AXI4 Streaming Interface Explained for Video I/O Purposes

ECE532
Kazi Sudipto Arif
Shariq Khalil Ahmed

Overview and Prerequisites

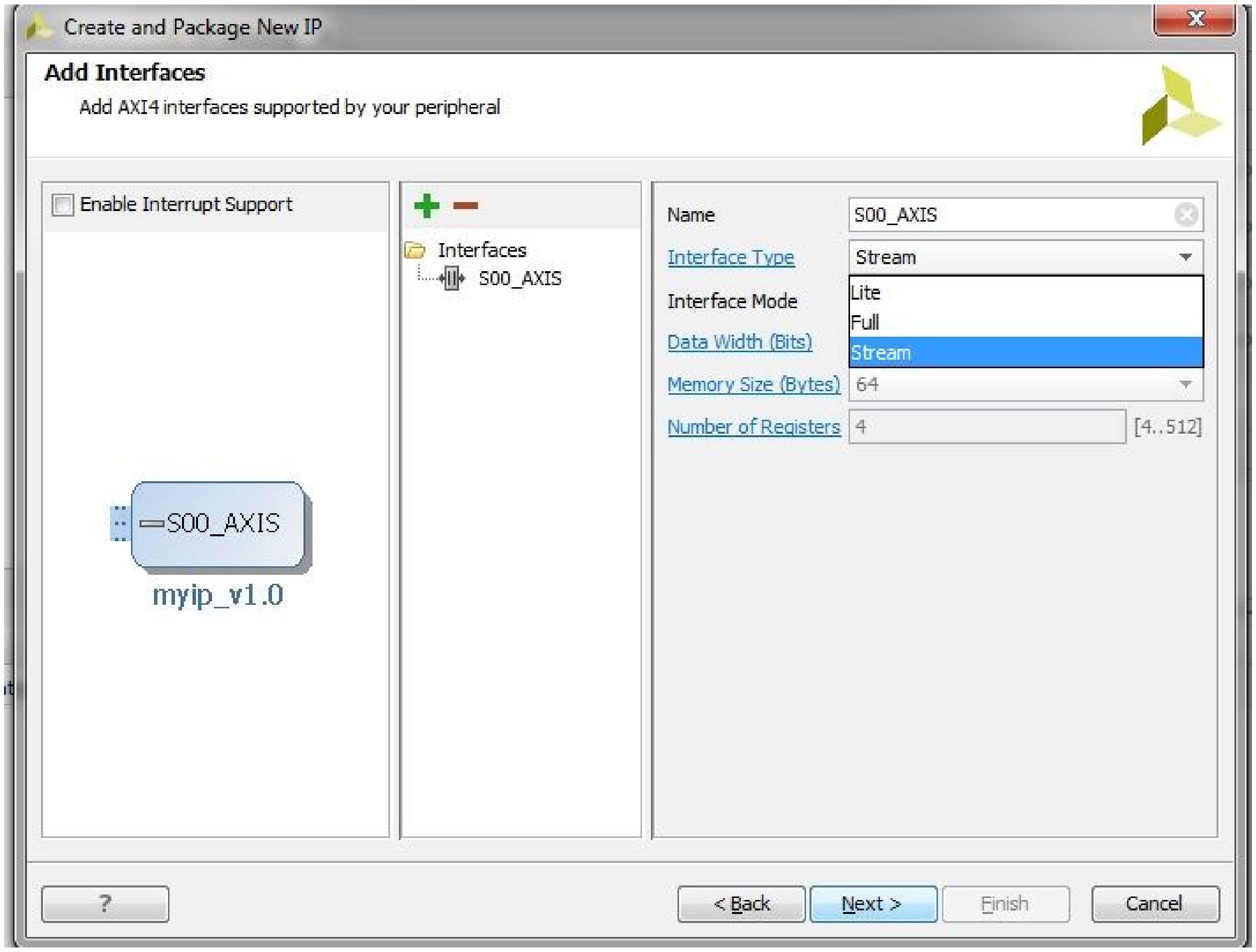
This document gives a general overview of the AXI4 Stream interface for use in custom IPs created via the IP Packager in Vivado 2016.2. In particular the focus is on streaming video via the AXI4 Stream interface. The official documentation outlining the interface for all purposes, not just video, can be found on Xilinx's website under the document code UG1037[1]. This document mainly references pages 96 to 100 of the Xilinx document and what it can mean for your design. This guide assumes that the user has completed all the tutorials posted on the course website, in particular the AXI slave peripheral tutorial.

