Using Altera CAD tools for NIOS Development

...adapted from Koren/Moritz, Burleson, and Wolf, Computers as Components, Morgan Kaufman, 2005

IP-- Intellectual Property in FPGAs

• Don’t re-invent the wheel, use IP!
• Shorter time to market
• Cheaper than developing yourself (lengthy design and verification times)
• Right first time (proven in many projects)
• Higher performance
• Shift functionality from SW to FPGA (better performance and cost)
• Source code delivery
• No royalty, you only pay once! (some vendors)
• Straight-forward standard contract (sign once)
Definition

- **Embedded system**: any device that includes a programmable computer but is not itself a general-purpose computer.

- Take advantage of application characteristics to optimize the design:
  - don’t need all the general-purpose bells and whistles.

Embedding a computer: a very simplified view...
Examples

- Personal digital assistant (PDA).
- Printer.
- Cell phone.
- Automobile: engine, brakes, dash, etc.
- Television.
- Household appliances.

More examples

- Cell phones, Ipod and MP 3 players, Webcams, Navigation Systems
- Routers, Blade servers, Wireless PC cards
- Automobiles, Car Alarms, Keyless Entry Systems
- Building Security, card swiping systems
- Embedded Medical Devices - Pacemakers
### Characteristics of embedded systems

- Sophisticated functionality.
- Real-time operation.
- Low manufacturing cost.
- Low power.
- *Reliable and secure*
- Designed to tight deadlines by small teams.

### Functional complexity

- Often have to run sophisticated algorithms or multiple algorithms.
  - Cell phone, laser printer.
- Often provide sophisticated user interfaces.
Real-time operation

- Must finish operations by deadlines.
  - **Hard real time:** missing deadline causes failure.
  - **Soft real time:** missing deadline results in degraded performance.
- Many systems are **multi-rate:** must handle operations at widely varying rates.

Non-functional requirements

- Many embedded systems are mass-market items that must have low manufacturing costs.
  - Limited memory, microprocessor power, etc.
- Power consumption is critical in battery-powered devices.
  - Excessive power consumption increases system cost even in wall-powered devices.
Design teams

- Often designed by a small team of designers.
- Often must meet tight deadlines.
  - 6 month market window is common.
  - E.g., can’t miss back-to-school window for calculator.

Microprocessor alternatives for embedded systems

- **Ordinary microprocessor**: CPU plus on-chip cache units.
- **Microcontroller**: includes I/O devices, on-board memory.
- **Digital signal processor (DSP)**: microprocessor optimized for digital signal processing.
- Hard core vs. soft core.
- Typical embedded word sizes: 8-bit, 16-bit, 32-bit.
Embedded microprocessors

- ARM, MIPS, Power PC, Freescale, 8051, X86
- Various purposes
  - Networks – MIPS
  - Mobile phone – ARM dominated
  - Industrial – Freescale Coldfire
  - Security – 8051 based, Infineon
  - High performance – X86, Intel Epic, other VLIW and superscalars

Von Neumann CPU Architecture

- Memory holds data and instructions.
- Central processing unit (CPU) fetches instructions from memory.
  - Separation between CPU and memory distinguishes programmable computer.
- CPU registers:
  - program counter (PC)
  - instruction register (IR)
  - general-purpose registers
  - etc
**CPU + memory**

```
200
ADD r5, r1, r3
```

**Harvard architecture**

```
data memory

address
data

program memory

address
instructions

CPU

PC
IR
```
RISC vs. CISC

- **Complex instruction set computer (CISC):**
  - many addressing modes
  - most operations can access memory
  - variable length instructions
- **Reduced instruction set computer (RISC):**
  - only load/store can access memory
  - fixed-length instructions
- **Instruction set architectures** – characteristics:
  - Fixed vs. variable length.
  - Addressing modes.
  - Number of operands.
  - Types of operands.

