# CSEE W4840 Embedded System Design Lab 3

## Stephen A. Edwards

## Due March 3, 2011

#### Abstract

Use Quartus and SOPC builder to create one of two mixed hardware/software designs: an FM sound synthesizer or a bouncing video ball.

### 1 Introduction

This lab is about combining your own hardware and software components. You have a choice of implementing one of two "canned" designs that we started for you: an FM sound synthesizer that generates pleasing-sounding notes under keyboard control or a bouncing video ball in which software controls the trajectory of a circle on the screen displayed by custom video hardware.

First, follow the instructions in Section 2 to gain some practice building a simple system using SOPC Builder. Then, choose one of the two projects described in Sections 3 and 4.

#### 2 Building a Nios II System with SOPC Builder

SOPC Builder is an Altera-supplied program for quickly assembling Nios II-based processor systems. It effectively writes VHDL for you.

The tutorial below explains how to make a simple "bouncing ball" LED display using SOPC Builder. Go though this tutorial first to see how the tools work, then start working on one of the three designs.

#### 2.1 Quartus, part 1

Create a new directory (e.g., "lab3"), *cd* into it, and start *quartus*.

Select File→New Project Wizard.

In the new project wizard dialog, select the directory (e.g., "lab3") you just created. Name the project something like "lab3." The two names do not have to match, but only use letters, digits, and underscores in the project name. See Figure 1.

Don't add any files to the project yet.

For for the device, select the "Cyclone II" family and the "EP2C35F672C6" chip. See Figure 2.

Click "Finish" to create the project.

#### 2.2 SOPC Builder

Inside Quartus, select Tools $\rightarrow$ SOPC Builder. This will probably ask you to start creating an SOPC builder system (if not, select File $\rightarrow$ New System). Name it differently than the project, e.g., "nios\_system," and select VHDL as the language. See Figure 3.

You should now be at the SOPC Builder main window (Figure 4). Make sure the Device Family is set to Cyclone II and that there is a single external 50 MHz clock listed.

| New Project Wizard: Directory, Name, Top-Level Entity [page 1   | of 5]                        | × |
|---|------------------------------|---|
|   |                              |   |
| What is the working directory for this project?   |                              |   |
| /home/user1/faculty/sedwards/4840-2008/lab3   |                              |   |
| What is the name of this project?   |                              |   |
| lab3  |                              |   |
| What is the name of the top-level design entity for this project? This nam<br>exactly match the entity name in the design file. | e is case sensitive and must |   |
| lab3  |                              |   |
|   |                              |   |
| Use Existing Project Settings   |                              |   |
|   |                              |   |
|   |                              |   |
|   |                              |   |
|   |                              |   |
|   |                              |   |
|   |                              |   |
|   |                              |   |
|   |                              |   |
|   |                              |   |
|   |                              |   |
|   |                              |   |
|   |                              |   |
| < Back Next >   | Finish Cancel                |   |

Figure 1: Naming a new Quartus project

| mily: Cyclone II<br>Target device<br>C Auto device select | Show in 'Available device' list<br>Package: Any<br>Pin <u>c</u> ount: Any |         |         |        |            |           |          |
|---|---|---------|---------|--------|------------|-----------|----------|
| Specific device sel                                       | Speed grad  | de: Any |         | -      |            |           |          |
|   |   |         |         | Show a | advanced o | levices   |          |
|   |   |         |         | HardCo | py compai  | ible only |          |
|   |   |         |         |        |            |           |          |
| ailable <u>d</u> evices:                                  |   |         |         |        |            |           |          |
| Name  | Core v  | LEs     | User I/ | Memor  | Embed      | PLL       |          |
| P2C20Q240C8   | 1.2V  | 18752   | 142     | 239616 | 52         | 4         |          |
| P2C35F484C6   | 1.2V  | 33216   | 322     | 483840 | 70         | 4         |          |
| P2C35F484C7   | 1.2V  | 33216   | 322     | 483840 | 70         | 4         |          |
| P2C35F484C8   | 1.2V  | 33216   | 322     | 483840 | 70         | 4         |          |
| P2C35F484I8   | 1.2V  | 33216   | 322     | 483840 | 70         | 4         |          |
| P2C35F672C6   | 1.2V  | 33216   | 475     | 483840 | 70         | 4         |          |
| P2C35F672C7   | 1.2V  | 33216   | 475     | 483840 | 70         | 4         |          |
| P2C35F672C8   | 1.2V  | 33216   | 475     | 483840 | 70         | 4         |          |
| P2C35E67218   | 1.20  | 32216   | 475     | 183810 | 70         | A         | ٠Ē       |
|   |   |         |         |        |            | <u></u>   | <u> </u> |
| Companion device  |   |         |         |        |            |           |          |
| HardCopy II:  |   |         | -       |        |            |           |          |
|   |   |         |         |        |            |           |          |

