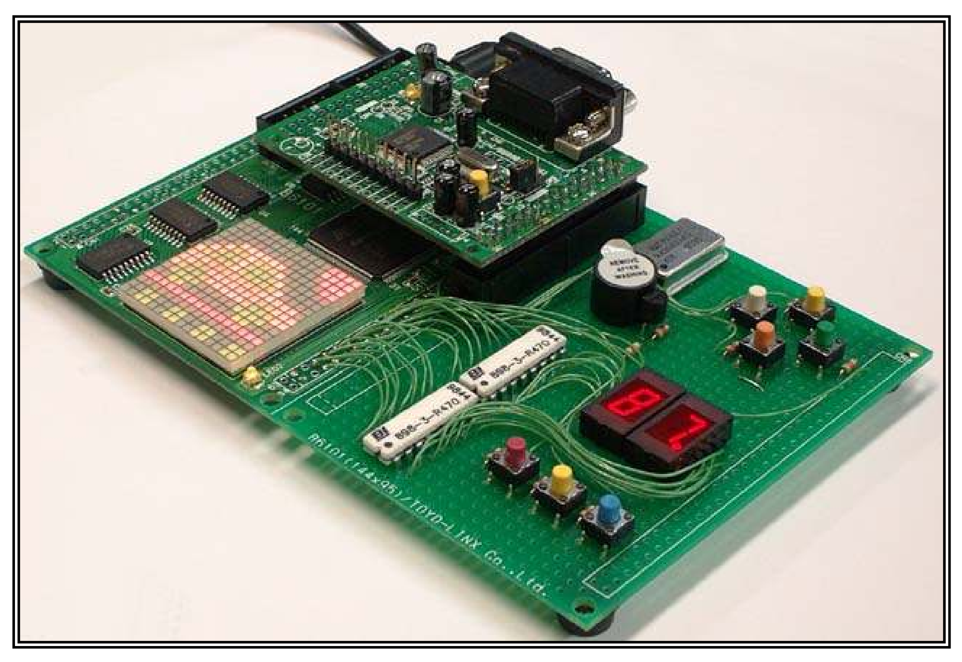
FPGAボード"B6101"を使った

FPGA事始め

Version 0.08



目次

はじめに ・・ P. 1
第 1 章　FPGA ボード"B6101"を眺めてみよう ・・・ P. 2
第 2 章　実習回路の組立て ・・・ P. 4
第 3 章　開発環境を整えよう ・・・ P. 8
第 4 章　Cyclone に回路を書き込んでみよう ・・・ P. 20
第 5 章　基本的な回路を入力して Cyclone に書き込んでみよう ・・・・・・・・・・・・・・・・・・・・・・・・・・・・ P. 28
第 6 章　ドットマトリックス LED に表示する ・・ P. 87
第 7 章　マイコンと組み合わせて使ってみよう ・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・ P. 100
第 8 章 ・・ P.
第 9 章 ・・ P.
第 10 章 ・・・ P.
第 11 章 ・・・ P.
第 12 章 ・・・ P.
第 13 章 ・・・ P.
付録（回路図, 参考資料） ・・ P. 115

(株)東洋リンクス

はじめに

その昔，論理回路の設計といえば，タイミングチャートや真理値表を作って設計し，AND や OR，NAND や NOR，NOT など，MIL 記号で回路図を書き，74 シリーズのデジタル IC を使ってユニバーサル基板に試作回路を組み，オシロスコープやロジアナを使ってデバッグする，という感じでした。最終的にはプリント基板をおこすわけですが、プリント基板にした後で変更が入ったりすると，パターンをカットしたり，ジャンパを飛ばしたり，IC を無理やり追加したり，という力業で対応したものです。

こうなってくると，「回路の変更が自由にできるデバイスが欲しい！」，という要望が出てきます。というわけで普及していったのがマイコンです。かなり複雑な動作でも，プログラムという言語の羅列で実現することができ，変更があってもプログラムをちょっと（？）いじれば対応できるようになりました。

しかし，マイコンにはどうしても避けられない弱点があります。まずはスピードです。マイコンの性質上，動作するには命令コードを読み込み，どんな命令か解析し，その命令を実行するというステップを踏まなければなりません。なので，どうしても一つの動作に時間がかかってしまいます。

別の弱点は，並行処理が苦手，ということです。マイコンはマルチタスクや割込み処理を使うことで幾つもの処理を並行して行なうことができるといわれています。しかし，マイコンが一度に一つの命令しか処理できない以上，完全な意味で並行処理をしているのではなく，厳密にはマルチタスクや割込みを使って並行処理をしているように見せかけているに過ぎません。

となると，「マイコンのように中身の動作を自由に書き変えることができ，なおかつ，純粋ハードの論理回路と同じような性質を持つデバイスが欲しい！」，という要望が出てきます。そこで登場したのが，プログラマブル・デバイス（PLD：Programmable Logic Device）です。FPGA（Field Programmable Gate Array）はその一種で，比較的大きな規模の回路を組み込むことができます。

FPGA はパソコン上で動作する開発ソフトに回路図を入力しダウンロードすると，その回路図どおりの動作を行なうデバイスに変身します。実際にやってみると感動ものですよ。さらに回路図ではなく HDL（Hardware Description Language）というハードウェア記述言語で回路設計を行なうと，回路図だと複雑かつ大きな回路でも比較的簡単に設計することができるようになります。

さて，このマニュアルは，これまで FPGA に触れたことがない人でも，FPGA を使えるようになることが目標です。コンセプトは「とにかく動かしてみよう」です。スマートな設計方法はあるのでしょうが，まずは使って動かしてみることが先決だと考えます。それから高度な使い方をマスターしても遅くはないでしょう。

それで，FPGA 自体の仕組みについてはいろいろな解説書で説明されていることもあり省略しました。ただ，一つだけ覚えておいていただきたいのは，FPGA の論理を構成する基本要素が SRAM でできているということです。そのため，電源オンのたびに回路データ（コンフィグレーションデータ）を FPGA に読み込ませる必要があり，そのためのデバイスをコンフィグレーション ROM と呼びます。FPGA とコンフィグレーション ROM はセットで使うものだと思っていただいてもいいでしょう。

FPGA はプログラマブル・デバイスですから，中身を自由に設計することができます。言ってみれば，LSI を自分で作れるということです。カスタム LSI を作るとなると，半導体工場で生産するため，高い開発費と一定の製造期間がかかります。商品として成り立たせるためには数千個単位で作らないと安くなりません。それが，自分のパソコンで，たとえ 1 個だけだったとしても，自由に作れるのですから面白いことになりましたよね。このテキストが HDL による回路設計の面白さを伝え，皆さんの技術向上のお役に立てば幸いです。

このマニュアルで使用している開発ツールは，「**QuartusⅡ Web Edition（V7.1SP1）**」です。ALTERA は対応デバイスを追加したり，ソフトを改良したり，不具合を修正したりするために，予告なしに QuartusⅡを どんどんバージョンアップしています。それで，いつも最新の QuartusⅡを入手されることをおすすめします。

なお，バージョンアップに伴い，「FPGA 事始め」のマニュアルの説明や画面などが最新版の QuartusⅡと異なることがあるかもしれませんが，これについて弊社は免責とさせていただきます。もちろん，折をみて更新し，弊社 Web ページからダウンロードできるようにしたいと考えています。

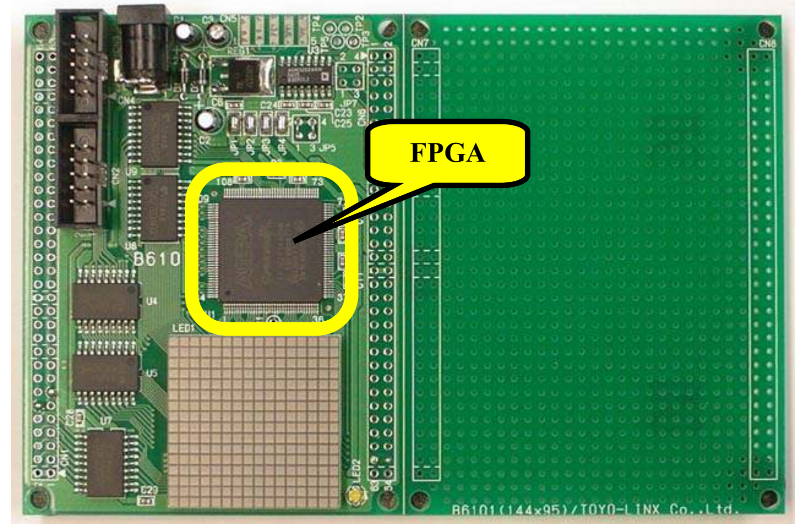
第1章	FPGA ボード"B6101"を眺めてみよう
	1. FPGA ボードの構成

キットの FPGA ボードを購入された方は，まず「FPGA トレーニングキット組立て手順書」を見て基板を組み立ててください。

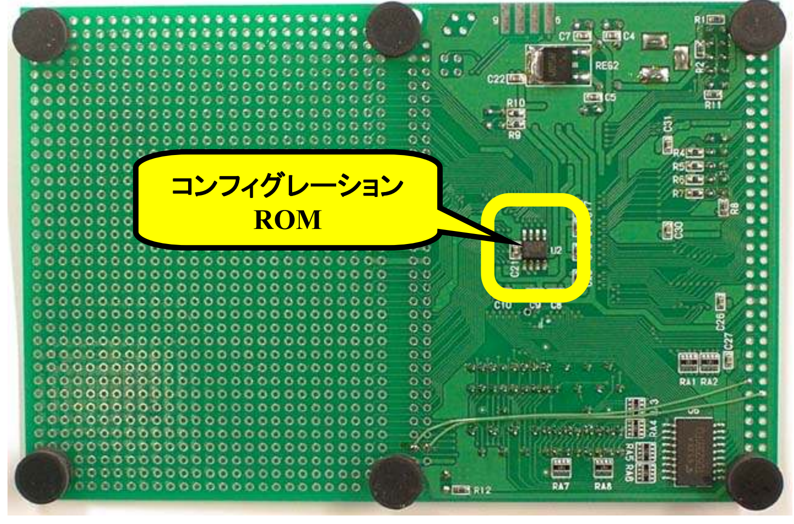
完成しましたか？では，今作った FPGA ボードを概観することから始めましょう。

1. FPGA ボードの構成

まずは組み立てた基板を眺めてみましょう。基板の中央付近に大きな LSI（Cyclone）が 1 個のっていますね。これが FPGA で，この中に論理回路をダウンロードすることで，設計したとおりの動作をするようになります。

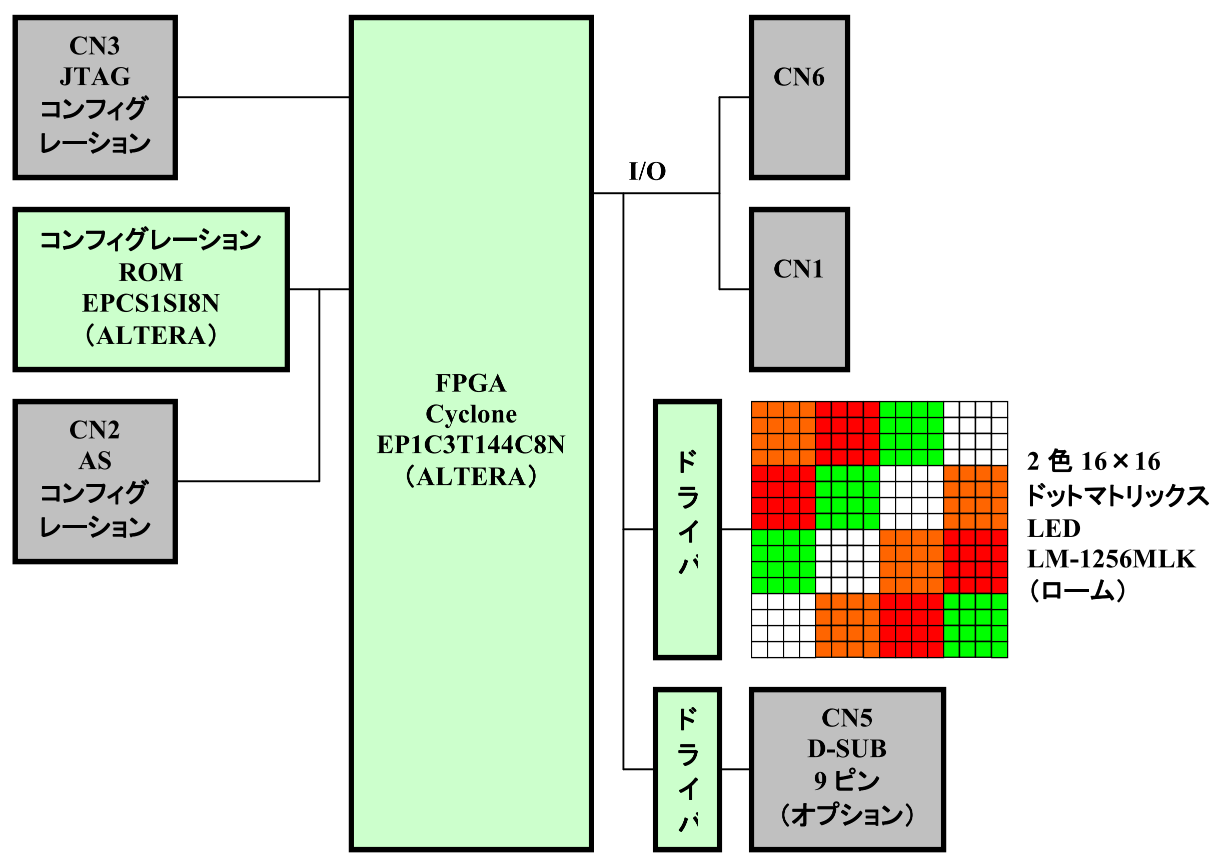


FPGA を動かすためにもう一つ重要なのは，コンフィグレーション ROM です。FPGA にダウンロードする回路データ（コンフィグレーションデータ）はコンフィグレーション ROM に書きこまれており，FPGA は電源オンでコンフィグレーション ROM からコンフィグレーションデータを読みこみます。



なお，FPGA 自体は電源オフでどんな回路だったか，きれいさっぱり忘れてしまいます。なので，コンフィグレーション ROM がないと FPGA は動作する回路として成り立ちません。また，電源オンでコンフィグレーションデータを全て読み込むまでには多少の時間がかかるため，マイコンなどと組み合わせて使うときその時間を考慮して動き始めないと「デバッグ中は大丈夫だったのに，製品に組み込んだら誤動作する･･･」と悩むことになります。

では，FPGA ボード“B6101”のブロック図をみてみましょう。



このブロック図からも分かるように，ハードの構成としては非常に単純化されています。問題は，FPGA の中にどんな回路を組み込むかという事です。というわけで，とりあえず大き目の FPGA を使って回路図を書き上げ，基板設計&基板作成を外注に頼み，基板ができあがるまでの間に FPGA の中身を設計する，という方法も，FPGA を使えば可能です。

さて，CN3 の「JTAG コンフィグレーション」とは，コンフィグレーションデータを直接 FPGA に書き込む方法です。そのため，電源をオフすると回路の内容は消えてしまいます。

CN2 の「AS コンフィグレーション」はコンフィグレーション ROM にコンフィグレーションデータを書き込みます。電源をオフしても回路の内容は残っています。再び電源をオンすれば，コンフィグレーション ROM からコンフィグレーションデータを読み込みます。

コンフィグレーション ROM はフラッシュメモリのため，書き込み回数には限界があります。それで，デバッグ中は JTAG コンフィグレーションで FPGA に直接書き込み，ある程度デバッグが完了したら AS コンフィグレーションでコンフィグレーション ROM に書き込むのがよいと思います。

第2章	実習回路を組み立てる
	1. 部品の確認 2. 実習回路の組み立て

1. 部品の確認

キットの内容を確かめて下さい。ちゃんとそろっていますか。

		部品名	メーカ	数	備考
1	FPGA ボード（キット or 完成品）発振モジュールが同封されています	B6101	東洋リンクス	1	
2	専用マイコンボード（同時購入されている場合）	TK-3687mini	東洋リンクス	1	
3	7 セグメント LED	TLR312	東芝	2	
4	プッシュスイッチ	SKHHAK/AM/DC	ALPS	7	
5	サウンダ	QMX-05	STAR	1	
6	トランジスタ	2SC1815		1	
7	抵抗	4．7kΩ		9	
8	抵抗アレイ	898-3-R470 （470Ω×8）	BI	2	
9	コネクタ	HIF3FC-30PA-2.54DSA	HRC	1	
10	ラッピングワイヤ	1m		1	

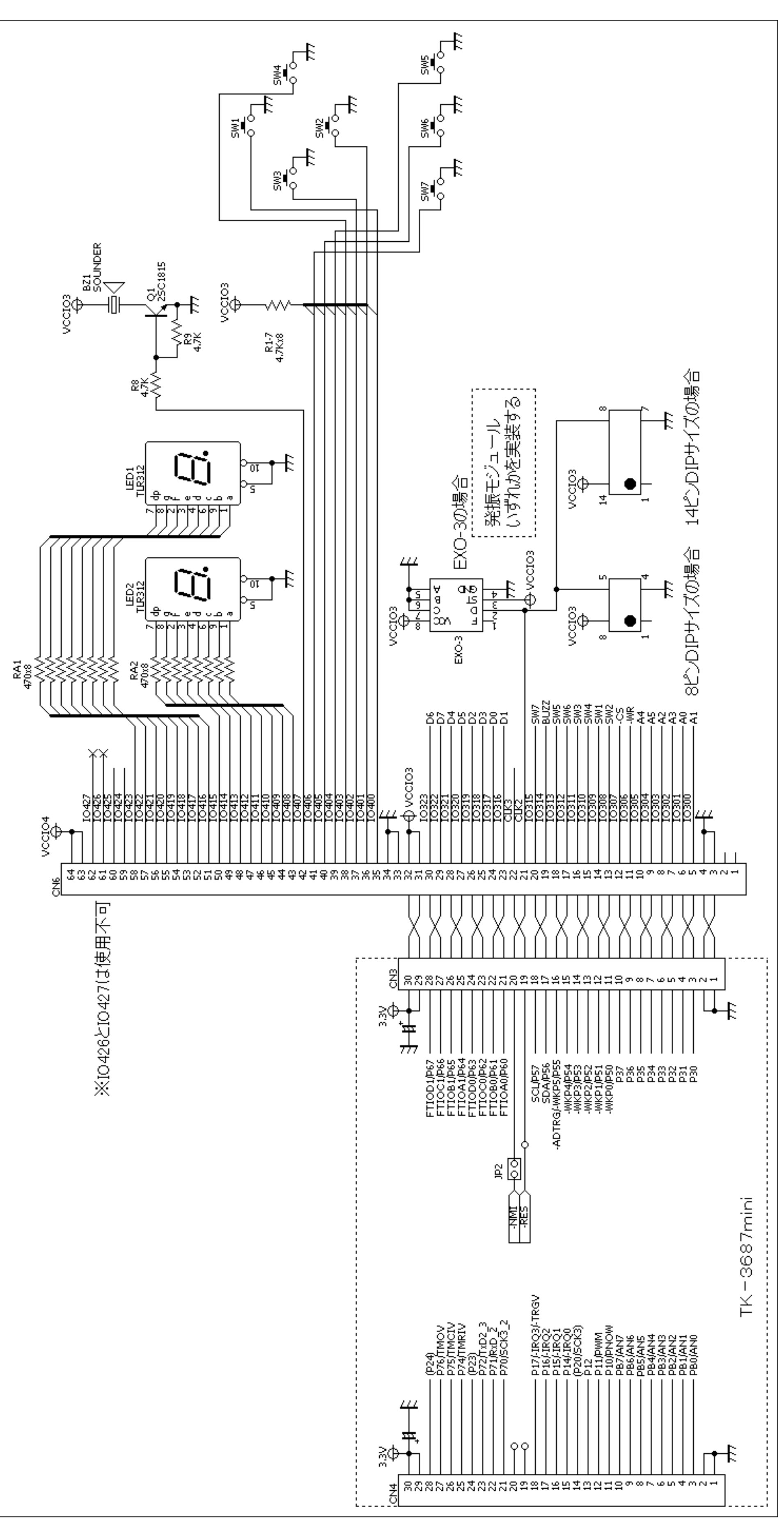
■ 相当品を使用することがあります。

■ FPGA ボード付属の発振モジュールも使います。（EXO-3，8 ピン DIP，14 ピン DIP サイズ）

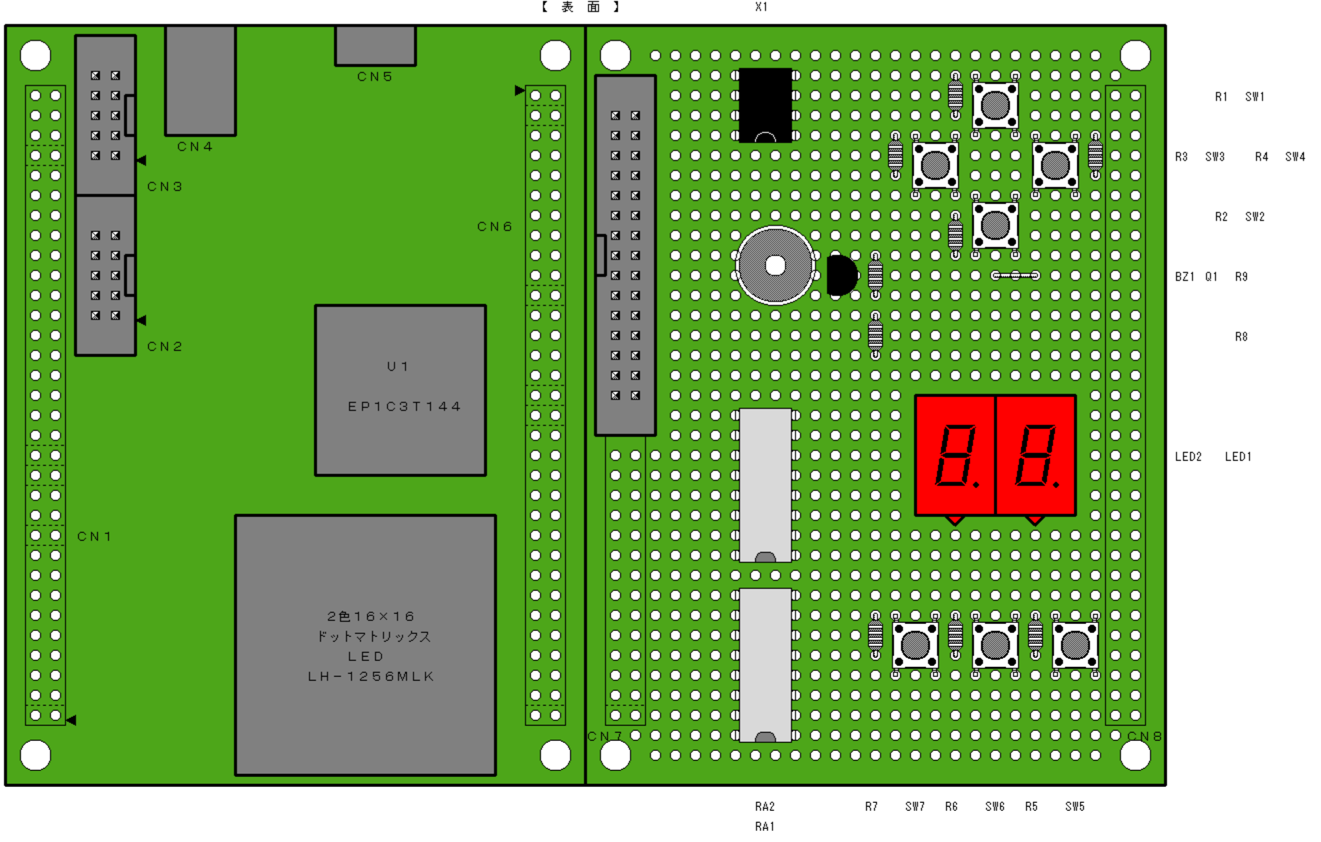
■ 不足部品があるときは，東洋リンクスまでお問い合わせください。（巻末の連絡先参照）

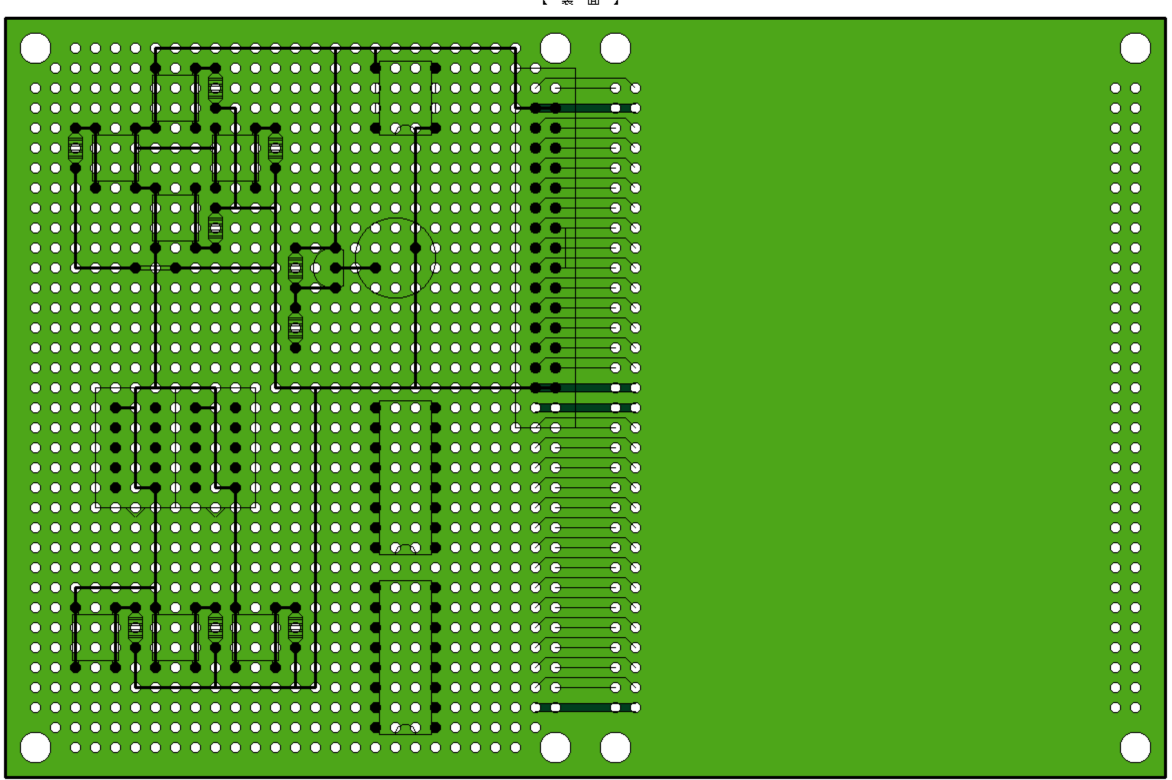
2. 実習回路の組立て

部品がそろっていることが確認できたら，右の回路図の実習回路をユニバーサルエリアに組み立てましょう。次ページの実装図や，その次のページの写真を参考にしてください。もちろん，回路図どおりであれば細かい配線の違いを気にすることはありません。



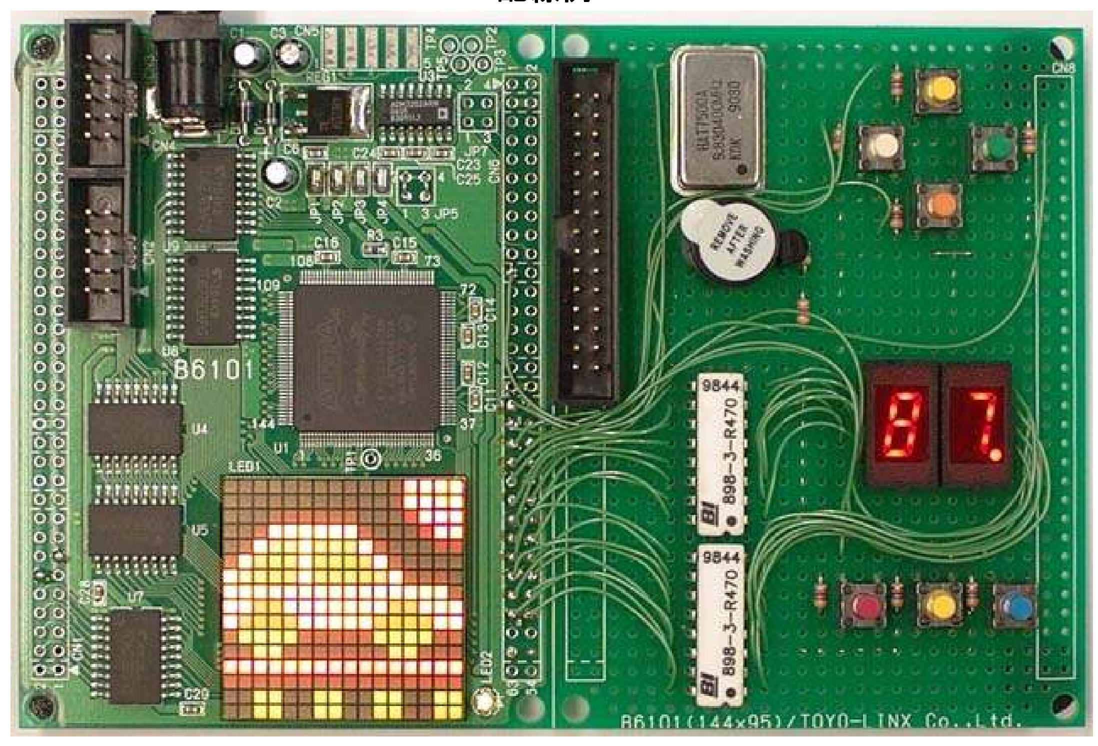
実装図

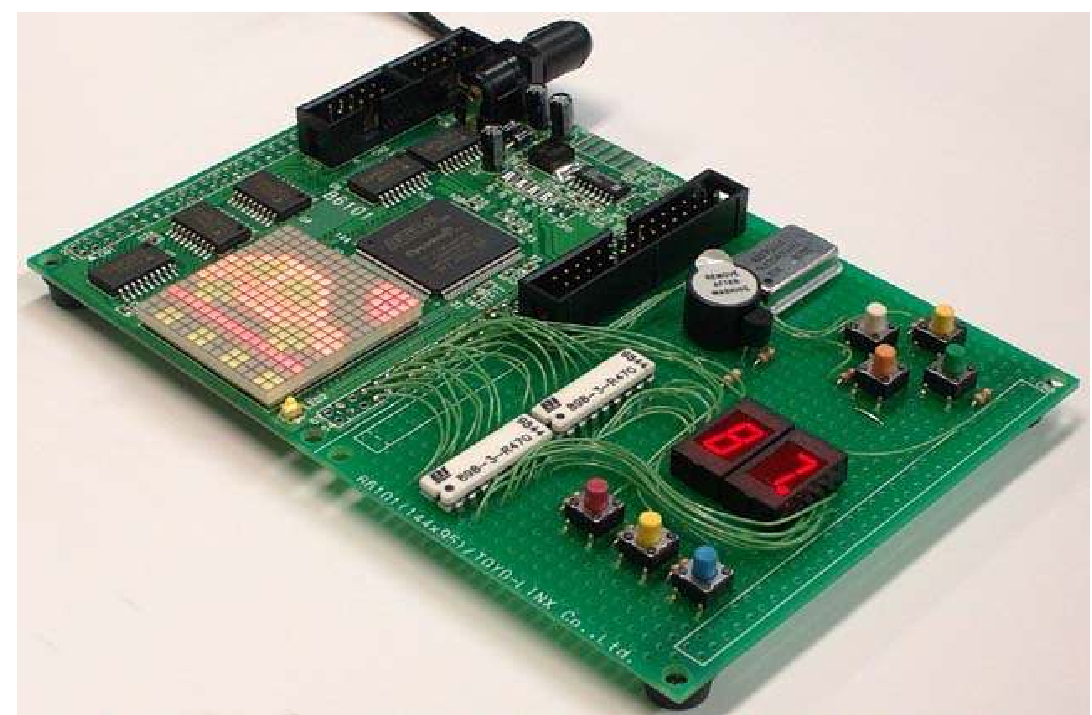




※この図の発振モジュールは 8 ピン DIP サイズになっています。

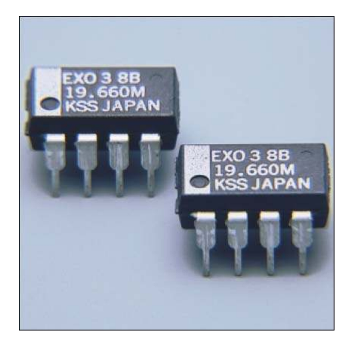
配線例





※この写真の発振モジュールは 14 ピン DIP サイズになっています。

※右の写真は，発振モジュール，EXO-3 の外形です。電源端子は 8 ピン DIP サイズの発振モジュールと同じです。



第3章 開発環境を整えよう

1. Quartus Ⅱを手に入れよう
2. Quartus Ⅱのインストール
3. ダウンロードケーブル
4. データ保存フォルダの作成

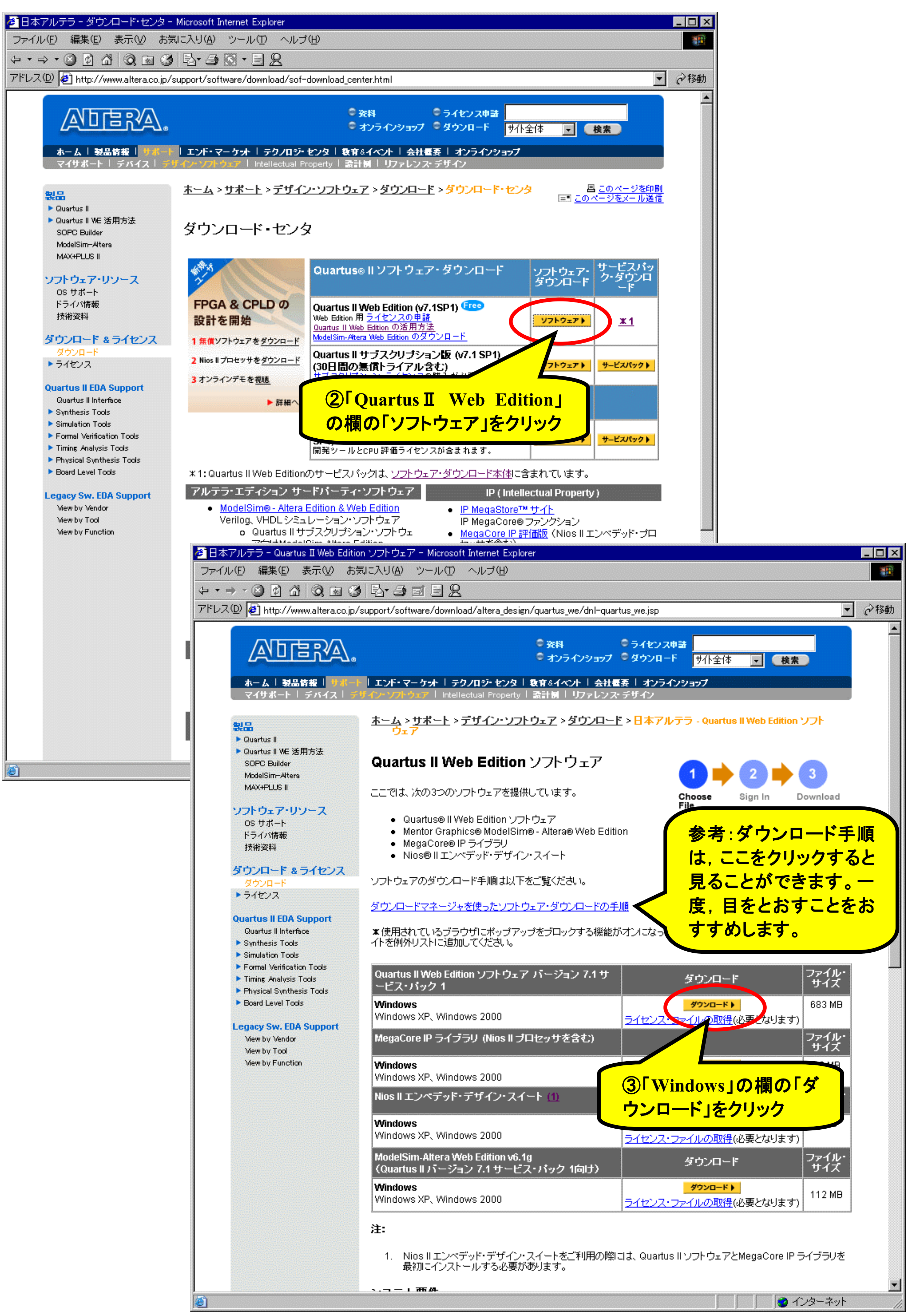
早速 FPGA を使いたいのですが，まずは開発環境を整えることにしましょう。FPGA の開発にはパソコンを使いますが，その条件は次のとおりです。

① OS は Windows2000，Xp（Vista は公式には対応していない）
② USB が使えること（USB-Blaster を使う場合）
③ LAN が使えること（≒インターネットに接続できること）

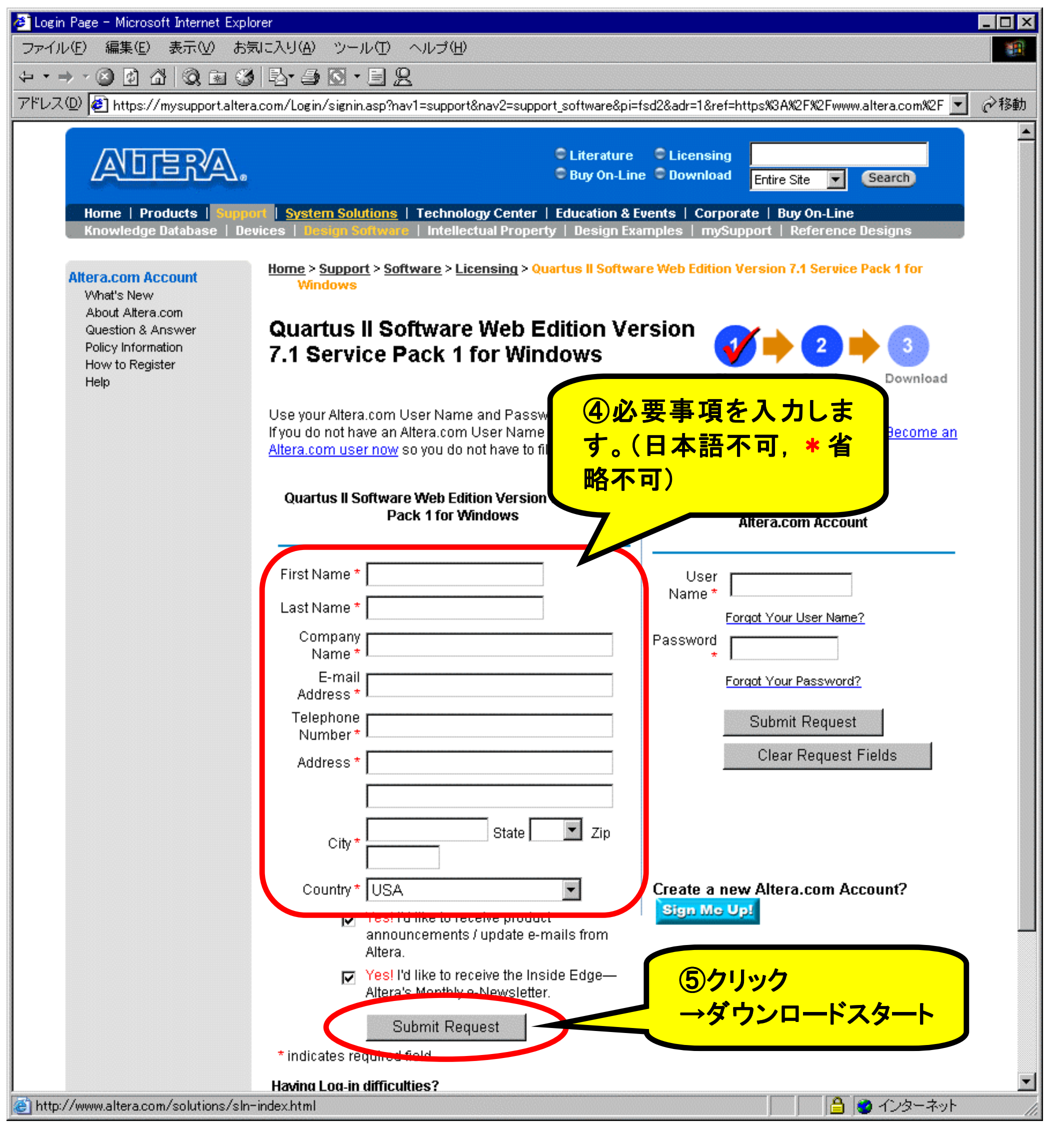
最近のパソコンなら，まず問題はありません。

1. Quartus Ⅱを手に入れよう

ALTERA は Cyclone を含む自社の FPGA や CPLD の開発ツールとして Quartus Ⅱ（クォルタス・ツー）というソフトウェアを提供しています。Quartus Ⅱは有償ですが，Quartus Ⅱ Web Edition という無償の評価版も用意されており，わたしたちが使うのはもちろん無償評価版です。ALTERA のホームページからダウンロードできます。まずは，この開発ツールの最新版を入手しましょう。ALTERA のアドレスは，「http://www.altera.co.jp」です（画面は 2007 年 7 月 9 日現在のものです）。



ダウンロード・センタ
②「Quartus Ⅱ Web Edition」の欄の「ソフトウェア」をクリック
Quartus II Web Edition ソフトウェア
参考：ダウンロード手順は，ここをクリックすると見ることができます。一度，目をとおすことをおすすめします。
③「Windows」の欄の「ダウンロード」をクリック



あとは画面の指示に従いダウンロードしてください。なお，Quartus II Web Edition は 683MB（CD-ROM1 枚分近く）あります。時間のあるときに気長に作業してください。言うまでもなく，ADSL や光回線などのブロードバンド環境でないとダウンロードは現実的ではありません。

2. Quartus II のインストール

ダウンロードしたファイルをダブルクリックしてインストールを開始します。あとは画面の指示に従ってください。Quartus II は英語のソフトなので，日本語入力には対応していません。文字は半角英数で入力してください。また，使用するフォルダ名やファイル名も半角英数のみとし，日本語や全角文字を含めないようにして下さい。

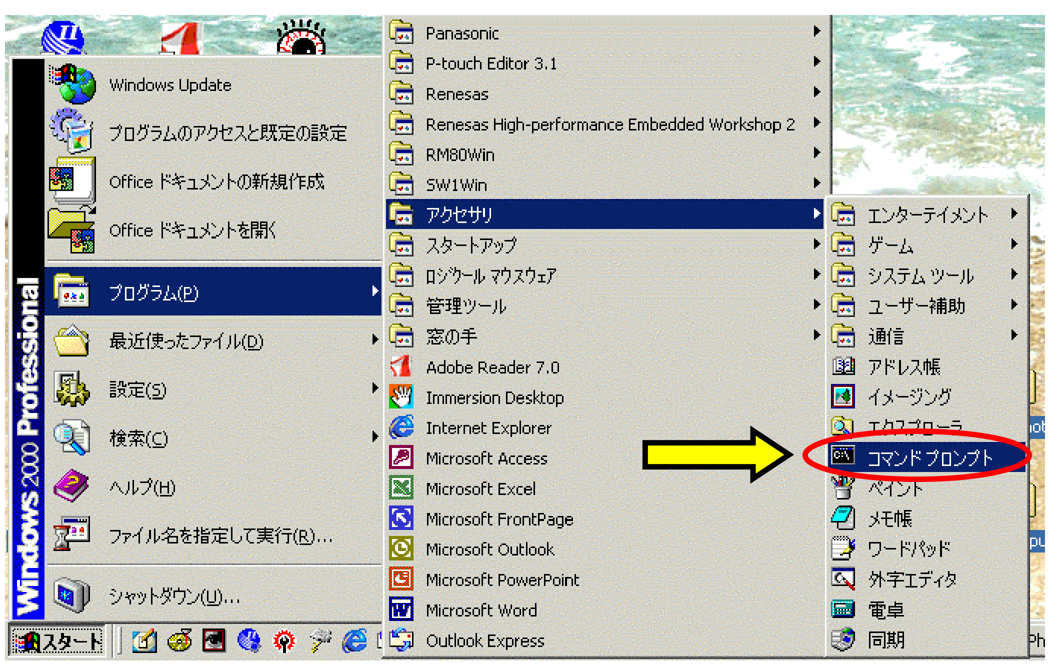
さて，インストールが終わったら Quartus II を起動したくなりますが，まだ起動しないで下さい。

■ ライセンスファイルを入手する

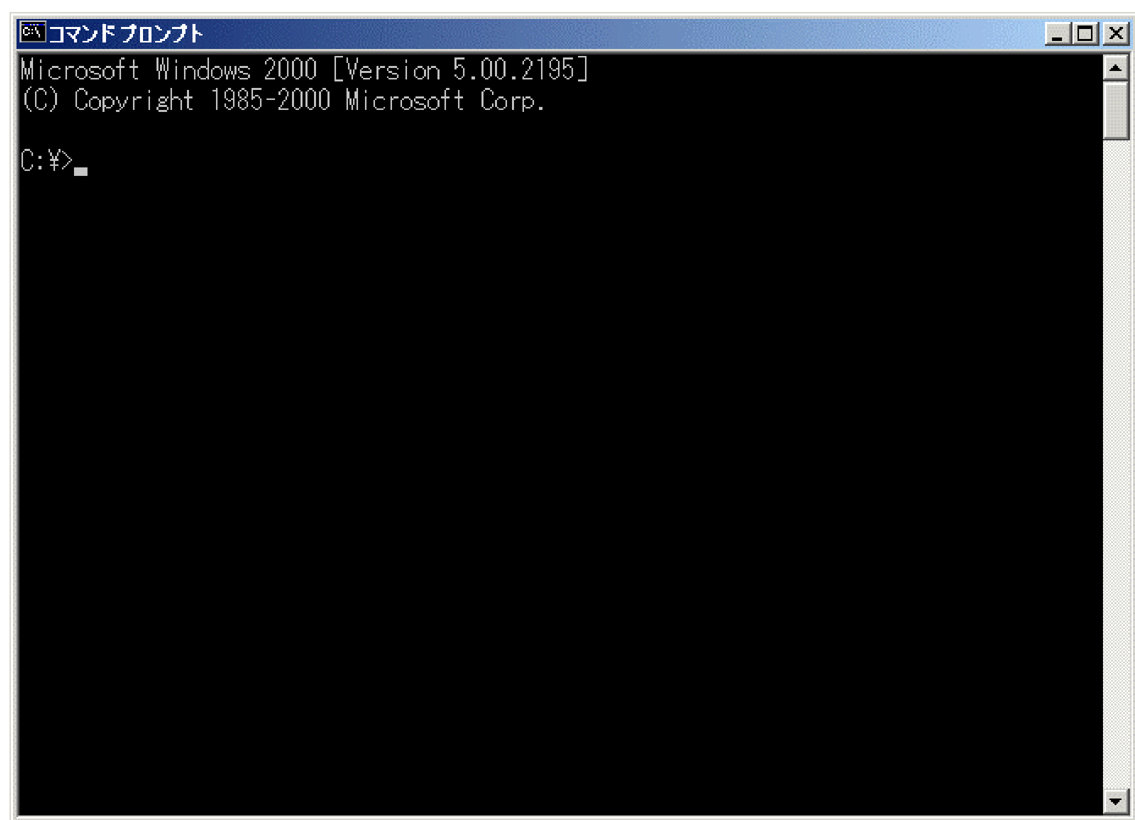
Quartus II Web Edition はパソコンにインストールしただけでは使うことができません。インターネット経由で名前やメールアドレスなどの情報を ALTERA に提供し，かわりにライセンスを発行してもらいます。ライセンスはデータファイルとしてメールアドレスに送られてきますが，このライセンスファイルを Quartus II に組み込むことで，使用できるようになります。ライセンスの有効期間は 150 日です。ライセンスは何度でも再発行してもらえますので，有効期間が切れたら新しいライセンスを入手しましょう。

なお，ライセンスファイル内には LAN カードの MAC アドレスに基づいたデータが組み込まれており，それによってパソコンを特定しています（MAC アドレスは固有の数値で世の中に同じものは存在しない）。そのため，一つのライセンスファイルで複数のパソコンの Quartus II を使うことはできません。必ず Quartus II をインストールしたパソコンごとに専用のライセンスを入手してください。

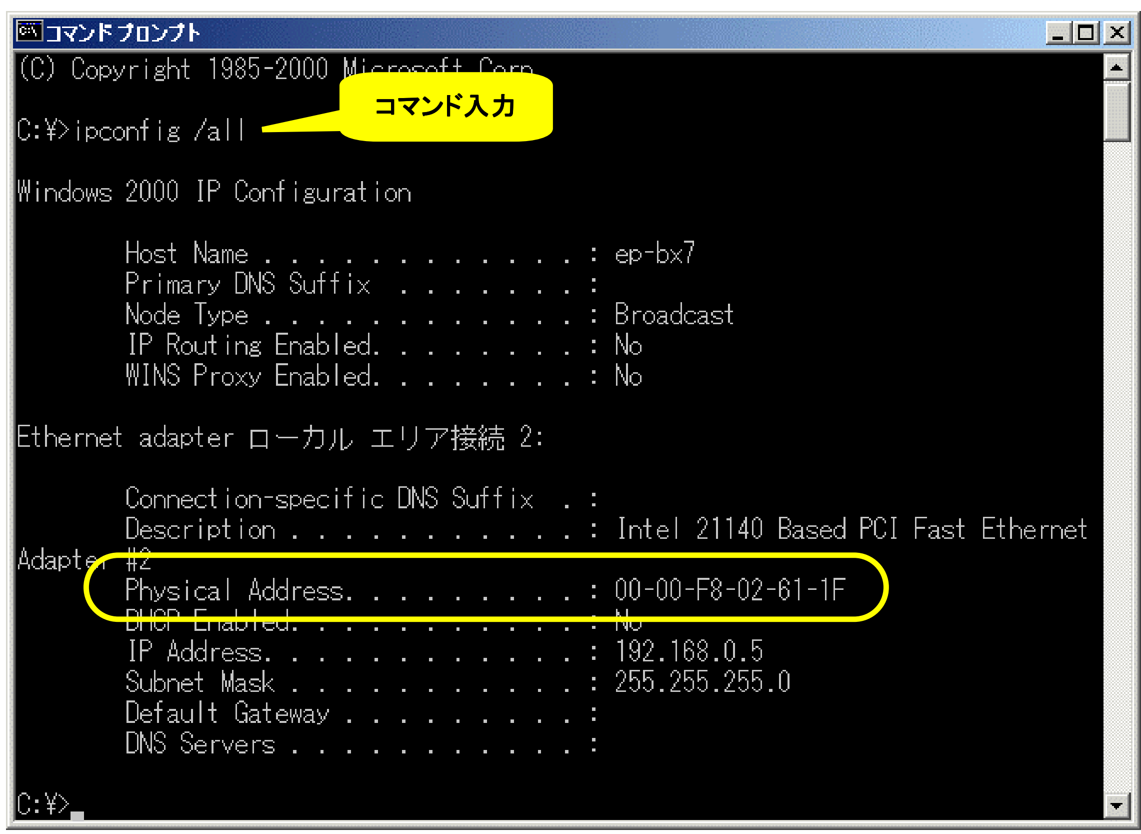
ではライセンスファイルを入手しましょう。まずは，パソコンに組み込まれている LAN カードの MAC アドレスを調べます。「コマンドプロンプト」を起動します。



