平成 24 年度　「東日本大震災からの復興を担う専門人材育成支援事業」

実践！自動車組込み技術者入門

ＦＰＧＡとマイコンの連携システム

ハードウェア編

~The first step is the only difficulty.

東北の復興を担う自動車組込みエンジニア育成支援プロジェクト

Content

0 はじめに

近年、コストの削減や、設計期間の短縮などにより、組込システムが多くのシステムに利用されていますが、システムの高機能化と共に、規模が拡大していき、従来のマイコンによる処理だけでは対応出来ない機能が出てきました。

特に注目を集めている衝突防止システムや、車の周辺環境を複数のカメラで把握し苦手な駐車スペースへ誘導するパーキングアシストシステムなど、画像信号の高速処理が必要となる部分（画像解析等）や、並列同時処理が必要な部分は、マイコンのみのシステムでは信号処理の負担が大き過ぎ、ハードウェアによる機能の実現を計ってきました。

しかし、ＡＳＩＣの様な専用デバイスでは、開発コストや期間が、現在の要求に対応出来きず、開発が容易で多機能な要求に対応できるデバイスが求められています。

そこに、近年急速に技術開発が進んでこの様な要求に対応できるデバイスとして注目されているのがＦＰＧＡというデバイスです。

現在、多くの製品で、このＦＰＧＡとマイコンが連携して高性能な機能を実現しています。

本書では、実際の製品開発に用いられる VerilogHDL による FPGA 開発とマイコンシステムを用いた連携システムにより、具体的な実装方法からアプリケーションの開発までを幅広く説明しています。

始めに、このＦＰＧＡとマイコンの連携システムの開発方法の説明をし、FPGA での簡単な距離計測システムを設計した後、マイコンと連携させた、より高機能なシステムを学びます。

FPGA の利用方法では、様々な書物が提供されています。実際に実装する場合になると、各メーカー毎の開発環境によって異なるため、本書では実際の製品開発の現場でも用いられるザイリンクス社が無償で提供している ISE@webPACK と VerilogHDL というハードウェア開発言語を用いて、入門者が最も手間の掛かる FPGA 実装の方法について具体的に説明しています。

本書が、車載システム開発の FPGA とマイクロコンピュータの連携システムを始めようという技術者、学生の良き参考書になれば幸いです。

第一章「FPGA とマイコンの連携システム」では、FPGA とマイコンが連携することの利点について、その理由を説明します。

第二章「FPGA 概論」、第三章「FPGA 開発環境」では、FPGA の特徴と学習用 FPGA ボードキットの製作手順と、ISE と呼ばれる HDL で記述されたソースコードから論理合成、インプリメンテーション、回路データの書き込み迄を統合した開発環境の概要と HDL の種類とその特徴を説明します。

第四章「VerilogHDL による信号処理」では、実際に演習を通して VerilogHDL による記述と実装方法を説明します。

第五章「超音波距離計の仕様」第六章「シリアルインターフェースモジュール」では、マイコンとの連携システムのために、超音波センサーを制御して距離計測を行うための方法と、計測データをマイコンシステムに転送するためのインターフェースの仕様を説明します。

1 FPGA とマイコンの連携システム

マイコンによる組込システムが多くのシステムに利用されていますが、マイコンにおける処理では、ソフトウェア処理のウィークポイントである、信号の高速処理が必要となる部分（画像解析等）や、並列同時処理が必要な部分において対応出来ない事が多々あります。

その様な場合、ハードウェアによるＡＳＩＣ等により機能の実現を計ってきました。しかし、ＡＳＩＣの様な専用デバイスでは、開発コストや期間が、現在の要求に対応出来なくなって来ました。

そこで注目されてきたのが、より簡単に機能検証や仕様変更に対応できるＦＰＧＡというデバイスです。

現在、多くの製品では、このＦＰＧＡとマイコンが連携して高性能な機能を実現しています。

マイコンによるソフトウェア処理に於いては、仕様の変更などに柔軟に対応出来る利点があり、多くのシステムに導入されてきました。

しかし、ソフトウェアの弱点である以下の項目が、近年のシステムでは問題になっています。

① －命令の読込・解析・実行－という手順に於いて一定の処理時間が必要となる。
② リアルタイム OS を用いたシステムでも、厳密には並列処理ではなく時分割処理になっている。

このような課題の解決にはハードウェアとの連携が必要であり、以前よりトレードオフと呼ばれる機能分割が行われてきました。

しかし、ハードウェアによる機能実現は、ソフトウェアに比べ高価になる事が大半で、可能な限りソフトウェアによる機能実現を進めてきました。しかし、最近のハードウェアによる機能実現の手法が大きく進み、特にＦＰＧＡと呼ばれるデバイスが脚光をあびております。

このような FPGA との連携システムの実現で上記のような問題が解消可能になってきました。

2 FPGA 概論

FPGA とは、「Field Programmable Gate Array」の略で、一度書き込みを行った後でも、仕様変更などにより回路構成が変更された後、再度書き換えが可能なデバイスです。また、FPGA では、IP コアと呼ばれる機能単位にまとめられたら回路情報を組込むことにより、同じ回路構成のままで異なった機能を、複数のシステムユニットを搭載しなくても実現可能になります。

FPGA が登場する前では、PLD の PAL などでは、内部回路をプロダクト・タームと呼ばれる積和形式（AND-OR）による論理式で実現してきましたが、FPGA では LUT（look-up table）と呼ばれる小規模なメモリーで実現しています。

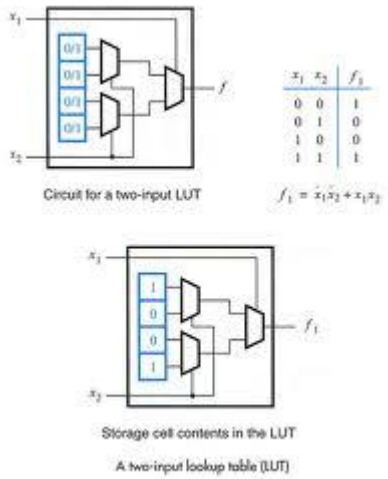


図 2.1-1　LUT の構造

LUT は入力パラメータをアドレスとして、そのパラメータによる論理式の値をメモリーにあらかじめ格納しておく事で、論理回路（組み合わせ）による複雑な回路構成を省略し、即座に結果を得ることが可能になる方法です。

次の第３章から、演習を通して具体的に FPGA の設計について解説していきます。

2.1 FPGA 学習ボード

FPGA 学習ボード

FPGA 学習ボードは、組み立てキットとなっており、マニュアルを参考にしながら、はんだ付けをして組み立てます。また、汎用部品を多く使用しているため、回路が分かり易く、ハードウェアの学習も同時に行うことができます。

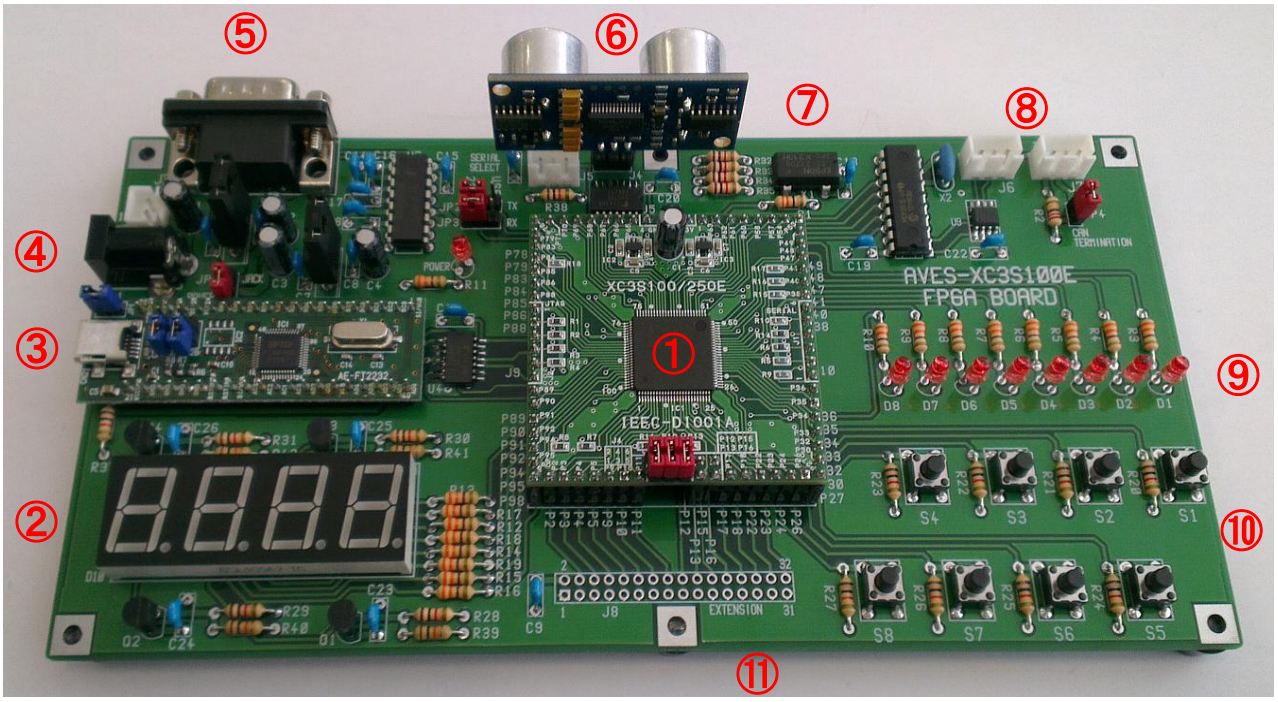


図 2.1　FPGA 学習ボード

2.1.1　機能の説明、

① FPGA として、Xilinx 社製の XC3S100E-4VQG100C を搭載しています。
② 大型の緑色 7 セグメント LED です。
③ USB により PC と接続します。JTAG インターフェイスによる FPGA プログラミングや、仮想 COM ポートによる通信を行うことができます。また、USB 給電により FPGA 学習ボードを動かすことも可能です。
④ 電源用の DC ジャックです。USB による給電を行わない場合、DC ジャックに 6V～12V の AC アダプタを接続することで FPGA 学習ボードを動作させることができます。
⑤ RS-232C 通信用の Dsub9 ピンコネクタです。RS-232C トランシーバを介して FPGA に接続されており、FPGA からシリアル通信を行うことができます。
⑥ Parallax 社製の超音波センサです。超音波の反射を利用して、物体までの距離を測ることができます。
⑦ FPGA にクロックを供給するオシレータです。FPGA 学習ボードでは 33.333MHz を搭載しています。
⑧ CAN(Control Area Network)用のコネクタです。自動車等に広く使用されている CAN 通信を行うことができます。CAN 通信に必要なプロトコルは、マイクロチップ社製の MCP2515 に搭載されているため、簡単に CAN 通信を行うことができます。
⑨ FPGA から直接点灯可能な赤色 LED が 8 個搭載されています。
⑩ FPGA から直接読み取り可能な SW が 8 個搭載されています。
⑪ FPGA 学習ボード上で使用していない XC3S100E の空きピンを拡張用コネクタとして取り出しています。

2.1.2　部品の確認

組み立てに入る前に、必要な部品がすべてそろっているか確認してください。
表 x にキットに含まれている部品の一覧を示します。

表 2.1.2　キット同梱部品一覧

数量	品名	型番	備考
4	100uF25V 電解コンデンサ	25PK100MEFC5X11	
22	0.1uF50V 積層セラミックコンデンサ	RPEF11H104Z2P1	
1	33Ω1/4W カーボン抵抗	RD-25TJ330	橙橙黒金
1	120Ω1/4W カーボン抵抗	RD-25TJ121	茶赤茶金
17	330Ω1/4W カーボン抵抗	RD-25TJ331	橙橙茶金
19	1KΩ1/4W カーボン抵抗	RD-25TJ102	茶黒赤金
4	10KΩ1/4W カーボン抵抗	RD25S 10K	茶黒橙金
1	FPGA 基板	IEEC-DI001A	
1	33.333MHz オシレータ	SG8002DC-33.333MHz-PCB	
1	10MHz セラミック発信子	CSTLS10M0G53-B0	
4	トランジスタ	2SA1015Y	
1	5V レギュレータ	NJU7223F50	
1	3.3V レギュレータ	NJU7223F33	
1	RS232C トランシーバ	ICL3232CPZ	
1	CAN コントローラ	MCP2515-I/P	
9	赤色 LED	OSDR3133A	
1	4 桁 7 セグメント LED	OSL40562-IG	
7	ジャンパピン	P-03688	
6	ゴム足	P-00298	
1	2.1mmDC ジャック	MJ-179P	
1	2P コネクタ	B2B-XH	
1	Dsub9 コネクタ オス	C-00644	
3	3P コネクタ	B3B-XH	
8	タクトスイッチ	DTS-6	
2	ネジ 10mmM3		
2	ナット M3		
1	USB ケーブル	C-05222	
1	超音波センサ	M-05400	
3	ピンソケット	C-05779	
3	ピンヘッダ	C-00167	
1	USB モジュール	M-02990	
1	プリント基板	AVES-XC3S1OOE	

2.1.3　事前準備

はんだ付け作業に入る前に、図 x のように、ピンヘッダとピンフレームを折って分割します。
ピンヘッダは、20 ピン 1 個、19 ピン 2 個、8 ピン 1 個、7 ピン 1 個、3 ピン 4 個、2 ピン 1 個にします。
ピンフレームは、20 ピン 1 個、19 ピン 2 個、8 ピン 1 個、7 ピン 1 個、3 ピン 1 個、2 ピン 1 個にします。

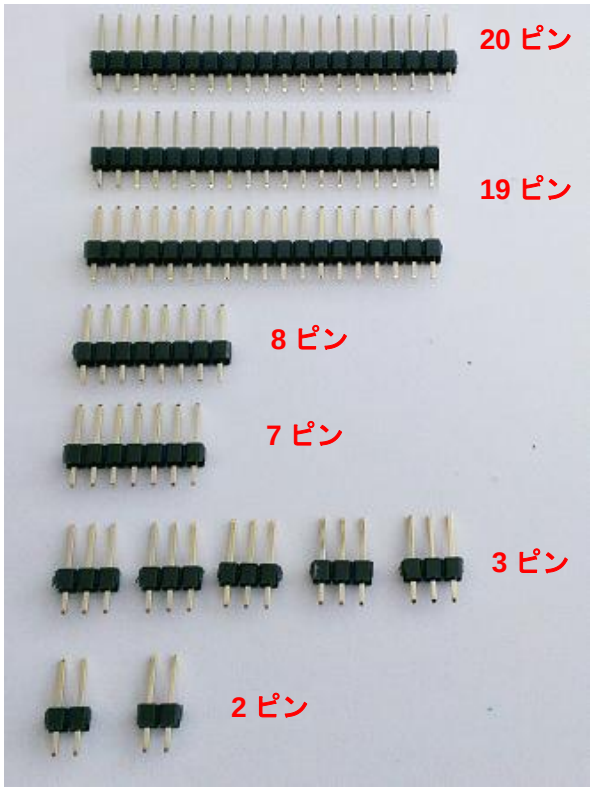


図 2.1.3-1　分割したピンヘッダ

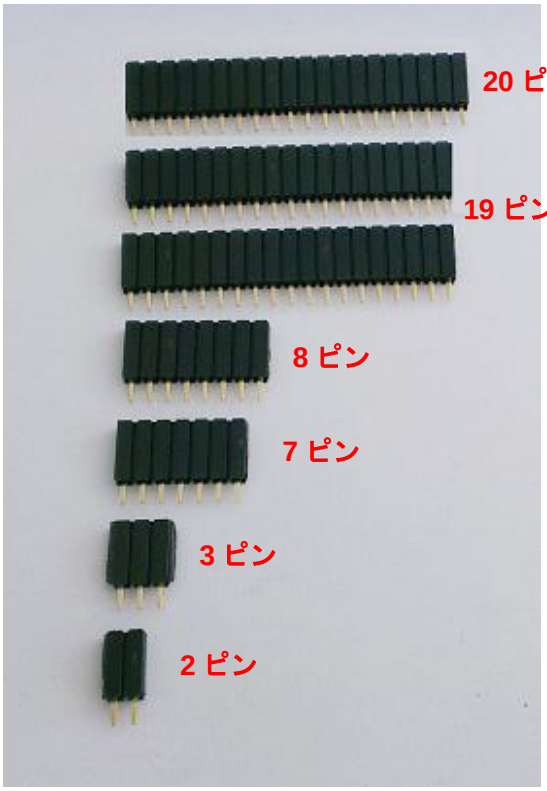


図 2.1.3-2　分割したピンフレーム

2.1.4　はんだ付けの前に、

図 x に示すプリント基板に対して、部品のはんだ付けを行います。図 x の様に、はんだ付けの難しい表面実装 IC は事前にはんだ付けを行ってあります。

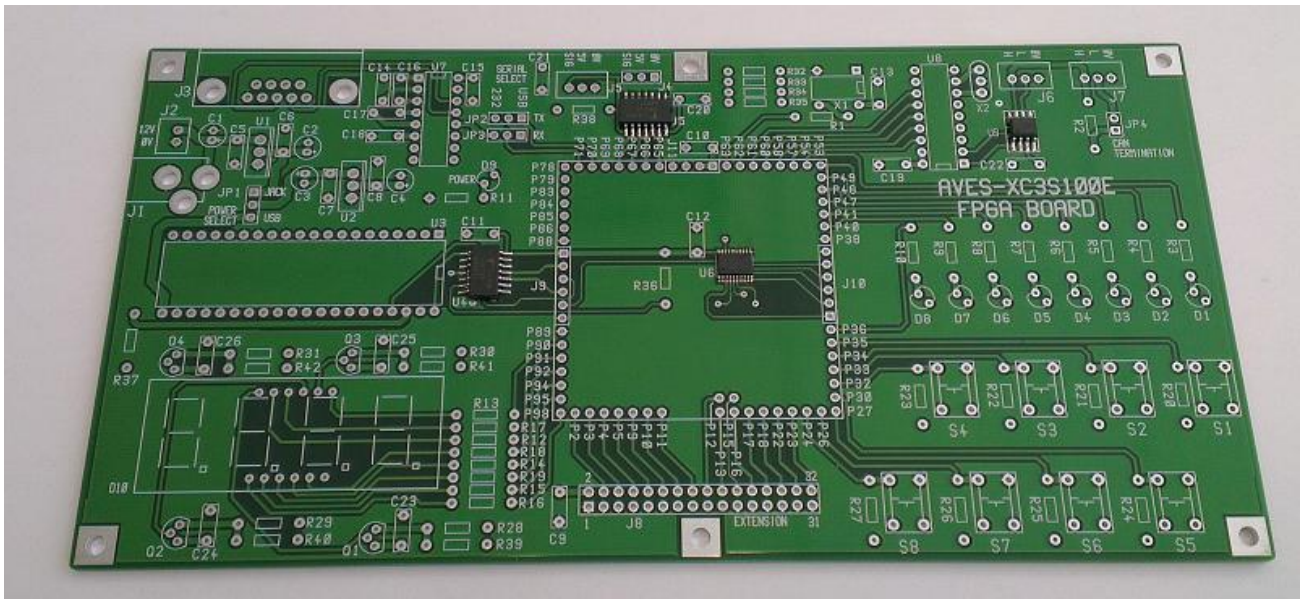


図 2.1.4　はんだ付け前のプリント基板

はんだ付けは、基本的に背の低い部品から順に行っていくと簡単です。

表 x に示す部品表の部品番号と、プリント基板に白文字で印字されているシルク番号を合わせて実装します。

表 2.1.4　FPGA 学習ボード部品表

部品番号	数量	品名	型番	メーカー	備考
U1	1	5V レギュレータ	NJU7223F50	JRC	
U2	1	3.3V レギュレータ	NJU7223F33	JRC	
U3	1	USBモジュール	M-02990	秋月	
U4-5	2	トライステートバッファ	TC74VHC125F(K,F)	東芝	実装済み
U6	1	フラッシュROM	xcf01svog20c	Xilinx	実装済み
U7	1	RS232Cトランシーバ	ICL3232CPZ	Intersil Americas Inc.	
U8	1	CANコントローラ	MCP2515-I/P	マイクロチップ	
U9	1	CANトランシーバ	MAX3051ESA+	マキシム	実装済み
Q1-4	4	トランジスタ	2SA1015Y	東芝	
D1-9	9	赤色LED	OSDR3133A	OSL	
D10	1	4桁7セグメントLED	OSL40562-IG	OSL	
X1	1	33.333MHzオシレータ	SG8002DC-33.333MHz-PCB	EPSON	
X2	1	10MHzセラミック発信子	CSTLS10M0G53-B0	ムラタ	
C1-4	4	100uF25V電解コンデンサ	25PK100MEFC5X11	ルビコン	
C5-26	22	0.1uF50V積層セラミックコンデンサ	RPEF11H104Z2P1	ムラタ	
R1	1	33Ω1/4Wカーボン抵抗	RD-25TJ330	KOA	
R2	1	120Ω1/4Wカーボン抵抗	RD-25TJ121	KOA	
R3-19	17	330Ω1/4Wカーボン抵抗	RD-25TJ331	KOA	
R20-38	19	1KΩ1/4Wカーボン抵抗	RD-25TJ102	KOA	
R39-42	4	10KΩ1/4Wカーボン抵抗	RD25S 10K	KOA	
SW1-8	8	タクトスイッチ	DTS-6	COSLAND	
J1	1	2.1mmDCジャック	MJ-179P	マル信無線	
J2	1	2P コネクタ	B2B-XH	日圧	
J3	1	Dsub9コネクタ オス	C-00644	秋月	
J4	1	3Px1 ピンソケット	C-05779	Useconn Electronics Ltd.	C-05779を分割して使用します。
J5-7	3	3P コネクタ	B3B-XH	日圧	
J8	1	未実装	-	-	
J9-10	2	6Px1 ピンソケット	C-05779	Useconn Electronics Ltd.	C-05779を分割して使用します。
J11	1	4Px1 ピンソケット	C-05779	Useconn Electronics Ltd.	C-05779を分割して使用します。
JP1-3	3	3Px1 ピンヘッダ	C-00167	Useconn Electronics Ltd.	C-00167を分割して使用します。
JP4	1	2Px1 ピンヘッダ	C-00167	Useconn Electronics Ltd.	C-00167を分割して使用します。
Px	59	ピンソケット	C-05779	Useconn Electronics Ltd.	C-05779を分割して使用します。

2.1.5　抵抗の実装

部品表の部品番号 R1～R42 をはんだ付けします。
極性はありません。どちら向きに実装しても大丈夫です。

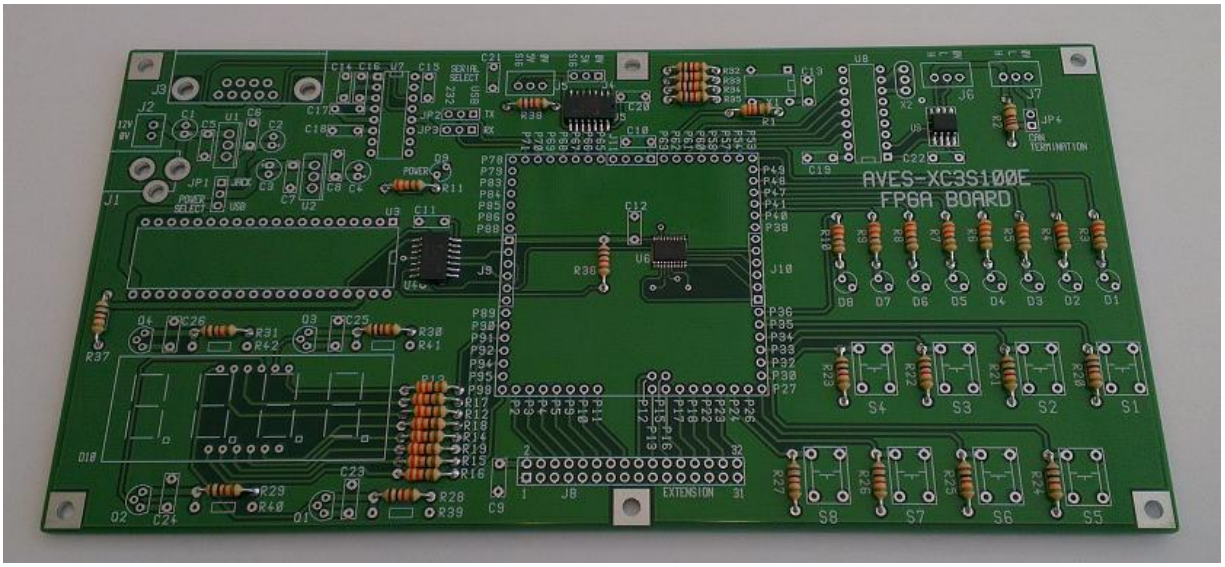


図 2.1.5　抵抗実装後のプリント基板

2.1.6　積層セラミックコンデンサの実装

部品表の部品番号 R5～R26 をはんだ付けします。
極性はありません。

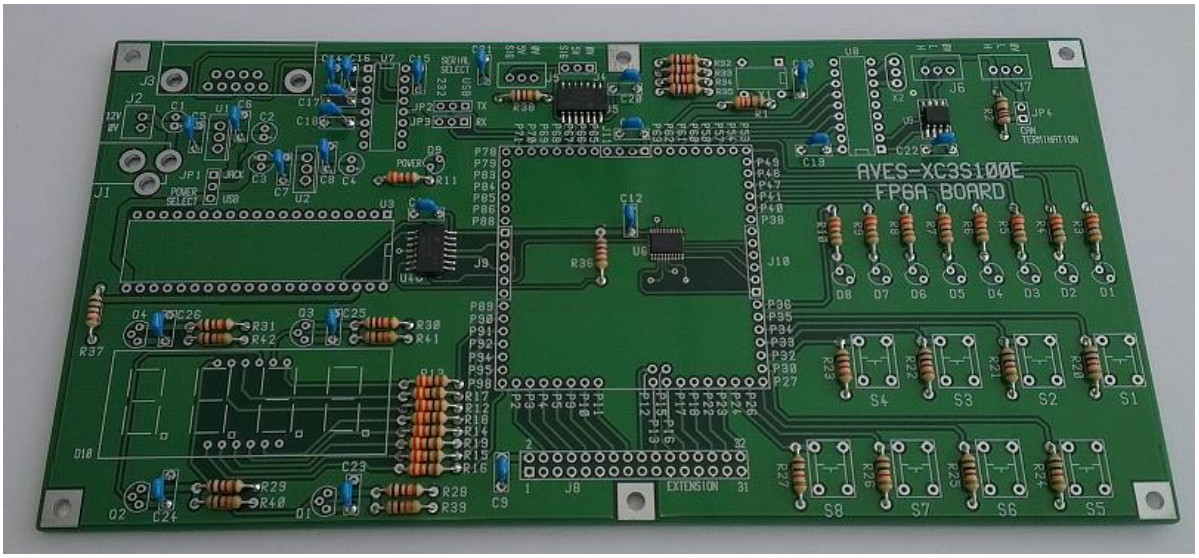


図 2.1.6　積層セラミックコンデンサ実装後のプリント基板

2.1.7　電界コンデンサの実装

部品表の部品番号 C1～C4 をはんだ付けします。　極性があります。
図 x に示すように電解コンデンサの側面に白い線が入っている方をシルク印刷の**＋が無い方**に実装します。

