# Qsys and IP Core Integration

Stephen A. Edwards
(after David Lariviere)

Columbia University

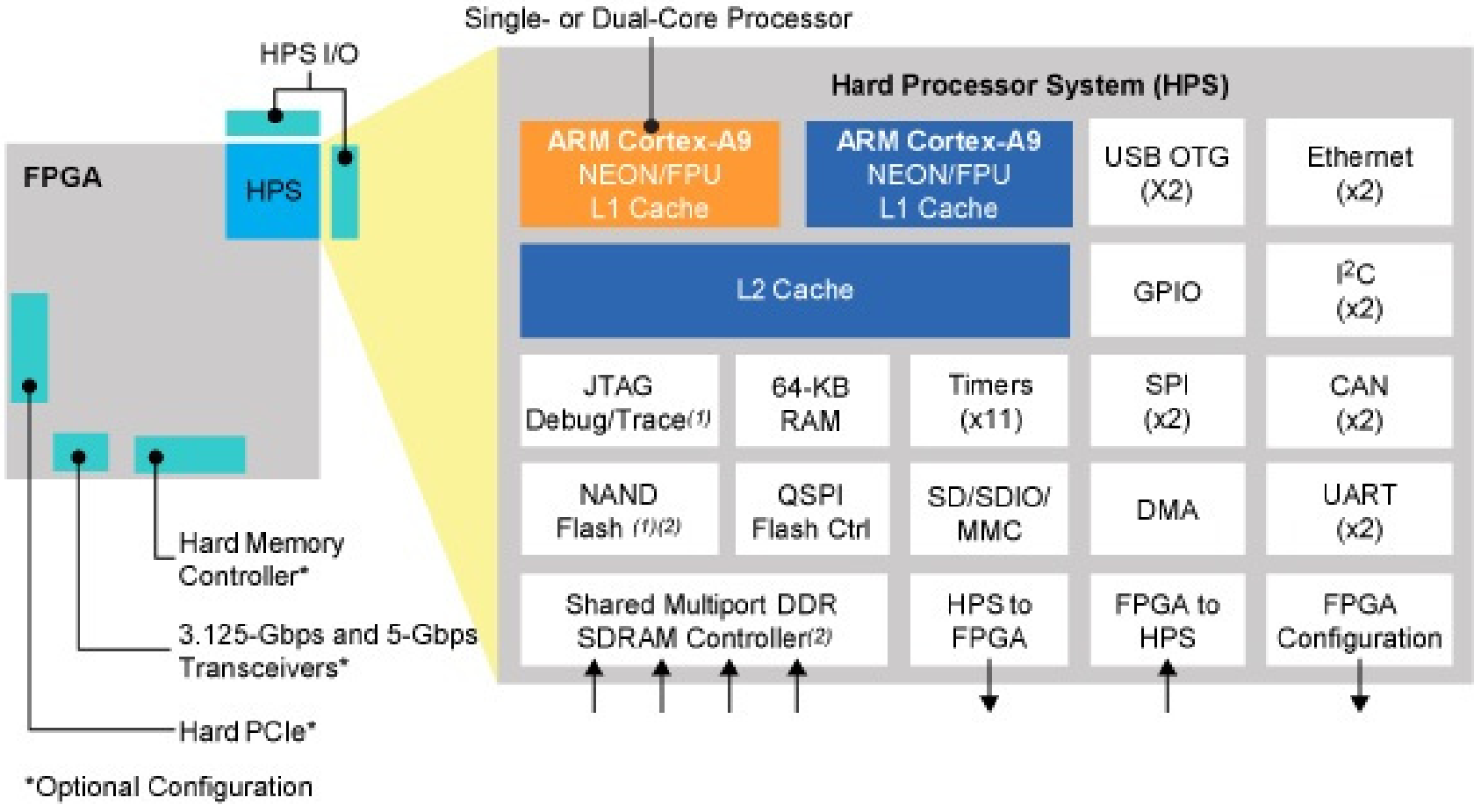
Spring 2020

# IP Cores

# Cyclone V SoC: A Mix of Hard and Soft IP Cores

IP = Intellectual Property  Core = block, design, circuit, etc.

Hard = wires & transistors  Soft = implemented w/ FPGA




Source: Altera

# Example IP Cores

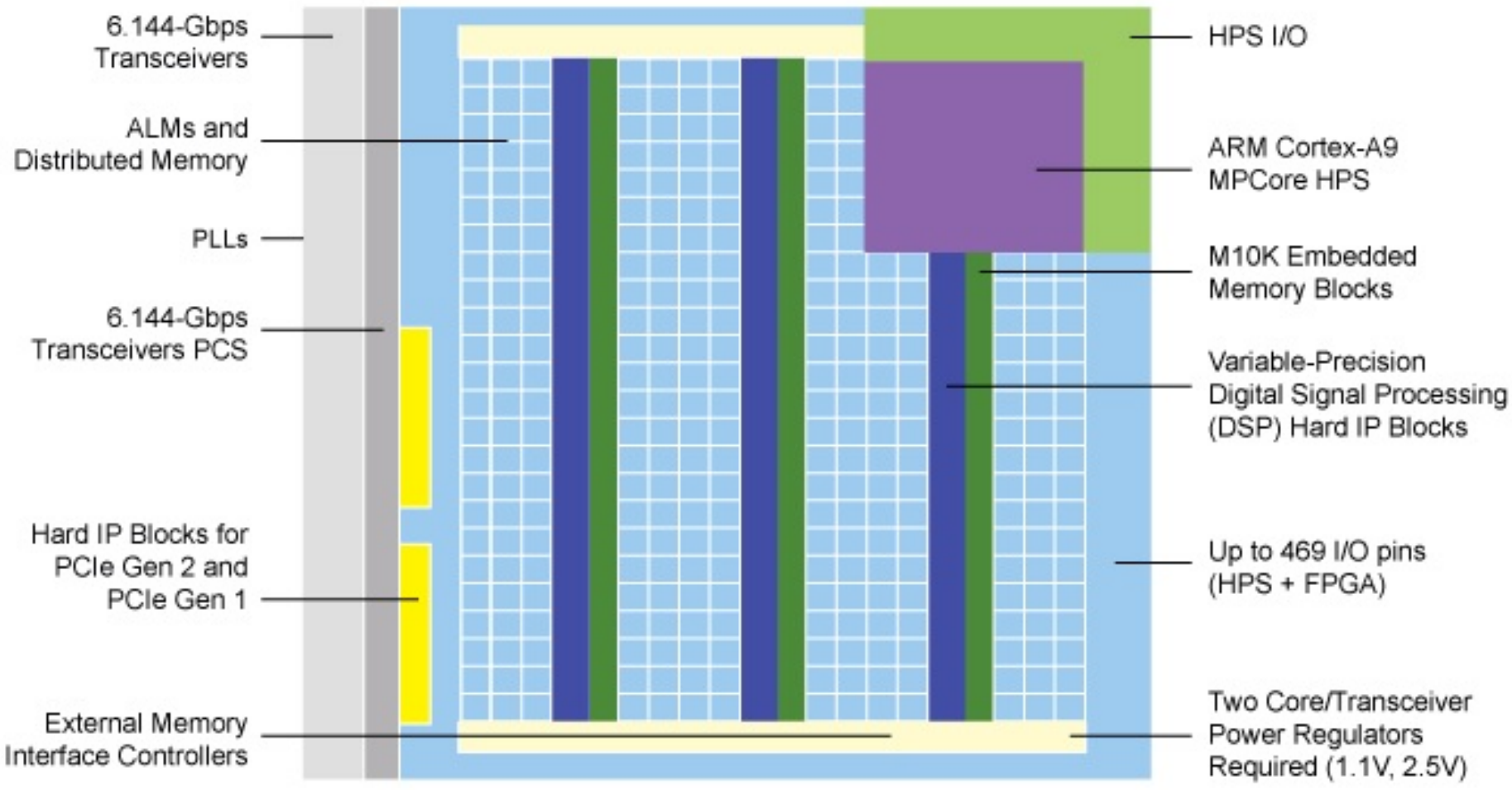
**CPUs**: ARM (hard), NIOS-II (soft)

**Highspeed I/O**: Hard IP Blocks for High Speed Transceivers (PCI Express, 10Gb Ethernet)

**Memory Controllers**: DDR3

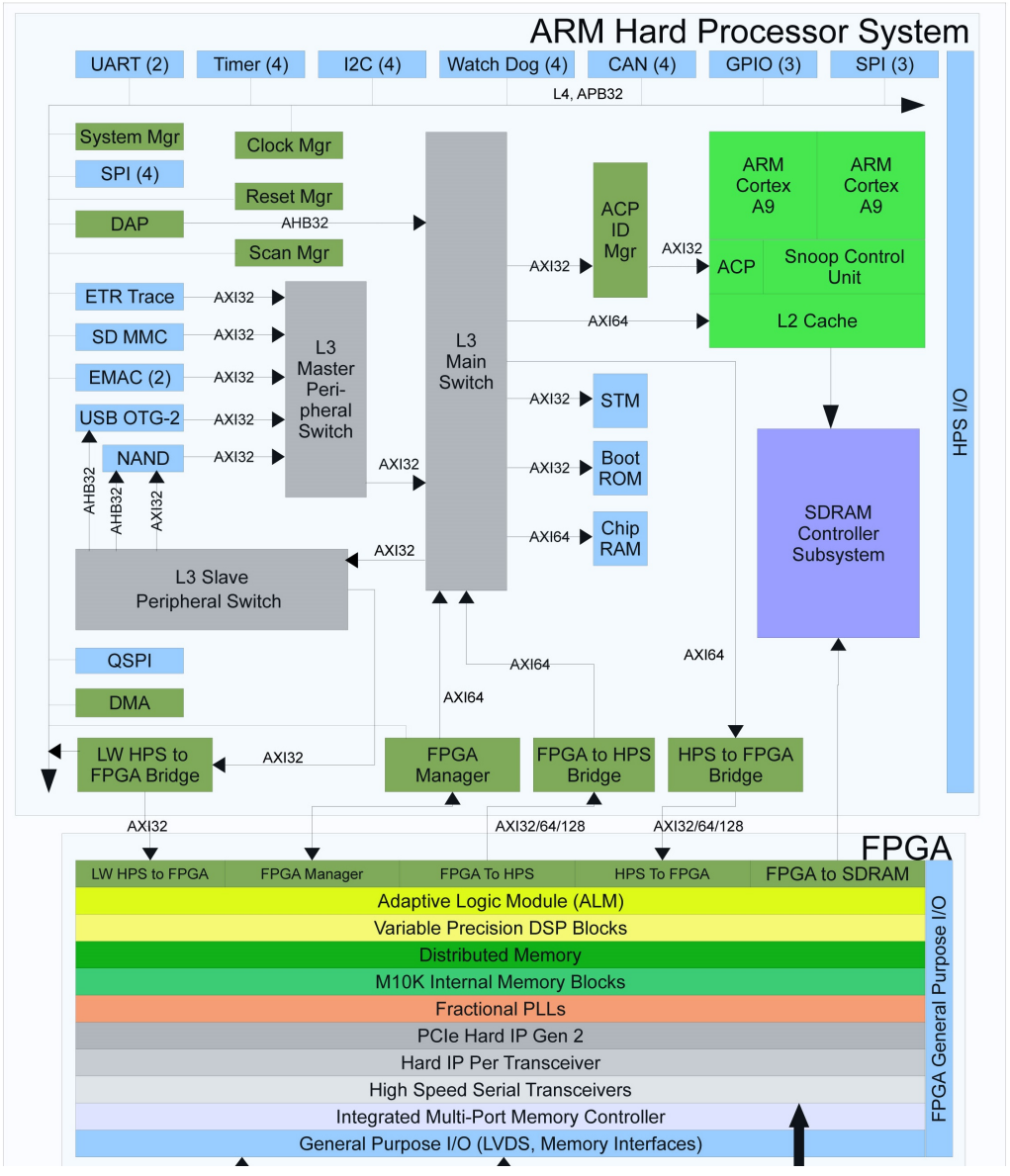
**Clock and Reset signal generation**: PLLs

