平平成２５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

実践！自動車組込み技術者講座

ＦＰＧＡとマイコンの連携システム
（ハードウェア応用編）

~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推進協議会

ＴＡＢＬＥ　ＯＦ　ＣＯＮＴＥＮＴＳ

０．はじめに

本書は、平成２５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」で作成しました。

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」で作成された、
「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」
を、基礎編と呼び、本書を応用編と呼ぶ事とします。

基礎編で設計した**超音波距離計を使用し**、応用編ではその計測データを送信する方法として、CAN（Controller Area Network）を使用します。
CAN は車載システムでは欠かせないネットワーク規格の一つです。

応用編では、CAN を使った通信に重点を絞り学習しますので、本書の構成は基礎編で学んだ項目と極力重複しないようにしています。ただし、応用編の説明で必要と思われる項目は、基礎編から抜粋して再度学習します。

第１章「**FPGA 学習ボード**」では、学習用 FPGA ボードキットの製作手順を説明します。

第２章「**FPGA 開発環境**」では、ザイリンクス社の FPGA 開発ツール ISE（ISE はザイリンクス社の登録商標です。）の使い方を中心に FPGA の開発環境を説明します。

第３章「FPGA 搭載機能」では、応用編の FPGA 搭載機能について説明します。

第４章「CAN ってなんだ」では、CAN の規格について説明します。

第５章「SPI ってなんだ」では、CAN バスの制御をする CAN コントローラとのインタフェースに使用する、SPI（Serial Peripheral Interface）の規格について説明します。

第６章「CAN コントローラを使ってみよう」では、VerilogHDL で記述した FPGA のソースコードから、CAN コントローラとのインタフェースについて説明します。

本書が、基礎編に続き、車載システム開発の FPGA とマイコンの連携システムを始めようという、技術者、学生の良き参考書となれば幸いです。

1．　FPGA 学習ボード

FPGA 学習ボード概要

　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、はんだ付けして自分で組み立てます。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。

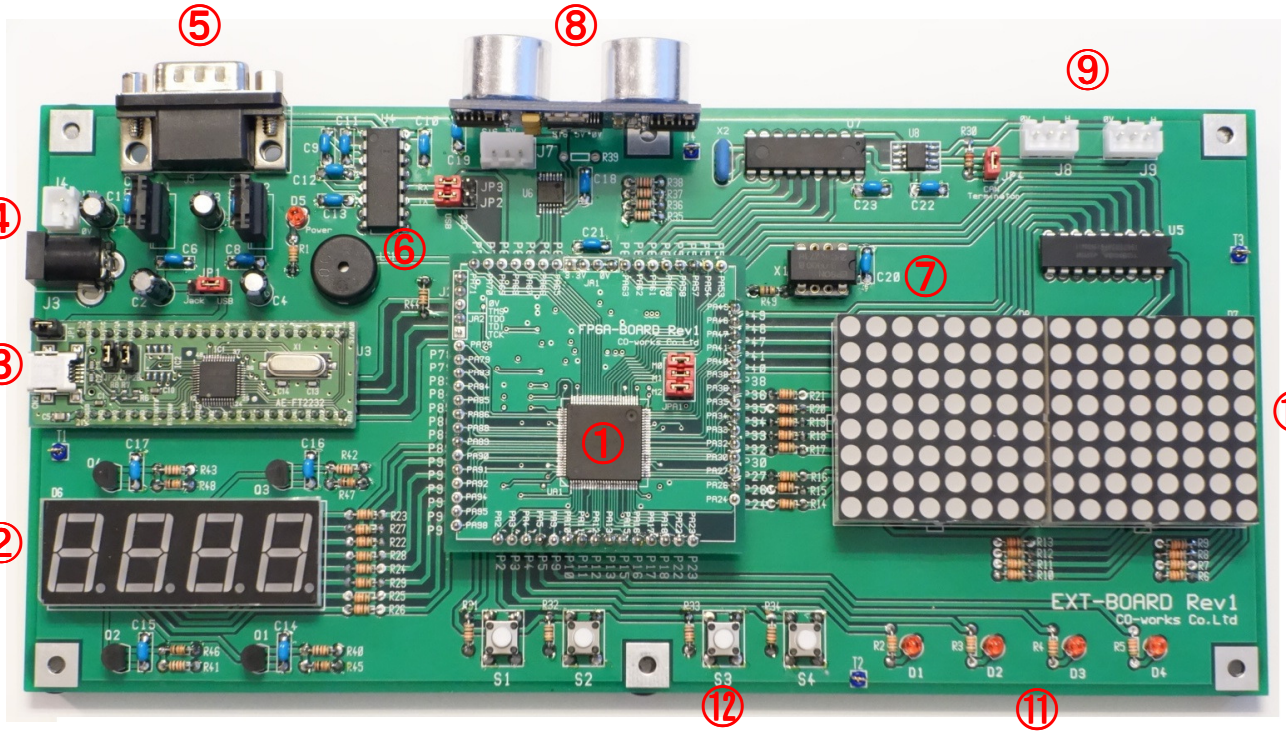

図 x　FPGA 学習ボード

各部の機能説明

①　FPGA-BOARD

FPGA-BOARD は、FPGA とその周辺回路が搭載された独立した基板になっており、取り外しが可能です。図 x に FPGA-BOARD の回路図を示します。回路図中の各部品の機能は、下記のようになっています。

UA1：ilinx 社製の XC3S100E-4VQG100C です。この FPGA は、100,000 個のゲートを内蔵しています。

UA2：FPGA 用のコンフィギュレーション ROM です。FPGA 内のコンフィギュレーションデータは、揮発性メモリであるため、電源を OFF にすると消えてしまします。そのため、電源を OFF にしても消えないフラッシュメモリである UA2 に書き込みます。FPGA は電源 ON 時に自動的に UA2 からコンフィギュレーションデータを読み取ります。

UA3：電圧のレベル変換を行っています。JA2 側の 3.3V から FPGA の JTAG インターフェイスの定格電圧である 2.5V への変換及び、その逆を行っています。

JPA1：FPGA の動作モードを設定するジャンパピンです。電源投入後のコンフィギュレーション方法や、I/O ピンの動作について設定ができます。本教材中では、全ピンをショートさせた状態で使用します。

UA4：1.2V 出力の三端子レギュレータです。3.3V から FPGA 内部で使用する 1.2V を生成します。

UA5：2.5V 出力の三端子レギュレータです。3.3V から FPGA 及びコンフィギュレーション ROM で使用する 2.5V を生成します。

JA1：電源用コネクタです。3.3V を受電します。

JA2：JTAG コネクタです。TCK（クロック）・TDI（データ入力）・TDO（データ出力）・TMS（状態制御）の 4 本の信号線を使用して、FPGA やコンフィギュレーション ROM と通信を行います。また、FPGA とコンフィギュレーション ROM は JTAG バス上で直列に接続されています。

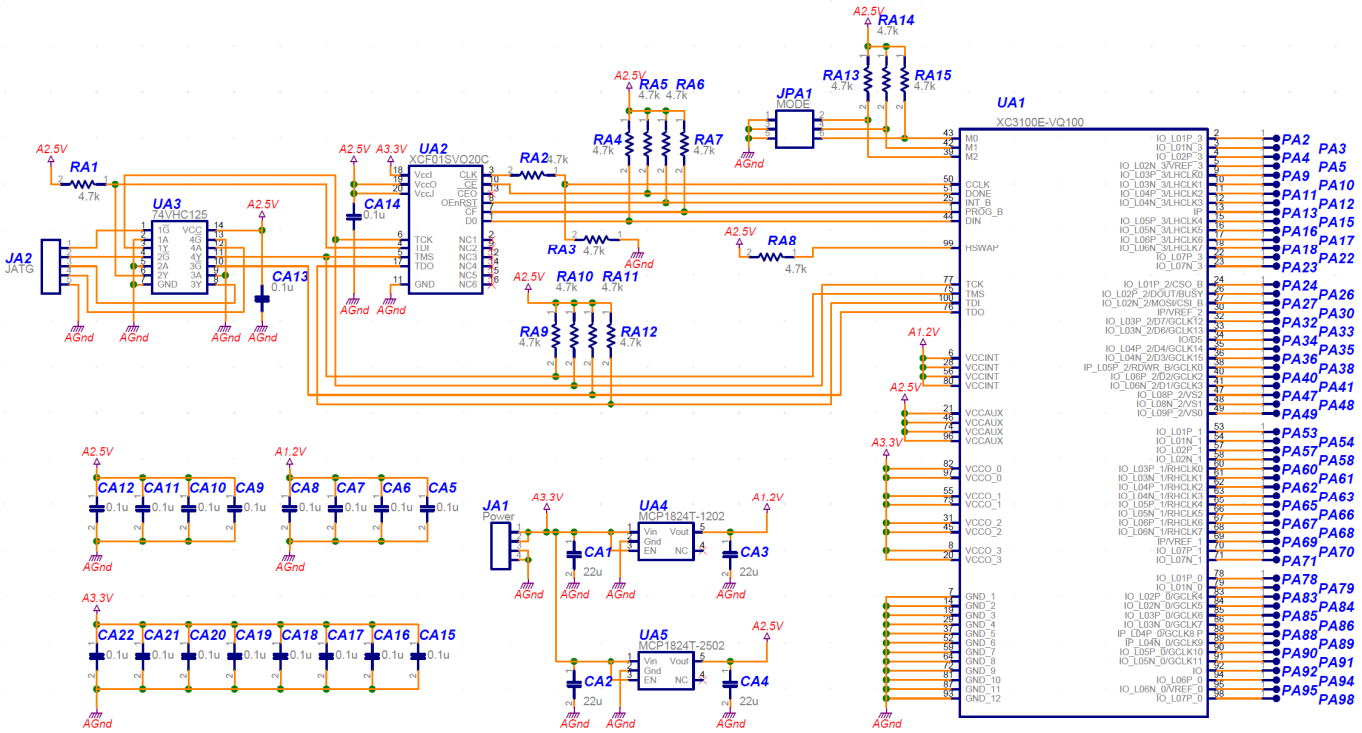

図１．１　FPGA-BOARD 回路図

② 緑色 7 セグメント LED

D6 の OSL04562 が、7 セグメント LED で、アノード（+）コモン接続です。

アノードは、それぞれの桁ごとにコモン（共通）になっており、1 コモンあたり、最大で 8 セグメント分の電流を制御します。そのため、FPGA の I/O では駆動できませんので、Q1～Q4 の各トランジスタにより ON/OFF し、このトランジスタの制御を FPGA の I/O から行います。

また、P85～P98 に接続されているセグメントは、全桁共通となっています。そのため、各桁に異なる表示を行うためには、ダイナミック点灯と呼ばる制御を行う必要があります。

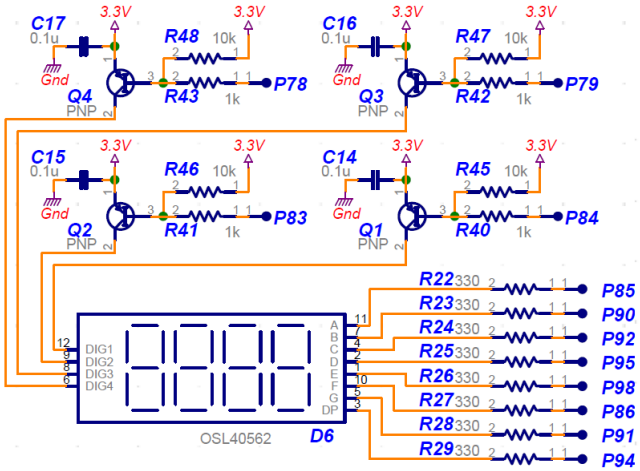

図１．２　7 セグメント LED 周辺回路

③ USB-JTAG

U3 は、秋月電子通商製の「FT2232D USB-シリアル 2ch 変換モジュール」です。FTDI 社製の FT2232D チップを搭載しており、PC と USB で接続されます。FT2232D には、USB シリアル変換機能と、独自の I/O 制御機能が搭載されており、これにより、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。

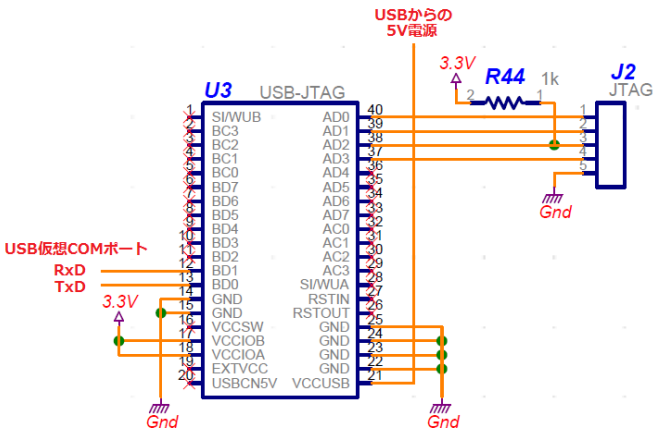

図１．３　USB-JTAG 周辺回路

④ 電源

J3 は、2.1mm センタープラスの DC ジャック、J4 は、2 極コネクタで、いずれからでも電源を入力できます。DC ジャックは、室内や実験中に、2 極コネクタは、機器への組込みや時などコネクタが抜けては困る場合に使用すると便利です。電源の入力範囲は、6V～12V です。

U1 は、5V 出力の三端子レギュレータ、U2 は、3.3V 出力の三端子レギュレータです。本ボードの多くの部分は、3.3V で駆動しますが、超音波センサのみ 5V で駆動します。

JP1 は、電源ソース選択用のジャンパピンで、J3・J4 コネクタ入力もしくは、USB バスパワー入力のいずれかを選択します。D5 は、パワーランプ LED で、本ボードに電源が供給されている限り点灯します。

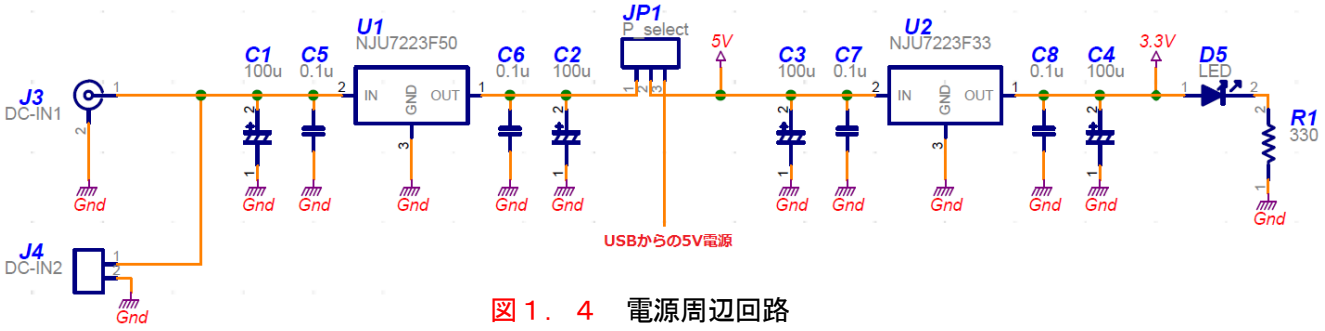

図１．４　電源周辺回路

⑤ シリアル通信

本ボードでは、シリアル通信を Dsub9 ピンオスコネクタによる RS-232C 通信もしくは、USB による仮想 COM ポート通信いずれかを切り替えて行うことができます。切り替えは、本ボードから見て送信を JP2、受信を JP3 のシャンパーピンで選択します。

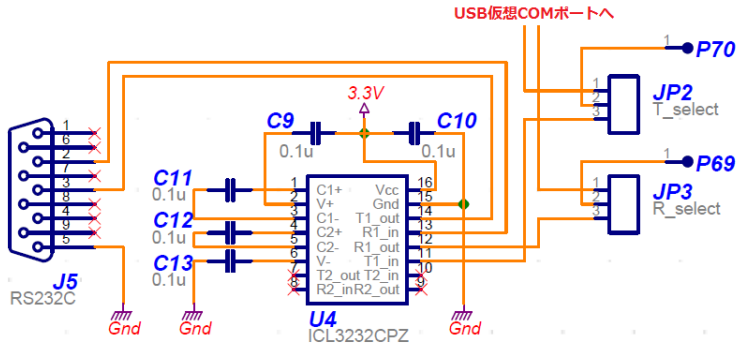

図１．５　シリアル通信周辺回路

RS-232C の場合は、U4 の RS-232C トランシーバにより信号レベルを±5V に変換し、送信は、J5 の 3 番ピン、受信は J5 の 2 番ピンから行われます。USB 仮想 COM ポートの場合は、③USB-JTAG に接続されます。PC 側で仮想 COM ポートを使用する場合は、ドライバのインストールが必要です。また FPGA は、P70 に接続されるポートを送信に設定し、P69 を受信に設定します。

⑥ 圧電スピーカ

この圧電スピーカは、発振回路を内蔵していないため、発音させるには、周波数を与える必要があり、50~20KHz 程度まで発音できます。また、発音させない場合は、素子を保護するため P71 を L レベルにすることを推奨します。

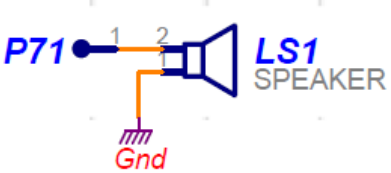

図１．６　圧電スピーカ回路

⑦ クリスタルオシレータ

クリスタルオシレータは、水晶発振子単体とはちがい、発振回路を内蔵しているため直接目的の周波数の信号が出力されます。本キットでは、33.333MHz のクリスタルオシレータを同梱していますが、他の周波数を使用することも可能です。その際は、エプソントヨコム製 SG-8002DC シリーズの 3.3V 品から選定して下さい。

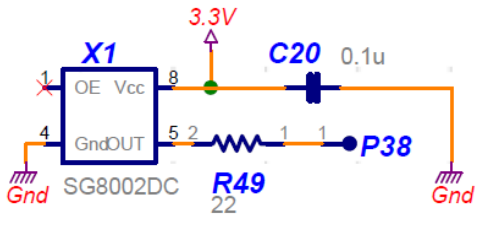

図１．７　クリスタルオシレータ回路

⑧ 超音波センサ

J6 は 3 ピンのピンソケットで Parallax 社製の超音波距離センサーモジュールをそのまま接続できます。J7 は 3 極コネクタで超音波距離センサーモジュールを延長ケーブル等で接続したい場合などに使用すると便利です。

Parallax 社製の超音波距離センサーモジュールは、電源 5V で 1 ワイヤーの信号線で動作します。FPGA は 3.3V で動作しているため、U6 の入力トレラント機能をもつスリーステートバッファでレベル変換を行っています。

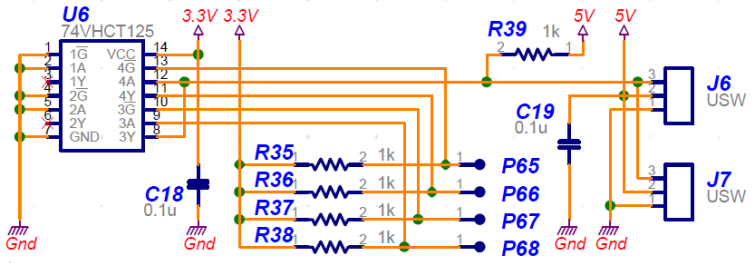

図１．８　超音波センサ周辺回路

FPGA と超音波距離センサーモジュール間で信号をやり取りするには、以下のようにします。

・USW へ H を出力する場合：　P67 を H にする(P65/P66/P68 の状態は不問)
・USW へ L を出力する場合：　P67/P68 を L にする(P65/P66 の状態は不問)
・USW の値を読み取る場合：P67 を H、P65 を L にした後 P66 の値を読み取る(P68 の状態は不問)
（通信仕様は超音波距離センサーモジュールのマニュアルを参照してください。）

また、R39 のプルアップ抵抗は、Parallax 社製の超音波距離センサーモジュールを使用する場合には未実装にします。それ以外のモジュールを接続する際には、値を適宜選定して実装してください。

⑨ CAN(Control Area Network)

本ボードには、自動車等に使用されている CAN(Control Area Network)のインターフェイスを搭載しています。CAN は、ドイツのボッシュ社が提唱した規格で現在は ISO として通信仕様や電気的仕様などが標準化されています。

U7：マイクロチップテクノロジ社製の MCP2515 というスタンドアロン型 CAN コントローラです。CAN のプロトコルは MCP2515 に実装されており、CAN データの送受信や、さらに受信データのフィルタリング等を自動で行ってくれるため､FPGA は CAN のプロトコルを実装する必要がありません｡FPGA と MCP2515 は、3 線式のシリアル通信である SPI を使用して通信を行います。MCP2515 を使用した CAN 通信の方法に関して詳細は、MCP2515 のデータシートを参照してください。

U8：CAN 通信用のトランシーバです。CAN では、2 本の信号線を用いて差動信号により情報を表現しています。そのための電気的な信号の変換ををを行う IC です。

JP4：終端抵抗選択用のジャンパピンです。CAN(ハイスピード規格)では、ノイズ低減のためバスの両端を 120Ω の抵抗で終端させることが定められています。本ボードが CAN バスの両端に当たるよう接続されている場合のみジャンパピンを用いて JP4 をショートさせて下さい。

J8/J9：外部との接続用コネクタです。J8 と J9 は並列に接続されているため、どちらでも使用可能です。CAN は、1 本のバスに複数台の機器を接続して使用することが多いため、本ボードを複数台用いて実験を行う場合などは、2 つのコネクタを用いて数珠つなぎにすると便利です。

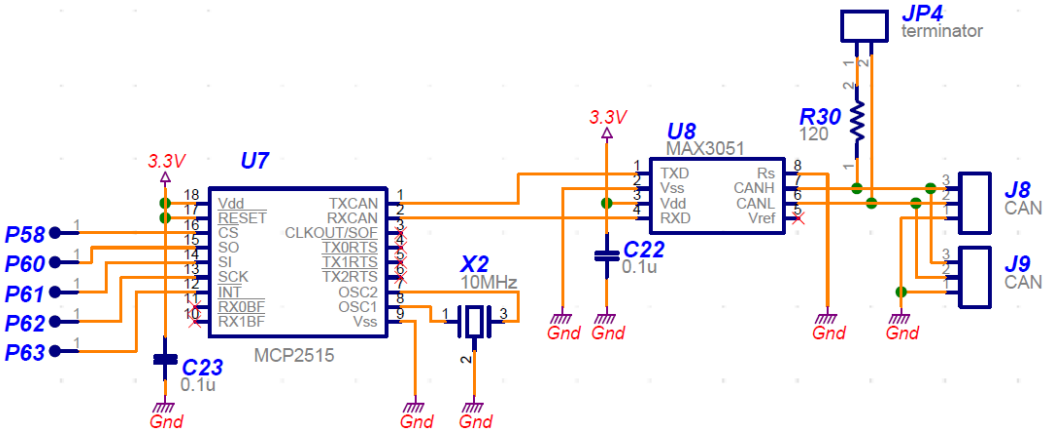

図１．９　CAN 周辺回路

⑩ ドットマトリクス LED

本ボードは、D7 と D8 に OSL641501-ARA というカソードコモンの 8x8 赤色ドットマトリクス LED を 2 個搭載しており、U5 はコモンをドライブするためのダーリントントランジスタアレイです。

このドットマトリクス LED は、図 x のようにピン番号と接続されているドットの位置が異なるため、制御をする際に注意が必要です。たとえば、D8 の一番左上を点灯させたい場合、FPGA の P34 と P53 を H にします。

また、コモン接続により横一列分 16 ドットのカソードが共通になっていますので、ドットマトリクス LED 全体に文字や絵を表示するにはダイナミック点灯と呼ばれる制御を行う必要があります。

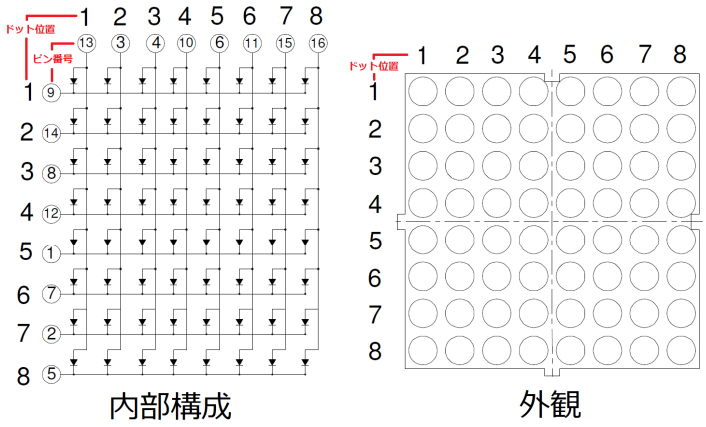

図 1.10　OSL641501-ARA 内部構成

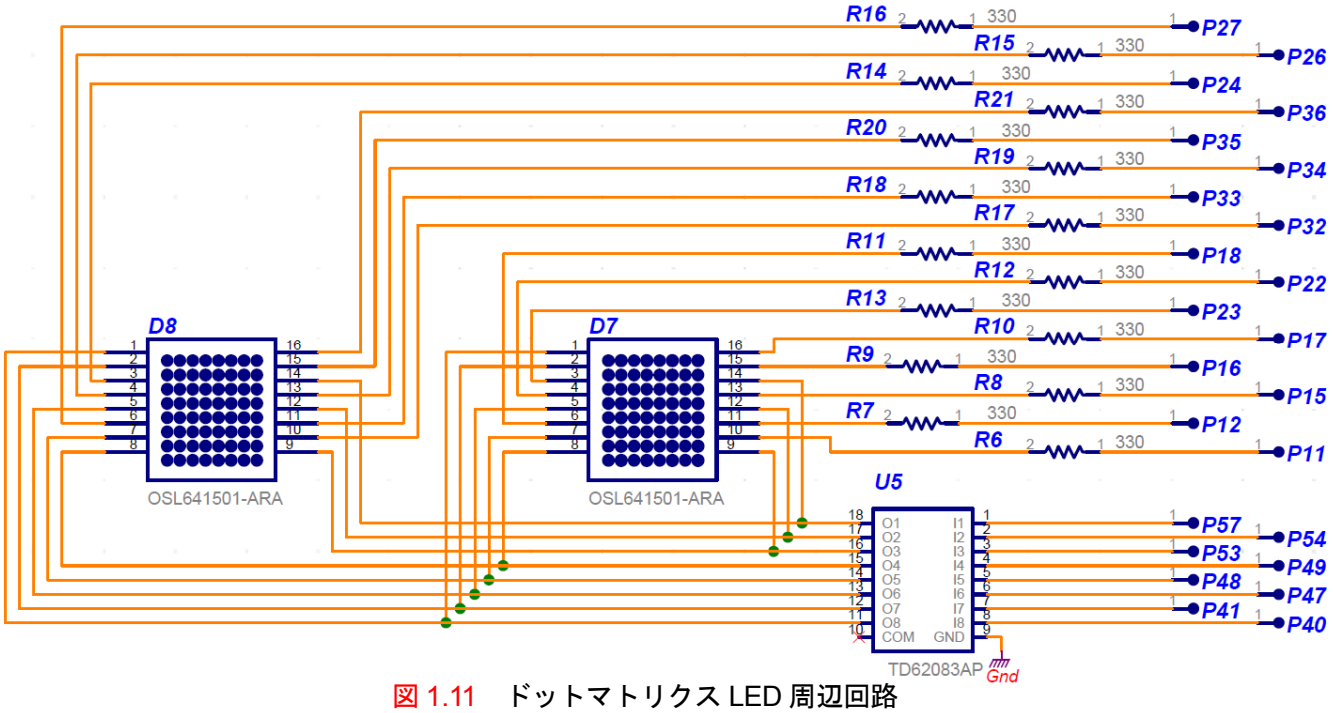

図 1.11　ドットマトリクス LED 周辺回路

⑪ LED

赤色φ3mm の LED です。アノードを 3.3V、カソードが FPGA に接続されているため、FPGA の I/O ピンを L にすると点灯します。

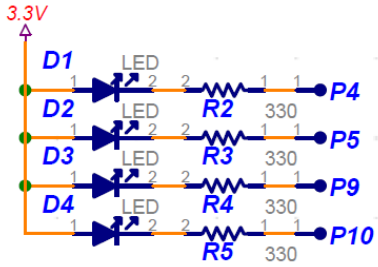

図 1.12　LED 周辺回路

⑫ スイッチ

タクトスイッチです。スイッチを押していない状態では、R31~R34 の抵抗により、プルアップされているため、FPGA の I/O ピンは H になります。スイッチを押すと、スイッチを介して FPGA の I/O ピンが GND に接続されるため、L になります。

また、タクトスイッチは押し下げ時にチャタリングが発生するため、FPGA でその対策が必要です。

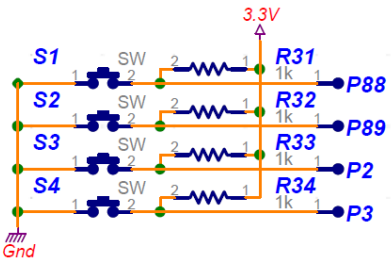

図 1.13　スイッチ周辺回路

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１．１と図 1.14 にキットに同梱されている部品一覧を示します。

表１．１　キット同梱部品一覧

数量	品名	型番	メーカ
1	5Vレギュレータ	NJU7223F50	JRC
1	3.3Vレギュレータ	NJU7223F33	JRC
1	USB-JTAG/シリアル変換	USBモジュールM-02990	秋月
1	RS-232Cトランシーバ	ICL3232CPZ	IntersilAmericasInc.
1	8chトランジスタアレイ	TD62083AP	東芝
1	CANコントローラ	MCP2515-I/P	マイクロチップ
4	PNPトランジスタ	2SA1015Y	東芝
5	赤色LED	OSDR3133A	OSL
1	緑4ケタ7セグメントLED	OSL40562-IG	OSL
2	赤8x8ドットマトリクスLED	OSL641501-ARA	OSL
29	330Ω 1/6Wカーボン抵抗	(橙橙茶金)	
1	120Ω 1/6Wカーボン抵抗	(茶赤茶金)	
14	1kΩ 1/6Wカーボン抵抗	(茶黒赤金)	
4	10kΩ 1/6Wカーボン抵抗	(茶黒橙金)	
1	47Ω 1/6Wカーボン抵抗	(黄紫黒金)	
4	100uF25V電解コンデンサ	25PK100MEFC5X11	ルビコン
19	0.1uF50V積層セラミック	FK28Y5V1H104ZN006	TDK
4	タクトスイッチ	DTS-6 白	COSLAND
1	33.333MHzオシレータ	SG8002DC-33.333MHz-PCB	EPSON
1	10MHz発信子	CSTLS10M0G53-B0	ムラタ
1	圧電スピーカ	PKM13EPYH4000-A0	ムラタ
4	テスト用GNDピン	SLC-2-G:青	サンハヤト
1	2.1mmDCジャック	MJ-179P	マル信無線
1	2ピンコネクタ	B2B-XH	日圧
3	3ピンコネクタ	B3B-XH	日圧
1	Dsub9コネクタ オス	C-00644	秋月
3	ピンヘッダ	C-00167	
2	ピンソケット	C-05779	
1	丸ピンソケット	P-01591	
1	8ピンICソケット	2227MC-08-03	
6	丸型プラ足	020-001-000	
7	ジャンパピン		
1	超音波センサ	M-05400	
1	USBケーブル	C-05222	
1	EXT-BOARD基板		
1	FPGA-BOARD基板		