Figure 2: Selecting the device in Quartus

| Create New System        | : |
|--------------------------|---|
| System Name: nios_system | Ī |
| Target HDL: 🔘 Verilog    |   |
| VHDL                     |   |
|                          | ĺ |
| OK Cancel                | 1 |

Figure 3: Naming a new system in SOPC Builder

| Ile Edit Module Syste<br>System Contents Syst<br>Altera SOPC Builder<br>Ile Create new con<br>- Nios Il Processol<br>- Bridges and Adaptet<br>- Interface Protocols<br>- Legacy Components<br>- Memories and Memories | m ⊻iew <u>T</u> o<br>em Generation<br>Target<br>Device Family | 1           | Clock Settings<br>Name | External | Source | 50.0      | MHz    | Add |
|---|---|-------------|------------------------|----------|--------|-----------|--------|-----|
| Peripherals     PLL     User Logic  | Use Con   | Module Name | Des                    | cription | Clock  | Base      | End    | IRG |
|   |   |             |                        |          |        |           |        |     |
| Add   |   | Remove Edit | . Move Lis             | Wove Dov | vn Ada | dress Map | Ejiter |     |
|   | dy to generate.   | Remove      | . Move Ly              | Move Dov | vn Adk | dress Map | Ejter  |     |

Figure 4: The SOPC Builder main window. Available components are listed on the left.

Add the processor by opening Avalon Components and double-clicking "Nios II Processor—Altera Corporation." This should bring up the Nios II dialog in Figure 5. Select the Nios II/e, the smallest of the three and click "Finish." You don't need to adjust the other parameters.

At this point (Figure 6), you have a single processor with a JTAG debug module connected to it. By itself, this is useless because it has no memory.

We will use the off-chip 512K SRAM by creating a new component (peripheral) that does the nearly-trivial translation from the protocol spoken by the Avalon bus (i.e., that is connected to the Nios II) to that for the SRAM.

First, you need a VHDL file for the component called de2\_sram\_controller.vhd. Its contents are shown in Figure 7.

This does almost nothing: it connects and inverts the various Avalon signals (named avs\_s1\_...) for the SRAM chip and controls the tri-state output drivers by indicating the SRAM\_DQ bus should only be driven when the Avalon *write* signal is asserted.

Create a new SOPC Builder component by selecting File $\rightarrow$ New Component. Under HDL Files, select this .vhd file. A dialog will come up showing the file is being parsed and give you a bunch of warnings about signals having type "export," which is fine. Make sure the Top Level Module is set to "de2\_sram\_controller."

Go to the "Signals" tab and change all the interfaces for the SRAM signals to "export\_0." The list should look like Figure 8.

Next, go to the "Interfaces" tab and click on "Remove Interfaces With No Signals." You should leave the interfaces in the state shown in Figure 9. Here, "Slave addressing" is an important choice. The "DYNAMIC" setting indicates that the bus will be dynamically resized to accomodate the data width of the

|                                 |                                | Nios II Processor - c                  | pu                                      |                 |
|---------------------------------|--------------------------------|--|---|-----------------|
| Nio<br>Megatore                 | s II Processor                 |  |   | Documentation   |
| 1 Parameter<br>Settings         |                                |  |   |                 |
|                                 | thes and Memory Interfaces     | Advanced Features JTAG D               | vebug Module Custom Instruction         | ne >            |
| Core Nios II                    |                                | /                                      |   | ~               |
| Select a Nios II core           |                                |  |   |                 |
| Select a Mos II cole            |                                |  |   | 1               |
|                                 | Nios II/e                      | ONios II/s                             | ONios II/f                              | J               |
| Nios II                         | RISC<br>32-bit                 | RISC<br>32-bit                         | RISC<br>32-bit                          |                 |
| Selector Guide                  | 02 DA                          | Instruction Cache                      | Instruction Cache                       |                 |
| Family: Cyclone II              |                                | Branch Prediction<br>Hardware Multiply | Branch Prediction<br>Hardware Multiply  |                 |
| f <sub>system:</sub> 50.0 MHz   |                                | Hardware Divide                        | Hardware Divide                         |                 |
| cpuid: 0                        |                                |  | Barrel Shifter                          |                 |
|                                 |                                |  | Data Cache<br>Dynamic Branch Prediction |                 |
| Performance at 50.0 N           | MHz Up to 5 DMIPS              | Up to 25 DMIPS                         | Up to 51 DMIPS                          |                 |
| Logic Usage                     | 600-700 LEs                    | 1200-1400 LEs                          | 1400-1800 LEs                           |                 |
| Memory Usage                    | Two M4Ks (or equiv.)           | Two M4Ks + cache                       | Three M4Ks + cache                      |                 |
| Hardware Multiply: 🗔            |                                |  |   |                 |
| Filler Contractor Contractory . | nbedded Multipliers            | Hardware Divide                        |   |                 |
|                                 |                                |  |   |                 |
| Reset Vector: N                 | femory:                        | ■ Offset: 0x0                          | )                                       |                 |
| Exception Vector: Me            | emory:                         |  | 0                                       |                 |
|                                 |                                |  |   |                 |
|                                 |                                |  |   |                 |
| 1) Warning: Reset ve            | ctor and Exception vector cann | ot be set until memory devices         | are connected to the Nios II proces     | sor             |
|                                 |                                |  |   |                 |
|                                 |                                |  |   |                 |
|                                 |                                |  |   |                 |
|                                 |                                |  | Cancel < Ba                             | ck Next > Einis |
|                                 |                                |  |   |                 |

