

Vivado/インタフェース生成ツールチュートリアル

高木研究室 谷

準備

- Vivado 2015.4のインストール
 - SDKのインストールも必要
 - Vivado 2015.4, Vivado HLS 2015.4, Xilinx SDKの3種類がインストールされているはず
- スクリプトの動作環境
 - LinuxなりCygwinなりでpythonとpython-clangが動く環境を作っておいてください

準備

- スクリプトの動作環境：実行を確認してるもの
 - Linux Mint 17.3/18 (Debian系なら大丈夫だと思います)
 - Cygwin
- Linuxでの環境構築
 - `$ apt-get install python-pip`
 - `$ apt-get install clang`
 - `$ apt-get install python-clang-3.8`
 - `$ pip install jsonschema`
 - `$ pip install clang`
 - 別のバージョンのpython-clangを使用する場合は
extractparameter.py と divider/analyzer.py の
Config.set_library_fileの設定パスのバージョンを変更してください

準備

- Cygwinでの環境構築
 - Cygwin64インストーラから, python 2.7.10/clang 3.8.1がインストールされていることを確認
 - `$ easy_install-2.7 --version`
setuptools 15.2
 - `$ easy_install-2.7 pip`
Searching for pip ...
 - `$ pip --version`
pip 8.1.2 from /usr/lib/python2.7/site-packages/pip-8.1.2-py2.7.egg (python 2.7)
 - `$ pip install jsonschema`
 - `$ pip install clang`

準備

入カソースC

- C言語サンプル(matrixmul.c/matrixmul_cs.c)の仕様
 - HW化する関数 matrixmul
 - SWで同じ計算をする関数 matrixmul_soft
 - メイン関数 main
 - 次頁のJSONファイルでHW化とそのインタフェースを指定
- (実行時間測定のためのライブラリ)
 - timer.c およびtimer.h

入カソースJSONの仕様

- software_tasks でSWの関数指定
- Hardware_tasks でHW化する関数の指定
 - “name” で関数名
 - “mode” は全体の制御で s_axilite固定
 - “arguments”内でそれぞれの関数の通信インタフェースを記述
 - “name” は引数名
 - “mode” は通信プロトコル
“s_axilite” “m_axi” “axis” どれか
 - “bundle” で同じプロトコルのポートをまとめられる(“axis”以外)
 - “m_axi”のときoffset = slave を追加
 - “direction” で入出力どちらかを指定

```
1 {
2   "software_tasks": [
3     {
4       "name": "main"
5     },
6     {
7       "name": "matrixmul_soft"
8     }
9   ],
10  "hardware_tasks": [
11    {
12      "name": "matrixmul",
13      "mode": "s_axilite",
14      "arguments": [
15        {
16          "name": "a",
17          "mode": "m_axi",
18          "bundle": "port_a",
19          "offset": "slave",
20          "direction": "in"
21        },
22        {
23          "name": "b",
24          "mode": "m_axi",
25          "bundle": "port_a",
26          "offset": "slave",
27          "direction": "in"
28        },
29        {
30          "name": "c",
31          "mode": "s_axilite",
32          "bundle": "port_b",
33          "direction": "out"
34        }
35      ]
36    }
37  ]
38 }
39
```

IF生成ツールの仕様

□ シェルで

`./ifmake.sh matrixmul /usr/lib/llvm-3.8/lib/libclang-3.8.so (Linux)`

`./ifmake.sh matrixmul /bin/cygclang-3.8.dll (Cygwin)`

を実行すれば

SWインタフェースレイヤ : `matrixmul_if.c`

HWインタフェース情報付HWタスク : `matrixmul_hw_re.c`

SWタスク : `matrixmul_sw.c`

が出力される

□ 中身では,

`python divide.py matrixmul.c matrixmul.json (SW/HW分割)`

`python ifmake.py matrixmul.c matrixmul.json (IFレイヤ生成)`

`python renamehwparams.py matrixmul_hw.c matrixmul.json`
(HWタスクのAXIプロトコルを使用する引数の通信処理の付加)

を順に実行している.

Vivado HLSによる高位合成

- プロジェクト作成・設定
 - Vivado HLSの起動画面でCreate New Projectを選択
 - Project Configuration でプロジェクト名と配置ディレクトリを選択 (今回は matrixmul_ambmcm_2015.4 とする)
 - Add/Remove files で matrixmul_hw_re.c を選択
Top Function に matrixmul を設定
 - Test bench は特になし
 - Solution Configuration で
Please Select part→board→Zedboard で使用ボードを選択
→finish

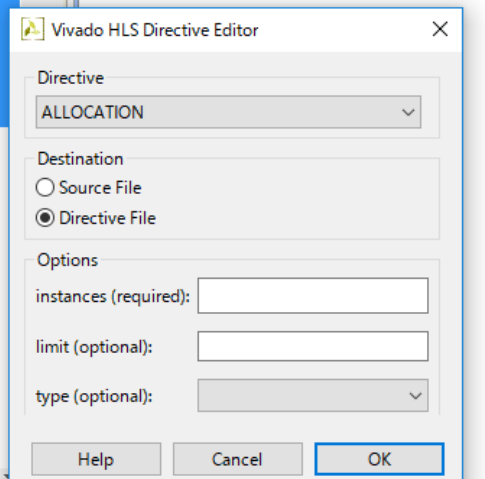
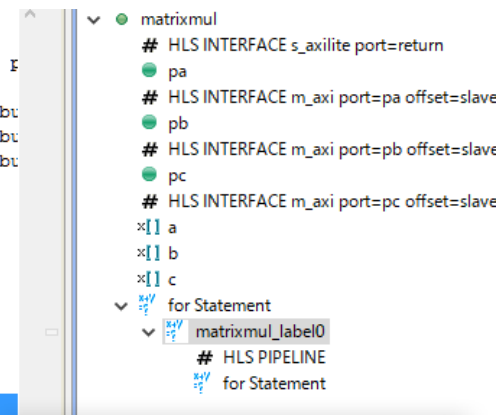
Vivado HLSによる高位合成

