

1. EDKの使い方 (初級編)

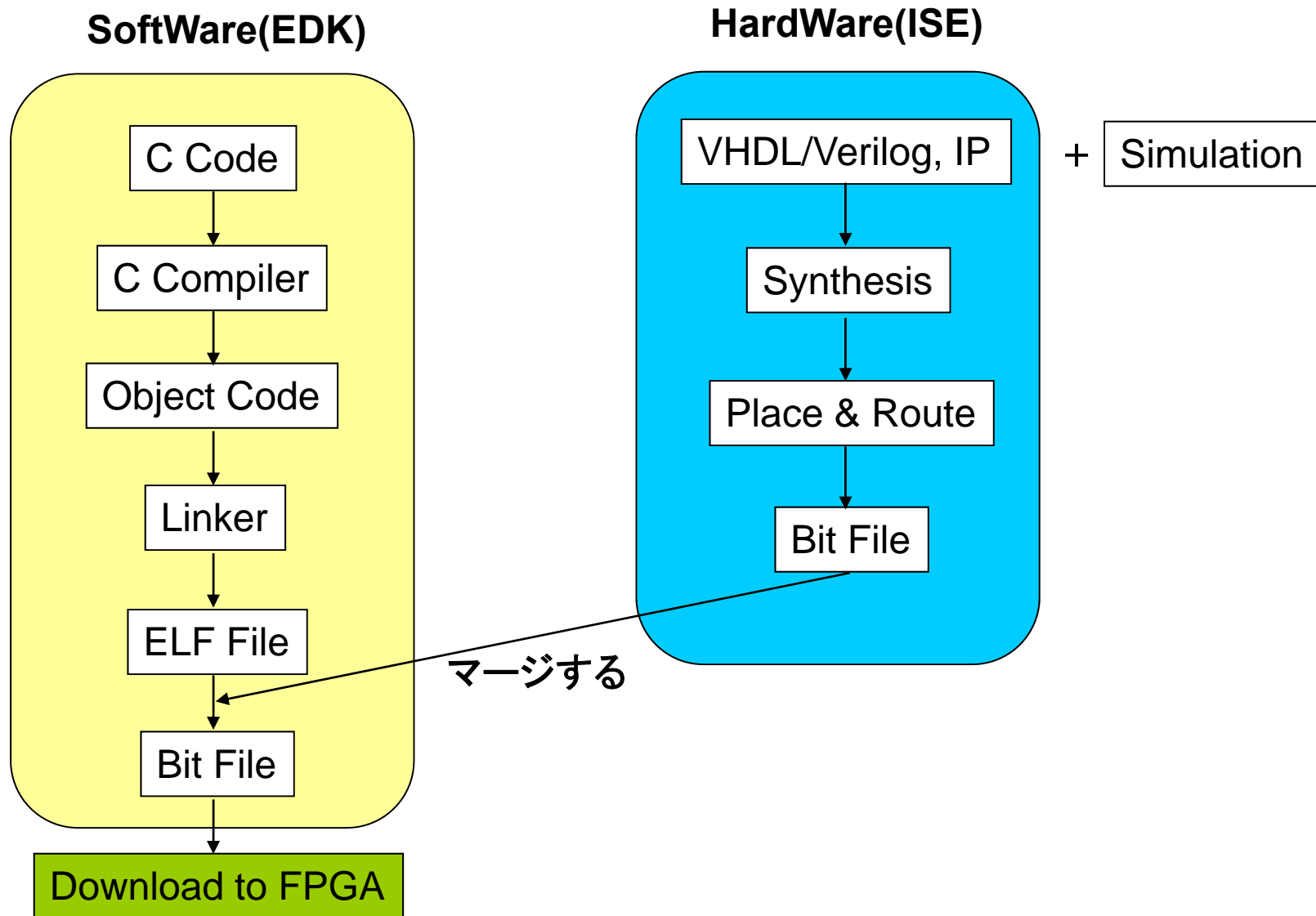
ザイリンクスPowerPC開発でよく使われる用語

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

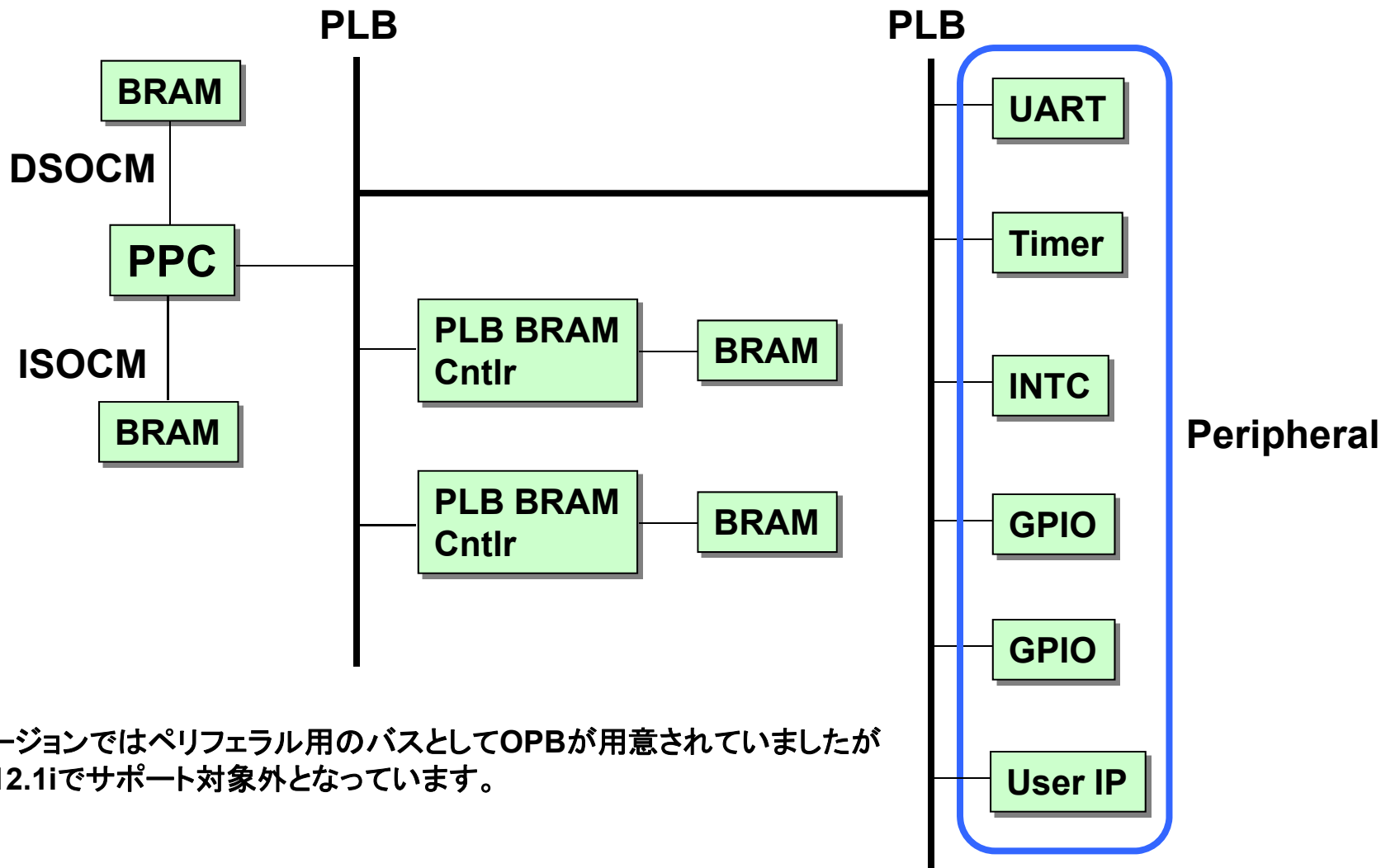
- PLB ... Processor Local Bus
- (OPB ... On-chip Peripheral Bus)
- OCM Bus ... On-Chip Memory Bus
- BRAM ... Block RAM

