LogiCORE IP Test Pattern Generator v4.00a

Product Guide

PG103 December 18, 2012

XILINX®

XILINX

Table of Contents

Appendix D: Application Software Development

Appendix E: Additional Resources

SECTION I: SUMMARY

Introduction

The Xilinx LogiCORE™ IP Test Pattern Generator core generates test patterns for video system bring up, evaluation, and debugging. The core provides a wide variety of tests patterns enabling you to debug and assess video system color, quality, edge, and motion performance. The core can be inserted in an AXI4-Stream video interface that allows user-selectable pass-through of system video signals or insertion of test patterns.

Features

- Color bars
- Zone plate with adjustable sweep and speed
- Temporal and spatial ramps
- Moving box with selectable size and color over any available test pattern
- RGB and YUV 422 support
- Low-footprint, high quality interpolation
- AXI4-Stream data interfaces
- Optional AXI4-Lite control interface
- Supports 8, 10, and 12-bits per color component input and output
- Built-in, optional throughput monitors
- Supports spatial resolutions from 32x32 up to 7680x7680
    - Supports 1080P60 in all supported device families [1]
    - Supports 4kx2k @ 24 Hz in supported high performance devices

1. Performance on low power devices may be lower.

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Zynq™-7000 [2], Artix™-7, Virtex®-7, Kintex™-7
Supported User Interfaces	AXI4-Lite, AXI4-Stream[3]
Resources	See Table 2-1 through Table 2-6.
Provided with Core	
Documentation	Product Guide
Design Files	VHDL
Example Design	Not Provided
Test Bench	Test Bench available with Vivado only.
Constraints File	Not Provided
Simulation Models	VHDL or Verilog Structural
Supported Software Drivers	Not Applicable
Tested Design Flows [5]	
Design Entry Tools	Vivado™ Design Suite 2012.4 [6], Xilinx Platform Studio (XPS)
Simulation[4]	Mentor Graphics ModelSim, Xilinx® ISim, Vivado Simulator
Synthesis Tools	Xilinx Synthesis Technology (XST) Vivado Synthesis
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.
2. Supported in Xilinx Platform Studio implementations only.
3. Video protocol as defined in the *Video IP: AXI Feature Adoption* section o (UG761) *AXI Reference Guide* [Ref 1].
4. For the supported versions of the tools, see the ISE Design Suite 14: Release Notes Guide.
5. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
6. Supports only 7 series devices.

Overview

The Test Pattern Generator core generates video test patterns which can be used when developing video processing cores or bringing up a video system. The test patterns can be used to evaluate and debug color, quality, edge, and motion performance, debug and assess video system color, quality, edge, and motion performance of a system, or stress the video processing to ensure proper functionality.

Feature Summary

The Test Pattern Generator core produces the following patterns in RGB or YCbCr 422 video format:

- Video input pass through
- Horizontal ramp
- Vertical ramp
- Temporal ramp
- Flat fields (red, green, blue, black and white)
- Color bars
- Tartan bars
- Zone plate
- Cross hairs
- Cross hatch
- Solid box
- Motion effect for ramps, zone plate, and solid box

Applications

The horizontal and vertical ramps can be used to test system linearity. The temporal ramp can be used to test the integrity of frame buffers that exist in the system. Flat fields are useful for detecting possible chroma issues with video processing cores. Color bars and tartan bars are used to verify the proper reproduction of color correction functions and the color gamut of a monitor.

Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite and ISE Design Suite tools under the terms of the Xilinx End User License. Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

Product Specification

Standards

The Test Pattern Generator core is compliant with the AXI4-Stream Video Protocol and AXI4-Lite interconnect standards. Refer to the *Video IP: AXI Feature Adoption* section of the *AXI Reference Guide* (UG761) [Ref 1] for additional information.

Performance

The following sections detail the performance characteristics of the Test Pattern Generator core.

Maximum Frequencies

The following are typical clock frequencies for the target devices. The maximum achievable clock frequency can vary. The maximum achievable clock frequency and all resource counts can be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools and other factors. Refer to in Table 2-1 through Table 2-6 for device-specific information.

Throughput

The Test Pattern Generator core supports bidirectional data throttling between its AXI4-Stream Slave and Master interfaces. If the slave side data source is not providing valid data samples (`s_axis_video_tvalid` is not asserted), the core cannot produce valid output samples after its internal buffers are depleted. Similarly, if the master side interface is not ready to accept valid data samples (`m_axis_video_tready` is not asserted) the core cannot accept valid input samples once its buffers become full.

If the master interface is able to provide valid samples (`s_axis_video_tvalid` is high) and the slave interface is ready to accept valid samples (`m_axis_video_tready` is high), typically the core can process one sample and produce one pixel per `ACLK` cycle.

However, at the end of each scan line and frame the core flushes internal pipelines for 7 clock cycles, during which the `s_axis_video_tready` is de-asserted signaling that the core is not ready to process samples.

When the core is processing timed streaming video (which is typical for most video sources), the flushing periods coincide with the blanking periods therefore do not reduce the throughput of the system.

When the core is processing data from a video source which can always provide valid data, e.g. a frame buffer, the throughput of the core can be defined as follows:

$$R_{MAX} = f_{ACLK} \times \frac{ROWS}{ROWS} \times \frac{COLS}{COLS + 7} \quad \text{Equation 2-1}$$

In numeric terms, 1080P/60 represents an average data rate of 124.4 MPixels/second (1080 rows x 1920 columns x 60 frames / second), and a burst data rate of 148.5 MPixels/sec.

To ensure that the core can process 124.4 MPixels/second, it needs to operate minimally at:

$$f_{ACLK} = R_{MAX} \times \frac{ROWS}{ROWS} \times \frac{COLS + 7}{COLS} = 124.4 \times \frac{1080}{1080} \times \frac{1927}{1920} = 124.8 \quad \text{Equation 2-2}$$

When operating on a streaming video source (i.e. not frame buffered data), the core must operate minimally at the burst data rate, for example, 148.5MHz for a 1080P60 video source.

Resource Utilization

For an accurate measure of the usage of primitives, slices, and CLBs for a particular instance, check the **Display Core Viewer after Generation**.

Resource Utilization using ISE Design Suite

The information presented in Table 2-1 through Table 2-3 is a guide to the resource utilization and maximum clock frequency of the Test Pattern Generator core for all input/output width combinations for Zynq-7000, Artix-7, Virtex-7, and Kintex-7 FPGA families using EDK tools v14.4. The Xtreme DSP count is 3 when the AXI4-Lite interface is included, and can be up to 3 in constant mode depending on the test pattern chosen. This core does not use any dedicated I/O or CLK resources. The design was tested using Xilinx design tools with default tool options for characterization data. By default, the number of rows was set to 1080, active pixels per scan line was set to 1920. The design was tested with and without the AXI4-Lite interface. When the AXI4-Lite interface was disabled, the Zone Plate test pattern was enabled.

Table 2-1: **Kintex-7**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48	Fmax (MHz)
With AXI Interfaces						
8	2100	1662	1749	0 / 1	3	173
10	2205	1734	1804	0 / 1	3	174
12	2272	1778	1865	0 / 1	3	172
Without AXI Interfaces						
8	251	233	142	0 / 1	0	251
10	251	233	143	0 / 1	0	204
12	250	233	143	0 / 1	0	237

1. Speedfile: XKC70T-1 ADVANCED 1.07c 2012-09-11

Table 2-2: **Artix-7**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48	Fmax (MHz)
With AXI Interfaces						
8	2008	1606	1748	0 / 1	3	150
10	2156	1622	1807	0 / 1	3	151
12	2204	1728	1868	0 / 1	3	153
Without AXI Interfaces						
8	255	236	143	0 / 1	0	166
10	254	236	144	0 / 1	0	174
12	250	240	144	0 / 1	0	149

1. Speedfile: XC7A100T-1 FGG484 ADVANCED 1.05d 2012-09-11

Table 2-3: **Virtex-7**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48	Fmax (MHz)
With AXI Interfaces						
8	2085	1696	1749	0 / 1	3	164
10	2236	1728	1804	0 / 1	3	163
12	2289	1785	1865	0 / 1	3	159
Without AXI Interfaces						
8	250	233	142	0 / 1	0	202
10	249	233	143	0 / 1	0	210
12	251	233	143	0 / 1	0	229

1. Speedfile: XC7V585T-1 FFG1157 ADVANCED 1.07c 2012-09-11

Resource Utilization using Vivado Design Suite

Table 2-4 through Table 2-6 were generated using Vivado Design Suite 2012.4 with default tool options for characterization data.

Table 2-4: **Kintex-7**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48	Fmax (MHz)
With AXI Interfaces						
8	2028	1525	1867	0 / 1	3	194
10	2044	1578	1940	0 / 1	3	188
12	2159	1605	2016	0 / 1	3	207
Without AXI Interfaces						
8	210	198	146	0 / 1	0	168
10	212	201	152	0 / 1	0	169
12	216	205	158	0 / 1	0	169

1. Speedfile: XKC70T-1 ADVANCED 1.07c 2012-09-21

Table 2-5: **Artix-7**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48	Fmax (MHz)
With AXI Interfaces						
8	2042	1525	1867	0 / 1	3	169
10	2111	1583	1940	0 / 1	3	163
12	2168	1606	2016	0 / 1	3	153
Without AXI Interfaces						
8	206	195	146	0 / 1	0	170
10	212	201	152	0 / 1	0	169
12	217	205	158	0 / 1	0	168

1. Speedfile: XC7A100T-1 FGG484 ADVANCED 1.05g 2012-09-21

Table 2-6: **Virtex-7**

Data Width	LUT-FF Pairs	LUTs	FFs	RAM 36 / 18	DSP48	Fmax (MHz)
With AXI Interfaces						
8	2009	1525	1867	0 / 1	3	188
10	2120	1580	1940	0 / 1	3	208
12	2178	1603	2016	0 / 1	3	202
Without AXI Interfaces						
8	209	198	146	0 / 1	0	169
10	212	201	152	0 / 1	0	169
12	216	205	158	0 / 1	0	168

1. Speedfile: XC7V585T-1 FFG1157 ADVANCED 1.07e 2012-09-22

Core Interfaces and Register Space

Port Descriptions

The Test Pattern Generator core uses industry standard control and data interfaces to connect to other system components. The following sections describe the various interfaces available with the core. Figure 2-1 illustrates an I/O diagram of the TPG core. Some signals are optional and not present for all configurations of the core. The AXI4-Lite interface and the `IRQ` pin are present only when the core is configured through the GUI with an AXI4-Lite control interface. The `INTC_IF` interface is present only when the core is configured through the GUI with the INTC interface enabled.

