

Embedded FPGA using the NIOS Processor



- Larry Landis – University Outreach Senior Manager, Intel Programmable Solutions Group
- 36 years in semiconductor industry – design, sales, marketing, training
- Adjunct Lecturer for ASICs/Digital Electronics Course– UC Berkeley, Santa Clara University, Sacramento State



BSEE



MSEE



Adjunct Lecturer



Intel® FPGA

Academic Ecosystem



- Increase Intel® FPGA presence in academia
- Train the next generation of FPGA Designers



- Academic access to the latest generation of Intel FPGAs for research



- Nurture the talent pipeline for Intel and our customers to highly skilled FPGA Designers

What is Embedded Design?

- Your Definition?
- What are some properties of an Embedded System?

Quadcopter



Micro SD Card?



Blu-Ray / Remote



Programmable
Thermostat



Roomba

What is Embedded Design?

- Your Definition?
- What are some properties of an Embedded System?

Quadcopter



Micro SD Card?



Blu-Ray / Remote



Programmable
Thermostat



Roomba

What is an Embedded System?

- The textbook definitions all have their limits
- An embedded system is simultaneously:
 1. “a digital system that provides service as part of a larger system” – G. De Micheli
 2. “any device that includes a programmable computer but is not itself a general-purpose computer” – M. Wolf
 3. “a less visible computer” - E. Lee
 4. “a single-functioned, tightly constrained, reactive computing system” – F. Vahid
 5. “a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints” – Wikipedia

Scale of Embedded Devices

- Even as Electrical and Computer Engineers it can be easy to understate the scale (both in terms of size and ubiquity) of embedded devices



- SanDisk microSD card
- 100 MHz ARM CPU



- But for what reason?



- Apple Lightning Digital AV Adapter
- 256 MB DDR2, ARM SoC

Embedded Design Skills

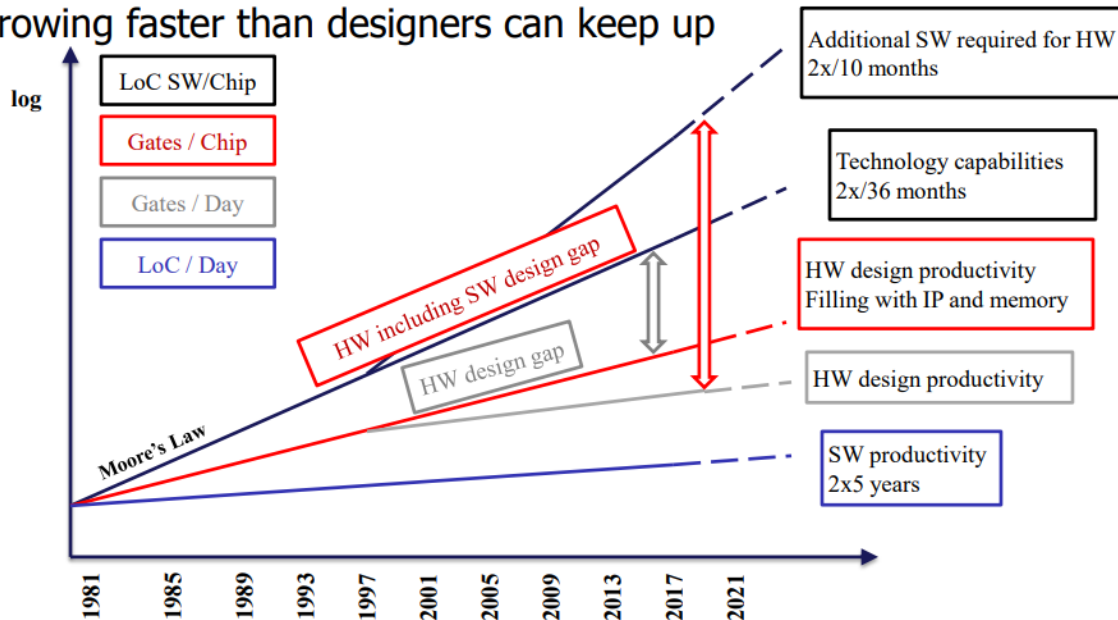
- SW development
- Peripheral and Memory Interfacing
- CPU Architecture
- RTL Design
- Circuits and Signals
- Critical Thinking
- Hard Work, Time Consuming

Design Challenges

- How much hardware do we need?
 - How big is the CPU? Memory?
- How do we meet our deadlines?
 - Faster hardware or cleverer software?
- How do we minimize power?
 - Turn off unnecessary logic?
 - Reduce memory accesses?
 - Data compression?
 - Multi-objective optimization in a vast design space

Designer Productivity Gap

- Embedded systems today are characterized by rapidly expanding functionality coupled with shrinking time to market
- Hardware capabilities (and accompanying software needs) are growing faster than designers can keep up