Figure 5: Adding an Nios II processor in SOPC Builder

| System Contents Syst<br>Attern SOPC Builder<br>Nos Increase new con<br>Bridges and Adapter<br>Interface Protocols<br>Cegacy Components<br>Memory Components<br>Memory Components | m View Ioole Nios II Help<br>em Generation<br>Target<br>Device Family: Cyclone II | Clock Settings                   | Source<br>External | MHz<br>50.0                              | Add<br>Remove |
|--|---|----------------------------------|--------------------|--|---------------|
| Perpherals     PLL     Our Depresentation     Video and Image Pro     Video and Image Pro  | Use Connec. Module Name   | Avalon Master<br>de Avalon Steve | cik                | Base Er<br>IRQ 0<br>⊯ 0x00000800 0x00000 | IRQ 31↔×      |
| Add  | Remove Edit   | Move Up                          | Move Down          | dress Map Filter                         |               |

Figure 6: The system with only the Nios II processor

peripheral—exactly what we want for the SRAM component. The "NATIVE" setting disables this: the bus always appears as 32 bits wide and the peripheral is expected to align its data on 32 bit boundaries.

Click on "Finish" and save your component.

Edit the "de2\_sram\_controller\_hw.tcl" that this just created and change the "isMemoryDevice" property from "false" to "true."

Return to the main SOPC builder window, select the new "de2\_sram\_controller" component in the left pane, and click on "Add..." and then "Finish." Right-click on the module name (it defaults to "de2\_sram\_controller\_inst") and rename it to "sram." Move the mouse into the "Connec..." section of the components in the system and click on the white circle at the intersection of the line from "instruction\_master" under the CPU to "avalon\_slave\_0" of the SRAM. This allows the CPU to store programs in the SRAM, not just data.

Congratulations: your processor system now has some memory and could actually run programs.

If you later change the VHDL code for your component (e.g., during the development process), you must re-edit the compo-

```
use ieee.std_logic_1164.all;
entity de2_sram_controller is
  port (
    signal chipselect : in std_logic;
    signal write, read : in std_logic;
    signal address : in std_logic_vector(17 downto 0);
    signal readdata : out std_logic_vector(15 downto 0);
    signal writedata : in std_logic_vector(15 downto 0);
    signal byteenable : in std_logic_vector(1 downto 0);
    signal SRAM_DQ : inout std_logic_vector(15 downto 0);
    signal SRAM_ADDR : out std_logic_vector(17 downto 0);
    signal SRAM_UB_N, SRAM_LB_N : out std_logic;
    signal SRAM_WE_N, SRAM_CE_N : out std_logic;
    signal SRAM_OE_N
                                  : out std_logic
    );
end de2 sram controller:
architecture dp of de2_sram_controller is
begin
  SRAM_DQ <= writedata when write = '1'</pre>
             else (others => 'Z');
  readdata <= SRAM_DQ;</pre>
  SRAM_ADDR <= address;</pre>
  SRAM_UB_N <= not byteenable(1);</pre>
  SRAM_LB_N <= not byteenable(0);</pre>
  SRAM_WE_N <= not write;</pre>
  SRAM_CE_N <= not chipselect;</pre>
```