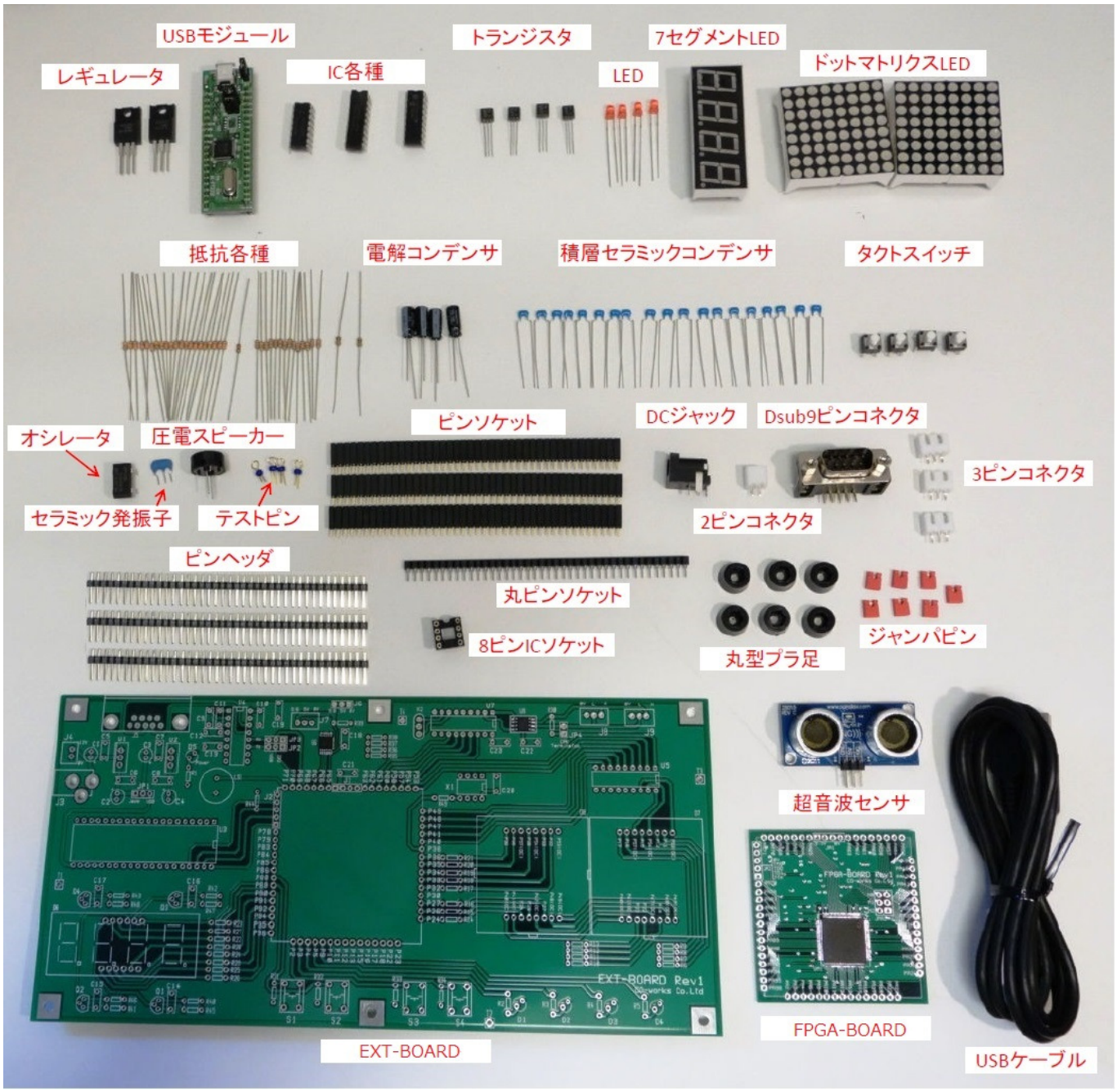

図 1.14　キット同梱部品一覧

事前準備

はんだ付け作業に入る前に、ピンヘッダ・ピンソケット・丸ピンソケットを折って分割します。

・40 ピンのピンヘッダ 3 本を、19 ピン x 2 個、15 ピン x 2 個、3 ピン x 5 個、2 ピン x 1 個に分割します

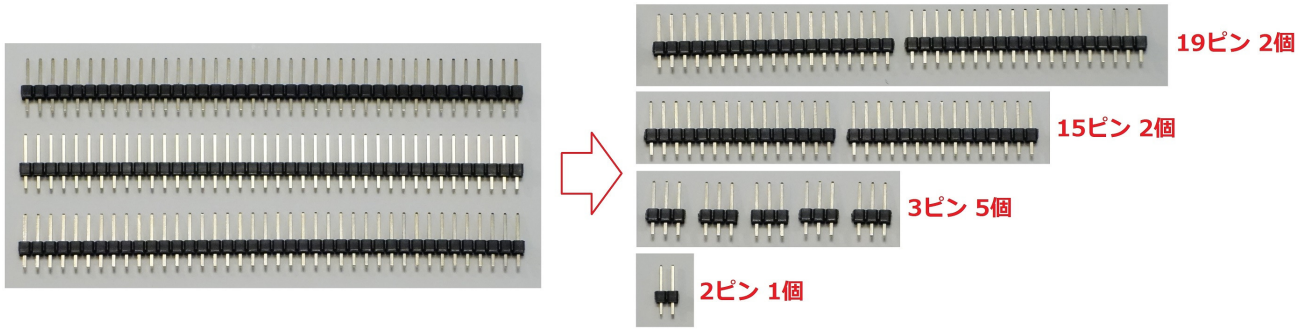

図 1.15　ピンヘッダ分割

・42 ピンのピンソケット 2 本を、19 ピン x 2 個、15 ピン x 2 個、3 ピン x 1 個に分割します。

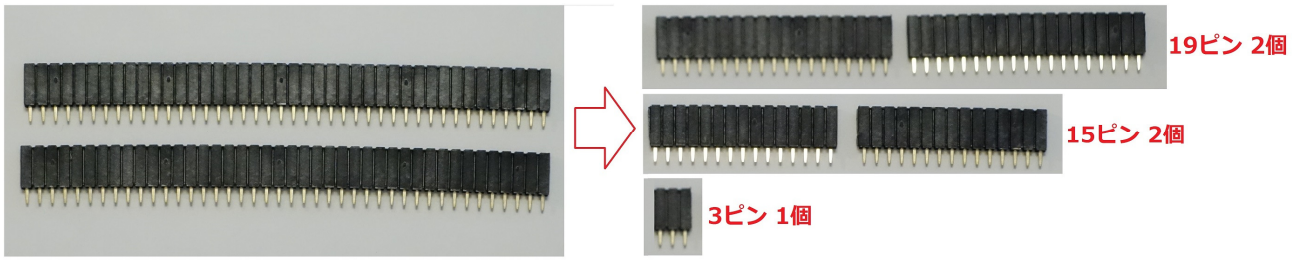

図 1.16　ピンソケット分割

・40 ピンの丸ピンソケット 1 本を、8 ピン x 4 個に分割します。

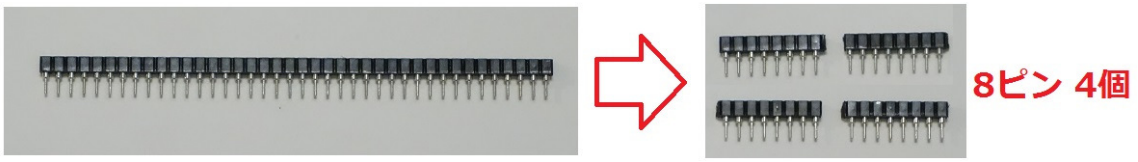

図 1.17　丸ピンソケット分割

はんだ付けの前に、

図 1.18 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。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。

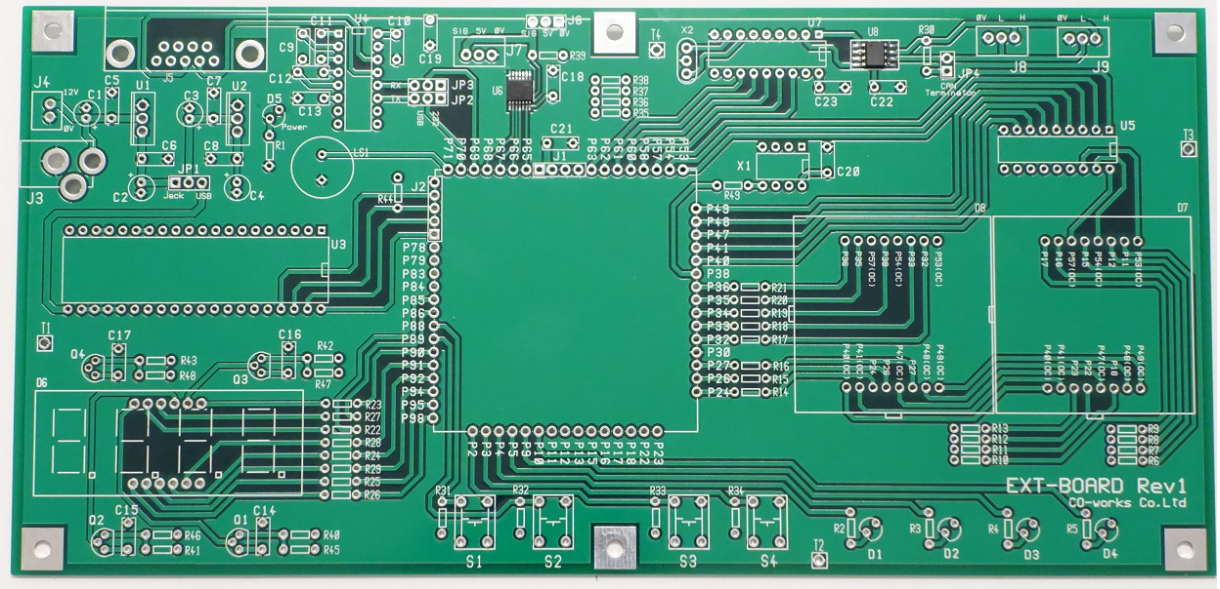

図 1.18　表面部品実装済みの EXT-BOARD

はんだ付けは、基本的に背の低い部品から順に行います。
以下の部品表に示す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１．２　EXT-BOARD 部品表

部品番号	数量	品名	型番	メーカ	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-JTAG/シリアル変換	USBモジュールM-02990	秋月	
U4	1	RS-232Cトランシーバ	ICL3232CPZ	IntersilAmericasInc.	
U5	1	8chトランジスタアレイ	TD62083AP	東芝	
U6	1	3ステートバッファ	TC74VHCT125AFT	東芝	表面実装済み
U7	1	CANコントローラ	MCP2515-I/P	マイクロチップ	
U8	1	CANトランシーバ	MAX3051ESA+	マキシム	表面実装済み
Q1-4	4	PNPトランジスタ	2SA1015Y	東芝	
D1-5	5	赤色LED	OSDR3133A	OSL	
D6	1	緑4ケタ7セグメントLED	OSL40562-IG	OSL	
D7-8	2	赤8x8ドットマトリクスLED	OSL641501-ARA	OSL	丸ピンソケット使用
R1-29	29	330Ω1/6Wカーボン抵抗			橙橙茶金
R30	1	120Ω 1/6Wカーボン抵抗			茶赤茶金
R31-44	14	1kΩ1/6Wカーボン抵抗			茶黒赤金
R45-48	4	10kΩ1/6Wカーボン抵抗			茶黒橙金
R49	1	47Ω1/6Wカーボン抵抗			黄紫黒金
C1-4	4	100uF25V電解コンデンサ	25PK100MEFC5X11	ルビコン	
C5-23	19	0.1uF50V積層セラミック	FK28Y5V1H104ZN006	TDK	
S1-4	4	タクトスイッチ	DTS-6 白	COSLAND	
X1	1	33.333MHzオシレータ	SG8002DC-33.333MHz-PCB	EPSON	ICソケット使用
X2	1	10MHz発信子	CSTLS10M0G53-B0	ムラタ	
LS1	1	圧電スピーカ	PKM13EPYH4000-A0	ムラタ	
T1-4	4	テスト用GNDピン	SLC-2-G:青	サンハヤト	
J1	1	4Px1ピンソケット	C-05779		分割して使用
J2	1	5Px1ピンソケット	C-05779		分割して使用
J3	1	2.1mmDCジャック	MJ-179P	マル信無線	
J4	1	2ピンコネクタ	B2B-XH	日圧	
J5	1	Dsub9コネクタ オス	C-00644	秋月	
J6	1	3Px1ピンソケット	C-05779		
J7-9	3	3ピンコネクタ	B3B-XH	日圧	
JP1-3	3	3Px1ピンヘッダ	C-00167		分割して使用
JP4	1	2Px1ピンヘッダ	C-00167		分割して使用
Px	59	ピンソケット	C-05779		分割して使用

表１．３　FPGA-BOARD 部品表

部品番号	数量	品名	型番	メーカ	備考
UA1	1	FPGA	XC3S100E-4VQG100C	Xilinx	表面実装済み
UA2	1	フラッシュメモリ	XCF01SVOG20C	Xilinx	表面実装済み
UA3	1	3ステートバッファ	TC74VHCT125AFK	東芝	表面実装済み
UA4	1	1.2Vレギュレータ	MCP1824T-1202E/OT	マイクロチップ	表面実装済み
UA5	1	2.5Vレギュレータ	MCP1824T-2502E/OT	マイクロチップ	表面実装済み
RA1,3	2	1.2kΩ	RK73B1JTTD122J	KOA	表面実装済み
RA2,5	2	470Ω	RK73B1JTTD471J	KOA	表面実装済み
RA4,6-15	11	4.7kΩ	RK73B1JTTD472J	KOA	表面実装済み
CA1-4	4	6.3V22uF	GRM21BB30J226M	MURATA	表面実装済み
CA5-22	18	0.1uF	GRM188B31H104K	MURATA	表面実装済み
JA1	1	4Px1ピンヘッダ	C-00167		分割して使用
JA2	1	5Px1ピンヘッダ	C-00167		分割して使用
JPA1	1	3Px2ピンヘッダ	C-00167		分割して使用
PBx	59	ピンヘッダ	C-00167		分割して使用

抵抗各種のはんだ付け

R1～R49 に抵抗を実装します。

極性はありません。どちら向きに実装しても大丈夫です。

※R39 は未実装にしてください。

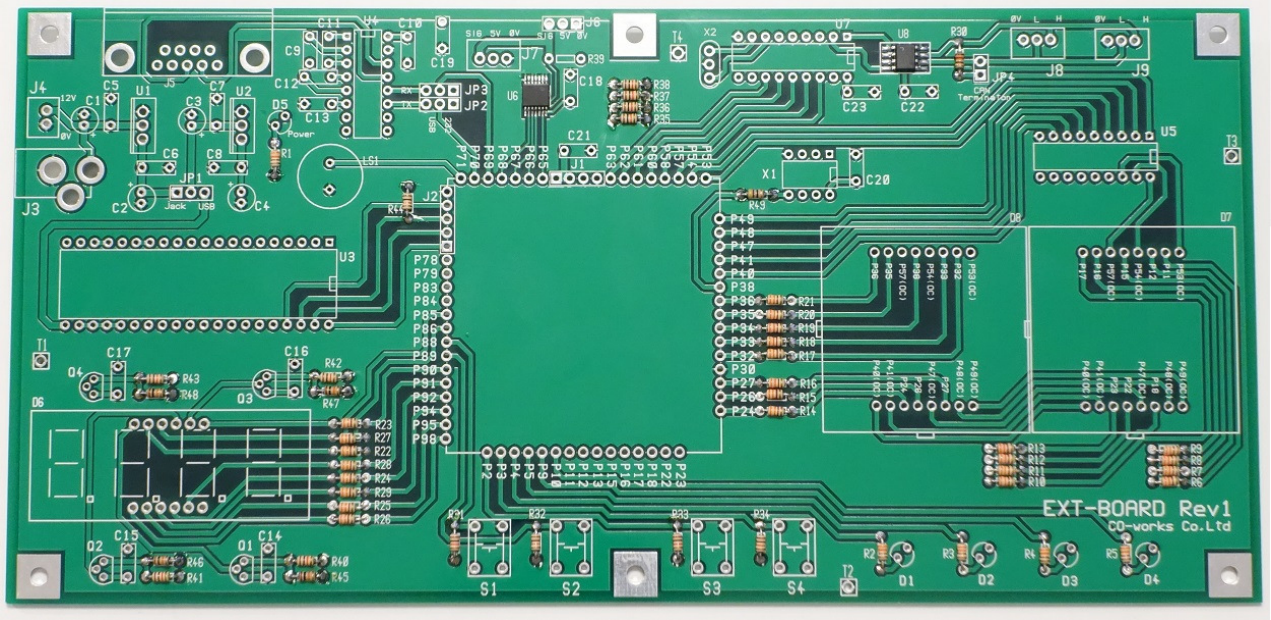

図 1.19　抵抗実装後の EXT-BOARD

積層セラミックコンデンサのはんだ付け

C5～C23 に積層セラミックコンデンサを実装します。

極性はありません。

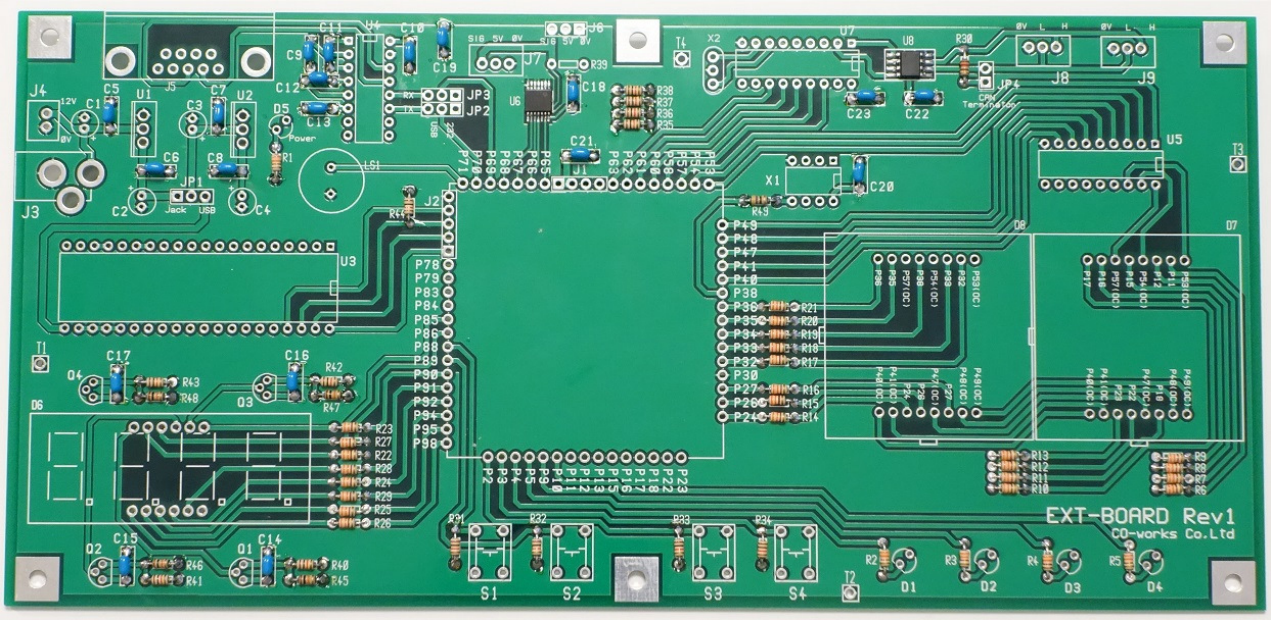

図 1.20　積層セラミックコンデンサ実装後の EXT-BOARD

丸ピンソケットのはんだ付け

D7～D8 に、事前に分割した、8 ピンの丸ピンソケット 4 個を実装します。

極性はありません。

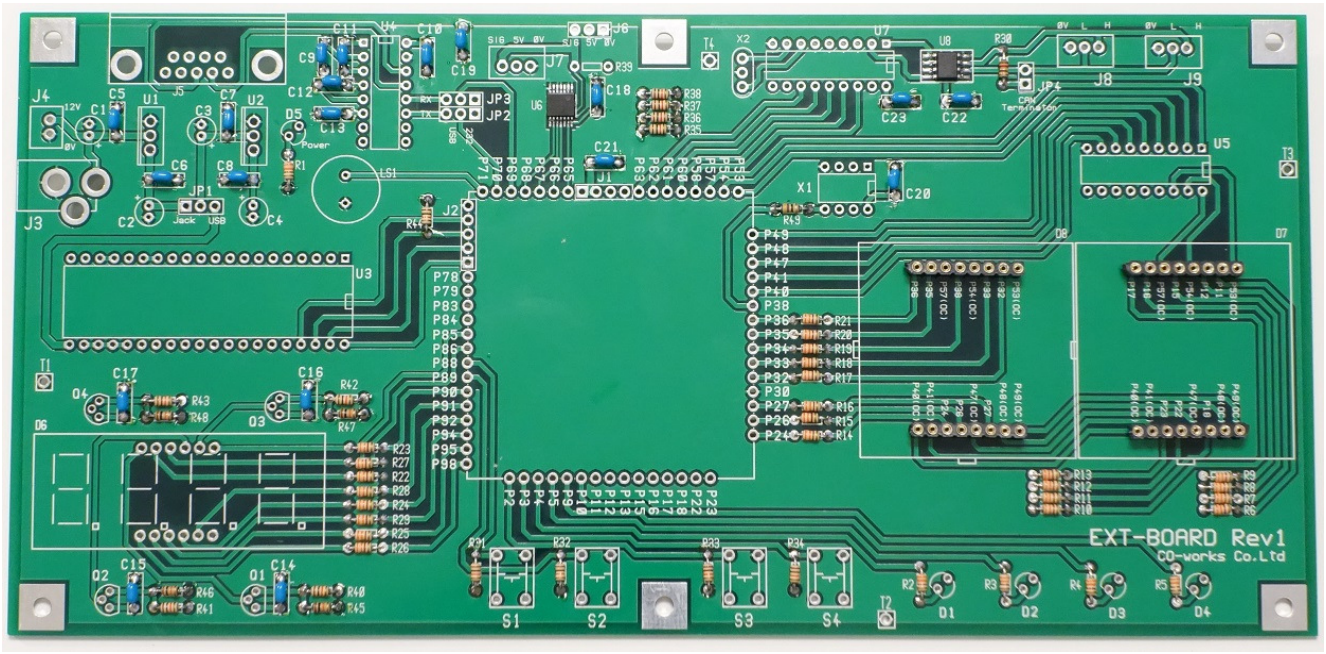

図 1.21　丸ピンソケット実装後の EXT-BOARD

テストピンのはんだ付け

T1～T4 にテストピンを実装します。

極性はありません。

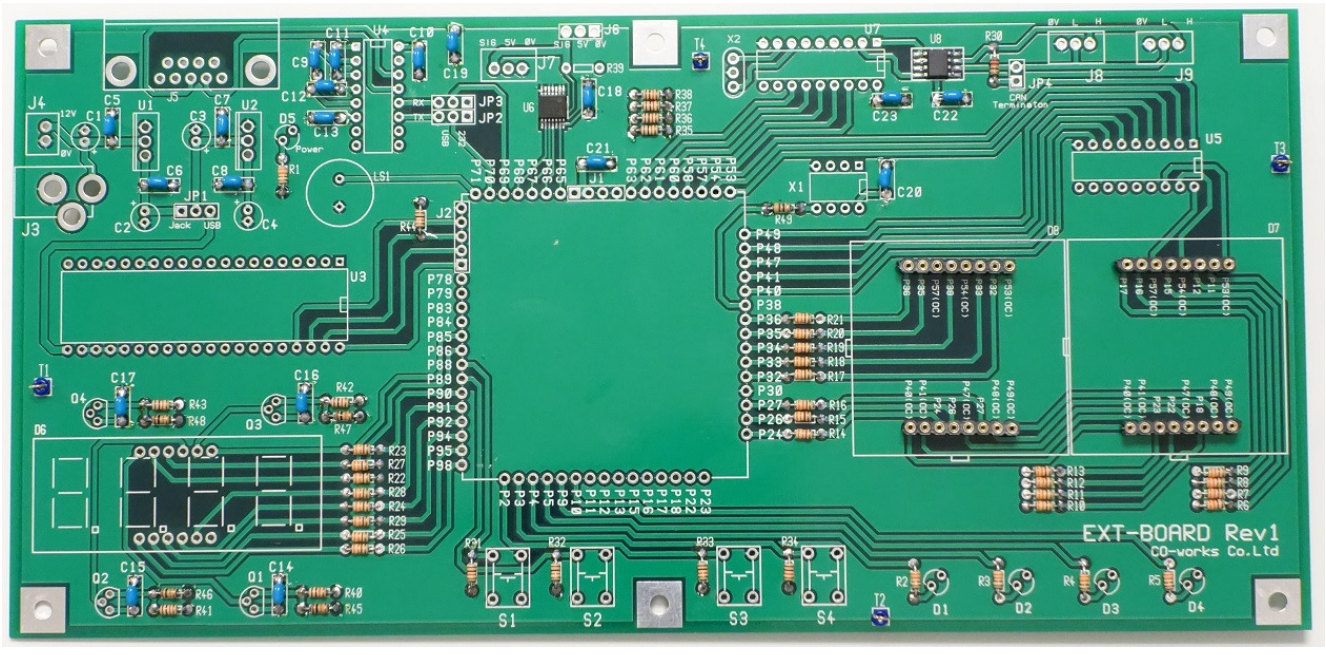

図 1.22　テストピン実装後の EXT-BOARD

8 ピン IC ソケットのはんだ付け

X1 に 8 ピン IC ソケットを実装します。

部品表で X1 は、33.333MHz オシレータとなっていますが、オシレータの交換が容易なように、IC ソケットを使用します。極性があります。図 x のように、切り欠きの方向を合わせます。

