SISTEMI EMBEDDED
AA 2013/2014

System Interconnect Fabric

Avalon-ST: Streaming Interface

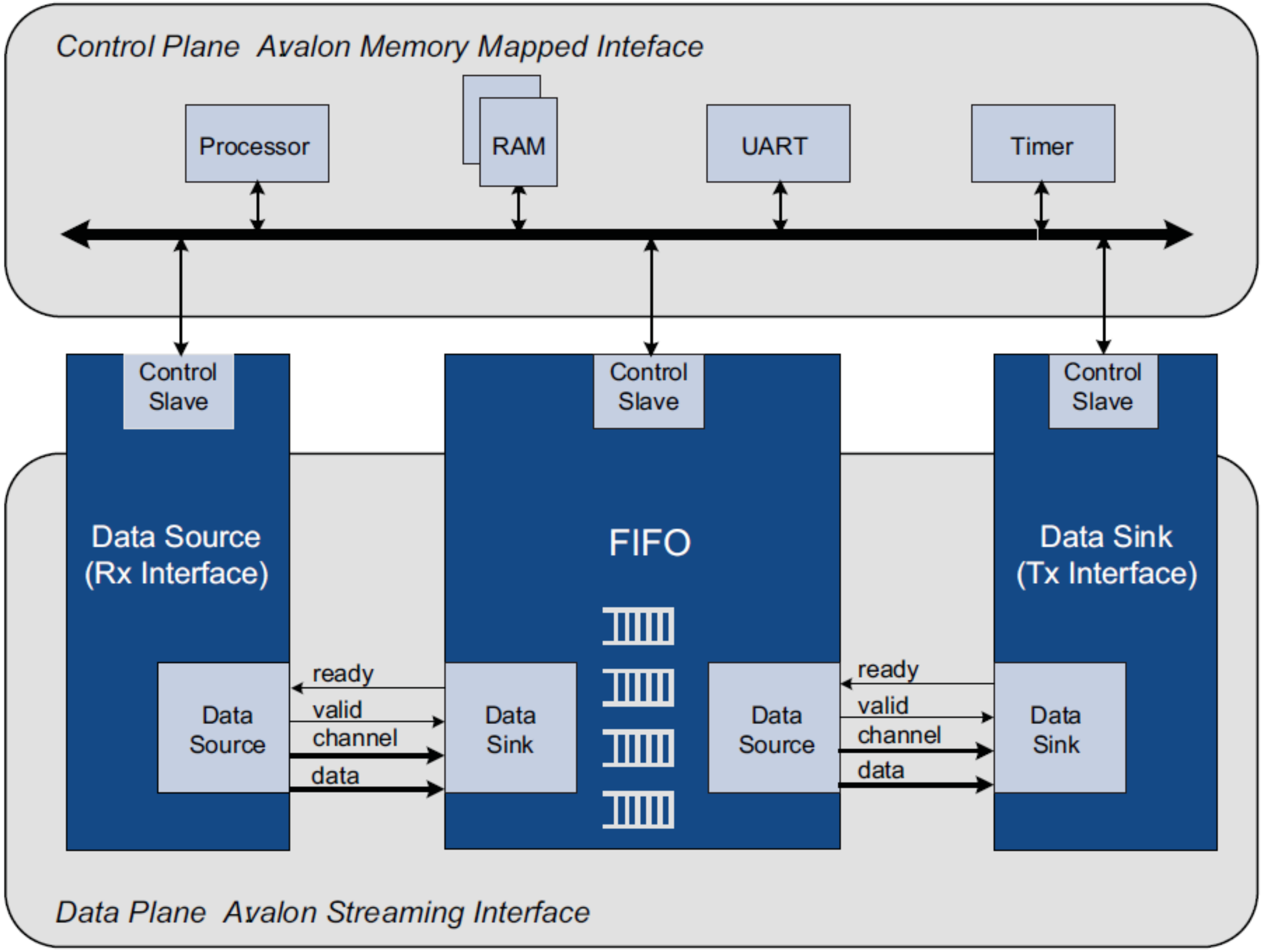
Video out: Pixel Buffer DMA component

Federico Baronti

Avalon Streaming Interface

- Low latency, high throughput, unidirectional point-to-point data transfer (stream)
- Support for complex protocols:
  - Packet transfers with packets interleaved across multiple channels
  - Data bursting
  - Sideband (from *source* to *sink*) signalling of channels, errors, start and end of packets