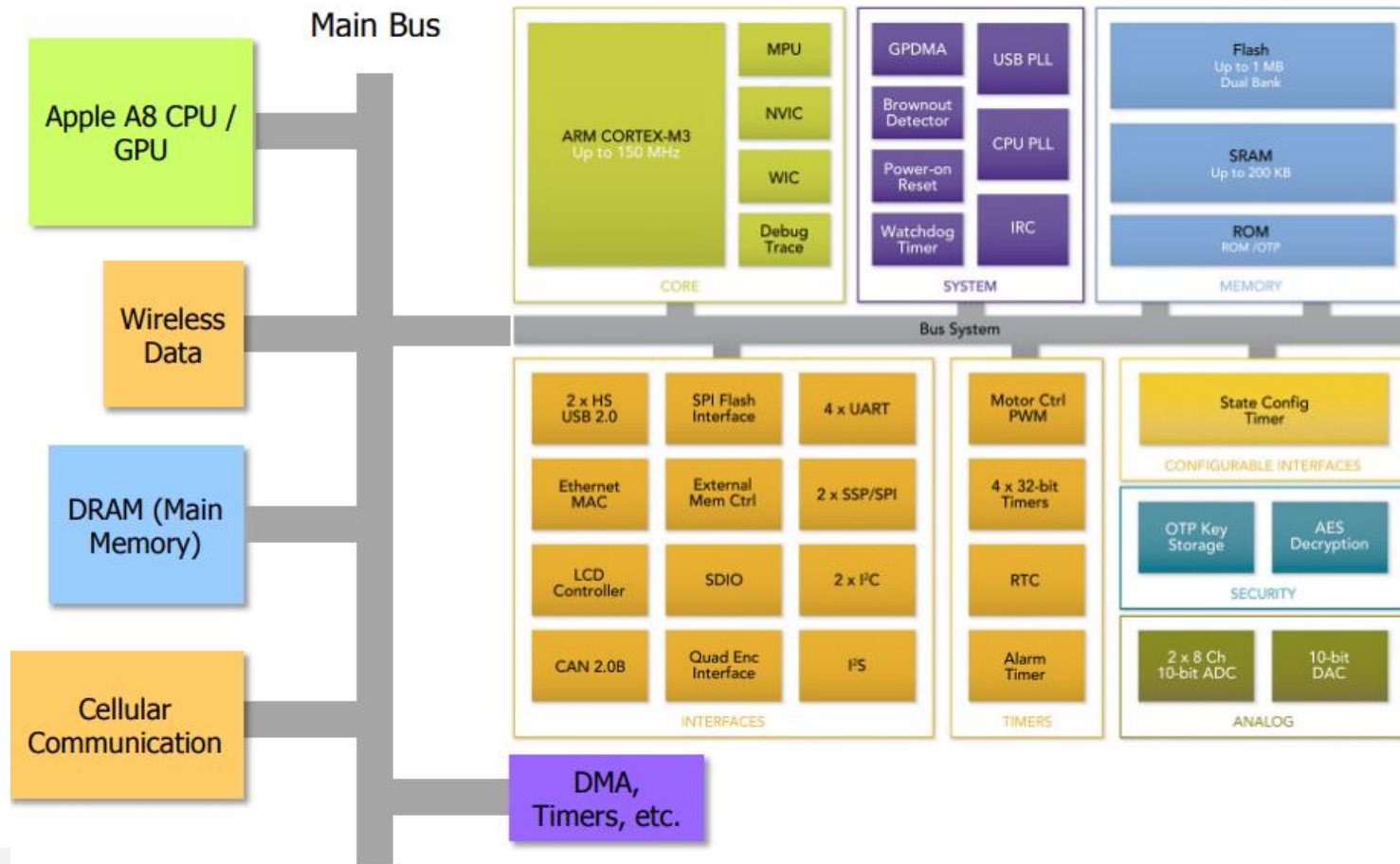
What makes embedded system design uniquely challenging?

- System reliability needs:
 - Can't crash, may not be able to reboot
 - Can't necessarily receive firmware / software updates – System performance and power constraints:
 - Real-time issues in many applications
 - (Potentially) limited memory and processing power –
- System cost:
 - Fast time to market on new products
 - Typically very cost competitive