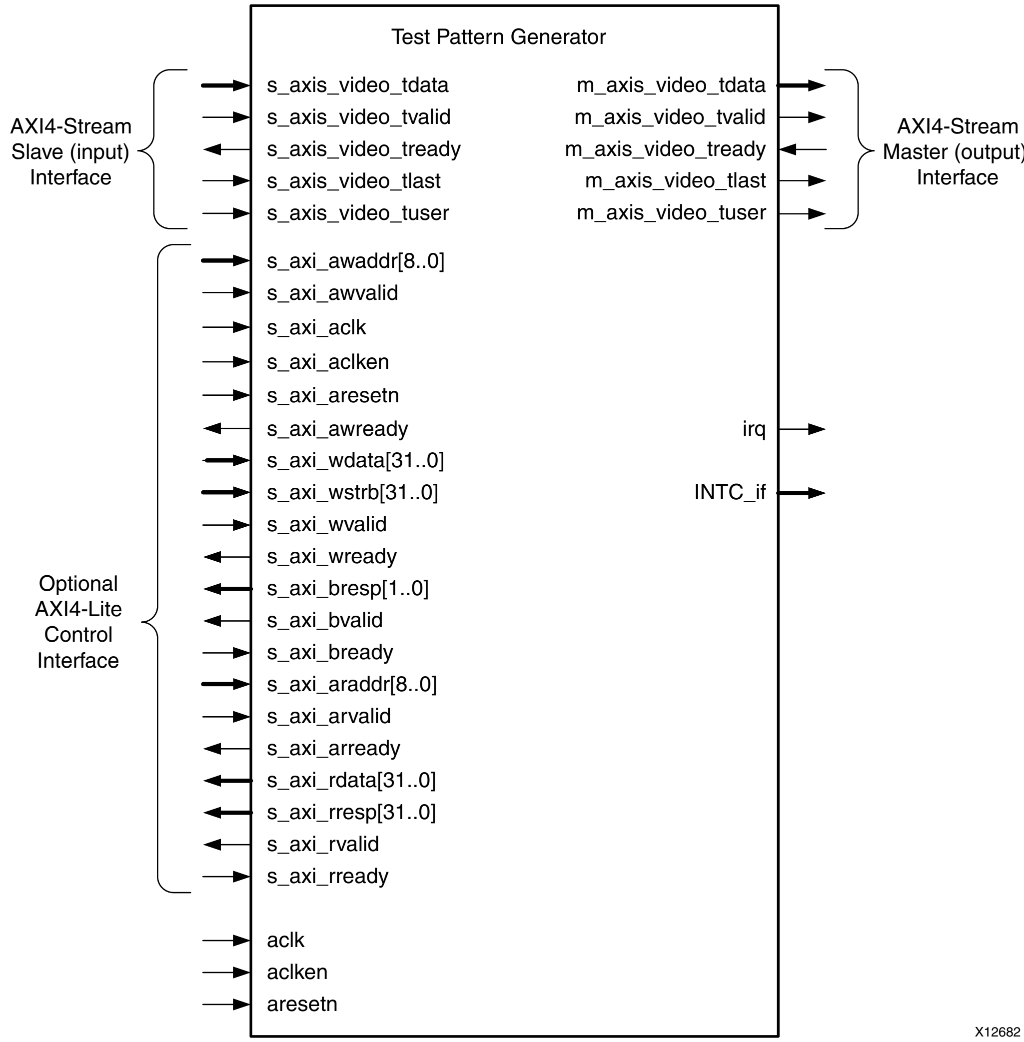

Figure 2-1: **TPG Core Top-Level Signaling Interface**

Common Interface Signals

Table 2-7 summarizes the signals which are either shared by, or not part of the dedicated AXI4-Stream data or AXI4-Lite control interfaces.

Table 2-7: **Common Interface Signals**

Signal Name	Direction	Width	Description
ACLK	In	1	Video Core Clock
ACLKEN	In	1	Video Core Active High Clock Enable
ARESETn	In	1	Video Core Active Low Synchronous Reset
INTC_IF	Out	6	Optional External Interrupt Controller Interface. Available only when `INTC_IF` is selected on GUI.
IRQ	Out	1	Optional Interrupt Request Pin. Available only when AXI4-Lite interface is selected on GUI.

The ACLK, ACLKEN and ARESETn signals are shared between the core, the AXI4-Stream data interfaces, and the AXI4-Lite control interface. Refer to The Interrupt Subsystem for a detailed description of the INTC_IF and IRQ pins.

ACLK

The AXI4-Stream interface must be synchronous to the core clock signal ACLK. All AXI4-Stream interface input signals are sampled on the rising edge of ACLK. All AXI4-Stream output signal changes occur after the rising edge of ACLK.

ACLKEN

The ACLKEN pin is an active-high, synchronous clock-enable input pertaining to AXI4-Stream interfaces. Setting ACLKEN low (de-asserted) halts the operation of the core despite rising edges on the ACLK pin. Internal states are maintained, and output signal levels are held until ACLKEN is asserted again. When ACLKEN is de-asserted, core inputs are not sampled, except ARESETn, which supersedes ACLKEN.

ARESETn

The ARESETn pin is an active-low, synchronous reset input pertaining to only AXI4-Stream interfaces. ARESETn supersedes ACLKEN, and when set to 0, the core resets at the next rising edge of ACLK even if ACLKEN is de-asserted.

Data Interface

The TPG core receives and transmits data using AXI4-Stream interfaces that implement a video protocol as defined in the *Video IP: AXI Feature Adoption* section of the (UG761) *AXI Reference Guide* [Ref 1].

AXI4-Stream Signal Names and Descriptions

Table 2-8 describes the AXI4-Stream signal names and descriptions.

Table 2-8: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
s_axis_video_tdata	In	16,24,32,40	Input Video Data
s_axis_video_tvalid	In	1	Input Video Valid Signal
s_axis_video_tready	Out	1	Input Ready
s_axis_video_tuser	In	1	Input Video Start Of Frame
s_axis_video_tlast	In	1	Input Video End Of Line
m_axis_video_tdata	Out	16,24,32,40	Output Video Data
m_axis_video_tvalid	Out	1	Output Valid

Table 2-8: **AXI4-Stream Data Interface Signal Descriptions**

Signal Name	Direction	Width	Description
m_axis_video_tready	In	1	Output Ready
m_axis_video_tuser	Out	1	Output Video Start Of Frame
m_axis_video_tlast	Out	1	Output Video End Of Line

Video Data

The AXI4-Stream interface specification restricts TDATA widths to integer multiples of 8 bits. Therefore, 10 and 12 bit data must be padded with zeros on the MSB to form a N*8 bit wide vector before connecting to s_axis_video_tdata. Padding does not affect the size of the core.

Similarly, data on the TPG output m_axis_video_tdata is packed and padded to multiples of 8 bits as necessary, as seen in Figure 2-2. Zero padding the most significant bits is only necessary for 10 and 12 bit wide data.

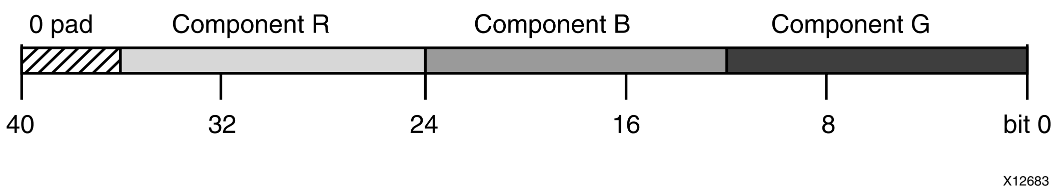

Figure 2-2: **12-bit RGB Data Encoding on m_axis_video_tdata**

READY/VALID Handshake

A valid transfer occurs whenever READY, VALID, ACLKEN, and ARESETn are high at the rising edge of ACLK, as seen in Figure 2-3. During valid transfers, DATA only carries active video data. Blank periods and ancillary data packets are not transferred through the AXI4-Stream video protocol.

Guidelines on Driving s_axis_video_tvalid

Once s_axis_video_tvalid is asserted, no interface signals (except the TPG core driving s_axis_video_tready) may change value until the transaction completes (s_axis_video_tready, s_axis_video_tvalid, and ACLKEN are high on the rising edge of ACLK). Once asserted, s_axis_video_tvalid may only be de-asserted after a transaction has completed. Transactions may not be retracted or aborted. In any cycle following a transaction, s_axis_video_tvalid can either be de-asserted or remain asserted to initiate a new transfer.

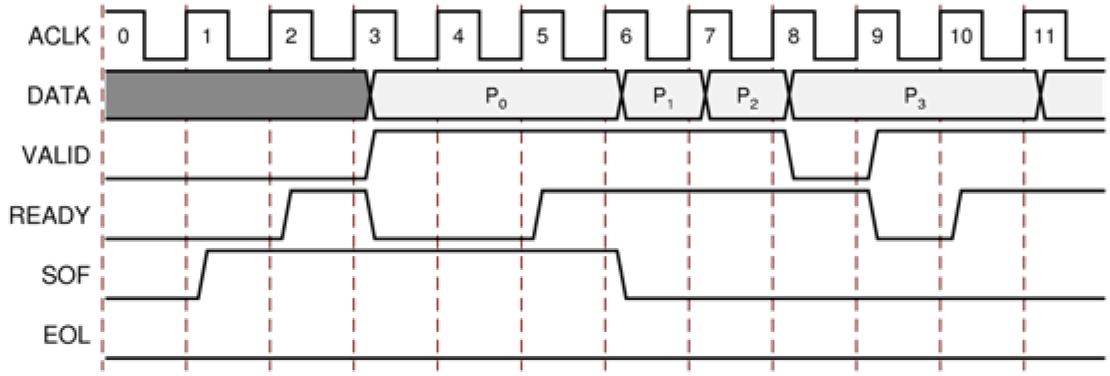

Figure 2-3: **Example of READY/VALID Handshake, Start of a New Frame**

Guidelines on Driving m_axis_video_tready

The `m_axis_video_tready` signal may be asserted before, during or after the cycle in which the TPG core asserted `m_axis_video_tvalid`. The assertion of m_axis_video_tready may be dependent on the value of `m_axis_video_tvalid`. A slave that can immediately accept data qualified by `m_axis_video_tvalid`, should pre-assert its `m_axis_video_tready` signal until data is received. Alternatively, `m_axis_video_tready` can be registered and driven the cycle following `VALID` assertion.

RECOMMENDED: *To minimize latency, your custom core's slave interface should drive READY independently, or pre-assert READY.*

Start of Frame Signals - m_axis_video_tuser0, s_axis_video_tuser0

The Start-Of-Frame (`SOF`) signal, physically transmitted over the AXI4-Stream `TUSER0` signal, marks the first pixel of a video frame. The SOF pulse is 1 valid transaction wide, and must coincide with the first pixel of the frame, as seen in Figure 2-3. The `SOF` signal serves as a frame synchronization signal, which allows downstream cores to re-initialize, and detect the first pixel of a frame. The `SOF` signal may be asserted an arbitrary number of `ACLK` cycles before the first pixel value is presented on `DATA`, as long as a `VALID` is not asserted.

End of Line Signals - m_axis_video_tlast, s_axis_video_tlast

The End-Of-Line (`EOL`) signal, physically transmitted over the AXI4-Stream `TLAST` signal, marks the last pixel of a line. The `EOL` pulse is 1 valid transaction wide, and must coincide with the last pixel of a scan-line, as seen in Figure 2-4.

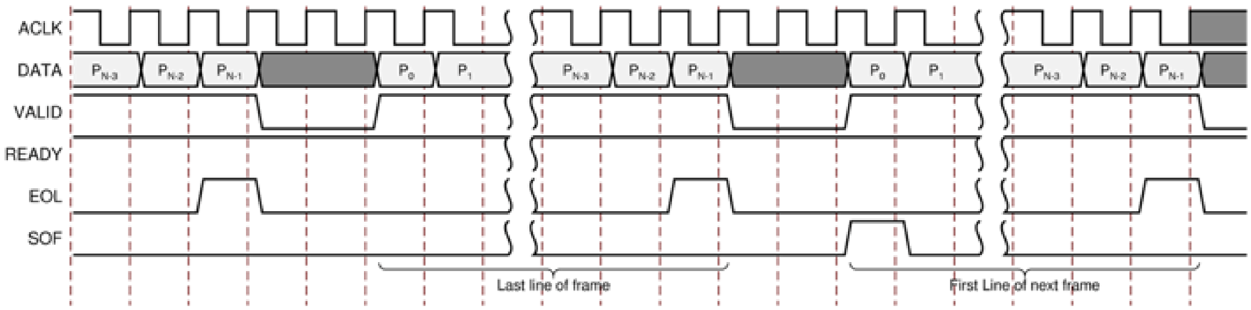

Figure 2-4: **Use of EOL and SOF Signals**

Control Interface

When configuring the core, you can add an AXI4-Lite register interface to dynamically control the behavior of the core. The AXI4-Lite slave interface facilitates integrating the core into a processor system, or along with other video or AXI4-Lite compliant IP, connected through AXI4-Lite interface to an AXI4-Lite master. In a static configuration with a fixed set of parameters (constant configuration), the core can be instantiated without the AXI4-Lite control interface, which reduces the core Slice footprint.

Constant Configuration

The constant configuration caters to users who will use the core in one setup that will not need to change over time. In constant configuration the image resolution (number of active pixels per scan line and the number of active scan lines per frame), and the test pattern is hard coded into the core via the TPG core GUI. Since there is no AXI4-Lite interface, the core is not programmable, but can be reset, enabled, or disabled using the `ARESETn` and `ACLKEN` ports. In constant configuration mode, the Test Pattern Generator produces a single, pre-determined test pattern. In this mode, the AXI4-Stream slave interface is removed so a video source for the core is not necessary.

AXI4-Lite Interface

The AXI4-Lite interface allows you to dynamically control parameters within the core. Core configuration can be accomplished using an AXI4-Lite master state machine, or an embedded ARM or soft system processor such as MicroBlaze.

