H.264 Encoder IP Core Setup Guide on Altera Quartus Qsys

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

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 Altera Quartus Qsys drag'n'drop instance deliverables for **implementing VISENGI's H264 Encoder IP core on Altera projects**.

Requirements

- Altera Quartus 14.0 or newer (Web or Subscription Edition).

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):

- **h264_encoder_p_hw.tcl:** H264E-P Encoder (High 4:4:4 Predictive Profile) Quartus Qsys instance file, with AXI4-Lite for the configuration interface, AXI3 for Auxiliary Reconstructed interface and AXI3 interfaces for Pixel Input and Coded Output.

- **h264_encoder_p_st_hw.tcl:** H264E-P Encoder (High 4:4:4 Predictive Profile) Quartus Qsys instance file, with AXI4-Lite for the configuration interface, AXI3 for Auxiliary Reconstructed Interface, and AXI4-Stream interface for Pixel Input and Coded Output.

- **h264_encoder_p_stin_hw.tcl:** H264E-P Encoder (High 4:4:4 Predictive Profile) Quartus Qsys instance file, with AXI4-Lite, AXI3 for Auxiliary Reconstructed Interface and Coded Output, and AXI4-Stream interface only for Pixel Input.

- **h264_encoder_i_hw.tcl:** H264E-I Encoder (CAVLC 4:4:4 Intra Profile) Quartus Qsys instance file, with AXI4-Lite for the configuration interface and AXI3 interfaces for Pixel Input and Coded Output.

- **h264_encoder_i_st_hw.tcl:** H264E-I Encoder (CAVLC 4:4:4 Intra Profile) Quartus Qsys instance file, with AXI4-Lite for the configuration interface and AXI4-Stream interface for Pixel Input and Coded Output.

- **h264_encoder_i_stin_hw.tcl:** H264E-I Encoder (CAVLC 4:4:4 Intra Profile) Quartus Qsys instance file, 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 .tcl files 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 Qsys system implemented here.

Setting up the deliverables

In order to design H264 systems with Altera Quartus' Qsys GUI tool, the accompanying files "h264_encoder_*_hw.tcl" must be added to Altera's IP library by following these steps:

1. Locate the Altera Quartus SW installation folder (for example "C:\Program Files\Altera\15.0") .

2. Create a new sub-folder called "visengi" within Altera SW's "ip" subfolder (for example: "C:\Program Files\Altera\15.0\ip\visengi").

3. Copy the files "h264_encoder_*_hw.tcl" to the new "visengi" folder.

Now, upon reopening Quartus Qsys, the new IP will be parsed and part of the IP library.

Instantiation

In order to instantiate the H264E IP core into a Qsys design, the following steps should be followed:

1. Open Altera Quartus SW, and then the project where the H264 is to be integrated.
2. Open the Qsys tool (Tools menu) and its .qsys project of choice (or create a new one).
3. In the IP Library, go to "DSP" and "Video and Image Processing"
4. The IP core will appear as "H264 Encoder (P)" (for AXI interfaces), "H264 Encoder (P) AXI4-ST Input", and "H264 Encoder (P) AXI4-ST" (for AXI and AXI4-Stream interfaces) and the same for the H264E-I, as "H264 Encoder (I)".
5. Select the preferred one and then click on the "Add" button.
6. The new instance will now appear in the design without any connections.

AXI vs AXI4-Stream Interfaces

The three H264E 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 interrupts 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.

- **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 three 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 **Cyclone V SX** (or Arria V SX) 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 Nios II) or even a simple FSM, can be used for the same purpose (on non-ARM enabled FPGAs).

The CPU is supposed to be the "Arria V/Cyclone V Hard Processor System" already at the top of the Qsys "System Contents" tab.

Double click on it to open the Parameters window and, on the **"FPGA Interfaces"** tab:

- Under the **"AXI Bridges"** section, make sure that the "Lightweight HPS-to-FPGA interface width" is set to "32-bit". This is where the AXI4-Lite Configuration interface (S_AXI) will go.