Coping with Complexity

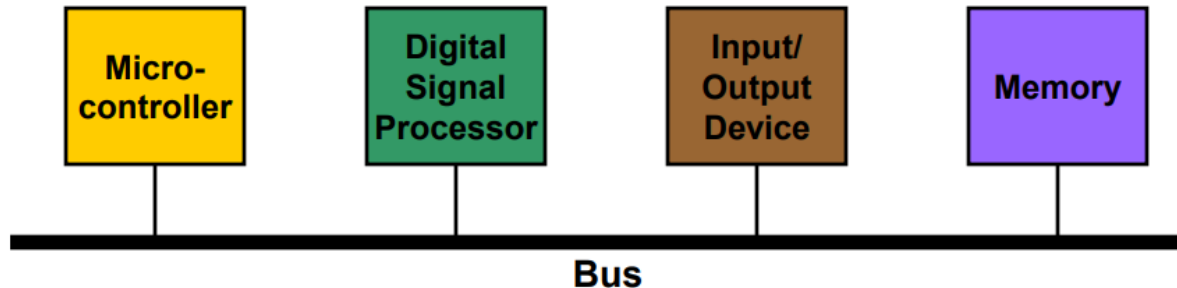
- Intellectual Property (IP) reuse:
 - Pre-designed hardware modules
 - Either soft cores (e.g. source, netlists) or hard cores (transistor or other layout)
 - Range in functionality from simple multipliers, to memories, to interfacing logic, to processors
 - Pre-designed software as well (e.g. drivers, OS)
 - Typically follow some set interfacing standards
- Platform-based design:
 - Design philosophy that focuses on IP-centric design and reuse
 - Platform: a customizable design for a particular type of system, consisting of embedded processors, peripherals, and software

iphone

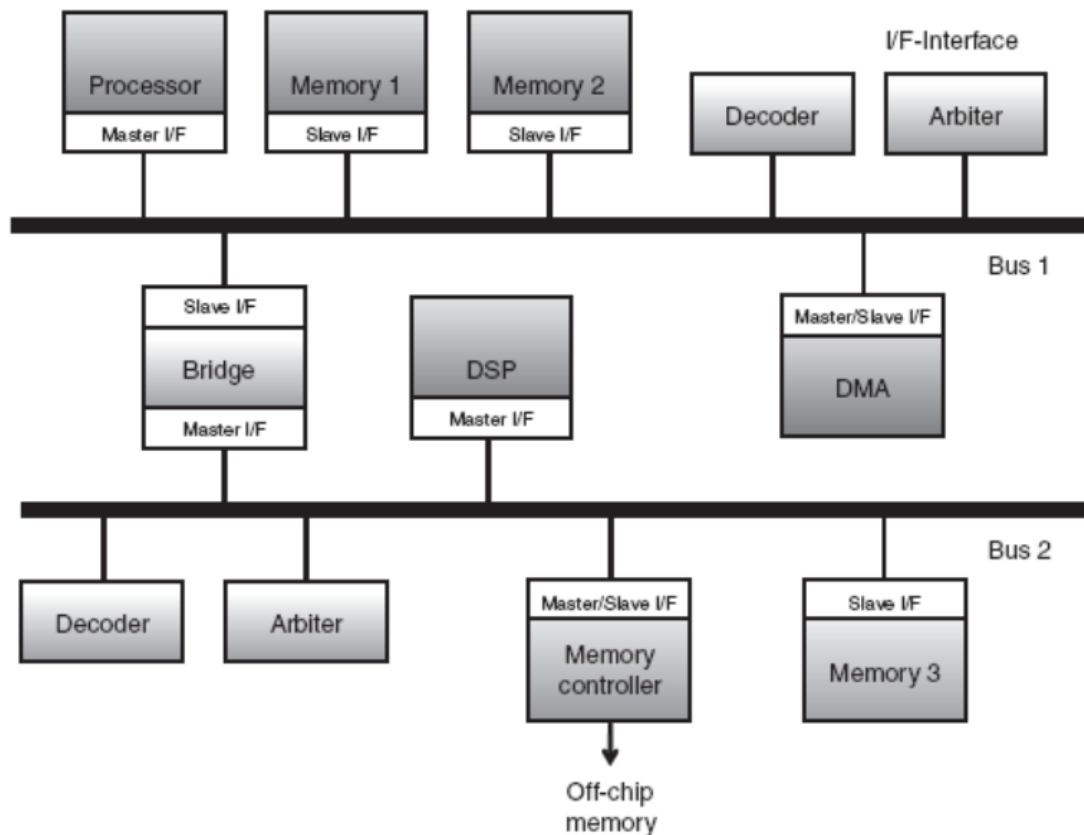


On Chip Communication Architectures

- Buses are the simplest and most widely used SoC interconnection networks
- Bus:
 - A collection of signals (wires) to which one or more communicating IP components are connected
 - A protocol associated with that communication
- Only one IP component can transfer data on the shared bus at any given time



Bus Terminology



Bus Terminology

- Master (or Initiator)
 - IP component that initiates a read or write data transfer
- Slave (or Target)
 - IP component that does not initiate transfers and only responds to incoming transfer requests
- Arbiter
 - Controls access to the shared bus
 - Uses an arbitration scheme to select master to grant access to bus
- Decoder
 - Determines which component a transfer is intended for
- Bridge
 - Connects two buses
 - Acts as slave on one side and master on the other

Bus Signal Lines

- A bus typically consists of three types of signal lines:

- Address

- Carry address of destination for which transfer is initiated
- Can be shared or separate for read, write data

