EDX－001ユーザーズマニュアル

第3版 R2

ヒューマンデータ

目次

はじめに 1
ご注意 1
製品の内容について 1
1 概要 2
2 開発環境 2
3 EDX-001 の構成 2
3．1 ダウンロードケーブル 2
3．2 設計手順 2
3．3 コンフィグレーションの仕組み 2
3．4 電源 2
3．5 クロック 2
3．6 ブザー 3
3．7 数字表示 LED 3
3．8 押しボタンスイッチ 3
3．9 汎用 LED 3
3．10 ドットマトリックスディスプレイ 3
3．11 USB インターフェース 4
4 仮想 COM ポートドライバ 5
4．1 インストール手順 5
4．2 インストール作業完了の確認 6
4．3 ドライバのアンインストール 6
5 コンフィグレーションと通信実験 7
5．1 TeraTerm の起動 7
5．2 TeraTerm による FPGA コンフィグレーション 7
5．3 TeraTerm による通信実験 7
5．4 ED Term による FPGA コンフィグレーション 7
5．5 ED Term を利用した通信実験 8
5．6 COPY コマンド 8
6 ISE による開発手順 9
6．1 Project Navigator の起動 9
6．2 プロジェクトの新規作成 9
6．3 VHDL によるデコーダの作成 9
6．4 デコーダのテストベンチ波形を作成 10
6．5 デコーダの VHDL テストベンチを作成 10
6．6 デコーダのシミュレーション実行 11
6．7 VHDL による分周回路の作成 12
6．8 分周回路のテストベンチ波形を作成 12
6．9 分周回路の VHDL テストベンチを作成 12
6．10 分周回路のシミュレーション 13
6．11 回路図シンボルの作成 13
6．12 最上位回路図の作成 13
6．13 UCF ファイルの作成 15
6．14 論理合成の実行 15
6．15 配置配線の実行 15
6．16 プログラミングファイルの生成 15
6．17 EDX-001 へのダウンロード 15
7 ダイナミック点灯 16
7．1 セグメントデコータ 16
7．2 セレクタ用カウンタ 16
7．3 セグメントセレクタ 16
8 ドットマトリックスディスプレイの点灯 17
8．1 表示手順 17

８．２ 表示モジュール 18
９ USB 通信実験 19
９．１ リードサイクルのタイミングチャート 19
９．２ ライトサイクルのタイミングチャート 19
９．３ 調停回路の状態遷移図 19
９．４ USB 調停回路 VHDL モジュール 20
１０ EDX-001 サンプル回路 22
１０．１ サンプル回路1 22
１０．２ サンプル回路2 22
１０．３ サンプル回路3 22
１０．４ サンプル回路4 23
１１ 付属 CD-ROM の内容 24
１２ トラブル Q&A 24
１３ EDX-001 参考資料について 24

はじめに

この度は，FPGA トレーナ EDX-001 をお買い上げいただきまして，誠にありがとうございます．

EDX-001 は，Xilinx 社の FPGA であるスパルタンⅡ（XC2S100TQ144：約 10 万ゲート）を実装したトレーナです．ドットマトリックスディスプレイ，4桁数字表示 LED，汎用 LED，電子ブザー，クロックモジュール，USB インターフェースを実装しておりますので，快適に FPGA 設計を進めることができます．

FPGA コンフィグレーションは，USB にて行うことができ，コンフィグレーション完了後は FPGA に記述したのユーザー回路で USB を利用することができます．VHDL や Verilog-HDL による論理回路設計の習得に，ご活用ください．

ご注意

1. 本書の内容は，改良のため将来予告なしに変更することがありますので，ご了承願います．
2. 本書の内容については万全の記して作成しましたが，万一誤りなど，お気づきの点がございましたら，ご連絡をお願いいたします．
3. 本製品の運用の結果につきましては，2．項にかかわらず当社は責任を負いかねますので，ご了承願います．
4. 本書に記載されている使用と異なる使用をされ，あるいは本書に記載されていない使用をされた場合の結果については，当社は責任を負いません．
5. 本書および，回路図，サンプル回路などを無断で複写，引用，配布することはお断りいたします．

製品の内容について

本パッケージには，以下のものが含まれています．万一，不足などがございましたら，弊社宛にご連絡ください．

FPGA トレーナ/EDX-001 本体	1
USB ケーブル	1
ドライバ CD	1
AC アダプタ（9V）	1
マニュアル（本書）	1
ユーザー登録はがき	1

1 概要

EDX-001 は，Xilinx 社の FPGA であるスパルタンⅡ（XC2S100TQ144：約 10 万ゲート）を実装した FPGA トレーナです．

7 セグメント表示器 4 桁，ドットマトリックスディスプレイ，汎用 LED を 8 個，ブザー，押しボタンスイッチ，クロックモジュールを備え，USB にて，FPGA のコンフィグレーションから通信実験まで行えます．FTDI 社の仮想 COM ポートドライバを利用すれば，通常の COM ポートのように操作できます．

Verilog-HDL や VHDL によるデジタル回路を習得，論理回路設計の試作評価に最適な環境を提供いたします．

2 開発環境

FPGA の内部回路設計には，回路図エディタや HDL 入力ツール，論理合成ツールが必要です．これらの開発は Xilinx 社が無償配布する WebPack ISE にて可能です．

使用する際には，インターネットによるライセンス登録が必要となります．

また，EDX-001 を動作させるには，以下のスペックを満たす必要があります．

PC	USB ポートを装備する PC/AT 互換機
OS	Windows 2000/XP
メモリ	64Mbyte 以上
HDD 空き容量	500M バイト以上

3 EDX-001 の構成

以下に EDX-001 のレイアウトを示します．

3．1 ダウンロードケーブル

FPGA へのコンフィグレーションには，専用のダウンロードケーブルを必要としません．添付の USB ケーブルをご使用ください．

3．2 設計手順

PC から，コンフィグレーションデータ（bit ファイル）を FPGA にダウンロードするには，次の手順が必要となります．

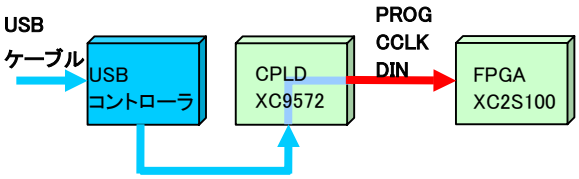
1.. Web Pack ISE にて bit ファイル を作成する
2. 初期化（INIT）スイッチを押し，DONE LED の消灯を確認（電源投入時は不要）
3. PC から，bit ファイルのダウンロード
4. ダウンロード完了（DONE LED）の点灯を確認

3．3 コンフィグレーションの仕組み

FPGA のコンフィグレーションモードは Slave Serial Mode に固定してあります．必要な制御は，CPLD が行いますが，初期化(INIT)ボタンを押す操作が必要です．

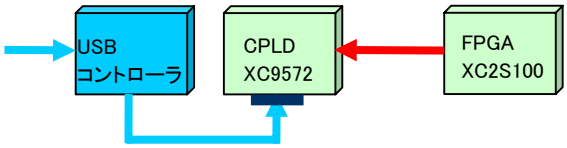
■FPGA へのコンフィグレーション

開発ツールで生成された bit ファイルを USB ケーブルで送信します．CPLD はコンフィグレーションに必要な情報を FPGA に与えます．



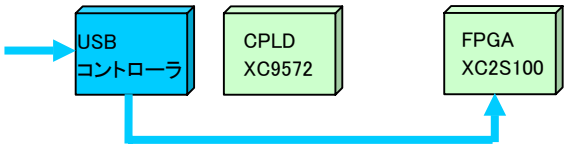
■コンフィグレーション完了

FPGA のコンフィグレーション完了は DONE 信号によって，CPLD に伝えられます．CPLD は USB の制御信号をハイインピーダンスにし，DONE LED を点灯させます．



■FPGA で USB を利用

USB コントローラからの信号は FPGA 側で利用することが可能です．CPLD 側に制御を戻すには 初期化(INIT)スイッチを押してください．DONE LED は消灯します．



以上のように，FPGA からの DONE 信号が H になると，CPLD は USB 制御信号をすぐに解放します．その時に，bit ファイルの末尾から 14byte 程度データである‘0x00’が USB コントローラのキューに残ります．

これは FPGA のコンフィグレーションと，共通のデータ線で通信実験を行うためですので，ご了承ください．

3．4 電源

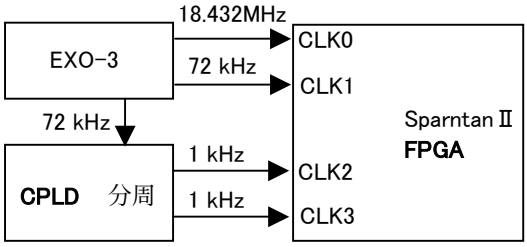
電源は USB から供給されます．USB からの電源供給が不十分な場合は AC アダプタを接続し，電源を供給してください．

内部で必要な，5V，3.3V，2.5V はオンボードのレギュレータにより生成されます．内部でブリッジダイオードを使用しているため，電源ジャックに極性はありません．2.1φの標準的なＡＣアダプタ(9 から 12V)を用いることができます．付属の AC アダプタをご使用ください．

3．5 クロック

EXO-3 より，FPGA に 18.432MHz と 72kHz のクロックを供給します．クロックの周波数は固定です．また，CPLD 内部回路にて 72kHz を 1kHz に分周し，FPGA に供給して

います．クロックピンはすべてグローバルピンです．表のピン番号は FPGA のピン番号です．



ピン名	周波数	ピン番号
CLK0	18.432 MHz	88番
CLK1	72 kHz	91番
CLK2	1 kHz	18番
CLK3	1 kHz	15番

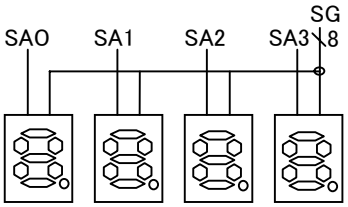
3．6 ブザー

任意の周波数で駆動できます．最も大きく鳴らすには，共振周波数である約4KHzのパルスを BUZZ に送ってください．

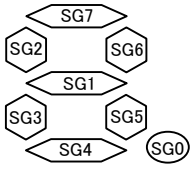
ピン名	ピン番号
BUZZ	96番

3．7 数字表示 LED

ダイナミック点灯方式を用いています．SA はセグメントの選択信号で，SG は共通のセグメントデータです．



各セグメントデータは以下のように定義されます．信号は負論理です．



ピン名	ピン番号
XSG7	29番
XSG6	30番
XSG5	31番
XSG4	40番
XSG3	41番
XSG2	42番
XSG1	43番
XSG0	44番

4 つの 7SEG の一つを選択する信号 SA0,SA1,SA2,SA3 のいずれかをLとし，その他をハイインピーダンスまたはオープンドレインにてオフとします．数 kHz で順番に表示させることによって，連続して点灯してるようにみえます．また，複数を同時に L とすると電流が過大となりますので，注意して下さい．

ピン名		ピン番号
SA0	左端	46番
SA1		47番
SA2		48番
SA3	右端	49番

3．8 押しボタンスイッチ

押すと L になります．このピンは FPGA に対して，入力となりますので，出力しないでください．

ピン名	ピン番号
PSW0	56番
PSW1	54番
PSW2	51番
PSW3	50番

3．9 汎用 LED

汎用 LED は負論理です．Low にて点灯します．完全に消灯させるには，ハイインピーダンスとするか，オープンドレインにてオフとしてください．

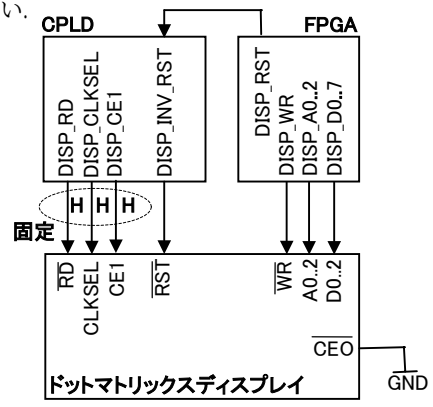
ピン名	ピン番号
LED0	57番
LED1	58番
LED2	59番
LED3	60番
LED4	62番
LED5	63番
LED6	64番
LED7	65番

