

1. EDKの使い方 (初級編)

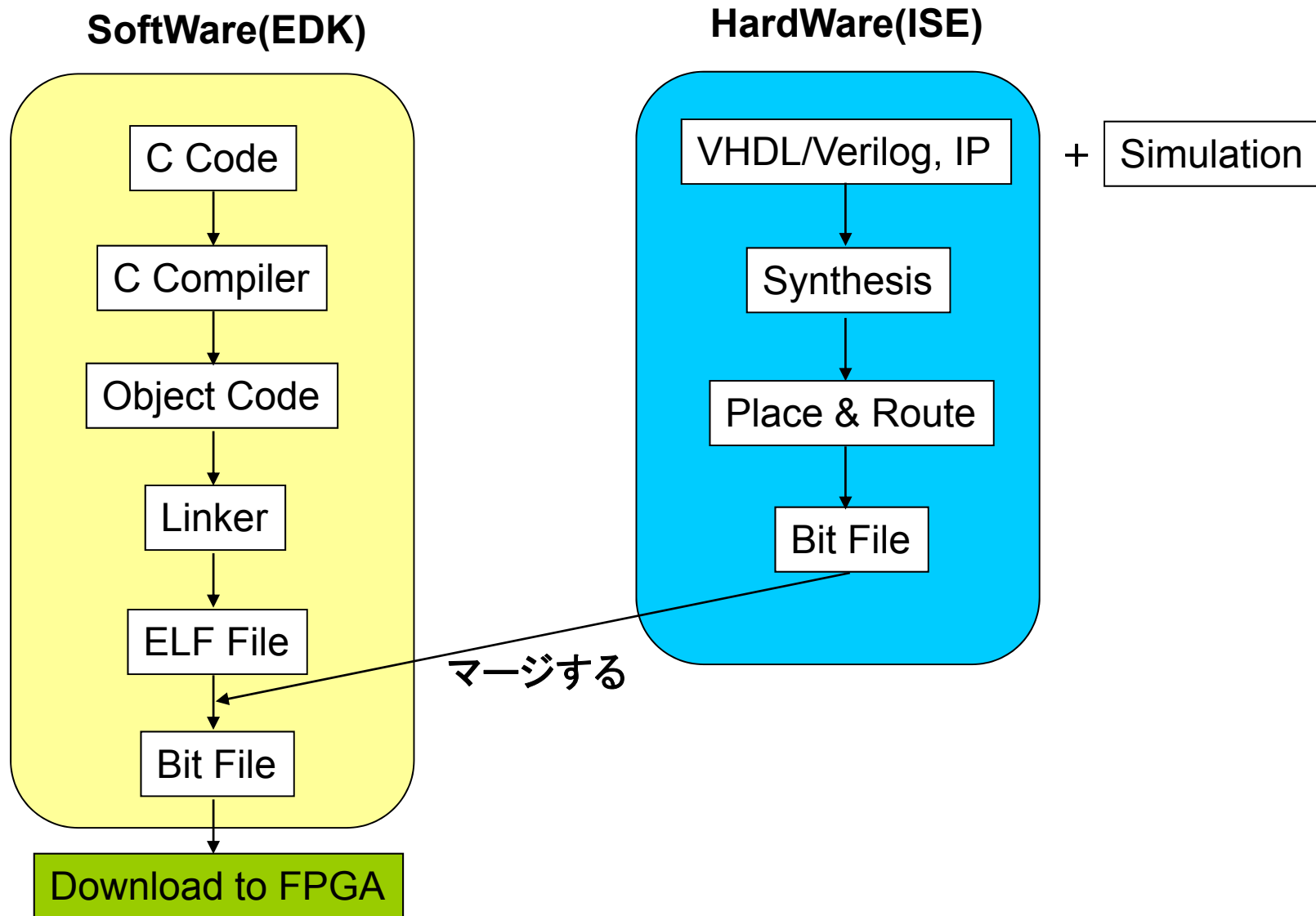
ザイリンクスPowerPC開発でよく使われる用語

- EDK ... Embedded Development Kit
- XPS ... Xilinx Platform Studio
- XMD ... Xilinx Microprocessor Debug
- MHS ... Microprocessor Hardware Specification

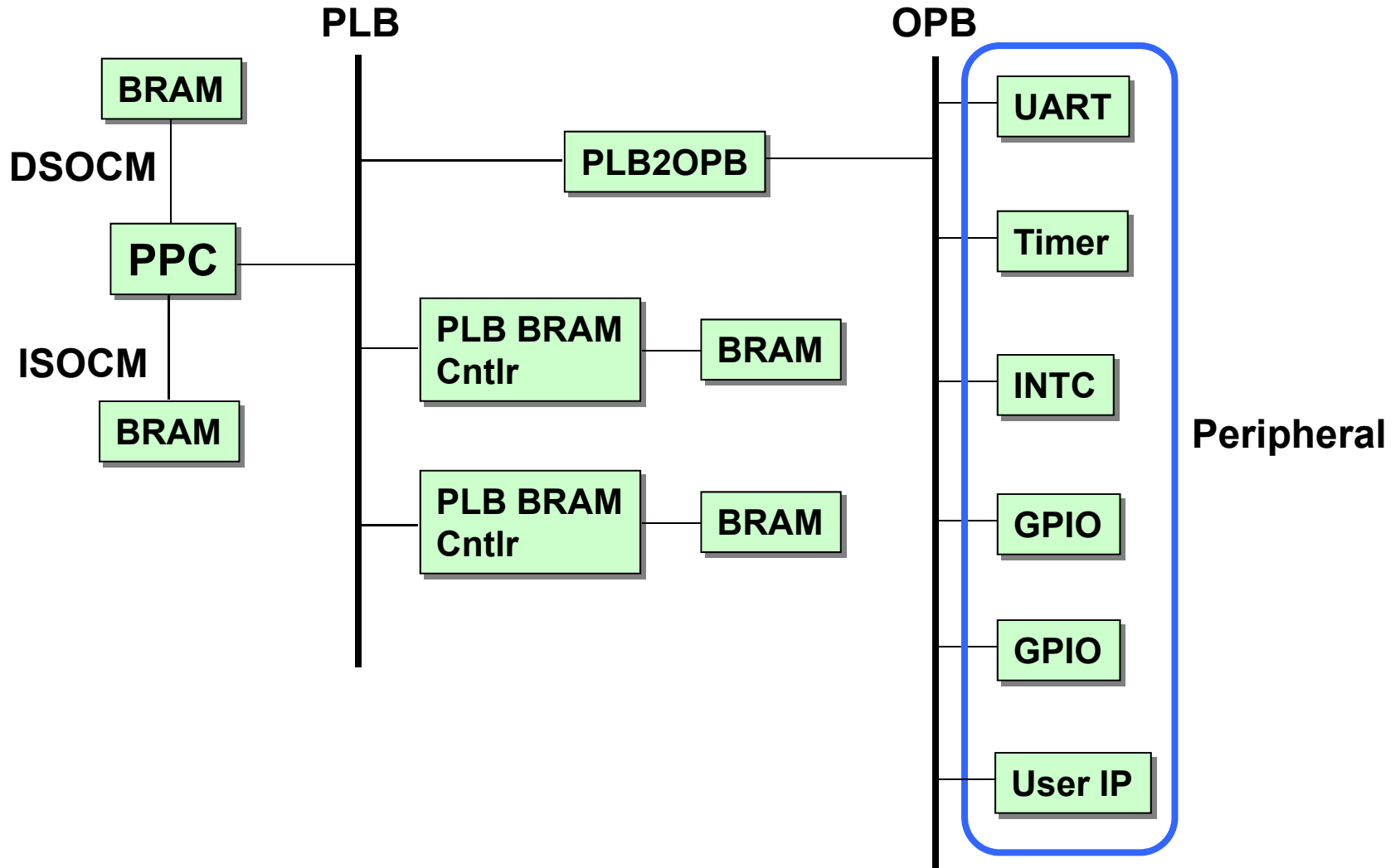
- PLB ... Processor Local Bus
- OPB ... On-chip Peripheral Bus
- OCM Bus ... On-Chip Memory Bus
- BRAM ... Block RAM

- ELF ... Executable and Link Format(実行ファイル)

開発フロー

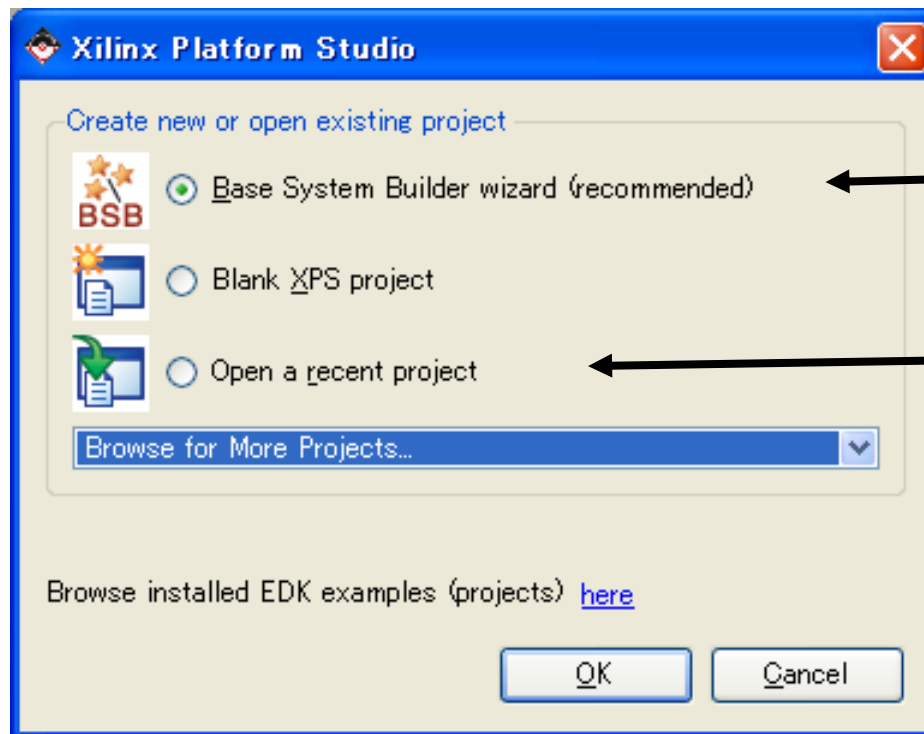


PowerPCシステムブロック図



BSB(Base System Builder)(1)

- Wizardを使って簡単にPowerPCシステムを構築
- 但し詳細な設定はできないので、通常はひな形作成程度に使用し、あとはユーザ側でシステム全体を構築するのが一般的だと思われる。

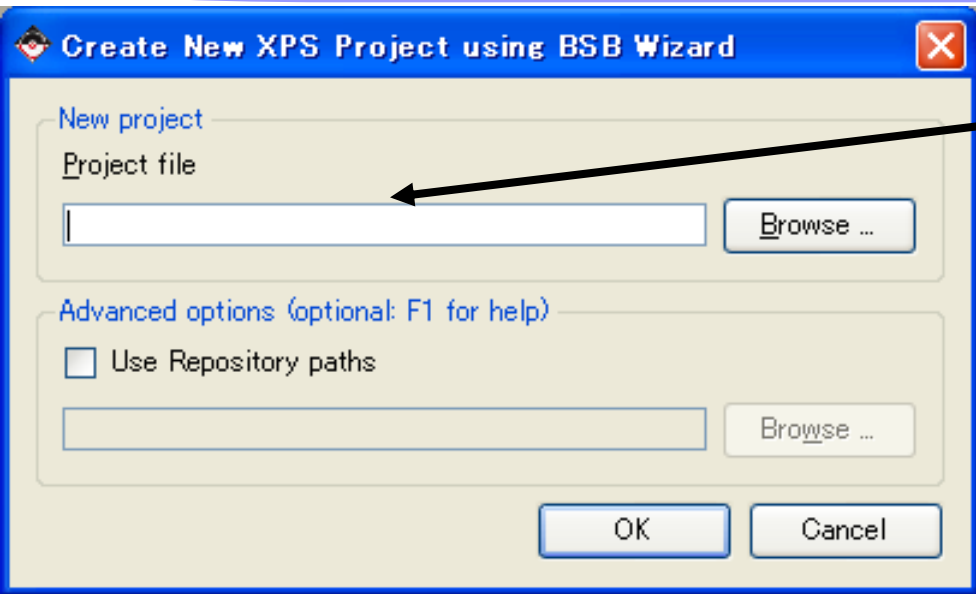


BSBを選択

既存のプロジェクトを開く時

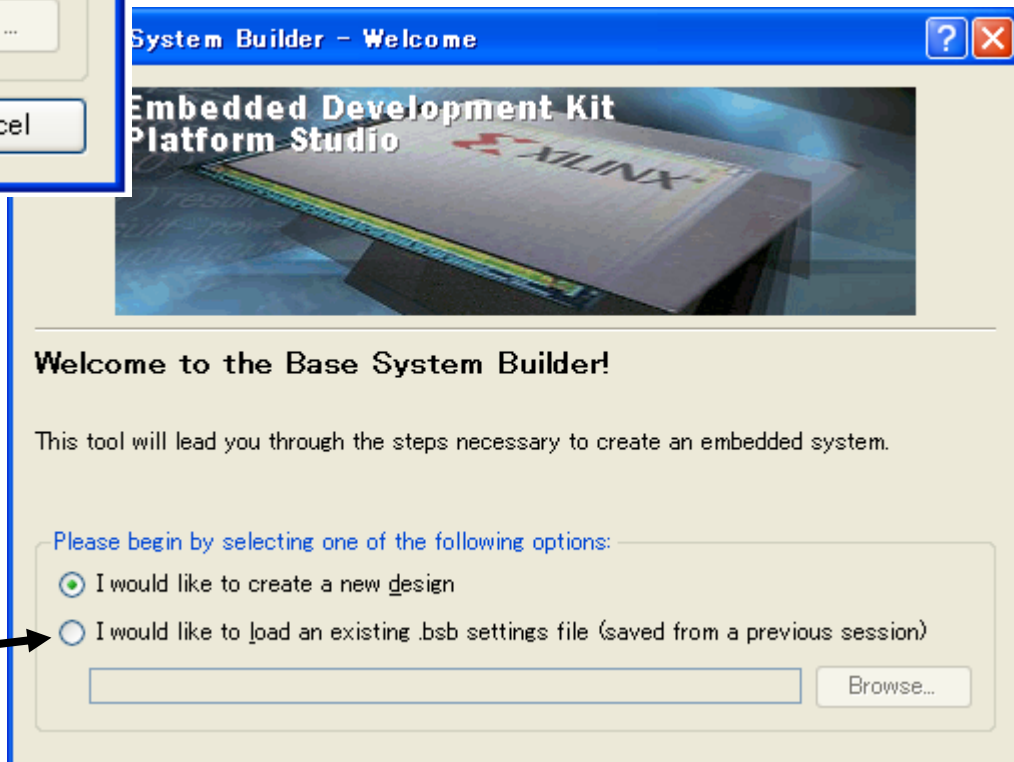
本講座では
ツールバージョン
EDK8.2i(ISE8.2i)
を対象としています。

