の BaseAddress と High Address と Size に間違いがないか確認してください。

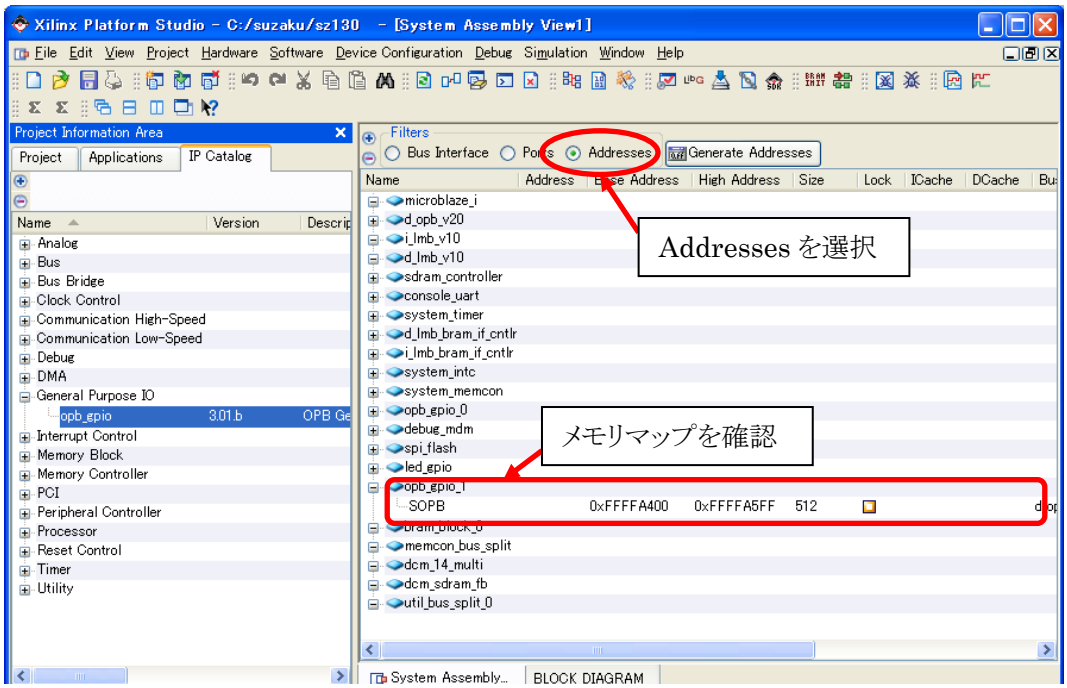


図 11-14 メモリマップ確認

11.2.5. 信号の定義

IP コアのバス以外への接続の指定をします。External Ports に登録すると、FPGA 外部信号を定義することができ、それ以外は内部信号になります。

Ports を選択し、opb_gpio_1 の⊞をクリックし開いてください。GPIO_d_out の Net の部分をクリックし、nLE と入力し、欄外をクリックして確定させてください。

図 11-15 Net 名入力

もう一度nLE の Net の部分をクリックし、今度は▼をクリックし、[Make External]を選択し、欄外をクリックして確定させてください。

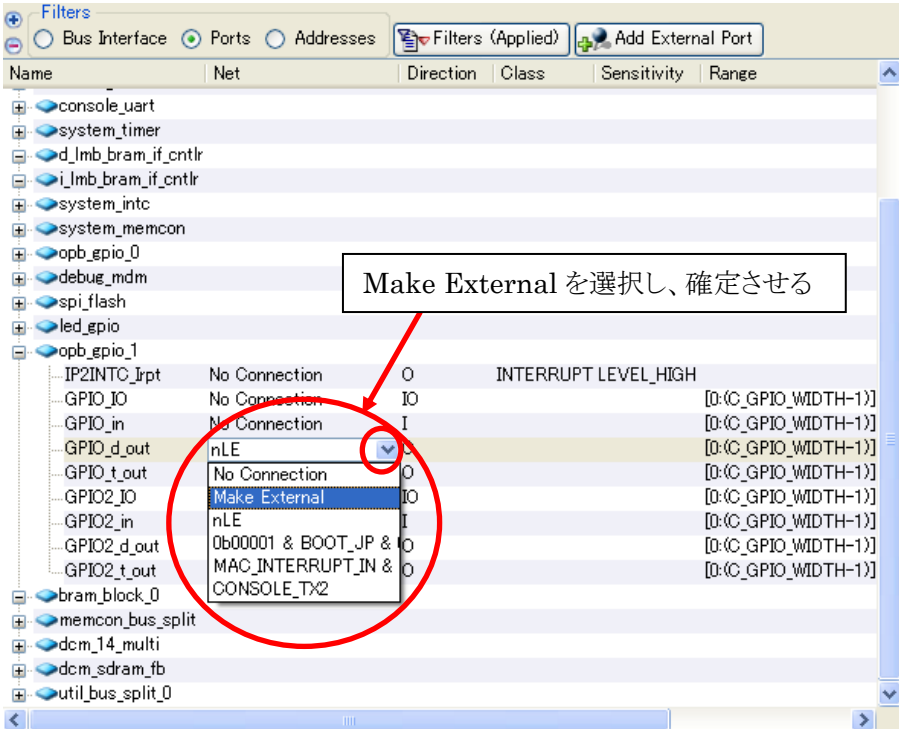


図 11-16 外部信号にする

External Ports の⊞をクリックして開いてください。External Ports にはシステムの外部に出力する信号が定義されています。この中に Name：nLE_pin という信号が出来上がっているので、nLE_pin をクリックし、名前を nLE に変更してください。これで外部出力信号 nLE が定義されます。

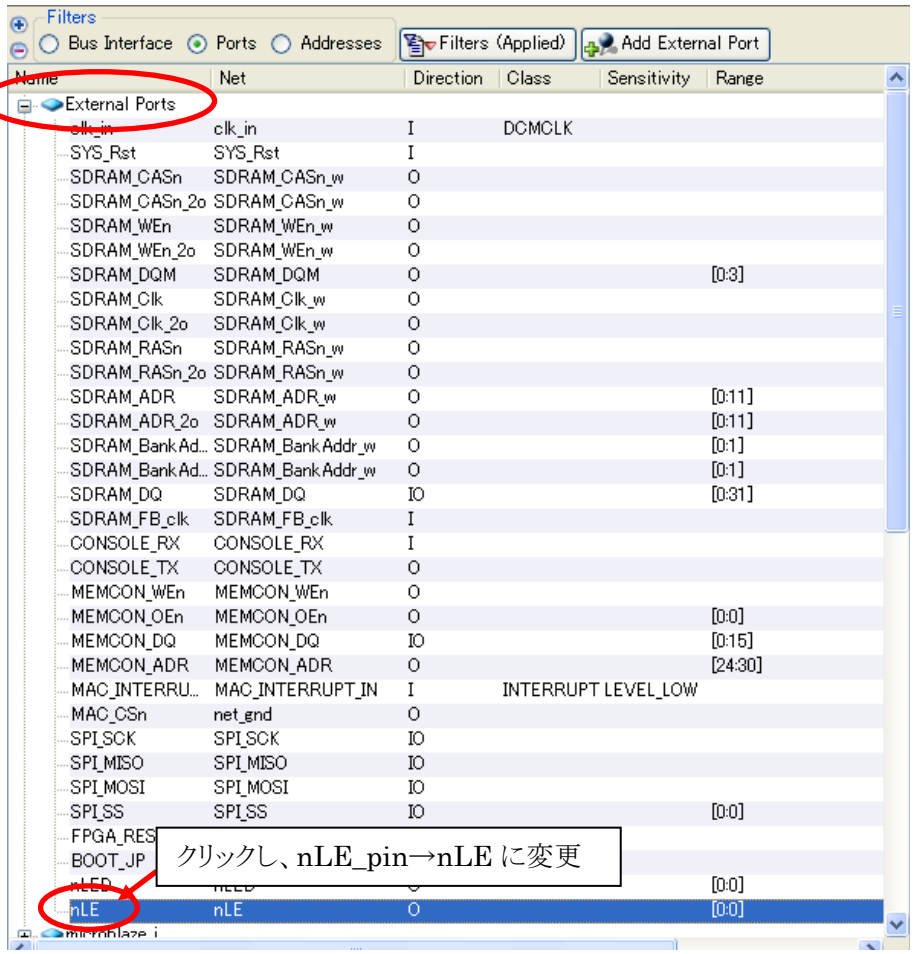


図 11-17 信号名変更

11.2.6. ピンアサイン

Project タブをクリックし、UCF File: data/xps_proj.ucf をダブルクリックして開いてください。ピンアサイン設定のファイルが開きます。単色 LED(D1)を点灯させるため、FPGA に信号を割り当てます。バス幅が 1 ビットでも nLE<0>のように、記述します。記述できたら[File]→[Save]をクリックし、保存してください。

表 11-2 nLE<0> ピンアサイン

	SZ010 SZ030	SZ130	SZ310
nLE<0>	B12	E12	L16

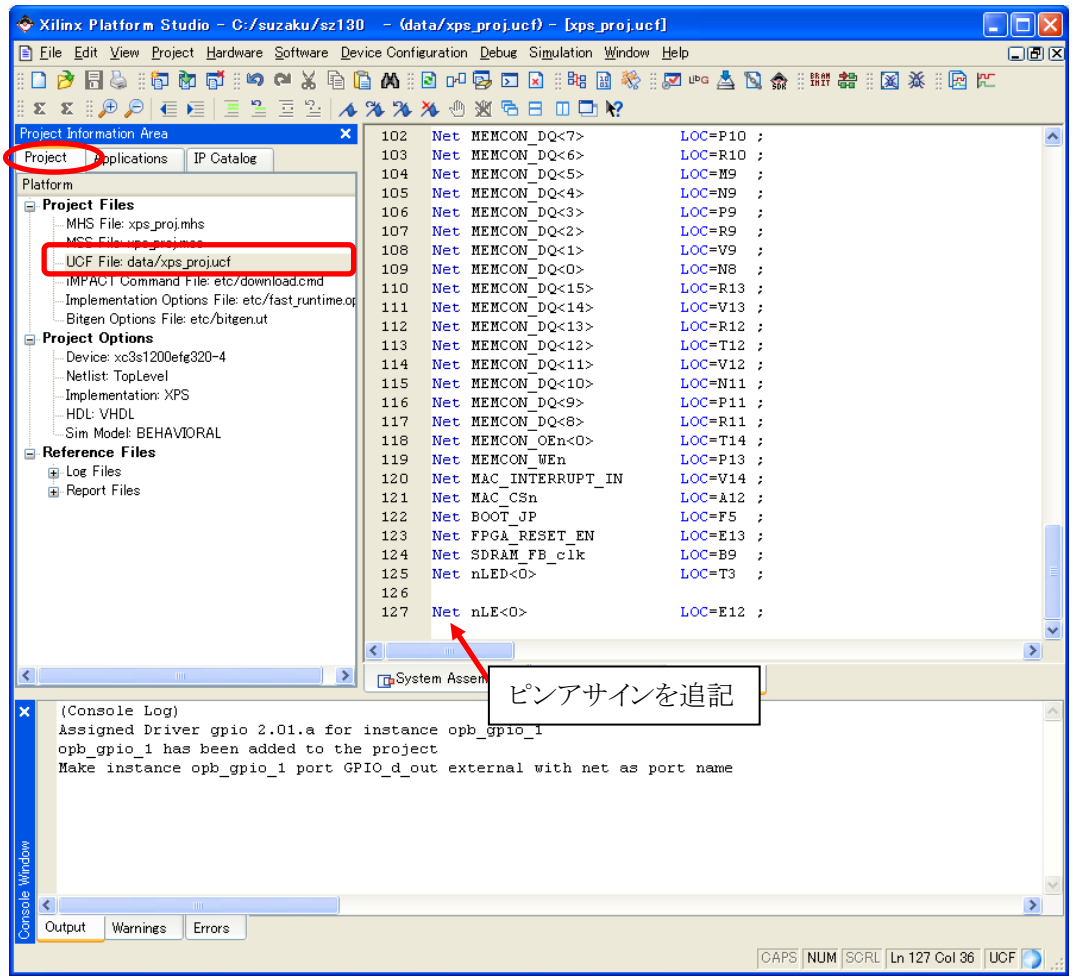
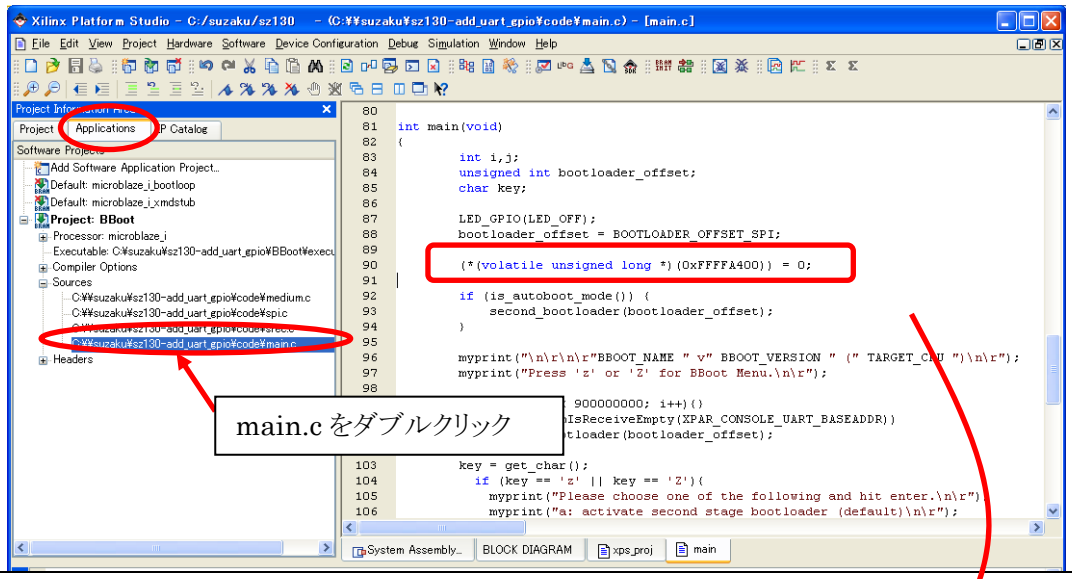


図 11-18 GPIO(xps_proj.ucf)

11.2.7. BBoot のソース編集

SUZAKU のデフォルトでは FPGA 内部の BRAM にファーストブートローダ、BBoot（Block RAM Boot）が入っています。この BBoot の中身を編集し、単色 LED を光らせる一文を追加します。

Applications タブをクリックしてください。Project:BBoot の Sources の main.c をダブルクリックして開いてください。main 文の中に以下の一文を追加してください。




```
--前略
int main(void)
{
int main(void)
{
    unsigned int bootloader_offset;
    char    key;
    int     i;

    LED_GPIO(LED_OFF);

    (*(volatile unsigned long *)(0x********)) = 0;

--後略
```

********には先ほど設定した
Base Address を記述する

図 11-19 単色 LED 点灯のソースコード追加(main.c)

追加できたら[File]→[Save]を選択し、保存してください。

11.2.8. bit ファイル作成

[Device Configuration]→[Update Bitstream]をクリックしてください。エラーがなければネットリストの生成と、配置配線が行われ、bit ファイルが生成されます。エラーがでたら間違いを修正して再び[Update Bitstream]をクリックしてください。

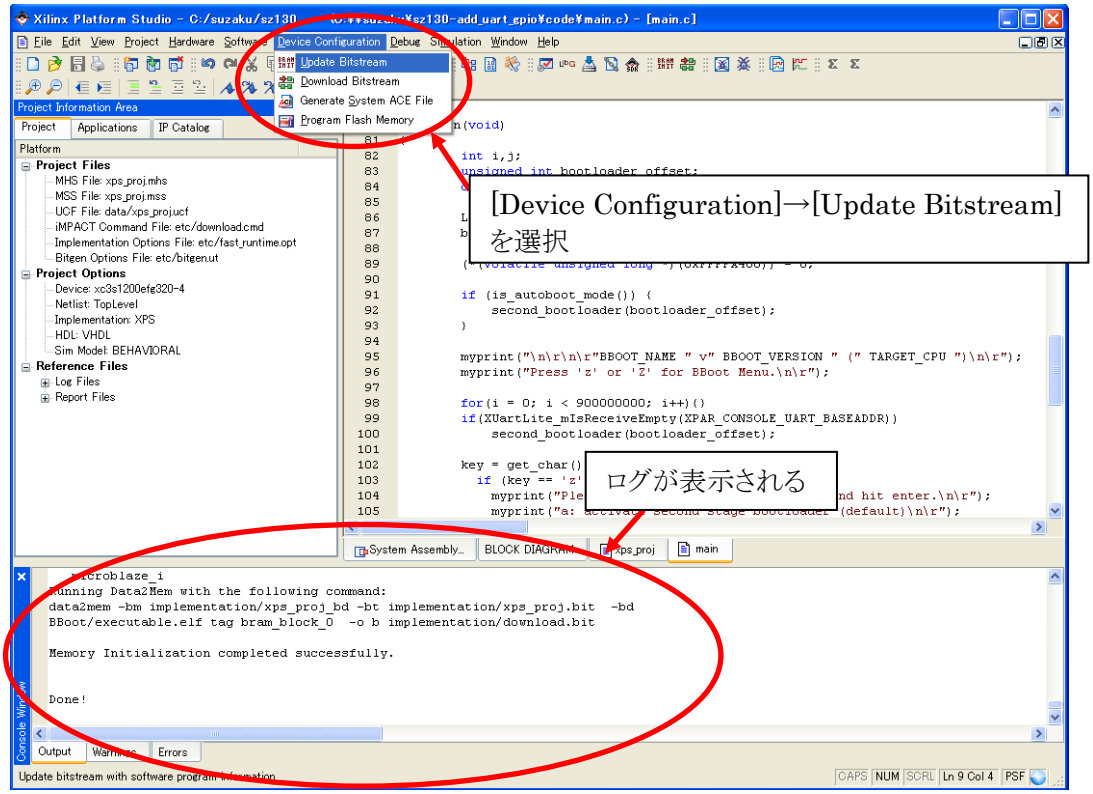


図 11-20 bit ファイル作成

11.2.9. コンフィギュレーション

bit ファイルを JTAG でコンフィギュレーションします。SUZAKU JP2 をショートさせ、SUZAKU CON7 にダウンロードケーブルを接続し、LES/SW CON6 に AC アダプタ 5V を接続し、電源を投入してください。

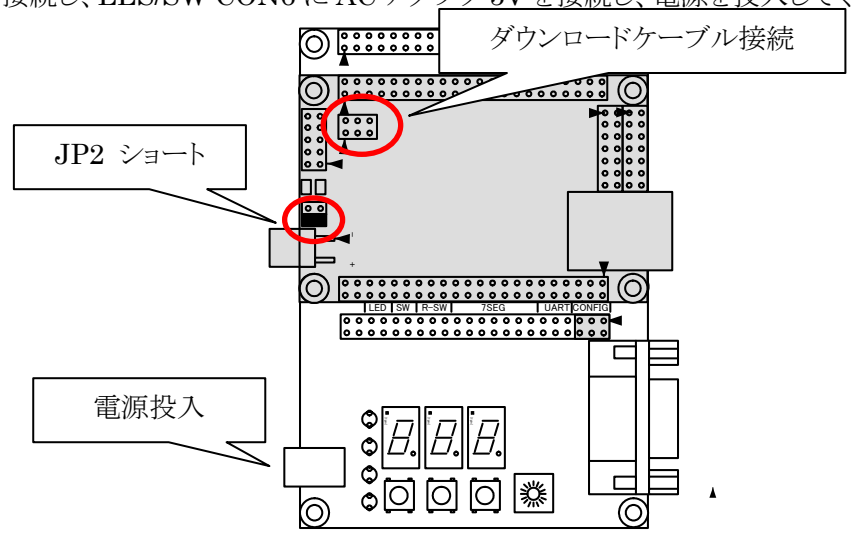


図 11-21 ジャンパの設定等

[Device Configuration]→[Download Bitstream]をクリックしてください。バッチモードの iMPACT を使用して FPGA に bit ファイルがコンフィギュレーションされます。

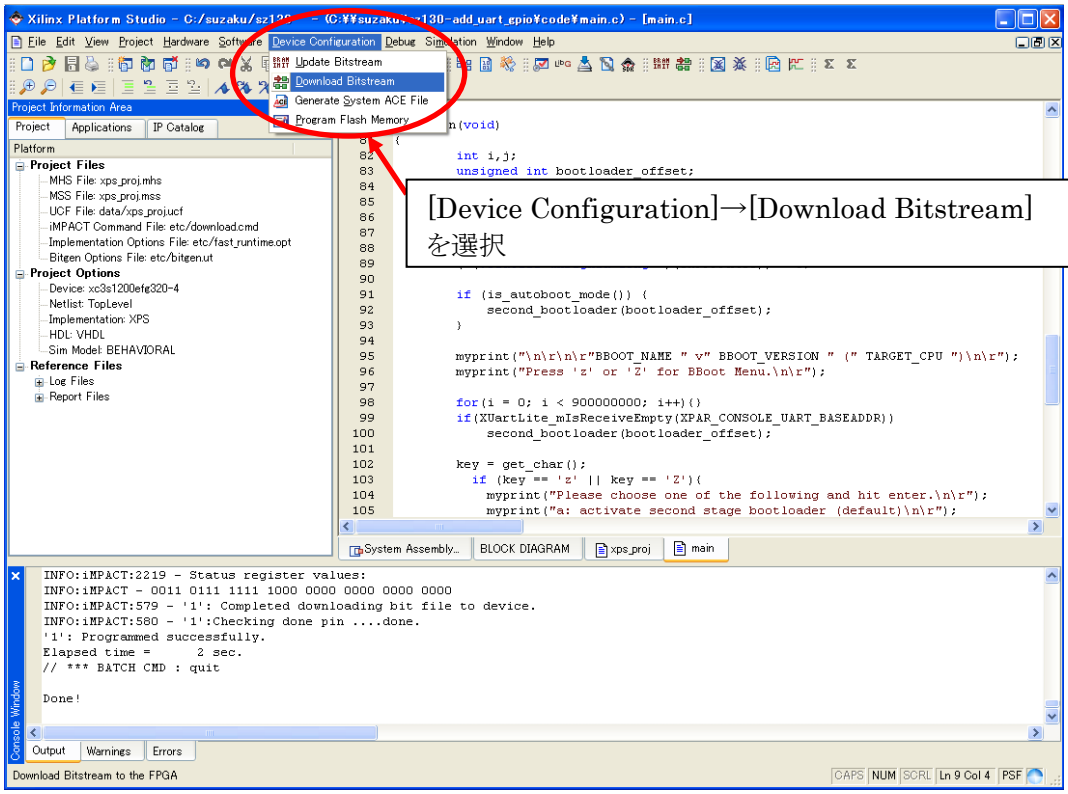


図 11-22 コンフィギュレーション

単色 LED(D1)が光ったでしょうか？

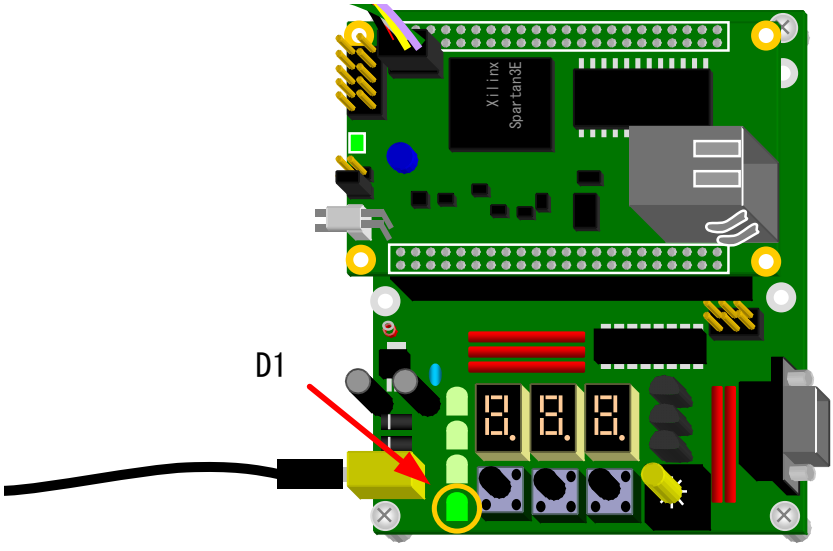


図 11-23 単色 LED(D1)点灯

Flash に書き込みたい場合は、”C:¥suzaku¥sz***-add_uart_gpio¥implementation”フォルダ内に download.bit が出来上がっているので、これを使って下さい。

11.2.10. 空きピン処理

ISE の時と同様、D2、D3、D4 が少し光っています。EDK で空きピン処理をしたい場合、UCF ファイルを編集します。PULLUP、PULLDOWN 等を指定することができます。

図 11-24 EDK での空きピンの処理

```
NET "port_name" {PULLUP | PULLDOWN};
```

11.2.11. Flat View

以下のような表示になって元に戻したいと思ったことはないでしょうか。右クリックしてメニューを出して Flat View のチェックをはずせば、元の表示に戻すことができます。

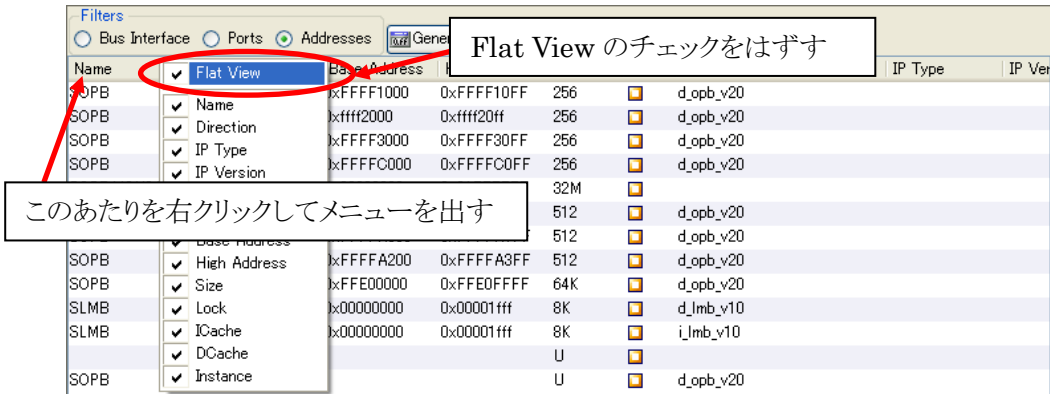


図 11-25 Flat View

11.3. UART の追加

UART を追加し、BRAM の中のソフトウェアに受信した文字をそのまま送信するソースコードを追加します。

11.3.1. IP コアの追加

IP Catalog のタブをクリックし、Communication Low-Speed の中にある opb_uartlite を右クリックしてメニューを出し、Add IP を選択してください。opb_uartlite_0 が追加されます。

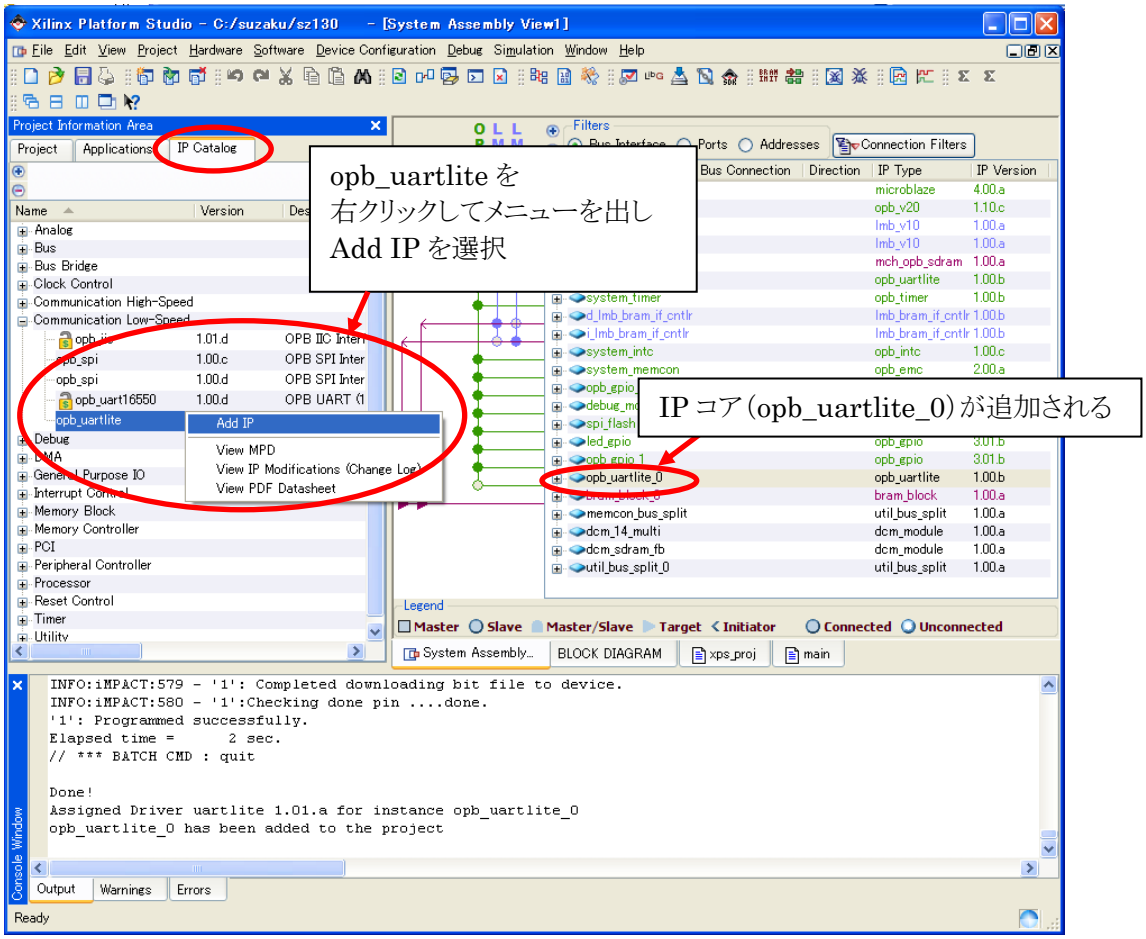


図 11-26 opb_uartlite の追加

11.3.2. OPB バスに接続

Bus Interface を選択し、opb_uartlite_0 の横の丸をクリックしてください。○ → ●
これで OPB バスに接続されます。

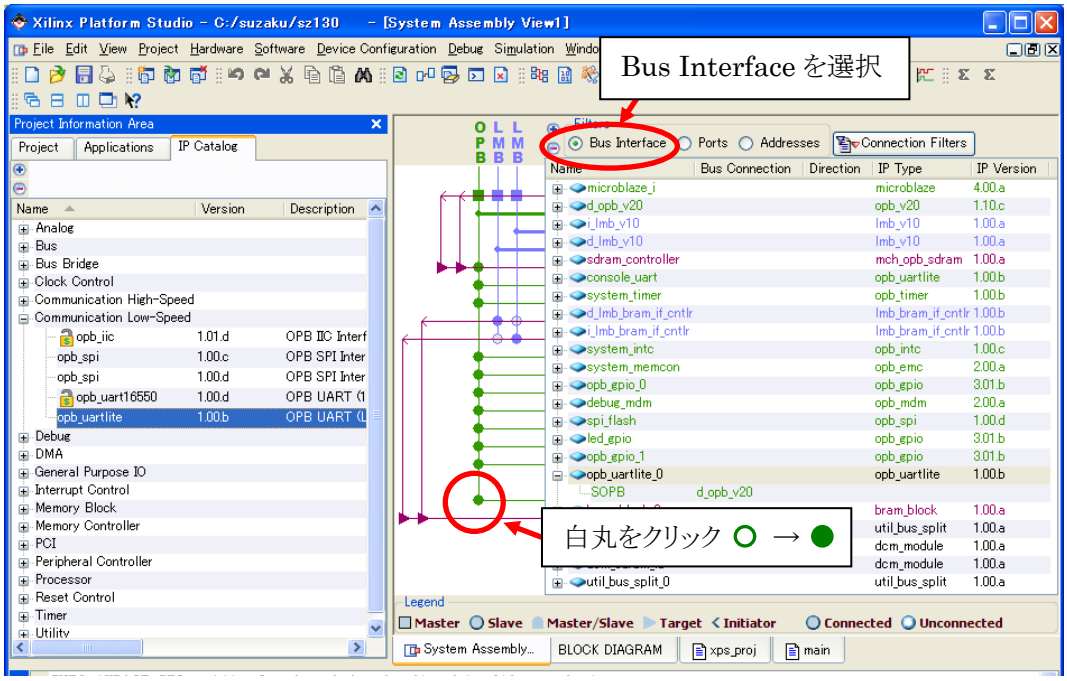


図 11-27 OPB バスに接続

11.3.3. IP コアの設定

opb_uartlite_0 を右クリックしてメニューを出し、Configure IP を選択してください。

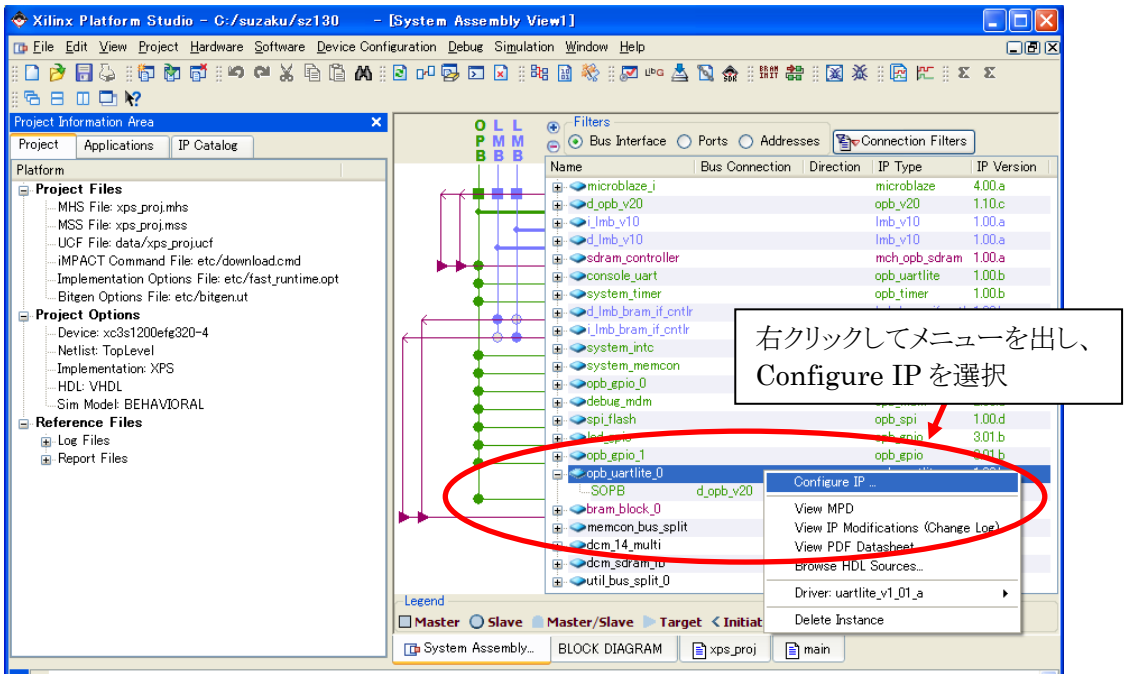


図 11-28 Configure IP

以下の設定にしてください。(”表 4-3 シリアルコンソールの設定” 参照)