3．10 ドットマトリックスディスプレイ

英数字や記号 (8 ビットのアスキーコード) を表示させることができます．制御は WR,A0,A1,A2 でおこないます．RD,CLKSEL,CE1 は CPLD 内部回路にて High に固定してあります．

電源投入時に，ディスプレイを無表示状態とするために，DISP_RST(リセット信号)を CPLD 内部回路でインバータに通してあります．

制御の詳細は，付属のデータシート(PD4435.pdf)を御覧ください．



ピン名	ピン番号
DISP_WR	77番
DISP_A0	93番
DISP_A1	94番
DISP_A2	95番
DISP_D0	87番
DISP_D1	86番
DISP_D2	85番
DISP_D3	84番
DISP_D4	83番
DISP_D5	80番
DISP_D6	79番
DISP_D7	78番
DISP_RST	74番

3．11　USB インターフェース

仮想 COM ポートドライバを利用することで，通常のシリアルポートのように扱えます．

コンフィグレーションや通信実験を行う前に，**仮想 COM ポートドライバ**を PC にインストールしてください．

コンフィグレーションが終了し，DONE 信号が H になると USB 制御信号は CPLD 側から FPGA 側に移ります．再び，CPLD 側にて USB 制御信号を扱うには，初期化(INIT)スイッチを押してください．コンフィグレーションが可能な状態に移ります．

ユーザー回路で USB を利用する際には，付属のデータシート(ds245b11.pdf) を御覧ください．

ピン名	入出力（FPGA）	ピン番号
USB_DATA7	入出力	19番
USB_DATA6	入出力	20番
USB_DATA5	入出力	21番
USB_DATA4	入出力	22番
USB_DATA3	入出力	23番
USB_DATA2	入出力	26番
USB_DATA1	入出力	27番
USB_DATA0	入出力	28番
USB_WR	出力	10番
USB_TXE	入力	11番
USB_RD	出力	12番
USB_RXF	入力	13番

4 仮想 COM ポートドライバ

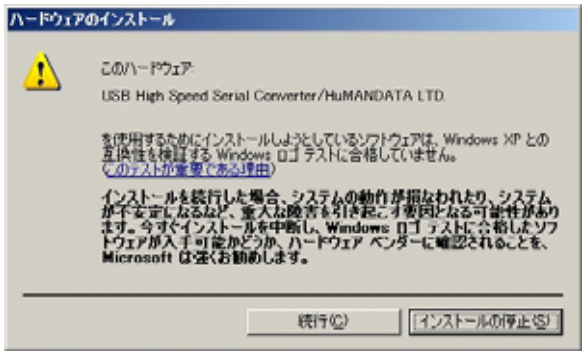
FPGA へのコンフィグレーションと，USB の通信実験の前には，FTDI 社の提供する仮想 COM ポートドライバを PC にインストールする必要があります．

ドライバのインストールは毎回行う必要はありません．本章で説明するインストール作業が完了後，通常の COM ポートのように操作できます．

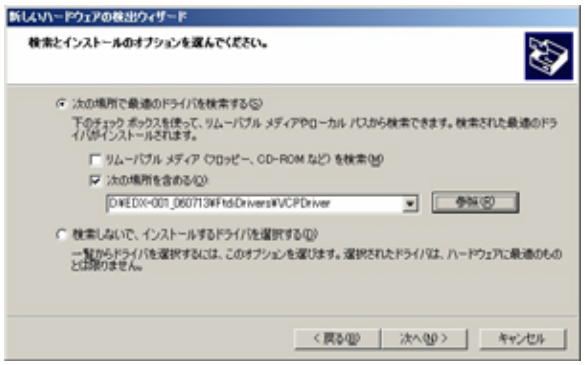
4．1 インストール手順

PC と EDX-001 を USB ケーブルで接続してください．WindowsXP の場合は，次のダイアログが表示されます．

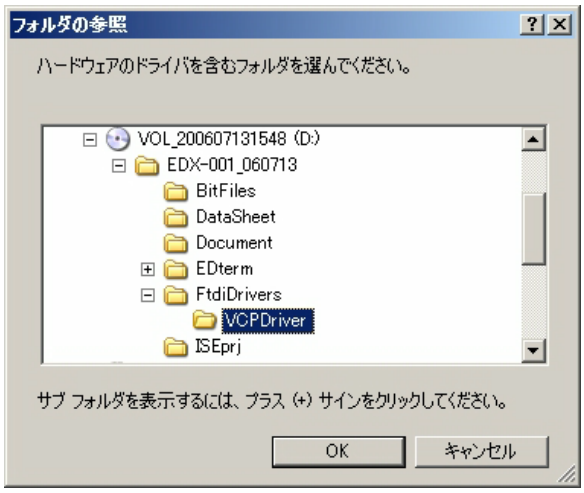
▲[一覧または特定の場所からインストールする (詳細)] を選択．



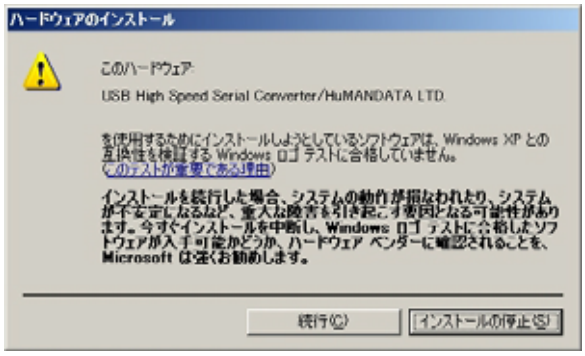
▲[続行]をクリック．



▲[次の場所を含める]を選択．



▲ [参照] をクリックし「 ftdibus.inf 」 の格納場所を指定します．付属 CD の FtdiDrivers/VcpDriver を選択して下さい．



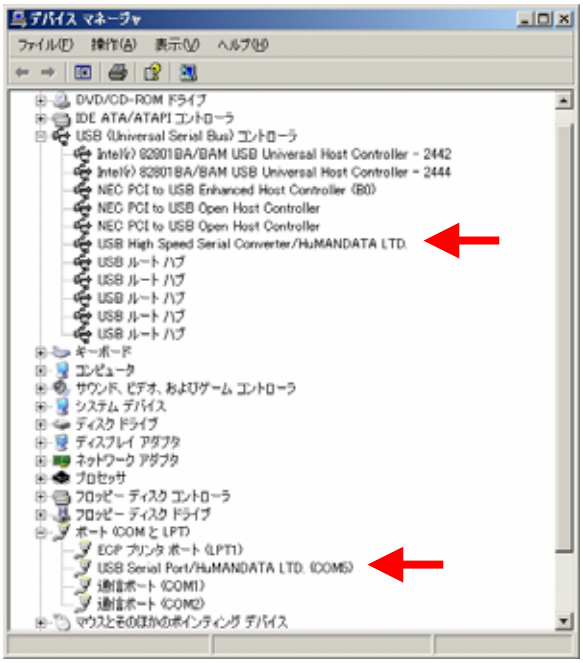
▲ [続行]をクリック．

▲[完了] をクリックすれば，1つめのインストール作業は完了です．

続けて，Windows は新しいハードウェアを認識します．同様の手順で，USB Serial Port のドライバをインストールします．「ftdiport.inf」の格納場所である付属 CD の FtdiDrivers/VcpDriver を選択して下さい．

4. 2 インストール作業完了の確認

コントロールパネルのデバイスマネージャを開いてください. **USB (Universal Serial Bus) コントローラ** と **ポート** の項目に, 次の表示が追加されているか確認してください.



4. 3 ドライバのアンインストール

インストールした仮想 COM ポートドライバをアンインストールする方法について説明します.

仮想 COM ポートドライバが格納されているフォルダに次のようなアイコンがあります. **PC と EDX-001 を接続しない状態で**, このアイコンをダブルクリックし, 実行してください.

FTDIUNIN.exe

▼ [Continue] をクリックします.

▼ [Finish] をクリックして, 完了です.

5 コンフィグレーションと通信実験

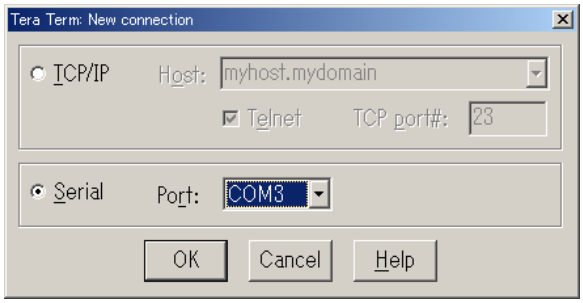
USB で通信実験をする方法を紹介します．ここで利用するダウンロード回路は，10 章で説明するサンプル回路4とします．また，USB ケーブルを外す際には，必ず**通信ソフトを終了**させてください．

フリーウェアである TeraTerm を利用します．このソフトは非常に便利な機能が充実しています．以下の URL からダウンロードし，一度お試し下さい．

http://hp.vector.co.jp/authors/VA002416/

5．1 TeraTerm の起動


をクリックして下さい．新規の接続の場合には次のようなウィンドウが表示されます．



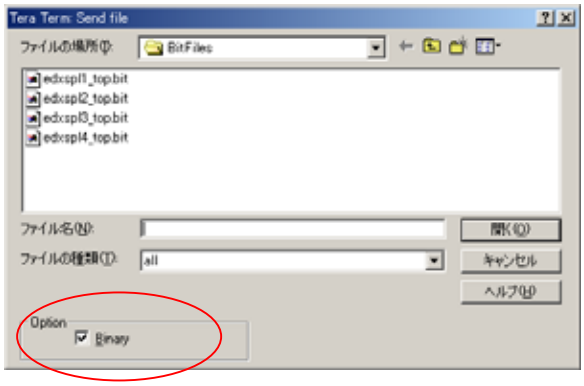
デバイスマネージャの**ポートの項目**を確認し，オープンするポート番号と一致させて下さい．


USB Serial Port/HuMANDATA LTD. (COM3)

上記のような状態であれば，ポート番号は 3 となります．また，番号を変更したい場合には上の [USB Serial Port] を右クリックし，[プロパティ] を選択することで設定の変更が行えます．

5．2 TeraTerm による FPGA コンフィグレーション

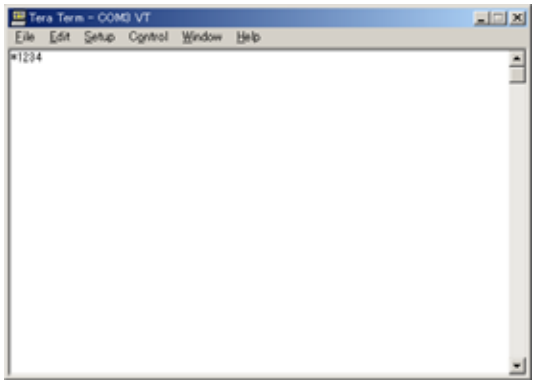
[File] – [Send file ..] を選択します．



EDX-001 の INIT ボタンを押し，[バイナリ] にチェックを入れます．ファイルは edxspl4_top.bit を選択し，[開く] をクリックします．

完了すると EDX-001 の DONE LED が点灯します．

5．3 TeraTerm による通信実験



次に，“*1234” と入力します．ウィンドウには“*1234” と表示されます．これは，キーボードから入力した文字をそのまま表示しているのではなく，USB ケーブル経由で，エコーされたデータです．

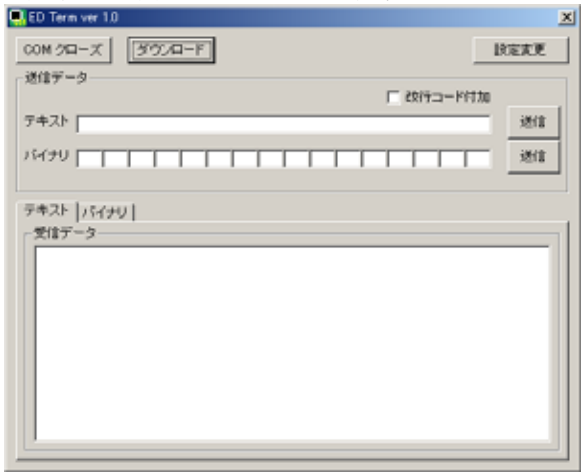
ドットマトリックスディスプレイにも“1234”と表示されます．‘*’は頭だしに利用しています．

