Image Processing using Dynamic Partial Reconfiguration on Zynq 7020

Shruti Karbhari
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mail: skarbhari@oakland.edu

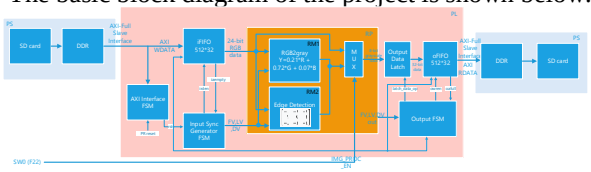
***Abstract*—This project demonstrates the use of Dynamic Partial Reconfiguration in the Xilinx Zynq 7020 for image processing. The PCAP interface is used to selectively load two image processing modules – color space conversion (rgb2gray) or edge detection (2D convolution).**

I. Introduction

Complex image processing algorithms are often resource-intensive when embedded in FPGAs or programmable logic. As a result, multiple algorithms may not fit in the resources available in a given device. If such algorithms don't need to reside in the PL at the same time, they can be selectively swapped in and out. This project demonstrates the Dynamic Partial Reconfiguration (DPR) methodology provided by Xilinx for the Zynq SOCs. These SOCs provide a PCAP interface to reconfigure portions of the Zynq PL. This interface can be controlled from the PS.

II. System Overview

The basic block diagram of the project is shown below.



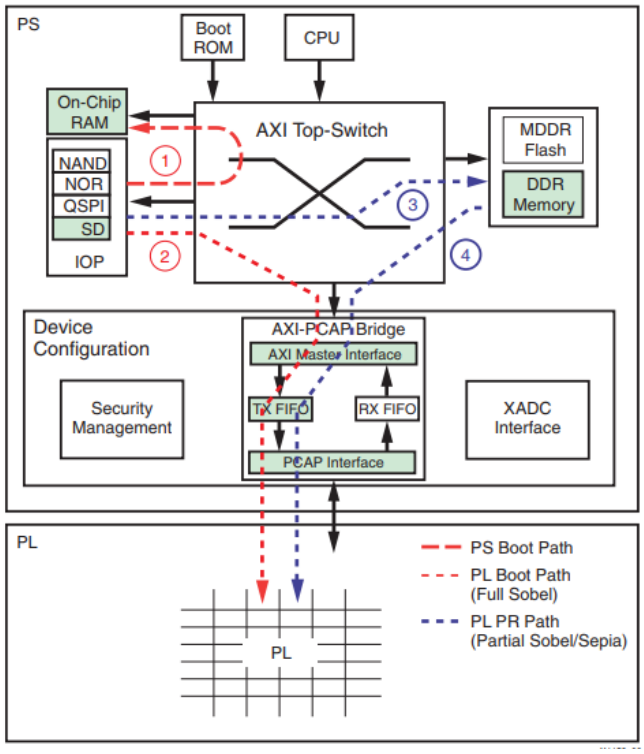
The project outline is as follows:

1. An input image of size 256x256 and format RGB888 is stored in the SD card. The PS reads it from the SD card into the DDR using the `xilffs` library.
2. The image data is transferred by the PS from DDR to the PL over an AXI-full slave interface. This interface is also used to transfer the image from the PL to the DDR.
3. The input sync generator FSM generates the frame valid, line valid and data valid sync signals based on the iFIFO status signals.
4. The output FSM also controls the data being written into the oFIFO.
5. The image processing reconfigurable partition contains two reconfigurable modules. Each of these modules implement an image processing algorithm. Both algorithms use the same input interface and the same output interface. Hence the control FSM does not need to be in the RP.

III. Dynamic Partial Reconfiguration in Xilinx Zynq

FPGA technology provides the flexibility of on-site programming and re-programming without going through re-fabrication with a modified design. Partial Reconfiguration (PR) takes this flexibility one step further, allowing the modification of an operating FPGA design by loading a partial configuration file, usually a partial BIT file. After a full BIT file configures the FPGA, partial BIT files can be downloaded to modify reconfigurable regions in the FPGA without compromising the integrity of the applications running on those parts of the device that are not being reconfigured.

The Device Configuration interface (DevC) provides an AXI-PCAP bridge for interfacing the PL configuration logic. The figure below illustrates the device configuration flow.