The Digilent HDMI demo for the Nexys Video board was used as the base design to test these features and can be found on Digilent's website[2]. The video demo project allows for input data from HDMI In to be output via HDMI Out on the Nexys board. This is a very good project to use for video processing projects as the HDMI to RGB to AXI4_VIDEO_STREAM conversion is done for you. This project was originally created in Vivado 2015.3 and needs to be updated to whichever version of Vivado you are using. After that, the design needs to generate bitstream to create the new hardware platform. The board support files provided might not work for newer versions of Vivado SDK and might require regenerating the board support files from the newly generated hardware platform.

[1]https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pd
f
[2] https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-video-hdmi-demo/start

How to use AXI4 Stream Interface for Video

To create an IP with AXI4 Stream interface follow all instructions for creating a normal IP in the AXI slave tutorial on the course website up until the Add Interfaces window. At the Add Interface screen you have to specify Interface Type as Stream and Interface Mode as Slave or Master (See Figure 1). A Master interface outputs stream data while a slave interface takes stream data as input.

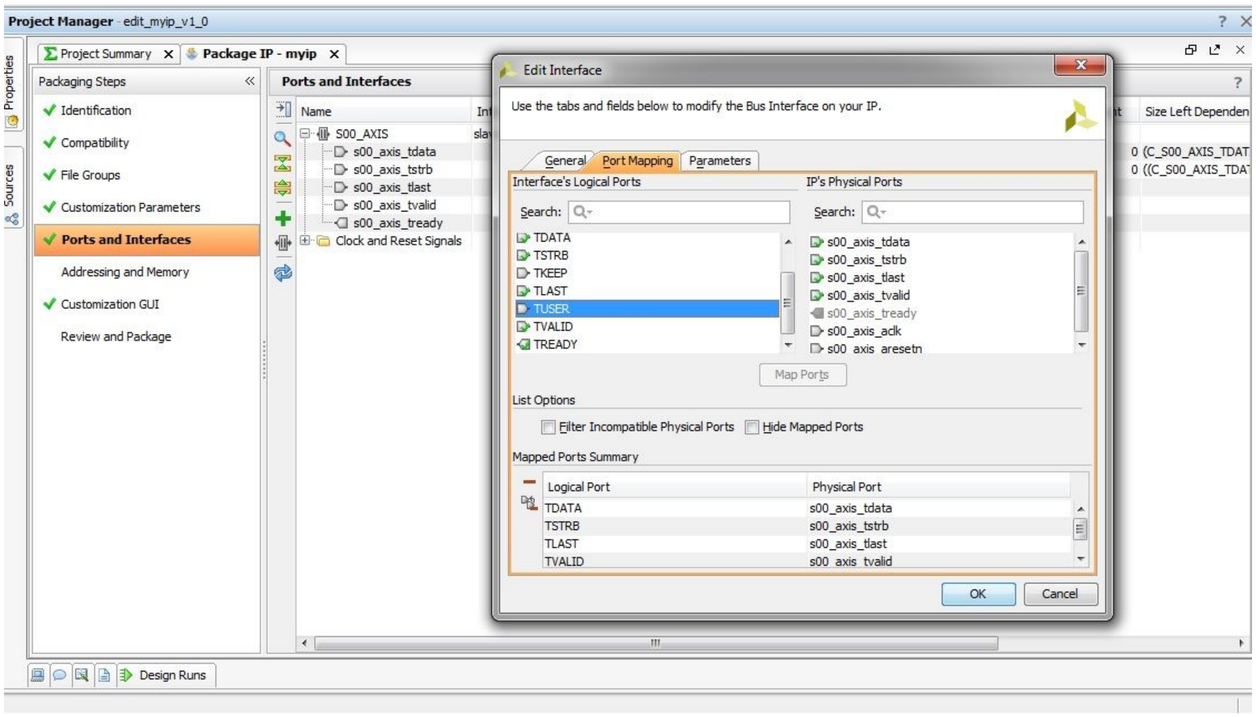


Name: S00_AXIS
Interface Type: Stream
Interface Mode
Data Width (Bits)
Memory Size (Bytes): 64
Number of Registers: 4 [4..512]