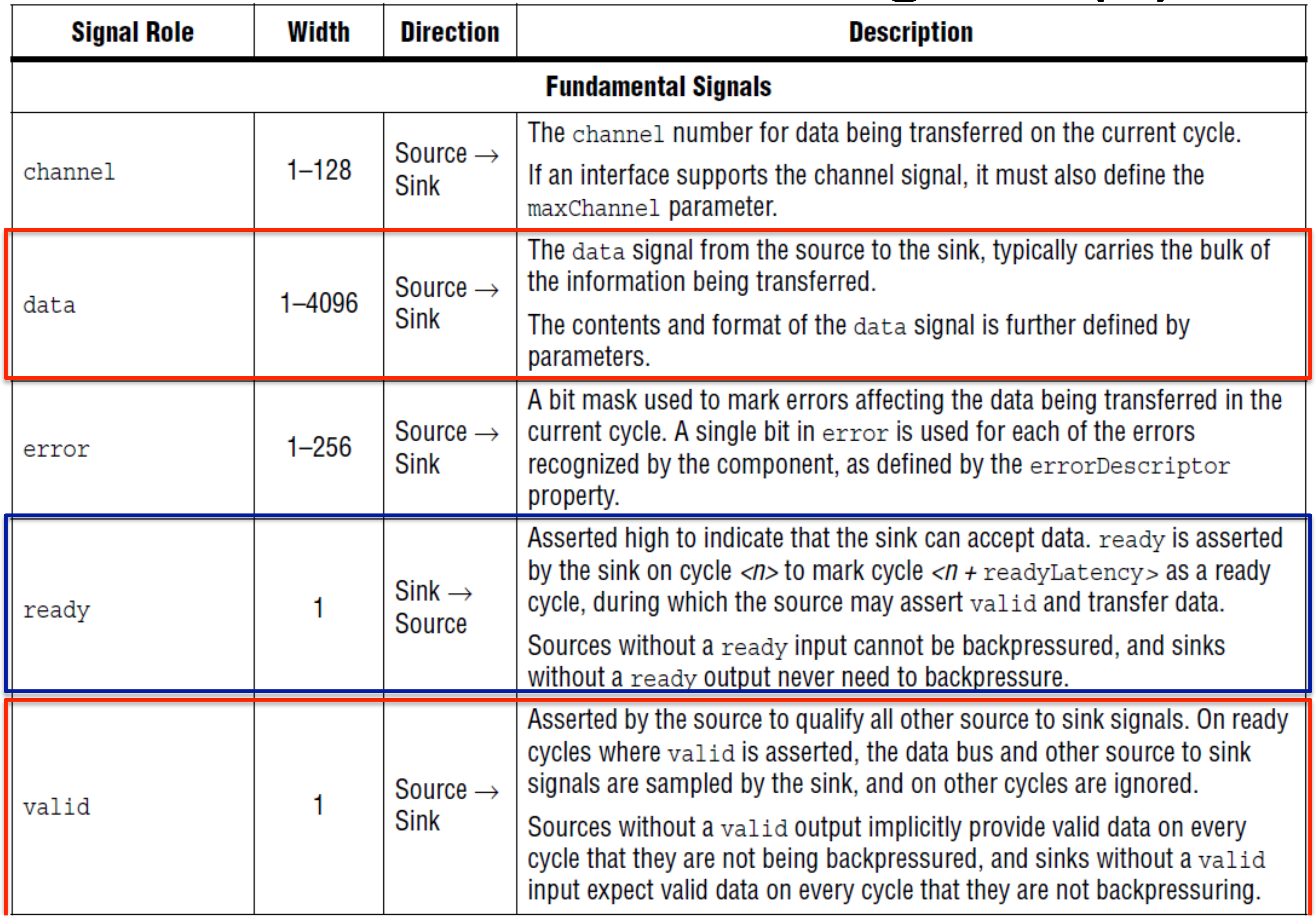
Example: Avalon-MM and Avalon-ST



Some terminology

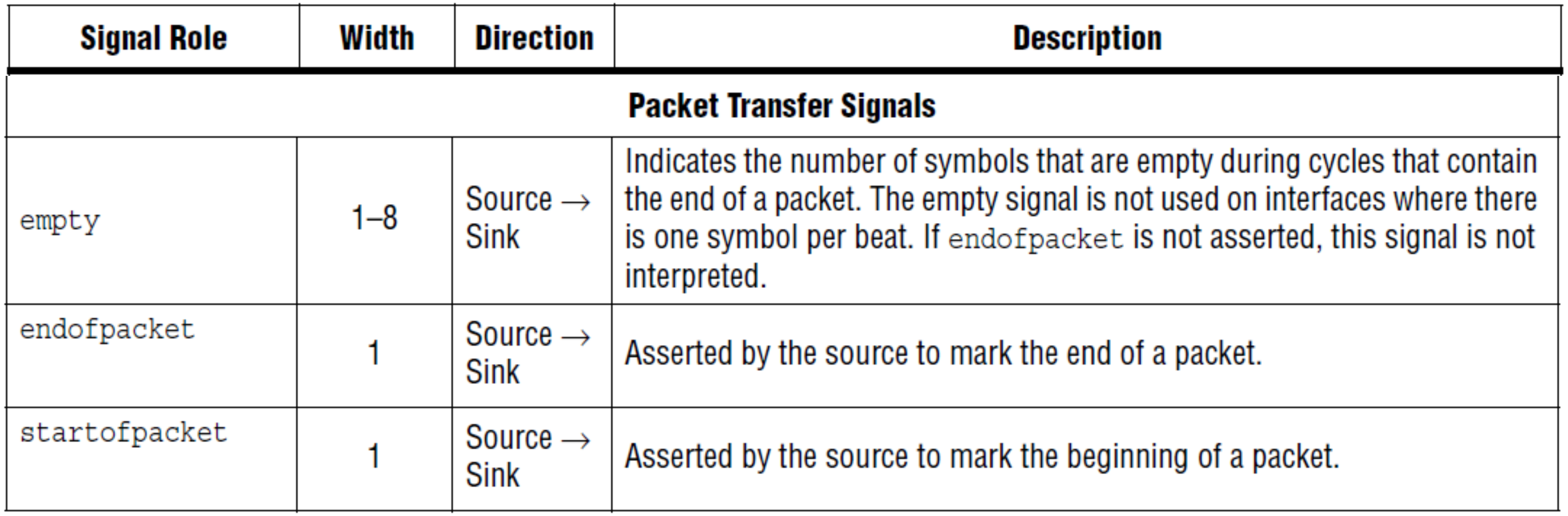
- **Symbol**: A symbol is the smallest unit of data. For most packet interfaces, a symbol is a byte. One or more symbols make up the single unit of data transferred in a cycle or **beat**
- **Beat**: A single cycle transfer between a source and a sink interface made up of one or more symbols
- **Channel**: 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 Signals (1)



Signal Role	Width	Direction	Description
Fundamental Signals			
`channel`	1–128	Source → Sink	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.
`data`	1–4096	Source → Sink	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.
`error`	1–256	Source → Sink	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.
`ready`	1	Sink → Source	Asserted high to indicate that the sink can accept data. `ready` is asserted by the sink on cycle <*n*> to mark cycle <*n* + `readyLatency`> as a ready cycle, during which the source may assert `valid` and transfer data. Sources without a `ready` input cannot be backpressured, and sinks without a `ready` output never need to backpressure.
`valid`	1	Source → Sink	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 backpressured, and sinks without a `valid` input expect valid data on every cycle that they are not backpressuring.

