XAPP1231 (v1.1) March 20, 2015

Partial Reconfiguration of a Hardware Accelerator with Vivado Design Suite for Zynq-7000 AP SoC Processor

Christian Kohn

Summary

As systems become more complex and designers are asked to do more with less, FPGA adaptability has become a critical asset. While Xilinx® devices have always provided the flexibility to do on-site device reprogramming, today's tougher cost, board space, and power consumption constraints demand even more efficient design strategies.

Xilinx Partial Reconfiguration (PR) extends the inherent flexibility of the FPGA by allowing specific regions of the device to be reprogrammed with new functionality while applications continue to run in the remainder of the device. Partial Reconfiguration offers the following key advantages over traditional full configuration:

- Reduced hardware resource utilization: the designer can fit more logic into an existing device by dynamically time-multiplexing functions of the design
- Increased productivity and scalability: only the modified function needs to be implemented in context with the already-verified remainder of the design
- Enhanced maintainability and reduced system down-time: new functions can be deployed and inserted dynamically while the system is up and running

This application note describes the tool flow, concepts and techniques for Partial Reconfiguration on Xilinx Zynq®-7000 AP SoC devices through Processor Configuration Access Port (PCAP) interface using the Vivado® Design Suite. It is a newer version of the *Partial Reconfiguration Hardware Accelerator for the Zynq-7000 AP SoC* XAPP1159 [Ref 1] which is based on the ISE Design Suite.

The provided reference design is built on the ZC702 Base Targeted Reference Design (TRD) [Ref 2], an embedded video processing application that demonstrates the benefits of offloading a compute-intensive video filter algorithm from software onto Programmable Logic (PL). The design demonstrates how to use software-controlled Partial Reconfiguration to dynamically reconfigure part of the logic with different video filter IP cores and observe the video output on a monitor.

Introduction

The Zynq-7000 AP SoC integrates a dual-core ARM Cortex-A9 based Processing System (PS) and Programmable Logic (PL) in a single device. This reference design makes use of both PS and PL portions and demonstrates how it is best to separate control (mapped onto the PS) and data path (mapped onto the PL). The PL implements a powerful, high-definition video design that consists of a capture pipeline, a memory-to-memory processing pipeline, and a display pipeline (see Figure 1).

The PS is used to configure the individual IP cores inside the PL and to control the data flow between the individual pipelines. The provided reference design offers the user to choose between two different Linux OS-based software applications running on the PS: one is controlled from the command line, the other provides a GUI using the Qt framework.

The ZC702 Base TRD [Ref 2] demonstrates the value of offloading a compute-intensive edge detection algorithm (Sobel filter) onto the PL. The benefits are two-fold:

1. Achieving real-time processing of a full HD video stream due to hardware acceleration
2. Freeing up CPU resources for user-specific tasks such as rendering a graphical user interface (GUI) on top of the processed video stream.

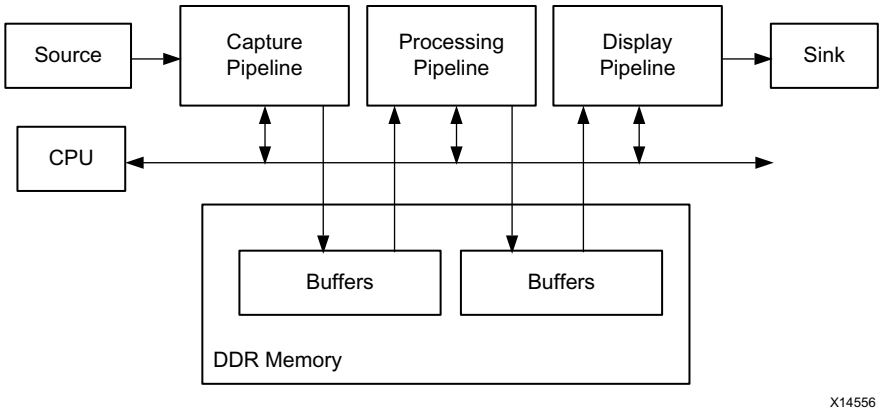

Figure 1: **Video Pipeline Architecture**

Based on the ZC702 Base TRD, this application note describes the methodology of implementing different video processing accelerators in the PL by using PR to load functionality on demand. The image filters used in this design are:

- **Posterize**: Converts a continuous gradation of tone to several regions of fewer tones, with abrupt changes from one tone to another.
- **Sobel**: Creates an image which emphasizes edges and transitions.
- **FAST** (Features from Accelerated Segment Test): A corner detection algorithm that can be used to extract feature points (for example: for object tracking).

Figure 2, page 3 shows a comparison of the original, versus the various filtered images. The RTL for the different video filter IP cores are generated from C-algorithm descriptions using the Vivado® High-Level Synthesis (HLS) tool. The algorithms and their implementations are not further discussed because it is not the scope of this application note.

See *Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries* (XAPP 1167) [Ref 3] for more details.

Figure 2: **Comparing Image Filters: Original (top left), Posterize (top right), Sobel (bottom left), FAST (bottom right)**

This application note assumes that the user is familiar with the ZC702 Base TRD [Ref 2] and its hardware and software components. The focus of this work is to provide guidance on:

- Identifying functions that can benefit from Partial Reconfiguration.
- Design considerations and interface requirements for Reconfigurable Modules (RM).
- Step-by-step Partial Reconfiguration design flow.
- Implementing software-controlled Partial Reconfiguration through the PS PCAP interface.

System Overview

This section gives a system-level overview of the Programmable Logic (PL) and the Processing System (PS) of the provided reference design. Figure 3 shows the corresponding block diagram.