すると，次のようなウィンドウが開き，コマンド入力待ちになります。

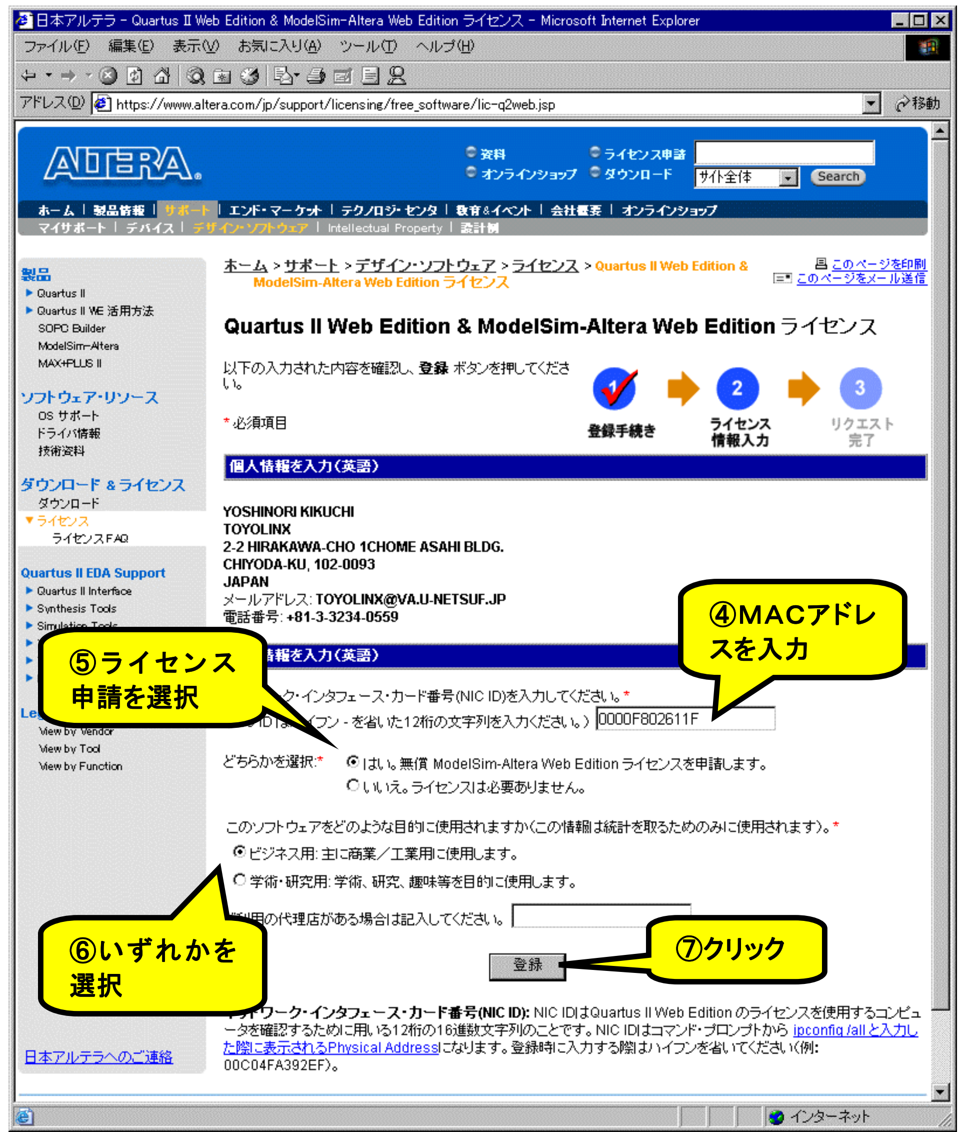


次に，「ipconfig /all」というコマンドを入力します。すると，次のように表示されます。



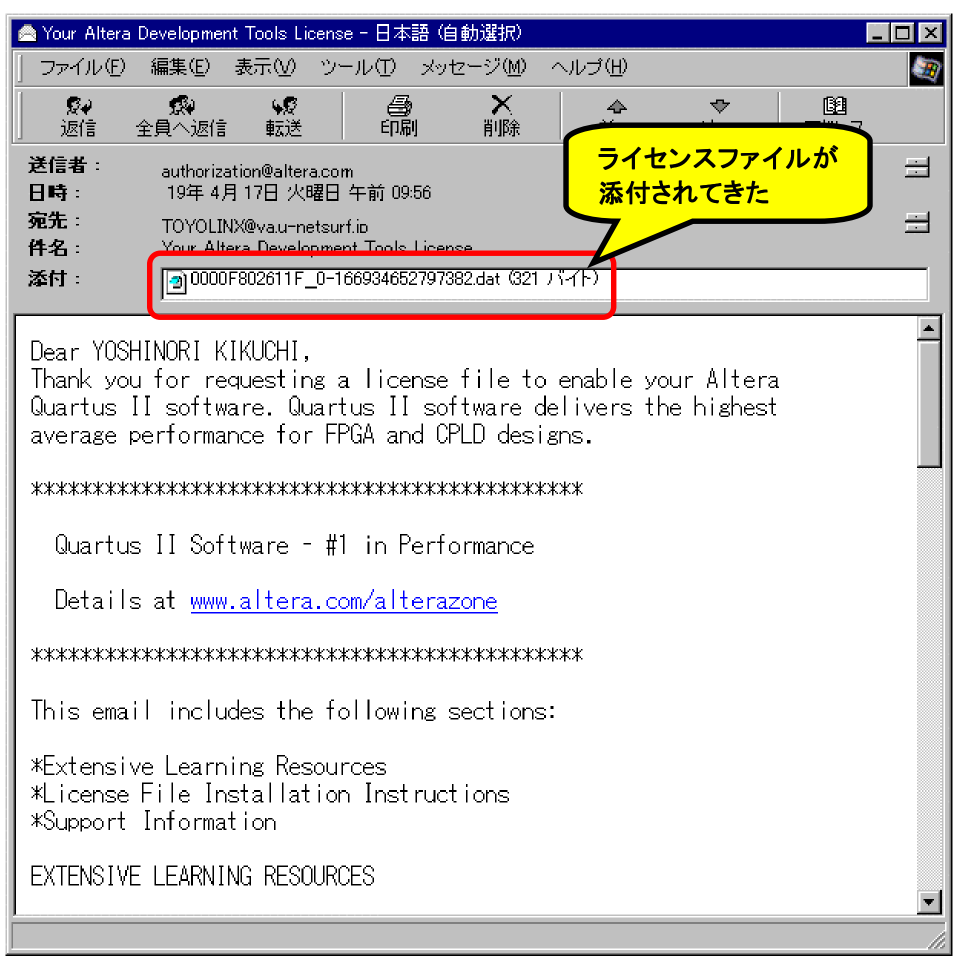
「Physical Address」が MAC アドレスです。当然ながら，パソコンごとに異なる数値になっています。この 16 進数の数字 6 個をメモしておきましょう。

次にインターネット上の ALTERA のライセンスの申請ページをアクセスします。ホームページ(http://www.altera.co.jp)の「ライセンス申請」をクリックします。

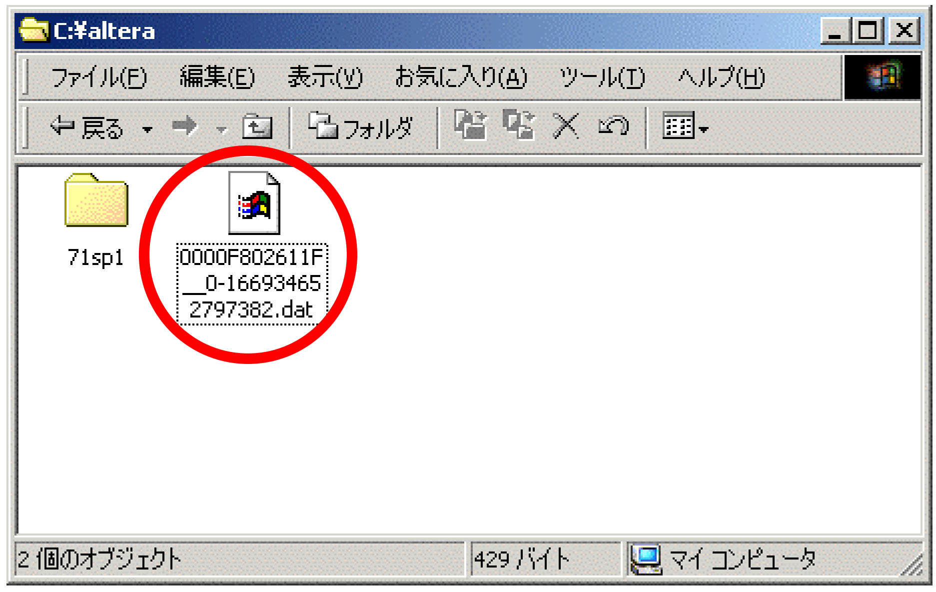


ipconfig で調べた MAC アドレスが「00-00-F8-02-61-1F」の場合，ネットワークインターフェースカード番号（NIC ID）は「0000F802611F」と入力します。

さて，しばらくすると ALTERA からライセンスファイルが添付されたメールが，先ほど登録したメールアドレス宛に送られてきます。もし，いくら待っても送られてこないようであれば，もう一度ライセンス申請してください。送られてこない理由のほとんどは，メールアドレスの入力ミスです。ライセンス申請は何度でも繰り返し行なうことができます。

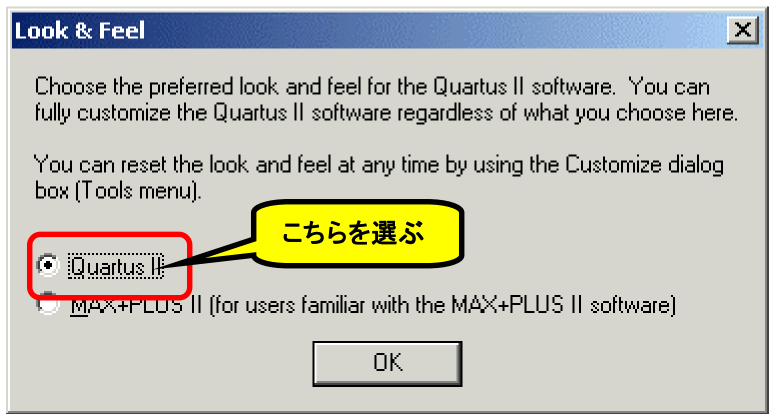


この添付ファイルを任意のフォルダ（例：c:¥altera）に保存します。あとでフォルダごと指定しますのでわかりやすい場所にして下さい。

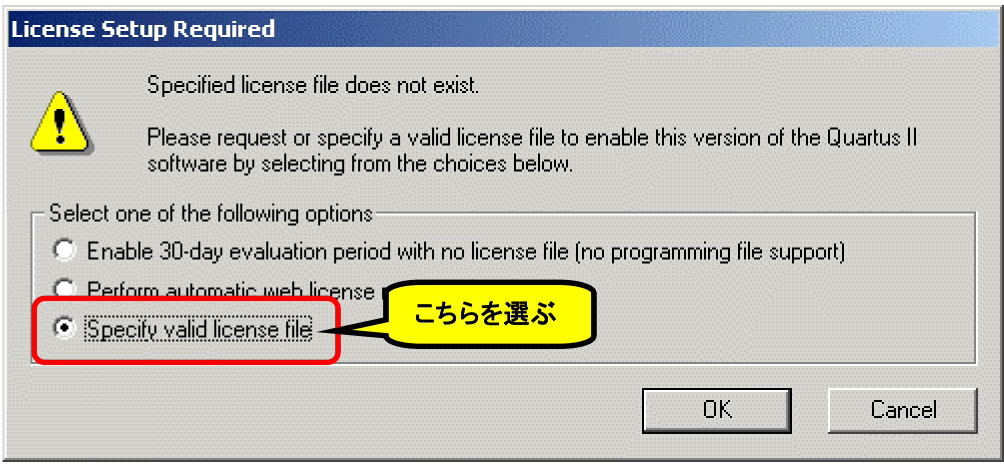


■ QuartusⅡにライセンスファイルを組み込む

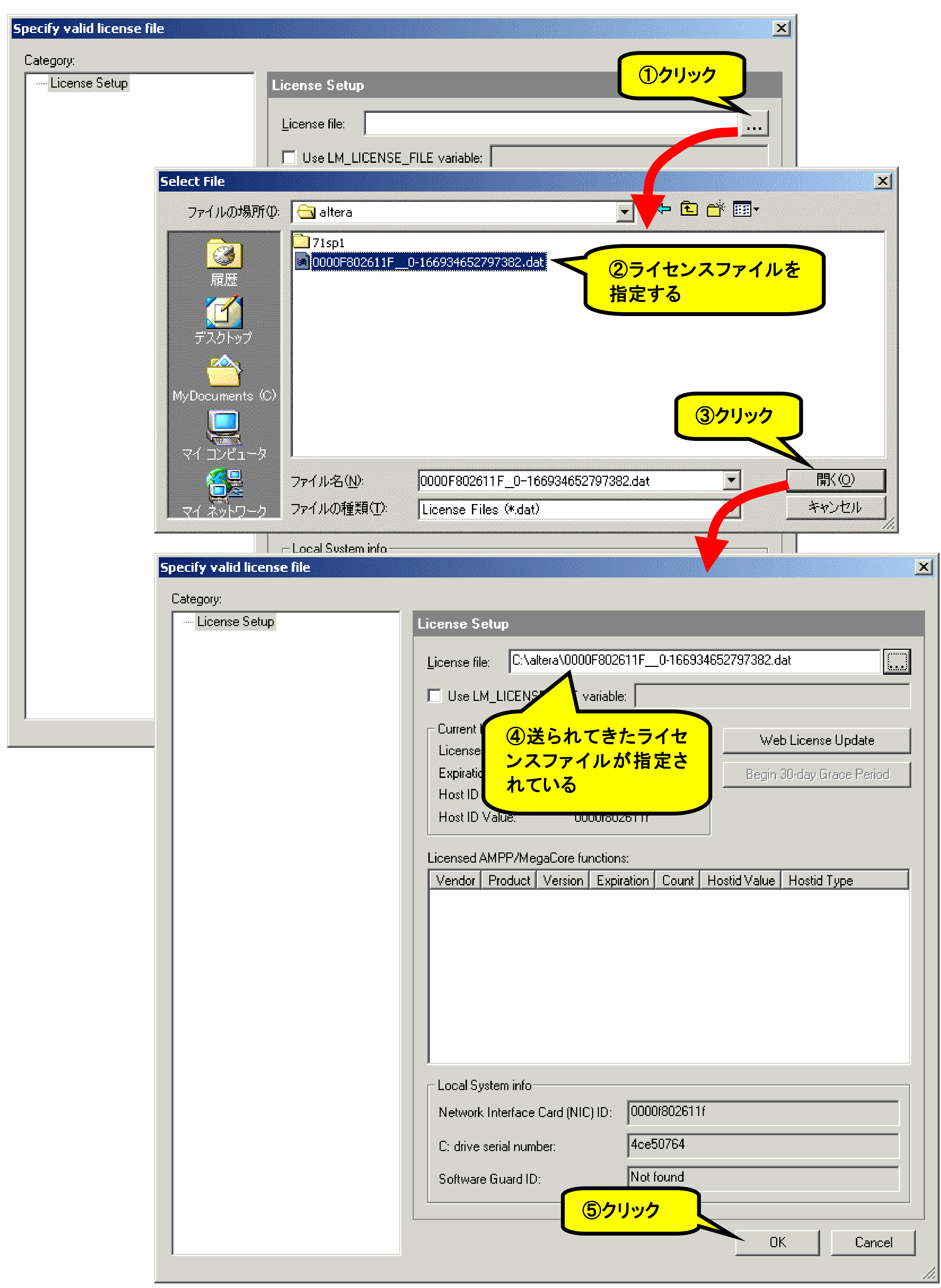
では，Quartus II を起動しましょう。最初に起動したときだけ，まず次のダイアログが開きます。「Quartus II」を選択します。



次に，ライセンスの形態を指定します。次のダイアログが開きますので，「Specify valid ricense file.」を選択します。



次にライセンス設定ダイアログが開きます。先ほど保存したライセンスファイルを指定します。

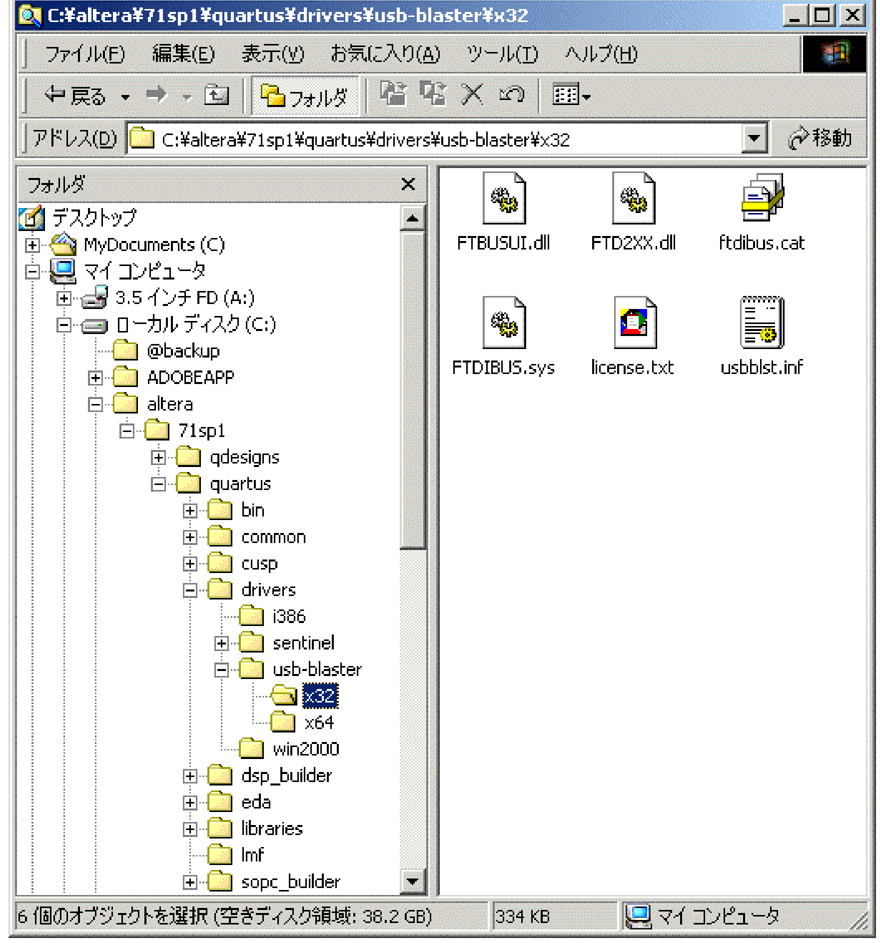


これで，ライセンスの登録は終了しましたので，150 日間 Quartus II を使用することができます。

3. ダウンロードケーブル

Quartus II で作成したコンフィグレーションデータは，専用のダウンロードケーブルで FPGA，もしくはコンフィグレーション ROM にダウンロードします。このマニュアルでは，ALTERA 純正の「USB-Blaster」，もしくは USB-Blaster 互換の「Terasic Blaster」を使用します。

まず，USB-Blaster，もしくは TerasicBlaster を利用するためにドライバをインストールします。ドライバはどちらも USB-Blaster のものを使います。Quartus II をインストールすると，USB-Blaster をはじめ，ALTERA のダウンロードケーブルのドライバがハードディスクにコピーされます。（右図参照）



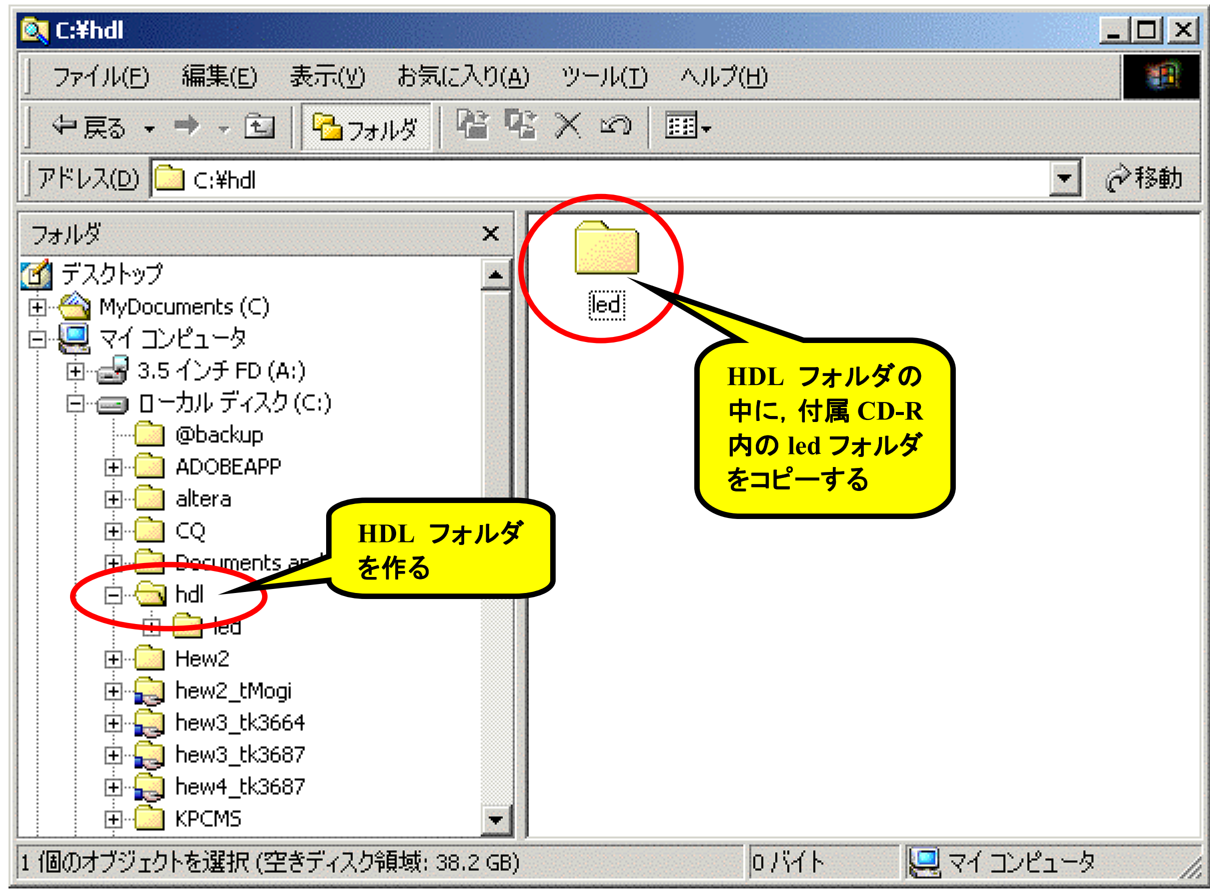
USB-Blaster，TerasicBlaster をパソコンの USB につないでください。自動的に検知し，ドライバのインストールが始まります。始まらない場合は，コントロールパネルの「ハードウェアの追加と削除」をクリックしてください。あとは，画面の指示にしたがいインストールを実行してください。

なお，以降マニュアルで「USB-Blaster」というときは互換品も含まれています。

4. データ保存フォルダの作成

Quartus II で作成するプロジェクトデータは任意の場所に作ることができますが，このマニュアルでは C ドライブの「hdl」フォルダの中に作成するものとして説明します。

また，付属の CD-R の中にはこのマニュアルの中で説明している HDL のサンプルデータが含まれています。説明を読みながら自分で入力する方が勉強になりますが，FPGA ボードの動作確認のため「led」フォルダだけはコピーしてください。



◆

これで準備が整いました。次の章では FPGA ボードの動作確認も兼ねて，あらかじめ用意されているコンフィグレーションデータを Cyclone に書き込んでみましょう。

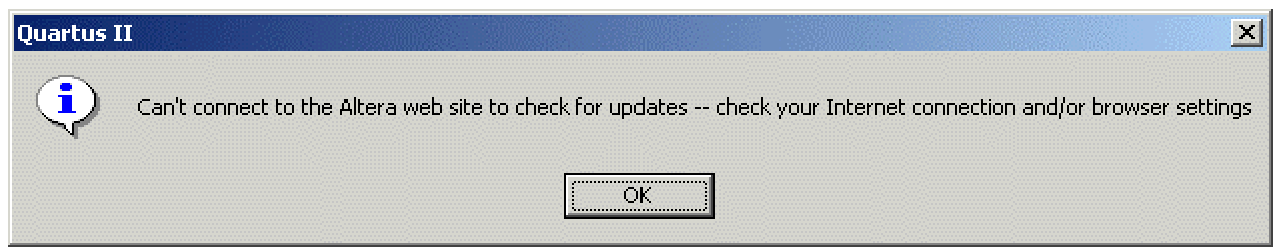
第4章	Cyclone に回路を書き込んでみよう
	1. Cyclone に回路を書き込む 2. コンフィグレーション ROM に回路を書き込む

1. Cyclone に回路を書き込む

今回書き込む回路は，前の章でコピーした「led」です。これは，7 セグメント LED を 1 箇所だけ点灯する回路です。

まずは「led」フォルダを開いて下さい。この中に「led. qpf」というファイルがあります。これをダブルクリックすると Quartus II が起動し，led プロジェクトが開きます。

途中，次のようなダイアログが開く場合もありますが，「OK」をクリックして先に進んでください。

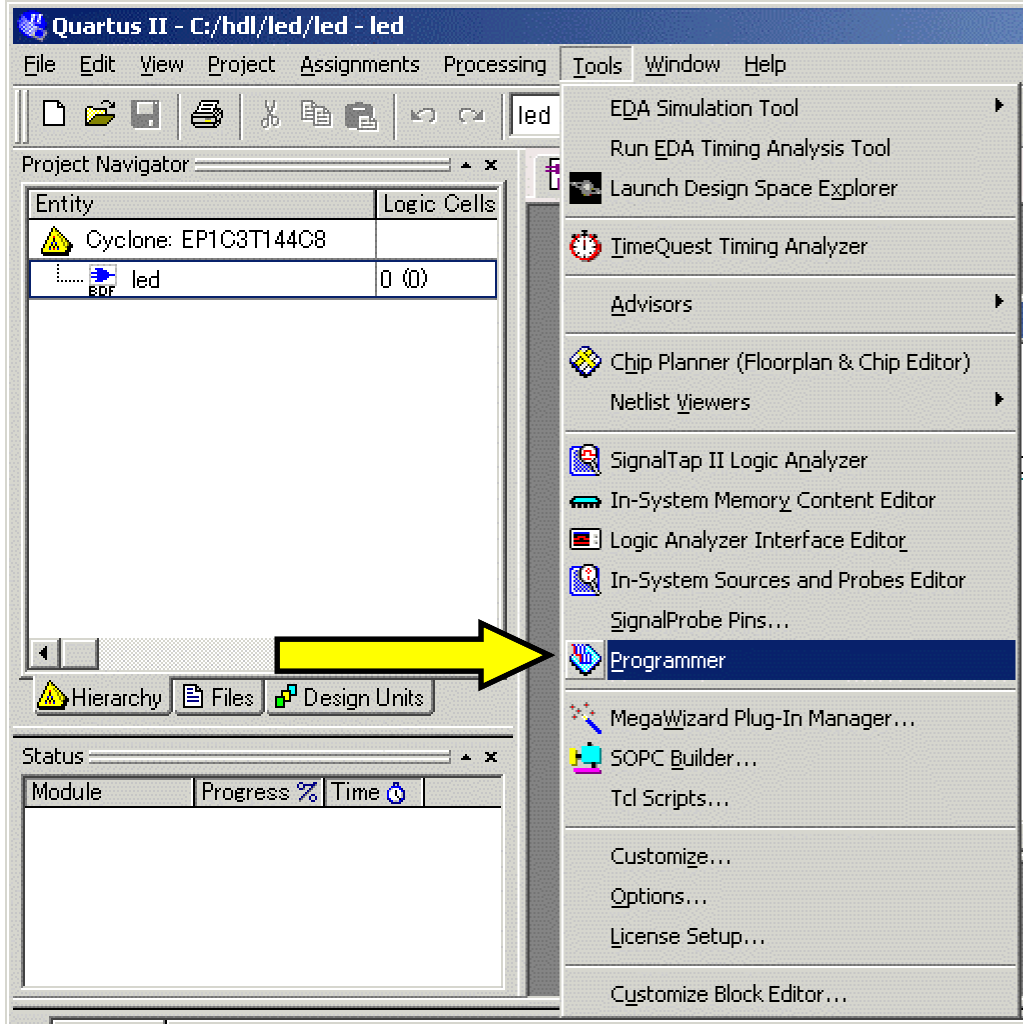


(ライセンスの有効期間が切れると，本マニュアル 16 ページ下のダイアログが開きます。そのときは，アルテラにもう一度ライセンスを申請し，新しいライセンスファイルを登録してください。)

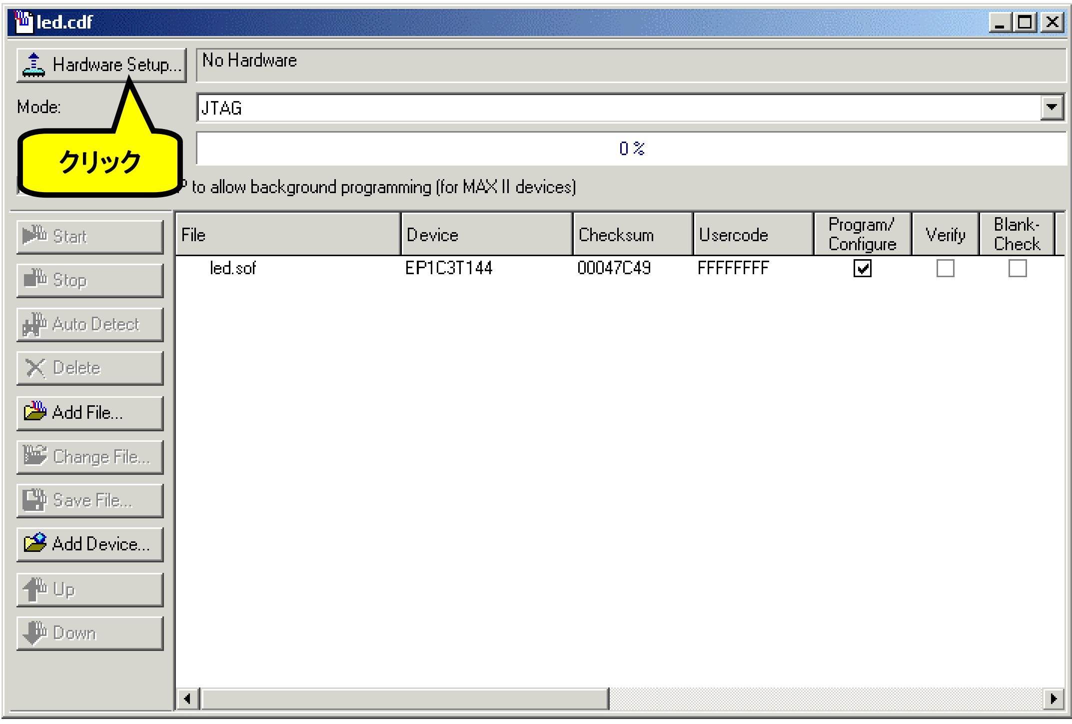
led プロジェクトが開くと Quartus II の画面は次のようになります。

パソコンの USB ポートに USB-Blaster の USB ケーブルを接続し，FPGA ボードの CN3 に USB-Blaster のフラットケーブルを接続します。そして，FPGA ボードの電源をオンにします。

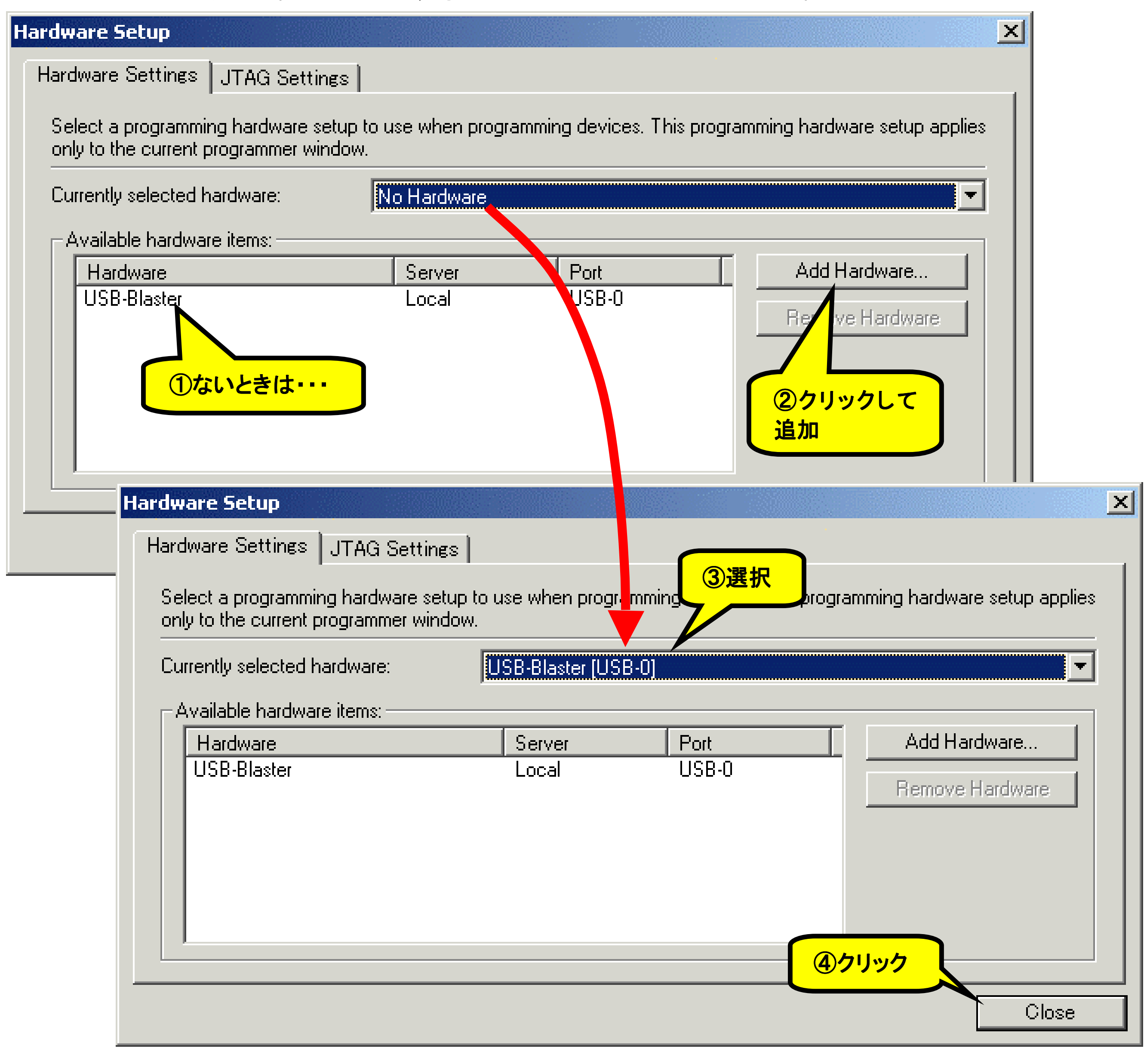
次に，ダウンロードケーブルのセットアップを行ないます（ドライバのインストールとは違いますよ）。Quartus II のメニューから［Tools］→［Programmer］をクリックします。



すると，書き込みツールが起動します。「Hardware Setup」ボタンをクリックしてください。

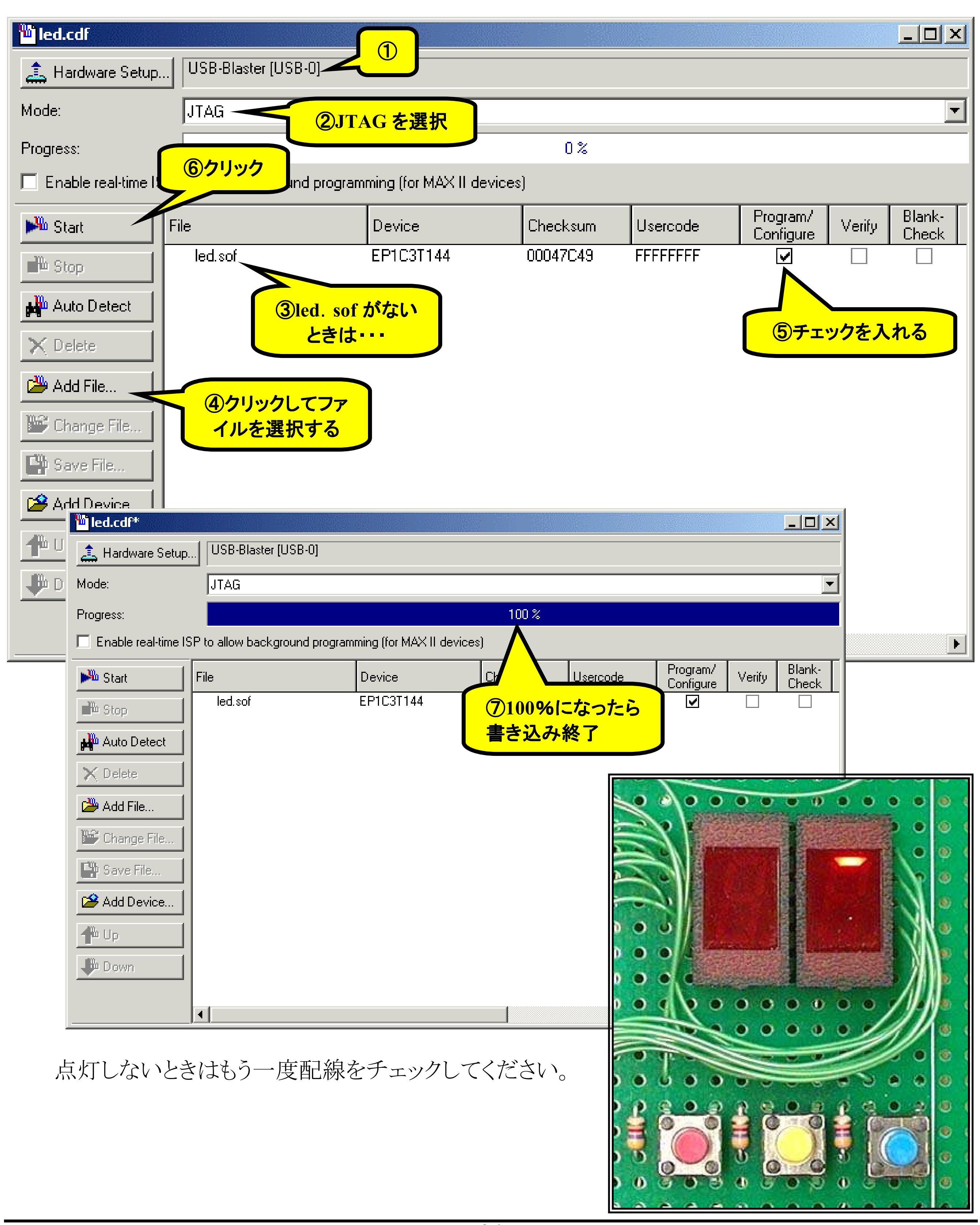


「Hardware　Setup」ダイアログが開きます。①「Available　hardware　items」の中に「USB-Blaster」があるか確認して下さい。ないときは，②「Add　Hardware」をクリックして追加します。その後，③「Currently　selected　hardware」のプルダウンメニューから「USB-Blaster[USB-n]」を選びます(n は Port 番号)。設定したら，④「Close」をクリックして閉じます。



すると，①USB−Blaster がセットされます。次に，②「Mode」が「JTAG」になっているか確認して下さい。なっていないときはプルダウンメニューから「JTAG」を選択します。③「File」に「led．sof」がセットされているか確認します。セットされていないときは，④「Add　File」をクリックして「led．sof」を選択します。⑤「Program/Configure」にチェックを入れます。

これで，準備完了です。⑥「Start」をクリックしてください。書き込みが始まります。⑦「Progress」が 100％になったら書き込み終了です。下の写真のように 7 セグメント LED が点灯するはずです。



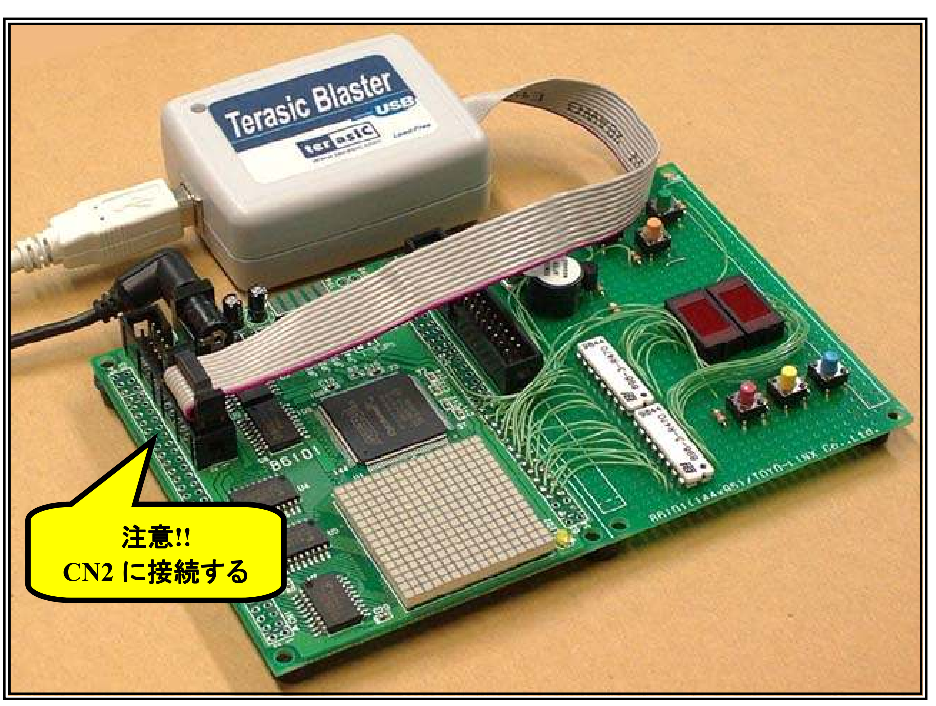
点灯しないときはもう一度配線をチェックしてください。

2. コンフィグレーション ROM に回路を書き込む

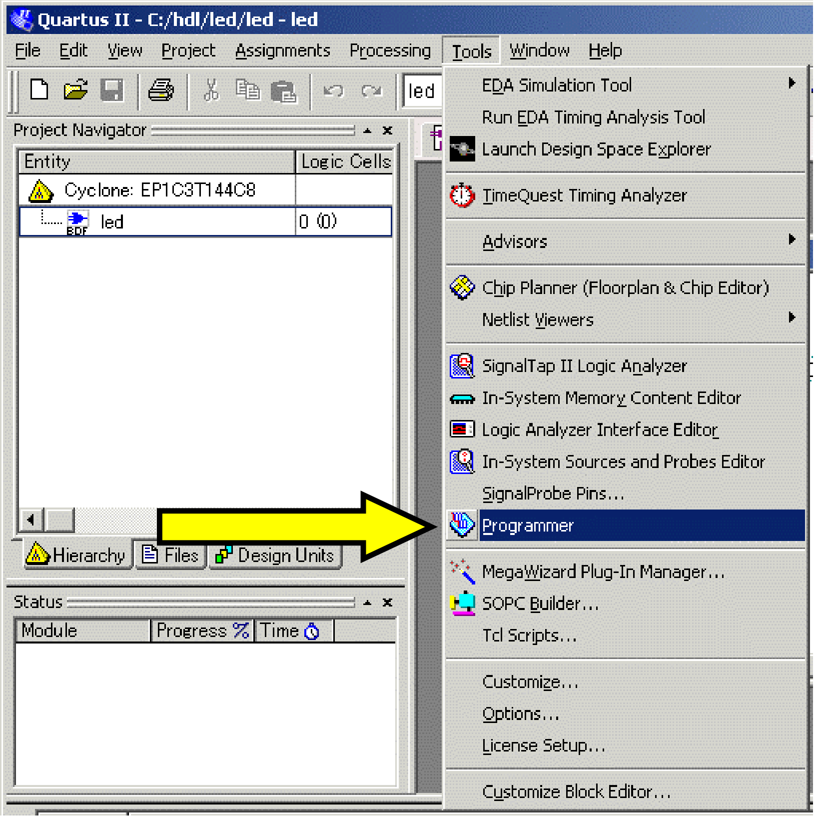
さて，前節で Cyclone の中に回路データ(コンフィグレーションデータ)を書き込むことができました。ただ，前にも述べたように Cyclone の論理を構成する基本要素は SRAM でできています。それで，電源をオフにするときれいさっぱり忘れてしまいます。ためしに，FPGA ボードの電源をオフにしてみてください。次に電源をオンにしても LED は点灯しません。

当然これでは製品としては使い物になりません。そこで，コンフィグレーションデータを不揮発性メモリに書き込んでおいて，電源オンで Cyclone に読み込ませることにします。このメモリがコンフィグレーション ROM です。次に，コンフィグレーション ROM に書き込んでみましょう。

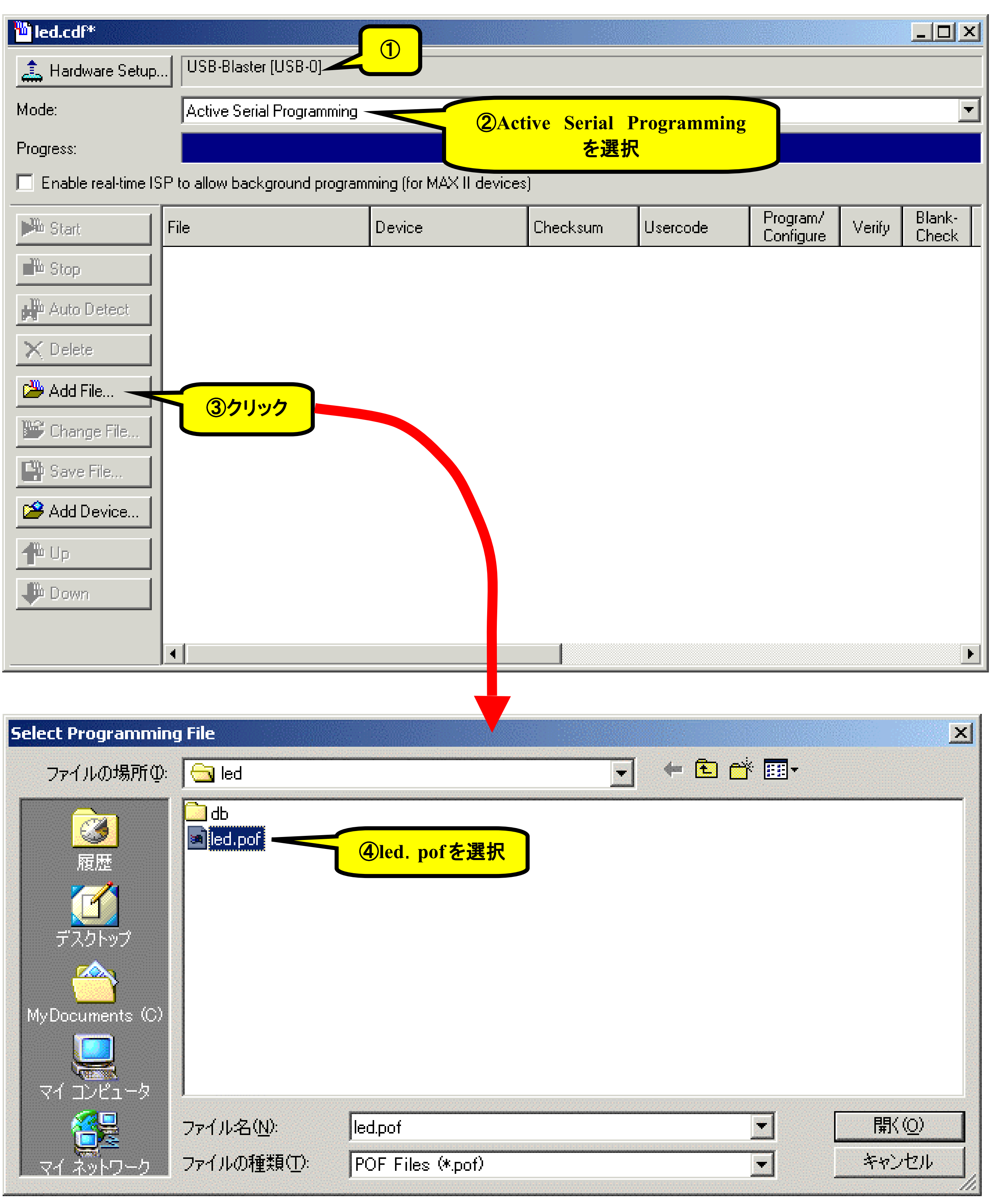
USB-Blaster のフラットケーブルを FPGA ボードの CN2 に接続します。



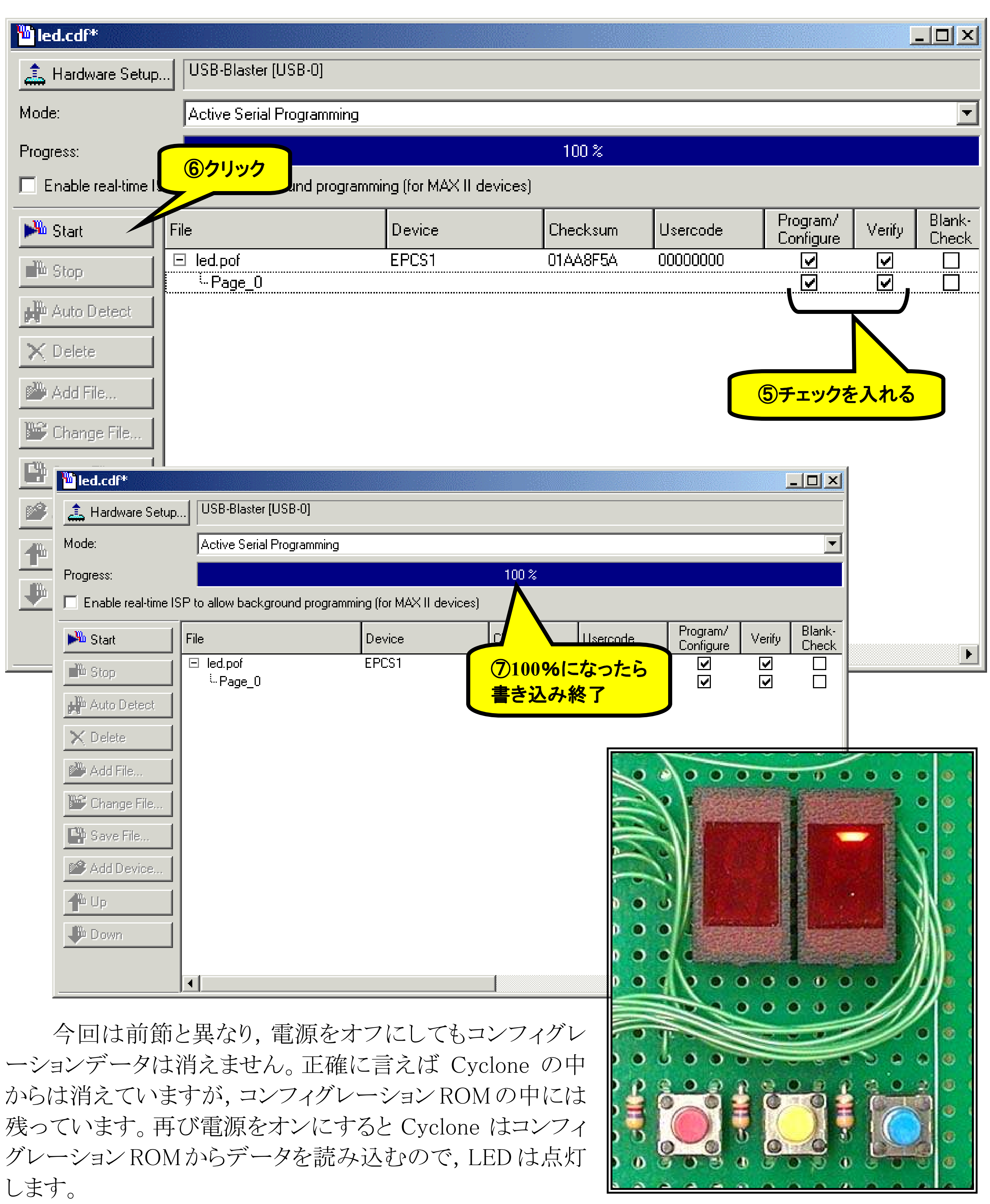
次に，書き込みツールを起動します。Quartus II のメニューから［Tools］→［Programmer］をクリックします。



