JPEG Encoder IP Core
Setup Guide
on
Xilinx Vivado

VISENGI

Hardware & Software Engineering

www.visengi.com

Revision History

Rev.	Date	Description
1.0	2014-11-20	Initial documentation
1.1	2015-08-06	Describe AXI interfaces' endianness. Update optional AXI4-ST output width.

Table of Contents

VISENGI

Hardware & Software Engineering

VISENGI S.L.
Address: c. Juan de Herrera, 24 - 3 D
Santander – 39002 – SPAIN
EU VAT#: ESB39736509

Web: www.visengi.com
Phone: (+34) 942 50 80 15
Email: support@visengi.com

www.visengi.com

Setting Up

The following guide describes how to set up the Xilinx Vivado drag'n'drop instance deliverables for **implementing VISENGI's JPEG Encoder IP core on Xilinx Vivado projects**.

Requirements

- Xilinx Vivado (WebPack or Full Editions)

List of deliverables

Three versions of the same JPEG Encoder IP core are delivered, the only differences being on the buses used on the Data I/O Interface (for Pixel Input and Coded JPG Output):

- **Folder "jpeg_encoder_v1_00_a":** JPEG Encoder Block Design instance files, with one AXI4-Lite interface for configuration, and one AXI3 interface for Pixel Input and JPG Output, to include the IP core into Xilinx Vivado's IP library and be able to instantiate it into designs.

- **Folder "jpeg_encoder_st_v1_00_a":** JPEG Encoder Block Design instance files, with one AXI4-Lite interface for configuration, one AXI4-Stream interface for Pixel Input and another AXI4- Stream interface for JPG Output, to include the IP core into Xilinx Vivado's IP library and be able to instantiate it into designs.

- **Folder "jpeg_encoder_stin_v1_00_a":** JPEG Encoder Block Design instance files, with one AXI4-Lite interface for configuration, one AXI4-Stream interface for Pixel Input and one AXI3 interface for JPG Output, to include the IP core into Xilinx Vivado's IP library and be able to instantiate it into designs.

The AXI3 interfaces contain DMA logic to enable connecting them directly to high speed buses to external memory (or directly to a memory controller port) for data I/O.

On the other hand, AXI-4 Stream interfaces are meant usually to be connected as part of a pixel processing pipeline, for example.

More in-depth information on these interfaces follows in the next pages.

Evaluation limits

The main purpose of these files is to show how simple it is to embed a JPEG encoder into any design.

As such, please note that the delivered folders are missing the JPEG Encoder netlist itself. Hence, the design can be fully connected/implemented but not simulated, nor synthesized into an FPGA bitstream.

The licensed version of the IP core consists of the same files, but with an added netlist, allowing bitstream generation without changes to the Block Design system implemented here.

Setting up the deliverables

In order to design JPEG systems with Xilinx Vivado's Block Design GUI tool, the accompanying folders must be extracted to a user folder and added to a project's IP repository by following these steps:

1) Open Xilinx Vivado SW and either click "Create New Project" or "Open Project" if one already exists for the target Xilinx board.

2) Once the project is created/open, go to "Project Settings" in the "Tools" menu .

3) On the "Project Settings" window click on the "IP" icon and then on "Add Repository..."

4) Select the folder that contains VISENGI's Xilinx IP description folders (jpeg_encoder_*) and click Ok .

5) The IP components in the repository ("JPEG Encoder ...") should appear at the bottom, click Ok .

Please note that <u>the folder</u> where the jpeg_encoder_* subfolders were extracted <u>must not be deleted</u> or the IP Cores will not be accessible (Vivado does not copy them into its IP library).

