SUZAKU
スターターキットガイド(FPGA 開発編)

IDS11.5 対応
SZ130-SIL
SZ410-SIL

Version 3.0.0
2010/09/24

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]

SUZAKU スターターキットガイド(FPGA 開発編)

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
TEL 011-207-6550 FAX 011-207-6570

製作著作 © 2006-2010 Atmark Techno, Inc.

Version 3.0.0
2010/09/24

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 ポートが大量に欲しい、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す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、フラッシュメモリ、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 用の電源を作る回路が組み込まれています。また、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。
- フラッシュメモリの中には、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。

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等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できます。

"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 1. FPGA 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。2. ソフトウェア開発については、本書と対となる「SUZAKU スターターキットガイド(Linux 開発編)」をご参照ください。

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。

1. 本書について

1.1. 対象となる読者

本書は SUZAKU の FPGA 開発者向けに書かれた入門書です。SUZAKU の FPGA には初めからプロセッサが搭載されており、ディジタル回路、プロセッサ、バス、メモリ等様々な要素が絡み合ってきます。このため、どこで何が行われているのか分からない、やりたいことがあっても、それを実現するためにどうすればよいのか分からないという方もおられると思います。

どこから手をつけていいか分からない方、SUZAKU をはじめて使う方に対して、SUZAKU での FPGA 開発方法について詳しく説明します。

1.2. 対象となる SUZAKU

本書は SZ130-U00、SZ410-U00 を対象に書かれています。他の SUZAKU(SZ010-U00、SZ030-U00、SZ310-U00)をご使用の場合は、ISE/EDK10.1i 版以前のスターターキットガイド(FPGA 開発編)をご参照ください。

1.3. 対象となる Xilinx ソフトウェアのバージョン

本書は ISE Design Suite11.5i を使用して書かれています。違うバージョンのソフトウェアをお使いの場合は、内容が合致しない場合がありますので、ご注意ください。

1.4. 本書の構成

本書では、SUZAKU スターターキットを使用してスロットマシンを製作しながら、SUZAKU の使い方について解説してきます。内容は 3 部構成となっています。

第 1 部では SUZAKU で FPGA 開発を行うために必要な知識や準備について説明をします。SUZAKU および LED/SW ボードについて(第 1、2、3 章)と、作業の前に必要な準備と簡単な SUZAKU の使い方(第 4、5、6 章)を説明します。

第 2 部では Xilinx の ISE というツールを用い、プロセッサを含まない FPGA の開発を、実際に SUZAKU スターターキットを用いて体験します。まず、単色 LED を 1 つだけ点灯する簡単な回路の作成から始まり(第 7 章)、組み合わせ回路、順次回路の説明をし(第 8 章)、ISE 付属のシミュレータで回路の動きを確認し(第 9 章)、これらを踏まえて、スロットマシンの要素となる回路(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)を作成していきます(第 10 章)。

第 3 部では Xilinx の EDK というツールを用い、プロセッサを含んだ FPGA の開発を実際に SUZAKU スターターキットを用いて体験します。まずは EDK がどのようなツールであるのかを説明し(第 11 章)、SUZAKU のデフォルトの構成とカスタマイズ方法を説明します(第 12 章)。その後、ISE で作成した回路を IP コアにして SUZAKU と接続し、スロットマシンを完成させます(第 13 章)。最後に、こんなこともやってみよう、という例を示します(第 14 章)。

スロットマシンを最後までつくり上げる頃には、SUZAKU の効果的な使い方を学んでいただけるのではないかと思います。

1.5. 表記について

本書は SZ130-U00、SZ410-U00 を対象に書かれています。内容によっては両方に当てはまらない場合があります。当てはまらないものがある場合は以下の記号で対象となる SUZAKU を示します。

また、これ以降型番の-U00 を省略して表記します。

本書では以下のようにフォントを使っています。

フォント例	説明
本文中のフォント	本文で使用しています。
太字	太字は読者が入力すべき箇所を示します。
`std_logic_vector`	ソースコード記述やプロンプトで使用しています。

ヒント

重要ではありませんが、知っていると便利な情報などは、このアイコンで示します。

注意

注意しなければいけない作業などは、このアイコンで示します。作業する場合には十分に気をつけて行なってください。

1.6. お問い合わせ先

本書に関するご意見やご質問は SUZAKU メーリングリスト [http://suzaku.atmark-techno.com/dev/mailinglist]にご連絡ください。

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。

発熱により高温になる部品があります。周囲温度や取り扱いによってはやけどの恐れがあります。電源が入っている状態および電源切断後しばらくは本製品に触れない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%以上の電圧を入力する、極性を間違う等しないでください。また、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、入出力方向を間違わないでください。

- 本製品の改造

  本製品について改造を行った場合は保証対象外となりますので、十分にご注意ください。(※コネクタ非搭載箇所へのコネクタの増設を除く。)

  コネクタを増設する際にはマスキングを行い、周囲の部品に半田くず、半田ボール等付着しない様十分にご注意ください。

  なお、改造を行う場合は、改造前の動作確認を必ず行うようお願いします。

- 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 CMOS 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 CMOS 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や、強い振動、遠心力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。

- 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

FPGA 使用に関して以下の点にご注意ください。

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 ISE Design Suite 等)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。

目次

図目次

表目次

例目次

1.SUZAKU について

初めに"SUZAKU"がどのようなボードであるのか簡単に説明します。SUZAKU の詳細については本書内いたるところにちりばめられていますので、ここでは概要をつかんでください。

1.1. SUZAKU の特徴

SUZAKU(朱雀)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。

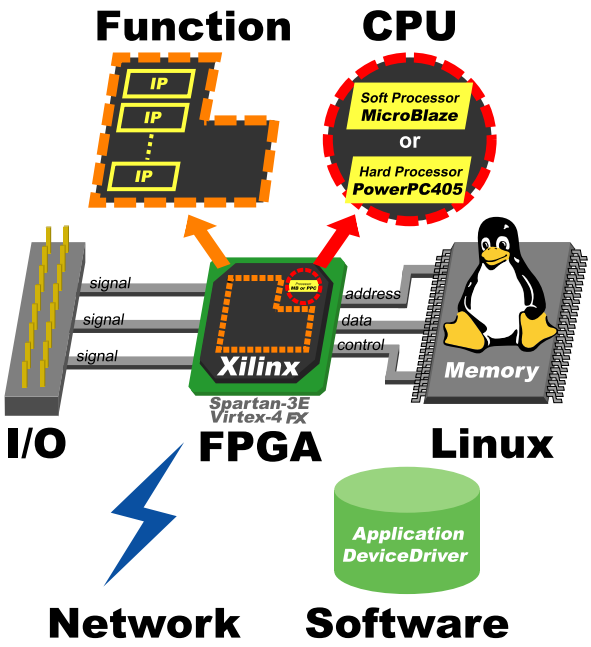
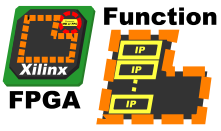
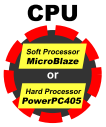


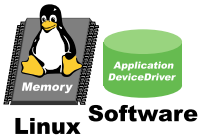
図 1.1 SUZAKU とは



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。SZ130 は Spartan-3E、SZ410 は Virtex-4 FX を搭載しています。FPGA の内部は Xilinx やサードパーティ各社から供給される IP(Intellectual Property)を使用することで、必要な機能を容易に追加することが出来ます。また、ユーザによってもカスタマイズが可能です。

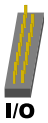


SZ130 は低コストで資産継承性の高いソフトプロセッサの MicroBlaze を採用し、SZ410 は高性能で実績の高いハードプロセッサの PowerPC405 を採用しています。



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。SZ130 は MMU(メモリ管理ユニット)不要の μClinux、SZ410 は標準的な Linux に対応しています。デバイスドライバか

ら各種サーバソフトウェアまで、オープンソースで開発された Linux 対応の豊富なソフトウェア資産を活用することが出来ます。



基板外周に 86 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。

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。

MicroBlaze

MicroBlaze は Xilinx が提供する 32 ビット RISC コアです。MicroBlaze のブロック図を以下に示します。

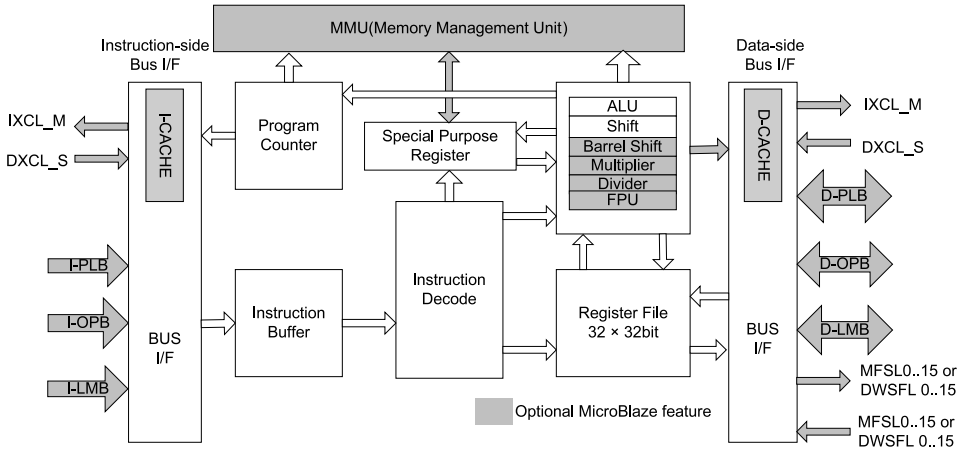


図 1.2 MicroBlaze ブロック図

主な機能

- 3 個のオペランドおよび 2 個のアドレス指定モードのある 32bit 命令ワード
- 32 個の汎用 32bit レジスタ
- 単一パイプライン
- ハードウェアデバッグロジック対応

1.2. SUZAKU のメリット

SUZAKU のメリットは色々ありますが、やはり複雑なハードウェアを容易に実現できるということが、一番のメリットです。一般的な SoC であればありえない構成であっても、回路規模が許す限り好きなように実現することができます。

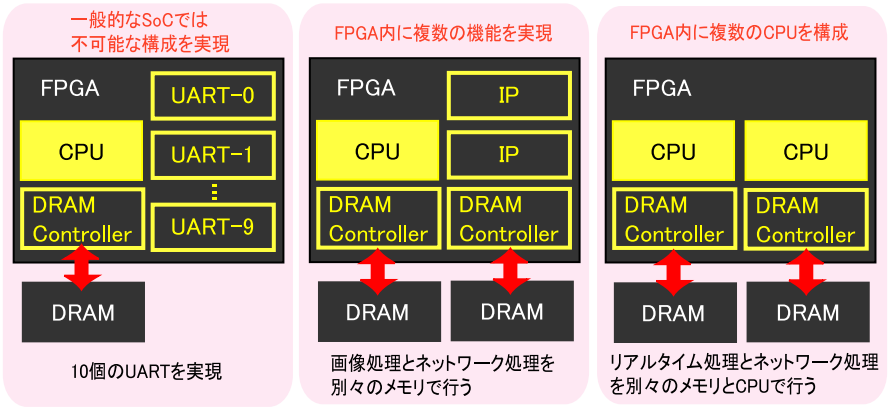


図 1.3 複雑なハードウェアを容易に実現

また、最新の FPGA が発表されたとしても、新しいデバイスに置き換えるだけで、同じ機能を実現させることができます。過去の資産を継続して利用しつづけることが可能であり、長期利用することができます。

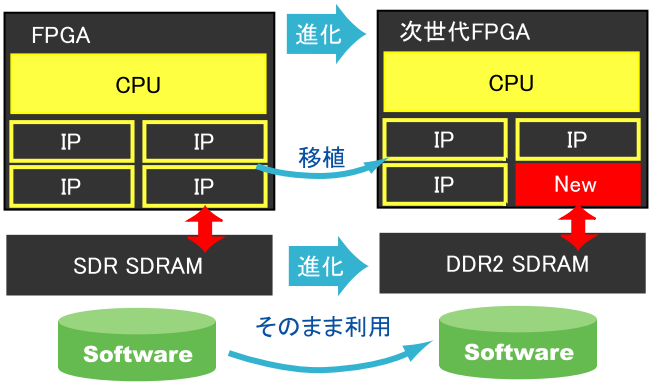


図 1.4 長期利用が可能

基本構成はそのままに、共通のプラットフォームとして利用可能であるということも、SUZAKU のメリットの一つです。

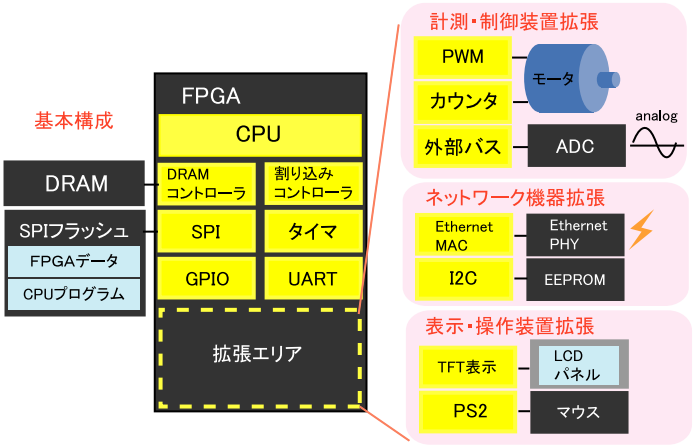
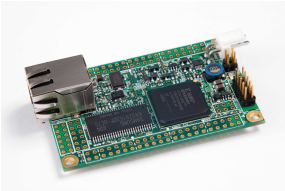
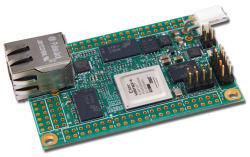


図 1.5 共通プラットフォームとして

1.3. 仕様

SUZAKU の主な仕様を以下に示します。

表 1.1 SUZAKU の仕様

	SUZAKU-S	SUZAKU-V
モデル	 図 1.6 SZ130	 図 1.7 SZ410
FPGA デバイス	Xilinx Spartan-3E (XC3S1200E)	Xilinx Virtex-4 FX (XC4VFX12)
CPU コア	MicroBlaze	PowerPC405
CPU クロック	51.6096MHz	350MHz
水晶発振器周波数	3.6864MHz	100MHz
DRAM	16MB×2	32MB×2
フラッシュメモリ	8MB (SPI)	8MB (SPI)
Ethernet	10BASE-T/100BASE-TX	
拡張 I/O ピン	86	
シリアル	1ch(UART : 115.2kbps)	
タイマ	2ch(1ch は OS で使用)	PowerPC 内蔵タイマ
コンフィギュレーション	SPI フラッシュメモリ	
基板サイズ	72x47mm	
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

1.4. 全体ブロック図

SUZAKU の全体ブロック図について説明をします。IP コアについてはここでは説明しません。「12.1. SUZAKU のデフォルト」をご参照ください。

1.4.1. SZ130

SZ130 の全体のブロック図は以下のとおりです。

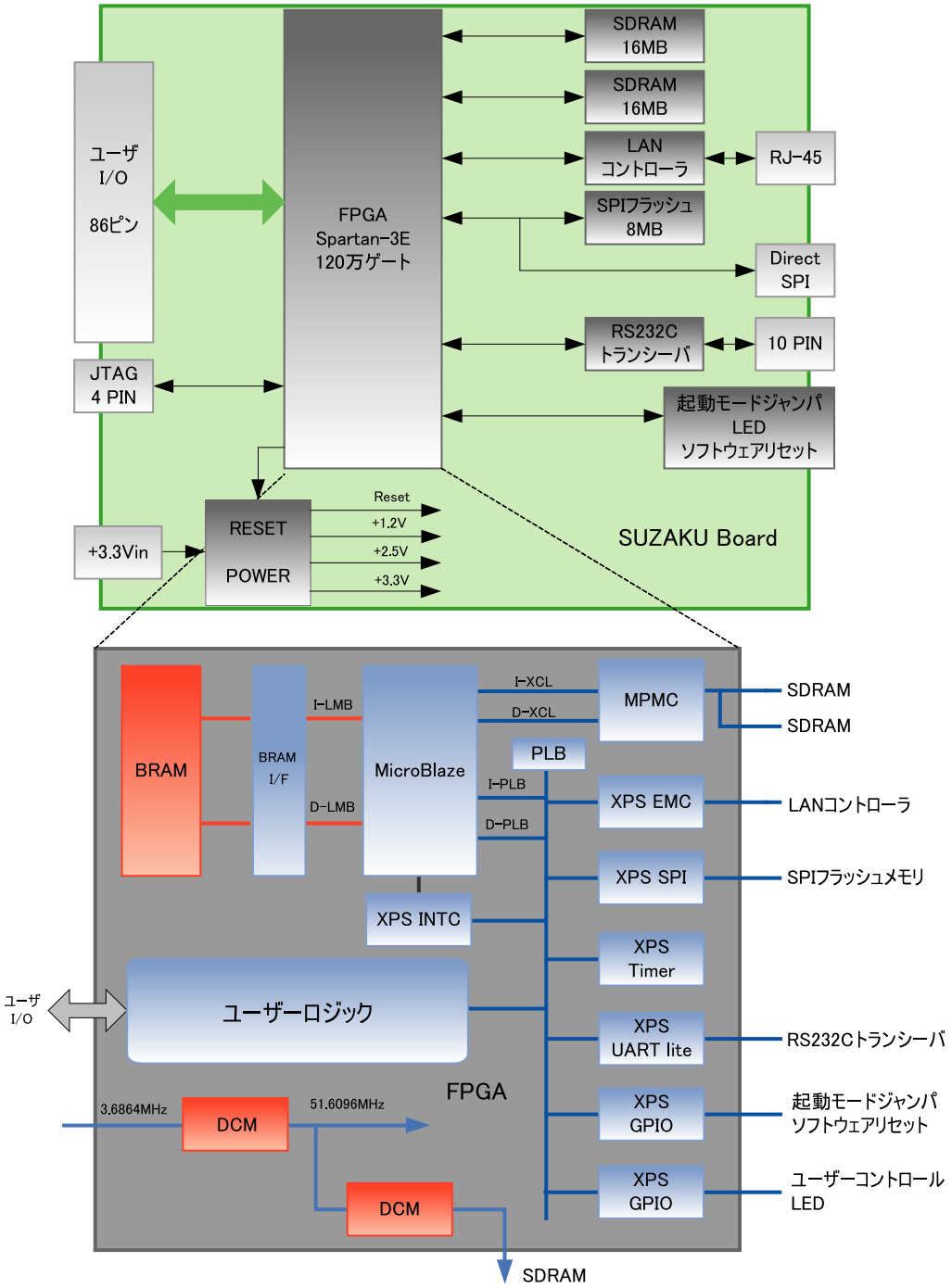


図 1.8 SZ130 の全体ブロック図

1.4.1.1. プロセッサ

FPGA 内部で MicroBlaze を使用しています。

1.4.1.2. バス

3 種類のバスで構成しています。

FPGA 内部 LMB(Local Memory Bus)	MicroBlaze と BRAM(FPGA 内部メモリ)を接続する専用バス
FPGA 内部 PLB(Processor Local Bus)	複数のペリフェラル IP コアを接続するバス
FPGA 外部バス	XPS EMC 及び MPMC を介し、外部メモリデバイスなどを接続するバス

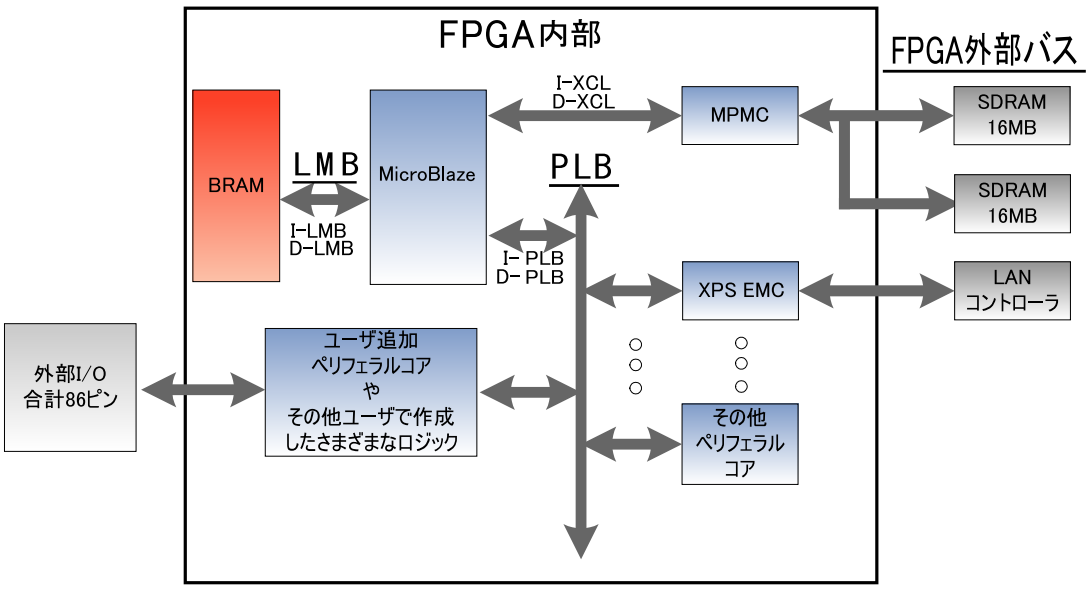


図 1.9 SZ130 のバス

1.4.1.3. メモリ

3 種類のメモリで構成しています。

FPGA 内部 BRAM(デフォルト 8KB)	ブートプログラム用として使用しています。起動完了後は、先頭の 32 バイト(割り込みベクタ領域)以外であれば、ユーザプログラムで使用することもできます。
FPGA 外部 SPI フラッシュメモリ	8MB を実装しています。ブートローダ Hermit-At や Linux、FPGA コンフィギュレーションデータなどの保存に使用しています。XPS SPI を使用し、PLB と接続しています。
FPGA 外部 SDRAM 16MB×2	Linux のメインメモリとして使用しています。MPMC と接続しています。2 枚の SDRAM の信号線は、完全に 2 つに分離して FPGA と接続しています。

1.4.1.4. シリアルコンソール

OS 用シリアルコンソールに XPS UART lite を使用しています。XPS UART lite は RS-232C トランシーバを介し、SUZAKU CON1 に接続しています。RS-232C トランシーバは、4 チャンネルタイプのものを実装しています。

1.4.1.5. LAN

LAN コントローラは LAN9115(メーカ：SMSC)を実装しています。LAN9115 は XPS EMC を使用し PLB と接続しています。

1.4.1.6. FPGA コンフィギュレーション

SPI コンフィギュレーションを採用しています。SPI フラッシュメモリは M25P64(メーカ：ST マイクロエレクトロニクス)を実装しています。

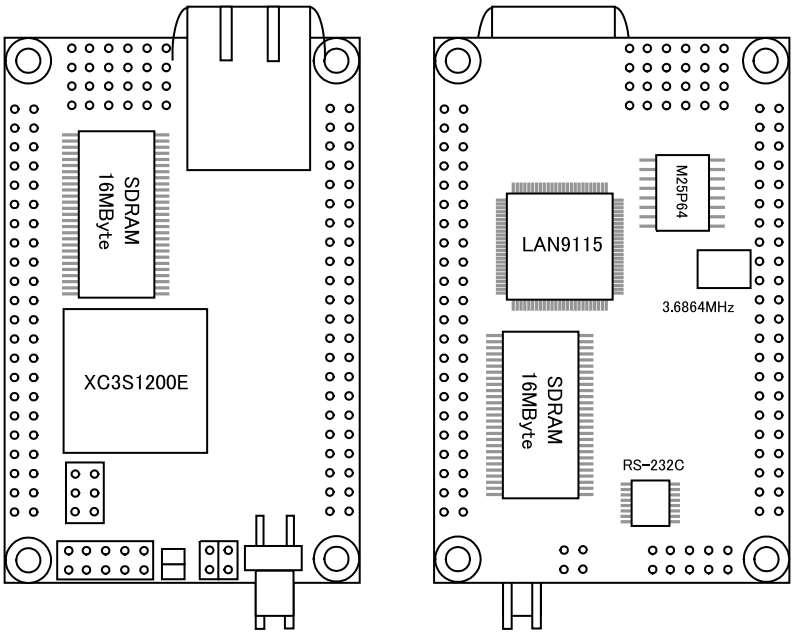


図 1.10 SZ130 の主要部品配置図

1.4.2. SZ410

SZ410 の全体のブロック図を以下に示します。

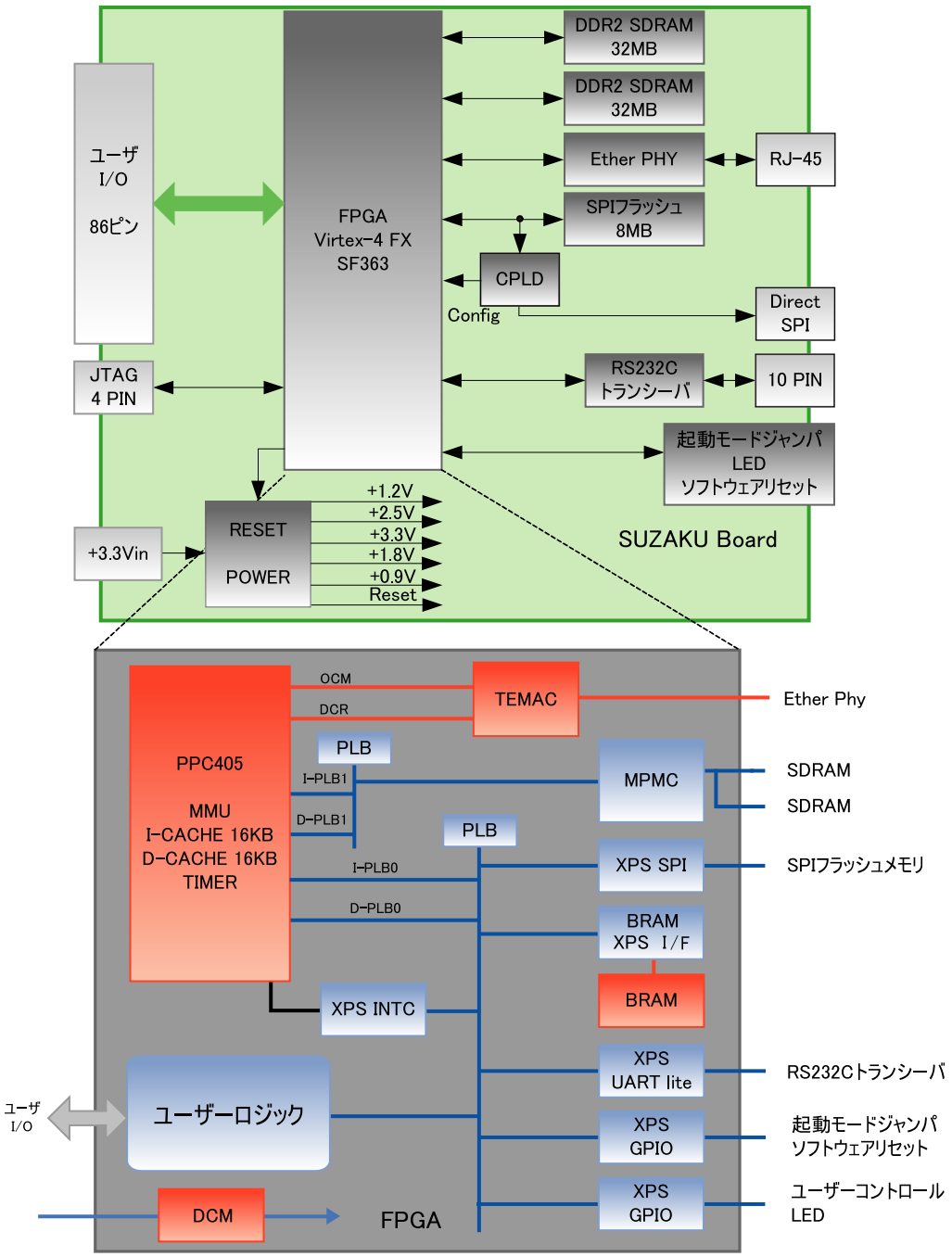


図 1.11 SZ410 の全体ブロック図

1.4.2.1. プロセッサ

FPGA 内部で PowerPC405 を使用しています。

1.4.2.2. バス

4 種類のバスで構成しています。

FPGA 内部 PLB(Processor Local Bus)	PowerPC405 と BRAM、MPMC などのペリフェラル IP コアを接続するバス
OCM(On-Chip Memory Bus)	PowerPC405 と TEMAC の FIFO を接続するバス
DCR(Device Controll Register Bus)	PowerPC405 と TEMAC を接続するバス
FPGA 外部バス	MPMC 等を介し、外部メモリデバイスなどを接続するバス

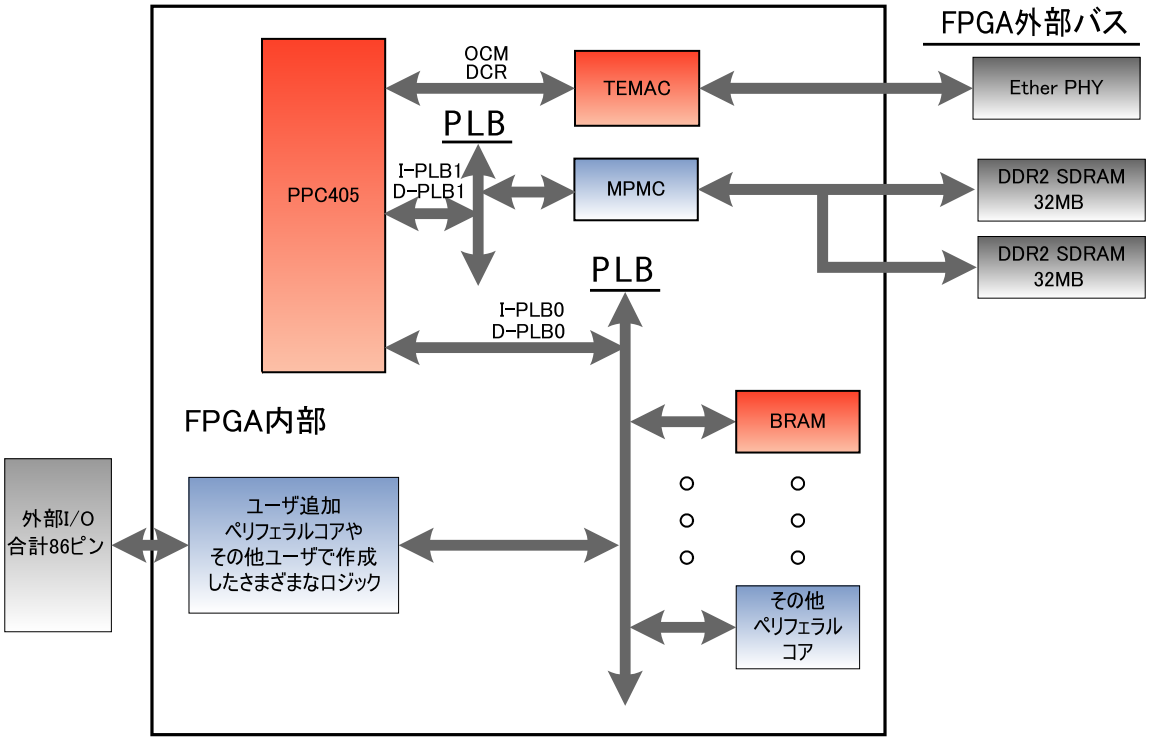


図 1.12 SZ410 のバス

1.4.2.3. メモリ

3 種類のメモリで構成しています。

FPGA 内部 BRAM(デフォルト 16KB)	ブートプログラム用として使用しています。ブート完了後は、ユーザプログラムで使用することもできます。
FPGA 外部 SPI フラッシュメモリ	8MB を実装しています。ブートローダ Hermit-At や Linux、FPGA コンフィグデータなどの保存に使用しています。XPS SPI SZ410 を使用し、PLB と接続しています。
FPGA 外部 DDR2 SDRAM 32MB×2	Linux のメインメモリとして使用しています。MPMC と接続しています。2 枚の DDR2 SDRAM の信号線は完全に 2 つに分離して、FPGA と接続しています。

1.4.2.4. シリアルコンソール

OS 用シリアルコンソールに XPS UART lite を使用しています。XPS UART lite は RS-232C トランシーバを介し、SUZAKU CON1 に接続しています。RS-232C トランシーバは、4 チャンネルタイプのものを使用しています。

1.4.2.5. LAN

Virtex-4 FX 内蔵の TEMAC(Tri Mode Ether MAC) と 10BASE-T/100BASE-TX の Ethrer PHY(メーカ：SMSC)を使用しています

1.4.2.6. FPGA コンフィギュレーション

CPLD を使用した SPI コンフィギュレーションを採用しています。SPI フラッシュメモリは M25P64(メーカ：ST マイクロエレクトロニクス)を実装しています。

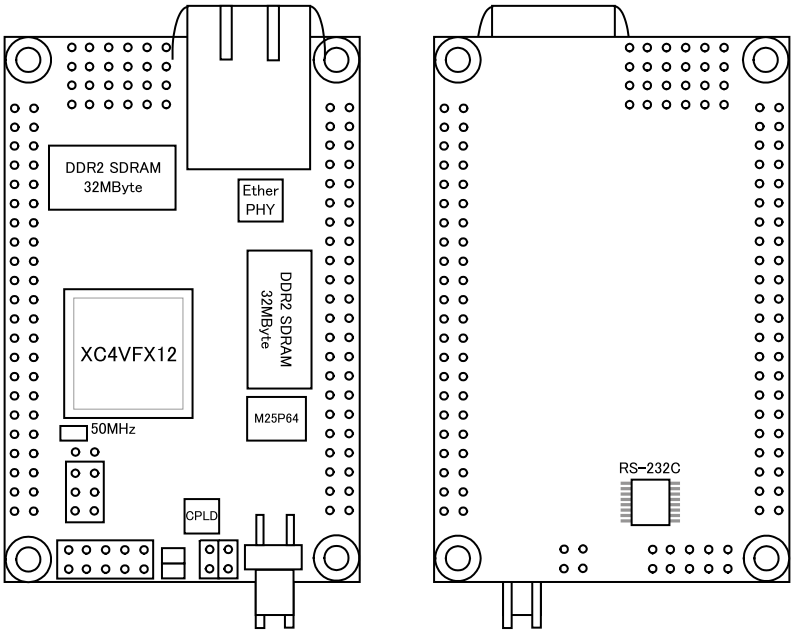


図 1.13 SZ410 の主要部品配置図

SZ410 の JTAG(CON7)コネクタの横のスルーホールは？

SZ410 の JTAG(CON7)コネクタの横のスルーホールは CPLD 書き込み用のスルーホールです。FPGA と CPLD の JTAG ピンは以下の図のように TCK、TMS を共用する形で配線されています。CPLD を書き換えることはもちろん出来ますが、書き換えに失敗してしまうと SUZAKU が動くようになるまでの復旧が大変なので、どうしてもということがないかぎりは、CPLD を書き換えないようにしてください。

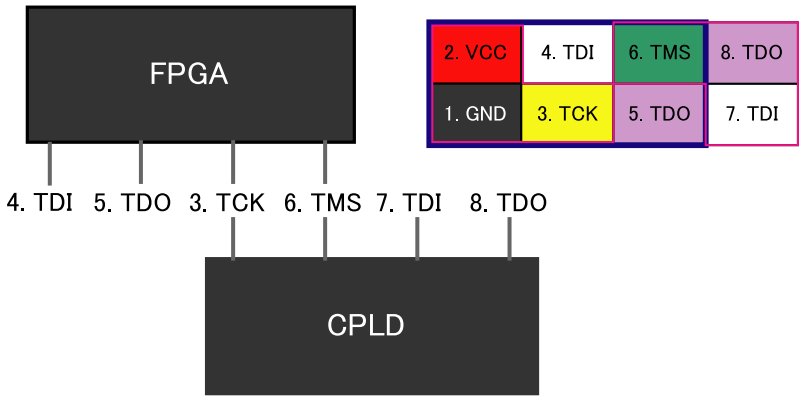


図 1.14 SZ410 の JTAG

SZ410 CPLD と SPI フラッシュメモリの採用理由

CPLD と SPI フラッシュメモリのセットを採用した理由は以下の 2 点です。

- シリアルビットストリームである SPI はそのままマスターシリアルに置き換えることが容易。
- 8mm×6mm の SPI フラッシュメモリと 5mm×5mm の CPLD(XC2C32QFP32 ピン)を使用すると、基板上の設置面積が小さくてすむ。

1.5. メモリマップ

1.5.1. SZ130

SZ130 のメモリマップは以下のとおりです。

表 1.2 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	BlockRAM 8KB
0x0000 2000	0x7FFF FFFF	Reserved	
0x8000 0000	0x8200 FFFF	MPMC	SDRAM 32MB
0x8201 0000	0xFEFF FFFF	Free	
0xFF00 0000	0xFF00 01FF	XPS SPI	SPI フラッシュメモリ 8MB
0xFF00 0200	0xFFDF FFFF	Free	
0xFFE0 0000	0xFFE0 FFFF	XPS EMC	LAN コントローラ
0xFFE1 0000	0xFFFF 0FFF	Free	
0xFFFF 1000	0xFFFF 10FF	XPS Timer	
0xFFFF 1100	0xFFFF 1FFF	Free	
0xFFFF 2000	0xFFFF 20FF	XPS UART lite	RS-232C
0xFFFF 2100	0xFFFF 2FFF	Free	
0xFFFF 3000	0xFFFF 30FF	XPS Interrupt Controller	
0xFFFF 3100	0xFFFF 9FFF	Free	
0xFFFF A000	0xFFFF A1FF	XPS 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	XPS GPIO	LED
0xFFFF A400	0xFFFF FFFF	Free	

1.5.2. SZ410

SZ410 のメモリマップは以下のとおりです。

表 1.3 SZ410 のメモリマップ

Start Address	End Address	ペリフェラル	デバイス
0x0000 0000	0x03FF FFFF	MPMC	DDR2 SDRAM 64MB
0x0400 0000	0xF0EF FFFF	Free	
0xF0FF 0000	0xF0FF 01FF	XPS SPI	SPI フラッシュメモリ 8MB
0xF0FF 0200	0xF0FF 1FFF	Free	
0xF0FF 2000	0xF0FF 20FF	XPS UART lite	RS-232C
0xF0FF 2100	0xF0FF 2FFF	Free	
0xF0FF 3000	0xF0FF 30FF	XPS Interrupt Controller	
0xF0FF 3100	0xF0FF 9FFF	Free	
0xF0FF A000	0xF0FF A1FF	XPS 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	XPS GPIO	LED
0xF0FF A400	0xFFFF BFFF	Free	
0xFFFF C000	0xFFFF FFFF	BRAM	BlockRAM 16KB

表 1.4 SZ410 の DCR メモリマップ

Start Address	End Address	ペリフェラル	デバイス
0x000	0x007	OCM TEMAC	Ethernet PHY

2.LED/SW ボードについて

SUZAKU スターターキットは"SUZAKU+LED/SW ボード"で構成されます。LED/SW ボードは SUZAKU の学習用ボードとして生み出されました。LED/SW ボードについて回路図をみながら簡単に説明します。詳細については後章で実際に LED/SW ボードに触りながら説明します。

2.1. 簡単な回路説明

以下の回路図が LED/SW ボードの回路図です。小さくて見えないという方は、付属 CD-ROM[1]に回路図及び部品表を収録しているのでご参照ください。

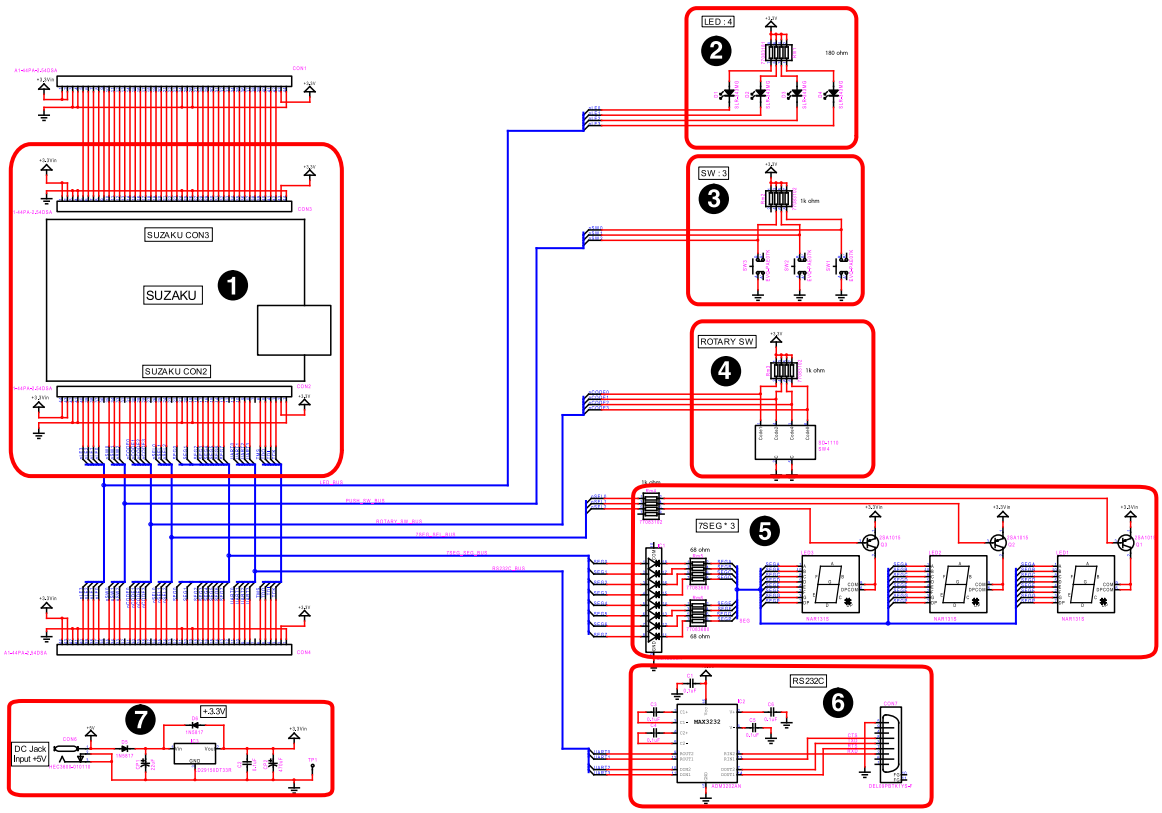


図 2.1 LED/SW 回路図(縮小版)

❶ SUZAKU です。CON2、CON3 から LED/SW ボードと接続します。

❷ 緑色の単色 LED が 4 つ(D1、D2、D3、D4)です。

❸ 押しボタンスイッチが 3 つ(SW1、SW2、SW3)です。

❹ ロータリコードスイッチ(SW4)です。

❺ 7 セグメント LED が 3 つ(LED1、LED2、LED3)です。

❻ シリアルポートです。

[1]"\suzaku-starter-kit\doc"以下に収録しています。

❼ 安定した+3.3V を得るために 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。

2.2. 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。

その他の SUZAKU + LED/SW ボードのピンアサインについては付録 A SUZAKU + LED/SW ボードのピンアサインをご参照ください。

表 2.1 クロック、リセット信号のピンアサイン

番号	信号名	I/O	機能	FPGA 接続先	
				SZ130	SZ410
	SYS_CLK	I	クロック信号	U10	Y6
	SYS_RST	I	リセット信号	D3	U3

表 2.2 機能用ピンアサイン(CON2)

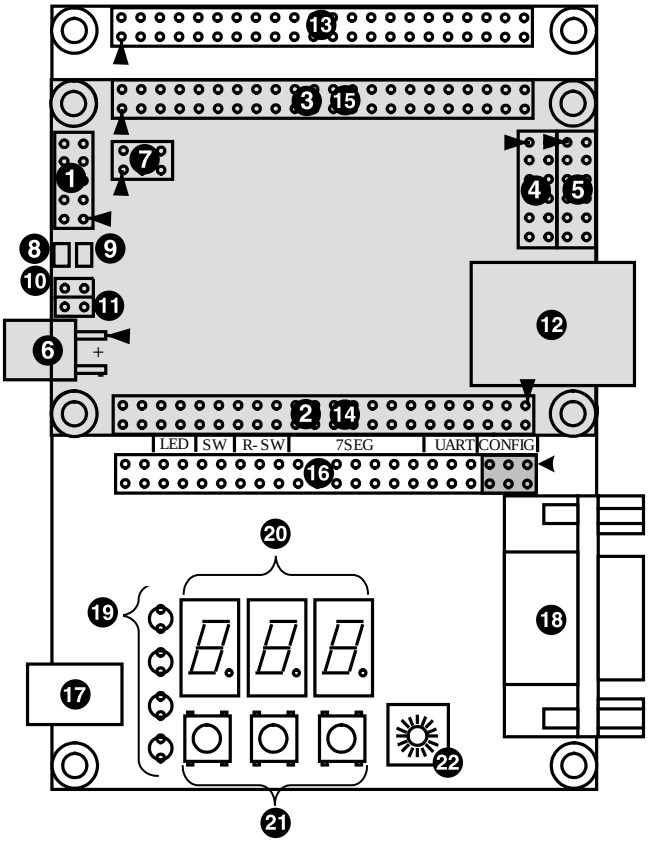
番号	信号名	I/O	機能	FPGA 接続先	
				SZ130	SZ410
8	UART3	I	RTS	N4	D15
9	UART2	O	TXD	M6	E15
10	UART1	O	CTS	M5	F15
11	UART0	I	RXD	M3	P4
13	SEG7	O	セグメント DP	L5	P1
14	SEG6	O	セグメント G	L6	P2
15	SEG5	O	セグメント F	L4	L2
16	SEG4	O	セグメント E	L3	M2
17	SEG3	O	セグメント D	L2	N2
18	SEG2	O	セグメント C	L1	N3
20	SEG1	O	セグメント B	C9	Y7
22	SEG0	O	セグメント A	D9	W7
24	nSEL2	O	7 セグメント LED3 選択	K6	N5
25	nSEL1	O	7 セグメント LED2 選択	K4	M3
26	nSEL0	O	7 セグメント LED1 選択	K3	M4
28	nCODE3	I	ロータリコードスイッチ 2^3	J1	H5
29	nCODE2	I	ロータリコードスイッチ 2^2	F9	E2
30	nCODE1	I	ロータリコードスイッチ 2^1	E9	D2
31	nCODE0	I	ロータリコードスイッチ 2^0	A10	U9
33	nSW2	I	押しボタンスイッチ SW3	D11	L1
34	nSW1	I	押しボタンスイッチ SW2	C11	M1
35	nSW0	I	押しボタンスイッチ SW1	F11	G4
37	nLE0	O	単色 LED(緑)D1	E12	G2
38	nLE1	O	単色 LED(緑)D2	F12	F2
39	nLE2	O	単色 LED(緑)D3	B11	F1
40	nLE3	O	単色 LED(緑)D4	A11	L13

3.SUZAKU+LED/SW ボードの構成

SUZAKU と LED/SW ボードをスタックした状態でのコネクタの配置やジャンパの設定等について説明します。誤挿入や誤配線等間違った使い方をすると、SUZAKU や LED/SW ボードが壊れてしまう可能性があります。しっかりとご確認ください。

3.1. 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 トランシー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。



▲のマークは基板上の 1 番ピンの位置を示します。

図 3.1 各種インターフェースの配置

表 3.1 SUZAKU のコネクタ配置

	図番	部品番号	説明
SUZAKU	❶	CON1	RS-232C コネクタ
	❷	CON2	外部 I/O、フラッシュメモリ用コネクタ (LED/SW CON2 と接続)
	❸	CON3	外部 I/O コネクタ(LED/SW CON3 と接続)
	❹	CON4	外部 I/O コネクタ
	❺	CON5	外部 I/O コネクタ
	❻	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)
	❼	CON7	FPGA JTAG 用コネクタ
	❽	D3	パワー ON LED(緑)
	❾	D1	ユーザーコントロール LED(赤)
	❿	JP1	起動モードジャンパ
	⓫	JP2	FPGA プログラム用ジャンパ
	⓬	L2	Ethernet 10BASE-T/100BASE-TX コネクタ

表 3.2 LED/SW のコネクタ配置

	図番	部品番号	説明
LED/SW	⓭	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)
	⓮	CON2	SUZAKU 接続コネクタ(SUZAKU CON2 と接続)
	⓯	CON3	SUZAKU 接続コネクタ(SUZAKU CON3 と接続)
	⓰	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)
	⓱	CON6	+5V 入力コネクタ
	⓲	CON7	RS-232C コネクタ
	⓳	D1 ～ 4	単色 LED(緑)Low レベルで点灯
	⓴	LED1 ～ 3	7 セグメント LEDHigh 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.電源を入れる前に

SUZAKU スターターキットに電源を入れる前に、開発をするために必要なものやインストールする必要のあるソフトウェア、開発環境についての説明をします。

4.1. SUZAKU スターターキットの準備

SUZAKU スターターキットは以下のものから構成されます。SUZAKU 単体から作業をスタートする場合は以下を参考に準備してください。

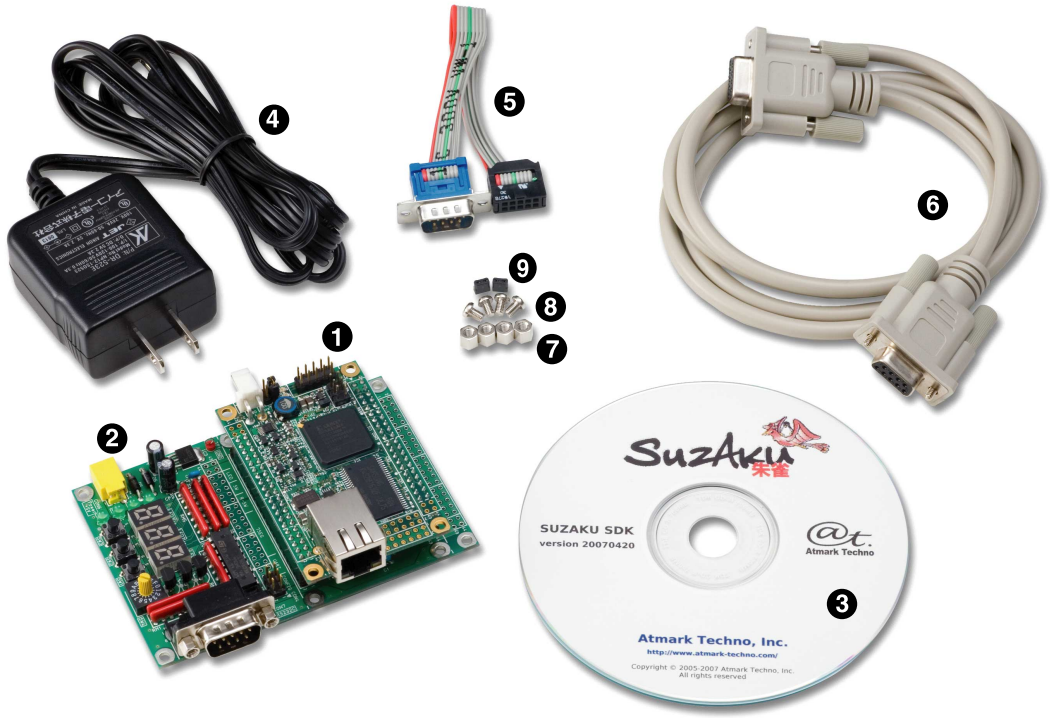


図 4.1 SUZAKU スターターキット(SZ130-SIL)

❶ SUZAKU
❷ LED/SW ボード[1]
❸ 付属 CD-ROM
❹ AC アダプタ 5V
❺ D-Sub9 ピン-10 ピン変換ケーブル
❻ D-Sub9 ピンクロスケーブル
❼ スペーサ×4

[1]SZ410 をお使いの場合は、SZ410 対応品の LED/SW ボード(SIL00-U01)をお使いください。

❽ ネジ×4

❾ ジャンパプラグ×2

4.2. その他に必要なものと開発環境について

SUZAKU スターターキットの他に以下のものが必要となります。

作業用 PC	Windows XP Pro(32bit) もしくは Windows Vista Business(32bit)が動作し、シリアルポート(1 ポート)、LAN ポート(1 ポート)、USB ポート(1 ポート)もしくはパラレルポート(1 ポート)[2]を持つ PC を準備してください。
LAN ケーブル	SUZAKU と LAN を経由した通信を行う場合に必要になります。作業用 PC と SUZAKU はスイッチングハブを介して接続、または直接接続してください。[3]
ダウンロードケーブル	Xilinx Platform Cable USB(I、II)、Xilinx Parallel Cable(III、IV)もしくはそれ相当のものを準備してください。[4]
シリアル通信ソフトウェア	シリアル通信ソフトウェアをインストールしてください。本書では Tera Term(Pro)を使用しています。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。
Xilinx ISE Design Suite11.5	Xilinx ISE Design Suite11.5(Embeded Edition もしくは System Edition)をインストールしてください。ISE WebPack Tool と Embeded Development Kit(EDK)の組み合わせでも構いません。Xilinx ISE Design Suite11.5 の評価版は Xilinx のホームページからダウンロードできます(2010 年 8 月現在)。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。[4]
ダウンローダ Hermit-At	ダウンローダ Hermit-At をインストールしてください。付属 CD-ROM[5]に収録しています。また、SUZAKU 公式サイトのダウンロードページから最新版をダウンロードすることが出来ます。
SPI Writer	ダウンロードケーブルが Xilinx Parallel Cable(III、IV)で、「6.2.3. SPI Writer で書き換える」の方法で書き込みをする場合は SPI Writer をインストールしてください。必ずしも必要ではありません。付属 CD-ROM[6]に収録しています。
バイナリエディタ	「6.2.4. ダウンローダ Hermit-At で書き換える」の方法で書き込みを行う場合はバイナリエディタをインストールしてください。

4.3. 付属 CD-ROM について

付属 CD-ROM には、開発に必要となる FPGA プロジェクト、ソフトウェア、開発環境、イメージファイル、各種マニュアルを収録しております。これら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。

[2]必要となるポートは使用するダウンロードケーブルによります。詳しくは「6. SUZAKU を書き換える」をご参照ください。
[3]SZ410 は半二重通信に非対応ですのでご注意ください。
[4]Xilinx 製品の詳細については、Xilinx のホームページをご覧になられるか、Xilinx 代理店にお問い合わせください。
[5]`"\suzaku\bootloader\hermit-at-win-x.x.x.zip"`に収録しています。
[6]`"\suzaku\tools\spi_writer-yyyymmdd.zip"`に収録しています。

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

4.4. SUZAKU スターターキットの組み立て

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(SUZAKU と LED/SW ボードの CON2 同士、CON3 同士を接続します)。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。SUZAKU スターターキットでは CON2 の 19 番ピンに逆挿し防止対策が施されています。

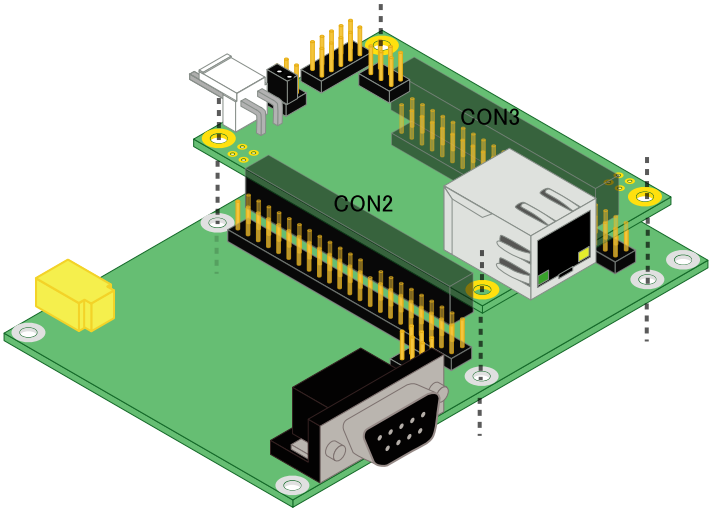


図 4.2 SUZAKU と LED/SW ボード接続

もし CON2、CON3 にコネクタが実装されていない場合、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。コネクタは 40 ～ 44 ピンのものをご用意ください。CON2 の 41 ～ 44 ピン、CON3 の 41 ～ 44 ピンにはコネクタを接続しなくても動作いたしますので、コネクタが 44 ピンに足りない場合は、1 ピン側によせて半田付けしてください。

半田付けする際はマスキングをし、周囲の部品に半田くず、半田ボール等付着しない様十分にご注意ください。

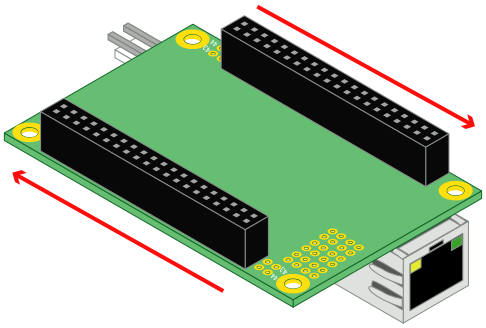


図 4.3 コネクタの半田付け

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。

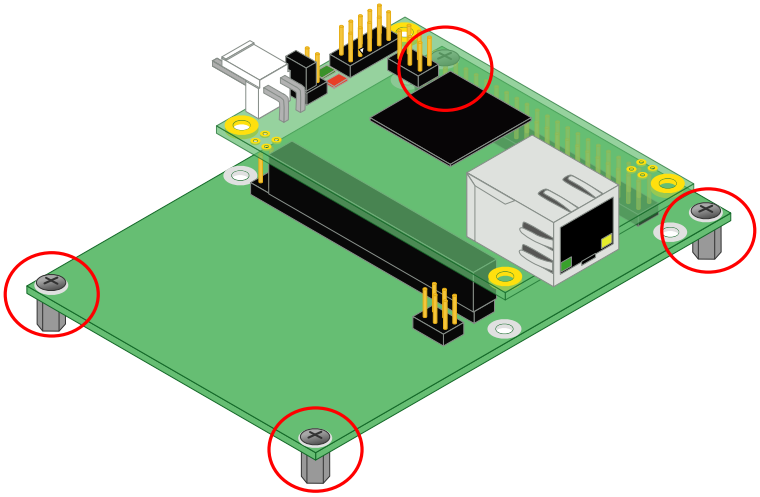


図 4.4 スペーサ取り付け

SZ410 をお使いの場合、SZ410 対応品シールが貼ってある LED/SW ボードをお使いください。 SZ410 は他の SUZAKU に比べ消費電流が多いため SZ410 対応品でない LED/SW ボードでは正常動作しないことがあります。

5.SUZAKU+LED/SW ボードを動かす

色々説明してきましたが、やっと SUZAKU スターターキット(SUZAKU + LED/SW ボード)を動かします。出荷状態の SUZAKU スターターキットですと、フラッシュメモリに 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 がどのような動きをするのか実際に体験してください。

SUZAKU はジャンパによりブートローダモード、オートブートモード、FPGA コンフィギュレーション待ちの 3 つの状態に設定できます。JP1 は起動モードジャンパ、JP2 は FPGA プログラム用ジャンパです。

ブートローダモードとオートブートモードで SUZAKU スターターキットを動かします。

表 5.1 ジャンパの設定と起動時の動作

JP1	JP2	起動時の動作	起動モード
ショート	オープン	ファーストブートローダ BBoot を起動	ブートローダモード
オープン	オープン	Linux カーネルを起動	オートブートモード
-	ショート	何も起動しません	FPGA コンフィギュレーション待ち

フラッシュメモリの中身が SUZAKU スターターキット出荷状態以外の方もおられると思います。この章の内容を行なうには、SUZAKU を書き換える必要があります。書き換え用のファイルは付属 CD-ROM[1]に収録しています。また、SUZAKU 公式サイト [http://suzaku.atmark-techno.com/series/stk/download]よりダウンロードすることも出来ます。書き換える方法については、「6. SUZAKU を書き換える」をご参照ください。

5.1. シリアル通信ソフトウェアの起動

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 5.2 シリアル通信ソフトウェアの設定

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

[1]Linux のイメージファイルは"\suzaku-starter-kit\image\image-sz***-sil.bin"、FPGA ファイルは"\suzaku-starter-kit\fpga\x.x\sz***\sz***-add_slot-yyyymmdd.zip"の default_bit_file フォルダの中に収録しています。

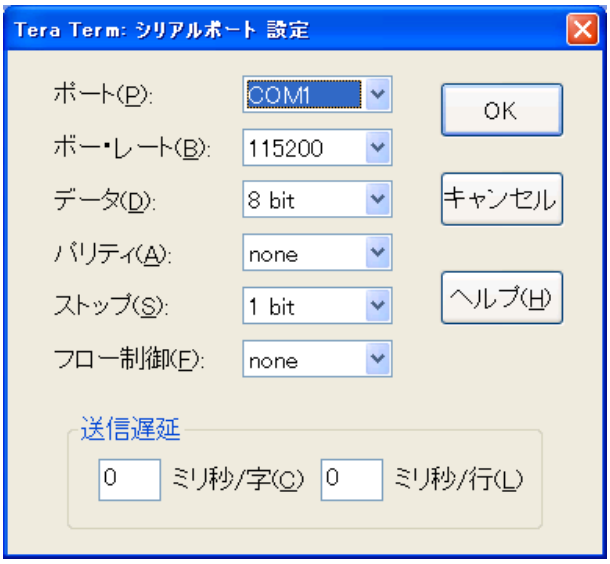


図 5.1 シリアル通信ソフトウェア(Tera Term)の設定

5.2. SUZAKU+LED/SW ボード起動準備

D-Sub9 ピン-10 ピン変換ケーブルを SUZAKU CON1 に、LAN ケーブルを SUZAKU L2 に接続してください。

SUZAKU CON1 に D-Sub9 ピン-10 ピン変換ケーブルを接続する際には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。

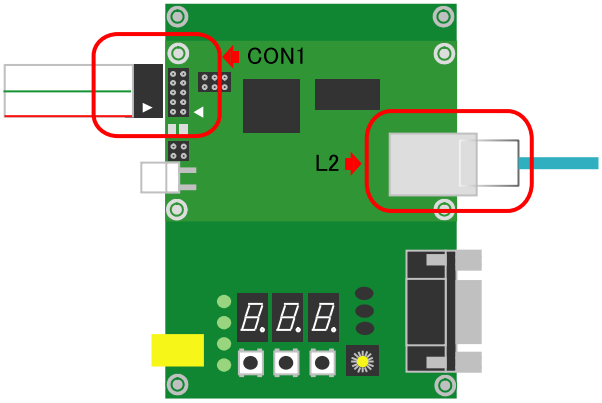


図 5.2 SUZAKU+LED/SW ボード配線

5.3. ブートローダモードでスロットマシンを動かす

まず、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。

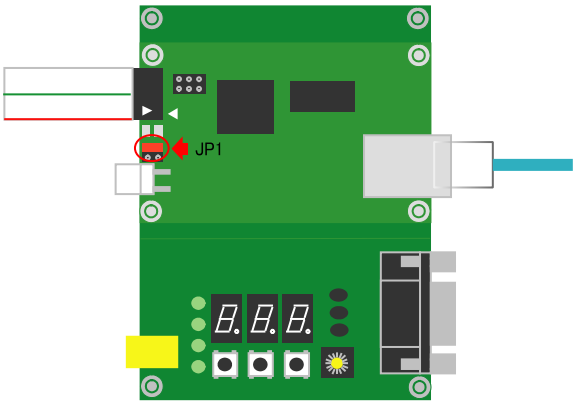


図 5.3 ブートローダモード ジャンパの設定

LED/SW CON6 に AC アダプタ 5V を接続してください。

ジャンパプラグをなくさないようにしよう

ジャンパプラグは小さいのでなくしてしまいがちです。たくさんジャンパプラグを持っている人はいいですが、持っていない人は、なくさないようにご注意ください。ジャンパプラグを使わない時には、ピンの片側だけにジャンパプラグを引っ掛けておくのがお勧めです。

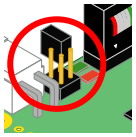


図 5.4 ジャンパプラグの扱い

電源系統について

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。

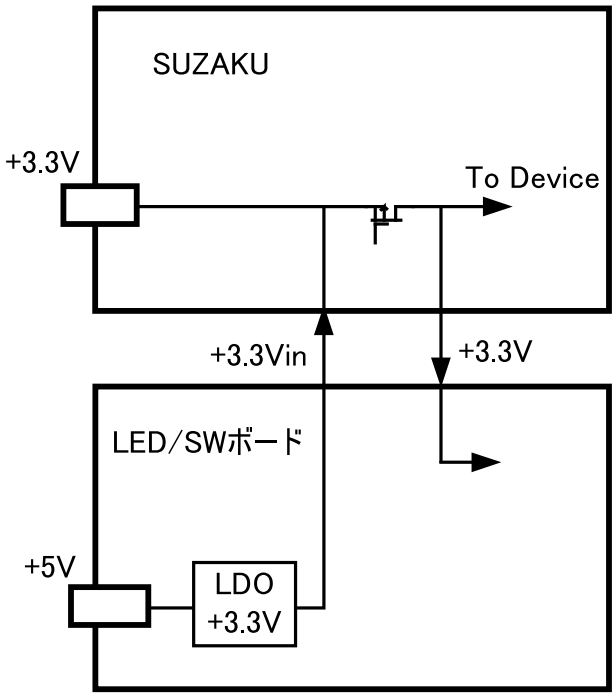


図 5.5 電源系統

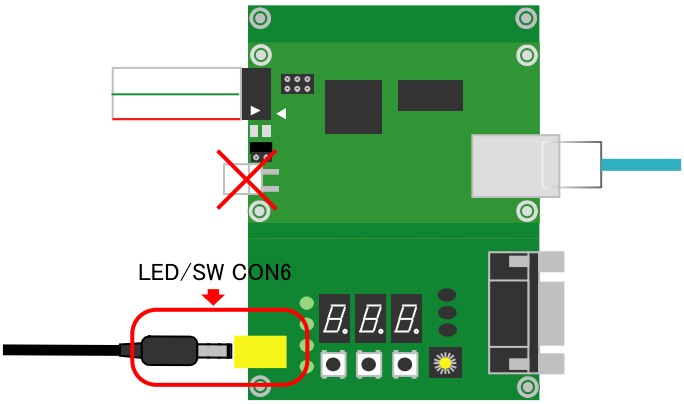


図 5.6 電源ケーブル接続の諸注意

5.3.1. スロットマシン起動

電源が供給されるとシリアル通信ソフトウェアの画面に以下のメッセージが表示され、スロットマシンを動かすことができるようになります。

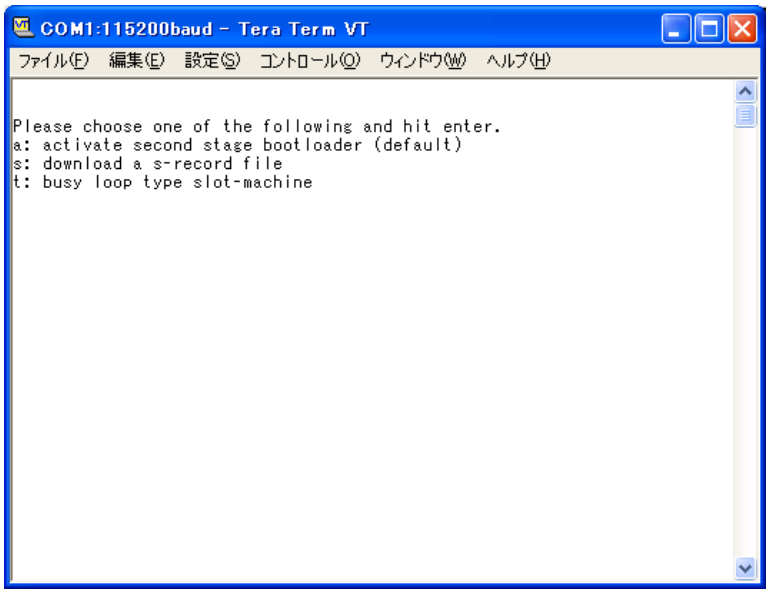


図 5.7 スロットマシンの起動

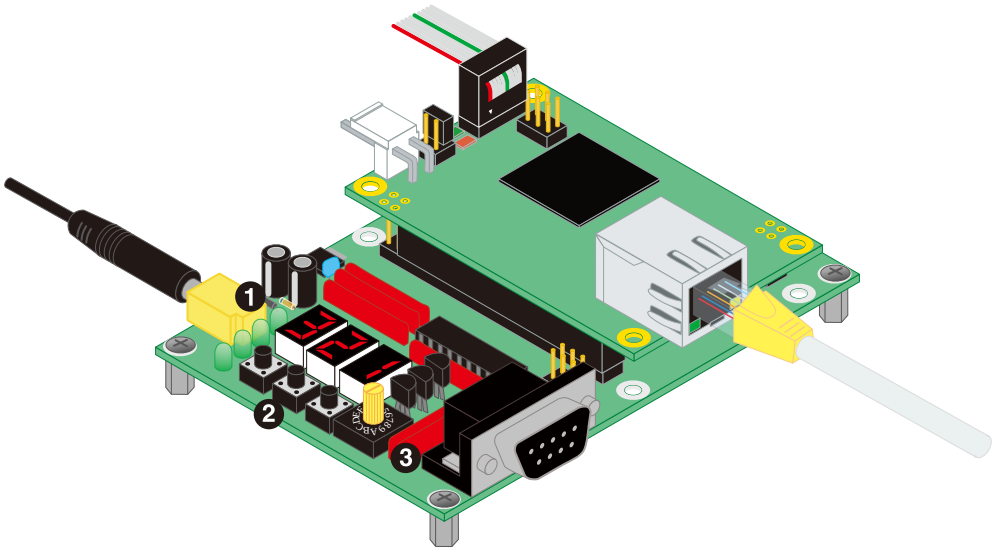


図 5.8 スロットマシンを動かしてみよう

❶ 数字がそろうと単色 LED が順次点灯します。

❷ 押しボタンスイッチを 2 つ以上一緒に押すとスロットの回転が始まり、1 つ押すとそれぞれに対応する 7 セグメント LED の回転が止まります。

❸ ロータリコードスイッチを 0→1→2 とまわすと数字の回転が速くなります。

5.3.2. 終了

SUZAKU スターターキットには電源ボタンがありません。終了するには、電源を切断する必要があります。AC アダプタをコンセントから抜いて終了してください。

5.4. オートブートモードで Linux を動かす

次にオートブートモードで Linux を動かします。JP1、JP2 をオープンにしてください。

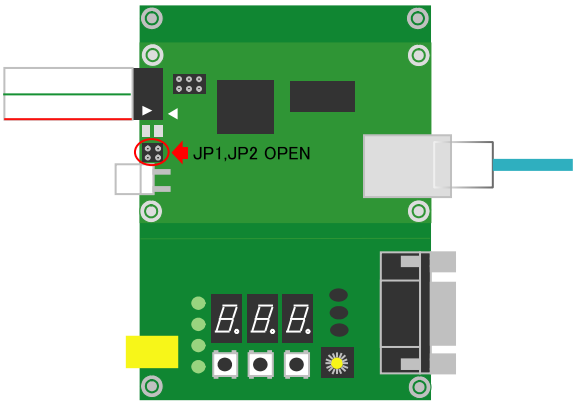


図 5.9 オートブートモード ジャンパの設定

5.4.1. Linux の起動

LED/SW CON6 に AC アダプタ 5V を接続し、電源を供給してください。シリアル通信ソフトウェアの画面に Linux の起動ログが表示されます。

例 5.1 SUZAKU の起動ログ(SZ130 の場合)

```
Linux version 2.4.32-uc0 (build@sv-build) (gcc version 3.4.1 ( Xilinx EDK 9.1
Build EDK_J.19 121007 )) #1 2008 年 3 月 26 日 水曜日 19:40:27 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... 25.65 BogoMIPS
Memory: 32MB = 32MB total
Memory: 29448KB available (957K code, 2001K 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
ttyS1 at 0xffffa600 (irq = 3) is a Microblaze UARTlite
Starting kswapd
xgpio #0 at 0xFFFFA000 mapped to 0xFFFFA000
Xilinx GPIO registered
sil7segc (1.0.1): 7seg-LED Driver of SUZAKU I/O Board -LED/SW- for CGI demo.
RAMDISK driver initialized: 16 RAM disks of 4096K size 1024 blocksize
eth0: LAN9115 (rev 1150001) at ffe00000 IRQ 2
Suzaku MTD mappings:
  Flash 0x800000 at 0xff000000
flash: Found an alies 0x800000 for the chip at 0x0, ST M25P64 device detect.
Creating 7 MTD partitions on "flash":
```

```
0x00000000-0x00800000 : "Flash/All"
0x00000000-0x00100000 : "Flash/FPGA"
0x00100000-0x00120000 : "Flash/Bootloader"
0x007f0000-0x00800000 : "Flash/Config"
0x00120000-0x007f0000 : "Flash/Image"
0x00120000-0x00420000 : "Flash/Kernel"
0x00420000-0x007f0000 : "Flash/User"
FLASH partition type: spi
uclinux[mtd]: RAM probe address=0x80125a5c size=0x1bf000
uclinux[mtd]: root filesystem index=7
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.
Mounting /etc/config:
Populating /etc/config:
flatfsd: Created 4 configuration files (149 bytes)
Setting hostname:
Setting up interface lo:
Starting DHCP client:
Starting inetd:
Starting thttpd:

SUZAKU-S.SZ130-SIL login:
```

5.4.2. ログイン

起動が終了すると、ログインプロンプトが表示されます。root ユーザでログインします。

表 5.3 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	権限
root	root	特権ユーザ

5.4.3. 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。例にある、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。