| About Signals      |                                  |              |                       | 107.00     | - PC - P |
|--------------------|----------------------------------|--------------|-----------------------|------------|----------|
| Name<br>Chipselect | Interface                        | - 1.1-       | Signal Type<br>select | 1 Vvidth   | Directio |
| write              | avalon_slave_0<br>avalon_slave_0 | cnip         |                       | 1          | input    |
| read               | avalon slave 0                   | read         | -                     | 1          | input    |
| address            | avalon slave 0                   | 1            | ress                  | 18         | input    |
| 🖉 readdata         | avalon slave 0                   |              | idata                 | 16         | output   |
| writedata          | avaion slave 0                   |              | edata                 | 16         | input    |
| byteenable         | avalon slave 0                   |              | enable                | 2          | input    |
| SRAM DQ            | export_0                         | exp          |                       | 16         | bidir    |
| SRAM ADDR          | export 0                         | exp          |                       | 18         | output   |
| SRAM_UB_N          | export_0                         | exp          |                       | 1          | output   |
| SRAM LB N          | export 0                         |              | export                |            | output   |
| SRAM WE N          | export 0                         |              | export                |            | output   |
| SRAM_CE_N          | export_0                         | exp          |                       | 1          | output   |
| W OD HIL OF H      | export 0                         | ▼ exp        | ort                   | <b>v</b> 1 | output   |
| ∑SRAM_OE_N         |                                  |              |                       |            |          |
| SRAM_OE_N          | Add Signal                       | Remove Signa | 1                     |            |          |
|                    |                                  |              |                       |            |          |

Figure 8: Associating the signals with interfaces

| Introduction    | HDL Files                    | Signals       | Interfaces | Component Wizard         |               |
|-----------------|------------------------------|---------------|------------|--------------------------|---------------|
| About Inter     | faces                        |               |            |                          |               |
|                 |                              |               |            |                          |               |
| ✓ "export_      | 0" (Conduit C                | utput) —      |            |                          |               |
| ٢               | lame: export                 | _0            |            |                          |               |
|                 | Type: Condu                  | it Output     |            | -                        |               |
| Associated (    | lock: none                   |               |            | -                        |               |
|                 | slave O"(A)                  | alon Slave) - |            |                          |               |
| _               |                              |               |            |                          |               |
|                 | lame: avalon<br>Type: Avalor |               |            | <b>•</b>                 |               |
| Associated (    |                              | r Slave       |            | <b>•</b>                 |               |
| Associated (    | JOCK. NONE                   |               |            |                          |               |
| -▼ Avalon S     | lave Settings                |               |            |                          |               |
| Cla             | /e Addressin                 |               |            |                          |               |
| Minimum Arb     |                              | -             |            |                          |               |
|                 | e stdout/stder               |               |            |                          |               |
|                 | erleave Burst                |               |            |                          |               |
|                 |                              |               |            |                          |               |
| -▼ Avalon S     | lave Timing –                |               |            |                          |               |
|                 | Setup 0                      |               | VVait 1    | Hold 0 Ur                | nits Cycles 🗸 |
|                 |                              | Write         | VVait 0    |                          |               |
| Pipelined Tr    |                              |               |            |                          |               |
|                 | Rea                          | ad Latency    | ) M        | ax Pending Read Transa   | actions 0     |
|                 |                              |               |            |                          |               |
|                 | [                            | Add Interf    | ace F      | Remove Interfaces With I | No Signals    |
|                 |                              |               |            |                          |               |
| ) Info: No erro | rs or warning                | js.           |            |                          |               |
|                 |                              |               |            |                          |               |
|                 |                              |               |            |                          |               |
|                 |                              |               |            |                          |               |
|                 |                              |               |            |                          |               |

Figure 9: Setting the interfaces

end dp;

SRAM\_OE\_N <= not read;</pre>

library ieee;

Figure 7: de2\_sram\_controller.vhd: VHDL source for the SRAM controller (inverters and a tristate buffer).

nent by right-clicking the component on the left menu and selecting "Edit."

Double-click on the cpu component and choose the "sram" memory for both the reset vector and the exception vector. This should turn off some warnings. If you can't select sram as the memory, you probably forgot to change the "isMemoryDevice" setting in "de2\_sram\_controller\_hw.tcl."

Using the same procedure, create a new component called "de2\_led\_flasher." The VHDL for this is shown in Figure 11. Again, remember to change the interface of the "leds" signal to "export\_1". Connect the "clk" and "reset\_n" signals the "clock" interface and set their types to "clk" and "reset\_n" respectively. The signals tab should look like Figure 10.

Add an instance of your new "led\_flasher" component to the system and rename it to "leds."