・UART Lite Baud Rate	115200
・Number of Data Bits in a Serial Frame	8
・Use Parity	FALSE
・Parity Type	ODD

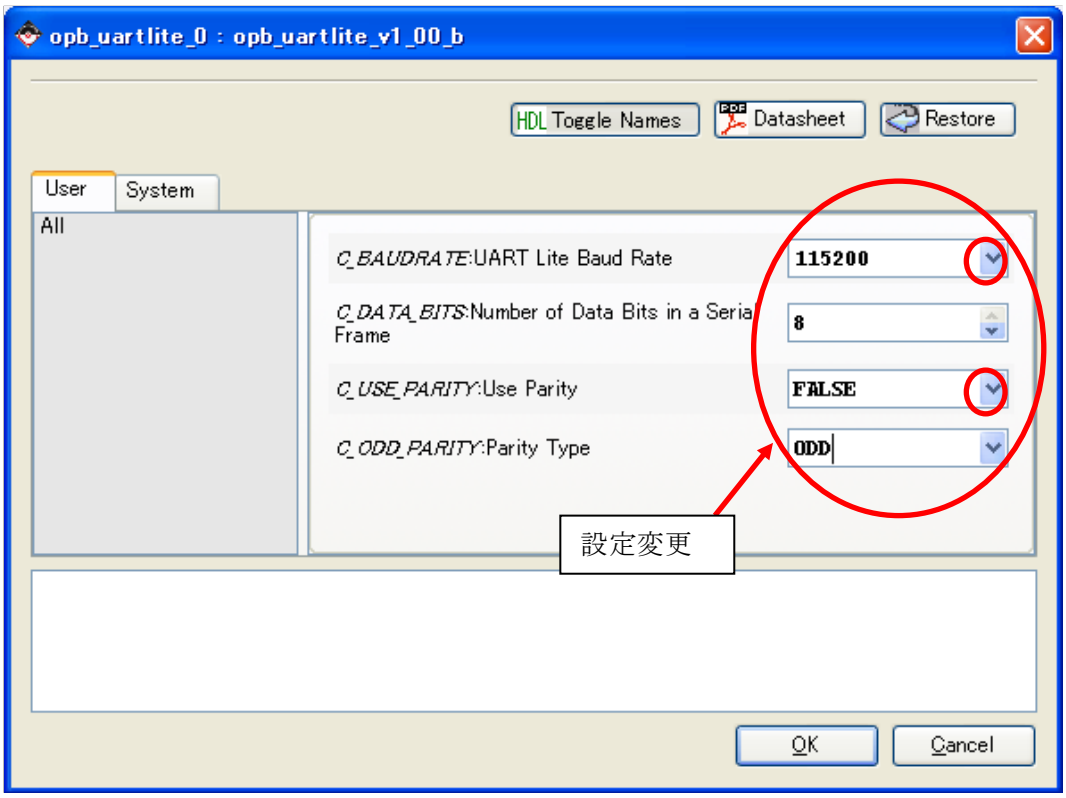


図 11-29 UART 設定変更

[System]タブをクリックし、[Base Address]に0xFFFFA600、[High Address]に0xFFFFA6FFと入力してください。メモリアドレスはSUZAKUのメモリマップでFreeと書いてあるところに割り当てます。(”5.3 SUZAKUメモリマップ”参照)

表 11-3 UART メモリアドレス

	SZ010、SZ030 SZ130	SZ310
Base Address	0xFFFFA600	0xF0FFA600
High Address	0xFFFFA6FF	0xF0FFA6FF

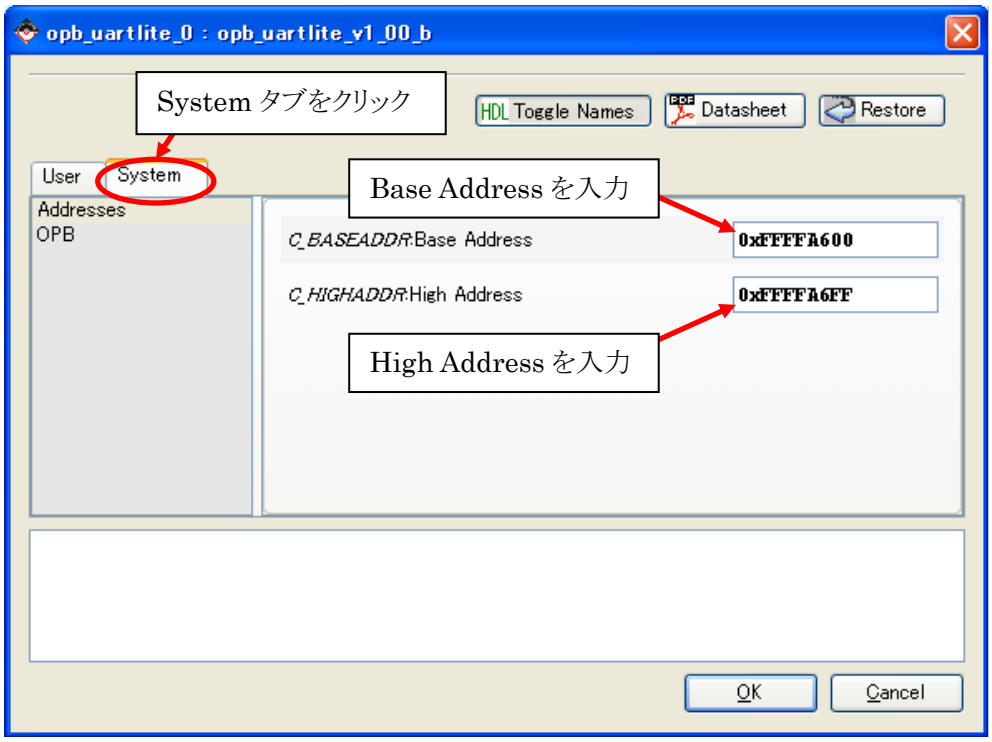


図 11-30 メモリアドレス設定

OPB をクリックし、[OPB Clock Frequency]にクロック周波数を入力して、[OK]をクリックしてください。

表 11-4 OPB Clock Frequency

	SZ010、SZ030 SZ130	SZ310
OPB Clock Frequency	51609600Hz	66355200Hz

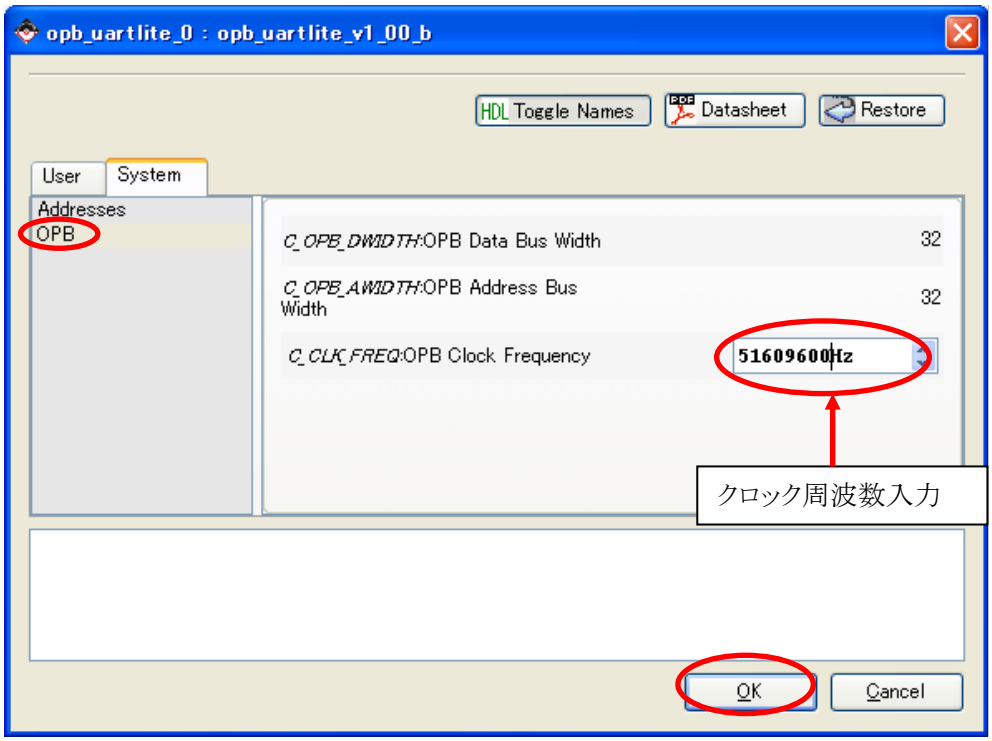


図 11-31 クロック周波数の設定

11.3.4. メモリマップ確認

Addresses を選択し、opb_uartlite_0 の BaseAddress と High Address と Size に間違いがないか、確認してください。

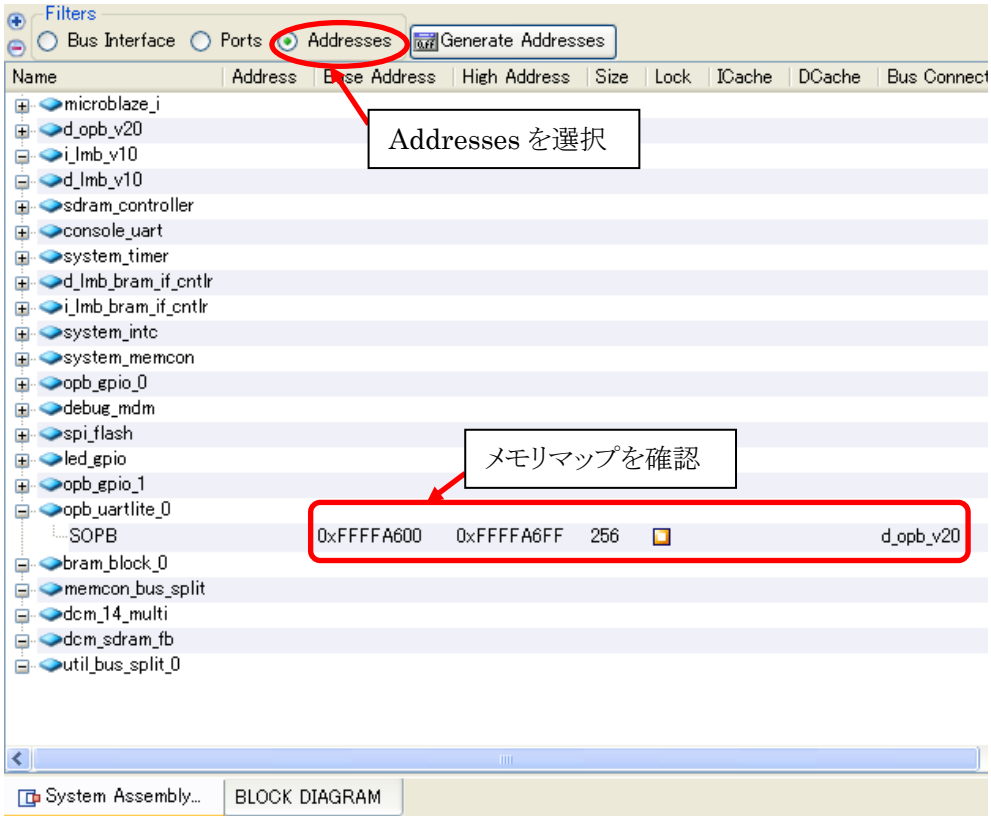


図 11-32 メモリマップ確認

11.3.5. 信号の定義

Ports を選択してください。opb_uartlite_0 の RX と TX の Net に名前をつけて確定し、Make External を選択して確定してください。

External Ports に信号が定義されるので、Name を変更してください。

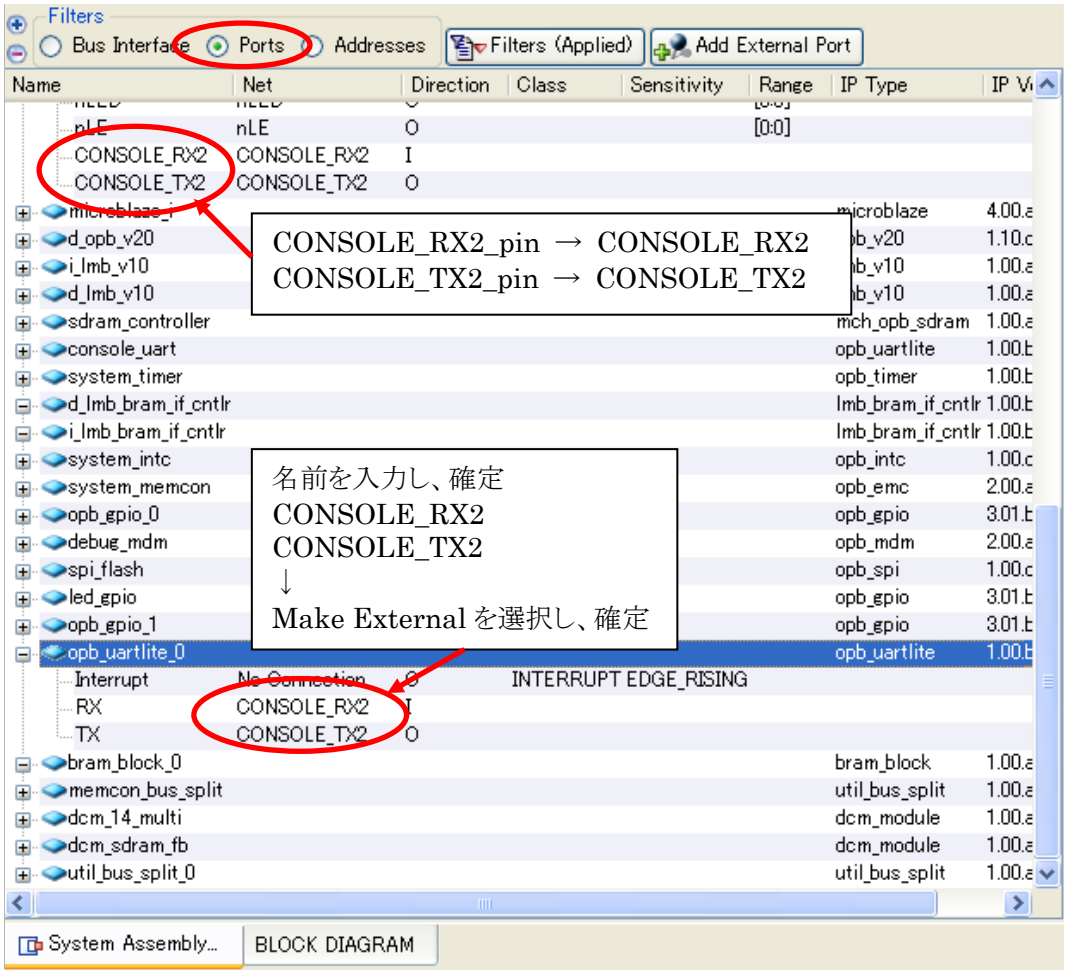


図 11-33 信号の定義

11.3.6. ピンアサイン

Project タブをクリックし、UCF ファイルを開き、増えた 2 ピンを追加し、保存してください。

表 11-5 nLE0 ピンアサイン

	SZ010 SZ030	SZ130	SZ310
CONSOLE_RX2	B4	M3	F13
CONSOLE_TX2	A3	M6	E13

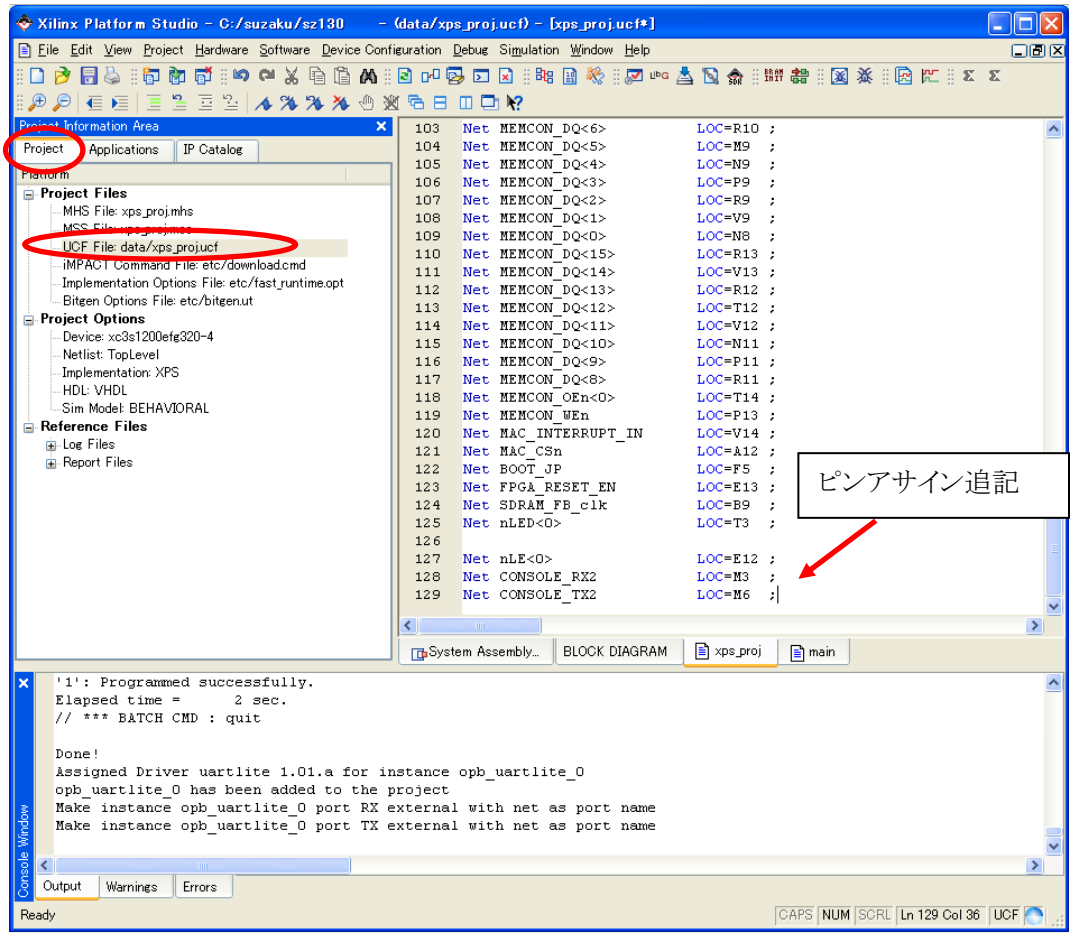
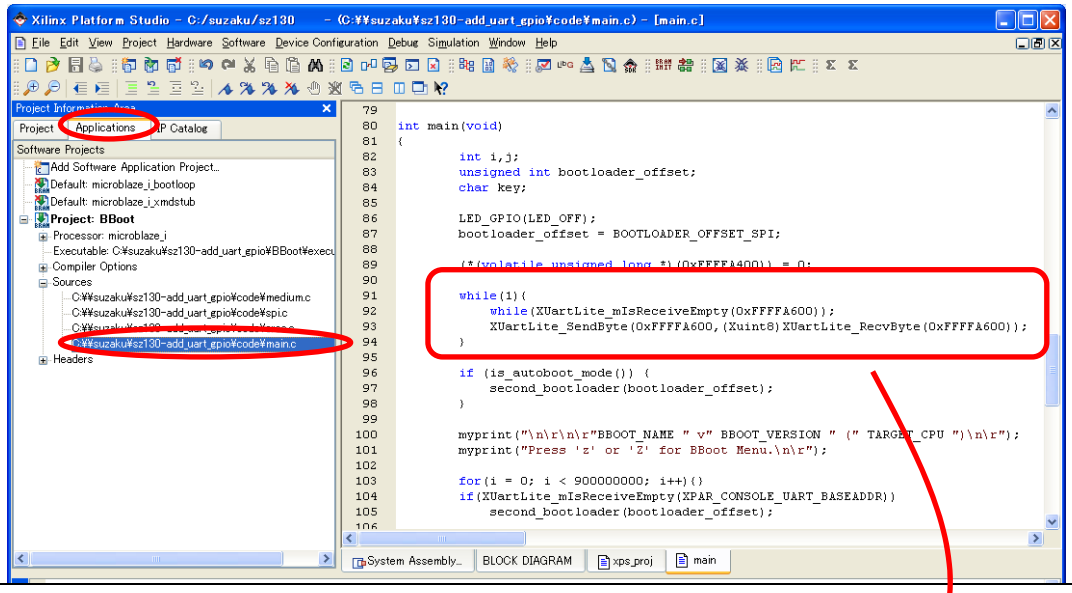


図 11-34 UART(xps_prj.ucf)

11.3.7. BBoot のソース編集

Applications タブをクリックし、main.c を開き、先ほど単色 LED を点灯するコードを追加した下に、受信した文字をそのまま送信するコードを追加し、保存してください。




```
--前略
int main(void)
{
int main(void)
{
    unsigned int bootloader_offset;
    char    key;
    int     i;

    LED_GPIO(LED_OFF);

    (*(volatile unsigned long *)(0x********)) = 0;

  while (1) {
      while(XUartLite_mIsReceiveEmpty(0x********));
      XUartLite_SendByte(0x********,(Xuint8)XUartLite_RecvByte(0x********));
  }

--後略
```

********には先ほど設定した Base Address を記述する

図 11-35 送受信ソースコード追加(main.c)

11.3.8. bit ファイルの作成、コンフィギュレーション

Update Bitstream をクリックしてください。エラーがなければネットリストの生成と、配置配線が行われ、bit ファイルが生成されます。bit ファイルは download.bit という名前で”C:¥suzaku¥sz***-add_uart_gpio¥implementation”フォルダに出来上がります。

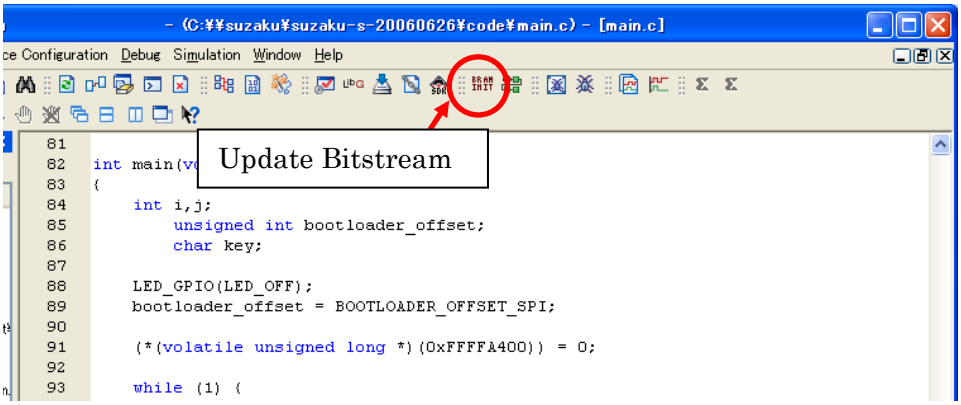


図 11-36 bit ファイルの作成

シリアル通信用ソフトウェアを立ち上げ、シリアル通信の設定を行ってください。(”表 4-3 シリアルコンソールの設定” 参照)

SUZAKU JP2 をショートし、SUZAKU CON7 にダウンロードケーブルを接続してください。

LED/SW CON7 にシリアルケーブルを接続してください。

最後に LED/SW CON6 に AC アダプタ 5V を接続し、電源を投入してください。

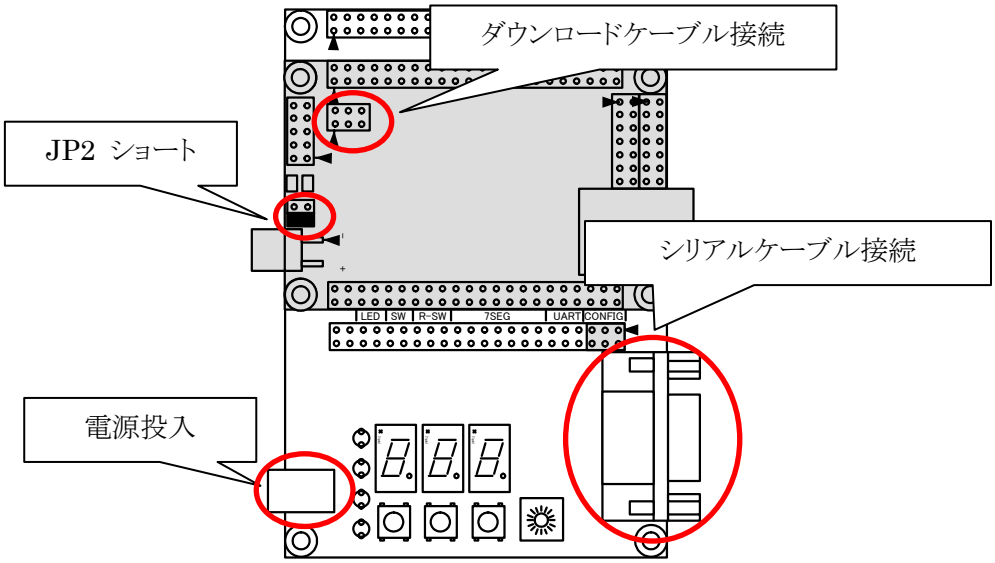


図 11-37 ジャンパの設定等

Download Bitstream をクリックしてください。バッチモードの iMPACT を使用して FPGA に bit ファイルがコンフィギュレーションされます。

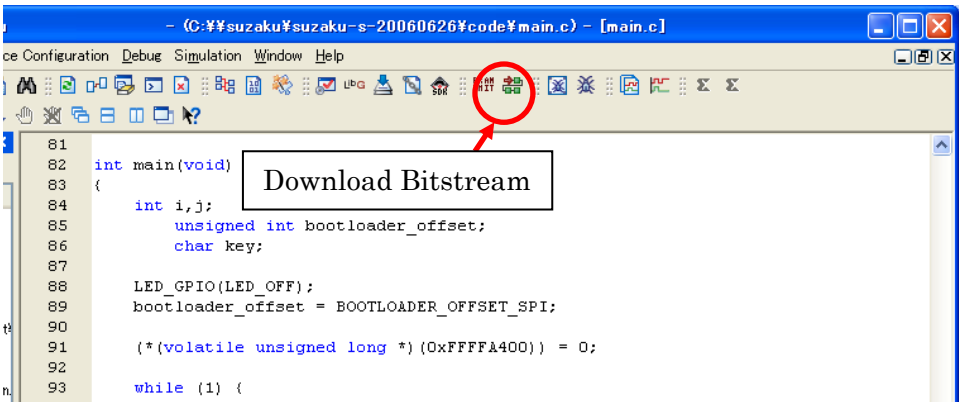


図 11-38 bit ファイルの作成

キーボードから何か文字を打ち込んでください。

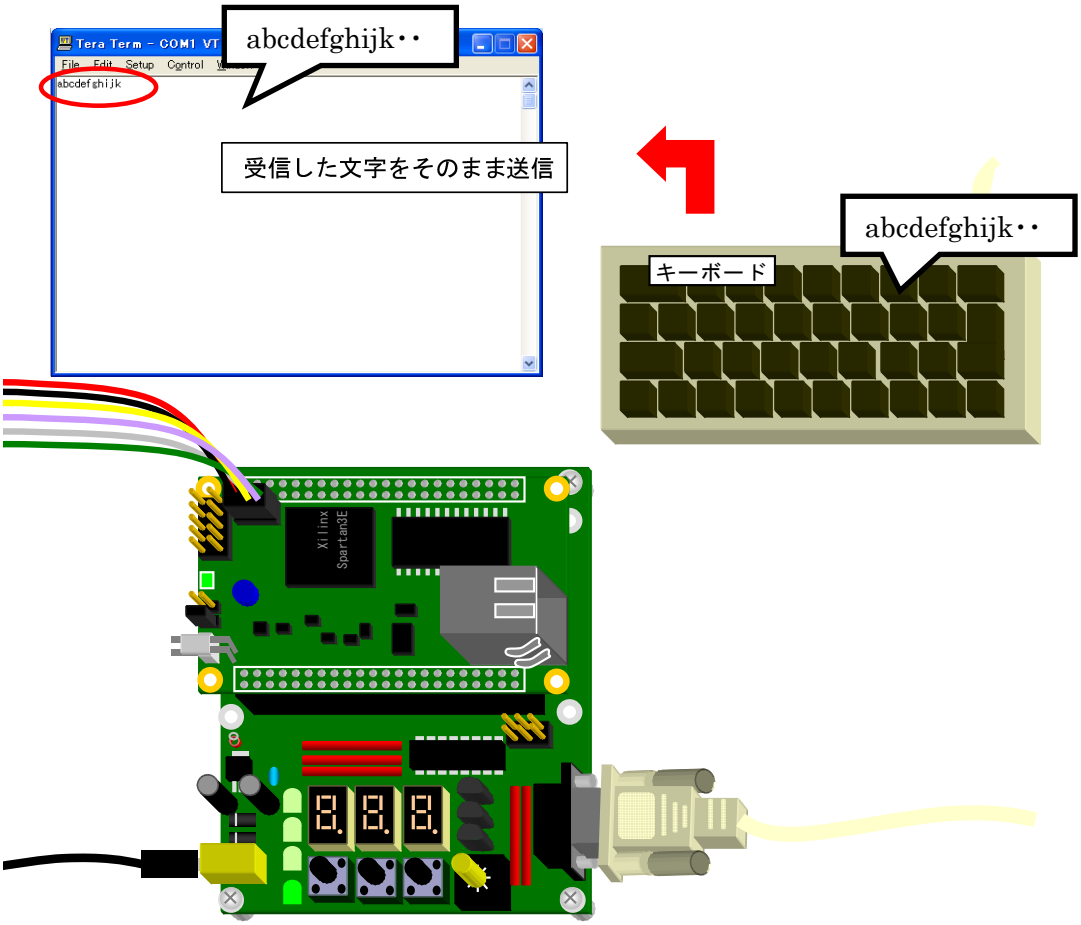


図 11-39 シリアル通信 動作確認

12. スロットマシンのコアを CPU で制御する

“10. FPGA 入門 スロットマシン製作”で作った回路を少し改造してコアにして、SUZAKU のデフォルトの回路に接続し、スロットマシンを作り上げます。作業手順は以下の通りです。

