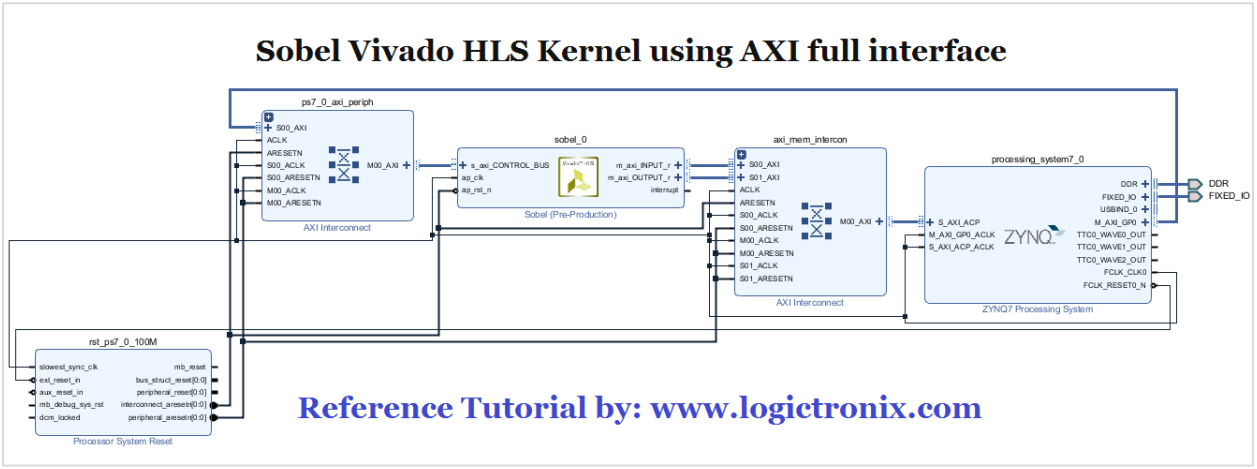
Sobel Vivado HLS Kernel using AXI full interface

Reference Tutorial of Fpgaworld



See the Original Article [with HLS, SDK Code Explanation] on the Footer Section of this tutorial!

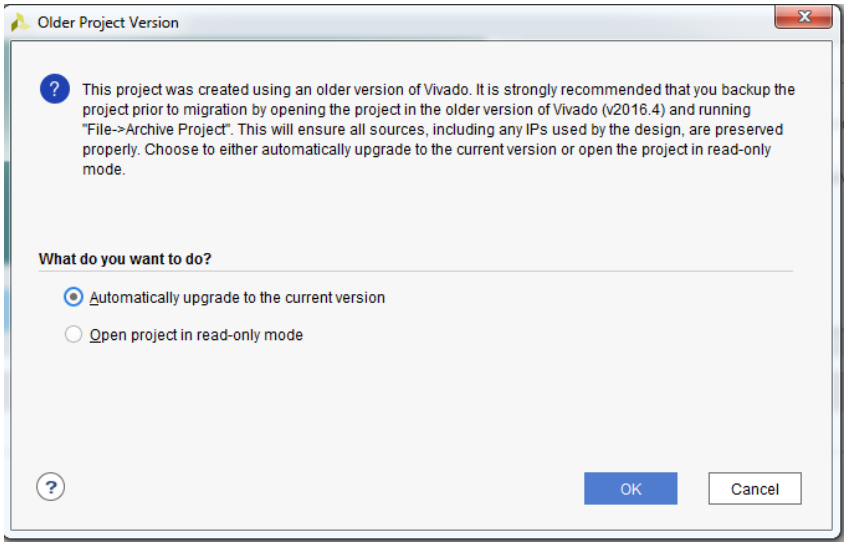
Here are the Tutorial Steps:

[Note: You will need Linux system for generating the binary image file to give on SD to ZedBoard and again getting the RGB image from Binary Output from ZedBoard, while having Windows and Linux both will be most favorable]

I. Steps followed on Windows 7:

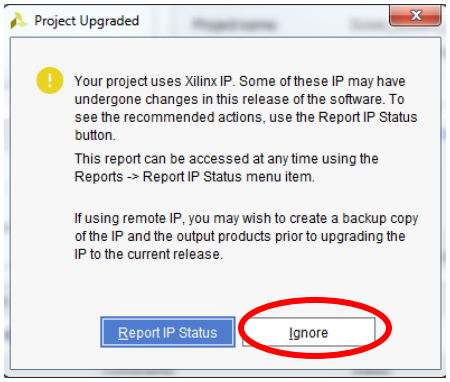
1. **Download the VIVADO Project File from this link: VIVADO Project**
2. **Extract the Project [Extract on any Local Disk, as on: E:/], you will see this type of files inside it [taken in windows 7]:**

3. **Now open the VIVADO project inside the Sobel_System/Sobel_system_ddr/**
4. The provided source is of VIVADO 2016.4, if you are using later version [newer version] of VIVADO then it will tell you about [older version than VIVADO 2016.4 does not open's it] :



Click on "Automatically Upgrade to the Current Version" and click on OK.

5. After VIVADO Opens the Project,

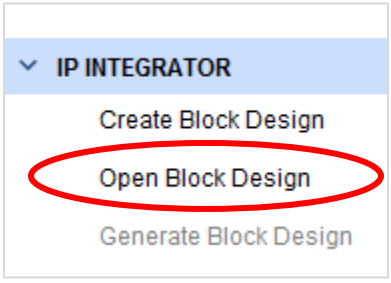


Just click on "Ignore"!

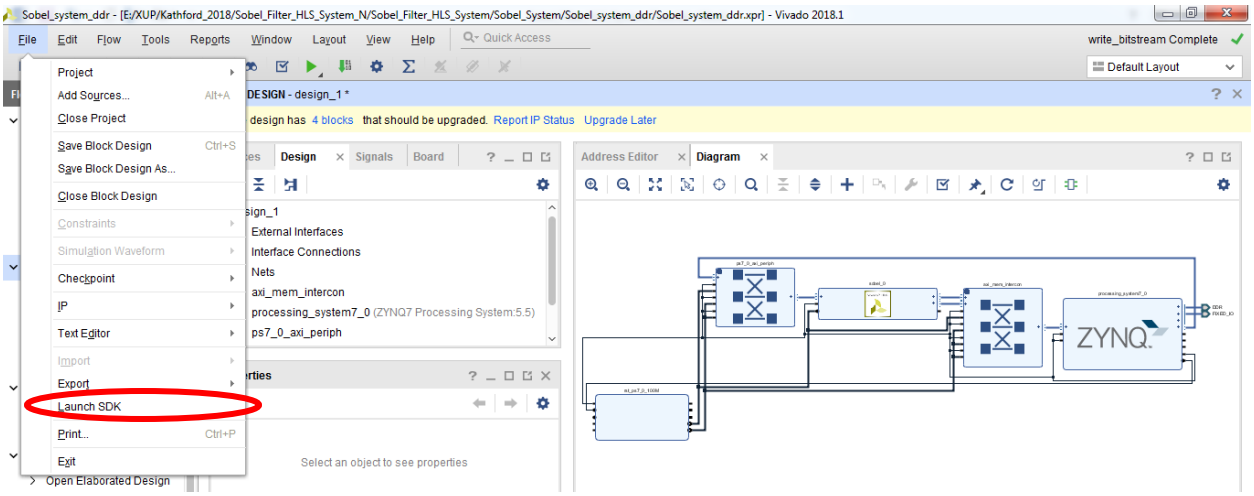
This option appears because of some of IP used on the block Design are Upgraded on the later version of VIVADO, so VIVADO recommends about, do you like to upgrade your IP to latest one or not?