BSB(Base System Builder)(2)

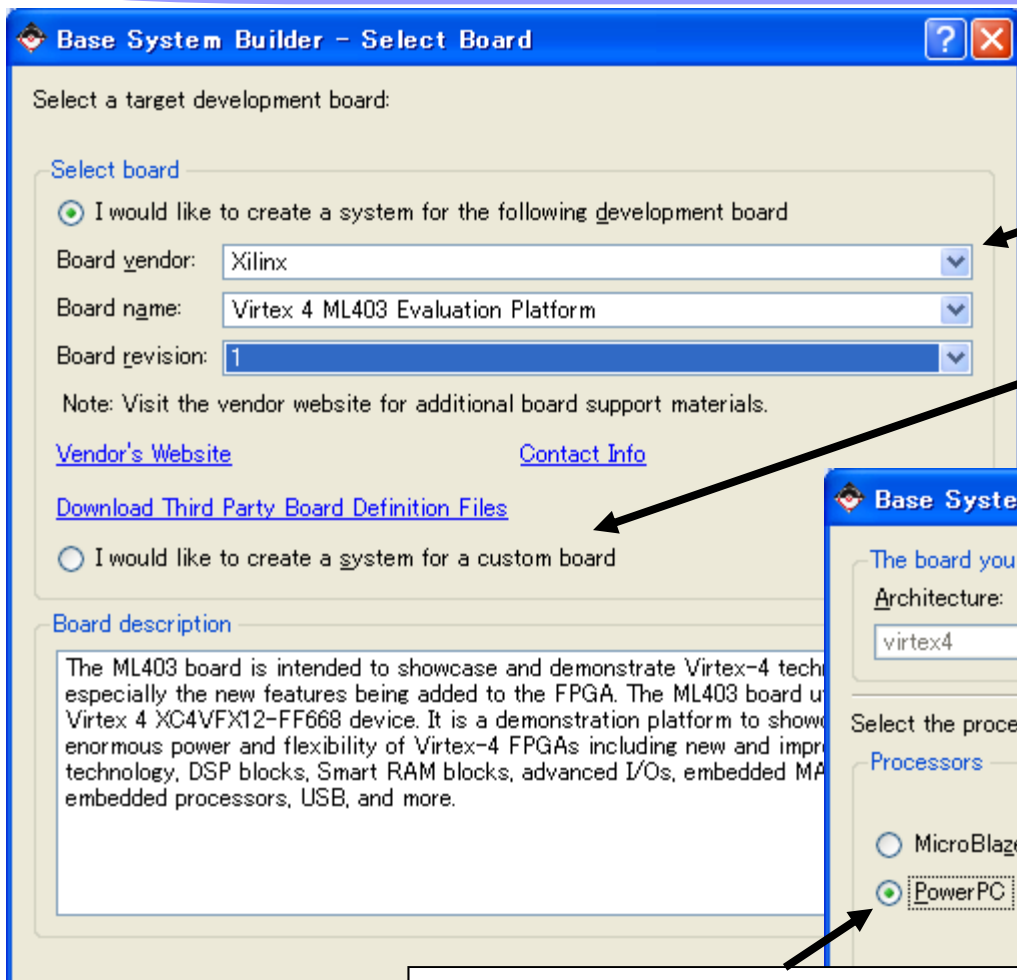


プロジェクトファイル名及びパス指定

使用するBSBファイルが
決まっていれば指定

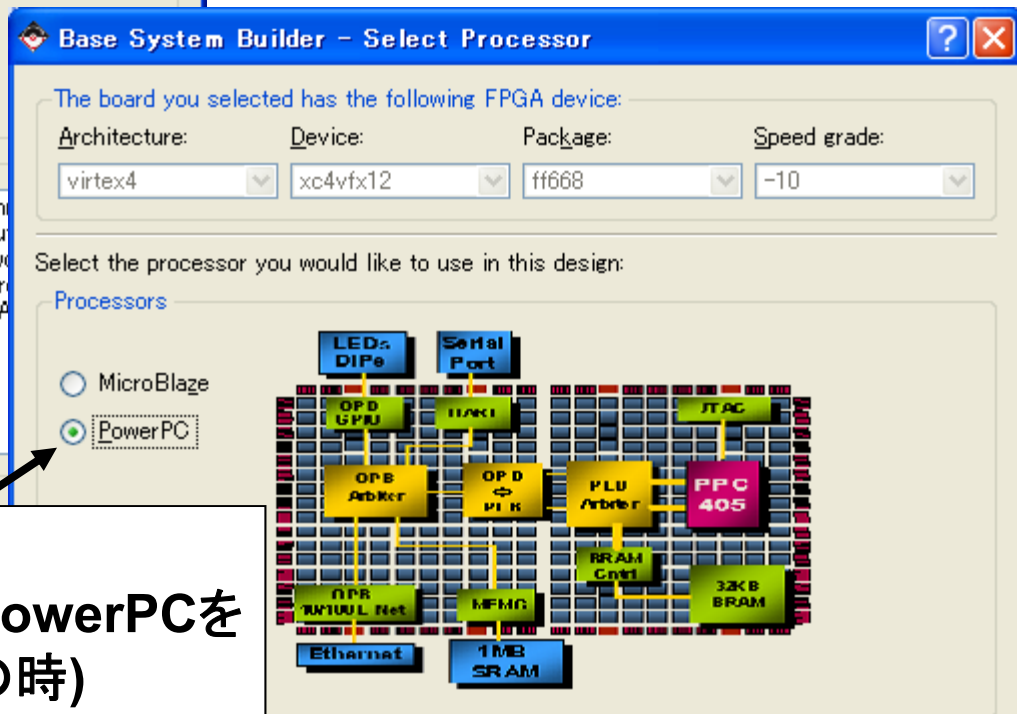


BSB(Base System Builder)(3)



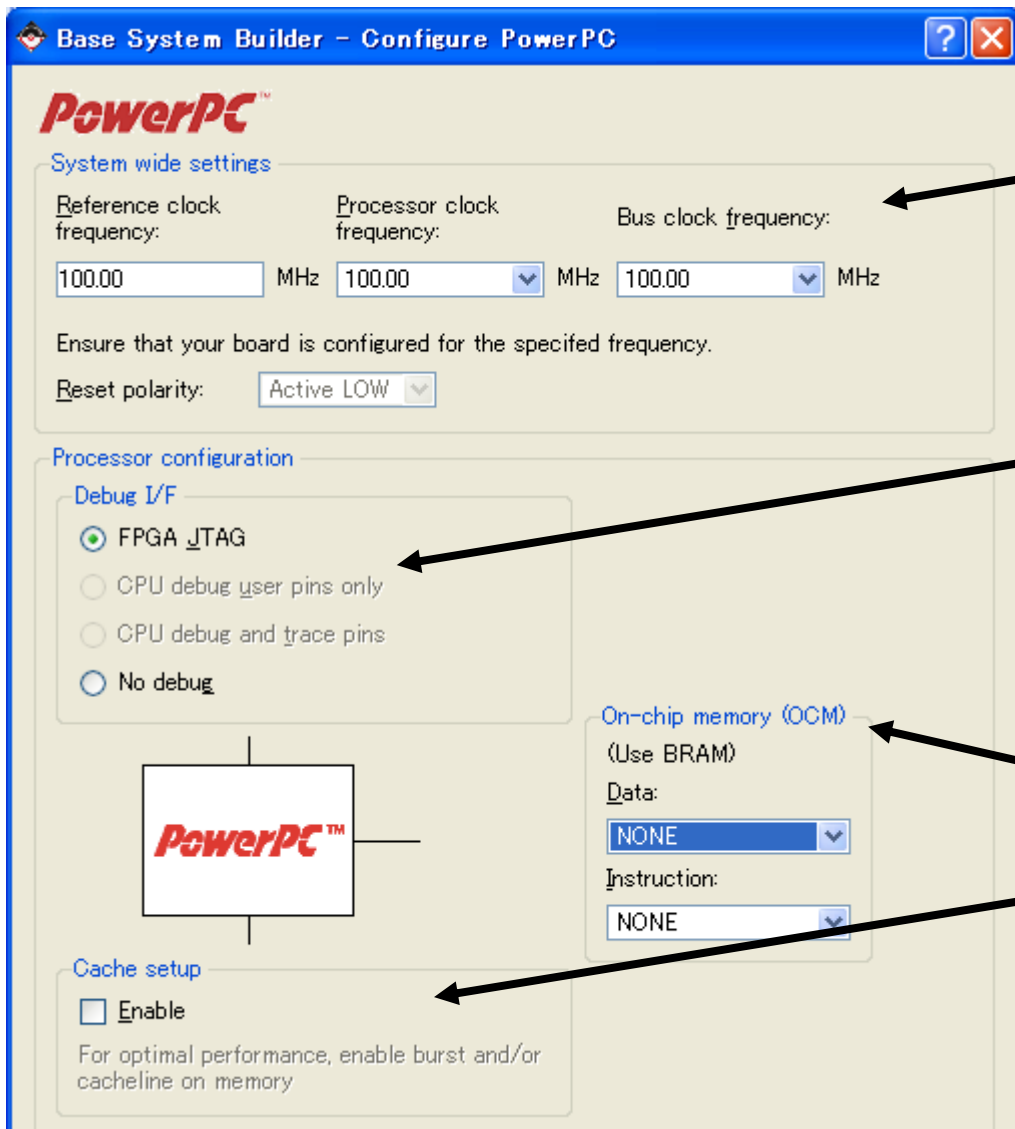
使用するボードを選択

一覧にない場合(カスタムボード)



PowerPCを選択
(MicroBlazeはPowerPCを
持たないFPGAの時)

BSB(Base System Builder)(4)