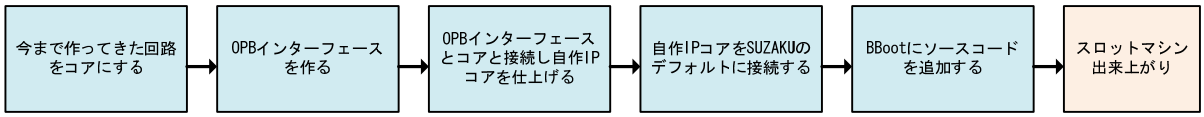


図 12-1 スロットマシンへの道のり

付属 CD-ROM の”¥suzaku¥fpga_proj¥8.1i¥sz***”の中の圧縮ファイル”sz***-xxxxxxxx.zip”(*は型式、x は更新日)をハードディスクに展開してください。ここでは展開後のフォルダを”C:¥suzaku”の下にコピーして作業を進めます。

”C:¥suzaku¥sz***-xxxxxxxx”の中の”xps_proj.xmp”をダブルクリックして開いてください。

Xilinx Platform Studio が起動し、SUZAKU のデフォルトが開きます。

12.1. 今まで作ってきた回路をコアにする

下図の仕様でコアを作成します。sil00u_core.vhd を新規作成してください。

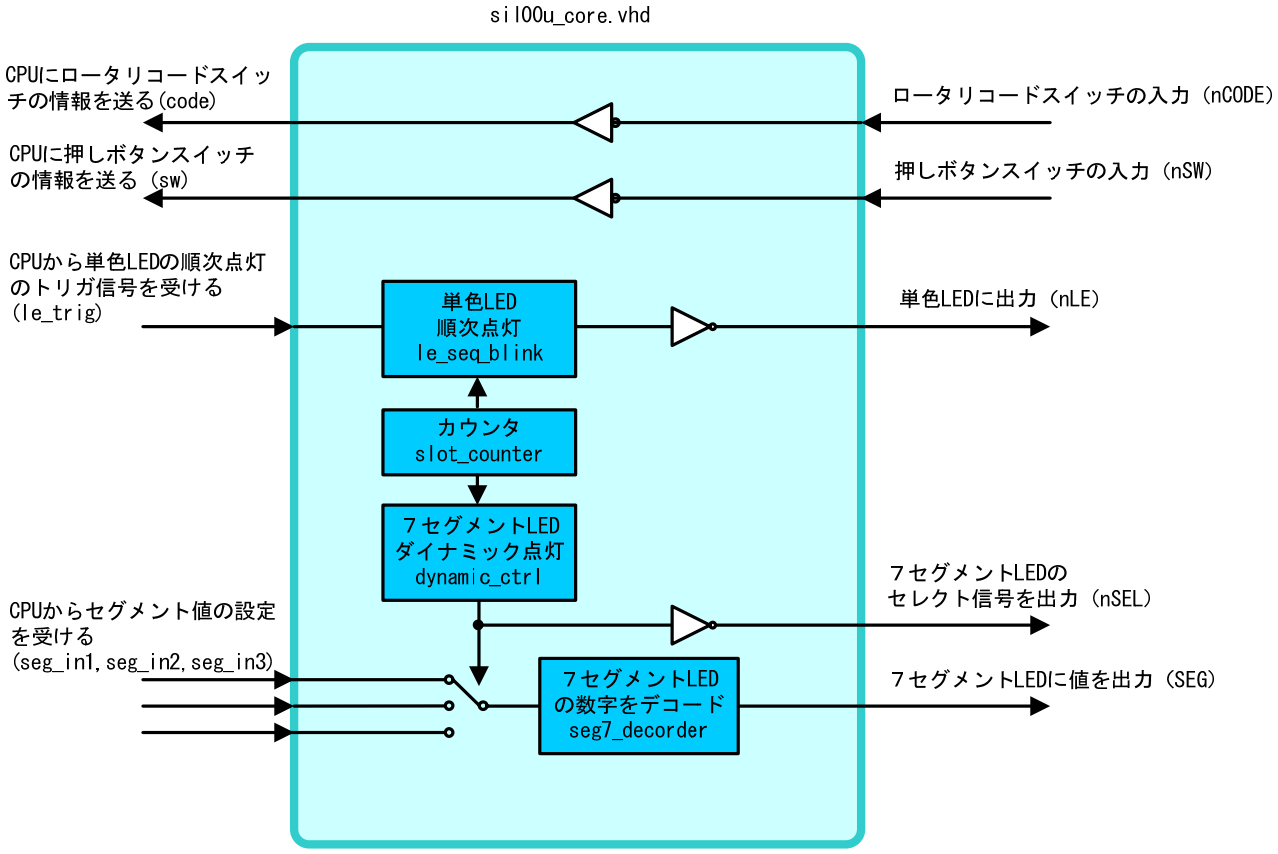


図 12-2 自作 IP コア(ソフト版)の仕様

■ sil00u_core.vhd

SUZAKU ではクロック 3.6864MHz を DCM で逓倍にしています。SZ010、SZ030、SZ130 ではクロックが 51.6096MHz、SZ310 では 66.3552MHz になっています。カウンタのビット数を4bit 増やし 23bit にします。sil00u_core.vhd を上位階層として今まで作った回路、slot_counter、le_seq_blink、seg7_decorder、dynamic_ctrl 回路を呼び出します。

例 12-1 コア(sil00u_core.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity sil00u_core is
    generic (
        C_CNT_WIDTH : integer := 23 --カウンタのビット幅
    );

    Port (
        SYS_CLK: in STD_LOGIC; --クロック信号
        SYS_RST : in STD_LOGIC; --リセット信号

-- External
    SEG : out STD_LOGIC_VECTOR(0 to 7); --7 セグ LED にダイナミック点灯で値を出力
    nSEL : out STD_LOGIC_VECTOR(0 to 2); --7 セグ LED にセレクトをを出力
    nLE : out STD_LOGIC_VECTOR(0 to 3); --単色 LED に順次点灯を出力
    nSW : in STD_LOGIC_VECTOR(0 to 2); --押しボタンスイッチを入力
    nCODE : in STD_LOGIC_VECTOR(0 to 3); --ロータリコードスイッチを入力

-- Register Write
    seg_in1 : in STD_LOGIC_VECTOR(0 to 3); --CPU からセグメント値の設定を受ける
    seg_in2 : in STD_LOGIC_VECTOR(0 to 3); --CPU からセグメント値の設定を受ける
    seg_in3 : in STD_LOGIC_VECTOR(0 to 3); --CPU からセグメント値の設定を受ける
    le_trig : in STD_LOGIC; --CPU から単色 LED の順次点灯のトリガ信号の設定を受ける

-- Register Read
    sw : out STD_LOGIC_VECTOR(0 to 2); --CPU に押しボタンスイッチの情報を送る
    code : out STD_LOGIC_VECTOR(0 to 3);  --CPU にロータリコードスイッチの情報を送る
    intr : out STD_LOGIC  --カウンタの出力を割り込みコントローラに送る
        );
end sil00u_core;

architecture IMP of sil00u_core is
        signal count : STD_LOGIC_VECTOR(0 to C_CNT_WIDTH-1);
        signal le : STD_LOGIC_VECTOR(0 to 3);
        signal le_t : STD_LOGIC_VECTOR(0 to 3);
        signal seg_data : STD_LOGIC_VECTOR(0 to 3);

  component slot_counter
    generic (
        C_CNT_WIDTH : integer := C_CNT_WIDTH
    );
    Port (
        SYS_CLK : in STD_LOGIC; --クロック信号
        SYS_RST : in STD_LOGIC; --リセット信号
        count : out STD_LOGIC_VECTOR(0 to C_CNT_WIDTH-1) --カウンタ値
        );
  end component;
```

```
component le_seq_blink
   Port (
        SYS_CLK: in STD_LOGIC; --クロック信号
        SYS_RST : in STD_LOGIC; --リセット信号
        le : out  STD_LOGIC_VECTOR(0 to 3); --単色 LED 出力信号
        le_timing : in  STD_LOGIC --タイミング信号
        );
end component;

component dynamic_ctrl
    Port (
          SYS_CLK : in STD_LOGIC; --クロック信号
          SYS_RST : in STD_LOGIC; --リセット信号
          nSEL  : out STD_LOGIC_VECTOR(0 to 2); --7 セグメント LED セレクト信号(負論理)
          seg7_timing : in  STD_LOGIC; --7 セグタイミング信号
          seg_in1 : in  STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED1 の値
          seg_in2 : in  STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED2 の値
          seg_in3 : in  STD_LOGIC_VECTOR(0 to 3); --7 セグメント LED3 の値
          seg_data : out  STD_LOGIC_VECTOR(0 to 3) --4 ビットバイナリコード
          );
end component;

component seg7_decorder
    Port (
          SEG : out STD_LOGIC_VECTOR(0 to 7); --7 セグメント LED への出力信号
          seg_data : in STD_LOGIC_VECTOR(0 to 3) --4 ビットバイナリコード
          );
  end component;

begin
  slot_counter_0 : slot_counter
    Port map(
      SYS_CLK => SYS_CLK,
      SYS_RST => SYS_RST,
      count => count
      );

  le_seq_blink_0 : le_seq_blink
    Port map(
      SYS_CLK => SYS_CLK,
      SYS_RST => SYS_RST,
      le => le,
      le_timing => count(0)
      );

  dynamic_ctrl_0 : dynamic_ctrl
    Port map(
      SYS_CLK => SYS_CLK,
      SYS_RST => SYS_RST,
      nSEL => nSEL,
      seg7_timing => count(8),
      seg_in1 => seg_in1,
      seg_in2 => seg_in2,
      seg_in3 => seg_in3,
      seg_data => seg_data
      );

  seg7_decorder_0 : seg7_decorder
```

```
    Port map(
      SEG => SEG,
      seg_data => seg_data
      );