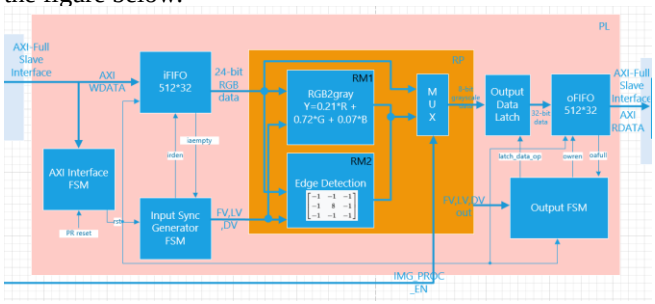


The AXI-PCAP bridge converts 32-bit AXI formatted data to the 32-bit PCAP protocol and vice versa. A transmit

and receive FIFO buffer data between the AXI and the PCAP interface. A DMA engine moves data between the FIFOs and a memory device, typically the OCM, the DDR memory, or one of the peripheral memories. The 32-bit PCAP interface is clocked at 100 MHz and supports 400 MB/s download throughput for non-secure PL configuration and 100 MB/s for secure PL configuration where data is sent only every 4th clock cycle. To transfer data across the PCAP interface a DevC driver function needs to be called. The driver will take care of setting the correct PCAP mode and initiating the DMA transfer. The function call will only return after both the AXI and the PCAP transfers are complete.

IV. Image Processing IP Overview

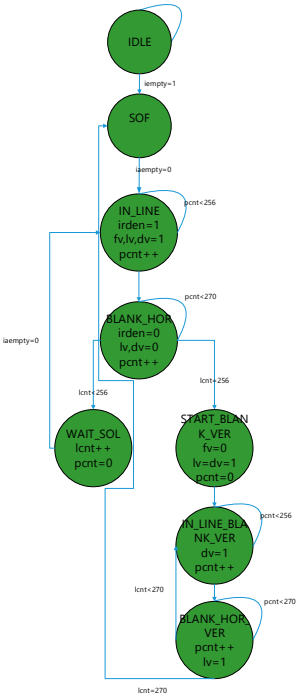
The main image processing capability is provided by the image processing IP. A detailed view of this IP is shown in the figure below.



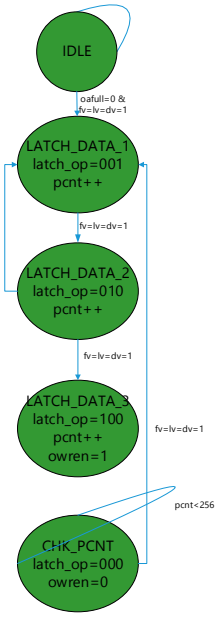
The IP interfaces to the PS through an AXI4-Full Slave interface. It decodes the AXI4 transactions and writes incoming data into an input FIFO and provides outgoing data from an output FIFO. Both FIFOs are 512 deep and 32-bits wide.

The image processing algorithms are contained in the reconfigurable partition (RP). This partition has two reconfigurable modules (RM). The two algorithms being used are RGB to grayscale conversion and 2D convolution (used as edge detection). Both algorithms use the same input interface (24-bit RGB, frame valid (FV), line valid (LV) and data valid (DV)) and the same output interface (8-bit grayscale, frame valid (FV), line valid (LV) and data valid (DV)).

An input sync generator FSM reads data from the input FIFO and provides it to the RM. It monitors the almost empty flag from the input FIFO. The threshold is 256. This means that the almost empty flag de-asserts when it has at least 256 words (or a line of the input image) in the FIFO. When this flag de-asserts, the FSM generates sync signals FV, LV and DV for one line (or 256 pixels). The 24-bit pixel data is also sent along with these control signals to the RM. This process continues until all the lines in the image are sent to the RM. The input FSM is shown below.



The output FSM writes data to the output FIFO based on the sync signals from the RM. Four 8-bit grayscale pixels are combined to create a 32-bit word which is written into the FIFO. The flag to latch the 4 pixels and the flag to write the word into the output FIFO is also created by the output FSM. The FSM monitors the almost full flag from the output FIFO. The threshold is 64 (since one line is 64 words on the output). When this flag deasserts, the FSM writes the data words into the FIFO based on signals FV, LV and DV for one line. This process continues until all the lines are written to the output FIFO. The output FSM is shown in detail below.



V. Reconfigurable Module 1: RGB2Gray

The RGB to grayscale conversion is done using the formula:

Y=0.21*R + 0.72*G + 0.07*B

Since the calculation is done in fixed-point format U0.8, the coefficients are converted as follows:

0.21 is expressed as 0.00110101 = X"35"
0.72 is expressed as 0.10111000 = X"B8"
0.07 is expressed as 0.00010001 = X"11"

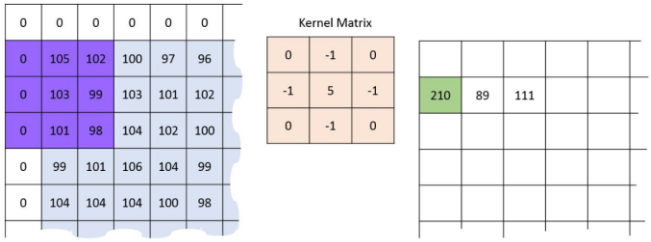
After the addition, the fractional part is ignored by dropping the lower 8 bits. The module has a latency of 3 clock cycles.