- 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

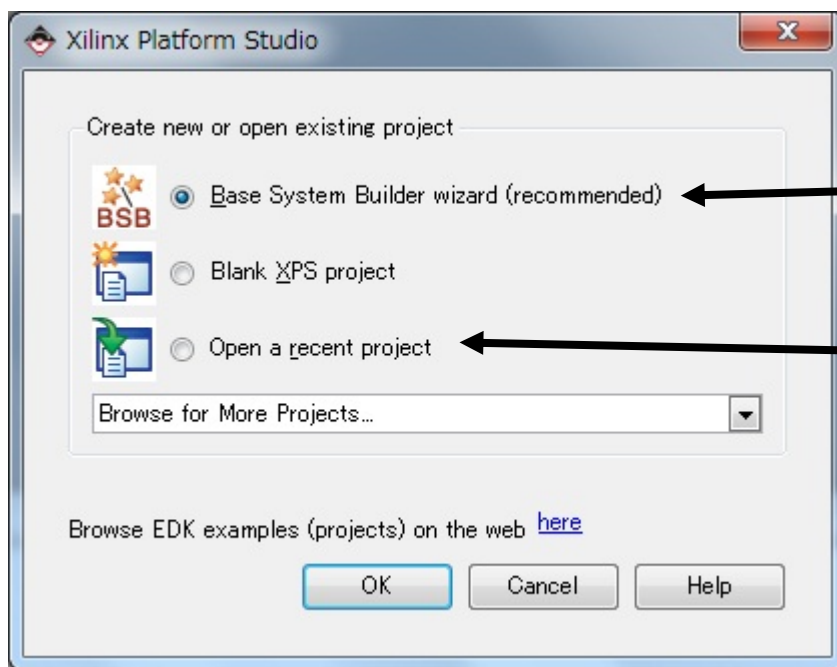


旧バージョンではペリフェラル用のバスとしてOPBが用意されていましたがEDK12.1iでサポート対象外となっています。

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。

ツール起動後の初期画面

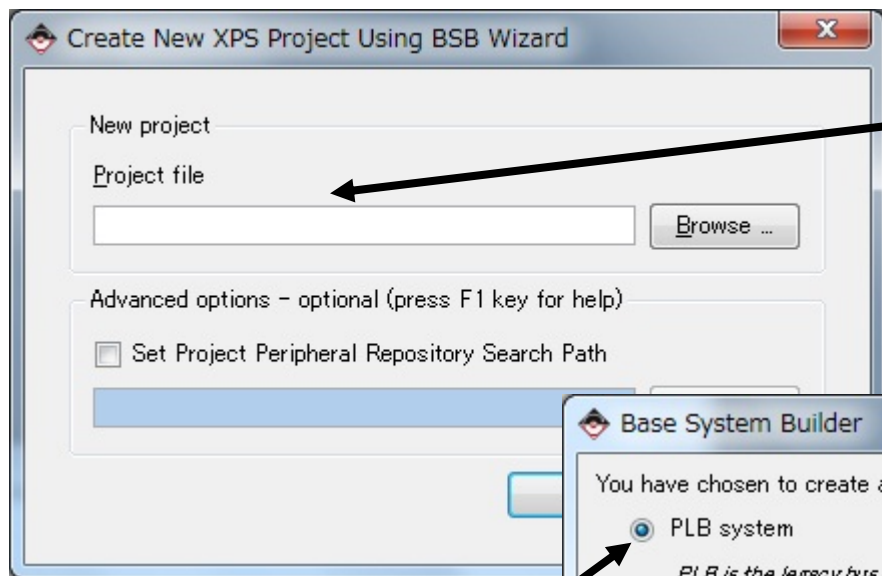


BSBを選択

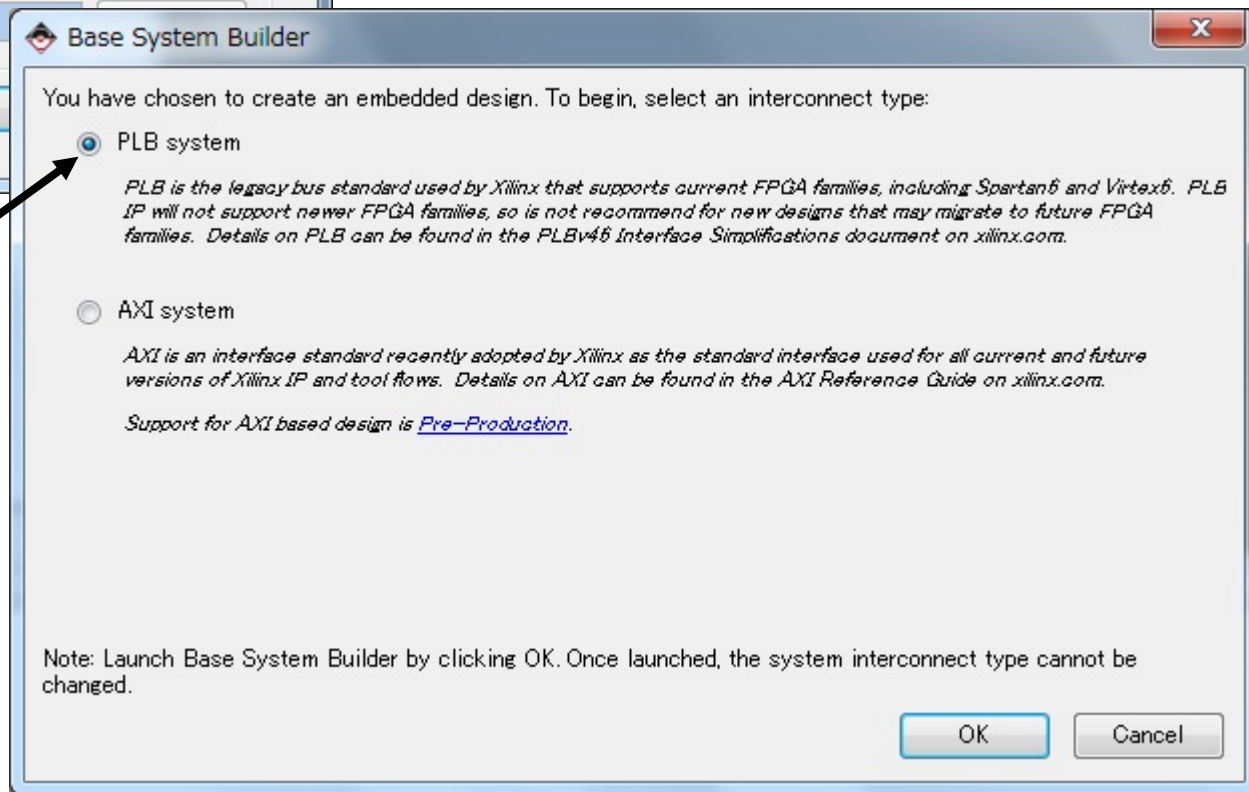
既存のプロジェクトを開く時

本講座では
ツールバージョン
EDK12.3i(ISE12.3i)
を対象としています。

BSB(Base System Builder)(2)

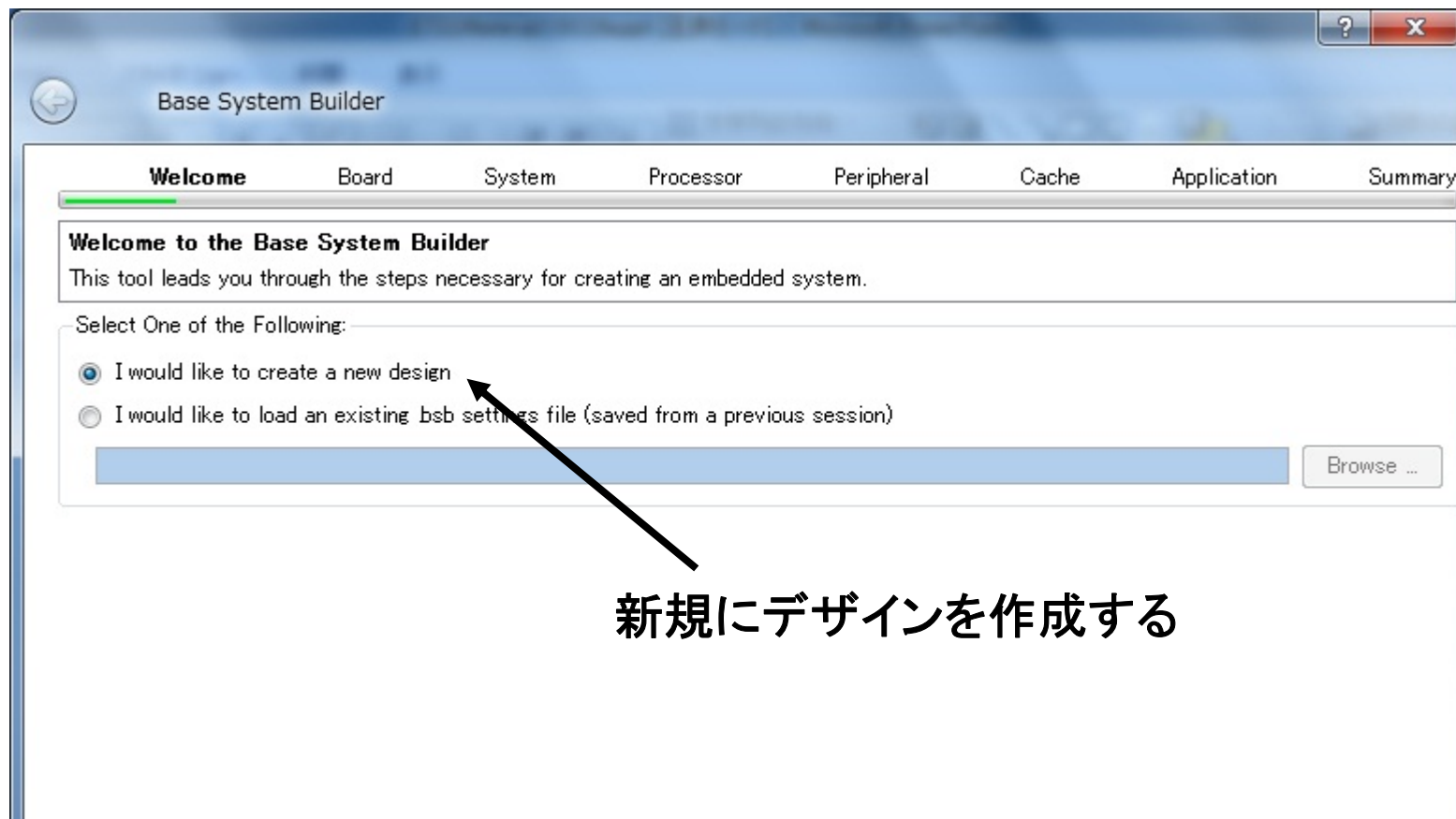


プロジェクトファイル名及びパス指定



PLBを使用する場合

BSB(Base System Builder)(3)



BSB(Base System Builder)(4)

The screenshot shows the 'Base System Builder' application window. At the top, there is a navigation bar with tabs: 'Welcome', 'Board', 'System', 'Processor', 'Peripheral', and 'Cache'. The 'Board' tab is currently selected and highlighted with a green underline. Below the navigation bar, the 'Board Selection' section is active. It contains the instruction 'Select a target development board.' and a 'Board' section with two radio button options. The first option, 'I would like to create a system for the following development board', is selected. Below this option are three input fields: 'Board Vendor' with the value 'Xilinx', 'Board Name' with the value 'Virtex 4 ML403 Evaluation Platform' (highlighted in blue), and 'Board Revision' with the value '1'. The second option, 'I would like to create a system for a custom board', is unselected. Below the radio buttons is the 'Board Information' section, which includes three dropdown menus for 'Architecture' (set to 'virtex4'), 'Device' (set to 'xc4vfx12'), and 'Package' (set to 'ff668'). There is also a checkbox for 'Use Stepping' which is unchecked, and a 'Reset Polarity' dropdown set to 'Active Low'. At the bottom of the form, there is a 'Related Information' section.

使用するボードを選択

一覧にない場合(カスタムボード)

BSB(Base System Builder)(5)

Base System Builder

Welcome Board **System** Processor Peripheral Cache Application Summary

System Configuration
Configure your system.

Single-Processor System
Select this option to create a design with a single processor. This Wizard will let you configure the processor, the peripheral set and some major configuration parameters for the peripherals.

Dual-Processor System
Select this option to create a design with two processors. This Wizard will let you configure the types of the processors, the peripherals accessible to the two processors and the peripherals shared by the two processors.

Processor 1

Processor 1 Peripherals
RS232 GPIO

Processor 1

Processor 1 Peripherals
RS232 GPIO

Processor 1

Processor 1 Peripherals
RS232 GPIO

Shared Peripherals
Mailbox Mutex

Processor 2

Processor 2 Peripherals
DDR EMAC

シングルプロセッサまたはデュアルプロセッサを選択

BSB(Base System Builder)(6)

Base System Builder

Welcome Board System **Processor** Peripheral Cache Application Summary

Processor Configuration
Configure the processor(s).

