

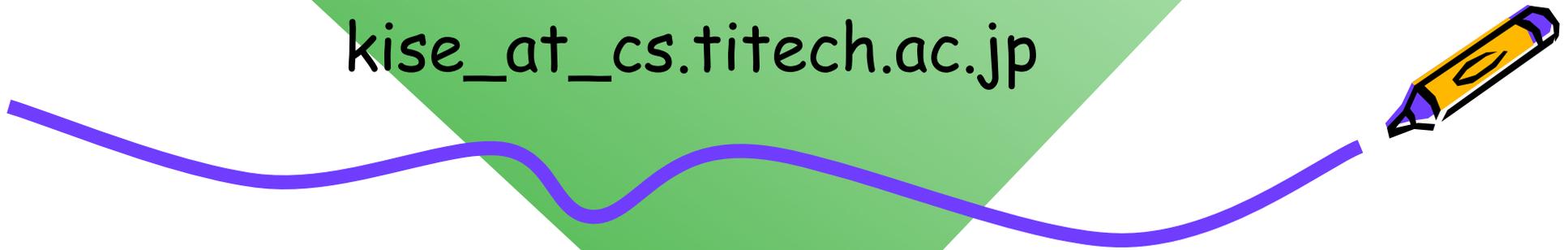
2013年度(平成25年度)版

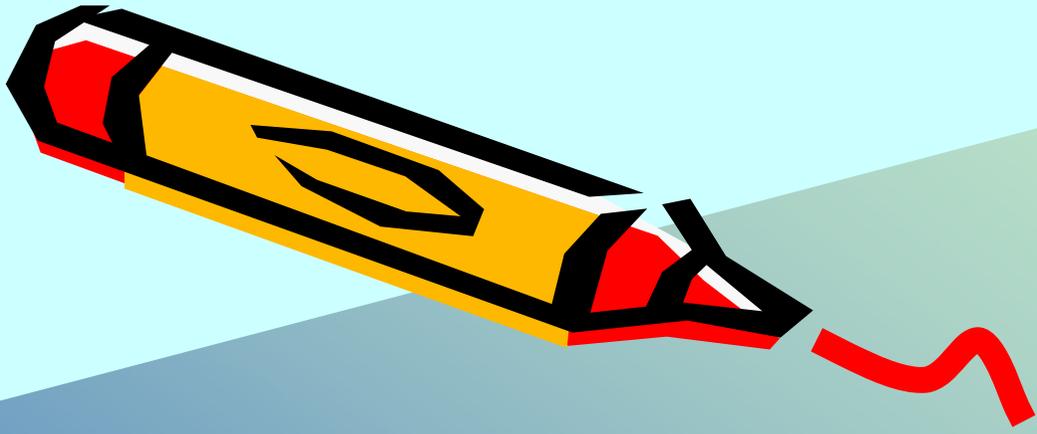
Ver. 2013-10-07



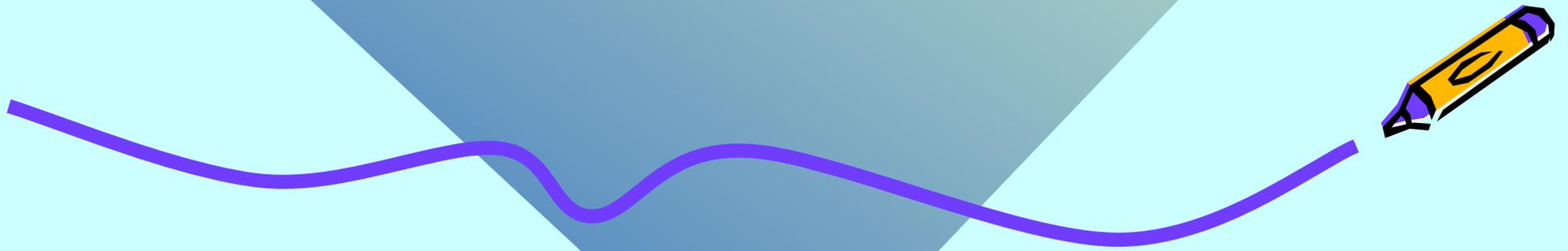
情報工学科 情報実験第四 組み込みシステム

情報工学科 吉瀬謙二
kise_at_cs.titech.ac.jp





実験 1日目



実験の目的, 注意, 参考書

• 目的

- ハードウェアおよびソフトウェアからのアプローチを通じて、組み込みシステムに関する知識と技術を習得する。

• 注意

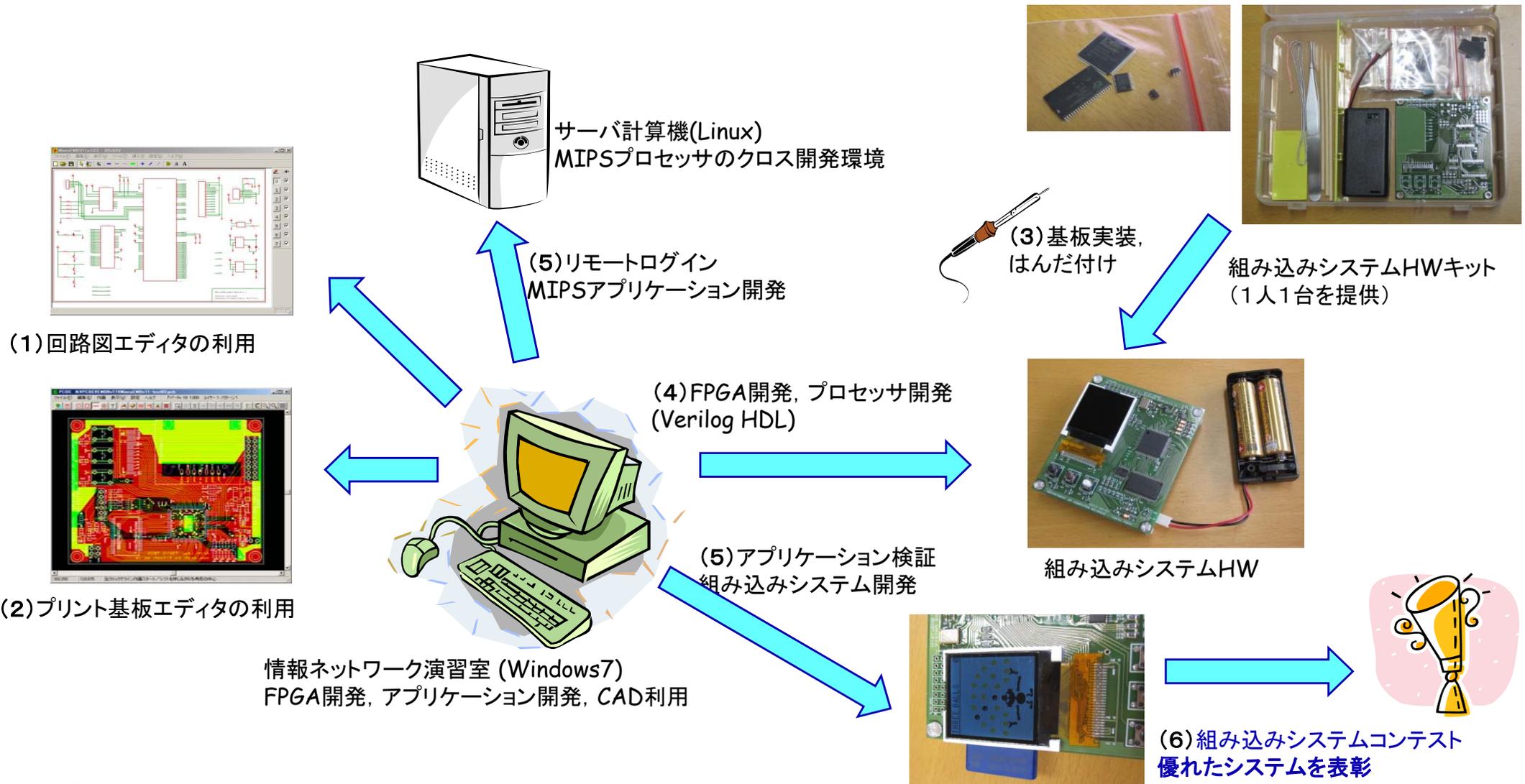
- 計算機アーキテクチャ第一 (6学期, ○, 2-0-0)
オペレーティングシステム (6学期, ○, 2-0-0)
を履修しておくことが望ましい。

• 参考書

- コンピュータの構成と設計 第3版, パターソン&ヘネシー (成田光彰 訳), 日経BP社, 2006
- オペレーティングシステム設計と実装 第3版, A.S.タネンバウム, A.S.ウッドハル, ピアソン・エデュケーション, 2007



情報実験第四「組み込みシステム」実験の概要



実験スケジュール, 実施場所

1. 実験の説明, セットアップ等 【A】
2. 組み込みシステムHWの制作と動作確認 【B】
3. 組み込みシステムHWの制作と動作確認 【B】
4. ハードウェア記述言語によるFPGA開発 【A】
5. FPGAへのプロセッサの実装 【A】
6. アセンブラによる組み込みアプリケーション開発 【A】
7. アセンブラによる組み込みアプリケーション開発 【A】
8. C言語による組み込みアプリケーション開発 【A】
9. C言語による組み込みアプリケーション開発 【A】
10. 組み込みシステム開発 【A】
11. 組み込みシステム開発 【A】
12. 組み込みシステムコンテスト 【A】

【A】は, 情報ネットワーク演習室 第1演習室(大岡山 南4号館 3階)で実施.

【B】は, VLSI設計室 <http://www.vdc.ss.titech.ac.jp/> で実施.

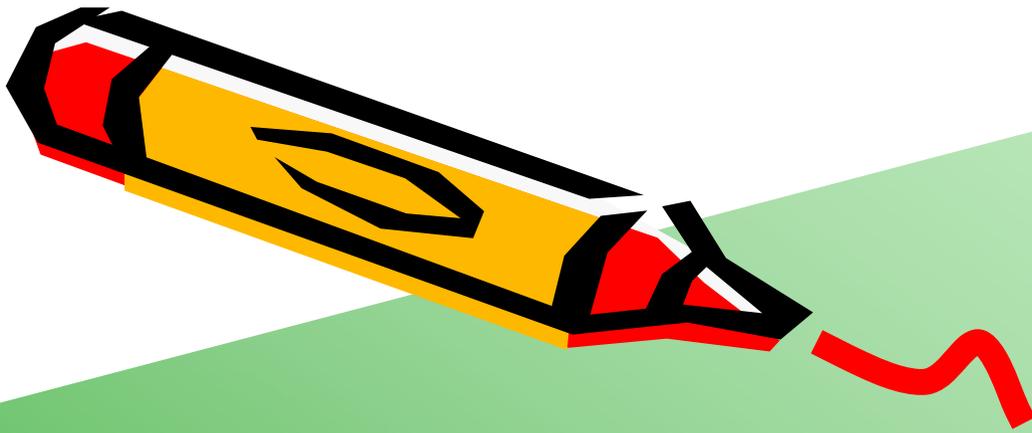
9:40 に集合(実験時間 9:45~12:15)してください. 開始時に出席をとります.



情報実験第四「組み込みシステム」補足

- 実験設備の制約から、最大受け入れ数を20人とします。
- チーム制ではなく、1人で1台のハードウェアを実装して利用します。
- 質問などは以下のアドレスにメールを送信してください。
 - `emb_at_arch.cs.titech.ac.jp`
- 「組み込みシステム」のホームページに最新情報を掲載します。
 - `www.arch.cs.titech.ac.jp/lecture/emb/index.html`
- 本スライドで (CPX)   は、チェックポイント, Check Point を意味します。ここにたどり着いたら、TAに知らせてください。





セットアップ(1)

教育用電子計算機システムのアカウント設定



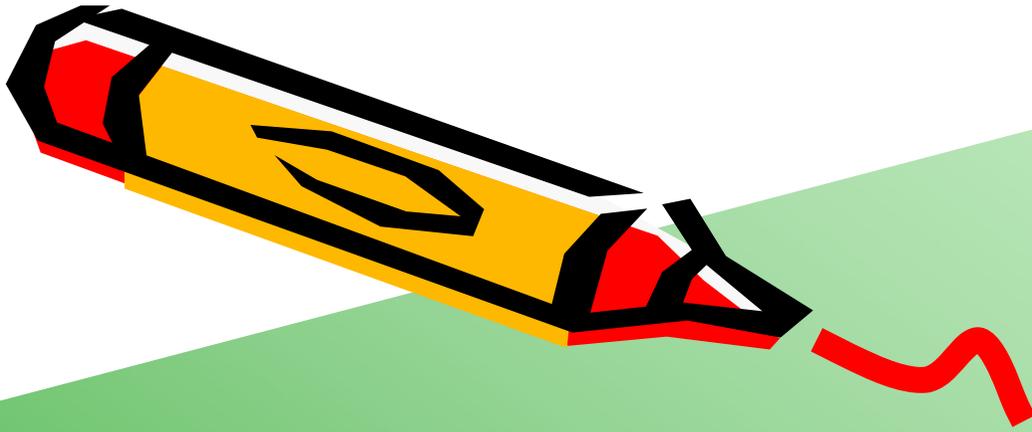
教育用電子計算機システムのアカウント設定

- TSUBAMEのアカウント, 教育用電子計算機システムのアカウントは共通です.
- **TSUBAMEのアカウントが有る場合**には, それを使ってログイン. アカウント設定の作業は必要ありません.
- **TSUBAMEのアカウントが無い場合**には, その取得が必要です.
- こちらを参照して, アカウントを設定してください.
 - <http://edu.gsic.titech.ac.jp/?q=account>
 - 0. 教育システムの端末に申請用アカウントでログインする.
 - 3. TSUBAMEアカウントを取得する.
 - 4. ログオフして教育システム端末に再度ログインする.



わからない時はTA/教員に質問





セットアップ(2) 実験で利用するファイルのコピー



実験で利用するファイルのコピー

- TAからUSBメモリを借りる.
- USBメモリを端末に挿入し、「Emb ディレクトリ」のすべてのファイルをZドライブにコピーする.
- USBメモリを返却する.



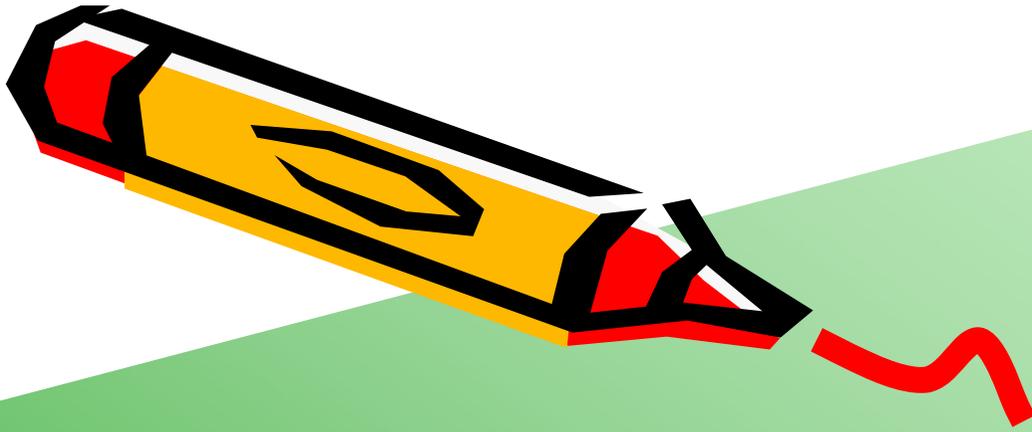
実験のディレクトリ構成

- Emb

- Doc
 - ... 実験のためドキュメントを格納
- Exe
 - ... 実行プログラムなど
- ISE
 - ... FPGA用のプロジェクトファイルなど
- Circuit
 - ... 回路図など
- SDK
 - ... アセンブリ言語/C言語のアプリケーション開発用
- Bitfile
 - ... FPGA用の構成ファイルなど

このファイルは Z:¥Emb¥Doc¥Emb-Jikken.ppt





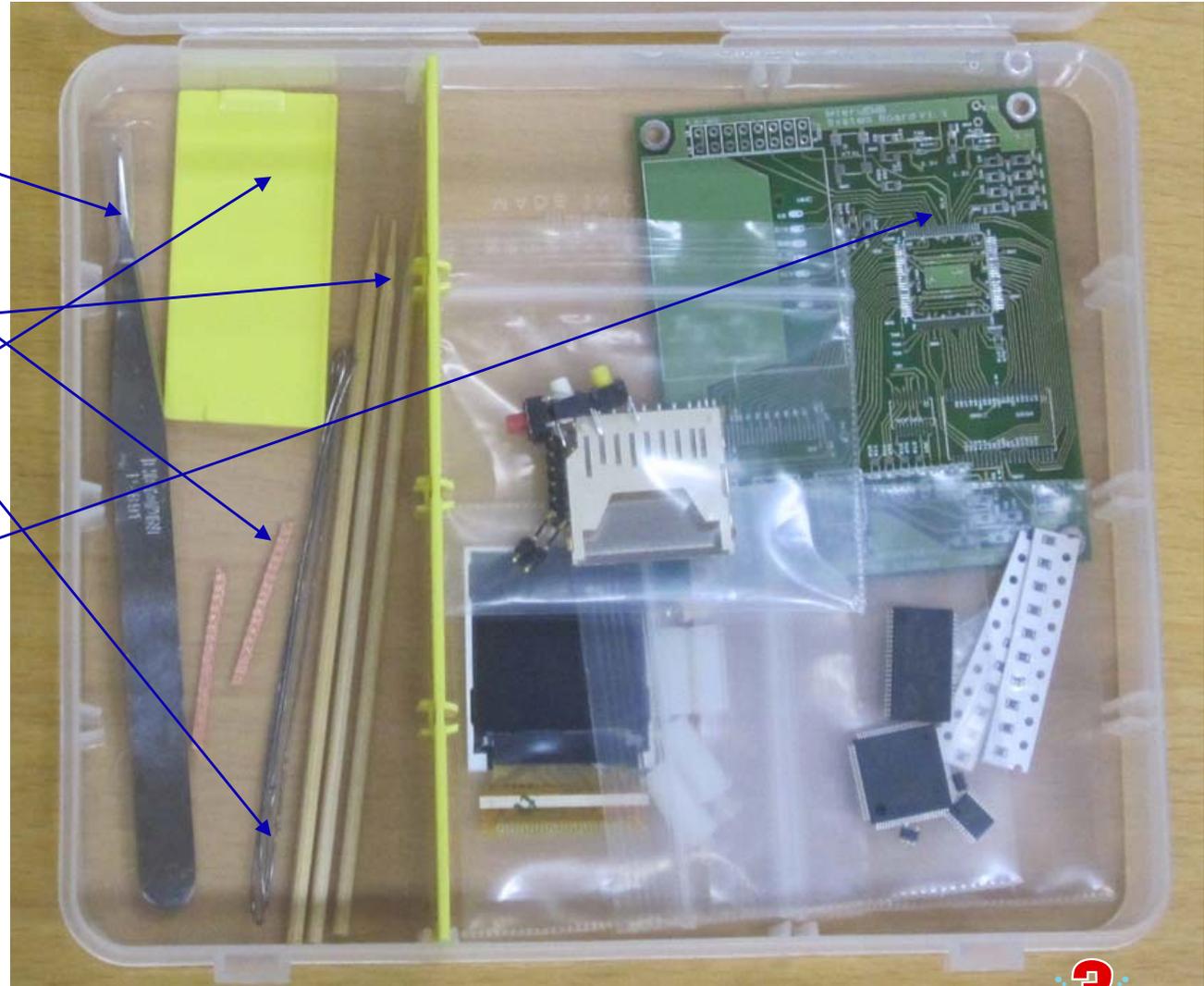
セットアップ(3) 組み込みシステムHWキットの確認



組み込みシステムHWキットの確認(1/2)

- 以下のすべての部品が揃っていることを確認する。
- **部品は丁寧に扱うこと。**

- ピンセット
- はんだ
- はんだ吸取線
- 竹串(3本)
- プラスチック片
- プリント基板

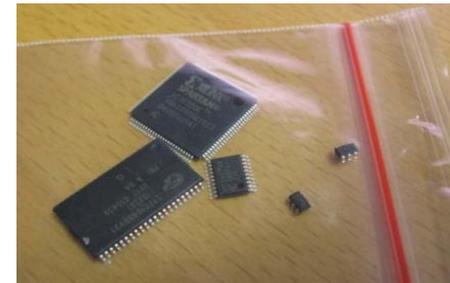


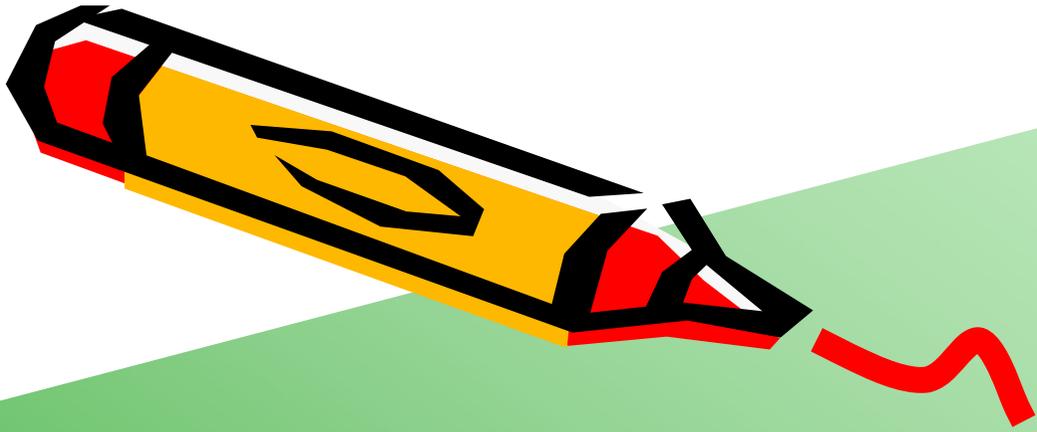
不足部品がある場合にはTA/教員に尋ねる。



組み込みシステムキットの確認(2/2)

- 以下のすべての部品が揃っていることを確認する。
 - FPGA
 - SRAM
 - PROM
 - レギュレータ 2個(PQFJ, T42の刻印)
 - 液晶モジュール ZY-FGD1442701V1
コントローラIC: ST7735
 - スペーサとネジ 4組
 - チップ抵抗 102 10個, 472 10個, 511 5個
 - チップコンデンサ 105 3個, 103 2個
(チップコンデンサには刻印が無いので注意)
 - 発光ダイオード 5個(または6個)
 - 6ピンヘッダ, 2ピンヘッダ
 - スイッチ 3個
 - 40MHz クロックオシレータ
 - SDカードスロット



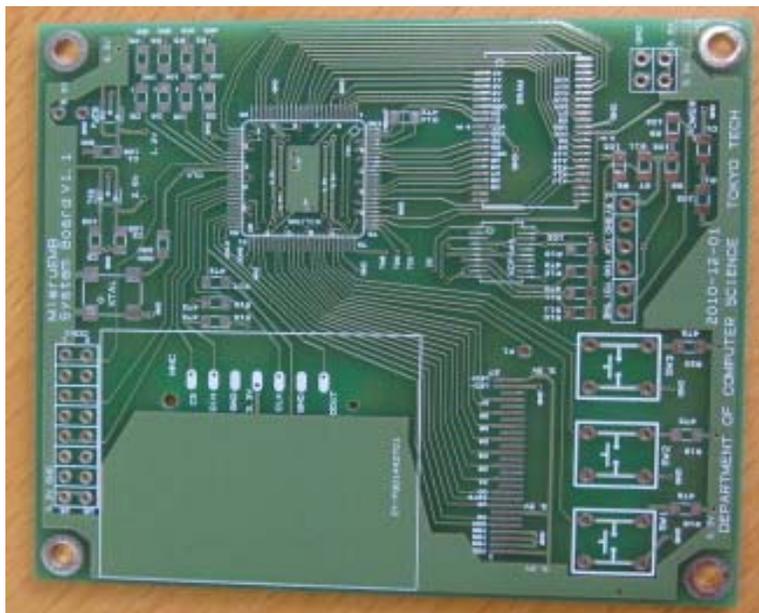


回路図エディタの使い方

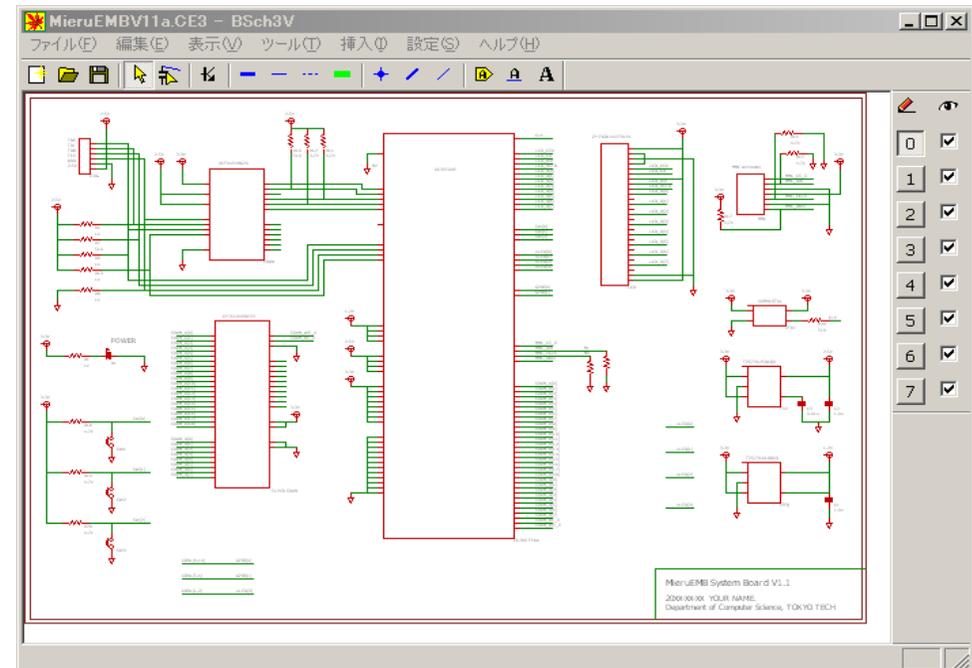


回路図エディタ BSch3V

- 目的: 回路図エディタの使い方を学ぶ. 実装するシステムの回路図表現を把握する.
- ソフトウェアの起動
 - Z:¥Emb¥Exe¥bs3vp¥bsch3v.exe
 - 概要や使い方は次を参照
 - Z:¥Emb¥Exe¥bs3vp¥README.htm
- ファイル(MieruEMB System Board V1.1の回路図)を読み込む.
 - Z:¥Emb¥Circuit¥MieruEMBV11a.CE3



MieruEMB System Board V1.1

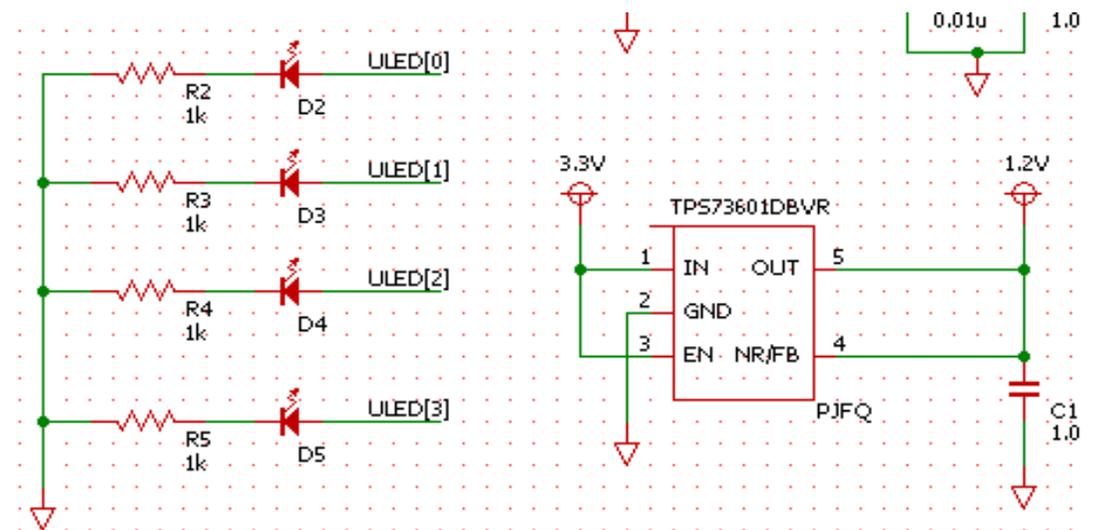
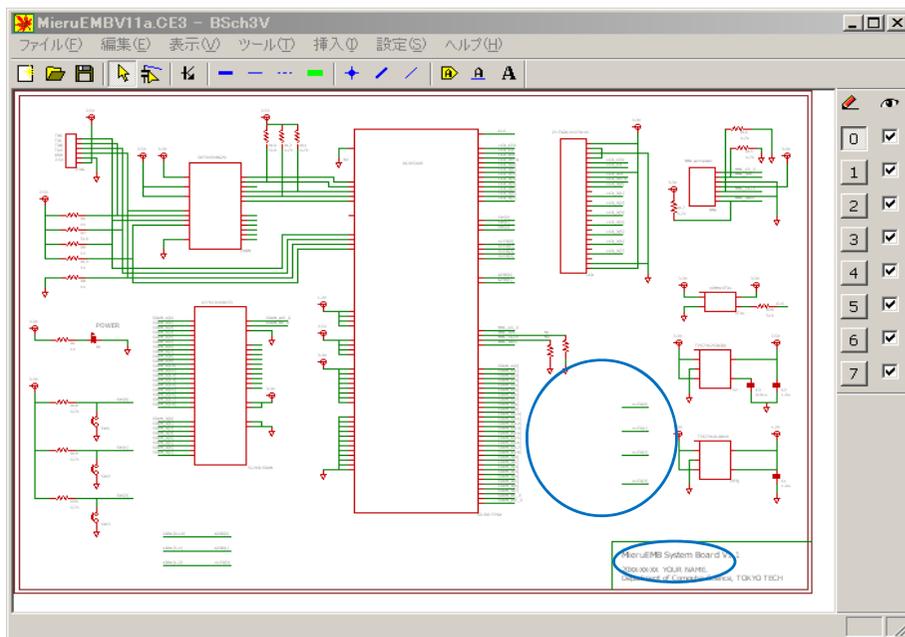