まず，①USB-Blaster になっていることを確認して下さい。次に，②｢Mode｣のプルダウンメニューから｢Active　Serial　Programming｣を選択します。そして，③｢Add　File｣をクリックして④｢led．pof｣を選択します。



ファイルがセットされたら，⑤「Program/Configure」と「Verify」にチェックを入れます。これで，準備完了です。⑥「Start」をクリックしてください。書き込みが始まります。⑦「Progress」が 100％になったら書き込み終了です。下の写真のように 7 セグメント LED が点灯するはずです。



今回は前節と異なり，電源をオフにしてもコンフィグレーションデータは消えません。正確に言えば Cyclone の中からは消えていますが，コンフィグレーション ROM の中には残っています。再び電源をオンにすると Cyclone はコンフィグレーション ROM からデータを読み込むので，LED は点灯します。

第5章 基本的な回路を入力して Cyclone に書き込んでみよう

1. 基本ゲートの回路図入力
2. 基本ゲートの VHDL 入力
3. デコーダとエンコーダ
4. フリップフロップ
5. カウンタ
6. シフトレジスタ
7. VHDL の記述方法について

前の章では用意された簡単な回路(といっても，52 番ピンを Vcc に固定しただけですが･･･)を書き込んでみました。この章ではさらに一歩進めて，一から回路を入力して Cyclone に書き込むところまで実習してみましょう。でも，やっぱり最初はいちばん簡単なところから始めます。

1. 基本ゲートの回路図入力

まずはおさらいです。論理回路を構成するもっとも基本的なゲートはAND・OR・NOTの3種類です。さらに，これらを組み合わせることで，NAND・NOR・EXOR といった，よく使われる基本的なゲートも作ることができます。真理値表は次のとおりです。

AND			OR			NOT		NAND			NOR			EXOR		
74HC08			74HC32			74HC04		74HC00			74HC02			74HC86		
入力		出力	入力		出力	入力	出力	入力		出力	入力		出力	入力		出力
A	B	Y	A	B	Y	A	Y	A	B	Y	A	B	Y	A	B	Y
0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0
1	0	0	1	0	1	1	0	1	0	1	1	0	0	1	0	1
0	1	0	0	1	1			0	1	1	0	1	0	0	1	1
1	1	1	1	1	1			1	1	0	1	1	0	1	1	0

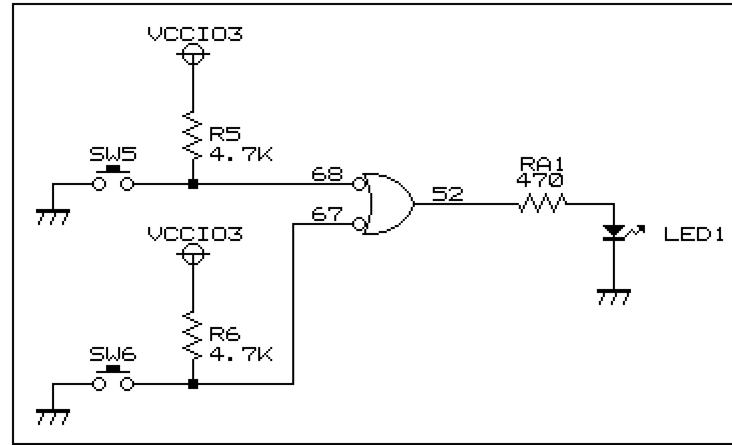
さて，上の回路記号は入力条件が‘1’になるときに注目しています。しかし，入力条件が‘0’になるときに注目すると次のように書き変えることができます。(EXOR は例外)

入力条件を‘1’で考えると(正論理)	74HC08	74HC32	74HC04	74HC00	74HC02
入力条件を‘0’で考えると(負論理)	74HC08	74HC32	74HC04	74HC00	74HC02

上の段と下の段は，真理値表は同じですが，入力条件で注目している論理が逆で，条件が成立したときの出力の論理も逆になっています。また，論理自体も，ANDとORは入れ替わっています。つまり，AND や OR は論理を逆にすることで変換できる，ということになります。これを「ド・モルガンの定理」と呼び，論理回路の基本的な定理です。

さて，論理回路の考え方のコツですが(経験からです)，あまり入出力の条件を考えて，ここはANDを正論理で使う，ここはNORを負論理で使う，とか考えないほうが良さそうです。それよりも，「全ての入力が･･･になったら出力する」場合は AND，「入力の何れか一つが･･･になったら出力する」場合は OR と考え，「･･･」が‘1’のときはそのままで‘0’のときは○をつける，‘1’を出力するときはそのままで‘0’を出力するときは○をつける，と考えた方がよいです。このように考えると，真理値表を作ったり見たりしなくても入出力の条件が頭に浮かびます。例えば，上の表の NAND の負論理を見てください。この記号を見ただけで OR の動作だということがわかります。入力の両方に○がついているので「‘0’になったら出力するんだ」，出力には○がついていないので「‘1’を出力するんだ」，ということが読み取れます。まとめると，「入力のいずれかが‘0’になったら出力を‘1’にする，それ以外の出力は‘0’になる」となります。

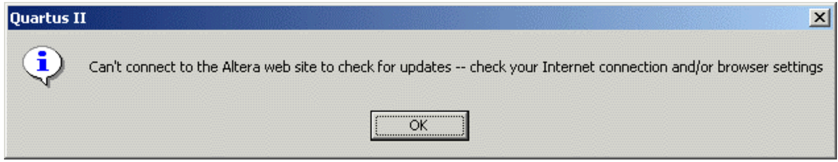
さて，おさらいはここまでにして，Cyclone に戻りましょう。2 入力の NAND 回路を FPGA に書き込んでみましょう。回路図に書くとこんな感じです。



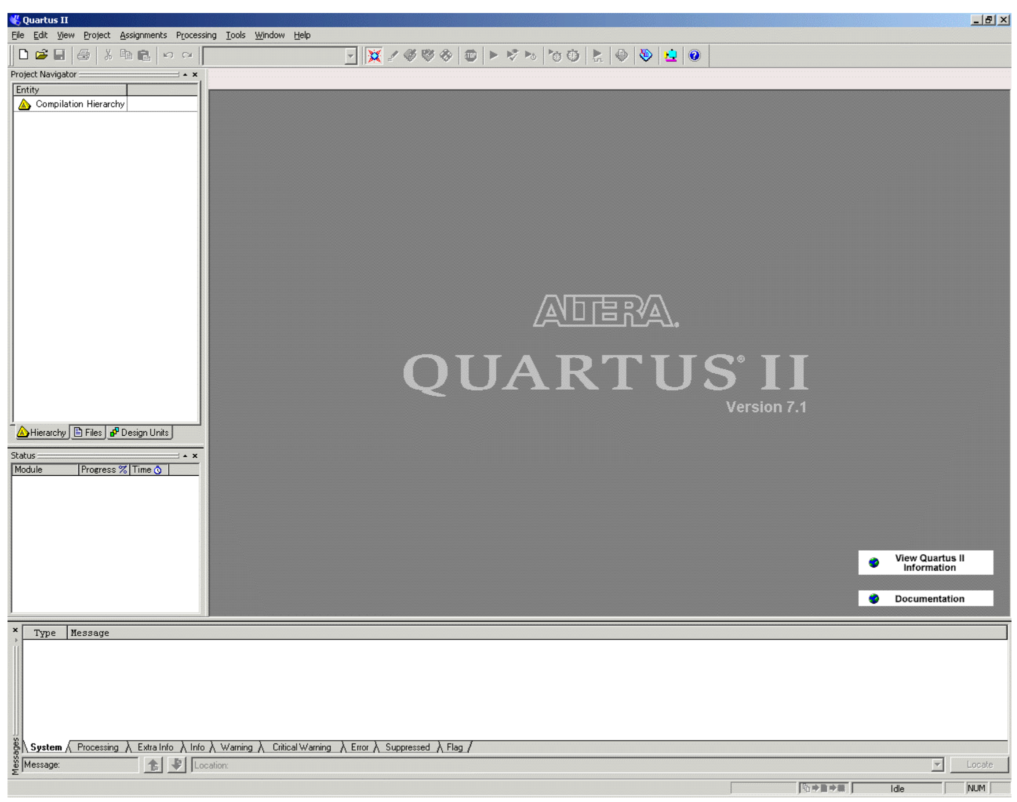
負論理で書いていますが NAND です。いずれかのスイッチが ON になると LED が点灯します。このうち，スイッチ，LED，抵抗はユニバーサルエリアに組み立て済みで Cyclone まで配線済みです（番号は Cyclone のピン番号）。あとは NAND の動作をするよう Cyclone に書き込むだけです。

■ QuartusⅡの回路図エディタで入力する

まずは QuartusⅡを起動してください。例によって，

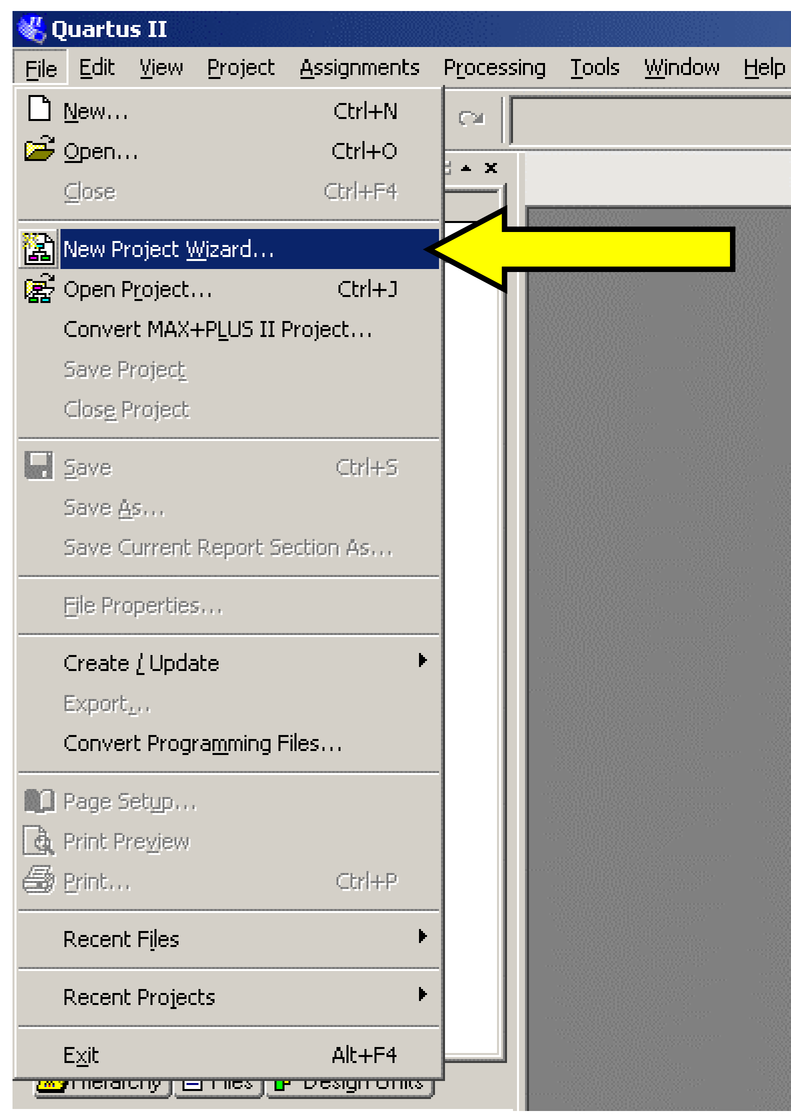


というダイアログが表示されることもありますが，「OK」をクリックして先に進みます。最終的に次のような画面になります。

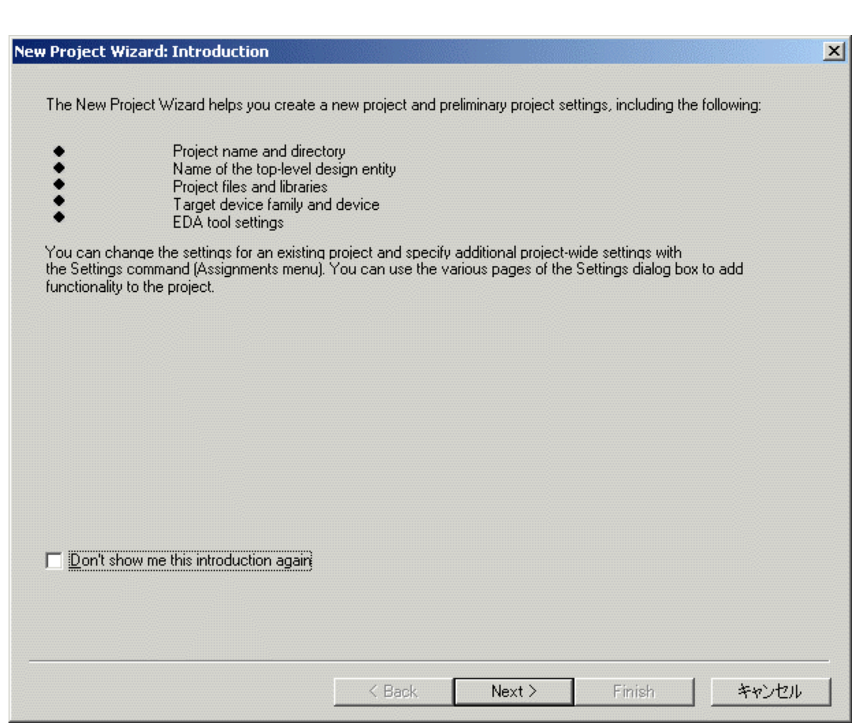


Quartus II では回路のデータを「プロジェクト」として扱います。それで，まずは新しいプロジェクトを作りましょう。

Quartus II のメニューから[File]→[New Project Wizard]をクリックし，プロジェクト作成ウィザードを起動します。

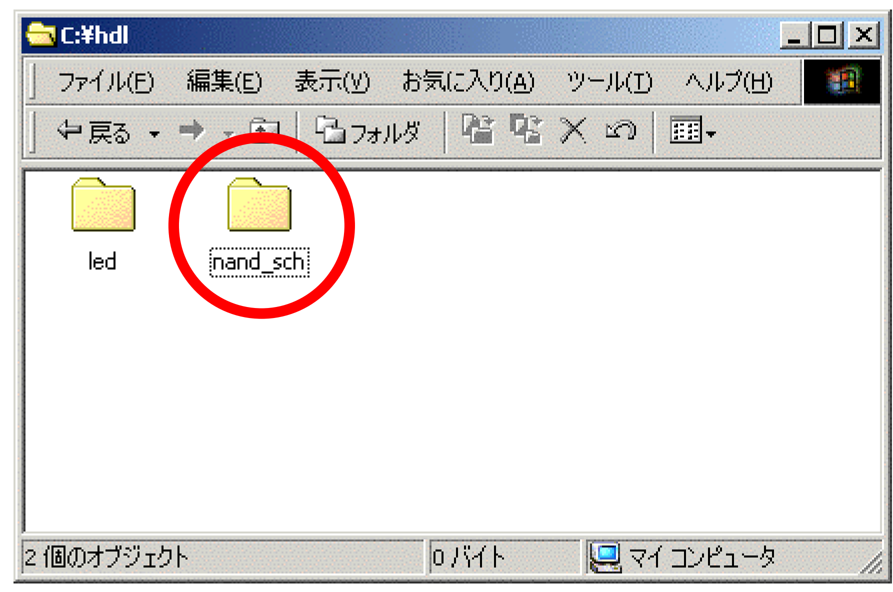


最初に，次のダイアログが表示されます。「Next」をクリックして次の画面に移ります。



「page 1 of 5」で，①プロジェクトを保存するフォルダ，②プロジェクト名，③回路の最上位の階層のエンティティ名を指定します。指定したら④「Next」をクリックします。

プロジェクト名は「nand_sch」にします。それで，フォルダも同じ名前，「C:¥hdl¥nand_sch」フォルダにします。Quartus II はまとまった機能を持つ回路ごとに階層構造で作成することができます。それぞれの回路をエンティティと呼びます。複雑な回路の場合，小さなエンティティを組み合わせて階層構造をとったほうが，わかりやすく効率的に回路設計を行なうことができます。最上位階層のエンティティは「nand_sch_top」という名前にします。



New Project Wizard: Directory, Name, Top-Level Entity [page 1 of 5]

What is the working directory for this project?
c:\hdl\nand_sch ①

What is the name of this project?
nand_sch ②

What is the name of the top-level design entity for this project? This name is case sensitive and must exactly match the entity name in the design file.
nand_sch_top ③

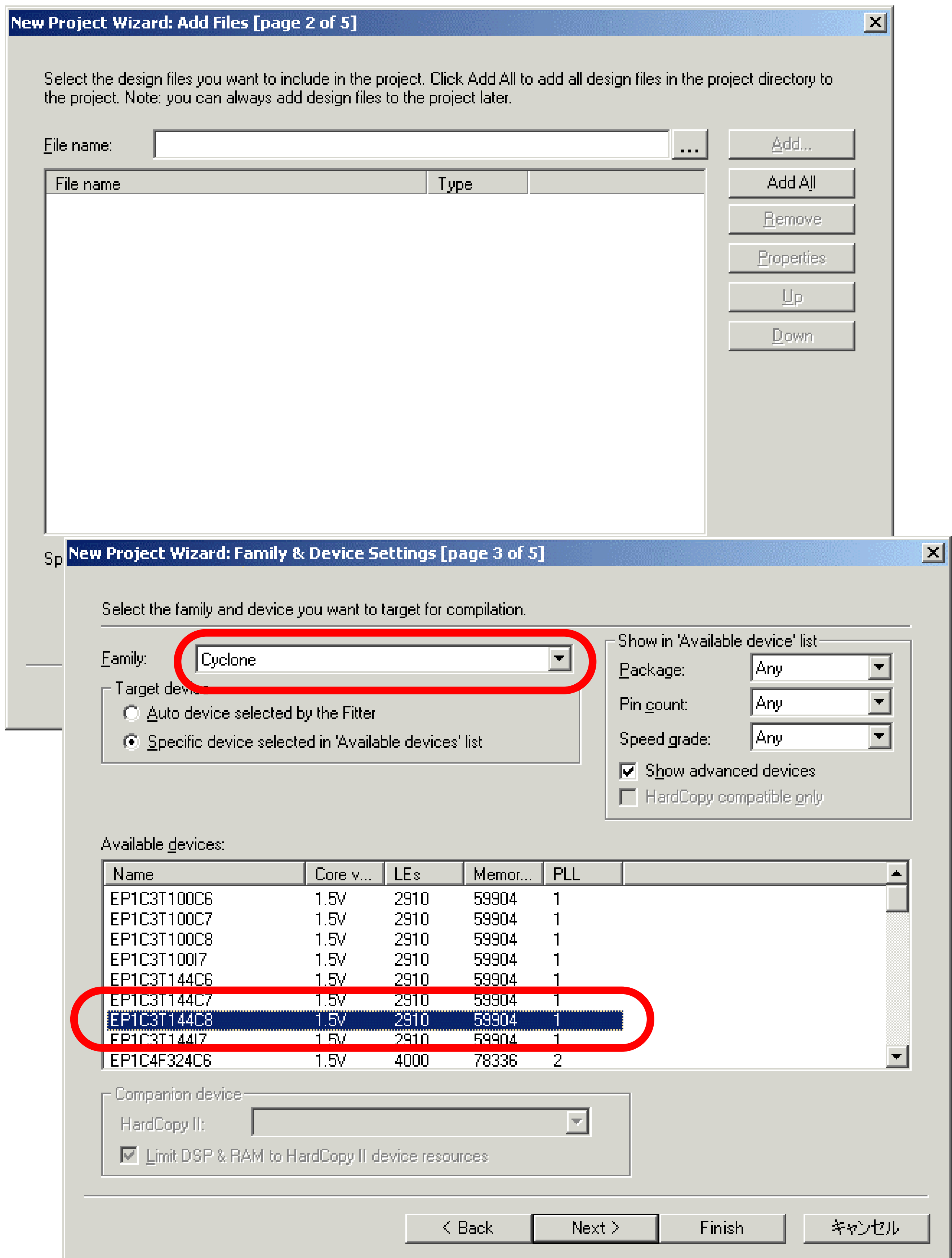
Use Existing Project Settings ...

④
< Back | Next > | Finish | キャンセル

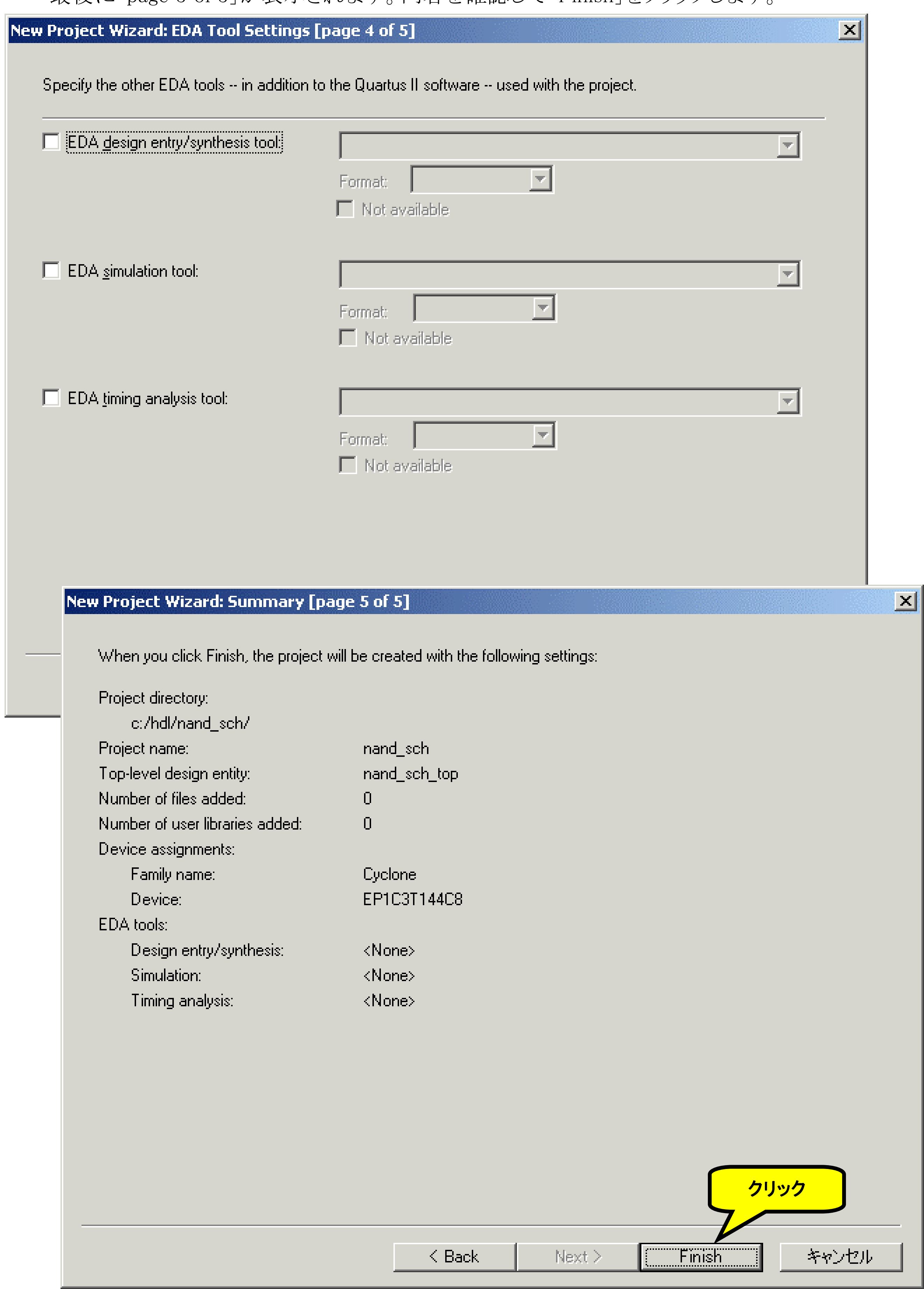
なお，Quartus II は英語版のソフトなので日本語入力には対応していません。フォルダを表す「¥」記号も「\」になっています。全角文字やカタカナを使うと識別不能で何が何だかわからなくなります。というわけで，Quartus II で文字を入力するときは半角英数のみにして下さい。

「page 2 of 5」は特に変更しません。「Next」をクリックします。

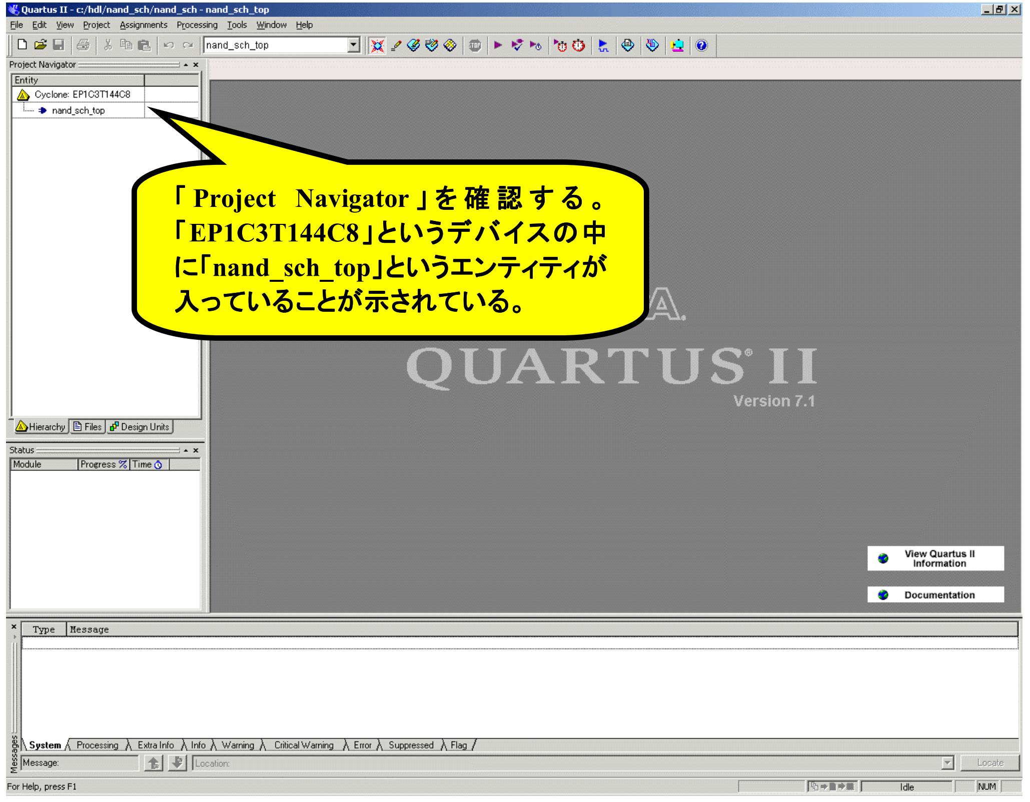
「page 3 of 5」で使用する FPGA の種類を指定します。FPGA ボード'B6101'に搭載されている FPGA は，Cyclone ファミリーの EP1C3T144C8 というデバイスです。それで，「Family」は「Cyclone」を，「Available Device」は「EP1C3T144C8」を選びます。指定したら「Next」をクリックします。



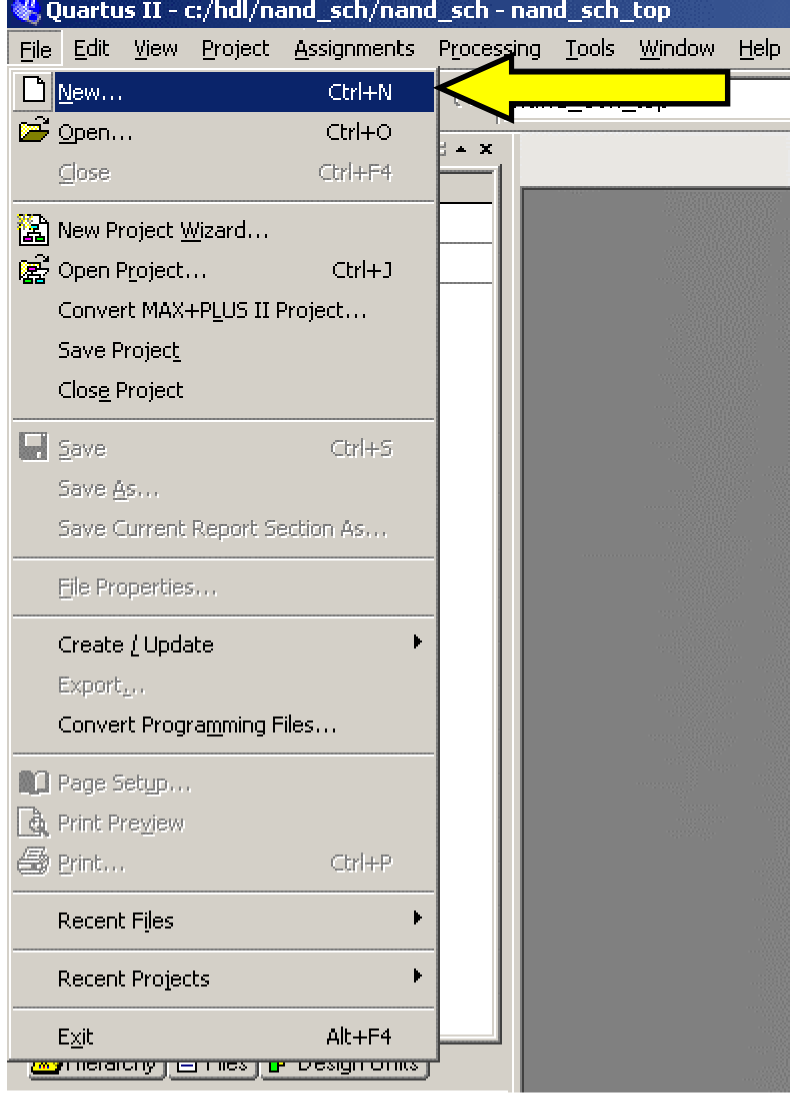
「page 4 of 5」は特に変更しません。「Next」をクリックします。
最後に「page 5 of 5」が表示されます。内容を確認して「Finish」をクリックします。



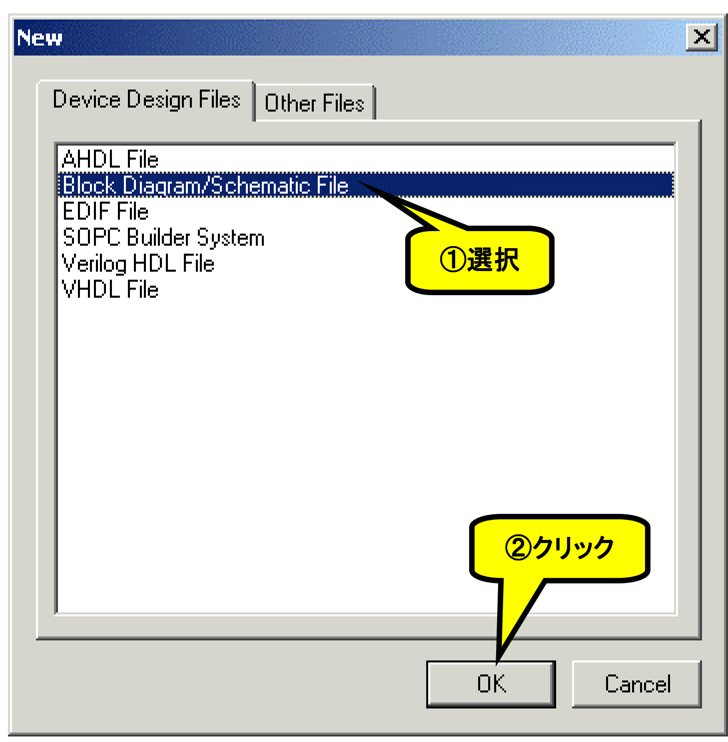
これでプロジェクトが完成しました。新規プロジェクト作成直後は右のような画面になります。



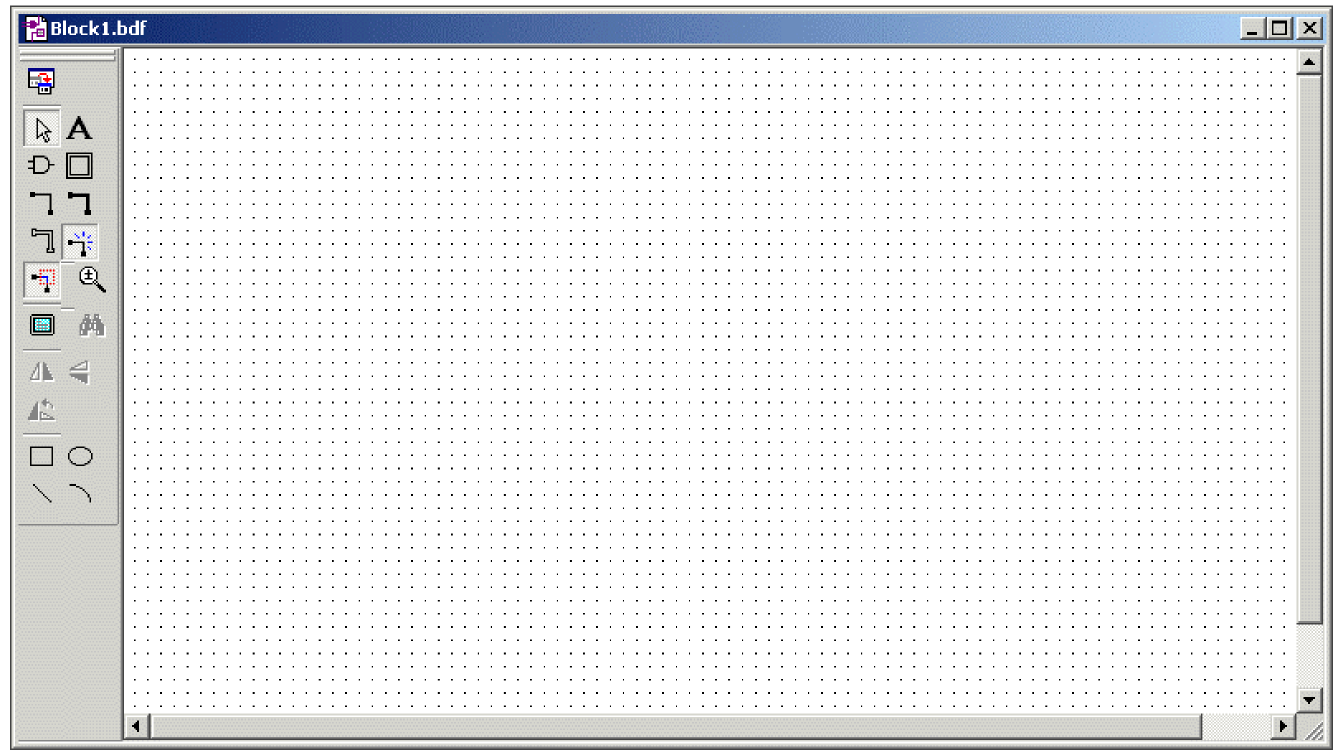
では，回路を入力しましょう。回路が簡単なので最上位階層である「nand_sch_top」エンティティに入力します。Quartus II のメニューから[File]→[New]をクリックします。



そうすると次のようなダイアログが表示されます。いくつかある選択肢の中から①「Block Diagram/Schematic File」を選び，②「OK」をクリックします。

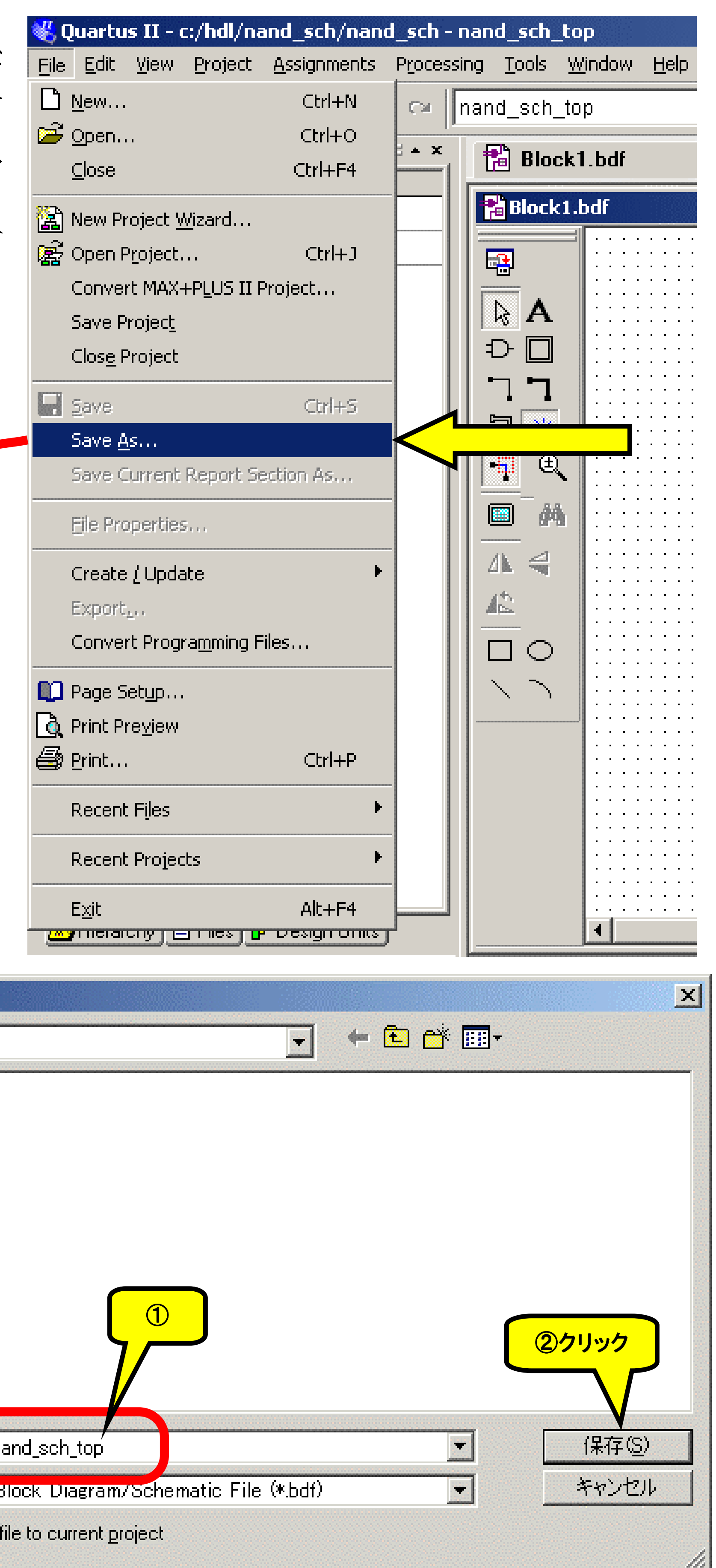


すると「Block1．Bdf」という名前のウィンドウが開きます。これが回路図エディタになります。



ただ，この「Block1．Bdf」という名前は Qurtus II が勝手に付けたものなので，最上位階層のエンティティ名に変更してセーブしておきましょう。Quartus II のメニューから［File］→［Save As］をクリックします。

「Save As」ダイアログが開きますので，①ファイル名に「nand_sch_top」と入力して②「保存」をクリックします。

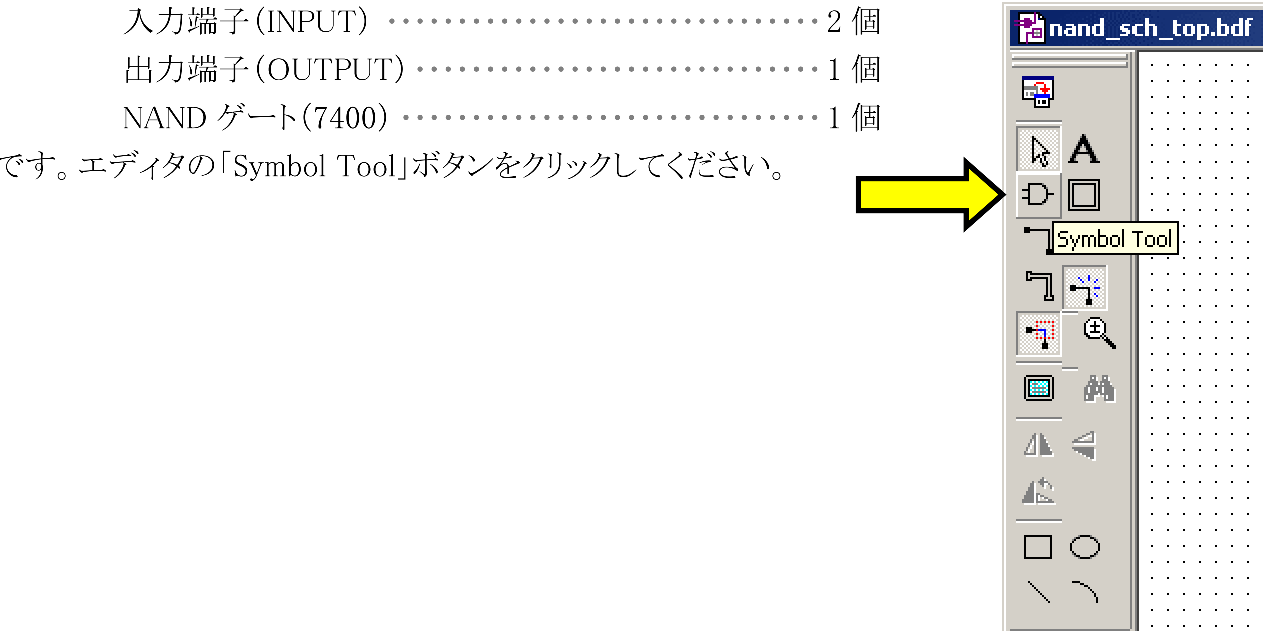


これで準備が整いました。Quartus II は次のような画面になっています。

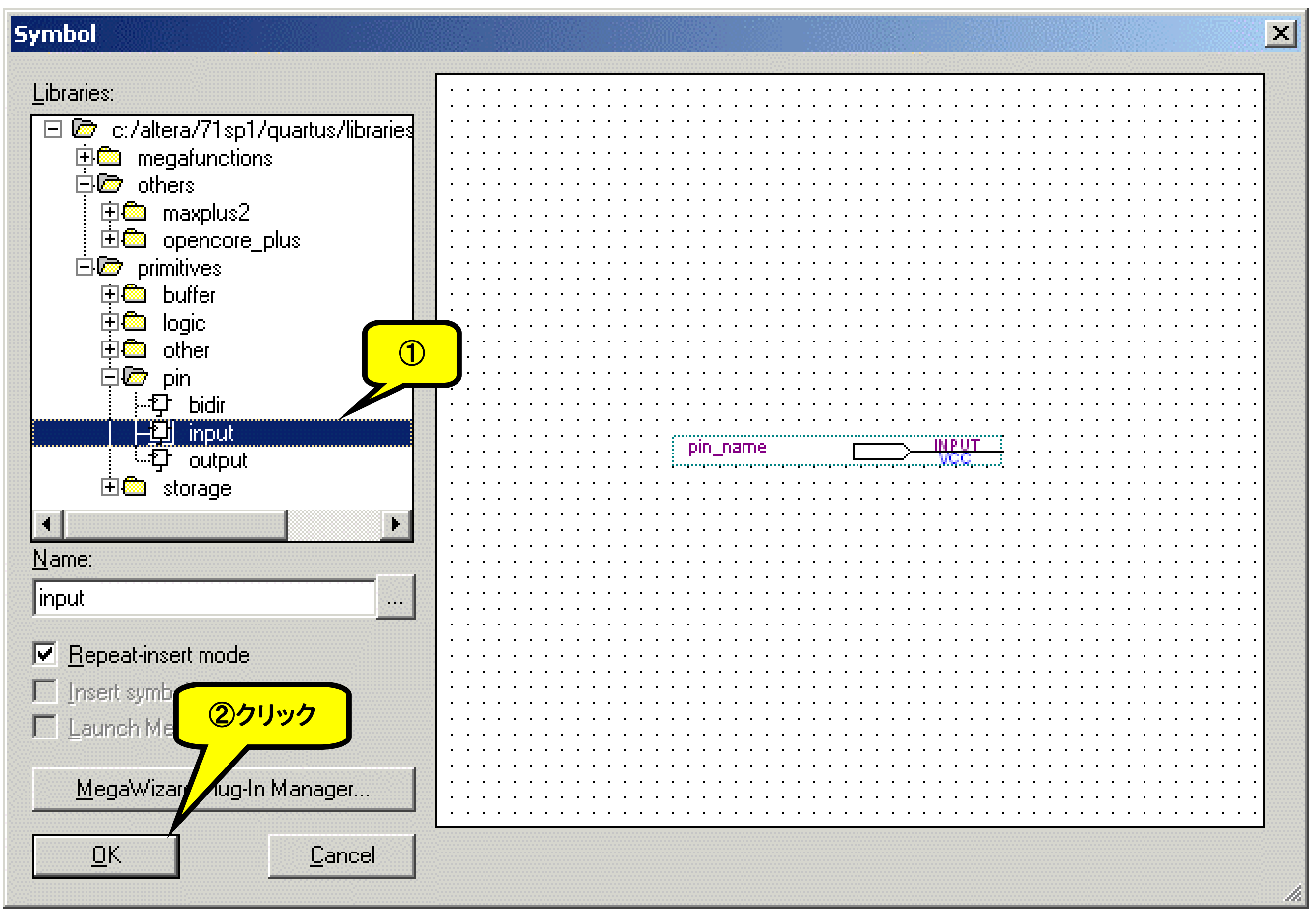
最初に部品をエディタ上に配置します。NAND ゲートは当然ですが，入出力ピンも部品に含まれます。それで，今回必要な部品は，

入力端子（INPUT）・・・・・・・・・・・・・・・・・・・・・・・・・・・・・・2 個
出力端子（OUTPUT）・・・・・・・・・・・・・・・・・・・・・・・・・・・・1 個
NAND ゲート（7400）・・・・・・・・・・・・・・・・・・・・・・・・・・・1 個

です。エディタの「Symbol Tool」ボタンをクリックしてください。



「Symbol」ダイアログが開きます。①左の「Libraries」の中から「input」を選択して下さい。②「OK」をクリックします。



マウスカーソルに INPUT 記号がくっついた状態になります。エディタ上の部品を起きたい場所で左クリックします。INPUT は 2 個使うので，もう 1 個起きたい場所で左クリックします。配置できたら ESC キーを押してカーソルを元の状態に戻します。

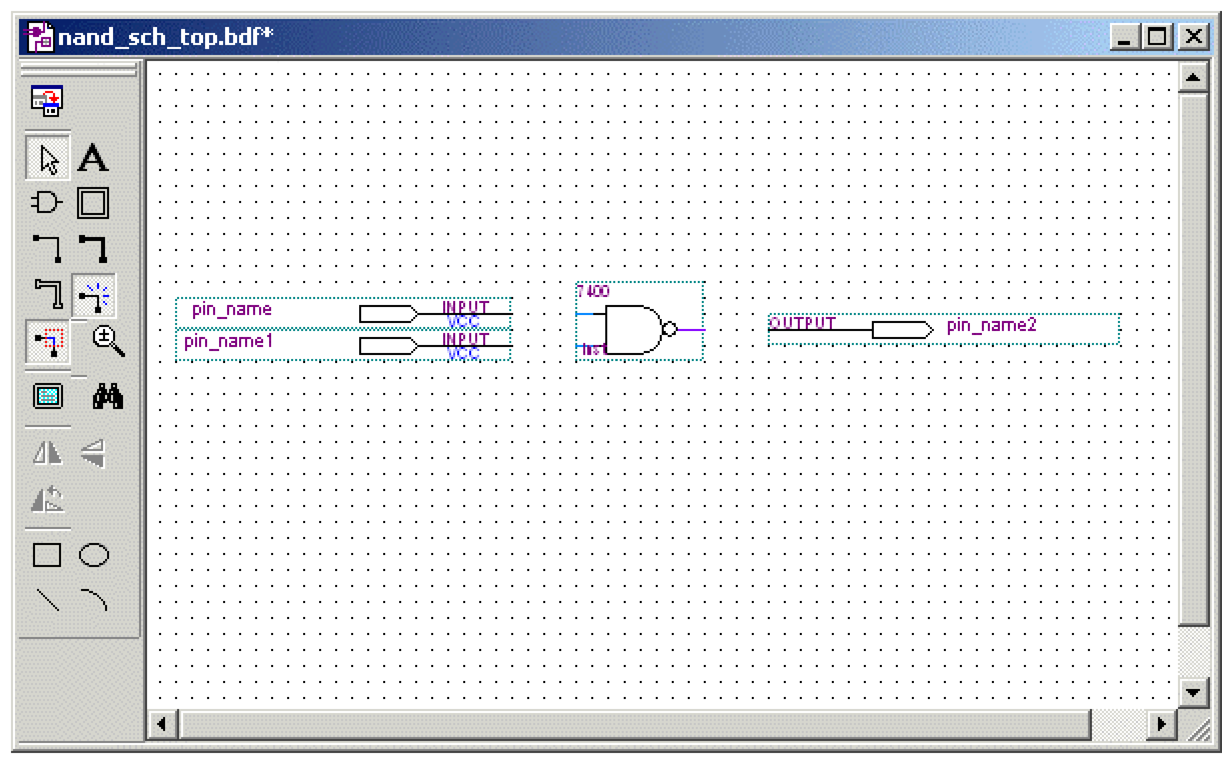
同じようにして OUTPUT を 1 個エディタ上に配置してください。

最後に NAND ゲートを配置します。NAND ゲートは 7400 というロジック IC を使います。このロジック IC の機能が Quartus II の部品として用意されています。「Symbol」ダイアログを開いて下さい。ロジック IC の部品は「Libraries」の中の「maxplus2」の中にあります。①左の「Libraries」の中から「7400」を選択して下さい。②「OK」をクリックします。

