H.264 Encoder IP Core Setup Guide on Xilinx Vivado

VISENGI

Hardware & Software Engineering

www.visengi.com

Revision History

Rev.	Date	Description
1.0	2014-11-03	Initial documentation
1.1	2014-11-20	Extended Instantiation and Interfacing
1.2	2015-10-27	Added mixed AXI and AXI4-ST interface version
1.3	2016-10-27	Changed AXI4-ST endianness to little-endian. Correct typos, add H264E-I netlists.

Table of Contents

Setting Up....4

Instantiation....6

Interfacing....8

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 H264 Encoder IP core on Xilinx Vivado projects**.

Requirements

- Xilinx Vivado (WebPack or Full Editions)

List of deliverables

Three versions of the same H264E IP core are delivered for each of its two variants (Intra only or Predictive), the only difference between versions being the bus used on the Data I/O Interface (for Pixel Input and Coded Video Output):

- **Folder "h264_encoder_p_v1_00_a":** H264E-P Encoder (High 4:4:4 Predictive Profile) Block Design instance files, with AXI4-Lite for the configuration interface, AXI3 for Auxiliary Reconstructed interface and AXI3 interfaces for Pixel Input and Coded Output.

- **Folder "h264_encoder_p_st_v1_00_a":** H264E-P Encoder (High 4:4:4 Predictive Profile) Block Design instance files, with AXI4-Lite for the configuration interface, AXI3 for Auxiliary Reconstructed Interface, and AXI4-Stream interface for Pixel Input and Coded Output.

- **Folder "h264_encoder_p_stin_v1_00_a":** H264E-P Encoder (High 4:4:4 Predictive Profile) Block Design instance files, with AXI4-Lite, AXI3 for Auxiliary Reconstructed Interface and Coded Output, and AXI4-Stream interface only for Pixel Input.

- **Folder "h264_encoder_i_v1_00_a":** H264E-I Encoder (CAVLC 4:4:4 Intra Profile) Block Design instance files, with AXI4-Lite for the configuration interface and AXI3 interfaces for Pixel Input and Coded Output.

- **Folder "h264_encoder_i_st_v1_00_a":** H264E-I Encoder (CAVLC 4:4:4 Intra Profile) Block Design instance files, with AXI4-Lite for the configuration interface and AXI4-Stream interface for Pixel Input and Coded Output.

- **Folder "h264_encoder_i_stin_v1_00_a":** H264E-I Encoder (CAVLC 4:4:4 Intra Profile) Block Design instance files, with AXI4-Lite for the configuration interface, AXI3 for Coded Output, and AXI4-Stream interface only for Pixel Input.

Please note the H264E-P has an additional AXI3 bus over H264E-I: the Auxiliary Reconstructed interface (or m_axi_aux), which is used to read/write to memory reference pictures, so they can be used to encode next frames' differences (P-frames).

Evaluation limits

The main purpose of these files is to show how simple it is to embed a powerful H264 video encoder into any design.

As such, please note that the delivered folders are missing the H264 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 H264 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 (h264_encoder_*) and click Ok .

5) The IP components in the repository ("H264 Encoder ...") should appear at the bottom, click Ok .

Please note that <u>the folder</u> where the h264_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 H264E IP core into a Xilinx Block Design, the following steps should be followed:

1. Open Xilinx Vivado SW, and then the project where the H264 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 H264 Encoders will appear
5. Double click the desired H264 Encoder (with AXI or with AXI4-Stream interfaces) 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 H264E-P instances delivered feature the most common interfacing possibilities of the IP core's Data I/O interface.

Firstly, let's review the interfaces that are the same in both 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 (frame_int_o):** A rising-edge interrupt is available, signaling when a frame has been processed and when video compression has finished. It can also be checked and/or masked through the Configuration register set. It is recommended to use this interrupt signal, instead of the second interrupt available ("h264e_int_o"), as it also contains the same functionality.

- **H264E-P Reconstructed Interface (m_axi_aux):** AXI3 Master interface with a data width of 128 bits, for reading/writing reconstructed (i.e. decoded) frames. It embeds a DMA engine so that it can be directly interfaced to a memory controller. Its addressing parameters are controlled through the Configuration register set.

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

1) "**H264 Encoder (P) (AXI IO)**" and "**H264 Encoder (I) (AXI IO)**", also referred to as H264E-P and H264E-I:

The "M_AXI" bus (used to read **input pixels** and write the coded **H.264 video output**) is one **AXI3/4** Master, with a 128 bits data width (**little-endian**), embedded DMA engine for direct connection to a memory controller, and user-set addressing parameters through the Configuration register set.