# Cyclone V SoC: FPGA layout



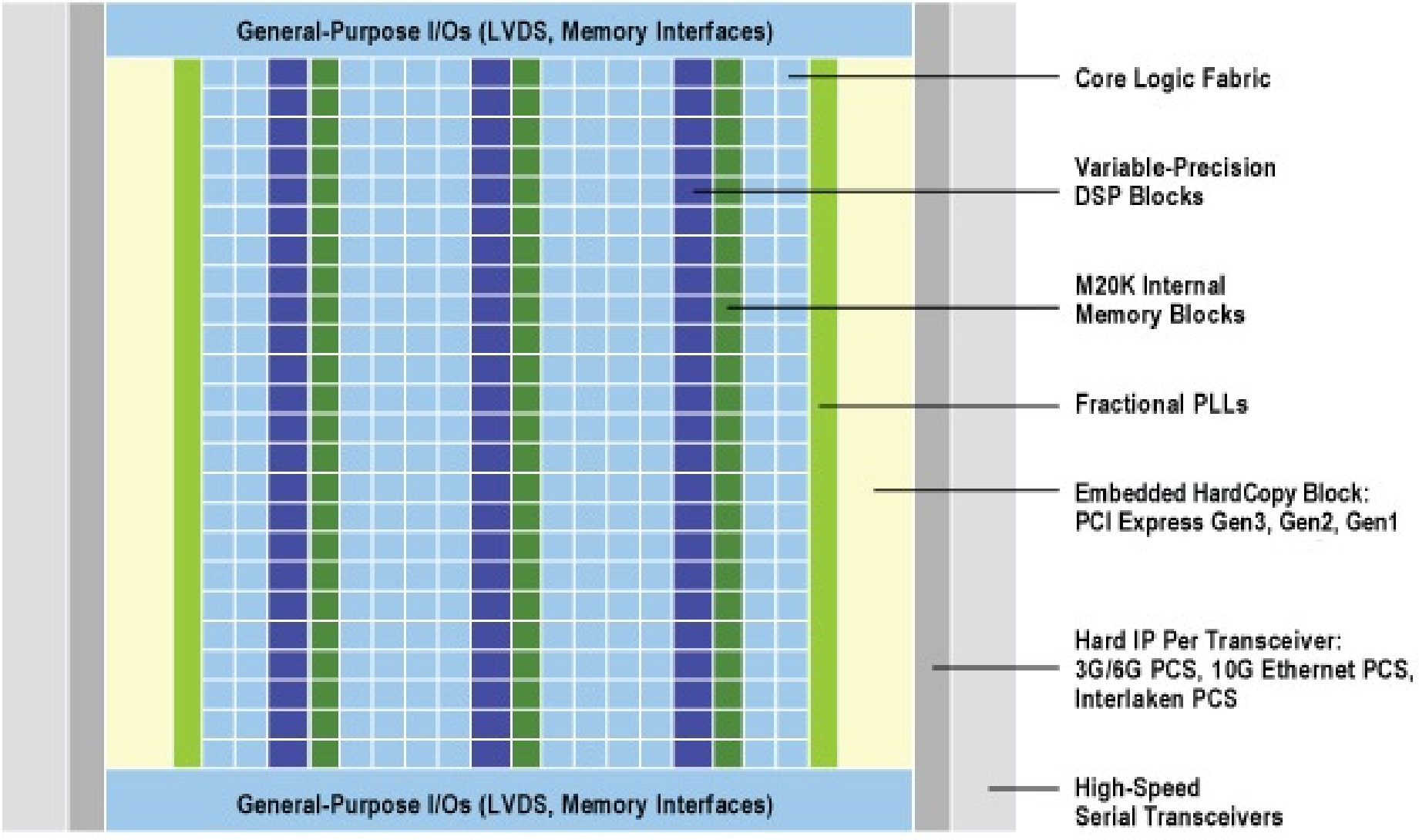

Source: Altera

# Cyclone V SoC: HPS Layout



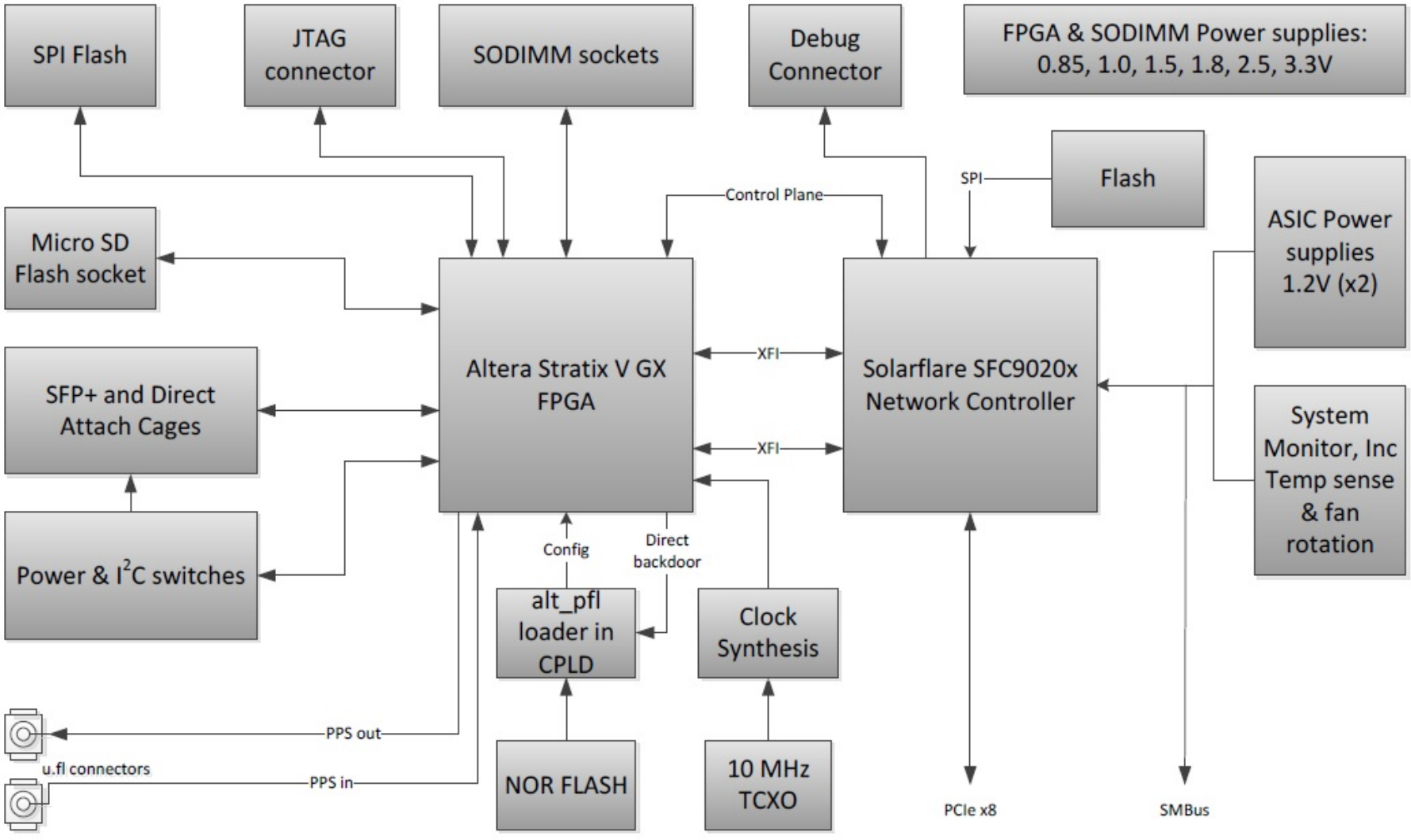

Source: NARD, LLC.