Reference Clock Frequency 100.00 MHz

Processor 1 Configuration

Processor Type PowerPC

Processor Clock Frequency 100.00 MHz

Bus Clock Frequency 50.00 MHz

On-chip Memory None

Debug Interface FPGA JTAG

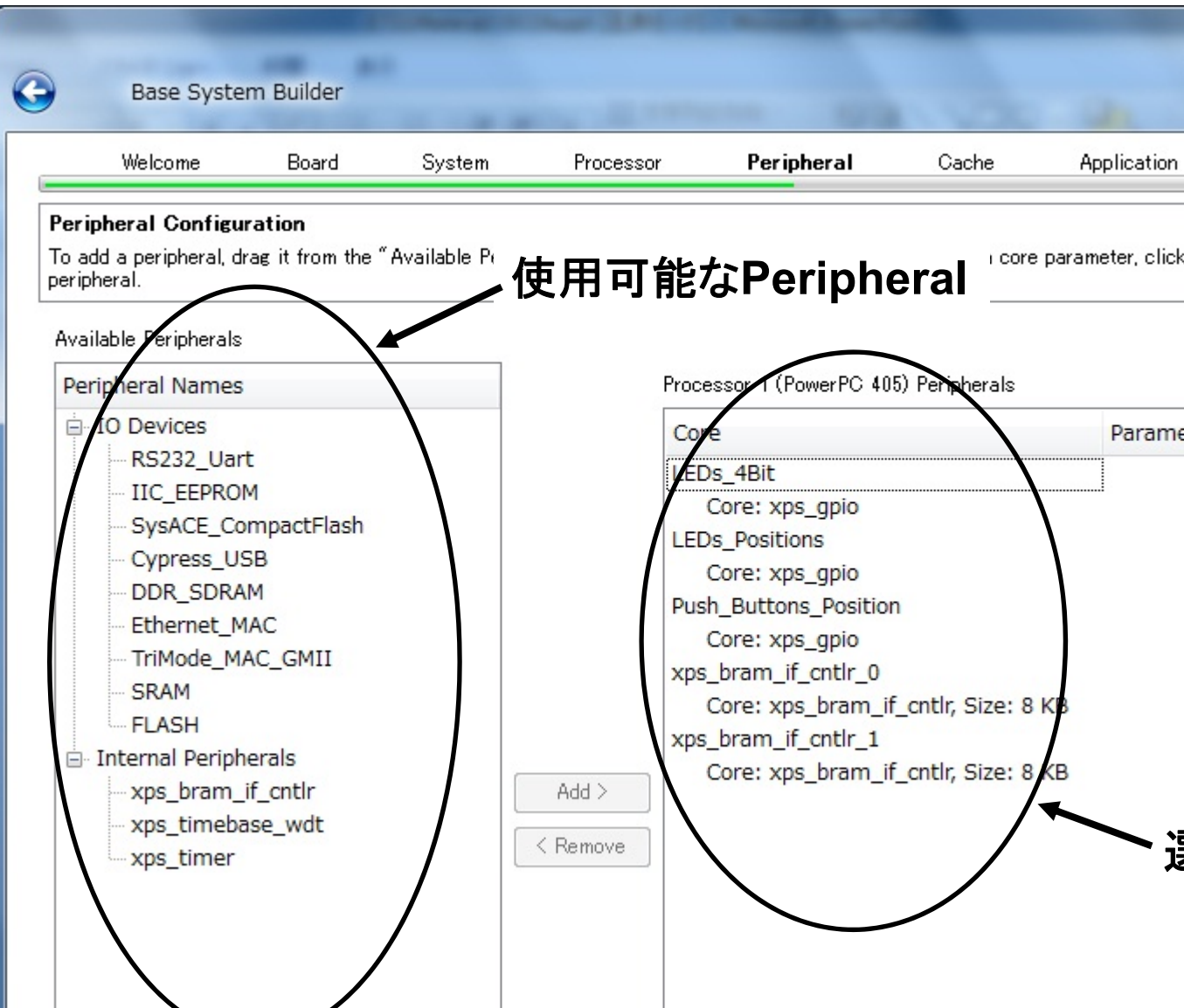
Enable Floating Point Unit

**PowerPCを選択
(MicroBlazeはPowerPCを
内蔵しないFPGAの時)**

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

OCMの使用/未使用

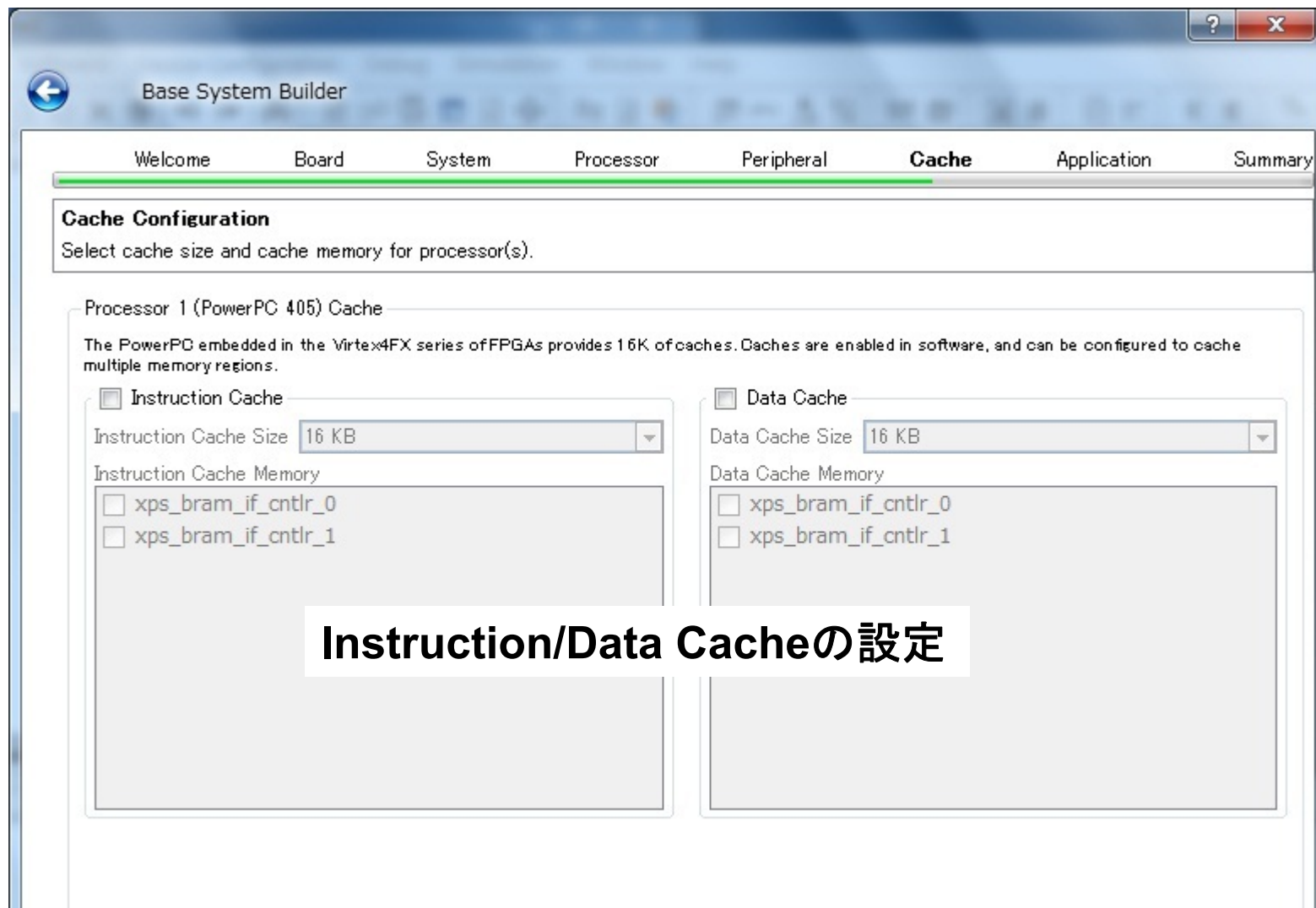
BSB(Base System Builder)(7)



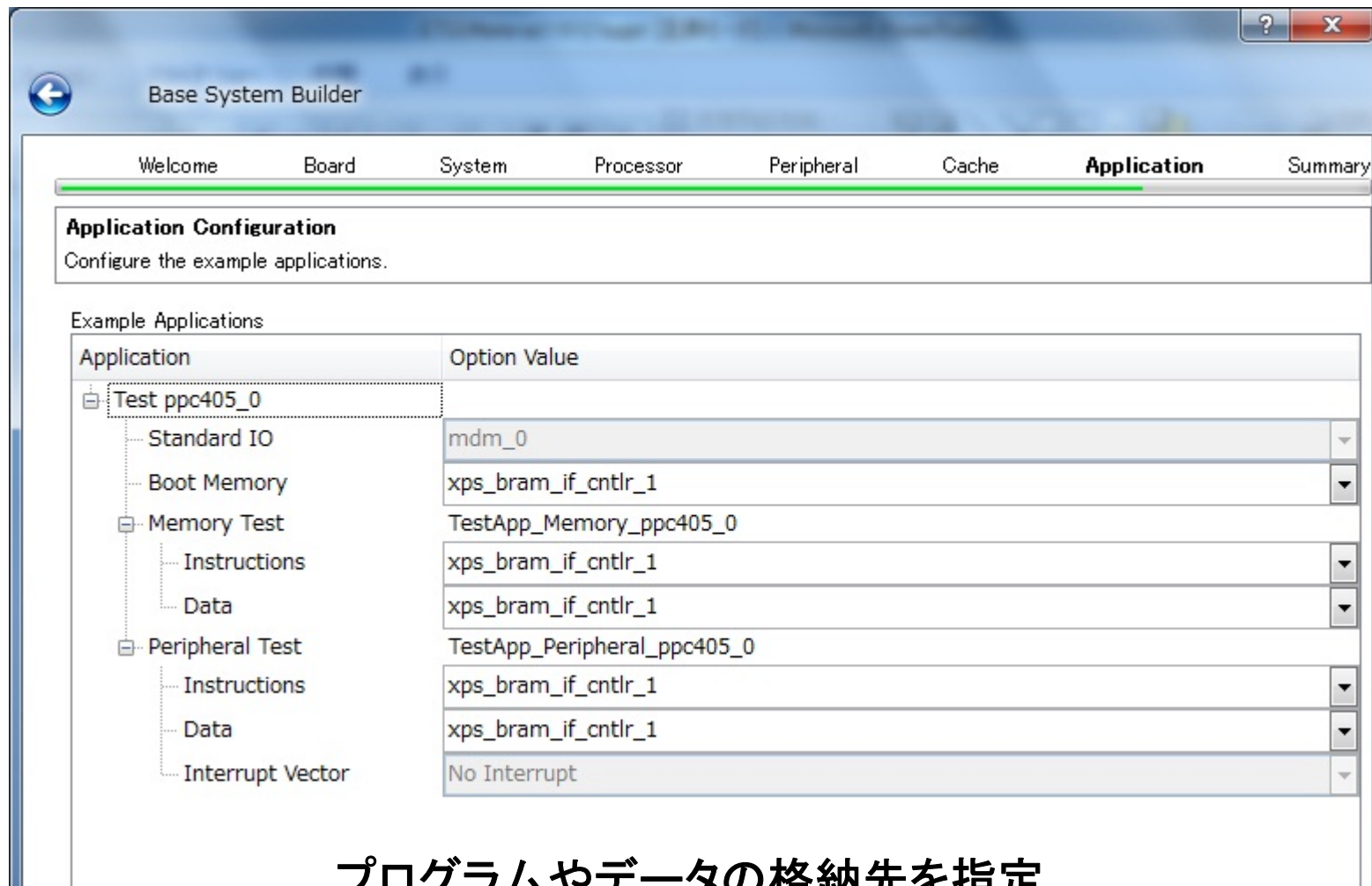
使用可能なPeripheral

選択済みのPeripheral

BSB(Base System Builder)(8)



BSB(Base System Builder)(9)



Base System Builder

Welcome Board System Processor Peripheral Cache **Application** Summary

Application Configuration
Configure the example applications.

Example Applications

Application	Option Value
[-] Test ppc405_0	
Standard IO	mdm_0
Boot Memory	xps_bram_if_cntlr_1
[-] Memory Test	TestApp_Memory_ppc405_0
Instructions	xps_bram_if_cntlr_1
Data	xps_bram_if_cntlr_1
[-] Peripheral Test	TestApp_Peripheral_ppc405_0
Instructions	xps_bram_if_cntlr_1
Data	xps_bram_if_cntlr_1
Interrupt Vector	No Interrupt

プログラムやデータの格納先を指定

BSB(Base System Builder)(10)

The screenshot shows the 'Base System Builder' application window. At the top, there is a navigation bar with tabs: 'Welcome', 'Board', 'System', 'Processor', 'Peripheral', and 'Ca'. The 'Processor' tab is currently selected. Below the navigation bar, there is a 'Summary' section with the text: 'Below is the summary of the system you are creating.' Underneath, there is a 'System Summary' section containing a table with the following data:

Core Name	Instance Name	Base Address	High Address
Processor 1	ppc405_0		
xps_gpio	LEDs_4Bit	0x81440000	0x8144FFFF
xps_gpio	LEDs_Positions	0x81420000	0x8142FFFF
xps_gpio	Push_Buttons_Position	0x81400000	0x8140FFFF
xps_bram_if_cntlr	xps_bram_if_cntlr_0	0x00000000	0x00001FFF
xps_bram_if_cntlr	xps_bram_if_cntlr_1	0xFFFFE000	0xFFFFFFFF

Below the table, there is a 'File Location' section with a tree view under 'Overall' showing the following file paths:

- C:\ETSS\MaterialNew\tmp\system.xmp
- C:\ETSS\MaterialNew\tmp\system.mhs
- C:\ETSS\MaterialNew\tmp\system.mss
- C:\ETSS\MaterialNew\tmp\data\system.ucf
- C:\ETSS\MaterialNew\tmp\etc\fast_runtime.opt
- C:\ETSS\MaterialNew\tmp\etc\download.cmd