- Data

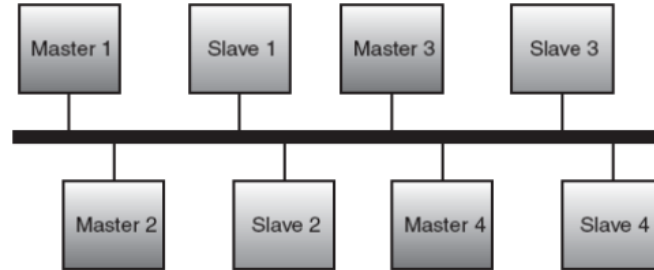
- Carry information between source and destination components
- Can be shared or separate for read, write data
- Choice of data width critical for application performance

- Control

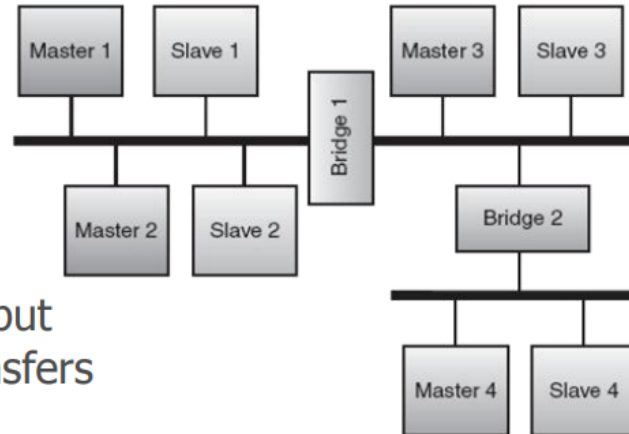
- Requests and acknowledgements
- Specify more information about type of data transfer
- Ex: byte enable, burst size, cacheable/bufferable, writeback/through, ...

Bus Topologies

- Shared bus

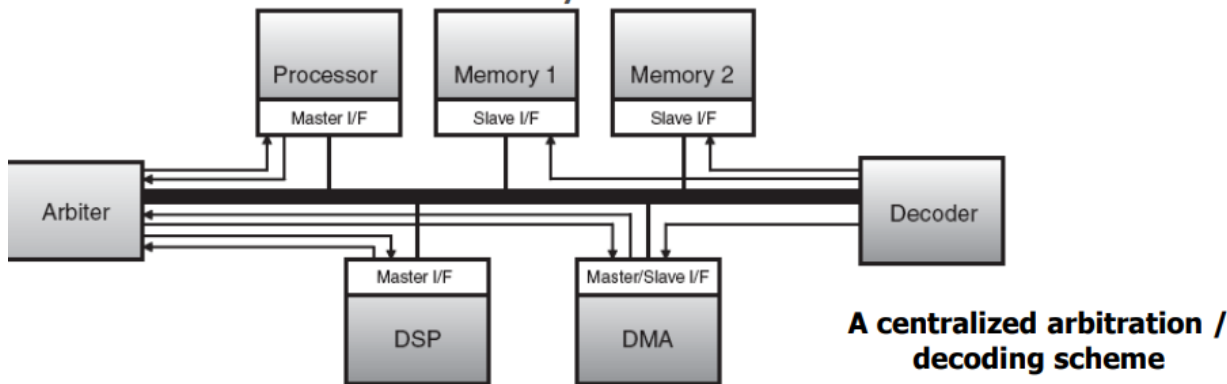


- Hierarchical shared bus
 - Improves system throughput
 - Multiple simultaneous transfers



Decoding and Arbitration

- A bus implementation includes logic for both decoding and arbitration (either distributed or centralized):
 - Decoding – determining the target for any transfer initiated by the master
 - Arbitration – deciding which master can use the shared bus if more than one master requested bus access simultaneously



