CSEE W4840 Embedded System Design Lab 1

Stephen A. Edwards

Due February 7, 2013 for last names starting A–N, February 14, 2013 for last names O–Z

Abstract

Learn to use the Altera Quartus development envrionment and the DE2 boards by implementing a small hardware design that displays and modifies the contents of a small memory.

1 Introduction

Use the Altera DE2 board to implement a simple hardware design. Describe its behavior using the VHDL language and use Altera's Quartus tools to synthesize and program the FPGA device. Use a VHDL simulator to verify and debug the design.

The circuit you program into the FPGA will display and modify the contents of a 16×8 bit RAM. Although there are dedicated RAM chips on the DE2 board, for simplicity use a RAM inside the FPGA. Use four pushbuttons as inputs and three seven-segment LED displays as outputs. Two push buttons should step up and down through the sixteen RAM locations; the other two should increment and decrement the contents of the currently-displayed memory location. One seven-segment display should show the current address (0–F), two others should display the contents of that location in hexadecimal (00–FF).

You will learn to set up a project in the Altera Quartus tool, run a VHDL simulation, and compile and download your design to the FPGA. VHDL is a hardware description language, and the process of using it is very different than developing programs in C++ or Java. You will need these skills in later labs and while you are developing your project.

Below, we introduce the DE2 board, show how to start a new project from a template, add VHDL code to a project, simulate it, and compile and download a design to the FPGA.

2 The DE2 Board

Figure 1 shows the Altera DE2 board. It consists of an Altera Cyclone II FPGA connected to a variety of peripherals including 512K of SRAM, 4 MB of Flash, 8 MB of SDRAM, VGA output, Ethernet, audio input and output, and USB ports. For this lab, we will use four of the eight seven-segment LEDs and the four blue pushbuttons. There are three USB connectors on the top of the board. The leftmost one—the one nearest the 9V DC connector—is for connecting the Altera "Blaster" cable to the workstation. It is through this connection that the FPGA will be programmed, that debugging information flows, etc. The other two USB ports can be used in projects.

The DE2 board holds two quartz crystal oscillators (clock sources: little silver boxes labeled with their frequencies). We will use the 50 MHz clock for this lab; there is also a 27 MHz clock designed for video timing.

The DE2 board has built-in configuration for testing and demonstration purpose. You can verify the board is working properly by observing this default behavior. Use the following procedure to power up the DE2 board.

Connect the USB blaster cable from the USB port on the workstation to the USB Blaster connector on the DE2 board. Connect the 9 V adapter to the DE2's power connector at the top left corner. Verify the RUN/PROG switch on the left edge of the DE2 board (just to the left of the LCD display) is in the RUN position.

Power on the DE2 board by pressing the red ON/OFF switch in the upper left corner. The LEDs should flash, the LCD should display "Welcome to the Altera DE2 Board," and the VGA output should display an Altera/Terasic logo page.

To download our design we use a JTAG connection (JTAG is a ubiquitous standard that stands for the IEEE Joint Test Action Group). The Altera Quartus tool running on the workstation sends the configuration bit stream through the USB cable to the Cyclone II FPGA. Once programmed, the FPGA retains its configuration as long as power is applied to the board; it is lost when the power is turned off. We cover the details of this process below.

3 Getting Started with Quartus

Quartus is Altera's development environment for FPGAs. It consists of an IDE and a "compiler" that can translate circuits described in VHDL into configuration data for the FPGA. Start the Quartus IDE by running the `quartus` command. This requires the PATH and LM_LICENSE_FILE environment variables to be set.

Altera provides a variety of reference designs for the DE2. For lab 1, we modified the *DE2_Top* design, which contains information about what each pin on the FPGA is connected to and a top-level VHDL module with a port for each pin.

Download the lab1.tar.gz file from the class website and extract it with "tar zxf lab1.tar.gz" This will place the project files, listed in Table 1, in the current directory.

DE2_TOP.qpf is the top Quartus project file. To open the project file, use File→Open Project and select *DE2_TOP*. Once the project is opened, you can see and change I/O pin assignments with Assignment→Assignment Editor→. Figure 2 shows this dialog.