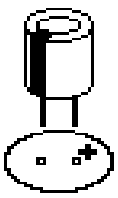


図 2.1.7-1　電界コンデンサの実装

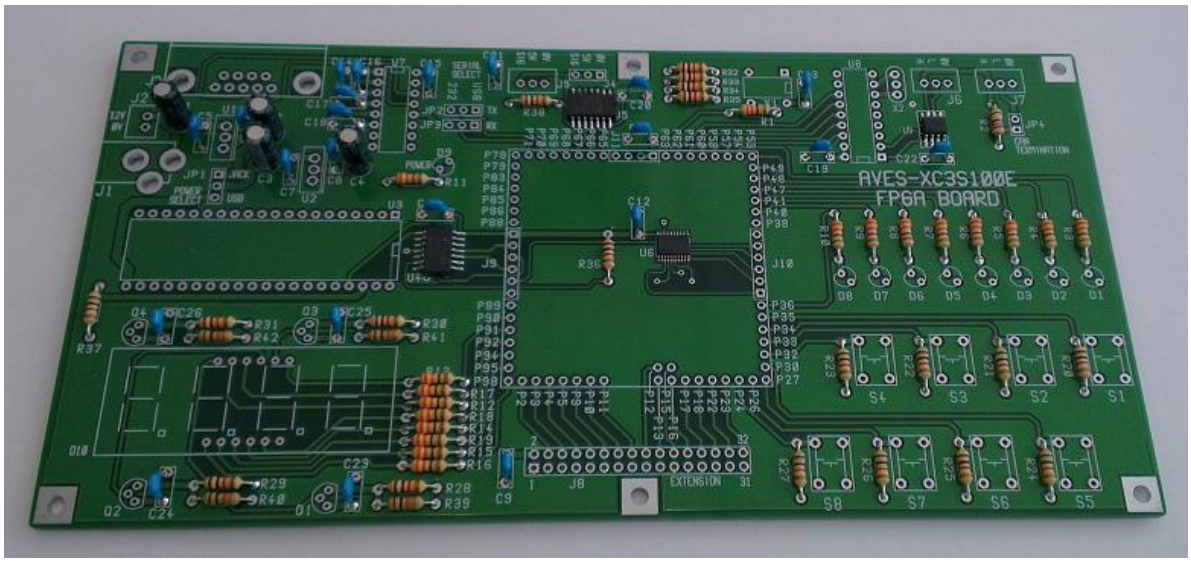


図 2.1.7-2　電界コンデンサ実装後のプリント基板

2.1.8　スイッチの実装

SW1～SW8 のタクトスイッチをはんだ付けします。タクトスイッチは、縦と横でピンの足の幅が違います。プリント基板の向きに合わせてはんだ付けしてください。

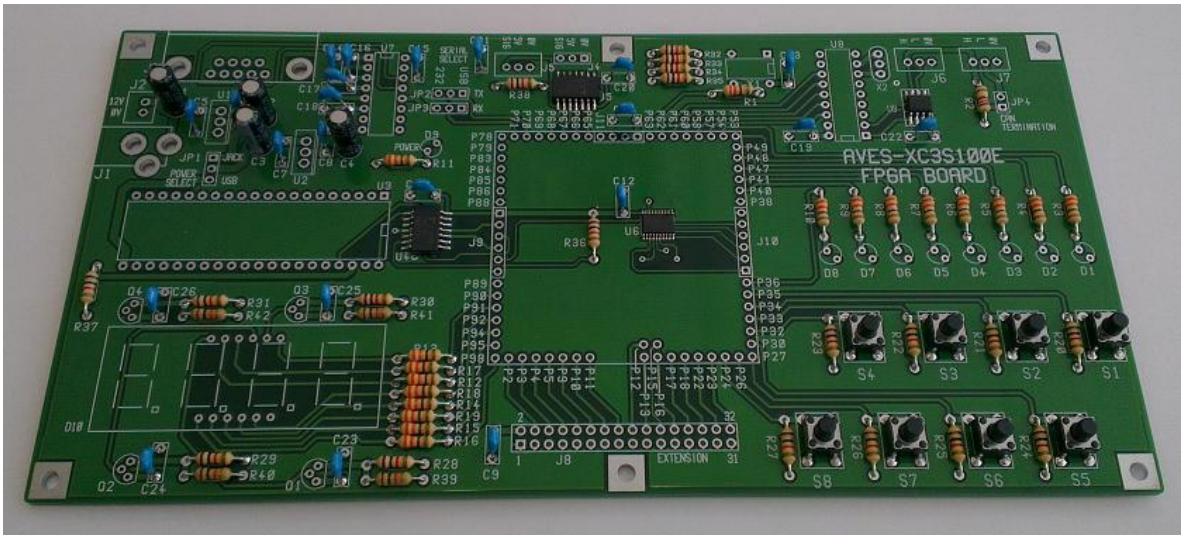


図 2.1.8　タクトスイッチ実装後のプリント基板

2.1.9　トランジスタの実装

Q1～Q4 のトランジスタをはんだ付けします。

図 x の様に、トランジスタの平らな面がシルク印刷と同じ向きになるように実装します。

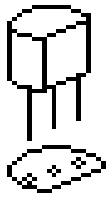


図 2.1.9-1　トランジスタの実装

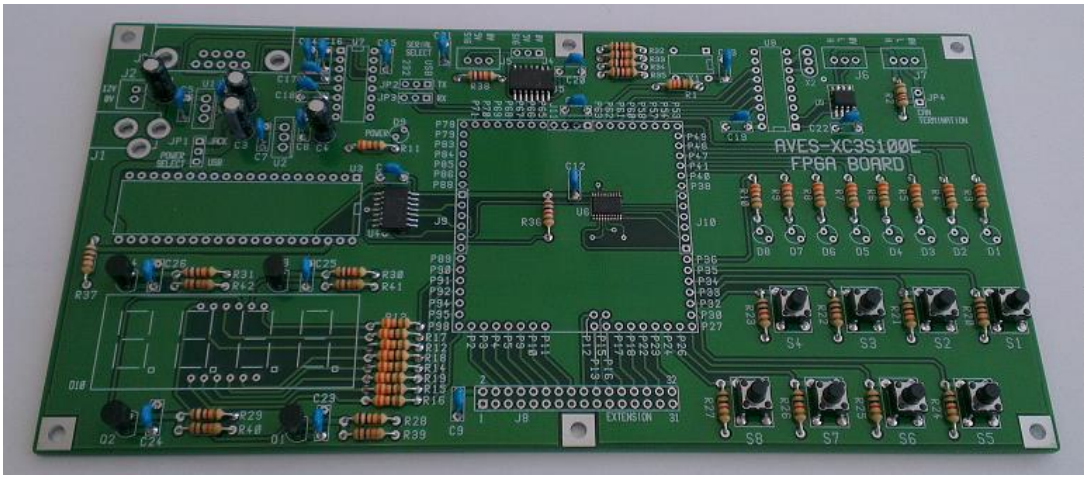


図 2.1.9-2　トランジスタ実装後のプリント基板

2.1.10　発光ダイオードの実装

D1～D9 の発光ダイオードを実装します。

図 x の様に、シルク印刷と発光ダイオードの平らな面がが合うように実装します。

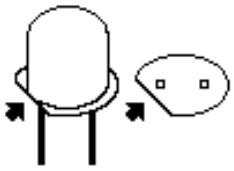


図 x　発光ダイオードの実装

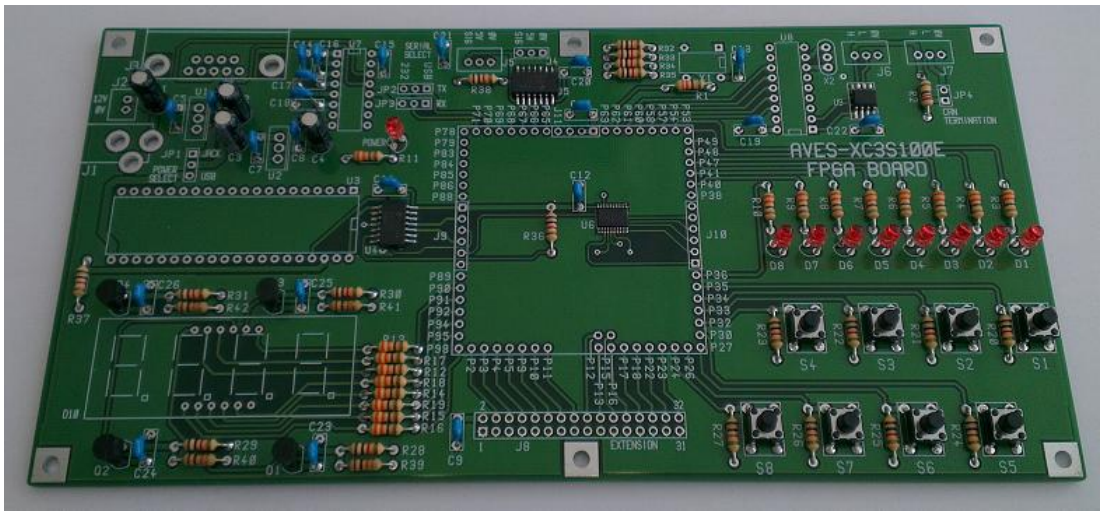


図 2.1.10　発光ダイオード実装後のプリント基板

2.1.11　セラミック発信子・オシレータ・各種 IC の実装

JP1～4 のジャンパピンをはんだ付けします。事前準備で割り折ったピンヘッダとピンフレームを使用します。

X2 のセラミック発信子を実装します。極性はありません。

X1 のオシレータと U8 と U9 の IC を実装します。基板のシルク印刷の向きにしたがって実装してください。

IC に切込みの無い物は表面に●など印がある側を合わせます。

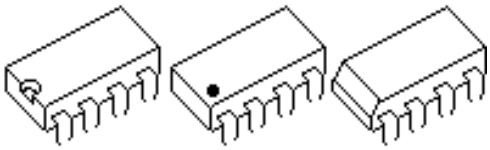


図 2.1.11-1　オシレータ・IC の実装

図 2.1.11-2　セラミック発信子・オシレータ・各種 IC 実装後のプリント基板

2.1.12　コネクタの実装

J1～J7 の各種コネクタをはんだ付けします。
J2・J5～J7 のコネクタは、基板のシルク印刷の四角に収まるように実装します。
（逆向きに実装すると、シルク印刷の四角からはみ出します）

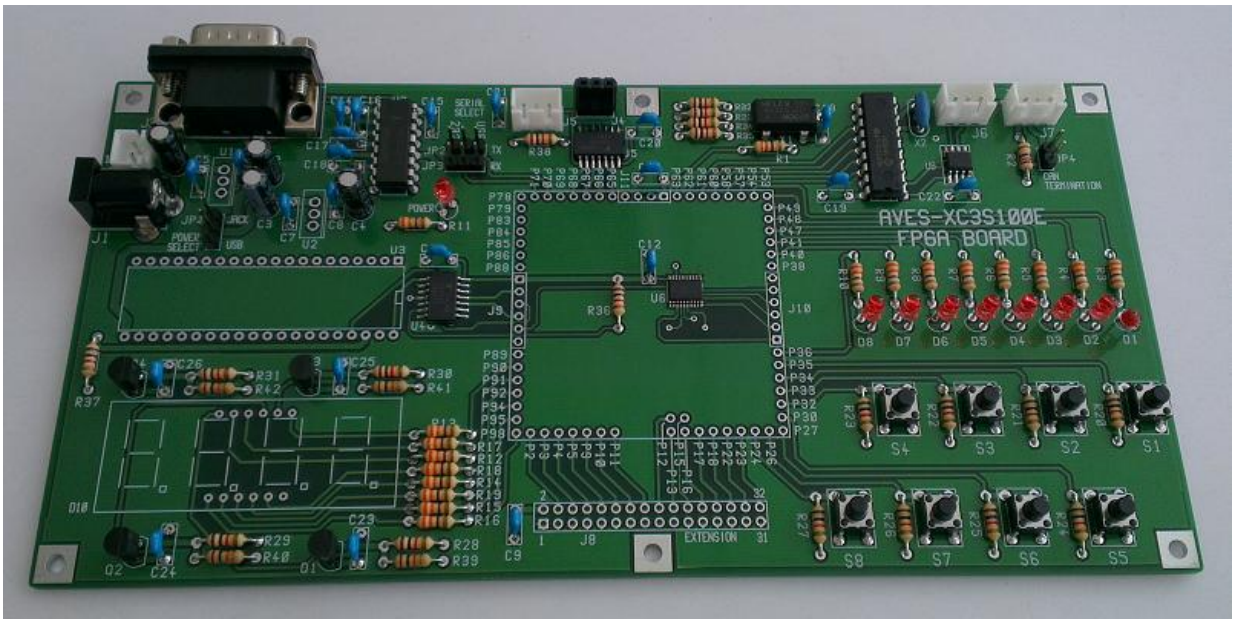


図 2.1.12　各種コネクタ実装後のプリント基板

2.1.13　7 セグメント LED の実装

D10 の 7 セグメント LED をはんだ付けします。
図 x の様に、7 セグメント LED の DP と基板のシルク印刷の DP マークを合わせて実装します。

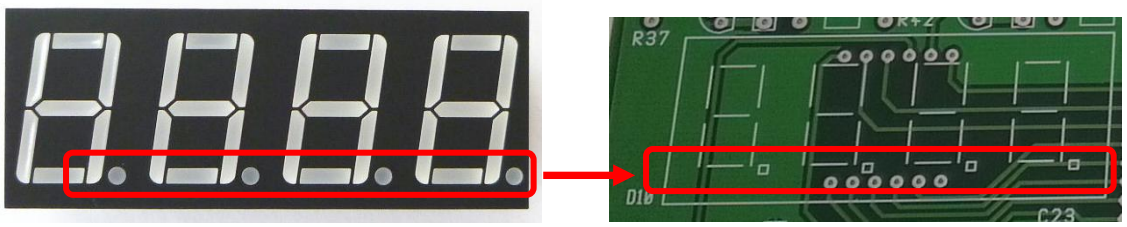


図 2.1.13-1　7 セグメント LED の実装

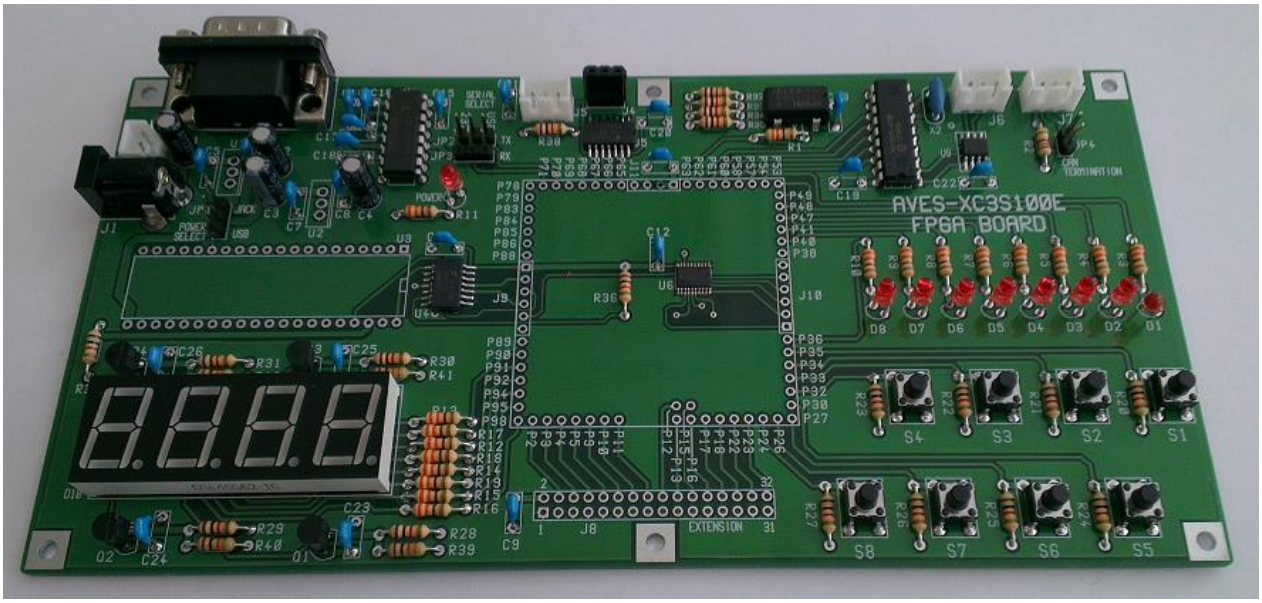


図 2.1.13-2　7 セグメント LED 実装後のプリント基板

2.1.14　USB モジュールとレギュレータの実装

U3 の USB モジュールをはんだ付けします。USB コネクタが基板の外を向くように実装します。

U1 と U2 のレギュレータを実装します。基板のシルク印刷の線があるほうにレギュレータの背中がくるように

実装します。

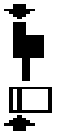


図 2.1.14-1　レギュレータの実装

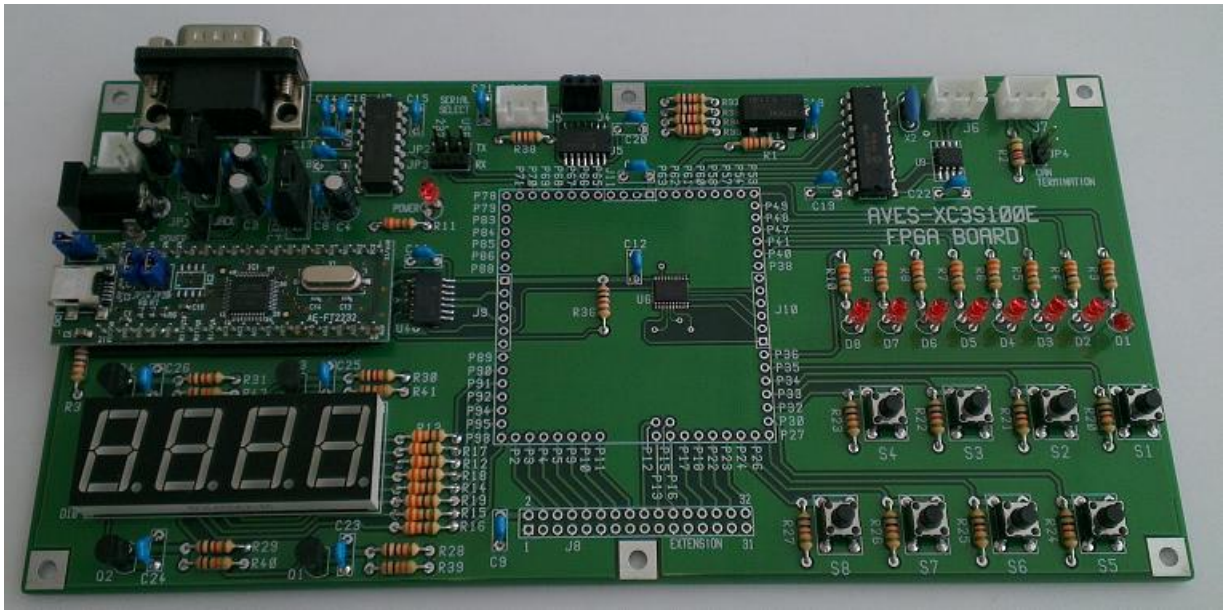


図 2.1.14-2　USB モジュールとレギュレータ実装後のプリント基板

2.1.15　FPGA 基板へのモード選択用ピンヘッダの実装

FPGA 基板の図 x に示す位置に、ピンヘッダを実装します。ピンヘッダは、事前準備で割り折ったピンヘッダ
を使用します。FPGA と同じ面に実装します。実装する面に注意してください。