le_t <= le and "1111" when le_trig = '1' else "0000"; --トリガ信号が'1'の時順次点灯
nLE <= not le_t; --外部に出力
sw <= not nSW; --正論理にして入力
code <= not nCODE; --正論理にして入力
intr <= count(4); --カウンタの出力を割り込みコントローラに送る
end IMP;
```

SUZAKU のデフォルトに自作 IP コアを追加します。

● SZ010、SZ030 の場合

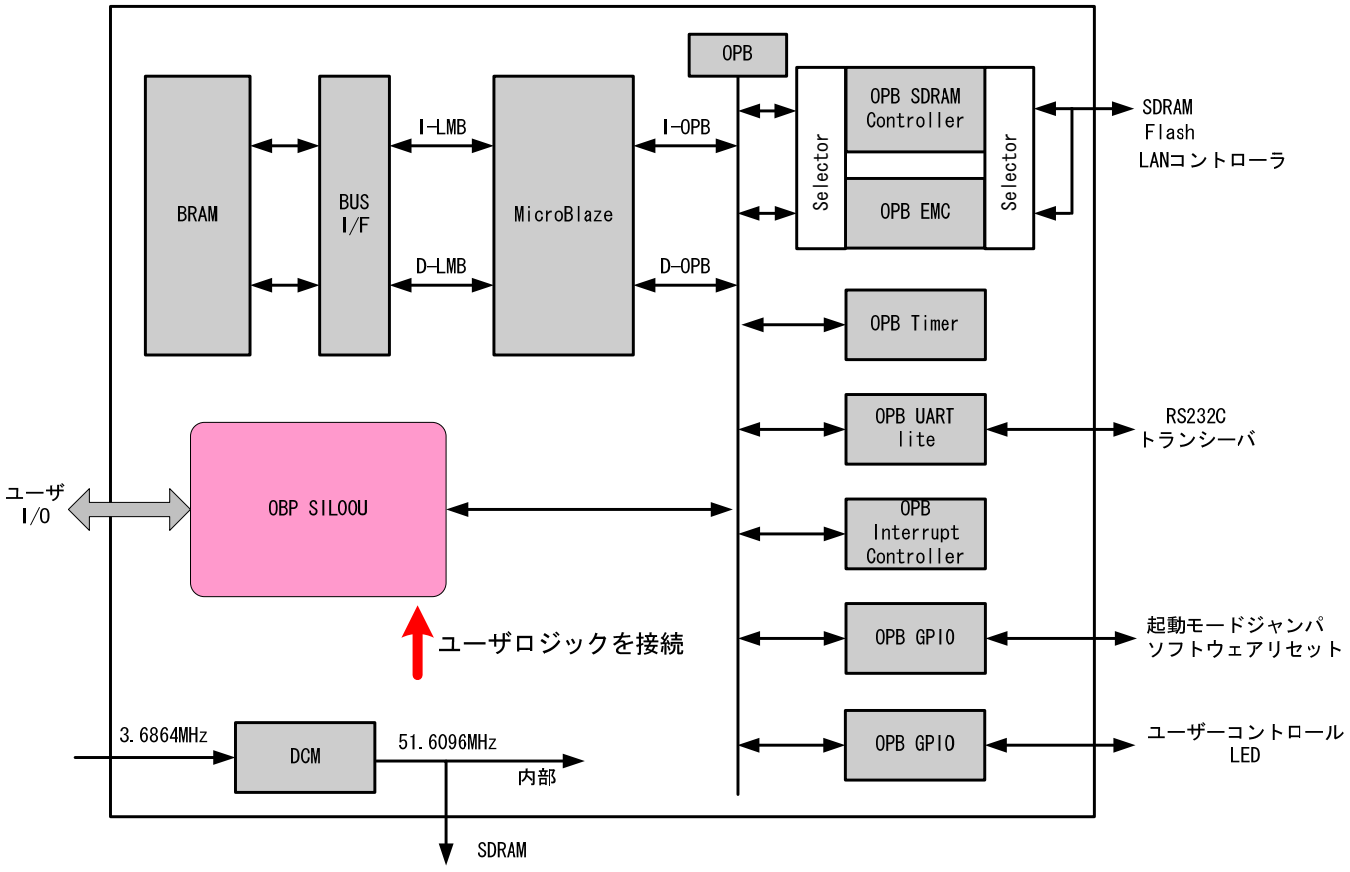


図 12-3 SZ010、SZ030 のデフォルトに自作 IP コアを追加

● SZ130 の場合

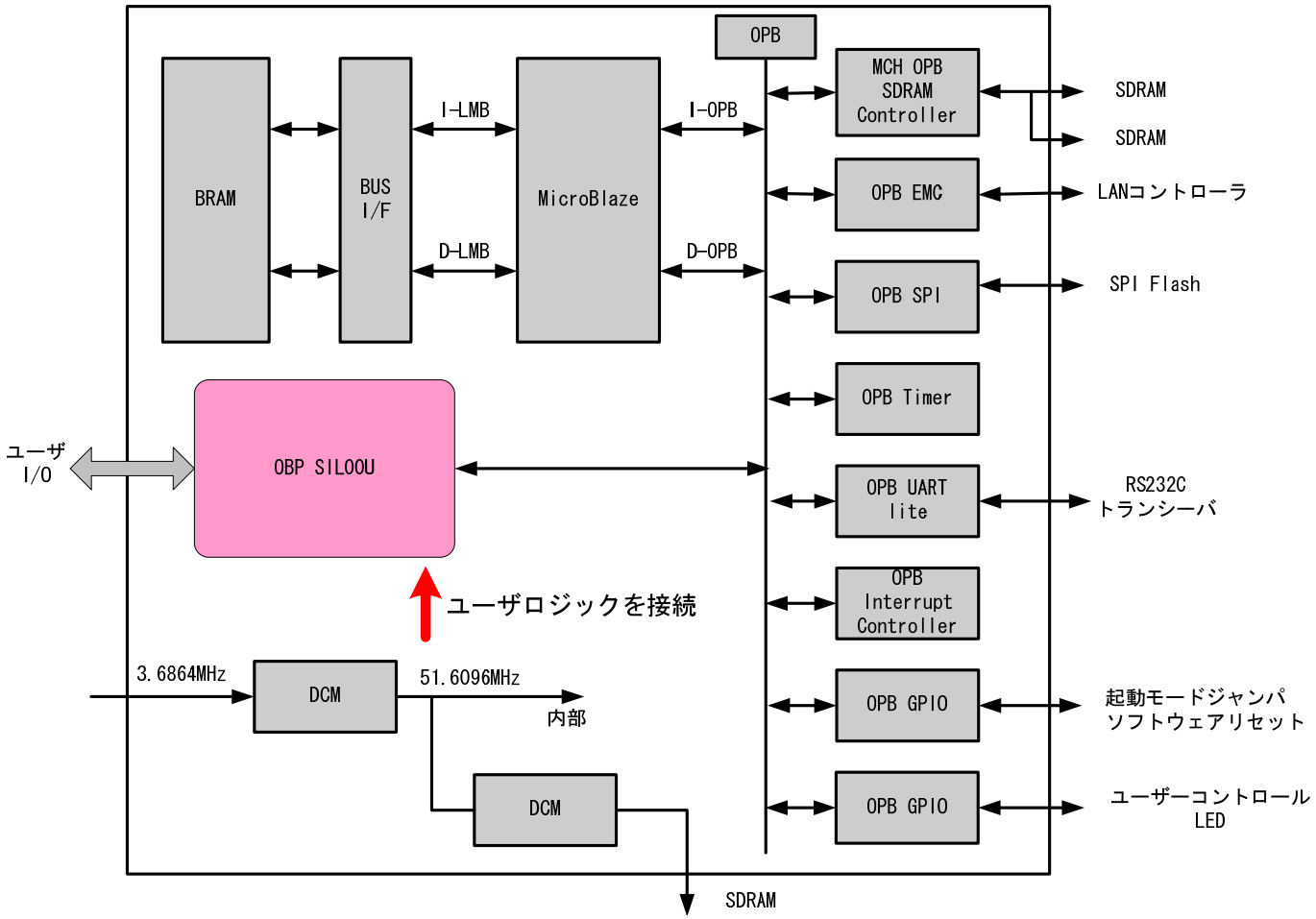


図 12-4 SZ130 のデフォルトに自作 IP コアを追加

● SZ310 の場合

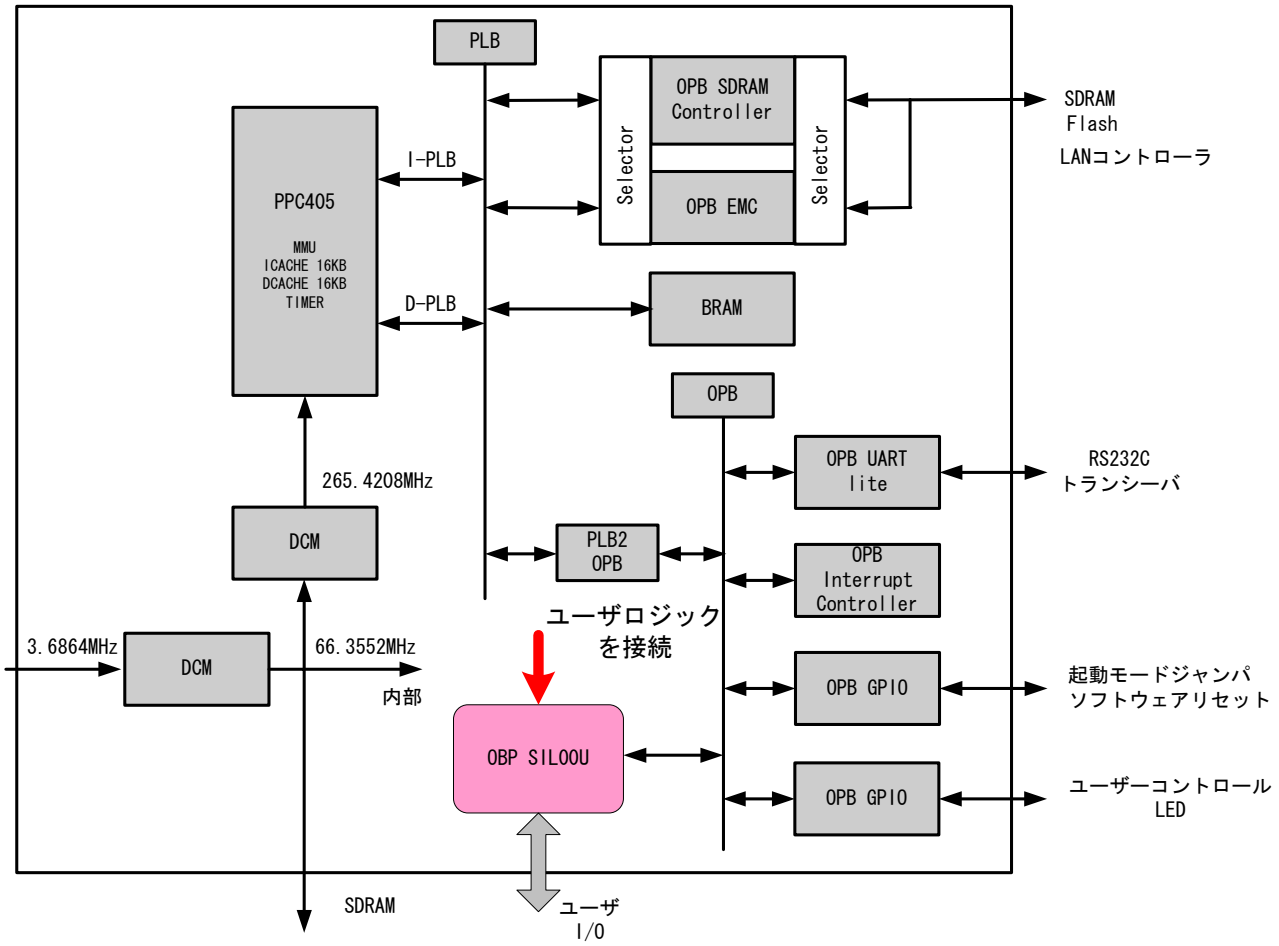


図 12-5 SZ310 のデフォルトに自作 IP コアを追加

12.2. ウィザードを使って OPB インターフェースをつくる

OPB バスにコアを追加するためのインターフェースを作ります。EDK にはインターフェースを作るウィザードが用意されています。

[Hardware]→[Create or Import Peripheral…]をクリックしてください。

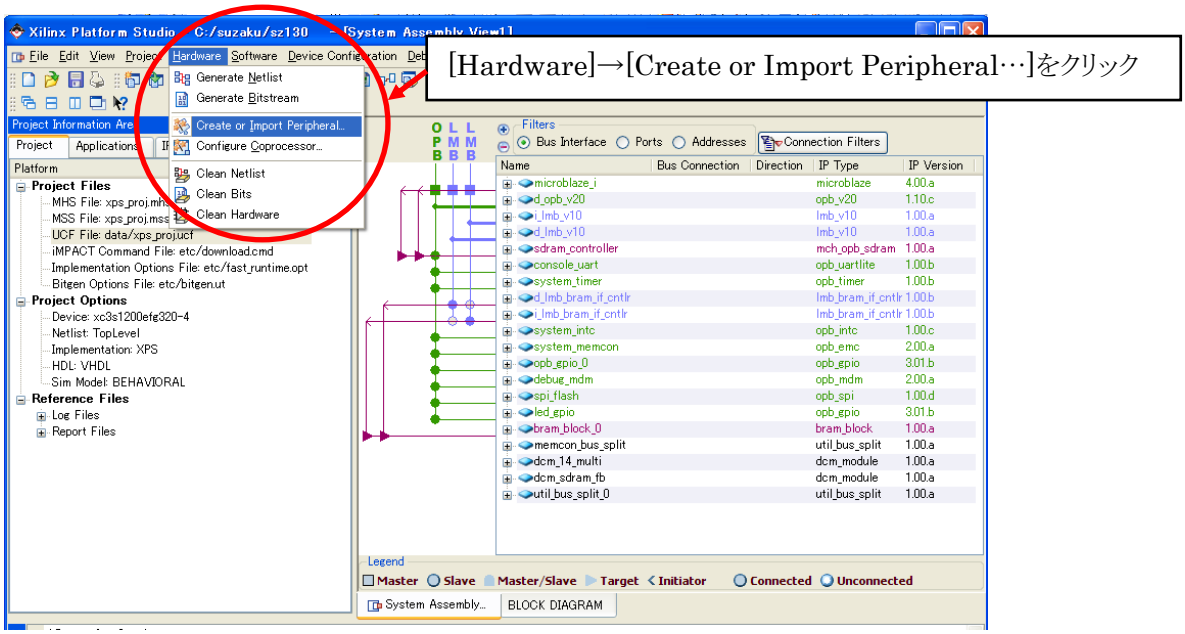


図 12-6 Create and Import Peripheral Wizard の起動のさせ方

Create and Import Peripheral Wizard が立ち上がります。[Next]をクリックして下さい。

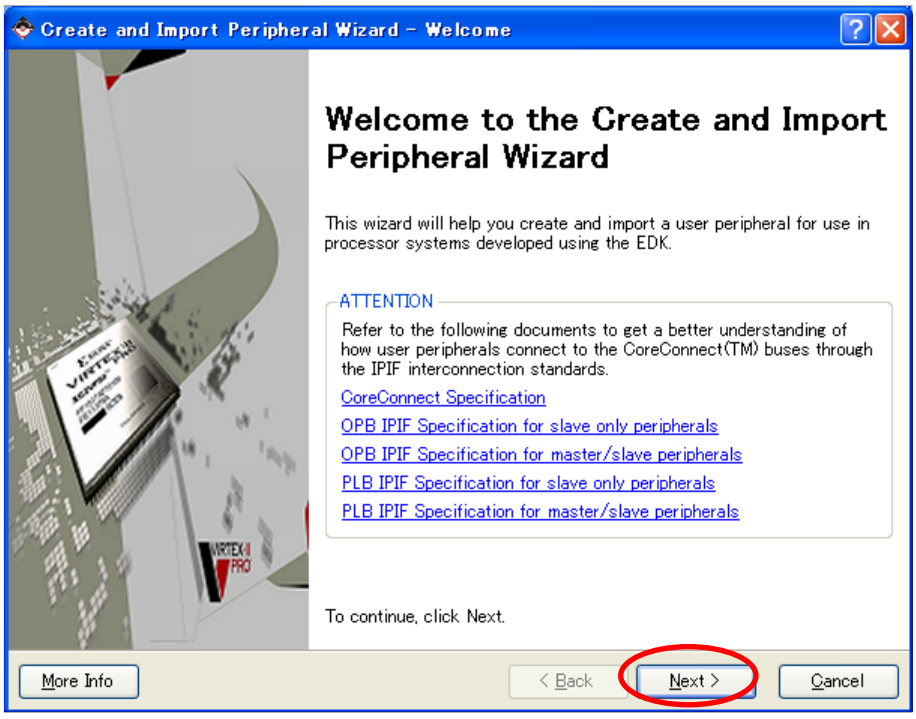


図 12-7 Create and Import Peripheral Wizard

Create and Import Peripheral Wizard が立ち上がります。新規で作るので Select flow の[Create templates for new peripheral]をチェックして[Next]をクリックしてください。

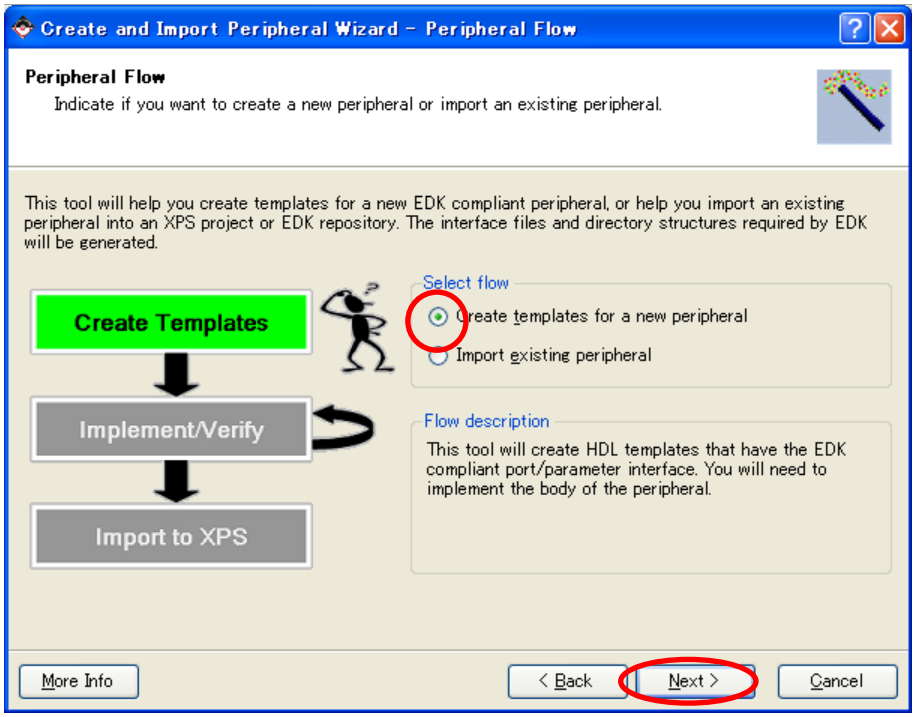


図 12-8 Peripheral Flow

コアを生成する場所を指定します。[To an XPS project]をチェックし、現在のプロジェクトの下に作成します。入力できたら[Next]をクリックして下さい。

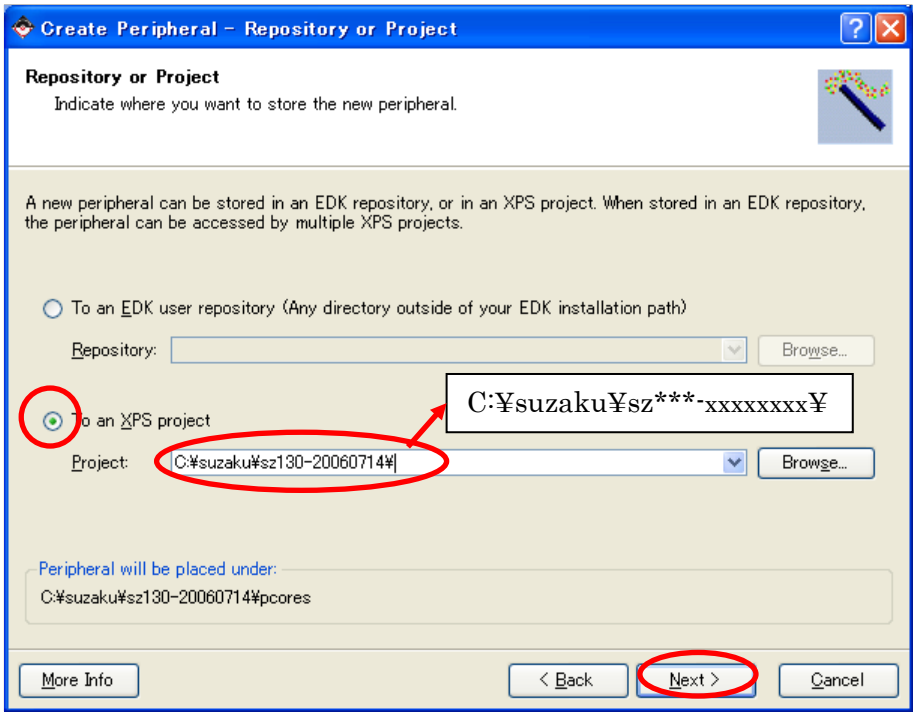


図 12-9 コアの生成場所の指定

コアに名前をつけます。[Name]に名前を入力してください。ここでは opb_sil00u とします。

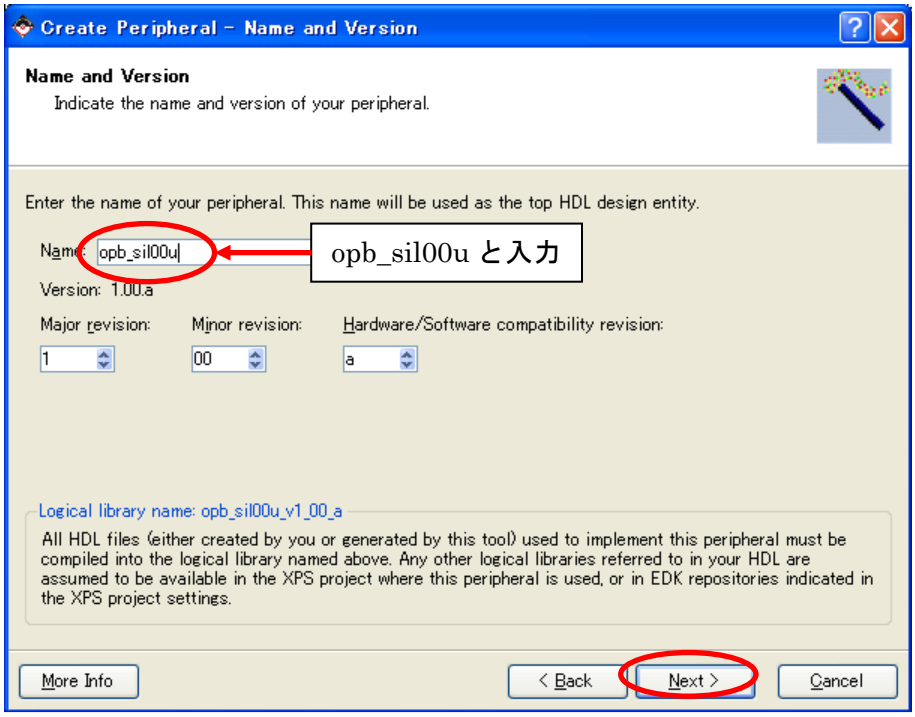


図 12-10 コアの名前

バスを選択します。OPB につなぐので[On-chip Peripheral Bus(OPB)]を選択して[Next]をクリックしてください。

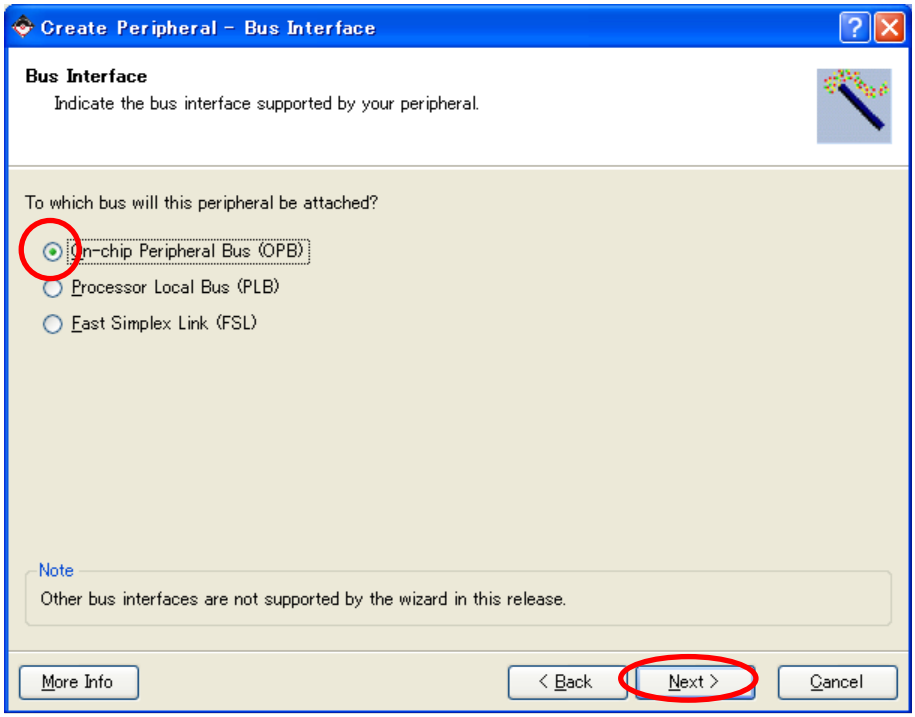


図 12-11 バスの選択

OPB Services でサービスを選択します。IPIF にはアドレスのデコード、バイト調整などの基本的な機能に加えて、ペリフェラルの作成を大幅に簡略化するオプションの機能が備わっています。選択したサービスに基づいて、OPB ペリフェラルテンプレートが生成されます。

今回は[User logic interrupt support]、[User logic S/W register support]を選択し、[Next]をクリックしてください。割り込みのユーザーテンプレート、ソフトウェアアクセス可能レジスタが生成されるユーザーテンプレートが追加されます。

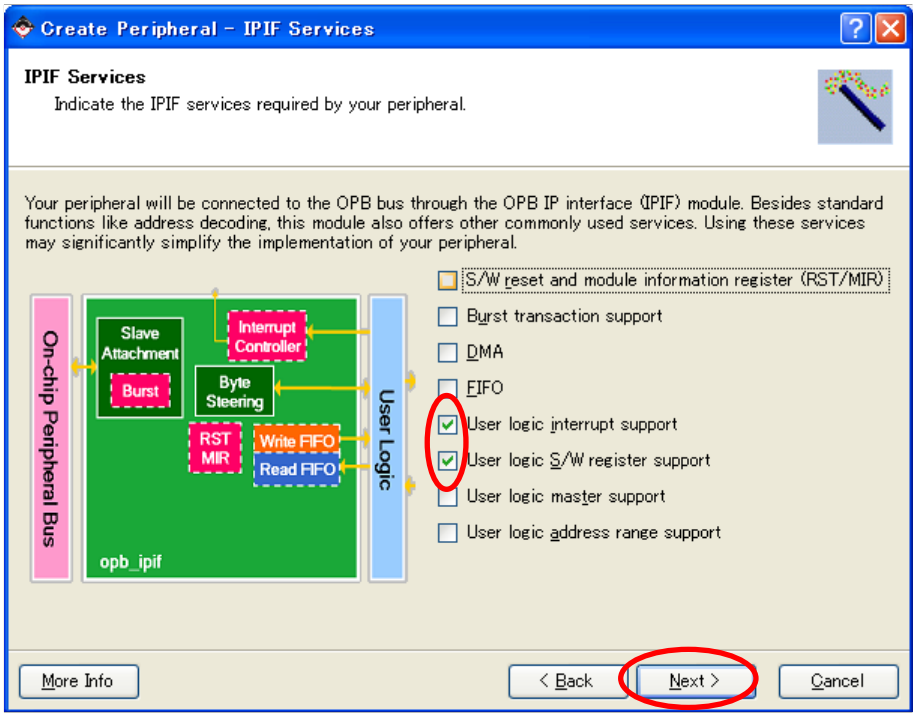


図 12-12 テンプレート追加

割り込みの設定をします。[Use Device ISC(interrupt source controller)のチェックボタンをはずし、Interrupt capture mode を[Rising Edge Detect]に設定して、[Next]をクリックして下さい。

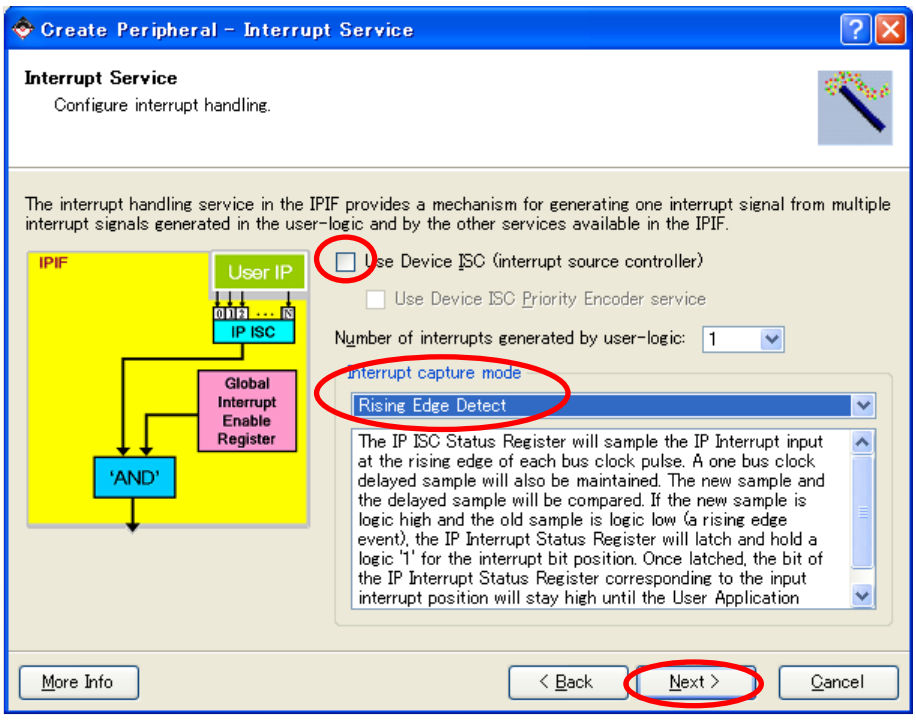


図 12-13 Interrupt 設定

ソフトウェアアクセス可能レジスタ数と、サイズ(バイト、ハーフワード、ワード、ダブルワード)を指定します。
[Number of software accessible resiters]を 6 にし、[Data width of each register]を 8bit にし、[Next]をクリックしてください。

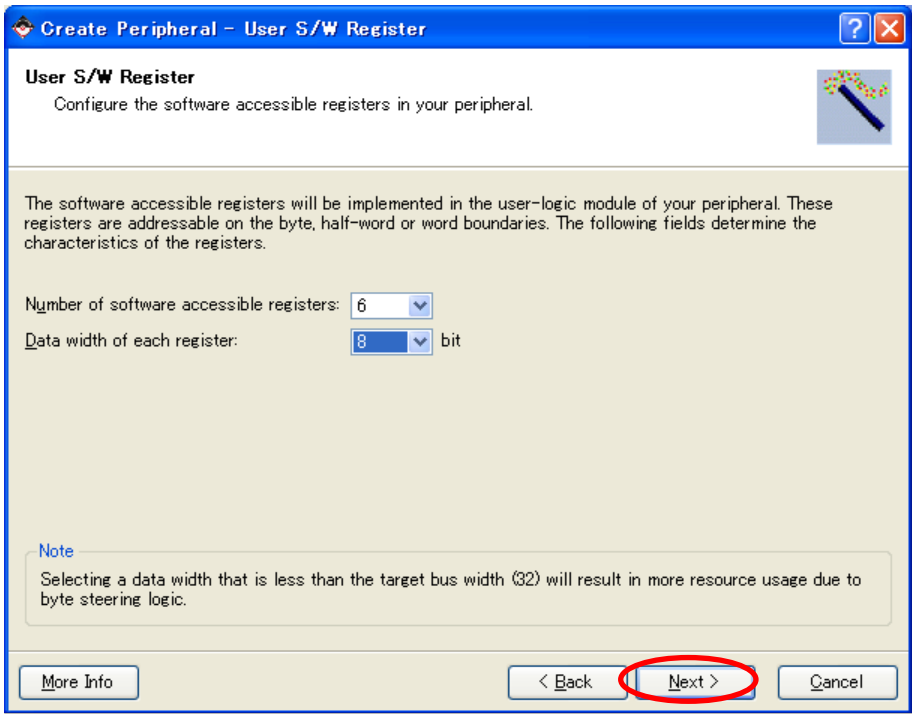


図 12-14 レジスタ数とバス幅指定

IPIC を設定します。すでにいくつか ON になっていますが、IPIF Services ページで指定した機能をインプリメントするために必要なものに自動でチェックされています。このまま[Next]をクリックしてください。

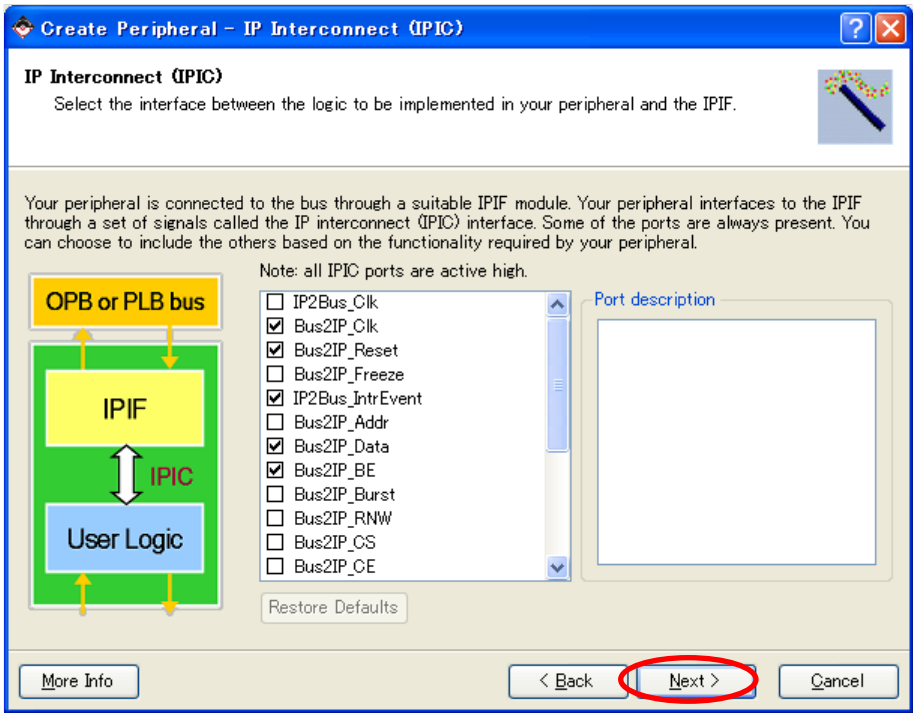


図 12-15 IPIC 設定

ここを ON にすると、カスタムロジックおよび機能のシミュレーションに使用するサポートファイルを生成できますが、今回は使いません。そのまま[Next]をクリックしてください。

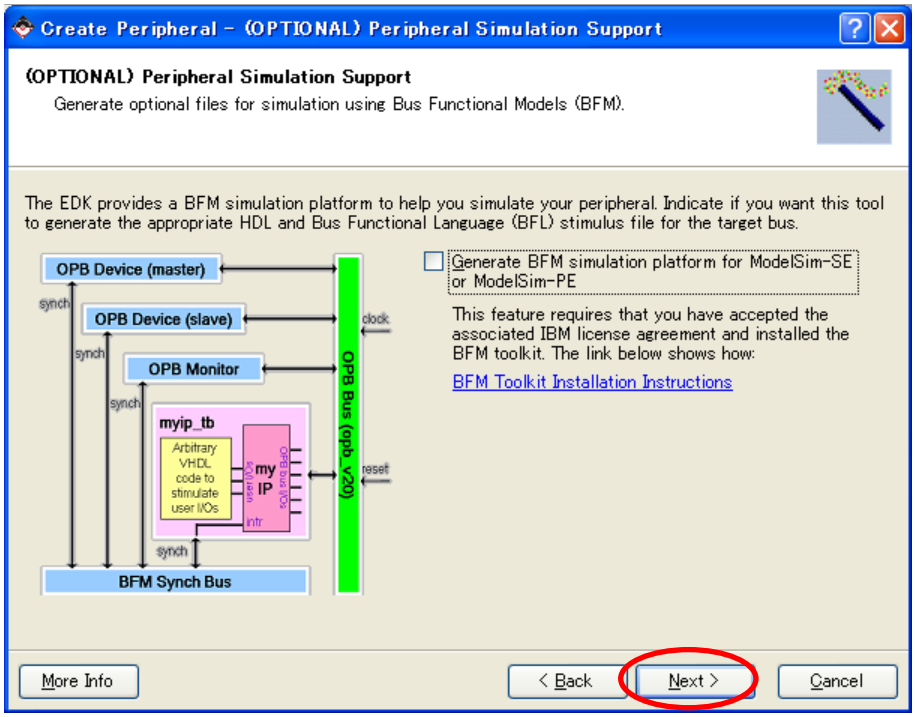


図 12-16 サポートファイル生成確認

下図の2つをチェックし、[Next]をクリックしてください。

ソフトウェアインターフェースのインプリメンテーションに使用するサポートファイル、ペリフェラルのドライバをインプリメントするためのソフトフェアドライバテンプレートファイルおよびドライバディレクトリ構成が作成されます。

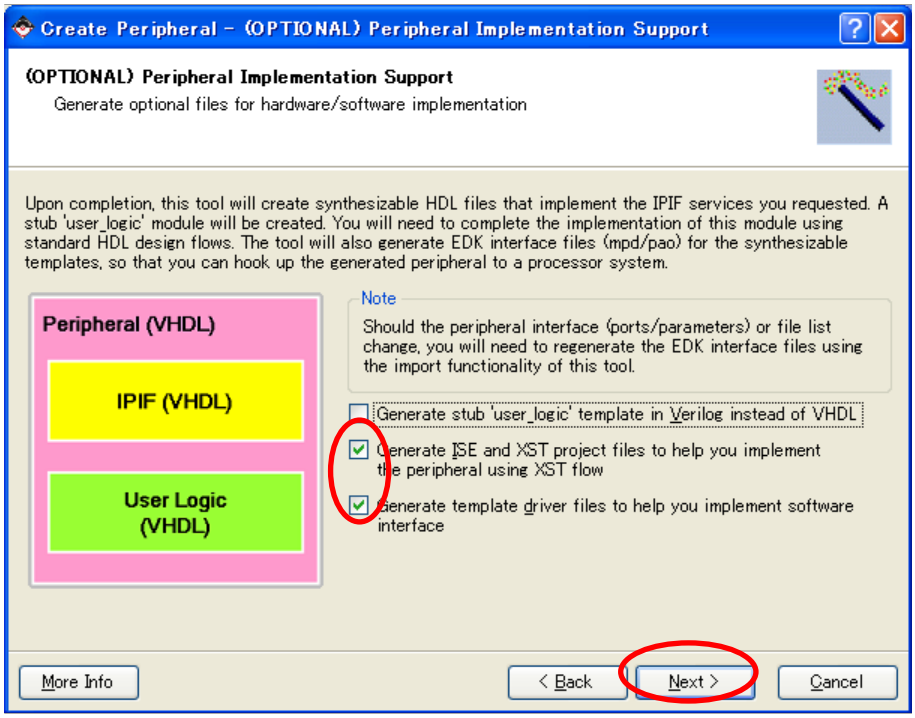


図 12-17 オプション設定

以上で終了です。[Finish]をクリックしてください。

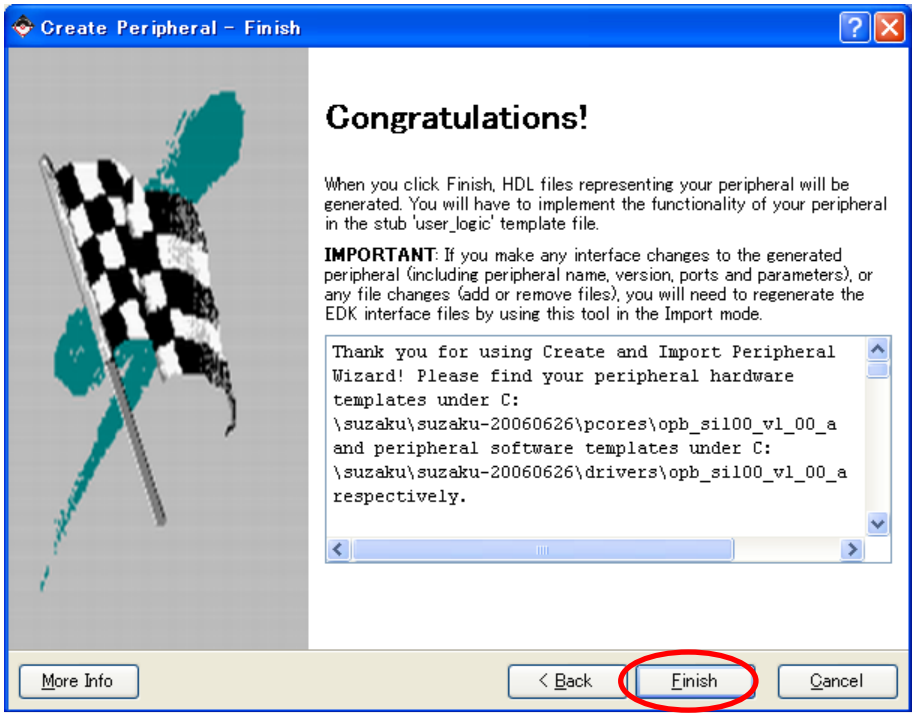


図 12-18 終了

“C:¥suzaku¥sz***-xxxxxxxx¥pcores”の下に自作コアの元が出来上がります。
生成されたファイルのディレクトリ構成は以下のようになります。

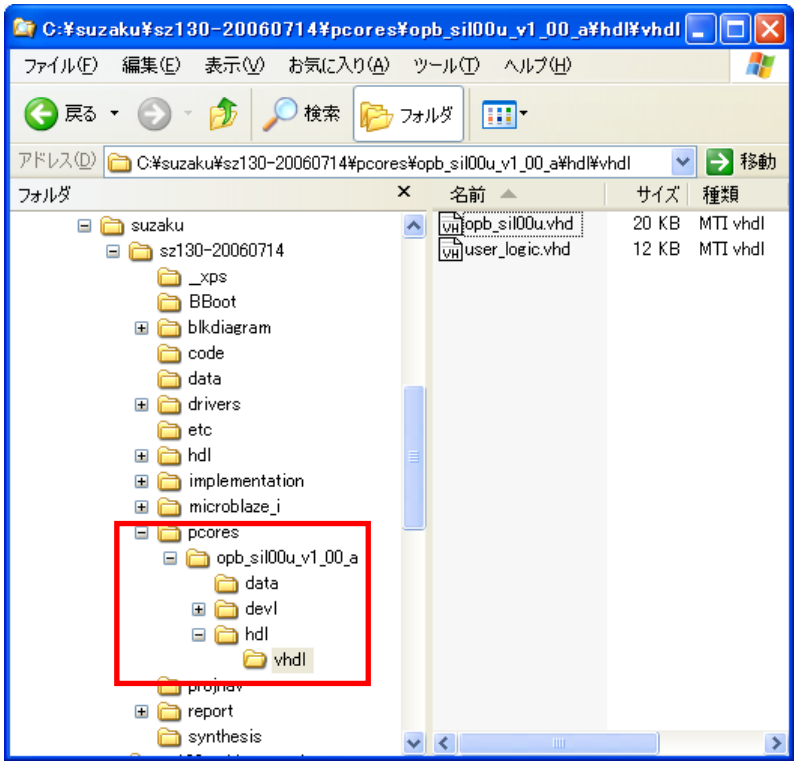


図 12-19 フォルダ構成

12.3. OPB インターフェースとコアを接続し、自作 IP コアを仕上げる

先ほど作ったコア sil00u_core.vhd を下図のように接続し、自作 IP コアを仕上げます。

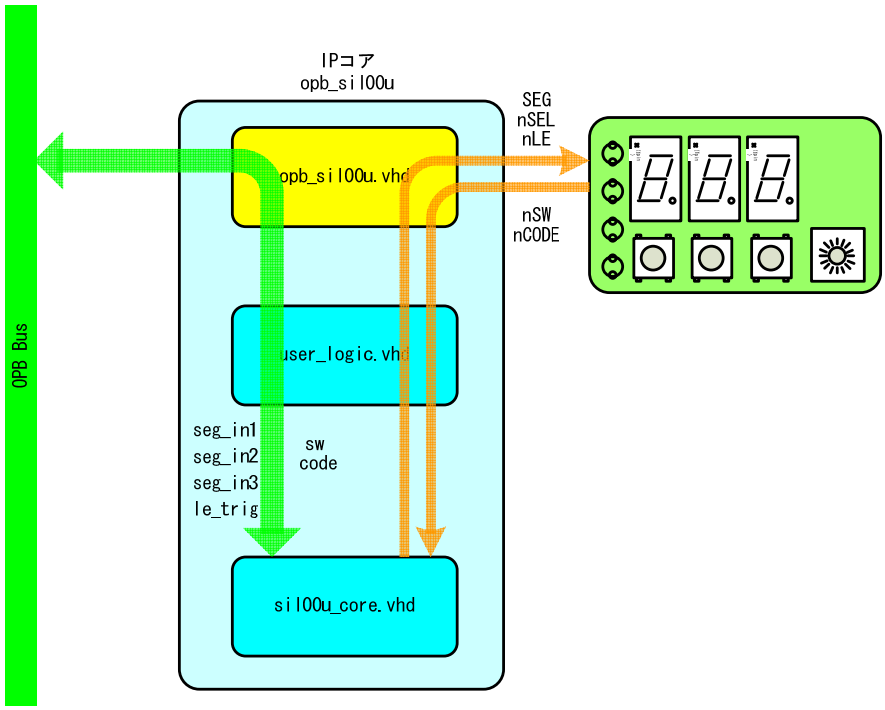


図 12-20 sil00u_core を接続

“C:¥suzaku¥sz***-xxxxxxxx¥pcores¥opb_sil00u_v1_00_a¥hdl¥vhdl”に sil00u_core.vhd、slot_counter.vhd、dynamic_ctrl.vhd、seg7_decorder.vhd、le_seq_blink.vhd をコピーしてください。

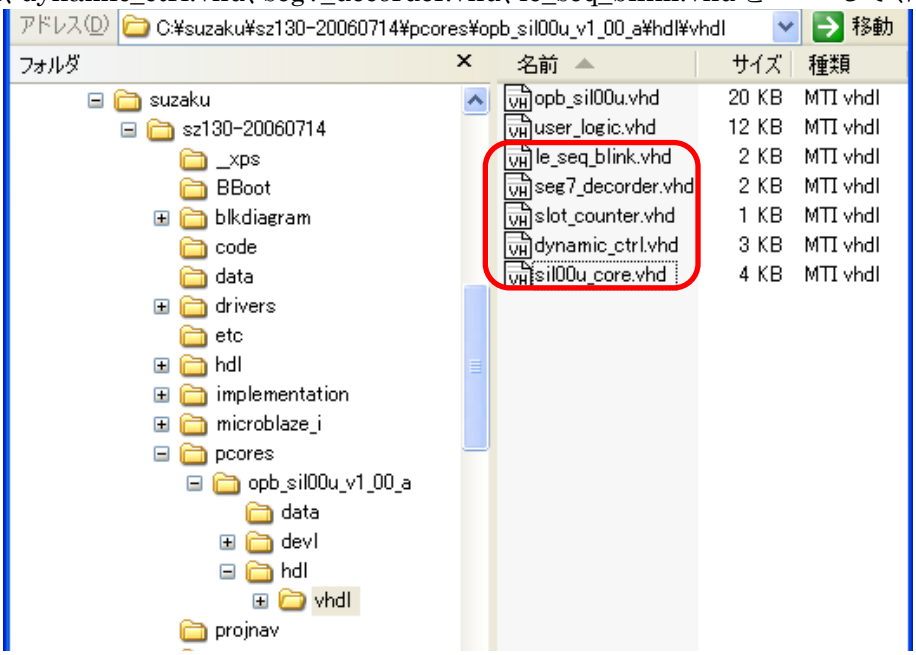


図 12-21 コアをコピー

■ user_logic.vhd

user_logic.vhd を開いてください。自動生成されたコードを編集していきます。user_logic を上位階層として、sil00u_core 回路を呼び出すソースコードを追加します。ソースコードを追加するところには、大体--USER xxx added here とコメントが入っているので、目印にしてください。

例 12-2 sil00u（user_logic.vhd）

```
------------------------------------------------------------------------------
-- user_logic.vhd - entity/architecture pair
------------------------------------------------------------------------------
--中略
-- DO NOT EDIT BELOW THIS LINE ---------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;

-- DO NOT EDIT ABOVE THIS LINE ---------------------

library opb_sil00u_v1_00_a;
use opb_sil00u_v1_00_a.all;

--中略
------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------

entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_DWIDTH                    : integer              := 8;
    C_NUM_CE                    : integer              := 6
    C_IP_INTR_NUM               : integer              := 1
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------

    SEG : out STD_LOGIC_VECTOR(0 to 7);
    nSEL : out STD_LOGIC_VECTOR(0 to 2);
    nLE : out STD_LOGIC_VECTOR(0 to 3);
    nSW : in STD_LOGIC_VECTOR(0 to 2);
    nCODE : in STD_LOGIC_VECTOR(0 to 3);
    intr : out STD_LOGIC;

    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                  : in  std_logic;
```

```
    Bus2IP_Reset                    : in  std_logic;
    Bus2IP_Data                     : in  std_logic_vector(0 to C_DWIDTH-1);
    Bus2IP_BE                       : in  std_logic_vector(0 to C_DWIDTH/8-1);
    Bus2IP_RdCE                     : in  std_logic_vector(0 to C_NUM_CE-1);
    Bus2IP_WrCE                     : in  std_logic_vector(0 to C_NUM_CE-1);
    IP2Bus_Data                     : out std_logic_vector(0 to C_DWIDTH-1);
    IP2Bus_Ack                      : out std_logic;
    IP2Bus_Retry                    : out std_logic;
    IP2Bus_Error                    : out std_logic;
    IP2Bus_ToutSup                  : out std_logic
    -- DO NOT EDIT ABOVE THIS LINE --------------------
  );
end entity user_logic;
------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------
architecture IMP of user_logic is

  signal slv_reg_r4                     : std_logic_vector(0 to C_DWIDTH-1);
  signal slv_reg_r5                     : std_logic_vector(0 to C_DWIDTH-1);

  ---------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ---------------------------------------
  signal slv_reg0                       : std_logic_vector(0 to C_DWIDTH-1);
  signal slv_reg1                       : std_logic_vector(0 to C_DWIDTH-1);
  signal slv_reg2                       : std_logic_vector(0 to C_DWIDTH-1);
  signal slv_reg3                       : std_logic_vector(0 to C_DWIDTH-1);
  signal slv_reg4                       : std_logic_vector(0 to C_DWIDTH-1);
  signal slv_reg5                       : std_logic_vector(0 to C_DWIDTH-1);
  signal slv_reg_write_select           : std_logic_vector(0 to 5);
  signal slv_reg_read_select            : std_logic_vector(0 to 5);
  signal slv_ip2bus_data                : std_logic_vector(0 to C_DWIDTH-1);
  signal slv_read_ack                   : std_logic;
  signal slv_write_ack                  : std_logic;
  ---------------------------------------
  -- Signals for user logic interrupt example
  ---------------------------------------
  signal interrupt                      : std_logic_vector(0 to C_IP_INTR_NUM-1);

begin

sil00u_core_0 : entity opb_sil00u_v1_00_a.sil00u_core
PORT MAP(
      SYS_CLK => Bus2IP_Clk,
      SYS_RST => Bus2IP_Reset,

-- External
      SEG => SEG,
      nSEL => nSEL,
      nLE => nLE,
      nSW => nSW,
      nCODE => nCODE,

-- Register Write
      seg_in1 => slv_reg0(4 to 7),
      seg_in2 => slv_reg1(4 to 7),
      seg_in3 => slv_reg2(4 to 7),
      le_trig => slv_reg3(7),
```

```
-- Register Read
      sw => slv_reg_r4(5 to 7),
      code => slv_reg_r5(4 to 7),
      intr => interrupt(0)
);

--中略

  slv_reg_write_select <= Bus2IP_WrCE(0 to 5);
  slv_reg_read_select <= Bus2IP_RdCE(0 to 5);
  slv_write_ack <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2)
                   or Bus2IP_WrCE(3) or Bus2IP_WrCE(4) or Bus2IP_WrCE(5);
  slv_read_ack <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2)
                   or Bus2IP_RdCE(3) or Bus2IP_RdCE(4) or Bus2IP_RdCE(5);

  -- implement slave model register(s)
  SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
  begin

