Digital Systems Design

Introduction to System on a Programmable Chip

System on a Programmable Chip

- Generally involves utilization of a large FPGA
  - Large number of logic elements
  - Memory on the device
- Typically includes
  - An Intellectual Property (IP) processor core
    - Common examples include various microcontrollers, microprocessors, etc.
  - A selection of custom hardware
    - Interfaces, busses
- Basic premise is to have all (or a large majority) of the components on a single chip
  - System-on-a-chip (SOC)
- Since we are using an FPGA we may call this a system-on-a-programmable-chip (SOPC)
Processor Cores

- Processor cores can be classified as **hard** or **soft**
  - Refers to the flexibility or configurability of the core
- Hard cores are less configurable but they have higher performance than soft cores
- Hard processor cores use an embedded processor core (in silicon) in addition to the FPGA’s normal memory and logic elements
- Examples
  - Altera offers an ARM processor core in some of its FPGAs
  - Xilinx also offers ARM processor core in some of its FPGAs

Processor Cores (continued)

- Soft cores (Altera’s NIOS II processor is one example) use existing FPGA logic elements to implement the processor logic
- Can be very feature rich, allowing specification of parameters such as
  - Memory/datapath width
  - ALU functionality
  - Number and types of peripherals
  - Memory and I/O address space
- Parameters specified at (hardware) compile time
- Disadvantages
  - Typically have slower clock rates and higher power consumption than hard core counterparts
**SOPC Development Tools**

- A CAD tool, specific to each soft processor core, is used to specify processor options
  - Register file size, hardware multiply and divide support, floating point support, interrupts, I/O hardware, custom hardware
- The tool outputs a synthesizable HDL model of the processor
  - HDL source code may be protected and not viewable
- After any additional (user) logic is added, the design can be synthesized using the CAD synthesis tool
- Application programs (usually in C or C++) are compiled by a custom compiler provided for the processor core

---

**SOPC Design Flow**

- SOPC processor core synthesis (and associated tools) are generally a superset of traditional CAD tools
  - Altera Qsys
- Various commercial and open source processor cores
  - NIOS II
  - www.opencores.org
  - www.leox.org
- Commercial soft processor cores tend to be more efficiently implemented on FPGAs
  - Typically optimized by manufacturer for a particular FPGA family (or families)
- Hardware and software can be designed concurrently
  - Hardware/Software Co-Design
**SOPC Design Flow**

- **Design Entry Tool**
- **Process Core Configuration Tool**
- **FPGA Synthesis Tool**
- **FPGA Place and Route Tool**
- **Program FPGA & Initialize Memory**
- **User Logic**
- **Processor**
- **Memory**
- **Software Design**
- **Application Source Code**
- **C/C++ Compiler for Processor**
- **Binary Program/Data Files**
- **Operating System and Libraries (optional)**

**Initializing Memory**

- Once the program/data file has been generated it must be loaded onto the processor's program and data memories.
- Can be accomplished in several ways
  - **On-chip Memory**
    - If the application is small, it may fit in the memory blocks available on the FPGA.
    - Initialization is through standard FPGA tools.
    - On-chip memory is typically limited so this solution is not often an option.
  - **Bootloader**
    - A small program, usually loaded onto on-chip memory or a EPROM, that is responsible for communicating with another device (a PC) and downloading an application program into external memory (SRAM, SDRAM, FLASH, etc.).
    - After downloading, control is transferred to the application program.
  - **External Non-volatile Storage**
Hardware Bootloader

- A hardware bootloader provides functionality similar to the software bootloader
  - Implemented in dedicated logic within the processor core
- Upon power up (or configuration), the bootloader stalls the processor and downloads application code (typically via a serial or JTAG interface)
  - Bootloader can start/stop the processor
  - Has access to at least some of the register set
  - Typically has direct access to the processor’s memory or memory registers in the datapath
- Altera’s NIOS II processor supports a hardware bootloader within a JTAG debug module
- Can support downloading new application code without recompiling (or even reconfiguring) the soft core processor