Symbol

Libraries:
c:/altera/71sp1/quartus/libra
megafunctions
others
maxplus2
161mux
16cudslr
16cudsrb
16dmux
16ndmux
1a2nor2
21mux
2a2nor2
2or2na2
2x8mux
4a2nor4
4count
7400
7402
①
Name:
7400
Repeat-insert mode
②クリック
MegaWiz Plug-In Manager...
OK
Cancel
7400

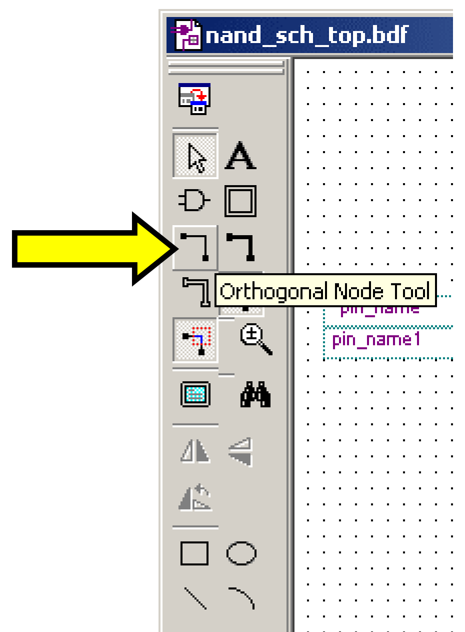
NAND ゲートを起きたい場所に配置します。最終的に次のようになります。

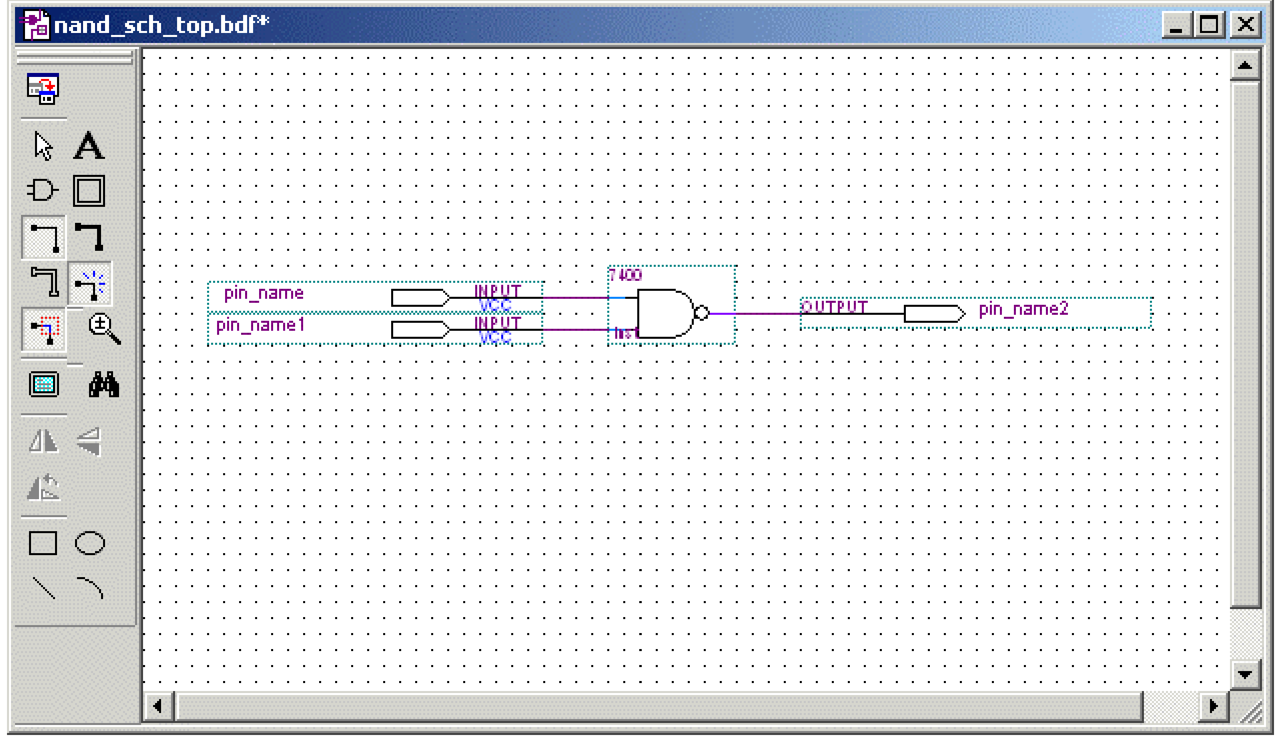


続いて，部品どうしを接続します。エディタの「Orthogonal Node Tool」ボタンをクリックして下さい。

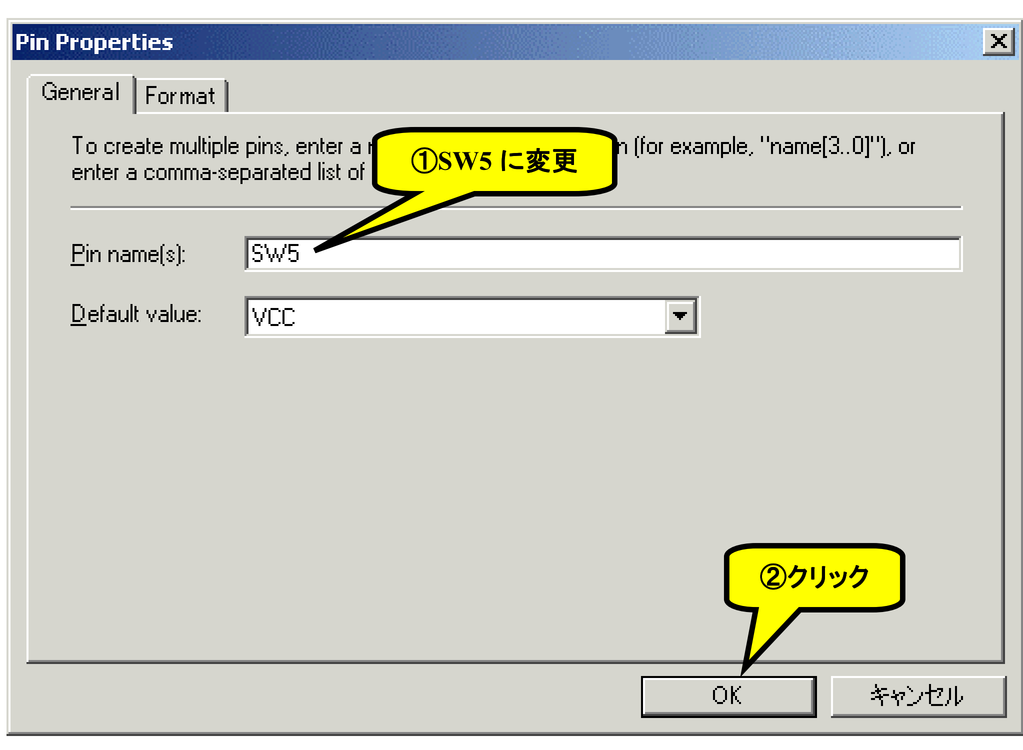
次にマウスカーソルを部品の端子のところで左クリックし，そのままドラッグして，相手の部品の端子のところで左ボタンを離します。そうすると，ドラッグの開始点と終了点が線で結ばれます。

この回路では3本の線を配置します。全てつなぐと次のようになります。



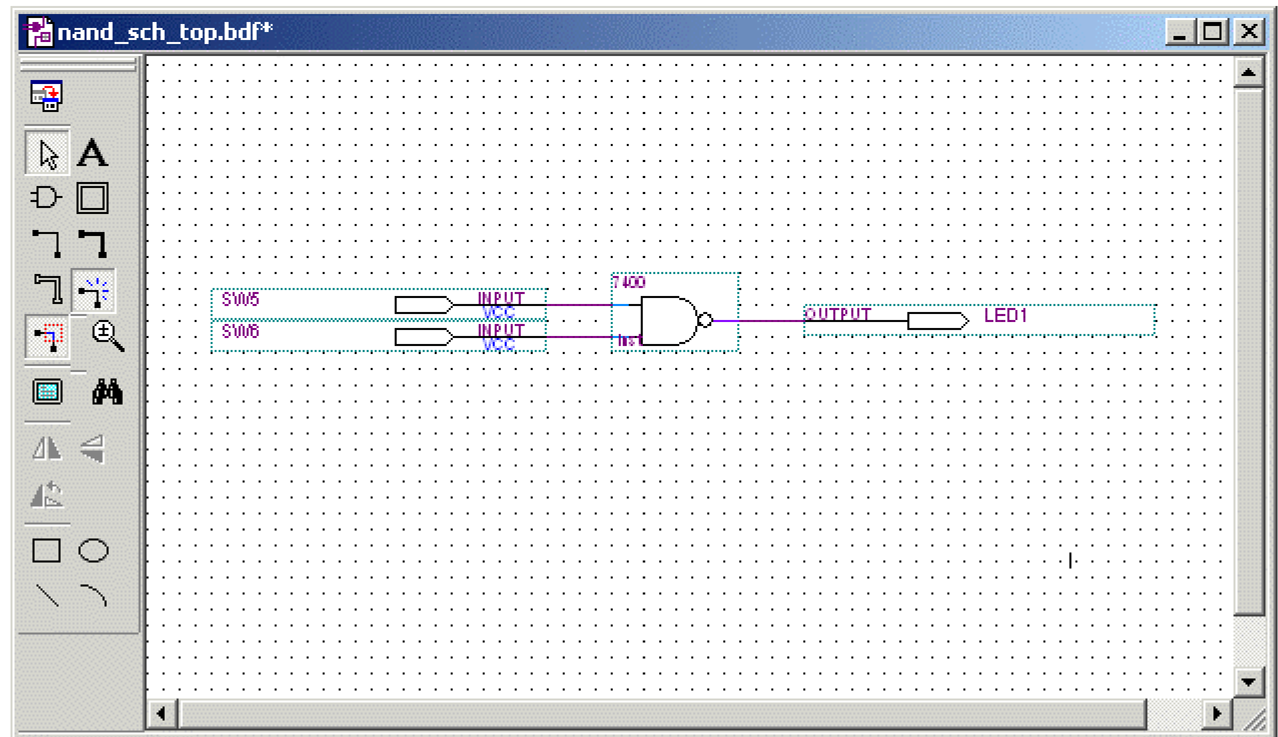


「INPUT」シンボルや「OUTPUT」シンボルはエンティティの外側の信号と接続するための部品です。Quartus II は「pin_name」という名前を付けますが，設計者がその端子にふさわしい名前を付けたほうがわかりやすいです。



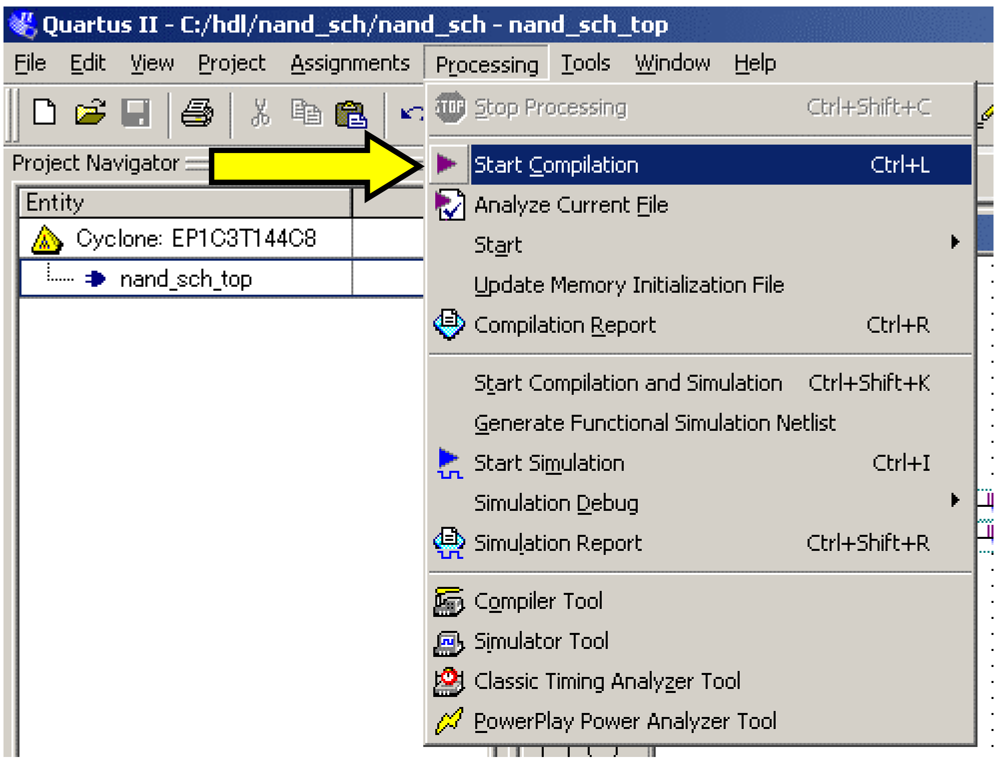
ESC キーを押してマウスカーソルを通常の状態に戻したあと，INPUT シンボルをダブルクックしてください。INPUT シンボルのプロパティのダイアログが開きます。ここで端子の名前を変更します。

同じようにしてもう一つの INPUT シンボルと OUTPUT シンボルの名前も変更しましょう。最終的に次のようになります。

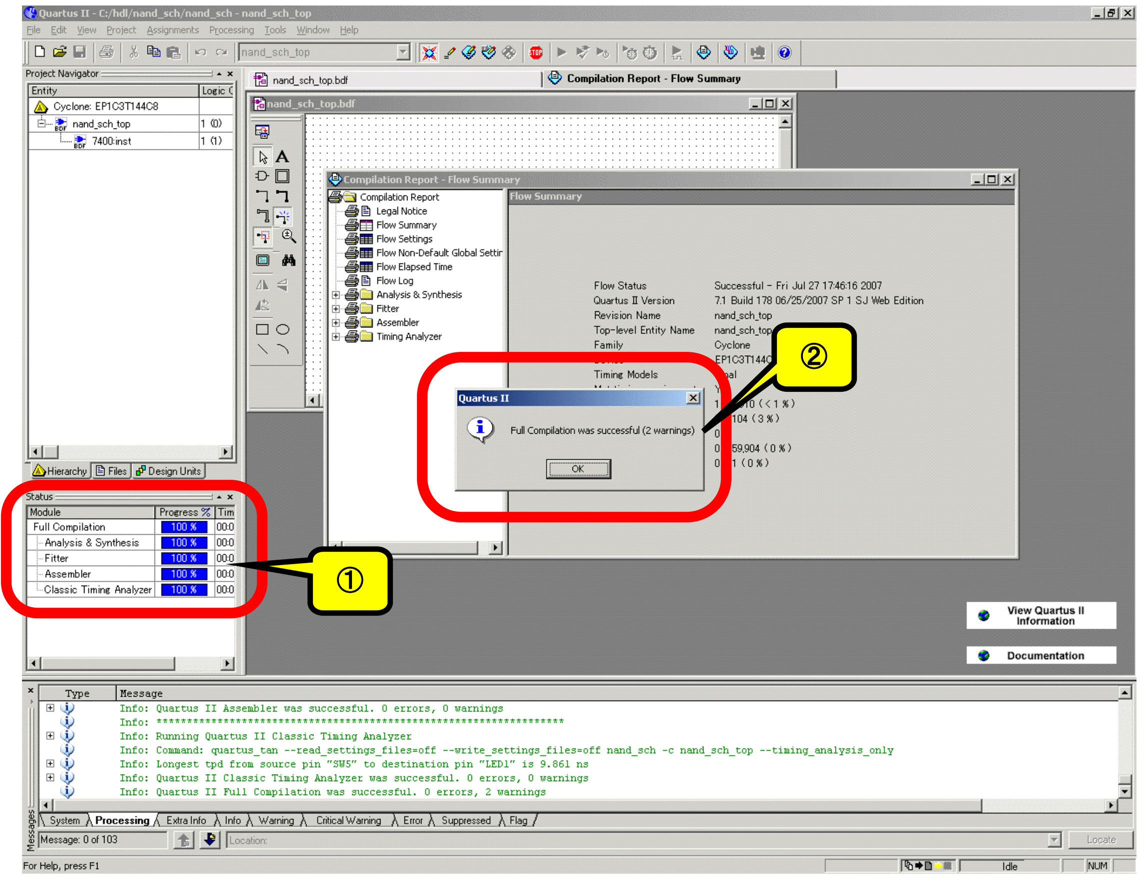


回路図の入力は終了しました。一旦 Quartus II のメニューから[File]→[Save]をクリックして保存しておきましょう。

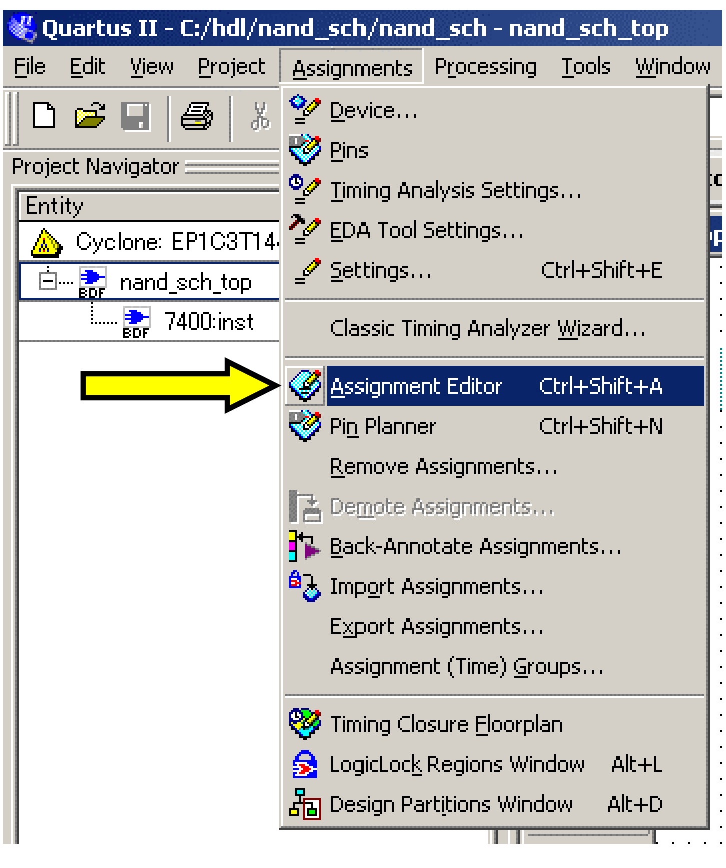
回路の入力が終了しました。次に，Cyclone に書き込むデータを作るためにコンパイルします。Quartus II のメニューから[Processing]→[Start Compilation]をクリックしてください。コンパイルが始まります。

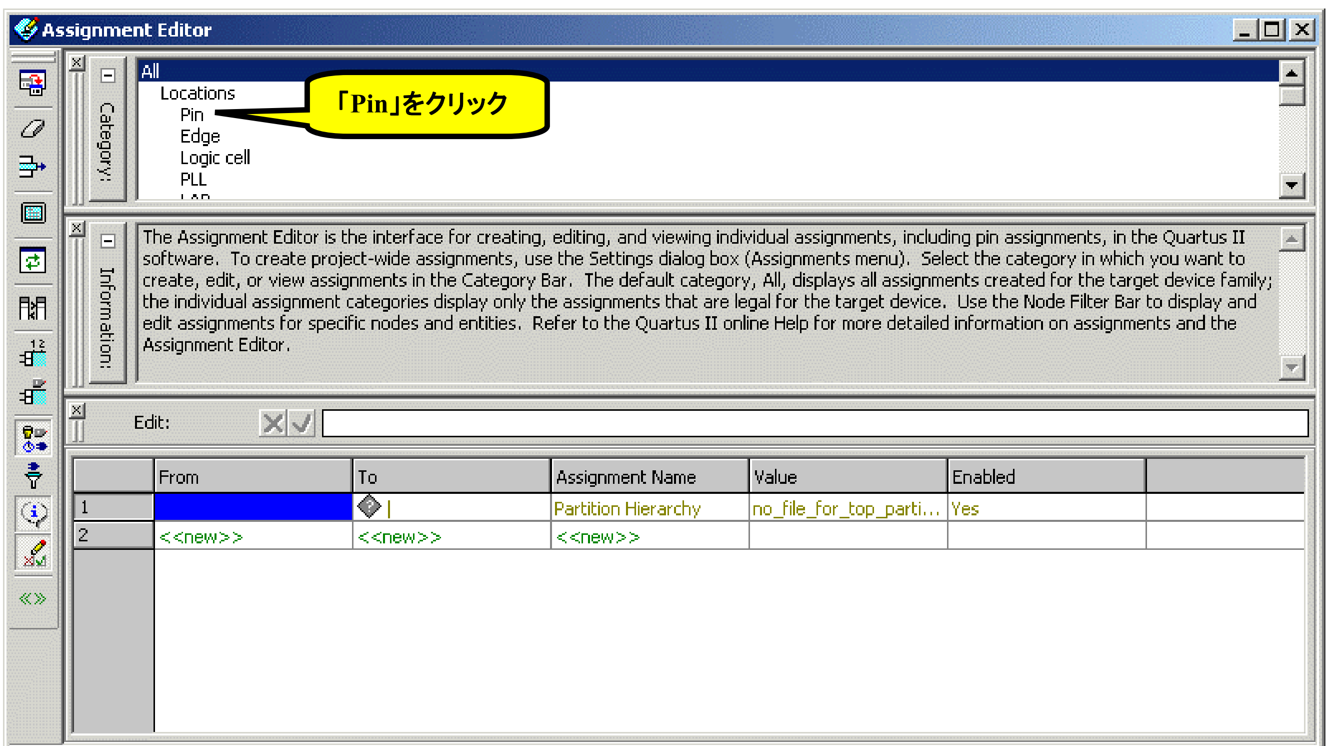


正常終了すると，①プログレスバーが全て 100%になり，②「successful」のダイアログが表示されます（warnings はそれほど気にしなくて大丈夫）。

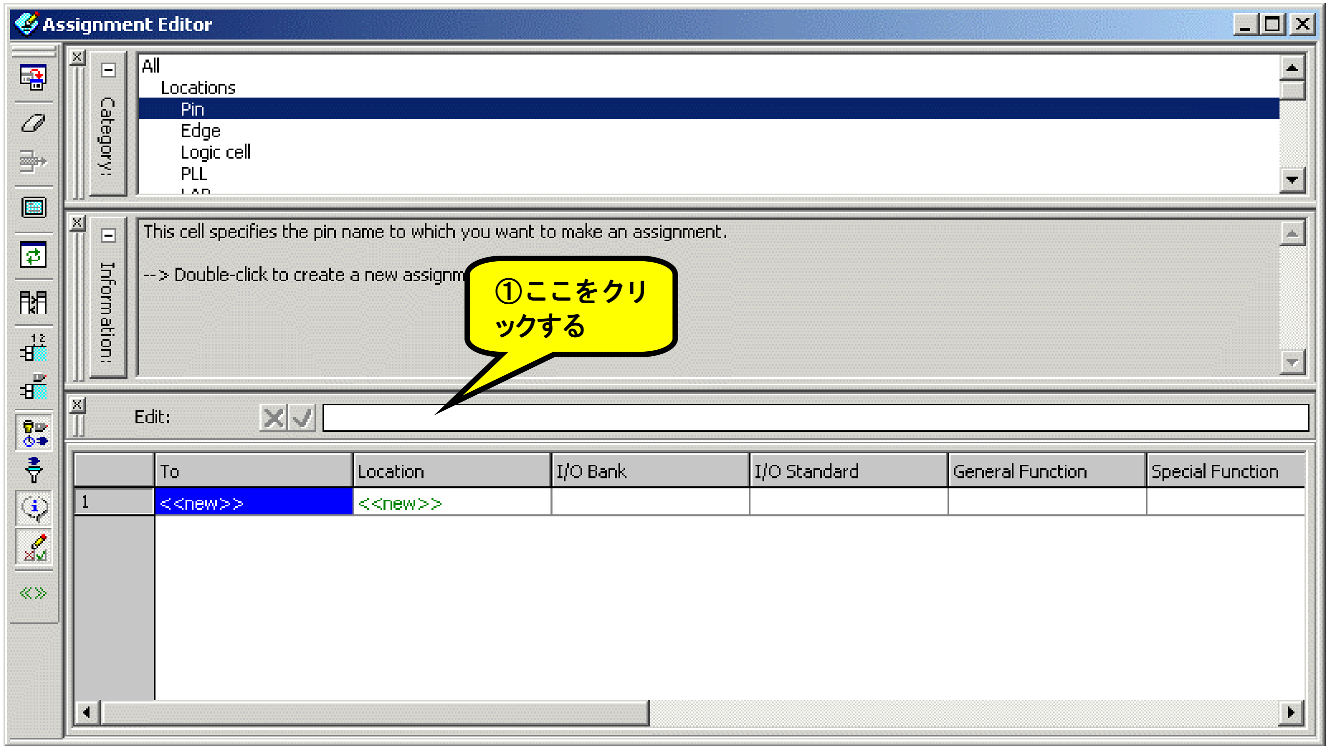


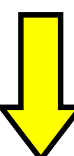
コンパイルも無事終わったので書き込みたいのですが，ここでもう一つ作業が残っています。それは，最上位階層(nand_sch_top)の入力端子と出力端子をCyclone のどの番号のピンに接続するか指定する，というものです。Quartus II のメニューから［Assignments］→［Assignment Editor］をクリックしてください。すると，Assignment Editor が起動します。

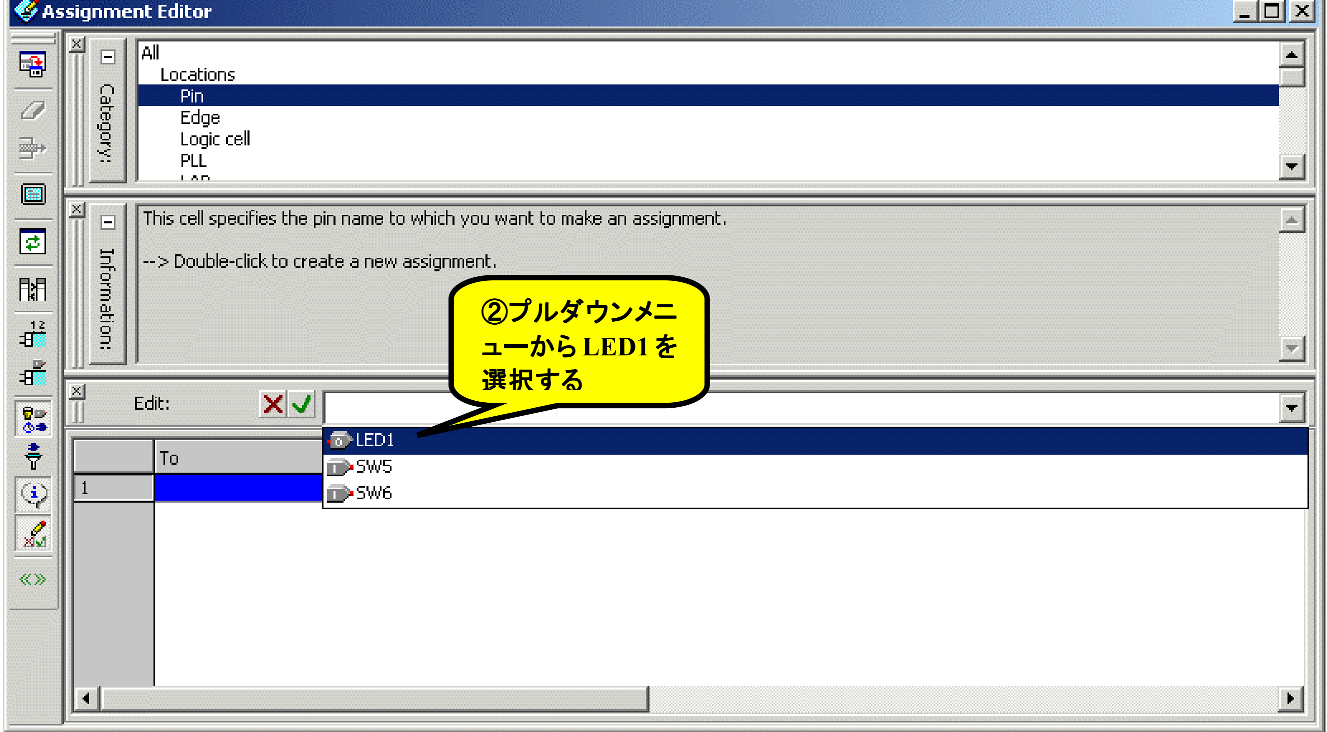


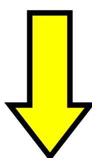


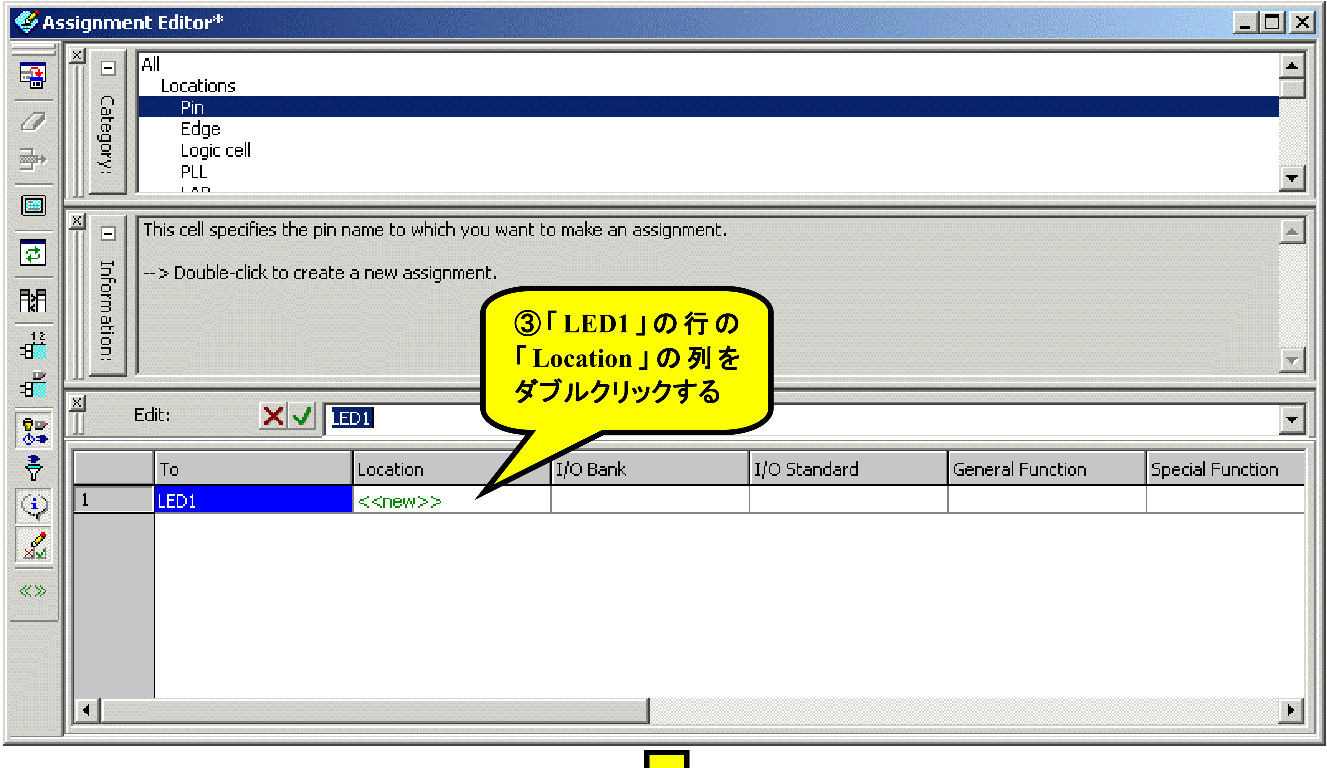
ここで設定したいのはピン番号なので，「Pin」をクリックします。すると，次のような画面になります。この画面でピン番号を設定します。


Assignment Editor
Category:
All
Locations
Pin
Edge
Logic cell
PLL
Information:
This cell specifies the pin name to which you want to make an assignment.
--> Double-click to create a new assignm
①ここをクリックする
Edit:
To
Location
I/O Bank
I/O Standard
General Function
Special Function
1
<<new>>
<<new>>

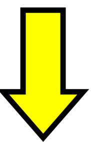


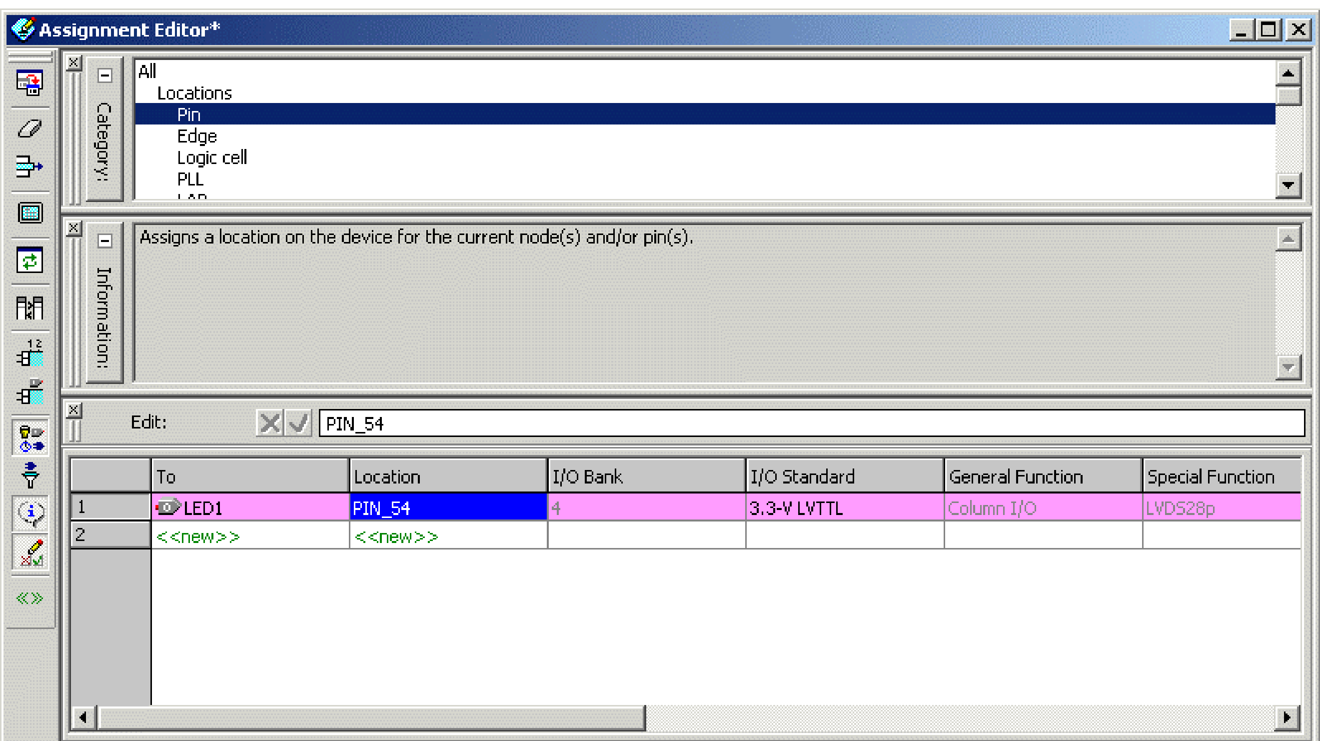

Assignment Editor
Category:
All
Locations
Pin
Edge
Logic cell
PLL
Information:
This cell specifies the pin name to which you want to make an assignment.
--> Double-click to create a new assignment.
②プルダウンメニューからLED1を選択する
Edit:
To
1
LED1
SW5
SW6



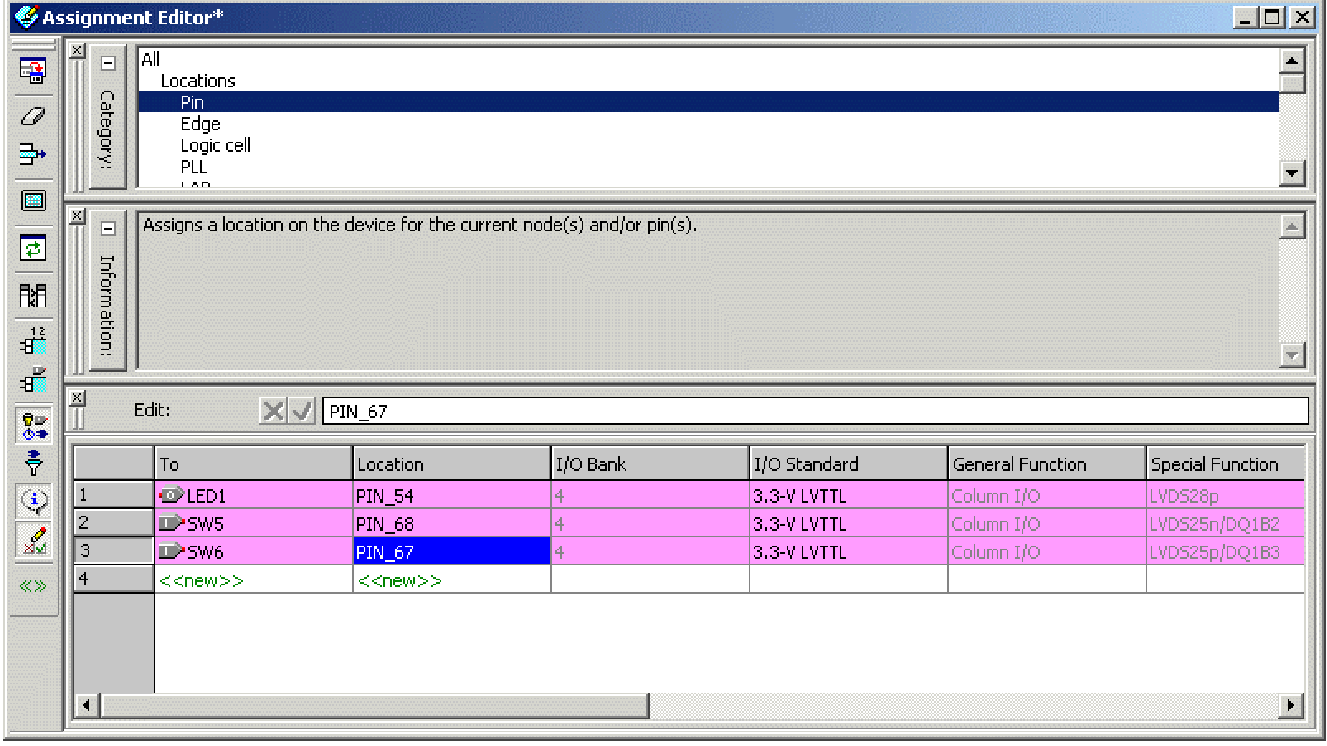


PIN_36	I/O Bank 1	Row I/O	LVDS0n
PIN_37	I/O Bank 4	Column I/O	LVDS33p
PIN_38	I/O Bank 4	Column I/O	LVDS33n
PIN_39	I/O Bank 4	Column I/O	LVDS32p/DQ1B7
PIN_40	I/O Bank 4	Column I/O	LVDS32n/DQ1B6
PIN_41	I/O Bank 4	Column I/O	LVDS31p/DQ1B5
PIN_42		...n I/O	LVDS31n/DQ1B4
PIN_47		...n I/O	DPCLK7/DQS1B
PIN_48		...n I/O	VREF2B4
PIN_49		...n I/O	
PIN_50	I/O Bank 4	Column I/O	LVDS30p
PIN_51	I/O Bank 4	Column I/O	LVDS30n
PIN_52	I/O Bank 4	Column I/O	LVDS29p
PIN_53	I/O Bank 4	Column I/O	LVDS29n
PIN_54	I/O Bank 4	Column I/O	LVDS28p
PIN_55	I/O Bank 4	Column I/O	LVDS28n
PIN_56	I/O Bank 4	Column I/O	VREF1B4
PIN_57	I/O Bank 4	Column I/O	LVDS27p/DM1B
PIN_58	I/O Bank 4	Column I/O	LVDS27n
PIN_59	I/O Bank 4	Column I/O	LVDS26p

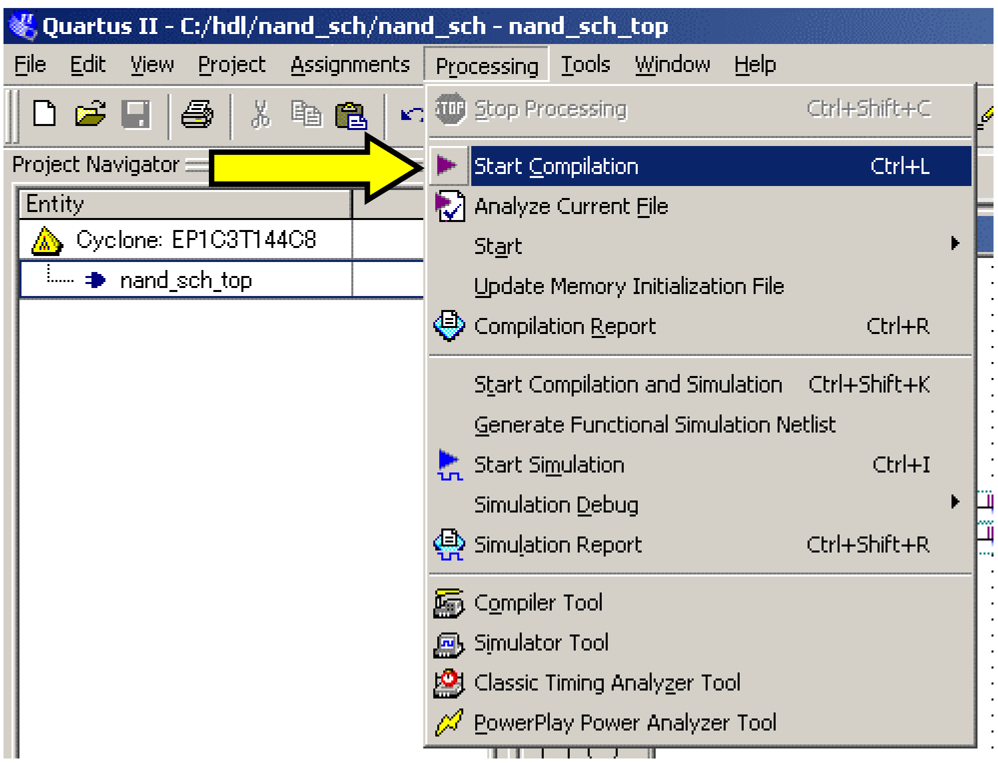




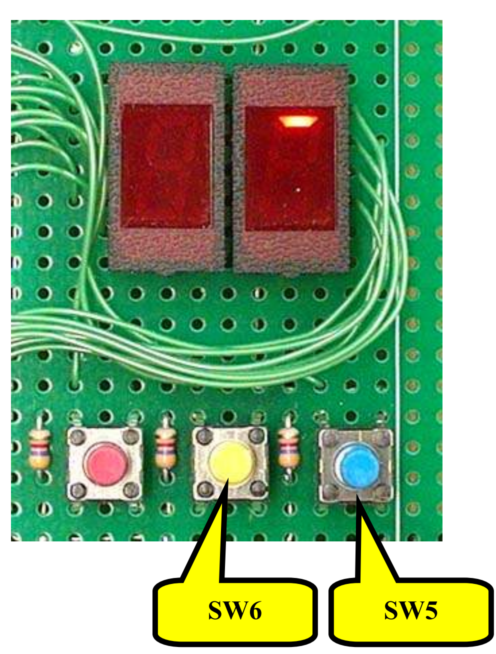
これで,「LED1」の端子を 52 番ピンに割り当てました。あとの欄は特に指定する必要はありません。「I/O Standard」の列が「3.3V LVTTL」になっていることを確認して下さい。同じようにして,「SW5」と「SW6」の端子を 68 番ピンと 67 番ピンに割り当ててください。次のようになります。



ここでもう一度コンパイルします。Quartus II のメニューから[Processing]→[Start Compilation]をクリックしてください。コンパイルが始まります。



正常に終了したら，第 4 章と同じ手順で Cyclone に書き込んでください。JTAG で書き込む時に選択するファイルは「nand_sch_top. Sof」です。SW5 か SW6 をオンすると LED が点灯します。



■ 練習問題

NAND で説明してきました。では，AND，OR，NOT，NOR，ExOR も同じように回路図入力で FPGA に書き込んでみてください。(解答例は付属の CD をご覧ください，「and_sch」，「or_sch」，「not_sch」，「nor_sch」，「exor_sch」)

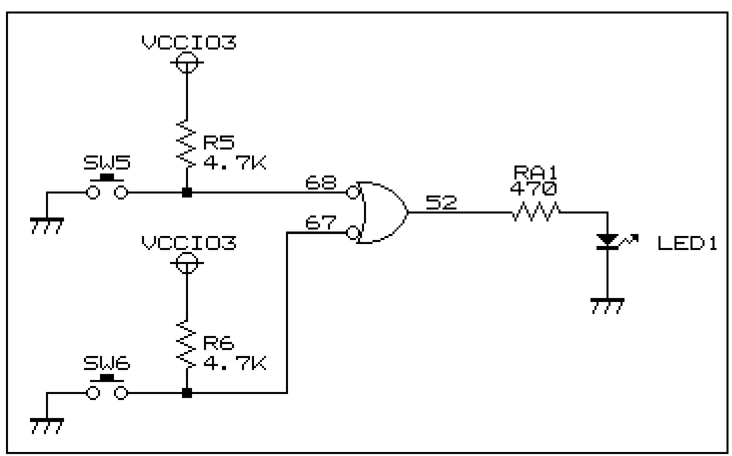
2. 基本ゲートの VHDL 入力

前項では回路図入力を使ってみました。回路図を書くだけで回路ができるなんて感動ものですね。きっと，いろいろと試してみたくなったことでしょう。

ところで試していくうちに，段々と面倒くさくなってきませんでしたか。部品を選んではエディタに配置し，線を選んでは部品と部品をつなぐ繰り返し。回路が小さい内はよくても，大きな回路を入力するとなると効率が悪くて仕方ありません。もっと簡単な方法はないのか，と言いたくなります。

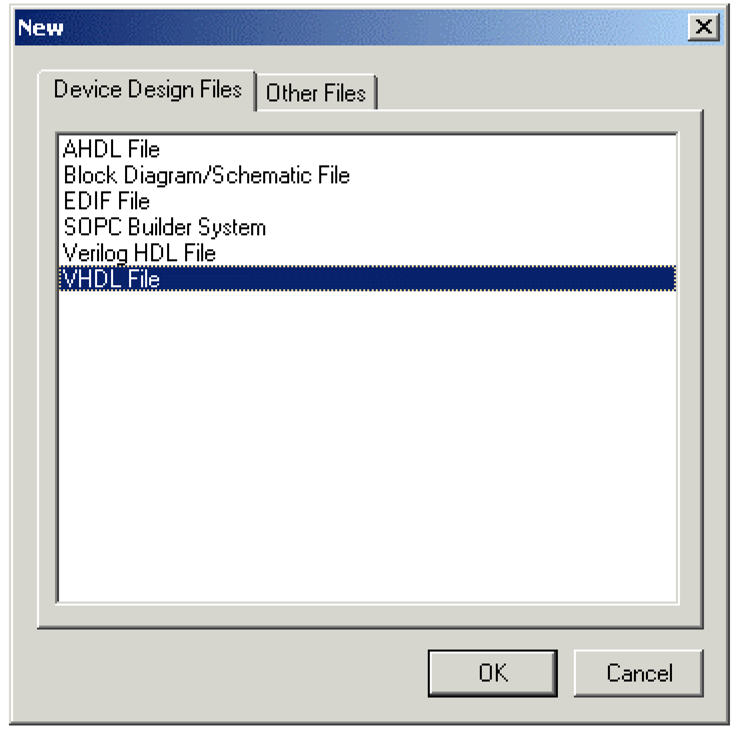
プログラムを書くように回路を書く方法があります。それが，ハードウェア記述言語（HDL）を使う方法です。これだとキーボードで回路設計ができてしまいます。回路図入力に比べると最初のとっつきは悪いのですが，一度覚えてしまうと簡単かつ速く，複雑な回路を記述することができます。

このマニュアルでは HDL の中でよく使われているものの一つ，「VHDL」を使って回路を書きます。やはり最初は最も簡単な回路から始めましょう。回路図は前項と同じ，次のとおりです。

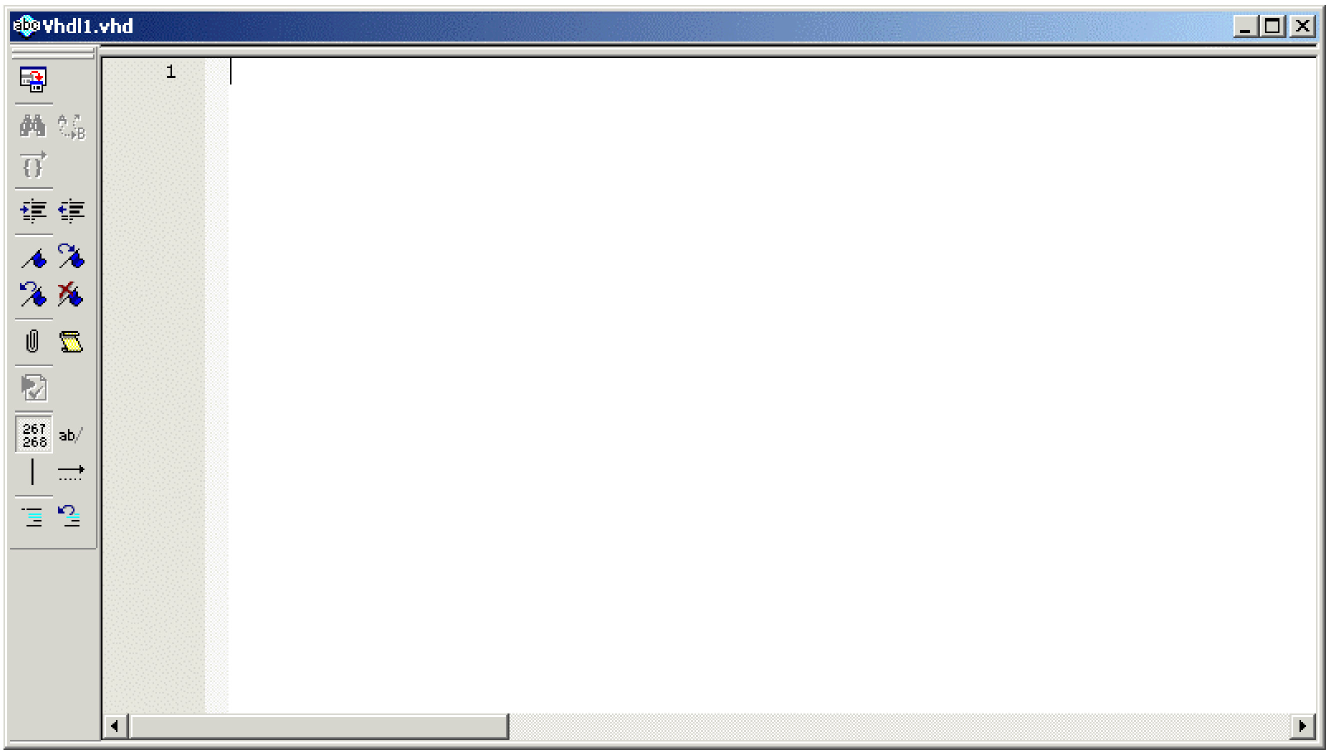


では，Quartus II を起動し，新しいプロジェクトを作成してください（ここまでは回路図入力と同じです）。プロジェクトを保存するフォルダは「nand_vhdl」，プロジェクト名は「nand_vhdl」，回路の最上位の階層のエンティティ名は「nand_vhdl_top」にします。