例 5.2 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 5.3 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
```

5.4.4. ウェブにアクセス

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。先ほど確認した 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 5.10 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。

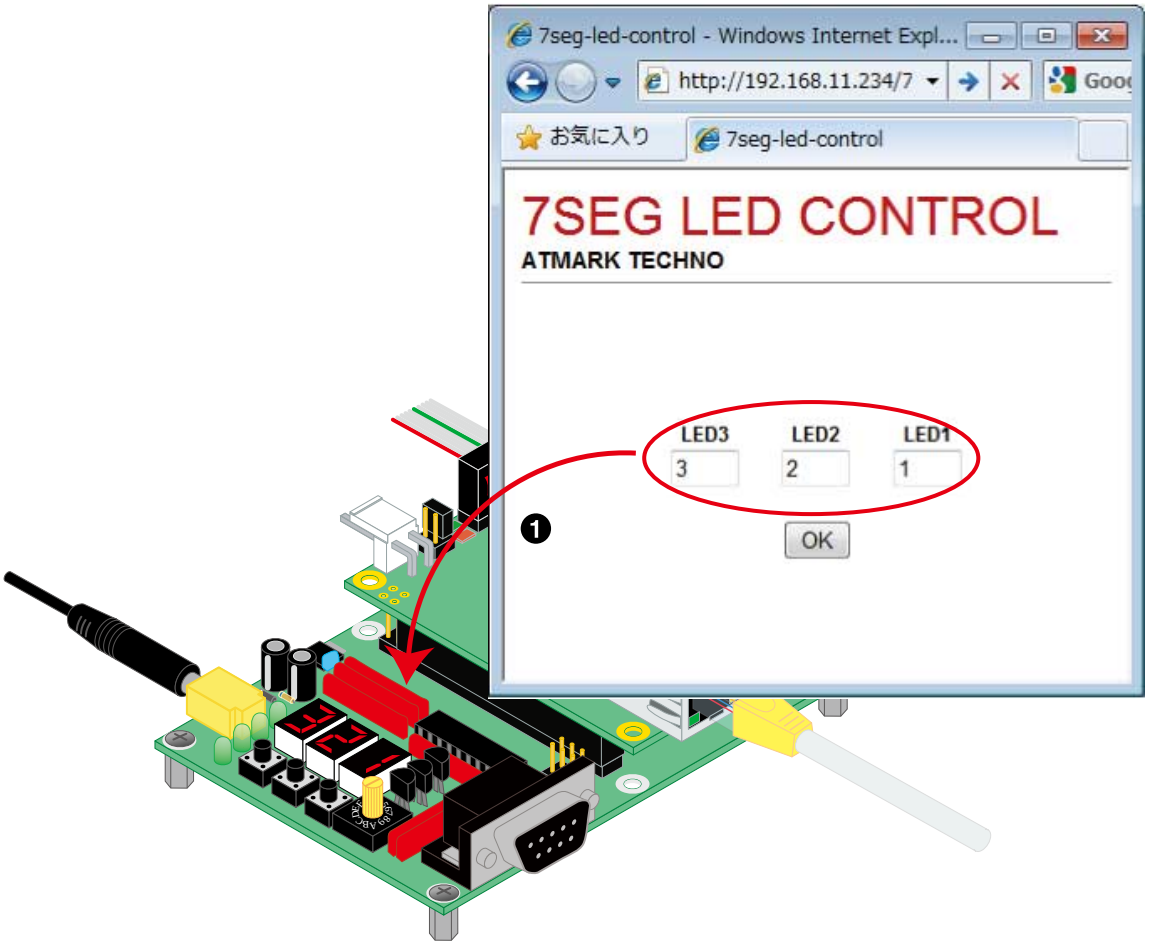


図 5.11 CGI を動かしてみる

❶ 1 ～ F(16 進数)の数字を設定して[OK]をクリックすると、7 セグメント LED に設定した数字が表示されます。

5.4.5. 終了

AC アダプタをコンセントから抜いて終了してください。

5.5. SUZAKU のブートシーケンス

SUZAKU スターターキットを動かしてみましたがいかがだったでしょうか。SUZAKU のブートシーケンスについて説明をします。

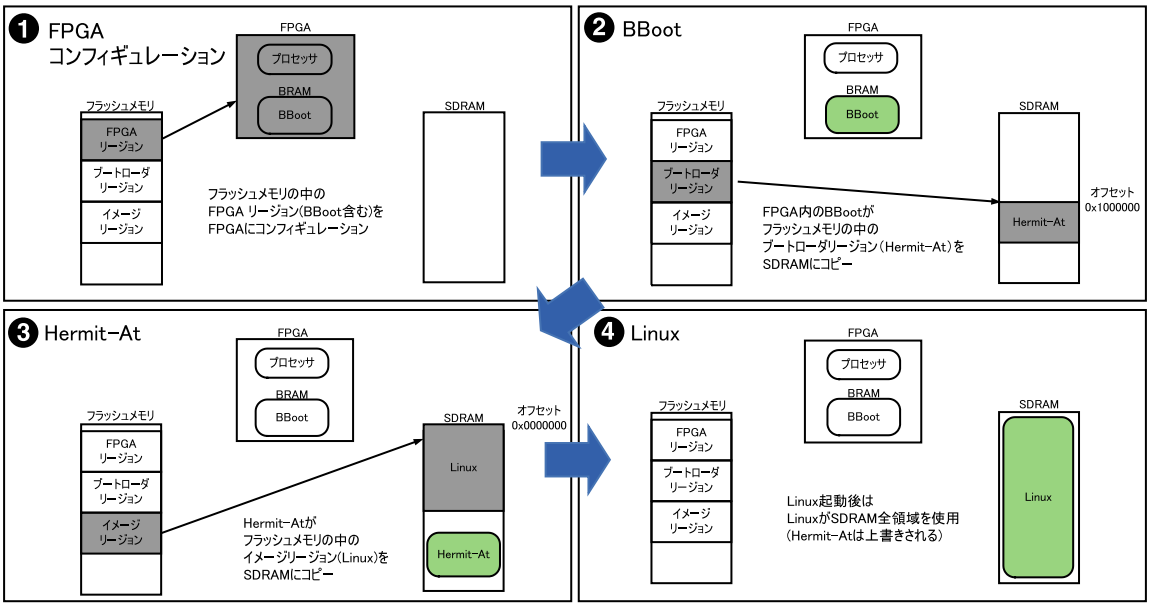


図 5.12 2 段階ブート

❶ SUZAKU に電源を投入すると、まず FPGA にフラッシュメモリの中の FPGA リージョン(BRAM 内ファーストブートローダ BBoot を含む)をコンフィギュレーションします。(フラッシュメモリのリージョンについては「6. SUZAKU を書き換える」で説明をします。)

❷ プログラムをスタートさせるリセットベクタのアドレス(MicroBlaze: 0x00000000 番地、PowerPC405: 0xFFFFFFFC 番地)に SUZAKU では FPGA の BRAM を割り当てているので、コンフィギュレーション終了後、FPGA の BRAM 内の BBoot が動作します。

❸ BBoot はフラッシュメモリの中のブートローダリージョン(Hermit-At)を SDRAM にコピーします。コピー終了後セカンドブートローダ Hermit-At の先頭アドレスにジャンプするようになっており、Hermit-At が起動します。

❹ Hermit-At はフラッシュメモリの中のイメージリージョン(Linux)を SDRAM にコピーします。コピー終了後 Linux の先頭アドレスにジャンプするようになっており、Linux が起動します。Linux が起動後は Linux が SDRAM の全領域を使用し、Hermit-At は必要ないので上書きしてしまいます。

また、SUZAKU スターターキットの BBoot は以下のようなフローで動作します。先ほどの動きを思い出して確認してみてください。

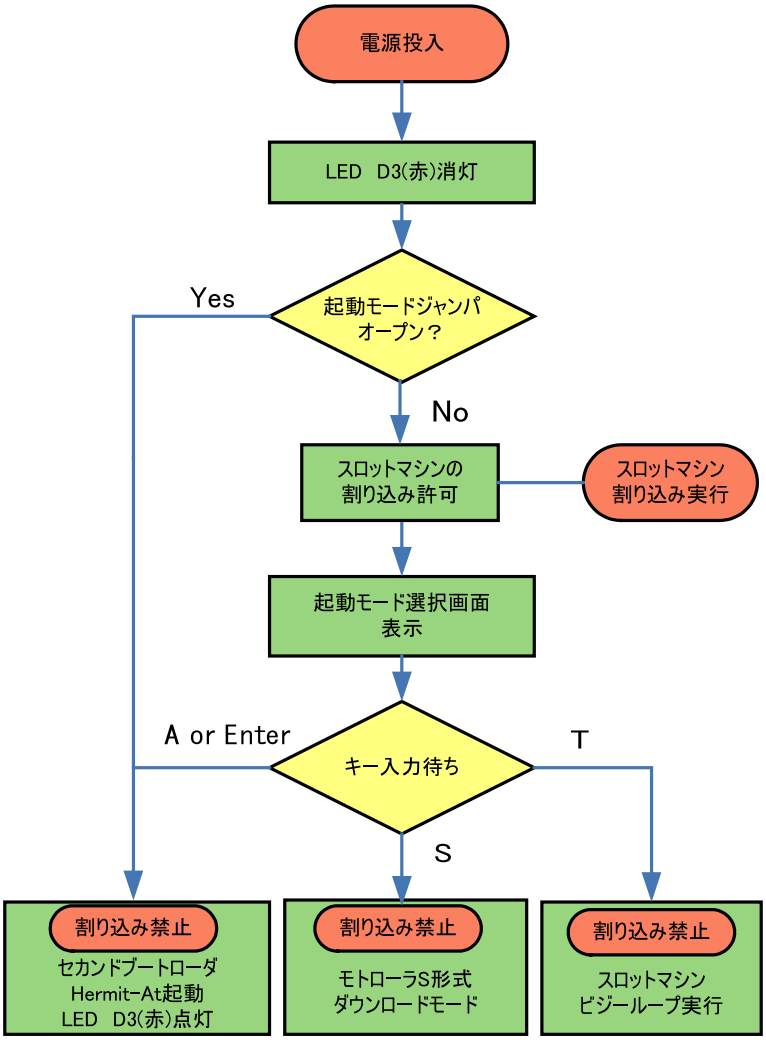


図 5.13 スターターキットの BBoot のフロー

2 段階ブート

SUZAKU のブートローダには以下の 2 つがあり、2 段階でブートします。

- ファーストブートローダ BBoot(FPGA の BRAM 内)
- セカンドブートローダ Hermit-At

2 段階のブートを使用しないことも可能ですが、SUZAKU ではあえて 2 段階でブートをしています。これには何らかの原因でフラッシュメモリの内容が書き換えられても、最低限のブートローダまでは動作させておきたかったという事と、ブートローダにより使用される BRAM のメモリ容量を最低限に抑えたかったという事の 2 つの理由があります。何らかの原因でフラッシュメモリの内容が書き換えられても、FPGA であれば JTAG からコンフィギュレーションデータをプログラムできるので、BBoot を復旧させることができ、フラッシュメモリの書き直しを行うことが出来ます。また、ブートローダは圧縮されたプログラムイメージを展開させたりするので、多くのメモリが必要となります。FPGA の BRAM は貴重なリソー

スなので、起動時にしか実行されないブートローダのために多くの容量を消費することを避け、セカンドブートローダ Hermit-At にこれらの機能を持たせることにしました。

6.SUZAKU を書き換える

SUZAKU で開発するためには SUZAKU を書き換える作業が必須となります。ここでは SUZAKU の書き換えかたを説明します。特に今すぐ SUZAKU を書き換える必要はないので、前から順に読んでいる方は本章をとばして次に進んで下さい。

「SUZAKU を書き換える=フラッシュメモリを書き換える」ことになります。フラッシュメモリには SUZAKU の基礎となる様々なデータが書き込まれています。

フラッシュメモリは大きく、FPGA リージョン、ブートローダリージョン、イメージリージョン、コンフィグリージョンという領域に分割され、データが書き込まれています。

- FPGA リージョン ： FPGA コンフィギュレーションデータが書き込まれています。
- ブートローダリージョン ： ブートローダ Hermit-At のデータが書き込まれています。
- イメージリージョン ： Linux のカーネルやユーザランドイメージが書き込まれています。
- コンフィグリージョン ： ネットワークの設定やパスワードが書き込まれています。

これらの領域はそれぞれ個別に書き換えることが出来ます。

FPGA のデータは FPGA メーカから提供されているシリアル ROM に記憶して FPGA に書き込むことが多いのですが、SUZAKU では Spartan-3E の機能や CPLD を使用し、市販のフラッシュメモリの FPGA リージョンに記憶して FPGA に書き込んでいます。

これとは別に、一時的に JTAG から FPGA にデータを書き込む方法もあります。電源を落とすと書き込んだ内容は消えてしまうのですが、フラッシュメモリを書き換えるよりも速いので、デバッグ時には大変有効です。

6.1. フラッシュメモリマップ

フラッシュメモリのリージョンの区分は、製品毎に異なります。

6.1.1. SZ130

SZ130 のフラッシュメモリマップは以下のとおりです。

表 6.1 フラッシュメモリマップ（SZ130 : 8MB）

アドレス	リージョン	サイズ	説明
0x00000000 | 0x007FFFFF	ALL	8MB	すべての領域を含みます。
0x00000000 | 0x000FFFFF	FPGA	1MB	FPGA コンフィギュレーションデータを格納する領域です。
0x00100000 | 0x0011FFFF	ブートローダ	128KB	ブートローダ Hermit-At を格納する領域です。
0x00120000 | 0x007EFFFF	イメージ	約 6.81MB	Linux カーネル・ユーザーランドイメージを格納する領域です。
0x007F0000 | 0x007FFFFF	コンフィグ	64KB	flatfsd が使用するコンフィグ領域です。

6.1.2. SZ410

SZ410 のフラッシュメモリマップは以下のとおりです。

表 6.2 フラッシュメモリマップ（SZ410 : 8MB）

アドレス	リージョン	サイズ	説明
0x00000000 | 0x007FFFFF	ALL	8MB	すべての領域を含みます。
0x00000000 | 0x000FFFFF	FPGA	1MB	FPGA コンフィギュレーションデータを格納する領域です。
0x00100000 | 0x0011FFFF	ブートローダ	128KB	ブートローダ Hermit を格納する領域です。

アドレス	リージョン	サイズ	説明
0x00120000 | 0x007EFFFF	イメージ	約 6.81MB	Linux カーネル・ユーザーランドイメージを格納する領域です。
0x007F0000 | 0x007FFFFF	コンフィグ	64KB	flatfsd が使用するコンフィグ領域です。

以下に SUZAKU のそれぞれのリージョンの書き換えかたと使用ファイル、書き換えるときの前提条件を示します。フラッシュメモリを書き換える方法は色々ありますが、FPGA リージョン、ブートローダリージョンを書き換える場合はダウンローダ Hermit-At、イメージリージョンを書き換える時は NetFlash を使用するのが速いです。

表 6.3 SUZAKU の書き換えかた[1][2]

		書き込み方法	使用ファイル	前提条件
FPGA		iMPACT	bit ファイル	なし
フラッシュメモリ	ALL リージョン	iMPACT(DirectSPI)	mcs ファイル	なし
	FPGA リージョン	SPI Writer	bit ファイル	なし
		ダウンローダ Hermit-At	bin ファイル	ブートローダ Hermit-At が起動する
	ブートローダリージョン	ダウンローダ Hermit-At	bin ファイル	ブートローダ Hermit-At が起動する
		BBoot モトローラ S 形式	srec ファイル	BBoot が起動する
	イメージリージョン	ダウンローダ Hermit-At	bin ファイル	ブートローダ Hermit-At が起動する
		NetFlash	bin ファイル	Linux が起動する
	コンフィグリージョン	ダウンローダ Hermit-At	bin ファイル	ブートローダ Hermit-At が起動する
		NetFlash	bin ファイル	Linux が起動する

[1]灰色になっているところは本書では説明しません。「SUZAKU ソフトウェアマニュアル」をご参照ください。
[2]SPI Writer を使用するには Xilinx Parallel Cable(III、IV)が必要になります。

6.2. FPGA の書き換えかた

SUZAKU の FPGA に FPGA ファイルを書き込むには、iMPACT を使って JTAG で FPGA に直接書き込む方法、iMPACT の DirectSPI モードや SPI Writer、ダウンローダ Hermit-At を使ってフラッシュメモリの FPGA リージョンに記憶させて書き込む方法があります。ここではこれらの方法について説明します。

フラッシュメモリの FPGA リージョンに書き込んだ FPGA ファイルが BBoot の起動しないものだった場合、ダウンローダ Hermit-At も起動しなくなります。何故そうなるのか分からない場合は「5.5. SUZAKU のブートシーケンス」をご参照ください。

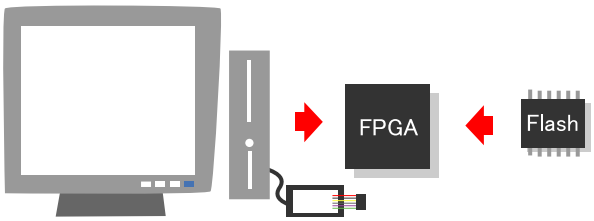


図 6.1 FPGA の書き込み

6.2.1. iMPACT で書き換える

iMPACT を使って JTAG で FPGA に直接 FPGA ファイルを書き込む方法を説明します。この方法は他と違い、電源を切ると書き込んだ内容が失なわれます。電源を切っても書き込んだ内容を失いたくない場合は、フラッシュメモリに FPGA ファイルを書き込む必要があります。ただし、Xilinx の FPGA が SRAM ベースのため、書き込みがとても速く、デバッグ時などには非常に重宝します。

6.2.1.1. iMPACT 書き込み準備

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、フラッシュメモリからのコンフィギュレーションデータの書き込みを停止させることができます。

SUZAKU CON7 にダウンロードケーブル、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、ダウンロードケーブルの LED が緑になっているか確認してください。

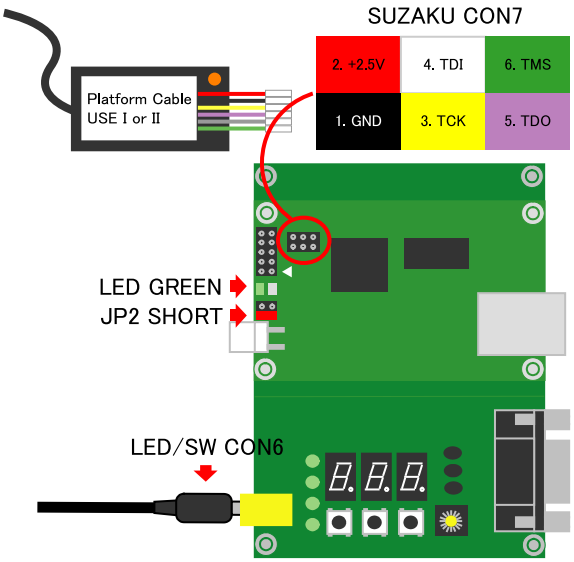


図 6.2 iMPACT 書き込み準備

6.2.1.2. iMPACT 起動から書き込み

iMPACT を起動してください。iMPACT は[スタートメニュー]→[すべてのプログラム]→[Xilinx ISE Design Suite x.x]→[ISE]→[アクセサリ]→[iMPACT]から起動できます。

iMPACT を初めて起動する場合は以下の画面が表示されます。[create a new project (ipf)]にチェックを入れ、[OK]をクリックしてください。表示されなかった場合は iMPACT のメニューで[File]→[New Project]を選択してください。

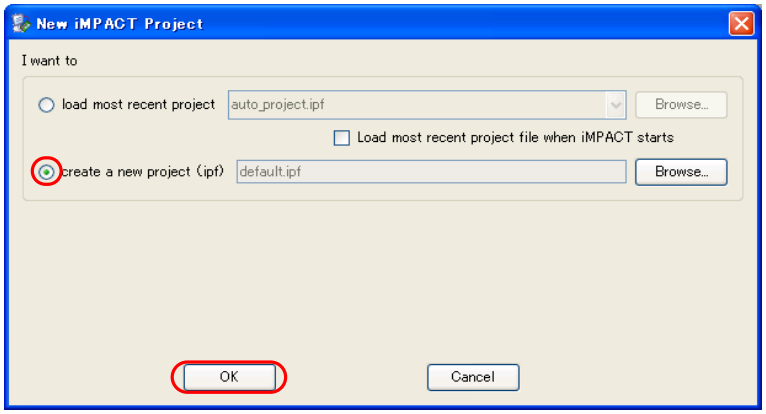


図 6.3 iMPACT 起動

[Configure devices using Boundary-Scan (JTAG)]にチェックを入れ、[OK]をクリックしてください。ダウンロードケーブルが自動で検索されます。もし、"Can not find cable, check cable setup!"と warning が表示された場合は、ダウンロードケーブルの接続に間違いがないか確認し、iMPACT のメニューで[Output]→[Cable Setup...]をクリックし、接続しているケーブルを選択してください。

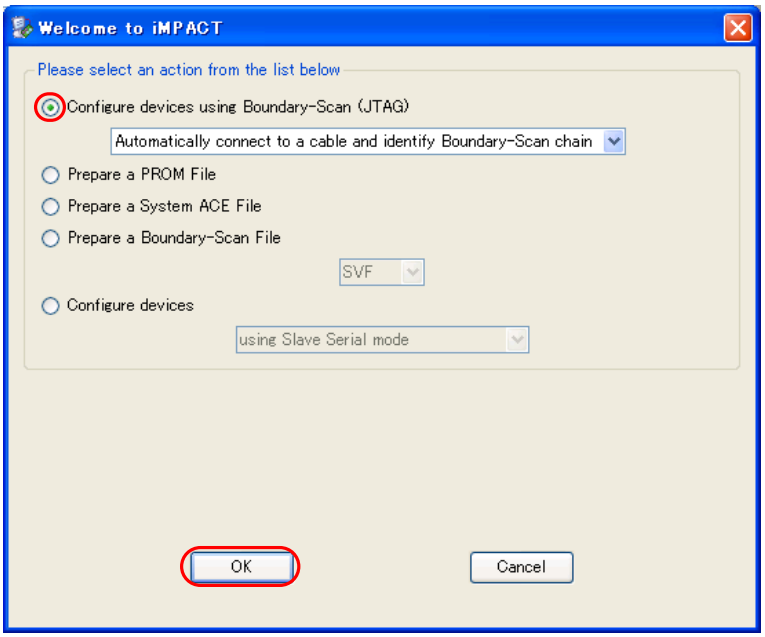


図 6.4 iMPACT 設定画面

ファイル選択画面が表示されるので書き込むファイルを選択し、[開く]をクリックしてください。書き込むのは **bit ファイル**です。ファイル選択画面が表示されなかった場合は、[Right click to Add Device or Initialize JTAG chain]の上で右クリックしてメニューを出し、[Add Xilinx Device...]を選択してください。

SUZAKU のデフォルトの bit ファイルおよびスロットマシンの bit ファイル(スターターキット出荷時の bit ファイル)は付属 CD-ROM[1]に収録しています。

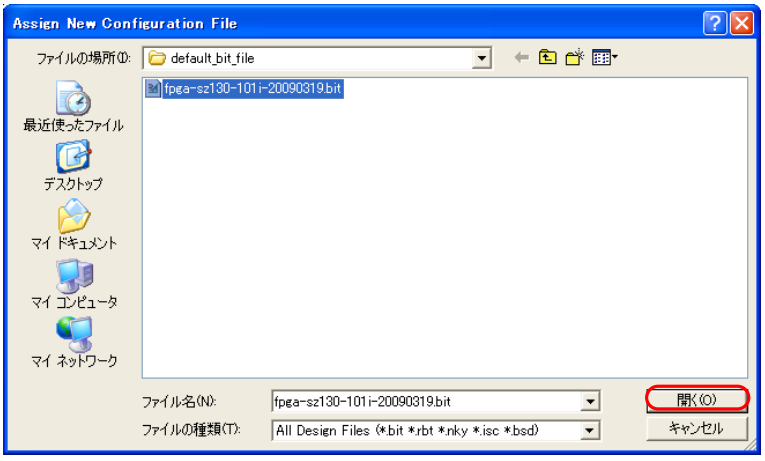


図 6.5 bit ファイル選択

SZ130 の場合、SPI フラッシュメモリに書き込むファイルがあるのか聞かれることありますが、[No]をクリックしてください。

[1] デフォルトの bit ファイルは"\suzaku\fpga\x.x\sz***\sz***-yyyymmdd.zip"を展開したフォルダの中の"default_bit_file"、スロットマシンの bit ファイルは"\suzaku-starter-kit\fpga\x.x\sz***\sz***-add_slot-yyyymmdd.zip"を展開したフォルダの中の"default_bit_file"に収録しています。

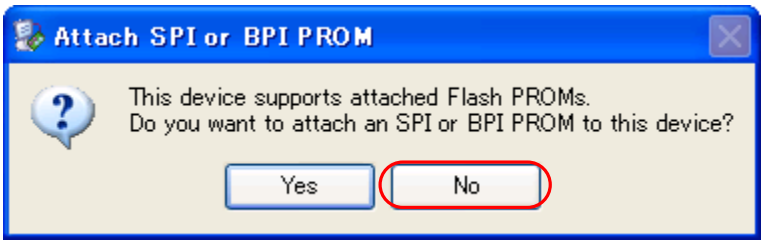


図 6.6 SPI フラッシュメモリへ書き込みファイル確認

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、下図のような状態になります。もしデバイスが発見されていなければ、接続を確認し、[File]→[Initialize Chain]をクリックしてください。

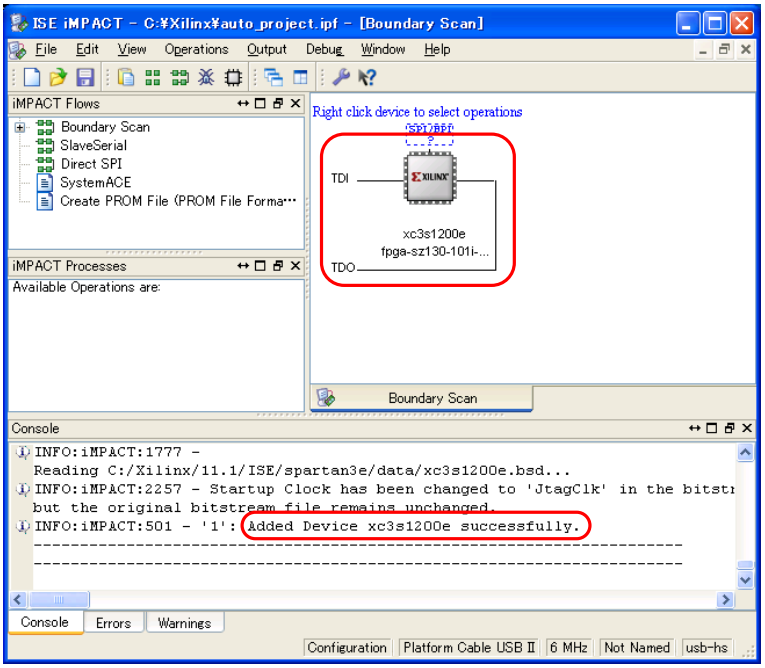


図 6.7 書き込み前の状態(SZ130 の場合)

デバイスをクリックし(灰色→緑色に変わります)、[Program]をダブルクリックしてください。

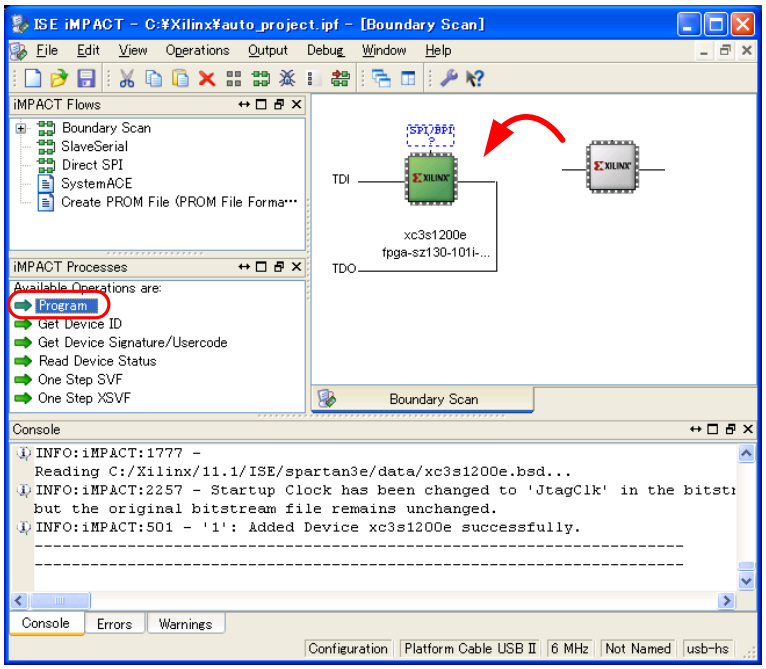


図 6.8 デバイス選択

書き込みオプションの設定画面が表示されます。何も変更せず、[OK]をクリックしてください。

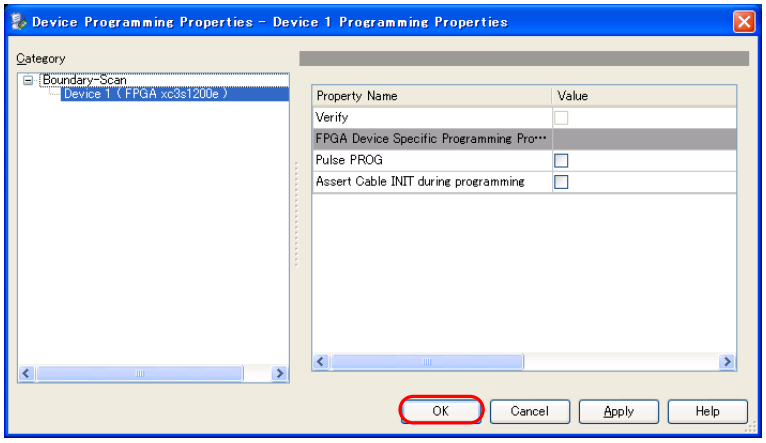


図 6.9 書き込みオプション設定

書き込みが開始します。"Program Succeeded"と表示されれば、書き込み成功です。接続状態によっては書き込みに失敗することもありますので、失敗した場合は接続状態を確認し(特にダウンロードケーブル周辺)再度書き込んでください。

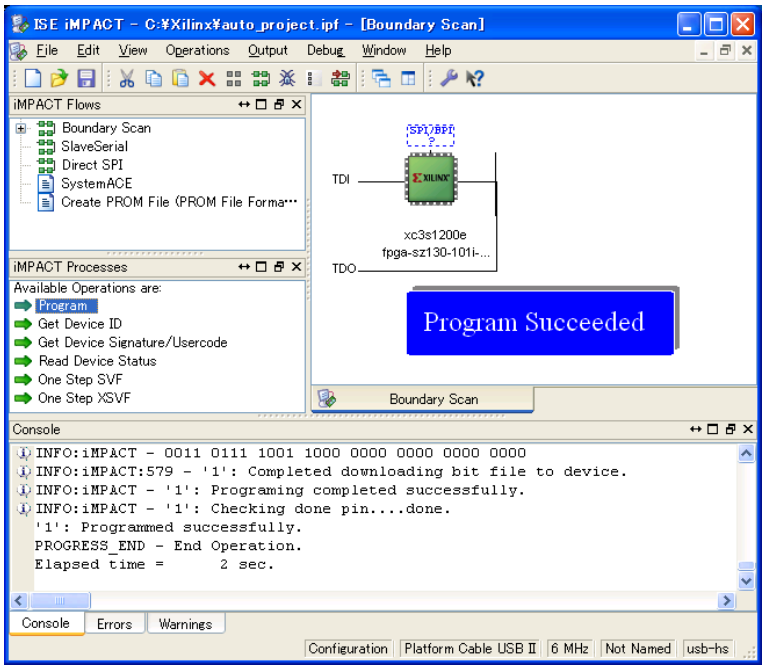


図 6.10 コンフィギュレーションデータ書き込み成功

6.2.1.3. iMPACT で書き換える 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)とダウンロードケーブルの LED(緑)が点灯していることを確認
5. iMPACT を起動し、データ書き込み

※電源を切ると、書き込んだ内容は失われます。

6.2.2. iMPACT の DirectSPI モードで書き換える

iMPACT の DirectSPI モード(以降 DirectSPI)を使って書き換える方法を説明します。DirectSPI は iMPACT の機能の一つで SPI フラッシュメモリを書き換えることができます。ただし DirectSPI は FPGA リージョンだけを書き換えることができず、ブートローダ、イメージリージョンまで消去してしまうため、FPGA、ブートローダ、イメージリージョンの 3 つのファイルを 1 つに結合して DirectSPI で書き込みます。

6.2.2.1. SZ130 の SPI フラッシュメモリ

SZ130

SZ130 の場合、コンフィギュレーションデータを M25P64(メーカ：ST マイクロエレクトロニクス)という SPI フラッシュメモリに記憶させ、再起動時に Spartan-3E の SPI モードで FPGA にコンフィギュレーションデータを書き込みます。SPI フラッシュメモリは SZ130 の裏面に実装しています。

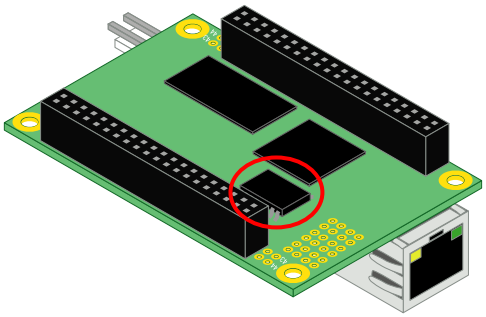


図 6.11 SZ130 の SPI フラッシュメモリの所在

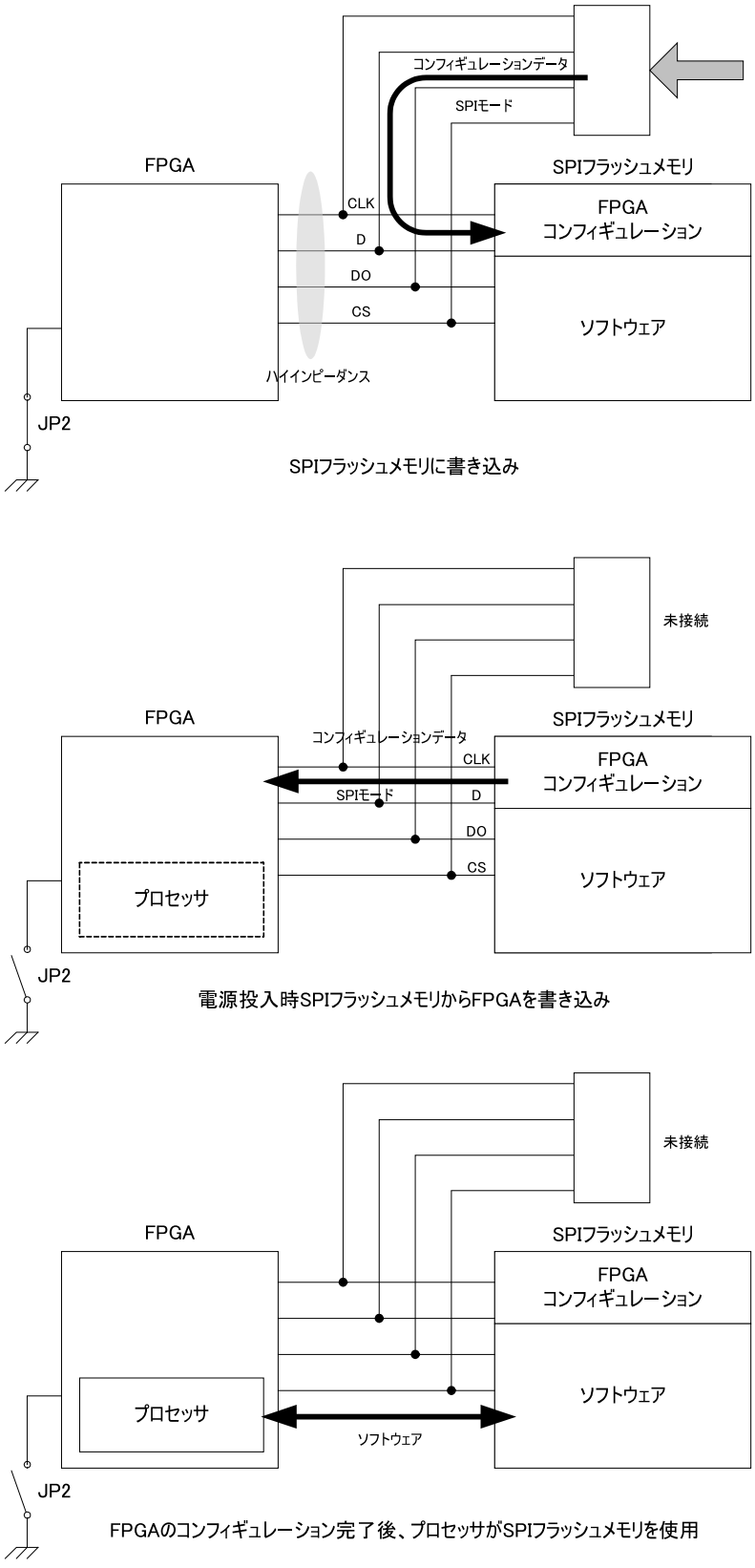


図 6.12 SPI モードの書き込み(SZ130)

6.2.2.2. SZ410 の SPI フラッシュメモリ

SZ410

SZ410 の場合、コンフィギュレーションデータを M25P64(メーカ：ST マイクロエレクトロニクス)という SPI フラッシュメモリに記憶させ、再起動時に CPLD で FPGA にコンフィギュレーションデータを書き込んでいます。CPLD および SPI フラッシュメモリは下図の位置に実装しています。

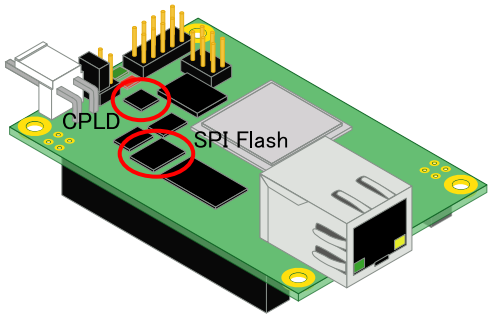


図 6.13 SZ410 の CPLD および SPI フラッシュメモリの所在

コンフィギュレーション時の CPLD の動作は以下の通りとなります。

1. JP2 をショートしてコンフィギュレーションデータを書き込むと、コンフィギュレーションデータを SPI フラッシュメモリにスルーで書き込む。
2. JP2 をオープンにして電源を投入すると、SPI のビットストリームをマスターシリアルに変換してコンフィギュレーションする。
3. コンフィギュレーション後は SPI フラッシュメモリのデータ線を開放し、SPI フラッシュメモリに FPGA の制御を渡す。

CPLD の中身は？

SZ410 に実装している CPLD で使用している VHDL コードは、Xilinx の XAPP800 というドキュメントを元に作成しています。興味のある方は探してみてください。

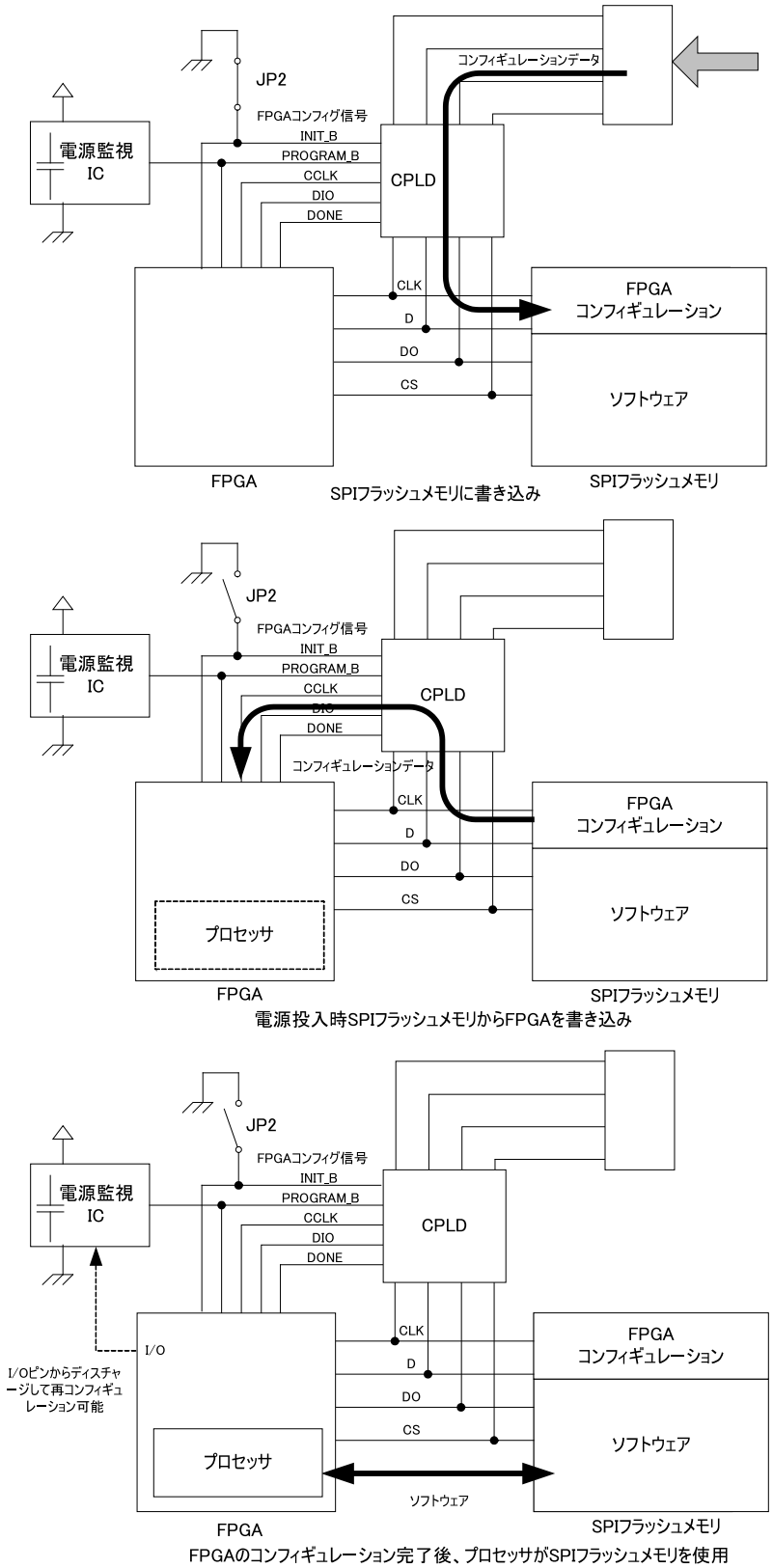


図 6.14 CPLD による書き込み(SZ410)

6.2.2.3. mcs ファイルを作る

DirectSPI で書き込めるファイルは **mcs ファイル**です。mcs ファイルは iMPACT で作成するか、PROMGen プログラムにより作成することができます。iMPACT も裏で PROMGen プログラムをよびだしています。GUI があると分かり安いので、ここでは iMPACT で mcs ファイルを作ります。SZ410 の場合、Virtex-4 FX が iMPACT でサポートされていないので、少しトリッキーな方法で mcs ファイルを作成します。

まずは FPGA、ブートローダ、イメージリージョンの 3 つのファイルを結合した mcs ファイルを作成します。書き込みたい FPGA の bit ファイル、ブートローダ、イメージリージョンに書き込む bin ファイルを準備してください。ブートローダ、イメージリージョンに書き込むデフォルトの bin ファイルは付属 CD-ROM[2]に収録しています。

iMPACT を起動してください。iMPACT は[スタートメニュー]→[すべてのプログラム]→[Xilinx ISE Design Suite x.x]→[ISE]→[アクセサリ]→[iMPACT]から起動できます。

iMPACT を初めて起動する場合は以下の画面が表示されます。[create a new project (ipf)]にチェックを入れ、[OK]をクリックしてください。表示されなかった場合は iMPACT のメニューで[File]→[New Project]を選択してください。

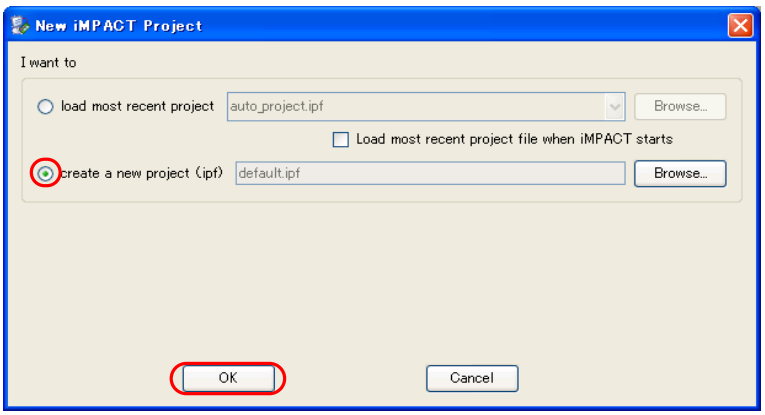


図 6.15 iMPACT 起動

[Prepare a PROM File]にチェックを入れ、[OK]をクリックしてください

[2]ブートローダ、イメージリージョンに書き込むデフォルトの bin ファイルは"\suzaku\image\"以下に収録しています。

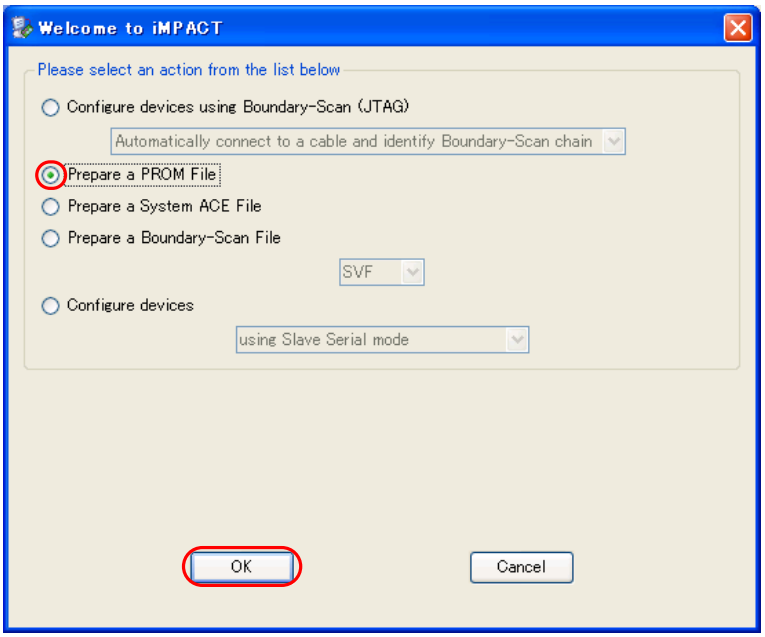


図 6.16 Prepare PROM File をチェック

PROM ファイルの設定をします。

Select Strage Target で[SPI Flash]→[Configure Single FPGA]を選択し、緑の矢印をクリックしてください。次に Add Strage Device(s)の Strage Device(bits)で[64M]を選択し、[Add Strage Device]をクリックし、緑の矢印をクリックしてください。Enter Data で Checksum Fill Value を[FF]にし、Output File Name に作成する mcs ファイルのファイル名を入力し(ここでは`[suzaku-all]`としています)、Output File Location に mcs ファイルを保存するフォルダを指定し(ここでは`[C:\suzaku\image]`としています)、File Format を[MCS]にし、Add Non-Configuration Data Files を[Yes]にしてください。すべて設定が終わったら[OK]をクリックしてください。

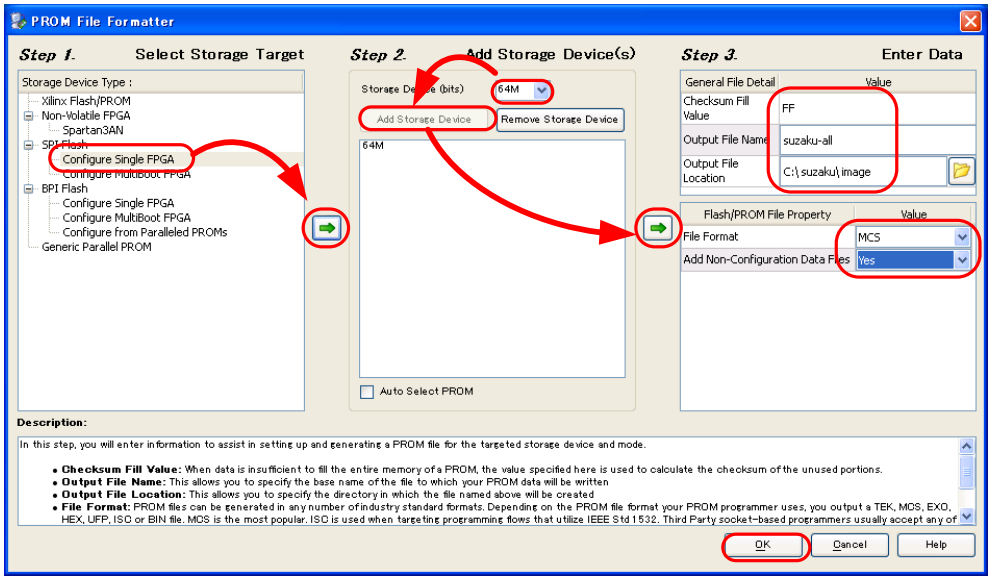


図 6.17 PROM ファイルの設定

デバイスファイルを追加するか聞かれるので[OK]をクリックしてください。

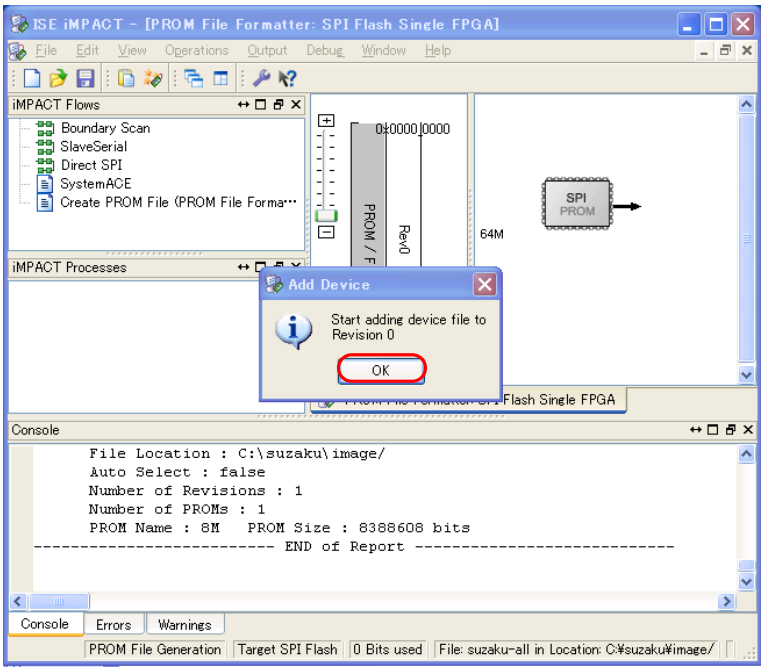


図 6.18 デバイスファイル追加

ファイル選択画面が表示されるので、書き込むデータを選択し[開く]をクリックしてください。SUZAKU のデフォルトの bit ファイルおよびスロットマシンの bit ファイル(スターターキット出荷時の bit ファイル)は付属 CD-ROM[3]に収録しています。SZ410 の場合、SZ410 用の bit ファイルは選択することができないので、とりあえず SZ130 用の bit ファイルを選択してください。

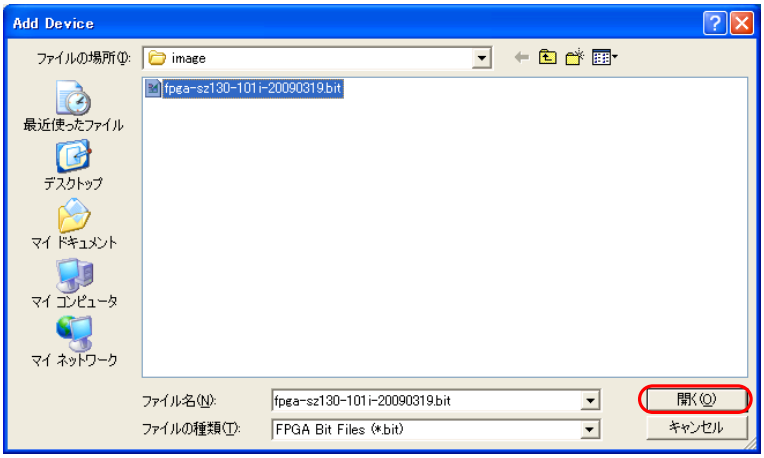


図 6.19 SZ130 用 bit ファイル選択

デバイスを他にも追加するか聞かれるので[No]をクリックしてください。

[3] デフォルトの bit ファイルは"\suzaku\fpga\x.x\sz***\sz***-yyyymmdd.zip"を展開したフォルダの中の"default_bit_file"、スロットマシンの bit ファイルは"\suzaku-starter-kit\fpga\x.x\sz***\sz***-add_slot-yyyymmdd.zip"を展開したフォルダの中の"default_bit_file"に収録しています。

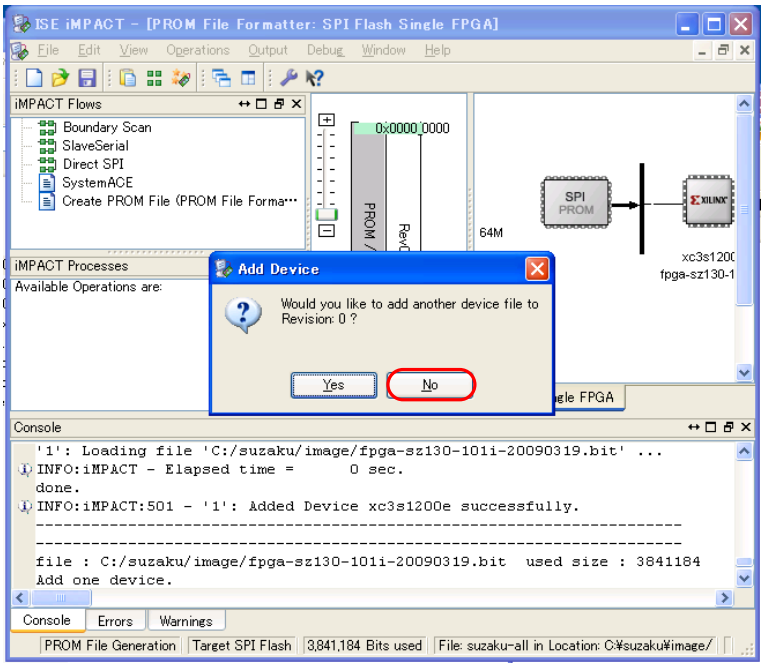


図 6.20 デバイス追加確認

データファイルを追加するか聞かれるので[Yes]をクリックしてください。

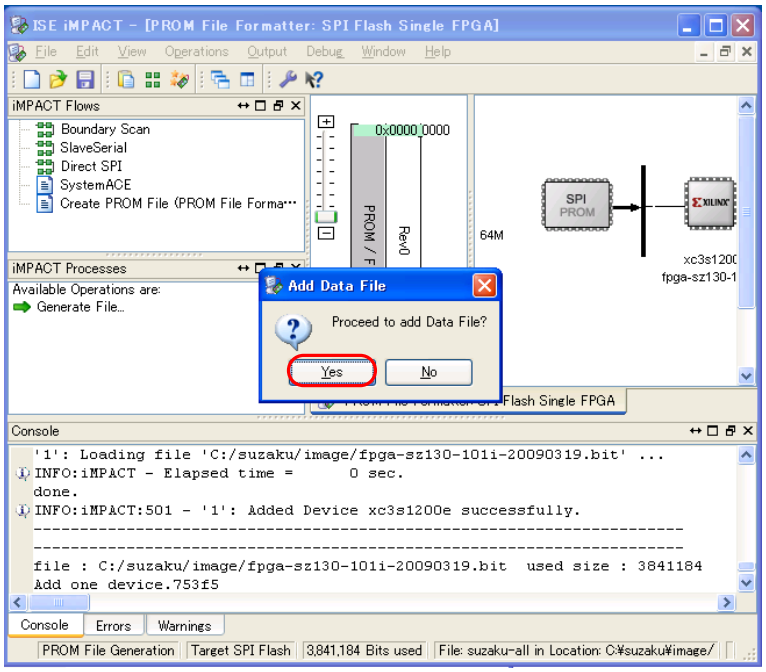


図 6.21 データファイル追加

スタートアドレスを聞かれるので、Start Address にブートローダリージョンの先頭番地[100000]を入力して[OK]をクリックしてください。

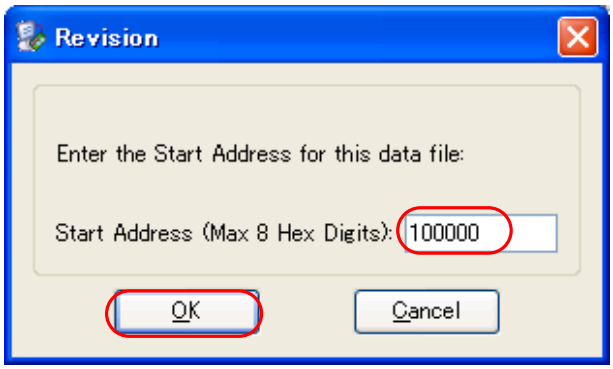


図 6.22 ブートローダリージョンの先頭番地

ファイルの種類を All Files(*.*)に変更し、ブートローダファイルを選択して[開く]をクリックしてください。デフォルトのブートローダの bin ファイルは付属 CD-ROM[4]に収録しています。

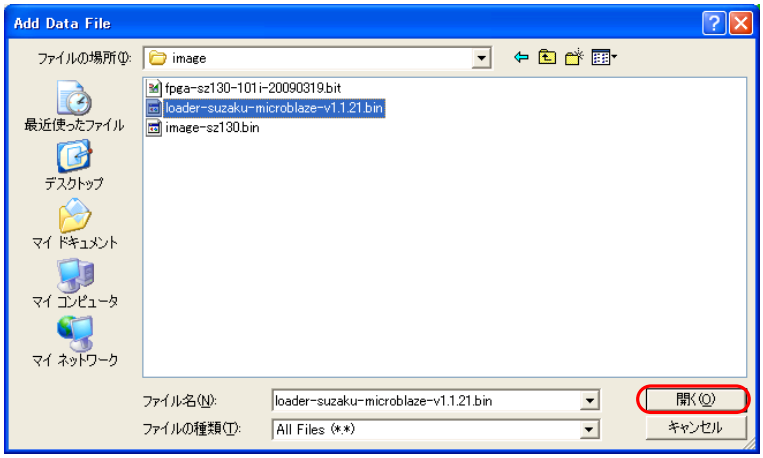


図 6.23 ブートローダファイル選択

さらにデータファイルを追加するか聞かれるので[Yes]をクリックしてください。

[4]SZ130 は "\suzaku\image\loader-suzaku-microblaze-vx.x.x.bin" 、 SZ410 は "\suzaku\image\loader-suzaku-powerpc-vx.x.x.bin"に収録しています。

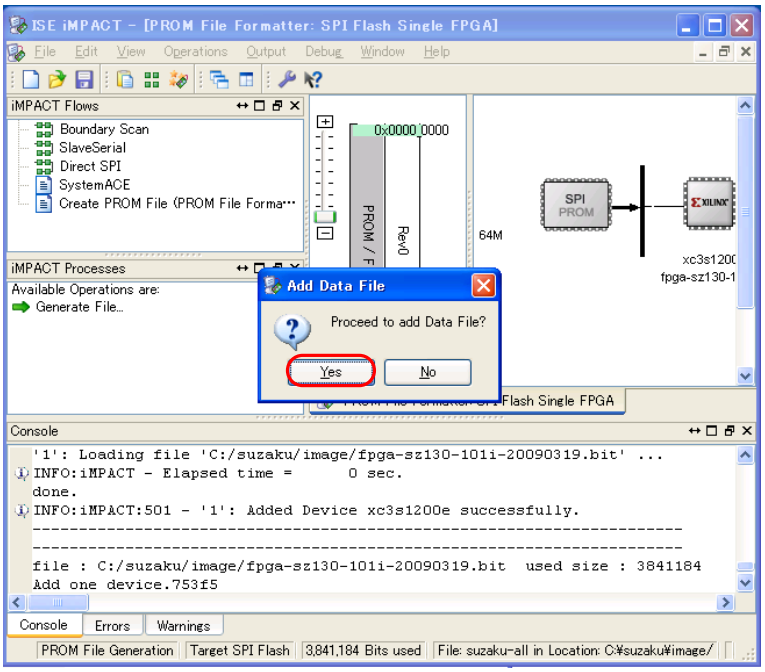


図 6.24 データファイル追加

再びスタートアドレスを聞かれるので、Start Address にイメージリージョンの先頭番地[120000]を入力して[OK]をクリックしてください。

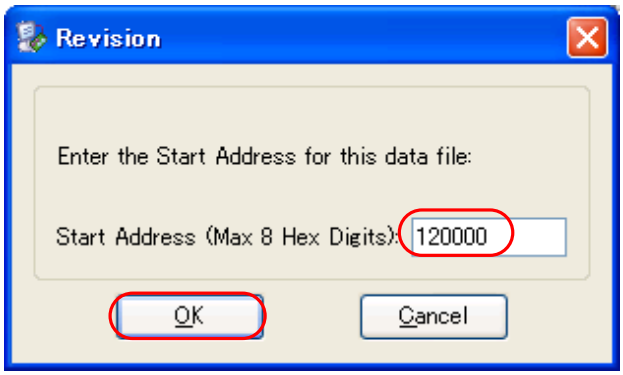


図 6.25 Linux イメージリージョンの先頭番地

ファイルの種類を All Files(*.*)に変更し、Linux イメージファイルを選択して[開く]をクリックしてください。デフォルトの Linux イメージの bin ファイルは付属 CD-ROM[5]に収録しています。

[5]SZ130 は"\suzaku\image\sz130-image.bin"、SZ410 は"\suzaku\image\sz410-image.bin"に収録しています。

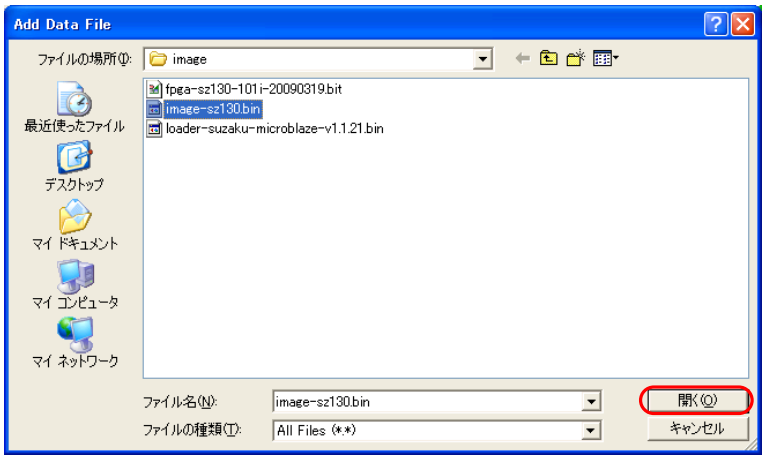


図 6.26 Linux イメージファイル選択

さらにデータファイルを追加するか聞かれますが、もう追加するファイルはないので[No]をクリックしてください。

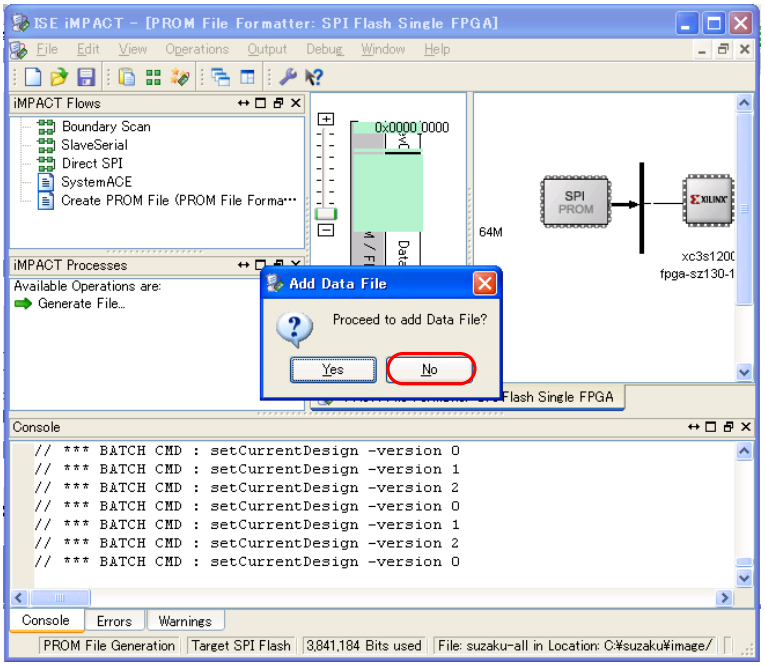


図 6.27 データファイル追加

これでファイルの設定完了です。[OK]をクリックしてください。

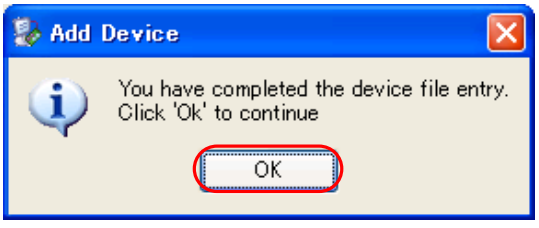


図 6.28 ファイル設定終了

追加したファイルの情報が表示されます。間違いがないか確認をし、[OK]をクリックしてください。

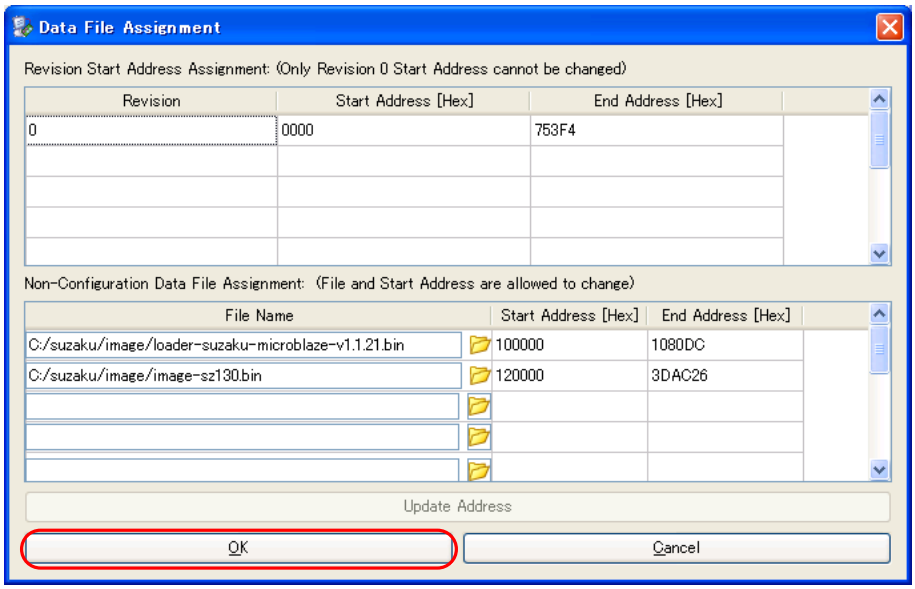


図 6.29 ファイルの確認

SZ410

SZ410 の場合、bit ファイルを選択しなおします。先ほどは SZ410 用のファイルが選択できませんでしたが、今度は選択できるようになっています。SZ130 の場合は選択しなおす必要はありません。デバイスを右クリックしてメニューを出し、[Assign New Configuration File]を選択してください。ファイルの選択画面が表示されるので、SZ410 用の bit ファイルを選択して、[開く]をクリックしてください。

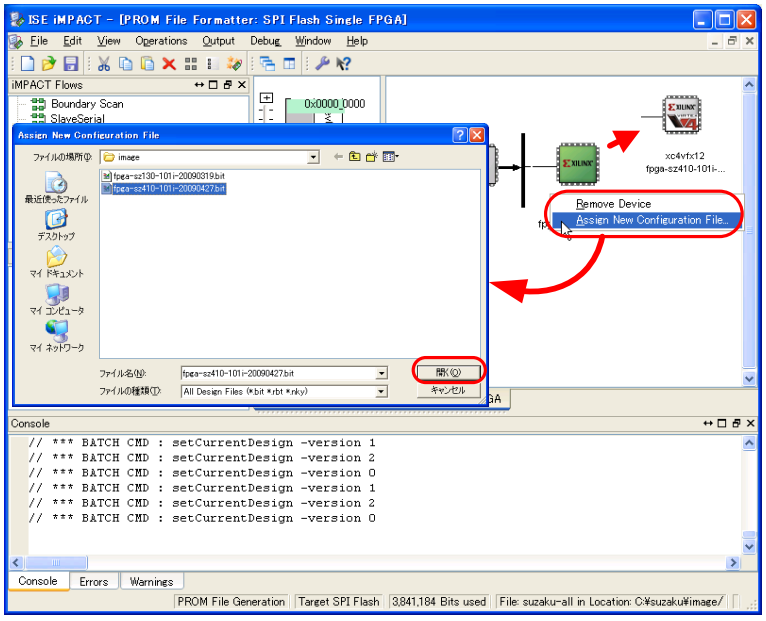


図 6.30 SZ410 の場合はファイル選択のやり直し

これで準備が整いました。Operations の[Generate File]をダブルクリックしてください。"Generate Succeeded"と表示されたら、mcs ファイルの作成完了です。頻繁に mcs ファイルを作成する人は何度

も同じ手順を繰り返すのが大変なので、[File]→[Save Project]をクリックし、プロジェクトを保存してください。

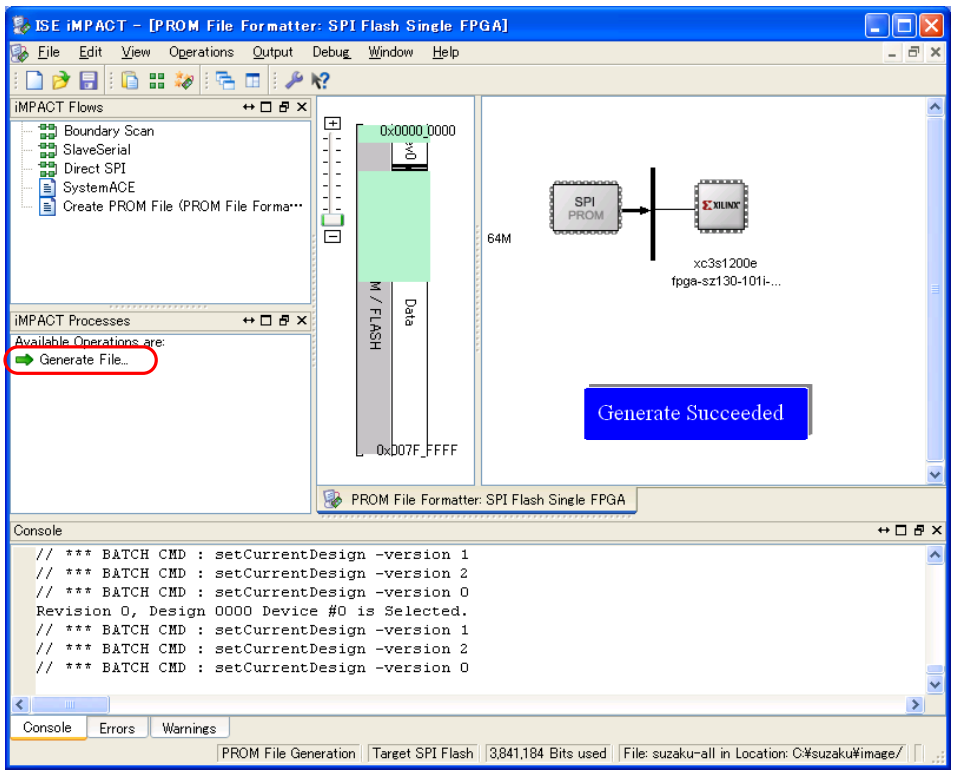


図 6.31 mcs ファイル作成完了

6.2.2.4. DirectSPI 書き込み準備

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、フラッシュメモリからのコンフィギュレーションデータの書き込みを停止させることができます。

LED/SW CON4(1 ～ 6 ピン)にダウンロードケーブルを接続し、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、ダウンロードケーブルの LED が緑になっているか確認してください。

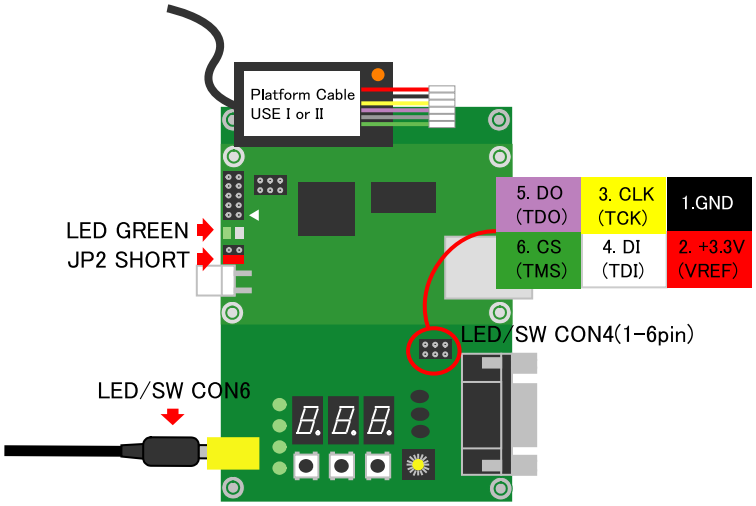


図 6.32 DirectSPIr 書き込み準備

6.2.2.5. DirectSPI の起動から書き込み

iMPACTを起動してください。先程からの続きで起動したままの場合は iMPACT のメニューで[File]→[New Project]を選択してください。

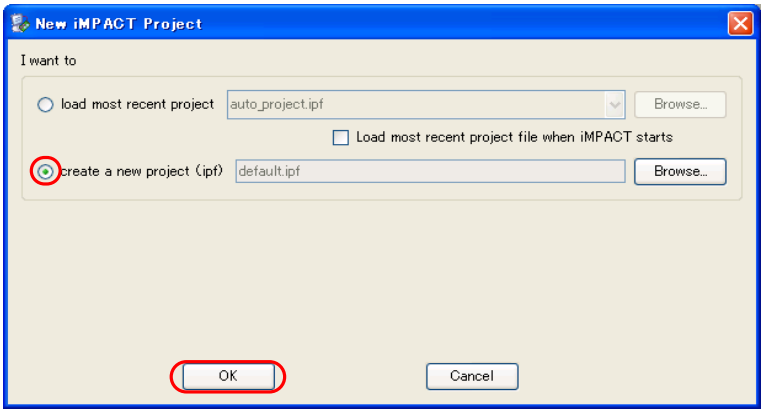


図 6.33 iMPACT 起動

Configure devices の[using Direct SPI Configuration mode]を選択して、[OK]をクリックしてください。

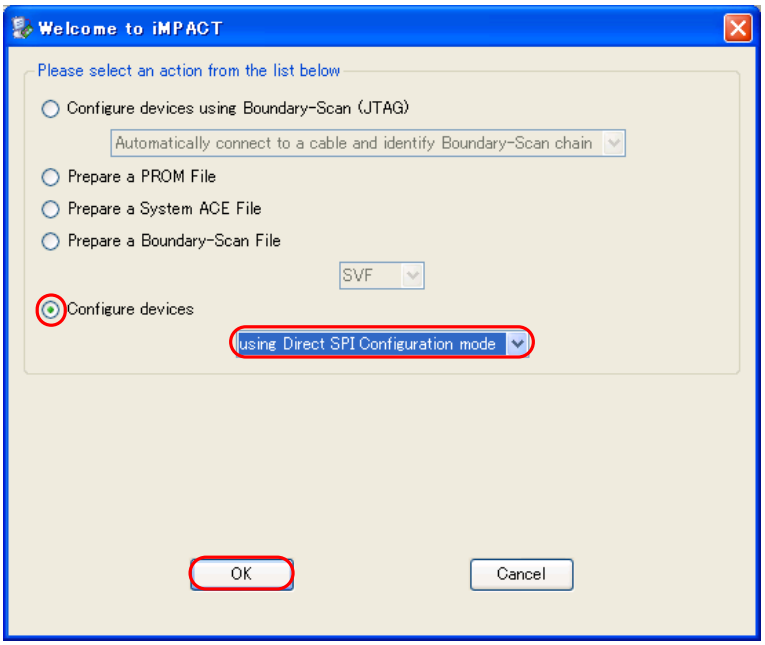


図 6.34 DirectSPI 開始

[Right click Add Device or Identify Device]の上で右クリックしてメニューを出し、[Add SPI Device]を選択してください。ファイル選択画面が表示されるので、先程作成した mcs ファイルを選択して、[開く]をクリックしてください。

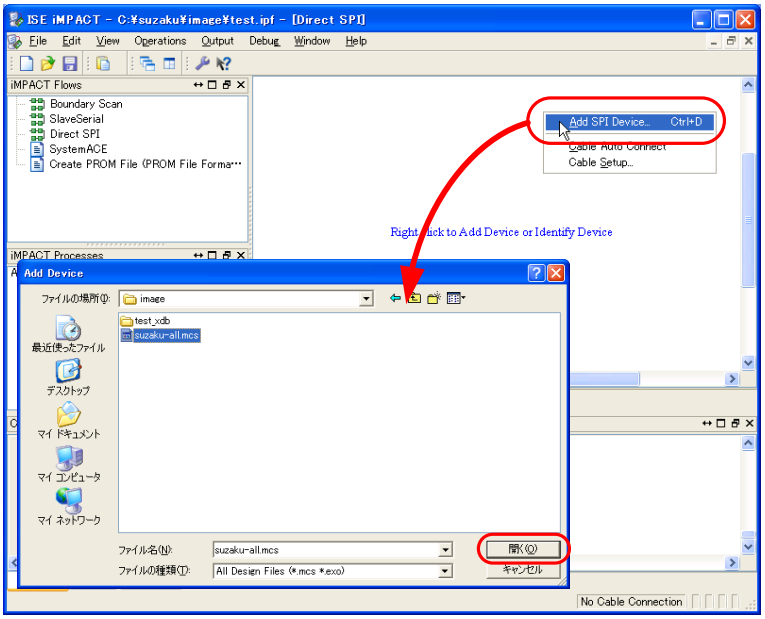


図 6.35 DirectSPI 用ファイル選択

デバイスの種類を聞かれるので、[M25P64]を選択して、[OK]をクリックしてください。

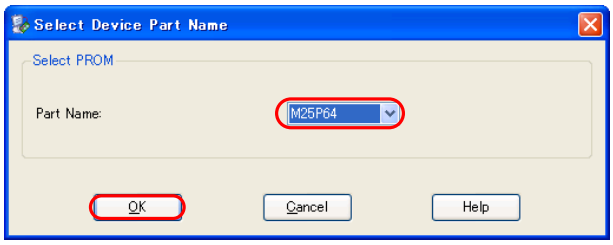


図 6.36 M25P64 選択

[Verify]と[Erase Before Programming]にそれぞれチェックを入れ、[OK]をクリックしてください。

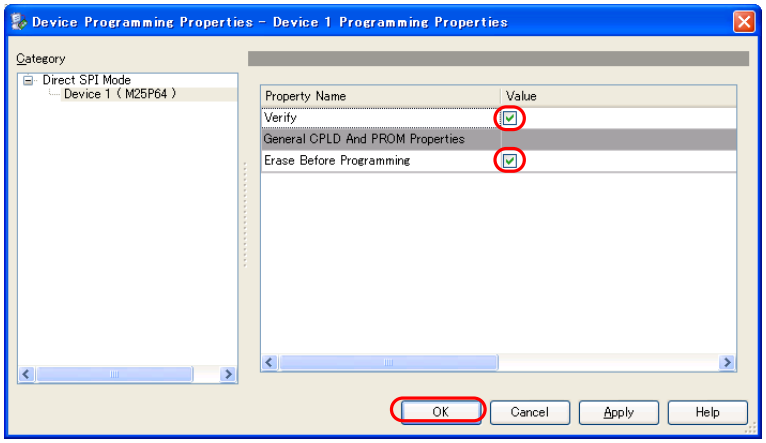


図 6.37 DirectSPI の設定

これで DirectSPI の設定完了です。デバイスをクリックし(灰色→緑色に変わります)、[Program]をダブルクリックしてください。

書き込みが終わるまで少々時間がかかります。"Program Succeeded"と表示されたら書き込み終了です。

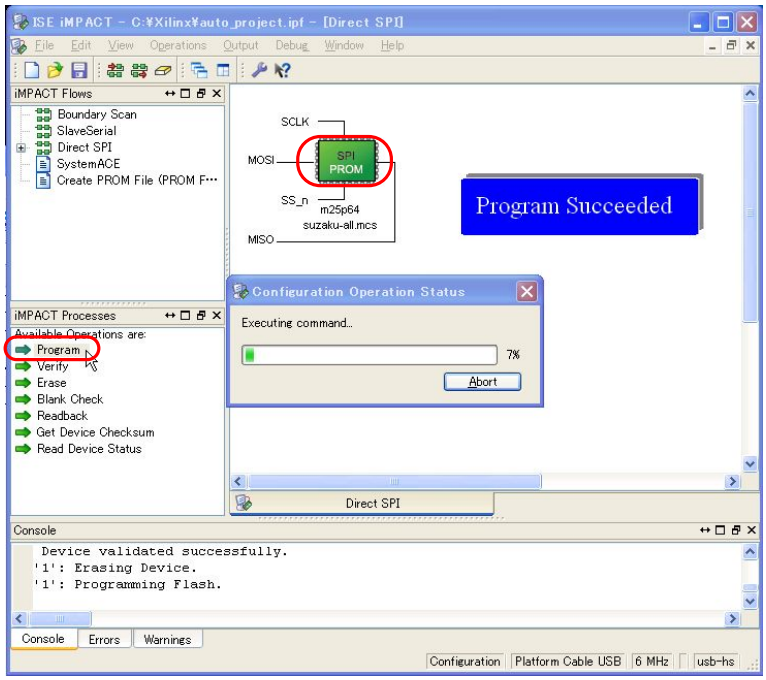


図 6.38 DirectSPI 書き込み終了

LED/SW CON6 から AC アダプタ 5V を抜いて電源を切り、ジャンパプラグと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。

6.2.2.6. DirectSPI で書き換える 手順まとめ

1. iMPACT を起動して mcs ファイル作成
2. SUZAKU JP2 にジャンパプラグをさしてショートさせる
3. LED/SW CON4 にダウンロードケーブルを接続する
4. LED/SW CON6 に AC アダプタ 5V を接続し、電源投入
5. SUZAKU D3 のパワー ON LED(緑)が点灯していることを確認
6. DirectSPI を起動して SPI フラッシュメモリにデータを書き込む

※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。

6.2.3. SPI Writer で書き換える

SPI Writer を使って書き換える方法を説明します。Xilinx Parallel Cable(III、IV)の場合は、SPI Writer が使用できます。[6]

SPI Writer は FPGA リージョンだけを書き換えることができるので、DirectSPI よりも速く書き換えることができます。

6.2.3.1. SPI Writer 書き込み準備

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、フラッシュメモリ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 にダウンロードケーブルを接続し、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。

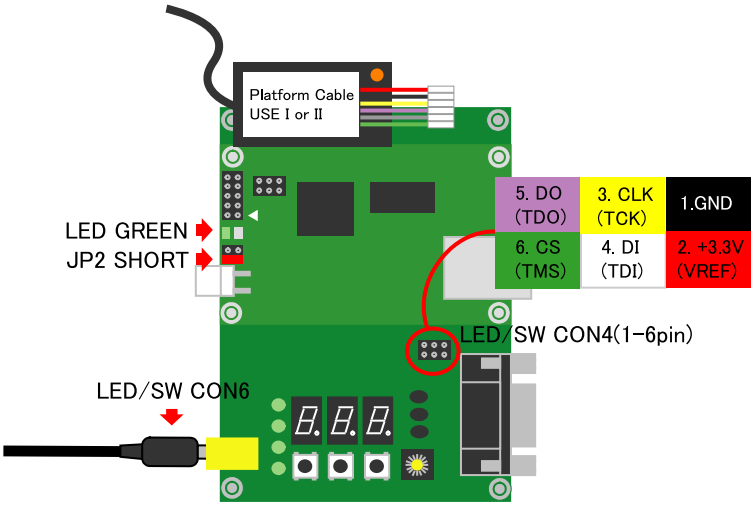


図 6.39 SPI Writer 書き込み準備

6.2.3.2. SPI Writer 起動から書き込み

SPI Writer を起動し、[...]をクリックしてください。ファイル選択画面が立ち上がります。

書き込むファイルを選択し、[開く]をクリックしてください。SPI Writer で書き込めるファイルは **bit ファイル**です。

[6]Platform Cable USB(I、II)の場合は使用できませんのでご注意ください。

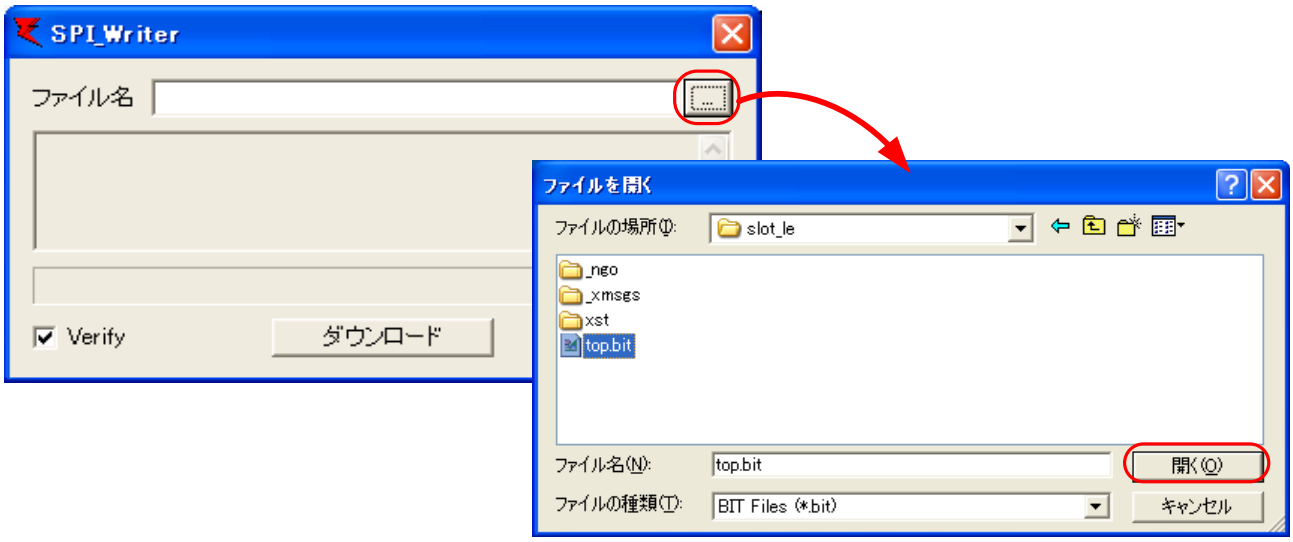


図 6.40 SPI_Writer

SUZAKU のデフォルトの bit ファイルおよびスロットマシンの bit ファイル(スターターキット出荷時の bit ファイル)は付属 CD-ROM[7]に収録しています。

SPI Write でのファイルの選択方法

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。

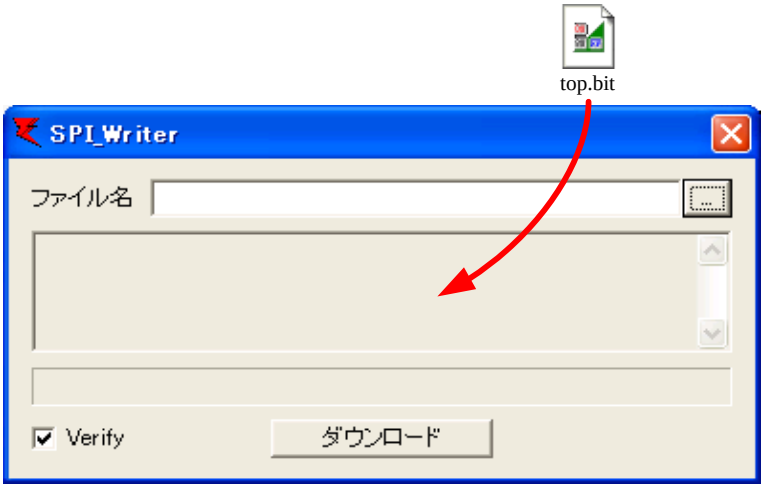


図 6.41 ドラッグ&ドロップ

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。

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。

[7] デフォルトの bit ファイルは"\suzaku\fpga\x.x\sz***\sz***-yyyymmdd.zip"を展開したフォルダの中の"default_bit_file"、スロットマシンの bit ファイルは"\suzaku-starter-kit\fpga\x.x\sz***\sz***-add_slot-yyyymmdd.zip"を展開したフォルダの中の"default_bit_file"に収録しています。

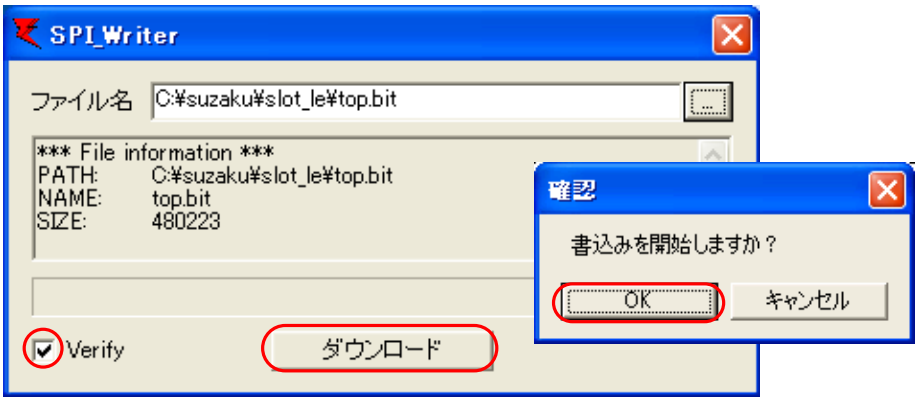


図 6.42 書き込み準備完了

コンフィギュレーションデータが SPI フラッシュメモリに書き込まれます。ここで"Please check windrvr.sys"というエラーが発生した場合は「Tips SPI Writer ERROR」をご参照ください。

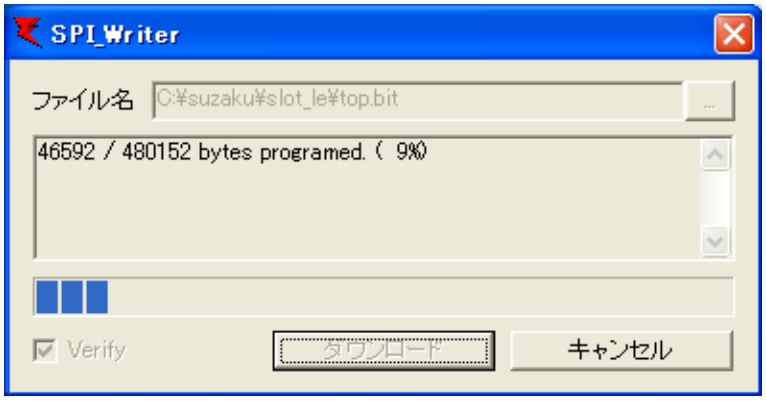


図 6.43 書き込み中

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。

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、再び書き込みを行ってください。

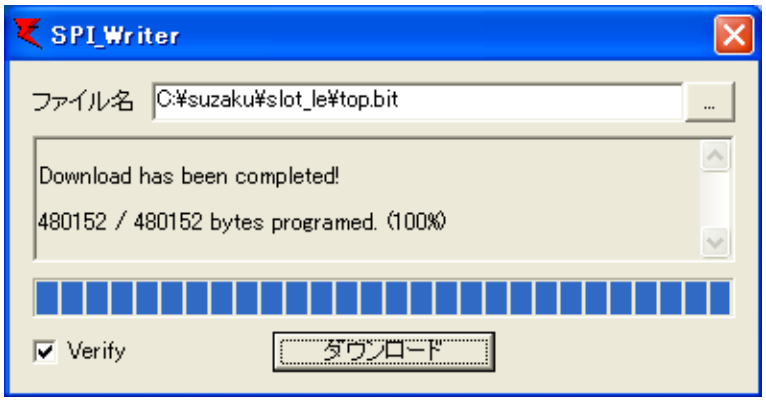


図 6.44 書き込み終了

LED/SW CON6 から AC アダプタ 5V を抜いて電源を切り、ジャンパプラグと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。

6.2.3.3. SPI Writer で書き換える 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. 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 を起動して SPI フラッシュメモリにコンフィギュレーションデータを書き込む

※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。

SPI Writer とは

SPI Writer は SPI フラッシュメモリの先頭から 1MB 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 フラッシュメモリ専用の書き込みツールです。

SUZAKU は SPI フラッシュメモリ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ずに書き込むことができます。

SPI Writer ERROR

SPI Writer で書き込む際、以下のエラーが出ることがあります。この場合ドライバのインストールが必要となります。

図 6.45 エラー表示

SPI Writer のフォルダの中に `wdreg.exe`、`difxapi.dll`、`wd811.cat`、`windrvr6.inf`、`windrvr6.sys` の 5 つのファイルがあることを確認してください。

コマンドプロンプトを起動して SPI Writer のフォルダに移動し、Administrator 権限ユーザで以下のコマンドを実行してください。

```
> wdreg -inf windrvr6.inf install
```

以下のようなログが表示されます。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。

```
> Installing a signed driver package
LOG Event: 1, ENTER:  DriverPackageInstallA
LOG Event: 1, ENTER:  DriverPackageInstallW
LOG Event: 1, Looking for Model Section [DeviceList]...
LOG Event: 1, windrvr6.inf: checking signature with catalog
'C:\spi_writer-20070119\wd811.cat' ...
LOG Event: 1, Driver package 'windrvr6.inf' is Authenticode
signed.
LOG Event: 1, Copied 'windrvr6.inf' to driver store...
LOG Event: 1, Copied 'wd811.cat' to driver store...
LOG Event: 1, Commiting queue...
LOG Event: 1, Copied file: 'C:\spi_writer-20070119\.
\windrvr6.sys' -> 'C:\WINDOWS\system32\DRVSTORE
\windrvr6_45AF516B2C99AB8FE1C0F3A3CBE523C199AE6F2B\.
\windrvr6.sys'.
LOG Event: 1, Installing INF file "C:\WINDOWS
\system32\DRVSTORE
\windrvr6_45AF516B2C99AB8FE1C0F3A3CBE523C199AE6F2B
\windrvr6.inf" of Type 6.
LOG Event: 1, Looking for Model Section [DeviceList]...
LOG Event: 1, Installing devices with Id "*WINDRVR6" using
INF "C:\WINDOWS\system32\DRVSTORE
\windrvr6_45AF516B2C99AB8FE1C0F3A3CBE523C199AE6F2B
\windrvrinstall: completed successfully
```

6.2.4. ダウンローダ Hermit-At で書き換える

ダウンローダ Hermit-At でを書き換える方法を説明します。

6.2.4.1. bin ファイルを作る

ダウンローダ Hermit-At で使用するファイルは **bin ファイル**です。bin ファイルは bit ファイルをバイナリエディタで加工して作成します。

SUZAKU のデフォルトの bin ファイルおよびスロットマシンの bin ファイル(スターターキット出荷時の bin ファイル)は付属 CD-ROM[8]に収録しています。

bit ファイルをバイナリエディタで開いてください。全く同じではありませんが、以下のような内容になっていると思います。

```
00 09 0F F0 0F F0 0F F0 0F F0 00 00 01 61 00 0D
78 70 73 5F 70 72 6F 6A 2E 6E 63 64 00 62 00 0A
32 76 70 34 66 67 32 35 36 00 63 00 0B 32 30 30
37 2F 31 30 2F 30 36 00 64 00 09 31 37 3A 32 32
3A 33 37 00 65 00 05 BC 04 FF FF FF FF AA 99 55
66 30 00 80 01 00 00 00 07 30 01 60 01 00 00 00
69 30 01 20 01 00 04 3F E5 30 01 C0 01 01 23 E0
93 30 00 C0 01 00 00 00 00 30 00 80 01 00 00 00
09 30 00 20 01 00 00 00 00 30 00 80 01 00 00 00
...
```

途中に**"FF FF FF FF"**があるので、その直前までを削除してください。削除できたら、拡張子を bin に変更して保存してください。これで bin ファイル作成完了です。

```
FF FF FF FF AA 99 55 66 30 00 80 01 00 00 00 07
30 01 60 01 00 00 00 69 30 01 20 01 00 04 3F E5
30 01 C0 01 01 23 E0 93 30 00 C0 01 00 00 00 00
30 00 80 01 00 00 00 09 30 00 20 01 00 00 00 00
30 00 80 01 00 00 00
...
```

6.2.4.2. Hermit-At での書き込み準備

まず、JP1 にジャンパプラグをさし、ショートさせてください。JP1 をショートさせるとブートローダモードになります。

SUZAKU CON1 に方向に気をつけてシリアルケーブルを接続し、シリアル通信ソフトウェアを起動してください。(「5.1. シリアル通信ソフトウェアの起動」参照)。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。

スターターキット出荷時の FPGA から書き換えていない場合、シリアル通信ソフトウェアの画面に以下のようなメッセージが表示されます。

```
Please choose one of the following and hit enter.
a: active second stage bootloader (default)
```

[8] デフォルトの bin ファイルは"\suzaku\image\fpga-sz***-xxi-yyyymmdd.bin"に、スロットマシンの bin ファイルは"\suzaku-starter-kit\image\fpga-sz***-sil-xxi-yyyymmdd.bin"に収録。

```
s: download a s-record file
t: busy loop type slot-machine
```

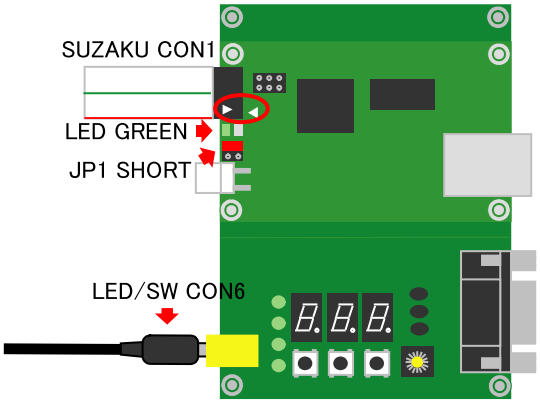


図 6.46 書き込み準備

"a"もしくは"Enter"キーを押してください。ブートローダ Hermit-At が起動します。以下のようなメッセージが表示されます。もし起動しない場合は「6.3. ブートローダ Hermit-At の書き換えかた」を参照してブートローダ Hermit-At を書き込んでください。

```
Hermit-At v1.1.21 (suzaku/microblaze) compiled at 18:40:12, Mar 25 2009
hermit>
```

ブートローダ Hermit-At が起動したのを確認したら、シリアルポートをシリアル通信ソフトウェアから切断します。[ファイル]→[接続断]を選択してください。

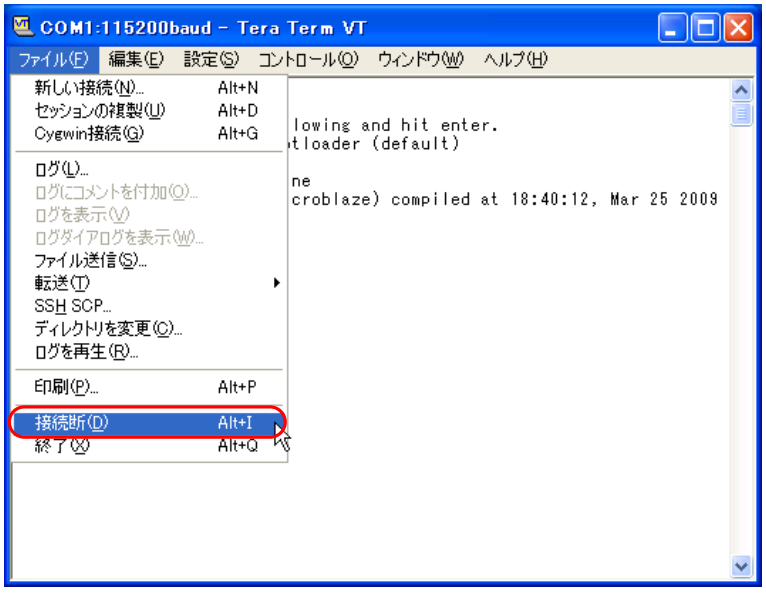


図 6.47 シリアルポートを切断

6.2.4.3. ダウンローダ Hermit-At 起動から書き込み

ダウンローダ Hermit-At を起動してください。[Download]ボタンをクリックすると Download 画面が表示されます。[Serial Port]に SUZAKU と接続しているシリアルポートを設定し、[Image]に先程作成した bin ファイルを選択してください。

[Region]を[fpga]にして[ForceLocked]をチェックし、[実行]をクリックしてください。

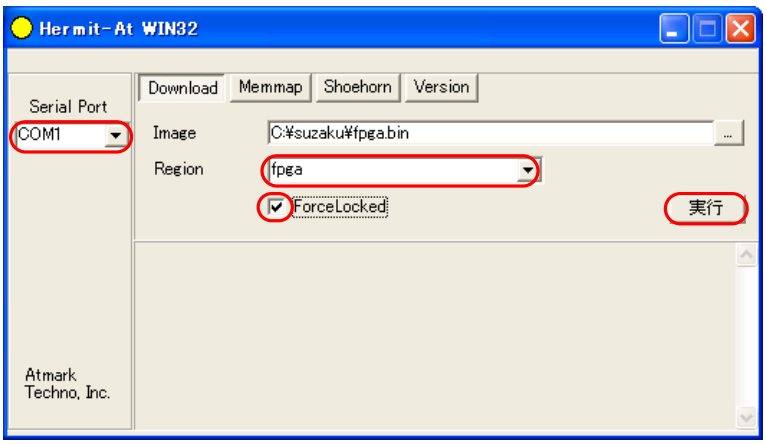


図 6.48 Download 画面

フラッシュメモリへの書き込みが始まります。書き込み中は、進捗状況が表示されます。

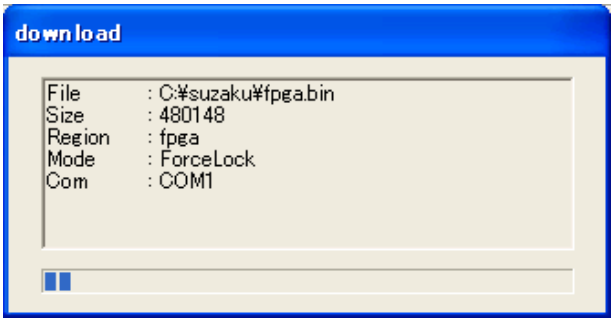


図 6.49 書き込み進捗ダイアログ

書き込みが終了すると、書き込み終了画面が表示されます。"Download COMPLETE"と表示されたら書き込み成功です。

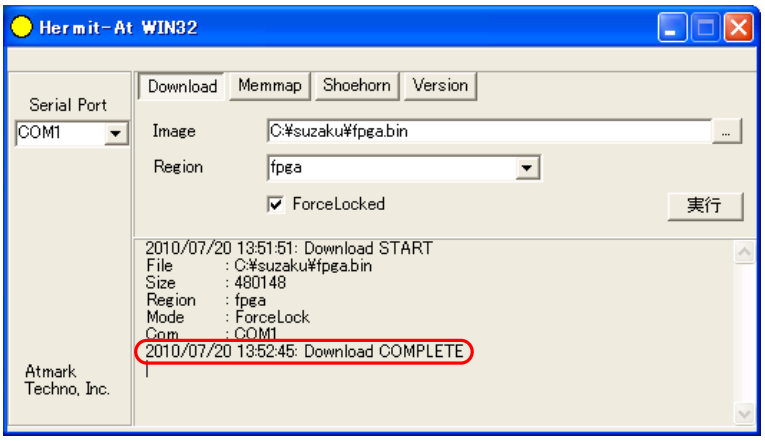


図 6.50 書き込み終了

LED/SW CON6 から AC アダプタ 5V を抜いて電源を切り、電源を再投入してください。

6.2.4.4. ダウンローダ Hermit-At で書き換える 手順まとめ

1. SUZAKU JP1 にジャンパプラグをさしてショートさせる
2. LED/SW CON1 にシリアルケーブルを接続する
3. シリアル通信ソフトウェアを立ち上げる
4. LED/SW CON6 に AC アダプタ 5V を接続し、電源投入
5. SUZAKU D3 のパワー ON LED(緑)が点灯していることを確認
6. シリアル通信ソフトウェアの画面を確認し、"a"もしくは"Enter"キーを入力
7. ブートローダ Hermit モードになったのを確認し、シリアルポートを切断
8. ダウンローダ Hermit を起動し、イメージファイルを書き込む

※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。

BBoot が起動しない SUZAKU を iMPACT とダウンローダ Hermit-At で復帰させる

BBoot が起動しなくなった SUZAKU で再び BBoot を起動させるためには、DirectSPI や SPI Writer で FPGA リージョンを書き換えることになります。パラレルポートがある PC ならばいいですが、DirectSPI は少々面倒です。これらの方法の他に iMPACT とダウンローダ Hermit-At で書き換えるという手もあります。(DirectSPI より速いです。)手順は以下の通りです。

1. JP2 をショートし、iMPACT で FPGA に BBoot(bit ファイル)をコンフィギュレーション

2. BBoot が起動するので、"a"キーを押し、ブートローダ Hermit-At を起動させる。
3. ダウンローダ Hermit-At でフラッシュメモリの FPGA リージョンに BBoot(bin ファイル)を書き込む

6.3. ブートローダ Hermit-At の書き換えかた

ブートローダ Hermit-At を書き換えるには BBoot で書き換える方法と、ダウンローダ Hermit-At で書き換える方法の 2 通りがあります。ここでは BBoot で書き換える方法を説明します。ダウンローダ Hermit-At の使い方については「6.4.1. ダウンローダ Hermit-At で書き換える」を参考にしてください。

6.3.1. BBoot で書き換える

BBoot でブートローダ Hermit-At を書き換える方法を説明します。この際、Hermit-At のイメージデータはモトローラ S 形式のものを使用します。

6.3.1.1. 準備から書き込み

まず、JP1 にジャンパプラグをさし、ショートさせてください。JP1 をショートさせるとブートローダモードになります。

SUZAKU CON1 に方向に気をつけてシリアルケーブルを接続し、シリアル通信ソフトウェアを起動してください。(「5.1. シリアル通信ソフトウェアの起動」参照)。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。

シリアル通信ソフトウェアの画面に以下のようなメッセージが表示されます。

SUZAKU スターターキット以外の場合は、電源投入直後に"z"キーを長押ししてください。何も立ち上がらない場合は、FPGA リージョンに BBoot が書き込まれていない可能性があります。「6.2. FPGA の書き換えかた」を参照して BBoot を含んだ FPGA を書き込んでください。

```
Please choose one of the following and hit enter.
a: active second stage bootloader (default)
s: download a s-record file
t: busy loop type slot-machine
```

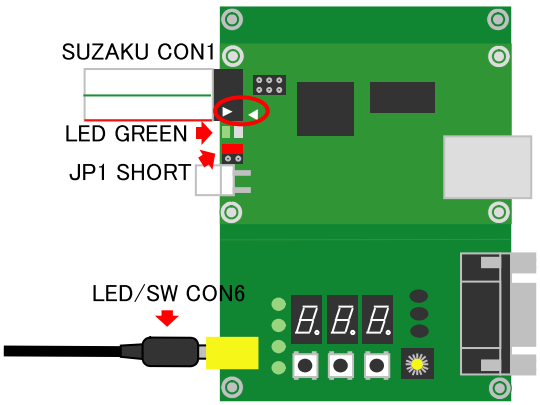


図 6.51 モトローラ S 形式書き換え準備

"s"キーを押してください。モトローラ S 形式ダウンロードモードになります。以下のようなメッセージが表示されます。

```
Start sending S-Record!!
```

書き込むモトローラ S 形式のファイルを選択してください。書き込むファイルは **srec ファイル**です。ブートローダ Hermit のファイルは付属 CD-ROM[9]に収録しています。

図 6.52 srec ファイルを送る

書き込みが始まると以下のような画面になります。

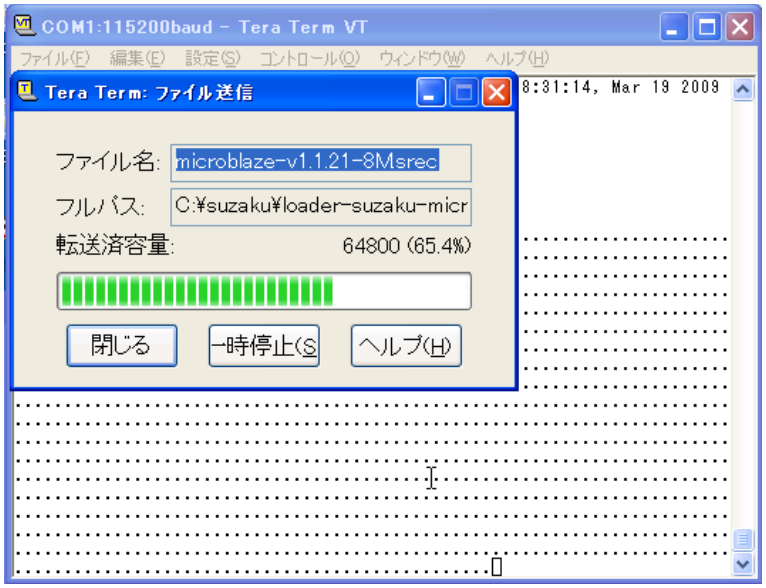


図 6.53 srec ファイル書き込み中

[9]SZ130 は"\suzaku\bootloader\s-record\loader-suzaku-microblaze-vx.x.x-8M.srec"、SZ410 は"\suzaku\bootloader\s-record\loader-suzaku-powerpc-vx.x.x-8M.srec"に収録しています。

以下のように表示されたら書き込み完了です。今書き込んだブートローダ Hermit-At が起動します。

```
Erasing SPI...
Programming SPI...
done.
Original checksum: 00XXXXXX
 Program checksum: 00XXXXXX
Reboot.