Table 1: Files in the DE2_TOP project

Name	Role
DE2_TOP.qpf	Quartus Project File
DE2_TOP.qsf	Pin assignments, etc.
DE2_TOP.vhd	Top-level VHDL file

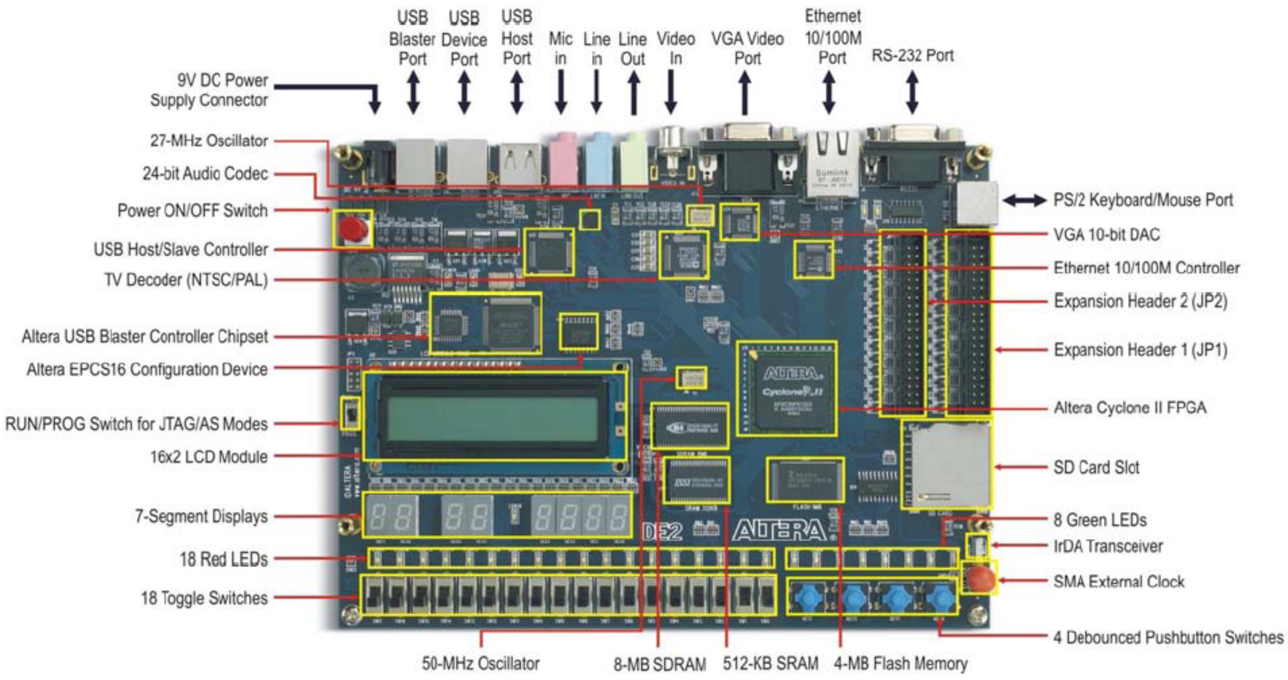
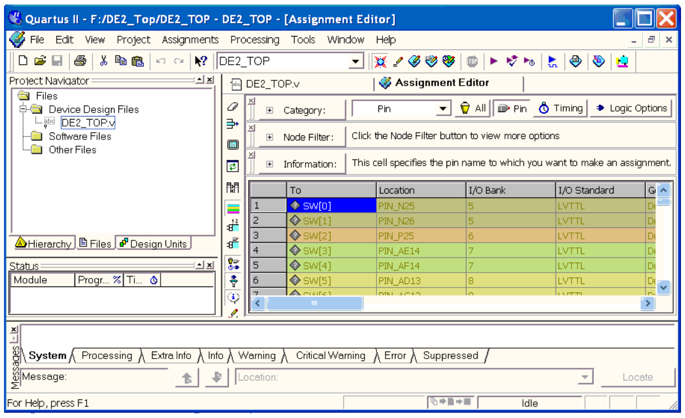


Figure 1: The Altera DE2 board


Quartus II - F:/DE2_Top/DE2_TOP - DE2_TOP - [Assignment Editor]
File Edit View Project Assignments Processing Tools Window Help
DE2_TOP
Project Navigator
Files
Device Design Files
DE2_TOP.v
Software Files
Other Files
Hierarchy Files Design Units
Status
Module Progr... % Ti...
DE2_TOP.v
Assignment Editor
All Pin Timing Logic Options
System Processing Extra Info Info Warning Critical Warning Error Suppressed
Message:
Location:
Locate
For Help, press F1
Idle