MieruEMB System Board V1.1の回路図



システムボードの回路図修正

- 回路図を修正(ダイオードと抵抗を追加する)し, Emb/Circuit/MieruEMBV11b.CE として保存
- 右下に, 名前, 今日の日付を記入
- 回路図を印刷 (CP1)
 - 本スライド最後の「Check Point確認シート」も印刷



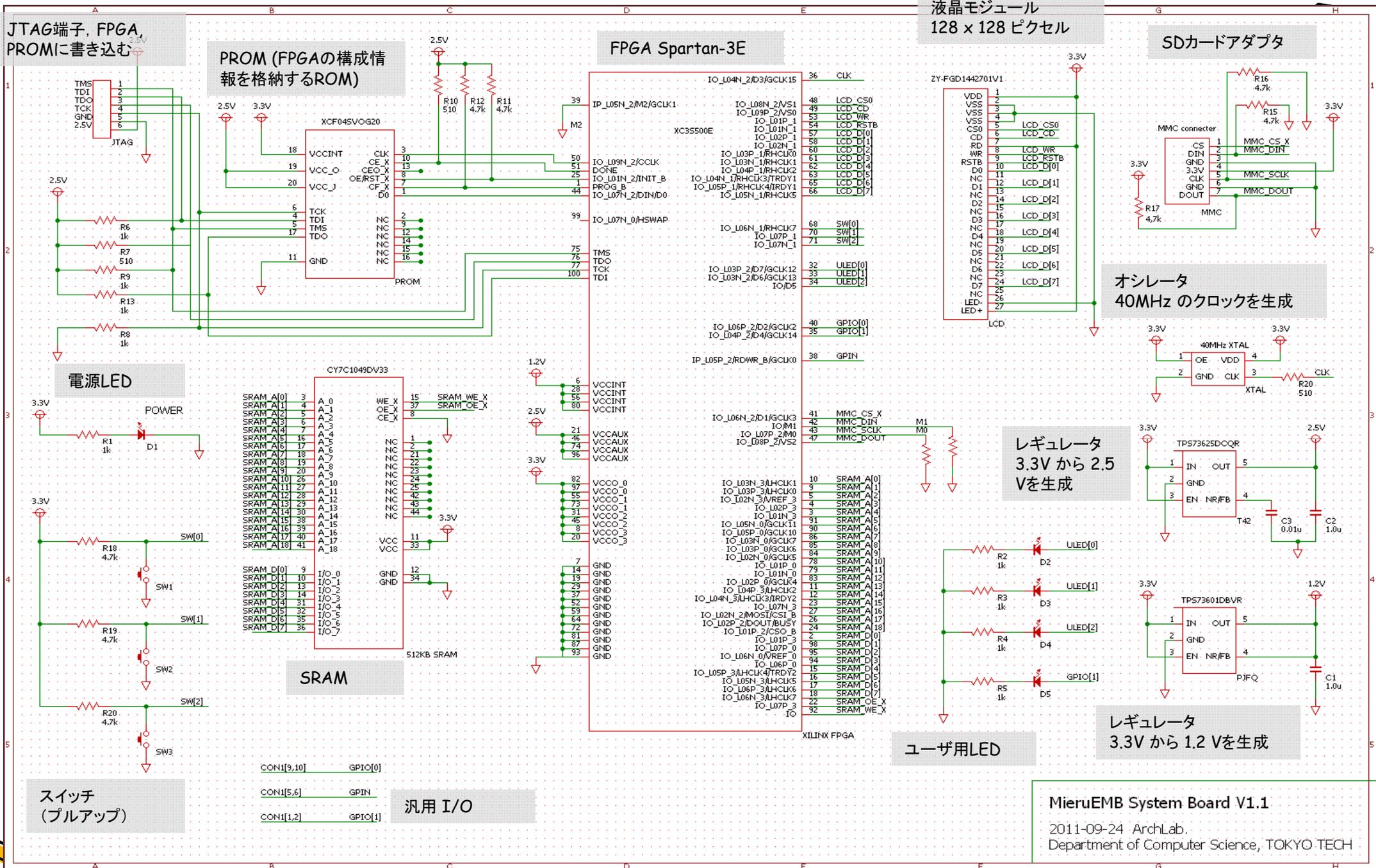
MieruEMB System Board V1.1

2011-09-24 ArchLab.

Department of Computer Science, TOKYO TECH



システムボードの回路図



JTAG端子, FPGA, PROMに書き込む

PROM (FPGAの構成情報を格納するROM)

FPGA Spartan-3E

液晶モジュール
128 x 128 ピクセル

SDカードアダプタ

電源LED

CY7C1049DV33

オシレータ
40MHz のクロックを生成

レギュレータ
3.3V から 2.5V を生成

レギュレータ
3.3V から 1.2V を生成

ユーザ用LED

スイッチ
(プルアップ)

汎用 I/O
CON1[3,10] GPIO[0]
CON1[5,6] GPIN
CON1[1,2] GPIO[1]

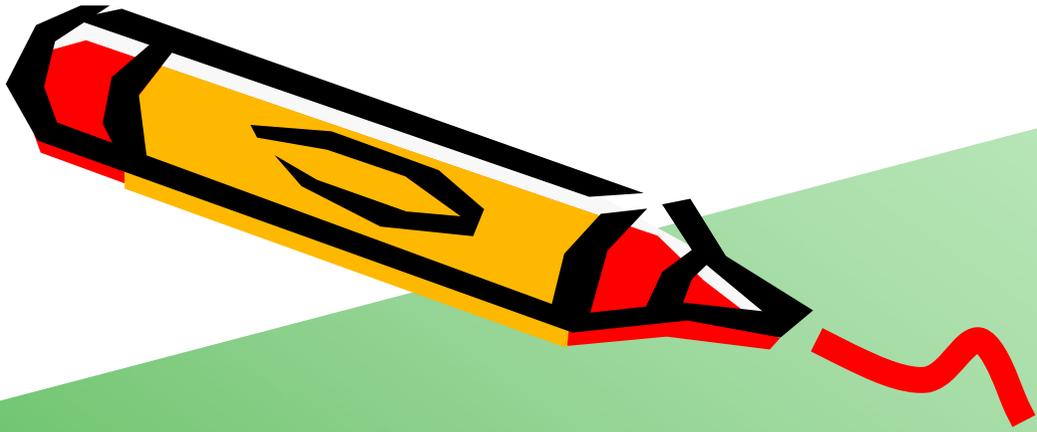
MieruEMB System Board V1.1

2011-09-24 ArchLab.
Department of Computer Science, TOKYO TECH

情報ネットワーク演習室における印刷

- 印刷した資料はファイリングして実験時に持参すること.
- プリンタの利用については次を参照
 - <http://edu.gsic.titech.ac.jp/?q=printer>
- 年間のプリント可能枚数に制限がある(今年は200枚)ので注意.
- 実験 2日目～3日目「組み込みシステムHWの実装」の全てのスライドと「ハードウェアデバッグの方法(Z:\¥Emb¥Doc¥ Emb-HWDebug.pdf)」の全てのスライドを印刷して、VLSI設計室に持参すること.
見やすいように、1スライドを1ページとして印刷することを推奨.



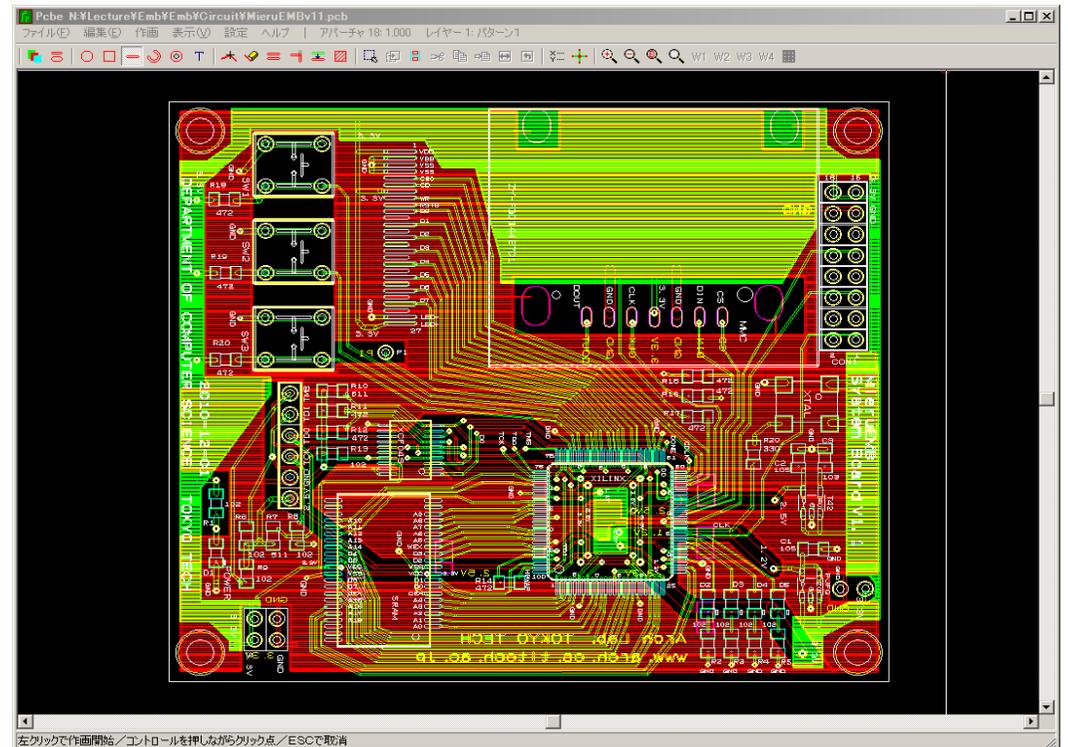
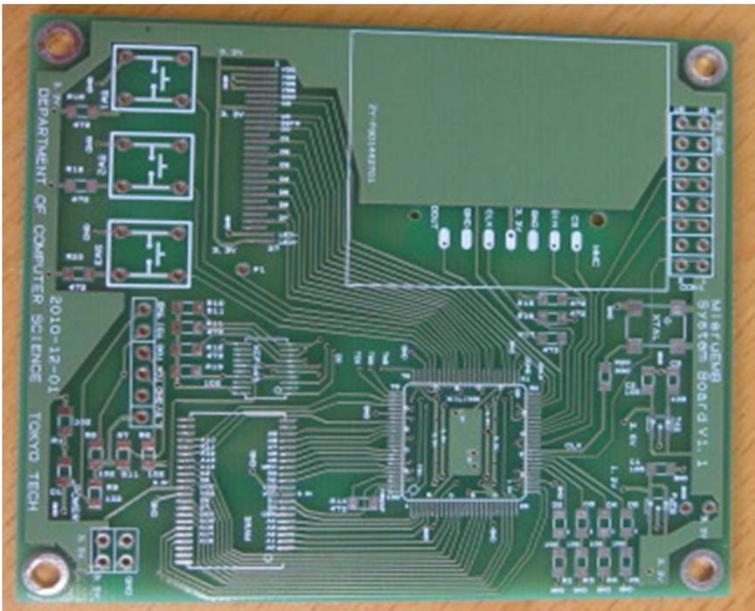


プリント基板エディタの使い方



プリント基板エディタ PCBE

- 目的: プリント基板エディタの使い方を学ぶ. 利用するプリント基板の構成を把握する.
- ソフトウェアの起動
 - Z:\¥Emb¥Exe¥pcbe¥pcbe.exe
- ファイル(MieruEMB System Board V1.1のデータ)を読み込む.
 - Z:\¥Emb¥Circuit¥MieruEMBV11a.pcb



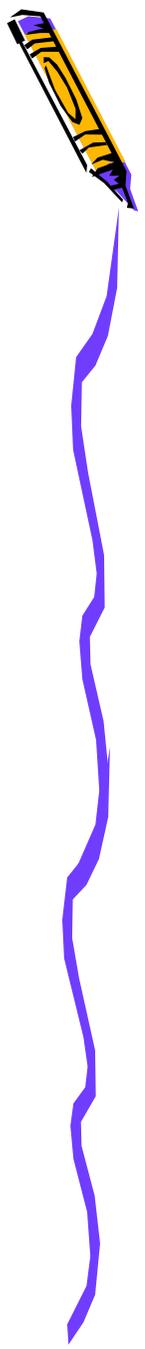
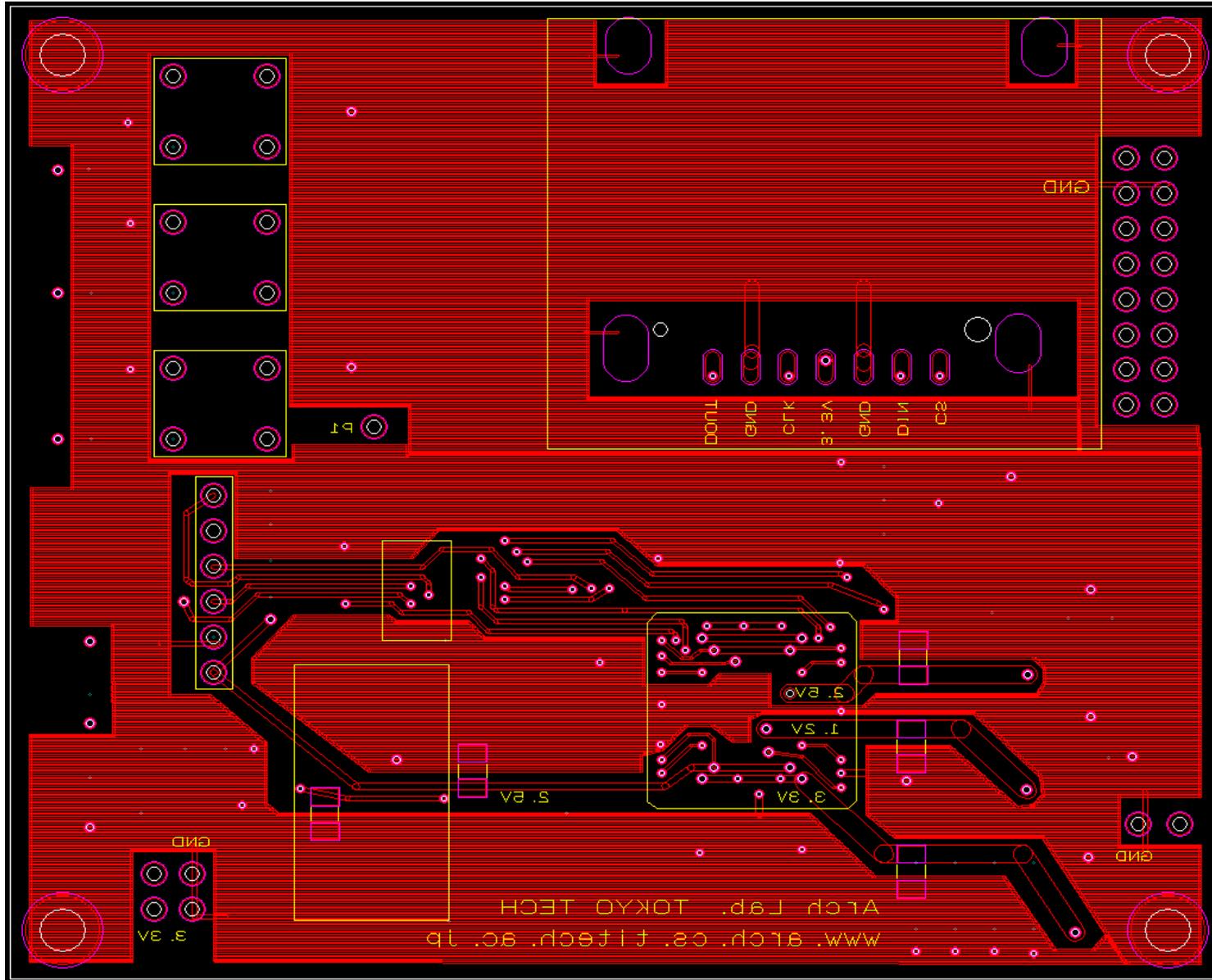
プリント基板エディタ PCBE

- MieruEMB System Board V1.1
 - 半田面(表), 部品面(裏)の2層で設計
- MieruEMB System Board V1.1 のレイヤー設定
 - レイヤー 1 : 半田面パターン
 - レイヤー 2 : 部品面パターン
 - レイヤー 3 : 半田面シルク
 - レイヤー 4 : 部品面シルク
 - レイヤー 5 : 半田面レジスト
 - レイヤー 6 : 部品面レジスト
 - レイヤー 7 : 外形
 - レイヤー 8 : 孔



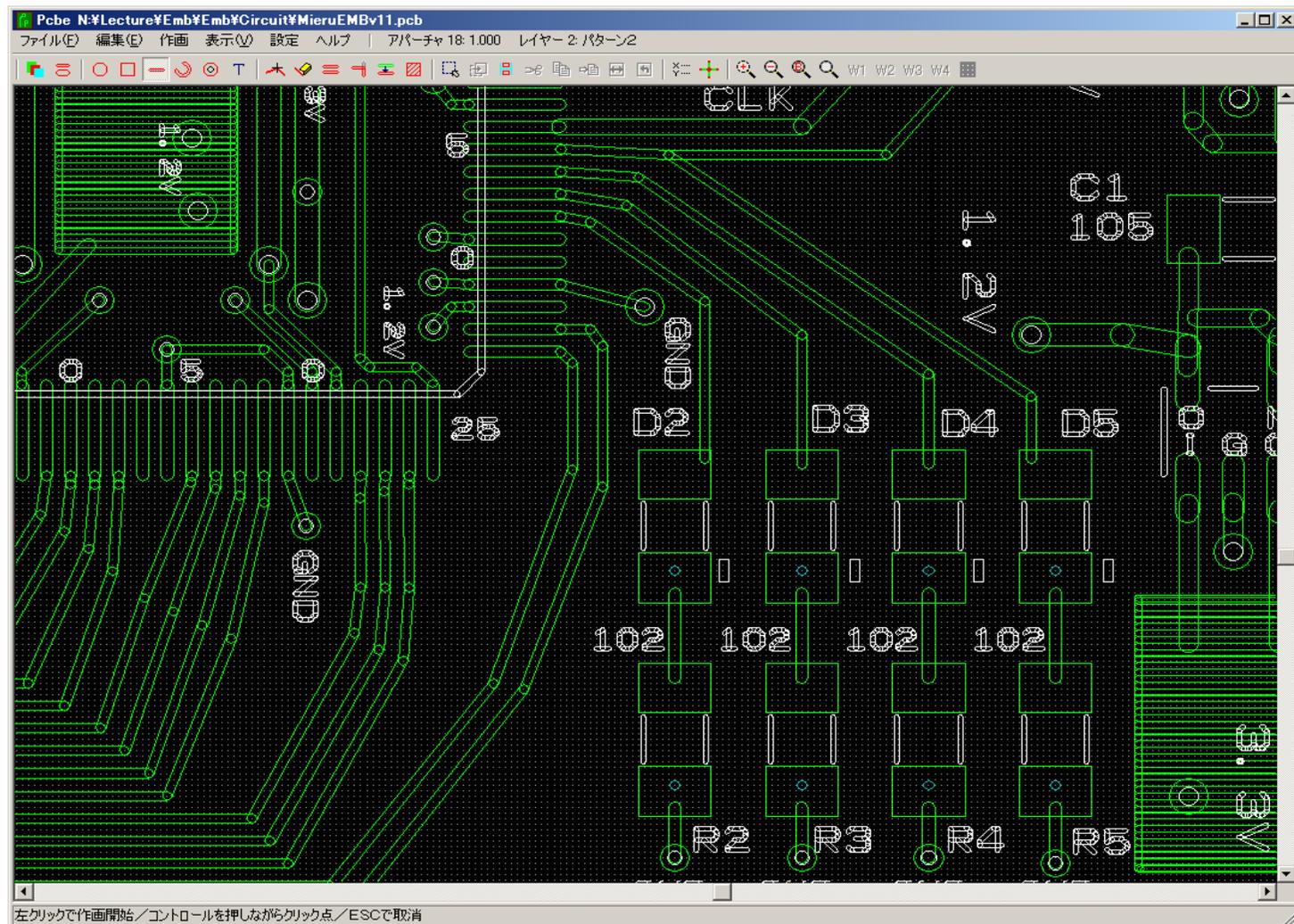
半田面(裏)

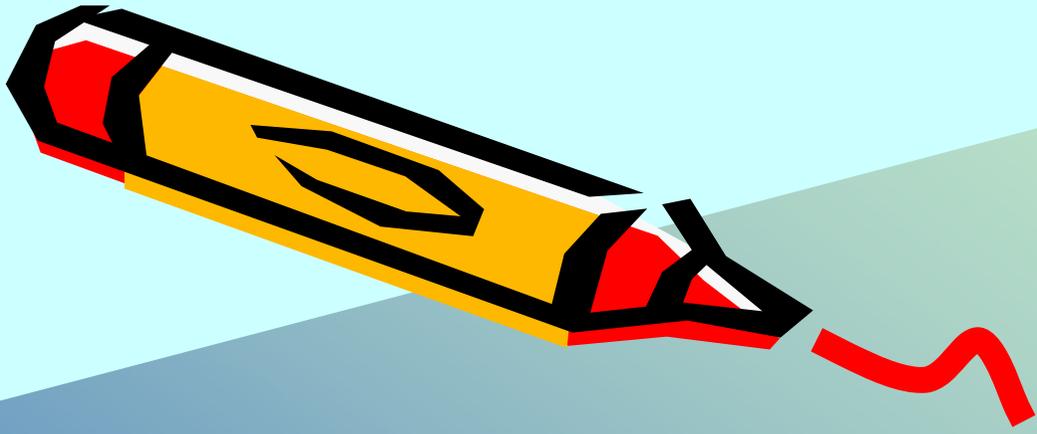
- レイヤー 1, 3, 5, 7 (半田面パターン, 半田面シルク, 半田面レジスト, 外形)



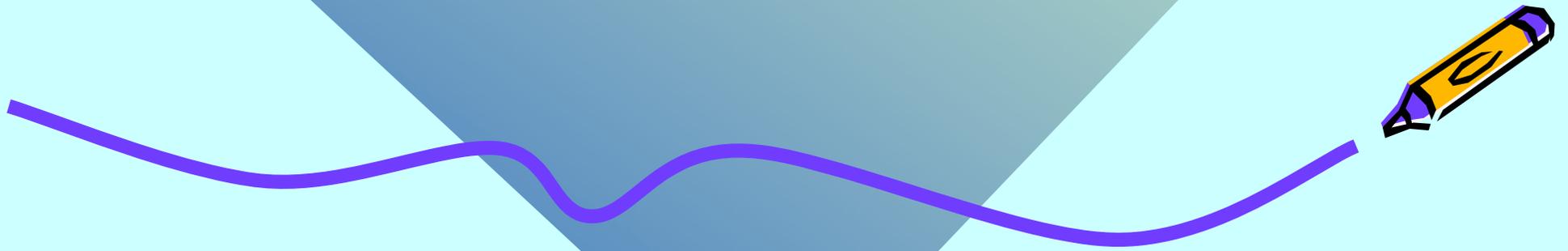
プリント基板エディタ PCBE

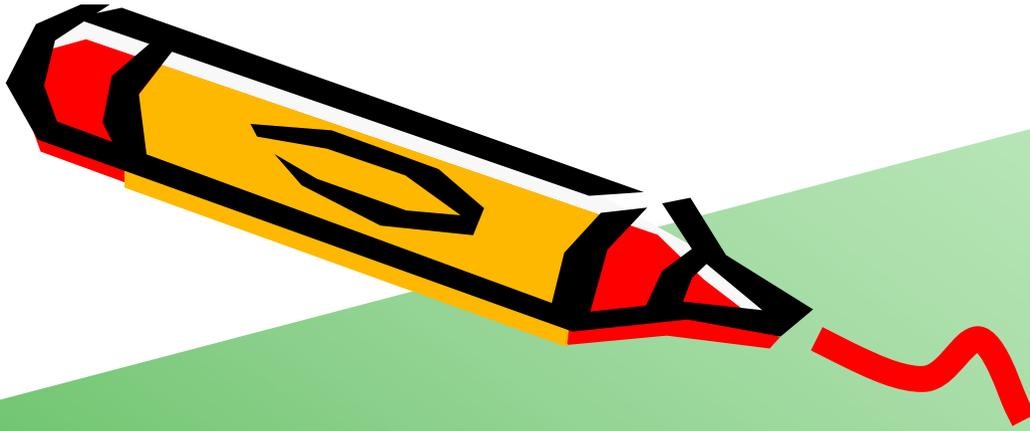
- LED周辺(D2, D3, D4, D5) に配線を追加。
 - Z:\¥Emb¥Circuit¥MieruEMBV11b.pcb として保存
- レイヤー 2, 4, 6, 7 を選択して印刷。 (CP2)





実験 2日目～3日目



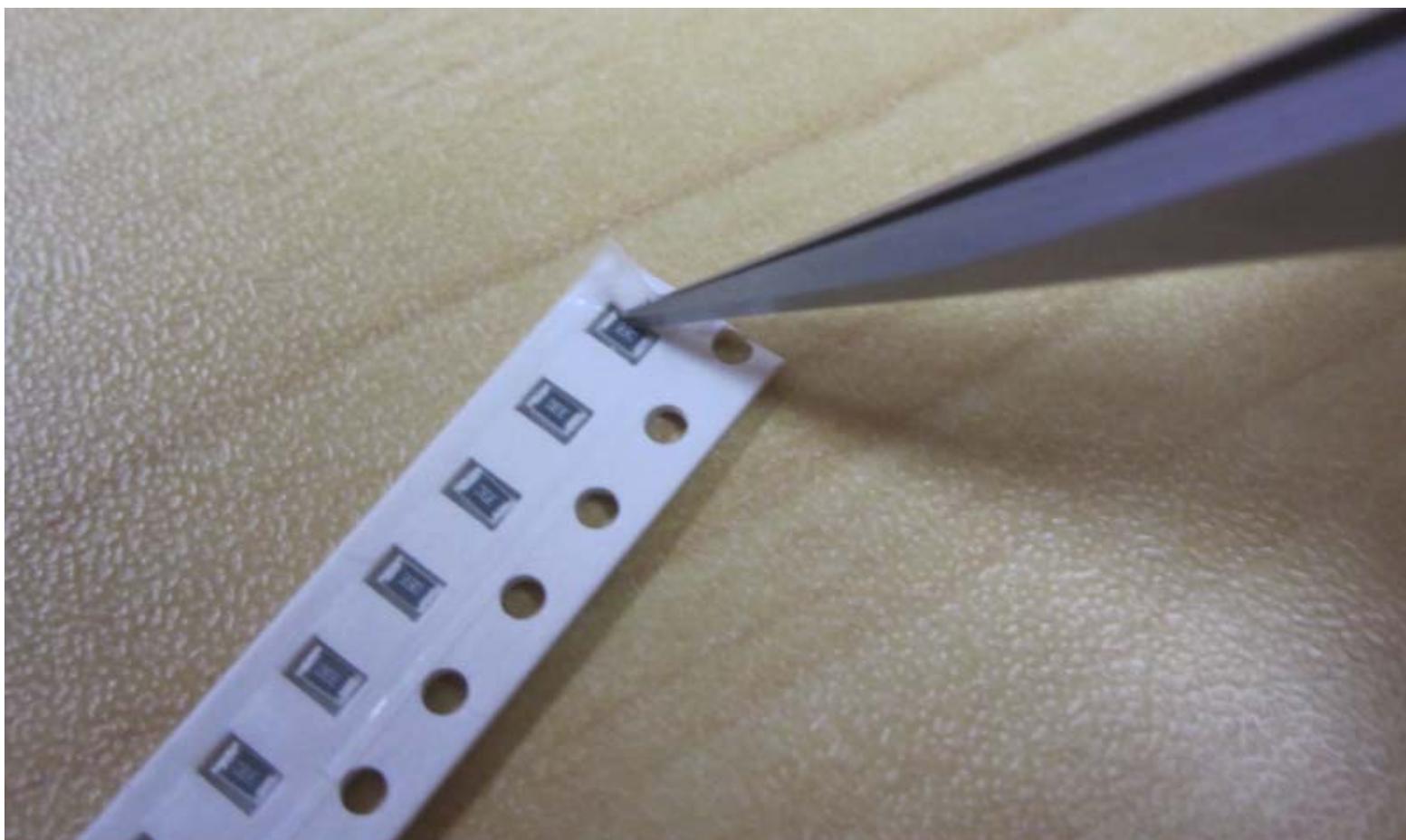


組み込みシステムHWの実装



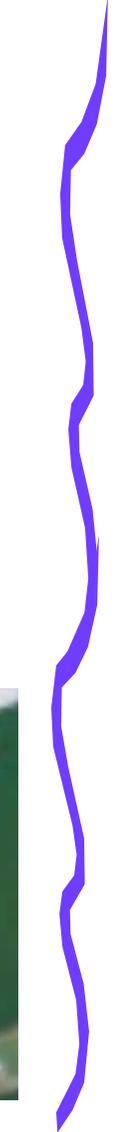
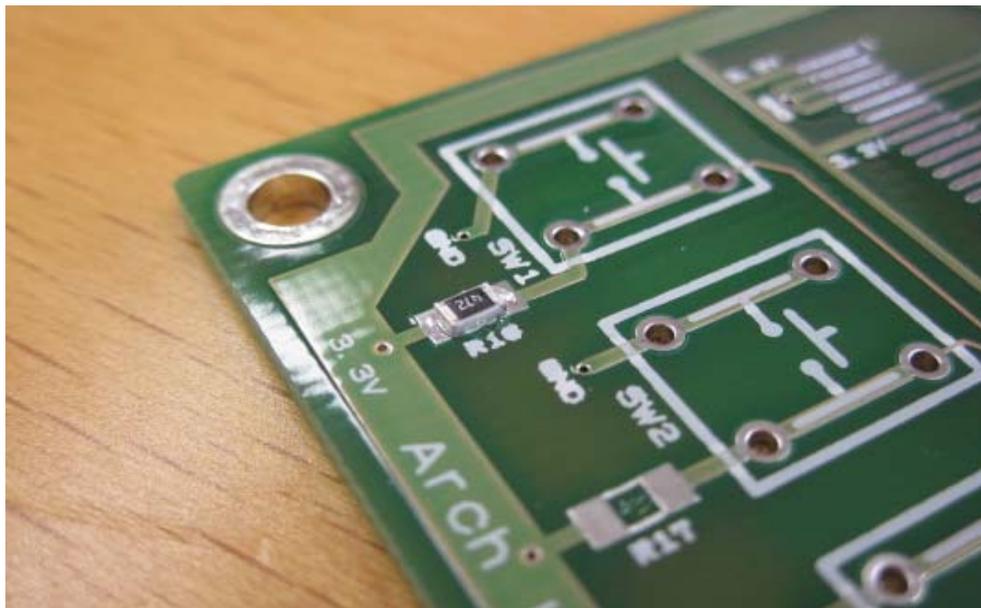
1. チップ抵抗, コンデンサ, ダイオードなどの取り出し方

- ピンセットを使って, ビニールをゆっくり剥がす.
- 小さい部品なので飛び散らないように注意すること.