Please choose one of the following and hit enter.
a: active second stage bootloader (default)
s: download a s-record file
t: busy loop type slot-machine
```

6.3.1.2. BBoot で書き換える 手順まとめ

1. SUZAKU JP1 にジャンパプラグをさしてショートさせる
2. LED/SW CON1 にシリアルケーブルを接続する
3. シリアル通信ソフトウェアを立ち上げる
4. LED/SW CON6 に AC アダプタ 5V を接続し、電源投入
5. SUZAKU D3 のパワー ON LED(緑)が点灯していることを確認
6. シリアル通信ソフトウェアの画面を確認し、"s"と入力し、モトローラ S 形式ダウンロードモードにする
7. srec ファイルを選択し、書き込む

6.4. Linux の書き換えかた

Linux のイメージを書き換えるにはダウンローダ Hermit-At で書き換える方法、NetFlash で書き換える方法の 2 通りがあります。ここではダウンローダ Hermit-At で書き換える方法について説明します。NetFlash の使い方については「SUZAKU ソフトウェアマニュアル」をご参照ください。

6.4.1. ダウンローダ Hermit-At で書き換える

ダウンローダ Hermit-At で Linux のイメージを書き換える方法を説明します。

6.4.1.1. 書き込み準備

まず、JP1 にジャンパプラグをさし、ショートさせてください。JP1 をショートさせるとブートローダモードになります。

SUZAKU CON1 に方向に気をつけてシリアルケーブルを接続し、シリアル通信ソフトウェアを起動してください。(「5.1. シリアル通信ソフトウェアの起動」参照)。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。

シリアル通信ソフトウェアの画面に以下のようなメッセージが表示されます。

```
Please choose one of the following and hit enter.
a: active second stage bootloader (default)
s: download a s-record file
t: busy loop type slot-machine
```

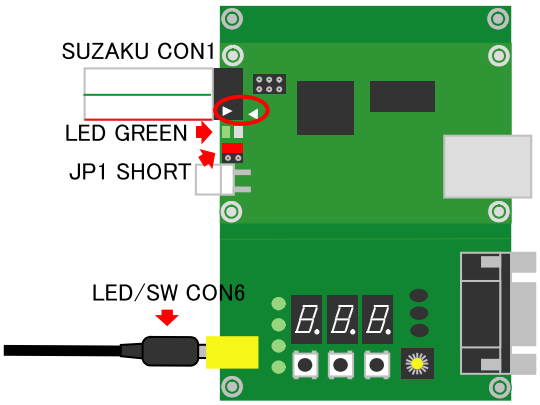


図 6.54 Linux 書き換え準備

"a"もしくは"Enter"キーを押してください。ブートローダ Hermit-At が立ち上がります。以下のようなメッセージが表示されます。もし立ち上がらない場合は「6.3. ブートローダ Hermit-At の書き換えかた」を参照してブートローダ Hermit-At を書き込んでください。

```
Hermit-At v1.1.15 (suzaku/microblaze) compiled at 19:28:48, Feb 16 2008
hermit>
```

ブートローダ Hermit-At が立ち上がったのを確認したら、シリアルポートをシリアル通信ソフトから切断します。[ファイル]→[接続断]を選択してください。

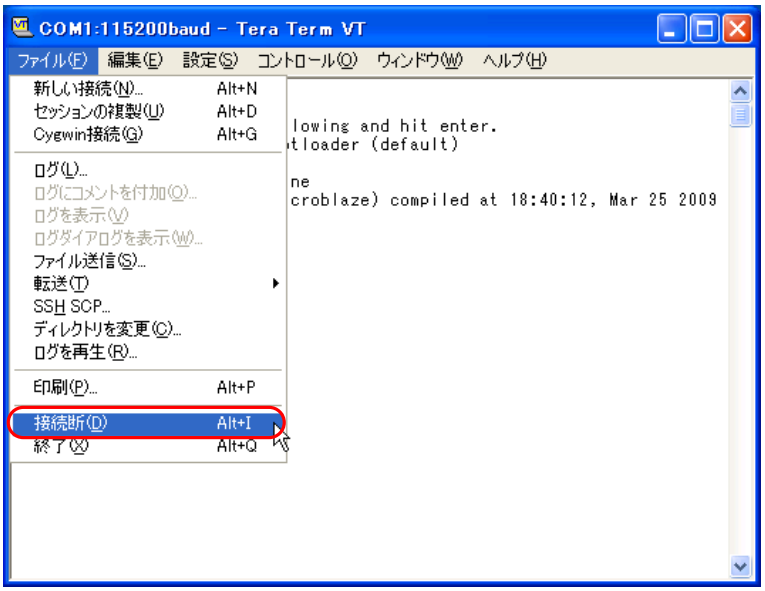


図 6.55 シリアルポートを切断

6.4.1.2. ダウンローダ Hermit-At 立ち上げから書き込み

ダウンローダ Hermit-At を起動してください。[Download]ボタンをクリックすると Download 画面が表示されます。[Serial Port]に SUZAKU と接続しているシリアルポートを設定し、[Image]に書き込むイメージファイルを指定してください。ダウンローダ Hermit-At では **bin ファイル**を書き込みます。

SUZAKU のデフォルトの bin ファイルとスロットマシンの bin ファイル(スターターキット出荷時の bin ファイル)は付属 CD-ROM[10]に収録しています。

[Region]には、書き込むリージョンまたは、アドレスを指定します。Linux はイメージリージョンに書き込みます。[image]を選択してください。

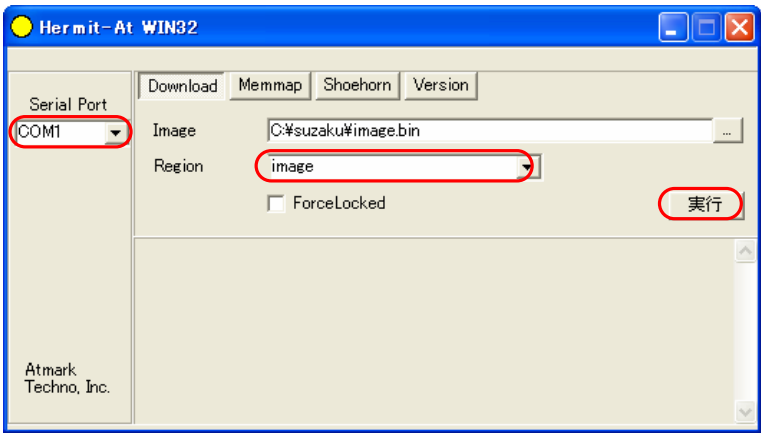


図 6.56 Download 画面

[実行]をクリックしてください。フラッシュメモリへの書き込みが始まります。書き込み中は、進捗状況が表示されます。

[10]SUZAKU のデフォルトの bin ファイルは"\suzaku\image\image-sz***.bin"にスロットマシンの bin ファイル(スターターキット出荷時の bin ファイル)は"\suzaku-starter-kit\image\image-sz***-sil.bin"に収録しています。

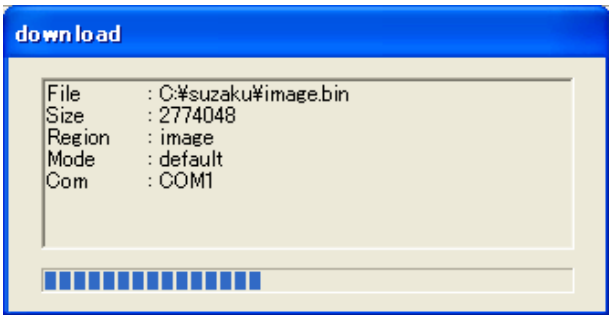


図 6.57 書き込み進捗ダイアログ

書き込みが終了すると、書き込み終了画面が表示されます。"Download COMPLETE"と表示されたら書き込み成功です。

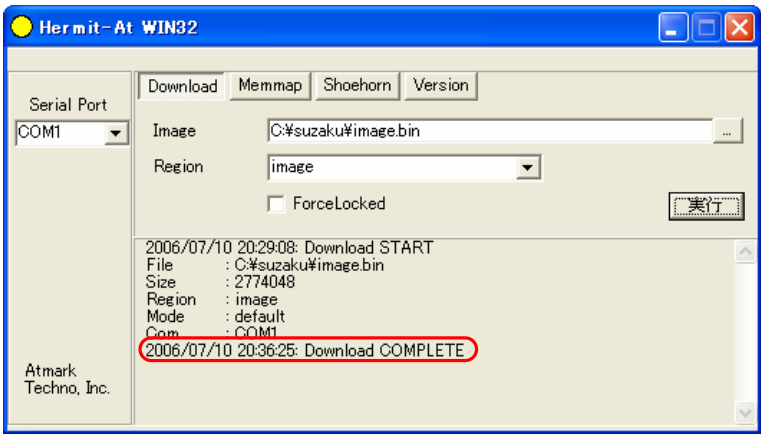


図 6.58 書き込み終了

Hermit-At を書き換える

ダウンローダ Hermit-At ではブートローダ Hermit-At も書き換えることが出来ます。bin ファイルは Linux の bin ファイルと同じフォルダに収録しています。[11]

[Region]に[bootloader]を選択し、[ForceLocked]をチェックして書き込んでください。

6.4.1.3. ダウンローダ Hermit-At で書き換える 手順まとめ

1. SUZAKU JP1 にジャンパプラグをさしてショートさせる
2. LED/SW CON1 にシリアルケーブルを接続する
3. シリアル通信ソフトウェアを立ち上げる

[11] SZ130 用 は "loader-suzaku-microblaze-vx.x.x.bin" 、 SZ410 用 は "loader-suzaku-powerpc-vx.x.x.bin"です。

4. LED/SW CON6 に AC アダプタ 5V を接続し、電源投入
5. SUZAKU D3 のパワー ON LED(緑)が点灯していることを確認
6. シリアル通信ソフトウェアの画面を確認し、"a"もしくは"Enter"キーを入力
7. ブートローダ Hermit モードになったのを確認し、シリアルポートを切断
8. ダウンローダ Hermit を起動し、イメージファイルを書き込む

7.ISE の使い方

SUZAKU の開発をするために、まずは ISE(Integrated Software Environment)が必要になります。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 の使い方を説明します。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ので、困った時にはそちらも参照してください。

ISE において以下の手順で作業を行います。ISE で作業をする場合、タイトルと作業の名前をそろえているので、どこの作業を行っているかはタイトルを目安にしてください。

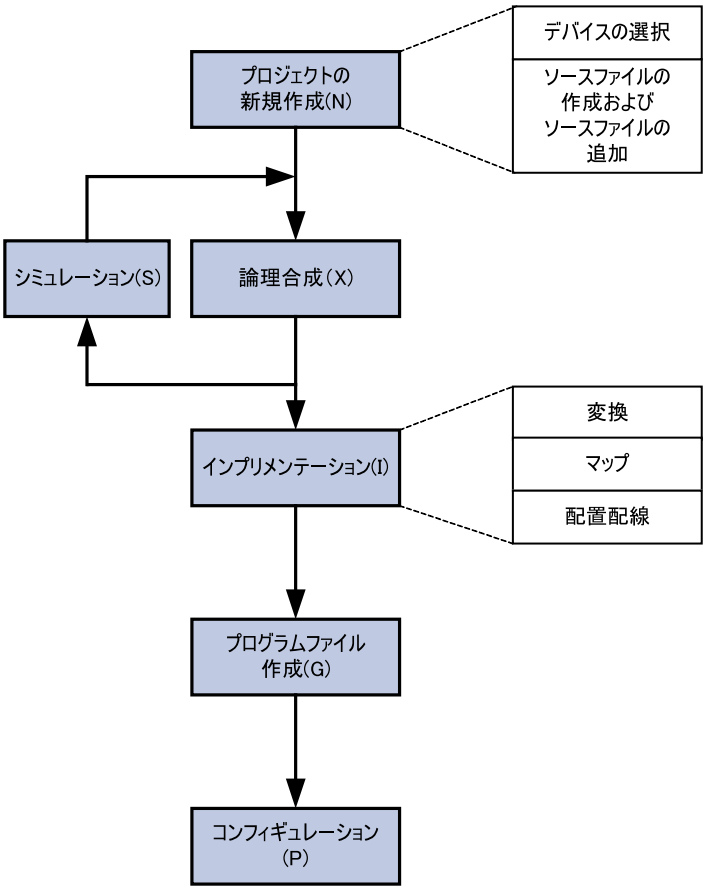


図 7.1 本書での ISE 開発フロー

7.1. 単色 LED を点灯させる

ISE を使って、LED/SW ボードに実装されている単色 LED(D1)を点灯させます。

7.1.1. 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。単色 LED は 180Ω の抵抗で 3.3V にプルアップされています。FPGA から Low を出力すると点灯し、High を出力すると消灯します。

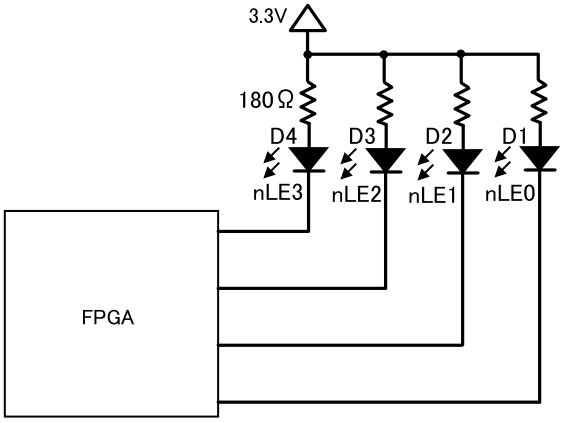


図 7.2 単色 LED 周辺回路

FPGA の入出力について

SUZAKU の FPGA の I/O ピンは CMOS+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。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 7.1 FPGA の入力と出力

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

7.2. プロジェクトの新規作成

ISE でプロジェクトを新規作成します。

7.2.1. プロジェクト作成

Project Navigator を起動してください。Project Navigator は[スタートメニュー]→[すべてのプログラム]→[Xilinx ISE Design Suite x.x]→[ISE]→[Project Navigator]から起動できます。

プロジェクトを作成します。[File]→[New Project]をクリックしてください。

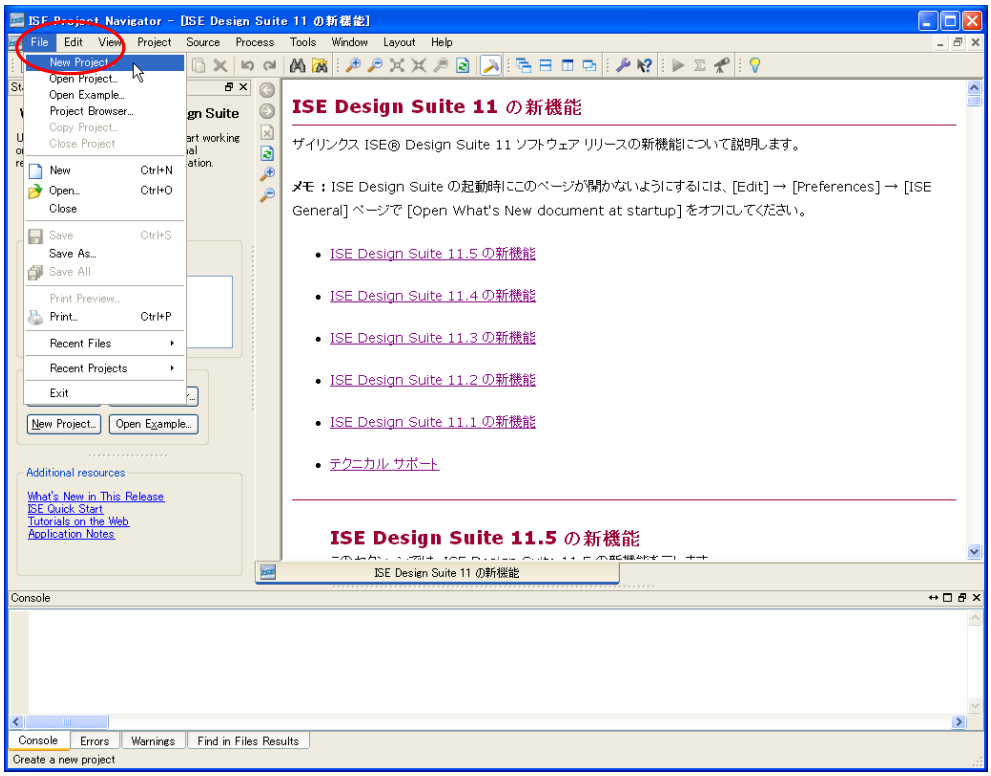


図 7.3 Project Navigator 起動

New Project Wizard が表示されます。[Name:]にプロジェクト名を入力します。ここでは[slot_le]とします。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。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。

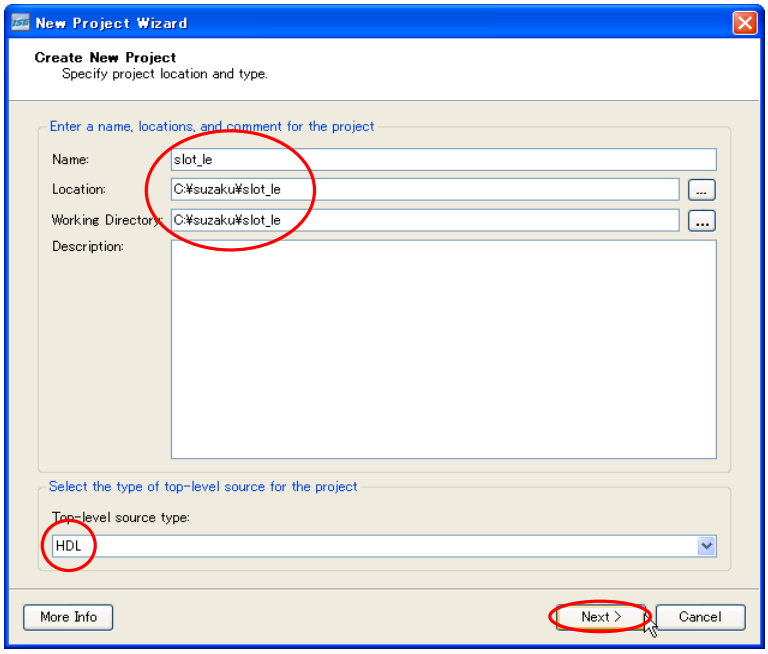


図 7.4 プロジェクトの新規作成

7.2.2. 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	SZ130	SZ410
Product Category	All	
Family	Spartan3E	Virtex4
Device	XC3S1200E	XC4VFX12
Package	FG320	SF363
Speed	-4	-10
Synthesis Tool	XST(VHDL/Verilog)	
Simulator	ISim(VHDL/Verilog)	
Preferred Language	VHDL	

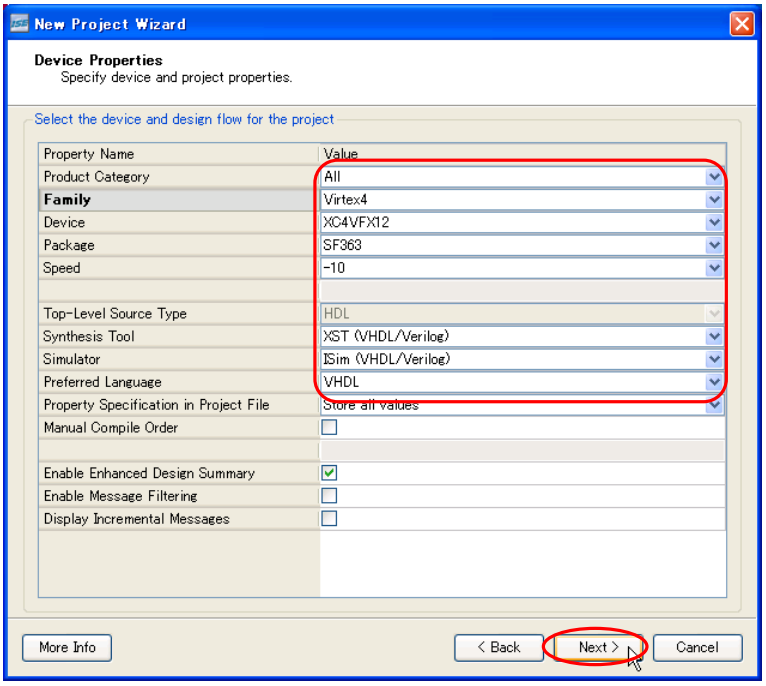


図 7.5 デバイスの選択(SZ410 の場合)

7.2.3. ソースファイル作成

新しくソースファイルを作成します。[New Source]をクリックしてください。

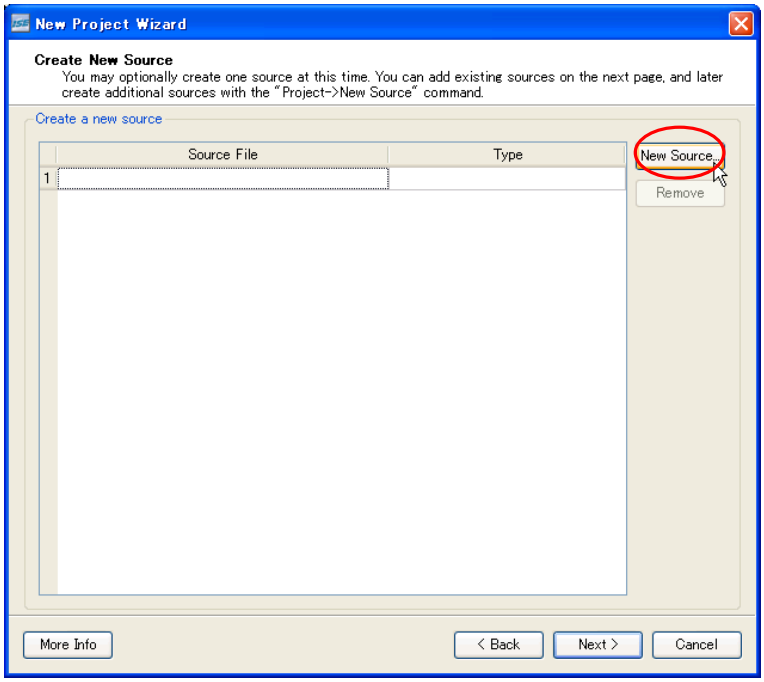


図 7.6 New Source 作成

[VHDL Module]を選択し、[File name:]にファイルの名前を入力します。ここでは[top]とします。[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。

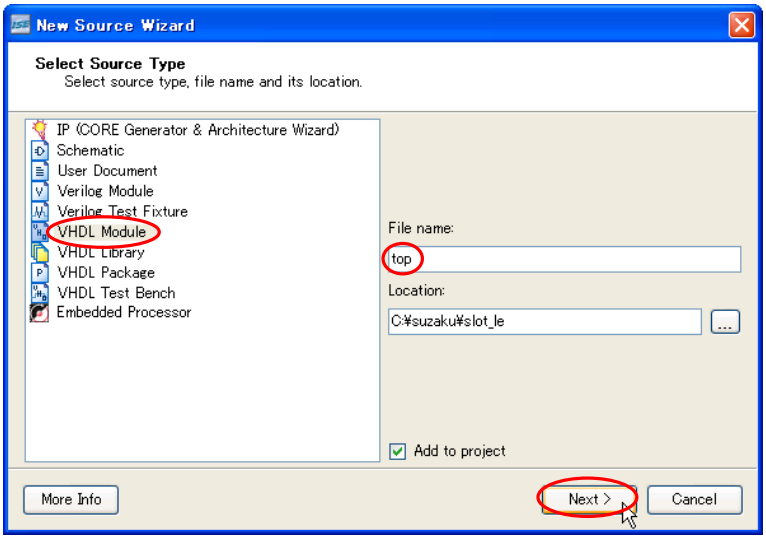


図 7.7 VHDL ソースファイル作成

[Architecture Name]を入力します。SUZAKU では[imp](implement の意味)としています。

LED を 1 つ点灯させるため、出力ピンを設定します。[Port Name]にピンの名前を入力します。ここでは[nLE0]とします。[Direction]は出力なので[out]にします。変更したら[Next]をクリックしてください。

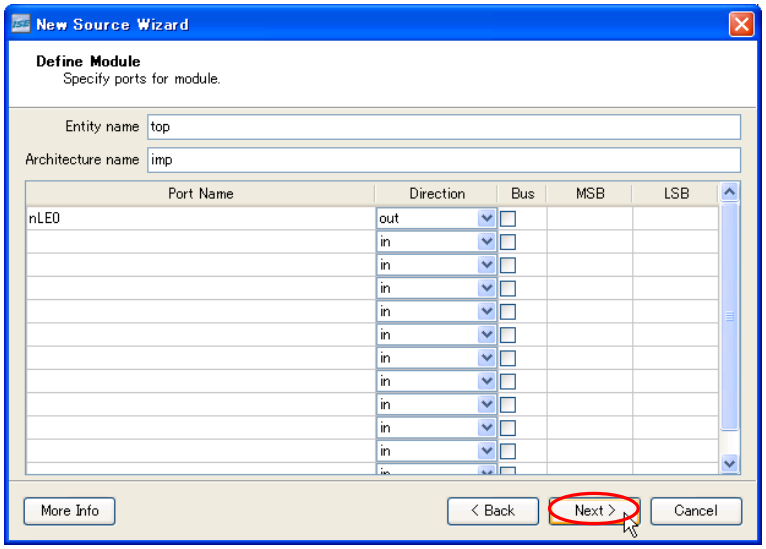


図 7.8 アーキテクチャ名定義

今作った VHDL ソースファイルの設定が表示されます。設定に間違いがないか確認をし、[Finish]をクリックしてください。

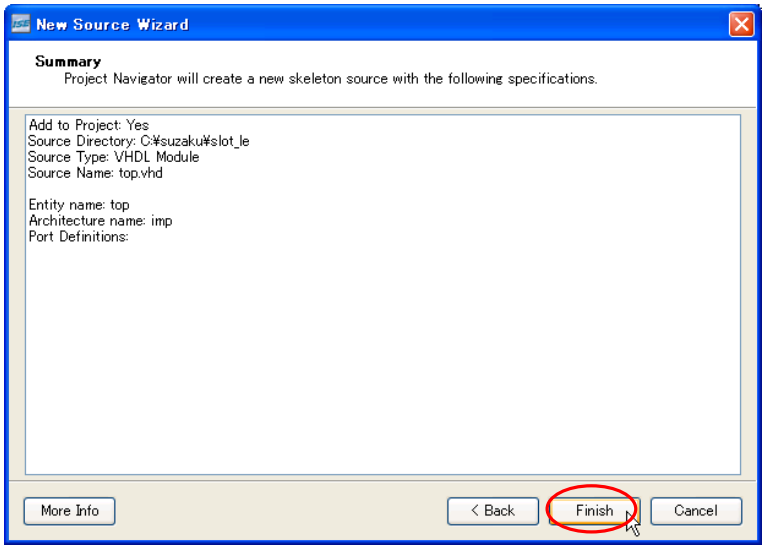


図 7.9 ソースファイル作成確認画面

以下の画面が出るまで[Next]をクリックしてください。内容を確認し、[Finish]をクリックしてください。

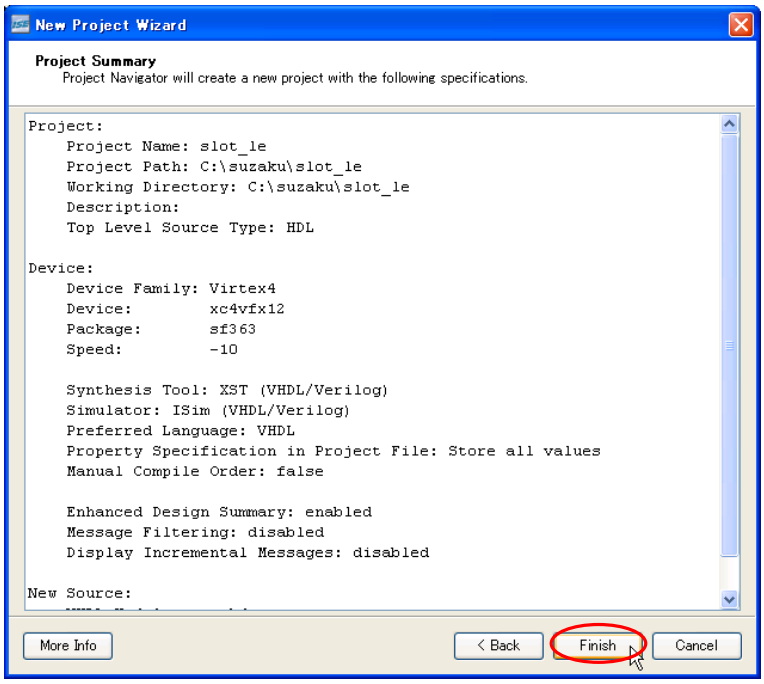


図 7.10 最終確認画面(SZ410 の場合)

以上でプロジェクトおよび VHDL ソースファイルのテンプレートが出来上がります。

7.2.4. ソースファイル編集

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。

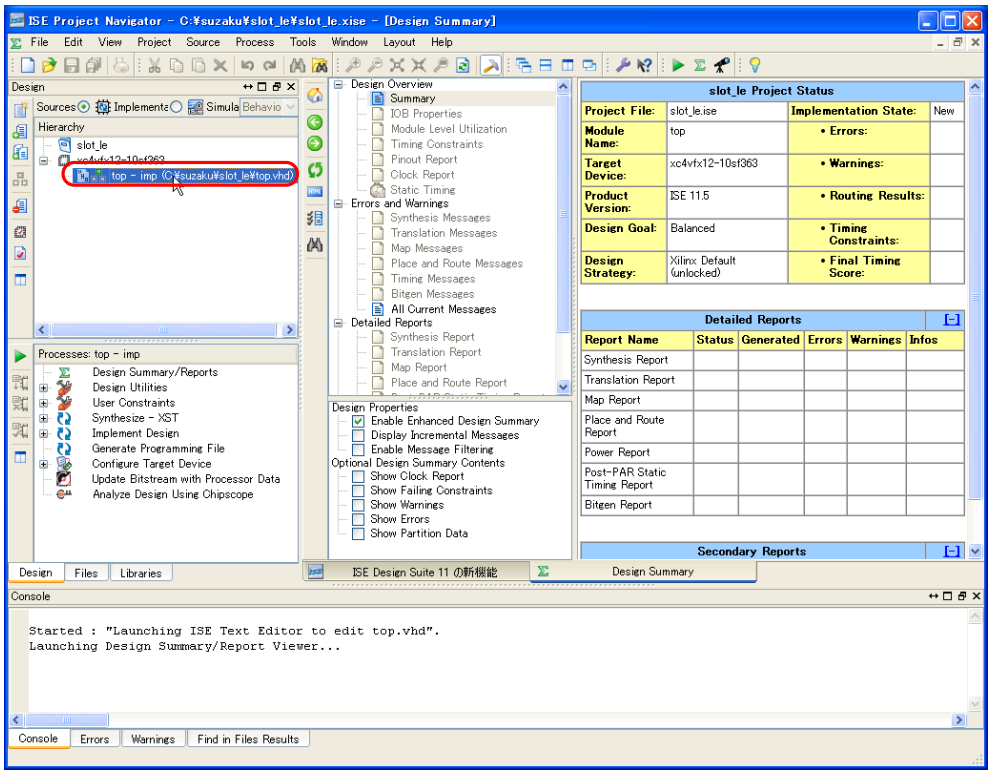
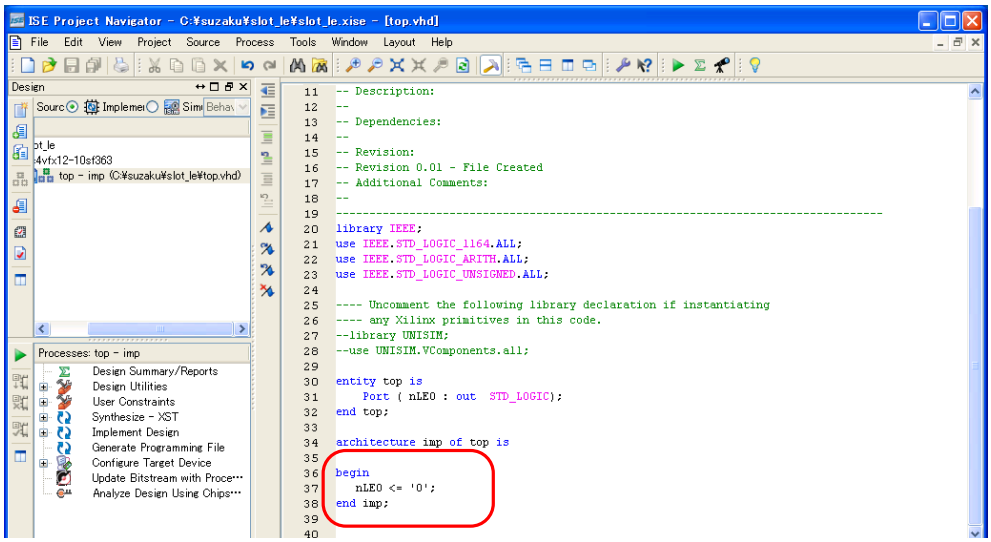


図 7.11 top.vhd を開く

テンプレートが自動生成されています。begin と end imp;の間にに単色 LED を点灯させる文 nLE0 <= '0';を追加します。太字の部分を参考に追加してください。




```
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 ( nLE0 : out  STD_LOGIC);
  end top;

  architecture imp of top is
  begin

    nLE0 <= '0';

  end imp;
```

図 7.12 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。

7.3. 論理合成

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。

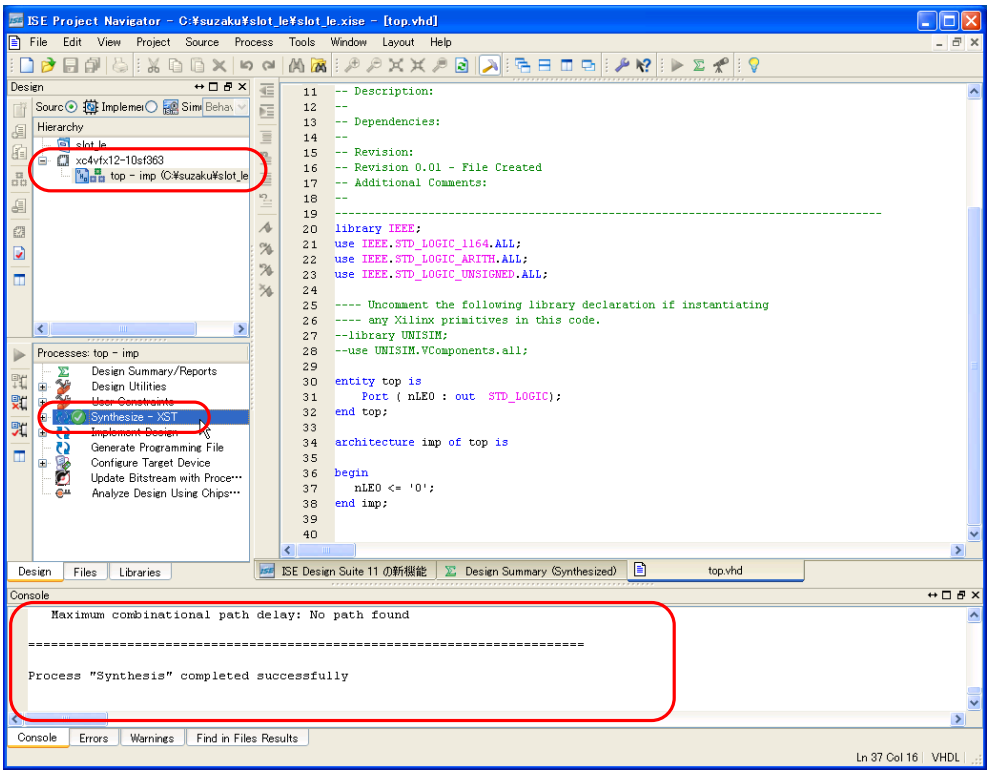


図 7.13 文法チェック

7.4. インプリメンテーション

User Constraints の横の⊞をクリックして開き、[Floorplan Area/IO/Logic(PlanAhead)]をダブルクリックしてください。その際に 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。

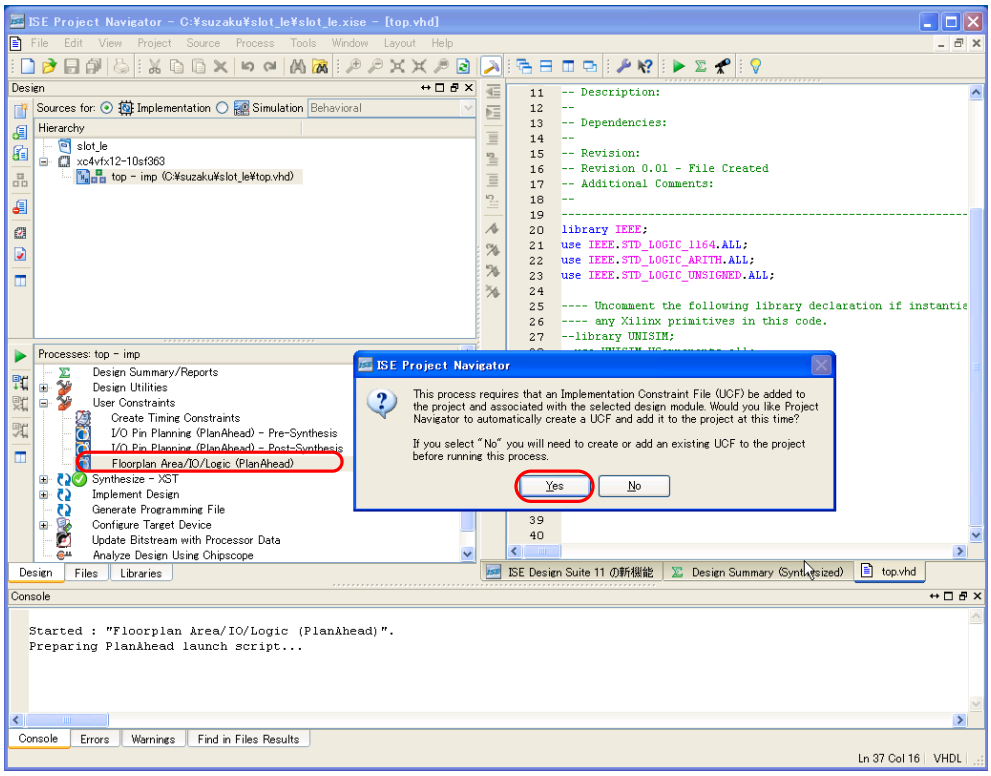


図 7.14 PlanAhead を立ち上げる

PlanAhead 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。

I/O Ports の[All ports]→[Scalar ports]→[nLE0]を右クリックしてメニューを出し、[Configure I/O Ports]を選択してください。Configure Ports ウィンドウが立ち上がるので、I/O Standard:を[LVCMOS33]にし、[OK]をクリックして下さい。

「表 2.2. 機能用ピンアサイン(CON2)」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。戻るのが面倒だという人のために以下にもピンアサインを記載します。[I/O Port Properties]の[Sute:]にピンアサインを入力し、[Apply]をクリックして下さい。

表 7.2 nLE0 ピンアサイン

	SZ130	SZ410
nLE0	E12	G2

[File]→[Save Project...]をクリックして保存し、PlanAhead を閉じて下さい。

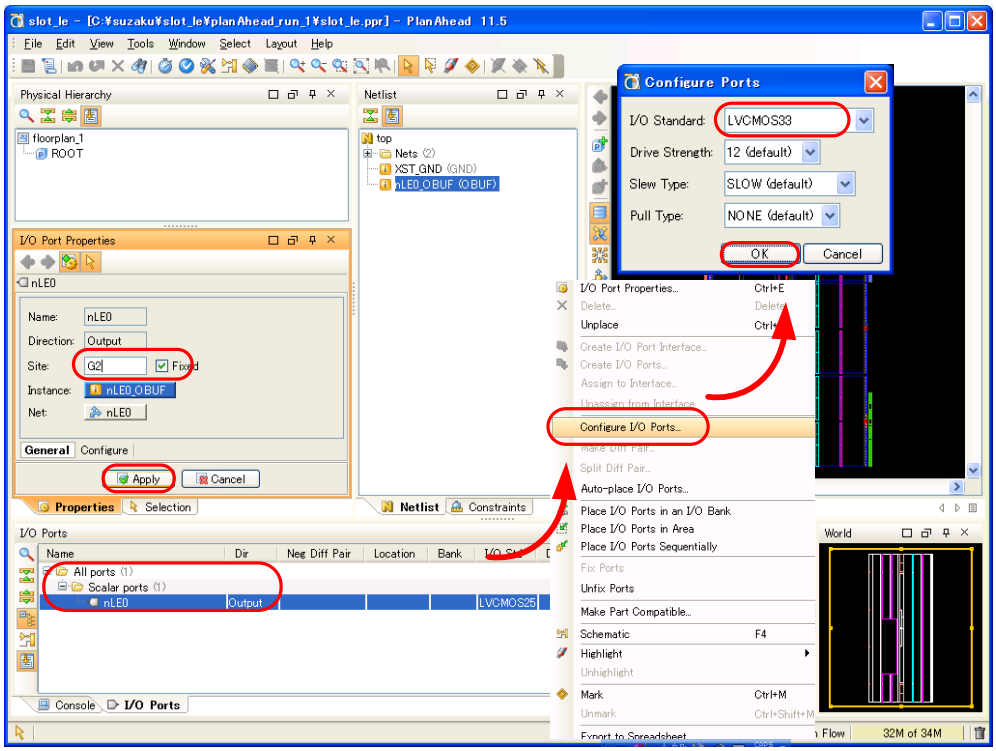


図 7.15 PlanAhead によるピンアサイン(SZ410 の場合)

I/O ピンのカスタマイズ

最近の FPGA は I/O のカスタマイズも自由に行うことが出来ます。数 10kΩ の Pull Up、Pull Down をつけたり、電流制限をつけたり(オーバーシュートやアンダーシュートの設定)、スルーレートの High、Low の設定、I/O の属性の変更(SUZAKU では I/O 電圧が 3.3V のため、属性は C-MOS のみ)等が出来ます。

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。`top.ucf` をクリックし(ダブルクリックするとPlanAhead が立ちあがってしまいます。)、Processes のウィンドウの[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。

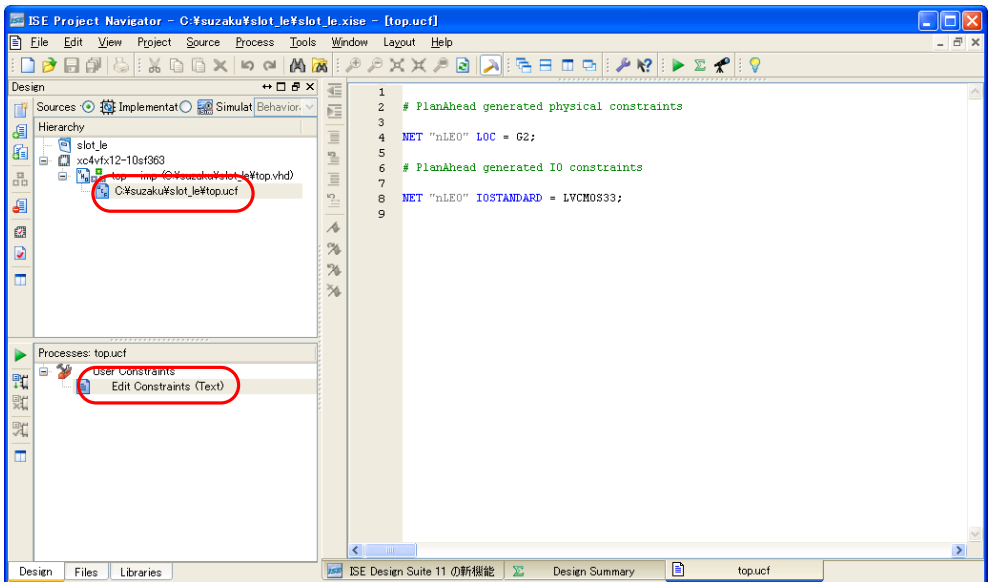


図 7.16 ピンアサインファイルの確認(SZ410 の場合)

top-imp(top.vhd)をクリックし、[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。

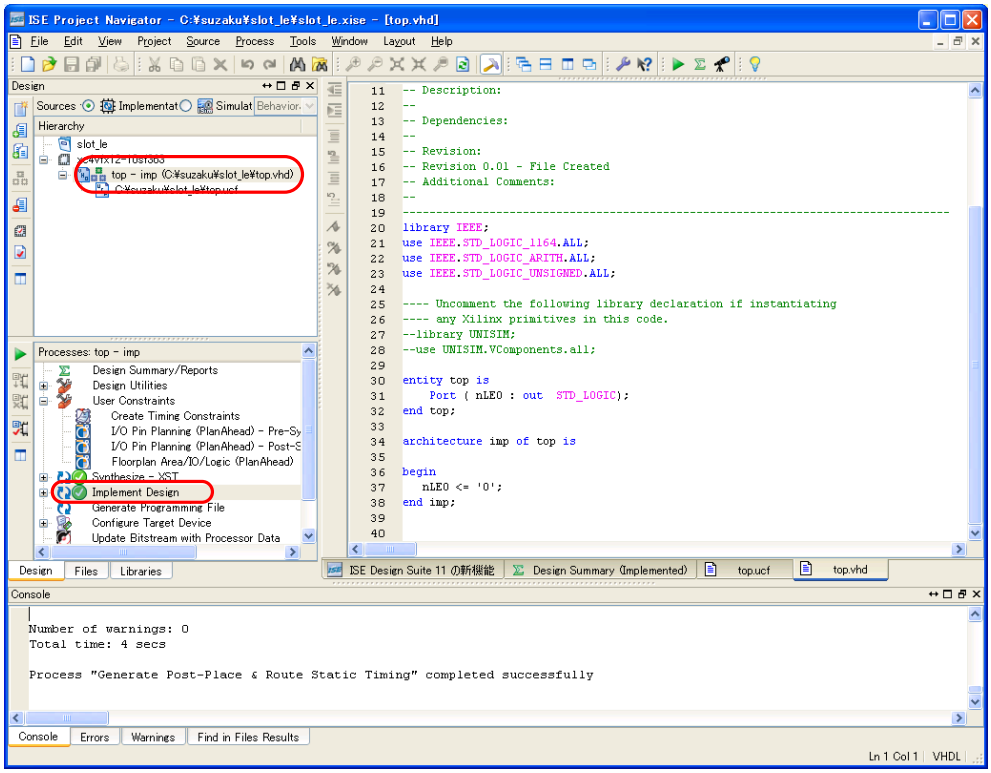


図 7.17 インプリメント

7.5. プログラムファイル作成

[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 ファイルが生成されます。

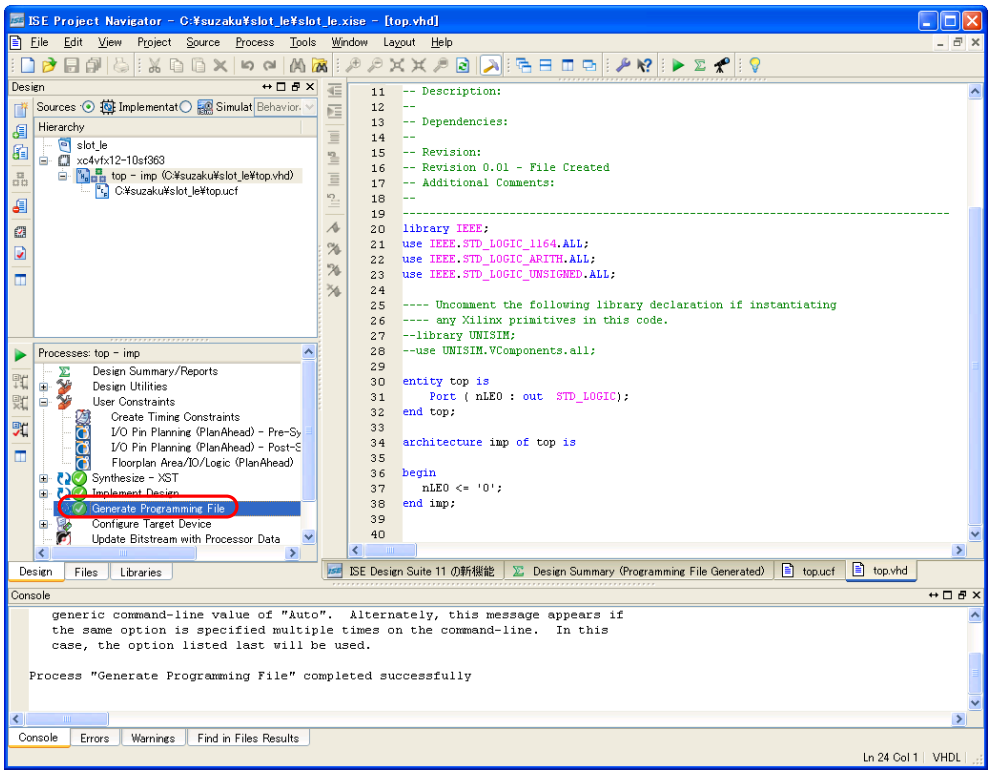


図 7.18 bit ファイル作成

7.6. コンフィギュレーション

FPGA に作成した top.bit を書き込みます。

7.6.1. JTAG でコンフィギュレーション

iMPACT で top.bit を書き込みます。bit ファイルは"C:\suzaku\slot_le\top.bit"にあります。iMPACT での FPGA の書き換え方については「6.2.1. iMPACT で書き換える」をご参照ください。iMPACT は[Manage Configuration Project(iMPACT)]をダブルクリックすることでも起動することが出来ます。

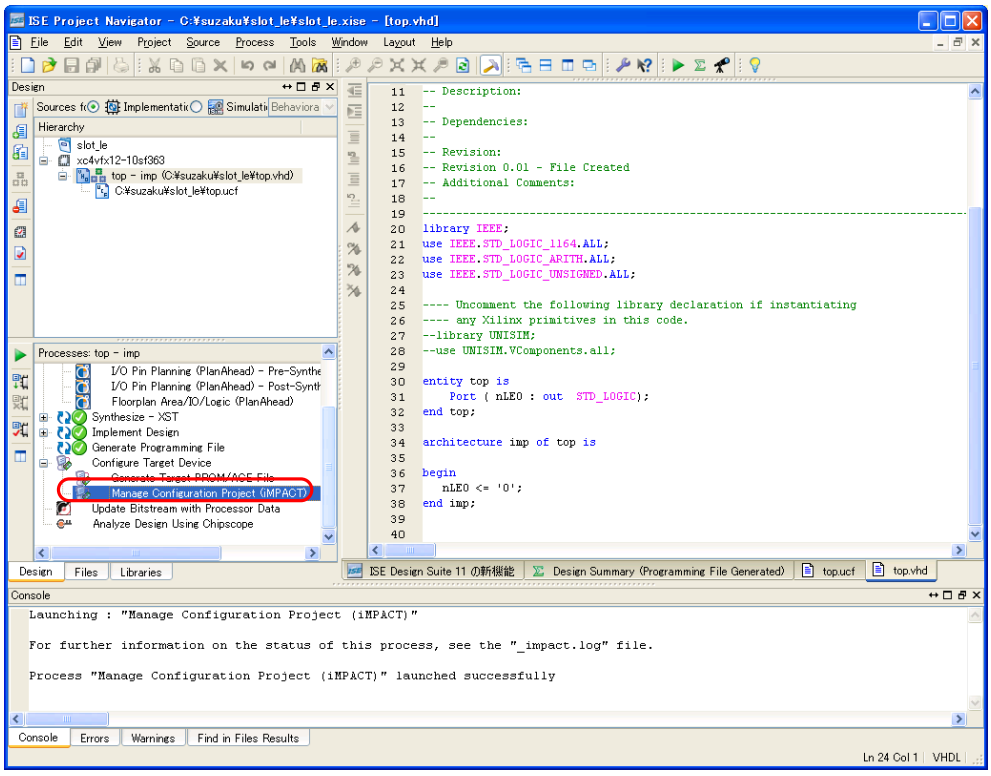


図 7.19 iMPACT 立ち上げ

単色 LED(D1)が光ったでしょうか？

どうしてもうまくいかない場合は付属 CD-ROM に vhdl ファイルと ucf ファイルを収録[1]しているので、比較してみてください。

[1]"\suzaku-starter-kit\fpga\slot_le.zip"に収録しています。

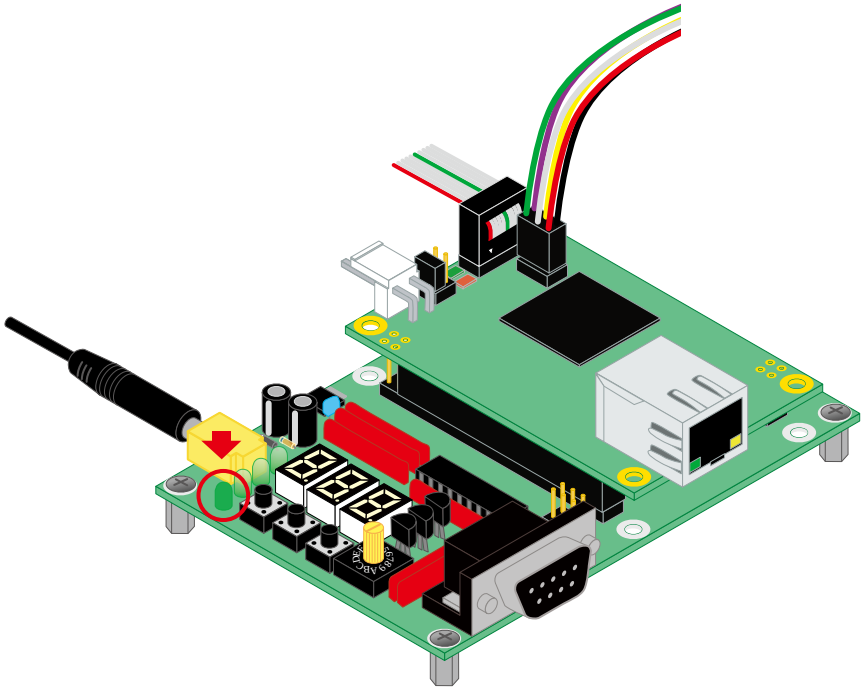


図 7.20 単色 LED(D1)点灯

7.7. 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？(SZ410 だとほとんど光りません)

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、[Process Properties]を選択してください。ウィンドウが立ち上がるので、[Configuration Option]を選択してください。ここで空きピンの終端処理を設定できます。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。このため下図のように D2、D3、D4 に電流が少し流れて LED が光ってしまいます。

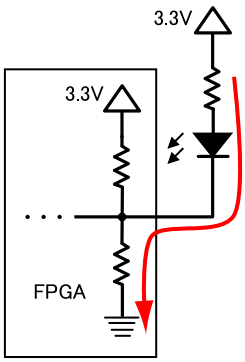


図 7.21 少し光る理由

設定を[Pull Up]にすると、D2、D3、D4 は光らなくなりますが、ここの設定では空きピンの終端処理を個別に設定することが出来ないため、SUZAKU が動かなくなってしまう可能性があります。SUZAKU には RESET 回路があり、RESET 回路につながっているピンは外部で Pull Down されています。このピンの終端処理は Low かハイインピーダンスにしておかなければならず、High や Pull Up、Float に変更

した場合、リセットがかかってしまうことがあります。また、空きピンから電圧が出力されているのはあまり良い状態ではありません。よって今回は D2、D3、D4 に信号を定義することで、この問題に対処します。top.vhd、top.ucf に D2、D3、D4 の信号の記述を加えてください。

例 7.1 信号の記述を追記(top.vhd)

```
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 (
    nLE0 : out  STD_LOGIC;
    nLE1 : out  STD_LOGIC;
    nLE2 : out  STD_LOGIC;
    nLE3 : out  STD_LOGIC

  );

end top;

architecture IMP of top is
begin

  nLE0 <= '0';
  nLE1 <= '1';
  nLE2 <= '1';
  nLE3 <= '1';

end imp;
```

表 7.3 ピンアサイン

	SZ130	SZ410
nLE1	F12	F2
nLE2	B11	F1
nLE3	A11	E1

8.VHDL によるロジック設計

本書を読むために必要となる最低限の VHDL の記述方法とロジック設計について説明します。VHDL の詳細やロジック設計については、世の中に詳しい書物が多数ありますのでそちらをご参照ください。

8.1. VHDL の基本構造

まず 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大文字と小文字を区別しません。例えば `Port` は `port` と記述しても `PORT` と記述しても同じに扱われます。予約語については「Tips VHDL 予約語」をご参照ください。もし、ISE 付属のテキストエディタを使っている場合、予約語は色が変わって表示されます。また、`--`で始めるとその行末までがコメントになります。

例 8.1 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;
```

VHDL 予約語

```
abs, access, after, alias, all, and ,architecture,
array, assert, attribute, begin, block, body, buffer,
bus, case, component, configuration, constant,
disconnect, downto, else, elsif, end, entity, exit,
file, for, function, generate, generic, guarded, if,
impure, in, inertial, inout, is, label, library,
linkage, literal, loop, map, mod, nand, new, next, nor,
not, null, of, on, open, or, others, out, package, port,
postponed, process, pure, range, record, register,
reject, rem, report, return, rol, ror, select,
severity, shared, signal, sla, sll, sra, srl, subtype,
then, to, transport, type, unaffected, units, until,
use, variable, wait, when, while, with, xnor, xor
```

8.2. 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。ライブラリは自分で作成することも出来ます。

表 8.1 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

8.3. 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 8.2 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;
```

8.3.1. 信号の定義

信号は以下の形式で宣言します。

例 8.3 信号の定義

```
ポート信号名：入出力方向 データタイプ名;
```

8.3.2. 入出力方向

入出力方向には in、out、inout 等を記述します。

表 8.2 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「8.4.1. 内部信号の定義」を参照してください。

8.3.3. 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 側がビット 0 になります。downto とすると LSB 側がビット 0 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。CoreConnect については「10.1.7. バスのビットラベルについて」をご参照ください。

表 8.3 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 8.1 to を使って定義

8.4. 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。

例 8.4 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  ❶

  process(SYS_CLK) -- プロセス文
  begin
    if SYS_CLK'event and SYS_CLK = '1' then  ❶
      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;
```

❶ 同時に処理される

8.4.1. 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、「8.3.3. データタイプ」をご参照ください。

例 8.5 内部信号定義

```
signal 信号名 : データタイプ名;
```

8.4.2. 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。

8.4.3. 信号代入文

A<=B;とすると、A に B が代入されます。

例 8.6 信号代入文

```
A<=B;
```

8.4.4. プロセス文

プロセス文は以下の形で記述します。

例 8.7 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。

8.5. 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。

8.5.1. 押しボタンスイッチ周辺回路

組み合わせロジックの入力として、押しボタンスイッチを利用します。押しボタンスイッチは以下のような回路になっています。単色 LED の周辺回路は「図 7.2. 単色 LED 周辺回路」をご参照ください。

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。

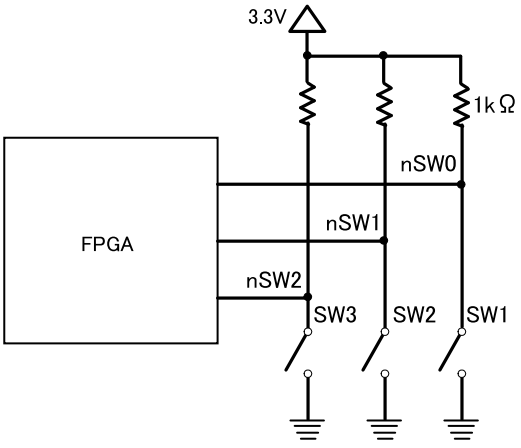


図 8.2 押しボタンスイッチ周辺回路

8.5.2. 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)

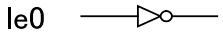
LED/SW ボードには正論理、負論理の信号が混在しているので、分かりやすくするために負論理の信号は FPGA 内部で反転させて正論理として扱うようにします。

8.5.2.1. not

負論理から正論理(正論理から負論理)は次の一文で記述できます。

例 8.8 not 記述

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



le0	nLE0
0	1
1	0

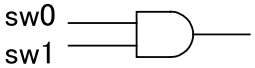
図 8.3 not 回路と真理値表

8.5.2.2. 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 8.9 and 記述

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



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

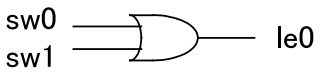
図 8.4 and 回路と真理値表

8.5.2.3. 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 8.10 or 記述

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



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

図 8.5 or 回路と真理値表

8.5.2.4. not, and, or の 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 8.11 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;
```

8.5.2.5. not, and, or のピンアサイン

ピンアサインは以下になります。

表 8.4 not、and、or のピンアサイン

	SZ130	SZ410
nLE0	E12	G2
nSW0	F11	G4
nSW1	C11	M1

8.6. 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。

8.6.1. D-FF(D 型フリップフロップ)

順序回路で重要なのは D-FF 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。

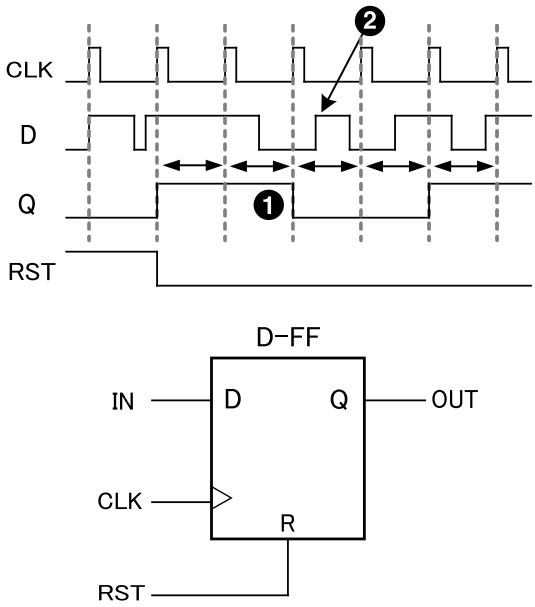


図 8.6 D-FF の動作

❶ 次のクロックまでデータ保持

❷ ここでは出力は変化しない

8.6.2. 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証されています。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証されています。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。

8.6.3. カウンタ

順序回路の基本的な例としてカウンタを上げます。カウンタ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)します。カウンタの回路は以下のように記述できます。

例 8.12 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;
```

8.6.3.1. 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 8.13 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
```

8.6.3.2. 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 8.14 同期リセット

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

8.6.3.3. if 文

if 文は以下の形式で記述します。

例 8.15 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;
```

8.6.3.4. others で初期化

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 8.16 other で初期化

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

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。

9.ISE Simulator の使い方

今まではとても簡単なロジックなので、シミュレーションを行なっていませんでしたが、実際の作業では HDL でコーディングを行ったら、PC 上でシミュレーションを行います。シミュレーションはロジック設計で重要な作業です。実際にデバイスに書き込んでからでは、各信号の挙動をとても検証しづらいですが、PC 上のシミュレーションでは、信号の挙動が明快にわかります。シミュレーションで自分の考えていた通りに動作しているか確認してから、実際のデバイスに書き込みます。

最も基本の順序回路であるカウンタの動きを確認すると共に、ISE に含まれているシミュレータ ISE Simulator の使い方を説明します。

9.1. プロジェクトの新規作成

プロジェクトを新規作成してください。ここではプロジェクト名を[slot_counter]とします。[New Source]で新規ソースファイルを作ります。ファイル名は `slot_counter.vhd` とします。プロジェクトの新規作成方法が分からない場合は、「7.2. プロジェクトの新規作成」を参照してください。

9.1.1. slot_counter.vhd

カウンタ回路を記述します。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。

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 9.1 カウンタ(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;
```

9.1.2. generic 文について

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 9.2 generic 文

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

9.2. 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。

[Simulation]をチェックし、slot_counter を右クリックしてメニューを出し、[New Source]を選択して下さい。

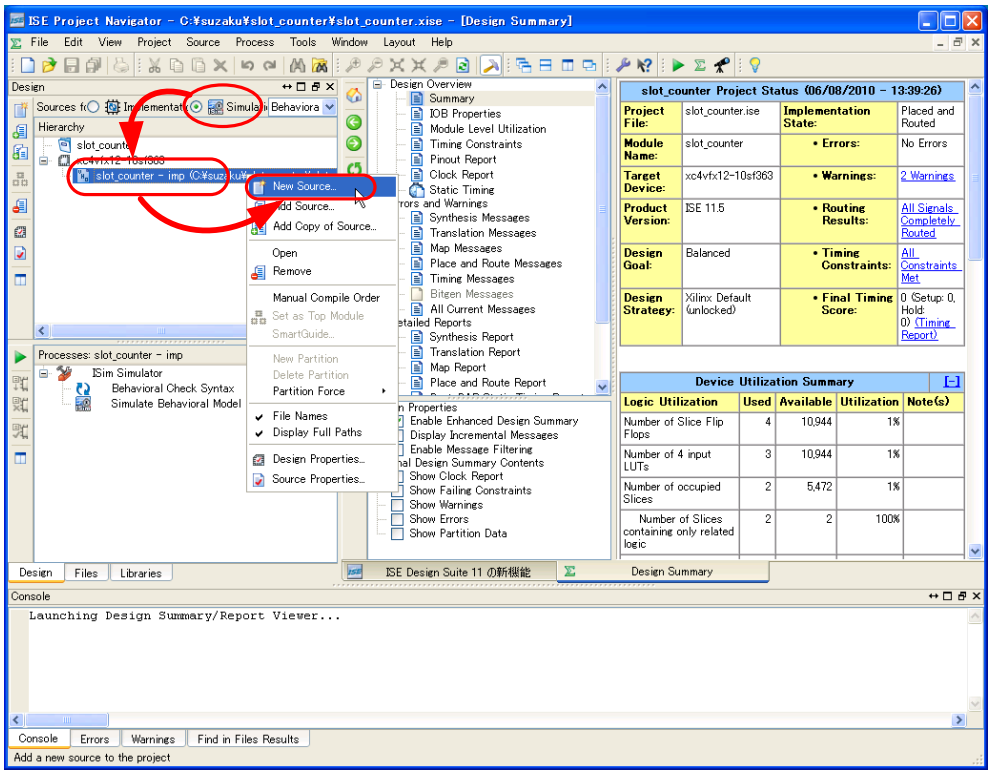


図 9.1 シミュレーション開始

[VHDL Test Bench]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。

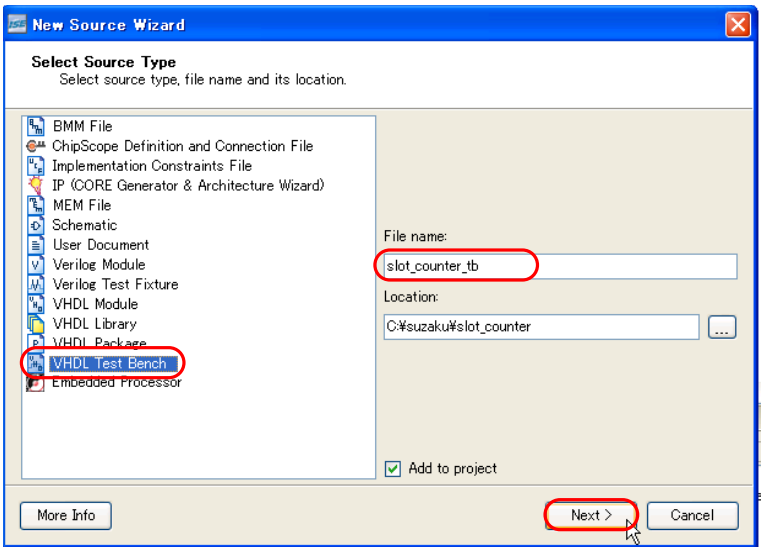


図 9.2 テストベンチ作成

次の画面が出るまで[Next]をクリックし、内容を確認し、[Finish]をクリックしてください。

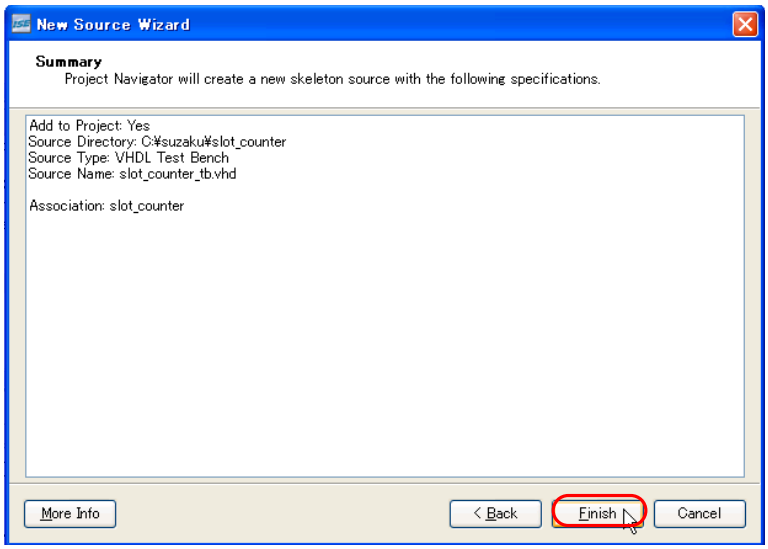
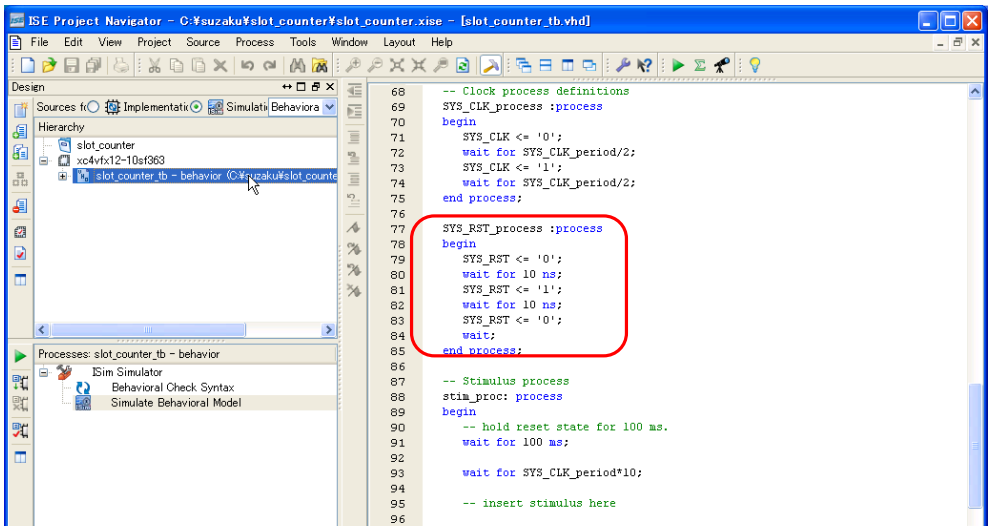


図 9.3 テストベンチ作成終了

テストベンチが出来上がります。slot_counter_tb-behavior(slot_counter_tb.vhd)をダブルクリックして下さい。`slot_counter_tb.vhd` が開きます。

テンプレートが自動生成されています。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、以下のようにリセット信号の記述を追加します。スタートして 10ns たったらリセットを立て、10ns 後下げるコードとなっています。追加できたら、[File]→[Save]をクリックし、保存してください。




```
-- 前略
-- Clock process definitions
SYS_CLK_process :process
begin
SYS_CLK <= '0';
wait for SYS_CLK_period/2;
SYS_CLK <= '1';
wait for SYS_CLK_period/2;
end process;

SYS_RST_process :process
begin
SYS_RST <= '0';
wait for 10 ns;
SYS_RST <= '1';
wait for 10 ns;
SYS_RST <= '0';
wait;
end process;

-- Stimulus process
--  後略
END;
```

図 9.4 リセット波形生成

[Behavioral Check Syntax]をダブルクリックし、文法チェックをして下さい。エラーがなければ[Simulate Behavioral Model]を右クリックしてメニューを出し、[Proccess Properties]を選択して下さい。シミュレーションの時間などが設定できます。今回は特に変更しないで、[OK]をクリックして下さい。

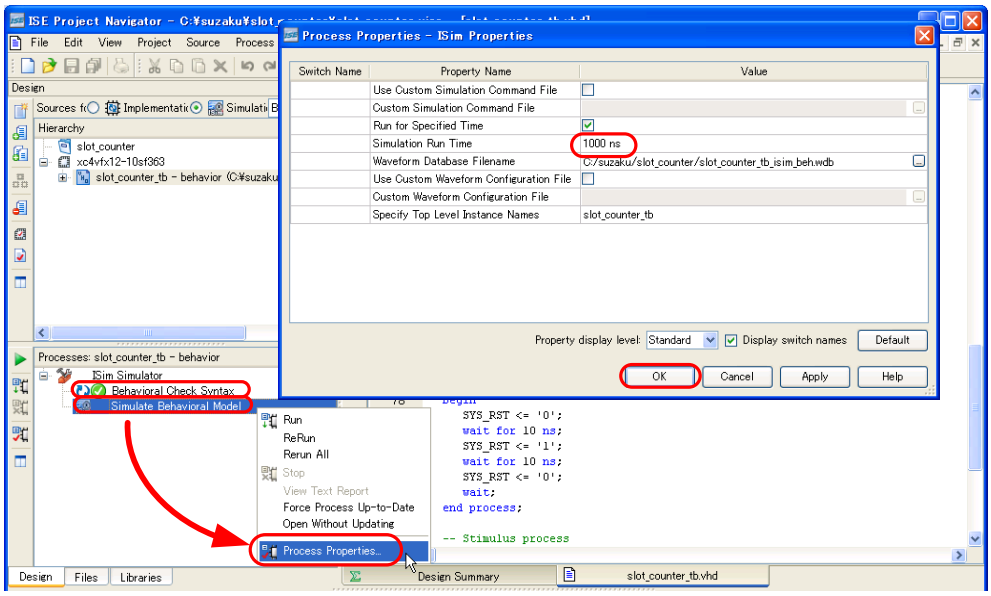


図 9.5 シミュレーション設定

9.3. シミュレーション実行

[Simulate Behavioral Model]を右クリックしてメニューを出し、[Run]をクリックして下さい。

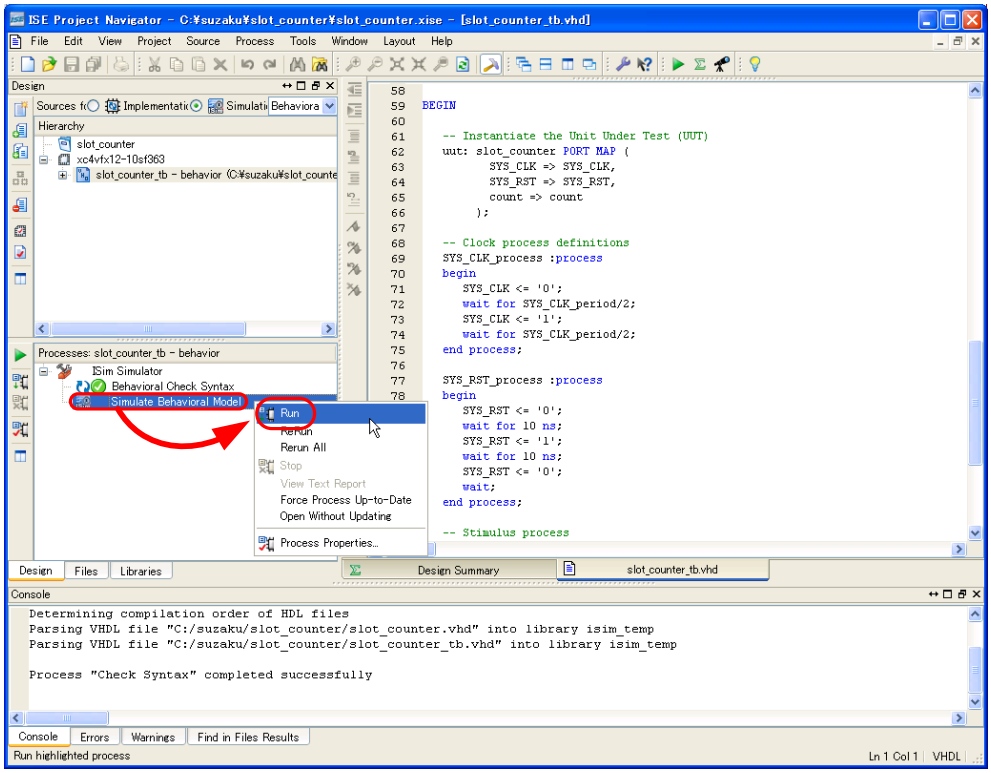


図 9.6 シミュレーション開始

ISE Simulator が立ち上がり、シミュレーション結果が表示されます。

[虫めがねマーク]などをクリックし、見やすい大きさに調整して下さい。[count]を右クリックしてメニューを出し、[Radix]を変更すると、バイナリ以外でも数値を表示することができます。その他、詳細な使い方については、ISE の Help や Xilinx の資料を参照してください。

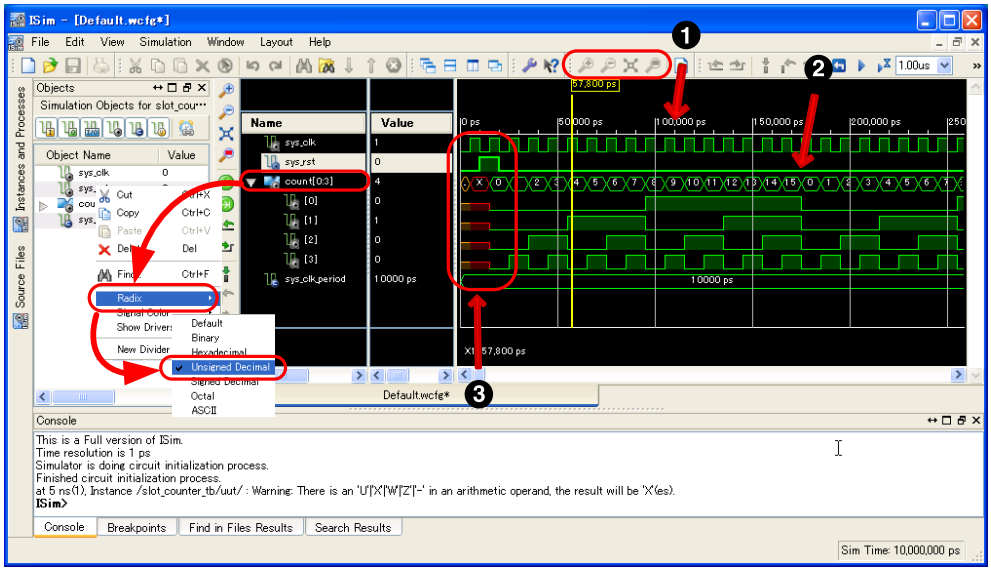


図 9.7 シミュレーション結果

❶ sys_clk は 1、0 を繰り返します。

❷ count は 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。

❸ count の一番初めは値が不定です。sys_rst が入力されると、初期化されて値が決定します。

10.スロットマシン製作 1(IP コアの中身を作る)

「5.3. 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。これと同じスロットマシン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、SUZAKU のデフォルトにスロットマシンの IP(Intellectual Property)コアを接続し、ソフトウェアで制御することによりスロットマシンを製作しています。

本章ではまず、下図の右側の IP コアの中身を製作します。左側のソフトウェアについては後の章で説明します。

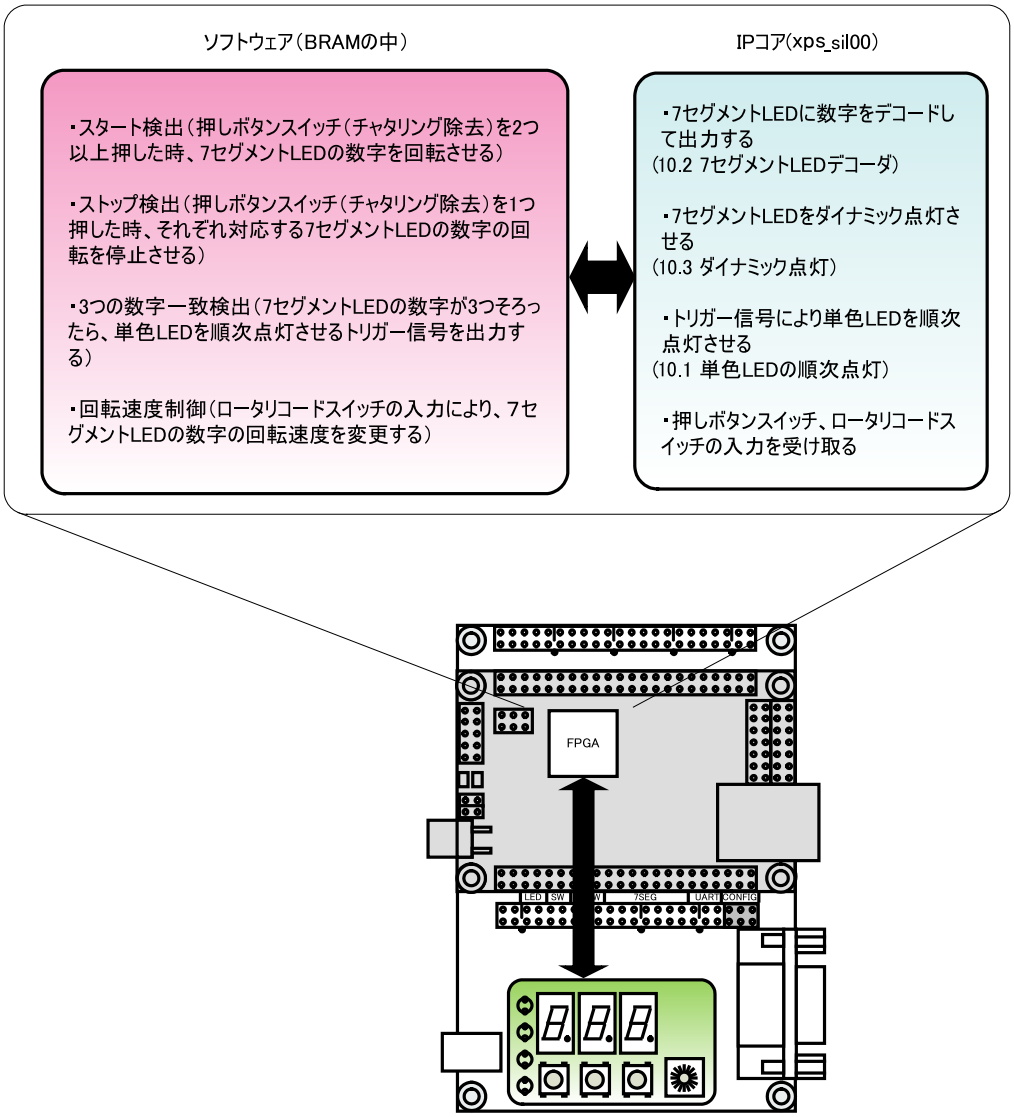


図 10.1 スロットマシンの構成

10.1. 単色 LED の順次点灯

まず、"トリガー信号により単色 LED を順次点灯させる"の単色 LED の順次点灯の部分を製作します。

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→・・・・)させます。

SZ130 のクロックは 3.6864MHz、SZ410 のクロックは 100MHz となっています。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。シミュレーションはカウンタのビット幅 4 で行います。シミュレーション後はビット幅を SZ130 の場合は 19 ビット、SZ410 の場合は 23 ビットに変更します。カウンタの最上位ビット count(0)の値は SZ130 の場合は 2^{19}=524288 カウントごとに(約 7Hz)、SZ410 の場合は 2^{23}=8388608 カウントごとに(約 12Hz)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 7.2. 単色 LED 周辺回路」をご参照ください。

10.1.2. プロジェクト新規作成、論理合成

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]とします。[New Source]で新規ソースファイルを作成します。ファイル名は `top.vhd` とします。

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。

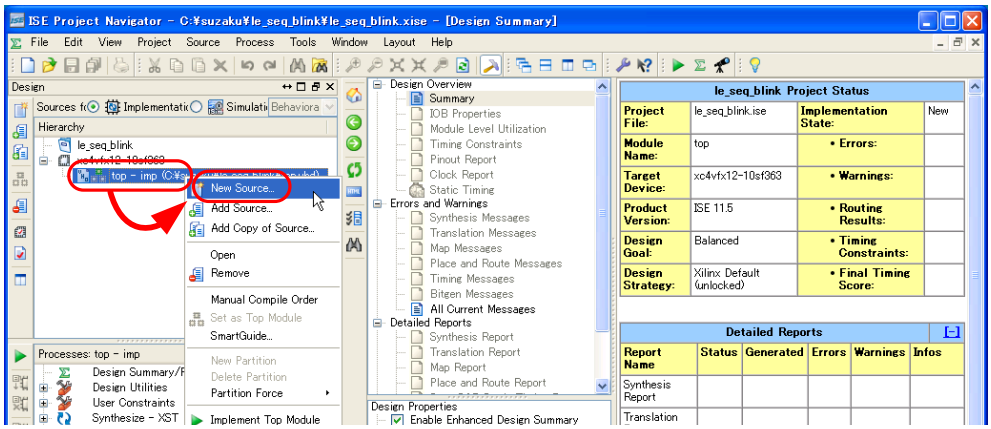


図 10.2 New Source の追加

New Source Wizard が立ち上がるので、[VHDL Module]を選択し、[File name]にファイル名を入力し、新しいソースファイルを作ります。ここでは `le_seq_blink.vhd` とします。

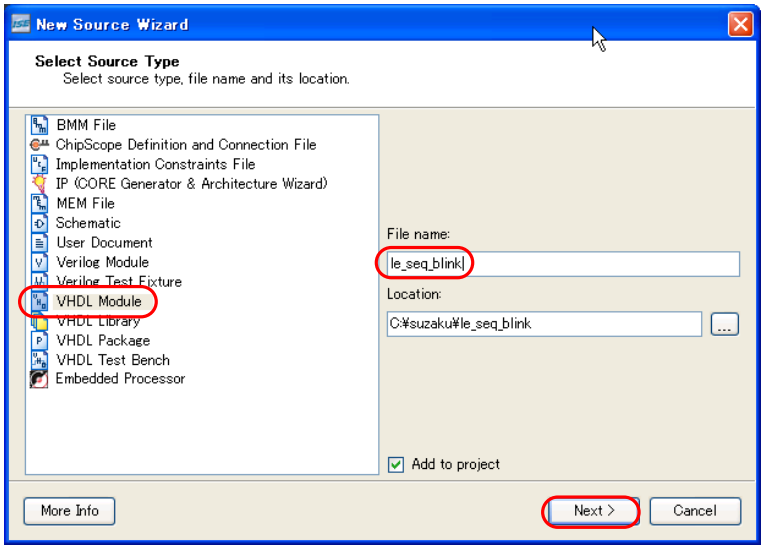


図 10.3 New Source 名前入力

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。

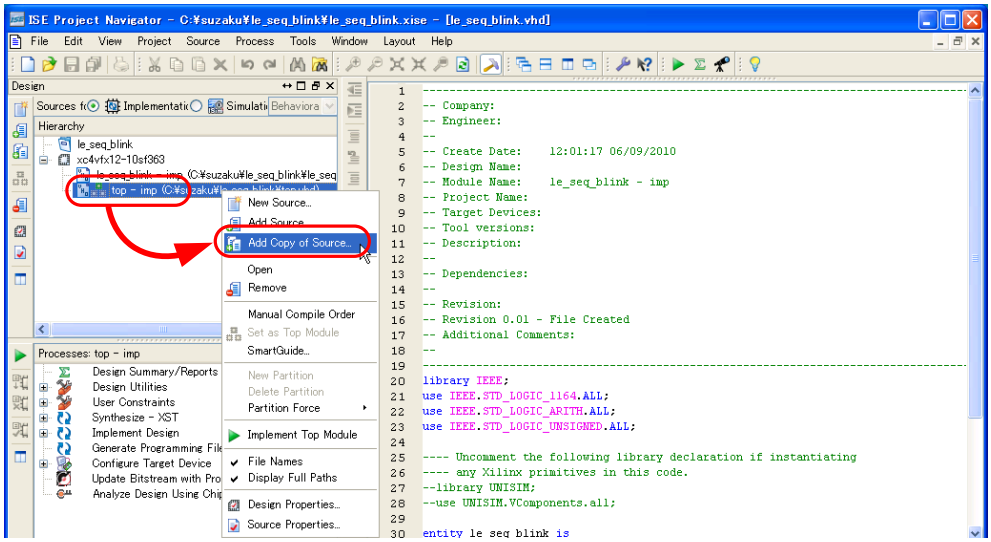


図 10.4 既存のソースファイル追加

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。既存のソースファイル追加時の確認が表示されるので、[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。

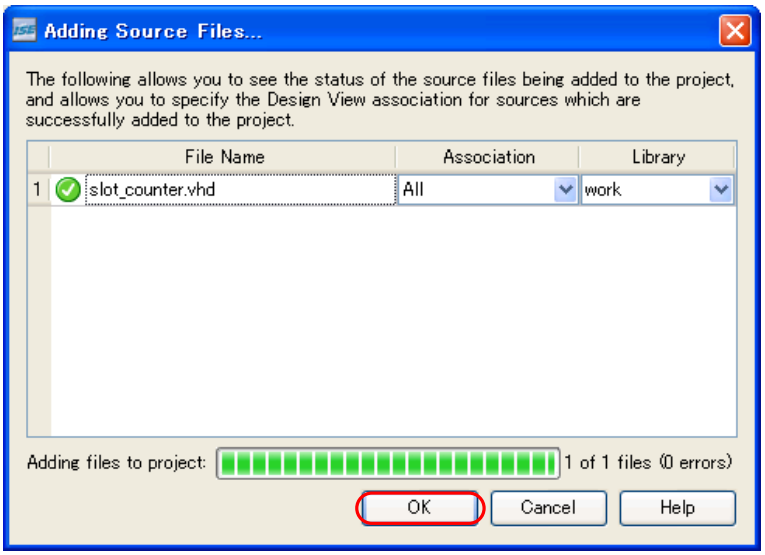


図 10.5 既存のソースファイル追加時の確認

10.1.2.1. ソースファイル編集(le_seq_blink.vhd)

le_seq_blink.vhd を開いてください。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;

  le_tim <= le_timing and (not le_tim_reg); -- 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_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;
```

10.1.2.2. ソースファイル編集(top.vhd)

top.vhd を開いてください。top を上位階層として slot_counter と le_seq_blink の回路を呼び出します。

カウンタのビット幅をシミュレーション用に一旦 4 ビットに設定します。実際のビット数でシミュレーションを行ってもいいのですが、LED が順次点灯する様子を見るのに、非常に長い時間かかります。今回はエッジ検出の様子とシフトレジスタの様子を確認したいだけなので 4 ビットにします。

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 := 4 -- カウンタのビット幅(シミュレーション用)
--    C_CNT_WIDTH : integer := 19 -- カウンタのビット幅(SZ410 の時は 23)
  );
  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;
```

10.1.2.3. コンポーネント文について

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 10.3 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 10.4 port map 文

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

10.1.3. シミュレーション

単色 LED の順次点灯のシミュレーションを行います。今回はシミュレーションについてさらっとしか説明しませんので、分からない場合は「9. ISE Simulator の使い方」をご参照ください。

[Simulation]をチェックし、[Project]→[New Source]でテストベンチを新規作成してください。ここではファイル名を[le_seq_blink_tb]とします。上位階層のファイルを聞かれるで、[top]を選択してください。

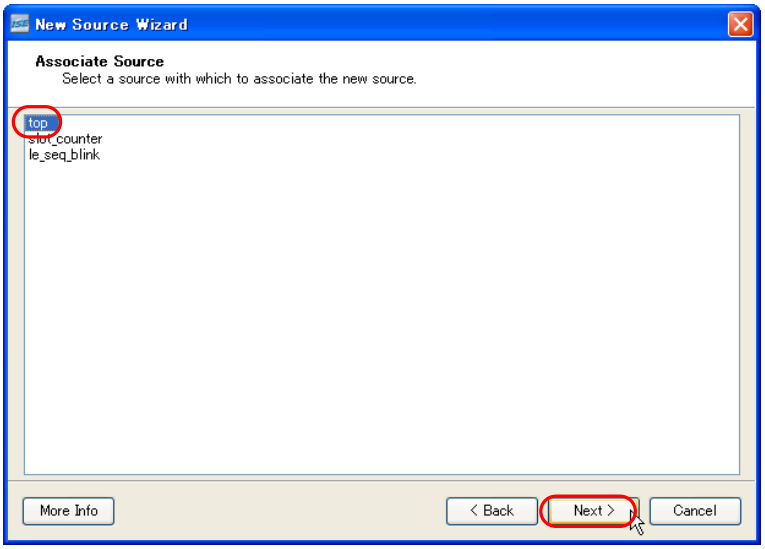


図 10.6 上位階層選択

le_seq_blink_tb-behavior(le_seq_blink_tb.vhd)をダブルクリックして開き、リセット信号の記述を前回と同様に追加して保存してください。

文法チェックを行ない、シミュレーションの設定を確認し、[Simulate Behavioral Model]をダブルクリックしてシミュレーションを開始してください。

このままでは確認したい波形を全てみれていないので、他の信号も追加します。[Instance and Process Name]ウィンドウの[le_seq_blink_tb]→[uut]→[le_seq_blink_0]を選択してください。選択できたら、[Object]ウィンドウの le_tim や le_tim_reg、le_timing、le を追加してください。ドラッグ&ドロップで追加できます。追加できたら[Simulation]→[Restart]、[Simulation]→[Run]をクリックしてください。シミュレーションが再び行われます。

エッジ検出やシフトの様子、LED の順次点灯の様子を確認してください。

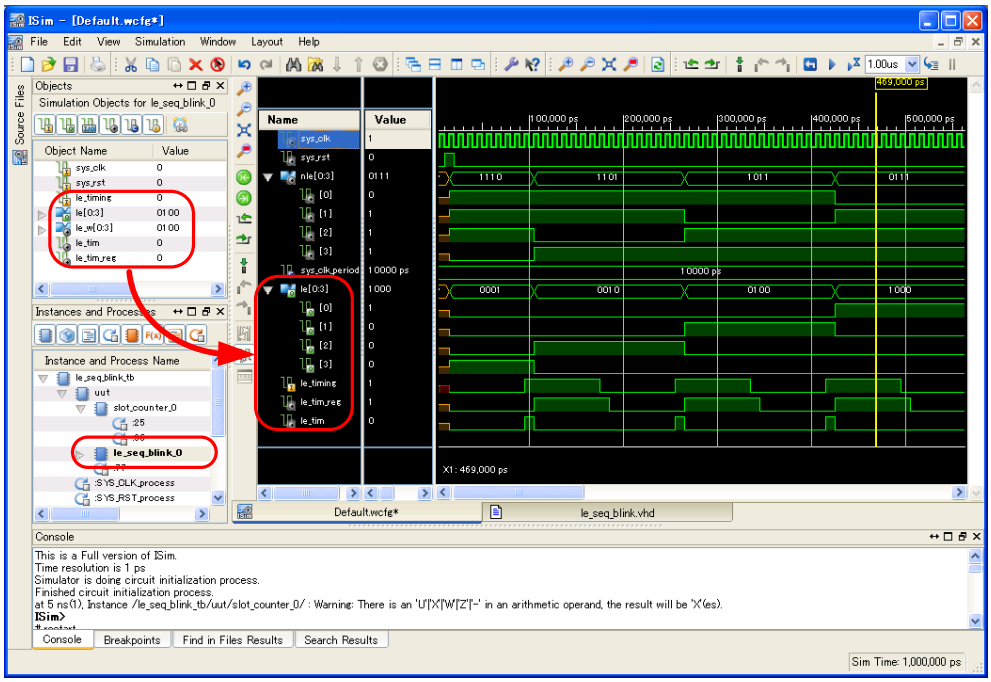


図 10.7 見たい信号を追加

10.1.3.1. タイミング信号生成(エッジ検出)について

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 10.5 エッジ検出

```
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_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;

le_tim <= le_timing and (not le_tim_reg); -- エッジ検出
```

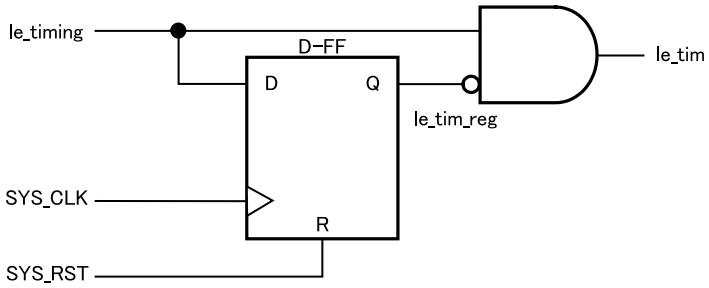


図 10.8 エッジ検出回路

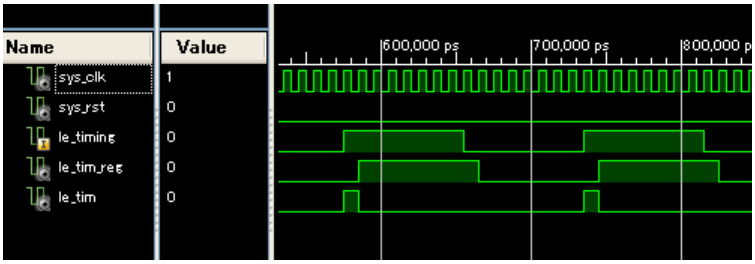


図 10.9 エッジ検出の波形

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

図 10.10 最上位ビットの動作

最上位ビットに注目してみて下さい。カウンタは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ます。

10.1.3.2. シフトレジスタについて

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。以下は左シフトの例となります。

例 10.6 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;
```

10.1.3.3. &について

&を使うと bit を連結することができます。

例 10.7 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。

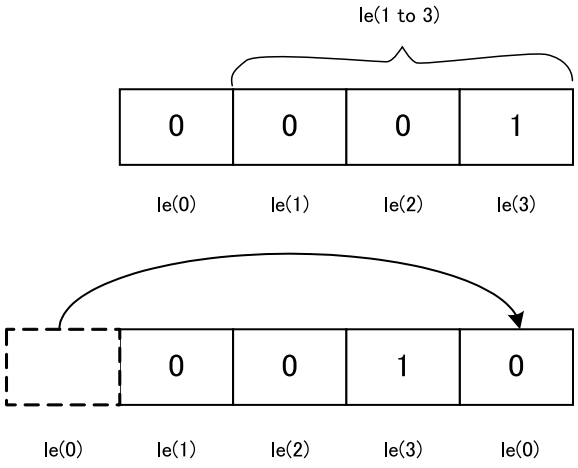


図 10.11 bit 連結

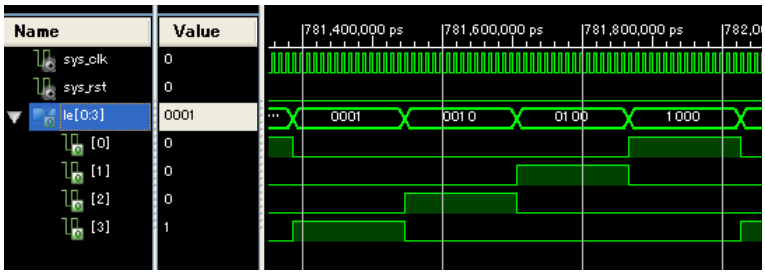


図 10.12 シフトレジスタの波形

10.1.4. 再度論理合成

[Implemantation]をチェックしなおしてください。シミュレーション用に 4 ビットにしていた `top.vhd` のカウンタのビット幅を 19 ビットにし、[Synthesize]をダブルクリックして文法チェックしてください。

10.1.5. インプリメンテーション

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 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。

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。

Signal Name	nLE3	nLE2	nLE3	nLE0
Pin Assign	nLE<0>	nLE<1>	nLE<2>	nLE<3>
	MSB			LSB

図 10.13 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

	SZ130	SZ410
SYS_CLK	U10	Y6
SYS_RST	D3	U3
nLE<0>	A11	E1
nLE<1>	B11	F1
nLE<2>	F12	F2
nLE<3>	E12	G2

10.1.6. プログラムファイル作成、コンフィギュレーション

[Generate Programming File]をダブルクリックし、bit ファイルを作成してください。bit ファイルが作成できたら、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→・・・・と順次点灯します。

うまくいかない場合は付属 CD-ROM に vhdl ファイルと ucf ファイルを収録[1]しているので、比較してみてください。

[1] `"\suzaku-starter-kit\fpga\le_seq_blink.zip"`に収録しています。

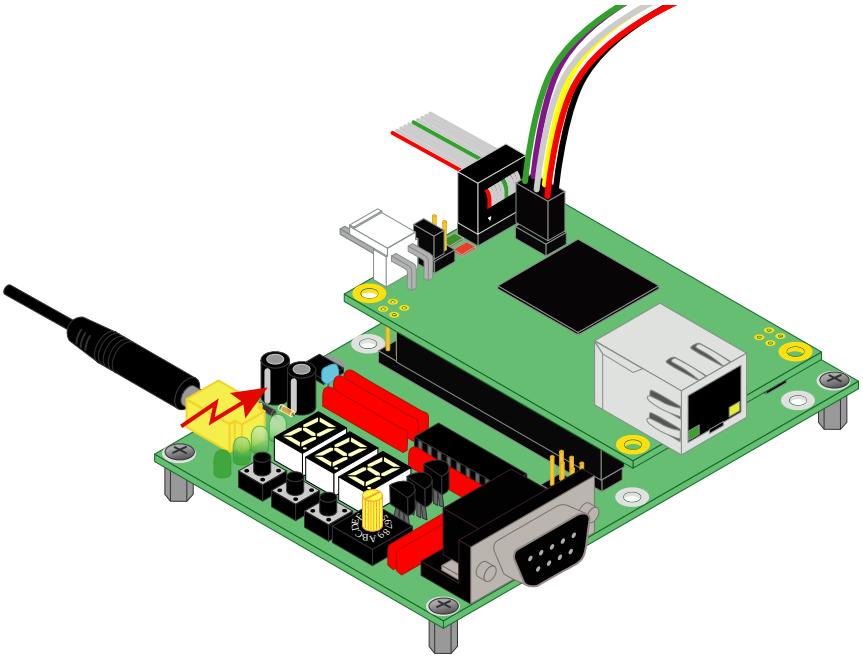


図 10.14 単色 LED 順次点灯

10.1.7. バスのビットラベルについて

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 側がビット 0 に定義されています。このため、LSB 側がビット 0 に定義されている外部デバイスと比べビットラベルが逆になります。LED/SW ボードも一般の外部デバイスと同じく、LSB 側をビット 0 に定義しています。