# Stratix V: FPGA Layout



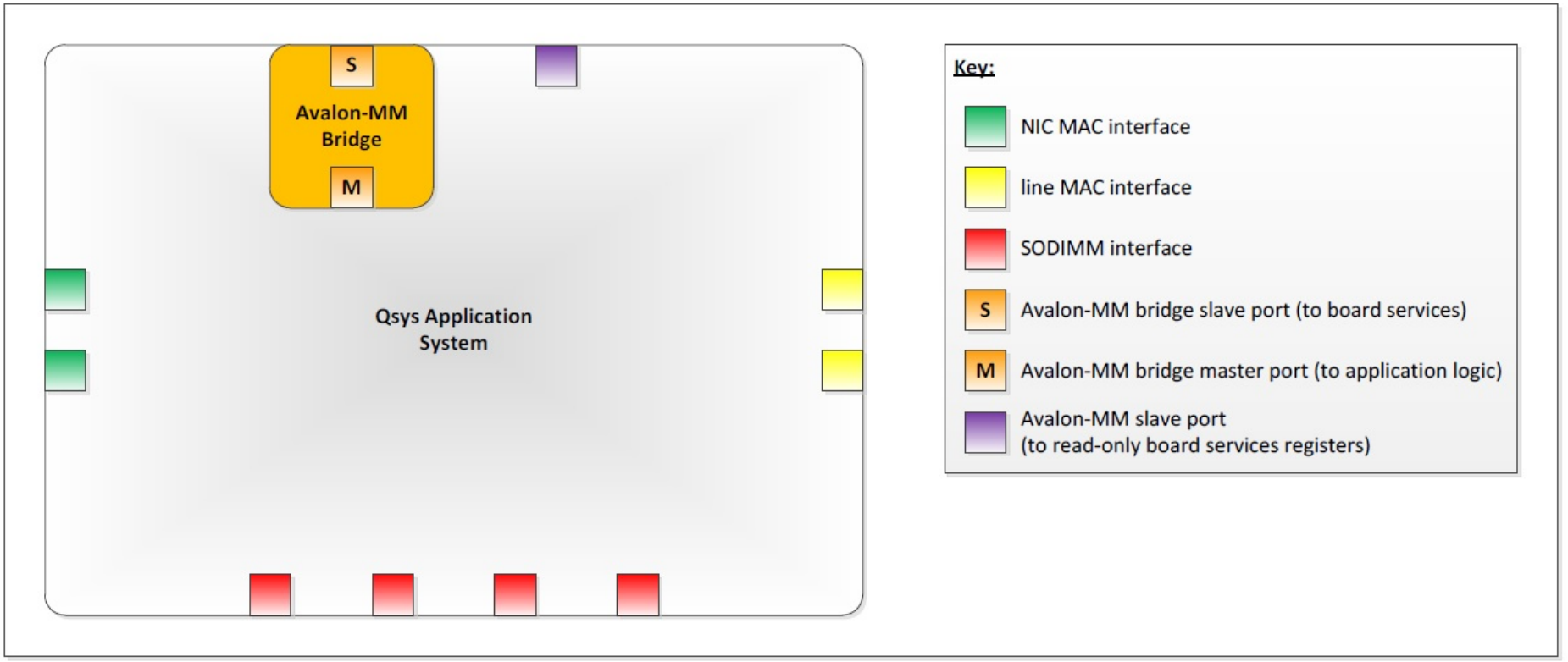

Source: Altera

# Stratix V: Solarflare AoE PCB Layout



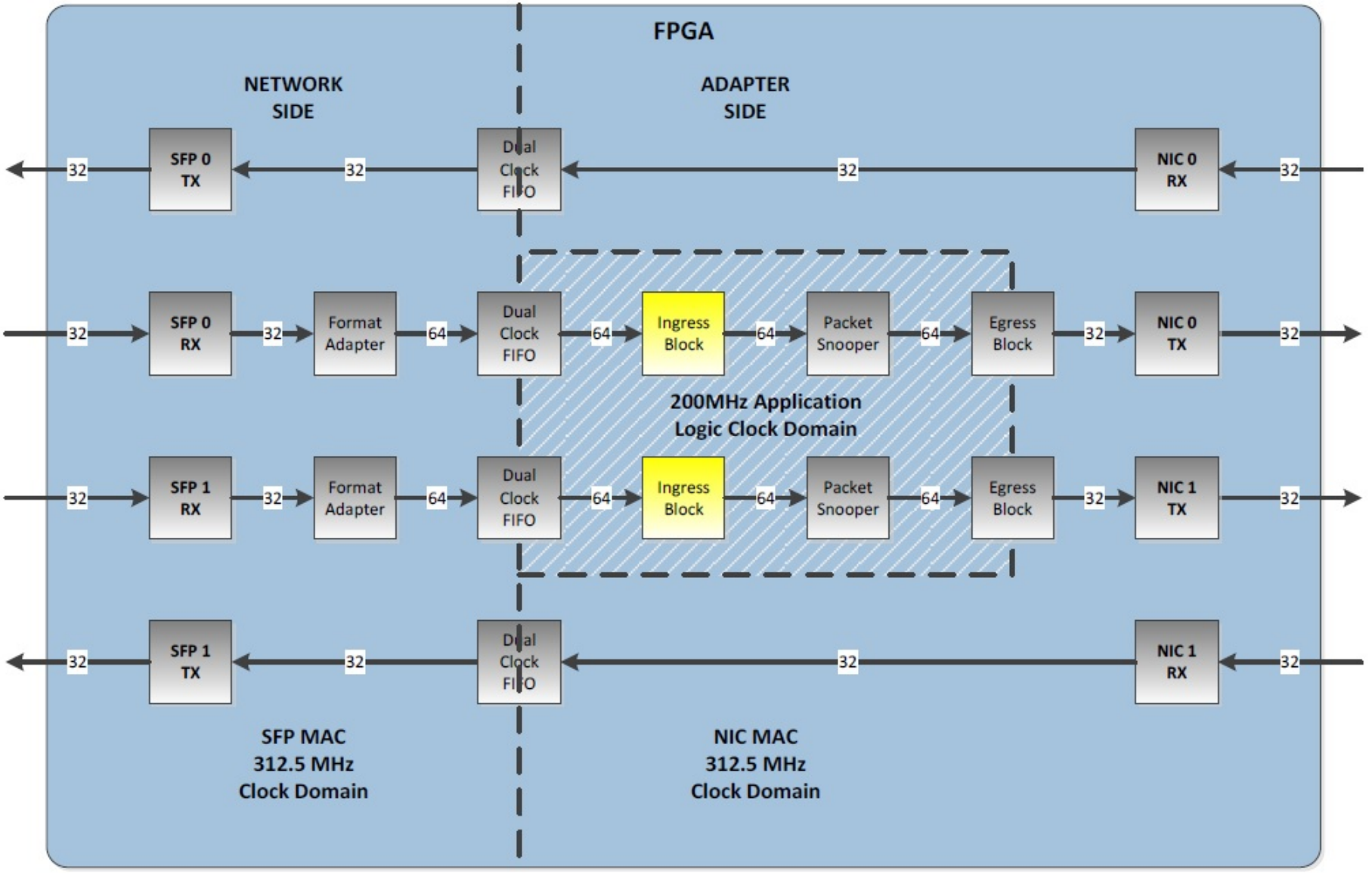

Source: Solarflare FDK

# Stratix V: Solarflare AoE Qsys Layout




Source: Solarflare FDK

# Stratix V: Solarflare AoE Qsys Example




Source: Solarflare FDK

# Bridges

A bridge connects two, often different, buses.

Enables multiple clock domains, different protocols (e.g., AXI $\leftrightarrow$ Avalon), bus widths, etc.

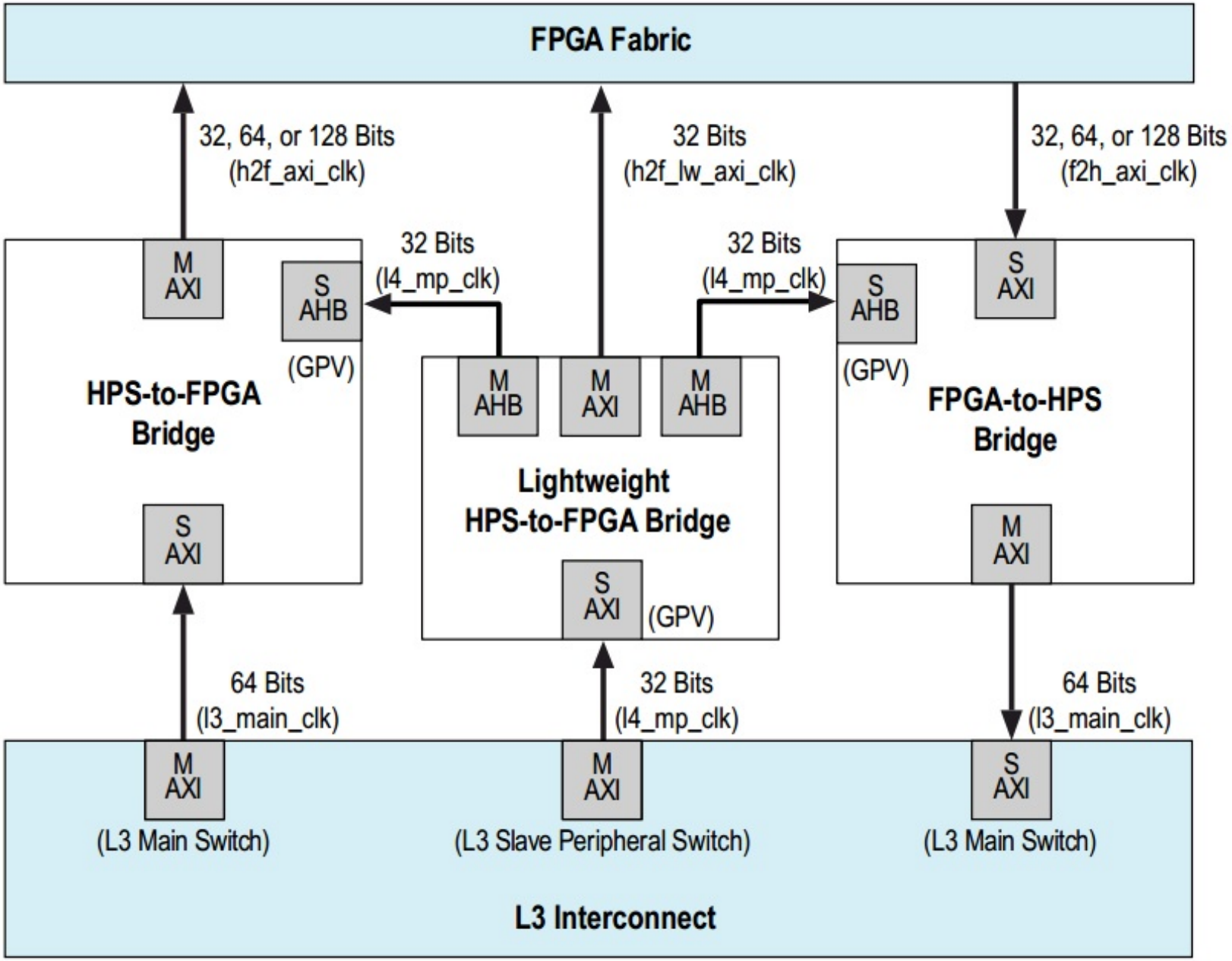
**Example Bridge Types**:

SOC HPS $\leftrightarrow$ FPGA Bridge

Avalon MM Clock Crossing Bridge

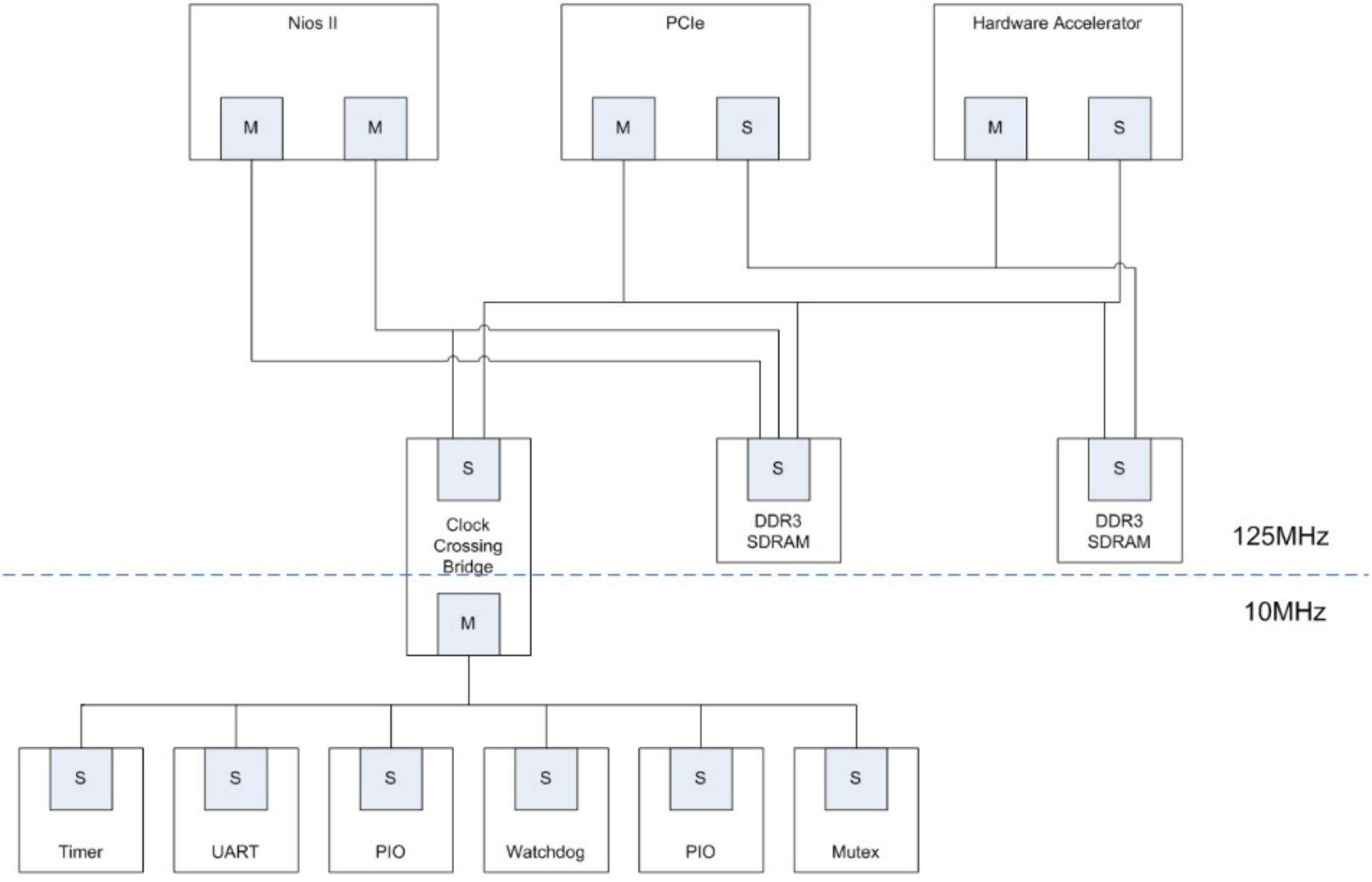
Avalon MM Pipeline Bridge

# Cyclone V SoC: FPGA ↔ HPS Bridge



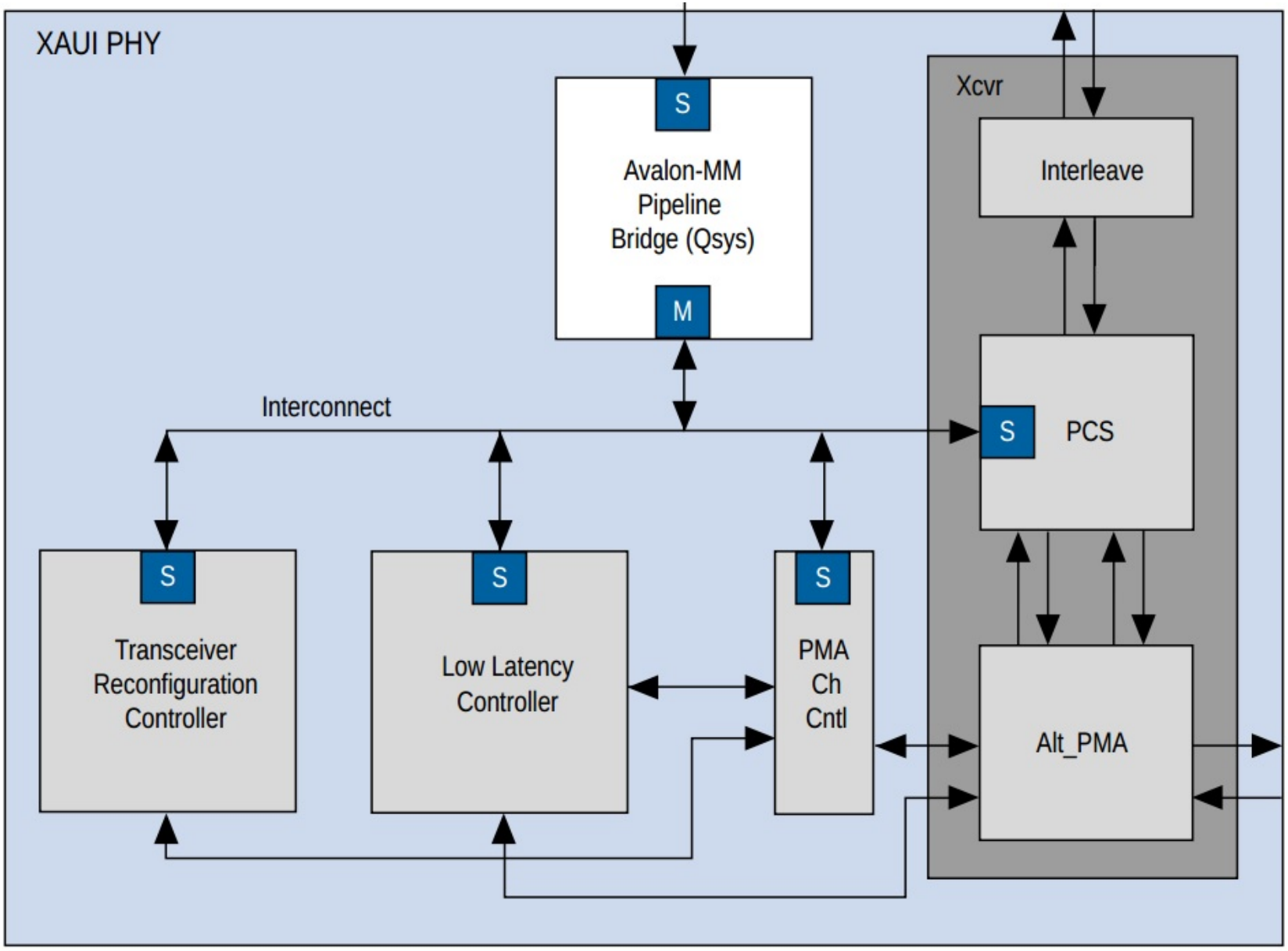

Source: Altera

# Clock Crossing Bridge Example




Source: Altera

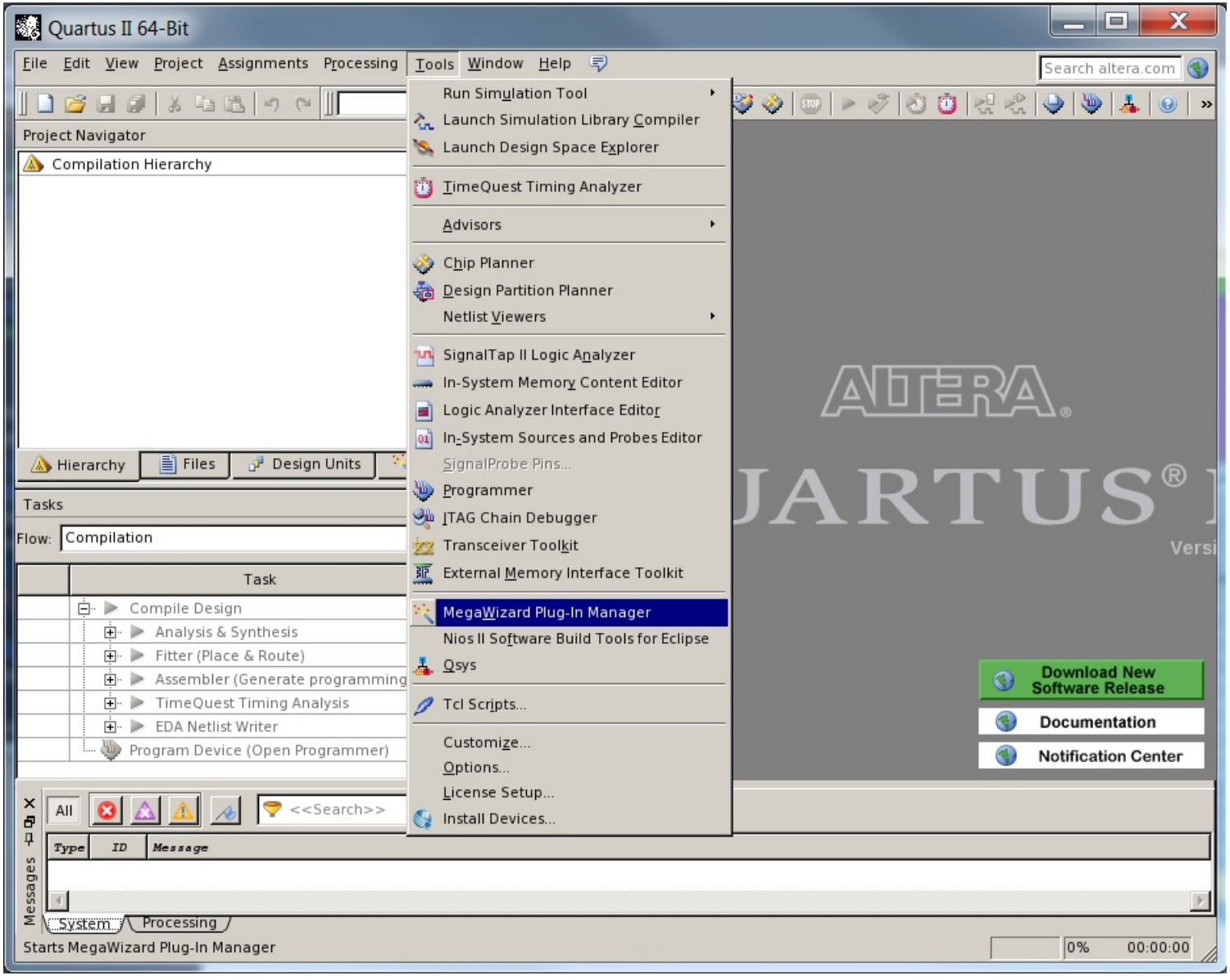
# Pipeline Bridge Example




Source: Altera

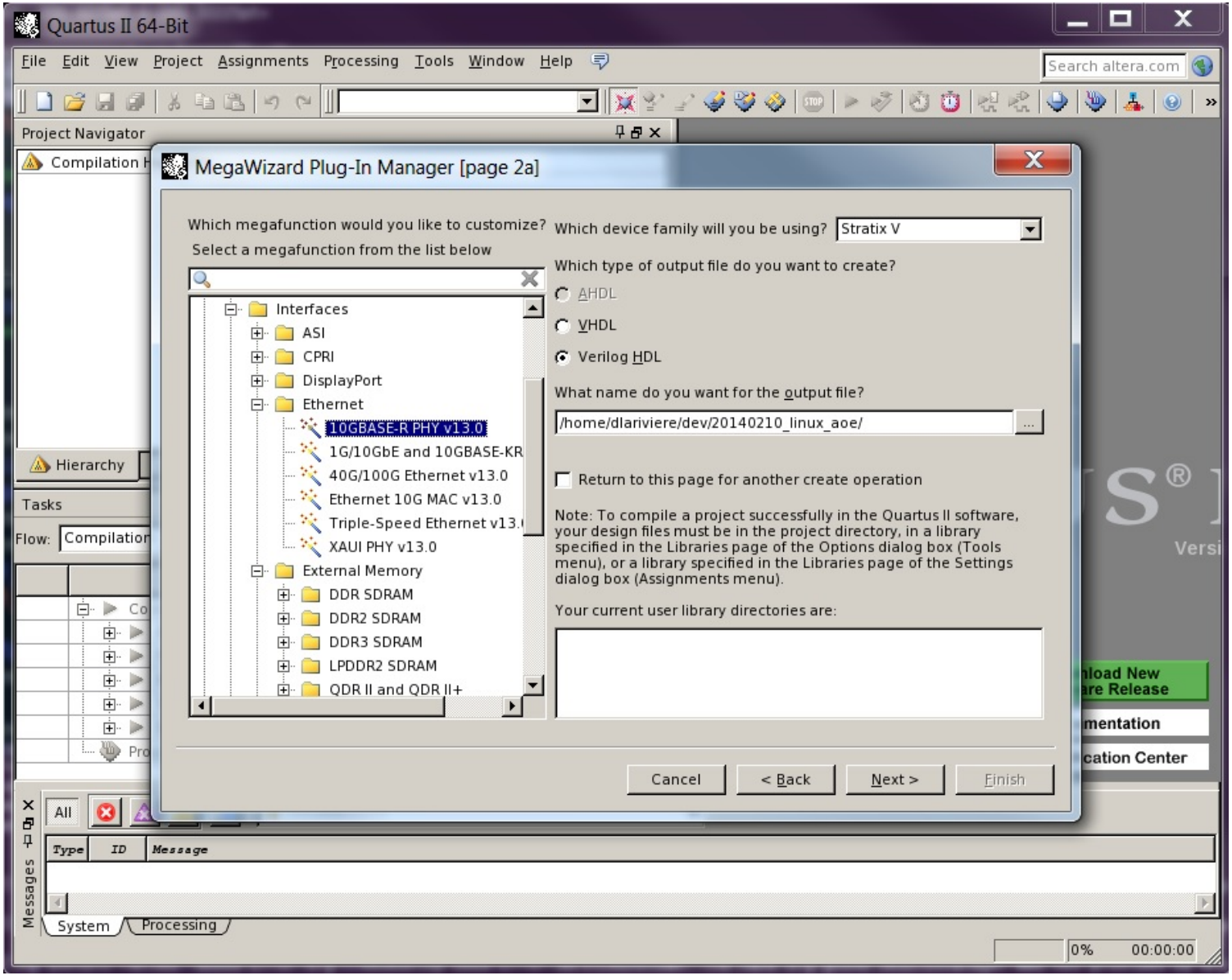
# Altera's IP Core Integration Tools

# The Quartus Megawizard




Source: Altera

# Megawizard: Example 10Gb Ethernet PHY




Source: Altera

# Megawizard IP Cores

**Arithmetic**: Addition, Subtraction, Multiplication, Division, Multiply-(add|accumulate), ECC

**Floating Point**:

**Gate Functions**: Shift