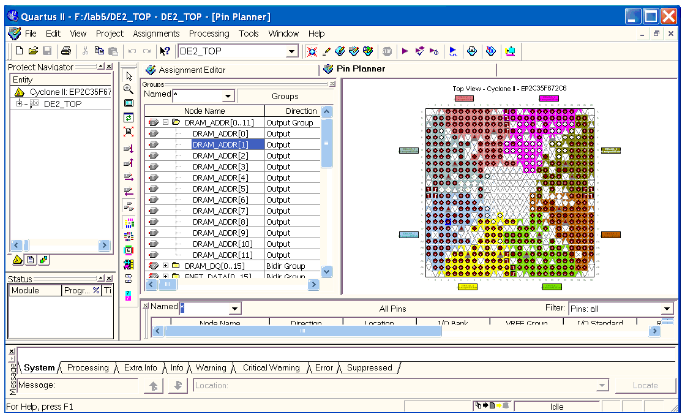
Category: Pin

Node Filter: Click the Node Filter button to view more options

Information: This cell specifies the pin name to which you want to make an assignment.

	To	Location	I/O Bank	I/O Standard
1	SW[0]	PIN_N25	5	LVTTL
2	SW[1]	PIN_N26	5	LVTTL
3	SW[2]	PIN_P25	6	LVTTL
4	SW[3]	PIN_AE14	7	LVTTL
5	SW[4]	PIN_AF14	7	LVTTL
6	SW[5]	PIN_AD13	8	LVTTL

Figure 2: Assigning Pins Textually


Quartus II - F:/lab5/DE2_TOP - DE2_TOP - [Pin Planner]
File Edit View Project Assignments Processing Tools Window Help
DE2_TOP
Project Navigator
Entity
Cyclone II: EP2C35F67
DE2_TOP
Status
Module Progr... % Ti
Assignment Editor
Pin Planner
Groups
Named *
Top View - Cyclone II - EP2C35F672C6
Named *
All Pins
Filter: Pins: all
Node Name Direction Location I/O Bank VREF Group I/O Standard
System Processing Extra Info Info Warning Critical Warning Error Suppressed
Message:
Location:
Locate
For Help, press F1
Idle

Node Name	Direction
DRAM_ADDR[0..11]	Output Group
DRAM_ADDR[0]	Output
DRAM_ADDR[1]	Output
DRAM_ADDR[2]	Output
DRAM_ADDR[3]	Output
DRAM_ADDR[4]	Output
DRAM_ADDR[5]	Output
DRAM_ADDR[6]	Output
DRAM_ADDR[7]	Output
DRAM_ADDR[8]	Output
DRAM_ADDR[9]	Output
DRAM_ADDR[10]	Output
DRAM_ADDR[11]	Output
DRAM_DQ[0..15]	Bidir Group

Figure 3: Assigning Pins Graphically

For Quartus to configure an FPGA, it must know which pins on the FPGA perform what roles (i.e., what each is named). This information is board-specific since the pins on the FPGA can be wired to arbitrary peripherals. The *DE2_TOP.qsf* file contains this information for the DE2 board.

DE2_TOP.vhd is the top-level VHDL module for the project, which mostly lists the top-level ports, i.e., the VHDL names for the pins. It also sets the state of the LEDs.

Although you do not need to modify I/O pin settings for this lab, you may need to do so in the future. Assignment→Pin Planner, shown in Figure 3, opens a display that shows the physical location the pins on the FPGA and their assignments.

Another way to assign pins is to start with a pin assignment file in CSV format. While you do not need one for this lab, you will need to do so later: go to Assignments→Import Assignments...

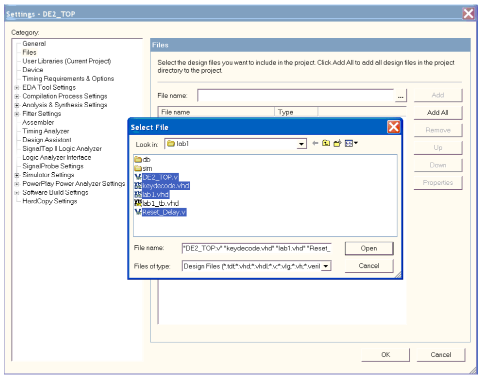
4 Compiling for the FPGA

The supplied project can be compiled and downloaded to the board, altough it does not do much. First, make sure all the source files are included in the project. From the Project navigator window, click on the *Files* tab. This will display the VHDL files that will be compiled into the FPGA. To add a file, select Project→Add/Remove Files in Project. This opens the window in Figure 4.

