CSCE 313: Embedded Systems

Introduction

Instructor: Jason D. Bakos
Examples
What are Embedded Systems?

- A computer system designed for certain specific or dedicated functions, often for control and sensing
  1. Has constraints:
     - real-time processing
     - physical volume, power consumption, heat generation
     - limited memory and I/O devices
  2. Often is lightweight, compared to a desktop PC:
     - no OS or specialized OS
     - Software interfaces directly with hardware
  3. In modern systems, hardware and software are “codesigned”

- Examples:
  - smart phone, game console, microwave, elevator, cruise control system, ATM machine, factory equipment, iPad
System-on-a-Chip (SoC)

- Most embedded processors contain multiple CPUs and integrated peripherals on a single chip, such as:
  - GPUs, video interface
  - Interfaces, e.g. USB, Ethernet, SPI
  - Memory, e.g. ROM, RAM, Flash
  - Analog components: PLL, ADC, DAC
  - Counters, timers
  - Video/image decoding/encoding

- SoCs also include on-chip busses to interface the CPUs with peripherals
- Set of on-chip features represents a rich space of design tradeoffs for target application

- SoCs implemented as ICs are fixed at time of manufacture
System-on-a-Chip

Apple A5 chip
Field Programmable Gate Arrays

- Programmable logic device

- Contains:
  - Ability to implement “soft logic”: programmable logic gates (CLBs) with programmable interconnect
  - “hard cores”: RAMs, multiplier/adders, IOS, PCIe interface, etc.
Field Programmable Gate Arrays
Field Programmable Gate Arrays

- Originally developed for “glue logic”
  - Interface between board-level components
  - Example: PCI interface chip, embedded microprocessor, ethernet interface chip

- Now used as system-on a-programmable chip (SoPC)

- Idea:
  - Implement:
    - customized “softcore” processor,
    - memory/cache subsystem,
    - I/O interfaces,
    - off-chip memory interfaces
  - ...entirely on an FPGA
  - Only board-level components needed are:
    - analog devices (i.e. analog, VGA), connector receptacles, memory chips, power components (voltage regulators, capacitors), digital interface that have a faster clock than is possible on an FPGA (i.e. USB2 interface)
Sum-of-Products

- **Behavior:**
  - \( S = A + B \)
  - Assume A is 2 bits, B is 2 bits, C is 3 bits

<table>
<thead>
<tr>
<th>( A )</th>
<th>( B )</th>
<th>( C )</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 (0)</td>
<td>00 (0)</td>
<td>000 (0)</td>
</tr>
<tr>
<td>00 (0)</td>
<td>01 (1)</td>
<td>001 (1)</td>
</tr>
<tr>
<td>00 (0)</td>
<td>10 (2)</td>
<td>010 (2)</td>
</tr>
<tr>
<td>00 (0)</td>
<td>11 (3)</td>
<td>011 (3)</td>
</tr>
<tr>
<td>01 (1)</td>
<td>00 (0)</td>
<td>001 (1)</td>
</tr>
<tr>
<td>01 (1)</td>
<td>01 (1)</td>
<td>010 (2)</td>
</tr>
<tr>
<td>01 (1)</td>
<td>10 (2)</td>
<td>011 (3)</td>
</tr>
<tr>
<td>01 (1)</td>
<td>11 (3)</td>
<td>100 (4)</td>
</tr>
<tr>
<td>10 (2)</td>
<td>00 (0)</td>
<td>010 (2)</td>
</tr>
<tr>
<td>10 (2)</td>
<td>01 (1)</td>
<td>011 (3)</td>
</tr>
<tr>
<td>10 (2)</td>
<td>10 (2)</td>
<td>100 (4)</td>
</tr>
<tr>
<td>10 (2)</td>
<td>11 (3)</td>
<td>101 (5)</td>
</tr>
<tr>
<td>11 (3)</td>
<td>00 (0)</td>
<td>011 (3)</td>
</tr>
<tr>
<td>11 (3)</td>
<td>01 (1)</td>
<td>100 (4)</td>
</tr>
<tr>
<td>11 (3)</td>
<td>10 (2)</td>
<td>101 (5)</td>
</tr>
<tr>
<td>11 (3)</td>
<td>11 (3)</td>
<td>110 (6)</td>
</tr>
</tbody>
</table>