6. This Project is already been generated [it is complete project including HLS Design, VIVADO IP integrator Design and VIVADO SDK Design, all the explanation is provided on the original website link on the top].

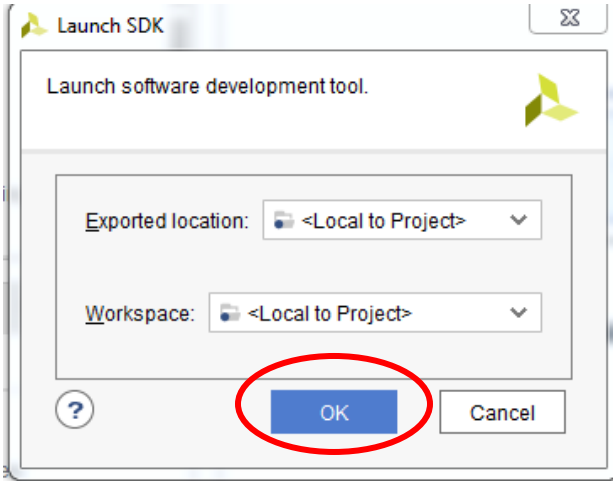
   If you want to see the Block Design of this project then do click on, Open Block Design,



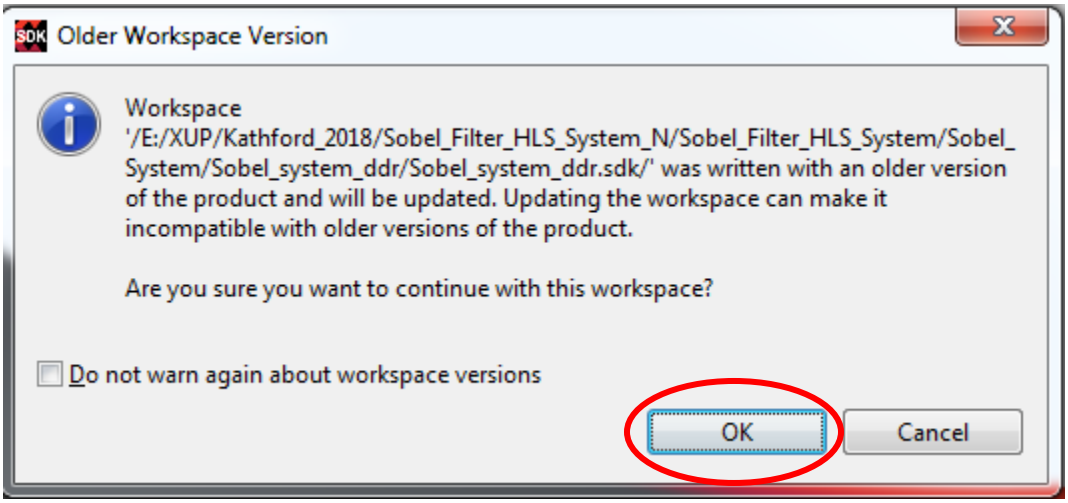
7. **Now launch the SDK from File> Launch SDK:**



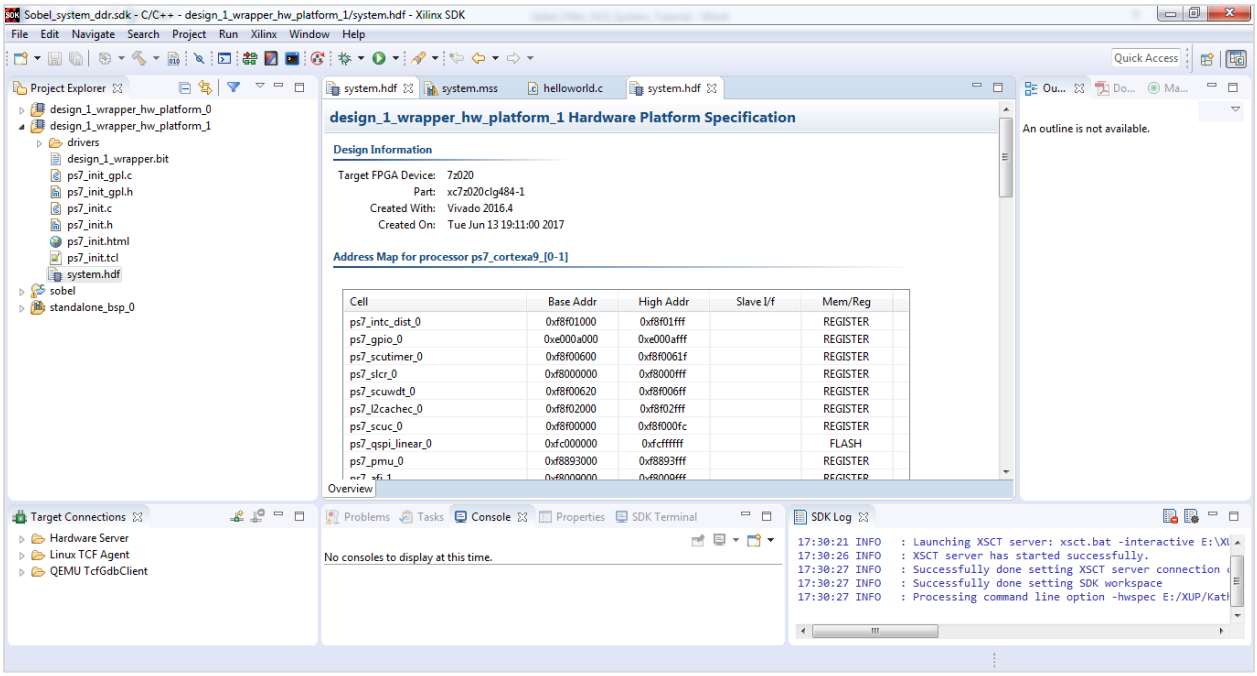
8. On this option, Click on OK:



9. Now again click OK, if this option appears:

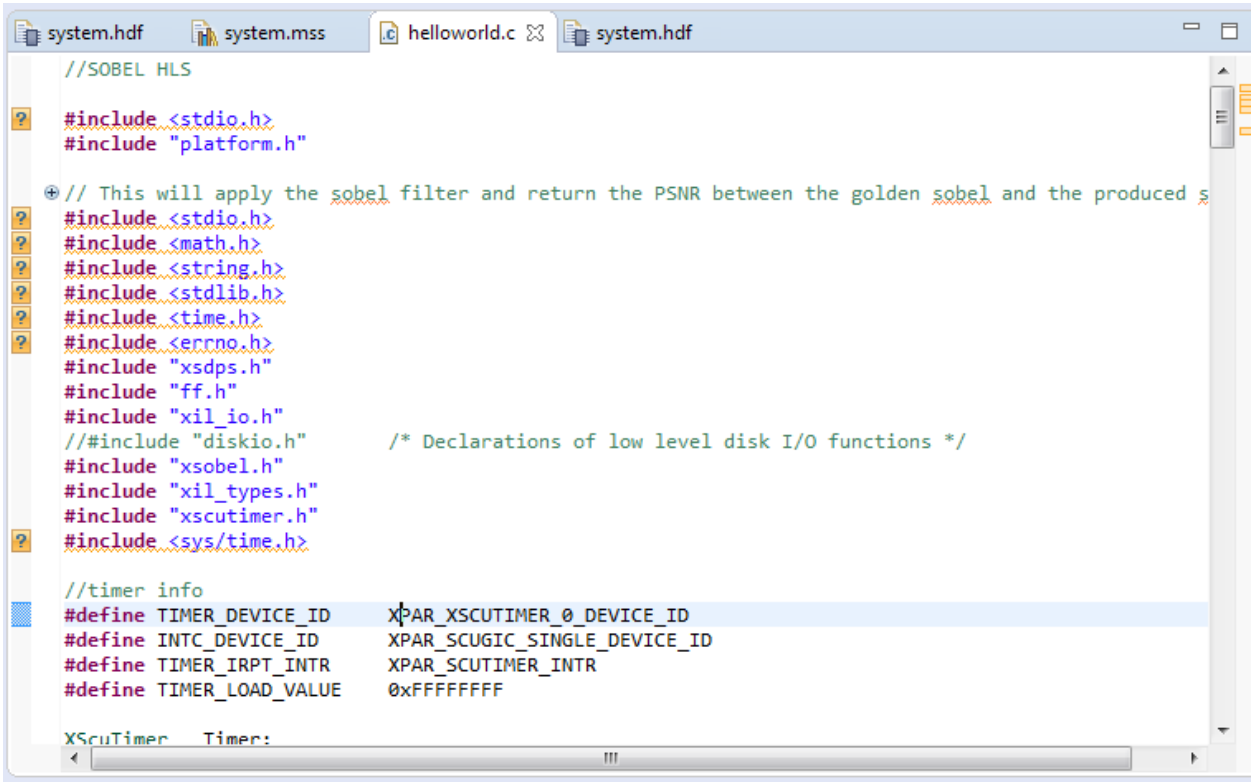


10. Now SDK Opens up automatically, it must be like this:

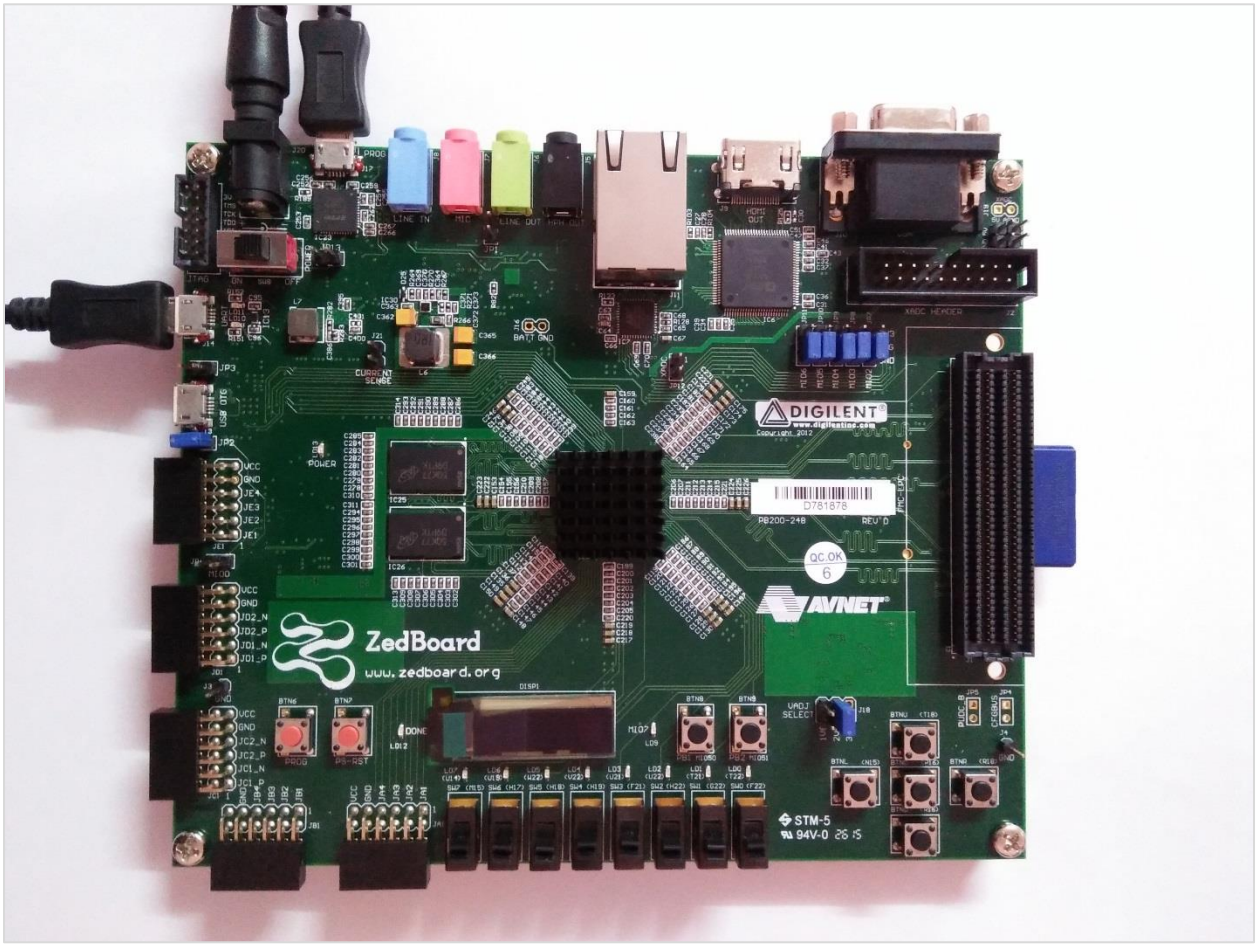


Note: If you are getting any error on SDK Log then you can close this SDK project and Launch it again from VIVADO. Sometimes there may have some .java error if the SDK won't configure well with the current workspace. You can also write us your difficulties if you stuck here at: info@logictronix.com .