The TPG core can be controlled through the AXI4-Lite interface using read and write transactions to the TPG register space.

Table 2-9: **AXI4-Lite Interface Signals**

Signal Name	Direction	Width	Description
s_axi_aclk	In	1	AXI4-Lite clock
s_axi_aclken	In	1	AXI4-Lite clock enable

Table 2-9: **AXI4-Lite Interface Signals *(Cont'd)***

Signal Name	Direction	Width	Description
s_axi_aresetn	In	1	AXI4-Lite synchronous Active Low reset
s_axi_awvalid	In	1	AXI4-Lite Write Address Channel Write Address Valid.
s_axi_awread	Out	1	AXI4-Lite Write Address Channel Write Address Ready. Indicates DMA ready to accept the write address.
s_axi_awaddr	In	32	AXI4-Lite Write Address Bus
s_axi_wvalid	In	1	AXI4-Lite Write Data Channel Write Data Valid.
s_axi_wready	Out	1	AXI4-Lite Write Data Channel Write Data Ready. Indicates DMA is ready to accept the write data.
s_axi_wdata	In	32	AXI4-Lite Write Data Bus
s_axi_bresp	Out	2	AXI4-Lite Write Response Channel. Indicates results of the write transfer.
s_axi_bvalid	Out	1	AXI4-Lite Write Response Channel Response Valid. Indicates response is valid.
s_axi_bready	In	1	AXI4-Lite Write Response Channel Ready. Indicates target is ready to receive response.
s_axi_arvalid	In	1	AXI4-Lite Read Address Channel Read Address Valid
s_axi_arready	Out	1	Ready. Indicates DMA is ready to accept the read address.
s_axi_araddr	In	32	AXI4-Lite Read Address Bus
s_axi_rvalid	Out	1	AXI4-Lite Read Data Channel Read Data Valid
s_axi_rready	In	1	AXI4-Lite Read Data Channel Read Data Ready. Indicates target is ready to accept the read data.
s_axi_rdata	Out	32	AXI4-Lite Read Data Bus
s_axi_rresp	Out	2	AXI4-Lite Read Response Channel Response. Indicates results of the read transfer.

S_AXI_ACLK

The AXI4-Lite interface must be synchronous to the `S_AXI_ACLK` clock signal. The AXI4-Lite interface input signals are sampled on the rising edge of `ACLK`. The AXI4-Lite output signal changes occur after the rising edge of `ACLK`. The AXI4-Stream interfaces signals are not affected by the `S_AXI_ACLK`.

S_AXI_ACLKEN

The `S_AXI_ACLKEN` pin is an active-high, synchronous clock-enable input for the AXI4-Lite interface. Setting `S_AXI_ACLKEN` low (de-asserted) halts the operation of the AXI4-Lite interface despite rising edges on the `S_AXI_ACLK` pin. AXI4-Lite interface states are maintained, and AXI4-Lite interface output signal levels are held until `S_AXI_ACLKEN` is asserted again. When `S_AXI_ACLKEN` is de-asserted, AXI4-Lite interface inputs are not sampled, except `S_AXI_ARESETn`, which supersedes `S_AXI_ACLKEN`. The AXI4-Stream interfaces signals are not affected by the `S_AXI_ACLKEN`.

S_AXI_ARESETn

The `S_AXI_ARESETn` pin is an active-low, synchronous reset input for the AXI4-Lite interface. `S_AXI_ARESETn` supersedes `S_AXI_ACLKEN`, and when set to 0, the core resets at the next rising edge of `S_AXI_ACLK` even if `S_AXI_ACLKEN` is de-asserted. The `S_AXI_ARESETn` signal must be synchronous to the `S_AXI_ACLK` and must be held low for a minimum of 32 clock cycles of the slowest clock. The `S_AXI_ARESETn` input is resynchronized to the `ACLK` clock domain. The AXI4-Stream interfaces and core signals are also reset by `S_AXI_ARESETn`.

Register Space

The standardized Xilinx Video IP register space is partitioned to control-, timing-, and core specific registers. The TPG core uses only one timing related register, `ACTIVE_SIZE` (0x0020), which allows specifying the input frame dimensions. Also, the core has thirteen core-specific registers which allow the user to dynamically control the operation of the core. Table 2-10 describes the register names.

0x0
Table 2-10: **Register Names and Descriptions**

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0000	CONTROL	R/W	N	Power-on-Reset : 0x0	Bit 0: SW_ENABLE Bit 1: REG_UPDATE Bit 30: FRAME_SYNC_RESET (1: reset) Bit 31: SW_RESET (1: reset)
0x0004	STATUS	R/W	No	0	Bit 0: PROC_STARTED Bit 1: EOF Bit 16: SLAVE_ERROR
0x0008	ERROR	R/W	No	0	Bit 0: SLAVE_EOL_EARLY Bit 1: SLAVE_EOL_LATE Bit 2: SLAVE_SOF_EARLY Bit 3: SLAVE_SOF_LATE
0x000C	IRQ_ENABLE	R/W	No	0	16-0: Interrupt enable bits corresponding to `STATUS` bits
0x0010	VERSION	R	N/A	0x0400a000	7-0: REVISION_NUMBER 11-8: PATCH_ID 15-12: VERSION_REVISION 23-16: VERSION_MINOR 31-24: VERSION_MAJOR

Table 2-10: **Register Names and Descriptions *(Cont'd)***

Address (hex) BASEADDR +	Register Name	Access Type	Double Buffered	Default Value	Register Description
0x0020	ACTIVE_SIZE	R/W	Yes	0x04380780	12-0: Number of Active Pixels per Scanline 28-16: Number of Active Lines per Frame
0x0100	PATTERN_CONTROL	R/W	Yes	0x0	12-0: Pattern selection. See register description below for information on the functionality of specific bits.)
0x0104	MOTION_SPEED	R/W	Yes	0x04	7-0: How quickly the temporal features of the supported test pattern will change from frame to frame
0x0108	CROSS_HAIRS	R/W	Yes	0x00640064	12-0: Horizontal cross hairs value 28-16: Vertical cross hairs value
0x010C	ZPLATE_HOR_CONTROL	R/W	Yes	0x1E	15-0: Sets a starting point in the ROM based sinusoidal values for the horizontal component 31-16: Manipulates how quickly the horizontal component changes.
0x110	ZPLATE_VER_CONTROL	R/W	Yes	0x1	15-0: Sets a starting point in the ROM based sinusoidal values for the vertical component 31-16: Manipulates how quickly the vertical component changes.
0x114	BOX_SIZE	R/W	Yes	0x32	12-0: Size of the box in pixel x pixel
0x118	BOX_COLOR	R/W	Yes	0x0	7-0: Blue, Y component of the box 15-8: Green, Cr component of the box 23-16: Red, Cb component of the box
0x11C	STUCK_PIXEL_THRESH	R/W	Yes	0x0	15-0: An upper limit count for a pseudo random number generator for the insertion of stuck pixels
0x120	NOISE_GAIN	R/W	Yes	0x0	7-0: Value to increase the noise added to each component when the noise feature is enabled

1. Only available if the debugging features option is enabled when the core is instantiated.

CONTROL (0x0000) Register

Bit 0 of the `CONTROL` register, `SW_ENABLE`, facilitates enabling and disabling the core from software. Writing '0' to this bit effectively disables the core halting further operations, which blocks the propagation of all video signals. After Power up, or Global Reset, the `SW_ENABLE` defaults to 0 for the AXI4-Lite interface. Similar to the `ACLKEN` pin, the `SW_ENABLE` flag is not synchronized with the AXI4-Stream interfaces: Enabling or Disabling the core takes effect immediately, irrespective of the core processing status. Disabling the core for extended periods may lead to image tearing.

Bit 1 of the CONTROL register, REG_UPDATE is a write done semaphore for the host processor, which facilitates committing all user and timing register updates simultaneously. The TPG core has many registers and are double buffered. One set of registers (the processor registers) is directly accessed by the processor interface, while the other set (the active set) is actively used by the core. New values written to the processor registers are copied over to the active set at the end of the AXI4-Stream interface frame, if and only if REG_UPDATE is set. Setting REG_UPDATE to 0 before updating multiple register values, then setting REG_UPDATE to 1 when updates are completed ensures all registers are updated simultaneously at the frame boundary without causing image tearing.

Bits 30 and 31 of the CONTROL register, FRAME_SYNC_RESET and SW_RESET facilitate software reset. Setting SW_RESET reinitializes the core to GUI default values, all internal registers and outputs are cleared and held at initial values until SW_RESET is set to 0. The SW_RESET flag is not synchronized with the AXI4-Stream interfaces. Resetting the core while frame processing is in progress causes image tearing. For applications where the software reset functionality is desirable, but image tearing has to be avoided a frame synchronized software reset (FRAME_SYNC_RESET) is available. Setting FRAME_SYNC_RESET to 1 resets the core at the end of the frame being processed, or immediately if the core is between frames when the FRAME_SYNC_RESET was asserted. After reset, the FRAME_SYNC_RESET bit is automatically cleared, so the core can get ready to process the next frame of video as soon as possible. The default value of both RESET bits is 0. Core instances with no AXI4-Lite control interface can only be reset through the ARESETn pin.

STATUS (0x0004) Register

All bits of the STATUS register can be used to request an interrupt from the host processor. To facilitate identification of the interrupt source, bits of the STATUS register remain set after an event associated with the particular STATUS register bit, even if the event condition is not present at the time the interrupt is serviced.

Bits of the STATUS register can be cleared individually by writing '1' to the bit position.

Bit 0 of the STATUS register, PROC_STARTED, indicates that processing of a frame has commenced through the AXI4-Stream interface.

Bit 1 of the STATUS register, End-of-frame (EOF), indicates that the processing of a frame has completed.

Bit 16 of the STATUS register, SLAVE_ERROR, indicates that one of the conditions monitored by the ERROR register has occurred.

ERROR (0x0008) Register

Bit 4 of the STATUS register, SLAVE_ERROR, indicates that one of the conditions monitored by the ERROR register has occurred. This bit can be used to request an interrupt from the host processor. To facilitate identification of the interrupt source, bits of the STATUS and ERROR registers remain set after an event associated with the particular ERROR register bit, even if the event condition is not present at the time the interrupt is serviced.

Bits of the ERROR register can be cleared individually by writing '1' to the bit position to be cleared.

Bit 0 of the ERROR register, EOL_EARLY, indicates an error during processing a video frame through the AXI4-Stream slave port. The number of pixels received between the latest and the preceding End-Of-Line (EOL) signal was less than the value programmed into the ACTIVE_SIZE register.

Bit 1 of the ERROR register, EOL_LATE, indicates an error during processing a video frame through the AXI4-Stream slave port. The number of pixels received between the last EOL signal surpassed the value programmed into the ACTIVE_SIZE register.

Bit 2 of the ERROR register, SOF_EARLY, indicates an error during processing a video frame through the AXI4-Stream slave port. The number of pixels received between the latest and the preceding Start-Of-Frame (SOF) signal was less than the value programmed into the ACTIVE_SIZE register.

Bit 3 of the ERROR register, SOF_LATE, indicates an error during processing a video frame through the AXI4-Stream slave port. The number of pixels received between the last SOF signal surpassed the value programmed into the ACTIVE_SIZE register.

IRQ_ENABLE (0x000C) Register

Any bits of the STATUS register can generate a host-processor interrupt request through the IRQ pin. The Interrupt Enable register facilitates selecting which bits of STATUS register asserts IRQ. Bits of the STATUS registers are masked by (AND) corresponding bits of the IRQ_ENABLE register and the resulting terms are combined (OR) together to generate IRQ.

Version (0x0010) Register

Bit fields of the Version Register facilitate software identification of the exact version of the hardware peripheral incorporated into a system. The core driver can take advantage of this Read-Only value to verify that the software is matched to the correct version of the hardware.

ACTIVE_SIZE (0x0020) Register

The ACTIVE_SIZE register encodes the number of active pixels per scan line and the number of active scan lines per frame. The lower half-word (bits 12:0) encodes the number of active pixels per scan line. Supported values are between 32 and the value provided in the **Maximum number of pixels per scan line** field in the GUI. The upper half-word (bits 28:16) encodes the number of active lines per frame. Supported values are 32 to 7680. To avoid processing errors, the user should restrict values written to ACTIVE_SIZE to the range supported by the core instance.