\[ C_2 = \overline{A_1} A_0 B_1 B_0 + A_1 \overline{A_0} B_1 B_0 + A_1 A_0 B_1 B_0 + \]
\[ A_1 A_0 B_1 B_0 + A_1 A_0 B_1 B_0 + A_1 A_0 B_1 B_0 \]

\[ C_1 = \overline{A_1} A_0 B_1 B_0 + \overline{A_1} A_0 B_1 B_0 + \overline{A_1} A_0 \overline{B_1} B_0 + \]
\[ A_1 A_0 B_1 B_0 + A_1 A_0 B_1 B_0 + A_1 A_0 B_1 B_0 \]

\[ A_1 A_0 B_1 B_0 + A_1 A_0 B_1 B_0 \]
FPGA Lookup Table

- Function generator:

\[
\begin{array}{c|c|c|c}
 x & y & z & xy + z' \\
\hline
 0 & 0 & 0 & 1 \\
 0 & 0 & 1 & 0 \\
 0 & 1 & 0 & 1 \\
 0 & 1 & 1 & 0 \\
 1 & 0 & 0 & 1 \\
 1 & 0 & 1 & 0 \\
 1 & 1 & 0 & 1 \\
 1 & 1 & 1 & 1 \\
\end{array}
\]

\[f\]
FPGA Fabric

- FPGA fabric:
Field Programmable Gate Arrays

- On chip resources:
  - Logic Elements (LEs)
    1. LUTS
    2. Registers
  - Onchip memories (M4Ks)
  - Multipliers
  - PLLs
Cyclone 2 Logic Element
Cyclone 2 Design

- Logic Array
- M4K Memory Blocks
- Embedded Multipliers
- Side I/O Elements with Support for PCI/PCI-X & Memory Interfaces
- Top & Bottom I/O Elements with Support for Memory Interfaces
- Phase-Locked Loops
Design of Large-Scale Digital Circuits

- Large-scale digital systems cannot be “directly” designed at the gate level by the designer.
Verilog Example

- Full adder:

  ```verilog
  module full_adder (input a,b,ci,output s,co);
      assign s = a ^ b ^ ci;
      assign cout = (a & b) | (a & ci) | (b & ci);
  endmodule
  ```

  - Synthesize: (Compile)

      | a | b | ci | s  | cout |
      |---|---|----|----|-----|
      | 0 | 0 | 0  | 0  | 0   |
      | 0 | 0 | 1  | 1  | 0   |
      | 0 | 1 | 0  | 1  | 0   |
      | 0 | 1 | 1  | 0  | 1   |
      | 1 | 0 | 0  | 1  | 0   |
      | 1 | 0 | 1  | 0  | 1   |
      | 1 | 1 | 0  | 0  | 1   |
      | 1 | 1 | 1  | 1  | 1   |
Mapping

- Assume our target FPGA has LUT2s
  - Can’t map an 3-input function to one LUT2...