5．4 ED Term による FPGA コンフィグレーション

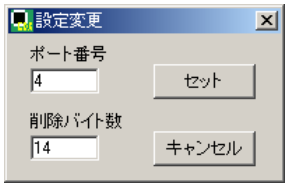
Visual Basic で作成したダウンロードソフトです．ソースコードはすべて公開しておりますので，機能の追加修正も容易に行えます．

付属 CD の EDterm/pkg の setup.exe を実行してください．PC にインストールされます．

起動すると次のようなウィンドウが表示されます．



ポート番号が 3 ではない場合には，[設定変更]をクリックし，次の ダイアログから，ポート番号を変更して下さい．変更が終わったら，[セット] をクリックします．



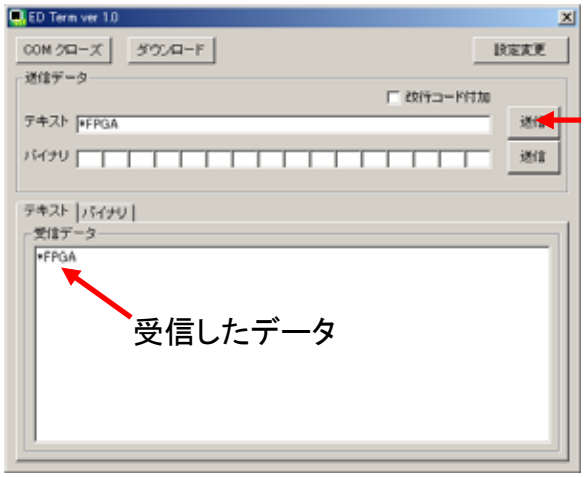
[COM オープン] をクリックします．次に [ダウンロード] をクリックし，bit ファイルを選択します．

完了すると EDX-001 の DONE LED が点灯します．

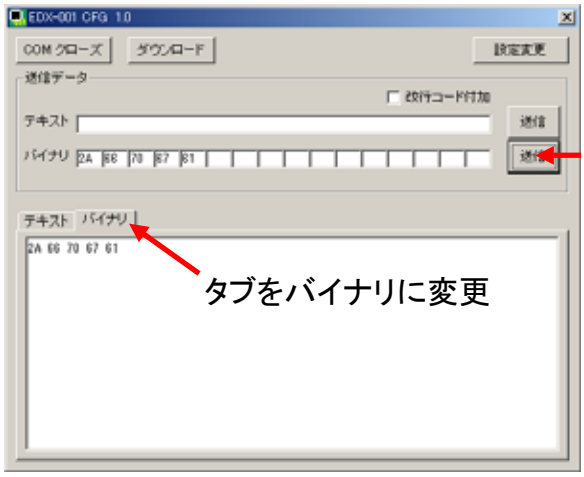
5．5　ED Term を利用した通信実験

以下のように，“*FPGA” と入力し，テキストデータを送信します．エコー回路なので，“*FPGA”という文字をそのまま受信します．EDX-001 に搭載されているドットマトリックスディスプレイには“FPGA” と表示されます．

「改行コード付加」にチェックを入れると，テキストデータの末尾に 0x0D，0x0A を付加して送信します．



同様の操作をバイナリデータとして表示させることも可能です．“*fpga” はバイナリデータとして表現すると，2A(*)，66(f)，70(p)，67(g)，61(a) ですので，次のように入力し，送信します．

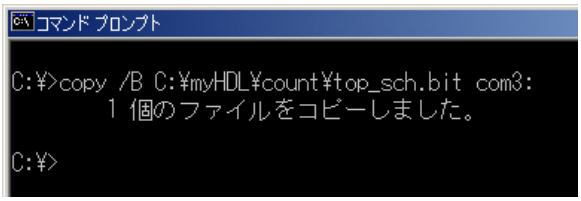


送信したデータが FPGA 内部回路によって，エコーされているのが確認できます．EDX-001 のドットマトリックスディスプレイには小文字で，“fpga” と表示されます．

5．6　COPY コマンド

コマンドプロンプトから COPY コマンドを利用することで，FPGA へのコンフィグレーションが可能です．ビットファイルの場所を指定して，COM3 に送信します．
バイナリデータとして，送信するので**/B** をつけてください．

C:¥>copy **/B** C:¥myHDL¥count¥top_sch.bit com3:



この方法は環境によって，**動作しない場合**があります．

6 ISE による開発手順

EDX-001 に搭載されている FPGA をコンフィグレーションするには bit ファイルが必要となります．このファイルを生成する手順について説明します．

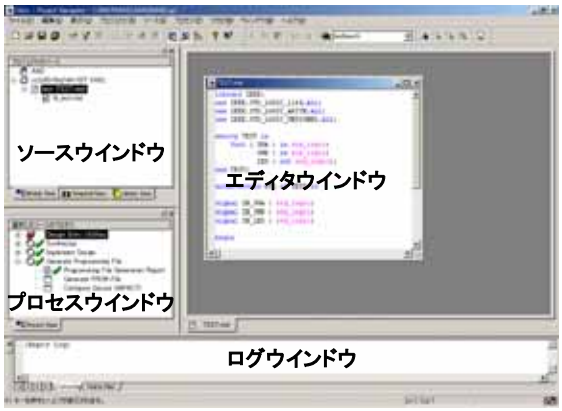
bit ファイルを生成するには次の開発ツールが必要となります．

- WebPack ISE もしくは Foundation ISE
- Model Sim

7セグメント LED を1つ使用し，1 秒おきに 0 から 9 までカウントアップする回路を VHDL と回路図で作成します．

6．1 Project Navigator の起動

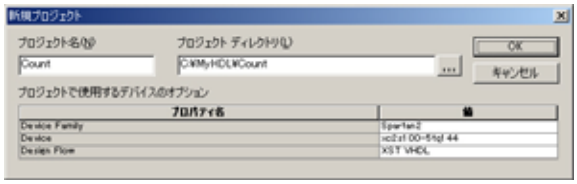
[スタート] → [プログラム] → [ザイリンクス] → [Project Navigator] を起動します．各ウインドウの名称は次のようになっています．



6．2 プロジェクトの新規作成

プロジェクトを新規作成するには，[ファイル] → [新規プロジェクト] をクリックします．ここで，ディレクトリとプロジェクト名，ターゲットとなるデバイスも選択します．

デバイスは変更が可能です．ソースウィンドウのデバイスの名前をフォーカス状態にし，右クリックで出現するダイアログの [プロパティ] を選択すると，新規作成と同様に以下のダイアログボックスが利用できます．



プロジェクトディレクトリ は「C:¥MyHDL」とします．プロジェクト名を「Count」と入力すると，自動的に「C:¥MyHDL¥Count」 に変更されます．

EDX-001 に搭載されているデバイスに変更します．次の値に変更してください．

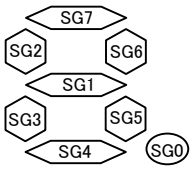
プロパティ名	値
Device Family	Spartan2
Device	xc2s**100**-5tq144
Design Flow	XST VHDL

6．3 VHDL によるデコーダの作成

これは4 ビットのデータを7セグメント LED で表示するモジュールです．

[プロジェクト] → [新規ソース] を選択し，ソースタイプを以下のように VHDL Module として下さい．

ファイル名は「SEG_DEC」とします．[次へ] をクリックし，[完了] をクリックします．



組み合わせ回路として，4 ビットの入力に対して，7 ビットの出力を得るように設計します．0 で点灯すると考えれば，次のような対応になります．

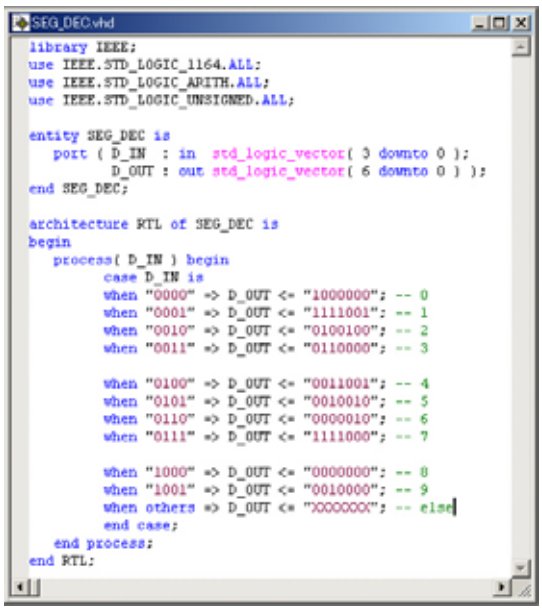
ここでは，小数点「SG0」は利用しません．

入力　4 ビット	SG1, 2, 3, 4, 5, 6, 7
0000	1000000
0001	1111001
0010	0100100
0011	0110000
0100	0011001
0101	0010010
0110	0000010
0111	1111000
1000	0000000
1001	0010000

FPGA のピン番号は次表のようになります．ピン番号を固定する方法は 6.13 UCF ファイルの項で説明します．

ピン名	FPGA のピン番号
SG7	29 番
SG6	30 番
SG5	31 番
SG4	40 番
SG3	41 番
SG2	42 番
SG1	43 番
SG0	44 番

このデコーダ全体の VHDL ソース（SEG_DEC.vhd）は次のようになります．

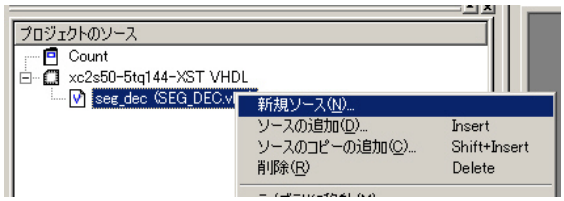


6. 4 デコーダのテストベンチ波形を作成

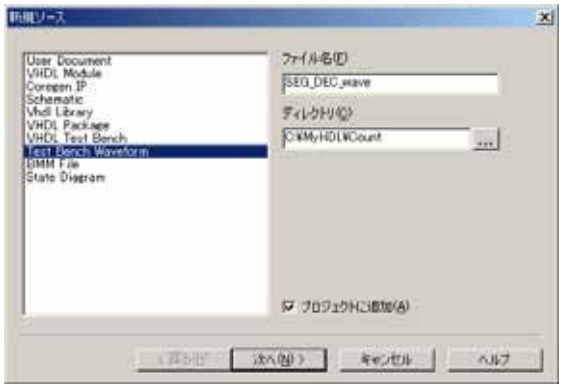
SEG_DEC.vhd のシミュレーションを行う前に，テストベンチ波形を作成します．これは記述したファンクションが仕様を満たしているか，テスト信号によって確認するためのものです．

テストベンチ波形は GUI ツールにて波形を描く方法と，言語で記述する方法があります．先に GUI ツールで波形を描く方法について先に説明します．

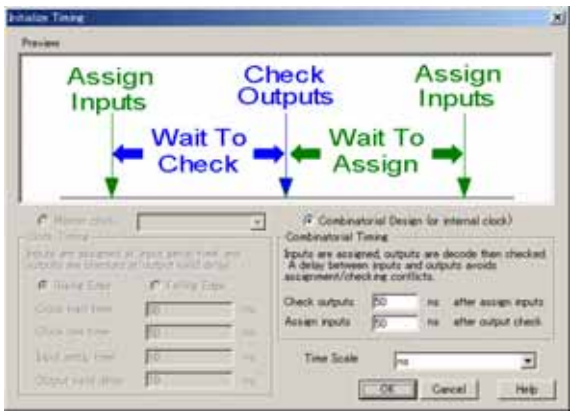
SEG_DEG. を選択し，右クリックします．[新規ソース] を選択します．



ソースタイプは [Test Bench Waveform] を選択します．ファイル名は VHDL ソース (SEG_DEC.vhd) と異なる名前にしてください．ここでは，SEG_DEC_wave とします．

