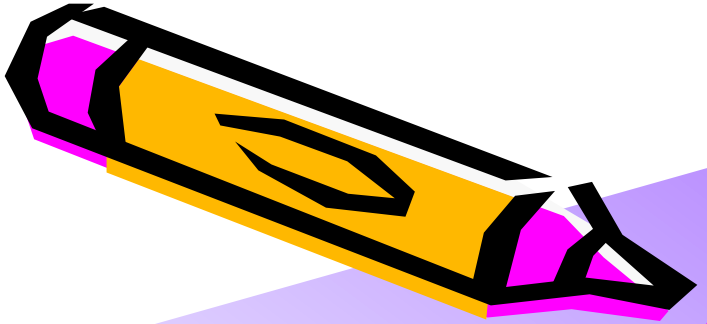
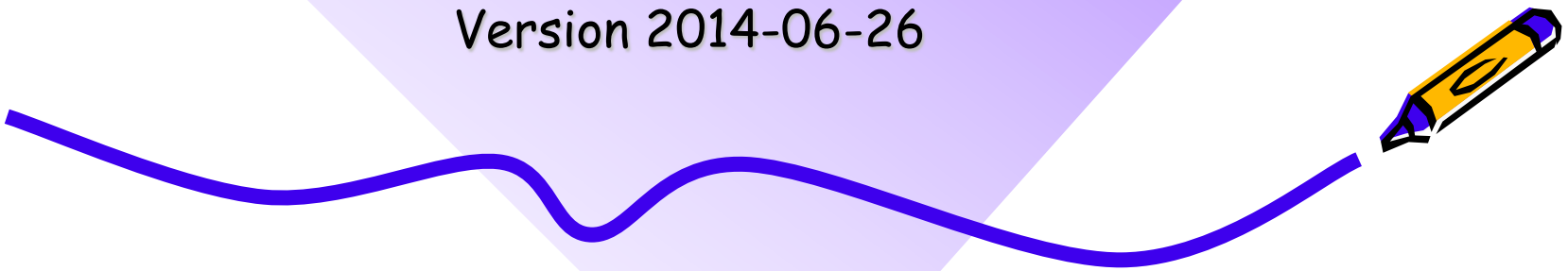


The 2nd ARC/CPSY/RECONF
High-Performance Computer System Design Contest

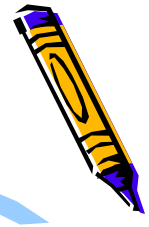
Application Specification of Computer System Design Category

コンテスト実行委員会コアチーム

Version 2014-06-26



このドキュメント



- このドキュメントでは，コンピュータシステム部門のアプリケーションプログラムの仕様を説明します
- また、SDKの使用方法について解説します．

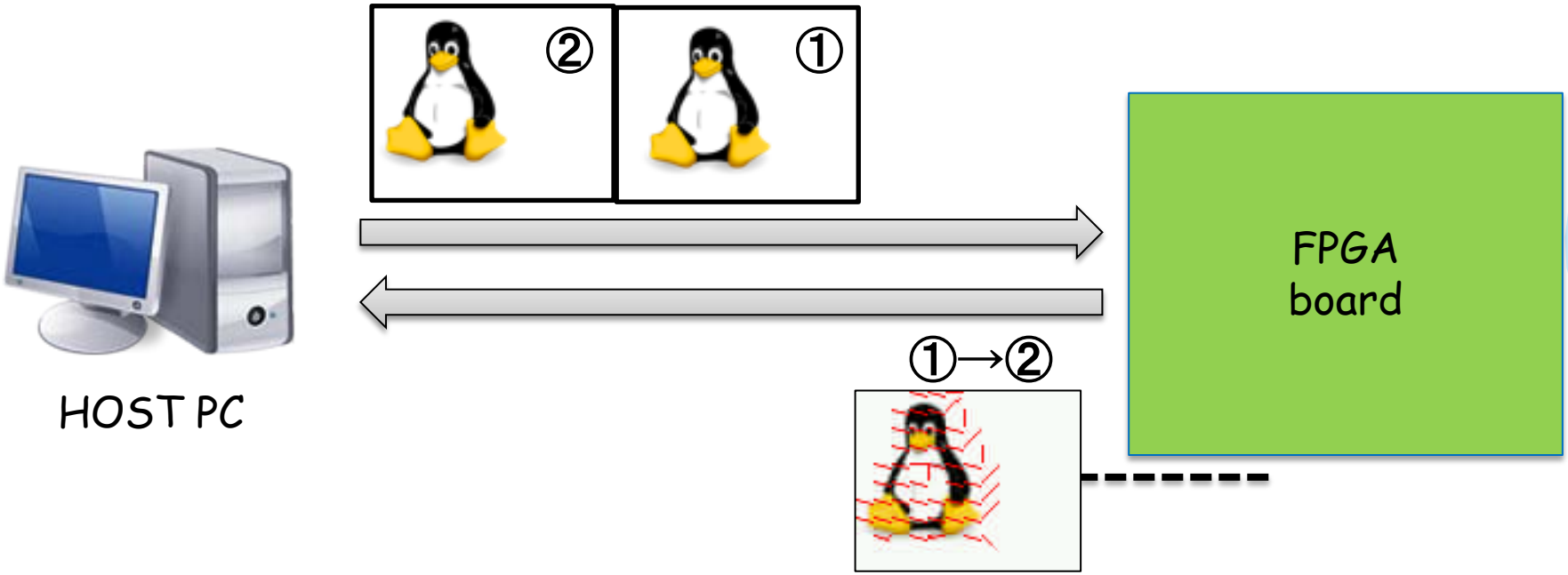
- 設計コンテストのWEBサイト
  - http://aquila.is.utsunomiya-u.ac.jp/contest/

- 不明な点は，以下のいずれかの方法でお問い合わせください．
  - メールアドレス(contest_support@virgo.is.utsunomiya-u.ac.jp)
  - twitter(#arc_procon)
  - 技術情報掲示板
    - Google Group: HpCpsyDC2014
    - https://groups.google.com/forum/?hl=ja#!forum/hpcpsy2014dc

コンピュータシステム設計部門のターゲット「ネットワーク画像オプティカルフロー処理」

- 目的
  - ホストPCから送られる2つの画像フレーム間の動きベクトルを、オプティカルフロー処理により計算し、画像に書き入れてホストPCに送り返す。

入力画像
（YCrCb成分をJPEGに似た形式で圧縮）



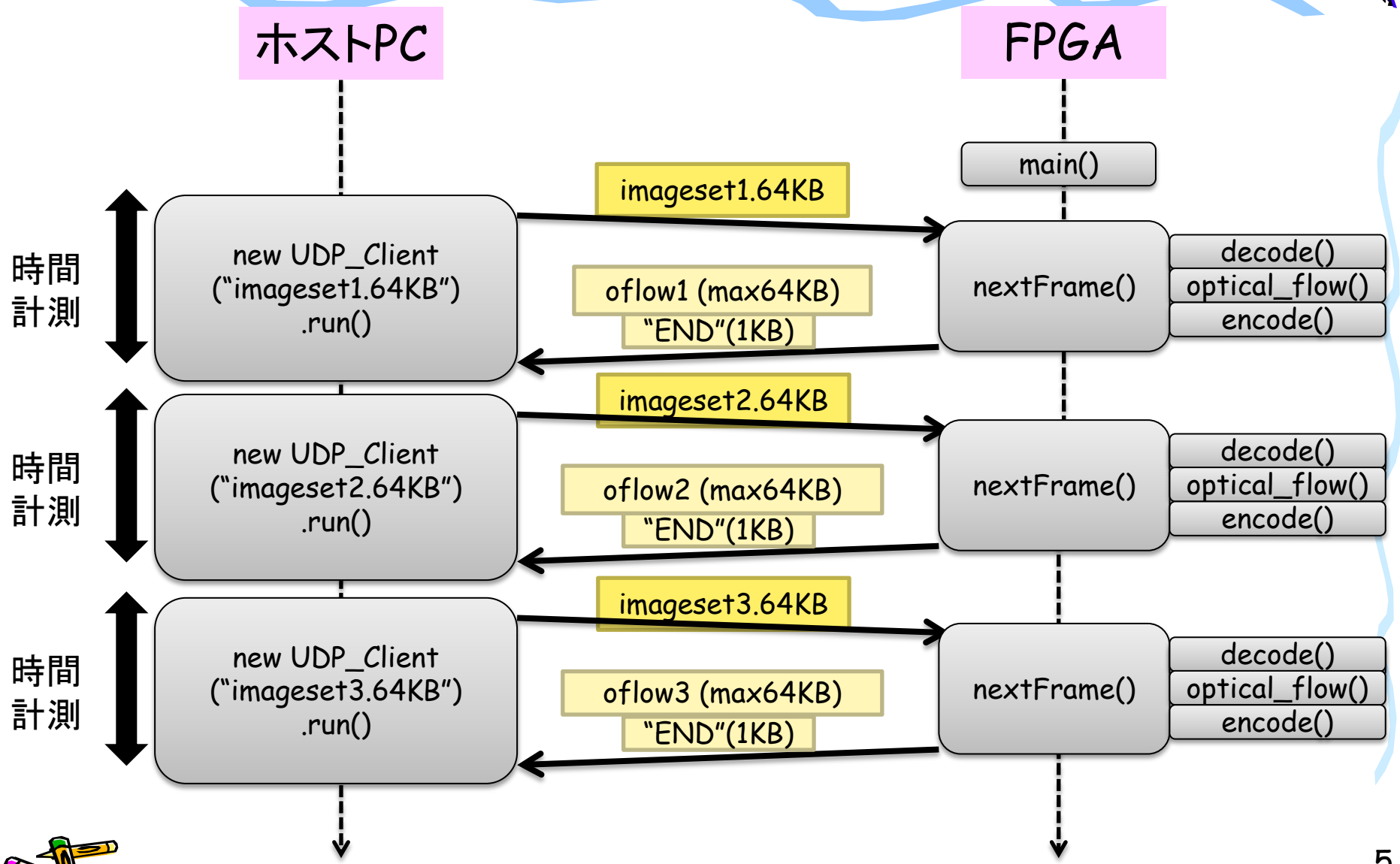
出力画像 （RGB成分を同様の形式で圧縮）

概要

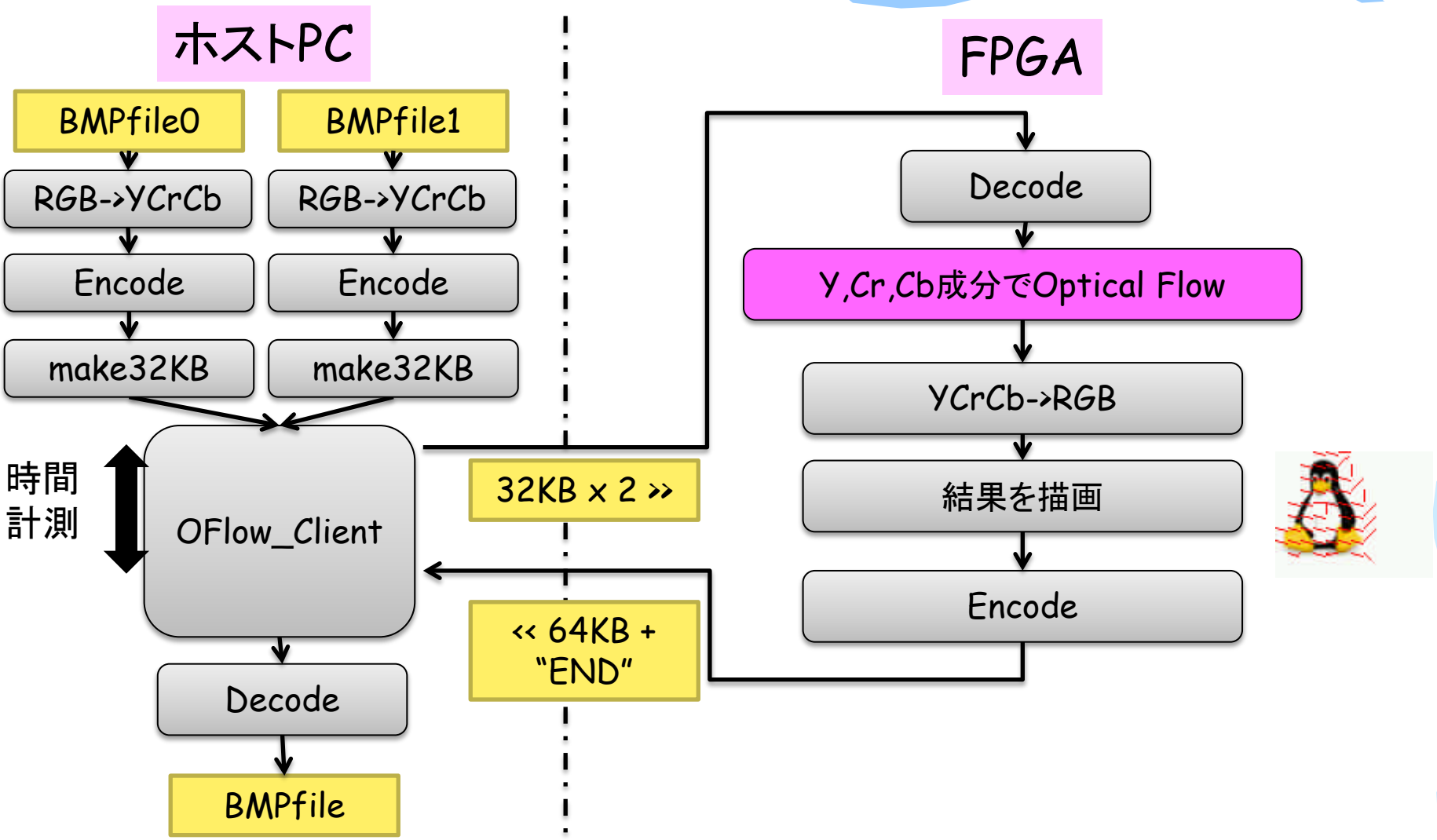
- コンピュータシステム部門では，以下の処理をFPGAボードで行います
  - ホストから2つの入力画像フレーム（32KB x 2）を受け取り、デコードする
    - 画像は独自のJPEGに似た形式で圧縮されたフォーマットで渡されます．
  - 2つのフレーム間のオプティカルフローを計算する
    - オプティカルフローを画像に書き入れる（赤線）
    - 結果画像をエンコードして結果画像フレームにする
  - 結果画像フレーム（max64KB）をホストに送信し、更に"END"（1KB）を送信
    - 注意）UDP・シリアル変換のexStickBridgeは1KB単位でのみパケットを送信

- 参考文献[1]
  - 昌達 慶仁 著，「詳解 画像処理プログラミング」，ソフトバンククリエイティブ株式会社，2008．

FPGA・ホストPC間の通信シーケンス



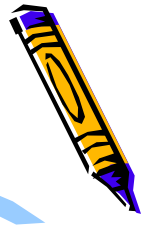
FPGA・ホストPCの処理フロー



ホストPCでの前処理内容

- 2つの入力画像（410_tux0.bmp，410_tux1.bmp ）をそれぞれYCrCb変換し，その変換後の画像をJPEG圧縮します．
  - 読み込み可能なBMP画像フォーマットについては，参考文献[1]を参考にして下さい
  - YCrCb変換に関しては参考文献[1]にある手法をベースに整数化したバージョンを使用しています．（参考文献[1]では浮動小数点を用いています）
  - YCrCb変換後の画像フォーマットとしては，BMP画像フォーマットのR部分にY，G部分にCr，B部分にCbを格納して保存しています
  - JPEG圧縮に関しては，参考文献[1]では1要素（RGBならRのみ，今回の場合Yのみ）に対しての圧縮処理でしたが，YCrCbそれぞれに対して圧縮処理をするように改良しています．
  - JPEG圧縮では，参考文献[1]では浮動小数点のDCTが使われていましたが，整数化したDCTに変更しました．
- 2つの入力画像をそれぞれ32KBのサイズに揃えます（0で埋める）
- 2つをあわせて64KBのデータとし、UDP/IP(8100番ポート)で送信します．
  - 送信したデータは、ToUDP.（画像データ名）.binとして保存されます

FPGAでの処理内容



- FPGAでは以下の処理をします．
  - 32KBの2つのデータを受信
  - 2つのJPEG圧縮画像（img0, img1）をデコード
  - デコードされた画像のそれぞれY, Cr, Cb成分を用いて，オプティカルフローを求めます．
  - 最初の入力画像（img0）に対してYCrCb->RGB変換（関数名はconvert2bmp）をし，RGB画像を得て，そのRGB画像に対してオプティカルフローの線を描画します
  - オプティカルフローが描画された画像をJPEG圧縮します
  - 最大64KBの結果画像データを送信し，最後に“END”を送る

ホストでの後処理内容　と　時間計測・結果検証

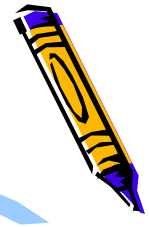
- FPGAからJPEG圧縮された画像をUDP/IP経由で受け取り，ファイルに保存します．ファイル名：FromUDP.(画像データ名).binとして保存します．
- 上記ファイルをデコードし，RESULT.(画像データ名).bmpとして保存します．
- 正しいデータと一致しているか確認します．

- 競技の時間計測について
  - ホストPCにおいて、以下のT2-T1の経過時間を計測します
    - T1: UDPにて64KBの画像データを送信
    - T2: UDPにて"END"を受信
  - その後、BMP画像データが、リファレンスデザインの結果BMP画像データと完全一致しているかを検証します

コンピュータシステム設計部門 リファレンスデザインSDKの 使用方法

- コンテストWEBサイトから、コンピュータシステム設計部門のリファレンスデザインSDK（400_oflow_v10.tgz）をダウンロードして展開。
- 開発環境： LinuxもしくはCygwin
  - http://aquila.is.utsunomiya-u.ac.jp/contest/toolkit.html

SDKの使い方 (1) ディレクトリ構成



- ディレクトリ構成の簡単な説明です.
  - common：共通ソースファイル
  - xilinx：Xilinx用ソースファイル
  - altera：Altera用ソースファイル
  - linux ：Linux用ソースファイル
  - client：ホスト用クライアントプログラムのディレクトリ

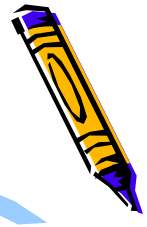
主なソースファイルの構成

ディレクトリ	ファイル名	概要
Platform (linux, xilinx, altera)	addressmap.h addressmap.c	アドレスマップ
	communication.h communication.c	ホストとの通信
	platform.h platform.c	プラットフォーム固有の処理
	bmp.h, bmp.c	ビットマップファイル関連(linuxのみ)
Common	codec.h codec.c	Codecその他画像の変換関連のライブラリ
	fileio.h fileio.c	メモリファイル入出力 （stdioのFILE置き換え）
	image.h, image.c	画像関連ライブラリ
	main.c	メイン
	oflow.h, oflow.c	オプティカルフロー関連処理

ユーティリティツール一覧

ディレクトリ	ファイル名	概要
Linux	decoder.c	デコーダ
	encoder.c	エンコーダ
	make32KBdata.c	32KBデータ作成
	rgb2ycbcr.c	RGB→YCrCb色変換

SDKの使い方 (2) Linuxプログラムの動作確認-1



- FPGAボードが無くても，リファレンスデザインの機能を確認することが出来ます
- トップディレクトリでのmake コマンドにより，FPGAボードでの動作を模擬するLinux上で動作するプログラムを作成することが出来ます．

```
$ make
```

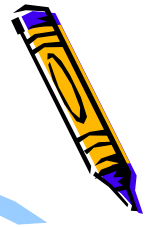
- 出来上がる主なファイルは次の通りです．
  - 400_oflow　　：Linux用のオプティカルフロー処理プログラム
  - encoder　　：BMPファイルのJPEG風エンコーダ
  - decoder　　：BMPファイルのJPEG風デコーダ
  - rgb2ycbcr　　：BMPファイルの色変換(RGB -> YCrCb)
- トップディレクトリで，以下のコマンドにより400_oflowプログラムを起動します．

```
$ ./400_oflow
```

- 以下の様に、ホストプログラムからの画像データを待ち受ける状態になります．
  - UDP/8100ポートにて32KBバイトの画像データ2セットを待ち受けます．

```
$ ./400_oflow
Starting 400_oflow (batchID:410) ===
Waiting recv 32KB at 0906c008
```

SDKの使い方 (2) Linuxプログラムの動作確認-2



- 次に，別ウィンドウを開き，以下のコマンドでホストプログラムを起動します．

```
$ cd client
$ ./all.sh
```

- all.shは画像データセットを送るスクリプトを4つ順次起動します．
  - (410_tux.sh, 411_anim.sh, 412_star.sh, 413_rectangle.sh)
  - この際，スクリプトの内部では，javaのUDP送信プログラムを起動します．
  - all.shスクリプトの引数には，UDPパケットの送信先IPアドレスを指定可能

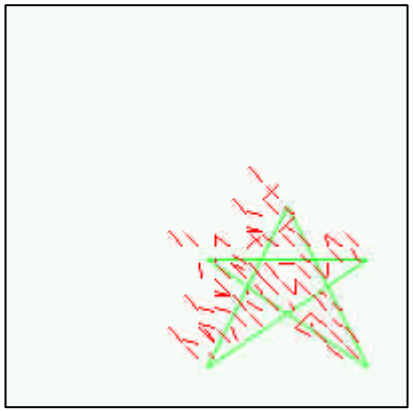
```
$ ./all.sh
targetHost:127.0.0.1 targetPort:8100
UDP Socket created. sending file '410_tux.64KB'
started!
64KB sent

finished!
after-before = 3331318965 (ns) = 3.331318965(s)
（続く）
```

