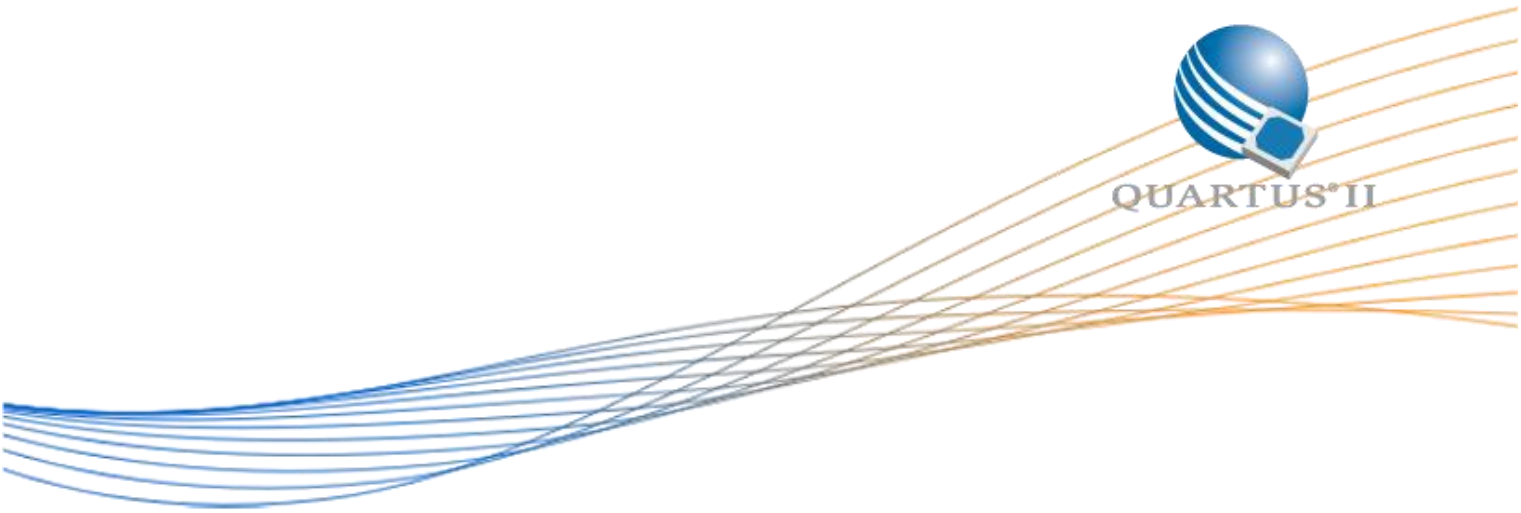


Video and Image Processing Design Example

Date: December, 2016

Revision: 1.2

Contents

Introduction 4
1 Installing the Example Design 7
2 System Requirements 9
2.1 Hardware Requirements 9
2.2 Software Requirements 9
3 Example Design Overview 10
3.1 Qsys 11
3.2 Quartus II software 12
3.3 Nios II Software Build Tools for Eclipse 12
3.4 System Console 13
4 Functional Description 14
4.1 Video Input 15
4.2 Video Processing 16
4.2.1 Composite Video Processing 16
4.2.2 DVI Video Processing 19
4.3 Video mixing 20
4.4 Video Output 20
4.5 Graphic Overlay (Radar) and Scan Layer 22
4.6 Interval Timer 23
4.7 Vectored Interrupt Controller 23
4.8 DDR3 SDRAM Controller and External DDR3 SDRAM 23
4.9 I2C Bus Masters 23
4.10 Debugging System 23
4.10.1 Trace System 24
4.10.2 Monitor 25
4.11 Nios II Processor and On-Chip Memory for Program Code 26
5 Running the Example Design 29
5.1 Opening the Quartus II Top-Level Project 29
5.2 Opening the Qsys System 29
5.3 Viewing the Parameters 30

5.4 Compiling the Design 30
5.5 Set up the Cyclone V Video Development Platform 30
5.6 Configure the Cyclone V Device 32
5.7 Download and Run the Nios II Control Program 33
6 Building, examining and debugging the Software in the Nios II SBT for Eclipse 34
6.1 Examining and Recompiling the Control Code 34
6.2 Building the Software in the Nios II SBT for Eclipse 34
6.3 Running the Nios II Control Program 34
6.4 Debugging the Application Source Code 35
6.5 Setting Up System Console for Debug Visualizations 36
6.6 Avalon-ST Video Monitor Capture Rate Value 40
7 Applications for the Example Design 41
7.1 Single Video Stream Input 41
7.2 Multiple Video Channel Input 42
7.3 Other Video Interface Standards 43
8 Conclusion 44
9 Revision History 45

Introduction

The Altera® Video and Image Processing Design Example demonstrates the following items:

- A framework for rapid development of video and image processing systems.
- Dynamic scaling, clipping, flashing, moving, sharpening and FIR filtering of both standard definition (SD) and high definition (HD) inputs.
- Picture-in-picture mixing with a background layer.
- Run-time control of different parts of the system, including a radar that uses on-screen display functions.
- Debugging components for monitoring the video stream from within the data path.

The design example runs from the Bitec HSMC DVI and Bitec HSMC Quad Video daughter cards. The standard definition video stream is in either National Television System Committee (NTSC) or Phase Alternation Line (PAL) format. The high-definition video stream uses a digital video interface (DVI) port.

The design example uses some of the parametrizable IP cores that are available in the Video and Image Processing Suite. Table 1 lists these IP cores.

➢ For more information about these IP cores and about the content of the Video and Image Processing Suite, refer to the *Video and Image Processing Suite MegaCore Functions*

Table 1 - Video and Image Processing Suite IP Cores Used in this Design

IP Core	Description
2D FIR Filter	Implements a 3x3, 5x5, or 7x7 finite impulse response (FIR) filter on a video data stream to smooth or sharpen images.
Chroma Resampler II	Changes the sampling rate of the chroma data for YCbCr image frames, for example from 4:2:2 to 4:4:4 or 4:2:2 to 4:2:0.
Clipper II	Clips video streams and can be configured at compile time or at run-time
Clocked Video Input II & Clocked Video Output II	The Clocked Video Interface IP cores convert clocked video formats (such as BT656, BT1120, and DVI) to Avalon-ST Video; and vice versa.
Color Plane Sequencer II	Changes how color plane samples are transmitted across the Avalon-ST interface. This function can be used to split and join video streams, giving control over the routing of color plane samples.
Color Space Converter II	Converts image data between a variety of different color spaces such as computer RGB or standard definition YCbCr (BT.601)
Deinterlacer II	Converts interlaced video formats to progressive video format. Multiple algorithms of varying complexities can be selected at compile-time to choose the appropriate trade-off between quality

	and hardware cost and meet the design objectives.
Frame Buffer II	Buffers video frames into external RAM. This core supports double or triple buffering with a range of options for frame dropping and repeating. This core can also be configured as a read-only/write-only module to stream data to and from memory which allows for external graphics generation or to offload image analytics.
Gamma Corrector II	Allows video streams color samples to be corrected for the physical properties of display devices.
Mixer II	Supports picture-in-picture mixing with optional alpha-blending. The alpha blending factor and the layers' position and ordering can be changed at run-time.
Scaler II	Changes the dimensions of a video stream. The scaling algorithm (nearest neighbor, bilinear, polyphase,…) is selected at compile-time. Output dimensions may be dynamically changed at run-time and the scaling coefficients for the most complex algorithms (polyphase scaling, edge-adaptive polyphase scaling) can also be reprogrammed at run-time for optimal quality.
Avalon-ST Video Monitor	This block is used for debug purposes. It captures video data packets without adding additional delays and connect to trace system IP for collecting video trace data.
Trace System	This block is used for debug purposes. The trace system collects and transmits the data captured by the video monitors. It connects to the host System Console via JTAG or USB for display of debug information.

These IP cores allow you to fully integrate common video functions with video interfaces, processors, and external memory controllers.

The design example targets an Altera Cyclone® V 5CGTFD9E development board connected by high-speed mezzanine card (HSMC) interfaces to Bitec HSMC Quad Video and DVI daughter cards. The Bitec daughter cards are available with the Cyclone V Video Processing Development Kit.

- For more information on Bitec HSMC Quad Video daughter card, refer to the *Bitec HSMC Quad Video Daughter Card*.
- For more information on Bitec DVI daughter card, refer to the *Bitec HSMC DVI Digital Audio/Video Daughter Card*.

➢ For information about the 5CGTFD9E5 development board, refer to the *Cyclone V Development Board Reference Manual*.

The design example accepts both composite and DVI video sources. The composite input is through an analog composite port on a Bitec HSMC Quad Video daughter card, which generates a digital output in BT656 format. The DVI input is through the RX port on a Bitec HSMC DVI daughter card.

The design example performs the following common video functions on the two input streams in the FPGA:

- Clipping
- Chroma resampling
- Motion adaptive deinterlacing
- Color space conversion
- Picture-in-picture mixing
- Gamma correction
- FIR filtering
- Polyphase scaling

The design example uses software on a Nios® II processor to configure and initialize the input and output video interfaces on the two HSMC daughter cards (TVP5154 and TFP410 chips). The Nios II software demonstrates how to control the inputs, outputs, and video processing functions at run-time.

The design example uses the Qsys system-level design tool to implement the video system. Qsys provides an easy path to system integration of the video processing data path. Qsys is a system development tool that allows you to create hardware and software system modules with a customized set of system peripherals. Qsys automatically creates the bus arbitration logic connecting the individual components together to create an overall system.

➢ For more information about Qsys, refer to the *System Design with Qsys* section in volume 1 of the *Quartus II Handbook*.

The Video and Image Processing Suite IP cores have common open Avalon-ST data interfaces and Avalon Memory-Mapped (Avalon-MM) control interfaces to ease connection of a chain of video functions and video system modeling. In addition, the IP cores use the Avalon-ST Video protocol to transmit video data between them. This helps building run-time controllable systems that recover seamlessly when the video input(s) or output(s) are changing (eg, cable connection/disconnection, output display renegotiating the output format, …) or from transmission errors.

➢ For a full description of how the Avalon-ST Video protocol and Avalon interfaces are implemented, refer to the *Interfaces* chapter in the *Video and Image Processing Suite User Guide.*

➢ For more information about the Avalon-MM and Avalon-ST interfaces, refer to the *Avalon Interface Specifications.*

1 Installing the Example Design

You can download the design example files from the *Altera Video Processing Reference Design* web page. Table 2 shows the directory structure for the design example files after you install the project template.

- ❖ Please do not use spaces or special characters in the path name when choosing an installation directory

Table 2 – Example Design Top-Level Directories

Directory	Description
<root>	Installation directory. Contains files listed in Table 3
- **ip**	Qsys components used in the design that are not in the Video IP Suite: • alpha_source_fix: Adds an alpha channel to a video stream. The alpha value is static over the whole frame • altera_avalon_i2c: I2C controller to communicate with the two Bitec daughter cards.
- **master_image**	Contains prebuilt **.sof** and **.elf** files to program the board without compilation
- **software**	The folder contains an archive for two Eclipse projects (the controller and the board support package) that can be imported in your local workspace to build the code and program the Nios II soft-processor used in the design. The controller project provides a C++ software application program interface for each Megacore function with an Avalon-MM slave control interface.
- **top**	The top level design files

Table 3 shows the files in the top-level directory:

Table 3 - Top-level files

File Name	Description
altpll_c5.qsys	Top-level PLL qsys parameterization file.
post_analysis_script.tcl	Compilation script to add assignments generated by the uniphy controller IP before the fitting stage.
top.jdi	JTAG debugging interface file. This file is necessary to locate the debug monitors when running System Console. It is automatically generated during Quartus compilation.
top.qpf	Quartus II project file.
top.qsf	Quartus II settings file.
vip_example_sys.qsys	Qsys system file.
vip_example_sys.sopcinfo	Qsys system info file. This file is necessary to setup the Board Support Package for the Nios II software. It is automatically generated during Quartus Compilation.