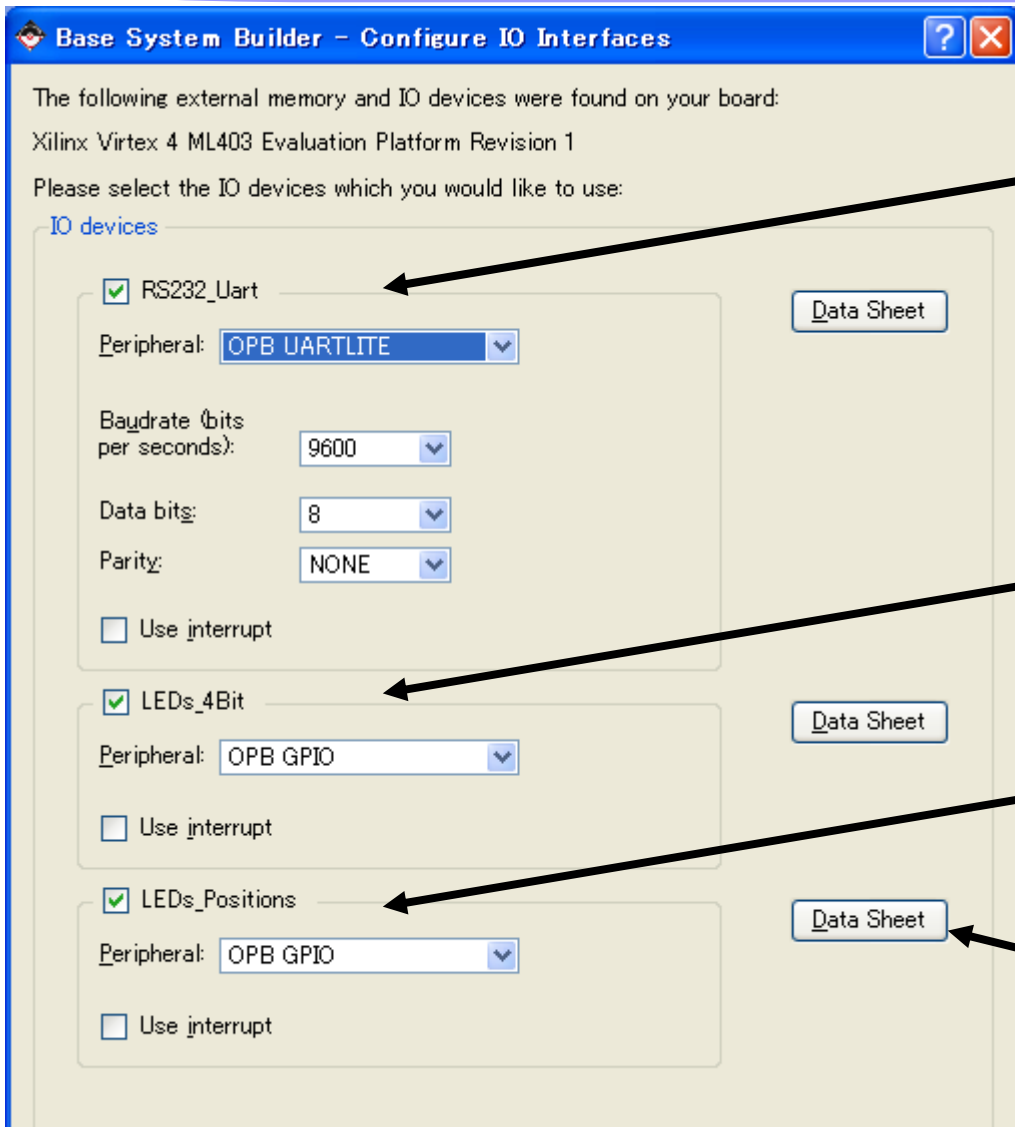


クロック周波数を選択
・リファレンスクロック
・CPUクロック
・バスクロック

JTAGデバッグを行う

OCMの使用/未使用
キャッシュのON/OFF

BSB(Base System Builder)(5)



RS232の使用/未使用
・ボーレートの設定

LEDの使用/未使用

LEDの使用/未使用

Data sheetの閲覧

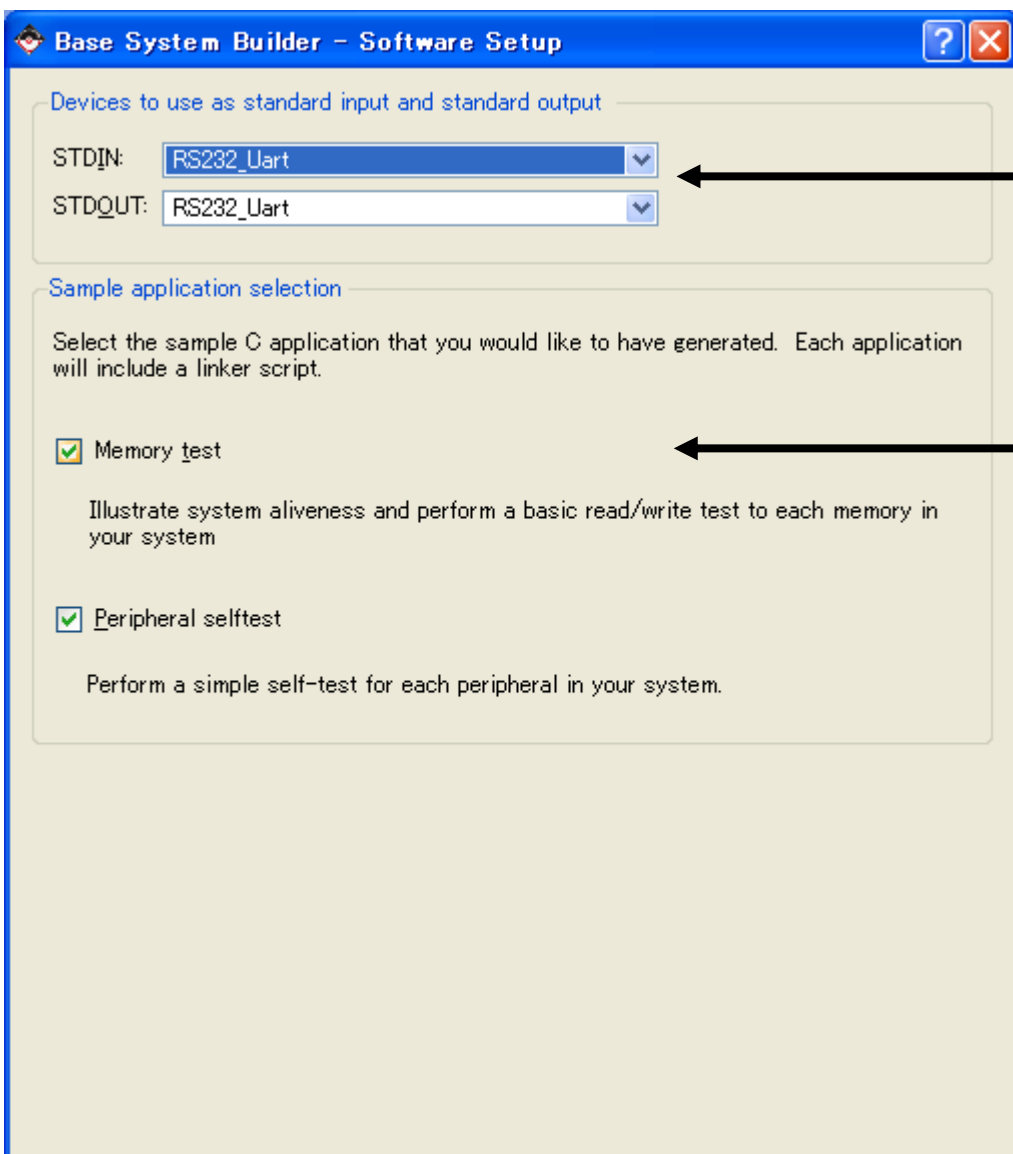
BSB(Base System Builder)(6)



RAMの追加

RAM容量の設定

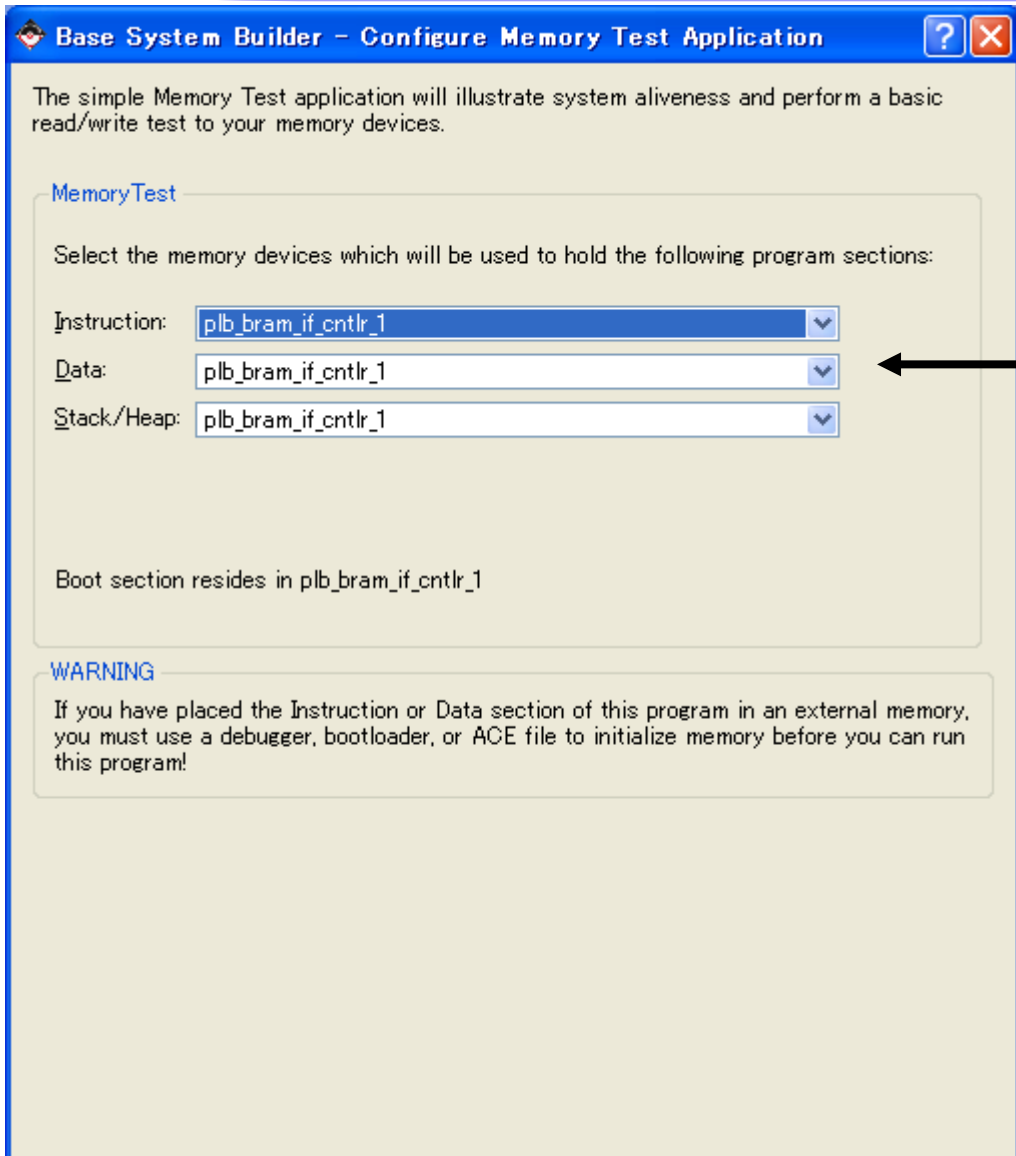
BSB(Base System Builder)(7)



標準入出力の設定

サンプルコードあり/なし

BSB(Base System Builder)(8)



← プログラムやデータの格納先を指定

BSB(Base System Builder)(9)

Base System Builder - System Created


Below is a summary of the system you have created. Please review the information below. If it is correct, hit <Generate> to enter the information into the XPS data base and generate the system files. Otherwise return to the previous page to make corrections.

Processor: PPC 405
Processor clock frequency: 100.000000 MHz
Bus clock frequency: 100.000000 MHz
Debug interface: FPGA JTAG
On Chip Memory : 16 KB
Total Off Chip Memory : 65 MB
- DDR_SDRAM_32Mx32 = 64 MB
- SDRAM_256Kx32 = 1 MB

The address maps below have been automatically assigned. You can modify the the editing features of XPS.

PLB Bus : PLB_V34 Inst. name: plb Attached Components:			
Core Name	Instance Name	Base Addr	High Addr
plb2opb_bridge	plb2opb_C_RNG0_BAS	0x20000000	0x3FFFFFFF
plb2opb_bridge	plb2opb_C_RNG1_BAS	0x40000000	0x7FFFFFFF
plb_ddr	DDR_SDRAM_32Mx32	0x00000000	0x03FFFFFF
plb_ethernet	Ethernet_MAC	0x80400000	0x8040FFFF
plb_bram_if_cntlr	plb_bram_if_cntlr_1	0xFFFFC000	0xFFFFFFFF

Base System Builder - Finish



The Base System Builder has successfully generated your embedded system!