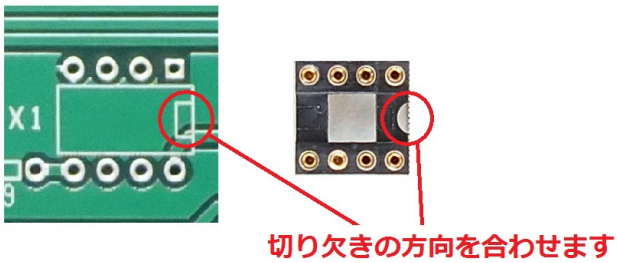

図 1.23　8 ピン IC ソケットの実装方向

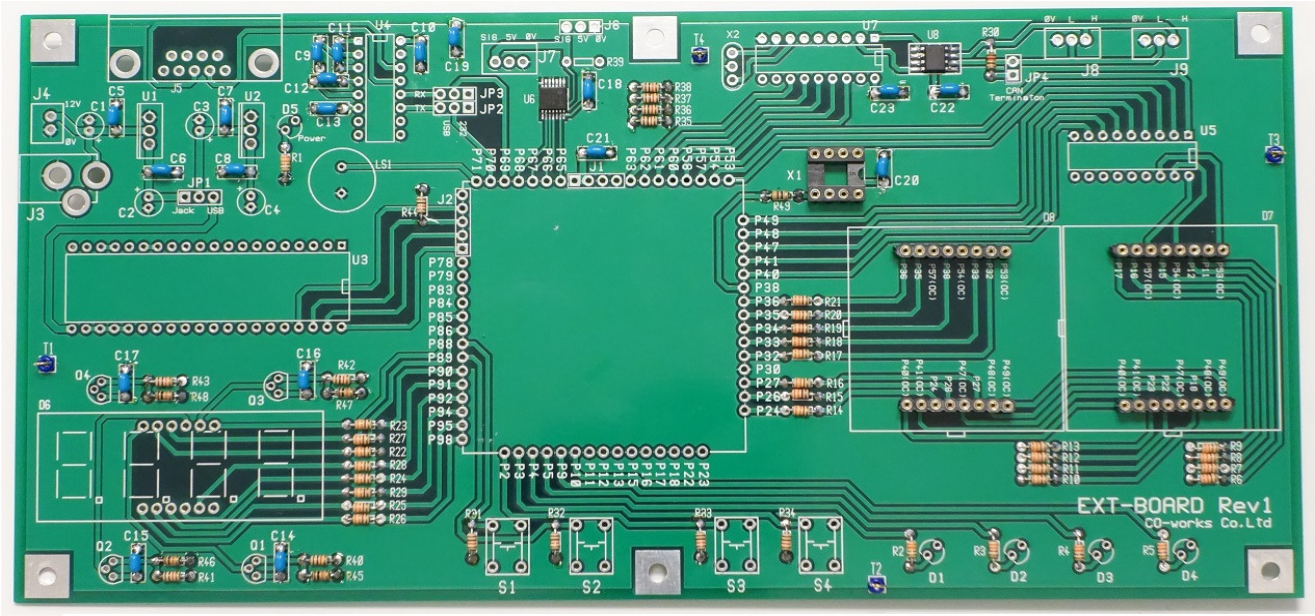

図 1.24　8 ピン IC ソケット実装後の EXT-BOARD

各種 IC のはんだ付け

U4・U5・U7 に IC を実装します。

極性があります。8 ピン IC ソケットと同じように切り欠きの方向を合わせて実装します。

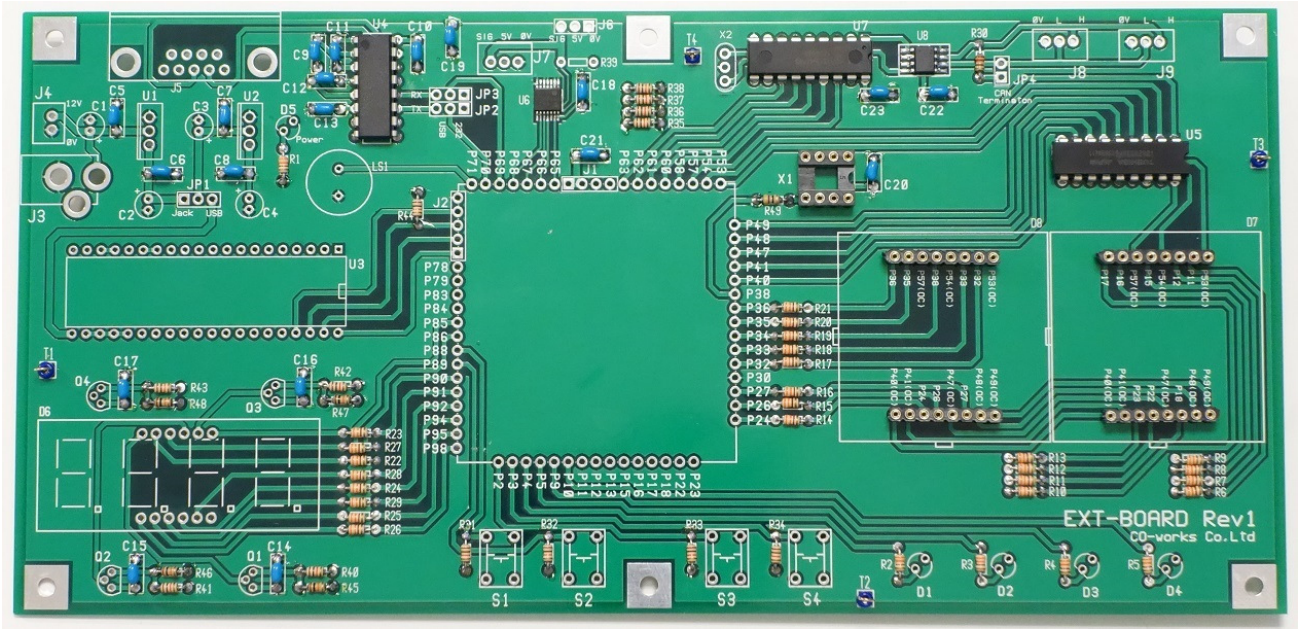

図 1.25　各種 IC 実装後の EXT-BOARD

10MHz 発信子のはんだ付け

X2 に 10MHz のセラミック発信子を実装します。

極性はありません。

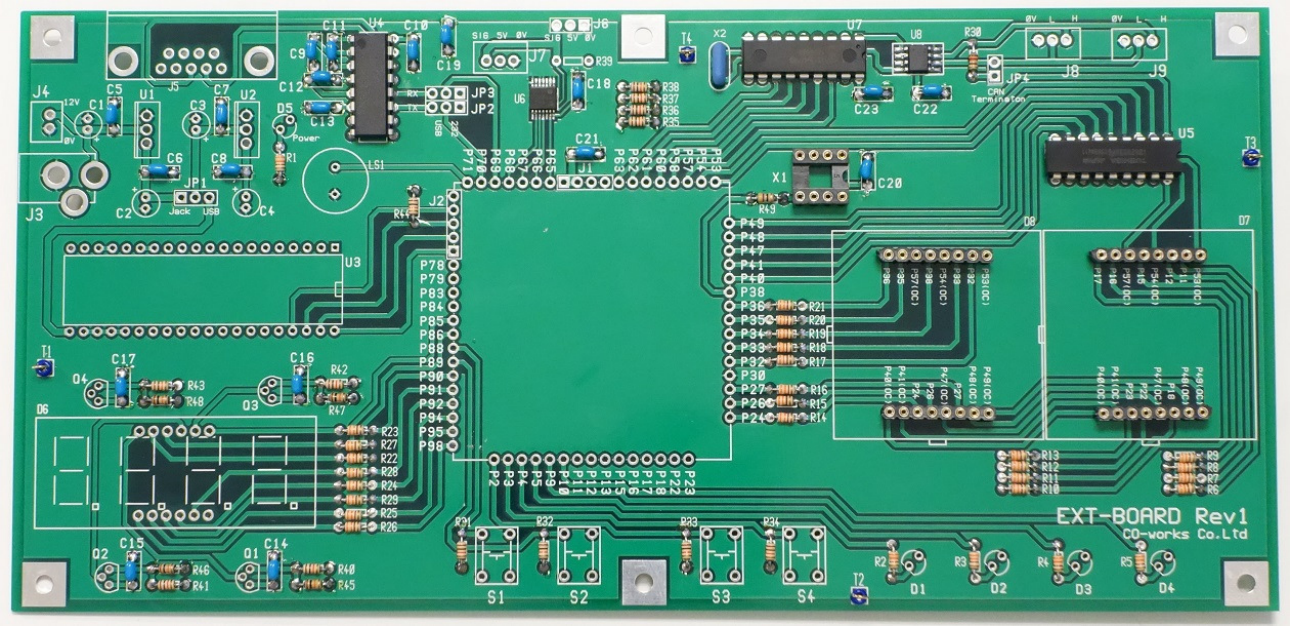

図 1.26　発信子実装後の EXT-BOARD

トランジスタの実装

Q1～Q4 のトランジスタを実装します。

極性があります、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 1.27　トランジスタの実装方向

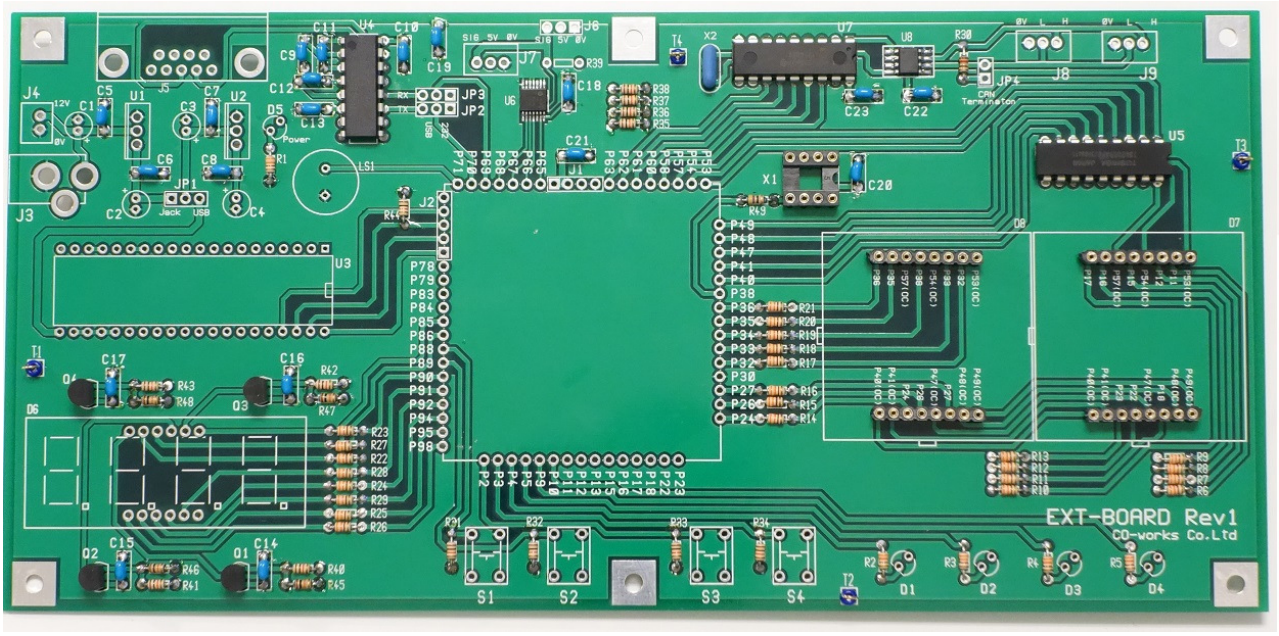

図 1.28　トランジスタ実装後の EXT-BOARD

圧電スピーカのはんだ付け

LS1 に圧電スピーカを実装します。

極性はありません。

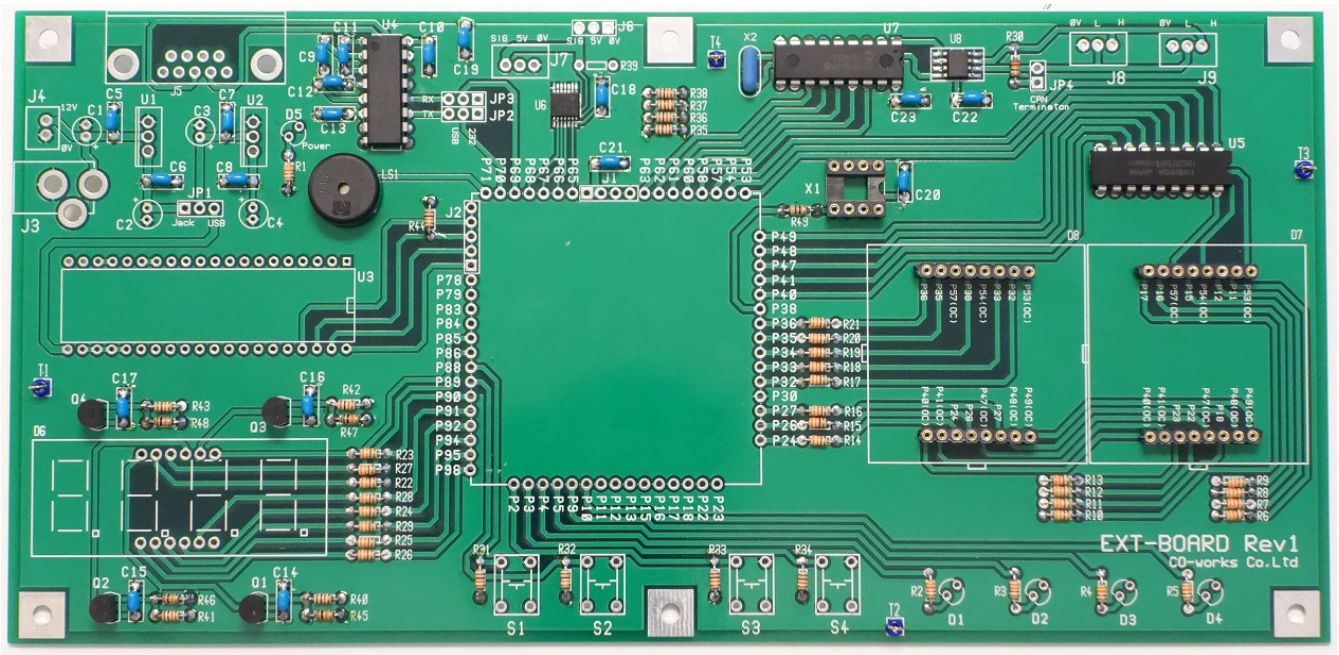

図 x1.29　圧電スピーカ実装後の EXT-BOARD

コネクタのはんだ付け

J4、J7～J9 のコネクタを実装します。

極性があります。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）

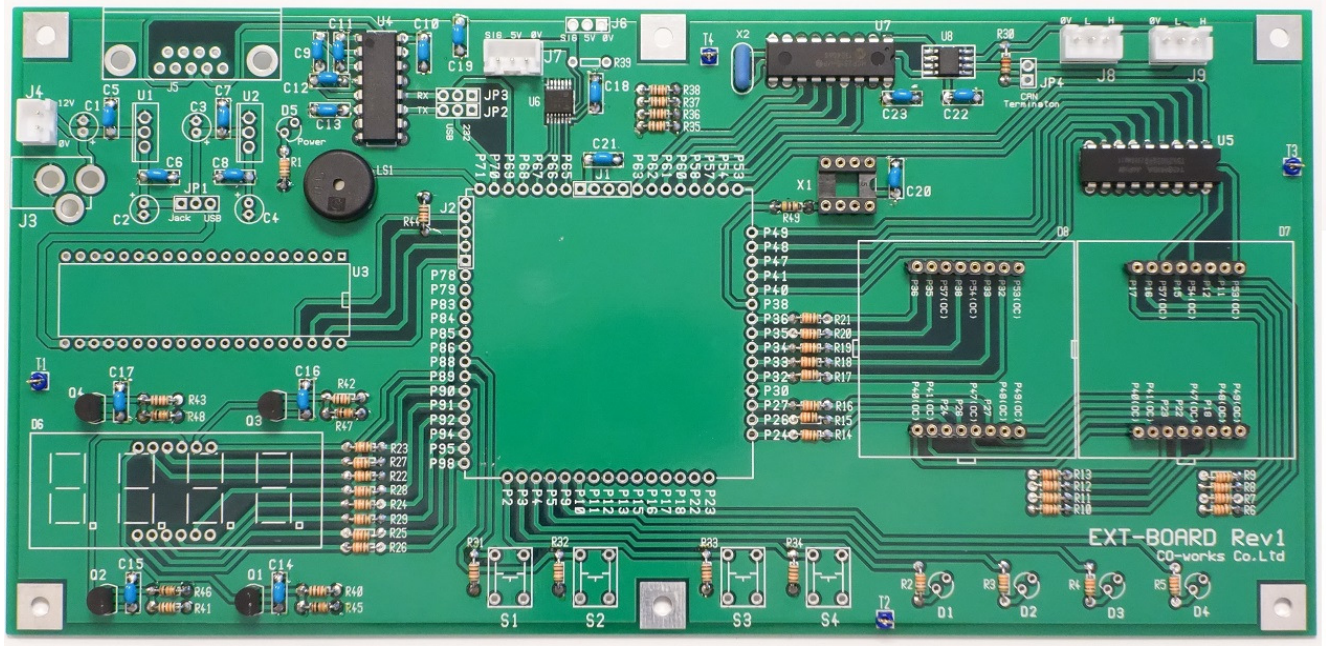

図 1.30　コネクタ実装後の EXT-BOARD

タクトスイッチのはんだ付け

S1～S4 にタクトスイッチを実装します。

極性があります。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。

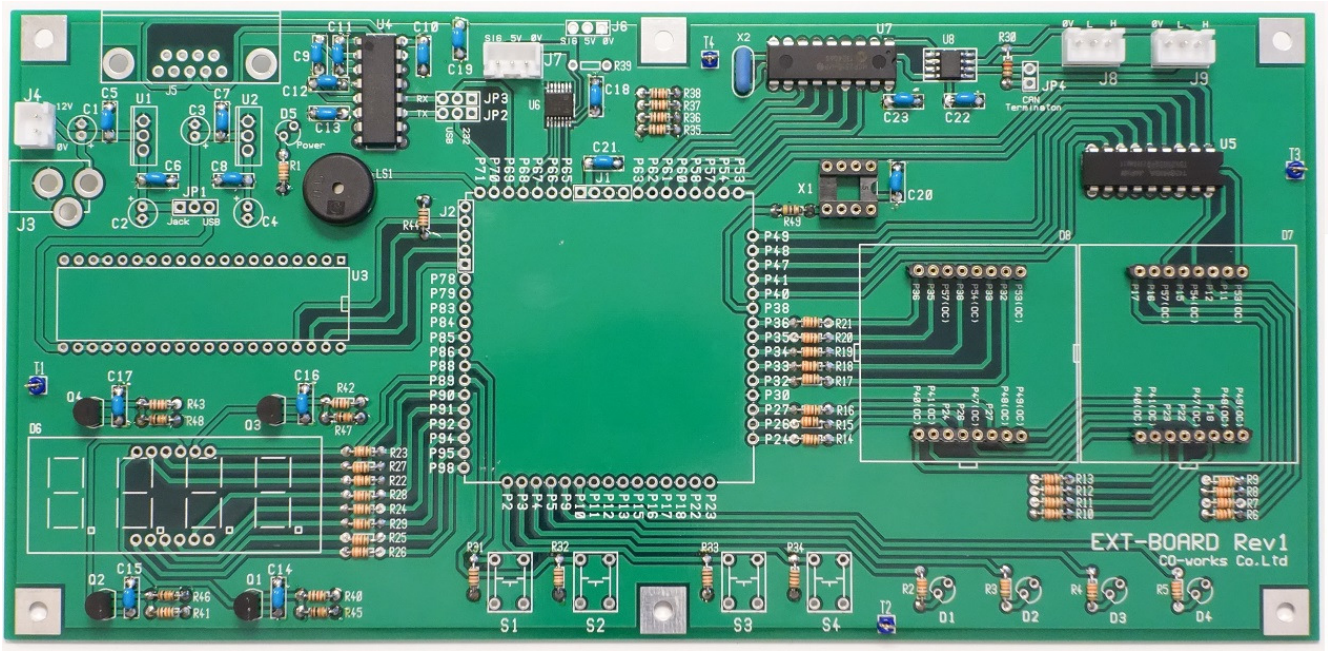

図 1.31　タクトスイッチ実装後の EXT-BOARD

ピンヘッダ/ピンソケットのはんだ付け

事前に分割しておいた、ピンヘッダとピンソケットを実装します。

極性はありません。

ピンヘッダ：JP1～JP4

ピンソケット：J6、Px

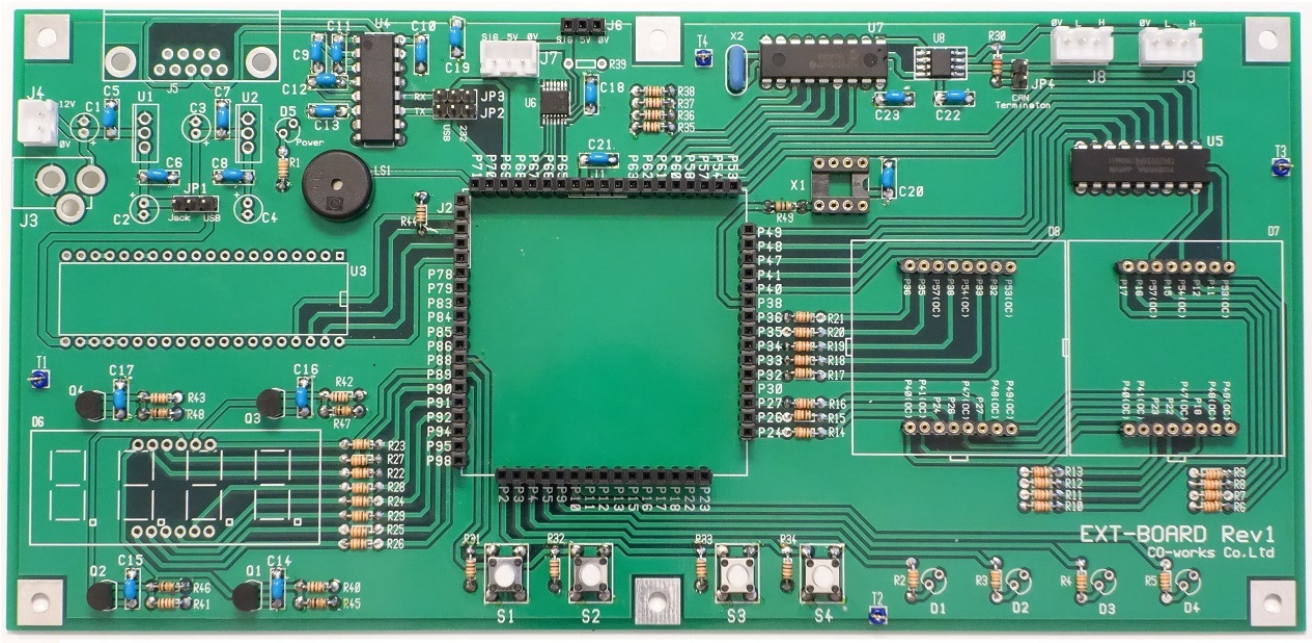

図 1.32　ピンヘッダ/ピンソケット実装後の EXT-BOARD

DC ジャックと電解コンデンサのはんだ付け

J3 の DC ジャックをはんだ付けします。　極性はありません。

C1～C4 の電解コンデンサをはんだ付けします。

極性があります。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。

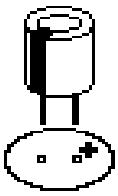

図 1.33　電解コンデンサの実装方向

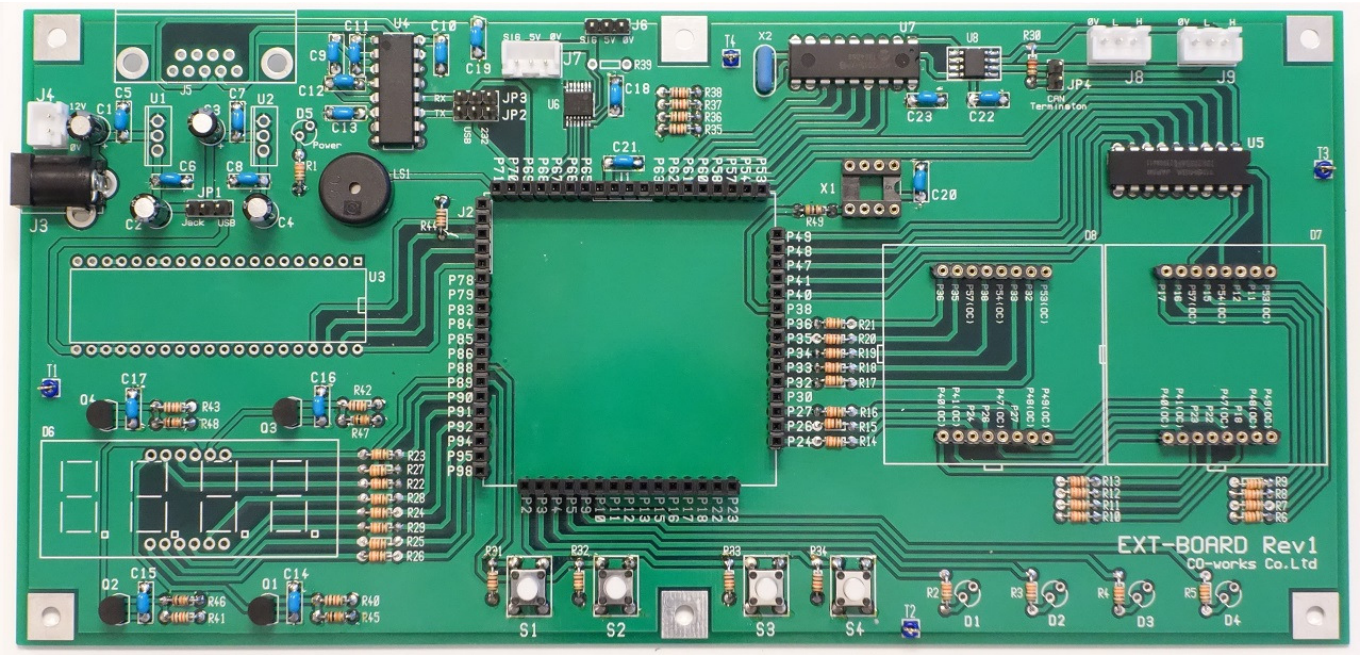

図 1.34　DC ジャックと電解コンデンサ実装後の EXT-BOARD

7 セグメント LED のはんだ付け

D6 の 7 セグメント LED を実装します。

極性があります。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。

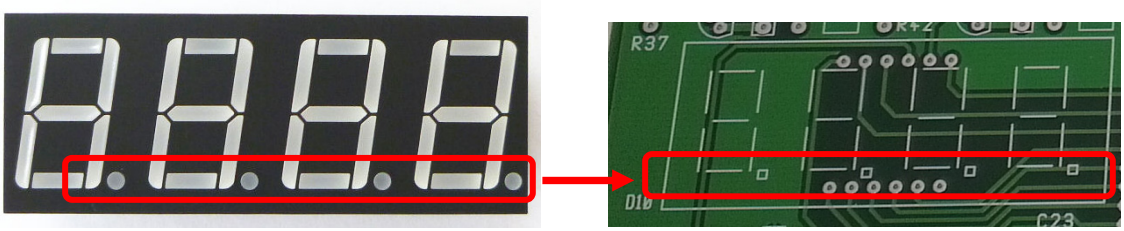

図 1.35　7 セグメント LED の実装方向

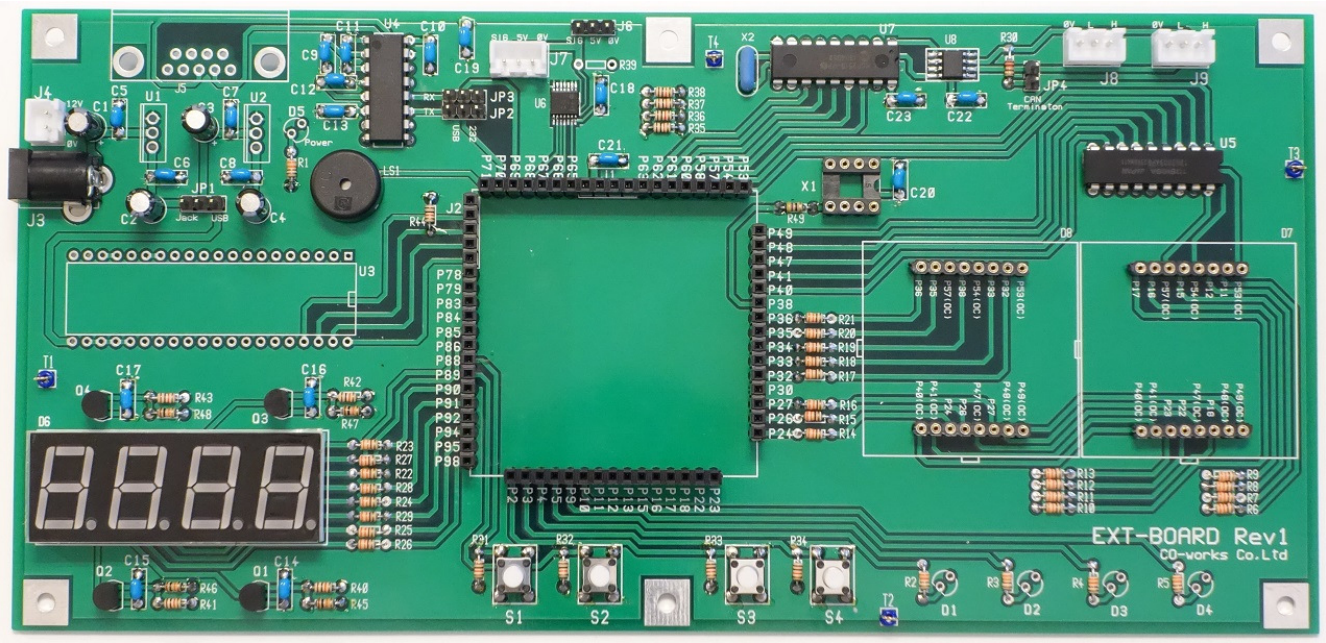

図 1.36　7 セグメント LED 実装後の EXT-BOARD

LED の実装

D1～D5 の LED を実装します。

極性があります。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。

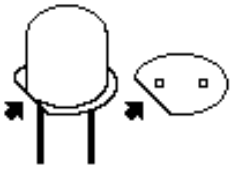

図 1.37　LED の実装方向

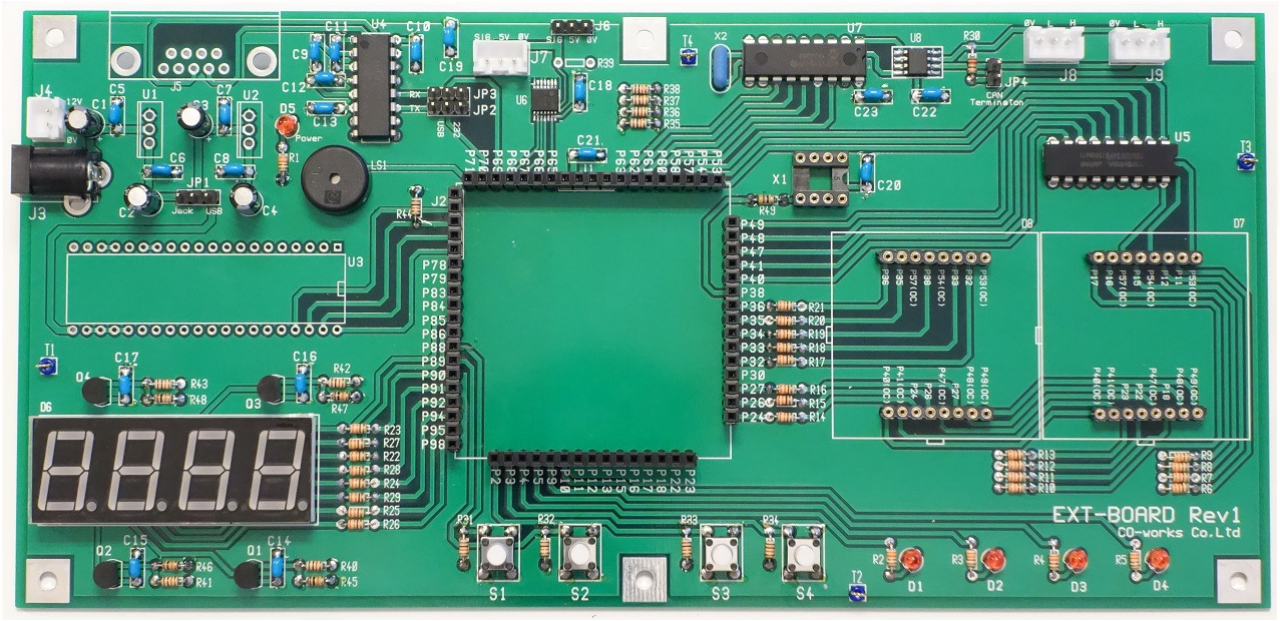

図 1.38　LED 実装後の EXT-BOARD

USB-JTAG/シリアル変換モジュールのはんだ付け

U1 に USB-JTAG/シリアル変換モジュールを実装します。

極性があります。USB コネクタが基板の外を向くように実装します。

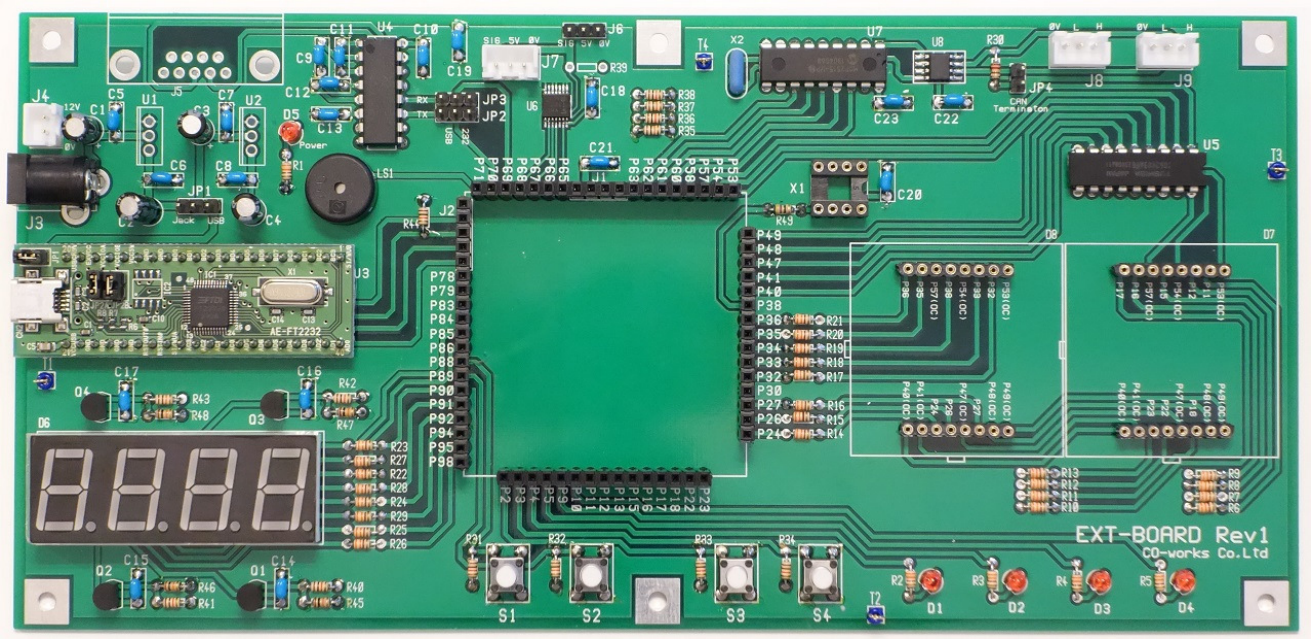

図 1.39　USB-JTAG/シリアル変換モジュール実装後の EXT-BOARD

レギュレータのはんだ付け

U1 に 5V レギュレータ、U2 に 3.3V レギュレータを実装します。

極性があります。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 1.40　レギュレータの実装方向

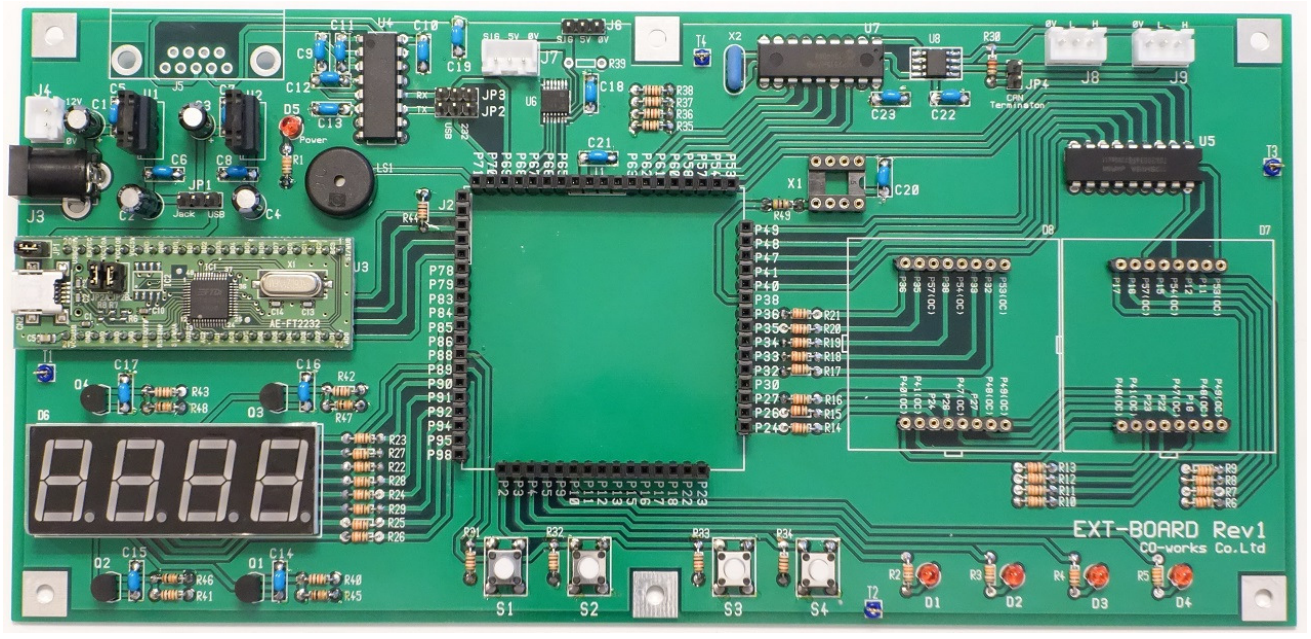

図 1.41　レギュレータ実装後の EXT-BOARD

Dsub9 ピンコネクタのはんだ付け

J5 に Dsub9 ピンのオスコネクタを実装します。

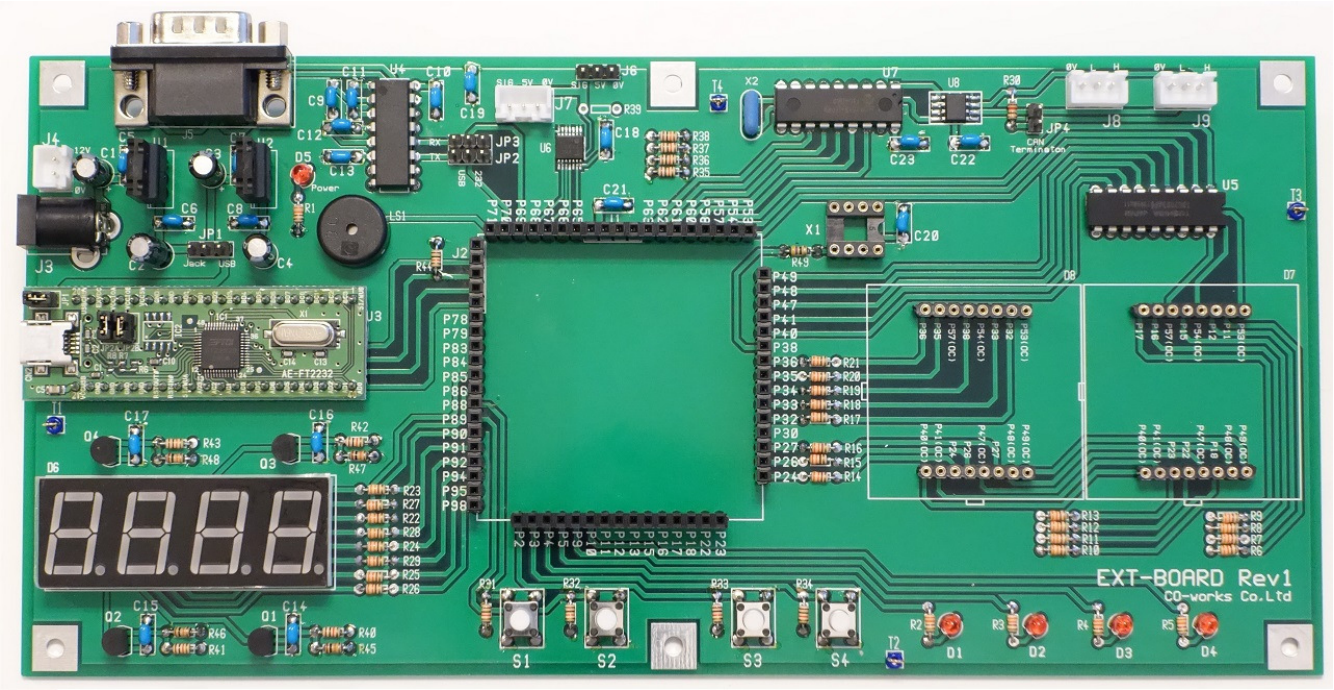

図 1.42　Dsub9 ピンコネクタ実装後の EXT-BOARD

プラ足の貼り付け

基板を裏返し、プラ足を貼り付けます。

プラ足は、両面テープになっているため、剥離紙をはがしてから基板へ貼り付けます。

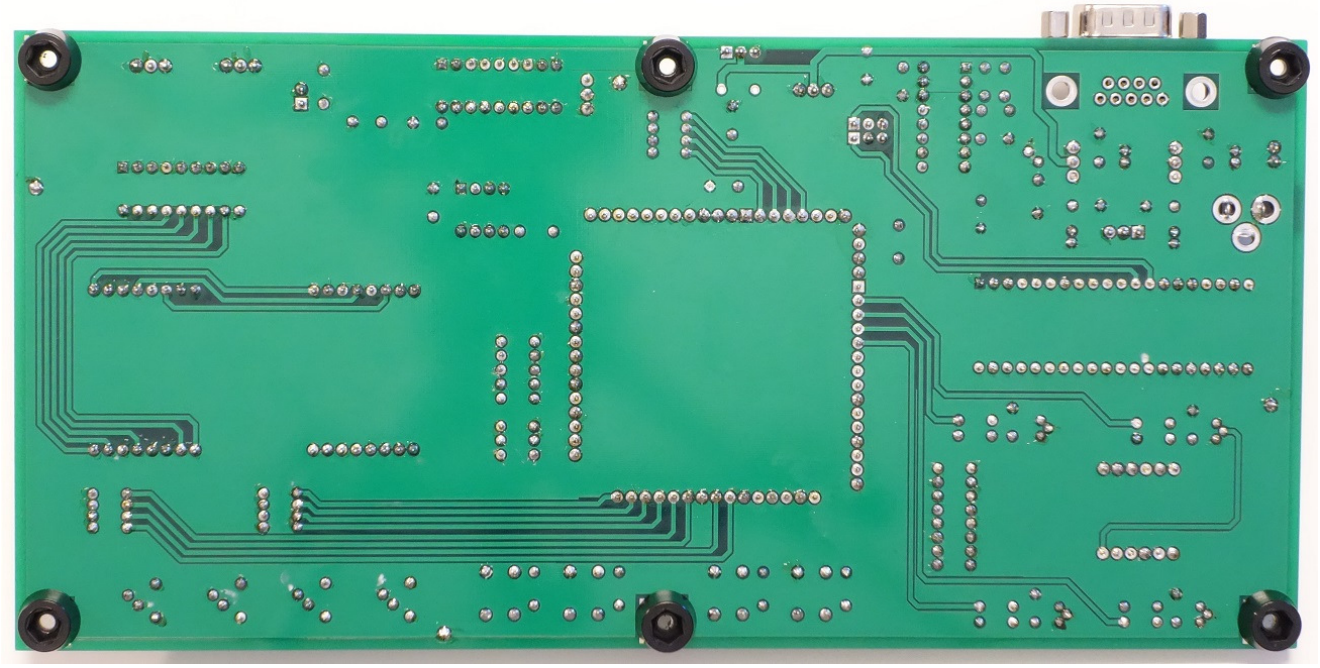

図 1.43　プラ足貼り付け後の EXT-BOARD

以上で、EXT-BOAD 基板への部品実装は完了です。

つづけて、FPGA 基板へのはんだ付けを行います。

FPGA 基板へのピンヘッダはんだ付け

FPGA 基板へ、事前に分割していたピンヘッダを実装します。

FPGA 基板は、FPGA のチップが実装してある面が表面です。

ピンヘッダの表と裏の実装方向に注意してください。JPA1 のみ表面で、それ以外は裏向きに実装します。

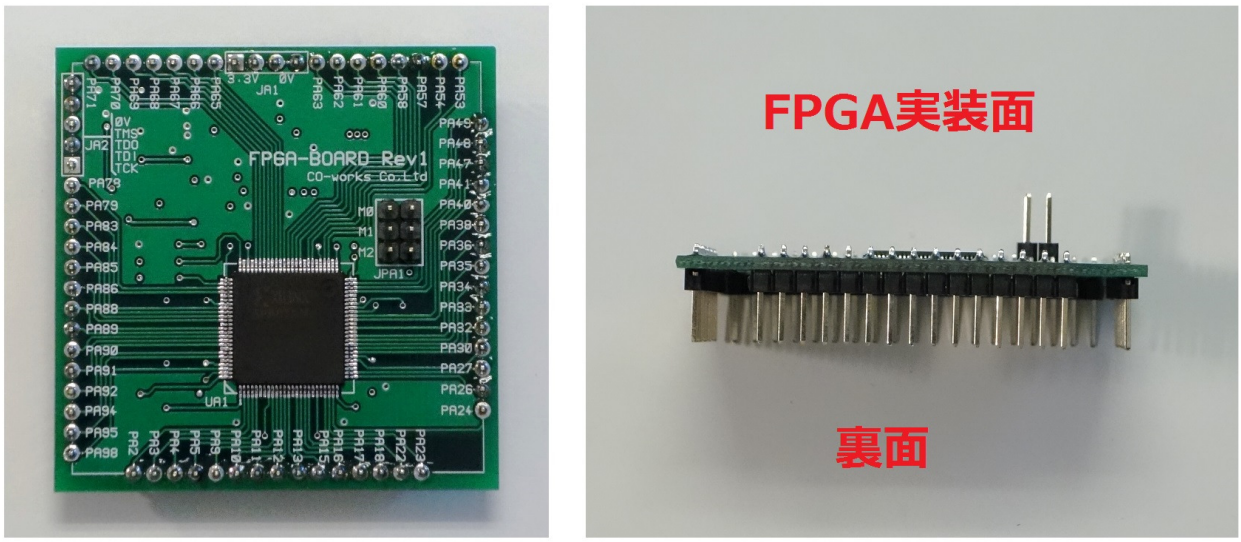

図 1.44　FPGA-BOARD 基板へのピンヘッダ実装方向

以上で、すべてのはんだ付けは終了です。

取り外し可能モジュールの取り付け

はんだ付けが完了した EXT-BOARD に FPGA-BOAR・超音波センサーモジュール・ドットマトリクス LED・33.33MH z オシレータを取りつけます。