11. [Optional] You can see the "helloworld.c" file which is the main file for configuring the Zynq PS which is on our Block Design:



12. **Now connect ZedBoard with USB Cable to your PC for programming through the SDK [Here is the Jumper setting and show up for Cable Connection]:**



II. Ubuntu Section for Image Conversion [BMP/JPEG into Binary and Vice Versa]

13. Now you need to open up the Linux PC [You need Linux PC for small Image conversion task, while there might also some options with windows for converting the BMP/JPEG image into the Binary image of .GRE.
    We are using Ubuntu 16.04 LTS for converting this BMP/JPEG image into Binary.
14. Now open the terminal of the Linux distro [Ubuntu on our case]: **Ctrl+Alt+T** also opens the Terminal.
    Login your system on Root, enter "su" and enter the password of your Linux System:

```
su
```

Install the "imagemagick" on your Linux system:

```
apt-get install imagemagick
```

15. **Now download any JPEG image, convert it into 300x168 pixel size using online or offline image resizing tool and place it on your desktop, place name "new.jpeg".**
    This is our imput image [you can have different image of JPEG of 300x168 dimension]:

16. Direct your terminal to desktop:

```
cd Desktop
```

17. **Now, We will convert "new.jpeg" into "input.gre" using this command:**

```
convert –depth 8 –size 1024x1024 new.jpeg GRAY:input.gre
```

    This command will convert the "new.jpeg" into the "input.gre" of 1024x1024 size and the "8-BPP monochrome".

18. **Now, We will again convert "new.jpeg" into "golden.gre" using this command:**

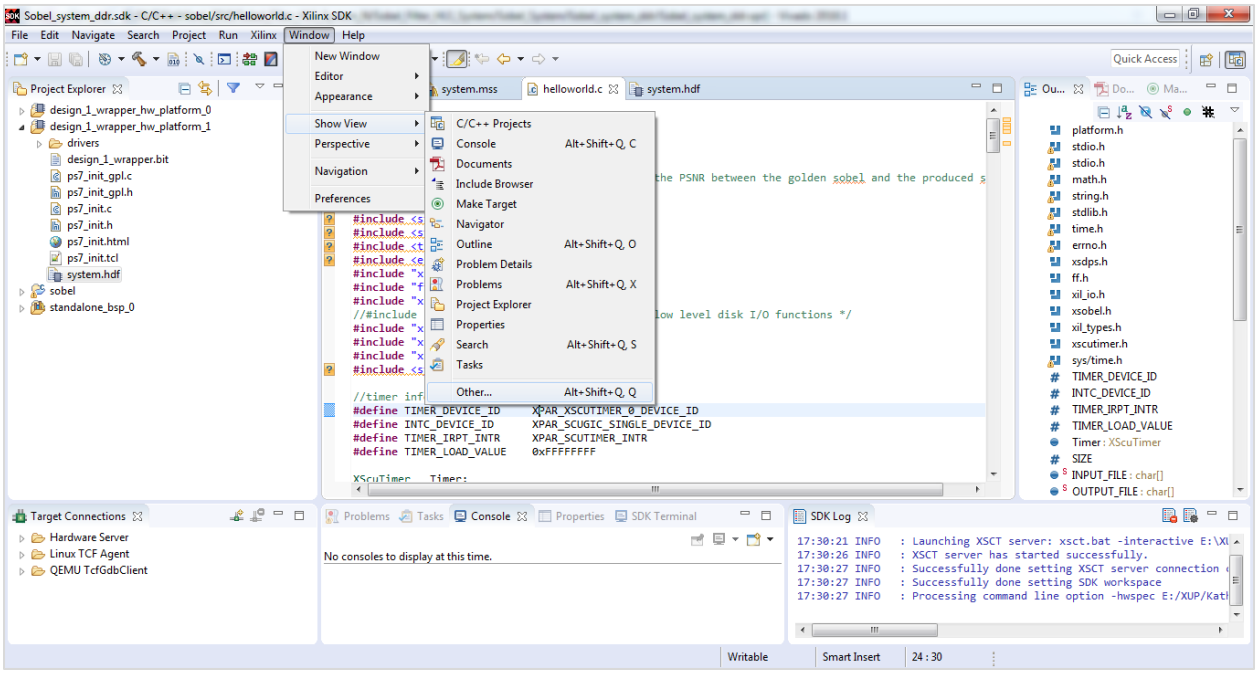
```
convert –depth 8 –size 1024x1024 new.jpeg GRAY:golden.gre
```

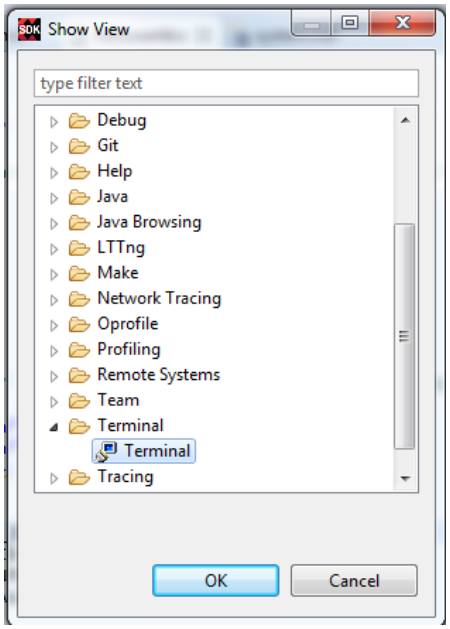
    This command will convert the "new.jpeg" into the "golden.gre" of 1024x1024 size and the "8-BPP monochrome".