    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Reset = '1' then
        slv_reg0 <= (others => '0');
        slv_reg1 <= (others => '0');
        slv_reg2 <= (others => '0');
        slv_reg3 <= (others => '0');
        slv_reg4 <= (others => '0');
        slv_reg5 <= (others => '0');
      else
        case slv_reg_write_select is
          when "100000" =>
            for byte_index in 0 to (C_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                 slv_reg0(byte_index*8 to byte_index*8+7)
                   <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
              end if;
            end loop;
          when "010000" =>
            for byte_index in 0 to (C_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg1(byte_index*8 to byte_index*8+7)
                  <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
              end if;
            end loop;
          when "001000" =>
            for byte_index in 0 to (C_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg2(byte_index*8 to byte_index*8+7)
                  <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
              end if;
            end loop;
          when "000100" =>
            for byte_index in 0 to (C_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg3(byte_index*8 to byte_index*8+7)
                  <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
              end if;
            end loop;
          when "000010" =>
            for byte_index in 0 to (C_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
```

```
            slv_reg4(byte_index*8 to byte_index*8+7)
              <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
        end loop;
      when "000001" =>
        for byte_index in 0 to (C_DWIDTH/8)-1 loop
          if ( Bus2IP_BE(byte_index) = '1' ) then
            slv_reg5(byte_index*8 to byte_index*8+7)
              <= Bus2IP_Data(byte_index*8 to byte_index*8+7);
          end if;
        end loop;
      when others => null;
    end case;
  end if;
end if;

end process SLAVE_REG_WRITE_PROC;

-- implement slave model register read mux

--  SLAVE_REG_READ_PROC : process( slv_reg_read_select, slv_reg0,
--   slv_reg1, slv_reg2, slv_reg3, slv_reg4, slv_reg5 ) is
 SLAVE_REG_READ_PROC : process( slv_reg_read_select, slv_reg0, slv_reg1,
                                slv_reg2, slv_reg3, slv_reg_r4, slv_reg_r5 ) is

begin

  case slv_reg_read_select is
    when "100000" => slv_ip2bus_data <= slv_reg0;
    when "010000" => slv_ip2bus_data <= slv_reg1;
    when "001000" => slv_ip2bus_data <= slv_reg2;
    when "000100" => slv_ip2bus_data <= slv_reg3;

--      when "000010" => slv_ip2bus_data <= slv_reg4;
--      when "000001" => slv_ip2bus_data <= slv_reg5;
    when "000010" => slv_ip2bus_data <= slv_reg_r4;
    when "000001" => slv_ip2bus_data <= slv_reg_r5;

    when others => slv_ip2bus_data <= (others => '0');
  end case;

end process SLAVE_REG_READ_PROC;

------------------------------------------
-- Example code to generate user logic interrupts
--
-- Note:
-- The example code presented here is to show you one way of generating
-- interrupts from the user logic. This code snippet infers a counter
-- and generate the interrupts whenever the counter rollover (the counter
-- will rollover ~21 sec @50Mhz).
------------------------------------------
--  INTR_PROC : process( Bus2IP_Clk ) is
--    constant COUNT_SIZE   : integer := 30;
--    constant ALL_ONES     : std_logic_vector(0 to COUNT_SIZE-1) := (others => '1');
--    variable counter      : std_logic_vector(0 to COUNT_SIZE-1);
--  begin
--
--    if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
--      if ( Bus2IP_Reset = '1' ) then
--        counter := (others => '0');
```

```
--       interrupt <= (others => '0');
--     else
--       counter := counter + 1;
--       if ( counter = ALL_ONES ) then
--         interrupt <= (others => '1');
--       else
--         interrupt <= (others => '0');
--       end if;
--     end if;
--   end if;
--
-- end process INTR_PROC;

  IP2Bus_IntrEvent <= interrupt;

  ----------------------------------------
  -- Example code to drive IP to Bus signals
  ----------------------------------------
  IP2Bus_Data       <= slv_ip2bus_data;

  IP2Bus_Ack        <= slv_write_ack or slv_read_ack;
  IP2Bus_Error      <= '0';
  IP2Bus_Retry      <= '0';
  IP2Bus_ToutSup    <= '0';

end IMP;
```

■ opb_sil00u.vhd

opb_sil00u.vhd を開いてください。自動生成されたコードを編集していきます。opb_sil00u を上位階層として、user_logic 回路を呼び出すコードを追加します。

例 12-3 sil00u（opb_sil00.vhd）

```
------------------------------------------------------------------------------
-- opb_sil00u.vhd - entity/architecture pair
------------------------------------------------------------------------------
-- 中略
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;
use proc_common_v2_00_a.ipif_pkg.all;
library opb_ipif_v3_01_c;
use opb_ipif_v3_01_c.all;

library opb_sil00u_v1_00_a;
use opb_sil00u_v1_00_a.all;

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------

entity opb_sil00u is
  generic
  (
    --中略
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------

    SEG : out  STD_LOGIC_VECTOR(0 to 7); --7 セグメント LED への出力信号
    nSEL: out STD_LOGIC_VECTOR(0 to 2); --7 セグメント LED セレクト信号
    nLE : out STD_LOGIC_VECTOR(0 to 3); --単色 LED への出力信号
    nSW  : in STD_LOGIC_VECTOR(0 to 2); --押しボタンスイッチからの入力信号
    nCODE : in STD_LOGIC_VECTOR(0 to 3); --ロータリコードスイッチからの入力信号

    -- ADD USER PORTS ABOVE THIS LINE ------------------
    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    OPB_Clk                     : in  std_logic;
    OPB_Rst                     : in  std_logic;
    Sl_DBus                     : out std_logic_vector(0 to C_OPB_DWIDTH-1);
    Sl_errAck                   : out std_logic;
    Sl_retry                    : out std_logic;
    Sl_toutSup                  : out std_logic;
    Sl_xferAck                  : out std_logic;
    OPB_ABus                    : in  std_logic_vector(0 to C_OPB_AWIDTH-1);
    OPB_BE                      : in  std_logic_vector(0 to C_OPB_DWIDTH/8-1);
    OPB_DBus                    : in  std_logic_vector(0 to C_OPB_DWIDTH-1);
    OPB_RNW                     : in  std_logic;
    OPB_select                  : in  std_logic;
    OPB_seqAddr                 : in  std_logic;
    IP2INTC_Irpt                : out std_logic
```

```
    -- DO NOT EDIT ABOVE THIS LINE --------------------
  );
  attribute SIGIS : string;
  attribute SIGIS of OPB_Clk      : signal is "Clk";
  attribute SIGIS of OPB_Rst      : signal is "Rst";
  attribute SIGIS of IP2INTC_Irpt  : signal is "INTR_LEVEL_HIGH";

end entity opb_sil00u;
------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------
architecture IMP of opb_sil00u is
--中略
begin
  ---------------------------------------
  -- instantiate the OPB IPIF
  ---------------------------------------
  OPB_IPIF_I : entity opb_ipif_v3_01_c.opb_ipif
    generic map
    (
--中略
    )
    port map
    (
--中略
    );
  ---------------------------------------
  -- instantiate the User Logic
  ---------------------------------------
  USER_LOGIC_I : entity opb_sil00u_v1_00_a.user_logic
    generic map
    (
--中略
    )
    port map
    (
      -- MAP USER PORTS BELOW THIS LINE ------------------
      SEG => SEG,
      nSEL => nSEL,
      nLE => nLE,
      nSW => nSW,
      nCODE => nCODE,
      -- MAP USER PORTS ABOVE THIS LINE ------------------
      Bus2IP_Clk                    => iBus2IP_Clk,
      Bus2IP_Reset                   => iBus2IP_Reset,
      Bus2IP_Data                    => uBus2IP_Data,
      Bus2IP_BE                     => uBus2IP_BE,
      Bus2IP_RdCE                    => uBus2IP_RdCE,
      Bus2IP_WrCE                    => uBus2IP_WrCE,
      IP2Bus_Data                    => uIP2Bus_Data,
      IP2Bus_Ack                     => iIP2Bus_Ack,
      IP2Bus_Retry                   => iIP2Bus_Retry,
      IP2Bus_Error                   => iIP2Bus_Error,
      IP2Bus_ToutSup                  => iIP2Bus_ToutSup
    );
--中略
end IMP;
```

● ライブラリ

ライブラリはデザインデータの集まりで、パッケージ宣言、エンティティ宣言、アーキテクチャ宣言などを格納します。ひとつのファイルに2つ以上のエンティティがある場合は、use によるパッケージ呼び出しが必要になります。

パッケージ文を呼び出すことによってコンポーネント宣言を省略することができます。

```
library proc_common_v2_00_a;
use proc_common_v2_00_a.proc_common_pkg.all;
```

■ opb_sil00u_v2_1_0.mpd、opb_sil00u_v2_1_0.pao

“C:¥suzaku¥sz***-xxxxxxxx¥pcores¥opb_sil00u_v1_00_a¥data”を開いてください。

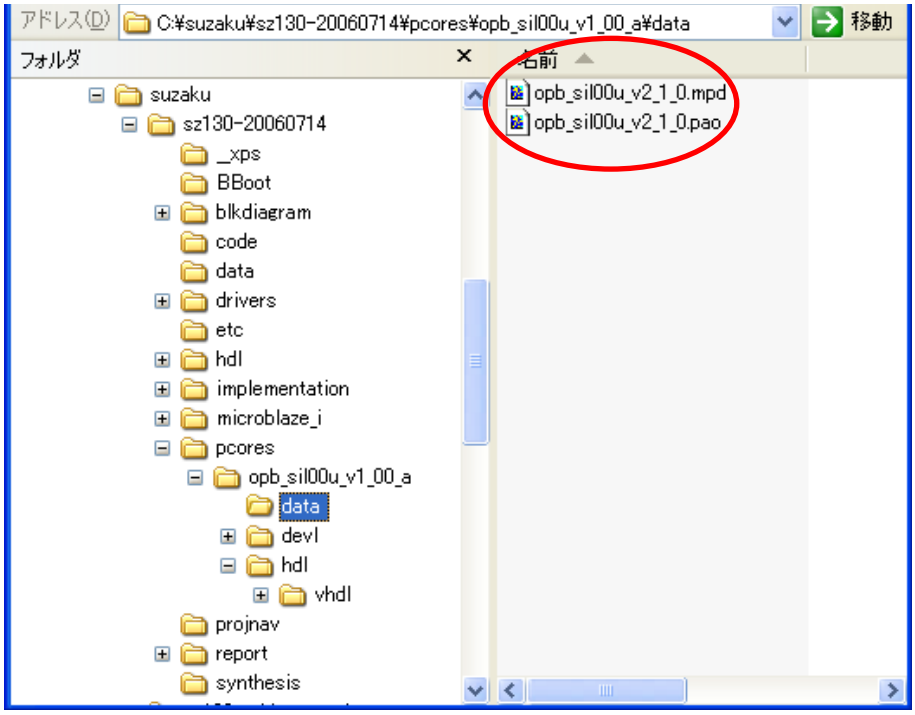


図 12-22 フォルダ構成

opb_sil00u_v2_1_0.mpd を編集します。mpd(Microprocessor Peripheral Definition)ファイルにはペリフェラルのインターフェースを定義します。

以下の文を一番下に追加してください。

例 12-4 opb_sil00u_v2_1_0.mpd

```
PORT SEG  = "", DIR = O, VEC = [0:7]
PORT nSEL  = "", DIR = O, VEC = [0:2]
PORT nLE  = "", DIR = O, VEC = [0:3]
PORT nSW  = "", DIR = I, VEC = [0:2]
PORT nCODE  = "", DIR = I, VEC = [0:3]
```

opb_sil00u_v2_1_0.pao を編集します。pao(Peripheral Analyze Order)ファイルはペリフェラルのコンパイル（構成およびシミュレーション用）に必要な HDL ファイルと、その解析順を指定します。

以下の文を一番下に追加してください。

例 12-5 opb_sil00u_v2_1_0.pao