PATTERN_CONTROL (0x0100) Register

The PATTERN_CONTROL register controls the majority of the pattern manipulations that the TPG core produces.

Bits 3:0 - These bits control which pattern are generated on the output of the core based on the following values:

- 0x0 - Pass the video input straight through the video output
- 0x1 - Horizontal Ramp which increases each component (RGB or CbCr) horizontally by 1
- 0x2 -Vertical Ramp which increases each component (RGB or CbCr) vertically by 1
- 0x3 - Temporal Ramp which increases every pixel by a value set in the MOTION_SPEED register for every frame. Luma (Y) stays at a fixed value of 2.
- 0x4 - Solid red output
- 0x5 - Solid green output
- 0x6 - Solid blue output
- 0x7 - Solid black output
- 0x8 - Solid white output
- 0x9 - Color bars
- 0xA - Zone Plate output produces a ROM based sinusoidal pattern. This option has dependencies on the MOTION_SPEED, ZPLATE_HOR_CONTR and ZPLATE_VER_CONTROL registers.
- 0xB - Tartan Color Bars
- 0xC -Draws a cross hatch pattern.

Bit 4 -Draws cross hairs one pixel in width on the output of the video. This feature depends on the CROSS_HAIRS register.

Bit 5 - Enables a moving box to be drawn over the video output. This option is dependent on the BOX_SIZE and BOX_COLOR registers

Bits 8:6 - Mask out a particular color component

- 0x0 - No masking
- 0x1 - Mask out the red, Y (for YCbCr mode) component

- 0x2 - Mask out the green, Cb (for YCbCr mode) component
- 0x4 - Mask out the blue, Cr (for YCbCr mode) component

Bit 9 - Enable a stuck pixel feature to test a pixel correction core. This feature is dependent on the `STUCK_PIXEL_THRESH` register.

Bit 10 - Enable a noisy output. This feature adds noise to the output video to test any type of noise reduction video cores. The `NOISE_GAIN` register is used to increase or decrease the noise produced on the output.

Bit 11 - Reserved

Bit 12 - Enable the motion features of the core. This bit will turn on the motion for the ramps (horizontal, vertical and temporal), and the box.

MOTION_SPEED (0x0104) Register

This register cause the ramps, box and zone plate to increment by that value every frame.

CROSS_HAIRS (0x0108) Register

Bits 12:0 -The row of the frame that will have the horizontal line of the cross hairs

Bits 28:16 - The column of the frame that will have the vertical line of the cross hairs

ZPLATE_HOR_CONTROL (0x010C) Register

Bits 15:0 - Sets how widely spaced the horizontal component of a sine function are placed together. The larger the number, the more narrow the spacing.

Bits 31:16 - Controls the horizontal component speed by manipulating the indexing into the ROM table that contains the sinusoidal values.

ZPLATE_VER_CONTROL (0x0110) Register

Bits 15:0 - Sets the vertical component starting point in the ROM table that contains the sinusoidal values.

Bits 31:16 - Controls the vertical component speed by manipulating the indexing into the ROM table that contains the sinusoidal values.

BOX_SIZE (0x0114) Register

Bits 12:0 - The number in this register will size the box as NxN where each N value is a pixel. The size of the box has to be set smaller than the frame size that is set in the `ACTIVE_SIZE` register.

BOX_COLOR (0x0118) Register

Bits 7:0 - Blue, Y (for YCbCr mode) color component of the box. The number is a standard RGB / YCbCr 8 bit value.

Bits 15:8 - Green, Cr (for YCbCr mode) color component of the box. The number is a standard RGB / YCrCb 8 bit value.

Bits 7:0 - Red, Cb (for YCbCr mode) color component of the box. The number is a standard RGB / YCrCb 8 bit value.

STUCK_PIXEL_THRESH (0x011C) Register

Bits 15:0 - This number is the upper count limit of the pseudo random number generator. Various bits of the PRNG number are used to create a stuck pixel at various locations in the frame. The PRNG is reset every frame so that the stuck pixels will appear in the same locations for every frame.

NOISE_GAIN (0x0120) Register

Bits 7:0 - A value multiplied with a continuously running PRNG the product of which is added to the color components of the video stream.

The Interrupt Subsystem

`STATUS` register bits can trigger interrupts so embedded application developers can quickly identify faulty interfaces or incorrectly parameterized cores in a video system. Irrespective of whether the AXI4-Lite control interface is present or not, the TPG core detects AXI4-Stream framing errors, as well as the beginning and the end of frame processing.

When the core is instantiated with an AXI4-Lite Control interface, the optional interrupt request pin (`IRQ`) is present. Events associated with bits of the `STATUS` register can generate a (level triggered) interrupt, if the corresponding bits of the interrupt enable register (`IRQ_ENABLE`) are set. Once set by the corresponding event, bits of the `STATUS` register stay set until the user application clears them by writing '1' to the desired bit positions. Using this mechanism the system processor can identify and clear the interrupt source.

Without the AXI4-Lite interface the user can still benefit from the core signaling error and status events. By selecting the **Enable INTC Port** check-box on the GUI, the core generates the optional `INTC_IF` port. This vector of signals gives parallel access to the individual interrupt sources listed in Table 2-11.

Unlike STATUS and ERROR flags, INTC_IF signals are not held, rather stay asserted only while the corresponding event persists.

Table 2-11: **INTC_IF Signal Functions**

INTC_IF signal	Function
0	Frame processing start
1	Frame processing complete
2	EOL Early
3	EOL Late
4	SOF Early
5	SOF Late

In a system integration tool, such as EDK, the interrupt controller INTC IP can be used to register the selected INTC_IF signals as edge triggered interrupt sources. The INTC IP provides functionality to mask (enable or disable), as well as identify individual interrupt sources from software. Alternatively, for an external processor or MCU the user can custom build a priority interrupt controller to aggregate interrupt requests and identify interrupt sources.

Chapter 3

Designing with the Core

General Design Guidelines

The TPG core provides a variety of test patterns to test a video processing core or system by feeding known patterns through the Video over AXI4-Stream interface.

The core accepts video data provided via an AXI4-Stream slave interface (when configured with an AXI4-Lite Control Interface), outputs pixels via an AXI4-Stream master interface, and can be controlled via an optional AXI4-Lite interface. The TPG block cannot change the input/output image sizes, the input and output pixel clock rates, or the frame rate.

RECOMMENDED: *When the AXI4-Lite and AXI4-Stream slave interfaces are included, the TPG core is designed to be used in conjunction with the Video In to AXI4-Stream core.*

Typically, the TPG core can be inserted in any part of a video system, such as the Image Sensor Pipeline shown in Figure 3-1.

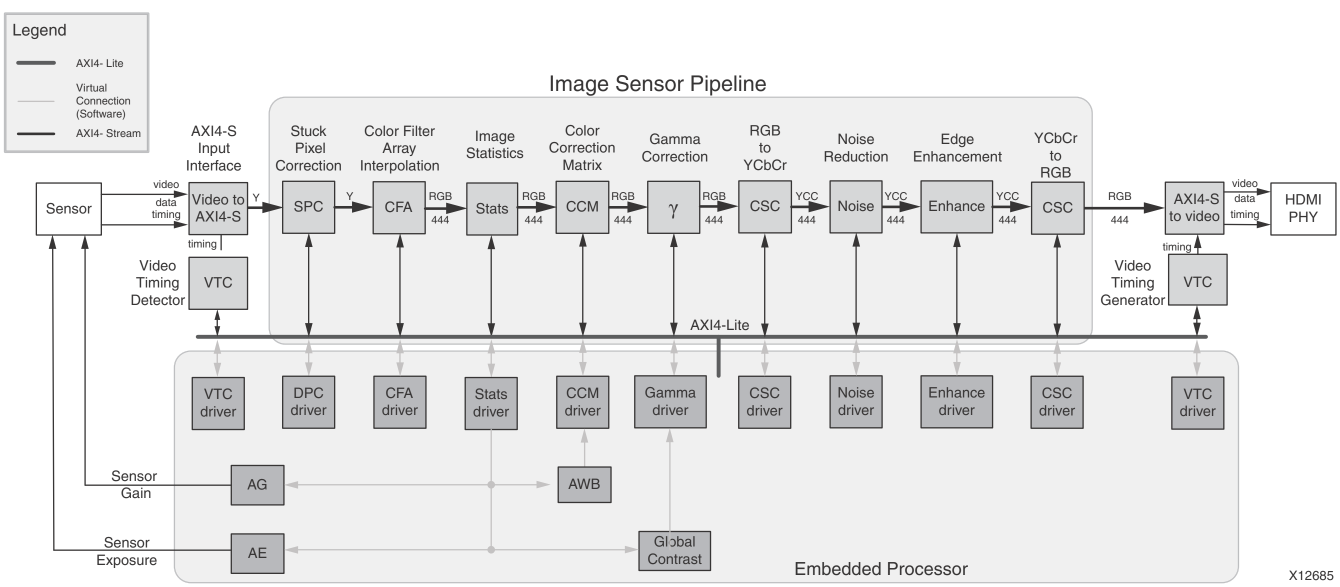

Figure 3-1: **Image Sensor Pipeline System with TPG Core**

Clock, Enable, and Reset Considerations

ACLK

The master and slave AXI4-Stream video interfaces use the ACLK clock signal as their shared clock reference, as shown in Figure 3-2.

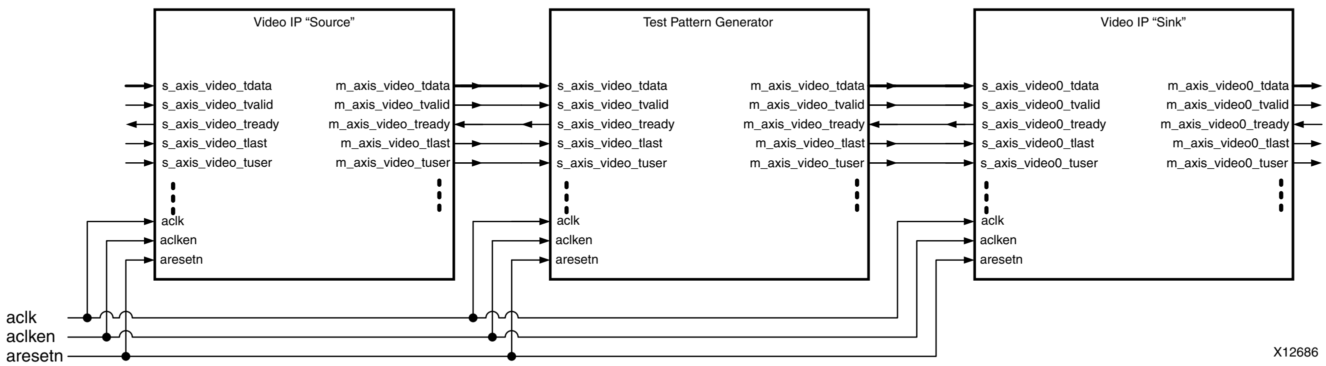

Figure 3-2: **Example of ACLK Routing in an ISP Processing Pipeline**

S_AXI_ACLK

The AXI4-Lite interface uses the A_AXI_ACLK pin as its clock source. The ACLK pin is not shared between the AXI4-Lite and AXI4-Stream interfaces. The TPG core contains clock-domain crossing logic between the ACLK (AXI4-Stream and Video Processing) and S_AXI_ACLK (AXI4-Lite) clock domains. The core automatically ensures that the AXI4-Lite transactions completes even if the video processing is stalled with ARESETn, ACLKEN or with the video clock not running.

ACLKEN

The TPG core has two enable options: the ACLKEN pin (hardware clock enable), and the software reset option provided via the AXI4-Lite control interface (when present).

ACLKEN is by no means synchronized internally to AXI4-Stream frame processing therefore de-asserting ACLKEN for extended periods of time may lead to image tearing.