19. **Now take out the SD card from the ZedBoard and Copy this two "input.gre" and "golden.gre" into the SD card. [You can copy this two .gre file from any of Linux or Windows]**

III. Now again back to Windows 7 system [where VIVADO SDK Opens].

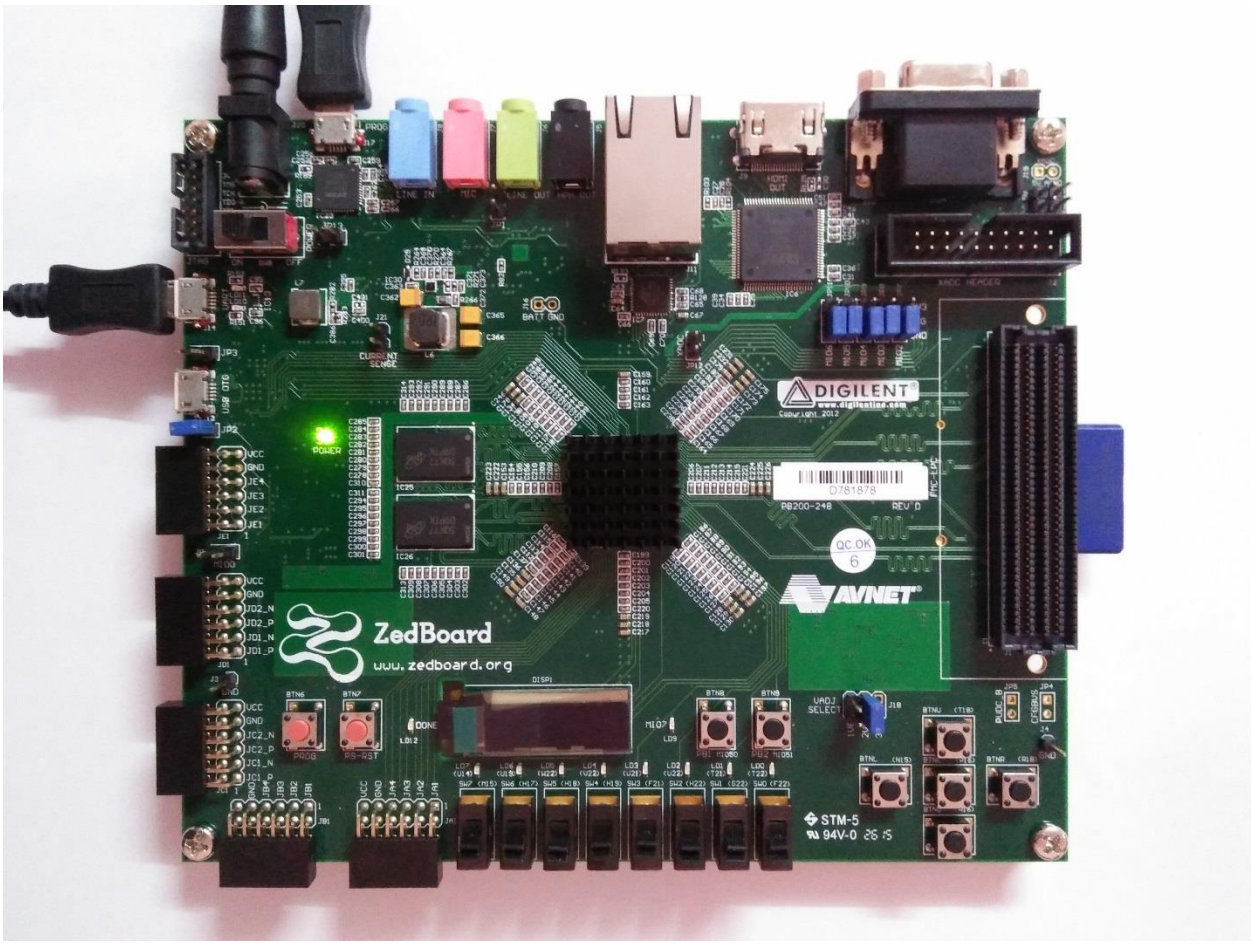
20. Now Goto VIVADO SDK, and Windows>show view>other>terminal



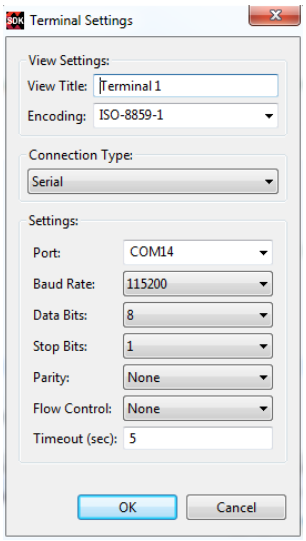


Now click on Terminal

21. Plug in the SD card on the ZedBoard and setup jumper as follows and turn the Power jumper:



22. Setup the Serial Connection with the ZedBoard



If SDK is not showing the COM port then do close SDK and launch again from the VIVADO. This COM port need an extra driver on your Winodws/Linux System, If you are using SDK terminal first time then you might need to install the Driver for the UART for ZedBoard. Follow "Cypress USB-to-UART Setup Guide".

For 1st time Working with ZedBoard ,Here is the Steps for "Cypress USB-to-UART Setup Guide" from this PDF:

3. Unplug your USB UART device.
4. Browse to the Cypress's website: https://secure.cypress.com/?mpn=CY7C64225-28PVXC
5. Scroll down to **Technical Documents** and download the "**Microsoft Certified USB UART Driver**" for your PC's operating system. You may need to setup or log into a Cypress account to download the driver.
6. Unzip the driver to a convenient location and run "Setup.exe". The new driver will be installed.
7. Connect the evaluation board's Cypress UART-USB interface to the PC.