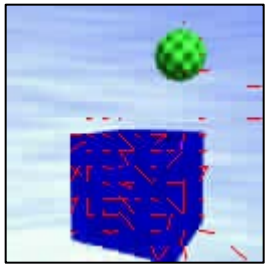
SDKの使い方 (2) Linuxプログラムの動作確認-3

- 正しく動くと、4回のオプティカルフロー計算処理が走ります。
- 結果は、ホストPC側（クライアント側）にRESULT.***.bmpファイルとして保存されます。

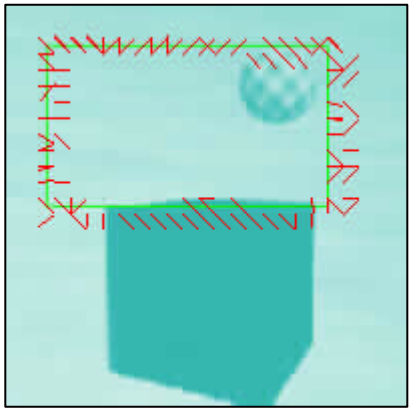
410_tux



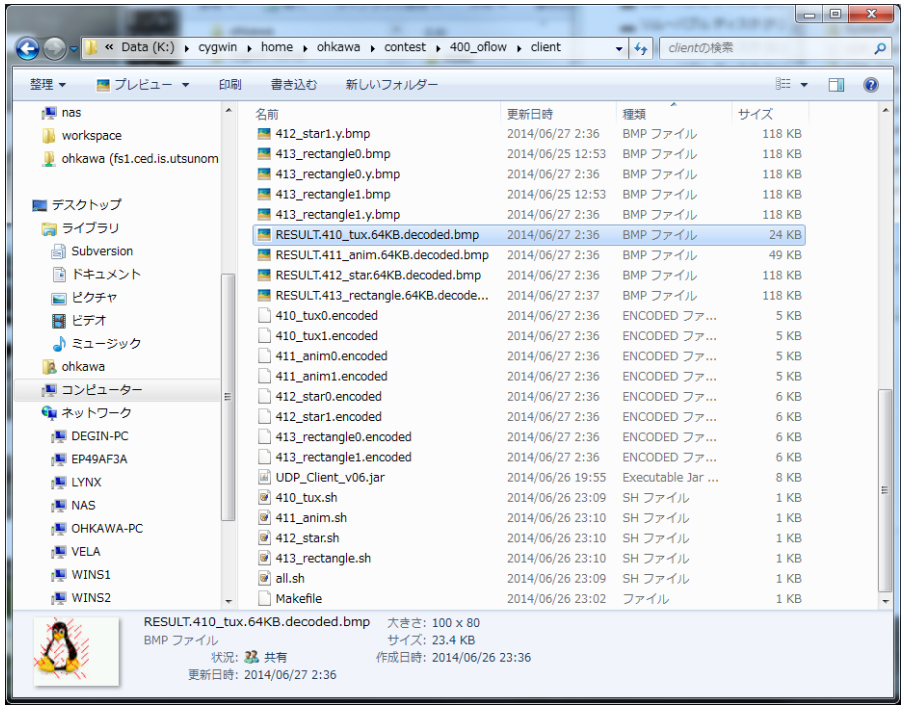
412_star



411_anim



413_rectangle



SDKの使い方 (3) Xilinx/Altera FPGAボード向けソースファイルのexport方法

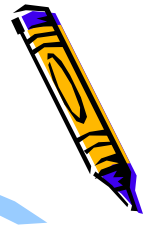
- FPGAボード上で動作するソフトウェアのソースファイルをexportするには，以下の様にします。

```
$ make export
```

- exportディレクトリ以下に、xilinx/srcディレクトリとaltera/srcディレクトリが出来ますので、その中の.cファイル・.hファイルを，Xilinx MicroBlaze環境もしくはAltera Nios2環境にコピーして使用してください。

- 以降は，別ドキュメントにて，Xilinx/Altera環境での動作方法について説明します．
  - P61　Xilinx環境
  - P62　Altera環境

SDKの使い方 (4) 既知の問題

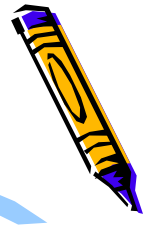


- 現在のリファレンスデザインには，以下に挙げる問題があることが分かっています．注意してお使いください．
  - 元画像(BMP)をエンコードし，デコードすると元画像に戻らない場合がある．（特に画像サイズが大きい場合）

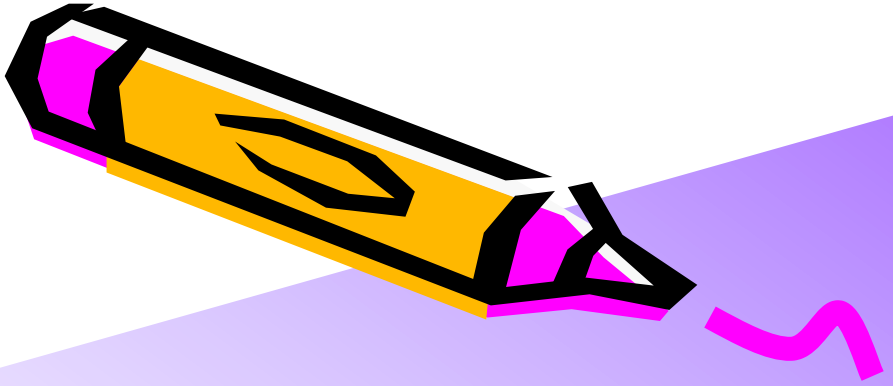
謝辞

- 参考文献[1]の著者である昌達慶仁氏，ならびに，ソフトバンククリエイティブ社には，書籍のプログラムを本コンテストで使用させて頂く事をご快諾して頂きました．おかげさまで，コンピュータシステム部門の競技としてJPEG圧縮，展開を使用した競技にすることができました．この場をお借りしまいて，厚く御礼申し上げます．

改訂履歴



- Ver.2014-06-26
  - コンピュータシステム設計部門のリファレンスデザイン(Xilinx,Altera)公開
- Ver.2014-06-06
  - 初版（第1回コンテストの内容に追加変更をした）

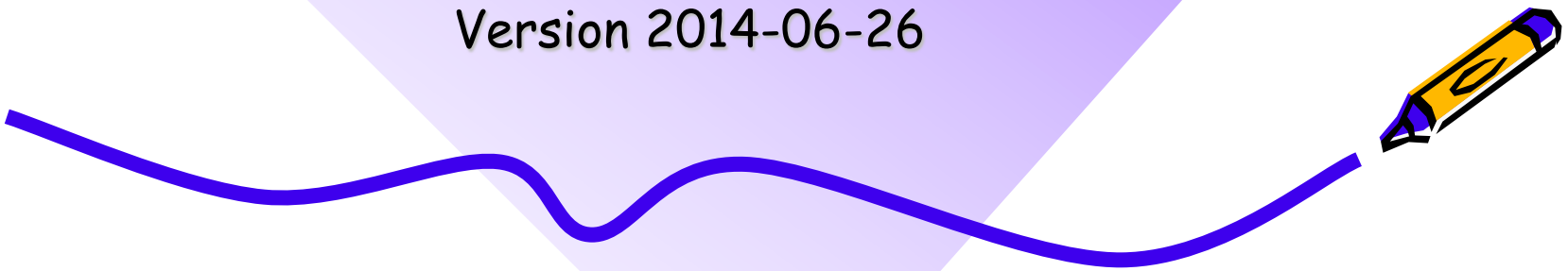


The 2nd ARC/CPSY/RECONF
High-Performance Computer System Design Contest

User Manual of Optical Flow System Reference Design (Xilinx ATLYS)

コンテスト実行委員会コアチーム

Version 2014-06-26



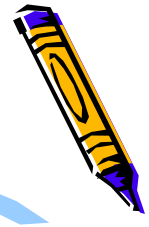
このドキュメント

- このドキュメントでは，Atlysボード用のリファレンスデザインに含まれるシステム構成について説明します．
- また，Xilinx Platform Studio (ISE14.7)を用いて，リファレンスデザインの回路ファイル（bitファイル）を生成し、プログラムしたFPGA上で400_oflowソフトウェアを動作させる方法を示します．

- 設計コンテストのWEBサイト
  - http://aquila.is.utsunomiya-u.ac.jp/contest/

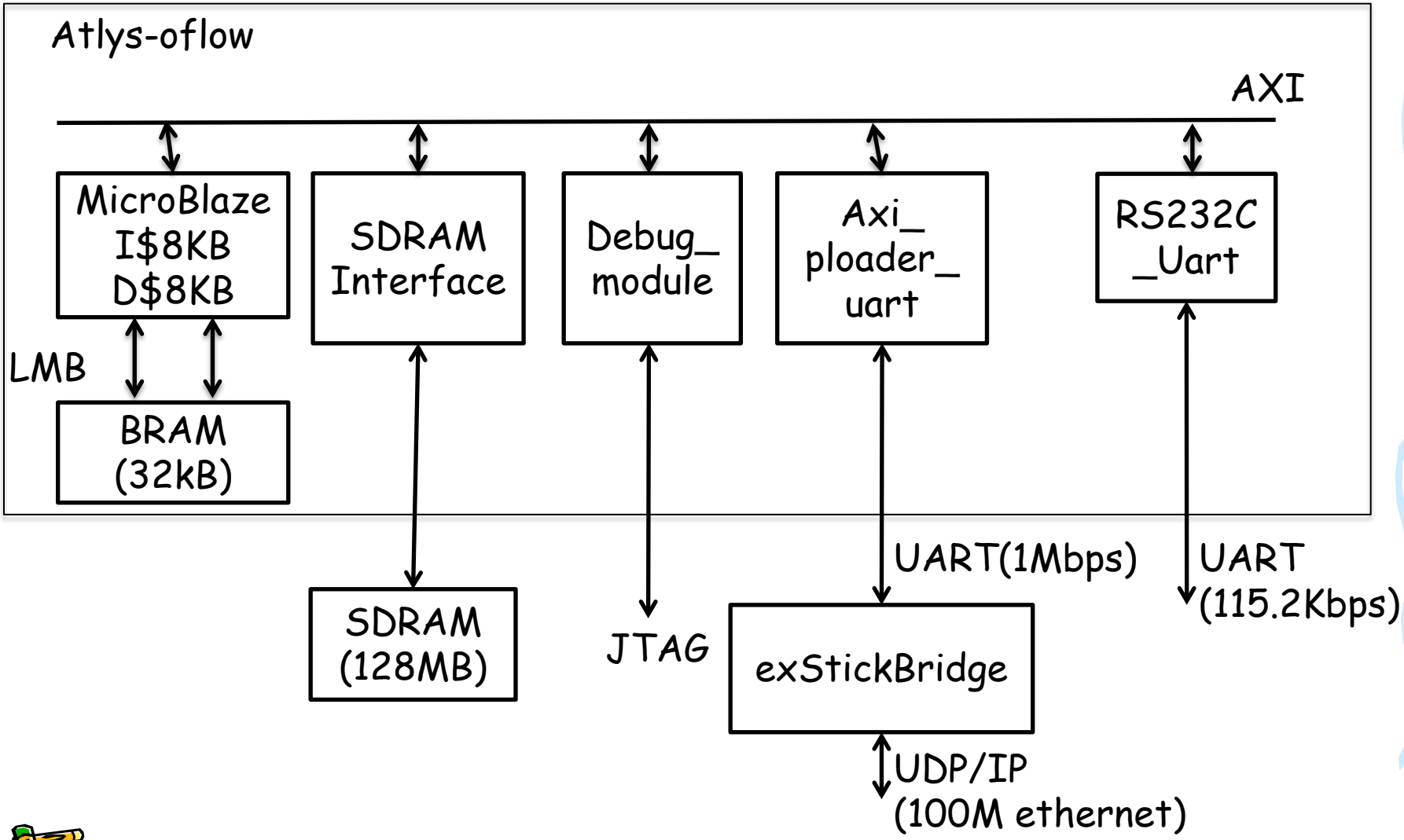
- 不明な点は，以下のいずれかの方法でお問い合わせください．
  - メールアドレス(contest_support@virgo.is.utsunomiya-u.ac.jp)
  - twitter(#arc_procon)
  - 技術情報掲示板
    - Google Group: HpCpsyDC2014
    - https://groups.google.com/forum/?hl=ja#!forum/hpcpsy2014dc

最初に

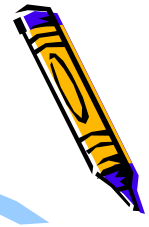


- Xilinx版とAltera版は機能的には同じですが，異なる所が多々ありますので，注意して下さい．
- Xilinx版のリファレンスデザインは，ISE14.7のEDK Xilinx Platform Studioで開発できます．ISE WEB Packでは開発できませんのでご注意ください．
- 大学関係者であればXilinx University ProgramのISEライセンス申請が可能です．
  - http://japan.xilinx.com/support/university.html
- またISEの無償評価版ライセンスがXilinxのホームページより申請可能です．
  - http://japan.xilinx.com/ise_eval/index.htm

リファレンスデザインAtlys-oflowのブロック図



システム構成の概要



- 本システムの構成は、Digilent社のWEBサイトにて提供されている，"Atlys board support files for EDK BSB wizard"を用いて作製しました.
  - http://www.digilentinc.com/

- 構成要素
  - MicroBlazeソフトコアプロセッサ
    - 設定）浮動小数点なし，32ビット乗算器搭載，I$8KB，D$8KB
  - Block RAM (BRAM) 32KB
  - SDRAM I/F (128MB)
  - MicroBlaze Debug Module (MDM) – JTAGによるデバッグ接続
  - RS232C_Uart
    - AtlysボードのUART-USB変換用のシリアル通信（115.2Kbps）
  - Axi_ploader_uart
    - 本コンテスト用のシリアル通信(1Mbps)
      → PMODコネクタ経由でexStickBridgeに接続

メモリマップ

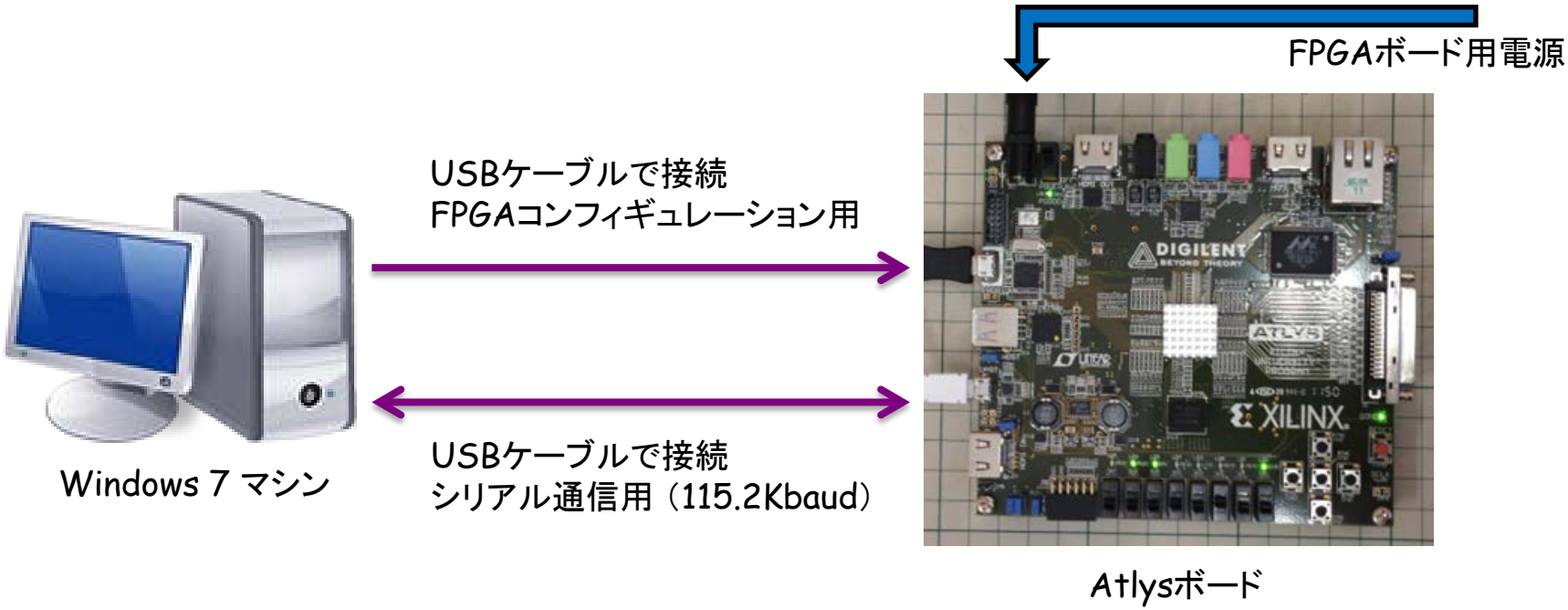
メモリアドレス		
開始アドレス	終了アドレス	
0x0000_0000	0x0000_7FFF	Block RAM (on-chip)
0x4060_0000	0x4060_FFFF	RS232_Uart (115.2kbps USB-UART)
0x77A0_0000	0x77A0_FFFF	Axi_ploader_uart (1Mbps UART via PMOD)
0xA800_0000	0xAFFF_FFFF	SDRAM (128MB)

コンピュータシステム設計部門 ATLYSボード用ハードウェアリファレンスデザインの使用方法

- コンテストWEBサイトから、コンピュータシステム設計部門のXilinx FPGA - Atlysボード用のリファレンスデザイン Atlys-oflow_v02.zipをダウンロードして展開します。
  - 展開先例）C:¥workspace-edk147¥Atlys-oflow
  - http://aquila.is.utsunomiya-u.ac.jp/contest/toolkit.html

ハードウェアセットアップ

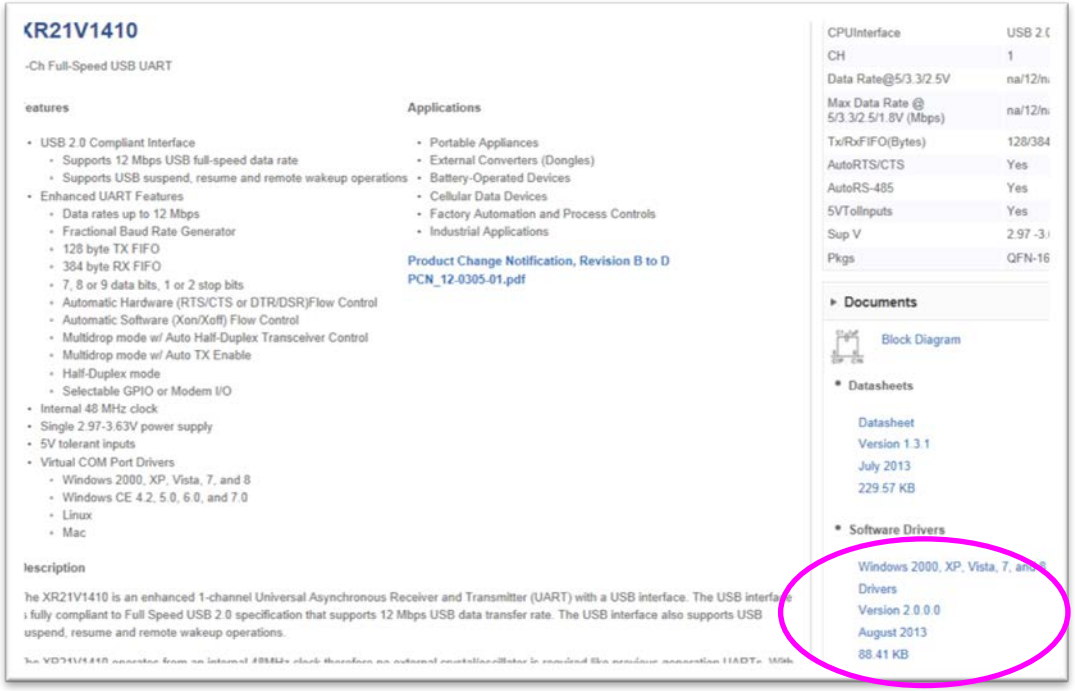
- Windows 7 マシン と Atlysボード を2本の USBケーブル で接続
- FPGAボード用電源 を使ってAtlysボードに電源供給



- 注意点
  - AtlysボードにはUSBケーブルが1本しか添付されていません．別途 1本のUSBケーブル（Aコネクタ-マイクロBコネクタ）を準備してください．

準備 USB-UARTドライバのインストール

- Windows 7 マシンにUSB-UARTドライバをインストール
  - http://www.exar.com/connectivity/uart-and-bridging-solutions/usb-uarts/xr21v1410
    - Software Drivers Windows 2000, XP, Vista, 7, and 8 Drivers Version 2.0.0.0 August 2013