The Top-level Verilog HDL design file **top\vip_top.v** instantiates and connects the following components:

- Clock source input into phase-locked loop (PLL).
- PLL megafunction to generate the Qsys system clock and DVI output pixel clocks.
- Input pins for composite input channel 1 (from the Bitec Quad Video daughter card on the HSMC interface J1).
- Input pins for DVI input channel 2 (from the Bitec DVI daughter card on the HSMC interface J2).
- Qsys system containing video processing functions.
- DDR3 output pins.
- Output pins system driving DVI output on Bitec HSMC DVI daughter card (J2).

The Quartus II settings file also contains pin assignment information for unused interfaces on the Quad Video daughter card and DVI daughter card.

2 System Requirements

This section describes the hardware and software requirements to run the video design example.

2.1 Hardware Requirements

The Video and Image Processing Design Example requires the following hardware components:

- Cyclone V Video Development Kit including:
  - Cyclone V 5CGTFD9E5 host development board.
  - Bitec HSMC Quad Video daughter card. Refer to the *Bitec HSMC Quad Video Daughter Card* for more information
  - Bitec HSMC DVI daughter card. Refer to the *Bitec HSMC DVI Digital Audio/Video Daughter Card* for more information
- Any NTSC or PAL video source with composite output (such as a DVD player or Apple iPod).
- Any video source with DVI output (optional).
- A DVI cable to connect the DVI video source input to the board (optional).
- A monitor or display with a DVI interface supporting 1024×768 resolution.
- A DVI cable to connect the DVI output to the monitor.
- Altera USB-Blaster™ cable.

Altera used the following hardware to verify the design example:

- Raspberry pie NTSC composite output.
- Raspberry pie HDMI output going through a HDMI->DVI converter.
- A LG HD monitor.

2.2 Software Requirements

Ensure that you download and install the design example **CVGT_VIP_EXAMPLE_DESIGN.par** file and install the software provided with the development kit.

- ❖ You must install the Quartus II software, version 16.1, which includes the MegaCore IP Library and the Nios II Embedded Design Suite, on your PC. For more information about the software installation, refer to the documentation provided with the *Cyclone V Video Development Kit*.

3 Example Design Overview

The design example demos a range of functions on a suitable screen. These functions include scaling, clipping, flashing, moving and sharpening of the composite input. If a DVI input source is present, the demonstration turns it on. It then clips, scales, moves and applies filters to it. Finally, the two video streams move to the corners of the screen and the design uses the Nios II software to demonstrate on-screen display with a radar view, which shows a sweeping scanner line and tracks "enemies" moving across the screen (Figure 1).

Figure 1 – Video Stream Output

The Video and Image Processing Design Example demonstrates a simple, yet highly parametrizable, design flow for rapid system development.

The design uses the following software tools:

- Qsys for abstract design capture, parametrizable IP cores, and switch fabric generation.
- The Quartus II software for pin mapping and to interface IP cores.
- The Nios II Software Build Tools for Eclipse for run-time software control and configuration.
- Qsys System Console for debugging.

The design provides the following features:

- Open interface and protocol standards to enable design reuse and connection of custom IP cores with off-the-shelf IP:

        - Data streaming interfaces and protocols for transmission of video data between IP cores in the system (Avalon-ST Video, a protocol layer over the Avalon-ST interface).
        - Control interfaces (Avalon-MM master and slave interfaces).
        - Random access to external memory (Avalon-MM master and slave interfaces).
    - System-level tools and design methodology for rapid system construction, integration, and redesign. The Qsys tool uses standard interfaces to present an abstract view of the design and generates an application-specific switch fabric to construct the system.
    - Parametrizable IP cores that enable you to quickly construct complete video systems.
    - System debugging functionality allows you to tap and view signals in the data path. You can decode them into accessible visual representations in real time, allowing for rapid debugging while creating designs.
    - Reference designs that demonstrate the capabilities of the video and image processing IP cores.
    - Development kits to rapidly prototype the designs.

3.1 Qsys

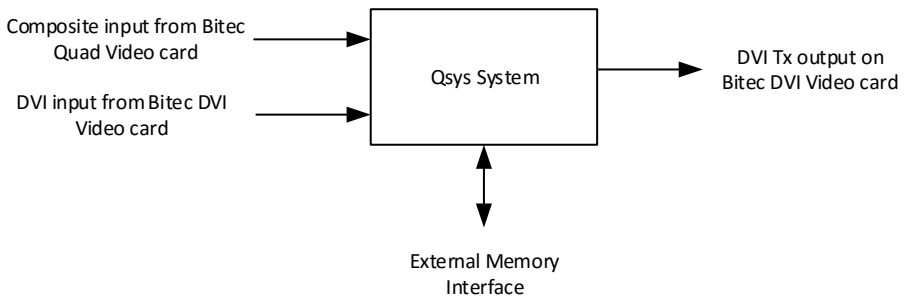
The Qsys flow is the primary design flow for rapid video system development. Specifically, Qsys simplifies the process of system design, including the data path, processor core, and external memory integration. Qsys enables you to capture the design at an abstract level, with single point-to-point data connections rather than connecting individual data and control interface wires.

All connections in the Qsys system use Avalon-ST and Avalon-MM interfaces. You can connect Video and Image Processing Suite IP cores that support Avalon-ST Video protocol for data transmission with the click of a button.

Qsys automatically generates an interconnect switch fabric, including arbitration logic to connect the memory mapped masters and slaves together. A common example is a system that uses a single memory controller but contains multiple Avalon-MM masters that buffer video data in an external memory.

Figure 2 shows the Qsys system main external connections.

Figure 2 - External Connections to Qsys System



- Video Input from an External Video Interface:
  The design uses parametrizable Clocked Video Input IP cores to connect to the external video interface. This IP core provides a bridge between a clocked video interface, such as a serial digital interface (SDI) IP core, and the Avalon-ST Video flow controlled domain.
- Video Output to an External Video Interface:
  The design uses a parametrizable Clocked Video Output IP core to connect to the external video interface. This IP core provides a bridge between the Avalon-ST Video flow controlled domain and a clocked video interface (such as DVI).
- Connection to an External Memory Interface:
  The design uses an Altera DDR3 SDRAM Controller IP core to connect to the external memory interface. Qsys generates the application-specific switch fabric to arbitrate between multiple masters trying to access the controller.

  ➢ For information about Qsys, refer to *Creating a System with Qsys* chapter in volume 1 of the *Quartus II Handbook*.

3.2 Quartus II software

The Quartus II software environment describes the top-level system and integrates the Qsys system into the top-level design.

The Quartus II software environment maps the Qsys system external connections as exported wires to video interface IP cores (such as SDI) and memory interfaces (such as DDR3). It also makes the appropriate pin assignments. The Quartus II software includes a wide range of tools to help with timing closure and perform hardware compilation to generate an FPGA programming file.

In addition, the Quartus II software provides the SignalTap™ II Logic Analyzer, a system-level debugging tool that captures and displays real-time signal behavior allowing you to observe interactions between hardware and software in system designs.

➢ For more information about the Quartus II software, you may refer to the *Quartus II Help.*

3.3 Nios II Software Build Tools for Eclipse

The Nios II Software Build Tools (SBT) for Eclipse is the primary software development tool for the Nios II family of embedded processors. You can perform all software development tasks within the Nios II SBT for Eclipse, including editing, building, and debugging programs. The Nios II SBT for Eclipse provides a consistent development platform that supports all Nios II processor systems.

You can configure video interfaces and control video processing functions in the Nios II SBT for Eclipse software. These features provide a very rapid development cycle for software control code changes, without requiring hardware recompilation. This environment provides you with all the

standard software debug tools, including breakpoints, views of memory, variables, and registers, and single stepping.

C++ software classes provide a software application programming interface (API) between the Nios II control code and the Video and Image Processing Suite IP cores. The C++ classes contain member functions to provide easy control of the IP cores and easy access to useful data flow information such as the number of frames that the design drops or repeats by a frame buffer in the data path.

- For information about the Nios II SBT for Eclipse software, refer to the *Nios II Software Developer's Handbook*.

3.4 System Console

System Console is the Altera flexible system-level debugging tool that helps you quickly and efficiently debug your design while the design is running at full speed in an FPGA. System Console enables you to send read and write system-level transactions into your Qsys system to help isolate and identify problems. It also provides a quick and easy way to check system clocks and monitor reset states, which can be particularly helpful during board bring-up. In addition, System Console allows you use graphical elements, such as buttons, dials, and graphs, to create your own custom verification or demonstration to represent many system-level transactions and monitor the processing of data.

If you connect System Console to a design that contains a Trace System IP core, such as this design example, the **Trace Table View** option appears under the Tools menu. This option allows you to view the contents of the monitors connected to the trace system.

The trace system allows you to view high-level information about the signals passing between video and image processing IP cores, allowing for rapid debugging of problems. You can tap and capture video data passing through by placing monitors into the system. System Console encapsulates and transfers the captured data through either a JTAG or a USB interface to an attached host machine.

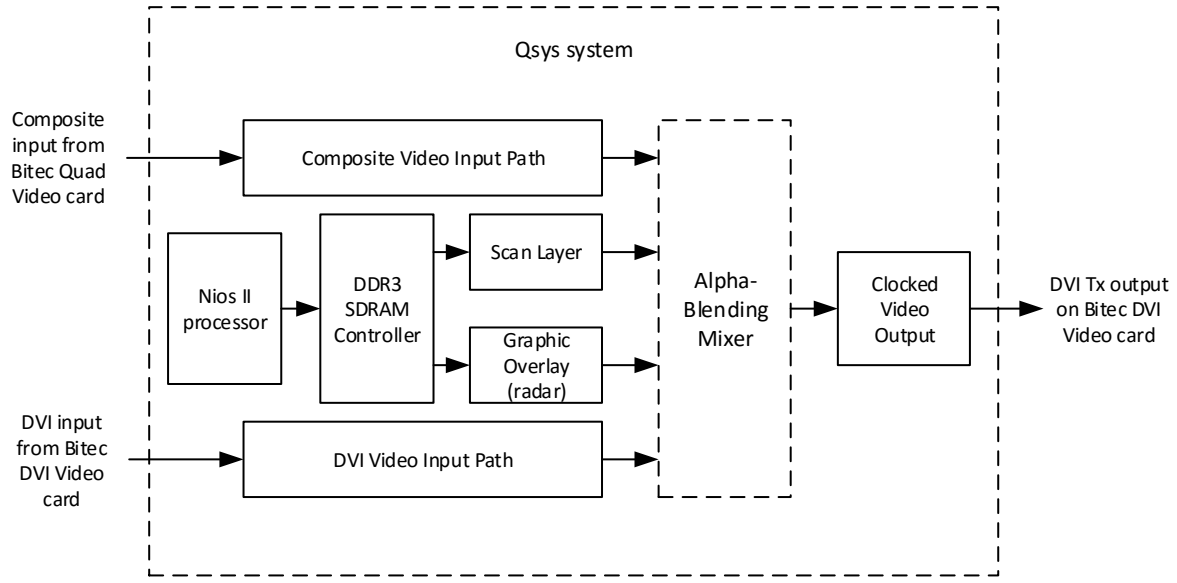
- For more information about System Console, refer to the *Analyzing and Debugging Designs with System Console* chapter in volume 3 of the *Quartus II Handbook*.

System Console receives this data and displays it in a real-time table view, with high-level decoding of each packet's contents. The table view allows you to filter and search the data stream. You can also communicate with the monitors in the system to further configure them.

4 Functional Description

Figure 3 shows a block diagram for the Video and Image Processing Design Example.

Figure 3 - Design Example Block Diagram



- Composite and DVI video path.
- A Nios II processor for run-time control.
- DDR3 SDRAM controller for external and on-chip memory.
- A four-layer Alpha-Blending Mixer with the following layers on top of a uniform black background:
  - Composite Video Input.
  - Graphic Overlay for the static radar background (also used for on-screen text display).
  - Scan layer for dynamics objects in the radar. This layer is double buffered and the design switches the output frame every time a frame is rendered.
  - DVI Video Input

The design also includes the following components (not represented in the block diagram):

- Interval timer
- Vectored interrupt controller
- I2C bus masters
- Debugging system

All layers require access to the DDR3 SDRAM. Because of bandwidth limitation, the design limits the DVI video to a maximum resolution of 960x540 after the first clipper. For higher bandwidth requirements, you can use the top bank of the DDR3 SDRAM.