```
lib opb_sil00u_v1_00_a sil00u_core vhdl
lib opb_sil00u_v1_00_a slot_counter vhdl
lib opb_sil00u_v1_00_a le_seq_blink vhdl
lib opb_sil00u_v1_00_a seg7_decorder vhdl
lib opb_sil00u_v1_00_a dynamic_ctrl vhdl
```

12.4. 自作 IP コアの追加

EDK に自作コアが作成されたことを伝えるため、[Project]→[Rescan User Repositories]をクリックしてください。

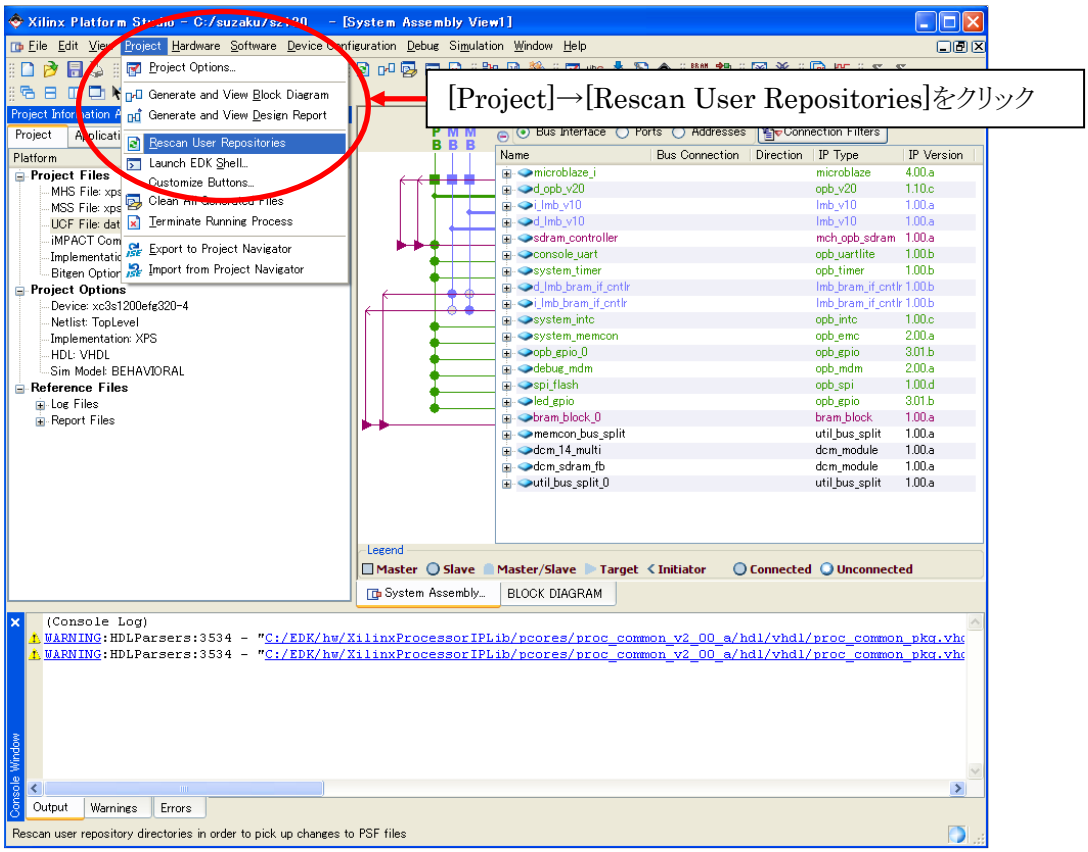


図 12-23 自作 IP コア読み込み

IP Catalog の Project Repository に自分のコア opb_sil00u が追加されます。もしうまく追加されなかった場合は、一回 Xilinx Platform Studio を閉じて、再起動し、xps_proj.xmp を開き直してください。

12.4.1. IP コアの追加

opb_sil00u を右クリックして出てくるメニューの Add IP を選択してください。
opb_sil00u が追加されます。

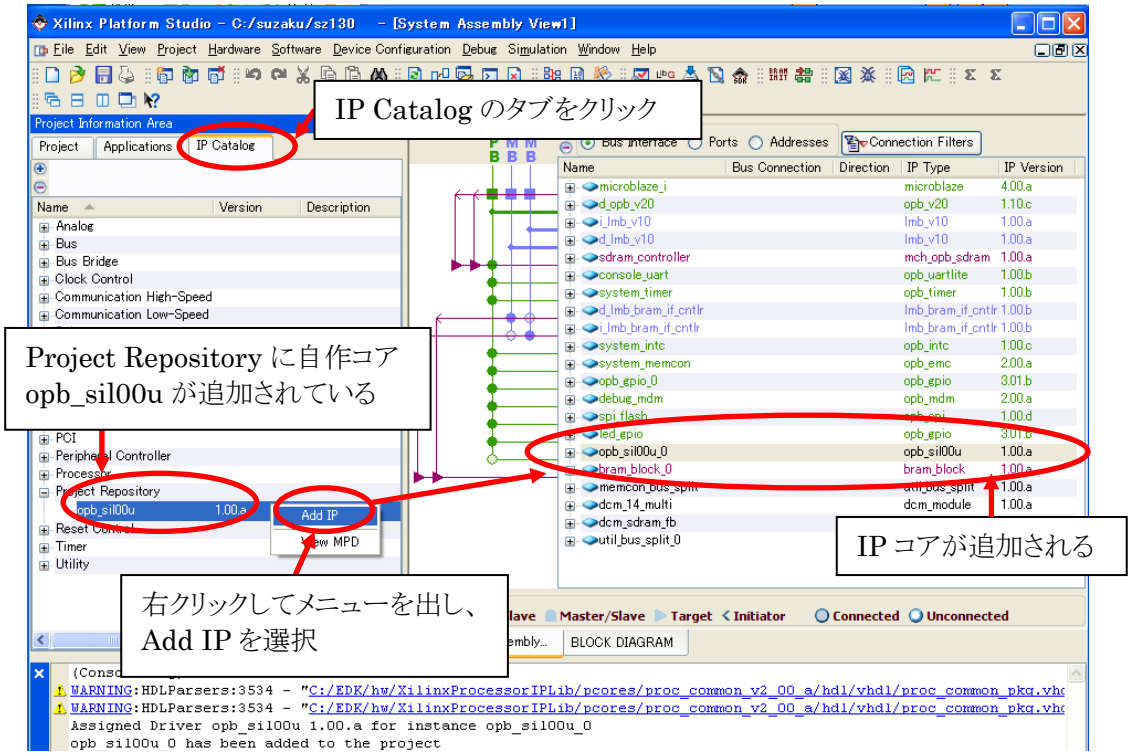


図 12-24 自作 IP コア追加

12.4.2. OPB バスに接続

Bus Interface を選択し、opb_sil00u_0の横の丸をクリックしてください。 ○ → ●
OPB バスに接続されます。

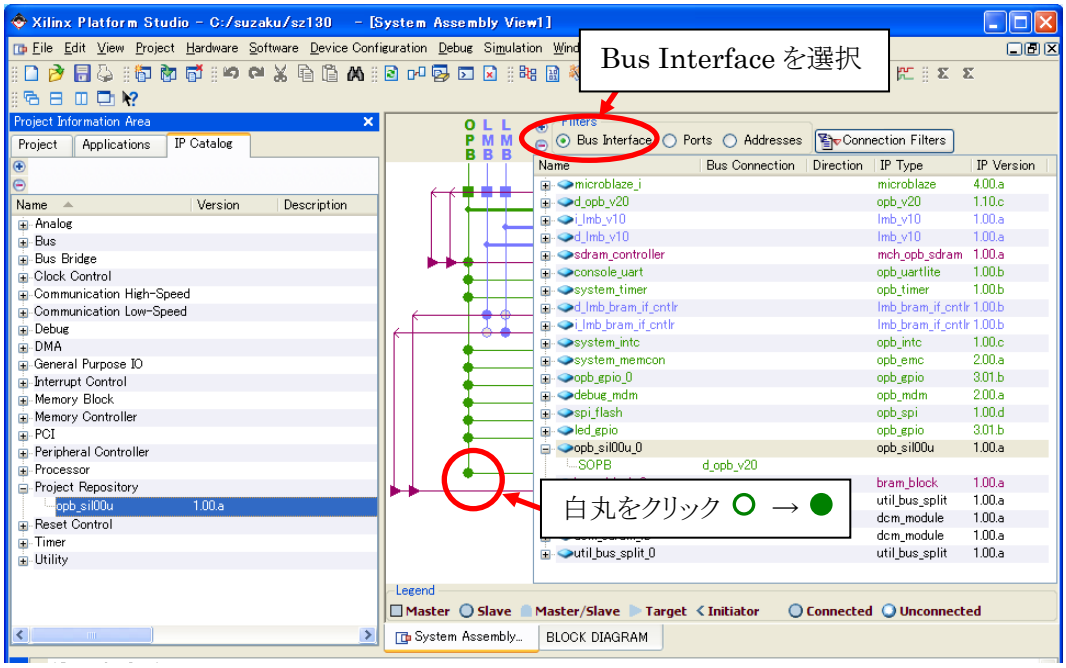


図 12-25 OPB バスに接続

12.4.3. IP コアの設定

opb_sil00u_0 を右クリックし、メニューの Configure IP...を選択してください。

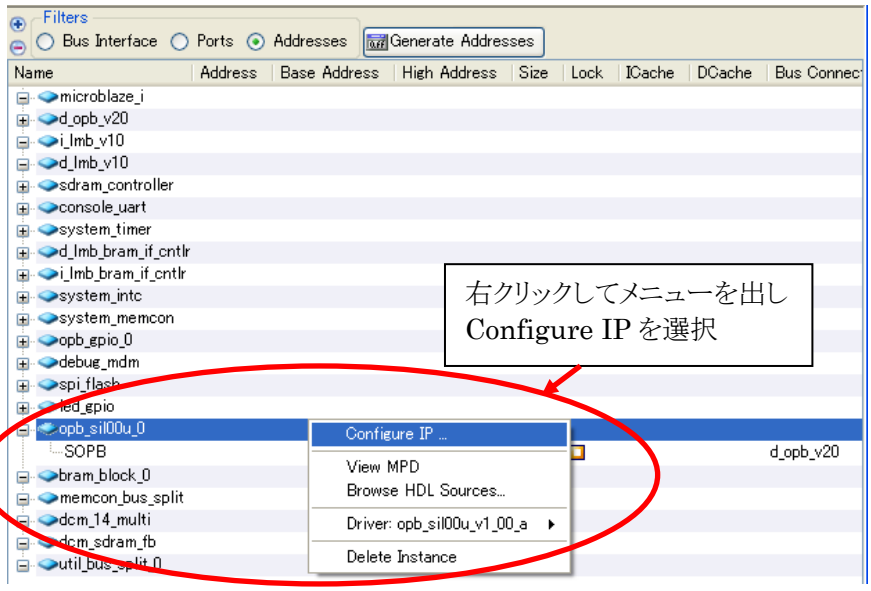


図 12-26 アドレス設定画面呼び出し

メモリアドレスを設定します。[BASEADDR]、[HIGHADDR]にメモリアドレスを入力し、[OK]をクリックして下さい。メモリアドレスは SUZAKU のメモリマップで Free と書いてあるところに割り当てます。(”5.3 SUZAKU メモリマップ” 参照)

表 12-1 sil00u メモリアドレス

	SZ010、SZ030 SZ130	SZ310
Base Address	0xFFFFD000	0xF0FFD000
High Address	0xFFFFD1FF	0xF0FFD1FF

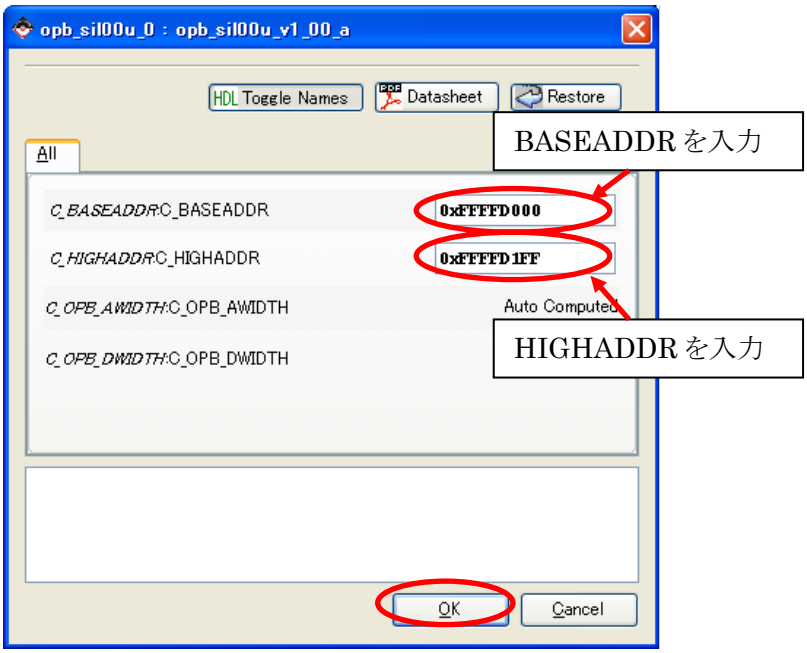


図 12-27 アドレス設定

12.4.4. メモリマップ確認

Addresses を選択し、opb_sil00u_0 の BaseAddress と High Address と Size に間違いがないか、確認してください。

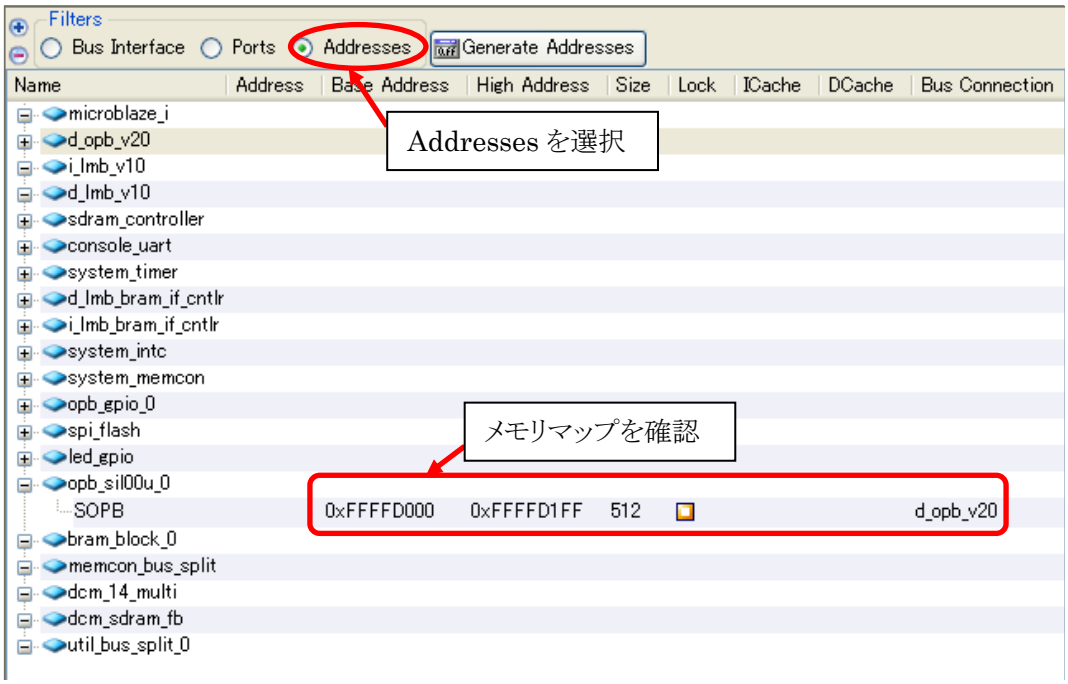


図 12-28 メモリマップ確認

12.4.5. 信号の定義

Ports を選択し、opb_sil00u_0 の⊞をクリックして開いてください。
SEG の Net の部分をクリックし、Net 名を SEG と入力し、欄外をクリックし確定させてください。

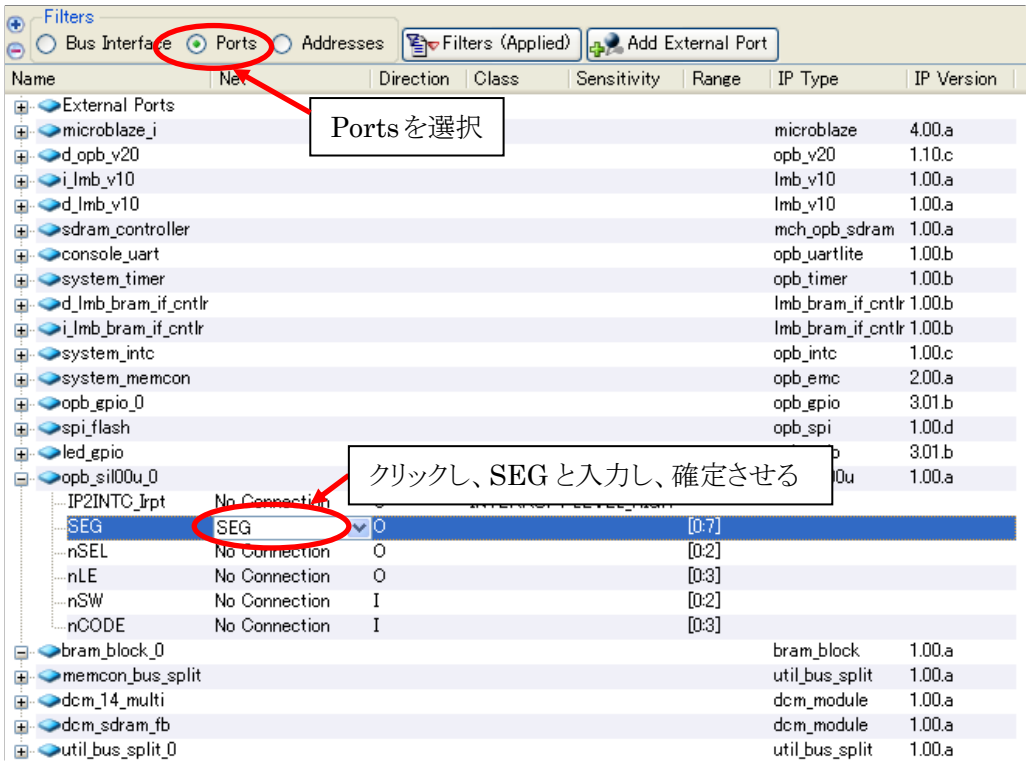


図 12-29 NET 名入力

もう一度 SEG の Net をクリックし、今度は▼をクリックし、 [Make External]を選択し、欄外をクリックして確定させてください。

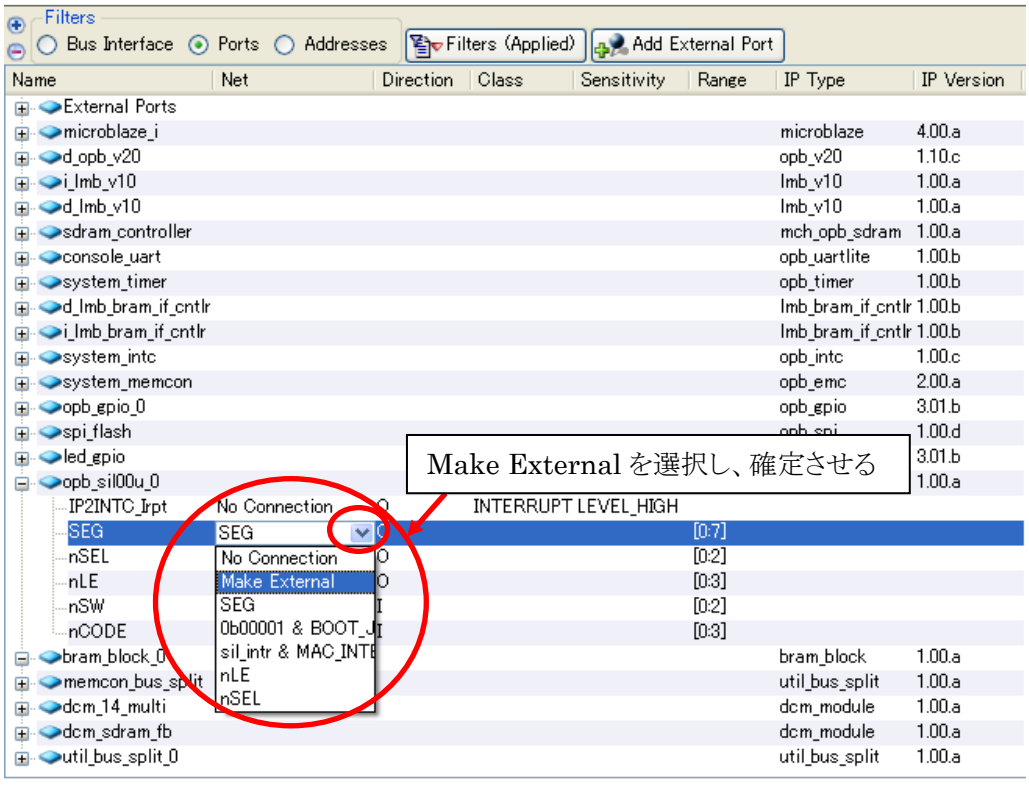


図 12-30 外部信号にする

External Ports の⊞をクリックして開いてください。Name:SEG_pin という信号が出来上がっているので、SEG_pin をクリックし、名前を SEG に変更してください。これで、外部出力信号 SEG が定義されます。

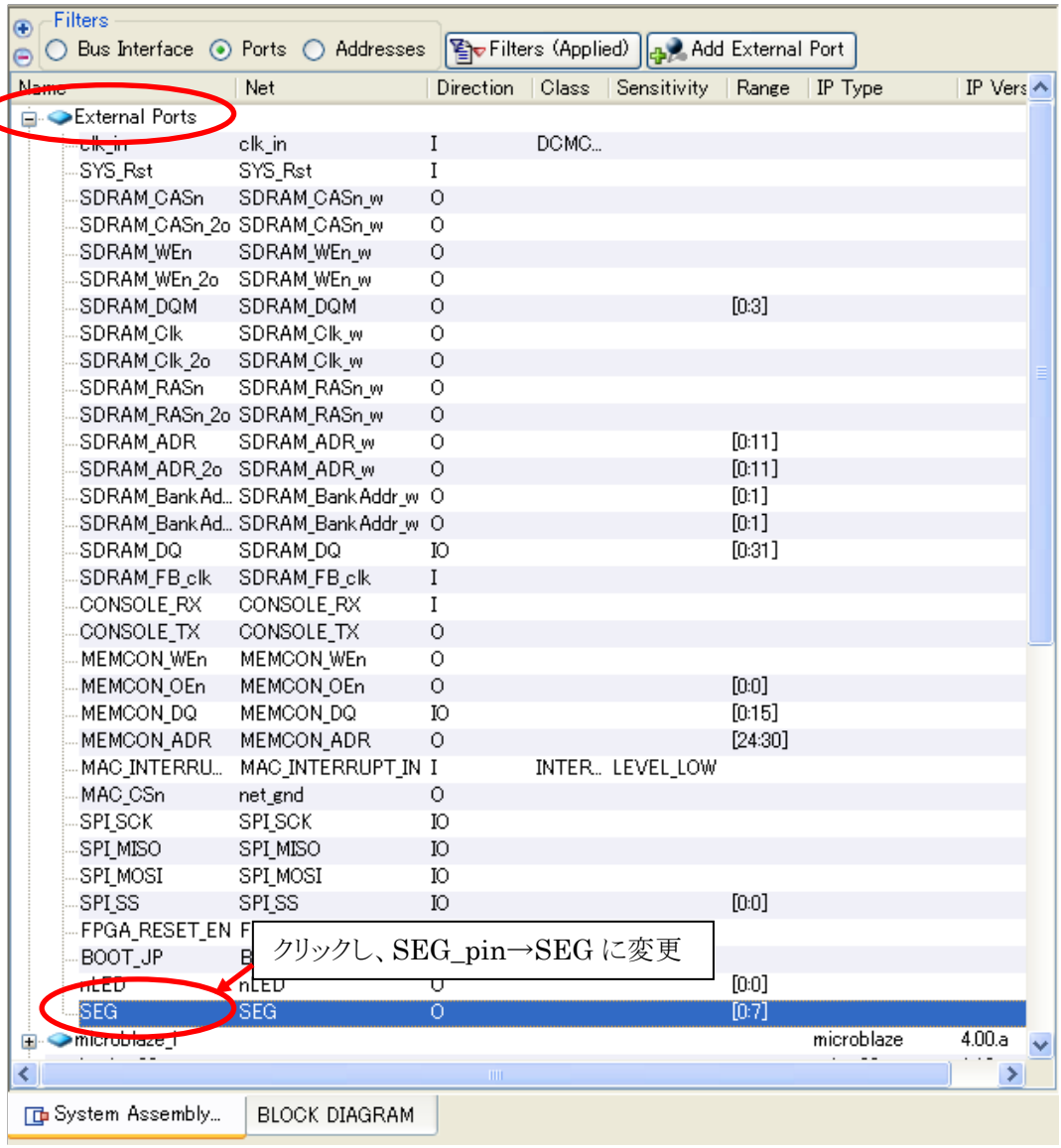
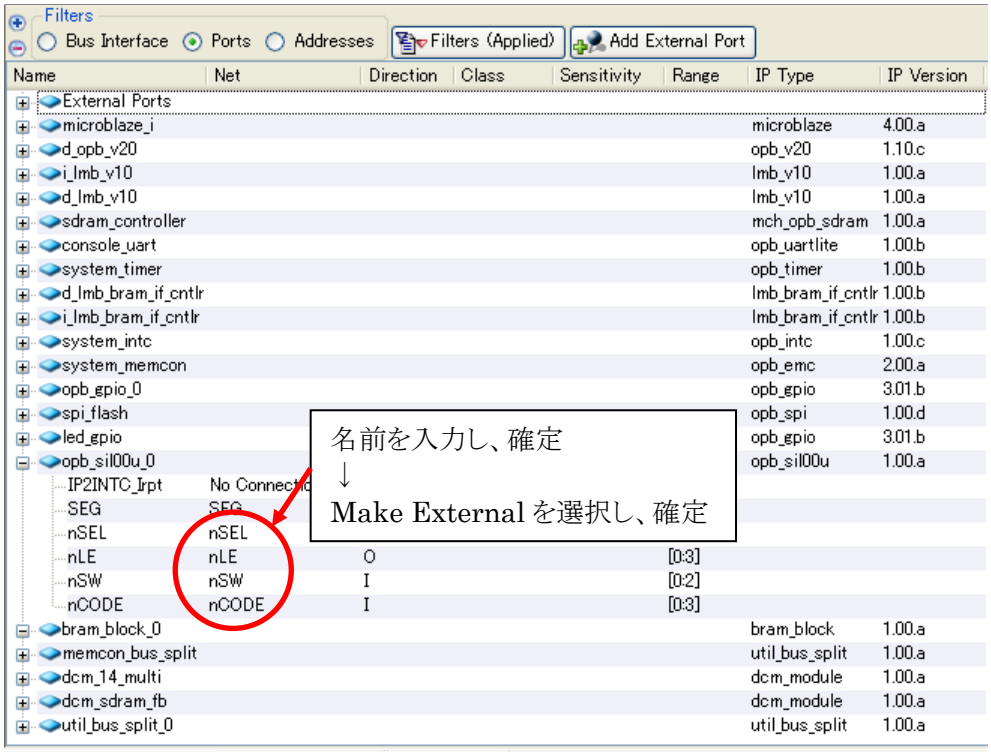


図 12-31 出力信号定義

nSEL、nLE、nSW、nCODE も SEG と同様の操作を行ってください。



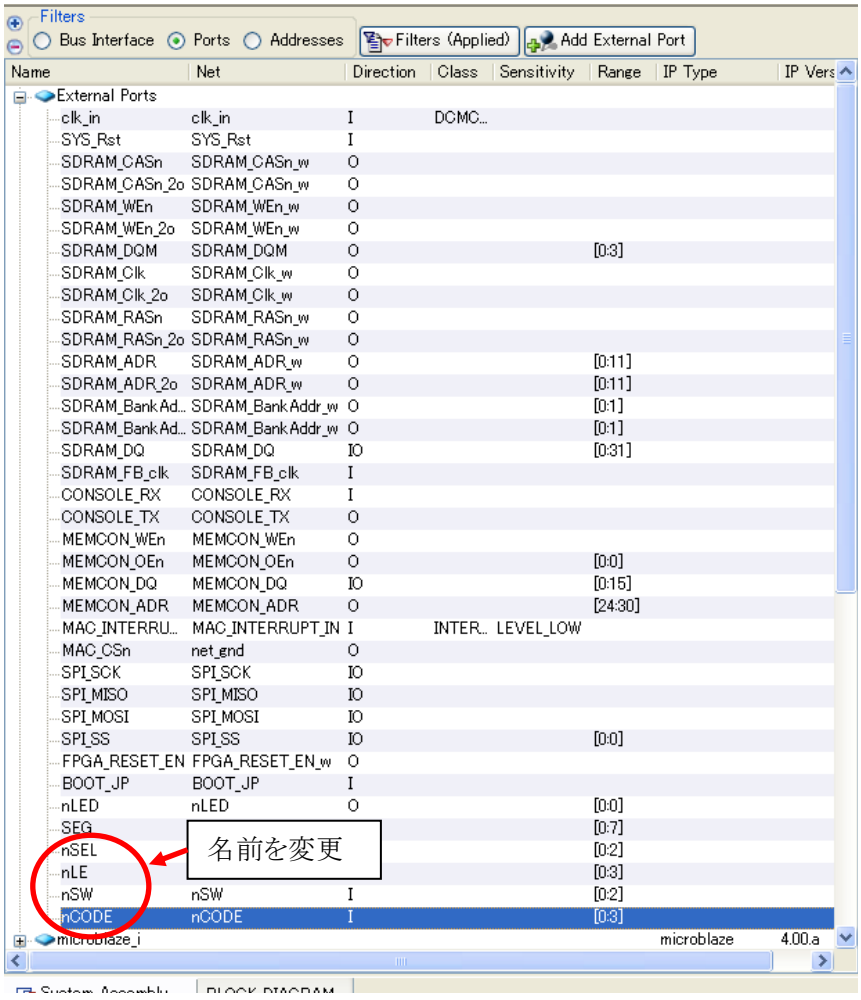


図 12-32 残り出力信号定義

SUZAKU では、複数の割り込み要因があるため、割り込みコントローラを用いて、割り込み処理を行います。割り込みが発生すると、割り込みハンドラが呼び出され、優先順位が高いものから割り込みが処理されます。

IP2INTC_Irpt の Net に sil_intr と入力し、system_inc の Intr の Net に、sil_intr & と追記してください。これで自作コアの割り込みが割り込みコントローラに接続されます。右側に書いたものほど割り込みの優先順位が高くなります。

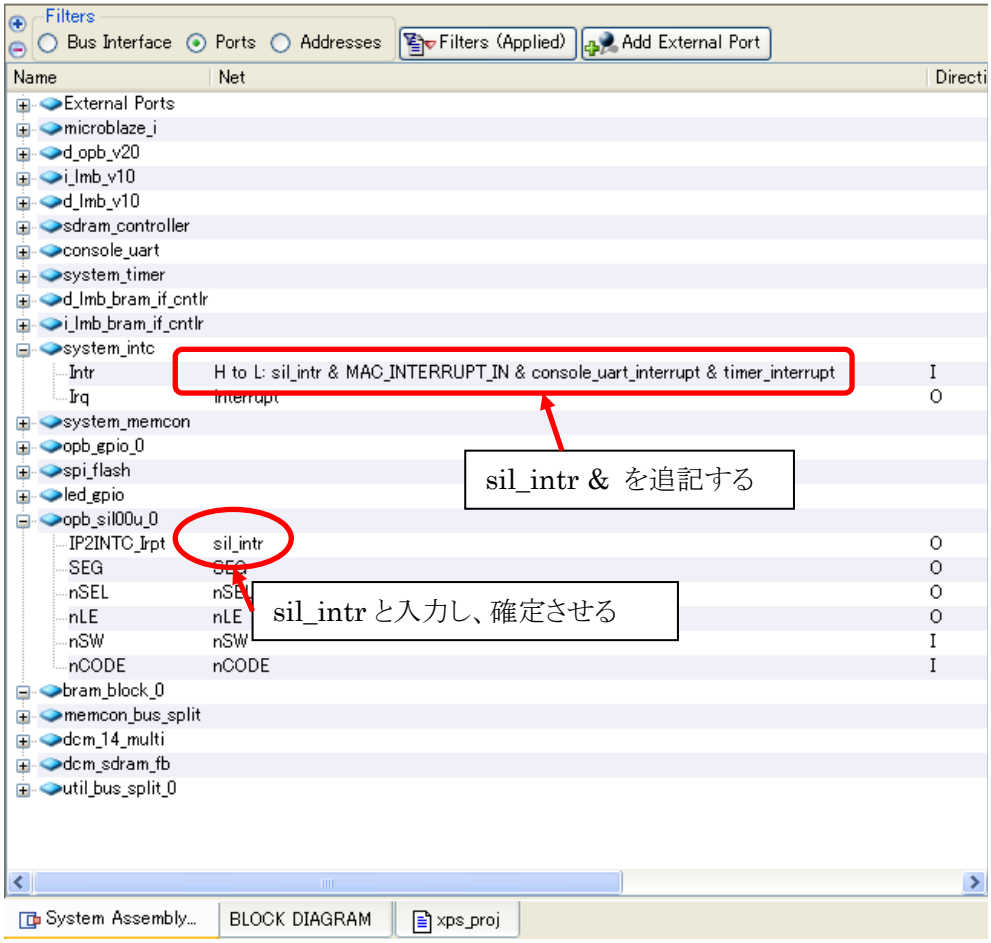


図 12-33 割り込みコントローラ

● SZ310 の場合(SZ010、SZ030、SZ130 の場合はとばしてください)

SZ310 はハードプロセッサの PowerPC なので、設定が SZ010、SZ030、SZ130 と違います。
PowerPC では、EVPR(例外ベクタプレフィックスレジスタ)に割り込みベクタをセットする必要があります。
EVPR は上位 16bit のみが使用され、下位 16nit は無視されるレジスタです。
デフォルトのプロジェクトでは、boot セクションが 0xFFFFFFFC にあり、BRAM を 0xFFFFC000～0xFFFFFFFF に割り当てています。
EVPR に割り込みベクタをセットするためには、0xFFFF0000 まで拡張しないとならないのですが、これだと BRAM の容量が 64kbyte になってしまいます。SZ310-U00 に採用している Virtex-II Pro は 48kbyte の BRAM しか内蔵していないため足りません。
今回は、0xFFFFC000～0xFFFFFFFF の他に、もうひとつ BRAM を用意し、0xFFFF0000～0xFFFF3FFF にセットし、ここに割り込みベクタを割り当てるようにします。
また、text セクションが 15kbyte になっていますので、rodata、data、bss なども、0xFFFF0000～0xFFFF3FFF 側に割り当てて、必要容量を分散することにします。

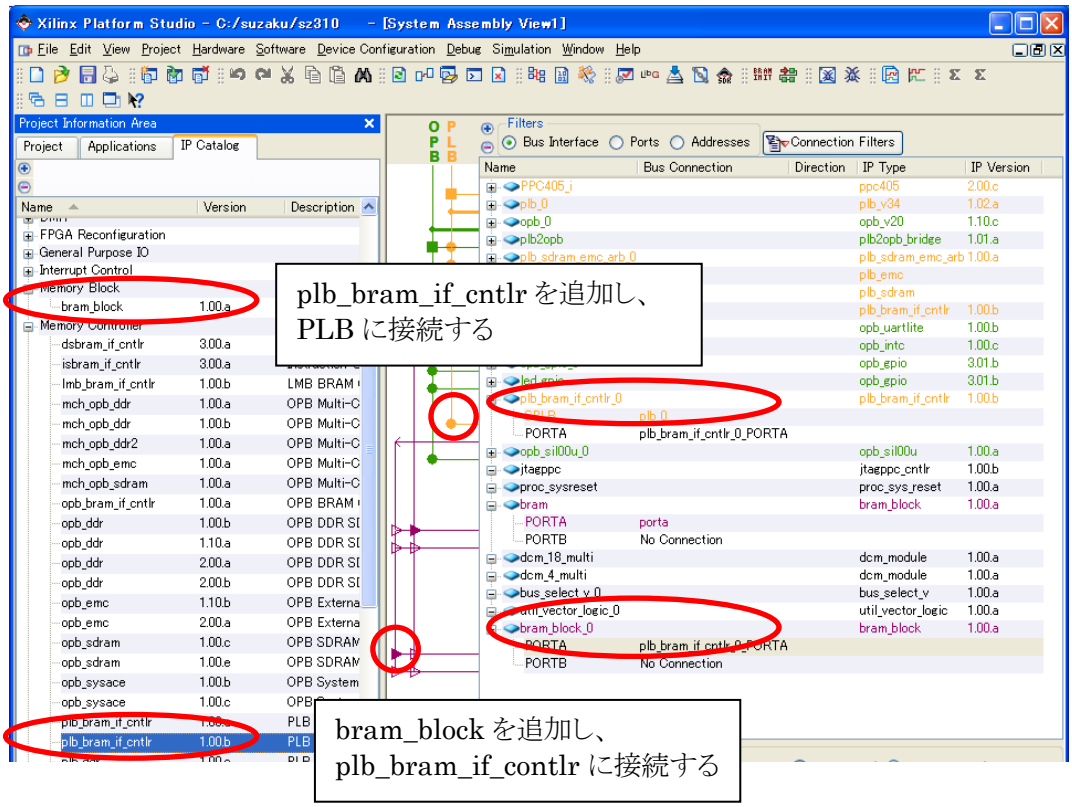


図 12-34 SZ310 割り込み設定1

plb_bram_if_cntlr_0 の設定画面を開き、Base Address に[0xFFFF0000]、High Address に[0xFFFF3FFF]と入力し、[OK]をクリックして下さい。

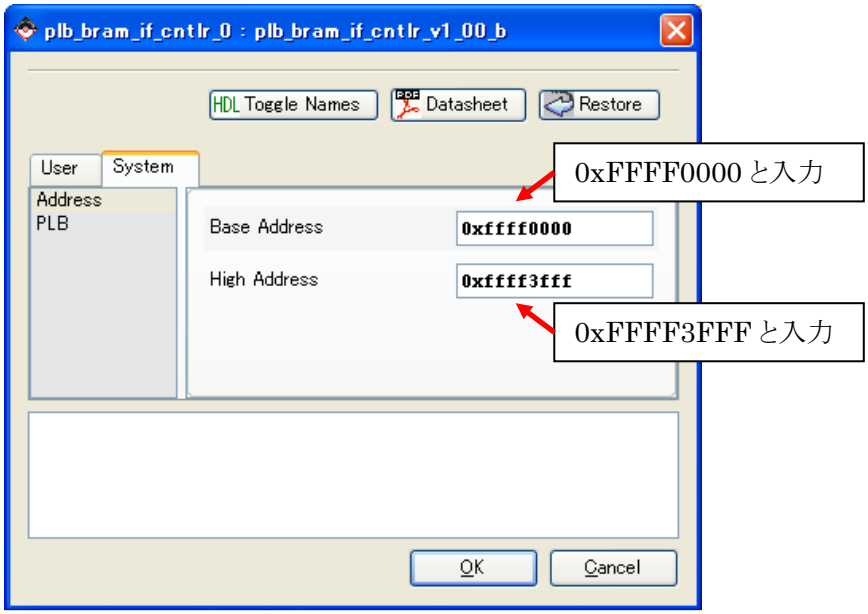


図 12-35 SZ310 割り込み設定 2

[Software]→[Generate Linker Script...]を選択してください。

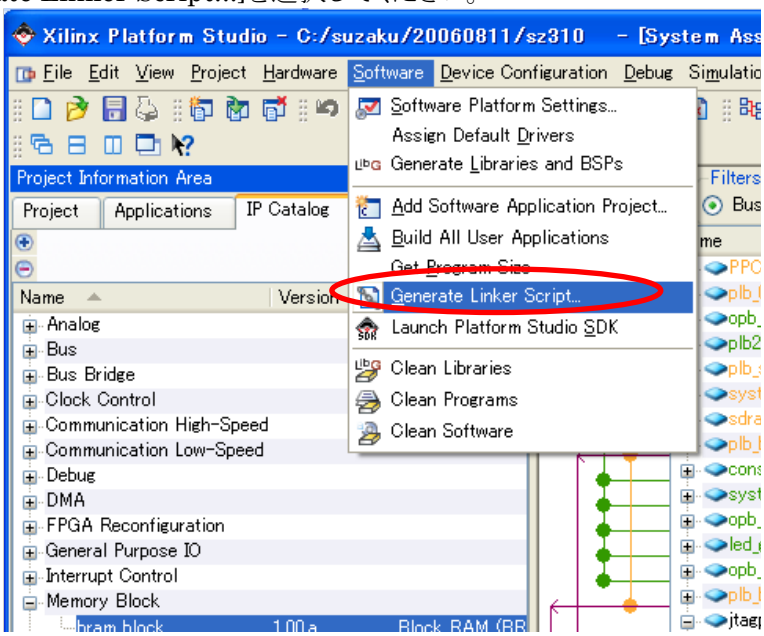


図 12-36 SZ310 割り込み設定 3

[Sections View]の Memory の部分を編集し、[Generate]をクリックして下さい。

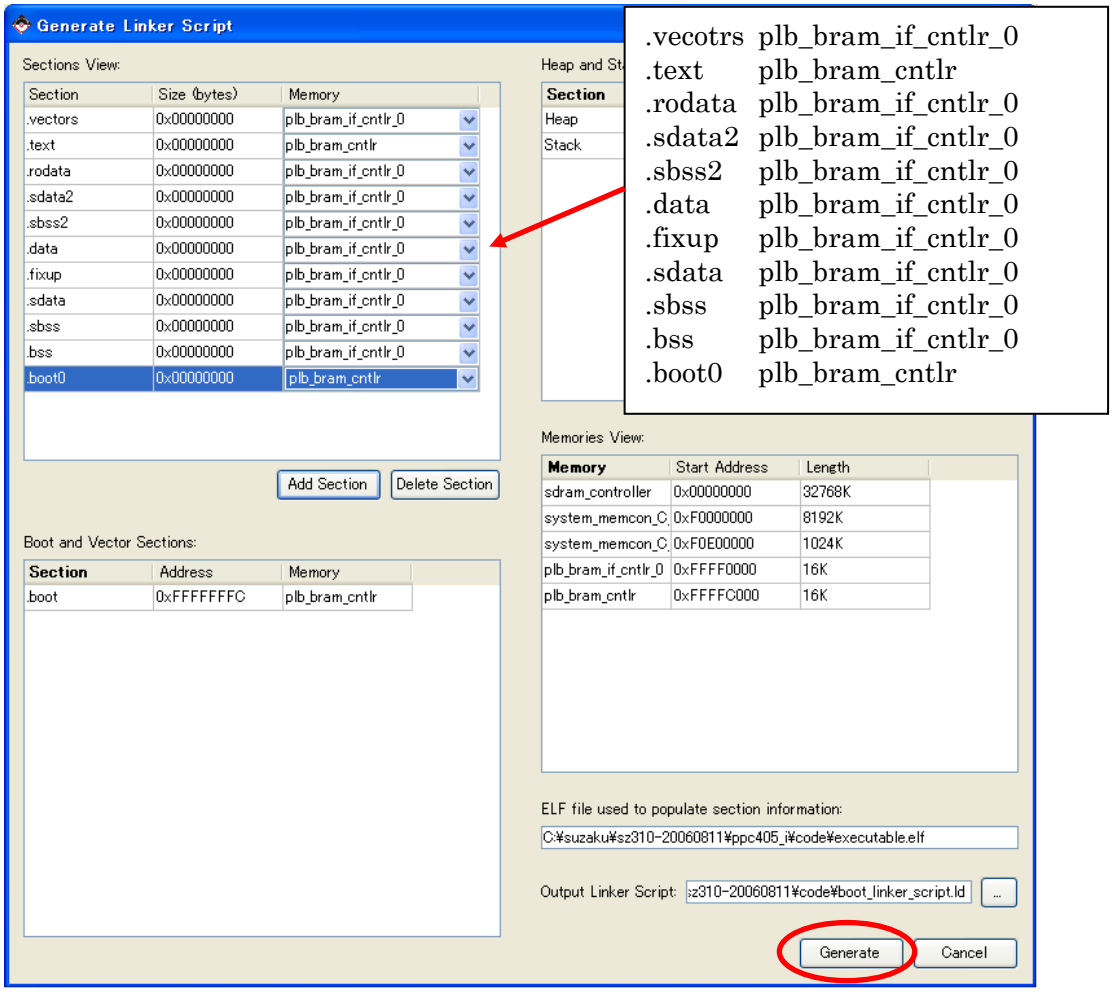


図 12-37 SZ310 割り込み設定 4

12.4.6. ピンアサイン

Project Files の UCF File:data/xps_proj.ucf をダブルクリックしてください。ピンアサインのファイルが開きます。ピンアサインを追加入力し、保存してください。

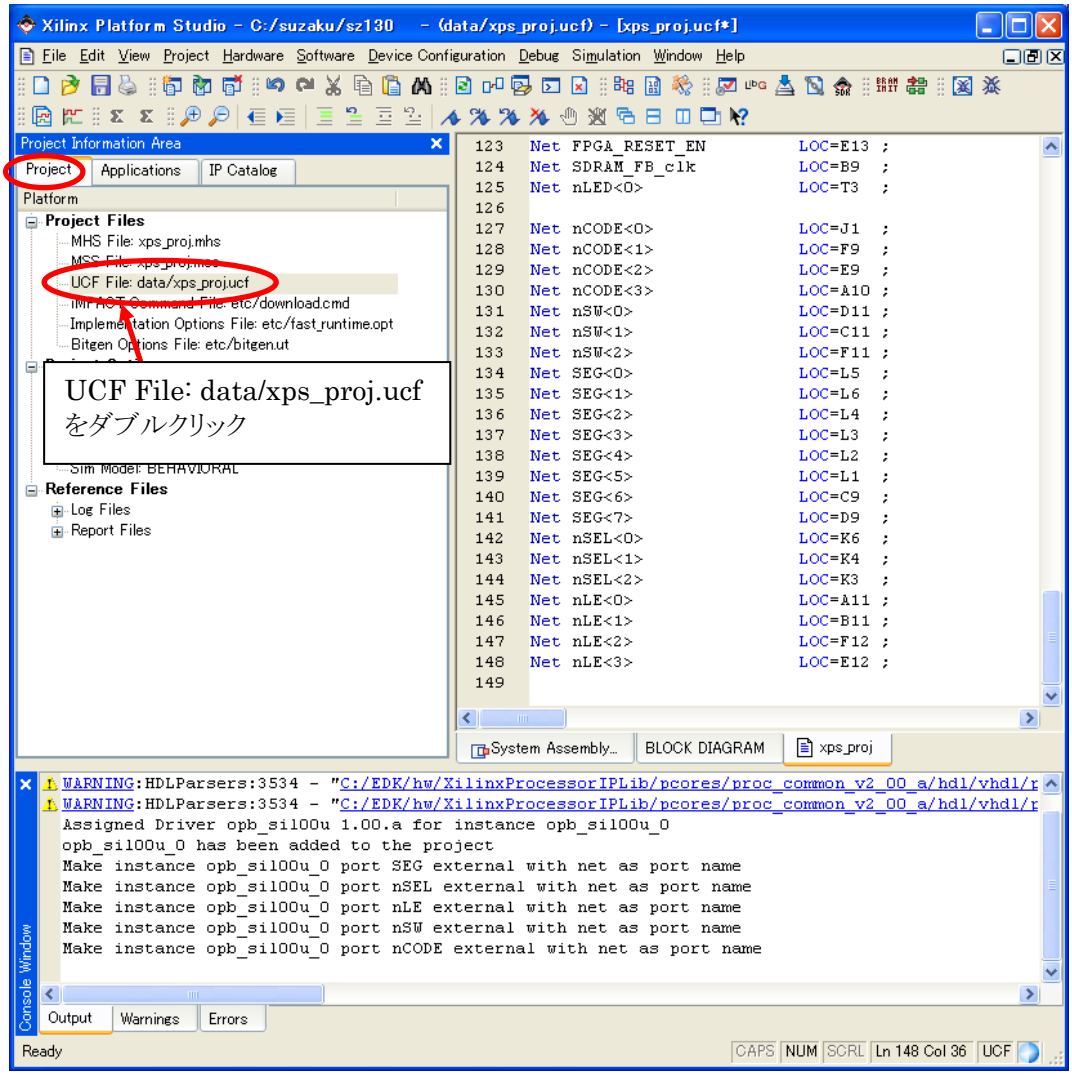


図 12-38 自作 IP コア(xps_proj.ucf)

表 12-2　　自作 IP コア ピンアサイン

	SZ010 SZ030	SZ130	SZ310
nCODE<0>	C8	J1	J16
nCODE<1>	A9	F9	J15
nCODE<2>	A12	E9	J14
nCODE<3>	C10	A10	J13
nSW<0>	A14	D11	K16
nSW<1>	B14	C11	K15
nSW<2>	A13	F11	K14
SEG<0>	C5	L5	F15
SEG<1>	B5	L6	F16
SEG<2>	E6	L4	G13
SEG<3>	D6	L3	G14
SEG<4>	C6	L2	G15
SEG<5>	B6	L1	G16
SEG<6>	A8	C9	N9
SEG<7>	B8	D9	P9
nSEL<0>	D7	K6	H13
nSEL<1>	C7	K4	H14
nSEL<2>	B7	K3	H15
nLE<0>	E11	A11	L13
nLE<1>	D11	B11	L14
nLE<2>	C12	F12	L15
nLE<3>	B12	E12	L16

これで自作 IP コアの追加が終了しました。自作 IP コアに間違いがないかチェックします。[Update BitStream]をクリックしてください。ネットリストの生成と、配置配線が行われ、bit ファイルが生成されます。

もし自作 IP コアにエラーがある場合、synthesis/opb_sil00u_0_wrapper_xst.srp にログが表示されるので、これを開いてエラーを確認し、修正を行い、再度[Update BitStream]をクリックしてください。

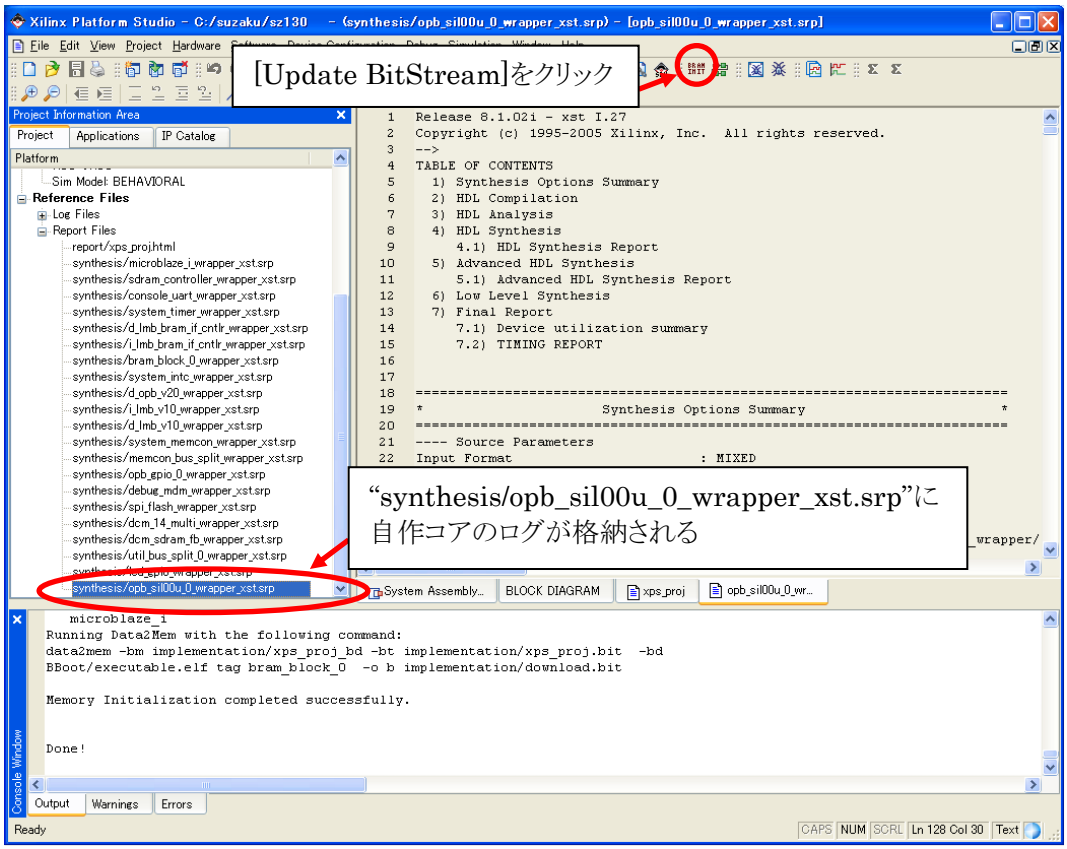


図 12-39 エラーレポート

12.5. CPU で制御する

12.5.1. BBoot

BBoot にスロットマシンのソフトウェアを追加します。BBoot の機能は、セカンドブートローダ（Hermit）のダウンロードと実行です。下図のように、BBoot にスロットマシンのソフトウェアを追加します。スロットマシンはビジーループモードと、割り込みモードの 2 通りで追加します。

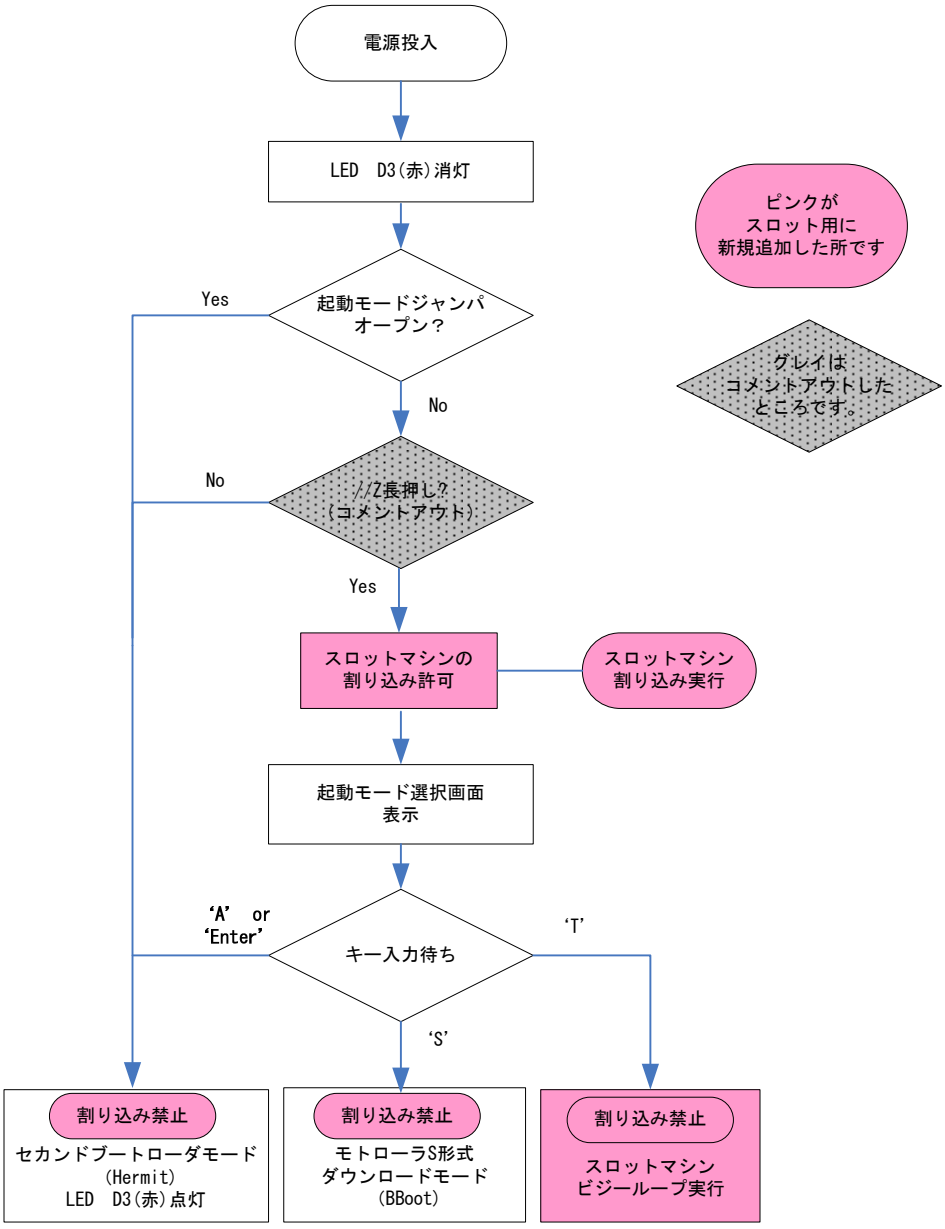


図 12-40 BBoot のフロー

12.5.2. プロジェクトにソースファイル追加

付属 CD-ROM の”¥suzaku-starter-kit¥fpga”の中の圧縮ファイル “slot_c_source.zip”を展開してください。
”C¥suzaku¥sz***-xxxxxxxx¥code”フォルダに展開後のフォルダの中の slot.c、interrupt.c、slot.h、interrupt.h をコピーしてください。

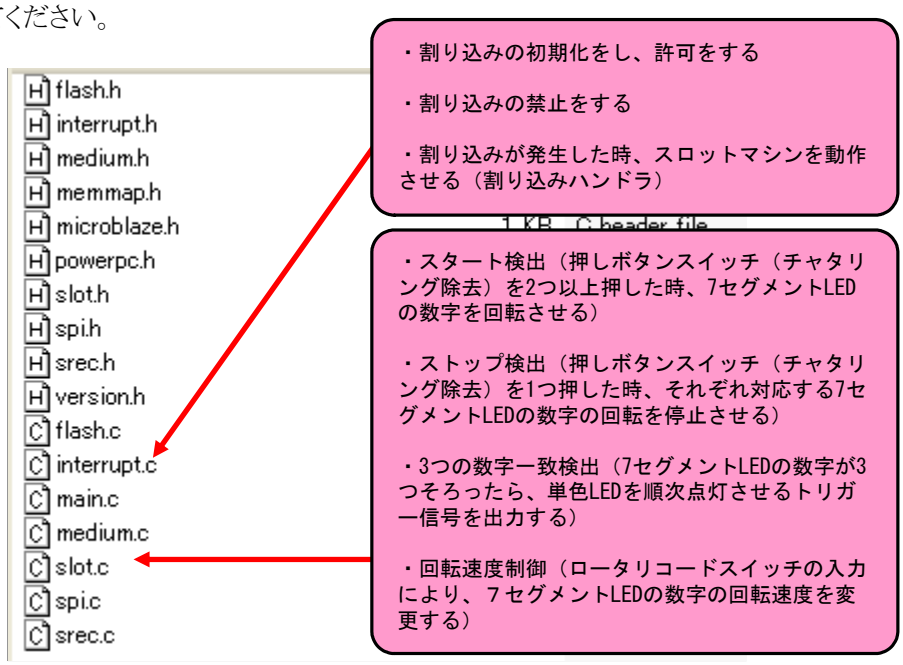


図 12-41 ソースファイルコピー

Applications の Sources を右クリックし、メニューの Add Existing Files...を選択し、Sources に slot.c、interrupt.c を追加してください。

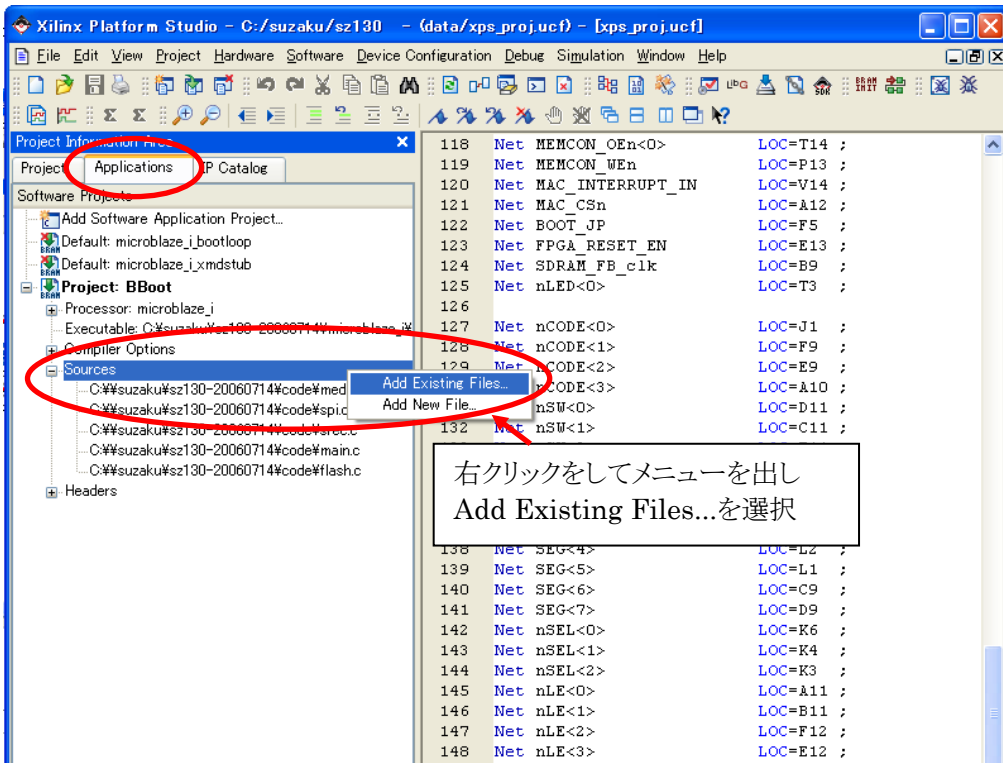


図 12-42 ソースファイル追加

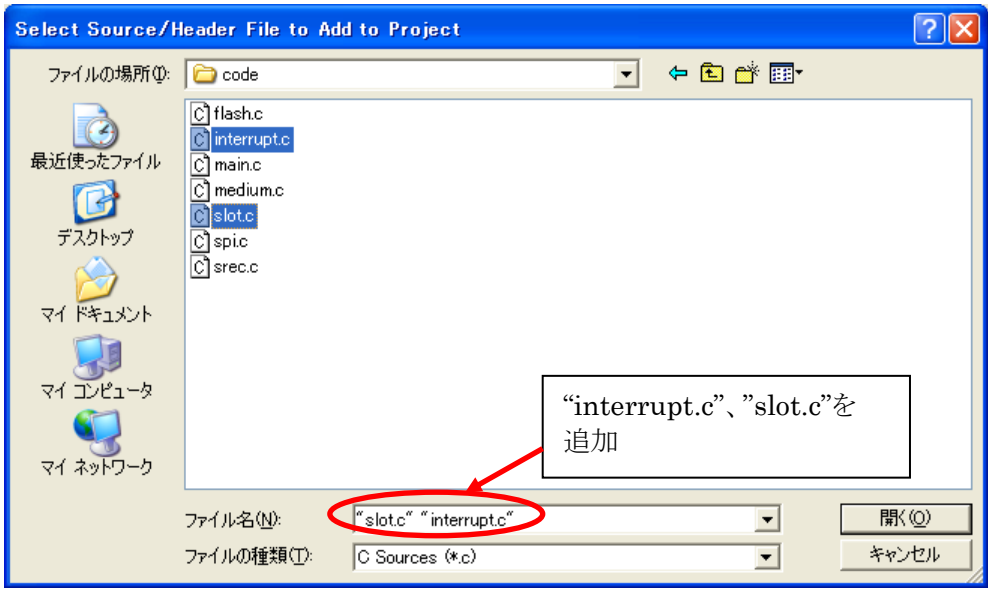


図 12-43 ソースファイル選択

Applications の Sources を右クリックし、メニューの Add Existing Files...を選択し、Headers に slot.h、interrupt.h を追加してください。

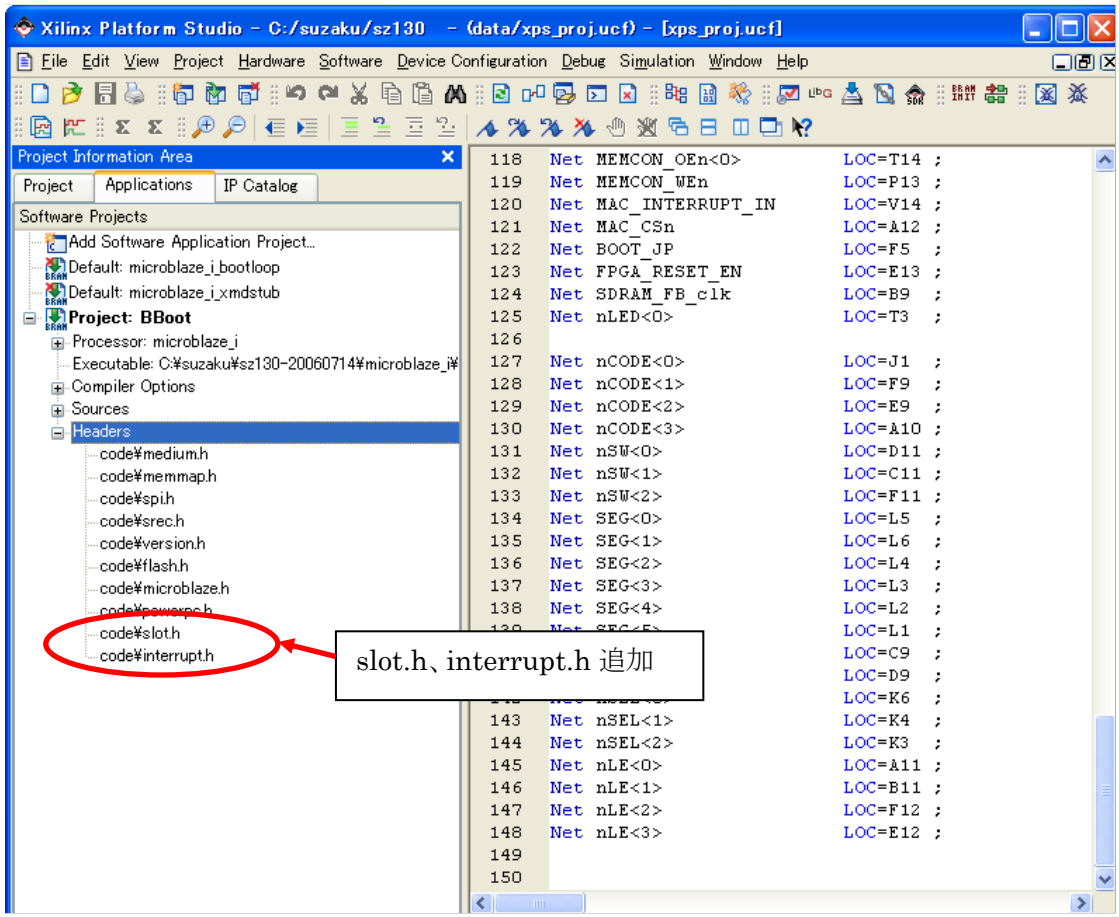


図 12-44 ヘッダファイル追加

12.5.3. 割り込みハンドラの登録

EDK では GUI 上から関数を設定するだけで割り込みベクタと割り込みハンドラのリンクが簡単に行えます。
[Software]→[Software Platform Settings]をクリックしてください。

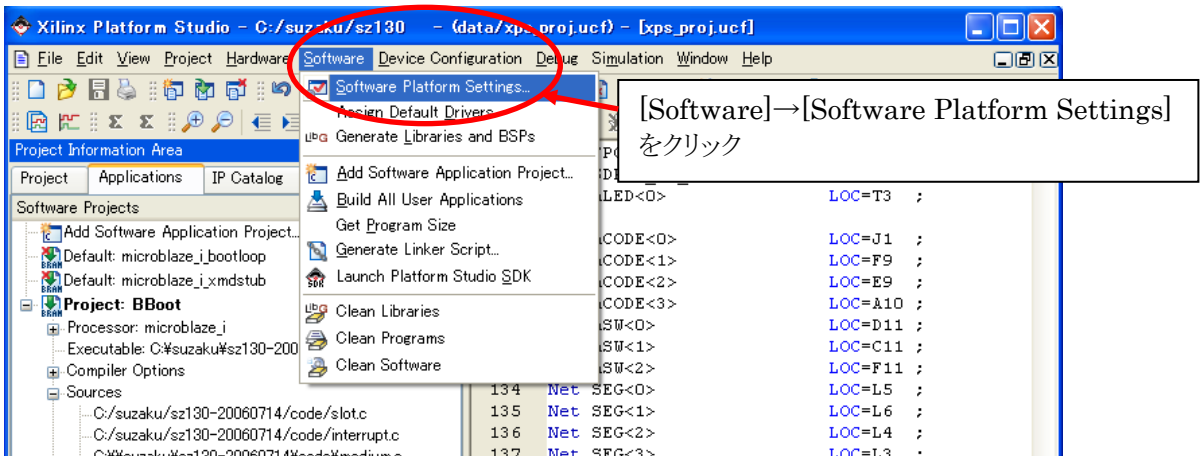


図 12-45 割り込み設定画面呼び出し

割り込みハンドラを登録します。
[Interrupt Handlers]を選択し、opb_sil00u_0 の Interrupt Handler に timer_interrupt_handler (interrupt.c にある関数)と入力し、[OK]をクリックして下さい。タイマ割り込みハンドラがリンクされます。

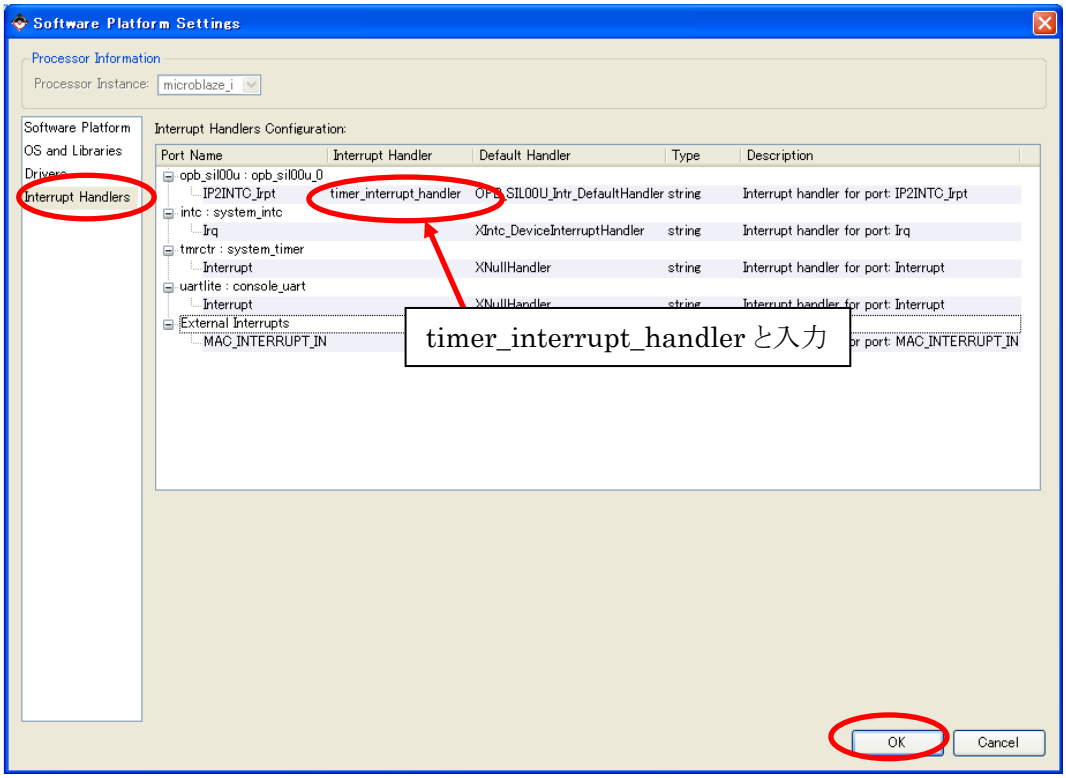


図 12-46 割り込み設定

12.5.4. BBoot のソース編集

main.c を編集します。Project: BBoot の main.c をダブルクリックして開いてください。

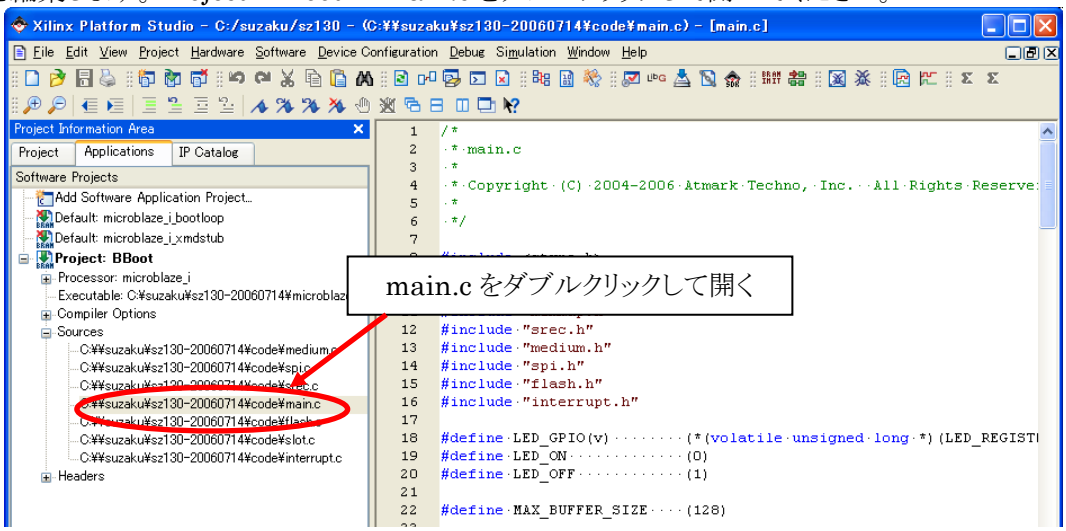


図 12-47 main.c を開く

● main.c

BBoot にスロットの機能を追加します。

例 12-6 自作 IP コア(main.c)