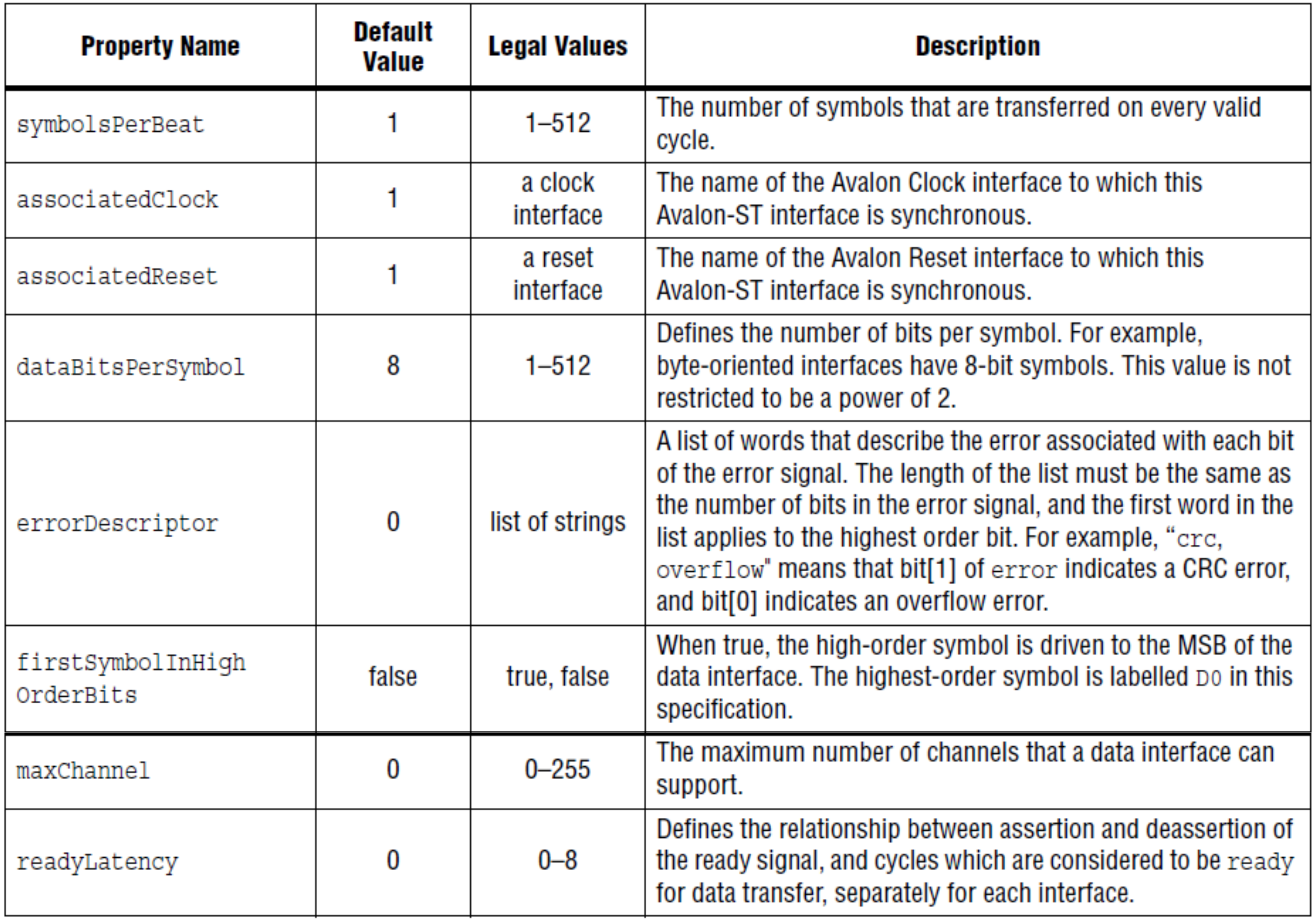
Avalon-ST Interface Signals (2)



Signal Role	Width	Direction	Description
Packet Transfer Signals			
`empty`	1–8	Source → Sink	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 `endofpacket` is not asserted, this signal is not interpreted.
`endofpacket`	1	Source → Sink	Asserted by the source to mark the end of a packet.
`startofpacket`	1	Source → Sink	Asserted by the source to mark the beginning of a packet.

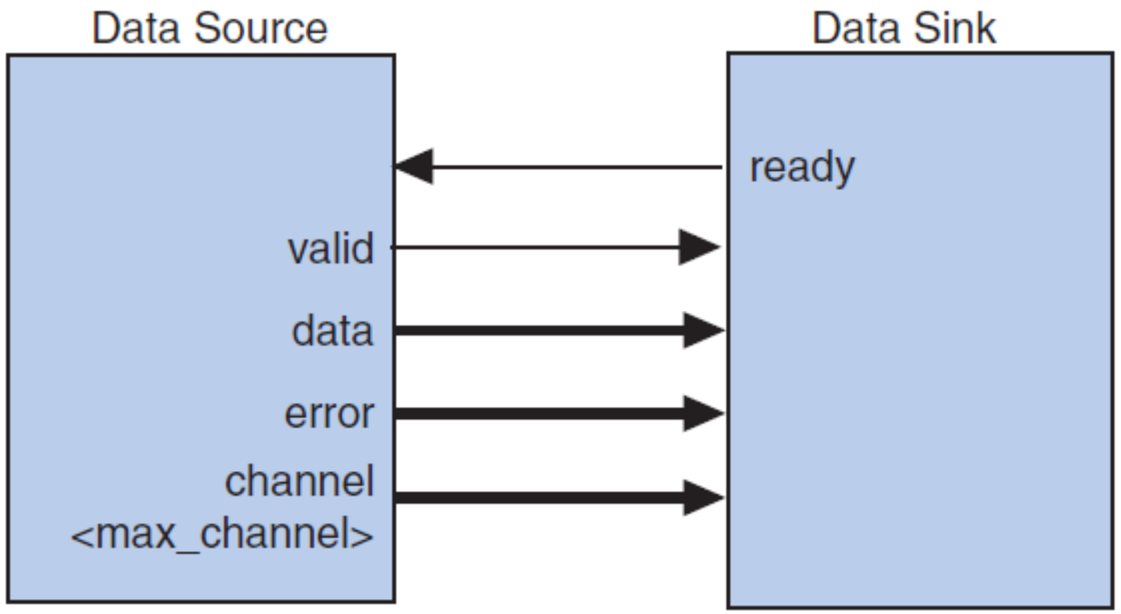
- All transfers of an Avalon-ST connection are synchronous with the rising edge of the associated clock signal
- All outputs from a source interface to a sink interface, including the data, channel, and error signals, must be registered on the rising edge of clock
- Inputs to a sink interface do not 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



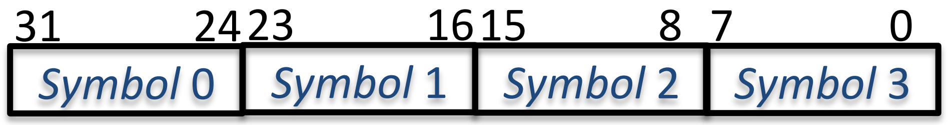
Property Name	Default Value	Legal Values	Description
`symbolsPerBeat`	1	1–512	The number of symbols that are transferred on every valid cycle.
`associatedClock`	1	a clock interface	The name of the Avalon Clock interface to which this Avalon-ST interface is synchronous.
`associatedReset`	1	a reset interface	The name of the Avalon Reset interface to which this Avalon-ST interface is synchronous.
`dataBitsPerSymbol`	8	1–512	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.
`errorDescriptor`	0	list of strings	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.
`firstSymbolInHighOrderBits`	false	true, false	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.
`maxChannel`	0	0–255	The maximum number of channels that a data interface can support.
`readyLatency`	0	0–8	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.

