# CSEE W4840 Embedded System Design Lab 3

# Stephen A. Edwards

# Due February 24, 2010

# 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 name is case sensitive exactly match the entity name in the design file.	e and must
lab3	
Use Existing Project Settings	
Dise Existing Project Settings	
< Back Next > Finish	Cancel

Figure 1: Naming a new Quartus project

mily: Cyclone II Target device O Auto device select	ed by the Fitter		•	– Show in 'A' <u>P</u> ackage: Pin <u>c</u> ount:	vailable de <sup>.</sup> Any Any	vice' list	- -
Specific device sel	lected in 'Availa	able devices	:' list	Speed grad	de: Any		•
		💌 S <u>h</u> ow a	advanced d	levices			
				🗖 HardCo	py compat	ible <u>o</u> nly	
vailable <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	
P2035F67208	1.2V 1.2V	33216 33216	475 475	483840 483840	70 70	4	
	1.70	3371K	475	A8 (8A1)		1	١Ē
Companying daying							-
Companion device							
HardCopy II:			$\nabla$				
			irces				

Figure 2: Selecting the device in Quartus

System N	lame: nic	os_syste	m	
Target Hl	۵L: 🔾 ۱	/erilog		
	۱۱	/HDL		
-				
-				 

Figure 3: Naming a new system in SOPC Builder

	Altera SOPC Builder - 19 Create new con - Nios Il Processor - Bridges and Adapter	Target Device Family:	Cyclone II	Clock Settings	lomo External	Source	50.0 Mi	+iz	Add
	<ul> <li>Legacy Components</li> <li>Memories and Memo</li> <li>Peripherals</li> <li>PLL</li> <li>User Logic</li> </ul>	Use Con	Module Name		Description	Clock	Base	End	IR
Add Remove Est A Move Up V Move Down Address Map Filer			Remove		nve i in 🖉 Move Don	an Add	Iress Man	Fiter	

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 -	cpu	
MogaCare' Nios	s II Processor			Documentation
Parameter Settings				
and the second	es and Memory Interfaces	Advanced Features > JTAG	Debug Module 🔰 Custom Instructions	<u> </u>
Core Nios II				/
Select a Nios II core:	-			
	Nios II/e	ONios II/s	ONios II/f	
Nios II Selector Guide Family: Cyclone II	RISC 32-bit	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply	RISC 32-bit Instruction Cache Branch Prediction Hardware Multiply	
f <sub>system</sub> : 50.0 MHz cpuid: 0		Hardware Divide	Hardware Divide Barrel Shifter Data Cache Dynamic Branch Prediction	
Performance at 50.0 M Logic Usage	600-700 LEs	Up to 25 DMIPS 1200-1400 LEs	Up to 51 DMIPS 1400-1800 LEs	
Memory Usage	Two M4Ks (or equiv.)	Two M4Ks + cache	Three M4Ks + cache	
	bedded Multipliers	Hardware Divide		
Reset Vector: Me	emory:	✓ Offset: 0x	0	
Exception Vector: Mer	mory:	→ Offset: 0x2	20	
🦄 Warning: Reset vec	tor and Exception vector can	not be set until memory devices	are connected to the Nios II process	ior.
			Cancel < Bac	k Next > Einis