```
#include <ctype.h>
#include <xuartlite_l.h>
#include "version.h"
#include "memmap.h"
#include "srec.h"
#include "medium.h"
#include "spi.h"
#include "flash.h"

#include "slot.h"
#include "interrupt.h"

#define LED_GPIO(v)    (*(volatile unsigned long *)(LED_REGISTER_BASEADDR) = (v)
#define LED_ON                  (0)
#define LED_OFF                 (1)

#define MAX_BUFFER_SIZE        (128)

#ifdef   XPAR_SPI_FLASH_BASEADDR
#define BOOTLOADER_OFFSET_SPI  (0x00100000)
#else
#define FLASH_4MiB (0x16)
#define FLASH_8MiB (0x17)
#define BOOTLOADER_OFFSET_4MiB_FLASH (0x00080000)
#define BOOTLOADER_OFFSET_8MiB_FLASH (0x00100000)
#endif

//中略
```

```
int main(void)
{
    unsigned int bootloader_offset;
    char     key;

    LED_GPIO(LED_OFF);

#ifdef __PPC__
    XCache_DisableDCache();
    XCache_DisableICache();
    XCache_InvalidateICache();
    busy_wait(1000000);
#endif

    if (get_bootloader_offset(&bootloader_offset) < 0)
        goto halt;

    if (is_autoboot_mode()) {
        second_bootloader(bootloader_offset);
    }
//  myprint("¥r¥n¥r¥n"BBOOT_NAME " v" BBOOT_VERSION " (" TARGET_CPU ")¥r¥n");
//  myprint("Press 'z' or 'Z' for BBoot Menu.¥r¥n");

  /* busy loop to wait getting a char 'z' or 'Z' */

//  busy_wait(150000000);
//  if (XUartLite_mIsReceiveEmpty(XPAR_CONSOLE_UART_BASEADDR) ||
//                        ((key = get_char()) != 0 && key != 'z' && key != 'Z'))
//      second_bootloader(bootloader_offset);

    interrupt_init();

    /* clear for long time pushing */
    clear_rx_fifo();

    myprint("¥r¥n¥r¥nPlease choose one of the following and hit enter.¥r¥n");
    myprint("a: activate second stage bootloader (default)¥r¥n");
    myprint("s: download a s-record file¥r¥n");
    myprint("t: busy loop type slot-machine¥r¥n");

    while (1) {
        key = get_char();
        switch (key) {
            case 'a':    /* activate second stage bootloader */
            case 'A':
            case '¥r':
            case '¥n':
                interrupt_clean();
                second_bootloader(bootloader_offset);
                break;
            case 's':
            case 'S':
```

```
                interrupt_clean();
#ifndef XPAR_SPI_FLASH_BASEADDR
                myprint("Erasing Flash...");
                flash_erase(bootloader_offset);
                myprint("Done¥r¥n");
#endif
                myprint("Start sending S-Record!!¥r¥n");
                download();
                break;

            case 't':
            case 'T':
                interrupt_clean();
                myprint("busy loop type slot-machine¥r¥n");
                while(1){
                    busy_wait(700000);
                    slot();
                }