- Under the **"FPGA-to-HPS SDRAM Interface"** section, click the "+" button to add a new interface, select type "AXI-3" and width "128". This fast-speed AXI Slave interface will be used for the "Reconstructed Interface (M_AXI_AUX)", since it connects directly to DDR memory.
If using the "H264 Encoder (P)" IP core (only AXI-3 interfaces) instance, then a second interface of the same type and width must be created, clicking again on the "+" button. This will be used for the "Data Interface (M_AXI)".

- Under the **"Interrupts"** section, make sure that there is a mark on the checkbox labeled "Enable FPGA-to-HPS Interrupts". This is where the H264 Interrupt output (frame_int_o) will connect to the CPU.

Close the Parameters window 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/Reconstructed interface/s should be connected to a DDR Memory Controller.

Interconnection:

Now all the necessary endpoints should be available in the Qsys "System Contents" 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 grayed out empty circles that make up the connections described, at the first column of the "System Contents" tab.

- The "reset_sink" Reset Input port must be connected to an active-low reset source.

  On Qsys, this could be the "clk_reset" Reset Output of a "Clock Source" block.

- The "clock_sink" Clock Input port must be connected to a rising-edge active clock source.

  On Qsys, this could be the "clk" Clock Output of a "Clock Source" block.

- The "frame_int_o" Interrupt Sender port must be connected to a rising-edge active interrupt receiver.

  On Qsys, this could be the "f2h_irq0" Interrupt Receiver of the "Arria V/Cyclone V Hard Processor System" block, the interrupt connections are done at the far-right side column of the "System Contents" tab.

- The "S_AXI" bus (used to access the H264E's Configuration Registers) is a standard AXI4-Lite interface of data width 32 bits.

  On Qsys, connect it to the low speed "h2f_lw_axi_master" bus of the "Arria

V/Cyclone V Hard Processor System" block. Multiple component may be sharing this bus, which is fine.

- 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 Qsys, for maximum performance, it is recommended to connect it to the "f2h_sdram0_data" AXI slave, which is directly connected to the CPU's shared DDR Memory Controller. For even better results, this could be wired to another, exclusive, DDR Memory Controller.

- 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 Qsys, for maximum performance, it is recommended to connect it to the "f2h_sdram1_data" AXI slave, which is directly connected to the CPU's shared DDR Memory Controller. For even better results, this could be wired to another, exclusive, DDR Memory Controller.

NOTE: "f2h_sdram*_data" 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 DDR 400 MHz 32 bits hard memory controller (on the FPGA side) available in Cyclone V S* devices, yielding 2 x 25.6 = 51.2 Gbps combined memory bandwidth.

Another option would be to upgrade to an Altera Arria V S* SoC device, which contains 4 hard-memory controllers of 32 bits and 533 MHz, sky-rocketing available bandwidth to 136 Gbps.

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 Map:

Finally, click on the "Address Map" Qsys tab and find the following intersections in the table:

- **Row "h264_encoder_p_*.S_AXI" and Column "hps_*.h2f_lw_axi_master"**: set here the assigned memory range to access the Configuration register set of the H264E-P IP core (remember that there is an added offset at run-time, which in the ARM on the SX FPGAs is 0xFF20_0000).

For example: typing "0x0001_1000" will make the address range become "0x0001_1000 – 0x0001_1fff" (hence, accessible on SW at 0xFF21_1000 to 0xFF21_1FFF).

- **Row "hps_*.f2h_sdram0_data" and Column "h264_encoder_p_*.M_AXI"**: set here the total memory range, by typing "0x0000_0000", which will become "0x0000_0000 – 0xFFFF_FFFF". The actual address range used by the H264E will be set in the Configuration register set.

- **Row "hps_*.f2h_sdram1_data" and Column "h264_encoder_p_*.M_AXI_AUX"**: set here the total memory range, by typing "0x0000_0000", which will become "0x0000_0000 – 0xFFFF_FFFF". The actual address range used by the H264E will be set in the Configuration register set.

To finalize, go to the File menu and Save, and then click on the button at the bottom-right labeled "Generate HDL....", accept the default settings and wait for the system to be validated and generated, if there are missing connections they will be reported.

When this is done, close the window and then click on the "Finish" button.

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