Pipelining

- Execute several instructions simultaneously but at different stages.
- Pipeline hazards
- Simple three-stage pipe:
Soft Core Processors

- Are soft, i.e. specified through field programming just like programmable logic
  - Shipped as hardware description files, which can be mapped onto FPGA. e.g: Nios 2.
  - Are bundled with software development tools (compiler, simulator, etc.)
- Offer flexibility as microprocessor parameters can be tuned to the application with tight on-chip interconnection with additional circuitry.
- Designs can be marketed quickly. You can test and validate many designs quickly without making any specific board; no soldering and no wiring!

What is Nios 2?

- A 32-bit soft core processor from Altera
- Comes in three flavors: Fast, Standard, Light
- The three cores trade FPGA area and power consumption for speed of execution.
- Is a RISC, Harvard Architecture: Simple instructions, separate data and instruction memories.
- Has 32 levels of interrupts.
- Uses the Avalon Bus interface
- Programs compiled using GNU C/C++ toolchain.
Three forms of Nios 2:

- **Nios II/f**—The Nios II/f “fast” core is designed for fast performance. As a result, this core presents the most configuration options allowing you to fine-tune the processor for performance.
- **Nios II/s**—The Nios II/s “standard” core is designed for small size while maintaining performance.
- **Nios II/e**—The Nios II/e “economy” core is designed to achieve the smallest possible core size. As a result, this core has a limited feature set, and many settings are not available when the Nios II/e core is selected.

All three are available to you!
Selection in SOPC (System On a Programmable Chip):

Why use microprocessors?

- Alternatives: random logic on a field-programmable gate arrays (FPGAs), custom logic, etc.
- Microprocessors are often very efficient: can use same logic to perform many different functions.
- Microprocessors simplify the design of families of products.
Power

- Custom logic is a clear winner for low power devices.
- Modern microprocessors offer features to help control power consumption.
- Software design techniques can help reduce power consumption.

Challenges in embedded system design

- 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?
Design methodologies

- A procedure for designing a system.
- Understanding your methodology helps you ensure you didn’t skip anything.
- Compilers, software engineering tools, computer-aided design (CAD) tools, etc., can be used to:
  - help automate methodology steps;
  - keep track of the methodology itself.

Design goals

- Performance.
  - Overall speed, deadlines.
- Functionality and user interface.
- Manufacturing cost.
- Power consumption.
- Other requirements (physical size, etc.)
Levels of abstraction

- requirements
- specification
- architecture
- component design
- system integration

Top-down vs. bottom-up

- **Top-down design:**
  - start from most abstract description;
  - work to most detailed.
- **Bottom-up design:**
  - work from small components to big system.
- **Real design uses both techniques.**
Designing hardware and software components

- Must spend time architecting the system before you start coding.
- Some components are ready-made, some can be modified from existing designs, others must be designed from scratch.
- Example: SOPC for Hardware design and Nios 2 IDE for Software Design.
SOPC

- **System On a Programmable Chip** – a hardware development tool.
- Used for integrating various hardware components together like:
  - Microprocessors, such as the Nios II processor
  - Timers
  - Serial communication interfaces: UART, SPI
  - General purpose I/O
  - Digital signal processing (DSP) functions
  - Communications peripherals
  - Interfaces to off-chip devices
    - Memory controllers
    - Buses and bridges
    - Application-specific standard products (ASSP)
    - Application-specific integrated circuits (ASIC)
    - Processors
- Generates files in Verilog or VHDL which can be added to the Quartus 2 project.

![Diagram of SOPC components]
Example SOPC system:

SOPC system having NIOS:
The SOPC (System on a Programmable Chip) editor is used to define a NIOS system.

A NIOS system will typically involve a NIOS core and other peripherals (counters, PIO (Parallel I/O), JTAG debug module, etc.).

SOPC demonstrates the flexibility of a soft-core processor. A custom system can be developed which contains only the hardware needed to perform a specific task.

---

Open Quartus II and select file -> New Project Wizard...

Specify a working directory for your project, and give your project a name. We haven’t defined a top-level design entity yet, so leave that field the same as the project name.

NOTE: Be sure your working directory contains no white-space. This will cause problems later when trying to compile your project from the SOPC Builder.
Here, a project called “hello_world” has been created, with working directory C:\hello_world.

The top-level design entity is also called hello_world.