□ パイプライン化プラグマの追加

- ループをダブルクリックし、パイプライン化のプラグマを追加できる

(ループにはラベルが張られる)

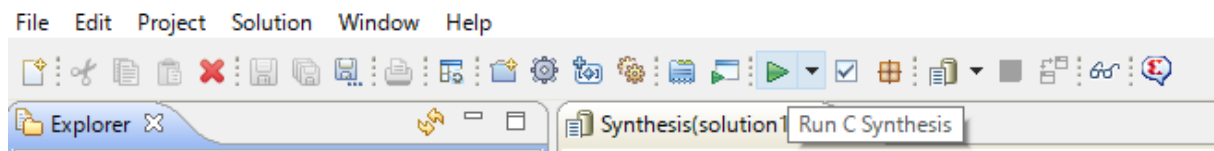
```
1 #include "string.h"
2 #define P 32
3 int matrixmul(int pa[32][32], int pb[32][32], int pc[32][32]) {
4 #pragma HLS INTERFACE s_axilite port=return
5 #pragma HLS INTERFACE m_axi port=pa offset=slave burst_depth=1
6 #pragma HLS INTERFACE m_axi port=pb offset=slave burst_depth=1
7 #pragma HLS INTERFACE m_axi port=pc offset=slave burst_depth=1
8     int a[32][32];
9     int b[32][32];
10    int c[32][32];
11    memcpy(a, pa, sizeof(int) * 1024);
12    memcpy(b, pb, sizeof(int) * 1024);
13
14    int i,j,k;
15
16    for (i = 0; i < P; i++){
17        matrixmul_label0:for (j = 0; j < P; j++){
18#pragma HLS PIPELINE
19            c[i][j] = 0;
20            for (k = 0; k < P; k++){
21                c[i][j] += a[i][k] * b[k][j];
22            }
23        }
24    }
25
26    memcpy(pc, c, sizeof(int) * 1024);
27    return 0;
28
29 }
30
```