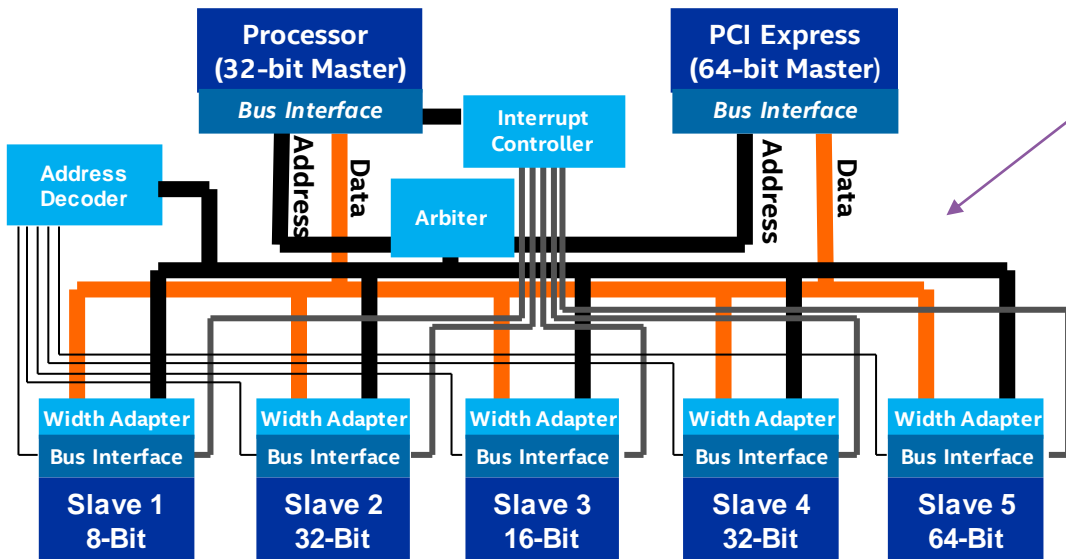
Bus Standards

- Bus standards are useful for defining a specific interface and data transfer protocol
- Ideally, all IP in a design are compatible
- In reality, several competing standards for SoC design:
 - AMBA/AXI (ARM) – Most Popular
 - CoreConnect (IBM)
 - Avalon (Altera/Intel FPGA)
 - Wishbone (OpenCores)
 - Quick Path (Intel)
 - STBus (STMicroelectronics)

TRADITIONAL SYSTEM DESIGN

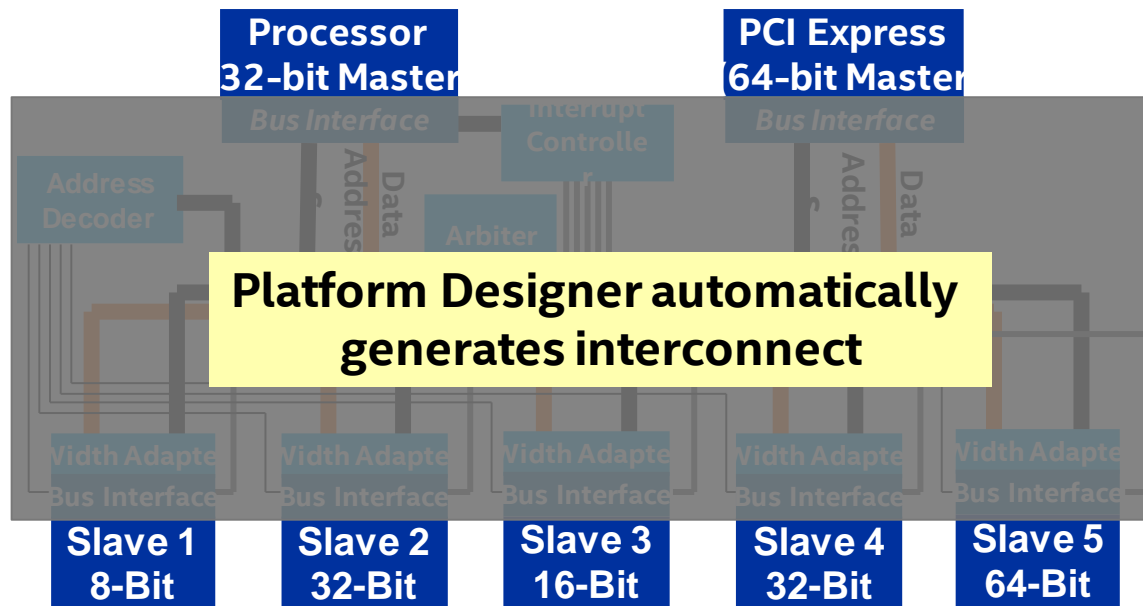
- Different interfaces (some standard, some non-standard)
- Significant engineering work required to design custom interface logic
- Integrating design blocks and intellectual property (IP) is tedious and error-prone

*Lots of wires!
Error prone to get
working!*




AUTOMATIC INTERCONNECT GENERATION

- Avoids error-prone integration
- Saves development time with automatic logic & HDL generation
- Enables you to focus on value-add blocks



PLATFORM DESIGNER FEATURES

- Easy IP integration with switch fabric connectivity
 - Dynamic system generation
 - High-level system visualization
 - Custom IP authoring
 - IP verification and bus functional models (BFMs)
 - Simulation support for ModelSim® and other 3rd-party simulation tools
 - Real-time system debug through tools like System Console
- 
- Today's Lab