Select VHDL files from the pop up window. If you have written multiple VHDL files, add each of them. Do not add any test benches (used for simulation) to the list of device design files since they cannot be compiled into hardware.

Make sure you indicate which file that contains the Top Entity: right-click the appropriate file in the main list of files (i.e., after you have added them).

Now we are ready to compile. Select Processing→Start Compilation to start the compilation process (Figure 5). The



Settings - DE2_TOP

Category:
- General
- Files
- User Libraries (Current Project)
- Device
- Timing Requirements & Options
- EDA Tool Settings
- Compilation Process Settings
- Analysis & Synthesis Settings
- Fitter Settings
- Assembler
- Timing Analyzer
- Design Assistant
- SignalTap II Logic Analyzer
- Logic Analyzer Interface
- SignalProbe Settings
- Simulator Settings
- PowerPlay Power Analyzer Settings
- Software Build Settings
- HardCopy Settings

Files

Select the design files you want to include in the project. Click Add All to add all design files in the project directory to the project.

Select File

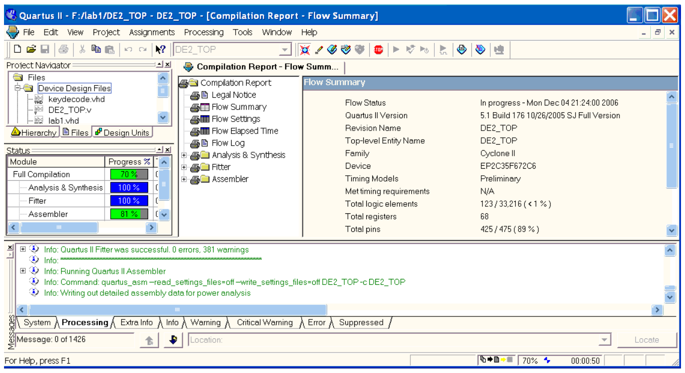
Look in: lab1

- db
- sim
- DE2_TOP.v
- keydecode.vhd
- lab1.vhd
- lab1_tb.vhd
- Reset_Delay.v

File name: "DE2_TOP.v" "keydecode.vhd" "lab1.vhd" "Reset_

Files of type: Design Files (*.tdf;*.vhd;*.vhdl;*.v;*.vlg;*.vh;*.veril

Figure 4: Adding files to a project



Quartus II - F:/lab1/DE2_TOP - DE2_TOP - [Compilation Report - Flow Summary]

Project Navigator: Files / Device Design Files: keydecode.vhd, DE2_TOP.v, lab1.vhd

Module	Progress %
Full Compilation	70 %
Analysis & Synthesis	100 %
Fitter	100 %
Assembler	81 %

Compilation Report: Legal Notice, Flow Summary, Flow Settings, Flow Elapsed Time, Flow Log, Analysis & Synthesis, Fitter, Assembler

Flow Summary

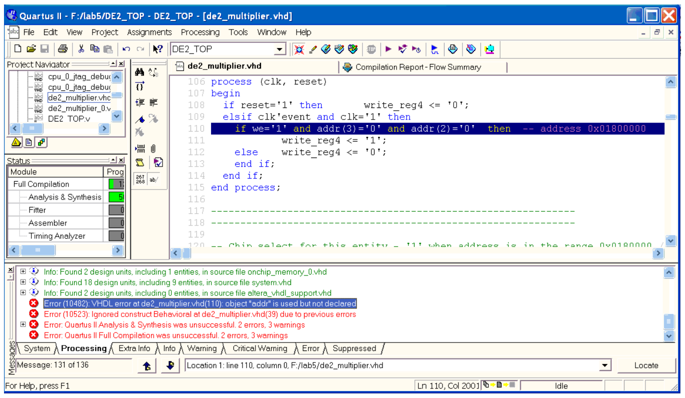
Flow Status	In progress - Mon Dec 04 21:24:00 2006
Quartus II Version	5.1 Build 176 10/26/2005 SJ Full Version
Revision Name	DE2_TOP
Top-level Entity Name	DE2_TOP
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Preliminary
Met timing requirements	N/A
Total logic elements	123 / 33,216 (< 1 %)
Total registers	68
Total pins	425 / 475 (89 %)

```
Info: Quartus II Fitter was successful. 0 errors, 381 warnings
Info: *******************************************************************
Info: Running Quartus II Assembler
Info: Command: quartus_asm --read_settings_files=off --write_settings_files=off DE2_TOP -c DE2_TOP
Info: Writing out detailed assembly data for power analysis
```