- Diagram:
  - Three inputs: a, b, ci
  - One output: s
  - LUT3 encodes information about b, ci
  - Two more LUT2s with inputs b, ci and a, respectively, leading to output s

Encodes information about b, ci
Mapping

- \( s = a \oplus b \oplus c \)

- Equivalent to...
  \[
  s = (a)(\neg b)(\neg c) + (\neg a)(b)(\neg c) + (\neg a)(\neg b)(c) + (a)(b)(c)
  \]

- Transform:
  \[
  s = (\neg a)[(b)(\neg c) + (\neg b)(c)] + (a)[(\neg b)(\neg c) + (b)(c)]
  s = (\neg a)[(b)(\neg c) + (\neg b)(c)] + (a)[\neg[(b+c)(\neg b+\neg c)]]
  s = (\neg a)[(b)(\neg c) + (\neg b)(c)] + (a)[\neg[(b)(\neg c) + (\neg b)(c) + (c)(\neg c)]]
  s = (\neg a)[(b)(\neg c) + (\neg b)(c)] + (a)[\neg[(b)(\neg c) + (\neg b)(c)]]
  \]

- Set \( s0 = (b)(\neg c) + (\neg b)(c) \)
- \( s = (\neg a)(s0) + (a)(\neg s0) \)
### Verilog Example

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>ci</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>ci</th>
<th>s0</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>ci</th>
<th>s0</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>ci</th>
<th>s0</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
### Place and Route

#### Table 1: b, ci, s0

<table>
<thead>
<tr>
<th>b</th>
<th>ci</th>
<th>s0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Table 2: a, s0, s

<table>
<thead>
<tr>
<th>a</th>
<th>s0</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Terasic DE2

- DE2 board:
  - Altera Cyclone II FPGA with 70K gates
DE2 Board

- 9V DC Power Supply Connector
- 27-MHz Oscillator
- 24-bit Audio Codec
- Power ON/OFF Switch
- USB Host/Slave Controller
- TV Decoder (NTSC/PAL)
- Altera USB Blaster Controller Chipset
- Altera EPCS16 Configuration Device
- RUN/PROG Switch for JTAG/AS Modes
- 16x2 LCD Module
- 7-Segment Displays
- 18 Red LEDs
- 18 Toggle Switches
- 50-MHz Oscillator
- 8-MB SDRAM
- 512-KB SRAM
- 4-MB Flash Memory
- Ethernet 10/100M Controller
- Expansion Header 2 (JP2)
- Expansion Header 1 (JP1)
- Altera Cyclone II FPGA
- SD Card Slot
- 8 Green LEDs
- IrDA Transceiver
- SMA External Clock
- 4 Debounced Pushbutton Switches
- PS/2 Keyboard/Mouse Port
- VGA 10-bit DAC
- RS-232 Port
System Design

• Processors communicate with the outside world using a simple transactional model:

  – READ:
    • Processor says READ and provides an address
    • Operations that depend on this data WAIT until data is returned

  – WRITE:
    • Processor says WRITE and provides an address and data

  – These operations correspond to the LOAD and STORE instructions

  – In this case, we assume that CPU is the master and devices responding to these operations are slaves
Processor Interface

Processor

Clock
Reset

Instruction In

Instruction Address
Instruction Read

Data In

Data Address
Data Out
Data Read
Data Write

Instruction Interface

Data Interface
Processor Interface
Two Bus Model

Diagram:

- On-chip memory controller
- Processor
- Off-chip memory controller
- 10/100/1000 ethernet core

System bus

Bridge

Peripheral bus

USB
UART
I2C
Programmed I/O

- Loads and stores to specially-mapped address ranges can be used to:
  - Read a word from a status register
    - Used to poll the state of a peripheral
  - Write a word to a control register
    - Used to send an “instruction” to a peripheral
Sample Address Map
Altera Tools

• Quartus II
  – Starting point for all designs (create and open projects)
  – Contains simple editors for HDL design and constraint files
  – Has a makefile-like design flow manager for synthesis, map, place and route, bitstream generation, and programming

• SOPC Builder
  – Allows for drag-and-drop creations of platform designs (processors, busses, peripherals)

• NIOS2 IDE for Eclipse
  – Source code editor and BSP generator for Altera HAL
SOPC Builder and Eclipse Tools

- SOPC Builder allows you to design the portion of your embedded system that is implemented on the FPGA.

- Using this information, the Eclipse tools can generate a BSP that corresponds to your system.