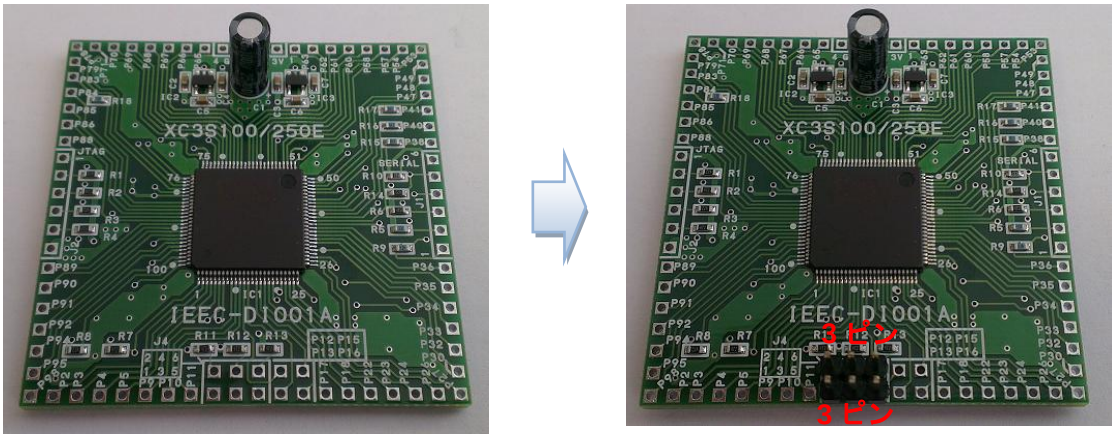


図 2.1.15　FPGA 基板へのモード選択用ピンヘッダの実装

2.1.16　FPGA 基板連結用コネクタの実装

FPGA 基板連結用のピンフレームをはんだ付けします。事前準備で割り折ったピンヘッダとピンフレームを
使用します。

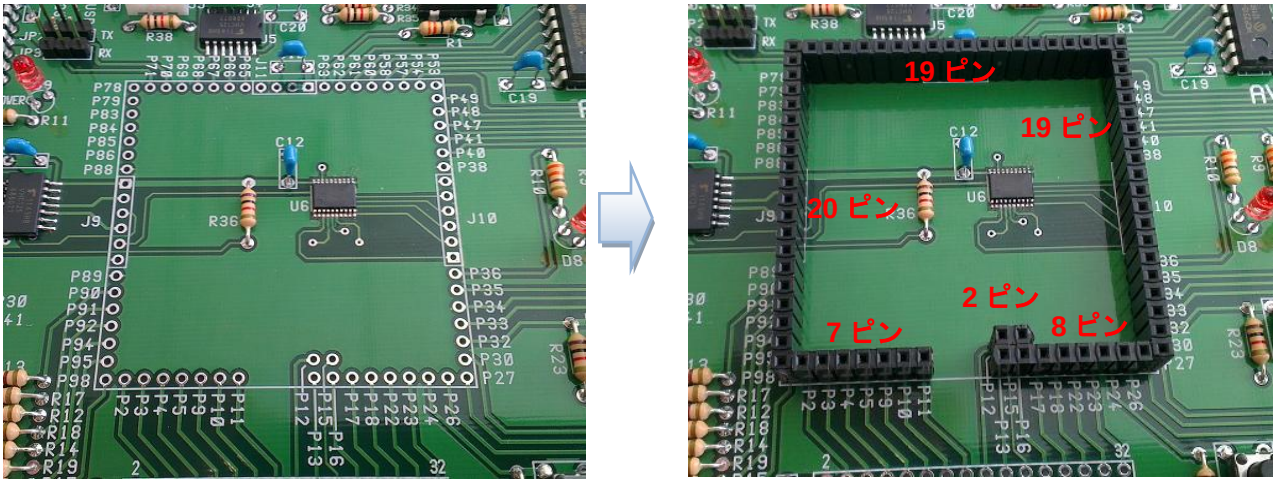


図 2.1.16-1　FPGA 基板連結用ピンフレームの実装

FPGA 基板側にピンフレームを実装します。

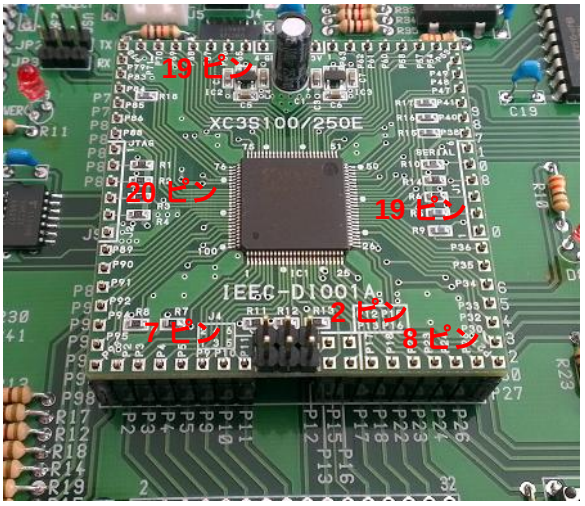


図 2.1.16-2　FPGA 基板へのピンヘッダ実装

完成

はんだ付けは、終了です。

ゴム足を基板裏に貼り付け、Dsub コネクタをねじとナットで固定しください。

超音波センサを、J4 に差し込んでください。

以上で完成です。

ジャンパピンの設定を行うまで電源は入れないでください。

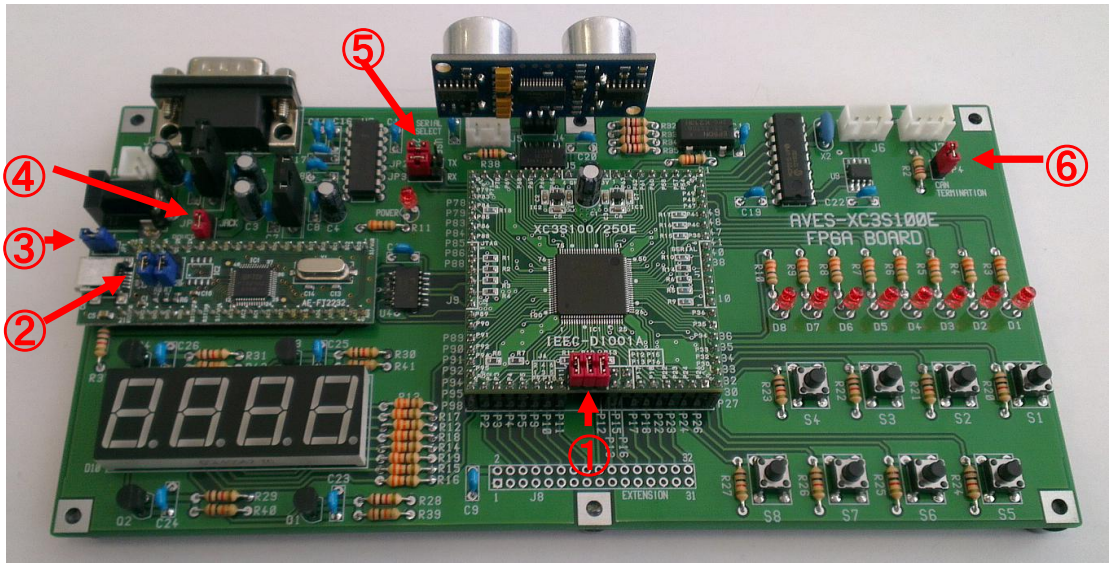


図 2.1.16-3　完成した FPGA 学習ボードとジャンパピン

2.1.17　ジャンパピンの設定

① FPGA 動作モード設定（3 箇所）

すべてをショートした状態に設定してください。

② USB モジュール IO 電圧設定（2 箇所）

すべてをオープンにした状態にしてください。

③ USB 給電 ON/OFF 設定

ショートした状態にしてください。

④ 電源切り替え

FPAG 学習ボードの電源を USB 給電もしくは、DC ジャックいずれかに選択します。

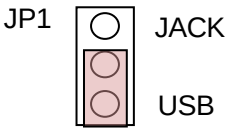


図 2.1.17-1　USB 給電の場合

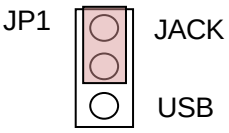


図 2.1.17-2　DC ジャック給電の場合

⑤ シリアル通信ポート切り替え

シリアルポートを USB 経由の仮想 COM ポートもしくは、Dsub9 ピンの RS-232C いずれかに選択します。

選択は、送信と受信を個別に設定できます。

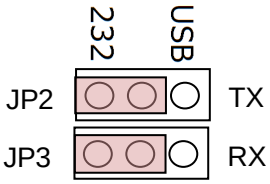


図 2.1.17-3　Dsub9 ピン RS-232C の場合

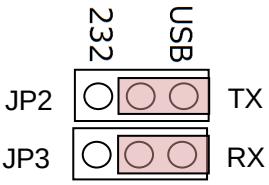


図 2.1.17-4　USB 仮想 COM ポートの場合

⑥ CAN 終端抵抗 ON/OFF 設定

CAN 通信を使用する際に、この基板が CAN バスの終端にあたる場合にショートします。

CAN 通信を使用しない場合は、オープン/ショートどちらでも構いません。

3 FPGA 開発環境

3.1 ISE(Integrated Software Environment)

ISE は Xilinx（ザイリンクス）社が提供している PLD の統合開発環境です。

HDL による記述と、ＵＣＦ（ユーザー制約ファイル）により HDL 合成およびシミュレーション、インプリメンテーション、デバイスへのフィット、JTAG プログラミングの機能を備えた、ダウンロード可能なソリューションです。

Xilinx 社では、この ISE を簡単に試すことが出来るように、無償の ISE WebPACK を提供しています。詳しい入手方法およびインストールは Xilinx 社のホームページを参照してください。

http://japan.xilinx.com/

3.2 ハードウェア記述言語

3.2.1 VerilogHDL

Ｖｅｒｉｌｏｇ（ヴェリログ）は、デジタル回路の設計用の論理シミュレータで、そこで使用するハードウェア記述言語でもあります。両者を区別する場合、言語の方を「ＶｅｒｉｌｏｇＨＤＬ」と呼びます。

言語の開発にあたっては、ソフトウェア開発者にも受け入れられるようにプログラム言語のＣ言語やＰａｓｃａｌの要素を取り入れたものとなっていてＩＥＥＥ 1364-2005 として標準化されています。

3.2.2 VHDL

ＶＨＤＬは、デジタル回路ハードウェア記述言語の一種で、ＥＤＡ（Electronic Design Automation）分野における標準の一つです。主として論理回路の設計に、特にＦＰＧＡやＡＳＩＣなどの設計で使われる。規格が存在し、ＩＥＥＥ 1076-2008 です。

4 VerilogHDL による信号処理

4.1 VerilogHDL 書式

VerilogHDL の基本書式は以下のようになります。

```
module モジュール名（入出力ポートリスト）;
        input  入力ポートリスト;
        output 出力ポートリスト;

        wire 結線リスト;
        reg レジスタ定義;

        assign 論理式;                    ・・・組合せ論理回路記述

        always @(動作トリガ定義)
        begin
                ff_q <= x;                ・・・順序論理回路記述
        end

endmodule
```

4.1.1　組合せ論理回路

組合せ論理回路は、その時の入力信号だけで出力が決まるデジタル回路です。

主な回路としては、基本ゲート回路があり、その応用としてはスイッチによる入力回路や、７セグメントＬＥＤの表示制御出力のようなエンコーダ、デコーダ回路があります。

回路の設計手順としては

状態図　→　真理値表　→　論理式　→　簡略式　→　論理回路

という手順で設計を進めていきます。

簡単な例として、多数決回路を考えてみましょう。

多数決回路では、命題に対し賛成と反対の各意見を表明し、その過半数で可決、または否決を決める物です。

説明を簡単にするため、最少人数でこれをデジタル回路で考えると、３人で命題を考え、それぞれ賛成と反対の意志をデジタル信号の「０」と「１」で、それぞれＡさんＢさんＣさんとして表明します。

そこで、この３人の意志を表す信号を「０」と「１」で考えた時、その組合せは以下の８通りになります。これを表で現した物を真理値表と言います。

そこで、この３人の意志を表す信号を「０」と「１」で考えた時、その組合せは以下の８通りになります。これを表で現した物を真理値表と言います。

	入　力			出　力	最　小　項
i	A	B	C	X	mi
0	0	0	0	0	m0＝／A・／B・／C
1	0	0	1	0	m1＝／A・／B・　C
2	0	1	0	0	m2＝／A・　B・／C
3	0	1	1	1	m3＝／A・　B・　C
4	1	0	0	0	m4＝　A・／B・／C
5	1	0	1	1	m5＝　A・／B・　C
6	1	1	0	1	m6＝　A・　B・／C
7	1	1	1	1	m7＝　A・　B・　C

表 4.1.1　真理値表

表４．１．１で、出力が「０」または「１」のいずれかの場合について考えます。

通常は正論理と呼ばれる方法で、出力が「１」の状態に着目して考える方法で、この時の出力が「１」の状態を**アクティブな状態**、「０」を**インアクティブな状態**と言います。

アクティブな状態とは、「電灯では電灯が点灯している」「スイッチではスイッチが押されている」
等の状態をいいます。
このアクティブな状態に着目して、論理式を導く方法を**加法標準形**と言います。

また「０」の場合を考える事を負論理と言い、インアクティブな状態に着目した考え方になります。アクティブとは、能動的な状態で、例えばラアンプなら点灯している時、プッシュプスイッチならばスイッチを押してＯＮにした、この時、**加法標準形**でこの回路の論理式を求めると

Ｘ＝ｍ３＋ｍ５＋ｍ６＋ｍ７

となります。

この式を＊ブール代数＊や、＊カルノー図＊により＊簡略化＊を行います。

簡略化は、論理式中の冗長な部分を無くし、無駄な回路を省く効果があり、それによりディスクリート基板では実装面積を小さくし、ＦＰＧＡの様なロジックデバイスでも、その消費電力を押さえることが期待出来るので＊回路設計上においては必須＊の課程です。

現在は、論理合成時にソフトウェアで自動的に行われるのが当たり前になっていますが、その手法を知っておくことは、回路の誤りを速やかに発見する上で＊大変重要なポイント＊となります。

簡略化の例

前の加法標準形により導かれた論理式を、ブール代数を用いて簡略すると

Ｘ＝ｍ３＋ｍ５＋ｍ６＋ｍ７を正確に記述すると

Ｘ＝（／Ａ・Ｂ・Ｃ）＋（Ａ・／Ｂ・Ｃ）＋（Ａ・Ｂ・／Ｃ）＋（Ａ・Ｂ・Ｃ）

となります。（／は否定：NOT を表す）

Ｘ＝（（／Ａ・Ｂ・Ｃ）＋（Ａ・Ｂ・Ｃ））
　＋（（Ａ・／Ｂ・Ｃ）＋（Ａ・Ｂ・Ｃ））
　＋（（Ａ・Ｂ・／Ｃ）＋（Ａ・Ｂ・Ｃ））

Ｘ＝（（／Ａ・Ｂ・Ｃ）＋（Ａ・Ｂ・Ｃ））・・・Ｂ・Ｃ・（／Ａ＋Ａ）＝Ｂ・Ｃ
　＋（（Ａ・／Ｂ・Ｃ）＋（Ａ・Ｂ・Ｃ））・・・Ａ・Ｃ・（／Ｂ＋Ｂ）＝Ａ・Ｃ
　＋（（Ａ・Ｂ・／Ｃ）＋（Ａ・Ｂ・Ｃ））・・・Ａ・Ｂ・（／Ｃ＋Ｃ）＝Ａ・Ｂ

Ｘ＝（Ｂ・Ｃ）＋（Ａ・Ｃ）＋（Ａ・Ｂ）

ここで求まった簡略式を VerilogHDL の書式にならってで記述すると以下の様になります。
ファイル名：ｍａｊｏｒｉｔｙ．ｖ

この多数決の論理回路を１つのモジュールとして、モジュール名、入出力ポートを定義します。

```
module majority(IN_A, IN_B, IN_C, OUT_X);
    input IN_A, IN_B, IN_C;
    output OUT_X;
```

次に、かっこ内の論理積の信号を引き出す線として

```
    wire    AND_BC, AND_AC, AND_AB;
```

続いて、定義したワイヤーに論理回路を割り付けると

```
    assign AND_BC = IN_B & IN_C;
    assign AND_AC = IN_A & IN_C;
    assign AND_AB = IN_A & IN_B;
```

この３つの論理積の出力を出力Ｘに論理和で接続します

```
    assign OUT_X = AND_BC | AND_AC | AND_AB;
```

最後にモジュールの宣言を完結する記述

```
endmodule
```

を書いて終了です。

※ＨＤＬリストのコメントは、Ｃ言語と同じ／＊～＊／または／／です。

ｕｃｆファイル
制約ファイルと呼ばれ、主にピンアサインの記述を行います。
ｕｃｆファイルの基本書式は以下のようになります。

NET　“ポート名”　ＬＯＣ　＝　“ピン番号”；

今回の多数決回路を例として記述すると以下のようになります。
ファイル名：ｍａｊｏｒｉｔｙ．ｕｃｆ

```
# INPUT PIN
NET "IN_A"          LOC = "P35";        # SW1
NET "IN_B"          LOC = "P34";        # SW2
NET "IN_C"          LOC = "P33";        # SW3

# OUTPUT PIN
NET "OUT_X"         LOC = "P54";        # LED1
```

※ｕｃｆリストのコメントは、＃でそれ以降行末までがコメントになります。

4.1.2　順序論理回路

順序論理回路は、その時の入力信号と出力の状態により次の出力が決まるデジタル回路です。
この回路では、現在の**信号状態**を**記憶する機能**が必要となります。
主な回路としては、フリップ・フロップ回路があり、その応用としてはカウンタ回路や、レジスタ回路
　　があります。

回路の設計手順としては
　　状態遷移図　→　状態遷移表　→　論理式　→　簡略式　→　論理回路
という手順で設計を進めていきます。

簡単な例として、電子サイコロを考えてみましょう。
電子サイコロ回路では、サイコロの目を一定時間間隔で１～６まで変化させていきます。
説明を簡単にするため、サイコロの目を１～６まで一定時間毎に順番に変化すると仮定します。
（本物のサイコロは、その目がランダムに出ますが、今回は順番に出る物と仮定します）

そこでまず、サイコロの目の変化を状態遷移図として記述します。

図４．１．２は、サイコロの目の移り変わりを図で示したものです。これを状態遷移図と言います。

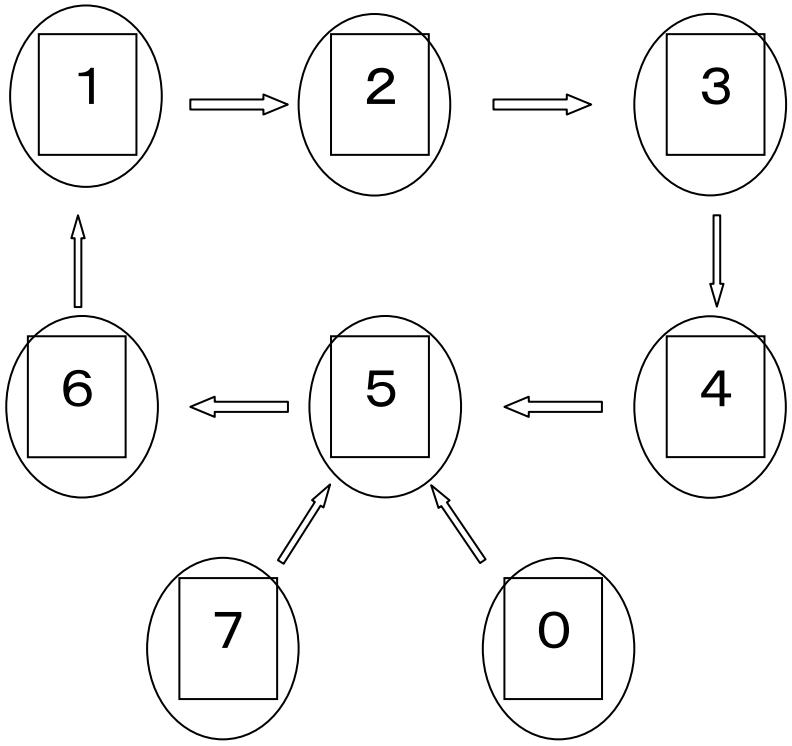


図 4.1.2　状態遷移図

サイコロの目の値は最大６で、この値を表すために３ビットが必要です。
これを D フリップ・フロップで実現するには３個のフリップ・フロップが必要で、各フリップ・フロップの入出力をＤ２、Ｄ１、Ｄ０、Ｑ２、Ｑ１、Ｑ０と割り振ると以下の状態遷移図が出来ます。

計数値	現在の状態			次の状態			制御入力		
i	Ｑ２	Ｑ１	Ｑ０	Ｑ２	Ｑ１	Ｑ０	Ｄ２	Ｄ１	Ｄ０
0	0	0	0	1	0	1	1	0	1
1	0	0	1	0	1	0	0	1	0
2	0	1	0	0	1	1	0	1	1
3	0	1	1	1	0	0	1	0	0
4	1	0	0	1	0	1	1	0	1
5	1	0	1	1	1	0	1	1	0
6	1	1	0	0	0	1	0	0	1
7	1	1	1	1	0	1	1	0	1

表 4.1.2　状態遷移表

表４．１．２は図４．１．２の状態遷移図を基に遷移状態を表にした状態遷移表です。次に、この状態遷移表から論理式を導きいます。

論理式の導き方は組み合わせ論理回路の場合と同じ手法で行います。
真理値表との対応は以下の通りです。

真理値表　　状態遷移表
　入力　　現在の状態
　出力　　制御入力（D フリップ・フロップの場合は次の状態と同じになる）

まず、真理値表の出力に相当する制御入力の「１」に着目（加法標準形）し、Ｄ２～Ｄ０までをそれぞれ

Ｄ２　＝　ｍ０＋ｍ３＋ｍ４＋ｍ５＋ｍ７
Ｄ１　＝　ｍ１＋ｍ２＋ｍ５
Ｄ０　＝　ｍ０＋ｍ２＋ｍ４＋ｍ６＋ｍ７

と導き出されます。

この時の論理式は、組み合わせ論理で行った加法標準系で、入力を現在の状態、出力を制御入力として導きます。

これは、現在の状態から次の状態へ遷移させるためにＤフリップ・フロップのＤ入力を現在の状態から導くことになります。
例えば、現在サイコロが１であれば次は２にしたいので
　Ｄ２には「０」
　Ｄ１には「１」
　Ｄ０には「０」
を与えることになり、ここではＤ１の論理式に現在の状態の最小項ｍ１（００１）が論理和として入力されることになります。

後は、組み合わせ論理式の時と同じくＤ２からＤ０の各論理式を簡略し回路を構成し、その結果をフリップ・フロップのＤ入力に与えれば良いのです。

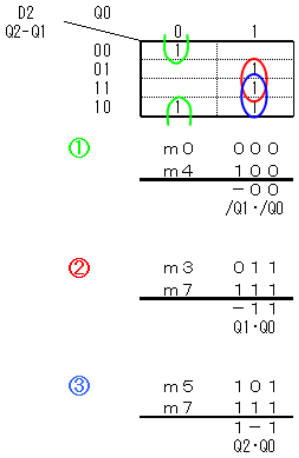
今回はカルノー図を用いて簡略します。

簡略後の格式は

Ｄ２　＝　ｍ０＋ｍ３＋ｍ４＋ｍ５＋ｍ７

Ｄ２　＝　①＋②＋③

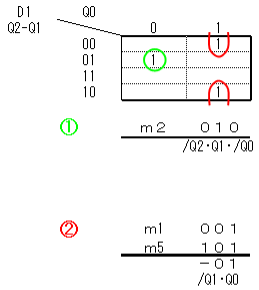
Ｄ２　＝　（／Ｑ１・／Ｑ０）
　　＋　（　Ｑ１・　Ｑ０）
　　＋　（　Ｑ２・　Ｑ０）



Ｄ１　＝　ｍ１＋ｍ２＋ｍ５

Ｄ１　＝　①＋②

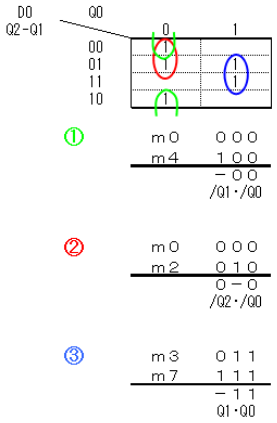
Ｄ１　＝　（／Ｑ２・　Ｑ１・／Ｑ０）
　　＋　（／Ｑ１・　Ｑ０）



Ｄ０　＝　ｍ０＋ｍ２＋ｍ４＋ｍ６＋ｍ７

Ｄ０　＝　①＋②＋③

Ｄ０　＝　（／Ｑ１・／Ｑ０）
　　＋　（／Ｑ２・／Ｑ０）
　　＋　（　Ｑ１・　Ｑ０）



となります。

4.1.3　ＩＳＥによる設計

ここでは、4.1.1 で設計した多数決回路を実際にＩＳＥで作成します。
今回使用する ISE のバージョンは８．２ｉです。

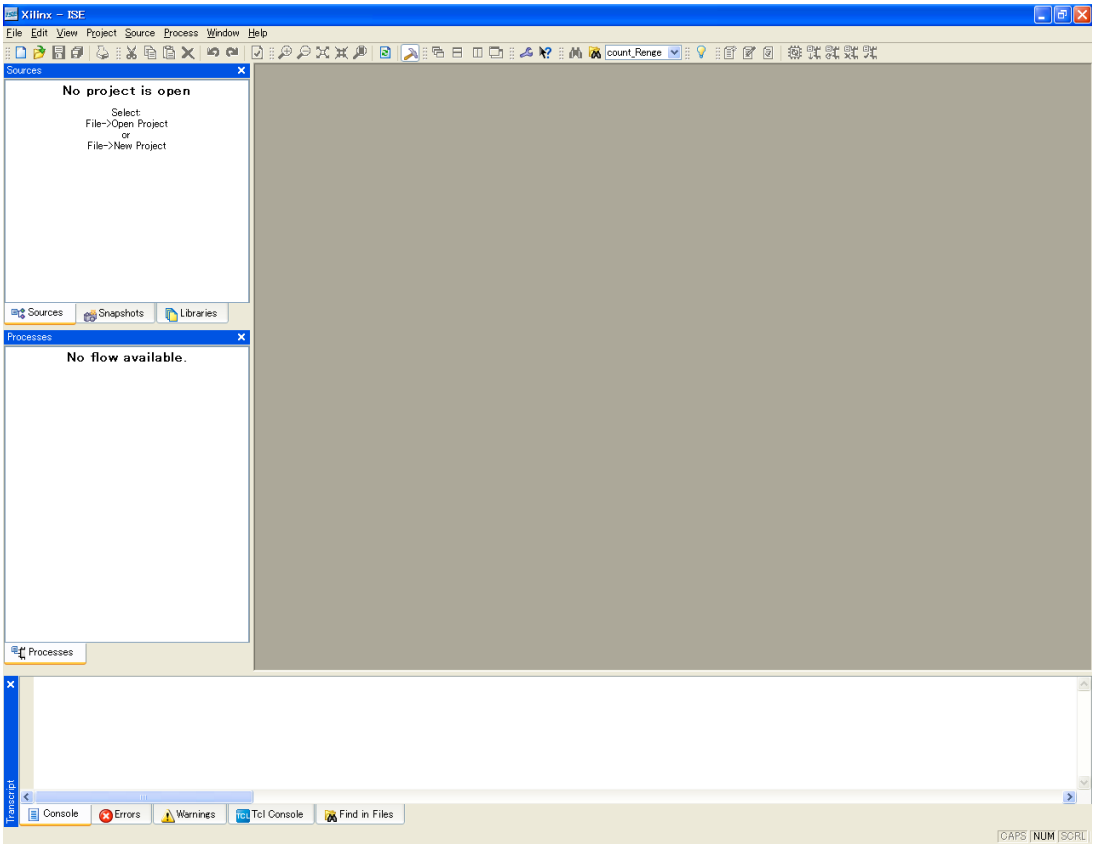
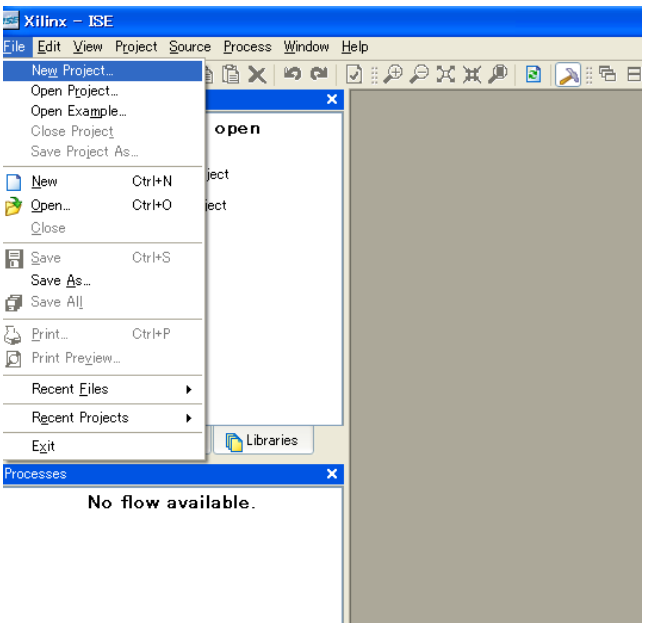