Click “next” to enter the next dialogue box. We have no files to add to this project, so click “next” to move on.

We are now asked for a target device. You can enter this information later, but since we know which hardware is being targeted, we will do so now.

Select Cyclone II in the drop-down box labeled “family”. Use device EP2C20F484C7.

At this point, click finish. All necessary project parameters have been configured.
Tutorial: Using the SOPC Builder

- Now that a Quartus project has been setup, select File -> New...
- We will be using the SOPC Builder System, so select this option and click OK.
- The SOPC Builder should automatically open. Should the window close, you can access the builder via Tools -> SOPC Builder...
- You are prompted for an SOPC system name. SOPC generates VHDL or Verilog to describe the system. Select the language you are more comfortable programming in.
Tutorial: Using the SOPC Builder

- You should now see a window similar to:

![Window similar to SOPC Builder](image)

- Select DE1 as the target board.
- The menu on the left side of the Builder shows a variety of modules that can be included in the system. We will start by adding a module central to all NIOS based SOPC systems: The NIOS II Processor.
- Double-clicking on the device will open a dialogue box containing a number of different options for configuring your NIOS II Processor.
As you can see, there are 3 different varieties of the NIOS II, which provide a trade-off between FPGA resource usage and performance.

The Cyclone II is MUCH larger than the PLDs you are accustomed to working with. Our design will be very small, so we are not too concerned with resource usage.

Select the 2nd option (NIOS II/s). Don’t worry about the other settings. These allow you to change options such as instruction cache size and Debug support. Do not disable debug support, as it will be impossible to program the Nios!
Tutorial: Using the SOPC Builder

- You have now successfully added the first module to your SOPC system! You should see a brief description of the module in the main, grey area of the window.
- Next, we will be adding a JTAG UART module, found under communications. This provides us a way to communicate with the NIOS processor.
- Add this item by double-clicking (as before). Leave all options at their default values.

You are on the right track if your system looks something like this:
Tutorial: Using the SOPC Builder

- Next, we will be adding an interval timer to the system. This provides a system “heart-beat” which will handle many operations that happen outside of normal program-flow (interrupts, bus-arbitration, etc.)
- The interval timer can be found under “other”. Leave all of the values at their defaults and click “Finish”.
- There is a subtle issue in our system that needs to be changed...

Tutorial: Using the SOPC Builder

- Looking at the right-most column (labeled IRQ) of the modules you have instantiated, you will notice that the JTAG debug module has a lower IRQ than the interval-timer.
- Lower IRQ means higher priority. We want the interval timer to have a higher priority than the JTAG UART.
- Swap the two IRQ assignments to give the interval timer a higher priority.
Tutorial: Using the SOPC Builder

- Your system should now look something like this:

![SOPC Builder Interface](image)

- One more module is necessary to specify a bare-bones system to run a program on the NIOS processor: a program memory.
- The simplest solution is to provide an on-chip memory.
- On-chip memories can be found under "Memory". Leave all options at their default values, but do change the memory size to 20kB.
- Your system should now be ready to go! Click generate to create your first NIOS II system.
Tutorial: Programming the FPGA

- Now select Processing -> start -> Start Analysis & Synthesis. This will perform a quick check on the files produced by the SOPC Builder.
- This step is necessary for the pins associated with our NIOS system to show up for assignment.
- A number of warnings will appear during the check. This is normal.

Tutorial: Programming the FPGA

- At this point, we are ready to assign the inputs of our NIOS system to pins on the FPGA. Only two assignments are necessary for this design: reset and the 50 MHz clock.
- The DE1 comes with a .csv (comma separated value) file that can be used to automatically provide more intuitive names to the generic names pins have by default.
Tutorial: Programming the FPGA

- To import this file select Assignments -> Import Assignments...
- Browse to the directory mentioned earlier. Be sure the file type being displayed includes .csv.
- This may seem like an unnecessary step, but as systems get more complex, pin-placement will become non-trivial. Importing pin assignments will expedite the placement process and leave less room for error.
- IMPORTANT NOTE: Incorrect pin assignments could potentially damage your board!
- Now select Assignments -> Pins.

