XAPP1159 (v1.0) January 21, 2013

Partial Reconfiguration of a Hardware Accelerator on Zynq-7000 All Programmable SoC Devices

Author: 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® FPGAs 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 FPGA 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 using partial reconfiguration on Xilinx Zynq™-7000 All Programmable SoC devices through the device configuration (DevC) / processor configuration access port (PCAP) interface. The provided reference design is built on top of the ZC702 Base Targeted Reference Design (TRD) [Ref 1], an embedded video processing application that demonstrates the benefits of offloading a compute-intensive video filter algorithm from software onto Programmable Logic. The design demonstrates how to use software-controlled partial reconfiguration to dynamically reconfigure part of the logic with one of two 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 pipeline that consists of input, pre-processing, core processing, post-processing, and output stages (see Figure 1). The PS is used to configure the individual IP cores inside the PL and to control the data flow. The provided reference design offers the user to choose between a bare-metal software application and two different Linux OS-based software applications running on the PS.

The ZC702 Base TRD [Ref 1] 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 1080p60 video stream due to hardware acceleration and (2) freeing up CPU resources for user-specific tasks such as rendering a graphical user interface (GUI) on top of the processed video stream.

© Copyright 2013 Xilinx, Inc. Xilinx, the Xilinx logo, Artix, ISE, Kintex, Spartan, Virtex, Vivado, Zynq, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. All other trademarks are the property of their respective owners.

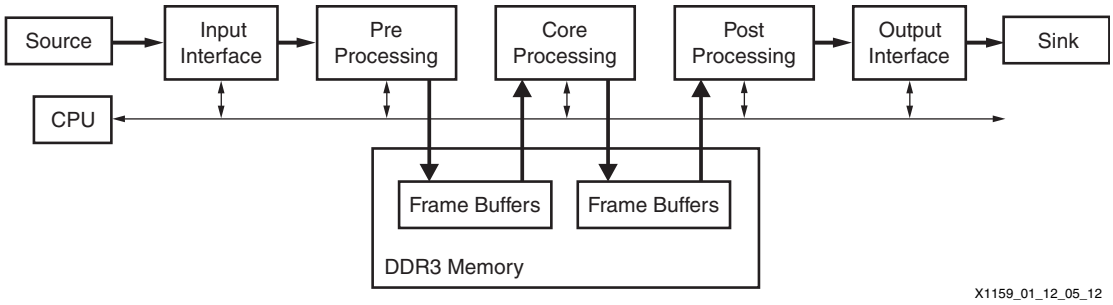

Figure 1: **Video Processing Pipeline**

Based on the ZC702 Base TRD, this application note describes the methodology of implementing two different video processing accelerators in the PL by using partial reconfiguration to load the desired functionality on demand. The first video filter IP core is a Sobel filter that detects and displays the edges in the input video stream; the second video filter IP core is a Sepia filter that applies a unique brown-tinted monochrome color to the input video stream. Figure 2 shows a comparison of original, Sobel-processed, and Sepia-processed images. The RTL for both video filter IP cores are generated from a C-algorithm description using the High-Level Synthesis tool Vivado™ HLS. For details on the Sobel filter algorithm, Vivado HLS tool flow, Linux device driver, and system integration, refer to XAPP890, Zy*nq All Programmable SoC Sobel Filter Implementation Using the Vivado HLS Tool* [Ref 2]. Similar concepts can be applied to the provided Sepia filter example.

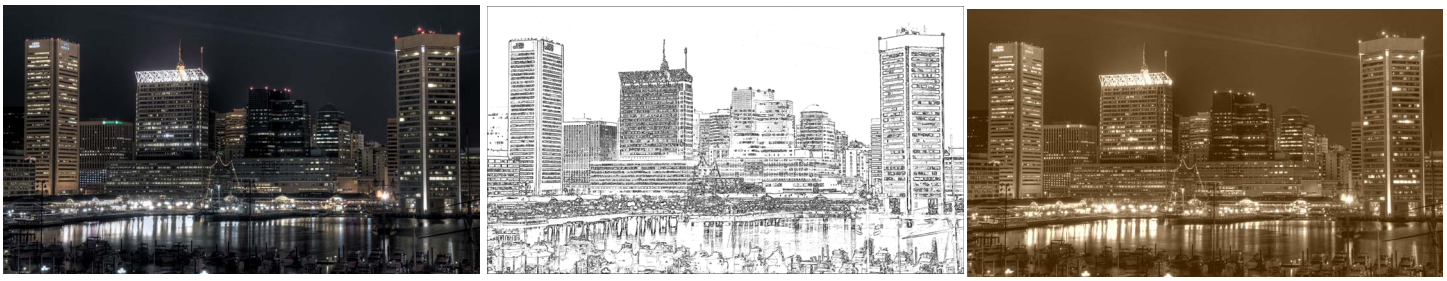

Figure 2: **Original Image (Left), Sobel-Processed Image (Center), Sepia-Processed Image (Right)**

This application note assumes that the user is familiar with the ZC702 Base TRD [Ref 1] 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
- Step-by-step partial reconfiguration design flow
- Implementing software-controlled partial reconfiguration through the PS DevC/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.