図 4.1.3-1

初めてＩＳＥを起動すると図のような画面になります。
もし、以前に何か設計した実績があれば、以前の設計を継承した画面になります。

① 新規プロジェクトの作成



左上の File タブから NewProject を選択

図 4.1.3-2

・プロジェクト名　格納場所　ソースタイプの選択

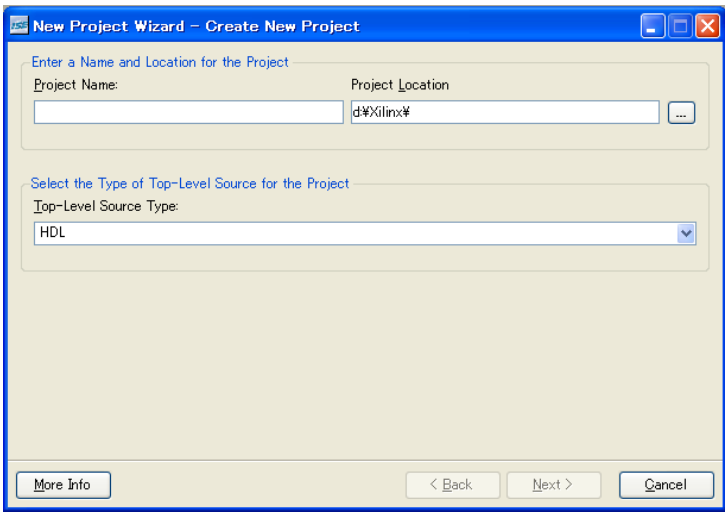


図 4.1.3-3

New Oroject Wizard

Project Name：
新規に作成するプロジェクト名
[majority]と入力

Project Location：
作成するプロジェクトの格納場所
各自の環境に合わせて場所を指定

Top-Level Source Type：
最上位階層のソースタイプ
[HDL]　デフォルト

・デバイスプロパティの選択

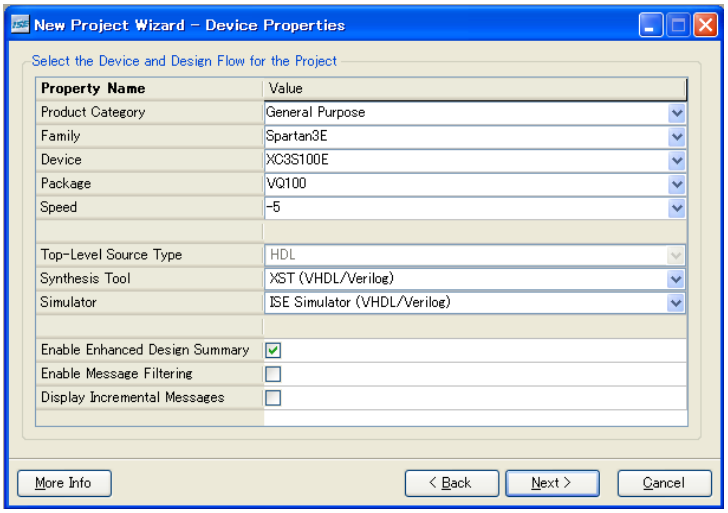


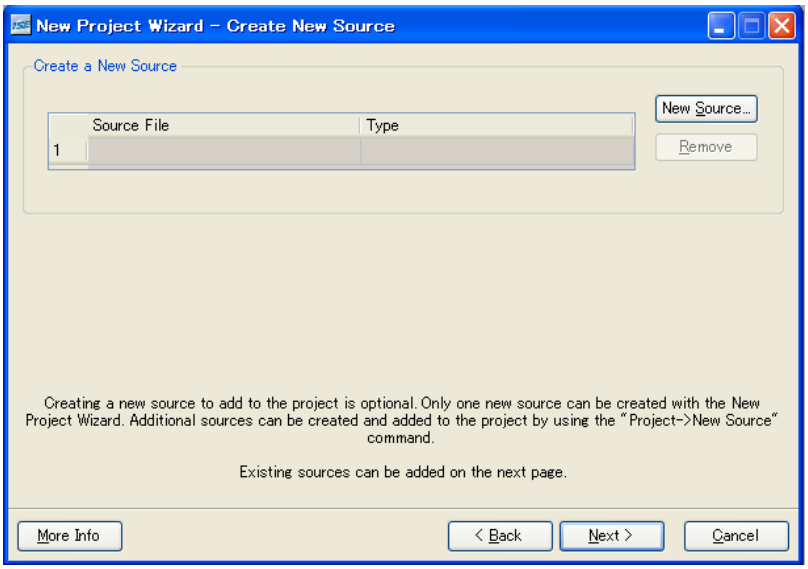
図 4.1.3-4

Family：
これから作成する FPGA の
ファミリーの選択
[Spartan3E]を選択

Device：
デバイス名の選択
[XC3S100E]を選択

Package：
パッケージの選択
[VQ100]を選択
　その他はデフォルトのままです。

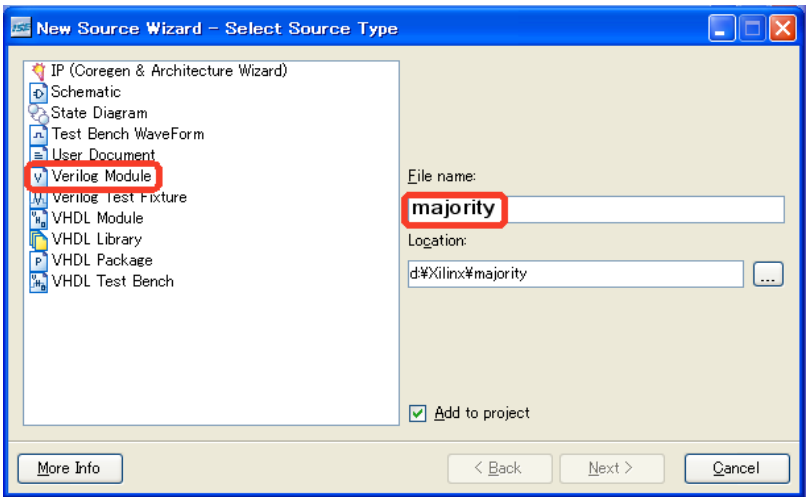
・ソースファイルの作成



New Sorce… :
ボタンをクリック

図 4.1.3-5

・ソースタイプ選択　ソース名入力

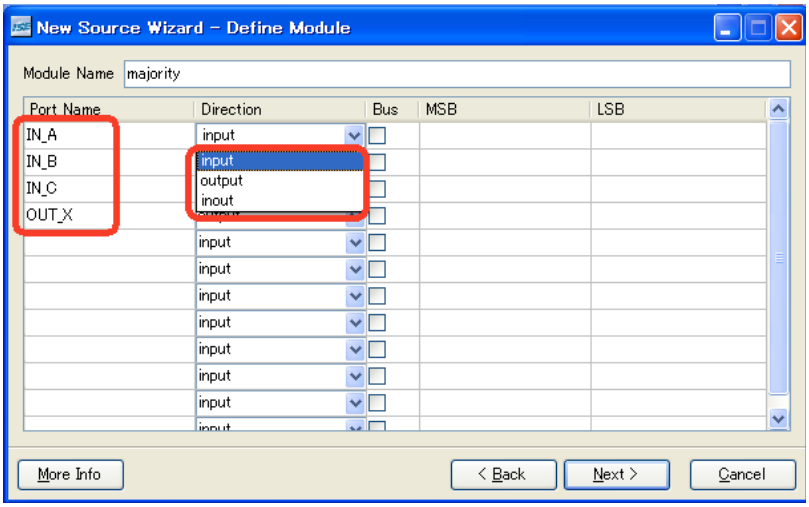


Select Sorce Type :
Verilog Module を選択

File name :
Majority と入力

図 4.1.3-6

・ポートリストの作成

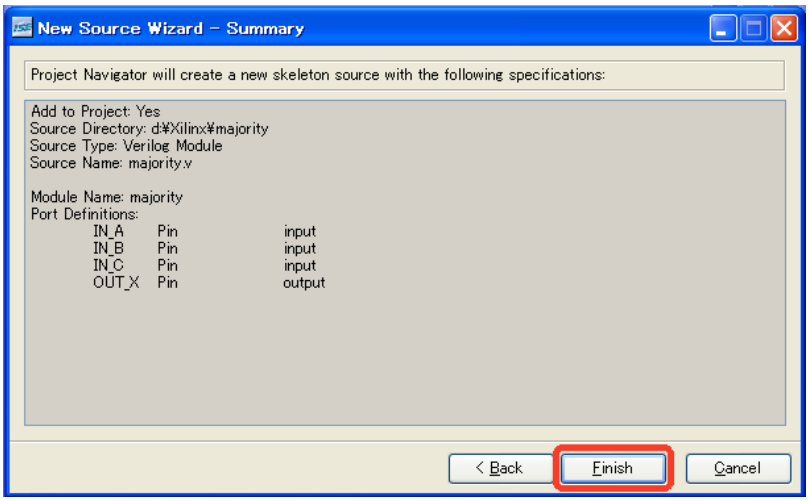


Port Name：
ポート名
IN_A
IN_B
IN_C
OUT_X
を入力

Direction：
入出力方向の選択
IN_A,IN_B,IN_C：input
OUT_X：output
を選択

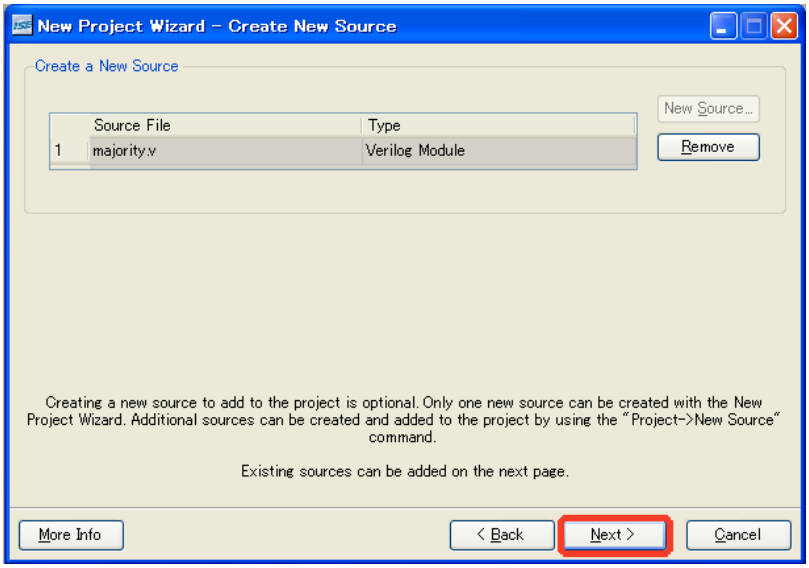
図 4.1.3-7

・ポートリストの作成



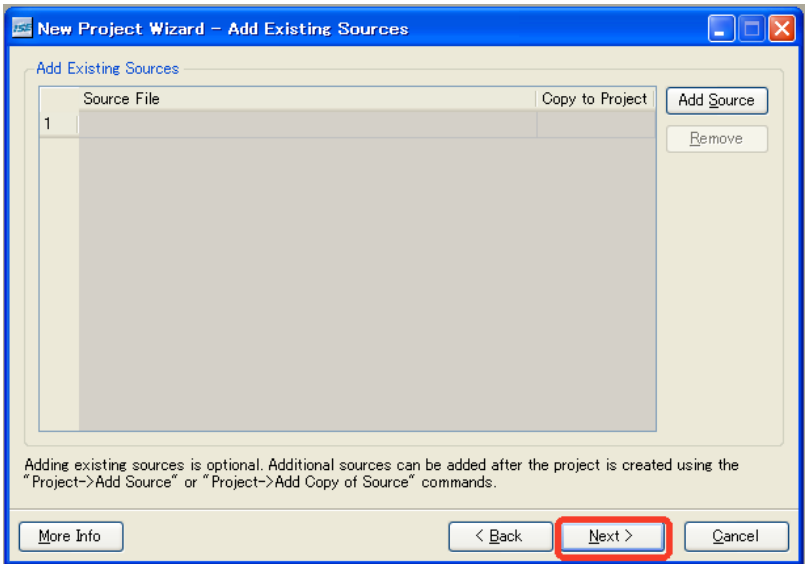
Finish：
クリック

図 4.1.3-8



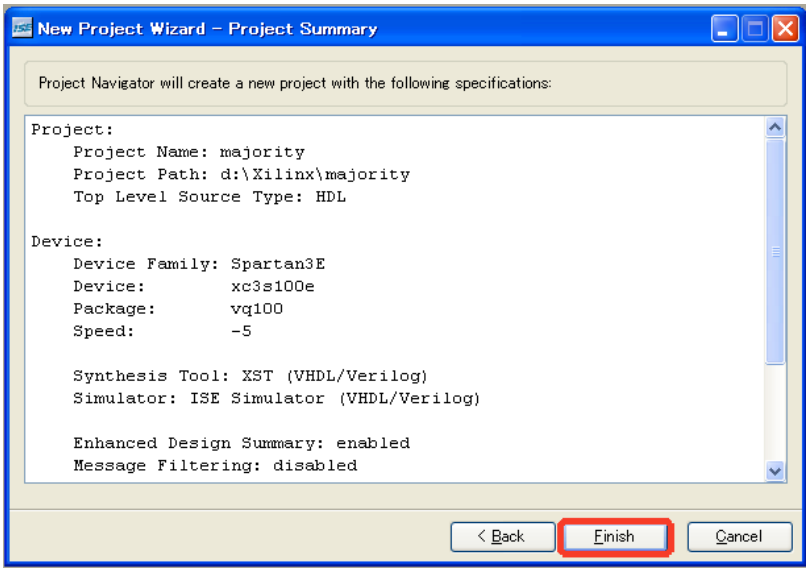
Next：
クリック

図 4.1.3-9



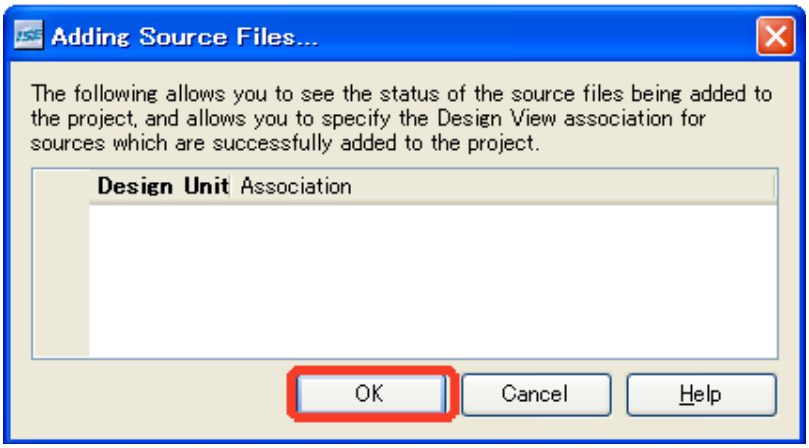
Next：
クリック

図 4.1.3-10



Finish：
クリック

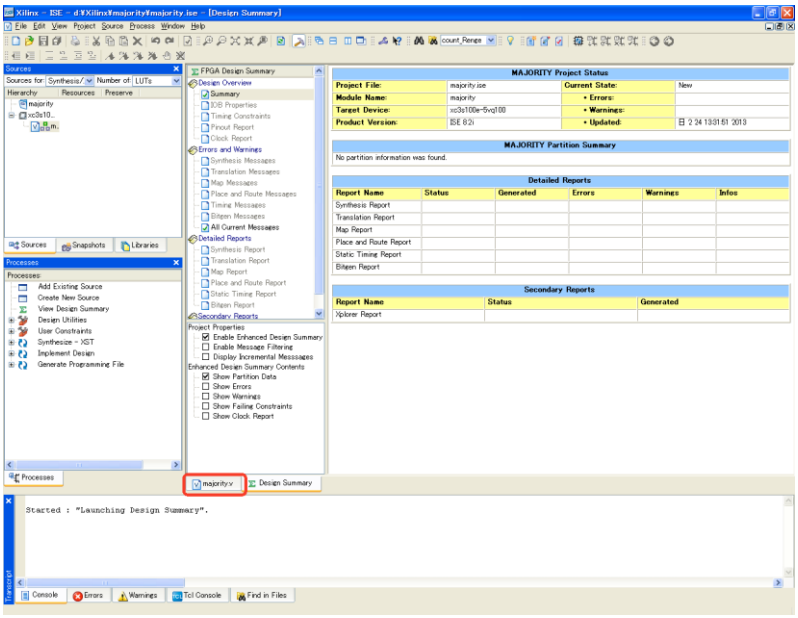
図 4.1.3-11



OK：
クリック

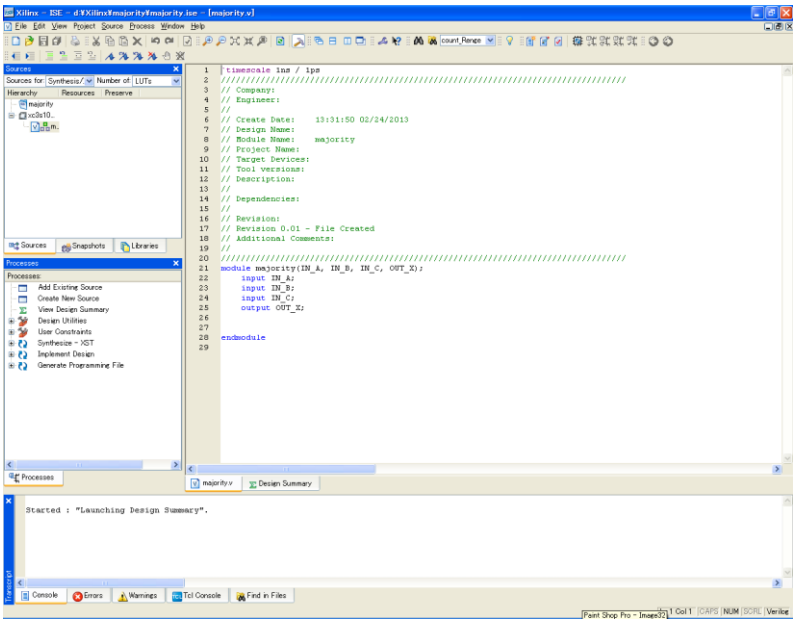
図 4.1.3-12

・最初のひな形が出来て新規プロジェクトになります



majority.v :
タブをクリック

図 4.1.3-13

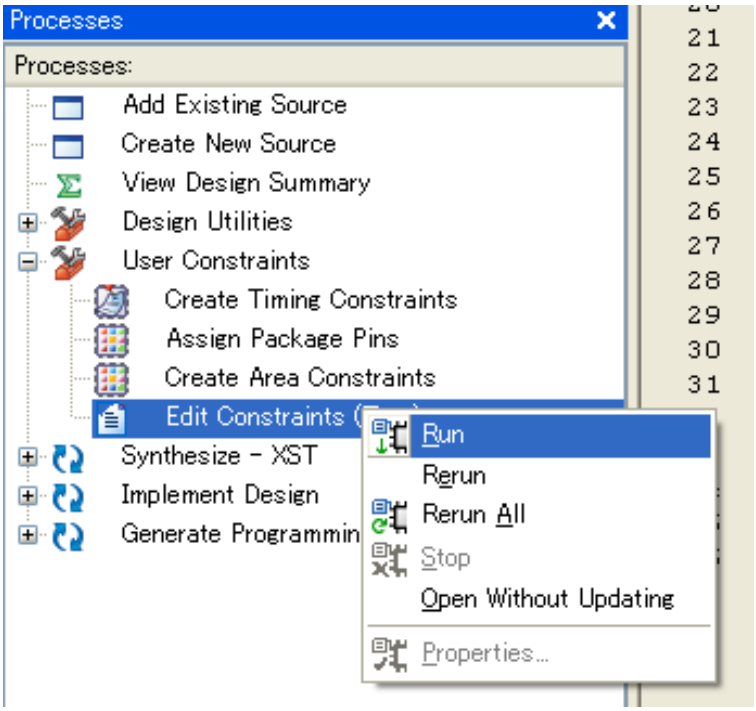


moduke majority のソースのひな形

この中に 4.1.1 で求めた論理式を基に、ソースコードを記述していきます。

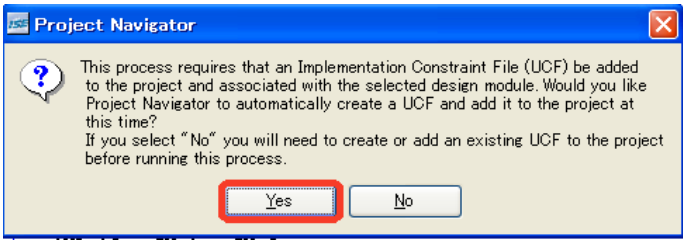
図 4.1.3-14

・制約ファイルの作成



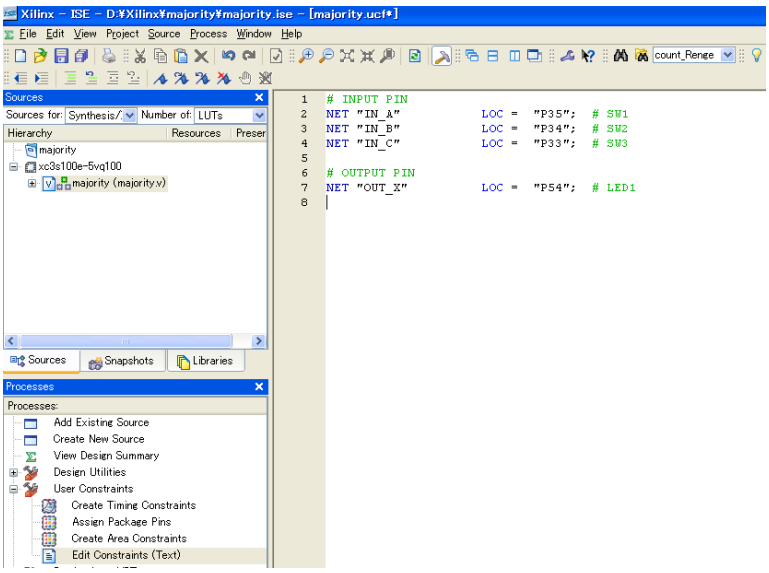
Processes ウィンドウの User Constraints → Edit Constraints(Text) → Run をクリック

図 4.1.3-15



確認ウィンドウで Yes をクリックすると、ucf ファイルの画面になります。

図 4.1.3-16



ここに、ピンアサイン情報を記述します。

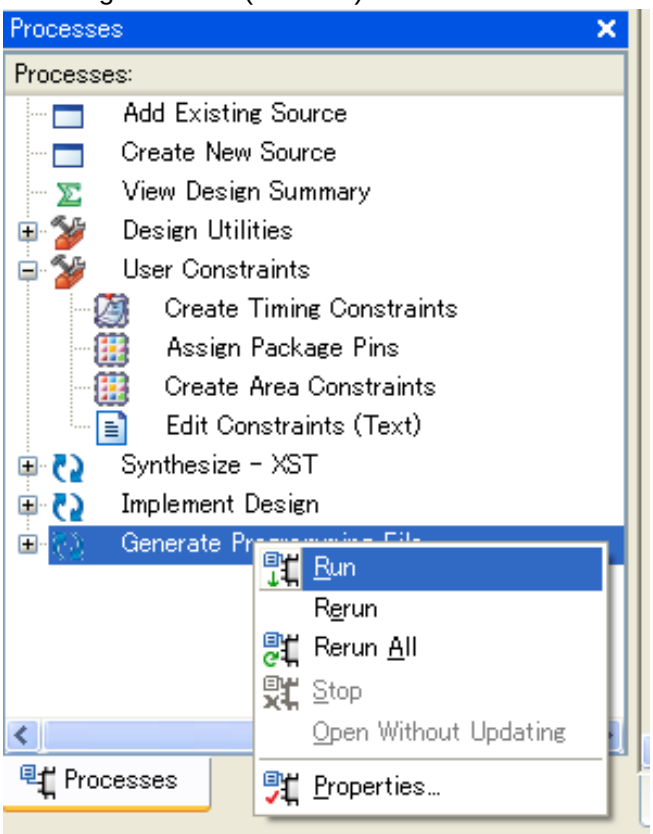
図 4.1.3-17

・ ISE による論理合成→実装配置→FPGA データ
ISE では、VerilogHDL ソースコードから、以下の処理を逐次的に自動実行されます。
◇ 論理合成（回路生成）
◇ 実装配置
◇ 書き込みデータ生成

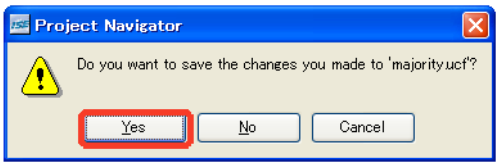
Synthesize –XST（XST (Xilinx® Synthesis Technology)）
Synthesize Report
RTL Schematic
Technology Schematic]
Check Syntax
Genetate Post-Synthesis Simulation Model

Implement Design
Translate
Map
Place & Route

Generate Programming File
Programming File Generation Report
Generate PROM, ACE or JTAG File
Configure Device(iMPACT)



Processes ウィンドウの
Generate Programinng File を右クリックして Run
を選ぶと一連の作業が始まります。



確認画面で Yes をクリックします。

図 4.1.3-18

・Xilinx Web Talk

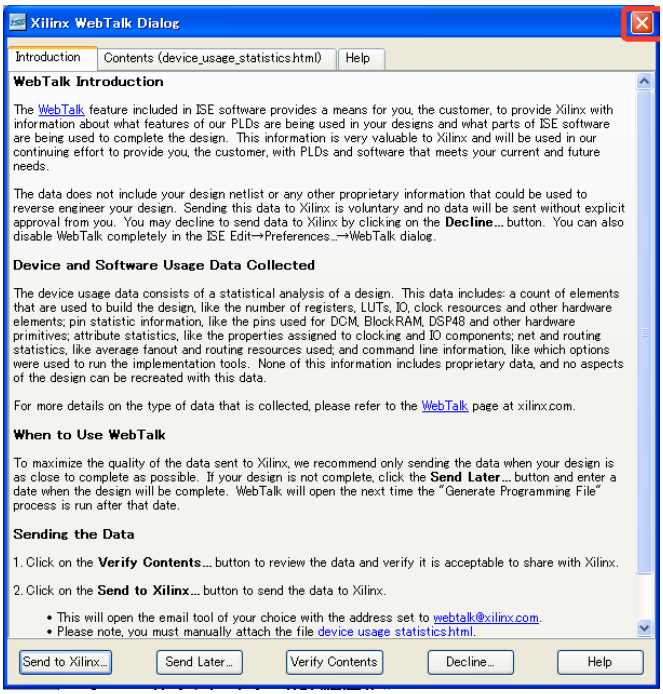


図 4.1.3-19

Xilinx Web Talk は ISE ユーザーがどのような設計で ISE を使っているかの情報を Xilinx 社に送り、製品開発へフィードバックするもので情報を送りたくない時はウィンドウの閉じるボタンで閉じても処理を続行できます。