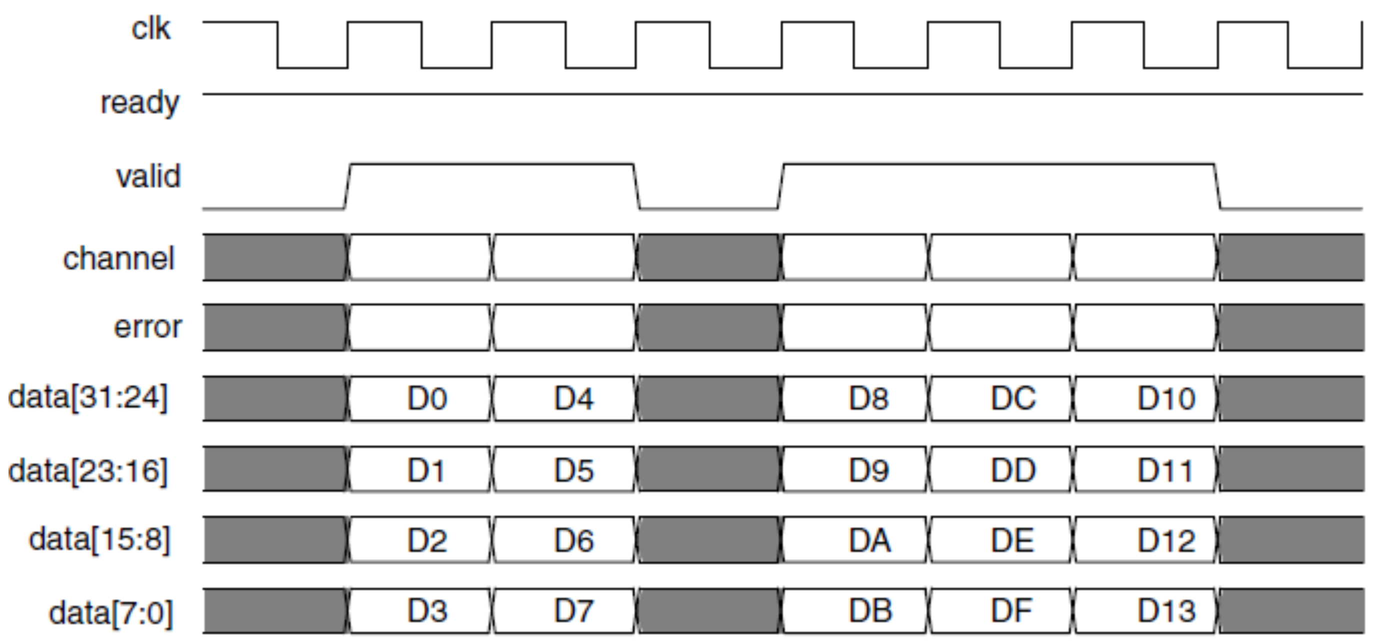
Example of Source-Sink connection



Data layout

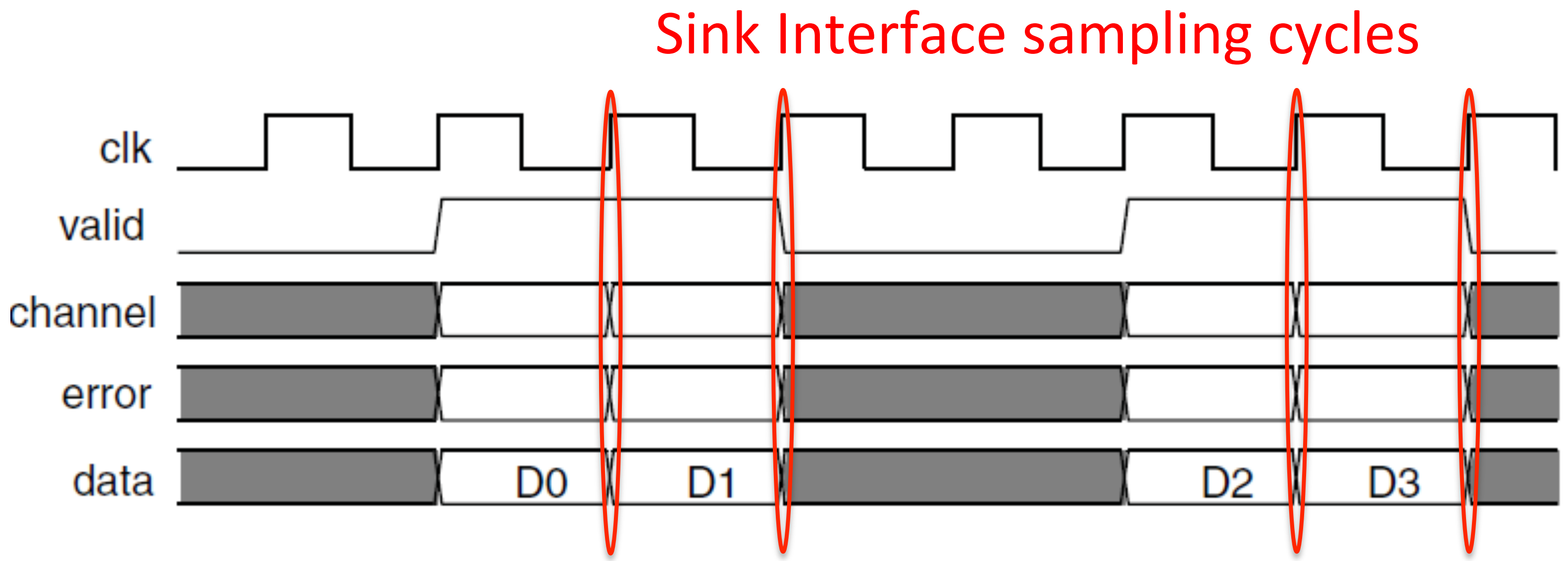
- Example: data width = 32 bits;
  dataBitsPerSymbol = 8
  firstSymbolInHighOrderBits = true