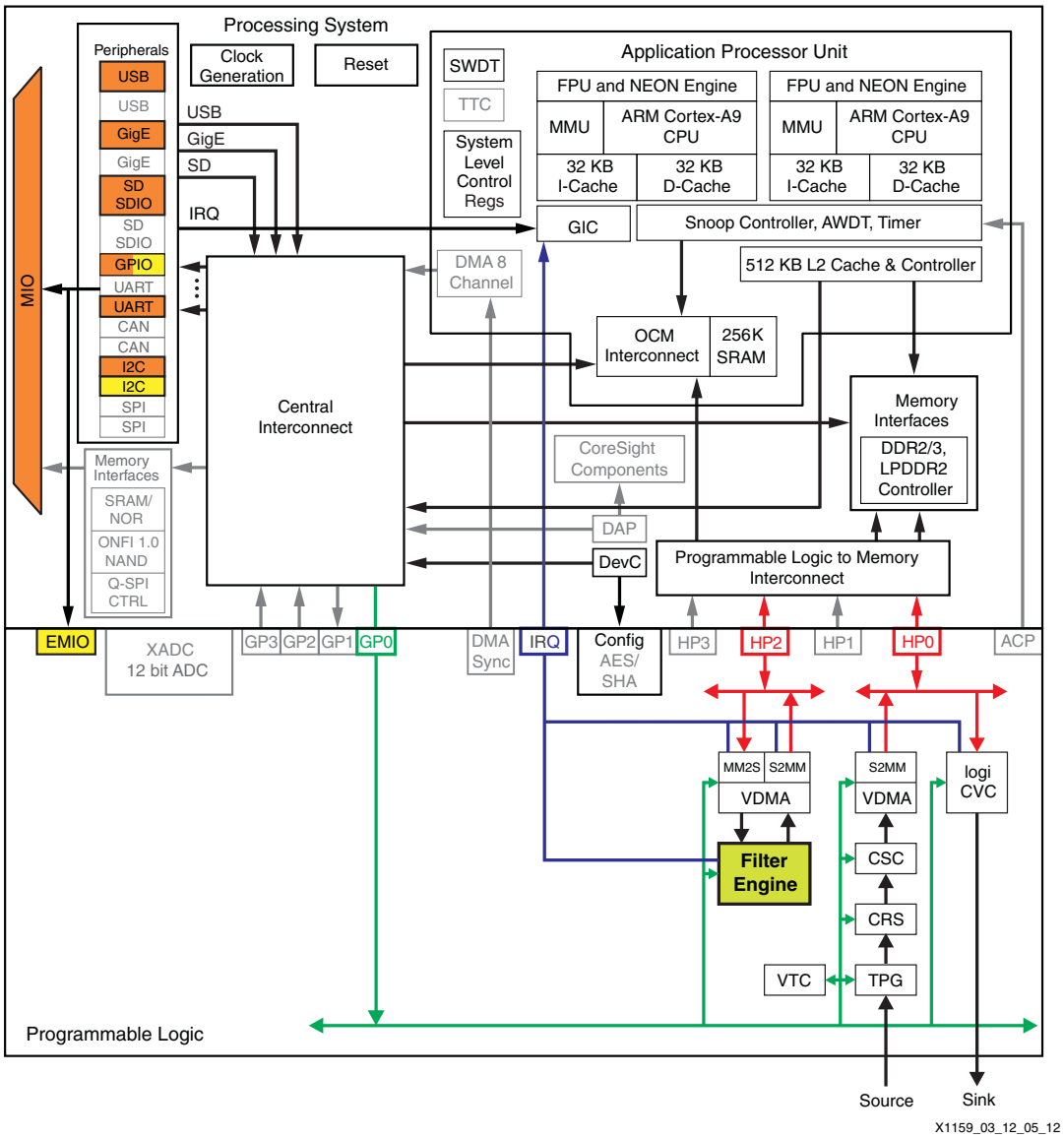

Figure 3: **System-Level Block Diagram of PR Reference Design**

Programmable Logic

The PL reference design includes the following IP cores:

- AXI Interconnect [Ref 14]
- AXI Video DMA (VDMA) [Ref 15]
- Xylon Compact Video Controller (logiCVC) [Ref 16]
- AXI Performance Monitor [Ref 17]
- Video Timing Controller (VTC) [Ref 18]
- Video Test Pattern Generator (TPG) [Ref 19]
- Chroma Resampler (CRS) [Ref 20]
- YCrCb to RGB Color-Space Converter (CSC) [Ref 21]
- Video In to AXI4-Stream [Ref 22]
- Filter Engine: Sobel or Sepia filter(1)

1. Vivado HLS generated IP cores

Referring to Figure 3, the pre-processing pipeline consists of the VTC, TPG, CRS, CSC, and VDMA. The VTC generates video timing signals required by the TPG to generate a test pattern inside the FPGA. Optionally, an external video input source can be connected through an FMC daughter card. In either case, the input video format is YCbCr 4:2:2. The CRS and CSC blocks convert the input video format to RGB 4:4:4. A VDMA writes the incoming video stream into dedicated video frame buffers inside DDR memory.

The core processing pipeline consists of the Filter Engine and a second VDMA. The core processing pipeline can be disabled to display the original video input stream. If enabled, the VDMA reads a video frame from DDR and sends it to the Filter Engine. After the video frame is processed inside the Filter Engine, it gets written back into DDR memory via the VDMA. The Filter Engine (light green box) is the block to be reconfigured in this design. We use the name Filter Engine when we are not referring to a particular implementation. The two available implementations are a Sobel and a Sepia filter. In the reference design, the Sobel filter is used as the initial filter. The Sobel or Sepia filter modules can be loaded dynamically at run-time using partial reconfiguration.

The post-processing pipeline consists of the logiCVC display controller. It has an integrated DMA engine to read frame buffers from DDR memory and a multi-layer alpha blender to alpha blend multiple video frames. Built-in color space converter and chroma resampler units convert the video data to YCbCr 4:2:2 output video format. The output video stream gets sent to the monitor display over HDMI.

Processing System

The PS is configured with the following I/O peripherals enabled (see Figure 3): USB0, Ethernet0, SD0, UART1, I2C0, I2C1, and GPIO. All I/O peripheral interfaces are configured for MIO (orange boxes), except for I2C1 and part of the GPIO which are configured for EMIO (yellow boxes). The I2C0 controller is used to configure the ADV7511 HDMI transmitter and the SI570 clock synthesizer (provides an accurate video clock) on the ZC702 board [Ref 3]. The I2C1 controller (signals are routed through PL) is used to configure the ADV7611 HDMI receiver on the Avnet FMC-IMAGEON daughter card [Ref 4] (optional). GPIO signals are routed to the PL and used to reset the TPG, reset the Filter Engine, and multiplex between TPG and external video input (from FMC). Interrupt signals are connected to three VDMA channels, the logiCVC, and the Filter Engine. The General Purpose Master Port GP0 is used to configure and control memory-mapped IP cores via AXI4-lite. The HP0 and HP2 High Performance Ports are connected via AXI4: the core processing pipeline is connected to the HP2 read/write channels; the input/pre-processing pipeline is connected to the HP0 write channel; the output/post-processing pipeline is connected to the HP0 read channel. A PS internal clock generator provides a 100 MHz clock to the PL (not drawn in figure) which in turn generates a 150 MHz clock (data path) and a 75 MHz clock (control path). Unused I/O peripherals or PL interfaces are grayed out.

Hardware Design

Refer to UG925, *Zynq-7000 All Programmable SoC ZC702 Base Targeted Reference Design User Guide* [Ref 1] 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. In this reference design, the Filter Engine core was identified as a good candidate for partial reconfiguration (PR) for the reasons described below.

Terminology

The following terminology will be used: reconfigurable partition (RP) refers to the physical location on the FPGA selected for partial reconfiguration; the remainder of the design is referred to as static logic. Filter Engine is a generic socket that is defined by its hardware interfaces and ports and is mapped onto the RP; a specific filter implementation can be plugged into this socket and is referred to as a reconfigurable module (RM). In this design, two RMs are provided, a Sobel and a Sepia filter. An RM in conjunction with the static logic is called a configuration; the two configurations in this design are called Sobel and Sepia configuration. A configuration describes a complete FPGA design and produces a full bitstream for an RM plus static logic and a partial bitstream for the RM.