本書内の VHDL ソースコード内でバスの定義を行う場合、IBM の CoreConnect に合わせて MSB 側をビット 0 にしています。これを外部デバイスと接続するために、FPGA のピンアサインの設定で MSB と LSB をひっくり返しています。

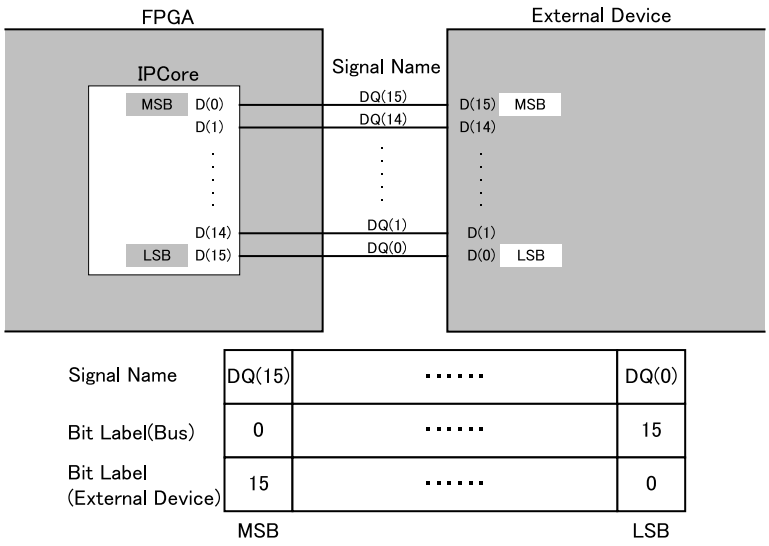


図 10.15 CoreConnect のビットラベルと信号

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 4 ビットで 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 10.2. ロータリコードスイッチ(正論理)」のようになります。

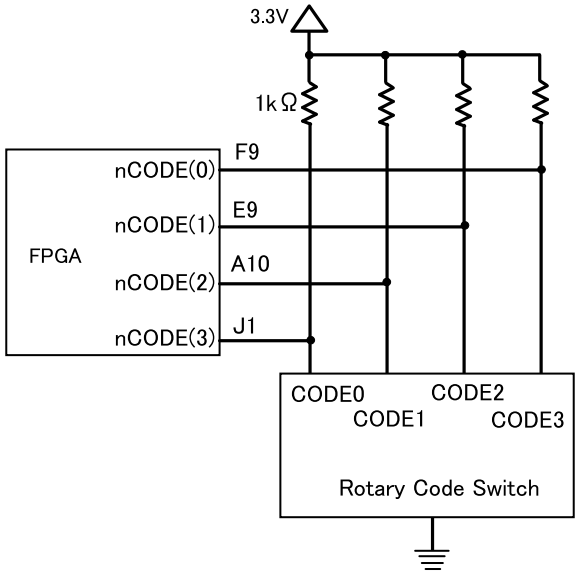


図 10.16 ロータリコードスイッチ周辺回路とピンアサイン

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

数字	CODE3	CODE2	CODE1	CODE0
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。

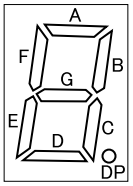


図 10.17 セグメントの配置

表 10.3 7 セグメント LED デコーダ(正論理)

数字	DP (SEG7)	G (SEG6)	F (SEG5)	E (SEG4)	D (SEG3)	C (SEG2)	B (SEG1)	A (SEG0)
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次の章で説明する、ダイナミック点灯という手法を用います。

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。

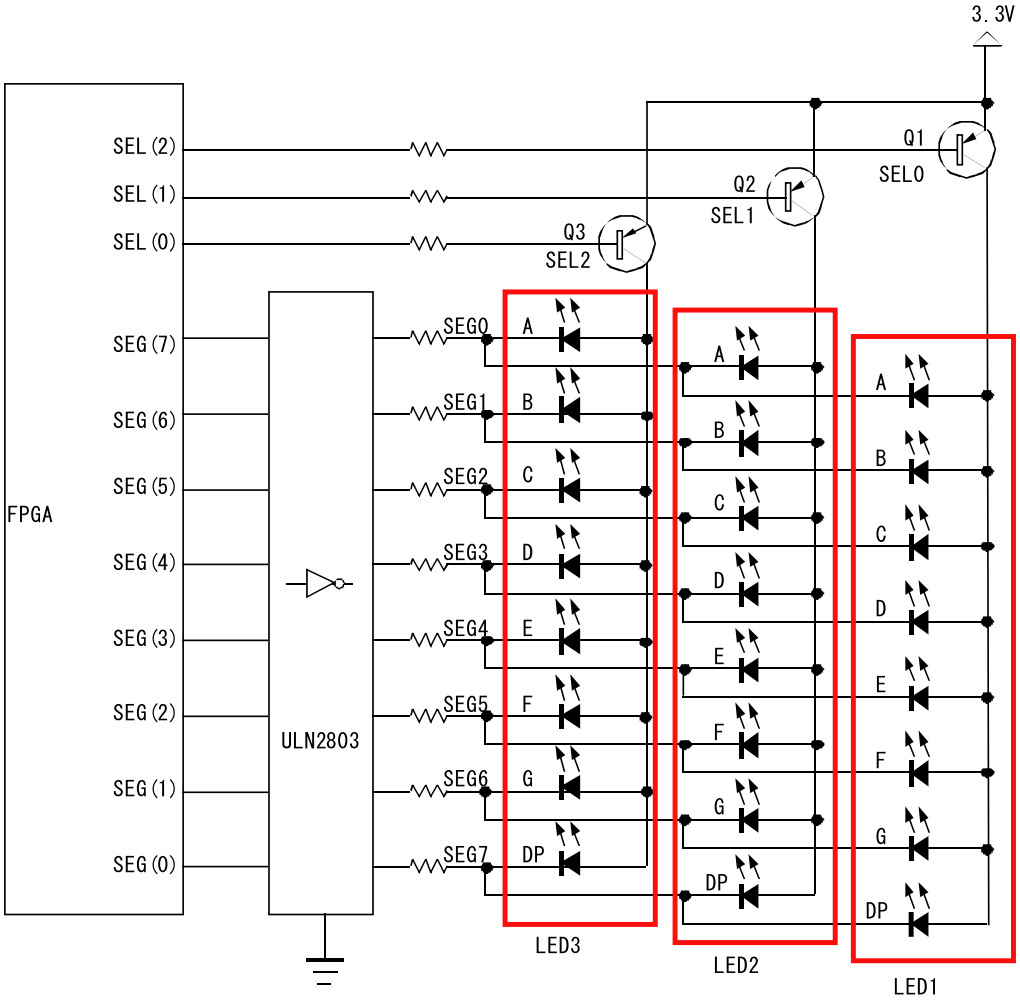


図 10.18 7 セグメント LED 周辺回路

10.2.3. プロジェクト新規作成、論理合成

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_decoder]とします。作成できたら[New Source]で新規ソースファイルを作成します。ファイル名は `top.vhd` とします。

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。New Source Wizard が立ち上がるので、[VHDL Module]を選択し、[File name]にファイル名を入力し、新しいソースファイルを作成します。ここでは `seg7_decoder.vhd` とします。

10.2.3.1. ソースファイル編集(seg7_decoder.vhd)

`seg7_decoder.vhd` を開いてください。7 セグメント LED のデコーダ回路を記述します。記述できたら、seg7_decoder-imp(seg7_decoder.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_decoder.vhd)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity seg7_decoder 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_decoder;

architecture imp of seg7_decoder 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 "1111" => SEG <= "01110001"; -- F
      when others => SEG <= "XXXXXXXX"; -- 0,1 以外(X,Z,U 等)の場合
    end case;
  end process;
end imp;
```

10.2.3.2. case 文について

case 文は次の書式で記述します。`when others => SEG <= "XXXXXXXX"`という記述がありますが、`std_logic_vector` は `0`、`1` の他に `X` や `Y`、`U` などの値を持っているため、残り全てを記述するために`"XXXXXXXX"`(出力不定)としています。

例 10.9 case 文

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

10.2.3.3. ソースファイル編集(top.vhd)

top.vhd を開いてください。top を上位階層として seg7_decoder の回路を呼び出します。7 セグメント LED は LED1 を点灯させることにします。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 10.10 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_decoder
    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_decoder_0 : seg7_decoder
    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;
```

10.2.4. シミュレーション

デコーダのシミュレーションを行います。今回も簡単にしか説明しませんので、分からなくなった場合は「9. ISE Simulator の使い方」をご参照ください。

[Simulation]をチェックし、[Project]→[New Source]でテストベンチを新規作成してください。ファイル名は[seg7_decoder_tb]とします。上位階層のファイルを聞かれるで、[top]を選択してください。

出来上がったテストベンチを開いて下さい。テンプレートが生成されています。必要のないクロックの記述があるので削除し、100ms 間隔でロータリコードスイッチの値を 1 ～ F まで変更するコードを追加します。[File]→[Save]をクリックし、保存してください。

例 10.11 デコーダのシミュレーション(seg7_decoder_tb.vhd)

```
-- 前略
BEGIN

  -- Instantiate the Unit Under Test (UUT)
   uut: top PORT MAP (
          nCODE =< nCODE,
          SEG =< SEG,
          nSEL =< nSEL
        );

   -- No clocks detected in port list. Replace >clock< below with
   -- appropriate port name

-- クロックの記述は削除
--   constant <clock>_period := 10 ns;
--   <clock>_process :process
--   begin
--    <clock> <= '0';
--    wait for <clock>_period/2;
--    <clock> <= '1';
--    wait for <clock>_period/2;
--   end process;

   -- Stimulus process
   stim_proc: process
   begin

-- ここの記述も必要ないので削除
      -- hold reset state for 100 ms.
--      wait for 100 ms;
--      wait for <clock>_period*10;

      -- insert stimulus here

-- ロータリコードの記述を追加
          nCODE <= "0000"; -- 0
          wait for 100 ms;
          nCODE <= "0001"; -- 1
          wait for 100 ms;
          nCODE <= "0010"; -- 2
          wait for 100 ms;
          nCODE <= "0011"; -- 3
          wait for 100 ms;
          nCODE <= "0100"; -- 4
          wait for 100 ms;
          nCODE <= "0101"; -- 5
          wait for 100 ms;
          nCODE <= "0110"; -- 6
          wait for 100 ms;
          nCODE <= "0111"; -- 7
          wait for 100 ms;
          nCODE <= "1000"; -- 8
          wait for 100 ms;
          nCODE <= "1001"; -- 9
```

```
            wait for 100 ms;
            nCODE <= "1010"; -- A
            wait for 100 ms;
            nCODE <= "1011"; -- B
            wait for 100 ms;
            nCODE <= "1100"; -- C
            wait for 100 ms;
            nCODE <= "1101"; -- D
            wait for 100 ms;
            nCODE <= "1110"; -- E
            wait for 100 ms;
            nCODE <= "1111"; -- F
            wait;

    end process;

END;
```

[Behavioral Check Syntax]をダブルクリックし、文法チェックをして下さい。エラーがなければ[Simulate Behavioral Model]を右クリックしてメニューを出し、[Proccess Properties]を選択して下さい。シミュレーションの時間(Simulation Run Time)を[1500 ms]に変更し、[OK]をクリックして下さい。

[Simulate Behavioral Model]を右クリックしてメニューを出し、[Run]を選択して下さい。

ロータリコードスイッチ nCODE は負論理の信号で `top.vhd` で正論理の信号 `code` にしています。負論理のままでは何がデコードされているのか分かりにくいので、正論理の信号 `code` を追加します。追加できたら[Simulation]→[Restart]、[Simulation]→[Run All]をクリックしてください。

ロータリコードスイッチの値がきちんとデコードされて 7 セグメント LED に渡されているのか確認します。「表 10.2. ロータリコードスイッチ(正論理)」、「表 10.3. 7 セグメント LED デコーダ(正論理)」を参照してデコードされた結果が正しいか確認してください。

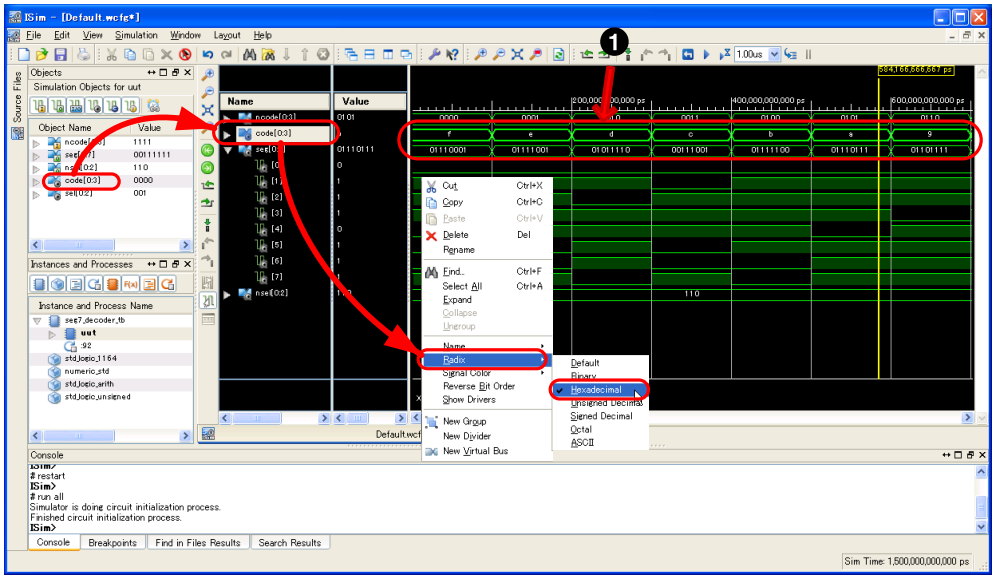


図 10.19 デコーダシミュレーション結果

❶ ロータリコードスイッチの値がデコードされて 7 セグメントに表示される。

10.2.5. インプリメンテーション

[Implementation]をチェックしてください。信号を `STD_LOGIC_VECTOR(0 to n)`で定義しているので、MSB 側がビット 0 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

	SZ130	SZ410
nCODE<0>	J1	H5
nCODE<1>	F9	E2
nCODE<2>	E9	D2
nCODE<3>	A10	U9
SEG<0>	L5	P1
SEG<1>	L6	P2
SEG<2>	L4	L2
SEG<3>	L3	M2
SEG<4>	L2	N2
SEG<5>	L1	N3
SEG<6>	C9	Y7
SEG<7>	D9	W7
nSEL<0>	K6	N5
nSEL<1>	K4	M3
nSEL<2>	K3	M4

10.2.6. プログラムファイル作成、コンフィギュレーション

[Generate Programming File]をダブルクリックし、bit ファイルを作成して下さい。bit ファイルが作成できたら、iMPACT を立ち上げコンフィギュレーションして下さい。

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。

うまくいかない場合は付属 CD-ROM に vhdl ファイルと ucf ファイルを収録[2]しているので、比較してみてください。

[2] `"\suzaku-starter-kit\fpga\seg7_decoder.zip"`に収録しています。

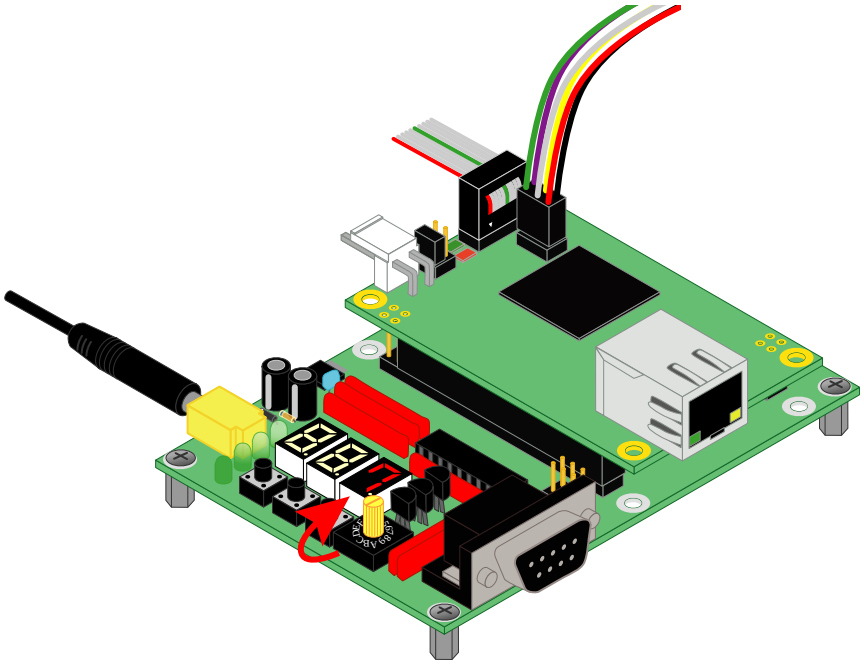


図 10.20 7 セグメント LED デコーダ

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。

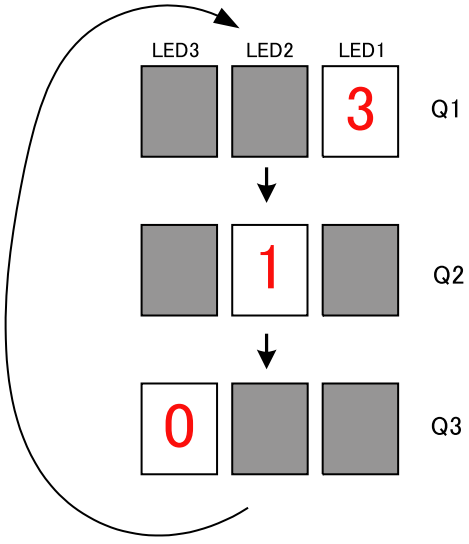


図 10.21 7 セグメント LED ダイナミック点灯

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 10.18. 7 セグメント LED 周辺回路」をご参照ください。

10.3.2. プロジェクト新規作成、論理合成

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]とします。作成できたら[New Source]で新規ソースファイルを作成します。ファイル名は `top.vhd` とします。

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。New Source Wizard が立ち上がるので、[VHDL Module]を選択し、[File name]にファイル名を入力し、新しいソースファイルを作成します。ここでは `dynamic_ctrl.vhd` とします。

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_decoder.vhd` を追加してください。

10.3.2.1. ソースファイル編集(dynamic_ctrl.vhd)

`dynamic_ctrl.vhd` を開いてください。ダイナミック点灯させる回路を記述します。記述できたら、dynamic_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 10.12 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 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;
```

```
  seg7_tim <= seg7_timing and (not seg7_tim_reg); -- 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)
        sel <= "001"; -- はじめに 7 セグメント LED 1 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;

  -- セレクト信号により代入する数字を変え,外部に出力
  seg_data <= seg_in1 when sel = "001" else seg_in2 when sel = "010" else seg_in3;
  nSEL <= not sel; -- 負論理に直して出力

end imp;
```

10.3.2.2. ソースファイル編集(top.vhd)

top.vhd を開いてください。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 を上位階層として slot_counter、seg7_decoder、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 10.13 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 (SZ410 の場合は 23)
  );
  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_decoder
    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_decoder_0 : seg7_decoder
    Port map(
      SEG      => SEG,
      seg_data => seg_data
    );
```

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

end imp;
```

10.3.3. シミュレーション

シミュレーションの説明はいたしませんので、各自シミュレーションしてみてください。下図のシミュレーション結果は、カウンタの値を 12 にしてシミュレーションを行った結果です。

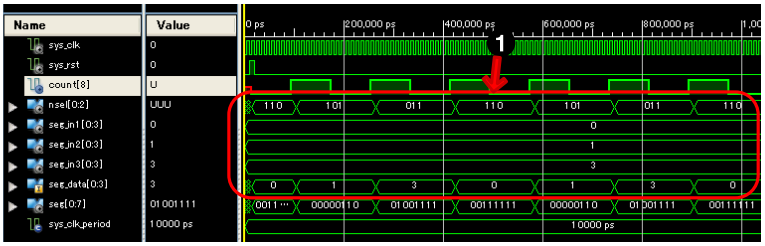


図 10.22 ダイナミック点灯シミュレーション結果

❶ nSEL の値により表示する 7 セグメント LED を切り替え

10.3.4. インプリメンテーション

「表 2.1. クロック、リセット信号のピンアサイン」、「表 2.2. 機能用ピンアサイン(CON2)」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。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。

10.3.5. プログラムファイル作成、コンフィギュレーション

[Generate Programming File]をダブルクリックし、bit ファイルを作成して下さい。bit ファイルが作成できたら、iMPACT を立ち上げコンフィギュレーションして下さい。

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。

うまくいかない場合は付属 CD-ROM に vhdl ファイルと ucf ファイルを収録[3]しているので、比較してみてください。

[3] "\suzaku-starter-kit\fpga\dynamic_ctrl.zip"に収録しています。

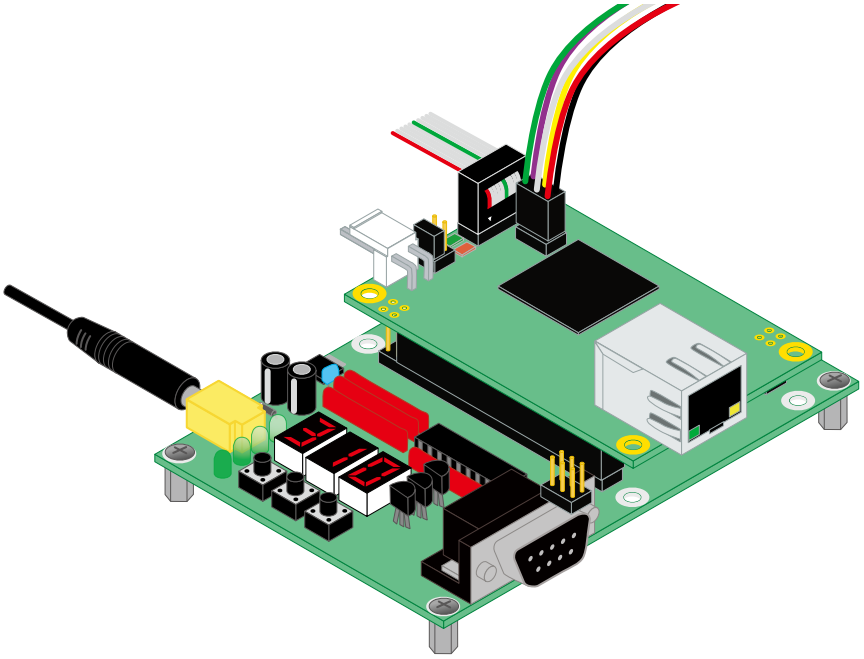


図 10.23 ダイナミック点灯

11.EDK の使い方

IP コアの中身が出き上がりました。次の作業には EDK(Embedded Development Kit)が必要になります。ここでは一旦スロットマシンと離れ、EDK の使い方を説明します。SUZAKU のデフォルトも EDK で構築しています。

EDK は Xilinx が提供する組み込み機器の統合開発環境です。XPS(Xilinx Platform Studio)、SDK(Software Development Kit)という開発ツールと共に、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などが登録されおり、GUI 環境下でデザインを構築、設定することができます。自分が作った IP コアも登録することができます。ISE の機能を取り込んでいるので、論理合成、マッピングなども行うことができ、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ができます。EDK には日本語のヘルプ、マニュアル等も用意されていますので、困った時にはそちらも参照してください。

EDK において以下の手順で作業を行います。EDK で作業を行う場合、タイトルと作業の名前をそろえているので、どこの作業を行っているかはタイトルを目安にしてください。

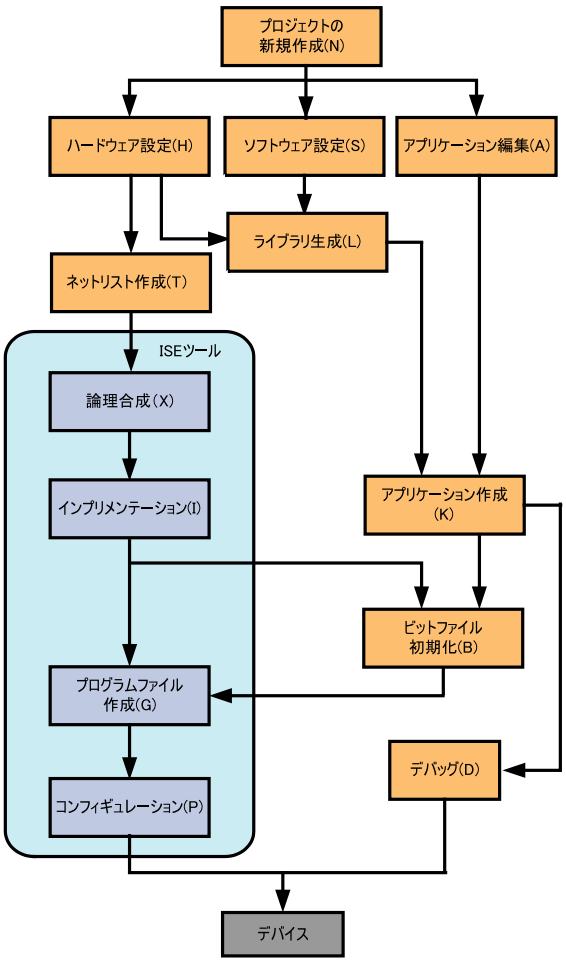


図 11.1 本書での EDK 開発フロー

11.1. BSB ではじめての MicroBlaze & PowerPC

まずは EDK に慣れるため、何もない状態からプロセッサが動くプロジェクトを作成します。EDK には BSB(Base System Builder)というウィザードが用意されており、プロセッサが動くプロジェクトを簡単に作ることができます。

ここでは BSB を使ってシリアル通信ソフトウェアの画面に Hello SUZAKU と表示するプロジェクトを作成します。構成は下図のようにします。

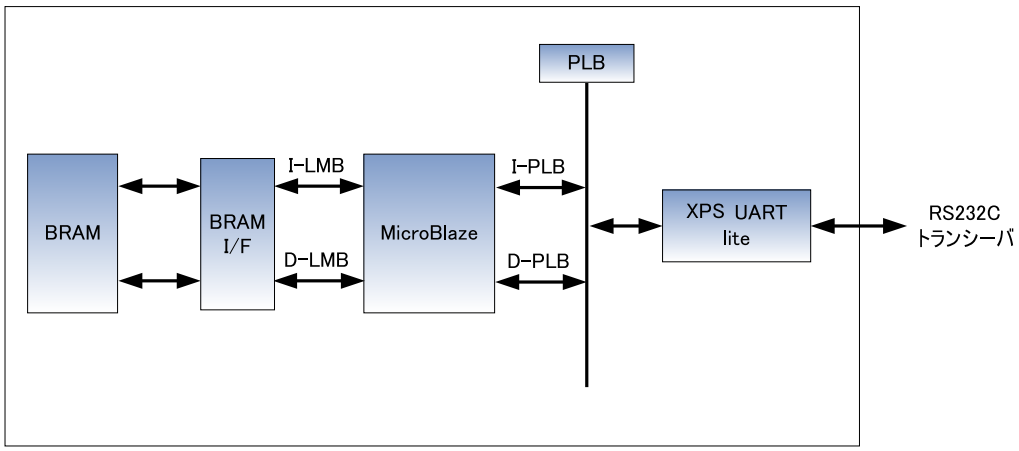


図 11.2 Hello SUZAKU プロジェクト(MicroBlaze)

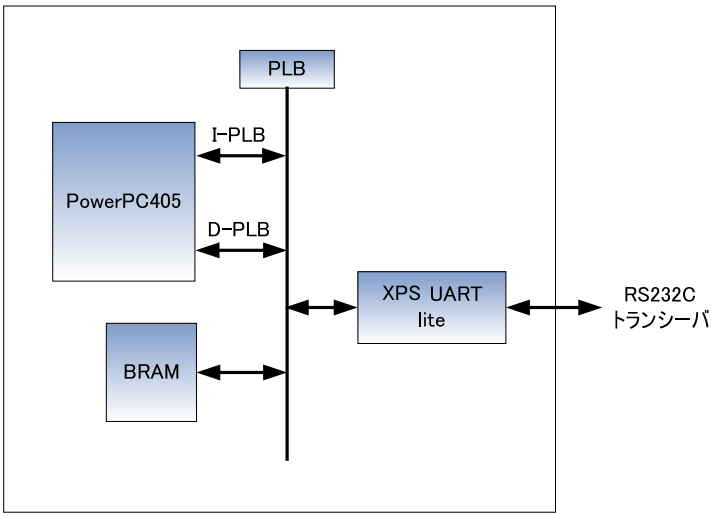


図 11.3 Hello SUZAKU プロジェクト(PowerPC)[1]

[1]SZ130 では PowerPC のプロジェクトは作成できません

11.2. プロジェクトの新規作成

XPS でプロジェクトを新規作成します。XPS を起動してください。XPS は[スタートメニュー]→[全てのプログラム]→[Xilinx ISE Design Suite X.X]→[EDK]→[Xilinx Platform Studio]から起動できます。

11.2.1. BSB 起動

以下の画面が表示されるので、[Base System Builder wizard]を選択して[OK]をクリックして下さい。もし表示されなかった場合は[File]→[New Project...]をクリックして下さい。

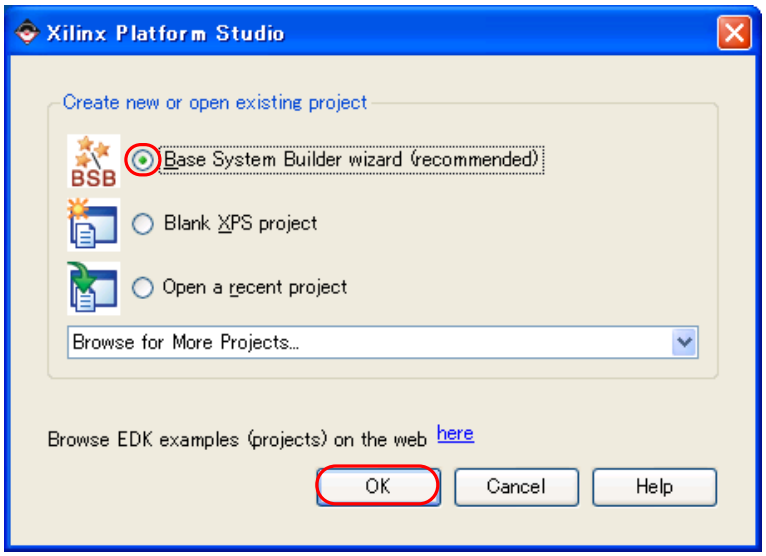


図 11.4 BSB 選択

11.2.2. プロジェクト作成

プロジェクトファイルの保存場所を聞かれます。ここでは[C:\suzaku\sz***\system.xmp](***は型式)とします。設定ができたら、[OK]をクリックして下さい。

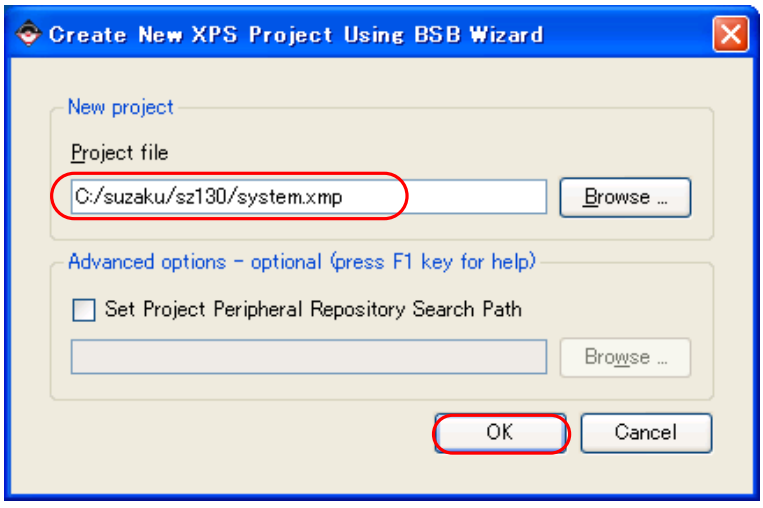


図 11.5 BSB ファイル保存

新しいデザインをはじめるので、[I would like to create a new design]を選択し、[Next]をクリックして下さい。

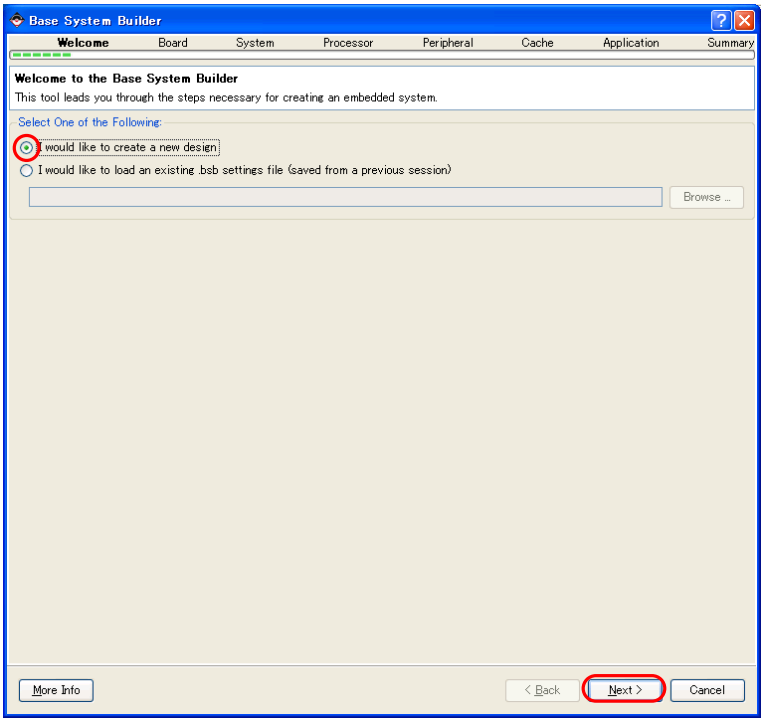


図 11.6 新しいデザインをはじめる

11.2.3. ターゲットボードとデバイスの選択

ターゲットとなるボードの選択を行います。1 から全てカスタムで作るので、[I would like to create a system for a custom board]を選択して下さい。

FPGA は、お使いの SUZAKU の型式の設定にしてください。[Reset Polarity]は[Active High]のままにして下さい。設定できたら、[Next]をクリックしてください。

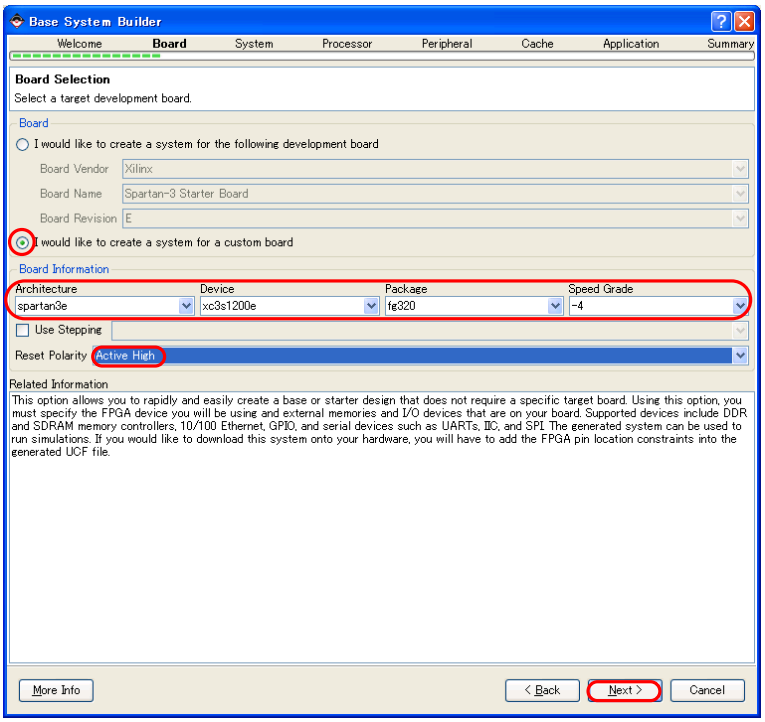


図 11.7 ターゲットボードの選択

型式	SZ130	SZ410
Architecture	spartan3e	virtex4
Device	xc3s1200e	xc4vfx12
Package	fg320	sf363
Speed grade	-4	-10

11.2.4. デザインの選択

シングルプロセッサかデュアルプロセッサか聞かれます。今回はシングルプロセッサにするので[Single-Processor System]をチェックし、[Next]をクリックして下さい。

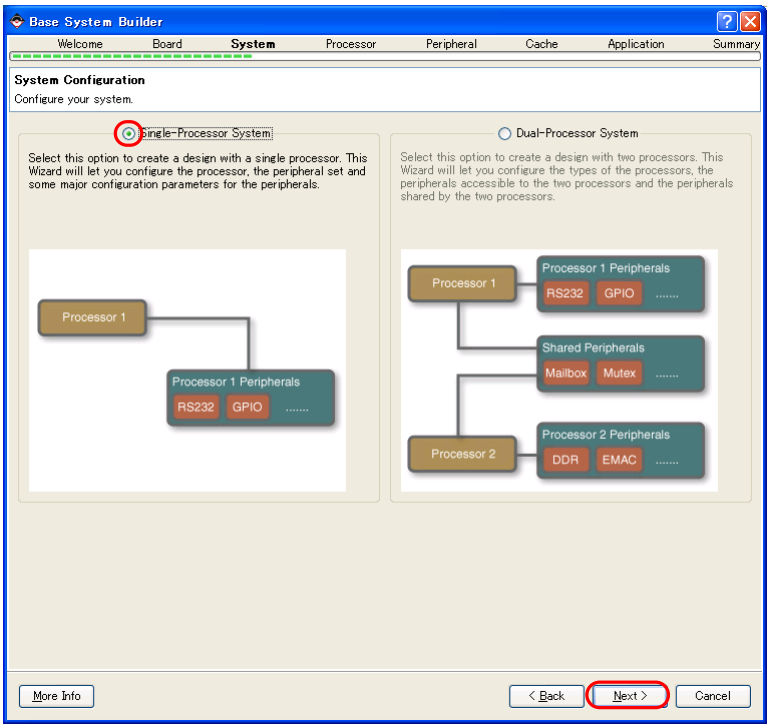


図 11.8 FPGA とプロセッサの設定

11.2.5. MicroBlaze の場合の設定(SZ130)

SZ130

SZ130 の場合クロックは 3.6864MHz なので、[Reference Clock Frequency]を[3.6864MHz]に設定したいところですが、エラーが出て設定することができません。とりあえず後の作業のことを考えて、[System Clock Frequency]を[50MHz]に設定し、[Next]をクリックして下さい。

11.2.6. MicroBlaze の場合の設定(SZ410)

SZ410

SZ410 の場合クロックは 100MHz なので、[Reference Clock Frequency]、[System Clock Frequency]を[100MHz]に設定し、[Next]をクリックして下さい。SZ410 の場合は PowerPC も選択できるので、プロセッサを PowerPC にする場合は PowerPC の場合の設定を確認して下さい。

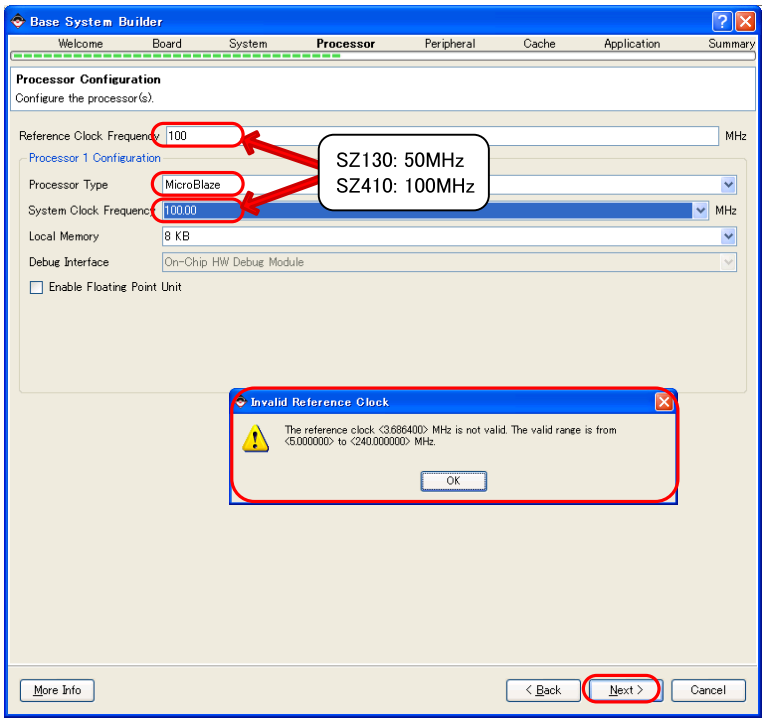


図 11.9 MicroBlaze の設定

11.2.7. PowerPC の場合の設定

SZ410

SZ410 は PowerPC も選択できます。[Reference Clock Frequency]を[100MHz]、[Processor Clock Frequence]は[300MHz]、[Bus Clock Frequency]は[100MHz]に設定して下さい。

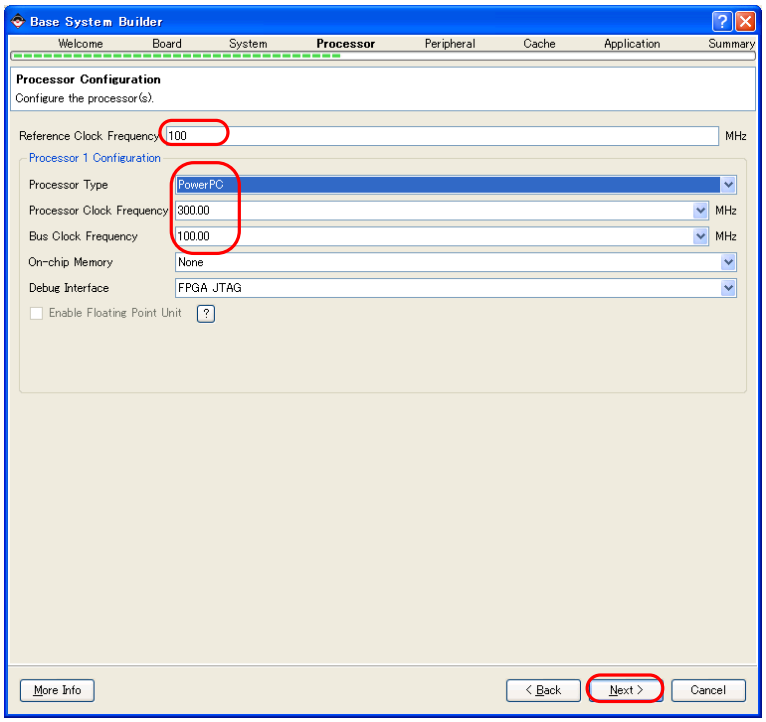


図 11.10 PowerPC の設定

11.2.8. ペリフェラルの設定

ペリフェラルの追加画面が表示されます。シリアル通信を行いたいので、UART のコアを追加します。[Add Device]をクリックし、[IO Interface Type]で[UART]を選択し、[Device]を[RS232]に変更し、[OK]をクリックして下さい。

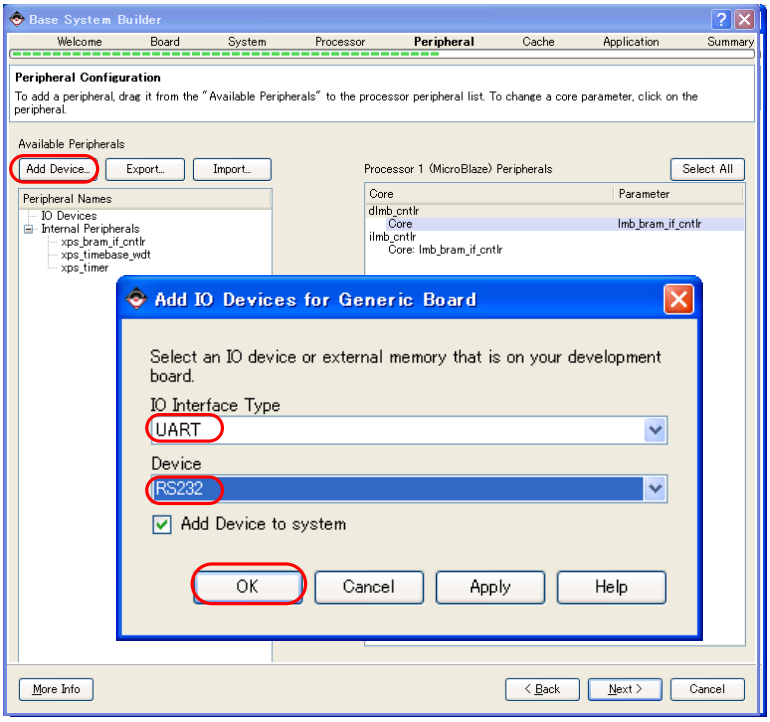


図 11.11 ペリフェラルの選択

以下のように UART が追加されます。[Baudrate]を[115200]に変更して下さい。PowerPC の場合、BRAM のサイズを[16KB]に変更します。xps_bram_if_cntlr_1 の Size を[16KB]に変更して下さい。変更できたら[Next]をクリックして下さい。

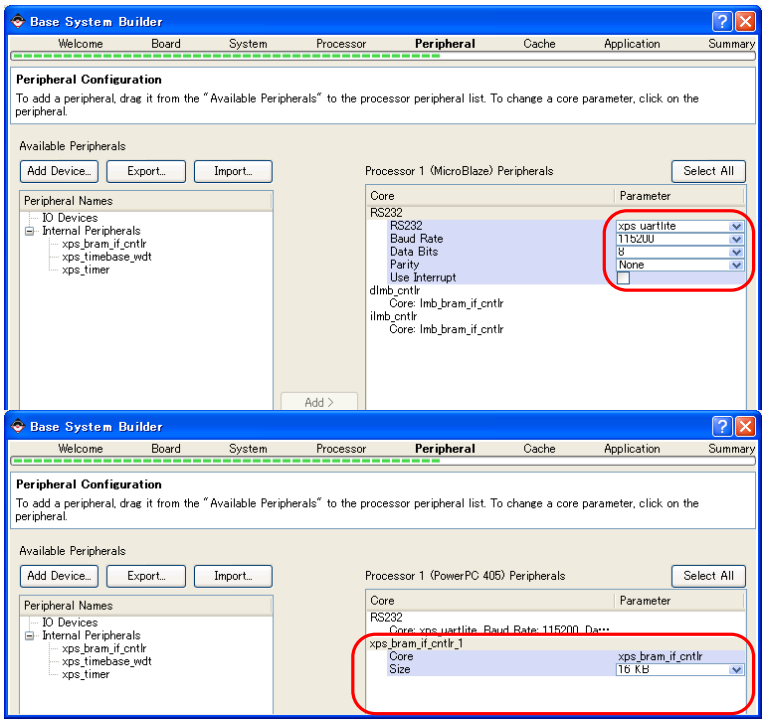


図 11.12 ペリフェラルの設定

キャッシュの設定画面が表示されます。今回はキャッシュはなしにします。変更せずに[Next]をクリックして下さい。

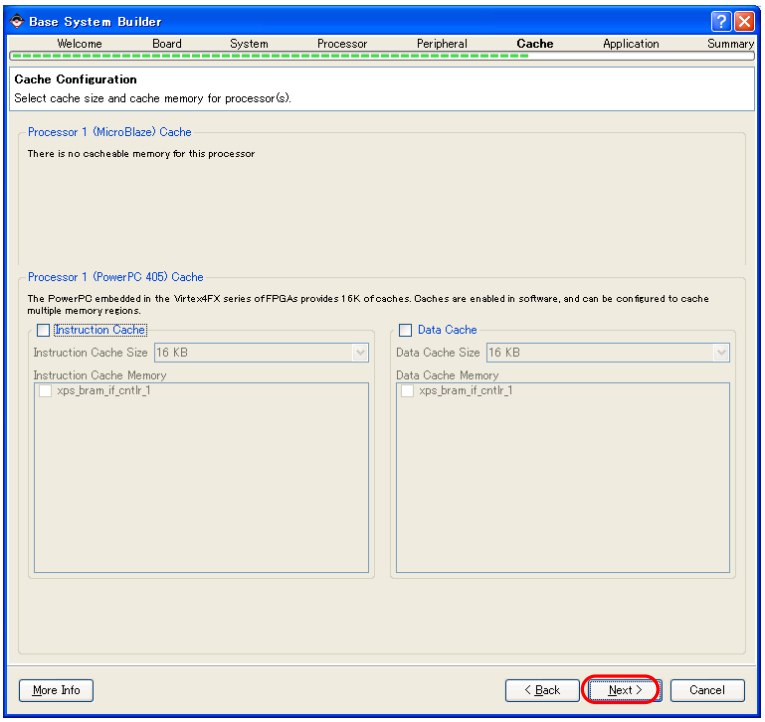


図 11.13 キャッシュの設定

11.2.9. ソフトウェアの設定

ソフトウェアの標準入出力とサンプルアプリケーションの選択画面が表示されます。変更せずに[Next]をクリックして下さい。

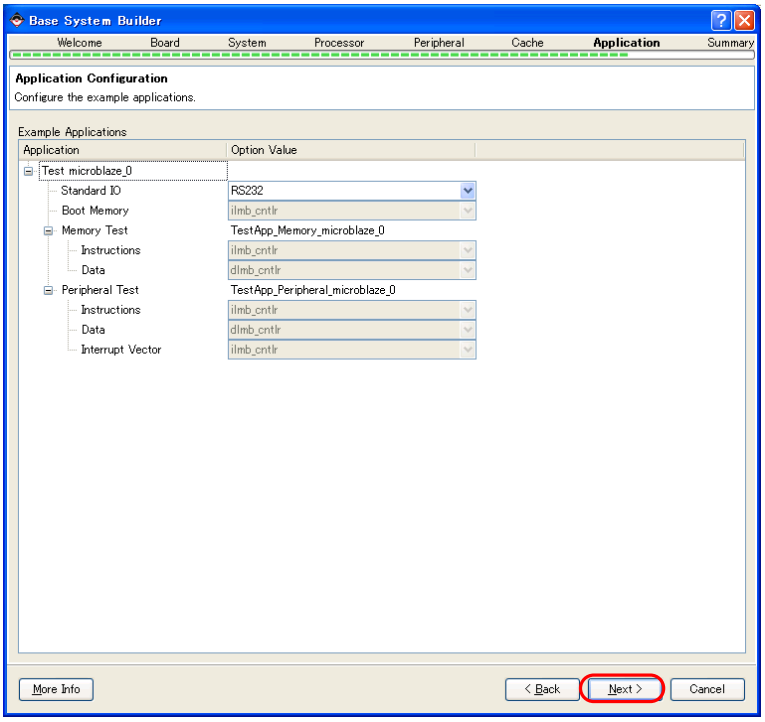


図 11.14 ソフトウェアに関する設定(MicroBlaze の場合)

11.2.10. デザインの設定確認

これまでに設定した項目が表示されます。内容を確認し、[Finish]をクリックしてください。BSB でデザインを作成した際の注意が表示されるので、こちらも内容を確認し[OK]をクリックして下さい。

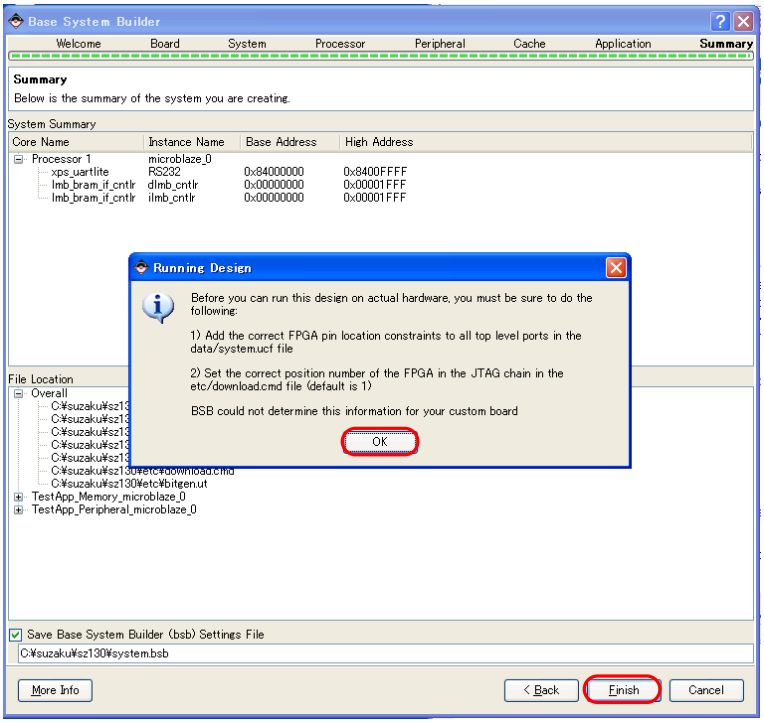


図 11.15 設定の確認(SZ130)

次に行う作業を聞かれるので、[Start using Plarform Studio]をチェックし、[OK]をクリックして下さい。

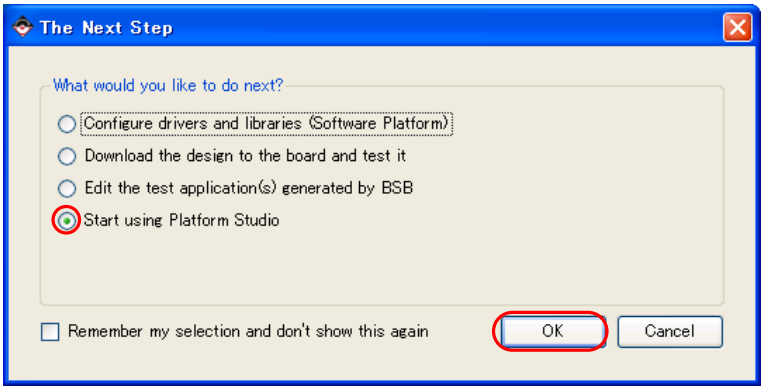


図 11.16 次の作業の決定

以下のような構成で自動生成されます。

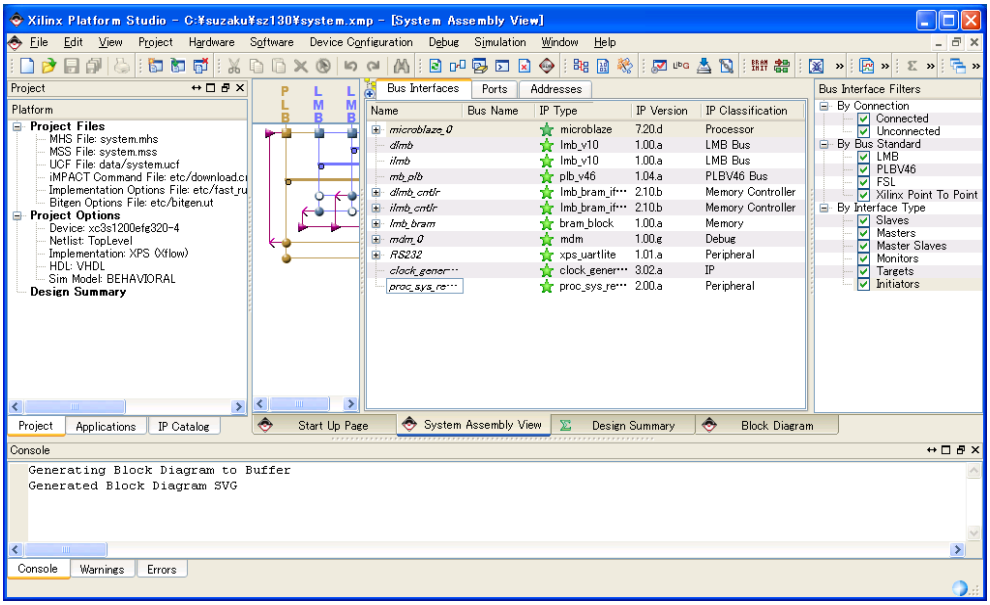


図 11.17 XPS の表示

DCM について

DCM は遅延ロックループ(DLL)、デジタル周波数合成(DFS)、位相シフト(Phase Shifter)、ステータスロジックの 4 つのユニットで構成されていて、これらは独立、または互いに関連して動作します。

DLL	クロック出力信号の伝搬遅延がゼロになるようにスキュー調整を行う。2 逓倍クロック、クロック分周、1/4 位相シフト出力を生成できる。
DFS	自分で設定した 2 つの整数により、逓倍、分周したクロックを生成できる。
Phase Shifter	CLKIN 入力に対するクロック出力の位相関係を制御する。
ステータスロジック	DCM の現在の状態を出力する。

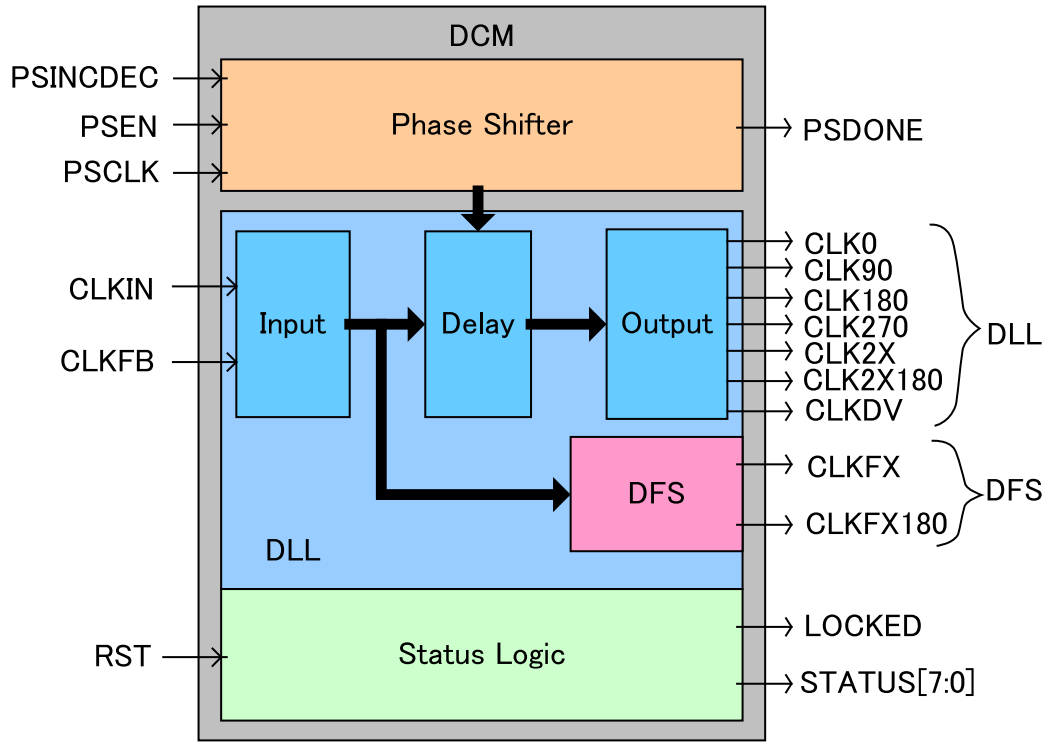


図 11.18 DCM の一部

SZ130 の場合、DLL の機能を使うと 3.6864MHz は範囲外なので入力できなくなってしまうため、DFS の機能だけ使うようにしなければなりません。

表 11.1 入力できるクロック周波数

型式	SZ130	SZ410
DLL(MHz)	5 ～ 200	24 ～ 180
DFS(MHz)	0.2 ～ 333	1 ～ 210

11.3. ハードウェア設定

11.3.1. GUI で編集(SZ130)

SZ130

SZ130 の場合エラーが出てクロック(3.6864MHz)が設定できませんでしたが、これは Clock Generator の制限によるものです。Clock Generator は DCM を自動で組み合わせてクロックを生成してくれる、とても便利な IP コアです。便利なのですが、自動で DCM の FB ピンまで使用してしまうため、常に DFS だけでなく DLL の機能まで使用してしまいます。DLL の機能を使用してしまうと、SZ130 で使用している Spartan-3E では、DCM に入力できるクロックが 5MHz からになってしまいます。[2] (DCM については「Tips DCM について」参照)

Clock Generator は残念ながら SZ130 では使用できないようなので、削除して DCM の IP コアを追加します。clock_generator_0 の上で右クリックしてメニューを出し、[Delete Instance]を選択して下さい。Clock Generator が削除されます。削除後接続していたピンの処理をどうするか聞かれます。今回は接続していたピンをそのまま使うことにします。[Delete instance but keep its ports]を選択して[OK]をクリックして下さい。

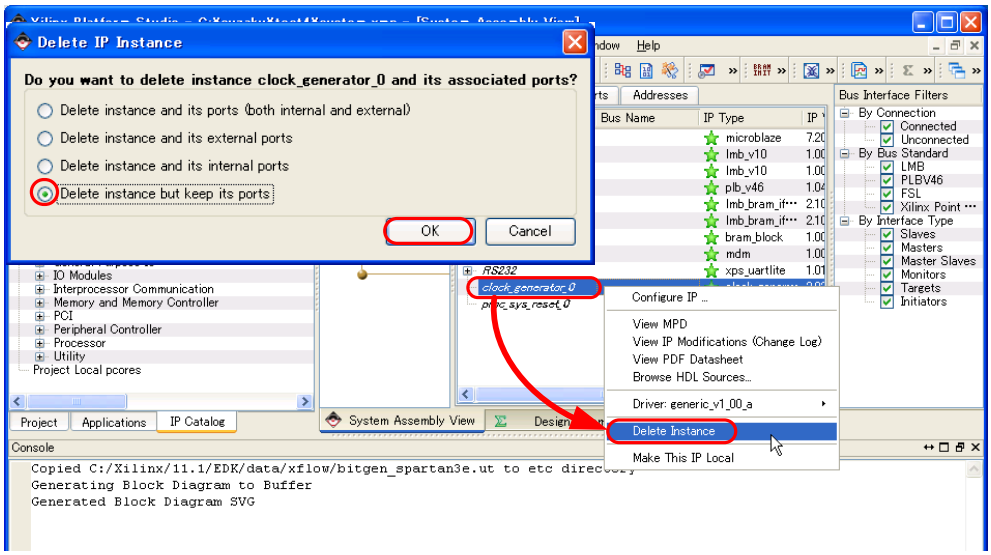


図 11.19 Clock Generator の削除

DCM を追加します。[IP Catalog]タブをクリックし、[Clock,Reset and Interrupt]→[Digital Clock Manager(DCM)]を右クリックしてメニューを出し、[Add IP]を選択して下さい。DCM が追加されます。

[2]もしかすると設定で DLL の機能を使用しないようにすることができるかもしれませんが、発見できませんでした。

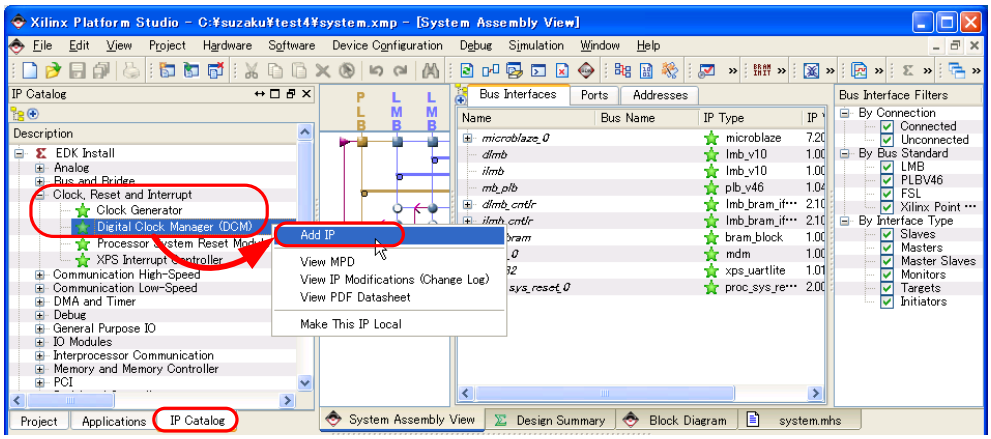


図 11.20 DCM の追加

DCM の設定を変更します。dcm_module_0 の上で右クリックしてメニューを出し、[Configure IP...]を選択して下さい。

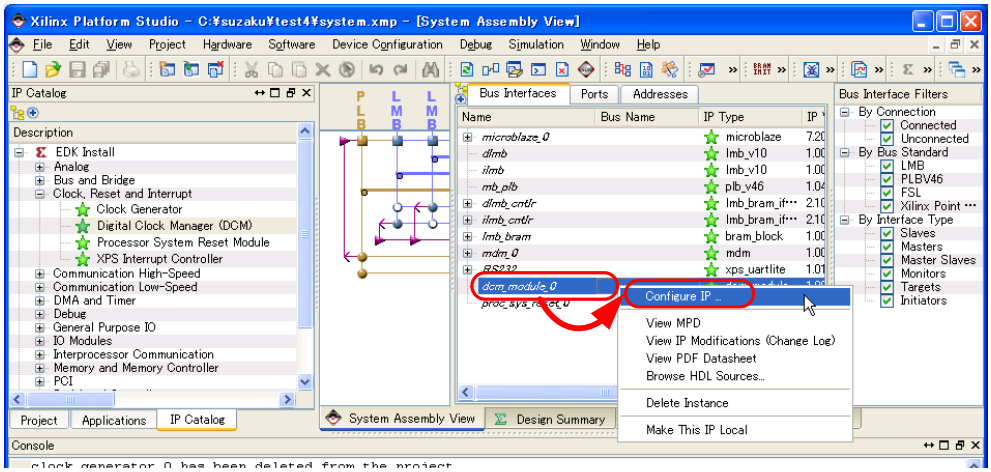


図 11.21 DCM の設定開始

設定画面が表示されます。[Clock Feedback Input]を[None]、[Multiply Value of the CLKFX Output]を[14]、[Input Clock Period]を左の[x]をクリックしてから[271.267361]に変更して下さい。

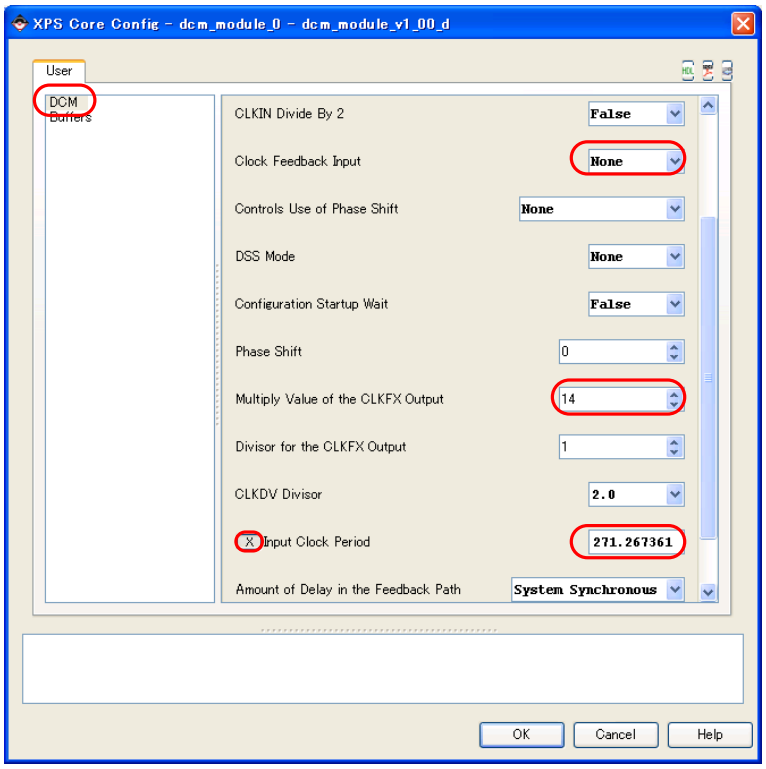


図 11.22 DCM の設定

バッファの設定をします。[Buffers]をクリックし、[Insert a BUFG for CLKFX]を[True]に設定して下さい。設定できたら[OK]をクリックして下さい。

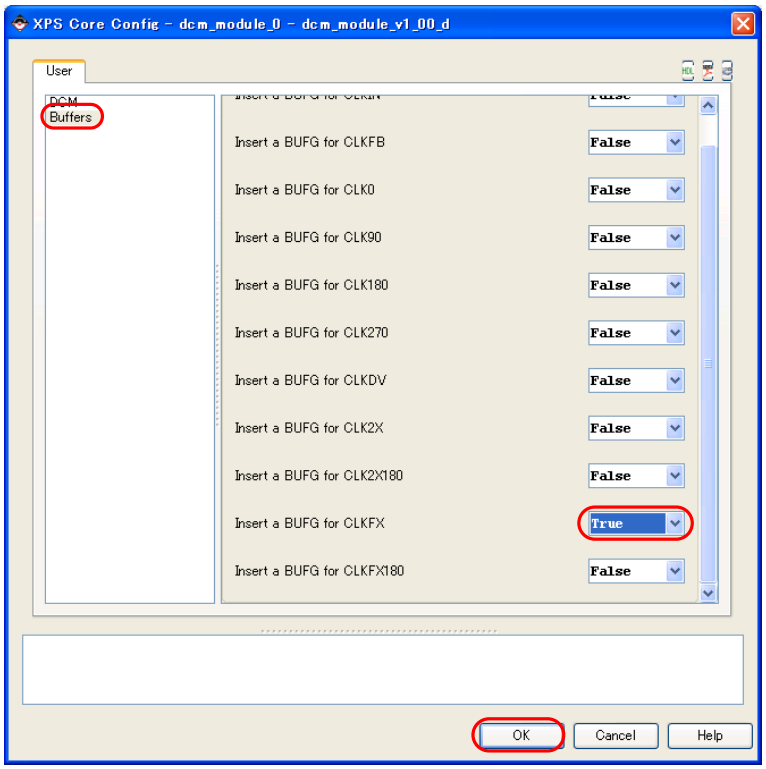


図 11.23 DCM のバッファ設定

次は DCM のピンの接続をします。[Ports]タブをクリックし、dcm_module_0 の左の⊞をクリックして開いてください。RST、CLKIN、LOCKED、CLKFX に信号を接続します。まずは RST の[No Connection]の横の矢印をクリックしてください。接続できるピンが表示されるので、[net_gnd]を選択して下さい。同様の手順で CLK_IN を[dcm_clk_s]、LOCKED を[Dcm_all_Locked]に接続して下さい。CLKFX は接続できるピンが表示されないので、[No_Connection] を削除して、手動で[clk_50_0000MHz]と記述して下さい。先程、[System Clock Frequency]を[50MHz]に設定したのは、ここに入力する名前が簡単なほうが良いという理由からでした。[Project]タブをクリックし、[MHS File: system.mhs]をダブルクリックして開くと、この名前を発見することができます。

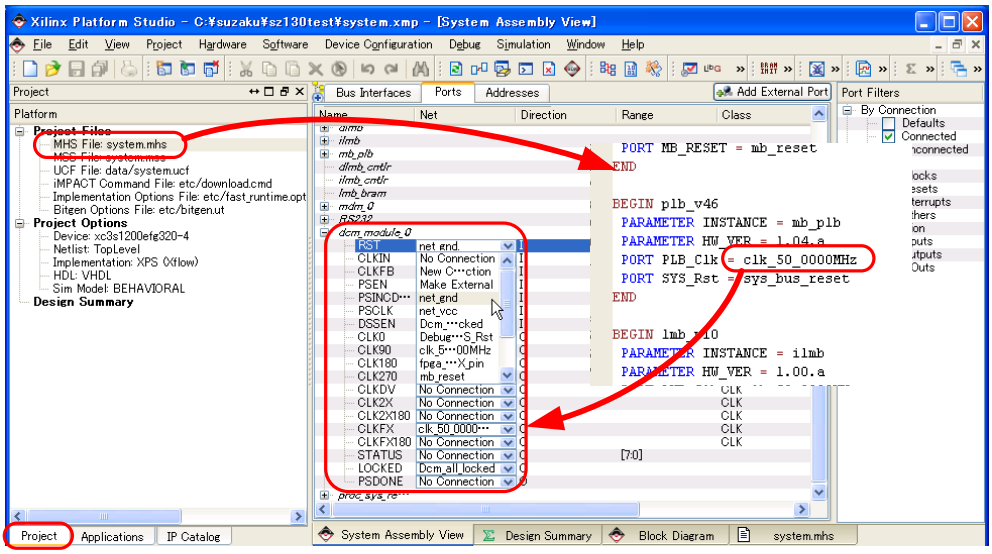


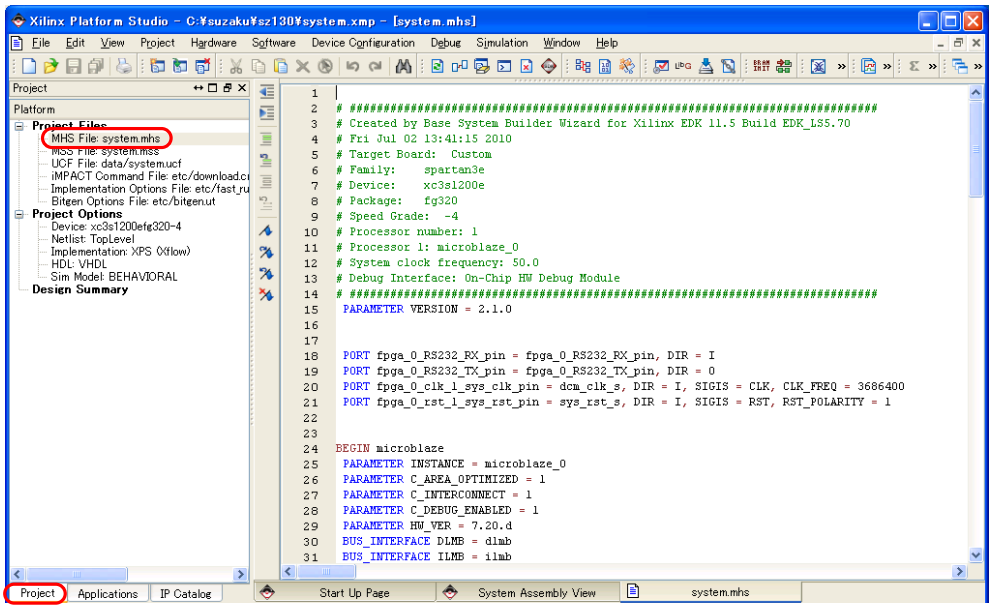
図 11.24 DCM のピンの接続

11.3.2. MHS ファイル編集(SZ130)

SZ130

クロックの記述を修正します。[Project]タブをクリックし、[MHS File: system.mhs]をダブルクリックして開いて下さい。

SZ130 の場合、上のほうに `CLK_FREQ=5000000` の記述があるので `CLK_FREQ=3686400` に変更します。




```
# 前略
PARAMETER VERSION = 2.1.0

PORT fpga_0_RS232_RX_pin = fpga_0_RS232_RX_pin, DIR = I
PORT fpga_0_RS232_TX_pin = fpga_0_RS232_TX_pin, DIR = O

PORT fpga_0_clk_1_sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ = 3686400

PORT fpga_0_rst_1_sys_rst_pin = sys_rst_s, DIR = I, SIGIS = RST, RST_POLARITY = 1
# 後略
```

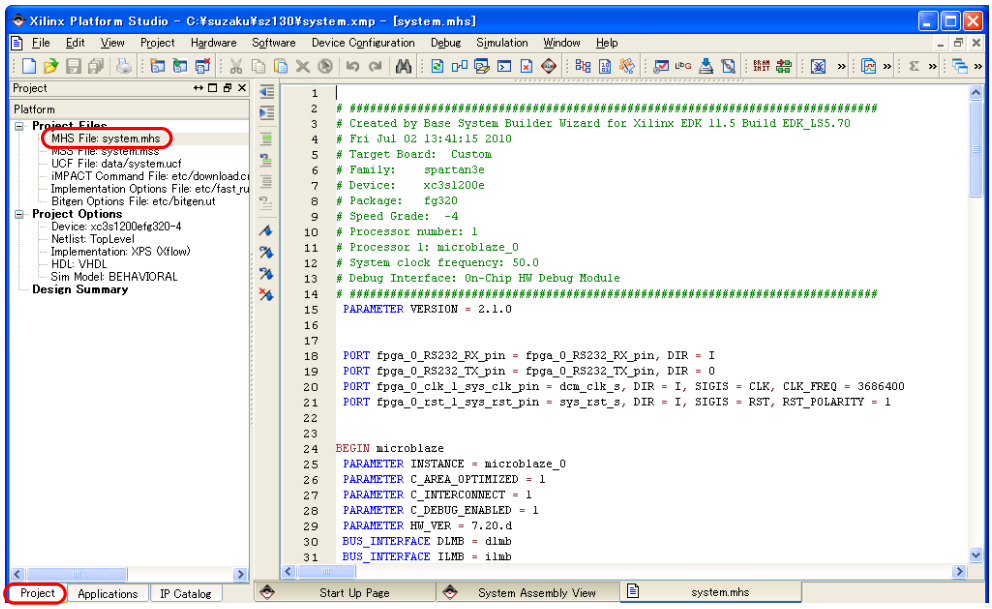
図 11.25 SZ130 の MHS ファイル編集(system.mhs)

11.3.3. MHS ファイル編集(SZ410)

SZ410

クロックの記述を修正します。[Project]タブをクリックし、[MHS File: system.mhs]をダブルクリックして開いて下さい。

SZ410 の場合、CLK_FREQ = 50000000 を CLK_FREQ = 100000000、C_CLKIN_FREQ = 50000000 を C_CLKIN_FREQ = 100000000 に変更します。修正箇所は全部で 2 箇所です。




```
# 前略
PARAMETER VERSION = 2.1.0

PORT fpga_0_RS232_RX_pin = fpga_0_RS232_RX_pin, DIR = I
PORT fpga_0_RS232_TX_pin = fpga_0_RS232_TX_pin, DIR = O

PORT fpga_0_clk_1_sys_clk_pin = dcm_clk_s, DIR = I, SIGIS = CLK, CLK_FREQ =
100000000

PORT fpga_0_rst_1_sys_rst_pin = sys_rst_s, DIR = I, SIGIS = RST, RST_POLARITY = 1
# 中略
BEGIN clock_generator
PARAMETER INSTANCE = clock_generator_0
PARAMETER C_EXT_RESET_HIGH = 1

PARAMETER C_CLKIN_FREQ =   100000000

PARAMETER C_CLKOUT0_FREQ = 100000000
PARAMETER C_CLKOUT0_PHASE = 0
PARAMETER C_CLKOUT0_GROUP = DCM0
PARAMETER C_CLKOUT0_BUF = TRUE
PARAMETER HW_VER = 3.02.a
PORT CLKIN = dcm_clk_s
# 後略
```

図 11.26 SZ410 の MHS ファイル編集(system.mhs)

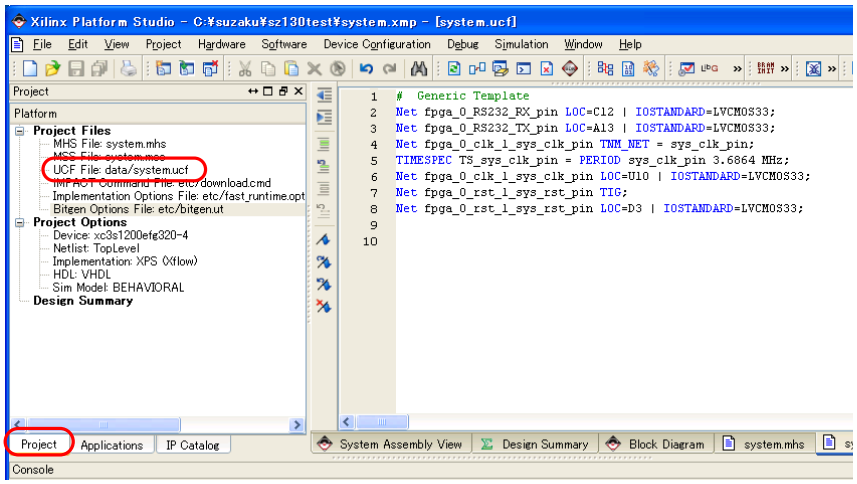
11.3.4. ピンアサインの設定

[Project]タブをクリックし、[UCF File: data/system.ucf]をダブルクリックして開いてください。ピンアサインを設定します。sys_clk_pin、sys_rst_pin、fpga_0_RS232_RX_pin、fpga_0_RS232_TX_pin をピンアサインしてください。それぞれコメントアウトした記述があると思います。それぞれのピンアサインの後ろに | IOSTANDARD = LVCMOS33 の記述も追記して下さい。SZ130 の場合、TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 3.6864 MHz;SZ410 の場合

TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100 MHz;に修正して下さい。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 ピンアサイン(system.ucf)

	SZ130	SZ410
sys_clk_pin	U10	Y6
sys_rst_pin	D3	U3
fpga_0_RS232_RX_pin	C12	Y4
fpga_0_RS232_TX_pin	A13	U4




```
#  Generic Template

Net fpga_0_RS232_RX_pin LOC=C12 | IOSTANDARD=LVCMOS33;
Net fpga_0_RS232_TX_pin LOC=A13 | IOSTANDARD=LVCMOS33;

Net fpga_0_clk_1_sys_clk_pin TNM_NET = sys_clk_pin;

TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 3.6864 MHz; # SZ410 の場合は 100 MHz
Net fpga_0_clk_1_sys_clk_pin LOC=U10 | IOSTANDARD=LVCMOS33;

Net fpga_0_rst_1_sys_rst_pin TIG;

Net fpga_0_rst_1_sys_rst_pin LOC=D3 | IOSTANDARD=LVCMOS33;
```

図 11.27 SZ130 の場合のピンアサイン(system.ucf)

11.4. アプリケーション編集

すでにサンプルアプリケーションができあがっているので、少し編集して Hello SUZAKU と表示するようにします。

[Applications] タブをクリックし、[Project: TestApp_Memory_microblaze] → [Sources] → [TestApp_Memory.c]をダブルクリックして開いてください。

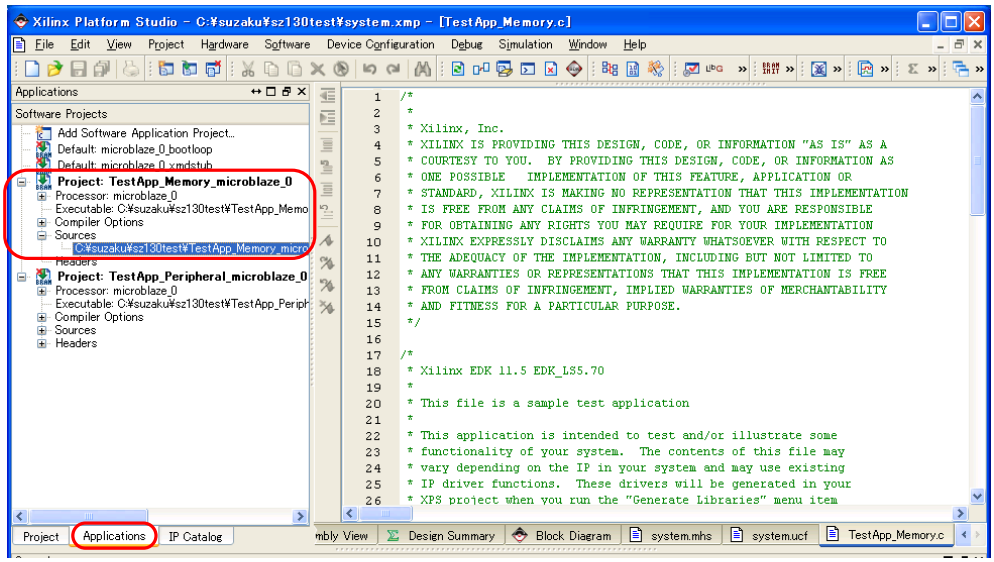
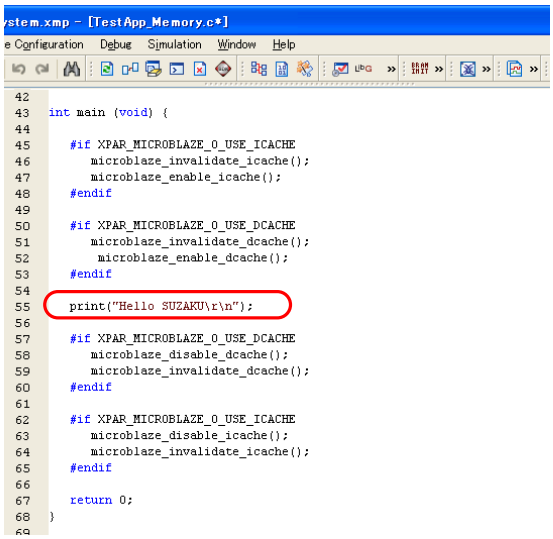


図 11.28 TestApp_Memory.c を開く

Hello SUZAKU だけ表示したいと思うので、2 箇所ある print 文の 1 箇所を Hello SUZAKU を表示するように変更し、もう一箇所は削除します。変更できたら[File]→[Save]をクリックし、保存してください。




```
-- 前略
int main (void) {
 /*
  * Enable and initialize cache
  */
 #if XPAR_MICROBLAZE_0_USE_ICACHE
    microblaze_invalidate_icache();
    microblaze_enable_icache();
 #endif

 #if XPAR_MICROBLAZE_0_USE_DCACHE
    microblaze_invalidate_dcache();
     microblaze_enable_dcache();
 #endif

//   print("-- Entering main() --\r\n");
 print("Hello SUZAKU\r\n");

-- 中略

//   print("-- Exiting main() --\r\n");

 return 0;
}
```

図 11.29 Hello SUZAKU のソースコード(main.c)

11.5. プログラムファイル作成

今回は一つずつ丁寧に作業を行なわずに、BSB を信じて一気にプログラムファイルを作成してしまいます。[Device Configuration]→[Update Bitstream] をクリックしてください。論理合成、インプリメンテーションなどが行なわれた後、bit ファイルが生成されます。残念ながら、エラーが出た場合はエラーログを確認してこれまでの作業を見直して下さい。

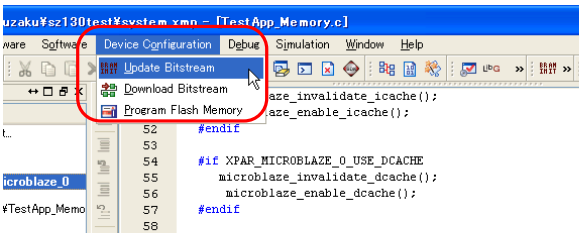
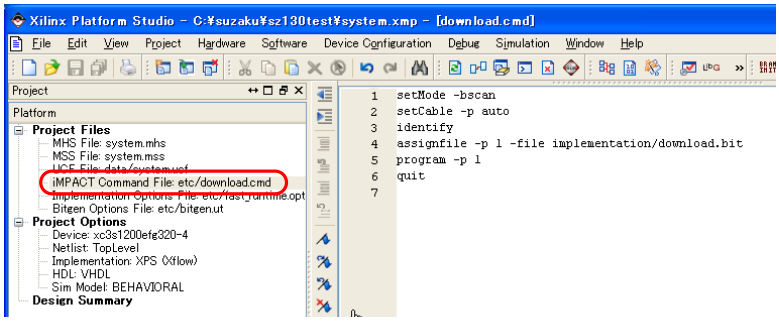


図 11.30 bit ファイル作成

11.6. コンフィギュレーション

今回はバッチモードの iMPACT を使用するため、iMPACT のコマンドを修正します。[Project]タブをクリックし、[iMPACT Command File: etc/download.cmd]をダブルクリックして開いてください。5 となっているところが 2 箇所あるので、1 に変更して下さい。




