SISTEMI EMBEDDED

System Interconnect Fabric
Avalon-ST: Streaming Interface
Video out: Pixel Buffer DMA component

Federico Baronti

Last version: 20160503
Avalon Streaming Interface

• Low latency, high throughput, unidirectional point-to-point (from a source to a sink interface) data transfer (stream)

• Support for complex protocols:
  – Packet transfers with packets interleaved across multiple channels
  – Sideband (from source to sink) signalling of channels, errors, start and end of packets
Example: Avalon-MM and Avalon-ST
Some terminology

- **Symbol**: A symbol is the smallest unit of data. For most packet interfaces, a symbol is a byte. One or more symbols make up the single unit of data transferred in a cycle or a **beat**.
- **Beat**: A single cycle transfer between a source and a sink interface consisting of one or more symbols.
- **Channel**: A channel is a physical or logical path or link through which information passes between two ports.
- **Packet**: A packet is an aggregation of data and control info that are transmitted together.
  - A packet may contain a header to help routers and other network devices direct the packet to the correct destination.
  - The packet format is defined by the application, not by the Avalon specification.
## Avalon-ST Interface Signal Roles

<table>
<thead>
<tr>
<th>Signal Role</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>channel</strong></td>
<td>1–128</td>
<td>Source → Sink</td>
<td>The channel number for data being transferred on the current cycle. If an interface supports the channel signal, it must also define the maxChannel parameter.</td>
</tr>
<tr>
<td><strong>data</strong></td>
<td>1–4096</td>
<td>Source → Sink</td>
<td>The data signal from the source to the sink, typically carries the bulk of the information being transferred. The contents and format of the data signal is further defined by parameters.</td>
</tr>
<tr>
<td><strong>error</strong></td>
<td>1–256</td>
<td>Source → Sink</td>
<td>A bit mask used to mark errors affecting the data being transferred in the current cycle. A single bit in error is used for each of the errors recognized by the component, as defined by the errorDescriptor property.</td>
</tr>
<tr>
<td><strong>ready</strong></td>
<td>1</td>
<td>Sink → Source</td>
<td>Asserted high to indicate that the sink can accept data. ready is asserted by the sink on cycle (&lt;n&gt;) to mark cycle (&lt;n + readyLatency&gt;) as a ready cycle, during which the source may assert valid and transfer data. Sources without a ready input cannot be back pressured, and sinks without a ready output never need to back pressure.</td>
</tr>
<tr>
<td><strong>valid</strong></td>
<td>1</td>
<td>Source → Sink</td>
<td>Asserted by the source to qualify all other source to sink signals. On ready cycles where valid is asserted, the data bus and other source to sink signals are sampled by the sink, and on other cycles are ignored. Sources without a valid output implicitly provide valid data on every cycle that they are not being back pressured, and sinks without a valid input expect valid data on every cycle that they are not backpressuring.</td>
</tr>
</tbody>
</table>
Avalon-ST Interface Signal Roles (2)

<table>
<thead>
<tr>
<th>Signal Role</th>
<th>Width</th>
<th>Direction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>empty</td>
<td>1–8</td>
<td>Source → Sink</td>
<td>Indicates the number of symbols that are empty during cycles that contain the end of a packet. The empty signal is not used on interfaces where there is one symbol per beat. If <code>endofpacket</code> is not asserted, this signal is not interpreted.</td>
</tr>
<tr>
<td>endofpacket</td>
<td>1</td>
<td>Source → Sink</td>
<td>Asserted by the source to mark the end of a packet.</td>
</tr>
<tr>
<td>startofpacket</td>
<td>1</td>
<td>Source → Sink</td>
<td>Asserted by the source to mark the beginning of a packet.</td>
</tr>
</tbody>
</table>

- All transfers of an Avalon-ST connection are synchronous with the rising edge of the associated clock signal
- All outputs from a source interface to a sink interface, including the data, channel, and error signals, must be registered on the rising edge of clock
- Inputs to a sink interface do not have to be registered
- Registering signals only at the source provides for high frequency operation while eliminating back-to-back registers with no intervening logic
# Avalon-ST Interface Properties