Filter Operation

To better understand the implications of our design choices, we first look at the operation of the Filter Engine IP core. The core has an AXI4-Lite interface that is used by the PS to configure and control the operation of the core. First, the IP core needs to be configured with the video dimensions (width and height) of the present video stream. Next, the core is started and after it has processed an entire video frame, an interrupt is issued. The interrupt handler identifies the interrupt source and then re-starts the core so it can process the next video frame. In this fashion the video core will continuously process an incoming video stream on a frame-by-frame basis. The user can stop the core by setting a flag that tells the interrupt handler not to re-start the core after the current frame has been processed. In addition, the core can be reset by driving the corresponding PS-GPIO signal connected to the core's reset line. In summary, the following functions are provided to operate the core: configure, start, stop, and reset.

Hardware Interface

The Filter Engine IP core has the following hardware interface and connections to the static portion of the system:

- 32-bit AXI4-Lite interface connected to GP0 Master interface
- 32-bit AXI4-Stream interface connected to VDMA MM2S Master Stream interface
- 32-bit AXI4-Stream interface connected to VDMA S2MM Slave Stream interface
- Interrupt signal connected to PS-GIC
- Reset signal connected to PS-GPIO controller
- Clock signal 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 the boundary of the RP and the static portion of the design. Hence, 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 connected to outputs of Reconfigurable Modules must ignore data from Reconfigurable Modules during partial reconfiguration. The Reconfigurable Modules will 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, attention needs to be paid to the AXI interfaces and the interrupt line. It is the designer's responsibility to make sure that there are no pending transactions on the AXI interfaces at the time of the partial reconfiguration, otherwise the AXI busses can be hung. It is also recommended to disable the interrupt handler at the GIC during reconfiguration which can trigger spurious interrupts and handling such an interrupt can hang the system e.g. by accessing a hardware register inside the filter module which is not fully loaded yet. In this reference design, software is used to prevent any of the above scenarios. In some cases it might be necessary to add hardware glue logic to the RM's hardware interfaces in order to appropriately decouple those. To ensure the RM is in a defined state after reconfiguration, it is recommended to apply a reset to the filter IP core, slightly before, during, and after the reconfiguration process. After reconfiguration is finished, the reset signal can be de-asserted and the core is ready to be configured and started. Refer to UG702, *Partial Reconfiguration User Guide* [Ref 5], Chapter 7, Design Considerations for more information.

Driver Architecture, Register Interface, and Address Map

The Filter Engine IP cores implement a single generic hardware register interface and use the same memory address map. While this is not a strict requirement, it greatly simplifies the software driver architecture. In this example, the only configurable filter core parameters are the video dimensions which both RMs have in common. In a more sophisticated example, the Sobel filter could be programmed with coefficients that control the edge detection sensitivity; the Sepia filter could be programmed with parameters that control the individual color components such that a different coloring effect than sepia can be achieved. In this scenario, the corresponding driver can be split into a generic core driver that provides basic functionality that all RMs share and a filter specific driver that provides functionality unique to the various implementations. In this example, the exact same driver can be used for both Sobel and Sepia filter implementations.

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 PlanAhead™ software tool, PR enabled projects start at the netlist level.

Prerequisites

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

1. Generate the Sobel and Sepia filter IP cores and their corresponding standalone or bare-metal drivers using the Vivado HLS tool. Vivado HLS provides a methodology for migrating algorithms coded in C, C++, or System-C from the Zynq PS onto the PL by generating RTL code. The generated RTL code can be exported as an XPS pcore which can then be easily dropped into an XPS project. For a tutorial on the Sobel filter implementation flow using Vivado HLS refer to XAPP890, *Zynq All Programmable SoC Sobel Filter Implementation Using the Vivado HLS Tool* [Ref 2] - the same methodology can be applied to the Sepia filter.
2. The hardware design is delivered as a PlanAhead/XPS project based on the ZC702 Base TRD which has the Sobel filter pcore (obtained in the first step) already instantiated. The Sobel filter design is synthesized to generate the corresponding netlists and constraint files for the overall system and the individual IP cores.
3. After the first synthesis run, the Sobel filter pcore is replaced with the Sepia filter pcore and the new design is re-synthesized. From this run, we only extract the netlist and constraint file for the Sepia filter IP core. All other netlists and constraint files are taken from the previous step.

Partial Reconfiguration Design Flow

Partial reconfiguration in the PlanAhead tool is a special license-enabled feature. Visit the Xilinx partial reconfiguration web page [Ref 13] for more information.

Below is a high-level description of the individual steps of the PlanAhead PR design flow:

1. Create a PlanAhead PR project targeting the ZC702 evaluation platform and import the netlists and constraint files generated in steps 2 and 3 of the Prerequisites section, except for the Sobel and Sepia filter netlist/constraint files. This represents the static logic of the design.
2. When loading the synthesized design, the Filter Engine module will be treated as black box since there is no netlist associated with it. Define the Filter Engine module as a Reconfigurable Partition (RP). Partitions ensure that the logic and routing common to each of the multiple designs is identical.
3. Create two Reconfigurable Modules (RM) for the above created RP by adding the corresponding netlist/constraint files for the Sobel and Sepia filter cores. Constraints that only apply to specific RMs must be scoped to the module level and should be provided alongside the corresponding netlist. Constraints applied to the static logic and any constraints that are shared across all RMs should be included in the top-level constraint file.
4. Floorplan the Reconfigurable Partition by setting the physical size of the partition and the types of resources desired. Xilinx FPGAs support reconfiguration of CLBs (Flip-Flops, LUTs, distributed RAM, Multiplexers, etc.), BRAM, and DSP blocks, plus all associated routing resources. The designer needs to floorplan the RP such that it can accommodate the resources required by the RMs (see Table 1). As a rule of thumb, 20% overhead should be accounted for routing resources. Where to place the RP inside the PL with respect to the static logic depends on the data flow and how the RMs 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, and then create a partition around this region that is big enough to hold all the resources. It is good practice to frame-align the RP to achieve best place and route results. 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. The RP physical region will be stored as AREA_GROUP RANGE constraints in the primary constraints file. Refer to UG702, *Partial Reconfiguration User Guide* [Ref 5] for details on how to floorplan an RP.

Table 1: **RP Resources Required for Sobel and Sepia RMs**

	RP	Sobel RM		Sepia RM	
Site Type	**Available**	**Required**	**% Utilized**	**Required**	**% Utilized**
LUT	1600	987	62	547	35
FD_LD	3200	1007	32	613	20
SLICEL	250	149	60	76	31
SLICEM	150	99	66	62	42
DSP48E1	20	2	10	5	25
RAMBFIFO18E1	20	3	15	0	0
RAMBFIFO36E1	10	0	0	0	0

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, We first implement the Sobel configuration. The PlanAhead tool ensures 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. After successful implementation, promote the Sobel configuration such that the implementation results of the static portion of the design can be re-used by the Sepia configuration.