```
setMode -bscan
setCable -p auto
identify
assignfile -p 1 -file implementation/download.bit
program -p 1
quit
```

図 11.31 download.cmd の変更

シリアル通信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(「5.1. シリアル通信ソフトウェアの起動」参照)。設定できたら SUZAKU JP2 をショートし、SUZAKU CON7 にダウンロードケーブル、SUZAKU CON1 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。

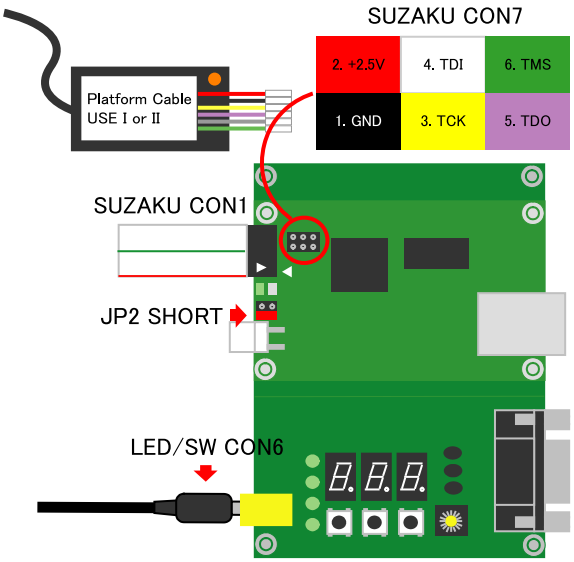


図 11.32 ジャンパの設定等

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

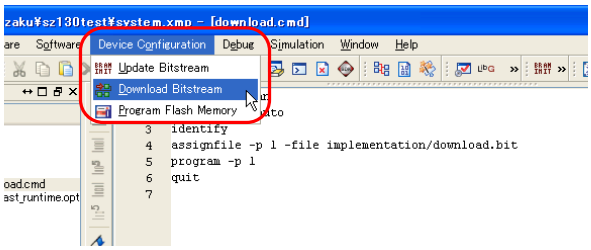


図 11.33 bit ファイル書き込み

シリアル通信ソフトウェアに Hello SUZAKU と表示されたら成功です。

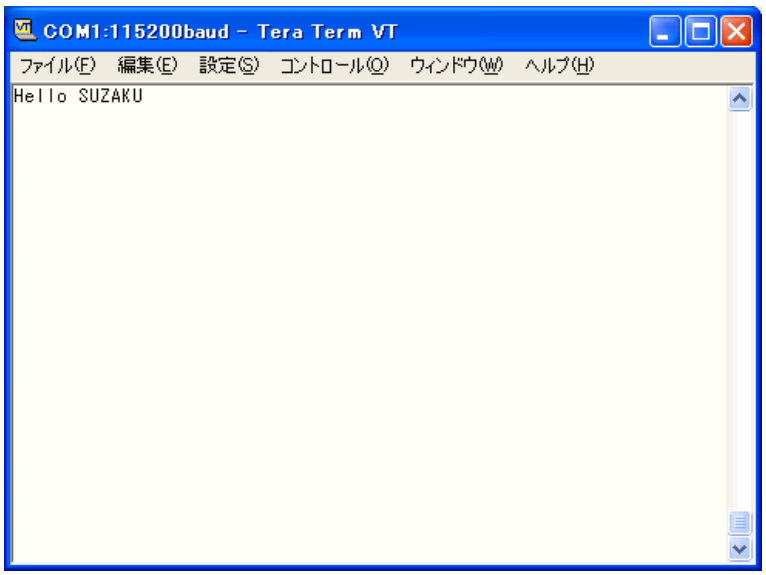


図 11.34 書き込み成功例

何もないところから作ることを考えると、簡単にプロセッサを動かすことが出来ました。EDK の感触はつかめたでしょうか。

今作った Hello SUZAKU を踏まえて次の SUZAKU のデフォルトを見てみてください。

フラッシュメモリに書き込む bit ファイル

現在の設定のままですとフラッシュメモリに書き込める bit ファイルを生成することが出来ません。フラッシュメモリに書き込む bit ファイルを作る場合は `bitgen.ut` ファイルを開き、`-g StartUpCLK:JTAGCLK -g StartUpClk:CCLK` に変更する必要があります。

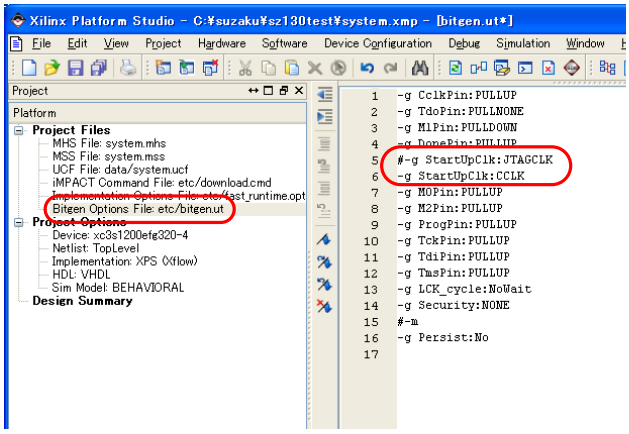


図 11.35 bitgen.ut の変更

12.SUZAKU のカスタマイズ

SUZAKU のデフォルトの FPGA プロジェクトは必要最低限の IP コアで構成されています。自由にカスタマイズしてこその SUZAKU なので、あえて必要最低限の IP コアだけにしています。

スロットマシンの IP コアは SUZAKU のデフォルトに接続するので、ここでは SUZAKU のデフォルトについて説明します。SUZAKU のデフォルトについて説明した後、SUZAKU の一番基本となるカスタマイズ

- GPIO の追加
- UART の追加

について説明します。

12.1. SUZAKU のデフォルト

SUZAKU のデフォルトを説明します。

付属 CD-ROM の"\suzaku\fpga_proj\x.x\sz***\sz***-yyyymmdd.zip"をハードディスクに展開してください。SUZAKU 公式サイト [http://suzaku.atmark-techno.com/series/stk/download] よりダウンロードすることも出来ます。ここでは展開後のフォルダを"C:\suzaku"の下にコピーします。XPS を起動してください。起動したら[File]→[Open Project...]をクリックし、"C:\suzaku\sz***-yyyymmdd\xps_proj.xmp"を開いてください。SUZAKU のデフォルトが開きます。

12.1.1. SZ130 の構成

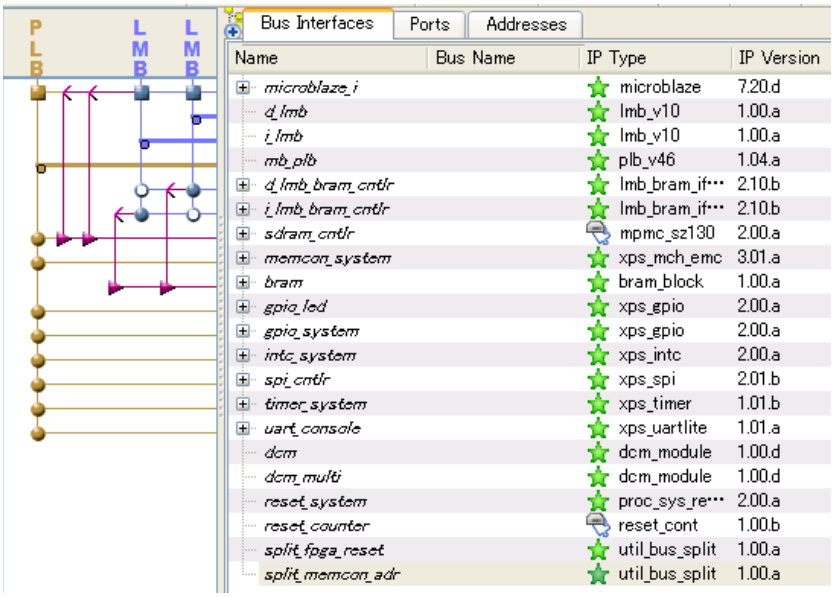


図 12.1 SZ130 のデフォルト(XPS)

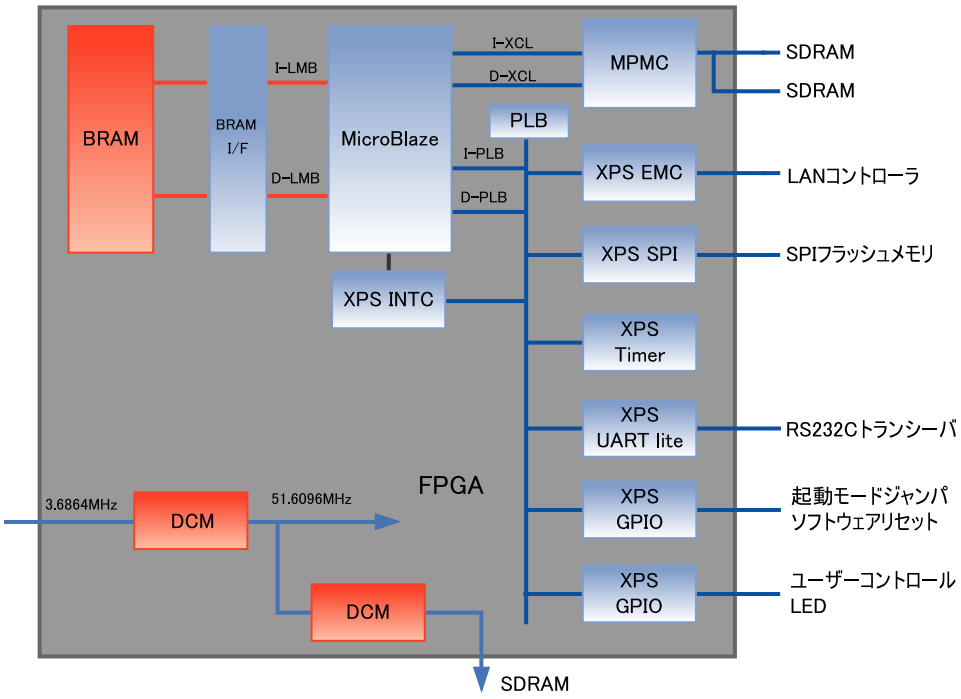


図 12.2 SZ130 デフォルトのブロック図

12.1.2. SZ410 の構成

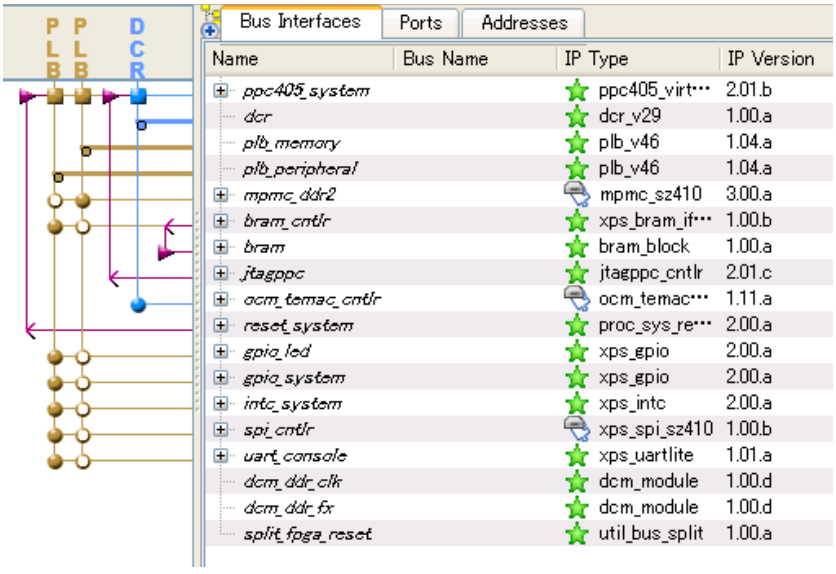


図 12.3 SZ410 のデフォルト(XPS)

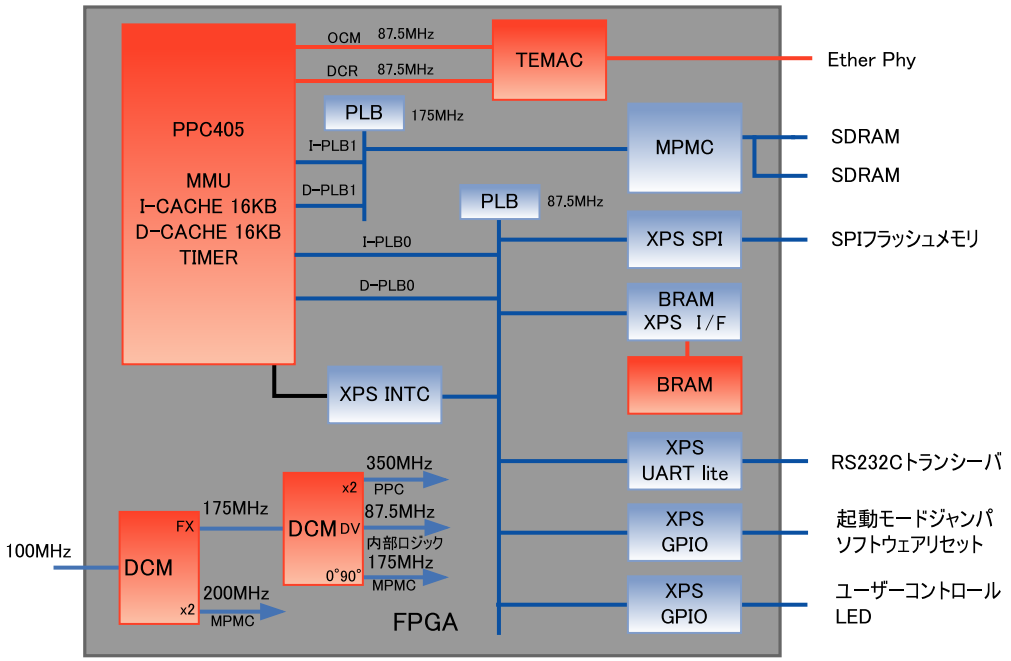


図 12.4 SZ410 デフォルトのブロック図

12.1.3. IP コア

SUZAKU の FPGA で使用している IP コアについて説明します。SUZAKU はプロセッサ、バス、FPGA 内部メモリ、外部メモリコントローラ、LAN コントローラ、割り込みコントローラ、タイマー、シリアル、GPIO、クロックの IP コアを使用しています。

12.1.3.1. プロセッサ(microblaze、ppc405)

SZ130 はソフトプロセッサの MicroBlaze、SZ410 はハードプロセッサの PowerPC405 を使用しています。

12.1.3.2. バス(lmb_v10、plb_v46、dcr_v29、ocm_v10)

SZ130 は LMB v1.0 と PLB v4.6 で構成しています。LMB v1.0 は MicroBlaze と BRAM を接続する専用バスです。PLB v4.6 はシリアルやタイマなどの IP コアの接続に使用しています。

SZ410 は PLB v4.6 と DCR v2.9、OCM v1.0 で構成しています。PLB v4.6 は PowerPC405 と BRAM の接続および MPMC やシリアルなどの IP コアの接続に使用しています。DCR v2.9 は PowerPC405 と TEMAC の接続、OCM v1.0 は PowerPC405 と TEMAC の FIFO の接続に使用しています。

12.1.3.3. FPGA 内部メモリ BRAM(bram_block)

FPGA は内部メモリに BRAM を持っており、SZ130 はプロセッサのプログラムを置く容量を 8KB、SZ410 は 16KB に設定しています。これは設定できる最小の容量となっているので、適宜増やしてお使いください。増やす場合、プロセッサのプログラムを置く容量を多く設定しすぎるとキャッシュやその他ユーザが設定できる RAM や FIFO の容量が減ってしまうため、最適になるように分配して下さい。

BRAM には FPGA のコンフィギュレーション時に初期値を書き込むことが出来ます。SUZAKU では BRAM に初期値としてブートローダ BBoot を書き込んでいます。BBoot が終わり次のプログラム(ブートローダ Hermit-At や Linux)が起動した後であれば、SZ130 は最初の 32 バイト以外、SZ410 は全領域をユーザプログラムが使用することも可能となります。

BRAM とバスを接続するために SZ130 では LMB BRAM Interface Controller、SZ410 では XPS Block RAM Interface Controller というコントローラを使用しています。

12.1.3.4. 外部メモリコントローラ(xps_mch_emc、mpmc)

外部メモリコントローラとして、SZ130 では XPS MCH EMC と MPMC を使用しています。XPS MCH EMC は LAN コントローラの接続に使用し、MPMC は SDRAM との接続に使用しています。SZ410 では MPMC を DDR2 との接続に使用しています

12.1.3.5. LAN コントローラ

SZ410 ではハード的に内蔵されている LAN コントローラ、hard_temac を使用しています。

12.1.3.6. 割り込みコントローラ(xps_intc)

割り込みコントローラに XPS Interrupt Controller を使用しています。最大 32 本の割り込み入力が可能で、それぞれの入力に対し、属性(ハイレベル/ローレベル/立ち上がりエッジ/立下りエッジ)の指定が出来ます。ツールは割り込みコントローラに接続されている IP コアの割り込み線の属性を見て、割り込みの受け付け回路を最適になるように自動生成します。

12.1.3.7. タイマー(xps_timer)

SZ130 は XPS TIMER、SZ410 は PowerPC405 の内部タイマを使用しています。

12.1.3.8. シリアル(xps_uartlite)

シリアルに XPS UART lite を使用しています。XPS UART lite は送受信 16MB ずつの FIFO を持っており、送信 FIFO が空になった時と、受信 FIFO にデータが入ってきた時に割り込みを発生します。ハードウェアフロー制御信号やモデム制御信号を持っていません。

CON1(RS232C 用 10 ピンヘッダ)には、TXD、RXD、RTS、CTS をピンアサインしています。そのうち XPS UART lite で使用している信号は TXD、RXD のみで、その他の信号は未使用となっています。これらの未使用の信号は GPIO やユーザロジックを接続してフロー制御したり別の XPS UART lite を接続して、2 ポート目のシリアルとすることも可能です。

12.1.3.9. GPIO(xps_gpio)

D1、D3 の LED の点灯や JP の判別に XPS GPIO を使用しています。

12.1.3.10. クロック(dcm_module)

SZ130 は SUZAKU のクロック 3.6864MHz を DCM で 14 逓倍し 51.6096MH z にして、SDRAM や内部バス、MicroBlaze に供給しています。SZ410 は SUZAKU のクロック 100MHz を 7/4 倍および 2 倍して 175MHz および 200MHz にして MPMC に供給しています。さらに、175MHz を 2 分周して 87.5MHz にして内部バスなどに供給し、175MHz を 2 倍して 350MHz にして PowerPC に供給しています。

12.1.3.11. SPI(xps_spi、xps_spi_sz410)

SZ130 では SPI コンフィギュレーションのために XPS SPI を使用しています。SZ410 では SPI コンフィギュレーション、EEPROM とのやりとりのために XPS_SPI_SZ410 を使用しています。XPS_SPI では 2 つのデバイスにアクセスする場合、個別にクロックを設定できないため、独自の IP コアを使用しています。

全ての IP を表示するには？

XPS のデフォルトセッティングの場合、IP Catalog に Status が★PREFERRED のものしか表示されません。たまに古い IP を使いたくなる時もあります。そんな時は[Edit]→[Preferences]をクリックし、[IP Catalog and IP Catalog Dialog]を選択し、IP Catalog で[Display]にチェックをいれてください。

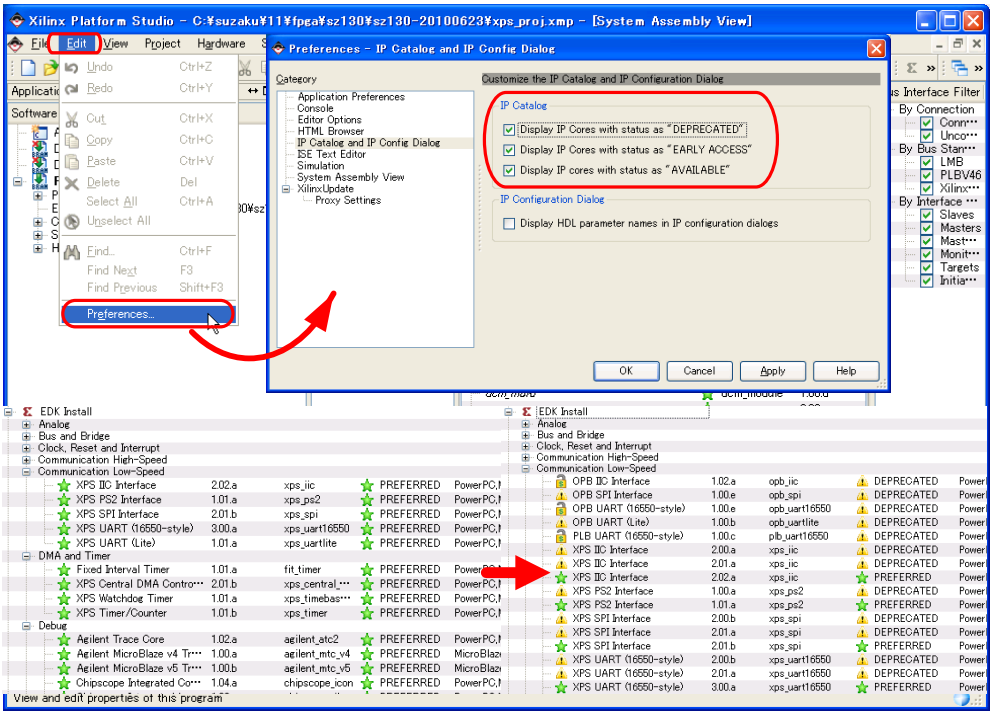


図 12.5 全 IP 表示

12.2. GPIO の追加

ISE で単色 LED(D1)を点灯させましたが、今回は SUZAKU のデフォルトに GPIO を追加して、単色 LED(D1)を点灯させます。GPIO は PLB バスに接続し、出力を単色 LED に接続します。

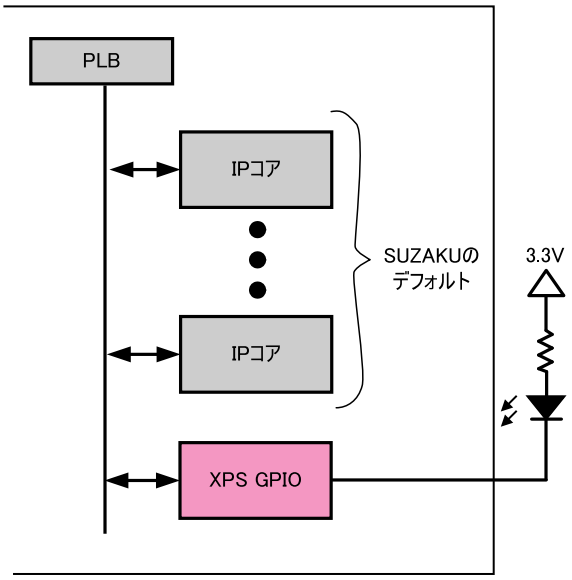


図 12.6 GPIO を追加して LED を点灯

先ほど"C:\suzaku"の下にコピーしたフォルダの名前を"sz***-add_uart_gpio"に変更して作業を進めます。

XPS を起動し、"C:\suzaku\sz***-add_uart_gpio\xps_proj.xmp"を開いてください。

12.2.1. ハードウェア設定

GUI でハードウェアの設定を行います。GUI で行ったハードウェアの設定は[Project]タブの[Project Files]→[MHS File: xps_proj.mhs]に反映されます

12.2.1.1. IP コア追加

IP コアを追加します。[IP Catalog]タブをクリックしてください。IP Catalog には XPS 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]→[XPS General Purpose IO(xps_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。IP コアが追加されます。

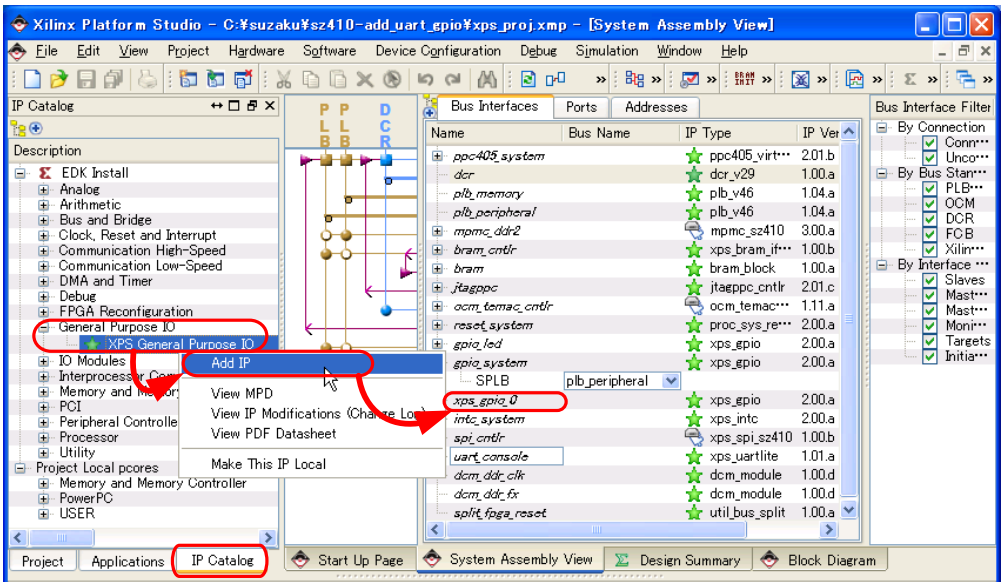


図 12.7 XPS General Purpose IO の追加

12.2.1.2. バスに接続

バスに GPIO を接続します。SZ410 の場合は plb_peripheral に接続します。[Bus Interface]タブをクリックし、追加した GPIO の横の丸をクリックしてください。○→●これでバスに GPIO が接続されます。

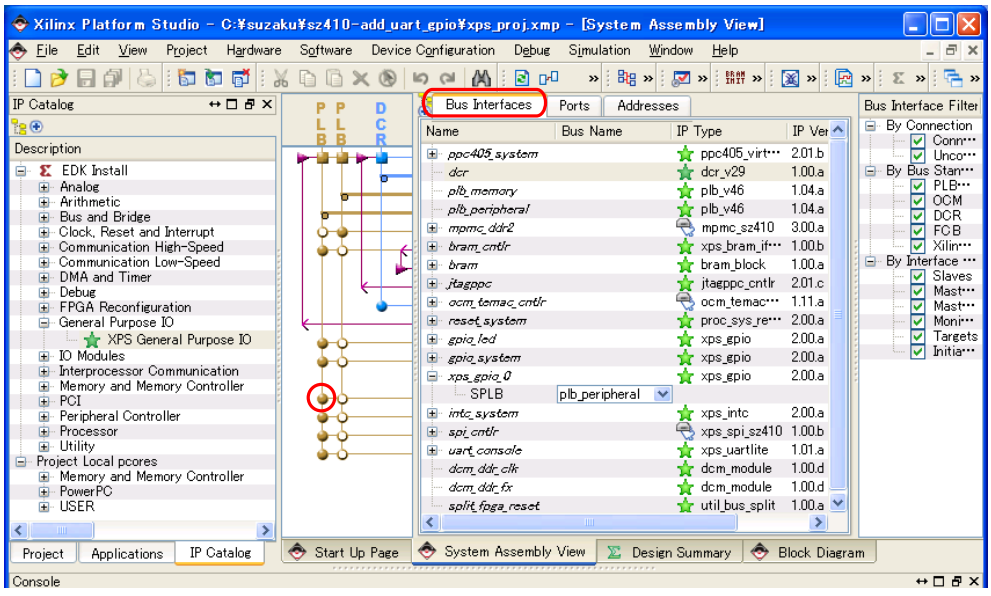


図 12.8 バスに接続

12.2.1.3. IP コアの設定

IP コアは様々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。

GPIO 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。

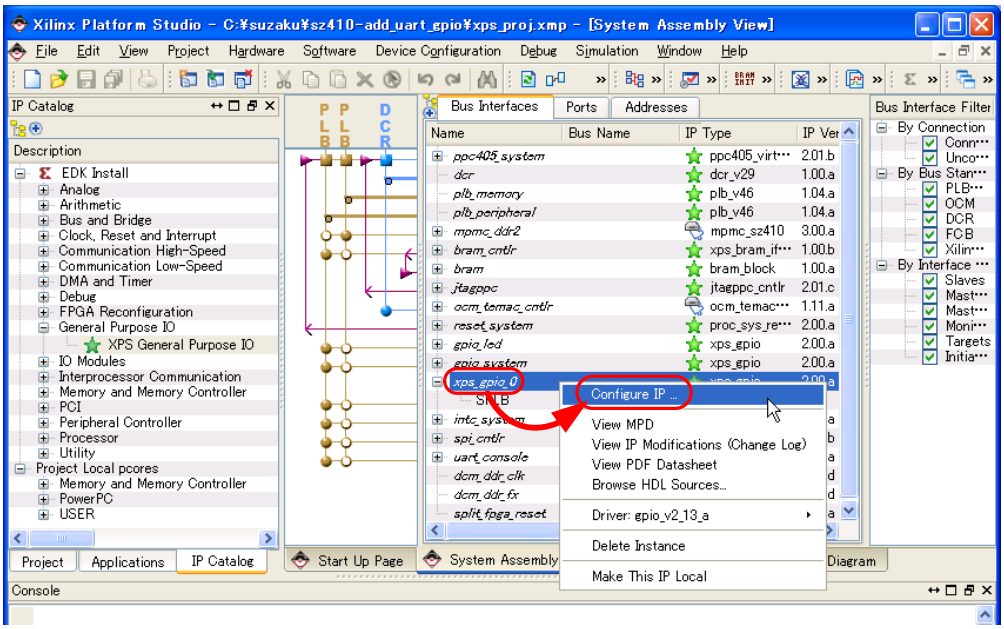


図 12.9 Configure IP

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。[GPIO Data Channel Width]を[1]に設定してください。

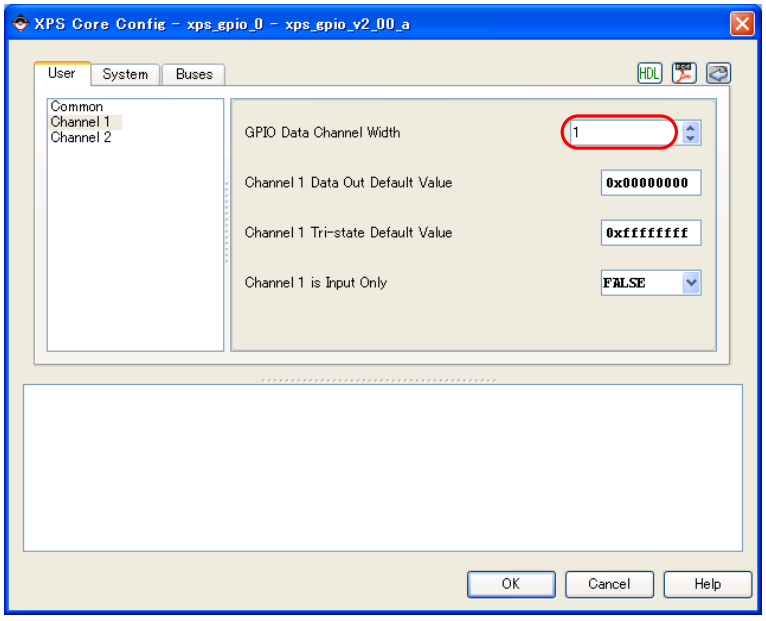


図 12.10 バス幅の設定

[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(「1.5. メモリマップ」参照)。メモリアドレスは同じバスにつながっている周辺回路を CPU が見分けるために使用する大事な値です。この値が任意に決められていることで、CPU や他のコアが通信できるようになります。

表 12.1 GPIO メモリアドレス

	SZ130	SZ410
Base Address	0xFFFFA400	0xF0FFA400
High Address	0xFFFFA5FF	0xF0FFA5FF

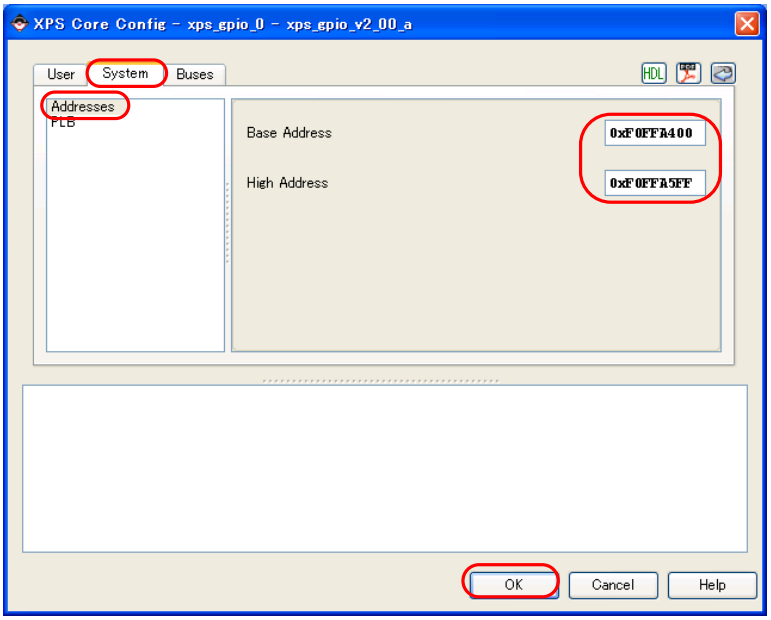


図 12.11 メモリアドレス設定

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。

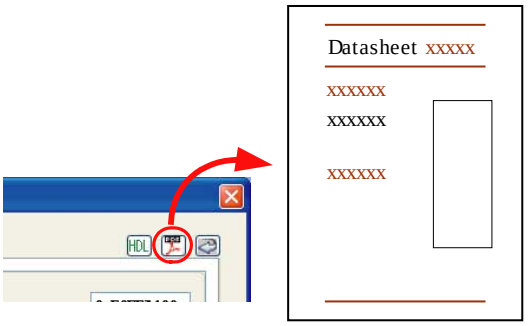


図 12.12 データシートの出し方

設定ができたら[OK]をクリックしてください。

12.2.1.4. メモリマップ確認

[Addresses]タブをクリックし、Base Address と High Address と Size に間違いがないか確認してください。

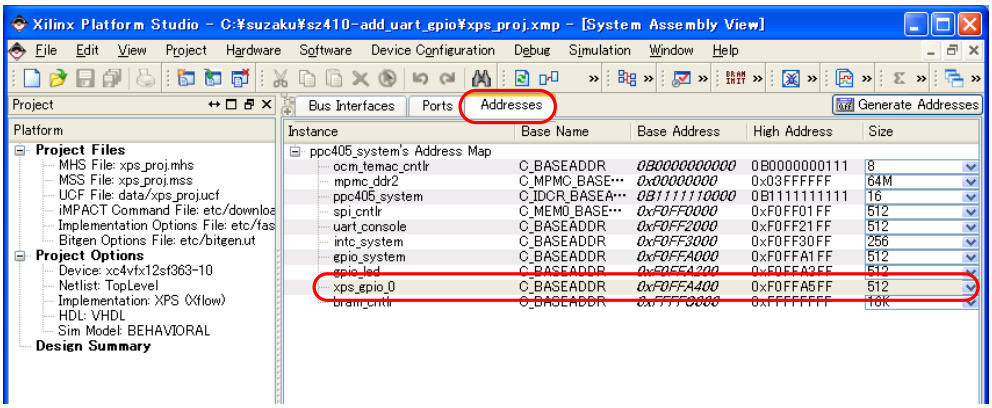


図 12.13 メモリマップ確認

12.2.1.5. 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]タブをクリックし、GPIO の⊞をクリックして開いてください。GPIO_IO_O の Net の部分をクリックし、[New Connection]を選択してください。Net が生成されます。

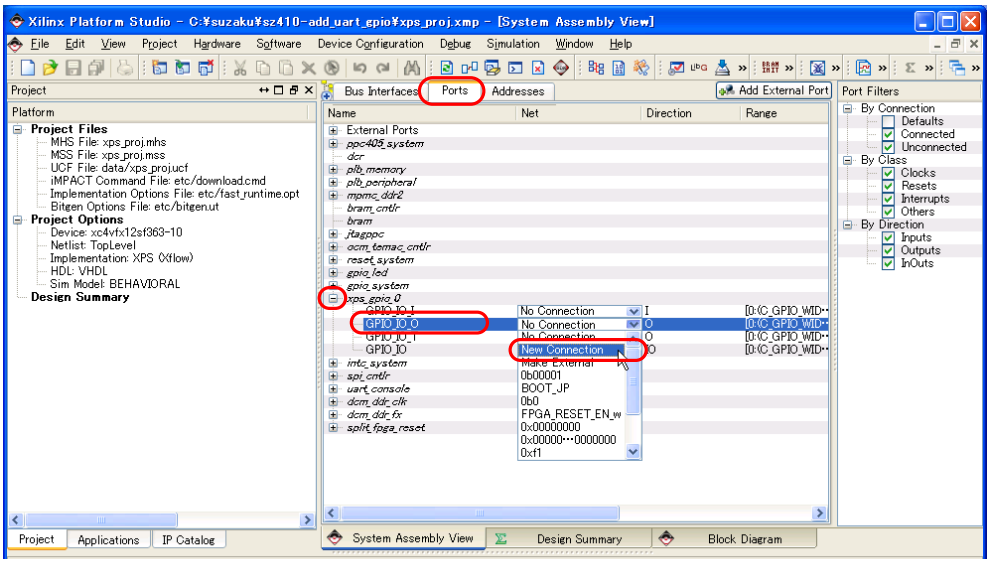


図 12.14 Net 生成

作成した信号を外部に出力したいので、もう一度 GPIO_IO_O の Net の部分をクリックし、今度は[Make External]を選択してください。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。この中に新たに Name：xps_gpio_0_GPIO_IO_O_pin という信号が生成されます。

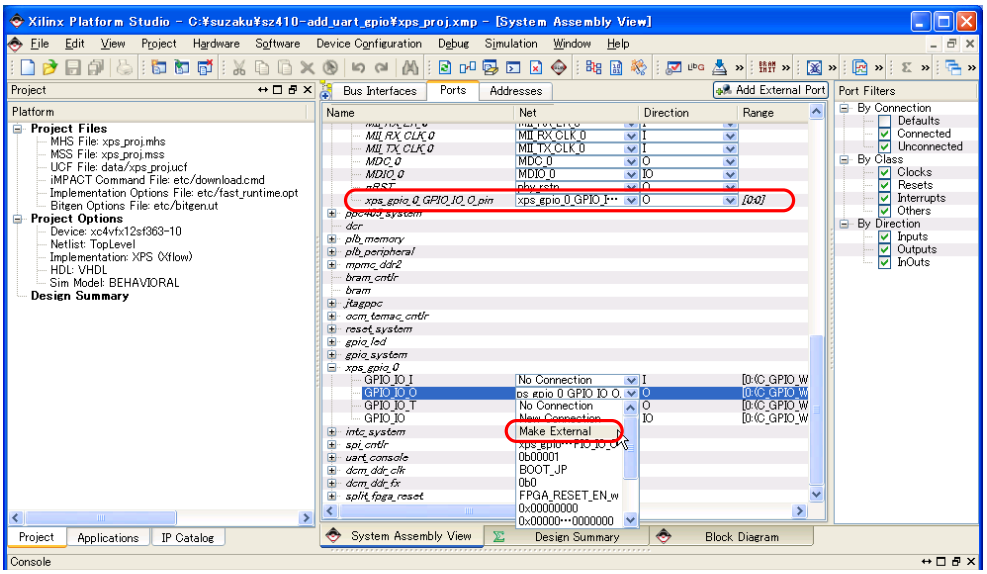


図 12.15 外部信号にする

ここまでに行った作業はすべて mhs ファイルに反映されます。mhs ファイルはハードウェアの構成や設定を記述するファイルです。[Project]タブをクリックし、[Project Files]→[MHS File: xps_proj.mhs]をダブルクリックして開いてください。外部信号の定義部分の一番下とファイルの一番下に作業が反映されています。

例 12.1 GPIO を追加した mhs ファイルの例(SZ410)

```
PARAMETER VERSION = 2.1.0

PORT SYS_RST_IN = SYS_RST_IN, DIR = I, RST_POLARITY = 1, SIGIS = RST
PORT SYS_CLK_IN = SYS_CLK_IN, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
# 中略
PORT xps_gpio_0_GPIO_IO_O_pin = nLE, DIR = O      # 外部への出力信号定義

# 中略

BEGIN xps_gpio                                    #IP コア名
 PARAMETER INSTANCE = xps_gpio_0                  # インスタンス名(Name のところ)
 PARAMETER HW_VER = 2.00.a                        # IP コアのバージョン
 PARAMETER C_GPIO_WIDTH = 1                       # 1 ビットに設定
 PARAMETER C_BASEADDR = 0xF0FFA400                # メモリマップの設定(BASE ADDRESS)
 PARAMETER C_HIGHADDR = 0xF0FFA5FF                # メモリマップの設定(HIGH ADDRESS)
 BUS_INTERFACE SPLB = plb_peripheral              # バスに接続
 PORT GPIO_IO_O = xps_gpio_0_GPIO_IO_O            # ネット名定義
END
```

12.2.1.6. ピンアサインの設定

[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。先ほど外部信号に設定した xps_gpio_0_GPIO_IO_O_pin<0>を追記してください。記述できたら[File]→[Save]をクリックし、保存してください。

表 12.2 xps_gpio_0_GPIO_IO_O_pin ピンアサイン

	SZ130	SZ410
xps_gpio_0_GPIO_IO_O_pin<0>	E12	G2

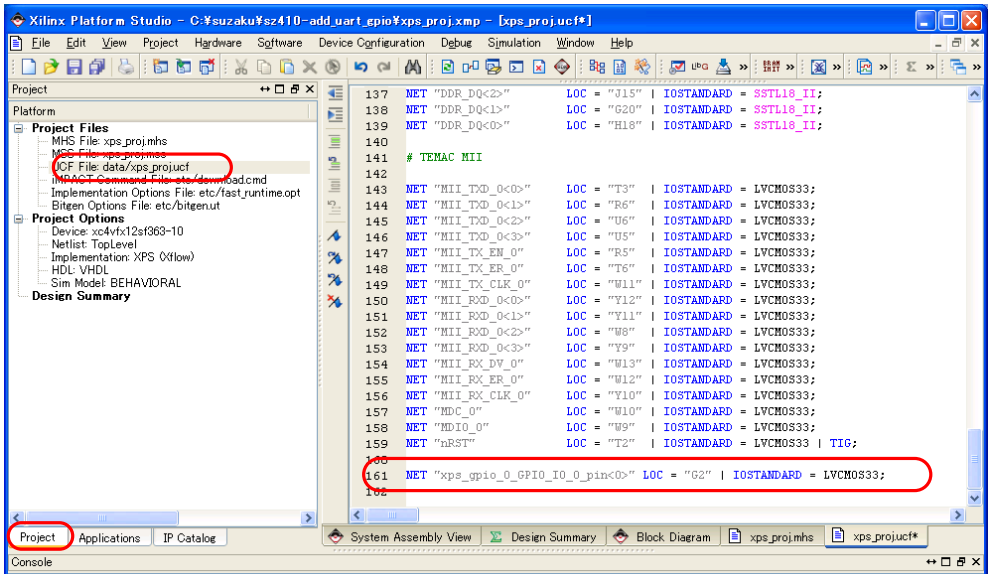


図 12.16 GPIO(xps_proj.ucf)

12.2.1.7. IP コア追加 作業まとめ

1. IP コアを SUZAKU のデフォルトに追加
2. IP コアをバスに接続
3. 追加した IP コアの設定変更
4. ピンの接続設定

12.2.2. ネットリスト作成とプログラムファイル(Hard のみ)作成

BSB での作業の時は、一気に最終 bit ファイルを作成してしまいましたが、今回は順を追って bit ファイルを作成します。[Hardware]→[Generate Netlist]をクリックしてください。ネットリストが生成されます。エラーが出た時はログをよく見てください。大体何が悪いかは、ログを見ると分かります。[1]

[1]分からない場合は、Xilinx のアンサーデータベースで検索すると、見つかることがあります。それでも分からない場合は SUZAKU のメーリングリストで質問してみてください。

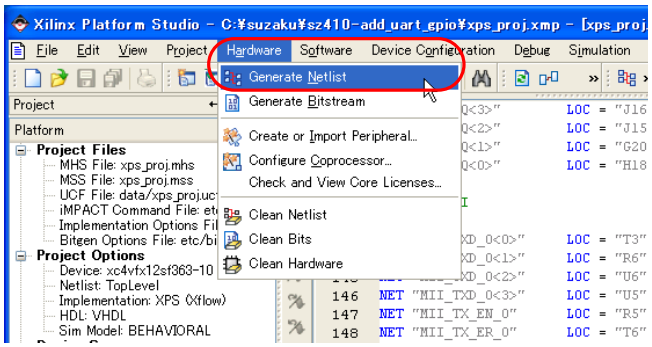


図 12.17 ネットリスト作成

[Hardware]→[Generate Bitstream]をクリックしてください。ソフトウェアを含まない bit ファイルが生成されます。

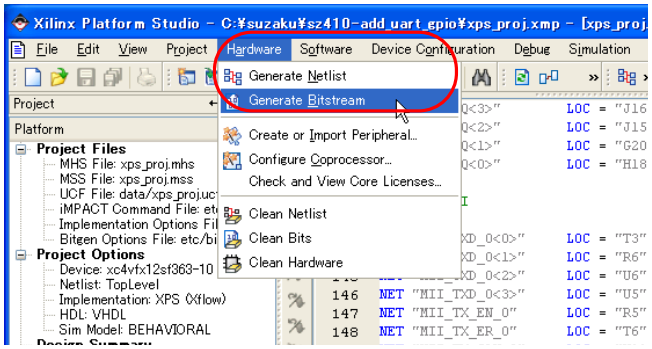


図 12.18 プログラムファイル(Hard のみ)作成

12.2.3. ソフトウェア設定

12.2.3.1. ライブラリ、ドライバ設定

[Software]→[Software Platform Settings]をクリックしてください。この機能は後のバージョンのツールでは使えなくなるようなので、注意が表示されますが[OK]をクリックしてください。ここではプロセッサ、OS、およびライブラリの設定を変更することが出来ます。[Drivers]をクリックすると、Driver の設定が表示されます。

今回は LED を点灯させたいだけなので、Driver は一番シンプルなものを選択します。追加した GPIO の Driver を[generic]に変更し、OK をクリックしてください。

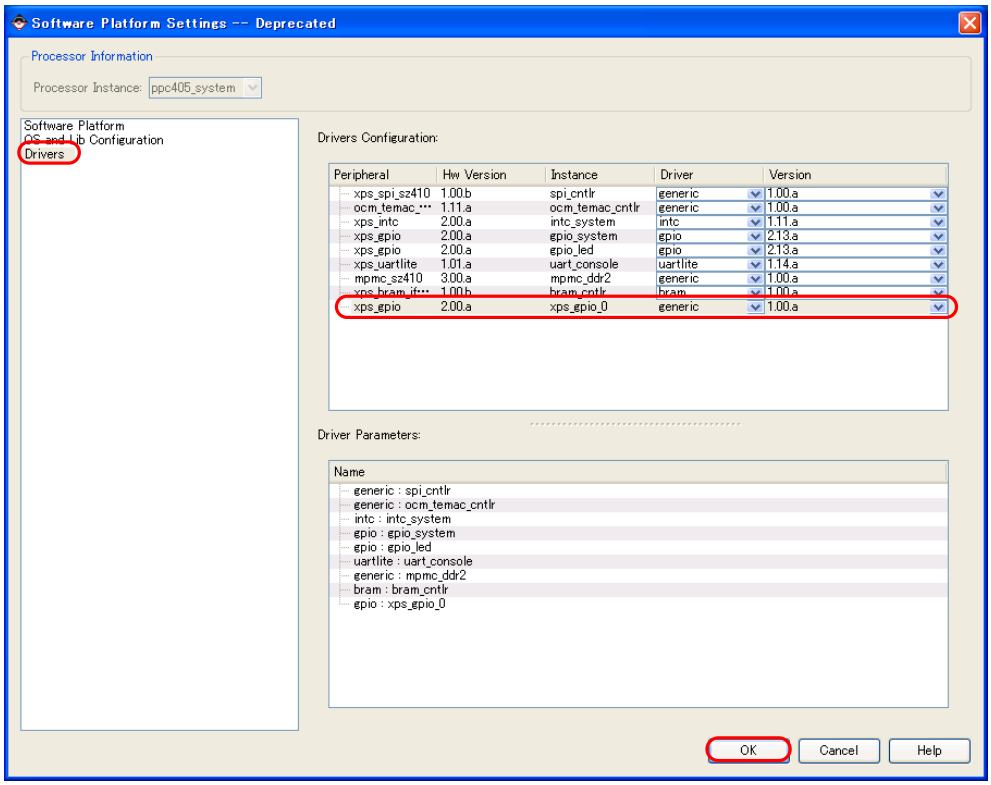


図 12.19 GPIO Driver 設定

ここで設定した内容は mss ファイルに反映されます。mss ファイルはライブラリやドライバの設定を記述するファイルです。[Project]タブをクリックし、[MSS File: xps_proj.mss]をダブルクリックして開いてください。一番下に設定した内容が反映されています。

例 12.2 GPIO の設定を追加した mss ファイルの例(SZ410)

```
BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = xps_gpio_0
END
```

12.2.3.2. ライブラリ、ドライバ生成

[Software]→[Generate Libraries and BSPs]をクリックしてください。ライブラリや様々な設定を定義したヘッダファイルが出来上がります。[Applications]タブをクリックし、[Project: BBoot]→[Processor: xxx→xparametaers.h]を開いてください。`xparameters.h` にはシステムのアドレスマップが定義されます。先ほど設定した GPIO の BASEADDR と HIGHADDR が自動で定義されています。後ほど BASEADDR の定義を使います。

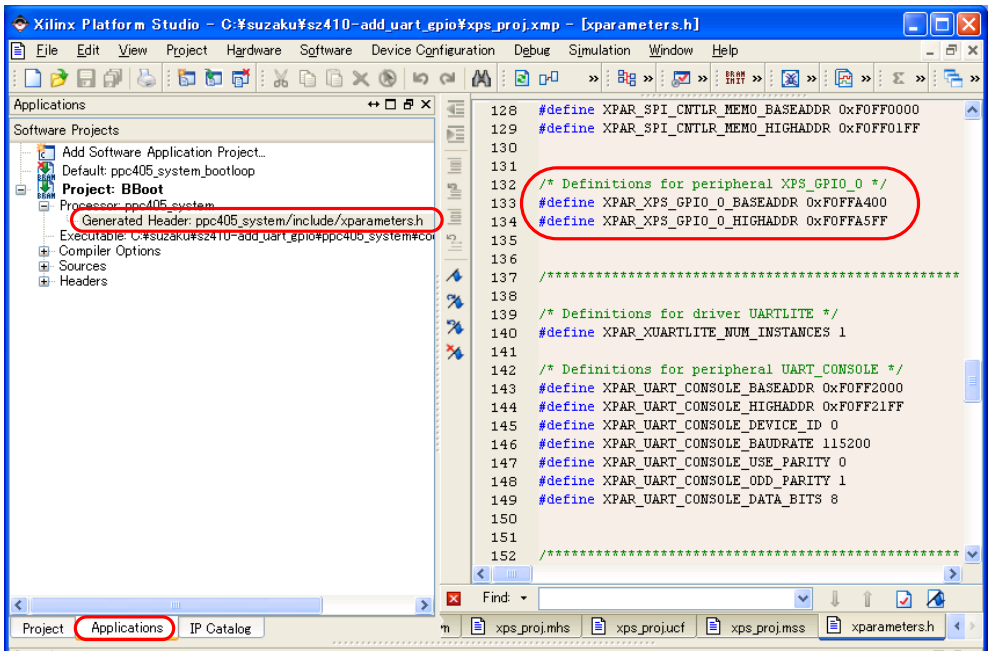


図 12.20 xparameters.h

12.2.4. アプリケーション編集

単色 LED を光らせるアプリケーションを作成します。[Applications]タブをクリックしてください。[Add Software Application Project]を右クリックしメニューを出し、[Add Software Application Project]をクリックしてください。プロジェクト名を聞かれるので、ここでは[hello-led]と入力します。入力できたら[OK]をクリックしてください。

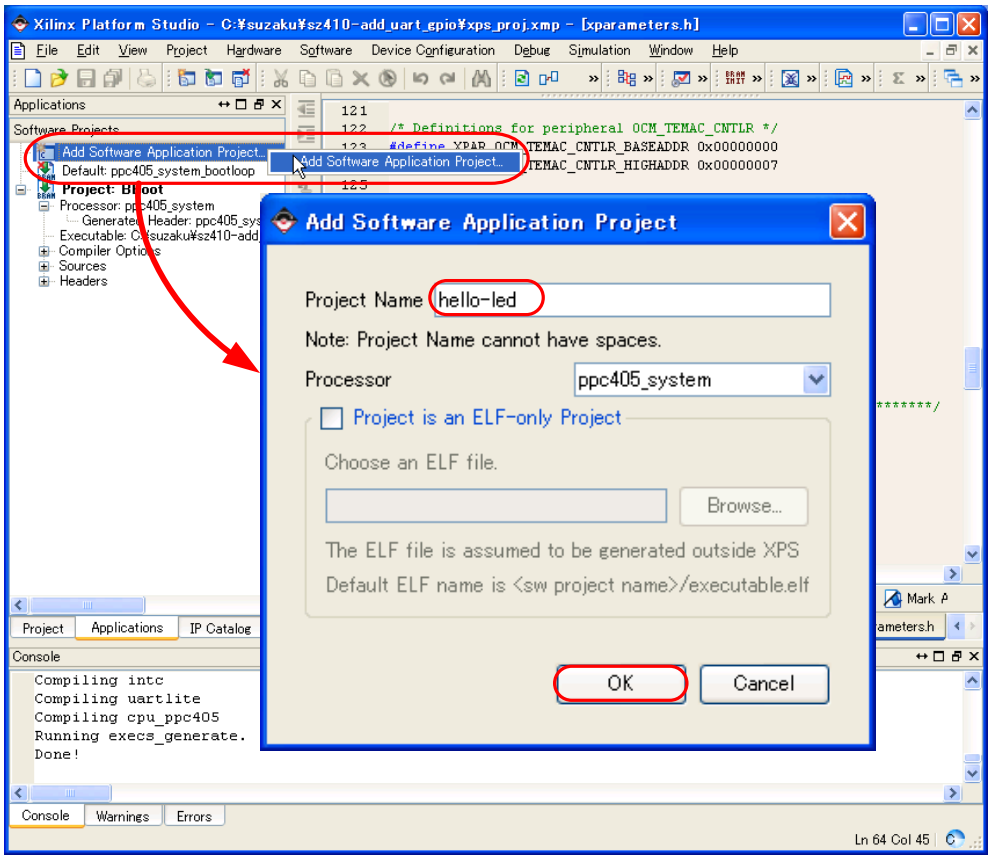


図 12.21 アプリケーション編集

[Project : hello-led]が出来上がります。[Sources]を右クリックしてメニューを出し、[Add New File]を選択してください。フォルダを作成し、作成したフォルダを開いてください。ここでは `hello-led` というフォルダを作成します。ファイル名は `main.c` とし、[保存]をクリックしてください。

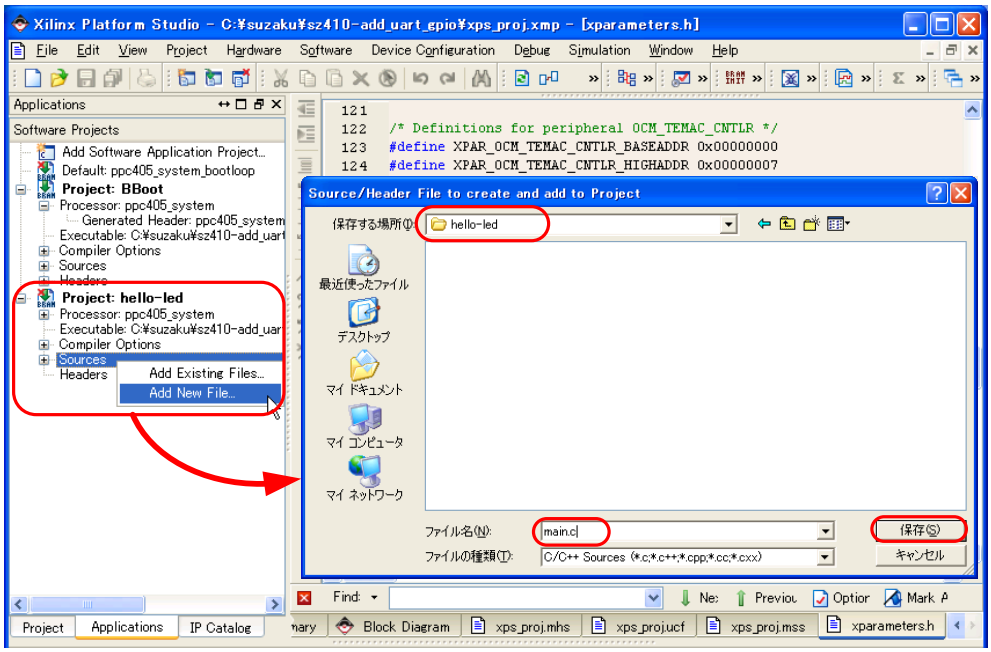
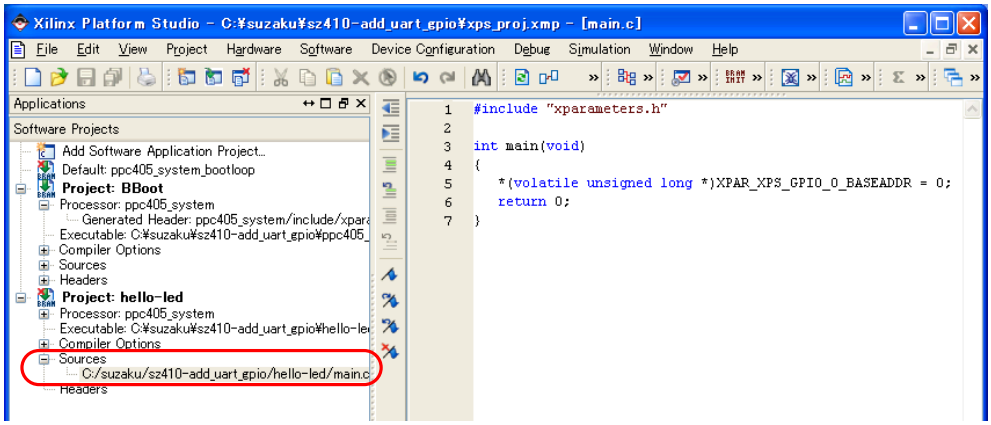


図 12.22 New File 作成

Sources に main.c が作成されます。ダブルクリックしてファイルを開いてください。

単色 LED を点灯するソースコードを記述します。XPAR_XPS_GPIO_0_BASEADDR は先ほど xparameters.h で確認した GPIO の BASEADDER です。




```
#include "xparameters.h"

int main(void)
{
  *(volatile unsigned long *)XPAR_XPS_GPIO_0_BASEADDR = 0;
  return 0;
}
```

図 12.23 単色 LED 点灯のソースコード(main.c)

記述できたら[File]→[Save]を選択し、保存してください。

[Project : hello-led]を右クリックして、[Mark to Initialize BRAMs]をクリックしてください。チェックマークがつき、[Project : hello-led]の左横のアイコンが  に変わります。これで hello-led が BRAM に初期値として書き込まれるようになります。左横のアイコンが  に変わった BBoot は書き込まれません。

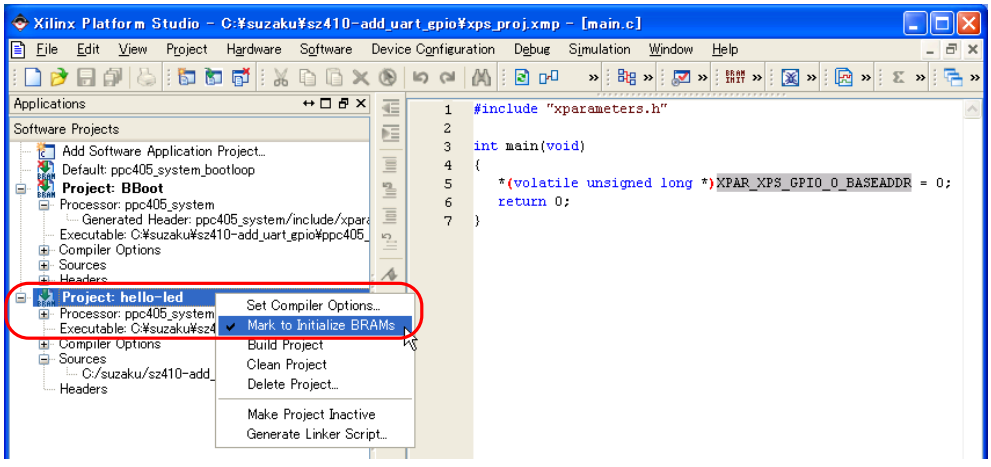


図 12.24 hello-led を書き込むように設定

12.2.4.1. リンカースクリプト設定

SZ410

SZ410 の場合リンカースクリプトの設定が必要です。[Project : hello-led]を右クリックしてメニューを出し、[Set Compiler Options]を選択してください。

[Use Default Linker Script]をチェックし、[Program Start Address]に[0xFFFFC000]と入力して[OK]をクリックしてください。0xFFFFC000 は BRAM の Base Address で、プログラムが BRAM から始まるように設定されます。

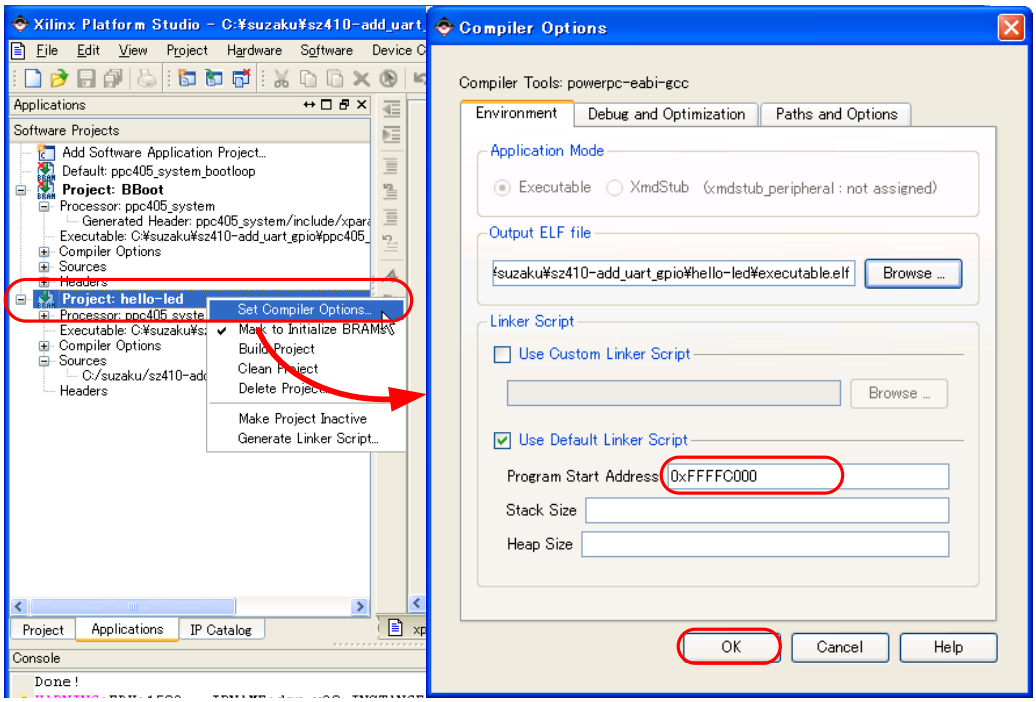


図 12.25 スタートアドレス設定

12.2.5. アプリケーション生成

[Software]→[Build All User Applications]をクリックしてください。コンパイラが起動され、アプリケーションのプログラムソースの設定が読み込まれます。エラーがなければ executable.elf が SZ130 の場合は `"C:\suzaku\sz***-add_uart_gpio\microblaze_i\code"`、SZ410 の場合は `"C:\suzaku\sz***-add_uart_gpio\ppc405_system\code"`の下に出来上がります。

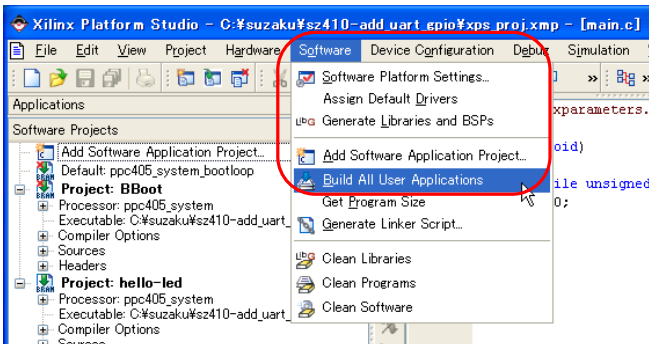


図 12.26 elf ファイル作成

12.2.6. プログラムファイル作成

ハードウェアでつくった bit ファイルの中にアプリケーションを書き込みます。[Device Configuration]→[Update Bitstream]をクリックしてください。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。

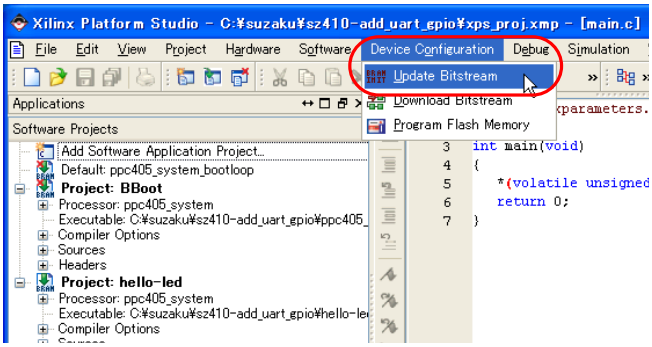


図 12.27 bit ファイル作成

変な ERROR がでたら

エラーを修正した後、[Update Bitstream]や[Build All User Application]を実行すると、どうにも理解できないエラーがでることがあります。そんな時は [Hardware] → [Clean Hardware] や [Software] → [Clean Software]を実行し、クリーンしてみてください。うまくいくことがあります。

12.2.7. 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 をショートし、SUZAKU CON7 にダウンロードケーブルを接続し、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。

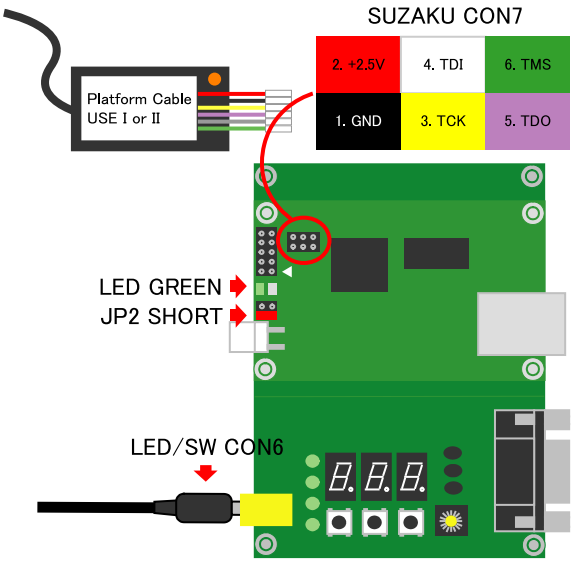


図 12.28 書き込み準備

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

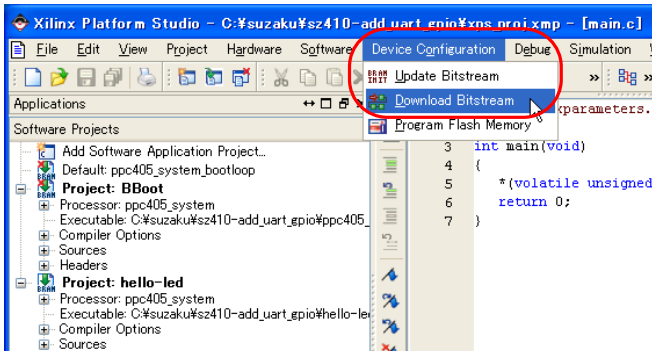


図 12.29 コンフィギュレーション

単色 LED(D1)が光ったでしょうか？

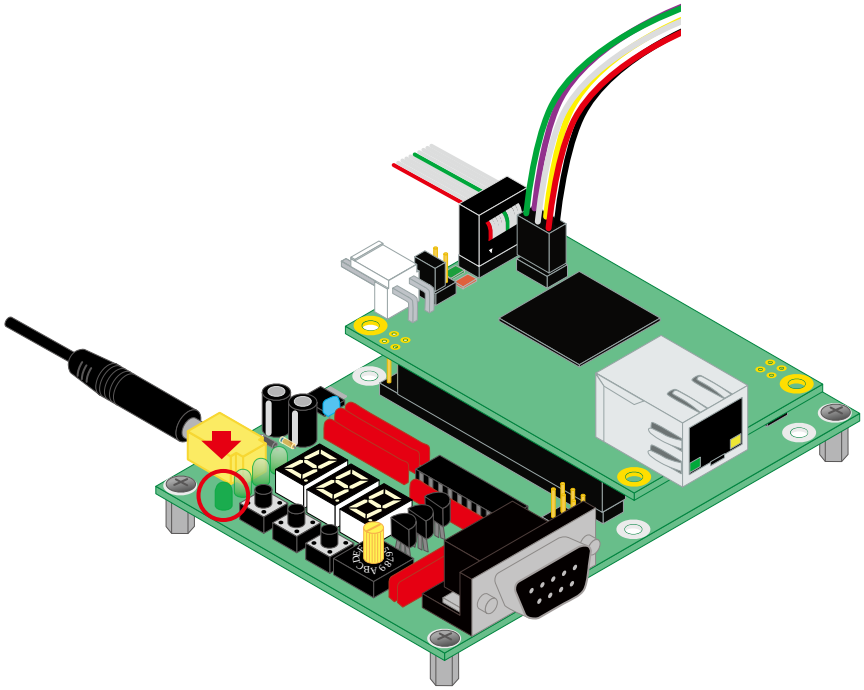


図 12.30 単色 LED(D1)点灯

出来上がった bit ファイルは C:\suzaku\sz***-add_uart_gpio\implementation\download.bit にあります。

12.2.8. 空きピン処理

ISE の時と同様、D2、D3、D4 が少し光っています(SZ410 ではほとんど光りません)。EDK で空きピン処理をしたい場合、bitgen.ut を編集します。ここに UnusedPin の設定を記述することで、終端処理を設定することが出来ます。(-g UnusedPin:PULLDOWN と追記する)デフォルトでこのピンは PullDown に設定されているため、SUZAKU のデフォルトでは明記していません。この設定を PullUp にすることで D2、D3、D4 は光らなくなります。ただ、空きピンから電圧が出力されているのはあまり良い状態ではないので、今回も D2、D3、D4 に信号を定義することで対処します。

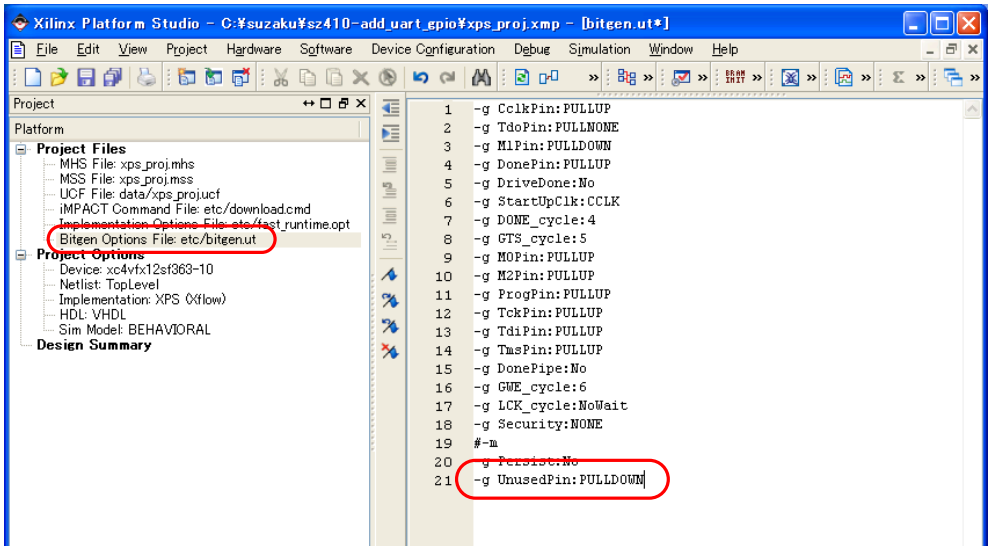


図 12.31 Bitgen のオプション設定

[Add External Port]をクリックすると、External Ports に ExternalPort_0 という名前の信号が追加されます。3 つ信号が欲しいので、[Range]のところをクリックして[2:0]と記述してください。[Direction]を[O]に設定し、[Net]に[net_vcc]と記述してください。Net に net_vcc と記述すると、1 が出力されます。

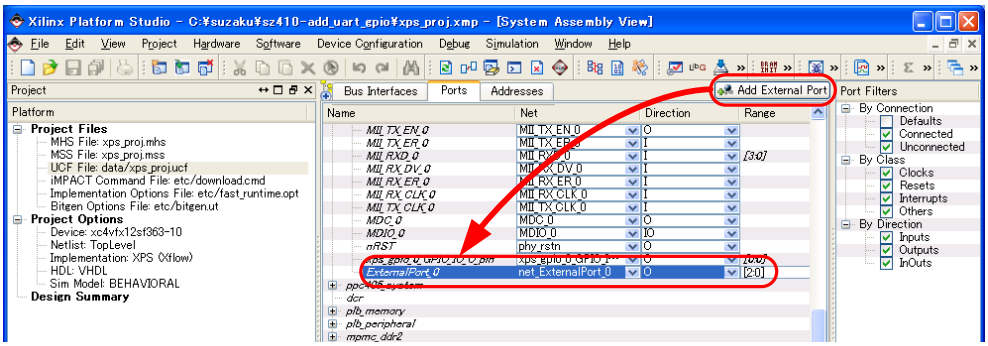


図 12.32 EDK での空きピンの処理

xps_proj.ucf を開き、ピンアサインを追記してください。

表 12.3 ピンアサイン

	SZ130	SZ410
ExternalPort_0<0>	F12	F2
ExternalPort_0<1>	B11	F1
ExternalPort_0<2>	A11	E1

12.3. UART の追加

SUZAKU のデフォルトに UART を追加して、受信した文字をそのまま送信するアプリケーションを作成します。

GPIO を追加したプロジェクトをそのままひきつづき使用します。

12.3.1. ハードウェア設定

12.3.1.1. IP コアの追加

IP コアを追加します。[IP Catalog]タブをクリックし、[Communication Low-Speed]→[XPS UART(lite)(xps_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。IP コアが追加されます。

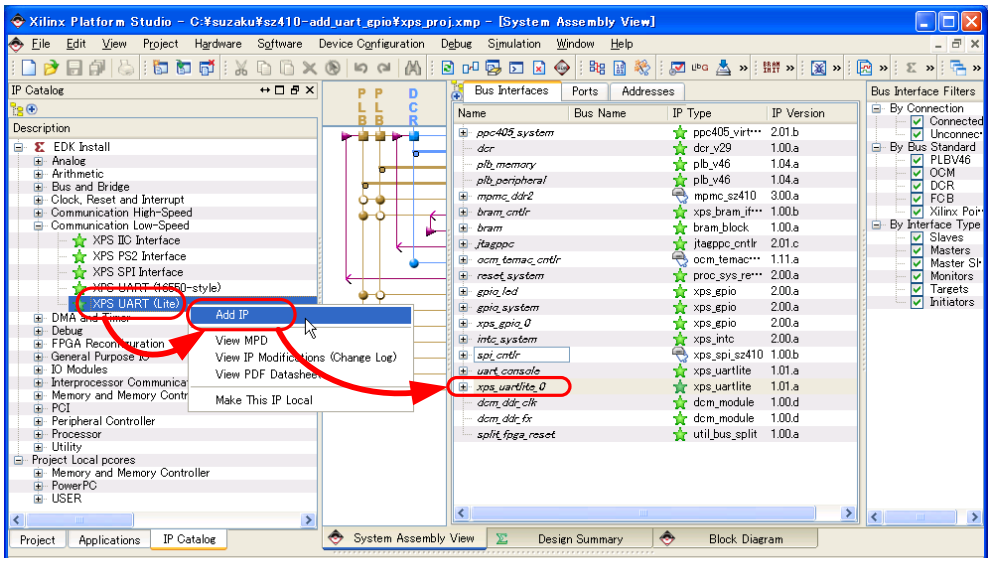


図 12.33 XPS UART(lite)の追加

12.3.1.2. バスに接続

バスに UART を接続します。SZ410 の場合は plb_peripheral に接続します。[Bus Interface]タブをクリックし、追加した UART の横の丸をクリックしてください。

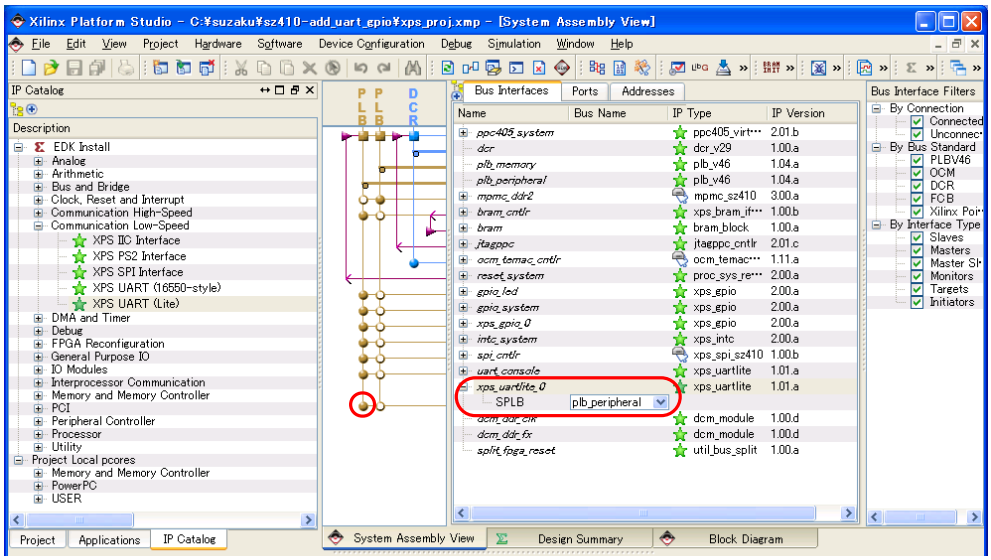


図 12.34 バスに接続

12.3.1.3. IP コアの設定

UART の設定を変更します。UART 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。

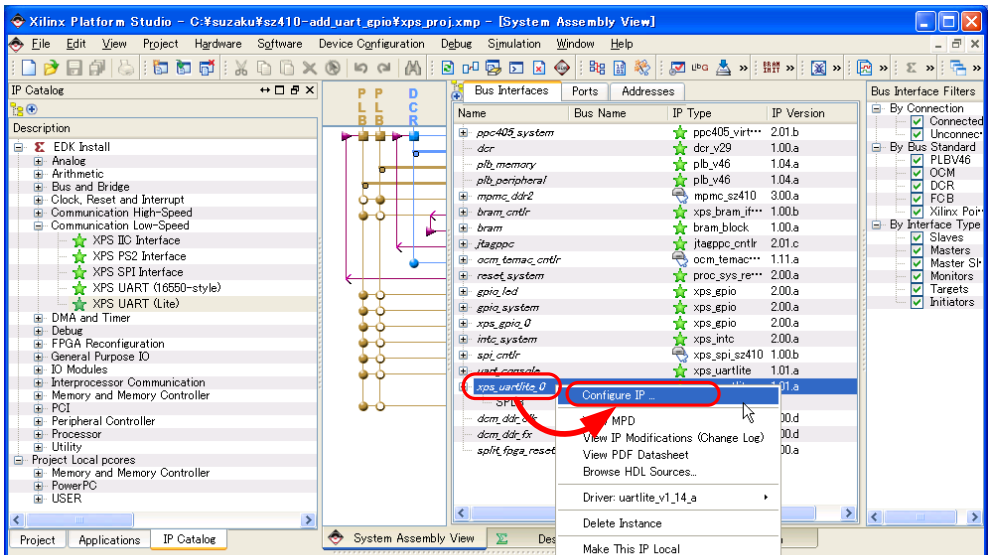


図 12.35 Configure IP

以下の設定にしてください。

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

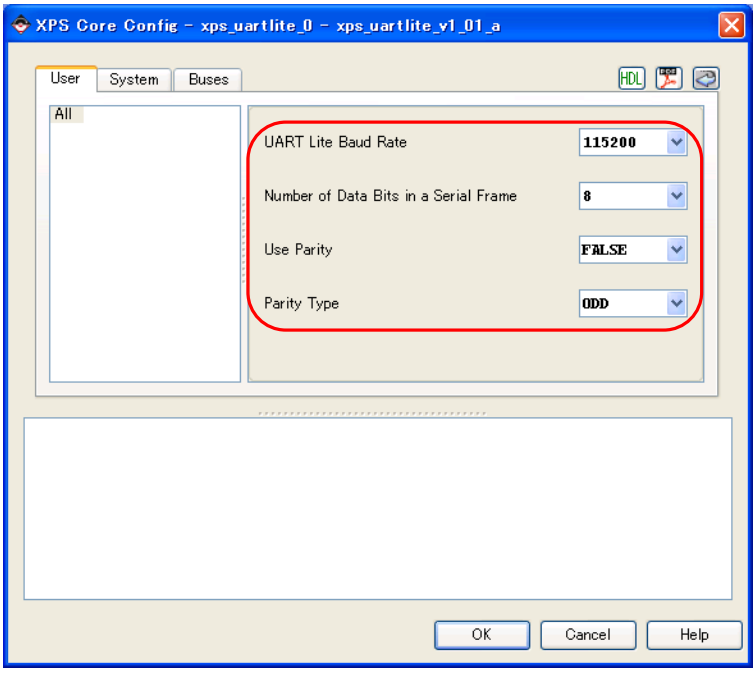


図 12.36 UART 設定変更

[System]タブをクリックし、[Base Address]、[High Address]を入力してください。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(「1.5. メモリマップ」参照)。設定できたら[OK]をクリックして下さい。

表 12.4 UART メモリアドレス

	SZ130	SZ410
Base Address	0xFFFFA600	0xF0FFA600
High Address	0xFFFFA6FF	0xF0FFA6FF

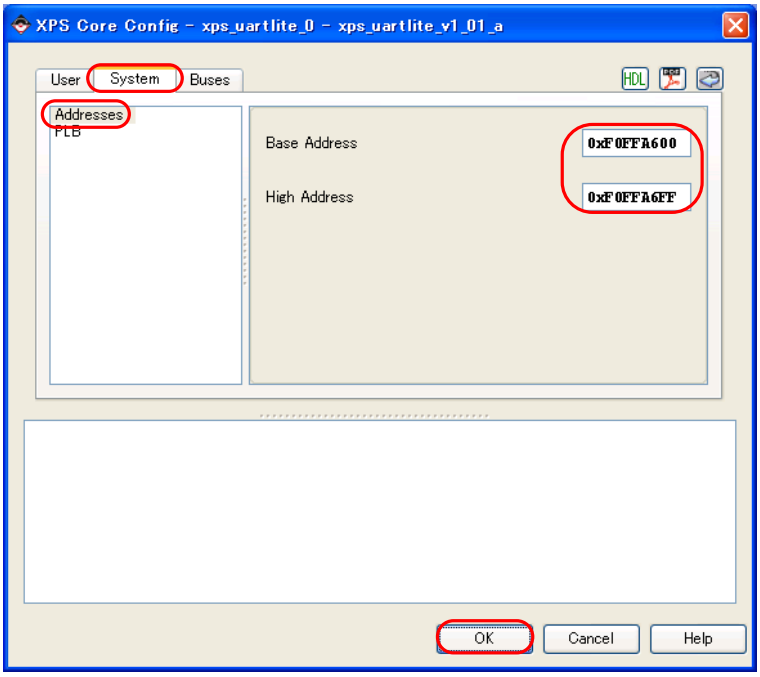


図 12.37 メモリアドレス設定

12.3.1.4. メモリマップ確認

[Addresses]タブをクリックし、Base Address と High Address と Size に間違いがないか確認してください。

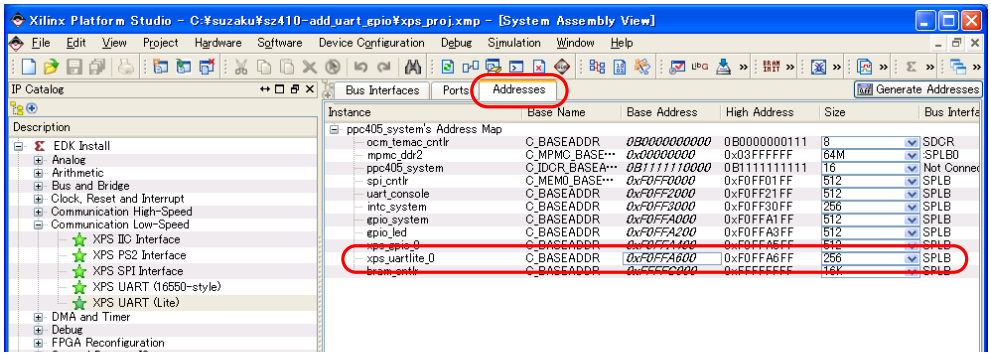


図 12.38 メモリマップ確認

12.3.1.5. 信号の定義

[Ports]タブをクリックして下さい。UART の RX と TX の Net のところで New Connection を選択してネットを生成し、Make External を選択して外部信号に設定して下さい。設定後、External Ports に信号が定義されているか確認してください。

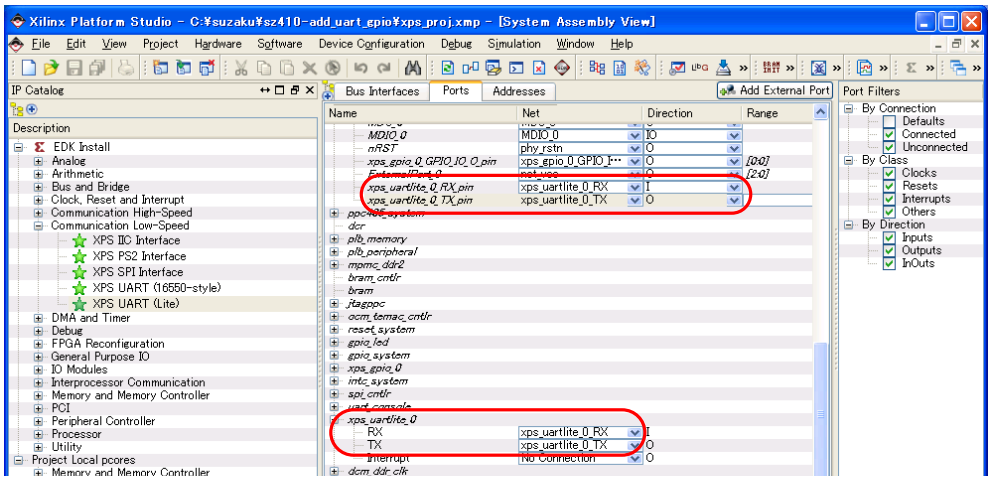


図 12.39 信号の定義

12.3.1.6. ピンアサインの設定

[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 12.5 UART ピンアサイン

	SZ130	SZ410
xps_uartlite_0_RX_pin	M3	P4
xps_uartlite_0_TX_pin	M6	E15

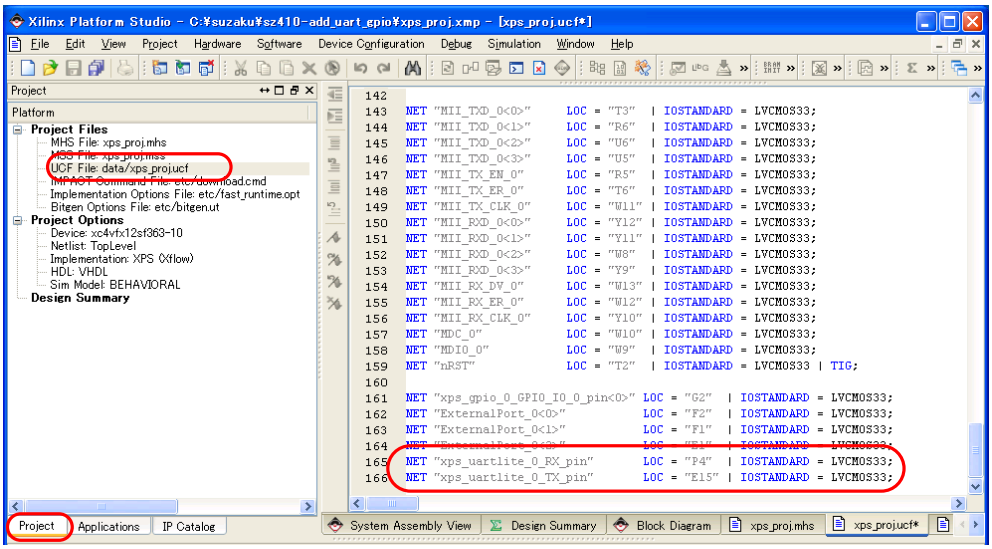


図 12.40 UART(xps_proj.ucf)

12.3.2. ネットリスト作成とプログラムファイル(Hard のみ)作成

[Hardware]→[Generate Netlist]をクリックして下さい。ネットリストが生成されます。

[Hardware]→[Generate Bitstream]をクリックして下さい。ソフトウェアを含まない bit ファイルが生成されます。

12.3.3. ソフトウェア設定

12.3.3.1. ライブラリ、ドライバ設定

[Software]→[Software Platform Settings]をクリックしてください。Driver は一番シンプルなものを選択します。追加した UART の Driver を generic に変更し、[OK]をクリックしてください。

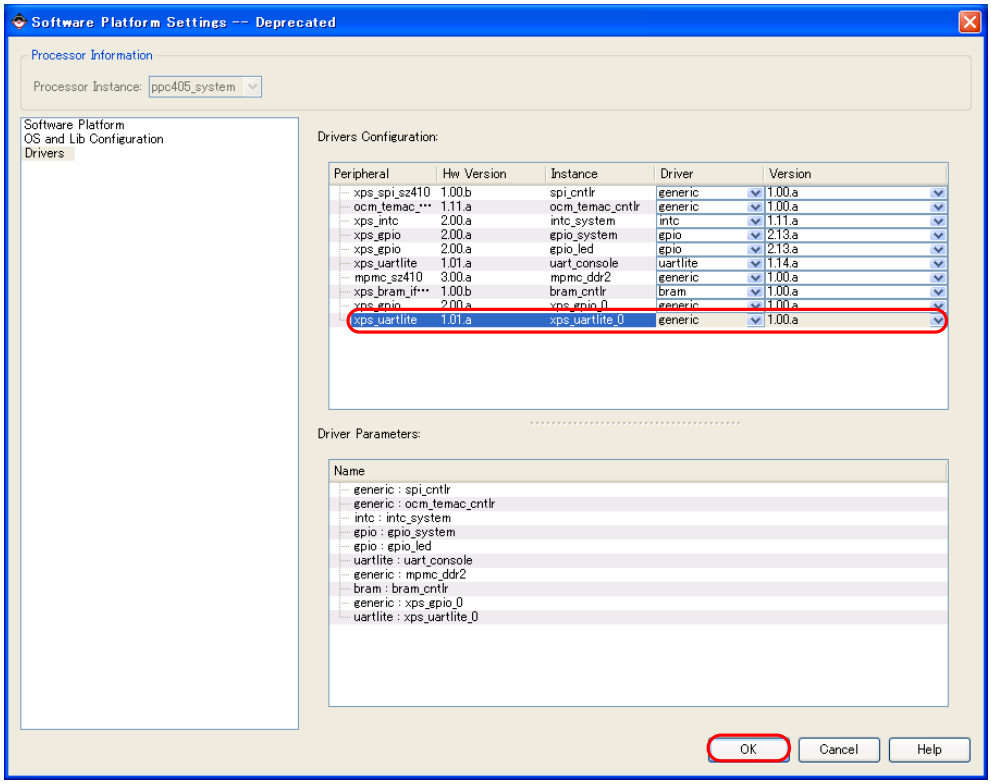


図 12.41 UART Driver 設定

12.3.3.2. ライブラリ、ドライバ生成

[Software]→[Generate Libraries and BSPs]をクリックしてください。ライブラリと様々な設定を定義したヘッダファイルが出来上がります。xparametaers.h を開いてください。先ほど設定した UART の BASEADDR と HIGHADDR も自動で定義されています。後ほど BASEADDR の定義を使います。

例 12.3 xparameters.h の定義の例

```
/* Definitions for peripheral XPS_UARTLITE_0 */
#define XPAR_XPS_UARTLITE_0_BASEADDR 0xFFFFA600
#define XPAR_XPS_UARTLITE_0_HIGHADDR 0xFFFFA6FF
```

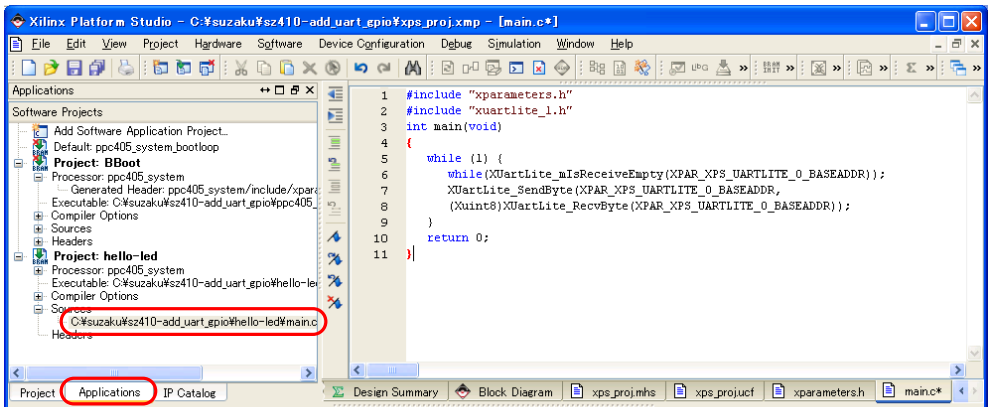
ソフトウェアに関するファイルは SZ130 の場合"C:\suzaku\sz***-add_uart_gpio\microblaze_i"、SZ410 の場合"C:\suzaku\sz***-add_uart_gpio\ppc405_system"の下に収められます。このフォルダの下の"include\xuartlite_l.h"を開いてください。UART を扱うことのできる関数等が定義されています。以下の関数を使います。

例 12.4 xuartlite_l.h に定義されている関数

```
/* 受信 FIFO のデータの有無をチェックする */
#define XUartLite_mIsReceiveEmpty(BaseAddress) \
  ((XUartLite_mGetStatusReg((BaseAddress)) & XUL_SR_RX_FIFO_VALID_DATA) != \
    XUL_SR_RX_FIFO_VALID_DATA)

/* 送信 FIFO に 1 バイト分 Write する */
void XUartLite_SendByte(Xuint32 BaseAddress, Xuint8 Data);

/* 受信 FIFO のデータを 1 バイト l 分 Read する */
Xuint8 XUartLite_RecvByte(Xuint32 BaseAddress);
```

12.3.4. アプリケーション編集

[Applications]タブをクリックし、先ほど作成した hello-led の `main.c` を開いてください。UART のヘッダファイルを追加し、先ほど書いた単色 LED を点灯する一文を消し、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。