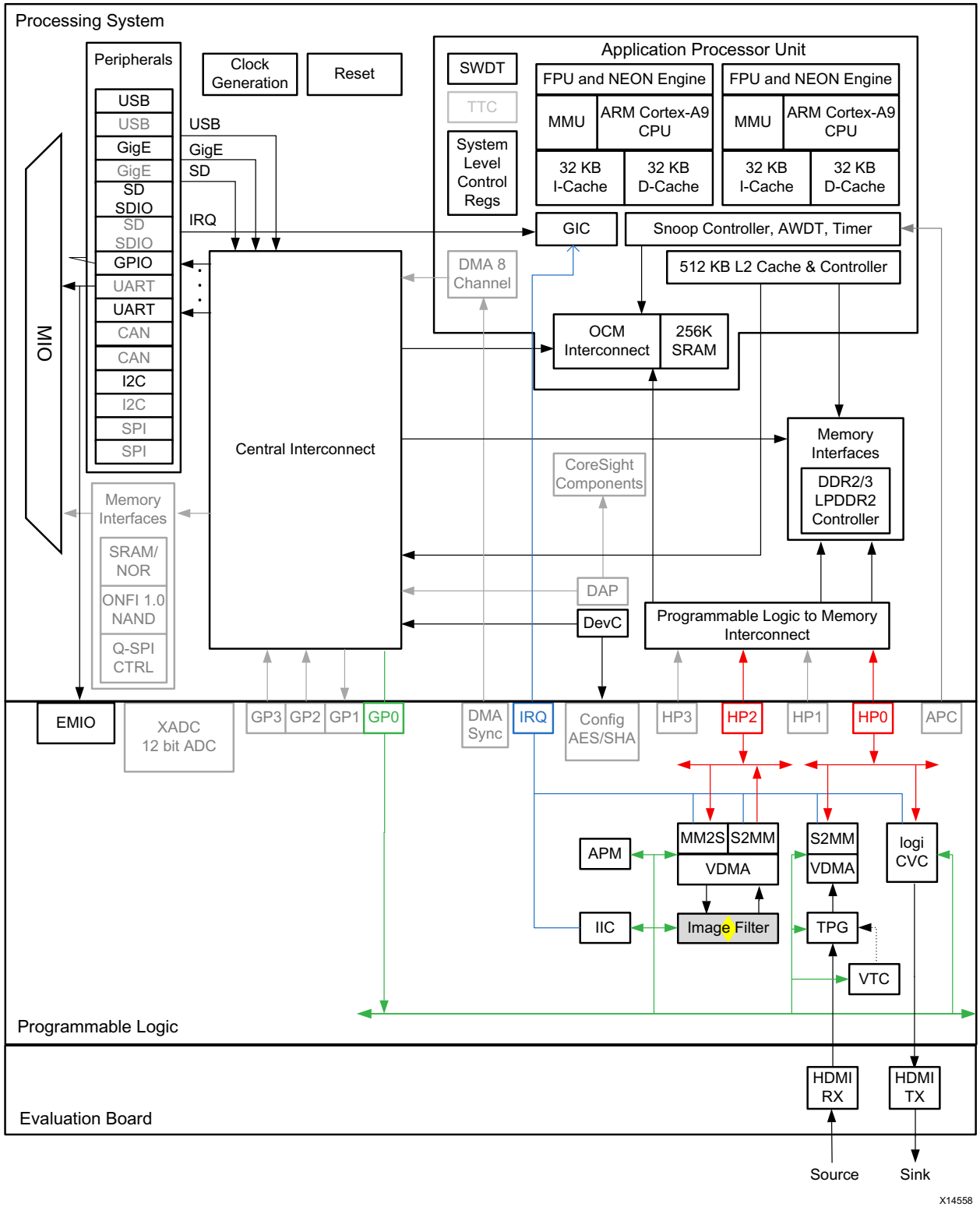

Figure 3: **System-Level Block Diagram of PR Reference Design**

Programmable Logic

The PL reference design includes the following IP cores:

- AXI Interconnect [Ref 13] (not represented in figure)
- AXI Video DMA (VDMA) [Ref 14]
- Compact Multilayer Video Controller [Ref 15]
- AXI Performance Monitor (APM) [Ref 16]
- Video Timing Controller (VTC) [Ref 17]
- Video Test Pattern Generator (TPG) [Ref 18]
- Video In to AXI4-Stream [Ref 19] (not represented in figure)
- AXI IIC Bus Interface (IIC) [Ref 20]
- HLS-generated Image Filter (Posterize, Sobel, FAST)

The video design uses a YUV (16-bit 4:2:2 sub-sampled) video format. Only the GUI layer of the display controller is configured for 32-bit RGB. The pipeline is designed for a maximum video resolution of 1920x1080 progressive at 60 frames per second (1080p60).

Referring to Figure 3, page 4, you see the capture pipeline is connected to the HP0 write port (red) and consists of a VTC, TPG, and VDMA (1 write channel). The VTC generates video timing signals required by the TPG to generate a test pattern inside the FPGA. A VDMA writes the incoming video frames into dedicated buffers inside memory. Optionally, an external HDMI input source can be connected through an Avnet FMC-IMAGEON daughter card [Ref 11]. The IIC controller configures the ADV7611 HDMI receiver on the FMC.

The memory-to-memory processing pipeline connects to the HP2 read and write ports (red) and consists of an HLS Image Filter and a VDMA (1 read and 1 write channel). You can bypass the processing pipeline to display the original unfiltered input stream. If enabled, the VDMA reads a video frame from memory and sends it to the HLS Image Filter. After the video frame is processed, the VDMA write channel writes it back into memory. The HLS Image Filter is the block to be reconfigured in this design. The generic name, HLS Image Filter, is used when are not referring to a particular implementation. The three available filter variants are: Posterize, Sobel, and FAST. You can load the filter modules dynamically at run-time through the PCAP interface using Partial Reconfiguration.

The display pipeline is connected to the HP0 read port (red) and consists of the logiCVC display controller. It has an integrated DMA engine to read buffers from memory and a multi-layer alpha blender to overlay and blend multiple frames. The output video stream is sent to the monitor over HDMI.

The General Purpose Master Port GP0 (green) configures and controls all memory-mapped IP cores. Interrupt signals (blue) are connected to the three VDMA channels, the APM, the IIC and the logiCVC. A PS internal clock generator (`FCLK0`) provides a 100 MHz reference clock to the PL which in turn generates a 150 MHz clock (data path) and a 50 MHz clock (control path). For simplicity, clocking and reset resources, as well as glue logic and bus interconnects, are omitted in this block diagram.

Processing System

This section focuses only on the enabled I/O peripherals: USB0, Ethernet0, SD0, UART1, I2C0, and GPIO (see Figure 3, page 4). Unused I/O peripherals or PS-PL interfaces are grayed out.

- Two General Purpose I/Os are routed to the PL using EMIO:
  - One GPIO resets the I2C Multiplexer on the FMC.
  - One GPIO switches between TPG and external HDMI input.
- The I2C0 controller configures the ADV7511 HDMI transmitter and the SI570 clock synthesizer (provides an accurate video clock) on the ZC702 board [Ref 11].
- SD0 is the boot device, and stores the entire Linux image on an SD card.
- A terminal emulator software can optionally be connected to UART1 to run the command line based demo application.
- Ethernet and USB are fully supported by the provided Linux image.

For more details on other PS features, see the *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585) [Ref 5].

Hardware Design

See the *Zynq-7000 All Programmable SoC ZC702 Base Targeted Reference Design User Guide* (UG925), [Ref 2] for a comprehensive study of the Base TRD hardware design, which is essentially the same as this design, including:

- PS configuration and key features
- IP core configuration, address map and key functionality
- Interrupt IDs for PL peripherals
- GPIO signal mapping
- Clocking and reset structure

Reconfigurable Module Design Considerations

This section discusses design considerations when selecting or designing an IP core for Partial Reconfiguration (PR). In this reference design, the HLS Image Filter is a good candidate for PR for the reasons described in the following sections.

Terminology

The terminology used in this application note is as follows:

- **Reconfigurable Partition (RP)**: Refers to the physical location on the FPGA selected for PR; the remainder of the design is called *static logic*.
- **HLS Image Filter**: Is a black box that is defined by its hardware interfaces and ports and is mapped onto the RP; a specific filter variant can be plugged into this socket.
- **Reconfigurable Module (RM)**: A specific filter implementation that you can plug into the HLS Image Filter. In this design, three RMs are provided: Posterize, Sobel, and FAST.
- **Configuration**: Is an RM in conjunction with the static logic. The three configurations in this design are named after their corresponding variants. A configuration describes a complete FPGA design and produces a full bitstream for the RM combined with the static logic, and a partial bitstream for the RM itself.