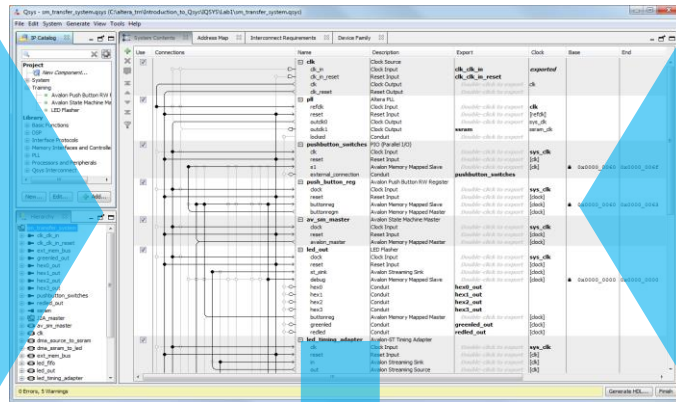
EASY TO USE SYSTEM-INTEGRATION UI



Catalog of available IP

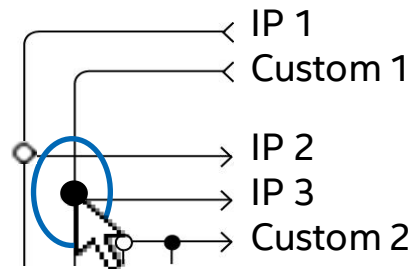
- Interface protocols
- Memory
- DSP
- Embedded
- Bridges
- PLL
- Custom systems