差し込み式になっているため、手作業で取り付けできます。取り付け方向は、下図を参照してください。

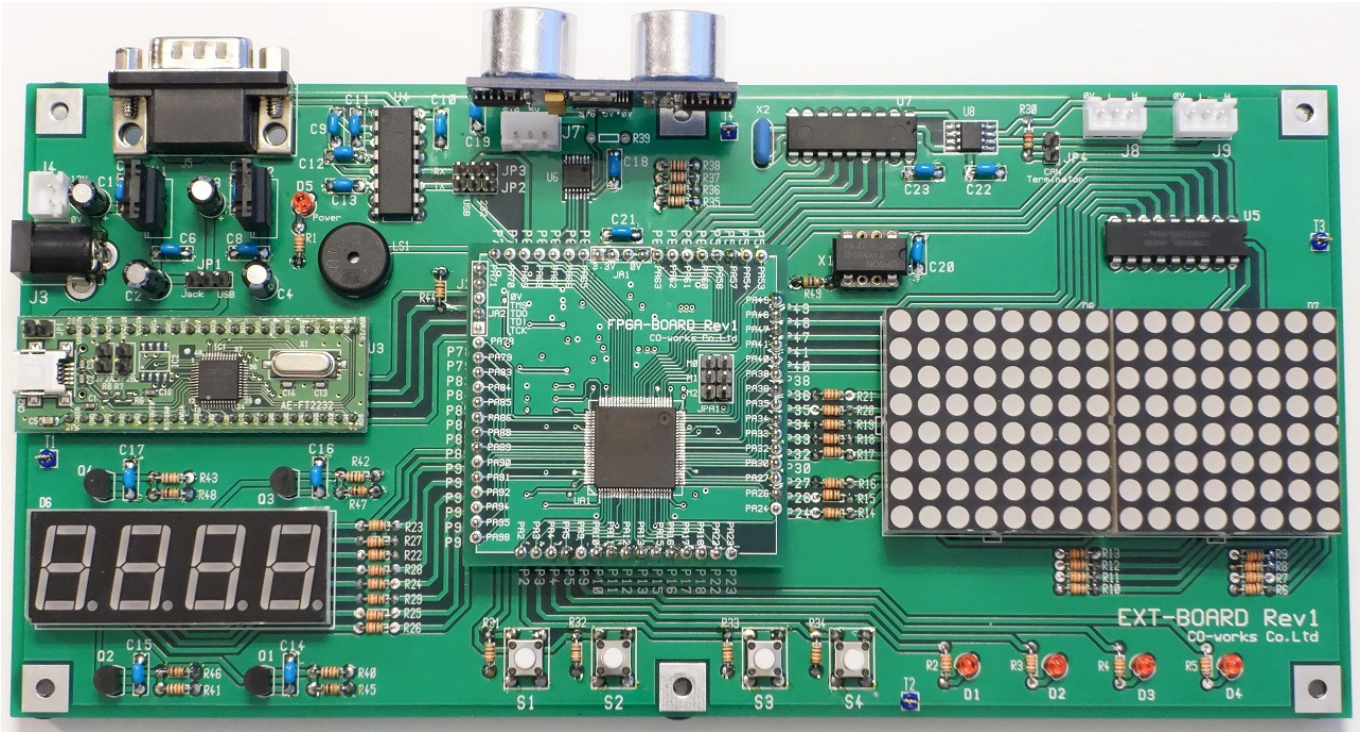

図 1.45　全モジュール取り付け後の EXT-BOARD

ジャンパピン設定

ジャンパピンの設定が完了するまで電源は入れないでください。

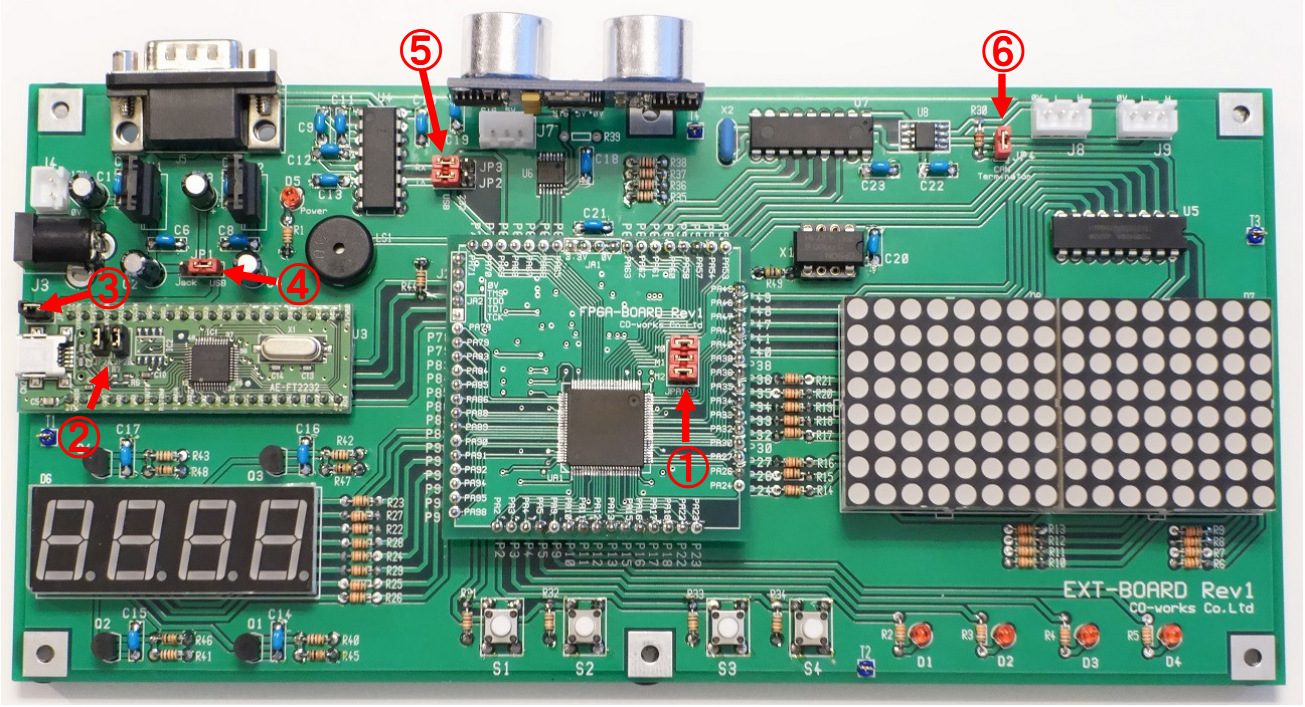

図１．４６　ジャンパピン設定

① FPGA 動作モード設定

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**（重要、FPGA を破損する可能性があります）**

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。

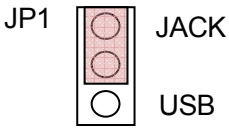

図 1.47 DC ジャック給電の場合

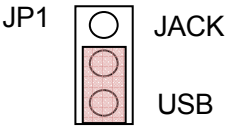

図 1.48　USB 給電の場合

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。

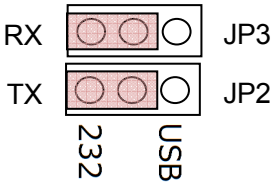

図 1.49　Dsub9 ピン RS-232C の場合

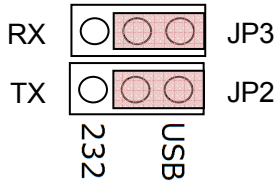

図 1.50　USB 仮想 COM ポートの場合

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。

参考資料

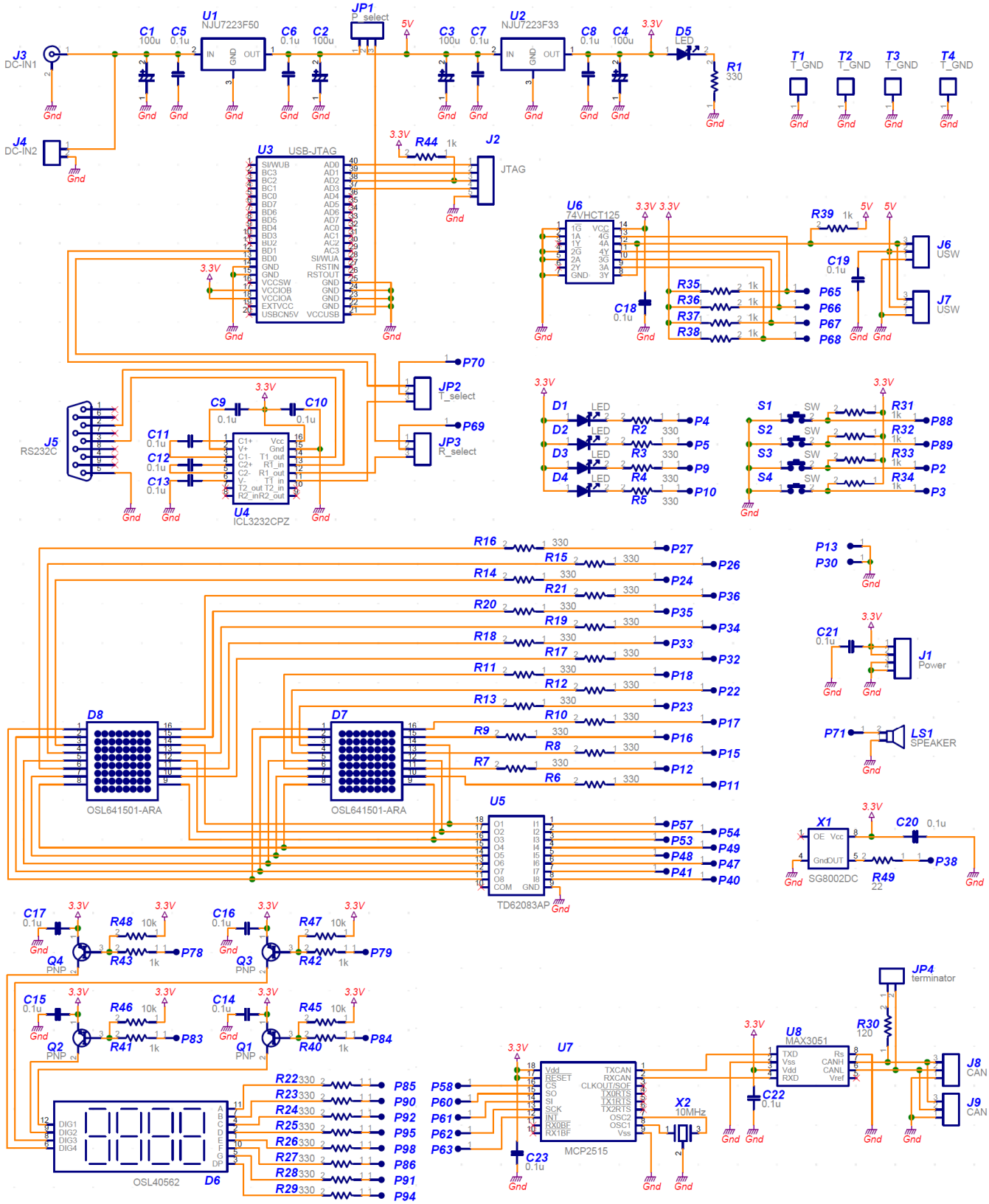

図 1.51　EXT-BOARD 全体回路図

表１．４　FPGA ピンアサイン 1～50

FPGA		コネクタ	最終接続先
ピン番号	ピン名称		
1	PROG_B	内部	XFC01SVO20CのCF(7ピン)
2	IO_L01P_3	P2	スイッチ3
3	IO_L01N_3	P3	スイッチ4
4	IO_L02P_3	P4	LED1
5	IO_L02N_3/VREF_3	P5	LED2
6	VCCINT	内部電源	1.2V
7	GND	J1の3,4ピン	0V
8	VCCO_3	J1の1,2ピン	3.3V
9	IO_L03P_3/LHCLK0	P9	LED3
10	IO_L03N_3/LHCLK1	P10	LED4
11	IO_L04P_3/LHCLK2	P11	マトリクス12
12	IO_L04N_3/LHCLK3	P12	マトリクス14
13	IP(入力専用)	P13	GND
14	GND	J1の3,4ピン	0V
15	IO_L05P_3/LHCLK4	P15	マトリクス9
16	IO_L05N_3/LHCLK5	P16	マトリクス15
17	IO_L06P_3/LHCLK6	P17	マトリクス16
18	IO_L06N_3/LHCLK7	P18	マトリクス13
19	GND	J1の3,4ピン	0V
20	VCCO_3	J1の1,2ピン	3.3V
21	VCCAUX	内部電源	2.5V
22	IO_L07P_3	P22	マトリクス11
23	IO_L07N_	P23	マトリクス10
24	IO_L01P_2/CSO_B	P24	マトリクス2
25	IO_L01N_2/INIT_B	内部	XFC01SVO20CのOE(8ピン)
26	IO_L02P_2/DOUT/BUSY	P26	マトリクス3
27	IO_L02N_2/MOSI/CSI_B	P27	マトリクス5
28	VCCINT	内部電源	1.2V
29	GND	J1の3,4ピン	0V
30	IP/VREF_2(入力専用)	P30	GND
31	VCCO_2	J1の1,2ピン	3.3V
32	IO_L03P_2/D7/GCLK12	P32	マトリクス4
33	IO_L03N_2/D6/GCLK13	P33	マトリクス6
34	IO/D5	P34	マトリクス1
35	IO_L04P_2/D4/GCLK14	P35	マトリクス7
36	IO_L04N_2/D3/GCLK15	P36	マトリクス8
37	GND	J1の3,4ピン	0V
38	IP_L05P_2/RDWR_B/GCLK0(入力専用)	P38	SG8002DC
39	IP_L05N_2/M2/GCLK1	内部	MODEジャンパ
40	IO_L06P_2/D2/GCLK2	P40	マトリクスコモン5
41	IO_L06N_2/D1/GCLK3	P41	マトリクスコモン7
42	IO/M1	内部	MODEジャンパ
43	IO_L07P_2/M0	内部	MODEジャンパ
44	IO_L07N_2/DIN/D0	内部	XFC01SVO20CのD0(1ピン)
45	VCCO_2	J1の1,2ピン	3.3V
46	VCCAUX	内部電源	2.5V
47	IO_L08P_2/VS2	P47	マトリクスコモン8
48	IO_L08N_2/VS1	P48	マトリクスコモン6
49	IO_L09P_2/VS0	P49	マトリクスコモン3
50	IO_L09N_2/CCLK	内部	XFC01SVO20CのCLK(3ピン)

表１．５　FPGA ピンアサイン 51～100

FPGA		コネクタ	最終接続先
ピン番号	ピン名称		
51	DONE	内部	XFC01SVO20CのCE(10ピン)
52	GND	J1の3,4ピン	0V
53	IO_L01P_1	P53	マトリクスコモン1
54	IO_L01N_1	P54	マトリクスコモン4
55	VCCO_1	J1の1,2ピン	3.3V
56	VCCINT	内部電源	1.2V
57	IO_L02P_1	P57	マトリクスコモン2
58	IO_L02N_1	P58	MCP2515のCS(16ピン)
59	GND	J1の3,4ピン	0V
60	IO_L03P_1/RHCLK0	P60	MCP2515のSO(15ピン)
61	IO_L03N_1/RHCLK1	P61	MCP2515のSI(14ピン)
62	IO_L04P_1/RHCLK2	P62	MCP2515のSCK(13ピン)
63	IO_L04N_1/RHCLK3	P63	MCP2515のINT(12ピン)
64	GND	J1の3,4ピン	0V
65	IO_L05P_1/RHCLK4	P65	74VHC125の4G(13ピン)
66	IO_L05N_1/RHCLK5	P66	74VHC125の4Y(11ピン)
67	IO_L06P_1/RHCLK6	P67	74VHC125の3G(10ピン)
68	IO_L06N_1/RHCLK7	P68	74VHC125の3Y(9ピン)
69	IP/VREF_1(入力専用)	P69	シリアル受信
70	IO_L07P_1	P70	シリアル送信
71	IO_L07N_1	P71	圧電スピーカ
72	GND	J1の3,4ピン	0V
73	VCCO_1	J1の1,2ピン	3.3V
74	VCCAUX	内部電源	2.5V
75	TMS	jtag	JATG
76	TDO	jtag	JATG
77	TCK	jtag	JATG
78	IO_L01P_0	P78	7SEGコモン4
79	IO_L01N_0	P79	7SEGコモン3
80	VCCINT	内部電源	1.2V
81	GND	J1の3,4ピン	0V
82	VCCO_0	J1の1,2ピン	3.3V
83	IO_L02P_0/GCLK4	P83	7SEGコモン2
84	IO_L02N_0/GCLK5	P84	7SEGコモン1
85	IO_L03P_0/GCLK6	P85	7SEG1
86	IO_L03N_0/GCLK7	P86	7SEG6
87	GND	J1の3,4ピン	0V
88	IP_L04P_0/GCLK8(入力専用)	P88	スイッチ1
89	IP_L04N_0/GCLK9(入力専用)	P89	スイッチ2
90	IO_L05P_0/GCLK10	P90	7SEG2
91	IO_L05N_0/GCLK11	P91	7SEG7
92	IO	P92	7SEG3
93	GND	J1の3,4ピン	0V
94	IO_L06P_0	P94	7SEG8
95	IO_L06N_0/VREF_0	P95	7SEG4
96	VCCAUX	内部電源	2.5V
97	VCCO_0	J1の1,2ピン	3.3V
98	IO_L07P_0	P98	7SEG5
99	IO_L07N_0/HSWAP	内部	2.5V
100	TDI	jtag	JATG

２．ＦＰＧＡ開発環境

今回のFPGAの設計では、Xilinx(ザイリンクス)社の開発ツールISEを使用します。

言語はVerilogHDLで、記述レベルはRTL(Register Transfer Level)です。RTLはクロックの概念が存在する詳細ブロック図のようなものと理解して下さい。HDL（Hardware Description Language）による設計では記述レベルが５階層あるとも言われていますが、ほとんどの設計では論理合成を行う都合から、RTLであると解釈して良いと思います。RTLと比較してビヘイビア・レベルという記述レベルがあり、クロックの概念が存在しない記述レベルです。

２．１　ＩＳＥについて

ISE（Integrated Software Environment）はザイリンクス社の登録商標です。ISEには各種バージョンがあり、さらにバージョン毎に有償版、無償版が準備されています。応用編でも基礎編同様Web上から無償で容易に入手できるWebPACKを使用します。バージョンについて、基礎編ではISE8.2iを使用しましたが、応用編ではバージョンアップをしISE10.1を使用してRTLを記述しました。

通常FPGAへのデータ書き込みについては、ザイリンクス社が有償供給するJTAGケーブルを使用するので、ISEのバージョンを選ぶことはありません。

今回は有償となるJTAGケーブルを使用しない方法として、FT2232と言うチップを搭載したUSBモジュールを採用し、USB/JTAG変換にて書き込みを行います。USB/JTAG変換ではWeb上から無償で入手可能な変換ソフトcblsrvを使用します。

FT2232のドライバと変換ソフトの組み合わせでは、ISEのバージョンを選ぶことが確認されています。正確にはISEに付いているiMPACTというデバイスへの書き込みツールのバージョンを選ぶようです。

応用編では ISE 11.1 と言うバージョンで試行しましたが、その結果上記組み合わせでの書き込みができませんでした。
そこで、ISE 10.1 にバージョンを下げて FT2232 のドライバと変換ソフトの組み合わせによる書き込みを実現させました。

ISE の無償版 WebPACK は下記ザイリンクス社のホームページからダウンロード可能です。

http://japan.xilinx.com/support/download/index.htm

２．２　チュートリアル

応用編で使用した ISE10.1 の使用方法を例に、FPGA 開発ツールの使い方を学ぶと同時に、6 章で説明する応用編のプロジェクトを作成します。
FPGA の開発では、その開発単位のことをプロジェクトと呼びます。この呼び方は、開発ツールのメーカーが異なっても同じようです。

同じメーカーのツールでも、バージョンが異なると使用方法が異なります。バージョンによっては大きく変わる場合もありますが、基礎編で使用した ISE8.2i と ISE10.1 では大差がありませんので、馴染みやすいと思います。

FPGA 開発ツールの使い方は操作（オペレーション）に関する項目ですから、HDL 言語による記述を学ぶものではありません。しかし、実際の開発現場では、ツールの使い方に精通していないとなかなか思うように設計できません。FPGA 開発技術者にとって、開発ツールはお箸でご飯を食べたり、鉛筆でメモを書くぐらいの感覚が要求されます。操作に関する項目ですが、ツールが何を実行しているのか、それが FPGA 回路のどの部分に該当するのか、最低限の原理を知っていると、使用するデバイスメーカーが変わり、使用するツールが変わっても、比較的柔軟に対応できるようになります。

ISE10.1 のチュートリアルに入る前に、開発ツールがどのようなプロセスで回路データを生成するのか、簡単に説明します。

プロジェクトを作成した後、C 言語で例えると実行ファイルに該当する回路データの生成を行います。

回路データ生成のために、最初に論理合成を行います。ここでは、シンタックスエラー（文法エラー）がないかをツールが判断し、エラーが無ければ記述された回路に沿ったロジックを作ると理解して下さい。

論理合成に成功すると、マッピングと呼ばれる配置に進みます。FPGA は言語で開発しても、あくまでハードウェアです。FPGA の内部に回路部品を配置して配線（プリント基板のアートワークと同じ）します。マッピング（配置）に成功すると、ルーティングと呼ばれる配線に進みます。

ここまで成功すると、書き込みに使用するプログラミングデータの作成に進みます。これが、C 言語の実行ファイルに該当するデータです。

Warning（ワーニング）について

論理合成から回路データ生成までの過程で、エラーが出てはいけません。では、ワーニングはどうでしょうか。

ワーニングも無いのが望ましいのですが、必須ではありません。ワーニングが残っていてもプログラミングデータは生成されます。そこで、優先順位をつけて考えてみましょう。

１）ワーニングは無いのがベスト。

２）ワーニングが残った場合、その内容を必ず確認する。

３）内容確認の結果、問題の無いものは無視する。

その判断ですが、記述上定義されているが実際の回路では使われていないので無視された場合などは、本当に無視してよい信号か。記述上の定義は必要だが、使い切る必要の無いカウンタなどがその例です。

初心者にとってワーニングの判断は難しいですが、学習の場合はまず試してみる。そして、実際のプロの現場では、一つずつ判断していくしかありません。

それでは、ISE10.1 のチュートリアルに進みましょう。

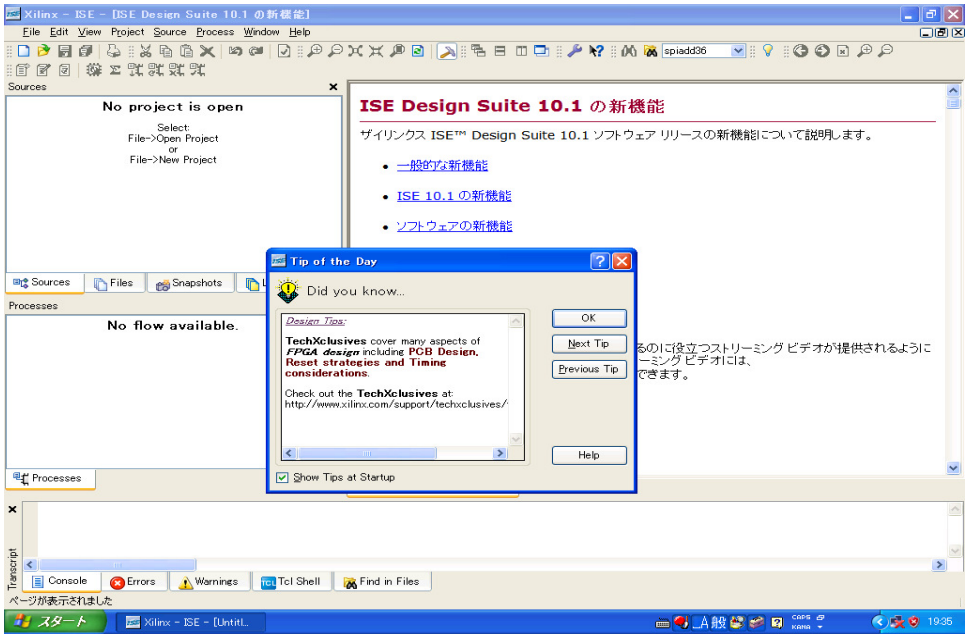

図２．１

既設計のプロジェクトがない状態で ISE10.1 を起動すると、図のような画面になります。Tip of the Day は使い方のヒントのようなものですから、×で終了します。