```
#include "xparameters.h"
#include "xuartlite_l.h"
int main(void)
{
  while (1) {
    while(XUartLite_mIsReceiveEmpty(XPAR_XPS_UARTLITE_0_BASEADDR));
    XUartLite_SendByte(XPAR_XPS_UARTLITE_0_BASEADDR,
    (Xuint8)XUartLite_RecvByte(XPAR_XPS_UARTLITE_0_BASEADDR));
  }
  return 0;
}
```

図 12.42 送受信ソースコード追加(main.c)

12.3.5. アプリケーション生成

[Software]→[Build All User Applications]をクリックして下さい。コンパイラが起動され、アプリケーションのプログラムソースの設定が読み込まれます。エラーがなければ `executable.elf` が出来上がります。

12.3.6. プログラムファイル作成

ハードウェアでつくった bit ファイルの中にアプリケーションを書き込みます。[Device Configuration]→[Update Bitstream]をクリックしてください。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

12.3.7. コンフィギュレーション

シリアル通信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(「5.1. シリアル通信ソフトウェアの起動」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。

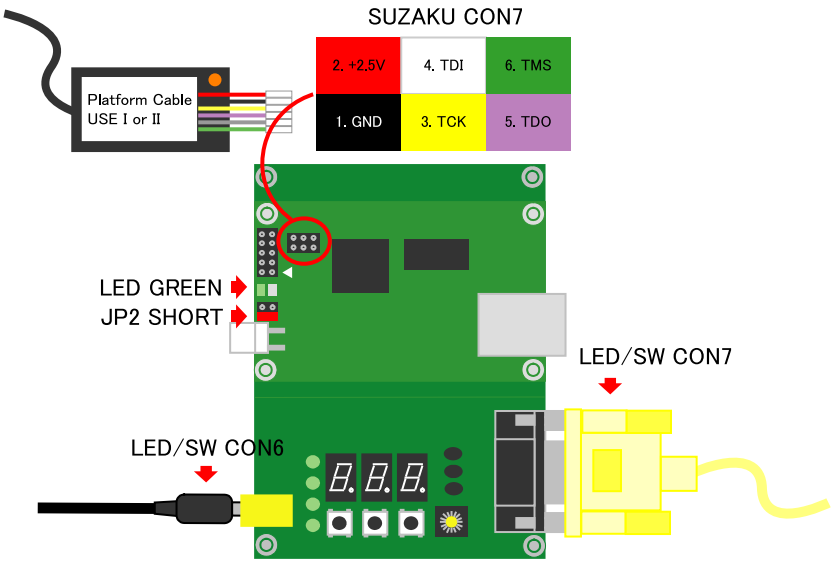


図 12.43 ジャンパの設定等

[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。

キーボードから何か文字を打ち込んでください。打ち込んだ文字がそのまま送信されてコンソールに表示されます。

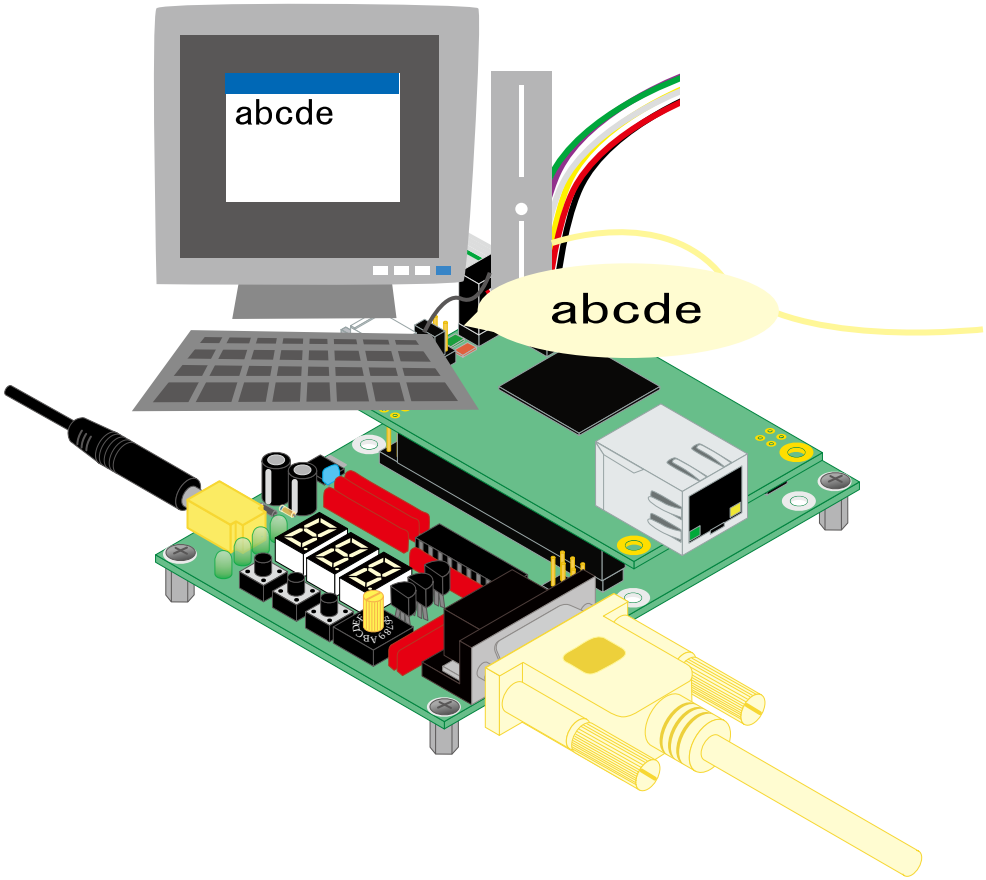


図 12.44 シリアル通信 動作確認

MPMC に注意

XPS 11.5i の GUI で MPMC を編集すると、MPMC と接続している信号の MSB と LSB が勝手にひっくり返されてしまいます。ひっくり返されてしまうと、もちろん正常動作しなくなってしまいます。毎回注意するのが嫌な場合は、ucf ファイルを編集してしまうのも一つの手です。

13.スロットマシンの製作 2(IP コアを CPU で制御する)

「10. スロットマシン製作 1(IP コアの中身を作る)」で作った回路をまとめて IP 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。

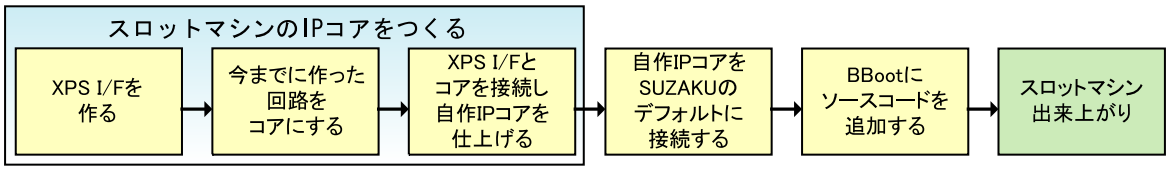


図 13.1 スロットマシンへの道のり

13.1. スロットマシンの IP コアをつくる

まずはスロットマシンの IP コアをつくります。

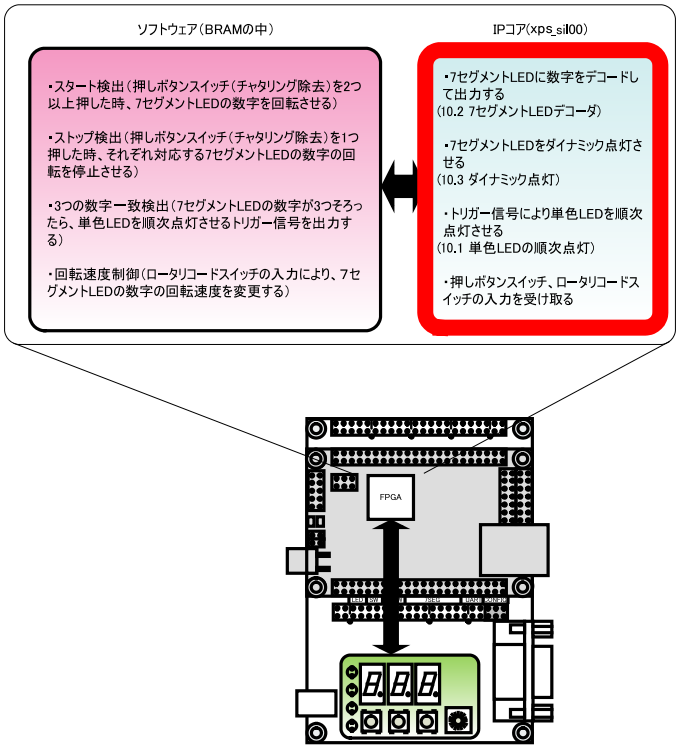


図 13.2 スロットマシンの IP コアをつくる

13.1.1. スロットマシンの IP コアの構成

スロットマシンの IP コアは"xps_sil00"と名付け、以下のような構成でつくります。

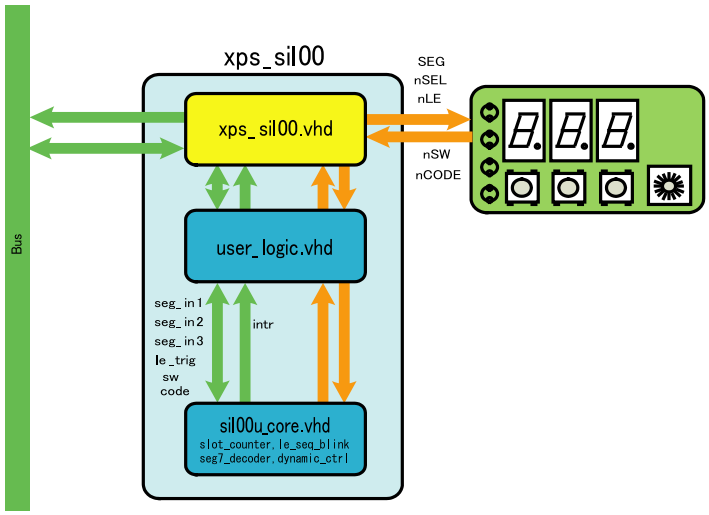


図 13.3 自作 IP コア

13.1.2. ウィザードを使って XPS インターフェースをつくる

付属 CD-ROM の"\suzaku\fpga_proj\x.x\sz***\sz***-yyyymmdd.zip"をハードディスクに展開してください。SUZAKU 公式サイト [http://suzaku.atmark-techno.com/series/stk/download] よりダウンロードすることも出来ます。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。XPS を起動し、"C:\suzaku\sz***- yyyymmdd\xps_proj.xmp"を開いてください。SUZAKU のデフォルトが開きます。

PLB バスに接続するインターフェースをつくります。XPS 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るので、簡単にインターフェースをつくることが出来ます。スロットマシンのコアを CPU から制御するためには、バスに接続しなければいけません。

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

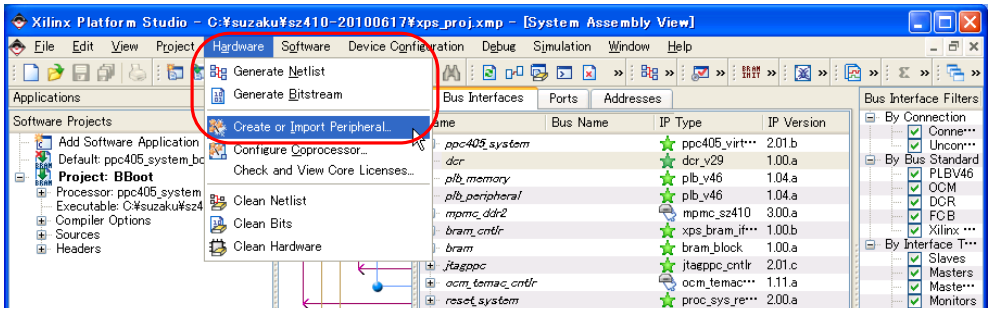


図 13.4 Create and Import Peripheral Wizard の起動のさせ方

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

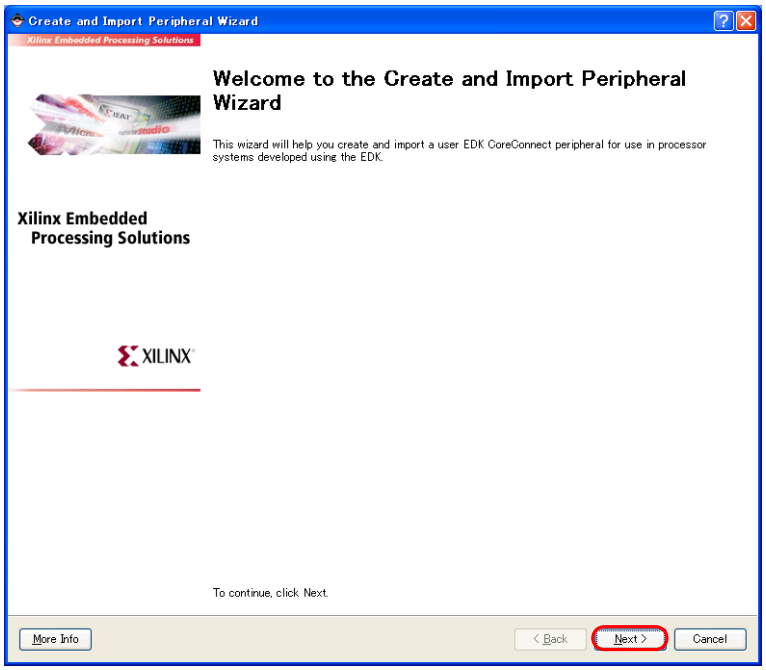


図 13.5 Create and Import Peripheral Wizard 起動画面

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 a new peripheral]をチェックして[Next]をクリックしてください。

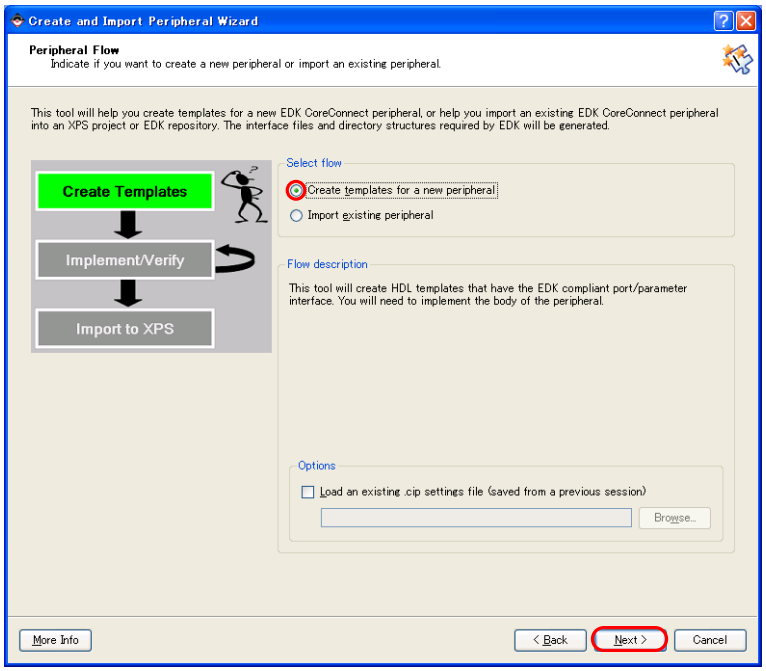


図 13.6 Peripheral Flow

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、[Next]をクリックして下さい。

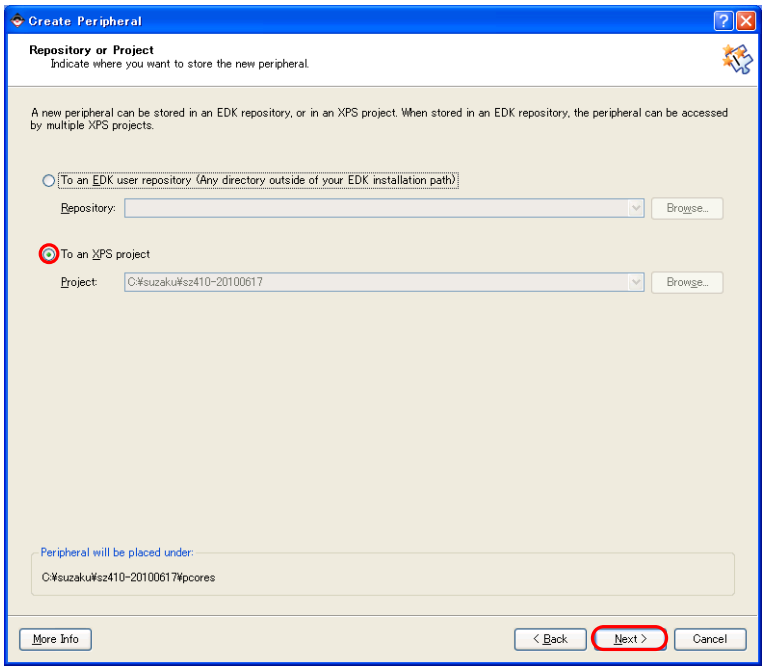


図 13.7 コアの生成場所の指定

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[xps_sil00]とします。

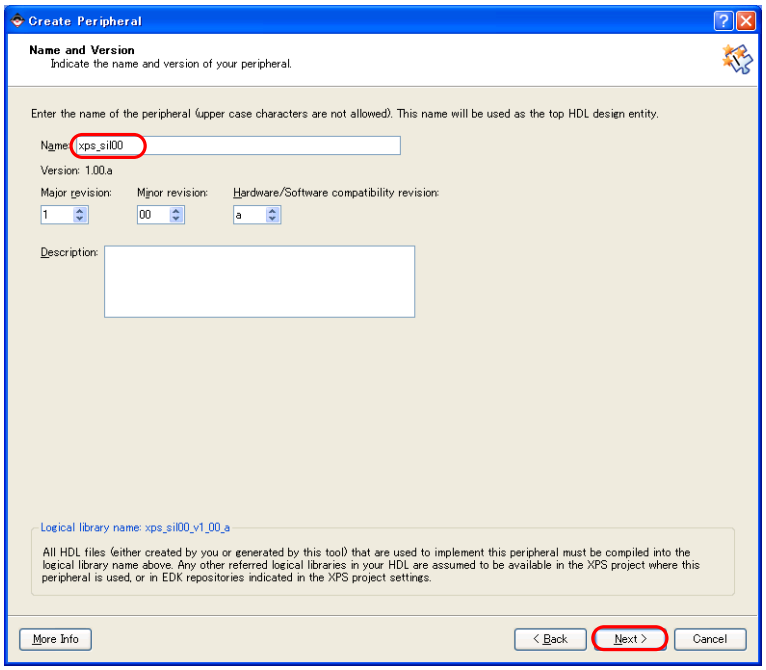


図 13.8 コアの名前

バスを選択します。[Processor Local Bus(PLB v4.6)]を選択して[Next]をクリックしてください。

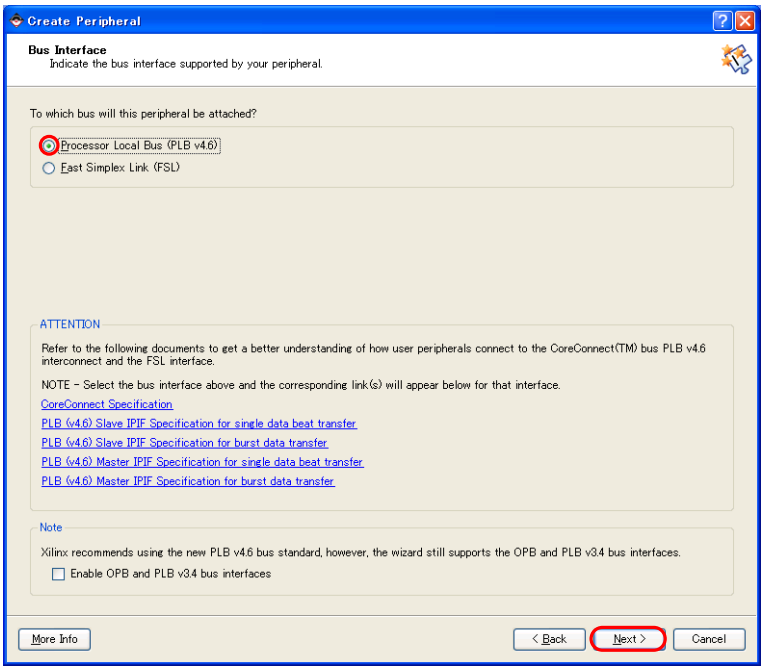


図 13.9 バスの選択

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。選択した項目により PLB ペリフェラルテンプレートが生成されます。今回は[Interrupt control]、[User logic software register]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。

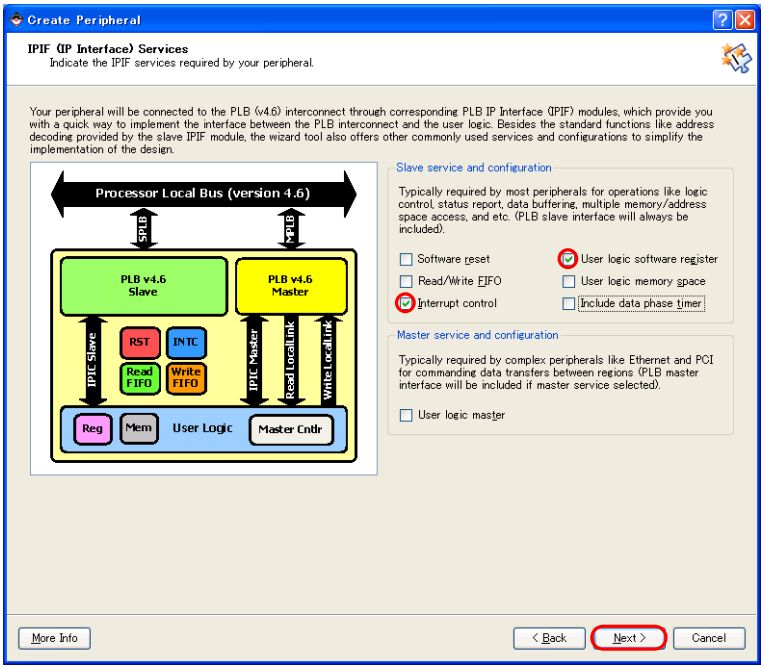


図 13.10 テンプレート追加

Slave Interface の設定ができます。ここでは何も変更せず[Next]をクリックしてください。

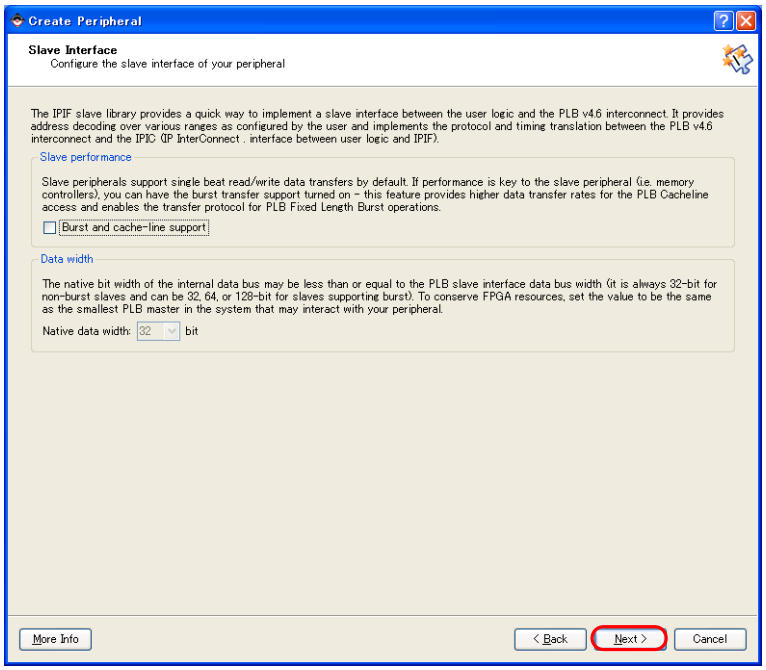


図 13.11 Slave Interface の設定

割り込みの設定をします。割り込みに使用するカウンタの Duty が 50%なのでエッジ取り込みにし、今回は立ち上がりエッジを使用します。[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]をクリックしてください。

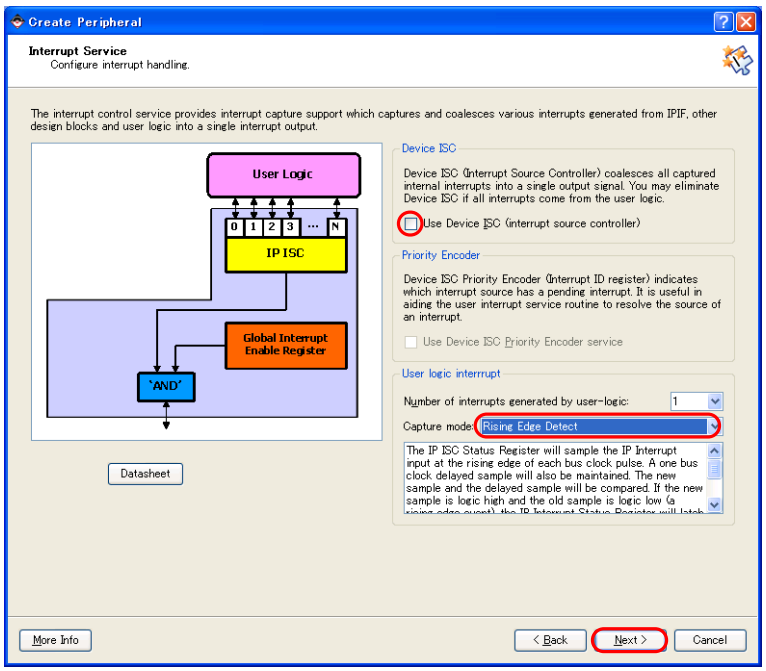


図 13.12 Interrupt 設定

ソフトウェアアクセス可能レジスタ数を指定します。今回必要となるのは書き込み/読み込みレジスタ1 つ(7 セグメント LED の値を設定する 3Byte と単色 LED のトリガ信号を設定する 1Byte)と読み込みレジ

スタ 1 つ(押しボタンスイッチの情報をやり取りする 1Byte、ロータリコードスイッチの情報をやり取りする 1Byte)です。

[Number of software accessible resiters]を[2]にし、[Next]をクリックしてください。

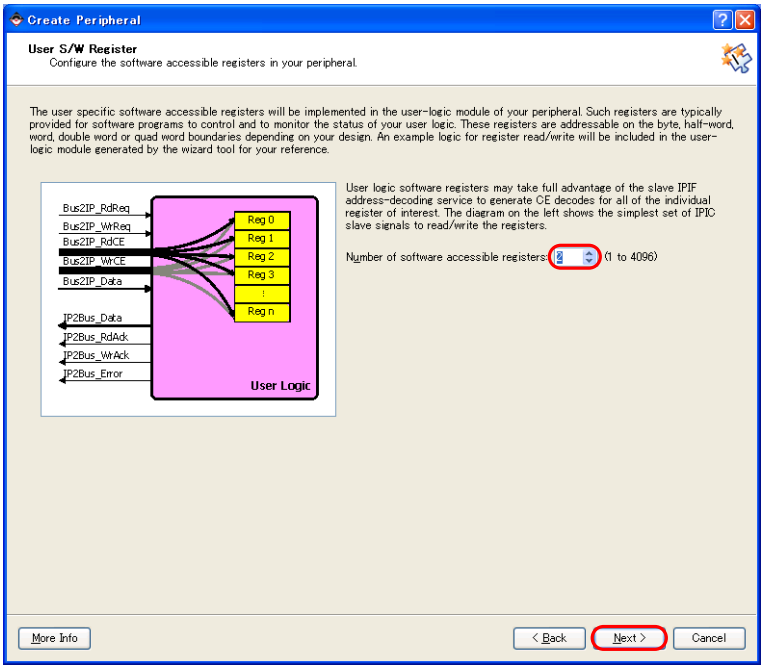


図 13.13 レジスタ数指定

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。

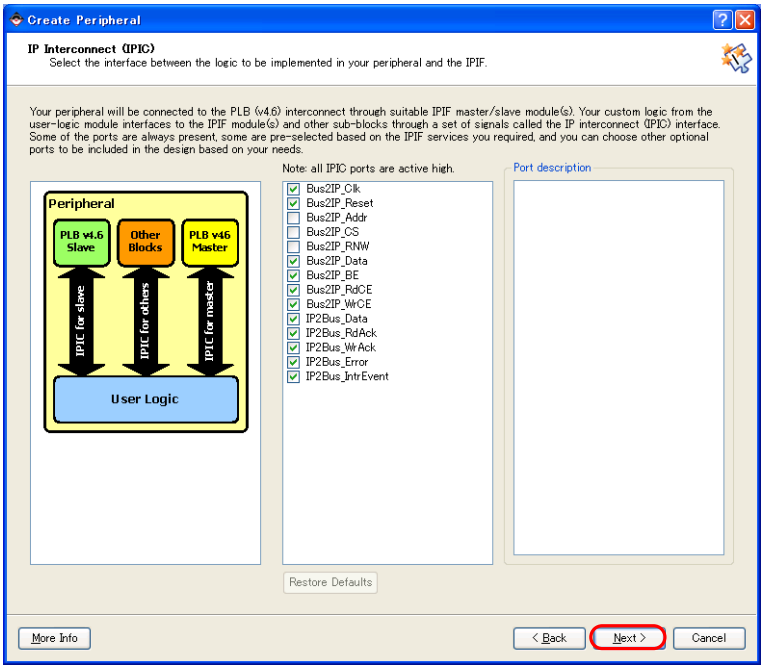


図 13.14 IPIC 設定

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。

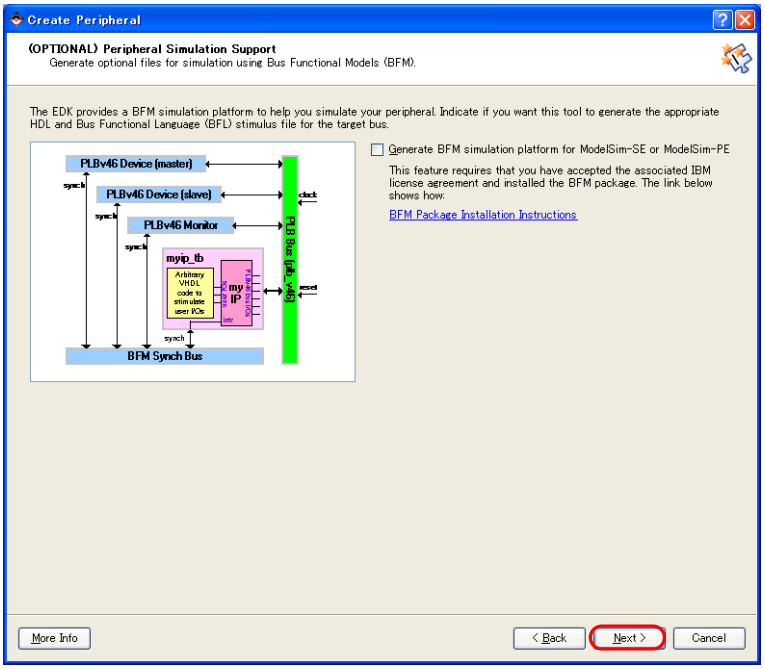


図 13.15 サポートファイル生成確認

[Generate template driver...]をチェックし、[Next]をクリックしてください。ソフトフェアドライバテンプレートファイルとドライバディレクトリ構成が作成されます。

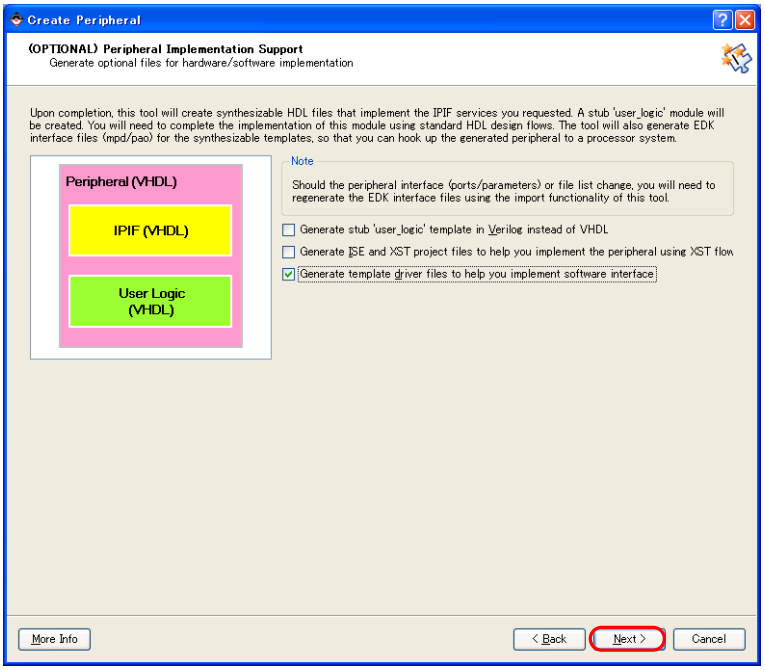


図 13.16 オプション設定

以上で終了です。[Finish]をクリックしてください。

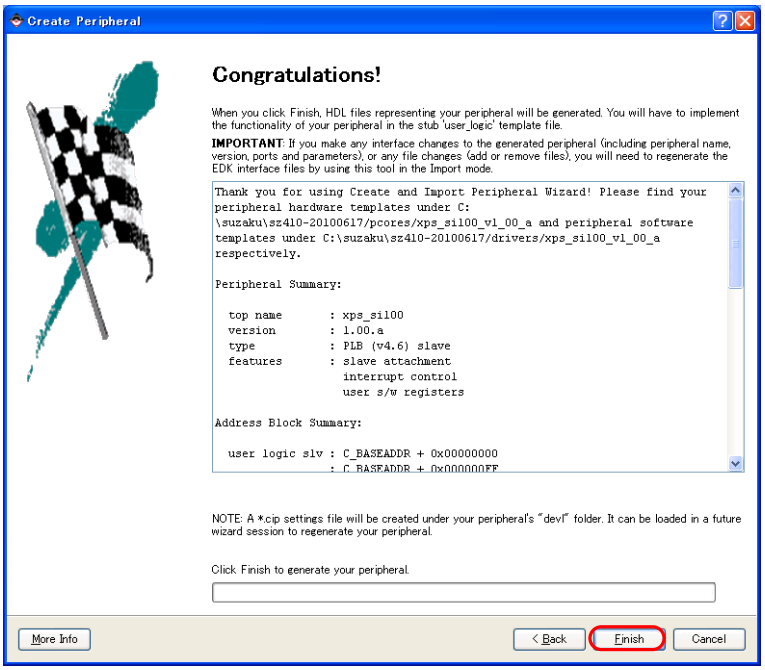


図 13.17 終了

"C:\suzaku\sz***-yyyymmdd\pcores"の下に PLB バスに接続するインターフェースの雛形が生成されます。

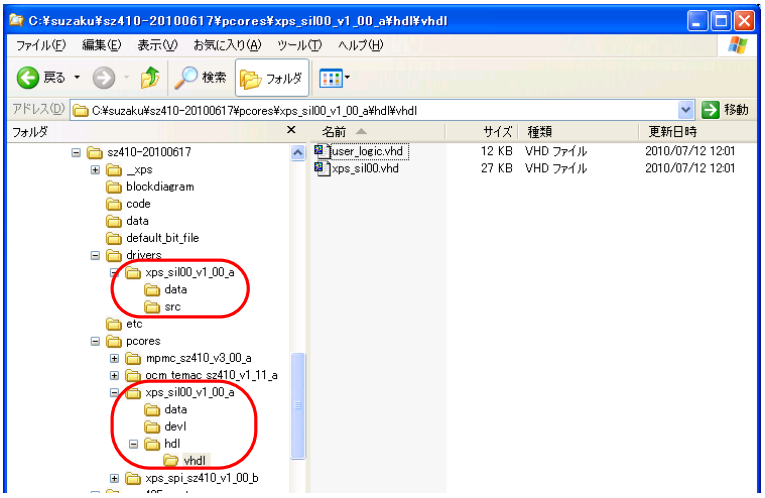


図 13.18 フォルダ構成

13.1.3. 今まで作った回路をまとめる

下図の仕様で今まで作ってきた回路を PLB バスに接続できるようにまとめます。sil00u_core.vhd をテキストエディタ等で新規作成してください。

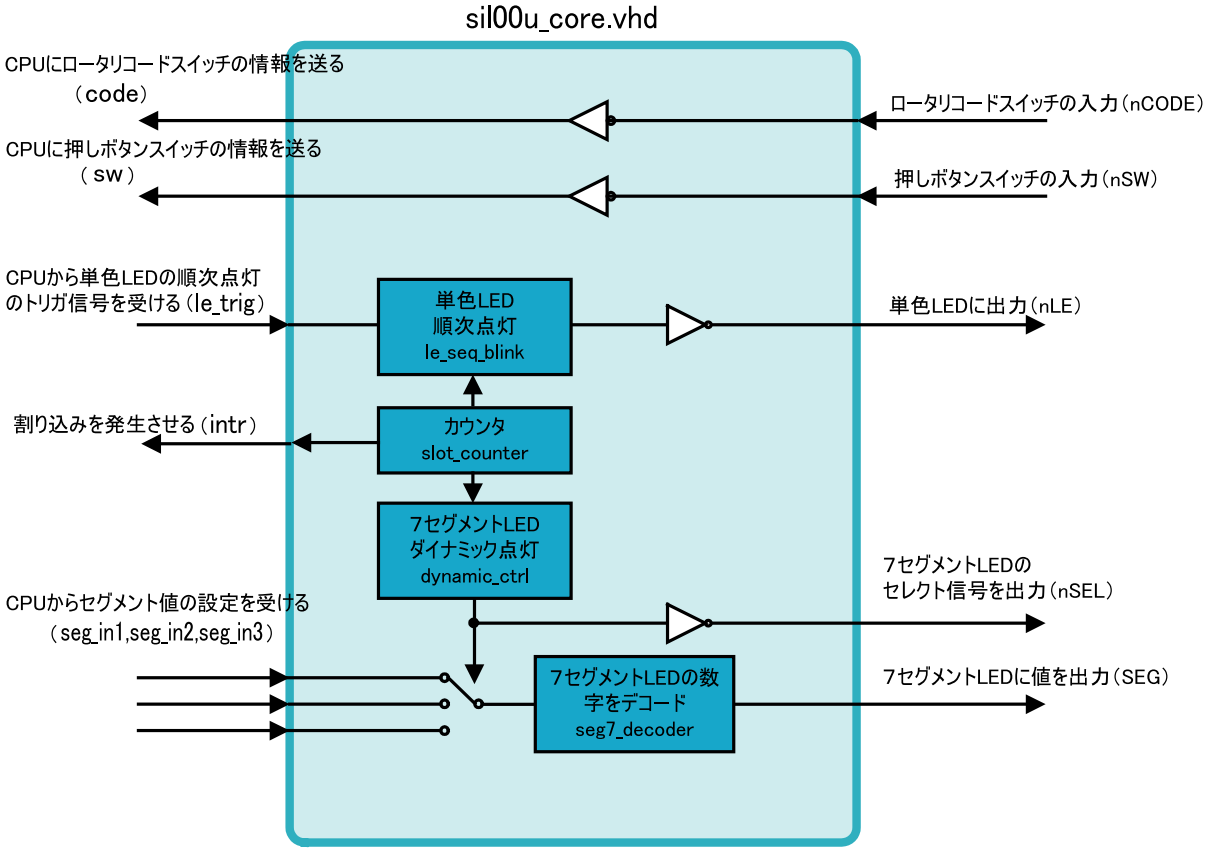


図 13.19 自作 IP コア(ソフト版)の仕様

13.1.3.1. ソースファイル編集(sil00u_core.vhd)

SZ130 ではバスクロックが 51.6096MHz、SZ410 では 87.5MHz になっています。カウンタのビット数を 4 ビット増やし 23 ビットにします。sil00u_core を上位階層として今まで作った回路、slot_counter、le_seq_blink、seg7_decoder、dynamic_ctrl 回路を呼び出します。また、押しボタンスイッチ、ロータリコードスイッチ、割り込みの信号の定義をします。

例 13.1 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
  );
  --  入出力を定義
  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);
  -- 4 つの回路のインスタンス
  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_decoder
    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
  -- 4 つの回路のコンポーネント宣言
  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_decoder_0 : seg7_decoder
    Port map(
      SEG => SEG,
      seg_data => seg_data
    );
-- トリガ信号が'1'の時順次点灯
  le_t <= le and "1111" when le_trig = '1' else "0000";
  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;
```

13.1.4. XPS 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

今まとめた回路のファイル 5 つ(sil00u_core.vhd、slot_counter.vhd、dynamic_ctrl.vhd、seg7_decoder.vhd 、 le_seq_blink.vhd) を "C:\suzaku\sz***-yyyymmdd\pcores\xps_sil00_v1_00_a\hdl\vhdl"にコピーしてください。

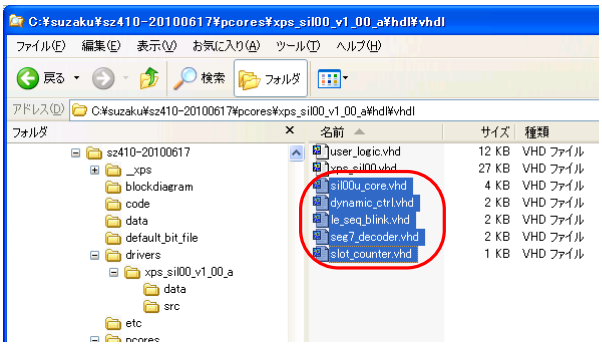


図 13.20 コアをコピー

13.1.4.1. ソースファイル編集(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。押しボタンスイッチ、ロータリコードスイッチは読み込むだけで書き込みは出来ません。この 2 つのために新たに、読み込み/書き込みレジスタではなく、読み込みレジスタを定義しています。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 13.2 xps_sil00(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 xps_sil00_v1_00_a;      -- ライブラリとして呼び出す
 use xps_sil00_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_SLV_DWIDTH    : integer              := 32;
    C_NUM_REG       : integer              := 2;
    C_NUM_INTR      : integer              := 1
```

```
    -- DO NOT EDIT ABOVE THIS LINE --------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------

    SEG         : out   STD_LOGIC_VECTOR(0 to 7);   -- 7 セグメント LED 出力
    nSEL        : out   STD_LOGIC_VECTOR(0 to 2);   -- セレクト出力
    nLE         : out   STD_LOGIC_VECTOR(0 to 3);   -- 単色 LED 出力
    nSW         : in    STD_LOGIC_VECTOR(0 to 2);   -- スイッチ入力
    nCODE       : in    STD_LOGIC_VECTOR(0 to 3);   -- ロータリ SW 入力

    -- 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_Clk          : in    std_logic;
    Bus2IP_Reset        : in    std_logic;
    Bus2IP_Data         : in    std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE           : in    std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE         : in    std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE         : in    std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data         : out   std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck        : out   std_logic;
    IP2Bus_WrAck        : out   std_logic;
    IP2Bus_Error        : out   std_logic;
    IP2Bus_IntrEvent    : out   std_logic_vector(0 to C_NUM_INTR-1)
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );

  attribute SIGIS                   : string;
  attribute SIGIS of Bus2IP_Clk     : signal is "CLK";
  attribute SIGIS of Bus2IP_Reset   : signal is "RST";

end entity user_logic;
------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  ---------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ---------------------------------------
  signal slv_reg0               : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg1               : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg_write_sel      : std_logic_vector(0 to 1);
  signal slv_reg_read_sel       : std_logic_vector(0 to 1);
  signal slv_ip2bus_data        : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_read_ack           : std_logic;
  signal slv_write_ack          : std_logic;

  ---------------------------------------
  -- Signals for user logic interrupt example
  ---------------------------------------
  signal intr_conuter    : std_logic_vector(0 to C_NUM_INTR-1); -- 割り込み用

begin
```

```
 -- 下位モジュール呼び出し sil00u_core インスタンス
 sil00u_core_0 : entity xps_sil00_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,

    -- R/W
    seg_in1 => slv_reg0(4 to 7),
    seg_in2 => slv_reg0(12 to 15),
    seg_in3 => slv_reg0(20 to 23),
    le_trig => slv_reg0(31),

    -- Register Read
    sw      => slv_reg1(5 to 7),
    code    => slv_reg1(12 to 15),
    -- intrrupt
    intr    => intr_counter(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');

      else
        case slv_reg_write_select is
          when "10" =>
            for byte_index in 0 to (C_SLV_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 "01" =>
  --            for byte_index in 0 to (C_SLV_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 others => null;
      end case;
    end if;
  end if;
end process SLAVE_REG_WRITE_PROC;

-- implement slave model software accessible register(s) read mux
-- CPU からのレジスタ読み込み
SLAVE_REG_READ_PROC : process( slv_reg_read_select, slv_reg0, slv_reg1 ) is

begin
  case slv_reg_read_select is
    when "10" => slv_ip2bus_data <= slv_reg0;
    when "01" => slv_ip2bus_data <= slv_reg1;
    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');
--        intr_counter <= (others => '0');
--      else
--        counter := counter + 1;
--        if ( counter = ALL_ONES ) then
--          intr_counter <= (others => '1');
--        else
--          intr_counter <= (others => '0');
--        end if;
--      end if;
--    end if;
--
--  end process INTR_PROC;

IP2Bus_IntrEvent <= intr_counter;  -- 割り込みを counter からの出力に接続

------------------------------------------
-- Example code to drive IP to Bus signals
------------------------------------------
IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else (others => '0');
IP2Bus_WrAck <= slv_write_ack;
IP2Bus_RdAck <= slv_read_ack;
IP2Bus_Error <= '0';
```

```
end IMP;
```

13.1.4.2. ソースファイル編集(xps_sil00.vhd)

xps_sil00.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。xps_sil00 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 13.3 xps_sil00(xps_sil00.vhd)

```
------------------------------------------------------------------------------
-- xps_sil00.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 interrupt_control_v2_00_a;
use interrupt_control_v2_00_a.interrupt_control;

library plbv46_slave_single_v1_00_a;
use plbv46_slave_single_v1_00_a.plbv46_slave_single;

library xps_sil00_v1_00_a;
use xps_sil00_v1_00_a.user_logic;

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
entity xps_sil00 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
    --中略
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
```

```
  attribute SIGIS : string;
  attribute SIGIS of SPLB_Clk       : signal is "Clk";
  attribute SIGIS of SPLB_Rst       : signal is "Rst";
  attribute SIGIS of IP2INTC_Irpt  : signal is "INTR_LEVEL_HIGH";

end entity xps_sil00;
------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------
architecture IMP of xps_sil00 is
--中略
begin
  ---------------------------------------
  -- instantiate plbv46_slave_single
  ---------------------------------------
  PLBV46_SLAVE_SINGLE_I : entity plbv46_slave_single_v1_00_a.plbv46_slave_single
    generic map
    (
      --中略
    )
    port map
    (
      --中略
    );

  ---------------------------------------
  -- instantiate interrupt_control
  ---------------------------------------
  INTERRUPT_CONTROL_I : entity interrupt_control_v2_00_a.interrupt_control
    generic map
    (
      --中略
    )
    port map
    (
      --中略
    );

  ---------------------------------------
  -- instantiate the User Logic
  ---------------------------------------
  USER_LOGIC_I : entity xps_sil00_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 ------------------
      --中略
    );
```

```
--中略
end IMP;
```

13.1.4.3. MPD ファイルの編集(xps_sil00_v2_1_0.mpd)

"C:\suzaku\sz***-yyyymmdd\pcores\xps_sil00_v1_00_a\data"を開いてください。

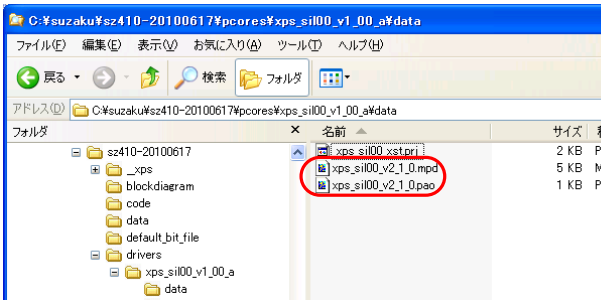


図 13.21 フォルダ構成

xps_sil00_v2_1_0.mpd を編集します。mpd(Microprocessor Peripheral Definition)ファイルでは信号の入出力の方向やビット幅等を定義します。7 セグメント LED、7 セグメント LED セレクト、単色 LED、押しボタンスイッチ、ロータリコードスイッチの信号を外部と接続できるように定義します。以下の文を一番下に追加してください。

例 13.4 MPD ファイルの編集(xps_sil00_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]
```

13.1.4.4. PAO ファイルの編集(xps_sil00_v2_1_0.pao)

xps_sil00_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。自分で書いたソースコードを追加します。以下の文を一番下に追加してください。

例 13.5 PAO ファイルの編集(xps_sil00_v2_1_0.pao)

```
lib xps_sil00_v1_00_a sil00u_core vhdl
lib xps_sil00_v1_00_a slot_counter vhdl
lib xps_sil00_v1_00_a le_seq_blink vhdl
lib xps_sil00_v1_00_a seg7_decoder vhdl
lib xps_sil00_v1_00_a dynamic_ctrl vhdl
```

13.1.4.5. ドライバの編集(xps_sil00.c)

"C:\suzaku\sz***-yyyymmdd\drivers\xps_sil00_v1_00_a\src\xps_sil00.c"を編集します。SUZAKU では stdio を使用しておらず、xil_printf() は生成されません。

XPS_SIL00_Intr_DefaultHandler 関数の中に記述されている xil_printf()の行をコメントアウトしてください。

例 13.6 ドライバの編集(xps_sil00.c)

```
void XPS_SIL00_Intr_DefaultHandler(void * baseaddr_p)
{
  Xuint32 baseaddr;
  Xuint32 IntrStatus;
  Xuint32 IpStatus;
  baseaddr = (Xuint32) baseaddr_p;

  {
//    xil_printf("User logic interrupt! \n\r");
    IpStatus = XPS_SIL00_mReadReg(baseaddr, XPS_SIL00_INTR_ISR_OFFSET);
    XPS_SIL00_mWriteReg(baseaddr, XPS_SIL00_INTR_ISR_OFFSET, IpStatus);
  }
}
```

これで自作 IP コアの完成です。

13.2. 自作 IP コアを SUZAKU のデフォルトに追加

自作 IP コアを SUZAKU のデフォルトに追加します。まず、SUZAKU のデフォルトに追加した時のブロック図を見てください。自作 IP コアは、PLB と接続され、外部(押しボタンスイッチや単色 LED)とつながります。

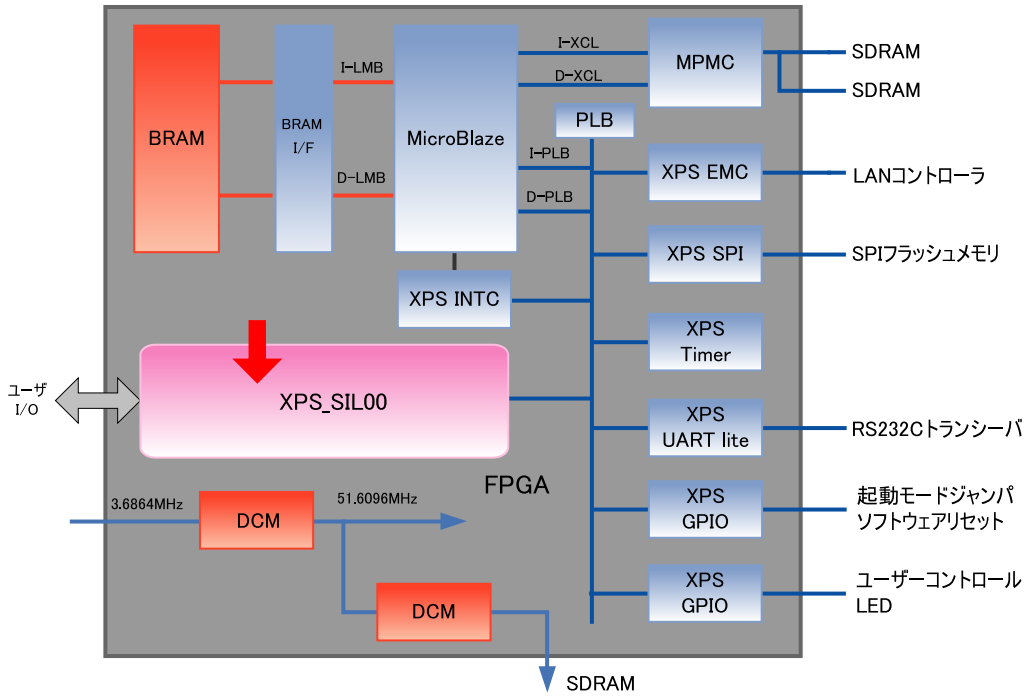


図 13.22 SZ130 のデフォルトに自作 IP コアを追加

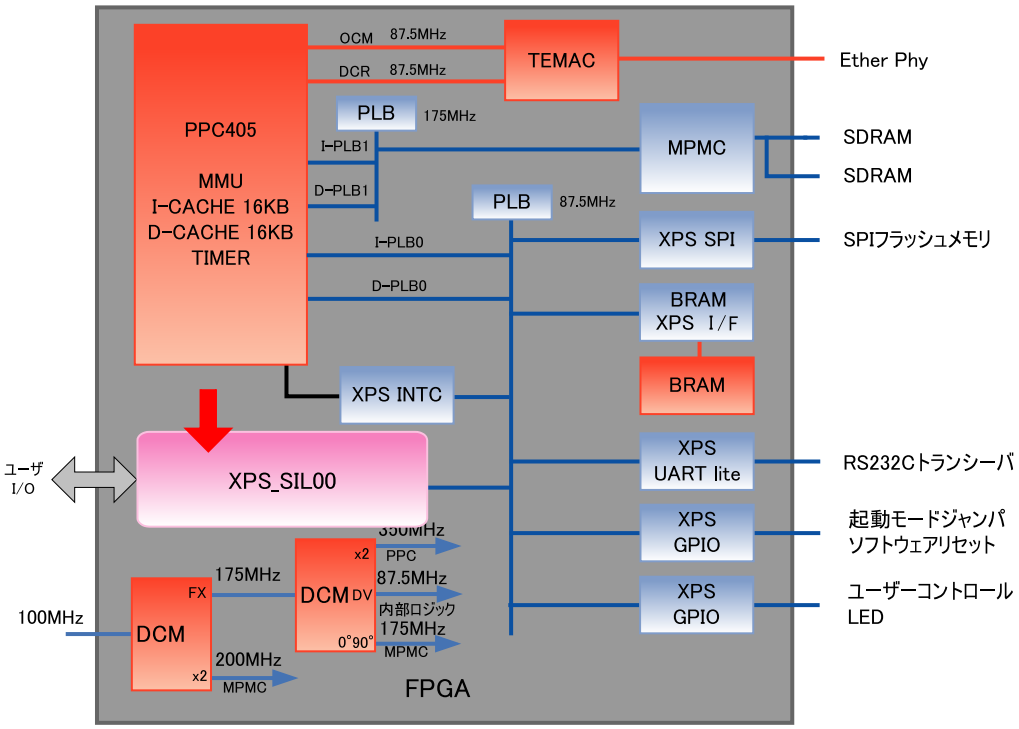


図 13.23 SZ410 のデフォルトに自作 IP コアを追加

13.2.1. ハードウェア設定

自作 IP コアが、XPS に読み込まれているか確認します。XPS に自作 IP 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。

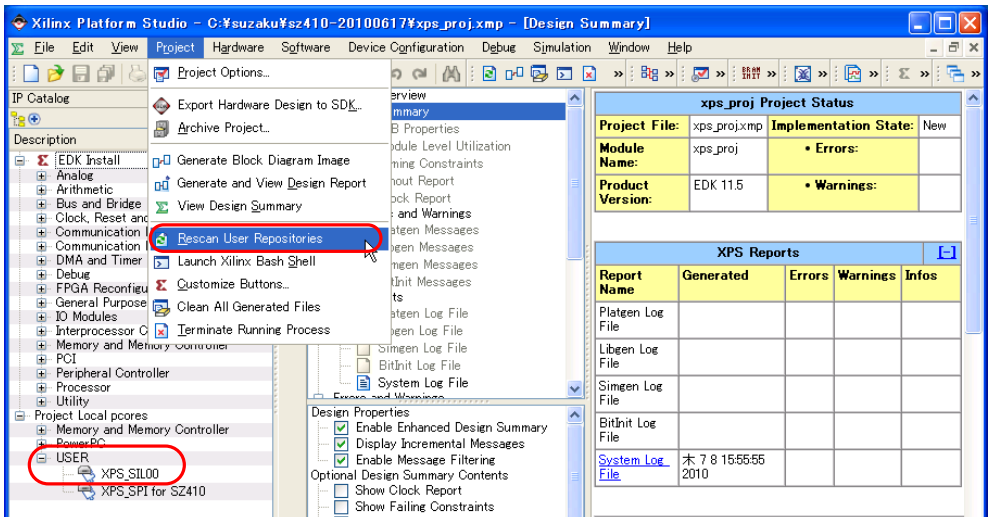


図 13.24 自作 IP コア読み込み

IP Catalog の Project Local pcores に自作 IP コア xps_sil00 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 XPS を閉じて、再起動するとうまくいくことがあります。

13.2.1.1. IP コアの追加とバス接続

[xps_sil00]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。xps_sil00 が追加されます。追加できたら xps_sil00_0 の横の丸をクリックして、バス(SZ410 の場合 plb_peripheral)に接続して下さい。

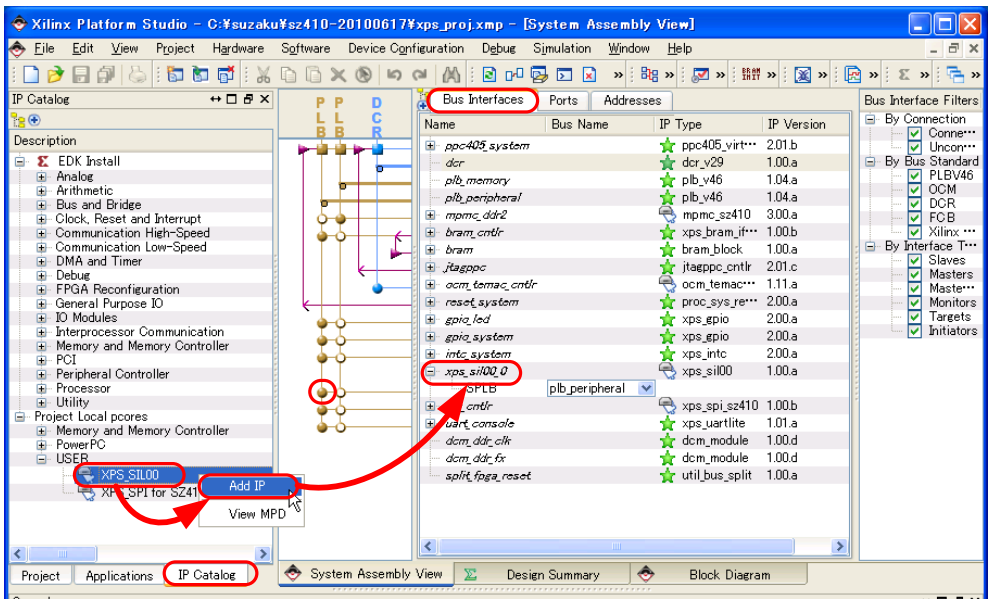


図 13.25 自作 IP コア追加してバスに接続

BBoot にはじめから仕込んでいる名前の都合上、インスタンス名を変更します。インスタンス名をクリックし、xps_sil00_0 から sil_cntlr に変更して下さい。

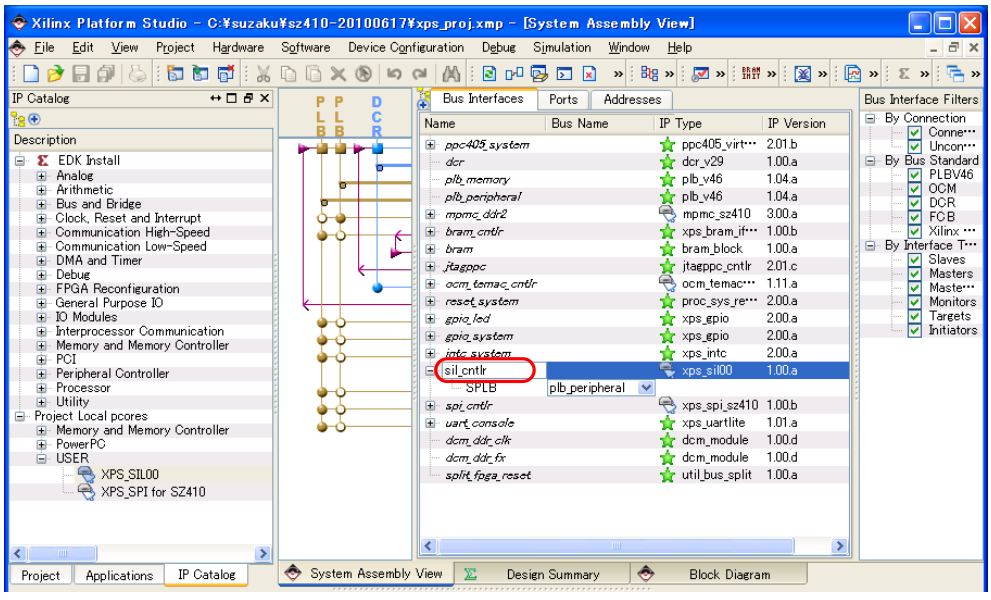


図 13.26 インスタンス名変更

13.2.1.2. IP コアの設定と確認

sil_cntlr 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。メモリアドレスを設定します。[C_BASEADDR]、[C_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(「1.5. メモリマップ」参照)。設定ができたら、[Addresses]タブを選択し、sil_cntlr の Base Address と High Address とSize に間違いがないか、確認してください。

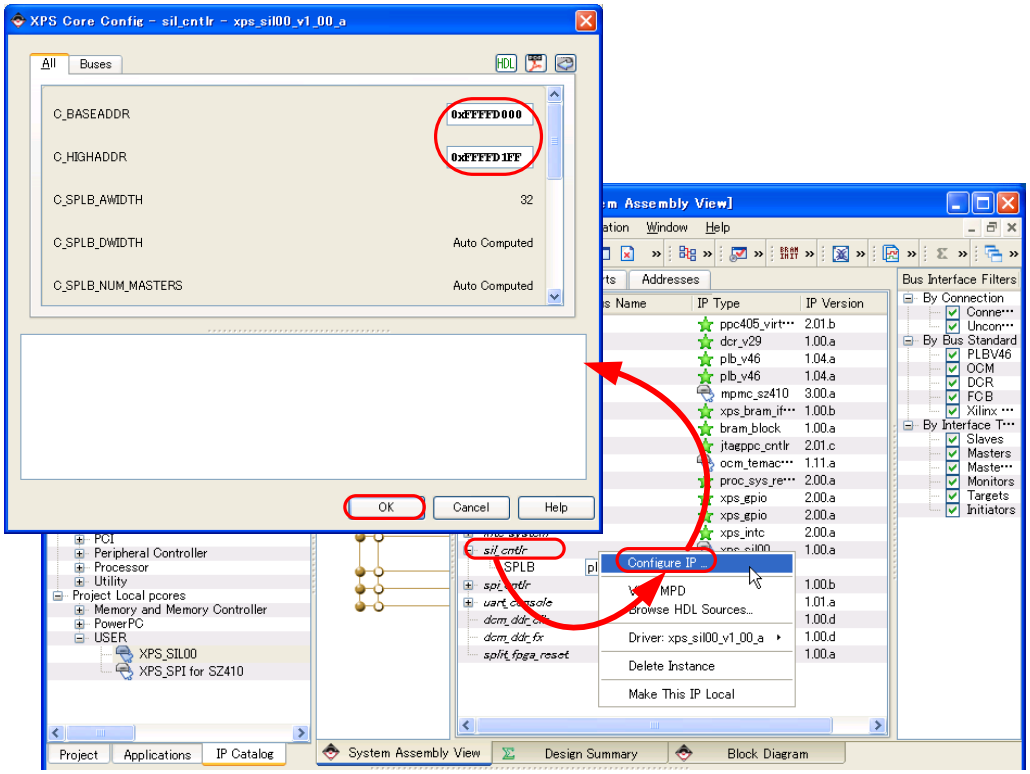


図 13.27 アドレス設定画面呼び出し

表 13.1 自作 IP のメモリアドレス

	SZ130	SZ410
Base Address	0xFFFFD000	0xF0FFD000
High Address	0xFFFFD1FF	0xF0FFD1FF

13.2.1.3. 信号の定義

[Ports]タブを選択し、sil_cntlr の⊞をクリックして開いてください。mpd ファイルで設定した信号線と自動生成された割り込み線が出来上がっていると思います。SEG の Net の部分をクリックし、Net 名に[SEG]と入力してください。

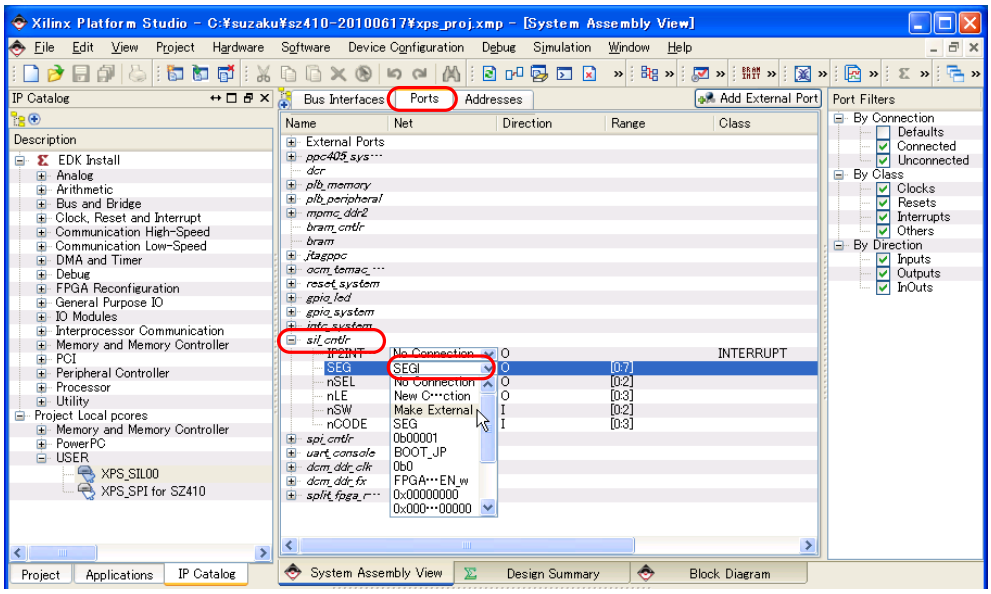


図 13.28 NET 名入力

もう一度 SEG の Net 部分をクリックし、[Make External]を選択してください。External Ports に sil_cntlr_SEG_pin が追加されたのを確認して下さい。

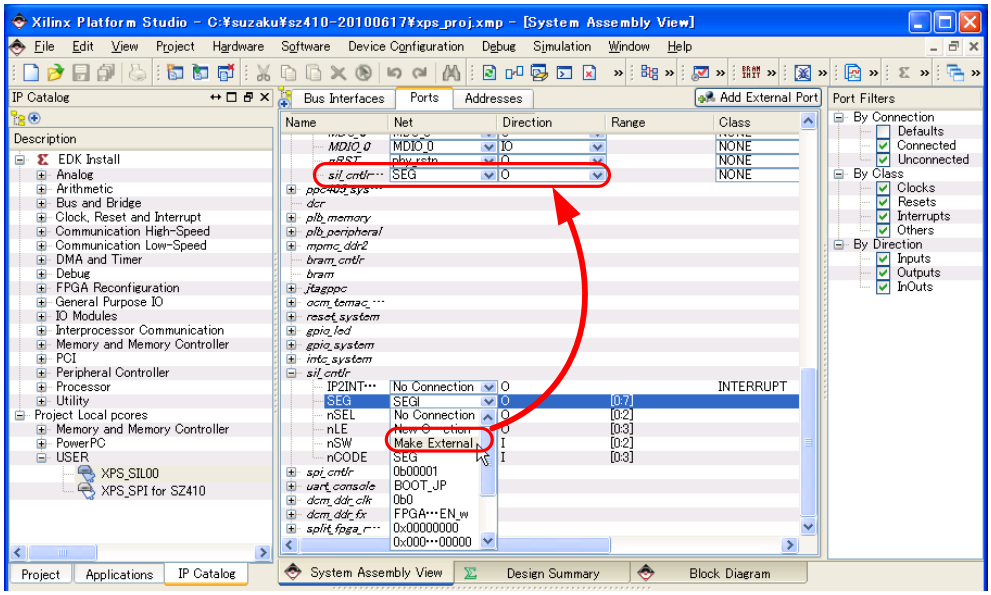


図 13.29 外部信号にする

長くてなんだか読みにくいので信号名を変更します。sil_cntlr_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。

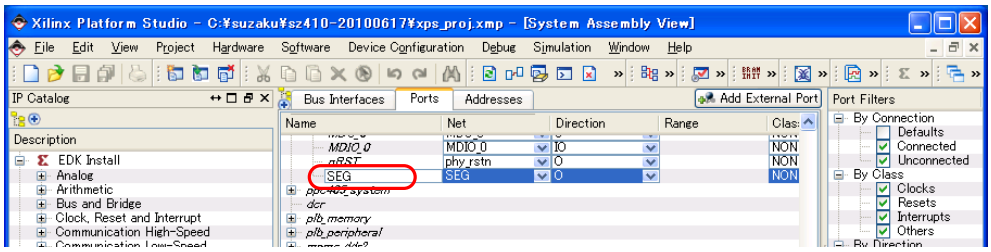


図 13.30 出力信号定義

nSEL、nLE、nSW、nCODE も SEG と同様の操作を行ってください。信号名はそれぞれネット名と同じに変更します。

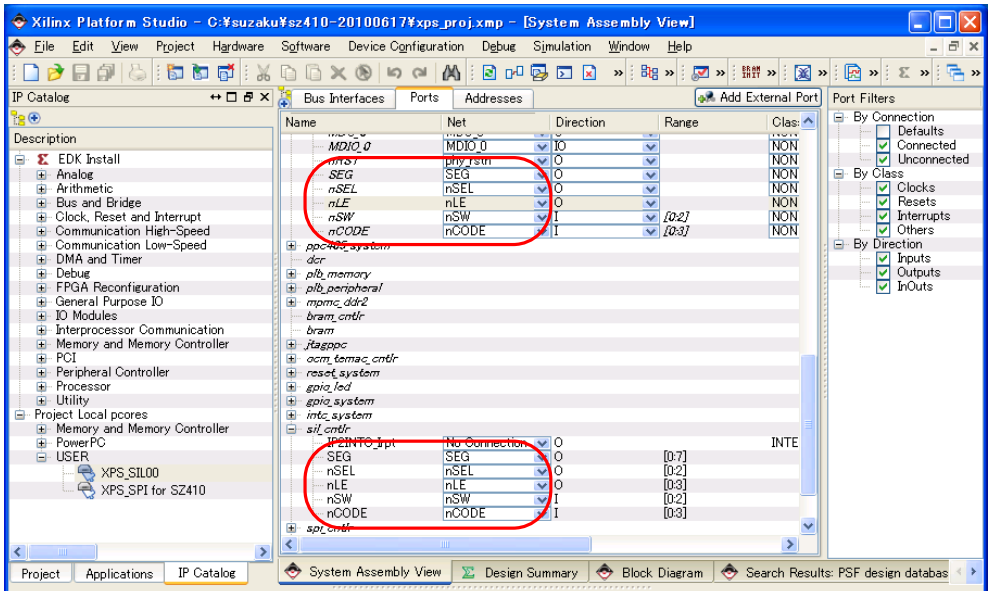


図 13.31 残り出力信号定義

13.2.1.4. 割り込み

MicroBlaze および PowerPC405 は外部から 1 つの割り込みしか受けることができません。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。

intc_system の Intr の Net をクリックしてください。割り込み信号の設定ウィンドが立ち上がります。sil_intr:IP2INTC_Irpt を選択し、青矢印をクリックして下さい。割り込みが追加されます。上にあるものほど割り込みの優先順位が低くなります。優先順位は一番低くていいので、上三角をクリックして、一番上にして下さい。設定できたら[OK]をクリックして下さい。

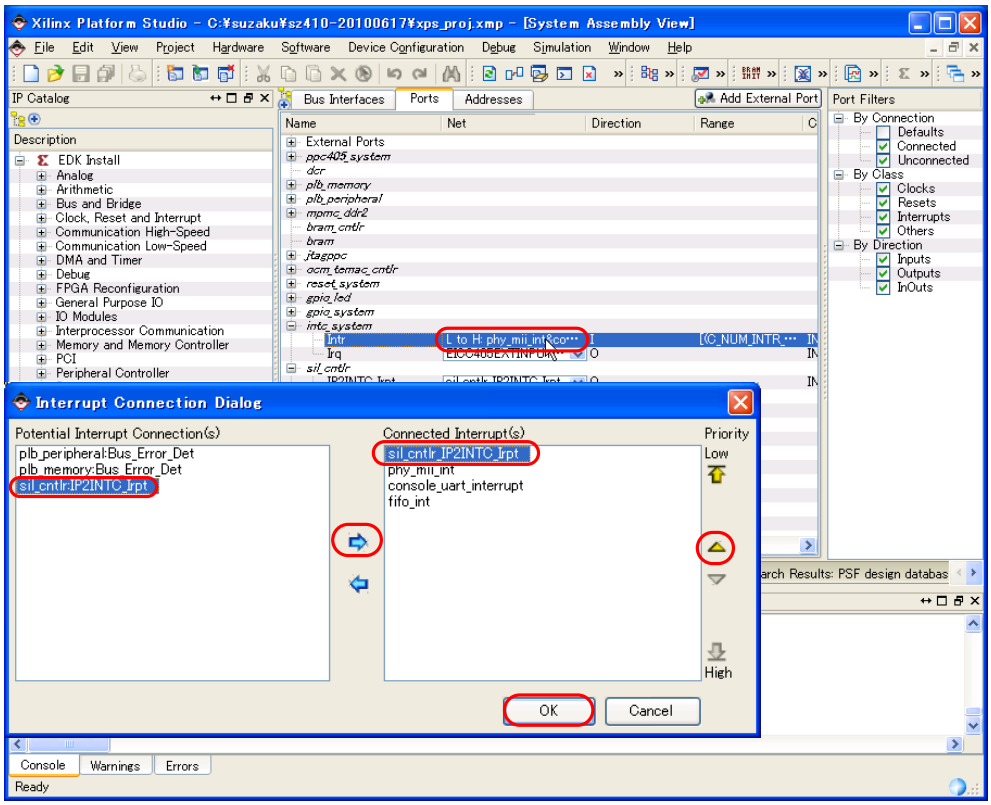


図 13.32 割り込みコントローラ

13.2.1.5. BRAM の容量を増やす

SZ130

SZ130 の場合、BRAM の容量が少し足りなくなるので BRAM の容量を 8KB から 16KB に増やします。

d_lmb_bram_if_cntlr 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。[LMB BRAM High Address]を[0x00003FFF]に変更し、[OK]をクリックして下さい。

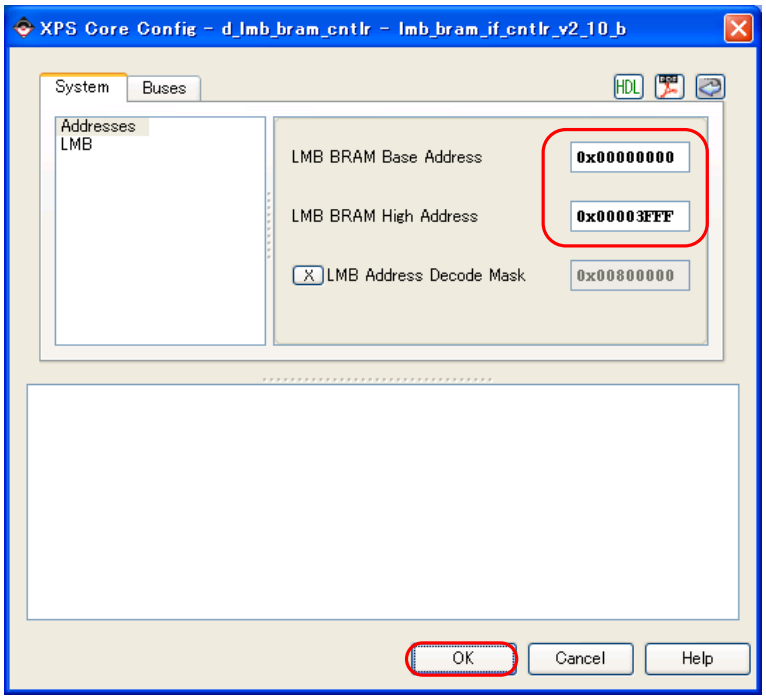


図 13.33 d_lmb_bram_if_cntlr 8KB→16KB に変更

i_lmb_bram_if_cntlr も同様に[0x00003FFF]に変更してください。これで BRAM 容量が 16KB に変更されます。

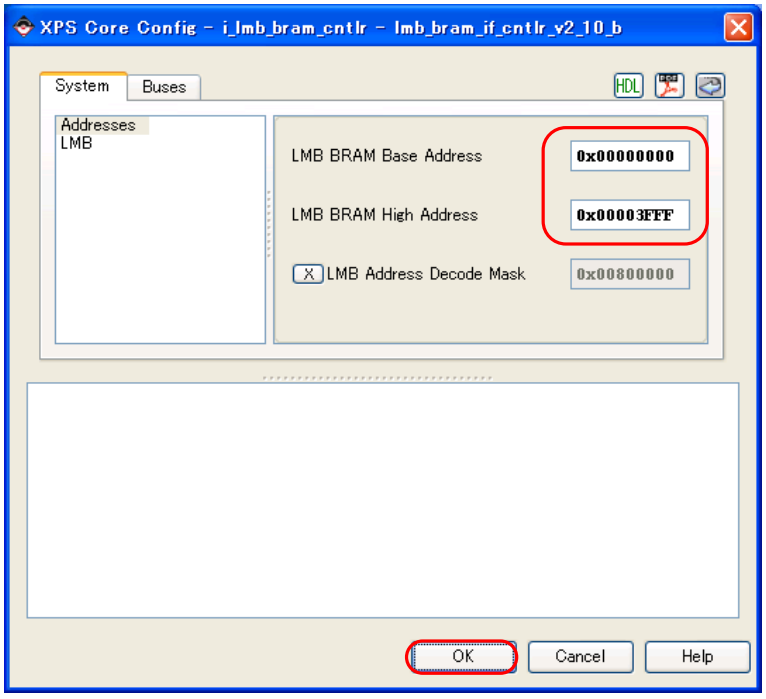


図 13.34 i_lmb_bram_if_cntlr 8KB→16KB に変更

13.2.1.6. BRAM を追加する

SZ410

SZ410 の場合、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ために、BRAM を追加します。EVPR は上位 16bit のみが使用され、下位 16bit 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 まで拡張しないといけないのですが、これだと BRAM の容量が 64kB になってしまいます。無駄に大きく確保するのはもったいないので、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。

[IP Catalog]タブをクリックし、[Memory and Memory Controller]→[xps_bram_cntlr]を追加してインスタンス名を bram_cntlr_vec に変更して、PLB(plb_peripheral)に接続して下さい。追加できたら今度は[Memory and Memory Controller]→[bram_block]を追加してインスタンス名を bram_vec に変更し、bram_cntlr_vec に接続してください。

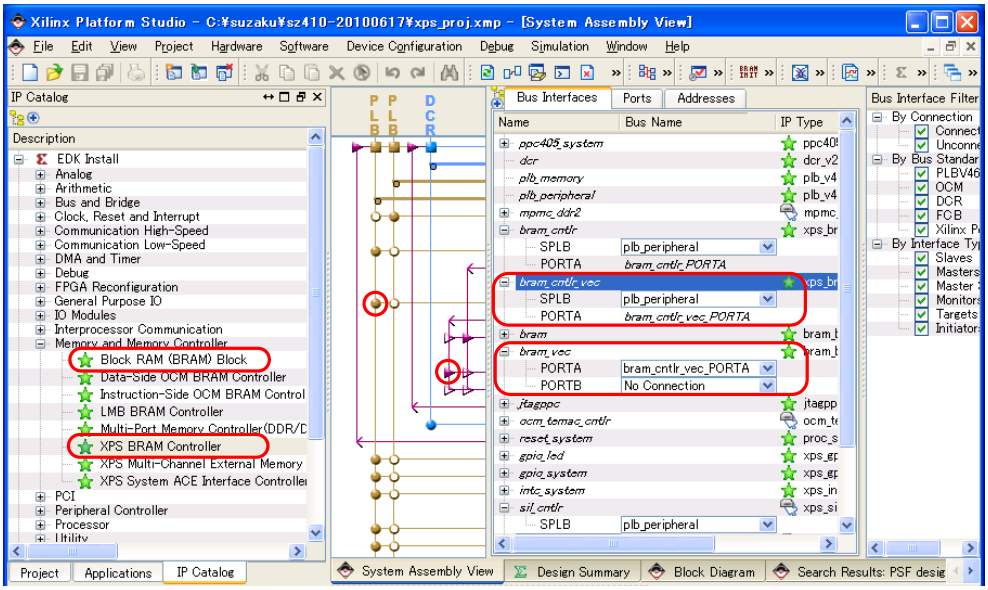


図 13.35 BRAM を追加(SZ410)

bram_cntrl_vec 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。

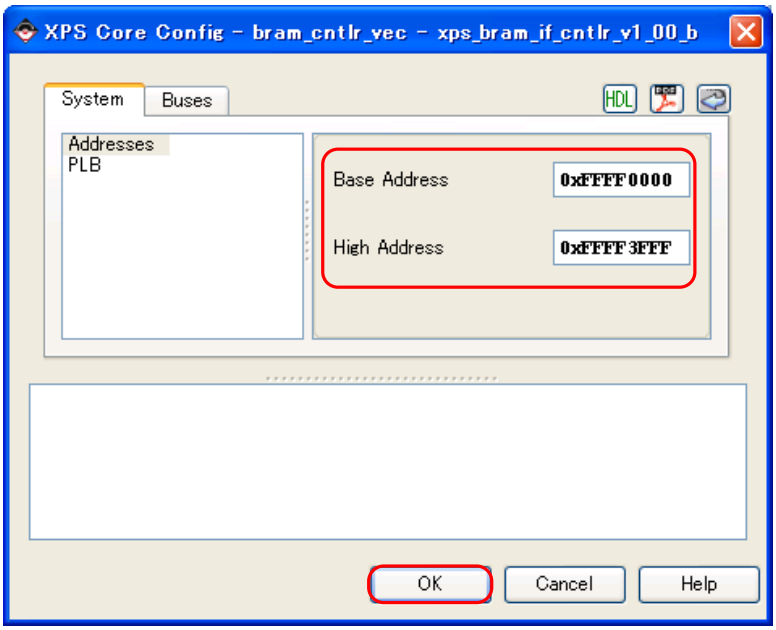


図 13.36 BRAM のアドレス設定(SZ410)

13.2.1.7. MPMC 編集

SZ410

SZ410 の場合、BRAM の容量をかせぐために、MPMC で BRAM をあまり使用しないように変更します。GUI で編集したいところですが、MPMC を GUI で編集すると、ツールに出力ピンのビットを勝手に反転されてしまうので、mhs ファイルを編集します。[Project]タブをクリックし、[MSS File: xps_proj.mss]をダブルクリックして開いてください。以下の 2 行を追加して保存してください。

例 13.7 mpmc の設定

```
BEGIN mpmc_sz410
 PARAMETER INSTANCE = mpmc_ddr2
 PARAMETER HW_VER = 3.00.a
 PARAMETER C_MEM_PARTNO = MT47H16M16-37E
 PARAMETER C_MPMC_CLK0_PERIOD_PS = 5714
 PARAMETER C_MPMC_BASEADDR = 0x00000000
 PARAMETER C_MPMC_HIGHADDR = 0x03FFFFFF
 PARAMETER C_MEM_DATA_WIDTH = 32
 PARAMETER C_DDR2_DQSN_ENABLE = 1
 PARAMETER C_MEM_CLK_WIDTH = 2
 PARAMETER C_PI0_RD_FIFO_TYPE = SRL
 PARAMETER C_PI0_WR_FIFO_TYPE = SRL
 BUS_INTERFACE SPLB0 = plb_memory
 PORT MPMC_Clk90 = DDR_SDRAM_64Mx32_mpmc_clk_90_s
 # 後略
```

13.2.1.8. ピンアサインの設定

[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。

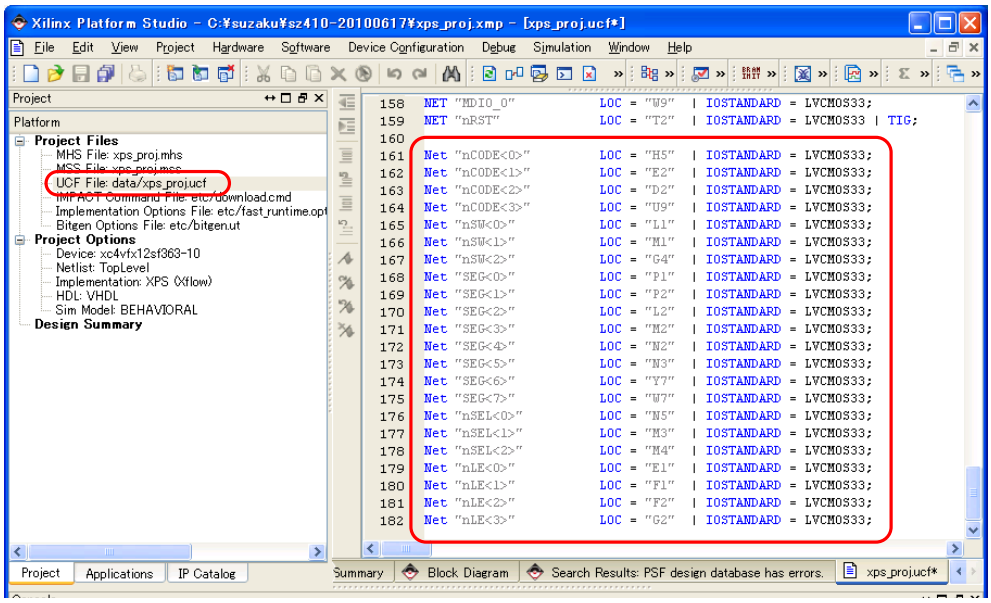


図 13.37 自作 IP コア(xps_proj.ucf)

表 13.2 自作 IP コア ピンアサイン

	SZ130	SZ410
nCODE<0>	J1	H5
nCODE<1>	F9	E2
nCODE<2>	E9	D2
nCODE<3>	A10	U9
nSW<0>	D11	L1
nSW<1>	C11	M1
nSW<2>	F11	G4
SEG<0>	L5	P1
SEG<1>	L6	P2
SEG<2>	L4	L2
SEG<3>	L3	M2
SEG<4>	L2	N2
SEG<5>	L1	N3
SEG<6>	C9	Y7
SEG<7>	D9	W7
nSEL<0>	K6	N5
nSEL<1>	K4	M3
nSEL<2>	K3	M4
nLE<0>	A11	E1
nLE<1>	B11	F1
nLE<2>	F12	F2
nLE<3>	E12	G2

以上で自作コアの追加は終わりです。

13.2.2. ネットリスト作成とプログラムファイル(Hard のみ)作成

自作 IP コアに間違いがないかチェックします。[Hardware]→[Generate Netlist]をクリックして下さい。ネットリストが生成されます。

[Hardware]→[Generate Bitstream]をクリックして下さい。ソフトウェアを含まない bit ファイルが生成されます。

エラーがある場合、ログを確認し修正を行ってください。自作 IP コアがエラーを出している場合、詳細なログが"C:\suzaku\sz***-yyyymmdd\synthesis\sil_cntlr_wrapper_xst.srp"にあるので確認してください。

13.2.3. ソフトウェア設定

13.2.3.1. ライブラリ、ドライバ生成

[Software]→[Generate Libraries and BSPs]をクリックして下さい。ライブラリと様々な設定を定義したヘッダファイルが出来上がります。

xparametaers.h を開いてください。自作 IP コアの BASEADDR と HIGHADDR も自動で定義されます。

例 13.8 xparameters.h の定義の例

```
/* Definitions for driver OPB_SIL00 */
#define XPAR_OPB_SIL00_NUM_INSTANCES 1

/* Definitions for peripheral OPB_SIL00_0 */
#define XPAR_OPB_SIL00_0_DEVICE_ID 0
#define XPAR_OPB_SIL00_0_BASEADDR 0xFFFFD000
#define XPAR_OPB_SIL00_0_HIGHADDR 0xFFFFD1FF
```

ソフトウェアに関するファイルは SZ130 の場合"C:\suzaku\sz***-yyyymmdd\microblaze_i"、SZ410 の場合"C:\suzaku\sz***-yyyymmdd\ppc405_system"の下に収められます。このフォルダの下の"\include\xps_sil00.h"に自作 IP コアを扱うことのできる関数等が定義されています。

13.2.4. アプリケーション(BBoot)編集

SUZAKU のデフォルトに入っている BBoot に手を加えてスロットマシンのアプリケーションを作成します。

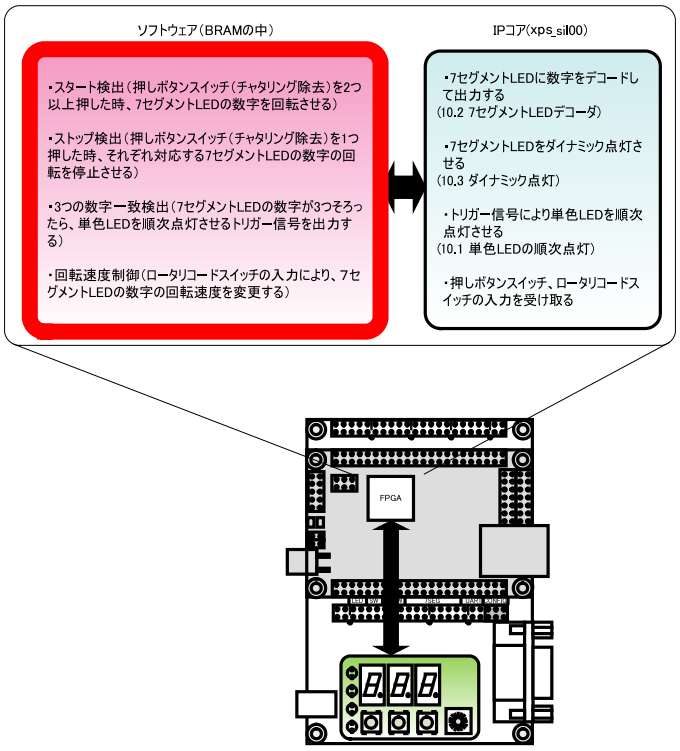


図 13.38 スロットマシンのアプリケーションをつくる

13.2.4.1. BBoot のフロー

デフォルトの SUZAKU の BBoot は、以下のようなフローで動作します。

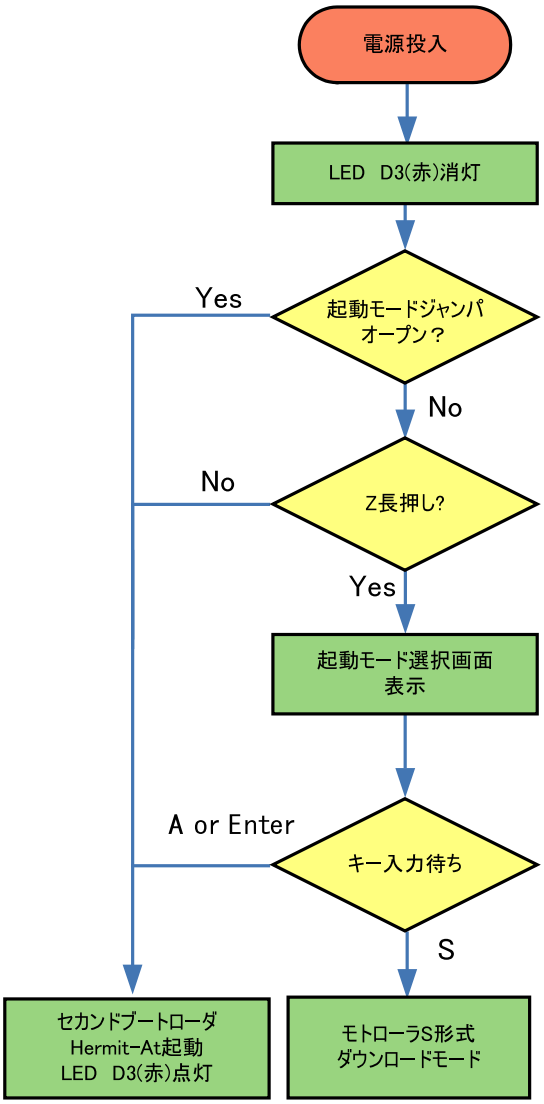
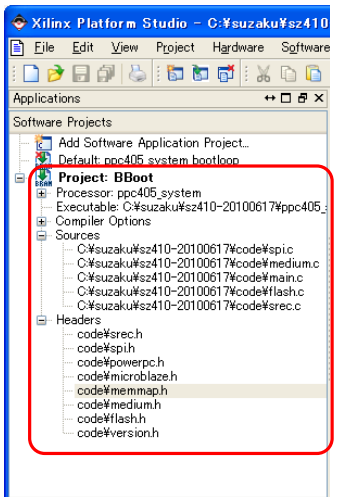


図 13.39 BBoot のフロー

[Applications]タブをクリックしてください。BBoot は以下のファイル等で構成されます。SUZAKU 全機種に対応しているので、SZ130、SZ410 に関係ないファイルも含まれています。



spi.c	SPI フラッシュメモリに対する Read、Write 等
medium.c	UART での送受信
main.c	ブートモードのチェック、コマンドの受信等 (XPS では、ベクタやリンカがデフォルトで設定されているため、main 関数を書けば、main 関数からプログラムがスタートするようにコンパイルしてくれます。)
srec.c	モトローラ S 形式のファイルの受信

図 13.40 BBoot の構成

スロットマシンのアプリケーションは、自作 IP コアからの割り込みでタイミングをはかるモードと、ビジーループでタイミングをはかるモードを実装します。組み込みソフトウェアにとって、割り込みは重要機能のひとつです。「13.4. ソフトウェアのデバッグ」でデバッガを起動し、サンプルソフトウェアの流れを確認しますので、詳細動作を確認してください。

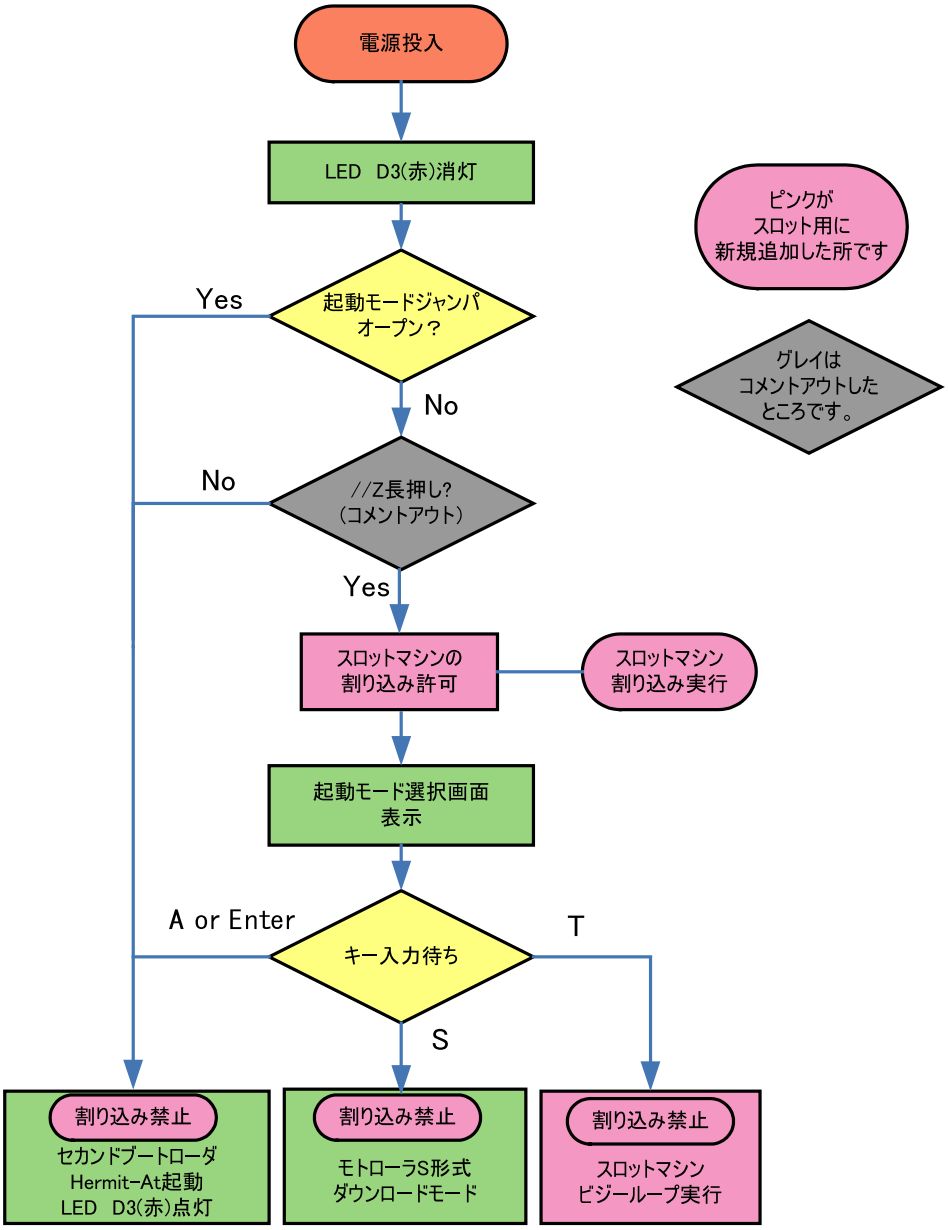


図 13.41 スロットマシンのフロー

13.2.4.2. BBoot ににソースファイル追加

BBoot のプロジェクトにスロットマシンのサンプルアプリケーションを追加します。追加するスロットマシンのアプリケーションの仕様は以下となります。

- スタート検出(押しボタンスイッチ(チャタリング除去)を 2 つ以上押した時、7 セグメント LED の数字を回転させる)
- ストップ検出(押しボタンスイッチ(チャタリング除去)を 1 つ押した時、それぞれ対応する 7 セグメント LED の数字の回転を停止させる)
- 3 つの数字一致検出(7 セグメント LED の数字が 3 つそろったら、単色 LED を順次点灯させるトリガー信号を出力する)

- 回転速度制御(ロータリコードスイッチの入力により、7 セグメント LED の数字の回転速度を変更する)
- 電源投入時は自作 IP コアからの割り込みでタイミングをはかり、コンソールから"t"の文字を受けたら、ビジーループでタイミングをはかるモードに変更

"C\suzaku\sz***-yyyymmdd\code" フォルダに slot.c、interrupt.c、slot.h、interrupt.h が入っているのを確認してください。入っていない場合は付属 CD-ROM からコピーしてください。

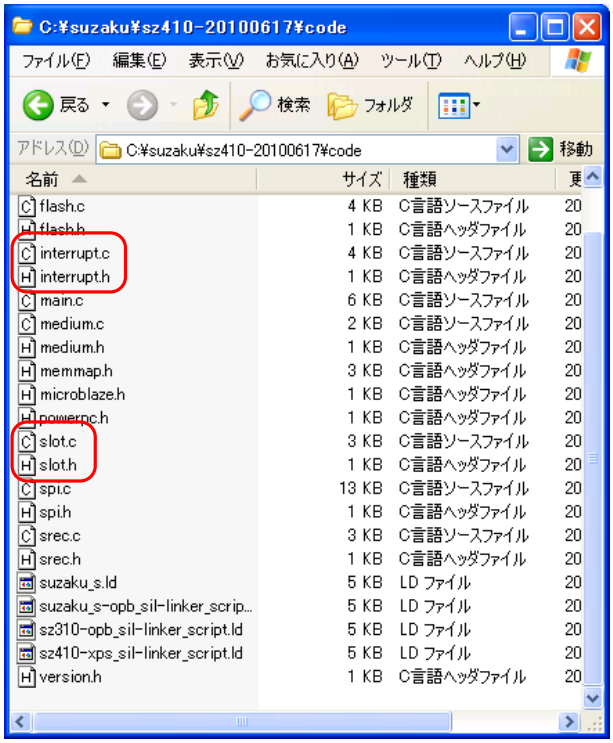


図 13.42 ソースファイル確認

[Applications]タブをクリックし、[Sources]を右クリックしてメニューを出し、[Add Existing Files...]を選択し、slot.c、interrupt.c を追加してください。