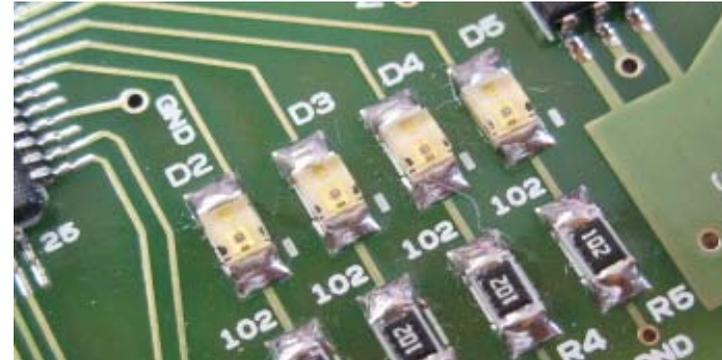
1. チップ抵抗, コンデンサ, ダイオードの固定方法

- プラスチック片に接着剤をのせる.
- 竹串の先に少量の接着剤をつける.
- **基板の部品固定部分に接着剤をぬる.**
- ピンセットを使って部品を固定する.
部品を固定場所に置いて, **接着剤の付いていない別の竹串で抑える**とうまくいく.

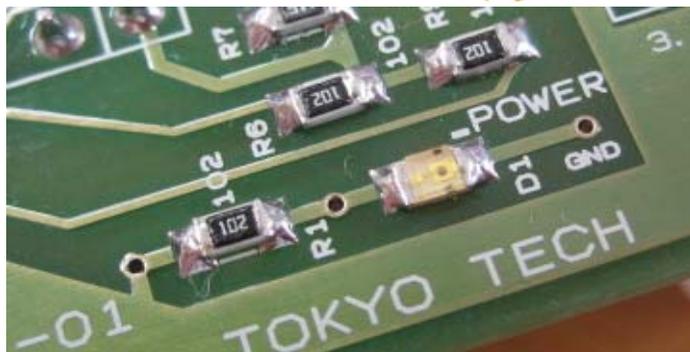


1. チップ抵抗, コンデンサ, ダイオードの固定(30~50分)

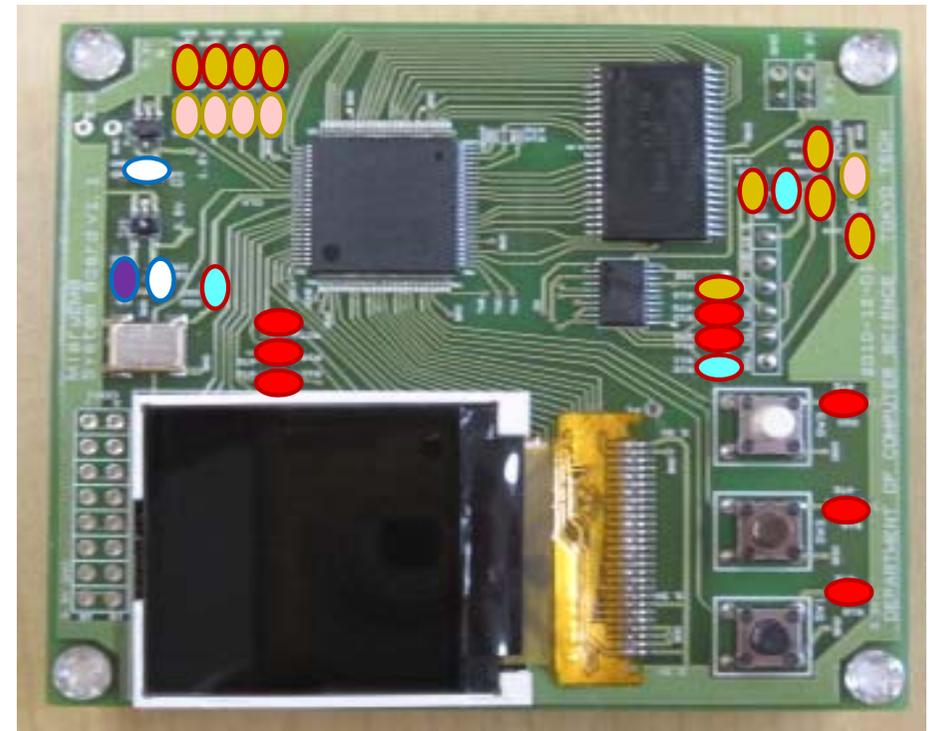
- (a) チップ抵抗を固定
 - 472 (4.7K Ω) 8個 
 - 102 (1K Ω) 9個 
 - 511 (510 Ω) 3個 
 - R20 330 にも 511(510 Ω)を使う.
- (c) チップコンデンサの固定
 - C1 と C2 は 105 (3個入りのパッケージ) 2個 
 - C3 は 103 (2個入りのパッケージ) 1個 
 - を使う. 要注意.
- (b) ダイオードの固定 5個 
 - **ダイオードには極性がある(正しい方向で固定).**
 - 黒色のマークがある方をGND側にする.
- 全てのチップ抵抗, チップコンデンサ, ダイオードを固定(CP3) 



写真のはんだ付け後のもの, ダイオードの向きに注意



写真のはんだ付け後のもの, ダイオードの向きに注意



2. チップ抵抗, コンデンサ, ダイオードのはんだ付け方法

- 接着剤が乾くまで5分~10分ほど待つ.
- こて台にスポンジを入れ少量の水を注入.
- はんだ付けする部分に**フラックス**をぬる.
- 以下の手順ではんだ付けを行う:
 - Step1. はんだ付けする部分をこてで加熱
 - Step2. 加熱部にはんだ線を付ける
 - Step3. 加熱部からはんだ線を離す
 - Step4. 加熱部からこてを離す
- はんだごての先は熱いのでやけどに注意すること.
- 十分に換気すること.



はんだごて(青), こて台(黒+スポンジ), フラックス(右下)



奥のチップ抵抗(R18)のはんだ付けをしたところ.
手前(R17)は固定したところ.



2. チップ抵抗, コンデンサ, ダイオードのはんだ付け(20~40分)

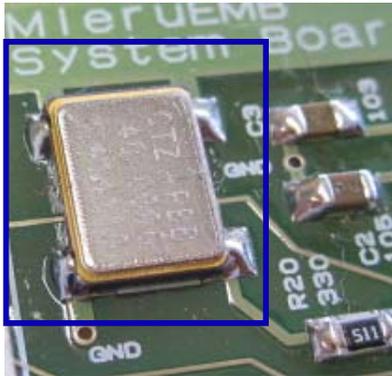
- 固定したチップ抵抗, コンデンサ, ダイオードのはんだ付け (CP4)



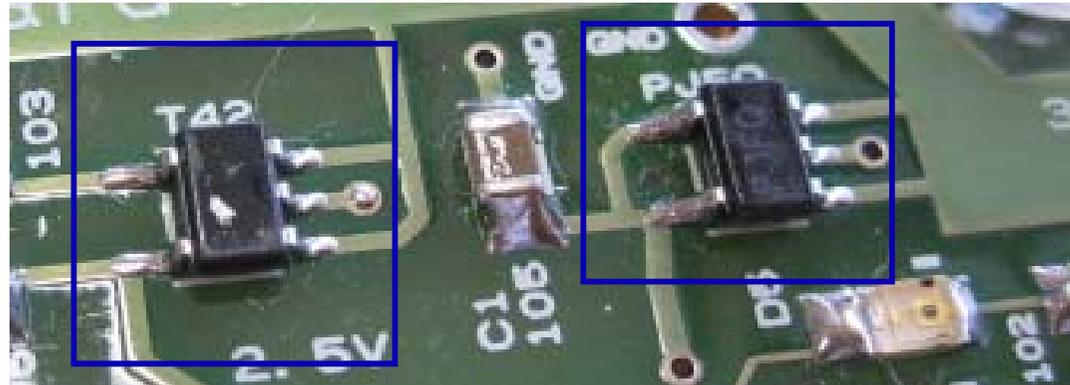
3.3Vの供給により,
POWER LED (D1)が点灯することを確認してもらう



3. オシレータ, レギュレータ, ICの固定 (30~50分)



(a) オシレータ



(b) レギュレータ 2個 (異なる部品なので固定場所に注意)



(c) PROM



(d) SRAM



(e) FPGA

写真ははんだ付け後のもの



3. オシレータ, レギュレータ, ICの固定 (30~50分)

- **悪い例**

- FPGAが右側に寄っている.
- 2つのピンがショートするため, 正しく動作しない.

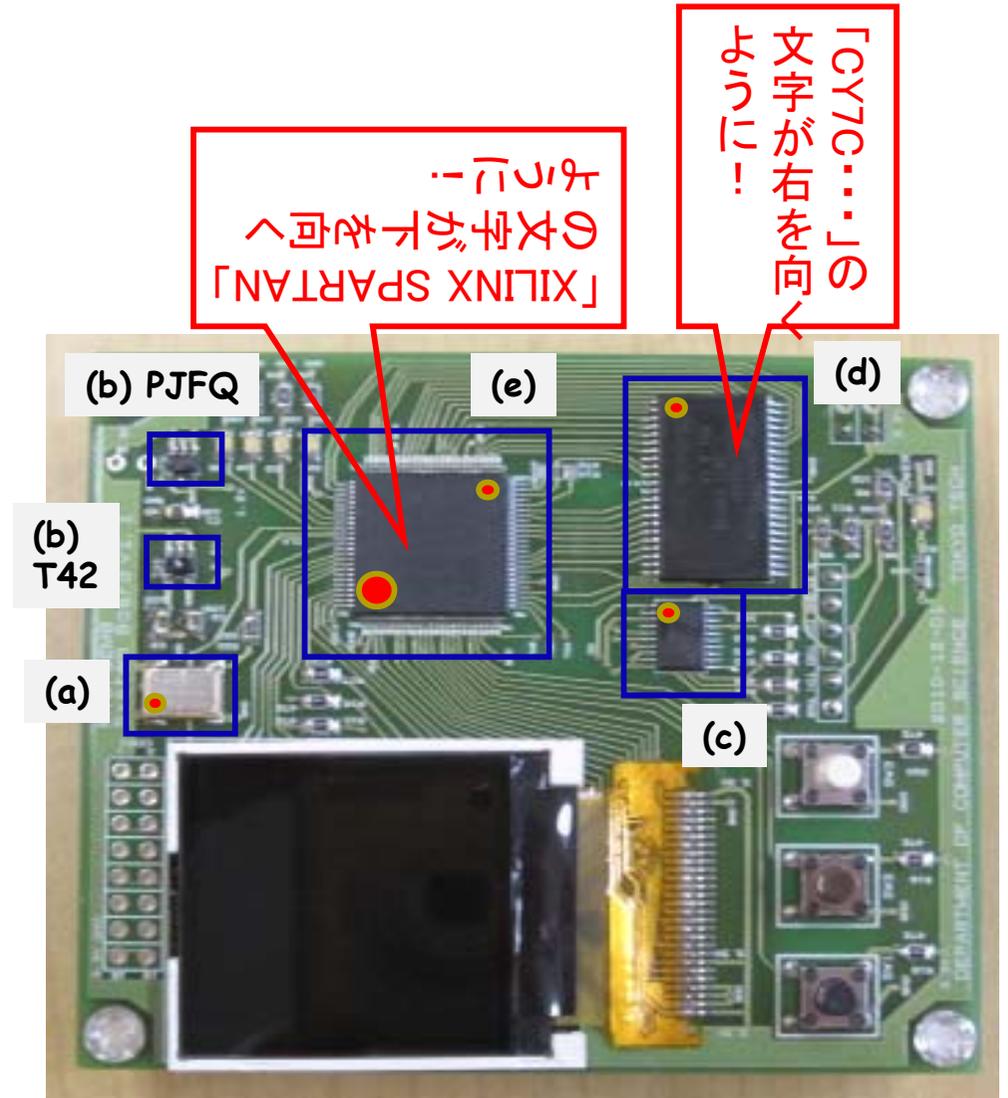


3. オシレータ, レギュレータ, ICの固定 (30~50分)



接着剤を用いて, オシレータ, レギュレータ, ICを固定

- (a) クロックオシレータ
 - ・ 方向に注意. 右写真, 丸印が左下になる.
- (b) レギュレータ
 - ・ PJFQの刻印のあるものを上, T42と刻印のあるものを下に.
- (c) PROM
 - ・ 丸印が左上になるように.
 - ・ **ピンを確実に接続するように位置調整.**
- (d) SRAM
 - ・ 丸印が左上になるように.
 - ・ CY7C...の文字が右を向くように.
 - ・ **ピンを確実に接続するように位置調整.**
- (e) FPGA
 - ・ 丸印(大)が左下, 丸印(小)が右上
 - ・ Xilinx SPARTANの文字が下を向くように.
 - ・ **ピンを確実に接続するように位置調整.**



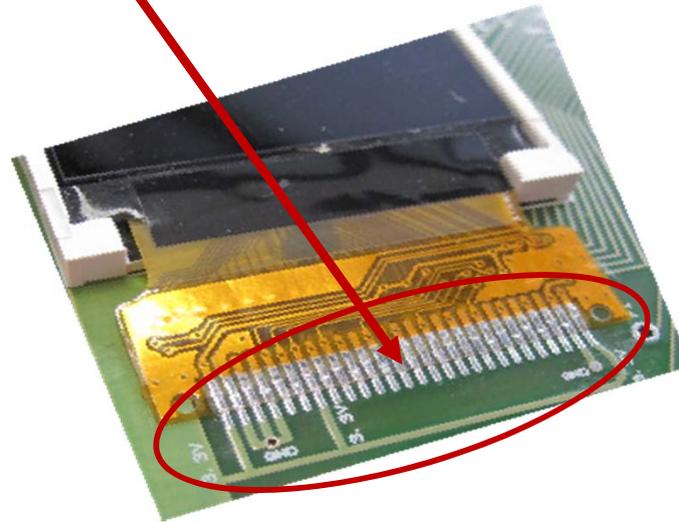
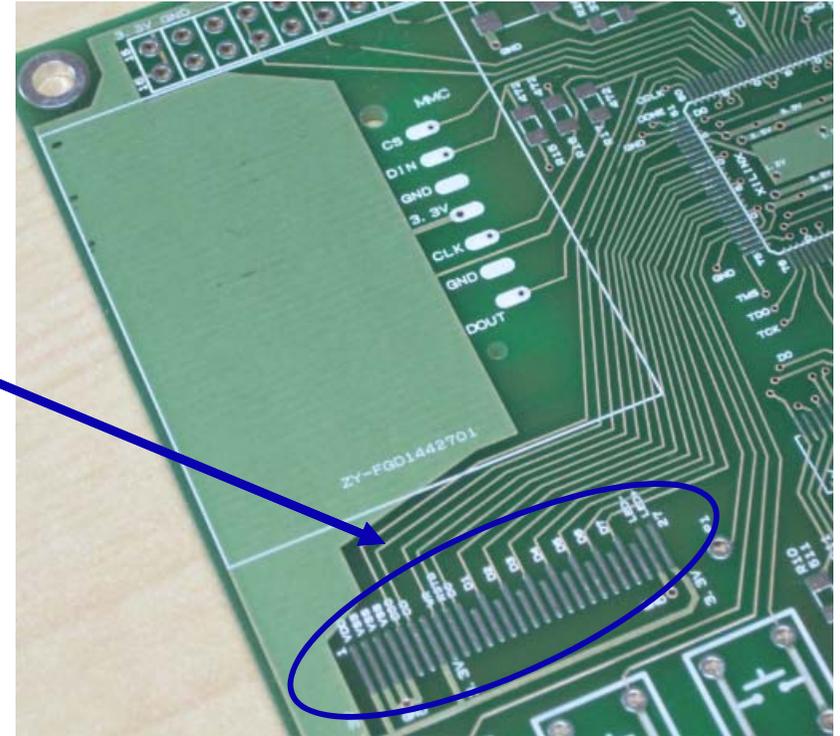
4. オシレータ, レギュレータ, ICのはんだ付け(30~50分)

- はんだ付けの前に, 完全に固定するまで10分の休憩をとる!
 - 使わない時にははんだごての電源を切ること.
- オシレータ, レギュレータ, ICのはんだ付け
 - **多量のフラックス**を利用する.
 - 2本のピンが接続されるブリッジがおきないように.
 - はんだが多すぎる場合には, はんだ吸取線を使う.
または, TAに相談.
- (CP6)



5. 液晶モジュール等のはんだ付け（20～40分）

- 液晶モジュール等のはんだ付け(1/2)
 - (a) 液晶モジュール
 - 基板のこの部分にフラックスをぬる.
 - 基板のこの部分にはんだをもる.
 - さらに, その上にフラックスをぬる.
 - **位置を慎重に固定**して, 液晶モジュールの上からはんだ付け.
 - 基板と液晶モジュールの裏側がはんだ付けされる.



5. 液晶モジュール等のはんだ付け (20~40分)

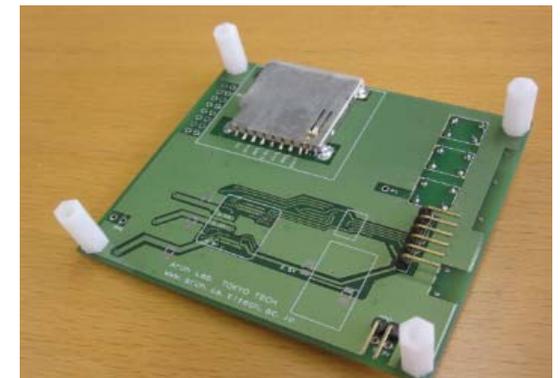
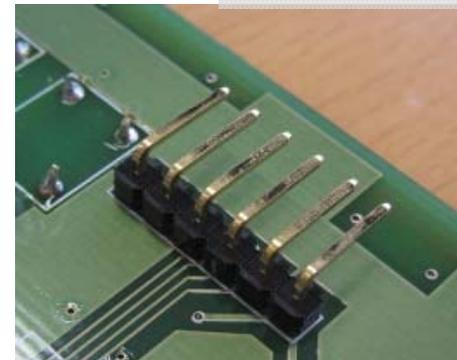
- 液晶モジュール等のはんだ付け(2/2)
 - (b) スイッチ
 - 3個のスイッチをはんだ付け.
 - (c) SDカードスロット
 - 基板裏面に, SDカードスロットをはんだ付け.
 - (d) 6ピンヘッダ
 - 基板裏面に, 6ピンヘッダをはんだ付け
 - (e) 2ピンヘッダ
 - 基板裏面に, 2ピンヘッダをはんだ付け
 - (d) スペーサ
 - ネジで4個のスペーサを固定する.



裏面にはんだ付け



裏面にはんだ付け



基板裏面

裏面にはんだ付け

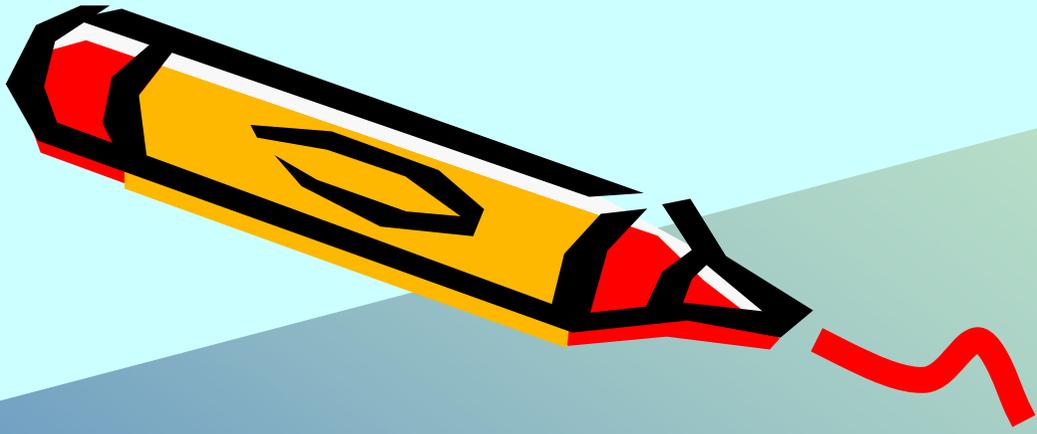


動作確認

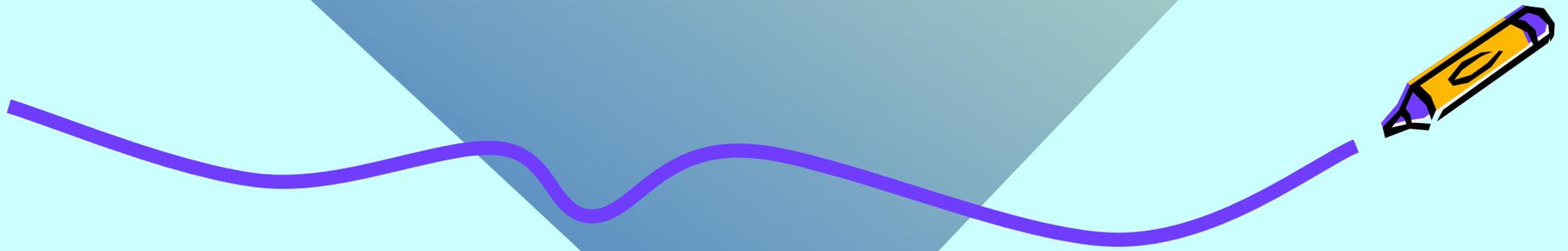
- テスターを用いて電圧を確認
 - 2.5V
 - 1.2V
- FPGAの動作確認
- PROMの動作確認
- スイッチの動作確認
- LEDの動作確認
- 液晶モジュールの動作確認
- SDカードの動作確認

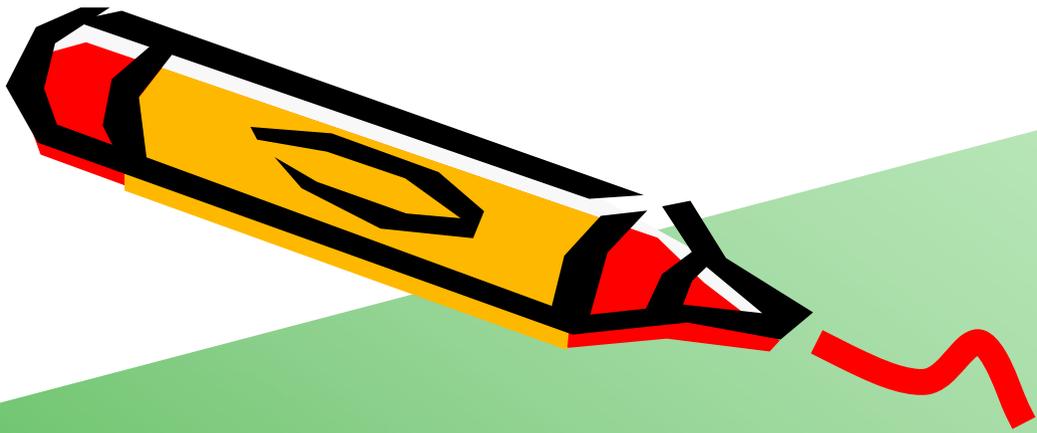
- テスターの操作マニュアルは以下
 - <http://akizukidenshi.com/download/P-10manual.pdf>
- ハードウェアデバッグについては以下
 - Z:\¥Emb¥Doc¥ Emb-HWDebug.pdf





実験 4日目～5日目



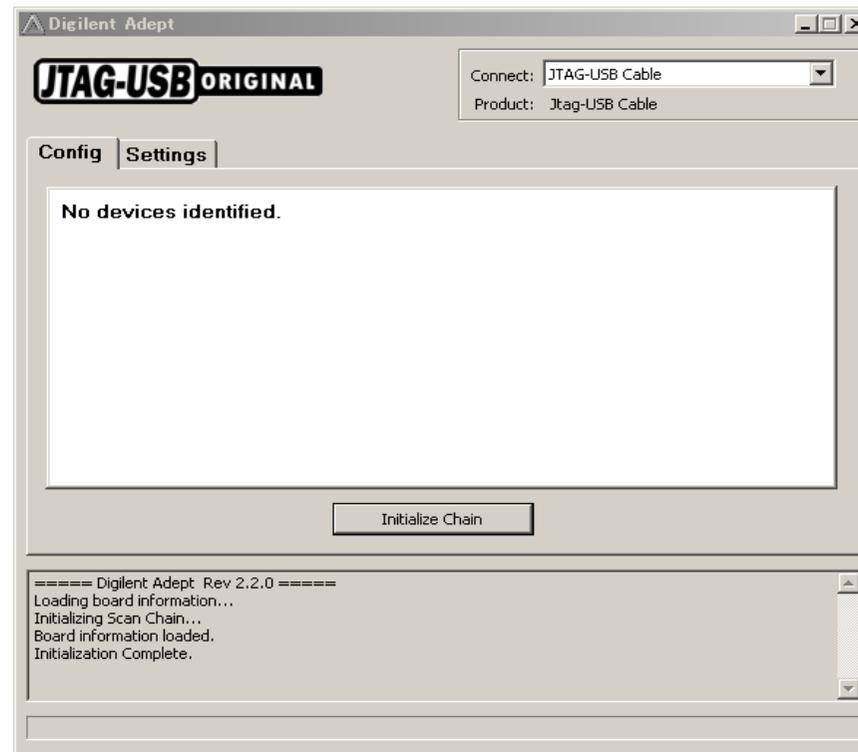


Digilent Adeptの使い方



Digilent Adept

- FPGAの構成データをFPGAやPROMに書き込むソフトウェア
- **ソフトウェアの起動**
 - C:\Program Files\Digilent\Adept\Adept.exe