2) "**H264 Encoder (P) (AXI4-ST IO)**" and "**H264 Encoder (I) (AXI4-ST IO)**":

The **Pixel Input AXI4-Stream** Slave Interface is named "S_AXIS_PIX_IN" and can be connected to a Video IP AXI4-Stream Interface that feeds packed 24 bpp YCbCr pixels in a row-wise manner (such as from an image sensor or video input). Data width is 128 bits and little endian.

The Coded **H.264 Video Output AXI4-Stream** Master Interface is named "M_AXIS_264_OUT". It outputs words of 16 bytes in a sequential manner and **little-endian** byte order. If they are written one after the other incrementally into a single file, the result is a .264 video file. Data width is 128 bits.

Check VISENGI's technical support for encapsulation as .mp4, .avi or .mkv video files.

3) "**H264 Encoder (P) (AXI4-ST In, AXI Out)**" and "**H264 Encoder (I) (AXI4-ST In, AXI Out)**":

This third version features a mix of the above:

The **Pixel Input AXI4-Stream** Slave Interface is named "S_AXIS_PIX_IN" and can be connected to a Video IP AXI4-Stream Interface that feeds packed 24 bpp YCbCr pixels in a row-wise manner (such as from an image sensor or video input). Data width is 128 bits and little endian.

The "M_AXI" bus (write only for the Coded **H.264 Video Output**) is **AXI3/4** Master, with a 128 bits data width (**little-endian**), embedded DMA engine for direct connection to a memory controller, and user-set addressing parameters through the Configuration register set.

NOTE: AXI3 interfaces are forward compatible with AXI4.

Interfacing

The following interfacing scheme is recommended for maximum performance, since bottlenecks in the Data or Reconstructed AXI interfaces are about the only way to slow down the H264 Encoder.

In order to allow the H264 IP core to maintain its constant 5.2 pixels encoded per clock cycle throughput, these interfaces must be connected to high speed AXI slaves with direct access to memory.

CPU/DDR Configuration:

In this **connection example** we will suppose a **Zynq 7000** FPGA, that is: an FPGA with embedded ARM CPUs.

> NOTE: the H264E 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" and "S AXI HP2 interface" are marked (Important: please make sure that interfaces 0 and 2 are enabled, and not other combinations). Open them and make sure that "DATA WIDTH" is 64 for both.

- 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 H264 Interrupt output (frame_int_o) will connect to the CPU.

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

> <u>NOTE:</u> 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/Reconstructed interface/s should be connected to a DDR Memory Controller.

<u>Interconnection:</u>

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.

First of all Add the IP corresponding to the H.264 Encoder IP core. In this case we will be using the one with default AXI interfaces (i.e. h264_encoder_p).

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.

- It is recommended to use the **"frame_int_o"** rising-edge active interrupt signal, as it also contains the "h264e_int_o" interrupt functionality.

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

- The **"S_AXI"** bus (used to access the H264E'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_aux"** bus (used to read and write reference/decoded frames for P-encoding) is a standard AXI3 interface of data width 128 bits, which can be directly connected to a memory controller, since it already embeds the necessary DMA engine.

On a Zynq device, for maximum performance, it is recommended to 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 and to adapt the data width to the S_AXI_HP's 64 bits).

- The **"m_axi"** bus (used to read input pixels and write the coded video file) is also a standard AXI3 interface of data width 128 bits, which can be directly connected to a memory controller, since it already embeds the necessary DMA engine.

On a Zynq device, for maximum performance, the same recommendation holds except that it should be connected to the S_AXI_HP2 port through an AXI Interconnect (never use S_AXI_HP0 and _HP1, or _HP2 and _HP3 since each pair shares the same internal bus).

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/should be used instead, in order to ensure the maximum throughput, since the CPU's DDR is also shared with the ARM CPUs.

Avoiding bottlenecks on production systems

On a production system it is strongly recommended to use the second DDR3 533 MHz hard memory controller (on the FPGA side) available on Zynq devices, which is able to yield an extra 34 Gbps throughput with exclusive access.

Please note that the M_AXI and M_AXI_AUX interfaces DO NOT need to be connected to the same memory chips. That is, each bus reads/writes data independent of the other, so that they access entirely different memory areas.

Address Editor:

Finally, click on the “Address Editor” tab and open the drop-down tree until finding an **“h264_encoder_p_*”** item, then right click and select **“Auto Assign Address”**.

The automatically assigned addresses correspond to these H264E items:

- Within the **“processing_system7_*”** leaf, find the item **“h264_encoder_p_* S_AXI”** : this is the assigned memory range to access the Configuration register set of the H264E-P IP core.

- Within the **“h264_encoder_p_*”** 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 H264E is set later in the Configuration register set.

- Within the **“h264_encoder_p_*”** leaf, find the item **“processing_system7_* S_AXI_HP2”**: 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 H264E 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 H264 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