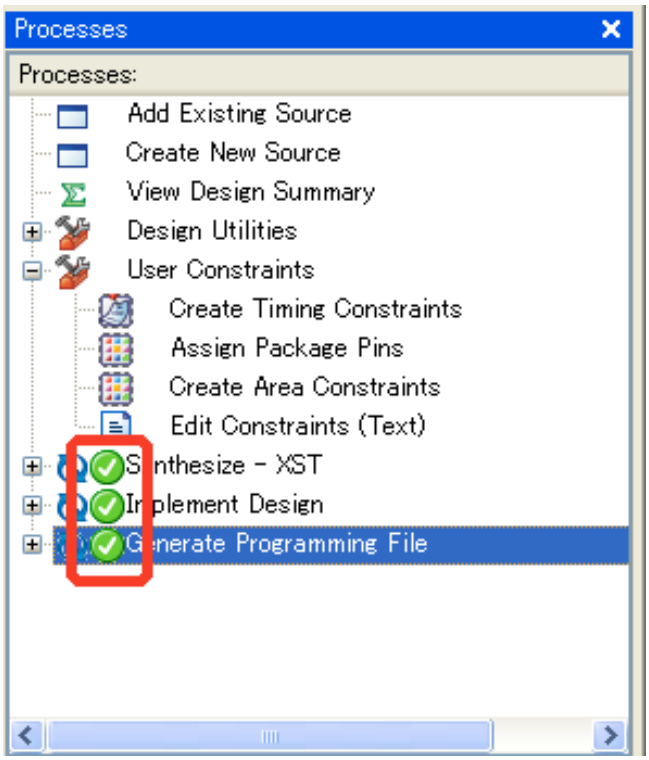


図 4.1.3-20

一連の処理が完了すると、各処理のところにチェックマークがつきます。この時、ワーニングが出てもファイルは生成されますので問題ありません。
ワーニングは、ISE のバージョンアップにより解決される物などがあります。

・FPGA ボードと接続し回路データを書き込みます

FPGA へのデータ書き込みでは、JTAG と呼ばれる規格のケーブルで行われますが、今回使用する AVES-XC3S100E ボードでは、その JTAG を USB 変換して接続しており、事前にドライバのインストールと、変換ソフト（cblsrv）の起動を行う必要があります。

ｗｅｂサイトで入手可能なソフトがありますので、そちらを使用しています。

以下が cblsrv を起動した画面です。（コマンド：cblsrv.exe -c amontec -p 10001）

図 4.1.3-21

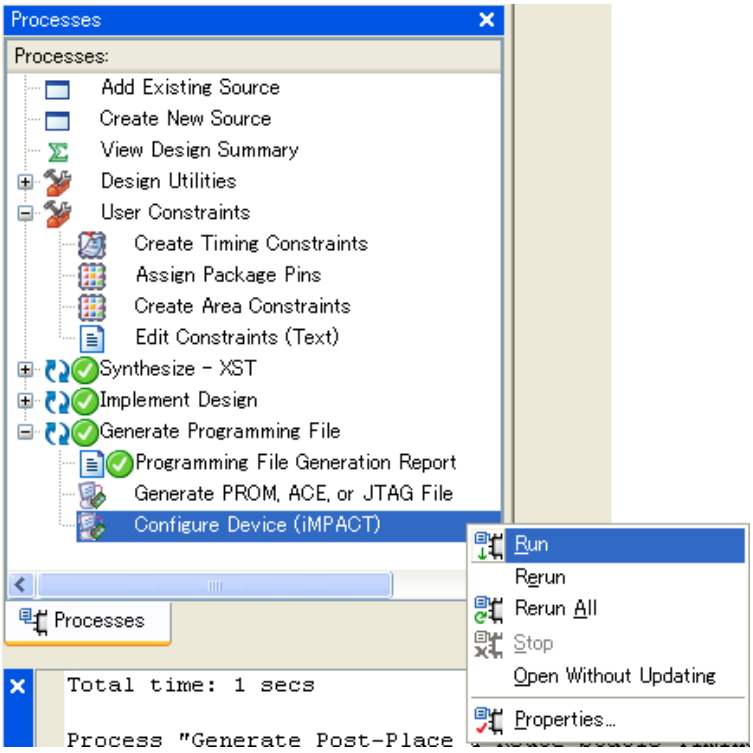
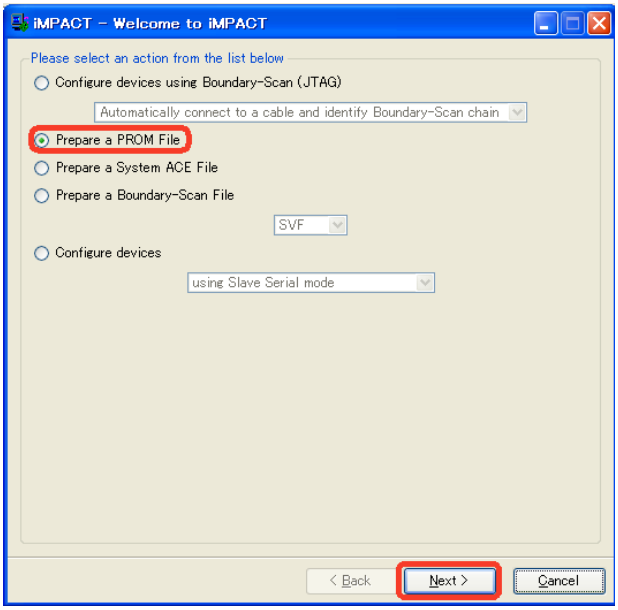


図 4.1.3-22

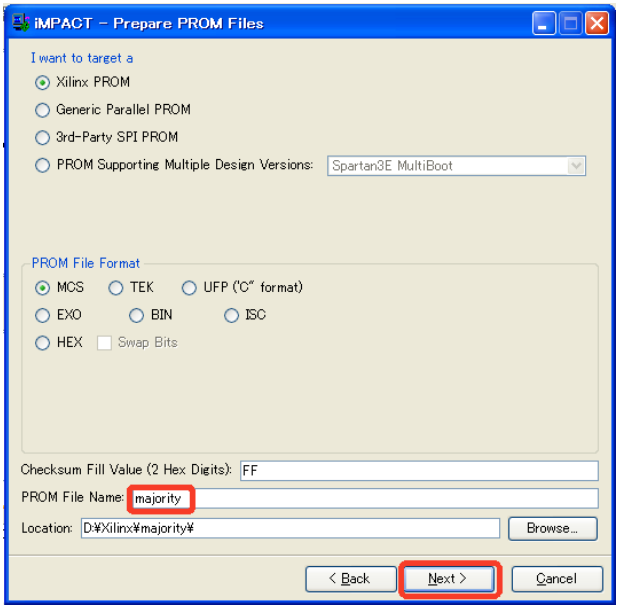
変換ソフトを起動後
Generate Programming File から
書き込みソフト iMPACT を起動します。

・ iMPACT の起動



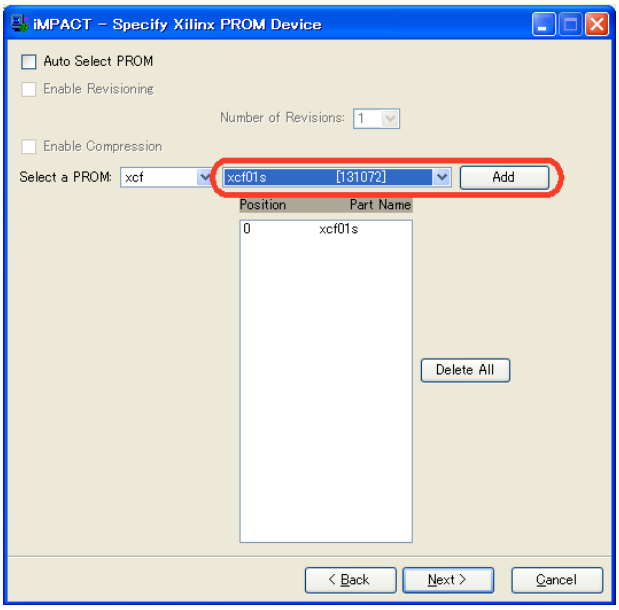
iMPACT を起動すると左のウインドウが表示されますので、Prepare a PROM File を選択して Next をクリックします。

図 4.1.3-23



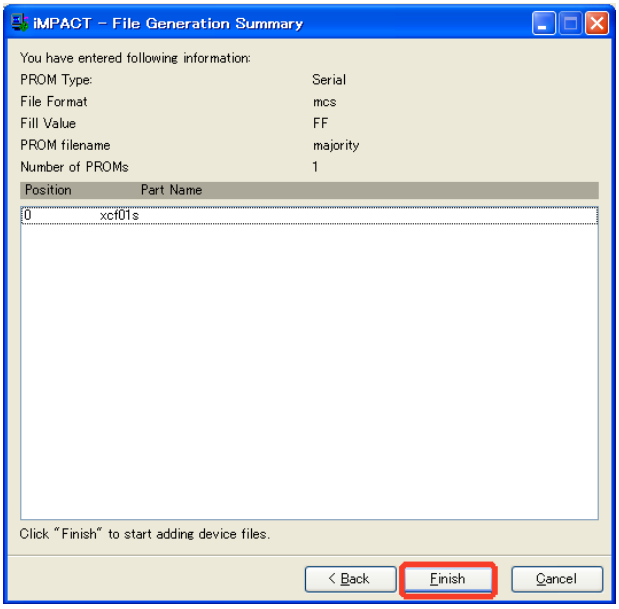
PROM File Name に majority と入力して Next をクリックします。

図 4.1.3-24



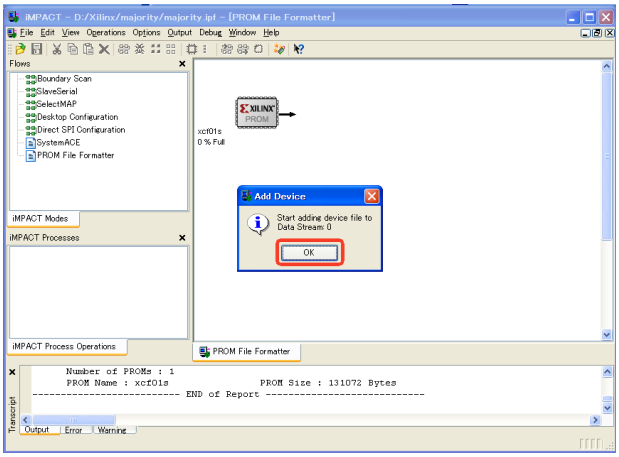
Select a PROM で xcf01s を選択して Add ボタンをクリックて Next ボタンをクリックします。

図 4.1.3-25



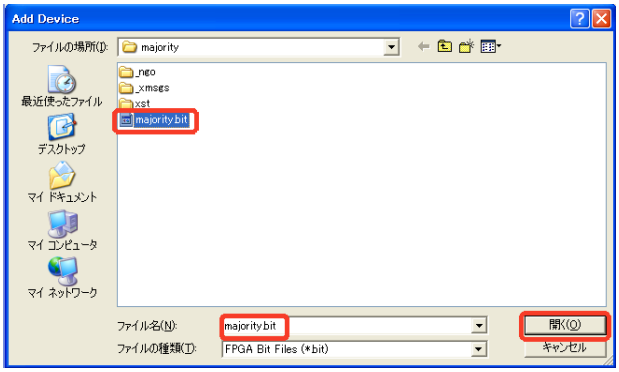
最後に Finish ボタンをクリックして確定します。

図 4.1.3-26



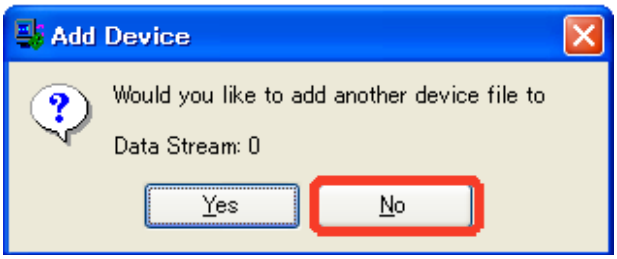
１回目は OK をクリックします。

図 4.1.3-27



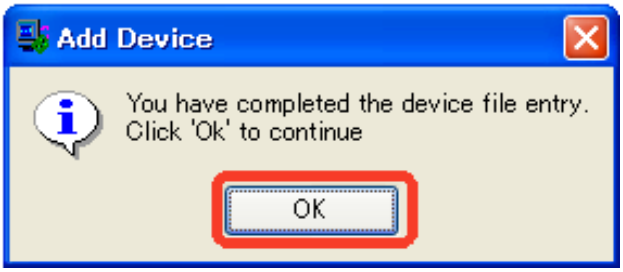
生成された majority.bit ファイルを選択し開くをクリックします。

図 4.1.3-28



Add Device の確認で No をクリックします。

図 4.1.3-29



OK をクリックしファイル選択を終了します。

図 4.1.3-30

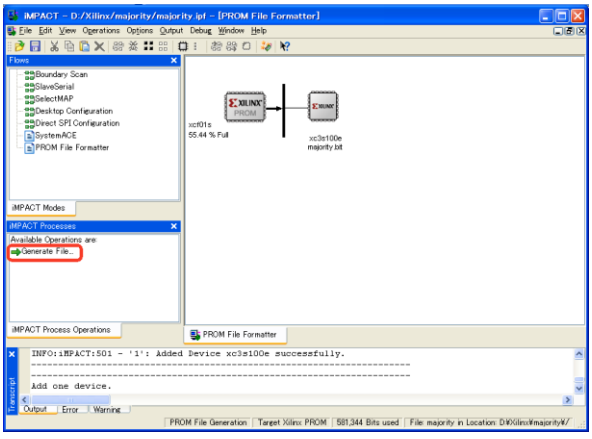


図 4.1.3-31

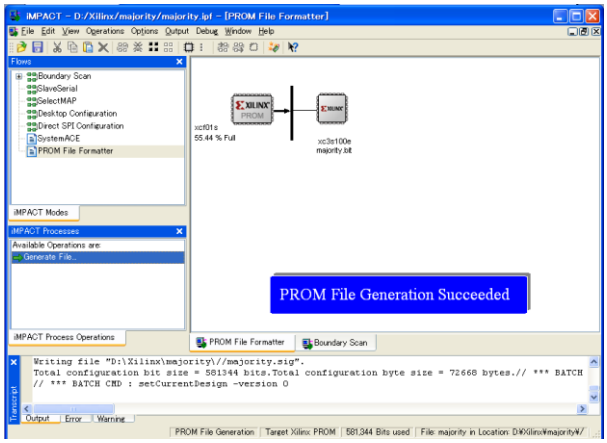
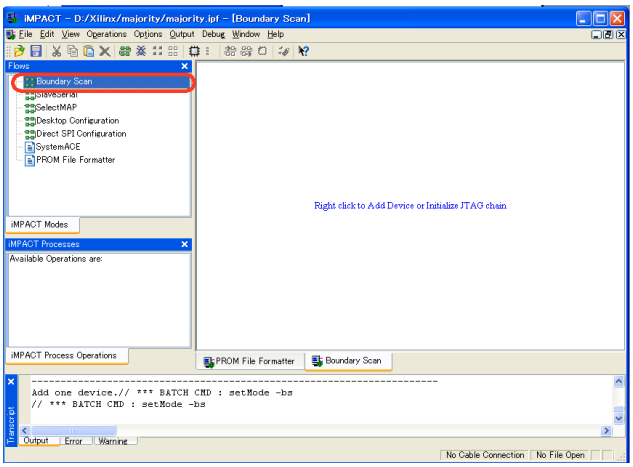


図 4.1.3-32

iMPACT Processes メニューから Generate File をダブルクリックして PROM への書き込みデータを生成します。



Bounday Scan をダブルクリックします。

図 4.1.3-33

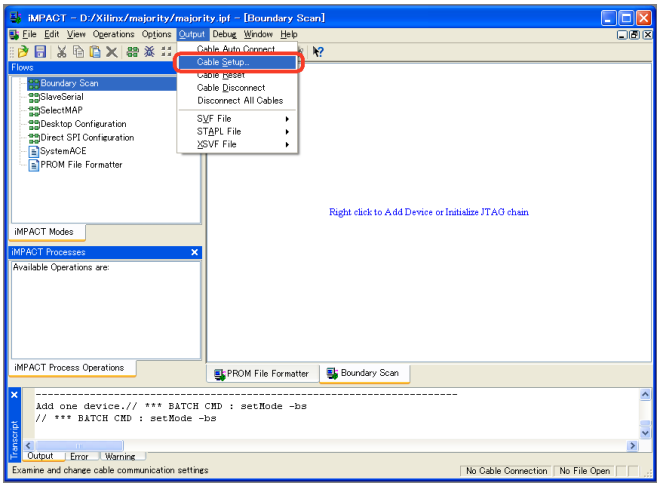


図 4.1.3-34

Output メニューから Cable Setup を選択します。

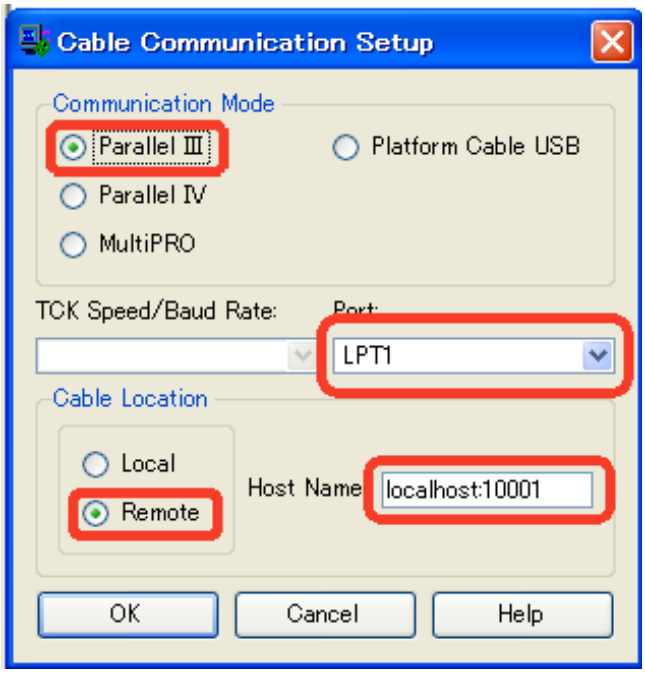


図 4.1.3-35

Communication Mode で ParallelⅢを選択し、Port で LPT1 を選択します。
続いて Cable Location で Remote を選択して Host Name に
Localhost:10001 を入力し最後に OK をクリックします。

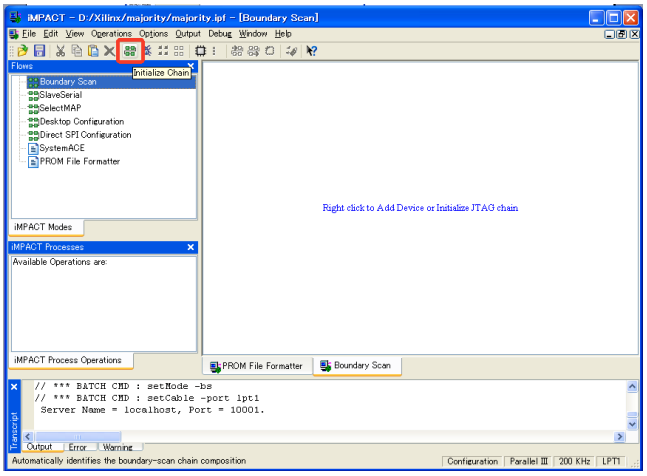
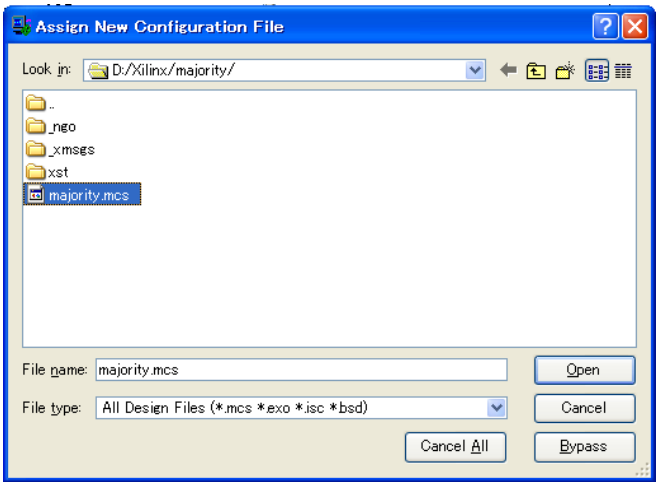


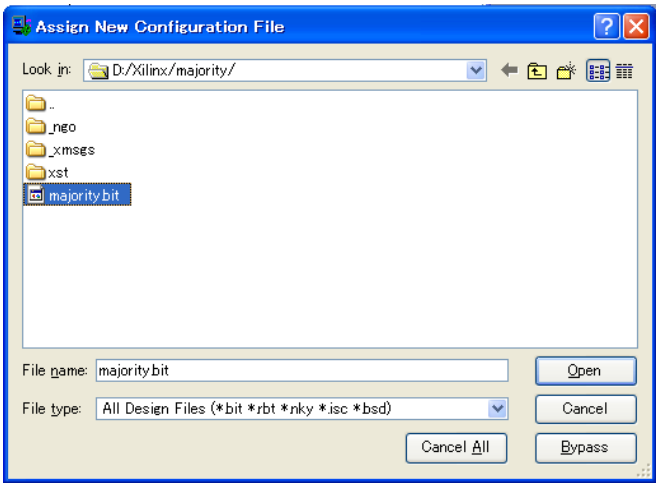
図 4.1.3-36

Intialize Chain をクリックし、接続デバイスをスキャンします。



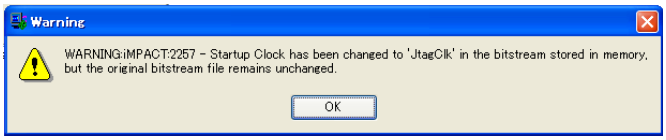
PROM File Formatter で生成された majority.mcf ファイルを選択します。

図 4.1.3-37



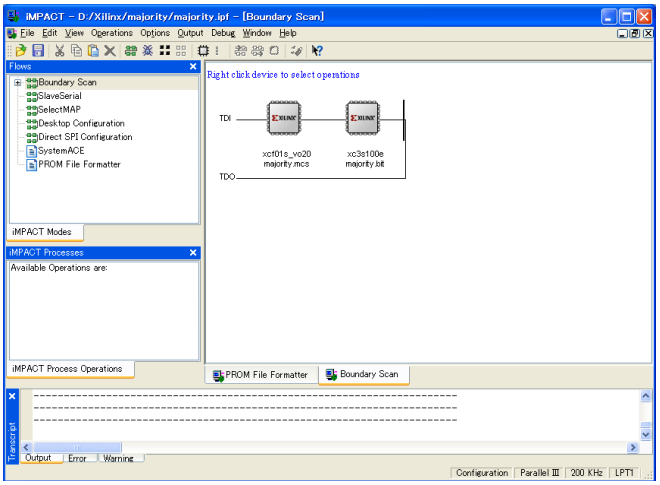
次に、FPGA 用回路データの majority.bin を選択します。

図 4.1.3-38



ここでワーニングがでても、そのまま OK をクリックします。

図 4.1.3-39



これで書き込み準備が出来ました。左の xcf01s_vo2 を選択し proguram を実行すると PROM に書き込まれます。

右側の xc3s100e を選択し program を実行すると FPGA へ直接書き込まれます。

図 4.1.3-40

※bit ファイルで直接 FPGA に書き込まれた場合は、電源が OFF になるとデータも消滅します。デバッグ時には FPGA へ書き込み動作チェックを行い確実に動作するようになったら PROM へ書き込みます。

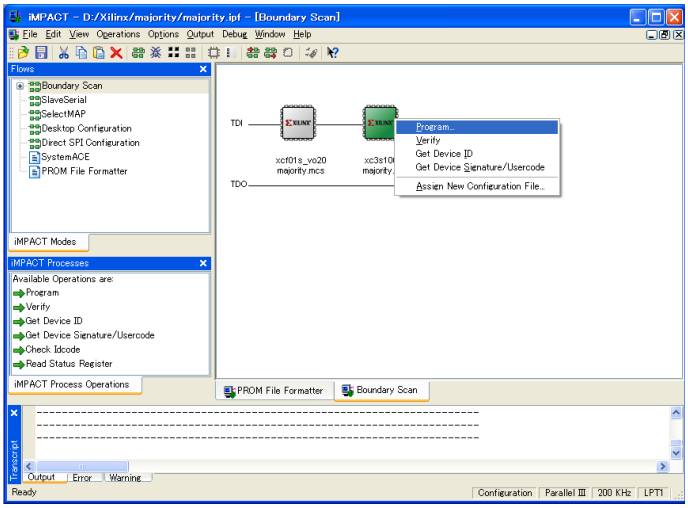


図 4.1.3-41

ここでは、直接 FPGA にデータを書き込んで見ましょう。

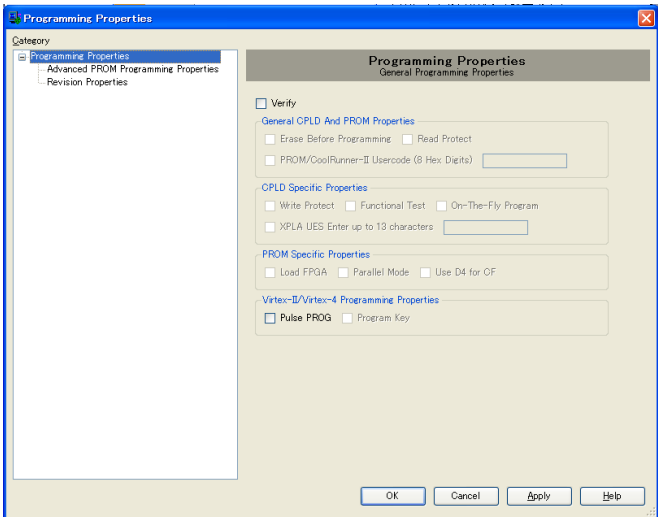


図 4.1.3-42

確認画面が表示されるので、そのまま OK ボタンをクリック。

4.2　モジュールの階層化

VerilogHDL で記述されたモジュールは、その後もモジュールブロックとして再利用が可能です。
モジュールの再利用には、逐次的な接続によるものと、階層構造的なものがあり、逐次的な接続では、各ポートを wire などのより接続します。また、階層構造として利用する場合は、同一のモジュール名で複数の同じ回路を使用出来るため、以下の書式により定義します。