23. Now program the Bitstream to the ZedBoard:
Click on Program FPGA option and Click on "Program", it must look like:

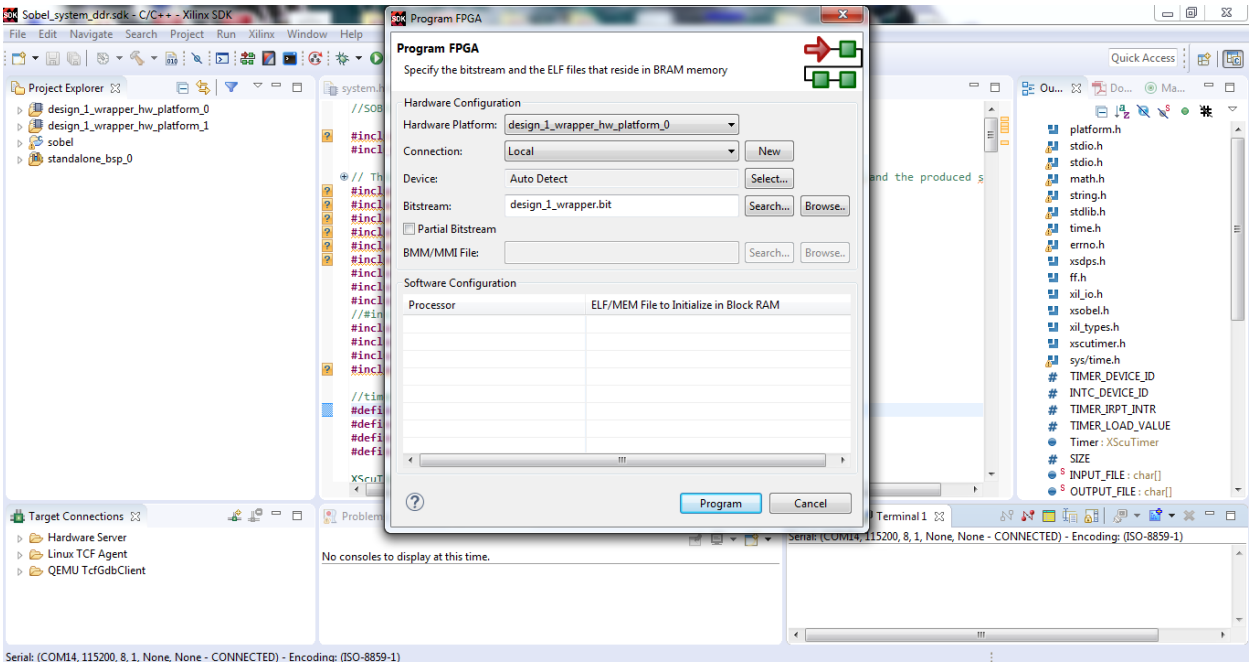


Fig: Program FPGA Option

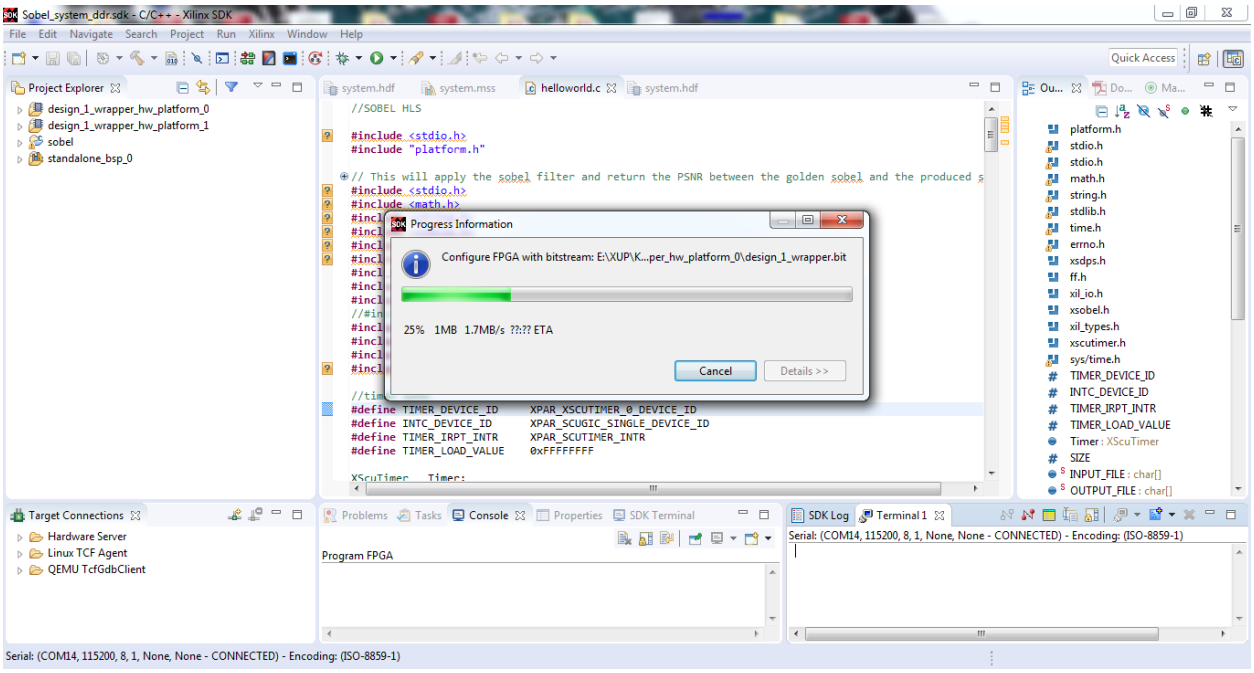
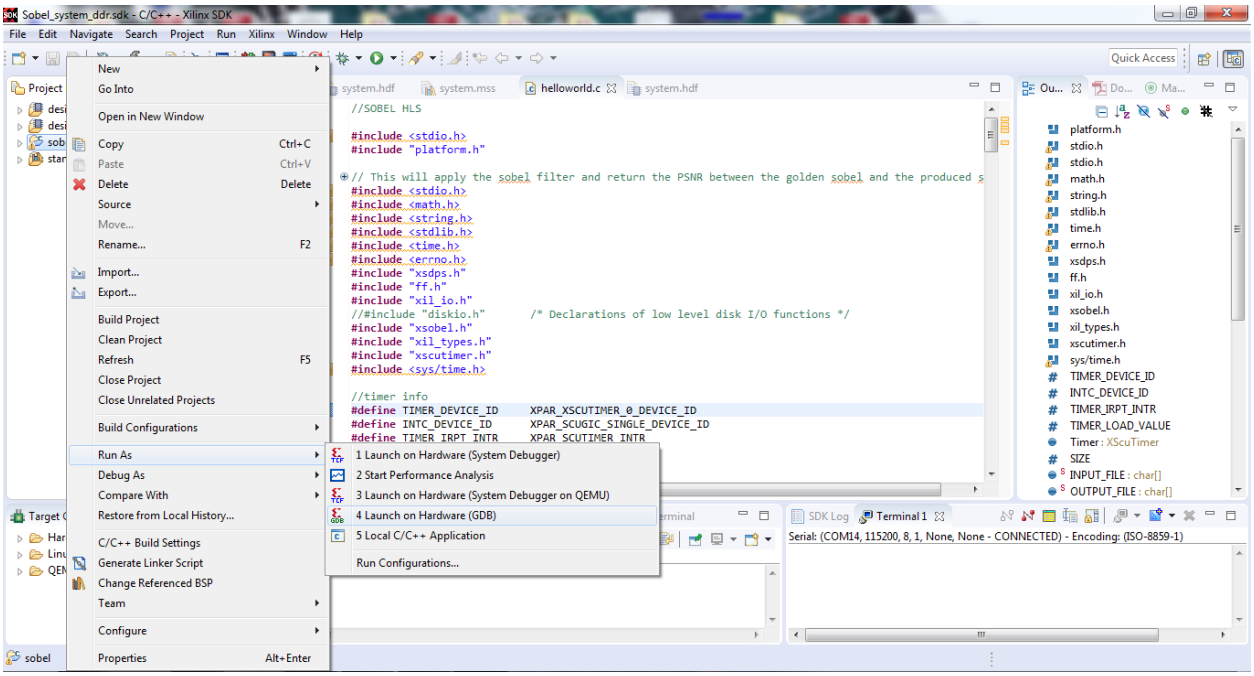
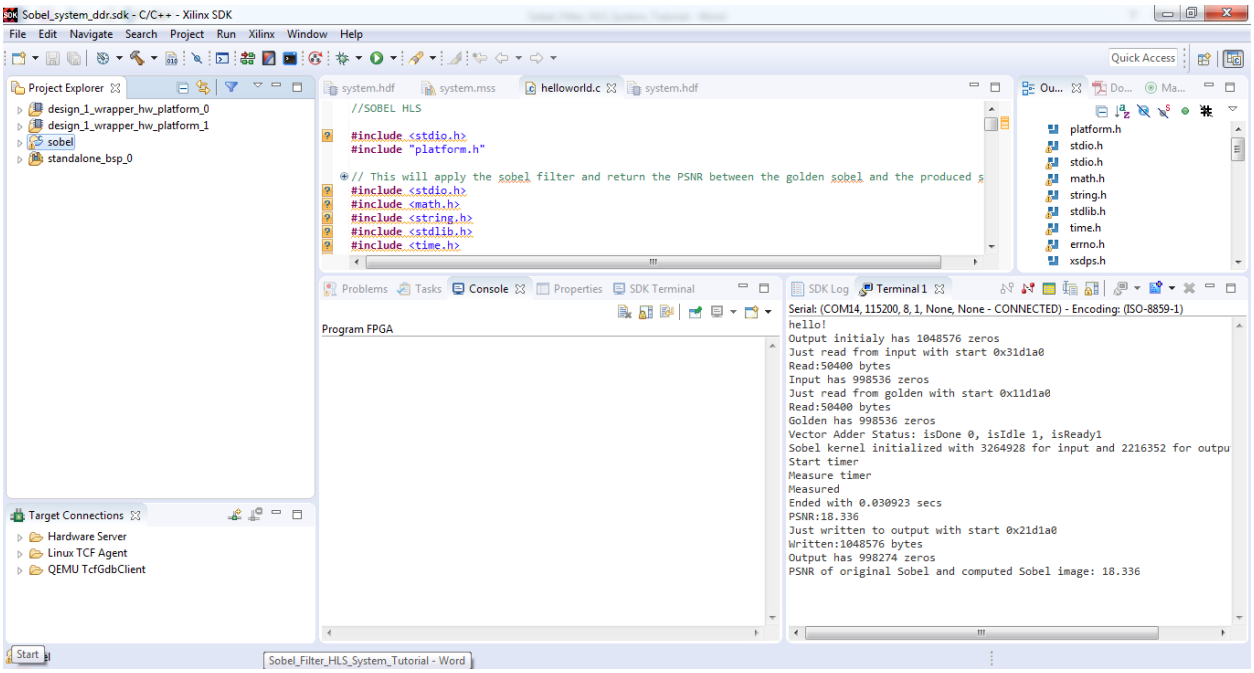