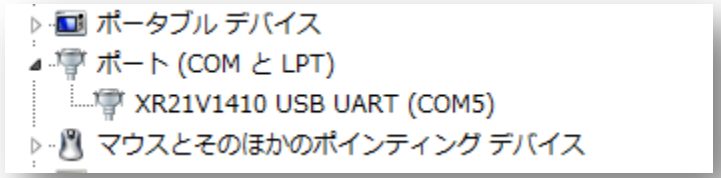


ソフトウェアセットアップ Tera Term のインストール

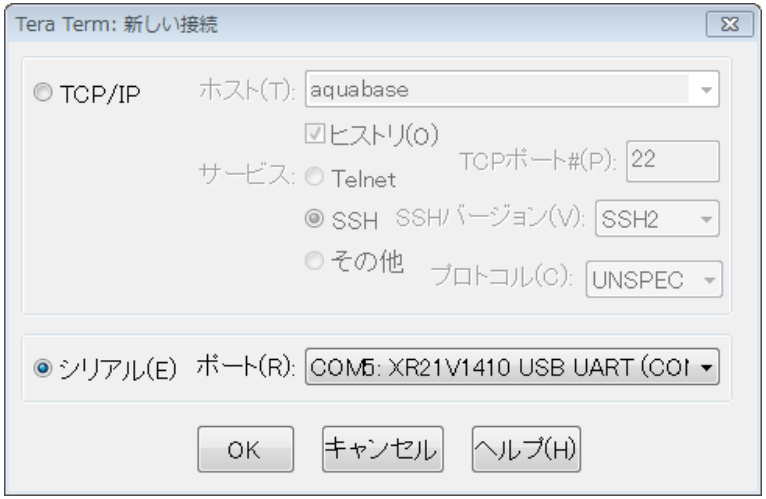
- Windows 7 マシンに Tera Termをインストール
  - http://sourceforge.jp/projects/ttssh2/
  - teraterm-4.78.exe

Tera Term の設定（1）

- ATLYSをコンピュータに接続し，COMポートが見えることを確認
  - Windows の デバイス マネージャー から確認
  - ポート番号(COM5)はコンピュータによって異なります．

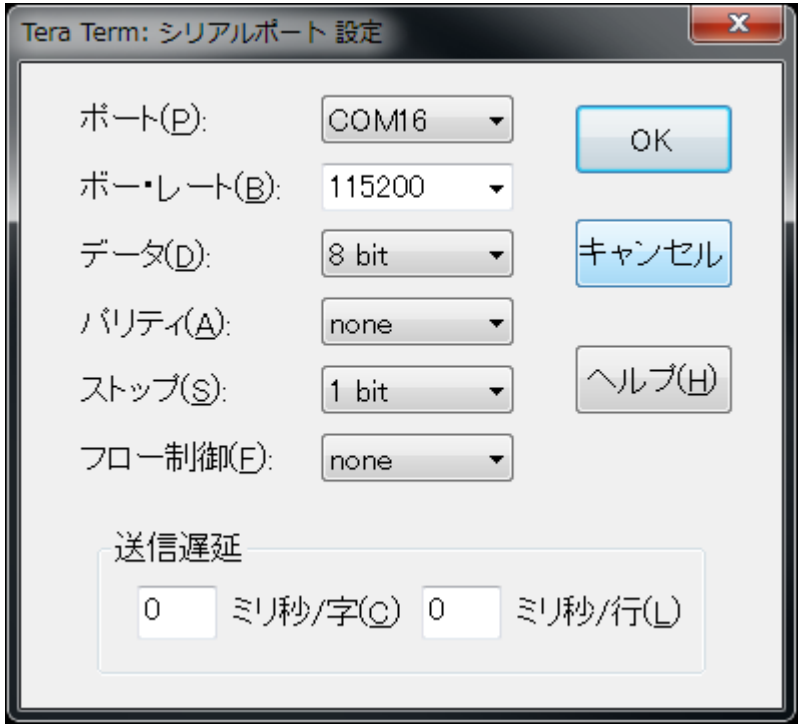


- Tera Term を立ち上げる(シリアル ポートを選択)



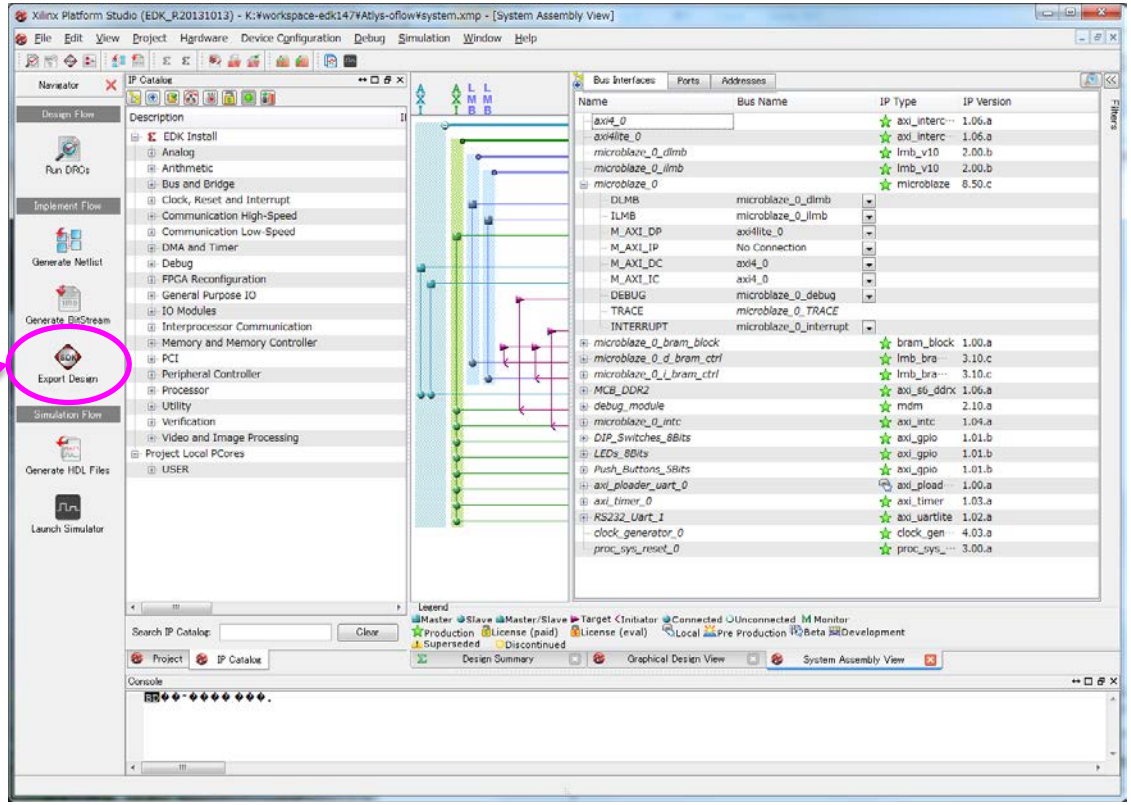
Tera Term の設定（2）

- Tera Termのシリアル設定を 115.2Kbaud に変更します.
  - 設定 -> シリアルポート -> ボー・レート
    115200を選択して，OK をクリック.



Xilinx Platform Studio (XPS)の起動とExport

- ISE Desugin Suite 14.7 → EDK → Xilinx Platform Studioを起動し、Atlys-oflow/system.xmpを開く
  - 以下、C:¥workspace-edk147¥Atlys-oflowに配置したとする
- 以下の画面になるので、"Export Design"をクリック



Export to SDK, XSDKの起動

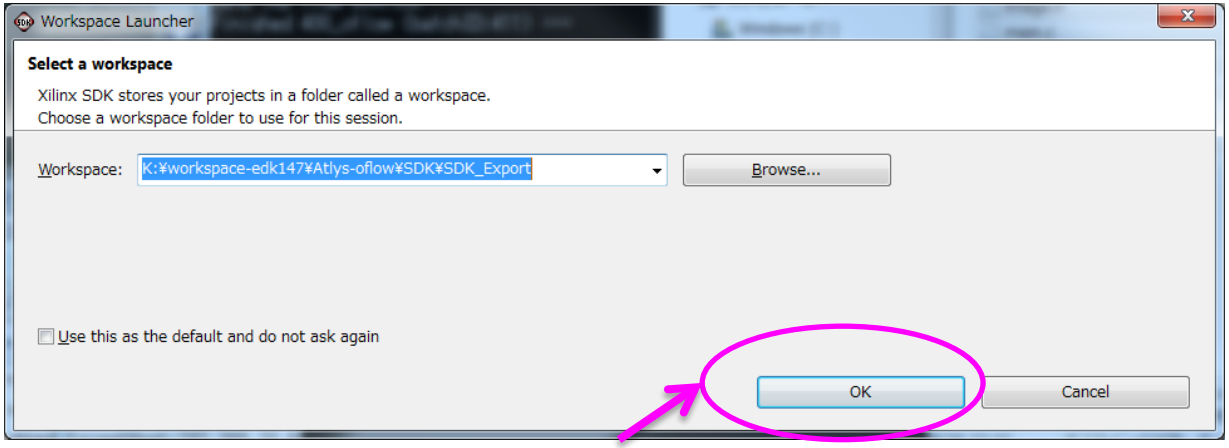
← Export & Launch SDKをクリック

<HDL生成・合成・配置配線・ビットストリーム生成に約20分程度かかる>

↓その後、以下の画面が出るので、

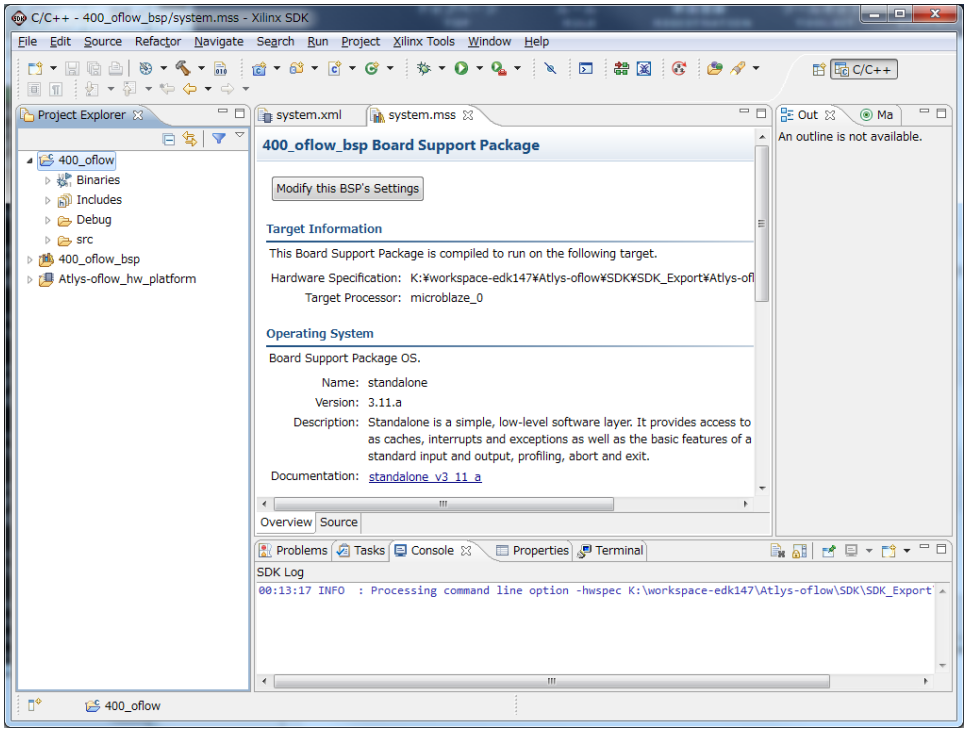
プロジェクトディレクトリの下の、"SDK/SDK_Export"ディレクトリを指定する。

例）C:¥workspace-edk147¥Atlys-oflow¥SDK¥SDK_Export



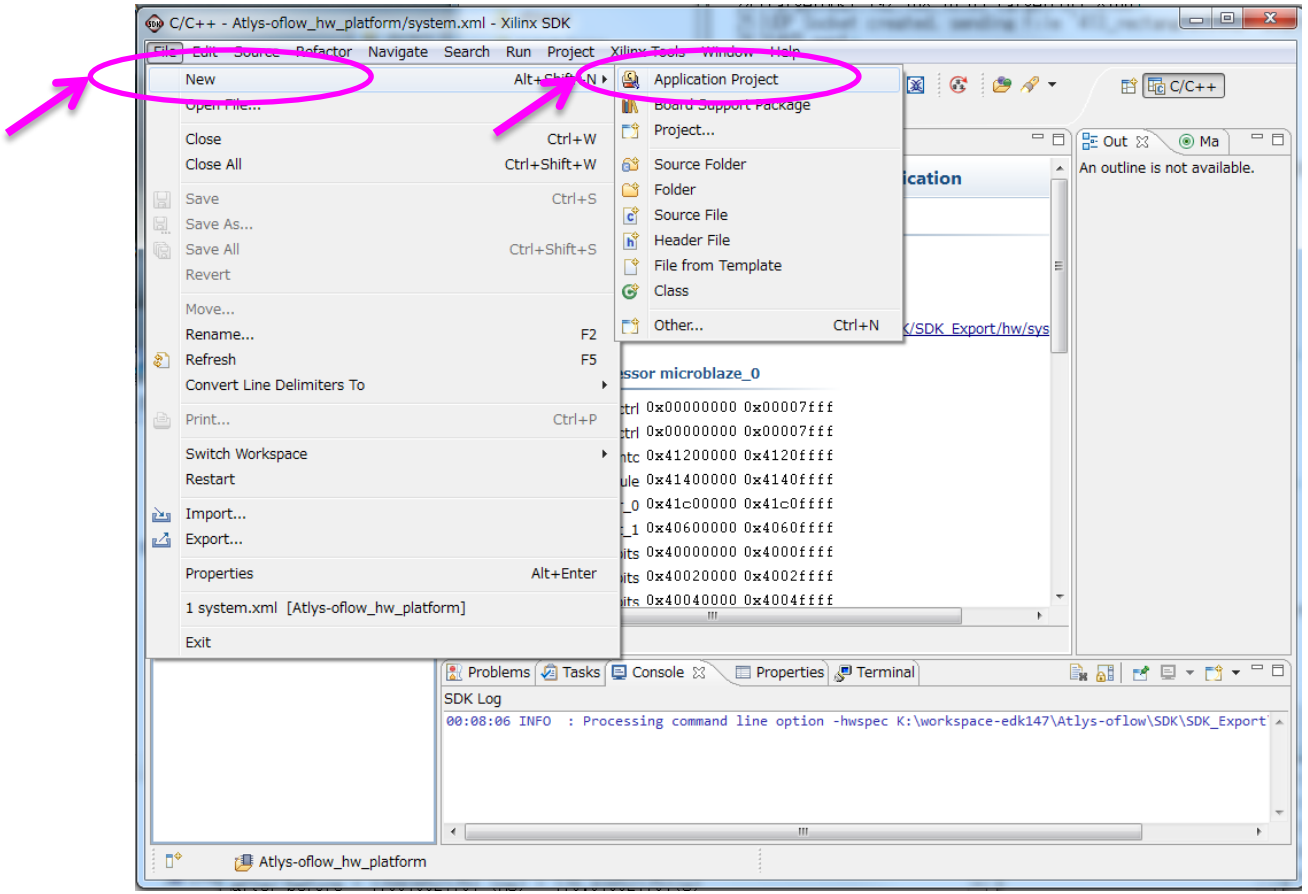
Xilinx SDKの起動

- 以下のような画面が現れるはず
- 次ページ以降は、リファレンスデザインのSDK(400_oflow_v02.tgz)からexportした.c, .hファイルを用いてアプリケーションを作成する方法を説明する。



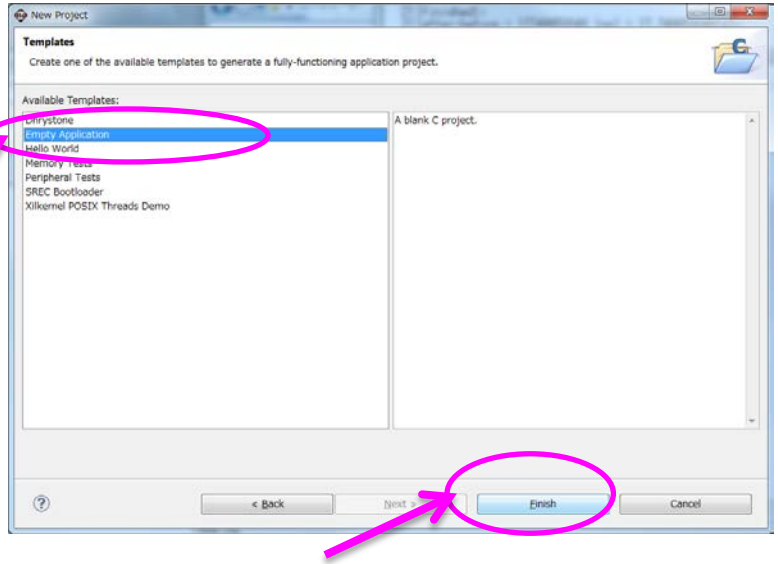
アプリケーションプロジェクトの作成 (1/3)

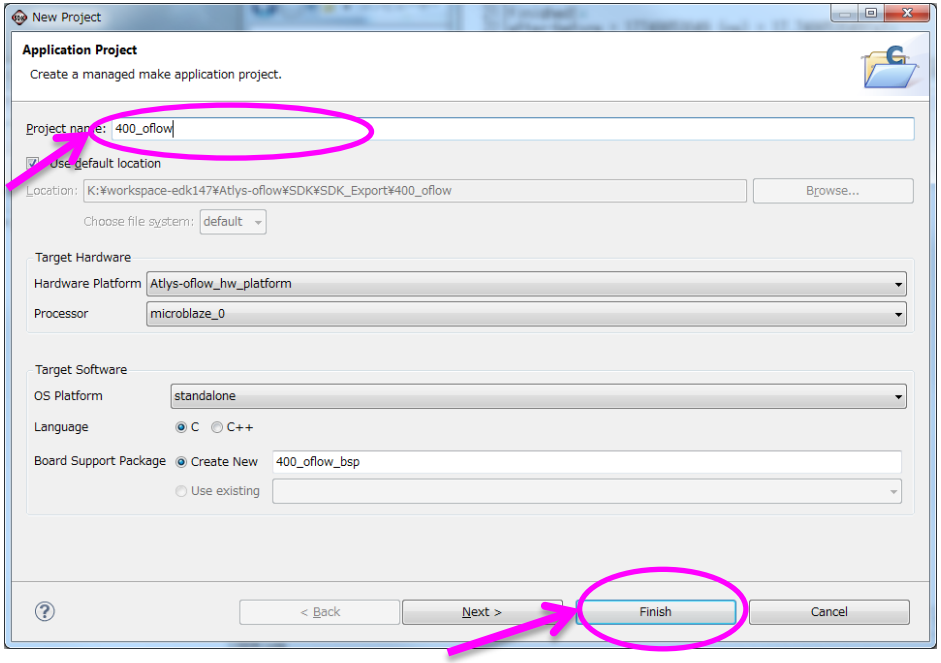
- メニューFile→New..から、Application Projectを選択



アプリケーションプロジェクトの作成 (2/3)

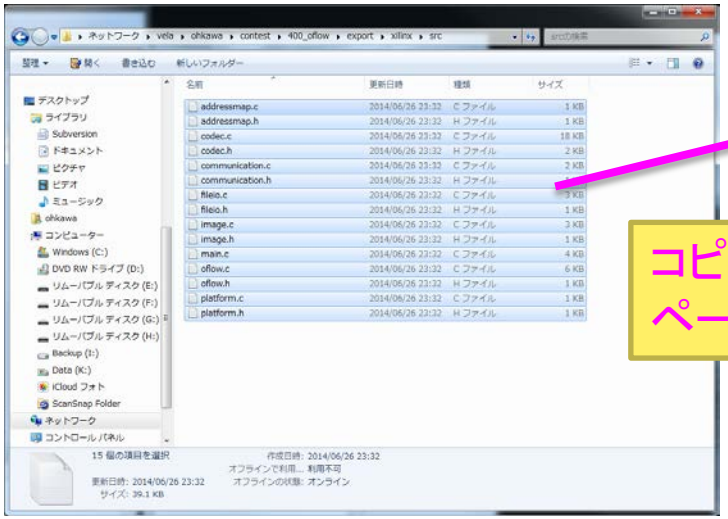
- プロジェクト名を設定（例：400_oflow）
  - Nextをクリック
- 次の画面で、Empty Projectを選択
  - Finishをクリック





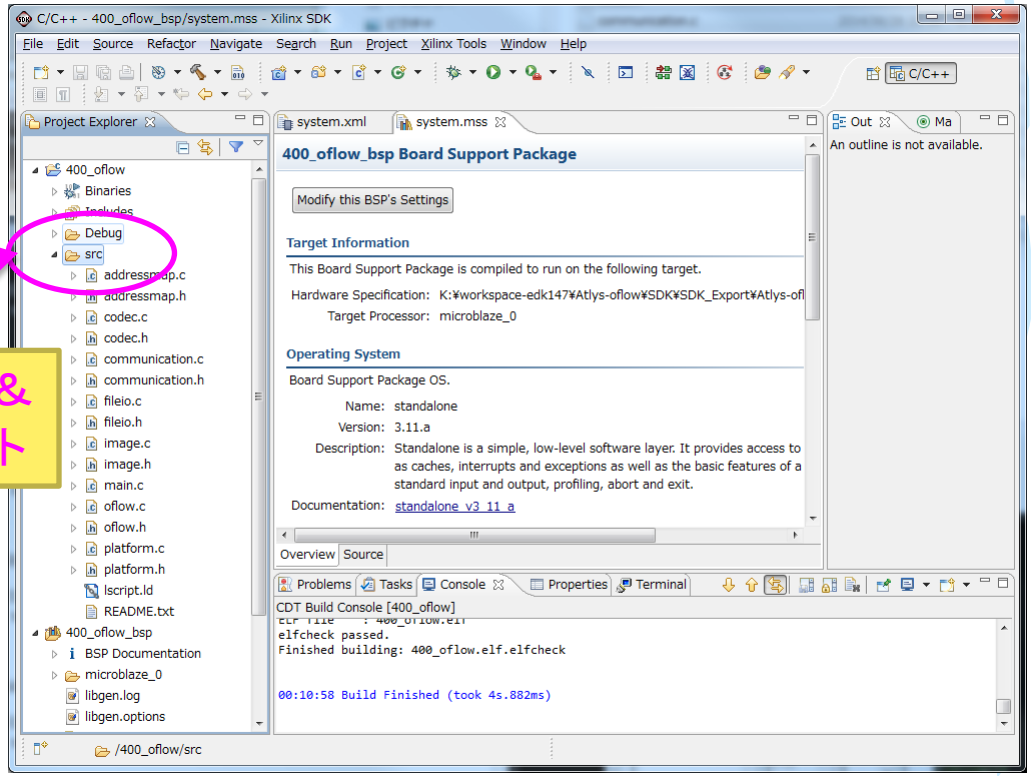
アプリケーションプロジェクトの作成 (3/3)