6. Implement the Sepia configuration. Make sure to import the static logic from the Sobel configuration generated in the previous step. Figure 4 shows a PlanAhead display comparing the physical resources inside the RP for the routed Sobel and Sepia configurations.

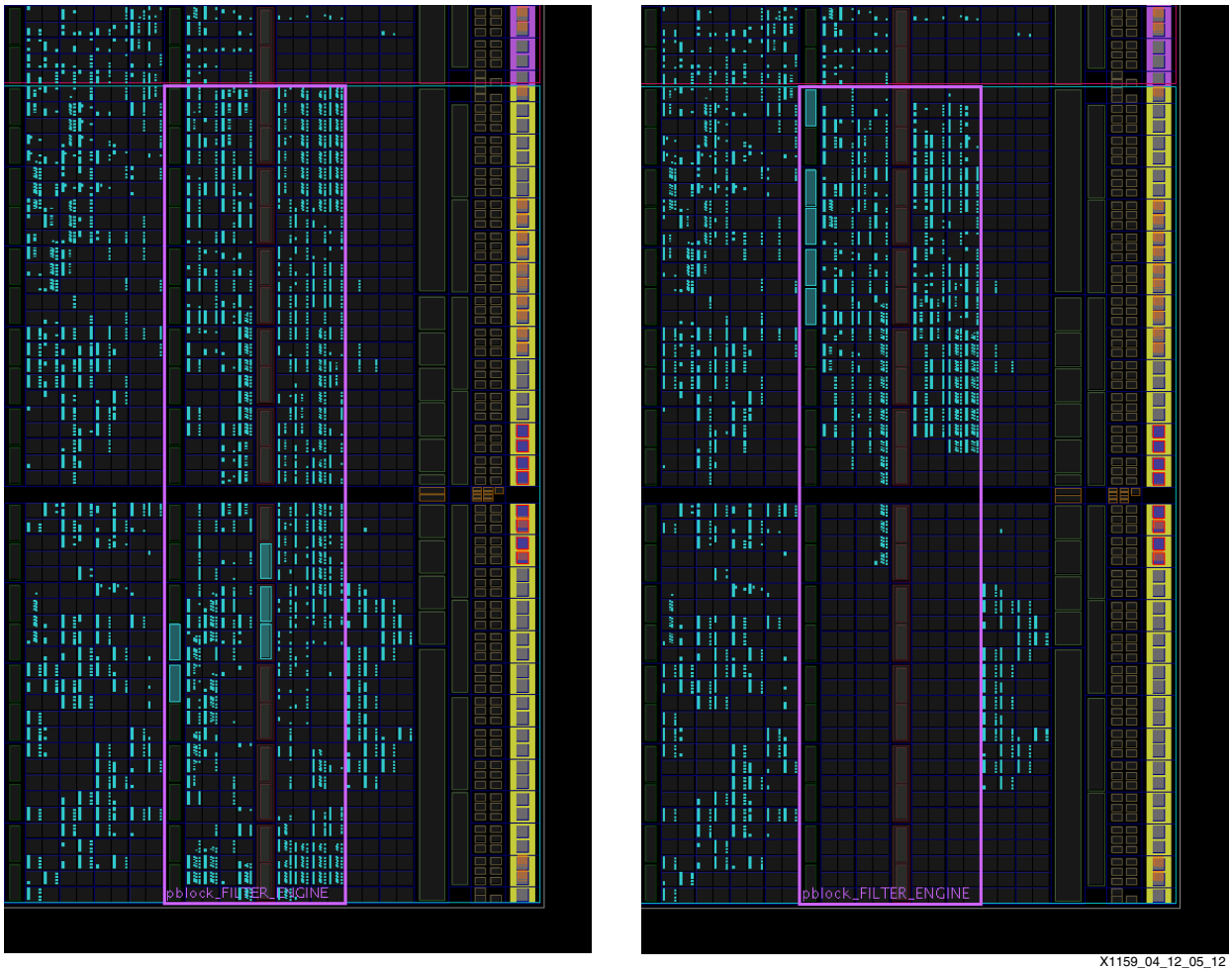

Figure 4: **PlanAhead Display - Placed RP Resources for Sobel Filter (Left) and Sepia Filter (Right)**

7. Run the PR Verify Configuration Utility to validate consistency between the implementations of the Sobel and Sepia configurations.
8. Generate full and partial bitstreams for the Sobel and Sepia configurations. We will use the full bitstream of the Sobel configuration as default start-up configuration when booting the Zynq device.
9. Convert the Sobel and Sepia partial bitstreams to binary format using the PROMGen tool. PROMGen also reports the size of the generated partial binary files which is required when sending the configuration data to the PL via the DevC's DMA engine.

Figure 5 illustrates the entire design flow in the context of this reference design. The wide dark gray boxes show the Sobel configuration, which is used in the first synthesis and implementation pass. For the second pass, the Sepia configuration is used (light gray boxes). In the end, four bitstreams are generated, of which three (solid boxes) are used in the final application; the full Sepia bitstream (dashed box) is discarded. Detailed tutorials for each of the individual steps are provided on the Zynq PR reference design wiki page [Ref 6]. Refer to

UG702, *Partial Reconfiguration User Guide* [Ref 5] for details on the PR design flow and design considerations.