Registers, Decoders, Multiplexers

**I/O Functions**: PLL, temp sensor, remote update, various high speed transceiver related

**Memory**: Single/Dual-port RAM or ROMs, Single/Dual-clock FIFOs, (RAM) Shift registers

**DSP**: FFT, ECC, FIR, etc (large suite specifically for graphics as well)

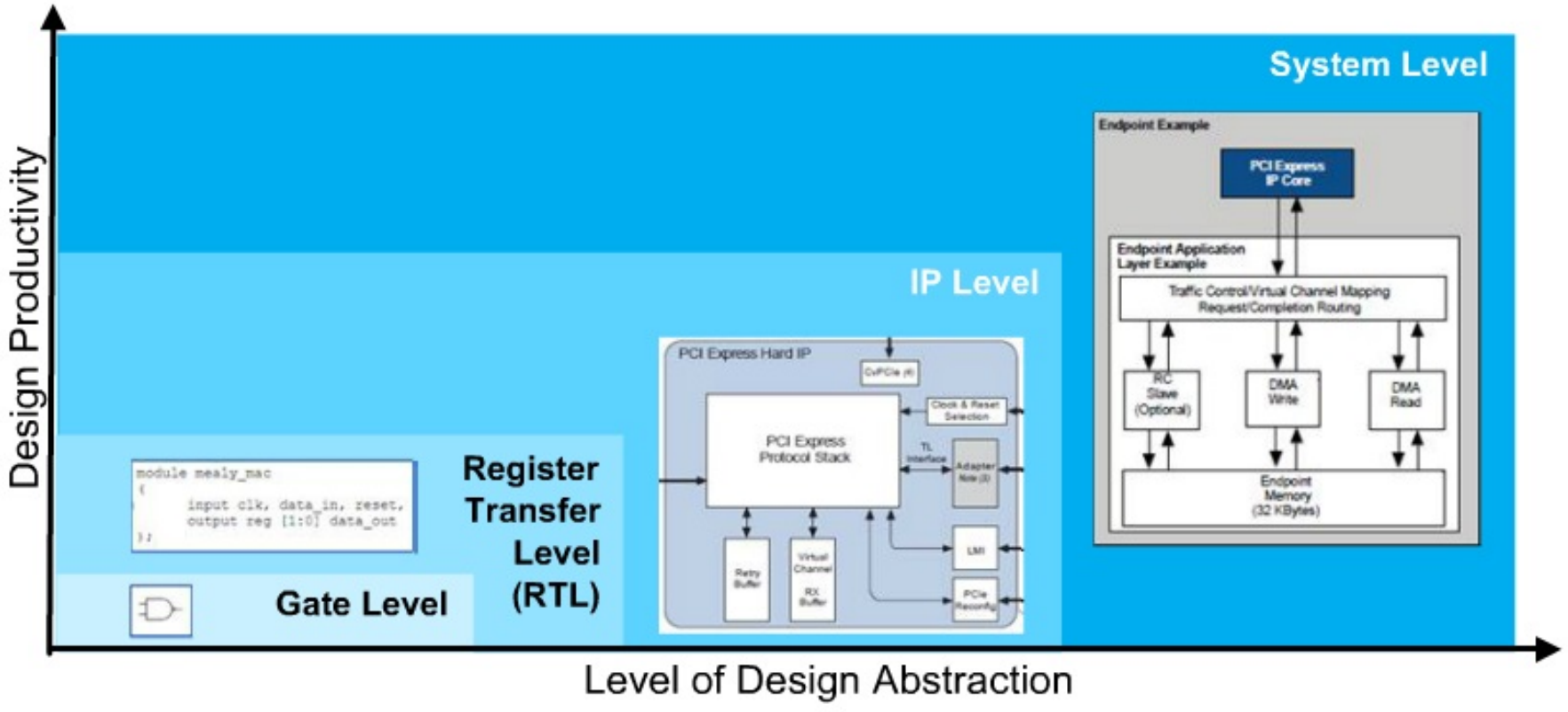
Note: some megafunctions are only available on certain FPGAs

# Qsys

Qsys is Altera's system integration tool for building Network-on-Chip (NoC) designs connecting multiple IP cores.

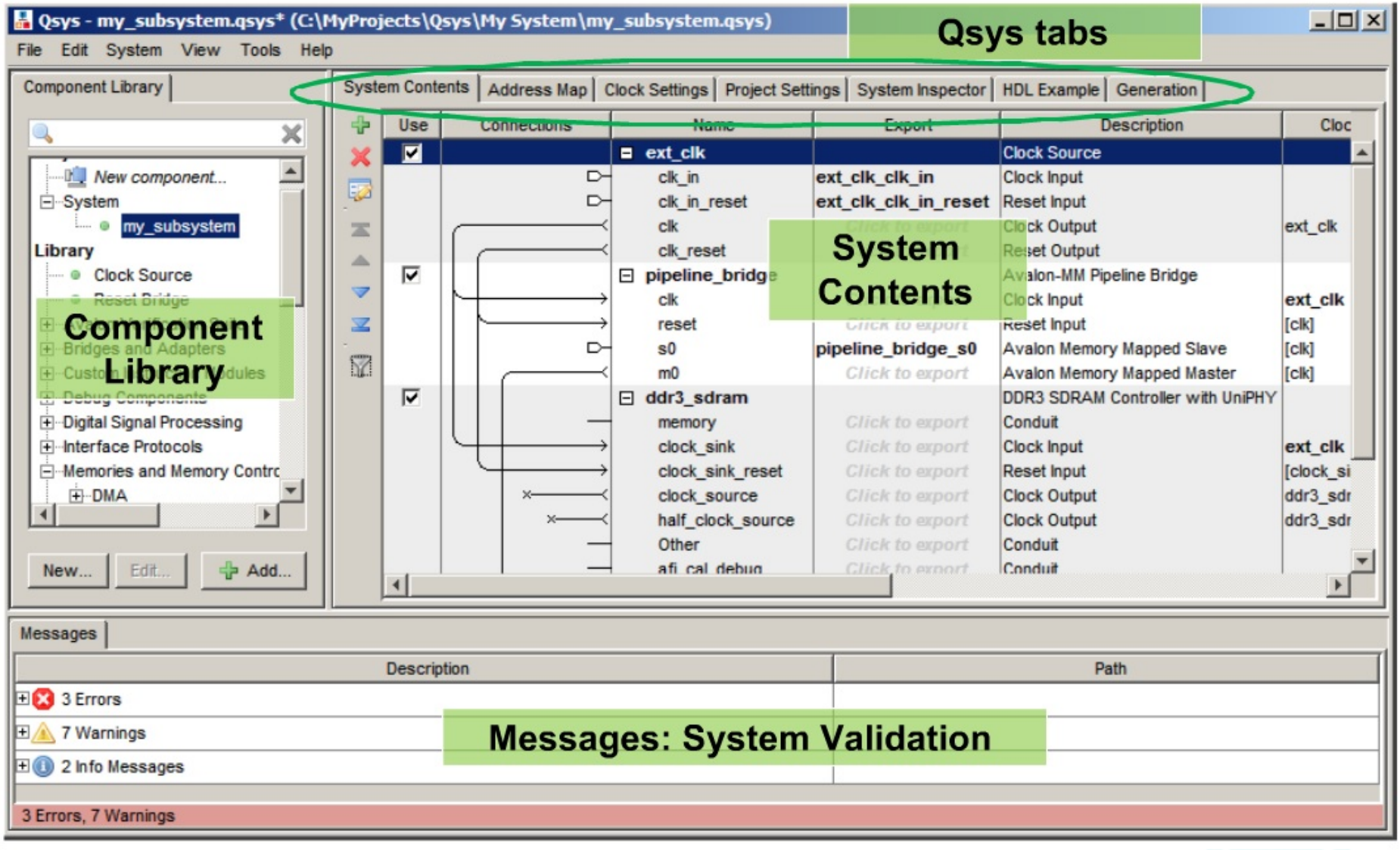
| You | Qsys |
|---|---|
| List the IP components and how you want them connected | Generates the interconnect (arbiters, etc.) adds adapters as necessary, warns of errors |