Filter Operation

To understand the implications of the design choices, first look at the operation of the HLS Image Filter IP core. The core has an AXI4-Lite interface that the PS uses to configure and control the operation of the core.

First, the IP core must be configured with the video dimensions (width and height) of the present video stream. After you start the core, it operates in auto-restart mode; that means it receives a video frame at its input interface, processes it, and sends the processed frame out on its output interface. The core is immediately ready to process the next incoming video frame without any user intervention, and continues to process video frames in this fashion until the entire processing pipeline and the image filter is halted.

Hardware Interface

The HLS Image filter IP core has the following hardware interfaces and connections to the static portion of the system:

- 32-bit AXI4-Lite interface connected to GP0 Master interface
- 16-bit AXI4-Stream interface connected to VDMA MM2S Master Stream interface
- 16-bit AXI4-Stream interface connected to VDMA S2MM Slave Stream interface
- Reset signal connected to reset controller
- Clock signals connected to the same clock domain as AXI4-Lite and AXI4-Stream interfaces (150 MHz in this design)

When choosing an IP core for Partial Reconfiguration, the designer has to make sure that the hardware interfaces and ports of all RMs are identical with respect to the static portion of the system. In the PR flow, ports are inserted at fixed locations inside the RP. All RMs (that is, all specific implementations mapped onto the RP) need to share the same ports at the same locations relative to the static logic.

Interface Decoupling

Because the reconfigurable logic is modified while the FPGA device is operating, the static logic must ignore data from Reconfigurable Modules during Partial Reconfiguration. The Reconfigurable Modules must not output valid data until Partial Reconfiguration is complete and the reconfigured logic is reset. Common design practices to mitigate this issue are mechanisms such as registering all output signals, handshaking, or disabling bus interfaces to avoid invalid transactions. The static logic should include the logic required for the data and interface management.

In this reference design, pay attention to the AXI interfaces. It is the responsibility of the designer to ensure that there are no pending transactions on the AXI interfaces at the time of the Partial Reconfiguration, otherwise the AXI buses can hang. In this reference design, software is used to prevent this from happening by stopping the entire video pipeline at any arbitrary time before reconfiguring the Reconfigurable Partition and then starting the pipeline back up after reconfiguration has finished. If software cannot guarantee that there are no outstanding transactions, it might be necessary to add custom hardware glue logic to the AXI interfaces of the Reconfigurable Module to appropriately decouple those. Hardware decoupling is beyond the scope of this application note.

To ensure the Reconfigurable Module is in a defined state after reconfiguration, it is recommended to reset the entire Reconfigurable Partition. This can be done automatically by setting the `RESET_AFTER_CONFIG` property in the tool flow when the Reconfigurable Partition is defined. See *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909) [Ref 4], Chapters 4 and 5, "Design Considerations" for more information.

Register Interface and Address Map

The HLS Image Filter hardware register interface is split into two segments.

- The first segment is identical for all variants and provides controls to start/stop the video stream, set the resolution (width and height), and handle interrupts.
- The optional second register segment is filter specific; for example, the Sobel filter provides controls for setting sensitivity threshold, inversion, and filter coefficients.

In this design, only the Sobel filter provides filter specific controls. All filter variants use the same memory base address and the same offset for the second segment (if available). While afore mentioned assumptions are not strict requirements for Partial Reconfiguration, they greatly simplify the software driver architecture.

Full System Design Flow

Implementing a partially reconfigurable FPGA design is similar to implementing multiple non-Partial Reconfiguration designs that share common logic. In the Vivado software tool, a project-less Tcl-based flow must be used.

Prerequisites

To generate the required design check points and constraint files, a three-step approach is used in this reference design:

1. Generate the HLS Image Filter IP cores using the Vivado HLS tool.
    a. First the algorithm is compiled and simulated using C Simulation. The provided testbench reports pass or fail and generates an output image file (BMP format) with the applied filter (see Figure 2, page 3).
    b. Next, a Verilog description of the RTL is generated from the C algorithm using High Level Synthesis.
    c. Finally, the generated RTL is packaged as an IP core for Vivado, which can then be placed in an IP integrator project.
2. The hardware design is based on the ZC702 Base TRD and is provided as a Vivado IP integrator project. The Base TRD has the Sobel filter (obtained in the first step) instantiated.
    a. Remove the Sobel filter by inserting a black box module defining the generic HLS Image Filter interface.
    b. Mark the black box module as a Reconfigurable Module (RM) by setting the `HD.RECONFIGURABLE` property.
    c. Synthesize the modified design and save the corresponding synthesis design check point for later.
3. After synthesis, the design must be floorplanned for PR.
    a. Open the synthesized design and draw a Pblock that contains the Reconfigurable Partition (RP) thereby setting the physical size of the partition and the types of required resources. Xilinx 7 series FPGAs and Zynq SoC devices support reconfiguration of CLBs (Flip-Flops, LUTs, distributed RAM, Multiplexers, etc.), BRAM, and DSP blocks, and all associated routing resources.
    b. Floorplan the RP such that it can accommodate the resources required by each of the RMs (see Table 1, page 10). As a rule, 20% overhead should be accounted for routing resources. The Vivado Partial Reconfiguration design flow ensures that the logic and routing common to each of the multiple designs is identical.

Table 1: **Reconfigurable Partition Resources Required for HLS Image Filter RMs**

	RP	Posterize RM		Sobel RM		FAST RM	
Site Type	**Available**	**Required**	**%Utilized**	**Required**	**% Utilized**	**Required**	**% Utilized**
SLICE	3000	208	7	1076	36	1810	60
DSP48	40	0	0	0	0	0	0
RAMB18	60	0	0	3	5	11	18
RAMB36	30	0	0	0	0	24	80

Where to place the partition inside the PL with respect to the static logic depends on the data flow and how the modules communicate with the rest of the design. A simple strategy is to:

- Implement the configuration with the highest resource utilization without floorplanning,
- Identify the region where most of the resources are placed.
- Create a partition around this region that is big enough to hold all the resources.

In the FPGA logic, a reconfigurable frame is the smallest size physical region that can be reconfigured and aligns vertically to clock region boundaries. In a 7 series device a reconfigurable frame is 50 CLBs high by 1 CLB wide. It is good practice to frame-align the Reconfigurable Partition to achieve best place an route results. The RP physical region is stored as Pblock constraints in the XDC constraints file. Also, frame alignment in 7 series and Zynq processors is required to use the dedicated GSR (`RESET_AFTER_RECONFIG`).

See the *Vivado Design Suite User Guide: Partial Reconfiguration* (UG909) [Ref 4] for details on how to floorplan an RP.