Click the Finish button to return to XPS to compile your hardware system and software application.

```
C:\asato\work\ITEP_ETSS\ETSS2006\FORMaterial\system.mhs
C:\asato\work\ITEP_ETSS\ETSS2006\FORMaterial\data\system.ucf
C:\asato\work\ITEP_ETSS\ETSS2006\FORMaterial\etc\fast_runtime.opt
C:\asato\work\ITEP_ETSS\ETSS2006\FORMaterial\etc\download.cmd
C:\asato\work\ITEP_ETSS\ETSS2006\FORMaterial\system.mss
C:\asato\work\ITEP_ETSS\ETSS2006\FORMaterial\TestApp_Memory\src\TestApp_Memory
C:\asato\work\ITEP_ETSS\ETSS2006\FORMaterial\TestApp_Memory\src\ddr_header.h
C:\asato\work\ITEP_ETSS\ETSS2006\FORMaterial\TestApp_Memory\src\TestApp_Memory
```

設定確認画面と終了画面

Bit File 生成

- **BitFile生成**
 - ◆ メニューのHardware → Generate Bitstream
 - ◆ ...¥implementation¥system.bitが生成される

- **BitFile再生成**
 - ◆ メニューのDevice Configuration → Update Bitstream
 - ◆ CソースをコンパイルしELFファイルを生成(ファイル名はexecutable.elf)
 - ◆ ELFをsystem.bitとマージし、...¥implementaiton¥download.bitを生成

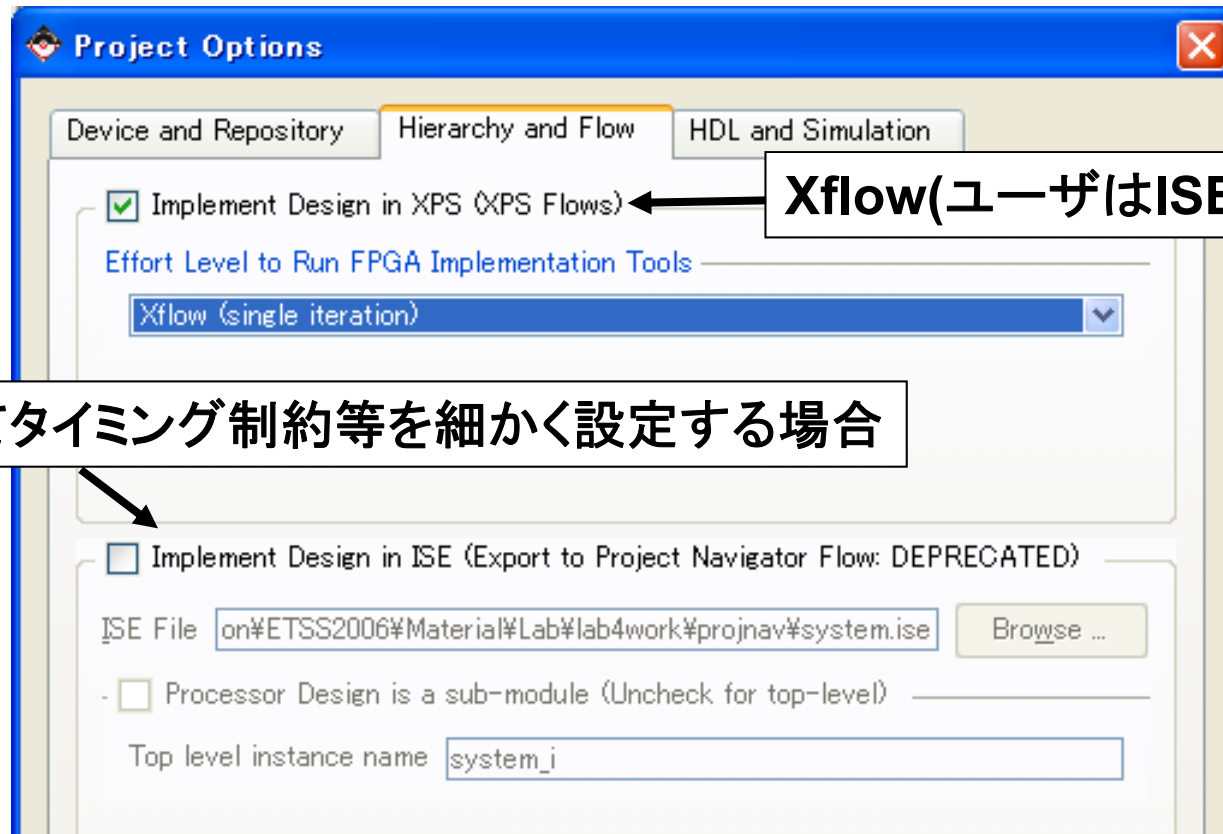
FPGAにダウンロード

- **FPGAボードへdownload**
 - ◆ **メニューのDevice Configuration → Download Bitstream**
 - ◆ **カスタムボードなどの時は、ISEのiMPACTを使って手動でダウンロードする**

オプション設定

- Project Option

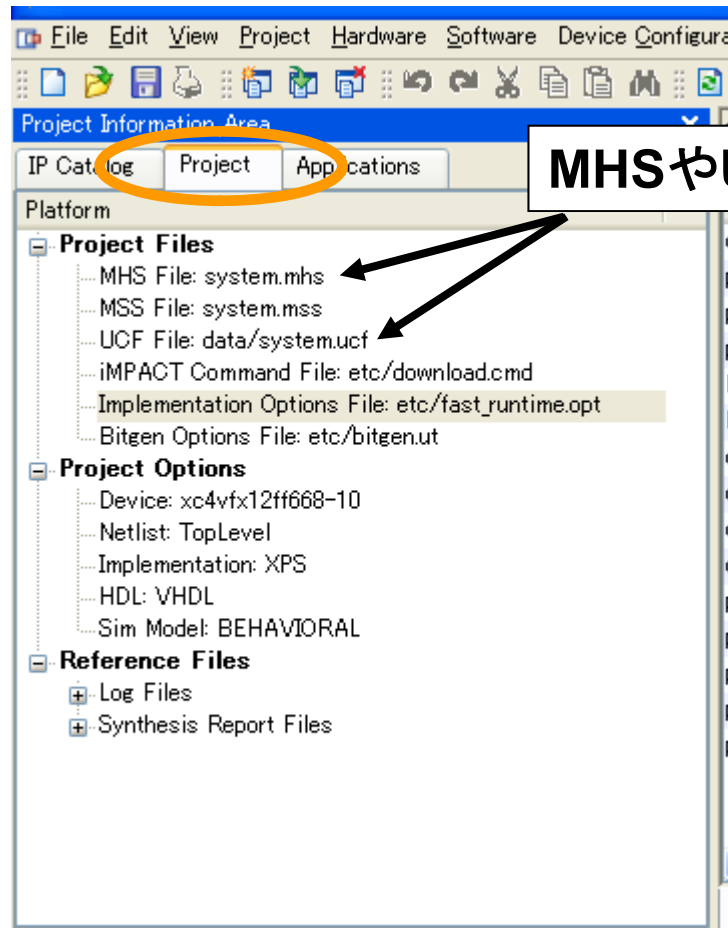
- ◆ メニューのProject → Project Options
- ◆ FPGAシリーズの変更、Xflow or ISEフローの選択



Xflow(ユーザはISEを操作しない)

ISEを使ってタイミング制約等を細かく設定する場合

- MHSファイルやUCFファイルを表示



MHSやUCFの参照、編集が可能

MHSファイル

- MHSファイルとは

- ◆ PowerPCと周辺RAMやPeripheralとの接続を表したTextファイル
- ◆ 外部ポートも記述
- ◆ 各コアのパラメータ設定
- ◆ 慣れるとGUIではなく、MHSを直接編集

```
PORT EXT_I = EXT_I, DIR = I, VEC = [0:7]
```

```
PORT SCLK = SCLK, DIR = I, SIGIS = CLK, CLK_FREQ = 100000000
```

```
BEGIN proc_sys_reset
```

```
PARAMETER C_EXT_RESET_HIGH = 0
```

```
PORT Ext_Reset_In = sys_rst_s
```

```
PORT Slowest_sync_clk = sys_clk_s
```

```
PORT Chip_Reset_Req = C405RSTCHIPRESETREQ
```

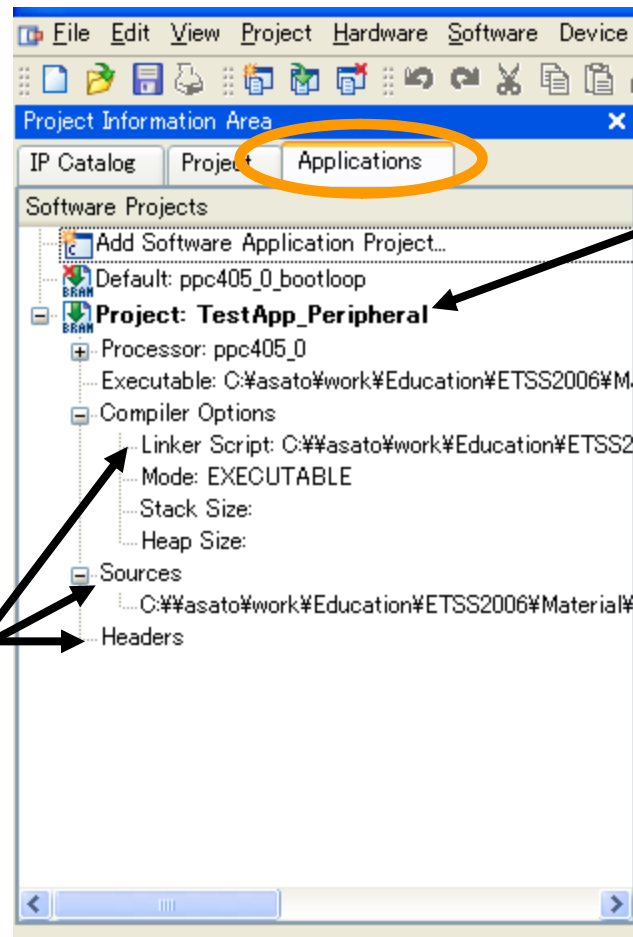
```
PORT Core_Reset_Req = C405RSTCORERESETREQ
```

```
END
```