# Qsys: Raising Level of Abstraction




Source: Altera

# Qsys UI




Source: Altera

# System Level Design: Why Use Qsys

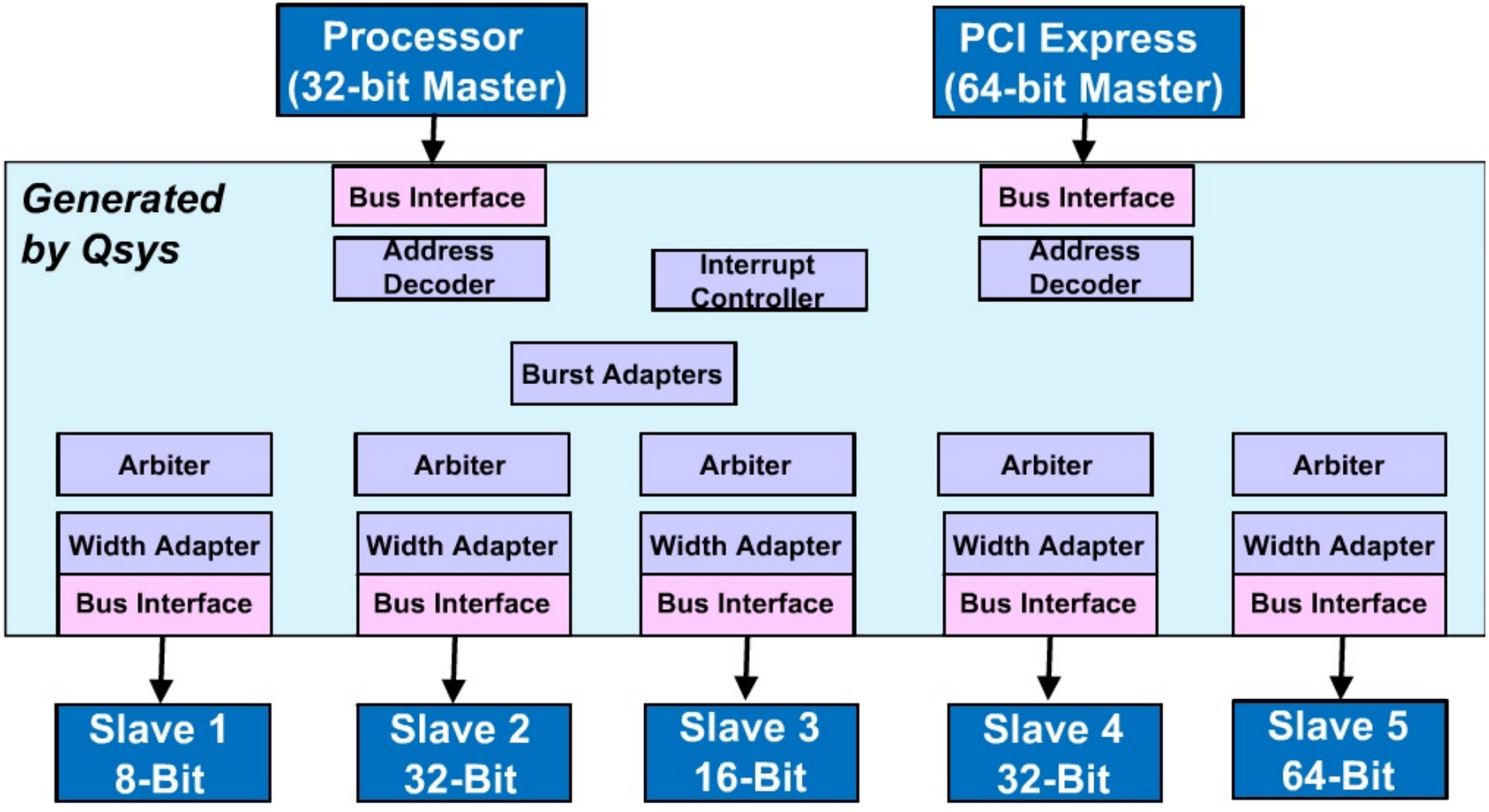
Avoids manually developing custom interconnect fabrics and signaling.

Instead of cycle-to-cycle coordination between every individual IP core, focus on transaction-level designs.

Design IP without knowing exactly when data will transfer and instead only focus on how (once it does).

(Only valid if you design your individual components to one of the standardized interfaces)

# Qsys-based Method of Design




Source: Altera

# Connecting IP Cores

# Interface Types

**Memory-mapped** Interfaces:

Avalon MM (Altera)

AXI (ARM, supported by Qsys now for SoC)

**Streaming** Interfaces: Avalon ST:

Avalon ST source port: outputs streaming data

Avalon ST sink port: receives incoming streaming data

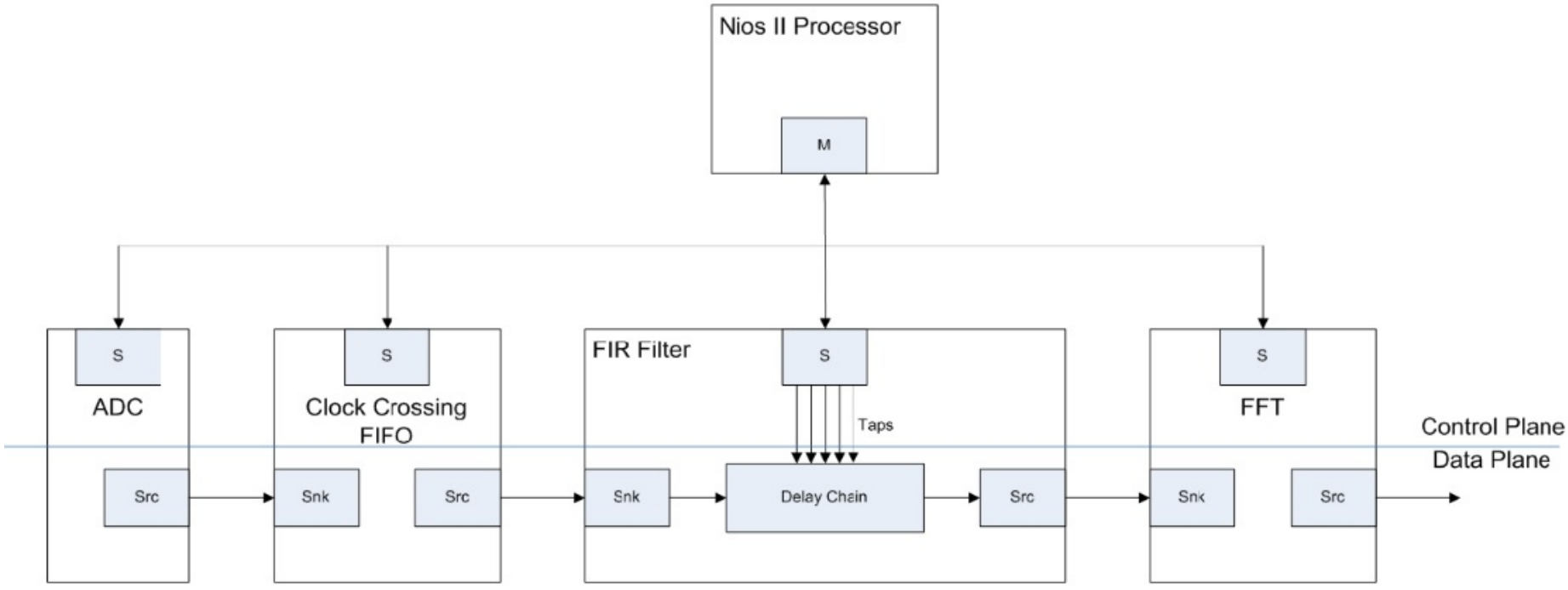
# Control vs. Data Planes

**Control Plane**: Memory mapped registers typically used for configuring devices, querying status, initiating transactions, etc (low bandwidth)

**Data Plane**: Streaming directed graphs for actually moving and processing large amounts of data (audio/video, network packets, etc); high bandwidth