The ACLKEN pin facilitates:

- Multi-cycle path designs (high speed clock division without clock gating),
- Standby operation of subsystems to save on power
- Hardware controlled bring-up of system components

IMPORTANT: *When ACLKEN (clock enable) pins are used (toggled) in conjunction with a common clock source driving the master and slave sides of an AXI4-Stream interface, to prevent transaction errors the ACLKEN pins associated with the master and slave component interfaces must also be driven by the same signal (Figure 2-2).*

IMPORTANT: *When two cores connected via AXI4-Stream interfaces, where only the master or the slave interface has an ACLKEN port, which is not permanently tied high, the two interfaces should be connected via the AXI4-Stream Interconnect or AXI-FIFO cores to avoid data corruption (Figure 2-3).*

ARESETn

The TPG core has two reset source: the `ARESETn` pin (hardware reset), and the software reset option provided via the AXI4-Lite control interface (when present).

IMPORTANT: *ARESETn is not synchronized internally to AXI4-Stream frame processing. De-asserting ARESETn while a frame is being process leads to image tearing.*

The external reset pulse needs to be held for 32 `ACLK` cycles to reset the core. The `ARESETn` signal only resets the AXI4-Stream interfaces. The AXI4-Lite interface is unaffected by the `ARESETn` signal to allow the video processing core to be reset without halting the AXI4-Lite interface.

IMPORTANT: *When a system with multiple-clocks and corresponding reset signals are being reset, the reset generator has to ensure all signals are asserted/de-asserted long enough so that all interfaces and clock-domains are correctly reinitialized.*

S_AXI_ARESETn

The `S_AXI_ARESETn` signal is synchronous to the `S_AXI_ACLK` clock domain, but is internally synchronized to the `ACLK` clock domain. The `S_AXI_ARESETn` signal resets the entire core including the AXI4-Lite and AXI4-Stream interfaces.

System Considerations

The Test Pattern Generator IP core must be configured for the actual input image frame-size to operate properly. To gather the frame size information from the image video stream, it can be connected to the Video In to AXI4-Stream input and the Video Timing Controller. The timing detector logic in the Video Timing Controller gathers the video timing signals. The AXI4-Lite control interface on the Video Timing Controller allows the system processor to read out the measured frame dimensions, and program all downstream cores, such as the TPG, with the appropriate image dimensions.

Programming Sequence

If processing parameters (such as the image size) need to be changed on the fly, or if the system needs to be reinitialized, it is recommended that pipelined Xilinx IP video cores are disabled/reset from system output towards the system input, and programmed/enabled from system output to system input. STATUS register bits allow system processors to identify the processing states of individual constituent cores, and successively disable a pipeline as one core after another is finished processing the last frame of data.

Error Propagation and Recovery

Parameterization and/or configuration registers define the dimensions of video frames video IP should process. Starting from a known state, based on these configuration settings the IP can predict when the beginning of the next frame is expected. Similarly, the IP can predict when the last pixel of each scan line is expected. SOF detected before it was expected (early), or SOF not present when it is expected (late), EOL detected before expected (early), or EOL not present when expected (late), signals error conditions indicative of either upstream communication errors or incorrect core configuration.

When SOF is detected early, the output SOF signal is generated early, terminating the previous frame immediately. When SOF is detected late, the output SOF signal is generated according to the programmed values. Extra lines / pixels from the previous frame are dropped until the input SOF is captured.

Similarly, when EOL is detected early, the output EOL signal is generated early, terminating the previous line immediately. When EOL is detected late, the output EOL signal is generated according to the programmed values. Extra pixels from the previous line are dropped until the input EOL is captured.

C Model Reference

There is no C model for this core.

SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Core

This chapter includes information about using Xilinx tools to customize and generate the core in the Vivado™ Design Suite environment.

GUI

The Xilinx Test Pattern Generator core is easily configured to meet the developer's specific needs through the Vivado Design Suite GUI. This section provides a quick reference to parameters that can be configured at generation time.

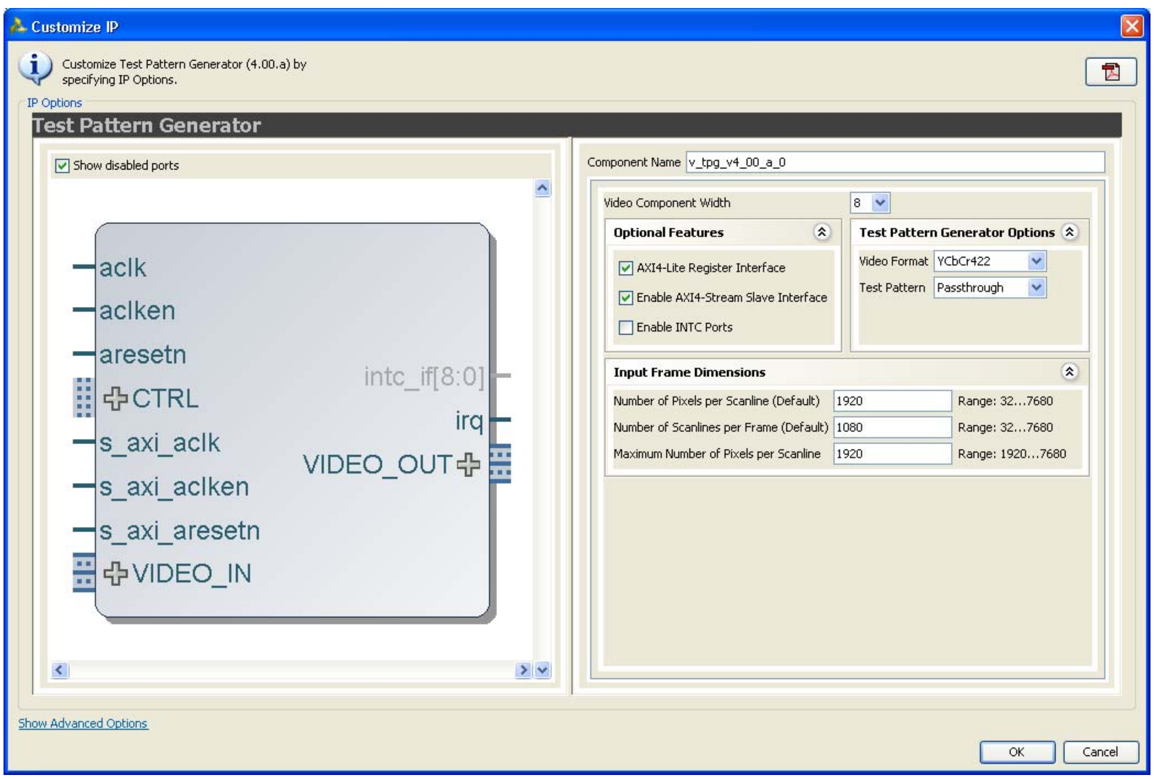

Figure 5-1: **Test Pattern Generator Vivado IP Catalog GUI**

The GUI displays a representation of the IP symbol on the left side, and the parameter assignments on the right side, which are described as follows:

- **Component Name:** The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and "_". The name `v_tpg_v4_00_a` cannot be used as a component name.
- **Video Component Width:** Specifies the bit width of input samples. Permitted values are 8, 10 and 12 bits.
- **Optional Features:**
  - **AXI4-Lite Register Interface:** When selected, the core is generated with an AXI4-Lite interface, which gives access to dynamically program and change processing parameters. When deselected, you generate only the test pattern selected and the AXI4-Stream slave interface is removed. For more information, see Control Interface in Chapter 2.
  - **Enable AXI4-Stream Slave Interface**: When selected, the core requires a video input stream. When deselected, the core can operate without a video input stream producing only the test patterns that the TPG can generate. When the AXI4-Stream interface is disabled and the Pass Through option is enabled, the core outputs either a black screen in RGB mode or a green screen in YCbCr422 mode. The AXI4-Stream slave interface is disabled when the AXI4-Lite interface is disabled, causing the Test Pattern Generator core to produce the test pattern selected in the GUI.
  - **INTC Interface:** When selected, the core generates the optional INTC_IF port, which gives parallel access to signals indicating frame processing status and error conditions. For more information, see The Interrupt Subsystem in Chapter 2.
- **Video Format**: Specify to use RGB or YCbCr video formats. Selecting the video format is dependent on the video format of your system. Once selected, you can only change this option if you regenerate the core.
- **Test Pattern**: This option sets up the default test pattern when the TPG core becomes alive in the system. The test pattern can be changed by writing a different value to the `PATTERN_CONTROL` register via a master component (usually a processor) on the AXI4-Lite interface. This selection will also determine the test pattern when the TPG is run in constant mode (no AXI4-Lite interface).
- **Input Frame Dimensions:**
  - **Number of Active Pixels per Scan line:** When the AXI4-Lite control interface is enabled, this generated core uses the value specified in Vivado's Customize IP GUI as the default value for the lower half-word of the `ACTIVE_SIZE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the horizontal size of the frames the generated core instance is to process.
  - **Number of Active Lines per Frame:** When the AXI4-Lite control interface is enabled, the generated core uses this value specified in the Vivado's Customize IP GUI as the default value for the upper half-word of the `ACTIVE_SIZE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines

the vertical size (number of lines) of the frames the generated core instance is to process.

- **Maximum Number of Active Pixels Per Scan line:** Specifies the maximum number of pixels per scan line that can be processed by the generated core instance. Permitted values are from 32 to 7680. Specifying this value is necessary to establish the depth of internal line buffers. The actual value selected for **Number of Active Pixels per Scan line**, or the corresponding lower half-word of the `ACTIVE_SIZE` register must always be less than the value provided by **Maximum Number of Active Pixels Per Scan line**. Using a tight upper-bound results in optimal block RAM usage. This field is enabled only when the AXI4-Lite interface is selected. Otherwise contents of the field are reflecting the actual contents of the **Number of Active Pixels per Scan line** field as for constant mode the maximum number of pixels equals the active number of pixels.

Output Generation

Vivado generates the files necessary to build the core and place those files in the `<project>/<project>.srcs/sources_1/ip/<core>` directory.

File Details

The Vivado output consists of some or all the following files.

Name	Description
v_tpg_v4_00_a	Library directory and encrypted source files for the v_tpg_v4_00_a core
v_tc_v5_01_a	Library directory and encrypted source files for the helper core used with the v_tpg_v4_00_a
v_tpg_v4_00_a.veo	Verilog instantiation template
v_tpg_v4_00_a.vho	VHDL instantiation template
v_tpg_v4_00_a.xci	IP-XACT XML file describing which options were used to generate the core. An XCI file can also be used as a source file for Vivado.
v_tpg_v4_00_a.xml	IP-XACT XML file describing how the core is constructed so Vivado can properly build the core.

Constraining the Core

Required Constraints

The ACLK pin should be constrained at the desired pixel clock rate for your video stream. The S_AXI_ACLK pin should be constrained at the frequency of the AXI4-Lite subsystem. In addition to clock frequency, the following constraints should be applied to cover all clock domain crossing data paths.

XDC

```
set_max_delay -to [get_cells -hierarchical -match_style ucf "*U_VIDEO_CTRL*/
*SYNC2PROCCLK_I*/data_sync_reg[0]*"] -datapath_only 2
set_max_delay -to [get_cells -hierarchical -match_style ucf "*U_VIDEO_CTRL*/
*SYNC2VIDCLK_I*/data_sync_reg[0]*"] -datapath_only 2
```

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for the Test Pattern Generator core.

Clock Frequencies

The pixel clock (ACLK) frequency is the required frequency for the Test Pattern Generator core. See Maximum Frequencies in Chapter 2. The S_AXI_ACLK maximum frequency is the same as the ACLK maximum.

Clock Management

The core automatically handles clock domain crossing between the ACLK (video pixel clock and AXI4-Stream) and the S_AXI_ACLK (AXI4-Lite) clock domains. The S_AXI_ACLK clock can be slower or faster than the `ACLK` clock signal, but must not be more than 128x faster than `ACLK`.

Clock Placement

There are no specific Clock placement requirements for the Test Pattern Generator core.

Banking

There are no specific Banking rules for the Test Pattern Generator core.

Transceiver Placement

There are no Transceiver Placement requirements for the Test Pattern Generator core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for the Test Pattern Generator core.

Detailed Example Design

Demonstration Test Bench

A demonstration test bench is provided with the core which enables you to observe core behavior in a typical scenario. This test bench is generated together with the core in Vivado design tools. You are encouraged to make simple modifications to the configurations and observe the changes in the waveform.

Generating the Test Bench

1. After customizing the IP, right-click on the core instance in **Sources** pane and select **Generate Output Products** (Figure 7-1).