For debugging output, add a Interface Protocols/Serial/JTAG UART component. Just click "Finish" to accept the default parameters.

Run System $\rightarrow$ Auto-Assign Base Addresses to locate each component in memory. The completed system configuration is shown in Figure 12.

Finally, click on the "System Generation" tab, make sure "Simulation. Create simulator project files" is disabled (simulation with the DE2 does not work well without models for the various off-chip peripherals) and click "Generate." This should

| ntroduction   HDL File  | s Signals Interfaces | Component Wizard |             |       |           |
|-------------------------|----------------------|------------------|-------------|-------|-----------|
| About Signals           |                      |                  |             |       |           |
| Name                    | Interface            |                  | Signal Type | Width | Direction |
| clk                     | clock                | 🔻 cik            |             | 1     | input     |
| reset_n                 | clock                | reset            | _n          | 1     | input     |
| read                    | avalon_slave_0       | read             |             | 1     | input     |
| write                   | avalon_slave_0       | write            |             | 1     | input     |
| chipselect              | avalon_slave_0       | chips            | elect       | 1     | input     |
| address                 | avalon_slave_0       | addre            | 88          | 5     | input     |
| readdata                | avalon_slave_0       | readd            | ata         | 16    | output    |
| writedata               | avalon_slave_0       | write            | data        | 16    | input     |
| leds                    | export_0             | expor            | t           | 16    | output    |
|                         |                      |                  |             |       |           |
|                         |                      |                  |             |       |           |
|                         | Add Signal           | Remove Signal    | _           |       |           |
| ) Info: No errors or wa |                      |                  |             |       |           |

Figure 10: Signals for the LED flasher

fill your project directory with many .vhd files.

When system generation completes (this takes a while), click on Exit and return to the Quartus II GUI.

#### 2.3 Quartus, part 2

Once SOPC Builder has generated the system, we need to import it into a Quartus II project.

First, you need to create a top-level VHDL file that instantiates the Nios II system that was just generated and whatever hardware you want to connect to it. In this case, we only need to wire the Nios II to the external clock and connect the SRAM and LEDs to their pins.

The nios\_system entity was generated by the SOPC Builder and is defined in nios\_system.vhd (along with a lot of other things). As usual, its component definition is essentially just the ports on the entity, which were named by SOPC Builder.

Figure 13 shows the top-level VHDL file. Put this in the project directory and add it to the Quartus project. Also add the relevant .vhd files that were generated by SOPC builder: cpu\_jtag\_debug\_module.vhd, cpu.vhd, jtag\_uart.vhd, leds.vhd, sram.vhd, and nios\_system.vhd. Make sure you put "lab3.vhd" *below* the others (it won't find the nios\_system entity otherwise).

By default, the name of the top-level entity is the name of the project2. Open lab3.vhd and use  $Project \rightarrow Set$  as Top-Level Entity to change this.

Match the pin names to locations by selecting Assignments→Import Assignments and choosing the DE2\_pin\_assignments.csv file.

Select an overall default frequency requirement, then set Default fmax to 50 MHz (Figure 14). Leave the defaults alone on the next window, then click Finish.

Compile the project and download it to the board. Congratulations! You just built a computer. library ieee; use ieee.std\_logic\_1164.all; use ieee.numeric\_std.all;

entity de2\_led\_flasher is

```
port (
  clk
             : in std_logic;
  reset_n
             : in
                   std_logic;
                  std_logic;
  read
             : in
                   std_logic;
  write
             : in
  chipselect : in
                  std_logic;
             : in unsigned(4 downto 0);
  address
  readdata
             : out unsigned(15 downto 0);
  writedata : in unsigned(15 downto 0);
             : out unsigned(15 downto 0)
  leds
  );
```

end de2\_led\_flasher;

architecture rtl of de2\_led\_flasher is

```
type ram_type is array(15 downto 0) of unsigned(15 downto 0);
signal RAM : ram_type;
signal ram_address, display_address : unsigned(3 downto 0);
signal counter_delay : unsigned(15 downto 0);
signal counter : unsigned(31 downto 0);
```

begin

```
ram_address <= address(3 downto 0);</pre>
```

#### 

```
readdata <= RAM(to_integer(ram_address));</pre>
    elsif write = '1' then
      RAM(to_integer(ram_address)) <= writedata;</pre>
    end if:
  else
    if write = '1' then
      counter_delay <= writedata;</pre>
    end if;
  end if;
else
  leds <= RAM(to_integer(display_address));</pre>
  if counter = x"00000000" then
    counter <= counter_delay & x"0000";</pre>
    display_address <= display_address + 1;</pre>
  else
    counter <= counter - 1;
  end if:
```

```
end if;
end process;
```

end if:

end if;

```
end rtl;
```

Figure 11: led\_flasher.vhd: VHDL source for the LED flash controller. This memory-maps a  $16 \times 16$  RAM into 16 halfwords and a single "delay" register into another 16. When the RAM is not being written, a counter steps through the contents of the RAM, displaying it on the LEDs. The delay register sets the hold time for each address.