SOPC Design Versus Traditional Design

- Traditional design includes
  - ASIC – Application Specific Integrated Circuit
  - Fixed-Processor Design
- SOPC design advantages
  - Reconfigurable nature
  - Short development cycle
- SOPC design disadvantages
  - Higher unit costs in production
  - Relatively high power consumption
Flexible Hardware Benefits

- Features and specifications modified throughout the design cycle
  - Marketing detects a shift in demand for additional features
    - Cell phones with cameras, MP3 players
  - A protocol or specification is updated
    - Introduction of USB 2.0 standard for example
- In traditional design modalities (ASIC and fixed processor), these changes could result in
  - ASIC design
  - Processor selection
  - Printed circuit board design
- These things tend to increase product time-to-market and product cost
- Also can affect multiple generations/versions of a single product or multiple product lines
  - Use of a single circuit board for multiple product lines
  - Use of a single circuit board design for multiple versions of a single product

Comparing SOPC, ASIC and Fixed Processor Design Modalities

<table>
<thead>
<tr>
<th>Feature</th>
<th>SOPC</th>
<th>ASIC</th>
<th>Fixed Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>S/W Flexibility</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>H/W Flexibility</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Reconfigurability</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Development cost/time</td>
<td>●</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>Peripheral Equipment Costs</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Performance</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Production Cost</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Power Efficiency</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>

● – Good; ○ – Moderate; ○ – Poor
Analyzing System Requirements

- The development flow begins with predesign activity which includes an analysis of the application requirements, such as the following questions:
  - What computational performance does the application require?
  - How much bandwidth or throughput does the application require?
  - What types of interfaces does the application require?
  - Does the application require multithreaded software?
Analyzing System Requirements

• Based on the answers to these questions, you can determine the concrete system requirements, such as:
  – Which Nios II processor core to use: smaller or faster.
  – What components the design requires and how many of each kind.
  – Which real-time operating system (RTOS) to use, if any.
  – Where hardware acceleration logic can dramatically improve system performance.
• Analyzing these topics involve both the hardware and software teams

Defining and Generating the System in Qsys

• After analyzing the system hardware requirements, you use Qsys to specify the Nios II processor core(s), memory, and other components your system requires.
• Qsys automatically generates the interconnect logic to integrate the components in the hardware system.
• The primary outputs of Qsys are the following file types:
  – Qsys Design File (.qsys) -- Contains hardware contents of the Qsys system.
  – SOPC Information File (.sopcinfo) -- Contains a description of the contents of the .qsys file in Extensible Markup Language File (.xml) format.
    • The Nios II Embedded Design Suite (EDS) uses the .sopcinfo file to create software for the target hardware.
  – Hardware description language (HDL) files -- Hardware design files that describe the Qsys system.
    • The Quartus software uses the HDL files to compile the overall FPGA design into an SRAM Object File (.sof).
Integrating the Qsys System into the Quartus Project

• After generating the Nios II system using Qsys, you integrate it into the Quartus project.
• Using the Quartus software, you perform all tasks required to create the final FPGA hardware design.
• Using the Quartus software, you also assign pin locations for I/O signals, specify timing requirements, and apply other design constraints.
• Finally, you compile the Quartus project to produce a .sof to configure the FPGA.

Developing Software with the Nios II Software Build Tools for Eclipse

• Using the Nios II Software Build Tools (SBT) for Eclipse™, you perform all software development tasks for your Nios II processor system.
  – After you generate the system with Qsys, you can begin designing your C/C++ application code immediately with the Nios II SBT for Eclipse.
  – Altera provides component drivers and a hardware abstraction layer (HAL) which allows you to write Nios II programs quickly and independently of the low-level hardware details.
  – In addition to your application code, you can design and reuse custom libraries in your Nios II SBT for Eclipse projects.
  – To create a new Nios II C/C++ application project, the Nios II SBT for Eclipse uses information from the .sopcinfo file.
Developing Software with the Nios II Software Build Tools for Eclipse

• The Nios II SBT for Eclipse can produce several outputs including:
  – system.h file -- Defines symbols for referencing the hardware in the system.
    • The Nios II SBT for Eclipse automatically create this file when you create a new board support package (BSP).
  – Executable and Linking Format File (.elf)—Is the result of compiling a C/C++ application project, that you can download directly to the Nios II processor.