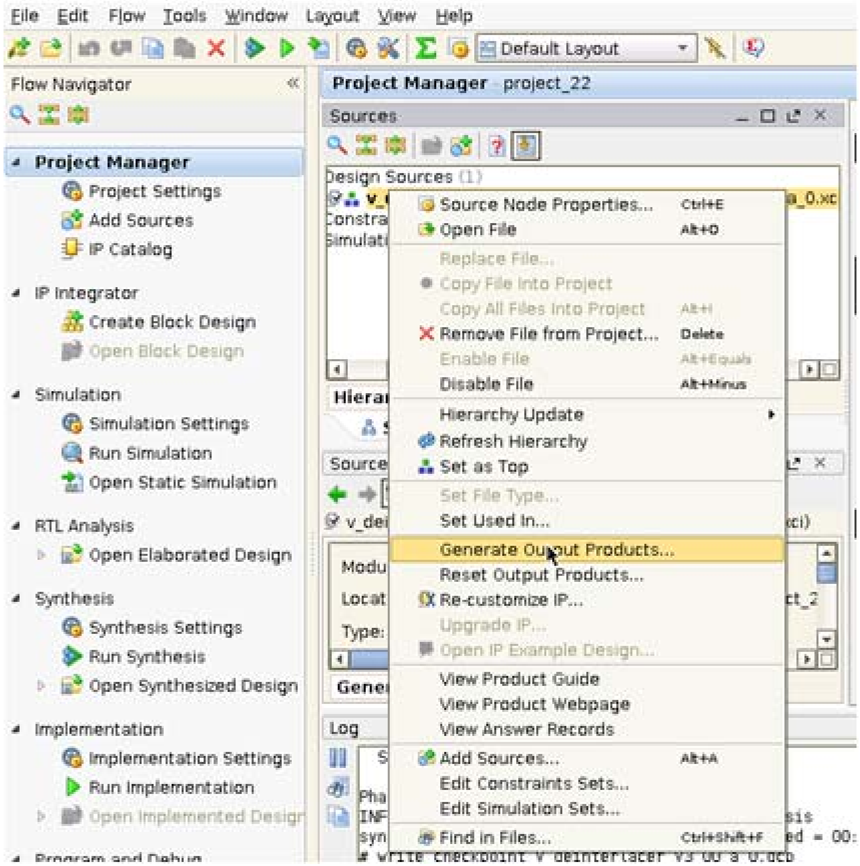

Figure 7-1: **Sources Pane**

A pop-up window prompts you to select items to generate.

2. Click on **Test Bench** and make sure **Action: Generate** is selected.

   The demo test bench package will be generated in the following directory (Figure 7-2):

```
<PROJ_DIR>/<PROJ_NAME>.srcs/sources_1/ip/<IP_INSTANCE_NAME>/<IP_INSTANCE_NAME>/
demo_tb/
```

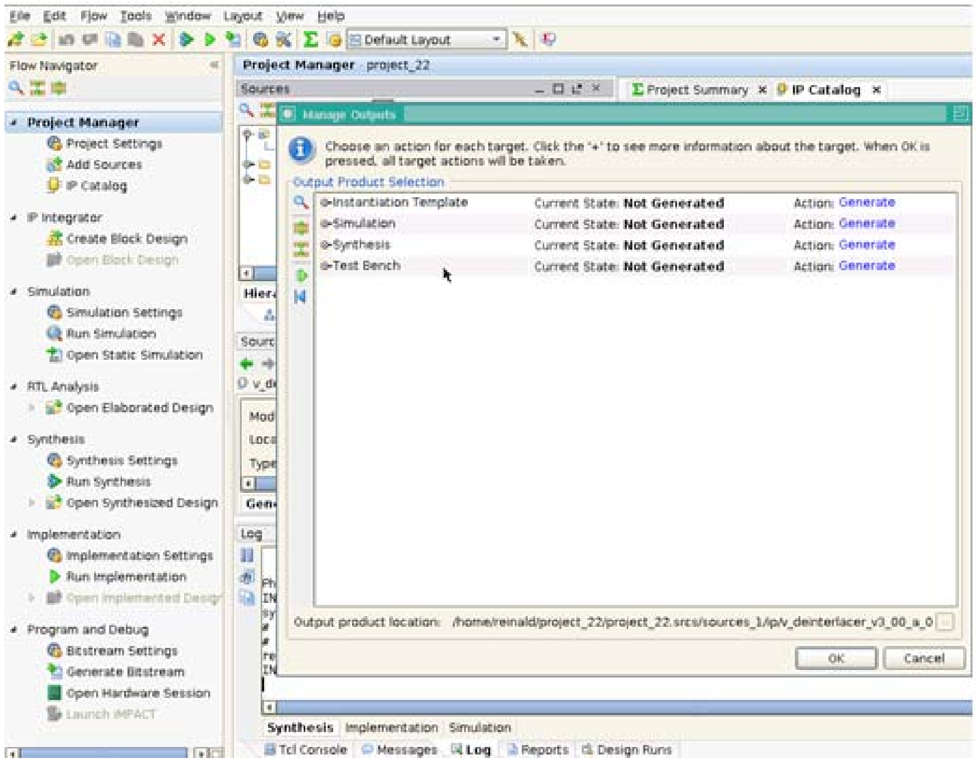

Figure 7-2: **Test Bench**

Directory and File Contents

The following files are generated in the demo test bench output directory:

- `axi4lite_mst.v`
- `axi4s_video_mst.v`
- `axi4s_video_slv.v`
- `ce_generator.v`
- `tb_<IP_instance_name>.v`

Test Bench Structure

The top-level entity is **`tb_<IP_instance_name>`**.

It instantiates the following modules:

- `DUT`

  The <IP> core instance under test.

- `axi4lite_mst`

The AXI4-Lite master module, which initiates AXI4-Lite transactions to program core registers.

- `axi4s_video_mst`

  The AXI4-Stream master module, which generates ramp data and initiates AXI4-Stream transactions to provide video stimuli for the core and can also be used to open stimuli files generated from the reference C-models and convert them into corresponding AXI4-Stream transactions.

  To do this, edit `tb_<IP_instance_name>.v`:

  a. Add define macro for the stimuli file name and directory path
     `define STIMULI_FILE_NAME<path><filename>`.

  b. Comment-out/remove the following line:
     `MST.is_ramp_gen(`C_ACTIVE_ROWS, `C_ACTIVE_COLS, 2);`
     and replace with the following line:
     `MST.use_file(`STIMULI_FILE_NAME);`

  For information on how to generate stimuli files, refer to C Model Reference.

- `axi4s_video_slv`

  The AXI4-Stream slave module, which acts as a passive slave to provide handshake signals for the AXI4-Stream transactions from the core's output, can be used to open the data files generated from the reference C-model and verify the output from the core.

  To do this, edit `tb_<IP_instance_name>.v`:

  a. Add define macro for the golden file name and directory path
     `define GOLDEN_FILE_NAME "<path><filename>"`.

  b. Comment-out the following line:
     `SLV.is_passive;`
     and replace with the following line:
     `SLV.use_file(`GOLDEN_FILE_NAME);`

  For information on how to generate golden files, refer to C Model Reference.

- `ce_gen`

  Programmable Clock Enable (`ACLKEN`) generator.

Running the Simulation

There are two ways to run the demonstration test bench.

Option 1: Launch Simulation from the Vivado GUI

This runs the test bench with the AXI4-Stream Master producing ramp data as stimuli, and AXI4-Stream Slave set to passive mode.

- Click **Simulation Settings** in the Flow Navigation window, change Simulation top module name to **tb_<IP_instance_name>**.
- Click **Run Simulation**. XSIM launches and you should be able to see the signals.
- You can also choose Modelsim for simulation by going to **Project Settings** and selecting Modelsim as the Target Simulator (Figure 7-3).

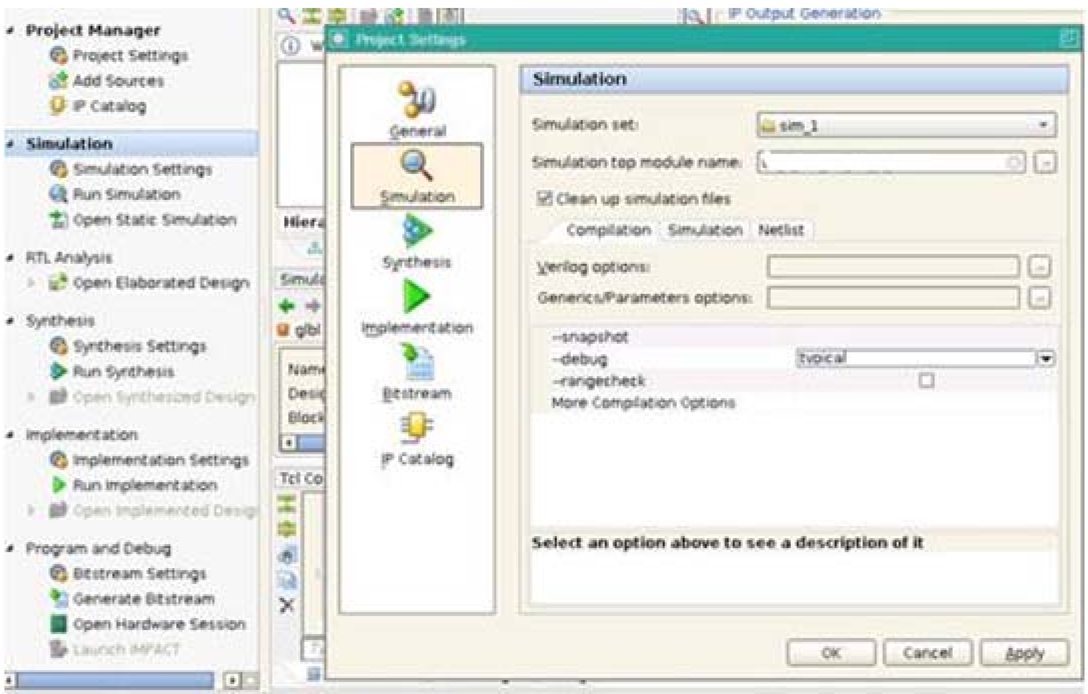

Figure 7-3: **Simulation GUI**

Option 2: Manually Compile and Run Simulation from Your Simulation Environment

- Add the generated test bench files to a new simulation set, along with the customized IP. For information on the location of generated test bench files, refer to Generating the Test Bench.
- Setup the environment variables for Xilinx libraries
- Compile the generated IP

- Compile the test bench files
- Run the simulation

RECOMMENDED: *Change the default simulation time from* ***1000 ns*** *to* ***all*** *to be able observe a full frame transaction.*

SECTION III: ISE DESIGN SUITE

Chapter 8

Customizing and Generating the Core

This chapter includes information on using Xilinx tools to customize and generate the core in the ISE Design Suite environment.

GUI

The Xilinx Test Pattern Generator core is easily configured to meet the developer's specific needs through the EDK GUI. This section provides a quick reference to parameters that can be configured at generation time.