- SDK(400_oflow_v02.tgz)からexportした.c, .hファイルをコピー(Ctrl+C)
- XSDKの400_oflowプロジェクトのsrcディレクトリをクリックした後に、ペースト(Ctrl+V)
  - Drag&DropでもOK
- 自動的にビルドが始まる



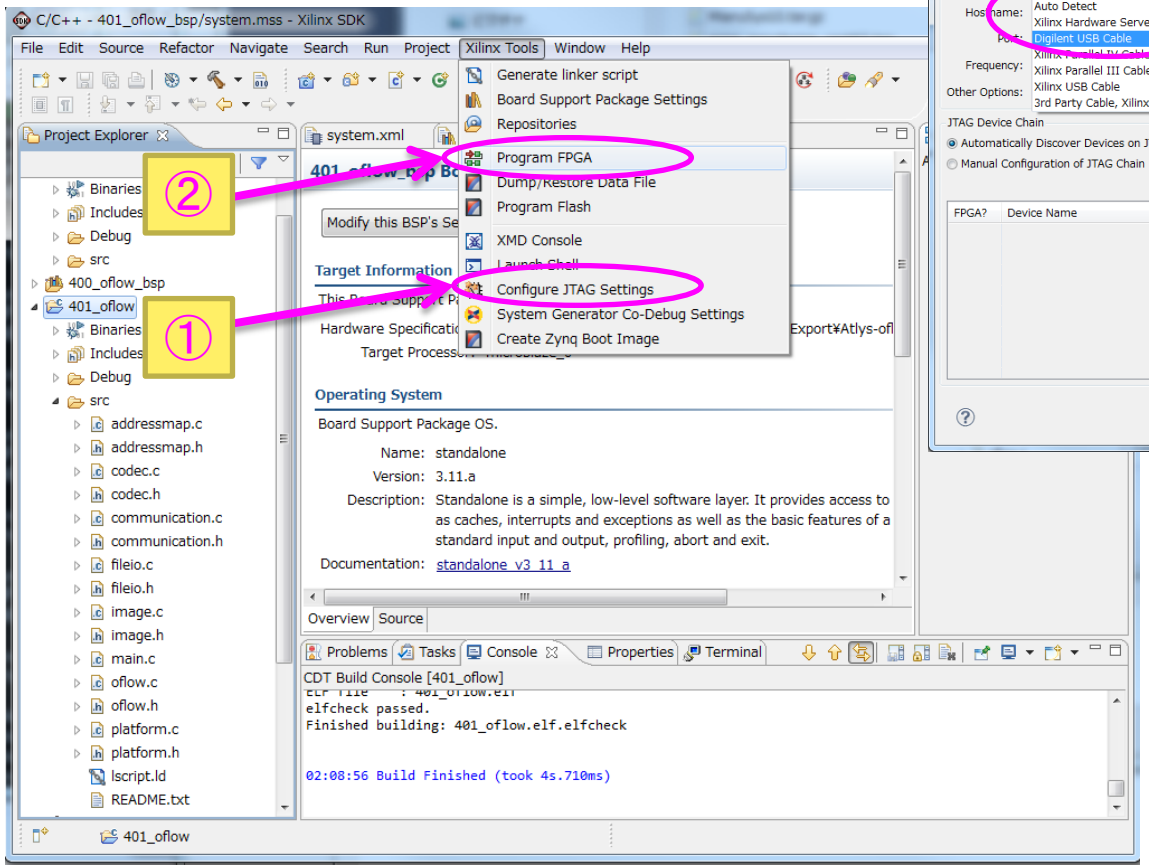
400_oflow¥export¥xilinx¥src

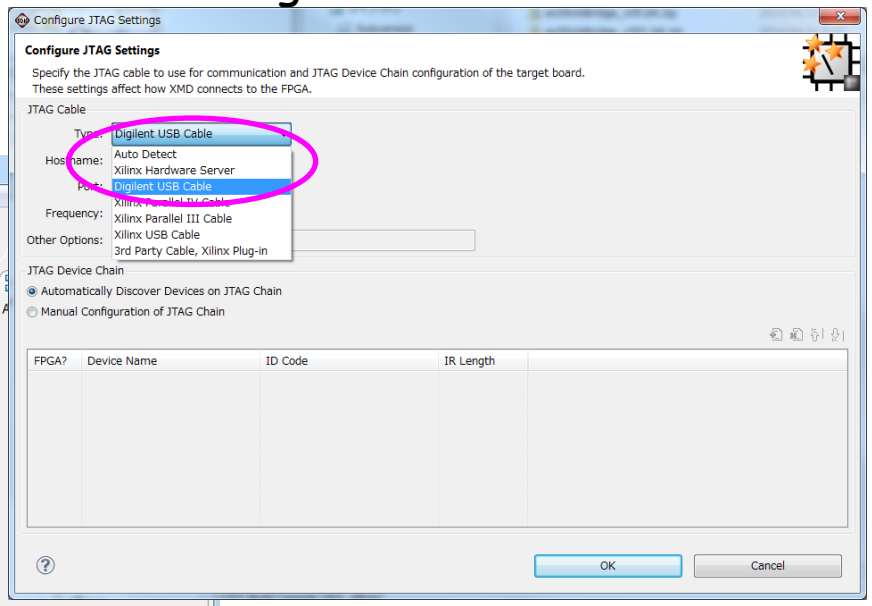
コピー＆ペースト

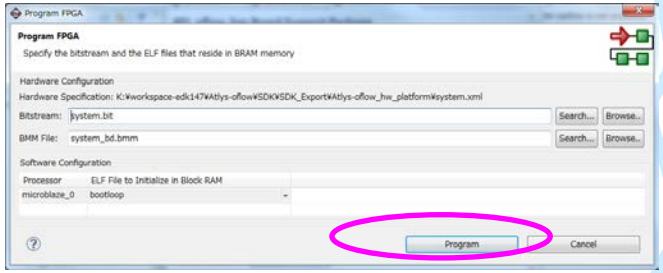


JTAGケーブル設定＆FPGAプログラム

- ①JTAGの設定をします。右のWindowが出るのでDigilentを選択
- ②Program FPGAして下さい。
  - 次のWindowでProgramをクリック

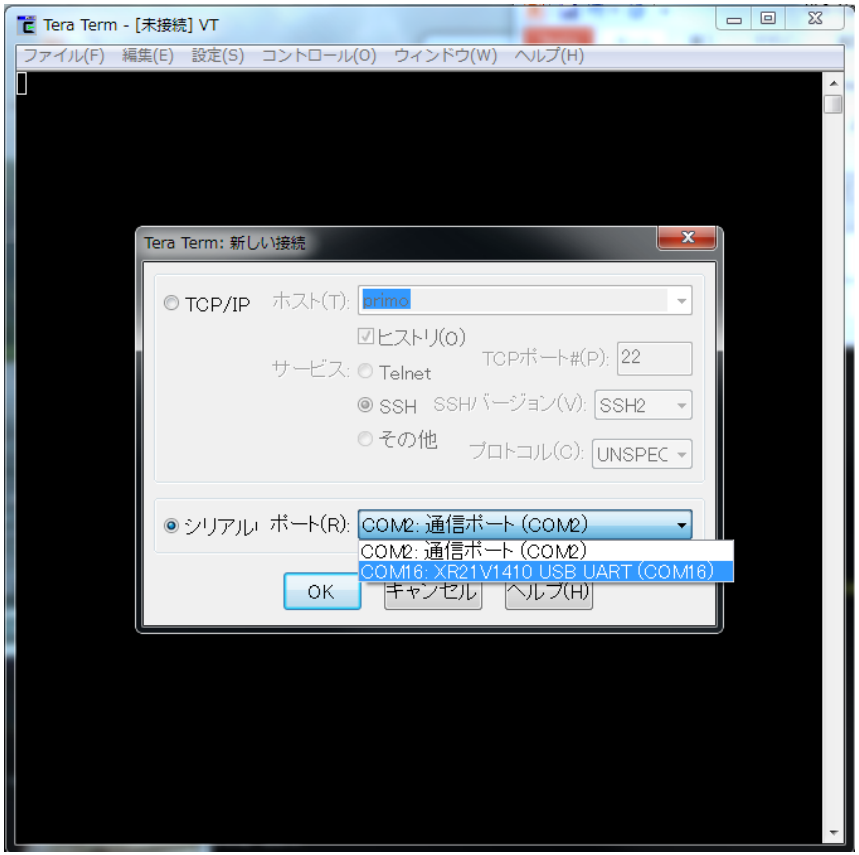






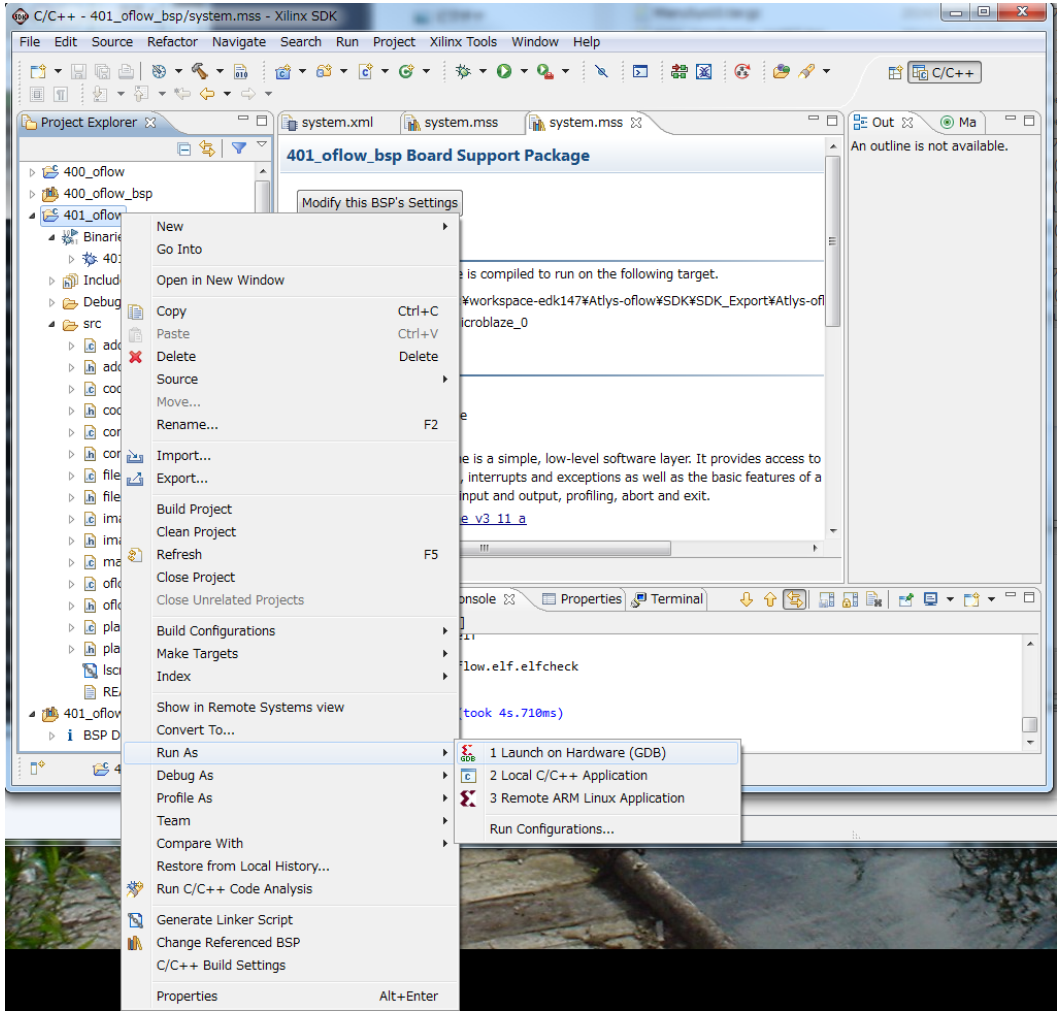
デバッグ用シリアルコンソールの接続(TeraTerm)

- プログラム実行前にTera Termにて、シリアルコンソールを立ち上げておきます。



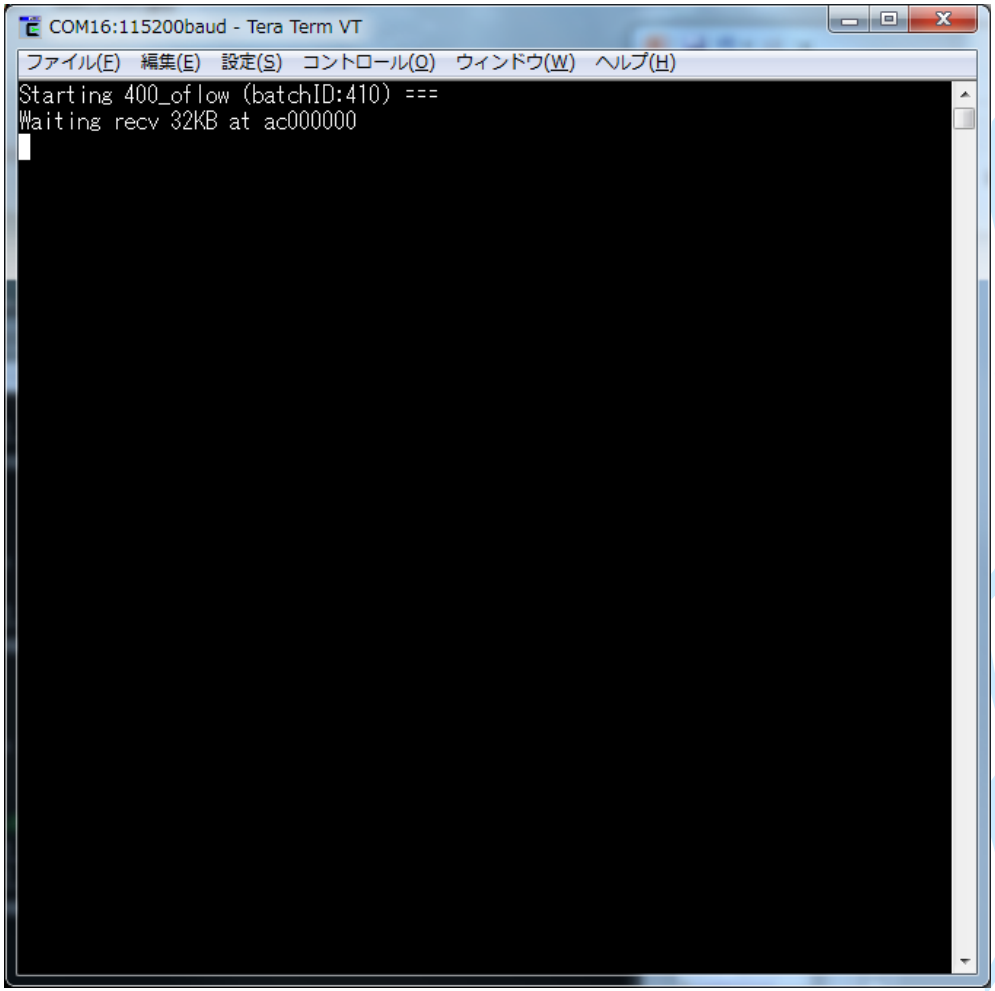
プログラムの実行

- プロジェクトを右クリックし
- Run Asのメニューを選び
- Launch on Hardware (GDB)をクリックしてください



400_oflowの起動

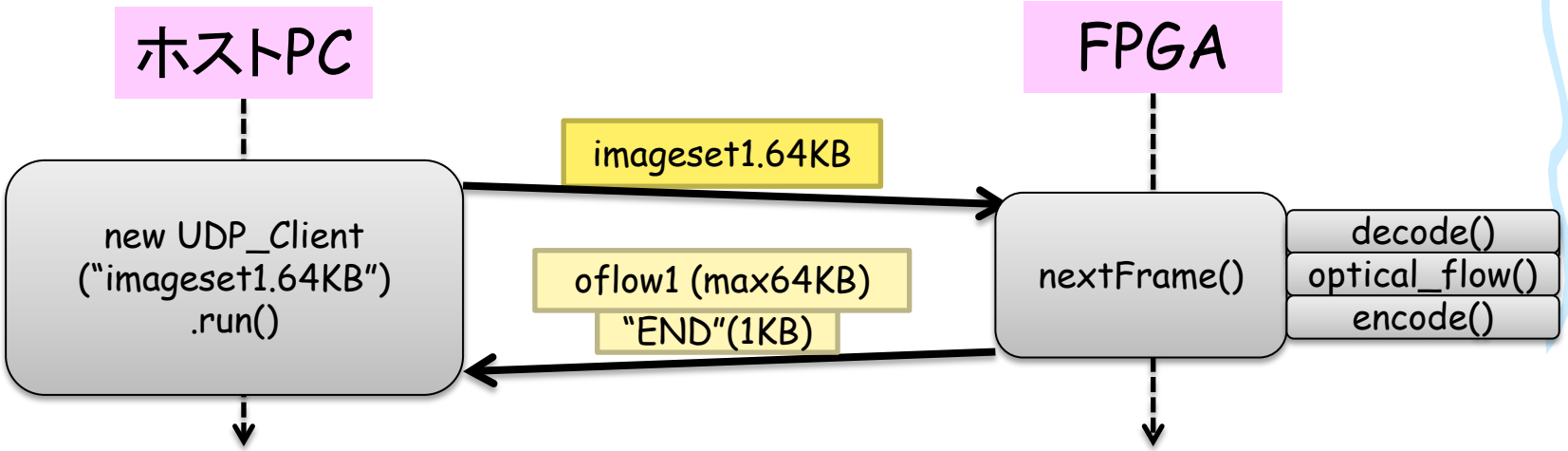
- Starting 400_oflow・・・という起動メッセージが表示されるはず
  - 表示されない場合は
    ここまでの作業を再確認
- 32KB＋32KBの画像データを
  UDP/8100番ポートで
  待ち受けています
  （exStickBridge経由）



ホストPCでのUDP通信プログラムの起動

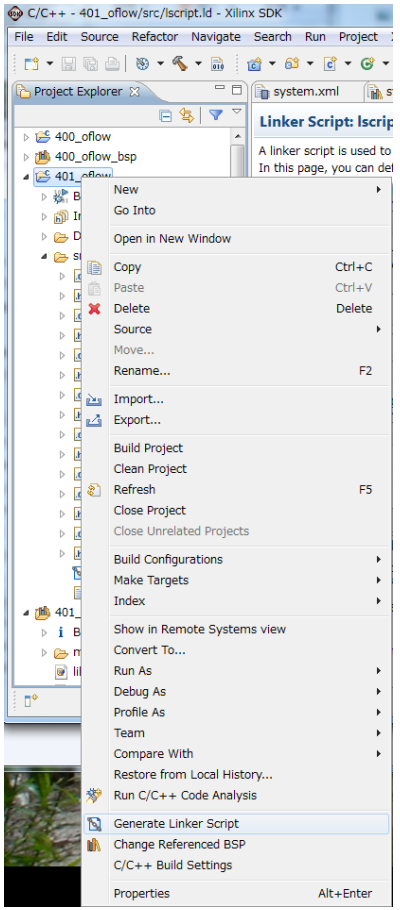
- ホストPCにて、画像データを用意し、UDPによりFPGAに転送する必要があります。
- リファレンスデザインのSDK(400_oflow_v02.tgz)を、ホストPCに展開し、clientディレクトリにて、all.shスクリプトを動かします。(IPアドレスを指定)

```
$ ./all.sh 192.168.10.64
targetHost:192.168.10.64 targetPort:8100
UDP Socket created. sending file '410_tux.64KB'
started!
64KB sent
```