Partial Reconfiguration Design Flow

Partial Reconfiguration in Vivado is a special license-enabled feature. See the Xilinx Partial Reconfiguration web page [Ref 12] for more information. A scripted design flow environment covers all aspects of the design flow, including top-level design synthesis, Out-Of-Context (OOC) (bottom-up) synthesis of Reconfigurable Modules, place and route, as well as generation of full and partial bitstreams.

The first section explains the content of the Tcl script used to set up the PR run.

1. Define target device matching the ZC702 evaluation board, and set up project variables to control the design flow.
2. Import the synthesized design checkpoint (DCP) for the top-level netlist from step 2, page 9 and the generated constraints file containing the Pblock properties from step 3, page 9 of the previous section. Because the top-level design is already synthesized in this case, synthesis is not re-invoked.
3. Define Reconfigurable Modules by pointing to the corresponding HDL/constraints files for the HLS Image Filter IP cores exported in step 1, page 9 of the previous section. Out-Of-Context (OOC) synthesis is performed on each of the modules, and creates synthesis design checkpoints.
4. Define one configuration per Reconfigurable Module and the corresponding settings for implementation, PR verification, and bitstream generation.

The second section explains the individual steps performed during the PR design flow. These steps are fully automated and do not require any user intervention.

5. When building configurations of a reconfigurable design, the first configuration to be chosen for implementation should be the most challenging one. If all RMs in the subsequent configurations are smaller or slower, it will be easier to meet their demands. Based on the utilization figures listed in Table 1, page 10, first implement the FAST configuration.

   The Vivado tools ensure that the resources used to construct the RM are completely contained within the defined physical region of the RP and that no interference with the static portion of the design occurs.

6. After the FAST configuration completes, this design checkpoint is saved, as it will be used later to create full and partial bitstreams. The physical and logical design for the Reconfigurable Module is removed, and the static-only design checkpoint is then locked and saved (see Figure 4, page 12 left). This fixed static result is used for all subsequent configurations.

7. The next configuration is implemented thereby reusing the locked static logic and routing from the FAST configuration generated in the previous step. This step is repeated for the remaining configurations. Place and route design checkpoints are saved for all configurations and Reconfiguration Modules (see Figure 4, page 12 right).

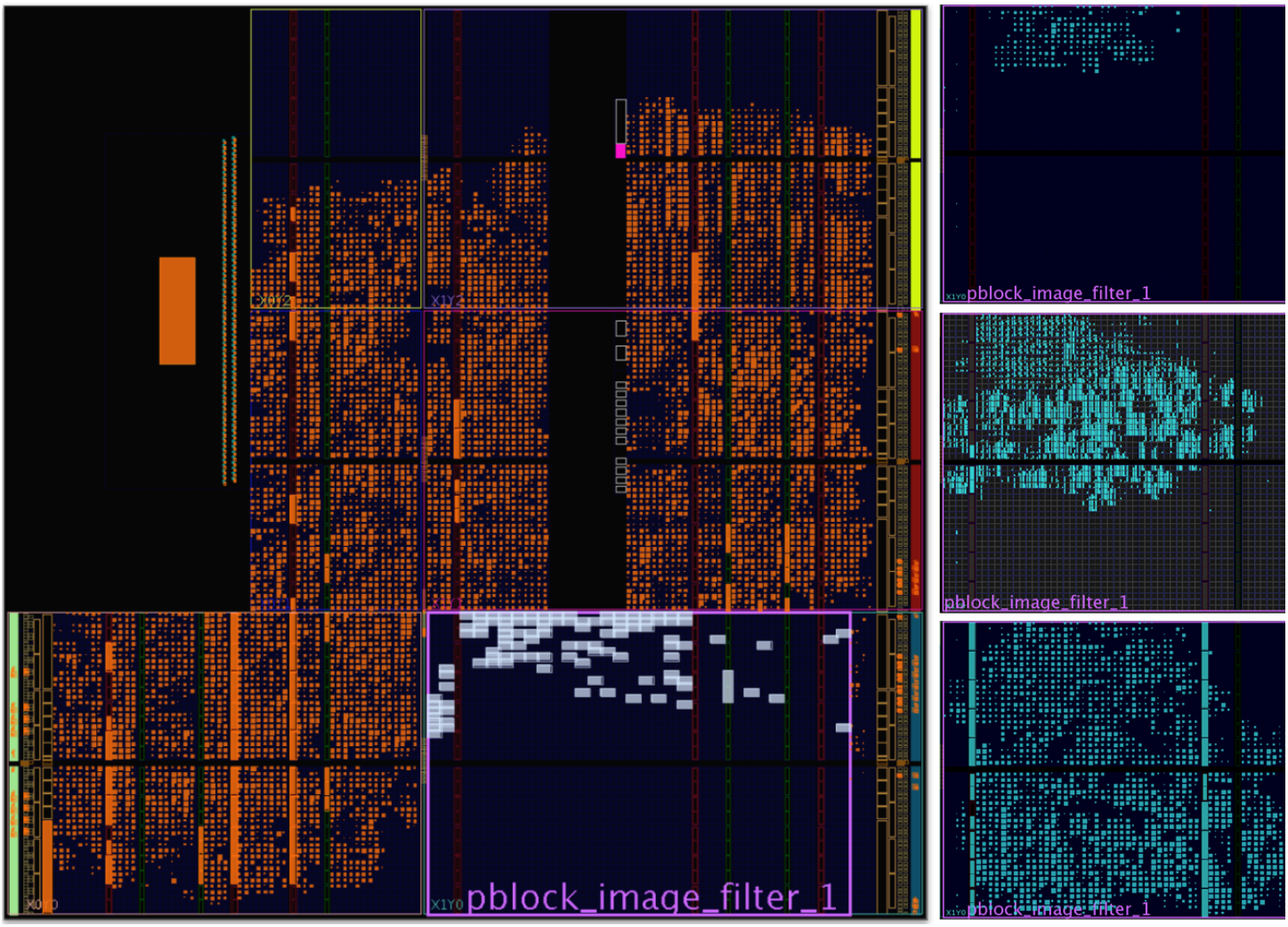

Figure 4: **Vivado Device View: Physical Resources for Static Logic with Black Box (left), Posterize (top right), Sobel (center right), and FAST (bottom right**

8. The Partial Reconfiguration Verify Configuration utility is run to validate consistency between the implementations of all the configurations.
9. Full and partial bitstreams are generated for each configuration. We will use the full bitstream of the Sobel configuration as default start-up configuration when booting the Zynq device.
10. All partial bitstreams are converted to binary format targeting the Zynq PCAP interface. The size of the generated partial binary files is reported; this is required when sending the configuration data to the PL using the DMA engine of the Device Configuration block.

Figure 5, page 13 illustrates the entire design flow in the context of this reference design as described in the previous steps. The yellow diamond symbol represents a black box module in the system. In the end, six bitstreams are generated, one full and one partial bitstream per configuration.

Detailed tutorials for each of the individual steps are provided on the Zynq-7000 Partial Reconfiguration reference design page [Ref 12]. See *Partial Reconfiguration User Guide,* (UG909) [Ref 4] for details on the Partial Reconfiguration design flow and design considerations.