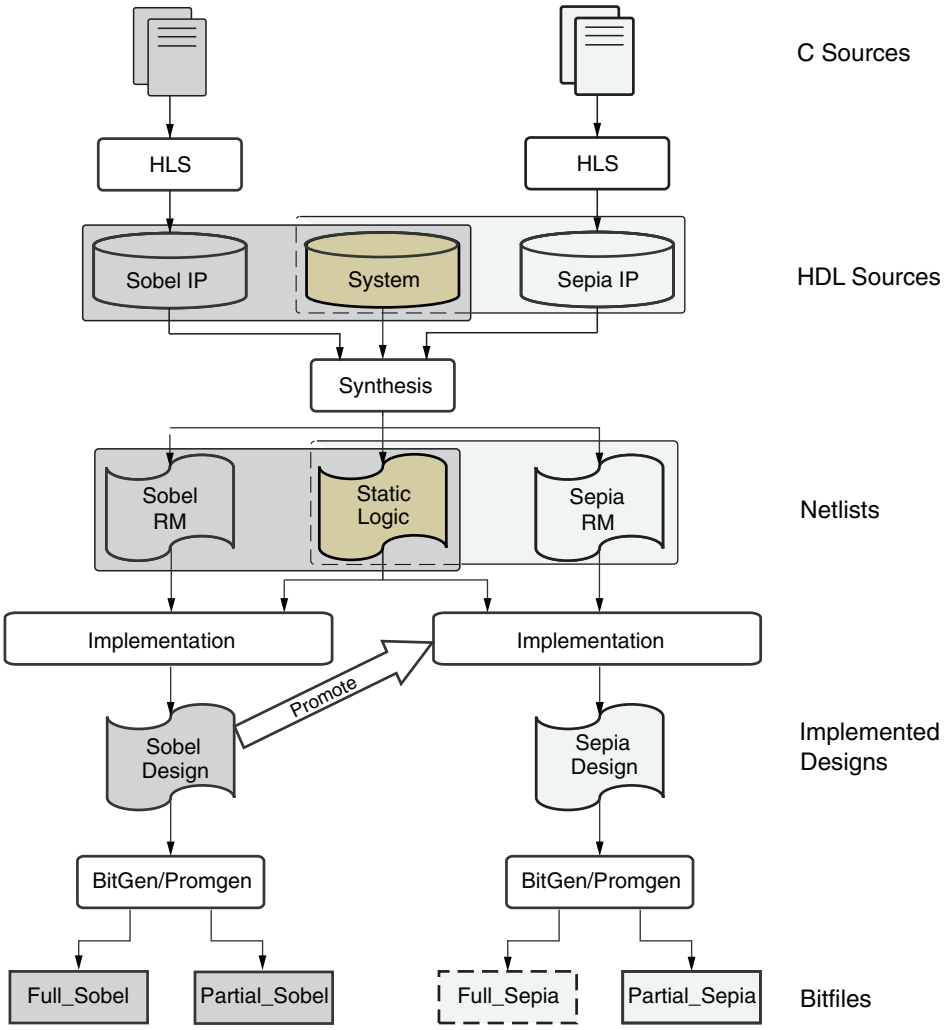

Figure 5: **Full System Design Flow for PR Reference Design**

Device Configuration

The Device Configuration interface (DevC), illustrated in Figure 6, has three main blocks: an AXI-PCAP bridge (center) for interfacing the PL configuration logic, device security management (left), and an XADC interface (right). For this application note, only the AXI-PCAP bridge is of interest.

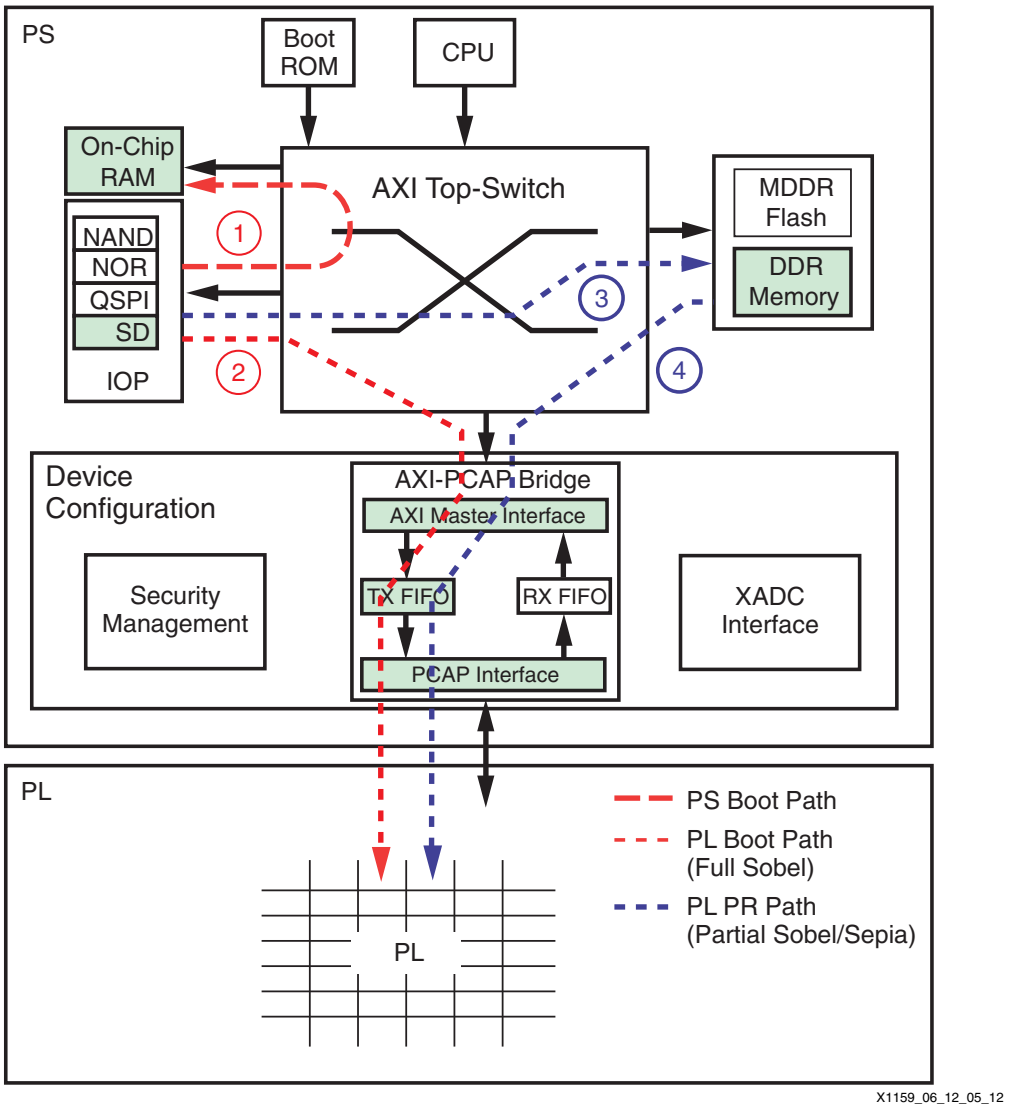

Figure 6: **Device Configuration Flow (Boot and Partial Reconfiguration)**

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 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.

Device Configuration and Boot Flow

The device configuration and partial reconfiguration flow is illustrated in Figure 6. 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 DevC's DMA 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 via the Processor Configuration Access Port (PCAP). The device is now fully configured and operational.
   a. Standalone: The FSBL loads and releases control to the standalone user application.
   b. Linux: 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 is started automatically.
3. 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.
4. At this point, the application can use the partial bitstreams at any time to modify the pre-defined PL regions while the rest of the FPGA remains fully active and uninterrupted. This is done by transferring either the partial Sobel or the partial Sepia bitstream from DDR to the PL via 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, since configuration frame addressing information is included in the partial bitstream. For more information on boot and configuration, refer to UG585, *Zynq-7000 All Programmable SoC Technical Reference Manual* [Ref 7], Chapter 6, Boot and Configuration, and UG821, *Zynq-7000 All Programmable SoC Software Developers Guide* [Ref 8], Chapter 3. Information on configuration in general can be found in UG470, *7 Series FPGAs Configuration User Guide* [Ref 9].