Applications タブ

- Cソースファイル、ヘッダファイル、リンクスクリプト追加・削除・編集
- コンパイルオプション設定

**linker script,
source file,
header file**

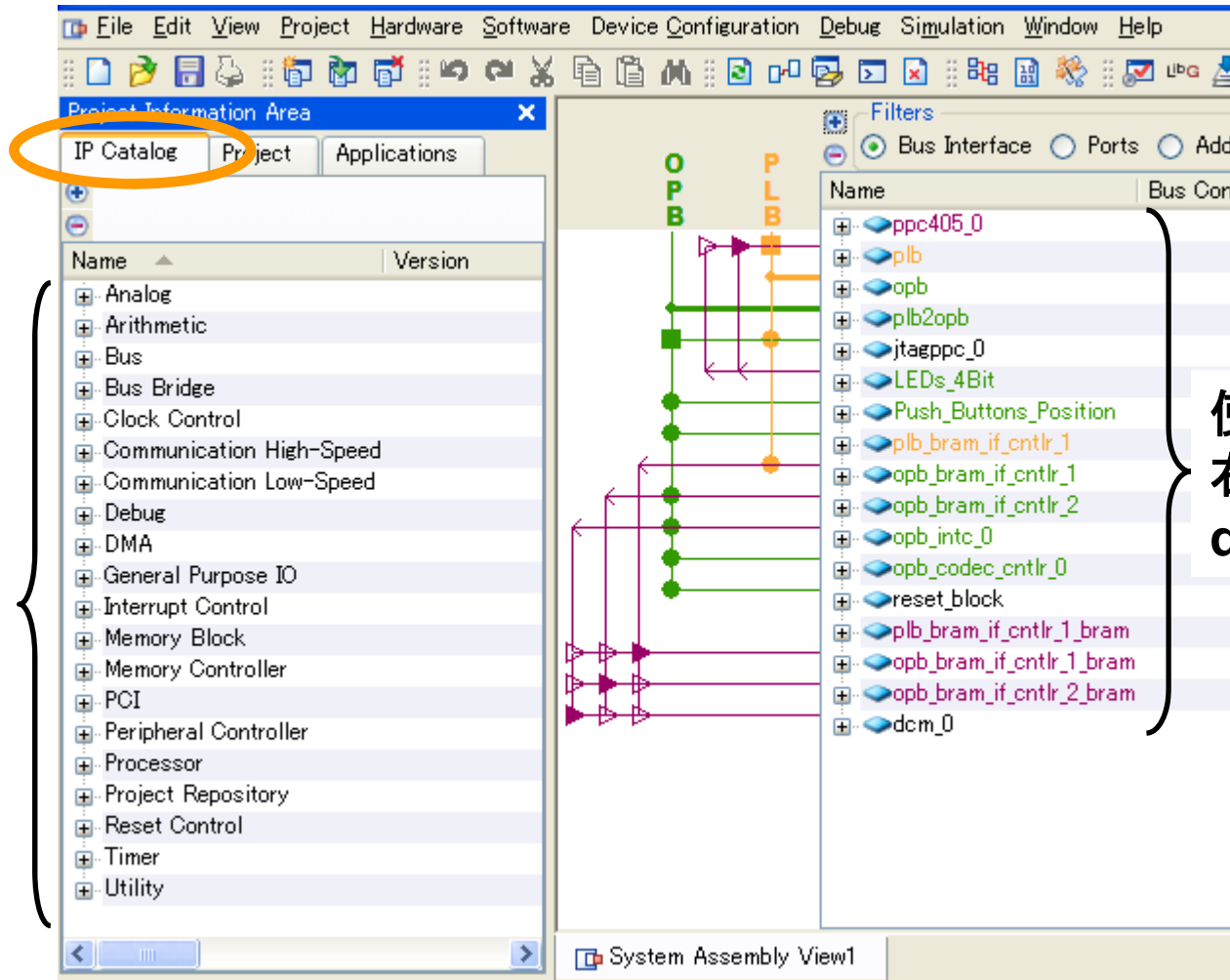


**右クリックで
Set Compiler Options**

コアの追加・削除

- コア (PowerPC, Peripheral) の追加・削除

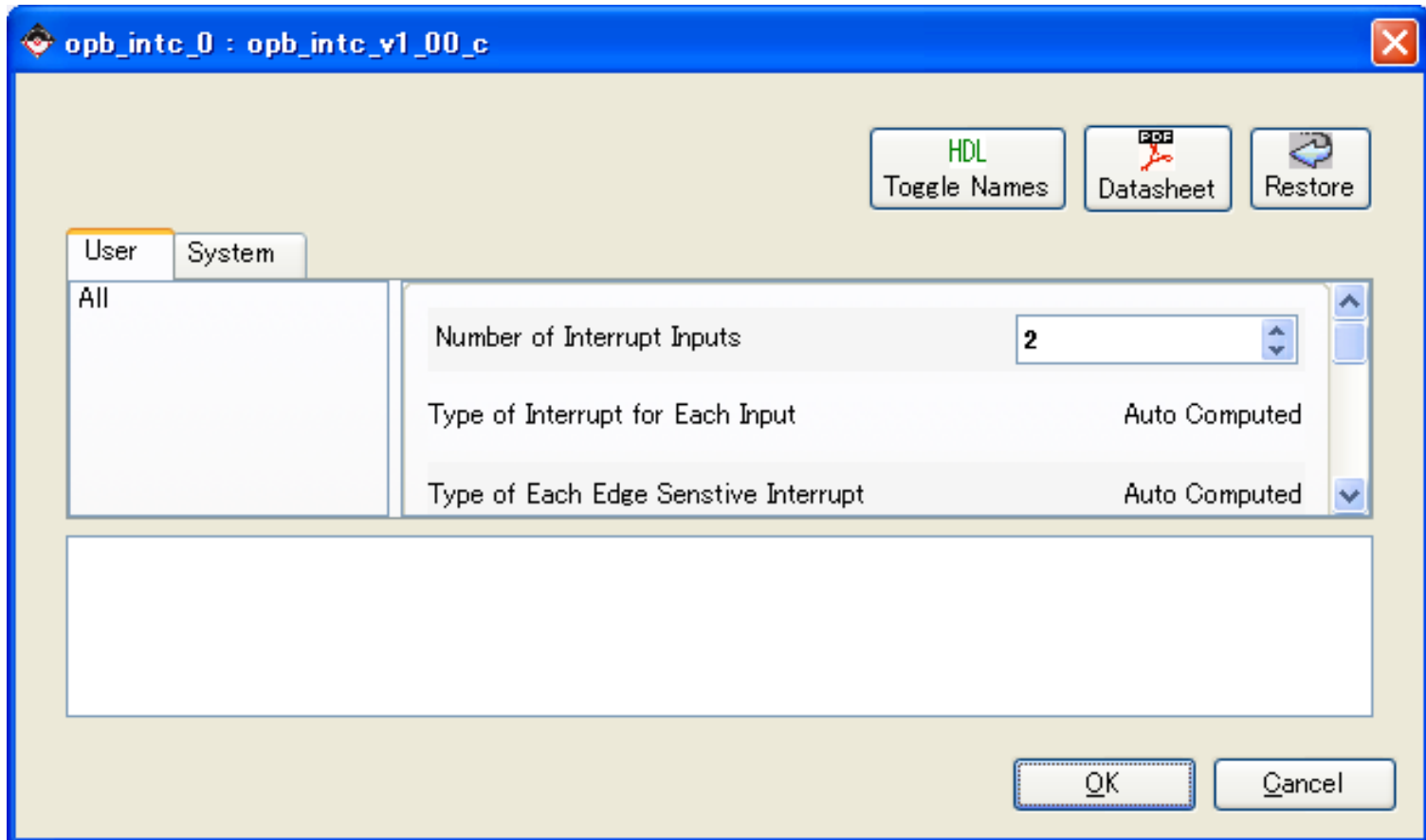
一覧から
コアを選択
して追加



使用中のコア
右クリックで
delete

コアのパラメータ設定

- 使用しているコアのパラメータ設定
 - ◆ コアを選択し右クリックでConfigure IP
 - ◆ MHSファイルへ直接記述する事も可能



Clean, Debugメニュー

- 生成された各種ファイルの削除
 - ◆ Clean Hareware
 - ◆ Clean Netlist
 - ◆ Clean Software
 - ◆ Clean Programs など
- デバッガ
 - ◆ メニューのDegug → Debug XMD、Debug Software Debuggerの実行でデバッガを接続可能
 - ◆ ブレイクポイント、ステップ実行、内部メモリ参照、変数参照等

演習Lab1を行ってください

2. PowerPCシステム設計 について

組み込みシステム設計でまず理解しておきたい事

- バスの種類と各バスの特徴
- 主要なPeripheralの機能
- アドレッシング
- RAM容量
- アーキテクチャに特化した記述
- スタックとヒープ
- リンカ

バスの種類と特徴(1)

PowerPCシステムが持つバス(その1)

- **PLB(Processor Local Bus)**
 - ◆ PowerPCと直接接続されるバス
 - ◆ このバスに接続されているスレーブ(RAM, Peripheral)は高速で動作可能
 - ◆ プログラムやデータ領域を格納するRAMを接続するバス
 - ◆ データは64bitアクセス可能。アドレスは32bitアクセス

- **OPB(On-chip Peripheral Bus)**
 - ◆ Bridge経由でPLB, PPCと接続
 - ◆ PLBに比べると低速で動作
 - ◆ データ、アドレスは32bitアクセス
 - ◆ 各種Peripheralを接続するバス(RAMも可)

バスの種類と特徴(2)

PowerPCシステムが持つバス(その2)

- OCM(On-Chip Memory Bus)
 - ◆ 最速のバス
 - ◆ Processor clockと同等もしくはそれに近いclockで動作可能(<300MHz)
 - ◆ メモリを接続し、高速なInstruction/Dataを格納

主要なPeripheralの機能

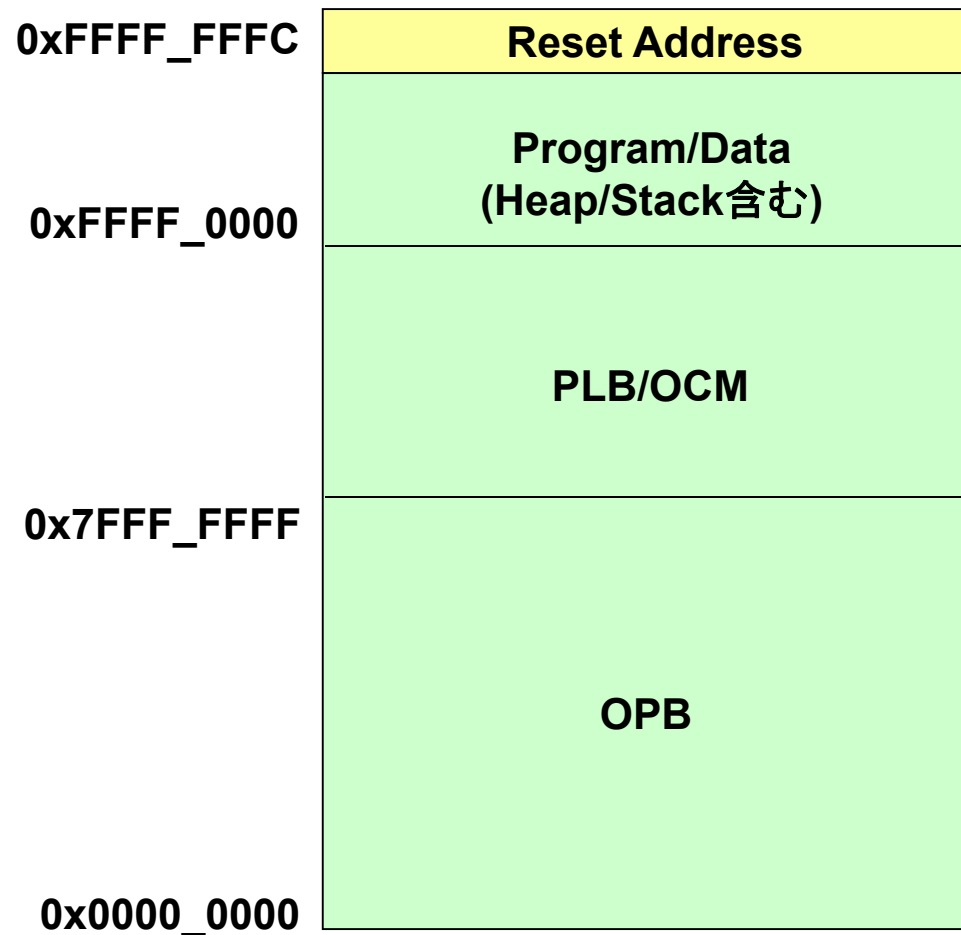
以下はEDKで使えるフリーのPeripheral(一部)

- GPIO ... '1', '0'を入力/出力する
- Uart ... RS232(シリアルポート)経由で通信
- Timer ... 時間の制御
- Interrupt Controller ... 割り込みの制御
- DMA Controller ... DMAの制御

- その他、サードパーティが提供する有償のPeripheral IPなどもある

- 各種Peripheralの機能詳細はData Sheetに記載

PowerPCのメモリマップ



アドレスを決める時、重複しないように
注意が必要！！
PeripheralはPLBかOPBかどちらかに
よって使う領域が違う

RAM容量

RAM容量は限られている！！

- PCやUNIXマシン上でのプログラム実行はRAM容量を意識しない
- 組み込みの場合はRAM容量に制限がある
 - ◆ FPGAが持つ総RAM容量
 - ◆ プログラム、データを格納するRAM容量

アーキテクチャに特化した記述

アーキテクチャに特化した記述を把握しておく

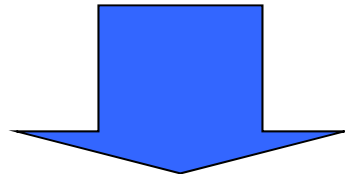
- 下記はアーキテクチャに特化した関数

```
sys_status = XGpio_Initialize(&leds, XPAR_LED_DEVICE_ID);
```

- 下記は物理アドレスを使った記述

```
#define rDMA_STS (*(volatile unsigned char *)0x40020000)
```

```
rDMA_STS &= 0x01;
```



移植する場合は等価な記述へ置換する

スタックとヒープ

スタックとヒープを意識する

- スタック
 - ◆ 実行途中のデータを一時格納する領域
 - ◆ サブルーチンが呼び出された時、処理中のデータや戻りアドレスを一時的に退避させる
- ヒープ
 - ◆ スタックは関数が終わると自動消去されるが、自動消去されない領域
 - ◆ 確保、開放をユーザが行う
- EDKではデフォルトでそれぞれ4KB。リンカスクリプトで設定可
- スタックは知らず知らずの内に積み上がるので要注意
- 容量を超えるとハングアップする

リンカ


リンカスクリプトを忘れてはならない

- 下記はリンカスクリプト一部

```
MEMORY
```

```
{
plb_bram_if_cntlr_1 : ORIGIN = 0x20000000, LENGTH = 0x00001FFF
plb_bram_if_cntlr_2 : ORIGIN = 0x40000000, LENGTH = 0x003FFFFFFF
}
```

RAMの容量を変更したら
ココも合わせなければならない



```
SECTIONS
```

```
{
    .text : { *(.text) } > plb_bram_if_cntlr_1
    .data : { *(.data) } > plb_bram_if_cntlr_2
    .bss  : { *(.bss) *(COMMON) } > plb_bram_if_cntlr_2
}
```

RAMのBASE ADDRESSを変更したら
ココも合わせなければならない



3. 基本的なソフトウェアの 記述

- ポーリングの例

```
do {  
  
    h_sts = read_hw_status( );  
    if(h_sts==TRUE) {  
        処理A  
    }  
  
    f_sts = func1( );  
    if(f_sts==TRUE) {  
        func2( );  
        func3( );  
    }  
  
}while(1)
```

- ある事象の発生を定期的にチェックし処理を行う
- func2,func3の処理が重い場合、処理Aのリアルタイム応答性が崩れる
- プログラムの記述自体は比較的簡単

よく使用するGPIOの記述(1)

- **GPIOの初期化記述**

```
sys_status = XGpio_Initialize(&leds, XPAR_LED_DEVICE_ID);
```

ポインタ, デバイスID

デバイスID

→ XPAR_インスタンス名_DEVICE_ID (すべて大文字)

→ インスタンス名はMHSに記載

- **GPIOのIN/OUT設定**

```
XGpio_SetDataDirection(&leds, 1, 0x80);
```

ポインタ, チャンネル, direction

チャンネル

→ 1 or 2 2チャンネルあるIOの内どちらを使うか

direction

→ 0=出力、1=入力 各ビット毎に設定する

よく使用するGPIOの記述(2)

- **GPIOで“0”や“1”を出力する記述**

```
XGpio_DiscreteWrite(&leds, 1, 0x5 );
```

ポインタ, チャンネル, 出力値

出力値

→ bit毎に “0” or “1”を指定

```
XGpio_DiscreteWrite(&leds, 1, out1 );
```

- **GPIOで入力値を取得する記述**

```
input1 = XGpio_DiscreteRead(&gpio_from, 1);
```

ポインタ, チャンネル

演習Lab2を行ってください

- 割り込みの例

```
main () {  
    do {  
        f_sts = func1 ( );  
        if (f_sts==TRUE) {  
            func2 ( );  
            func3 ( );  
        }  
    } while (1)  
}  
  
sts_handler () { // 割り込みハンドラ  
    処理A  
}
```

- リアルタイム応答性が保たれる
- プログラムの難易度がややup(割り込みの制御が面倒になる)

割り込みセットアップ(1)