Vivado HLSによる高位合成

□ 合成

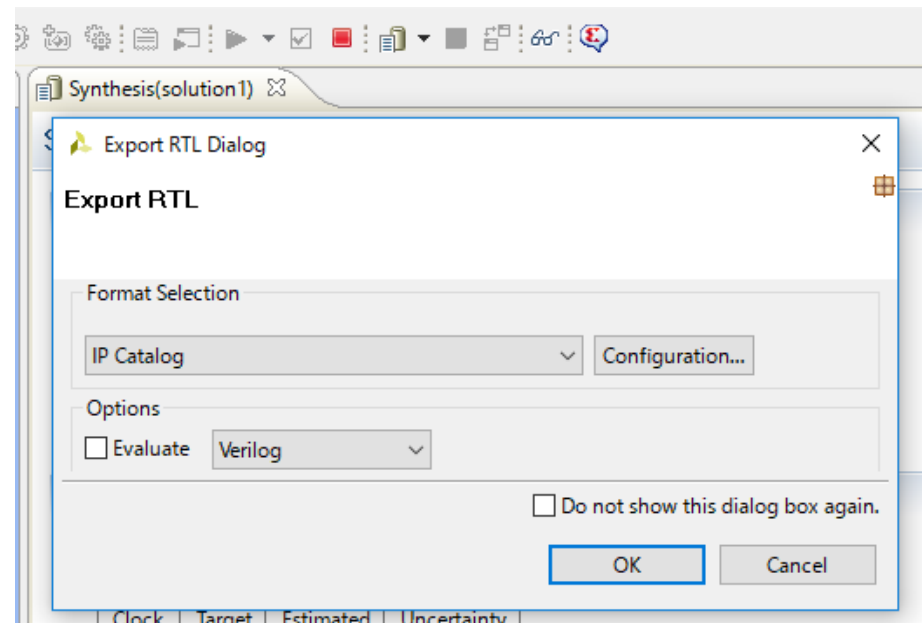
Run C Synthesis



□ IPとして出力

□ Export RTL

(Run C Synthesisの2つ隣)
からIP Catalogを選択し
Vivadoで使用できるIPとして
出力



□ Vivado HLSにおける作業は終了

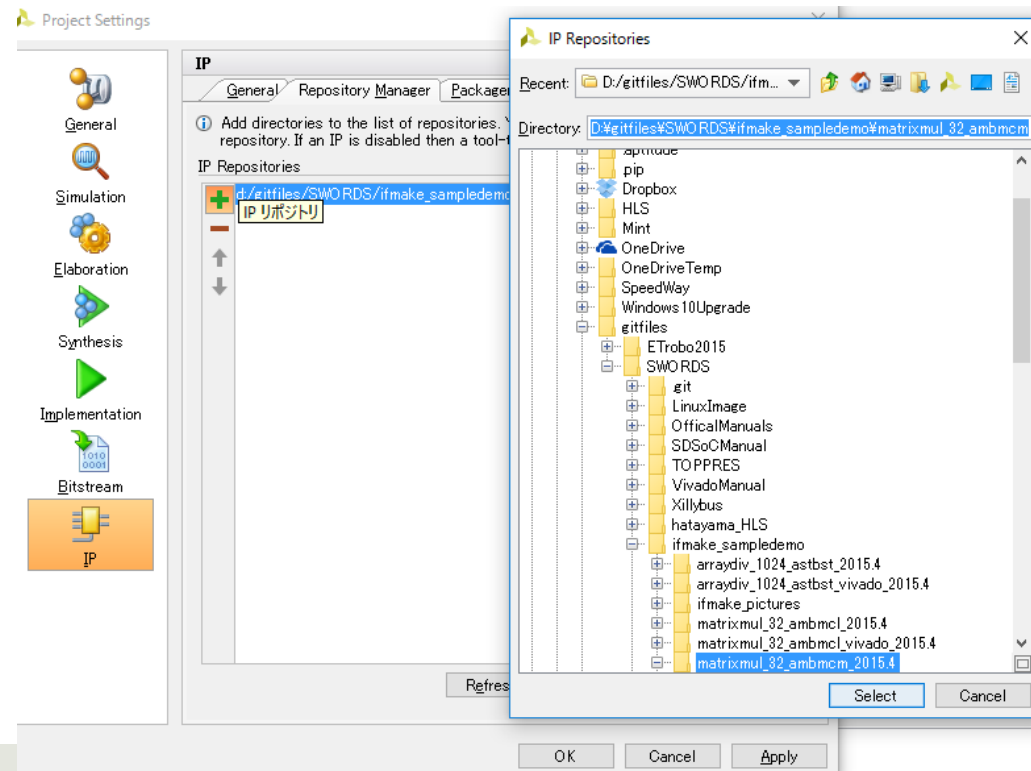
Vivadoによるデザイン

- プロジェクト作成
 - Vivado の起動画面でCreate New Projectを選択
 - Project Nameでプロジェクト名
(今回は matrixmul_ambmcm_vivado_2015.4 とする) を入力,
ディレクトリを指定
 - Project Type → RTL Project
 - Add Sources/Existing IP/Constraints → 特になし
 - Default Part で Board→ZedBoard を対象に選択
→finish

Vivadoによるデザイン

- デザインの作成
 - 左の IP Integrator → Create Block Design からデザイン名 (ここでは matrixmul_system) としてデザインを作成
- ライブラリへの高位合成IPの追加

- 空のデザインを右クリックするとメニューが現れるので IP Settings を選択
- IP → Repository Manager → + から Vivado HLSで作ったプロジェクトのディレクトリを追加, IPが認識される

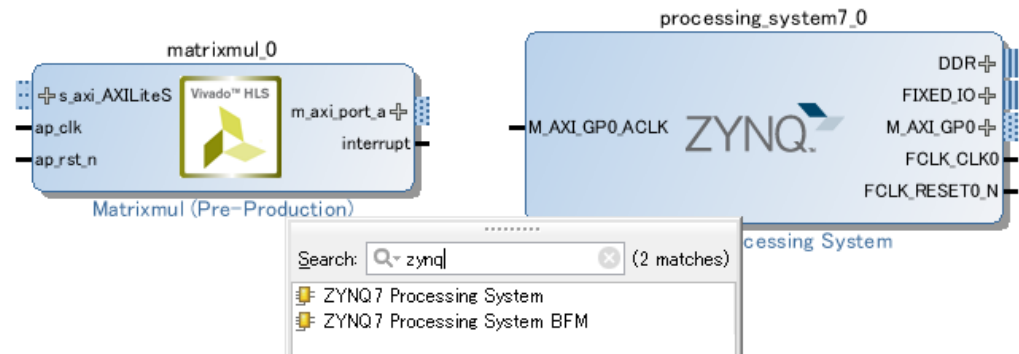


Vivadoによるデザイン

- デザインにIPを追加
 - 右クリック → Add IP... からZynq7 Processing System と Matrixmul を追加

- ARM-FPGA間通信ポートの追加

- Zynqをダブルクリックし、割込み信号とACPポートの追加を行う



Re-customize IP

ZYNQ7 Processing System (5.5)

Interrupt Port	ID	Description
<input type="checkbox"/> Fabric Interrupts		Enable PL Interrupts to PS and vice versa
<input type="checkbox"/> PL-PS Interrupt Ports		
<input checked="" type="checkbox"/> IRQ_F2P[15:0]	[91:84], [68:61]	Enables 16-bit shared interrupt port from the PL.
<input type="checkbox"/> Core0_nIRQ	28	Enables fast private interrupt signal for CPU0 from
<input type="checkbox"/> Core0_nIRQ	31	Enables private interrupt signal for CPU0 from the
<input type="checkbox"/> Core1_nIRQ	28	Enables fast private interrupt signal for CPU1 from
<input type="checkbox"/> Core1_nIRQ	31	Enables private interrupt signal for CPU1 from the

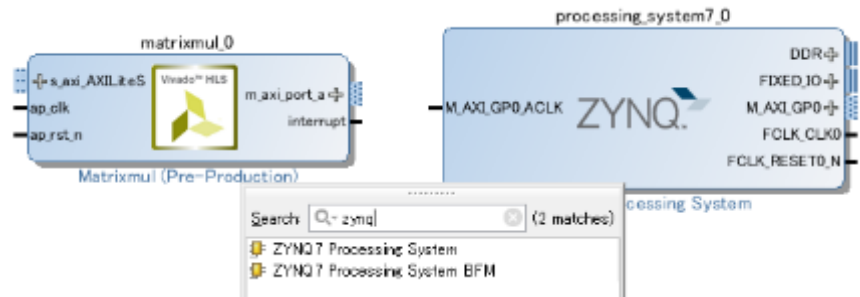
Re-customize IP

ZYNQ7 Processing System (5.5)

