ECE 552 – Designing with the XUP VirtexIIPro FPGA board

Saumil Merchant <smerchant@utk.edu>
(Revised and partially tested by D. Bouldin on 3/5/07)

Platform based design:

In this motif, a user design (core) interfaces to a pre-existing platform, with the user mainly concentrating on the core design problem and not on the issues related to the platform development. The platform here is a programmable SoC that has a processor, memory, local and peripheral busses, and other I/O devices that have been pre-designed and are ready to be used. The user core interfaces with the platform via a peripheral bus to which it communicates as a slave module. The bus connects the core in a memory-mapped I/O fashion to the processor (in our case, the PowerPC or PPC405) address space.

Figure 1 shows the SoC platform we will use for our design. 'UTCore' is connected on the OPB (On-chip Peripheral Bus) as the user core. The user core has to conform to a preset interface (or port map) so that it can communicate seamlessly with rest of the platform.

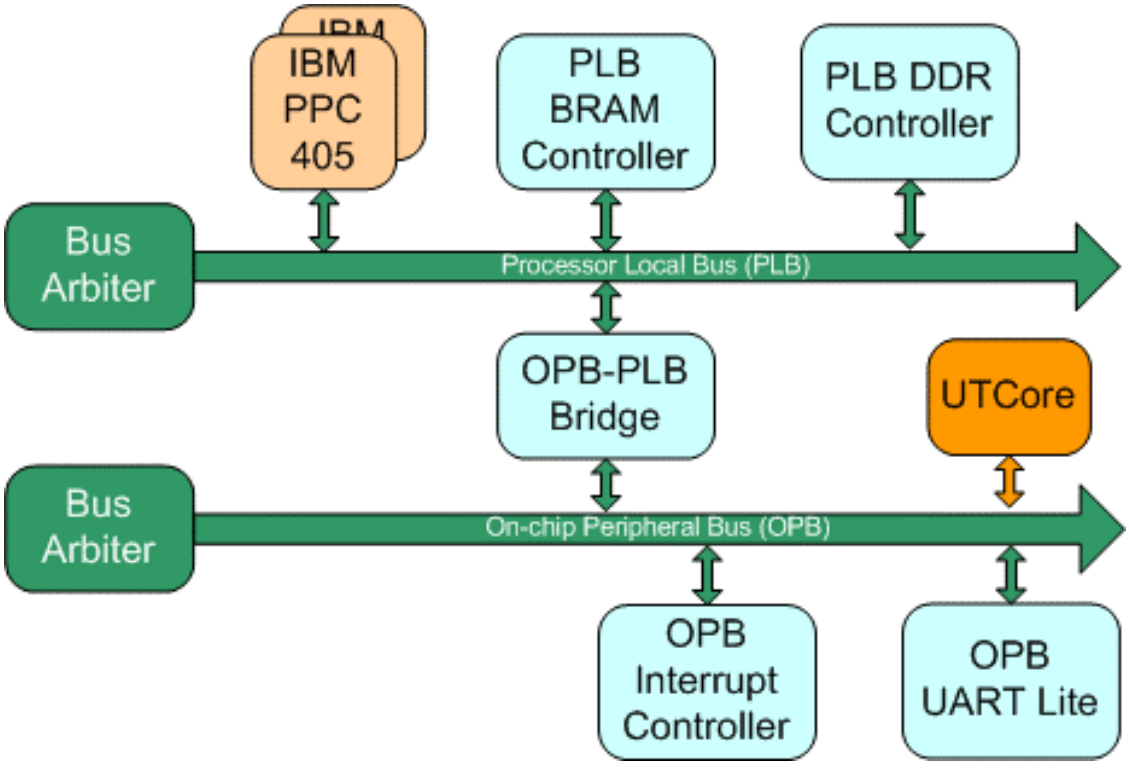

Figure 1 SoC Design Platform

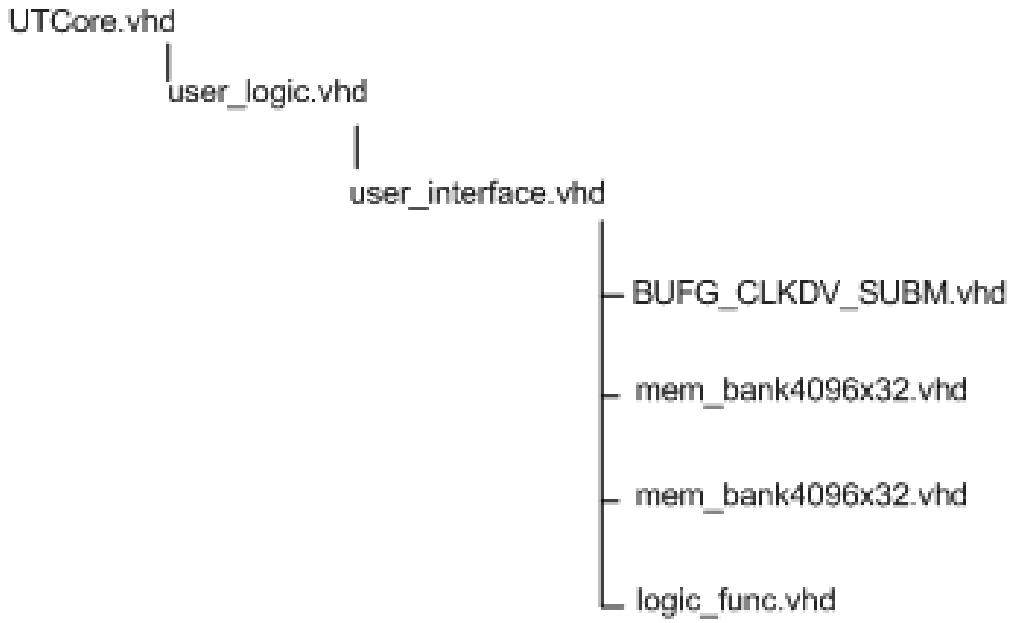

Figure 2 Design Hierarchy

Figure 2 shows the design hierarchy for *UTCore.vhd*. The primary wrapper, *user_interface.vhd,* instantiates two user memory banks, *mem_bank4096x32*. (Note that each of these 32-bit block RAMs consists of four 8-bit *dpram4096x8* modules which were generated using the Xilinx core generator tool.) Both of these 32-bit memory banks can be 'read from' and 'written to' from your logic function as well as the PPC405 processor. The data bus width is 32 bits and hence you can read/write an 8-bit byte, 16-bit word or a 32-bit word in a single read/write cycle. *user_interface.vhd* also instantiates a CLK divider that divides the input OPB CLK (100MHz) by 2. This divided CLK (50MHz) is fed to your core. Please make sure that your design can run at 50 MHz or else you will need to generate the appropriate internal CLK division for your logic. In case of using internal CLK division, make sure that the interface between the memory bank and your logic still runs at 50 MHz. *user_interface.vhd* also instantiates your core *logic_func.vhd*. In most cases, it is our hope that you will not need to edit *user_interface.vhd* and your entire core design will be under *logic_func.vhd.*