All Video and Image Processing Suite IP cores use the Avalon-ST Video protocol to transmit data, which increases productivity through IP flexibility and reuse.

> ➢ For more information about the Avalon-ST Video protocol, refer to the *Video and Image Processing Suite User Guide*.

4.1 Video Input

The design example has the following video processing datapaths:

- Composite (NTSC and PAL)
- DVI (720p60, 720p50, 720p25, 720p24, 1080p25 or 1080p24 video stream)

You may input a composite video stream (BT656 format) via the first composite input port on the Bitec Quad Video daughter card or for DVI input via the RX DVI port. The TVP51 54 video decoder chip on the Bitec Quad Video daughter card performs analog-to-digital conversion of the video input signals, which you must configure using the I2C interface.

The Nios II processor controls the configuration of the decoder chip by writing and reading data to and from the I2C bus controller, which performs writes and reads on the external bus as requested.

The video stream, with active picture data in YCbCr 4:2:2 or RGB format and associated embedded synchronization signals, is input from a daughter card into the Clocked Video Input I P core on the FPGA. This IP core converts from a notionally clocked video format (such as BT656 or DVI) to the flow controlled Avalon-ST Video protocol. The clocked video input forms the boundary of the Qsys design at the input. the design exports the clocked video signals to the top-level system for connection to the input pins from the Quad Video daughter card.

The Clocked Video Input IP core strips the incoming clocked video of horizontal and vertical blanking, leaving active picture data. The Clocked Video Input IP core uses this data with horizontal and vertical synchronization information, to create the necessary Avalon-ST Video control and active picture packets.

The Clocked Video Input does not convert the active picture data; the color plane information remains the same as in the clocked video format.

The clocked video input also provides clock crossing capabilities to allow video formats running at different frequencies to enter the system. The clocked video input also provides greater design flexibility and enables the design to decouple the video processing system clock from the video input pixel clock. In this example, the pixel clock is 27 MHz and the system clock is 100 MHz.

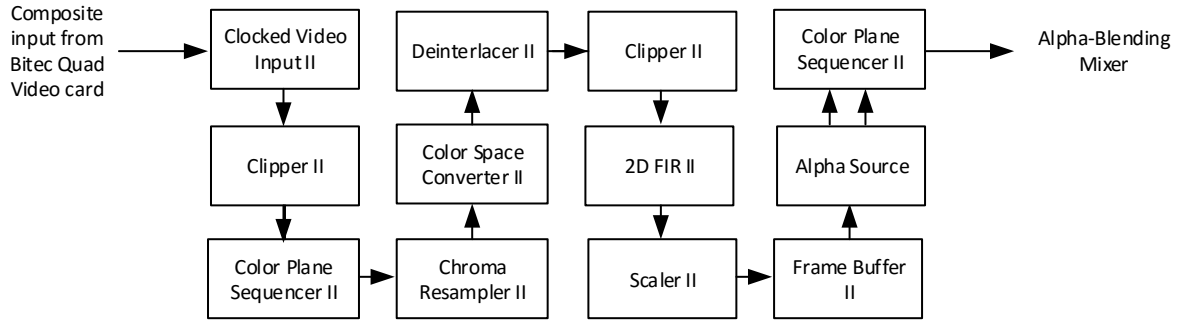
In addition to configuring the TVP5154 decoder chip, the Nios II processor starts all the clocked video inputs in the system.

4.2 Video Processing

4.2.1 Composite Video Processing

The video processing data path is a chain of parametrizable video processing, integration, and buffering functions from the Video and Image Processing Suite. Figure 4 shows a block diagram of the composite video input data path.

Figure 4 - Composite Video Input Data Path



The **top\vip_top.v** file shows the exported signals in system instantiation (Figure 5).

Figure 5 - Signals Exported from the Quad Video Daughter card

```
`ifdef BITEC_QV_IN
        //CVI II interface
        .cvi_ntsc_0_clocked_video_vid_clk              (BITEC_QV_CH1_IN_CLK), //
        .cvi_ntsc_0_clocked_video_vid_data             (BITEC_QV_CH1_IN_D),   //
        .cvi_ntsc_0_clocked_video_vid_de               (BITEC_QV_CH1_IN_AVID),//
        .cvi_ntsc_0_clocked_video_vid_datavalid        (1'd1),                //
        .cvi_ntsc_0_clocked_video_vid_locked           (1'd1),                //
        .cvi_ntsc_0_clocked_video_vid_f                (),                    //
        .cvi_ntsc_0_clocked_video_vid_v_sync           (),                    //
        .cvi_ntsc_0_clocked_video_vid_h_sync           (),                    //
        .cvi_ntsc_0_clocked_video_vid_color_encoding   (8'd0),                //
        .cvi_ntsc_0_clocked_video_vid_bit_width        (8'd0),                //
        .cvi_ntsc_0_clocked_video_sof                  (),                    //
        .cvi_ntsc_0_clocked_video_sof_locked           (),                    //
        .cvi_ntsc_0_clocked_video_refclk_div           (),                    //
        .cvi_ntsc_0_clocked_video_overflow             (overflow_at_input),   //

        .quad_video_i2c_master_global_sda_pad_io       (BITEC_QV_TVP5154_SDA      ),
        .quad_video_i2c_master_global_scl_pad_io       (BITEC_QV_TVP5154_SCL      ),
`endif
```

- **NTSC Video Stream Input**

The NTSC video stream input into the datapath is 8 bits wide, in YCbCr 4:2:2 interlaced format, with each field input containing 720 pixels per line, and either 244 lines for each even field (f0) and 243 lines for each odd field (f1). The design transmits the luma (Y) and the chroma (Cb and Cr) color planes in sequence in the CbYCrY order.

- **Clocked Video Input (NTSC)**

In this design, the clocked video input function accepts video data with two color planes in sequence, namely, C-Y. The design treats the subsampled Cb and Cr color planes as the C color plane. The clocked video input can compute the resolution of a video frame only after it has received a full frame of data. Before the design outputs the first video frame, it outputs a default command packet. This packet matches the resolution and format of the expected NTSC video input. The design configures the Clocked Video Input block to use separate clocks for the clocked video side and the flow controlled domain. The Clock Video Input block also exposes an Avalon-MM slave port. This port connects to the Nios II processor data master in this system and gives the Nios II processor control over when the clocked video input starts accepting data. The register map contains many status registers that provide feedback on the format of video entering the system (resolution, interlaced, or progressive). You could use a status interrupt and write software code to react to a resolution change but this design does not demonstrate that feature.

➢ For further information about the control register map, refer to the *Video and Image Processing Suite User Guide*.

- **Clipper (720x480)**

The first processing function after the clocked video input in the data path is a Clipper IP core, clocked in the system clock domain (100 MHz). It clips a rectangular region of 720 pixels by 240 lines from each field. It offsets three lines from the top of each F0/F1 fields, and outputs fields with an equal number of odd and even lines (240 lines), for further processing. The clipper can automatically react to changes in the image resolution at run-time up to a maximum width and height that you specify in the MegaWizard interface. In this design, the clipper is configured at compile-time to perform the same clipping on NTSC and PAL inputs. You could include an Avalon-MM slave interface that enables run-time changes to the clipping region and monitor changes of input resolution to perform a different clipping on PAL and NTSC inputs.

- **Color Plane Sequencer (sequence to parallel converter)**

The design uses the Color Plane Sequencer IP core to convert the clipped image video data from two colors in sequence (the Luma Y, and Chroma C, color planes) to two colors in parallel. This IP core demonstrates how you can change the format of the video data to suit the processing requirements of the functions in the data path and to satisfy the performance requirements of the application.

- **Chroma Resampler (4:2:2 -> 4:4:4)**

The design uses the Chroma Resampler IP core to chroma resample the Avalon-ST Video data to YCbCr 4:4:4 format. The chroma resampler uses a filtered luma-adaptive algorithm to convert the YCbCr 4:2:2 subsampled pixel stream to a 4:4:4 sampled stream.

  ❖ For legacy reasons, the example design upsamples the data before deinterlacing. Newer designs should typically use the deinterlacer 4:2:2 support mode and, if needed, upsample the data **after** deinterlacing to save memory bandwidth.

- **Color Space Converter**

The design uses the Color Space Converter IP core to convert the YCbCr color space to the RGB color space.

- **Deinterlacer**

In this design, the deinterlacer II applies a motion adaptive algorithm. The design stores previous fields in external memory, because the algorithm uses information from previous fields to compute the progressive frame pixel values as a function of motion between fields.

In this mode, the deinterlacer II has five Avalon-MM master ports (two to write, and three to read data), which the design connects to an external memory with enough space to store multiple frames of video data and one frame of motion data.

The deinterlacer II allows you to specify a burst length target for writing and reading data to external memory. Also, you can configure the FIFO depths between the Avalon-MM masters and the data path. Multiple components access the external memory in this design. The FIFOs are necessary to smooth memory accesses without stalling the data path.

- **Clipper (run-time configurable)**

A second Clipper IP core clips the progressive data output from the deinterlacer II. The clipping region can be changed at run-time to demonstrate a “zoom” effect with the help of the downstream scaler that maintains a constant output resolution.

- **2D Fir Filter**

The 2D FIR filter produces a softening or sharpening effect. The Nios II processor set the coefficients of the FIR filter at run time.

- **Scaler (upscale up to 1920×1080)**

The Scaler II IP core upscales the video output. The design uses an Avalon-MM slave interface to configure the Scaler at run-time and to control the scaler output width and height up to a maximum of 1920×1080 pixels. The Scaler II uses the Polyphase scaling algorithm (configured for four vertical and four horizontal taps) with 16 vertical and horizontal phases using fixed compile-time coefficients.

- **Frame Buffer**

The design uses the frame buffer IP core to buffers the video stream in external memory. The maximum resolution that this Frame Buffer instance supports is 1920×1080 pixels. The Frame Buffer streams data from the Avalon-ST sink interface into a dual clock FIFO buffer. This design sets the Frame Buffer to support rate conversion by dropping and repeating frames.

When the number of data items in the FIFO buffer is above the burst target, the Avalon-MM bursting write master drives a write request to external memory, with a burst length typically equal to the burst target. Similarly, the bursting read master drives a read request to read data with a burst length typically of length equal to the burst target.

Building video systems gives many challenges. One common design requirement is to achieve the mandatory level of external memory bandwidth when multiple components store video frames in a

shared DDR external memory. This requirement often necessitates a flexible system, where the design can decouple the data processing rate from external memory clock rate constraints.

Both the Frame Buffer and Deinterlacer IP cores use dual clock FIFOs to separate the data processing clock domain from the external memory controller clock domain to provide built-in clock crossing capability.

Data is input via the Avalon-ST sink interface as 24-bit wide data (three colors in parallel). The design reads and writes 48 bits (two pixels) of data from external memory via an external memory port width of 64 bits.

In addition, the design enables run-time control of the reader thread, which provides an Avalon-MM slave port. This system connects the Nios II processor data master to the Avalon-MM slave, which allows the Nios II processor to control when the Frame Buffer starts outputting data.

- **Alpha source**

The video data passes through an alpha source that provides alpha data, as the alpha blending mixer requires an alpha channel with the color data. The Alpha Source IP core propagates the input video stream unchanged and generates an Avalon-ST Video stream with a fixed value alpha channel that matches the length of the original video stream. The Color Plane Sequencer II IP that follows the alpha generator concatenates the alpha stream with the RGB video data since this is the format the Mixer II expects.

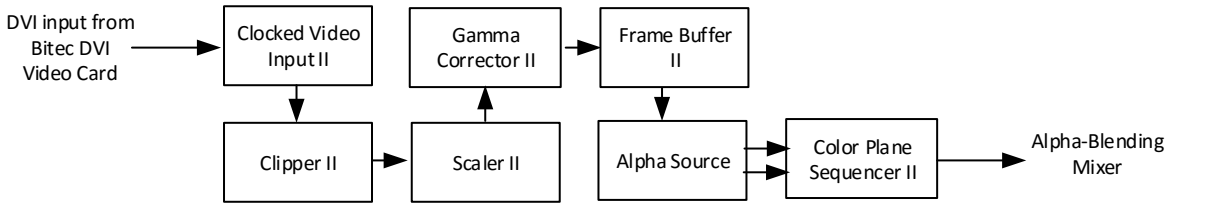
- **Avalon-ST Video monitors**

The design adds a couple of Avalon-ST Video Monitors to intercept and monitor Avalon-ST Video data between IP cores. The monitors are enabled and disabled at run-time with the System Console tool (see *"Setting Up System Console for Debug Visualizations"*). For debug purposes, the monitor captures all control packets and can also capture video pixel data if requested. The monitor does not affect the content nor the throughput of the data streaming through.