Lite
Full
Stream

Figure 1: Stream interface setup menu

The default AXIS stream interface is missing a signal necessary for streaming video for the video interfaces in the HDMI project demo. You will need to add a port labelled m_axis_video_tuser or s_axis_video_tuser in the top level module depending on if the interface is a master or slave respectively. Then you will need to add this to the interface bus by editing it in the Ports and Interfaces menu to map TUSER in the interface bus to that signal (see Figure 2).



Use the tabs and fields below to modify the Bus Interface on your IP.

Logical Port	Physical Port
TDATA	s00_axis_tdata
TSTRB	s00_axis_tstrb
TLAST	s00_axis_tlast
TVALID	s00_axis_tvalid

Figure 2: Ports and Interface Edit to add TUSER signal to interface

An example of an IP with a master AXI4 Stream Interface and a slave AXI4 Stream Interface is shown in Figure 3. It also has an AXI4 Lite Interface to act as a MicroBlaze peripheral. If you are using the HDMI demo project then your IP's slave_axis_clk and master_axis_clk signals need to be tied to the Video In to AXI4 Stream block's aclk signal. The AXIS slave_axis_resetn and slave_axis_resetn signals need to be tied to Processor System Reset's peripheral_resetn signal. These signals need to be connected to integrate your IP into the pre-existing HDMI project.

