SISTEMI EMBEDDED

System Interconnect Fabric

Federico Baronti

Last version: 20180418
System Interconnect Fabric

• **Interconnect and logic resources** to manage whole connectivity among all components in an Altera SoPC system

• Is automatically generated by **Qsys**
  – Components must comply with the standardized Avalon® interfaces, which are specialized for:
    • Reading and writing registers and memory
    • Streaming high-speed data
    • Controlling off-chip devices
Example of a **SoPC system**

Custom logic can interact with a **Computer system** by:

- an **Avalon** interface, so that it can be part of the SoPC
- a **generic** interface, which can be connected to a PIO peripheral inside the SoPC
Avalon interfaces (1)

• **Avalon Memory Mapped Interface (Avalon-MM)**
  – An address-based read/write interface typical of master–slave connections

• **Avalon Streaming Interface (Avalon-ST)**
  – Supports unidirectional flow of data, including multiplexed streams, packets, and DSP data

• **Avalon Interrupt Interface (Sender/Receiver)**
  – An interface that allows components to signal events to other components
Avalon interfaces (2)

- **Avalon Clock Interface**
  - An interface that drives or receives clocks (all Avalon interfaces are synchronous)

- **Avalon Reset Interface**
  - An interface that provides reset connectivity

- **Avalon Conduit Interface**
  - An interface type that accommodates individual signals or groups of signals that do not fit into any of the other Avalon interfaces

- **Avalon Tri-State Conduit Interface (Avalon-TC)**
  - An interface to support connections to off-chip peripherals. Multiple peripherals can share pins through signal multiplexing, reducing the pin count of the FPGA (and the number of traces on the PCB)
Example of component interconnections within a Nios II system
Tri-state Conduit Pin Sharer
Avalon interfaces (3)

• Each of the Avalon interfaces defines a number of **signal roles** and their behavior.
  – Many signal roles are optional, allowing component designers the flexibility to select only the signal roles necessary to implement the required functionality.
  – With the exception of Avalon Conduit interfaces, each interface may include only one signal for each signal role.
  – Active-low signal alternatives are permitted for many signal roles.

• Avalon interfaces use **properties** to describe their behavior. E.g. the *clockRate* property of the Avalon Clock interface provides the frequency of a clock signal.
Avalon Memory Mapped (MM) (1)

• Interconnect fabric based on Avalon MM interfaces supports
  – Any number of master and slave components
    • The master-to-slave relationship can be one-to-one, one-to-many, many-to-one, or many-to-many
  – Connection to both on- and off-chip devices (microprocessors, memories, UARTs, DMAs, timers,...)
  – Master and slaves of different data widths
  – Components operating in different clock domains
  – Components using multiple Avalon-MM ports
Avalon Memory Mapped (MM) (2)

Example of a Avalon MM-based interconnect fabric system
Example of an Avalon MM slave component (Write operation on a PIO peripheral)
Avalon MM Signals (1)

• Each signal has a role
• An Avalon MM may contain only a restricted number of available signals
• Common signal roles are:
  – address, writedata
  – chipselect (chipselect_n), read (read_n), write(write_n),...
  – readdata
  – waitrequest (waitrequest_n)
Avalon MM Signals (2)

- **Address** (Master -> Slave)
  - **For masters**, the address signal represents a byte address. The value of the address must be aligned to the data width. **To write to specific bytes within a data word, the master must use the byteenable signals.**
  - **For slaves**, the interconnect translates the byte address into a word address in the slave’s address space so that each slave access is for a word of data from the perspective of the slave. For example, address=0 selects the first word of the slave and address=1 selects the second word of the slave.
Avalon MM Signals (3)

- byteenable/byteenable\_n (Master -> Slave)
- Example for a 32 bits slave:
  - byteenable operation
  - 1111 writes full 32 bits
  - 0011 writes lower 2 bytes
  - 1100 writes upper 2 bytes
  - 0001 writes byte 0 only
  - 0010 writes byte 1 only
  - 0100 writes byte 2 only
  - 1000 writes byte 3 only
Avalon MM Properties

• Specified when a component is inserted in a Qsys Library

• Describes interface behavior with regard to:
  – Latency
  – Pipeline
  – Burst
  – Read and Write setup and hold times
Functions of Avalon MM fabric

• Address Decoding
• Datapath Multiplexing
• Wait State Insertion
• Pipelined Read Transfers
• Arbitration for Multimaster Systems
• Burst Adapters
Address decoding (1)

• Address decoding logic forwards appropriate addresses to each slave
• Address decoding logic simplifies component design in the following ways:
  – The system interconnect fabric selects a slave whenever it is being addressed by a master. Slave components do not need to decode the address to determine when they are selected
  – Slave addresses are properly aligned to the slave interface
  – Changing the system memory map does not require to edit HDL manually
Address decoding (2)

• Example of address decoding in case of 1 master and 2 slaves

\[ \text{write1, write2 are asserted only when} \]
\[ \text{write is high and the address matches} \]
\[ \text{the corresponding slave base address} \]

• The address decoding logic is controlled by the base address setting in Qsys

• The \((S+1)\) LSBs of Slave 1 base address and the \((T+1)\) LSBs of Slave 2 base address must be 0
Address decoding (3)

- Example of address decoding in case of 1 master and 2 slaves

`address[31..0]` 

`write` 

`write1, write2` are asserted only when `write` is high and the address matches the corresponding slave base address

`write1`
Data path multiplexing

- Drives the `writedata` signal from the granted master to the selected slave (actually to all the slaves), and the `readdata` signal from the selected slave back to the requesting master.
- Example of the data path multiplexing logic for 1 master and 2 slaves.

