Overview

A critically important aspect of debugging video designs is to ensure that the video stream is of the proper frame size and that it adheres to the given interface specification (which, for most Xilinx Video IP, is the Video over AXI4 Stream protocol).

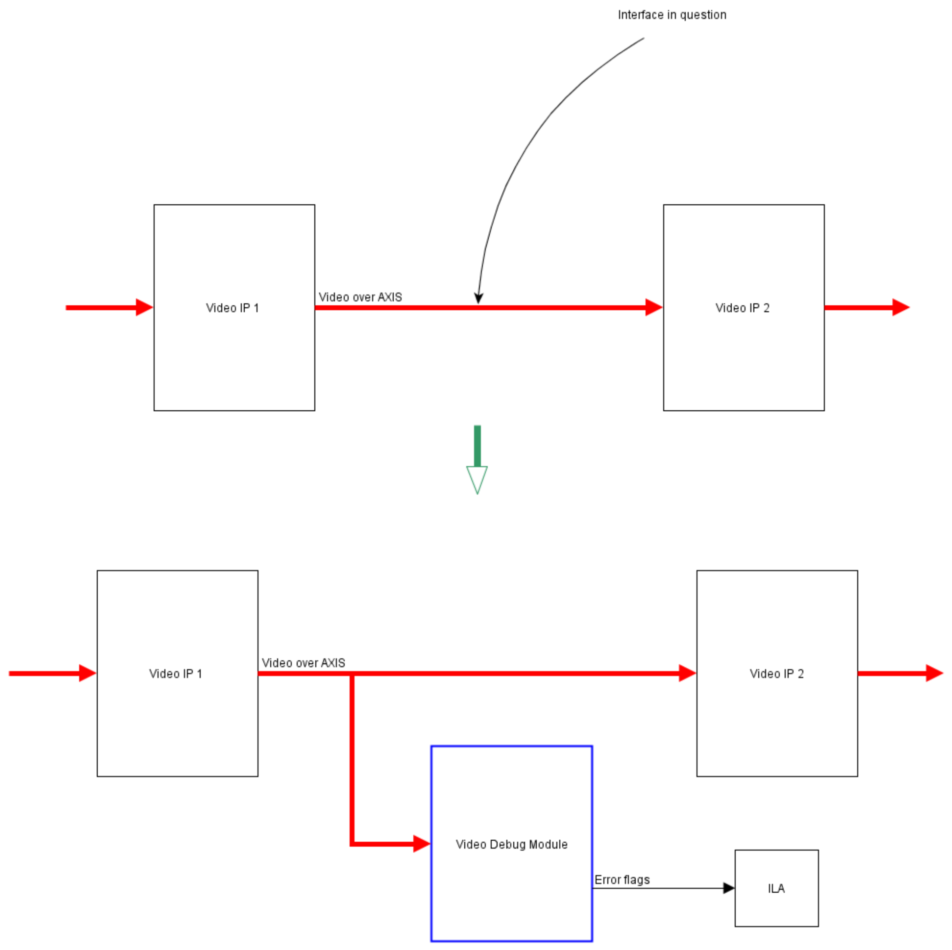
The Video Debug Module provided with this article will help to ensure that a given video stream is well formed and does not contain any frame size errors. Features of the Video Debug Module include:

- *Frame, Line, and Pixel counters*
- *Frame size error checking*
- *Storage of 1024 previous frame sizes and error conditions (accessible via TCL or C API)*
- *Programmable flag for triggering tools such as Vivado Logic Analyzer*
- *Optional AXI4-Lite or JTAG interface*
- *Easy-to-use software/TCL API for interacting with the core*

Using the Core

The Video Debug Module is provided as a packaged Vivado IP core which can be used like any other IP core in Vivado and/or IP Integrator. Simply extract the attached zip file and add the parent directory to the IP Repository in any Vivado Project settings. Then the Video Debug Module will show up in your IP Catalog for use in your design.

The core contains an *s_axis* interface which is a packaged as a 'monitor' type interface so that it can be attached to an existing AXIS bus in an IPI design. Therefore, the core should be placed in the system as depicted here:



When instantiating the Video Debug Module, it should be configured for the expected frame size (Vsize, Hsize, and Tdata Width parameters). If the frame size is configured incorrectly at generation time or if the frame size changes at run time, the counters may still function properly but the error flags will be invalid. In such cases, the frame size may be set during run time through the AXI4-Lite interface.

The core has the ability to optionally include the Frame Size Logger submodule which keeps a history of 1024 frame sizes and errors. Including this feature costs 1 additional RAMB36 as well as a small amount of additional logic resources.

Note that the Frame Size Logger submodule logs the last pixel count of the frame, so you should be cautious when interpreting the data and an error is indicated. For example, if the core is configured for 1920x1080 frame size and the first line of the first frame in the video stream is only 1000 pixels long (but all other lines are correctly 1080 pixels long), the Frame Size Logger will indicate that the frame was 1920x1080 in size, but that an EOL Early error was asserted. In this case, you should insert Vivado Logic Analyzer to determine why the error occurred.