Instantiation

In order to instantiate the JPEGE IP core into a Xilinx Block Design, the following steps should be followed:

1. Open Xilinx Vivado SW, and then the project where the JPEGE is to be integrated.
2. Select "Open Block Design" (or "Create Block Design" if none exists)
3. Once the block design is open, right click on the background and click on "Add IP..."
4. In the search box that appears, type "VISENGI" and the JPEG Encoders will appear as "JPEG Encoder (AXI IO)" for the AXI only I/O interface, "JPEG Encoder (AXI4-ST IO)" for AXI4-Stream only I/O interfaces, and "JPEG Encoder (AXI4-ST In, AXI Out)" for AXI4-Stream pixel input and AXI data output interfaces
5. Double click the desired JPEG Encoder or drag-and-drop it onto the block design's background
6. The new instance will now appear in the design without any connections.

AXI vs AXI4-Stream Interfaces

The three JPEG Encoder instances delivered feature the three most common interfacing possibilities of the IP core's Data I/O interface.

Firstly, let's review the interfaces that are the same in all versions:

- **Configuration Interface (S_AXI):** AXI4-Lite slave with a 32 bits interface to control all the necessary parameters of encoding through a small Configuration register set. This is a low speed interface.

- **Interrupt Interface (jpege_rdy_int_o):** A rising-edge interrupt is available, signaling when one image encoding has finished. It can also be checked through the Configuration register set.

Actually, the only interface that varies between the provided deliverables is the Pixel Input and Coded JPG Output, or **Data I/O, interface**:

1) For "**JPEG Encoder (AXI IO)**" IP core name (folder "jpeg_encoder_v1_00_a"):

The "M_AXI" bus (used to read **input pixels** and write the coded **JPG output**) is one **AXI3** Master, with a 64 bits data width, embedded DMA engine for direct connection to a memory controller, and user-set addressing parameters through the Configuration register set. Data is in **little-endian** format.

2) For "**JPEG Encoder (AXI4-ST IO)**" IP core name (folder "jpeg_encoder_st_v1_00_a"):

The **Pixel Input AXI4-Stream** Slave Interface is labeled "S_AXIS_PIX_IN" and can be connected to a Video IP AXI4-Stream Interface that feeds pixels in a row-wise manner (such as from an image sensor or video input). Data width is 24 bits.

The Coded **JPG Output AXI4-Stream** Master Interface is labeled "M_AXIS_JPG_OUT". It outputs bytes in a sequential manner in **big-endian** format. If they are written one after the other incrementally into a single file, the result is a .jpg file. Data width is 64 bits.
Check VISENGI's documentation for encapsulation of .jpg files as one MJPEG .avi video file.

3) For "**JPEG Encoder (AXI4-ST In, AXI Out)**" IP core name (folder "jpeg_encoder_stin_v1_00_a"):

This third version features a mix of the above:

The **Pixel Input AXI4-Stream** Slave Interface is labeled "S_AXIS_PIX_IN" and can be connected to a Video IP AXI4-Stream Interface that feeds pixels in a row-wise manner (such as from an image sensor or video input). Data width is 24 bits.

The "M_AXI" bus (write only for the Coded **JPG Output**) is **AXI3** Master, with a 64 bits data width, embedded DMA engine for direct connection to a memory controller, and user-set addressing parameters through the Configuration register set. Data is in **little-endian** format.

Interfacing

The following interfacing example will feature the AXI I/O variant of the JPEG Encoder, since a complete example is possible without other IP cores like a pixel input pipeline (from a camera).

In order to allow the JPEGE IP core to maintain its constant number of pixels encoded per clock cycle, the Data I/O interface must be connected to a high speed AXI slave with direct access to memory, capable of the required pixel bandwidth.

CPU/DDR Configuration:

In this **connection example** we will suppose a **Zynq 70X0** FPGA, that is: an FPGA with embedded ARM CPUs.

> NOTE: the JPEGE IP core is fully autonomous, the use of a CPU in this example is just for convenience. It is only employed to configure the compression parameters (frame width/height, quality, DMA addresses, etc) through the AXI4-Lite interface. However, a soft-core CPU (such as Microblaze) or even a simple FSM, can be used for the same purpose (on non-ARM enabled FPGAs).

As stated, the CPU is supposed to be the "ZYNQ7 Processing System" (ARM) which should already appear as a block in the Block Design in the "Diagram" tab.