Fig: Programming FPGA, Uploading the Bitstream to ZedBoard

24. **Now LED "LD12" of ZedBoard will glow, this is "Blue" LED.**
25. **Now we are ready for launching the SDK project on ZedBoard:**
    Right click on the Project "Sobel", goto "Run As", and click on "Launch on Hardware(GDB)".



26. After program run successfully on your Zynq Processing System, you will get this update on the UART Terminal:



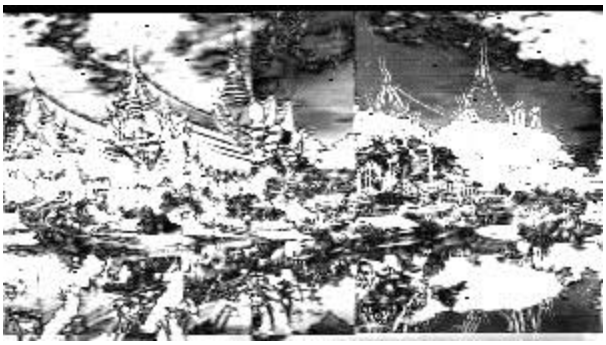
27. So the Project Run well. The Result shows the time taken by the Zynq PS for processing this Binary Image File and Generating it's Sobel Output. Output has been generated on the SD card of the ZedBoard.

III. Now again back to Linux System [For Generating JPEG Image file from the output.gre File].

28. **Take the SD Card from the ZedBoard and Plug into the Linux System:**
29. **Copy the "OUTPUT.GRE" file on the desktop of your Linux [Ubuntu in our case] System.**
30. **Open the Terminal and change the directory to the Desktop of the Linux [command is shown in II Section above].**
31. **Now use this command for converting the "OUTPUT.GRE" file into the "output.jpeg".**
    In this step we will convert the binary file into the JPEG file of the original size 300x168 Pixels [state on step 15 above]

```
convert -depth 8 -size 300x168 GRAY:OUTPUT.GRE output.jpeg
```

32. Finally you will get the output image after implementation of Sobel Filter. Our Output Image is like this:



Sobel Filtered Output Image.

33. We think there may have some minor changes on the conversion of .GRE file into the JPEG, the above image is not very well. This distortion is happen due to it's conversion of binary into JPEG and vice versa.

However the main Objective of this session is make you familiar with the process [support you with the main article of Sobel HLS kernel implementation on Zedboard] of creating Accelerator [HLS Kernel's], Interfacing it with Zynq PS and other peripherals on VIVADO IP integrator and writing the Software Application on VIVADO SDK for Zynq PS for reading binary image from the SD card from ZedBoard and process it then generate the binary output on the SD card.

Thank you for following this tutorial!

Hope it helps you on learning the process for the HLS Sobel Filter Implementation on ZedBoard!