- 割り込みハンドラ登録

```
XExc_RegisterHandler ( XEXC_ID_NON_CRITICAL_INT,  
                      (XExceptionHandler) XIntc_InterruptHandler,  
                      &intc );
```

- 割り込みハンドラコネクト

```
sys_status = XIntc_Connect( &intc,  
                           XPAR_OPB_INTC_OPB_USR_CNTLR_EVT_INTR,  
                           (XInterruptHandler) CDC_Handler,  
                           (void *)0 );
```

→XPAR_INTコントローラインスタンス名_割り込み発生デバイスインスタンス名_ポート名_INTR (すべて大文字)

→ (XInterruptHandler)割り込みハンドラ名

割り込みセットアップ(2)

- GPIO割り込みイネーブル

```
XGpio_InterruptEnable(&pushs, 0x1);  
XGpio_InterruptGlobalEnable(&pushs);
```

- 割り込み許可

```
XIntc_Enable(&intc, XPAR_OPB_INTC_OPB_USR_GNTRL_R_EVT_INTR);
```

→XPAR_INTコントローラインスタンス名_割り込み発生デバイスインスタンス名_ポート名_INTR (すべて大文字)

割り込みセットアップ(3)

- 割り込みクリア（セットアップ時）

```
XIntc_Acknowledge(&intc, XPAR_OPB_INTC_OPB_USR_GNTR_EVT_INTR);
```

- 割り込み許可（全体）

```
XExc_mEnableExceptions(XEXC_NON_CRITICAL);
```

- 割り込みスタート

```
sys_status = XIntc_Start(&intc, XIN_REAL_MODE);
```

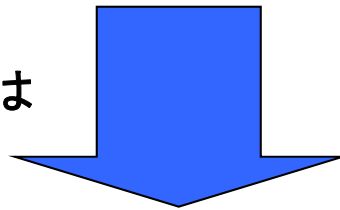
- GPIO割り込みクリア（割り込み処理終了時）

```
XGpio_InterruptClear(&pushs, 0x1);
```

4. EDKの使い方 (中級編)

既存のPeripheralだけでは仕様を満たせない！

そんな時は



必要な機能を自作すればよい！

- EDKに設計済みのRTL回路をユーザIPとして取り込む事が可能
- PowerPCとのデータのやり取りは基本的にBRAM(dual port)経由

ユーザIPの追加(2)

- メニューのHardware → Create or Import Peripheralを実行

新規 or 既存のIPをimport

Create Templates



Implement/Verify



Import to XPS

... EDK compliant peripheral, or help you import an existing peripheral into an ... and directory structures required by EDK will be generated.

Select flow

- Create templates for a new peripheral
- Import existing peripheral

Flow description

This tool will create HDL templates that have the EDK compliant port/parameter interface. You will need to implement the body of the peripheral.

ユーザIPの追加(3)

共有IP or 単一プロジェクト

To an EDK user repository (Any directory outside of your EDK installation path)

Repository: Browse...

To an XPS project

Project: Browse...

格納されるpath

Peripheral will be placed under:

ユーザIPの追加(4)

Create Peripheral - Name and Version

Name and Version
Indicate the name and version of your peripheral.

Enter the name of your peripheral. This name will be used as the top HDL design entity.

Name:

Version:

Major revision: Minor revision: Hardware/Software compatibility revision:

IP名称

IPバージョン

ユーザIPの追加(5)

To which bus will this peripheral be attached?

- On-chip Peripheral Bus (OPB)
- Processor Local Bus (PLB)
- Fast Simplex Link (FSL)

使用するバス指定

ATTENTION

Refer to the following documents to get a better understanding of buses through the IPIF interconnection standards.

[CoreConnect Specification](#)

[OPB IPIF Specification for slave only peripherals](#)

[OPB IPIF Specification for master/slave peripherals](#)

[PLB IPIF Specification for slave only peripherals](#)

[PLB IPIF Specification for master/slave peripherals](#)

[FSL IPIF Specification for master/slave peripherals](#)

ここでのinterruptのチェックは
はずしておく

User Logic

Basic slave service and support
Common and typically required by most peripherals for operations like logic control, status report, and etc.

- S/W reset and MIR
- User logic interrupt support
- User logic S/W register support

Advance slave service and support
Typically required by peripherals that need data buffering or multiple memory/address spaces access.

- Burst transaction support
- FIFO
- User logic address range support

Master service and support
Typically required by complex peripherals like Ethernet and PCI for command data transfers between regions.

- DMA
 - Simple mode
 - Packet mode Scatter Gather
- User logic master support

< Back Next > Cancel

ユーザIPの追加(6)

The software accessible registers will be implemented in the user-logic module of your peripheral addressable on the byte, half-word or word boundaries. The following fields determine the char

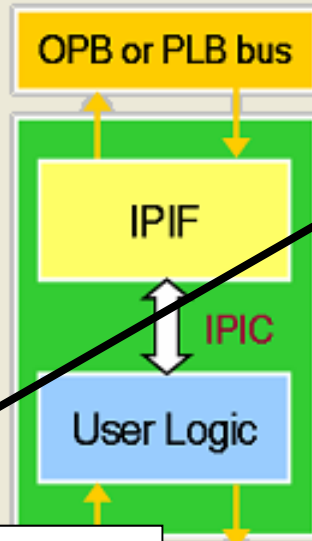
Number of software accessible registers:
 Data width of each register: bit

レジスタ数、ビット幅

Write Mode

Instead of the usual *acknowledge write* behavior, a *posted write* behavior, the IPIF unconditionally writes to the custom user logic will retire the data immediately

- Enable posted write behavior
- Disable posted write behavior for normal acknowledge
- Allow dynamic posted/acknowledged write behavior



Note: all IPIC ports are active high.

- IP2Bus_Clk
- Bus2IP_Clk
- Bus2IP_Reset
- Bus2IP_Freeze
- Bus2IP_Addr
- Bus2IP_Data
- Bus2IP_BE
- Bus2IP_Burst
- Bus2IP_RNW
- Bus2IP_CS
- Bus2IP_CE
- Bus2IP_RdCE
- Bus2IP_WrCE
- IP2Bus_Data
- IP2Bus_Ack

Port descr

Restore Defaults

IP内部で接続するネットの指定
(通常デフォルトでよい)

More Info

< Back

ユーザIPの追加(7)

simulation platform to help you simulate your peripheral. Indicate if you want this tool to generate functional Language (BFL) stimulus file for the target bus.

Generate BFM simulation platform for ModelSim-SE or ModelSim-PE

This feature requires that you have accepted the associated IBM license agreement and installed the BFM toolkit. The link below shows how:

[BFM Toolkit Installation Instructions](#)

ModelSim用simulationモデル生成

flows. The tool will also generate EDK interface files to connect the generated peripheral to a processor system.

Verilogで記述している場合はチェックする

Peripheral (VHDL)

IPIF (VHDL)

User Logic (VHDL)

Note

Should the peripheral interface (ports/parameters) or file list change, you will need to regenerate the EDK interface files using the import functionality of this tool.

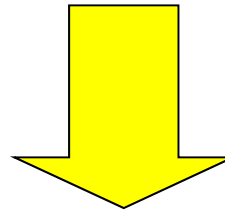
Generate stub 'user_logic' template in Verilog instead of VHDL

Generate ISE and XST project files to help you implement the peripheral using XST flow

Generate template driver files to help you implement software interface

ユーザIPの追加(8)

- この時点でまだユーザIPのラッパ一部分しか作成していない



この後の作業

- RTLの微修正 (BRAMに接続する記述)
- MPDファイルの修正
- PAOファイルの修正
- MHSファイルの修正
- UCFファイルの修正 (必要に応じて)

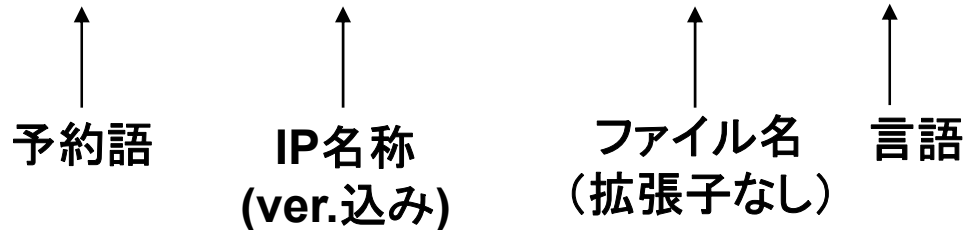
MPD, PAO ファイル

● MPDファイル

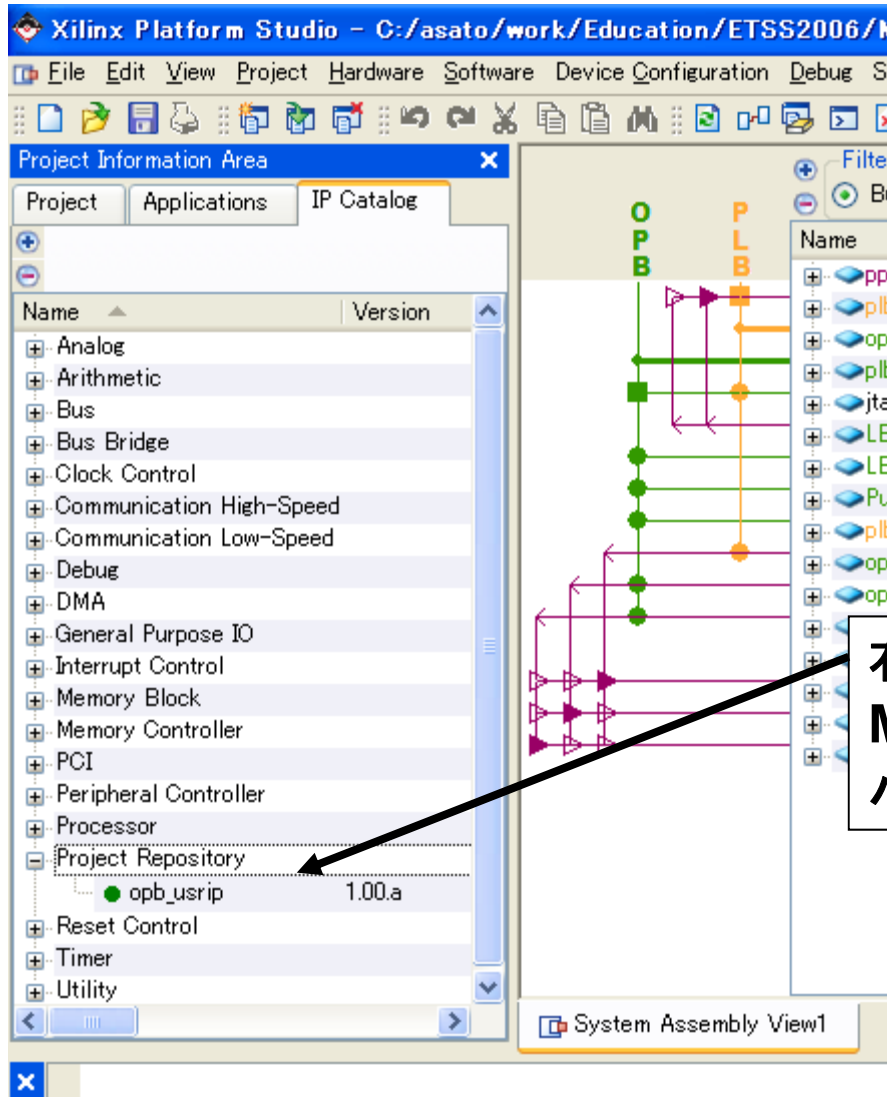
- ◆ ユーザIPのポートが記述されたTextファイル
- ◆ 追加するポート(RAM I/F, その他)を記述する
- ◆ PORT wen = "", DIR = 0, VEC = [0:3]
- ◆ PORT intr = "", DIR = 0, SIGIS = INTERRUPT, SENSITIVITY = EDGE_FALLING

● PAOファイル

- ◆ ユーザIPで使用するファイル(VHDL, Verilog)をリストする
- ◆ lib opb_usrip_v1_00_a user_logic vhd



ユーザIPの追加(9)