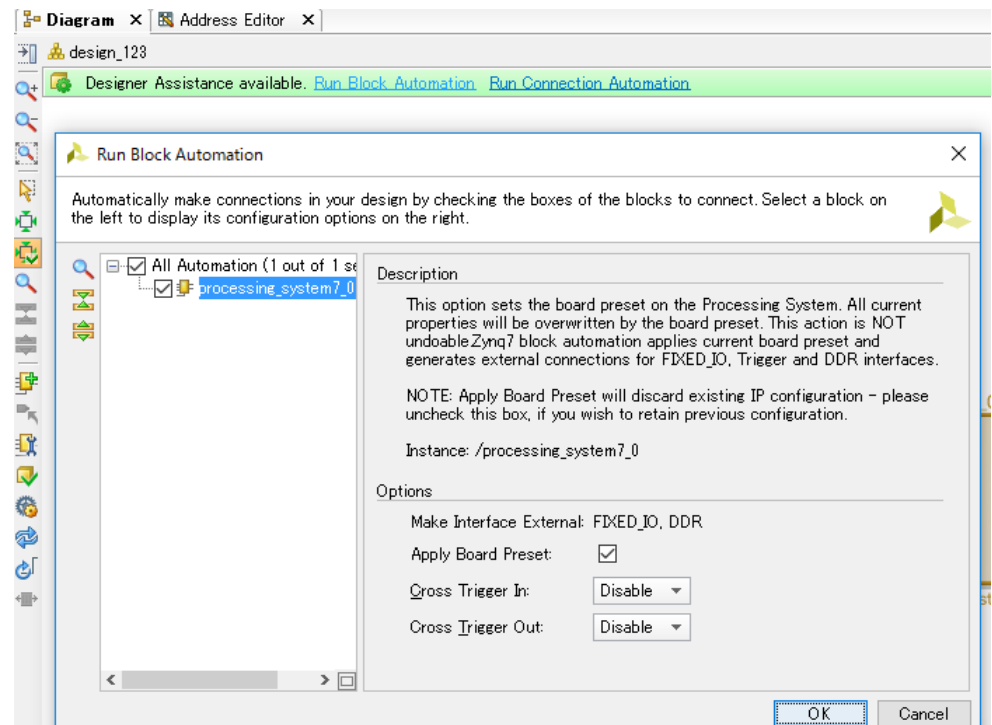
Name	Select	Description
<input type="checkbox"/> AXI Non Secure Enablement	0	Enable AXI Non Secure Transaction
<input type="checkbox"/> GP Slave AXI Interface		
<input type="checkbox"/> HP Slave AXI Interface		
<input type="checkbox"/> ACP Slave AXI Interface		
<input type="checkbox"/> S AXI ACP interface	<input checked="" type="checkbox"/>	Enables AXI coherent 64-bit slave interface
<input checked="" type="checkbox"/> Tie off AxUSER	<input checked="" type="checkbox"/>	Tie off AxUSER signals to high, enabling c
<input type="checkbox"/> DMA Controller		
<input type="checkbox"/> PS-PL Cross Trigger interface	<input type="checkbox"/>	Enables PL cross trigger signals to PS and

Vivadoによるデザイン

- デザインにIPを追加
 - 右クリック → Add IP... から Zynq7 Processing System と Matrixmul を追加

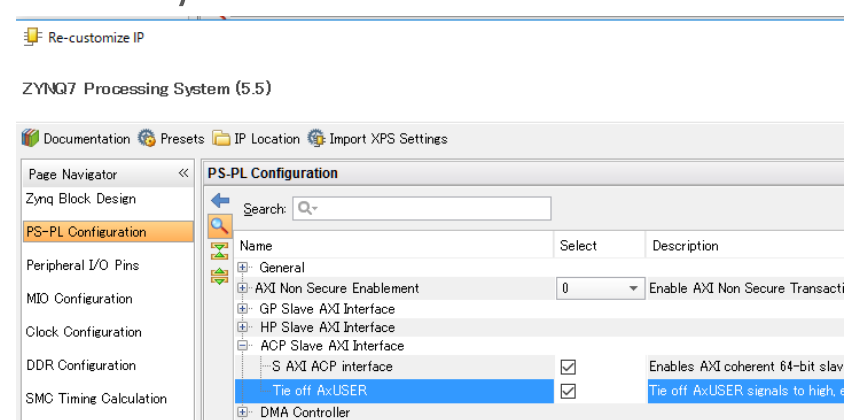
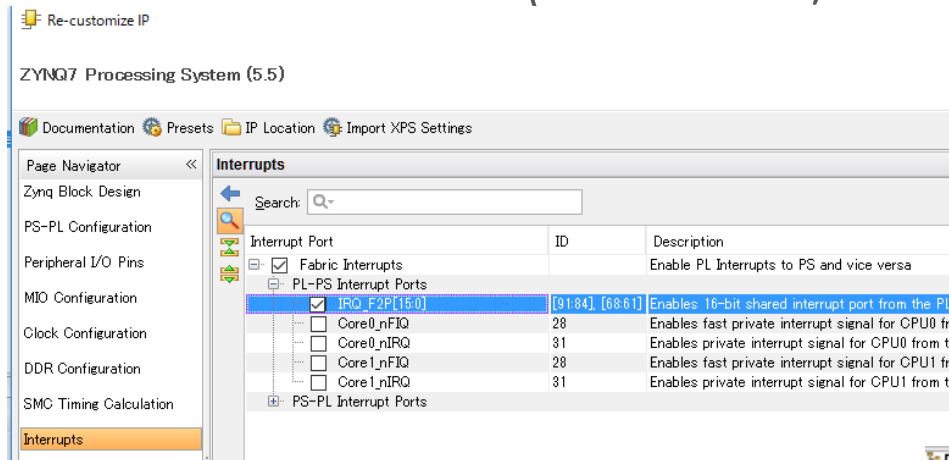