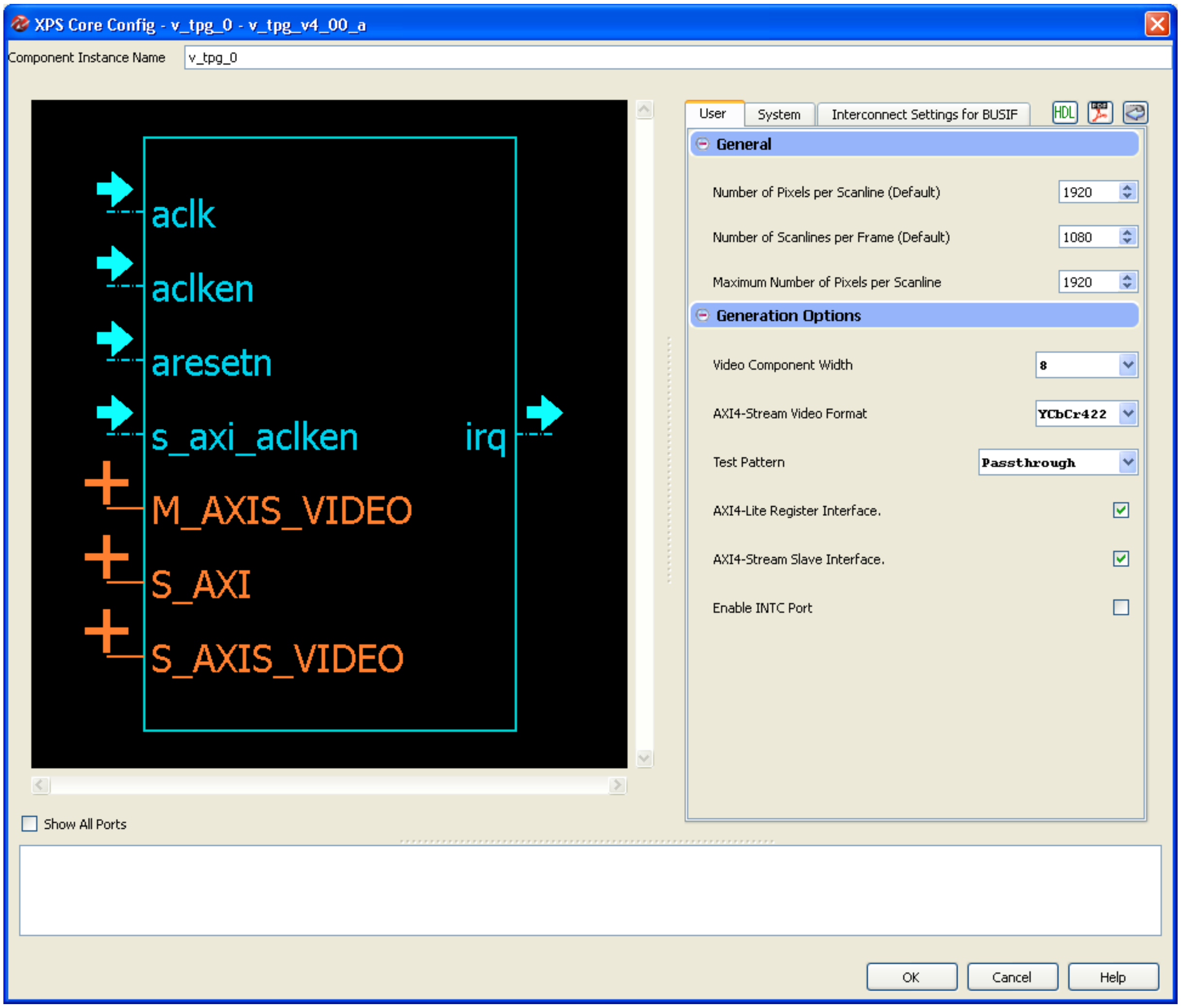

Figure 8-1: **Test Pattern EDK GUI**

The GUI displays a representation of the IP symbol on the left side, and the parameter assignments on the right side, which are described as follows:

- **Component Instance Name:** The component name is used as the base name of output files generated for the module. Names must begin with a letter and must be composed from characters: a to z, 0 to 9 and "_". The name `v_tpg_v4_00_a` cannot be used as a component name.
- **Input Frame Dimensions:**
  - **Number of Pixels per Scanline (Default)**: When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the CORE Generator GUI as the default value for the lower half-word of the `ACTIVE_SIZE` register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the horizontal size of the frames the generated core instance is to process.
  - **Number of Scanlines per Frame (Default)**: When the AXI4-Lite control interface is enabled, the generated core uses the value specified in the CORE Generator GUI as

the default value for the upper half-word of the ACTIVE_SIZE register. When an AXI4-Lite interface is not present, the GUI selection permanently defines the vertical size (number of lines) of the frames the generated core instance is to process.

- **Maximum Number of Pixels Per Scanline**: Specifies the maximum number of pixels per scan line that can be processed by the generated core instance. Permitted values are from 32 to 7680. Specifying this value is necessary to establish the depth of internal line buffers. The actual value selected for **Number of Active Pixels per Scanline**, or the corresponding lower half-word of the ACTIVE_SIZE register must always be less than the value provided by **Maximum Number of Active Pixels Per Scan line**. Using a tight upper-bound results in optimal block RAM usage. This field is enabled only when the AXI4-Lite interface is selected. Otherwise contents of the field reflect the actual contents of the **Number of Active Pixels per Scan line** field because the maximum number of pixels equals the active number of pixels in constant mode.

- **Video Component Width:** Specifies the bit width of input samples. Permitted values are 8, 10, and 12 bits.
- **AXI4-Stream Video Format**: Specify to use RGB or YCbCr video formats. Selecting the video format is dependent on the video format of your system. Once selected, you can only change this option if you regenerate the core.
- **Test Pattern**: This option sets up the default test pattern when the TPG core becomes alive in the system. The test pattern can be changed by writing a different value to the PATTERN_CONTROL register via a master component (usually a processor) on the AXI4-Lite interface. This selection will also determine the test pattern when the TPG is run in constant mode (no AXI4-Lite interface).
- **Optional Features:**
  - **AXI4-Lite Register Interface:** When selected, the core is generated with an AXI4-Lite interface, which gives access to dynamically program and change processing parameters. When deselected, you generate only the test pattern selected and the AXI4-Stream slave interface is removed. For more information, see Control Interface in Chapter 2.
  - **Enable AXI4-Stream Slave Interface**: When selected, the core requires a video input stream. When deselected, the core can operate without a video input stream producing only the test patterns that the TPG can generate. When the AXI4-Stream interface is disabled and the Pass Through option is enabled, the core outputs either a black screen in RGB mode or a green screen in YCbCr422 mode. The AXI4-Stream slave interface is disabled when the AXI4-Lite interface is disabled, causing the Test Pattern Generator core to produce the test pattern selected in the GUI.
  - **INTC Interface:** When selected, the core generates the optional INTC_IF port, which gives parallel access to signals indicating frame processing status and error conditions. For more information, see The Interrupt Subsystem in Chapter 2.

Constraining the Core

Required Constraints

The ACLK pin should be constrained at the desired pixel clock rate for your video stream. The S_AXI_ACLK pin should be constrained at the frequency of the AXI4-Lite subsystem. In addition to clock frequency, the following constraints should be applied to cover all clock domain crossing data paths.

UCF

```
INST "*U_VIDEO_CTRL*/*SYNC2PROCCLK_I*/data_sync_reg[0]*" TNM =
"async_clock_conv_FFDEST";
TIMESPEC "TS_async_clock_conv" = FROM FFS TO "async_clock_conv_FFDEST" 2 NS
DATAPATHONLY;
INST "*U_VIDEO_CTRLk*/*SYNC2VIDCLK_I*/data_sync_reg[0]*" TNM =
"vid_async_clock_conv_FFDEST";
TIMESPEC "TS_vid_async_clock_conv" = FROM FFS TO "vid_async_clock_conv_FFDEST" 2 NS
DATAPATHONLY;
```

Device, Package, and Speed Grade Selections

There are no device, package, or speed grade requirements for the Test Pattern Generator core.

Clock Frequencies

The pixel clock (ACLK) frequency is the required frequency for the Test Pattern Generator core. See Maximum Frequencies in Chapter 2. The S_AXI_ACLK maximum frequency is the same as the ACLK maximum.

Clock Management

The core automatically handles clock domain crossing between the ACLK (video pixel clock and AXI4-Stream) and the S_AXI_ACLK (AXI4-Lite) clock domains. The S_AXI_ACLK clock can be slower or faster than the ACLK clock signal, but must not be more than 128x faster than ACLK.

Clock Placement

There are no specific Clock placement requirements for the Test Pattern Generator core.

Banking

There are no specific Banking rules for the Test Pattern Generator core.

Transceiver Placement

There are no Transceiver Placement requirements for the Test Pattern Generator core.

I/O Standard and Placement

There are no specific I/O standards and placement requirements for the Test Pattern Generator core.

SECTION IV: APPENDICES

Verification, Compliance, and Interoperability

Simulation

A highly parameterizable test bench was used to test the Test Pattern Generator core. Testing included the following:

- Register accesses
- Processing multiple frames of data
- AXI4-Stream bidirectional data-throttling tests
- Testing detection, and recovery from various AXI4-Stream framing error scenarios
- Testing different `ACLKEN` and `ARESETn` assertion scenarios
- Testing of various frame sizes
- Varying parameter settings

Hardware Testing

The Test Pattern Generator core has been validated in hardware at Xilinx to represent a variety of parameterizations, including the following:

- A test design was developed for the core that incorporated a MicroBlaze™ processor, AXI4-Lite interconnect and various other peripherals. The software for the test system included pre-generated input and output data along with live video stream. The MicroBlaze processor was responsible for:
    - Initializing the appropriate input and output buffers
    - Initializing the TPG core
    - Launching the test
    - Comparing the output of the core against the expected results

- Reporting the Pass/Fail status of the test and any errors that were found

Interoperability

The core slave (input) AXI4-Stream interface can work directly with any core that produces RGB or YCbCr 422 video data.

Migrating

For information about migration from ISE Design Suite to Vivado Design Suite, see *Vivado Design Suite Migration Methodology Guide* (UG911).

For a complete list of Vivado User and Methodology Guides, see the Vivado Design Suite - 2012.3 User Guides web page.

The Test Pattern Generator is a newly released core starting with version 4.00.a so there are currently no migration issues.

Debugging

This appendix includes details about resources available on the Xilinx Support website and debugging tools. In addition, this appendix provides a step-by-step debugging process and a flow diagram to guide you through debugging the Test Pattern Generator core.

The following topics are included in this appendix:

- Finding Help on Xilinx.com
- Debug Tools
- Hardware Debug
- Interface Debug
- AXI4-Stream Interfaces

Finding Help on Xilinx.com

To help in the design and debug process when using the Test Pattern Generator core, the Xilinx Support web page (www.xilinx.com/support) contains key resources such as product documentation, release notes, answer records, information about known issues, and links for opening a Technical Support Web Case.

Documentation

This product guide is the main document associated with the Test Pattern Generator core. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

Release Notes

Known issues for all cores, including the Test Pattern Generator core are described in the IP Release Notes Guide (XTP025).

Known Issues

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core are listed below, and can also be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

Contacting Technical Support

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- A block diagram of the video system that explains the video source, destination and IP (custom and Xilinx) used.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

Debug Tools

There are many tools available to address Test Pattern Generator core design issues. It is important to know which tools are useful for debugging various situations.

Example Design

The Vivado version of the Test Pattern Generator core is delivered with a functional test bench. Information about the example design can be found in *Chapter 6, Example Design for the Vivado™ Design Suite.*

ChipScope Pro Tool

The ChipScope™ Pro tool inserts logic analyzer, bus analyzer, and virtual I/O cores directly into your design. The ChipScope Pro tool allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool. For detailed information for using the ChipScope Pro tool, see www.xilinx.com/tools/cspro.htm.

Vivado Lab Tools

Vivado Lab Tools inserts logic analyzer, bus analyzer, and virtual I/O cores directly into your design. Vivado Lab Tools allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed.

Reference Boards

Various Xilinx development boards support Test Pattern Generator core. These boards can be used to prototype designs and establish that the core can communicate with the system.

C-Model Reference

No C model is available at this time.

License Checkers

If the IP requires a license key, the key must be verified. The ISE and Vivado tool flows have a number of license check points for gating licensed IP through the flow. If the license check succeeds, the IP may continue generation. Otherwise, generation halts with error. License checkpoints are enforced by the following tools:

- ISE flow: XST, NgdBuild, Bitgen
- Vivado flow: Vivado Synthesis, Vivado Implementation, write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Hardware Debug

Hardware issues can range from link bring-up to problems seen after hours of testing. This section provides debug steps for common issues. The ChipScope tool is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the ChipScope tool for debugging the specific problems.

Many of these common issues can also be applied to debugging design simulations. Details are provided on:

- General Checks

General Checks

Ensure that all the timing constraints for the core were properly incorporated from the example design and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the `LOCKED` port.
- If your outputs go to 0, check your licensing.

Interface Debug

AXI4-Lite Interfaces

Table C-1 describes how to troubleshoot the AXI4-Lite interface.