階層構造でのモジュールの使用

モジュール名　インスタンス名（ポートリスト）;

例えば、自動車には４本のタイヤがあり、すべて同じ構造をしていますが、取り付けられる場所が異なります。そのため、各タイヤには識別名が必要となります。

モジュール名：タイヤ　インスタンス名：右前タイヤ
モジュール名：タイヤ　インスタンス名：左前タイヤ
モジュール名：タイヤ　インスタンス名：右後ろタイヤ
モジュール名：タイヤ　インスタンス名：左後ろタイヤ

のような定義文です。

また、各ポートリストの接続では、記述順による接続方法と、記述名による接続方法があります。

・記述順による接続方法

従属モジュール（利用される側のモジュール定義）

```
module slv(a, b, c,x);
        ・
endmodule
```

上位モジュール（利用する側の記述）

```
module master(port_list);
        ・
        wire w, x, y, z;
        ・
        slv  sub1(w, x, y, z);
        ・
endmodule
```

この場合は、ポートリストに記述した順にそれぞれ

上位側　　従属側

a → w
b → x
c → y
x → z

に接続されます。
（従属側と同じポート名があっても名前ではなく、あくまで記述順で接続されます）

・記述名による接続方法

従属モジュール（利用される側のモジュール定義）

```
module slv(a, b, c,x);
        ・
endmodule
```

上位モジュール（利用する側の記述）

```
module master(port_list);
        ・
        wire w, x, y, z;
        ・
        slv  sub1(.a(w), .c(x), .x(y), .b(z));
        ・
endmodule
```

この場合は、ワイヤーリストとポートリストに記述した名前でそれぞれ

上位側　　従属側

w → a

z → b

x → c

y → x

に接続されます。

（従属側ポートと同じ接続信号名があった場合でも、記述順では無くあくまで両モジュールで定義されている名前で接続されます）

5 超音波距離計の仕様

5.1 超音波距離計の原理

超音波距離計は、その名の通り超音波を発信する振動子と受信する振動子を用いてその時間的なずれを検出する事で距離を計測するものです。

人間のみみで聞こえる可聴波は２０～２０ｋHz ですが、超音波はおおむね４０ｋHz の音波を用います。この周波数の場合は人間の耳で聞く事は出来ません。
超音波の発信、受信に使用される振動子にバイモルフ振動子が使われます。

バイモルフ型振動子は、端子接続方向に伸縮する圧電素子を接合し、一方が伸びると他方が縮むように構成したもので撓み振動子となります。これは逆に撓みを与えると端子間には交流電界を出力するため、超音波センサーではこの二つの振動子の先にコーンを付け応用したものです。

本教材では、この振動子を応用した、パララックス社超音波距離センサーモジュールを FPGA 側からの信号で制御し距離計測を行います。

5.1.1 センサーモジュールの仕様

このセンサーモジュールは、SIG、Vcc、GND の３端子で接続され、SIG にトリガ（計測開始パルス）を印可すると一定時間後に計測された距離に応じたパルス幅の信号を同じ SIG に出力されます。
このパルス幅をを FPGA 内部ブロックで距離換算を行いその値をシリアルで出力しています。

トリガ： 2 µs (min), 5 µs typical
データ出力開始： 350 µs
距離データ信号： 115 µs ～18.5 ms

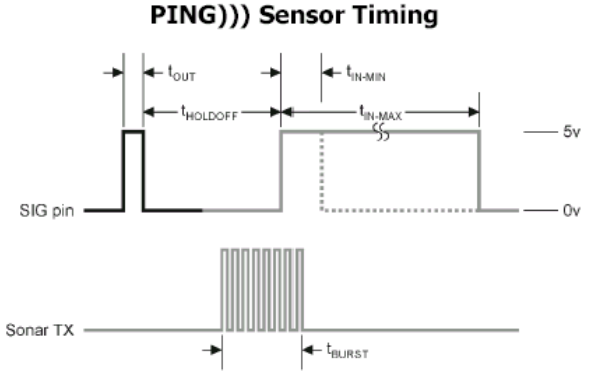


図 5.1.1

5.1.2　モジュール構成

①TOP module RangFinderForAVES(CLK, TX_SELECT, DATA_IN, RxD,
TRG_EN, DATA_EN, TRG, PLS_1mm, LED, COM, SEG7, TP_TRG, TxD);

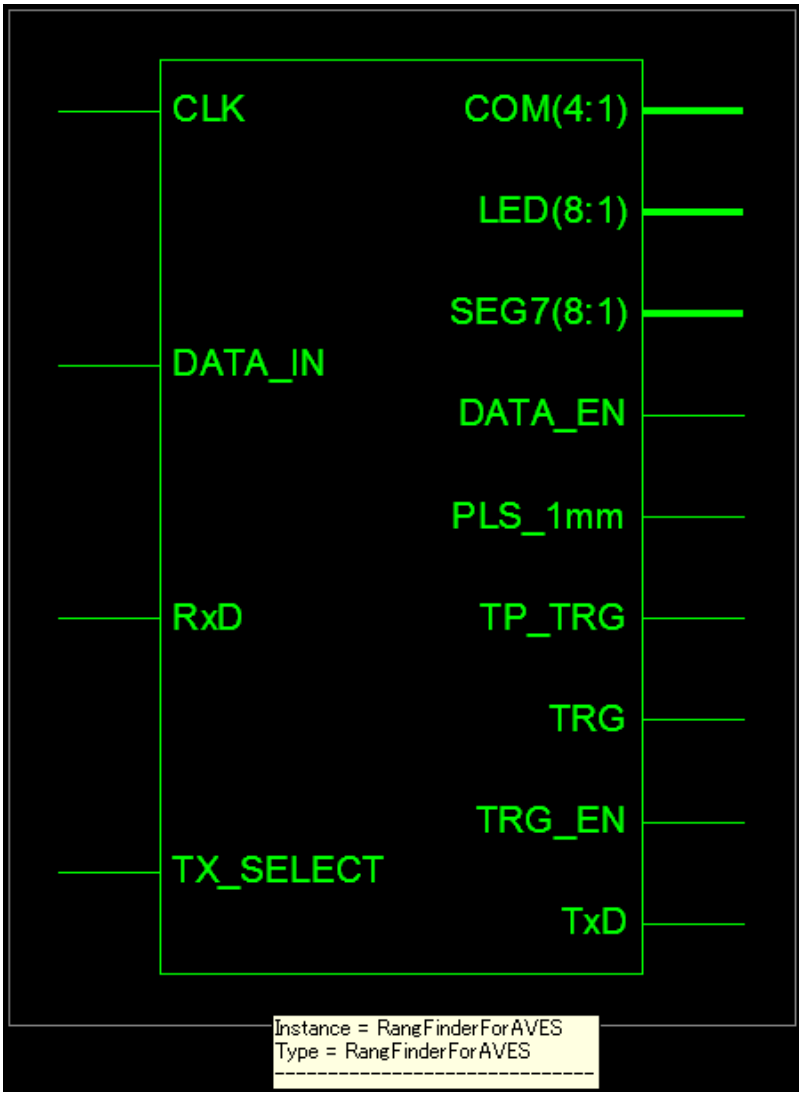


図 5.1.2-1

入力ポート

CLK :	33.333MHz Mastar Clock
DATA_IN :	Sensor Data
RxD :	RS232C Data Receive Port
TX_SELECT :	Input Transmit Data Mode Select

出力ポート

COM :	Select 7segLED
LED :	Monitor mm
SEG7 :	Display 7segment LED
DATA_EN :	Data Enable
PLS_1mm :	Monitor 1mm Pulse Clock
TP_TRG :	Monitor Trigger
TRG :	Trigger
TRG_EN :	Trigger Enable
TxD :	Send Data

②**U0** Gen_Com_Scan U0(CLK, ComScan);

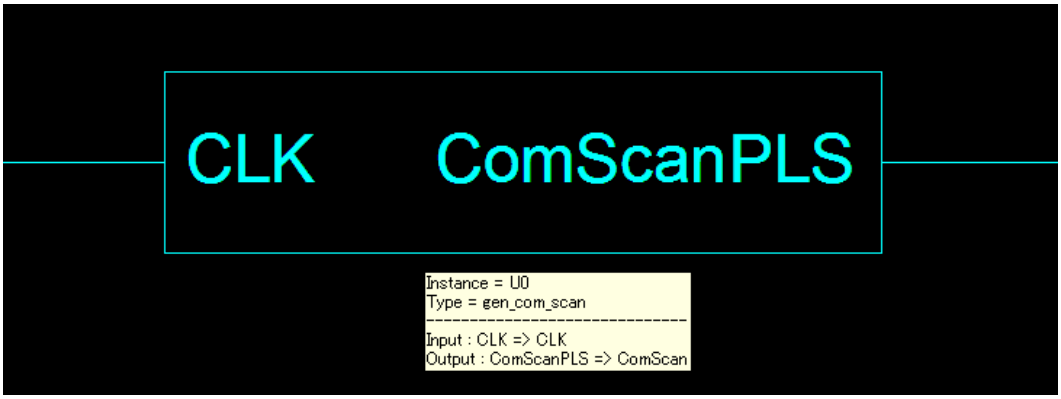


図 5.1.2-2

入力ポート

CLK : 33.333MHz Mastar Clock

出力ポート

ComScanPLS : Select 7segLED Pulse

③**U1** RS232C_CLKGEN U1(CLK, RST, TX_CK, RX_CK);

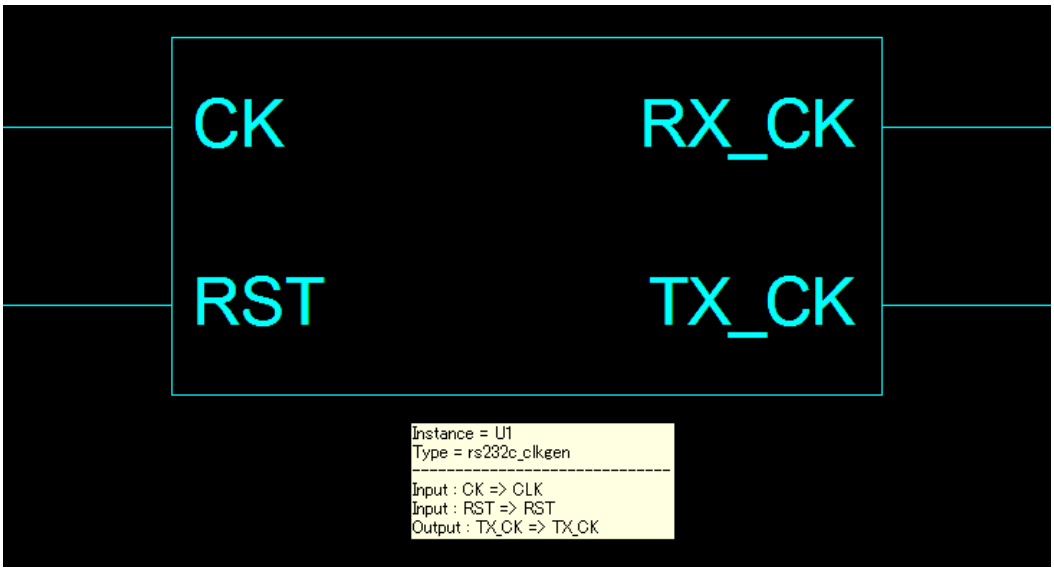


図 5.1.2-3

入力ポート

CLK : 33.333MHz Mastar Clock

RST :

出力ポート

RX_CK : Receive Clock

TX_CK : Transmit Clock

④U2 Send_Data_Slect U2(Tx_Data_Cnt, MODE, hold_renge_dec, Send_Data);

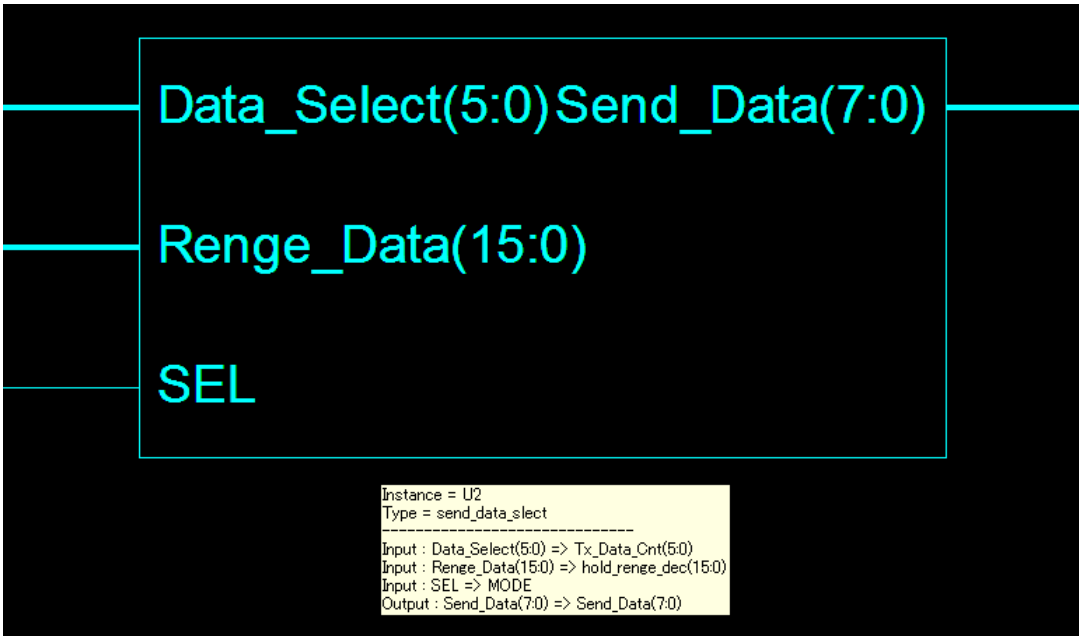


図 5.1.2-4

入力ポート

Data_Select :	Transmit Data Select
Renge_Data :	Renge Count Data
SEL :	Transmit Data Mode Select

出力ポート

Send_Data :	Transmit Data

⑤U3 RS232C_TX(TX_CK, RST, TX_EN, WR, PD_IN, SD_OUT) ;

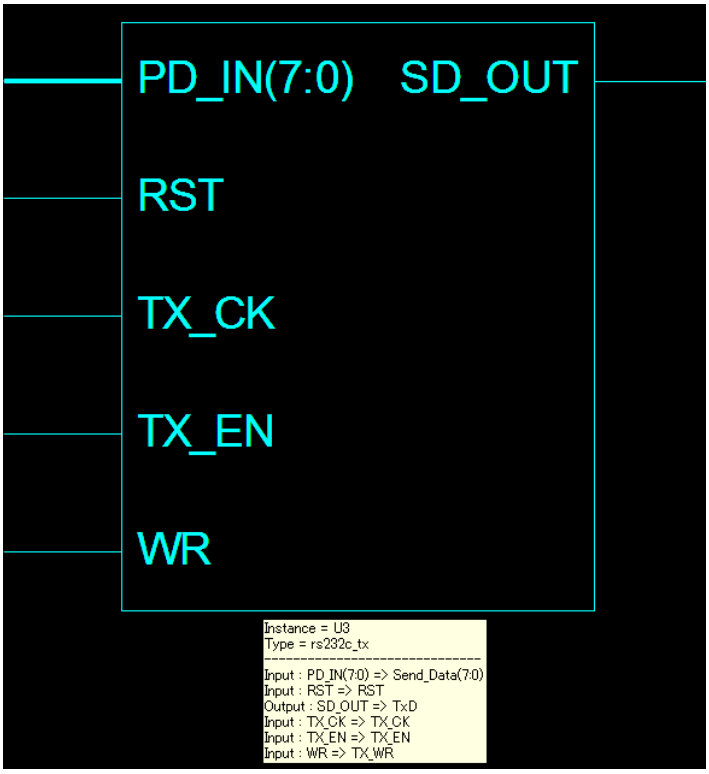


図 5.1.2-5

入力ポート

PD_IN :	Parallel Data Input
RST :	Reset
TX_CK :	Transmit Clock
RX_CK :	Receive Clock
WR :	Transmit Data Write

出力ポート

SD_OUT :	Serial Data Output

⑤U4 Gen_1mm_Pulse(CLK33M, pls_1mm);

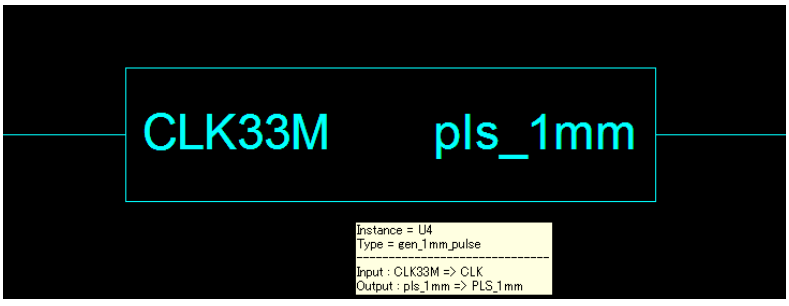


図 5.1.2-6

入力ポート

CLK33M :	33.333MHz Mastar Clock

出力ポート

pls_1mm :	1mm Range Pulse

⑤U5 Count_Renge(check_counter, RST, DATA_IN_EN, PLS_1mm, renge_dec, renge_bin);

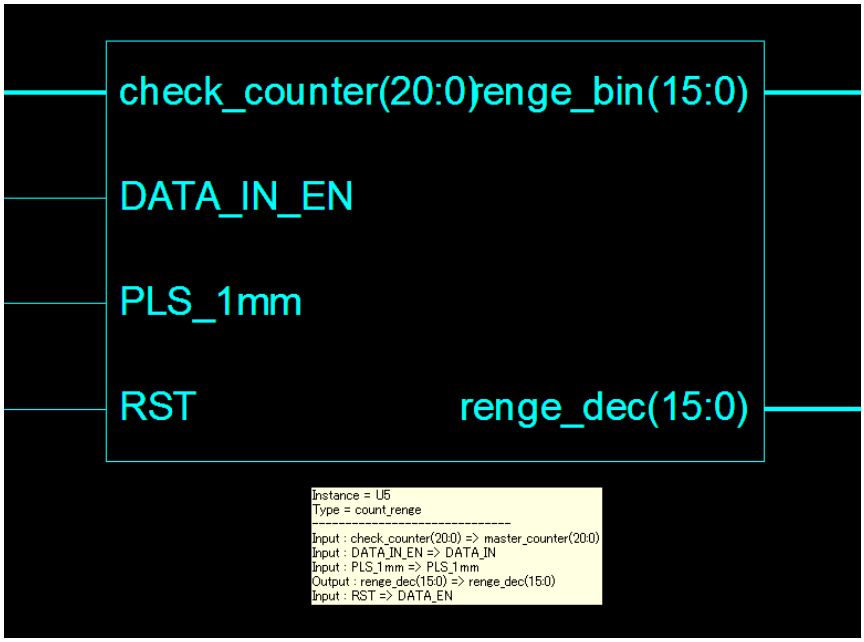


図 5.1.2-7

入力ポート

check_counter :	Check Count Number
DATA_IN_EN :	Range Data Enable
PLS_1mm :	1mm Range Pulse
RST :	Counter Reset

出力ポート

renge_bin :	Range Count Binary
renge_bin :	Range Count Decimal

⑤U6 Disp_Counter(SelCLK, Count_Data, Seg_Data, Sel);

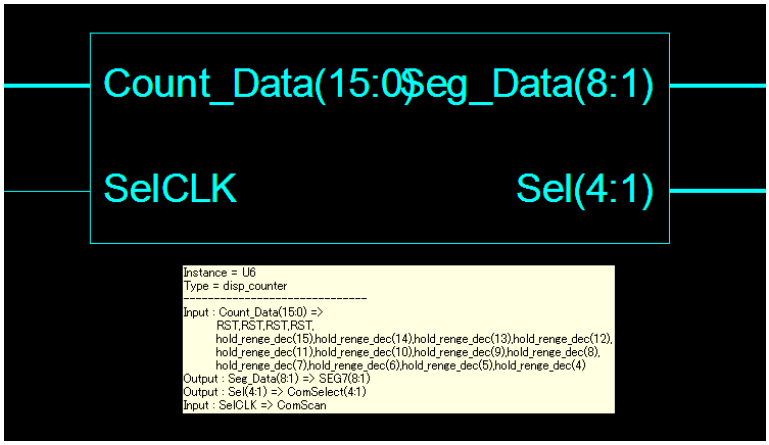


図 5.1.2-8

入力ポート

Count_Data :	Hold Renge Count Data
SelCLK :	Counter Reset

出力ポート

Seg_Data :	7segment LED Display Data
Sel :	Select 7segment LED

```
////////// Module Session

  // 7segLED common select PLS
  Gen_Com_Scan  U0(CLK, ComScan);

  // RS232C 送受信ｸﾛｯｸ生成 38400bps
  RS232C_CLKGEN  U1( CLK, RST, TX_CK, RX_CK );

  // Send_Data_Slect
  Send_Data_Slect  U2(Tx_Data_Cnt, MODE, hold_renge_dec, Send_Data);

  // RS232C ﾃﾞｰﾀ送信 data bit:8 stop bit:2 parity:none x-on/off:none
  RS232C_TX  U3( TX_CK, RST, TX_EN, TX_WR, Send_Data, TxD );

  // Generating 1mm Pulse(33.333MHz -> 99.799KHz(173.609375KHz))
  Gen_1mm_Pulse  U4(CLK, PLS_1mm);

  // Count_Renge
  Count_Renge  U5(master_counter, data_en_flg, DATA_IN, PLS_1mm, renge_dec, renge_bin);

  // Didplay Renge Data
  Disp_Counter  U6(ComScan, {4'h0,hold_renge_dec[15:4]}, SEG7, ComSelect);
```

5.1.3　センサーモジュールの信号処理手順

まず全体の処理手順を記述すると、以下の手順になります。

① 計測開始トリガイネーブル信号　　ON
② 計測開始トリガ信号　　ON
③ 計測開始トリガ信号　　OFF
④ 計測開始トリガイネーブル信号　　OFF

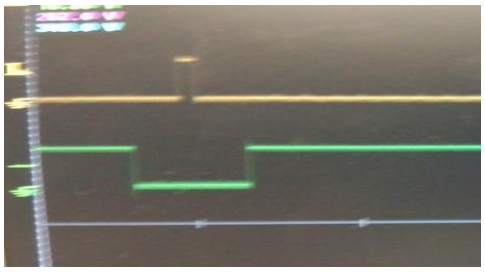


図 5.1.3-1　トリガ（上）とイネーブル信号（下）

⑤ 距離データイネーブル信号　　ON
⑥ 距離データ信号入力
⑦ 距離データイネーブル信号　　OFF

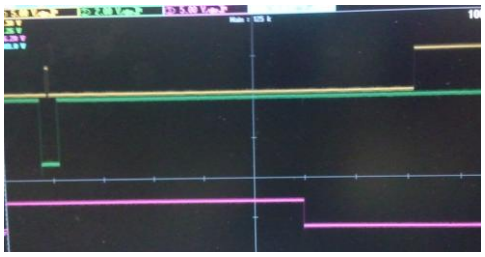


図 5.1.3-2　データ（上）とイネーブル信号（下：３本目）

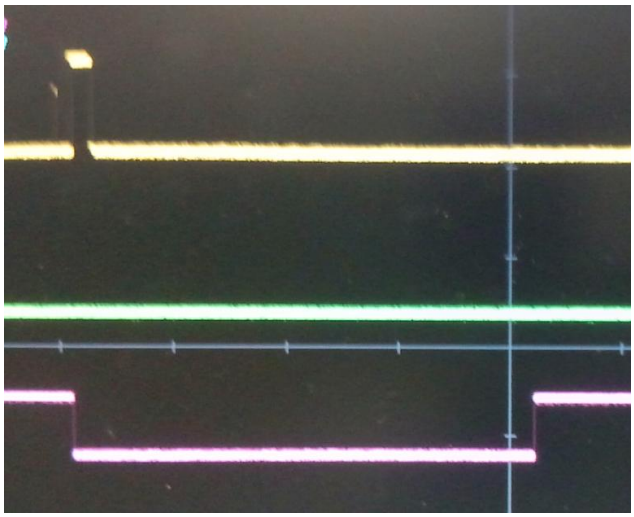


図 5.1.3-3　時間軸を縮小　（上：トリガとデータ）　（下：データイネーブル）

⑧ 計測距離カウント値　　ラッチ
⑨ 計測値シリアル出力

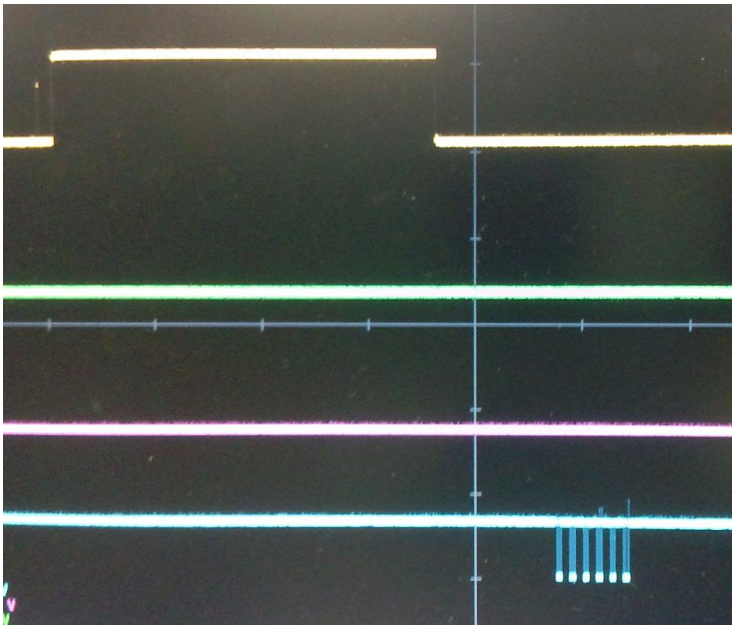


図 5.1.3-4　データ信号収束後（１本目）、シリアル転送（６バイト：４本目）

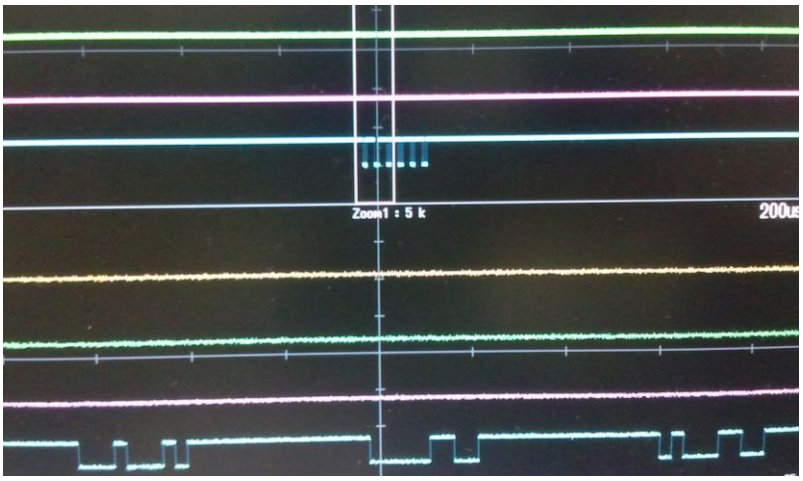


図 5.1.3-5　シリアルタイミング拡大図（最下線）

5.1.4　トリガおよびイネーブル信号の生成

ＦＰＧＡボードのマスタークロックは３３．３３３ＭＨｚであり周期は０．０３マイクロ秒です。
このクロックをベースにカウンタを構成し、5.1.2 の各タイミングを生成します。