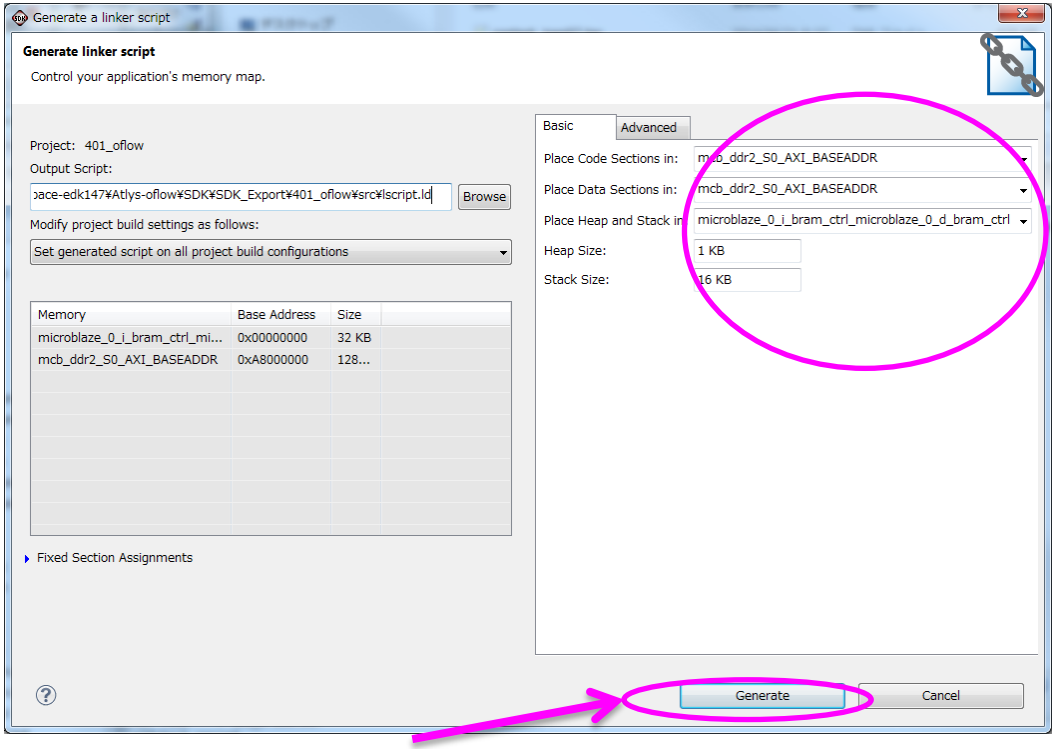
スタック領域の設定 (1/2)

- しかし、最初の画像を読み込んだ後、止まってしまうかもしれません。
  - XSDKのデフォルトでは、スタック領域が1KB(0x400)バイトしかありませんので、上手く動きません。
- スタック領域の大きさを設定するには、
  プロジェクトを右クリックして、Generate Linker Script を選択します



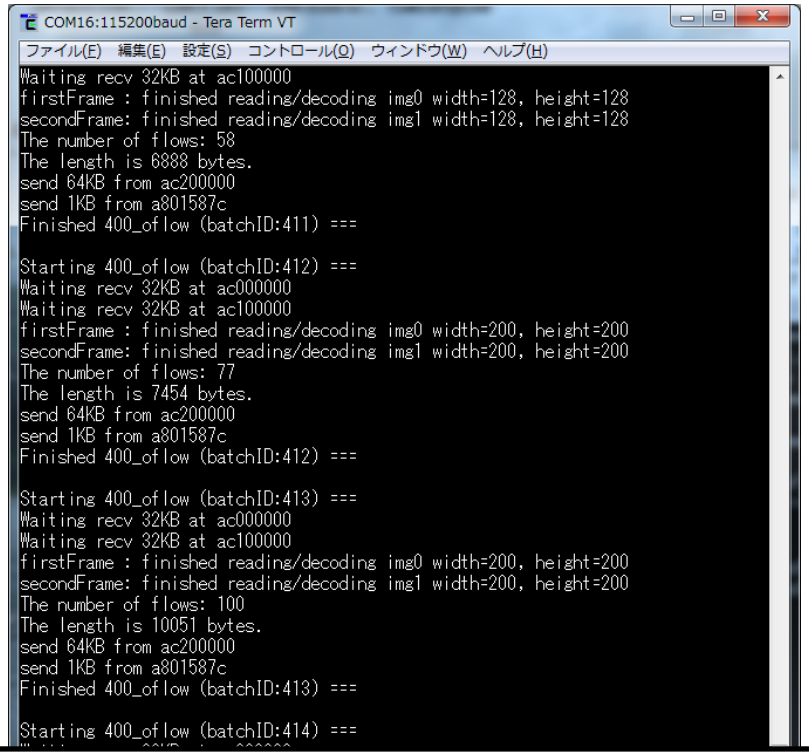
スタック領域の設定 (2/2)

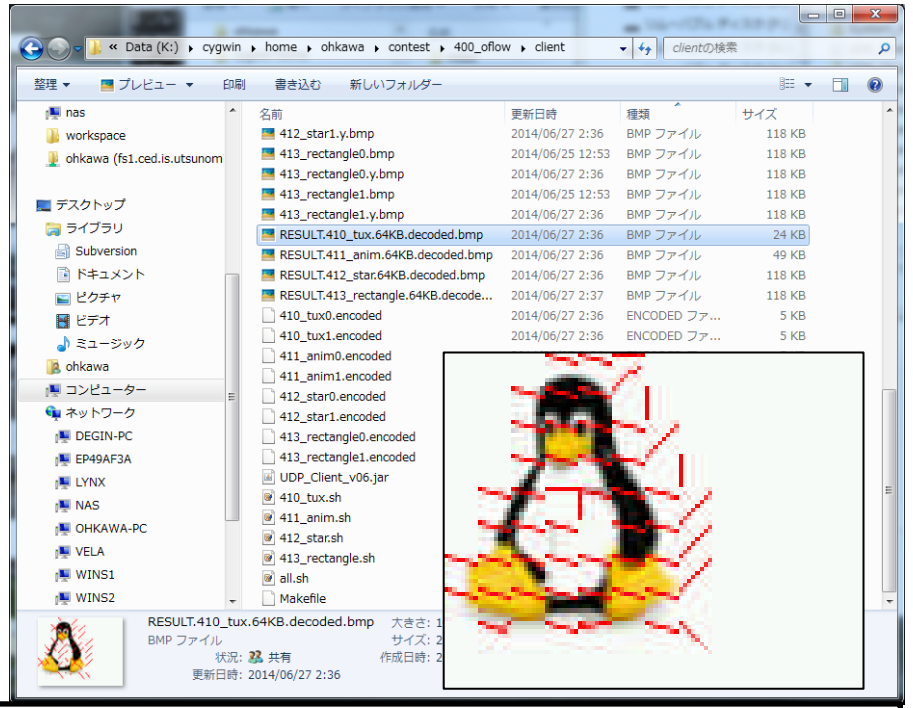
- Generate Linker Scirptダイアログで、例えば以下の設定をします。
  - CodeSectionの配置: DDR2
  - DataSectionの配置：DDR2
  - Heap&Stackの配置：bram
  - HeapSize: 1 KB
  - StackSize: 16 KB
- その後、Generateすると Overwriteするか聞かれる のでlscript.ldファイルを 更新します。



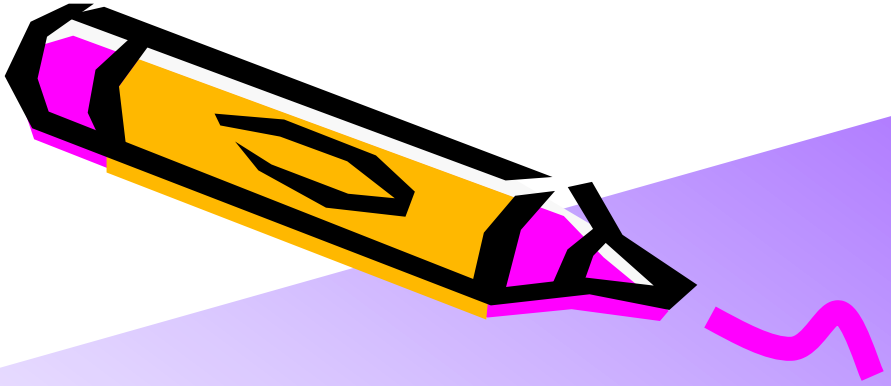
400_oflowの動作確認

- さて、動くでしょうか・・・？400_oflowプロジェクトを右クリックして、Runします
- ホストPC用のクライアントも再度動かしてください。($ ./all.sh 192.168.10.64)
- Linux上で動作させたときと同じ様にRESULTのBMPファイルが出来ればOK！





それではいよいよ高速化に挑戦する開発を始めましょう！！

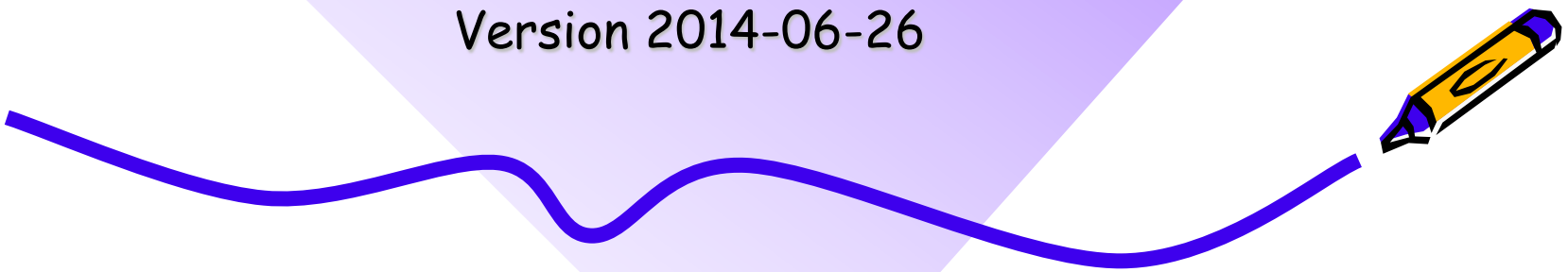


The 2nd ARC/CPSY/RECONF
High-Performance Computer System Design Contest

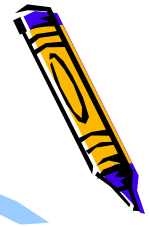
User Manual of Optical Flow System Reference Design (Altera DE2-115)

コンテスト実行委員会コアチーム

Version 2014-06-26



このドキュメント



- このドキュメントでは，Altera DE2-115ボード用のリファレンスデザインに含まれるシステム構成について説明します．
- また，Altera Quartusを用いて，リファレンスデザインの回路ファイル（sofファイル）を生成する方法を示します．

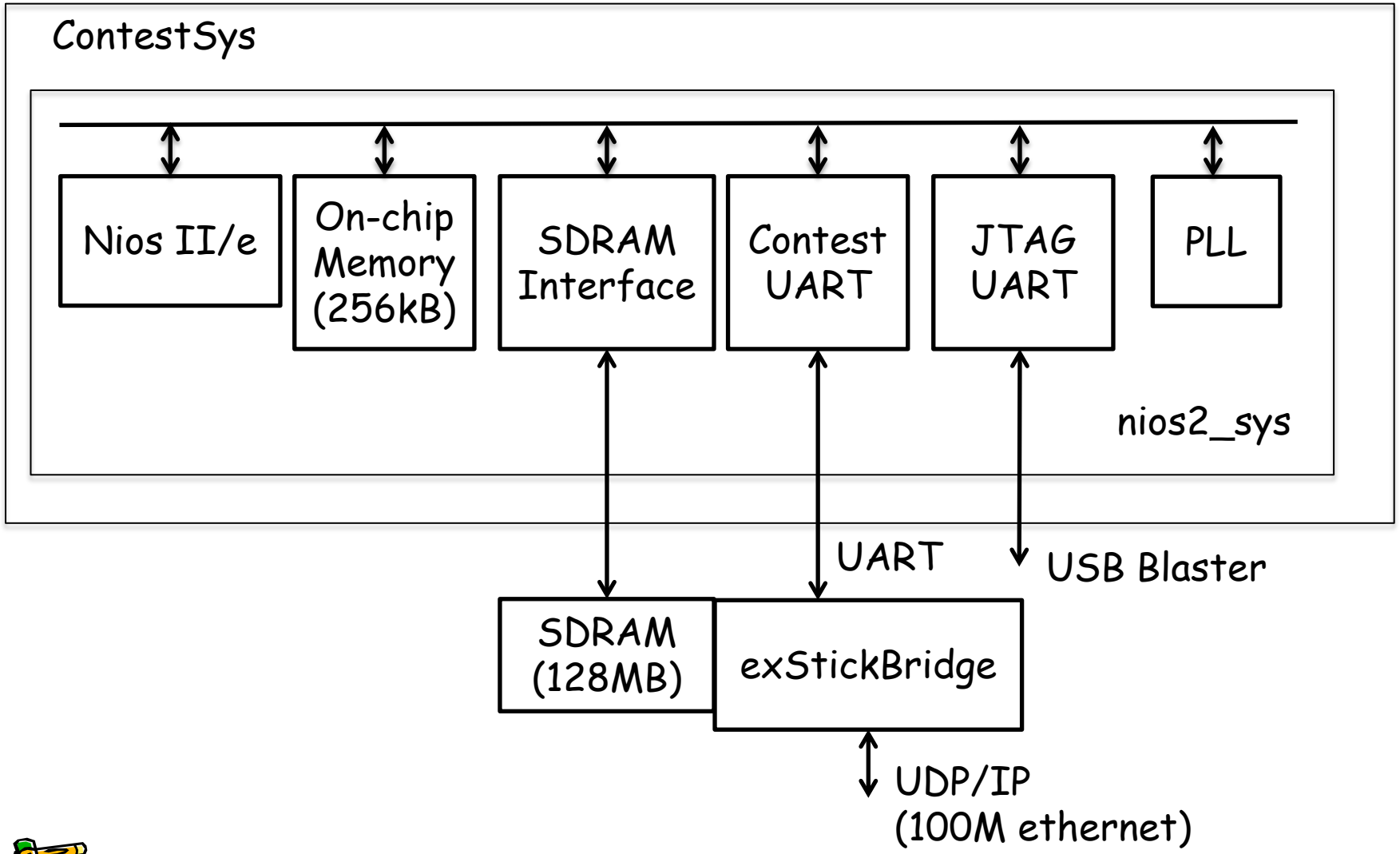
- 設計コンテストのWEBサイト
  - http://aquila.is.utsunomiya-u.ac.jp/contest/

- 不明な点は，以下のいずれかの方法でお問い合わせください．
  - メールアドレス(contest_support@virgo.is.utsunomiya-u.ac.jp)
  - twitter(#arc_procon)
  - 技術情報掲示板
    - Google Group: HpCpsyDC2014
    - https://groups.google.com/forum/?hl=ja#!forum/hpcpsy2014dc

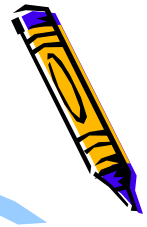
最初に

- Altera版ContestSysはXilinx版と機能的には同じですが，異なる所が多々ありますので，注意して下さい．
- Altera版のリファレンスデザインは，Quartus Web Editionで開発できるように考慮しております．そのため，Nios2の実装にはNios2/eを使用し，キャッシュ等は搭載しておりません．
- 大学関係者の方はAcademic License（無料）がAlteraのホームページより申請可能ですので，利用される事をお勧め致します．以下，簡単に手順方法を記載しておきます．
  1. AlteraのUniversity Program（http://www.altera.com/education/univ/unv-index.html）のページの左にあるメニューの中の，Member->License Requestをクリックする
  2. myAlteraのアカウント名，パスワードを入力して，申請ページへ行く．

リファレンスデザインContestSysのブロック図

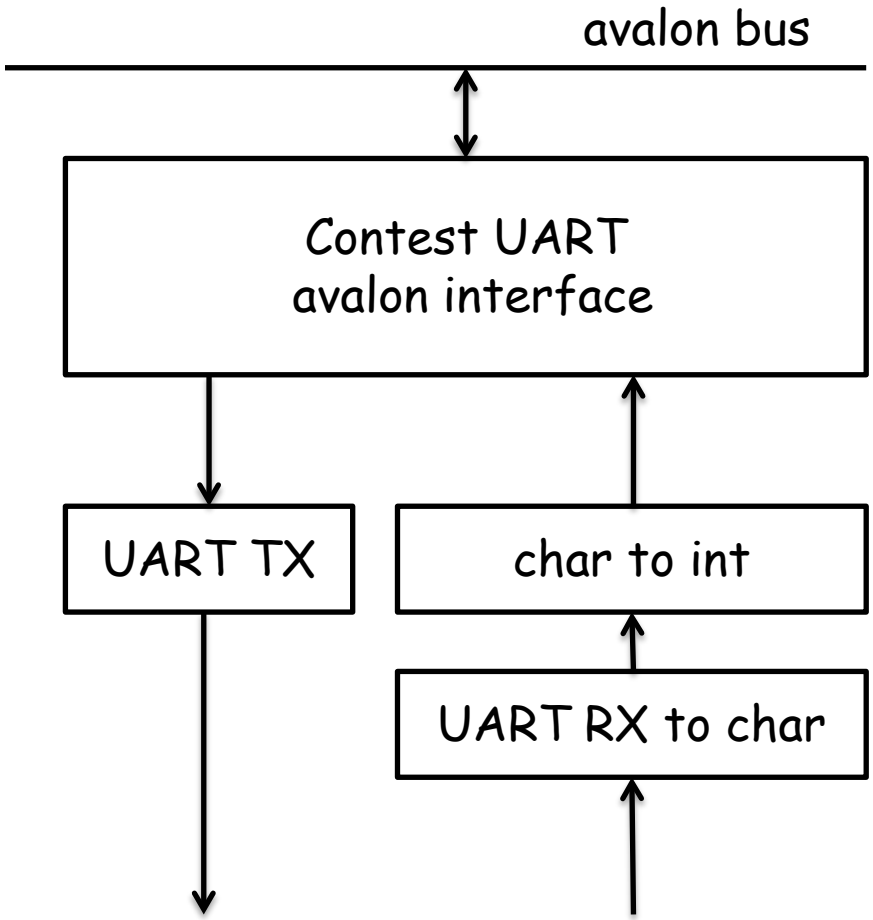


各機能の役割

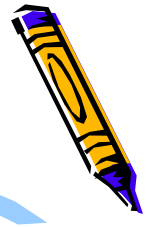


- PLL
  - 50MHzのクロックを供給
- On-chip Memory（256kB）
  - 命令メモリとして使用
- SDRAM
  - NiosIIの命令メモリ，データメモリとして使用．
- Contest UART
  - データ転送用のインターフェース．
  - UARTから受信したデータを32ビットのデータとして格納する．
  - 8ビットのデータをUART経由で送信する
- JTAG UART
  - NiosIIでの実行経過などをprintfで出力するインターフェース
- NiosII/e
  - オプティカルフローを計算するCPU

Contest UARTの詳細



Contest UARTの詳細



- contest UART avalon interface
  - UART RXやUART TXに対してavalonバスとのinterfaceになる
- char to int
  - UART RXを8ビットのデータにしたものを32ビットのデータに変換する
- UART RX to char
  - シリアルで入力されるUART RXのデータを8ビットのデータに変換する
- UART TX
  - 8ビットのデータをシリアルで送信する

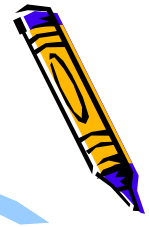
メモリマップ

メモリアドレス		
開始アドレス	終了アドレス	
0x0000_0000	0x0003_FFFF	on-chip memory
0x0004_1020	0x0004_102f	Contest UART
0x0005_0000	0x0005_07FF	Nios II/e
0x0800_0000	0x0FFF_FFFF	SDRAM

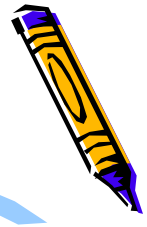
補足

- SDRAMの領域は0x0800_0000番地から使用出来るのですが，
  リファレンスデザインでは0x0c00_0000番地から使用しています．
  （SDRAM領域の0x0800_0000番地からの領域をデバッグに用いていたため）

リファレンスデザインの作成について

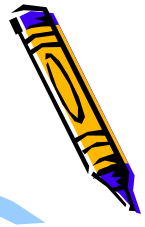


- 次ページ以降で作成するリファレンスデザインの作成方法では，Verilog-HDLの記述は既に完成しているものを使用するとします．
- 本ドキュメントで参照するファイルは以下のものです．
  - ContestSys05.tar.gz
    - DE2-115ボード用コンピュータシステム設計部門リファレンスデザインのプロジェクトファイル
  - DE2-115.qsf
    - DE2-115ボード用ピン設定ファイル．下記のURLから取得できます
      http://www.altera.com/education/univ/materials/boards/de2-115/unv-de2-115-board.html