For any Queries and more tutorials, please visit:

www.logictronix.com or www.digitronixnepal.com

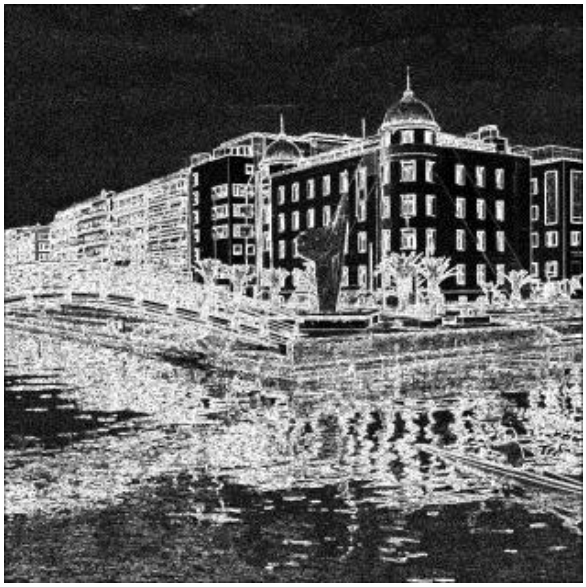
We respond on your queries from email:

info@logictronix.com

Original Article from FPGAWorld

Sobel Vivado HLS Kernel using AXI full interface

ON 9 MAY 2017 BY NIKKATSA



In this post we will explore the steps from creating and exporting an HLS IP to integrating it in a Zynq Design. Finally we will create an app in SDK that uses this peripheral in order to apply a sobel filter in an image read from a SD card connected to the board. We use the Zedboard development kit and Vivado 2016.4 tools for this project.

The project is here.

First of all we create our Sobel filter as a HLS Kernel. The implementation is basic but we added some pragmas and techniques in order to achieve better performance both in memory transactions and computations. Bear in mind that the problem is memory bound so we focused in this aspect.

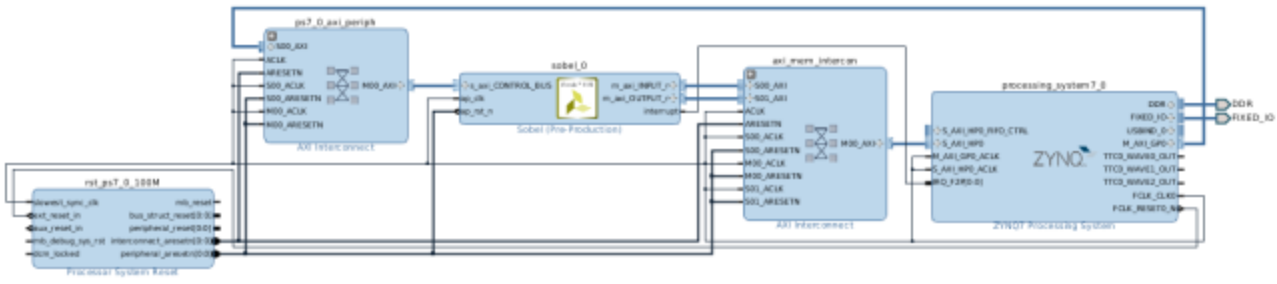
In order to reduce the bottleneck created by the ddr accesses we used block rams to store the input data, process it, and then write it back to the ddr. For the transactions we used the memcpy command which gives us the ability to transfer data with bursts. The size of the block ram is due to the restrictions of the block ram slices in the Zedboard. In other case we would have used bigger block rams because we would have benefited handsomely from the bursts.

After we simulate the kernel and verify its correct functionality then we export it as an IP.

In order to use the implemented IP in Vivado we have to add the HLS project in the repository manager. Then in the add IP icon type the name of your IP and insert it in the block design.

Now its time to build our system. First of all we add the processing system and enable the S_AXI_HPO interface(set in 64 data width). Then we have to add a concat module as well as a gpio peripheral which we will use to connect the ap_ctrl interface with it.

The final block design screenshot is this.



We export the hardware and launch the SDK. We create a board support package and enable the xilffs library in order to have access to a SD card. We create a new application project and enable the -m flag in the gcc linker.

First of all we have to mount the sd card so we use this command to do that.

```
static FATFS FS_instance;

 const char *Path = "0:/";

 FRESULT result;

 result = f_mount(&FS_instance,Path, 0);

 if (result != FR_OK) {

 printf("Cannot mount sd\n");

 return XST_FAILURE;

}
```

Then we must properly initialize the GPIO and the HLS kernel with these functions

```
void sobel_init(unsigned char *input_addr,unsigned char *output_addr){

 //Kernel - Init

    XSobel_CfgInitialize(&Sbl,&Sbl_CONF);

    XSobel_InterruptGlobalDisable(&Sbl);

    XSobel_InterruptDisable(&Sbl, 1);

    XSobel_Set_in_pointer(&Sbl,(u32)input_addr);
```

```
    XSobel_Set_out_pointer(&Sbl,(u32)output_addr);

    printf("Sobel kernel initialized with %d for input and %d for
output\n",(int)XSobel_Get_in_pointer(&Sbl),(int)XSobel_Get_out_pointer(&Sb
l));

}
```

The data structures and the functions are in the header files generated for the peripherals.

In order now to read an image from the sd card we use these set of commands

```
FRESULT f_in, f_out, f_golden;



Log_File = (char *)INPUT_FILE;

 f_in = f_open(&file1, Log_File,FA_READ);

 if (f_in!= FR_OK) {

 printf("File INPUT_FILE not found\n");

 return XST_FAILURE;

 }



f_read(&file1,&input[0],SIZE*SIZE,&readBytes);
```

```
f_close(&file1);
```

and to write

```
Log_File = (char *)OUTPUT_FILE;

 f_out = f_open(&file3, Log_File, FA_CREATE_ALWAYS | FA_WRITE);

 if (f_out!= FR_OK) {

 printf("File OUTPUT_FILE not found\n");

 return XST_FAILURE;

 }



off =0;

 uint writtenBytes=0;

 while(writtenBytes!=SIZE*SIZE) {

 f_out = f_write(&file3,&output[off],SIZE*SIZE,&writtenBytes);

 if (f_out!=0) {

 xil_printf(" ERROR: f_write2 returned %d\r\n",f_out);

 return XST_FAILURE;
```

```
 }

 off+=writtenBytes;

}

f_close(&file3);
```

The most important part is the one that calls the init functions with the proper values in order to start the kernel and then monitor if the kernel has finished the processing.

```
 sobel_init(input,output);

 XSobel_Start(&Sbl);

 while(!XSobel_IsDone(&Sbl)){}
```

The process described above can be used for any peripheral that can read and write to the ddr memory.

Summary

After some experiments the average time in seconds of each implementation is:

```
SOFTWARE -O0|SOFTWARE -O3|HLS KERNEL

1.62s       |0.09s       |0.03s
```

As we can see this implementation achieves better performance than any software implementation.

Authors -Fpgaworld