|  | ation                     |               |                       |                                 |           |         |           |                        |                    |
|--|---------------------------|---------------|-----------------------|---------------------------------|-----------|---------|-----------|------------------------|--------------------|
| Altera SOPC Builder  | Targ                      | et            | r                     | Clock Settings                  |           |         |           |                        |                    |
| – 🔯 Create new component   | Device Family: Cyclone II |               |                       | Name                            |           | Source  |           | Hz                     | Add                |
| <ul> <li>de2_led_flasher</li> </ul>                                    | Demo                      | or anny. C    | Joonon •              | clk                             | External  |         | 58.8      |                        | Add                |
| <ul> <li>de2_sram_controller</li> </ul>                                |                           |               |                       |                                 |           |         |           |                        | Remove             |
| <ul> <li>Nios II Processor</li> <li>Bridges and Adapters</li> </ul>    |                           |               |                       |                                 |           |         |           |                        |                    |
| <ul> <li>Interface Protocols</li> </ul>                                |                           |               |                       |                                 |           |         |           |                        |                    |
| Legacy Components  | Use                       | Connec        | Module Name           | Description                     | 0         | lock    | Base      | End                    | IRQ                |
| <ul> <li>Memories and Memory Controlle</li> <li>Peripherals</li> </ul> | V                         |               | 🖻 cpu                 | Nios II Processor               |           |         |           |                        |                    |
| Periprierais   |                           |               | instruction_master    | Avalon Master                   | clk       |         |           |                        |                    |
| - User Logic   |                           |               | data_master           | Avalon Master                   |           |         | IRQ O     |                        | ) 31 <del>(~</del> |
| Video and Image Processing   |                           | $\rightarrow$ | jtag_debug_module     |                                 |           | iii 6   | x00100800 | 0x00100fff             |                    |
|  | ×                         |               | 🖻 sram                | de2_sram_controller             |           |         |           |                        |                    |
|  |                           | $\rightarrow$ | avalon_slave_0        | Avalon Slave                    |           | #° 0    | x00080000 | 0×000 <del>11111</del> |                    |
|  | M                         |               | eds<br>avalon_stave_0 | de2_led_flasher<br>Avalon Slave | clk       |         |           | 0x0010103f             |                    |
|  | V                         | -             | avaion_siave_0        | JTAG LIART                      | CIK       |         | X00101000 | 0x00101031             |                    |
|  |                           |               | avaion tag slave      |                                 | clk       | ~ 0     | 00101040  | 0x00101047             | -6                 |
| I I I  |                           |               | aread Grag Source     |                                 |           |         |           |                        | re                 |
|  |                           |               |                       |                                 |           |         |           |                        |                    |
| Add  |                           |               | Remove Edit           | A Move Up                       | Move Down | Address | s Map     | Eiter                  |                    |
|  |                           |               |                       |                                 |           |         |           |                        |                    |

Figure 12: The final configuration of the system

#### 2.4 Nios II IDE

Next, create a new software project for your new computer system. Since each system is different (e.g., different memory layout, different peripherals), the software is tied to the system.

Run nios2-ide and switch the workspace to your project directory.

Select File $\rightarrow$ New $\rightarrow$ Nios II C/C++ Application.

Name the new (software) project something like lab3\_software (this is arbitrary—it creates a directory with this name in your project directory).

Select the "nios\_system.ptf" file in your project directory as the SOPC Builder System. This should set the CPU to "cpu\_0."

Finally, select the "Hello World" template and click Finish.

At this point, you can build and run the project on your board, but it does not do much. Instead, replace "hello\_world.c" in the lab3\_software directory (i.e., the name of the software project you specified) with the code in Figure 15, which exercises the LED flasher peripheral we added earlier.

#### 3 An FM Sound Synthesizer

This project is a stripped-down version of Ron Weiss, Gabriel Glaser, and Scott Arfin's *Terrormouse* project from 4840 in spring 2004. Feel free to use it as reference and adapt what VHDL you can, but make sure you understand what you are using.

