Creating an IP from Vivado HLS and Add it to Block Design

Goal

- Use Vivado HLS to build an IP core using C code
- Be able to add the new IP to your design
- Communicate to the microblaze processor through XMD

Requirements

- Xilinx Vivado software
- Xilinx Vivado HLS software
- Xilinx Nexys 4 board and a programming cable
- Enough disk space for the project files

Introduction

In this tutorial you create a simple IP for an Artix-7 FPGA using Vivado High Level Synthesis tool. For the purpose of the tutorial, the IP you create is an adder. You can modify the C code to do other operations if you wish. You can use a machine on any Operating System for this tutorial.

This tutorial targets the Xilinx Nexys 4 FPGA Evaluation Board, and uses the 2014.1 version of Vivado Design Suite. To test your system on a Nexys 4 board, you must use a terminal emulation program such as TeraTerm or Hyperterminal or the Terminal in SDK.

You must also ensure that you have the device drivers for the board installed correctly.

1. Create a Project

1. Invoke the Vivado HLS
2. From the Getting Started page, select **Create New Project**
3. In the **Project Configuration** dialog box, type the project name and location.
4. In the **Add/Remove Files** dialog box, enter **addition** as Top Function. Click Next
5. Click Next to skip testbench section.
6. Choose xc7a100tcsg324-1 in **Part Selection** and leave others as default.
7. Click **Finish** to finish creating the project.

2. Create a Source File

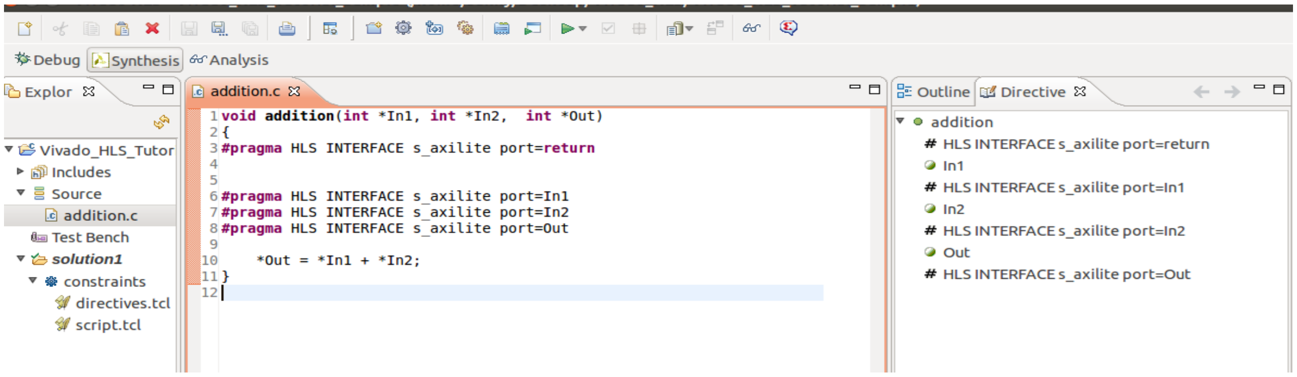
1. From Explorer > Source, right click on source and choose **New File.**

2. Specify the name (i.e. addition.c) and click OK.

3. Double-click the source file to open it in editor.

4. Copy the following code to your source.

```
void addition(int *In1, int *In2,  int *Out)
{
#pragma HLS INTERFACE s_axilite port=return
#pragma HLS INTERFACE s_axilite port=In1
#pragma HLS INTERFACE s_axilite port=In2
#pragma HLS INTERFACE s_axilite port=Out
            *Out = *In1 + *In2;
}
```

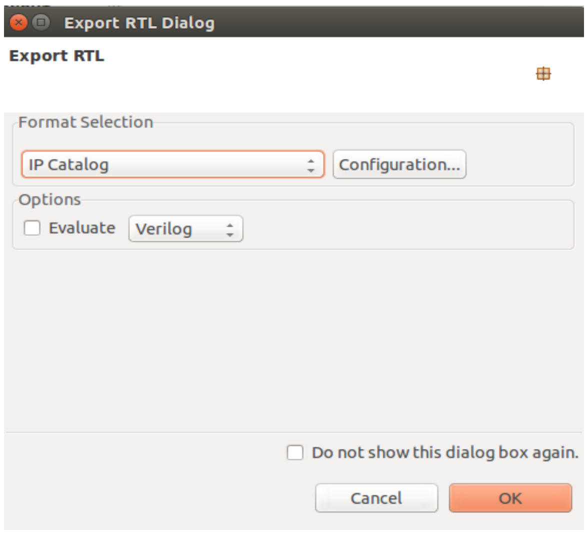


```
void addition(int *In1, int *In2,  int *Out)
{
#pragma HLS INTERFACE s_axilite port=return


#pragma HLS INTERFACE s_axilite port=In1
#pragma HLS INTERFACE s_axilite port=In2
#pragma HLS INTERFACE s_axilite port=Out

    *Out = *In1 + *In2;
}
```