- 自動配線①
 - Run Block Automation を実行



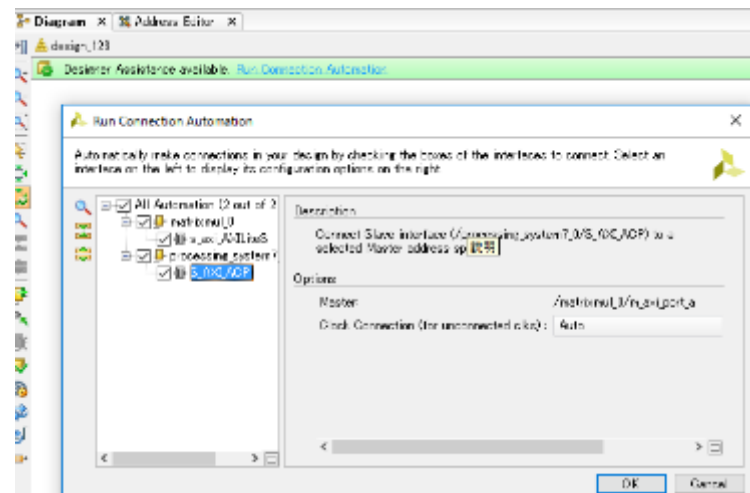
Vivadoによるデザイン

- ZynqのProcessing Systemをダブルクリックし、ARM-FPGA間通信(割り込み信号, ACPポート)を用意



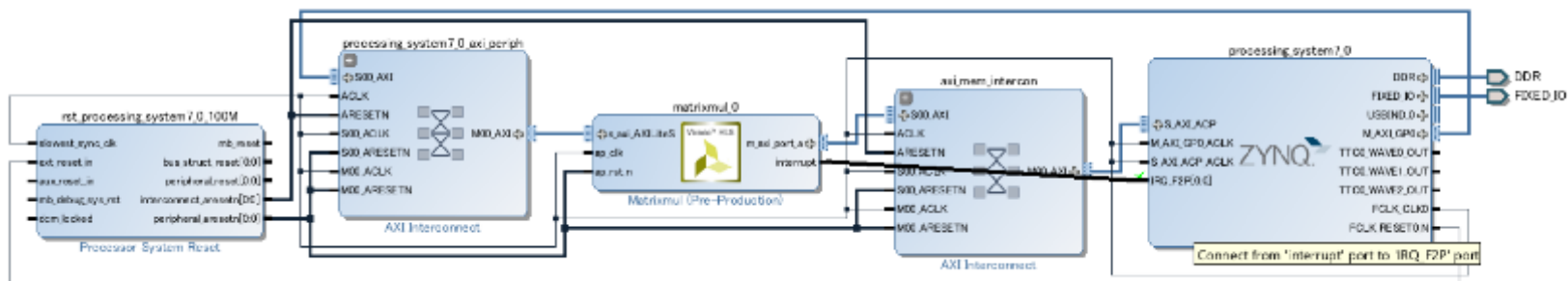
- 自動配線②

- Run Connection Automation
を実行(チェックをつける)