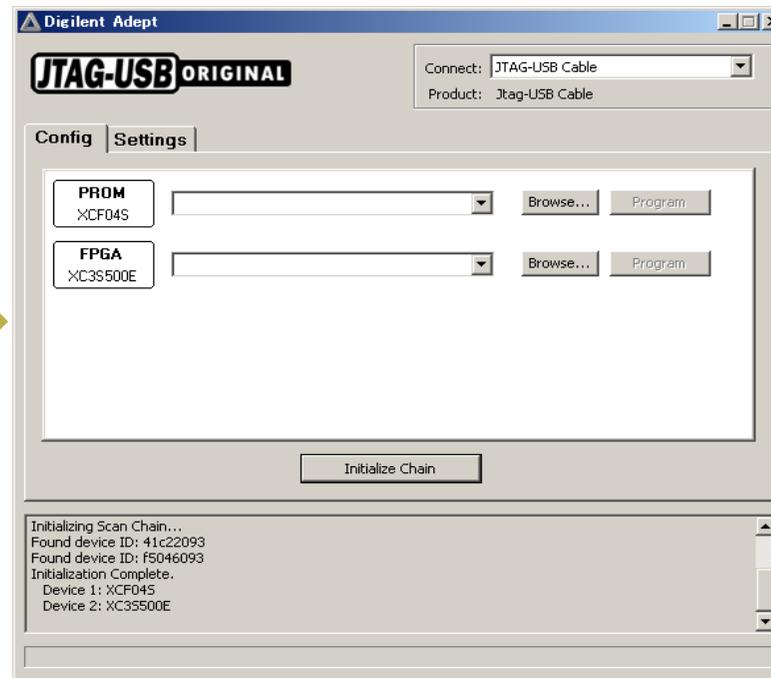
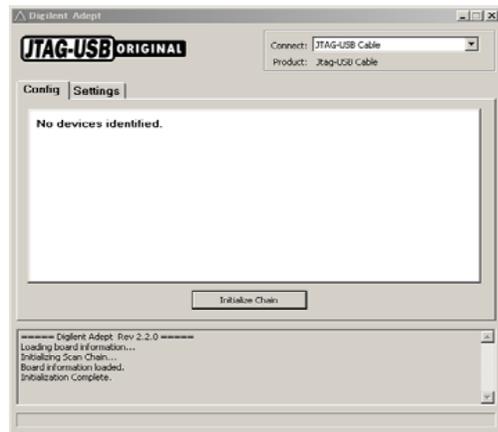


FPGAへの書き込み準備

- JTAGケーブルを計算機のUSBポートに接続する。
- JTAGケーブルと組み込みシステムHWを接続する。
 - 方向に注意, 6ピンを確実に接続
 - 間違えると, ケーブル破損することがある
- 組み込みシステムHWの電源を入れる。
- Digilent Adept の Initialize Chain ボタンを押す。
- Adept に PROM, FPGA が表示される。

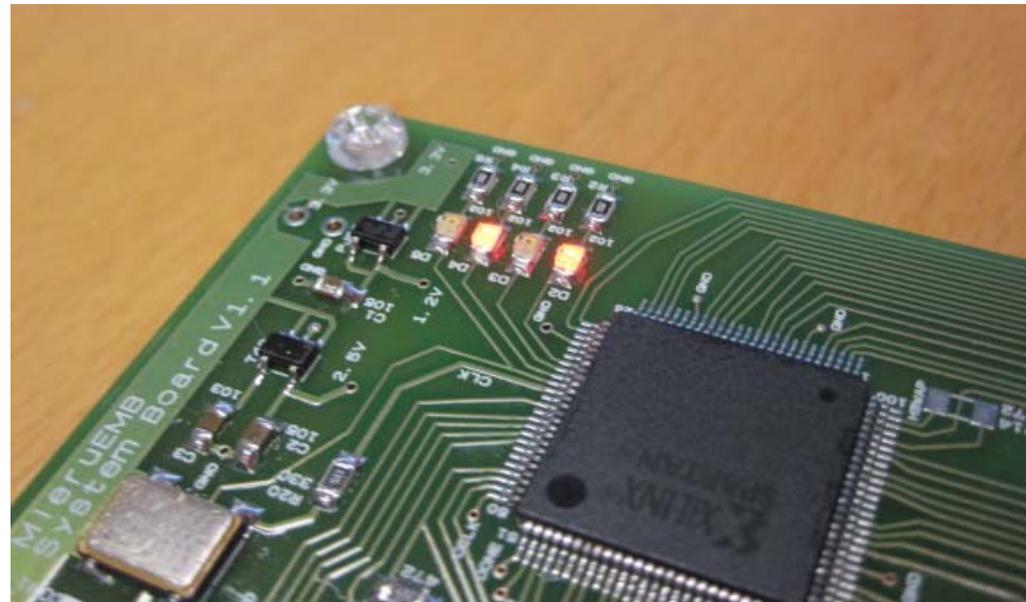
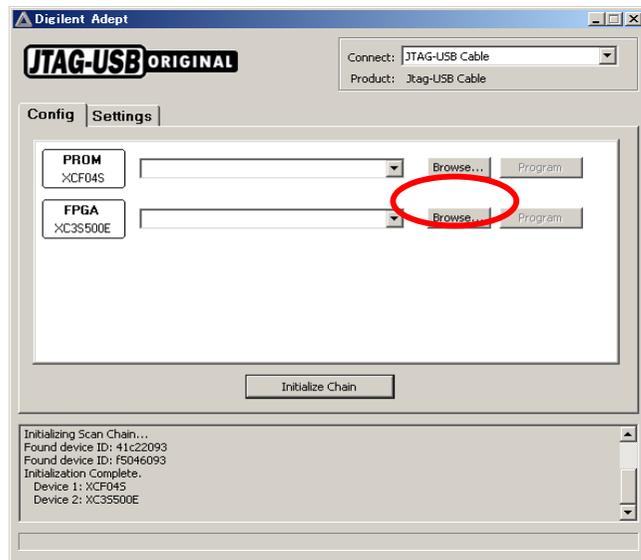


JTAGケーブル



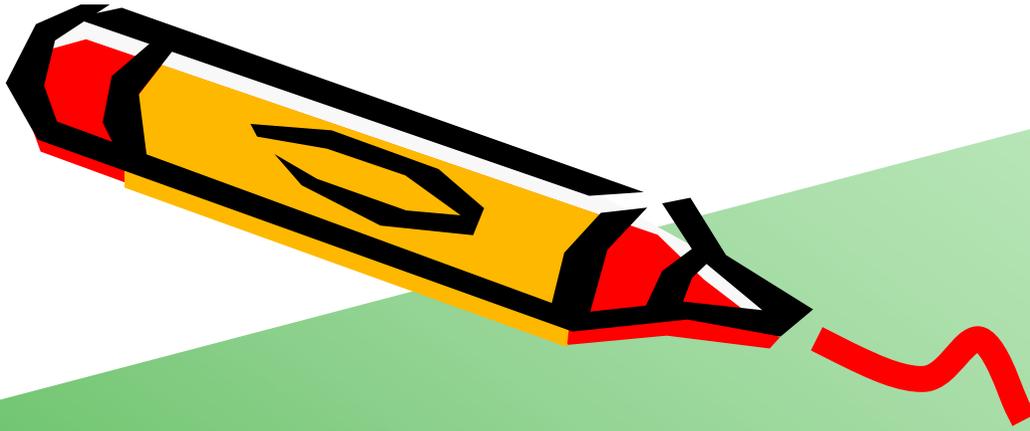
FPGA/PROMへの書き込み

- FPGA右の Browseボタンをクリック。
 - Z:\¥Emb¥Bitfile¥main01.bit を選択
 - “Startup clock for this file is ... ” といったメッセージには Yes をクリック。
 - Programボタンをクリックすると、FPGAに回路情報が書き込まれる。
 - FPGAは揮発性なので、電源を切ると回路情報が消えてしまう。
 - **不揮発性のPROM**に書き込むと、電源投入時に自動でその回路情報がFPGAにロードされる。



main01.bit には D2, D4が点灯する回路情報が格納されている。



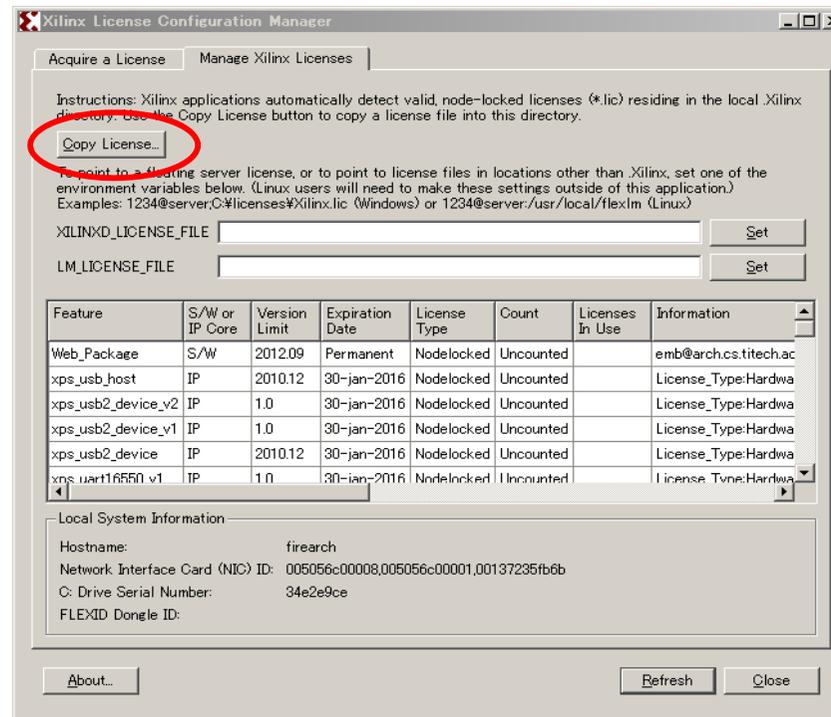


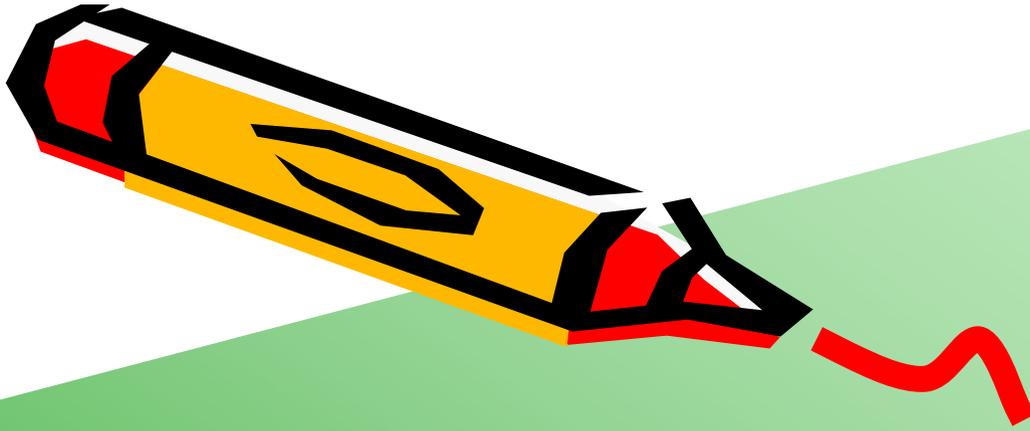
Xilinx ISE WebPACKの設定



Xilinx ISE WebPACK ライセンスファイルの設定

- ISE WebPACKを起動してライセンスファイルを設定する。
 - Help → Manage License
 - Copy Licenseボタンをクリック
 - **Z:¥Emb¥Doc¥Xilinx.lic** を選択
 - ライセンスファイルがZドライブにコピーされて、利用可能になる。



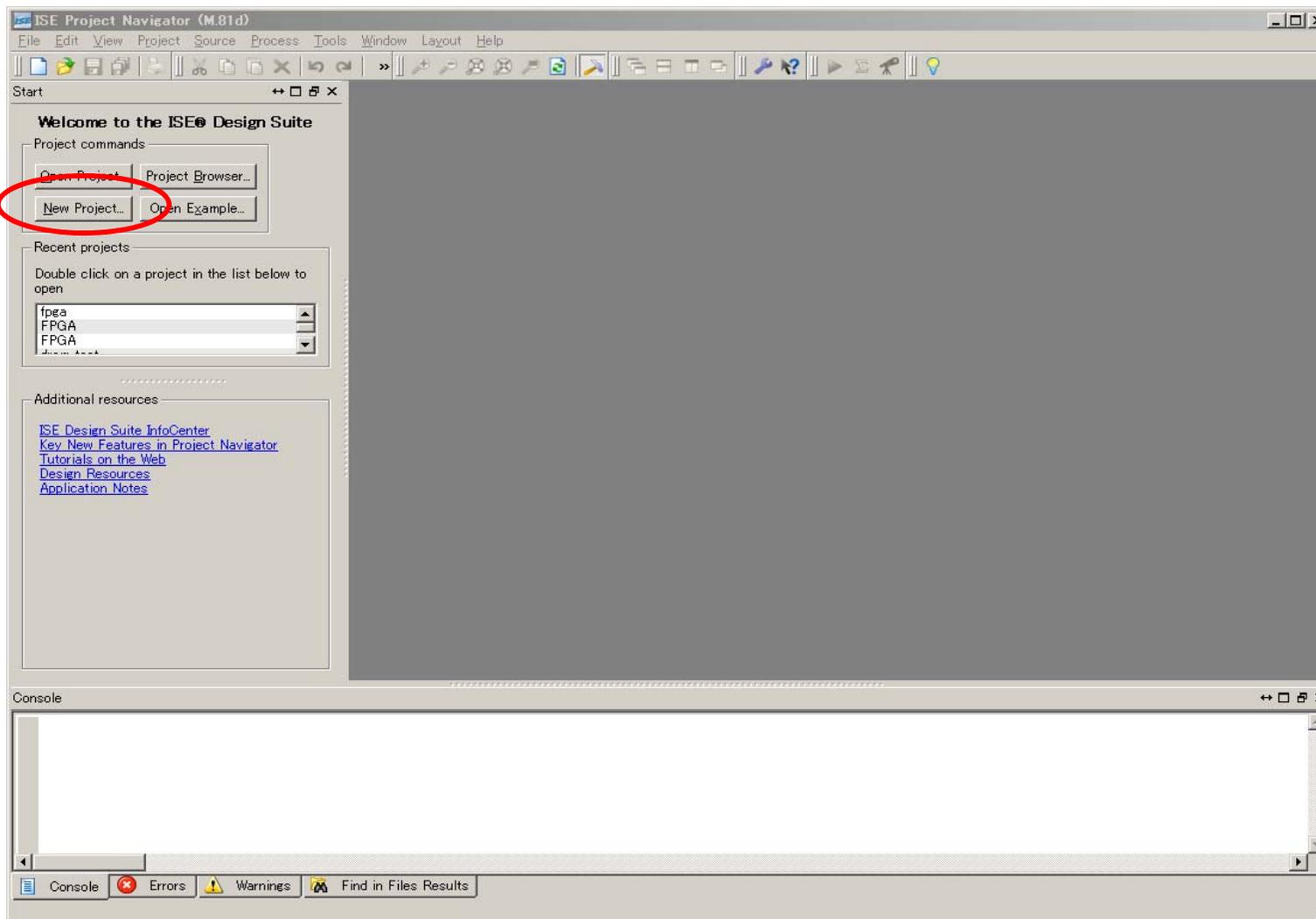


ISE WebPACKを用いたFPGA開発(1)



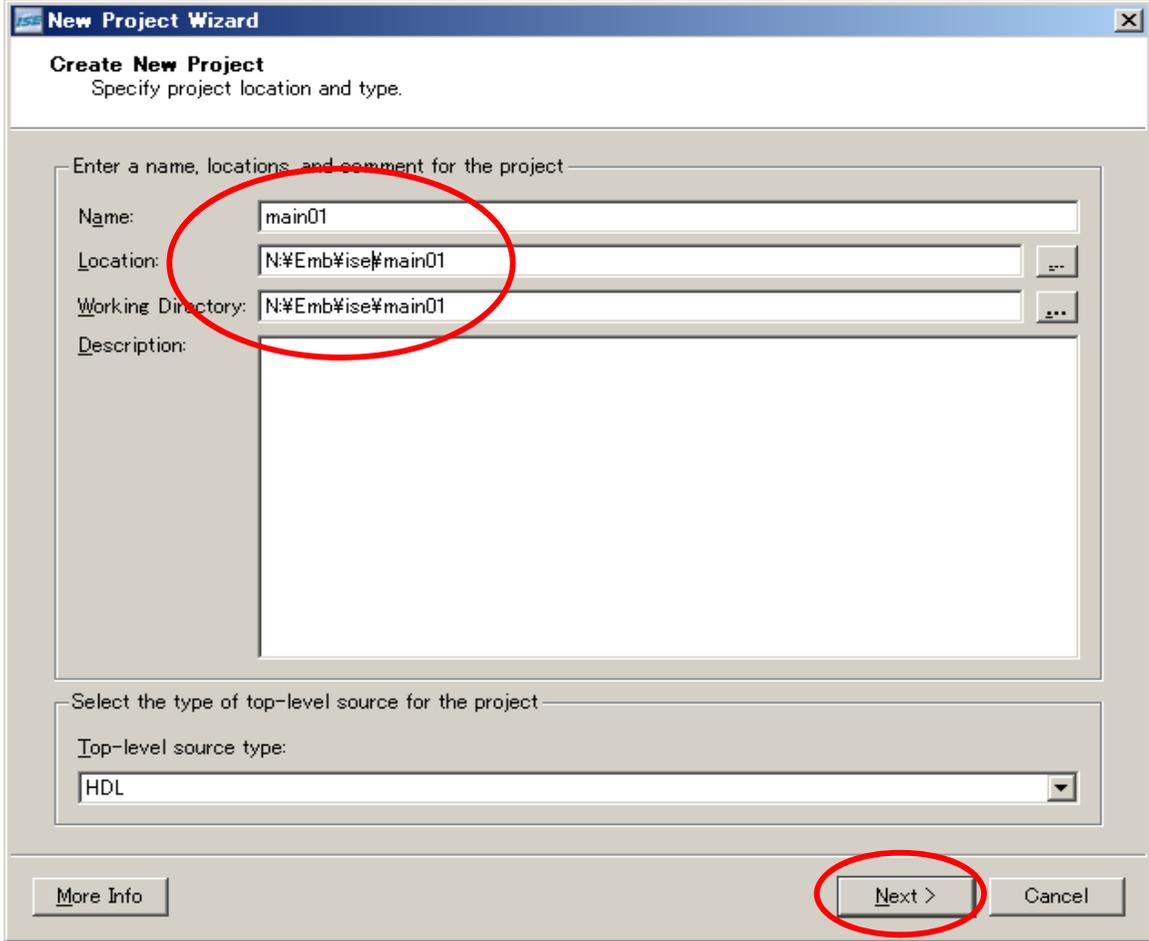
ISE Project Navigator の起動, プロジェクトの新規作成

- New Project ボタンをクリック



プロジェクトを保存するディレクトリを指定

- ドライブはZドライブを用いるので、Location に、**Z:¥Emb¥ise** を指定、Name に、**main01** を指定する。
- Next ボタンをクリック



The screenshot shows the 'New Project Wizard' dialog box with the following fields:

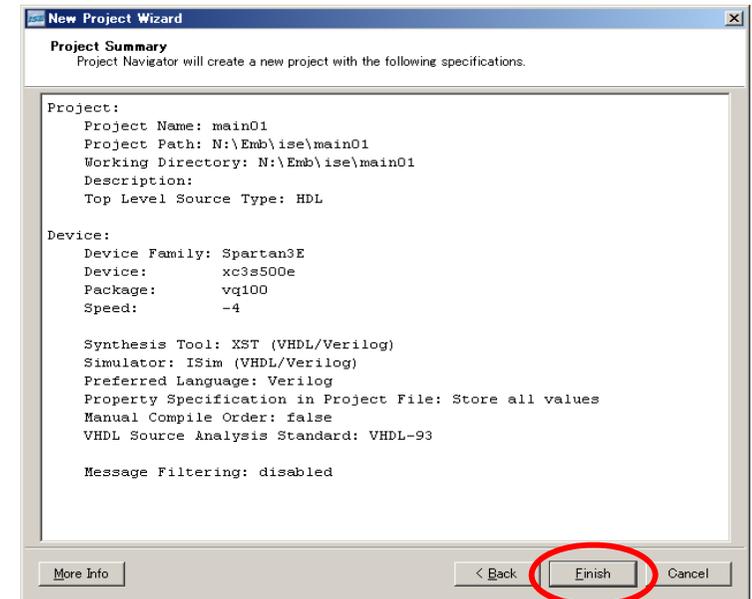
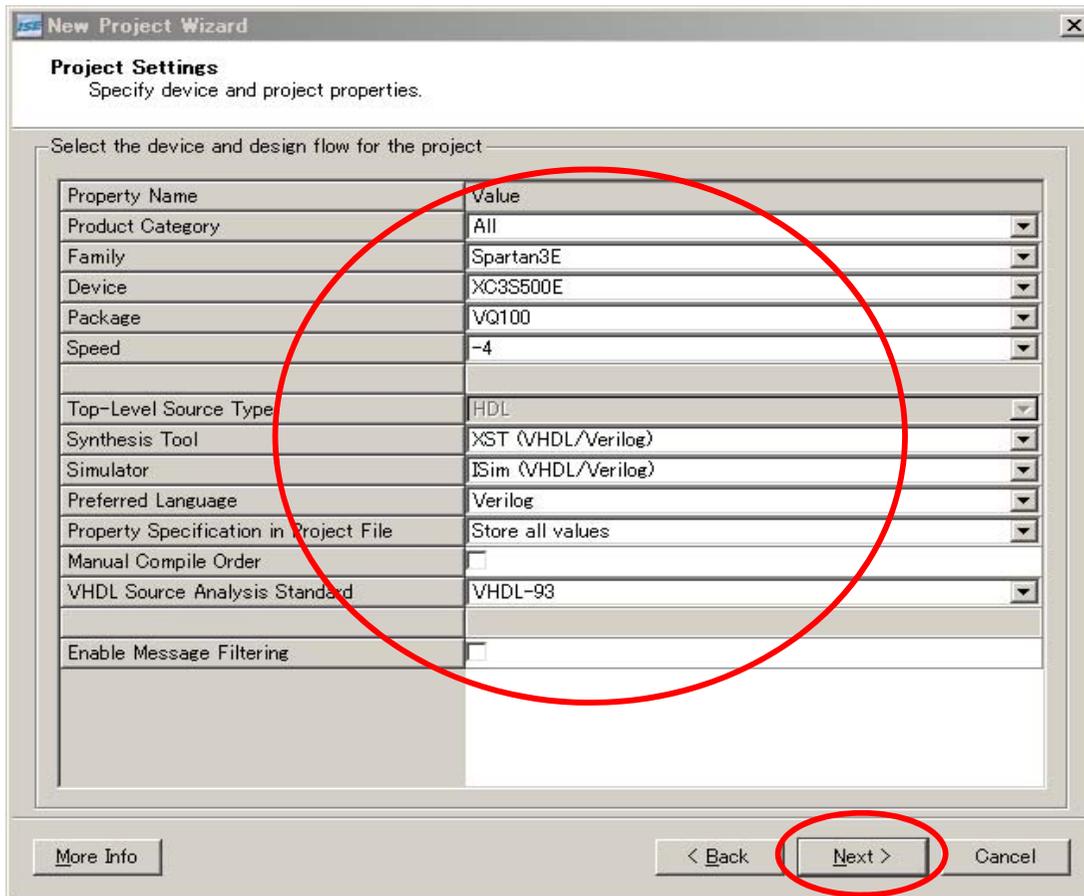
- Name: main01
- Location: N:¥Emb¥ise¥main01
- Working Directory: N:¥Emb¥ise¥main01
- Description: (empty)
- Top-level source type: HDL

The 'Next >' button is circled in red, indicating the next step in the wizard.



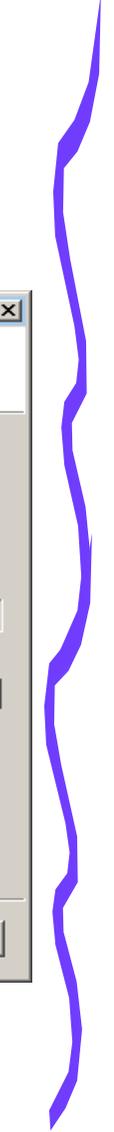
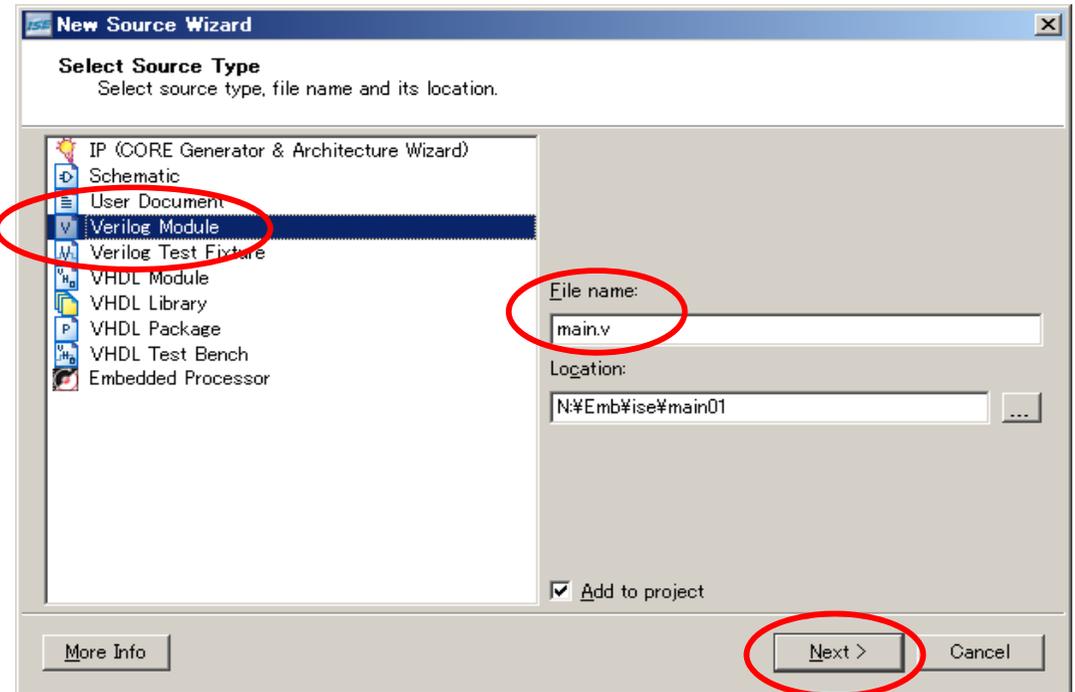
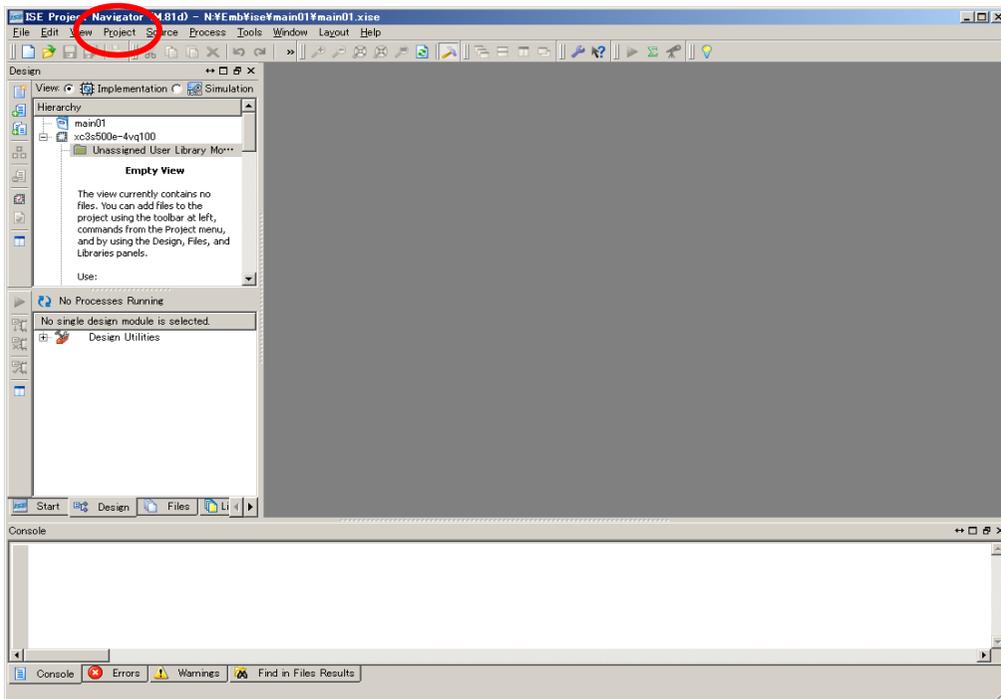
利用するFPGA等の設定