In Qsys the generation of data path multiplexing logic is specified using the Connections column on the System Contents tab.
Wait state insertion (1)

- Avalon-MM Master expects that each transfer is concluded in **one clock cycle**
- How to accommodate slaves that require more than one clock cycle for read ($\text{readWaitTime} > 0$) or write ($\text{writeWaitTime} > 0$) operations?
- Master receives a **waitrequest** signal. If asserted the Master must extend the current transfer.
  - Generated by the **wait insertion logic**, according to the selected slave properties and the **waitrequest** signal if generated by the selected slave
Wait state insertion (2)

- Wait states extend the duration of a transfer by one or more clock cycles.
- Wait state insertion logic accommodates the timing needs of each slave or the wait due to arbitration in a multi-master system.
- System interconnect fabric also inserts wait states in cases when slave read and write signals have specific setup or hold time requirements.
Pipelined read transfer

- The Avalon-MM interface supports pipelined read transfers, allowing a pipelined master to start multiple read transfers in succession without waiting for the previous transfers to complete.
- Pipelined transfers allow master-slave pairs to achieve higher throughput, even though the slave requires one or more cycles of latency to return data for each transfer.
- Qsys generates logic to handle pipeline latency based on the properties of the master and slaves in the system. When configuring a system in Qsys, there are no settings that directly control the pipeline management logic in the system interconnect fabric.
Read/Write transfers

Fixed wait states (readWaitTime=1; writeWaitTime=2). Signals at the slave interface

---

Pipelined w/ waitrequest and fixed wait states (readWaitTime=2). Signals at the slave interface

Generates by the slave
Interrupts (1)

• In systems where components have interrupt request (IRQ) sender interfaces, the system interconnect fabric includes interrupt controller logic

• A separate interrupt (controller) router is generated for each interrupt receiver

• The interrupt (controller) router aggregates IRQ signals from all interrupt senders, and maps them to user-specified values on the receiver inputs
Interrupts (2)

• Individual Requests IRQ Scheme
Reset distribution

• **Qsys** generates the logic to route the reset pulse to all components

• The system interconnect fabric distributes the reset signal conditioned for each clock domain
  – The duration of the reset signal is at least one clock period

• The system interconnect fabric asserts the system-wide reset in the following conditions:
  – The global reset input to the **Qsys**/system is asserted
  – Any component asserts its resetrequest signal (e.g. Watchdog)

• The global reset and reset requests are ORed together. This signal is then synchronized to each clock domain associated to an Avalon-MM port, **which causes the asynchronous resets to be de-asserted synchronously**
Reset synchronizer

Guideline: EVERY ASIC USING AN ASYNCHRONOUS RESET SHOULD INCLUDE A RESET SYNCHRONIZER CIRCUIT!!

Without a reset synchronizer, the usefulness of the asynchronous reset in the final system is void even if the reset works during simulation.

The reset synchronizer logic of Figure 9 is designed to take advantage of the best of both asynchronous and synchronous reset styles.

Figure 9 - Reset Synchronizer block diagram

An external reset signal asynchronously resets a pair of master reset flip-flops, which in turn drive the master reset signal asynchronously through the reset buffer tree to the rest of the flip-flops in the design. The entire design will be asynchronously reset.

Reset removal is accomplished by de-asserting the reset signal, which then permits the d-input of the first master reset flip-flop (which is tied high) to be clocked through a reset synchronizer. It typically takes two rising clock edges after reset removal to synchronize removal of the master reset.

Two flip-flops are required to synchronize the reset signal to the clock pulse where the second flip-flop is used to remove any metastability that might be caused by the reset signal being removed asynchronously and too close to the rising clock edge. As discussed in section 5.4, these synchronization flip-flops must be kept off of the scan chain.

Component development flow

• **Specification and definition**
  – Define the functionality of the component
  – Determine component interfaces, such as Avalon-MM, Avalon-ST, interrupt, or other interfaces
  – Determine the component clocking requirements; what interfaces are synchronous to what clock inputs
  – If you want a microprocessor to control the component, determine the interface to software, such as the register map

• **Implement the component in Verilog or VHDL**

• **Import the component into Qsys**
  – Use the component editor to create a `<component_name>_hw.tcl` script file that describes the component
  – Instantiate the component into a Qsys system
References

• Altera, “Avalon Interface Specifications,”
  mnl_avalon_spec.pdf