Note: #progma is a pre-processor directive that helps perform some compiler specific task. In this case, it specifies the port to be axilite slave port (i.e. s_axilite).

5. Click on to ▶▾ run C synthesis and choose **Current Solution**.

6. Click ⊞ on to export RTL design.



Export RTL Dialog

Export RTL

Format Selection: IP Catalog

Configuration...

Options: ☐ Evaluate Verilog

☐ Do not show this dialog box again.

Cancel OK

7. Press OK to finish exporting the design. Now, a new IP is created in your design directory and you can start adding it in your block design.

3. Use IP in Vivado

1. Launch Vivado and create a new design.

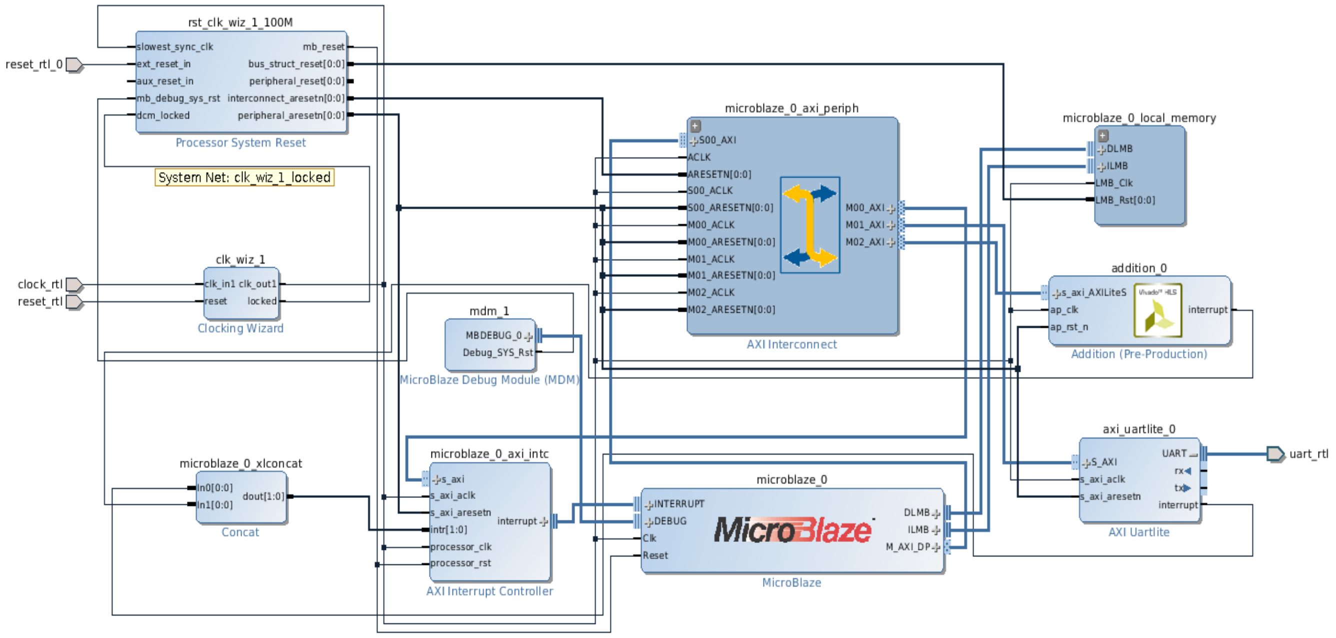
2. Click on IP Catalog in Flow Navigator and click on IP settings .

3. Click on **Add Repositories** and navigate to <your design file>/solution1/impl/ip.

Click OK. The IP will automatically be added under VIVADO HLS IP folder.