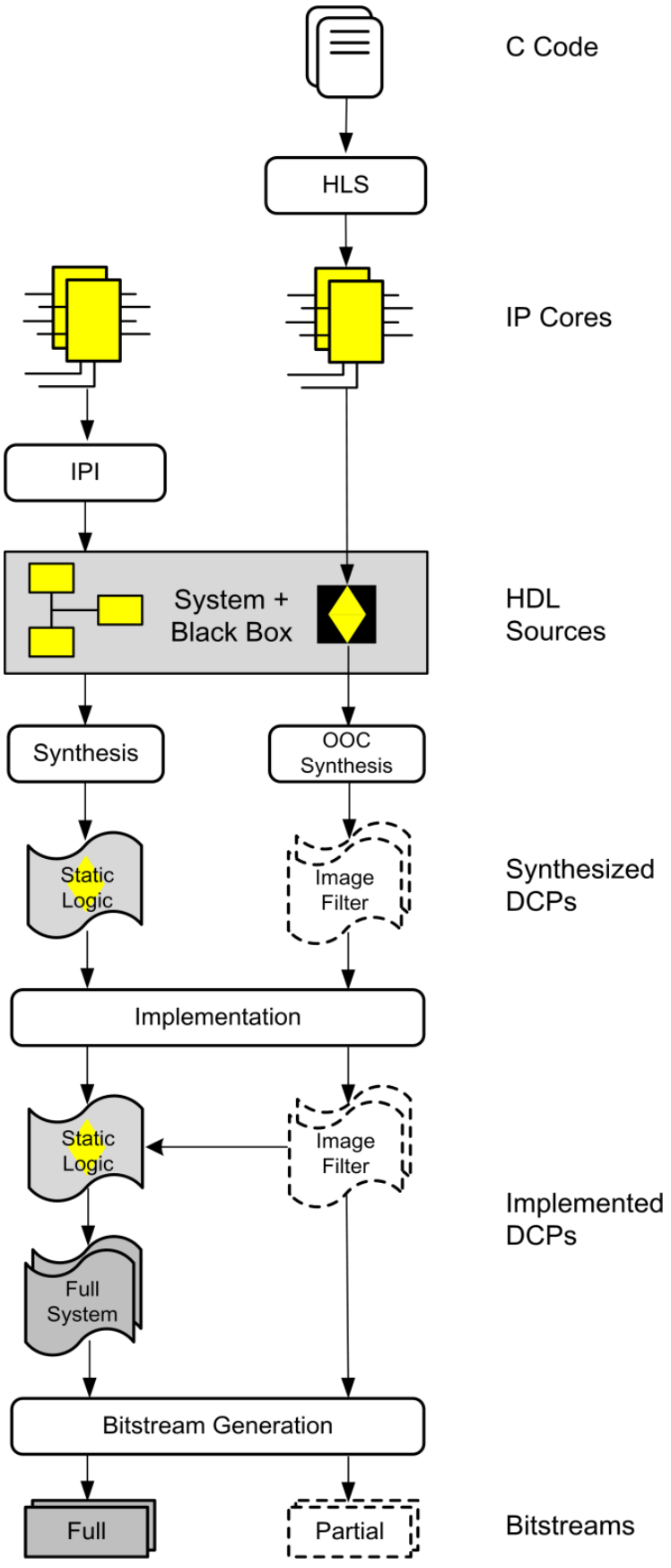

Figure 5: **Full System Design Flow for PR Reference Design**

Device Configuration

The Device Configuration (DevC) block, illustrated in the following figure, has three main sub-blocks:

- AXI-PCAP bridge (center) for interfacing the Programmable Logic (PL)
- Device Security Management (left)
- XADC interface (right)

For this application note, only the AXI-PCAP bridge is of interest.

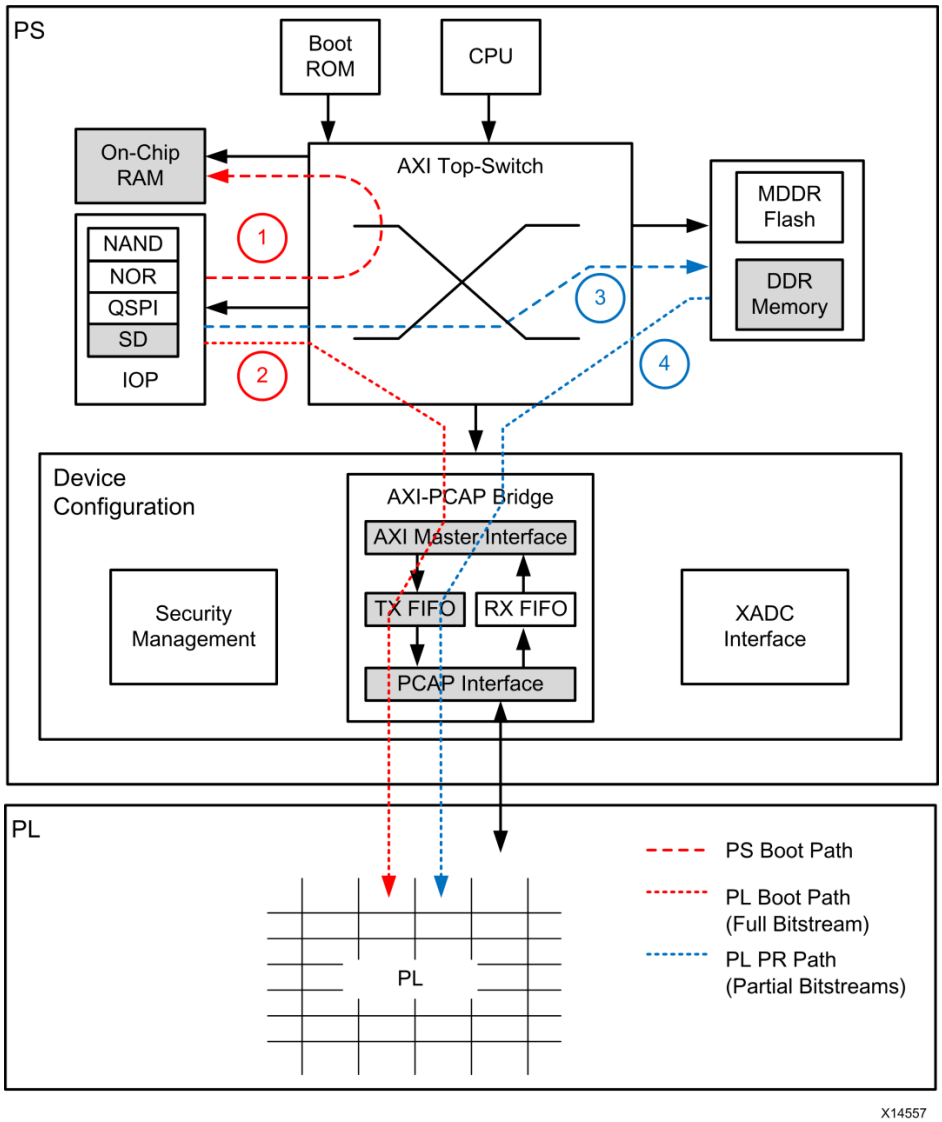

Figure 6: **Device Configuration Flow (Boot and Partial Reconfiguration)**

AXI-PCAP Bridge

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

- A transmit and receive FIFO buffers 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, you must call a DevC driver function. The driver sets the correct PCAP mode and initiates the DMA transfer. The function call returns only after both the AXI and the PCAP transfers are complete.