            default:
                myprint("Invalid selection.¥r¥n");
            case 'z':
            case 'Z':
                clear_rx_fifo();
                break;          }
    }
halt:
    myprint("Halting...¥r¥n");
    return 0;
}
```

12.5.5. コンフィギュレーション

mian.c の編集が終わったら、Update Bitstream をクリックしてください。ネットリストの生成と、配置配線が行われ、ビットファイルが生成されます。

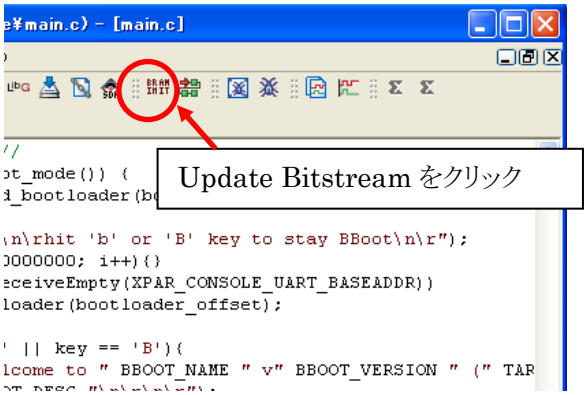


図 12-48 bit ファイル生成

JP1,JP2 をショートし、JTAG のコネクタを接続し、シリアルケーブルを向きに注意して接続してください。シリアル通信用ソフトウェアを立ち上げ、AC アダプタ 5V を接続して電源を入れてください。

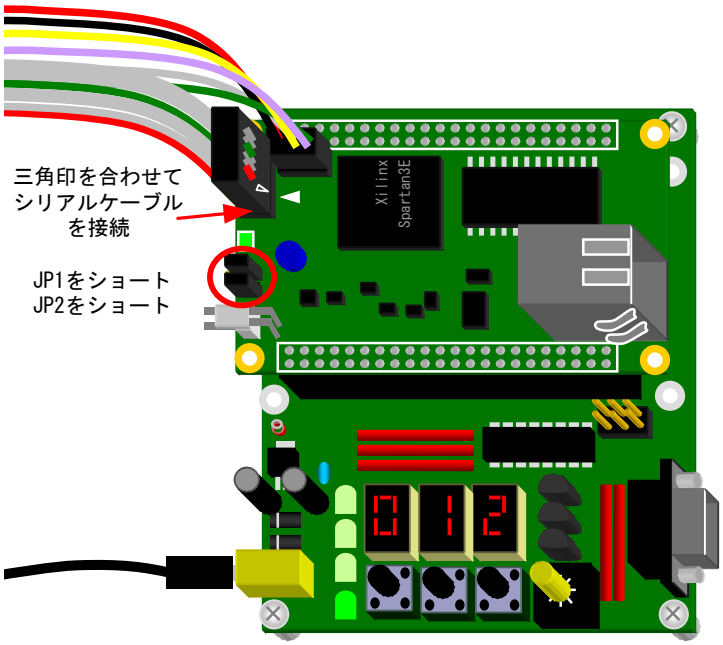


図 12-49 セッティング

Download Bitstream をクリックしてください。コンフィギュレーションされます。

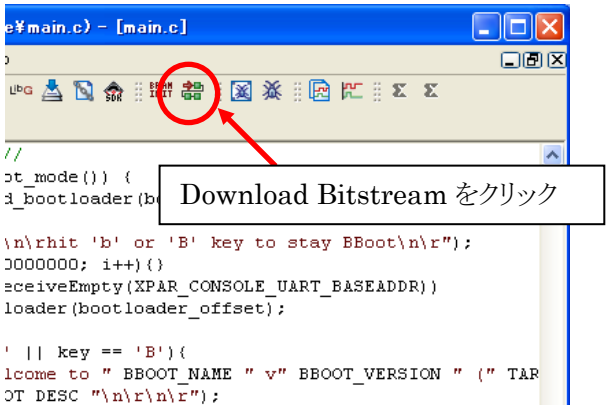


図 12-50 コンフィギュレーション

12.5.6. スロットマシン動作確認

スロットが割り込みモードで動きます。色々触って動きを確認してみてください。
下図のように表示されるので、”T”を押してください。

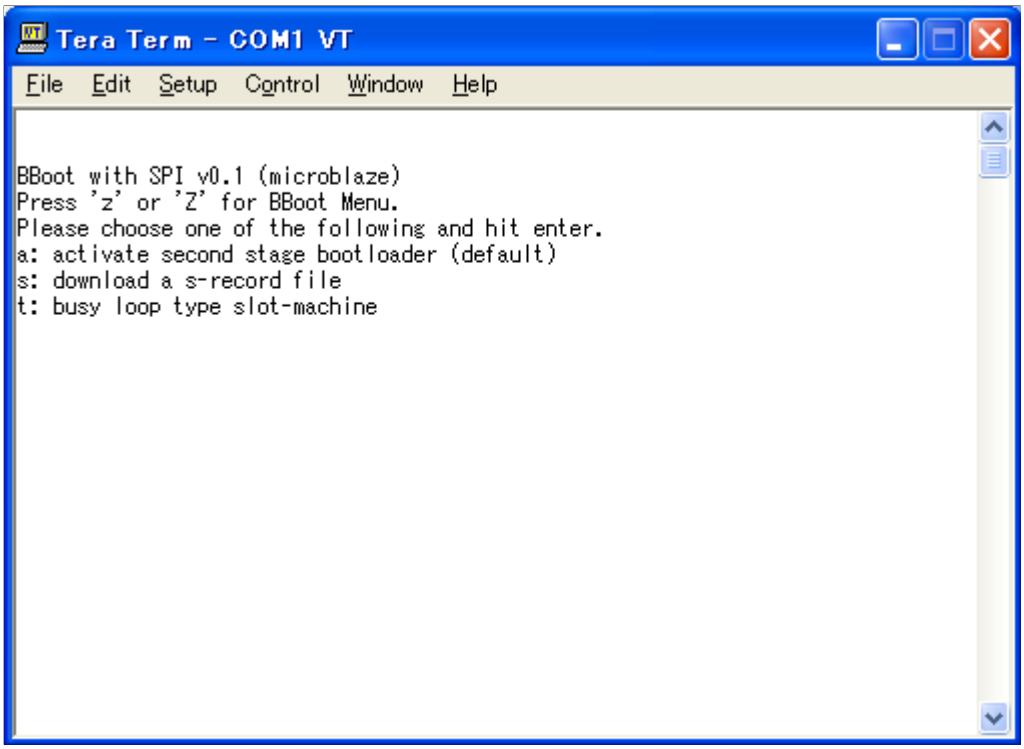


図 12-51 スロットマシン実行画面 1

スロットマシンがビジーループモードで動きます。スロットを色々触って動きを確かめてみてください。

スロットマシンの動きにはほとんど変わりはありませんが大きな違いが一つあります。さきほどまではシリアルコンソールでキー入力を受け付けていましたが、受け付けなくなっていると思います。割り込みでは同時に平行して複数の作業を行うことができます。割り込みが使えると、できる作業の幅がビジーループに比べ格段に増えます。

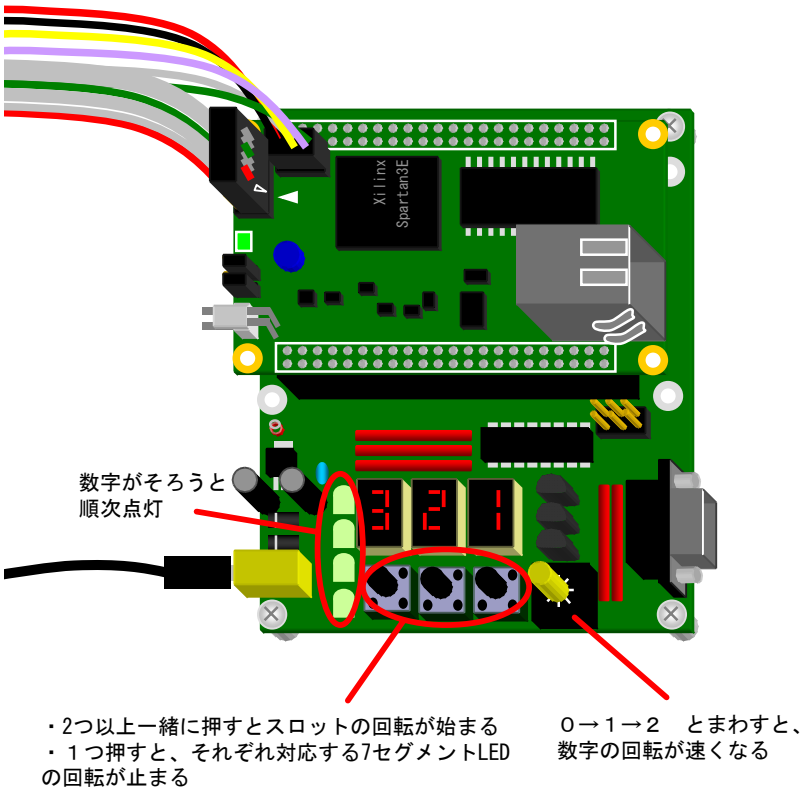


図 12-52 スロットマシン完成

12.6. スロットマシン完成

以上でスロットマシンが完成しました。FPGA 開発する上での基礎知識、ISE や EDK といった専用開発ツールの使い方、VHDL 言語の記述方法、FPGA に搭載される MicroBlaze の使用方法、そして SUZAKU の効果的な使い方は身についたでしょうか。本スターターキットを通して学んだことは、ほんの足掛かりにすぎません。ここからは自ら調べ、情報を仕入れ、勉強をし、アイデアを練り、SUZAKU 開発者のスペシャリストを目指してください。

13. こんなこともやってみよう

13.1. IP コア(ハード版)

先ほどソフトウェアで実現したスロットマシンの機能をハードウェアに置き換え、ソフトの負担を減らすことができます。

実はこちらの方法のほうが SUZAKU らしいやり方といえます。slot.vhd の中身の説明はしませんので、各自見て考えてみてください。ソフト版と違うのはカウンタのみで、他はほぼ同じ作りになっています。

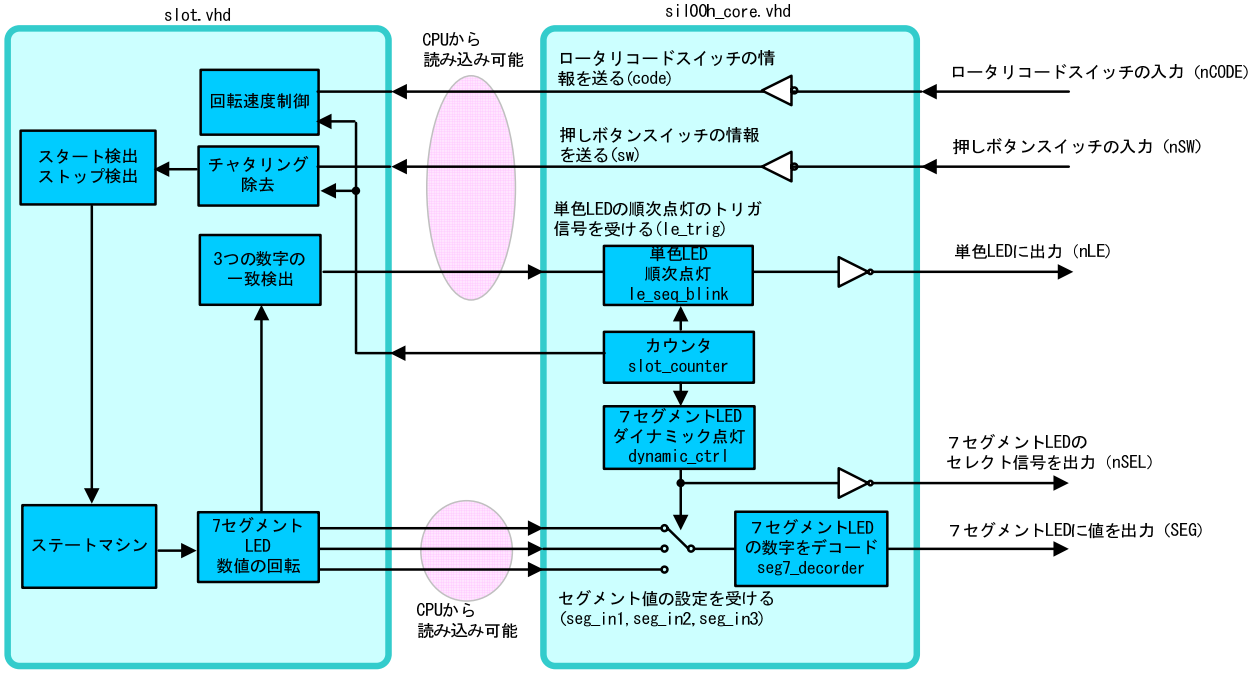


図 13-1 IP コア(ハード版)の仕様

“C¥suzaku¥sz***-xxxxxxxx”をコピーしてその場にペーストし、名前を変更してください。

ここでは”C¥suzaku¥sz***-h-xxxxxxxx”として作業を進めます。

付属 CD-ROM の”suzaku-starter-kit¥fpga¥sz***-sil”の中の圧縮ファイル”opb_sil00h_v1_00_a.zip”をハードディスクに展開してください。

展開後のフォルダ”opb_sil00h_v1_00_a”を”C¥suzaku¥sz***-h-xxxxxxxx¥pcores”にコピーしてください。

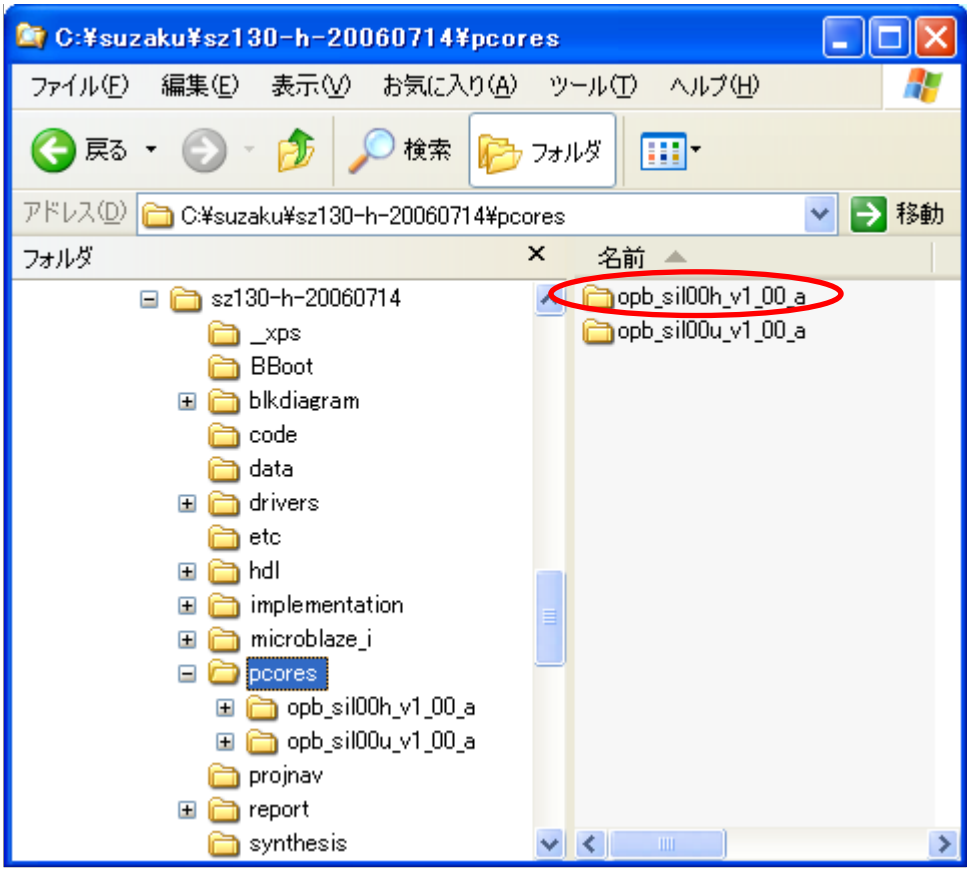


図 13-2 IP コア(ハード版)追加

”C:¥suzaku¥szz***-h-xxxxxxxx”の中の”xps_proj.xmp”を開いてください。
IP Catalog の Project Repository に opb_sil00h があるのを確認してください。

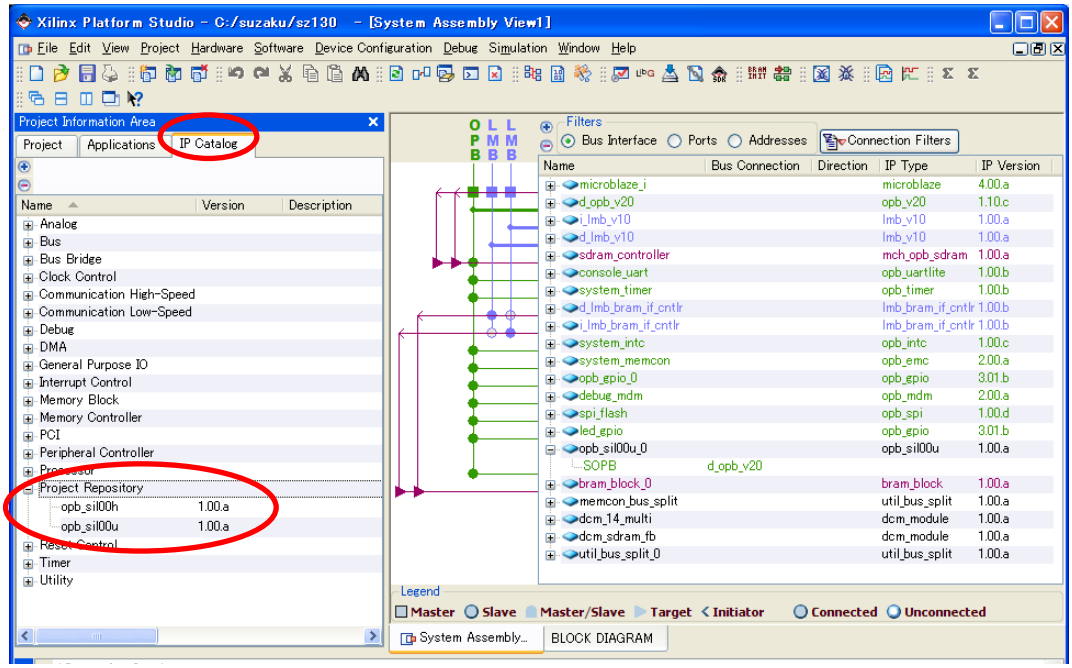


図 13-3 IP コア(ハード版)追加確認

自作 IP コア（ソフト版）opb_sil00u を IP コア（ハード版）opb_sil00h に置き換えます。
Project の MHS File: xps_proj.mhs をダブルクリックして開いてください。
BEGIN opb_sil00u と記述されているところを探して BEGIN opb_sil00h に変更し、PORT IP2INTC_Irpt = sil_intr と記述されているところを探して消去し、保存してください。

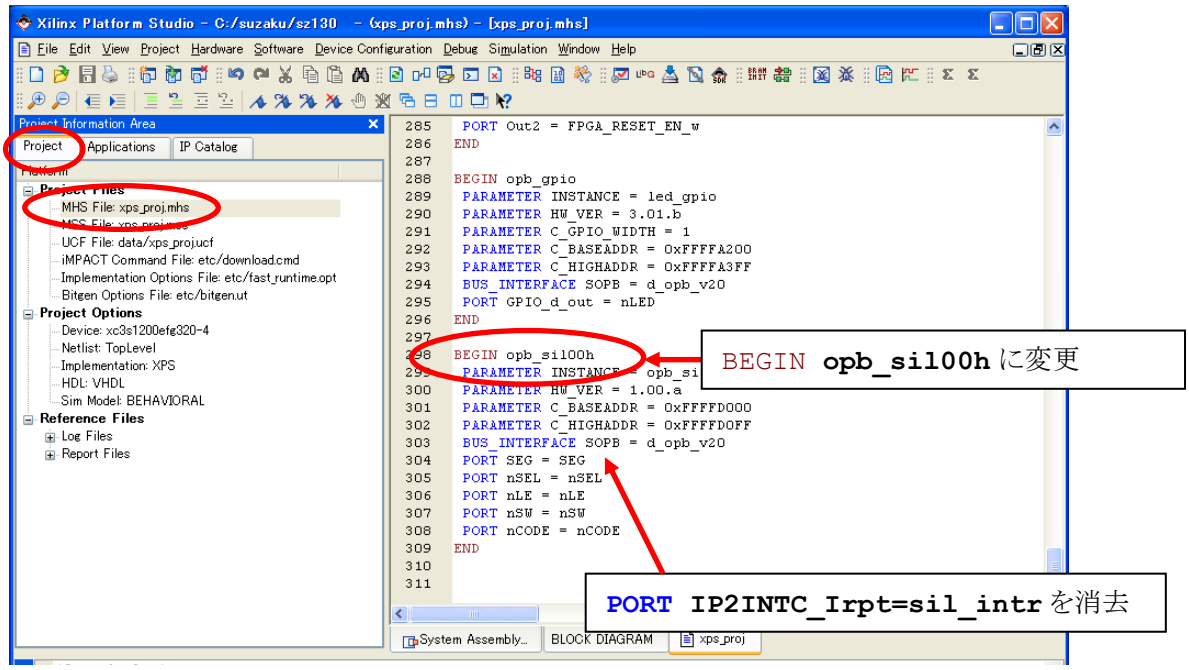


図 13-4 MHS File 変更

Project の MSS File: xps_proj.mss をダブルクリックして開いてください。
PARAMETER DRIVER_NAME = opb_sil00u と記述されているところを探し（1 ヶ所）、PARAMETER DRIVER_NAME = opb_sil00h に変更し、保存してください。
もし、PARAMETER DRIVER_NAME = generic と記述されている場合、変更はいりません。

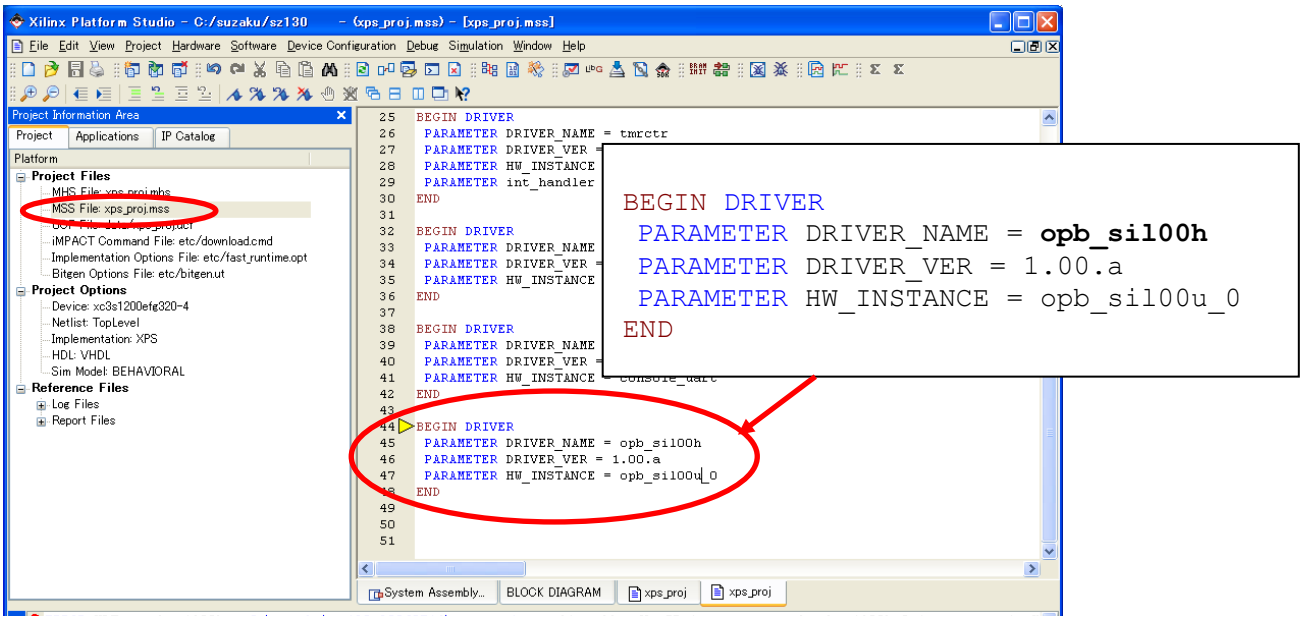


図 13-5 MSS File 変更

以上で IP コアが置き換わりました。
IP Type が opb_sil00h に変更されます。

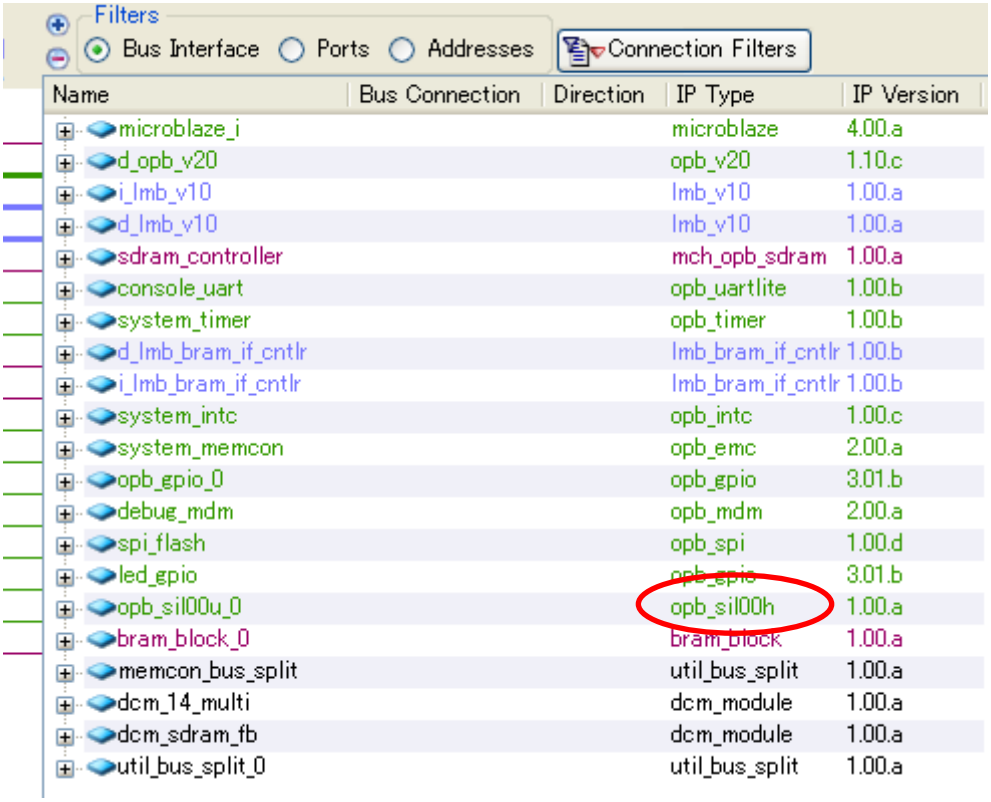


図 13-6 IP コア(ハード版)に置き換え

コンフィギュレーションしてください。数字の回転が少し速いですが、ソフト版とほぼ同じスロットマシンが動きます。

13.2. CGI で 7 セグメント LED をコントロール

自分で作ったスロットマシンの IP コアを CGI でコントロールします。
“C:¥suzaku¥sz***-xxxxxxxx¥implementation”の中にある download.bit をつかって Flash を書き換えてください。

Flash の中に入っている Linux では最初から CGI が動作しています。

(Flash の中の Linux を書き換えてしまっている場合は、Flash の image を書き直して下さい。i image は付属 CD-ROM の”¥suzaku-starter-kit¥image”の中の image-sz***-sil.bin を使ってください。Flash の中の Linux を書き換える方法については、「SUZAKU Software Manual」を参照してください。

シリアル通信ソフトウェアを起動後、SUZAKU スターターキットの JP1、JP2 をオープンにして電源を投入してください。Linux が起動するので、ネットワークの設定をしてください。

IP アドレスを確認し、お使いのブラウザで”http://IP アドレス/7seg-led-control.cgi”にアクセスしてください。スロットマシンの7セグメント LED の回路がブラウザから制御できます。

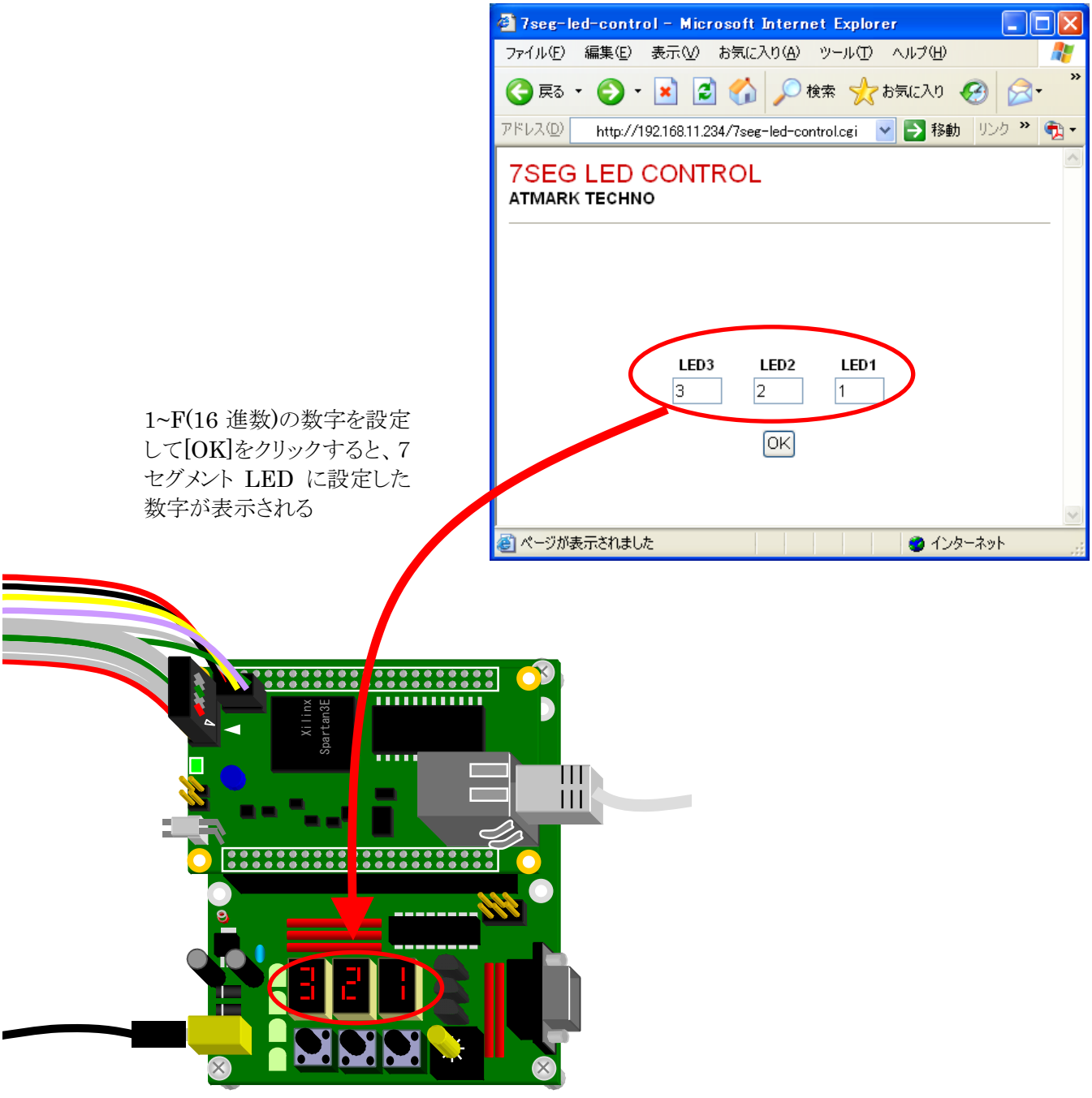


図 13-7 自作のコアをコントロール

これは、以下のソースコードで CGI を作成することにより実現しています。
コンパイル方法や、Flash に書き込むためのデータの作成方法、および実際の書込み方法については、
「SUZAKU_Software Manual」、「uClinux-dist Developers Guide」を参照してください。

■ 7seg-led-control.c

例 13-1 CGI で 7 セグメント LED をコントロール(7seg-led-control.c)

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define PROGRAM_NAME              "7seg-led-control"
#define CGI_PATH          PROGRAM_NAME".cgi"

#define DEV_NAME                "/dev/sil7segc"

#define FORM_OK_BUTTON            "ok_button"
#define FORM_LED1_TEXT_BOX        "led1"
#define FORM_LED2_TEXT_BOX        "led2"
#define FORM_LED3_TEXT_BOX        "led3"

static void print_content_type(void)
{
        printf("Content-Type:text/html¥n¥n");
}

static void print_style_sheet(void)
{
        printf("<style type=¥"text/css¥">¥n¥n");

        printf("body {¥n");
        printf("margin: 0 0 0 0;¥n");
        printf("padding: 10px 10px 10px 10px;¥n");
        printf("font-family: Arial, sans-serif;¥n");
        printf("background: #ffffff;¥n");
        printf("}¥n¥n");

        printf("h1 {¥n");
        printf("margin: 0 0 0 0;¥n");
        printf("padding: 0 0 0 0;¥n");
        printf("color: #cc0000;¥n");
        printf("font-weight: normal;¥n");
        printf("}¥n¥n");

        printf("h2 {¥n");
        printf("margin: 0 0 0 0;¥n");
        printf("padding: 0 0 0 0;¥n");
        printf("font-size: 14px;¥n");
```

```
        printf("}¥n¥n");

        printf("hr {¥n");
        printf("height: 1px;¥n");
        printf("background-color: #999999;¥n");
        printf("border: none;¥n");
        printf("margin: 5px 0 70px 0;¥n");
        printf("}¥n¥n");

        printf(".leds {¥n");
        printf("font-size: 12px;¥n");
        printf("font-weight: bold;¥n");
        printf("line-height: 20px;¥n");
        printf("}¥n¥n");

        printf("</style>¥n¥n");
}

static void print_html_head(void)
{
        printf("<!DOCTYPE html PUBLIC ¥"-//W3C//DTD XHTML 1.0 Transitional//EN¥"
                        ¥"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd¥">¥n");
        printf("<html xmlns=¥"http://www.w3.org/1999/xhtml¥" lang=¥"ja¥" xml:lang=¥"ja¥">¥n¥n");

        printf("<head>¥n¥n");

        printf("<meta http-equiv=¥"content-type¥" content=¥"text/html; charset=utf-8¥"/>¥n¥n");
        printf("<title>%s</title>¥n¥n", PROGRAM_NAME);

        print_style_sheet();

        printf("</head>¥n¥n");

        printf("<body>¥n¥n");
}

static void print_html_tail(void)
{
        printf("</body>¥n¥n");
        printf("</html>¥n");
}

static void display_page(int fd)
{
        unsigned char leds[3];

        read(fd, leds, 3);

        print_content_type();

        print_html_head();

        printf("<h1>7SEG LED CONTROL</h1>¥n");
```

```
        printf("<h2>ATMARK TECHNO</h2>¥n¥n");

        printf("<hr />¥n¥n");

        printf("<form action=¥"%s¥" method=¥"get¥">¥n¥n", CGI_PATH);

        printf("<table border=¥"0¥" cellpadding=¥"10¥" cellspacing=¥"0¥" width=¥"200px¥"
                                                        align=¥"center¥" class=¥"leds¥">¥n");

        printf("<tr>¥n");

        printf("<td align=¥"center¥">");

        printf("LED3<br />¥n");
        printf("<input type=¥"text¥" name=¥"%s¥" value=¥"%x¥" size=¥"1¥" maxlength=¥"1¥" />¥n",
                                                                FORM_LED3_TEXT_BOX, leds[2]);

        printf("</td>¥n<td align=¥"center¥">");

        printf("LED2<br />¥n");
        printf("<input type=¥"text¥" name=¥"%s¥" value=¥"%x¥" size=¥"1¥" maxlength=¥"1¥" />¥n",
                                                                FORM_LED2_TEXT_BOX, leds[1]);

        printf("</td>¥n<td align=¥"center¥">");

        printf("LED1<br />¥n");
        printf("<input type=¥"text¥" name=¥"%s¥" value=¥"%x¥" size=¥"1¥" maxlength=¥"1¥" />¥n",
                                                                FORM_LED1_TEXT_BOX, leds[0]);

        printf("</td>¥n");

        printf("</tr><tr>¥n");

        printf("<td colspan=¥"3¥" align=¥"center¥">¥n");

        printf("<input type=¥"submit¥" value=¥"OK¥" name=¥"%s¥" />¥n", FORM_OK_BUTTON);

        printf("</td>¥n");

        printf("</tr>¥n");

        printf("</table>¥n¥n");

        printf("</form>¥n¥n");

        print_html_tail();
}

static unsigned int get_query_pair_hex_value(char *query, char *query_pair_name)
{
        char *pair_start, *pair_value;
        unsigned int hex_value = 0;
```

```
        pair_start = strstr(query, query_pair_name);
        if (pair_start) {
                pair_value = strchr(pair_start, '=') + 1;
                if (pair_value) {
                        sscanf(pair_value, "%x", &hex_value);
                }
        }

        return hex_value;
}

static void handle_query(int fd)
{
        char *query;
        unsigned char leds[3];

        query = getenv("QUERY_STRING");
        if (!query) {
                return;
        }

        if (!strstr(query, FORM_OK_BUTTON)) {
                return;
        }

        leds[0] = (unsigned char) get_query_pair_hex_value(query, FORM_LED1_TEXT_BOX);
        leds[1] = (unsigned char) get_query_pair_hex_value(query, FORM_LED2_TEXT_BOX);
        leds[2] = (unsigned char) get_query_pair_hex_value(query, FORM_LED3_TEXT_BOX);
        write(fd, leds, 3);
}

int main(int argc, char *argv[])
{
        int fd;

        fd = open(DEV_NAME, O_RDWR);
        handle_query(fd);
        display_page(fd);
        close(fd);

        exit(EXIT_SUCCESS);
}
```

13.3. 最新版のダウンロード

本マニュアルで紹介いたしましたソースコードやファイルは、不具合解決や機能増強等のアップグレードを行うことがあります。

下記サイトに最新版がございますのでダウンロードしてお使いください。

開発に関するファイル	http://suzaku.atmark-techno.com/downloads/all
各種マニュアル	http://suzaku.atmark-techno.com/downloads/docs

● 本書記載の社名、製品名について
本書に記載されている社名、および製品名は、一般的に開発メーカの登録商標です。
なお、本文中では ™、®、©の各名称を明記しておりません。

改訂履歴

Ver.	年月日	改訂内容
1.0.0	2006 年 7 月 14 日	初版作成
1.0.1	2006 年 7 月 19 日	誤記訂正
1.0.2	2006 年 7 月 24 日	ピンアサイン訂正（CON4 の 9、10 ピン）
2.0.0	2006 年 8 月 11 日	SZ010、SZ030、SZ310 対応のための全面変更（以下重要な変更のみ記） “JTAG Clock に変更する”の記載を消去 BBoot の変更、CD-ROM の内容変更 自作コアに割り込み機能追加
2.0.1	2006 年 8 月 18 日	TE7720 の図の文字化けを修正
2.0.2	2006 年 8 月 23 日	誤記訂正

SUZAKU スターターキットガイド(FPGA 開発編) 2006 年 8 月 23 日 version 2.0.2

株式会社アットマークテクノ
060-0035 札幌市中央区北 5 条東 2 丁目 AFT ビル 6F 011-207-6550 FAX: 011-207-6570