Software Application

The included reference design provides three software applications: a standalone or bare-metal software application, a command line based Linux application, and a Qt GUI based Linux application. All three software applications provide the same functionality and offer the user to switch between the following operating modes: original unprocessed video stream (core processing pipeline is bypassed), Sobel-processed video stream, and Sepia-processed video stream. In addition, the user can select between the PL internal test pattern generator (TPG) and an external video source. Note that the external video source requires the FMC-IMAGEON daughter card [Ref 4] in addition to the ZC702 evaluation board [Ref 3]. The Linux software application is based on the ZC702 Base TRD software. Refer to UG925, *Zynq-7000 All Programmable SoC ZC702 Base Targeted Reference Design User Guide* [Ref 1] for details on the software architecture. This application note focuses on the enhancements made to enable partial reconfiguration of the Filter Engine under software control.

Software Control Flow

The provided software applications use a fixed video resolution, therefore the following video timing sensitive peripherals need to be configured only once during program execution: VTC, logiCVC, and SI570 clock synthesizer. In the Linux application(s), the clock synthesizer is configured by the logiCVC frame buffer driver whereas in the standalone application it is a separate step. The ADV7511 HDMI transmitter and the ADV7611 HDMI receiver (if an FMC is present) are initialized on start-up in the standalone application; when using one of the Linux applications, the initialization is done inside the FSBL.

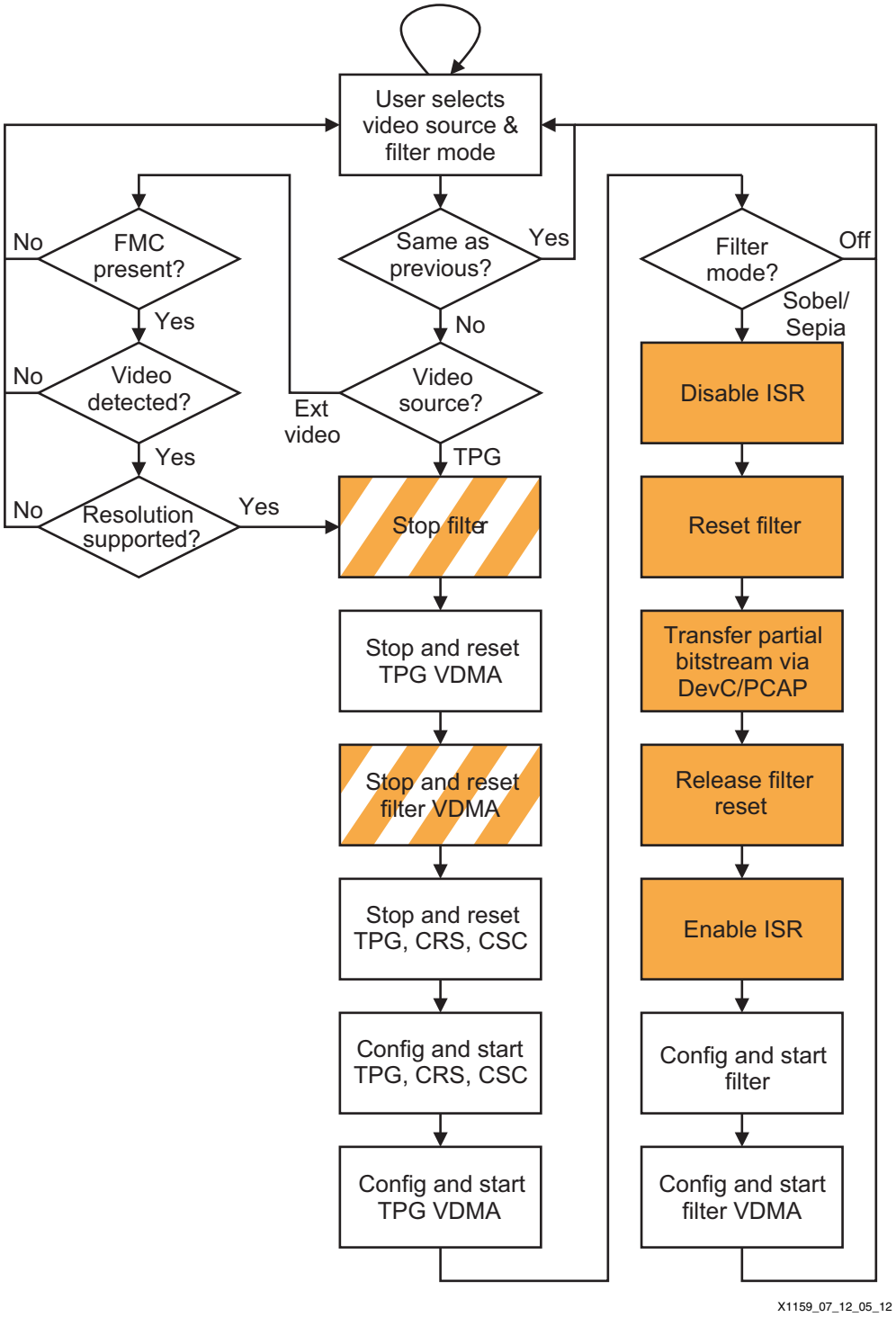

Figure 7: **Software Control Flow**

Once the start-up initialization is finished, the software application waits for user input either through the command line or a graphical user interface. Figure 7 shows the software control flow of the standalone application from there on. A similar flow is maintained in the Linux application. The orange filled boxes are specific to the partial reconfiguration flow and can be skipped otherwise. The two orange striped boxes are not exclusive to the PR control flow but are critical for making sure that there are no pending transactions on the AXI4-Streaming interfaces before initiating the partial reconfiguration. If software cannot be used to guarantee this, hardware glue logic might be necessary to decouple the AXI bus interfaces at the partition boundary.

Before transferring a partial bitstream across the PCAP interface, the user has to make sure that the interrupt handler is disabled at the interrupt controller, otherwise the reconfiguration can trigger a spurious interrupt and the ISR in turn try to access hardware inside the RM that is not loaded yet. For the filter to come up in a known good state, it is recommended to assert its reset before, during and shortly after the partial reconfiguration. To initialize the DMA transfer via PCAP, the address or buffer where the partial bitstream is stored in DDR memory and the size of the bitstream (obtained from the PROMgen tool) need to be passed to the corresponding driver function. When the bitstream transfer is finished, the reset can be released, the ISR enabled, and the filter configured and started.

Filter Engine Driver

In the Linux case, the filter start and stop functions will automatically enable or disable the ISR. The ISR is part of the driver and executes in kernel mode. In the standalone case, the ISR enable/disable functions are called separately and the ISR resides in the user application instead of the driver. The filter reset function is implemented using the PS GPIO driver. The standalone filter driver is generated by the Vivado HLS tool and can be used out of the box. Application note XAPP890, *Zynq All Programmable SoC Sobel Filter Implementation Using the Vivado HLS Tool* [Ref 2] shows how to write a Linux driver based on the generated standalone driver. The Linux driver is delivered as a patch against the Linux kernel and the standalone driver is delivered inside an SDK user repository. In both cases, the same driver can be used for the Sobel and Sepia filter implementations since they use the same register interface and address map.