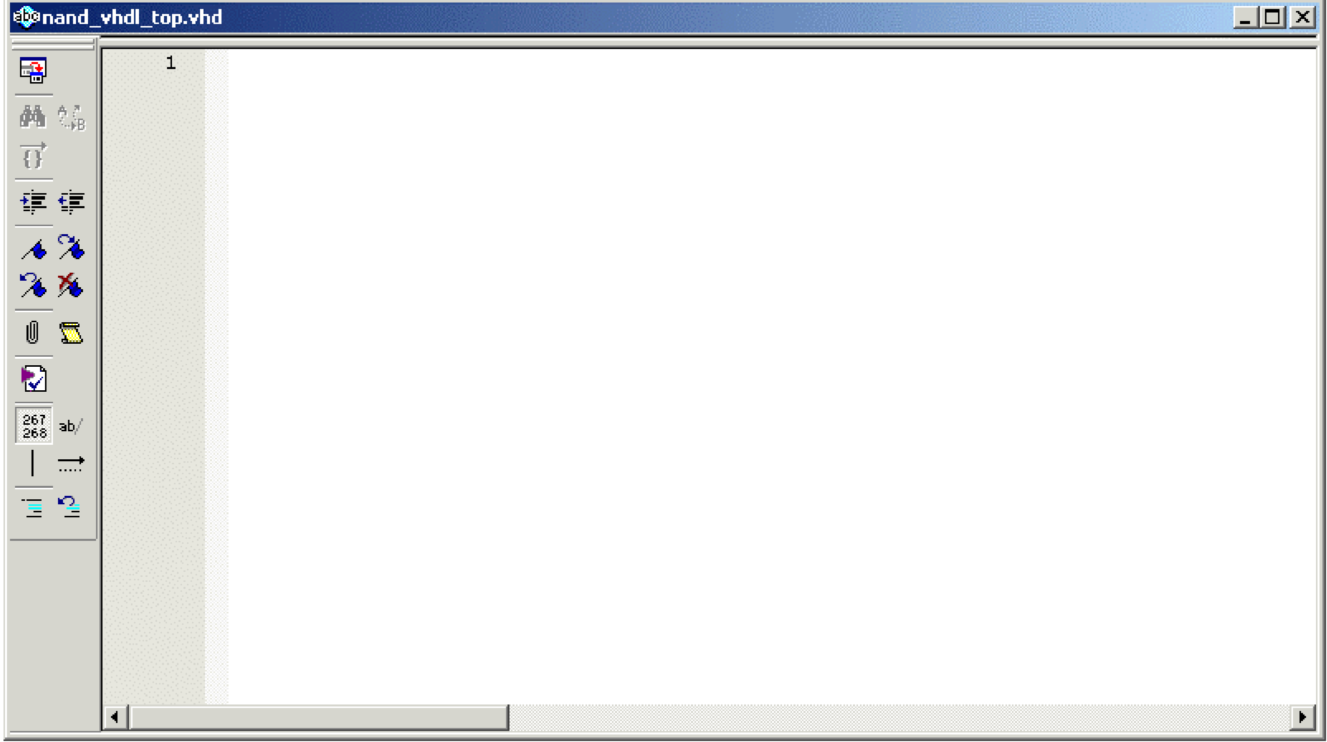
では，回路を入力しましょう。前項と同じく回路が単純なので，最上位階層である「nand_vhdl_top」エンティティに入力します。Quartus II のメニューから[File]→[New]をクリックします。そうすると右のダイアログが表示されます。いくつかある選択肢の中から「VHDL File」を選び，「OK」をクリックします。



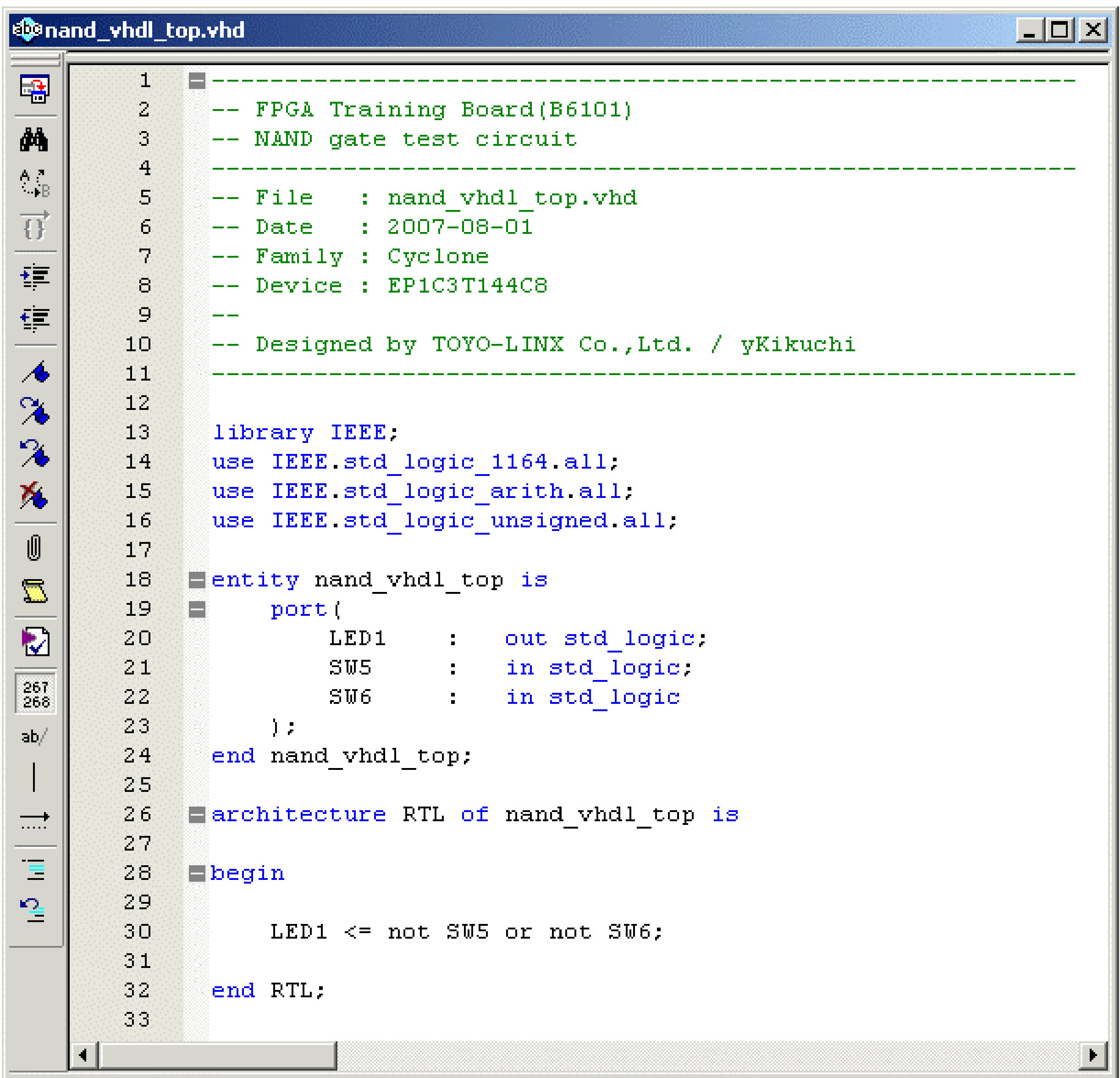
　すると「Vhdl1. Vhd」という名前のウィンドウが開きます。これがエディタになります。



　ただ，この「Vhdl1. Vhd」という名前は Qurtus II が勝手に付けたものなので，最上位階層のエンティティ名に変更してセーブしておきましょう。Quartus II のメニューから[File]→[Save As]をクリックすると「Save As」ダイアログが開きますので，ファイル名に「nand_vhdl_top」と入力して「保存」をクリックします。



では，エディタに次のリストを入力して下さい。




```
--------------------------------------------------------------
-- FPGA Training Board(B6101)
-- NAND gate test circuit
--------------------------------------------------------------
-- File    : nand_vhdl_top.vhd
-- Date    : 2007-08-01
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
--------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity nand_vhdl_top is
    port(
        LED1    :   out std_logic;
        SW5     :   in std_logic;
        SW6     :   in std_logic
    );
end nand_vhdl_top;

architecture RTL of nand_vhdl_top is

begin

    LED1 <= not SW5 or not SW6;

end RTL;

```

簡単にリストの説明をしましょう。

■ 1～11 行

二つのハイフン「--」を書くと，それ以降から文の最後まではコメントとして扱われます。行の途中から記述することもできます。この部分は回路としては扱われませんので，あとからソースリストを読んだときにわかりやすくするために利用します。コード内のどんな場所にでも記述することもできます。

■ 13～16 行

ライブラリとパッケージの使用宣言部です。VHDL のコード内で使う単語（AND，OR，std_logic など）を使用できるようにするために，あらかじめ宣言しなければなりません。とりあえず難しいことは考えずに，13～16 行については「おまじない」として必ず書かなければいけない，と覚えておいてかまいません。

■ 18～24 行

エンティティ宣言部です。ここには，回路の入出力端子の情報を書きます。18 行はエンティティ名を定義しているところで，この回路のエンティティ名は「nand_vhdl_top」になります。19 行からは port 文で，入出力端子の信号線を定義します。例えば 20 行は「信号線名：LED1，方向：出力，データタイプ：std_logic」となります。VHDL には多くのデータタイプがありますが，回路設計のほとんどは std_logic 型を使います。24 行はエンティティ宣言部の終了を定義しています。

まとめると，「nand_vhdl_top」エンティティには「LED1」という出力端子と「SW5」・「SW6」という入力端子がある，ということになります。

■ 26～32 行

アーキテクチャ宣言部です。エンティティの内部がどんな回路になるかを指定します。26 行は「nand_vhdl_top」エンティティの「rtl」アーキテクチャがここから始まることを示しています。アーキテクチャ名は自由に付けられるのですが，VHDL の場合，慣例的に「rtl」が使われています。

このコードでは定義されていませんが，26 行と 28 行の間で内部信号線を定義します。

28 行以降が回路の定義です。「begin」から「end」までを同時処理文と呼び，ここに書かれている全ての文が同時に処理されます。32 行はアーキテクチャ宣言部の終了を定義しています。

これで，コードの入力は終了しました。一旦 Quartus II のメニューから[File]→[Save]をクリックして保存しておきましょう。

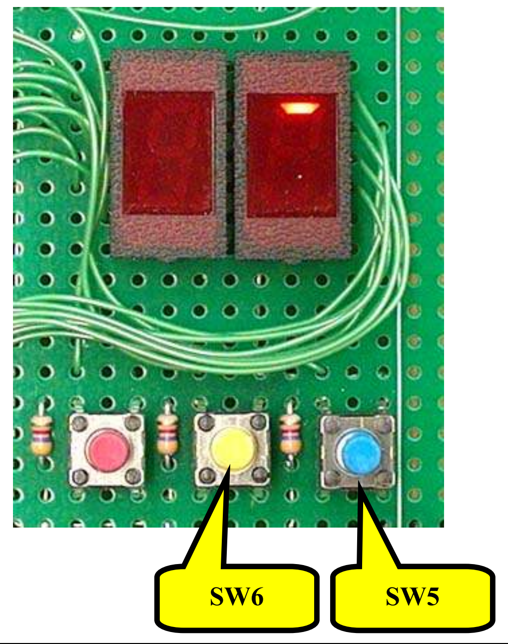
あとは回路図入力のときと同じです。

Cyclone に書き込むデータを作るためにコンパイルします。Quartus II のメニューから[Processing]→[Start Compilation]をクリックしてください。コンパイルが始まります。正常終了すると，プログレスバーが全て 100％になり，「successful」のダイアログが表示されます（warnings はそれほど気にしなくて大丈夫）。

次に最上位階層（nand_sch_top）の入力端子と出力端子を Cyclone のどの番号のピンに接続するか指定します。Quartus II のメニューから[Assignments]→[Assignment Editor]をクリックしてください。すると，Assignment Editor が起動します。回路図入力のときと同じようにして指定して下さい。

ここでもう一度コンパイルします。Quartus II のメニューから[Processing]→[Start Compilation]をクリックしてください。コンパイルが始まります。

正常に終了したら，Cyclone に書き込んでください。JTAG で書き込む時に選択するファイルは「nand_vhdl_top.Sof」です。SW5 か SW6 をオンすると LED が点灯します。



ところで，30 行は次のようなものでした。

```
LED1 <= not SW5 or not SW6;
```

これは単純に回路記号をVHDL で記述したものです。言葉で書けば「SW5 が 0 のとき，あるいは，SW6 が 0 のとき，LED1 に 1 を出力する」となります。

さて，同じ回路記号を正論理で書くと NAND になります。30 行のコードを，NAND を使って記述すると，

```
LED1 <= SW5 nand SW6;
```

となります。動かしてみるとわかりますが，全く同じ動作をします。ただし，設計者の考えは，「SW5 が 0 のとき，あるいは，SW6 が 0 のとき，LED1 に 1 を出力する」というものであることは元の回路図から明らかです。それで，今回は NAND ではなく OR を使ったコードの方がより良いコードといえるでしょう。

■ 練習問題

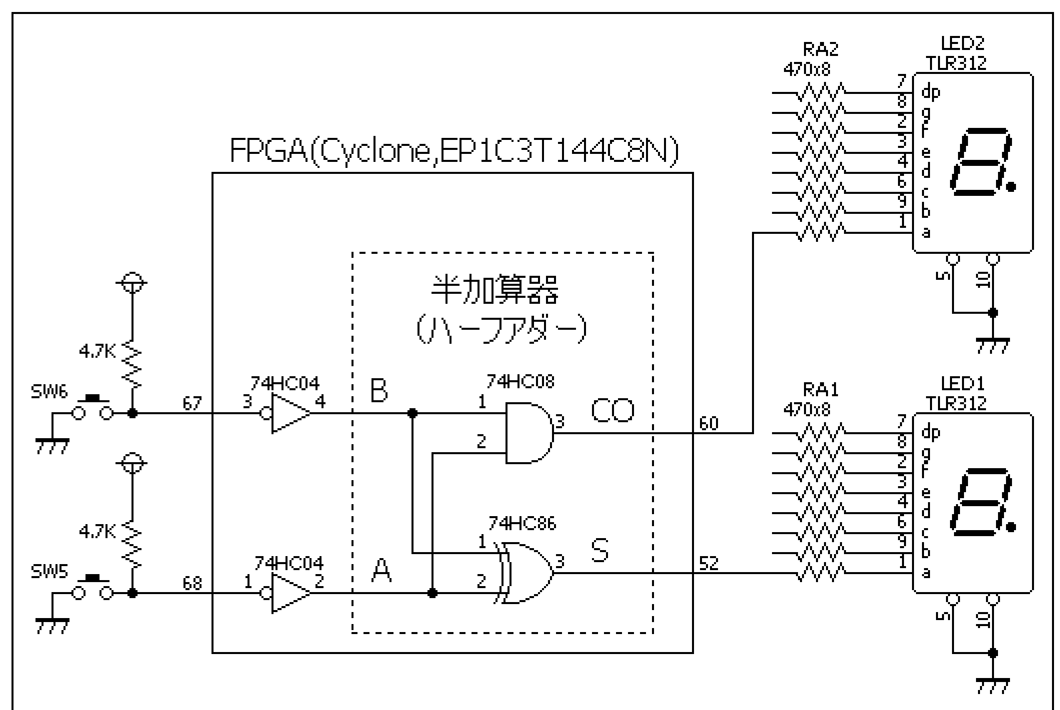
AND，OR，NOT，NOR，ExORをVHDL 入力でFPGA に書き込んでみてください。(解答例は付属の CD をご覧ください，「and_vhdl」，「or_vhdl」，「not_vhdl」，「nor_vhdl」，「exor_vhdl」)

■ 練習問題

入力 A と B を加算した信号(S)を出力し，桁上がりがあればキャリー(CO)を出力する回路を作ってみましょう。真理値表と回路図は次のとおりです。A と B にはスイッチを反転した信号を入力します。回路図入力とVHDL 入力で考えてみてください。(解答例は付属の CD をご覧ください，VHDL 入力では ExOR は使っていません，「half_adder_sch」と「half_adder_vhdl」)

入力		出力	
A	B	S	CO
0	0	0	0
1	0	1	0
0	1	1	0
1	1	0	1

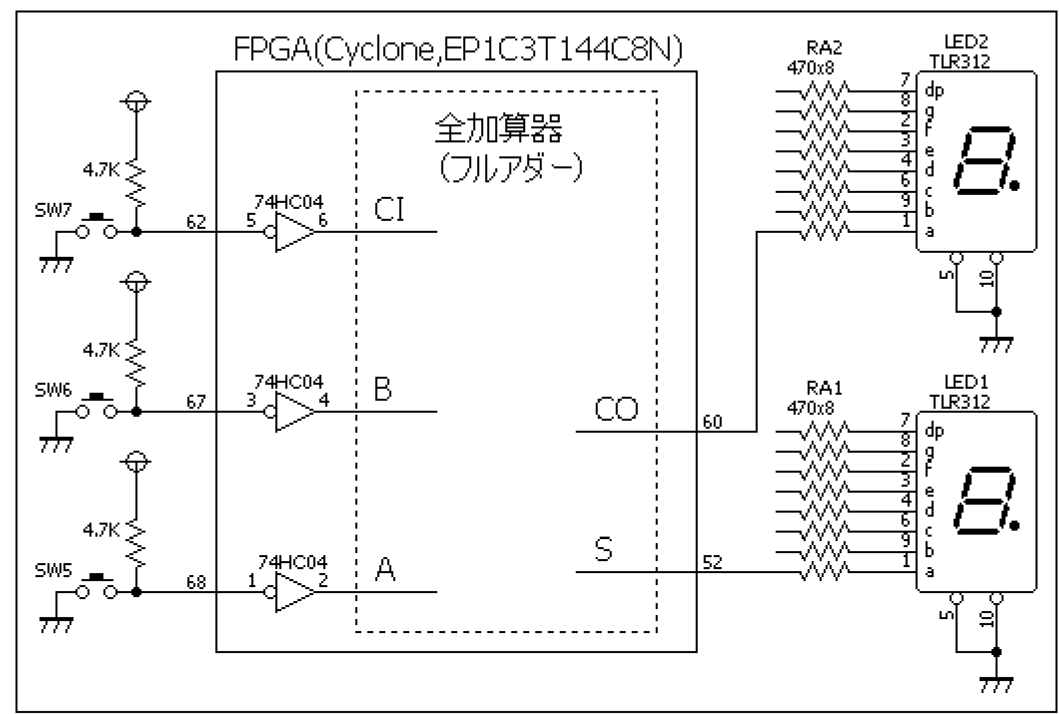
この回路は演算回路の基本です。桁上がり信号の出力はありますが，下位の桁からの桁上がり信号の入力がありません。そういう意味では完全な加算器にはなっていないので，半加算器(ハーフアダー)と呼ばれています。



■ **練習問題**

今度は下位の桁からの桁上がり信号（CI）の入力にも対応した，全加算器（フルアダー）を作ってみましょう。A と B と CI にはスイッチを反転した信号を入力します。下の回路図のうち，ブラックボックスになっている点線の枠内の回路を VHDL で考えてみてください。（解答例は付属の CD をご覧ください，「full_adder_vhdl」）

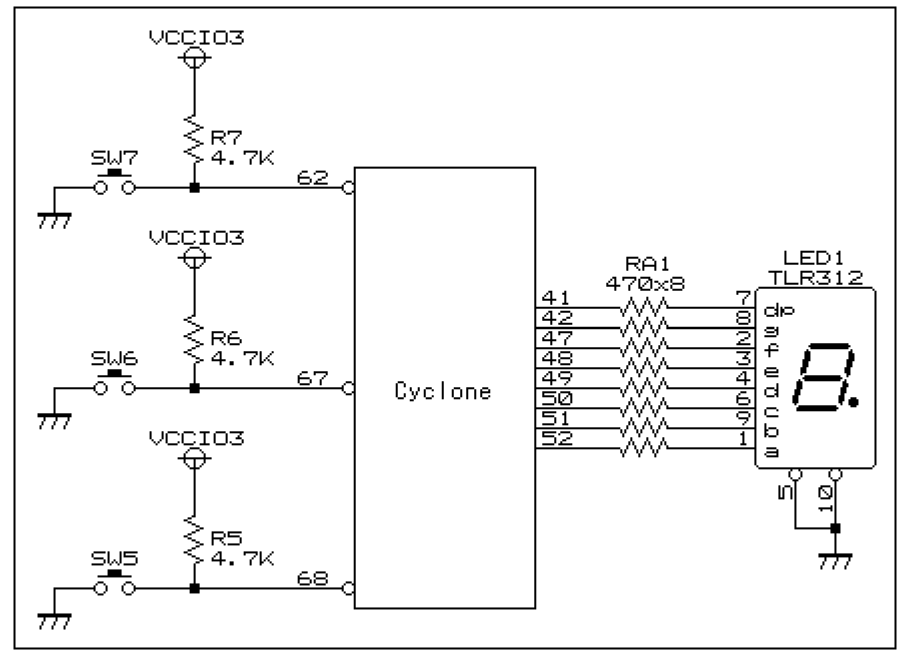
入力			出力	
CI	A	B	S	CO
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
1	1	0	0	1
0	0	1	1	0
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1



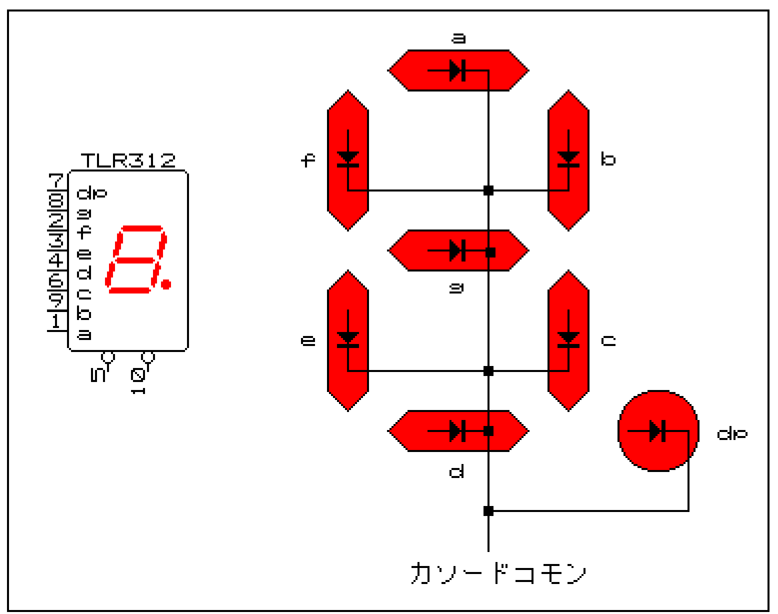
3. デコーダとエンコーダ

基本ゲートを組み合わせた回路で良く使われるのはデコーダとエンコーダです。はじめに，7 セグメント LED デコーダを作ってみましょう。74 シリーズには 7447 という 7 セグメント LED デコーダが用意されています。Quartus II の回路図入力でも使うことができます。でも，ここはあえて VHDL で書いてみます。

回路図は次のようになります。



まずは 7 セグメント LED の仕組みを見てみましょう。内部回路は次のようになっています。



それで，SW5～SW7 の押し方によって LED がつながっている端子に High を出力し，‘0’～‘7’を 7 セグメント LED に表示させます。

ではここで，真理値表を作成してみましょう。

スイッチの状態			数字	7 セグメント LED への出力							
SW7	SW6	SW5		a	b	c	d	e	f	g	dp
OFF	OFF	OFF	0	1	1	1	1	1	1	0	0
OFF	OFF	ON	1	0	1	1	0	0	0	0	0
OFF	ON	OFF	2	1	1	0	1	1	0	1	0
OFF	ON	ON	3	1	1	1	1	0	0	1	0
ON	OFF	OFF	4	0	1	1	0	0	1	1	0
ON	OFF	ON	5	1	0	1	1	0	1	1	0
ON	ON	OFF	6	1	0	1	1	1	1	1	0
ON	ON	ON	7	1	1	1	0	0	0	0	0

では，これをもとに VHDL で書いてみます。フォルダ名は「decode_7seg_vhdl」，プロジェクト名は「decode_7seg_vhdl」，最上位階層のエンティティ名は「decode_7seg_vhdl_top」とします。今までどおりプロジェクトを作成し，「decode_7seg_vhdl_top. Vhd」をエディタで開いて下さい。コードは次のとおりです。

```
----------------------------------------------------------
-- FPGA Training Board(B6101)
-- 7Segment Decode circuit
----------------------------------------------------------
-- File   : decode_7seg_vhdl_top.vhd
-- Date   : 2007-08-07
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity decode_7seg_vhdl_top is
    port(
        LED_A    :   out std_logic;
        LED_B    :   out std_logic;
        LED_C    :   out std_logic;
        LED_D    :   out std_logic;
        LED_E    :   out std_logic;
        LED_F    :   out std_logic;
        LED_G    :   out std_logic;
        LED_DP   :   out std_logic;
        SW5      :   in std_logic;
        SW6      :   in std_logic;
        SW7      :   in std_logic
    );
```

```
end decode_7seg_vhdl_top;

architecture RTL of decode_7seg_vhdl_top is

    signal number :    std_logic_vector(2 downto 0);

begin

    number(2) <= not SW7;
    number(1) <= not SW6;
    number(0) <= not SW5;

    LED_A <= ‘1’ when number = 0 else
             ‘0’ when number = 1 else
             ‘1’ when number = 2 else
             ‘1’ when number = 3 else
             ‘0’ when number = 4 else
             ‘1’ when number = 5 else
             ‘1’ when number = 6 else
             ‘1’ when number = 7 else
             ‘0’;

    LED_B <= ‘1’ when number = 0 else
             ‘1’ when number = 1 else
             ‘1’ when number = 2 else
             ‘1’ when number = 3 else
             ‘1’ when number = 4 else
             ‘0’ when number = 5 else
             ‘0’ when number = 6 else
             ‘1’ when number = 7 else
             ‘0’;

    LED_C <= ‘1’ when number = 0 else
             ‘1’ when number = 1 else
             ‘0’ when number = 2 else
             ‘1’ when number = 3 else
             ‘1’ when number = 4 else
             ‘1’ when number = 5 else
             ‘1’ when number = 6 else
             ‘1’ when number = 7 else
             ‘0’;

    LED_D <= ‘1’ when number = 0 else
             ‘0’ when number = 1 else
             ‘1’ when number = 2 else
             ‘1’ when number = 3 else
             ‘0’ when number = 4 else
             ‘1’ when number = 5 else
             ‘1’ when number = 6 else
             ‘0’ when number = 7 else
             ‘0’;

    LED_E <= ‘1’ when number = 0 else
```

SW5, SW6, SW7 という信号を number という一つの信号名称で扱うために，ベクタタイプの内部信号を定義する。

SW5, SW6, SW7 は負論理なので，正論理に直した上で，内部信号 number に接続する。

A <= B when ＜条件＞ else C;
条件が成立するときは信号 A に B を出力し，成立しないときは C を出力する。else を省略することはできない。
例えば LED_A は number=0,2,3,5,6,7 のとき 1 を出力し，number=1,4，それ以外のときは 0 を出力する。

```
            ‘0’ when number = 1 else
            ‘1’ when number = 2 else
            ‘0’ when number = 3 else
            ‘0’ when number = 4 else
            ‘0’ when number = 5 else
            ‘1’ when number = 6 else
            ‘0’ when number = 7 else
            ‘0’;

    LED_F <= ‘1’ when number = 0 else
            ‘0’ when number = 1 else
            ‘0’ when number = 2 else
            ‘0’ when number = 3 else
            ‘1’ when number = 4 else
            ‘1’ when number = 5 else
            ‘1’ when number = 6 else
            ‘0’ when number = 7 else
            ‘0’;

    LED_G <= ‘0’ when number = 0 else
            ‘0’ when number = 1 else
            ‘1’ when number = 2 else
            ‘1’ when number = 3 else
            ‘1’ when number = 4 else
            ‘1’ when number = 5 else
            ‘1’ when number = 6 else
            ‘0’ when number = 7 else
            ‘0’;

    LED_DP <= ‘0’ when number = 0 else
             ‘0’ when number = 1 else
             ‘0’ when number = 2 else
             ‘0’ when number = 3 else
             ‘0’ when number = 4 else
             ‘0’ when number = 5 else
             ‘0’ when number = 6 else
             ‘0’ when number = 7 else
             ‘0’;

end RTL;
```

いかがでしょうか。ちゃんと表示されましたか。

さて，今回新たに出てきた表現の一つは次の「std_logic_vector」です。

```
    signal number :    std_logic_vector(2 downto 0);
```

これは「std_logic」データタイプをバスタイプで定義することを意味しています。つまり，上の定義により，number(2)，number(1)，number(0)という 3 本の信号線が定義されます。さらに「number」という表現でまとめて扱うことができ，「number <= 3;」とすれば，number(2)=0，number(1)=1，number(0)=1 になります。なお，「number <= 3;」は「number <= “011”;」と表現することもできます。

今回は「2 downto 0」で定義しました。これは，最上位ビットが「2」ということを表しています。これを「0 to 2」で定義することもでき，その場合は最上位ビットが「0」になります。

もう一つ新たに出てきた表現は「when」文です。

```
    LED_A <= ‘1’ when number = 0 else
             ‘0’ when number = 1 else
             ‘1’ when number = 2 else
             ‘1’ when number = 3 else
             ‘0’ when number = 4 else
             ‘1’ when number = 5 else
             ‘1’ when number = 6 else
             ‘1’ when number = 7 else
             ‘0’;
```

基本形は、

A <= B when **＜条件＞** else C;

です。条件が成立するときは信号 A に B を出力し，成立しないときは C を出力します。Else を使うことで条件をたくさん並べることができます。なお，else を省略することはできません。

■ 別の方法でデコーダを作る

これでデコーダは完成ですが，プログラムがいろいろな方法で同じ動作ができるのと同じように，HDL も同じことを別の方法で実現することができます。フォルダ名は「decode2_7seg_vhdl」，プロジェクト名は「decode2_7seg_vhdl」，最上位階層のエンティティ名は「decode2_7seg_vhdl_top」とします。今までどおりプロジェクトを作成し，「decode2_7seg_vhdl_top．Vhd」をエディタで開いて下さい。そして，次のソースリストを入力・実行してみましょう。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- 7Segment Decode circuit Ver.2
-----------------------------------------------------------
-- File   : decode2_7seg_vhdl_top.vhd
-- Date   : 2007-08-09
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity decode2_7seg_vhdl_top is
    port(
        LED_A    :    out std_logic;
        LED_B    :    out std_logic;
        LED_C    :    out std_logic;
        LED_D    :    out std_logic;
```

```
        LED_E    :    out std_logic;
        LED_F    :    out std_logic;
        LED_G    :    out std_logic;
        LED_DP   :    out std_logic;
        SW5      :    in std_logic;
        SW6      :    in std_logic;
        SW7      :    in std_logic
    );
end decode2_7seg_vhdl_top;

architecture RTL of decode2_7seg_vhdl_top is

    signal number      :    std_logic_vector(2 downto 0);
    signal led_data    :    std_logic_vector(7 downto 0);

begin

    number(2) <= not SW7;
    number(1) <= not SW6;
    number(0) <= not SW5;

    process(number)
    begin
        case number is
            when "000" =>
                led_data <= "00111111";
            when "001" =>
                led_data <= "00000110";
            when "010" =>
                led_data <= "01011011";
            when "011" =>
                led_data <= "01001111";
            when "100" =>
                led_data <= "01100110";
            when "101" =>
                led_data <= "01101101";
            when "110" =>
                led_data <= "01111101";
            when "111" =>
                led_data <= "00000111";
            when others =>
                led_data <= "00000000";
        end case;
    end process;

    LED_A  <= led_data(0);
    LED_B  <= led_data(1);
    LED_C  <= led_data(2);
    LED_D  <= led_data(3);
    LED_E  <= led_data(4);
    LED_F  <= led_data(5);
    LED_G  <= led_data(6);
    LED_DP <= led_data(7);
```

```
end RTL;
```

表示される結果だけ見ると全く同じです。でも，ソースリストはだいぶコンパクトにまとまりました。では，もう少し詳しく見てましょう。

新たに出てきた表現は「process」文です。基本形は，

```
process（センシティビティ・リスト）
begin
  ～
プロセス文内のロジック
  ～
end process;
```

です。センシティビティ・リストとはプロセス文内のロジックを起動する信号のリストです。この信号が変化すると，プロセス文内のロジックが順次処理されます。ただし，処理されるのは評価判定のみで，代入操作は「end process」に到達したときに一気に行なわれます。組み合わせ回路でプロセス文を使う場合，センシティビティ・リストには，そのプロセス内で使用される入力信号を全て記述しなければなりません。

次ぎは「case」文です。プロセス文内のロジックで使用しました。基本形は次のとおりです。

```
Case 信号 is
    when 条件 1 =>
        条件 1 のときのロジック
    when 条件 2 =>
        条件 2 のときのロジック
    ～
    when others =>
        どの条件にも当てはまらない場合のロジック
end case;
```

指定した信号が when の後の条件に一致したときに「=>」の後に記述されているロジックを実行します。「others」はどの条件に当てはまらないときのロジックをまとめて記述する部分です。なお，入力条件は‘0’や‘1’だけではなく，‘X’，‘Z’，‘U’など，VHDL で使用可能な条件全てが含まれます。

ところで，「when」文を使わず最初から「case」文を使えばいいのに，と思った方もおられることでしょう。見た目のすっきり感や，リストの読みやすさからしても，「case」文の方がわかりやすいからです。でも，最初の例では使えなかったのです。なぜなら，「case」文は「process」文の中でしか使えない，からです。それに対し，「when」文はアーキテクチャ部で自由に使うことができます。

さて，ここでもう一つ大切な概念をおさえておきましょう。このリストのアーキテクチャ部は①②③の 3 つのブロックに分けることができます。これは，①をまず実行し，次に②を実行し，最後に③を実行して①に戻る，という意味ではありません。VHDL では①②③は式の順序に関係なく同時に処理されます。VHDL はあくまでハードウェアを記述する，つまり配線の仕方を記述する方法だからです。この同時処理という概念をぜひ覚えておきましょう。

■　練習問題

本文ではデコーダの例として 7 セグメント LED デコーダを考えてみました。しかし，7 セグメント LED デコーダよりも，よく使われているのはメモリセレクト回路で使われるアドレスデコーダです。手始めに，2 ビット 2 進入力－4 ビット出力のデコーダを，回路図入力と VHDL 入力で考えてみてください。真理値表は次のとおりです。（解答例は付属の CD をご覧ください，「decode_2to4_sch」と「decode_2to4_vhdl」）

入力			出力			
EN	A0	A1	Y0	Y1	Y2	Y3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	1	0	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	1

■　練習問題

もう一つ，アドレスデコーダを考えてみましょう。3 ビット 2 進入力‐8 ビット出力のデコーダを VHDL 入力で考えてみてください。真理値表は次のとおりです。（解答例は付属の CD をご覧ください，「decode_3to8_vhdl」）

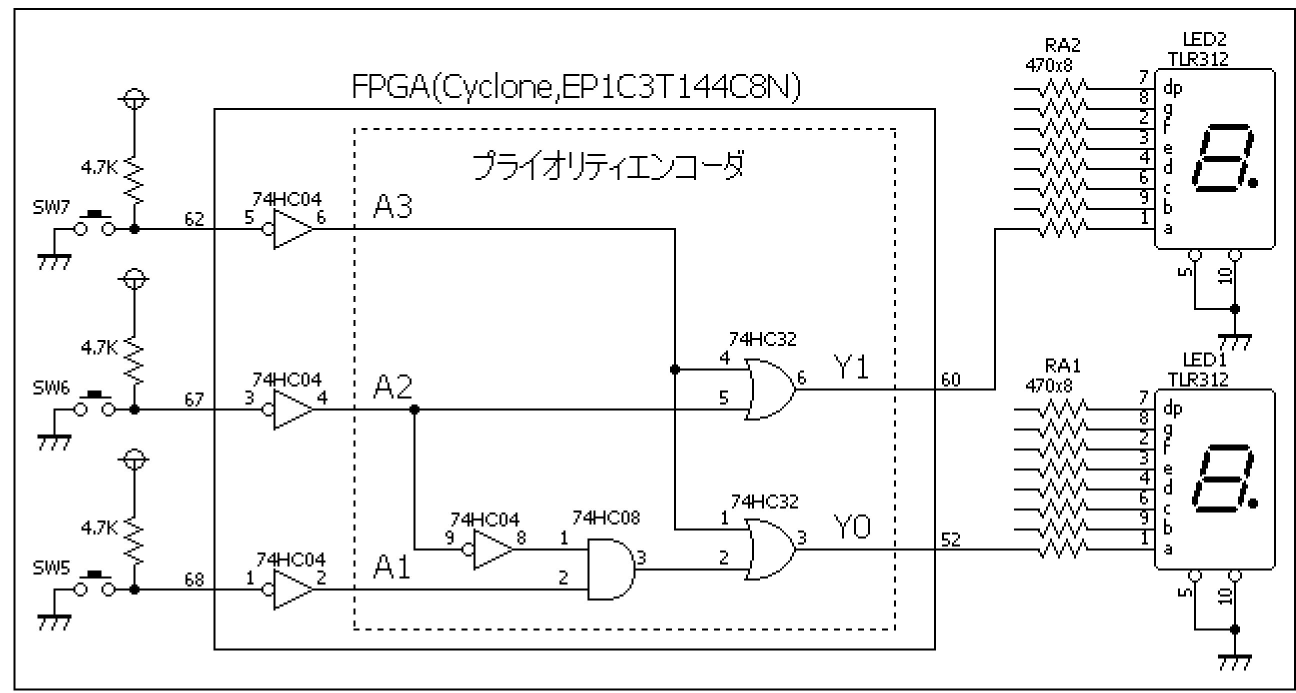
入力			出力							
A0	A1	A2	Y0	Y1	Y2	Y3	Y4	Y5	Y6	Y7
0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
1	1	0	0	0	0	1	0	0	0	0
0	0	1	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

■ エンコーダ

続いて，エンコーダ回路を考えてみましょう。エンコーダはデコーダと逆で，複数の信号線の入力に応じた 2 進数の信号を出力する回路です。言葉ではわかりづらいので，真理値表をみてください。(何も入力がないときも含め)4 入力-2 ビット 2 進出力のエンコーダです。

入力			出力	
A1	A2	A3	Y0	Y1
0	0	0	0	0
1	0	0	1	0
X	1	0	0	1
X	X	1	1	1

入力状態に‘X’がありますが，これは‘0’でも‘1’でも構わないことを意味しています。つまり，複数の入力が同時に‘1’になっても，もっとも優先順位の高い入力に対応する信号を出力します。このようなエンコーダをプライオリティエンコーダと呼びます。この真理値表の回路図は次のようになります。回路図入力で考えてみましょう。フォルダ名は「encode_4to2_sch」，プロジェクト名は「encode_4to2_sch」，最上位階層のエンティティ名は「encode_4to2_sch_top」とします。



次に，同じ回路を VHDL であらわしてみましょう。フォルダ名は「encode_4to2_vhdl」，プロジェクト名は「encode_4to2_vhdl」，最上位階層のエンティティ名は「encode_4to2_vhdl_top」とします。今までどおりプロジェクトを作成し，「encode_4to2_vhdl_top. vhd」をエディタで開いて下さい。そして，次のソースリストを入力・コンパイル・実行してみましょう。

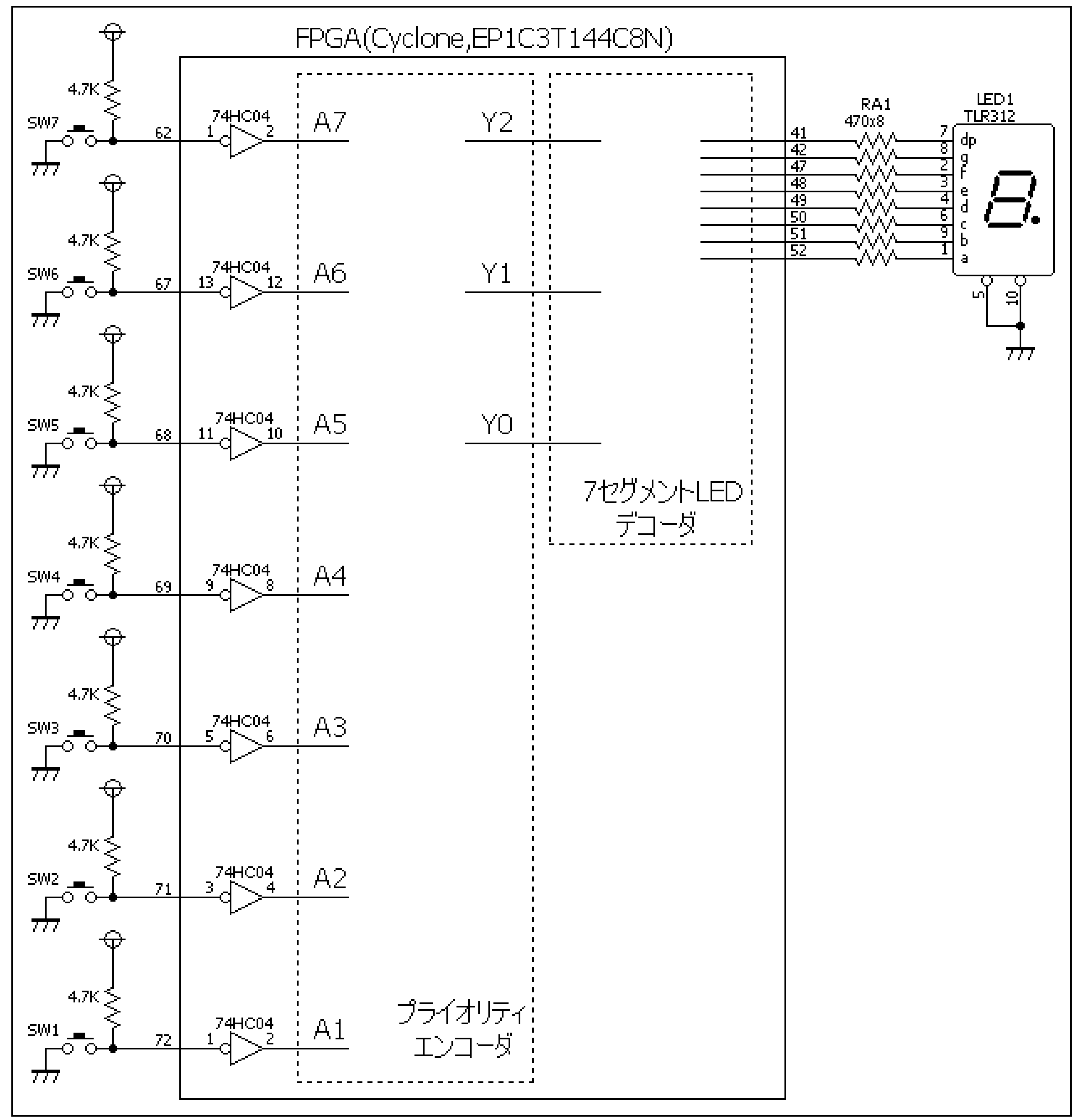
```
------------------------------------------------------------
-- FPGA Training Board(B6101)
-- 4 Line Input to 2 Line Output Priority Encoder Circuit
------------------------------------------------------------
-- File   : encode_4to2_vhdl_top.vhd
-- Date   : 2009-06-05
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
```

```
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity encode_4to2_vhdl_top is
    port(
        LED_1    :   out std_logic;
        LED_2    :   out std_logic;
        SW5      :   in std_logic;
        SW6      :   in std_logic;
        SW7      :   in std_logic
    );
end encode_4to2_vhdl_top;

architecture RTL of encode_4to2_vhdl_top is

    signal A1:   std_logic;
    signal A2:   std_logic;
    signal A3:   std_logic;
    signal Y :   std_logic_vector(1 downto 0);

begin

    A1 <= not SW5;
    A2 <= not SW6;
    A3 <= not SW7;

    process (A1,A2,A3)
    begin
        if(A3 = '1') then
            Y <= "11";
        elsif(A2 = '1') then
            Y <= "10";
        elsif(A1 = '1') then
            Y <= "01";
        else
            Y <= "00";
        end if;
    end process;

    LED_1 <= Y(0);
    LED_2 <= Y(1);

end RTL;
```

■ 練習問題

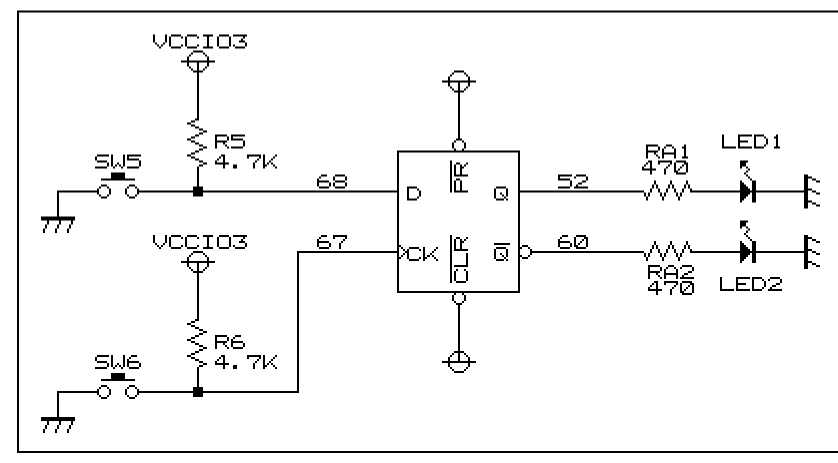
プライオリティエンコーダと7セグメントLEDデコーダを組み合わせて，SW1～7が押されたら，7セグメントLEDに‘1’～‘7’を表示してみましょう。何も押されていない時は‘0’を表示します。全体の回路図と，プライオリティエンコーダの真理値表は次のとおりです。（解答例は付属の CD をご覧ください，「encode_8to7seg_vhdl」）

入力							出力		
A1	A2	A3	A4	A5	A6	A7	Y0	Y1	Y2
0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	0	0
X	1	0	0	0	0	0	0	1	0
X	X	1	0	0	0	0	1	1	0
X	X	X	1	0	0	0	0	0	1
X	X	X	X	1	0	0	1	0	1
X	X	X	X	X	1	0	0	1	1
X	X	X	X	X	X	1	1	1	1



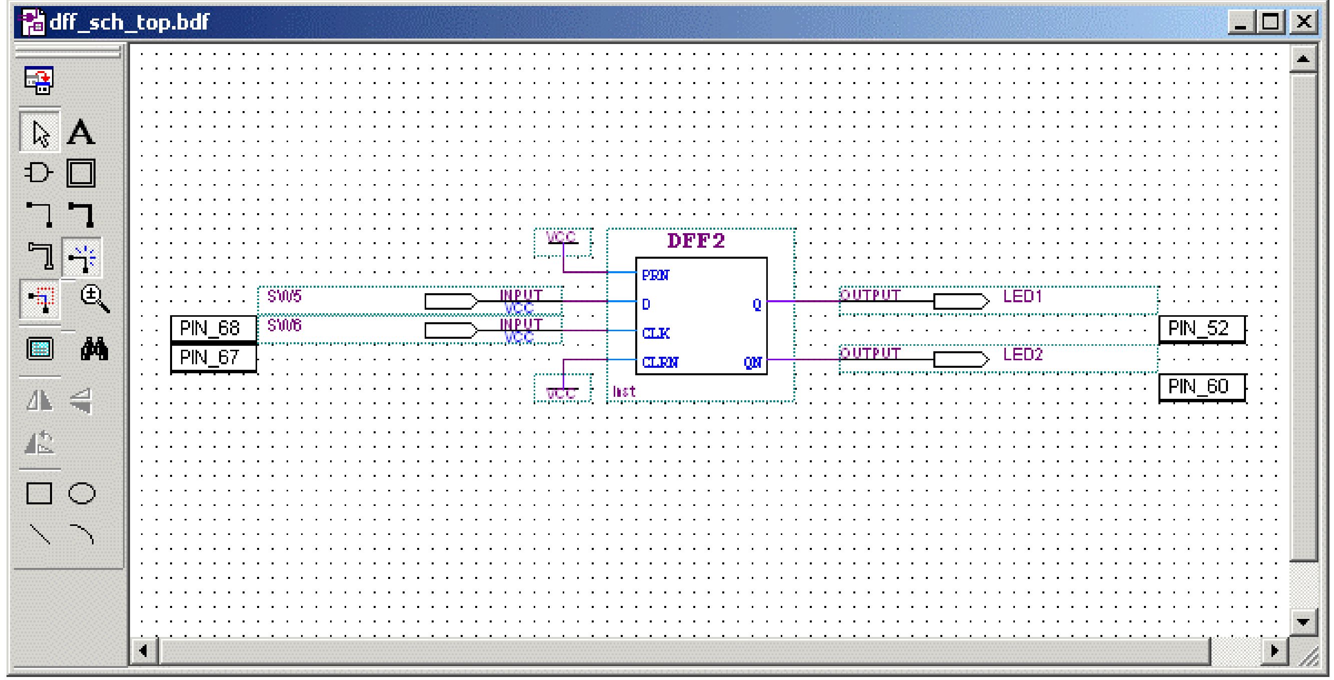
4. フリップフロップ

ロジック回路を大きく二つに分けると，今まで考えてきた組み合わせ回路と，順序回路に分けられます。組み合わせ回路は入力に応じて出力が一義的に決定される回路です。一方，順序回路はクロック信号に同期して出力が決定されます。順序回路のもっとも基本的な要素はフリップフロップです。この項ではフリップフロップを FPGA に組み込んでみましょう。回路図は次のとおりです。



■ 回路図入力

まずは回路図入力で考えてみましょう。フォルダ名は「dff_sch」，プロジェクト名は「dff_sch」，最上位階層のエンティティ名は「dff_sch_top」とします。今までどおりプロジェクトを作成し，「dff_sch_top.Bdf」をエディタで開いて下さい。そして，次の回路図を入力・コンパイル・実行してみましょう。



この「DFF2」の真理値表は次のようになります。

入力				出力	
PRN	CLRN	CLK	D	Q	QN
0	1	X	X	1	0
1	0	X	X	0	1
0	0	X	X	不定	不定
1	1	0	X	変化なし	変化なし
1	1	↑	0	0	1
1	1	↑	1	1	0
1	1	1	X	変化なし	変化なし
1	1	↓	X	変化なし	変化なし

この真理値表で特に注目したいのは黄色でマークした部分です。入力 D の状態が CLK の立ち上がりで出力 Q と QN に伝わります。入力の状態が出力にすぐ伝わるのではなく，次に CLK が立ち上がるまで遅れるので「ディレイフリップフロップ」と呼ばれています。

さて，今回 Cyclone に書きこんだ回路は PRN と CLRN を Vcc に固定しています。それで，CLK と D のみが意味を持ちます。

SW5(D)をオン/オフしてもすぐに LED 表示に反映されません。SW5(D)をオン/オフした状態のまま，SW6(CLK)をオンからオフに変化させたとき，つまり CLK が 0 から 1 に変化したときに LED 表示が変化します。(SW6 をオフからオンにしたときに表示が変わることがあるかもしれません。これはスイッチのチャタリングの影響です。チャタリングの除去については「5. カウンタ」で検討します。)

■ VHDL 入力

では，同じ回路を VHDL であらわしてみましょう。なお，入力の PRN と CLRN は省略し D と CLK に限定します。フォルダ名は「dff_vhdl」，プロジェクト名は「dff_vhdl」，最上位階層のエンティティ名は「dff_vhdl_top」とします。今までどおりプロジェクトを作成し，「dff_vhdl_top. Vhd」をエディタで開いて下さい。そして，次のソースリストを入力・コンパイル・実行してみましょう。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- Delay Flip Flop circuit
-----------------------------------------------------------
-- File    : dff_vhdl_top.vhd
-- Date    : 2007-08-14
-- Family  : Cyclone
-- Device  : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity dff_vhdl_top is
    port(
```

```
        LED1     :   out std_logic;
        LED2     :   out std_logic;
        SW5      :   in std_logic;
        SW6      :   in std_logic
    );
end dff_vhdl_top;

architecture RTL of dff_vhdl_top is

    signal D     :   std_logic;
    signal CLK   :   std_logic;
    signal Q     :   std_logic;
    signal QN    :   std_logic;

begin

    D   <= SW5;
    CLK <= SW6;

    process(CLK)
    begin
        if(CLK'event and CLK='1') then
            Q <= D;
        end if;
    end process;
    QN <= not Q;

    LED1  <= Q;
    LED2  <= QN;

end RTL;
```