計測トリガから、データ転送完了までを１ターンとし、これを４０ミリ秒とします。
これをベースとして、このカウンタを構成すると全体周波数は２５Ｈｚの処理サイクルとなります。

最小パルス幅であるトリガを正確に扱うために、マイクロ秒単位でカウンタの信号処理を行うためマスタークロック信号を、そのままカウンタクロックとして使用しています。

以下は、各信号タイミングです。

経過時間（μ秒）　　カウント値　　イベント

０・・・・・・・・・・０・・・・カウント・スタート

――――――センサー・コントロール・シーケンス

２０・・・・・・・・・６６６・・・・トリガイネーブル　ＯＮ
３５・・・・・・・・・１１６６・・・・トリガ　ＯＮ
４０・・・・・・・・・１３３３・・・・トリガ　ＯＦＦ
５５・・・・・・・・・１８３３・・・・トリガイネーブル　ＯＮ

――――――データ・カウント・シーケンス

５４０・・・・・・・１７９９９・・・・データイネーブル　ＯＮ（データカウントＯＮ）
２１０００・・・・・６９９９９３・・・・データイネーブル　ＯＦＦ（データカウントＯＦＦ）
２１１５０・・・・・７０４９９２・・・・データカウントラッチ

――――――データ・転送・シーケンス

（Reset Stop:3bit + Write:3bit + Dummy:3bit + TX_EN:13(S1/D8/ST2/DUMMY2)）

２４０００・・・・・８００００・・・・データ転送：１バイト目
２４６３０・・・・・８２１０００・・・・データ転送：２バイト目
２５２６０・・・・・８４２０００・・・・データ転送：３バイト目
２５８９０・・・・・８６３０００・・・・データ転送：４バイト目
２６５２０・・・・・８８４０００・・・・データ転送：５バイト目
２７１５０・・・・・９０５０００・・・・データ転送：６バイト目

――――――カウンタ・リセット・シーケンス

４００００・・・・１３３３３２０・・・・カウンタリセット

```
////////// Generating Triger    & Enable Pulse
    always @(posedge CLK)
    begin
            if(master_counter == 1333320) begin                    // 25Hz(40 msec)
                    master_counter <= 0;
                    trg_flg <= 0;
                    trg_en_flg <= 1;
                    data_en_flg <= 1;
                    Tx_Data_Cnt <= 6'b111110;
            end
    ////////////////////////////
    ///// triger Sequence
            else begin
                    /// triger enable
                    if(master_counter == 667) begin                        // 20 usec
                            trg_en_flg <= 0;
                    end
                            /// triger
                            if(master_counter == 1167) begin               // 35 usec
                                    trg_flg <= 1;
                            end
                            if(master_counter == 1333) begin               // 40 usec(39.99usec)
                                    trg_flg <= 0;
                            end
                    if(master_counter == 1834) begin                       // 55 usec
                            trg_en_flg <= 1;
                    end

    ////////////////////////////
    ///// Data Count Sequence
    ///// Data Count Start
                    if(master_counter == 18000) begin              // 540 usec
                            data_en_flg <=    0;
                    end

    ///// Count Data Save
                    if(master_counter == 700000) begin             // 21 msec
                            data_en_flg <= 1;
                    end

    ///// Data Count Stop
                    if(master_counter == 705000) begin             // 21.15 msec
                            hold_renge_dec <= renge_dec;
                    end
```

```
    ////////////////////////
    ///// RS232C Sequence Data 1 - 24.0002 msec /////
                    if(master_counter == data1_start) begin                 //  Sequence
Start
                    ///// transmit reset Stop
                            TX_RST = 0;
    // Reset transmit module
                    ///// Data Select Tx Data Type [B] or [D]
                            Tx_Data_Cnt <= 6'b111110;
                    end
                    ///// transmit data set
                    if(master_counter == (data1_start + bit3)) begin                //
130usec
                            TX_WR <=    1;
                    end
                    if(master_counter == (data1_start + bit6)) begin                //
260usec
                            TX_WR <=    0;
                    end
                    if(master_counter == (data1_start + bit9)) begin                //
312usec
                            TX_EN <=    1;
                    end
                    if(master_counter == (data1_start + bit24)) begin               //
312usec
                    ///// transmit reset
                            TX_RST = 1;
    // Reset transmit module
                            TX_EN <=    0;
                    end
```

```
    ///// RS232C Sequence Data 2 - 24.6302 msec /////
                ///// transmit reset
                if(master_counter == data2_start) begin                    //    Sequence
Start
                ///// transmit reset Stop
                        TX_RST = 0;
    // Reset transmit module
                ///// Data Select Tx Data Bit15-12
                        Tx_Data_Cnt <= 6'b111101;
                end
                ///// transmit data set
                if(master_counter == (data2_start + bit3)) begin                    //
130usec
                        TX_WR <=    1;
                end
                if(master_counter == (data2_start + bit6)) begin                    //
260usec
                        TX_WR <=    0;
                end
                if(master_counter == (data2_start + bit9)) begin                    //
312usec
                        TX_EN <=    1;
                end
                if(master_counter == (data2_start + bit24)) begin                    //
312usec
                ///// transmit reset
                        TX_RST = 1;
    // Reset transmit module
                        TX_EN <=    0;
                end
```

```
    ///// RS232C Sequence Data 3 - 25.2603 msec /////
                    ///// transmit reset
                    if(master_counter == data3_start) begin                    //    Sequence
Start
                    ///// transmit reset Stop
                        TX_RST = 0;
    // Reset transmit module
                    ///// Data Select Tx Data Bit11-8
                        Tx_Data_Cnt <= 6'b111011;
                    end
                    ///// transmit data set
                    if(master_counter == (data3_start + bit3)) begin                //
130usec
                        TX_WR <=   1;
                    end
                    if(master_counter == (data3_start + bit6)) begin                //
260usec
                        TX_WR <=   0;
                    end
                    if(master_counter == (data3_start + bit9)) begin                //
312usec
                        TX_EN <=   1;
                    end
                    if(master_counter == (data3_start + bit24)) begin               //
312usec
                        TX_RST = 1;
    // Reset transmit module
                        TX_EN <=   0;
                    end
```

```
	///// RS232C Sequence Data 4 - 25.8903 msec /////
			///// transmit reset
			if(master_counter == data4_start) begin		//	Sequence
Start
			///// transmit reset Stop
				TX_RST = 0;
	// Reset transmit module
			///// Data Select Tx Data Bit7-4
				Tx_Data_Cnt <= 6'b110111;
			end
			///// transmit data set
			if(master_counter == (data4_start + bit3)) begin		//
130usec
				TX_WR <=  1;
			end
			if(master_counter == (data4_start + bit6)) begin		//
260usec
				TX_WR <=  0;
			end
			if(master_counter == (data4_start + bit9)) begin		//
312usec
				TX_EN <=  1;
			end
			if(master_counter == (data4_start + bit24)) begin		//
312usec
				TX_RST = 1;
	// Reset transmit module
				TX_EN <=  0;
			end
```

```
    ///// RS232C Sequence Data 5 - 26.5203 msec /////
                    ///// transmit reset
                    if(master_counter == data5_start) begin                    //    Sequence
Start
                    ///// transmit reset Stop
                            TX_RST = 0;
    // Reset transmit module
                    ///// Data Select Tx Data Bit3-0
                            Tx_Data_Cnt <= 6'b101111;
                    end
                    ///// transmit data set
                    if(master_counter == (data5_start + bit3)) begin                    //
130usec
                            TX_WR <=    1;
                    end
                    if(master_counter == (data5_start + bit6)) begin                    //
260usec
                            TX_WR <=    0;
                    end
                    if(master_counter == (data5_start + bit9)) begin                    //
312usec
                            TX_EN <=    1;
                    end
                    if(master_counter == (data5_start + bit24)) begin                    //
312usec
                            TX_RST = 1;
    // Reset transmit module
                            TX_EN <=    0;
                    end
```

```
        ///// RS232C Sequence Data 6 - 27.1503 msec/////
                        ///// transmit reset
                        if(master_counter == data6_start) begin                 //  Sequence
Start
                        ///// transmit reset Stop
                                TX_RST = 0;
        // Reset transmit module
                        ///// Data Select Tx Data End Mark [E]
                                Tx_Data_Cnt <= 6'b011111;
                        end
                        ///// transmit data set
                        if(master_counter == (data6_start + bit3)) begin                //
130usec
                                TX_WR <=    1;
                        end
                        if(master_counter == (data6_start + bit6)) begin                //
260usec
                                TX_WR <=    0;
                        end
                        if(master_counter == (data6_start + bit9)) begin                //
312usec
                                TX_EN <=    1;
                        end
                        if(master_counter == (data6_start + bit24)) begin               //
312usec
                                TX_RST = 1;
        // Reset transmit module
                                TX_EN <=    0;
                        end

        ///// RS232C Sequence end /////
        ////////////////////////////////

                        master_counter <= master_counter + 1;

                end
        end
```

6 シリアルインターフェースモジュール

6.1 モジュール概要

シリアルモジュールは、パラレルデータ（PD_IN）、モジュールリセット（RST）、転送クロック（TX_CK）、転送イネーブル（TX_EN）、出力データセット（WR）を入力ポート、シリアルデータ（SD_OUT）を出力ポートとしています。

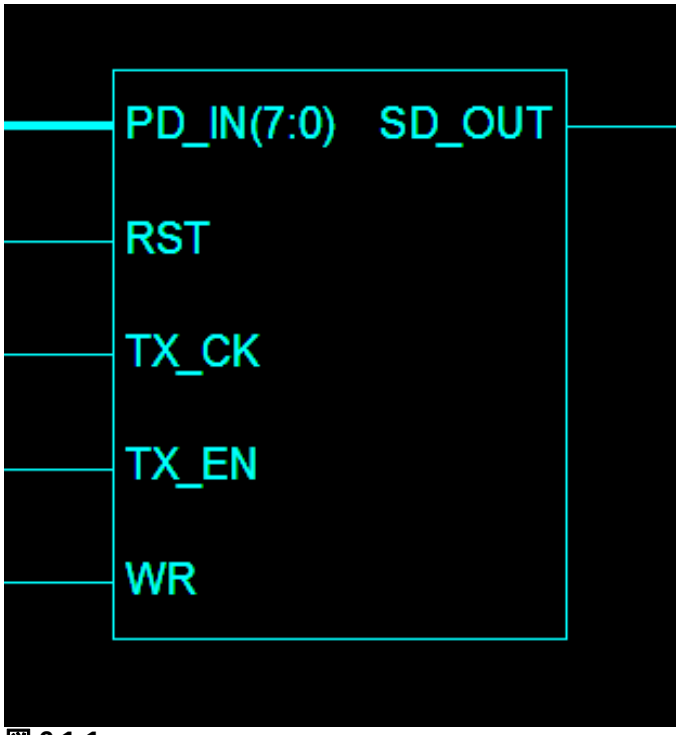


図 6.1-1

通常 UART デバイス等では、転送クロックの１６倍や、６４倍のクロックを必要としますが、これは調歩同期に於いて、わずかな転送レートの誤差があっても、ボーレートの安定をはかるためです。
今回は、送信モジュールのみで、誤差を修正するのは受信側に依存する為ボーレートと同じ等倍のクロックを使用します。

今回は、クロックジェネレータで作り出された３８４００Hz を使用します。
実際にボーレートは受信側の環境や計測周期と転送データ量で決まります。

以下は、送信クロックジェネレータのモジュールです。

```
/***                */
/**  RS232C_CLKGEN  **/
/*                ***/
module RS232C_CLKGEN( CK, RST, TX_CK, RX_CK );
        input  CK;
        input  RST;
        output TX_CK;
        output RX_CK;

        // ﾎﾞｰﾚｰﾄ用ｸﾛｯｸ生成ｶｳﾝﾄ値
        parameter bps110   = 303027;
        parameter bps150   = 222220;
        parameter bps300   = 111110;
        parameter bps1200  =  27777;
        parameter bps2400  =  13888;
        parameter bps4800  =   6944;
        parameter bps9600  =   3472;
        parameter bps19200 =   1736;
        //parameter bps38400 =    868;
        parameter bps38400 =    434;

        parameter bps = bps38400;

        reg [19:0] TX_CNT;
        reg [19:0] RX_CNT;
        reg RX_FLG;
        reg TX_FLG;

        always @( posedge CK )
        begin
                if( RST ) begin
                        RX_CNT <= 0;
                        RX_FLG <= 0;
                        TX_CNT <= 0;
                        TX_FLG <= 0;
                end
                else begin
                        if( RX_CNT == ( bps / 16 ) ) begin       // TX_CNT -> 1/16
                                RX_CNT <= 0;
                                RX_FLG <= 1;
                        end
                        else begin
                                RX_CNT <= RX_CNT + 1;
                                RX_FLG <= 0;
                        end
                        if( TX_CNT == bps ) begin// 38400bps
                                TX_CNT <= 0;
        //                      TX_FLG <= 1;
                                TX_FLG <= TX_FLG ^ 1;
                        end
```

```
                        else begin
                                TX_CNT <= TX_CNT + 1;
//                              TX_FLG <= 0;
                        end
                end
        end

        assign RX_CK = RX_FLG;
        assign TX_CK = TX_FLG;

endmodule
```

モジュールリセットは、転送モジュールのビットカウンタのリセット、送信データのラッチフラグのリセットを行い、送信データ計測中はリッセトの状態のままとし、計測終了後、送信データが決定した後リセットを解除します。

この時、最低のリセット時間を５ビット幅として確実にリセットが行えるようにしてあります。

リセットが解除されると、内部バッファに送信データがラッチされ、送信イネーブルを待ちます。

リセット解除後送信イネーブルが入る時間も５ビット幅を確保しデータのラッチを確実に行う様に設定します。

この後、送信クロックのエッジをとりがとして、最初のクロックでスタートビット（０）の送出からビット数をカウントしながらストップビットまでを順次送出します。

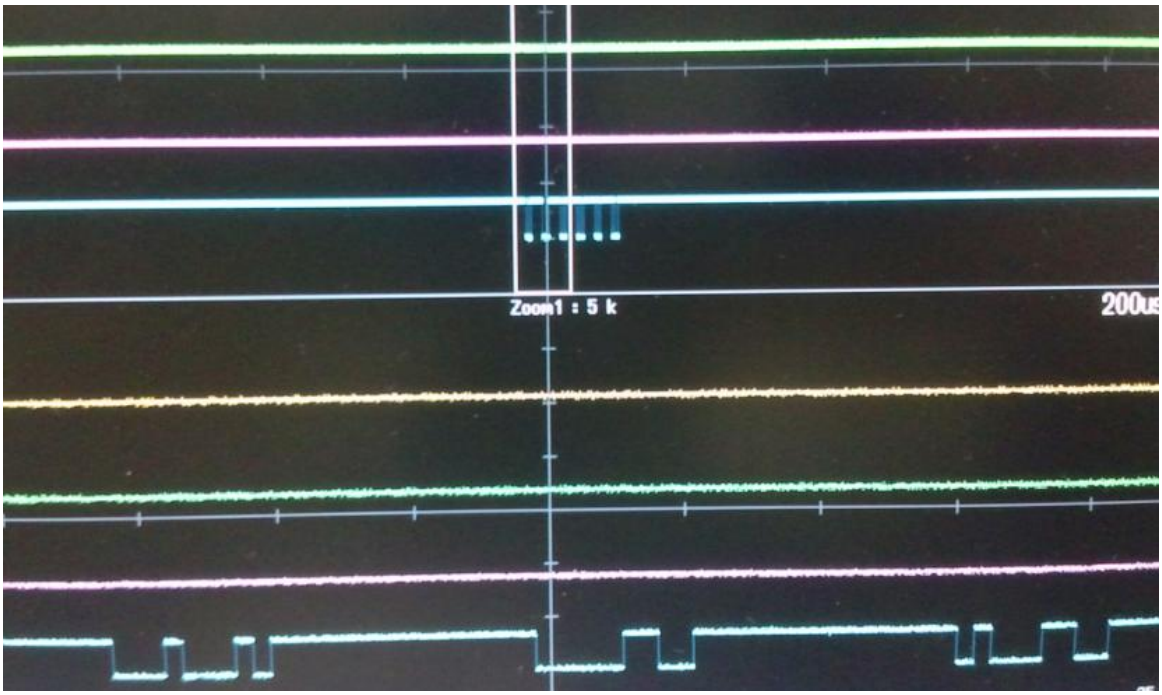


図 6.1-2　送信波形

はじめの１バイト目の波形を拡大すると、

図 6.1-3　１バイトの波形

０：スタートビット
０：ｂｉｔ０（ＬＳＢ）
０：ｂｉｔ１
１：ｂｉｔ２
０：ｂｉｔ３
０：ｂｉｔ４
０：ｂｉｔ５
１：ｂｉｔ６
０：ｂｉｔ７（ＭＳＢ）
１：ストップビット１
１：ストップビット２

となり、

データビットをＭＳＢからＬＳＢに並び替えると

０１００・０１００　　０ｘ４４（ＪＩＳコード）・・・Ｄ

即ち、英大文字の「Ｄ」で、これは送信される計測距離データが１０進数であることを示しています。

```
/***                  */
/**    RS232C_TX     **/
/*   d8 pn s1 xoff  ***/
module RS232C_TX( TX_CK, RST, TX_EN, WR, PD_IN, SD_OUT ) ;
        input  TX_CK;
        input  RST;
        input  TX_EN;
        input  WR;
        input  [7:0] PD_IN;      // ﾊﾟﾗﾚﾙﾃﾞｰﾀ入力
        output SD_OUT;                   // ｼﾘｱﾙﾃﾞｰﾀ出力

        reg    SD_OUT;

        reg    LOAD;
        reg    [7:0] SERIAL;
        reg    [3:0] CBIT;

        always @( posedge TX_CK ) begin
                if( RST ) begin
                        SD_OUT <= 1'b1;
                        CBIT <= 4'h0;
                        LOAD  <= 1'b0;
                end
                else begin
                        if( WR ) begin
                                LOAD   <= 1'b1;
                                SERIAL <= PD_IN;
                        end
                        else begin
                                if( TX_EN && LOAD ) begin
                                        case( CBIT )
                                                // ｽﾀｰﾄﾋﾞｯﾄの送出(0)
                                                0 :
                                                begin
                                                        SD_OUT <= 1'b0;
                                                        CBIT <= CBIT + 4'h1;
                                                end
```

```
                                                // ﾃﾞｰﾀﾋﾞｯﾄの送出
                                                1,2,3,4,5,6,7,8 :
                                                begin
                                                        SD_OUT   <= SERIAL[0];
                                                        SERIAL <= { 1'b1, SERIAL[7:1]};
                                                        CBIT   <= CBIT + 4'h1;
                                                end
                                                // ｽﾄｯﾌﾟﾋﾞｯﾄの送出(1bit 目)
                                                9 :
                                                begin
                                                        SD_OUT  <= 1'b1;
                                                        CBIT   <= CBIT + 4'h1;
//                                                      LOAD  <= 1'b0;
                                                end

//      ｽﾄｯﾌﾟﾋﾞｯﾄが 1 ﾋﾞｯﾄの場合以下をｺﾒﾝﾄ
////////////////////////////////////////
//                                              // ｽﾄｯﾌﾟﾋﾞｯﾄの送出(2bit 目)
                                                10 :
                                                begin
                                                        SD_OUT  <= 1'b1;
                                                        CBIT  <= 4'h0;
                                                        LOAD  <= 1'b0;
                                                end
////////////////////////////////////////
                                                default :
                                                begin
                                                        SD_OUT <= 1'b1;
                                                        CBIT <= 4'h0;
                                                end
                                        endcase
                                end     //      if (TX_EN && LOAD)
                        end     //end else (WR)
                end     // end else (RST)
        end     // end always

endmodule
```

7 おわりに

ハードウェアとソフトウェア

本ハードウェア編では FPGA の開発手法として、VerilogHDL にる記述をご紹介しマイコンとの連携の解説はソフトウェア編でおこないます。

実際のシステムに於いては、当初より処理速度・開発／実装コストによってハードウェアとソフトウェアのトレードオフが行われてきました。これまではコスト面や開発環境面でマイコンによる機能実現が最も効率の良いシステムでした。しかし、近年デジタルデバイスの飛躍的な技術革新により、ハードウェアによる機能実現が可能になってきました。

ハードウェアによる機能実現では、何よりも処理速度および並列処理が可能である事が挙げられます。

これは、マイコンによるソフトウェアの命令デコード処理を無くし、更に、リアルタイムに並列信号処理が可能であると言うことです。では、全てがこの様なハードウェアに置き換えれば良いのでしょうか？

答えは“NO”です。確かに「処理速度」・「並列処理」等ではハードウェアによる実装が高いパフォーマンスを実現できます。しかし、実数演算などを行う場合、ハードウェアではかなり複雑な回路を必要とし、コストパフォーマンスに於いてマイコンによるソフトウェアには及びません。そこでマイコンとの連携システムにより、より高機能で、高精度なシステムを実現できるのです。

とかく、システムを構築する場合に得意な分野に偏りがちになります。これは、ある意味当然の事だと思います。不得意な分野と組み合わせて時間をかけるより、得意な分野で進めれば無駄な時間をかけずにシステムの設計が出来ます。でも・・・もし、システムにとってそれが最適な構成？と考えた場合、ソフトウェアとハードウェアによる連携システムが最善の構成となる事が良くあるのではないでしょうか。

近年の技術革新により、今までよりも多くの場面でハードウェアが利用可能な環境が整ってきました。特に、FPGA によるシステムは、今までソフトウェアの開発を行ってきた方々でも、HDL による設計で比較的容易に利用できます。文法的にソフトウェア言語と共通点が多いので比較的容易に理解できると思います。

強いて言えば、ソフトウェア言語との大きな違いは、処理が記述順ではなく全てが同時実行されるという超リアルタイム処理と言うことです。この感覚はいくつかの経験を積んで行く事が理解の為の早道です。