Device Configuration Driver

The Linux DevC device driver is built on top of sysfs, 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 needs to be set to 1. A write file operation on the DevC device node is used to transfer the partial bitstream. The DevC write driver function initiates the DMA transaction and then waits for an interrupt signaling that the transfer is completed. Partial reconfiguration can also be initiated outside a user application from a shell, for example:

```
% echo 1 > /sys/devices/amba.0/f8007000.devcfg/is_partial_bitstream
% cat /mnt/sobel.bin > /dev/xdevcfg
```

Note that in most cases, a user application will be required to execute pre and post partial reconfiguration steps to make sure that the system is in a defined state. In the standalone case, a simple function call is used to transfer the partial bitstream. Polling is used to determine when the DMA transaction and PCAP transfer are completed.

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), the total memory throughput of this design with the filter disabled is:

(1920 x 1080 x 4 Bytes x 60 Hz) x 2 = 497 MB/s x 2 ≈ 1 GB/s

or

(1920 x 1080 x 4 Bytes x 60 Hz) x 4 = 497 MB/s x 4 ≈ 2 GB/s

with the filter enabled.

When using the GUI based Linux application, another 125 MB/s are consumed by the GUI layer. The GUI plots the measured performance numbers for the HP0 and HP2 ports to monitor the memory throughput in real-time. With the 32-bit wide DDR3 memory clocked at 533 MHz (1066 MHz data rate), the theoretical memory throughput is:

$$4 \text{ Bytes} \times 1066 \text{ MHz} = 4.264 \text{ GB/s}$$

Therefore the maximum memory utilization in this design is close to 50%:

$$(2 \text{ GB/s} + 0.125 \text{ GB/s}) / 4.264 \approx 0.5$$

Configuration Time

The 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.

Table 2 compares the full and partial bitstream sizes of this design as well as the measured configuration time in a standalone or Linux application. The configuration time is measured between the beginning and the end of the DevC DMA transfer driver function call.

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

XC7Z020 Part	Full Bitstream	Partial Bitstream
Bitstream Size	4,045,564 Bytes	134,392 Bytes
Configuration Time - Standalone	32 ms(1)	1,060 µs(1)
Configuration Time - Linux	44 ms(1)	2,000 µs(1)

Notes:

1. Measured values are average values.

In the context of video applications, real-time processing can be a strict requirement and dropping a video frame might not be acceptable in safety-critical systems. In order to determine the feasibility of partial reconfiguration in such a real-time system, we need to calculate the time between two consecutive video frames and see if the window is long enough to perform partial reconfiguration and then re-start the video pipeline.

A video frame period consists of active and non-active time intervals, also called blanking intervals. During active intervals, video pixel data is transferred; historically, CRT monitors used horizontal and vertical blanking intervals to reposition the electron beam from the end of a line to the beginning of the next line (horizontal blanking) or from the end of a frame (bottom right) to the start of a frame (top left) (vertical blanking). Figure 8 illustrates the active and blanking intervals of a video frame. Refer to UG934, *AXI4-Stream Video IP and System Design Guide* [Ref 10] for details on video timing parameters.

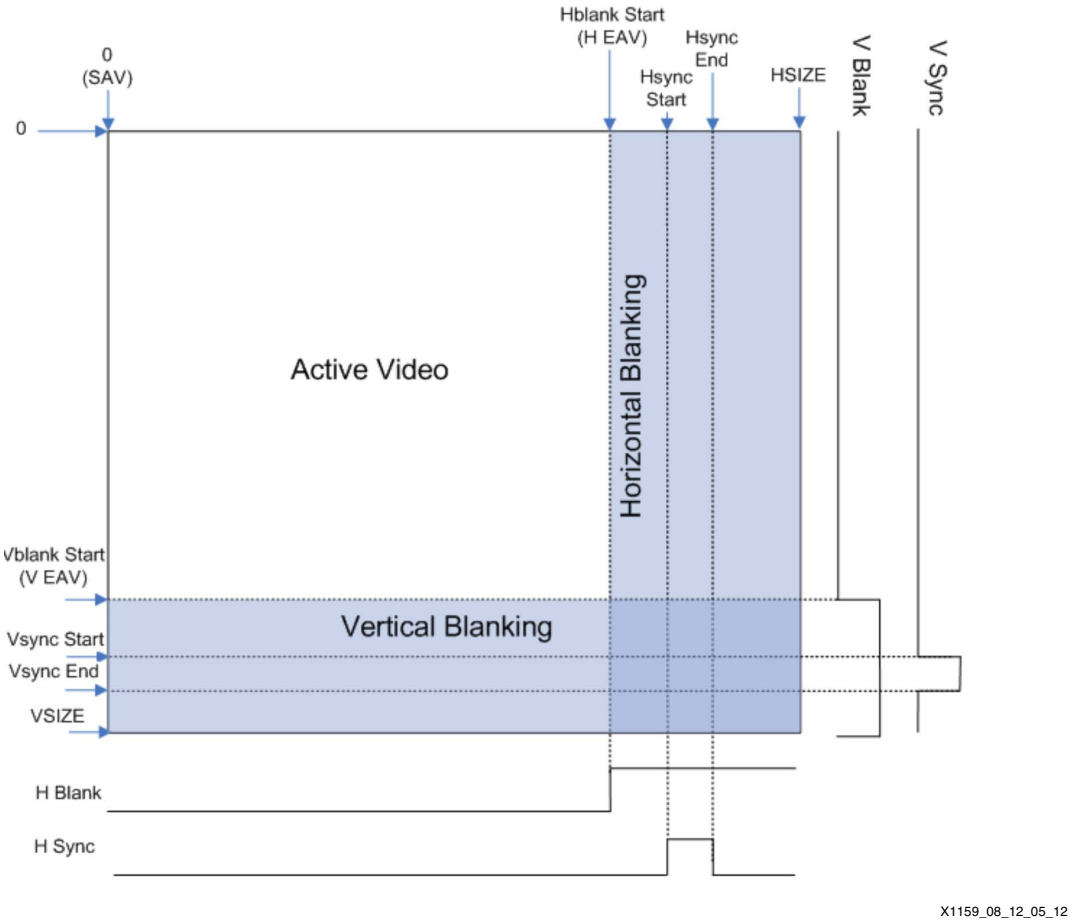

Figure 8: **Video Timing Parameters**

In a 1080p60 video stream, the vertical blanking accounts for 45 pixels per line and the horizontal blanking for 280 lines per video frame. The required video clock frequency is calculated as

$$(1920 + 280) \times (1080 + 45) \times 60\ \text{Hz} = 148.5\ \text{MHz}$$

Partial reconfiguration of a video accelerator should ideally take place between the last pixel of frame *f-1* and the first pixel of frame *f*, which corresponds to the vertical blanking period plus a single horizontal blanking period. The available time window is

$$(1 \times 45 + 280 \times (1080 + 45)\ ) / 148.5\ \text{MHz} \approx 2.1\ \text{ms}$$