Summary確認画面

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

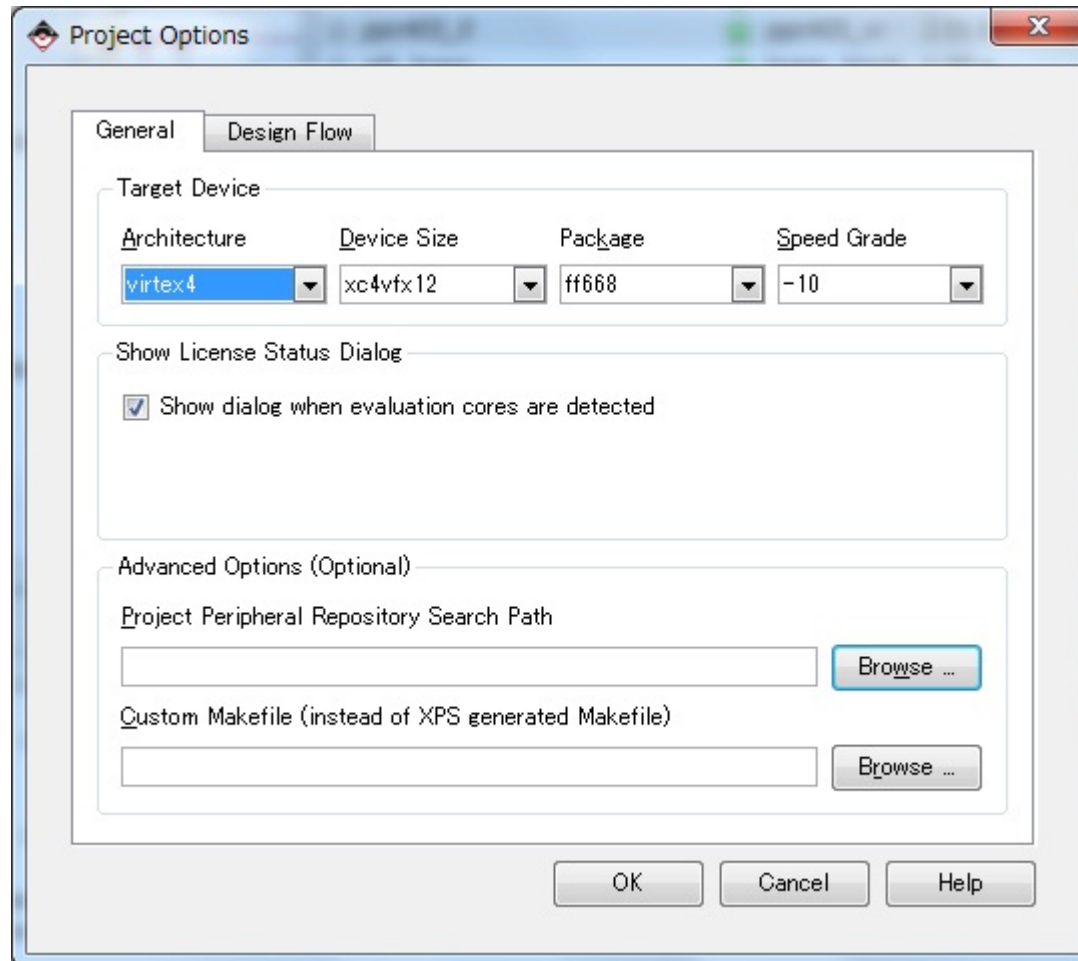
FPGAにダウンロード

- **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

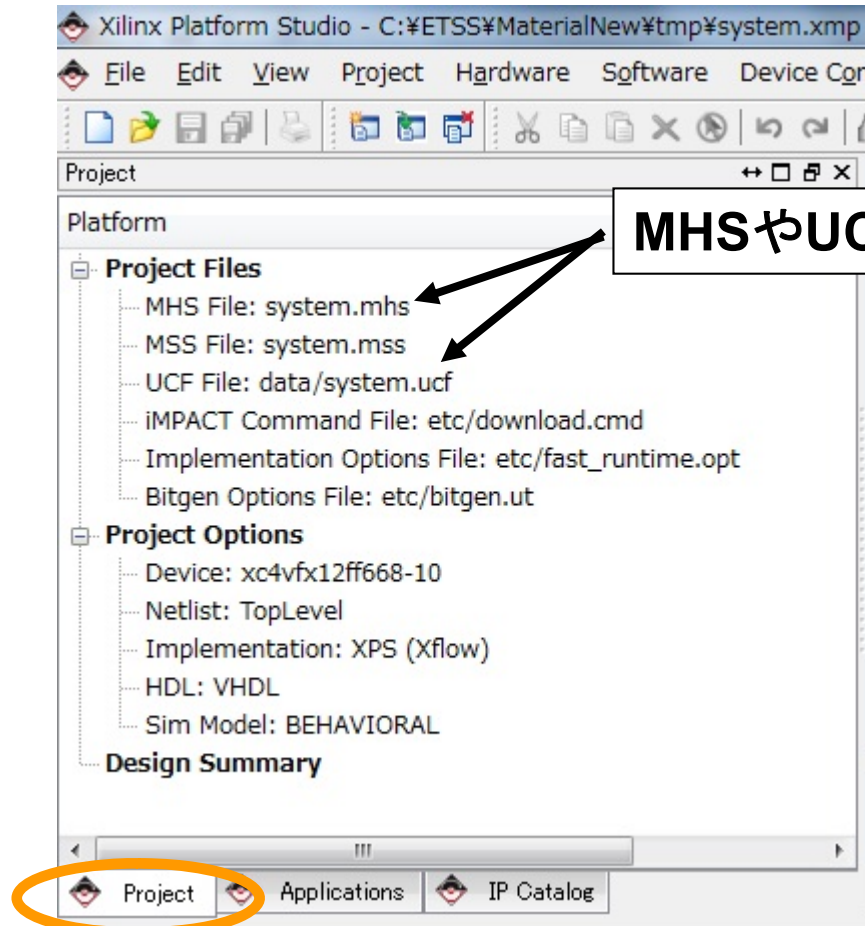
- Project Option

- ◆ メニューのProject → Project Options
- ◆ FPGAデバイスの変更やユーザIPの格納先を指定



Project タブ

- 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
```

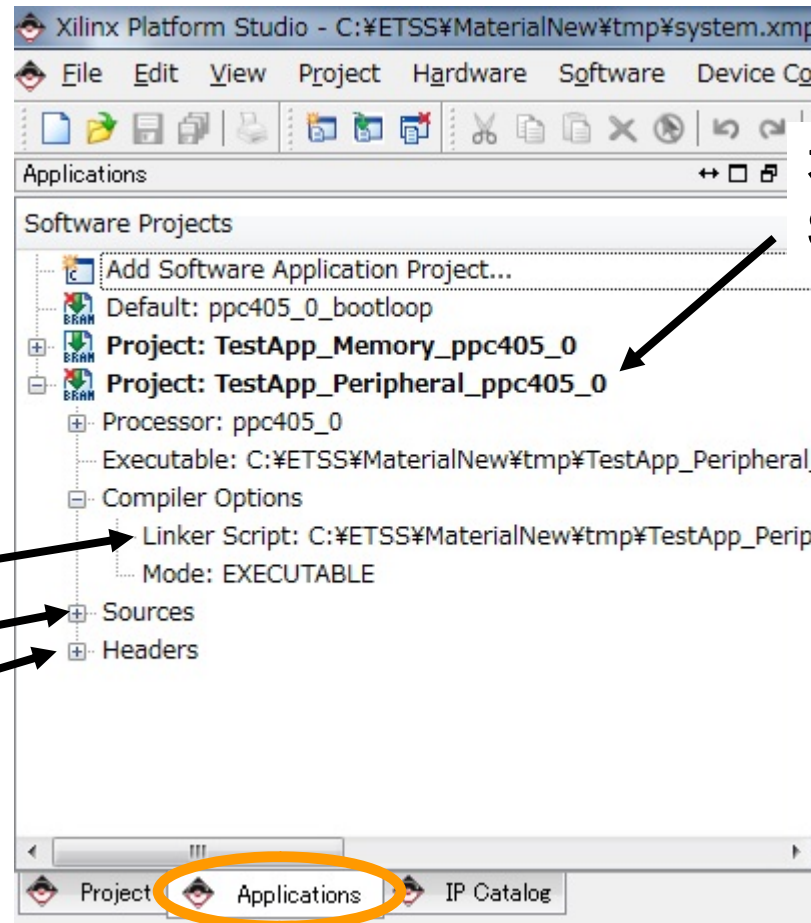
```
PORT Core_Reset_Req = C405RSTCORERESETREQ
```

```
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

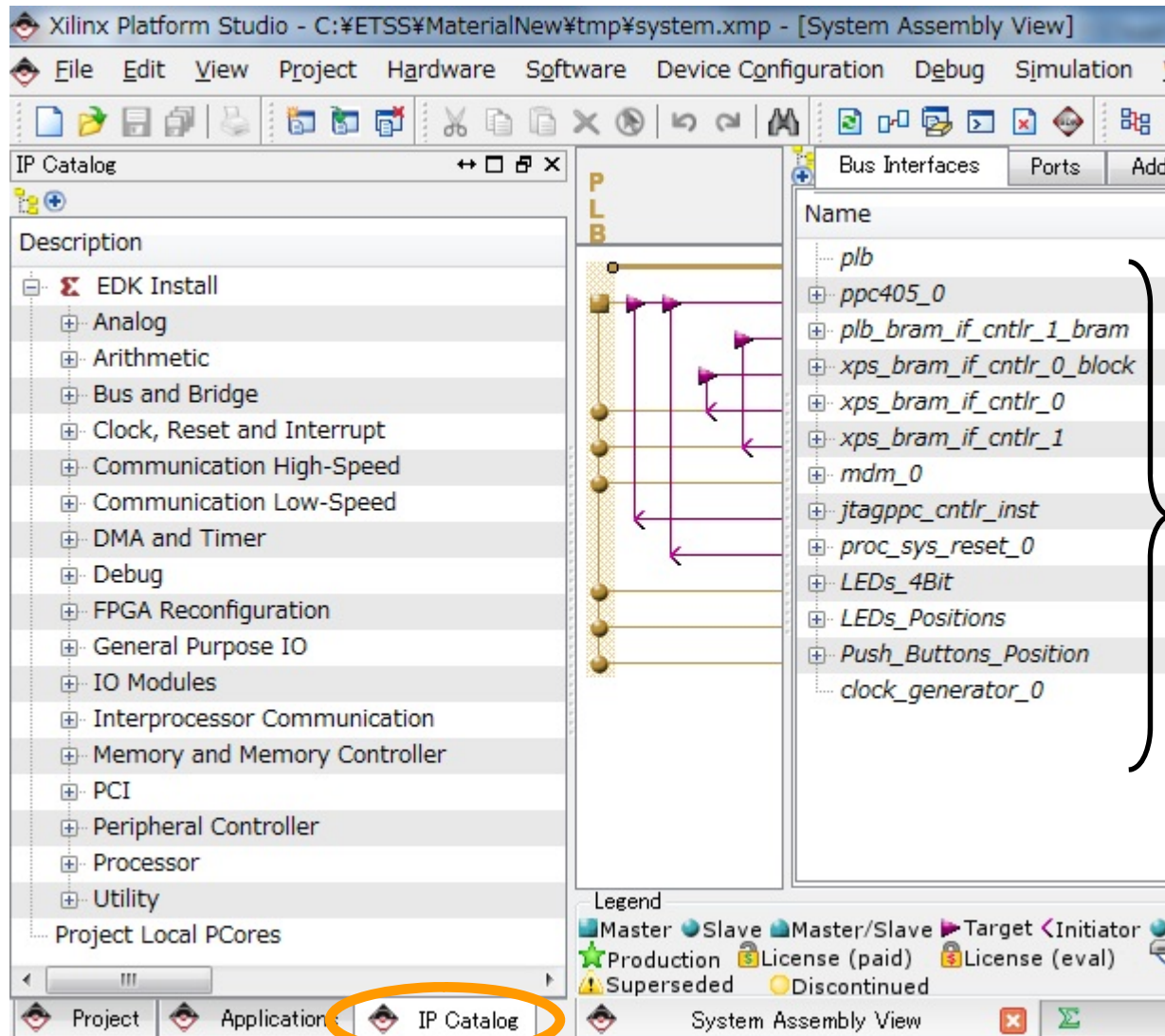
linker script,
source file,
header file



右クリックで
Set Compiler Options

コアの追加・削除

● コア (Peripheral) の追加・削除

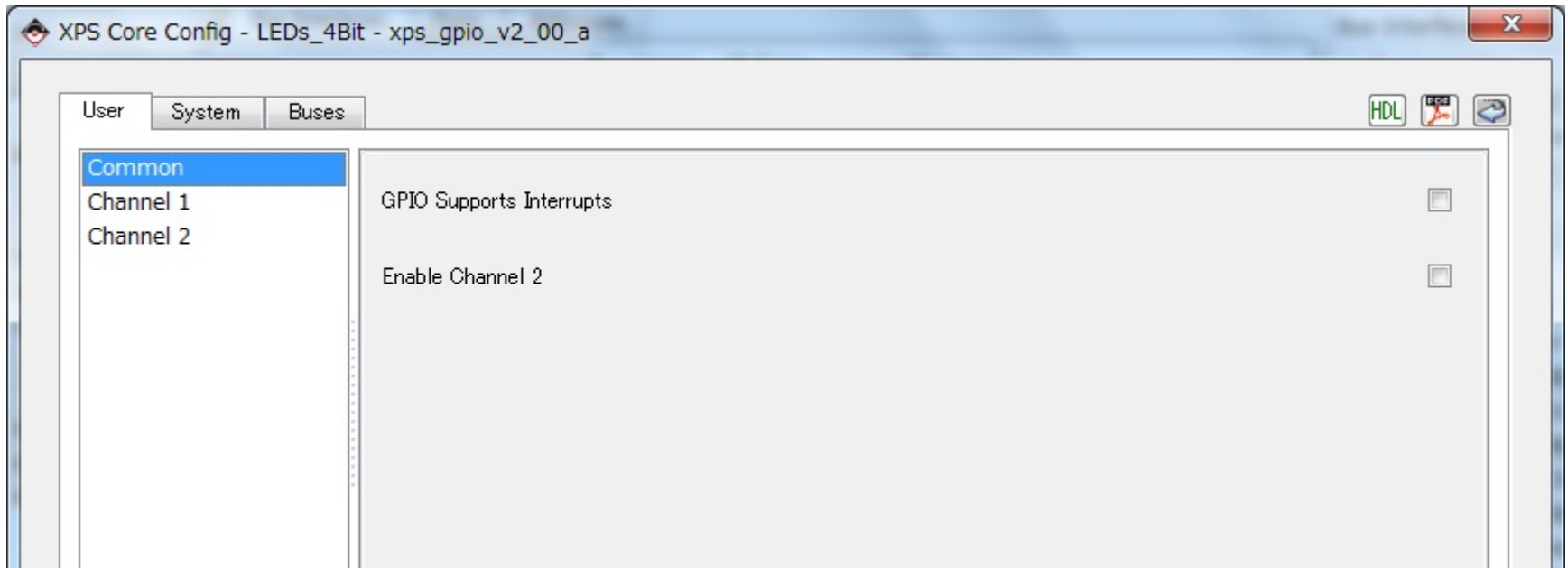


一覧から
コアを選択
して追加

使用中のコア
右クリックで
delete、設定

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
- デバッグ
 - ◆ メニューのDegug → Debug XMD、Debug Software Debuggerの実行でデバッグを接続可能
 - ◆ 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
 - ◆ 各種Peripheralを接続するバス