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Default Value</th>
<th>Legal Values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>symbolsPerBeat</td>
<td>1</td>
<td>1–512</td>
<td>The number of symbols that are transferred on every valid cycle.</td>
</tr>
<tr>
<td>associatedClock</td>
<td>1</td>
<td>a clock interface</td>
<td>The name of the Avalon Clock interface to which this Avalon-ST interface is synchronous.</td>
</tr>
<tr>
<td>associatedReset</td>
<td>1</td>
<td>a reset interface</td>
<td>The name of the Avalon Reset interface to which this Avalon-ST interface is synchronous.</td>
</tr>
<tr>
<td>dataBitsPerSymbol</td>
<td>8</td>
<td>1–512</td>
<td>Defines the number of bits per symbol. For example, byte-oriented interfaces have 8-bit symbols. This value is not restricted to be a power of 2.</td>
</tr>
<tr>
<td>errorDescriptor</td>
<td>0</td>
<td>list of strings</td>
<td>A list of words that describe the error associated with each bit of the error signal. The length of the list must be the same as the number of bits in the error signal, and the first word in the list applies to the highest order bit. For example, “crc, overflow” means that bit[1] of error indicates a CRC error, and bit[0] indicates an overflow error.</td>
</tr>
<tr>
<td>firstSymbolInHighOrderBits</td>
<td>false</td>
<td>true, false</td>
<td>When true, the high-order symbol is driven to the MSB of the data interface. The highest-order symbol is labelled D0 in this specification.</td>
</tr>
<tr>
<td>maxChannel</td>
<td>0</td>
<td>0–255</td>
<td>The maximum number of channels that a data interface can support.</td>
</tr>
<tr>
<td>readyLatency</td>
<td>0</td>
<td>0–8</td>
<td>Defines the relationship between assertion and deassertion of the ready signal, and cycles which are considered to be ready for data transfer, separately for each interface.</td>
</tr>
</tbody>
</table>
Example of a Source-Sink connection
Data layout

- Example: data width = 32 bits;
  \[ \text{dataBitsPerSymbol} = 8 \]
  \[ \text{firstSymbolInHighOrderBits} = \text{true} \]
Data Transfer without Backpressure

• When the source interface wants to send data, it drives the *data* and the optional *channel* and *error* signals and asserts *valid*

• The sink interface samples these signals on the rising edge of the reference clock when *valid* is asserted

![Sink Interface sampling cycles](image)
Data Transfer with Backpressure (1)

- The sink indicates to the source that it is ready for an active cycle by asserting \textit{ready} for a single clock cycle. \textit{Cycles during which the sink is ready for data are called ready cycles}
- During a ready cycle, the source may assert valid and provide data to the sink. If it has no data to send, it deasserts valid and can drive data to any value
- Each interface that supports backpressure defines the \textit{readyLatency} parameter to indicate the number of cycles from the time that \textit{ready} is asserted until valid data can be driven
- If \textit{readyLatency} has a nonzero value, the source interface must consider cycle \textit{\langle n + \text{readyLatency} \rangle} to be a ready cycle if \textit{ready} is asserted on cycle \textit{\langle n \rangle}
- When \textit{readyLatency} = 0, data is transferred only when ready and valid are asserted on the same cycle. In this mode of operation, the source does not receive the sink's ready signal before it begins sending valid data. The source provides the data and asserts valid whenever it can and waits for the sink to capture the data and assert \textit{ready}. The source can change the data it is providing at any time. The sink only captures input data from the source when \textit{ready} and \textit{valid} are both asserted
- When \textit{readyLatency} \geq 1, the sink asserts \textit{ready} before the ready cycle itself. The source can respond during the appropriate cycle by asserting \textit{valid}. It may not assert valid during a cycle that is not a ready cycle
Data Transfer with Backpressure (2)

- $\text{readyLatency} = 4$
Data Transfer with Backpressure (3)

- $\text{readyLatency} = 0$

- $\text{readyLatency} = 1$
Packet Data Transfer (1)

- Three additional signals:
  - `startofpacket`, `endofpacket`, `empty`
- Both source and sink interfaces must include these additional signals
- No automatic adaptation (by system interconnect fabric)
Packet Data Transfer (2)