4.2.2 DVI Video Processing

The DVI data path allows higher-definition inputs and demonstrates the Gamma Corrector IP core. The design uses the DVI data path as an input to the Alpha Blending Mixer IP core, to show how you combine multiple different video sources. Figure 6 shows a block diagram of the DVI data path.

Figure 6 - DVI Input Data Path



The video processing functions for the DVI stream are similar to the composite path, but it does not require color plane conversion. There is no deinterlacer in this path either, so the input format must be progressive. The video stream input into the processing data path is 24-bits wide (three colors in parallel), in RGB progressive format. Each frame input contains either 1,280 or 1,920 pixels per line, and 720 or 1,080 lines, respectively.

The first processing function in the data path clips a rectangular region of 800 pixels by 600 lines from each field, offset by 0 from the top left of the input video stream.

The design then uses the polyphase algorithm of the parametrizable scaling function (with four horizontal and four vertical taps) to scale the clipped progressive video stream.

To smooth out the burstiness of the video data flow, the design buffers the scaled video stream in external memory. The frame buffer also manages the rate change across the system video input and output by dropping and repeating video frames.

- **Gamma corrector (run-time configurable)**

To correct any gamma problems present in the video stream, the gamma corrector can implement ITU-709 correction with adjustment for black and white levels and uses lookup tables for each color plane in RGB. The gamma corrector has three lookup tables for each color plane in RGB and implements ITU-709 correction with adjustment for black and white level. The gamma corrector assumes that the DVI input stream is non-linearly encoded and is not decoded. Performing decoding on a decoded video stream gives the video excess brightness. For decoded input video streams, the gamma corrector should be switched to linear mode.

4.3 Video mixing

The design mixes the buffered input video streams and the Nios II generated graphics against a uniform background layer of resolution 1024×768 pixels (i.e., picture-in-picture mixing). The Nios II graphic layers are generated with a per-pixel alpha value so that the text and the radar display can be layered directly on top of the video inputs. The Mixer II IP exposes an Avalon-MM Slave interface to program the location of all the foreground layers, relative to the top-left pixel of the background layer, at run-time. The color of the uniform background and the alpha value used for the NTSC and DVI video streams are modifiable at run-time. In this design, the Nios II processor writes control data to the mixer and other functions in the data path to control the demonstration.

4.4 Video Output

The design streams the video data output from the mixer into the clocked video output function, which supplies the DVI. The Clocked Video Output IP core converts from the Avalon-ST Video protocol to clocked video formats (such as BT656 and DVI). It uses the Avalon-ST Video control and active picture packets to format Avalon-ST Video into clocked video by inserting horizontal and vertical blanking and generating horizontal and vertical sync information. The active picture data is not modified; the color plane information remains the same as in the Avalon-ST Video format.

This design configures the Clocked Video Output IP core to accept video data with three color planes in parallel, namely, RGB. The IP core also outputs clocked video data with the sync signals on separate wires, suitable to transmit data over a DVI interface. The clocked video output also provides clock crossing capabilities to provide greater design flexibility by enabling the video processing system clock to be decoupled from the video output pixel clock. The output pixel clock is set to 65 MHz and the system clock is 100 MHz.

The Clocked Video output exposes an Avalon-MM slave interface and this design connects the Nios II processor data master to it. This gives the Nios II processor control over when the Clocked Video Output starts accepting data.

The register map also contains many status registers that provide feedback on the format of video entering the Clocked Video Output (resolution, interlaced or progressive). You can use a status interrupt, but this design does not demonstrate that feature. In addition, the clocked video output provides a queue where pixels can wait when the DVI output is in blanking and does not require pixel data. If this FIFO buffer becomes full, the flow controlled interface indicates that the clocked video output is not ready for data and earlier parts of the pipe are stalled.

The design outputs a video stream (progressive RGB data) via the DVI Tx port on the Bitec HSMC DVI daughter card. The Nios II processor controls the configuration of the TFP410 DVI chip by writing or reading data to or from the I2C bus master component, which performs writes and reads on the external bus as requested. The design exports the clocked video signals to the top-level system for connection to the output pins for the Bitec HSMC DVI daughter card. The file **top\vip_top.v** shows the exported signals for the DVI input and the DVI output in the system instantiation (Figure 7).

Figure 7 - Signals Exported to/from the Bitec DVI Daughter Card

```
`ifdef BITEC_DVI
        // CVI II interface
        .cvi_dvi_3_clocked_video_vid_clk            (BITEC_DVI_IO_IN_ODCK), // cv
        .cvi_dvi_3_clocked_video_vid_data           (BITEC_DVI_IO_IN),      //
        .cvi_dvi_3_clocked_video_vid_de             (BITEC_DVI_IO_IN_DE),   //
        .cvi_dvi_3_clocked_video_vid_datavalid      (1'd1),                 //
        .cvi_dvi_3_clocked_video_vid_locked         (1'd1),                 //
        .cvi_dvi_3_clocked_video_vid_f              (),                     //
        .cvi_dvi_3_clocked_video_vid_v_sync         (BITEC_DVI_IO_IN_VSYNC),//
        .cvi_dvi_3_clocked_video_vid_h_sync         (BITEC_DVI_IO_IN_HSYNC),//
        .cvi_dvi_3_clocked_video_vid_color_encoding (8'd0),                 //
        .cvi_dvi_3_clocked_video_vid_bit_width      (8'd0),                 //
        .cvi_dvi_3_clocked_video_sof                (),                     //
        .cvi_dvi_3_clocked_video_sof_locked         (),                     //
        .cvi_dvi_3_clocked_video_refclk_div         (),                     //
        .cvi_dvi_3_clocked_video_overflow           (),                     //

        // CVO II interface
        .cvo_dvi_clocked_video_vid_clk              (BITEC_DVI_IO_OUT_IDCKp), //
        .cvo_dvi_clocked_video_vid_data             (BITEC_DVI_IO_OUT      ), //
        .cvo_dvi_clocked_video_underflow            (underflow_at_output),    //
        .cvo_dvi_clocked_video_vid_mode_change      (),                       //
        .cvo_dvi_clocked_video_vid_std              (),                       //
        .cvo_dvi_clocked_video_vid_datavalid        (BITEC_DVI_IO_OUT_DE   ), //
        .cvo_dvi_clocked_video_vid_v_sync           (BITEC_DVI_IO_OUT_VSYNC), //
        .cvo_dvi_clocked_video_vid_h_sync           (BITEC_DVI_IO_OUT_HSYNC), //
        .cvo_dvi_clocked_video_vid_f                (),                       //
        .cvo_dvi_clocked_video_vid_h                (),                       //
        .cvo_dvi_clocked_video_vid_v                (),                       //

        .dvi_out_i2c_master_global_sda_pad_io       (BITEC_DVI_IO_OUT_DVI_DAT  ),
        .dvi_out_i2c_master_global_scl_pad_io       (BITEC_DVI_IO_OUT_DVI_CLK  ),
`endif
```

4.5 Graphic Overlay (Radar) and Scan Layer

The graphic overlay generates the radar visualization. The design stores all the rendered frames in DDR3 SDRAM and uses the Nios II processor software plotting functions to modify the frames. The design uses two frame readers to feed the overlays to the mixer — the scan layer frame reader and the graphic overlay frame reader. Each frame reader reads a set of video frames from memory and sends them to the alpha blending mixer.

The video data is written and stored in external memory in the RGBA color format. This is coincidentally the format that the Mixer II IP expects so no further conversion is necessary after the two frame reader cores.

4.6 Interval Timer

The design includes an interval timer to raise an interrupt every two seconds. This is used by the Nios software to calculate the output frame rate and to calculate the OSD refresh rate.

4.7 Vectored Interrupt Controller

For better performance, the design includes a vectored interrupt controller to manage the interrupts of the Nios II processor. The interrupt from the interval timer is set to have the highest priority, then the interrupt from scan Frame Reader. JTAG has the lowest interrupt priority.

4.8 DDR3 SDRAM Controller and External DDR3 SDRAM

A DDR3 SDRAM Controller arbitrates access to the external memory. The external memory is used by the frame buffering and the deinterlacing functions to store video frames. The external memory is also used by the Nios II and the two frame reader IP cores to render and output the graphic overlay layers. The Altera DDR3 SDRAM Controller controls the data flow to and from the external DDR3 SDRAM. The design uses the controller in half-rate mode, operating at 150 MHz, with a local data interface width of 128 bits. This configuration provides a theoretical maximum available bandwidth of 75×128 = 9.6 Gbps. The design uses the 100-MHz system clock as the PLL reference clock.

4.9 I2C Bus Masters

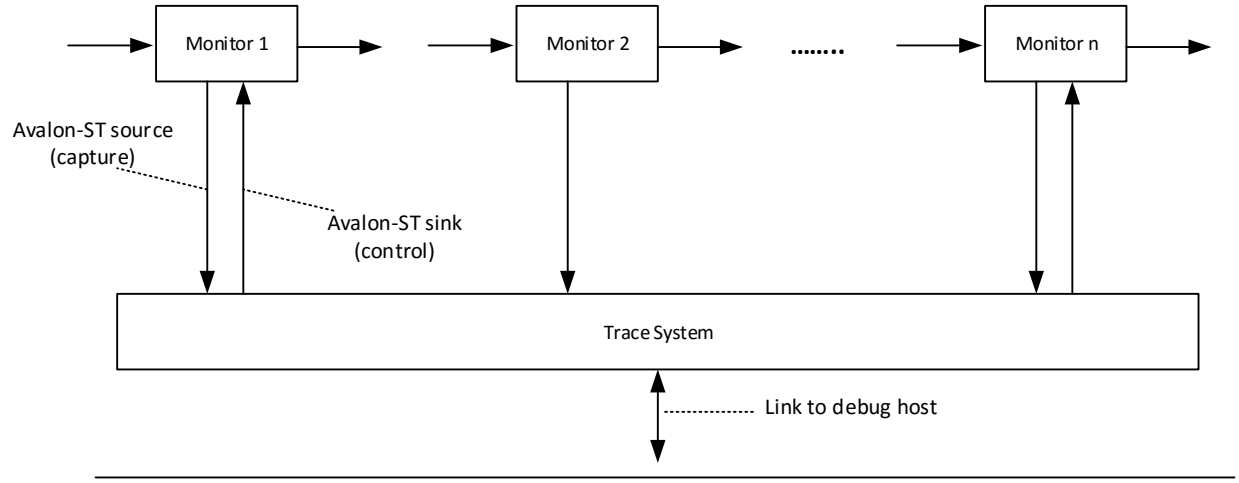
This design instantiates two OpenCore I2C bus masters to enable you to configure both the TVPS154 and TFP410 video interface chips on the HSMC Quad Video and HSMC DVI daughter cards, respectively. The I2C bus master Qsys components provide a convenient interface to configure the video interface chips. An Avalon-MM slave interface receives read and write commands, which the design translates into I2C bus transactions. The I2C bus masters are not parametrizable through a user interface in Qsys, the HDL source code is included in the local project directory (**ip/altera_avalon_i2c)**.

The Nios II processor controls the configuration of the video interface chips by writing and reading data to and from the I2C bus master component.

4.10 Debugging System

The debugging system allows you to monitor signal data passing between Qsys components and send this data to a host machine running System Console. System Console decodes this data, to provide high-level information about the contents of the signals passing between the components. It also monitors flow control statistics about the connection, allowing you to see how many cycles the design uses to transfer data or to wait for either source or sink to be ready. You can gather this information in real-time from a system over JTAG or USB links. Figure 8 shows the debugging system.

Figure 8 - Debugging System



The debugging system comprises two major components: the monitors and the trace system IP core. The design includes monitors in the data path to capture data. The trace system is the central location to connect all the monitors to and acts as the gateway to the host machine running System Console.