旧バージョンではペリフェラル用のバスとしてOPB(On-chip Peripheral Bus)が用意されていましたが、EDK12.1iでサポート対象外となっています。

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

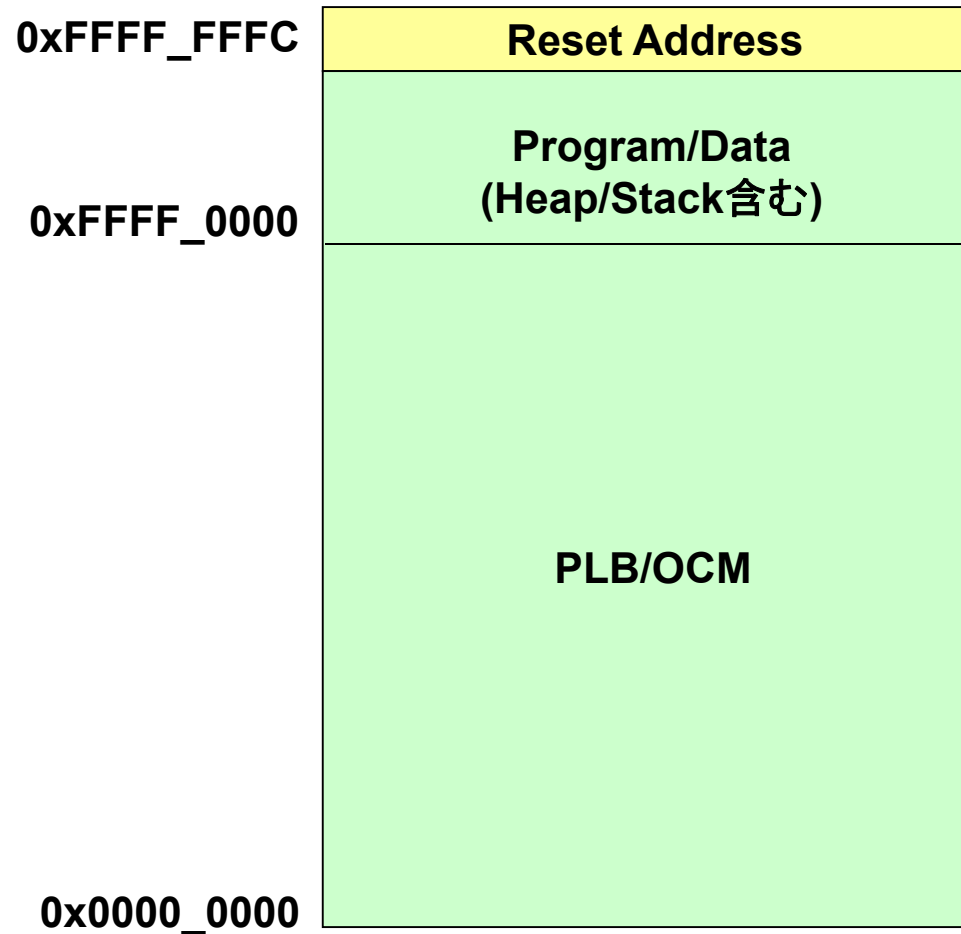
以下はEDKで使えるフリーのPeripheral(一部)

- 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！！

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

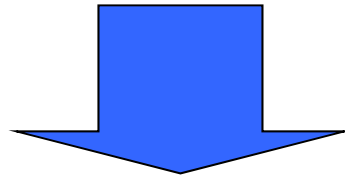
- 下記はアーキテクチャに特化した関数

```
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1KB。リンク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

リンクスクリプトを忘れてはならない

- 下記はリンクスクリプト一部

```
MEMORY
```

```
{
```

```
xps_bram_if_cntlr_1 : ORIGIN = 0x20000000, LENGTH = 0x00001FFF
```

```
xps_bram_if_cntlr_2 : ORIGIN = 0x40000000, LENGTH = 0x003FFFFFFF
```

```
}
```

```
SECTIONS
```

```
{
```

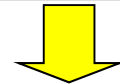
```
.text : { *(.text) } > xps_bram_if_cntlr_1
```

```
.data : { *(.data) } > xps_bram_if_cntlr_2
```

```
.bss : { *(.bss) *(COMMON) } > xps_bram_if_cntlr_2
```

```
}
```

RAMの容量を変更したら
ココも合わせなければならない



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

```
sys_status = XIntc_Connect( &intc,  
                           XPAR_XPS_INTC_PLB_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_XPS_INTC_PLB_USR_GNTLR_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_XPS_INTC_PLB_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);
```

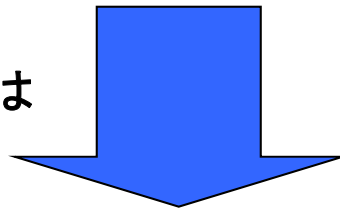
- GPIO割り込みクリア（割り込み処理終了時）

```
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

This tool will help you create templates for a new EDK CoreConnect peripheral, or help you import an existing EDK CoreConnect peripheral into an XPS project or EDK repository. The interface files and directory structures required by EDK will be generated.

新規作成 or 既存のIPをimport

ユーザIPの追加(3)

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

Repository: Browse...

To an XPS project

Project: Browse...

共有IP or 単一プロジェクト

Peripheral will be placed under:

More Info < Back Next > Cancel

ユーザIPの追加(4)

Create Peripheral

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

Enter the name of the peripheral (upper case characters are not allowed). This name will be used as the top HDL design entity.

Name: plb_codec_ctrl

Version: 1.00.a

Major revision: 1 Minor revision: 00 Hardware/Software compatibility revision: a

Description:

IP名称

IPバージョン

ユーザIPの追加(5)

Create Peripheral

Bus Interface

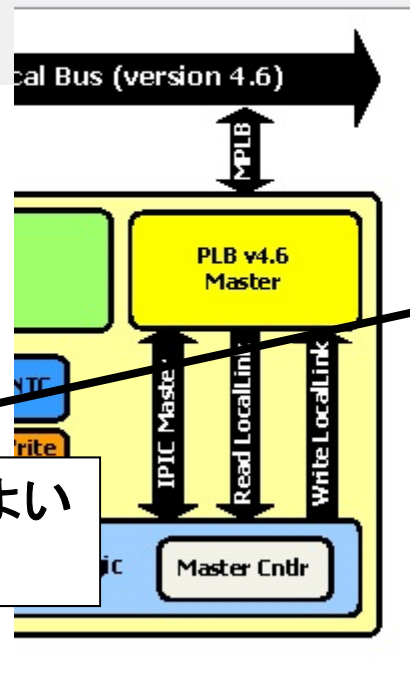
Indicate the bus interface supported by your peripheral.

To which bus will this peripheral be attached?

- Processor Local Bus (PLB v4.6)
- Fast Simplex Link (FSL)

使用するバスを指定

ed to the PLB (v4.6) interconnect through corresponding PLB IP Interface (IPIF) modules, which provide you the interface between the PLB interconnect and the user logic. Besides the standard functions like address : IPIF module, the wizard tool also offers other commonly used services and configurations to simplify the



通常はデフォルト設定でよい
(Interruptは外しておく)

Slave service and configuration

Typically required by most peripherals for operations like logic control, status report, data buffering, multiple memory/address space access, and etc. (PLB slave interface will always be included).

- Software reset
- Read/Write FIFO
- Interrupt control
- User logic software register
- User logic memory space
- Include data phase timer

Master service and configuration

Typically required by complex peripherals like Ethernet and PCI for commanding data transfers between regions (PLB master interface will be included if master service selected).

- User logic master

ユーザIPの追加(6)

Create Peripheral

Slave Interface
Configure the slave interface of your peripheral

The IPIF slave library provides an interface between the user logic and the PLB address decoding and implements the protocol and timing transactions between the user logic and the IPIC (IP InterConnect) interface between user logic and IPIF.

Slave performance

Slave peripherals support single beat read/write data transfers by default. If performance is key to the slave controller, you can have the burst transfer support turned on - this feature provides higher data transfer rates and enables the transfer protocol for PLB Fixed Length Burst operations.

Burst and cache-line support

Data width

The native bit width of non-burst slaves is as the smallest native data width.

バースト転送する時

Create Peripheral

User S/W Register
Configure the software accessible registers in your peripheral.

The user specific software accessible registers will be implemented in the user-logic module of your peripheral. Such registers are typically provided for software programs to control and to monitor the status of your user logic. These registers are addressable on the byte, half-word, word, double word or quad word boundaries depending on the user logic for register read/write will be included in the user-logic module generated by the wizard tool for your reference.

レジスタ数

User logic software registers may take full advantage of the slave IPIF address-decoding service to generate CE decodes for all of the individual register of interest. The diagram on the left shows the simplest set of IPIF slave signals to read/write the registers.

Number of software accessible registers: (1 to 4096)

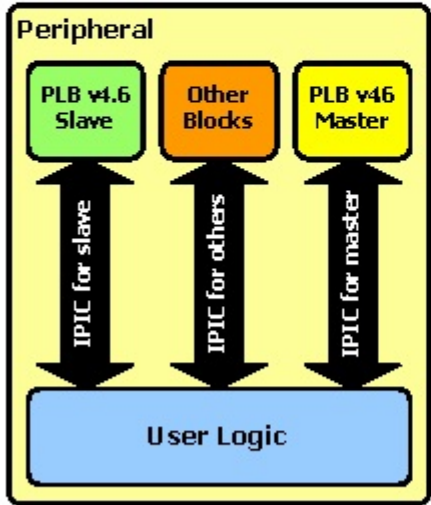
ユーザIPの追加(7)

Create Peripheral

IP Interconnect (IPIC)
Select the interface between the logic to be implemented in your peripheral and the IPIF.

Your peripheral will be connected to the PLB (v4.6) interconnect through suitable IPIF master/slave module(s). Your custom logic from the user-logic module interfaces to the IPIF module(s) and other sub-blocks through a set of signals called the IP interconnect (IPIC) interface. Some of the ports are always present, some are pre-selected based on the IPIF services you required, and you can choose other optional ports to be included in the design based on your needs.

Note: all IPIC ports are active high.



Port description

- Bus2IP_Clk
- Bus2IP_Reset
- Bus2IP_Addr
- Bus2IP_CS
- Bus2IP_RNW
- Bus2IP_Data
- Bus2IP_BE
- Bus2IP_RdCE
- Bus2IP_WrCE
- IP2Bus_Data
- IP2Bus_RdAck
- IP2Bus_WrAck
- IP2Bus_Error

IP内部で接続するネットの指定
(通常はデフォルトでよい)

ユーザIPの追加(8)

Support
using Bus Functional Models (BFM).



form to help you simulate your peripheral. Indicate if you want this tool to generate a stimulus file for the target bus.

ModelSim用simulationモデル生成

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 package. The link below shows how:

[BFM Package Installation Instructions](#)

Create Peripheral

(OPTIONAL) Peripheral Implementation Support
Generate optional files for hardware/software implementation

Upon completion, this tool will create synthesizable templates. You will need to complete the implementation of this module using standard HDL design flows. The tool will also generate EDK interface files (mpd/pao) for the synthesizable templates, so that you can hook up the generated peripheral to a processor system.

Verilogで記述している場合はチェックする

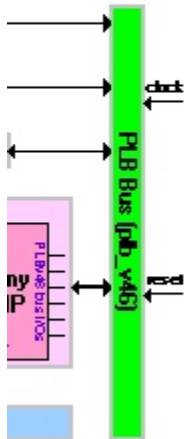
Note
Should the peripheral interface (ports/parameters) or file list change, you will need to regenerate the EDK interface files using the import functionality of this tool.