右クリックでAdd IPを実行。
MHSに反映されるので、追加ポートや
パラメータを記述する

演習Lab3を行ってください

5. PowerPCシステム設計 (実践編)

本講座の最終ターゲット

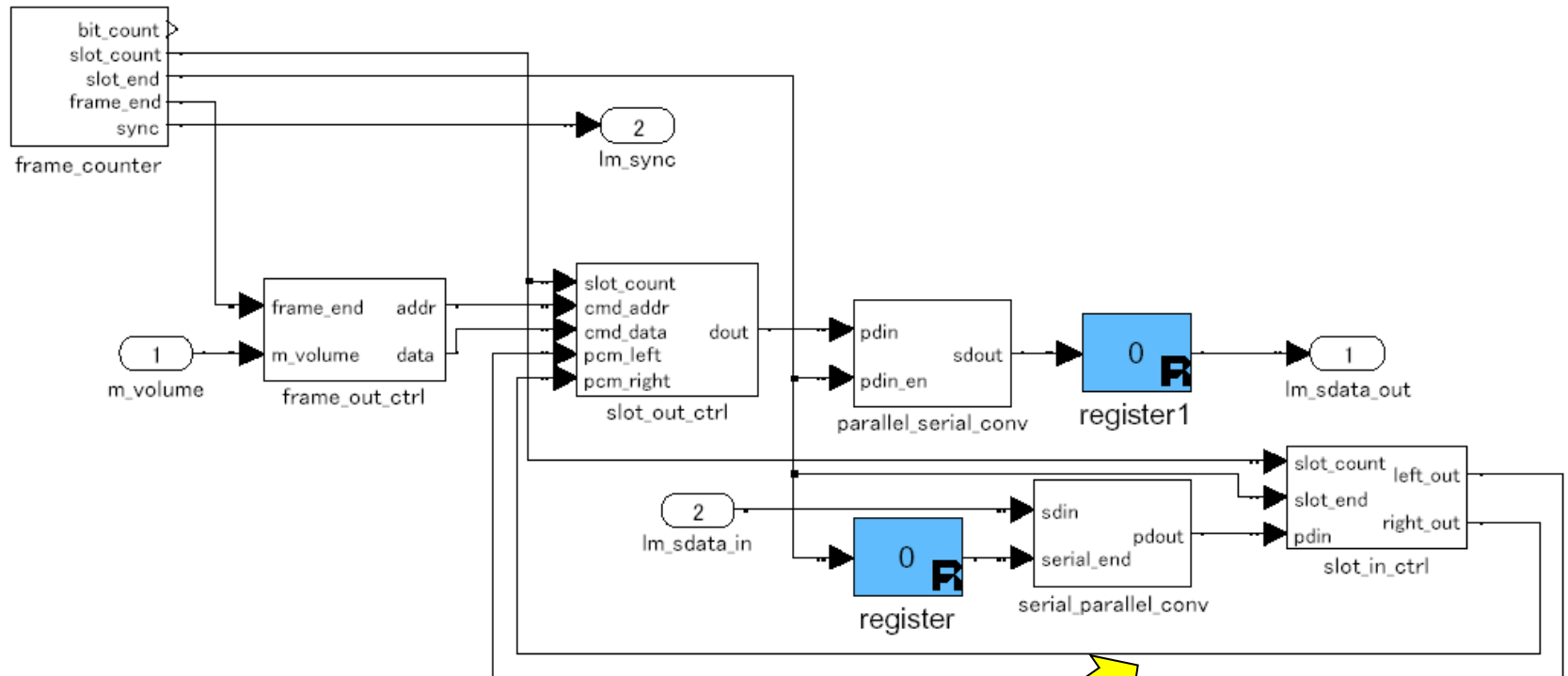
既に設計済みのRTL回路にソフト処理を挿入する。
⇒PowerPCを入れ、ソフト処理を行うフェーズを追加する。
つまり、組み込みシステムを完成させる！！

既に設計したRTL回路

履修済みの講座で設計したCodec制御回路の概要

- 外部インターフェースはシリアルCodecチップ
- フレームから音声データを抽出
- シリパラ変換し20bitの音声データを生成
- 音声データをパラシリシ、フレーム生成の後Codecチップへ出力

Codec 制御回路



- ・LeftチャンネルとRightチャンネルの音声データ
- ・各々20bit
- ・このパスにPowerPCを入れ、ソフト処理させる

システム完成までのStep

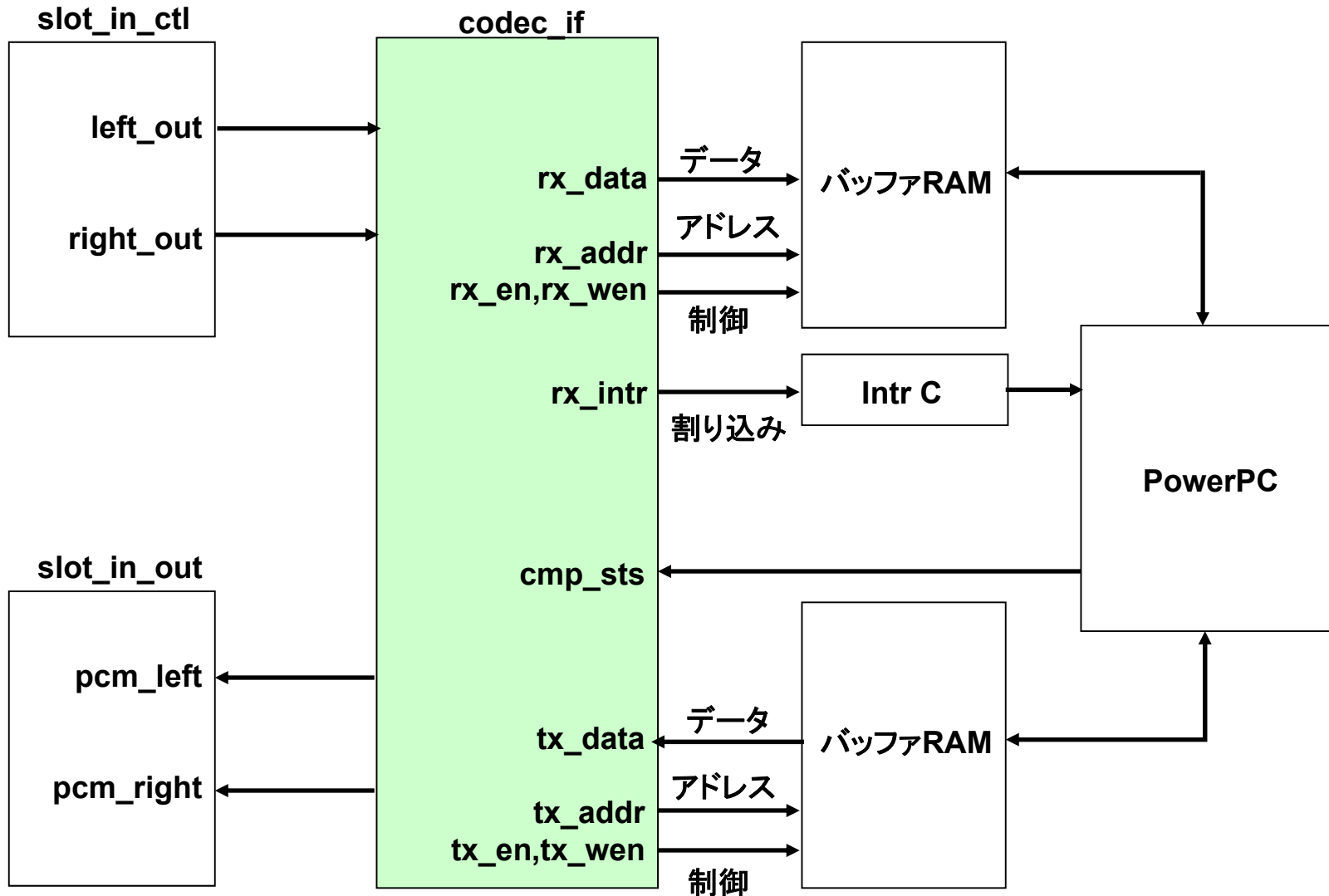
1. バッファRAMを使い、データをスルーさせる仕組みを作る。
→動作的には変化なし
2. ノイズを混入するソフト処理を実装する。
ON/OFF機能あり
3. (ノイズフィルタを実装する)

Step1

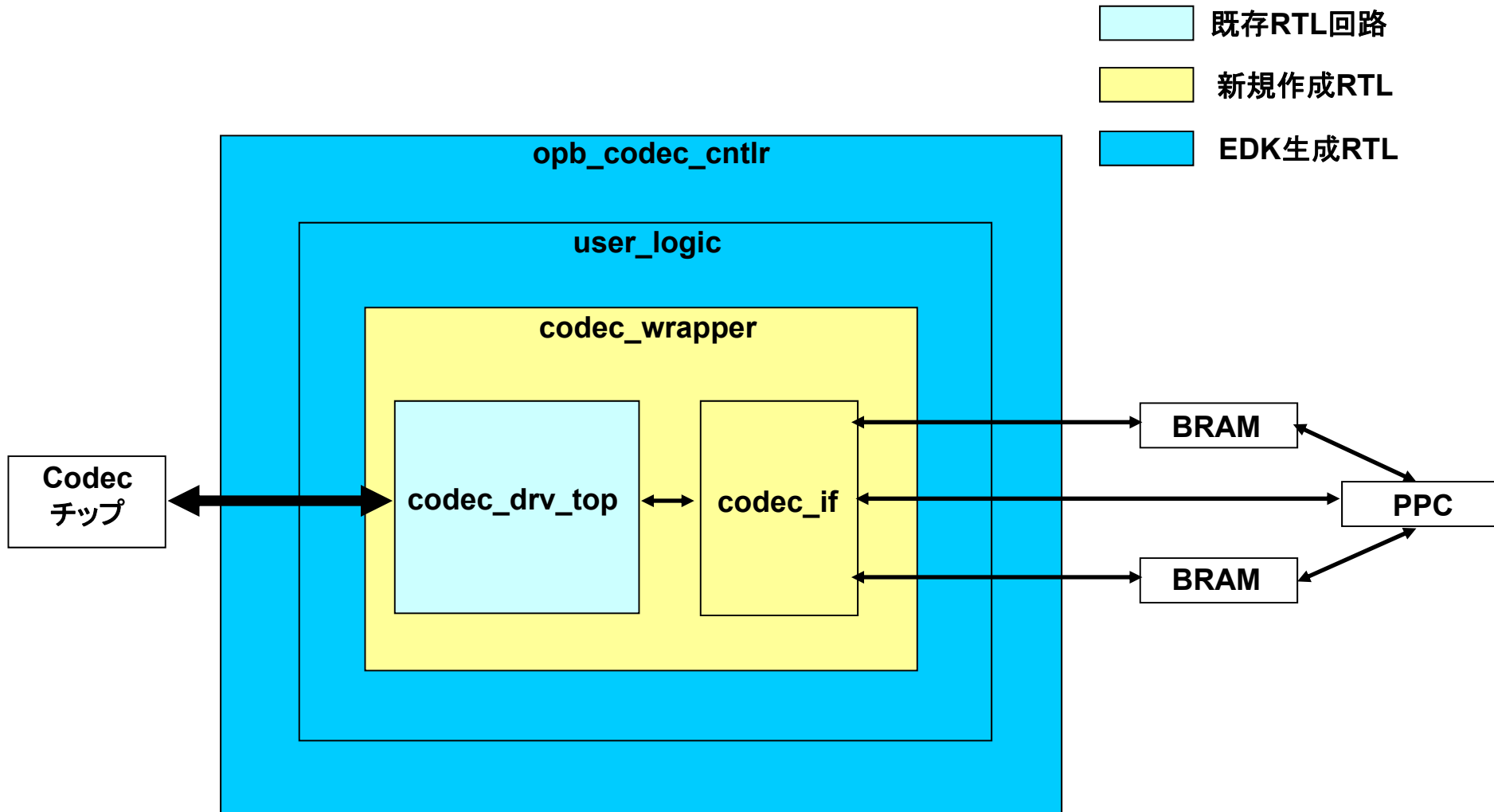
バッファRAMを使い、データをスルーさせる仕組みを作る

- Codec制御回路をユーザIPとして取り込む
- CodecチップからのデータレートとPowerPCバスのデータレートが異なるので、それを吸収するバッファRAMが必要
同様にPowerPC→CodecもバッファRAMが必要
- RAMを読み書きする為のコントローラが必要
(RAMイネーブル生成)
- 今回はPingPongバッファである必要はない
また、DMAによるデータ転送も必要なし
- Cの記述は配列から配列にデータを代入するだけ