**Accelerate
development**



HDL

Connect custom IP and systems



**Simplify
integration**

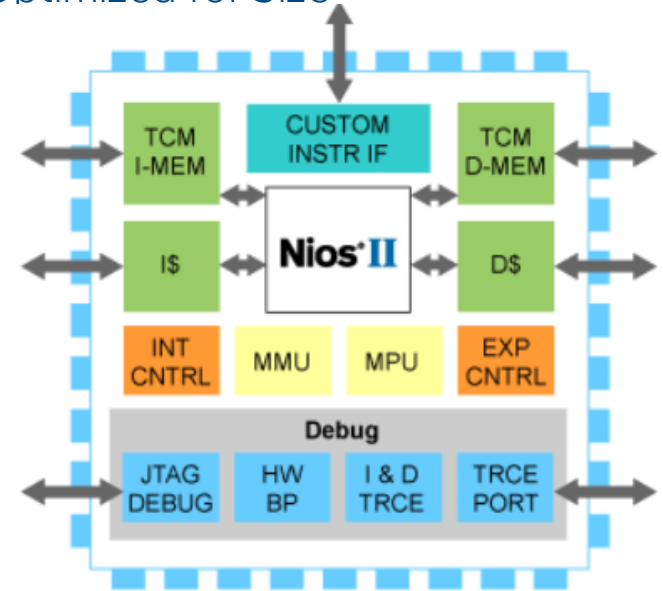
Automate error-prone integration tasks

Nios[®] II Processor

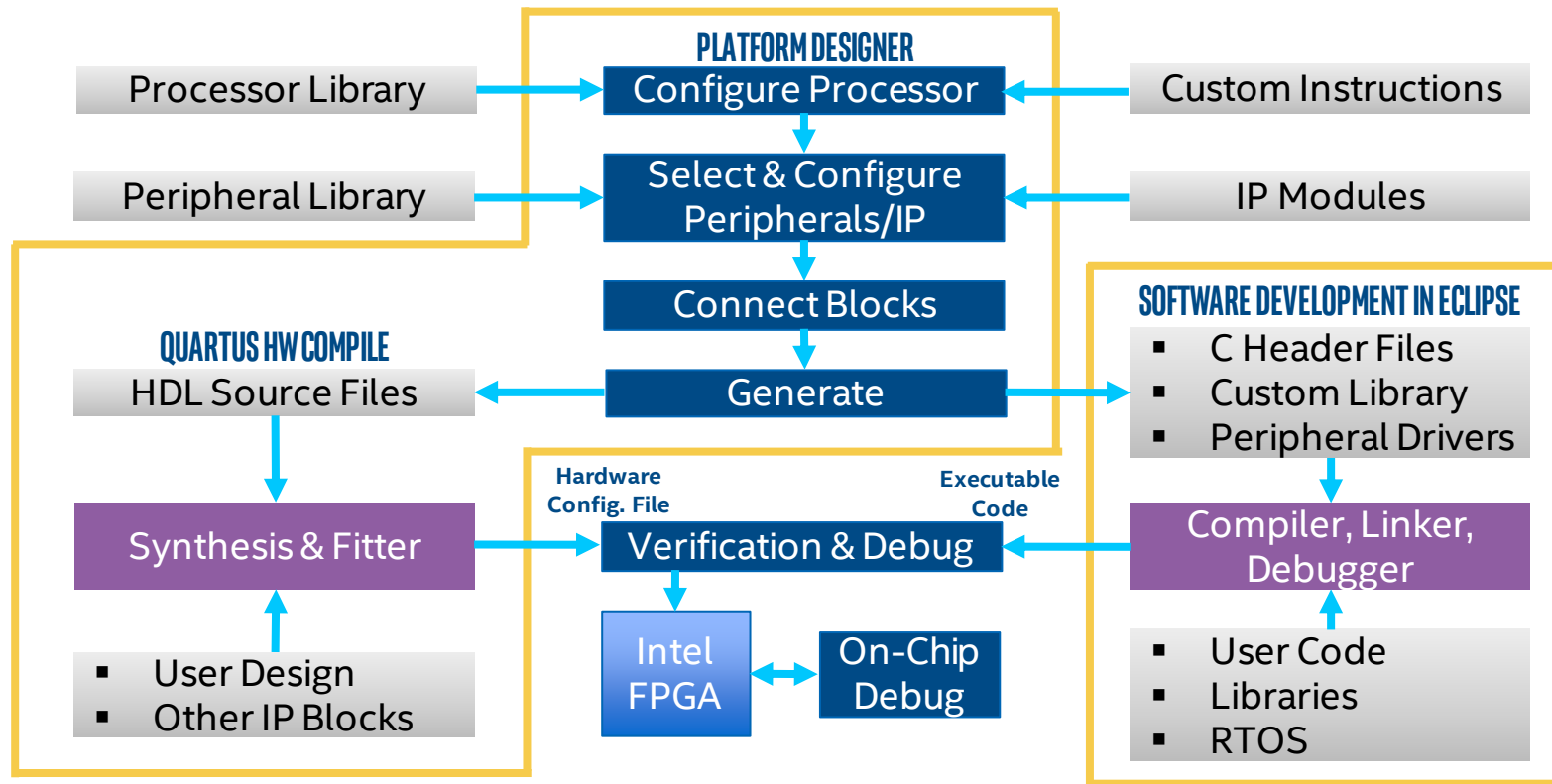


NIOS II PROCESSOR

- Intel's 2nd Generation 32-Bit RISC Soft Microprocessor
- Two Variants: **Fast - Optimized for Speed; Economy - Optimized for Size**
- Configurable features
 - Cache size
 - Tightly coupled memories
 - Arithmetic implementation
 - Interrupt controller
 - MMU/MPU
- Custom instructions and peripherals
- Compatible with all Intel FPGAs



NIOS II DESIGN FLOW



NIOS II EMBEDDED DESIGN SUITE FEATURES

Software Build Tools (SBT)

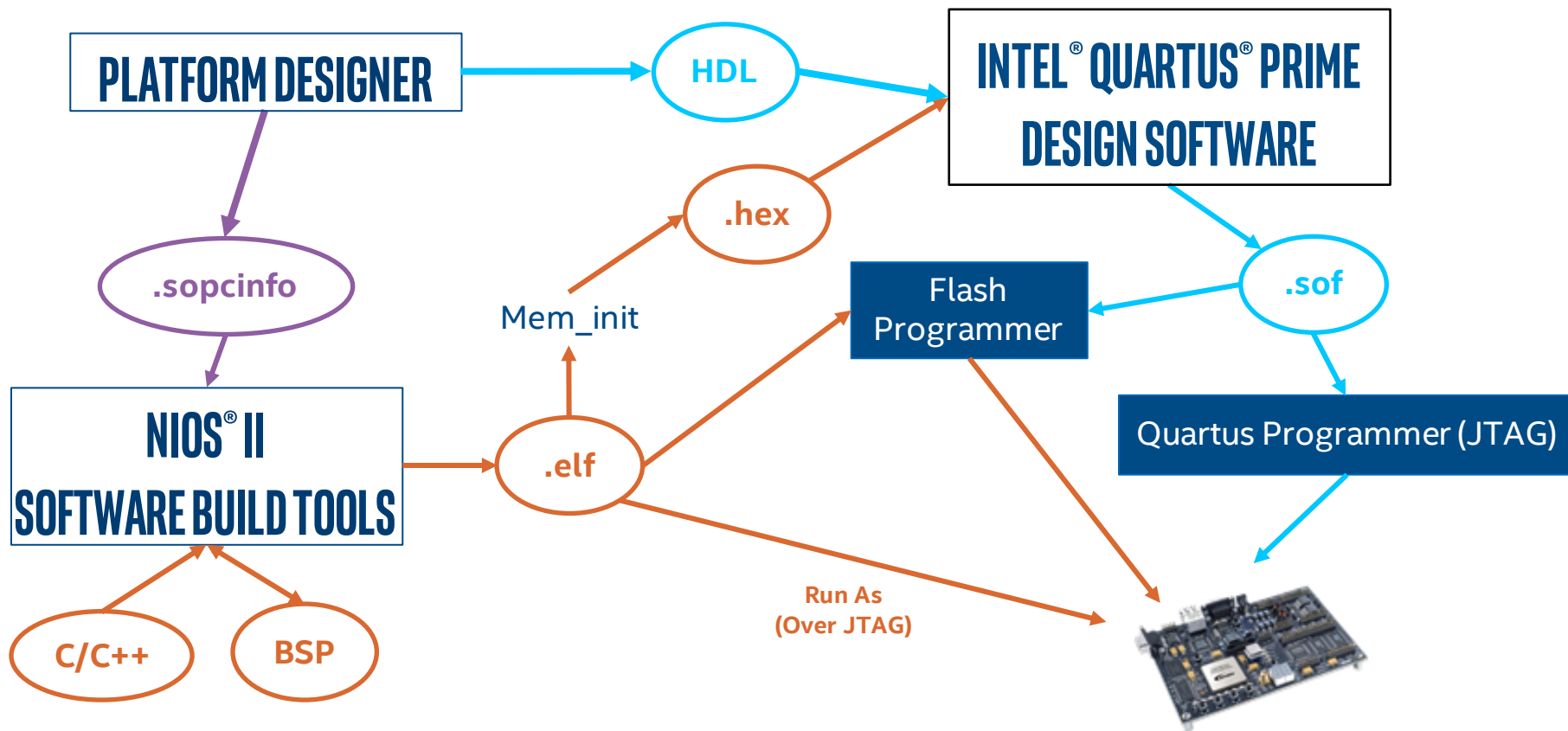
- Proprietary and open-source tools for creating Nios® II programs
- Commands, utilities & scripts
- GUI for developing and debugging software
- Plug-ins to industry standard Eclipse IDE
- Automated board support package (BSP) creation