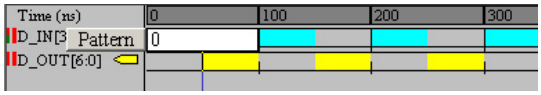


[次へ] を押し，[完了] を選択します．HDL Bencher が起動します．組み合わせ回路なので，クロック周期等の定義はできません．デフォルトのまま [OK] を押します．



テストベンチ波形が表示されます．入力波形は青色で，出力波形の黄色となっています．

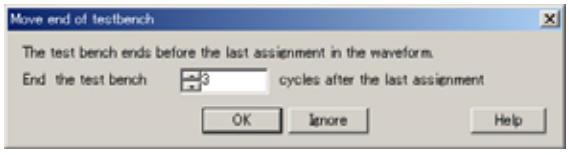
青色の部分をクリックすると，以下のように [Pattern] と表示され，数値を入力できるようになります．



次のスイッチボタンを変更することで，16 進，10 進，2 進の数値で入力可能です．

次のように 0 から 10 まで D_IN を定義します．10 を入力したときの出力は不定値 (X) になるはずです．

右上の[Save Waveform]を押すと，次のダイアログボックスが表示されます．反転させた末尾のデータから，何サイクルを有効とするか入力します．ここでは，D_IN の「10」から 3 サイクルとしておきます．

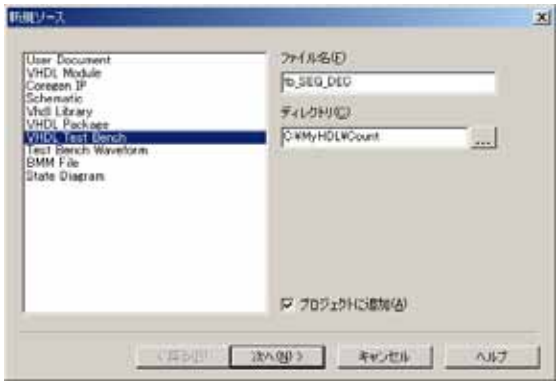


これで，テストベンチの波形は完成です．

6. 5 デコーダの VHDL テストベンチを作成

HDL Bencher で作成した，波形と同じものを言語で記述してみます．同様にして，新規ソースを選択します．

ソースタイプは [VHDL Test Bench]を選択し，ファイル名は「tb_SEG_DEC」とします．



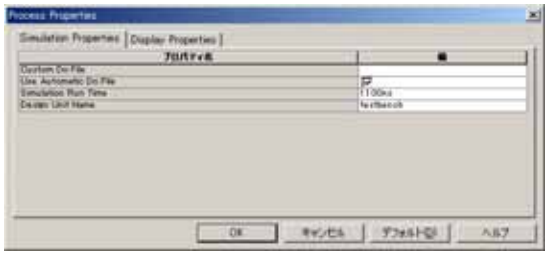
[完了] を押すと，VHDL テストベンチの雛型が作成されます．以下のように，緑色のコメントで囲まれた部分にコードを追加します．

```
-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
      D_IN <= "0000";  -- 0
      wait for 100 ns;
      D_IN <= "0001";  -- 1
      wait for 100 ns;
      D_IN <= "0010";  -- 2
      wait for 100 ns;
      D_IN <= "0011";  -- 3
      wait for 100 ns;
      D_IN <= "0100";  -- 4
      wait for 100 ns;
      D_IN <= "0101";  -- 5
      wait for 100 ns;
      D_IN <= "0110";  -- 6
      wait for 100 ns;
      D_IN <= "0111";  -- 7
      wait for 100 ns;
      D_IN <= "1000";  -- 8
      wait for 100 ns;
      D_IN <= "1001";  -- 9
      wait for 100 ns;
      D_IN <= "1010";  -- 10
      wait; -- will wait forever
   END PROCESS;
-- *** End Test Bench - User Defined Section ***
```

自動で生成されたテストベンチの雛型のエンティティ名は，「testbench」となっています．

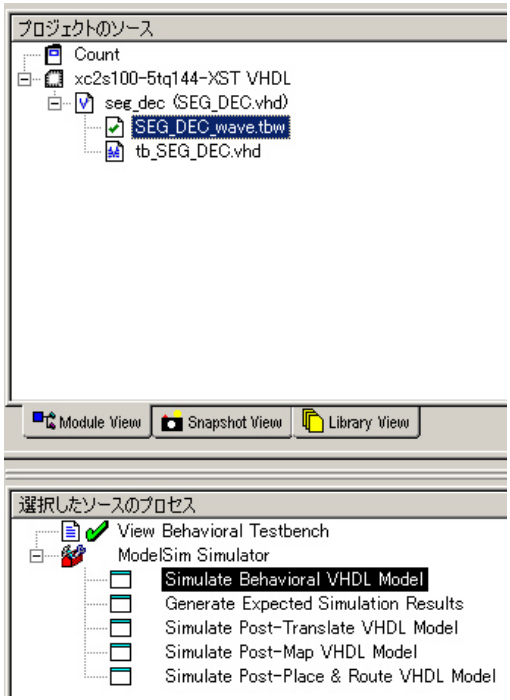
```
ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS
```

ソースウインドウで「tb_SEG_DEC」，プロセスウインドウで，「Simulate Behavioral VHDL Module」を選択し，プロパティダイアログを表示してください．Design Unit Name は，テストベンチのエンティティ名と同じにしてください．また，Simulation Run Time は「1100 ns」とします．



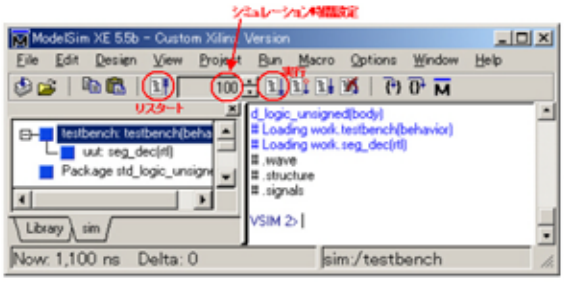
6．6 デコーダのシミュレーション実行

HDL Bencher で作成したテストベンチを選択し，プロセスウインドウの [Simulate Behavioral VHDL Module] をダブルクリックします．ModelSim を実行すると，複数のウインドウが起動します．



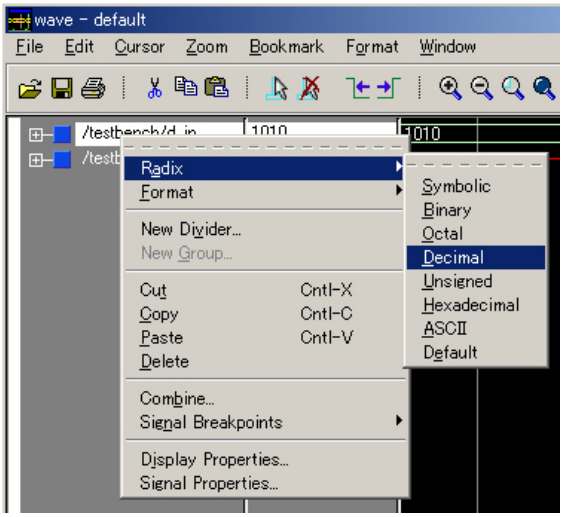
正しく動作しているか波形を確認します．

VHDL で記述したテストベンチも同様にして，ModelSim を起動します．シミュレーション時間を変更するには，メインウインドウの時間設定を変更します．



また，コマンドで「run xx ns」と入力することもできます．再実行するには「restart」と入力するか，ツールバーを使用してください．波形ウインドウのツールバーは次のようになっています．

波形を2進数表示から10進等で表示させる場合には，信号の［右クリック］→［Radix］で切り替え可能です．



6．7　VHDL による分周回路の作成

18.432MHz のクロックを分周して，約1秒を生成する回路を設計します．名前は CLKDIV とします．

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

entity CLKDIV is
  --▼ 1/18432000 カウンタ
  --    ビット幅は 25
  generic ( WIDTH : integer := 25;
            N : integer := 18432000 );
  --▼ 入出力ポート
  port    ( CLK   : in  std_logic;
            RST   : in  std_logic;
            SEC   : out std_logic );
end CLKDIV;

architecture RTL of CLKDIV is
-- ▼ 初期値/上位層から WIDTH を受け取る
signal CNT : std_logic_vector( WIDTH - 1 downto
0 );

begin
--▼ CLK, RST の変化をみる
process( CLK, RST ) begin
    if( RST = '1') then
        CNT     <= (others=>'0');
        SEC <= '0';
    --▼ CLK の立ち上がりで動作
    elsif( CLK'event and CLK = '1') then
        --▼ 18432000 に到達
        if( CNT = N - 1 ) then
            CNT <= (others=>'0');
            SEC <= '1';
        else
            CNT <= CNT + '1';
            SEC <= '0';
        end if;
    end if;
end process;
end RTL;
```

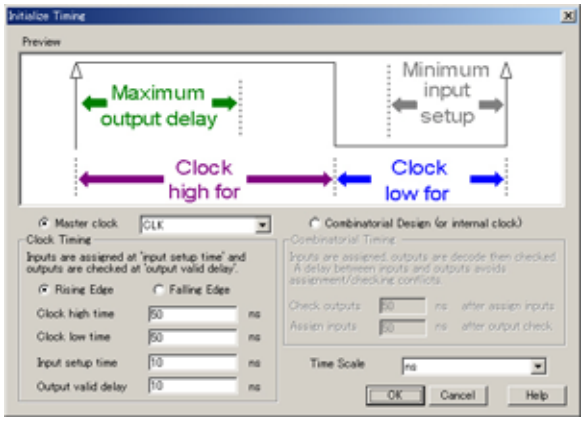
CNT が 18432000 になるまで，シミュレーションするのは大変なので，シミュレーション用のコードを追加します．

```
--▼ 入出力ポート
port    ( CLK   : in  std_logic;
          RST   : in  std_logic;
          LOAD  : in  std_logic; --追加
          SEC   : out std_logic );

--▼ 18432000 に到達
        if( CNT = N - 1 ) then
            CNT <= (others=>'0');
            SEC <= '1';
        elsif( LOAD = '1') then  --追加
            CNT <= "1000110010011111111111111";
        else
            CNT <= CNT + '1';
            SEC <= '0';
```

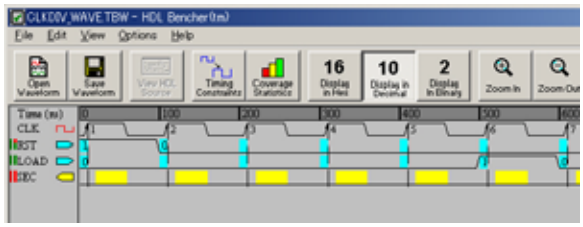
6．8　分周回路のテストベンチ波形を作成

CLKDIV_wave として，HDLBench を使用して，テストベンチを作成します．今回は組み合わせ回路ではないので，クロックを定義しなければなりません．

モジュールのファンクションを確認するだけなので，デフォルトのままで「OK」を押します．



初期状態を定義するために，以下のように RST を「H」にします．SEC の変化を確認するために記述した LOAD を「H」にし，ファイルを保存します．



6．9　分周回路の VHDL テストベンチを作成

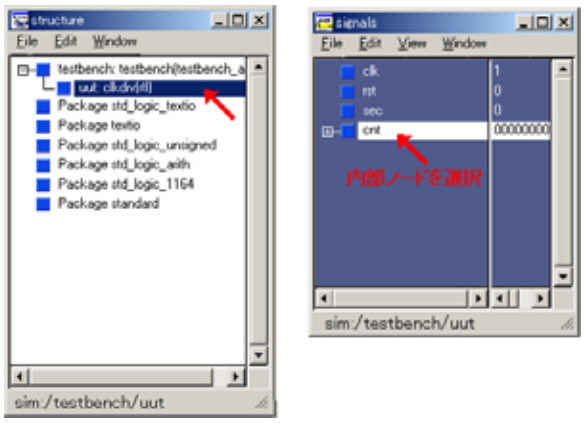
HDLBench で作成したテストベンチを VHDL で記述した場合，次頁のようになります．

CLK <= not CLK; と記述したときに初期値が定義されていないと不定値となりますので，注意してください．テストベンチのより詳しい記述のしかたは文法書を参照してください．

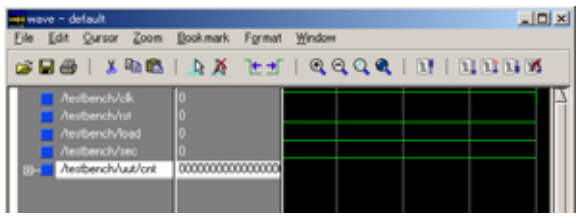
```
COMPONENT clkdiv
PORT(   CLK  : IN std_logic;
        RST  : IN std_logic;
        LOAD : IN std_logic;
        SEC : OUT std_logic );
END COMPONENT;
--▼初期値を定義する
SIGNAL CLK  :  std_logic := '0';
SIGNAL RST  :  std_logic;
SIGNAL LOAD :  std_logic;
SIGNAL SEC  :  std_logic;