The user can also configure a programmable flag output signal which asserts combinatorially when the frame, line, and pixel counters reach the desired values. This can be useful for triggering other debug tools such as Vivado Logic Analyzer at specific points in the video stream.

The Video Debug Module has several additional outputs including Error Flags, Error Latches, frame/line/pixel counts, and a flag indicating that the Frame Size Logger is full. The typical use case will be to connect any or all of these signals to an ILA core in Vivado and then view them using the Analyzer tool. Alternatively, you could rely on the frame size logger inside the Video Debug Module and access frame sizes and errors as described later.

There are two sets of output signals for each type of frame size error. The signals with *_flag suffix are asserted combinatorially with errors (with 1-2 cycles of latency) and they will deassert when the error condition is resolved. By contrast, signals with the *_latch suffix are asserted at the first error of that type and persist until the core is reset.

The core traps four types of frame size errors:

- End of Line Early (err_eol_early_* signals) - Occurs when tlast asserts *before* the pixel counter reaches the configured Input Hsize parameter
- End of Line Late (err_eol_late_* signals) - Occurs when tlast asserts *after* the pixel counter reaches the configured Input Hsize parameter
- Start of Frame Early (err_sof_early_* signals) - Occurs when tuser asserts *before* the line counter reaches the configured Input Vsize parameter
- Start of Frame Late (err_sof_late_* signals) - Occurs when tuser asserts *after* the line counter reaches the configured Input Hsize parameter

Additionally, the err_* signals are the logical 'OR' of all four frame errors of that type (latch or flag). The err_* signals assert if any other type of error is detected.

The core provides one of three interfacing options:

- *None* - The core does not log frame size history data, nor does it allow you to control it dynamically. It will log errors and count frame size in real time. As such, you should monitor the output signals directly when using this mode.
- *JTAG* - The core can be controlled via JTAG and frame size history data may optionally be logged and accessed. When JTAG mode is used, several TCL API functions are exposed which may be used for accessing core data and controlling the core at run-time
- *AXI4-Lite* - For processor-based systems, an AXI4-Lite interface may be enabled. This interface allows the same functionality of JTAG option, but can be accessed directly via AXI4-Lite. When AXI4-Lite mode is used, several C API functions are exposed which may be used for accessing core data and controlling the core at run time.

Configuring the Core

- *Which Interface*
  - *None* – No external interface is exposed
  - *JTAG* – JTAG interface is exposed
  - *AXI4-Lite* – AXI4-Lite interface is exposed

- *Include Frame Size Logger (only available when JTAG or AXI4-Lite Interface is selected)*
    - *True* – Frame Size Logger submodule is included
    - *False* – Frame Size Logger submodule is not included
- *Init Vsize* – Initial number of active lines per frame to check against. This value is restored when the core is reset.
- *Init Hsize* – Initial number of active pixels per line to check against. This value is restored when the core is reset
- *Max Vsize* – Maximum number of active lines per frame to check against. This value sets the width of the line_cnt signal
- *Max Hsize* – Maximum number of active pixels per line to check against. This value sets the width of the pixel_cnt signal
- *Tdata Width* – Width of the s_axis_tdata signal
- *Pixel To Trigger** – Which pixel in the line the programmable flag should trigger
- *Line To Trigger** – Which pixel in the line the programmable flag should trigger
- *Frame To Trigger** – Which pixel in the line the programmable flag should trigger

* The programmable flag signal (prog_flag) only asserts when Pixel To Trigger, Line To Trigger, and Frame To Trigger are all matched by their respective counters.

Register Space

Address Offset	Name	Reset Value	Access	Description
0x00	Control	0x00000000	R/W	Bit 0: Software Reset Bit 1: Clear Error Flags
0x04	Status	0x00000002	RO	Bit 0: Frame Size History FIFO full Bit 1: Frame Size History FIFO empty
0x08	Error	0x00000000	RO	Bit 0: err_latch signal is asserted Bit 1: err_eol_early_latch signal is asserted Bit 2: err_eol_late_latch signal is asserted Bit 3: err_sof_early_latch signal is asserted Bit 4: err_sof_late_latch signal is asserted
0x0C	Horizontal Size	Init Vsize	R/W	Number of pixels per line to check against
0x10	Vertical Size	Init Hsize	R/W	Number of lines per frame to check against
0x14	Frame Size History	0x00000000	RO	Frame Size History data FIFO

0x00 – Control Register

- *Bit 0: Software Reset* – Reset the entire core by writing a ‘1’ to this bit. This will cause all errors to be cleared, the Frame Size History FIFO to be flushed, and the error signals to be checked against Init Vsize and Init Hsize. This bit will be cleared automatically upon the start of the first frame after the completion of reset.
- *Bit 1: Clear Error Flags* – Writing a ‘1’ to this bit causes all err_*_latch signals to be cleared and all errors in the Error Register (0x08) to be cleared. The user must manually de-assert this bit.