4.10.1 Trace System

The trace system transports messages describing captured events from trace monitor components (for example the Avalon-ST Video Monitor) to a host computer running System Console.

You should parameterize the trace system to select the number of monitors that you require. The trace system exposes the `capture_i` and `control_i` interfaces. Connect each pair of interfaces to the appropriate trace monitor component. The design has four monitors, with **Number of inputs** is set to 4, which creates `capture_0`, `control_0`, `capture_1`, `control_1`. The **Bit width of capture interface** is the data bus width of the Avalon-ST interface, which sends the captured information. The design has this parameter set to 32 to match this parameterization in the monitors. This design uses the JTAG connection to host.

The trace system provides access to the control interfaces on the monitors (`control_i`). You can use these control ports to change capture settings on the monitors, for example to control the type of information captured by the monitors or the maximum data rate sent by the monitor.

When enabled, each monitor sends information about events through its capture interface (`capture_i`). The trace system multiplexes these data streams. When the trace system runs, it stores them into a FIFO buffer. System Console uses as much bandwidth as possible to stream the contents of this buffer to the host.

The amount of buffering required depends on the amount of jitter inserted by the link. The design uses a buffer size of 8,192 (8 K).

You can add pipeline stages to the design to speed up the system, to give it a greater $f_{MAX.}$ However, adding pipelining increases the amount of logic required in the design. The design uses **Insert pipeline stages on all capture inputs**.

System Console uses the .**sopcinfo** and **.jdi** files to discover the connections between the trace system and the monitors. If you use HDL to manually instantiate and connect the trace system and the monitors, System Console cannot see them.

4.10.2 Monitor

The design has monitors in the data path to capture signals flowing between components. The monitors only tap the wires, and do not add backpressure, so the component cannot stop or stall the flow of data and behave transparently in a design. Nevertheless, whenever you add any extra component, this addition can affect the Quartus II fitter, and changing the placement of other components could affect timing.

Monitors encapsulate raw data and statistics from the wires that they tap and send this data to the trace system, which forms the data into packets and sends it to the host machine running System Console, which decodes them into accessible visual representations of the packets. You use Qsys to insert the monitor into the data path between the two components that you want to monitor.

For example, to monitor communications between two components A and B that send data via the Avalon-ST protocol, insert an Avalon-ST video monitor into the design that connects **A.dout** to **monitor.din**, and **monitor.dout** to **B.din**.

The Trace System uses the control port to control monitors. Every monitor has a register map with common functionality, such as enabling and disabling the monitor from capturing. In addition to these shared functions, monitors can have specific controls relating to their own operation. A capture port also exists in the monitor, which sends the captured data from the monitor to the trace system. Connect both the capture and control ports to the trace system, at `capture_n` and `control_n`, respectively.

Connect the clock to the same source that the monitored components connect to. Connect reset from the trace system's reset.

The monitors' parameters must match the values of the IP cores you insert them between. The **Bits per pixel per color plane** parameter is the number of bits the design requires to represent one part of a color space. For example, a value of 10 when using a color space of RGB means that every pixel requires 10 bits for each red, green and blue color planes, producing a total of 30 bits required for each pixel. If you use the ARGB color space, the design requires 40 bits to represent alpha, red, green, and blue. **Number of color planes in parallel** and **Number of color planes in sequence** describe how the video system packs and transfers data between IP cores. The **Bit width of capture interface(s)** is the width of the data bus from the monitor to the trace system. These values must match.

When you turn on **Capture video pixel data**, Qsys instantiates hardware that allows monitors to capture pixels from the data packets. You can then look at the contents of the images that are passing between two components, in real time.

For more information about how to see this data, refer to "*Setting Up System Console for Debug Visualizations*".

4.11 Nios II Processor and On-Chip Memory for Program Code

The Nios II processor initializes and configures multiple video system components, including the I2C external bus masters and the Video and Image Processing Suite IP cores.

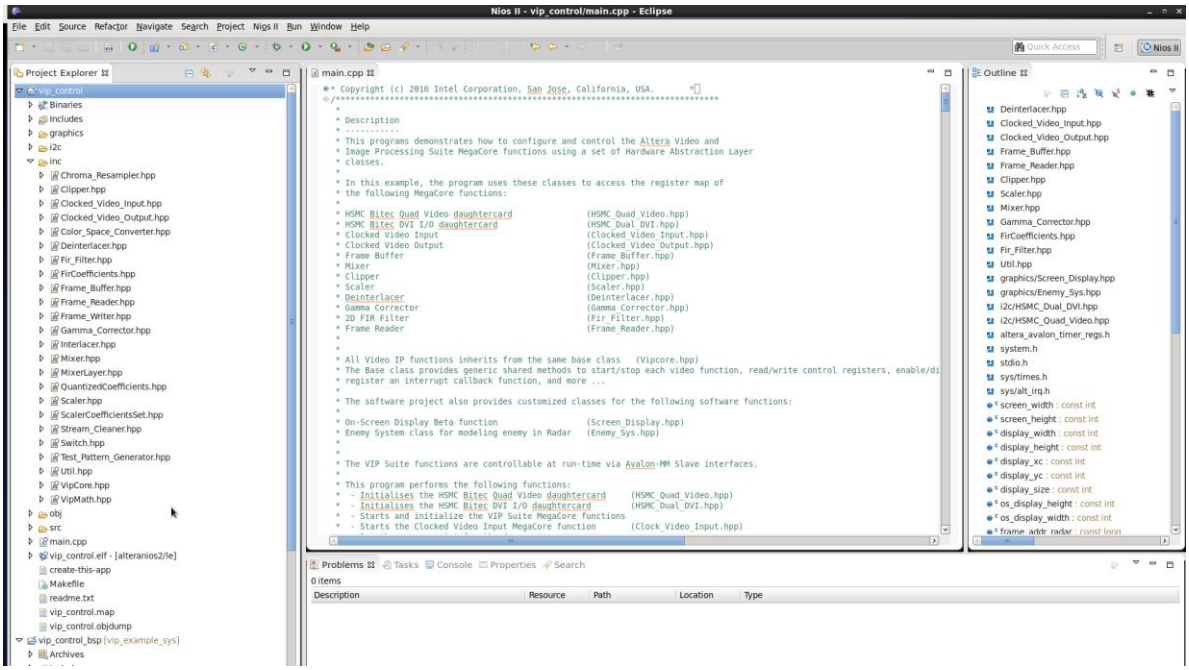
The design connects the Nios II Avalon-MM data master to each Avalon-MM slave port it needs access to. Qsys automatically constructs the bus architecture. The system stores the Nios II program code in on-chip memory, with 100 Kbytes of available space.

Nios II processor performs the following functions:

- Configures the Bitec Quad Video daughter card video chips (TVP5154 and Chrontel)
- Configures the Bitec DVI daughter card video chips (TFP410)
- Initializes and starts the Video and Image Processing Suite IP cores via the Avalon-MM slave control interfaces:
  - Controls the clippers to change the height and width of the clipped video region
  - Controls the scalers output resolution
  - Controls the 2D FIR filter to switch between soft, hard, and edge sharpening
  - Controls the gamma corrector to switch between inverted colors mode and ITU-709 mode.
  - Controls the mixer:
    a) To turn the inputs on and off (and flashing them)
    b) To change the location of the video streams relative to each other on the 1024×768 background.

The **vip_control** application project contains a description of the application in the **main.cpp** file (Figure 9).

Figure 9 - main.cpp in Nios II SBT for Eclipse



The **vip_control** project includes C++ classes. The C++ classes provide a software API between the Nios II control code and the Video and Image Processing Suite IP cores. The classes provide many member functions to accelerate object-oriented code development and increase visibility of the data flow. For example, the clocked video input class (**Clocked_Video_Input.hpp**) member functions can report the level of the input FIFO buffer. The member functions in the frame buffer class (**Frame_Buffer.hpp**) can report the number of frames that the Frame Buffer drops or repeats in the data path.

All Video and Image Processing Suite IP core classes derive from the base class **VipCore.hpp**, which contains methods common to all functions such as starting and stopping the video function processing at a frame boundary. The base class also provides support for adding interrupt handling functions.

- Main Function

The `main` function in **main.cpp** provides examples of how to use the C++ class member functions to configure and control the data path functions.

The main function sets up and initializes all the IP cores at the beginning, then runs the scene script to show the demonstration. At the end, the main function plots the radar and the design enters the main loop to update the radar. The main loop runs forever (unless terminated).

- Scene Script

The `scene_script` function controls the demonstration during each stage. Variable instant states the current time. Different commands are executed at certain time instances. For example:

```
switch (instant) {
   case 100: Turn on DVI input; break;
   case 150: Draw text "This is DVI input"; break;
   …
}
```

For some continuous actions, like moving or scaling, keep the time instant at the instant of that command so that this command can be repeated multiple times. Hence variable "transition" states the lack of time instant. Calculate an effective instant by subtracting the transition from the current time instant. For example:

```
scene loop {
   instant++;
   effective_instant = instant - transition;
   switch (effective_instant) {
      case 100: Turn on DVI input; break;
      case 150: Draw text "This is DVI input"; break;
      case 200: {
         transition++;
         if (transition == 50) reset transition to 0;
         set layer position (x, transition);
         break;
      }
      case 250: Draw text "This is Composite input"; break;
      …
   }
}
```

In this design, a moving action occurs at time 200 until time 250. By increasing the transition variable each loop and subtracting it from the instant variable, the effective time instant is kept at 200 until transition reach 50. Thus, the set position command is repeated 50 times, with different parameter value. After that, the time instant continues at 250.

When the whole scene finishes, the function returns false to exit the loop. This scene script is compatible to run within the main loop, so that the control of the video inputs can be run at the same time as other tasks, for example rendering the radar. However, this design does not use it in the main loop.

The .**hpp** files included in the application project allow you to understand the control capability of the Nios II software environment.

5 Running the Example Design

This section describes how to open and run the Video and Image Processing Design Example. This section includes the following walkthroughs:

- *"Opening the Quartus II Top-Level Project"*
- *"Opening the Qsys System"*
- *"Viewing the Parameters"*
- *"Compiling the Design"*
- *"Set up the Cyclone V Video Development Platform"*
- *"Configure the Cyclone V Device"*

Once the design is programmed onto the board, you can download the Nios II program, which runs the design example, to the FPGA in one of two ways:

- Build the binary from the Nios II SBT for Eclipse. Use this flow to understand how to build the software and debug the design example. Continue to *"Building, examining and debugging the Software in the Nios II SBT for Eclipse"*.
- From a prebuilt binary object file via the Nios II command shell. Use this flow to quickly see the demonstration working on the hardware. Continue to *"Download and Run the Nios II Control Program*".

5.1 Opening the Quartus II Top-Level Project

To open the top-level Quartus II project, perform the following steps:

1) Launch the Quartus II software.
2) On the File menu, click **Open Project**, browse to <*design example install directory*>, and select the **top.qpf** Quartus II project file
3) On the File menu, click **Open,** browse to <*design example install directory*>**\top**, and select the **vip_top.v** top-level Verilog HDL design file.

5.2 Opening the Qsys System

To open the design example Qsys system, with the **top.qpf** project open in the Quartus II software, click **Qsys** on the Tools menu.