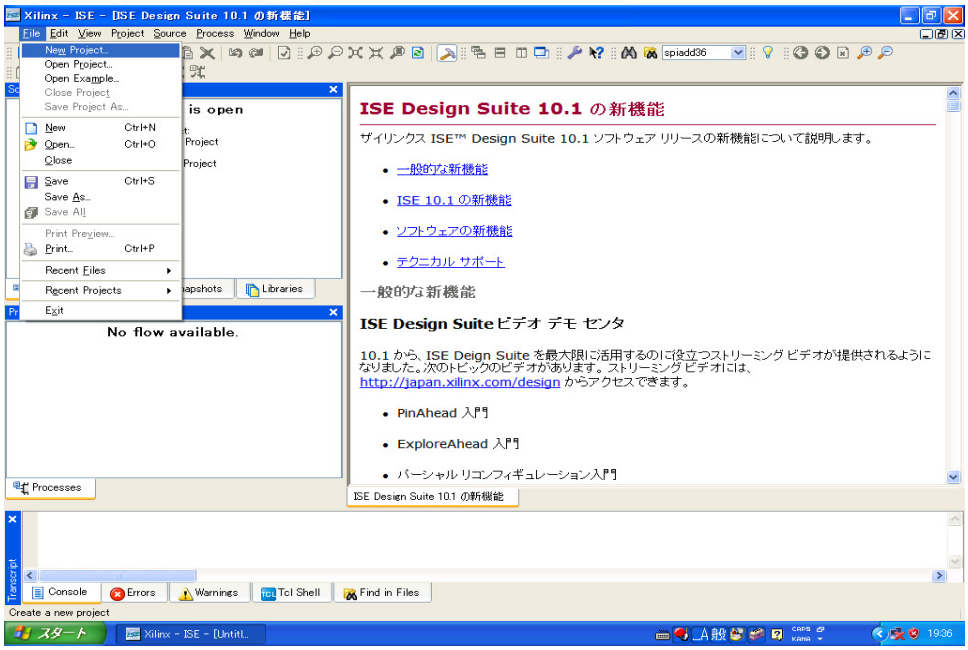

図２．２

File の New Project をクリックします。

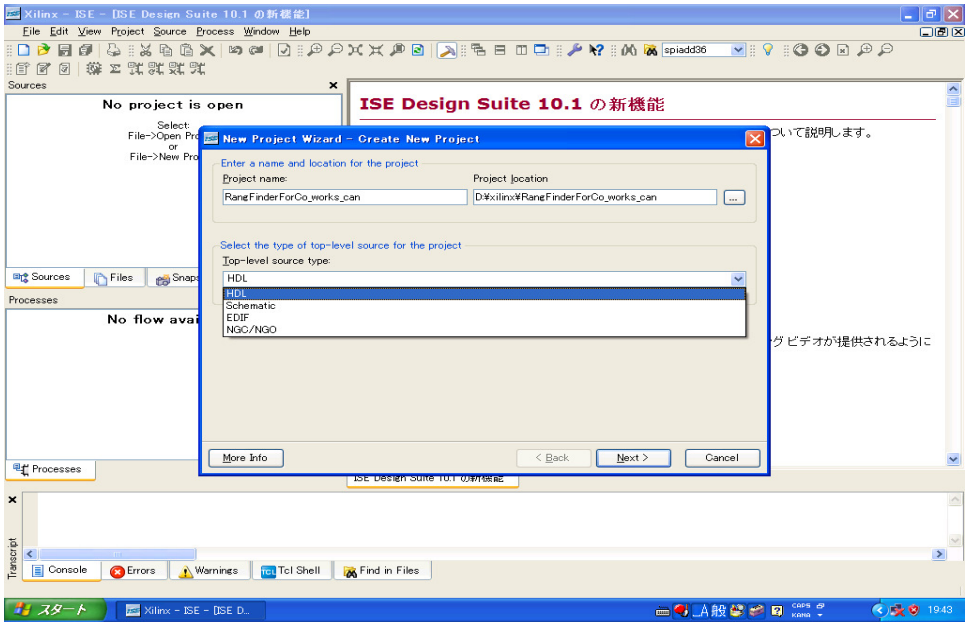

図２．３

Project location を参照して決定し、Project name を入力します。

Top-level source type で HDL を選択し、Next をクリックします。

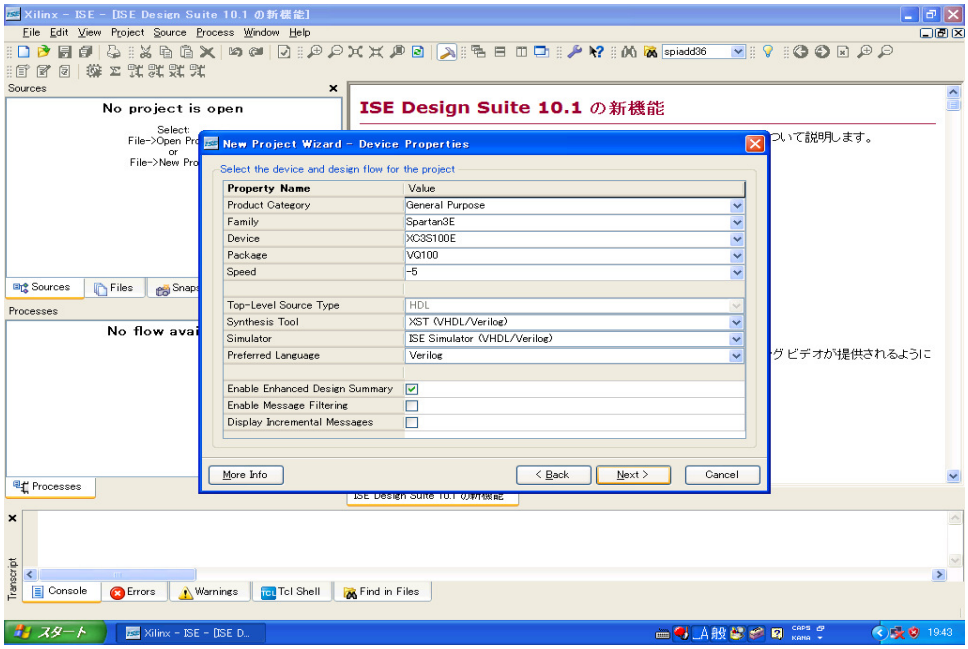

図２．４

Family Spartan3E　　Device XC3S100E VQ100　　Language Verilog

これらを選択し、Next をクリックします。

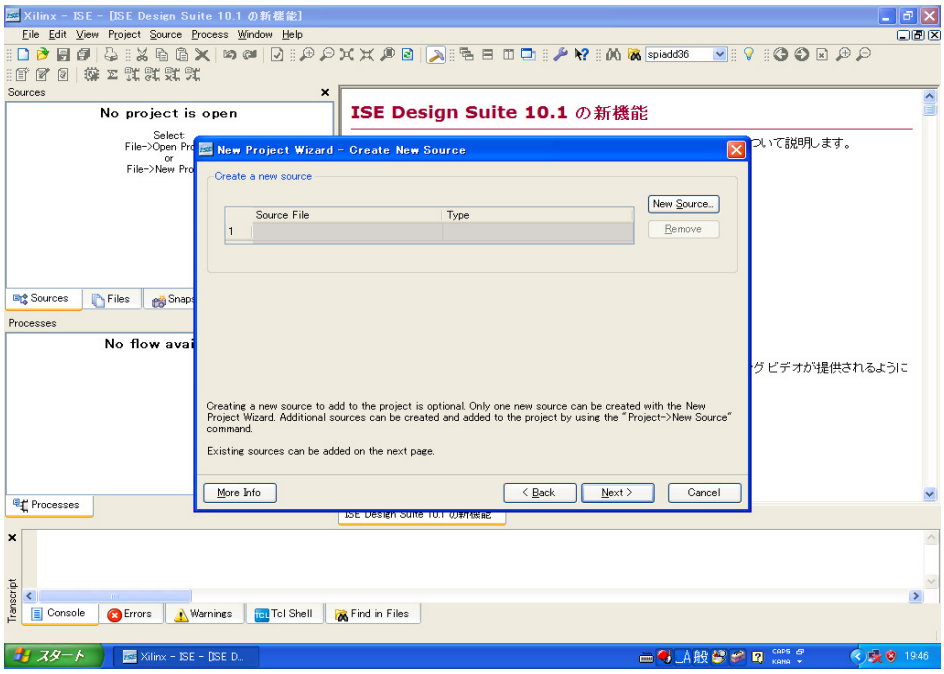

図２．５

Next をクリックします。

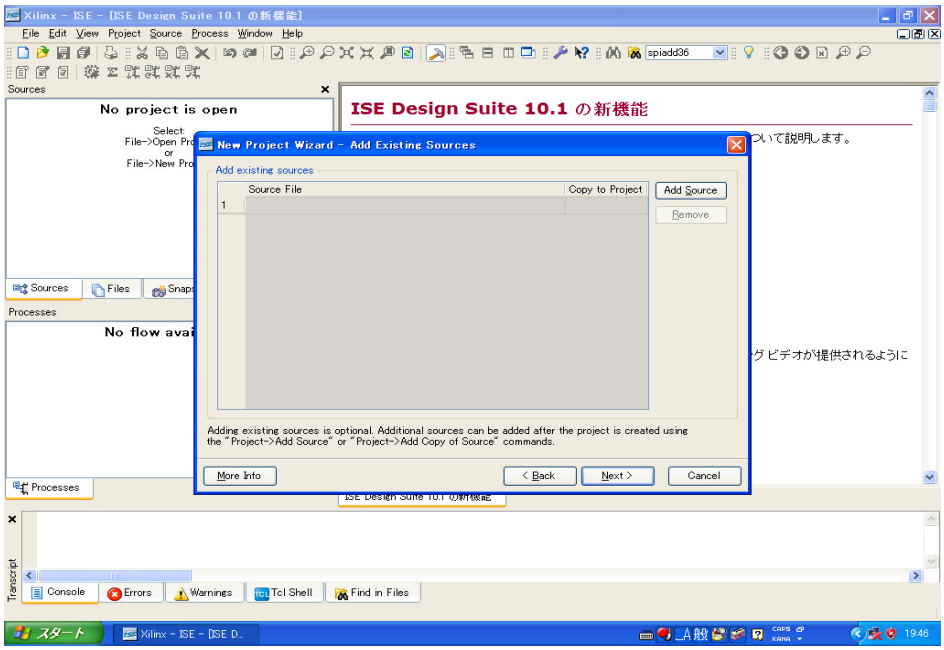

図２．６

Next をクリックします。

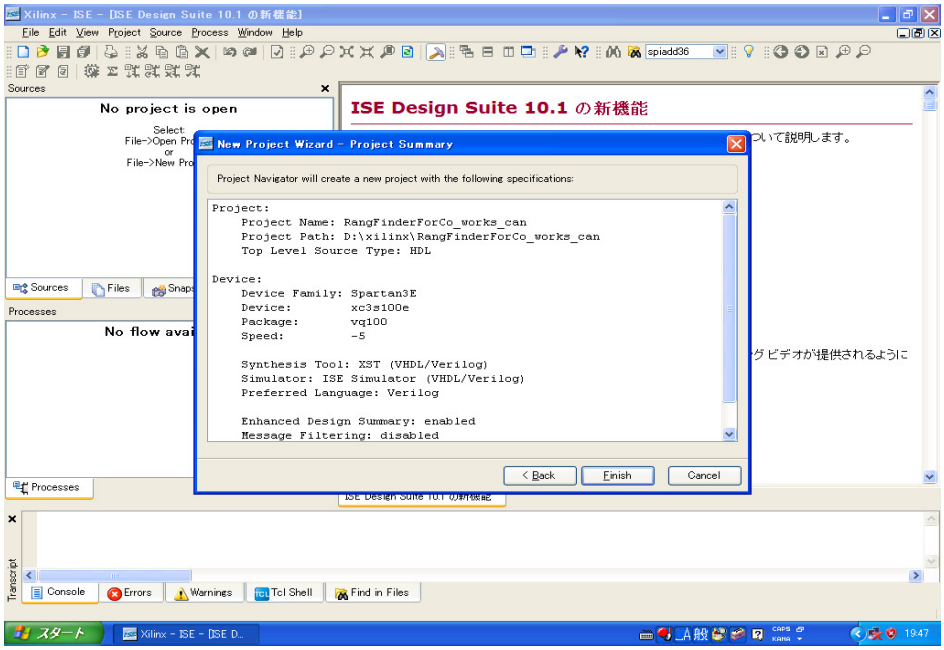

図２．７

Finish をクリックします。

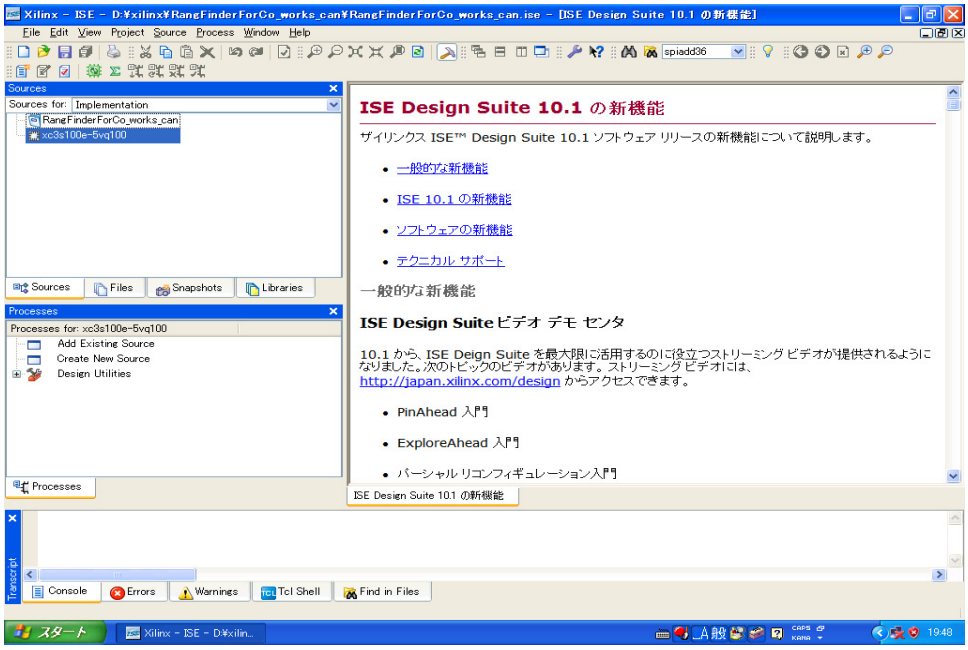

図２．８

Source の xc3s100e-5vq100 をクリックします。

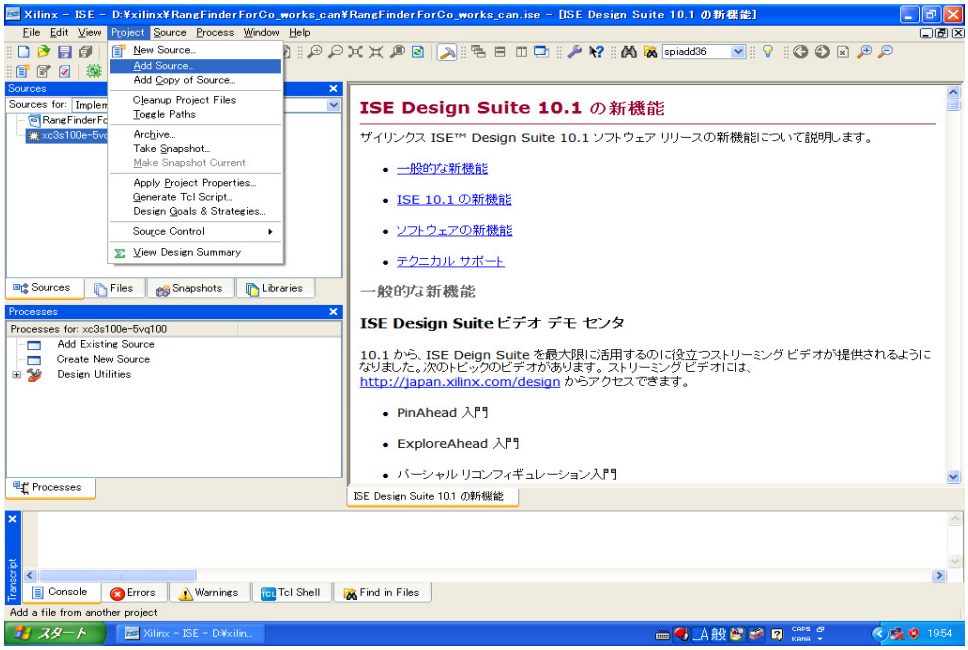

図２．９

Project の Add Source をクリックします。

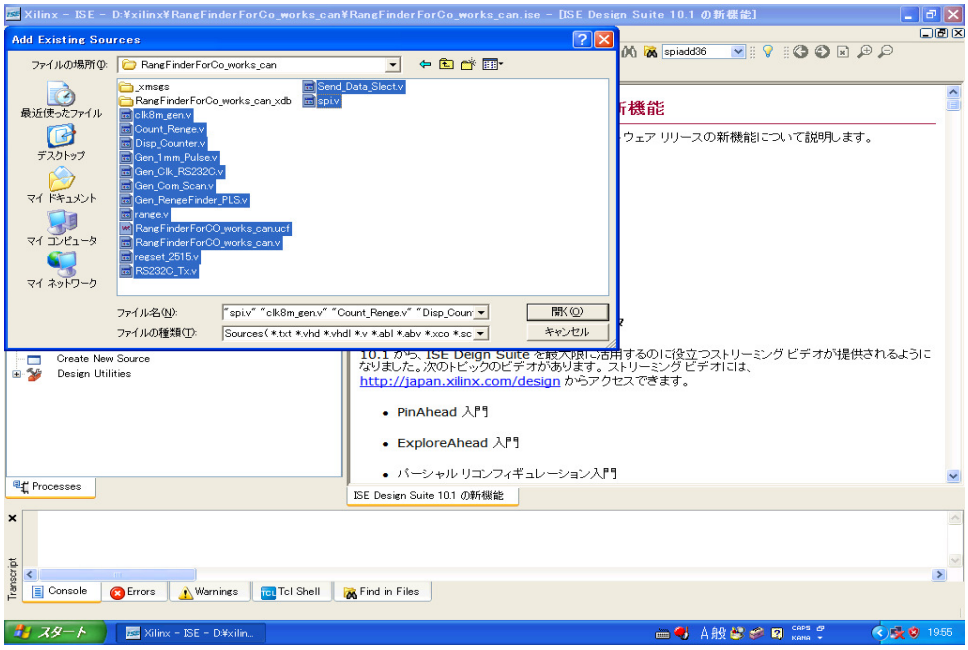

図２．１０

Project に加えるソースを選択し、開くをクリックします。

.ucf と.v を選択します。

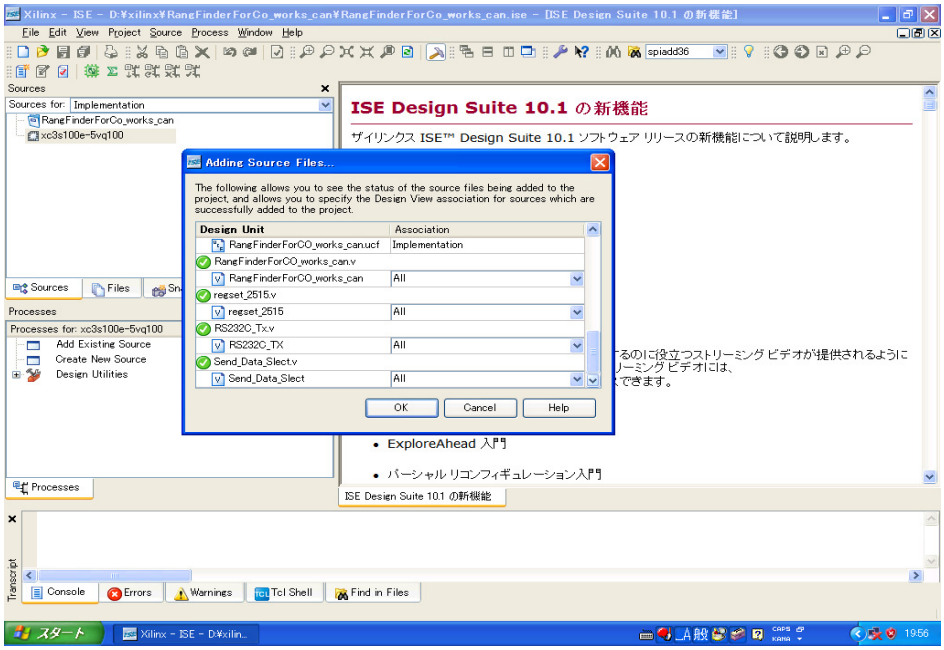

図２．１１

OK をクリックします。

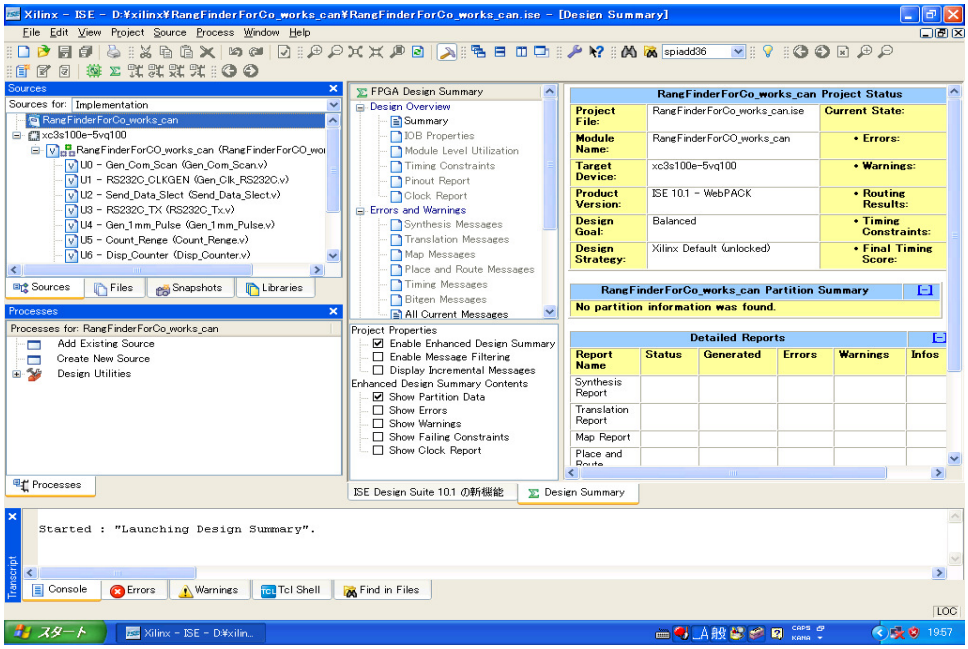

図２．１２

ここで、プロジェクトの完成です。

このプロジェクトの回路データを生成するため、Source のプロジェクト名をクリックします。

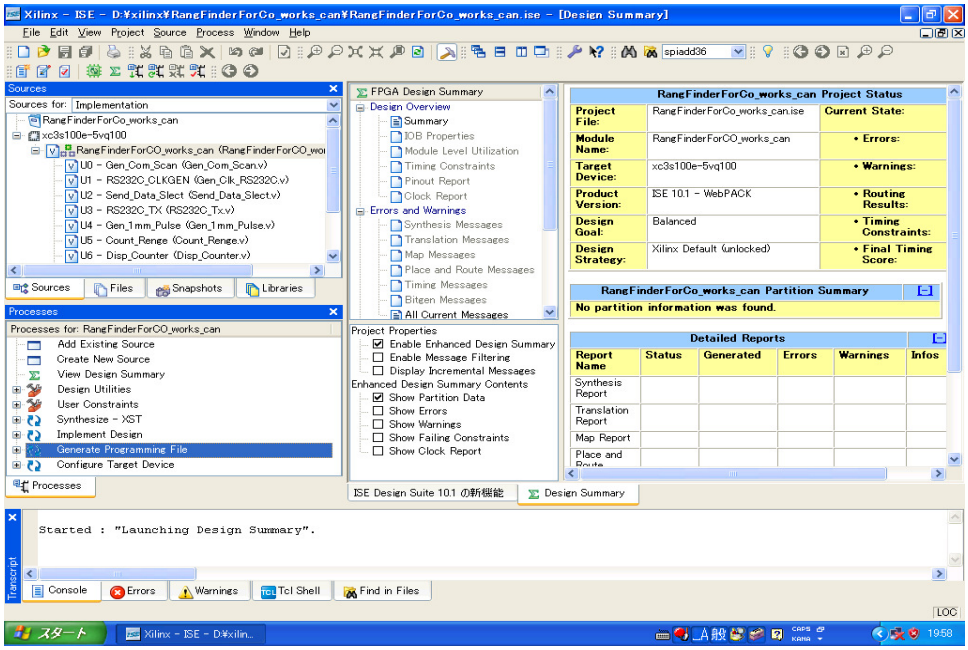

図２．１３

Processes の Generate　Programming File をクリックします。

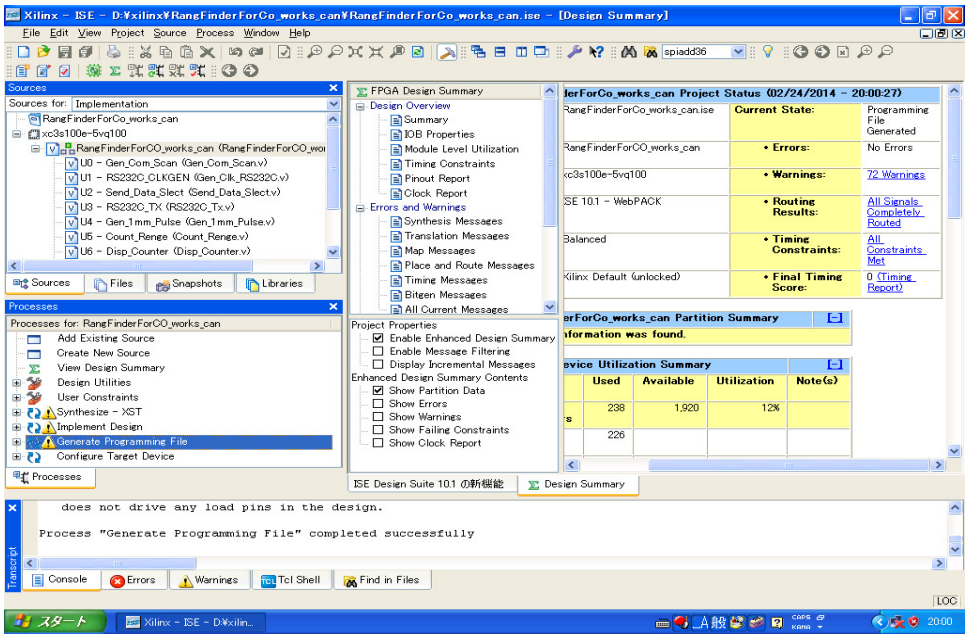

図２．１３

Process　“Generate Programming File”　completed successfuly

回路データの生成に成功しました。

ここからはデバイスへの書き込みです

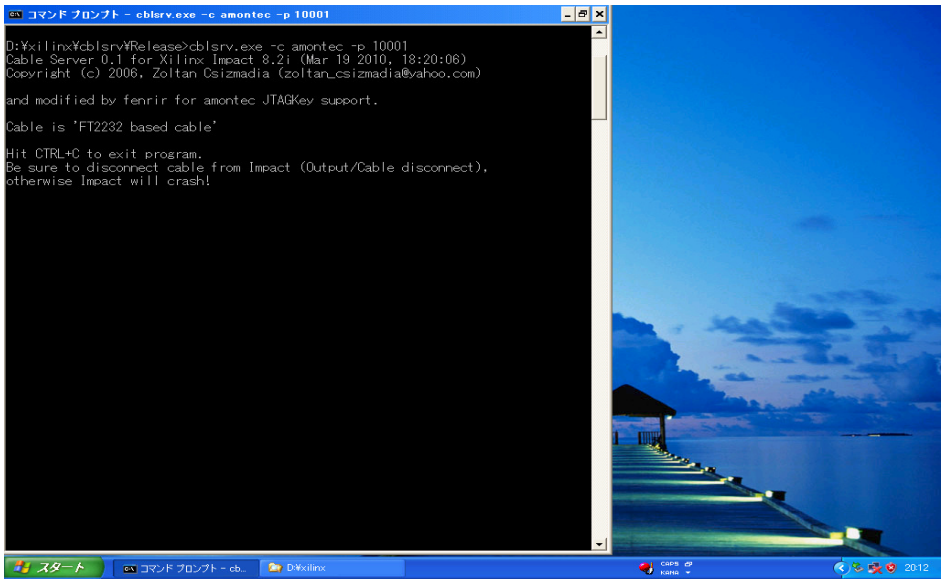

図２．１４

コマンドプロンプを起動し、cblsrv と FT2232 のドライバのあるディレクトリに移動します。Cblsrv.exe -c amontec -p 10001 と入力し、起動します。