Double click on it to open the "Re-customize IP" window and:

- In the **"PS-PL Configuration"** section, open the **"GP Master AXI interface"** drop-down and make sure that the "M AXI GP0 interface" is marked. This is where the AXI4-Lite Configuration interface (S_AXI) will go.

- In the same section, open the **"HP Slave AXI interface"** drop-down and make sure that "S AXI HP0 interface" is marked (Important: if another interface is used by user-logic, for performance reasons, make sure that interfaces 0 and 2 are the ones enabled, and not other combinations). Open it and make sure that "DATA WIDTH" is 64.

- In the **"Interrupts"** section, make sure that there is a mark on the checkbox labeled "Fabric Interrupts", open the drop-down, and then open the item labeled **"PL-PS Interrupt Ports"**, then make sure that the checkbox labeled "IRQ_F2P[15:0]" is marked. This is where the JPEG Encoder Interrupt output (jpege_rdy_int_o) will connect to the CPU.

Click OK to close this window and for the changes to take effect.

> NOTE: On a non-ARM enabled FPGA, the S_AXI AXI4-Lite low speed bus should be connected to a soft-core CPU, or a suitable FSM, for configuration; whereas the AXI3 Data I/O interface should be connected to a DDR Memory Controller.

Interconnection:

Now all the necessary endpoints should be available in the Vivado Block Design's "Diagram" tab. To properly connect the IP core these instructions should be followed.

To create the connections between ports/buses: click on the source port (short line coming out of the block with the indicated port name) and drag it to the target port, a routed line will appear connecting both.

- The **"ARESETN"** input port must be connected to an active-low reset source.

  On a Zynq device this could be the "peripheral_aresetn" output of a Xilinx "Processor System Reset" block.

- The **"ACLK"** input port must be connected to a rising-edge active clock source.

  On a Zynq device this could be, for example, port FCLK0.

- The **"jpege_rdy_int_o"** port is a rising-edge active interrupt signal.

  On a Zynq device, connect it to the IRQ_F2P port.

- The **"S_AXI"** bus (used to access the JPEGE's Configuration Registers) is a standard AXI4-Lite interface of data width 32 bits.

  On a Zynq device it can be directly connected to the PS' M_AXI_GPn port. If there are multiple AXI4-Lite devices connected to that bus, a Xilinx "AXI Interconnect" block will be necessary to mux them all into a single M_AXI_GPn bus.

- The **“m_axi”** bus (used to read input pixels and write the coded JPG file) is a standard AXI3 interface of data width 64 bits, which can be directly connected to a memory controller, since it already embeds the necessary DMA engine.

> On a Zynq device, connect it to the S_AXI_HP0 port (AXI High Performance), through a Xilinx "AXI Interconnect" block (even if it's the only master, to allow different frequencies at each side).

NOTE: “S_AXI_HP*” buses all connect the FPGA to the CPU's shared DDR memory controller. However, if there are other memory controllers on the FPGA side, they may be used instead, in order to ensure the maximum throughput for both the ARM CPU and JPEGE IP core.

Address Editor:

Finally, click on the "Address Editor" tab and open the drop-down tree until finding an **"jpeg_encoder_*"** item, then right click and select **"Auto Assign Address"**.

The automatically assigned addresses correspond to these JPEGE items:

- Within the **"processing_system7_*"** leaf, find the item **"jpeg_encoder_* S_AXI"** : this is the assigned memory range to access the Configuration register set of the JPEGE IP core.

- Within the **"jpeg_encoder_*"** leaf, find the item **"processing_system7_* S_AXI_HP0"**: here the total memory range should have been set, usually from "0x0000_0000" to "0x3FFF_FFFF" for 1GB systems. The actual address range used by the JPEGE is set later in the Configuration register set.

In order to check that the block design is correctly connected, go to the Tools menu and click on "Validate Design" or hit F6.

The JPEG encoding system is ready!

For any question on how to best interface this IP core for your application, please do not hesitate to send an email to info@visengi.com