### You should see assignments similar to the screenshot found below:

<table>
<thead>
<tr>
<th>To</th>
<th>Location Y</th>
<th>IO Bank</th>
<th>[Y] Standard</th>
<th>General Function</th>
<th>Special Function</th>
<th>Reserved</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCE0</td>
<td>Y1000</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>PCE1</td>
<td>Y1001</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>PCE2</td>
<td>Y1002</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>PCE3</td>
<td>Y1003</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>PCE4</td>
<td>Y1004</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE0</td>
<td>Y1010</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE1</td>
<td>Y1011</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE2</td>
<td>Y1012</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE3</td>
<td>Y1013</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE4</td>
<td>Y1014</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE5</td>
<td>Y1015</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE6</td>
<td>Y1016</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE7</td>
<td>Y1017</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE8</td>
<td>Y1018</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE9</td>
<td>Y1019</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE10</td>
<td>Y1020</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE11</td>
<td>Y1021</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE12</td>
<td>Y1022</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE13</td>
<td>Y1023</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE14</td>
<td>Y1024</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
<tr>
<td>FCE15</td>
<td>Y1025</td>
<td>6</td>
<td>JVTIL</td>
<td>Row 00</td>
<td>(VSS 199)</td>
<td></td>
</tr>
</tbody>
</table>
## Tutorial: Programming the FPGA

- Clicking the leftmost column (To), will allow the entries to be alphabetized.
- Find the entry “CLOCK_50”, and replace it with our unassigned pin “clk”. Doing so will connect the clock found in our design to the 50 MHz oscillator on the DE1 board.
- Now assign the reset_n input signal to any of the SW[x] pins. This will connect the reset signal to one of the switches found on the DE1 board.
- Now select Processing -> Start Compilation. If compilation is successful, a programming file to be written to the FPGA will be generated.

---

## Tutorial: Programming the FPGA

- Select Tools -> Programmer. The generated programming file should automatically be selected. Be sure the DE1 is connected to the computer via the provided USB cable. Check the Program/Configure box to configure the Cyclone II FPGA as the NIOS system defined in the SOPC Builder.
- A dialogue box will open, mentioning we are using a time-limited version of the NIOS processor. Leave this box open. The FPGA has now been successfully configured!
Tutorial: Using the NIOS II IDE

- Now that you have successfully developed an SOPC system, we will write a very simple program to run on our new processor.
- Open the NIOS IDE. By default, this should be found under start -> All Programs -> Altera -> NIOS II 5.1 -> NIOS II IDE.
- Upon opening, NIOS II IDE will prompt you to select a workspace. A workspace maintains a history of different projects developed in the IDE. Go ahead and create a new workspace.

Tutorial: Using the NIOS II IDE

- A welcome window will open, but we don’t need it for this tutorial. Close the window.
- Select File -> New -> Project. A new dialogue box should open.
- Select C/C++ Application. Click next.
- A variety of project templates are provided as a starting point. We will be using the “Hello World” template.
- In order to develop a NIOS project, the IDE needs a .ptf (plain text file) file that indicates various peripherals unique to our design.
Tutorial: Using the NIOS II IDE

- If we had developed a multi-processor design, we have the option of selecting a CPU, but as our design only contains one, we may only select cpu_0.
- Now click finish. The compiler will generate various libraries necessary to interact with the hardware. These libraries are dependent upon the type of system we have defined in SOPC.

Tutorial: Using the NIOS II IDE

- You should now be looking at a window like this:
Before we compile this project a couple of optimizations are necessary.

Using the default libraries to compile our project will result in a memory footprint larger than the 20kB provided by our on-chip memory.

Under the “C/C++ Projects” tab, right-click hello_world_0. Select System Library Properties.

Uncheck “clean exit” and check “small C library”.

Click OK to update the library settings.

Under the “C/C++ Projects” tab, right-click hello_world_0. This time select Run As -> NIOS II Hardware.

This option will compile and write the program to the on-chip memory we specified in SOPC.
HELLO WORLD!

- If compilation was successful, and the hardware is properly connected, you should see a greetings message printed to the console from the DE1 board.