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

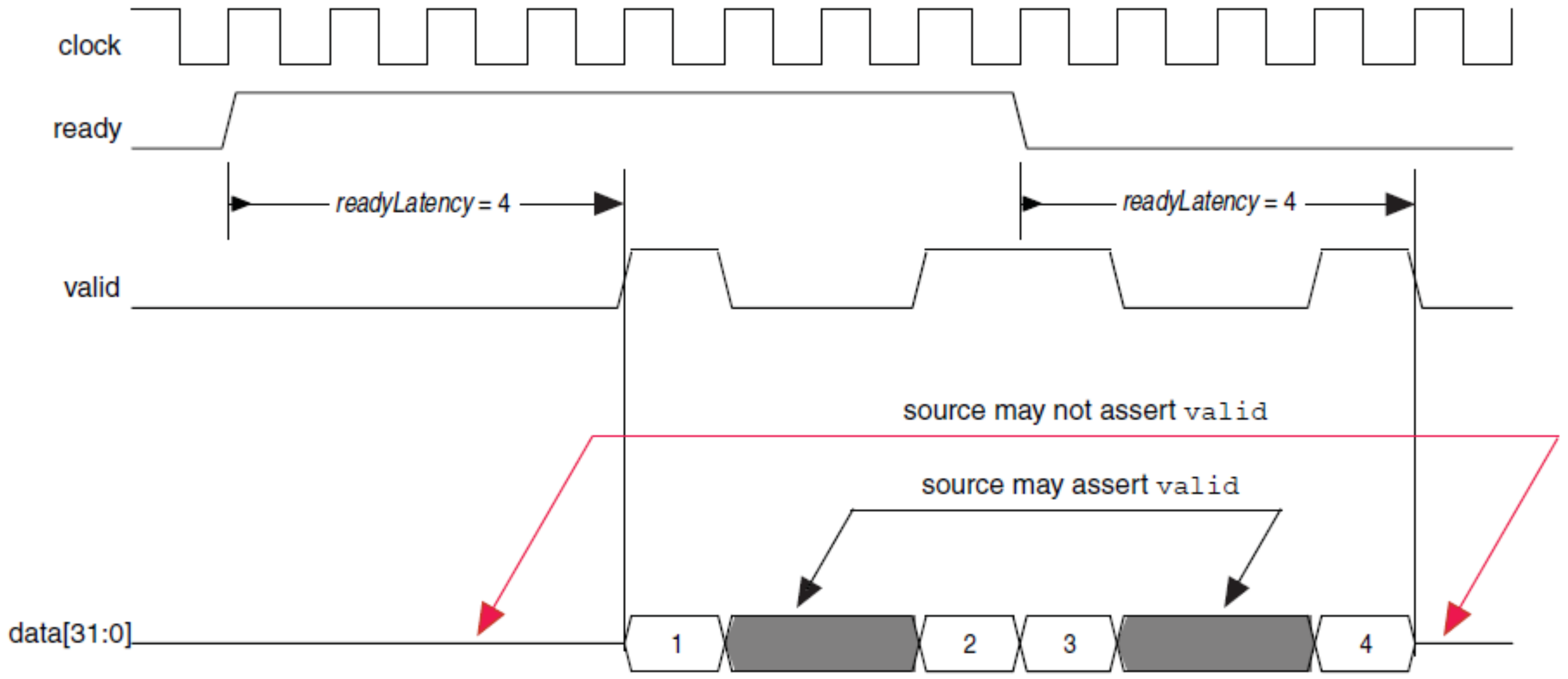


Data Transfer with Backpressure (1)

- The sink indicates to the source that it is ready for an active cycle by asserting *ready* for a single clock cycle. **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 *readyLatency* parameter to indicate the number of cycles from the time that *ready* is asserted until valid data can be driven
- If *readyLatency* has a nonzero value, the interface considers cycle <n + readyLatency> to be a ready cycle if *ready* is asserted on cycle <n>
- When *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 *ready*. The source can change the data it is providing at any time. The sink only captures input data from the source when *ready* and *valid* are both asserted
- When readyLatency >= 1, the sink asserts *ready* before the ready cycle itself. The source can respond during the appropriate cycle by asserting *valid*. It may not assert valid during a cycle that is not a ready cycle

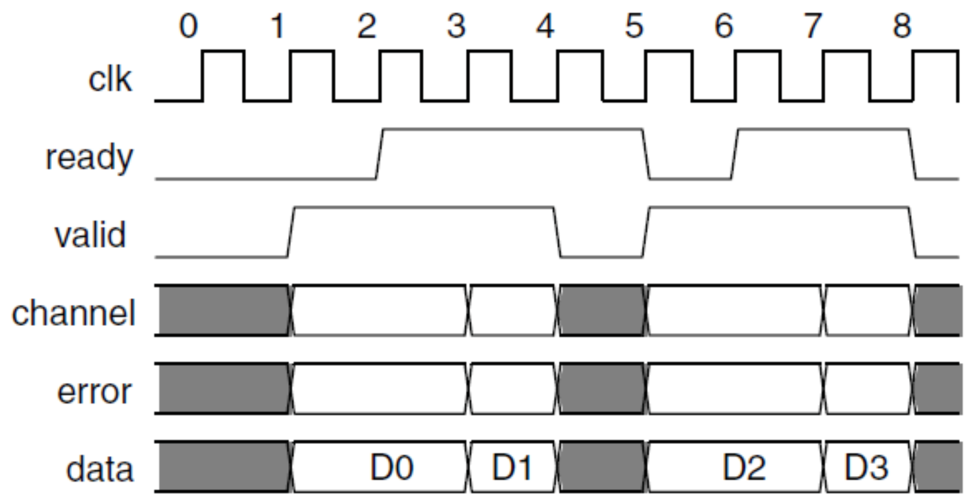
Data Transfer with Backpressure (2)

- *readyLatency* = 4

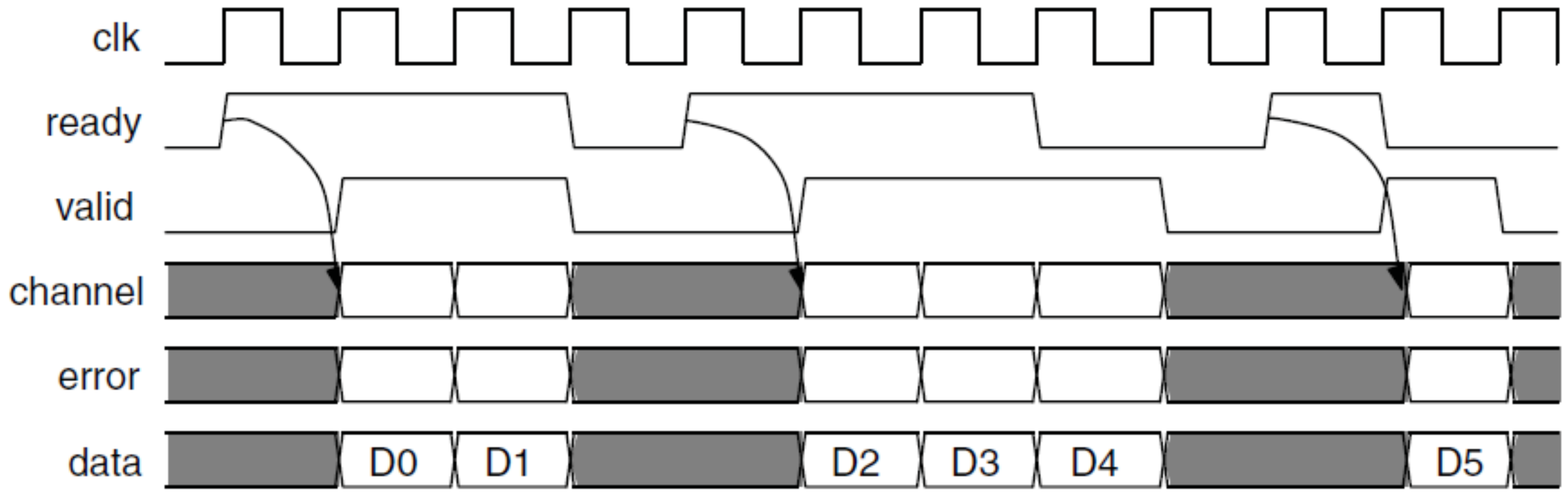


Data Transfer with Backpressure (3)