Message: 0 of 1426

For Help, press F1 — 70% — 00:00:50

Figure 5: Compiling a design



Quartus II - F:/lab5/DE2_TOP - DE2_TOP - [de2_multiplier.vhd]

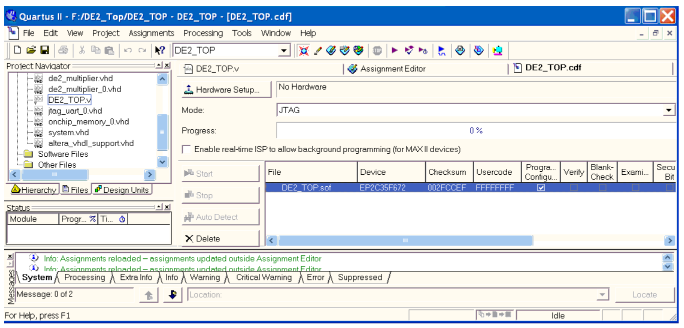
```
106 process (clk, reset)
107 begin
108   if reset='1' then       write_reg4 <= '0';
109   elsif clk'event and clk='1' then
110     if we='1' and addr(3)='0' and addr(2)='0'  then  -- address 0x01800000
111            write_reg4 <= '1';
112     else   write_reg4 <= '0';
113     end if;
114   end if;
115 end process;
116
117 ---------------------------------------------------------------
118 ---------------------------------------------------------------
119
120 -- Chip select for this entity - '1' when address is in the range 0x0180000
```

```
Info: Found 2 design units, including 1 entities, in source file onchip_memory_0.vhd
Info: Found 18 design units, including 9 entities, in source file system.vhd
Info: Found 2 design units, including 0 entities, in source file altera_vhdl_support.vhd
Error (10482): VHDL error at de2_multiplier.vhd(110): object "addr" is used but not declared
Error (10523): Ignored construct Behavioral at de2_multiplier.vhd(39) due to previous errors
Error: Quartus II Analysis & Synthesis was unsuccessful. 2 errors, 3 warnings
Error: Quartus II Full Compilation was unsuccessful. 2 errors, 3 warnings
```

Message: 131 of 136 — Location 1: line 110, column 0, F:/lab5/de2_multiplier.vhd

For Help, press F1 — Ln 110, Col 2001 — Idle

Figure 6: Diagnosing errors



Quartus II - F:/DE2_Top/DE2_TOP - DE2_TOP - [DE2_TOP.cdf]

Hardware Setup... No Hardware

Mode: JTAG

Progress: 0 %

Enable real-time ISP to allow background programming (for MAX II devices)

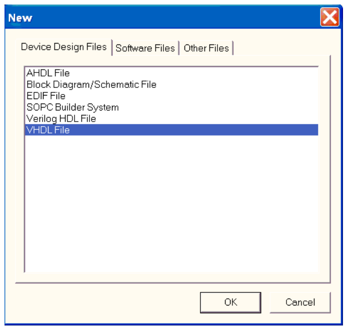
File	Device	Checksum	Usercode	Progra... Configu...	Verify	Blank-Check	Exami...	Secu Bit
DE2_TOP.sof	EP2C35F672	002FCCEF	FFFFFFFF	✓				

```
Info: Assignments reloaded -- assignments updated outside Assignment Editor
Info: Assignments reloaded -- assignments updated outside Assignment Editor
```

Message: 0 of 2

For Help, press F1 — Idle

Figure 7: Programming the FPGA



New

Device Design Files | Software Files | Other Files

- AHDL File
- Block Diagram/Schematic File
- EDIF File
- SOPC Builder System
- Verilog HDL File
- VHDL File

Figure 8: Creating a new VHDL file

window on the left reports progress.