さて，ポイントの一つは前項にも出てきた「process」文です。

Process（**センシティビティ・リスト**）
begin
　～
プロセス文内のロジック
　～
end process;

フリップフロップは「CLK」の変化に同期して出力が変化します。それで，センシティビティ・リストには「CLK」を記述します。

もう一つのポイントは「if」文です。基本形は，

if **条件** 1 then
　ロジック 1
elsif **条件** 2 then
　ロジック 2
else
　ロジック 3
end if;

となります。「elsif」は省略したり条件をさらに追加したりすることができます。また，「else」は残りの全ての場合という意味ですが，省略することができます。

ところで，前項ででてきた「case」文と意味がよく似ていますね。どちらも条件によって動作を変えるときに使います。どのように使い分けるのでしょうか。「case」文の条件の判定は同時に評価され，優先的に評価される信号はありません。一方，「if」文の条件の判定は上から順番に評価されるため，優先順位のある回路を記述するときは「if」文を使うことになります。

もう一つ，今回の条件の中ででてくる表現もよく使われます。

```
If (CLK'event and CLK='1') then
```

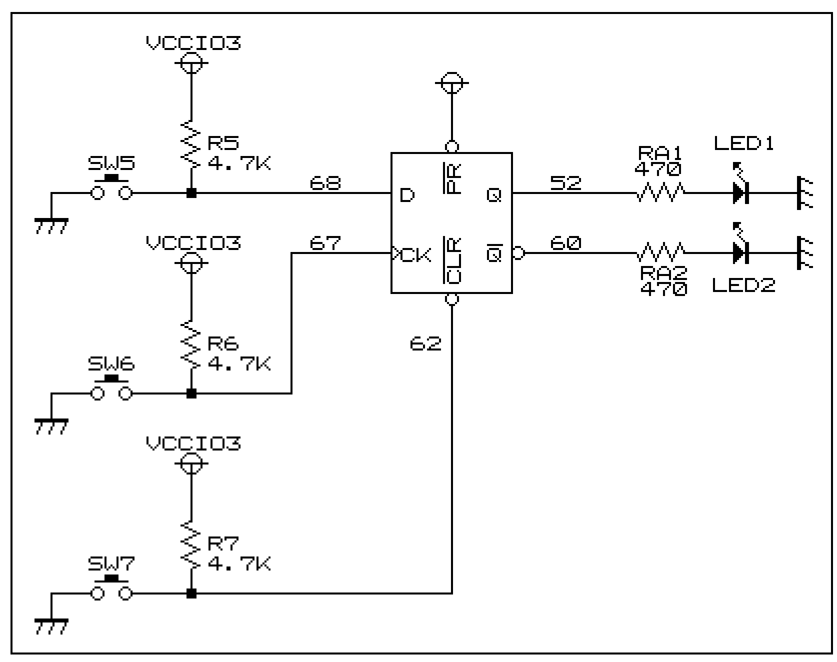
信号の変化をイベントといいますが，「'event」は「CLK」が変化したら，という意味です。そのあと「and」が記述されているので，「if」文実行のためにはその次の条件も満たす必要があります。「CLK='1'」つまりイベントが起こった結果「CLK」が 1 になったら，ということです。まとめると「CLK が 0 から 1 に変化したら（立ち上がりエッジ）実行する」ということになります。もし，

```
if (CLK'event and CLK='0') then
```

となっていたら，立ち下がりエッジで実行することになります。信号の立ち上がりエッジや立ち下がりエッジで実行する回路はよくあるので，この表現も「おまじない」的に覚えてしまうと良いでしょう。

■ 練習問題

今まで Vcc に固定していた CLRN を SW7 につないで，CLRN=0 で強制リセット（Q=0，QN=1）する回路を作ってみましょう。もちろん，強制リセット機能は優先順位が高いです。（「dff_clrn_vhdl」）



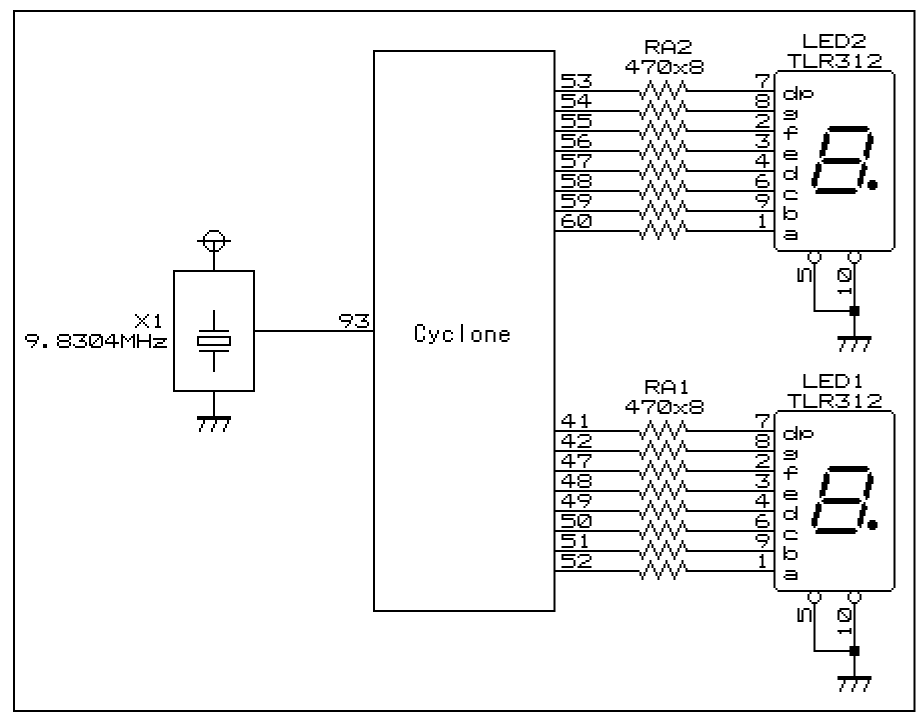
解答例

```
------------------------------------------------------------
-- FPGA Training Board(B6101)
-- Delay Flip Flop with Clear circuit
------------------------------------------------------------
-- File   : dff_clrn_vhdl_top.vhd
-- Date   : 2007-08-17
-- Family : Cyclone
-- Device : EP1C3T144C8
```

```
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity dff_clrn_vhdl_top is
    port(
        LED1      :    out std_logic;
        LED2      :    out std_logic;
        SW5       :    in std_logic;
        SW6       :    in std_logic;
        SW7       :    in std_logic
    );
end dff_clrn_vhdl_top;

architecture RTL of dff_clrn_vhdl_top is

    signal D      :    std_logic;
    signal CLK    :    std_logic;
    signal CLRN   :    std_logic;
    signal Q      :    std_logic;
    signal QN     :    std_logic;

begin

    D    <= SW5;
    CLK  <= SW6;
    CLRN <= SW7;

    process(CLK,CLRN)
    begin
        if(CLRN='0') then
            Q <= '0';
        elsif(CLK'event and CLK='1') then
            Q <= D;
        end if;
    end process;
    QN <= not Q;

    LED1  <= Q;
    LED2  <= QN;

end RTL;
```

5. カウンタ

順序回路の例としてカウンタを作ってみましょう。カウンタを動かすクロックは発振モジュールを分周することにし，カウンタの出力を 7 セグメント LED に表示します。



フォルダ名は「count_vhdl」，プロジェクト名は「count_vhdl」，最上位階層のエンティティ名は「count_vhdl_top」とします。今までどおりプロジェクトを作成し，「count_vhdl_top. vhd」をエディタで開いて下さい。そして，次のソースリストを入力・コンパイル・実行してみましょう。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- Counter circuit
-----------------------------------------------------------
-- File   : count_vhdl_top.vhd
-- Date   : 2007-08-17
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------


library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity count_vhdl_top is
    port(
        CLK      :   in std_logic;
        LED1A    :   out std_logic;
        LED1B    :   out std_logic;
        LED1C    :   out std_logic;
```

```
        LED1D    :   out std_logic;
        LED1E    :   out std_logic;
        LED1F    :   out std_logic;
        LED1G    :   out std_logic;
        LED1DP   :   out std_logic;
        LED2A    :   out std_logic;
        LED2B    :   out std_logic;
        LED2C    :   out std_logic;
        LED2D    :   out std_logic;
        LED2E    :   out std_logic;
        LED2F    :   out std_logic;
        LED2G    :   out std_logic;
        LED2DP   :   out std_logic
    );
end count_vhdl_top;

architecture RTL of count_vhdl_top is

    signal counter    :   std_logic_vector(7 downto 0);
    signal div_cnt    :   std_logic_vector(31 downto 0);
    signal div_clk    :   std_logic;
    signal led1       :   std_logic_vector(7 downto 0);
    signal led2       :   std_logic_vector(7 downto 0);

begin

    process(CLK)
    begin
        if(CLK'event and CLK='1') then
            div_cnt <= div_cnt + 1;
        end if;
    end process;
    div_clk <= div_cnt(22);

    process(div_clk)
    begin
        if(div_clk'event and div_clk='1') then
            counter <= counter + 1;
        end if;
    end process;

    process(counter)
    begin
        case counter(7 downto 4) is
            when "0000" =>
                led2 <= "00111111";
            when "0001" =>
                led2 <= "00000110";
            when "0010" =>
                led2 <= "01011011";
            when "0011" =>
                led2 <= "01001111";
            when "0100" =>
```

```
                led2 <= “01100110”;
            when “0101” =>
                led2 <= “01101101”;
            when “0110” =>
                led2 <= “01111101”;
            when “0111” =>
                led2 <= “00000111”;
            when “1000” =>
                led2 <= “01111111”;
            when “1001” =>
                led2 <= “01101111”;
            when “1010” =>
                led2 <= “01110111”;
            when “1011” =>
                led2 <= “01111100”;
            when “1100” =>
                led2 <= “00111001”;
            when “1101” =>
                led2 <= “01011110”;
            when “1110” =>
                led2 <= “01111001”;
            when “1111” =>
                led2 <= “01110001”;
            when others =>
                led2 <= “00000000”;
        end case;
        case counter(3 downto 0) is
            when “0000” =>
                led1 <= “00111111”;
            when “0001” =>
                led1 <= “00000110”;
            when “0010” =>
                led1 <= “01011011”;
            when “0011” =>
                led1 <= “01001111”;
            when “0100” =>
                led1 <= “01100110”;
            when “0101” =>
                led1 <= “01101101”;
            when “0110” =>
                led1 <= “01111101”;
            when “0111” =>
                led1 <= “00000111”;
            when “1000” =>
                led1 <= “01111111”;
            when “1001” =>
                led1 <= “01101111”;
            when “1010” =>
                led1 <= “01110111”;
            when “1011” =>
                led1 <= “01111100”;
            when “1100” =>
                led1 <= “00111001”;
```

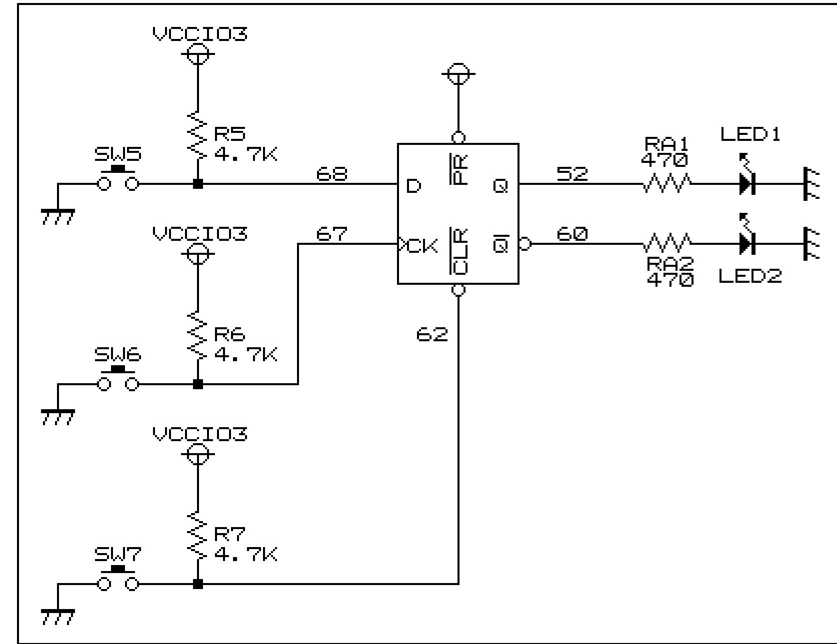
```
            when “1101” =>
                led1 <= “01011110”;
            when “1110” =>
                led1 <= “01111001”;
            when “1111” =>
                led1 <= “01110001”;
            when others =>
                led1 <= “00000000”;
        end case;
    end process;

    LED1A  <= led1(0);
    LED1B  <= led1(1);
    LED1C  <= led1(2);
    LED1D  <= led1(3);
    LED1E  <= led1(4);
    LED1F  <= led1(5);
    LED1G  <= led1(6);
    LED1DP <= led1(7);
    LED2A  <= led2(0);
    LED2B  <= led2(1);
    LED2C  <= led2(2);
    LED2D  <= led2(3);
    LED2E  <= led2(4);
    LED2F  <= led2(5);
    LED2G  <= led2(6);
    LED2DP <= led2(7);

end RTL;
```

「00」～「FF」までカウントアップします。ちゃんと表示されましたか？

■ 練習問題

今作ったのは「00」～「FF」までの 16 進数カウンタでした。では，「00」～「99」までの 10 進数カウンタを作ってみましょう。（解答例は付属 CD をご覧ください，「bcd_count_vhdl」）

■　**チャタリングの除去について**

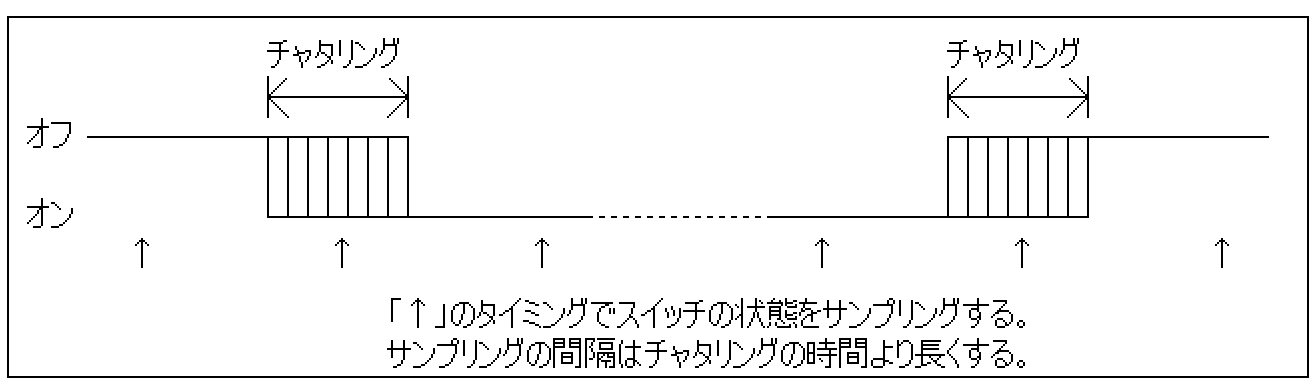


カウンタ回路ができるようになると，それを利用してスイッチのチャタリングの除去ができます。もう一度，右の回路を考えてみましょう。

「4．フリップフロップ」でこの回路を作ったときは，SW6（CLK）をオンからオフにしたときだけでなく，オフからオンにしたときにも出力が変化したことでしょう。それは，スイッチの構造から生じた現象です。

通常，このようなメカニカルなスイッチはバネなどでテンションのかかった接点同士を開いたり閉じたりします。そのためスイッチをオンしたとき，あるいはオフしたときに，接点がぶつかりあって何度も開いたり閉じたりを繰り返した後に安定します。その結果，ごく短い時間（1ms～10ms 程度）に何度もスイッチがオン/オフしたと認識されてしまいます。これをチャタリングと呼び，この現象を除去してからでないと誤動作を引き起こします。

除去する方法はいろいろありますが，今回はカウンタ回路を使って発振モジュールの 9.8304MHz を分周して，10 数 ms 間隔でスイッチ信号をサンプリングすることで除去します。タイミングチャートをご覧ください。



サンプリングの間隔がチャタリングの時間より長ければ，チャタリング中ではなく安定状態のときにサンプリングできるかもしれません。また，チャタリング中にサンプリングしたとしても，それがオンであれオフであれ，前後の安定したオン/オフのどちらかと連続します。結果としてチャタリングを除去することができます。

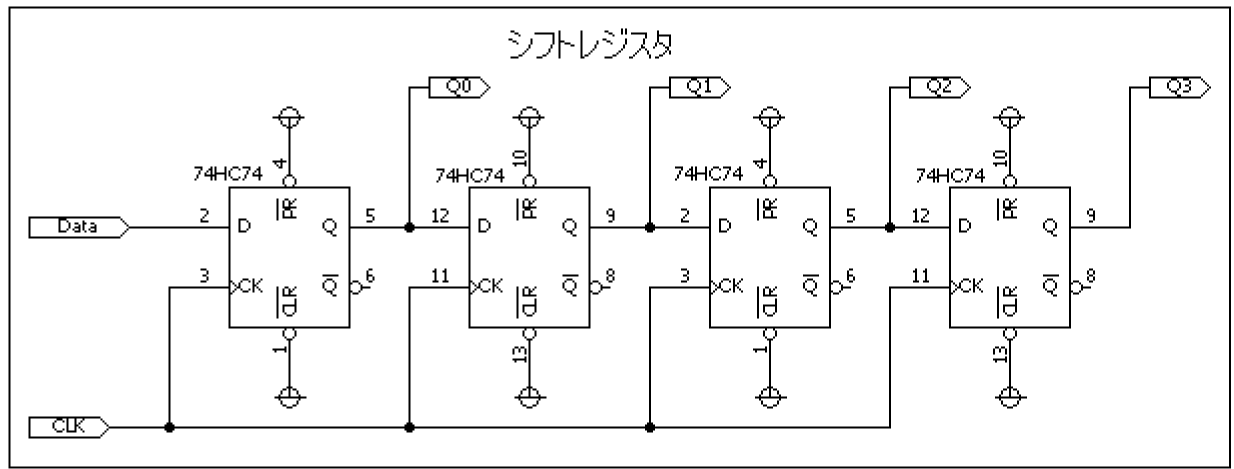
それでは VHDL 入力で考えてみましょう。フォルダ名は「dff_clrn_2_vhdl」，プロジェクト名は「dff_clrn_2_vhdl」，最上位階層のエンティティ名は「dff_clrn_2_vhdl_top」とします。今までどおりプロジェクトを作成し，「dff_clrn_2_vhdl_top．vhd」をエディタで開いて下さい。そして，次のソースリストを入力・コンパイル・実行してみましょう。

```
---------------------------------------------------------------
-- FPGA Training Board(B6101)
-- Delay Flip Flop with Clear Circuit
---------------------------------------------------------------
-- File   : dff_clrn_2_vhdl_top.vhd
-- Date   : 2009-06-08
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
---------------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity dff_clrn_2_vhdl_top is
    port(
        CLKIN    :    in std_logic;
        LED1     :    out std_logic;
        LED2     :    out std_logic;
        SW5      :    in std_logic;
        SW6      :    in std_logic;
        SW7      :    in std_logic
    );
end dff_clrn_2_vhdl_top;

architecture RTL of dff_clrn_2_vhdl_top is

    signal div_cnt    :    std_logic_vector(23 downto 0);
    signal div_clk_sw :    std_logic;
    signal sw         :    std_logic_vector(2 downto 0);

    signal D          :    std_logic;
    signal CLK        :    std_logic;
    signal CLRN       :    std_logic;
    signal Q          :    std_logic;
    signal QN         :    std_logic;

begin

    process(CLKIN)
    begin
        if(CLKIN'event and CLKIN='1') then
            div_cnt <= div_cnt + 1;
        end if;
    end process;
    div_clk_sw <= div_cnt(16);

    process(div_clk_sw)
```

```
    begin
        if(div_clk_sw'event and div_clk_sw='1') then
            sw(0) <= SW5;
            sw(1) <= SW6;
            sw(2) <= SW7;
        end if;
    end process;

    D    <= sw(0);
    CLK  <= sw(1);
    CLRN <= sw(2);

    process(CLK,CLRN)
    begin
        if(CLRN='0') then
            Q <= '0';
        elsif(CLK'event and CLK='1') then
            Q <= D;
        end if;
    end process;
    QN <= not Q;

    LED1  <= Q;
    LED2  <= QN;

end RTL;
```

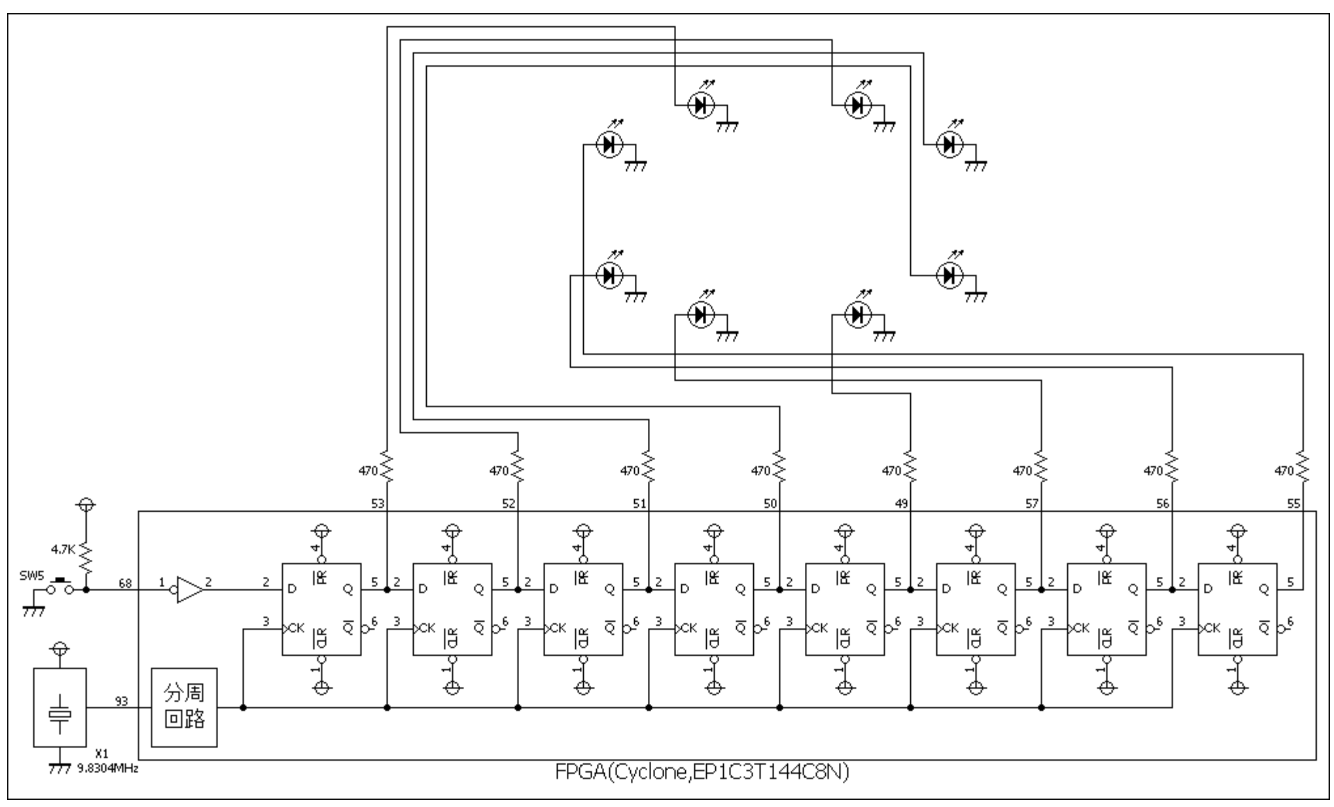
6. シフトレジスタ

フリップフロップの応用としてシフトレジスタを作ります。シリアル⇔パラレル変換で使われています。シフトレジスタの基本形は次のようなものです。



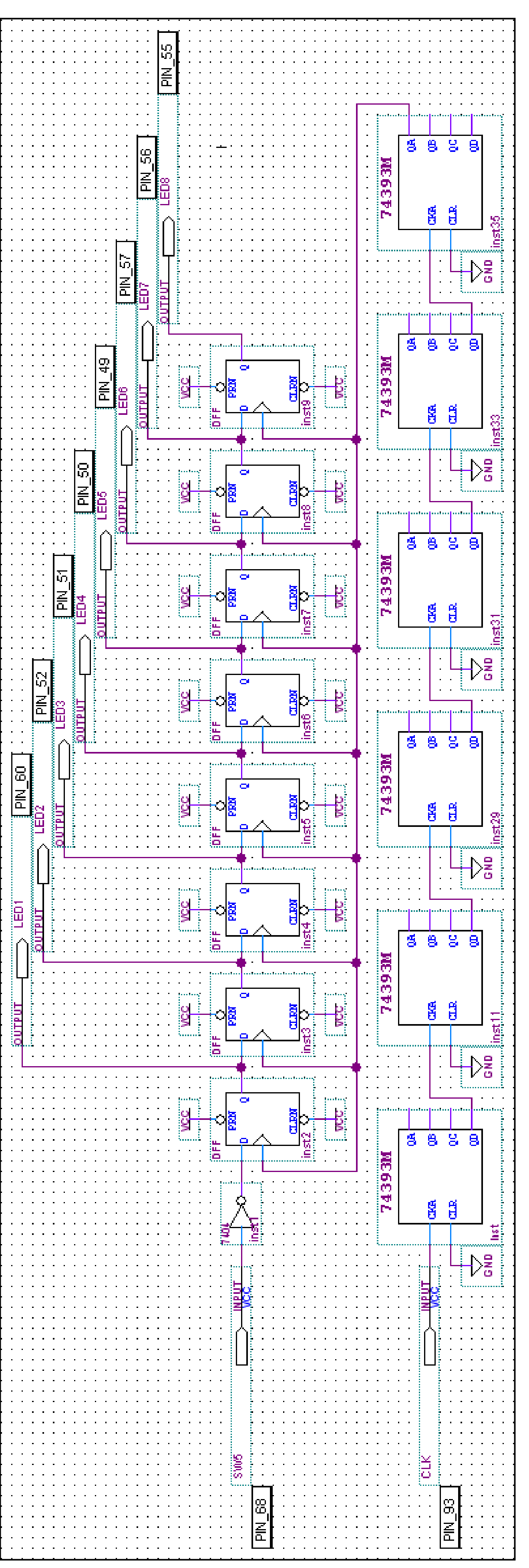
クロック（CLK）に同期してシリアルデータ（Data）が順番に取り込まれていきます。4 クロック入力後，Q0～Q3 からパラレルデータを取り出すことができます。フリップフロップを追加すれば，任意のビット数のパラレル変換ができます。

では実際にシフトレジスタを作ってみましょう。回路図は次のようなものです。スイッチの状態を取り込んで，LED に表示します。



■　回路図入力

　まずは回路図入力で考えてみましょう。フォルダ名は「shift_reg_sch」，プロジェクト名は「shift_reg_sch」，最上位階層のエンティティ名は「shift_reg_sch_top」とします。今までどおりプロジェクトを作成し，「shift_reg_sch_top．Bdf」をエディタで開いて下さい。そして，右の回路図を入力・コンパイル・実行してみましょう。



■ VHDL 入力

では，同じ回路をVHDLであらわしてみましょう。フォルダ名は「shift_reg_vhdl」，プロジェクト名は「shift_reg_vhdl」，最上位階層のエンティティ名は「shift_reg_vhdl_top」とします。今までどおりプロジェクトを作成し，「shift_reg_vhdl_top．Vhd」をエディタで開いて下さい。そして，次のソースリストを入力・コンパイル・実行してみましょう。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- Shift Register Circuit
-----------------------------------------------------------
-- File   : shift_reg_vhdl_top.vhd
-- Date   : 2009-06-05
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity shift_reg_vhdl_top is
    port(
        CLK     :   in std_logic;
        SW5     :   in std_logic;
        LED1    :   out std_logic;
        LED2    :   out std_logic;
        LED3    :   out std_logic;
        LED4    :   out std_logic;
        LED5    :   out std_logic;
        LED6    :   out std_logic;
        LED7    :   out std_logic;
        LED8    :   out std_logic
    );
end shift_reg_vhdl_top;

architecture RTL of shift_reg_vhdl_top is

    signal D        :   std_logic;
    signal Q        :   std_logic_vector(7 downto 0);
    signal div_cnt  :   std_logic_vector(31 downto 0);
    signal div_clk  :   std_logic;

begin

    process(CLK)
    begin
        if(CLK'event and CLK='1') then
```

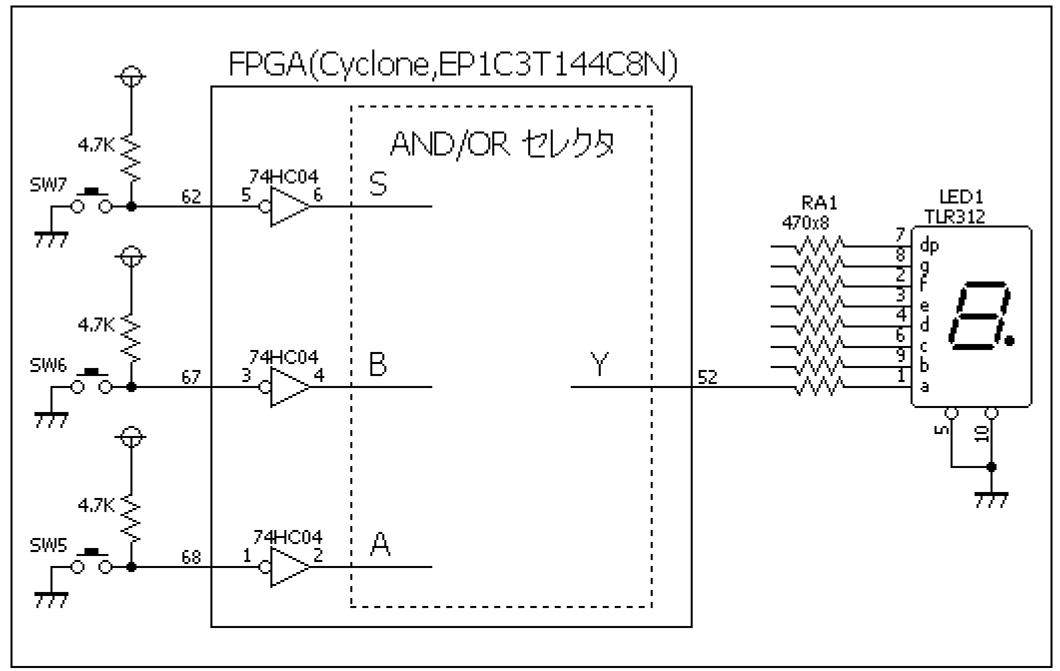
```
            div_cnt <= div_cnt + 1;
        end if;
    end process;
    div_clk <= div_cnt(20);

    D <= not SW5;

    process(div_clk)
    begin
        if(div_clk'event and div_clk='1') then
            Q(7) <= Q(6);
            Q(6) <= Q(5);
            Q(5) <= Q(4);
            Q(4) <= Q(3);
            Q(3) <= Q(2);
            Q(2) <= Q(1);
            Q(1) <= Q(0);
            Q(0) <= D;
        end if;
    end process;

    LED1 <= Q(0);
    LED2 <= Q(1);
    LED3 <= Q(2);
    LED4 <= Q(3);
    LED5 <= Q(4);
    LED6 <= Q(5);
    LED7 <= Q(6);
    LED8 <= Q(7);

end RTL;
```

7. VHDL の記述方法について

これまで VHDL による回路の記述を幾例か取り上げてきました。ここで記述方法についてまとめておきましょう。

論理回路を設計するとき，要求仕様からどのような動作をさせるか検討します。その際，タイミングチャートや真理値表を作ることになるかもしれません。その後 VHDL で回路を設計・入力していきます。この，設計・入力する際に二つの方法に分かれます。「回路図・論理式にして基本論理回路の組み合わせを考えて記述する」方法と，「動作の説明文を考えて記述する」方法です。具体例を元に考えてみましょう。

例題：入力‘A’と‘B’，出力‘Y’があるとき，選択信号‘S’が‘0’のときは‘A’と‘B’の AND を‘Y’に出力し，選択信号‘S’が‘1’のときは‘A’と‘B’の OR を‘Y’に出力する回路を設計せよ。（下回路図の AND/OR セレクタの部分）



真理値表は次のようになります。

入力			出力
S	A	B	Y
0	0	0	0
0	1	0	0
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	1
1	0	1	1
1	1	1	1

では，それぞれの方法で設計していきます。

■ **「回路図・論理式にして基本論理回路の組み合わせを考えて記述する」方法（その 1）**

出力‘Y’が‘1’になる条件に注目し論理式にします。

$$Y = \overline{S} \cdot A \cdot B + S \cdot A \cdot \overline{B} + S \cdot \overline{A} \cdot B + S \cdot A \cdot B$$

これをそのまま VHDL にすると，次のようになります。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- Select AND OR Circuit (1)
-----------------------------------------------------------
-- File   : select_and or_1_vhdl_top.vhd
-- Date   : 2009-06-08
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity select_and_or_1_vhdl_top is
    port(
        LED1 :   out std_logic;
        SW5  :   in std_logic;
        SW6  :   in std_logic;
        SW7  :   in std_logic
    );
end select_and_or_1_vhdl_top;

architecture RTL of select_and_or_1_vhdl_top is

    signal A :   std_logic;
    signal B :   std_logic;
    signal S :   std_logic;
    signal Y :   std_logic;

begin

    A <= not SW5;
    B <= not SW6;
    S <= not SW7;

    Y <= (not S and A and B) or (S and A and not B) or (S and not A and B) or (S and A and B);

    LED1 <= Y;

end RTL;
```

■ 「回路図・論理式にして基本論理回路の組み合わせを考えて記述する」方法(その 2)

「その 1」の論理式は真理値表のままなので冗長に感じるかもしれません。論理式をまとめて論理圧縮してみましょう。(論理圧縮で使用するブール代数の公式は巻末の付録を参照)

$$
\begin{aligned}
Y &= \overline{S} \cdot A \cdot B + S \cdot A \cdot \overline{B} + S \cdot \overline{A} \cdot B + S \cdot A \cdot B \\
&= \overline{S} \cdot A \cdot B + S \cdot A \cdot \overline{B} + S \cdot \overline{A} \cdot B + S \cdot A \cdot B + S \cdot A \cdot B + S \cdot A \cdot B \\
&= \overline{S} \cdot A \cdot B + S \cdot A \cdot B + S \cdot A \cdot \overline{B} + S \cdot A \cdot B + S \cdot \overline{A} \cdot B + S \cdot A \cdot B \\
&= A \cdot B \cdot (\overline{S} + S) + S \cdot A \cdot (\overline{B} + B) + S \cdot B \cdot (\overline{A} + A) \\
&= A \cdot B + S \cdot A + S \cdot B
\end{aligned}
$$

これを VHDL にすると, 次のようになります。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- Select AND OR Circuit (2)
-----------------------------------------------------------
-- File   : select_and or_2_vhdl_top.vhd
-- Date   : 2009-06-08
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity select_and_or_2_vhdl_top is
    port(
        LED1 :   out std_logic;
        SW5  :   in std_logic;
        SW6  :   in std_logic;
        SW7  :   in std_logic
    );
end select_and_or_2_vhdl_top;

architecture RTL of select_and_or_2_vhdl_top is

    signal A :   std_logic;
    signal B :   std_logic;
    signal S :   std_logic;
    signal Y :   std_logic;

begin

    A <= not SW5;
    B <= not SW6;
```

```
    S <= not SW7;

    Y <= (A and B) or (S and A) or (S and B);

    LED1 <= Y;

end RTL;
```

だいぶ簡潔になりました。もっとも，この式を見て例題の内容を逆に読み取ることは難しいかもしれません。

■ 「動作の説明文を考えて記述する」方法（その 1）

例題の「入力‘A’と‘B’，出力‘Y’があるとき，選択信号‘S’が‘0’のときは‘A’と‘B’の AND を‘Y’に出力し，選択信号‘S’が‘1’のときは‘A’と‘B’の OR を‘Y’に出力する回路」という文章を，「when」文を使ってそのまま表わすと次のようになります。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- Select AND OR Circuit (3)
-----------------------------------------------------------
-- File   : select_and or_3_vhdl_top.vhd
-- Date   : 2009-06-08
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity select_and_or_3_vhdl_top is
    port(
        LED1 :   out std_logic;
        SW5  :   in std_logic;
        SW6  :   in std_logic;
        SW7  :   in std_logic
    );
end select_and_or_3_vhdl_top;

architecture RTL of select_and_or_3_vhdl_top is

    signal A :   std_logic;
    signal B :   std_logic;
    signal S :   std_logic;
    signal Y :   std_logic;
    signal Y1:   std_logic;
    signal Y2:   std_logic;
```

```
begin

    A <= not SW5;
    B <= not SW6;
    S <= not SW7;

    Y1 <= A and B;
    Y2 <= A or B;
    Y <= Y1 when (S = '0') else Y2;

    LED1 <= Y;

end RTL;
```

■ 「動作の説明文を考えて記述する」方法（その 2）

例題の「入力‘A’と‘B’，出力‘Y’があるとき，選択信号‘S’が‘0’のときは‘A’と‘B’の AND を‘Y’に出力し，選択信号‘S’が‘1’のときは‘A’と‘B’の OR を‘Y’に出力する回路」という文章は，「process」文と「if」文を使って次のように表わすこともできます。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- Select AND OR Circuit (4)
-----------------------------------------------------------
-- File   : select_and or_4_vhdl_top.vhd
-- Date   : 2009-06-08
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity select_and_or_4_vhdl_top is
    port(
        LED1 :   out std_logic;
        SW5  :   in std_logic;
        SW6  :   in std_logic;
        SW7  :   in std_logic
    );
end select_and_or_4_vhdl_top;

architecture RTL of select_and_or_4_vhdl_top is

    signal A :   std_logic;
```

```
    signal B :    std_logic;
    signal S :    std_logic;
    signal Y :    std_logic;

begin

    A <= not SW5;
    B <= not SW6;
    S <= not SW7;

    process(A,B,S)
    begin
        if(S='0') then
            Y <= A and B;
        else
            Y <= A or B;
        end if;
    end process;

    LED1 <= Y;

end RTL;
```

◆

どちらの方法で記述するかはケースバイケースです。このマニュアルで取り上げている程度の規模であれば，どちらの方法を使っても大差はありません。しかしながら，回路規模が大きく複雑になるほど，論理回路の組み合わせを考えて記述するより，動作を説明文で考えて記述するほうが，コンパクトでわかりやすく直感的に表わすことができ，しかも，あとで見直しても理解しやすいためメンテナンスが容易になります。

第6章 ドットマトリックス LED に表示する

1. ドットマトリックス LED の表示方法
2. イラストを表示しよう
3. スイッチで表示を切り替える

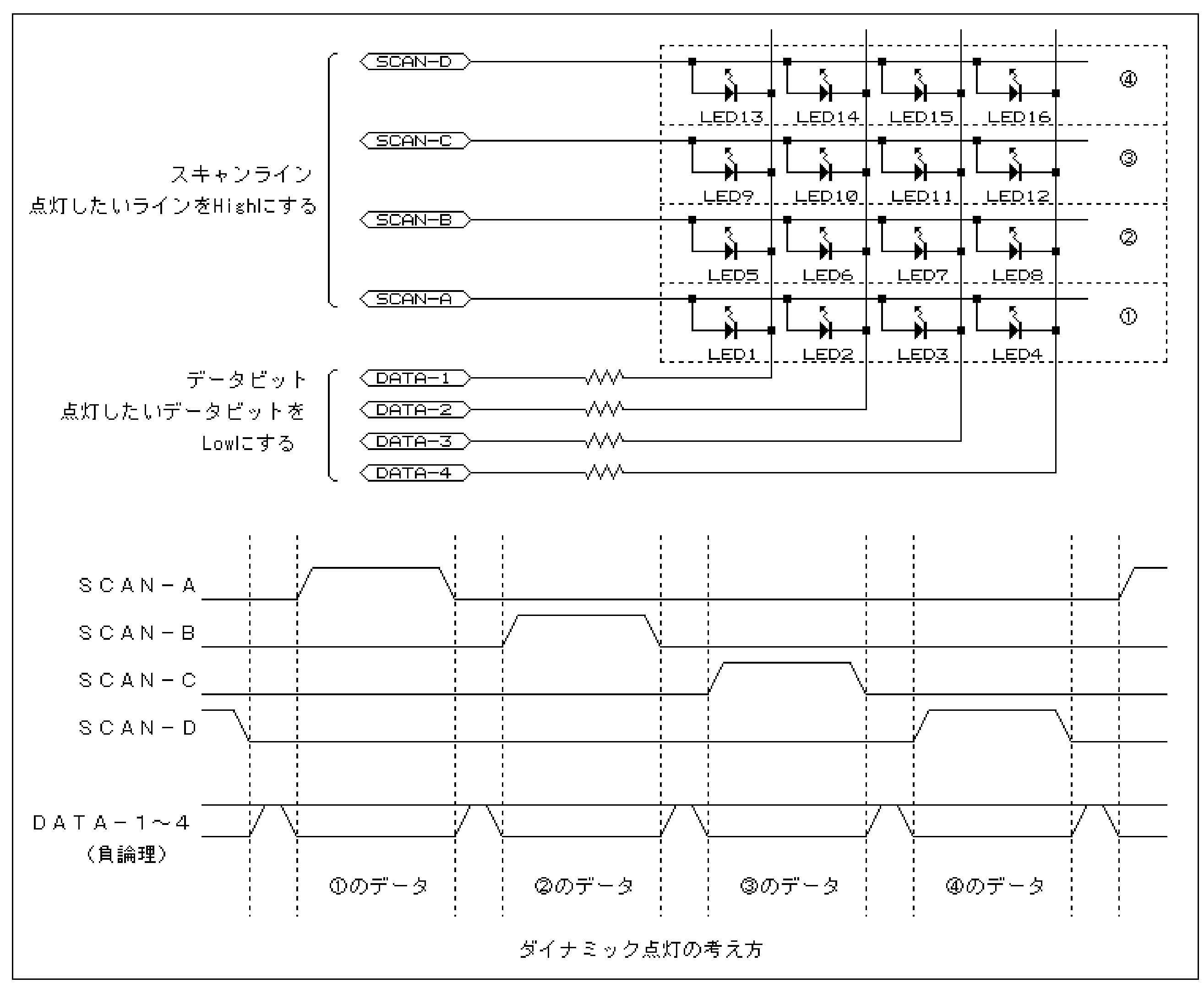
FPGA ボードには 2 色 16×16ドットマトリックスLED が実装されています。この章では，この LED にイラストを表示してみましょう。

1. ドットマトリックス LED の表示方法

FPGA ボードに実装されている LED は，1 ドットに赤と緑の LED が組み込まれています。それで，赤と緑，さらに両方点灯して橙の 3 色で点灯させることができます。

さて，そうするとLED は全部で何個になるでしょうか。16ドット×16ドット×2 色＝512 個です。なので，まともに LED を制御しようとすると 512 ビットの出力が必要になります。ただ LED を点灯するだけで 512 ビットも使うのはもったいないですよね。そもそも FPGA ボードに実装されている Cyclone は 104 本しか I/O ピンはありませんし・・・。

このようなときは，ダイナミック点灯という方法を使うのが定番です。ダイナミック点灯を使えば 512 個の LED も 48 ビットの出力で制御できます。そもそも，この LED はダイナミック点灯を使うことを前提に作られています。ダイナミック点灯とはどのような方法でしょうか。少し省略して 16 個の LED を制御する回路とタイミングチャートを見てみましょう。



ダイナミック点灯の考え方

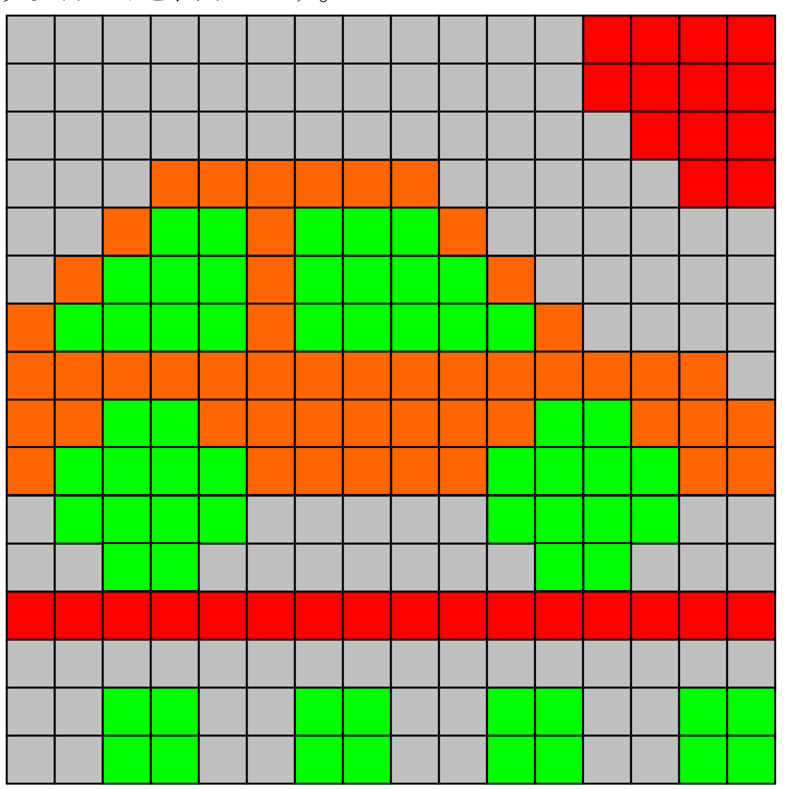
①の LED を光らせる場合，まず SCAN-A を High にします。次に光らせたい部分を Low にしたデータ(負論理)を DATA1～4 にセットします。同じようにして②，③，④の LED を光らせます。あとはこれを繰り返します。

もちろん，瞬間瞬間を見れば最大で 4 個の LED しか点灯していないのですが，人間の目には残像現象という性質があるため，LED が消えてもすぐにはわかりません。で，わからないうちにもう一度同じ LED を点灯すると，その LED が消えたと感じないわけです。

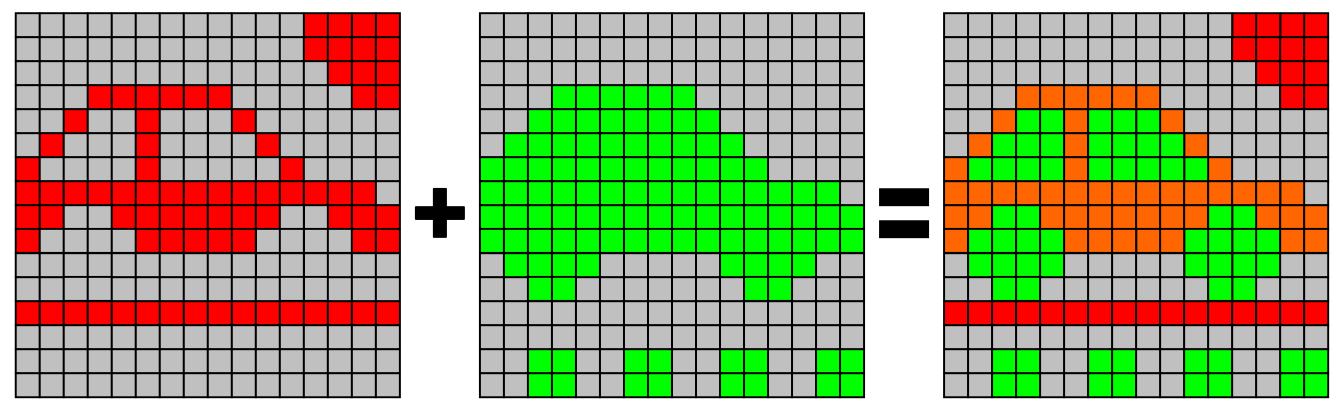
ということで，①→②→③→④→①･･･という切り替えを人間の目で分からないくらいの速さで行なえば，全ての LED が同時に点灯しているように見せかけることができます。