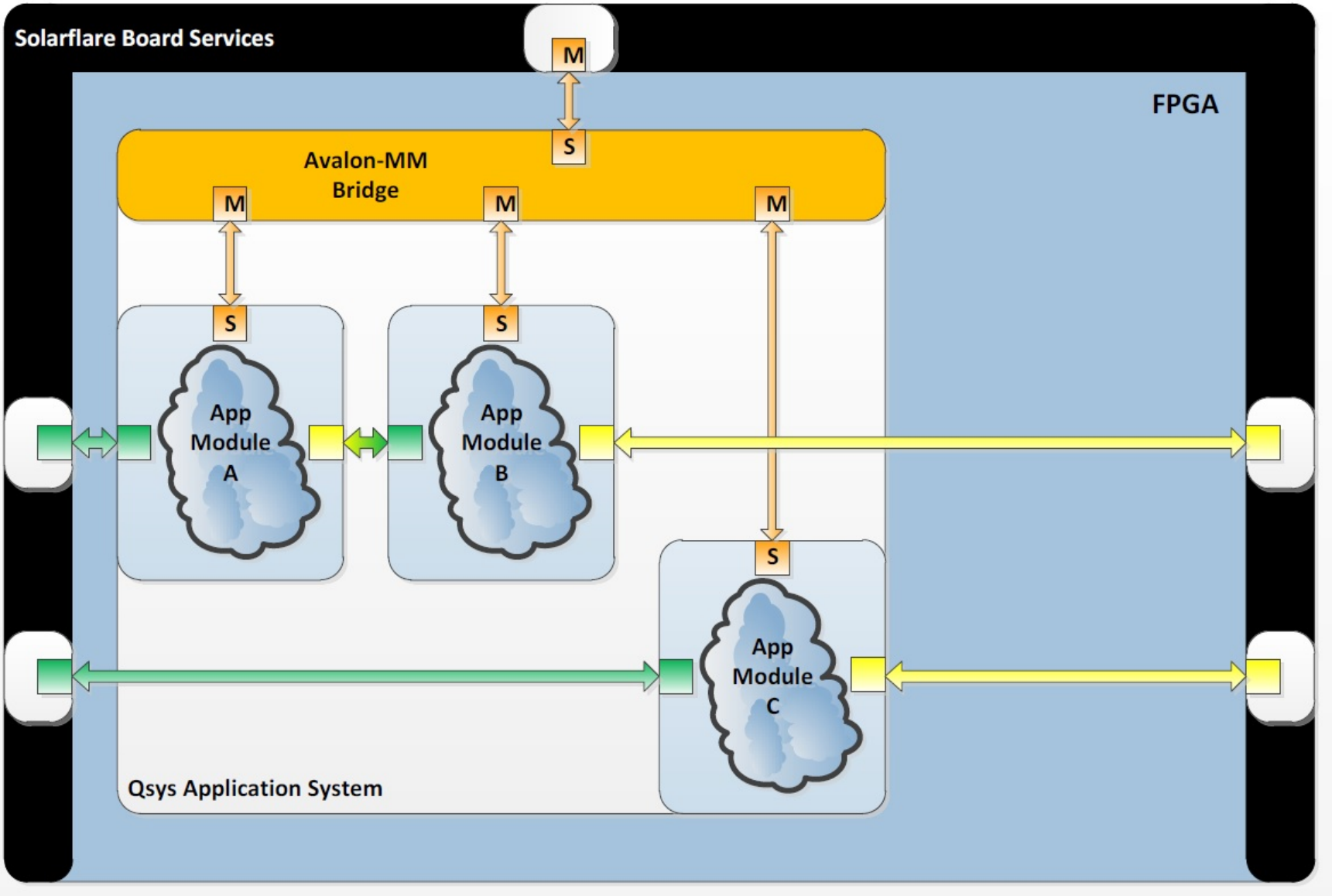
A single IP core can have both MM and ST interfaces (including multiple of each).

# Control and Data Planes Example




Source: Altera

# Control and Data Planes Example: Solarflare AoE




Source: Altera

# Qsys signal types

Clock

Reset

Interrupt

Avalon MM signals (Memory-Mapped)

Avalon ST signals (Streaming)

Tristate

Conduit (your own custom signals)

# Why explicitly label signal types?

...vs. simply making everything a wire/conduit

Allows Qsys to protect you from yourself!

Ensure matching between signal types (e.g., “clock out” $\rightarrow$ “clock in”)

Detect and automatically insert dual clock crossing domains (only if it knows which clock domains IPs are in)

Automatically convert data widths, formats, error flags (convert 32 bit master into four 8-bit slave reads, etc)

Automatically synchronize and OR-gate multiple resets

Automatically insert pipeline stages to improve fmax

# Avalon MM Master Signals

| Signal type | Width | Direction | Required | Description |
|---|---|---|---|---|
| address | 1-64 | Output | Y | Byte address corresponding to slave for transfer request *(discussed later)* |
| waitrequest<br>waitrequest_n | 1 | Input | Y | Forces master to stall transfer until deasserted; other Avalon-MM interface signals must be held constant |
| read<br>read_n | 1 | Output | N | Indicates master issuing read request |
| readdata | 8, 16, 32, 64, 128, 256, 512, 1024 | Input | N | Data returned from read request |
| write<br>write_n | 1 | Output | N | Indicates master issuing write request |
| writedata | 8, 16, 32, 64, 128, 256, 512, 1024 | Output | N | Data to be sent for write request |
| byteenable<br>byteenable_n | 2, 4, 8, 16, 32, 64, 128 | Output | N | Specifies valid byte lane(s) for readdata or writedata (width = data width / 8) |
| lock<br>lock_n | 1 | Output | N | Once master is granted access to shared slave, locks arbiter to master until deasserted |

Source: Altera

# Avalon MM Slave Signals

| Signal type | Width | Direction | Required | Description |
|---|---|---|---|---|
| address | 1-64 | Input | N | Word address of slave for transfer request *(discussed later)* |
| waitrequest<br>waitrequest_n | 1 | Output | N | Allows slave to stall transfer until deasserted (other Avalon-MM interface signals must be held constant) |
| read<br>read_n | 1 | Input | N | Indicates slave should respond to read request |
| readdata | 8, 16, 32, 64, 128, 256, 512, 1024 | Output | N | Data provided to Qsys interconnect in response to read request |
| write<br>write_n | 1 | Input | N | Indicates slave should respond to write request |
| writedata | 8, 16, 32, 64, 128, 256, 512, 1024 | Input | N | Data from the Qsys interconnect for a write request |
| byteenable<br>byteenable_n | 2, 4, 8, 16, 32, 64, 128 | Input | N | Specifies valid byte lane for readdata or writedata (width = data width / 8) |
| begintransfer<br>begintransfer_n | 1 | Input | N | Asserts at the beginning (first cycle) of any transfer |

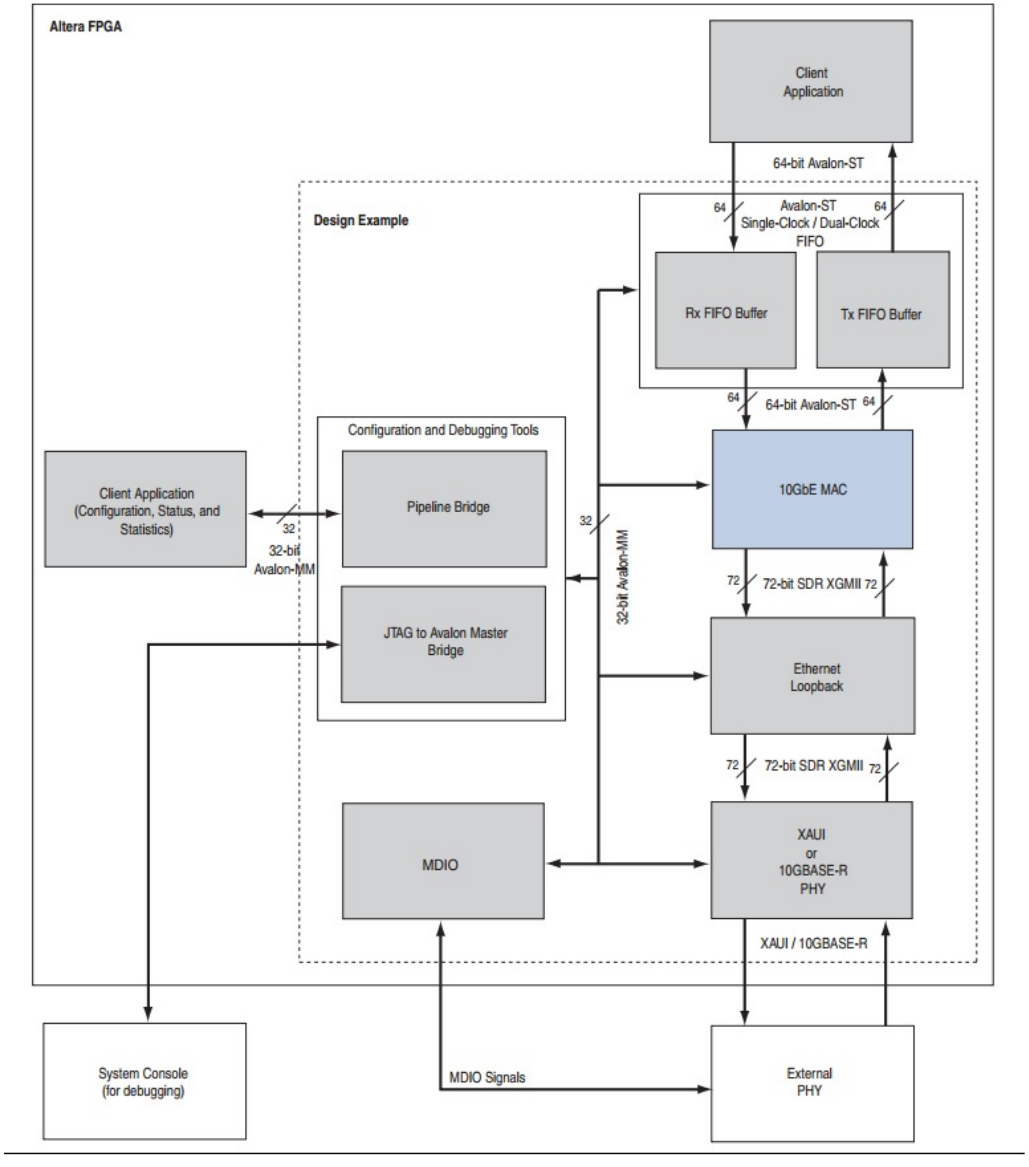
Source: Altera

# Avalon ST Signals

| Signal type | Width | Direction | Description |
|---|---|---|---|
| **Fundamental signals** | | | |
| ready | 1 | Sink → Source | Indicates the sink can accept data |
| valid | 1 | Source → Sink | Qualifies all source to sink signals |
| data | 1-4096 | Source → Sink | Payload of the information being transmitted |
| channel | 1-128 | Source → Sink | Channel number for data being transferred (if multiple channels supported) |
| error | 1-255 | Source → Sink | Bit mask marks errors affecting the data being transferred |
| **Packet transfer signals** | | | |
| startofpacket | 1 | Source → Sink | Marks the beginning of the packet |
| endofpacket | 1 | Source → Sink | Marks the end of the packet |
| empty | 1-8 | Source → Sink | Indicates the number of symbols that are empty during cycles that contain the end of a packet |