4. Create a block design and design it as follows.

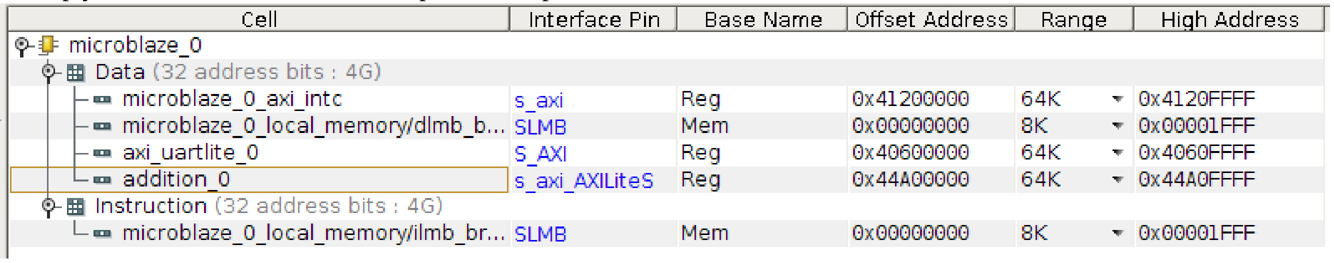
For steps on configuring microblaze and making connections, see tutorial **Configuring Microblaze**.



5. Create a constraint file and copy the following constraints. Note, make sur SW15 is high or else the processor would be in reset.

```
set_property PACKAGE_PIN E3 [get_ports clock_rtl]

set_property IOSTANDARD LVCMOS33 [get_ports clock_rtl]
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clock_rtl]

set_property PACKAGE_PIN V10 [get_ports {reset_rtl_0}]
set_property IOSTANDARD LVCMOS33 [get_ports {reset_rtl_0}]

set_property PACKAGE_PIN U11 [get_ports {reset_rtl}]
set_property IOSTANDARD LVCMOS33 [get_ports {reset_rtl}]

set_property PACKAGE_PIN D4 [get_ports uart_rtl_txd]
set_property IOSTANDARD LVCMOS33 [get_ports uart_rtl_txd]
set_property PACKAGE_PIN C4 [get_ports uart_rtl_rxd]
set_property IOSTANDARD LVCMOS33 [get_ports uart_rtl_rxd]
```

6. Map your addresses. The address map this example uses is as follows.



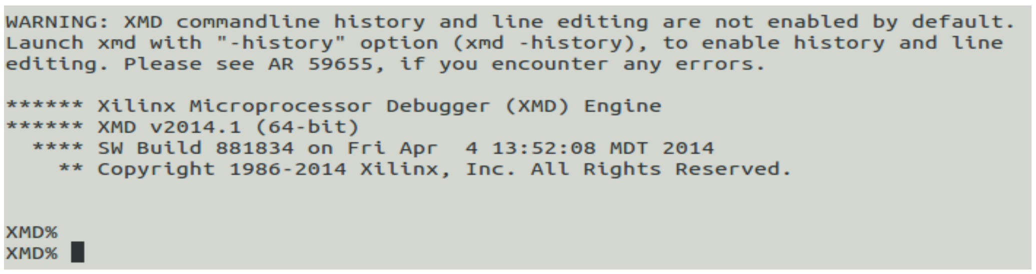
Cell	Interface Pin	Base Name	Offset Address	Range	High Address
microblaze_0					
Data (32 address bits : 4G)					
microblaze_0_axi_intc	s_axi	Reg	0x41200000	64K	0x4120FFFF
microblaze_0_local_memory/dlmb_b...	SLMB	Mem	0x00000000	8K	0x00001FFF
axi_uartlite_0	S_AXI	Reg	0x40600000	64K	0x4060FFFF
addition_0	s_axi_AXILiteS	Reg	0x44A00000	64K	0x44A0FFFF
Instruction (32 address bits : 4G)					
microblaze_0_local_memory/ilmb_br...	SLMB	Mem	0x00000000	8K	0x00001FFF

7. Click on Run Synthesis.
8. Click on Run Implementation.
9. Click on Generate Bitstream.
10. Program it on the board.

4. Test functionality using XMD

1. Open up <your design directory>/solution1/impl/ip/drives/addition_v1_0/src/xaddition_hw.h to see where the base addresses are for inputs and outputs. In this example, In1 is at base + 0x10, In2 is at base + 0x18 and Out is at base +0x20.