2. イラストを表示しよう

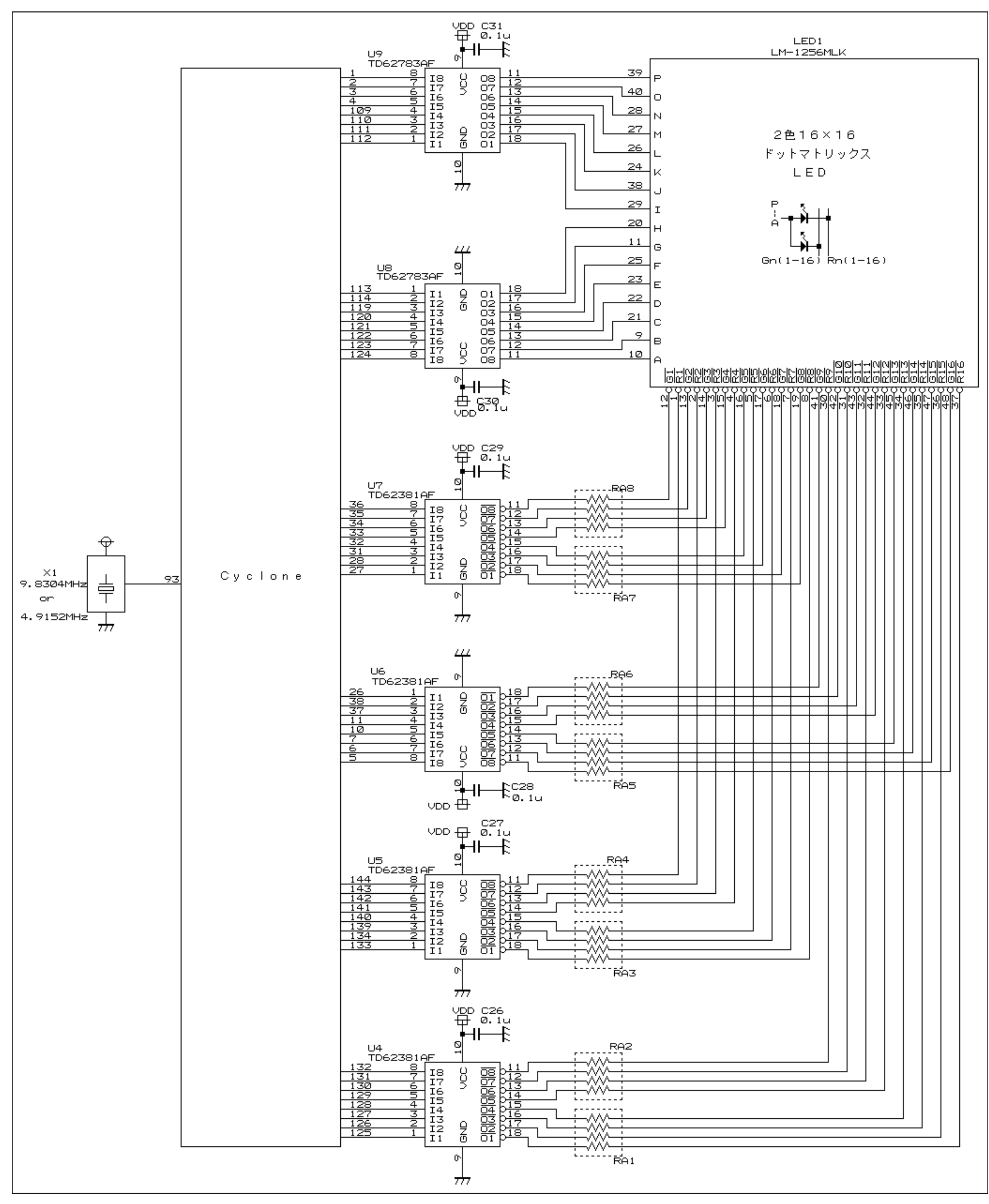
制御方法がわかったところで，実際に LED にイラストを表示してみましょう。最初ですから，一枚の静止画です。次のようなイラストを表示します。



このうち橙は赤と緑の両方を点灯します。



さて，LED の部分の回路図は次のようになります。



フォルダ名は「matrix_led_vhdl」，プロジェクト名は「matrix_led_vhdl」，最上位階層のエンティティ名は「matrix_led_vhdl_top」とします。今までどおりプロジェクトを作成し，「matrix_led_vhdl_top．Vhd」をエディタで開いて下さい。そして，次のソースリストを入力・コンパイル・実行してみましょう。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- 2Color 16x16 Dot Matrix LED Display Module Test
-----------------------------------------------------------
-- File   : matrix_led_vhdl_top.vhd
-- Date   : 2007-09-06
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity matrix_led_vhdl_top is
    port(
        CLK          :   in std_logic;
        LED_SEL      :   out std_logic_vector(15 downto 0);
        LED_DATA_R   :   out std_logic_vector(0 to 15);
        LED_DATA_G   :   out std_logic_vector(0 to 15)
    );
end matrix_led_vhdl_top;

architecture RTL of matrix_led_vhdl_top is

    signal div_cnt     :   std_logic_vector(31 downto 0);
    signal div_clk     :   std_logic;
    signal scan_cnt    :   std_logic_vector(3 downto 0);

begin

    process(CLK)
    begin
        if(CLK'event and CLK='1') then
            div_cnt <= div_cnt + 1;
        end if;
    end process;
    div_clk <= div_cnt(11);

    process(div_clk)
    begin
        if(div_clk'event and div_clk='1') then
            scan_cnt <= scan_cnt + 1;
        end if;
    end process;
```

CLKを分周する スキャン用のタイミングを作る

スキャンカウンタをインクリメントする

```
process(scan_cnt)
begin
     case scan_cnt is
          when “1111” =>
               LED_DATA_R <= “0000000000001111”;
               LED_DATA_G <= “0000000000000000”;
               LED_SEL    <= “1000000000000000”;
          when “1110” =>
               LED_DATA_R <= “0000000000001111”;
               LED_DATA_G <= “0000000000000000”;
               LED_SEL    <= “0100000000000000”;
          when “1101” =>
               LED_DATA_R <= “0000000000000111”;
               LED_DATA_G <= “0000000000000000”;
               LED_SEL    <= “0010000000000000”;
          when “1100” =>
               LED_DATA_R <= “0001111110000011”;
               LED_DATA_G <= “0001111110000000”;
               LED_SEL    <= “0001000000000000”;
          when “1011” =>
               LED_DATA_R <= “0010010001000000”;
               LED_DATA_G <= “0011111111000000”;
               LED_SEL    <= “0000100000000000”;
          when “1010” =>
               LED_DATA_R <= “0100010000100000”;
               LED_DATA_G <= “0111111111100000”;
               LED_SEL    <= “0000010000000000”;
          when “1001” =>
               LED_DATA_R <= “1000010000010000”;
               LED_DATA_G <= “1111111111110000”;
               LED_SEL    <= “0000001000000000”;
          when “1000” =>
               LED_DATA_R <= “1111111111111110”;
               LED_DATA_G <= “1111111111111110”;
               LED_SEL    <= “0000000100000000”;
          when “0111” =>
               LED_DATA_R <= “1100111111100111”;
               LED_DATA_G <= “1111111111111111”;
               LED_SEL    <= “0000000010000000”;
          when “0110” =>
               LED_DATA_R <= “1000011111000011”;
               LED_DATA_G <= “1111111111111111”;
               LED_SEL    <= “0000000001000000”;
          when “0101” =>
               LED_DATA_R <= “0000000000000000”;
               LED_DATA_G <= “0111100000111100”;
               LED_SEL    <= “0000000000100000”;
          when “0100” =>
               LED_DATA_R <= “0000000000000000”;
               LED_DATA_G <= “0011000000011000”;
               LED_SEL    <= “0000000000010000”;
          when “0011” =>
```

スキャンカウンタの値によって，表示データとスキャン信号を出力する

```
                    LED_DATA_R <= “1111111111111111”;
                    LED_DATA_G <= “0000000000000000”;
                    LED_SEL    <= “0000000000001000”;
               when “0010” =>
                    LED_DATA_R <= “0000000000000000”;
                    LED_DATA_G <= “0000000000000000”;
                    LED_SEL    <= “0000000000000100”;
               when “0001” =>
                    LED_DATA_R <= “0000000000000000”;
                    LED_DATA_G <= “0011001100110011”;
                    LED_SEL    <= “0000000000000010”;
               when “0000” =>
                    LED_DATA_R <= “0000000000000000”;
                    LED_DATA_G <= “0011001100110011”;
                    LED_SEL    <= “0000000000000001”;
               when others =>
                    LED_DATA_R <= “0000000000000000”;
                    LED_DATA_G <= “0000000000000000”;
                    LED_SEL    <= “0000000000000000”;
          end case;
     end process;

end RTL;
```

（前からの続き）スキャンカウンタの値によって，表示データとスキャン信号を出力する

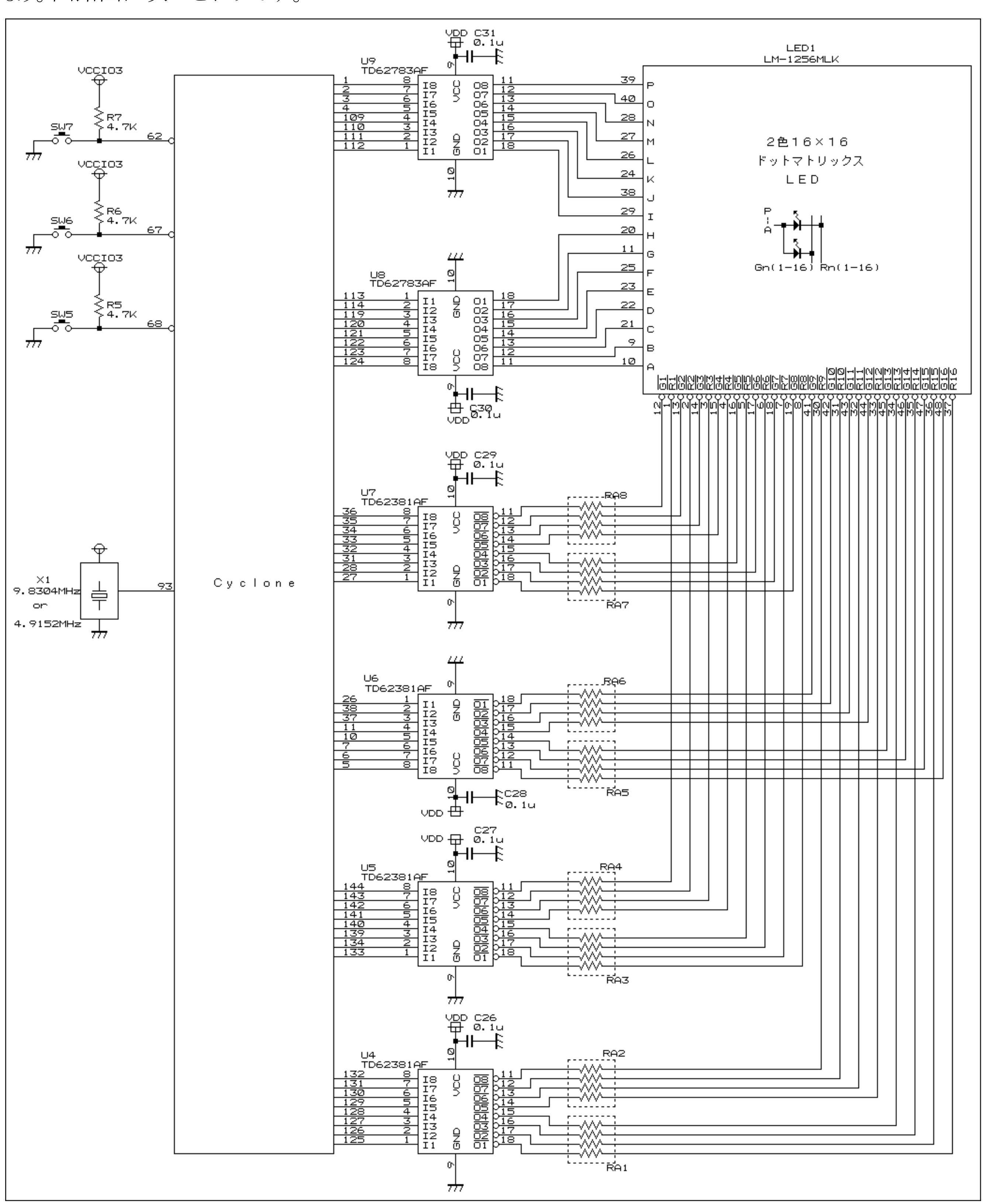
リストが長いので複雑に見えますが，今まで出てきた表現ばかりです。それほど難しくはないですよね。

■ 練習問題

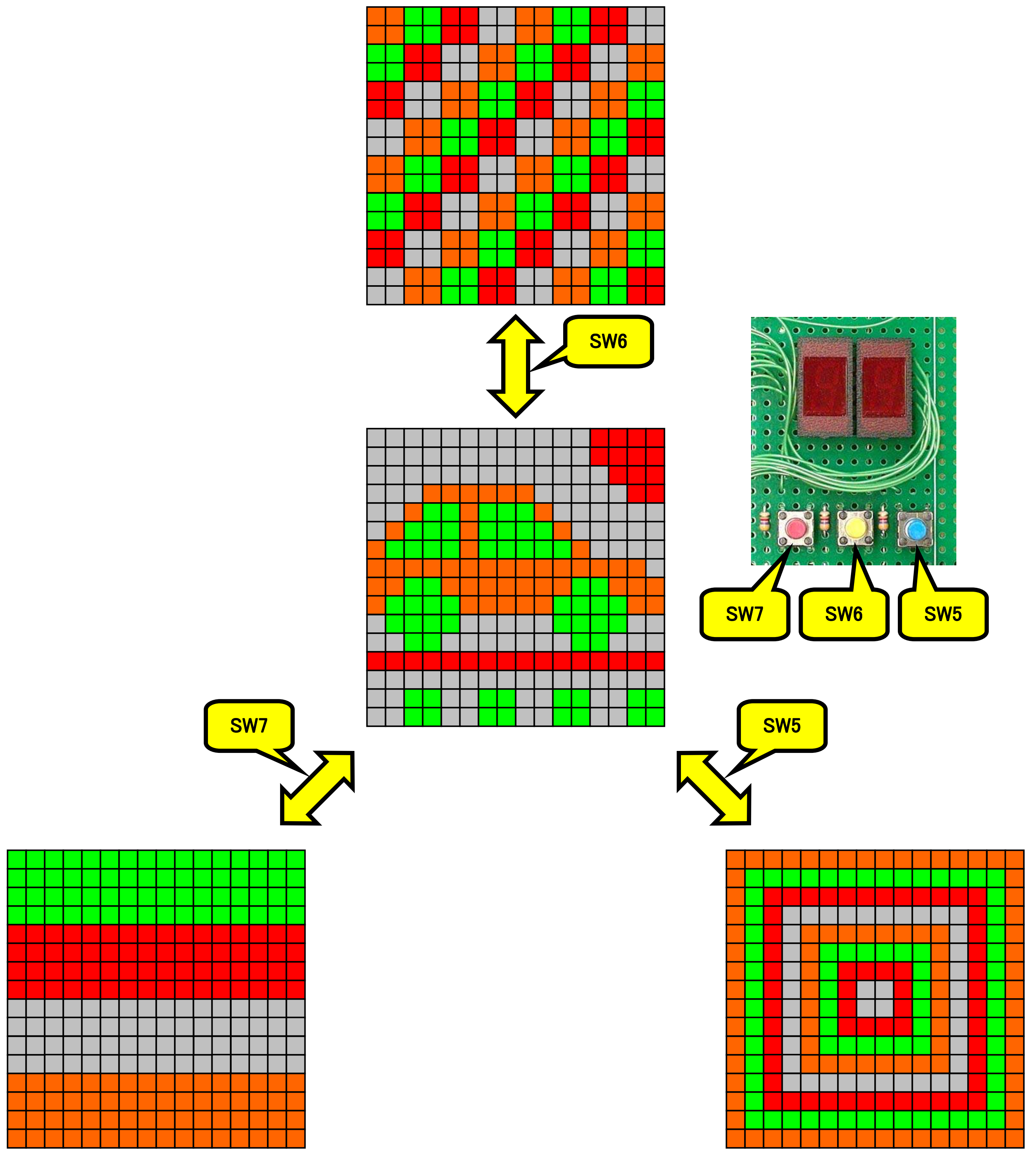
いろいろなイラストを表示してみましょう。

3. スイッチで表示を切り替える

前項でドットマトリックス LED に表示できるようになりましたが，これだとほかの表示にすることができません。それで，スイッチの状態でドットマトリックス LED の表示を切り替える回路を考えてみましょう。回路図は次のとおりです。



SW5，6，7 が押されている間，次のようにイラストを切り替えます。スイッチが複数押されたときの優先順位は「SW5＞SW6＞SW7」にしましょう。何も押されていないときは中央のイラストを表示します。



フォルダ名は「sw_matrix_led_vhdl」，プロジェクト名は「sw_matrix_led_vhdl」，最上位階層のエンティティ名は「sw_matrix_led_vhdl_top」とします。今までどおりプロジェクトを作成し，「sw_matrix_led_vhdl_top. Vhd」をエディタで開いて下さい。そして，次のソースリストを入力・コンパイル・実行してみましょう。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- 2Color 16x16 Dot Matrix LED Display Module Test
-----------------------------------------------------------
-- File   : sw_matrix_led_vhdl_top.vhd
-- Date   : 2007-09-07
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity sw_matrix_led_vhdl_top is
    port(
        CLK         :   in std_logic;
        LED_SEL     :   out std_logic_vector(15 downto 0);
        LED_DATA_R  :   out std_logic_vector(0 to 15);
        LED_DATA_G  :   out std_logic_vector(0 to 15);
        SW          :   in std_logic_vector(7 downto 5)
    );
end sw_matrix_led_vhdl_top;

architecture RTL of sw_matrix_led_vhdl_top is

    signal div_cnt    :   std_logic_vector(31 downto 0);
    signal div_clk    :   std_logic;
    signal scan_cnt   :   std_logic_vector(3 downto 0);

    signal led_reg_0  :   std_logic_vector(0 to 31);  -- Green 0-15,Red 16-31
    signal led_reg_1  :   std_logic_vector(0 to 31);
    signal led_reg_2  :   std_logic_vector(0 to 31);
    signal led_reg_3  :   std_logic_vector(0 to 31);
    signal led_reg_4  :   std_logic_vector(0 to 31);
    signal led_reg_5  :   std_logic_vector(0 to 31);
    signal led_reg_6  :   std_logic_vector(0 to 31);
    signal led_reg_7  :   std_logic_vector(0 to 31);
    signal led_reg_8  :   std_logic_vector(0 to 31);
    signal led_reg_9  :   std_logic_vector(0 to 31);
    signal led_reg_A  :   std_logic_vector(0 to 31);
    signal led_reg_B  :   std_logic_vector(0 to 31);
    signal led_reg_C  :   std_logic_vector(0 to 31);
    signal led_reg_D  :   std_logic_vector(0 to 31);
```

```
    signal led_reg_E   :   std_logic_vector(0 to 31);
    signal led_reg_F   :   std_logic_vector(0 to 31);

begin
    process(SW)
    begin
        if(SW(5)='0') then
            --------------------Green-----------------Red---------
            led_reg_F <= “1111111111111111” & “1111111111111111”;
            led_reg_E <= “1111111111111111” & “1000000000000001”;
            led_reg_D <= “1100000000000011” & “1011111111111101”;
            led_reg_C <= “1100000000000011” & “1010000000000101”;
            led_reg_B <= “1100111111110011” & “1010111111110101”;
            led_reg_A <= “1100111111110011” & “1010100000010101”;
            led_reg_9 <= “1100110000110011” & “1010101111010101”;
            led_reg_8 <= “1100110000110011” & “1010101001010101”;
            led_reg_7 <= “1100110000110011” & “1010101001010101”;
            led_reg_6 <= “1100110000110011” & “1010101111010101”;
            led_reg_5 <= “1100111111110011” & “1010100000010101”;
            led_reg_4 <= “1100111111110011” & “1010111111110101”;
            led_reg_3 <= “1100000000000011” & “1010000000000101”;
            led_reg_2 <= “1100000000000011” & “1011111111111101”;
            led_reg_1 <= “1111111111111111” & “1000000000000001”;
            led_reg_0 <= “1111111111111111” & “1111111111111111”;
        elsif(SW(6)='0') then
            --------------------Green-----------------Red---------
            led_reg_F <= “1111000011110000” & “1100110011001100”;
            led_reg_E <= “1111000011110000” & “1100110011001100”;
            led_reg_D <= “1100001111000011” & “0011001100110011”;
            led_reg_C <= “1100001111000011” & “0011001100110011”;
            led_reg_B <= “0000111100001111” & “1100110011001100”;
            led_reg_A <= “0000111100001111” & “1100110011001100”;
            led_reg_9 <= “0011110000111100” & “0011001100110011”;
            led_reg_8 <= “0011110000111100” & “0011001100110011”;
            led_reg_7 <= “1111000011110000” & “1100110011001100”;
            led_reg_6 <= “1111000011110000” & “1100110011001100”;
            led_reg_5 <= “1100001111000011” & “0011001100110011”;
            led_reg_4 <= “1100001111000011” & “0011001100110011”;
            led_reg_3 <= “0000111100001111” & “1100110011001100”;
            led_reg_2 <= “0000111100001111” & “1100110011001100”;
            led_reg_1 <= “0011110000111100” & “0011001100110011”;
            led_reg_0 <= “0011110000111100” & “0011001100110011”;
        elsif(SW(7)='0') then
            --------------------Green-----------------Red---------
            led_reg_F <= “1111111111111111” & “0000000000000000”;
            led_reg_E <= “1111111111111111” & “0000000000000000”;
            led_reg_D <= “1111111111111111” & “0000000000000000”;
            led_reg_C <= “1111111111111111” & “0000000000000000”;
            led_reg_B <= “0000000000000000” & “1111111111111111”;
            led_reg_A <= “0000000000000000” & “1111111111111111”;
            led_reg_9 <= “0000000000000000” & “1111111111111111”;
            led_reg_8 <= “0000000000000000” & “1111111111111111”;
            led_reg_7 <= “0000000000000000” & “0000000000000000”;
```

この「＆」は連結子です。上位 16 ビットと下位 16 ビットを連結して合計 32 ビットにしています。

```
            led_reg_6 <= “0000000000000000” & “0000000000000000”;
            led_reg_5 <= “0000000000000000” & “0000000000000000”;
            led_reg_4 <= “0000000000000000” & “0000000000000000”;
            led_reg_3 <= “1111111111111111” & “1111111111111111”;
            led_reg_2 <= “1111111111111111” & “1111111111111111”;
            led_reg_1 <= “1111111111111111” & “1111111111111111”;
            led_reg_0 <= “1111111111111111” & “1111111111111111”;
        else
            -------------------Green----------------Red---------
            led_reg_F <= “0000000000000000” & “0000000000001111”;
            led_reg_E <= “0000000000000000” & “0000000000001111”;
            led_reg_D <= “0000000000000000” & “0000000000000111”;
            led_reg_C <= “0001111110000000” & “0001111110000011”;
            led_reg_B <= “0011111111000000” & “0010010001000000”;
            led_reg_A <= “0111111111100000” & “0100010000100000”;
            led_reg_9 <= “1111111111110000” & “1000010000010000”;
            led_reg_8 <= “1111111111111110” & “1111111111111110”;
            led_reg_7 <= “1111111111111111” & “1100111111100111”;
            led_reg_6 <= “1111111111111111” & “1000011111000011”;
            led_reg_5 <= “0111100000111100” & “0000000000000000”;
            led_reg_4 <= “0011000000011000” & “0000000000000000”;
            led_reg_3 <= “0000000000000000” & “1111111111111111”;
            led_reg_2 <= “0000000000000000” & “0000000000000000”;
            led_reg_1 <= “0011001100110011” & “0000000000000000”;
            led_reg_0 <= “0011001100110011” & “0000000000000000”;
        end if;
    end process;


    process(CLK)
    begin
        if(CLK’event and CLK=’1’) then
            div_cnt <= div_cnt + 1;
        end if;
    end process;
    div_clk <= div_cnt(11);

    process(div_clk)
    begin
        if(div_clk’event and div_clk=’1’) then
            scan_cnt <= scan_cnt + 1;
        end if;
    end process;

    process(scan_cnt)
    begin
        case scan_cnt is
            when “1111” =>
                LED_DATA_R <= led_reg_F(16 to 31);
                LED_DATA_G <= led_reg_F( 0 to 15);
                LED_SEL    <= “1000000000000000”;
            when “1110” =>
                LED_DATA_R <= led_reg_E(16 to 31);
```

LED_DATA_G(0 to 15)に led_reg_F(0 to 15)を，LED_DATA_R(0 to 15)に led_reg_F(16 to 31)をつなぎます。このようにベクタタイプの信号の一部をスライスすることができます。

```
            LED_DATA_G <= led_reg_E( 0 to 15);
            LED_SEL    <= "0100000000000000";
        when "1101" =>
            LED_DATA_R <= led_reg_D(16 to 31);
            LED_DATA_G <= led_reg_D( 0 to 15);
            LED_SEL    <= "0010000000000000";
        when "1100" =>
            LED_DATA_R <= led_reg_C(16 to 31);
            LED_DATA_G <= led_reg_C( 0 to 15);
            LED_SEL    <= "0001000000000000";
        when "1011" =>
            LED_DATA_R <= led_reg_B(16 to 31);
            LED_DATA_G <= led_reg_B( 0 to 15);
            LED_SEL    <= "0000100000000000";
        when "1010" =>
            LED_DATA_R <= led_reg_A(16 to 31);
            LED_DATA_G <= led_reg_A( 0 to 15);
            LED_SEL    <= "0000010000000000";
        when "1001" =>
            LED_DATA_R <= led_reg_9(16 to 31);
            LED_DATA_G <= led_reg_9( 0 to 15);
            LED_SEL    <= "0000001000000000";
        when "1000" =>
            LED_DATA_R <= led_reg_8(16 to 31);
            LED_DATA_G <= led_reg_8( 0 to 15);
            LED_SEL    <= "0000000100000000";
        when "0111" =>
            LED_DATA_R <= led_reg_7(16 to 31);
            LED_DATA_G <= led_reg_7( 0 to 15);
            LED_SEL    <= "0000000010000000";
        when "0110" =>
            LED_DATA_R <= led_reg_6(16 to 31);
            LED_DATA_G <= led_reg_6( 0 to 15);
            LED_SEL    <= "0000000001000000";
        when "0101" =>
            LED_DATA_R <= led_reg_5(16 to 31);
            LED_DATA_G <= led_reg_5( 0 to 15);
            LED_SEL    <= "0000000000100000";
        when "0100" =>
            LED_DATA_R <= led_reg_4(16 to 31);
            LED_DATA_G <= led_reg_4( 0 to 15);
            LED_SEL    <= "0000000000010000";
        when "0011" =>
            LED_DATA_R <= led_reg_3(16 to 31);
            LED_DATA_G <= led_reg_3( 0 to 15);
            LED_SEL    <= "0000000000001000";
        when "0010" =>
            LED_DATA_R <= led_reg_2(16 to 31);
            LED_DATA_G <= led_reg_2( 0 to 15);
            LED_SEL    <= "0000000000000100";
        when "0001" =>
            LED_DATA_R <= led_reg_1(16 to 31);
            LED_DATA_G <= led_reg_1( 0 to 15);
```

```
                    LED_SEL     <= “0000000000000010”;
               when “0000” =>
                    LED_DATA_R <= led_reg_0(16 to 31);
                    LED_DATA_G <= led_reg_0( 0 to 15);
                    LED_SEL     <= “0000000000000001”;
               when others =>
                    LED_DATA_R <= “0000000000000000”;
                    LED_DATA_G <= “0000000000000000”;
                    LED_SEL     <= “0000000000000000”;
          end case;
     end process;

end RTL;
```

かなり長くなりましたが，ほとんどは表示データです。がんばって入力してみてください。ちゃんと切り替わりましたか。

第7章 マイコンと組み合わせて使ってみよう

1. マイコンとFPGAの比較
2. マイコンから7セグメントLEDに表示しよう(回路図入力)
3. 同じ回路をVHDLで入力する
4. シフトレジスタを利用する

前書きのところでマイコンには避けられない弱点があると書きました。しかし，FPGA にも弱点があり，その弱点の部分がマイコンにとっては長所になっていることもあります。それで，マイコンとFPGA を組み合わせ，それぞれの長所を引き出すことで，より効率的で高機能なシステムを作ることができます。この章では，マイコンとFPGAを組み合わせて使う基本的な方法を考えてみましょう。

1. マイコンとFPGAの比較

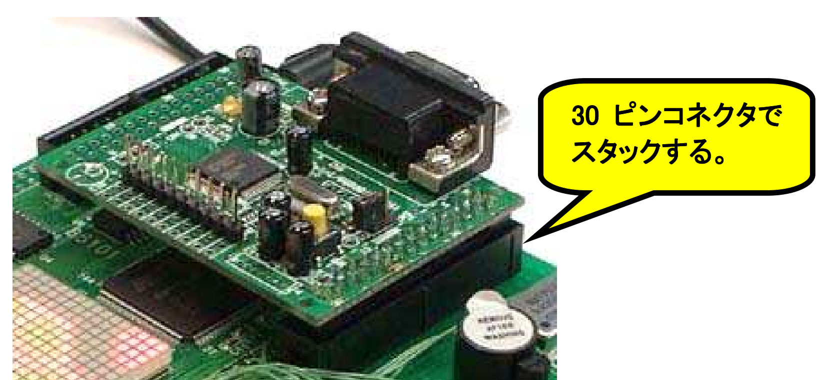
まずは，それぞれの得意な分野と苦手な分野をまとめてみましょう。黄色でマーキングしているところが得意な部分です。

	マイコン	FPGA
I/O	ほとんどが兼用ピンで，内蔵機能の使い方によって使えないI/Oが多い。	全てのI/Oは基本的に同じ機能であり，自由に使うことができる。
内蔵機能	いろいろな機能があらかじめ組み込まれていて便利。無い機能は外部に用意しない限り使えない。	基本的に何も組み込まれていない。全部自分で作らなければならないが，回路サイズが許す限り，なんでも自由に組み込むことができる。
並列処理	苦手。マルチタスクや割り込み処理で，擬似的に並列処理しているように見せかけることはできる。	得意。組み込まれた回路は全て並列処理される。
乗算・除算	C言語なら命令として用意されており，アセンブラでも用意されていることが多く，普通に実行できる。	苦手。複雑なアルゴリズムを組み込もうとするとすぐに回路サイズがパンクして入りきらなくなる。
多量データの記憶	メモリサイズが許すかぎり，いくらでも記憶しておくことができる。	苦手。あまりに無駄が多い。
処理スピード	命令を一つずつ順番に解析して実行するため，専用ハードウェアに比べれば遅い。割り込み処理に移るまでに一定の時間が必要なため，信号の変化に対する反応が遅い。	ハードウェアで即反応するため非常に高速。
記述言語	アセンブラやC言語が主流。特にアセンブラはマイコンが変わると命令を一から勉強しなおす必要がある。	VHDL や Verilog-HDL が主流。FPGA が変わっても文法は変わらないので移行しやすい。

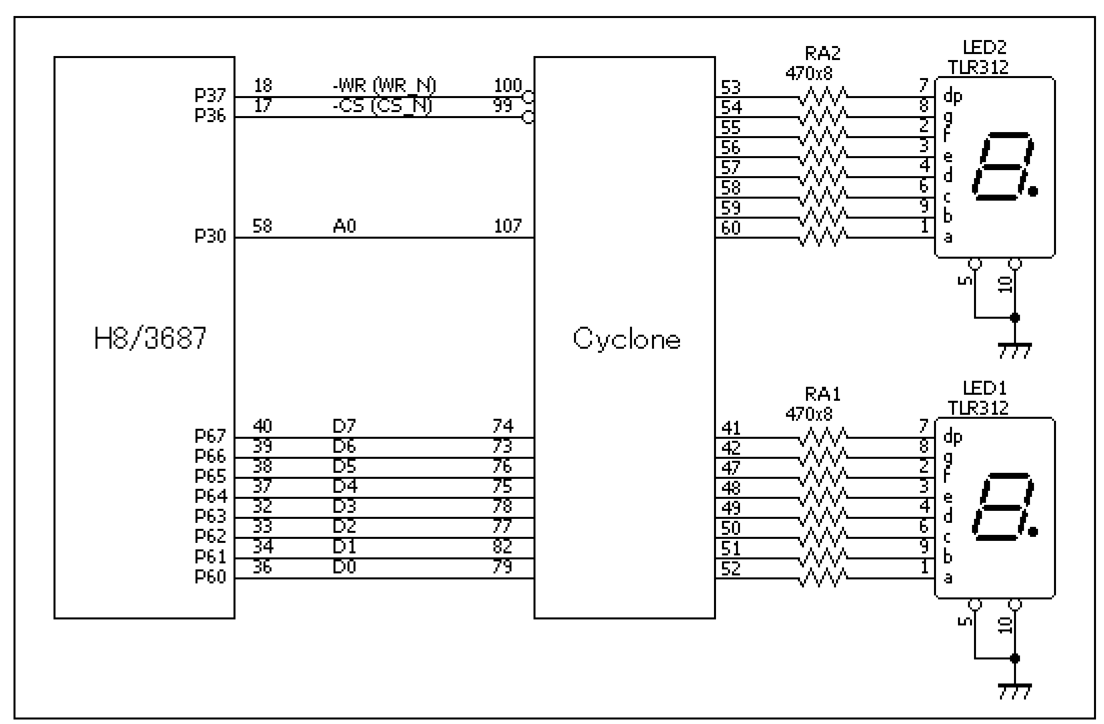
上の表からわかるように，乗算や除算を含んでいたり，複雑なアルゴリズムを処理したりするところは，マイコンで行なう方が効果的です。また，多量のデータを記憶しておくところもマイコンを採用したほうが効率がよいでしょう。一方，厳密な並列処理が必要な部分，超高速処理が必要な部分は，FPGA の出番です。また，マイコンだけでは足りない I/O の数を補ったり，単純な繰り返し処理(ダイナミック表示のためのスキャンなど)をマイコンから切り離すためにFPGAに置き換えたり，というのもありです。

2. マイコンから 7 セグメント LED に表示しよう（回路図入力）

では，簡単な例として，マイコンから FPGA に 7 セグメント LED に表示するデータをセットし，表示自体は FPGA で行なうことを考えてみましょう。写真のように TK-3687mini を FPGA ボードに接続してください。



回路図は次のように考えました。

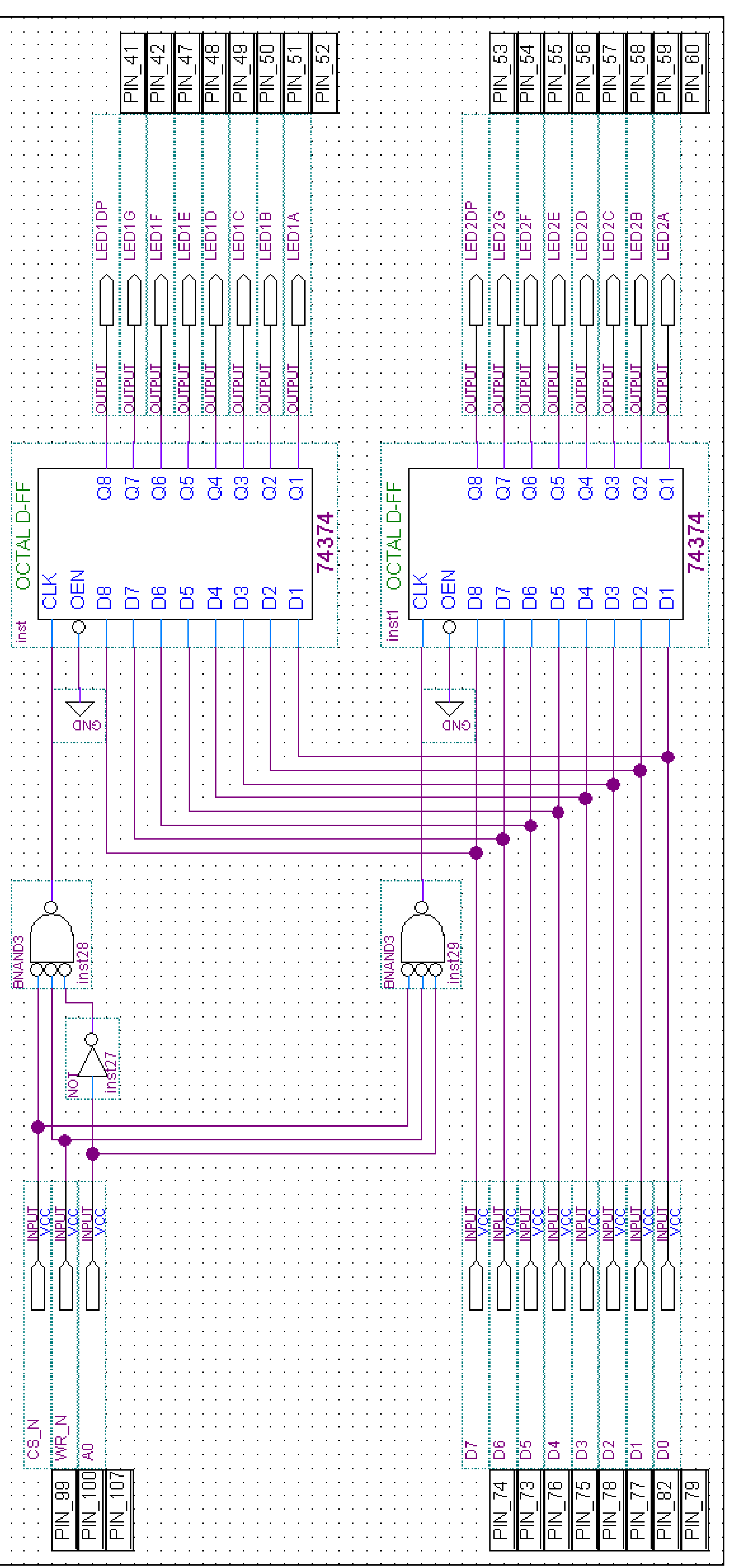


次に，Cyclone の中身を設計することになります。真理値表は次のようにします。

入力				出力	
A0	-CS（CS_N）	-WR（WR_N）	D0～D7	LED2	LED1
0	0	↑	D0～D7	D0～D7	変化なし
1	0	↑	D0～D7	変化なし	D0～D7
X	1	X	X	変化なし	変化なし
X	X	0	X	変化なし	変化なし
X	X	1	X	変化なし	変化なし

簡単に言えば，8 ビットのフリップフロップを二つ用意し，A0 でどちらのフリップフロップかを選択，WR_N の立ち上がりでラッチする，というものです。

この真理値表から回路を考えてみました。フォルダ名は「micom_7seg_sch」，プロジェクト名は「micom_7seg_sch」，最上位階層のエンティティ名は「micom_7seg_sch_top」とします。今までどおりプロジェクトを作成し，「micom_7seg_sch_top．bdf」をエディタで開いて下さい。そして，次の回路図を入力・コンパイル・実行してみましょう。



実行しても何も表示されませんね。当然です。TK-3687miniのプログラムでCycloneに表示データをセットしないと何も表示されません。そこで，TK-3687mini にプログラムをダウンロードして実行しましょう。TK-3687mini にはハイパーH8 が書き込まれています。ハイパーH8 を使ったプログラムの実行方法は，CD 内の「マイコン事始めキット」のマニュアルをご覧ください。「prog01_7seg. mot」をダウンロード・実行してみましょう。ソースリストは次のとおりです。

```
/**********************************************************************/
/*                                                                    */
/*  FILE        :prog01_7seg.c                                        */
/*  DATE        :Tue, Nov 13, 2007                                    */
/*  DESCRIPTION :Main Program                                         */
/*  CPU TYPE    :H8/3687                                              */
/*                                                                    */
/*  This file is ·rogrammed by TOYO-LINX Co.,Ltd. / yKikuchi           */
/*                                                                    */
/**********************************************************************/

/**********************************************************************
     インクルードファイル
**********************************************************************/
#include <machine.h>   // H8 特有の命令を使う
#include “iodefine.h” // 内蔵 I/O のラベル定義

/**********************************************************************
     定数エリアの定義(ROM)
**********************************************************************/
//表示データ
const unsigned char SegLedData[][2] = {
     {0x01,0x00},{0x02,0x00},{0x04,0x00},{0x08,0x00},   // 0, 1, 2, 3
     {0x10,0x00},{0x20,0x00},{0x01,0x00},{0x00,0x01},   // 4, 5, 6, 7
     {0x00,0x02},{0x00,0x04},{0x00,0x08},{0x00,0x10},   // 8, 9,10,11
     {0x00,0x20},{0x00,0x01},{0x00,0x02},{0x00,0x04},   //12,13,14,15
     {0x00,0x08},{0x08,0x00},{0x10,0x00},{0x40,0x00},   //16,17,18,19
     {0x00,0x40},{0x00,0x02},{0x00,0x01},{0x01,0x00},   //20,21,22,23
     {0x20,0x00},{0x40,0x00},{0x00,0x40},{0x00,0x04},   //24,25,26,27
     {0x00,0x08},{0x08,0x00},{0x10,0x00},{0x20,0x00}    //28,29,30,31
};

/**********************************************************************
     グローバル変数の定義とイニシャライズ(RAM)
**********************************************************************/
// LED 表示に関係した変数 ----------------------------------------------
unsigned long SegLedBuf[2]; //表示バッファ

/**********************************************************************
     関数の定義
**********************************************************************/
void     init_io(void);
void     main(void);
void     set_7segLED_cont(void);
void     wait(void);
```

```
/*************************************************************************
    メインプログラム
*************************************************************************/
void main(void)
{
    unsigned char I;

    init_io();    // I/O ポートイニシャライズ

    while(1){
        for (I=0; I<32; I++){
            SegLedBuf[0] = SegLedData[I][0];
            SegLedBuf[1] = SegLedData[I][1];
            set_7segLED_cont();
            wait();
        }
    }
}

/*************************************************************************
    7 セグメント LED コントローラに表示データをセット
*************************************************************************/
void set_7segLED_cont(void)
{
    unsigned char a;

    a = 0x00;//アドレス

    IO.PDR3.BYTE      =    a | (IO.PDR3.BYTE & 0xc0);  //アドレスセット
    IO.PDR3.BIT.B6    =    0;                          //-CS=Low
    IO.PDR6.BYTE      =    SegLedBuf[0];               //データセット
    IO.PDR3.BIT.B7    =    0;                          //-WR=Low
    IO.PDR3.BIT.B7    =    1;                          //-WR=High
    IO.PDR3.BIT.B6    =    1;                          //-CS=High

    a++;

    IO.PDR3.BYTE      =    a | (IO.PDR3.BYTE & 0xc0);  //アドレスセット
    IO.PDR3.BIT.B6    =    0;                          //-CS=Low
    IO.PDR6.BYTE      =    SegLedBuf[1];               //データセット
    IO.PDR3.BIT.B7    =    0;                          //-WR=Low
    IO.PDR3.BIT.B7    =    1;                          //-WR=High
    IO.PDR3.BIT.B6    =    1;                          //-CS=High

}

/*************************************************************************
    I/O ポート イニシャライズ
*************************************************************************/
void init_io(void)
{
    IO.PCR3       =    0xff;     //ポート 3,P30-37 出力
    IO.PDR3.BYTE  =    0xc0;
```

```
	IO.PMR5.BYTE	=	0x00;	//ポート5,汎用入出力ポート
	IO.PUCR5.BYTE	=	0x00;	//ポート5,内蔵プルアップオフ
	IO.PCR5		=	0x80;	//ポート5,P50-56入力,P57出力
	IO.PDR5.BYTE	=	0x00;

	IO.PCR6		=	0xff;	//ポート6,P60-67出力
	IO.PDR6.BYTE	=	0x00;
}

/************************************************************************
	ウェイト
************************************************************************/
void wait(void)
{
	unsigned long l;

	for (l=0;l<400000;l++){}
}
```

いかがでしょうか。今度こそは 7 セグメント LED に，ちょっとしたデモが表示されたはずです。単純に「Seg_Led_Data」で定義したデータを順番にセットしているだけですが，意外と面白い動きだと思いませんか。

3. 同じ回路を VHDL で入力する

フォルダ名は「micom_7seg_vhdl」，プロジェクト名は「micom_7seg_vhdl」，最上位階層のエンティティ名は「micom_7seg_vhdl_top」とします。今までどおりプロジェクトを作成し，「micom_7seg_vhdl_top.vhd」をエディタで開いて下さい。そして，次のソースリストを入力・コンパイル・実行してみましょう。

```
----------------------------------------------------------
-- FPGA Training Board(B6101)
-- 7Segment LED Controller
----------------------------------------------------------
-- File   : micom_7seg_vhdl_top.vhd
-- Date   : 2007-11-14
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity micom_7seg_vhdl_top is
    port(
        D       :   in std_logic_vector(7 downto 0);
        A0      :   in std_logic;
        WR_N    :   in std_logic;
        CS_N    :   in std_logic;
        LED1A   :   out std_logic;
        LED1B   :   out std_logic;
        LED1C   :   out std_logic;
        LED1D   :   out std_logic;
        LED1E   :   out std_logic;
        LED1F   :   out std_logic;
        LED1G   :   out std_logic;
        LED1DP  :   out std_logic;
        LED2A   :   out std_logic;
        LED2B   :   out std_logic;
        LED2C   :   out std_logic;
        LED2D   :   out std_logic;
        LED2E   :   out std_logic;
        LED2F   :   out std_logic;
        LED2G   :   out std_logic;
        LED2DP  :   out std_logic
    );
end micom_7seg_vhdl_top;

architecture RTL of micom_7seg_vhdl_top is

    signal led1  :   std_logic_vector(7 downto 0);
    signal led2  :   std_logic_vector(7 downto 0);
```

```
begin
    process(WR_N)
    begin
        if(WR_N'event and WR_N = '1')then
            if(CS_N = '0')then
                if(A0 = '0')then
                    led2 <= D;
                else
                    led1 <= D;
                end if;
            end if;
        end if;
    end process;

    LED1A  <= led1(0);
    LED1B  <= led1(1);
    LED1C  <= led1(2);
    LED1D  <= led1(3);
    LED1E  <= led1(4);
    LED1F  <= led1(5);
    LED1G  <= led1(6);
    LED1DP <= led1(7);
    LED2A  <= led2(0);
    LED2B  <= led2(1);
    LED2C  <= led2(2);
    LED2D  <= led2(3);
    LED2E  <= led2(4);
    LED2F  <= led2(5);
    LED2G  <= led2(6);
    LED2DP <= led2(7);

end RTL;
```

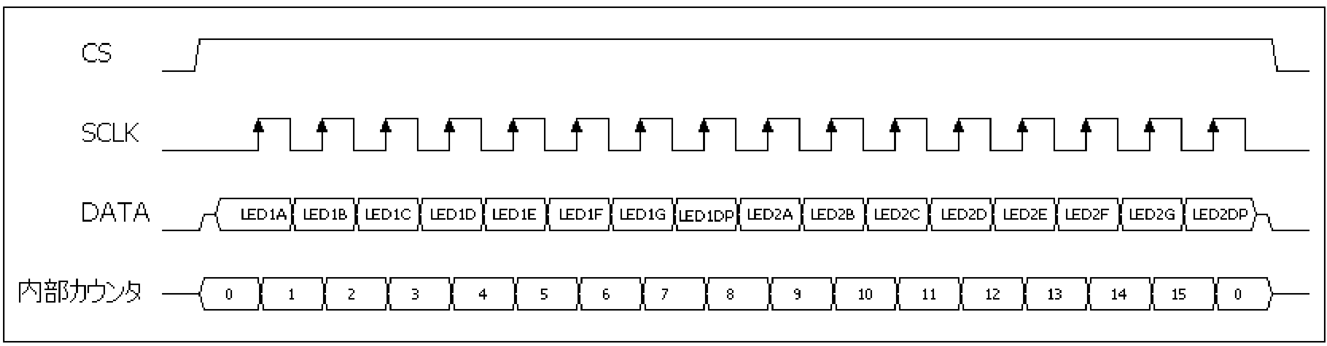
TK-3687mini のプログラムは同じです。ダウンロードして実行してみてください。先ほどと同じデモが始まります。

4. シフトレジスタを利用する

これで，マイコンとFPGAを組み合わせて7セグメントLEDに表示することができました。今回は7セグメントLEDも2個だけですし，H8/3687は比較的I/Oが豊富なので，別にFPGAを組み合わせなくても，と思うかもしれません。しかし，LEDや他の出力が何ビットにもなったり，マイコンのI/Oが非常に限られていたりすると，この方法の価値が出てきます。

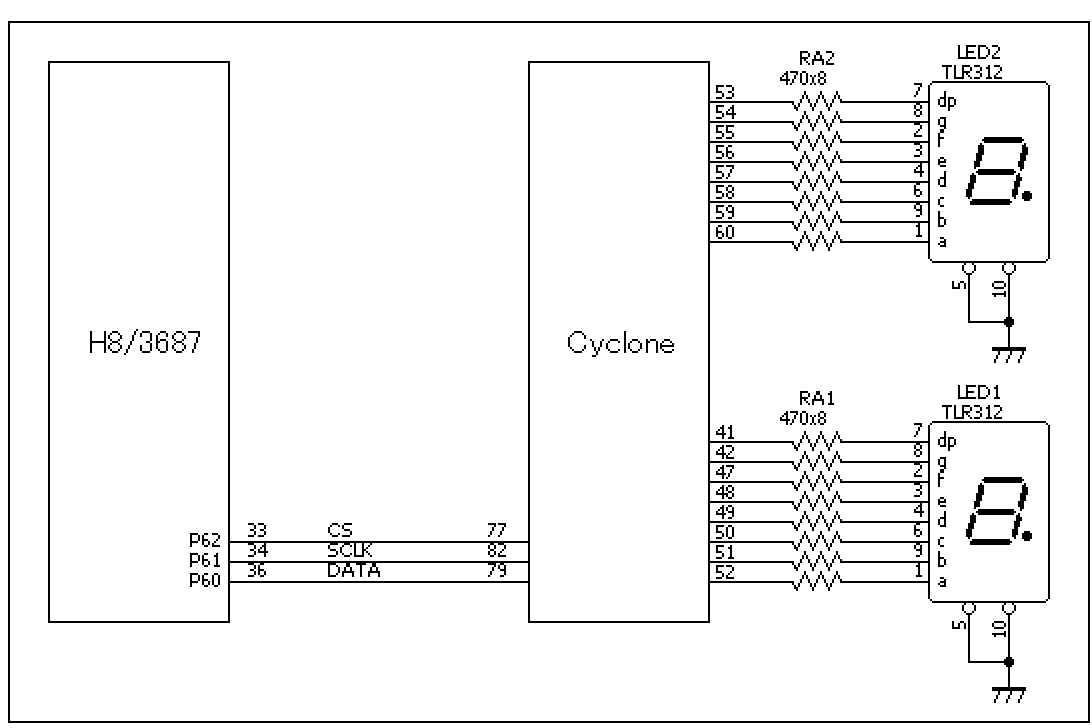
でも，よく考えてみると，今回16個のLED表示のためにマイコンが使用したI/Oは11ビットです。まあ，今回は実習が目的なのでよしとする，というふうに流してもよいのですが，もうちょっと何とかしたいな，という気持ちもあります。前項の考え方の基本はパラレルでインターフェースするものでした。それで，もうちょっと何とかするためにシリアルでインターフェースすることを検討してみましょう。

デバイス間のシリアルインターフェースにもいろいろな規格があります（Microwire，SPI，I^{2}Cなど）。ただ，今回は全部で16ビットのデータだけなので独自規格にしました。次のタイミングチャートをご覧ください。