- The BSP includes the drivers for the peripherals that you add in SOPC Builder.
  - As such, it must be regenerated each time you make a change in your system design.

- For each system you design, a unique system ID and timestamp is generated.

- The BSP’s ID and timestamp must match this:
  - This ensures a consistency between the system and the BSP.
SOPC Builder
Setting Up the Tools for Your Account

- We have the Quartus tools installed in a shared directory
- In order to run them you need to auto-execute a script each time you log in
- To do this, add the following line to your .profile:

  ```bash
  source /usr/local/3rdparty/cad_setup_files/altera.bash
  ```

- Once added, log in and back out
- You only need to do this once
Quartus

• Always begin by launching Quartus by opening a terminal and typing “quartus&” on the command line
• First time you’ll see:
Quartus
Creating a New Project

- File | New | New Quartus II Project...
- Working directory = /acct/s1/<username>/lights
- Project name = “lights”
- Top-level design entity = “lights”
- Skip to page 3
- For device, choose
  - **Family**: Cyclone II
  - **Package**: FBGA
  - **Pin count**: 672
  - **Speed grade**: 6
  - **Device**: EP2C35F672C6
- Click Finish

- Go to Tools | SOPC Builder
- System name = “nios_system”
- Target HDL = Verilog
SOPC Builder
Adding Components

- **Add a processor**
  - In the component library:
    - Processors | Nios II Processor
  - Select **Nios II/f**, then FINISH

- **Add an interface to the SDRAM on the DE2**
  - In the component library:
    - Memories and Memory Controllers | External Memory Interfaces | SDRAM Interfaces | SDRAM Controller
  - Presets=Custom, bits=16, chip select=1, banks=4, row=12, column=8, then FINISH

- **Add a clock manager**
  - In the component library:
    - University Program | Clocks Signals for DE-Series Board (DE2 board)
  - Uncheck Video and Audio (leave SDRAM checked), then FINISH
  - In Clock Settings, rename (double-click the mouse):
    - clocks_0_sys_clk => sys_clk
    - clocks_0_sdram_clk => sdram_clk
  - In the system configuration pane:
    - Change the clock for the cpu to sys_clk
    - Change the clock for the sdram to sdram_clk
    - Leave the clock for clocks_0 as “clk_0”
Adding Components
Adding Components

• Add a system ID peripheral
  – In the component library:
    • Peripherals | Debug and Performance | System ID Peripheral
  – FINISH, then rename (right-click the mouse) it “sysid” and put it on the sys_clk

• Add a JTAG UART for the console
  – In the component library:
    • Interface Protocols | Serial | JTAG UART
  – FINISH, then put it on the sys_clk

• Add parallel I/O for the LEDs
  – In the component library:
    • Peripherals | Microcontroller Peripherals | PIO (Parallel I/O)
  – Width=26, output ports only, FINISH, rename it as “leds”, then put it on the sys_clk

• Add parallel I/O for the keys (buttons)
  – Same as above, but 3 bits, input only
  – Under “Input Options”, turn on “Synchronously capture”, then FINISH
  – Rename as “keys” and put it on the sys_clk
Adding Components
Adding Components

• Add the interface for the LCD
  – In the component library:
    • Peripherals | Display | Character LCD
  – Put it on sys_clk
• Done adding components! **Save as nios_system.sopc**

• Double-click cpu_0
  – Select sdram_0 for the reset and exception vectors and click FINISH
• Click System | Auto Assign Base Addresses
• File | Save
• Click the GENERATE button
• Go to the “System Generation” tab and click “Nios II Software Build Tools for Eclipse”
Nios II IDE

Usage Data Upload

It's time to upload your usage data.

The Eclipse Usage Data Collector (UDC) collects data on how you have been using the workbench. It would now like to upload the data to a server at the Eclipse Foundation. No data is sent unless you agree.