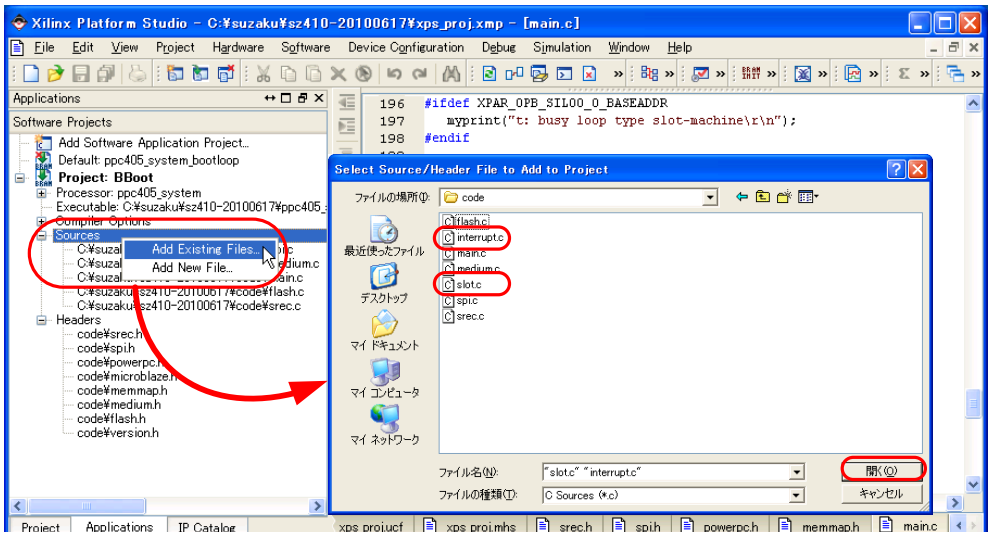


図 13.43 ソースファイル選択

[Headers] を右クリックしてメニューを出し、[Add Existing Files...] を選択し、`slot.h`、`interrupt.h` を追加してください。

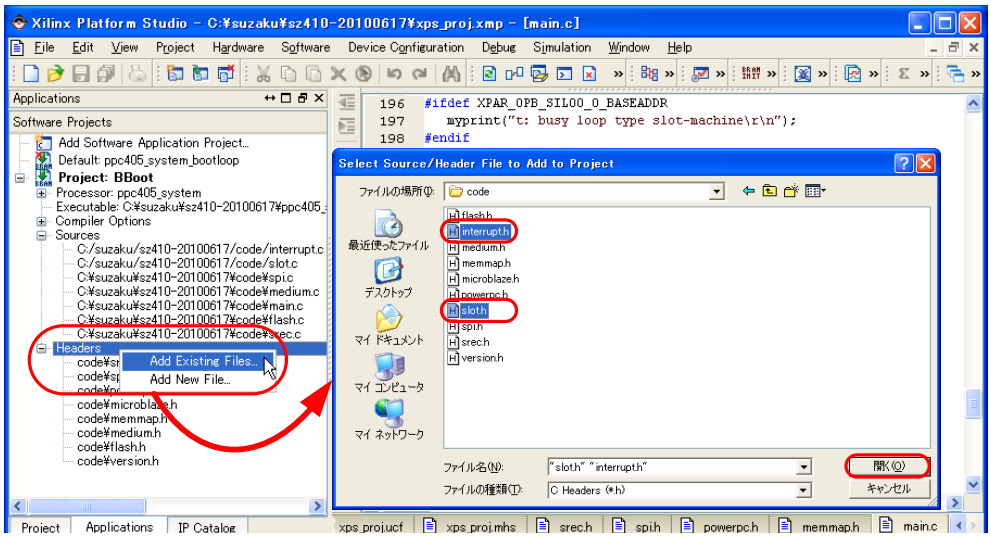


図 13.44 ヘッダファイル追加

13.2.4.3. ソースコード編集(main.c)

[Applications]タブをクリックしてください。[BBoot]→[Sources]の `main.c` をダブルクリックして開き、太字の部分を確認してください。

例 13.9 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"
#ifdef XPAR_OCM_TEMAC_SZ410_0_BASEADDR
#include "xpseudo_asm.h"
#endif

#ifdef  XPAR_OPB_SIL00U_0_BASEADDR      //自分の定義名と違う場合は変更
#define XPAR_OPB_SIL00_0_BASEADDR   XPAR_OPB_SIL00U_0_BASEADDR
#elif   XPAR_SIL_CNTLR_BASEADDR
#define XPAR_OPB_SIL00_0_BASEADDR   XPAR_SIL_CNTLR_BASEADDR
#endif

#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   SPI_REGISTER_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);

// 中略

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

    if (is_autoboot_mode()) {
        second_bootloader(bootloader_offset);
    }

#ifdef XPAR_OPB_SIL00_0_BASEADDR
        interrupt_init();       //割り込み許可

#else

  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(BBOOTMENU_WAITTIME);
  if (XUartLite_mIsReceiveEmpty(XPAR_CONSOLE_UART_BASEADDR) ||
```

```
                          ((key = get_char()) != 0 && key != 'z' && key != 'Z'))
      second_bootloader(bootloader_offset);
#endif

    /* 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");

#ifdef XPAR_OPB_SIL00_0_BASEADDR
    myprint("t: busy loop type slot-machine\r\n");
#endif

    while (1) {
        key = get_char();
        switch (key) {
            case 'a':    /* activate second stage bootloader */
            case 'A':
            case '\r':
            case '\n':

#ifdef XPAR_OPB_SIL00_0_BASEADDR
                interrupt_clean();      //割り込み禁止
#endif

                second_bootloader(bootloader_offset);
                break;
            case 's':
            case 'S':

#ifdef XPAR_OPB_SIL00_0_BASEADDR
                interrupt_clean();      //割り込み禁止
#endif

                myprint("Start sending S-Record!!\r\n");
                download();
                break;

#ifdef XPAR_OPB_SIL00_0_BASEADDR       //T が入力されたらビジーループでスロットマシンを実行
            case 't':
            case 'T':
                interrupt_clean();      //割り込み禁止
                myprint("busy loop type slot-machine\r\n");
                while (1) {
                     busy_wait(10000);
                     slot();
                }
#endif

#ifdef UART2_BASEADDR
            case 'u':
            case 'U':
                myprint("Check Uart2\n\r");
                while (XUartLite_mIsReceiveEmpty(UART2_BASEADDR));
                    XUartLite_SendB(UART2_BASEADDR,
                            (Xuint8)XUartLite_RecvB(UART2_BASEADDR));
                    myprint("Done\n\r");
```

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

}
```

13.2.4.4. コンパイラオプション変更

SZ130

SZ130 の場合コンパイラオプションを変更します。[Project: BBoot]→[Compiler Option]を右クリックしてメニューを出し、[Set Compiler Options]を選択してください。[Paths and Options]タブをクリックし、Other Compiler Options to Append の[-xl-mode-novectors]を削除し、[OK]をクリックしてください。-xl-mode-novectors を設定すると、ベクタの命令が含まれなくなり、コードサイズを小さくすることができます。BBoot ではベクタの命令は特に必要ないので、このオプションを設定しています。スロットマシンのアプリケーションでは割り込みベクタを使用するので、このオプションを削除します。

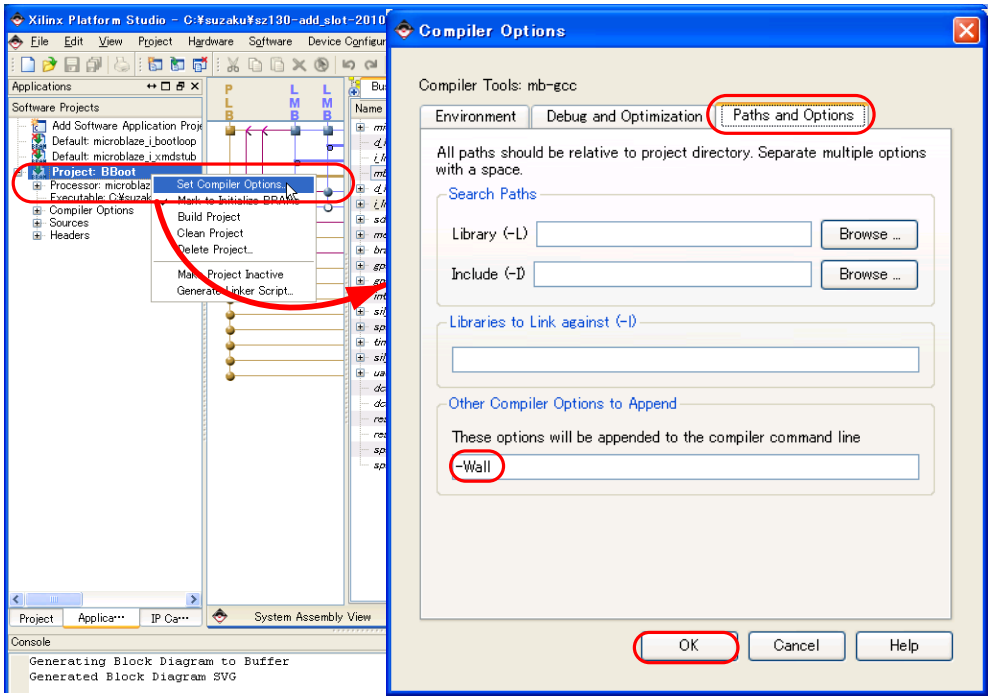


図 13.45 -xl-mode-novectors を削除

13.2.4.5. リンカースクリプト作成

SZ410

SZ410 の場合リンカースクリプトを作ります。[Software]→[Generate Linker Script...]を選択してください。

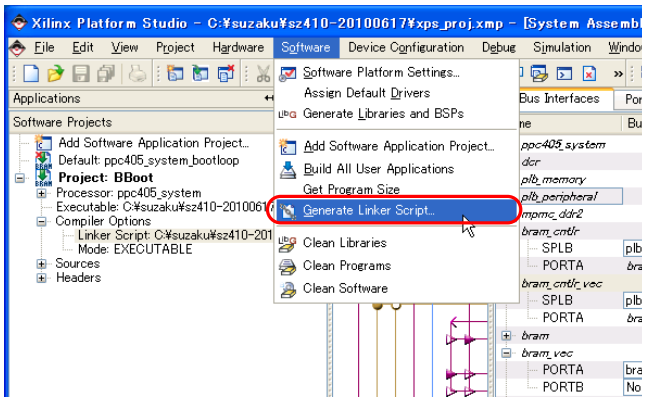


図 13.46 割り込み設定(リンカースクリプト)

[Sections View]の Memory の部分を編集し、[Output Linker Sctipt:]を["C:\suzaku\sz***-yyyyxxdd\code\BBoot_linker_script.ld"]に変更し[1]、[OK]をクリックして下さい。

.vectors を 0xFFFF0000 ～ 0xFFFF3FFF(追加した BRAM)側に割り当てます。また、text セクションが結構大きくなってしまうので、rodata、data、bss なども、0xFFFF0000 ～ 0xFFFF3FFF 側に割り当てて、必要容量を分散します。

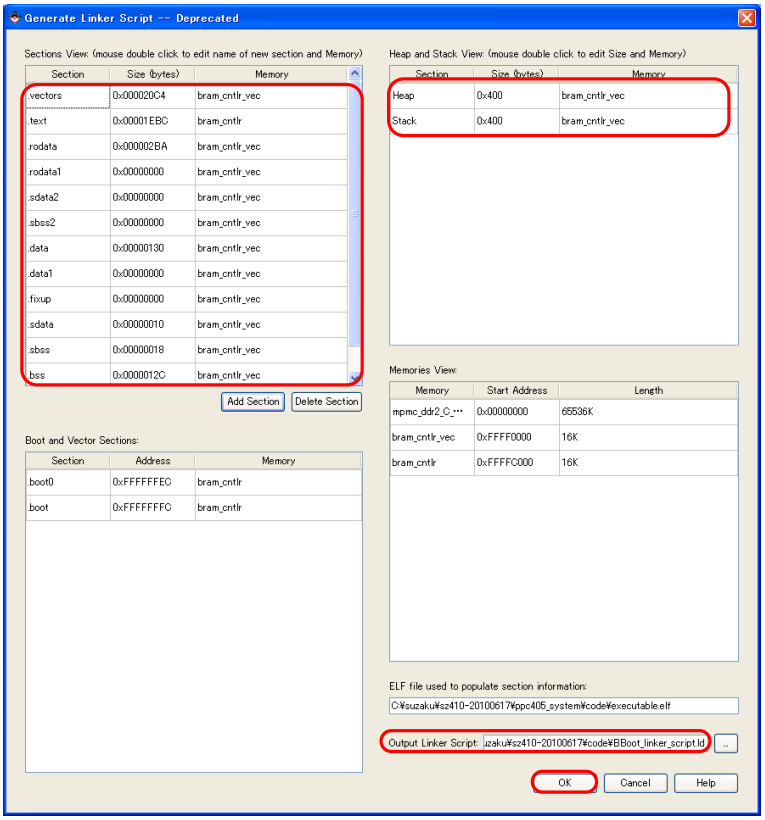


図 13.47 リンカースクリプト設定

[1]存在しないフォルダを指定するとエラーになるので、フォルダを事前に作成してください。

Section	Memory
.vecotrs	bram_cntlr_vec
.text	bram_cntlr
.rodata	bram_cntlr_vec
.rodata1	bram_cntlr_vec
.sdata2	bram_cntlr_vec
.sbss2	bram_cntlr_vec
.data	bram_cntlr_vec
.data1	bram_cntlr_vec
.fixup	bram_cntlr_vec
.sdata	bram_cntlr_vec
.sbss	bram_cntlr_vec
.bss	bram_cntlr_vec
.Heap	bram_cntlr_vec
.Stack	bram_cntlr_vec

13.2.5. アプリケーション生成

[Software]→[Build All User Applications]をクリックして下さい。コンパイラが起動され、各ソフトウェアアプリケーションのプログラムソースの設定が読み込まれます。エラーがなければ `executable.elf` が出来上がります。

13.2.6. プログラムファイル作成

ハードウェアでつくった bit ファイルの中にソフトウェアを書き込みます。[Device Configuration]→[Update Bitstream]をクリックしてください。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。

13.2.7. コンフィギュレーション

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。

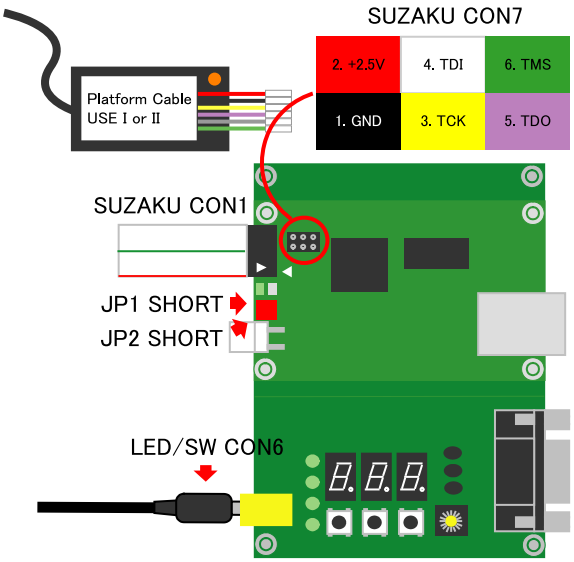


図 13.48 ジャンパの設定等

[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。

13.3. スロットマシン完成

以上でスロットマシンの完成です！

残念ながらうまくいかない場合は付属 CD-ROM に IP コア、FPGA プロジェクトを収録[2]しているので、比較してみてください。

13.3.1. 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。

[2]IP コアは`\suzaku-starter-kit\fpga\xps_sil00_vx_xx_x.zip"`、FPGA プロジェクトは`\suzaku-starter-kit\fpga\x.x\sz***\sz***-add_slot_yyyymmdd.zip"`に収録しています。

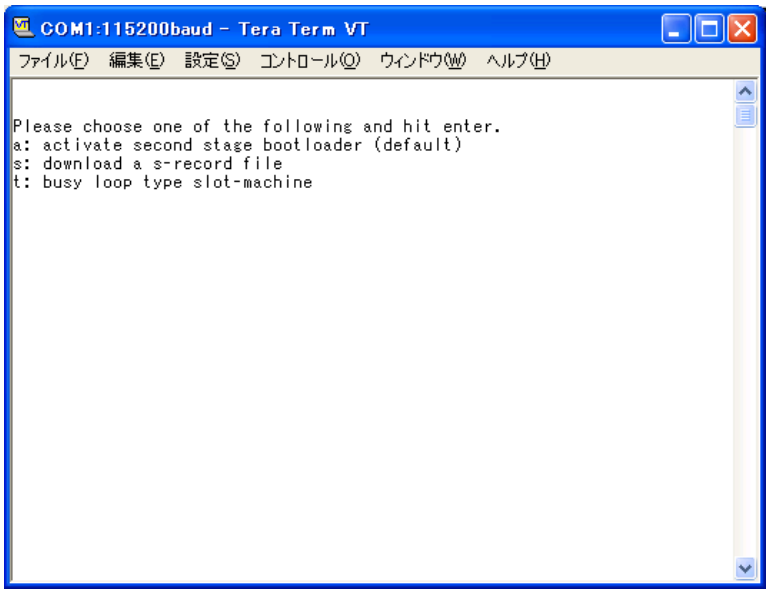


図 13.49 スロットマシン実行画面 1

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。

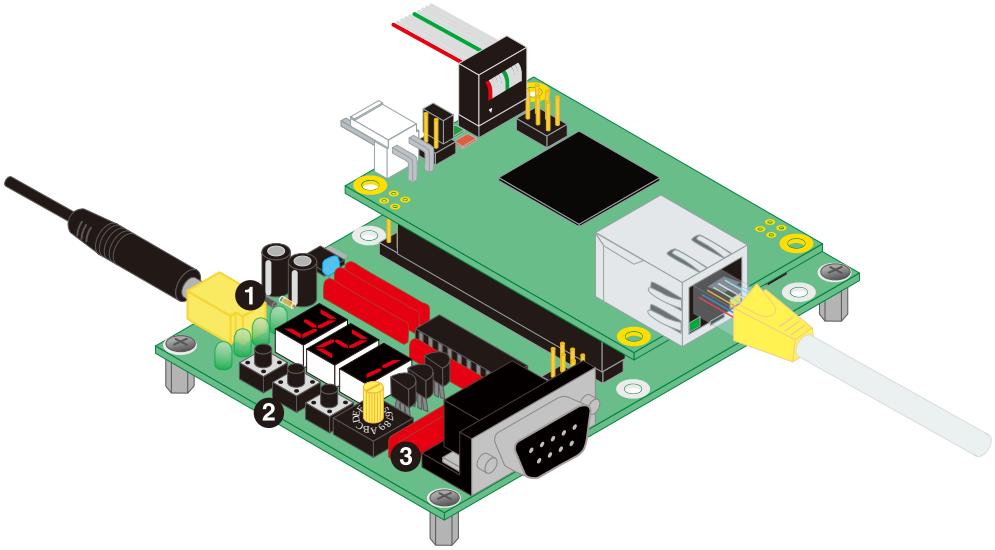


図 13.50 スロットマシン完成

❶　数字がそろうと単色 LED が順次点灯

❷　押しボタンスイッチを 2 つ以上一緒に押すとスロットの回転が始まり、1 つ押すとそれぞれに対応する 7 セグメント LED の回転が止まる

❸　ロータリコードスイッチを 0→1→2 とまわすと数字の回転が速くなる

13.4. ソフトウェアのデバッグ

XPS にはプロセッサと通信し、ソフトウェアをデバッグする機能がついています。ソフトウェアの処理の流れや変数の確認、メモリのダンプ、CPU レジスタのダンプなどが行え、ソフトウェアの開発にとても便利です。このデバッグ機能を用いてスロットマシンのプログラム `interrupt.c`、`slot.c` の動作を確認します。今回は、下記の 2 つの動作をステップ実行で確認します。

- 割込みが発生したときの流れ
- スロットの動作

13.4.1. ソフトウェアデバッグ用に IP コア追加

SZ130

まずは、デバッグが行えるようプロジェクトを更新します。SZ410 の場合はプロジェクトを更新しなくてもデバッグを行えるので、次のデバッガの設定に進んでください。

microblaze_i 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。[Debug]タブをクリックし、[Enable MicroBlaze Debug Module Interface]をチェックしてください。デバッグロジックがイネーブルになります。次にブレークポイント数を設定します。最大 8 まで設定することが出来ます。ここでは、[Number of PC Breakpoints]を 2 にします。設定できたら[OK]をクリックして下さい。

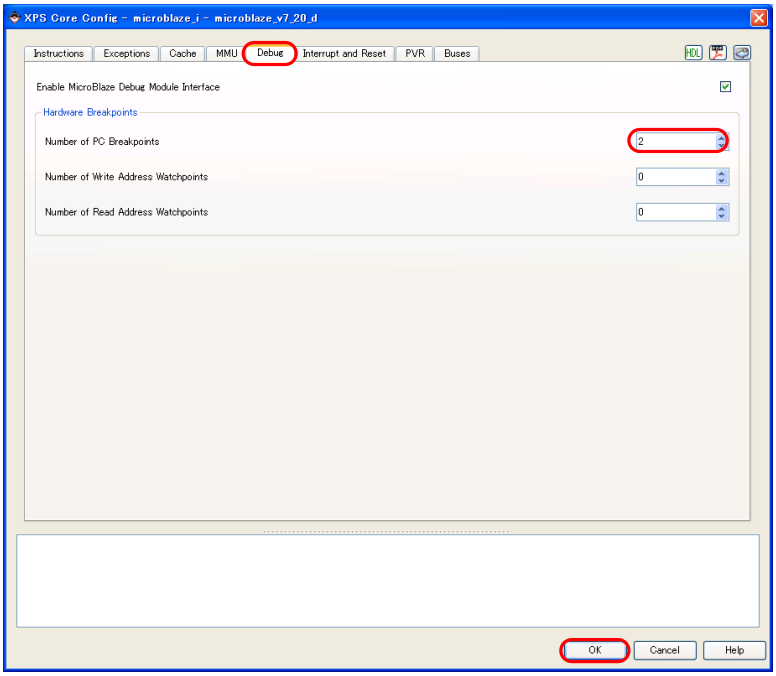


図 13.51 MicroBlaze のデバッグ設定

デバッグのための IP コアを追加します。[IP Catalog]タブをクリックし、[Debug]→[mdm 1.00g]を追加し、バスに接続してください。

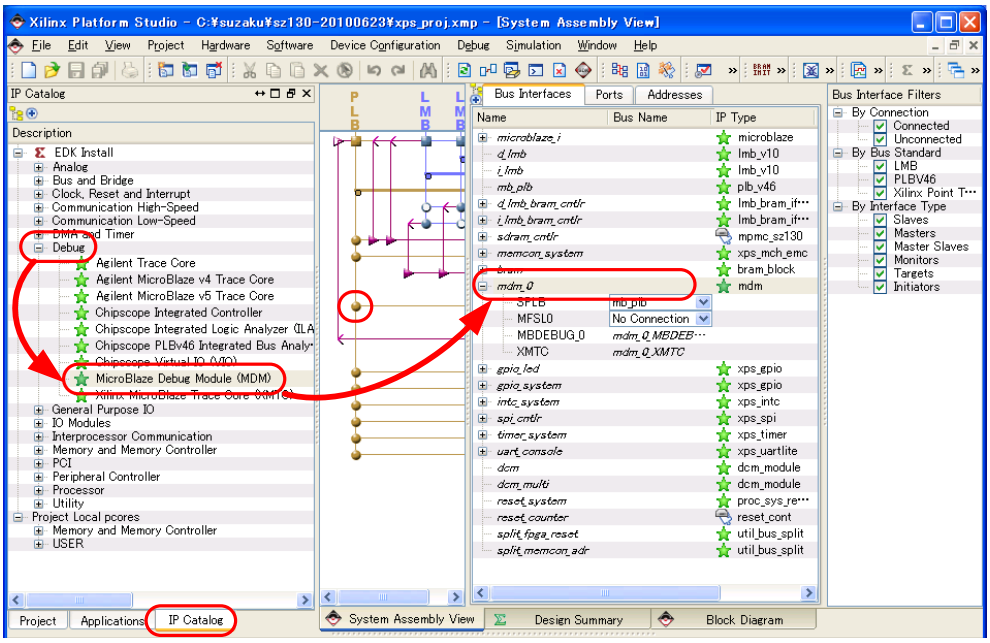


図 13.52 mdm を追加してバスに接続

mdm_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。[System]タブをクリックし、[Base Address]に[0xFFFFE000]、[High Address]に[0xFFFFE0FF]と入力し、[OK]をクリックして下さい。アドレスは SUZAKU のメモリマップの Free のところならばどこでも構いません。(「1.5. メモリマップ」参照)

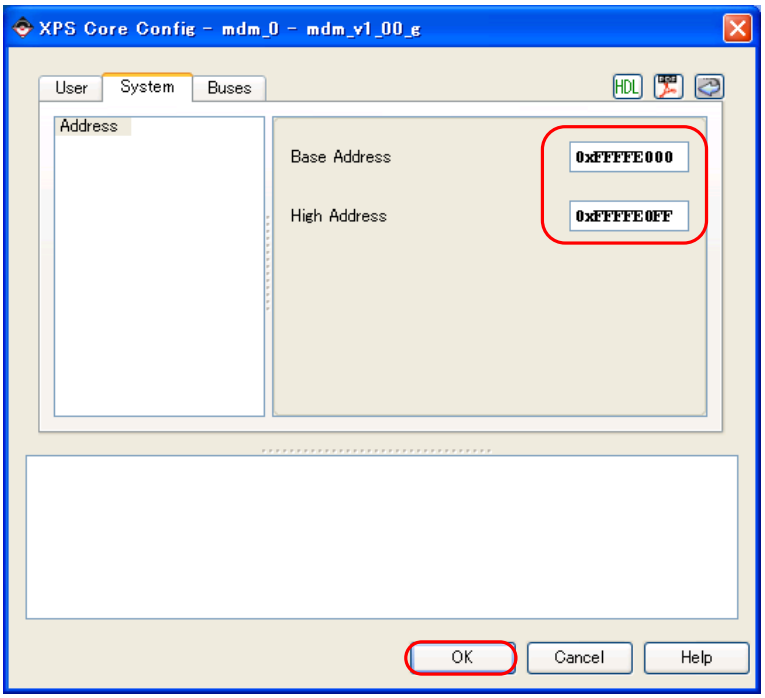


図 13.53 デバッガのアドレス設定

デバッガを MicroBlaze と接続します。[Bus Interfaces]タブをクリックし、microblaze_i の DEBUG の Bus Name をクリックし、[mdm_0_MBDEBUG_0]を選択してください。

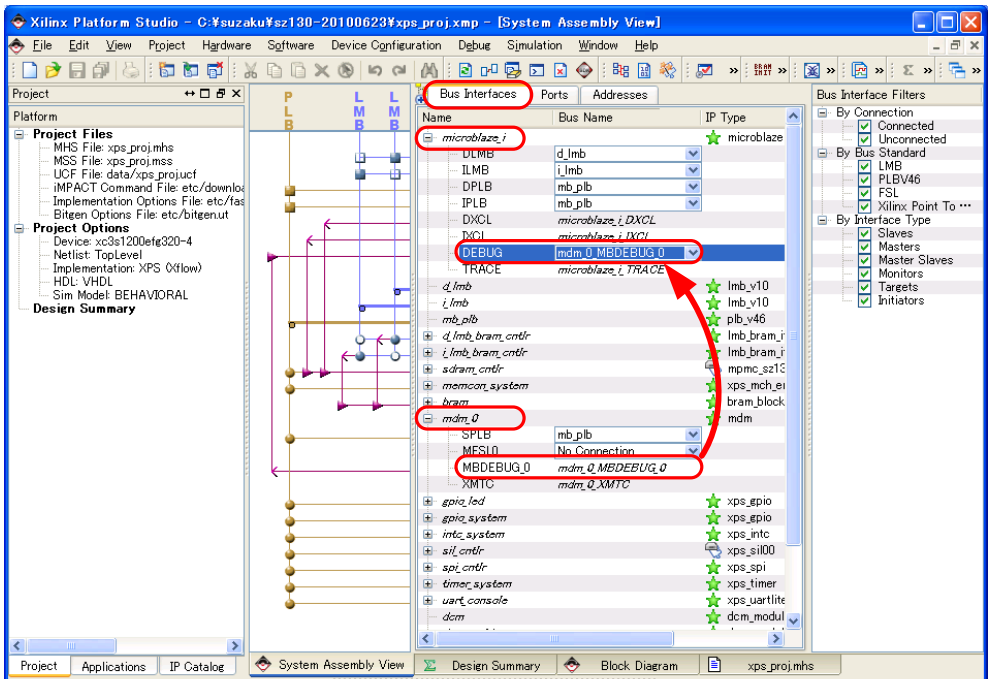


図 13.54 Microblaze とデバッガを接続する

デバッガにリセット信号を接続します。[Ports]タブをクリックし、mdm_0 の Debug_SYS_Rst の Net をクリックし、[reset_system_MB_Debug_Sys_Rst]を選択してください。

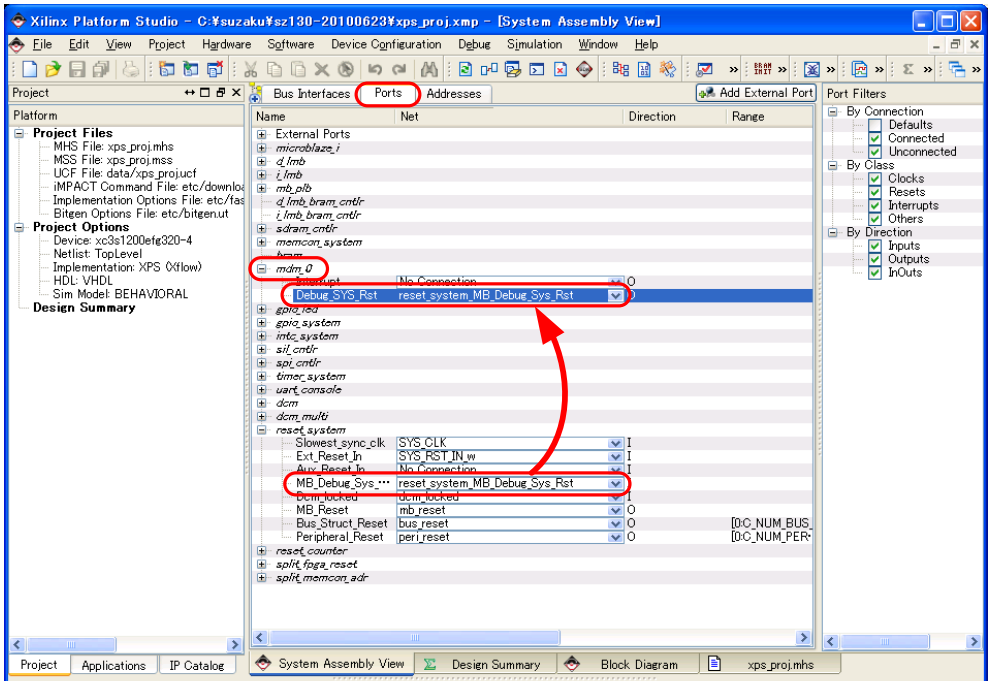


図 13.55 デバッガにリセット信号を接続

13.4.2. デバッガの設定

[Applications]タブをクリックし、[Project: BBoot]→[Compiler Option]を右クリックしてメニューを出し、[Set Compiler Options]を選択してください。[Debug and Optimization]タブをクリックし、

最適化をしないので、[Optimization Level]を[No Optimization]にしてください。[Generate Debug Symbols] 、[Create Symbols for Debugging] がチェックされているのを確認し、[OK]をクリックしてください。

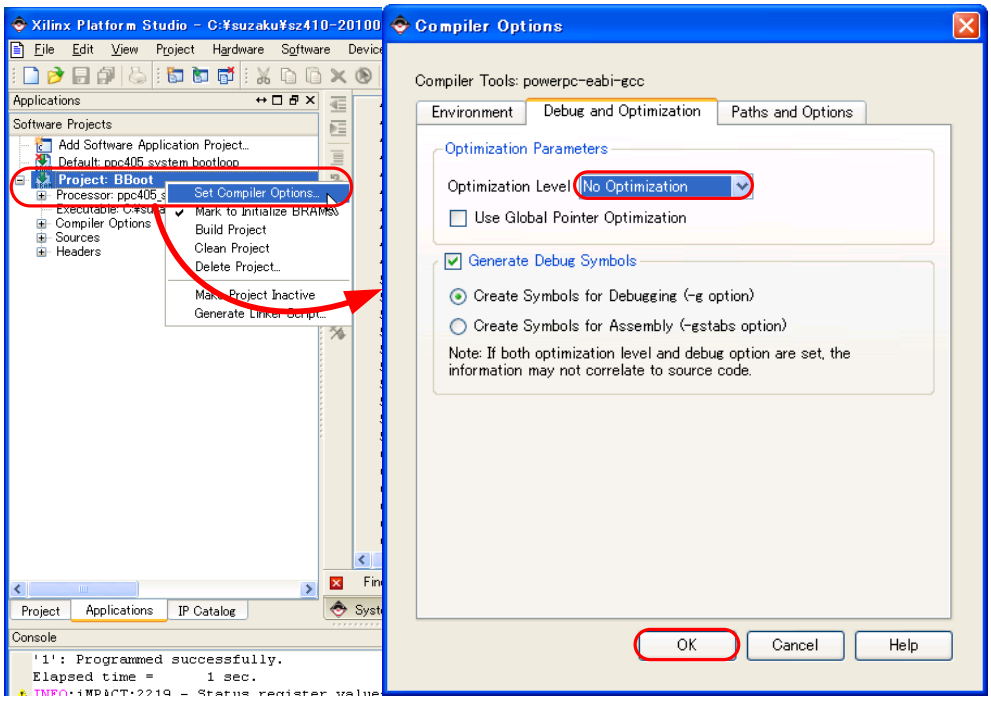


図 13.56 最適化なしに変更

[Debug]→[XMD Debug Options] をクリックしてください。[Connection Type]を[Hardware]、[JTAG Cable]の Type を[Auto]にし、[Auto Discover JTAG Chain Definition]をチェックしてください。[JTAG Cable]の Type はお使いのケーブルを選んでいただいてもかまいません。設定できたら[OK]をクリックし、設定を保存してください。

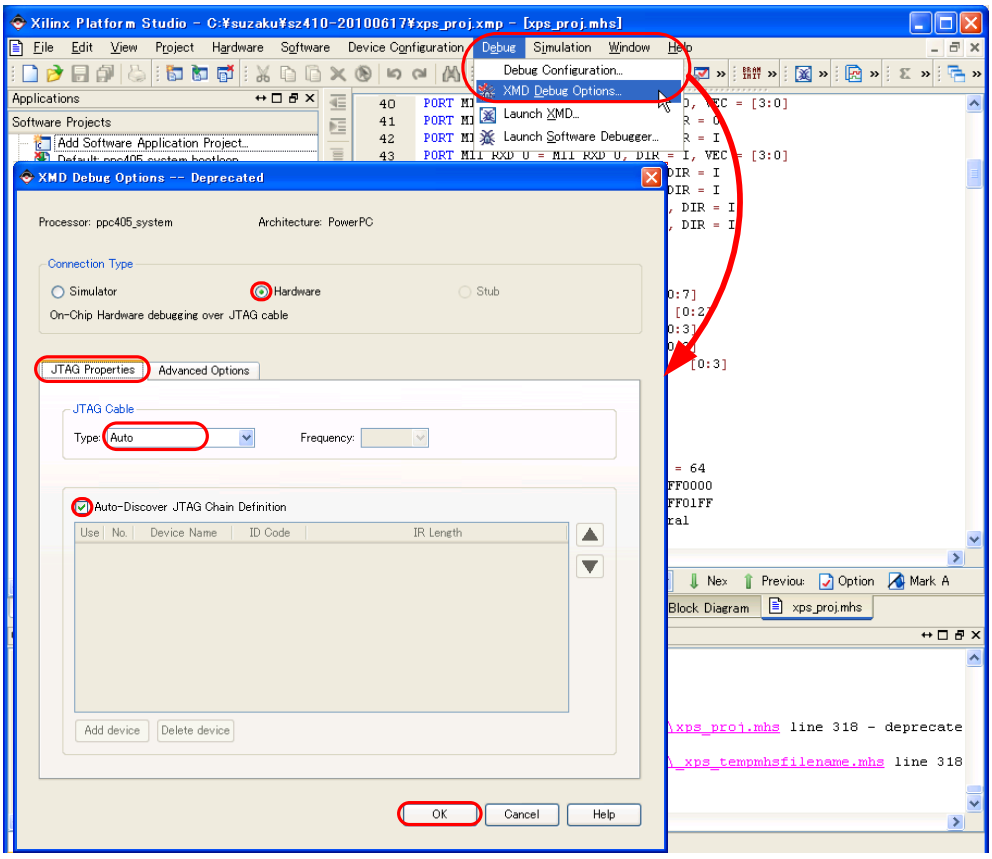


図 13.57 デバッグオプション

以上でプロジェクトの更新は終わりです。

[Device Configuration]→[Update Bitstream]をクリックしてください。

13.4.3. XMD の起動

XMD とは、プロセッサ(MicroBlaze や PowerPC)と PC のデバッグアプリケーションの仲立ちをしてくれるものです。XMD とデバッグアプリケーションは、TCP 経由でやり取りをしますので、ネットワーク経由でも接続できます。

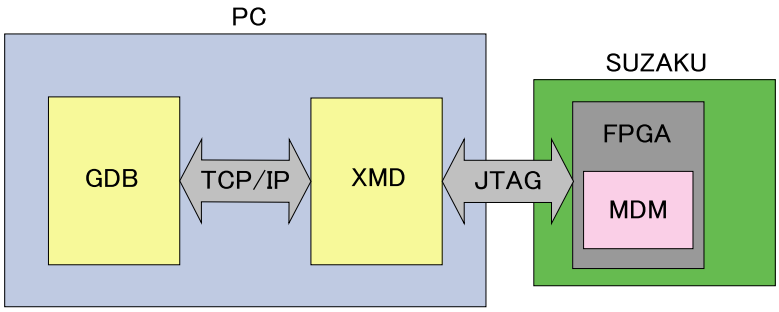


図 13.58 XMD の接続

[Device Configuration]→[Download Bitstream]をクリックして下さい。FPGA にデバッグ機能付きのスロットマシンのコンフィギュレーションデータがダウンロードされます。

[Debug]→[Launch XMD...]をクリックして下さい。コマンドプロンプトが立ち上がり、以下のように表示されます。

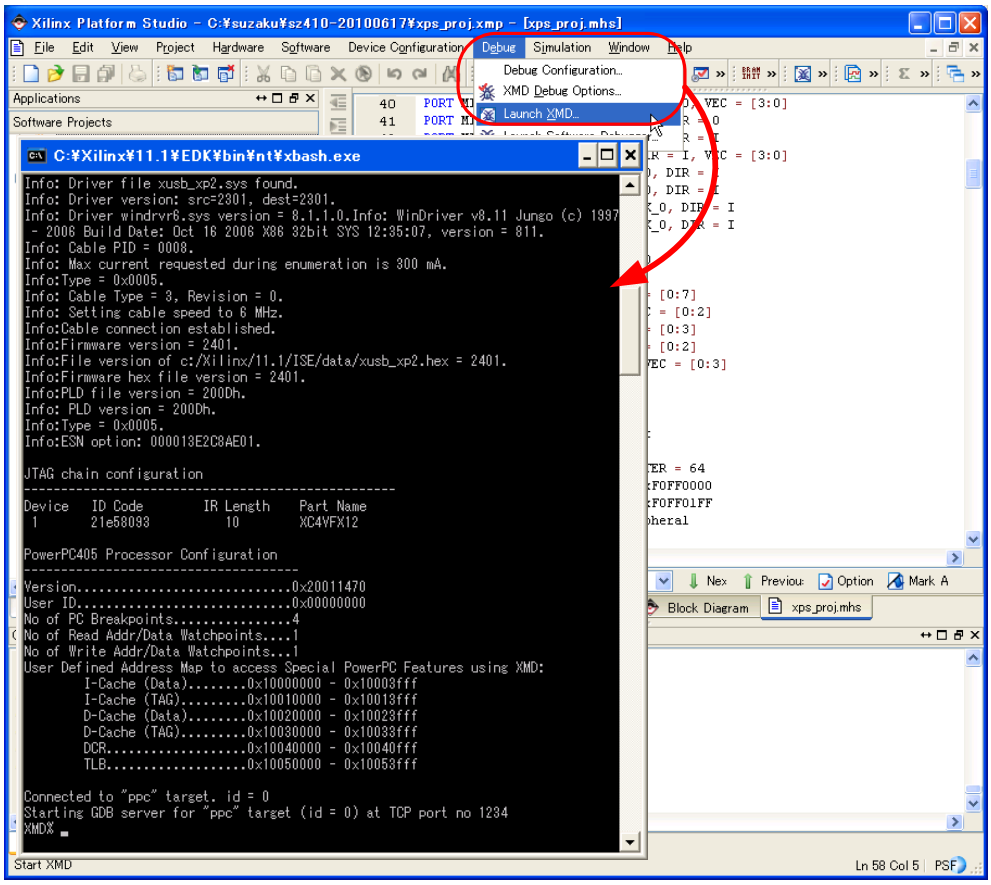


図 13.59 デバッガの起動

例 13.10 XMD の起動ログ(SZ410 の場合)

```
Xilinx Microprocessor Debugger (XMD) Engine
Xilinx EDK 11.5 Build EDK_LS5.70
Copyright (c) 1995-2009 Xilinx, Inc.  All rights reserved.
Release 11.5 - psf2Edward EDK_LS5.70 (nt)
Copyright (c) 1995-2009 Xilinx, Inc.  All rights reserved.

中略

XMD%
Info:AutoDetecting cable. Please wait.
Info:Connecting to cable (Usb Port - USB21).
Info:Checking cable driver.
Info: Driver file xusb_xp2.sys found.
Info: Driver version: src=2301, dest=2301.
Info: Driver windrvr6.sys version = 8.1.1.0.Info: WinDriver v8.11 Jungo (c) 1997
 - 2006 Build Date: Oct 16 2006 X86 32bit SYS 12:35:07, version = 811.
Info: Cable PID = 0008.
Info: Max current requested during enumeration is 300 mA.
Info:Type = 0x0005.
Info: Cable Type = 3, Revision = 0.
```

```
Info: Setting cable speed to 6 MHz.
Info:Cable connection established.
Info:Firmware version = 2401.
Info:File version of c:/Xilinx/11.1/ISE/data/xusb_xp2.hex = 2401.
Info:Firmware hex file version = 2401.
Info:PLD file version = 200Dh.
Info: PLD version = 200Dh.
Info:Type = 0x0005.
Info:ESN option: 000013E2C8AE01.

JTAG chain configuration
-----------------------------------------------
Device   ID Code        IR Length    Part Name
 1       21e58093          10        XC4VFX12

PowerPC405 Processor Configuration
-------------------------------------
Version..............................0x20011470
User ID..............................0x00000000
No of PC Breakpoints.................4
No of Read Addr/Data Watchpoints....1
No of Write Addr/Data Watchpoints...1
User Defined Address Map to access Special PowerPC Features using XMD:
        I-Cache (Data)........0x10000000 - 0x10003fff
        I-Cache (TAG).........0x10010000 - 0x10013fff
        D-Cache (Data)........0x10020000 - 0x10023fff
        D-Cache (TAG).........0x10030000 - 0x10033fff
        DCR...................0x10040000 - 0x10040fff
        TLB...................0x10050000 - 0x10053fff

Connected to "ppc" target. id = 0
Starting GDB server for "ppc" target (id = 0) at TCP port no 1234
XMD%
```

13.4.4. GDB を起動し、ソフトウェアをスタートさせる

GDB はソフトウェアデバッグのユーザインターフェースになります。まずは GDB を起動します。[Debug]→[Launch Software Debugger]をクリックして下さい。以下の画面が立ち上がります。ここではデフォルト設定のままとします。そのまま[OK]をクリックして下さい。立ち上がらない場合は[File]→[Target Settings...]から以下の画面を立ち上げることが出来ます。

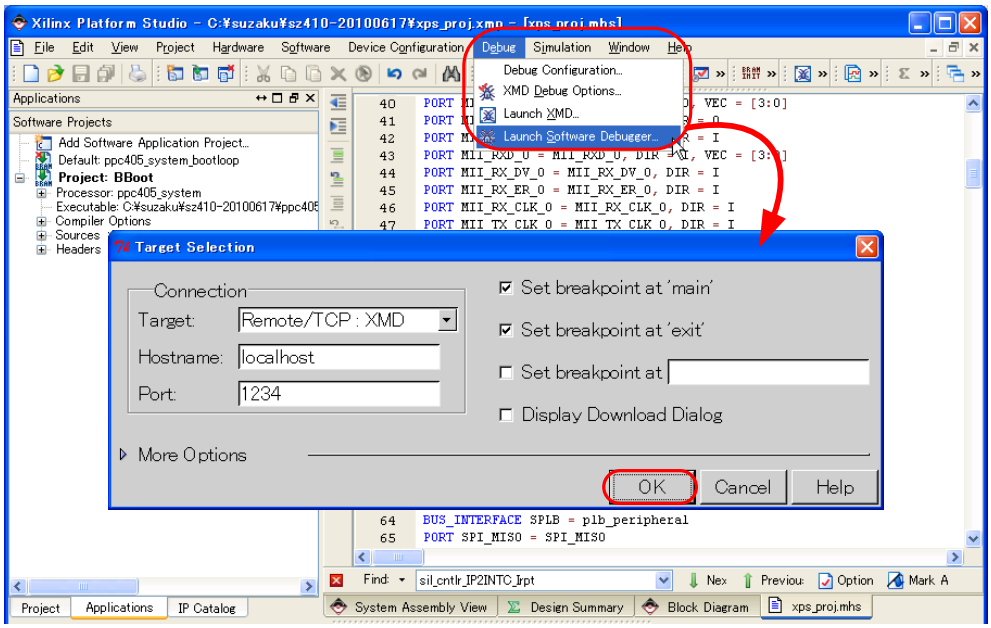


図 13.60 デバッグ設定

以下の画面が立ち上がります。[Run]→[Run]をクリックして下さい。main 関数で Break します。LED_GPIO(LED_OFF)が緑色にハイライトされるとおもいます。

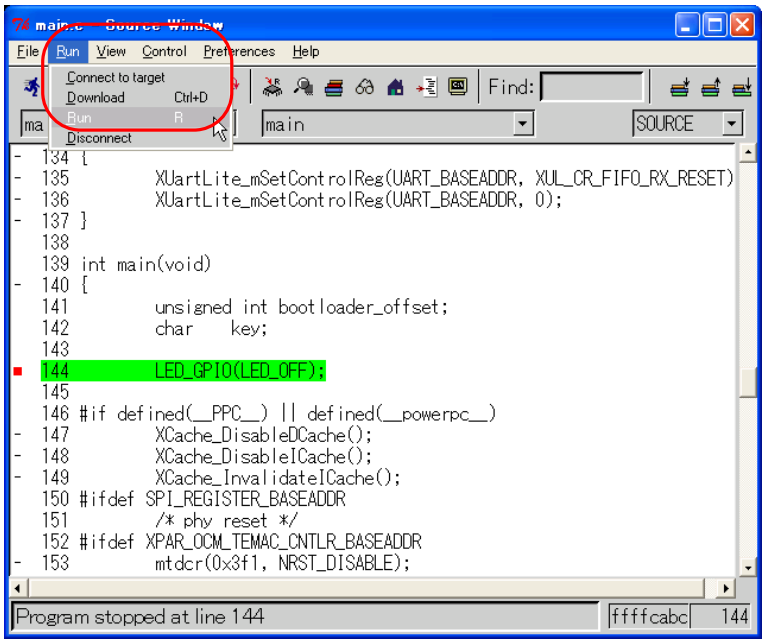


図 13.61 main で Break

13.4.5. ステップ実行で割り込みの流れをみる

[View]→[Breakpoints]をクリックして下さい。現在の Breakpoint が表示されます。main と exit に Breakpoint が設定されています。Breakpoint は割り込みベクタだけにしたいので、[Global]→[Remove All]をクリックし、Breakpoint を消去してください。

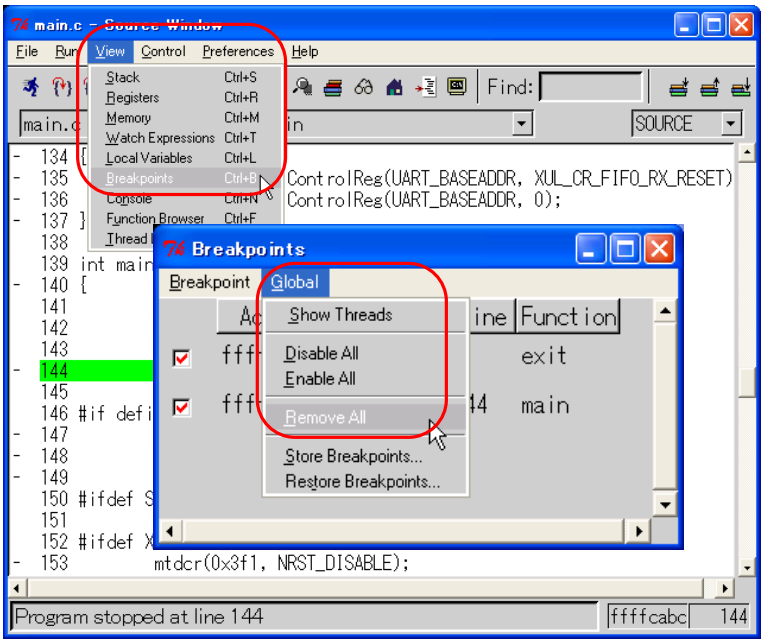


図 13.62 Breakpoint 設定

MicroBlaze の割り込みベクタは 0x00000010、PowerPC の割り込みベクタは 0xFFFF0500 です。ここに Breakpoint を設定します。XMD で以下のコマンドを実行してください。

例 13.11 Breakpoint 設定(SZ130 の場合)

```
XMD% bps 0x00000010 hw
```

例 13.12 Breakpoint 設定(SZ410 の場合)

```
XMD% bps 0xFFFF0500 hw
```

XMD コマンド

コマンド	使用例	説明
bps <address/function> <hw/sw>	bps 0x10 sw bps main hw	address または function の開始部分にハードウェアまたはソフトウェアブレークポイントを設定
bpr <address/function/all>	bpr 0x10 bpr main bpr all	ブレークポイントを削除
bpl	bpl	現在のブレークポイントを表示

[Control]→[Continue]をクリックしてください。割り込みベクタで Break します。MicroBlaze の場合、割込みが入ると割込みベクタ 0x00000010 にジャンプします。PowerPC405 の場合、割り込みが入ると割り込みベクタ 0xFFFF0500 にジャンプします。これらのアドレスは、FPGA 内部の BRAM の領域です。

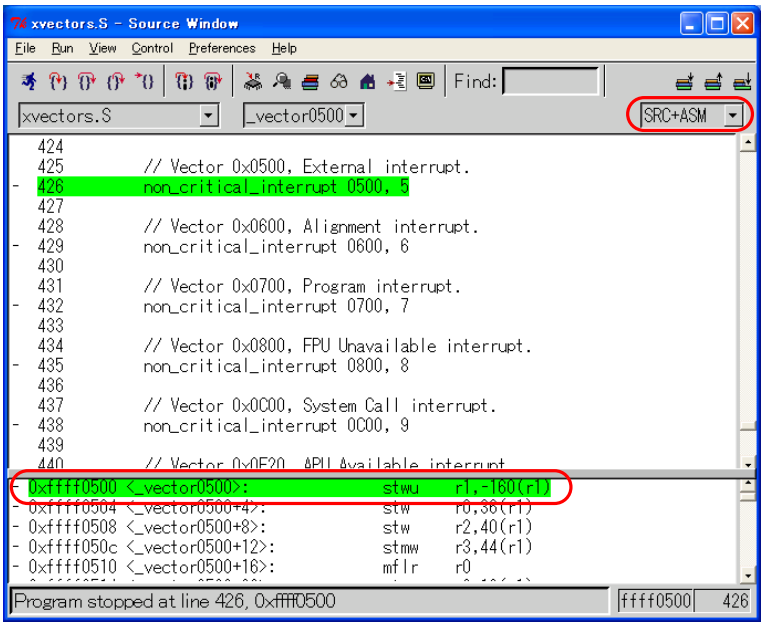


図 13.63 割り込みベクタで Break

ここから[Step]で順に動きを見ます。[Control]→[Step]を押してください。MicroBlaze の場合、割込みハンドラ `_interrupt_handler()` で Break します。`_interrupt_handler()` には、INTERRUPT に接続されているコア (SUZAKU では XPS INTC) の割込みハンドラ (`XIntc_DeviceInterruptHandler()`)へジャンプする命令が記述されています。PowerPC405 の場合、`XIntc_DeviceInterruptHandler()`で Break します。`XIntc_DeviceInterruptHandler()` には、割り込みコントローラに接続されているコア達の中から、実際に割込みを発生させたコアを見つけ、そのコアの割込みハンドラへのジャンプする命令が記述されています。

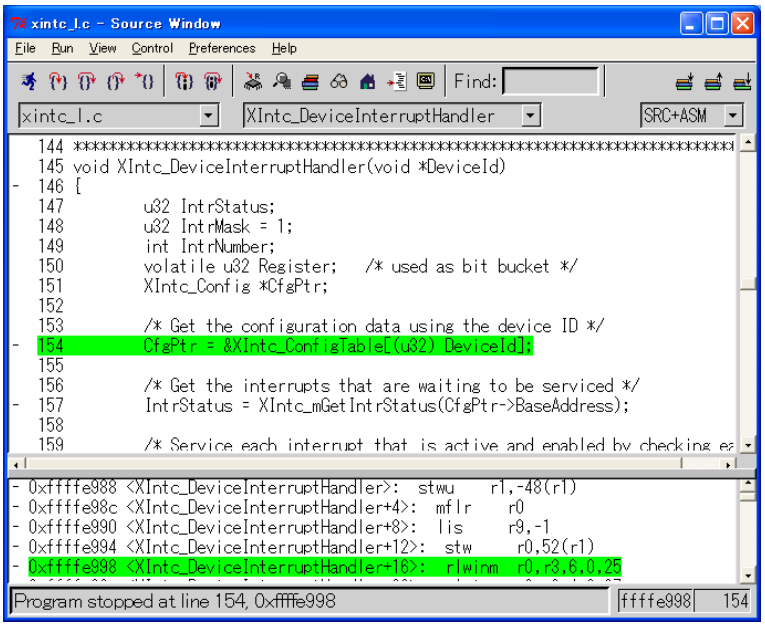


図 13.64 割り込みハンドラで Break

スロットマシンの場合 Default Handler ではなく、`timer_interrupt_handler()`を使用しています。何度か[Control]→[Step]を押してください。タイマー割り込みで Break します。中々タイマー割り込みで Break しない場合、`interrupt.c` を開いてタイマー割り込みに Breakpoint を追加して、[Control]→[Continue]をクリックして下さい。

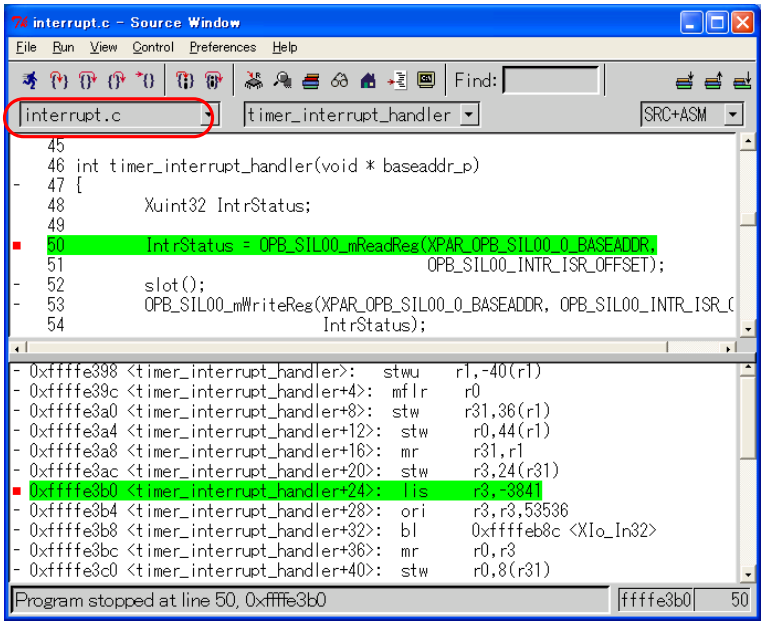


図 13.65 タイマー割り込みで Break

13.4.6. slot.c の動作を確認してみる

何度か[Control]→[Step]を押してください。`slot.c` で Break します。

[View]→[Loval Variables]をクリックして下さい。ローカル変数の一覧が表示されます。[View]→[Stack]をクリックして下さい。現在スタックしている関数の一覧が表示されます。ステップ実行やその

他メニューの[View]で開くことのできる各種情報を元に、ソースコードと合わせてプログラムの流れやローカル変数を参照し、スロットの動作を確認してください。

　かなりのステップ実行をしなくてはいけませんが、7 セグメント LED に 3→2→1 と数字が表示される様子を見ることが出来ます。また、slot.c の一通りの動作が終わると、スタックにしたがって、関数が戻っていく様子を見ることが出来ます。

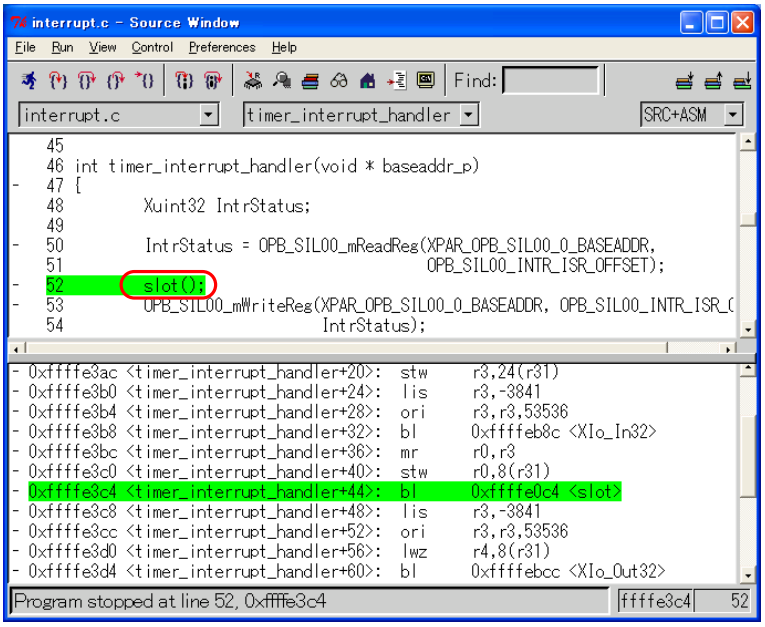


図 13.66 スロットで Break

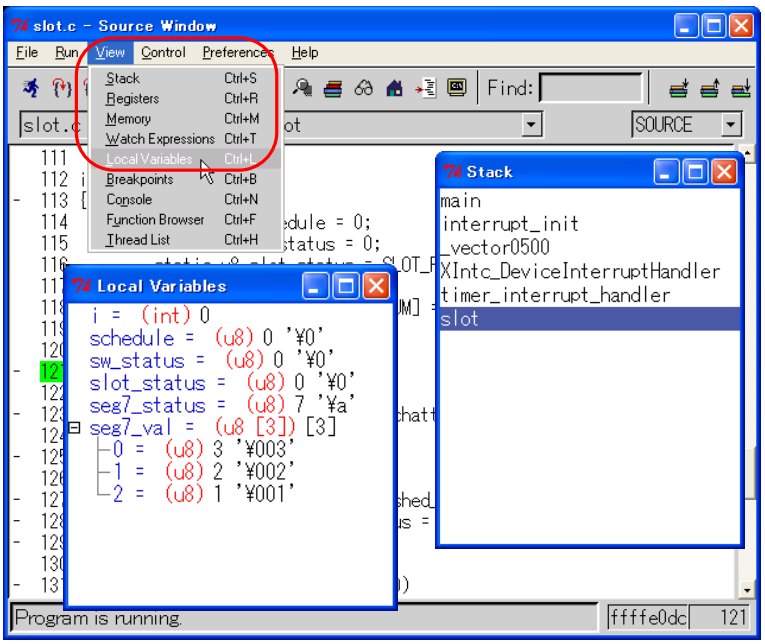


図 13.67 ローカル変数やスタックの一覧

14.こんなこともやってみよう

最後にこんなこともできますというのを紹介します。

14.1. XPS デザインを ISE のサブモジュールとして読み込む

ここでは XPS デザインを ISE のサブモジュールとして読み込む方法を説明します。

SUZAKU は XPS だけで構築されています。XPS だけで作業しても良いのですが、XPS だけで作業すると、自分で書いたロジックを追加したい時に、XPS に読み込めるようにしなければいけません。自分のロジックのほかにロジック追加用の設定ファイル(mpd ファイル、pao ファイル)を書かなければいけなかったり、Xilinx の命名規則にのっとっていなければならなかったりします。XPS に読み込めるようにしておけば、再利用しやすいというメリットはあるのですが、少々面倒です。

XPS と ISE は連携しており、XPS で構築生成させたネットリストを ISE でサブモジュールとして読み込ませることができます。この機能を使えば、XPS で GPIO や簡単な PLB インターフェースだけを用意しておき、XPS の External 設定することで、ISE でこれらの信号をサブモジュールの入出力として取り扱えるようになり、ISE で作りこんだ自分のロジックへ容易に組み込むことができます。

さらに、自分のロジックのみに変更があった場合、開発フローの ISE のみを実行するだけですむので、コンパイル時間を半減させることができます。また、ISE は配置配線ツールや制約ツール、タイミング解析ツールなど GUI で設定することができ分かりやすいです。

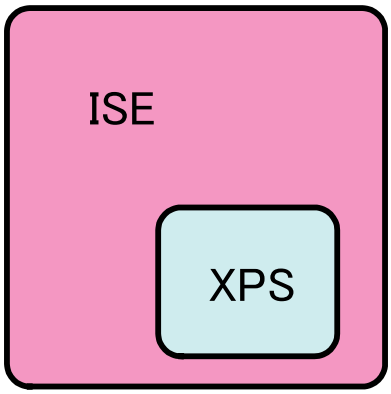


図 14.1 ISE で XPS をとりこむ

14.1.1. XPS で作業

例として SUZAKU のデフォルトの XPS プロジェクトを ISE に取り込みます。

付属 CD-ROM の"\suzaku\fpga_proj\x.x\sz***\sz***-yyyymmdd.zip"をハードディスクに展開してください。適当な名前のフォルダを作り、フォルダの下に、展開した XPS フォルダをコピーしてください。ここでは"C:\suzaku\suzaku-ise\sz***-yyyymmdd"にコピーして作業を進めます。XPS を起動し、"C:\suzaku\suzaku-ise\sz***-yyyymmdd\xps_proj.xmp"を開いてください。SUZAKU のデフォルトが開きます。

[Hardware]→[Generate Netlist]をクリックし、ネットリストを作成してください。ネットリストを作成すると、"C:\suzaku\suzaku-ise\sz***-yyyymmdd\hdl"に xps_proj_stub.vhd というファイルが出来上がります。

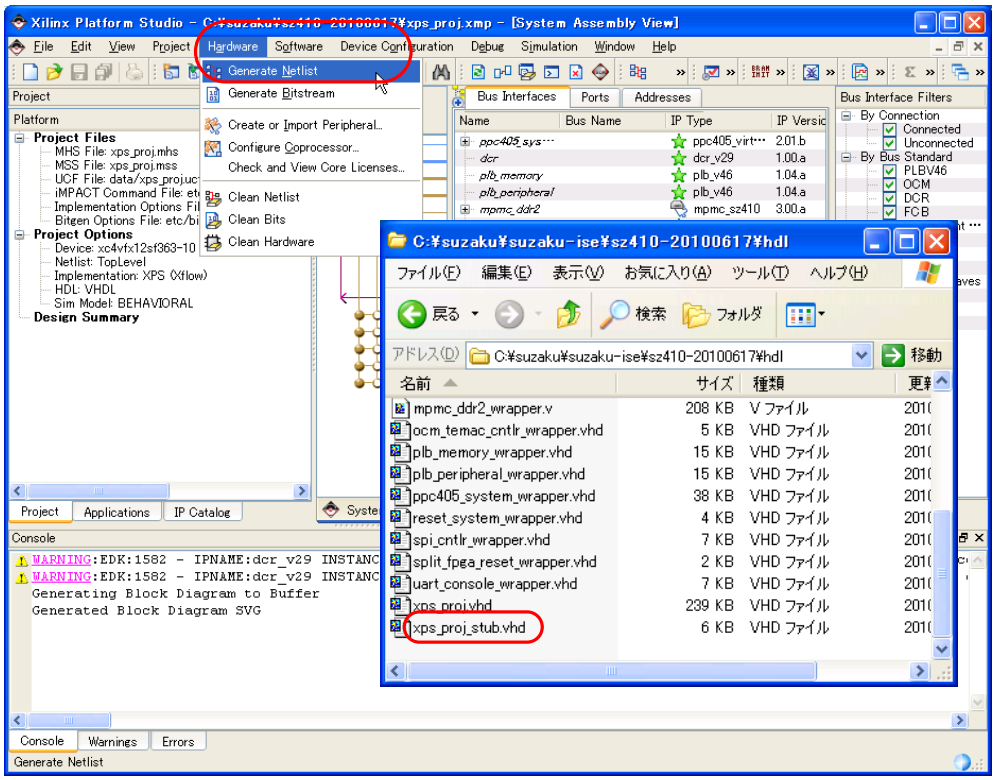


図 14.2 xps_proj_stub.vhd を作成

14.1.2. XPS から ISE へ移行

この xps_proj_stub.vhd を"C:\suzaku\suzaku-ise"以下にコピーしてください。この際、この名前のままだと少し分かりづらいので、ファイルの名前を top.vhd に変更します。(もちろん変えなくても良いです。。)

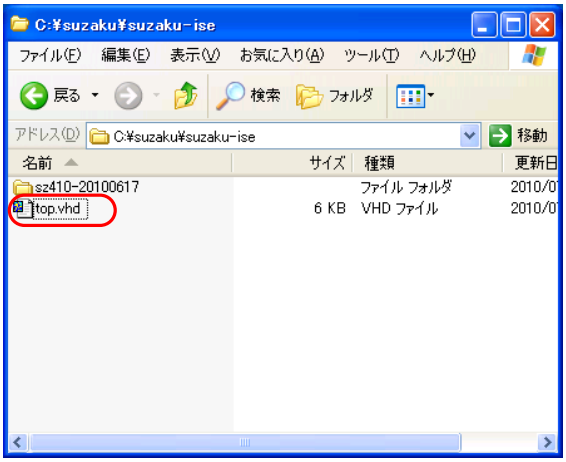


図 14.3 xps_proj_stub.vhd をコピー

14.1.3. ISE で作業

14.1.3.1. 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。

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\suzaku-ise` となります。[Project Name]にプロジェクト名を入力します。top 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。

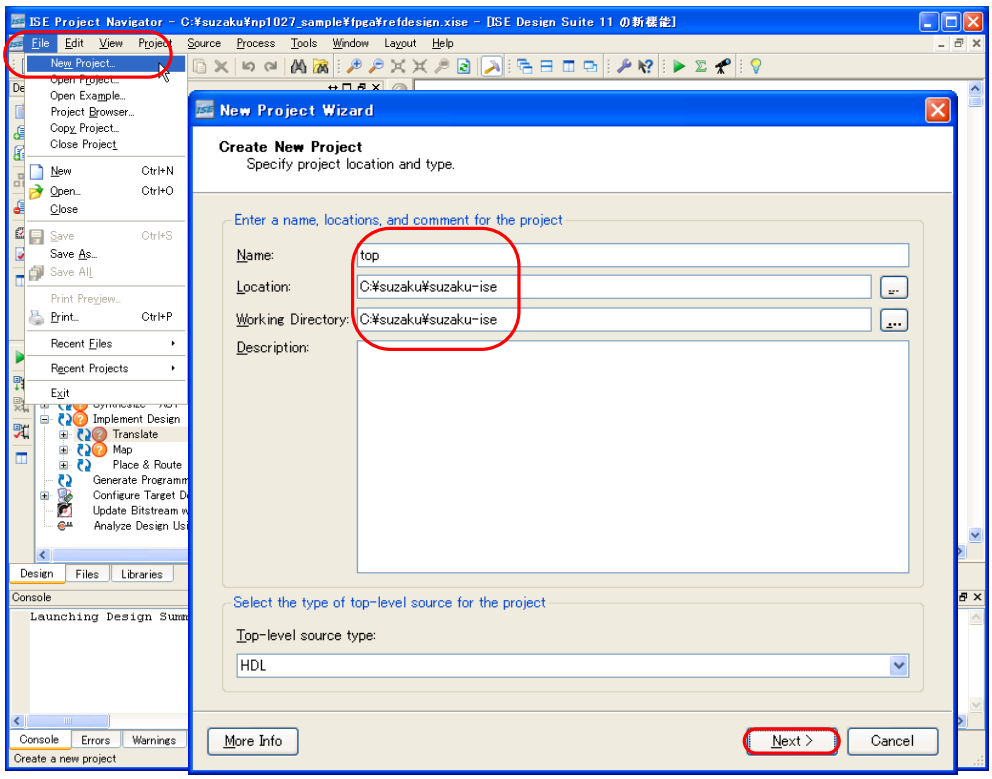


図 14.4 Project Navigator 起動

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	SZ130	SZ410
Product Category	ALL	
Family	Spartan3E	Virtex4
Device	XC3S1200E	XC4VFX12
Package	FG320	SF363
Speed	-4	-10
Synthesis Tool	XST(VHDL/Verilog)	
Simulator	ISim(VHDL/Verilog)	

この後は特に設定の変更をしないので、[Next]、[Finish]をクリックし、ウィザードを終了してください。

`top.vhd` と `xps_proj.xmp` と `xps_proj.ucf` をプロジェクトに追加します。[Project]→[Add Source]をクリックしてファイルを追加してください。`xps_proj.xmp` は`"C:\suzaku\suzaku-ise\sz***-yyyymmdd"` の下に `xps_proj.ucf` は `"C:\suzaku\suzaku-ise\sz***-yyyymmdd\data"`の下にあります。

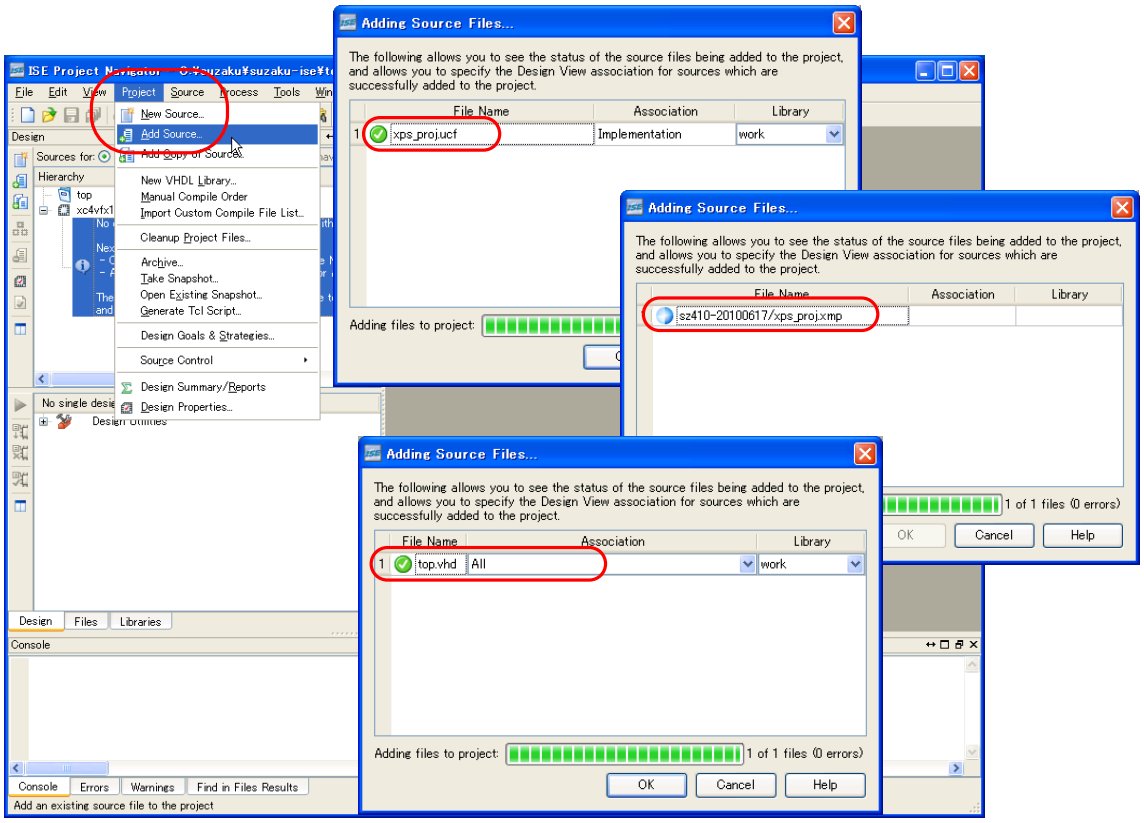
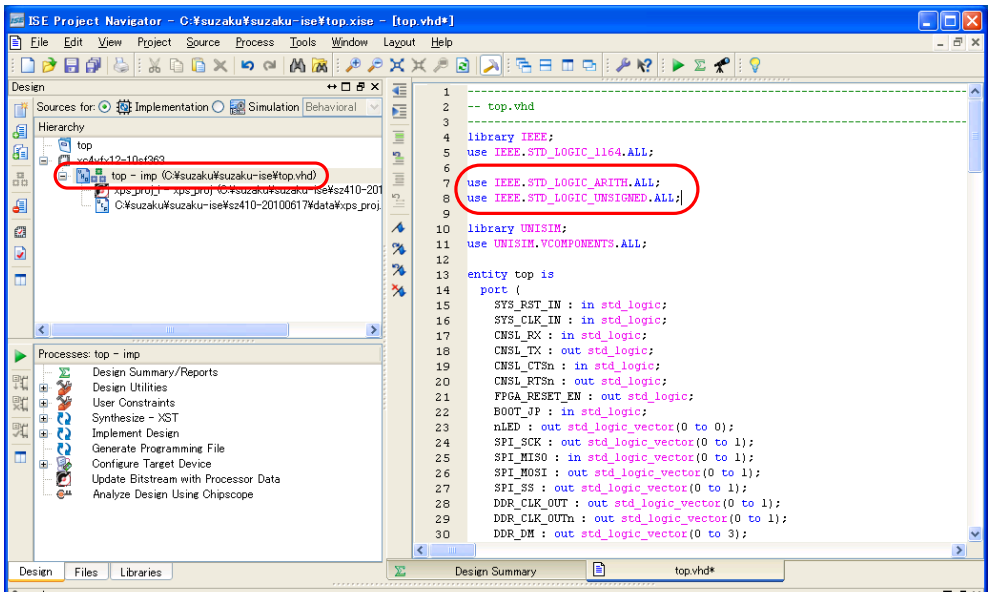


図 14.5 ソースファイル追加

top.vhd をダブルクリックして開いてください。ライブラリの追加を行います。また、少し分かりづらいので、STRUCTURE となっているアーキテクチャ名を imp に変更し(2 箇所)、xps_proj_stub となっているエンティティ名を top に変更します(3 箇所)。(ライブラリの追加は必須です。名前の変更は必須ではありません。。)変更ができたら[File]→[Save]をクリックし、保存してください。



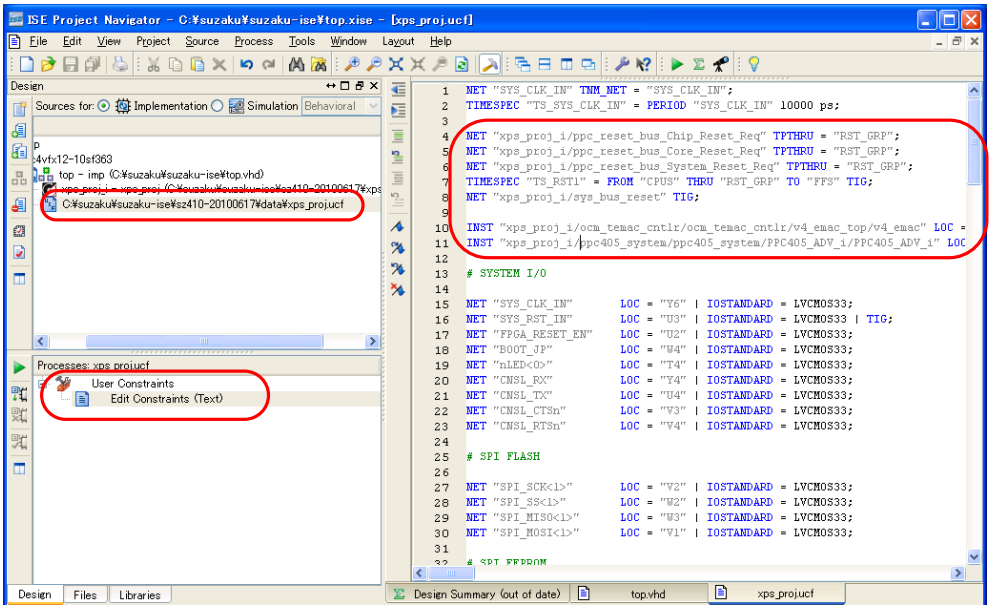

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;      -- ライブラリ追加
use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;
use UNISIM.VCOMPONENTS.ALL;

entity top is
  port (
    --中略
  );
end top;

architecture IMP of top is
  component xps_proj is
  port (
    --中略
  );
  end component;
begin
  xps_proj_i : xps_proj
    port map (
    --中略
    );
end architecture IMP;
```

図 14.6 ソースファイル編集(top.vhd)

14.1.3.2. UCF ファイルの修正

SZ410

SZ410 の場合、デザインの構造が少し変わったので、ucf ファイルに記述しているネット名を修正する必要があります。xps_proj.ucf をクリックし、Edit Constraints(Text)をダブルクリックして開き、6 箇所に xps_proj_i を追記してください。




```
NET "SYS_CLK_IN" TNM_NET = "SYS_CLK_IN";
TIMESPEC "TS_SYS_CLK_IN" = PERIOD "SYS_CLK_IN" 10000 ps;

NET "xps_proj_i/ppc_reset_bus_Chip_Reset_Req" TPTHRU = "RST_GRP";
NET "xps_proj_i/ppc_reset_bus_Core_Reset_Req" TPTHRU = "RST_GRP";
NET "xps_proj_i/ppc_reset_bus_System_Reset_Req" TPTHRU = "RST_GRP";

TIMESPEC "TS_RST1" = FROM "CPUS" THRU "RST_GRP" TO "FFS" TIG;

NET "xps_proj_i/sys_bus_reset" TIG;

INST "xps_proj_i/ocm_temac_cntlr/ocm_temac_cntlr/v4_emac_top/v4_emac" LOC =
"EMAC_X0Y0";
INST "xps_proj_i/ppc405_system/ppc405_system/PPC405_ADV_i/PPC405_ADV_i" LOC =
"PPC405_ADV_X0Y0";

# SYSTEM I/O
--後略
```

図 14.7 UCF ファイル修正(SZ410)

14.1.3.3. プログラムファイル作成とコンフィギュレーション

[Generate Programming File]をダブルクリックしてください。ソフトウェアを含まない bit ファイルが生成されます。[Update Bitstream with Processor Data]をダブルクリックしてください。ハードウェアでつくった bit ファイルの中にアプリケーションを書き込みます。top_download.bit が出来上がります。

Timing error が出る場合は、[Implement Design]を右クリックしてメニューを出し、[Process Properties...]を選択し、オプションを変更したり、UCF に制約を追加したりしてください。XPS のデザ

インで使用していたオプションは"C:\suzaku\suzaku-ise\sz***-yyyymmdd\etc\fast_runtime.opt"で確認できます。

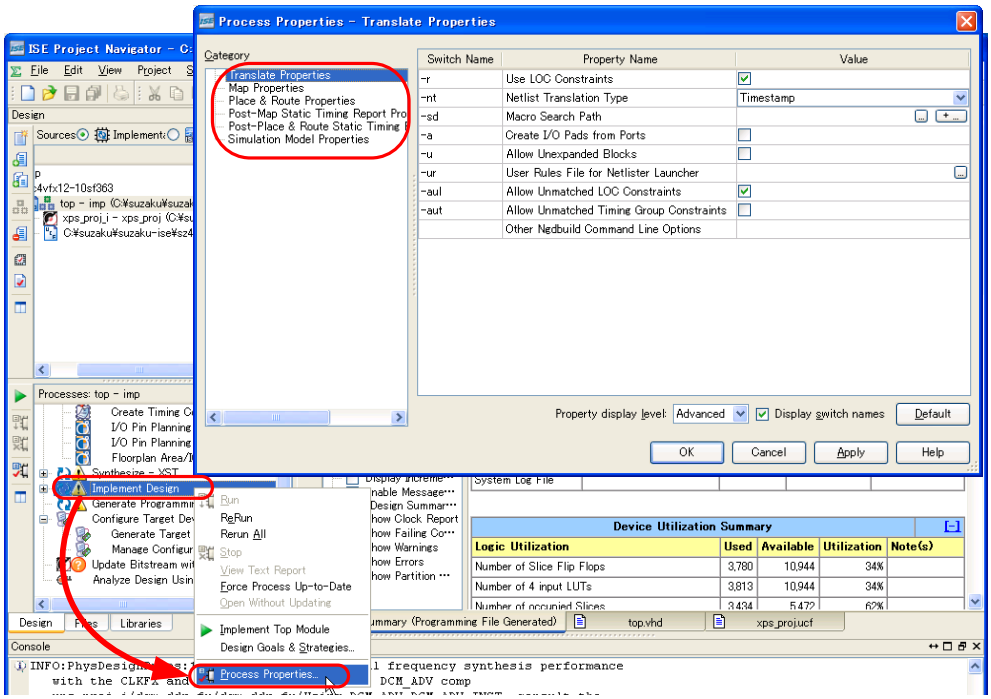


図 14.8 Translate や Map のオプション変更

iMPACT を立ち上げ、top_download.bit を書き込んでください。SUZAKU のデフォルトが書き込まれます。

14.1.3.4. XPS で変更を加えた時は？

XPS で External の信号を追加削除した場合は、ネットリストを作成し直し、新たに生成された xps_proj_stub.vhd を参考に、top.vhd の xps_proj の component 宣言および、そのインスタンスに変更を加えてください。ISE はコンパイル前に XPS デザインに変更がないか確認をします。XPS で mhs ファイルや mss ファイル、ソフトウェアを変更した場合、自動的にバックエンドで XPS を動作させコンパイルを実行してくれます。

14.2. 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。

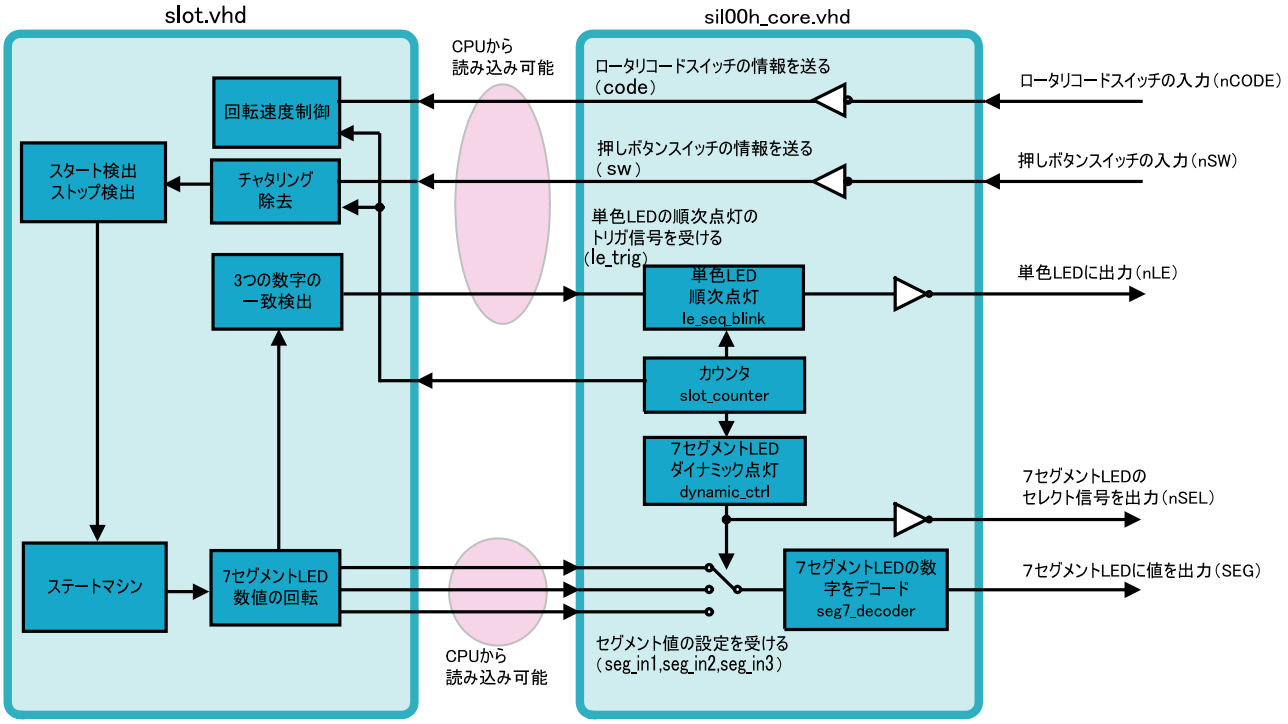


図 14.9 Create and Import Peripheral Wizard の起動のさせ方

"C\suzaku\sz***- yyyymmdd"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- yyyymmdd"として作業を進めます。付属 CD-ROM の"suzaku-starter-kit\fpga\xps_sil00h_vx_xx_x.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 xps_sil00h_vx_xx_x を"C\suzaku\sz***-h- yyyymmdd \pcores"以下にコピーしてください。

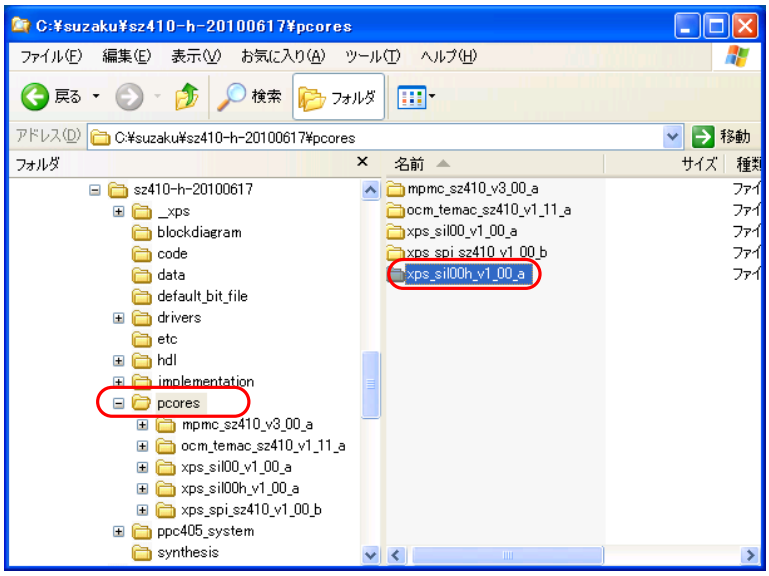


図 14.10 IP コア(ハード版)追加

XPS を起動し、"C:\suzaku\szz***-h- yyyymmdd\xps_proj.xmp を開いてください。[IP Catalog]タブをクリックし、Project Local pcores に xps_sil00h があるのを確認してください。

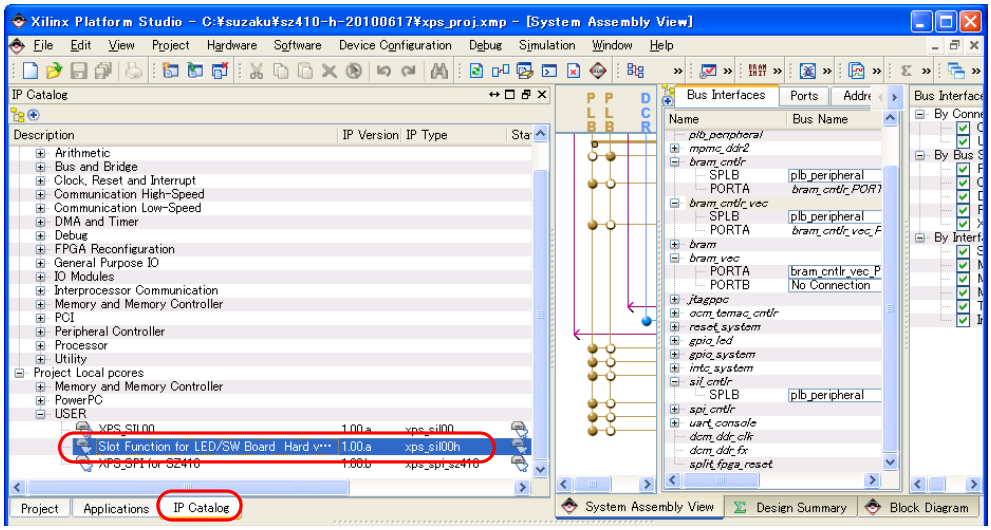


図 14.11 IP コア(ハード版)追加確認

自作 IP コア(ソフト版)xps_sil00 を IP コア(ハード版)xps_sil00h に置き換えます。xps_sil00h は割り込みを使用していないので、割り込みに関する記述を削除します。

まず、ハードウェアの変更をします。

[Project]タブをクリックし、[MSS File: xps_proj.mss]をダブルクリックして開いてください。xps_sil00 のドライバの記述をしているところを探して削除し、保存してください。xps_sil00h ではドライバを使用しません。

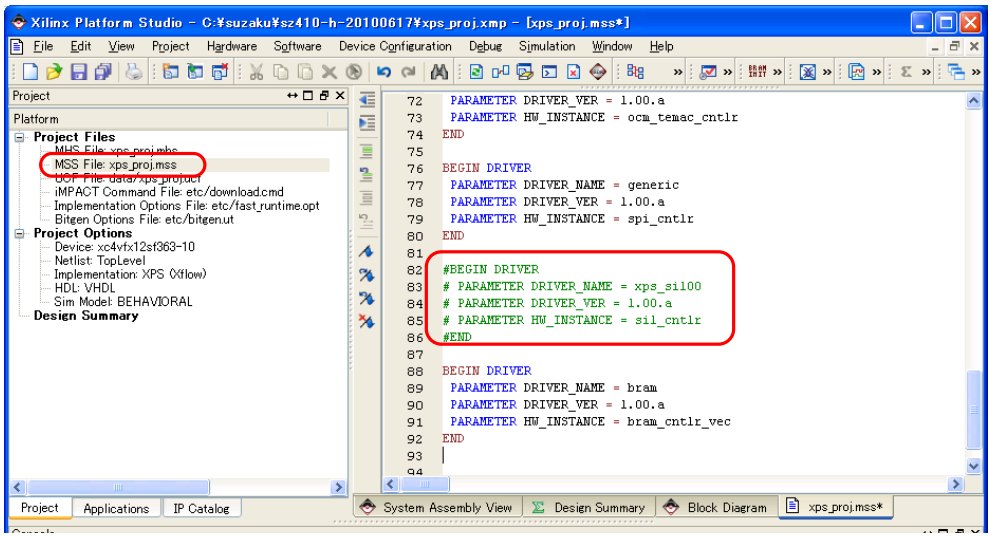
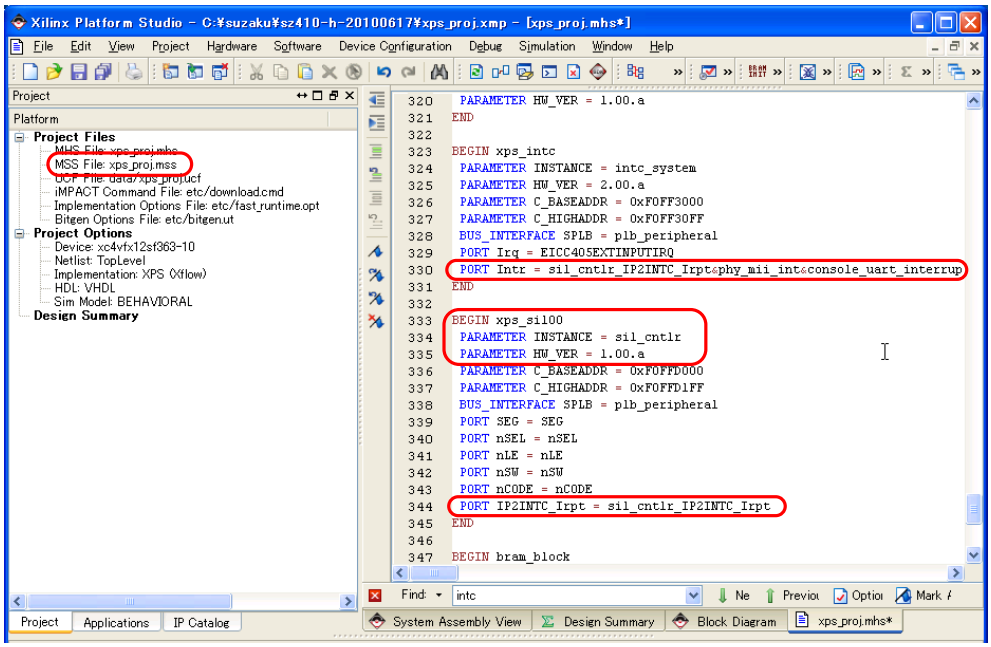


図 14.12 MSS File 変更

[Project]タブをクリックし、[MHS File: xps_proj.mhs]をダブルクリックして開いてください。`xps_sil00` を記述しているところを探してこれを `xps_sil00h` の記述に変更します。その際、`HW_VER` があっているかも確認してください。`xps_intc` を記述しているところを探して、割り込みの記述を削除します。変更が終わったら保存してください。




```
BEGIN xps_intc
 PARAMETER INSTANCE = intc_system
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_BASEADDR = 0xF0FF3000
 PARAMETER C_HIGHADDR = 0xF0FF30FF
 BUS_INTERFACE SPLB = plb_peripheral
 PORT Irq = EICC405EXTINPUTIRQ
 PORT Intr = phy_mii_int&console_uart_interrupt&fifo_int
END