VI. Reconfigurable Module 2: Edge Detection

The edge detection module is a 2D-convolution with the kernel

-1 -1 -1
-1 8 -1
-1 -1 -1

The 2D convolution process on an image is basically the process of adding each element of the image to its local neighbors, weighted by the kernel. The kernel is overlaid on top of the image, the center of the kernel being on the pixel of interest. Each element of the kernel is multiplied with the corresponding pixel. All these are summed together to get a final value for the pixel of interest. An example is shown below.



The value for pixel (2,1) is calculated as:

210 = 0*0 + 105*-1 + 102*0 + 0*-1 + 103*5 + 99*-1 + 0*0 + 101*-1 + 98*0

For our use case, the image size is 256x256. The pixels on the border (row 0 and 255 and pixel 0 and 255 in each row) do not have all the required pixels to do the convolution. For these pixels, the output from the module is set to zero.

Given the negative values in our kernel, it is possible that some pixels result in a negative value. In such a case, the value is converted to its absolute value. Since the input image has 3 channels, we get three edge results after the convolution is complete. The output of the RM needs to be 8-bit grayscale so these three values are averaged to get one value. After the averaging step, in case the output exceeds 255, it is saturated to 255.

VII. Image Input/Output Process in PS

The PS transfers the image data from the SD card to DDR using the `xilffs` library. `xilffs` is a generic FAT file system that is primarily added for use with SD/eMMC driver.

The input image is stored as the R channel followed by the G channel followed by the B channel, all in one binary file. The PS application combines the three channels together to get words with all three channels.

The input image in our case is 256x256 RGB. The output image is 256x256 grayscale. The PS sends one line i.e. 256 32-bit words at one time to the IP. For subsequent lines, it sends one line into the IP and reads one line i.e. 64 32-bit words out of the IP. This continues until all the lines are sent. At the end, the last line is read out to flush out the output FIFO.

VIII. Dynamic Partial Reconfiguration Process on the PS

The basic flow followed by the PS application is:

1. Program the FPGA with the RGB2Gray bitstream.
2. Send the image, run image processing, retrieve the image, write to SD card.
3. Transfer the edge detection bitstream to the PL.
4. Reset the RP.
5. Send the image, run image processing, retrieve the image, write to SD card.
6. Transfer the RGB2Gray bitstream to the PL.
7. Reset the RP.
8. Send the image, run image processing, retrieve the image, write to SD card.

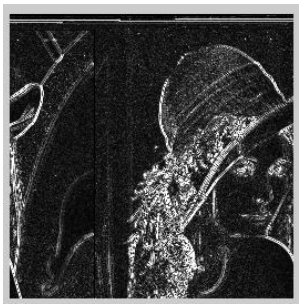
The important thing to remember is to reset the RP after the partial bitstream has been transferred to the PL. This ensures that the input and output FIFOs are reset and the FSMs and RMs start from the idle state. This is done via a simple software command (we write the word 0xAA995577 onto address 101100).

IX. Results

The dynamic partial reconfiguration process worked in principle and the application was able to swap the bitstreams in and out of the PL. The RP reset logic worked correctly and the results were evident with both RMs (RGB2Gray and Edge Detection).

The hardware output of RGB2Gray before and after PR matched perfectly. It also closely matched the Matlab reference. The images are shown below.


The hardware output of Edge Detection before and after PR matched each other perfectly. However, there is some problem with the overall Edge Detection design when the dynamic bitstream is created. The images are shown below to illustrate the problem.

The following debug steps were taken to root cause this issue:

1. The image processing IP was instantiated in a static design and the bitstream was flashed on to the FPGA. The PS application of the static design was used. This gave an image which closely matched the Matlab reference image.
2. The image processing IP was instantiated in a static design and the bitstream was flashed on to the FPGA. The PS application of the dynamic design (with the PR reset) was used. This gave an image which closely matched the Matlab reference image.
3. The image processing IP was used in the dynamic design and the edge detection configuration bitstream was flashed on to the FPGA. The PS application of the dynamic design was used. This resulted in the incorrect shifted image. The same result was obtained after partial reconfiguration was done.

This means that the issue is with the Edge Detection in the DPR flow. It is also most likely on the output side as the input side is the same for color conversion. The next step would be to put an ILA instance in the IP and examine the signals related to the output FIFO.

X. Conclusion

The DPR methodology has its advantages but the following challenges need to be addressed:

1. Floorplanning: The selected Pblock should accommodate elements required by all RMs in the least amount of space.
2. Interface definition: The interface should be common between the different RMs in a given RP.
3. Proper reset procedure should be followed to ensure the RP starts from an idle state
4. Due to the TCL flow that is used to set up the dynamic design, the ability to include ILA cores in the design is hampered. Care should be taken to bring out relevant signals from the IP into the block design so they can be viewed in the hardware.

XI. References

[1] https://en.wikipedia.org/wiki/Grayscale

[2] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_4/ug909-vivado-partial-reconfiguration.pdf