- 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

Peripheral (VHDL)

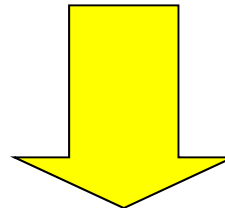
IPIF (VHDL)

User Logic (VHDL)



ユーザIPの追加(9)

- この時点では、まだユーザIPのラッパー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

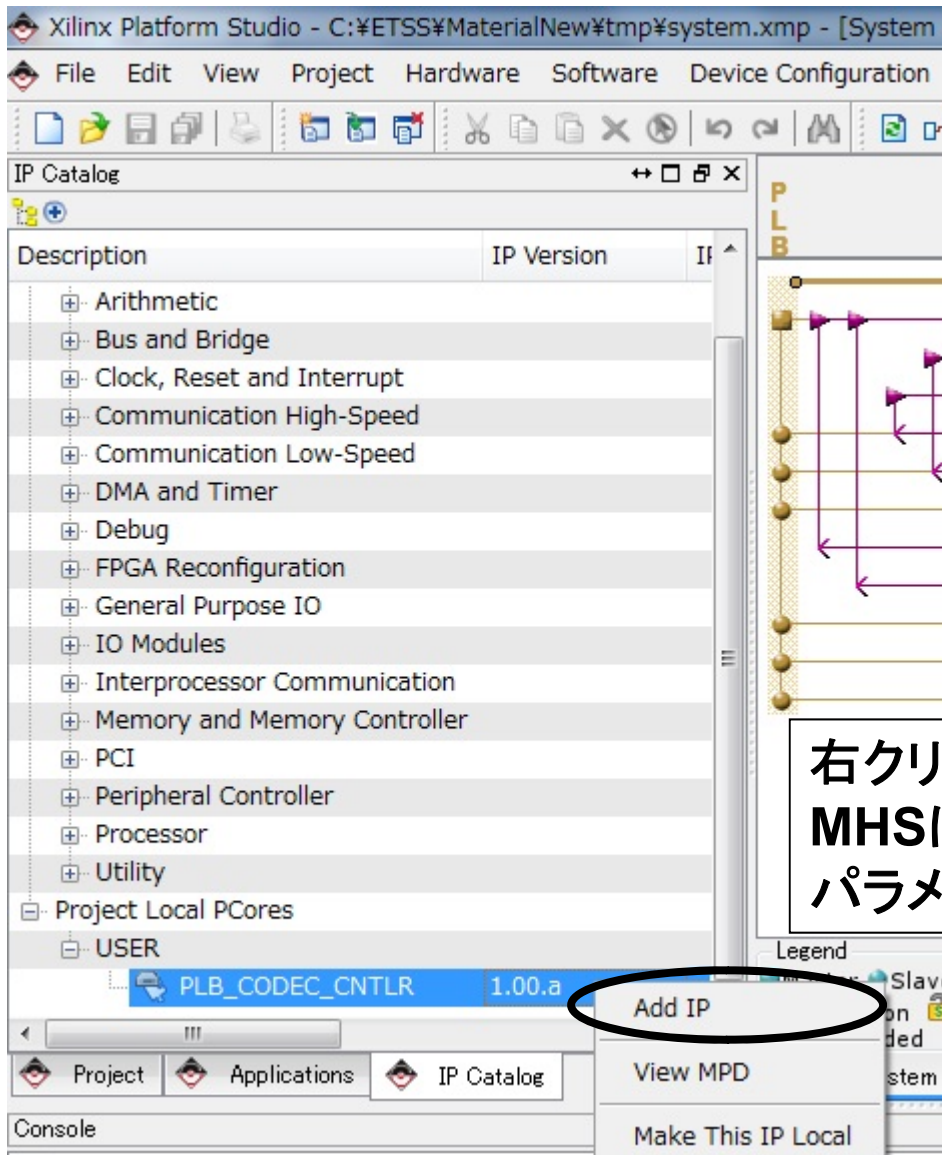
- ◆ ユーザIPで使用するファイル(VHDL, Verilog)をリストする

◆ lib opb_usrip_v1_00_a user_logic vhdl

↑ ↑ ↑ ↑

予約語 IP名称 ファイル名 言語
(ver.込み) (拡張子なし)

ユーザIPの追加(10)



右クリックでAdd IPを実行。
MHSに反映されるので、ポートの追加や
パラメータの追加・変更をする

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

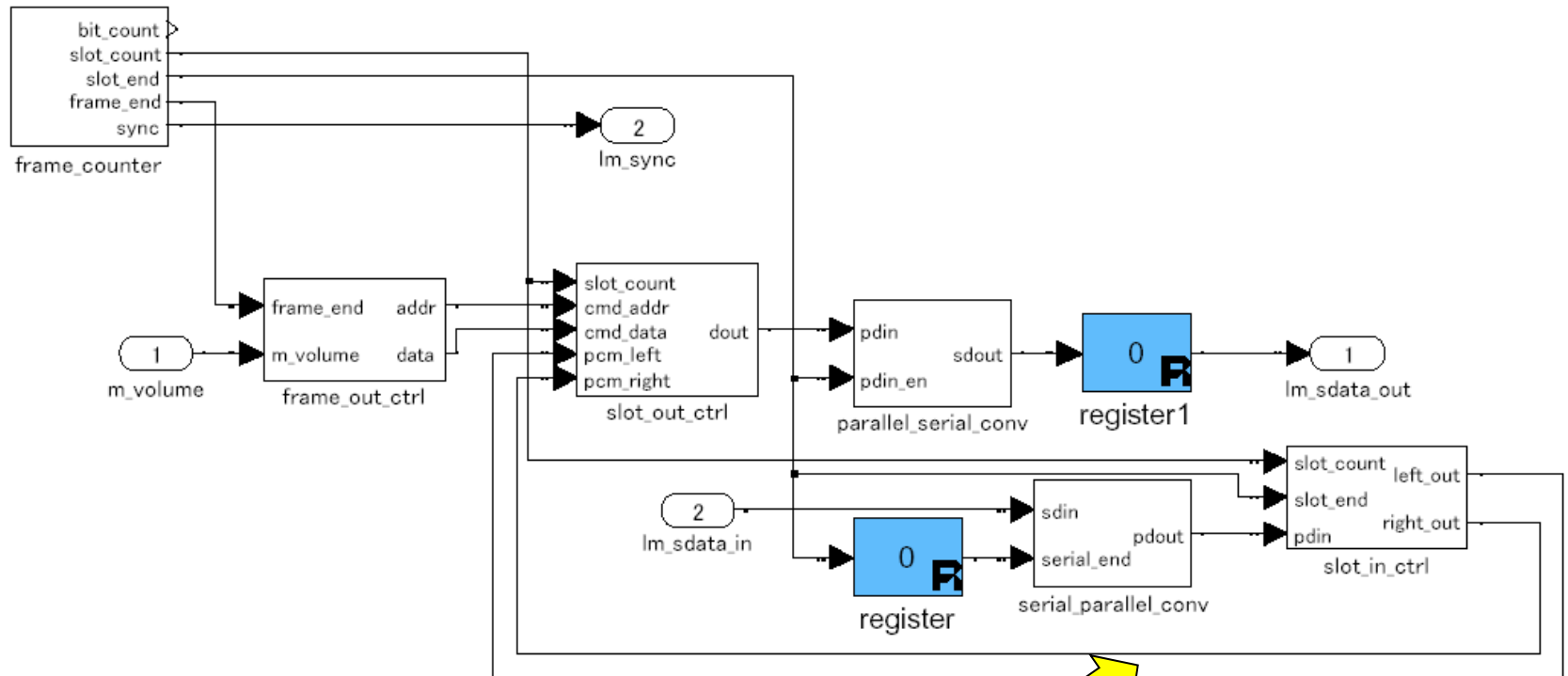
既に設計済みのRTL回路にソフト処理を挿入する。
⇒PowerPCを入れ、ソフト処理を行うフェーズを追加する。
つまり、組み込みシステムを完成させる！！

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

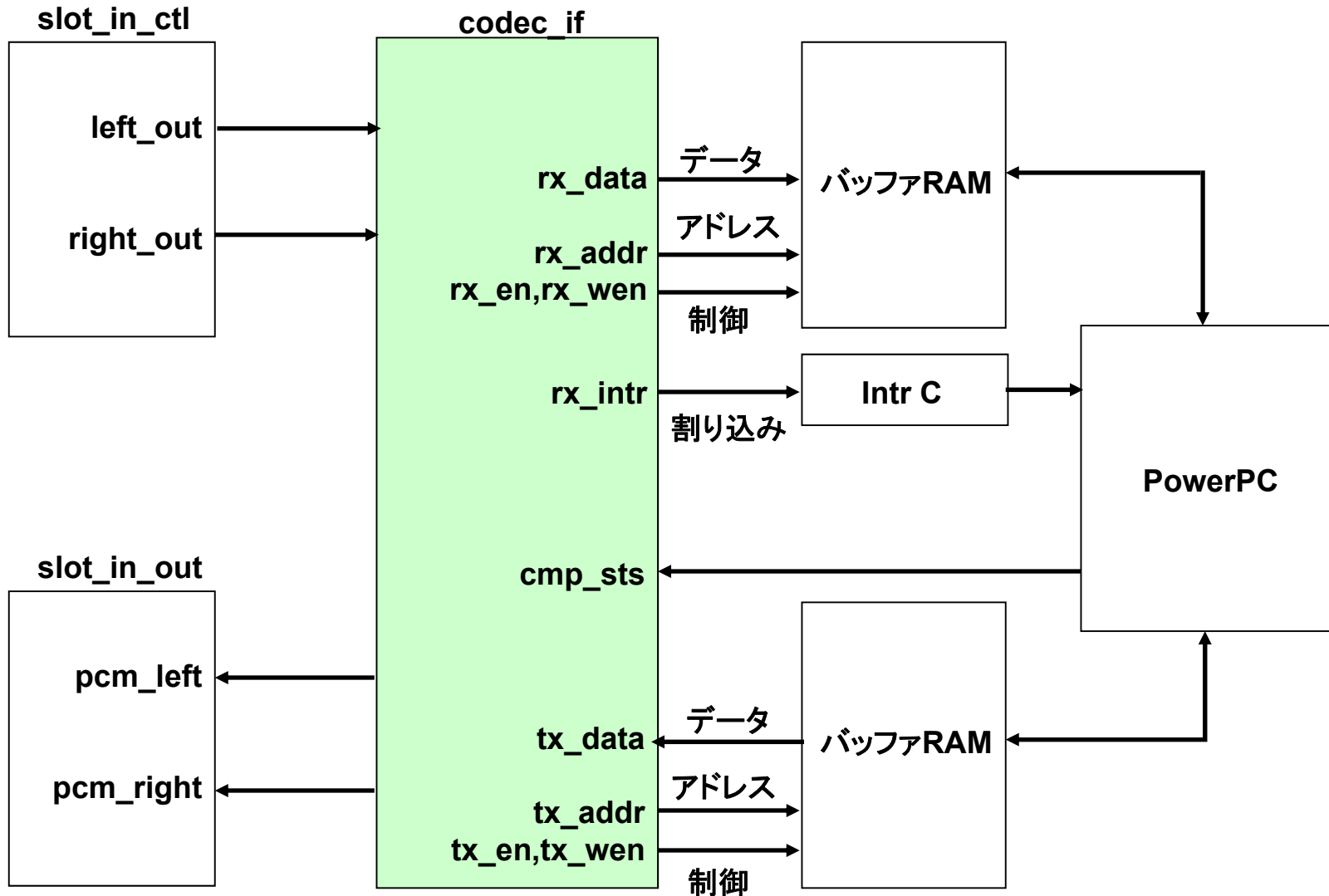
1. バッファRAMを使い、データをスルーさせる仕組みを作る。
→動作的には変化なし
2. ノイズを混入するソフト処理を実装する。
ON/OFF機能あり
3. LCDに情報を表示する
4. (ノイズフィルタを実装する)