Hardware Abstraction Layer (HAL)

- Unix-style API
- Interface with peripherals, timers, interrupts, etc.
- Device access and initialization
- All hardware related software libraries automatically updated

<code>_exit()</code>	<code>gettimeofday()</code>	<code>sbrk()</code>	<code>open()</code>
<code>close()</code>	<code>ioctl()</code>	<code>stat()</code>	<code>opendir()</code>
<code>closedir()</code>	<code>isatty()</code>	<code>usleep()</code>	<code>read()</code>
<code>fstat()</code>	<code>kill()</code>	<code>wait()</code>	<code>readdir()</code>
<code>getpid()</code>	<code>lseek()</code>	<code>write()</code>	<code>rewinddir()</code>

29 NIOS II SBT FLOW



ECOSYSTEM & SUPPORT



- Quartus Prime debugging tools available for all Nios systems
- System Console
 - Perform register/address level transactions through a scriptable Tcl environment without software
- External memory interface (EMIF) debug toolkit
- Transceiver toolkit
- SignalTap™ II logic analyzer
 - Signal and logic level FPGA hardware debug

ADDITIONAL NIOS® II RESOURCES

- [Nios® II Documentation](#) on intel.com
 - Including One-Click Download zip file of all available documentation
 - Nios® II Hardware and Software Developer's Handbooks
- [Intel Forum](#) - Thousands of users and topics
- [Design Store](#) - Lots of designs (search for Nios, Qsys, or Platform Designer)
- [Build your own custom components](#)
- Eclipse Help Content and Welcome Page

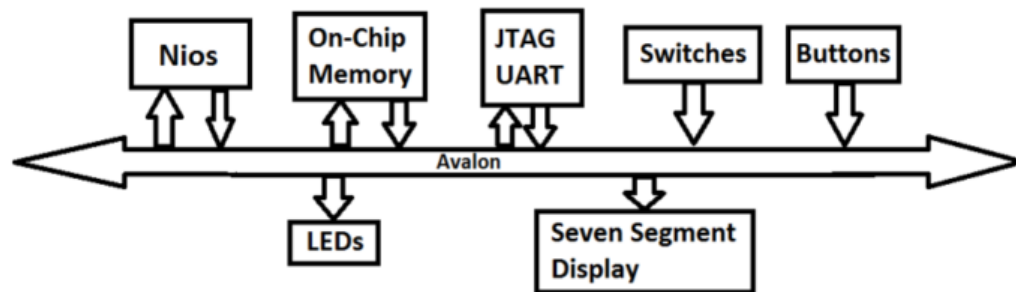
EMBEDDED NIOS LAB



OBJECTIVES

- Upon completion of this workshop, you should have an understanding of:
 1. Platform Designer system development tools - basic features and how to run the tool
 2. How to develop applications on the Nios[®] II processor
 3. Eclipse Integrated Development Environment (IDE) and development of “bare metal” applications utilizing Nios and associated FPGA hardware

1. Build system block diagram in Qsys
2. Connect to a top level Verilog module
3. Export memory map into Eclipse environment
4. Build project and compile code
5. Download hardware.
6. Download software
7. Try some other cool code and download it.



1. Follow connections in exactly! Double check inputs vs. outputs.
2. Name things exactly! No spaces in files OR folder names.
3. The instruction and data master is connected to the on chip memory (not just data master)!
4. Remember **Platform Designer** → **System** → **Assign Base Addresses** after edits
5. When connecting Eclipse to the board, “Refresh Target” is hidden on the right.
6. To reprogram HW, first stop eclipse from running software (use “stop” button) or the USB blaster will be hung!
7. If HW fails to program, hit Start a second time!
8. Have fun and try more embedded projects on your own!