BEGIN
uut: clkdiv PORT MAP(
        CLK => CLK,
        RST => RST,
        LOAD => LOAD,
        SEC => SEC );

-- *** Test Bench - User Defined Section ***
   tb : PROCESS
   BEGIN
        RST <= '1'; LOAD <= '0';
        --▼100ns 待つ
        wait for 100 ns;
        RST <= '0';
        wait for 500 ns;
        LOAD <= '1';
        wait for 100 ns;
        LOAD <= '0';
        wait; -- will wait forever
   END PROCESS;

   --▼クロックを定義する
   process begin
        --▼ 50ns ごとに反転する
        wait for 50 ns;
        CLK <= not CLK;
   end process;
-- *** End Test Bench - User Defined Section ***
```

6. 10　分周回路のシミュレーション

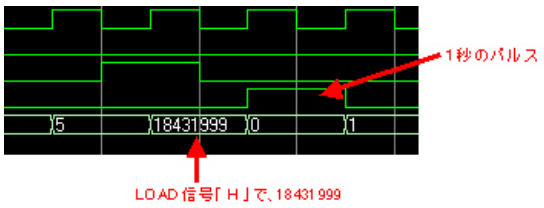
内部ノード cnt を確認するので，以下のように Structure ウインドウと Signal ウインドウから cnt を選択してください．



内部ノード cnt を波形ウインドウに，ドラッグ＆ドロップします．cnt が追加されます．



シミュレーションをリスタートします．cnt が ”00…00”と表示されているとわかりにくいので，[右クリック] → [Radix] → [unsigned] とします．CNT が 18431999 まで，数え sec が 1 周期だけ「H」になっています．約 1 秒のパルスを生成されていることを確認できました．

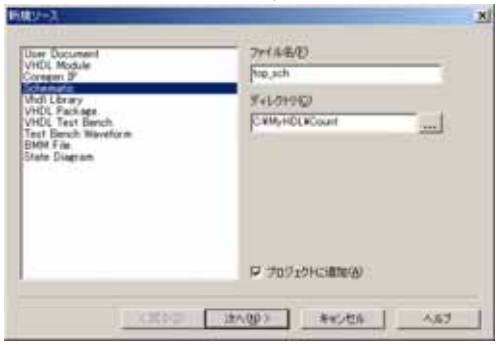


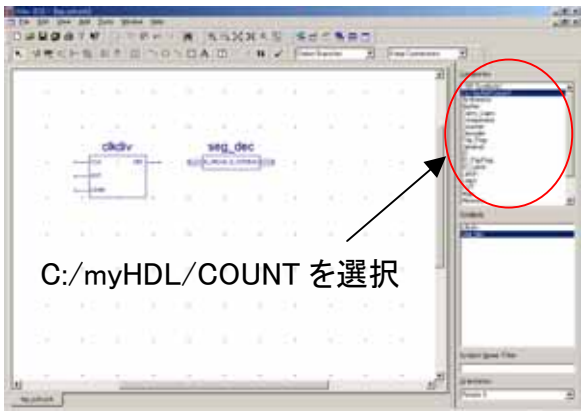
6. 11　回路図シンボルの作成

デコーダと分周回路を回路図で使用できるようにします．各モジュールを選択し，「Create Schematic Symbol」を実行します．両方のモジュールで行ってください．

6. 12　最上位回路図の作成

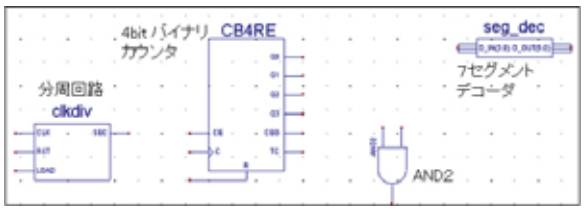
ファイルの新規作成を行います．ファイルタイプとファイル名を「Schematic」と「top_sch」にしてください．





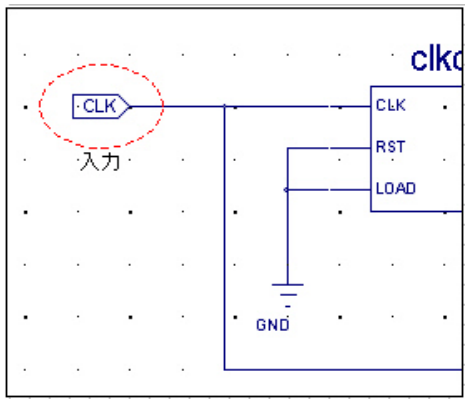
回路図エントリーツールの右上の [Categories] には c:/myHDL/COUNT を選択し，[Symbols] リストから clkdiv と seg_dec を選びます．

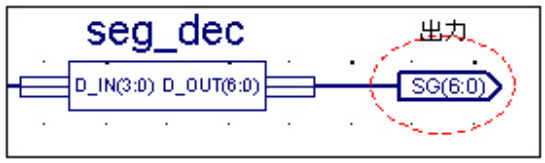
10 進カウンタを回路図上で作成するので，[CB4RE] と [AND2] を回路図の [Symbols] より選択し，配置してください．



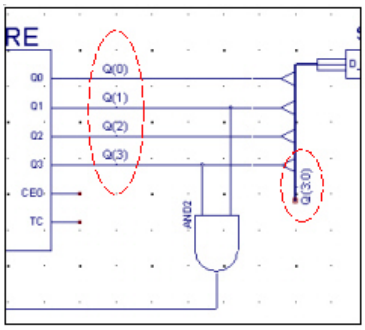
回路図全体のように，[ワイヤの追加]で結線してください．バスも [ワイヤの追加] で行えます．ESC を押すことでモードを終了することができます

次に I/O マーカーの追加をします．入力のワイヤを CLK とし，出力のバスを SG(6:0) と名前をつけます．ダブルクリックして，SG(6:0) は [output] と属性を変更してください．



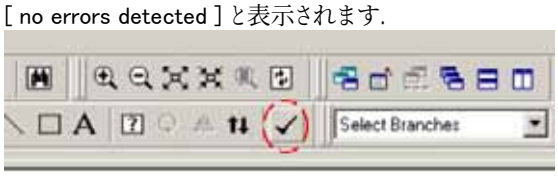


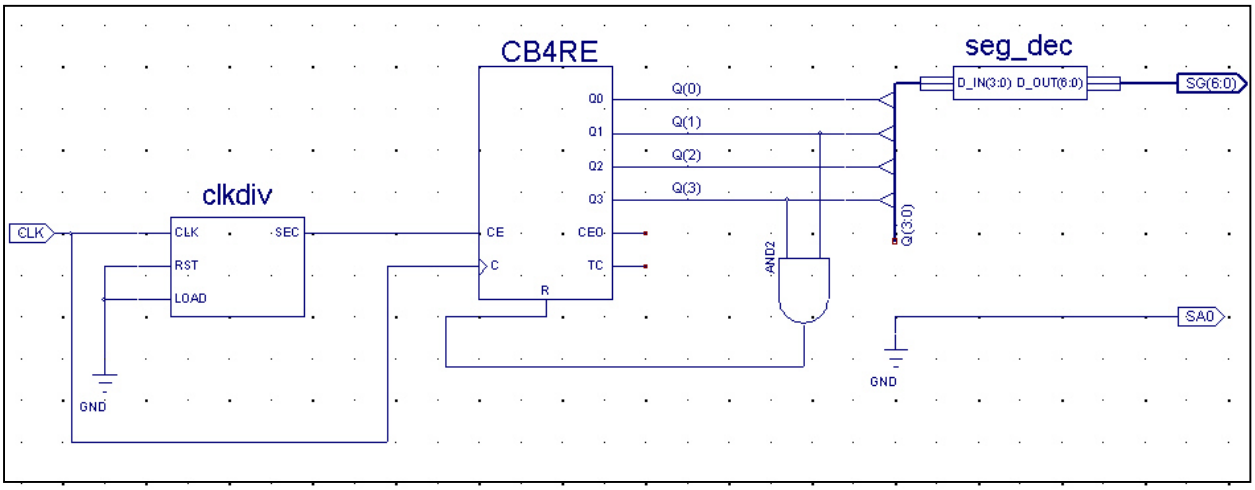
10 進カウンタとデコーダの間のバスに名前を追加します．



4 つの 7 セグメントのうち SA0 を点灯させるために，SA0 を Low にします．

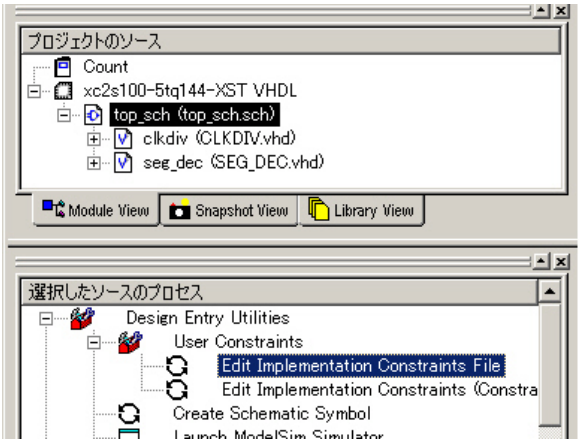
最後に回路図のチェックを行います．ツールバーの以下のボタンを押してください．回路図に問題がなければ，[no errors detected] と表示されます．





6．13　UCF ファイルの作成

FPGA のピンを固定するために，UCF ファイルを作成します．top_sch の [Edit Implementation Constraints File] を実行してください．メモ帳が起動します．



行の先頭に“#”を付けるとコメント文となります．