Device Configuration and Boot Flow

The device configuration and Partial Reconfiguration flow is illustrated in Figure 6, page 14. The sequence is as follows:

1. After power-on reset, the Boot ROM determines the external memory interface or boot mode (SD flash memory) and the encryption status (non-secure). The Boot ROM uses the DMA of the DevC to load the First Stage Boot Loader (FSBL) into on-chip RAM (OCM).
2. The Boot ROM shuts down and releases CPU control to the FSBL which in turn configures the PL with the full Sobel bitstream using the Processor Configuration Access Port (PCAP). The device is now fully configured and operational.
3. The FSBL loads and releases control to the second stage boot loader U-boot. U-boot in turn loads the Linux kernel image, the Linux device tree binary and the Linux root file system. It releases control to boot the Linux kernel. At the end of the boot process, the Linux Qt user application starts automatically.
4. The user application loads the partial bitstreams into DDR memory upon start-up. This is to maximize the configuration throughput over the PCAP interface and hence speed up the configuration time and take advantage of caching.
5. At this point, the application can use the partial bitstreams at any time to modify the pre-defined RP while the rest of the FPGA remains fully active and uninterrupted. This is done by transferring any of the partial bitstreams from DDR to the PL using the PCAP.

A single configuration engine handles both full configuration and Partial Reconfiguration. The task of loading a partial bitstream into the PL does not require knowledge of the physical location of the Reconfigurable Module, because configuration frame addressing information is included in the partial bitstream.

For more information on boot and configuration, see the following documents:

- *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585), [Ref 5], Chapter 6, "Boot and Configuration"
- *Zynq-7000 All Programmable SoC Software Developers Guide (*UG821) [Ref 6], Chapter 3.
- *7 Series FPGAs Configuration User Guide* (UG470) [Ref 7] for information on configuration in general.

Software Application

The included reference design provides two Linux software applications: one is controlled from the command line, the other from a Qt based GUI. Both software applications provide the same core functionality. Options in the reference design are as follows:

- You can choose to display the unfiltered video stream or to apply one of available image filters and display the processed video stream.
- You can select if the selected filter is run as a software algorithm on the PS or accelerated in hardware using the PL.
- You can switch between a Test Pattern Generator (TPG) implemented in the PL or an external video source connected through HDMI. The external video source requires the FMC-IMAGEON daughter card [Ref 11].
- The Qt application lets you select between different TPG patterns, control the edge sensitivity and inversion property of the Sobel filter, as well as the GUI transparency.

The Linux software application is based on the ZC702 Base TRD software. See the *Zynq-7000 All Programmable SoC ZC702 Base Targeted Reference Design User Guide* (UG925), [Ref 2] for details on the software architecture.

This application note focuses on the enhancements made to enable Partial Reconfiguration of the HLS Image Filter. The Partial Reconfiguration feature is enabled by providing a control switch when executing the application.

Linux Software Stack

The Linux software stack is composed of different kernel frameworks and user space APIs.

- The capture pipeline and the memory-to-memory processing pipeline and their subcomponents are implemented as kernel drivers using the Video4Linux (V4L2) framework.
- The Media framework is used to describe the topology of subcomponents and the data flow inside the V4L2 pipelines.
- The display pipeline and its subcomponents are implemented as kernel drivers using the Direct Rendering Manager (DRM) framework.
- The DMA Buffer Sharing (DMA-Buf) framework is used to share video buffers between the V4L2 and DRM pipelines.

Open source libraries and IOCTLs are available to access the various kernel frameworks from user space. More details on the driver architecture can be found on the Xilinx Linux Drivers WIKI page [Ref 21].

The provided user-space software is organized into three libraries and two applications. The three libraries are for performance monitoring, filter control, and general video pipeline control:

- The performance monitoring library provides APIs to read the memory throughput on the HP ports.
- The filter library provides functions to run the different image filter algorithms in software using OpenCV libraries. It also provides APIs to load the partial filter bitstreams and to configure the respective IP cores.
- The video library provides functions to set the resolution, pixel format, color space, and to start/stop the various pipelines using V4L2, Media Controller, and DRM libraries. It also manages the buffer sharing between the individual pipelines.

The two applications build on top of these libraries and provide high level controls that can be accessed from the Linux command line or a Qt based GUI.

Software Control Flow

The three main control variables are modified through a single control function (`vlib_change_mode`).

- The first control variable, `video_src`, selects the video input source which can be TPG or HDMI; the latter requires an FMC-IMAGEON daughter card.
- The second control variable, `filter_mode`, selects the operating mode: options are disabled (`Off`), software (`SW`), or hardware (`HW`).
- The third control variable, `filter_type`, selects the filter to be applied: Posterize, Sobel, or FAST.

When one of the control variables changes due to user interaction, the control function is called and the pipelines are configured accordingly.

Figure 7, page 18 shows the flow of the control function on the left. The control function in turn spawns another thread (`event_loop`) shown on the right to manage the buffer flow between the capture (`video_cap`), processing (`video_m2m`), and display pipeline.