The entity descriptions of *user_interface.vhd* as well as *logic_func.vhd* are:

```
entity user_interface is
      port (
            -- memory mapped input registers
            reg1 : in std_logic_vector(31 downto 0);
            reg2 : in std_logic_vector(31 downto 0);
            reg3 : in std_logic_vector(31 downto 0);
            reg4 : in std_logic_vector(31 downto 0);
            reg5 : in std_logic_vector(31 downto 0);
            reg6 : in std_logic_vector(31 downto 0);
            reg7 : in std_logic_vector(31 downto 0);
            reg8 : in std_logic_vector(31 downto 0);
            reg9 : in std_logic_vector(31 downto 0);
            reg10 : in std_logic_vector(31 downto 0);
            reg11 : in std_logic_vector(31 downto 0);
            reg12 : in std_logic_vector(31 downto 0);
            reg13 : in std_logic_vector(31 downto 0);
            reg14 : in std_logic_vector(31 downto 0);
            reg15 : in std_logic_vector(31 downto 0);

            -- memory mapped output registers
            reg16 : out std_logic_vector(31 downto 0);
            reg17 : out std_logic_vector(31 downto 0);
            reg18 : out std_logic_vector(31 downto 0);
            reg19 : out std_logic_vector(31 downto 0);
            reg20 : out std_logic_vector(31 downto 0);
            reg21 : out std_logic_vector(31 downto 0);
            reg22 : out std_logic_vector(31 downto 0);
            reg23 : out std_logic_vector(31 downto 0);
            reg24 : out std_logic_vector(31 downto 0);
            reg25 : out std_logic_vector(31 downto 0);
            reg26 : out std_logic_vector(31 downto 0);
            reg27 : out std_logic_vector(31 downto 0);
            reg28 : out std_logic_vector(31 downto 0);
            reg29 : out std_logic_vector(31 downto 0);

            -- Note: reg0 is mapped to start pulse
            -- reg30 is used for counter output
            -- reg31 is mapped to oprdy signal

            reg0 : in std_logic_vector(31 downto 0);
            reg30 : out std_logic_vector(31 downto 0);
            reg31 : out std_logic_vector(31 downto 0);

            Clk   : in  std_logic;
            Reset : in  std_logic;
            Addr1 : in  std_logic_vector(31 downto 0);
            Din1  : in  std_logic_vector(31 downto 0);
            BE1   : in  std_logic_vector(3 downto 0);
            RNW1  : in  std_logic;
```

```
            CS1   : in  std_logic;
            Dout1 : out std_logic_vector(31 downto 0);
            Ack1  : out std_logic;
            Addr2 : in  std_logic_vector(31 downto 0);
            Din2  : in  std_logic_vector(31 downto 0);
            BE2   : in  std_logic_vector(3 downto 0);
            RNW2  : in  std_logic;
            CS2   : in  std_logic;
            Dout2 : out std_logic_vector(31 downto 0);
            Ack2  : out std_logic);
end entity user_interface;
```

```
entity logic_func is
      port (
            din1 : in std_logic_vector(31 downto 0);
            dout1 : out std_logic_vector(31 downto 0);
            addr1 : out std_logic_vector(13 downto 0);
            wen1 : out std_logic;
            en1   : out std_logic;
            din2 : in std_logic_vector(31 downto 0);
            dout2 : out std_logic_vector(31 downto 0);
            addr2: out std_logic_vector(13 downto 0);
            wen2 : out std_logic;
            en2   : out std_logic;
            start : in std_logic;
            oprdy : out std_logic;
            reset : in std_logic;
            clk : in std_logic );
end logic_func;
```