簡易ブロック図

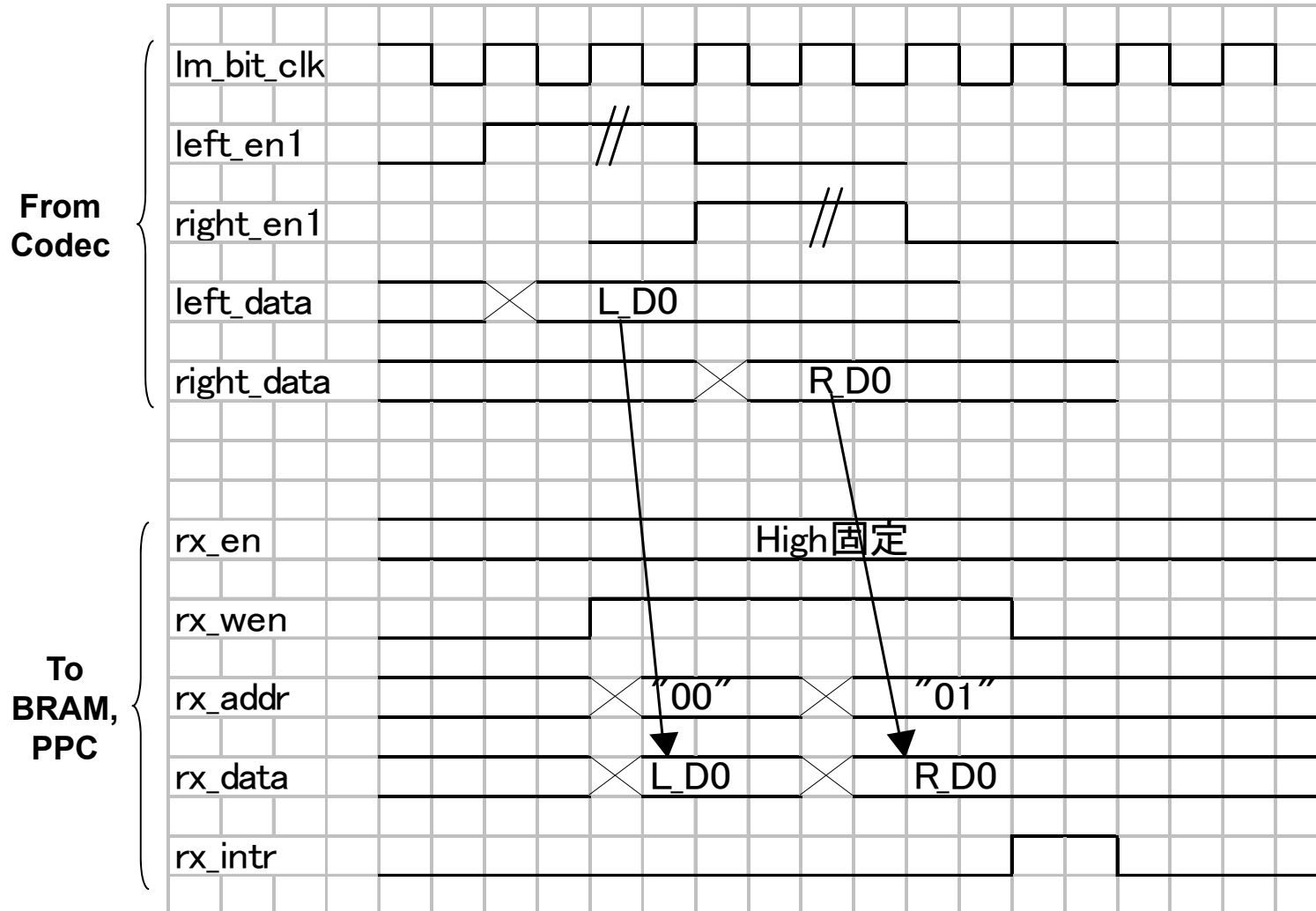


階層構造



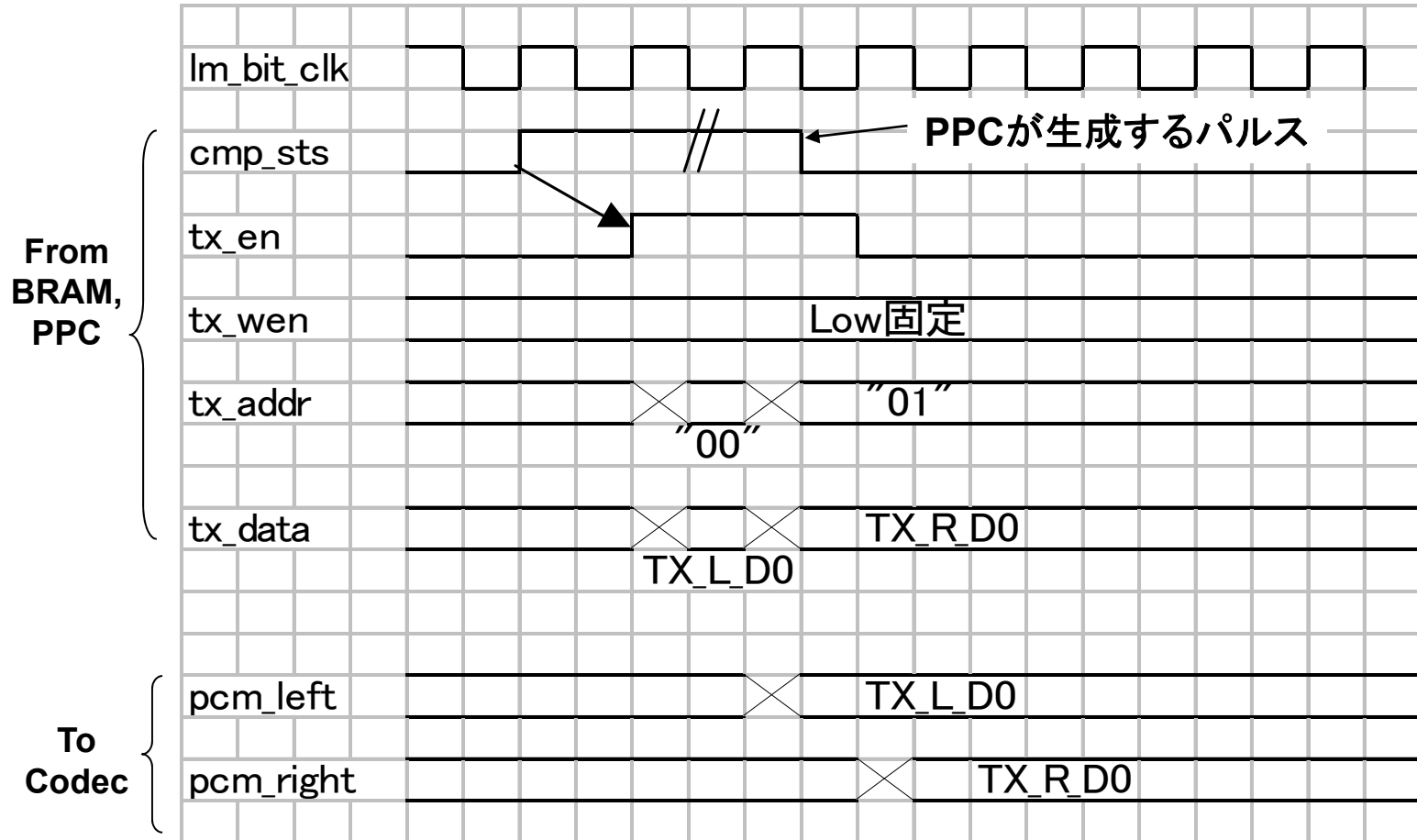
タイミングチャート(1)

- codec_ifのタイミング (Codec → BRAM,PPC)



タイミングチャート(2)

- codec_ifのタイミング (PPC, BRAM → Codec)



演習Lab4を行ってください

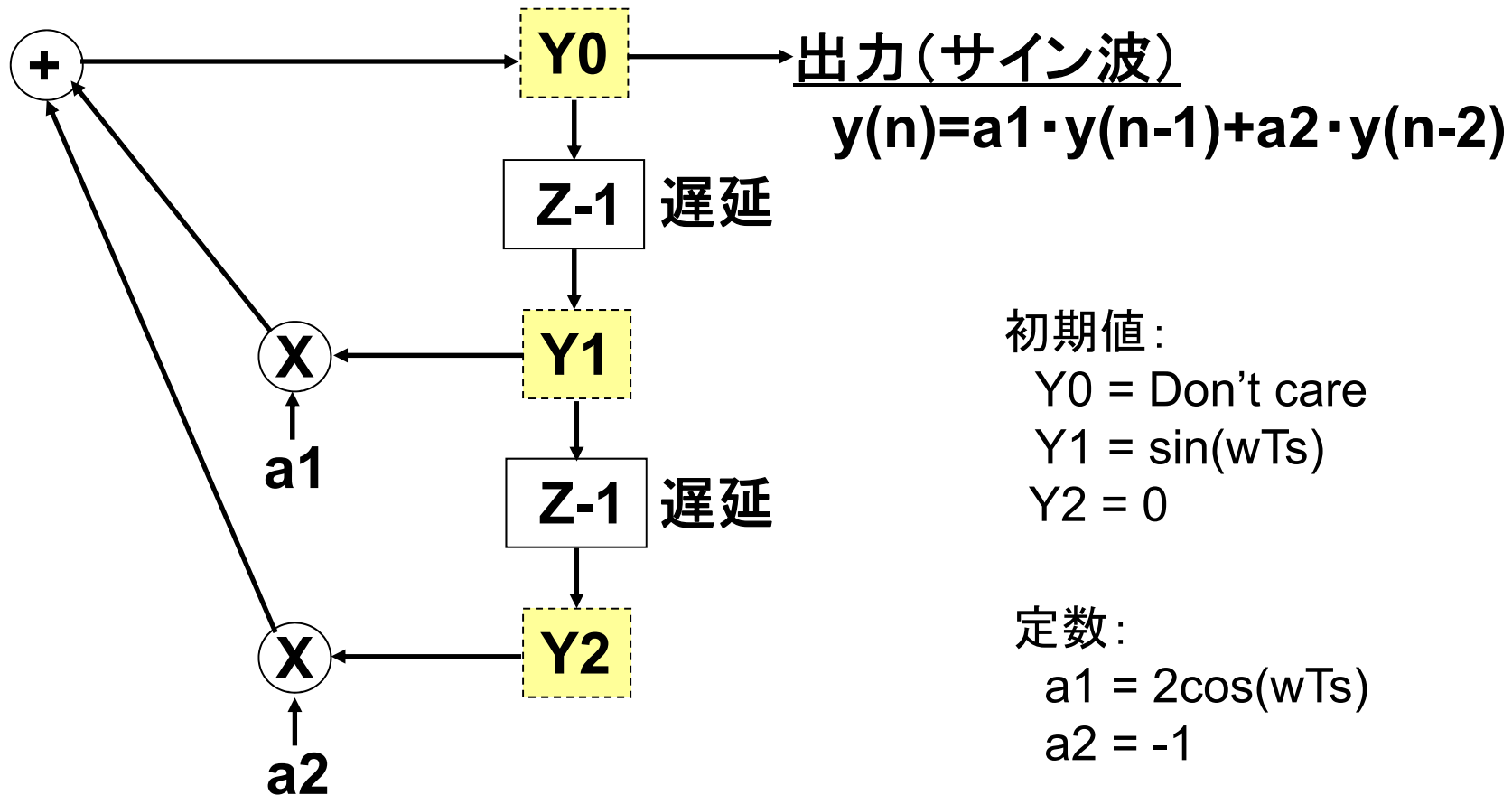
Step2

ノイズを混入するソフト処理を実装する

- 割り込みハンドラでサイン波「ノイズ」を生成し、元の音楽データに付加する
- プッシュスイッチでノイズのON/OFFを切り替えられるようにする
 - ◆ プッシュを押している間はノイズOFF
 - ◆ プッシュを離している間はノイズON

サイン波「ノイズ」

- 今回はIIRフィルタを利用したサイン波発生器を作成します



浮動小数点なら

- 例えば、サイン波の周波数が440Hz, サンプル周波数が8000Hzの場合

定数:

$$w = 2 * \text{PI} * 440 = 880 * \text{PI}$$

$$T_s = 1 / 8000$$

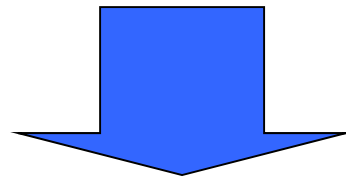
$$a1 = 2 \cos(w T_s) = 2 \cos(880 * \text{PI} / 8000) = 1.8817615$$

$$a2 = -1$$

初期値:

$$Y2 = 0$$

$$Y1 = \sin(w T_s) = 0.33873792$$



- floatで型宣言した変数に代入し、演算するだけ

PowerPC405 (固定小数点)の場合

- 固定小数点化する必要がある
- 例えばshort型(16bit)に変更するとして、小数点の位置をどこにするか？（ユーザが判断する事）
- +2~-2の範囲を表せればよいので、16bit中14bitを小数点にする
- 前ページの数値を変換すると、

$$a2 = -1 = 0xc000$$

$$\begin{aligned} a1 &= 0x4000(1) \times 1.8817615 = 16384 \times 1.8817615 \\ &= 0x786f \end{aligned}$$

$$a1と同様に考えて、Y1 = 0x15ae$$

PowerPC405 (固定小数点)の場合

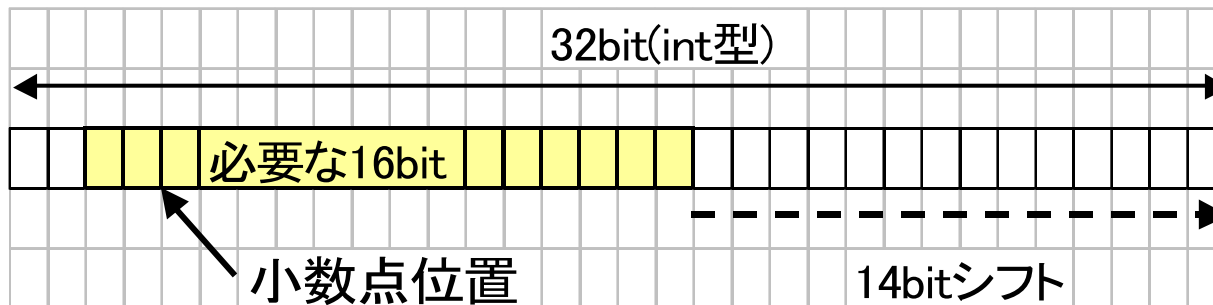
- 演算部分も変換する必要がある

$y[0] = a1*y[1] + a2*y[2];$ //浮動小数点

float型

$y[0] = ((int)a1*y[1] + (int)a2*y[2]) \gg 14;$ //固定小数点

short型



演習Lab5を行ってください