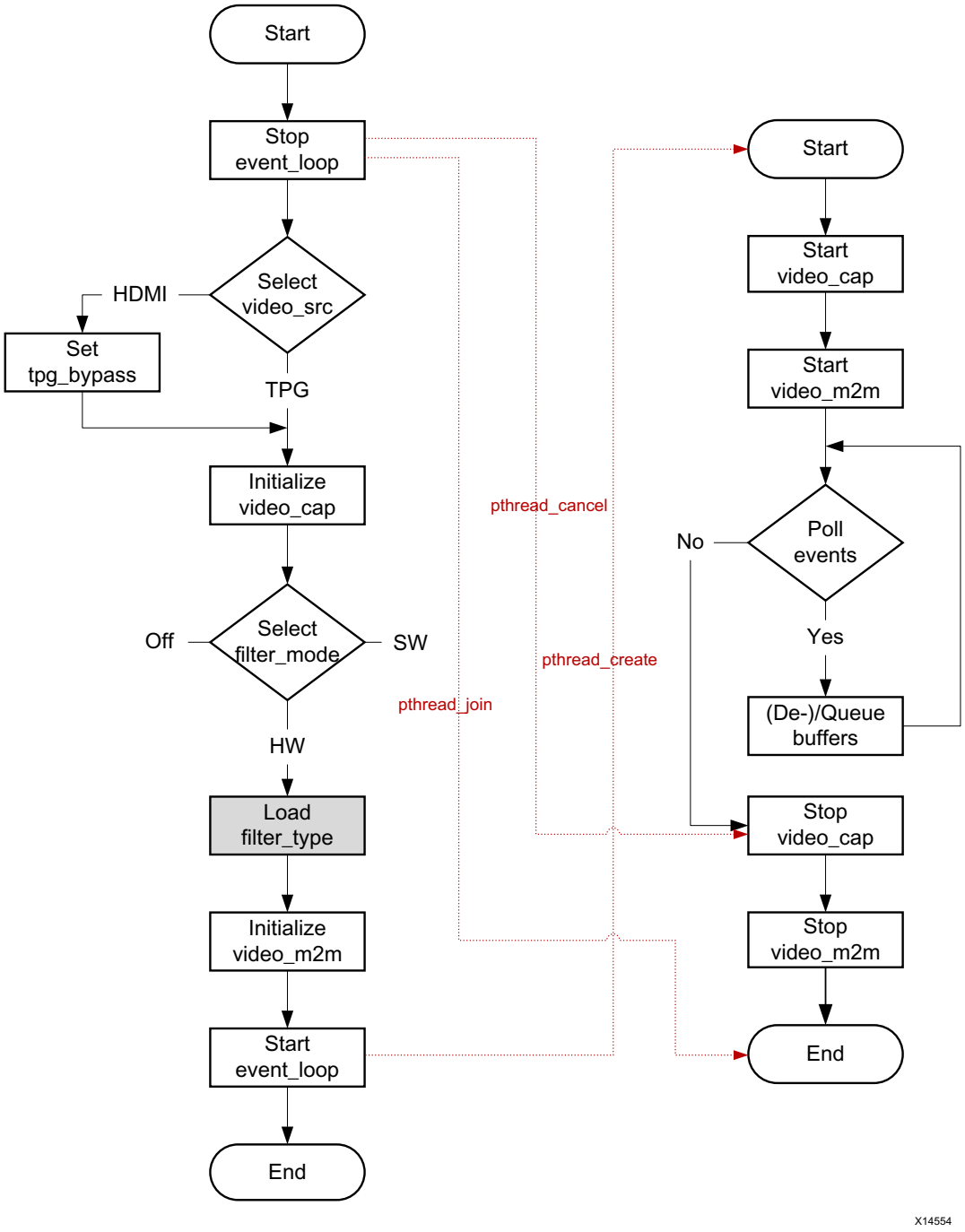

Figure 7: **Software Control Flow**

When the control function is first invoked, it selects the video source and the filter mode. Only the `HW` filter mode path is shown in this figure; the flows for the `Off` and `SW` filter modes are similar. The control function then initializes the capture and processing pipelines, loads the partial bitstream for the filter into the PL and configures the filter. An event loop starts in a separate thread and the control function exits.

The event loop thread starts the capture and processing pipelines and then manages the buffer flow between the pipelines by queuing and de-queuing buffers to the respective DMAs. This operation is done in an infinite loop until the next invocation of the control flow function.

At the beginning, the control flow function checks if there is an event loop thread running already and if so cancels it; the thread is joined with the main control thread and a new event loop thread is started at the end of the function based on the selected filter mode (`Off/SW/HW`).

Adding support for Partial Reconfiguration to the baseline software application as used in the Base TRD is straightforward. As described in Device Configuration, page 14, the partial bitstreams are loaded into DDR memory on start-up to speed up the DMA transfer time for reconfiguration.

The only other additional step is to invoke a function to load the selected partial bitstream from DDR into the PL using the PCAP interface and the DevC driver (gray box). To initialize this DMA transfer, the buffer variable that holds the partial bitstream and the size of the bitstream are passed to the corresponding driver function. After the partial bitstream is loaded, the RP is automatically reset and the filter control registers are in their original state.

It is the responsibility of the user to ensure that there are no pending transactions on any of the AXI4-Streaming interfaces or the AXI4-Lite interface of the filter before initiating Partial Reconfiguration.

In this design, the software application stops the capture pipeline and the memory-to-memory processing pipeline before the new filter is loaded and configured. Only thereafter the pipelines are started again and data is flowing to the display. If software cannot be used to safely stop and re-start the data flow, hardware glue logic might be required to decouple the AXI bus interfaces at the partition boundary during reconfiguration.

HLS Image Filter Driver

As mentioned in Register Interface and Address Map, page 8, the register map is divided into two segments, as follows:

- The first segment provides access to generic control functions shared by all filters.
- The second segment provides filter specific controls.

The corresponding Linux kernel driver implements a V4L2 sub-device.

- The first register segment is accessed through standard functions of the Video4Linux (V4L2) user space API.
- The second segment is accessed through generic IOCTLs for which the filter library provides convenient access functions from user space for the various filter variants.

Given this driver architecture, the same V4L2 sub-device driver can be used for all filter variants because they share the same register map for the first segment. Segment base addresses and offsets are defined in the corresponding device tree node for the HLS Image Filter. The same device tree node is used for all filter variants.

Note: This is a simplification in the driver architecture that cannot be generalized for all use cases and frameworks. In general, you must ensure that the loaded device tree node and driver match the currently active module.

Device Configuration Driver

The Linux DevC device driver uses the `sysfs` interface, a virtual file system provided by Linux to export information about devices and drivers from kernel to user space. Before transferring the partial bitstream over the PCAP interface, the `is_partial_bitstream` device attribute must be set to 1. A write file operation on the DevC device node transfers the partial bitstream. The DevC write driver function initiates the DMA transaction and then waits for an interrupt signaling that the transfer is completed. PR can also be initiated outside a user application from a shell, for example:

```
% echo 1 > /sys/devices/soc0/amba/f8007000.devcfg/is_partial_bitstream
% cat /media/card/partial/sobel.bin > /dev/xdevcfg
```

Note: In most cases, a user application is required to execute pre- and post-Partial Reconfiguration steps to ensure that the system is in a defined state and operational.

Performance Metrics

For this design, two types of performance metrics are discussed: memory throughput and configuration time.

Memory Throughput

For a video resolution of 1920 by 1080 pixels at 60 frames per second (1080p60) and a 16-bit YUV 4:2:2 subsampled pixel format, the total memory throughput of this design with the filter disabled is:

```
(1920 x 1080 x 2 Bytes x 60 Hz) x 2 = 249 MB/s x 2 = 0.5 GB/s
```

or

```
(1920 x 1080 x 2 Bytes x 60 Hz) x 4 = 249 MB/s x 4 = 1 GB/s
```

with the filter enabled. These numbers differ, depending upon the targeted video resolution.

When using the Linux Qt application, a second layer for the GUI is displayed on top of the video layer. The pixel format used for the GUI is 32-bit RGB. The memory throughput for the GUI layer when maximized is:

```
(1920 x 300 x 4 Bytes x 60 Hz) = 138 MB/s
```

or

```
(1920 x 80 x 4 Bytes x 60 Hz) = 37 MB/s
```

if minimized. These numbers are in addition to the values listed above for the video layer.