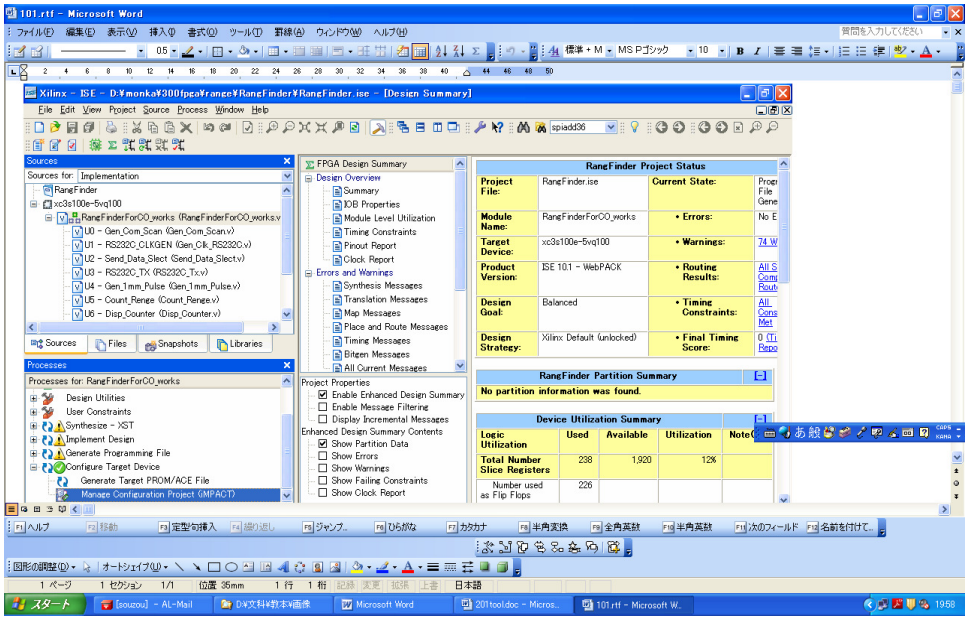

図２．１５

Source 画面で top module を選び、Processes 画面の Configure Target Device の iMPACT をクリック。

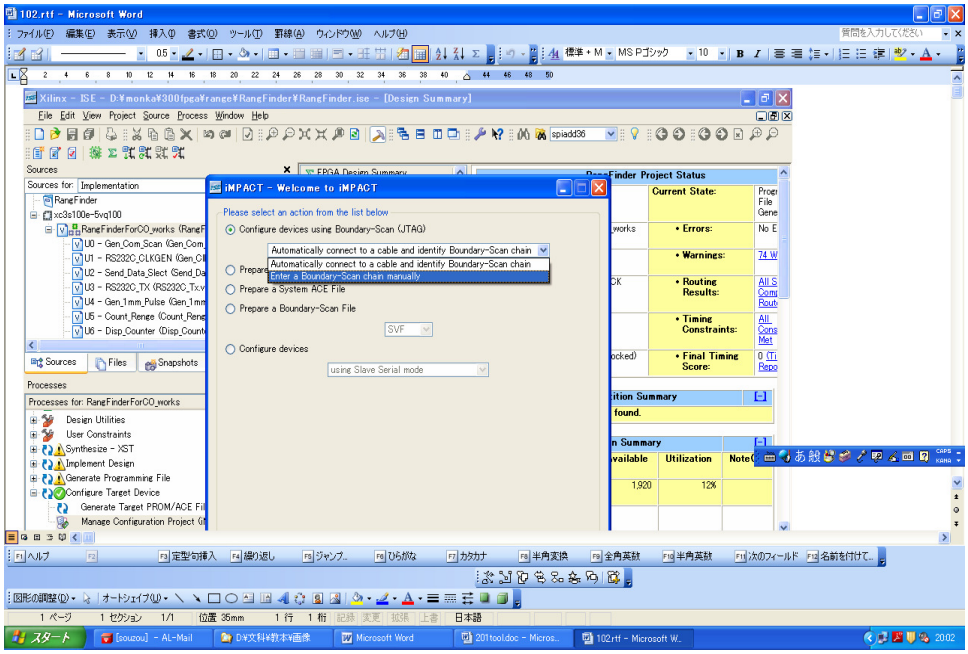

図２．１６

Enter a Boundary-Scan chain manually を選択して、Finish をクリック。

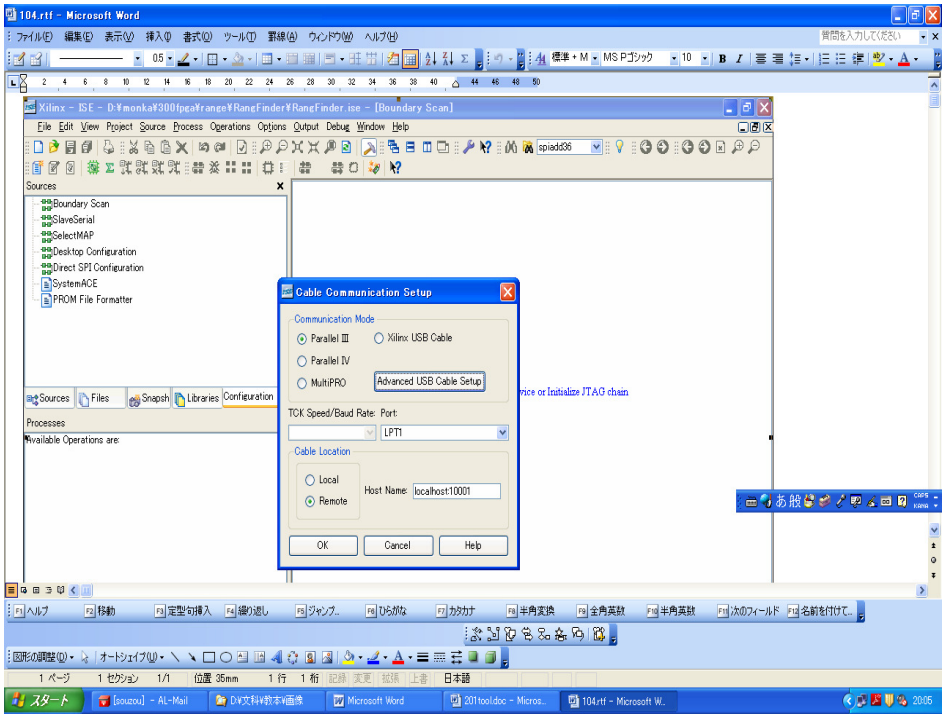

図２．１７

Parallel Ⅲを選択。

Remote を選択。 localhost:10001 を入力し、OK をクリック。

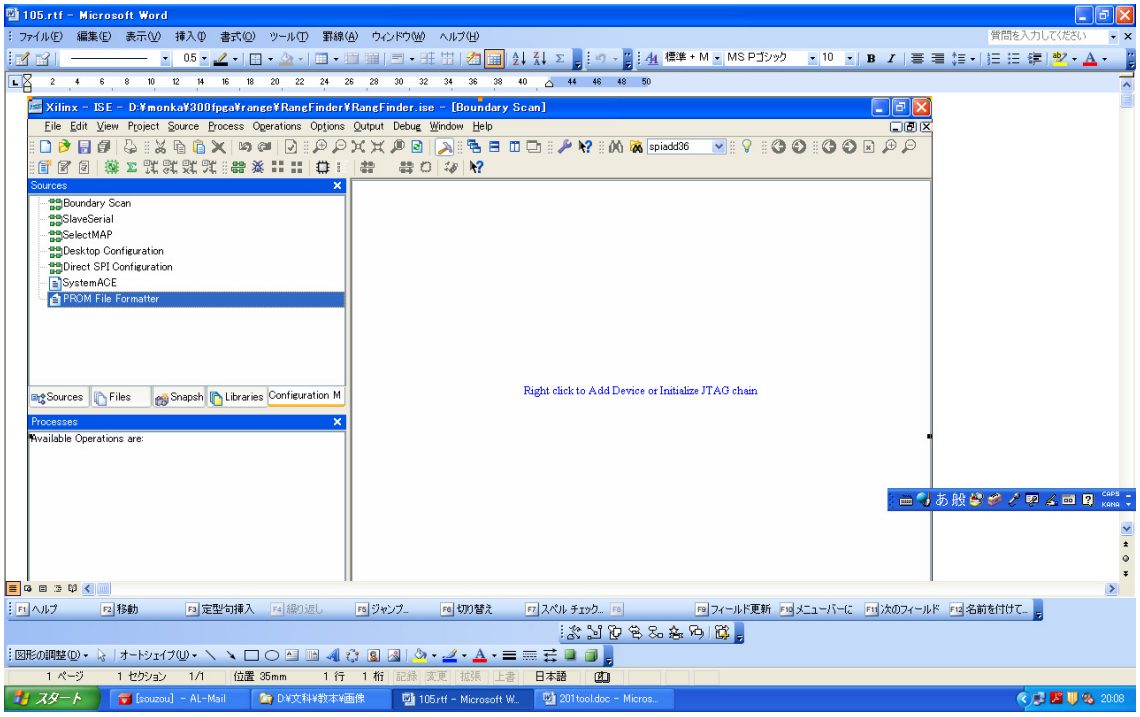

図２．１８

PROM File Formatter をクリック。

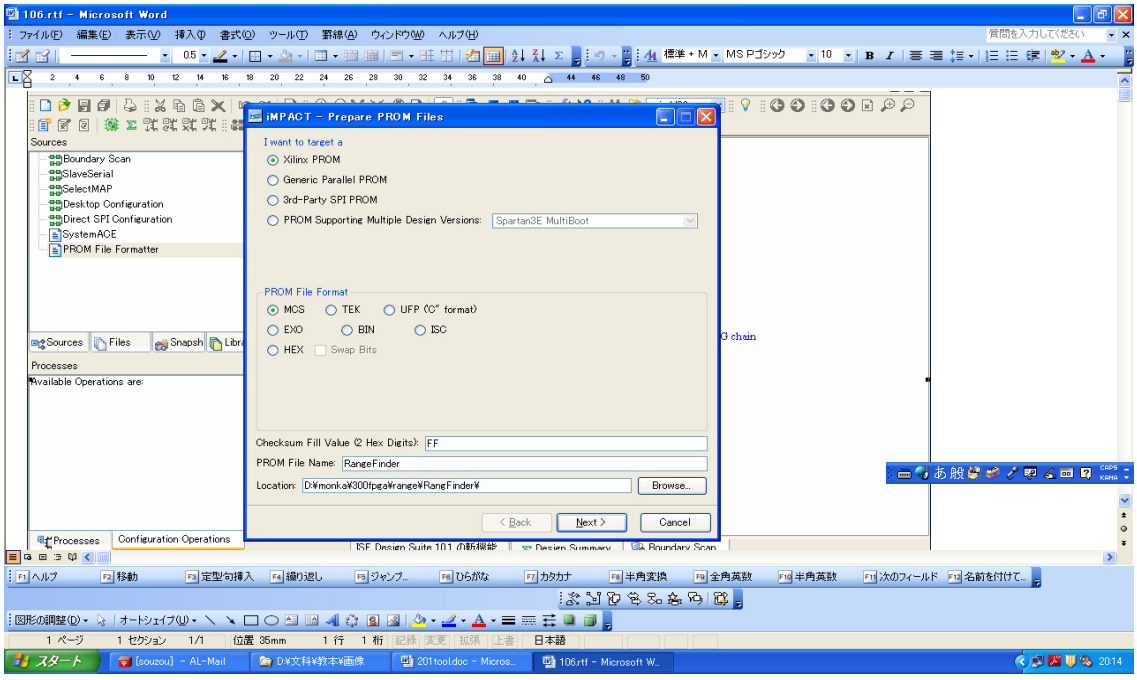

図２．１９

Xilinx PROM を選択。mcs を選択。PROM File Name を入力。

Next をクリック。

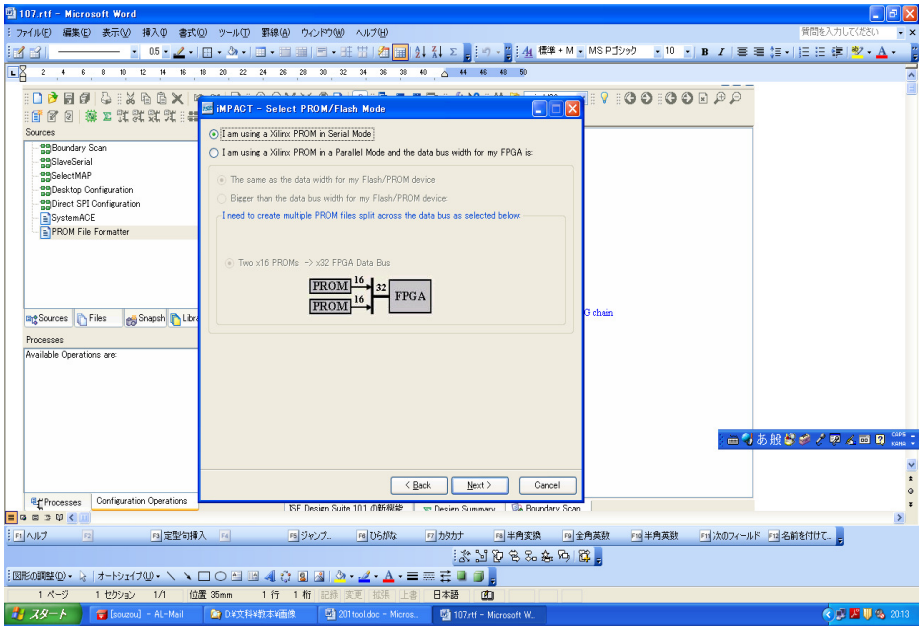

図２．２０

I am using a Xilinx PROM in serial mode を選択。

Next をクリック。

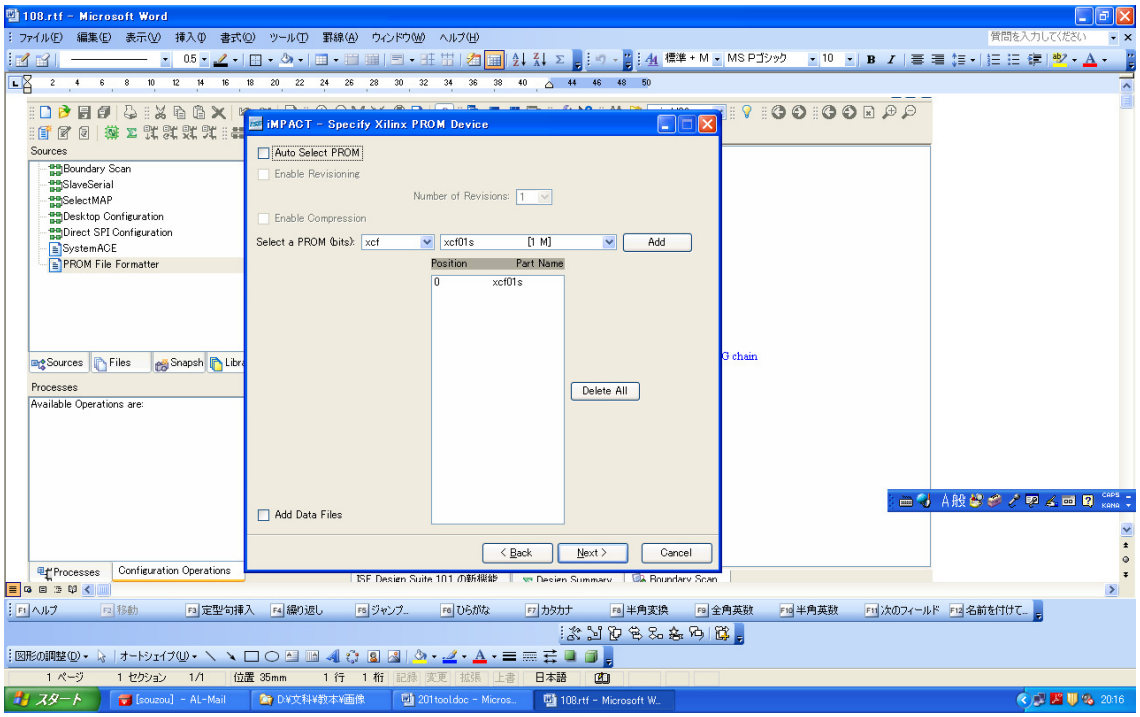

図２．２１

xcf01f を選択し、Add をクリック。

Next をクリック。

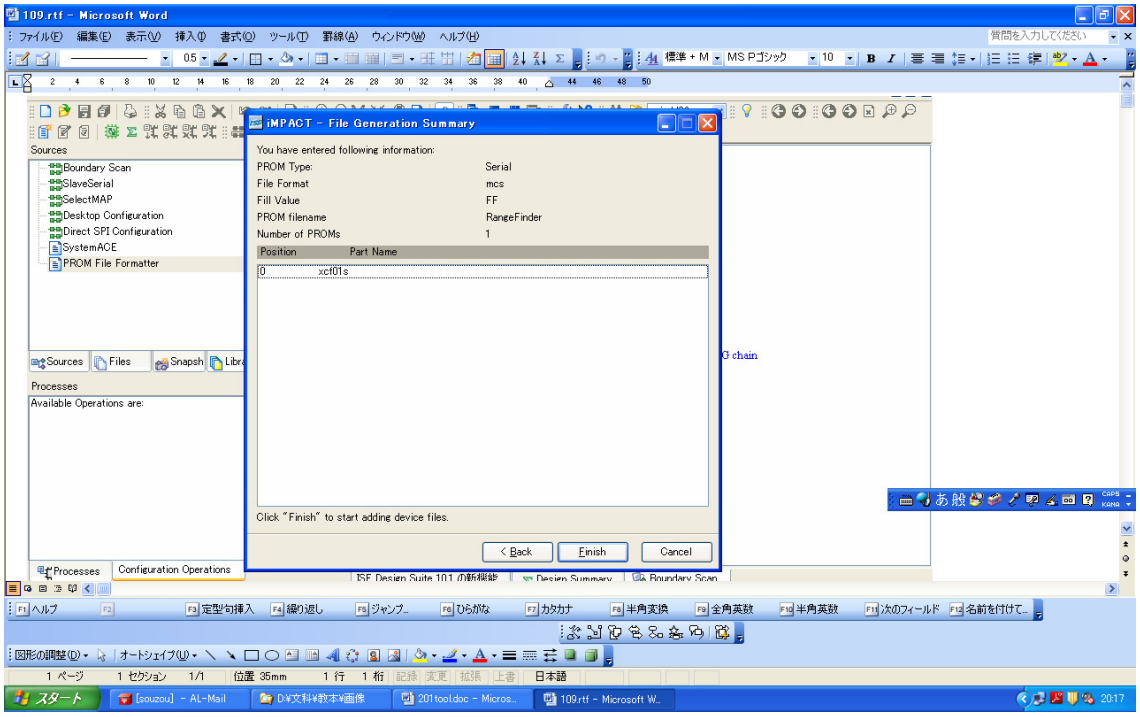

図２．２２

Finish をクリック。

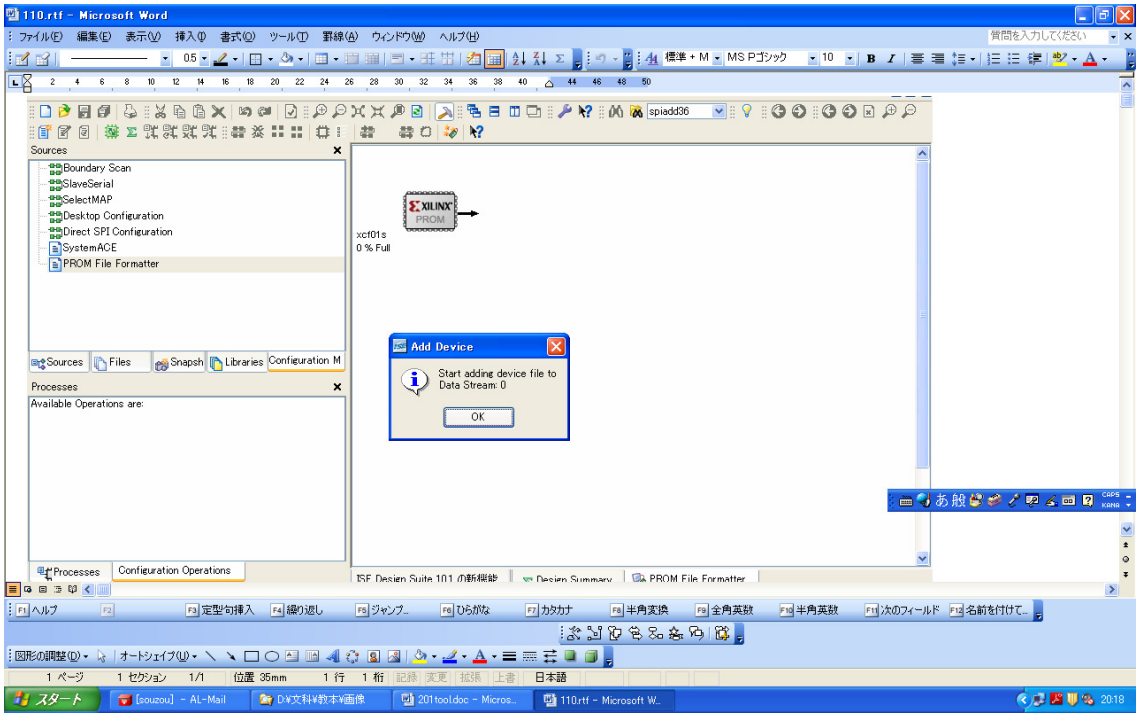

図２．２３

OK をクリック。

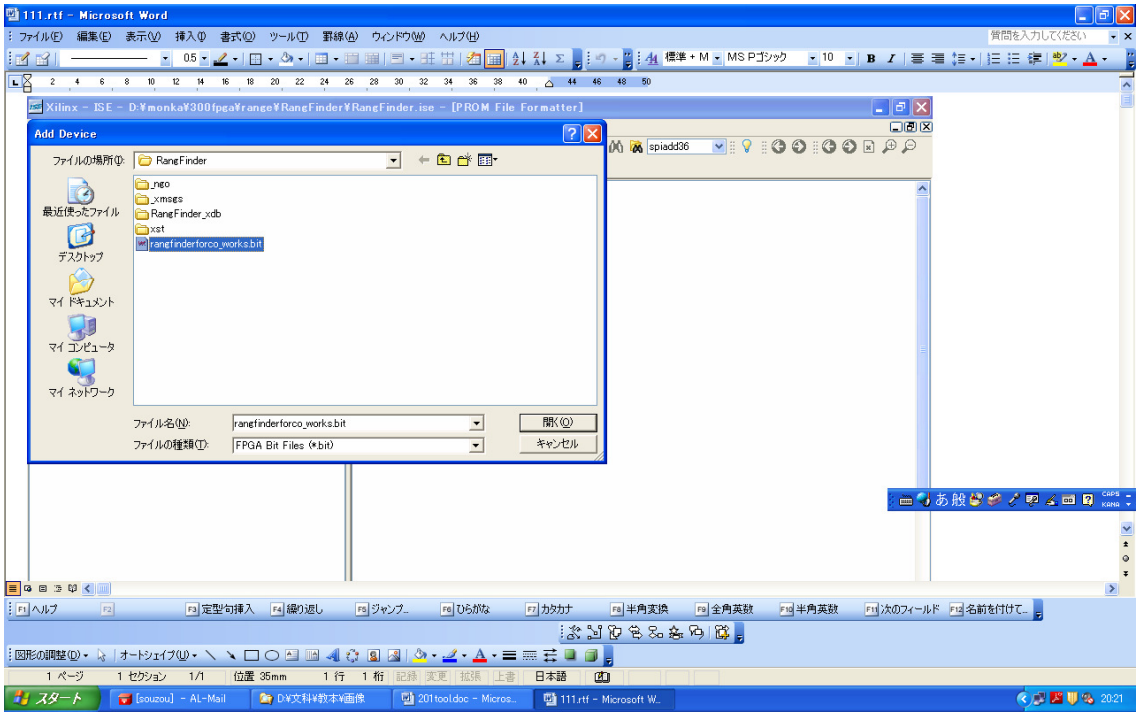

図２．２４

.bit ファイルを選択し、開くをクリック。

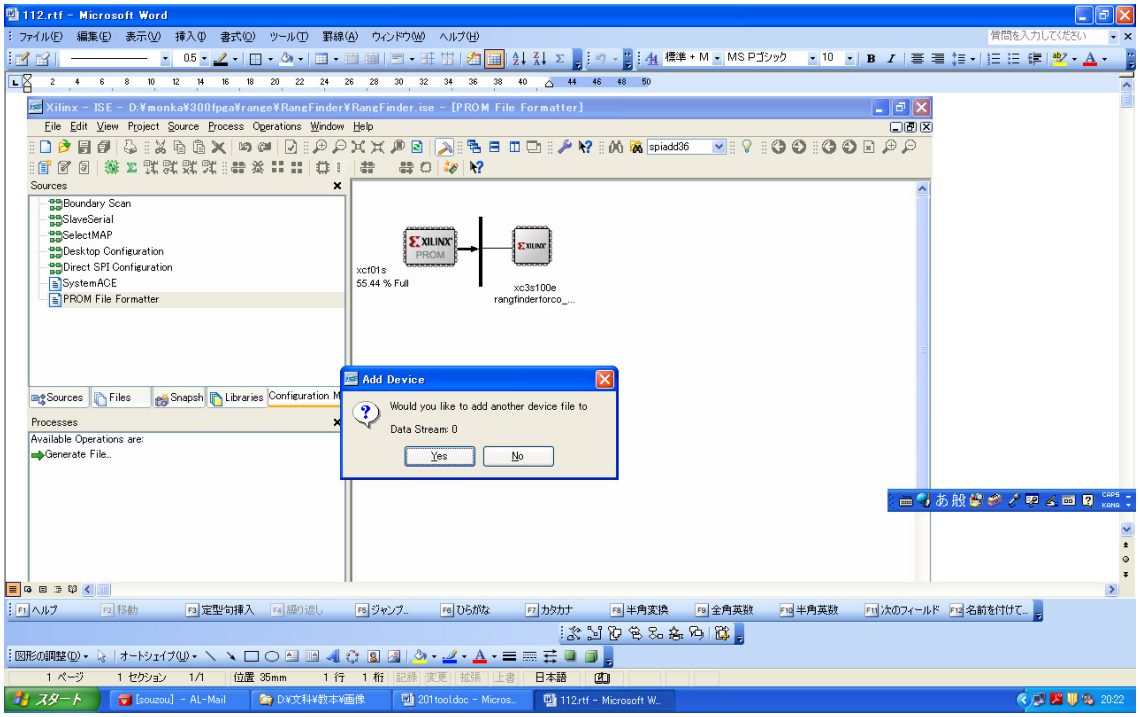

図２．２５

No をクリック。

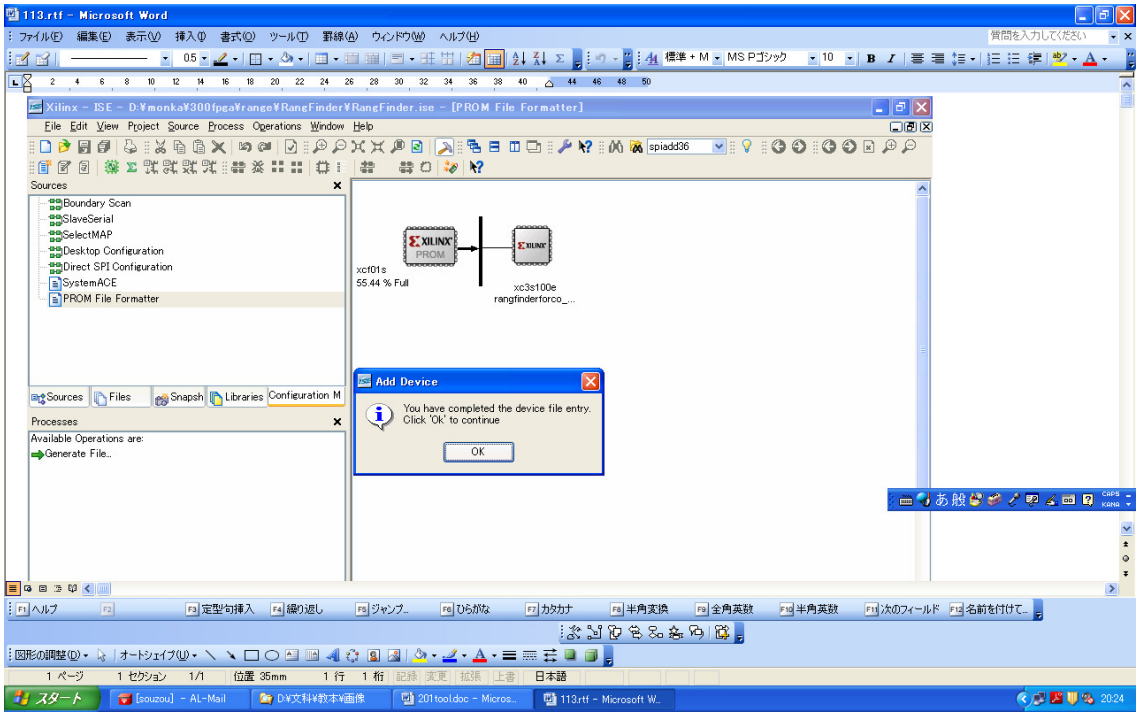

図２．２６

OK をクリック。

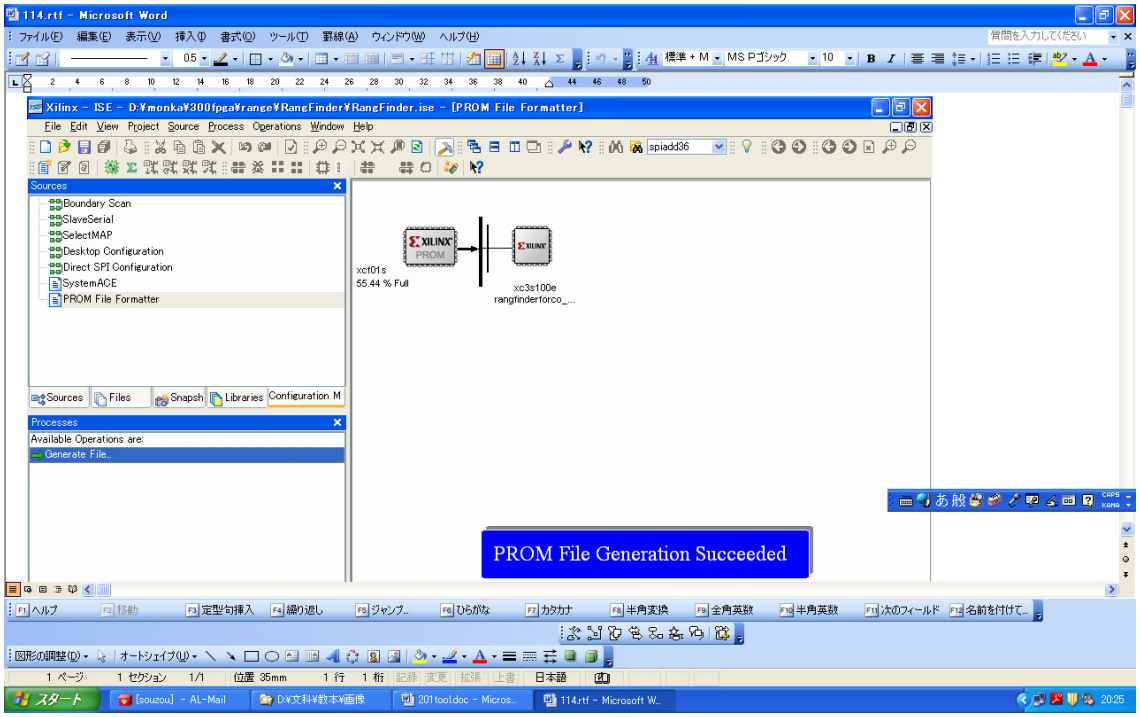

図２．２７

Generate File をクリック。

PROM File Generation Succeeded が表示されることを確認。

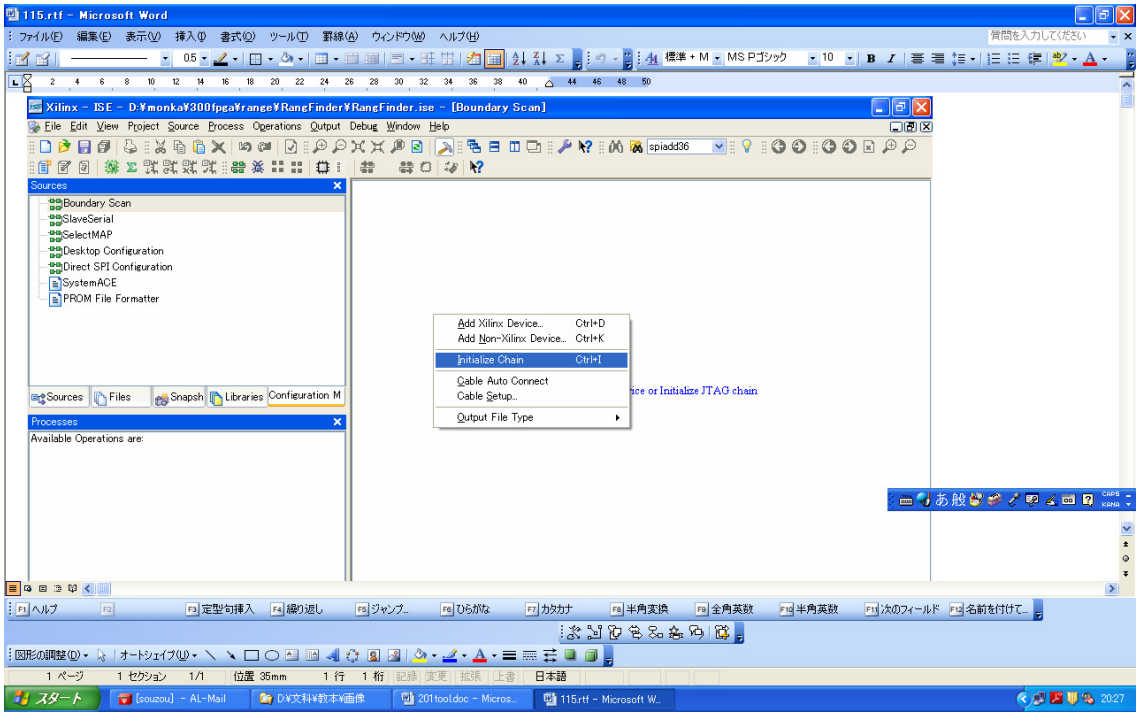

図２．２８

Boundary Scan を選択し、Initialize Chain をクリック。

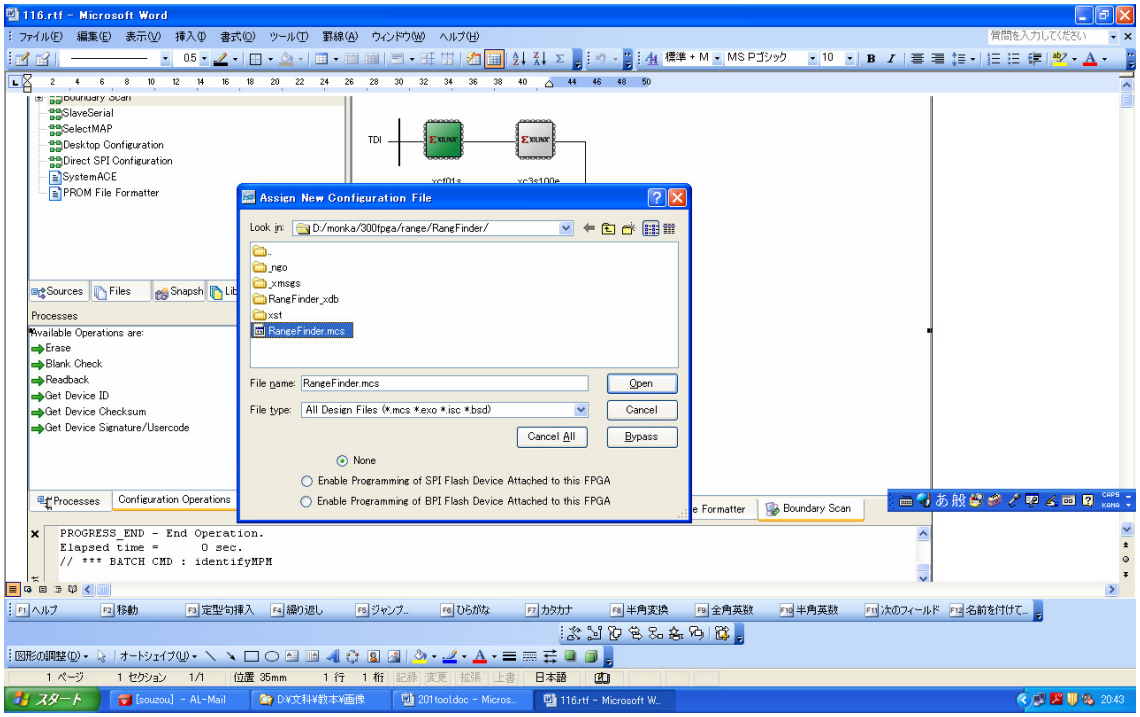

図２．２９

.mcs を選択し、Open をクリック。

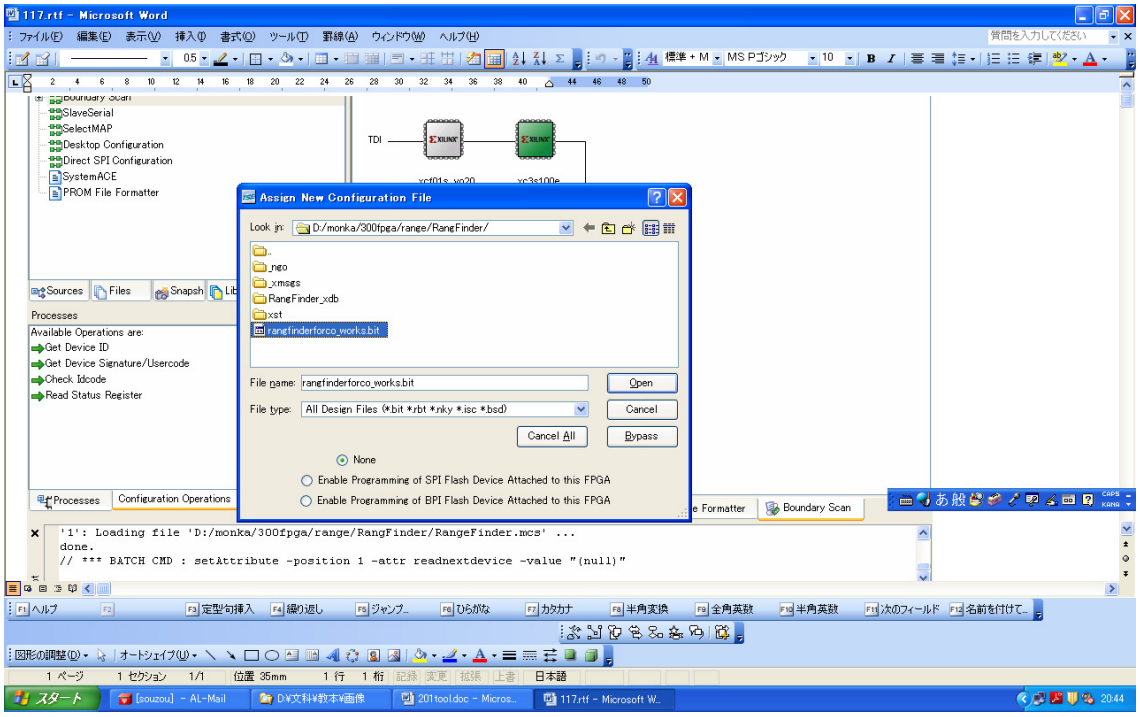

図２．３０

.bit を選択し、Open をクリック。

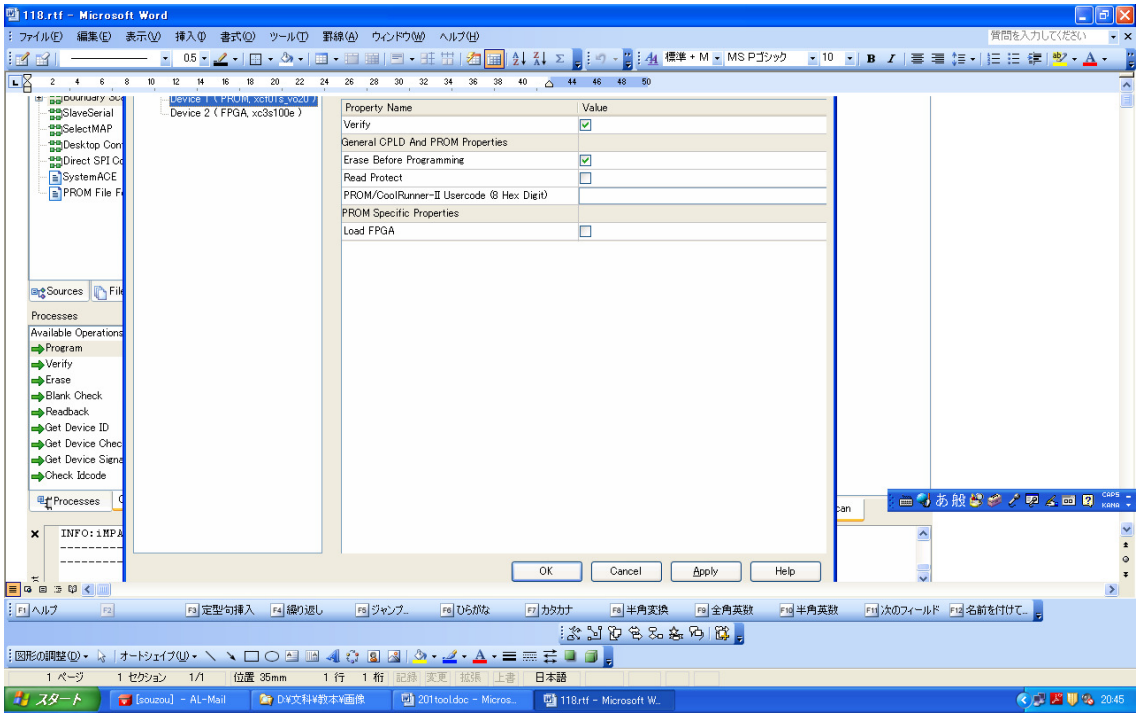

図２．３１

Device1 を選択し、OK をクリック。

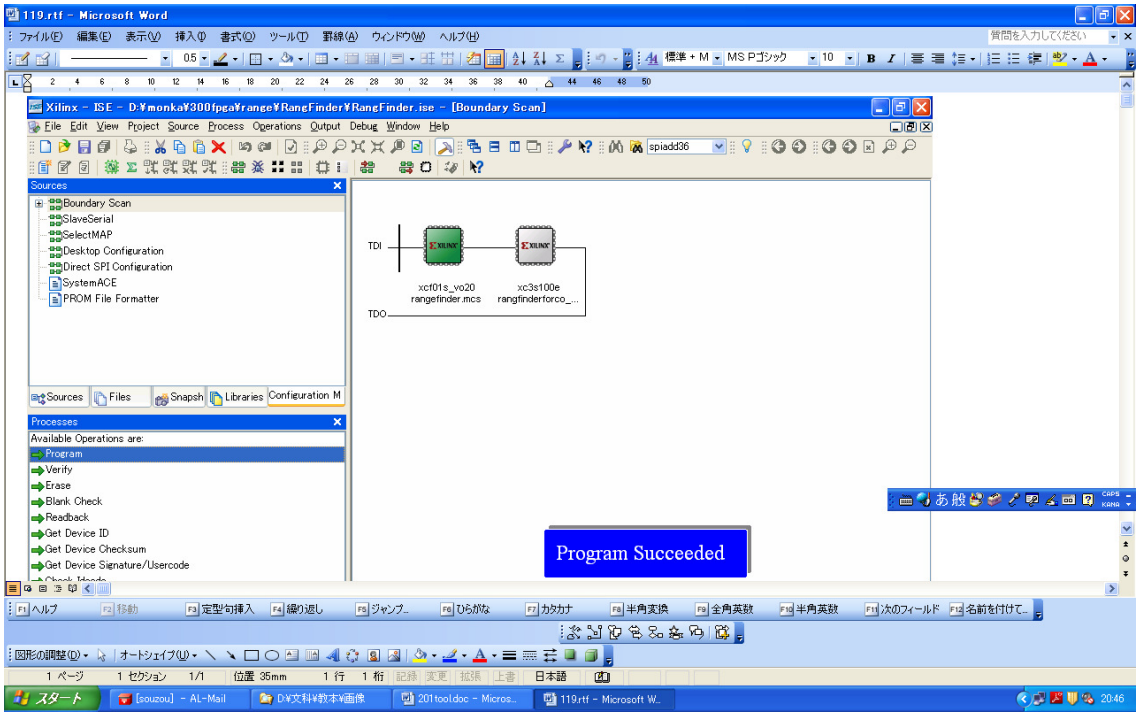

図２．３２

Program をクリック。

Program Succeeded が表示されることを確認。

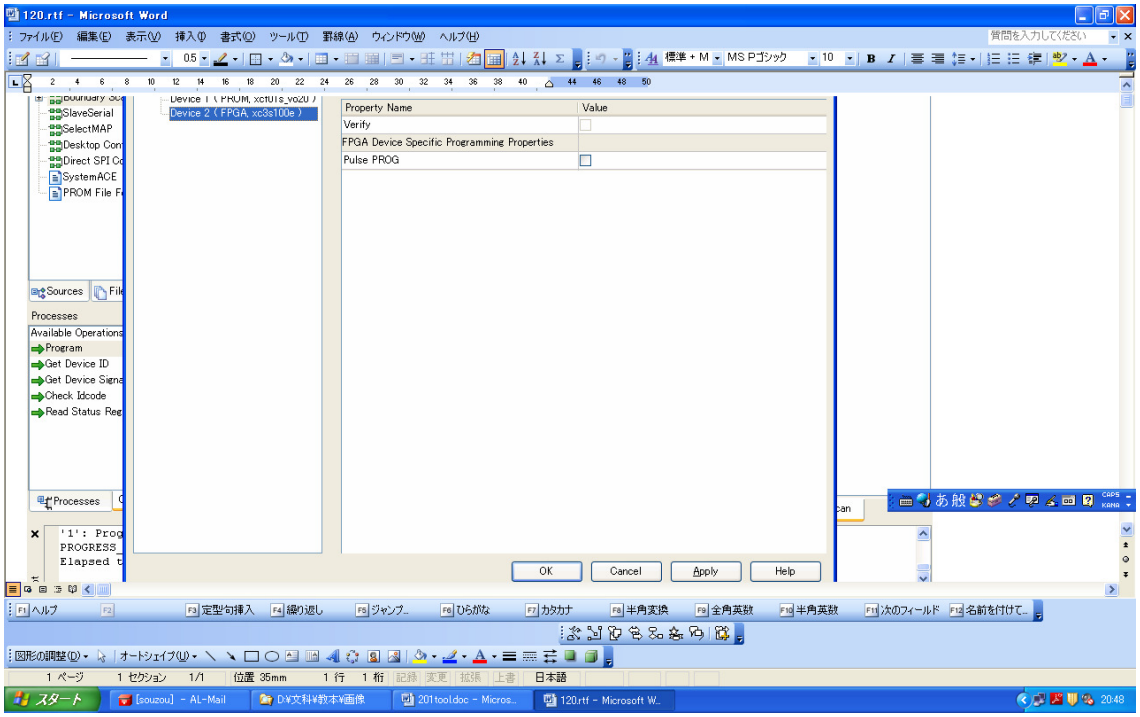

図２．３３

Device2 を選択し、OK をクリック。

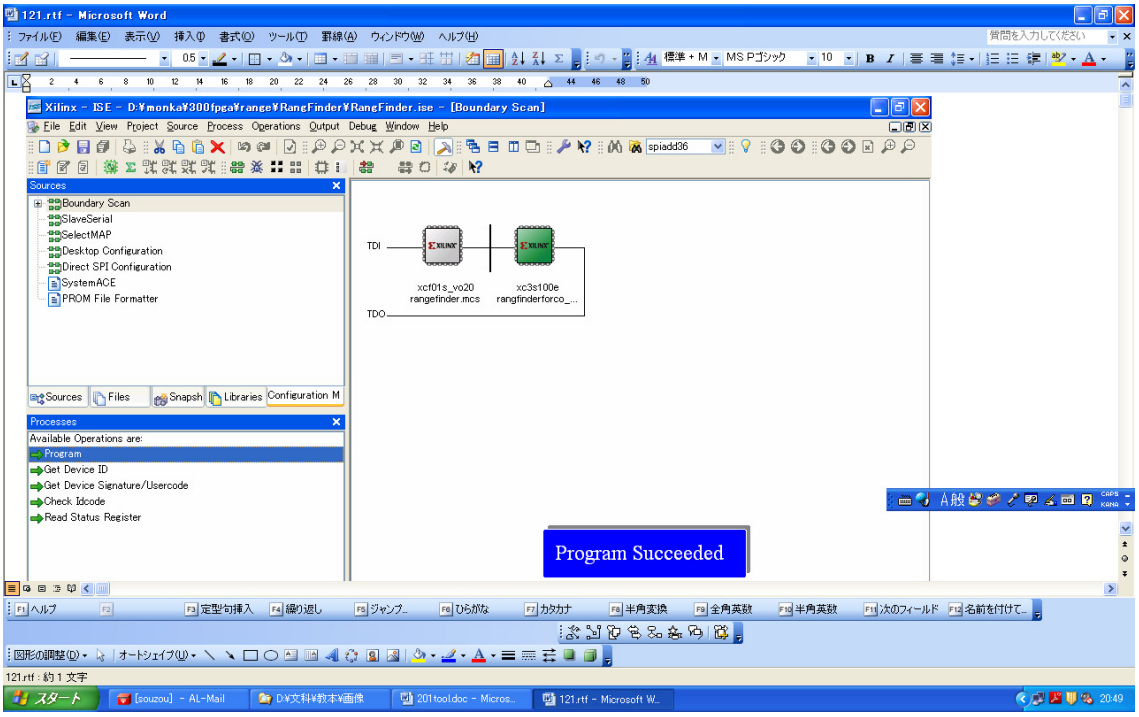

図２．３４

Program をクリック。

Program Succeeded が表示されることを確認。

これで、PROM、FPGA 双方への書き込みが完了しました。

PROM に書き込むのは、電源を断しても書き込んだデータを保持するためです。

デバッグで、すぐに書き換える場合は、.mcs の書き込みをスキップして.bit を FPGA に書き込むだけでも構いません。

３．ＦＰＧＡ搭載機能

【コラム１】組み込みについて少々

FPGA 搭載機能について説明する前に、組み込みについて考えてみましょう。組み込みシステムとは、ハードウェアとソフトウェアの融合です。人間に例えると、ハードウェアは身体で、ソフトウェアは魂ではないでしょうか。どちらか片方だけでは、組み込みシステムは正常に動作しません。健全な身体に健全な魂が宿ると思います。どちらか片方の性能が良くても片方の出来が悪いと、良い組み込みシステムはできません。そう考えると組み込みのハード技術者はソフトの事もある程度知らなければいけません。逆も真なり、ソフト技術者もハードの事をある程度知らなければいけません。自分が設計したハードウェア（ソフトウェア）のデバッグを、ソフト（ハード）技術者への依存なく、自信を持って切り分けてデバッグできると良いですよね。

