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**
  - 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
Avalon interfaces (3)

- 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.
- 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 of each signal role.
  - Active-low signal alternatives are permitted for many signal roles.
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
Avalon Memory Mapped (MM) (3)

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 slave

• The address decoding logic is controlled by the **Base address** setting in **Qsys**
Data path multiplexing

- Drives the *writedata* signal from the granted master to the selected slave, 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 panel on the System Contents menu.
Wait state insertion

- 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 prior 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

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

- Individual Requests IRQ Scheme

Within the System Interconnect Fabric

Within the Nios II processor
Reset distribution

- **Qsys** generates the logic to drives 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 reset request signal (eg. 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

When reset is de-asserted asynchronously ...

... masterrst_n is removed synchronously

pad_clk  clock
pad_rst_n  reset

Asynchronous reset assertion

reset distribution buffer tree

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 VHDL or Verilog HDL

• Import the component into Qsys
  – Use the component editor to create a _hw.tcl file that describes the component
  – Instantiate the component into a Qsys system
References

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