In 1973, John Chowing introduced the idea of FM synthesis and the world has not sounded the same since. His basic insight is that FM waveforms are easy to produce and are "natural sounding." The basic FM equation is

$$x(t) = \sin\left(\omega_c t + I\sin(\omega_m t)\right)$$

where x(t) is the amplitude at time t,  $\omega_c$  is the carrier frequency (the fundamental tone we hear),  $\omega_m$  is the modulating frequency, and I is the modulation depth. The timbre of the sound is largely determined by the ratio  $\omega_c/\omega_m$ , which is generally set to an integer ratio (e.g.,  $\omega_c = 3\omega_m$ ).

The fundamental frequency of musical notes follow an exponential scale. The A above middle C is 440 Hz, and going up an octave doubles the frequency.

Western music is built on a scale of twelve semitones, each in equal ratio. Thus, the frequencies of a standard scale are of the

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity lab3 is
  port (
    signal CLOCK_50 : in std_logic; -- 50 MHz
    signal LEDR : out std_logic_vector(17 downto 0); -- LEDs
    SRAM_DQ : inout std_logic_vector(15 downto 0);
    SRAM_ADDR : out std_logic_vector(17 downto 0);
    SRAM_UB_N,
                           -- High-byte Data Mask
    SRAM_LB_N,
                           -- Low-byte Data Mask
    SRAM_WE_N,
                           -- Write Enable
    SRAM_CE_N,
                           -- Chip Enable
    SRAM_OE_N : out std_logic -- Output Enable
    ):
end lab3;
architecture rtl of lab3 is
  signal counter : unsigned(15 downto 0);
  signal reset_n : std_logic;
begin
  LEDR(17) <= '1';
  LEDR(16) <= '1';
  process (CLOCK_50)
  begin
    if rising_edge(CLOCK_50) then
      if counter = x"ffff" then
        reset_n <= '1';
      else
        reset_n <= '0';</pre>
        counter <= counter + 1;</pre>
      end if;
    end if:
  end process;
  nios : entity work.nios_system port map (
                                   \Rightarrow CLOCK_50,
    clk
    reset n
                                   => reset_n,
    leds_from_the_leds
                              \Rightarrow LEDR(15 downto 0),
    SRAM_ADDR_from_the_sram
                                   => SRAM_ADDR,
    SRAM_CE_N_from_the_sram
                                   => SRAM_CE_N,
    SRAM_DO_to_and_from_the_sram => SRAM_DO,
    SRAM_LB_N_from_the_sram
                                   \Rightarrow SRAM_LB_N,
    SRAM_OE_N_from_the_sram
                                   => SRAM_OE_N,
    SRAM_UB_N_from_the_sram
                                   => SRAM_UB_N,
    SRAM_WE_N_from_the_sram
                                   => SRAM_WE_N
    );
```

```
end rtl;
```

Figure 13: lab3.vhd: The top-level entity

| Classic Timing Analyzer Wizard: Project-Wide Defaults  | Classic Timing Analyzer Wizard: Default Frequency (fmax)   |
|--|--|
| Do you want to geddy an overal default thegancy regularment (final) for al clock in this<br>provides the second seco | Virba default inquericy limaid do you want for this project?<br>Qefault Insacc.<br>The scaling provide<br>Resulting provide<br>2000 min<br>Acide II you water force you can be a more clock signals, this default finan<br>assignment will be grownid. |
| <back next=""> Fritin Cancel</back>  |  |

Figure 14: Imposing a global timing constraint