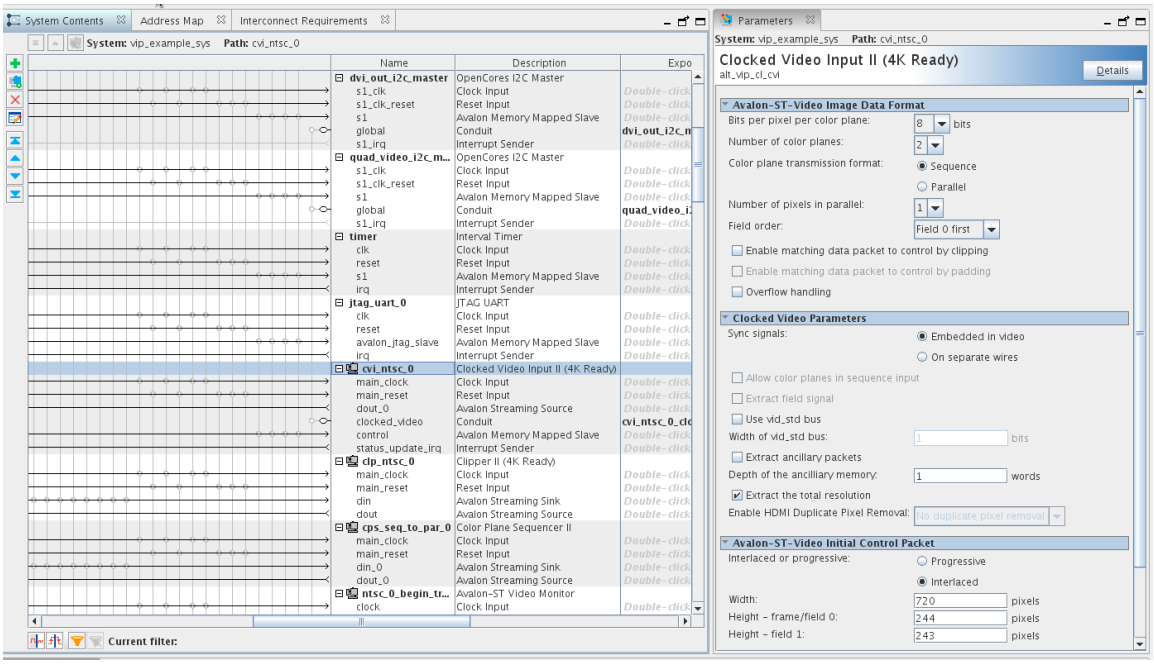
❖ When you launch Qsys, a dialog box displays for you to select the design file. Always select the **vip_example_sys.qsys** file for this design.

5.3 Viewing the Parameters

You can view the parameters of each IP component, to see what parameterizations are available for customized designs. At this stage, do not change any of the parameters. To view the parameters, perform the following steps:

In Qsys, in the **Name** column, click **cvi_ntsc_0** to display the parameterization interface for the Clocked Video Input II (Figure 10).

Figure 10 - Parameter Settings for the Clocked Video Input II



Select other components as desired to display their parameterization.

5.4 Compiling the Design

With the **top.qpf** project open in the Quartus II software, click **Compile** on the Tools menu. Compiling the design is expected to take between 20mn and 2 hours depending on computer performance and RAM available.

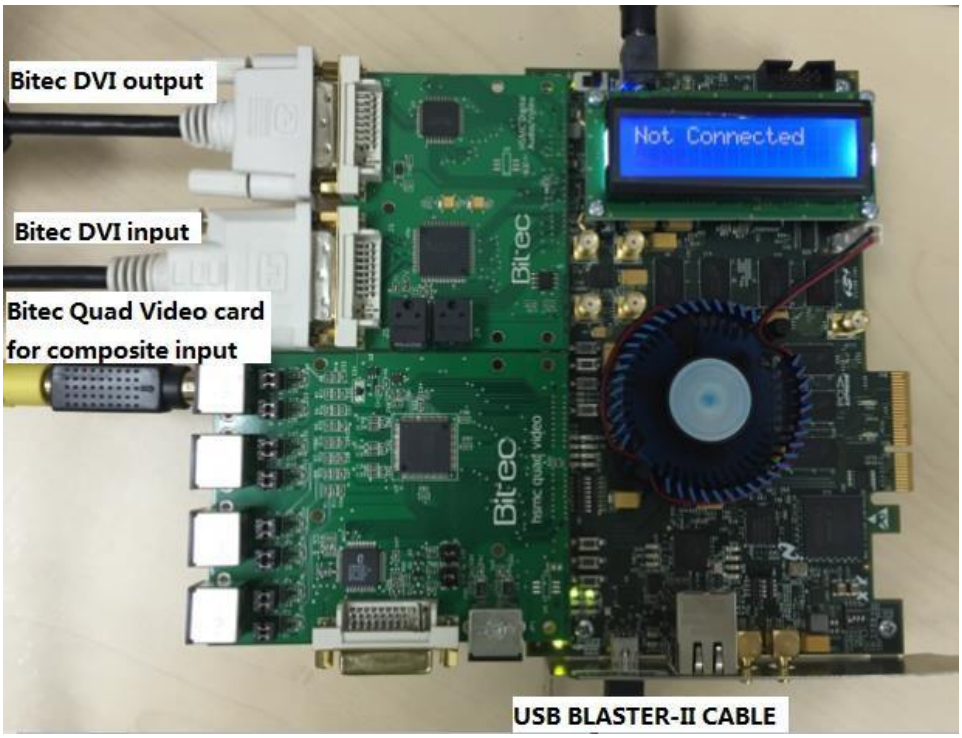
- Intel provides a prebuilt image **master_image\top.sof** so you can program the design example without having to recompile it

5.5 Set up the Cyclone V Video Development Platform

This section describes how to set up the Cyclone V Video Development Board to showcase the example design.

Figure 11 shows the hardware setup.

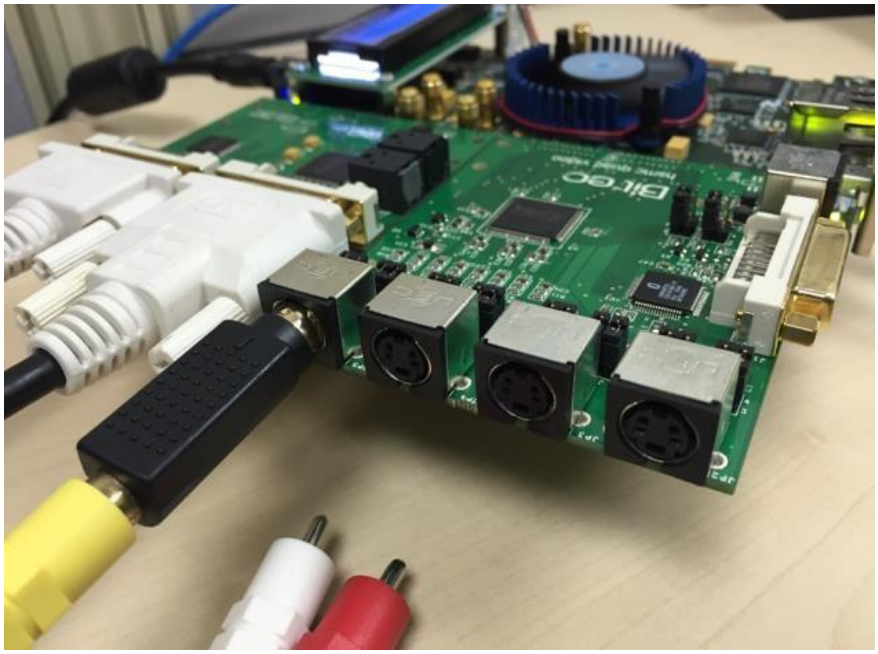
Figure 11 - Cyclone V Development Board Hardware Setup



To set up the Cyclone V development board, perform the following steps:

1) Ensure you disconnect the power cable from the Cyclone V development board.
2) Connect one end of the USB-Blaster cable to the USB port on your PC.
3) Connect the other end of the USB Blaster cable to the 10-pin header labeled (JTAG) on the Cyclone V development board.
4) Connect the Bitec HSMC Quad Video daughter card to HSMC interface port A(J1) on the Cyclone V development board.
5) Connect the video source composite input cable to the top left input connector (J10) on the Bitec Quad Video daughter card (Figure 12).
6) Connect the Bitec HSMC DVI video daughter card to the HSMC interface port B(J2) on the Cyclone V development board.
7) Connect one end of the DVI cable to a DVI monitor, capable of 1024×768 @ 60Hz. Connect the other end to the TX connector on the Bitec DVI daughter card.
8) (optional) Connect one end of the second DVI cable to a suitable DVI input video source. Connect the other end to the RX connector on the Bitec DVI daughter card.
9) Connect the power cable to the Cyclone V development board.
10) Locate the switch on the board located close to the port for the power cable. Change the switch to the **On** position so that the LED labeled **POWER** illuminates.

Figure 12 - Composite Video Connection to Quad Video Daughter Card



- For details about installing the USB Blaster software driver on the host PC (located at **<*quartus_install_dir*>\drivers\usb-blaster-ii**), refer to the *USB-Blaster Download Cable User Guide*.

- For details about the Cyclone V GT development board, refer to the *Cyclone V Development Board Reference Manual*.

5.6 Configure the Cyclone V Device

To configure the Cyclone V device, download the **top.sof** file to the development board by following these steps:

1) In the Quartus II software, on the Tools menu, click **Programmer**.
2) On the File menu, click **Save As**.
3) Navigate to the design installation directory, and in the **Save As** dialog box, type **vip_top.cdf** in the **File Name** box.
4) In the **Save as type** list, select **Chain Description File**.
5) Click **Save**.
6) In the Programmer window, select **JTAG** in the **Mode list**.
7) Click **Hardware Setup** to configure the programming hardware. The **Hardware Setup** dialog box appears.
8) In the **Hardware** column, click **USB Blaster-ii**.
9) In the **Currently select hardware**, select **USB-Blaster-ii**.
10) Click **Close** to exit the **Hardware Setup** dialog box.

11) Click **Auto Detect**, select device **5CGTFD9E5** from the list that appears. Click **Yes** on the popup window that appears to update the programmer's device list
12) Two devices have been detected on the JTAG chain. Select the **5CGTFD9E5** device and click **Change File…**
13) Browse to the design installation directory, click **top.sof**, or **master_image\top.sof** if using the precompiled sof, and click **Open**.
14) Turn on **Program/Configure** for **top.sof**.
15) Click **Start**.

The Programmer begins to download the configuration data to the FPGA. The **Progress** field displays the percentage of data that downloads. A message appears when the configuration is complete.

❖ If you do not use a licensed version of the Video and Image Processing Suite, a message may appear indicating that you are running a time-limited configuration file on your target hardware.

5.7 Download and Run the Nios II Control Program

Intel provides a prebuilt file so you may run the design example without using the Nios II SBT tools for Eclipse. This method only demonstrates the design example and does not allow you to learn how to build the software or how to debug the design example. This is covered in *"Building, examining and debugging the Software in the Nios II SBT for Eclipse*". To download, perform the following steps:

1. Complete all of the steps in *"Set up the Cyclone V Video Development Platform"* and *"Configure the Cyclone V Device"*.
2. Start a Nios II Command Shell (On the Windows Start menu, point to **Programs**, then **Altera**, then **Nios II EDS <***version***>**, and click **Nios II <***version***> Command Shell**).
3. Change the directory to **<***design example install***> / master_image**.
4. Download the **vip_control.elf** file and start the Nios II terminal window to display print output using the command: `nios2-download -g vip_control.elf; nios2-terminal`

   Provided the composite input is properly connected and recognized, the resolutions of the input video fields (F0 and F1) display in the Nios II terminal window and a video stream appears on the screen shortly after the introduction sequence.

6 Building, examining and debugging the Software in the Nios II SBT for Eclipse

6.1 Examining and Recompiling the Control Code

To use the Nios II SBT for Eclipse to examine and recompile the Nios II C++ control code and associated board support package (BSP), perform the following steps:

1) On the Windows Start menu, point to **All Programs**, point to **Altera**, point to **Nios II EDS** <*version*>, and then click **Nios II** <*version*> **Software Build Tools for Eclipse**. The **Workspace Launcher** dialog box appears.
2) Select a workspace folder by browsing to the design example root directory and specifying a workspace subdirectory. For example, <*install directory*>**\workspace**
3) Click **OK** to create a new workspace.
4) Copy the file <*install directory*>**\vip_example_sys.sopcinfo** at the root of the Nios II workspace directory <*install directory*>**\workspace**
5) On the File menu, point to **Import…**, and then click "**Existing Projects into Workspace".** Select archive file <*install directory*>**\software\vip_control_sw.zip.** Then click **Finish.**

6.2 Building the Software in the Nios II SBT for Eclipse

To build the software in the Nios II SBT for Eclipse, perform the following steps:

1) Right-click **vip_control** in the Project Explorer view, point to **Run as**, and then click **Nios II Hardware**. The Nios II SBT for Eclipse compiles the **vip_control_bsp** and **vip_control** software projects and builds the **vip_control.elf** software executable. After the build stage completes, the **Run Configurations** dialog box appears.
2) Verify that **vip_control** is in the **Project name** box, and the newly created **vip_control.elf** file is in the **ELF file name** box.
3) Click the **Target Connection** tab, and then click **Refresh Connections**.
4) Verify that the USB-Blaster and 5CGTFD9E configuration device appear under **Connections**.
   ❖ The .sof must be downloaded on the board to proceed.