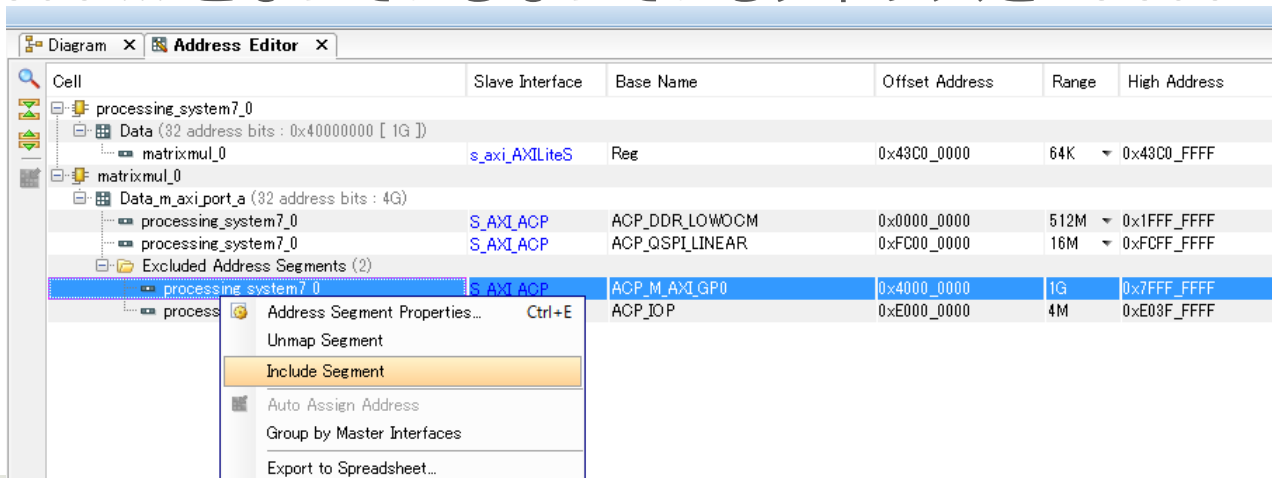


Vivadoによるデザイン

- 割込み信号を手動接続し，ダイアグラムの完成



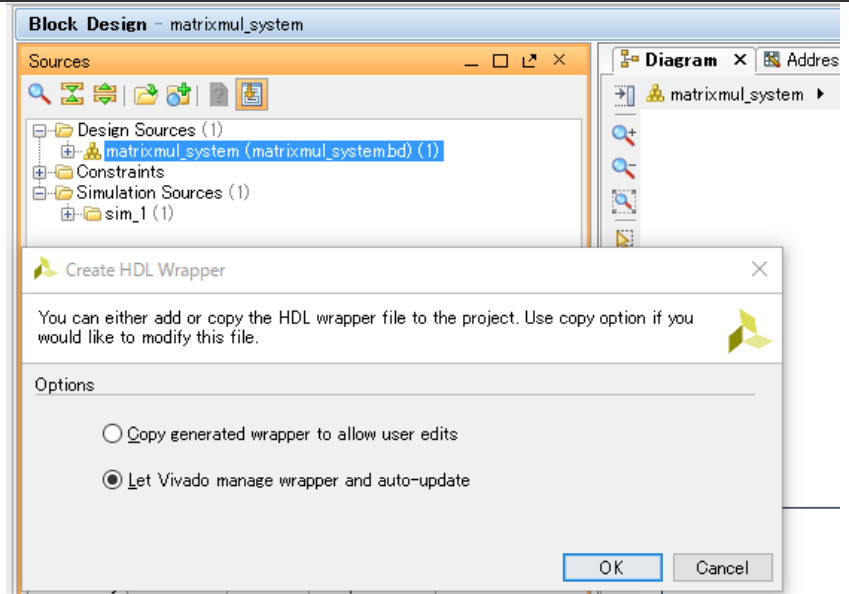
- Address Editor で Exclude ... となっているアドレスを Include