```
NET "CLK"　LOC = "P88";
NET "SG<0>" LOC = "P44";
NET "SG<1>" LOC = "P43";
NET "SG<2>" LOC = "P42";
NET "SG<3>" LOC = "P41";
NET "SG<4>" LOC = "P40";
NET "SG<5>" LOC = "P31";
NET "SG<6>" LOC = "P30";
#▼SG<7>は使用しないのでコメント
#NET "SG<7>" LOC = "P29";
#▼7セグを選択
NET "SA0" LOC = "P46";
#NET "SA1" LOC = "P47";
#NET "SA2" LOC = "P48";
#NET "SA3" LOC = "P49";
```

6．14　論理合成の実行

ターゲットとなる回路図である top_sch を選択します．プロセスウインドウの [Synthesize] をダブルクリックします．

この作業によって，RTL の HDL 記述から，論理設計段階の回路図が生成されます．回路図にオープンのピンが存在するので，**黄色マーク**（ワーニング）がつきます．原因がわかっているので，このまま放置します．

6．15　配置配線の実行

[Implemant Design] をダブルクリックします．

プロジェクトの新規作成時に選択した FPGA に合わせて配置配線されます．clkdiv や seg_dec を選択して，論理合成/配置配線をしないようにしてください．その場合，目的の回路が生成されません．

6．16　プログラミングファイルの生成

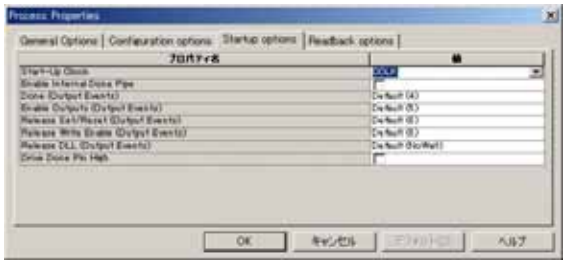
EDX-001 に搭載している FPGA(XC2S100)のコンフィグレーションに必要なプログラミングファイルを生成します．

[Generate Programming File] をダブルクリック

正常に終了すれば，ログウインドウに
Done: completed successfully.
と表示されます．

c:¥myHDL¥count 内に **top_sch . bit** というプログラミングファイルができました．

注意）
[Generate Programming File] のプロパティの[Start-UP Clock] を [Jtag Clock] とした場合はコンフィグレーションできません．デフォルトの CCLK としてください．



6．17　EDX-001 へのダウンロード

EDX-001 に実装されている FPGA のコンフィグレーションを行うために，仮想 COM ポートドライバを必ず PC にインストールしてください(4章参照)．

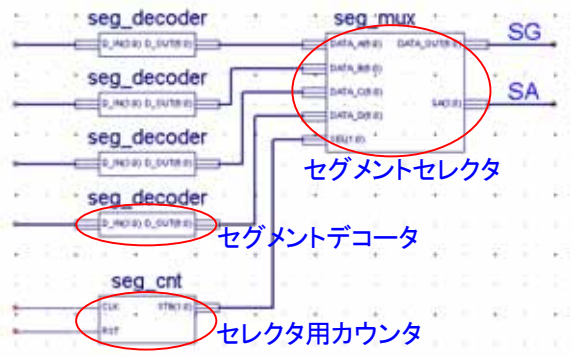
コンフィグレーションの方法については，5章を参照してください．コンフィグレーションが完了すると，DONE-LED が点灯し，7 セグメント LED が 0 から 9 までカウントアップしているのが確認できます．

新たなデータを FPGA にダウンロードするには，INIT スイッチを押して，初期化してください．

7 ダイナミック点灯

前項では 7 セグメント LED を 1 つの場合について説明しました．本項では複数の LED を同時に点灯させる方法について説明します．

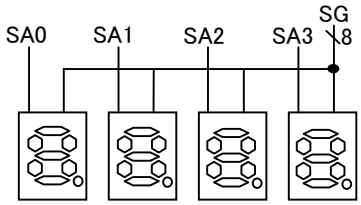
ダイナミック点灯の回路図は以下のようになります．



SA0 から SA3 のうち，1 つを 0 (Low) にし，それ以外をハイインピーダンスにします．SA が 0 (Low) のときの 7 セグメントだけが表示されます．この SA に合わせて，各桁の SG (セグメントデータ) も出力させます．

この動作を数 kHz で動作させれば全体が同時に，点灯しているように見えます．

ここでは小数点は使用しません．



7．1 セグメントデコータ

このモジュールは前項で説明した VHDL ソースと同じです．

7．2 セレクタ用カウンタ

▼SEG_CNT

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

entity SEG_CNT is
--▼表示させる桁数を指定 0～4
  generic ( MAX_SEG : integer := 4 );
  port ( CLK  : in  std_logic;
         RST  : in  std_logic;
         SEL  : out std_logic_vector( 1 downto 0 ) );
end SEG_CNT;

architecture RTL of SEG_CNT is
--▼CLK が 18.432MHz のとき CE は 2.25kHz のパルスとなる
constant MAX_DIVCNT : integer := 4095;

signal CNT    : std_logic_vector( 1 downto 0 );
signal DIVCNT : std_logic_vector(11 downto 0 );
signal CE     : std_logic;

SEL <= CNT;

process( CLK, RST ) begin
  if( RST = '1') then
    CNT <= "00";
  elsif( CLK' event and CLK = '1' ) then
    --▼2.25kHz ごとにカウントアップ
    if( CE = '1') then
      if( CNT = MAX_SEG - 1 ) then
        CNT <= "00";
      else
        CNT <= CNT + '1';
      end if;
    end if;
  end if;
end process;

--▼クロックイネーブルを生成する
process( CLK, RST ) begin
  if( RST = '1') then
    DIVCNT <= (others=>'0');
  elsif( CLK'event and CLK = '1') then
    if( DIVCNT = MAX_DIVCNT ) then
      DIVCNT <= (others=>'0');
      CE <= '1';
    else
      DIVCNT <= DIVCNT + '1';
      CE <= '0';
    end if;
  end if;
end process;

end RTL;
```

桁数を変更するには，generic (MAX_SEG : integer := 4); を“1”から“4”に変更してください．“4”の場合には，SEL が“00→01→10→11→00”とカウントアップします．“3”の場合には，SEL が“00→01→10→00”となります．

7．3 セグメントセレクタ

セレクタ用カウンタが 00→01→10→11 になることを利用して，セグメントデコータを作成します．

SEL（入力）	SA0～3（出力）	SG0～6（出力）
00	0ZZZ	DATA_A
01	Z0ZZ	DATA_B
10	ZZ0Z	DATA_C
11	ZZZ0	DATA_D

▼SEG_MUX

```
--ライブラリの宣言は省略
entity SEG_MUX is

port( DATA_A,
      DATA_B,
      DATA_C,
      DATA_D : in  std_logic_vector( 6 downto 0 );
      SEL    : in  std_logic_vector( 1 downto 0 );
      DATA_OUT : out std_logic_vector( 6 downto 0 );
      SA     : out std_logic_vector( 3 downto 0 ) );
end SEG_MUX;
```

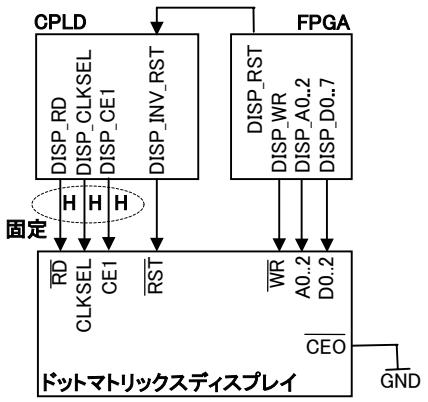
```
architecture RTL of SEG_MUX is
begin
--▼SG を定義
   DATA_OUT <= DATA_A when SEL = "00" else
               DATA_B when SEL = "01" else
               DATA_C when SEL = "10" else DATA_D;
--▼SA を定義
   SA       <= "0ZZZ" when SEL = "00" else
               "Z0ZZ" when SEL = "01" else
               "ZZ0Z" when SEL = "10" else
               "ZZZ0" when SEL = "11" else "ZZZZ";
end RTL;
```

組み合わせ回路をそのまま出力としているので，開発ツールによってはハザードの警告がでます．7セグメント LED なので考慮していませんが，D-FF を挿入すれば解決できます．

8 ドットマトリックスディスプレイの点灯

ドットマトリックスディスプレイ(PD4435)に文字を表示させる手順について説明します．英数字や記号 (8 ビットのアスキーコード) を表示させることができます．

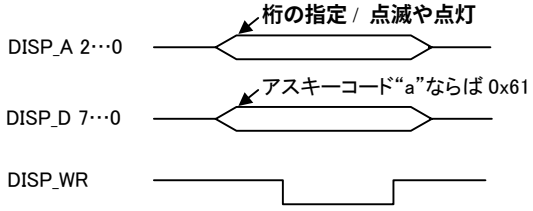
例えば，“A”であれば，0x41 を DISP_D0～7 に出力します．



DISP_A と DISP_WR 以外の制御信号は CPLD 内部で 1(High)に固定してありますので，簡単に制御することができます．リセット信号である，DISP_RST は**正論理**です．

FPGA で制御する信号は DISP_A，DISP_WR のみです．リセット信号である DISP_RST はドットマトリックスディスプレイを使用しない場合を考慮して，CPLD にインバータを挿入してあるので，電源投入時は無表示となります．

タイミングのイメージは次のようになります．



DISP_A0 と A1 は4つの桁のうちどの桁に表示させるか指定します．DISP_A2 は点滅や点灯の指定します．

DISP_WR は 25℃時に 70ns 以上を Low とします．より詳しいタイミングはデータシートを参照してください．

▼ アドレス指定

Address			Contents
A2	A1	A0	
0	X	X	Control Word
1	0	0	Digit 0 (rightmost)
1	0	1	Digit 1
1	1	0	Digit 2
1	1	1	Digit 3 (leftmost)

上記のデータシートより，右端を点灯させるには A2 を H とし，A1 と A0 を L としてください．

▼ コントロールワード時

D7	D6	D5	D4	D3	D2	D1	D0	Operation
0	0	X	X	X	X	0	0	Blank
0	0	X	X	X	X	0	1	25% brightness
0	0	X	X	X	X	1	0	50% brightness
0	0	X	X	X	X	1	1	Full brightness

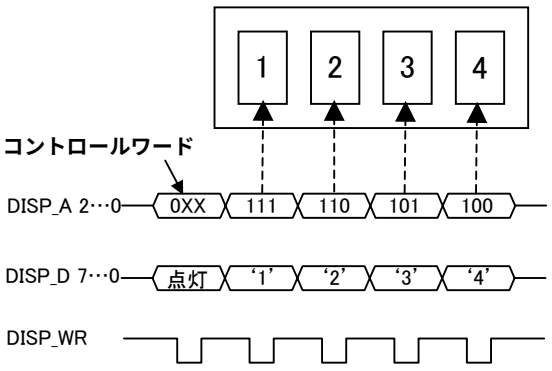
X=don't care

DISP_A2, 1, 0 を 0, X, X とした時に，DISP_D7～0 を上記のように指定すれば，点滅等が行えます．

8．1 表示手順

実際に文字を表示させる手順について説明します．“＊”で表示開始とし，つづけて4文字のアスキーコードを表示させる VHDL モジュールを作成します．

ドットマトリックスディスプレイに対しての出力波形イメージは次のようになります．



‘＊’を認識して，頭だしとコントロールワードの点灯(Full brightness)を指示します．このときの DISP_A2…0 には ”0XX” を出力し，DISP_D7…0 は“00XXXX11” を出力します．

USB からの制御することとし，USB_DAT と，そのリードイネーブル信号 USB_RE を入力とします．

DISP_WR を Low に 70ns にする期間ですが，クロックが 18.432MHz の場合には，1 周期が約 54ns なので，厳密には分周クロックをお使いください．

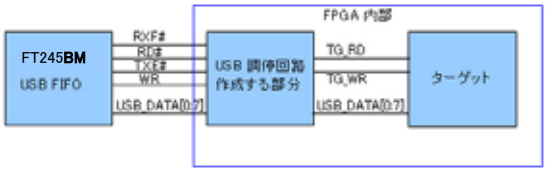
8．2 表示モジュール

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

entity PD_LED is
Port ( CLK : in std_logic;
       RST : in std_logic;
       USB_DAT : in std_logic_vector( 7 downto 0 );
       USB_RE  : in std_logic;
       DISP_WR_N : out std_logic;
       DISP_ADR  : out std_logic_vector( 2 downto 0 );
       DISP_DAT  : out std_logic_vector( 7 downto 0 ));
end PD_LED;

architecture RTL of PD_LED is

signal STATE : std_logic_vector( 3 downto 0 );

--▼ステート番号固定の場合はすべて記述
constant S0 : std_logic_vector := "0000";
constant S1 : std_logic_vector := "0001";
constant S2 : std_logic_vector := "0010";
constant S3 : std_logic_vector := "0011";
constant S4 : std_logic_vector := "0100";
constant S5 : std_logic_vector := "0101";
constant S6 : std_logic_vector := "0110";
constant S7 : std_logic_vector := "0111";
constant S8 : std_logic_vector := "1000";
constant S9 : std_logic_vector := "1001";
constant S10: std_logic_vector := "1010";
constant S11: std_logic_vector := "1011";
constant S12: std_logic_vector := "1100";
constant S13: std_logic_vector := "1101";
constant S14: std_logic_vector := "1110";
constant S15: std_logic_vector := "1111";