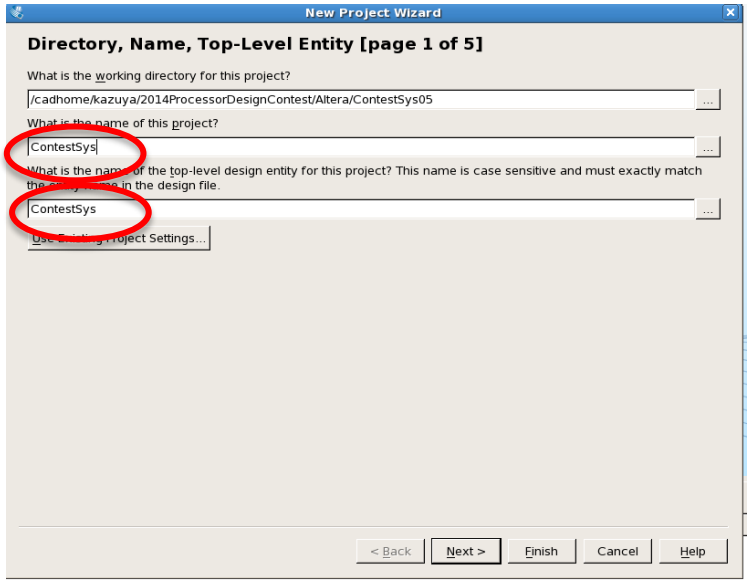
ハードウェアデザイン

新規プロジェクトの作成

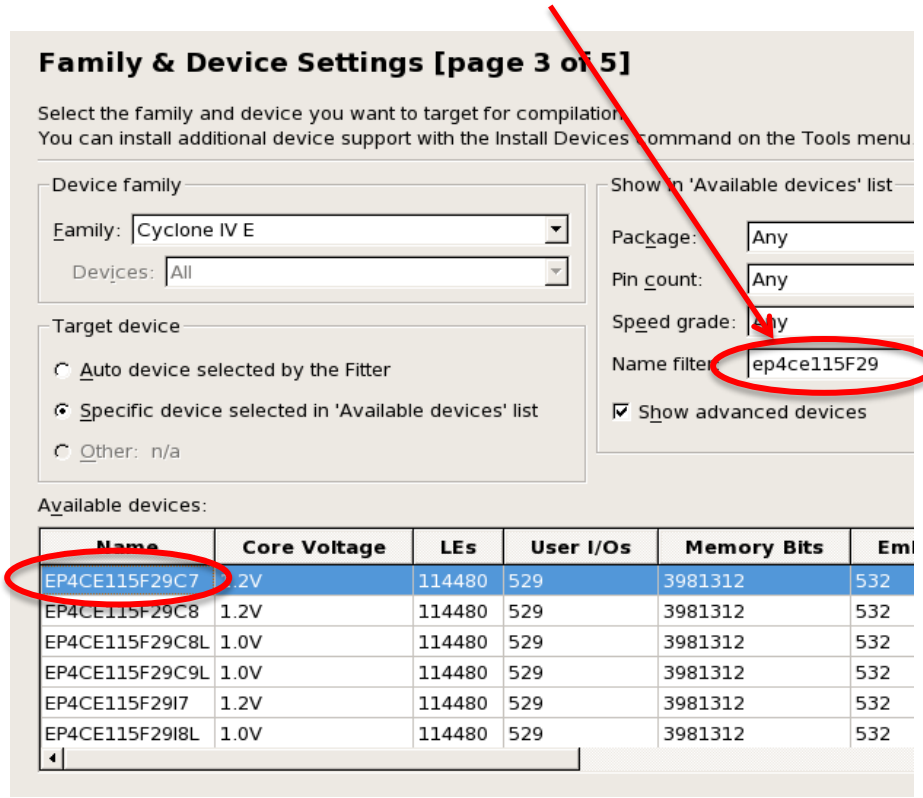


1. プロジェクトを置くディレクトリは新規に空のディレクトリを作成する．ここではContestSys06とします
2. 新規に作成したディレクトリでQuartus を起動する
3. File -> New Project Wizardを選択
    1. プロジェクト名をトップモジュール名（ContestSys）にする（次ページ左写真）．その後Next
    2. Add fileでは何も追加しない
    3. デバイス名はCyclone IV E EP4CE115F29C7を選ぶ（次ページ右写真）
    4. EDA toolの設定でSimulationツールとしてModelsimを選んでいる場合はFormatをVerilog HDLにする
    5. その他はデフォルトでNextを押し，finishまでいく

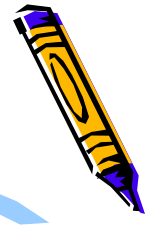
新規プロジェクトの作成



Name filterでデバイス名の候補をfilteringできる



Qsysの起動と外部クロックの設定

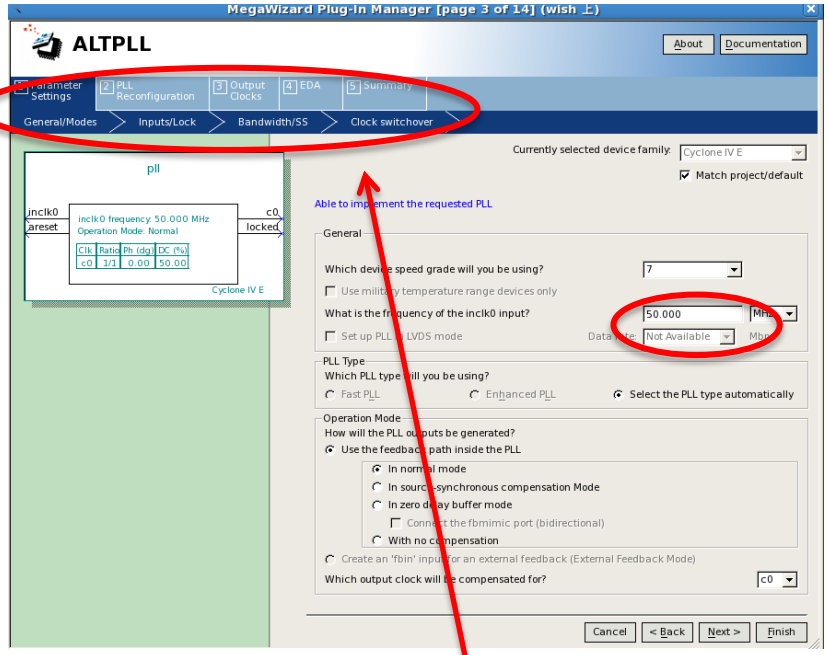


- ここではQsysのシステムとしてnios2 sysを作成します.
- Tools -> QsysでQsysを起動する
  （以下Qsysでの操作です．）
- File -> Save でnios2_sysという名前をつけて保存

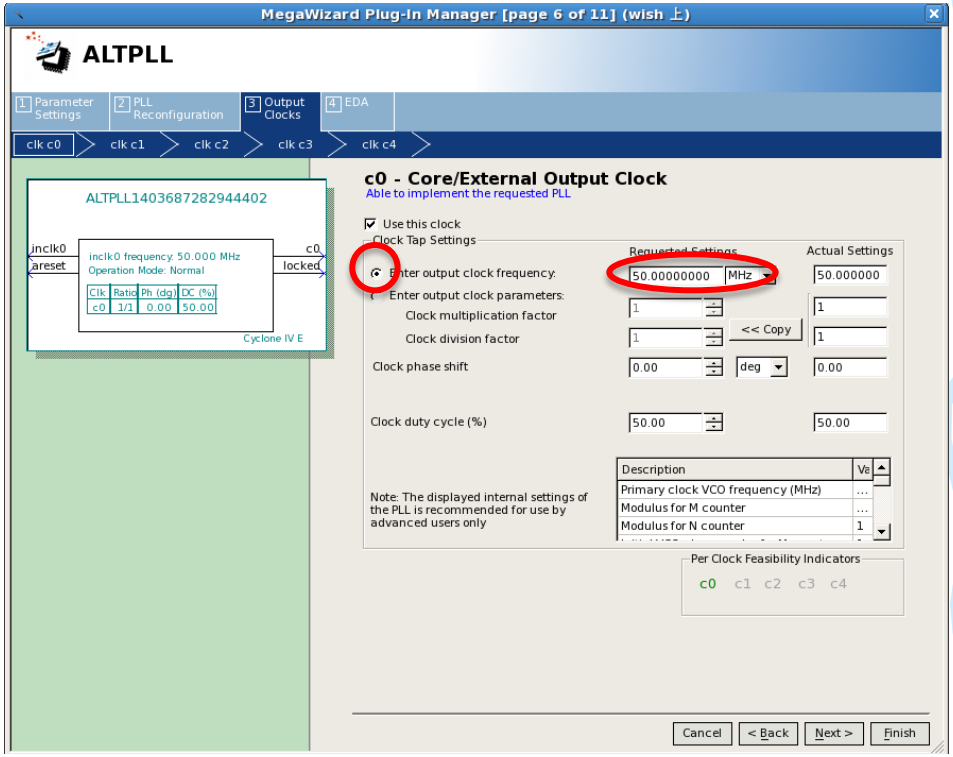
PLLの追加

- Library から PLL -> Avalon PLL を選択し，Add
- 現れたWindowで以下を設定
  - Parameter Setting -> General/Mode内の入力クロックを50MHzに設定（次ページの画面1）
  - Output Clock -> clk c0のEnter output clock frequencyのラジオボタンを選択し，50MHzと入力（次ページの画面2）
  - Output Clock -> clk c1のEnter output clock frequencyのラジオボタンを選択し，50MHzと入力し，Clock phase shiftを-3nsとする．（2ページ先の画面3）
  - Finishボタンを押す
- 追加したPLLの名前をpllに変更
- PLLの配線を以下の用にする
  - pllのclk_in_primaryとclk_0のclkを接続
  - pllのclk_in_primary_resetとclk_0のclk_resetを接続
- PLLのc1をexport欄をダブルクリックしてexportし，export名をsdram_clkとする
- PLLのlocked_conduitのexport欄をダブルクリックしてexportし，export名をlockedとする（2ページ先の画面4）

PLLの追加



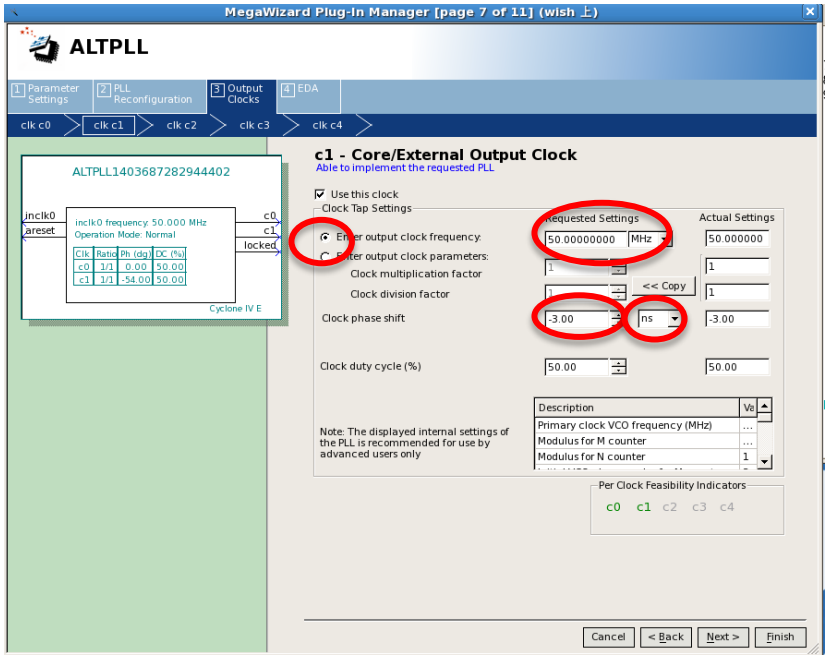
画面1



画面2

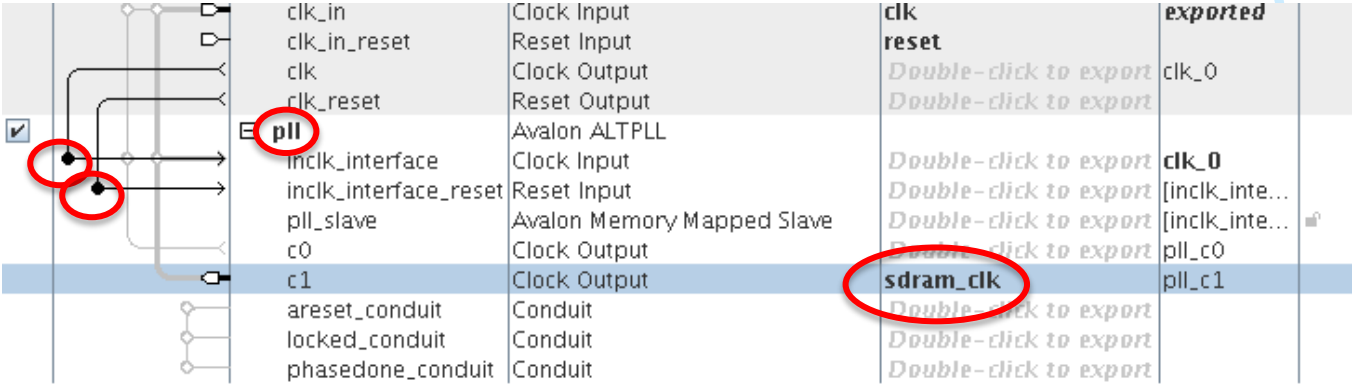
ここの欄が現在設定している
項目を示すタブになっている

PLLの追加



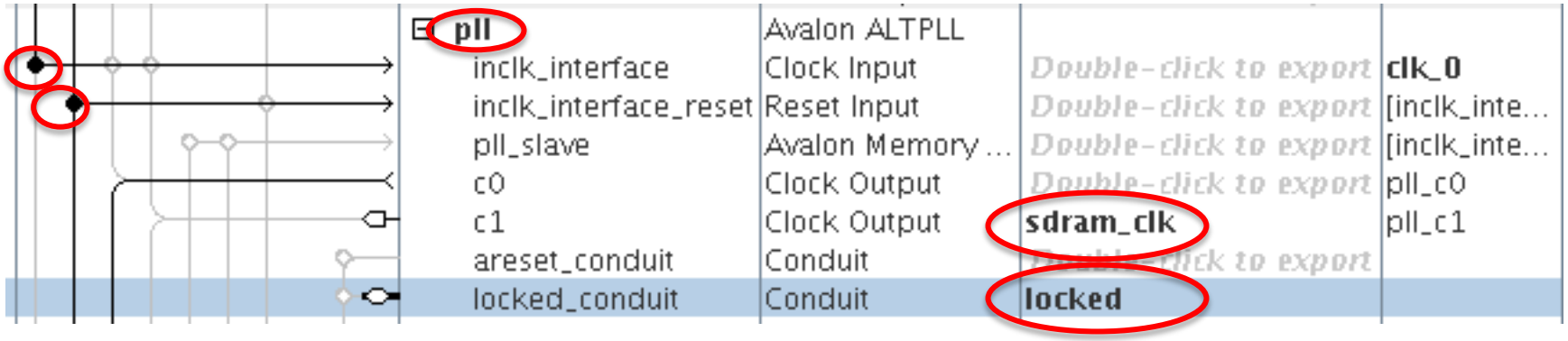
画面3

画面4



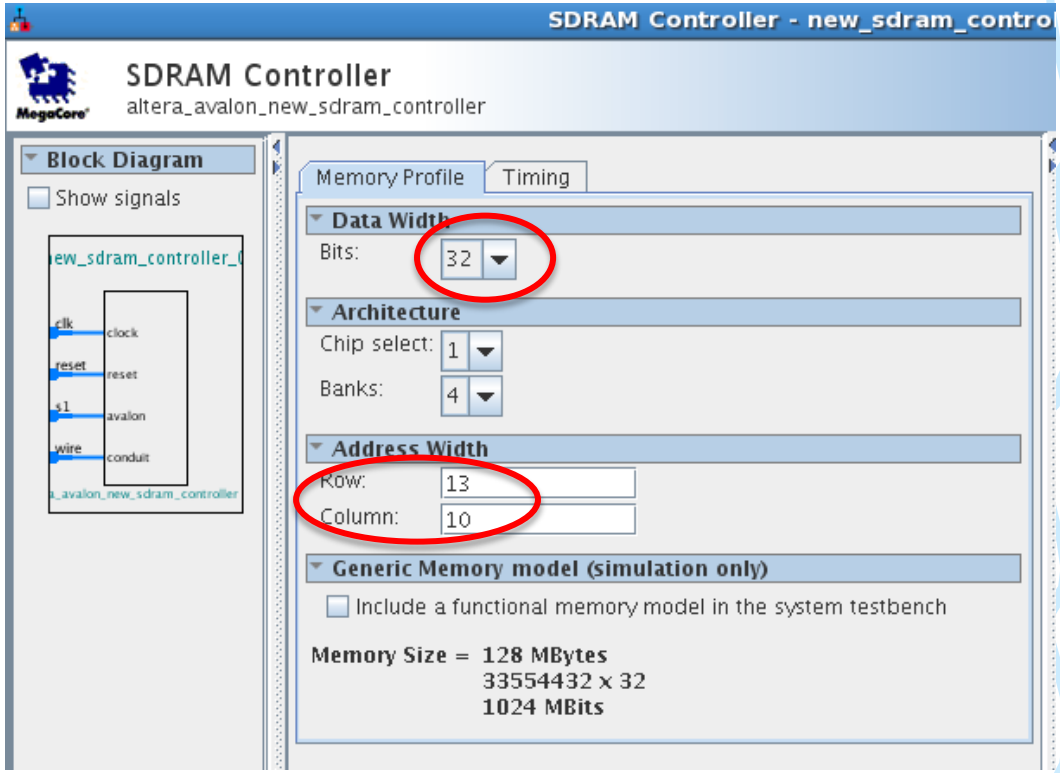
PLLの設定

- 追加したPLLの名前をpllに変更
- PLLの配線を以下の用にする
  - pllのclk_in_primaryとclk_0のclkを接続
  - pllのclk_in_primary_resetとclk_0のclk_resetを接続
- PLLのc1をexport欄をダブルクリックしてexportし，export名をsdram_clkとする
- PLLのlocked_conduitのexport欄をダブルクリックしてexportし，export名をlockedとする



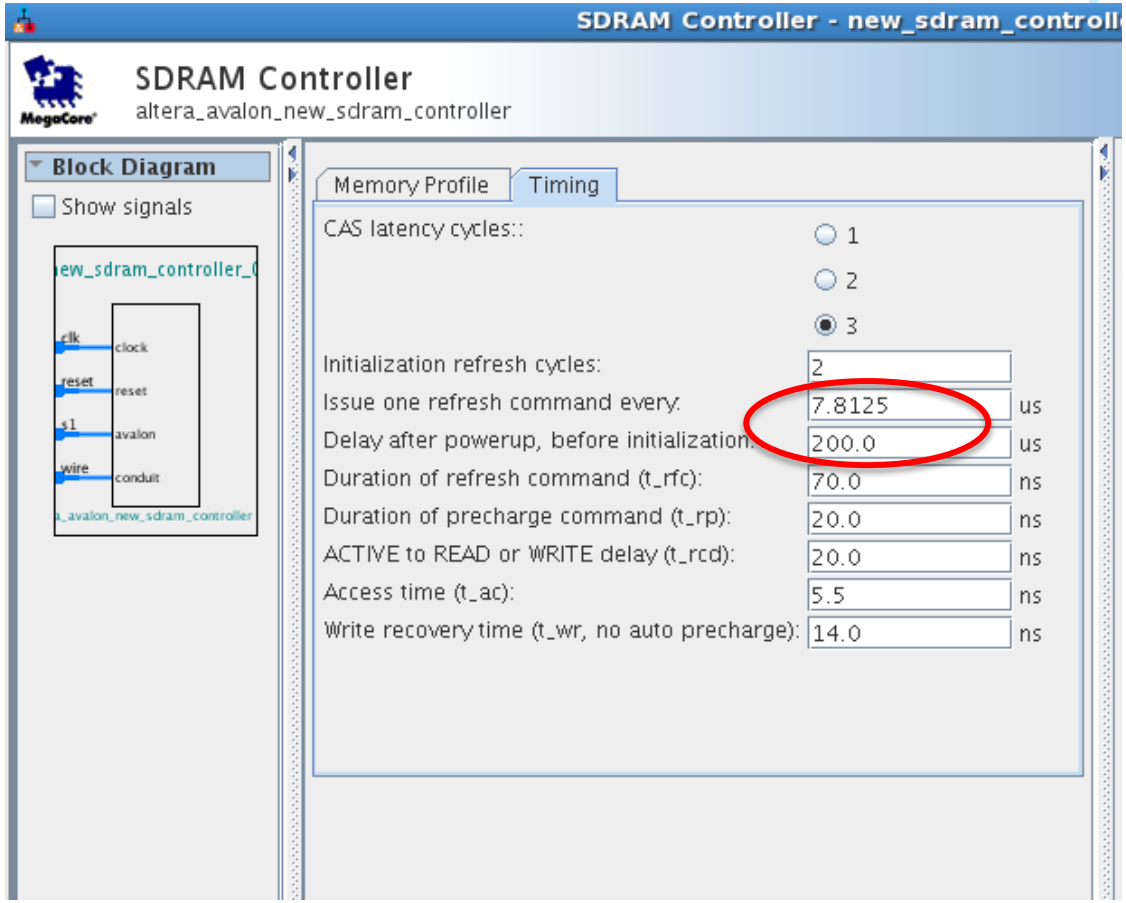
SDRAM Controllerの追加

- Library から Memories and Memory Controllers->External Memory Interfaces-> SDRAM Interfaces -> SDRAM Controllerを選択し，Add
- 現れたWindowで以下を設定する
  - Data Width: 32
  - Address Width
    - Row: 13，Colum: 10