The GUI plots the measured performance numbers for the HP0 and HP2 ports to monitor the memory throughput in real-time. Alternatively, the system performance analysis feature of the SDK can be used to monitor metrics like throughput and latency using JTAG while the application is running on the target. For more details see the *SDK User Guide: System Performance Analysis* (UG1145) [Ref 9], Chapter 11.

With the 32-bit wide DDR3 memory clocked at 533 MHz (1066 MHz data rate), the theoretical memory throughput is:

```
4 Bytes x 1066 MHz = 4.264 GB/s
```

Therefore the maximum memory utilization for the Qt application with maximized GUI is about 27%:

```
(1 GB/s + 0.138 GB/s) / 4.264 = 0.27
```

Configuration Time

The PCAP configuration time scales fairly linearly as the bitstream size grows with the number of reconfigurable frames with small variances depending on the location and the contents of the frames. The PCAP interface is 32 bits wide and clocked at 100 MHz.

The video pipeline start/stop time is the time required to stop and re-start the current video stream. This affects both the capture and the processing pipeline; the display pipeline keeps running at all times. After the video stream is stopped, the partial bitstream is loaded via PCAP, the capture and processing pipelines are initialized and re-started.

The following table compares the full and partial bitstream sizes of this design, the measured PCAP configuration time, as well as the measured video pipeline start/stop time. The PCAP configuration time is measured between the beginning and the end of the PCAP DMA transfer (`xdevcfg` write function). The time required to stop and re-start the video pipelines is measured between the first `VIDIOC_STREAMOFF` and the last `VIDIOC_STREAMON` IOCTLs.

Table 2: **Configuration Time for Full Versus Partial Bitstream**

XC7Z020 Part	Full Bitstream	Partial Bitstream
Bitstream Size	4,045,564 Bytes	753,848 Bytes
PCAP Configuration Time	83ms	17ms
Video Pipeline Start/Stop Time	N/A	153ms (TPG)/172ms (HDMI)

The video pipeline start/stop time for a full bitstream is not available because reconfiguring the entire PL comes with undesired side effects and therefore does not allow for an apples-to-apples comparison.

For example, the display pipeline implemented in the PL is initialized during boot when the corresponding DRM driver is probed. Performing a full reconfiguration will cause the display pipeline to temporarily disappear and the monitor to go out of sync. It can easily take a couple seconds for the monitor to re-sync depending on the model.

Other peripherals connected to the PL, for example the AXI IIC are affected as well because their respective drivers are probed only once during the boot process. Some peripherals are thus in an uninitialized state and not fully operational after full reconfiguration.

Proper driver loading and un-loading needs to be done to avoid these side-effects which results in additional overhead and complexity. Therefore, full reconfiguration is beyond the scope of this application note.

For Partial Reconfiguration, the video pipeline start/stop time increases from 153ms for TPG input to 172ms for HDMI. In the latter case, the resolution is detected at the HDMI receiver and the driver is configured accordingly which causes the overhead.

The time to stop and re-start the video pipelines is significant compared to the time to reconfigure part of the PL. The PCAP configuration time only accounts for about 11% of the pipeline start/stop time if TPG is selected and is thus negligible. In this design, about 19% of the PL resources are reconfigured based on the full and partial bitstream sizes.

At a frame rate of 60fps, the time period each individual frame is displayed is a little less than 17ms (1/60fps). Putting the video start/stop time in relation to the frame rate, it takes about 10 video frames (172ms/17ms) to stop and re-start the video stream if HDMI is selected.

Building and Running the Reference Design

For detailed information and tutorials on implementing and running the reference design, visit the Zynq-7000 Partial Reconfiguration Reference Design WIKI page [Ref 10].

Software Tools and System Requirements

See "Section 1" of the Zynq-7000 Partial Reconfiguration Reference Design WIKI page [Ref 10] for the following information:

- Hardware/software requirements and licensing
- Design overview and project directory structure

Running the Pre-Built Reference Design

See "Section 2" of the Zynq-7000 Partial Reconfiguration Reference Design WIKI page [Ref 10] for the following information:

- Setup of the ZC702 hardware platform
- Execution and operation of the reference design

Building the Hardware

See "Sections 3, 4, and 5 of the Zynq-7000 Partial Reconfiguration Reference Design WIKI page [Ref 10] for the following information:

- Vivado HLS Image Filter build flow tutorial
- Vivado IP integrator system design flow tutorial
- Vivado Partial Reconfiguration design flow tutorial

Building the Software

See "Sections 6 and 7" of the Zynq-7000 Partial Reconfiguration Reference Design WIKI page [Ref 10] for the following information:

- XSDK application build flow tutorial
- Petalinux Linux image build flow tutorial

Reference Design

You can download the Reference Design Files for this application note from the Xilinx website.

The following table shows the reference design matrix.

Table 3: **Reference Design Checklist**

Parameter	Description
General	
Developer name	Xilinx
Target devices	Zynq-7000 AP SoC Device
Source code provided	Yes
Source code format	VHDL, Verilog, C (some sources encrypted)
Design uses code or IP from existing reference design, application notes, third-party or Vivado software?	Zynq Base TRD 2014.4, XAPP1159, Xilinx IP, Xylon IP
Simulation	
Functional simulation performed	Only HLS modules
Timing simulation performed	N/A
Testbench provided for functional and timing simulations	Yes
Testbench format	C
Simulator software/version used	CSim (Vivado HLS 2014.4)
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tool(s) and version	Vivado Design Suite 2014.4, Vivado HLS 2014.4
Implementation software tool(s) and version	Vivado Design Suite 2014.4
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	ZC702 evaluation board

References

1. *Partial Reconfiguration Hardware Accelerator for the Zynq-7000 AP SoC* (XAPP1159)
2. *Zynq-7000 All Programmable SoC ZC702 Base Targeted Reference Design (Vivado Design Suite 2014.4)* (UG925)
3. *Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries* (XAPP1167)
4. *Vivado Design Suite: Partial Reconfiguration User Guide* (UG909)
5. *Zynq-7000 All Programmable SoC Technical Reference Manual* (UG585)
6. *Zynq-7000 All Programmable SoC Software Developers Guide* (UG821)
7. *7 Series FPGAs Configuration User Guide* (UG470)
8. *Vivado Design Suite Tutorial: Partial Reconfiguration* (UG947)
9. *SDK User Guide: System Performance Analysis* (UG1145)

Web Sites

The following web sites contain more information:

10. Zynq-7000 Partial Reconfiguration Reference Design WIKI page
11. HDMI Input/Output FMC Module web page
12. Partial Reconfiguration web page
13. AXI Interconnect
14. AXI Video DMA
15. Compact Multilayer Video Controller
16. AXI Performance Monitor
17. Video Timing Controller
18. Test Pattern Generator
19. Video In to AXI4-Stream
20. AXI IIC Bus interface
21. Xilinx Linux Drivers WIKI page

Please Read: Important Legal Notices

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/20/2015	1.1	Renamed to enable search by Device.
02/27/2015	1.0	Initial Xilinx release.