Figure 5: Adding an Nios II processor in SOPC Builder

Nos IProcesso     Nos IPr	Hoad Processor     Hoad Pro	Altera SOPC Builder	Target Device Family: Cyclone II		k Settings Name	Sour	e .	м	Hz .	l insert
Percipitential PL User logic User logic Valeo and Image Pro Valeo and Image Pro Valeo and Image Pro Valeo and Image Pro Jag_debug_module Protection_matter Protection_matter Protection_matt	Pergeratis PL, Log     Use Torrac     Model hame     Description     Clock     Base     End     In       Use Sorrac     Bege     Bege     Nos IP rotextion Matter Avison Matter     Nos IP rotextion Matter     IR0	Nos Il Processo     Bridges and Adapter     Interface Protocols     Legacy Components     Memories and Memo	Device Family: Cyclone II	clk		External		58.8		Add
Video and Image Pol	- Video and Image Point  - Video and Image Point - Video and Image - Video and Image Point - Video an	Peripherals     PLL		ame		on (	lock	Base	End	IRG
	To Do: cpu: No reset vector has been specified for this CPU. Please parameterize the CPU to resolve this issue		instruction_r data_master		Avalon Master Avalon Master	clk	e 0:			31
To Dio <b>spu</b> No exception vector has been specified for this CPU. Please parameterize the CPU to resolve this issue Warning: <b>spu</b> : Reset vector and Exception vector cannot be set until memory devices are connected to the Nios Il processor		Add To Do: cpu: No reset v To Do: cpu: No excepti	Instruction dda metal Jag_debug_ Regione	Edt Please par CPU. Pleas	Avalon Master Avalon Master Avalon Stave	Move Down	Address §	×00000800	0×00000 fff	31

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>
```

```
File Templates
 Introduction HDL Files Signals Interfaces Component Wizard
 About Signals
                                                                                     Width Direction
           Name
                                                                  Signal Type
                                        Interface
 chipselect
                         avalon_slave_0
                                                             chipselect
                                                                                            nput
₩rite

read
                         avalon_slave_0
avalon_slave_0
                                                             write
read
                                                                                            input
                                                                                            nput
🕅 address
                         avalon slave O
                                                             address
                                                                                    18
                                                                                           input
 readdata
                         avalon slave_(
                                                              readdata
                                                                                             output
                         avalon_slave_0
                                                             writedata
                                                                                    16
                                                                                            input
 💹 byteenable
                         avalon slave O
                                                             byteenable
                                                                                            nput
 SRAM_DQ
                         export_0
                                                             export
                                                                                    16
                                                                                             oidir
                         export_0
                                                             export
                                                                                            output
 SRAM UB N
                         export 0
                                                             export
                                                                                            output
  SRAM_LB_M
                         export_(
                                                              ехрог
                                                                                             .
utput
 SRAM_WE_N
                         export_0
                                                             export
                                                                                            output
 SRAM CE M
                         export_
                                                              export
                                                                                            outout
 SRAM_OE_N
                                                                                            output
                         export_0
                                                             ехрог
                                    Add Signal Remove Signal
Info: No errors or warnings
                         Help
                                        Prev
                                                     Next 🕨
                                                                      Einish.
```

Figure 8: Associating the signals with interfaces

File Templates	
Introduction HDL	Files Signals Interfaces Component Wizard
About Interfaces	
<b>▽ "export_0"</b> (Co	anduit Output)
Name:	export_0
Туре:	Conduit Output
Associated Clock:	none 🗨
	(Avalon Slave)
Name:	avalon_slave_0
	Avalon Slave
Associated Clock:	
Avalon Slave S	Settings
Slave Ad	dressing DYNAMIC -
Minimum Arbitration	Shares 1
Can receive stdo	ut/stderr
Interleav	e Bursts
-▼ Avalon Slave 1	liming
- Artalorrolato	-
Setu	p 0 Read Wait 1 Hold 0 Units Cycles ▼ Write Wait 0
Dis allo al Turco de	
Pipelined Transfe	Read Latency 0 Max Pending Read Transactions 0
	Read Latency U Max Pending Read Transactions U
	Add Interface Remove Interfaces With No Signals
<ol> <li>Info: No errors or</li> </ol>	warninge
J 110. NO CITOIS OF	TT MITHING .

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

h altra d Olamada	es Signals Interfaces Co	omponent Wizard			
About Signals Name	Interface	Signal 1	Type Vidth	Direction	
clk	clock	⊂ cik	Type vwaar	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	chipselect	1	input	
address	avalon_slave_0	address	5	input	
readdata	avalon slave 0	readdata	16	output	
writedata	avalon_slave_0	16	input		
writedata avalon_slave_0 writedata 16 input leds export_0 export 16 output					
	Add Signal	Remove Signal			
hifo: No errors or we					

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.

Impose a global timing constraint by choosing Assignments→Classic Timing Analyzer Wizard.

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;
           : in std_logic;
read
                std_logic;
write
           : in
chipselect : in std_logic;
address
          : in unsigned(4 downto 0);
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>
```