6.3 Running the Nios II Control Program

To run the Nios II control program, perform the following steps:

1) Click **Run**. Log information appears in the Nios II Console output including the following log:

```
Composite input F0 width x height= 720 x 244
Composite input F1 width x height= 720 x 243
```

2) Verify that the design starts and displays text on the screen that guides you through the demonstration of the various functions.

3) You can rebuild, edit, and debug the control code in the Nios II SBT for Eclipse with the software API classes provided to access the Video and Image Processing Suite functions.

6.4 Debugging the Application Source Code

To debug the application source code, perform the following steps:

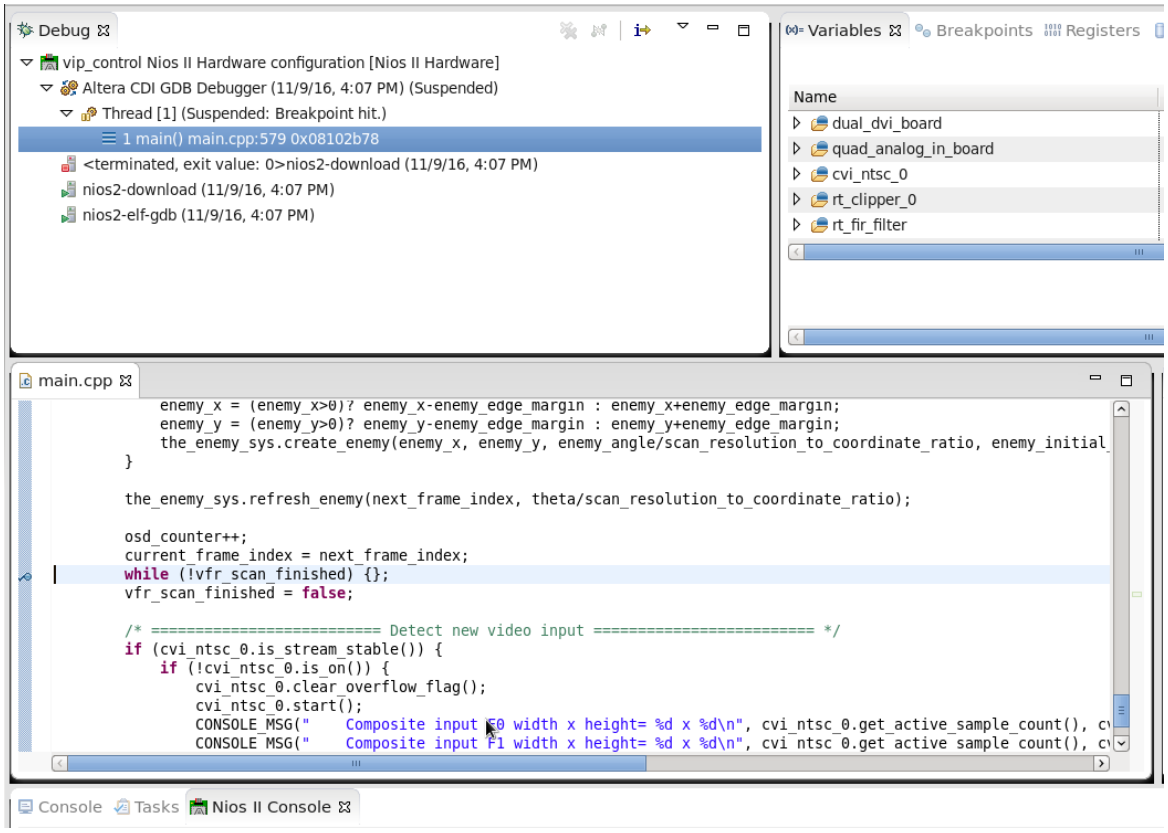
1) Right-click **vip_control**, point to **Debug As**, and then click **Nios II Hardware**.

❖ A message asking you if you want to switch perspective might appear. Clicking **Yes** switches to the debug perspective view. The VIP hardware stops manipulating the video, and the program pauses at the start of the **main()** function.

2) Insert a breakpoint by clicking in the margin next to the following line (Figure 13):

```
while (!vfr_scan_finished) {};
```

Figure 13 - Inserting a Breakpoint in the Nios II Debug Window



3) Right-click and click **toggle breakpoint**.

4) Click **Resume (F8)** to run the initialization code. The program runs through the video input demonstration and breaks at the `while (!vfr_frame_finished) {};` line, just when the radar display appears.

5) The output drops because the frame reader ISRs is not serviced and the Mixer stalls when active layers drop. Depending on your monitor, you may be able to see the output flashing when clicking **Resume (F8)** multiple times.

   ❖ In a debug session, you can perform many debug operations including breakpoint insertion, viewing registers, viewing memory data, viewing variables, and single stepping instructions.

6) To end the debug session, click **Terminate**.

   ➢ For more information about running a Nios II debug session, refer to "Getting Started" in the *Getting Started with the Graphical User Interface* chapter of the *Nios II Software Developer's Handbook*.

6.5 Setting Up System Console for Debug Visualizations

To set up the development board and host computer to use the trace system with System Console for debugging, follow these steps:

1) In Eclipse, right-click **vip_control**, point to **Debug As**, and then click **Nios II Hardware**.

   ❖ A message asking you if you want to switch perspective might appear. Click **Yes** to switch to the debug perspective view. The software running on the Nios II processor stops controlling the VIP cores and the program pauses at the start of the main() function.

2) Find the following section of **main.cpp**:

```
case START_COMP + 620:
    // Scale up the composite video
    if (transition > 70) {
           transition = 0;
    } else {
           transition++;
           comp_scaler.stop_and_wait();
           int new_width = int((float(transition)/100.0f+1.0f)*360.0f);
           int new_height = int((float(transition)/100.0f+1.0f)*240.0f);
           comp_scaler.set_output_resolution(new_width, new_height);
           comp_scaler.start();
    }
    break;
```

3) Insert a breakpoint by clicking in the margin next to the line:

```
"if (transition > 70) {"
```

This breakpoint stops the debugger when it reaches the point in the program where it is scaling up the composite input.

4) Click **Resume** to run the initialization code. The program runs through the video input demonstration until it breaks just before it starts to scale the composite video input. While the Nios II code is suspended, the composite video path is unaffected, and continues to display live video from the input source, because the Nios II processor only controls the parameters of the VIP IP cores. Suspending the code does not affect the operation of the VIP IP cores.

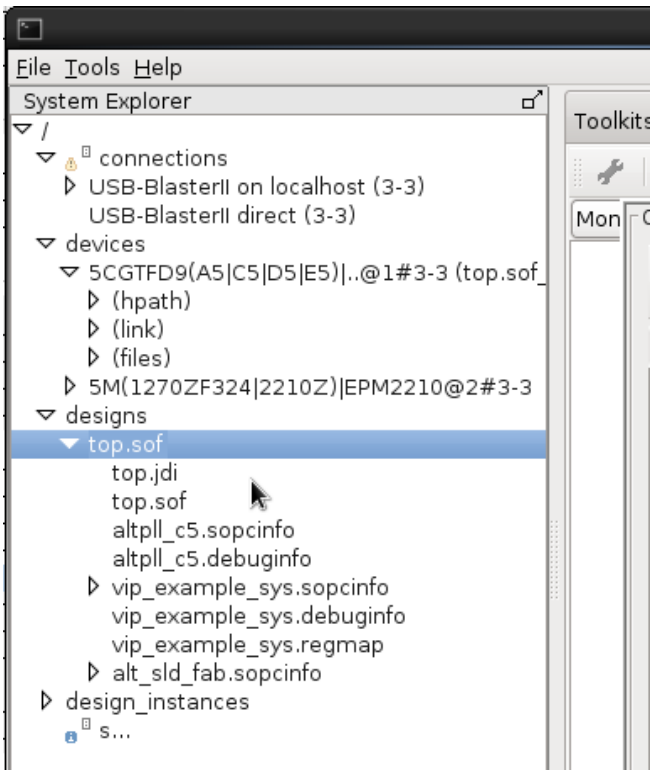
5) Leave Nios II SBT for Eclipse at this breakpoint while you start System Console.

6) Open System Console. In Qsys, on the Tools menu, select **System Console**.

7) Under **System Explorer**, expand **designs** and expand **top.sof** (Figure 14). If you cannot expand **designs**, perform one or both of the following steps:

   - Click on **File**, then click **Load Design…** and navigate to the **top.qpf** file.
   - Expand **devices**, right-click on the part number for the Cyclone development board **5CGTFD9E** and select **Link device to**, then click on **top.sof**.

Figure 14 - System Console



8) On the Tools menu select **Trace Table View**. The **Capture Settings** dialog box appears.

❖ This option only appears with projects that contain a trace system.

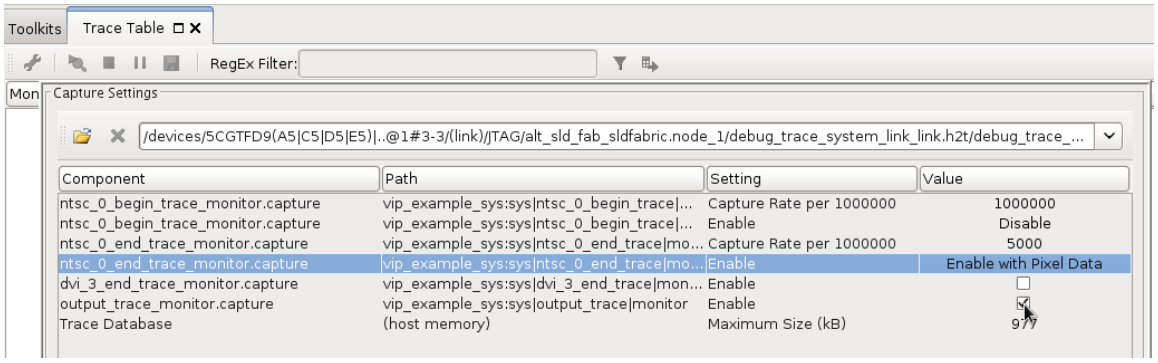
9) In the **Select Hardware** drop-down box select **devices/5CGTFD9(E5| |A5|C5|D5)…**. A table appears that displays options. The first six options are for the monitors that are present in the system. The last option controls how large the database is within memory, which controls how many events System Console can capture at any one time before discarding the oldest captured events.

10) In the **Value** column, specify the following options (Figure 15), and then click **OK**:

- For **ntsc_0_end_trace_monitor -> Capture Rate per 1000000**, change how many pixels the monitor captures from each data packet to 5000 (see *"Avalon-ST Video Monitor Capture Rate Value"*).
- For **ntsc_0_end_trace_monitor -> Enable**, select **Enable with pixel data**, to capture pixels and allow you to view the contents of video data packets.
- Tick the checkbox **output_trace_monitor -> Enable**

  - The two monitor settings are different because **Capture video pixel data** was turned on in the Qsys parameters for the **ntsc_0_begin** monitor but not for the **output_trace_monitor** monitor.

Figure 15 - Monitor Settings



11) Click the **Start** icon to start capturing data. The table fills up with events. The data that System Console sends between IP cores is in packets. The packets have the following general order:

- Video control packet, which details resolution and format.
- Video data packet, which contains the actual video data to display.

12) Click the Pause icon to temporarily halt the current display, which allows you to view a snapshot of the captured data while System Console continues to collect new events in the background.