Step1

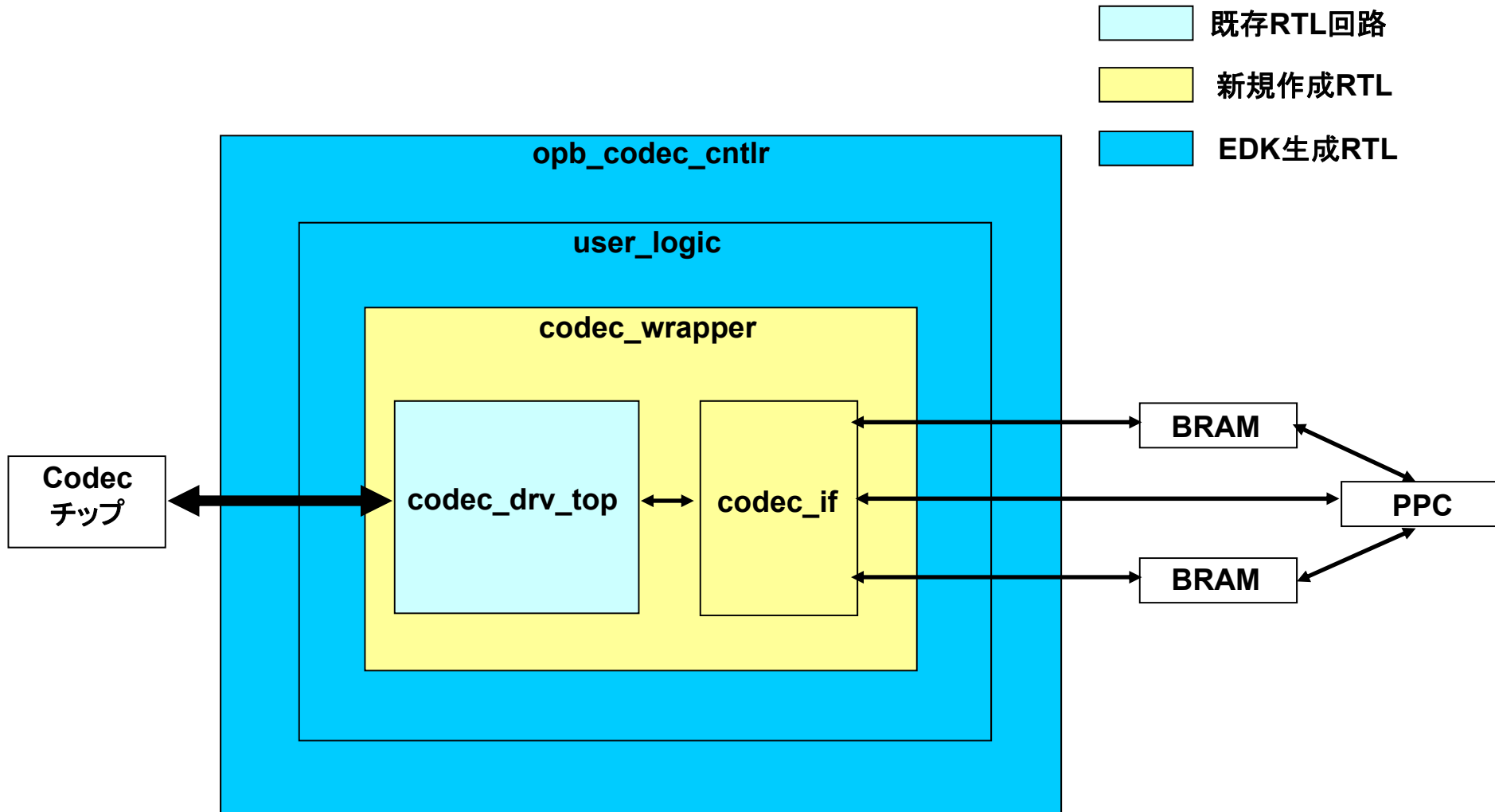
バッファRAMを使い、データをスルーさせる仕組みを作る

- 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

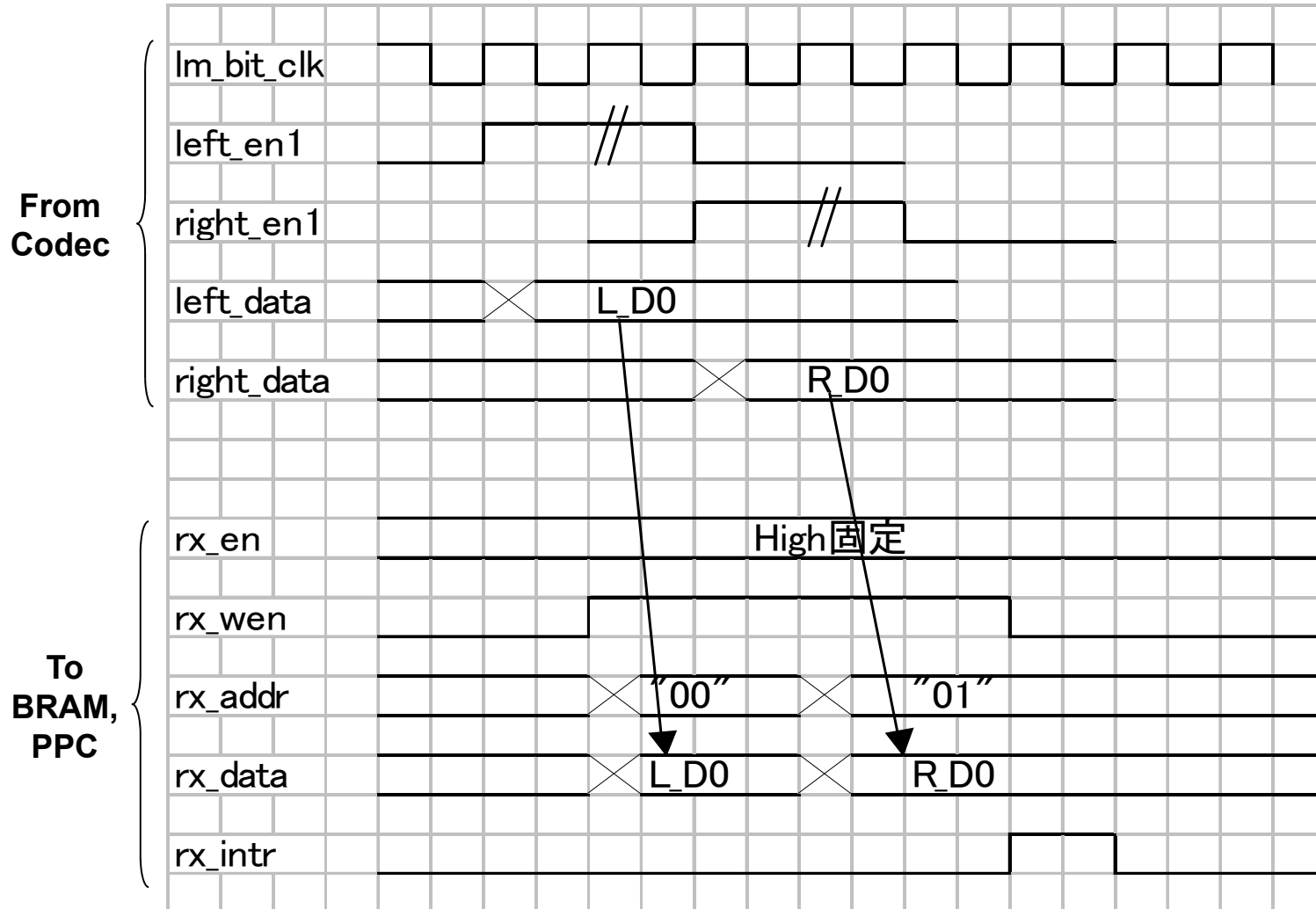


階層構造



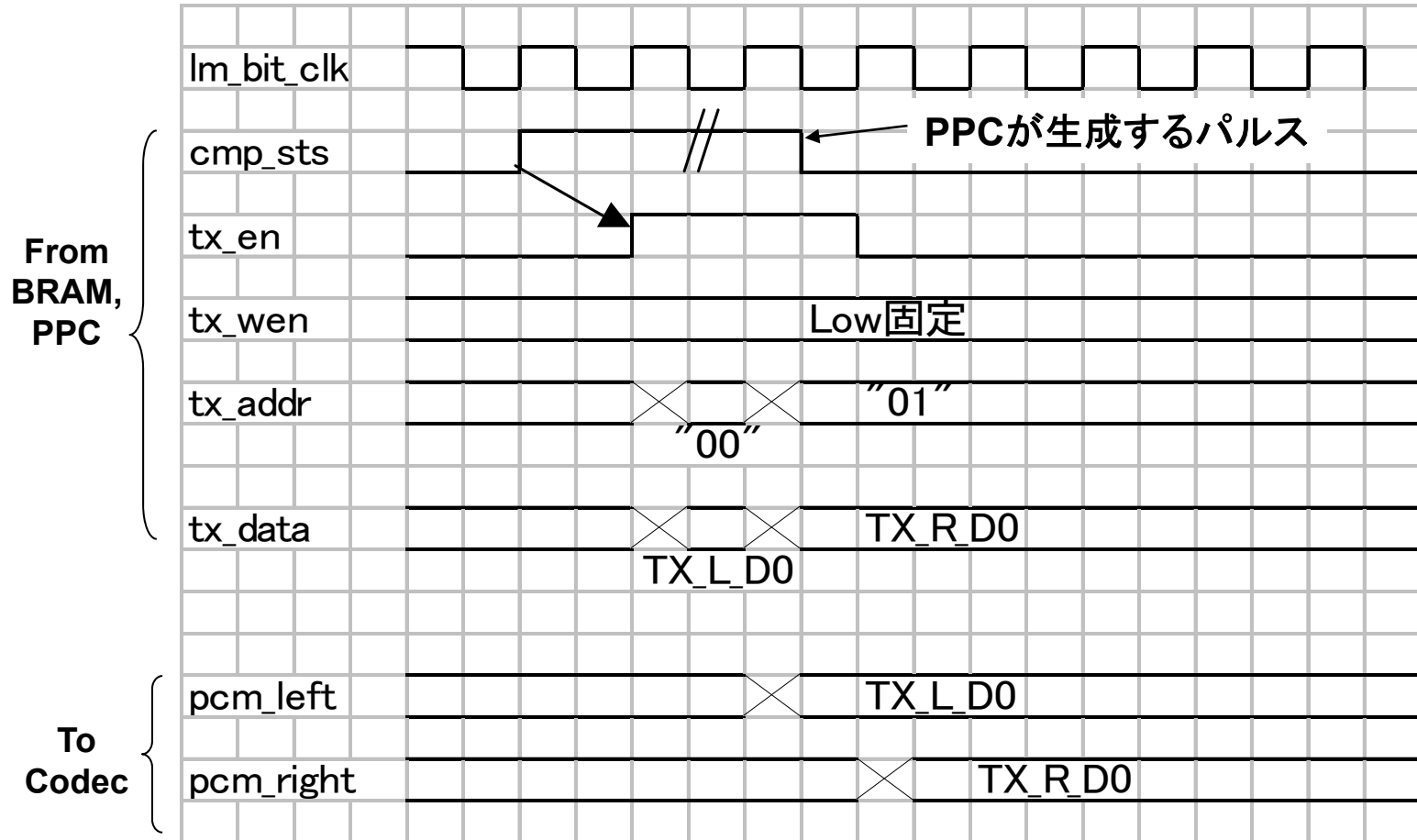
タイミングチャート(1)

- 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

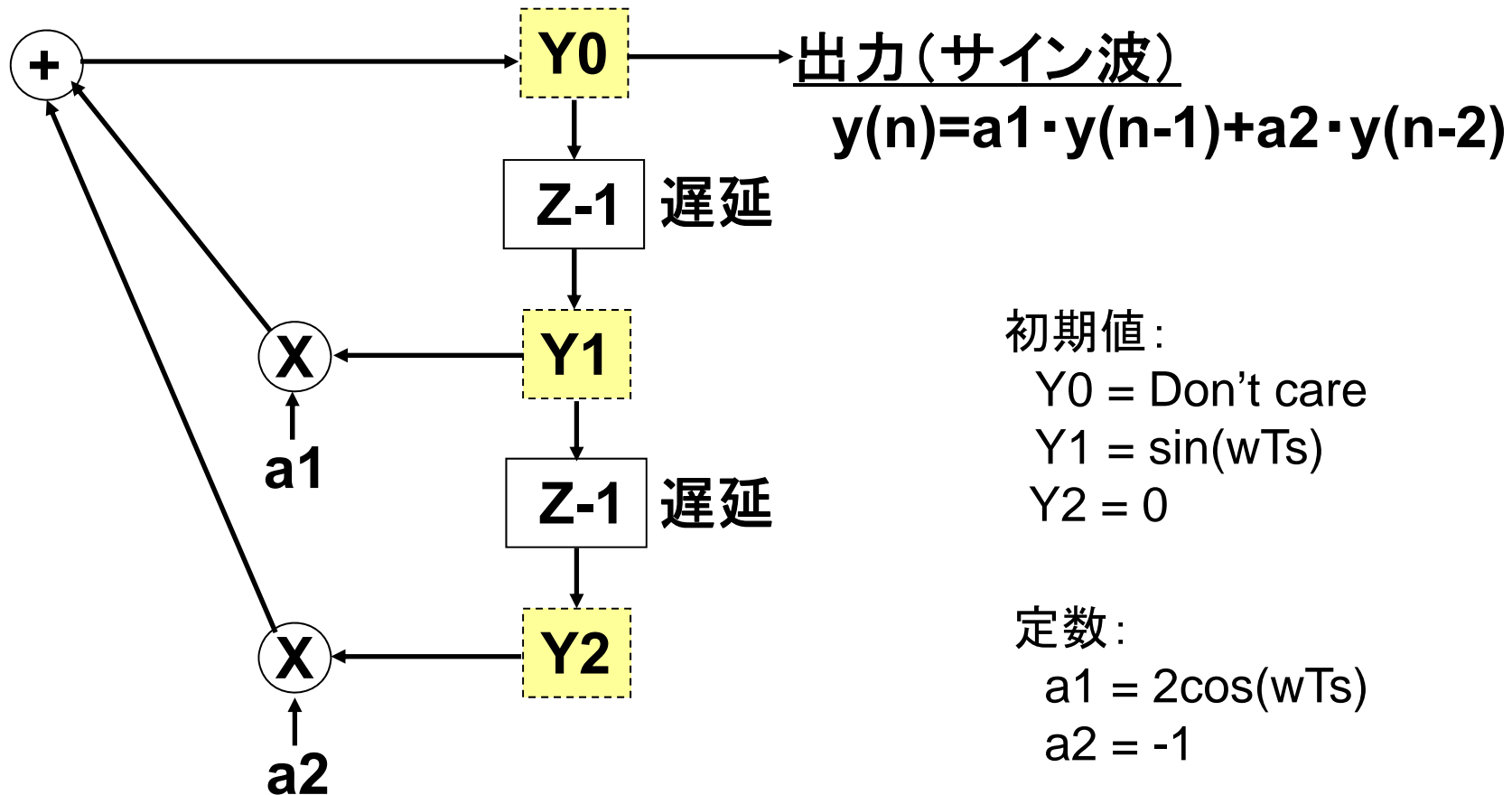
Step2

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:

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

$$T_s = 1 / 8000$$

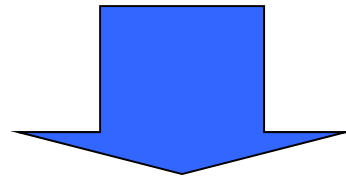
$$a1 = 2 \cos(w T_s) = 2 \cos(880 * \text{PI} / 8000) = 1.8817615$$

$$a2 = -1$$

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

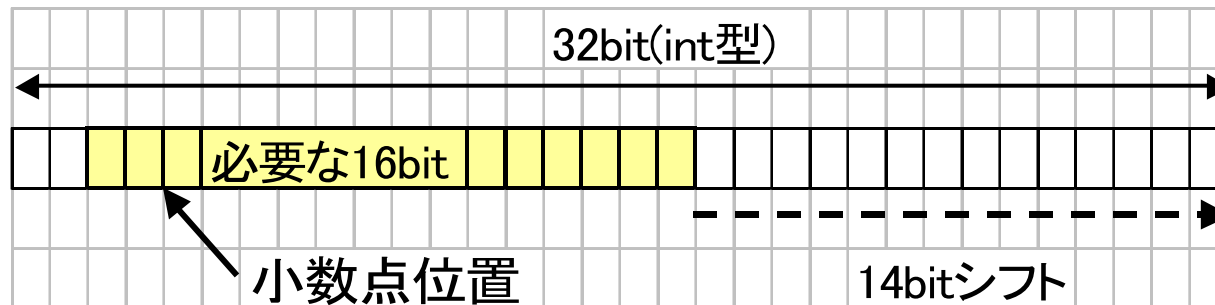
- 演算部分も変換する必要がある

$y[0] = a1*y[1] + a2*y[2];$ //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

Step3

LCDパネルに以下の情報を表示する

- 混入しているサイン波「ノイズ」の周波数
- サイン波「ノイズ」の状態(ONまたはOFF)
 - ◆ プッシュスイッチを押している間はノイズOFF
 - ◆ プッシュスイッチを離している間はノイズON

S	i	n	e	W	a	v	e	:	X	X	K	H	z		
N	o	i	s	e	:	O	N								
S	i	n	e	W	a	v	e	:	X	X	K	H	z		
N	o	i	s	e	:	O	F	F							

LCDパネル表示例

LCDの仕様

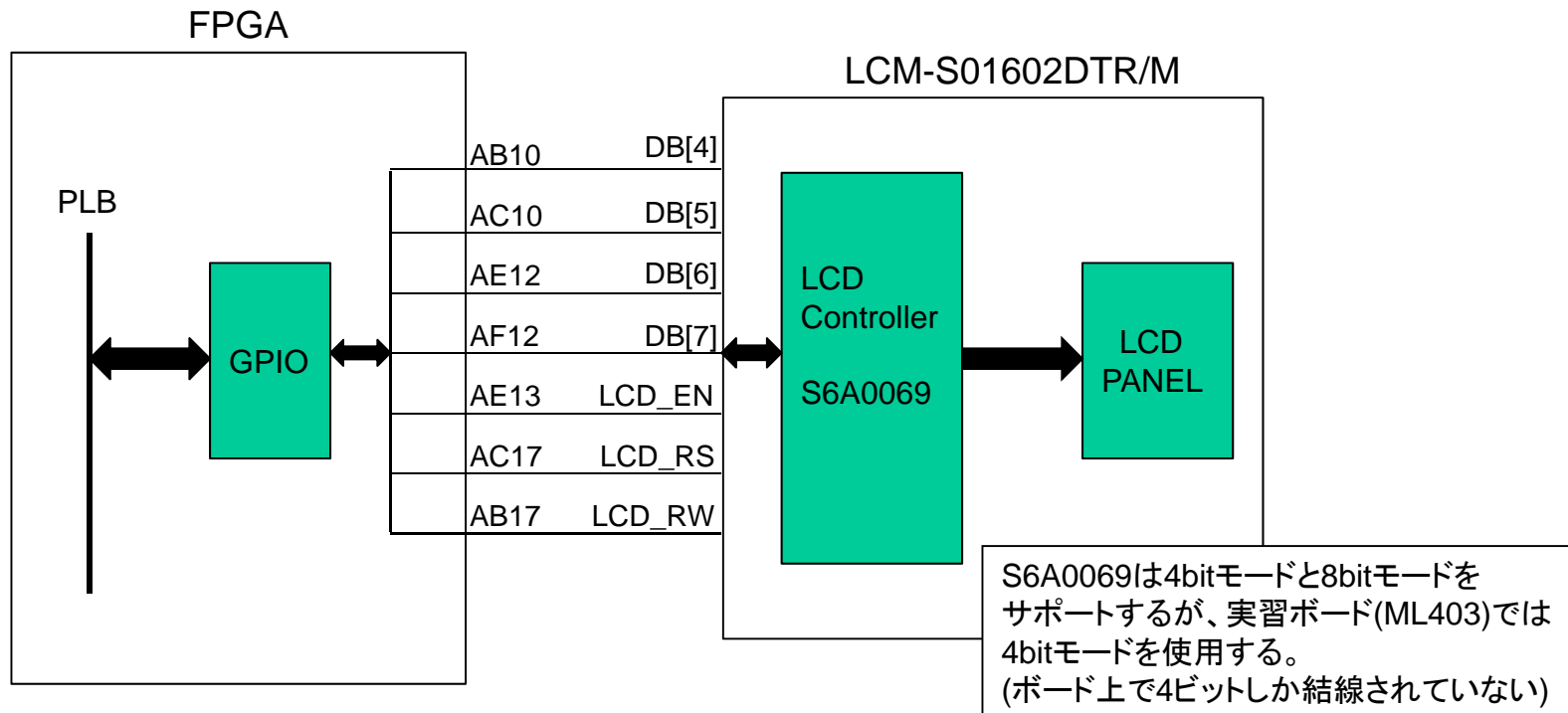
- 実習ボード上のLCDモジュール

✓ 型名 : **LCM-S01602DTR** ※実習ボード(ML403)ユーザガイドより

LCDコントローラ : S6A0069(SAMSUNG)

LCDパネル : 16文字 2行

- FPGAとLCDモジュールの接続



LCDの使い方(1)

- S6A0069(LCDコントローラ)の制御
 - ◆ 4ビットモード
8ビットモードでは1回でDB[7:0]の書き込みを行うが、
4ビットモードではDB[7:4]、DB[3:0]の2回に分けて書き込む。

 - ◆ 初期化
LCDは初期化が必要。
初期化シーケンスはデータシートを参照。
(初期化時に4ビットモードか8ビットモードかを選択する)

LCDの使い方(2)

- **S6A0069(LCDコントローラ)の制御**
 - ◆ インストラクション(カーソル移動、LCD表示をクリアする等)
インストラクションの種類はデータシートを参照。
 - ◆ 文字の表示
アルファベット等の文字列は以下のように直接表示させる事が可能。

LCD_Display(&lcd, "LCD Test");

<LCD_Displayはユーザ定義の関数>

数字の場合は、アスキーコードへの変換が必要。

例えば'0'を表示したい場合、0x30をDB[7:0]でコントローラに書き込む。

アスキーコードはデータシート参照。

※S6A0069のデータシートにはアスキーコードの記載がないので、他のコントローラのデータシートを参照する(例えば、ST7066U Sitronix社)

LCDの使い方(3)

- ビットアサイン

4ビットモードを使うのでビットアサインは以下の通りとする。

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
-	E	RS	R/W	DB7[DB3]	DB6[DB2]	DB5[DB1]	DB4[DB0]

E=Read/Write Enable

‘1’の時、disble

‘0’の時、enable

RS=Register Select

‘1’の時、データ

‘0’の時、インストラクション

R/W=Read/Write

‘1’の時、 read operation

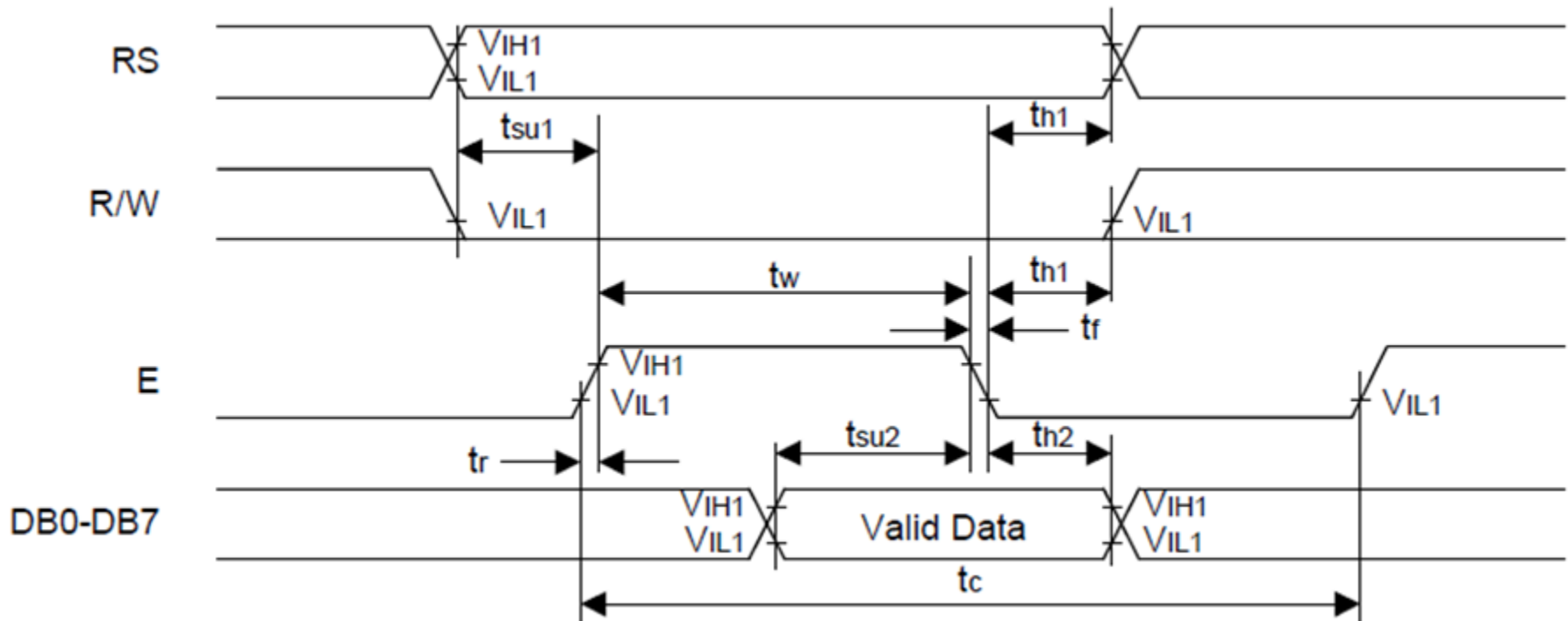
‘0’の時、 write operation

詳細はデータシートを参照

LCDの使い方(4)

- タイミングチャート

初期化、インストラクション、データの書き込みはデータシート記載のタイミングチャートに従ってください。



詳細はデータシートを参照

演習Lab6を行ってください