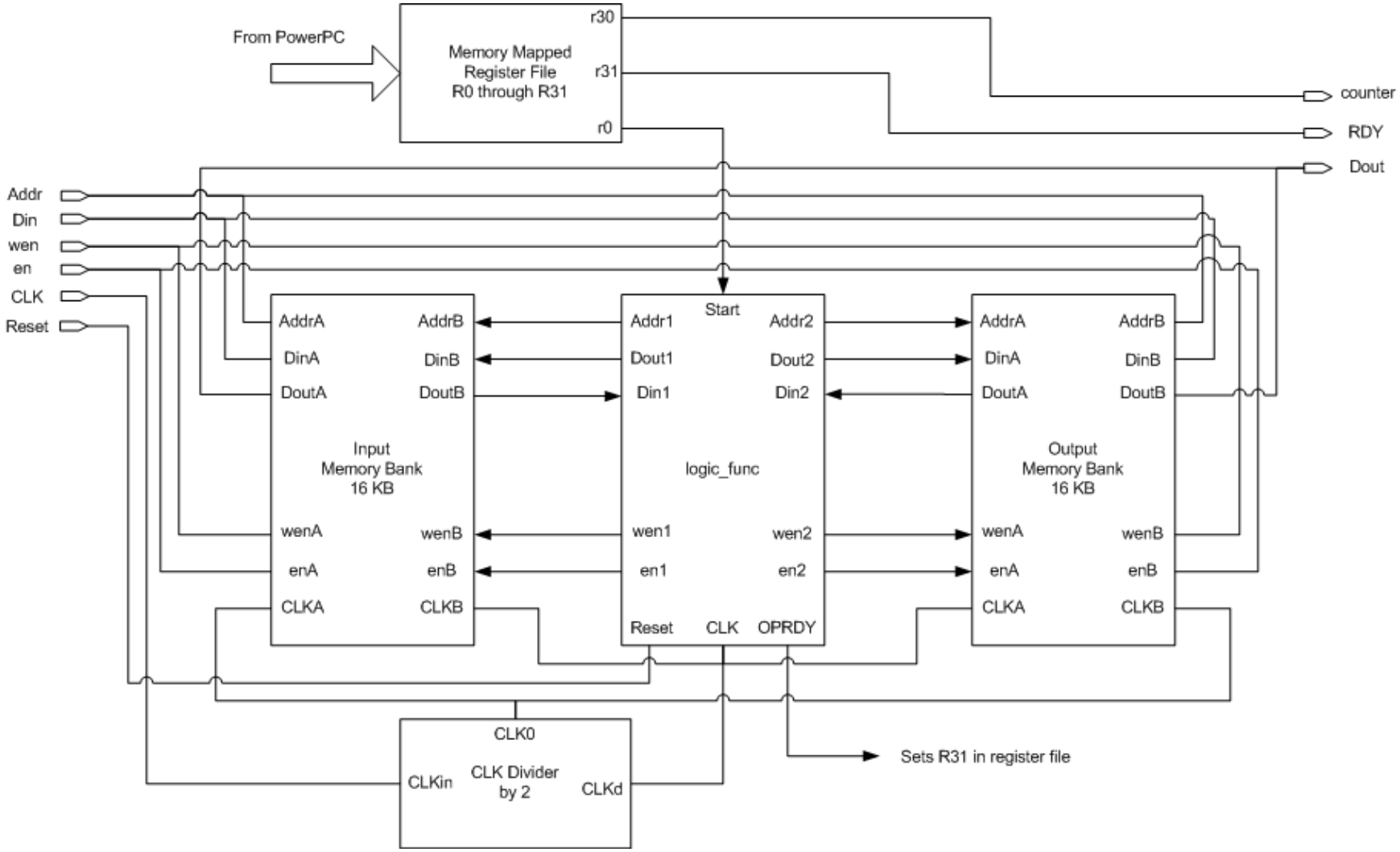

Figure 3 Logic Diagram for *user_interface.vhd*

Figure 3 illustrates the logic diagram for *user_interface.vhd* which also has a memory-mapped register file with 32 general-purpose registers that can be used for input and output from the PowerPC. These registers are unidirectional with respect to the PowerPC. Registers1 through 15 can be used for inputs to the user core from the PowerPC. Registers 16 through 29 can be used as outputs from the user core to the PowerPC. Register0 has been reserved for the start signal, register31 for the output ready signal and register30 for the CLK tick counter output. This is a free running counter that resets to zero at every start pulse and its value is latched in register30 on the OPRDY pulse. The counter is clocked by the divided clock from the CLK divider, thus incrementing at 50MHz.

The software that we will use for this lab is

1. Xilinx Embedded Development Kit (EDK) for generating and downloading the bit stream to FPGA.
2. Synplify Pro from Synplicity for synthesizing user cores.

We will use a C program that runs on the PowerPC to send test vectors to our logic core and read back its output. The output will be seen on the serial terminal window. This C program will be compiled and encoded along with the FPGA bit stream in the internal PowerPC instruction memory. The C code will also be used to transfer a 128 x 128 x 8 raw image from the host machine to the input memory bank of *user_interface.vhd* via the UART (serial) interface. We will use an internal PowerPC counter (Time Base) to measure the time it takes to load the image via the UART. The reported value will be in number of time base ticks. The time base is a free running, incrementing counter which is clocked at 300 MHz. Note, this counter is different from the counter embedded in *user_interface.vhd* and is clocked via a different clock.

This lab has been divided in two parts.

Part A

In this part you will synthesize and download a FPGA bit stream on the XUP. You will fill up the entire input BRAM with dummy data (0x5A5A5A5A). Upon receiving a start pulse from the PowerPC, *logic_func.vhd* reads this data from the input BRAM, inverts it and writes it to the output BRAM. When it finishes its job, it will assert the OPRDY pulse. Upon receiving the OPRDY pulse, the C code in the PPC will read back the computed data from the output BRAM.

Follow these steps to synthesize the hardware files and download it to the FPGA.

1. *mkdir 552-hw2 ; cp /usr/cad/course/xup.tar.gz 552-hw2*
2. *cd 552-hw2; gunzip xup_tutorial.tar.gz ; tar - xvf xup_tutorial*
3. *cd xup_tutorial*
4. Open the *user_core/user_interface.vhd, user_core/logic_func.vhd* and *C_testcode/parta/TestApp_Memory.c* to understand what they do.
5. To synthesize your design, type: *cd user_core; synth_synplicity*

   This will produce *rev_1/user_interface.edf* and *rev_1/user_interface.srr* [results]

6. cp *rev_1/user_interface.edf ../platform/pcores/UTCore_v1_00_a/netlist*

7. cp *dpram4096x8.edn ../platform/pcores/UTCore_v1_00_a /netlist*

8. Open the following file *"xup_tut/platform/pcores/UTCore_v1_00_a/data/UTCore_v2_1_0.bbd"* file and make sure that all the EDIFs (*user_interface.edf* and *dpram4096x8.edn)* in your design are listed in this file. If any of the EDIFs are not listed, please list them.