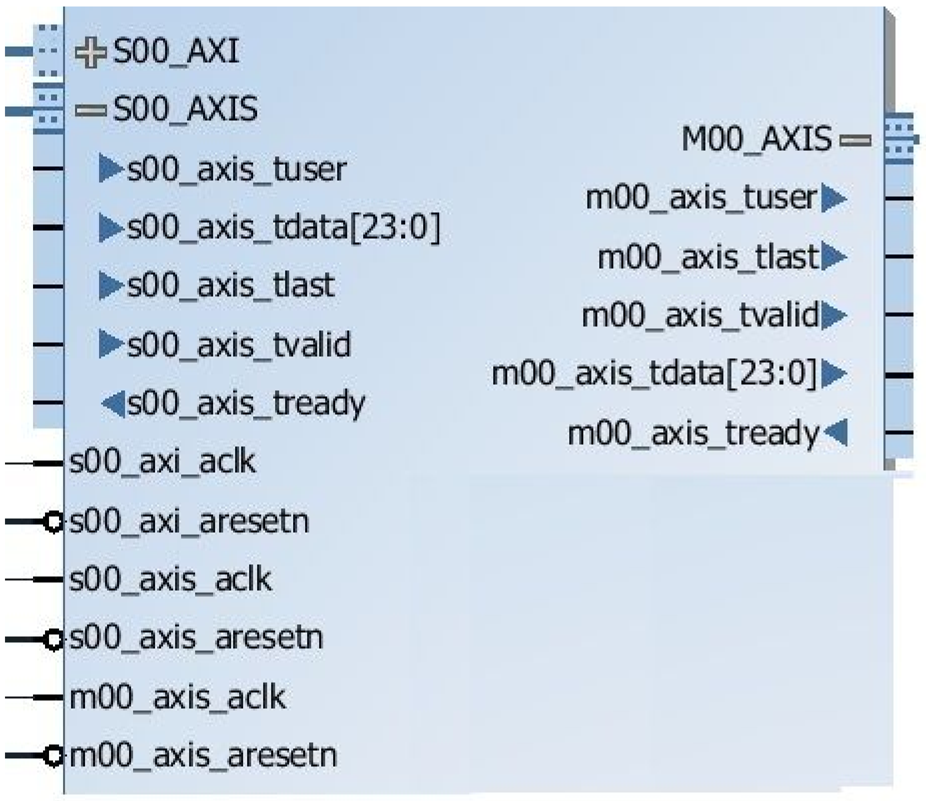


Figure 3: Example block with an AXIS Master and Slave port

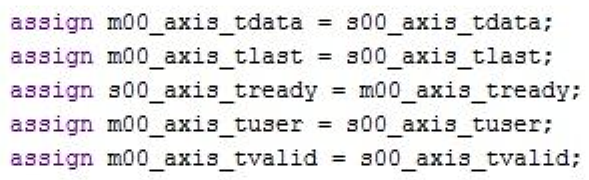
The AXI4 Stream Interface is a bus consisting of 5 signals. They are outlined below (see Table 1) along with what they represent when it comes to streaming video input/output.

Table 1: Summary of AXI4 Stream Bus

AXI4 Stream Signal Name	**Function**	**Width**	**Direction (Slave/Master)**
s/m_axis_video_tdata	Pixel Data	Depends on pixel data format (i.e. RGB is 24 bits)	IN/OUT
s/m_axis_video_tvalid	Pixel Valid Signal	1	IN/OUT
s/m_axis_video_tready	Pixel Ready	1	OUT/IN
s/m_axis_video_tlast	End of line	1	IN/OUT
s/m_axis_video_tuser	Start of Frame	1	IN/OUT

- The pixel data taken as input via *m_axis_video_tdata* can be of any data size and this has to be specified in the parameter C_S00_AXIS_TDATA_WIDTH inside the top level module of the custom IP you created. By default this is set to 32 and has to be modified to the width of your pixel data structure. The 8-bits per channel RGB specification for Xilinx is that the upper 8 bits are red, middle 8 bits are blue and lower 8 bits are green for a total pixel size of 24 bits.
- The *m_axis_video_tvalid* signal needs to be checked at all times when reading/writing pixel data to make sure that it is valid otherwise it should not be read.
- The *m_axis_video_tready* signal is an optional signal that can be used by the IP to let the block receiving or sending data from it to have a reference signal for stopping the flow of data. This can be done in instances where it is necessary to buffer data.
- The *m_axis_video_tuser* signal indicates the start of a frame. When this is asserted that means that the last pixel on the screen has been read and now the entire stream is starting from the first pixel and the next valid pixel data is for the first pixel. Note that this signal can be asserted for longer than a clock cycle so if acting as input to an FSM, make sure the FSM accounts for this.
- The *m_axis_video_tlast* signal representing end-of-line is asserted when the end of a line in the frame is reached. This would mean that the x-coordinate of the current pixel on the screen is equal to the width of the screen. When this is asserted the next line’s first pixel will be the next valid pixel sent to the IP. Note that this signal can be asserted for as long as a valid signal so if acting as input to an FSM, make sure the FSM accounts for this.

The pre-generated verilog modules for the AXI streaming interfaces use an FSM to handle reading and writing pixels. An easy way to avoid the complexity of the read-write logic of the stream interface is to simply tie the master and slave AXI stream signals to each other. This is useful in the case where the custom IP does not need to buffer the input stream. To this end it also makes sense to comment out the actual stream interface instantiation code provided by the IP Creation tool as you can then handle the pixel data yourself instead of relying on their read-write FSM. For example if your IP is simply analyzing a single pixel or operating on a single pixel at a time this makes sense. The following code snippet shows this:



```
assign m00_axis_tdata = s00_axis_tdata;
assign m00_axis_tlast = s00_axis_tlast;
assign s00_axis_tready = m00_axis_tready;
assign m00_axis_tuser = s00_axis_tuser;
assign m00_axis_tvalid = s00_axis_tvalid;
```

Figure 4: Code snippet showing tying the input and output stream signals

The above code snippet simply lets the pixel data pass through. If it is necessary for the pixel to be changed then m00_axis_tdata can be overwritten for HDMI output. This can be useful if your project requires altering the HDMI input (i.e. coloring over parts of the screen).