- `startofpacket` is required by all the interfaces supporting packet transfers and marks the active cycle containing the start of the packet. This signal is only interpreted when `valid` is asserted.
- `endofpacket` is required by all interfaces supporting packet transfer and marks the active cycle containing the end of the packet. This signal is only interpreted when `valid` is asserted.
- `empty` is optional and indicates the number of symbols that are empty during the cycles that mark the end of a packet. The sink only checks the value of the empty during active cycles that have `endofpacket` asserted.
  - The `empty` symbols are always the last symbols in data, those carried by the low-order bits when `firstSymbolInHighOrderBits` = true.
  - The empty signal is required on all packet interfaces whose data signal carries more than one symbol of data and have a variable length packet format.
  - The size of the empty signal in bits is `ceiling(log2(<symbols per cycle>))`. 
Packet Data Transfer (3)

- Transfer of a 17-byte packet \((\text{readyLatency} = 0)\)
- Data transfer occurs on cycles 2, 3, 5, 6, and 7, when both \textit{ready} and \textit{valid} are asserted
- During cycle 1, \textit{startofpacket} is asserted, and the first 4 bytes of packet are transferred
- During cycle 6, \textit{endofpacket} is asserted, and empty has a value of 3, indicating that this is the end of the packet and that 3 of the 4 symbols are empty
- D16 is transmitted over \text{data}[31:24] \((\text{firstSymbolInHighOrderBits} = \text{true})\)
Avalon-ST: Adapters

• Adapters are configurable Qsys/SOPC Builder components that are part of the streaming interconnect fabric. They are used to connect source and sink interfaces that do not match exactly.

• Qsys includes the following four adapters:
  – Data Format Adapter
  – Timing Adapter
  – Channel Adapter
  – Error Adapter

• If you connect mismatched Avalon-ST sources and sinks in Qsys without inserting adapters, Qsys generates error messages.
Example of Avalon-ST Component

- Pixel Buffer DMA Controller
  - The DMA controller uses its Avalon-MM master interface to read video frames from an external memory. Then, it sends the video frames out via the Avalon-ST interface. The controller’s Avalon-MM slave interface is used to read/write the controller’s internal registers by the processor.
Pixel Buffer DMA Controller (1)

- Avalon-MM Slave interface:

<table>
<thead>
<tr>
<th>Table 4. Pixel Buffer register map</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offset in bytes</td>
</tr>
<tr>
<td>0</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5. Status register bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit number</td>
</tr>
<tr>
<td>31 - 24</td>
</tr>
<tr>
<td>23 - 16</td>
</tr>
<tr>
<td>7 - 4</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>0</td>
</tr>
</tbody>
</table>
Pixel Buffer DMA Controller (1)

- The **Buffer** register holds the 32-bit address of the start of the memory buffer. This register is read-only, and shows the address of the first pixel of the frame currently being sent out via the Avalon-ST

- The **BackBuffer** register allows the start address of the frame to be changed under program control
  - To change the frame being displayed, the desired frame’s start address is first written into the **BackBuffer** register
  - Then, a second write operation is performed on the **Buffer** register. The value of the data provided in this second write operation is discarded. Instead, it interprets a write to the Buffer register as a request to swap the contents of the **Buffer** and **BackBuffer** registers.
  - **The swap does not occur immediately.** Instead, the swap is done after the Pixel **Buffer** reaches the last pixel associated with the current frame. In the meantime, bit S of the Status register will be set to 1. After the current frame is finished, the swap is performed and bit S is cleared.
module VGA_Pixel_Buffer ( 
  // Inputs 
  clk, 
  reset, 
  slave_address, 
  slave_byteenable, 
  slave_read, 
  slave_write, 
  slave_writedata, 
  master_address, 
  master_arbiterlock, 
  master_read, 
  // Bi-Directional 
  slave_readdata, 
  // Outputs 
  master_readdata, 
  master_readdatavalid, 
  master_waitrequest, 
  stream_data, 
  stream_startofpacket, 
  stream_endofpacket, 
  stream_empty, 
  stream_valid 
);
Pixel Buffer DMA Controller (2)

FIFO: Image_Buffer

Avalon-MM Master IF

Avalon-MM Slave IF

FSM manages Avalon-MM Master transfers to keep FIFO level between almost-full and almost-empty.

VGA_Pixel_Buffer.v
References

- Altera, “Avalon Interface Specifications,” *mnl_avalon_spec.pdf*
  - 5. Avalon Streaming Interfaces