The partial reconfiguration time depends on which software application and OS is used (see Table 2). For standalone, it is considerably less than the available 2.1 ms. Additional overhead is required for resetting and configuring the video pipeline IP cores when switching video modes. Note, that this reference design is not intended to meet the requirements of a real-time video application as described above. No special considerations are taken to switch video modes only during the blanking period between frames such that no frames are dropped. However, we do believe that given the 2.1ms time window and the measured timing numbers in this design, such a system is feasible in combination with a low-latency bare-metal or real-time operating system (RTOS).

Implementing 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 website:

http://wiki.xilinx.com/zynq-pr-rd

Running the Pre-Built Reference Design

Please refer to Section 7 on the Zynq-7000 Partial Reconfiguration Reference Design wiki website [Ref 6] for the following information:

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

Software Tools and System Requirements

Please refer to Section 1 on the Zynq-7000 Partial Reconfiguration Reference Design wiki website [Ref 6] for the following information:

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

Building the Hardware

Please refer to Sections 2, 3 and 4 on the Zynq-7000 Partial Reconfiguration Reference Design wiki website [Ref 6] for the following information:

- Vivado HLS design flow tutorial to generate the Sobel/Sepia filter IP cores
- PlanAhead/XPS system design flow tutorial
- PlanAhead Partial Reconfiguration design flow tutorial

Building the Software

Please refer to Sections 5 and 6 on the Zynq-7000 Partial Reconfiguration Reference Design wiki website [Ref 6] for the following information:

- U-boot boot loader and Linux kernel compile flow tutorial
- SDK tutorial for FSBL and standalone/Linux software applications compile flow and standalone/Linux boot image creation

Reference Design

The reference design can be downloaded from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=199619

Table 3 shows the reference design checklist.

Table 3: **Reference Design Checklist**

Parameter	Description
General	
Developer name	Xilinx
Target devices (stepping level, ES, production, speed grades)	Zynq-7000 AP SoC (ES)
Source code provided	Yes
Source code format	VHDL, Verilog, C (some sources encrypted)
Design uses code/IP from existing Xilinx application notes/reference designs, CORE Generator tool, or 3rd-party companies	Yes
Simulation	
Functional simulation performed	N/A
Timing simulation performed	N/A
Testbench used for functional and timing simulations	N/A
Testbench format	N/A
Simulator software/version used	N/A
SPICE/IBIS simulations	N/A
Implementation	
Synthesis software tools/version used	Vivado HLS 2012.4, XST 14.4
Implementation software tools/versions used	ISE Design Suite 14.4 System Edition
Static timing analysis performed	Yes
Hardware Verification	
Hardware verified	Yes
Hardware platform used for verification	ZC702 evaluation board

Table 4 shows the device utilization for the Sobel configuration, which is the larger design of the two. The figures for the Sepia configuration are slightly lower.

Table 4: **Device Utilization**

Item	Value
Device	XC7Z020
Device speed	-2
Grade package	CLG484
Slice registers	24,190 (22%)
Occupied slices	9,049 (68%)
Slice LUTs	20,185 (37%)
I/Os	42 (21%)
RAMB36E1s	21 (15%)

Table 4: **Device Utilization**

Item	Value
RAMB18E1s	15 (5%)
DSP48E1s	18 (8%)

References

These documents provide supplemental material useful with this application note.

1. UG925, *Zynq-7000 All Programmable SoC ZC702 Base Targeted Reference Design (ISE Design Suite 14.4) User Guide*
2. XAPP890, *Zynq All Programmable SoC Sobel Filter Implementation Using the Vivado HLS Tool*
3. Xilinx Zynq-7000 SoC ZC702 Evaluation Kit web page:
   http://www.xilinx.com/products/boards-and-kits/EK-Z7-ZC702-G.htm
4. HDMI Input/Output FMC Module web page:
   http://www.em.avnet.com/en-us/design/drc/Pages/HDMI-Input-Output-FMC-module.aspx
5. UG702, *Partial Reconfiguration User Guide*
6. Zynq-7000 Partial Reconfiguration Reference Design:
   http://wiki.xilinx.com/zynq-pr-rd
7. UG585, *Zynq-7000 All Programmable SoC Technical Reference Manual*
8. UG821, *Zynq-7000 All Programmable SoC Software Developers Guide*
9. UG470, *7 Series FPGAs Configuration User Guide*
10. UG934, *AXI4-Stream Video IP and System Design Guide*
11. UG744, *Partial Reconfiguration of a Processor Peripheral Tutorial*
12. Zynq-7000 Base Targeted Reference Design 14.4:
    http://wiki.xilinx.com/zynq-base-trd-14-4
13. Partial Reconfiguration web page:
    http://www.xilinx.com/tools/partial-reconfiguration

These web pages contain information for the IP cores used in the reference design:

14. AXI Interconnect
    http://www.xilinx.com/products/intellectual-property/axi_interconnect.htm
15. AXI Video DMA (VDMA)
    http://www.xilinx.com/products/intellectual-property/axi_video_dma.htm
16. Xylon Compact Video Controller (logiCVC)
    http://www.xilinx.com/products/intellectual-property/logiCVC.htm
17. AXI Performance Monitor
    http://www.xilinx.com/products/intellectual-property/axi_perf_mon.htm
18. Video Timing Controller (VTC)
    http://www.xilinx.com/products/intellectual-property/EF-DI-VID-TIMING.htm
19. Test Pattern Generator (TPG)
    http://www.xilinx.com/products/intellectual-property/tpg.htm
20. Chroma Resampler (CRS)
    http://www.xilinx.com/products/intellectual-property/EF-DI-CHROM-RESAMP.htm

21. YCrCb to RGB Color-Space Converter (CSC)
http://www.xilinx.com/products/intellectual-property/YCrCb_to_RGB.htm
22. Video In to AXI4-Stream
http://www.xilinx.com/products/intellectual-property/video_in_to_axi4_stream.htm

Revision History

The following table shows the revision history for this document.

Date	Version	Description of Revisions
01/21/13	1.0	Initial Xilinx release.

Notice of Disclaimer

The information disclosed to you hereunder (the “Materials”) is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of the Limited Warranties which can be viewed at http://www.xilinx.com/warranty.htm; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in Critical Applications: http://www.xilinx.com/warranty.htm#critapps.