Vivadoによるデザイン

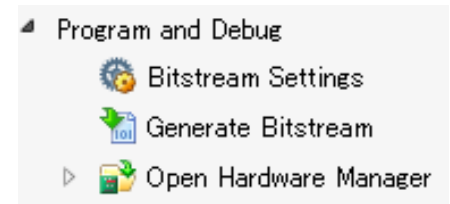
□ HDL Wrapper の作成

- Sources にある.bdファイルを選択し、右クリックして Create HDL Wrapper... を実行



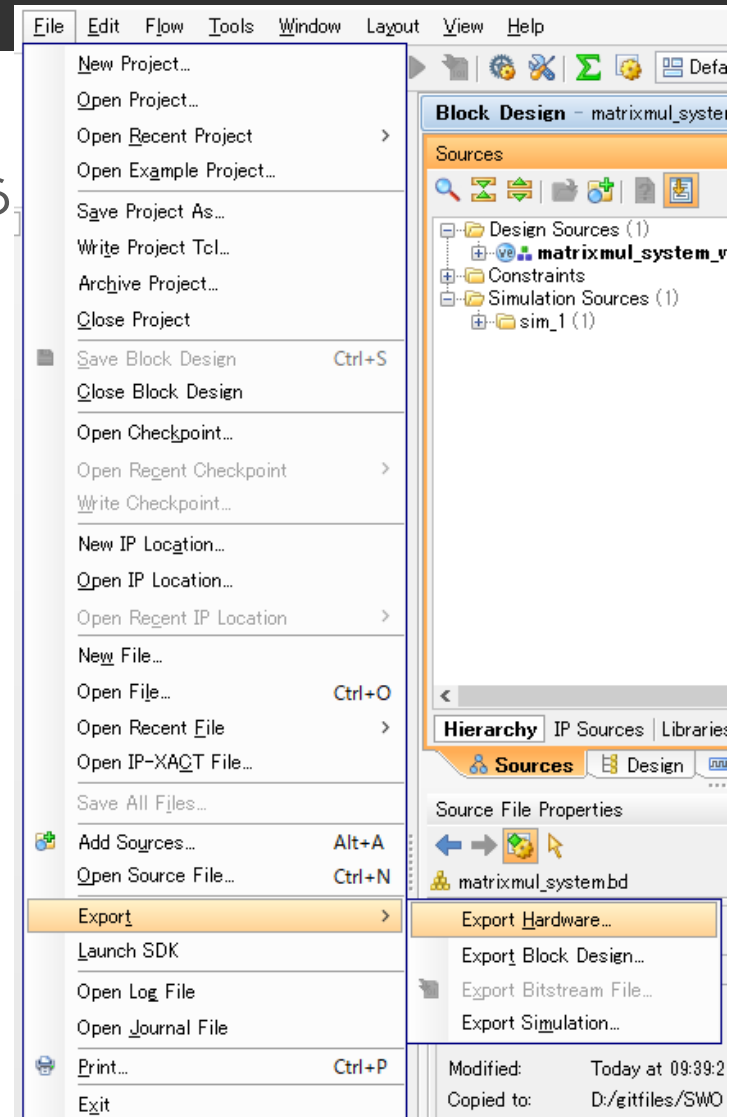
□ 論理合成・配置配線・ビットストリーム生成

- Program and Debug のGenerate Bitstream... を実行すれば途中の論理合成・配置配線も通しで実行される



Vivadoによるデザイン

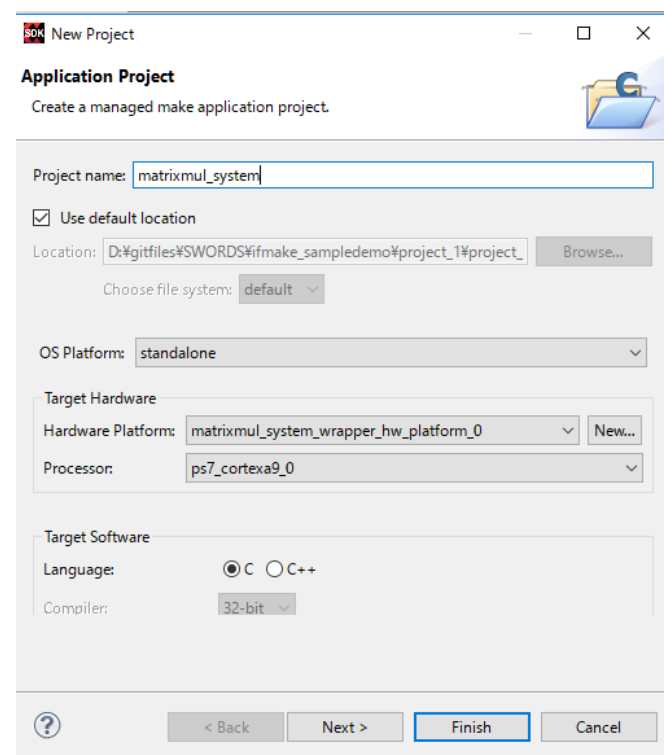
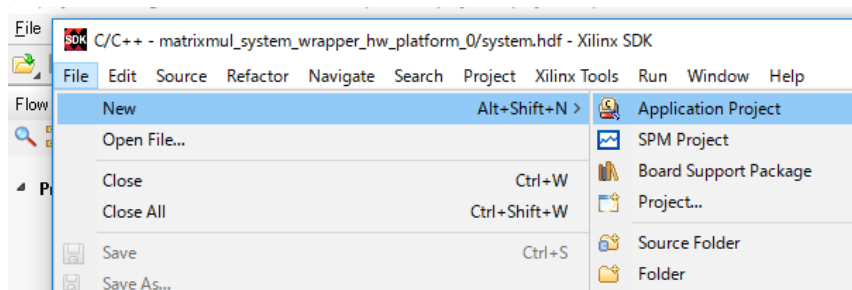
- ビットストリーム出力・SDKの呼び出し
- File→Export→Export Hardware...からビットストリームファイルを出力 (Include bitstreamにチェック)
- Launch SDK からSDKを起動
- これでVivadoでの作業は終了です



SDKでのSW作成・実行

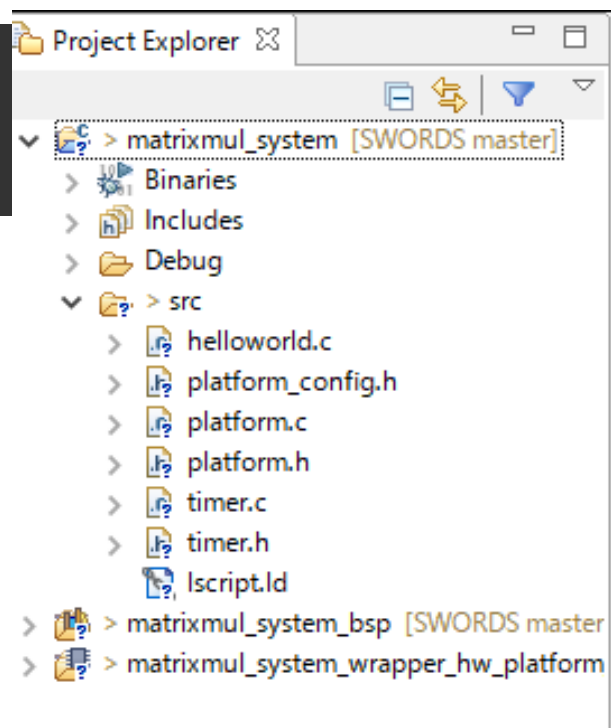
- アプリケーションプロジェクトの作成
 - File→NEW→Application Project から Project nameを matrixmul_system として作成

スタンドアロン(OSなし)とする



SDKでのSW作成・実行

- 時間計測ライブラリの追加
 - matrixmul system→srcに timer.c/timer.h をドラッグドロップして追加(copyで)
- SWの追加
 - matrixmul_sw.c の中身で helloworld.c を置換
- SWインタフェースレイヤの追加
 - matrixmul_if.c の中身を先ほどのhelloworld.c の下に追加