A pop-up appears when compilation completes. If there are errors, use the Messages window to locate them (Figure 6). As usual, the first error listed is most trustworthy; any others may be due to earlier errors. A compilation process usually generates some warnings. Most are harmless, but it is worth fixing them them to avoid masking a genuine problem.

Double clicking on an error message will highlight the suspect VHDL in the editor window. The compiler may also display warning messages, which can be explored in the same way. You can obtain more information about a specific error or warning by selecting it and pressing the F1 key.

4.1 Programming the FPGA

Once your design has been compiled, it can be downloaded to the FPGA. Select Tools→Programmer, which will display the window in Figure 7. It should list the DE2_TOP.sof file to be programmed into the EP2C35F672 device (Altera's charming name for the FPGA on the DE2).

You may have to click on the "Hardware Setup..." button and select the USB-Blaster cable. Make sure the board's USB cable is plugged into the port marked "blaster" (i.e., nearest the power connector).

Click the check box under *Program/Configure* for the DE2_TOP.sof file destined for the FPGA and then click *Start* to download your design to the FPGA. If all goes well, the design should spring to life.

5 Editing VHDL

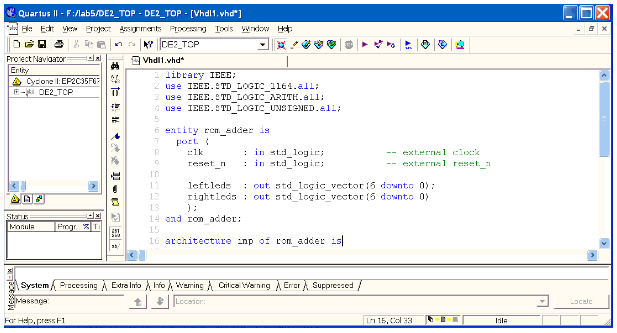
The next step is to code your circuit in VHDL. Quartus provides a good VHDL text editor, which provides syntax highlighting, language templates, and other aspects of a good IDE. To create a new VHDL file in your project, select File→New. This will bring up the dialog in Figure 8.

Select the VHDL file option and click OK. This brings up a window where you can enter VHDL code (Figure 9).

The verbose syntax of VHDL is probably unfamiliar to you. To help, the Quartus tool provides a collection of VHDL templates, which provide examples of various types of VHDL constructs, such as an entity declaration, a process statement, and an assignment statement.

To use a VHDL template, select Edit→Insert Template. This will open a window such as Figure 10.

Select "VHDL" and the type of template you want. The OK button inserts the template in the active source file. Then fill


Quartus II - F:/lab5/DE2_TOP - DE2_TOP - [Vhdl1.vhd*]
File Edit View Project Assignments Processing Tools Window Help
DE2_TOP
Project Navigator
Entity
Cyclone II: EP2C35F67
DE2_TOP
Status
Module Progr... % Ti
Vhdl1.vhd*
System Processing Extra Info Info Warning Critical Warning Error Suppressed
Message:
Location:
Locate
For Help, press F1
Ln 16, Col 33
Idle

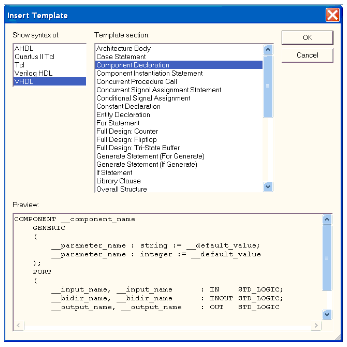
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;

entity rom_adder is
  port (
    clk       : in std_logic;           -- external clock
    reset_n   : in std_logic;           -- external reset_n

    leftleds  : out std_logic_vector(6 downto 0);
    rightleds : out std_logic_vector(6 downto 0)
    );
end rom_adder;

architecture imp of rom_adder is
```

Figure 9: Editing a VHDL file



```
COMPONENT __component_name
    GENERIC
    (
        __parameter_name : string := __default_value;
        __parameter_name : integer := __default_value
    );
    PORT
    (
        __input_name, __input_name      : IN    STD_LOGIC;
        __bidir_name, __bidir_name      : INOUT STD_LOGIC;
        __output_name, __output_name    : OUT   STD_LOGIC
```

Figure 10: Inserting a VHDL template

in the details in the template, such as the name of an entity.

6 The Lab 1 Design

Your goal is to implement a memory display/modification circuit whose block diagram is shown in Figure 11. Input ports are on the left; output ports are on the right.

To add the lab1 component to the project, instantiate it in the top-level architecture in DE2_TOP.vhd.