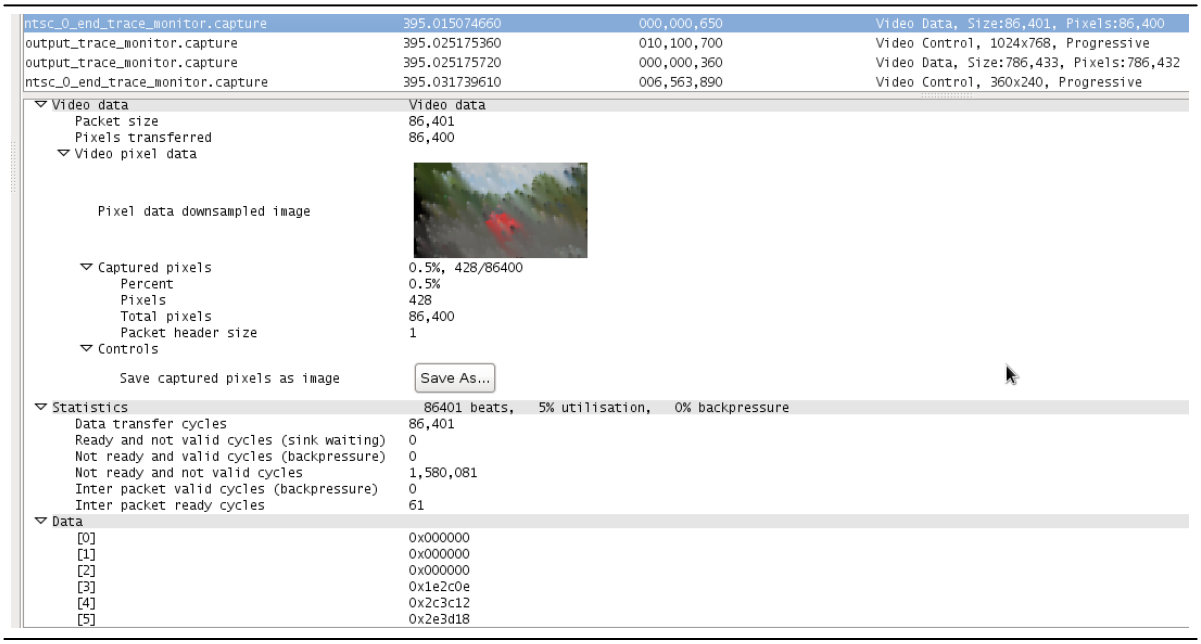
13) Select a video control packet from the table and then expand **Video control**, **Statistics**, and **Data** in the bottom area of the trace table tab. System Console displays high-level decoded information about this packet: the decoded information within the packet (width, height and format data), the statistics about the connection (the number of cycles spent performing certain tasks), and the raw data contained in the packet.

14) Select a video data packet from **ntsc_0_end_trace_monitor** (Figure 16).

a Expand the **Video Data** and **Video pixel data** nodes to show a reconstruction of the captured pixels from this data packet. As the capture rate per frame is typically only a few hundred pixels, the reconstruction algorithm can make the image appear blurred, but it is a good indication of the content of that data packet.

b Expand the **Captured pixels** node to check the pixel capture rate.

Figure 16 - Hierarchical View of the Data Packet



15) Click the Pause icon again to unpause the table, and then click on the Filter Monitors icon. A menu appears that allows you to show and hide monitors, but still have them capture data in the background. Deselect all monitors but **ntsc_0_end_trace_monitor.**

16) Enter `Video Data` in the **RegEx Filter** text box. The filter just shows video data packets from **ntsc_0_end_trace_monitor**.

    ❖ Note: the RegEx filter is case-sensitive.

17) View the captured video pixel data as it arrives into the system.

18) Open the Nios II SBT for Eclipse window next to the System Console window, to see them simultaneously. The Nios II SBT for Eclipse is still paused on the breakpoint.

19) Press **Resume** to move the program one step forward and scale the image on the screen up by an increment. Each time you press **Resume**, the resolution of the packets in System Console Trace Table View increases. The resolution starts at 360x240 and increases by around three pixels in each dimension with every increment of the Nios II program. This action demonstrates the System Console trace system working with the Nios II debugger. It also shows the trace system debugging designs that use the Video and Image Processing Suite IP cores.

6.6 Avalon-ST Video Monitor Capture Rate Value

You can specify the **Capture rate per 1000000** value for every monitor that captures video pixel data.

The value depends on:

- The size of the image
- The number of frames per second at which the video system runs
- The available bandwidth of the debug link

Typically, over a JTAG link and viewing a data path that runs at 60 frames per second, you can capture a few hundred pixels per frame. The value you enter for **Capture rate per 1000000** depends mainly on your image size. For example, for a 25×20 pixel video stream, enter `1000`; for a 1920×1080 pixel video stream, enter `500`. Then run the system and view the output. At any time, you can stop the trace system, adjust the value, and restart the trace system.

If the value is too small, the pixel capture view does not have enough data to create a meaningful picture. If the value is too large, the system does not have not enough bandwidth to send all the captured pixels and it displays a black and purple checkerboard pattern (Figure 17). Adjusting the value allows you to see a representation of the image that the data packet contains.

Figure 17 - Setting Capture Rate Values

7 Applications for the Example Design

Many new video innovations, such as HDTV and digital cinema, involve video and image processing and this technology's rapid evolution. Leaps forward in image capture and display resolutions, advanced compression techniques, and video intelligence are the driving forces behind the technological innovation.

The move from standard definition (SD) to high definition (HD) represents a six times increase in data processing requirements. Video surveillance is also moving from common intermediate format (CIF 352×288) to D1 format (704×576) as a standard requirement, with some industrial cameras moving to HD (1280×720). Military surveillance, medical imaging, and machine vision applications are also moving to very high resolution images. With expanding resolutions, you need high-performance designs while keeping architectures flexible to allow for quick upgradeability.

All Altera Video and Image Processing Suite IP cores are configurable to allow you to satisfy common performance and cost requirements over a wide range of SD and HD resolutions. Starting from the 16.1 release, all series-II Video IP cores can support up to 4K resolution by streaming up to four pixels in parallel. This section describes some of the many video systems that you construct with the Video and Image Processing Suite IP cores.

7.1 Single Video Stream Input

The design example is a good starting point for developing single input or multiple input video stream applications. The design example demonstrates how to construct a video data path that performs deinterlacing, clipping, chroma resampling, finite impulse response (FIR) filtering, gamma correction, color space conversion, mixing and scaling.

The Altera Video and Image Processing IP cores all have common data and control interfaces. These common interfaces, and open protocol for transmitting video, make it easy to extend the data path in the design example to perform additional video processing functions.

The design example also demonstrates how to use a Nios II processor to configure and control the video interface chips, clocked video input, clocked video output, frame buffer, and image mixer. The standardization on Avalon-MM control interface ports makes the Nios II processor a convenient choice for run-time control, enabling a rapid development cycle in a software environment.

You can enable the run-time control interfaces on video processing functions, such as the Scaler and Color Space Converter, to modify their mode of operations. These Avalon-MM slave control ports are enabled from the IP parameterization interfaces. Once Qsys regenerates and the Quartus II software recompiles the hardware system, the IP can be controlled from the software domain.

The Video IP core register maps provide both control and visibility of the data flowing in the data path. For example, you can perform the following actions:

- Perform real-time changes to the size of mixer background and change the output resolution
- Update the scaling coefficients to improve image quality at run-time depending on your scaling ratio.
- Change the Color Space Converter coefficients at run-time to target YCbCr SD or YCbCr HD as required
- Track the number of frames that the frame buffer drops or repeats due to data rate transitions.
- Monitor the number of words in the dual clock FIFO of the Clocked Video Input and Clocked Video Output IP cores, to track overflow, underflow and peak FIFO usage.

7.2 Multiple Video Channel Input

Video systems frequently contain multiple video streams, which process in parallel and are synchronized. Altera's Video and Image Processing IP cores are suitable for constructing video systems with small numbers of channels (such as converter boxes) up to systems with hundreds of video streams (such as switchers and multi viewers).

Use the Cyclone V Video Development Kit to easily extend the design example. It can support four composite or S-Video video stream inputs via the Bitec HSMC Quad Video daughter card, and the DVI input on the Bitec HSMC DVI daughter card.

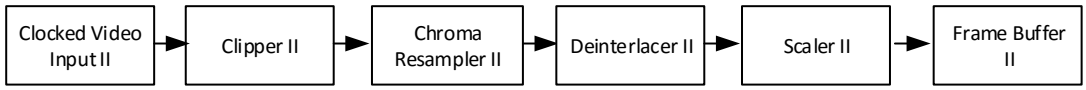
The Nios II software allows you to configure the TVP5154 chip on the Bitec HSMC Quad Video daughter card for four composite inputs.

To extend the design example to support additional composite input video streams, follow these steps:

1) In Qsys, create an additional video processing data pipeline to input into the mixer, with the processing functions (Figure 18).

   ❖ You must ensure that the video functions process data at the required rate to avoid overflow at the input of the system and underflow at the output. When adding functions to a system that connect to external memory, such as a deinterlacer, ensure that there is sufficient external memory bandwidth available.

Figure 18 - Qsys Processing Functions



2) Increase the number of inputs to the Alpha Blending Mixer.
3) Connect the new data pipeline to the new mixer input.
4) Regenerate the Qsys system.
5) Export the new Qsys system I/O in the top-level Quartus II **top\vip_top.v** file, add the appropriate Bitec Quad Video card input pins (the project already contains the correct

assignments), connect the exported wires from the Qsys system to the new input pins, save the **vip_top.v** file, and recompile the Quartus II project.

6) Update the software project to initialize and control the mixer with five layers.

Systems with multiple input streams raise many challenges, such as achieving the required level of external memory bandwidth, often with a shared DDR external memory; synchronizing multiple streams of video at different data rates; or processing at HD data rates. Altera is continuously developing the framework to simplify such system and design challenges.

7.3 Other Video Interface Standards

The design example allows you to develop video systems with NTSC, PAL or DVI video inputs and DVI video output. This design example requires appropriate hardware ports on the video development board, analog-to-digital conversion and FPGA hardware interface components.

You see a variety of different hardware interface standards and protocols in applications throughout the broadcast, consumer, automotive, surveillance, medical, and document imaging markets. These standards include High Definition Multimedia Interface (HDMI), component, S-Video or Y/C (luminance and color signals), SDI, high definition SDI (HD-SDI) and 3GBit/s SDI (3G SDI).

Programmable FPGAs are ideal for migrating systems that support different interfaces. You can use the design example as a framework for development of such FPGA hardware interface components. You can reparameterize the Clocked Video Input and Clocked Video Output IP cores in the Qsys system to match the video format and adapting the top-level design with the appropriate video interface and pin assignments.

For example, you can develop video systems with SDI video input and output ports that use the Stratix® II GX board and the Altera SDI/HD-SDI IP core by applying the same video framework methodology that this design example uses.

8 Conclusion

The Video and Image Processing Design Example demonstrates a reusable and flexible video framework for rapid development of video and image processing designs. The design uses standard open interfaces and protocols throughout the system. You can reuse parametrizable IP cores from the Altera MegaCore IP Library (which includes the Video and Image Processing Suite) or add your own IP to the framework to build further applications.

The design implements system control code in software that runs on a Nios II processor, which provides a rapid edit debug cycle. A C++ class based software API provides an interface to the Video and Image Processing Suite IP cores from the Nios II control software to ease software development.

The video framework does not preclude using HDL to connect the IP components. However, the design example demonstrates that the Qsys environment significantly accelerates system design by:

- Providing automatic generation of an application-specific switch fabric including arbitration scheme.
- Providing an abstracted view of the video system.
- Detecting and displaying Altera and user IP in an immediately accessible form.

9 Revision History

Table 4 - Reference Design Document Revision History shows the revision history for the "Video and Image Processing Design Example" reference design document.

Table 4 - Reference Design Document Revision History

Version	Date	Change Summary
1.0	Sep. 2015	▪ First release of this document

1.1	Sep. 2016	▪ Use Quartus II software version 16.0
		▪ Use the following Version II IP cores in Quartus II software version 16.0
		  ▪ Clocked Video Input II
		  ▪ Clocked Video output II
		  ▪ Color Space Converter II
		  ▪ Deinterlacer II
		  ▪ Frame Buffer II
		  ▪ Scaler II
		  ▪ Clipper II

1.2	Dec. 2016	▪ Update to Quartus II software version 16.1
		▪ Switch all legacy Video IP cores to version II
		  ▪ Mixer II
		  ▪ Chroma Resampler II
		  ▪ Fir Filter II
		  ▪ Gamma Corrector II
		  ▪ Color Plane Sequencer II
		▪ Remove Test Pattern Generator and reorganize the mixer layers since the background generation is now embedded in Mixer II and the alpha channel is expected with the video stream