SDKでのSW作成・実行

- SWへの追加
 - グローバル変数a~eの宣言とtimer.hのインクルードを追加

```
XMatrixmul_Set_pa(&matrixmulx, a);
XMatrixmul_Set_pb(&matrixmulx, b);
XMatrixmul_Set_pc(&matrixmulx, c);
XMatrixmul_Start(&matrixmulx);

while(1){
    if (XMatrixmul_IsIdle(&matrixmulx) == 1){
        break;
    }
}

return XMatrixmul_Get_return(&matrixmulx);
}
//ここまでmatrixmul_if.c

//ここからmatrixmul_sw.c
#include "string.h"
#define P 32

//手動追加
#include "timer.h"
int a[P][P], b[P][P], c[P][P], d[P][P], e[P][P];
//手動追加ここまで

int matrixmul_soft(int a[32][32], int b[32][32], int c[32][32]){

    int i,j,k;

    for (i = 0; i < P; i++){
        for (j = 0; j < P; j++){
            c[i][j] = 0;
            for (k = 0; k < P; k++){
                c[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    return 0;
}

int main()
{

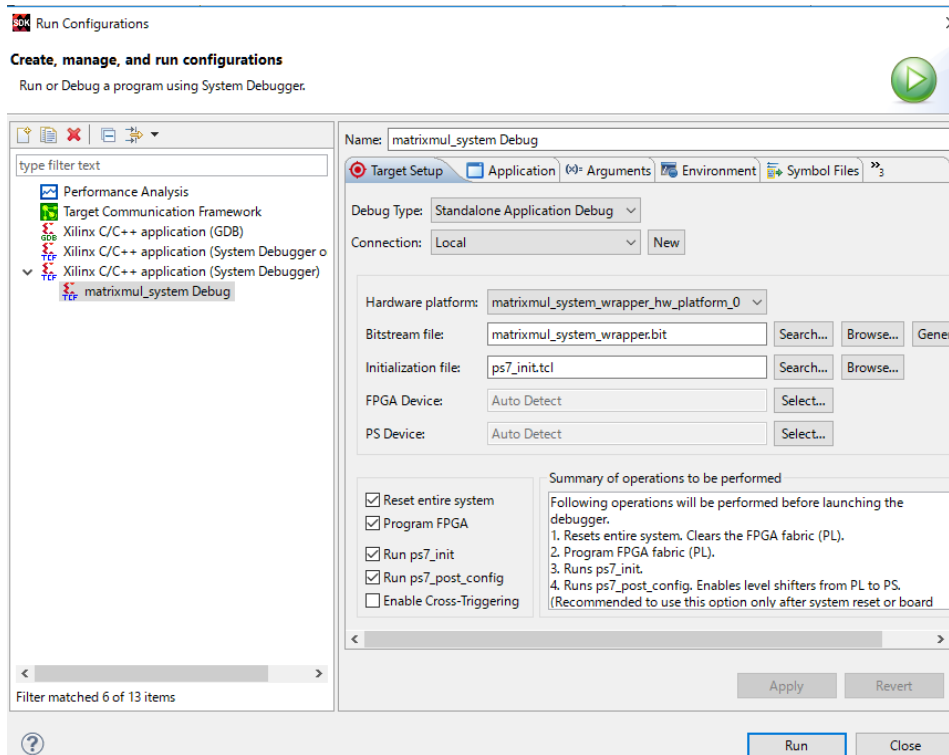
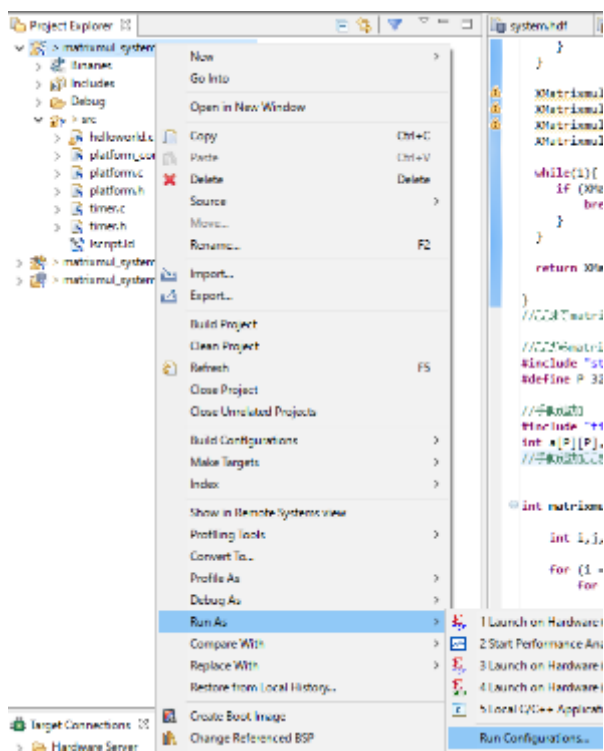
    int tmp1,tmp2,tmp3,tmp4,tmp5,tmp6;
```

SDKでのSW作成・実行

Run Configurations の作成

Xilinx C/C++ application (System Debugger)をダブルクリックするとコンフィギュレーションが作成できる

Reset entire system および Program FPGA にチェックしApply



SDKでのSW作成・実行

- PCとZedBoardの接続
 - UART と PROG と書いているZedBoardのmicroUSBポート2つをPCとmicroUSBケーブルで接続
 - Tera Term で COM1以外にポートが出現しているのでそれを選択
- 実行
 - 先ほど生成したRun Configurations で Run を押し実行
 - HW(割り込みを使用するもの) HW(割り込みを使用しないもの) SW の実行時間と結果がTera Termに出力されます

```
a[31][18] = 216 b[31][18] = 704 c[31][18] = 958368 d[31][18] = 958368 e[31][18] = 958368
a[31][19] = 124 b[31][19] = 25 c[31][19] = 468961 d[31][19] = 468961 e[31][19] = 468961
a[31][20] = 242 b[31][20] = 7 c[31][20] = 590089 d[31][20] = 590089 e[31][20] = 590089
a[31][21] = 203 b[31][21] = 203 c[31][21] = 503228 d[31][21] = 503228 e[31][21] = 503228
a[31][22] = 253 b[31][22] = 241 c[31][22] = 832072 d[31][22] = 832072 e[31][22] = 832072
a[31][23] = 66 b[31][23] = 112 c[31][23] = 518368 d[31][23] = 518368 e[31][23] = 518368
a[31][24] = 89 b[31][24] = 57 c[31][24] = 487718 d[31][24] = 487718 e[31][24] = 487718
a[31][25] = 187 b[31][25] = 75 c[31][25] = 546868 d[31][25] = 546868 e[31][25] = 546868
a[31][26] = 248 b[31][26] = 84 c[31][26] = 556642 d[31][26] = 556642 e[31][26] = 556642
a[31][27] = 243 b[31][27] = 20 c[31][27] = 513465 d[31][27] = 513465 e[31][27] = 513465
a[31][28] = 138 b[31][28] = 168 c[31][28] = 562998 d[31][28] = 562998 e[31][28] = 562998
a[31][29] = 22 b[31][29] = 168 c[31][29] = 497937 d[31][29] = 497937 e[31][29] = 497937
a[31][30] = 3 b[31][30] = 250 c[31][30] = 676264 d[31][30] = 676264 e[31][30] = 676264
a[31][31] = 251 b[31][31] = 67 c[31][31] = 507998 d[31][31] = 507998 e[31][31] = 507998
hard interr time:68045
hard polling time:85941
soft time:570945
(matrixmul!)
```


matrixmul とmatrixmul_cs の違い

- matrixmul_cs は 出力引数cをAXI4-Liteプロトコルで通信するようにしています(matrixmul_cs.jsonを参照)(元ソースは同一)
- プラグマやIFレイヤが少し異なってきます
- Vivadoでのデザインも異なってきます