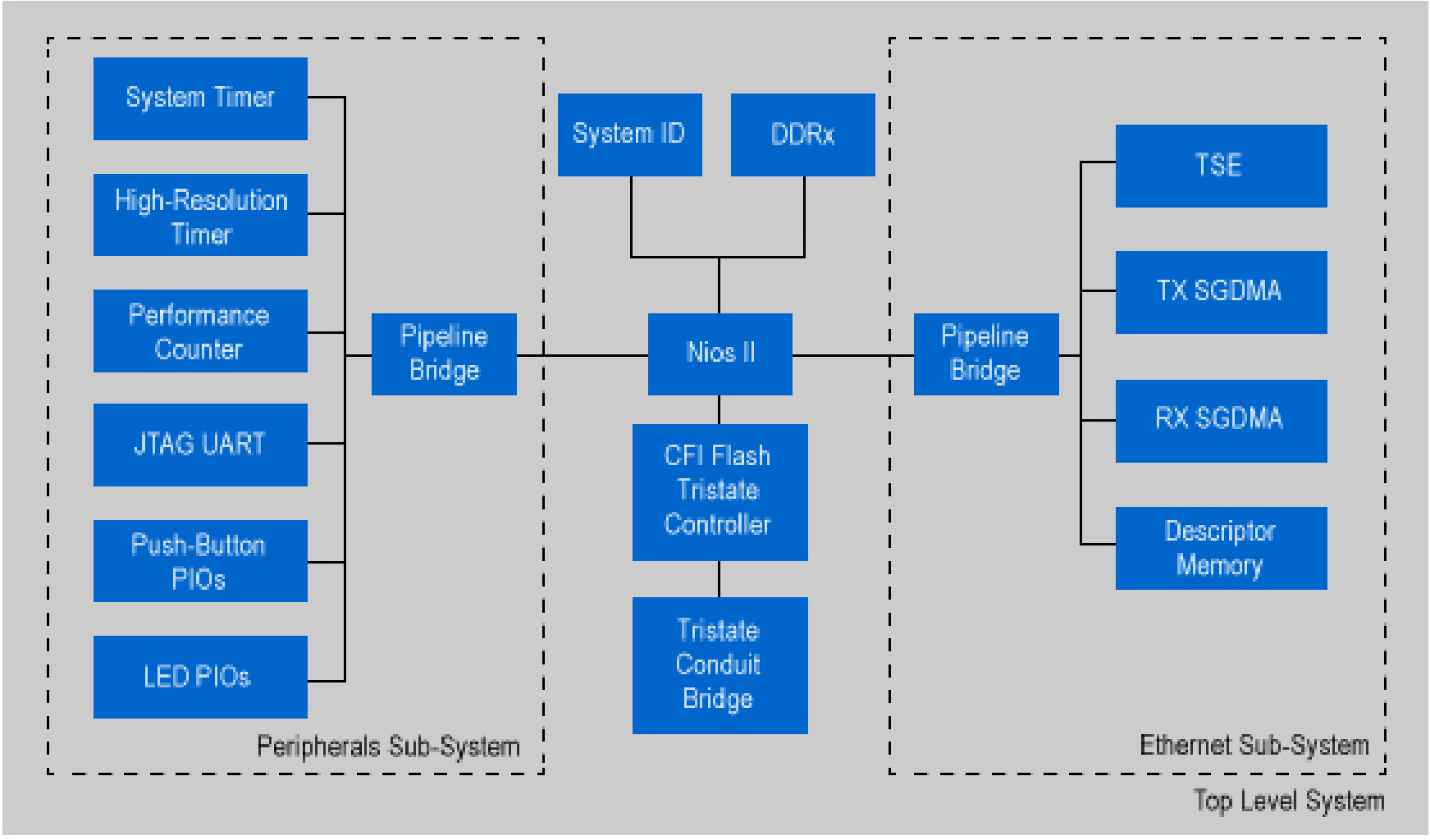
Source: Altera

# Example Qsys Layout: 10Gb Reference Design



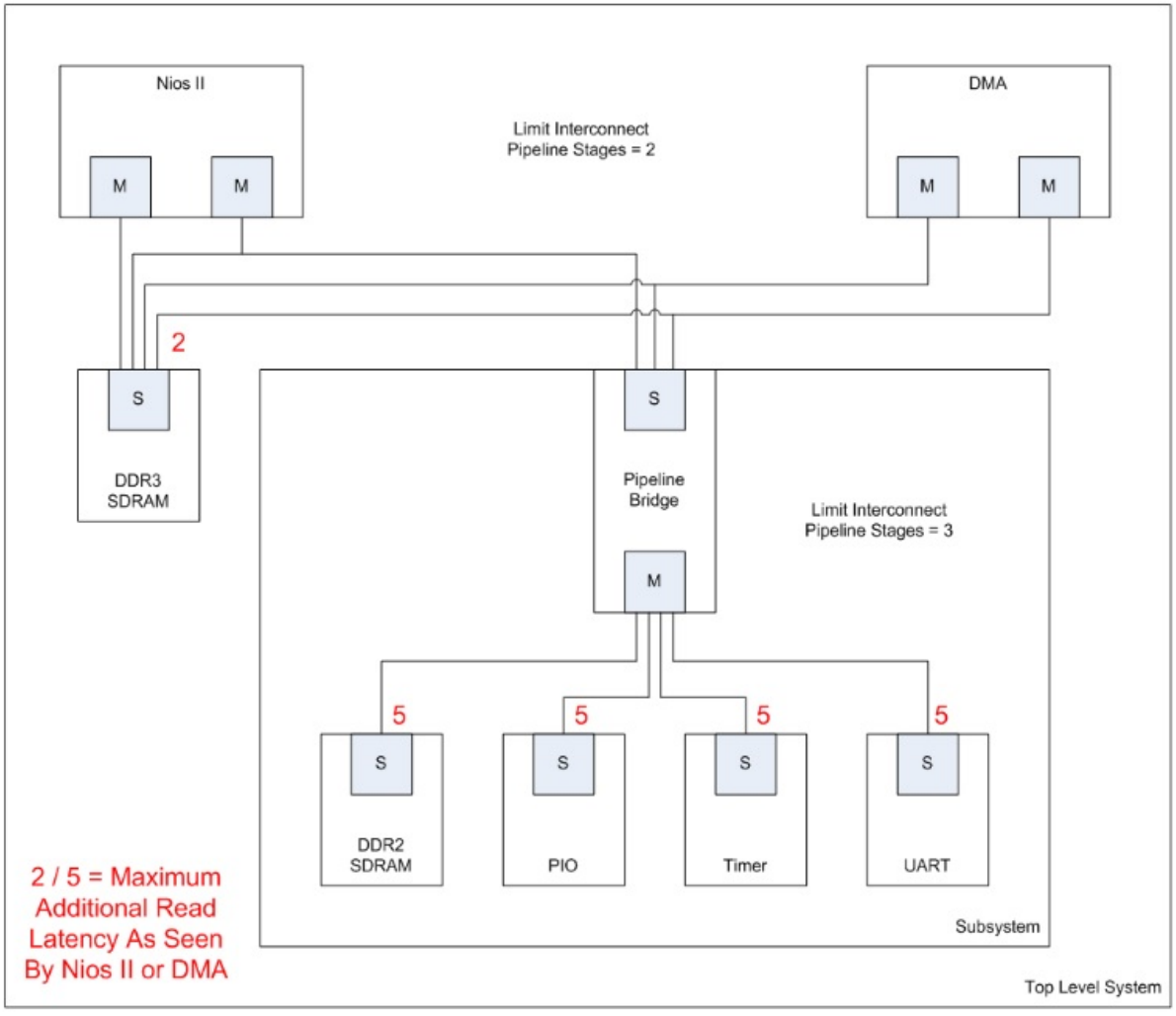

Source: Altera

# Advanced: Qsys Hierarchical Designs



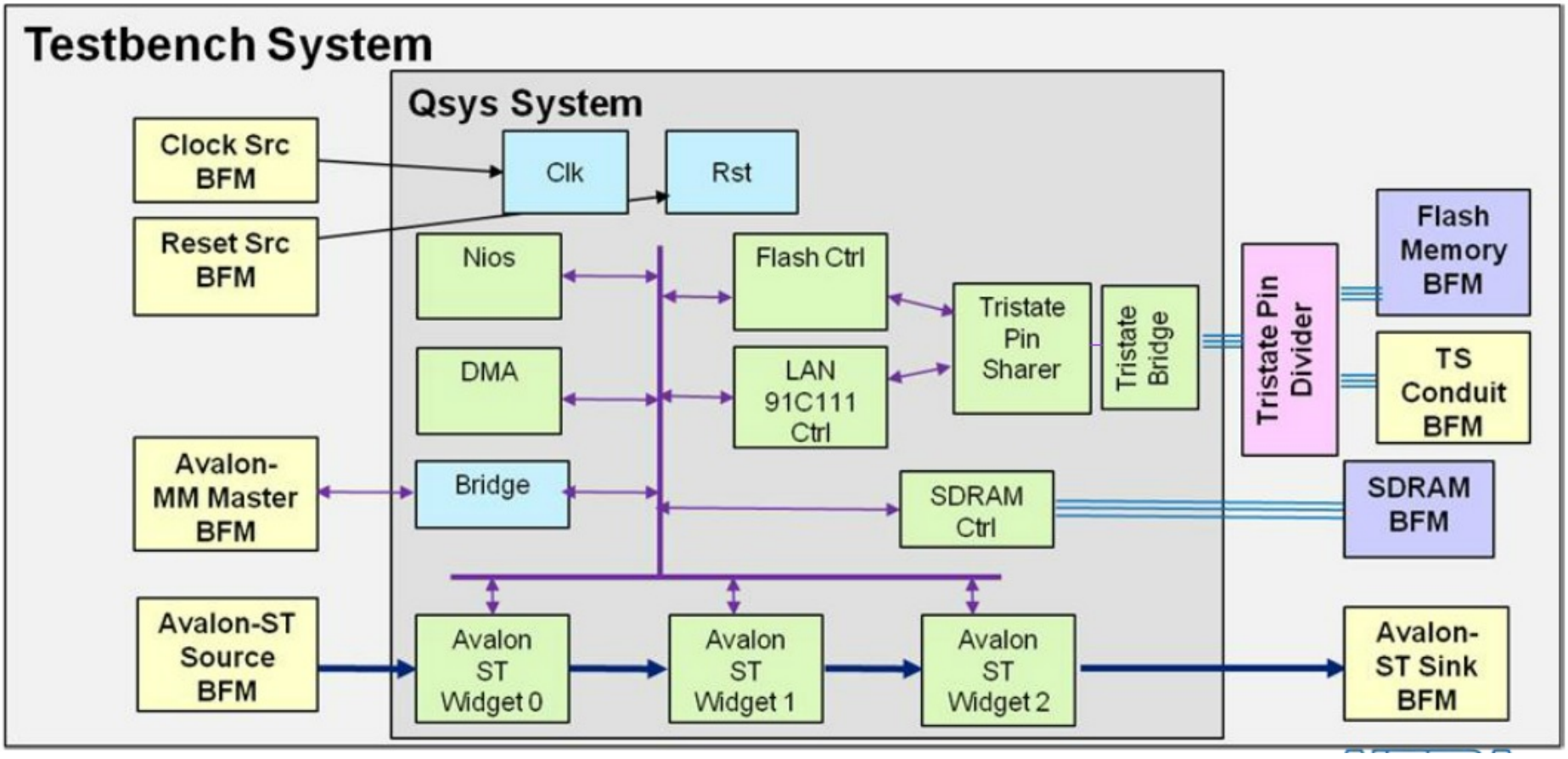

Source: Altera

# Advanced: Qsys Automatic Pipelining




Source: Altera

# Advanced: Qsys Testbench Generation




Source: Altera

# Additional References

Altera online training lectures: (HIGHLY recommended; many of these slides are taken directly from them)

`http://www.altera.com/education/training/curriculum/trn-curriculum.html`

Introduction to Qsys

Advanced System Design Using Qsys

Custom IP Development Using Avalon and AXI Interfaces