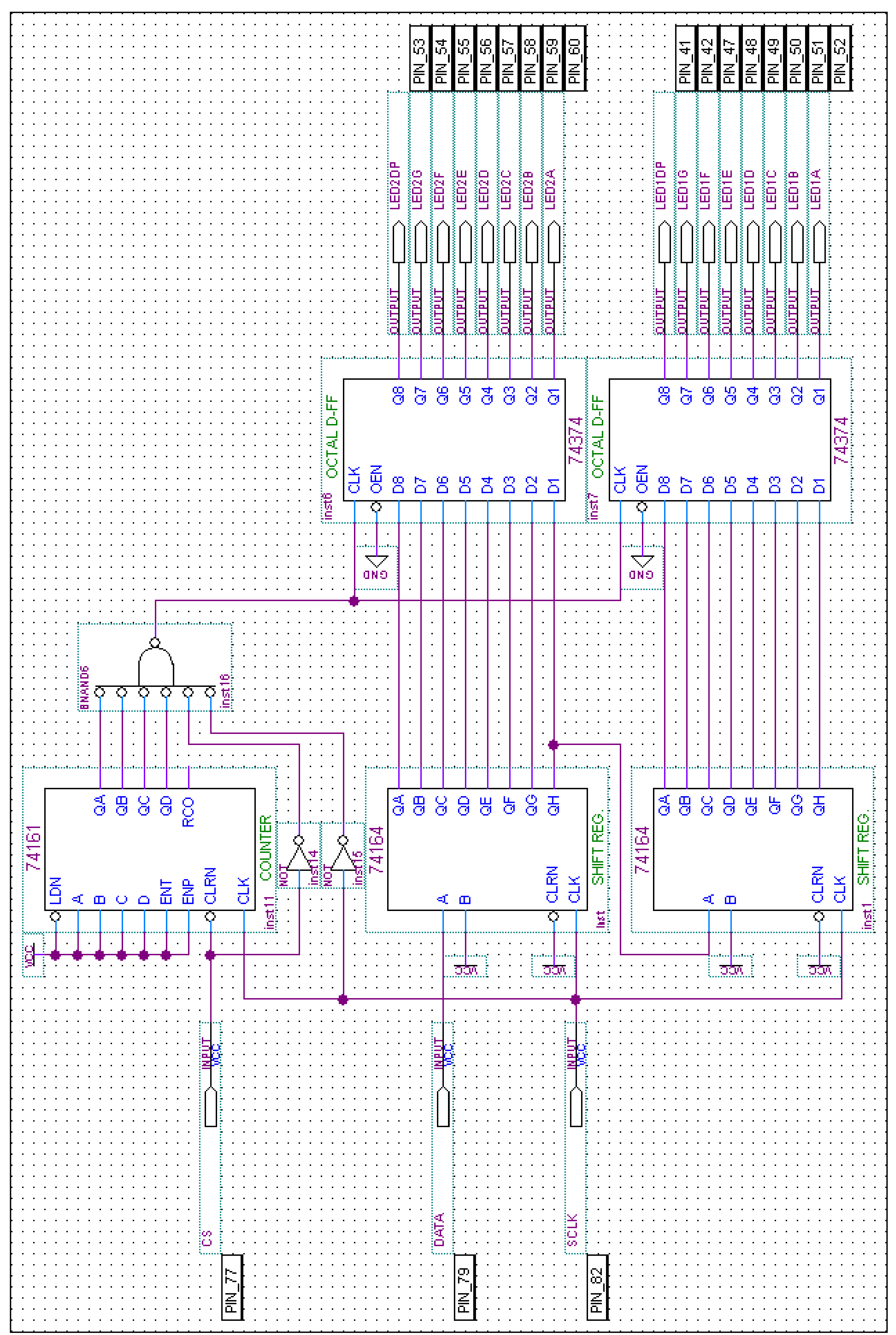


CSをHighにしてからSCLKの立ち上がりでLEDデータを1ビットずつ入力します。内部でビット数をカウントし，16ビット入力したらSCLKの立ち下がりで表示を更新します。

回路図は右のようになります。かなりすっきりしましたね。



では，Cycloneの中身を設計してみましょう。タイミングチャートから検討してシフトレジスタを利用することにします。フォルダ名は「micom_7seg_2_sch」，プロジェクト名は「micom_7seg_2_sch」，最上位階層のエンティティ名は「micom_7seg_2_sch_top」とします。今までどおりプロジェクトを作成し，「micom_7seg_2_sch_top．bdf」をエディタで開いて下さい。そして，次の回路図を入力・コンパイル・実行してみましょう



PIN_53
PIN_54
PIN_55
PIN_56
PIN_57
PIN_58
PIN_59
PIN_60
PIN_41
PIN_42
PIN_47
PIN_48
PIN_49
PIN_50
PIN_51
PIN_52
LED2DP
LED2G
LED2F
LED2E
LED2D
LED2C
LED2B
LED2A
LED1DP
LED1G
LED1F
LED1E
LED1D
LED1C
LED1B
LED1A
OUTPUT
OCTAL D-FF
74374
inst6
inst7
CLK
OEN
D8
D7
D6
D5
D4
D3
D2
D1
Q8
Q7
Q6
Q5
Q4
Q3
Q2
Q1
GND
8NAND6
inst16
74161
COUNTER
inst11
LDN
A
B
C
D
ENT
ENP
CLRN
QA
QB
QC
QD
RCO
NOT
inst14
inst15
74164
SHIFT REG.
inst
inst1
QE
QF
QG
QH
VCC
INPUT
CS
DATA
SCLK
PIN_77
PIN_79
PIN_82

さて，当然のことながら，TK-3687mini のプログラムも変更しなければなりません。ハイパーH8 で「prog02_7seg. mot」をダウンロード・実行してみましょう。ソースリストは次のとおりです。

```
/**********************************************************************/
/*                                                                    */
/*  FILE        :prog02_7seg.c                                        */
/*  DATE        :Fri, Nov 16, 2007                                    */
/*  DESCRIPTION :Main Program                                         */
/*  CPU TYPE    :H8/3687                                              */
/*                                                                    */
/*  This file is programed by TOYO-LINX Co.,Ltd. / yKikuchi           */
/*                                                                    */
/**********************************************************************/

/**********************************************************************
    インクルードファイル
**********************************************************************/
#include <machine.h>   // H8 特有の命令を使う
#include "iodefine.h"  // 内蔵 I/O のラベル定義

/**********************************************************************
    定数エリアの定義(ROM)
**********************************************************************/
//表示データ
const unsigned char SegLedData[][2] = {
    {0x01,0x00},{0x02,0x00},{0x04,0x00},{0x08,0x00},  // 0, 1, 2, 3
    {0x10,0x00},{0x20,0x00},{0x01,0x00},{0x00,0x01},  // 4, 5, 6, 7
    {0x00,0x02},{0x00,0x04},{0x00,0x08},{0x00,0x10},  // 8, 9,10,11
    {0x00,0x20},{0x00,0x01},{0x00,0x02},{0x00,0x04},  //12,13,14,15
    {0x00,0x08},{0x08,0x00},{0x10,0x00},{0x40,0x00},  //16,17,18,19
    {0x00,0x40},{0x00,0x02},{0x00,0x01},{0x01,0x00},  //20,21,22,23
    {0x20,0x00},{0x40,0x00},{0x00,0x40},{0x00,0x04},  //24,25,26,27
    {0x00,0x08},{0x08,0x00},{0x10,0x00},{0x20,0x00}   //28,29,30,31
};

/**********************************************************************
    グローバル変数の定義とイニシャライズ(RAM)
**********************************************************************/
// LED 表示に関係した変数 -----------------------------------------------
unsigned long SegLedBuf[2]; //表示バッファ

/**********************************************************************
    関数の定義
**********************************************************************/
void    init_io(void);
void    main(void);
void    set_7segLED_cont(void);
void    wait(void);

/**********************************************************************
    メインプログラム
**********************************************************************/
```

```
void main(void)
{
    unsigned char i;

    init_io();    // I/O ポートイニシャライズ

    while(1){
        for (i=0; i<32; i++){
            SegLedBuf[0] = SegLedData[i][0];
            SegLedBuf[1] = SegLedData[i][1];
            set_7segLED_cont();
            wait();
        }
    }
}

/*************************************************************************
    7 セグメント LED コントローラに表示データをセット
*************************************************************************/
void set_7segLED_cont(void)
{
    unsigned char i;
    unsigned int d;

    d = SegLedBuf[0] * 0x0100 + SegLedBuf[1];

    IO.PDR6.BIT.B2 = 1;    //CS = High

    for (i=0; i<16; i++){
        if ((d & 0x0001)==0x0001)  {IO.PDR6.BIT.B0 = 1;}  //DATA Set
        else                       {IO.PDR6.BIT.B0 = 0;}
        IO.PDR6.BIT.B1 = 1;    //SCLK = High
        IO.PDR6.BIT.B1 = 0;    //SCLK = Low
        d = d / 2;    //右シフト
    }

    IO.PDR6.BIT.B2 = 0;    //CS = Low
}

/*************************************************************************
    I/O ポート イニシャライズ
*************************************************************************/
void init_io(void)
{
    IO.PCR6      =    0x07;    //ポート 6,P60-62 出力,P63-67 入力
    IO.PDR6.BYTE =    0x00;
}

/*************************************************************************
    ウェイト
*************************************************************************/
void wait(void)
{
```

```
    unsigned long i;

    for (i=0;i<400000;i++){}
}
```

メインルーチンは一切変更していません。表示データテーブルの構造もあえて変更していません。Cyclone にセットするところ（“set_7seg_LED_cont”ルーチン）だけ変更しました。前項と同じように動きましたか。

■ VHDL に置き換える

次に，回路図入力をVHDL 入力に置き換えてみましょう。フォルダ名は「micom_7seg_2_vhdl」，プロジェクト名は「micom_7seg_2_vhdl」，最上位階層のエンティティ名は「micom_7seg_2_vhdl_top」とします。今までどおりプロジェクトを作成し，「micom_7seg_2_vhdl_top．vhd」をエディタで開いて下さい。そして，次のソースリストを入力・コンパイル・実行してみましょう。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- 7Segment LED Controller
-----------------------------------------------------------
-- File   : micom_7seg_2_vhdl_top.vhd
-- Date   : 2007-11-19
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity micom_7seg_2_vhdl_top is
    port(
        DATA    :   in std_logic;
        SCLK    :   in std_logic;
        CS      :   in std_logic;
        LED1A   :   out std_logic;
        LED1B   :   out std_logic;
        LED1C   :   out std_logic;
        LED1D   :   out std_logic;
        LED1E   :   out std_logic;
        LED1F   :   out std_logic;
        LED1G   :   out std_logic;
        LED1DP  :   out std_logic;
        LED2A   :   out std_logic;
        LED2B   :   out std_logic;
        LED2C   :   out std_logic;
        LED2D   :   out std_logic;
```

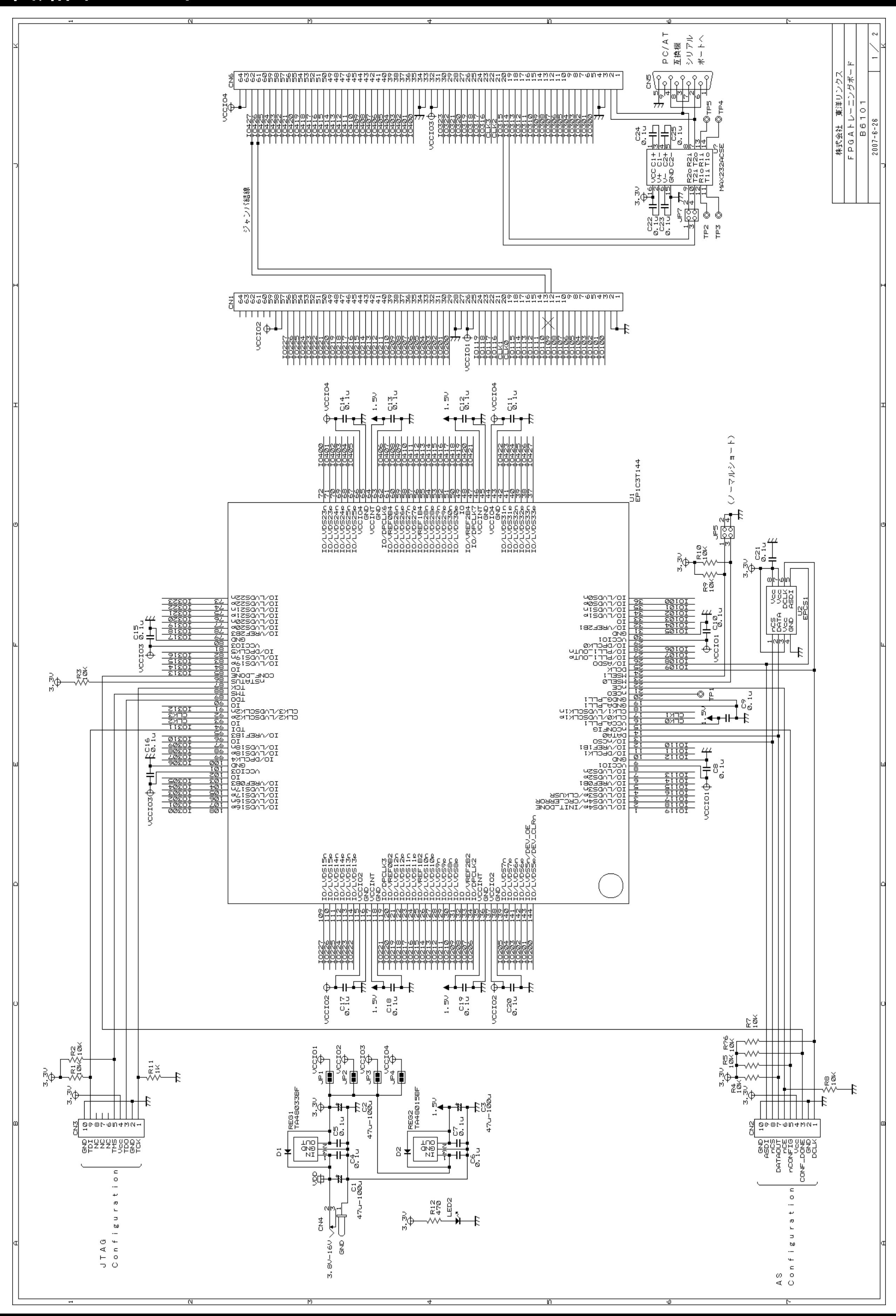
```
        LED2E    :    out std_logic;
        LED2F    :    out std_logic;
        LED2G    :    out std_logic;
        LED2DP   :    out std_logic
    );
end micom_7seg_2_vhdl_top;

architecture RTL of micom_7seg_2_vhdl_top is

    signal led1            :    std_logic_vector(7 downto 0);
    signal led2            :    std_logic_vector(7 downto 0);
    signal counter         :    std_logic_vector(3 downto 0);
    signal shift_reg       :    std_logic_vector(15 downto 0);

begin
    process(SCLK,CS)
    begin
        if(CS='0') then
            counter <= "0000";
        else
            if(SCLK'event and SCLK='1') then
                shift_reg(0)  <= shift_reg(1);
                shift_reg(1)  <= shift_reg(2);
                shift_reg(2)  <= shift_reg(3);
                shift_reg(3)  <= shift_reg(4);
                shift_reg(4)  <= shift_reg(5);
                shift_reg(5)  <= shift_reg(6);
                shift_reg(6)  <= shift_reg(7);
                shift_reg(7)  <= shift_reg(8);
                shift_reg(8)  <= shift_reg(9);
                shift_reg(9)  <= shift_reg(10);
                shift_reg(10) <= shift_reg(11);
                shift_reg(11) <= shift_reg(12);
                shift_reg(12) <= shift_reg(13);
                shift_reg(13) <= shift_reg(14);
                shift_reg(14) <= shift_reg(15);
                shift_reg(15) <= DATA;

                counter <= counter + 1;
            end if;

            if(SCLK'event and SCLK='0') then
                if(counter=0) then
                    led2 <= shift_reg(15 downto 8);
                    led1 <= shift_reg(7 downto 0);
                end if;
            end if;
        end if;
    end process;

    LED1A  <= led1(0);
    LED1B  <= led1(1);
    LED1C  <= led1(2);
```

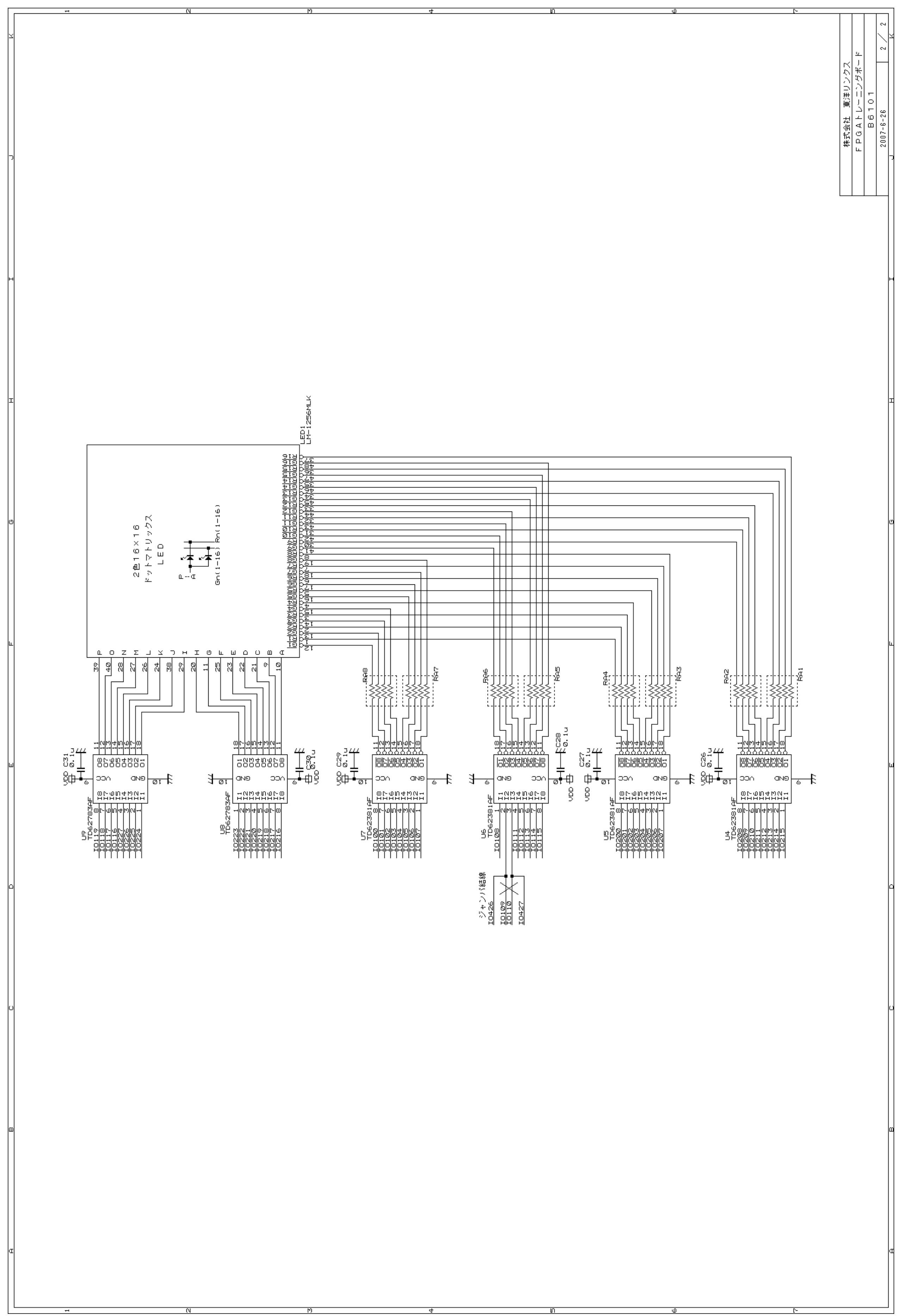
```
    LED1D  <= led1(3);
    LED1E  <= led1(4);
    LED1F  <= led1(5);
    LED1G  <= led1(6);
    LED1DP <= led1(7);
    LED2A  <= led2(0);
    LED2B  <= led2(1);
    LED2C  <= led2(2);
    LED2D  <= led2(3);
    LED2E  <= led2(4);
    LED2F  <= led2(5);
    LED2G  <= led2(6);
    LED2DP <= led2(7);

end RTL;
```

TK-3687mini のプログラムは同じです。ダウンロードして実行してみてください。先ほどと同じデモが始まります。

付録

回路図：FPGA ボード“B6101”




株式会社 東洋リンクス
FPGAトレーニングボード
B6101
2007-6-26
2色16×16
ドットマトリックス
LED

回路図：TK-3687mini

TK-3687mini 部品表

	部品番号	型名，規格	メーカ	数量	備考
1	U1	HD64F3687FP	RENESAS	1	
2	U2	MAX232ACSE	MAXIM	1	
3	U3	S24Cxx(NM/NW)	ST	1	I2CBus仕様EEPROM
4	REG1	TA48M05F(S)	東芝	1	実装しない
5					
6	X1	CX-8045G (20MHz)	キンセキ	1	※DIPセラロック兼用
7	X2	32.768kHz		1	※リードタイプ
8					
9	D3	1SS133-T72	ROHM	1	※リードタイプ
10	LED1			1	※リード・Flat兼用
11					
12	R1	1k～10kΩ(1608)		1	LED1の輝度で調整
13	R2	100Ω(1608)		1	
14	R3,4,5,6	4.7kΩ(1608)		4	
15	RA1,2	BCN16-4AB471J	BI	2	3216サイズ・角アリ・4素子
16	RA3,4	BCN16-4AB472J～103J	BI	2	3216サイズ・角アリ・4素子
17					
18	C1,2,5,7,8,13,14,15,16,21	0.1μF (1608)		10	
19	C3,19	47μF/16V (電解)		2	※リードタイプ，C19は実装しない
20	C4,6,17,18,20	10μF/6.3～16V (電解)		5	※リードタイプ
21	C9,10,11,12	15pF (1608)		4	
22					
23	SW1	SKHHAK/AM/DC	ALPS	1	※DIPパッケージ
24					
25	CN1	B2P-SHF-1AA	JST	1	電源用，実装しない
26	CN3,4	HIF3FB-30DA-2.54DSA	HRS	2	CN4は実装しない
27	CN5	D-Sub9pin		1	メス，ライトアングル
28	CN6	3pin スルー		1	サブ シリアルポート，実装しない
29	CN7	B7B-ZR	JST	1	E8接続用・1.5mmピッチスルー，実装しない
30	CN8			1	サーボモータ電源，実装しない
31	(CN)P60～67	3pin スルー		8	サーボモータ用・2.54mmピッチスルー，実装しない
32					
33	JP1	2pinスルー		1	オンボード書き込み用・2.54mmピッチスルー
34	JP2	パターンジャンパ		1	
35	JP3	パターンジャンパ		1	
36					
37	PCB	B6090	東洋リンクス	1	
38					

Cyclone“EP1C3T144C8N”のピンアサインマップ

信号名称	ピン番号	CN1	CN6		信号名称	ピン番号	CN1	CN6	
IO100	36	3		ドットマトリックスLED，R1	IO300	108		5	H8/3687，P31
IO101	35	4		ドットマトリックスLED，R2	IO301	107		6	H8/3687，P30
IO102	34	5		ドットマトリックスLED，R3	IO302	106		7	H8/3687，P33
IO103	33	6		ドットマトリックスLED，R4	IO303	105		8	H8/3687，P32
IO104	32	7		ドットマトリックスLED，R5	IO304	104		9	H8/3687，P35
IO105	31	8		ドットマトリックスLED，R6	IO305	103		10	H8/3687，P34
IO106	28	9		ドットマトリックスLED，R7	IO306	100		11	H8/3687，P37
IO107	27	10		ドットマトリックスLED，R8	IO307	99		12	H8/3687，P36
IO108	26	11		ドットマトリックスLED，R9	IO308	98		13	H8/3687，P51
IO109	25				IO309	97		14	H8/3687，P50
IO110	12				IO310	96		15	H8/3687，P53
IO111	11	14		ドットマトリックスLED，R12	IO311	94		16	H8/3687，P52
IO112	10	15		ドットマトリックスLED，R13	IO312	91		17	H8/3687，P55
IO113	7	16		ドットマトリックスLED，R14	IO313	85		18	H8/3687，P54
IO114	6	17		ドットマトリックスLED，R15	IO314	84		19	H8/3687，P57
IO115	5	18		ドットマトリックスLED，R16	IO315	83		20	H8/3687，P56
IO116	4	21		ドットマトリックスLED，M	IO316	82		23	H8/3687，P61
IO117	3	22		ドットマトリックスLED，N	IO317	79		24	H8/3687，P60
IO118	2	23		ドットマトリックスLED，O	IO318	78		25	H8/3687，P63
IO119	1	24		ドットマトリックスLED，P	IO319	77		26	H8/3687，P62
					IO320	76		27	H8/3687，P65
					IO321	75		28	H8/3687，P64
					IO322	74		29	H8/3687，P67
					IO323	73		30	H8/3687，P66

信号名称	ピン番号	CN1	CN6		信号名称	ピン番号	CN1	CN6	
IO200	144	29		ドットマトリックスLED，G1	IO400	72		35	SW1
IO201	143	30		ドットマトリックスLED，G2	IO401	71		36	SW2
IO202	142	31		ドットマトリックスLED，G3	IO402	70		37	SW3
IO203	141	32		ドットマトリックスLED，G4	IO403	69		38	SW4
IO204	140	33		ドットマトリックスLED，G5	IO404	68		39	SW5
IO205	139	34		ドットマトリックスLED，G6	IO405	67		40	SW6
IO206	134	35		ドットマトリックスLED，G7	IO406	62		41	SW7
IO207	133	36		ドットマトリックスLED，G8	IO407	61		42	サウンダー
IO208	132	37		ドットマトリックスLED，G9	IO408	60		43	7セグメントLED-2，a
IO209	131	38		ドットマトリックスLED，G10	IO409	59		44	7セグメントLED-2，b
IO210	130	39		ドットマトリックスLED，G11	IO410	58		45	7セグメントLED-2，c
IO211	129	40		ドットマトリックスLED，G12	IO411	57		46	7セグメントLED-2，d
IO212	128	41		ドットマトリックスLED，G13	IO412	56		47	7セグメントLED-2，e
IO213	127	42		ドットマトリックスLED，G14	IO413	55		48	7セグメントLED-2，f
IO214	126	43		ドットマトリックスLED，G15	IO414	54		49	7セグメントLED-2，g
IO215	125	44		ドットマトリックスLED，G16	IO415	53		50	7セグメントLED-2，dp
IO216	124	45		ドットマトリックスLED，A	IO416	52		51	7セグメントLED-1，a
IO217	123	46		ドットマトリックスLED，B	IO417	51		52	7セグメントLED-1，b
IO218	122	47		ドットマトリックスLED，C	IO418	50		53	7セグメントLED-1，c
IO219	121	48		ドットマトリックスLED，D	IO419	49		54	7セグメントLED-1，d
IO220	120	49		ドットマトリックスLED，E	IO420	48		55	7セグメントLED-1，e
IO221	119	50		ドットマトリックスLED，F	IO421	47		56	7セグメントLED-1，f
IO222	114	51		ドットマトリックスLED，G	IO422	42		57	7セグメントLED-1，g
IO223	113	52		ドットマトリックスLED，H	IO423	41		58	7セグメントLED-1，dp
IO224	112	53		ドットマトリックスLED，I	IO424	40		59	
IO225	111	54		ドットマトリックスLED，J	IO425	39		60	
IO226	110	55		ドットマトリックスLED，K	IO426	38	12	61	ドットマトリックスLED，R10
IO227	109	56		ドットマトリックスLED，L	IO427	37	13	62	ドットマトリックスLED，R11

信号名称	ピン番号	CN1	CN6		信号名称	ピン番号	CN1	CN6	
CLK0	16	19			CLK2	93		21	発振モジュール 9.8304MHz
CLK1	17	20			CLK3	92		22	

ブール代数の演算子と公式

■　ブール代数の演算子

論理回路をブール代数の式で表わすときに使う演算子は下の表のように，NOT は「$\overline{}$」，AND は「・」，OR は「+」，EXOR は「⊕」です。NAND と NOR は，AND と OR と NOT を組み合わせて表現します。

AND			OR			NOT		NAND			NOR			EXOR		
$A\cdot B$			$A+B$			$\overline{A}$		$\overline{A\cdot B}$			$\overline{A+B}$			$A\oplus B$		
74HC08			74HC32			74HC04		74HC00			74HC02			74HC86		
入力		出力	入力		出力	入力	出力	入力		出力	入力		出力	入力		出力
A	B	Y	A	B	Y	A	Y	A	B	Y	A	B	Y	A	B	Y
0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0
1	0	0	1	0	1	1	0	1	0	1	1	0	0	1	0	1
0	1	0	0	1	1			0	1	1	0	1	0	0	1	1
1	1	1	1	1	1			1	1	0	1	1	0	1	1	0

■　ブール代数の公式

①交換法則

$$A+B=B+A$$
$$A\cdot B=B\cdot A$$

②結合法則

$$A+(B+C)=(A+B)+C$$
$$A\cdot(B\cdot C)=(A\cdot B)\cdot C$$

③分配法則

$$A\cdot(B+C)=A\cdot B+A\cdot C$$
$$A+(B\cdot C)=(A+B)\cdot(A+C)$$

④同一法則

$$A+A=A$$
$$A\cdot A=A$$

⑤吸収法則

$$A+A\cdot B=A$$
$$A\cdot(A+B)=A$$

⑥補元法則

$$A+\overline{A}=1$$
$$A\cdot\overline{A}=0$$

⑦復元法則

$$\overline{\overline{A}}=A$$

⑧ド・モルガンの定理

$$\overline{A+B}=\overline{A}\cdot\overline{B}$$
$$\overline{A\cdot B}=\overline{A}+\overline{B}$$

⑨恒等法則

$$A+0=A$$
$$A\cdot 1=A$$

⑩

$$A+1=1$$
$$A\cdot 0=0$$

デモプログラム

FPGA ボードの動作確認用プログラムです。FPGA ボード完成品には出荷時にあらかじめ書き込まれています。フォルダは「demo_01」です。

```
-----------------------------------------------------------
-- FPGA Training Board(B6101)
-- Demo-01
-----------------------------------------------------------
-- File   : demo_01_top.vhd
-- Date   : 2007-09-26
-- Family : Cyclone
-- Device : EP1C3T144C8
--
-- Designed by TOYO-LINX Co.,Ltd. / yKikuchi
-----------------------------------------------------------

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity demo_01_top is
    port(
        CLK             :   in std_logic;
        LED_SEL         :   out std_logic_vector(15 downto 0);
        LED_DATA_R      :   out std_logic_vector(0 to 15);
        LED_DATA_G      :   out std_logic_vector(0 to 15);
        SW              :   in std_logic_vector(7 downto 1);
        BUZZ            :   out std_logic;
        LED1A           :   out std_logic;
        LED1B           :   out std_logic;
        LED1C           :   out std_logic;
        LED1D           :   out std_logic;
        LED1E           :   out std_logic;
        LED1F           :   out std_logic;
        LED1G           :   out std_logic;
        LED1DP          :   out std_logic;
        LED2A           :   out std_logic;
        LED2B           :   out std_logic;
        LED2C           :   out std_logic;
        LED2D           :   out std_logic;
        LED2E           :   out std_logic;
        LED2F           :   out std_logic;
        LED2G           :   out std_logic;
        LED2DP          :   out std_logic
    );
end demo_01_top;

architecture RTL of demo_01_top is

    signal div_cnt      :   std_logic_vector(31 downto 0);
    signal div_clk      :   std_logic;
    signal scan_cnt     :   std_logic_vector(3 downto 0);
```

```
    signal shift_clk   :   std_logic;
    signal direction   :   std_logic_vector(1 downto 0) := "00";
    signal buzz_osc    :   std_logic_vector(6 downto 0);
    signal count_clk   :   std_logic;
    signal counter0    :   std_logic_vector(3 downto 0);
    signal counter1    :   std_logic_vector(3 downto 0);
    signal count0_en   :   std_logic := '1';
    signal count1_en   :   std_logic := '1';
    signal led1        :   std_logic_vector(7 downto 0);
    signal led2        :   std_logic_vector(7 downto 0);

    ------------------------------------------------------------Green----------------------Red---------
    signal led_reg_F   :   std_logic_vector(0 to 31) := "0000000000000000" & "1110000000011111";
    signal led_reg_E   :   std_logic_vector(0 to 31) := "0000000000000000" & "1100000000001111";
    signal led_reg_D   :   std_logic_vector(0 to 31) := "0000000000000000" & "1000000000000111";
    signal led_reg_C   :   std_logic_vector(0 to 31) := "0001111110000000" & "0001111110000000";
    signal led_reg_B   :   std_logic_vector(0 to 31) := "0011111111000000" & "0010010001000000";
    signal led_reg_A   :   std_logic_vector(0 to 31) := "0111111111100000" & "0100010000100000";
    signal led_reg_9   :   std_logic_vector(0 to 31) := "1111111111110000" & "1000010000010000";
    signal led_reg_8   :   std_logic_vector(0 to 31) := "1111111111111100" & "1111111111111100";
    signal led_reg_7   :   std_logic_vector(0 to 31) := "1111111111111110" & "1100111111100110";
    signal led_reg_6   :   std_logic_vector(0 to 31) := "1111111111111110" & "1000011111000010";
    signal led_reg_5   :   std_logic_vector(0 to 31) := "0111100000111100" & "0000000000000000";
    signal led_reg_4   :   std_logic_vector(0 to 31) := "0011000000011000" & "0000000000000000";
    signal led_reg_3   :   std_logic_vector(0 to 31) := "0000000000000000" & "1111111111111111";
    signal led_reg_2   :   std_logic_vector(0 to 31) := "0000000000000000" & "0000000000000000";
    signal led_reg_1   :   std_logic_vector(0 to 31) := "0011001100110011" & "0000000000000000";
    signal led_reg_0   :   std_logic_vector(0 to 31) := "0011001100110011" & "0000000000000000";

begin
    process(CLK)
    begin
        if(CLK'event and CLK='1') then
            div_cnt <= div_cnt + 1;
        end if;
    end process;
    div_clk <= div_cnt(11);
    shift_clk <= div_cnt(19);
    buzz_osc <= div_cnt(15 downto 9);
    count_clk <= div_cnt(18);

    process(div_clk)
    begin
        if(div_clk'event and div_clk='1') then
            scan_cnt <= scan_cnt + 1;
        end if;
    end process;

    process(scan_cnt)
    begin
        case scan_cnt is
            when "1111" =>
                LED_DATA_R <= led_reg_F(16 to 31);
                LED_DATA_G <= led_reg_F( 0 to 15);
                LED_SEL    <= "1000000000000000";
```

```
            when "1110" =>
                LED_DATA_R <= led_reg_E(16 to 31);
                LED_DATA_G <= led_reg_E( 0 to 15);
                LED_SEL    <= "0100000000000000";
            when "1101" =>
                LED_DATA_R <= led_reg_D(16 to 31);
                LED_DATA_G <= led_reg_D( 0 to 15);
                LED_SEL    <= "0010000000000000";
            when "1100" =>
                LED_DATA_R <= led_reg_C(16 to 31);
                LED_DATA_G <= led_reg_C( 0 to 15);
                LED_SEL    <= "0001000000000000";
            when "1011" =>
                LED_DATA_R <= led_reg_B(16 to 31);
                LED_DATA_G <= led_reg_B( 0 to 15);
                LED_SEL    <= "0000100000000000";
            when "1010" =>
                LED_DATA_R <= led_reg_A(16 to 31);
                LED_DATA_G <= led_reg_A( 0 to 15);
                LED_SEL    <= "0000010000000000";
            when "1001" =>
                LED_DATA_R <= led_reg_9(16 to 31);
                LED_DATA_G <= led_reg_9( 0 to 15);
                LED_SEL    <= "0000001000000000";
            when "1000" =>
                LED_DATA_R <= led_reg_8(16 to 31);
                LED_DATA_G <= led_reg_8( 0 to 15);
                LED_SEL    <= "0000000100000000";
            when "0111" =>
                LED_DATA_R <= led_reg_7(16 to 31);
                LED_DATA_G <= led_reg_7( 0 to 15);
                LED_SEL    <= "0000000010000000";
            when "0110" =>
                LED_DATA_R <= led_reg_6(16 to 31);
                LED_DATA_G <= led_reg_6( 0 to 15);
                LED_SEL    <= "0000000001000000";
            when "0101" =>
                LED_DATA_R <= led_reg_5(16 to 31);
                LED_DATA_G <= led_reg_5( 0 to 15);
                LED_SEL    <= "0000000000100000";
            when "0100" =>
                LED_DATA_R <= led_reg_4(16 to 31);
                LED_DATA_G <= led_reg_4( 0 to 15);
                LED_SEL    <= "0000000000010000";
            when "0011" =>
                LED_DATA_R <= led_reg_3(16 to 31);
                LED_DATA_G <= led_reg_3( 0 to 15);
                LED_SEL    <= "0000000000001000";
            when "0010" =>
                LED_DATA_R <= led_reg_2(16 to 31);
                LED_DATA_G <= led_reg_2( 0 to 15);
                LED_SEL    <= "0000000000000100";
            when "0001" =>
                LED_DATA_R <= led_reg_1(16 to 31);
                LED_DATA_G <= led_reg_1( 0 to 15);
```

```
                LED_SEL    <= "0000000000000010";
            when "0000" =>
                LED_DATA_R <= led_reg_0(16 to 31);
                LED_DATA_G <= led_reg_0( 0 to 15);
                LED_SEL    <= "0000000000000001";
            when others =>
                LED_DATA_R <= "0000000000000000";
                LED_DATA_G <= "0000000000000000";
                LED_SEL    <= "0000000000000000";
        end case;
    end process;

    process(shift_clk)
    begin
        if(shift_clk'event and shift_clk='1') then
            if(direction="00") then
                for I in 15 downto 1 loop
                    led_reg_F(I+16) <= led_reg_F(I+15);
                    led_reg_E(I+16) <= led_reg_E(I+15);
                    led_reg_D(I+16) <= led_reg_D(I+15);
                    led_reg_C(I+16) <= led_reg_C(I+15);
                    led_reg_B(I+16) <= led_reg_B(I+15);
                    led_reg_A(I+16) <= led_reg_A(I+15);
                    led_reg_9(I+16) <= led_reg_9(I+15);
                    led_reg_8(I+16) <= led_reg_8(I+15);
                    led_reg_7(I+16) <= led_reg_7(I+15);
                    led_reg_6(I+16) <= led_reg_6(I+15);
                    led_reg_5(I+16) <= led_reg_5(I+15);
                    led_reg_4(I+16) <= led_reg_4(I+15);
                    led_reg_3(I+16) <= led_reg_3(I+15);
                    led_reg_2(I+16) <= led_reg_2(I+15);
                    led_reg_1(I+16) <= led_reg_1(I+15);
                    led_reg_0(I+16) <= led_reg_0(I+15);
                    led_reg_F(I)    <= led_reg_F(I-1);
                    led_reg_E(I)    <= led_reg_E(I-1);
                    led_reg_D(I)    <= led_reg_D(I-1);
                    led_reg_C(I)    <= led_reg_C(I-1);
                    led_reg_B(I)    <= led_reg_B(I-1);
                    led_reg_A(I)    <= led_reg_A(I-1);
                    led_reg_9(I)    <= led_reg_9(I-1);
                    led_reg_8(I)    <= led_reg_8(I-1);
                    led_reg_7(I)    <= led_reg_7(I-1);
                    led_reg_6(I)    <= led_reg_6(I-1);
                    led_reg_5(I)    <= led_reg_5(I-1);
                    led_reg_4(I)    <= led_reg_4(I-1);
                    led_reg_3(I)    <= led_reg_3(I-1);
                    led_reg_2(I)    <= led_reg_2(I-1);
                    led_reg_1(I)    <= led_reg_1(I-1);
                    led_reg_0(I)    <= led_reg_0(I-1);
                end loop;
                led_reg_F(16) <= led_reg_F(31);
                led_reg_E(16) <= led_reg_E(31);
                led_reg_D(16) <= led_reg_D(31);
                led_reg_C(16) <= led_reg_C(31);
                led_reg_B(16) <= led_reg_B(31);
```

```
            led_reg_A(16) <= led_reg_A(31);
            led_reg_9(16) <= led_reg_9(31);
            led_reg_8(16) <= led_reg_8(31);
            led_reg_7(16) <= led_reg_7(31);
            led_reg_6(16) <= led_reg_6(31);
            led_reg_5(16) <= led_reg_5(31);
            led_reg_4(16) <= led_reg_4(31);
            led_reg_3(16) <= led_reg_3(31);
            led_reg_2(16) <= led_reg_2(31);
            led_reg_1(16) <= led_reg_1(31);
            led_reg_0(16) <= led_reg_0(31);
            led_reg_F( 0) <= led_reg_F(15);
            led_reg_E( 0) <= led_reg_E(15);
            led_reg_D( 0) <= led_reg_D(15);
            led_reg_C( 0) <= led_reg_C(15);
            led_reg_B( 0) <= led_reg_B(15);
            led_reg_A( 0) <= led_reg_A(15);
            led_reg_9( 0) <= led_reg_9(15);
            led_reg_8( 0) <= led_reg_8(15);
            led_reg_7( 0) <= led_reg_7(15);
            led_reg_6( 0) <= led_reg_6(15);
            led_reg_5( 0) <= led_reg_5(15);
            led_reg_4( 0) <= led_reg_4(15);
            led_reg_3( 0) <= led_reg_3(15);
            led_reg_2( 0) <= led_reg_2(15);
            led_reg_1( 0) <= led_reg_1(15);
            led_reg_0( 0) <= led_reg_0(15);
        elsif(direction="01") then
            for I in 0 to 14 loop
                led_reg_F(I+16) <= led_reg_F(I+17);
                led_reg_E(I+16) <= led_reg_E(I+17);
                led_reg_D(I+16) <= led_reg_D(I+17);
                led_reg_C(I+16) <= led_reg_C(I+17);
                led_reg_B(I+16) <= led_reg_B(I+17);
                led_reg_A(I+16) <= led_reg_A(I+17);
                led_reg_9(I+16) <= led_reg_9(I+17);
                led_reg_8(I+16) <= led_reg_8(I+17);
                led_reg_7(I+16) <= led_reg_7(I+17);
                led_reg_6(I+16) <= led_reg_6(I+17);
                led_reg_5(I+16) <= led_reg_5(I+17);
                led_reg_4(I+16) <= led_reg_4(I+17);
                led_reg_3(I+16) <= led_reg_3(I+17);
                led_reg_2(I+16) <= led_reg_2(I+17);
                led_reg_1(I+16) <= led_reg_1(I+17);
                led_reg_0(I+16) <= led_reg_0(I+17);
                led_reg_F(I)    <= led_reg_F(I+1);
                led_reg_E(I)    <= led_reg_E(I+1);
                led_reg_D(I)    <= led_reg_D(I+1);
                led_reg_C(I)    <= led_reg_C(I+1);
                led_reg_B(I)    <= led_reg_B(I+1);
                led_reg_A(I)    <= led_reg_A(I+1);
                led_reg_9(I)    <= led_reg_9(I+1);
                led_reg_8(I)    <= led_reg_8(I+1);
                led_reg_7(I)    <= led_reg_7(I+1);
                led_reg_6(I)    <= led_reg_6(I+1);
```

```
                led_reg_5(I)    <= led_reg_5(I+1);
                led_reg_4(I)    <= led_reg_4(I+1);
                led_reg_3(I)    <= led_reg_3(I+1);
                led_reg_2(I)    <= led_reg_2(I+1);
                led_reg_1(I)    <= led_reg_1(I+1);
                led_reg_0(I)    <= led_reg_0(I+1);
            end loop;
            led_reg_F(31) <= led_reg_F(16);
            led_reg_E(31) <= led_reg_E(16);
            led_reg_D(31) <= led_reg_D(16);
            led_reg_C(31) <= led_reg_C(16);
            led_reg_B(31) <= led_reg_B(16);
            led_reg_A(31) <= led_reg_A(16);
            led_reg_9(31) <= led_reg_9(16);
            led_reg_8(31) <= led_reg_8(16);
            led_reg_7(31) <= led_reg_7(16);
            led_reg_6(31) <= led_reg_6(16);
            led_reg_5(31) <= led_reg_5(16);
            led_reg_4(31) <= led_reg_4(16);
            led_reg_3(31) <= led_reg_3(16);
            led_reg_2(31) <= led_reg_2(16);
            led_reg_1(31) <= led_reg_1(16);
            led_reg_0(31) <= led_reg_0(16);
            led_reg_F(15) <= led_reg_F(0);
            led_reg_E(15) <= led_reg_E(0);
            led_reg_D(15) <= led_reg_D(0);
            led_reg_C(15) <= led_reg_C(0);
            led_reg_B(15) <= led_reg_B(0);
            led_reg_A(15) <= led_reg_A(0);
            led_reg_9(15) <= led_reg_9(0);
            led_reg_8(15) <= led_reg_8(0);
            led_reg_7(15) <= led_reg_7(0);
            led_reg_6(15) <= led_reg_6(0);
            led_reg_5(15) <= led_reg_5(0);
            led_reg_4(15) <= led_reg_4(0);
            led_reg_3(15) <= led_reg_3(0);
            led_reg_2(15) <= led_reg_2(0);
            led_reg_1(15) <= led_reg_1(0);
            led_reg_0(15) <= led_reg_0(0);
        elsif(direction="10") then
            led_reg_F <= led_reg_E;
            led_reg_E <= led_reg_D;
            led_reg_D <= led_reg_C;
            led_reg_C <= led_reg_B;
            led_reg_B <= led_reg_A;
            led_reg_A <= led_reg_9;
            led_reg_9 <= led_reg_8;
            led_reg_8 <= led_reg_7;
            led_reg_7 <= led_reg_6;
            led_reg_6 <= led_reg_5;
            led_reg_5 <= led_reg_4;
            led_reg_4 <= led_reg_3;
            led_reg_3 <= led_reg_2;
            led_reg_2 <= led_reg_1;
            led_reg_1 <= led_reg_0;
```

```
                led_reg_0 <= led_reg_F;
            elsif(direction="11") then
                led_reg_F <= led_reg_0;
                led_reg_E <= led_reg_F;
                led_reg_D <= led_reg_E;
                led_reg_C <= led_reg_D;
                led_reg_B <= led_reg_C;
                led_reg_A <= led_reg_B;
                led_reg_9 <= led_reg_A;
                led_reg_8 <= led_reg_9;
                led_reg_7 <= led_reg_8;
                led_reg_6 <= led_reg_7;
                led_reg_5 <= led_reg_6;
                led_reg_4 <= led_reg_5;
                led_reg_3 <= led_reg_4;
                led_reg_2 <= led_reg_3;
                led_reg_1 <= led_reg_2;
                led_reg_0 <= led_reg_1;
            else
                led_reg_F <= led_reg_F;
                led_reg_E <= led_reg_E;
                led_reg_D <= led_reg_D;
                led_reg_C <= led_reg_C;
                led_reg_B <= led_reg_B;
                led_reg_A <= led_reg_A;
                led_reg_9 <= led_reg_9;
                led_reg_8 <= led_reg_8;
                led_reg_7 <= led_reg_7;
                led_reg_6 <= led_reg_6;
                led_reg_5 <= led_reg_5;
                led_reg_4 <= led_reg_4;
                led_reg_3 <= led_reg_3;
                led_reg_2 <= led_reg_2;
                led_reg_1 <= led_reg_1;
                led_reg_0 <= led_reg_0;
            end if;
        end if;
    end process;

    process(shift_clk)
    begin
        if(shift_clk'event and shift_clk='1') then
            if   (SW(4)='0')then
                direction <= "00";
            elsif(SW(3)='0')then
                direction <= "01";
            elsif(SW(2)='0')then
                direction <= "11";
            elsif(SW(1)='0')then
                direction <= "10";
            end if;
        end if;
    end process;

    process(SW)
```

```
    begin
        if   (SW(7)='0')then
            BUZZ <= buzz_osc(6);
        elsif(SW(6)='0')then
            BUZZ <= buzz_osc(5);
        elsif(SW(5)='0')then
            BUZZ <= buzz_osc(4);
        elsif(SW(4)='0')then
            BUZZ <= buzz_osc(3);
        elsif(SW(3)='0')then
            BUZZ <= buzz_osc(2);
        elsif(SW(2)='0')then
            BUZZ <= buzz_osc(1);
        elsif(SW(1)='0')then
            BUZZ <= buzz_osc(0);
        else
            BUZZ <= '0';
        end if;
    end process;

    process(count_clk)
    begin
        if(count_clk'event and count_clk='1') then
            if   (SW(7)='0')then
                count0_en <= '1';
                count1_en <= '1';
            elsif(SW(6)='0')then
                count0_en <= '0';
            elsif(SW(5)='0')then
                count1_en <= '0';
            end if;
        end if;
    end process;

    process(count_clk)
    begin
        if(count_clk'event and count_clk='1') then
            if(count0_en='1')then
                counter0 <= counter0 + 1;
            end if;
            if(count1_en='1')then
                counter1 <= counter1 + 1;
            end if;
        end if;
    end process;

    process(counter0,counter1)
    begin
        case counter0 is
            when "0000" =>
                led2 <= "00111111";
            when "0001" =>
                led2 <= "00000110";
            when "0010" =>
                led2 <= "01011011";
```

```
            when "0011" =>
                led2 <= "01001111";
            when "0100" =>
                led2 <= "01100110";
            when "0101" =>
                led2 <= "01101101";
            when "0110" =>
                led2 <= "01111101";
            when "0111" =>
                led2 <= "00000111";
            when "1000" =>
                led2 <= "01111111";
            when "1001" =>
                led2 <= "01101111";
            when "1010" =>
                led2 <= "01110111";
            when "1011" =>
                led2 <= "01111100";
            when "1100" =>
                led2 <= "00111001";
            when "1101" =>
                led2 <= "01011110";
            when "1110" =>
                led2 <= "01111001";
            when "1111" =>
                led2 <= "01110001";
            when others =>
                led2 <= "00000000";
        end case;
        case counter1 is
            when "0000" =>
                led1 <= "00111111";
            when "0001" =>
                led1 <= "00000110";
            when "0010" =>
                led1 <= "01011011";
            when "0011" =>
                led1 <= "01001111";
            when "0100" =>
                led1 <= "01100110";
            when "0101" =>
                led1 <= "01101101";
            when "0110" =>
                led1 <= "01111101";
            when "0111" =>
                led1 <= "00000111";
            when "1000" =>
                led1 <= "01111111";
            when "1001" =>
                led1 <= "01101111";
            when "1010" =>
                led1 <= "01110111";
            when "1011" =>
                led1 <= "01111100";
            when "1100" =>
```

```
                    led1 <= "00111001";
               when "1101" =>
                    led1 <= "01011110";
               when "1110" =>
                    led1 <= "01111001";
               when "1111" =>
                    led1 <= "01110001";
               when others =>
                    led1 <= "00000000";
          end case;
     end process;

     LED1A  <= led1(0);
     LED1B  <= led1(1);
     LED1C  <= led1(2);
     LED1D  <= led1(3);
     LED1E  <= led1(4);
     LED1F  <= led1(5);
     LED1G  <= led1(6);
     LED1DP <= led1(7);
     LED2A  <= led2(0);
     LED2B  <= led2(1);
     LED2C  <= led2(2);
     LED2D  <= led2(3);
     LED2E  <= led2(4);
     LED2F  <= led2(5);
     LED2G  <= led2(6);
     LED2DP <= led2(7);

end RTL;
```

株式会社東洋リンクス

※ご質問はメール，またはFAXで…
ユーザーサポート係（月～金 10:00～17:00，土日祝は除く）
〒102-0093　東京都千代田区平河町 1-2-2　朝日ビル
TEL：03-3234-0559
FAX：03-3234-0549
E-mail：toyolinx@va.u-netsurf.jp
URL：http://www2.u-netsurf.ne.jp/~toyolinx

20090612