```
#include <io.h>
#include <system.h>
#include <stdio.h>
#define IOWR_LED_DATA(base, offset, data) \
  IOWR_16DIRECT(base, (offset) * 2, data)
#define IORD_LED_DATA(base, offset) \
  IORD_16DIRECT(base, (offset) * 2)
#define IOWR_LED_SPEED(base, data) \
  IOWR_16DIRECT(base + 32, 0, data)
int main()
{
  int i;
  printf("Hello_Michael\n");
  IOWR_LED_SPEED(LEDS_BASE, 0x0040);
  for (i = 0 ; i < 8 ; i++) {
    IOWR_LED_DATA(LEDS_BASE, i, 3 << (i * 2));</pre>
    printf("writing_%x\n", i);
  }
 for (i = 8 ; i < 16 ; i++) {
    IOWR_LED_DATA(LEDS_BASE, i, 3 << (32 - (i * 2)));</pre>
    printf("writing_%x\n", i);
  }
  for (i = 0 ; i < 16 ; i++) {</pre>
    printf("reading_%x_=_%x\n"
                                 i,
           IORD_LED_DATA(LEDS_BASE, i));
  }
 printf("Goodbye\n");
 return 0;
}
```

Figure 15: A hello\_world.c file that imitates KITT from Knight Rider (yes, I lived through the 80s). It sets the cycling speed, fills the LED\_flasher peripheral with a pattern, then reads it back to verify it works as memory.

form

$$f = 440 \cdot 2^{p/12}$$

where f is the frequency in Hertz, p = 0 is the A above middle C, p = 1 is A $\sharp$ , p = 2 is B, p = 3 is C, p = 12 is the A the octave above, p = -12 is the A the octave below, etc.

#### 3.1 Starting Points

In the lab3.tar.gz file, we have supplied some helpful files you should use as a starting point. The most interesting is de2\_wm8731\_audio.vhd, which implements an interface to the Wolfson WM8371 audio codec on the DE2 board. This operates either in a test mode that generates a sinewave (a pure tone), or as a parallel-to-serial converter.

We included two Verilog files that configure the WM8371: de2\_i2c\_controller.v and de2\_i2c\_av\_config.v. You should be able to just instantiate them without modification. They send initialization commands through the two-wire I<sup>2</sup>C bus.

lab3\_audio.vhd is a simple top-level module that instantiates the audio controller in test mode and the two I<sup>2</sup>C bus components. You can build a new Quartus project with this as a starting point and should hear a tone on line out.

Finally, we have included a PS/2 keyboard controller.

#### 3.2 The PS/2 Controller

The file de2\_ps2.vhd is the core of an Avalon peripheral that can read data coming from a PS/2 keyboard. This is simpler than the one you used in lab 2 (e.g., it cannot send data to the keyboard), but will suffice. Use SOPC Builder to create a new component around it and connect the two PS/2 lines (clock and data) to the appropriate pins.

Important: for this peripheral, set the "Slave addressing" mode to "NATIVE." This affects whether the peripheral will appear as two words (register mode) or two bytes (memory mode).

This peripheral presents a simple two-word interface: reading the first byte of the first word returns 1 if a byte is available and zero otherwise. Reading the first byte of the second word returns the byte received from the keyboard.

Thus, if DE2\_PS2\_BASE is the base address of the PS/2 controller peripheral, you can wait for the next data byte using

#### unsigned char code;

while (!IORD\_8DIRECT(DE2\_PS2\_BASE, 0)) ; /\* Poll the status \*/
code = IORD\_8DIRECT(DE2\_PS2\_BASE, 4);
/\* Get received byte \*/

#### 3.3 What To Do

You have two things to design: an Avalon peripheral that can generate an FM waveform under software control that you feed to the supplied WM8371 audio controller, and a C program that translates key events from the PS/2 keyboard into commands for your FM oscillator. Basically, make the PS/2 keyboard behave like a dumb piano keyboard.

Using the LED flasher example peripheral, build an Avalon peripheral that presents registers that control the oscillation frequency, the modulation depth, and a simple volume control (on/off) that lets you turn off the oscillator when no key is pressed.

Use a sinewave lookup table to generate the waveform. Step through it at different rates to generate the different tones.

First, develop the oscillator functionality first using Model-Sim to test that your waveform is as you expect. Then, integrate it with the supplied audio codec controller and make a VHDL-only design that actually generates sound. Finally, add an Avalon interface to your oscillator, use SOPC Builder to integrate a Nios II, the supplied PS/2 keyboard controller, and your new component, and develop the software.

#### 4 A Bouncing Video Ball

After you implement this project, you will feel a much stronger connection with Nolan Bushnell, the inventor of the first commercially-successful videogame, Pong. Of course, you won't find it quite as lucrative.

You have two things to design: an Avalon component that displays a small white circle on the screen under software control, and a C program that controls the position of this circle.

Use the code in de2\_vga\_raster.vhd as a starting point for your Avalon component. It is a simple VGA controller that displays a large white rectangle against a blue background. It currently does not have a bus interface. You need to add one and change its behavior so that it displays a small circle. The lab3\_vga.vhd file holds a simple top-level for this component that can be used to build a skeleton project.

First, adapt the video generator to display a circle instead of a rectangle. Make sure you add signals that control where on the screen the circle appears. While developing this, you can just set these to constants; later software will supply them.

Your other challenge is building an Avalon peripheral. Use the LED flasher from the tutorial as a basis for building a peripheral. First, get an Avalon peripheral working by building the registers you plan to use in the end for your video controller and connect them to some LEDs to verify you can communicate from the software to the hardware.

Once you have a working peripheral, integrate your modified video controller with it.

Finally, write a simple C program that bounces the ball around the screen.