--▼ ' * ' はアスキーコード : 0x2A
constant AST: std_logic_vector := "00101010";

begin
process( CLK, RST ) begin
   if ( RST ='1') then
      DISP_DAT   <= "00000000";
      DISP_ADR   <= "000";
      DISP_WR_N  <= '1';
      STATE      <= S0;
   elsif( CLK'event and CLK='1') then
      case STATE is
         when S0 =>
            --▼USB からの信号が ' * '
            if( USB_RE='1' and USB_DAT=AST ) then
               --▼Full brightness とする
               DISP_DAT <= "00XXXX11";
               DISP_ADR <= "0XX";
               DISP_WR_N <= '1';
               STATE <= S1;
            else
               STATE <= S0;
            end if;
         when S1 =>
            DISP_WR_N <= '0';
            STATE <= S2;
         when S2 =>
            DISP_WR_N <= '1';
            STATE <= S3;
         when S3 =>
            if( USB_RE='1') then
               --▼USB からの信号を代入
               DISP_DAT  <= USB_DAT;  --１文字目
               DISP_ADR  <= "111";
               DISP_WR_N <= '1';
               STATE <= S4;
            else
              STATE <= S3;
            end if;
         when S4 =>
            DISP_WR_N <= '0';
            STATE <= S5;
         when S5 =>
            DISP_WR_N <= '1';
            STATE <= S6;
         when S6 =>
            if( USB_RE='1') then
               DISP_DAT  <= USB_DAT; --２文字目
               DISP_ADR  <= "110";
               DISP_WR_N <= '1';
               STATE <= S7;
            else
               STATE <= S6;
            end if;
         when S7 =>
            DISP_WR_N <= '0';
            STATE <= S8;
         when S8 =>
            DISP_WR_N <= '1';
            STATE <= S9;
         when S9 =>
            if( USB_RE='1') then
               DISP_DAT  <= USB_DAT; --３文字目
               DISP_ADR  <= "101";
               DISP_WR_N <= '1';
               STATE <= S10;
            else
               STATE <= S9;
            end if;
         when S10 =>
            DISP_WR_N <= '0';
            STATE <= S11;
         when S11 =>
            DISP_WR_N <= '1';
            STATE <= S12;
         when S12 =>
            if( USB_RE='1') then
               DISP_DAT  <= USB_DAT; --４文字目
               DISP_ADR  <= "100";
               DISP_WR_N <= '1';
               STATE <= S13;
            else
               STATE <= S12;
            end if;
         when S13 =>
               DISP_WR_N <= '0';
               STATE <= S14;
         when S14 =>
               DISP_WR_N <= '1';
               STATE <= S15;
         when others => STATE <= S0;
      end case;
   end if;
end process;
end RTL;
```

9 USB 通信実験

EDX-001 で USB を利用するための，調停回路について説明します．

▼ブロック図

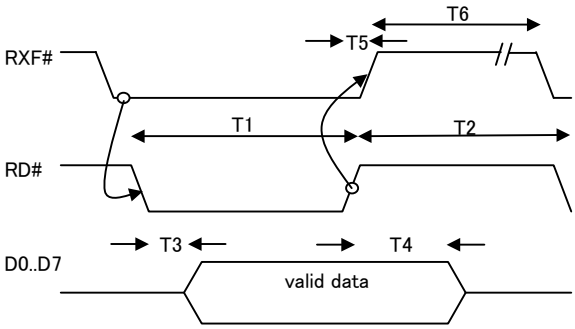


9．1 リードサイクルのタイミングチャート

リードサイクルですが， PC から USB-001 に文字を送ると FIFO(FT245BM)にデータ格納され，RXF#は Low になります．

RD#を下げ，立ち上がりで FIFO は1つのデータが読まれたことを知ります．ここで，FIFO の中身が，空であれば，RXF#は HIGH の状態を維持します．

D0..D7 を読むタイミングは RD#の下がりから T3(30ns)，RD#が立ち上がり T4(10ns)の間が有効となります．



Time	Description	Min	Max
T1	RD Active Pulse Width	50	
T2	RD to RD Pre-Change Time	50+T6	
T3	RD Active to Valid Data		30
T4	Valid Data Hold Time from RD inactive	10	
T5	RD inactive to RXF#	5	25
T6	RXF inactive after RD cycle	80	

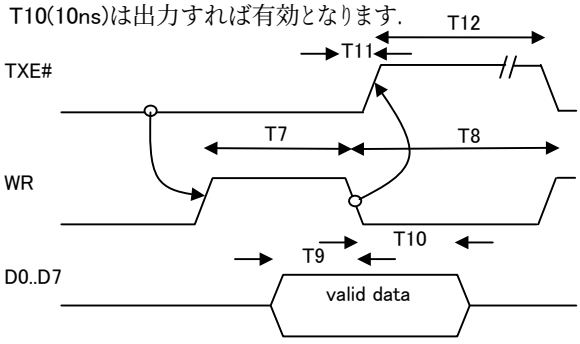
（単位：ns）

9．2 ライトサイクルのタイミングチャート

次に，ライトサイクルのタイミングですが，TXE#が Low のとき，FIFO(FT245BM)にデータを出すことができます．

WRの立下りで，TXE#が T12(80ns)以上は High になり，FIFO にデータが格納されます．FIFO のデータが満杯であれば，TXE#は High のままです．

データは WR の立下り前から T9(20ns)，立下り後は T10(10ns)は出力すれば有効となります．



Time	Description	Min	Max
T7	WR Active Pulse Width	50	
T8	WR to WR Pre-Change Time	50	
T9	Data Setup Time before WR inactive		20
T10	Data Hold Time from WR inactive	10	
T11	WR inactive to TXE#	5	25
T12	TXE inactive after RD cycle	80	

（単位：ns）

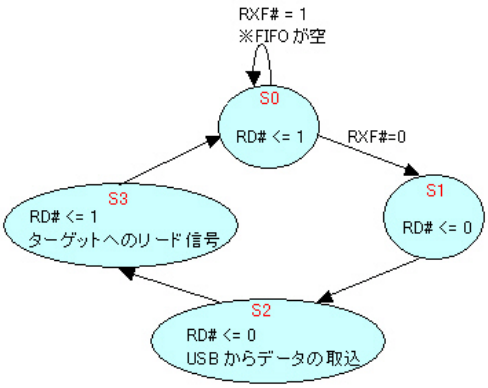
※詳しくは FTDI 社データシートをご覧下さい．

9．3 調停回路の状態遷移図

タイミングチャートより，リードサイクルとライトサイクルの状態遷移図を作成します．作成するのは FPGA 内部の USB の調停回路です．

USB 信号は双方向なので，リードとライトを同時に行うことはできません．また，USB FIFO やターゲットのモジュールに無制限にデータを送れないようにするために，Busy 信号などを考慮しなくてはなりません．

▼リードサイクル時の状態遷移図

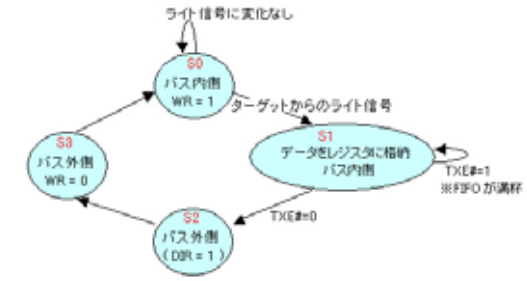


S0 の時に，条件が RXF#=1 つまり，PC 側から何も送信されていないことを意味します．USB FIFO は空です．ここで，RXF#=0 となれば，USB FIFO に文字があるという意味で，S1 に移り，その時の状態を RD#<=0 とします．ここでは USB からのデータを取込ません．タイミングチャートによれば，T3(30ns)となっているからです．

無条件に S2 に移動します．S2 で，RD#は下げたままで，USB からのデータを取込ます．

S3 で，RD#<=1 に戻します．ここで，ターゲットにリード信号を出します．これは，ターゲットのモジュールに対して，USB のデータの読み取りを知らせる信号です．

▼ライトサイクル時の状態遷移図



S0 の時に，ターゲットからのライト要求(TG_WR=1)があれば，S1 に移ります．S1 のとき，ターゲットからのデータをレジスタに格納します．

TXE#=1 であれば，FIFO がフルなので，USB FIFO にデータを送ることはできません．条件が TXE#=0 であれば，状態を S2 に移します．このとき，データバスは FPGA 内部からみて外側とします．データ線が双方向なので，方向には注意して下さい．

S3 のとき，WR=0 にします．データバスが外向きにして，S1 でレジスタに格納したデータを出力します．WR の立下りで USB FIFO はデータを取込みます．

9．4 USB 調停回路 VHDL モジュール

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

entity USB_ARBITER is
Port (CLK : in  std_logic;
      RST : in  std_logic;
      RXF_N : in std_logic;
      TXE_N : in std_logic;
      RD_N  : out std_logic;
      WR    : out std_logic;
      BUS_BUSY : out std_logic;
      TG_READY : in  std_logic;
      USB_DATA : inout std_logic_vector( 7 downto
0 );
      TG_DO    : out std_logic_vector( 7 downto 0 );
      TG_DI    : in  std_logic_vector( 7 downto 0 );
      TG_RD    : out std_logic;
      TG_WR    : in  std_logic );
end USB_ARBITER;

architecture RTL of USB_ARBITER is

signal RXF_REG      : std_logic;
signal TXE_REG      : std_logic;
signal USB_DATA_REG : std_logic_vector( 7 downto
0 );

signal DIR : std_logic;

signal RD_STATE : std_logic_vector( 1 downto 0 );
signal WR_STATE : std_logic_vector( 1 downto 0 );

constant S0 : std_logic_vector := "00";
constant S1 : std_logic_vector := "01";
constant S2 : std_logic_vector := "10";
constant S3 : std_logic_vector := "11";

begin

▼USB データの方向です．通常はハイインピーダンスにしておきます．
USB_DATA <= USB_DATA_REG when( DIR = '1') else
"ZZZZZZZZ";

▼このモジュールが処理中であれば，BUSY 信号を High にします．ステートマシンがライトとリード共に IDLE 状態であれば，BUSY は Low です．
BUS_BUSY <= '0' when( WR_STATE="00" and
RD_STATE="00" ) else '1';


▼念のため，D-FF を入れておきます．このとき，RXF#とTXE#を負論理から正論理に変えます．D-FF を入れなくても動作します．
process( CLK, RST ) begin
  if ( RST ='1') then
    RXF_REG <= '0';
    TXE_REG <= '0';
  elsif( CLK'event and CLK='1') then
    RXF_REG <= not RXF_N;
    TXE_REG <= not TXE_N;
  else
    NULL;
  end if;
end process;

▼リードサイクルのステートマシン
process( CLK, RST ) begin
▼リセットはシミュレーション時に初期値を代入するためでもあります．
  if ( RST ='1') then
    TG_DO <= "XXXXXXXX";
    TG_RD <= '0';
    RD_N <= '1';
    RD_STATE <= S0;
  elsif( CLK'event and CLK='1') then
    case RD_STATE is
      when S0 =>
▼TG_READY とはターゲットの読み取り準備がＯＫという意味です．ライトサイクルと重ならないように WR_STATE="00"としています．
        if( RXF_REG='1'  and
            TG_READY='1' and
            WR_STATE="00") then

▼この状態は S1 時の RD_N です．つまり，次の状態を記述しています．
            RD_N <= '0';
            RD_STATE <= S1;
        else
          RD_STATE <= S0;
        end if;
      when S1 =>
        TG_DO <= USB_DATA;
        RD_STATE <= S2;
      when S2 =>
        RD_N <= '1';
        TG_RD <= '1';
        RD_STATE <= S3;
      when S3 =>
        TG_RD <= '0';
        RD_STATE <= S0;
      when others => RD_STATE <= S0;
```