本書が、ソフトウェア編と１対となり活用されることを望んでいます。

ところで組み込み技術者にとって一番大切な事は何でしょうか。制御対象を知ること、私はそう考えます。例えば自動車でエンジン制御の ECU を設計する場合、内燃機関を知らずに設計できません。サイクルのうちどこに圧縮上死点があるかとか、どこで燃料を噴射するとか、カムプロフィールがどうとか、空燃比がどうしたとか、知るべきことが沢山あります。無線の組み込みの場合、スプリアスがどうした、歪みがどうした、送信出力誤差が何 dB とか、きりがありません。一つの分野の組み込みを長くやって行くと、ハード、ソフトの垣根ではなく、システム全体に精通して行くものです。

３．１　ＦＰＧＡ搭載機能概要

応用編で FPGA に搭載する機能は、CAN(Controller Area Network)のバスインタフェースです。

CAN のプロトコルに沿った CAN バスの直接的な制御は FPGA 外部に接続される CAN コントローラに依存しますが、FPGA がこの CAN コントローラを制御することで CAN バスの制御が可能になると理解して下さい。

FPGA と CAN コントローラは、SPI(Serial Peripheral Interface)で接続します。したがって、CAN コントローラを制御するための SPI モジュールを、CAN バスインタフェースの一部として今回の FPGA に搭載しています。

CAN と SPI の詳細については、後述します。

今回搭載する CAN バスインタフェースだけでは、CAN バスに転送するデータを生成できません。転送するデータは、基礎編で設計した超音波距離計のモジュールで生成されたデータを使用します。
モジュール構成は、超音波距離計が上位となり、下位に CAN バスインタフェースが接続されます。
CAN バスインタフェースは、SPI クロック生成モジュール、CAN コントローラ制御モジュール、SPI モジュール、距離データ収集モジュールで構成されます。
このうち、CAN バスインタフェースの主要モジュールとなる、SPI モジュールについては 3.2 章、CAN コントローラ制御モジュールについては 3.3 章で概要説明します。図３．１にモジュール間接続を示します。

図３．１　モジュール間接続

```
Top : RangFinderForCO_works_can
|__  U10 : clk8m_gen
|__  U20 : regset_2515
|     |__  U21 : spi
|__  U30 : range
```

３．２　ＳＰＩ

SPI の詳細については後述しますので、ここでは FPGA の搭載機能として SPI モジュールの概要を説明します。

今回のモジュール構成では、CAN コントローラ制御モジュールが上位となり、下位に SPI モジュールが接続されます。SPI は同期シリアルインタフェースのため、転送クロックが必要です。転送クロックは、SPI クロック生成モジュールから供給されます。

SPI モジュールは、CAN コントローラとの物理的接続機能を提供し、データ長が最大 3byte の可変長で設計しています。データ長は CAN コントローラの制御仕様から決定しています。転送データは上位モジュールから接続されます。

SPI は CAN の制御に必須のものではありません。今回使用した CAN コントローラのデバイス仕様に依存していると理解して下さい。

【コラム２】外部回路について少々

ハードウェアというと、プリント基板やその回路をイメージする人が多いのではないでしょうか。回路を構成するハード部品のうちソフト依存があるのがマイコンです。ハードのコアな部分として FPGA があります。そこで今回は FPGA に絞り込みハードウェアについて学びます。

では、FPGA の外部回路についても少々。

リセット回路について考えてみます。リセット信号は複数接続されるデバイスを初期化する信号です。リセットにより、マイコンでは内部回路が動作する前にレジスタの初期設定などが行われ、ジャンプベクターの最初の番地にプログラムカウンタがセットされます。しかし、一つのデバイスの初期化だけではなく、複数接続されたデバイスをシーケンシャルに起動する信号としてリセットは必要な信号です。

ところで、リセットは必ず論理ゼロでしょうか。この考え方は危険です。確かに回路に電位が生じないと論理１にならないので、論理ゼロのリセットは自然かもしれません。しかし、回路によっては論理１のリセットを要求するものもあるので、注意が必要です。

FPGA の場合、電源投入後 ROM から回路データが FPGA にロードされ回路が確定します。この動作をコンフィギュレーションと呼び、その間は FPGA の回路は記述どおりの動作をしません。FPGA 同様プログラマブルなデバイスに CPLD というのがあります。両者の大きな違いは、コンフィギュレーションの要否です。CPLD はパワーオンリセットと同時に記述どおりの動作を開始します。このようなところでも、リセットは重要になってきます。

次に、回路の基本、GND グランドについて考えてみます。

GND ってなんですか。測定器の GND をつなぐところとか答えられても、正確に答えられる初心者は意外と少ないかもしれませんね。

電気（電子）回路の動作の基本は電位差で、何ボルトとか皆さんが表現している値です。この差はどうやって求めるのでしょうか。そうです、GND と回路のある点との電位の差が、電位差です。したがって、GND は回路の基準で、電流が流れ込んだり、流れ出したりする基点と考えてください。

３．３　ＣＡＮコントローラインタフェース

今回のモジュール構成では、CAN コントローラ制御モジュールが SPI モジュールへのデータ接続機能を提供することで、CAN コントローラインタフェースを構築します。

CAN コントローラ制御モジュールは SPI モジュールと同期を取る必要があるため、SPI クロック生成モジュールが供給するクロックを使用します。

CAN コントローラの各レジスタに必要な要素を FPGA が書き込むことで、CAN バス制御が実行されます。CAN バスインタフェースの要素を一から全て FPGA でモジュール化することも可能ですが、それは複雑な回路となってしまいます。FPGA の搭載機能を絞り込むため、容易に入手できる CAN コントローラを外部デバイスに使用して、CAN バスインタフェースを実現します。

CAN コントローラレジスタ設定の詳細については、後述します。

４．ＣＡＮってなんだ

４．１　ＣＡＮの概要

CANは、車載される複数のユニットを接続するために開発されたプロトコルです。
Controller Area Networkの略称で、1986年ドイツの電装機器メーカー「ロバート・ボッシュ社」が開発したシリアル通信プロトコルです。

自動車に搭載される各種制御ユニットの事をECU(Electronic Control Unit)と呼び、自動車に搭載される電気系統の部品の事を総称して電装品(系)と呼ぶ場合があります。
これらECUを接続するめには多量のワイヤーハーネスが必要となるため、その削減がCAN開発の目的のひとつでした。車種にもよりますが、現在では1台で100種類近いECUを搭載している自動車もあり、CANはその接続に欠かせないネットワーク規格のひとつになっています。

この様に、当初は自動車用に開発されたプロトコルでしたが、その後ISOにて標準規格化され、現在では自動車以外の機器にも応用されています。

４．２　ＣＡＮの仕様

CAN は 1986 年に公表されたバージョン 1.0 から改版され、現在は CAN Specification Version2.0 Part-B（以下 CAN2.0B）まで改版されています。また、ISO11519 認証、ISO11898 認証、ISO11898 改定、と変遷し、拡張フォーマットの追加が行われました。CAN の物理層の規定はビットタイミング、同期化の手順まででしたが、バスの電気的特性等が ISO で規定されました。
OSI 基本参照モデルに対する CAN2.0B の規定範囲は、物理層（１層）の上位、データリンク層（２層)、トランスポート層（４層）の上位です。

CAN の概略仕様

表４．１

項目	仕様
伝送方式	半二重シリアル通信
ネットワークトポロジー	マルチマスタバス型式
アクセス制御方式	CSMA/NBA
同期方式	調歩同期位相補正付
データ長	可変長最大 8byte
エラー検出方式	CRC
符号化方式	NRZ ビットスタッフ付
ビットレート	最大 1Mbps(ISO11898)

ここで、CAN の符号化方式について説明します。

NRZ（Non Return to Zero）はゼロ復帰なしの符号化方式です。 ビットとビットの間に符号化ビットを挿入せず、ビットの論理を伝送する方式です。この方式だけを考えると当たり前に聞こえますが、伝送の符号化方式にはビット間に必ず電位ゼロを挿入する RZ や、電位の組み合わせでビットの論理を表す CMI などがあるため、ゼロ復帰なしと定義されています。

NRZ の長所は、データビット長そのものが伝送されるため、ビットレートがそのまま伝送容量になるところです。RZ や CMI はビットの論理を表現するためビット間に符号化ビットが挿入されるため、ビットレートの半分しか伝送できません。

ではなぜ、RZ や CMI はビット間に符号化ビットを挿入するのでしょう。これは同期はずれ（Los Of Frame）を防ぐためです。連続した０または１が伝送されると、フレーム同期が取れなくなる場合があるためです。

こういった事から、CAN ではビットスタッフが採用されています。ビットスタッフは各ビット間に符号化ビットを挿入するのではなく、連続した論理が続いた場合に符号化ビットを挿入するものです。

CAN では連続した論理が 5 回繰り返されると、その論理を反転したビットを挿入する、ビットスタッフを行います。

次にCANのバスの値と伝送路について、説明します。

バスの値は、論理０と１に対応する２値で、ドミナントとレセシブと呼ばれます。ドミナントが優先的なのに対しレセシブは受容的です。各ユニットがレセシブであればバスもレセシブですが、ひとつでもドミナントが出力されるとバスもドミナントになります。

伝送路は通常 2 線式の平衡伝送です。CAN トランシーバは CANHigh と CANLow を送信します。低速バスに限り、1線でも可となっています。

アービトレーション（バス調停）について説明します。

CAN のネットワークトポロジーはマルチマスタです。バスがレセシブであれば、どのユニットでも送信を開始します。レセシブを検出して送信する Carrier Sense Multiple Access です。そこで、バスが衝突した場合、アービトレーションを行います。各メッセージの ID で調停を行い、負けたユニットは送信を停止します。

４．３　ＣＡＮのプロトコル

CAN バスは、表４．２の 4 種のフレームと、直前のフレームとの間を分離するインターフレームスペースと呼ばれるひとつのスペースで制御されます。

表４．２

Frame	Description
データフレーム	メッセージ送信
リモートフレーム	他のユニットへの送信要求
エラーフレーム	エラー通知
オーバーロードフレーム	受信未完了などを他のユニットへ通知

データフレームについて、概要を説明します。

スタートオブフレームと呼ぶ、ドミナントビットで開始します。

アービトレーションフィールドと呼ぶ、11bit の ID(Identifier)、SRR、IDE、18bit の拡張 ID、RTR が順に送信されます。

アービトレーションフィールドは、標準フォーマットと拡張フォーマットで異なります。先の説明は拡張フォーマットの場合です。

コントロールフィールドと呼ぶ、r1、r0、DLC が順に送信されます。DLC はデータ長を示します。コントロールフィールドも標準フォーマットと拡張フォーマットで異なります。先の説明は拡張フォーマットの場合です。

最大 64 ビットのデータフィールド、CRC シーケンス ACK フィールドと続き、7 ビットのエンドオブフレームで終了します。

CAN の ID について少し。

CAN の ID にはバスの優先権が含まれ、調停の判断となります。

また、CAN の ID はユニットの識別子ではなく、データの意味の識別子です。つまり、エンジン回転数なのか、ワイパー制御情報なのかということを識別します。OBDⅡ（On-board diagnostics）も ID から情報を取得します。

各フレームの構成、詳細については MCP2515 のデータシートを参照下さい。

【コラム３】CAN ドライバーについて少々

今回の CAN バス制御では、FPGA の外部デバイス MCP2515 という CAN コントローラを使用します。

では、CAN コントローラだけあれば CAN バスに接続できるのでしょうか。答えは No です。

CAN バスは通常 2 線式の平衡伝送路です。したがって、レベル変換するための、CAN ドライバーが必要です。CAN ドライバーまで知らなくても FPGA の回路記述はできるかもしれません。しかし、そこまで知らないと接続部のハードデバッグはできません。CAN コントローラが対応するのがデータリンク層（２層）とトランスポート層（４層）に対し、CAN ドライバーは物理層（１層）に対応します。

５．ＳＰＩってなんだ

５．１　ＳＰＩの概要

SPI（Serial Peripheral Interface）は、米国モトローラ社(現在のフリースケール・セミコンダクタ社)が提唱した、同期式シリアルインタフェースです。変遷することで伝送速度が向上し、現在では 10Mbps のビットレートに対応するデバイスが出ています。広範囲の分野で使用されているシリアルインタフェースで、片方向の通信であればクロック、データ、スレーブセレクトの最低 3 線で接続可能です。

５．２　ＳＰＩの仕様

SPI はマスタースレーブで接続されます。

クロックはマスター側からスレーブ側へ供給される 1 本だけです。したがってスレーブ側からの送信も、マスター側からのクロックに同期して行われます。

通信（メッセージ）の開始は、スレーブ側デバイスの CS(Chip Select)を、マスター側がスレーブセレクトを制御し選択するところから始まります。

データはマスター側からの送信を MOSI（Master Out Slave IN)、スレーブ側からの送信を MISO（Master In Slave Out）と呼び、これらのデータはクロックの立ち上がりもしくは立ち下りで確定されます。

データ確定の極性を SPI モードと呼び、送受同じモードでなければ通信できません。

６．ＣＡＮコントローラを使ってみよう

６．１　MCP2515 の概要

MCP2515 は、CAN V2.0B を内蔵した、マイクロチップテクノロジ社の CAN コントローラです。CAN バスとインタフェースする機能と、その制御レジスタとのインタフェースとして SPI を内蔵しています。

CAN バスの制御は､SPI により MCP2515 の内部レジスタの設定をすることで実現されます。今回の FPGA とのインタフェースも、この SPI で実現します。

MCP2515 の SPI の転送レートは、最大 10MHz の高速度 SPI です。

MCP2515 の CAN バス制御は､転送信号をビットスタッフ付の NRZ に符号化、CRC を生成、全ての CAN フレームを構成し、アービトレーション、エラー検出など､CAN バス制御に必要な機能を満足しています。そのため､CAN バス制御用のレジスタには、いろいろな種類があります。

CAN バスへのアクセスの仕方で、設定が必要なレジスタも変わってきますので、全てのレジスタを設定する必要は必ずしもありません。CAN を使いこなす過程で、設定が必要なレジスタも変わると理解して下さい。また、レジスタは MCP2515 に電源を投入することもしくはリセットをかけることで、初期化されますので、デフォルト（初期状態）のまま使用する場合も設定の必要はありません。

ここでは、今回 FPGA に搭載した設定機能に絞って、MCP2515 のレジスタを表６．１に示します。

MCP2515 には５つの動作モードがありますが、今回使用するのは、コンフィギュレーションモードと通常モードの 2 種です。

MCP2515 には２個の受信バッファと３個の送信バッファがありますが、今回使用するのは送信バッファ０の１個です。

表６．１　MCP2515 主要レジスタ

Address	Register	Description
0x0F	CANCTRL	MCP2515 動作モード設定
0x28	CNF3	CAN ビットタイムセグメント設定
0x29	CNF2	CAN ビットタイムセグメント設定
0x31	TXB0SIDH	CAN Base ID の設定
0x32	TXB0SIDL	CAN Base ID / 拡張 ID の設定
0x33	TXB0EID8	CAN 拡張 ID の設定
0x34	TXB0EID0	CAN 拡張 ID の設定
0x35	TXB0DLC	CAN フレーム/DLC の設定
0x36 - 0x3D	TXB0D0 - TXB0D7	CAN バスへの 8yte 送信データバッファ

SPI で転送するデータにはレジスタ設定のほかに､MCP2515 に対する命令セットが９種ありますが、今回使用するのは、書き込み命令とメッセージ送信要求命令の 2 種です。

今回使用する MCP2515 SPI 命令セットを、表６．２に示します。

表６．２　MCP2515 SPI 命令セット

Instruction	CODE	Description
書き込み	b00000010	レジスタへの書き込み命令
RTS	b10000nnn	送信バッファのメッセージ送信要求 nnn で送信バッファを指定する

RTS は、メッセージ送信要求命令

送信バッファ０のメッセージ送信の場合、8'b10000001

６．２　ＳＰＩの構築

SPI は SPI モジュールで構築されます。

SPI モジュールのモジュール間接続

```
Top : RangFinderForCO_works_can
|__  U10 : clk8m_gen
|__  U20 : regset_2515
      |__  U21 : spi
```

上図のように、spi は上位モジュール regset_2515 から呼び出されます。regset_2515 は Top モジュール RangFinderForCO_works_can から呼び出されます。
また、spi に供給される SPI 転送クロックは、Top モジュールから呼び出される clk8m_gen で生成されます。したがって、SPI 転送クロックの接続は Top モジュールで行われ、spi の上位モジュール regset_2515 から供給されることになります。

６．２．１　トップモジュールの説明

モジュール名　　：　RangFinderForCO_works_can
モジュールの機能　：　トップモジュール

RangFinderForCO_works_can のうち、SPI モジュール接続に関連する部分を抜粋して説明します。

```
module RangFinderForCO_works_can(
        CLK,
        DATA_IN,
        TRG_EN,
        DATA_EN,
        TRG,
        //PLS_1mm,      //2013 edition
        LED,
```

```
COM,
SEG7,
TX_SELECT,
RxD,
TxD,
SCK,
SO,
CS
);
```

上記はモジュール宣言とポート・リストです。
トップモジュールには、下位の各モジュールの呼び出しとFPGA外部に入出力する外部端子の接続を記述します。したがって、そのポート・リストは、FPGA外部に入出力する外部端子に接続される信号になります。

```
input   CLK;          // Input Master clock

      .
      .
      .

//2013 edition
output SCK;           //SPI Clock
output SO;            //SPI MOSI
output CS;            //SPI Slave select
```

上記は、SPIモジュールとの接続に関連する信号のポート宣言です。

CLK　：　FPGA外部から供給される動作の基準となるクロックです。
応用編の学習ボードでは、33.333...MHzです。
基準のクロックの事を、マスタークロックとかグローバルクロックと呼ぶことがあります。本書もその呼び方で記述します。

SCK　：　SPIのデータ転送クロックです。FPGAがマスターなので、FPGA内部で生成し外部に出力します。SPIの転送クロックの速度はインタフェースするデバイスの仕様に依存します。

FPGAとインタフェースするMCP2515の転送クロックは最大10MHzですから、それ以下の周波数であれば接続可能です。今回は、マスタークロック33.333...MHzを4分周した8.333...MHzとしました。

SO　：　SPIの転送データです。
このデータがMCP2515を制御し、その結果CANバスが制御されます。FPGAがマスターなので、SPIのMOSI(Master Out Slave IN)となり、FPGA内部で生成し外部に出力します。

・
・
・

```
wire clk8m;   //SCK 8.33MHz
wire cs20;    //SPI slave select
wire so20;    //SPI MOSI
wire [39:0] rengedt;//range data 1byte SOF + 6byte data
```

上記はモジュール内部信号のネット宣言です。

clk8m　：　SPIのデータ転送クロックです。下位モジュールclk8m_genで生成された信号です。

cs20　：　SPIのスレーブセレクト信号です。MCP2515のチップセレクトに接続します。SPIモジュールspiで生成します。

so20　：　SPIの転送データMOSIです。MCP2515のデータ入力に接続します。SPIモジュールspiで生成します。

[39:0] rengedt　　:　基礎編で使用した超音波距離計で生成されたデータです。40bitの信号で上位の8bitはマイコンボードとの通信プロトコルより、データの属性(BCD/ASCII)を示します。下位の32bitが距離データです。

・
・
・

```
//2013 edition SPI assign

      assign SCK = clk8m;
      assign CS = cs20;
      assign SO = so20;
```

上記は、回路の記述です。

clk8m_genで生成されたSPIデータ転送クロックの外部出力。
spiで生成されたSPIスレーブセレクト信号の外部出力。
spiで生成されたSPI転送データMOSIの外部出力。

これらの回路機能を、assign文を用いて記述しています。

```
            .
            .
            .

////////// Module Session

            .
            .
            .

      // Generat SCK
      clk8m_gen U10(CLK,clk8m);

      // MCP2515 register set
      regset_2515 U20(clk8m,rengedt,cs20,so20);

      // Colect range data
      range U30(CLK,rengedt,hold_renge_dec,MODE);

            .
            .
            .

endmodule
```

上記は、下位モジュールの呼び出しです。

clk8m_gen、regset_2515、range を呼び出しています。モジュール間のポート接続はポート・リストの順番による接続にしています。

range モジュールの詳細は、後述します。

６．２．２　SPI クロック生成モジュールの説明

モジュール名　　：　clk8m_gen
インスタンス名　：　U10

SPI クロック生成モジュール clk8m_gen の全文を、記述順序のとおりに説明します。

```
//////////////////////////////////////////
//            SPI CLOCK GEN              //
//     Module Name:     clk8m_gen        //
//     Instance Name:  U10               //
//////////////////////////////////////////
//            Revision record
//     ver 1.0 New RTL 2014.02.06

module clk8m_gen(GCLK, clk8m);
```

上記はモジュール宣言とポート・リストです。
clk8m_genは、トップモジュールRangFinderForCO_works_canから呼び出されます。

```
input GCLK;                         //Global Clock 33.333MHz
output clk8m;                       //SCK Source
```

上記は、ポート宣言です。
信号の詳細は、トップモジュールを参照して下さい。

```
// 33.33MHz/4=8.33MHz

parameter spirate0 = 3;    //4count clock = 0
parameter spirate1 = 1;    //2count clock = 1
```

上記は、パラメータ宣言です。

マスタークロックを４分周するためのパラメータです。
spirate0は、マスタークロック４カウントのパラメータです。
Spirate1は、マスタークロック２カウントのパラメータです。

```
reg   [3:0]sckcou;                  //Global Clock counter
reg   sckflg;                       //SCK Pulse
```

上記はモジュール内部信号のレジスタ宣言です。

[3:0] sckcou　：　マスタークロックをカウントする4bitカウンタです。回路記述上は2bitあれば動作しますが余裕を持たせて宣言しています。

Sckflg　　：　SPIクロックclk8mを生成する信号です。

次からが、回路の記述になります。

下記は、マスタークロックを４分周する回路の記述です。
always文を用いた順序回路で、マスタークロックの立ち上がりで動作します。

```
//GCLK Divide 4

always @(posedge GCLK)
begin
	if(sckcou == spirate0) begin		// 1/4GCLK
		sckcou <= 0;
		sckflg <= 0;			//SCK = 0
	end
```

上記は、マスタークロックを４カウントした時に動作する順序回路です。
記述中の組み合わせ回路で、マスタークロックカウンタsckcouをゼロにリセットし、次のマスタークロックの立ち上がりで１をカウントするようにします。SPIクロックの生成信号sckflgを論理０にします。SPIクロックLowの状態です。

```
	else begin
		if(sckcou == spirate1) begin	// 1/2GCLK
			sckcou <= sckcou + 1;
			sckflg <= 1;		//SCK = 1
		end
```

上記は、マスタークロックを２カウントした時に動作する順序回路です。
sckcouが１の状態でマスタークロックが立ち上がった時に動作しますので、マスタークロックを２カウントした時の動作になります。
記述中の組み合わせ回路で、マスタークロックカウンタsckcouをインクリメント（１加算）し、SPIクロックの生成信号sckflgを論理１にします。SPIクロックHighの状態です。

```
            else begin
                    sckcou <= sckcou + 1;
            end
```

上記は、マスタークロックを１または3カウントした時に動作する順序回路です。
記述中の組み合わせ回路で、マスタークロックカウンタsckcouをインクリメント（１加算）します。

```
        end
end
```

always文を用いた順序回路の終了です。

下記は、信号の出力です。
SPIクロックclk8mを、トップモジュールに出力します。

```
//SCK GEN assign

assign clk8m = sckflg;

endmodule
```

SPI クロック生成モジュール clk8m_gen の終了です。

６．２．３　SPIモジュールの説明

モジュール名　　：　spi
インスタンス名　：　U21

SPIモジュールspiの全文を、記述順序のとおりに説明します。
上位モジュールregset_2515の詳細は、６．３章で説明します。

```
//////////////////////////////////////////
//            SPI data transfer          //
//     Module Name:     spi               //
//     Instance Name:  U21                //
//////////////////////////////////////////
//                   Revision record
//     ver 1.0 New RTL 2014.02.15

module spi(clk8m,RST,length_f,cs1,so1,spicom,spiadd,spidat);
```

上記はモジュール宣言とポート・リストです。
spiは、上位モジュールregset_2515から呼び出されます。

```
input clk8m;                        //SCK GEN
input RST;                          //SPI module reset
input length_f;                     //SPI data length
input [7:0]spicom;                  //MCP2515 command
input [7:0]spiadd;                  //MCP2515 register address
input [7:0]spidat;                  //MCP2515 register data

output cs1;                         //SPI slave select
output so1;                         //SPI MOSI
```

ポート宣言された信号について説明します。

clk8m　　：　６．２．２章を参照して下さい。

RST　　：　上位モジュールregset_2515が生成する、SPIモジュールのリセット信号です。
SPIデータフレームのビットカウンタをゼロにリセットし、送出するデータの先頭バイトをセットします。

length_f　　：　上位モジュールregset_2515が生成する、SPIフレームのデータ長を示すステータス信号です。
0 : 3byte
1 : 1byte

[7:0]spicom　：　上位モジュールregset_2515が生成する、MCP2515の命令コードです。

[7:0]spiadd　：　上位モジュールregset_2515が生成する、MCP2515のレジスタアドレスです。

[7:0]spidat　：　上位モジュールregset_2515が生成する、MCP2515のレジスタに転送するデータです。

cs1　　：　spiで生成する、SPIのスレーブセレクト信号です。

so1　　：　spiで生成する、SPIの転送データMOSIです。

```
reg [4:0]gencou;                        //clk8m counter
reg cssts;                              //SPI sleve select status
reg sodt;                               //SPI MOSI status
reg [7:0]spicom_wr;                     //SPI data buffer
```

モジュール内部でレジスタ宣言された信号について説明します。

[4:0]gencou　：　clk8mの立ち下がりでカウントアップする、SPIデータフレームのビットカウンタで、5bitカウンタです。

cssts　：　SPIのスレーブセレクト信号を生成する信号です。

sodt　：　SPIの転送データMOSIを生成する信号です。

[7:0]spicom_wr：　SPIの転送データ1byteを格納するバッファです。
上位モジュールregset_2515が生成するSPIの転送データを、1byte単位で保持します。

ここからが、回路の記述になります。

SPIのフレームを構成する回路の記述です。
always文を用いた順序回路で、clk8mの立ち下がりで動作します。

clk8mの立ち下がりで動作する回路にしているのは、SPIの転送データMOSIがclk8mの立ち上がりで外部デバイスのMCP2515に転送されるためです。
clk8mの立ち下がりでデータを確定させ､その直後の立ち上がりでデータをFPGA外部に転送します。

```
always @(negedge clk8m)
begin
	if(RST) begin                          //SPI module reset RST=1 loop
		gencou <= 0;
		spicom_wr <= spicom;//SPI 1st byte set
	end
```

上記は、clk8m の立下りに同期し、SPIモジュールのリセット信号が論理１の時に動作する順序回路です。

RST信号の論理
0 : Reset解除
1 : Reset状態

リセット信号が論理１の間はこの回路のみ動作し、always文内の他の回路は動作しません。
記述中の組み合わせ回路で､SPIデータフレームのビットカウンタをゼロにリセットし、SPIの転送データバッファに最初の1byteを格納します。

else begin以下の回路は、リセット信号が解除（論理０）されると、動作を開始します。

```
	else begin
		if(length_f) begin          //SPI data length 1byte
```

SPIフレームのデータ長が1byteの時に動作する回路で、if(length_f) begin以下は、case文を使用した条件分岐になっています。

```
case(gencou)
        0:      //clk8m 1st count Data bit7(MSB)
        begin
                gencou <= gencou + 1;
                cssts <= 0;             //CS=0
                sodt <= spicom_wr[7];//MSB data set
                spicom_wr <= {spicom_wr[6:0],1'b0};
                                        //Next bit
        end
```

gencouが0の時に動作する回路で、最初の1bitをSPIに転送開始する直前の状態です。次のclk8mの立ち下がりでの動作を決めるため、ビットカウンタgencouをインクリメント（１加算）します。

スレーブセレクトcsstsを論理０にすることで、外部デバイスMCP2515にSPI転送開始を伝えます。

直後のclk8mの立ち上がりで転送するSPI転送データとして、SPI転送データバッファのMSB1bitを信号として確定させます。
ここで気をつけて欲しいのは、この記述を実行した時にSPI転送データが転送されるのではなく、実行直後のclk8mの立ち上がりで転送されることです。

次のclk8mの立ち下がりで2bit目を確定させるよう、SPI転送データバッファの内容を1bit左に（MSB方向に）シフトさせます。

```
spicom_wr <= {spicom_wr[6:0],1'b0};
```

このような記述をシフト・レジスタと呼び、パラレル/シリアルの変換でよく使用します。

```
1,2,3,4,5,6:          //bit6-1
begin
        gencou <= gencou + 1;
        sodt <= spicom_wr[7];
        spicom_wr <= {spicom_wr[6:0],1'b0};
end
```

SPI転送データ2bit目以降7bit目まで確定させる回路です。

gencouが1の場合を例にすると、1bit目の転送が完了し、2bit目の転送を待つ状態です。

次のclk8mの立ち下がりでの動作を決めるため、ビットカウンタgencouをインクリメント（１加算）します。

SPI転送データバッファの2bit目を信号として確定させ、バッファの内容を1bit左に（MSB方向に）シフトさせます。

```
7:
//bit0
begin
        gencou <= gencou + 1;
        sodt <= spicom_wr[7];
end
```

gencouが7の時に動作する回路で、7bit目の転送が完了し、8bit目の転送を待つ状態です。

次のclk8mの立ち下がりでの動作を決めるため、ビットカウンタgencouをインクリメント（１加算）し、SPI転送データバッファの8bit目を信号として確定させます。

```
                        8,9,10,11,12:       //After 8count CS=1
                        begin
                              cssts <= 1;
                        end
```

8bit目の転送が完了し、9bit目のクロックの立ち上がりを待つ状態です。

9bit目のクロックが立ち上がる直前に、スレーブセレクトcsstsを論理１にすることで、外部デバイスMCP2515にSPI転送終了を伝えます。

上位モジュールregset_2515からSPIモジュールのリセット（論理１）が確定されるまで、この状態を保持します。

```
                  endcase
```

case文の終了です。

```
            end
```

SPIフレームのデータ長が1byteの時に動作する回路記述の終了です。

else begin以下の記述は、SPIフレームのデータ長が3byteの時に動作する回路です。

```
            else begin    //SPI data length 3byte
```

以降assign文の終了までは、SPIフレームのデータ長が3byteになっただけで、1byteの時の回路と同様です。

```
                case(gencou)
                        0:      //clk8m 1st count Data bit23(MSB)
                        begin
                                cssts <= 0;          //CS=0
                                gencou <= gencou + 1;
                                sodt <= spicom_wr[7];//MSB data set
                                spicom_wr <= {spicom_wr[6:0],1'b0};
                                                      //Next bit
                        end


1,2,3,4,5,6,8,9,10,11,12,13,14,16,17,18,19,20,21,22:
                        begin
                                gencou <= gencou + 1;
                                sodt <= spicom_wr[7];
                                spicom_wr <= {spicom_wr[6:0],1'b0};
                        end

                        7:              //Before 8count
                        begin
                                gencou <= gencou + 1;
                                sodt <= spicom_wr[7];
                                spicom_wr <= spiadd;
                                        //Next byte register address
                        end

                        15:             //Before 16count
                        begin
                                gencou <= gencou + 1;
                                sodt <= spicom_wr[7];
                                spicom_wr <= spidat;
                                        //Next byte register data
                        end
```

```
                        23:             //Before 24count
                        begin
                                gencou <= gencou + 1;
                                sodt <= spicom_wr[7];
                        end

                        24,25,26,27,28,29,30://After 24count CS=1
                        begin
                                cssts <= 1;
                        end

                endcase
        end
    end
end
```

assign文により、スレーブセレクト信号と、SPI転送データを、上位モジュールregset_2515に出力します。

```
// CS,MOSI assign

assign cs1 = cssts;
assign so1 = sodt;

endmodule
```

SPI モジュール spi の終了です。

６．３　ＣＡＮコントローラ命令

モジュール名　　：　regset_2515
インスタンス名　：　U20

CAN 制御モジュール regset_2515 の全文を、記述順序のとおりに説明します。

```
//////////////////////////////////////////
//              MCP2515 Register set    //
//      Module Name:    regset_2515     //
//              Instance Name:  U20     //
//////////////////////////////////////////
//      Revision record
//      ver 1.0 New RTL 2014.02.21

module regset_2515(clk8m,rengedt,cs20,so20);
```

上記はモジュール宣言とポート・リストです。
regset_2515は、トップモジュールRangFinderForCO_works_canから呼び出されます。
また、下位モジュールspiを呼び出します。

```
input clk8m;                            //SCK GEN

//Range data
//[39:32] = Delimit 0x42=BCD 0x44=ASCII
//[31:24] = Range data 1000mm
//[23:16] = Range data  100mm
//[15:8] = Range data    10mm
//[7:0] = Range data      1mm

input [39:0] rengedt;
```

```
output cs20;                    //SPI Sleve select
output so20;                    //SPI MOSI
```

ポート宣言された信号について説明します。

clk8m　　：　６．２．２章を参照して下さい。

[39:0] rengedt　：　トップモジュールを参照して下さい。

cs20　　：　spiで生成する、SPIのスレーブセレクト信号の出力です。

so20　　：　spiで生成する、SPIの転送データMOSIの出力です。

```
wire cs1;                       //SPI Sleve select status
wire so1;                       //SPI MOSI status
```

モジュール内部でワイヤ宣言された信号について説明します。

cs1　：　spiで生成されたcs20を上位モジュールに出力する信号です。

so1　：　spiで生成されたso20を上位モジュールに出力する信号です。

```
reg RST = 1;                    //SPI Module Reset
reg length_f = 0;               //SPI data length status 0=3byte 1=1byte

reg [11:0]framecou = 0;                 //clk8m counter
reg [7:0]addcou = 0;                    //frame counter
reg [7:0]spicom;                        //MCP2515 command
reg [7:0]spiadd;                        //MCP2515 register address
reg [7:0]spidat;                        //MCP2515 register data
```