0x04 – Status Register

- *Bit 0: Frame Size History FIFO full* – This read-only bit indicates that the Frame Size History FIFO is full and ready for reading
- *Bit 1: Frame Size History FIFO empty* – This read-only bit indicates that the Frame Size History FIFO is empty

0x08 – Error Register

- *Bit 0: err_latch asserted* – This read-only bit indicates that the err_latch signal is presently asserted
- *Bit 1: err_eol_early_latch asserted* – This read-only bit indicates that the err_eol_early_latch signal is presently asserted
- *Bit 2: err_eol_late_latch asserted* – This read-only bit indicates that the err_eol_late_latch signal is presently asserted
- *Bit 3: err_sof_early_latch asserted* – This read-only bit indicates that the err_sof_early_latch signal is presently asserted
- *Bit 4: err_sof_late_latch asserted* – This read-only bit indicates that the err_sof_late_latch signal is presently asserted

0x0C – Horizontal Size Register

- *Bits 31:0: Horizontal Size* – Number of pixels per line that the core uses to check for errors. While the register is 32 bits wide, undefined results occur if this register is set to a value greater than the *Max Hsize* parameter. After setting this register, it is recommended that you clear the error flags (by writing to bit 1 of the control register) to avoid confusion in the case that flags were already set prior to changing this register. Note that the Frame Size History FIFO will not be cleared. Therefore, you should flush it manually by reading its entire contents to avoid using stale data.

0x10 – Vertical Size Register

- *Bits 31:0: Vertical Size* – Number of lines per frame that the core uses to check for errors. While the register is 32 bits wide, undefined results occur if this register is set to a value greater than the *Max Vsize* parameter. After setting this register, it is recommended that you clear the error flags (by writing to bit 1 of the control register) to avoid confusion in the case that flags were already set prior to changing this register. Note that the Frame Size History FIFO will not be cleared. Therefore, you should flush it manually by reading its entire contents to avoid using stale data.

0x14 – Frame Size History Register

- *Bits 31:0: Frame Size History data FIFO* – This read-only register provides access to the Frame Size History submodule’s stored data. Use the appropriate API for formatting details

Software API

```
// Soft reset of the video debug module. This will reset the entire core which
// will cause it to re-lock on the video stream and re-populate the fifo.
int rst_vid_dbg_module(unsigned int baseaddr);

// Read the entire frame size history buffer.
int read_frame_size_history(unsigned int baseaddr);

// Get current status of error latches.
int get_cur_errs(unsigned int baseaddr);

// Clear flags and frame size history FIFO, but don't reset VSIZE and HSIZE.
int clear_flags(unsigned int baseaddr);

// Get horizontal size.
int get_hsize(unsigned int baseaddr);

// Set horizontal size.
int set_hsize(unsigned int baseaddr, unsigned int new_hsize);

// Get vertical size.
int get_vsize(unsigned int baseaddr);

// Set vertical size.
int set_vsize(unsigned int baseaddr, unsigned int new_vsize);
```

TCL API

To use the TCL API, first source the vid_dbg_cmds.tcl file from a Vivado project. Then, when running a Hardware Manager, you can run these commands.

```
#
# Soft reset of the video debug module. This will reset the entire core which
# will cause it to re-lock on the video stream and re-populate the fifo.
#
# Example: rst_vid_dbg_module [get_hw_axis hw_axi_1]
#
proc rst_vid_dbg_module {hw_axi_inst}

#
# Read the entire frame size history buffer.
#
# Example: read_frame_size_history [get_hw_axis hw_axi_1]
#
proc read_frame_size_history {hw_axi_inst}

#
# Get current status of error latches.
#
# Example: get_cur_errs [get_hw_axis hw_axi_1]
#
proc get_cur_errs {hw_axi_inst}
```

```
#
# Clear flags and frame size history FIFO, but don't reset VSIZE and HSIZE.
#
# Example: clear_flags [get_hw_axis hw_axi_1]
#
proc clear_flags {hw_axi_inst}

#
# Get horizontal size.
#
# Example: get_hsize [get_hw_axis hw_axi_1]
#
proc get_hsize {hw_axi_inst}

#
# Set horizontal size.
#
# Example: set_hsize [get_hw_axis hw_axi_1] 0x00000780
#
proc set_hsize {hw_axi_inst new_hsize}

#
# Get vertical size.
#
# Example: get_vsize [get_hw_axis hw_axi_1]
#
proc get_vsize {hw_axi_inst}

#
# Set vertical size.
#
# Example: set_vsize [get_hw_axis hw_axi_1] 0x00000438
#
proc set_vsize {hw_axi_inst new_vsize}
```

Miscellaneous

- Null bytes indicated by tkeep are not supported. The core assumes all bytes of each beat are data bytes.
- The core assumes 1 pixel per beat. You may use more than this by sizing *Tdata Width, Init Hsize, and Max Hsize* appropriately.
- The core assumes progressive scan video only. The core can be used to accurately track interlaced frames as long as the number of even fields is the same as the number of odd fields and *Init Vsize* and *Max Vsize* are set appropriately.