```
architecture datapath of DE2_TOP is

begin

U1: entity work.lab1 port map (
   clock => clock_50,
   key => key,
   hex6 => hex6,
   hex5 => hex5,
   hex4 => hex4
);
```

Here, the ports on your lab1 entity are mapped to top level ports. The naming of these top level ports, such as CLOCK_50, SW, KEY and HEX6 4, are all defined in the Quartus .qsf file. The ports named in the DE2_TOP.vhd and QSF file must match.

Here, U1 is the name of the instance. Note that you can pick any valid VHDL name for it; you can use this name to refer to the instance in simulation.

Remember to disable the constant assignments to HEX4, HEX5, and HEX6 in the DE2_TOP.vhd file when you add your lab1 component.

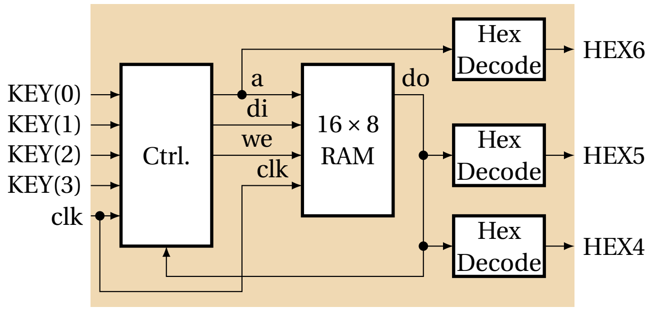


Figure 11: The block diagram of lab 1

6.1 RAM

Your design should include a 16 × 8 bit RAM, but what kind of RAM? The DE2 board contains an SRAM chip, an SDRAM chip, and RAM within the FPGA itself. The SDRAM chip provides the highest capacity but requires a complicated controller. The SRAM chip is smaller, much simpler to use, and provides more storage than RAM on the FPGA. However, RAM internal to the FPGA, so-called "block RAM," is the smallest, fastest, and easiest to use. Use it for this lab.

The FPGA block RAM can be configured many different ways, e.g., as one big memory, as many small regions, and as bits, bytes, or words. The easiest way to ask for a particular type of RAM is to is to allow the Quartus tool to infer it from the use of an array in VHDL. Below is code from which Quartus will infer a small RAM block.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity raminfr is
  port (
    clk : in std_logic;
    we  : in std_logic;
    a   : in unsigned(3 downto 0);
    di  : in unsigned(7 downto 0);
    do  : out unsigned(7 downto 0)
  );
end raminfr;

architecture rtl of raminfr is
type ram_type is array (0 to 15) of unsigned(7 downto 0);
signal RAM : ram_type;
signal read_a : unsigned(3 downto 0);

begin

process (clk)
begin
  if rising_edge(clk) then
    if we = '1' then
      RAM(to_integer(a)) <= di;
    end if;
    read_a <= a;
  end if;
end process;
```

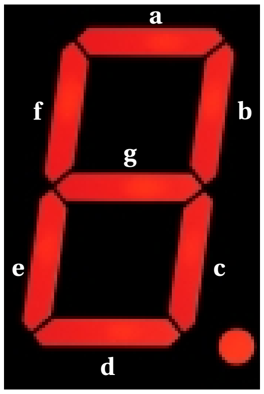


Figure 12: A seven-segment LED display. E.g., hex0(0) is a; hex0(6) is g

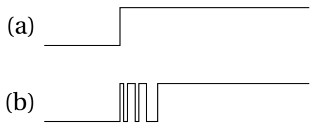


Figure 13: Keybounce illustrated. (a) The ideal response. (b) What actually happens.

```
do <= RAM(to_integer(read_a));

end rtl;
```

Here, the *to_integer* function converts an input in the form *unsigned* to an integer index for accessing the array.

6.2 Seven-Segment LEDs

The block diagram in Figure 11 includes three seven-segment LED output decoders. Each segment of each LED is connected to a pin on the FPGA. Driving a pin low (to 0) lights the corresponding segment. Figure 12 shows how the segments are arranged. Thus, to display a "1," drive the port to "1111001."