#### process (clk) begin if rising\_edge(clk) then if reset\_n = '0' then readdata <= (others => '0'); display\_address <= (others => '0'); counter <= (others => '0'); counter\_delay <= (others => '1'); else if chipselect = '1' then if address(4) = '0' then if read = '1' then 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 if;
end if;
end if;
```

end process;

```
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.

System Contents System General	tion									
🛄 Altera SOPC Builder	Targe	4		Clock Settings						
- 🛄 Create new component	Device	Family:	yclone I	Name		Source		M	Hz	
<ul> <li>de2_led_flasher</li> </ul>	Devio	s i unity. E	AYCHURD I	clk	External			50.0		Add
<ul> <li>e de2_srem_controller</li> </ul>										Remove
<ul> <li>Nios Il Processor</li> </ul>										
<ul> <li>Bridges and Adapters</li> <li>Interface Protocols</li> </ul>										
Legacy Components	Line	Connec	Module Name	Description		Clock		Base	End	IRQ
- Memories and Memory Controlle	- C000		E CDU	Nos I Processor		CIOCA	-	Liase	LIN	1102
Peripherals     PLL			instruction master			clk				
- PLL User Logic			data master	Avalon Master				IRQ O	IRC	31
- Video and Image Processing		$  \rightarrow \rangle$	itag_debug_modul	le Avalon Slave			in 6	x00100800	0x00100fff	
	2		🖃 sram	de2_sram_controller						
		$\rightarrow$	avalon_slave_0	Avalon Slave			if 0	×00080000	0×000 <del>1111</del>	
	V		⊡ leds	de2_led_flasher						
		$\rightarrow$	avalon_slave_0	Avalon Slave		clk	if 6	x00101000	0×0010103f	
	2	L L	🖃 jtag_uart	JTAG UART						
		$\rightarrow$	avalon_itag_slave	Avalon Slave		clk	if 0	x00101040	0×00101047	⊢-10
< II >										
Add			Remove Edt	A Move Up	Move Dov	vn A	ddres	s <u>M</u> ap	<u>F</u> ilter	
		_				enerate				
			Egit Help	Prev Next						

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
                                   \Rightarrow SRAM_OE_N,
    SRAM_UB_N_from_the_sram
                                   \Rightarrow 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     X	Classic Timing Analyzer Wizard: Default Frequency (fmax)
Dy sport average to specify an overal default frequency regularment (finals) for a clocks in the properties of the start finals for specific clocks. Which never an end default image requesters undefault is to specify to this project? Default af mateging of default for the programments and project to specify to this project? If a priority default is to specify the specific clocks. If a priority default is to specify the specific clocks. If a priority default is to specify the specific clocks. If a priority default is to specify the specific clocks. If a priority default is to specify the specific clocks. If a priority default is to specify the specific clocks. If a priority default is to specify the specific clocks. If a priority default is the specific clocks. If a priority default is the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clock of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specific clocks. If a priority default is a priority of the specif	Vota default negating (final) do you want for this project? [order in final: Resulting proof. 2000 min Note: If you reter that negatineness for one or more clock signals, this default finas assignment will be ground.
Kack Next> Frish Cancel	<back next=""> Finds 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++) {</pre>
    IOWR_LED_DATA(LEDS_BASE, i, 3 << (i * 2));</pre>
    printf("writing_%x\n", i);
  }
 for (i = 8 ; i < 16 ; i++) {</pre>
    IOWR_LED_DATA(LEDS_BASE, i, 3 << (32 - (i * 2)));
    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#, 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.