- 利用するFPGAの種類などを正確に指定する。
- 上から, **All, Spartan3E, XC3S500E, VQ100, -4, XST, ISim, Verilog, Store all values, VHDL-93**
- Nextボタンをクリック, 確認画面で Finishボタンをクリック



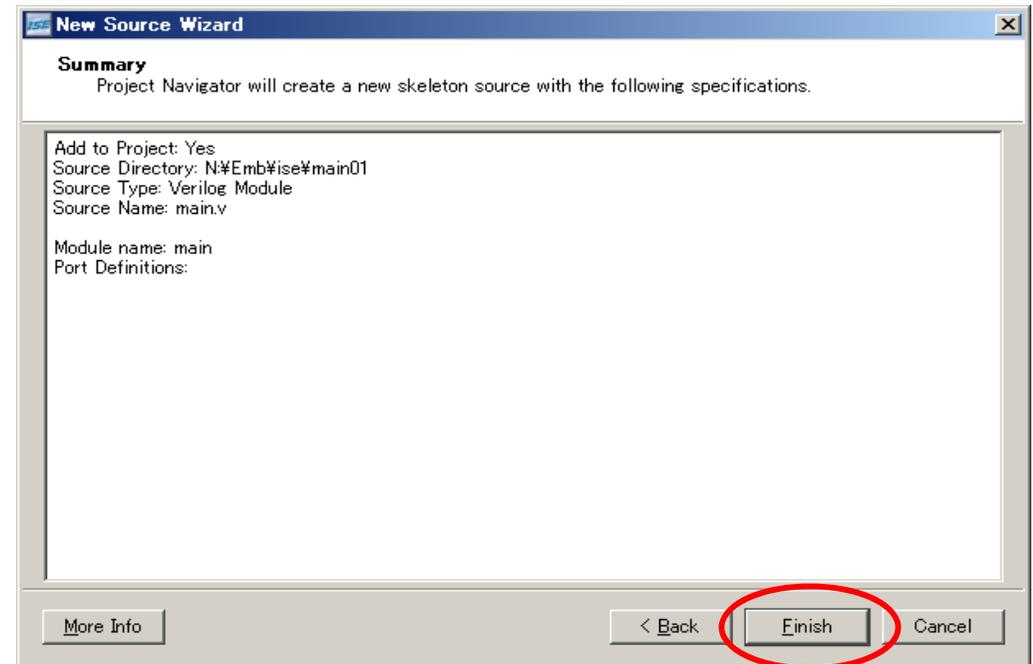
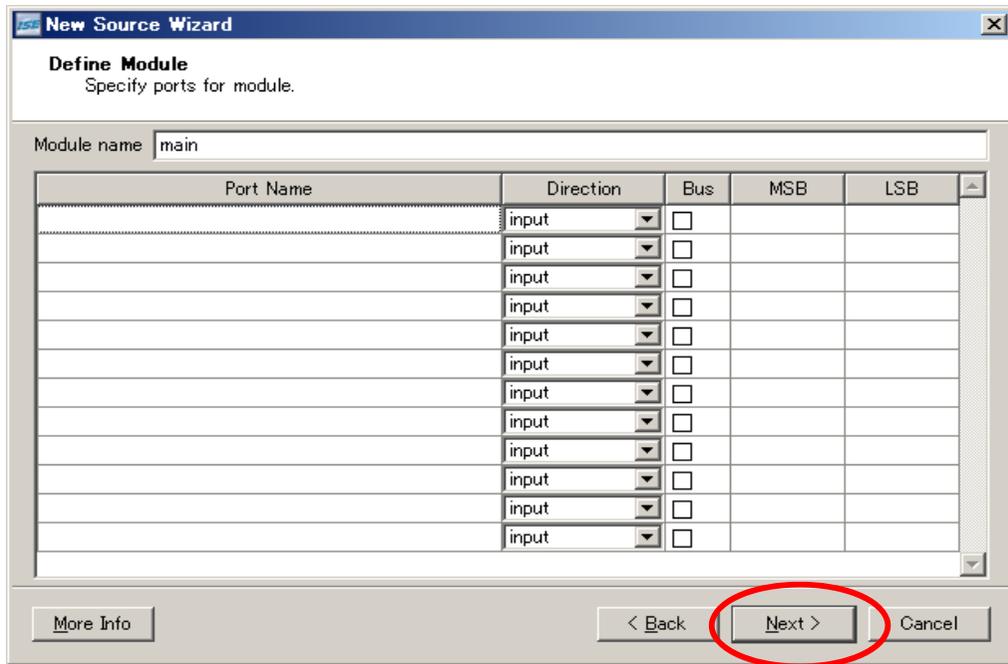
Verilogソースコードの追加

- Project -> New Source を選択
- New Source Wizard にて, **Verilog Module** を選択し, ファイル名 **main.v** を入力し, Nextボタンをクリック



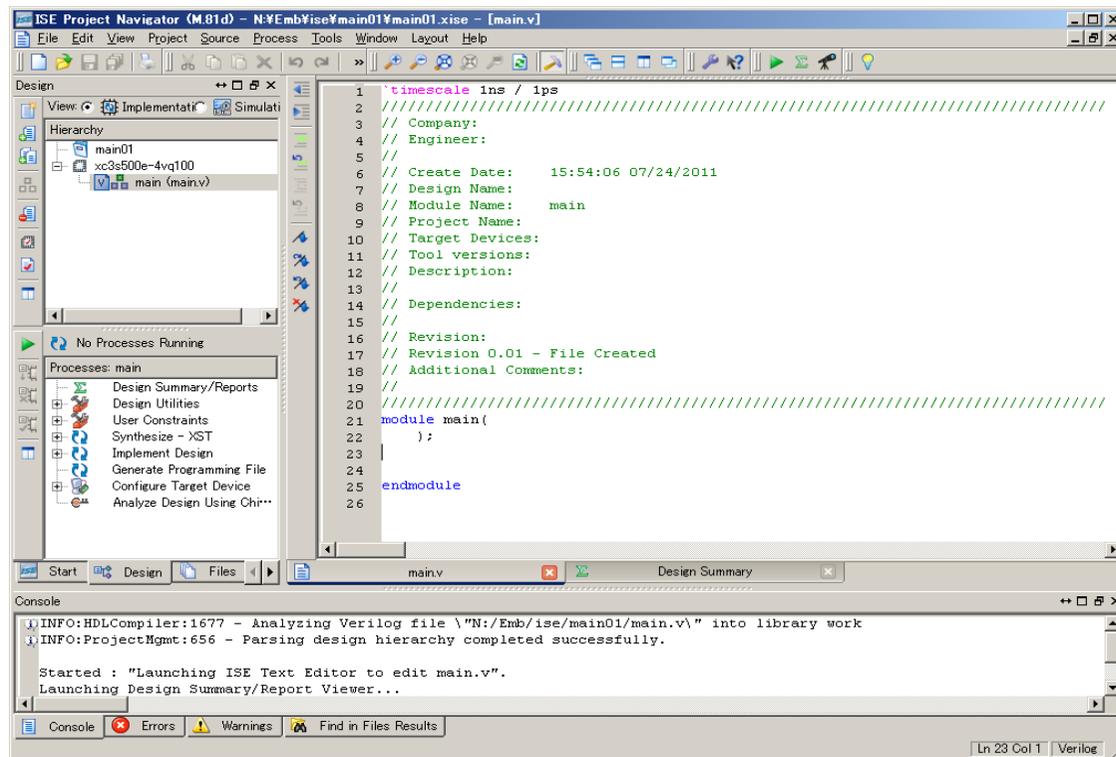
Verilogソースコードの追加

- Define Module では, 何も入力せずに, Nextボタンをクリック
- サマリが表示される. Finishボタンをクリック



Verilog HDLの編集

- module main を編集する.

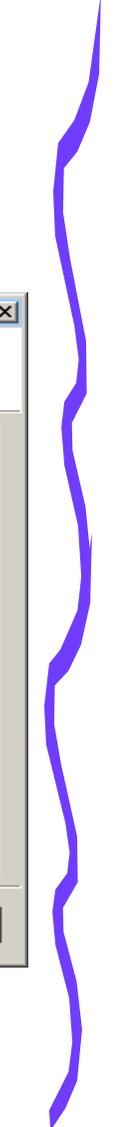
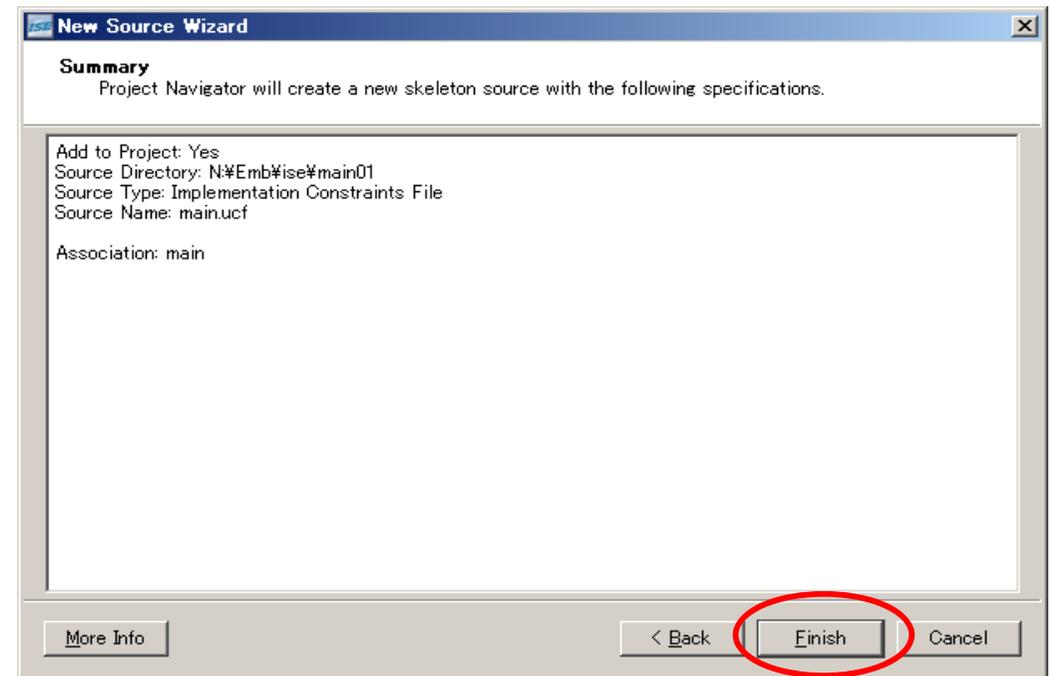
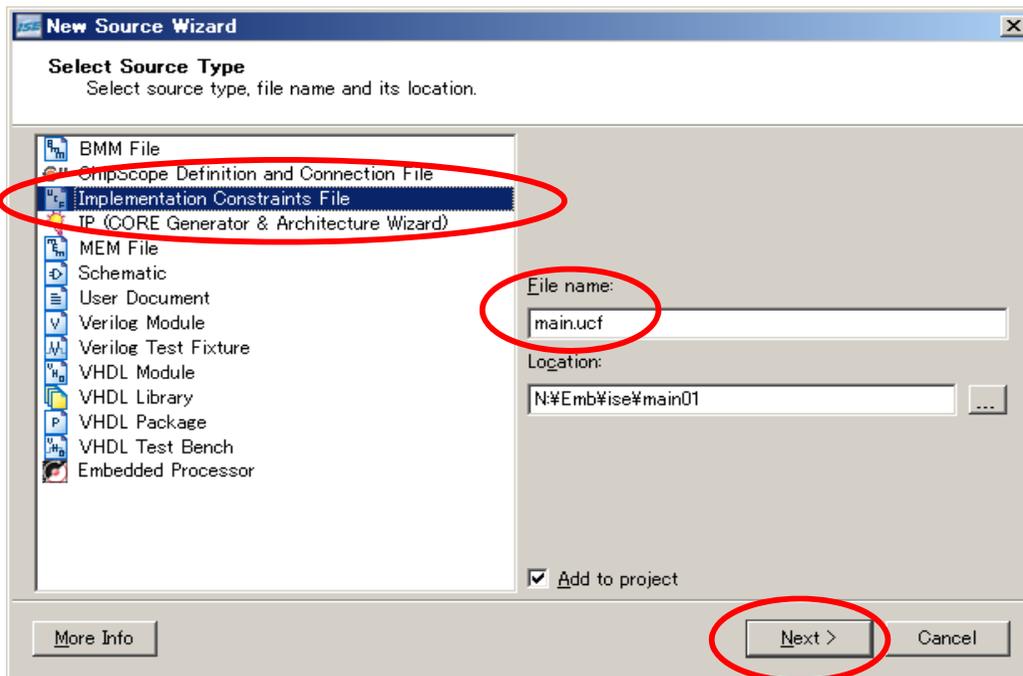


```
20 ///////////////////////////////////////////////////
21 module main(ULED
22     );
23     output [3:0] ULED;
24     assign ULED = 5;
25
26 endmodule
27
```



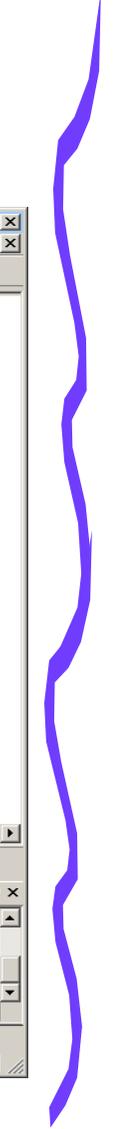
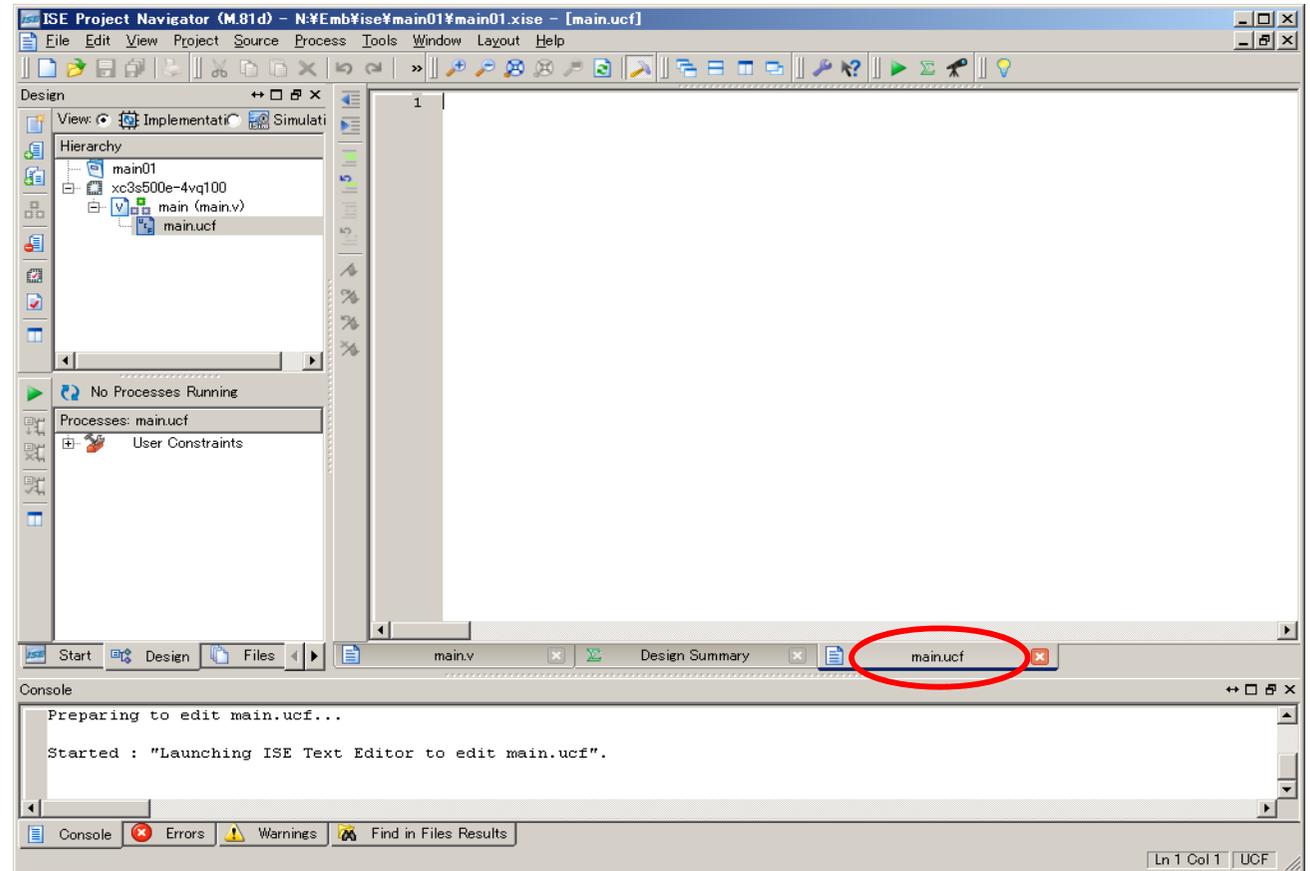
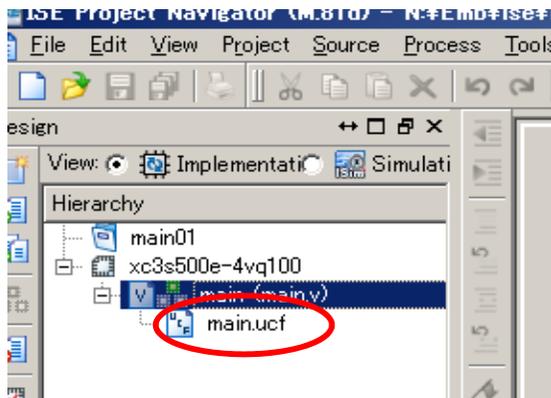
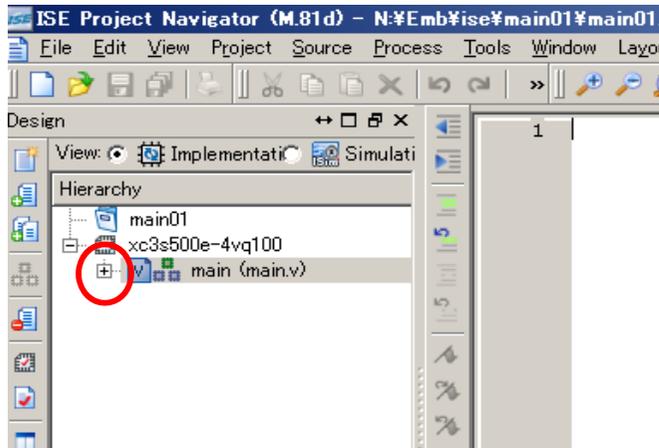
UCF(User Constraints File)の追加

- Project -> New Source を選択
- New Source Wizard にて, **Implementation Constraints File** を選択し, ファイル名 **main.ucf** を入力し, Nextボタンをクリック
- サマリが表示される. Finishボタンをクリック



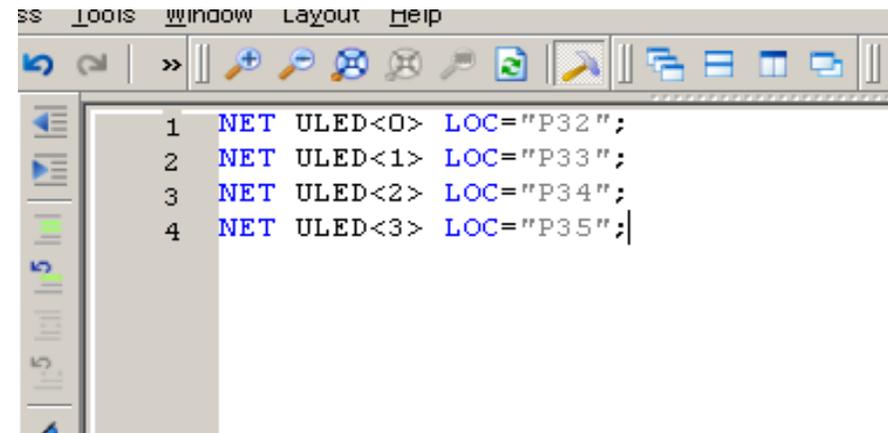
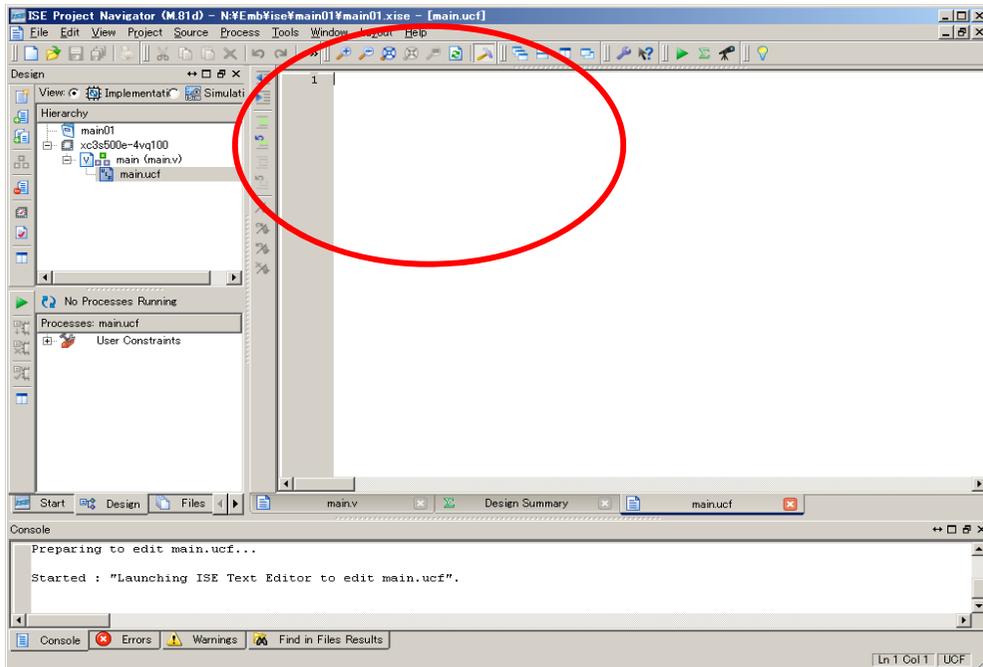
UCFの追加と編集の準備

- Hierarchy の main 左の + をクリック
- main.ucf が現れる.
- main.ucf の上でダブルクリック, main.ucf が編集可能となる.



UCFの編集

- UCFを右図の様に編集して保存,
保存は「Ctrl + S」のショートカット, または, File -> Save を選択

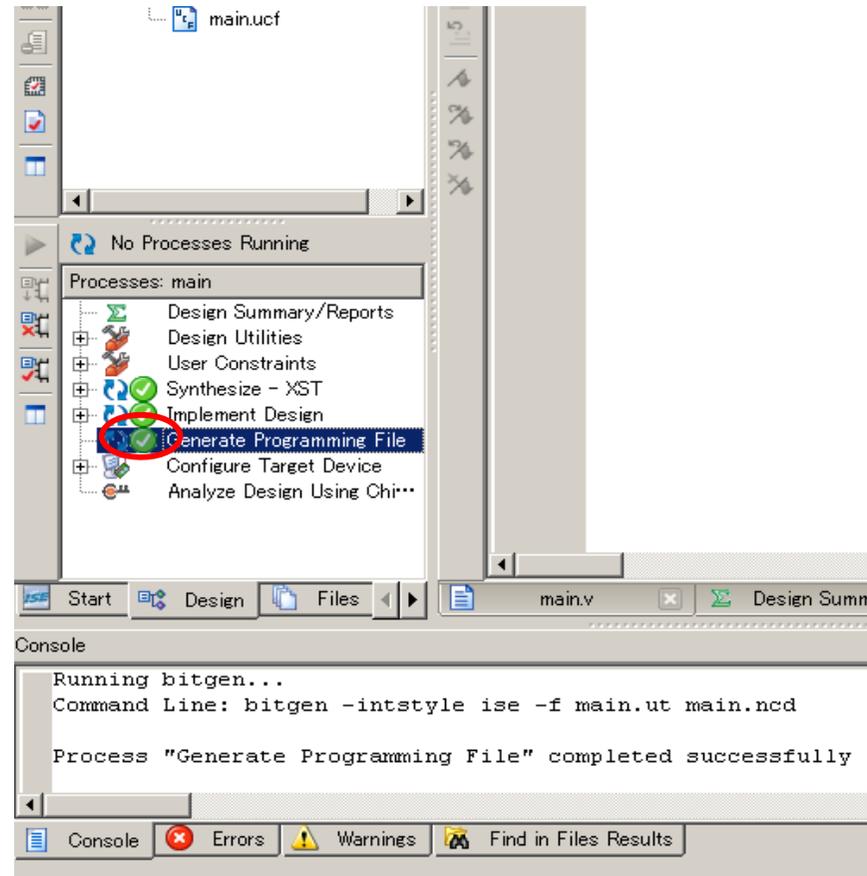
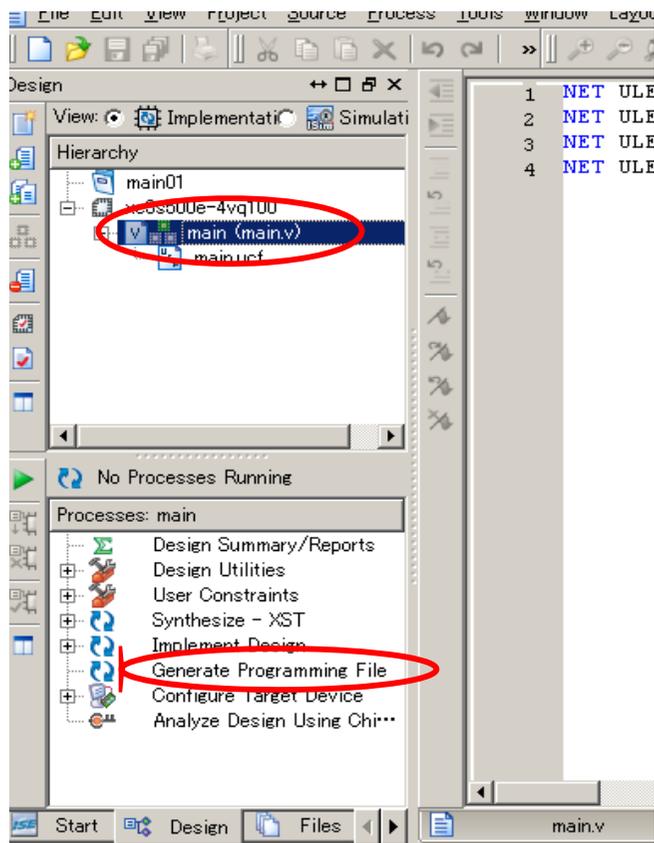
A screenshot of the ISE Text Editor window showing the UCF code. The code is as follows:

```
1 NET ULED<0> LOC="P32";  
2 NET ULED<1> LOC="P33";  
3 NET ULED<2> LOC="P34";  
4 NET ULED<3> LOC="P35";
```



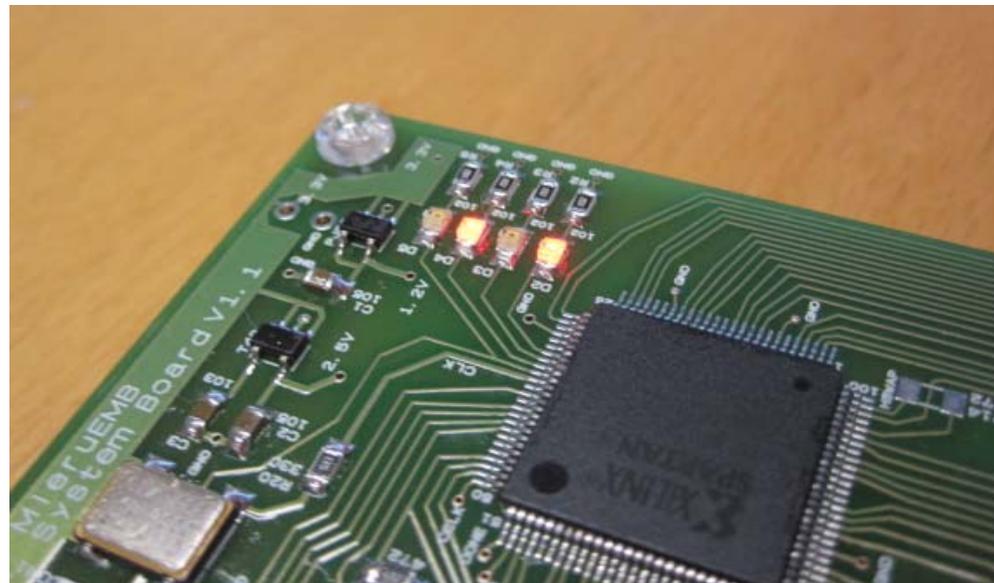
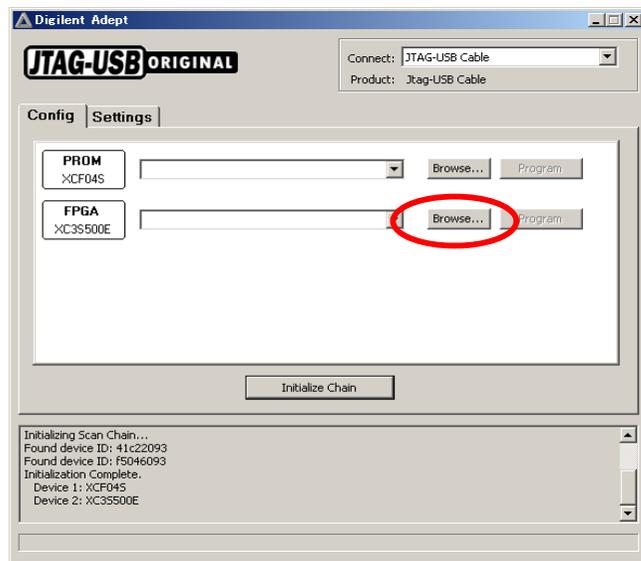
論理合成

- main (main.v) をクリックすると, Processes に項目が表示される.
- Processes: main から, Generate Programming File をダブルクリック, 論理合成を始める.
- しばらくすると, Generate Programming File の左に緑のチェックが表示されれば成功.