▼つづき

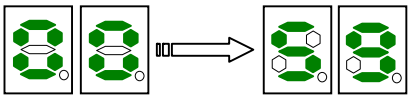
```
  end case;
  end if;
end process;

▼ライトサイクル時のステートマシン
process( CLK, RST ) begin
  if( RST ='1') then
    WR <= '1';
    WR_STATE <= S0;
    DIR <= '0';
  elsif( CLK'event and CLK='1') then
    case WR_STATE is
      when S0 =>
        ▼ターゲットからのライト要求
        if( TG_WR='1') then
          USB_DATA_REG <= TG_DI;
          WR_STATE <= S1;
        else
          WR_STATE <= S0;
        end if;

  ▼FIFO に書込み可能　かつ
    リードサイクルと重ならない
       if( TXE_REG='1' and RD_STATE="00") then
          WR <= '1';
          DIR <='1';
          WR_STATE <= S2;
        else
          WR_STATE <= S1;
        end if;
      when S2 =>
        WR <= '0';
        DIR <= '1';
        WR_STATE <= S3;
      when S3 =>
        WR <= '1';
        DIR <= '0';
        WR_STATE <= S0;
      when others => WR_STATE <= S0;
    end case;
  end if;
end process;
```

10 EDX-001 サンプル回路

これまでに説明した回路をもとに，応用回路を付属の CD に収録しています．ご活用ください．

10．1 サンプル回路1

7 セグメント LED のダイナミック点灯を利用し，一秒おきに“00”から“59”まで，カウントアップする回路です．



EDXSPL1_TOP	トップモジュール
CLK60	10 秒桁と 1 秒桁のカウンタ
CLKDIV	約1秒を生成
SEG_CTL	以下のモジュールを接続
SEG_CNT	セレクタ用カウンタ
SEG_DECODER	セグメンデコーダ
SEG_MUX	セグメントセレクタ

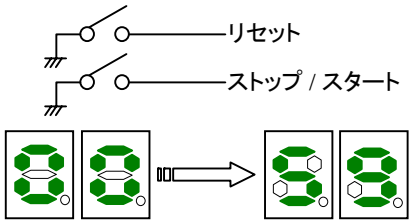
CLKDIV.vhd（一秒周期のカウンタ）について説明します．これは，一秒を生成するための 1/18432000 カウンタです．この SEC を CLK60 のクロックイネーブル信号として使用しています．

```
entity CLKDIV is
  ▼18432000 は25ビット
  generic ( WIDTH : integer := 25;
                N : integer := 18432000 );

  port    (   CLK : in  std_logic;
              RST : in  std_logic;
              SEC : out std_logic );
end CLKDIV;
```

10．2 サンプル回路2

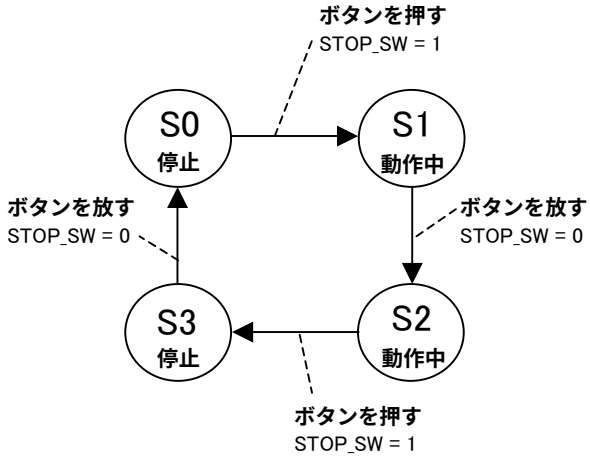
サンプル回路1にストップウォッチのように停止の状態を加え，リセットボタンで 0 に戻る回路です．

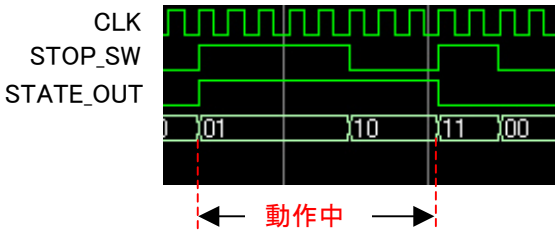
ここでは，押しボタンを使用するので，チャタリングがおこらないように 100Hz でサンプリングします．



EDXSPL2_TOP	トップモジュール
CLK60	10 秒桁と 1 秒桁のカウンタに停止と初期化の条件を加えたもの
CLKDIV	入力パラメータ変更により，約 1Hz と約 100Hz を生成
CHATTERING	チャタリングを取除く
STATE_MACHINE	″停止″と″開始″の2つの状態を定義
SEG_CTL	サンプル1同様

STATE_MACHINE.vhd（ステートマシン）について説明します．ここでは，ボタンを押す/放すでステートが切り替わるようにします．





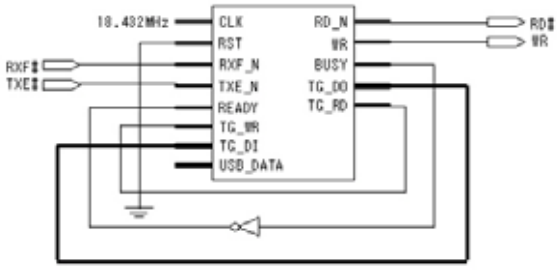
10．3 サンプル回路3

USB 調停回路を利用したエコー回路です．9 章で作成した調停回路を次のように結線すれば，送信した文字がそのまま PC にエコーされます．

文字の送受信を行うには，付属のサンプルソフトを使用するか，Windows 付属のハイパーターミナルなどの通信ソフトをご利用ください．

必要な回路は，USB の調停回路と，エコー用に結線したトップモジュールのみです．

▼結線図



上記の回路図を VHDL にて記述すると，次のようになります．

▼トップモジュール

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

▼つづき

```
entity EDXSPL3_TOP is
Port(CLK  : in std_logic; -- SYSTEM CLOCK
    PSW3  : in std_logic; -- SYSTEM RESET
    RXF_N : in  std_logic;
    TXE_N : in  std_logic;
    RD_N  : out std_logic;
    WR    : out std_logic;
    USB_DATA : inout std_logic_vector(7 downto 0));
end EDXSPL3_TOP;

architecture RTL of EDXSPL3_TOP is

component USB_ARBITER is
Port(CLK   : in  std_logic;
    RST   : in  std_logic;
    RXF_N : in std_logic;
    TXE_N : in std_logic;
    RD_N  : out std_logic;
    WR    : out std_logic;
    BUS_BUSY : out std_logic;
    TG_READY : in  std_logic;
    USB_DATA : inout  std_logic_vector(7 downto 0);
    TG_DO    : out std_logic_vector( 7 downto 0 );
    TG_DI    : in  std_logic_vector( 7 downto 0 );
    TG_RD : out std_logic;
    TG_WR : in  std_logic );
end component;

signal RST   : std_logic;
signal DATA  : std_logic_vector( 7 downto 0 );
signal ENB   : std_logic;
signal READY : std_logic;
signal BUSY  : std_logic;

begin

▼押しボタンは，押すとＬになるので否定
RST    <= not PSW3;
READY  <= not BUSY;

U0 : USB_ARBITER port map
( CLK => CLK,
  RST => RST,
  RXF_N => RXF_N,
  TXE_N => TXE_N,
  RD_N  => RD_N,
  WR    => WR,
  BUS_BUSY => BUSY,
  TG_READY => READY,
  USB_DATA => USB_DATA,
  TG_DO    => DATA,  -- USB からの得たデータを
  TG_DI    => DATA,  -- そのまま出力する
  TG_RD    => ENB,   -- ターゲットのリード ENB は
  TG_WR    => ENB ); -- そのままライト ENB となる
end RTL;
```

10．4　サンプル回路4

USB からデータをやり取りして，ドットマトリックスディスプレイに表示する回路です．8 章で説明した’＊’で頭だしする回路(pd_led.vhd)とサンプル回路3を組み合わせて，通信ソフトから任意のアスキーコードを表示させます．

サンプル回路3のトップモジュールに pd_led のコンポーネント宣言を加え，以下の文を加えます．ドットマトリックスディスプレイを使用する際は，リセットの論理に注意してください．DISP_RST は**正論理**です．

```
  U1 : PD_LED port map
  ( CLK => CLK,
    RST => RST,
    USB_DAT => DATA, --USB からのデータ
    USB_RE  => ENB,  --上のデータが有効という ENB 信号
    ▼この下の信号はドットマトリックスへの出力
    DISP_WR_N => DISP_WR_N,
    DISP_ADR  => DISP_ADR,
    DISP_DAT  => DISP_DAT  );
  ▼押しボタンスイッチを押すとドットマトリックス
    ディスプレイにリセットがかかる
DISP_RST <= not PSW3;
```

11 付属 CD-ROM の内容

付属 CD-ROM には，本マニュアルで説明した通信ソフト ED Term やサンプル回路などを収録しています．
各フォルダは，次のような内容となっています．

【BitFilses】
サンプル回路 1～4 のコンフィグレーションデータ

【DataSheet】
PD4435(ドットマトリックスディスプレイ),
FT245BM(FTDI 社 USB FIFO)のデータシート

【FtdiDrivers】
仮想 COM ポートドライバ(VcpDriver),

【ISEprj】
ISE のサンプルプロジェクト 1～4

【EDterm】
通信ソフト ED Term のセットアップファイルと VB ソース

【Document】
ユーザーズマニュアルと回路図

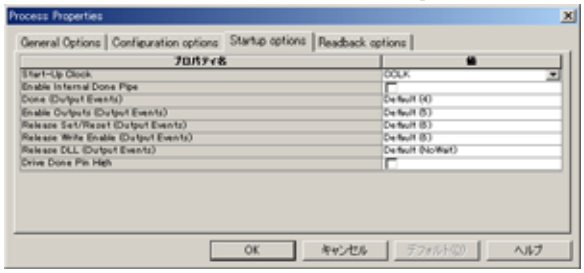
12 トラブル Q&A

通信やコンフィグレーション時の主なトラブルについて，原因とその解決策について説明します．

→ PC と EDX-001 の接続時，すべての制御ができなくなる
マシンの環境によって，デバイスが認識されるまで10秒以上かかる場合があります．

→ デバイスマネージャに表示されない
パソコンと EDX-001 が未接続の状態ではデバイスマネージャには表示されません．これは，Plug&Play 機能によるものです．

→ コンフィグレーションできない ①
Start-Up Clock を JTAG Clock とすると，コンフィグレーションできません．デフォルトのまま「**CCLK**」としてください．



→ コンフィグレーションできない ②
仮想 COM ポートドライバが正常にインストールされていない場合があります．4 章を参考にして，デバイスマネージャ上で認識されているか確認して下さい．

→ コンフィグレーションできない ③
初期化(INIT)スイッチを押さないと，CPLD は USB 信号を処理できません．USB にて送信されたコンフィグレーションデータは USB FIFO に格納され続け，満杯となります．事前に，INIT スイッチを押して下さい．

→ 通信ソフトがポートを認識しない
通信ソフトなどでポートのオープンした状態で，USB ケーブルを外すと，再びケーブルを接続しても，不安定になり，ポートを認識しない場合があります．USB ケーブルを外す際には，必ず**ポートをクローズ**してください．

→ 電源供給が不十分
ノートPCや環境により，電源供給が不十分となる場合があります．付属の AC アダプタを利用して下さい．

13 EDX-001 参考資料について

追加資料や参考資料がつくられた場合は，

【製品サポートページ】
http://www.hdl.co.jp/support_c.html

にデータをアップロードすることにいたします。ファイルの拡張子が “.exe” のときは、自己解凍ファイルです。

パスワードを求められたときは”thanks”を入力していただければ開けます。

スパルタン 2 FPGA トレーナ
EDX-001

ユーザーズマニュアル

2002/08/21 初版（R1）

2003/12/08 第2版（R2）

2007/01/15 第3版（R2）

有限会社ヒューマンデータ

〒567－0034

大阪府茨木市中穂積1－2－10

ジブラルタ生命茨木ビル

TEL 072-620-2002

FAX 072-620-2003

ＵＲＬ　http://www.hdl.co.jp

Ｍａｉｌ　spc2@hdl.co.jp