モジュール内部でレジスタ宣言された信号について説明します。

RST　　：　６．２．３章を参照して下さい。
length_f　　：　６．２．３章を参照して下さい。

[11:0]framecou　：clk8mの立ち下がりでカウントアップする、SPIメッセージのビットカウンタで、12bitカウンタです。

[7:0]addcou　：clk8mの立ち下がりでSPIの１メッセージの送信が完了するとカウントアップする、8bitカウンタです。

[7:0]spicom　：SPIで送信するMCP2515の命令コードをメッセージ単位で保持します。

[7:0]spiadd　：SPIで送信するMCP2515のレジスタアドレスをメッセージ単位で保持します。

[7:0]spidat　：SPIで送信するMCP2515のレジスタへの書き込みデータをメッセージ単位で保持します。

ここからは、パラメータ宣言です。

```
//*****************************//
//      MCP2515 Parameters    //
//*****************************//

//Write to register command
parameter spicom02 = 8'b00000010;
```

上記は、MCP2515のレジスタ書き込み命令コードです。

```
//CANCTRL 0x0F
//Out of config
parameter spiadd0F = 8'b00001111;       //Reg Address 0x28
parameter spidat0F = 8'b00000100;       //MCP2515 Normal mode
parameter spidat0FC = 8'b10000100;      //MCP2515 Config mode
```

MCP2515のCANCTRLレジスタの、アドレスとデータです。
アドレスは0x0F、MCP2515をノーマルモードまたはコンフィギュレーションモードに設定し、ワンショットモードを禁止、CLKOUTピンへの出力を有効に設定します。

```
//CNF3 0x28
//in oder to 500Kbps
parameter spiadd28 = 8'b00101000;       //Reg Address 0x28
parameter spidat28 = 8'b00000010;       //CAN bit time PS2=3TQ
```

上記は、MCP2515のCNF3レジスタの、アドレスとデータです。
アドレスは0x28、CANバスのタイムセグメントを設定します。

```
//CNF2 0x29
//in oder to 500Kbps
parameter spiadd29 = 8'b00101001;       //Reg Address 0x29
parameter spidat29 = 8'b10010010;       //CAN bit time PS1=PRP=3TQ
```

上記は、MCP2515のCNF2レジスタの、アドレスとデータです。
アドレスは0x29、マイコンボードとの通信速度500Kbpsになるよう、CANバスのタイムセグメントを設定します。

```
//TXB0SIDH 0x31
parameter spiadd31 = 8'b00110001;       //Reg Address 0x31
parameter spidat31 = 8'b00000000;       //CAN Base ID 10-3 = 0x00
```

上記は、MCP2515のTXB0SIDHレジスタの、アドレスとデータです。
アドレスは0x31、送信用CAN標準IDの上位8bitを設定します。

```
//TXB0SIDL 0x32
parameter spiadd32 = 8'b00110010;       //Reg Address 0x32
parameter spidat32 = 8'b00101000;       //CAN Base ID 2-0 = 001
                                        //Ext ID enable = 1,Ext ID17-16 = 00
```

上記は、MCP2515のTXB0SIDLレジスタの、アドレスとデータです。
アドレスは0x32、送信用CAN標準IDの下位3bitと送信用CAN拡張IDの上位2bit、そしてCANを拡張フォーマットに設定します。

```
//TXB0EID8 0x33
parameter spiadd33 = 8'b00110011;       //Reg Address 0x33
parameter spidat33 = 8'b00000000;       //CAN Ext ID 15-8 = 0x00
```

上記は、MCP2515のTXB0EID8レジスタの、アドレスとデータです。
アドレスは0x33、送信用CAN拡張IDのbit15からbit8を設定します。

```
//TXB0EID0 0x34
parameter spiadd34 = 8'b00110100;       //Reg Address 0x34
parameter spidat34 = 8'b00000000;       //CAN Ext ID 7-0 = 0x00
```

上記は、MCP2515のTXB0EID0レジスタの、アドレスとデータです。
アドレスは0x34、送信用CAN拡張IDのbit7からbit0を設定します。

```
//TXB0DLC 0x35
parameter spiadd35 = 8'b00110101;       //Reg Address 0x35
parameter spidat35 = 8'b00001000;       //CAN is Data frame. length 8byte
```

上記は、MCP2515のTXB0DLCレジスタの、アドレスとデータです。
アドレスは0x35、DLC（データ長）とRTR（リモートフレーム/データフレーム）を設定します。

以下はCANバス送信データバッファレジスタのアドレスです。

```
//CAN Data frame

//TXB0D0 0x36 Data buffer byte0
parameter spiadd36 = 8'b00110110;       //Reg Address 0x36

//TXB0D1 0x37 Data buffer byte1
parameter spiadd37 = 8'b00110111;       //Reg Address 0x37

//TXB0D2 0x38 Data buffer byte2
parameter spiadd38 = 8'b00111000;       //Reg Address 0x38
```

```
//TXB0D3 0x39 Data buffer byte3
parameter spiadd39 = 8'b00111001;       //Reg Address 0x39

//TXB0D4 0x3A Data buffer byte4
parameter spiadd3A = 8'b00111010;       //Reg Address 0x3A

//TXB0D5 0x3B Data buffer byte5
parameter spiadd3B = 8'b00111011;       //Reg Address 0x3B
parameter spidat3B = 8'b01000101;       //End of frame delimit 0x45

//TXB0D6 0x3C Data buffer byte6
parameter spiadd3C = 8'b00111100;       //Reg Address 0x3C
parameter spidat3C = 8'b00000000;       //Dummy byte

//TXB0D7 0x3D Data buffer byte7
parameter spiadd3D = 8'b00111101;       //Reg Address 0x3D
parameter spidat3D = 8'b00000000;       //Dummy byte
```

下記は、MCP2515のメッセージ送信要求命令コードです。

```
//Command in order to send TXB0D0
parameter spicom81 = 8'b10000001;
```

ここからは、回路の記述になります。
SPIのフレーム制御回路です。
clk8mの立下りに同期して動作します。

```
//*************************//
//      SPI frame control    //
//*************************//
```

```
always @(negedge clk8m)
begin
        if (length_f == 1) begin                        //Data length 1byte
```

SPIメッセージのデータ長が1byteの時に動作する回路です。

```
                if (framecou == 1) begin                //Before 1byte
                        RST <= 0;                       //SPI module enable
                        framecou <= framecou+1;
                end
```

SPIメッセージのビットカウンタが１の時、1bit目を送信する前に動作する回路です。
SPIモジュールリセット信号を論理ゼロにクリアしResetを解除します。また、SPIメッセージのビットカウンタを１カウントアップします。

```
                if (framecou == 10) begin               //After 1byte
                        RST <= 1;                       //SPI module reset
                        framecou <= framecou+1;
                end
```

SPIメッセージのビットカウンタが１０、8bit目の送信完了時に動作する回路です。SPIモジュールリセット信号を論理１にセットしReset状態にします。また、SPIメッセージのビットカウンタを１カウントアップします。

```
                //if (framecou == 11) begin
                if (framecou == 3000) begin     //Wait for next CAN data frame
                        framecou <= 0;
                        addcou <= 9;            //Return to frame No.9
                end
```

SPIメッセージのビットカウンタが3000の時、動作する回路です。SPIに対しCANの転送レートが低く伝送量も多いため、次のSPI送信をウエイトします。

SPIメッセージのビットカウンタを０にし、SPIメッセージカウンタを９にすることで、1byteメッセージの最初の1byteからの繰り返し動作を選択します。

```
        else begin
                framecou <= framecou+1;
        end
```

SPIメッセージのビットカウンタが１、１０、１１の時以外に動作する回路です。SPIメッセージのビットカウンタを１カウントアップします。

```
    end
```

1byteメッセージの送信終了です。

elsebegin以下は、SPIメッセージのデータ長が3byteの時に動作する回路です。

```
    else begin
    //Data length 3byte
        if (framecou == 1) begin                //Before 1st byte
            RST <= 0;                           //SPI module enable
            framecou <= framecou+1;
        end
```

SPIメッセージのビットカウンタが１の時、1bit目を送信する前に動作する回路です。
SPIモジュールリセット信号を論理ゼロにクリアしResetを解除します。また、SPIメッセージのビットカウンタを１カウントアップします。

```
        if (framecou == 26) begin               //After 3byte
            RST <= 1;                           //SPI module reset
            framecou <= framecou+1;
        end
```

SPIメッセージのビットカウンタが２６、２４bit目の送信完了時に動作する回路です。
SPIモジュールリセット信号を論理１にセットしReset状態にします。また、SPIメッセージのビットカウンタを１カウントアップします。

```
if (framecou == 27) begin
        framecou <= 0;                  //reset clk8m counter
        addcou <= addcou+1;
end
```

SPIメッセージのビットカウンタが２７の時に動作する回路です。
SPIメッセージのビットカウンタを０にします。
SPIメッセージカウンタを１カウントアップし、次のメッセージの送信に動作を移します。

```
else begin
        framecou <= framecou+1;
end
```

SPIメッセージのビットカウンタが１、２６、２７の時以外に動作する回路です。
SPIメッセージのビットカウンタを１カウントアップします。

```
    End
```

1byteメッセージの送信終了です。

```
end
```

SPIのフレーム制御回路の終了です。

ここからは、CANコントローラへのデータ転送回路です。
clk8mの立下りに同期して動作します。
case文を使用した条件分岐で、SPIの何個目のメッセージかを選択する回路になっています。SPIメッセージカウンタの値で動作を選択し値ゼロが１メッセージ目を示します。

```
//****************************//
//      MCP2515 register set  //
//****************************//

always @(negedge clk8m)
begin
        case(addcou)                                    //SPI Frame Number

// Config mode(default)
// Register set

                0:
                begin                           //CANCTRL MCP2515 Config mode
                        length_f <= 0;                  //SPI data length 3byte
                        spicom <= spicom02;
                        spiadd <= spiadd0F;
                        spidat <= spidat0FC;
                end
```

上記は１メッセージ目（3byte長）です。CANCTRLレジスタにコンフィギュレーションモード設定を書き込みます。SPIメッセージ長を3byteに指定します。

```
                1:
                begin                           //CNF3 in oder to 500Kbps
                        spiadd <= spiadd28;
                        spidat <= spidat28;
                end
```

上記は２メッセージ目（3byte長）です。CNF3レジスタに書き込みます。

```
                2:
                begin                           //CNF2 in oder to 500Kbps
                        spiadd <= spiadd29;
                        spidat <= spidat29;
                end
```

３メッセージ目（3byte長）です。CNF2レジスタに書き込みます。

```
3:
begin                           //CANCTRL MCP2515 Normal mode
        spiadd <= spiadd0F;
        spidat <= spidat0F;
end
```

上記は４メッセージ目（3byte長）です。CANCTRLレジスタにノーマルモード設定を書き込みます。

```
// Normal mode
// Register set
```

```
4:
begin                           //TXB0SIDH CAN Base ID 10-3 = 0x00
        spiadd <= spiadd31;
        spidat <= spidat31;
end
```

上記は５メッセージ目（3byte長）です。TXB0SIDHレジスタに書き込みます。

```
5:                      //TXB0SIDL CAN Base ID 2-0 = 001
begin                   //Ext ID enable = 1,Ext ID17-16 = 00
        spiadd <= spiadd32;
        spidat <= spidat32;
end
```

上記は６メッセージ目（3byte長）です。TXB0SIDLレジスタに書き込みます。

```
6:
begin                   //TXB0EID8 CAN Ext ID 15-8 = 0x00
        spiadd <= spiadd33;
        spidat <= spidat33;
end
```

上記は７メッセージ目（3byte長）です。TXB0EID8レジスタに書き込みます。

```
7:
begin                   //TXB0EID0 CAN Ext ID 7-0 = 0x00
        spiadd <= spiadd34;
        spidat <= spidat34;
end
```

上記は８メッセージ目（3byte長）です。TXB0EID0レジスタに書き込みます。

```
8:
begin                   //TXB0DLC CAN is Data frame. length 8byte
        spiadd <= spiadd35;
        spidat <= spidat35;
end
```

上記は９メッセージ目（3byte長）です。TXB0DLCレジスタに書き込みます。

3byte長メッセージの終了です。3byte長メッセージは電源投入時1回動作するだけです。その後は次の1byte長メッセージの繰り返しです。SPIのフレーム制御回路で、SPIメッセージカウンタを９にしたことを、思い出して下さい。

ここからは、CANバスに送出するデータの書き込みです。

図６．１に示す、マイコンボードとの通信仕様に沿った動作をします。

```
//      CAN Data frame

                9:
                begin                           //Start of frame
                        length_f <= 0;
                        spicom <= spicom02;
                        spiadd <= spiadd36;
                        spidat <= rengedt[39:32];
                end

                10:                             //CAN Data3
                begin
                        spiadd <= spiadd37;
                        spidat <= rengedt[31:24];
                end

                11:                             //CAN Data2
                begin
                        spiadd <= spiadd38;
                        spidat <= rengedt[23:16];
                end

                12:                             //CAN Data1
                begin
                        spiadd <= spiadd39;
                        spidat <= rengedt[15:8];
                end

                13:                             //CAN Data0
                begin
                        spiadd <= spiadd3A;
                        spidat <= rengedt[7:0];
                end
```

```
                14:                             //End of frame
                begin
                        spiadd <= spiadd3B;
                        spidat <= spidat3B;
                end

                15:                             //CAN Dummy byte
                begin
                        spiadd <= spiadd3C;
                        spidat <= spidat3C;
                end

                16:                             //CAN dummy byte
                begin
                        spiadd <= spiadd3D;
                        spidat <= spidat3D;
                end
```

下記は、MCP2515の送信バッファをCANバスに送信する命令です。
SPIメッセージ長を1byteに指定します。

```
                17:                             //Buffer send
                begin
                        length_f <= 1;          //SPI data length 1byte
                        spicom <= spicom81;
                end
        endcase
end
```

次は、下位モジュールspiの呼び出しです。

```
// SPI module

spi U21(clk8m,RST,length_f,cs1,so1,spicom,spiadd,spidat);
```

spiで生成されたSPIスレーブセレクトとSPI MOSIデータを上位モジュールに出力する記述です。

```
assign cs20 = cs1;                                    //SPI sleve select
assign so20 = so1;                                    //SPI MOSI

endmodule
```

CAN制御モジュールの終了です。

■ CAN通信仕様

通信プロトコル
・転送速度　　：500Kbps
・フォーマット　　：拡張フォーマット2.0アクティブ(IDは29ビット)
・メッセージフレーム　　：データフレーム(IDは29ビット)
・ID　　：ハード側 1／ソフト側 2
・コントロールフィールド　　：DLC(Data Length Code)＝8
・データフィールド　　：FPGAボードとマイコンボード間のシリアル通信(UART)のデータフレームを極力変更しない。
データフレームの8バイトを埋めるために最後にダミーデータ2バイト(0x00)を追加する。

BYTE	1	2	3	4	5	6	7	8
note	SOD StartOfData	1000位 単位mm	100位 単位mm	10位 単位mm	1位 単位mm	EOD EndOfData	dummy	dummy
	0x42(B)	・・・	0x00～0x09 × 4		・・・	0x45(E)	0x00	0x00
	BinaryData							
	0x44(D)	・・・	0x30～0x39 × 4		・・・	0x45(E)	0x00	0x00
	DecimalData							

例:42．8cmのデータ

Binary	0x42	0x00	0x04	0x02	0x08	0x45	0x00	0x00
Decimal	0x44	0x30	0x34	0x32	0x38	0x45	0x00	0x00

図６．１

６．３　ＣＡＮ通信

応用編でCANバスを使用して通信するデータは、基礎編で使用した超音波距離計の計測データです。このデータをマイコンボードとの仕様で通信します。

マイコンボードとの通信仕様により、CAN制御モジュールのSPIメッセージカウンタ９から１６の８ｂｙｔｅに、このデータを転送します。

超音波距離計と接続する超音波距離計インタフェースモジュールの説明をします。

モジュール名　　：　range
インスタンス名　：　U30

CAN制御モジュールrangeの全文を、記述順序のとおりに説明します。

```
//////////////////////////////////////////
//              RangeFinder Interface     //
//      Module Name:    range             //
//      Instance Name:  U30               //
//////////////////////////////////////////
//              Revision record
//      ver 1.0 New RTL 2014.02.21

module range(GCLK,rengedt,hold_renge_dec,MODE);

input GCLK;                             //Global clock
//RangeFinder colect data
//[15:12] = Range data 1000mm
//[11:8]  = Range data  100mm
//[7:4]   = Range data   10mm
//[3:0]   = Range data    1mm

input [15:0] hold_renge_dec;
```

GCLKはマスタークロックです。
hold_renge_decは、超音波距離計の16bit計測データです。

```
//BCD/ASCII select

input MODE;
```

MODEはスイッチ入力で、マイコンボードへの送信データをBCDまたはASCIIどちらかの切り換えに使用します。

```
//Range data
//[39:32] = Delimit 0x42=BCD 0x44=ASCII
//[31:24] = Range data 1000mm
//[23:16] = Range data  100mm
//[15:8] = Range data    10mm
//[7:0] = Range data      1mm

output [39:0] rengedt;

reg [39:0] rangedt_buf = 0;                                          //Range
data buffer
```

rengedt40bitはCAN制御モジュールに接続する、マイコンボード向けデータです。
rangedt_buf40bitはスイッチ切り換えにより選択された値で、rengedtに出力します。

ここから回路記述です。

マスタークロックの立ち上がりに同期してスイッチの状態を判定し、１であればASCII、０であればBCDデータを選択します。

```
//*******************************//
//      Colect range data       //
//*******************************//

always @(posedge GCLK)
begin
        if(MODE) begin                          //ASCII
                rangedt_buf <= {8'b01000100,    //Start of frame delimit 0x44
                                4'b0011,hold_renge_dec[15:12],
                                4'b0011,hold_renge_dec[11:8],
                                4'b0011,hold_renge_dec[7:4],
                                4'b0011,hold_renge_dec[3:0]};
        end

        else begin                              //BCD
                rangedt_buf <= {8'b01000010,    //Start of frame delimit 0x42
                               4'b0000,hold_renge_dec[15:12],
                               4'b0000,hold_renge_dec[11:8],
                               4'b0000,hold_renge_dec[7:4],
                               4'b0000,hold_renge_dec[3:0]};
        end
end

// Range data assign

assign rengedt[31:0] = rangedt_buf[31:0];

endmodule
```

7. Appendix

応用編で追加されたVerilogHDLのソースリスト

7．1　clk8m_gen.v（U10）

```
//////////////////////////////////////////
//              SPI CLOCK GEN           //
//      Module Name:    clk8m_gen       //
//      Instance Name:  U10             //
//////////////////////////////////////////
//      Revision record
//      ver 1.0 New RTL

module clk8m_gen(GCLK, clk8m);

input GCLK;                             //Global Clock 33.333MHz
output clk8m;                           //SCK Source

// 33.33MHz/4=8.33MHz

parameter spirate0 = 3;  //4count clock = 0
parameter spirate1 = 1;  //2count clock = 1

reg  [3:0]sckcou;                       //Global Clock counter
reg  sckflg;                            //SCK Pulse

//GCLK Divide 4

always @(posedge GCLK)
begin
        if(sckcou == spirate0) begin     // 1/4GCLK
                sckcou <= 0;
                sckflg <= 0;                                    //SCK = 0
        end

        else begin
                if(sckcou == spirate1) begin     // 1/2GCLK
                        sckcou <= sckcou + 1;
                        sckflg <= 1;                            //SCK = 1
                end

                else begin
                        sckcou <= sckcou + 1;
                end
        end
end
```

```
//SCK GEN assign

assign clk8m = sckflg;

endmodule
```

７．２　regset_2515.v (U20)

```
////////////////////////////////////////
//      MCP2515 Register set          //
//      Module Name:    regset_2515   //
//      Instance Name:  U20           //
////////////////////////////////////////
//      Revision record
//      ver 1.0 New RTL

module regset_2515(clk8m,rengedt,cs20,so20);

input clk8m;             //SCK GEN

//Range data
//[39:32] = Delimit 0x42=BCD 0x44=ASCII
//[31:24] = Range data 1000mm
//[23:16] = Range data  100mm
//[15:8] = Range data    10mm
//[7:0] = Range data      1mm

input [39:0] rengedt;

output cs20;                    //SPI Sleve select
output so20;                    //SPI MOSI

wire cs1;                               //SPI Sleve select status
wire so1;                               //SPI MOSI status

reg RST = 1;                    //SPI Module Reset
reg length_f = 0;               //SPI data length status 0=3byte 1=1byte

reg [11:0]framecou = 0;         //clk8m counter
reg [7:0]addcou = 0;                    //frame counter
reg [7:0]spicom;                        //MCP2515 command
reg [7:0]spiadd;                        //MCP2515 register address
reg [7:0]spidat;                        //MCP2515 register data

//*****************************//
//      MCP2515 Parameters    //
//*****************************//

//Write to register command
parameter spicom02 = 8'b00000010;

//CANCTRL 0x0F
//Out of config
parameter spiadd0F = 8'b00001111; //Reg Address 0x28
parameter spidat0F = 8'b00000100; //MCP2515 Normal mode
parameter spidat0FC = 8'b10000100;      //MCP2515 Config mode
```

```
//CNF3 0x28
//in oder to 500Kbps
parameter spiadd28 = 8'b00101000; //Reg Address 0x28
parameter spidat28 = 8'b00000010; //CAN bit time PS2=3TQ

//CNF2 0x29
//in oder to 500Kbps
parameter spiadd29 = 8'b00101001; //Reg Address 0x29
parameter spidat29 = 8'b10010010; //CAN bit time PS1=PRP=3TQ

//TXB0SIDH 0x31
parameter spiadd31 = 8'b00110001; //Reg Address 0x31
parameter spidat31 = 8'b00000000; //CAN Base ID 10-3 = 0x00

//TXB0SIDL 0x32
parameter spiadd32 = 8'b00110010; //Reg Address 0x32
parameter spidat32 = 8'b00101000; //CAN Base ID 2-0 = 001

                //Ext ID enable = 1,Ext ID17-16 = 00
//TXB0EID8 0x33
parameter spiadd33 = 8'b00110011; //Reg Address 0x33
parameter spidat33 = 8'b00000000; //CAN Ext ID 15-8 = 0x00

//TXB0EID0 0x34
parameter spiadd34 = 8'b00110100; //Reg Address 0x34
parameter spidat34 = 8'b00000000; //CAN Ext ID 7-0 = 0x00

//TXB0DLC 0x35
parameter spiadd35 = 8'b00110101; //Reg Address 0x35
parameter spidat35 = 8'b00001000; //CAN is Data frame. length 8byte

//CAN Data frame

//TXB0D0 0x36 Data buffer byte0
parameter spiadd36 = 8'b00110110; //Reg Address 0x36

//TXB0D1 0x37 Data buffer byte1
parameter spiadd37 = 8'b00110111; //Reg Address 0x37

//TXB0D2 0x38 Data buffer byte2
parameter spiadd38 = 8'b00111000; //Reg Address 0x38

//TXB0D3 0x39 Data buffer byte3
parameter spiadd39 = 8'b00111001; //Reg Address 0x39

//TXB0D4 0x3A Data buffer byte4
parameter spiadd3A = 8'b00111010; //Reg Address 0x3A

//TXB0D5 0x3B Data buffer byte5
parameter spiadd3B = 8'b00111011; //Reg Address 0x3B
parameter spidat3B = 8'b01000101; //End of frame delimit 0x45
```

```
//TXB0D6 0x3C Data buffer byte6
parameter spiadd3C = 8'b00111100; //Reg Address 0x3C
parameter spidat3C = 8'b00000000; //Dummy byte

//TXB0D7 0x3D Data buffer byte7
parameter spiadd3D = 8'b00111101; //Reg Address 0x3D
parameter spidat3D = 8'b00000000; //Dummy byte

//Command in order to send TXB0D0
parameter spicom81 = 8'b10000001;

//*****************************//
//      SPI frame control    //
//*****************************//

always @(negedge clk8m)
begin
        if (length_f == 1) begin                //Data length 1byte
                if (framecou == 1) begin        //Before 1byte
                        RST <= 0;
        //SPI module enable
                        framecou <= framecou+1;
                end

                if (framecou == 10) begin       //After 1byte
                        RST <= 1;
        //SPI module reset
                        framecou <= framecou+1;
                end

                //if (framecou == 11) begin
                if (framecou == 3000) begin     //Wait for next CAN data frame
                        framecou <= 0;
                        addcou <= 9;                    //Return to frame No.9
                end

                else begin
                        framecou <= framecou+1;
                end
        end

        else begin
        //Data length 3byte
                if (framecou == 1) begin        //Before 1st byte
                        RST <= 0;
        //SPI module enable
                        framecou <= framecou+1;
                end

                if (framecou == 26) begin       //After 3byte
                        RST <= 1;               //SPI module reset
                        framecou <= framecou+1;
                end
```

```
                if (framecou == 27) begin
                        framecou <= 0;          //reset clk8m counter
                        addcou <= addcou+1;
                end

                else begin
                        framecou <= framecou+1;
                end
        end
end

//*****************************//
//      MCP2515 register set //
//*****************************//

always @(negedge clk8m)
begin
        case(addcou)                                                    //SPI Frame Number

// Config mode(default)
// Register set

                0:
                begin                           //CANCTRL MCP2515 Config mode
                        length_f <= 0;          //SPI data length 3byte
                        spicom <= spicom02;
                        spiadd <= spiadd0F;
                        spidat <= spidat0FC;
                end

                1:
                begin                           //CNF3 in oder to 500Kbps
                        spiadd <= spiadd28;
                        spidat <= spidat28;
                end

                2:
                begin                           //CNF2 in oder to 500Kbps
                        spiadd <= spiadd29;
                        spidat <= spidat29;
                end

                3:
                begin                           //CANCTRL MCP2515 Normal mode
                        spiadd <= spiadd0F;
                        spidat <= spidat0F;
                end

// Normal mode
// Register set
```

```
                4:
                begin                   //TXB0SIDH CAN Base ID 10-3 = 0x00
                        spiadd <= spiadd31;
                        spidat <= spidat31;
                end

                5:                      //TXB0SIDL CAN Base ID 2-0 = 001
                begin                   //Ext ID enable = 1,Ext ID17-16 = 00
                        spiadd <= spiadd32;
                        spidat <= spidat32;
                end

                6:
                begin                   //TXB0EID8 CAN Ext ID 15-8 = 0x00
                        spiadd <= spiadd33;
                        spidat <= spidat33;
                end

                7:
                begin                   //TXB0EID0 CAN Ext ID 7-0 = 0x00
                        spiadd <= spiadd34;
                        spidat <= spidat34;
                end

                8:
                begin                   //TXB0DLC CAN is Data frame. length 8byte
                        spiadd <= spiadd35;
                        spidat <= spidat35;
                end

//      CAN Data frame

                9:
                begin                           //Start of frame
                        length_f <= 0;
                        spicom <= spicom02;
                        spiadd <= spiadd36;
                        spidat <= rengedt[39:32];
                end

                10:                             //CAN Data3
                begin
                        spiadd <= spiadd37;
                        spidat <= rengedt[31:24];
                end

                11:                             //CAN Data2
                begin
                        spiadd <= spiadd38;
                        spidat <= rengedt[23:16];
                end
```

```
                12:                             //CAN Data1
                begin
                        spiadd <= spiadd39;
                        spidat <= rengedt[15:8];
                end

                13:                             //CAN Data0
                begin
                        spiadd <= spiadd3A;
                        spidat <= rengedt[7:0];
                end

                14:                             //End of frame
                begin
                        spiadd <= spiadd3B;
                        spidat <= spidat3B;
                end

                15:                             //CAN Dummy byte
                begin
                        spiadd <= spiadd3C;
                        spidat <= spidat3C;
                end

                16:                             //CAN dummy byte
                begin
                        spiadd <= spiadd3D;
                        spidat <= spidat3D;
                end

                17:                             //Buffer send
                begin
                        length_f <= 1;          //SPI data length 1byte
                        spicom <= spicom81;
                end
        endcase
end

// SPI module

spi U21(clk8m,RST,length_f,cs1,so1,spicom,spiadd,spidat);

assign cs20 = cs1;                                      //SPI sleve select
assign so20 = so1;                                      //SPI MOSI

endmodule
```

７．３　spi.v (U21)

```
////////////////////////////////////////
//              SPI data transfer        //
//      Module Name:    spi              //
//      Instance Name:  U21              //
////////////////////////////////////////
//      Revision record
//      ver 1.0 New RTL

module spi(clk8m,RST,length_f,cs1,so1,spicom,spiadd,spidat);

input clk8m;                                    //SCK GEN
input RST;                                      //SPI module reset
input length_f;                         //SPI data length
input [7:0]spicom;                      //MCP2515 command
input [7:0]spiadd;                      //MCP2515 register address
input [7:0]spidat;                      //MCP2515 register data

output cs1;                                     //SPI sleve select
output so1;                                     //SPI MOSI

reg [4:0]gencou;                        //clk8m counter
reg cssts;                                      //SPI sleve select status
reg sodt;                                       //SPI MOSI status
reg [7:0]spicom_wr;                     //SPI data buffer

always @(negedge clk8m)
begin
        if(RST) begin                   //SPI module reset RST=1 loop
                gencou <= 0;
                spicom_wr <= spicom;    //SPI 1st byte set
        end



        else begin
                if(length_f) begin                              //SPI data length
1byte
                        case(gencou)
                                0:              //clk8m 1st count Data bit7(MSB)
                                begin
                                        gencou <= gencou + 1;
                                        cssts <= 0;
                                                //CS=0
                                        sodt <= spicom_wr[7];   //MSB data set
                                        spicom_wr <= {spicom_wr[6:0], 1'b0};
                                                                //Next bit
                                end
```

```
                1,2,3,4,5,6:                    //bit6-1
                begin
                        gencou <= gencou + 1;
                        sodt <= spicom_wr[7];
                        spicom_wr <= {spicom_wr[6:0], 1'b0};
                end

                7:                              //bit0
                begin
                        gencou <= gencou + 1;
                        sodt <= spicom_wr[7];
                end

                8,9,10,11,12:           //After 8count CS=1
                begin
                        cssts <= 1;
                end
        endcase
end


else begin                      //SPI data length 3byte
        case(gencou)
                0:              //clk8m 1st count Data bit23(MSB)
                begin
                        cssts <= 0;
                                //CS=0
                        gencou <= gencou + 1;
                        sodt <= spicom_wr[7];   //MSB data set
                        spicom_wr <= {spicom_wr[6:0], 1'b0};
                                                //Next bit
                end

        1,2,3,4,5,6,8,9,10,11,12,13,14,16,17,18,19,20,21,22:
                begin
                        gencou <= gencou + 1;
                        sodt <= spicom_wr[7];
                        spicom_wr <= {spicom_wr[6:0], 1'b0};
                end

                7:
                        //Before 8count
                begin
                        gencou <= gencou + 1;
                        sodt <= spicom_wr[7];
                        spicom_wr <= spiadd;
                                //Next byte register address
                end
```

```
                                15:                     //Before 16count
                                begin
                                        gencou <= gencou + 1;
                                        sodt <= spicom_wr[7];
                                        spicom_wr <= spidat;
                                                        //Next byte register data
                                end

                                23:                     //Before 24count
                                begin
                                        gencou <= gencou + 1;
                                        sodt <= spicom_wr[7];
                                end

                                24,25,26,27,28,29,30:   //After 24count CS=1
                                begin
                                        cssts <= 1;
                                end

                        endcase
                end
        end
end

// CS,MOSI assign

assign cs1 = cssts;
assign so1 = sodt;

endmodule
```

７．４　range.v (U30)

```
////////////////////////////////////////
//              RangeFinder Interface    //
//      Module Name:    range            //
//      Instance Name:  U30              //
////////////////////////////////////////
//      Revision record
//      ver 1.0 New RTL

module range(GCLK,rengedt,hold_renge_dec,MODE);

input GCLK;                                                     //Global clock
//RangeFinder colect data
//[15:12] = Range data 1000mm
//[11:8]  = Range data  100mm
//[7:4]   = Range data   10mm
//[3:0]   = Range data    1mm

input [15:0] hold_renge_dec;

//BCD/ASCII select

input MODE;

//Range data
//[39:32] = Delimit 0x42=BCD 0x44=ASCII
//[31:24] = Range data 1000mm
//[23:16] = Range data  100mm
//[15:8] = Range data    10mm
//[7:0] = Range data      1mm

output [39:0] rengedt;

reg [39:0] rangedt_buf = 0;                             //Range data buffer

//*********************************//
//      Colect range data        //
//*********************************//

always @(posedge GCLK)
begin
        if(MODE) begin                                  //ASCII
                rangedt_buf <= {8'b01000100,      //Start of frame delimit 0x44
                                4'b0011,hold_renge_dec[15:12],
                                4'b0011,hold_renge_dec[11:8],
                                4'b0011,hold_renge_dec[7:4],
                                4'b0011,hold_renge_dec[3:0]};
        end
```

```
        else begin                                          //BCD
                rangedt_buf <= {8'b01000010,      //Start of frame delimit 0x42
                                4'b0000,hold_renge_dec[15:12],
                                4'b0000,hold_renge_dec[11:8],
                                4'b0000,hold_renge_dec[7:4],
                                4'b0000,hold_renge_dec[3:0]};
        end
end

// Range data assign

assign rengedt[31:0] = rangedt_buf[31:0];

endmodule
```

７．５　RangFinderForCO_works_can.ucf（ピンアサイン）

```
#////////////////////////////////////////////////////////////////////////////

#// Global Clock 33.333MHz
NET "CLK"                          LOC = "P38"; #2012-P63

#// Input Transmit Data Select
NET "TX_SELECT"  LOC = "P88"; # SW1  2012-P24 SW8

#// 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
#// 2013 delete
//NET "TP_TRG"           LOC = "P58"; #2012-P57 MCP2515 CS
//NET "PLS_1mm"          LOC = "P61"; #2012-P60 MCP2515 SI
#//

#////////// 2013 Version Deleted
#NET "LED<8>"            LOC = "NONE"; #2012-P36 LED8
#NET "LED<7>"            LOC = "NONE"; #2012-P40 LED7
#NET "LED<6>"            LOC = "NONE"; #2012-P41 LED6
#NET "LED<5>"            LOC = "NONE"; #2012-P47 LED5

NET "LED<4>"             LOC = "P10"; #2012-P48 LED4
NET "LED<3>"             LOC = "P9";  #2012-P49 LED3
NET "LED<2>"             LOC = "P5";  #2012-P53 LED2
NET "LED<1>"             LOC = "P4";  #2012-P54 LED1

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

#// 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 = "P69";     #2012-P71 RS232C Rx DATA
NET "TxD"                       LOC = "P70";     #2012-P70 RS232C Tx DATA

#//     SPI in order to CAN control
#//     2013 edition

NET "SCK"               LOC = "P62";     #//MPC2515 SCK
NET "SO"                LOC = "P61";     #//MPC2515 SI
NET "CS"                LOC = "P58";     #//MPC2515 CS
```

平成 25 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講座　ＦＰＧＡとマイコンの連携システム
（ハードウェア応用編）

平成 26 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
TEL：022-224-6501