FPGA/PROMへの書き込み

- Digilent Adept を起動, FPGA右の Browseボタンをクリック。
 - Z:\¥Emb¥ise¥main01¥main.bit を選択
 - Programボタンをクリックすると, FPGAに回路情報が書き込まれる。
 - FPGAは揮発性なので, 電源を切ると回路情報が消えてしまう。
 - 不揮発性のPROMに書き込むと, 電源投入時に自動でその回路情報がFPGAにロードされる。



main01.bit には D2, D4が点灯する回路情報が格納されている。



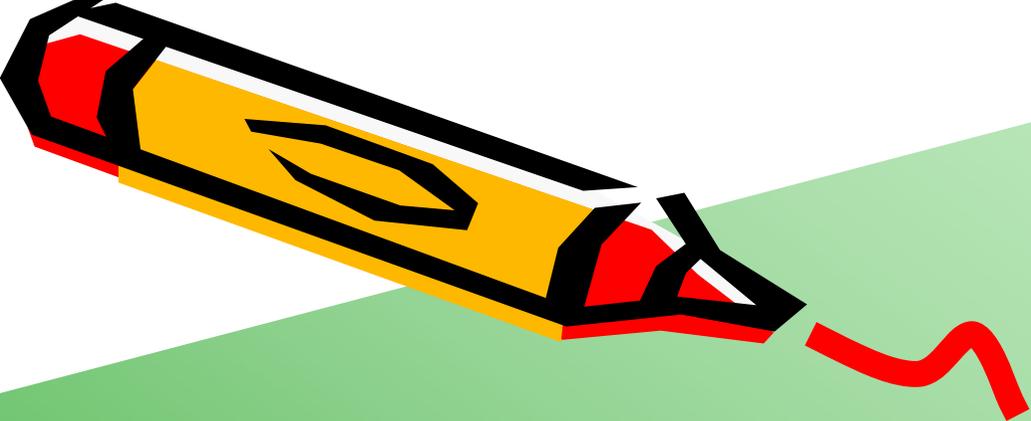
Verilog HDLの修正

- 24行目の ULED = 5; の値を変更して、どのような変化が起きるか試してみる。

```
20 ///////////////////////////////////////////////////  
21 module main(ULED  
22     );  
23     output [3:0] ULED;  
24     assign ULED = 5;  
25  
26 endmodule  
27
```

2進数で記述すると,
assign ULED = 4'b101; となる。





ISE WebPACKを用いたFPGA開発(2)



シンプルな回路(AND回路とOR回路)の例

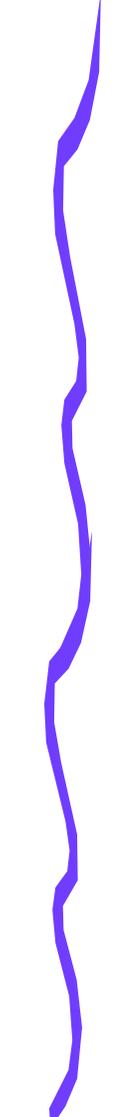
- 先の例と同様に,
Z:¥Emb¥ise¥main02 のプロジェクトを作成
- main.v と main.ucf を示す様に入力し, 論理合成, FPGAに書き込む.
- スイッチを押して, LEDがどのように変化するか確認する.

```
module main(SW, ULED
);
input [2:0] SW;
output [3:0] ULED;

wire sw0 = ~SW[0];
wire sw1 = ~SW[1];
wire sw2 = ~SW[2];

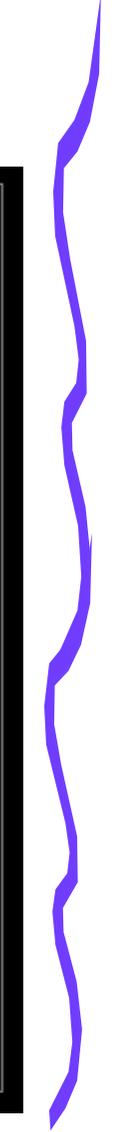
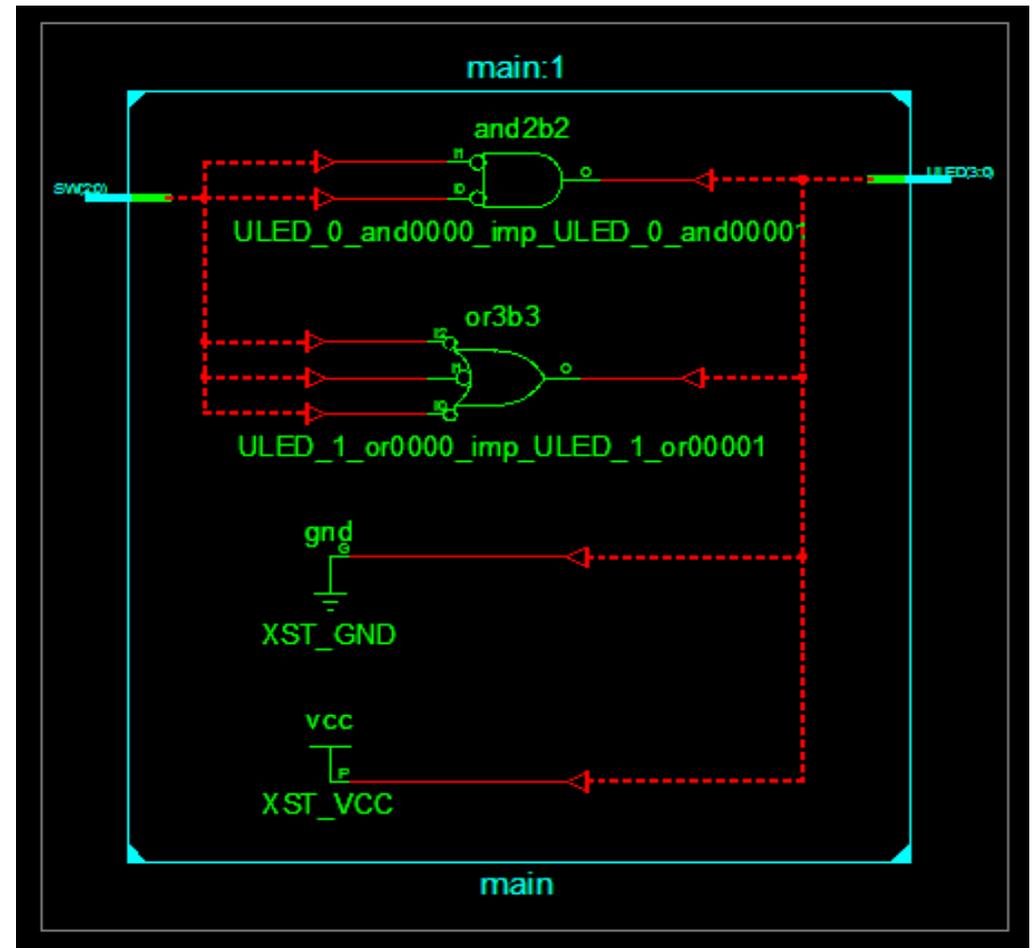
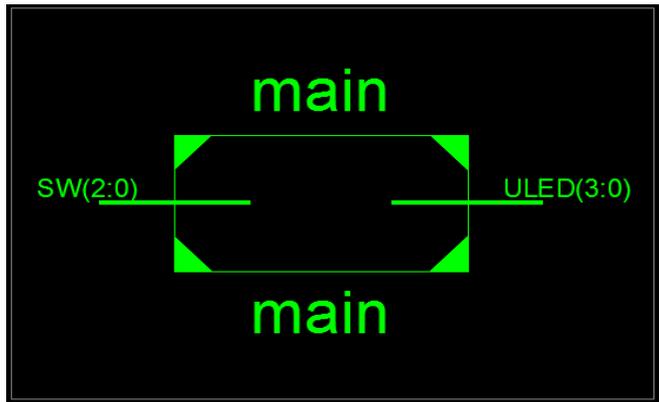
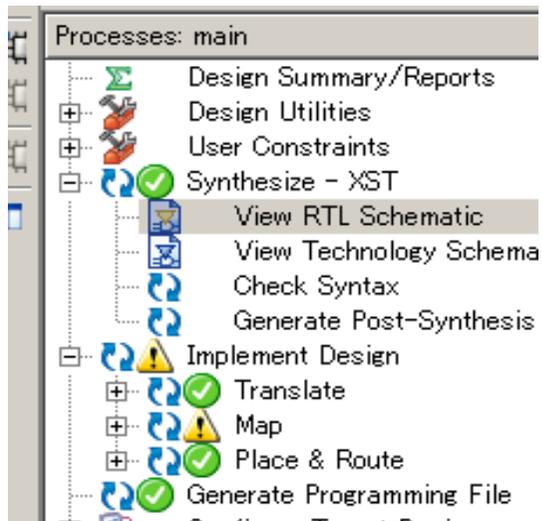
assign ULED[0] = sw0 & sw1;
assign ULED[1] = sw0 | sw1 | sw2;
assign ULED[2] = 0;
assign ULED[3] = 1;
endmodule
```

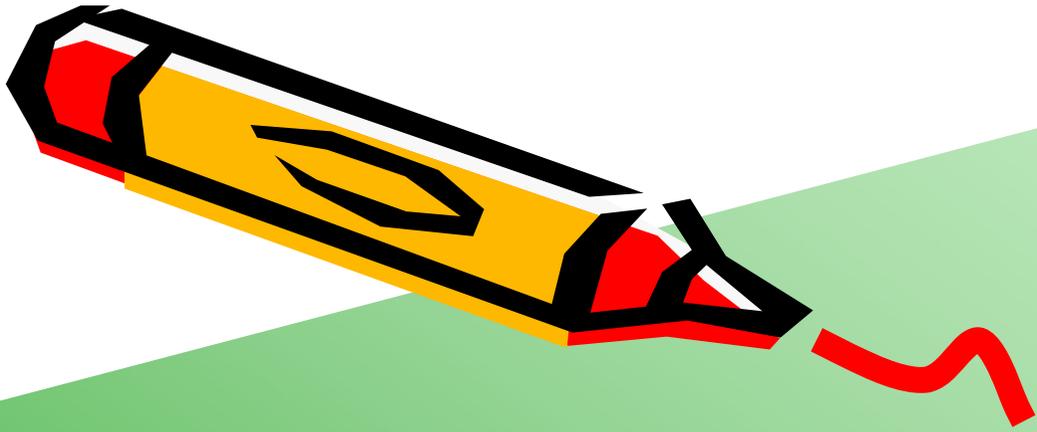
```
1 NET ULED<0> LOC="P32";
2 NET ULED<1> LOC="P33";
3 NET ULED<2> LOC="P34";
4 NET ULED<3> LOC="P35";
5 NET SW<0> LOC="P68";
6 NET SW<1> LOC="P70";
7 NET SW<2> LOC="P71";
8
```



回路の確認

- Synthesis -XST から, View RTL Schematic をダブルクリック
- ブロック図の main をダブルクリック
- AND回路, OR回路(3入力)が生成されていることを確認できる.





ISE WebPACKを用いたFPGA開発(3)



LEDを点滅させる回路(順序回路の例)

- 新規に, **Z:¥Emb¥ise¥main03** のプロジェクトを作成
- 入力の CLK は, 40MHz のクロック
- main.v と main.ucf を示す様に入力し, 論理合成, FPGAに書き込む.

```
module main(CLK, SW, ULED
);
input CLK;
input [2:0] SW;
output [3:0] ULED;

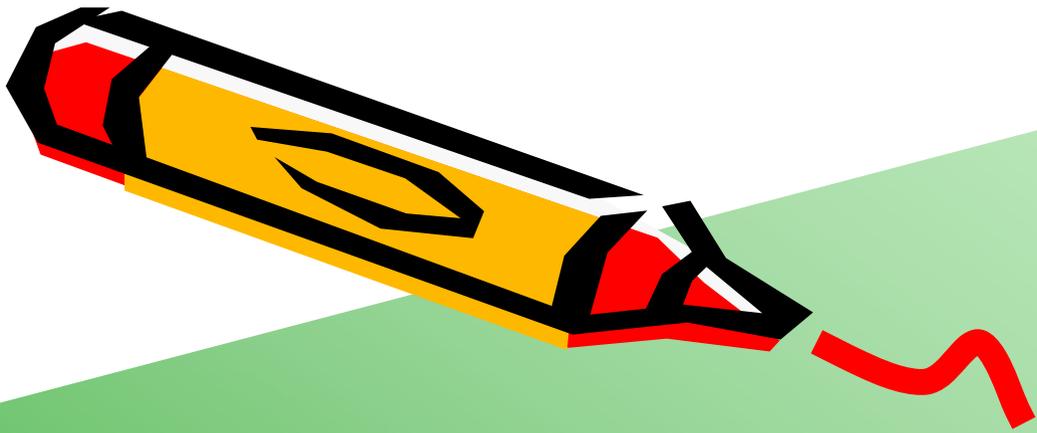
reg [25:0] cnt;

always @(posedge CLK) begin
    cnt <= cnt + 1;
end

assign ULED[0] = cnt[22];
assign ULED[1] = cnt[23];
assign ULED[2] = cnt[24];
assign ULED[3] = cnt[25];
endmodule
```

```
NET ULED<0> LOC="P32";
NET ULED<1> LOC="P33";
NET ULED<2> LOC="P34";
NET ULED<3> LOC="P35";
NET SW<0> LOC="P68";
NET SW<1> LOC="P70";
NET SW<2> LOC="P71";
NET CLK LOC="P36";
```



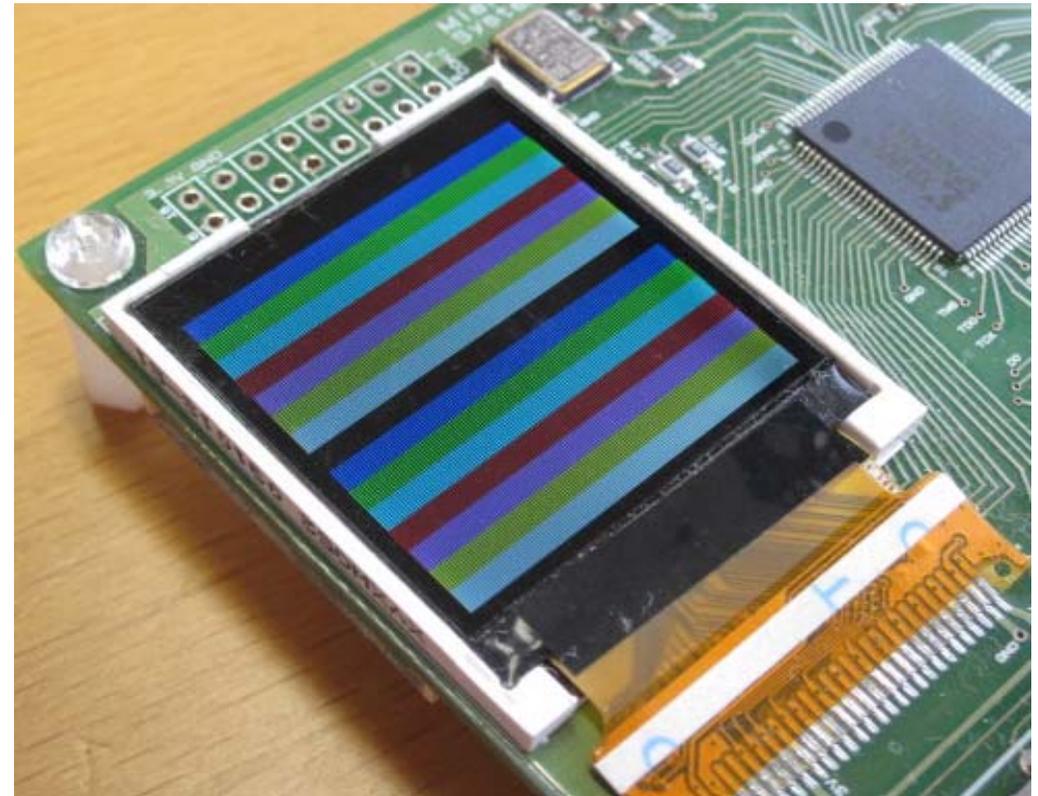
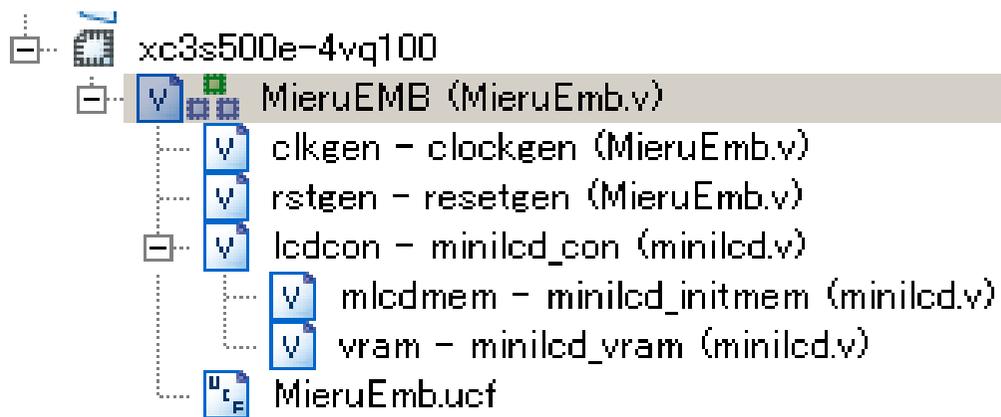


ナイトライダー回路



液晶モジュールのサンプルプロジェクト lcd01

- File → Open Project
 - Z:\¥Emb¥ISE¥lcd01¥fpga¥main.xise
- 論理合成して, FPGAに書き込み.



液晶モジュールのサンプルプロジェクト lcd01



- minilcd_con の利用方法
 - VRAM_ADDR に 14ビットのアドレスを指定
 - VRAM_DATA に 3ビットの色データを指定
ただし、4ビット幅で接続
 - VRAM_WE を 1 にすることで、そのアドレスに指定色を書き込む。
- 色は Red, Green, Blue それぞれ1ビットの3ビットで表現。8色を表示可能
 - 例えば、3'b111 は白色、3'b100 は赤色、3'b000 は黒色
 - Verilog HDLの擬似コード

```
wire red, green, blue;  
wire [2:0] color;  
assign color = {reg, green, blue};
```
- アドレスは14ビットで表現
 - 128 x 128ピクセル
 - 液晶の左上を (0, 0), 右下を (127, 127) として、(x, y) のピクセルのアドレス ADDR は、以下で定義
 - $ADDR = y * 128 + x$
 - Verilog HDL の擬似コード

```
wire [6:0] x, y;  
wire [13:0] addr;  
assign addr = {y, x};
```

```
/* MieruEmb Top Module
*****
module MieruEMB(CLK, SW, ULED, LCD_CS0, LCD_CD, LCD_WR, LCD_RSTB, LCD_D);
  input      CLK;
  input [2:0] SW;
  output [3:0] ULED;
  output      LCD_CS0, LCD_CD, LCD_WR, LCD_RSTB;
  output [7:0] LCD_D;

  assign ULED = 0;
  wire FCLK, RST_X, LOCKED;

  clockgen clkgen(CLK, FCLK, LOCKED);
  resetgen rstgen(FCLK, ((SW[0] | SW[1] | SW[2]) & LOCKED), RST_X);

  reg [14:0] cnt;
  always @(posedge FCLK or negedge RST_X) begin
    if (!RST_X) cnt <= 0;
    else cnt <= cnt + 1;
  end

  reg [13:0] adr;
  always @(posedge cnt[14] or negedge RST_X) begin
    if (!RST_X) adr <= 0;
    else adr <= adr + 1;
  end

  wire [2:0] color = adr[12:10];
  minilcd_con lcdcon(.CLK(FCLK), .RST_X(RST_X),
                    .VRAM_ADDR(adr), .VRAM_DATA({1'b0, color}), .VRAM_WE(1),
                    .LCD_CS0(LCD_CS0), .LCD_CD(LCD_CD),
                    .LCD_RSTB(LCD_RSTB), .LCD_D(LCD_D), .LCD_WR(LCD_WR));
endmodule
```

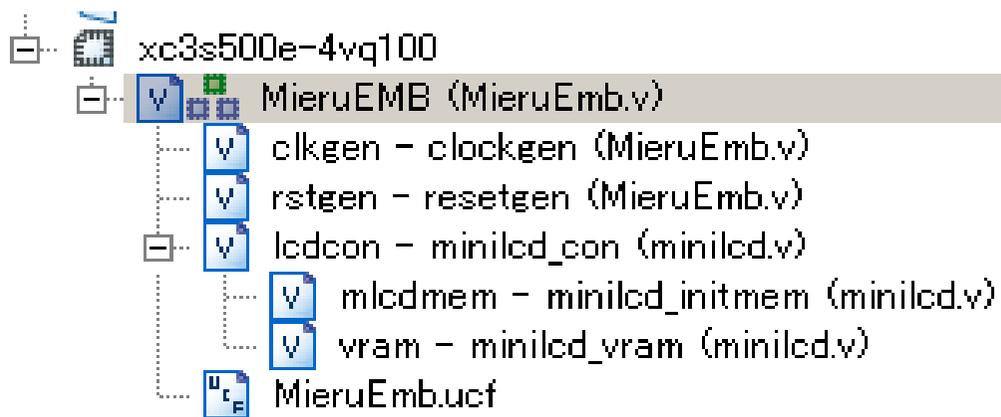
clockgen は、40MHz のクロックから、30MHz のクロック FCLK を生成。

cnt[14] をクロックとして利用している点に注意。



液晶モジュールのサンプルプロジェクト lcd02

- File → Open Project
 - Z:\¥Emb¥ISE¥lcd02¥fpga¥main.xise
- 論理合成して, FPGAに書き込み.



液晶モジュールのサンプルプロジェクト lcd02



- C言語の擬似コード

```
int x = 0;
int y = 33;
int color = 7;

while(1) {
    x++;
    draw_dot(x, y, color);
}
```

```
module MieruEMB(CLK, SW, ULED, LCD_CS0, LCD_CD, LCD_WR, LCD_RSTB, LCD_D);
    input        CLK;
    input  [2:0]  SW;
    output  [3:0] ULED;
    output        LCD_CS0, LCD_CD, LCD_WR, LCD_RSTB;
    output  [7:0] LCD_D;

    assign ULED = 0;
    wire FCLK, RST_X, LOCKED;

    clockgen clkgen(CLK, FCLK, LOCKED);
    resetgen rstgen(FCLK, ((SW[0] | SW[1] | SW[2]) & LOCKED), RST_X);

    /*****
    reg [22:0] cnt;
    always @(posedge FCLK or negedge RST_X) begin
        if (!RST_X) cnt <= 0;
        else        cnt <= cnt + 1;
    end

    reg [6:0] x; // x location
    always @(posedge cnt[22] or negedge RST_X) begin
        if (!RST_X) begin
            x <= 0;
        end else begin
            x <= x + 1;
        end
    end

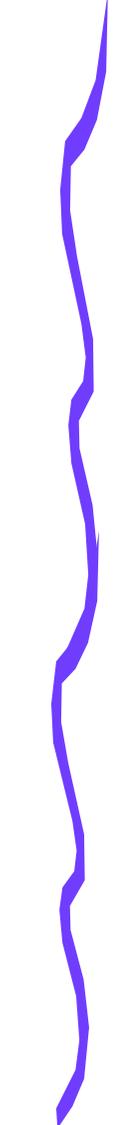
    wire [2:0] color = 3'b111;
    wire [6:0] y = 33;

    miniled_con lcdcon(.CLK(FCLK), .RST_X(RST_X),
        .VRAM_ADDR({y, x}), .VRAM_DATA({1'b0, color}), .VRAM_WE(1),
        .LCD_CS0(LCD_CS0), .LCD_CD(LCD_CD),
        .LCD_RSTB(LCD_RSTB), .LCD_D(LCD_D), .LCD_WR(LCD_WR));

endmodule
```

resetgen は、リセット信号 RST_X を生成。
3個のスイッチが押されるとリセットとしている。

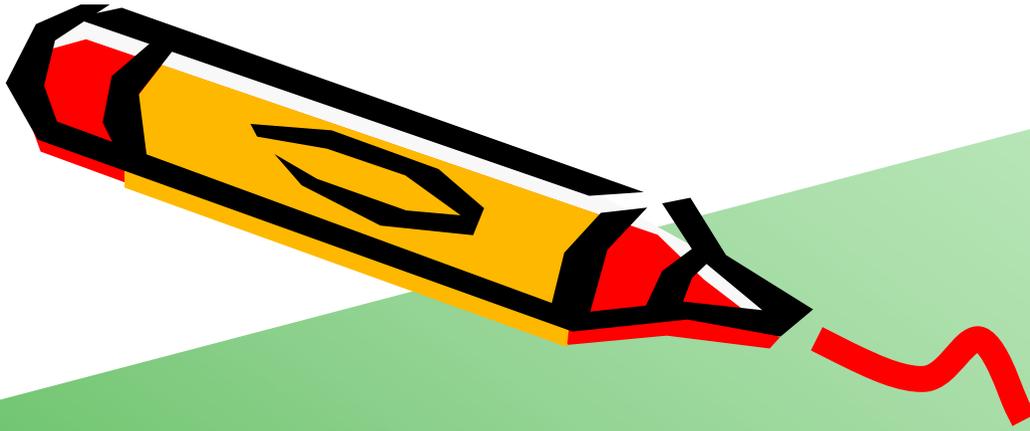
cnt[22] をクロックとして
利用している点に注意。



ナイトライダー回路

- File → Open Project
 - Z:\¥Emb¥ISE¥lcd03¥fpga¥main.xise
- lcd03 は, 先のlcd02と同じ内容. これをベースに編集する.
- MieruEMB.v を編集して, 画面上に, ドットが左右に反射しながら移動する回路を作成する.
- ナイトライダー回路の動作確認 (CP8)





FPGAへのプロセッサの実装



MieruEMB System V1.1

- **FPGA XC3S500E-4VQG100C**
 - プロセッサコア
 - MIPS32準拠(浮動小数点演算なし, キャッシュなし)
 - マルチサイクル, MieruPC-2010のプロセッサコアをベース
 - 35MHz クロック
 - SRAMコントローラ
 - I/Oコントローラ
 - Mini-LCDコントローラ
- **Mini-LCD ZY-FGD1442701V1**
 - 128 x 128 pixel, 8色カラー, Video RAM (8KB)
- **SRAM CY7C1049DV33-10ZSXI**
 - 512KB (152 x 8)
- **MieruEMB System Board V1.1**
- **3個のスイッチの長押しでリセット**
- **LED**
 - D4は一定間隔で点滅, D3はスイッチのどれかが押された時に点灯, D2はSDの読み込みが完了するまで点灯



MieruEMB System V1.1

- File → Open Project
 - Z:¥Emb¥ISE¥emb01¥fpga¥fpga.xise (test64)
- 論理合成して, PROMに書き込み.
- TAから **SDカード** と **SDカードアダプタ** を受け取る.
 - MieruEMBシステムにSDカードを挿入して起動 (CP9)