BEGIN xps_sil00h
 PARAMETER INSTANCE = sil_cntlr
PARAMETER HW_VER = 1.00.a
 PARAMETER C_BASEADDR = 0xF0FFD000
 PARAMETER C_HIGHADDR = 0xF0FFD1FF
 BUS_INTERFACE SPLB = plb_peripheral
 PORT SEG = SEG
 PORT nSEL = nSEL
 PORT nLE = nLE
 PORT nSW = nSW
 PORT nCODE = nCODE
#削除
END
```

図 14.13 MHS File 変更

以上で IP コアが置き換わりました。IP Type が xps_sil00h に変更されます。

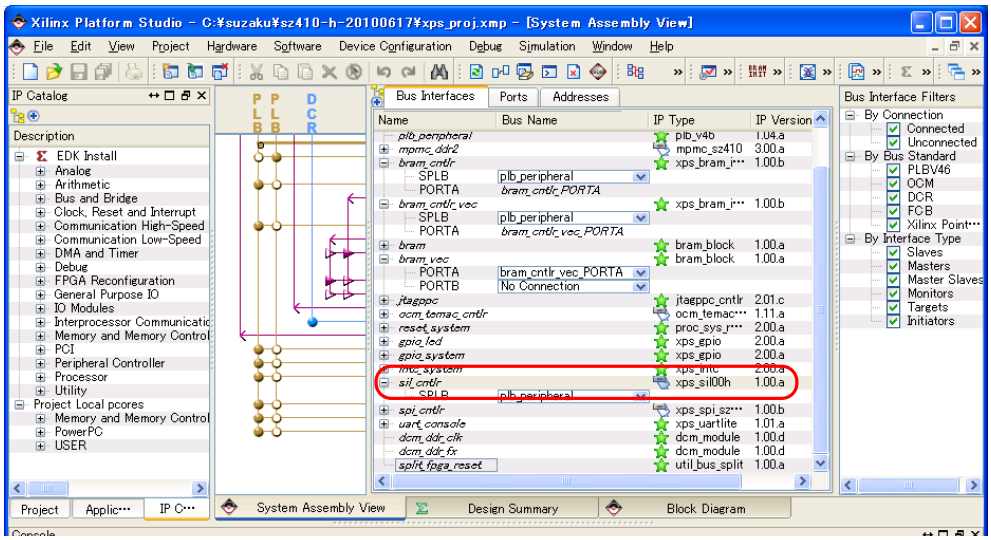


図 14.14 IP コア(ハード版)に置き換え

次にソフトウェアから割り込みの設定、記述を削除していきます。[Applications]タブをクリックし、Sources から `interrupt.c`、`slot.c`、Headers から `interrupt.h`、`slot.h` を削除してください。これらのファイルは使用しません。

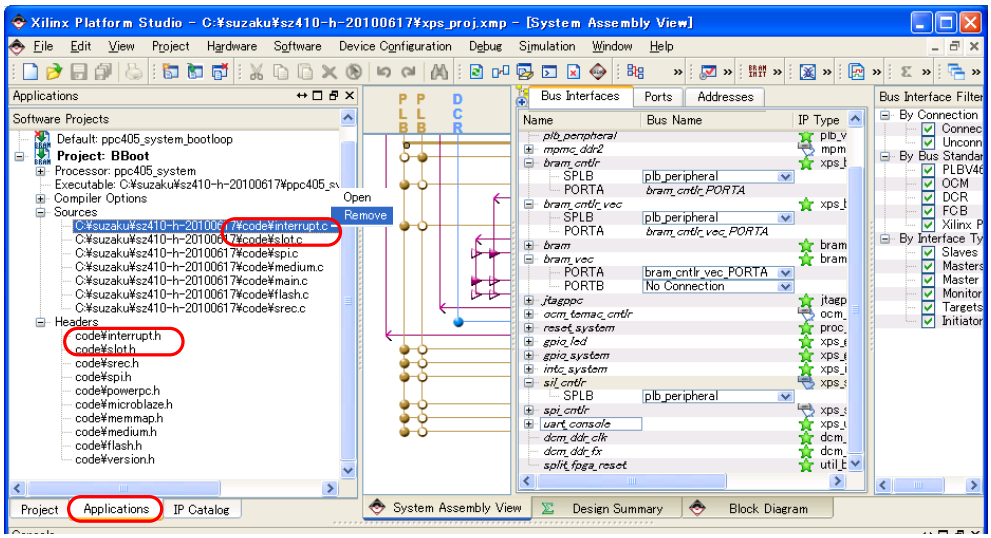


図 14.15 不要なファイルの削除

これで変更は終了です。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。

14.3. 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***-yyyymmdd \implementation"の中にある download.bit をつかってフラッシュメモリを書き換えてください。

フラッシュメモリの中に入っている Linux では最初から CGI が動作しています。(フラッシュメモリの中の Linux を書き換えてしまっている場合は、フラッシュメモリの 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。)

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。

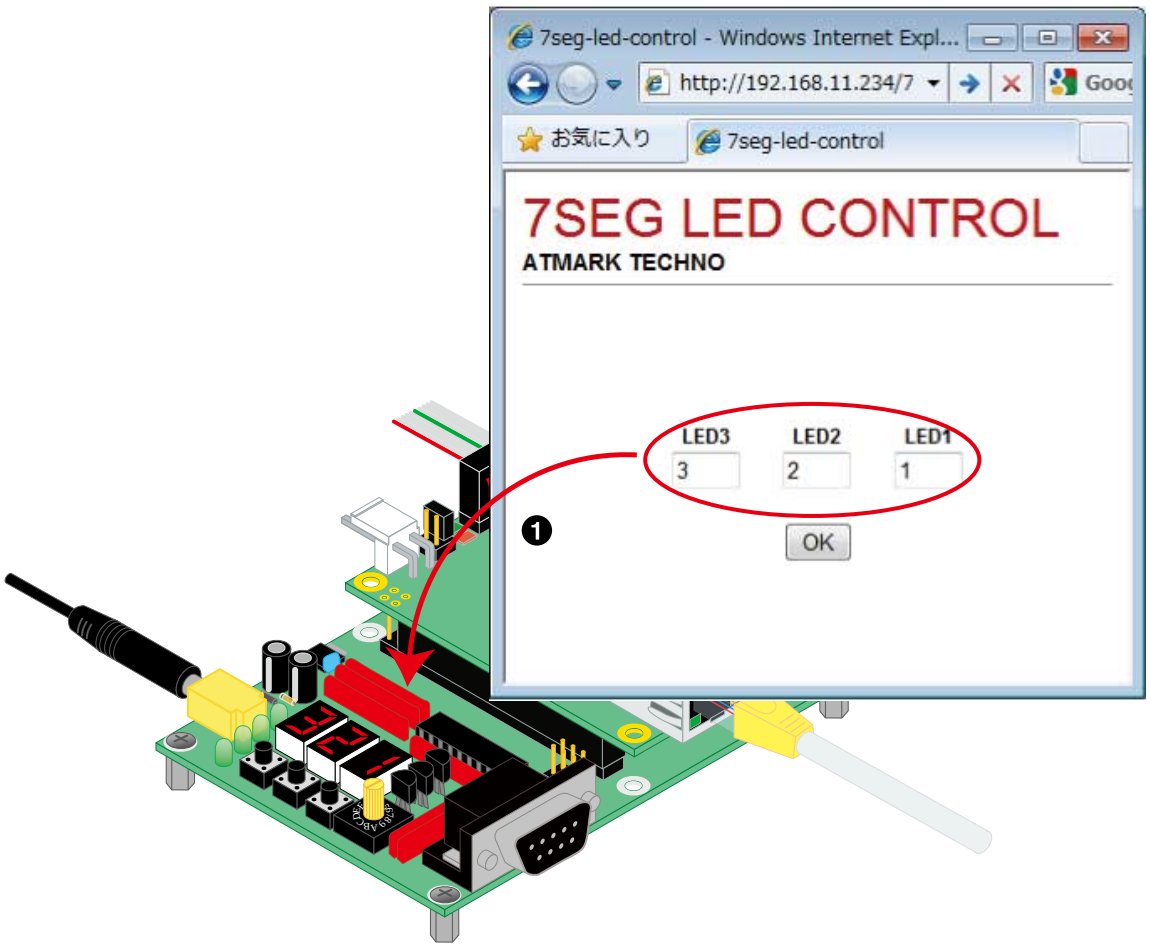


図 14.16 自作のコアをコントロール

❶ 1 ～ F(16 進数)の数字を設定して[OK]をクリックすると、7 セグメント LED に設定した数字が表示される。

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。以下のソースコードだけでは CGI を作成することはできませんので、ご注意ください。

CGI の作成方法やコンパイル方法、フラッシュメモリに書き込むためのデータの作成方法等ついては、「SUZAKU スターターキットガイド(Linux 開発編)」、「SUZAKU ソフトウェアマニュアル」、「uClinux-dist 開発者ガイド」等のソフトウェアのマニュアルをご参照ください。

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

例 14.1 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)
{

                    \"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd\">
\n");
  printf("<html xmlns=\"http://www.w3.org/1999/xhtml\" xml: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);
}
```

14.4. SDK を使ってデバッグ

Eclipse ベースの SDK(Software Develpment Kit)でデバッグします。実は現在こちらの方法が主流です。「13.4. ソフトウェアのデバッグ」と基本的に出来ることは同じです。

「12. SUZAKU のカスタマイズ」の作業まで行ったプロジェクトを開き、[Project]→[Export Hardware Design to SDK]をクリックして下さい。XML フォーマットのハードウェア記述ファイルを含むハードウェアプラットフォーム情報が SDK にエクポートされます。

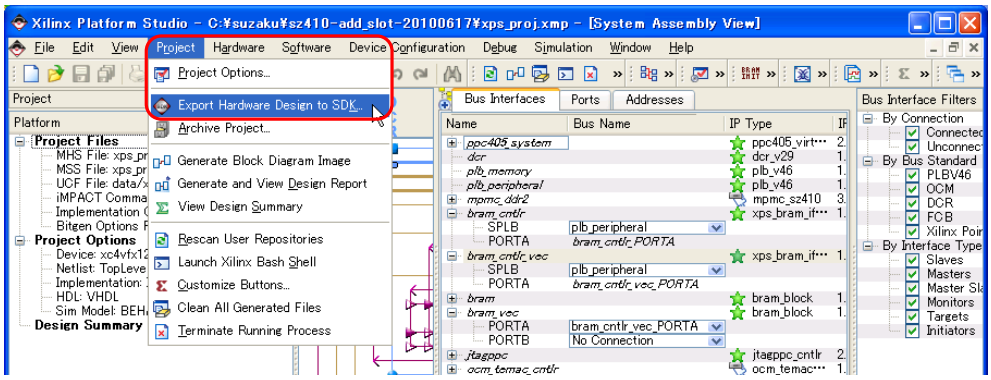


図 14.17 SDK にエクスポート

[Export & Launch]をクリックして下さい。ハードウェアプラットフォーム情報が生成され、SDK が起動します。

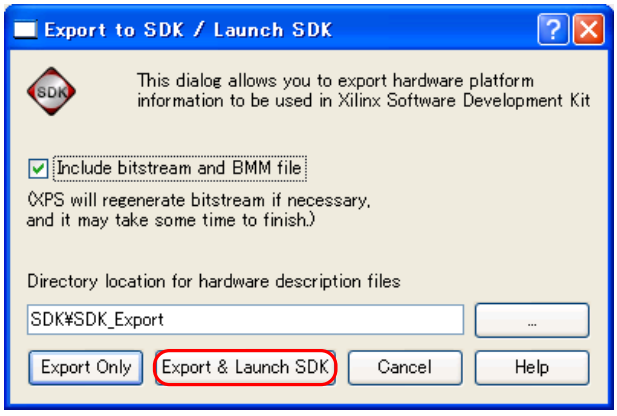


図 14.18 ハードウェアプラットフォーム情報の生成

起動後なにをすればいいのか表示されるので、確認をし[OK]をクリックしてください。

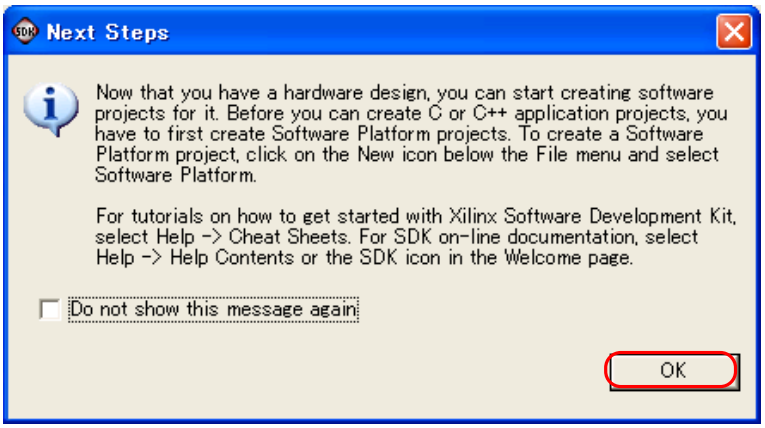


図 14.19 SDK 起動後にやる事

[Tools]→[Software_Repositories]をクリックし、自作 IP コアのドライバが読みこまれるように設定します。自作 IP コアのドライバ(xps_sil00_v1_00a)の入っている drivers フォルダの 2 つ上の階層をユーザーレポジトリに設定してください。ここでは[C:\suzaku]に設定します。

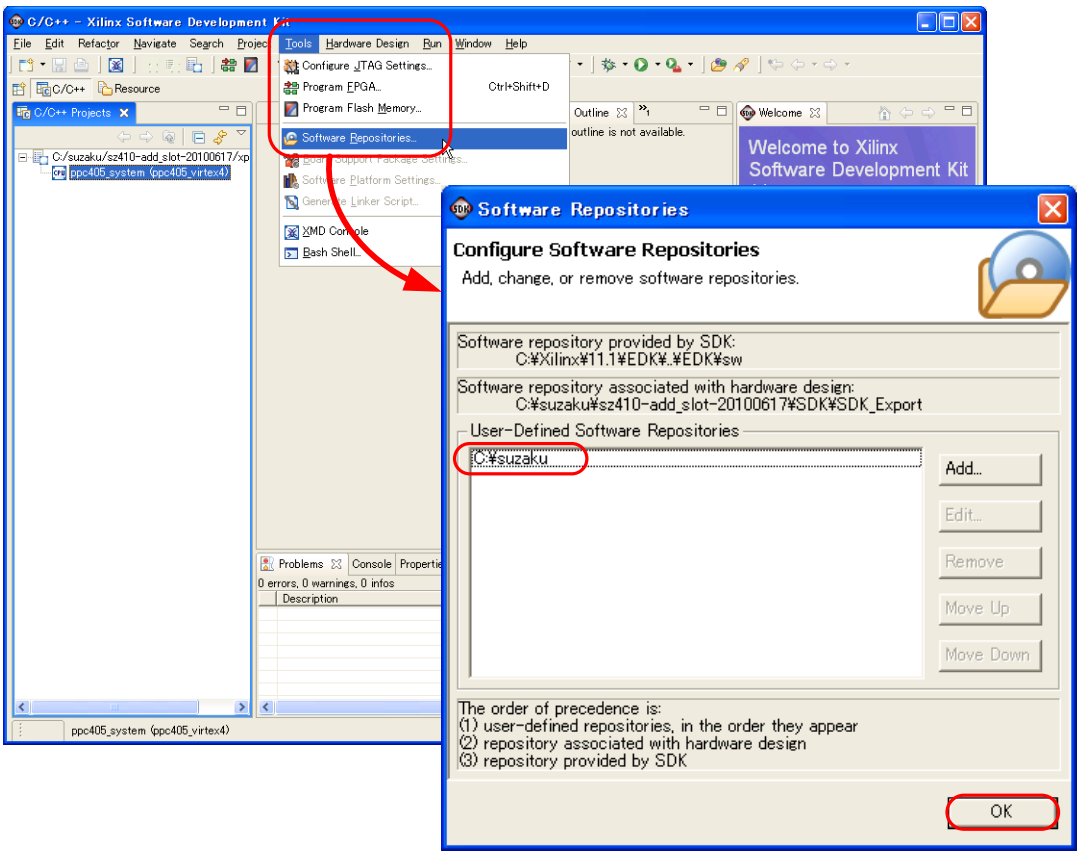


図 14.20 新しいプロジェクト作成

プロジェクトを作成します。[File]→[New]→[Project...]をクリックしてください。

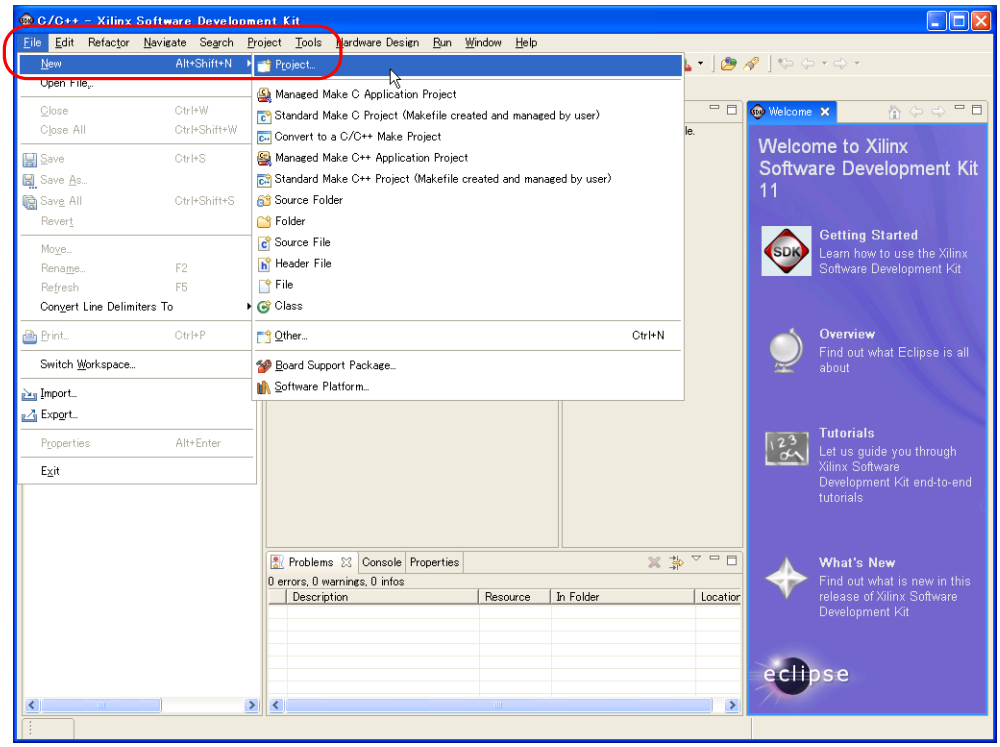


図 14.21 新しいプロジェクト作成

プロジェクトの種類を聞かれるので[Software Platform]を選択し[Next]をクリックしてください。

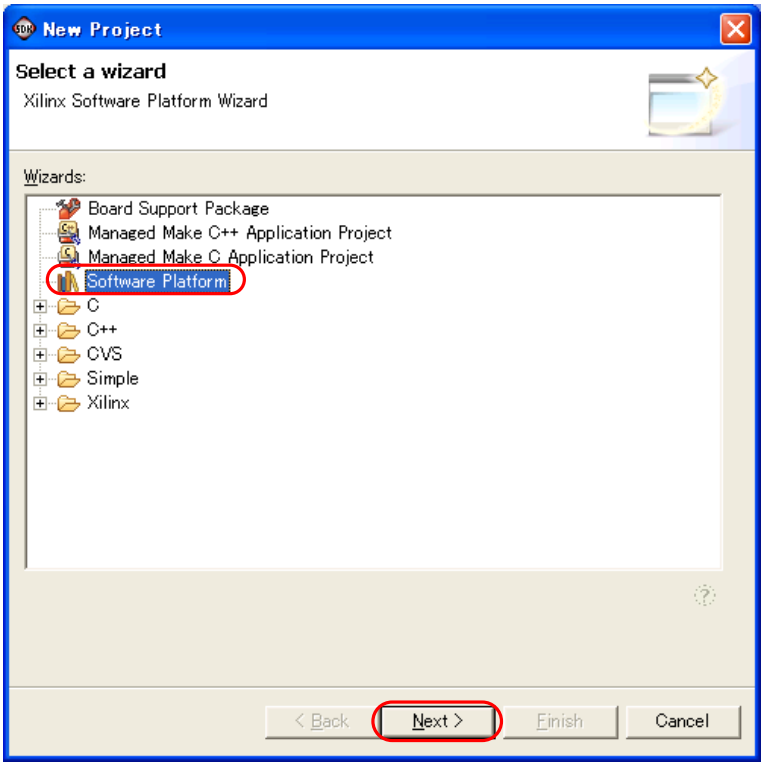


図 14.22 プロジェクトの種類選択

プロジェクト名を入力してください。ここでは[suzaku]としています。入力できたら[Finish]をクリックしてください。

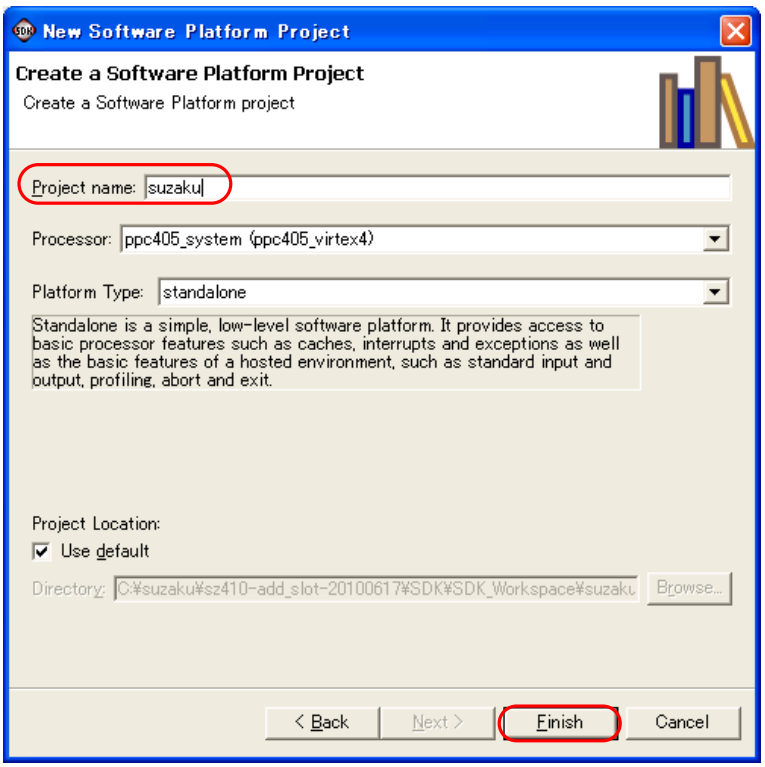


図 14.23 プロジェクト名入力

先ほどエクスポートしたハードウェアプラットフォーム情報により、ドライバなどが読みこまれます。

"C:\suzaku\sz410-add_slot-20100617\drivers\xps_sil00_v1_00_a" を "C:\suzaku\sz410-add_slot-20100617\SDK\SDK_Workspace\suzaku\ppc405_system\libsrc"以下にコピーしてください。

次は C のアプリケーションプロジェクトを作成します。[File]→[New]→[Managed Make C Application Project]をクリックしてください。

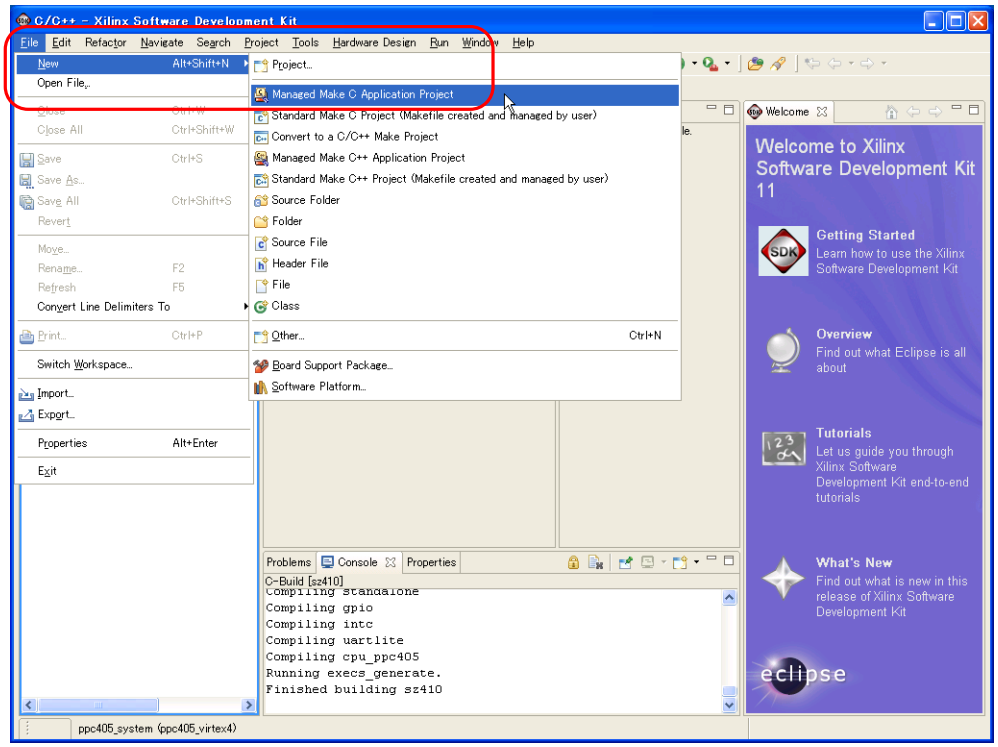


図 14.24 自作 IP コアのドライバの追加

アプリケーションプロジェクトの設定をします。Project Name:にプロジェクト名を入力してください。BBoot をインポートするので、ここでは[BBoot]とします。特にサンプルアプリケーションはいらないので、[Empty Application]をクリックし、[Finish]を選択してください。

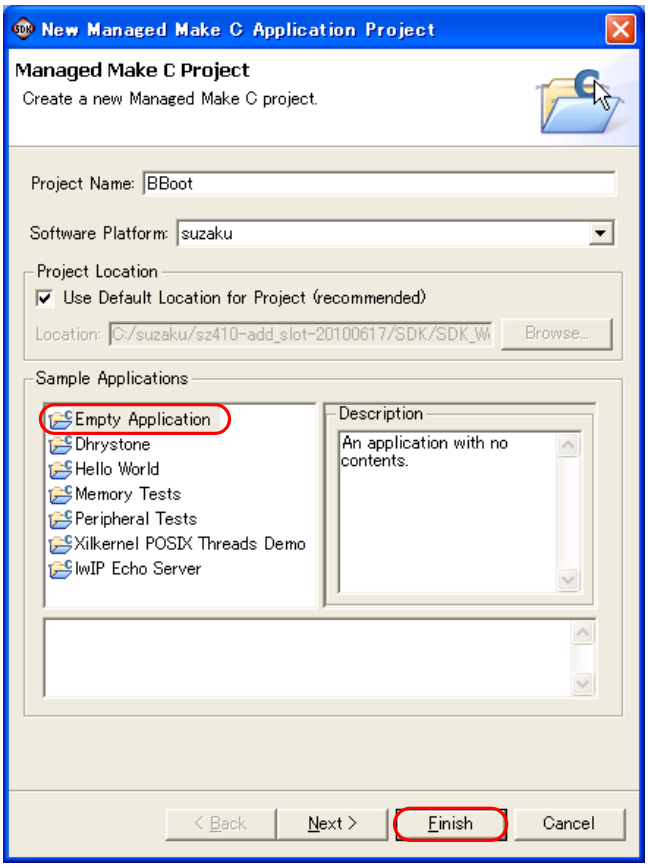


図 14.25 アプリケーションプロジェクトの設定

BBoot のソースコードをインポートします。[BBoot]の上で右クリックしてメニューを出し、[Import]を選択してください。

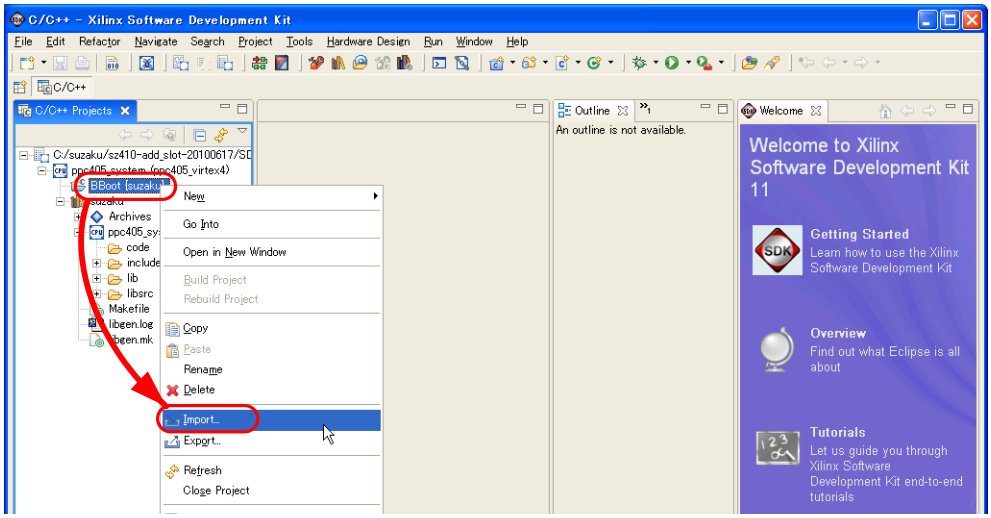


図 14.26 アプリケーションプロジェクトの設定

インポートの形式を聞かれるので[File system]を選択し、[Next]をクリックしてください。

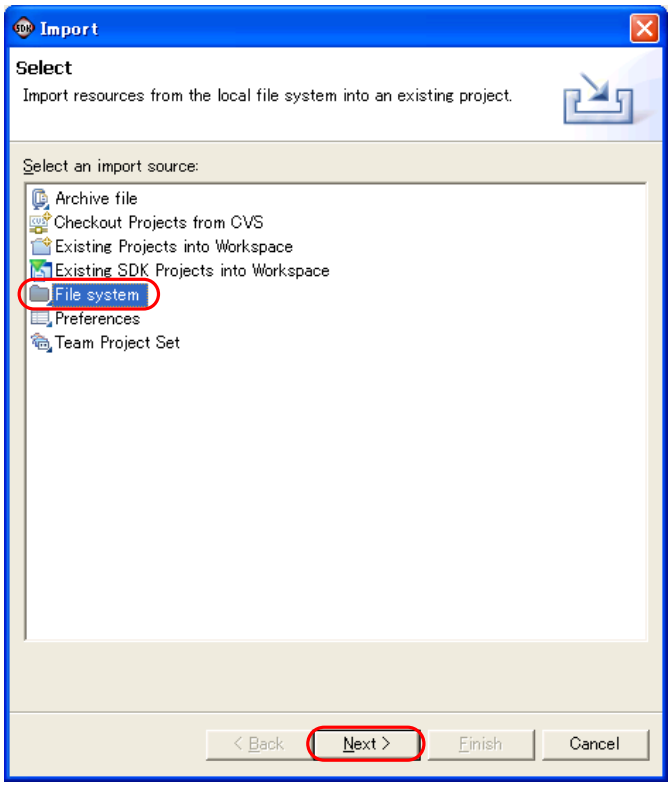


図 14.27 File system をインポート

[From directory:]を"C:\suzaku\sz410-add_slot-20100617\code"とし、code をチェックし、[Finish]をクリックしてください。(code 以下には必要のないファイルも含まれているので、面倒でなければ除外してください。)

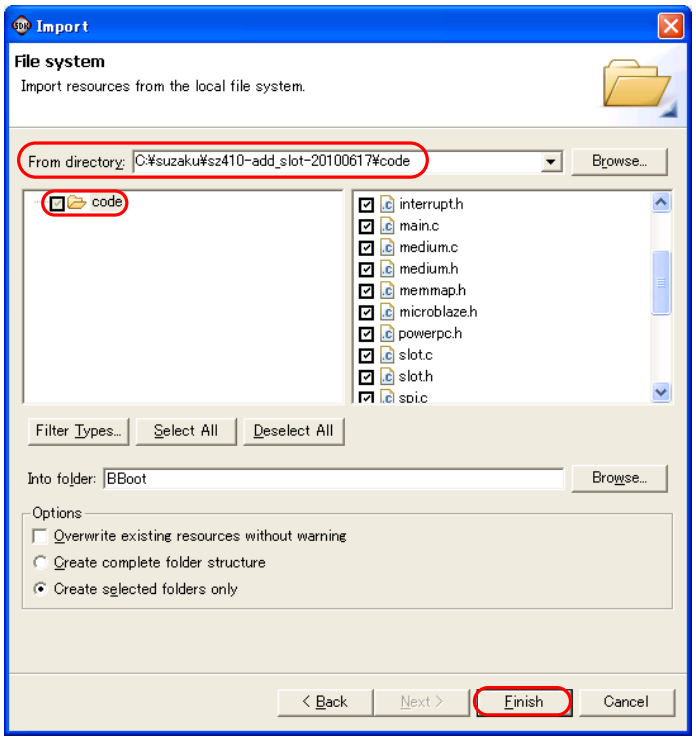


図 14.28 BBoot をインポート

自動で Build が行われますが、リンカースクリプトが設定されていないため、エラーとなります。[BBoot]の上で右クリックしてメニューを出し[Properties]を選択してください。

[C/C++ Build]をクリックし、[Tool Settings]タブをクリックし、Linker Script を選択してください。[+]ボタンをクリックし、Linker Script を設定してください。リンカースクリプトは["C:\suzaku\sz410-add_slot-20100617\SDK\SDK_Workspace\BBoot\BBoot_linker_script.ld"]を選択します。

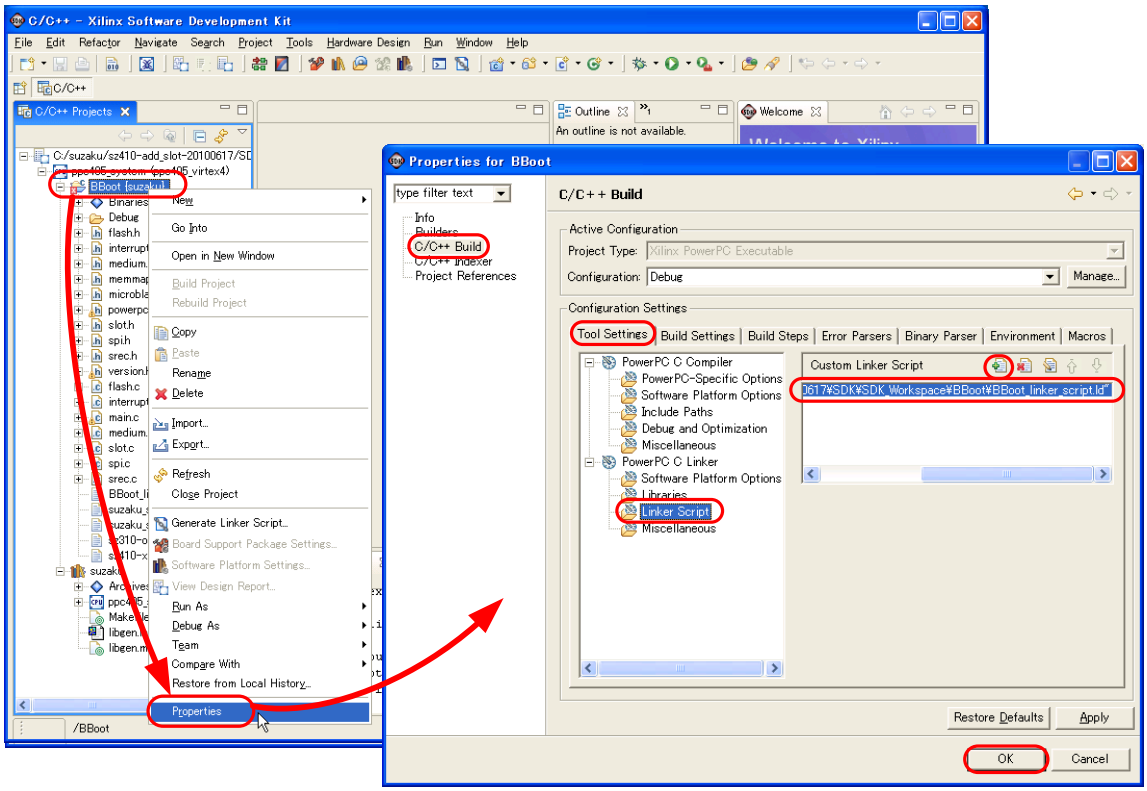


図 14.29 リンカースクリプト設定

自動的に Build されます。自動的に Build されない場合は[Project]→[Build All]をクリックしてください。

[Tools]→[Program FPGA...]をクリックし、[Initialization ELF]に先ほど作成した `BBoot.elf` ファイルを指定し、[Save and Program]をクリックして下さい。FPGA にデバッグ機能付きのスロットマシンのコンフィギュレーションデータがダウンロードされます。

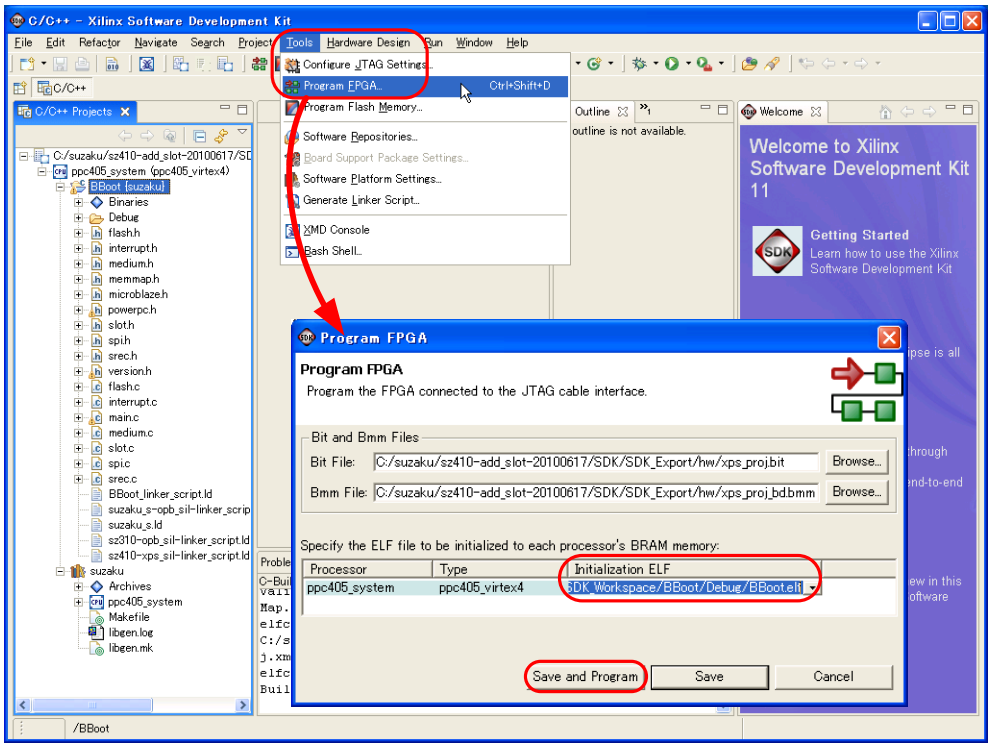


図 14.30 コンフィギュレーションデータダウンロード

[BBoot]の上でクリックをしてメニューを出し[Debug As]→[Debug on Hardware]を選択してください。デバッガが起動します。

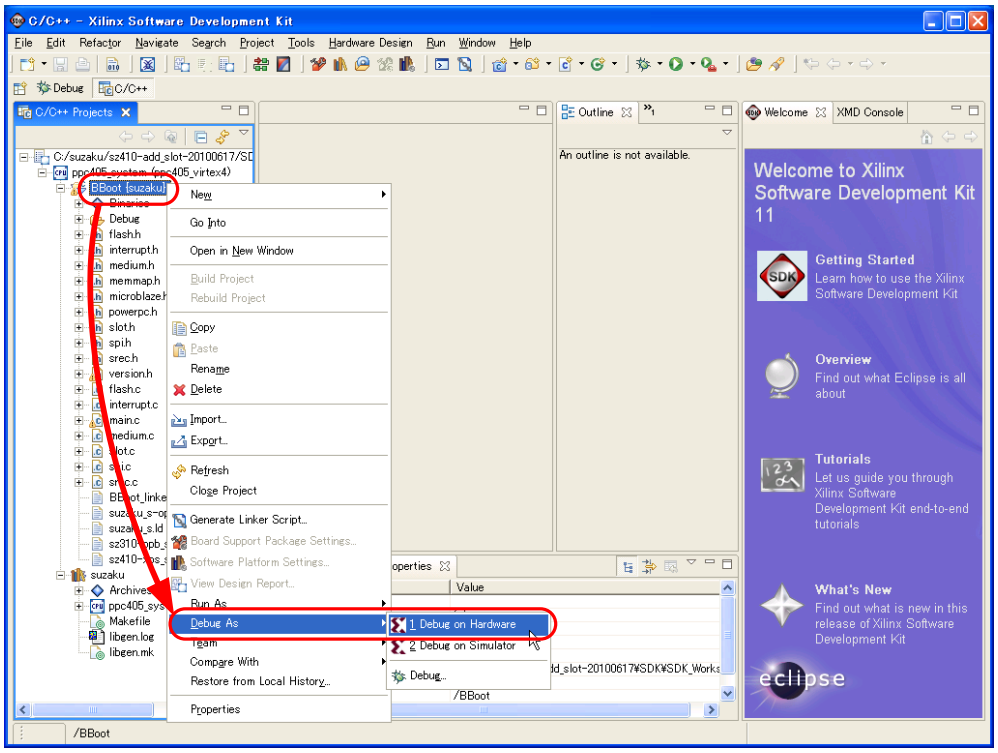


図 14.31 デバッガ起動

BreakPoint を設定します。今回は BreakPoint を timer_interrupt_handler に設定します。`interrupt.c` を開き、`int timer_interrupt_handler` と書いてある行を探し、行の数字の横でダブルクリックして下さい。BreakPoint が設定されます。

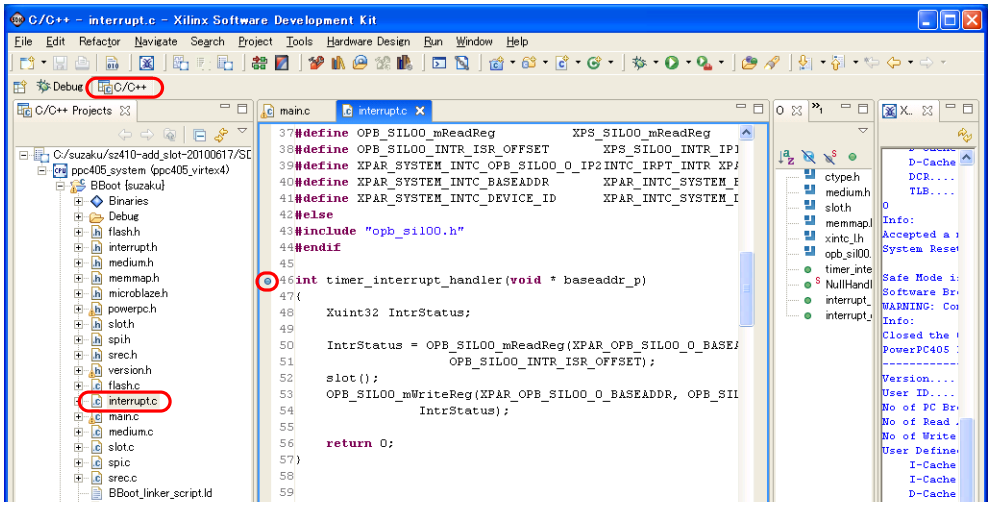


図 14.32 BreakPoint 設定

Resume をクリックして下さい。先ほど設定した BreakPoint で Break します。Step Into 等をクリックしてください。Instruction Stepping Mode をクリックすると、インストラクション単位でステップ実行できるようになります。

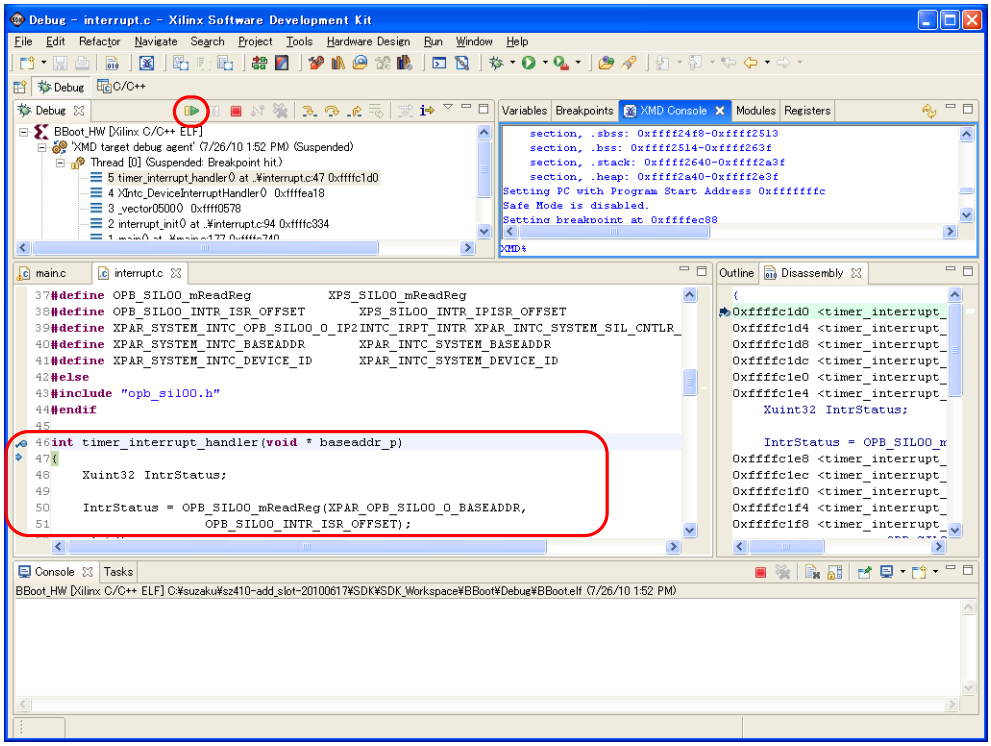


図 14.33 timer_interrupt_handler で Break

ローカル変数やスタックの一覧を確認してみてください。

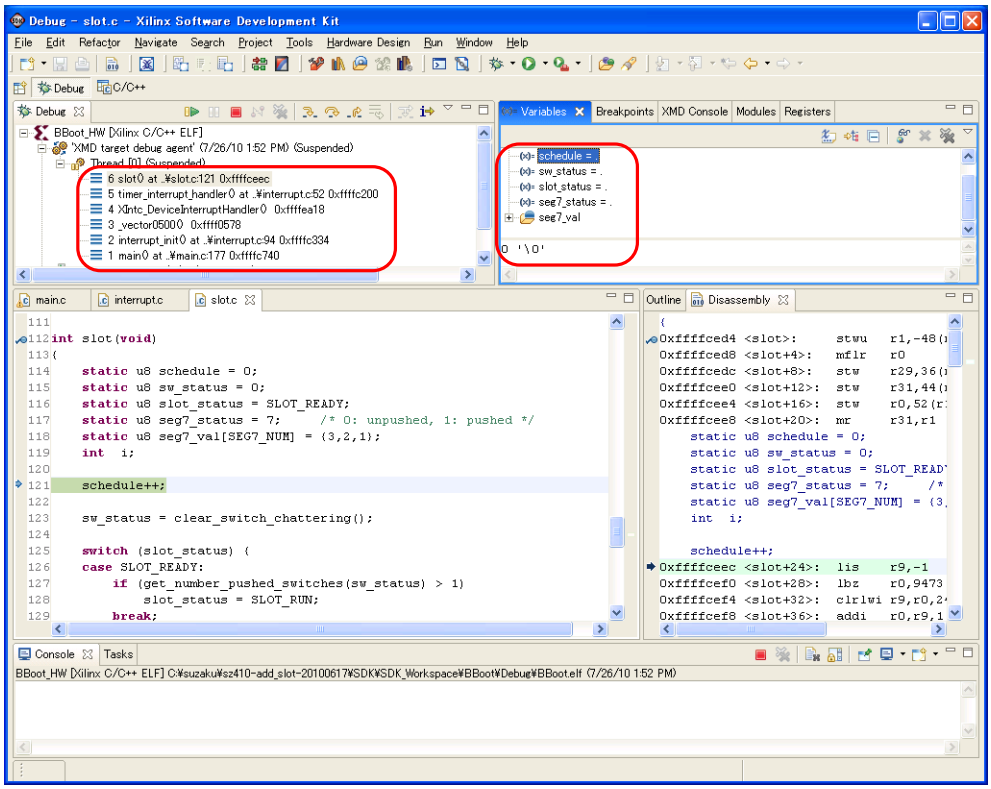


図 14.34 スタック一覧やローカル変数を確認

14.5. スターターキットガイド(Linux 開発編)への布石

本書と対になる Linux 開発編なので、本書で作った IP コアが題材であればよかったのですが、色々な都合上違います。

ここでは Linux 開発編で使用する FPGA ファイルを作成します。

ハードウェアは既に決まっていて、ソフトウェアを作成するのがよくあるパターンですが、ハードウェアも自由に変更できるので、ソフトウェアからハードウェアに要求を出すことができます。両方から要求をだしあい変更できるところが、SUZAKU のおもしろいところでもあり、難しいところでもあります。

14.5.1. 要求仕様

以下に要求仕様を記載します。Linux 開発編をみてみるのも良いかもしれません。

14.5.1.1. 単色 LED

- 4 ビットのレジスタから単色 LED にそれぞれアクセスできるようにする。レジスタは Read/Write 可能とする。
- 1 で点灯、0 で消灯とする。
- Base Address は SZ130 の場合、0xFFFFD200、SZ410 の場合、0xF0FFD200 とする。

表 14.1 単色 LED のレジスタ仕様

レジスタ	接続先
0(LSB)	D1
1	D2
2	D3
3	D4

14.5.1.2. 押しボタンスイッチ

- 押しボタンスイッチの状態を 3 ビットのレジスタから取得できるようにする。割りこみを検知できるようにする(割り込み番号 5)。
- 1 で押されている状態、0 で押されていない状態とする。
- Base Address は SZ130 の場合、0xFFFFD400、SZ410 の場合、0xF0FFD400 とする。

表 14.2 押しボタンスイッチのレジスタ仕様

レジスタ	接続先
0(LSB)	SW1
1	SW2
2	SW3

14.5.1.3. ロータリコードスイッチ

- ロータリコードスイッチの値 4 ビットのレジスタから取得できるようにする。割り込みを検知できるようにする(割り込み番号 4)。

・ Base Address は SZ130 の場合、0xFFFFD600、SZ410 の場合、0xF0FFD600 とする。

表 14.3 ロータリコードスイッチのレジスタ仕様

レジスタ	接続先
0(LSB)	CODE1
1	CODE2
2	CODE3
3	CODE4

14.5.1.4. 7 セグメント LED

・ ダイナミック点灯で 7 セグメント LED3 つを表示できるようにする。24bit レジスタを用意し、LSB 側から 8bit 境界で、LED1、LED2、LED3 をマップし、各 LED は、下位 bit 側から SEG0 ～ SEG7 をマップする。レジスタは Read/Write 可能にする。

・ bit は 1 で点灯、0 で消灯する。

・ Base Address は SZ130 の場合、0xFFFFD000、SZ410 の場合、0xF0FFD000

表 14.4 7 セグメント LED のレジスタ仕様

レジスタ	接続先
0(LSB)	LED1(SEG0)
1	LED1(SEG1)
2	LED1(SEG2)
3	LED1(SEG3)
4	LED1(SEG4)
5	LED1(SEG5)
6	LED1(SEG6)
7	LED1(SEG7)
8	LED2(SEG0)
9	LED2(SEG1)
10	LED2(SEG2)
11	LED2(SEG3)
12	LED2(SEG4)
13	LED2(SEG5)
14	LED2(SEG6)
15	LED2(SEG7)
16	LED3(SEG0)
17	LED3(SEG1)
18	LED3(SEG2)
19	LED3(SEG3)
20	LED3(SEG4)
21	LED3(SEG5)
22	LED3(SEG6)
23	LED3(SEG7)

14.5.1.5. シリアル

・ シリアルを 1 ポート使用できるようにする。割り込みを検知できるようにする(割り込み番号 3)。

表 14.5 シリアルの設定

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

14.5.2. XPS で作業

実際に作業を行ってみて分かったと思いますが、バスに接続する IP コアを作成するのは中々大変です。今回は自作 IP コアを使用せず、すでにある IP コアを使って要求を満たす FPGA ファイルを作成します。

「14.1. XPS デザインを ISE のサブモジュールとして読み込む」で作成したプロジェクトで作業を行ないます。ISE を起動し、[File]→[Open Project...]で `C:\suzaku\suzaku-ise\top.xise"`を開いてください。開いたら、[xps_proj_i-xps_proj(xps_proj.xmp)]をダブルクリックしてください。XPS が起動します。

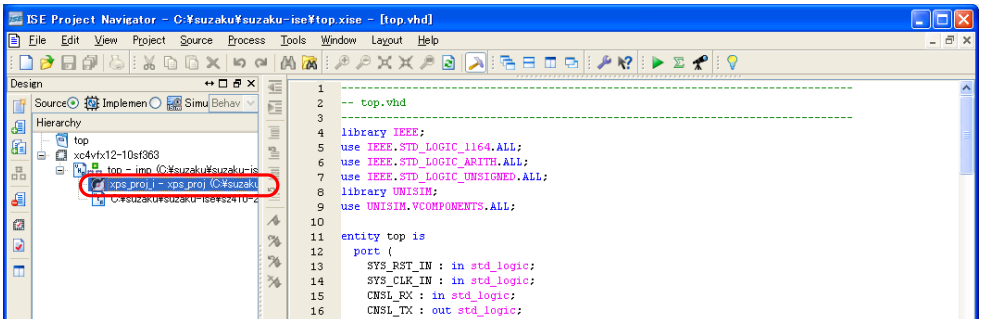


図 14.35 XPS の起動

単色 LED、押しボタンスイッチ、ロータリコードスイッチ、7 セグメント LED のためにそれぞれ XPS GPIO を追加してください。また、シリアルのために XPS UART lite を追加してください。追加したらバスに接続し、設定を変更してください。すべて、外部信号に設定し、単色 LED、7 セグメント LED の値は読み込み可能にするため、内部でも接続します。ISE 側でロジックを書くために、バスクロック(SZ130 では 51.6096MHz、SZ410 では 87.5MHz)を外部信号に設定してください。

例 14.2 mhs ファイルの例(SZ410)

```
PARAMETER VERSION = 2.1.0

PORT SYS_RST_IN = SYS_RST_IN, DIR = I, RST_POLARITY = 1, SIGIS = RST
PORT SYS_CLK_IN = SYS_CLK_IN, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
# 中略
PORT CNSL_RX2 = CNSL_RX2, DIR = I                    # UART RX 信号
PORT CNSL_TX2 = CNSL_TX2, DIR = O                    # UART TX 信号
PORT SEG_IN = SEG_IN_w, DIR = O, VEC = [0:23]        # 7 セグメント LED の信号
PORT LE = LE_w, DIR = O, VEC = [0:3]                 # 単色 LED の信号
```

```
PORT SW = SW, DIR = I, VEC = [0:2]                  # 押しボタンスイッチの信号
PORT CODE = CODE, DIR = I, VEC = [0:3]              # ロータリコードスイッチの信号
PORT SYS_CLK_S_OUT = sys_clk_s, DIR = O             # バスクロック

# 中略

BEGIN xps_intc
 PARAMETER INSTANCE = intc_system
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_BASEADDR = 0xF0FF3000
 PARAMETER C_HIGHADDR = 0xF0FF30FF
 BUS_INTERFACE SPLB = plb_peripheral
 PORT Irq = EICC405EXTINPUTIRQ
 PORT Intr =
sil_rsw_intr&sil_sw_intr&sil_uart_console_intr&phy_mii_int&console_uart_interrupt
&fifo_int
END

BEGIN xps_gpio
 PARAMETER INSTANCE = sil_7seg                      # 7 セグメント LED
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_GPIO_WIDTH = 24                        # 24 ビットに設定
 PARAMETER C_BASEADDR = 0xF0FFD000                  # メモリマップの設定(BASE ADDRESS)
 PARAMETER C_HIGHADDR = 0xF0FFD1FF                  # メモリマップの設定(HIGH ADDRESS)
 BUS_INTERFACE SPLB = plb_peripheral                # バスに接続
 PORT GPIO_IO_I = SEG_IN_w                          # 読み込みできるようにする
 PORT GPIO_IO_O = SEG_IN_w                          # 出力信号
END

BEGIN xps_gpio
 PARAMETER INSTANCE = sil_led                       # 単色 LED
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_GPIO_WIDTH = 4                         # 4 ビットに設定
 PARAMETER C_BASEADDR = 0xF0FFD200                  # メモリマップの設定(BASE ADDRESS)
 PARAMETER C_HIGHADDR = 0xF0FFD3FF                  # メモリマップの設定(HIGH ADDRESS)
 BUS_INTERFACE SPLB = plb_peripheral                # バスに接続
 PORT GPIO_IO_I = LE_w                              # 読み込みできるようにする
 PORT GPIO_IO_O = LE_w                              # 出力信号
END

BEGIN xps_gpio
 PARAMETER INSTANCE = sil_sw                        # 押しボタンスイッチ
 PARAMETER HW_VER = 2.00.a
 PARAMETER C_GPIO_WIDTH = 3                         # 3 ビットに設定
 PARAMETER C_ALL_INPUTS = 1                         # 入力信号のみに設定
 PARAMETER C_BASEADDR = 0xF0FFD400                  # メモリマップの設定(BASE ADDRESS)
 PARAMETER C_HIGHADDR = 0xF0FFD5FF                  # メモリマップの設定(HIGH ADDRESS)
 PARAMETER C_INTERRUPT_PRESENT = 1                  # 割りこみを使用する
 BUS_INTERFACE SPLB = plb_peripheral                # バスに接続
 PORT GPIO_IO_I = SW                                # 入力信号
 PORT IP2INTC_Irpt = sil_sw_intr                    # 割り込み信号
END

BEGIN xps_gpio
 PARAMETER INSTANCE = sil_rsw                       # ロータリコードスイッチ
 PARAMETER HW_VER = 2.00.a
```

```
 PARAMETER C_GPIO_WIDTH = 4                      # 4 ビットに設定
 PARAMETER C_ALL_INPUTS = 1                      # 入力信号のみに設定
 PARAMETER C_BASEADDR = 0xF0FFD600               # メモリマップの設定(BASE ADDRESS)
 PARAMETER C_HIGHADDR = 0xF0FFD7FF               # メモリマップの設定(HIGH ADDRESS)
 PARAMETER C_INTERRUPT_PRESENT = 1               # 割りこみを使用する
 BUS_INTERFACE SPLB = plb_peripheral             # バスに接続
 PORT GPIO_IO_I = CODE                           # 入力信号
 PORT IP2INTC_Irpt = sil_rsw_intr                # 割り込み信号
END

BEGIN xps_uartlite
 PARAMETER INSTANCE = sil_uart_console           # シリアル
 PARAMETER HW_VER = 1.01.a
 PARAMETER C_BAUDRATE = 115200                   # 転送レート 115.2kbps
 PARAMETER C_USE_PARITY = 0                      # パリティなし
 PARAMETER C_BASEADDR = 0xF0FFA600               # メモリマップの設定(BASE ADDRESS)
 PARAMETER C_HIGHADDR = 0xF0FFA6FF               # メモリマップの設定(HIGH ADDRESS)
 PARAMETER C_SPLB_CLK_FREQ_HZ = 87500000         # バスクロック
 BUS_INTERFACE SPLB = plb_peripheral             # バスに接続
 PORT RX = CNSL_RX2                              # RX 信号
 PORT TX = CNSL_TX2                              # TX 信号
 PORT Interrupt = sil_uart_console_intr          # 割り込み信号
END
```

[Hardware]→[Generate Netlist]をクリックしてください。ネットリストが作成されます。

Linux で作業する際、xparameters.h を使います。SZ410 では xparameters.h からハードウェアのパラメータを渡しています。SZ130 では手動で変更しなければいけませんが、一覧になってると少し便利です。追加した IP すべてに generic のドライバを設定します。generic のドライバを設定すると、BASEADDR と HIGHADDR が定義された xparameters.h が出来上がります。

例 14.3 mss ファイル(押しボタンスイッチ)の例

```
BEGIN DRIVER
 PARAMETER DRIVER_NAME = generic
 PARAMETER DRIVER_VER = 1.00.a
 PARAMETER HW_INSTANCE = sil_sw
END
```

[Software]→[Generate Libraries and BSPs]をクリックしてください。xparameters.h が作成されます。

SZ410 の場合 xparametes.h は atmark-dist-yyyymmdd/linux-2.6.x/arch/ppc/platforms/4xx/xparameters 以下においてあるので、Linux 開発編を行う際に確認してみてください。

14.5.3. ISE で作業

14.5.3.1. XPS の変更を反映

"C:\suzaku\suzaku-ise\sz***-yyyymmdd\hdl\xps_proj_stub.vhd"を開いてください。先ほど XPS で追加した信号が増えているので、その記述を top.vhd に反映させます。外部に出力する信号は後で設定します。

例 14.4 top.vhd に信号追加(SZ410)

```
entity top is
  port (
    -- 中略
  );
end top;

architecture imp of top is

  component xps_proj is
    port (
      -- 中略
      CNSL_RX2 : in std_logic;
      CNSL_TX2 : out std_logic;
      SEG_IN : out std_logic_vector(0 to 23);
      LE : out std_logic_vector(0 to 3);
      SW : in std_logic_vector(0 to 2);
      CODE : in std_logic_vector(0 to 3);
      SYS_CLK_S_OUT : out std_logic
    );
  end component;

  signal seg_in : std_logic_vector(0 to 23);
  signal le : std_logic_vector(0 to 3);
  signal sw : std_logic_vector(0 to 2);
  signal code : std_logic_vector(0 to 3);
  signal sys_clk_s_out: std_logic;

  attribute BOX_TYPE : STRING;
  attribute BOX_TYPE of xps_proj : component is "user_black_box";

begin

  xps_proj_i : xps_proj
    port map (
      -- 中略
      CNSL_RX2 => CNSL_RX2,
      CNSL_TX2 => CNSL_TX2,
      SEG_IN => seg_in,
      LE => le,
      SW => sw,
      CODE => code,
      SYS_CLK_S_OUT => sys_clk_s_out
    );

end architecture imp;
```

14.5.3.2. 回路を追加

ダイナミック点灯をするために dynamic_ctrl と slot_counter を使用します。dynamic_ctrl は少しインターフェースが違うので、変更する必要があります。seg_in1、seg_in2、seg_in3、seg_data、seg_data_w を std_logic_vector(0 to 7)に変更してください。

例 14.5 dynamic_ctrl.vhd の変更

```
entity dynamic_ctrl is

  Port (
      -- 中略
      seg_in1 : in  STD_LOGIC_VECTOR(0 to 7);
      seg_in2 : in  STD_LOGIC_VECTOR(0 to 7);
      seg_in3 : in STD_LOGIC_VECTOR(0 to 7);
      seg_data : out STD_LOGIC_VECTOR(0 to 7)
  );
end dynamic_ctrl;

architecture imp of dynamic_ctrl is

  signal sel : STD_LOGIC_VECTOR(0 to 2);
  signal seg7_tim_reg : STD_LOGIC;
  signal seg_data_w : STD_LOGIC_VECTOR(0 to 7);

  -- 後略
```

dynamic_ctrl.vhd と slot_counter.vhd をプロジェクトに追加し、top.vhd を上位階層として dynamic_ctrl、slot_counter 回路を呼び出すコードを追記します。単色 LED、押しボタンスイッチ、ロータリコードスイッチは論理を反転させて出力します。

例 14.6 dynamic_ctrl、slot_counter 回路を呼び出す

```
entity top is
  generic (
      C_CNT_WIDTH : integer := 15
  );
  port (
    -- 中略
    CNSL_RX2 : in std_logic;
    CNSL_TX2 : out std_logic;
    nSEL : out std_logic_vector(0 to 2);
    SEG : out std_logic_vector(0 to 7);
    nLE : out std_logic_vector(0 to 3);
    nSW : in std_logic_vector(0 to 2);
    nCODE : in std_logic_vector(0 to 3)
  );
end top;

-- 中略

architecture imp of top is

  component xps_proj is
    port (
      -- 中略
    );
  end component;

  component slot_counter
    generic (
```

```
      C_CNT_WIDTH             : integer := C_CNT_WIDTH
    );
    port (
      SYS_CLK : in std_logic;
      SYS_RST : in std_logic;
      count : out std_logic_vector(0 to C_CNT_WIDTH-1)
    );
  end component;

  component dynamic_ctrl
    port (
      SYS_CLK : in  std_logic;
      SYS_RST : in  std_logic;
      nSEL : out  std_logic_vector(0 to 2);
      seg7_timing : in std_logic;
      seg_in1 : in  std_logic_vector(0 to 7);
      seg_in2 : in  std_logic_vector(0 to 7);
      seg_in3 : in std_logic_vector(0 to 7);
      seg_data : out std_logic_vector(0 to 7)
    );
  end component;

  signal seg_in : std_logic_vector(0 to 23);
  signal le : std_logic_vector(0 to 3);
  signal sw : std_logic_vector(0 to 2);
  signal code : std_logic_vector(0 to 3);
  signal count : std_logic_vector(0 to C_CNT_WIDTH-1);
  signal sys_clk_s_out: std_logic;

  attribute BOX_TYPE : STRING;
  attribute BOX_TYPE of xps_proj : component is "user_black_box";

begin

  -- 中略

  nLE <= not le;
  sw <= not nSW;
  code <= not nCODE;

  slot_counter_0 : slot_counter
    port map (
      SYS_CLK => sys_clk_s_out,
      SYS_RST => SYS_RST_IN,
      count => count
    );

  dynamic_ctrl_0 : dynamic_ctrl
    port map (
      SYS_CLK => sys_clk_s_out,
      SYS_RST => SYS_RST_IN,
      nSEL => nSEL,
      seg7_timing => count(0),
      seg_in1 => seg_in(16 to 23),
      seg_in2 => seg_in(8 to 15),
      seg_in3 => seg_in(0 to 7),
      seg_data => SEG
    );
```

```
end architecture imp;
```

14.5.3.3. ピンアサインの変更

UCF にも信号を追加します。`xps_proj.ucf` をクリックし、Edit Constraints(Text)をダブルクリックして開き、「表 13.2. 自作 IP コア ピンアサイン」を参考に追記してください。

14.5.3.4. プログラムファイル作成

[Generate Programming File]をダブルクリックしてください。ソフトウェアを含まない bit ファイルが生成されます。[Update Bitstream with Processor Data]をダブルクリックして下さい。ハードウェアでつくった bit ファイルの中にアプリケーションを書き込みます。`top_download.bit` が出来上がります。

是非、Linux 開発編でこの bit ファイルを使ってみてください。

どうしてもうまくいかない場合は付属 CD-ROM に FPGA プロジェクトを収録[1]しているので、比較してみてください。

[1] `"\suzaku-starter-kit\fpga\x.x\sz***\sz***-sil-gpio_control-yyyymmdd.zip"`に収録しています。

14.6. SUZAKU I/O ボードを使おう

スターターキットガイドを読み終わって、いざ SUZAKU を拡張しよう、といってもまだまだ難しいといった方も多いと思います。そんな方のために SUZAKU には本書で使用した LED/SW ボードの他に、A/D ボード、AV ボードといった拡張ボードが存在します。

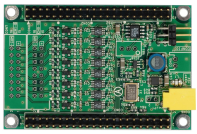
LED/SW ボードと同様に A/D ボード、AV ボードも回路図、IP コア、Linux デバイスドライバ、アプリケーションを公開しており、これらを参考にカスタマイズして SUZAKU に新たな機能を追加することが可能です。

ここでは、少しだけ A/D ボードと AV ボードの紹介をします。詳細については SUZAKU 公式サイトをご参照ください。

14.6.1. A/D ボードを使おう

A/D ボードは A/D コンバータ(分解能 12bit、変換速度 MAX 150Ksps)を 8 チップを搭載したボードです[2]。SUZAKU との組み合わせにより、簡単にネットワーク計測機器を開発することができます。A/D ボードの仕様は以下のとおりです。

表 14.6 A/D ボードの仕様

モデル	 図 14.36 SID00-U00
分解能	12bit
入力数	8 チャンネル
変換速度	150Ksps(MAX)
チャンネル間同期性	全チャンネル同時サンプリング
A/D コンバータ	LTC1860LCMS8(メーカ：リニアテクノロジ)
A/D コンバータ実装数	8 個
サンプリング用源発振周波数	24MHz 発振器実装
SUZAKU 拡張 I/O 使用数	19 ピン
外形サイズ	基板サイズ 72x47[mm]
電源	5V±5%
消費電流	約 30mA(SUZAKU の消費電流は含まず)
SUZAKU 供給電源	+3.3V 1.2A 供給回路搭載
使用温度範囲	0 〜+60℃

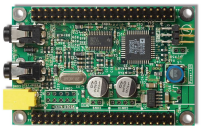
A/D ボードの FPGA プロジェクトの作成の方法は A/D ボードの IP コアのデータシートの Implementation の章に記載しています。A/D ボードの IP コア(xps_sid00_vx.xx)は付属 CD-ROM[3]に収録しています。SUZAKU 公式サイトからもダウンロードすることができます。

[2]受注生産となりますが、16bit タイプ、16 チップタイプもあります。
[3]A/D ボードの IP コアは"\suzaku-io-boards\ad\fpga\xps_sid00_vx_xx_x.zip"に収録しています。

14.6.2. AV ボードを使おう

AV ボードは AV 端子とコンバータを備えたボードです。SUZAKU でオーディオとビデオの信号処理が可能になり、IP 電話やネットワークカメラ、画像処理などの開発に最適です。AV ボードの仕様は以下のとおりです。

表 14.7 AV ボードの仕様

モデル		 図 14.37 SIV00-U00
ビデオ	対応規格	NTSC
	入力抵抗	75Ω
	入力レンジ	1.92Vp-p
	デコーダ IC	ADV7180(メーカ：アナログデバイセス)
	出力抵抗	75Ω
	出力レンジ	47mV ～ 1268mV
	エンコーダ IC	ADV7171(メーカ：アナログデバイセス)
オーディオ	Line 入力レンジ(LR)	1VRMS
	Line 出力レンジ(LR)	1VRMS
	マイクロホン入力	コンデンサマイク
	ヘッドホン最大出力	30mV(32Ω)、40mV(16Ω)
	コーデック IC	TLV320AIC23B(メーカ：テキサスインスツルメンツ)
	対応サンプリング	8k、32k、48k、96kHz(44.1kHz 系は未サポート)
AV コネクタ		4 極 3.5mm ピンジャック(入力、出力)
SUZAKU 拡張 I/O 使用数		31 ピン
外形サイズ		基板サイズ 72x47[mm]
電源		5V±5%
消費電流		約 300mA(SUZAKU の消費電流は含まず)
SUZAKU 供給電源		+3.3V 1.2A 供給回路搭載
使用温度範囲		0 ～+60℃

AV ボードの FPGA プロジェクトの作成の方法は AV ボードの IP コアのデータシートの Implementation の章に記載しています。AV ボードの IP コア(xps_siv00_vx.xx)は付属 CD-ROM[4]に収録しています。SUZAKU 公式サイトからもダウンロードすることができます。

[4]AV ボードの IP コアは"\suzaku-io-boards\av\fpga\xps_siv00_vx_xx_x.zip"に収録しています。

14.7. Microblaze に MMU を搭載する

SZ130 では CPU に Microblaze、OS に μCLinux を使用しています。μCLinux では必要ないので搭載していませんが、Microblaze にはオプションで MMU(メモリ管理ユニット)を搭載することができます。MMU があると、標準的な Linux を使用することができるようになります。まだ正式版ではありませんが、SUZAKU-S 用 MMU 対応 Linux テストリリース [http://suzaku.atmark-techno.com/dev/suzaku-s_mmu_linux_testrelease]からダウンロードすることができます。

14.7.1. MMU を実装する

まずは SZ130 のデフォルトの FPGA プロジェクトを XPS で開いてください。デフォルトの FPGA プロジェクトは付属 CD-ROM[5]もしくは SUZAKU 公式サイト [http://suzaku.atmark-techno.com/series/stk/download]にあります。

microblaze_i の上で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。microblaze の設定画面が立ち上がるので[MMU]タブをクリックし、Memory Management を[VIRTUAL]に変更し、[OK]をクリックしてください。これで出来上がりです。

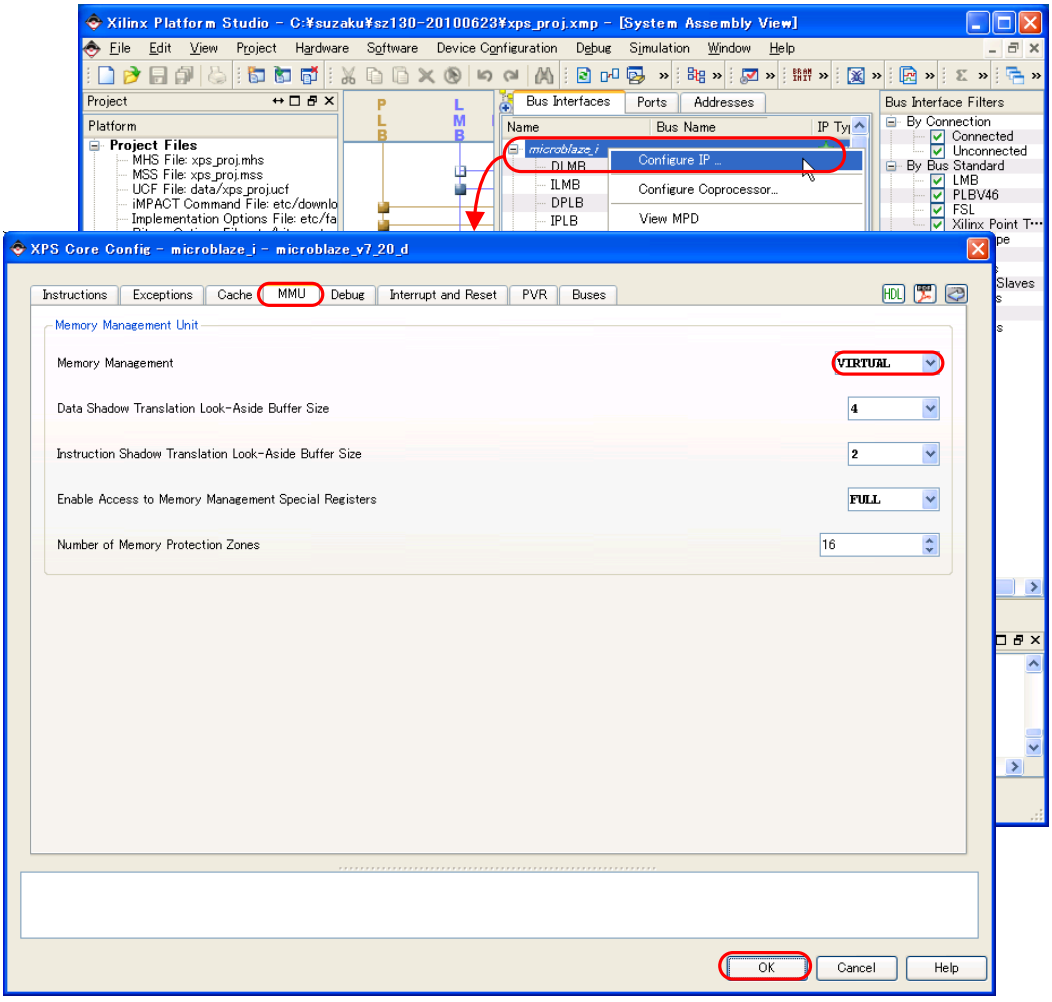


図 14.38 MMU ON

[5]"\suzaku\fpga_proj\x.x\sz130\sz130-yyyymmdd.zip"に収録しています。

14.8. これから先は・・・

本書はこれで終わりです。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 に搭載されるプロセッサの使用方法、そして SUZAKU の効果的な使い方は身についたでしょうか。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、情報を発信し、SUZAKU 開発者のスペシャリストを目指してください。

本書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]

本書と対になる SUZAKU スターターキットガイド(Linux 開発編)では本書とは違った切り口で SUZAKU の開発を行うので、是非ご一読ください！

付録 A SUZAKU + LED/SW ボードのピンアサイン

SUZAKU と LED/SW ボードの全ピンアサインを載せます。SUZAKU で新たに何かを開発する時などにご参照ください。

A.1. SUZAKU のピンアサイン

A.1.1. 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、A1-10PA-2.54DSA(メーカ：ヒロセ(相当品))です。

表 A.1 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 A.2 SUZAKU CON1 RS-232C

番号	信号名	I/O	機能	FPGA 接続ピン番号	
				SZ130	SZ410
1			空き		
2			空き		
3	RXD	I		C12	Y4
4	RTS	O		B13	V4
5	TXD	O		A13	U4
6	CTS	I		D12	V3
7			空き		
8			空き		
9	GND		グランド		
10	+3.3VOUT	O	内部ロジック用電源出力+3.3V		

A.1.2. SUZAKU CON2 外部 I/O、フラッシュメモリ用コネクタ

外部 I/O 及びフラッシュメモリ用コネクタです。LED/SW ボードの CON2 とコネクタ接続します。

表 A.3 SUZAKU CON2 外部 I/O、フラッシュメモリ用コネクタ

番号	I/O	機能	FPGA 接続ピン番号	
			SZ130	SZ410
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	CLK	CLK
4	I	FPGA プログラム用	D	D
5	O	FPGA プログラム用	DO	DO
6	I	FPGA プログラム用	nCS	nCS
7	I/O	外部 I/O	N5	E14
8	I/O	外部 I/O	N4	D15
9	I/O	外部 I/O	M6	E15
10	I/O	外部 I/O	M5	F15
11	I/O	外部 I/O	M3	P4
12	I/O	外部 I/O	M4	P5
13	I/O	外部 I/O	L5	P1
14	I/O	外部 I/O	L6	P2
15	I/O	外部 I/O	L4	L2
16	I/O	外部 I/O	L3	M2
17	I/O	外部 I/O	L2	N2
18	I/O	外部 I/O	L1	N3
19		グランド or 誤挿入防止用		
20	I/O	外部 I/O(GCLK)	C9	Y7
21		グランド		
22	I/O	外部 I/O(GCLK)	D9	W7
23	I/O	外部 I/O	K5	N4
24	I/O	外部 I/O	K6	N5
25	I/O	外部 I/O	K4	M3
26	I/O	外部 I/O	K3	M4
27	I/O	外部 I/O	J2	H4
28	I/O	外部 I/O	J1	H5
29	I/O	外部 I/O	F9	E2
30	I/O	外部 I/O	E9	D2
31	I/O	外部 I/O	A10	U9
32	I/O	外部 I/O	B10	V10
33	I/O	外部 I/O	D11	L1
34	I/O	外部 I/O	C11	M1
35	I/O	外部 I/O	F11	G4
36	I/O	外部 I/O	E11	G5
37	I/O	外部 I/O	E12	G2
38	I/O	外部 I/O	F12	F2
39	I/O	外部 I/O	B11	F1
40	I/O	外部 I/O	A11	E1

番号	I/O	機能	FPGA 接続ピン番号	
			SZ130	SZ410
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		

A.1.3. 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 とコネクタ接続します。

表 A.4 SUZAKU CON3 外部 I/O コネクタ

番号	I/O	機能	FPGA 接続ピン番号	
			SZ130	SZ410
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	B14	K3
6	I/O	外部 I/O	A14	K2
7	I/O	外部 I/O	D14	K1
8	I/O	外部 I/O	C14	J2
9	I/O	外部 I/O	B16	H3
10	I/O	外部 I/O	A16	H2
11	I/O	外部 I/O	C18	L5
12	I/O	外部 I/O	C17	L4
13	I/O	外部 I/O	D17	K5
14	I/O	外部 I/O	D16	K4
15	I/O	外部 I/O	F15	J6
16	I/O	外部 I/O	F14	J5
17	I/O	外部 I/O	G14	H1
18	I/O	外部 I/O	G13	G1
19	I/O	外部 I/O	F18	F3
20	I/O	外部 I/O	F17	E3
21	I/O	外部 I/O	G15	C3
22	I/O	外部 I/O	G16	C2
23	I/O	外部 I/O(GCLK)	E10	W5
24		グランド		
25	I/O	外部 I/O(GCLK)	D10	Y5
26		グランド		
27	I/O	外部 I/O	H14	B2
28	I/O	外部 I/O	H15	C1
29	I/O	外部 I/O	H16	A3
30	I/O	外部 I/O	H17	B3

番号	I/O	機能	FPGA 接続ピン番号	
			SZ130	SZ410
31	I/O	外部 I/O	J12	J4
32	I/O	外部 I/O	J13	J3
33	I/O	外部 I/O	J15	D4
34	I/O	外部 I/O	J14	D3
35	I/O	外部 I/O	J17	D5
36	I/O	外部 I/O	J16	E5
37	I/O	外部 I/O	K15	B4
38	I/O	外部 I/O	K14	C4
39	I/O	外部 I/O	K13	C6
40	I/O	外部 I/O	K12	C5
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		

A.1.4. 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 A.5 SUZAKU CON4 外部 I/O コネクタ

番号	I/O	機能	FPGA 接続ピン番号	
			SZ130	SZ410
1		空き		
2		空き		
3	I/O	外部 I/O	L18	B5
4	I/O	外部 I/O	L17	A5
5	I/O	外部 I/O	L16	A6
6	I/O	外部 I/O	L15	B6
7	I/O	外部 I/O	N18	D8
8	I/O	外部 I/O	M18	C8
9	I/O	外部 I/O	M16	M5
10	I/O	外部 I/O	M15	M6
11	I/O	外部 I/O	P17	C13
12	I/O	外部 I/O	P18	D13

A.1.5. 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 A.6 SUZAKU CON5 外部 I/O コネクタ

番号	I/O	機能	FPGA 接続ピン番号	
			SZ130	SZ410
1		グランド		

番号	I/O	機能	FPGA 接続ピン番号	
			SZ130	SZ410
2	O	内部ロジック用電源出力 +3.3V		
3	I/O	外部 I/O	M14	F4
4	I/O	外部 I/O	M13	F5
5	I/O	外部 I/O	R15	F6
6	I/O	外部 I/O	R16	E6
7	I/O	外部 I/O	R18	D6
8	I/O	外部 I/O	T18	E7
9	I/O	外部 I/O	U18	D9
10	I/O	外部 I/O	T17	C9
11	I/O	外部 I/O	T15	C12
12	I/O	外部 I/O	R14	D12

A.1.6. 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。

A.1.7. SUZAKU CON7 FPGA JTAG 用コネクタ

FPGA JTAG 用コネクタです。コンフィギュレーションする時は SUZAKU JP1、JP2 をショートしてください。

表 A.7 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	O	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

A.1.8. 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 A.8 SUZAKU D1、D3 LED

番号	I/O	機能	FPGA 接続ピン番号	
			SZ130	SZ410
D1	O	ユーザーコントロール LED	T3	T4
D3	O	SUZAKU ボードに + 3.3V が供給されると点灯		

A.1.9. SUZAKU JP1、JP2 設定用ジャンパ

ジャンパによりオートブートモード、ブートローダモード、FPGA コンフィギュレーション待ちの 3 つの状態に設定します。

表 A.9 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、フラッシュメモリ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)

A.1.10. SUZAKU L2 Ethernet 10BASE-T/100BASE-TX

ボード側で使用しているコネクタは J0026D21B(メーカ：PULSE)です。

表 A.10 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)

A.2. LED/SW ボードのピンアサイン

A.2.1. 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。

A.2.2. LED/SW CON2 SUZAKU 接続コネクタ

SUZAKU CON2 と接続します。

表 A.11 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	
				SZ130	SZ410
1	GND		グランド		
2	+3.3V	I	+3.3V SUZAKU 側から供給		
3	CONF_C			CLK	CLK
4	CONF_I			D	D
5	CONF_O			DO	DO
6	CONF_S			nCS	nCS
7	NC			N5	E14
8	UART3	I	RTS	N4	D15

番号	信号名	I/O	機能	FPGA 接続ピン番号	
				SZ130	SZ410
9	UART2	O	TXD	M6	E15
10	UART1	O	CTS	M5	F15
11	UART0	I	RXD	M3	P4
12	NC			M4	P5
13	SEG7	O	セグメント DP High"で点灯	C5	P1
14	SEG6	O	セグメント G High で点灯	L6	P2
15	SEG5	O	セグメント F High で点灯	L4	L2
16	SEG4	O	セグメント E High で点灯	L3	M2
17	SEG3	O	セグメント D High で点灯	L2	N2
18	SEG2	O	セグメント C High で点灯	L1	N3
19			誤挿入防止用		
20	SEG1	O	セグメント B High で点灯	C9	Y7
21	GND		グランド		
22	SEG0	O	セグメント A High で点灯	D9	W7
23	NC			K5	N4
24	nSEL2	O	7 セグメント LED3 Low でコモン選択	K6	N5
25	nSEL1	O	7 セグメント LED2 Low でコモン選択	K4	M3
26	nSEL0	O	7 セグメント LED1 Low でコモン選択	K3	M4
27	NC			D8	H4
28	nCODE3	I	ロータリスイッチ 4 ビット目選択時 Low	J1	H5
29	nCODE2	I	ロータリスイッチ 3 ビット目選択時 Low	F9	E2
30	nCODE1	I	ロータリスイッチ 2 ビット目選択時 Low	E9	D2
31	nCODE0	I	ロータリスイッチ 1 ビット目選択時 Low	A10	U9
32	NC			B10	V10
33	nSW2	I	押しボタンスイッチ SW3 押下で Low	D11	L1
34	nSW1	I	押しボタンスイッチ SW2 押下で Low	C11	M1
35	nSW0	I	押しボタンスイッチ SW1 押下で Low	F11	G4
36	NC			E11	G5
37	nLE0	O	単色 LED(緑)D1 Low で点灯	E12	G2
38	nLE1	O	単色 LED(緑)D2 Low で点灯	F12	F2
39	nLE2	O	単色 LED(緑)D3 Low で点灯	B11	F1
40	nLE3	O	単色 LED(緑)D4 Low で点灯	A11	E1
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		

A.2.3. LED/SW CON3 SUZAKU 接続コネクタ

SUZAKU CON3 と接続します。

表 A.12 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			

番号	信号名	I/O	機能
42			
43	+3.3V	I	+3.3V SUZAKU 側から供給
44	GND		グランド

A.2.4. 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。信号についての詳細は CON2 を参照してください。1 ～ 6 ピンにフラッシュメモリ書き込み用コネクタが実装されています。SUZAKU と接続時、フラッシュメモリにデータを書き込めます。書き込む時は SUZAKU JP1、JP2 をショートしてください。

表 A.13 LED/SW CON4 フラッシュメモリ書き込み用コネクタ

番号	信号名	I/O	機能	FPGA 接続ピン番号	
				SZ130	SZ410
1	GND				
2	+3.3V				
3	CONF_C	I		CLK	
4	CONF_I	I		D	
5	CONF_O	O		DO	
6	CONF_S	I		nCS	

A.2.5. 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 は添付品をご使用ください。(+5V 出力 EIAJ #2)

表 A.14 LED/SW CON6 +5V 入力コネクタ

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

図 A.1 +5V センタープラスピン

A.2.6. LED/SW CON7 RS-232C コネクタ

D-Sub9 ピンが実装されています。

表 A.15 LED/SW CON7 RS-232C コネクタ

番号	信号名	I/O	機能	FPGA 接続ピン番号	
				SZ130	SZ410
1					
2	UART0	I	RXD	M3	P4
3	UART2	O	TXD	M6	E15

番号	信号名	I/O	機能	FPGA 接続ピン番号	
				SZ130	SZ410
4					
5	GND		グランド		
6					
7	UART3	O	RTS	N4	D15
8	UART1	I	CTS	M5	F15
9					

A.2.7. 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 A.16 LED/SW 7 セグメント LED セレクタ

番号	信号名	I/O	機能	FPGA 接続ピン番号	
				SZ130	SZ410
LED1	nSEL0	O	LED1 コモン Low で選択	K3	M4
LED2	nSEL1	O	LED2 コモン Low で選択	K4	M3
LED3	nSEL2	O	LED3 コモン Low で選択	K6	N5

A.2.8. 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 A.17 LED/SW LED1~3 7 セグメント LED

番号	信号名	I/O	機能	FPGA 接続ピン番号	
				SZ130	SZ410
A	SEG0	O	セグメント A High で点灯	D9	W7
B	SEG1	O	セグメント B High で点灯	C9	Y7
C	SEG2	O	セグメント C High で点灯	L1	N3
D	SEG3	O	セグメント D High で点灯	L2	N2
E	SEG4	O	セグメント E High で点灯	L3	M2
F	SEG5	O	セグメント F High で点灯	L4	L2
G	SEG6	O	セグメント G High で点灯	L6	P2
DP	SEG7	O	セグメント DP High で点灯	L5	P1

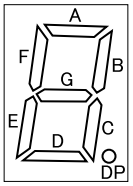


図 A.2 7 セグメント LED

A.2.9. 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 A.18 LED/SW D1 ～ 4 単色 LED(緑)

番号	信号名	I/O	機能	FPGA 接続ピン番号	
				SZ130	SZ410
D1	nLE0	O	単色 LED(緑)D1 Low で点灯	E12	G2
D2	nLE1	O	単色 LED(緑)D2 Low で点灯	F12	F2
D3	nLE2	O	単色 LED(緑)D3 Low で点灯	B11	F1
D4	nLE3	O	単色 LED(緑)D4 Low で点灯	A11	E1

A.2.10. 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 A.19 LED/SW SW1 ～ 3

番号	信号名	I/O	機能	FPGA 接続ピン番号	
				SZ130	SZ410
SW1	nSW0	I	押しボタンスイッチ SW1 押下で Low	F11	G4
SW2	nSW1	I	押しボタンスイッチ SW2 押下で Low	C11	M1
SW3	nSW2	I	押しボタンスイッチ SW3 押下で Low	D11	L1

A.2.11. 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 A.20 LED/SW SW4

番号	信号名	I/O	機能	FPGA 接続ピン番号	
				SZ130	SZ410
SW4	nCODE0	I	ロータリコードスイッチ 2^0 選択で Low	A10	U9
	nCODE1	I	ロータリコードスイッチ 2^1 選択で Low	E9	D2
	nCODE2	I	ロータリコードスイッチ 2^2 選択で Low	F9	E2
	nCODE3	I	ロータリコードスイッチ 2^3 選択で Low	J1	H5

参考文献

[1] 「*エンベデッドシステムツールリファレンスマニュアル*」. Xilinx 株式会社.

[2] 「*開発システムリファレンスガイド*」. Xilinx 株式会社.

[3] 「*Platform Studio Help*」. Xilinx 株式会社.

[4] 「*ISE Help*」. Xilinx 株式会社.

[5] 「*VHDL によるハードウェア設計入門*」. CQ 出版社. 長谷川裕恭 .

[6] 「*デザインウェーブマガジン 2004 年 5 月号ソフト・マクロの CPU で Linux を動かす(前編)*」. CQ 出版社.

[7] 「*改訂 初めてでも使える HDL 文法ガイド*」. CQ 出版社.

改訂履歴

バージョン	年月日	改訂内容
1.0.0	2006/07/14	・初版作成
1.0.1	2006/07/19	・誤記訂正
1.0.2	2006/07/24	・ピンアサイン訂正(CON4 の 9、10 ピン)
2.0.0	2006/08/11	・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	2006/08/18	・TE7720 の図の文字化けを修正
2.0.2	2006/08/23	・誤記訂正
2.0.3	2006/10/18	・Linux 編にあわせて章構成変更 ・保証に関する注意事項追記 ・半田付けの際の注意を追記 ・コネクタ説明箇所に JTAG、Flash メモリ書き込みについて追記 ・8.2i 対応のための内容を追記
2.1.2	2006/11/30	・構成を大幅変更(以下重要な変更のみ記) ・ダイナミック点灯、単色 LED 順次点灯、デコーダの VHDL 修正 ・SZ130 のメモリマップ 誤記訂正 ・SZ310 の構成図 誤記訂正 ・IP コアハード編の項 誤記訂正 ・EDK の項 ソフトウェアについて追記 ・EDK の項 IP コアの説明を追記 ・シミュレーションの記述を追記 ・SUZAKU の書き込み方を一つにまとめて記述 ・デバッグの項追加 ・参考文献を追加 ・EDK を ISE のサブモジュールにする方法を追加
2.1.3	2006/12/06	・メモリマップ誤記訂正 ・JTAG、SPI のピンアサインの図に色づけ ・その他誤記訂正 ・表紙デザイン改版
2.1.4	2007/01/19	・EDK の使い方の章に SZ310 のリンカースクリプトについて追記 ・EDK の使い方の章に BSB の使い方を追加
2.1.5	2007/02/16	・SUZAKU についての章に SUZAKU 全体ブロック図等追記 ・TIPS パラレルポートがなくても追加 ・TIPS FPGA の bin ファイルの作り方追加 ・TIPS XMD コマンド追加 ・TIPS MicroBlaze 追加 ・その他修正、構成変更
2.1.6	2007/03/16	・2 段階 Boot の図を修正、誤記訂正
2.1.7	2007/04/20	・BSB で MicroBlaze の章に PowerPC の内容を追記 ・デバッグの章の内容修正
2.1.8	2007/07/20	・ISE/EDK9.1i にて内容確認&内容修正 ・静的割り込みから動的割り込みに変更 ・UCF に CMOS 3.3V を明記 ・SDK を使ってデバッグの章を追加 ・その他文章訂正

2.2.0	2007/10/10	・SZ410 対応のための全面変更
2.2.1	2007/10/19	・TIPS 全 IP 表示追加
2.2.2	2007/12/14	・ISE/EDK9.2i にて内容確認&内容修正 ・9.2i 用 BSB の章追記 ・IPIF のキャプチャを 9.2i に変更 ・デバッグの章修正 ・9.2i で SZ130 でスロットのプログラムが 8kB を超えるため ・16kB に BRAM を増やすよう内容追記 ・TIPS 目次追加 ・必要なものに Parallel Cable Fly Leads 追記
2.3.0	2008/02/15	・SZ410 のデフォルトプロジェクト変更にともなう内容修正 ・BBoot2.3 -> 2.4 にともなう内容修正
2.3.1	2008/03/14	・SZ410 のデフォルトプロジェクト変更にともなう内容修正 ・MPMC で BRAM のリソースを使用しないように変更
2.3.2	2008/04/21	・「6.3.1. BBoot で書き換える」修正 ・BBoot にファイルのチェックサム機能が新たに追加されたため内容修正
2.3.3	2008/06/20	・表紙に ISE/EDK の対応バージョン明記 ・誤記訂正
2.4.0	2008/06/20	・ISE/EDK10.1i 用に内容修正
2.4.1	2008/07/18	・誤記修正
2.4.2	2008/09/26	・タイトルを英語表記からカタカナ表記に ・BRAM を増やした場合のリンカスクリプト更新方法を追記
2.4.3	2009/01/09	・誤記修正
2.4.4	2009/03/19	・「2. LED/SW ボードについて」2 章のタイトル修正 ・参照先を記述する際の表記を統一 ・表記ゆれを修正
2.4.5	2009/07/17	・本文のレイアウト統一 ・文字化け修正 ・文字が読みにくい画像の差し替え
2.4.6	2009/07/29	・製品保証に関する記載を http://www.atmark-techno.com/support/warranty-policy に移動(2009/08/03 適用)
2.4.7	2010/09/17	・表のレイアウト統一 ・外部リンクを付属 CD-ROM へのリンクに変更 ・表記ゆれを修正 ・誤記修正
3.0.0	2010/09/24	・IDS11.5i 用に書き直し

SUZAKU スターターキットガイド(FPGA 開発編)
Version 3.0.0
2010/09/24

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 TEL 011-207-6550 FAX 011-207-6570