- *readyLatency* = 0

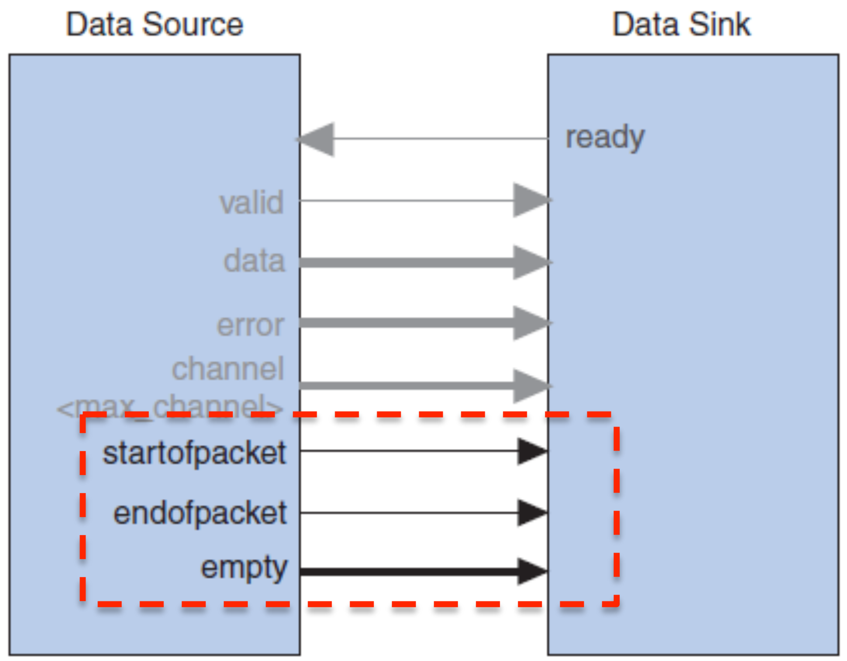


- *readyLatency* = 1



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)

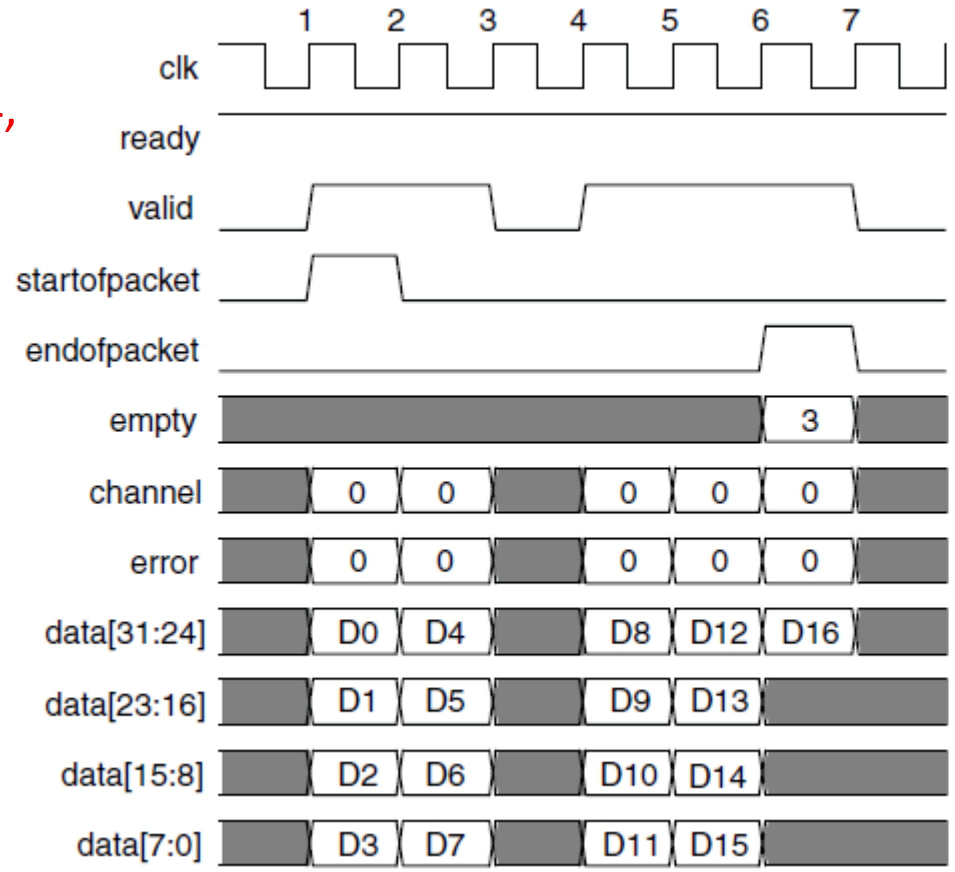


Packet Data Transfer (2)

- *startofpacket* is required by all 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($\log_2$(<*symbols per cycle*>))

Packet Data Transfer (3)

- Transfer of a 17-byte packet (*readyLatency* = 0)
- Data transfer occurs on cycles 1, 2, 4, 5, and 6, when both *ready* and *valid* are asserted
- During cycle 1, *startofpacket* is asserted, and the first 4 bytes of packet are transferred
- During cycle 6, *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 data[31:24] (*firstSymbolInHighOrderBits* = true)