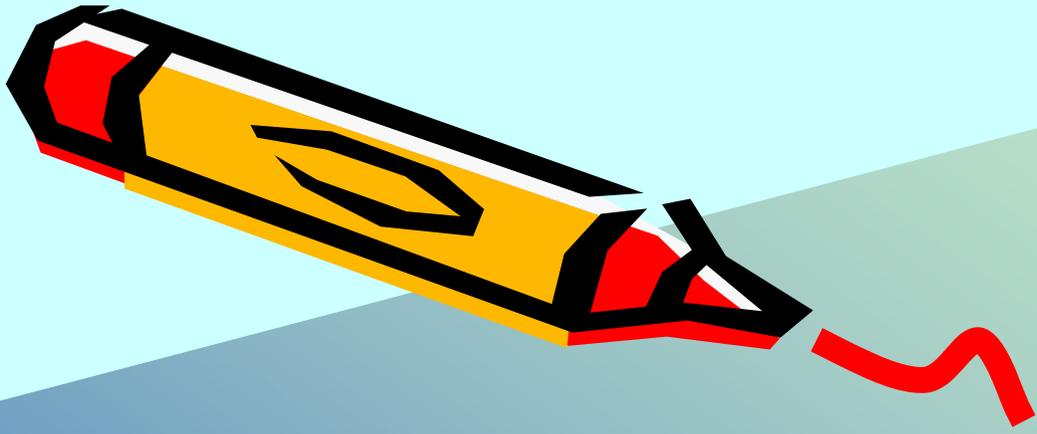


SDカードとアダプタ

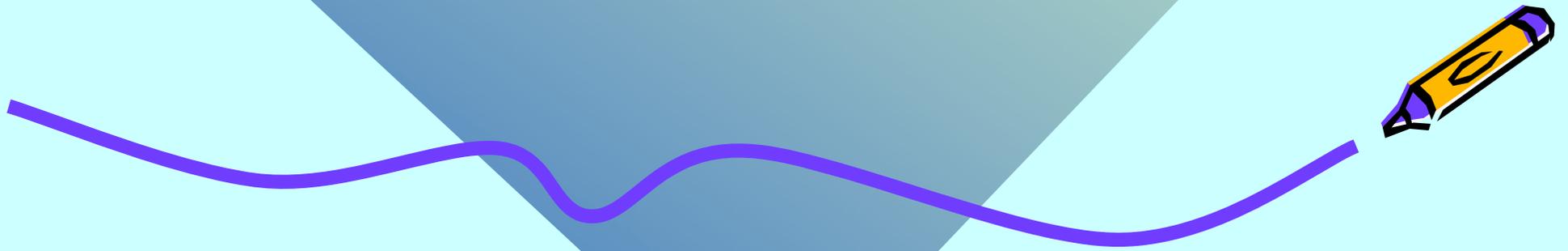


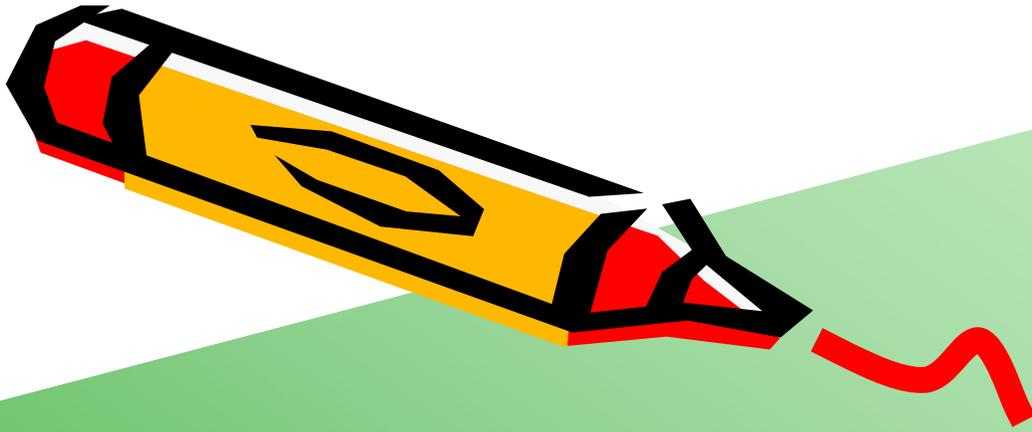
125_space
Z:¥Emb¥Bitfile¥125space.bit





実験 6日目～9日目



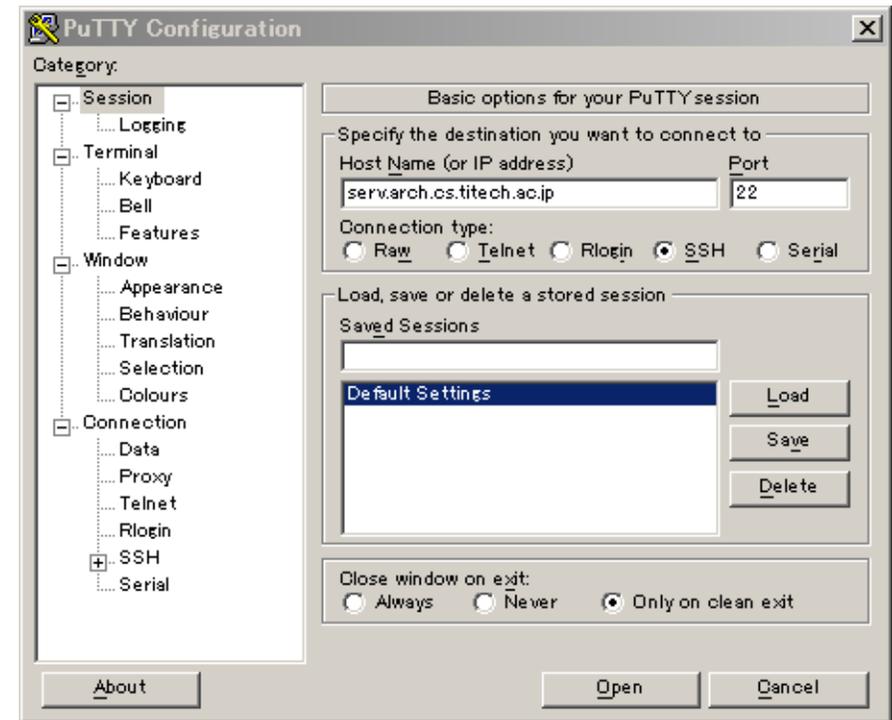


実験用サーバ計算機 (MIPSクロス開発環境)の使い方



実験用サーバ計算機(MIPSクロス開発環境)

- サーバ計算機 `serv.arch.cs.titech.ac.jp`
 - アカウント名, パスワードをTAから受け取ってください。
- MIPSクロス開発環境
 - 構築方法はこちら <http://www.arch.cs.titech.ac.jp/mcore/buildroot.html>
 - `/home/share/cad/mipsel-emb/usr/bin/`
 - `mipsel-linux-gcc`
 - `mipsel-linux-as`
 - `mipsel-linux-ld`
 - `mipsel-linux-objdump` など
- サーバにログイン `putty` を起動
 - `Z:%Emb%Exe%putty%putty.exe`
- ファイル転送には `WinSCP` を使う
 - `Z:%Emb%Exe%winscp%WinSCP.exe`



SDカードの使い方と初期化方法



- MieruEMBシステム (MieruPC-2010)



- SRAMを512KBのメインメモリとして利用.
- 8MBのSDカードを利用.
- 電源投入時およびリセット時に, SDカードから512KBのデータをメインメモリにコピーして, 実行を開始する.
- SDカードの**ブロックサイズを512B**として, **81~1104番目のブロック**をメインメモリにコピー.
- SDカードからはSPI(Serial Peripheral Interface)モードにて読み出し.

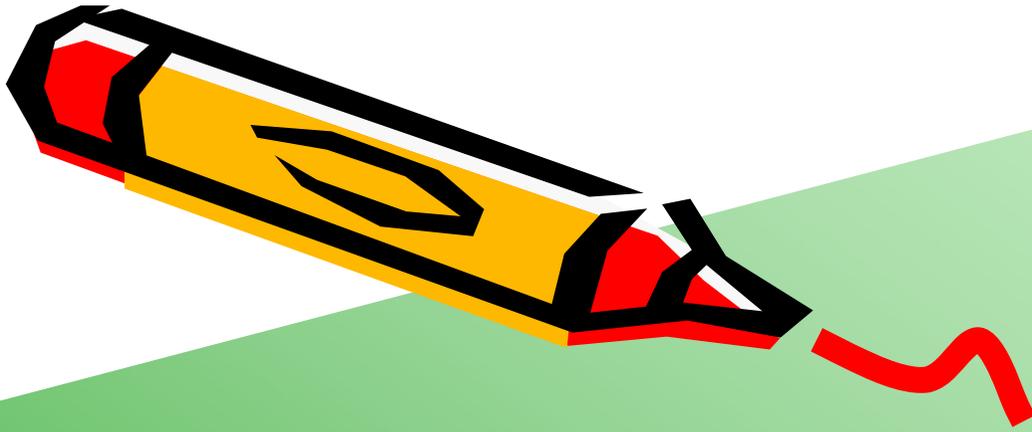
- Windows 7 / XP

- SDカードをフォーマットして, ファイルをコピーすると, 経験上, 最初のファイルのデータは 81番目のブロックから格納されることが知られている.
- ただし, まれに, そうでない場合がある. この場合には, 次の方法で, SDカードを初期化する(すべてのブロックに適切なデータを書き込む)必要がある.

- WindowsにおけるSDカードの初期化(全ブロックへの書き込み)

- SDカードが Dドライブ とすると, コマンドプロンプトで以下のコマンドを使う. **ドライブ名を間違えるとシステムが破壊されるので, 細心の注意を.**
- `cd Z:¥Emb¥Exe¥dd` (<http://www.chrysocome.net/dd> からダウンロードしたもの)
- `$ dd if=master.dat of=¥¥.¥d:`
- その後, Windows で通常通りフォーマットする.





アセンブラによる 組み込みアプリケーション開発



MIPS I (MIPS R3000)

Instruction Set Architecture (ISA)

■ Instruction Categories

- Computational
- Load / Store
- Jump and Branch
- Floating Point
 - coprocessor
- Memory Management
- Special

Registers

R0 - R31

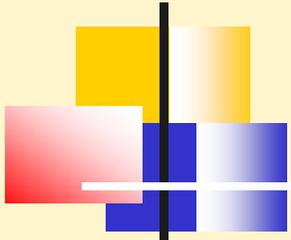
PC

HI

LO

3 Instruction Formats: **all 32 bits wide**

OP	rs	rt	rd	sa	funct	R format
OP	rs	rt	immediate			I format
OP	jump target					J format



MIPS Arithmetic Instructions

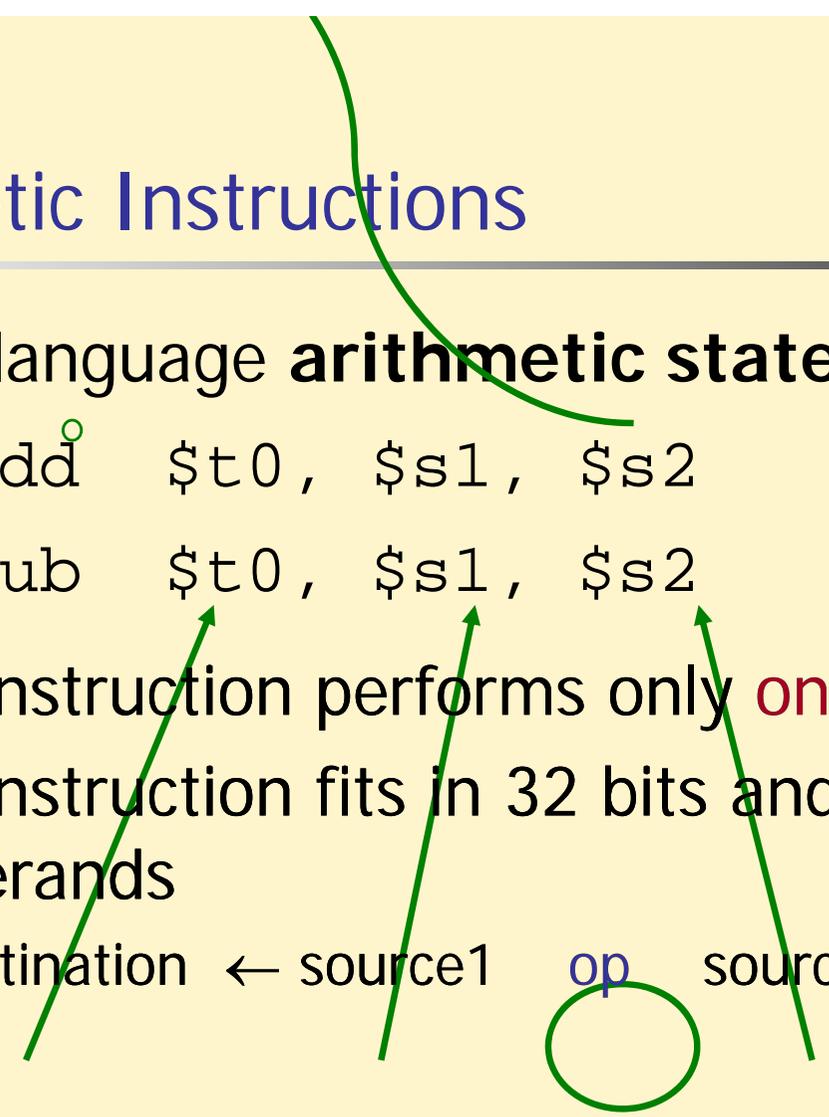
- MIPS assembly language **arithmetic statement**

add ^o \$t0, \$s1, \$s2

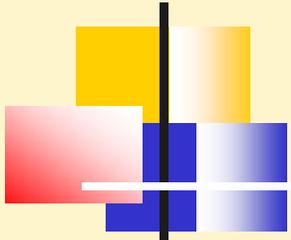
sub \$t0, \$s1, \$s2

- Each arithmetic instruction performs only **one** operation
- Each arithmetic instruction fits in 32 bits and specifies exactly **three** operands

destination ← source1 **op** source2



- Operand order is fixed (destination first)
- Those operands are contained in the **register file** (\$t0, \$s1, \$s2) – **indicated by \$**



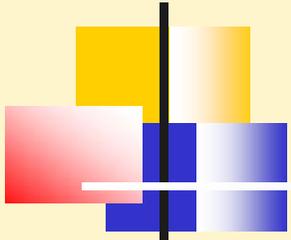
MIPS Memory Access Instructions

- MIPS has two basic **data transfer** instructions for accessing memory

```
lw    $t0, 4($s3)    # load word from memory
```

```
sw    $t0, 8($s3)    # store word to memory
```

- The data is loaded into (lw) or stored from (sw) a register in the register file
- The memory address – a 32 bit address – is formed by adding the contents of the **base address register** to the **offset** value
 - A 16-bit field is limited to memory locations within a region of $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of the address in the base register



MIPS Control Flow Instructions

- MIPS **conditional branch** instructions:

```
bne $s0, $s1, Lbl #go to Lbl if $s0≠$s1
```

```
beq $s0, $s1, Lbl #go to Lbl if $s0=$s1
```

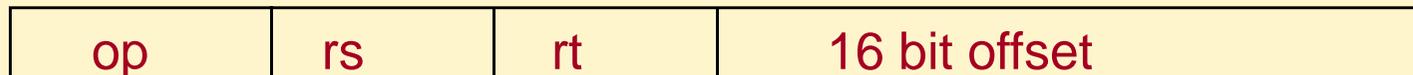
- Ex: **if (i==j) h = i + j;**

```
bne $s0, $s1, Lbl1
```

```
add $s3, $s0, $s1
```

```
Lbl1:    ...
```

- Instruction Format (**I** format):



- How is the branch destination address specified?

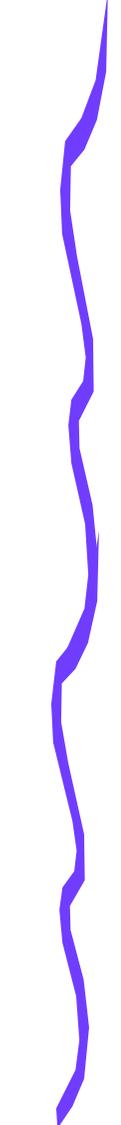
メモリマップドI/O

- MieruEmb System では, **メモリマップドI/O** (Memory Mapped I/O) を採用
 - メモリのリード, ライトのための命令(load, store)を入出力機器にも利用
 - 0x00000 ~ 0x7ffff の512KB は SRAMの物理メモリに割り当て
 - 0x80000 ~ 0x7fffffff は未定義
 - 0x800000 以降は, **メモリマップドI/O**に割り当て
 - 0x80010c : 1KHz タイマ
 - 0x8001f0 : GPIO[0] 汎用入出力
 - 0x8001f1 : GPIO[1] 汎用入出力
 - 0x8001fc : SW[0] スイッチ
 - 0x8001fd : SW[1] スイッチ
 - 0x8001fe : SW[2] スイッチ
 - 0x8001ff : GPIN 汎用入力
 - 0x900000 ~ 0x903fff : **LCD用ビデオメモリ 16KB (1byte/1pixel)**
 - 液晶に描くためには, この領域にストアすればよい.
 - 液晶の左上を (0, 0), 右下を (127, 127) として, (x, y) のピクセルのアドレス ADDR は, 次式で定義
 - $ADDR = 0x900000 + y*128 + x$
 - バイト単位で書き込み, ただし色は3ビットで指定するため下位3ビットのみが有効となる.



MIPS命令セットアーキテクチャ

- MIPS Reference Card を印刷
 - Z:¥Emb¥Doc¥mipsref.pdf
- アセンブリ言語による開発の準備
 - Z:¥Emb¥SDK/asm の内容をサーバ計算機の /home/username/Emb/SDK/asm にコピー
- Putty にてサーバ計算機にログイン
 - cd ~/Emb/SDK/asm/001_dot/
 - emacs, vi などでファイルを編集, コンパイル
(Windows 上で編集して, サーバ計算機にコピーしてもOK)



001_dot: ドットを描くプログラム

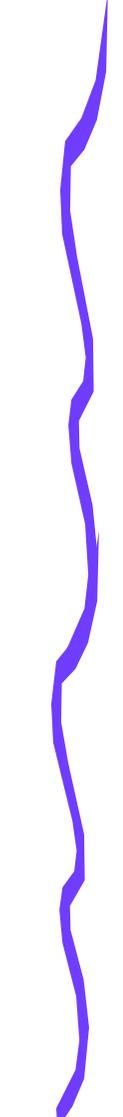


- サーバ計算機にて
 - cd ~/Emb/asm/001_dot/
 - make コマンドにて, init.bin が生成される.
- WinSCP で init.bin をSDカードにコピー
- MieruEMBにて起動
- サーバ計算機にて
 - make dump で, オブジェクトダンプ
 - make clean で生成したファイルを削除
 - make image はMIPSの実行ファイルをMieruEMBのSDに書き込むメモリイメージを生成する.
 - make read は ELF の情報を読む.

memgen は独自開発のプログラム, コードは以下
Z:¥Emb¥SDK¥etc¥memgen¥

Makefile

```
#####  
# MieruEMB System V1.0 2011-10-01 Arch Lab, TOKYO TECH  
#####  
  
TARGET = init  
OBJS = startup.o main.o  
CMDPREFIX = /home/share/cad/mipsel/usr/bin/  
  
MIPSCC = $(CMDPREFIX)mipsel-linux-gcc  
MIPSAS = $(CMDPREFIX)mipsel-linux-as  
MIPSLD = $(CMDPREFIX)mipsel-linux-ld  
OBJDUMP = $(CMDPREFIX)mipsel-linux-objdump  
MEMGEN = memgen  
  
AFLAGS =  
LFLAGS = -static  
  
.SUFFIXES:  
.SUFFIXES: .o .S  
#####  
all:  
    $(MAKE) $(TARGET)  
    $(MAKE) image  
  
$(TARGET): $(OBJS)  
    $(MIPSLD) $(LFLAGS) -T stdld.script $(OBJS) -o $(TARGET)  
  
.S.o:  
    $(MIPSAS) $(AFLAGG) $(@D)/$(<F) -o $(@D)/$(@F)  
  
image:  
    $(MEMGEN) -b $(TARGET) 512 > $(TARGET).bin  
  
dump:  
    $(OBJDUMP) -S $(TARGET)  
  
read:  
    readelf -a $(TARGET)  
  
clean:  
    rm -f *.o *~ log.txt $(TARGET) $(TARGET).bin  
#####
```



001_dot: ドットを描くプログラム

~/Emb/SDK/asm/001_dot/main.S

```

kterm
#####
# Sample Program for MieruEMB System v1.0
#####
        .text
        .align 2
        .globl main
        .ent main

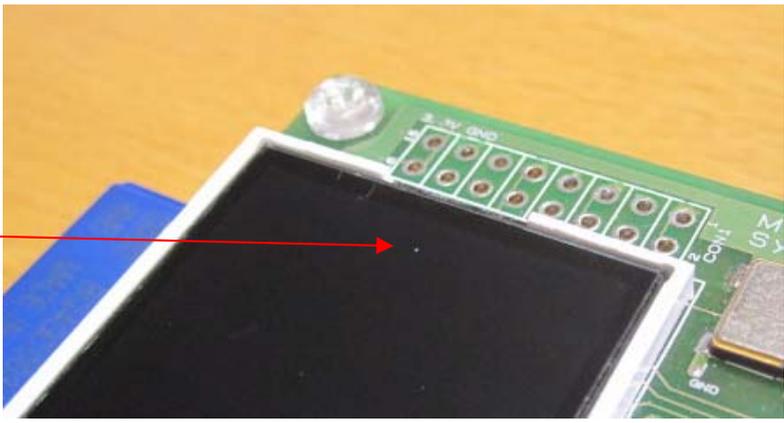
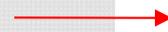
main:
        .set noneorder

        li $3, 0x900000 # $3 = vram address
        li $2, 7 # $2 = 7 (white)
        sw $2, 4288($3) # vram[4288] = 7;
                        # (33 * 128 + 64) = 4288

$L1:
        j $L1 # while(1);
        nop #

        .end main
    
```

メモリへのストアにより
ドットを描く
swではなくsbを使っ
てください。



main.S

ドット



```

kterm
#####
# Sample Program for MieruEMB System v1.0
#####
        .text
        .globl _start
        .ent _start

_start:
        .set noneorder
        .set noat

        nop
        move $1, $0
        move $2, $0
        move $3, $0
        move $4, $0
        move $5, $0
        move $6, $0
        move $7, $0
        move $8, $0
        move $9, $0
        move $10, $0
        move $11, $0
        move $12, $0
        move $13, $0
        move $14, $0
        move $15, $0
        move $16, $0
        move $17, $0
        move $18, $0
        move $19, $0
        move $20, $0
        move $21, $0
        move $22, $0
        move $23, $0
        move $24, $0
        move $25, $0
        move $26, $0
        move $27, $0
        move $28, $0
        move $29, $0
        move $30, $0
        move $31, $0
        li $sp, 0x7ff00 # stack pointer 512KB
        j main # jump to the main
        nop

        .end _start
    
```

すべてのレジスタを
値0で初期化する。

スタックポインタの値
を適切に設定し、ジャンプ

stack pointer 512KB
jump to the main

start.S



001_dot: ドットを描くプログラム

● kterm

ENTRY(_start)

SECTIONS

```
{
.startup 0x0000 : { startup.o(.text) }
. = 0x0200;

.init          : { KEEP *(.init) } = 0
.plt          : { *(.plt) }
.text         : { *(.text .stub .text.* .gnu.linkonce.t.*)
                KEEP *(.text) } = 0
.fini         : { KEEP *(.fini) } = 0
.rodata       : { *(.rodata .rodata.* .gnu.linkonce.r.*) }
.tdata        : { *(.tdata .tdata.* .gnu.linkonce.td.*) }
.tbss         : { *(.tbss .tbss.* .gnu.linkonce.tb.*) *(.tcommon) }
.ctors        : { start_ctors = .;
                KEEP *(SORT(.ctors.*))
                KEEP *(.ctors)
                end_ctors = .; }
.dtors        : { start_dtors = .;
                KEEP *(SORT(.dtors.*))
                KEEP *(.dtors)
                end_dtors = .; }
.data         : { *(.data .data.* .gnu.linkonce.d.*)
                SORT(CONSTRUCTORS) }
.got.plt     : { *(.got.plt) }
. = .;
_gp = ALIGN(16) + 0x7ff0;
.got         : { *(.got) }
.bss         : { *(.dynbss)
                *(.bss .bss.* .gnu.linkonce.b.*)
                *(COMMON) }
}
```

stdld.script

リンカスクリプトとディスアセンブル出力

make dump
コマンドによりディスアセンブル

init: file format elf32-tradlittlemips

Disassembly of section .startup:

```
00000000 <_start>:
0: 00000000      nop
4: 00000821      move    at,zero
8: 00001021      move    v0,zero
c: 00001821      move    v1,zero
10: 00002021     move    a0,zero
14: 00002821     move    a1,zero
18: 00003021     move    a2,zero
1c: 00003821     move    a3,zero
20: 00004021     move    t0,zero
24: 00004821     move    t1,zero
28: 00005021     move    t2,zero
2c: 00005821     move    t3,zero
30: 00006021     move    t4,zero
34: 00006821     move    t5,zero
38: 00007021     move    t6,zero
3c: 00007821     move    t7,zero
40: 00008021     move    s0,zero
44: 00008821     move    s1,zero
48: 00009021     move    s2,zero
4c: 00009821     move    s3,zero
50: 0000a021     move    s4,zero
54: 0000a821     move    s5,zero
58: 0000b021     move    s6,zero
5c: 0000b821     move    s7,zero
60: 0000c021     move    t8,zero
64: 0000c821     move    t9,zero
68: 0000d021     move    k0,zero
6c: 0000d821     move    k1,zero
70: 0000e021     move    gp,zero
74: 0000e821     move    sp,zero
78: 0000f021     move    s8,zero
7c: 0000f821     move    ra,zero
80: 3c1d0007     lui    sp,0x7
84: 37bdf00      ori    sp,sp,0xff00
88: 1000005d     b      200 <main>
8c: 00000000      nop
```

Disassembly of section .text:

```
00000200 <main>:
200: 3c030090     lui    v1,0x90
204: 24020007     li     v0,7
208: ac6210c0     sw     v0,4288(v1)
20c: 1000ffff     b      20c <main+0xc>
210: 00000000     nop
...
```



002_bar: カラーバーを描くプログラム

~/Emb/SDK/asm/002_bar/main.S

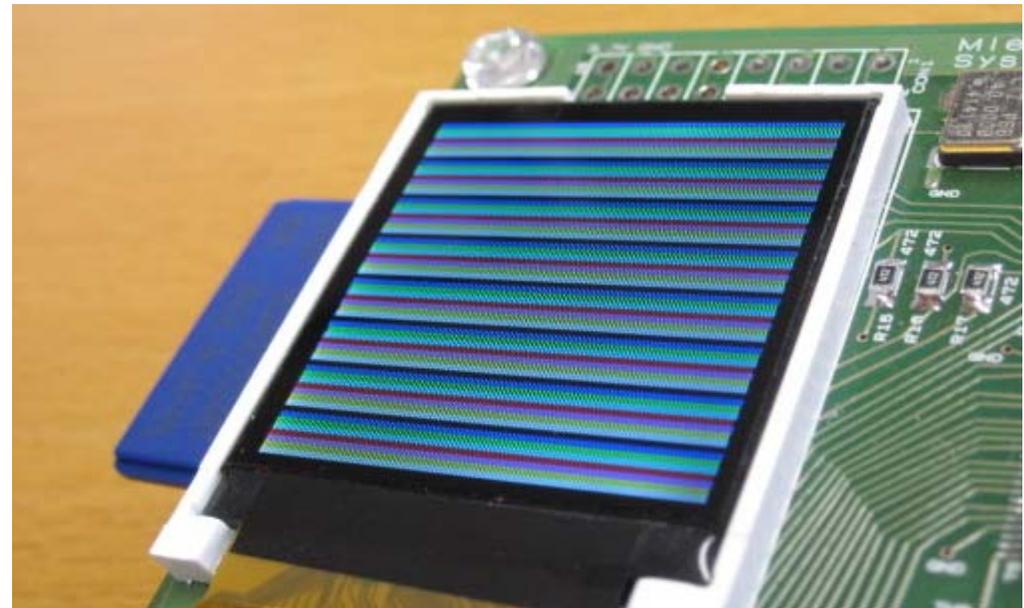
遅延分岐の影響を排除するため、分岐とジャンプの後には nop を挿入すること。

```
#####;
# Sample Program for MieruEMB System v1.0
#####;
    .text
    .align 2
    .globl main
    .ent main

main:
    .set noreorder

    li $3,0x900000    # $3 = vram address
    li $5, 16384      #
    li $6, 0          #
L1:
    srl $7, $6, 10    #
    sw $7,0($3)       #
    addi $3, $3, 1    #
    addi $6, $6, 1    #
    bne $6, $5, L1   #
    nop               #
$L1:
    j $L1             #
    nop               #
    .end main
```