Steps to build *download.bit*:

9. Change directory to the *xup_tutorial/platform/* directory.
10. Execute: *source /usr/cad/.cshrc ; xilinx_latest_tools*
11. *cp ../C _testcode/parta/TestApp_Memory.c TestApp_Memory/src*
    cp ppc405_0/libsrc/UTCore_v1_00_a/src/UTCore.h TestApp_Memory/src
12. Clean up any previous synthesis files: *make –f system.make hwclean*
13. To synthesize the netlist, execute [50 mins]: *make –f system.make netlist*
14. To PAR and generate a bit file, execute [77 mins]:
    make -f system.make bits
15. To build the software libraries, execute: *make -f system.make libs*
16. To build the program files, execute: *make -f system.make program*
    This compiles the *TestApp_Memory.c*.
17. To initialize the BRAMs with the generated program file, execute:
    make -f system.make init_bram

Steps to configure the FPGA with the configuration file:

18. Copy *xup_tutorial/platform/implementation/download.bit* to *xup_tutorial/download/*
19. Copy *xup_tutorial/platform/etc/download.cmd* to *xup_tutorial/download/*
20. Transfer the folder "download" and "image" (partB only) and its contents to a Windows PC in the 425 Ferris Hall lab or your own PC.

Using the XUP board

The XUP board manufactured by Digilent, Inc., has the following features.

- Virtex-2 Pro XC2VP30 FPGA with 30,816 Logic Cells, 136 18-bit multipliers, 2,448Kb of block RAM, and two PowerPC Processors
- DDR SDRAM DIMM that can accept up to 2Gbytes of RAM
- 10/100 Ethernet port
- USB2 port
- Compact Flash card slot
- XSGA Video port
- Audio Codec
- SATA, and PS/2, RS-232 ports
- High and Low Speed expansion connectors with a large collection of available expansion boards

The reference manual for the board can be found at <http://www.digilentinc.com/>

Figure 4 shows the XUP board.

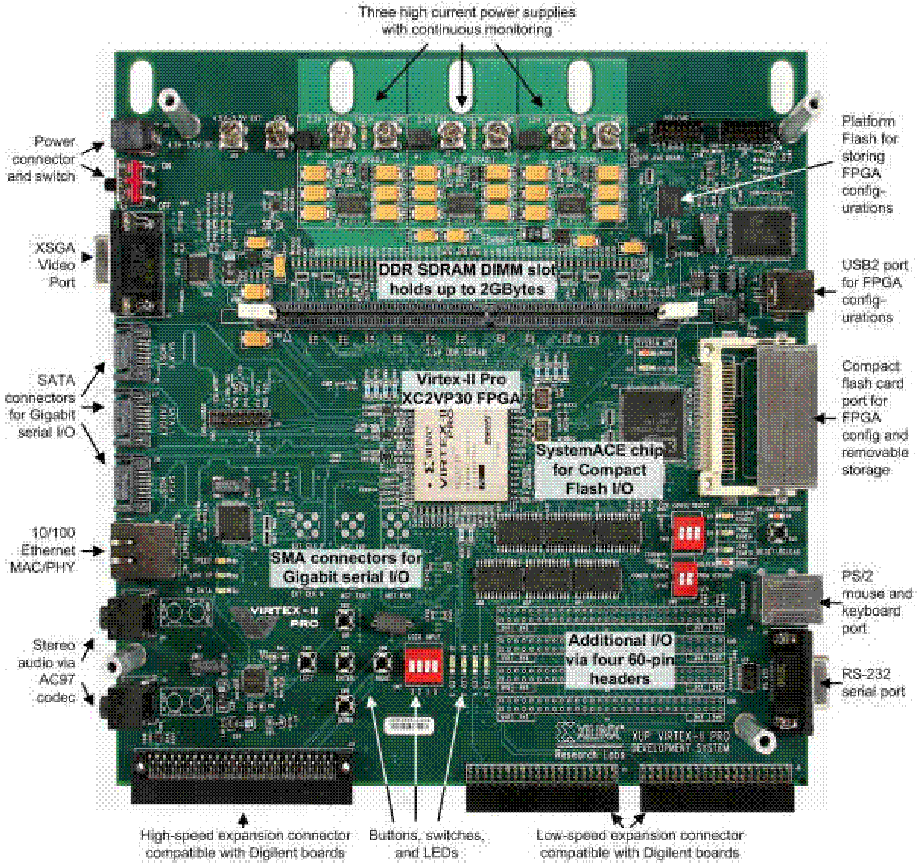

Figure 4. XUP board from Digilent Inc.