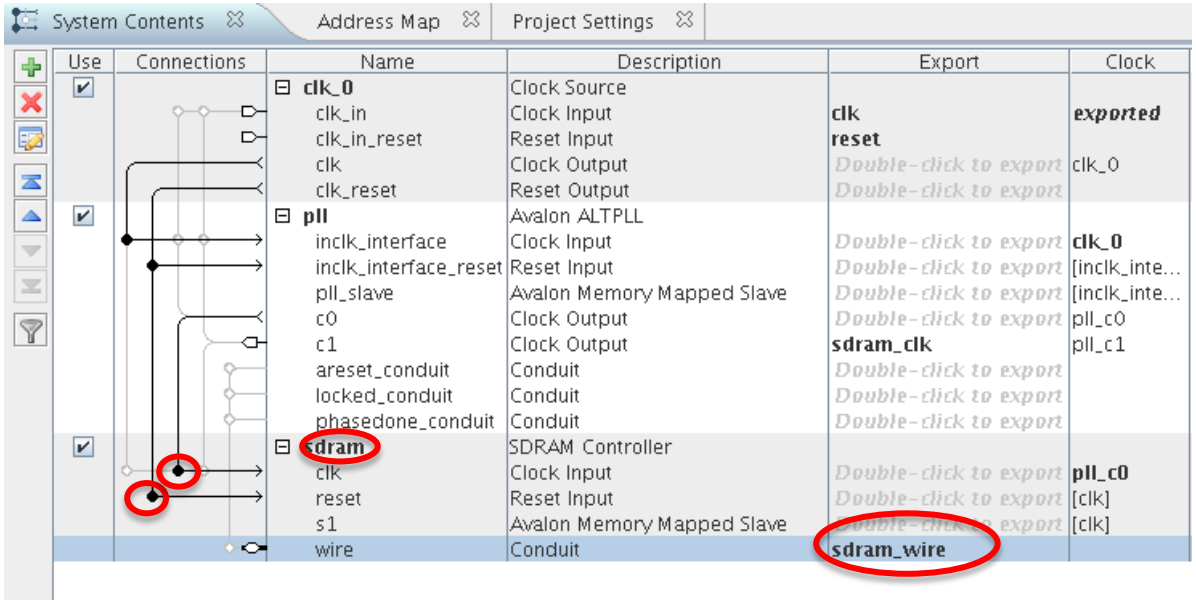
SDRAM ControllerのTiming設定

- Timingタブをクリック
  - Issue one refresh command every を7.8125us
  - Delay after powerupを200us
- Finishボタンをクリック



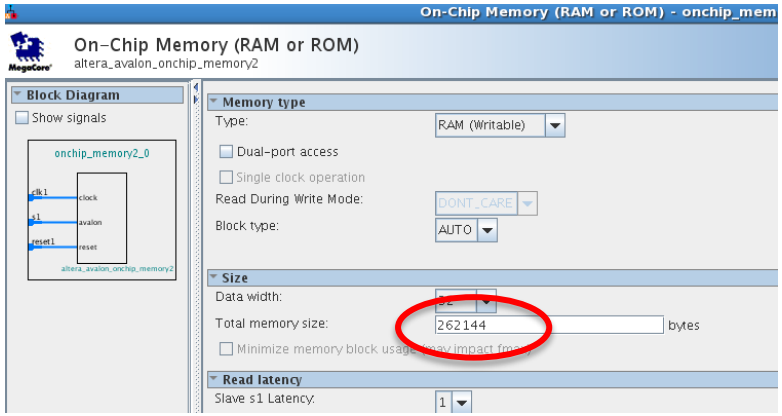
SDRAM Controllerのclk配線など

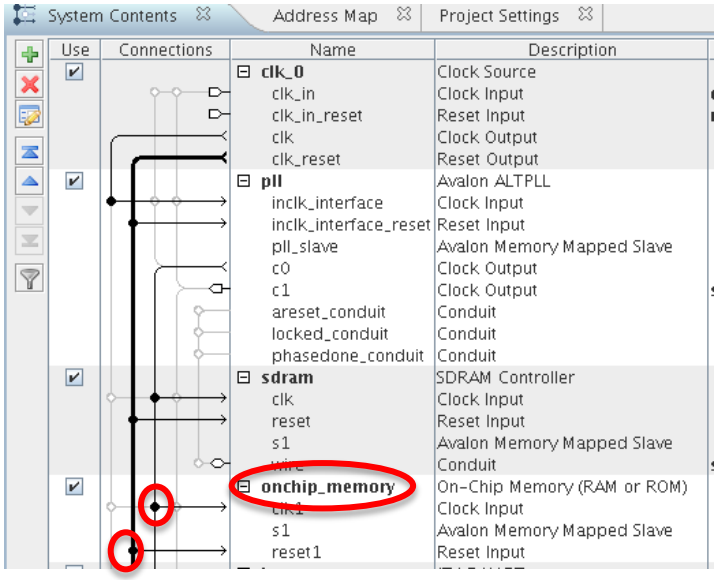
- 追加したsdram controllerの名前をsdramに変更（new_sdram_controller_0という名前の上で右クリックを押して現れるメニューからRenameを選択）
- pllのc0をsdram controllerのclkに配線
- clk_0のclk_resetをsdramのresetに配線
- sdramのwireをExportするようにExport欄をダブルクリック．（Export名をsdram_wireとする）



On-Chip memoryの追加

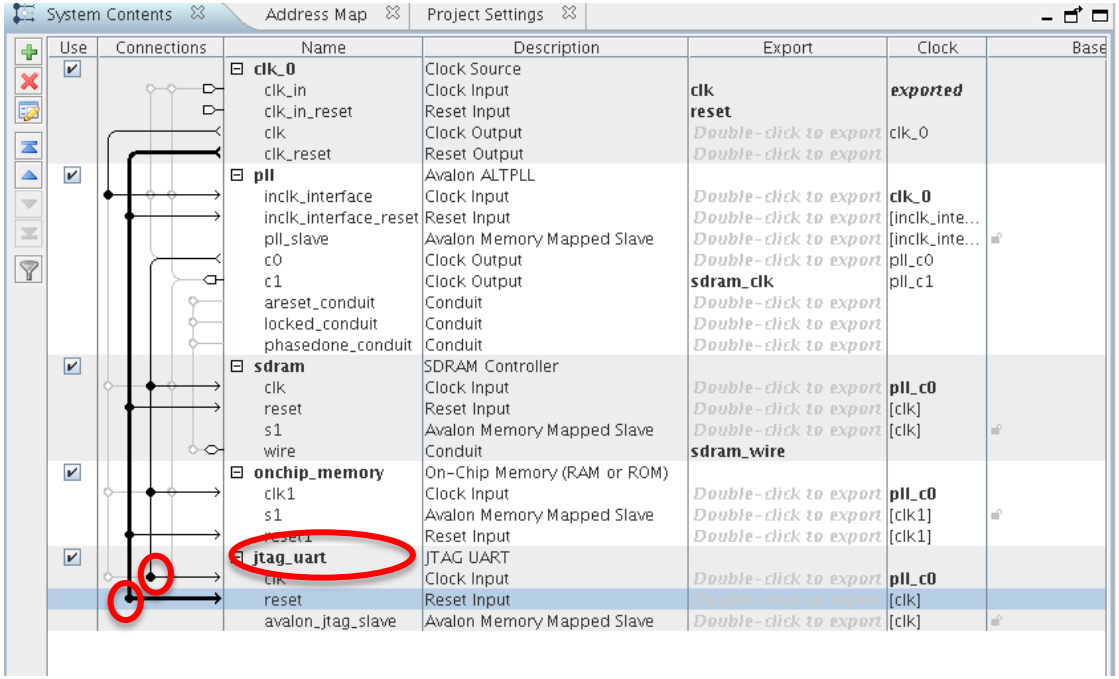
- Library から Memories and Memory Controllers-> On-Chip -> On-Chip Memory(RAM or ROM)を選択し，Add
  - Total Memory Sizeを262144とする(256k)
  - Finishをクリックする
- 名前をonchip_memoryに変更
- onchip_memoryのclk1をpllのc0と接続する
- onchip_memoryのreset1をclk_0のclk_resetと接続する





JTAG UARTの追加

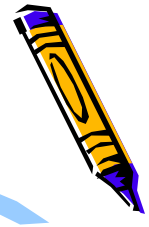
- Library から Interface Protocols-> Serial -> JTAG UARTを選択し，Add
  - Finishをクリックする
- 名前をjtag_uartに変更
- jtag_uartのclkをpllのc0と接続する
- jtag_uartのresetをclk_0のclk_resetと接続する



contest uart avalon interfaceをComponentとして登録

- ProjectのNew Componentを選択して，Add
- Component Typeタブ内
  - Name, Display Nameを"contest_uart_avalon_interface"とする
  - Groupを"My Own IP Core"とする
- Filesタブ内
  - （プロジェクトディレクトリ内にリファレンスデザインのプロジェクトディレクトリにある以下のファイルをコピーしておく
    - contest_uart_avalon_interface.v
    - system.v
    - define.v
  - Synthesis Filesとして，define.v以外の上記2つのVerilog HDLファイルを追加する（contest_uart_avalon_interface.vがTop-level Fileとなっているのを確認）
  - Analyze Synthesis Filesボタンをクリック

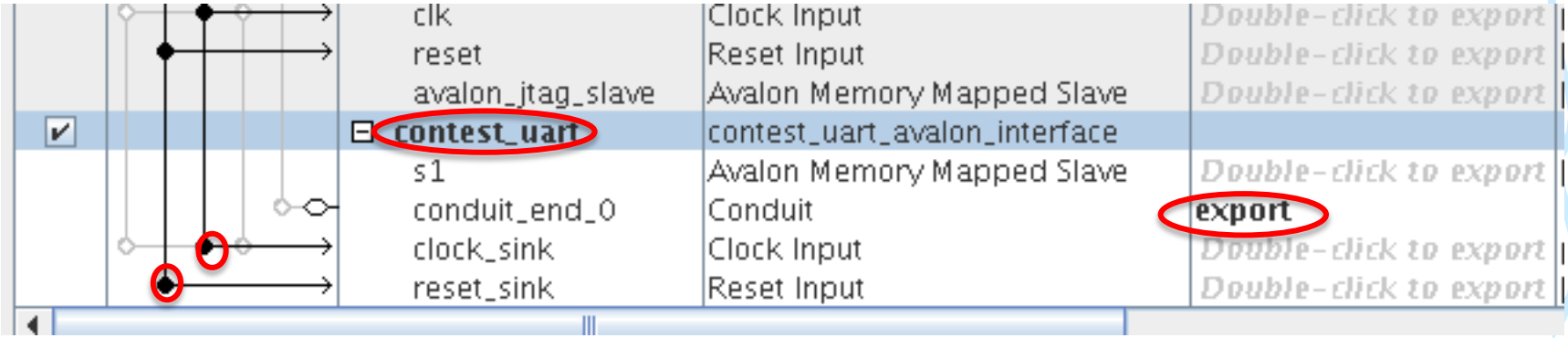
contest uart avalon interfaceをComponentとして登録



- Signalsタブ内
  - clockのinterfaceの所を選択して，new Clock Inputを選択し，interface欄がclock_sinkとなるようにし，Signal Typeをclkとする
  - resetのinterfaceの所を選択して，new Reset Inputを選択し，interface欄がreset_sinkとなるようにする.
- Interfaceタブ内
  - Remove Interfaces With No Signalsボタンを押す
  - s1のAssociated Clockをclock_sinkに，Associated Resetをreset_sinkにする
  - conduit_end_0のAssociated Clock, Associated Resetも同様にする
  - reset_sinkのAssociated Clockをclock_sinkにする
- Finishボタンを押し，保存するかどうかを訪ねるWindowが出たら，Saveを選ぶ.

contest uart avalon interfaceを追加

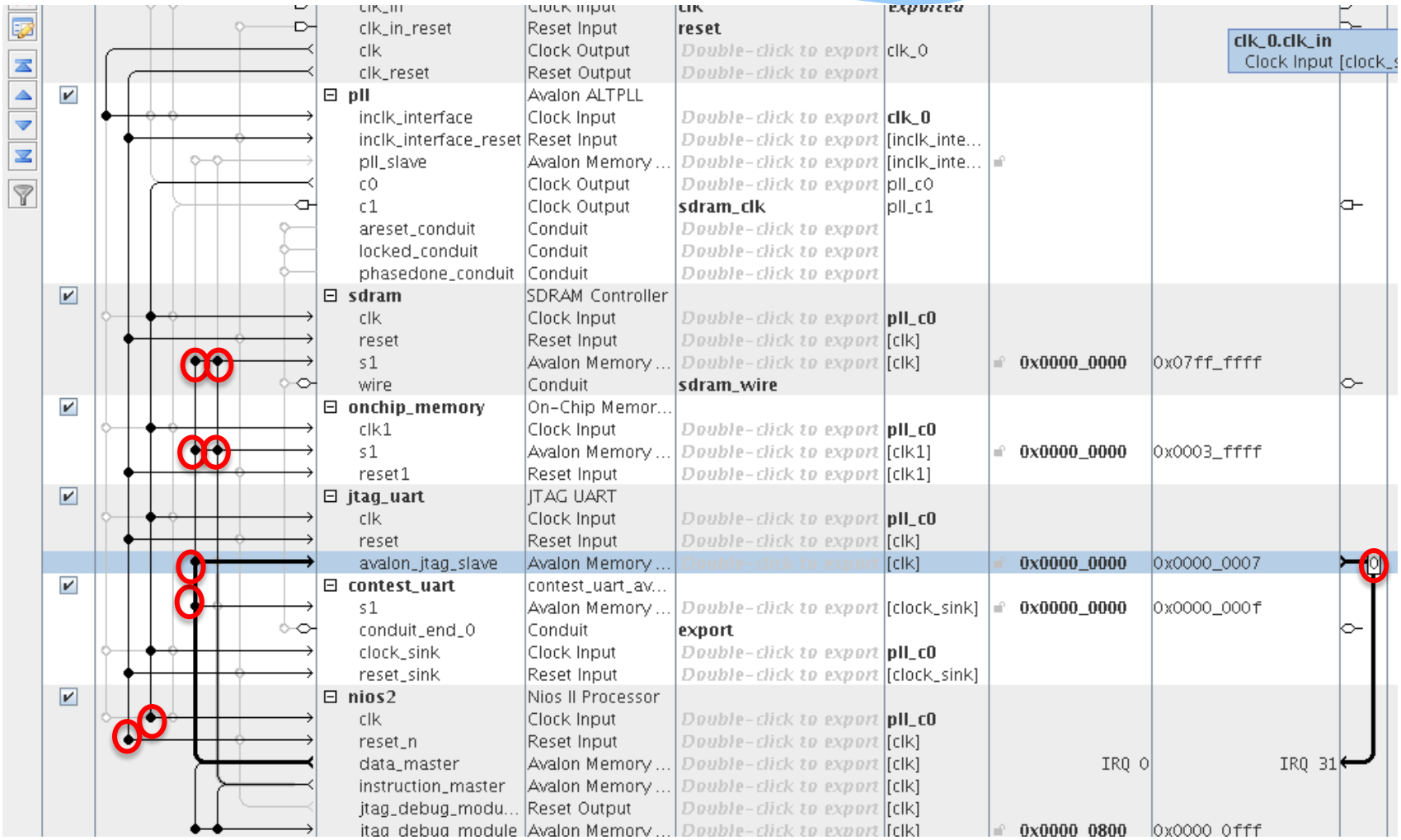
- ProjectのMy Own IP Core内のcontest_uart_avalon_interfaceを選択してAdd
  - Finishボタンをクリック
- 名前をcontest_uartに変更
- contest_uart_avalon_interfaceのclock_sinkをpllのc0と接続
- contest_uart_avalon_interfaceのreset_sinkをclk_0のclk_resetと接続
- contest_uart_avalon_interfaceのconduit_end_0のexport欄をダブルクリックして，export名をexportとする



Nios II/eの追加

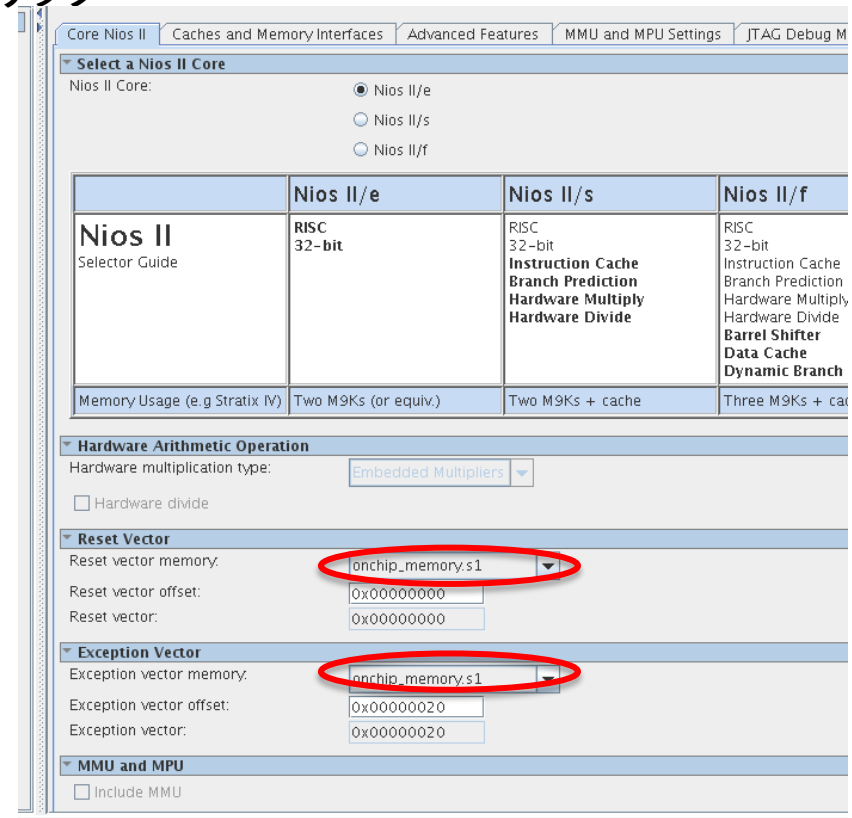
- Library から Embedded Processors-> Nios II Processorを選択し，Add
  - Nios II CoreとしてNios II/eを選択する
  - Finishをクリックする
- 名前をnios2に変更
- nios2のclkをpllのc0と接続する
- nios2のreset_nをclk_0のclk_resetと接続する
- nios2のdata_masterをcontest_uartのs1，jtag_uartのavalon_jtag_slave，onchip_memoryのs1，sdramのs1と接続する
- nios2のinstruction_masterをonchip_memoryのs1，sdramのs1と接続する
- nios2のIRQとjtag_uartのIRQを接続する

NiosII/eの追加


clk_in_reset Reset Input reset
clk Clock Output Double-click to export clk_0
clk_reset Reset Output Double-click to export
clk_0.clk_in
Clock Input
pll Avalon ALTPLL
inclk_interface Clock Input Double-click to export clk_0
inclk_interface_reset Reset Input Double-click to export
pll_slave Avalon Memory ... Double-click to export
c0 Clock Output Double-click to export pll_c0
c1 Clock Output sdram_clk pll_c1
areset_conduit Conduit Double-click to export
locked_conduit Conduit Double-click to export
phasedone_conduit Conduit Double-click to export
sdram SDRAM Controller
clk Clock Input Double-click to export pll_c0
reset Reset Input Double-click to export
s1 Avalon Memory ... Double-click to export 0x0000_0000 0x07ff_ffff
wire Conduit sdram_wire
onchip_memory On-Chip Memor...
clk1 Clock Input Double-click to export pll_c0
s1 Avalon Memory ... Double-click to export 0x0000_0000 0x0003_ffff
reset1 Reset Input Double-click to export
jtag_uart JTAG UART
clk Clock Input Double-click to export pll_c0
reset Reset Input Double-click to export
avalon_jtag_slave Avalon Memory ... Double-click to export 0x0000_0000 0x0000_0007
contest_uart contest_uart_av...
s1 Avalon Memory ... Double-click to export 0x0000_0000 0x0000_000f
conduit_end_0 Conduit export
clock_sink Clock Input Double-click to export pll_c0
reset_sink Reset Input Double-click to export
nios2 Nios II Processor
clk Clock Input Double-click to export pll_c0
reset_n Reset Input Double-click to export
data_master Avalon Memory ... Double-click to export IRQ 0 IRQ 31
instruction_master Avalon Memory ... Double-click to export
jtag_debug_modu... Reset Output Double-click to export
jtag_debug_module Avalon Memory ... Double-click to export 0x0000_0800 0x0000_0fff

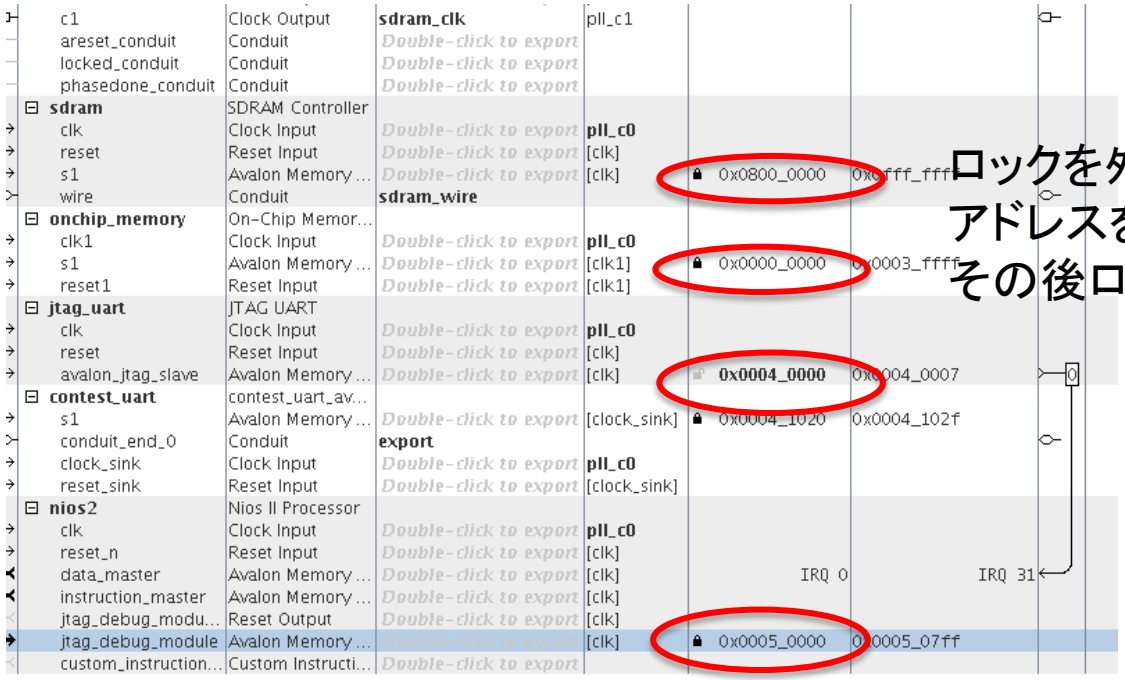
nios2の設定

- nios2を選択して，右クリックし，Editを選択する
- Reset vector memoryとException vector memoryとして，onchip_memory.s1を選択する
- Finishボタンをクリック