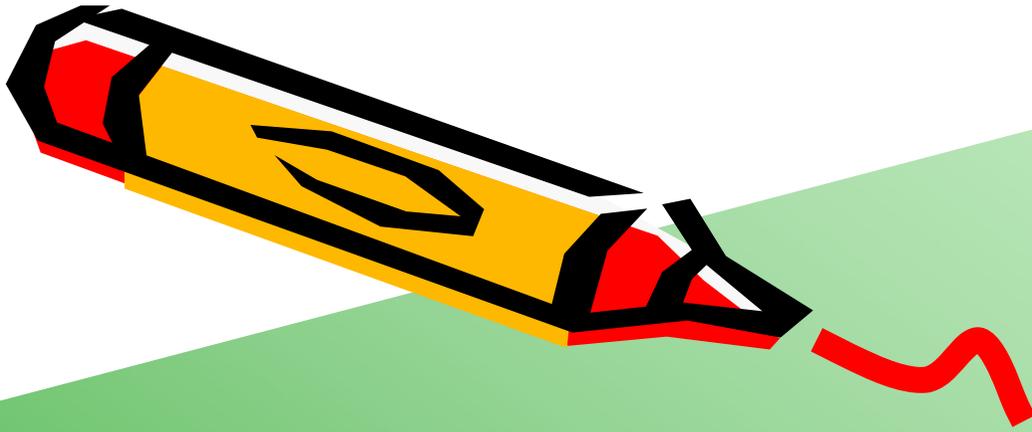
main.S



ナイトライダー(アセンブリ言語版)

- Z:¥Emb¥SDK¥asm¥003_night¥main.S
 - 002_bar と同じ内容. これをベースに編集する.
- アセンブリ言語にて, 画面上に, ドットが左右に反射しながら移動する回路を作成する.
 - 美しくみえる様に修正する.
- アセンブリコードのデバッグ方法については, 以下の資料を参照.
 - Z:¥Emb¥Doc¥ Emb-ASMDebug.ppt
- ナイトライダーの動作確認 (CP10) 





C言語による 組み込みアプリケーション開発



101_dot: ドットを描くプログラム (C言語版)

Z:\¥Emb¥SDK¥app¥101_dot¥main.c

```
● kterm
/**** Sample Program for MieruEMB System v1.0 ****/
/*****
volatile char *e_vram = (char*)0x900000;
volatile int *e_time = (int *)0x80010c;
volatile char *e_gp1 = (char*)0x8001f0;
volatile char *e_gp2 = (char*)0x8001f1;
volatile char *e_sw1 = (char*)0x8001fc;
volatile char *e_sw2 = (char*)0x8001fd;
volatile char *e_sw3 = (char*)0x8001fe;
volatile char *e_gin = (char*)0x8001ff;

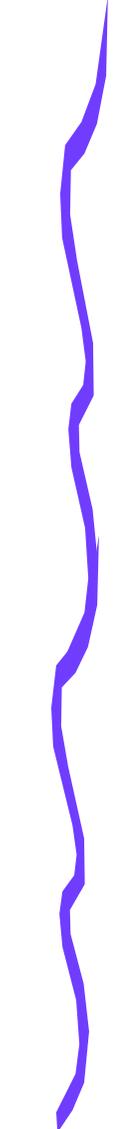
/*****
int main(void) {
    int x = 63;
    int y = 63;

    e_vram[x+y*128] = 7;
    while(1);
}
/*****
```

メモリマップドI/Oの
ための変数の定義

x,y で指定したドットを白色にする.

main.c



102_bar: カラーバーを描くプログラム(C言語版)

Z:\Emb\SDK\app\102_bar\main.c

```
.....  
/**** Sample Program for MieruEMB System v1.0 ****/  
/*****  
volatile char *e_vram = (char*)0x900000;  
volatile int *e_time = (int *)0x80010c;  
volatile char *e_gp1 = (char*)0x8001f0;  
volatile char *e_gp2 = (char*)0x8001f1;  
volatile char *e_sw1 = (char*)0x8001fc;  
volatile char *e_sw2 = (char*)0x8001fd;  
volatile char *e_sw3 = (char*)0x8001fe;  
volatile char *e_gin = (char*)0x8001ff;  
  
/*****  
int main(void){  
    int x, y;  
    for(x=0; x<127; x++)  
        for(y=0; y<127; y++)  
            e_vram[x+y*128] = (y>>3) & 7;  
  
    while(1);  
}  
/*****
```

main.c



103_aba: 文字を表示するプログラム(C言語版)



Z:¥Emb¥SDK¥app¥103_aba¥main.c

```
/******  
static const unsigned char fonts[2][16][8] = {  
  {0,0,0,0,0,0,0,0}, //A  
  {0,0,0,0,0,0,0,0},  
  {0,0,0,0,0,0,0,0},  
  {0,0,0,0,1,0,0,0},  
  {0,0,0,0,1,0,0,0},  
  {0,0,0,1,0,1,0,0},  
  {0,0,0,1,0,1,0,0},  
  {0,0,0,1,0,1,0,0},  
  {0,0,0,1,0,1,0,0},  
  {0,0,0,1,0,1,0,0},  
  {0,0,1,0,0,0,1,0},  
  {0,0,1,1,1,1,1,0},  
  {0,0,1,0,0,0,1,0},  
  {0,0,1,0,0,0,1,0},  
  {0,0,0,0,0,0,0,0},  
  {0,0,0,0,0,0,0,0},  
  {0,0,0,0,0,0,0,0}},  
  
  {0,0,0,0,0,0,0,0}, //B  
  {0,0,0,0,0,0,0,0},  
  {0,0,0,0,0,0,0,0},  
  {0,0,1,1,1,1,0,0},  
  {0,0,1,0,0,0,1,0},  
  {0,0,1,0,0,0,1,0},  
  {0,0,1,0,0,0,1,0},  
  {0,0,1,1,1,1,0,0},  
  {0,0,1,0,0,0,1,0},  
  {0,0,1,0,0,0,1,0},  
  {0,0,1,0,0,0,1,0},  
  {0,0,1,0,0,0,1,0},  
  {0,0,1,1,1,1,0,0},  
  {0,0,0,0,0,0,0,0},  
  {0,0,0,0,0,0,0,0},  
  {0,0,0,0,0,0,0,0}}  
};
```

文字 A のフォントデータ.

文字 B のフォントデータ.

```
/******  
void mylib_putc(int x, int y, char c, int color){  
  int i, j;  
  
  for(i=0; i<16; i++){  
    for(j=0; j<8; j++){  
      if(fonts[(int)(c-'A')][i][j]) e_vram[(x+j)+(y+i)*128] = color;  
    }  
  }  
}  
  
/******  
int main(void){  
  mylib_putc(0, 0, 'A', 7); x,y で指定した場所に色 7 にて, A という文字を表示.  
  mylib_putc(8, 0, 'B', 7);  
  mylib_putc(16, 0, 'A', 2);  
  
  while(1);  
}  
/******
```

main.c



104_pic : 画像表示とスイッチ入力のサンプル(C言語版)

Z:¥Emb¥SDK¥app¥104_pic¥

- SW1 左移動, SW2 上移動, SW3 右移動, SW1 & SW3 下移動



ナイトライダー(C言語版)

- Emb¥SDK¥app¥200_night¥
 - 104_pic と同じプログラム, これをベースに修正.
- C言語にて, 画面上に, ドットが左右に反射しながら移動する回路を作成する.
 - 美しくみえる様に修正する.
- ナイトライダーの動作確認 (CP11) 
- ナイトライダーを, (1)Verilog HDLによるハードウェア実装, (2)アセンブリ言語によるソフトウェア実装, (3)C言語によるソフトウェア実装という3種類で記述した.
それぞれの利点, 欠点を考えてみる.
- app 以下にたくさんのサンプルプログラムがあるのでこれらを動かしてみる.
 - 106_fig, 107_gpio, 125_space など



組み込みシステムの仕様策定

- 実装する組み込みシステムの仕様書(A4で1枚程度)を作成する.
- 仕様書を印刷 (CP12)

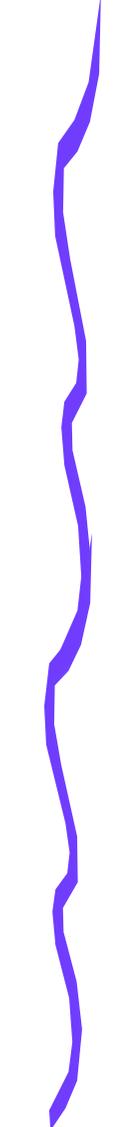
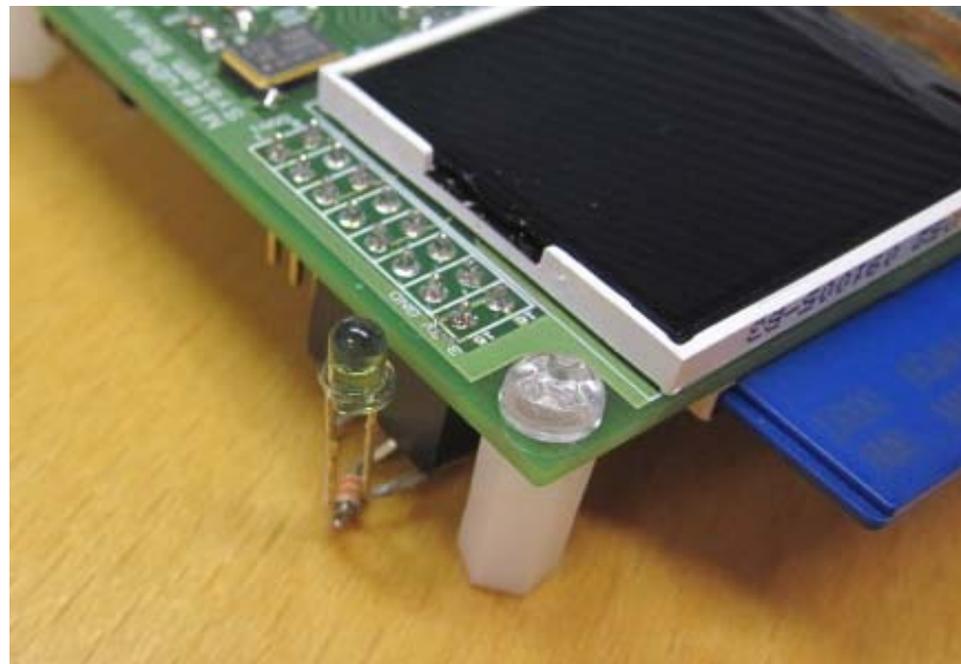
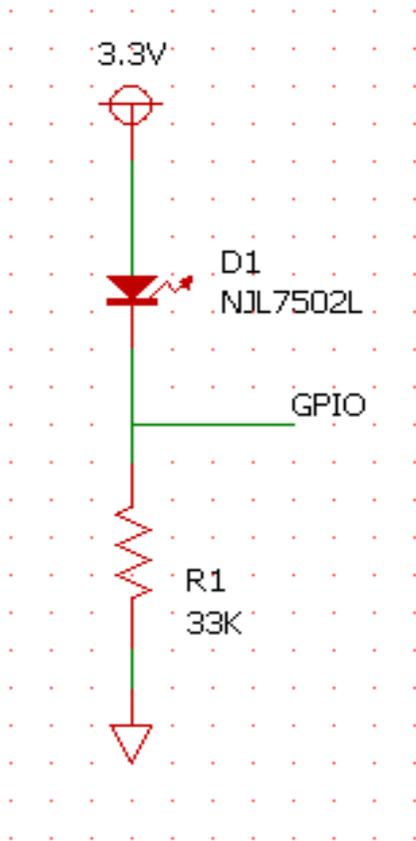
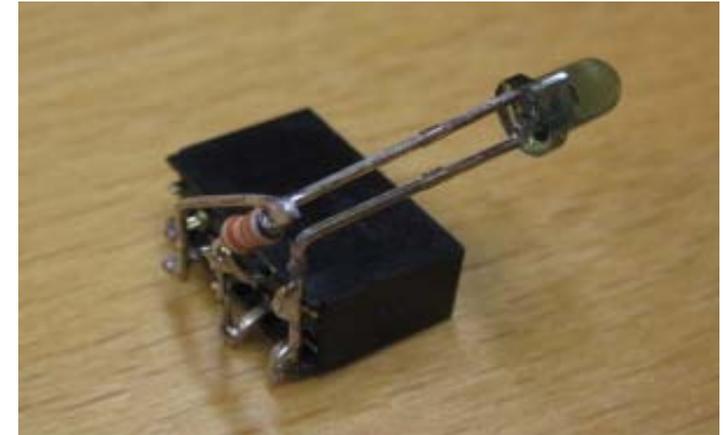


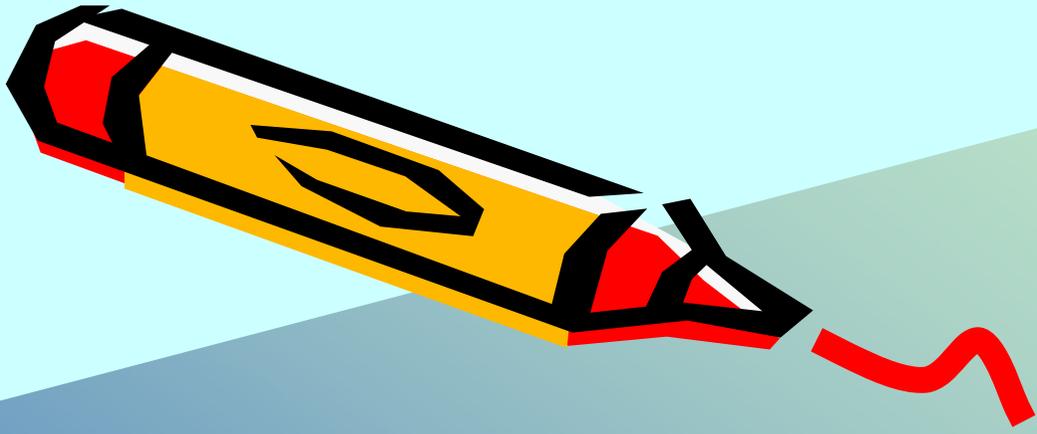
- 目的(用途)
- 実装カ所の例
 - ハードウェア
 - センサーの追加, コントローラの追加, I/Oの追加
 - ロボットなどへの組み込み(ルンバの高性能化)
 - FPGA
 - プロセッサの高速化・高性能化, I/Oの高性能化・高速化
 - 命令キャッシュの実装
 - ソフトウェア
 - アセンブリ言語, C言語
 - ファイルシステム
 - オペレーティングシステム
 - ハードウェア・ソフトウェアの協調
 - 専用命令の追加による高速化



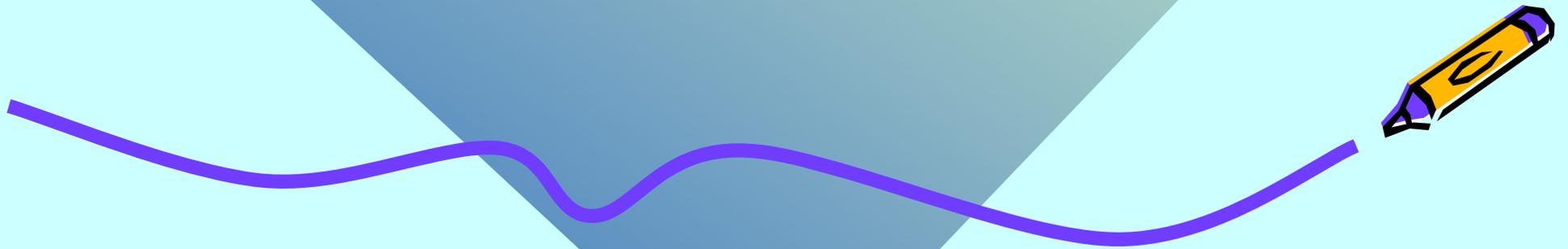
センサの追加：光センサの接続例

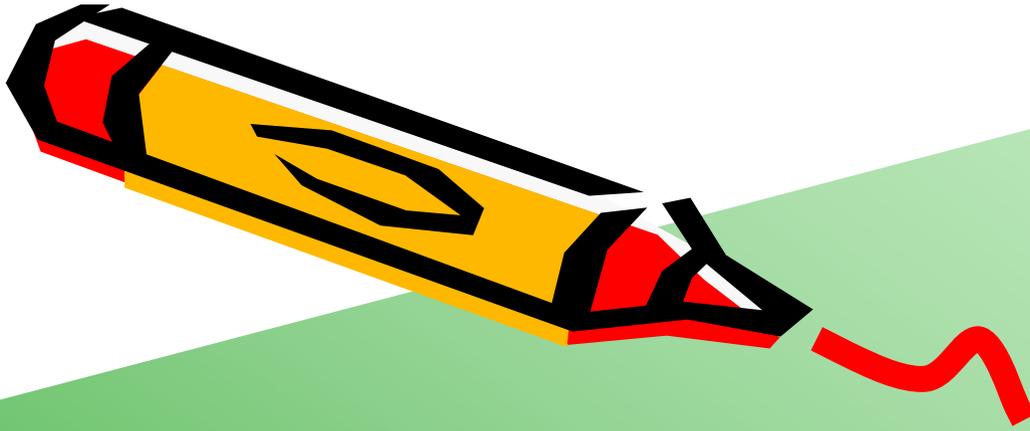
- 光センサ(フォトランジスタ) NJL7502L
 - 明るい時, フォトランジスタに電流が流れる.
 - 暗くなるとGPIOの入力がLOWになる.



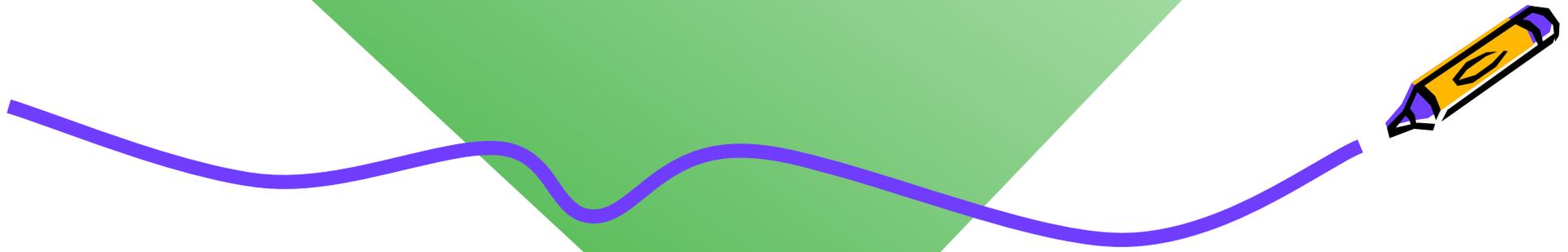


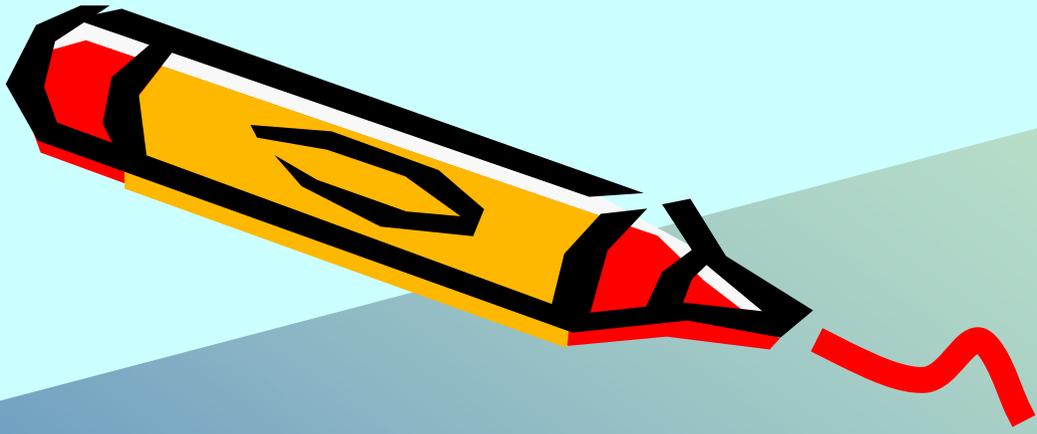
実験 10日目～11日目



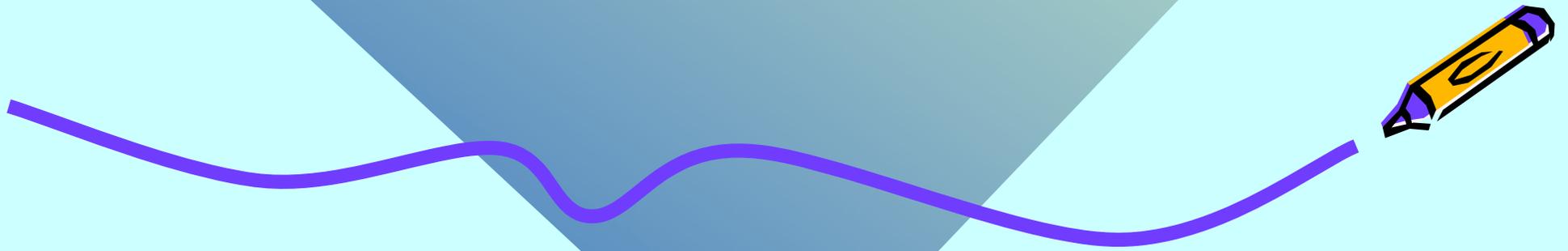


組み込みシステム開発





実験 12日目



組み込みシステムコンテスト

- 1人5分の持ち時間
 - PowerPointのスライドを用いた3分間のプレゼンテーション
 - 組み込みシステム(実機)を用いた2分間のプレゼンテーション
- 参加学生 20名の投票(1人, 2件に投票)により, 優秀なシステムを決定
 - 最優秀賞
 - 優秀賞
- 評価基準
 - アイデア(新規性), 実用性(有用性), 完成度
 - これらを伝えるプレゼンテーション能力も重要
- 教員の評価により, 幾つかの賞を授与



コンテストに利用したソフトウェアやデータは, 原則, フリーウェアとして公開してください。

組み込みシステムコンテストの準備（コンテスト前日17:00までに！）

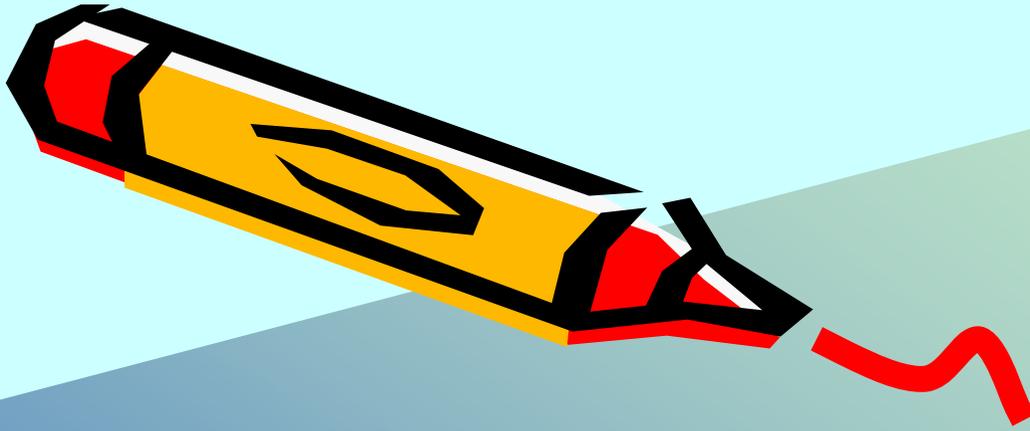


- スライドの作成
 - 3分間のプレゼンテーション用のスライドを作成して、サーバ計算機に次のディレクトリを作成して、ファイル名 **slide.ppt** として保存する。
Emb/Contest/
- 実装する組み込みシステムの仕様書(A4で1枚程度)
 - 先に作成した仕様書を、ファイル名 **siyou.ppt** として(拡張子は利用するアプリケーションによって異なる)、次のディレクトリに保存する。
Emb/Contest/
- 作成したプログラム, 説明など
 - その他, 作成したプログラムやハードウェア記述などを次のディレクトリに保存する。また, その使い方を **README.txt** に記述する。
Emb/Contest/

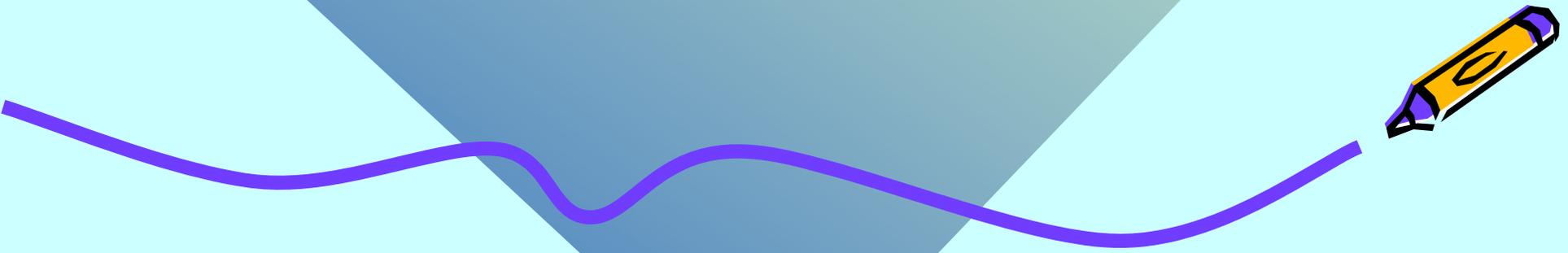


コンテストに利用したソフトウェアやデータは、原則、フリーウェアとして公開してください。





補足



参考資料, 参考URL

- 東工大 情報工学科 情報実験第四 組み込みシステムのホームページ
 - www.arch.cs.titech.ac.jp/lecture/emb/
- プリント基板製造 P板.com
 - <http://www.p-ban.com/>
- 秋月電子通商
 - <http://akizukidenshi.com/>
- 千石電商
 - <http://www.sengoku.co.jp/>
- 東京エレクトロデバイス株式会社
 - <http://www.teldevice.co.jp/>
- ザイリンクス株式会社
 - <http://japan.xilinx.com/>
- Digilent Inc.
 - <http://www.digilentinc.com/>
- Digi-Key Corporation
 - <http://jp.digikey.com/>



参考資料, 参考URL

- Xilinx FPGA XC3S500E-4VQG100C
 - ザイリンクス DS312 Spartan-3E FPGA ファミリ データシート
- SRAM CY7C1049DV33-10ZSXI
 - CY7C1049DV33データシート
- Mini-LCD ZY-FGD1442701V1
 - <http://www.aitendo.co.jp/product/1621>
 - コントローラIC ST7735
- フォトランジスタ NJL7502L
 - <http://akizukidenshi.com/catalog/g/gI-02325/>
- クロックオシレータ 40MHz
 - <http://akizukidenshi.com/catalog/g/gP-03617/>
- SDカード 8MB
 - <http://akizukidenshi.com/catalog/g/gS-02549/>
- スペーサ, ネジ
 - <http://akizukidenshi.com/catalog/g/gP-01861/>
- タクトスイッチ
 - <http://akizukidenshi.com/catalog/g/gP-01282/>



参考資料, 参考URL

- 電池ボックス
 - <http://akizukidenshi.com/catalog/g/gP-00310/>
- チップLED 赤
 - http://www.sengoku.co.jp/mod/sgk_cart/search.php?multi=TLRE1002A&cond8
- チップ抵抗
 - http://www.sengoku.co.jp/mod/sgk_cart/search.php?multi=ERJ6GEYJ
- コンフィギュレーションケーブル
 - <http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,395,523&Prod=JTAG-USB>
- MIPS32のクロス開発環境の構築
 - <http://www.arch.cs.titech.ac.jp/mcore/buildroot.html>
- dd for windows
 - <http://www.chrysocome.net/dd>



Check Point 確認シート

名前		学籍番号	
----	--	------	--



		日付	時刻	TAサイン
CP1	回路図の修正と印刷			
CP2	プリント基板の修正			
CP3	チップ抵抗などの固定			
CP4	チップ抵抗などのはんだ付け			
CP5	FPGAなどの固定			
CP6	FPGAなどのはんだ付け			
CP7	液晶などのはんだ付け			
CP8	FPGA版ナイトライダー回路			
CP9	MieruEMBの動作確認			
CP10	アセンブリ言語版ナイトライダー			
CP11	C言語版ナイトライダー			
CP12	組み込みシステム仕様策定			
	記入例	2013-11-15	10:45	Kise