You can preview the data before it is uploaded on the Preview page.

Questions about the UDC? Check out our Frequently Asked Questions.

- Upload now
  - Upload the usage data now. Ask before uploading again.
- Upload always
  - Upload the usage data now. Don't ask next time; just do the upload in the background. Note that you can change this setting in the preferences.
- Don't upload now
  - Do not upload usage data at this time. You will be asked to do the upload later.
- Turn UDC feature off
  - Stop collecting data. The UDC will be turned off and data will never be uploaded.

You agree to provide this data under the Usage Data Collector Terms of Use.
Nios II IDE

- File | New | Nios II Application and BSP from Template

- Browse for your SOPC information file name
  - This is generated the first time you GENERATE the system

- Project name = lights

- Select “Hello World”
- FINISH
Eclipse Tools
Eclipse Tools

- Right-click on lights_bsp and select Nios II | BSP Editor...
- Change stderr to lcd_0
- Click Generate
- Click Exit
Eclipse Tools

- Double-click on hello_world.c
Eclipse Tools

Any time you make a change in the system, you must re-generate the BSP (do this now):

- Right-click “lights_bsp” | Nios II | Generate BSP
- Right-click “lights_bsp” | Build Project

Under BSP...

- system.h contains definitions for your system
- The header files under /drivers contain information about how to interface with the hardware you added to your system
Software

- Open hello_world.c
- Add header files:
  
  ```c
  #include <stdio.h>
  #include <unistd.h>
  #include "system.h"
  #include "altera_avalon_pio_regs.h"
  #include "alt_types.h"
  ```
Software

- New main () code:

```c
alt_u32 current_value;
alt_u32 current_state;
alt_u8 current_direction;
alt_u32 keys;

current_state=3;
current_value=1;
current_direction=0;

printf ("Program running (UART)...\n");
fprintf (stderr,"Program running (LCD)...\n");
```
while (1) {
    // read the current state of the keys
    keys=IORD_ALTERA_AVALON_PIO_DATA(KEYS_BASE);
    // switch speed if necessary
    if ((keys != 7) && (keys != current_state)) {
        if (keys == 3) printf("speed set to 250 ms\n");
        else if (keys == 5) printf("speed set to 150 ms\n");
        else if (keys == 6) printf("speed set to 50 ms\n");
        current_state=keys;
    }
    // switch direction if necessary
    if (((current_direction==0) && (current_value==(1 << 25))) current_direction=1;
    else if (((current_direction==1) && (current_value==1)) current_direction=0;
    // move light
    else if (current_direction==0) current_value = current_value << 1;
    else current_value = current_value >> 1;
    // update lights
    IOWR_ALTERA_AVALON_PIO_DATA(LEDS_BASE,current_value);
    // wait
    if (current_state==3) usleep (250000);
    else if (current_state==5) usleep (125000);
    else usleep (50000); }
Error Messages

• “Cannot find ELF”
  – Check error log for compilation
Whenever you make a change to your system design in SOPC Builder, you must follow these steps in order:

1. **SOPC Builder:** Regenerate HDL (GENERATE button)
2. **Quartus:** Modify VHDL, re-compile HDL
3. **Quartus:** Program FPGA
4. **Eclipse:** Regenerate BSP, clean BSP, restart Eclipse(?)
5. **Eclipse:** Re-build project (run)
6. **Eclipse:** Execute ELF
Quartus

- Back to Quartus...
- Now we need to write a top-level Verilog HDL file for lights
- File | New | Verilog HDL File
Top-Level Design

FPGA “lights”

nios_system (“NiosII”)
Top-Level Design

- File | New | Verilog HDL File
- Save as lights.v

module lights (  
  // 50 MHz clock
  input          CLOCK_50,

  // 4 blue buttons
  input   [3:0]  KEY,

  // 18 black switches
  input   [17:0]  SW,
Top-Level Design

// 8 7-segment LEDs
output [6:0]    HEX0,
output [6:0]    HEX1,
output [6:0]    HEX2,
output [6:0]    HEX3,
output [6:0]    HEX4,
output [6:0]    HEX5,
output [6:0]    HEX6,
output [6:0]    HEX7,