Table C-1: **Troubleshooting the AXI4-Lite Interface**

Symptom	Solution
Readback from the Version Register via the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Are the `S_AXI_ACLK` and `ACLK` pins connected? In EDK, verify that the `S_AXI_ACLK` and `ACLK` pin connections in the system.mhs file. The `VERSION_REGISTER` readout issue may be indicative of the core not receiving the AXI4-Lite interface.
Readback from the Version Register via the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Is the core enabled? Is `s_axi_aclken` connected to `vcc`? In EDK, verify that signal `ACLKEN` is connected in the **`system.mhs`** to either `net_vcc` or to a designated clock enable signal.
Readback from the Version Register via the AXI4-Lite interface times out, or a core instance without an AXI4-Lite interface seems non-responsive.	Is the core in reset? `S_AXI_ARESETn` and `ARESETn` should be connected to `vcc` for the core not to be in reset. In EDK, verify that the `S_AXI_ARESETn` and `ARESETn` signals are connected in the **`system.mhs`** to either `net_vcc` or to a designated reset signal.
Readback value for the `VERSION_REGISTER` is different from expected default values	The core and/or the driver in a legacy EDK/SDK project has not been updated. Ensure that old core versions, implementation files, and implementation caches have been cleared.

Assuming the AXI4-Lite interface works, the second step is to bring up the AXI4-Stream interfaces.

AXI4-Stream Interfaces

Table C-2 describes how to troubleshoot the AXI4-Stream interface.

Table C-2: **Troubleshooting AXI4-Stream Interface**

Symptom	Solution
Bit 0 of the `ERROR` register reads back set.	Bit 0 of the `ERROR` register, `EOL_EARLY`, indicates the number of pixels received between the latest and the preceding End-Of-Line (`EOL`) signal was less than the value programmed into the `ACTIVE_SIZE` register. If the value was provided by the Video Timing Controller core, read out `ACTIVE_SIZE` register value from the VTC core again, and make sure that the `TIMING_LOCKED` flag is set in the VTC core. Otherwise, using Chipscope, measure the number of active AXI4-Stream transactions between `EOL` pulses.
Bit 1 of the `ERROR` register reads back set.	Bit 1 of the `ERROR` register, `EOL_LATE`, indicates the number of pixels received between the last End-Of-Line (`EOL`) signal surpassed the value programmed into the `ACTIVE_SIZE` register. If the value was provided by the Video Timing Controller core, read out `ACTIVE_SIZE` register value from the VTC core again, and make sure that the `TIMING_LOCKED` flag is set in the VTC core. Otherwise, using Chipscope, measure the number of active AXI4-Stream transactions between `EOL` pulses.

Table C-2: **Troubleshooting AXI4-Stream Interface *(Cont'd)***

Symptom	Solution
Bit 2 or Bit 3 of the ERROR register reads back set.	Bit 2 of the ERROR register, SOF_EARLY, and bit 3 of the ERROR register SOF_LATE indicate the number of pixels received between the latest and the preceding Start-Of-Frame (SOF) differ from the value programmed into the ACTIVE_SIZE register. If the value was provided by the Video Timing Controller core, read out ACTIVE_SIZE register value from the VTC core again, and make sure that the TIMING_LOCKED flag is set in the VTC core. Otherwise, using Chipscope, measure the number EOL pulses between subsequent SOF pulses.
s_axis_video_tready stuck low, the upstream core cannot send data.	During initialization, line-, and frame-flushing, the CFA core keeps its s_axis_video_tready input low. Afterwards, the core should assert s_axis_video_tready automatically. Is m_axis_video_tready low? If so, the CFA core cannot send data downstream, and the internal FIFOs are full.
m_axis_video_tvalid stuck low, the downstream core is not receiving data	• No data is generated during the first two lines of processing. • If the programmed active number of pixels per line is radically smaller than the actual line length, the core drops most of the pixels waiting for the (s_axis_video_tlast) End-of-line signal. Check the ERROR register.
Generated SOF signal (m_axis_video_tuser0) signal misplaced.	Check the ERROR register.
Generated EOL signal (m_axis_video_tlast) signal misplaced.	Check the ERROR register.
Data samples lost between Upstream core and the CFA core. Inconsistent EOL and/ or SOF periods received.	• Are the Master and Slave AXI4-Stream interfaces in the same clock domain? • Is proper clock-domain crossing logic instantiated between the upstream core and the CFA core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the CFA ACLK pin lower than the reported Fmax reached?
Data samples lost between Downstream core and the CFA core. Inconsistent EOL and/ or SOF periods received.	• Are the Master and Slave AXI4-Stream interfaces in the same clock domain? • Is proper clock-domain crossing logic instantiated between the upstream core and the CFA core (Asynchronous FIFO)? • Did the design meet timing? • Is the frequency of the clock source driving the CFA ACLK pin lower than the reported Fmax reached?

If the AXI4-Stream communication is healthy, but the data seems corrupted, the next step is to find the correct configuration for this core.

Other Interfaces

Table C-3 describes how to troubleshoot third-party interfaces.

Table C-3: **Troubleshooting Third-Party Interfaces**

Symptom	Solution
Severe color distortion or color-swap when interfacing to third-party video IP.	Verify that the color component logical addressing on the AXI4-Stream TDATA signal is in according to Data Interface in Chapter 2. If misaligned: In HDL, break up the TDATA vector to constituent components and manually connect the slave and master interface sides. In EDK, create a new vector for the slave side TDATA connection. In the MPD file, manually assign components of the master-side TDATA vector to sections of the new vector.
Severe color distortion or color-swap when processing video written to external memory using the AXI-VDMA core.	Unless the particular software driver was developed with the AXI4-Stream TDATA signal color component assignments described in Data Interface in Chapter 2 in mind, there are no guarantees that the software correctly identifies bits corresponding to color components. Verify that the color component logical addressing TDATA is in alignment with the data format expected by the software drivers reading/writing external memory. If misaligned: In HDL, break up the TDATA vector to constituent components, and manually connect the slave and master interface sides. In EDK, create a new vector for the slave side TDATA connection. In the MPD file, manually assign components of the master-side TDATA vector to sections of the new vector.

Application Software Development

Programmers Guide

The software API is provided to allow easy access to the TPG AXI4-Lite registers defined in Table 2-7. To utilize the API functions, the following two header files must be included in the user C code:

```
#include "TPG.h"
#include "xparameters.h"
```

The hardware settings of your system, including the base address of your TPG core, are defined in the `xparameters.h` file. The `TPG.h` file contains the macro function definitions for controlling the TPG core.

For examples on API function calls and integration into a user application, the drivers subdirectory of the pCore contains a file, `example.c`, in the `tpg_v4_00_a0_a/example` subfolder. This file is a sample C program that demonstrates how to use the TPG API.

Table D-1: **TPG Driver Function Definitions**

Function Name and Parameterization	Description
TPG_Enable (uint32 BaseAddress)	Enables a TPG instance.
TPG_Disable (uint32 BaseAddress)	Disables a TPG instance.
TPG_Reset (uint32 BaseAddress)	Immediately resets a TPG instance. The core stays in reset until the RESET flag is cleared.
TPG_ClearReset (uint32 BaseAddress)	Clears the reset flag of the core, which allows it to re-sync with the input video stream and return to normal operation.
TPG_FSync_Reset (uint32 BaseAddress)	Resets a TPG instance at the end of the current frame being processed, or immediately if the core is not currently processing a frame.
TPG_ReadReg (uint32 BaseAddress, uint32 RegOffset)	Returns the 32-bit unsigned integer value of the register. Read the register selected by RegOffset (defined in Table 2-10).

Table D-1: **TPG Driver Function Definitions *(Cont'd)***

Function Name and Parameterization	Description
TPG_WriteReg (uint32 BaseAddress, uint32 RegOffset, uint32 Data)	Write the register selected by RegOffset (defined in Table 2-10. Data is the 32-bit value to write to the register.
TPG_RegUpdateEnable (uint32 BaseAddress)	Enables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.
TPG_RegUpdateDisable (uint32 BaseAddress)	Disables copying double buffered registers at the beginning of the next frame. Refer to Double Buffering for more information.

Software Reset

Software reset reinitializes registers of the AXI4-Lite control interface to their initial value, resets FIFOs, forces `m_axis_video_tvalid` and `s_axis_video_tready` to 0. `TPG_Reset()` and `TPG_FSync_Reset ()` reset the core immediately if the core is not currently processing a frame. If the core is currently processing a frame calling `TPG_Reset()`, or setting bit 30 of the `CONTROL` register to 1 causes image tearing. After calling `TPG_Reset()`, the core remains in reset until `TPG_ClearReset()` is called.

Calling `TPG_FSync_Reset()` automates this reset process by waiting until the core finishes processing the current frame, then asserting the reset signal internally, keeping the core in reset only for 32 `ACLK` cycles, then deasserting the signal automatically. After calling `TPG_FSync_Reset()`, it is not necessary to call `TPG_ClearReset()` for the core to return to normal operating mode.

Note: Calling `TPG_FSync_Reset()` does not guarantee prompt, or real-time resetting of the core. If the AXI4-Stream communication is halted mid frame, the core does not reset until the upstream core finishes sending the current frame or starts a new frame.

Double Buffering

Most of the core's registers are double-buffered to ensure no image tearing happens if values are modified during frame processing. Values from the AXI4-Lite interface are latched into processor registers immediately after writing, and processor register values are copied into the active register set at the Start Of Frame (SOF) signal. Double-buffering decouples AXI4-Lite register updates from the AXI4-Stream processing, allowing software a large window of opportunity to update processing parameter values without image tearing.

If multiple register values are changed during frame processing, simple double buffering would not guarantee that all register updates would take effect at the beginning of the same frame. Using a semaphore mechanism, the `RegUpdateEnable()` and `RegUpdateDisable()` functions allows synchronous commitment of register changes. The TPG core starts using the updated double buffered values only if the `REGUPDATE` flag of the `CONTROL` register is set (1), after the next Start-Of-Frame signal (`s_axis_video_tuser0`) is received. Therefore, it is recommended to disable the register update before writing multiple double-buffered registers, then enable register update when register writes are completed.

Reading and Writing Registers

Each software register that is defined in Table 2-10 has a constant that is defined in `tpg.h` which is set to the offset for that register listed in Table D-2.

RECOMMENDED: *It is recommended that the application software uses the predefined register names instead of register values when accessing core registers, so future updates to the TPG drivers which may change register locations does not affect the application dependent on the TPG driver.*

Table D-2: **Predefined Constants Defined in tpg.h**

Constant Name Definition	Value	Target Register
TPG_CONTROL	0x0000	CONTROL
TPG_STATUS	0x0004	STATUS
TPG_ERROR	0x0008	ERROR
TPG_IRQ_EN	0x000C	IRQ_ENABLE
TPG_VERSION	0x0010	VERSION
TPG_ACTIVE_SIZE	0x0020	ACTIVE_SIZE
TPG_PATTERN_CONTROL	0x0100	PATTERN_CONTROL
TPG_MOTION_SPEED	0x0104	MOTION_SPEED
TPG_CROSS_HAIRS	0x0108	CROSS_HAIRS
TPG_ZPLATE_HOR_CONTROL	0x010C	ZPLATE_HOR_CONTROL
TPG_ZPLATE_VER_CONTROL	0x0110	ZPLATE_VER_CONTROL
TPG_BOX_SIZE	0x0114	BOX_SIZE
TPG_BOX_COLOR	0x118	BOX_COLOR
TPG_STUCK_PIXEL_THRESH	0x11C	STUCK_PIXEL_THRESH
TPG_NOISE_GAIN	0x120	NOISE_GAIN

Additional Resources

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

http://www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

http://www.xilinx.com/company/terms.htm.

For a comprehensive listing of Video and Imaging application notes, white papers, reference designs and related IP cores, see the *Video and Imaging Resources* page at:

http://www.xilinx.com/esp/video/refdes_listing.htm#ref_des.

Solution Centers

See the Xilinx Solution Centers for support on devices, software tools, and intellectual property at all stages of the design cycle. Topics include design assistance, advisories, and troubleshooting tips.

References

These documents provide supplemental material useful with this user guide:

1. UG761 AXI Reference Guide.
2. UG932 KC724 GTX Transceiver Characterization Board User Guide
3. Vivado Design Suite Migration Methodology Guide (UG911)
4. Vivado™ Design Suite user documentation

Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

See the *IP Release Notes Guide* (XTP025) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
12/18/2011	1.0	Initial Xilinx release of Product Guide.

Notice of Disclaimer