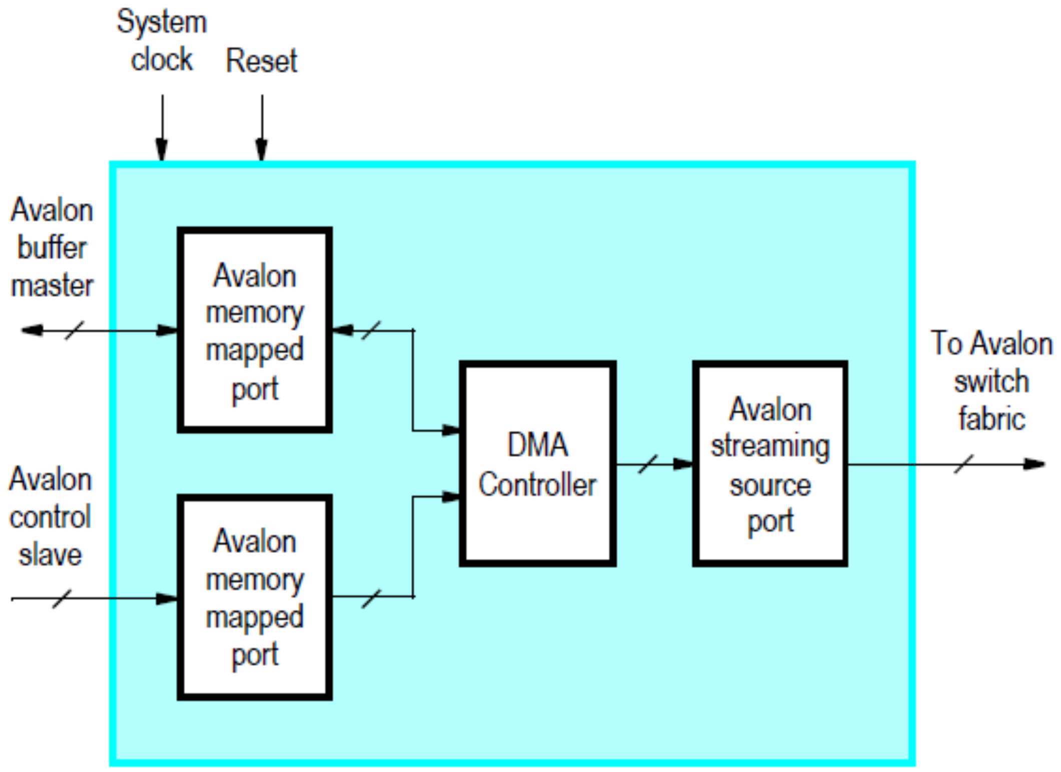


Avalon-ST: Adapters

- 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 exactly match
- **Qsys**/SOPC Builder 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**/SOPC Builder without inserting adapters, **Qsys**/SOPC Builder 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:

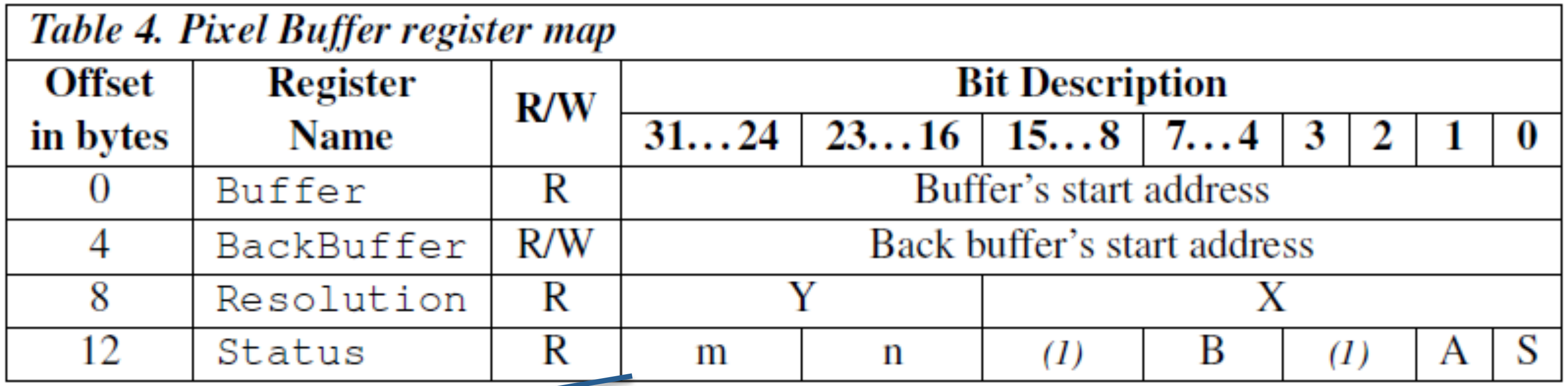


Table 4. Pixel Buffer register map

Offset in bytes	Register Name	R/W	Bit Description							
			31…24	23…16	15…8	7…4	3	2	1	0
0	`Buffer`	R	Buffer's start address							
4	`BackBuffer`	R/W	Back buffer's start address							
8	`Resolution`	R	Y		X					
12	`Status`	R	m	n	*(1)*	B	*(1)*		A	S