The on-board FPGA can be programmed using configuration flash memory (or a compact flash card). We will use the configuration flash along with USB JTAG cable to configure the FPGA. You should have received one USB cable, one serial cable (male DB9 connectors on both ends) and power supply along with the board. The following steps show how to connect the board.

21. Make sure that the ON/OFF switch is in the OFF position.
22. Connect one end of the serial cable to the XUP COM port and the other end to your host machine.
23. Connect one end of the USB cable to the XUP board and other to the host machine.
24. Connect the power supply to the XUP board.
25. Make sure that all the toggle switches in SW8 are in the ON position and in SW9 are in the OFF position before attempting to download a bit stream to the FPGA.
26. Once you turn the power ON, you will notice a red LED flashing next to the DIMM slot. You may ignore this since it is not an error. All it indicates is that it didn't find a compact flash card in the CF slot.
27. Open: Start > Programs > Xilinx Platform Studio > Xilinx cygwin shell
28. Change directory to the download folder. For example: *cd /cygdrive/c/download*
29. Open TeraTermPro and set up a serial connection with following parameters:
    Baud Rate – 115200
    Data – 8 bits
    Parity – None
    Stop Bit – 1
    Flow Control – None
    Serial Port – COM port to which the serial cable from XUP is connected.
30. Turn the XUP board power ON.
31. In the cygwin shell window, execute: *impact –batch download.cmd*

Part B:

In this part we will write pixel data from an image to the input BRAM instead of dummy data. The *logic_func.vhd* will as before perform some simple logic function (invert in our case) and write the computed data to output BRAM. The C code than reads back the computed data.

Change to the *xup_tutorial* directory and copy the new C code:

cp C_testcode/partb/TestApp_Memory.c platform

Change to the platform directory and repeat the parta steps beginning with Step 16. This will compile the new software and initialize the BRAMs. There is no need to re-synthesize the hardware since there is no change in the hardware logic files.

Steps to generate raw image data file:

The PNM (portable anymap – PGM or PPM) image format is used due to its simplicity. Basically, PNM images have a three-line-long header and a raster scan of the pixels. For more information, check '*man pnm*'. This flow assumes that the input image uses 8 bits for its grayscale and is 128 x 128 pixels. For larger grayscale images or color images (red, green, and blue channels) that can be partitioned on the host and reassembled after processing by the XUP board.

1. Navigate to the partb image directory: cd *{path}/C_testcode/partb/image*
2. Convert the input image into grayscale portable pixmap format by typing: *anytopnm lena.jpg > lena.pgm* (or use XV or IrfanView (for PC) to save the image as pgm).
   (for color:
   anytopnm lena_c.jpg > lena_c.ppm
   ppmtorgb3 lena_c.ppm
   makes lena_c.red lena_c.grn lena_c.blu)

3. View grayscale image.
   xv lena.pgm &

4. Separate header and data:
   head -3 lena.pgm > header
   tail -1 lena.pgm > datafile.bin

   The file *datafile.bin* now holds the raw grayscale image data from lena.jpg.

Now use the Steps to configure the FPGA with the configuration file to download the bit stream and test the hardware. When prompted (in the TeraTermPro window) for an input file, send *datafile.bin* generated above as input file. (File > send file > *datafile.bin*).

Steps to view the generated output image
(To be done on PC)

5. Copy and paste the output dump of the image hex values as seen on the TeraTermPro screen to a binary file on your host. Use notepad to do this. Save the file as '*image/dataout.bin*'.
6. Run *IMparser.exe* by double clicking on it. Enter the header filename, data output (*dataout.bin*) filename and an output image name (*result.pgm*)

7. View result image. *result.pgm*

8. If desired, convert image to any another format.

Appendix

To view the layout, type:

xilinx_latest_tools; fpga_editor platform/implementation/system.ncd