2. Open up terminal in Linux or command prompt in windows, type xmd and press enter. The following messages are printed in the console.

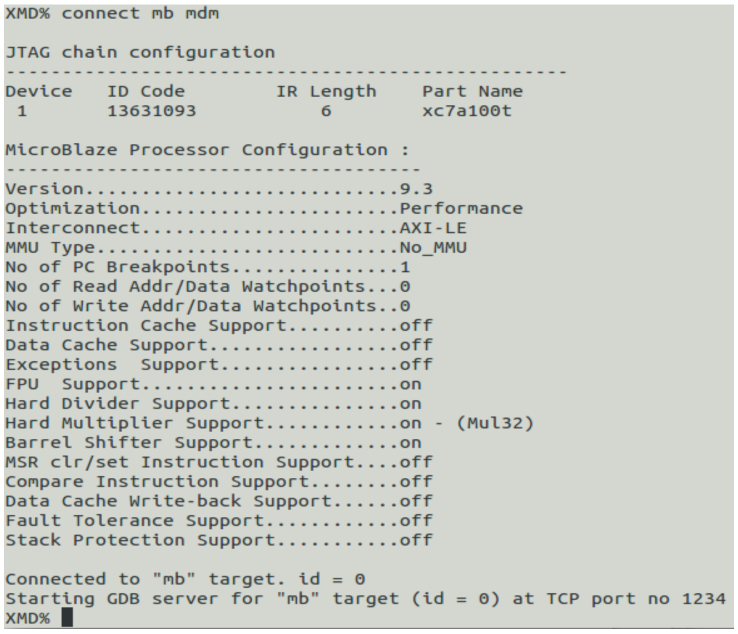


```
WARNING: XMD commandline history and line editing are not enabled by default.
Launch xmd with "-history" option (xmd -history), to enable history and line
editing. Please see AR 59655, if you encounter any errors.

****** Xilinx Microprocessor Debugger (XMD) Engine
****** XMD v2014.1 (64-bit)
  **** SW Build 881834 on Fri Apr  4 13:52:08 MDT 2014
    ** Copyright 1986-2014 Xilinx, Inc. All Rights Reserved.


XMD%
XMD%
```

3. Type connect mb mdm in the prompt and press enter. This connects the terminal to your microblaze processor.

Note: **mb** represents microblaze processor and **mdm** represents microprocessor debug module.

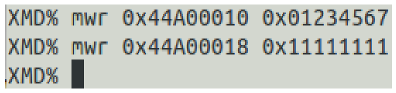
Don't forget to program the FPGA before you connect.



```
XMD% connect mb mdm

JTAG chain configuration
--------------------------------------------------
Device   ID Code        IR Length    Part Name
 1       13631093           6        xc7a100t

MicroBlaze Processor Configuration :
-------------------------------------
Version............................9.3
Optimization.......................Performance
Interconnect.......................AXI-LE
MMU Type...........................No_MMU
No of PC Breakpoints...............1
No of Read Addr/Data Watchpoints...0
No of Write Addr/Data Watchpoints..0
Instruction Cache Support..........off
Data Cache Support.................off
Exceptions  Support................off
FPU  Support.......................on
Hard Divider Support...............on
Hard Multiplier Support............on - (Mul32)
Barrel Shifter Support.............on
MSR clr/set Instruction Support....off
Compare Instruction Support........off
Data Cache Write-back Support......off
Fault Tolerance Support............off
Stack Protection Support...........off

Connected to "mb" target. id = 0
Starting GDB server for "mb" target (id = 0) at TCP port no 1234
XMD%
```

4. Try writing to 0x01234567 to In1 and 0x11111111 to In2. Type mwr 0x44A00010 0x01234567 to write to In1. Type mwr 0x44A00018 0x11111111 to write to In2.

Note: mwr represents memory write

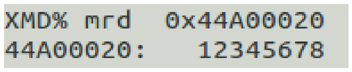


```
XMD% mwr 0x44A00010 0x01234567
XMD% mwr 0x44A00018 0x11111111
XMD%
```

5. Start the addition by writing 0x00000001 to control register. Type mwr 0x44A00000 0x00000001 to write to control register.

6. Try getting the result of the addition by reading value from 0x44A00020. Type mrd 0x44A00020 to read the result.

Note: mrd represents memory read.



```
XMD% mrd  0x44A00020
44A00020:   12345678
```

12345678 is your result from adding In1 and In2.