// 9 green LEDs
output [8:0]    LEDG,

// 18 red LEDs
output [17:0]   LEDR,
Top-Level Design

// DRAM interface signals
inout [15:0] DRAM_DQ,
output [11:0] DRAM_ADDR,
output DRAM_LDQM,
output DRAM_UDQM,
output DRAM_WE_N,
output DRAM_CAS_N,
output DRAM_RAS_N,
output DRAM_CS_N,
output DRAM_BA_0,
output DRAM_BA_1,
output DRAM_CLK,
output DRAM_CKE,
Top-Level Design

// LCD interface signals
inout [7:0] LCD_DATA,
output LCD_ON,
output LCD_BLON,
output LCD_RW,
output LCD_EN,
output LCD_RS;
Hardware

wire clk_0;
wire [ 2: 0] in_port_to_the_keys;
wire [ 25: 0] out_port_from_the_leds;
wire reset_n;
wire sdram_clk;
wire sys_clk;

assign HEX0 = 7'h00;
assign HEX1 = 7'h00;
assign HEX2 = 7'h00;
assign HEX3 = 7'h00;
assign HEX4 = 7'h00;
assign HEX5 = 7'h00;
assign HEX6 = 7'h00;
assign HEX7 = 7'h00;

assign LCD_ON = 1'b1;
assign LCD_BLON = 1'b1;
assign LEDR = out_port_from_the_leds[25 : 8];
assign LEDG = out_port_from_the_leds[ 7 : 0];
assign DRAM_CLK = sdram_clk;

assign clk_0 = CLOCK_50;
assign reset_n = KEY[0];
assign in_port_to_the_keys = KEY[3:1];
Hardware

//Set us up the DUT
nios_system DUT
{
  .LCD_E_from_the_lcd_0           (LCD_EN),
  .LCD_RS_from_the_lcd_0          (LCD_RS),
  .LCD_RW_from_the_lcd_0          (LCD_RW),
  .LCD_data_to_and_from_the_lcd_0 (LCD_DATA),
  .clk_0                          (clk_0),
  .in_port_to_the_keys           (in_port_to_the_keys),
  .out_port_from_the_leds        (out_port_from_the_leds),
  .reset_n                        (reset_n),
  .sdram_clk                      (sdram_clk),
  .sys_clk                         (sys_clk),
  .zs_addr_from_the_sdram_0       (DRAM_ADDR),
  .zs_ba_from_the_sdram_0         (DRAM_BA_0),
  .zs_cas_n_from_the_sdram_0      (DRAM_CAS_N),
  .zs_cke_from_the_sdram_0        (DRAM_CKE),
  .zs_cs_n_from_the_sdram_0       (DRAM_CS_N),
  .zs_dq_to_and_from_the_sdram_0  (DRAM_DQ),
  .zs_dqm_from_the_sdram_0        (DRAM_LDQM),
  .zs_ras_n_from_the_sdram_0      (DRAM_RAS_N),
  .zs_we_n_from_the_sdram_0       (DRAM_WE_N)
};
endmodule
• Double-click on Compile Design
Hardware

• Assignments | Import Assignments

• Select:
  – /usr/local/3rdparty/csce611/CPU_support_files/DE2_pin_assignments.csv

• Go to Assignments | Pin Planner to check your pin assignments
Hardware

- Re-compile the design...
- Program the FPGA...
  - Double-click on Program Device
Hardware

- Click Start
Nios II Tools

• Back to the Nios II Tools...

• Let’s run the software

• You can debug using the debug button
  – Set breakpoints
  – Inspect data
  – Step (into, over, out)
Nios II Debug Environment