8 Appendix

超音波距離計のための VerilogHDL　ソースリスト

8.1 RangFinderForAVES.v （TOP Module）

```
////////////////////////////////////////////////////////////////////////////////
module RangFinderForAVES(
  CLK, TX_SELECT, DATA_IN, TRG_EN,
  DATA_EN, TRG, PLS_1mm,
  LED, COM, SEG7, TP_TRG,
  RxD, TxD);

  input  CLK;                          // Input Master clock
  input    TX_SELECT;                  // Input Transmit Data Select
  input  DATA_IN;                      // Input Ultrasonic sensor
  input  RxD;                          // RS232C 受信ﾃﾞｰﾀ
  output TRG_EN;                       // Sensing trigger enable
  output DATA_EN;                      // Data enable
  output TRG;                          // Trigger
  output PLS_1mm;
  output [8:1]LED;                     // Monitor mm
  output [4:1]COM;                     // Select 7segLED
  output [8:1]SEG7;                    // Display
  output TP_TRG;                       // ﾄﾘｶﾞ・ﾓﾆﾀｰ
  output TxD;                          // RS232C 送信ﾃﾞｰﾀ

  parameter data1_start = 800000;      // 24.0002 msec
  parameter data2_start = 821000;      // 24.6302 msec
  parameter data3_start = 842000;      // 25.2603 msec
  parameter data4_start = 863000;      // 25.8903 msec
  parameter data5_start = 884000;      // 26.5203 msec
  parameter data6_start = 905000;      // 27.1503 msec

  parameter bit3  = 2604;              //  78.12 usec --- Tx Reset
  parameter bit6  = 5208;              // 156.24 usec --- Tx Data Write
  parameter bit9  = 7812;              // 234.36 usec --- Hold
  parameter bit24 = 20832;             // 624.97 usec --- Tx Start

  reg [20:0]master_counter = 0;        // Internal master counter initialize
  reg trg_en_flg = 1;                  // Sensing trigger enable flag
```

```
  reg trg_flg = 0;                                  // Sensing trigger flag
  reg data_en_flg = 1;                          // Range data enable flag

  reg [7:0]count_1mm = 0;                   // 1mm counter initialize
  reg count_1mm_flg = 0;                    // 1mm counter flag

  reg TX_WR = 0;                                // Enable transmit flag
  reg TX_EN = 0;                                // Enable transmit flag
  reg TX_RST = 1;                               // Reset transmit module
  reg MODE = 1;

  reg [5:0]Tx_Data_Cnt = 6'b111110;         // Tx Data Select Pulse

  reg [15:0]hold_renge_dec = 0;             // Tx & Display Hold Data(Dec)

  wire [15:0]renge_dec;                     // Count Renge Data (Dec)
  wire [15:0]renge_bin;                     // Count Renge Data (Bin)

  wire [15:0]hold_renge_sel;
  wire Count_Data_Enable;
  wire ComScan;
  wire [6:1]ComSelect;
  wire [7:0]Send_Data;

  wire TX_CK;
  wire RX_CK;

////////// Assign I/O Pulse
  assign  TRG_EN = trg_en_flg;
  assign  TRG = trg_flg;
  assign  DATA_EN = data_en_flg;
  assign  TP_TRG = trg_en_flg & trg_flg;

  assign  LED[4:1] = ~hold_renge_dec[3:0];
  assign  LED[8:5] = {MODE, TX_SELECT, 2'b11};
  assign  COM = ComSelect[4:1];

////////// Generating Transmit Sel Flg
  always @(posedge TX_SELECT)
  begin
          MODE <= MODE ^ 1;
  end

////////// Module Session
  // 7segLED common select PLS
  Gen_Com_Scan U0(CLK, ComScan);

  // RS232C 送受信ｸﾛｯｸ生成 38400bps
  RS232C_CLKGEN U1( CLK, RST, TX_CK, RX_CK );
```

```
  // Send_Data_Slect
  Send_Data_Slect U2(Tx_Data_Cnt, MODE, hold_renge_dec, Send_Data);

  // RS232C ﾃﾞｰﾀ送信 data bit:8 stop bit:2 parity:none x-on/off:none
  RS232C_TX     U3( TX_CK, RST, TX_EN, TX_WR, Send_Data, TxD );

  // Generating 1mm Pulse(33.333MHz -> 99.799KHz(173.609375KHz))
  Gen_1mm_Pulse U4(CLK, PLS_1mm);

  // Count_Renge
  Count_Renge U5(master_counter, data_en_flg, DATA_IN, PLS_1mm, renge_dec, renge_bin);

  // Didplay Renge Data
  Disp_Counter U6(ComScan, {4'h0,hold_renge_dec[15:4]}, SEG7, ComSelect);

////////// Generating Triger  & Enable Pulse
  always @(posedge CLK)
  begin
          if(master_counter == 1333320) begin                       // 25Hz(40 msec)
                  master_counter <= 0;
                  trg_flg <= 0;
                  trg_en_flg <= 1;
                  data_en_flg <= 1;
                  Tx_Data_Cnt <= 6'b111110;
          end
  ////////////////////////
  ///// triger Sequence
          else begin
                  /// triger enable
                  if(master_counter == 667) begin                   // 20 usec
                          trg_en_flg <= 0;
                  end
                          /// triger
                          if(master_counter == 1167) begin          // 35 usec
                                  trg_flg <= 1;
                          end
                          if(master_counter == 1333) begin          // 40 usec(39.99usec)
                                  trg_flg <= 0;
                          end
                  if(master_counter == 1834) begin                  // 55 usec
                          trg_en_flg <= 1;
                  end

  ////////////////////////
  ///// Data Count Sequence
  ///// Data Count Start
                  if(master_counter == 18000) begin                 // 540 usec
                          data_en_flg <=  0;
                  end
```

```
///// Count Data Save
                if(master_counter == 700000) begin                // 21 msec
                        hold_renge_dec <= renge_dec;
                end

///// Data Count Stop
                if(master_counter == 705000) begin                // 21.15 msec
                        data_en_flg <= 1;
                end

//////////////////////////
///// RS232C Sequence Data 1 - 24.0002 msec /////
                if(master_counter == data1_start) begin   // Sequence Start
                ///// transmit reset Stop
                        TX_RST = 0;                       // Reset transmit module
                ///// Data Select Tx Data Type [B] or [D]
                        Tx_Data_Cnt <= 6'b111110;
                end
                ///// transmit data set
                if(master_counter == (data1_start + bit3)) begin  // 130usec
                        TX_WR <=  1;
                end
                if(master_counter == (data1_start + bit6)) begin  // 260usec
                        TX_WR <=  0;
                end
                if(master_counter == (data1_start + bit9)) begin  // 312usec
                        TX_EN <=  1;
                end
                if(master_counter == (data1_start + bit24)) begin // 312usec
                ///// transmit reset
                        TX_RST = 1;             // Reset transmit module
                        TX_EN <=  0;
                end

///// RS232C Sequence Data 2 - 24.6302 msec /////
                ///// transmit reset
                if(master_counter == data2_start) begin   // Sequence Start
                ///// transmit reset Stop
                        TX_RST = 0;                       // Reset transmit module
                ///// Data Select Tx Data Bit15-12
                        Tx_Data_Cnt <= 6'b111101;
                end
                ///// transmit data set
                if(master_counter == (data2_start + bit3)) begin  // 130usec
                        TX_WR <=  1;
                end
                if(master_counter == (data2_start + bit6)) begin  // 260usec
                        TX_WR <=  0;
                end
                if(master_counter == (data2_start + bit9)) begin  // 312usec
```

```
                        TX_EN <=  1;
                end
                if(master_counter == (data2_start + bit24)) begin // 312usec
                ///// transmit reset
                        TX_RST = 1;             // Reset transmit module
                        TX_EN <=  0;
                end

        ///// RS232C Sequence Data 3 - 25.2603 msec /////
                ///// transmit reset
                if(master_counter == data3_start) begin   // Sequence Start
                ///// transmit reset Stop
                        TX_RST = 0;                       // Reset transmit module
                ///// Data Select Tx Data Bit11-8
                        Tx_Data_Cnt <= 6'b111011;
                end
                ///// transmit data set
                if(master_counter == (data3_start + bit3)) begin  // 130usec
                        TX_WR <=  1;
                end
                if(master_counter == (data3_start + bit6)) begin  // 260usec
                        TX_WR <=  0;
                end
                if(master_counter == (data3_start + bit9)) begin  // 312usec
                        TX_EN <=  1;
                end
                if(master_counter == (data3_start + bit24)) begin // 312usec
                        TX_RST = 1;             // Reset transmit module
                        TX_EN <=  0;
                end

        ///// RS232C Sequence Data 4 - 25.8903 msec /////
                ///// transmit reset
                if(master_counter == data4_start) begin   // Sequence Start
                ///// transmit reset Stop
                        TX_RST = 0;                       // Reset transmit module
                ///// Data Select Tx Data Bit7-4
                        Tx_Data_Cnt <= 6'b110111;
                end
                ///// transmit data set
                if(master_counter == (data4_start + bit3)) begin  // 130usec
                        TX_WR <=  1;
                end
                if(master_counter == (data4_start + bit6)) begin  // 260usec
                        TX_WR <=  0;
                end
                if(master_counter == (data4_start + bit9)) begin  // 312usec
                        TX_EN <=  1;
                end
                if(master_counter == (data4_start + bit24)) begin // 312usec
```

```
                        TX_RST = 1;             // Reset transmit module
                        TX_EN <=  0;
                end

        ///// RS232C Sequence Data 5 - 26.5203 msec /////
                ///// transmit reset
                if(master_counter == data5_start) begin   // Sequence Start
                ///// transmit reset Stop
                        TX_RST = 0;                       // Reset transmit module
                ///// Data Select Tx Data Bit3-0
                        Tx_Data_Cnt <= 6'b101111;
                end
                ///// transmit data set
                if(master_counter == (data5_start + bit3)) begin  // 130usec
                        TX_WR <=  1;
                end
                if(master_counter == (data5_start + bit6)) begin  // 260usec
                        TX_WR <=  0;
                end
                if(master_counter == (data5_start + bit9)) begin  // 312usec
                        TX_EN <=  1;
                end
                if(master_counter == (data5_start + bit24)) begin // 312usec
                        TX_RST = 1;             // Reset transmit module
                        TX_EN <=  0;
                end

        ///// RS232C Sequence Data 6 - 27.1503 msec/////
                ///// transmit reset
                if(master_counter == data6_start) begin   // Sequence Start
                ///// transmit reset Stop
                        TX_RST = 0;                       // Reset transmit module
                ///// Data Select Tx Data End Mark [E]
                        Tx_Data_Cnt <= 6'b011111;
                end
                ///// transmit data set
                if(master_counter == (data6_start + bit3)) begin  // 130usec
                        TX_WR <=  1;
                end
                if(master_counter == (data6_start + bit6)) begin  // 260usec
                        TX_WR <=  0;
                end
                if(master_counter == (data6_start + bit9)) begin  // 312usec
                        TX_EN <=  1;
                end
                if(master_counter == (data6_start + bit24)) begin // 312usec
                        TX_RST = 1;             // Reset transmit module
                        TX_EN <=  0;
                end
```

```
    ///// RS232C Sequence end /////
    ////////////////////////////////

                    master_counter <= master_counter + 1;

            end
    end

endmodule
```

8.2 Gen_Com_Scan.v（U0 Module）

```
////////////////////////////////////////////////////////////////////////////////
Module Gen_Com_Scan(CLK, ComScanPLS);
  input CLK;
  output ComScanPLS;

  reg [19:0]counter = 0;
  reg ComSan_flg = 0;

  assign ComScanPLS = ComSan_flg;

  always @(posedge CLK) begin
//                if(counter == 16666) begin
          if(counter == 90000) begin
                  counter <= 20'h00000;
                  ComSan_flg <= ComSan_flg ^ 1'b1;
          end
          else begin
                  counter <= counter + 1;
          end
  end

endmodule
```

8.3 Gen_Clk_R232C.v（U1 Module）

```
////////////////////////////////////////////////////////////////////////////////
/***                 */
/**  RS232C_CLKGEN  **/
/*                 ***/
module RS232C_CLKGEN( CK, RST, TX_CK, RX_CK );
input  CK;
input  RST;
output TX_CK;
output RX_CK;

// ﾎﾞｰﾚｰﾄ用ｸﾛｯｸ生成ｶｳﾝﾄ値
parameter bps110   = 303027;
parameter bps150   = 222220;
parameter bps300   = 111110;
parameter bps1200  =  27777;
parameter bps2400  =  13888;
parameter bps4800  =   6944;
parameter bps9600  =   3472;
parameter bps19200 =   1736;
//parameter bps38400 =    868;
parameter bps38400 =    434;

parameter bps = bps38400;

reg [19:0] TX_CNT;
reg [19:0] RX_CNT;
reg RX_FLG;
reg TX_FLG;

always @( posedge CK )
begin
  if( RST ) begin
          RX_CNT <= 0;
          RX_FLG <= 0;
          TX_CNT <= 0;
          TX_FLG <= 0;
  end
  else begin
          if( RX_CNT == ( bps / 16 ) ) begin        // TX_CNT -> 1/16
                  RX_CNT <= 0;
                  RX_FLG <= 1;
          end
          else begin
                  RX_CNT <= RX_CNT + 1;
                  RX_FLG <= 0;
          end
          if( TX_CNT == bps ) begin// 38400bps
                  TX_CNT <= 0;
```

```
//                          TX_FLG <= 1;
                    TX_FLG <= TX_FLG ^ 1;
            end
            else begin
                    TX_CNT <= TX_CNT + 1;
//                          TX_FLG <= 0;
            end
    end
end

assign RX_CK = RX_FLG;
assign TX_CK = TX_FLG;

endmodule
```

8.4 Send_Data_Slect.v（U2 Module）

```
////////////////////////////////////////////////////////////////////////////////
module Send_Data_Slect(Data_Select, SEL, Renge_Data, Send_Data);
  input [5:0]Data_Select;
  input SEL;
  input [15:0]Renge_Data;
  output [7:0]Send_Data;

  assign Send_Data = Tx_data(SEL);

  // RS232 port transmit data select Bin
  function [7:0]Tx_data ;
  input select ;
          case (select)
                  1'b0:    Tx_data = Tx_bin(Data_Select) ;  // Binary Data
                  1'b1:    Tx_data = Tx_dec(Data_Select) ;  // Decimal Data
          endcase
  endfunction

  // RS232 port transmit data select Bin
  function [7:0]Tx_bin ;
  input [5:0]in_data ;
          case (in_data)
                  6'b111110:       Tx_bin = 8'h42 ; // B
                  6'b111101:       Tx_bin = {4'b0000,Renge_Data[15:12]} ;    //
                  6'b111011:       Tx_bin = {4'b0000,Renge_Data[11:8]} ;     //
                  6'b110111:       Tx_bin = {4'b0000,Renge_Data[7:4]} ;      //
                  6'b101111:       Tx_bin = {4'b0000,Renge_Data[3:0]} ;      //
                  6'b011111:       Tx_bin = 8'h45 ; // E
          endcase
  endfunction

  // RS232 port transmit data select Dec
  function [7:0]Tx_dec ;
  input [5:0]in_data ;
          case (in_data)
                  6'b111110:       Tx_dec = 8'h44 ; // D
                  6'b111101:       Tx_dec = {4'b0011,Renge_Data[15:12]} ;    // 1000mm
                  6'b111011:       Tx_dec = {4'b0011,Renge_Data[11:8]} ;     // 10mm
                  6'b110111:       Tx_dec = {4'b0011,Renge_Data[7:4]} ;      // 10mm
                  6'b101111:       Tx_dec = {4'b0011,Renge_Data[3:0]} ;      // 1mm
                  6'b011111:       Tx_dec = 8'h45 ; // E
          endcase
  endfunction

endmodule
```

8.5 RS232C_TX.v（U3 Module）

```
////////////////////////////////////////////////////////////////////////////////
/***                  */
/**    RS232C_TX     **/
/*   d8 pn s1 xoff  ***/
module RS232C_TX( TX_CK, RST, TX_EN, WR, PD_IN, SD_OUT ) ;
  input  TX_CK;
  input  RST;
  input  TX_EN;
  input  WR;
  input  [7:0] PD_IN;      // ﾊﾟﾗﾚﾙﾃﾞｰﾀ入力
  output SD_OUT;           // ｼﾘｱﾙﾃﾞｰﾀ出力

  reg    SD_OUT;

  reg    LOAD;
  reg    [7:0] SERIAL;
  reg    [3:0] CBIT;

  always @( posedge TX_CK ) begin
          if( RST ) begin
                  SD_OUT <= 1'b1;
                  CBIT <= 4'h0;
                  LOAD  <= 1'b0;
          end
          else begin
                  if( WR ) begin
                          LOAD   <= 1'b1;
                          SERIAL <= PD_IN;
                  end
                  else begin
                          if( TX_EN && LOAD ) begin
                                  case( CBIT )
                                          // ｽﾀｰﾄﾋﾞｯﾄの送出(0)
                                          0 :
                                          begin
                                                  SD_OUT <= 1'b0;
                                                  CBIT <= CBIT + 4'h1;
                                          end

                                          // ﾃﾞｰﾀﾋﾞｯﾄの送出
                                          1,2,3,4,5,6,7,8 :
                                          begin
                                                  SD_OUT   <= SERIAL[0];
                                                  SERIAL <= { 1'b1, SERIAL[7:1]};
                                                  CBIT   <= CBIT + 4'h1;
                                          end
                                          // ｽﾄｯﾌﾟﾋﾞｯﾄの送出(1bit 目)
                                          9 :
```

```
                                        begin
                                                SD_OUT  <= 1'b1;
                                                CBIT    <= CBIT + 4'h1;
//                                                      LOAD  <= 1'b0;
                                        end
//        ｽﾄｯﾌﾟﾋﾞｯﾄが 1 ﾋﾞｯﾄの場合以下をｺﾒﾝﾄ
//////////////////////////////////////////
//                                              // ｽﾄｯﾌﾟﾋﾞｯﾄの送出(2bit 目)
                                        10 :
                                        begin
                                                SD_OUT  <= 1'b1;
                                                CBIT  <= 4'h0;
                                                LOAD  <= 1'b0;
                                        end
//////////////////////////////////////////
                                        default :
                                        begin
                                                SD_OUT <= 1'b1;
                                                CBIT <= 4'h0;
                                        end
                                endcase
                        end     //      if (TX_EN && LOAD)
                end     //end else (WR)
        end     // end else (RST)
  end     // end always

endmodule
```

8.6 Gen_1mm_Pulse.v（U4 Module）

```
////////////////////////////////////////////////////////////////////////////////
module Gen_1mm_Pulse(CLK33M, pls_1mm);
  input CLK33M;
  output pls_1mm;

  reg [7:0]count_1mm = 0;
  reg count_1mm_flg = 0;

  assign pls_1mm = count_1mm_flg;

  ////////// Generating 1mm Pulse(33.333MHz -> 99.799KHz(173.609375KHz))
  // １ミリメール　パルスジェネレータ
  always @(posedge CLK33M)
  begin
          if(count_1mm == 93) begin         // 2.78 usec ->  5.57 usec / 1mm
                  count_1mm_flg = count_1mm_flg ^ 1;
                  count_1mm <= 0;
          end
          else begin
                  count_1mm <= count_1mm + 1;
          end
  end
endmodule
```

8.7 Count_Renge.v（U5 Module）

```
////////////////////////////////////////////////////////////////////////////////
module Count_Renge(check_counter, RST, DATA_IN_EN, PLS_1mm, renge_dec, renge_bin);
  input [20:0]check_counter;
  input RST;
  input DATA_IN_EN;
  input PLS_1mm;
  output [15:0]renge_dec;
  output [15:0]renge_bin;

  reg [15:0]data_count_Bin = 16'h0000;      // Range data counter(binary)
  reg [15:0]data_count_Dec = 16'h0000;      // Range data counter(decimal)

  assign renge_dec = data_count_Dec;
  assign renge_bin = data_count_Bin;

//////////// Count_Renge     超音波距離計計測パルスカウンタ
  always @(posedge PLS_1mm)
  begin
          // Count Enable //
          if(check_counter > 8000) begin

                  // Count Start //
                  if(DATA_IN_EN == 1'b1) begin
                          data_count_Bin <= data_count_Bin + 1;

                          // column-1 //
                          if(data_count_Dec[3:0] == 4'h9) begin
                                  data_count_Dec[3:0] <= 4'h0;
                                  data_count_Dec[7:4] <= data_count_Dec[7:4] + 4'h1;

                                  // column-2 //
                                  if(data_count_Dec[7:4] == 4'h9) begin
                                          data_count_Dec[7:4] <= 4'h0;
                                          data_count_Dec[11:8] <= data_count_Dec[11:8] +
4'h1;

                                          // column-3 //
                                          if(data_count_Dec[11:8] == 4'h9) begin
                                                  data_count_Dec[11:8] <= 4'h0;
                                                  data_count_Dec[15:12]              <=
data_count_Dec[15:12] + 4'h1;

                                                  // column-4 //
                                                  if(data_count_Dec[15:12] == 4'h9) begin
                                                          data_count_Dec[15:12] <= 4'h0;
                                                  end
                                                  else begin
                                                          data_count_Dec[15:12]      <=
```

```
data_count_Dec[15:12] + 4'h1;
                                                        end
                                                        //4-4-4-4-4//

                                                end
                                                else begin
                                                        data_count_Dec[11:8]                <=
data_count_Dec[11:8] + 4'h1;
                                                end
                                                //3-3-3-3-3//

                                        end
                                        else begin
                                                data_count_Dec[7:4]  <=  data_count_Dec[7:4]  +
4'h1;
                                        end
                                        //2-2-2-2-2//

                                end
                                else begin
                                        data_count_Dec[3:0] <= data_count_Dec[3:0] + 4'h1;
                                end
                                //1-1-1-1-1//

                        end
                end

                // count disable //
                if(RST == 1) begin
                        data_count_Bin <= 0;
                        data_count_Dec <= 16'h0000;
                end
    end
  endmodule
```

8.8 Disp_Counter.v（U6 Module）

```
////////////////////////////////////////////////////////////////////////////////
module Disp_Counter(SelCLK, Count_Data, Seg_Data, Sel);
  input SelCLK;
  input [15:0]Count_Data;
  output [8:1]Seg_Data;
  output [4:1]Sel;

  reg [4:1]com_select = 4'b1110;

  function [3:0]col ;
  input [3:0]sel_data ;
          case (sel_data)
                  4'b1110:col = Count_Data[3:0] ;
                  4'b1101:col = Count_Data[7:4] ;
                  4'b1011:col = Count_Data[11:8] ;
                  4'b0111:col = Count_Data[15:12] ;
          endcase
  endfunction

  function [7:0]dec ;
  input [3:0]in_data ;
          case (in_data)
                  4'b0000:dec = 8'b00111111 ;      // 0
                  4'b0001:dec = 8'b00000110 ;      // 1.
                  4'b0010:dec = 8'b01011011 ;      // 2
                  4'b0011:dec = 8'b01001111 ;      // 3.
                  4'b0100:dec = 8'b01100110 ;      // 4
                  4'b0101:dec = 8'b01101101 ;      // 5.
                  4'b0110:dec = 8'b01111101 ;      // 6
                  4'b0111:dec = 8'b00100111 ;      // 7.
                  4'b1000:dec = 8'b01111111 ;      // 8
                  4'b1001:dec = 8'b01100111 ;      // 9.
                  4'b1010:dec = 8'b01110111 ;      // A.
                  4'b1011:dec = 8'b01111100 ;      // b.
                  4'b1100:dec = 8'b00111001 ;      // C.
                  4'b1101:dec = 8'b01011110 ;      // d.
                  4'b1110:dec = 8'b01111001 ;      // E.
                  4'b1111:dec = 8'b01110001 ;      // F.
          endcase
  endfunction

  assign Sel = com_select;
  assign Seg_Data = ~dec(col(com_select[4:1])) ;

  always @(posedge SelCLK)begin
                  com_select[2] <= com_select[1];
                  com_select[3] <= com_select[2];
                  com_select[4] <= com_select[3];
```

```
                com_select[1] <= com_select[4];
    end
endmodule
```

8.9 RangFinderForAVES.ucf（ピンアサイン）

```
#////////////////////////////////////////////////////////////////////////////////
#// Mastar Clock 33.333MHz
NET "CLK"                   LOC = "P63";

#// Input Transmit Data Select
#NET "S1"           LOC = "P35"; # S1
#NET "S2"           LOC = "P34"; # S2
#NET "S3"           LOC = "P33"; # S3
#NET "S4"           LOC = "P32"; # S4
#NET "S5"           LOC = "P30"; # S5
#NET "S6"           LOC = "P27"; # S6
#NET "S7"           LOC = "P26"; # S7
NET "TX_SELECT"    LOC = "P24"; # S8

#// Ultra Sonic Sencer Enable Triger
NET "TRG_EN"                LOC = "P67";
NET "TRG"                   LOC = "P68";

#// Ultra Sonic Sencer Enable Data
NET "DATA_IN"               LOC = "P66";
NET "DATA_EN"               LOC = "P65";

#// Ultra Sonic Sencer Pulse Monitor
NET "TP_TRG"                LOC = "P57";
NET "PLS_1mm"               LOC = "P60";

NET "LED<8>"                LOC = "P36";
NET "LED<7>"                LOC = "P40";
NET "LED<6>"                LOC = "P41";
NET "LED<5>"                LOC = "P47";
NET "LED<4>"                LOC = "P48";
NET "LED<3>"                LOC = "P49";
NET "LED<2>"                LOC = "P53";
NET "LED<1>"                LOC = "P54";

#// 7segmant LED Select(common)
NET "COM<1>"                LOC = "P78";     #comD
NET "COM<2>"                LOC = "P79";     #comC
NET "COM<3>"                LOC = "P83";     #comB
NET "COM<4>"                LOC = "P84";     #comA

#// 7segmant LED Data
NET "SEG7<1>"               LOC = "P85";     # seg a
NET "SEG7<2>"               LOC = "P90";     # seg b
NET "SEG7<3>"               LOC = "P92";     # seg c
NET "SEG7<4>"               LOC = "P95";     # seg d
NET "SEG7<5>"               LOC = "P98";     # seg e
NET "SEG7<6>"               LOC = "P86";     # seg f
```

```
NET "SEG7<7>"              LOC = "P91";     # seg g
NET "SEG7<8>"              LOC = "P94";     # seg dp

#// RS232C Tx Rx Data
NET "RxD"                  LOC = "P71";     #RS232C Rx DATA
NET "TxD"                  LOC = "P70";     #RS232C Tx DATA
```

平成 24 年度文部科学省委託　「東日本大震災からの復興を担う専門人材育成支援事業」
東北の復興を担う自動車組込みエンジニア育成支援プロジェクト

平成 24 年度文部科学省委託
「東日本大震災からの復興を担う専門人材育成支援事業」
東北の復興を担う自動車組込みエンジニア育成支援プロジェクト

実践！自動車組込み技術者入門
ＦＰＧＡとマイコンの連携システム　ハードウェア編

平成 25 年 3 月

学校法人日本コンピュータ学園（東北電子専門学校）
〒980-0013　宮城県仙台市青葉区花京院一丁目３番１号

問合せ先　有限会社ザ・ライスマウンド
〒164-0003　東京都中野区東中野 1-57-8　辻沢ビル 3F
電話：03-5332-5080　FAX　03-5332-5083