メモリマップの作成

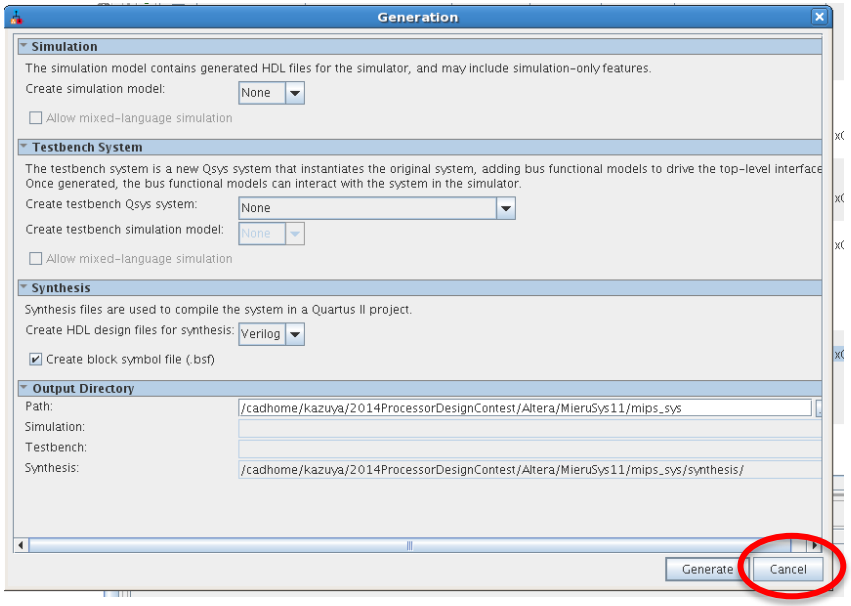
- メモリマップとしてbase addressを以下のように指定して，ロック
  - sdramのmemory: 0x0800_0000
  - onchip_memory:0x0000_0000
  - contest_uart: 0x0004_1020
  - nios2: 0x0005_0000
- jtag_uartはどこに割り当ててもよいので，System->Assign Base Addressで割り当てる



ロックを外した状態で
アドレスを入力し，
その後ロックをかける

nios2_sysの生成

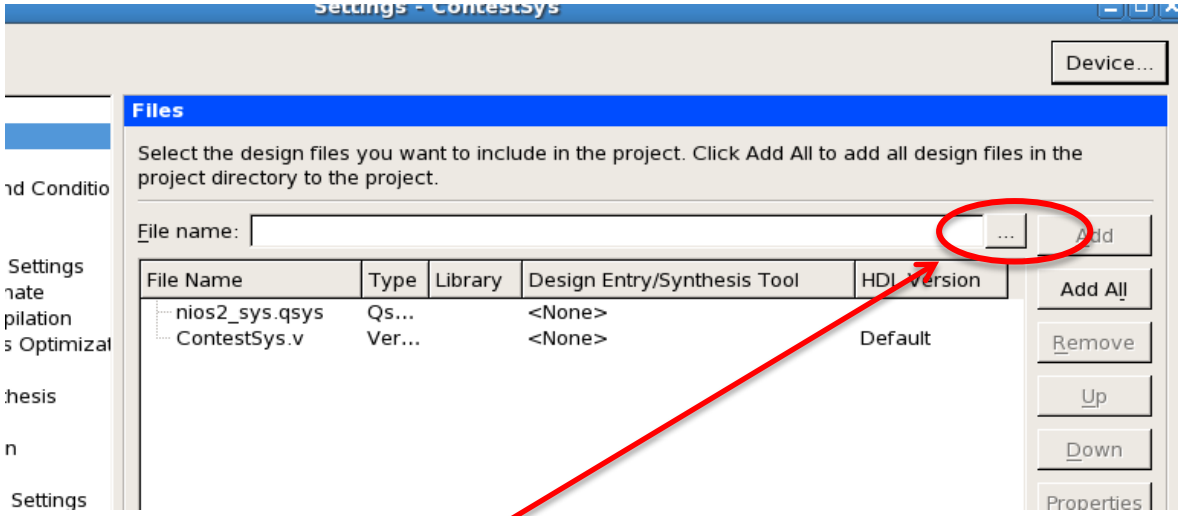
- メニューSystem -> Create Global Reset Networkをクリック
- この時点で，Qsys下部のMessageからエラー表示がなくなっているはず
- mips_sysを保存
- メニューGenerate -> Generateをクリック
  - Generateボタンをクリック
- ちなみに，mips_avalon_interface.vなどを変更する度に，QsysでGenerateする必要があります
- 以上で，Qsysは終了してOK



Generateでは
右下のGenerateボタンを
押すだけでよい

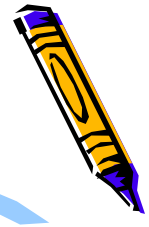
プロジェクトへのファイルの追加

- あらかじめ，ContestSys.vをプロジェクトディレクトリにコピーしておく
- Porject -> Add/Remove files in Projectで以下のファイルを追加する
  - ContestSys.v
  - nios2_sys.qsys
- 追加したらOKボタンを押してウィンドウを閉じる



ここをクリックして追加するファイルを選択し，その後addボタンを押す

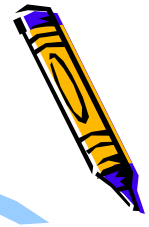
ピン配置情報のimportと割当



- Assignments -> Import Assignmentsをクリック
  - DE2_115.qsfを選択し，importする
- Assignments -> Pin Plannerをクリック
  - 現れるウィンドウの下の方にある各ピンの設定で，GPIO[3]のI/O Standardを3.3V LVCMOSにする
  - 同様に GPIO[5], GPIO[7], GPIO[9]も同様に変更する
  - メニューFile -> closeをクリックして，ウィンドウを閉じる

Node Name	Location	I/O Standard
GPIO[3]	PIN_Y17	3.3V LVCMOS
GPIO[5]	PIN_Y16	3.3V LVCMOS
GPIO[7]	PIN_AE16	3.3V LVCMOS
GPIO[9]	PIN_AE15	3.3V LVCMOS

ピン配置情報のimportと割当

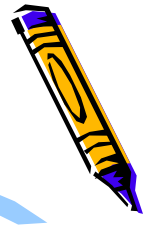


Edges

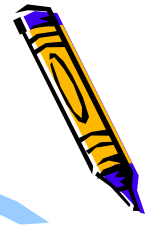
Named: *GPIO[?] Edit: Filter: Pin

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strengtl	Slew Rate	Differential Pair
GPIO[0]	Unknown	PIN_AB22	4	B4_N0	3.3-V LVTTL		8mA (default)		
GPIO[1]	Unknown	PIN_AC15	4	B4_N2	3.3-V LVTTL		8mA (default)		
GPIO[2]	Unknown	PIN_AB21	4	B4_N0	3.3-V LVTTL		8mA (default)		
GPIO[3]	Unknown	PIN_Y17	4	B4_N0	3.3-V LVCMOS		2mA (default)		
GPIO[4]	Unknown	PIN_AC21	4	B4_N0	3.3-V LVTTL		8mA (default)		
GPIO[5]	Unknown	PIN_Y16	4	B4_N0	3.3-V LVCMOS		2mA (default)		
GPIO[6]	Unknown	PIN_AD21	4	B4_N0	3.3-V LVTTL		8mA (default)		
GPIO[7]	Unknown	PIN_AE16	4	B4_N2	3.3-V LVCMOS		2mA (default)		
GPIO[8]	Unknown	PIN_AD15	4	B4_N2	3.3-V LVTTL		8mA (default)		
GPIO[9]	Unknown	PIN_AE15	4	B4_N2	3.3-V LVCMOS		2mA (default)		
<<new node>>									

構成情報の生成

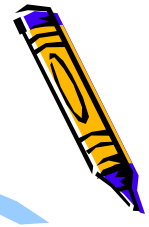


- Processing -> Start Compilationをクリックして，論理合成＆配置配線
- DE2-115ボードの電源をいれる．
- Taskウィンドウ内のProgram Device(Open Programmer)をダブルクリック
- Programmer内にて，Hardware Setupボタンをクリック．
  - No Hardwareとなっている所をUSB-Blaster USBを選択して，closeボタンをクリック
- Start ボタンを押して，Progress が100%(Successful)になればOK
- Programmerを閉じる

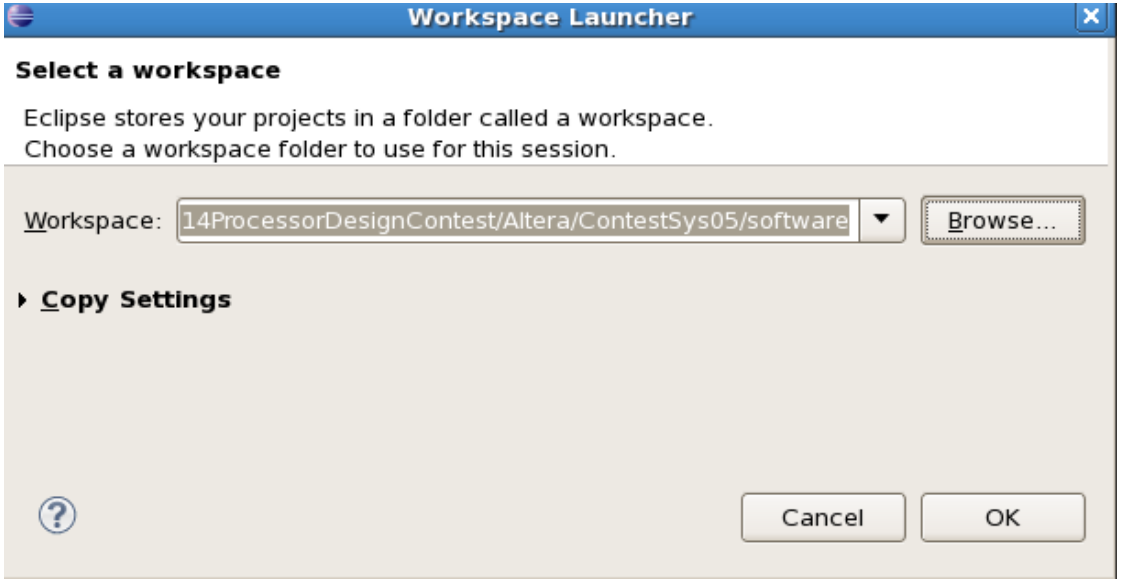


ソフトウェアデザイン

Workspaceの設定

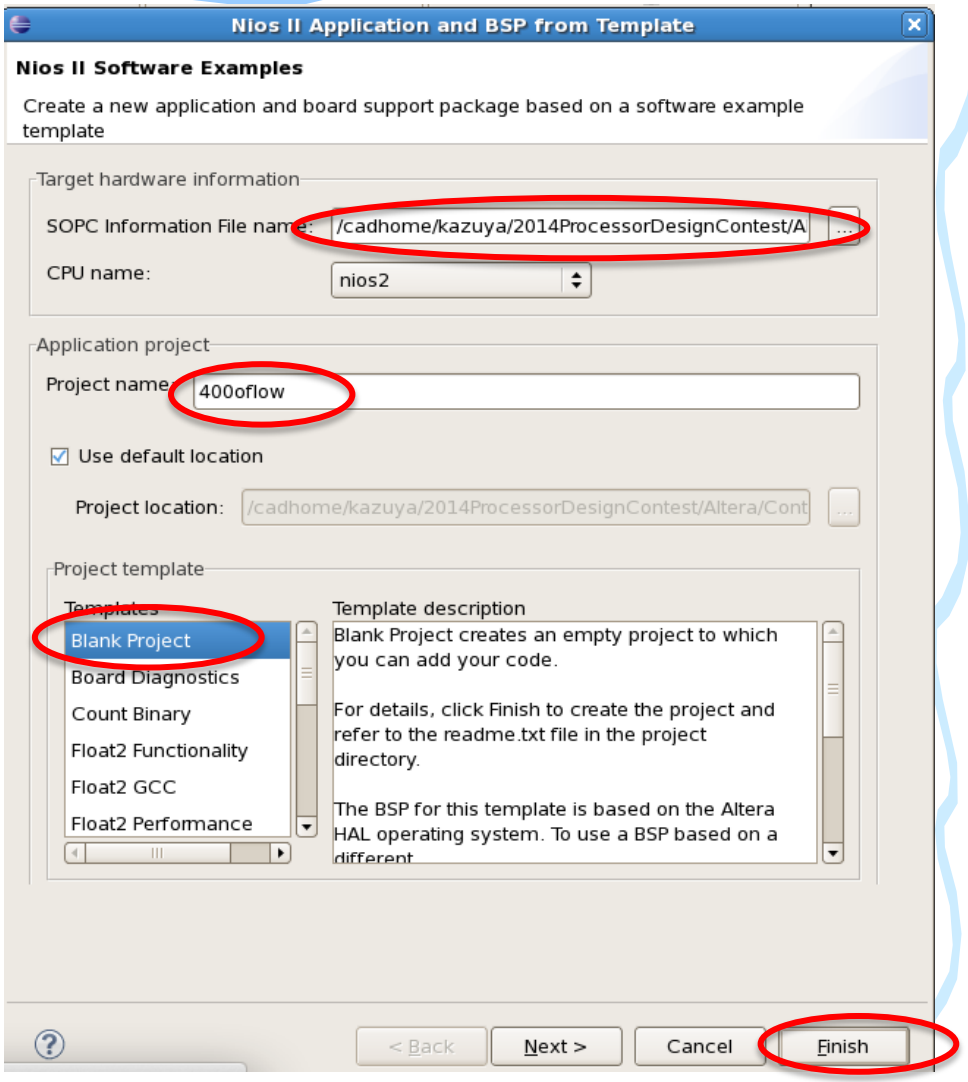


- ハードウェアのプロジェクトディレクトリの下にsoftwareというディレクトリを作る．（例：ContestSys05/software）
- QuartusのメニューTools->Nios II Software Build Tools for Eclipseを選択する
- EclipseのメニューFile -> Switch Workspace -> Otherを選択し，上記で作成したContestSys05/softwareを選択する．（初めてEclipseを立ち上げた場合は，いきなりWorkspaceの選択から入るかもしれない）



新規プロジェクトの作成

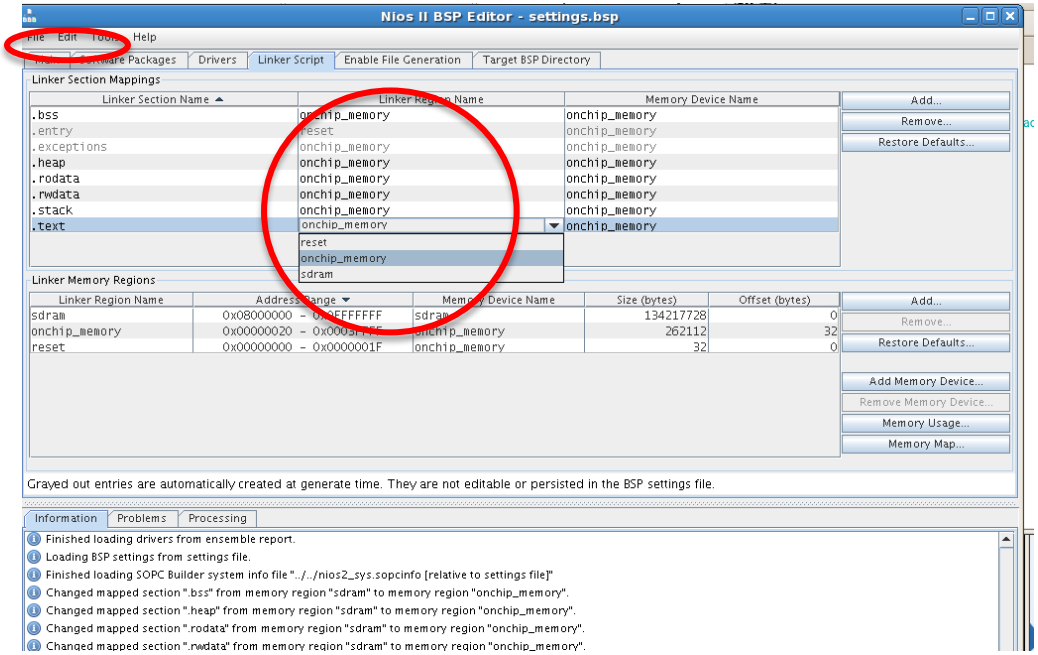
- File -> New -> Nios II Application and BSP from Templateを選択
- Target hardware informationのSOPC Information File nameには，Quartusのプロジェクトディレクトリにあるnios2_sys.sopcinfoを選択する
- Project nameを「400oflow」とする
- Project templateにはBlank Projectを選ぶ
- 以上で，Finishボタンを押す



BSPの設定

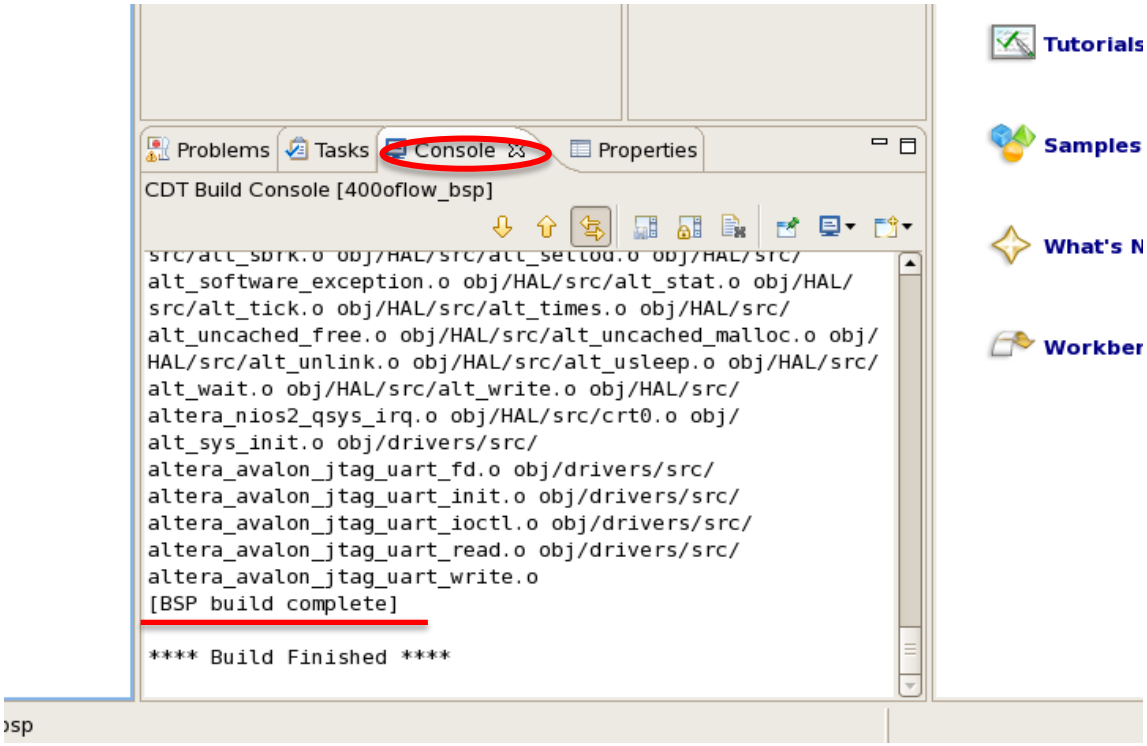
- 400oflow_bspを選択した状態で，右クリックし現れるメニューで，Nios II -> BSP Editorを選択
- 現れるWindowの中のLinker Scriptを選択し，".bss", ". heap", などを全てonchip_memoryに置くように設定する
- Nios II BSP EditorのFile メニューの中からSaveを選択する
- Generateボタンを押し，Exitボタンを押す

BSPを保存するときのFileメニュー



BSPのBuild

- 400oflow_bspを選択した状態で，右クリックし現れるメニューで，Build Projectをクリック
- Consoleにて，“[BSP build complete]”が出力されている事を確認



400oflowのBuild

- ファイルの準備
  - コンテストのサイトから配布されている400oflow_v07.tar.gzを展開し，400_oflowディレクトリに移動し，そこでmake exportをする．
  - 上記により，export/altera/src/の下にnios用のソースファイルが生成される
  - software/400oflowディレクトリに移動し，export/altera/srcディレクトリにあるファイルを全てコピーする．
- Eclipse上で，400oflowプロジェクトを選択した状態で，右クリックし現れるメニューで，
  - Refreshをクリックする
  - Build projectをクリックする
  - Run As -> Nios II Hardwareをクリックする
- 後はホストPCから画像の転送プログラムなどを動かせばOK