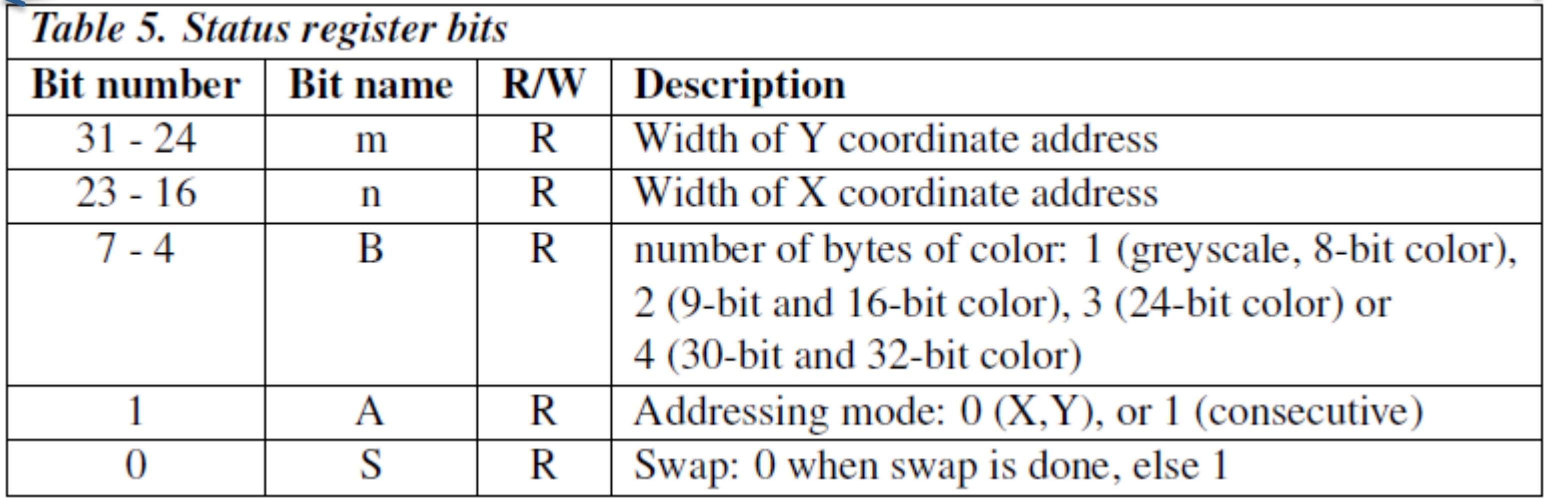


Table 5. Status register bits

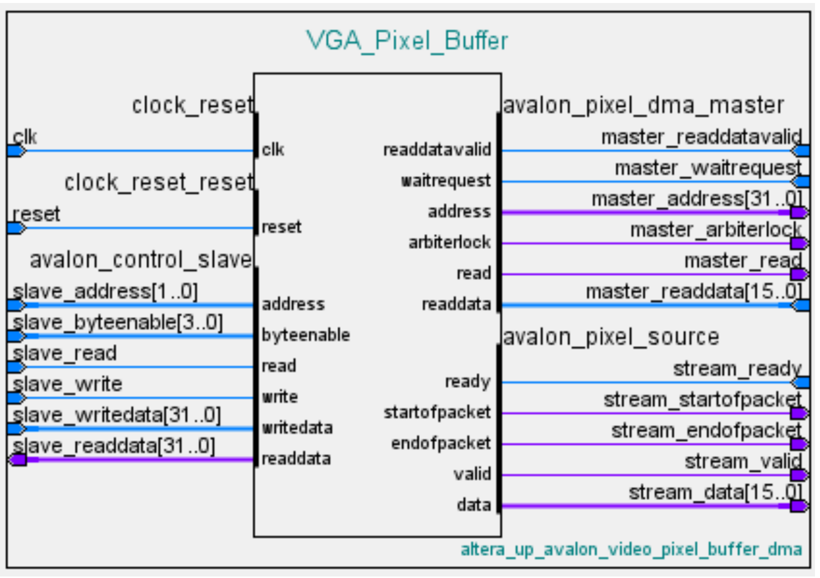
Bit number	Bit name	R/W	Description
31 - 24	m	R	Width of Y coordinate address
23 - 16	n	R	Width of X coordinate address
7 - 4	B	R	number of bytes of color: 1 (greyscale, 8-bit color), 2 (9-bit and 16-bit color), 3 (24-bit color) or 4 (30-bit and 32-bit color)
1	A	R	Addressing mode: 0 (X,Y), or 1 (consecutive)
0	S	R	Swap: 0 when swap is done, else 1

Pixel Buffer DMA Controller (1)

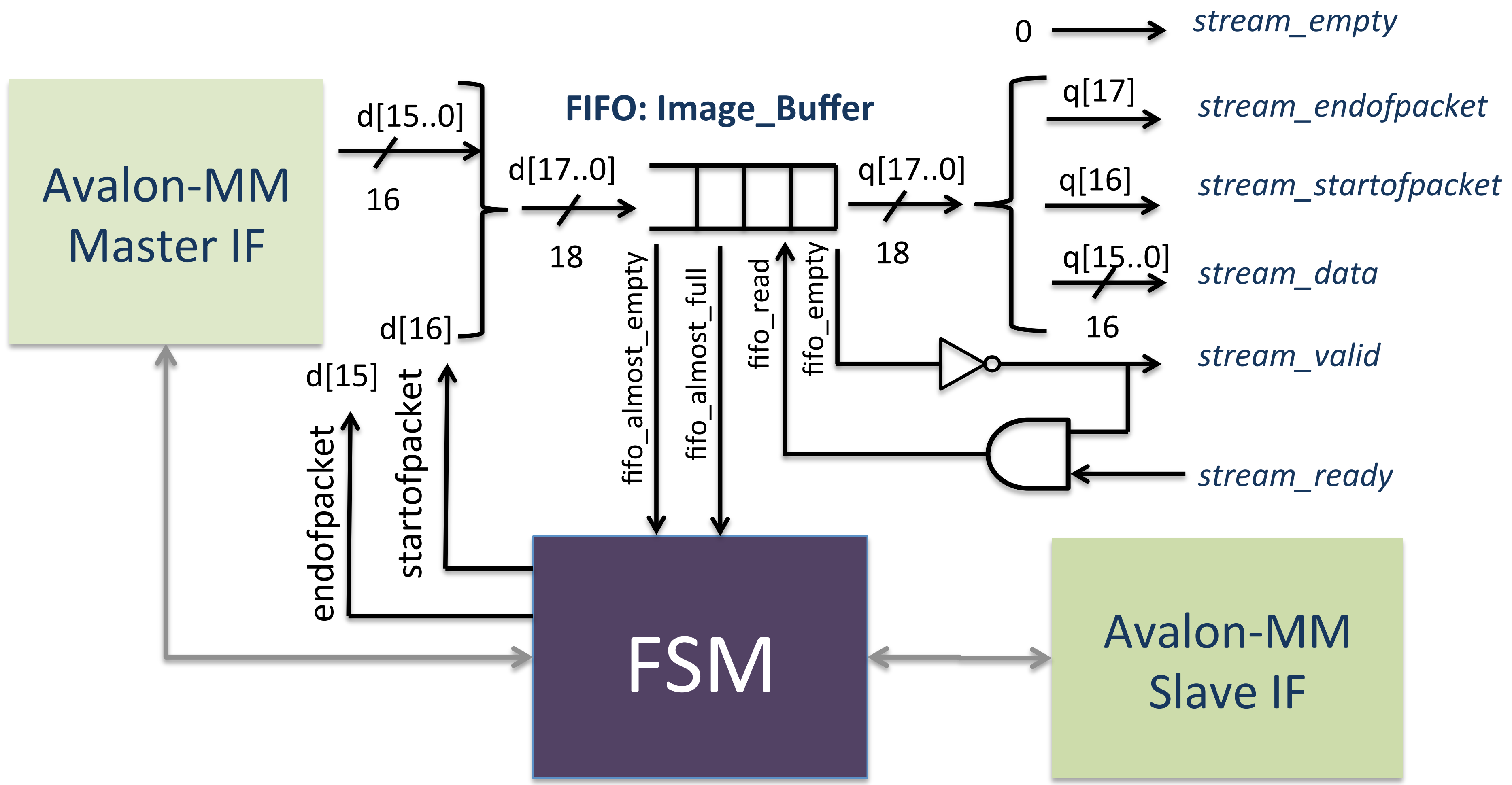
- The Buffer register holds the 32-bit address of the start of the memory buffer. This register is read-only, and shows the address of the first pixel of the frame currently being sent out via the Avalon-ST
- The BackBuffer register allows the start address of the frame to be changed under program control
  - 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 not used by the pixel buffer. 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 frame currently being output. While the Pixel Buffer is not yet finished outputting the current frame, bit S of the Status register will be set to 1. After the current screen is finished, the swap is performed and bit S is set to 0.

Pixel Buffer DMA Controller (1)

```
module VGA_Pixel_Buffer (
        // Inputs
        clk,
        reset,

        slave_address,
        slave_byteenable,
        slave_read,
        slave_write,
        slave_writedata,

        master_readdata,
        master_readdatavalid,
        master_waitrequest,

        stream_ready,
        //....

        // Bi-Directional

        // Outputs
        slave_readdata,

        master_address,
        master_arbiterlock,
        master_read,

        stream_data,
        stream_startofpacket,
        stream_endofpacket,
        stream_empty,
        stream_valid

        );
```



Pixel Buffer DMA Controller (2)



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

References

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