6.3 Keybounce

Like most switches, the buttons on the DE2 are a bunch of plastic designed to bring two pieces of metal together. When a button is depressed, the piece of metal shorts a wire to ground; otherwise, a resistor "pulls" the wire to a "1" voltage. So a "0" means the button is depressed and a "1" means it is not, so looking for when a button has just been pushed should amount to looking for a 1-to-0 transition.

Keybounce complicates this. Despite careful mechanical design, most buttons "bounce," meaning that they quickly connect and disconnect a few times before staying connected for a while. Thus, if you look for a 1-to-0 transition to indicate a button press, you can easily find many of them in a short time. Figure 13 illustrates the problem.

The solution comes from noting fingers are much slower than electronics; once a transition has occurred, the next genuine change can only occur, say, at least 200 ms later, so ignore any that come before then.

7 VHDL Simulation

For many reasons, hardware is tricker to design than software. For example, the usual edit-compile-debug cycle is longer because the hardware compiler has more details to consider. Another reason is that the behavior of hardware is harder to observe. It is difficult to put a print statement in hardware, but not impossible: designers often use LEDs as one-bit debugging outputs. It is even harder to probe a wire inside a chip.

One way out of this conundrum is to simulate VHDL before compiling it onto the FPGA. This is faster than compilation and makes it easy to observe everything going on inside your design, but can be thousands of times slower than running the actual hardware.

7.1 Testbenches and the Synthesizable Subset

There are actually two dialects of VHDL: the complete language, which the simulator accepts, and the synthesizable subset—what can be translated into hardware. The non-synthesizable part of the language is mostly useful for writing testbenches.

You need two things to run an interesting simulation of a system: a description of the system itself and some input for it. This latter component is known as a testbench and you need to write VHDL for your testbench when you simulate a design. A testbench instantiates the desing you are testing, stimulates the design, e.g., by applying clocks and inputs, and monitors its response. A test bench can be thought of as a signal generator and oscilloscope.

A testbench can use non-synthesizable VHDL statements. The *wait* statement, which can delay a precise amount of time, is typical. It is not possible to build hardware that does this, although you can build something that delays a precise number of clock cycles, but it is easily done in simulation. For example, *wait* can be used to provide a reset signal that goes low for 200 ns:

```
process
begin
    resetn <= '0';
    wait for 200 ns;
    resetn <= '1';
    wait;
end process;
```

The final *wait* stops the process so it does not automatically repeat and generate multiple resets.

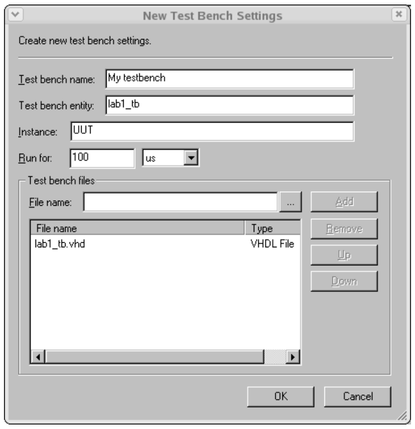
Wait is also useful for modeling clocks. Here is a way to generate a clock with a 40 ns period.

```
process
begin
    clock <= '0';
    wait for 20 ns;

    loop
        clock <= '1';
        wait for 20 ns;
        clock <= '0';
        wait for 20 ns;
    end loop;
end process;
```

The *loop* statement tells the simulation to generate clock pulses forever.

Wait can also be used to separate assignment statements to generate specific input stimulus.



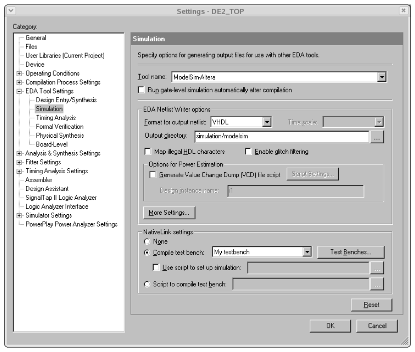
Create new test bench settings.

Test bench name: My testbench
Test bench entity: lab1_tb
Instance: UUT
Run for: 100 us

Test bench files
File name:

File name	Type
lab1_tb.vhd	VHDL File

Figure 14: Telling Quartus about a new test bench



Simulation

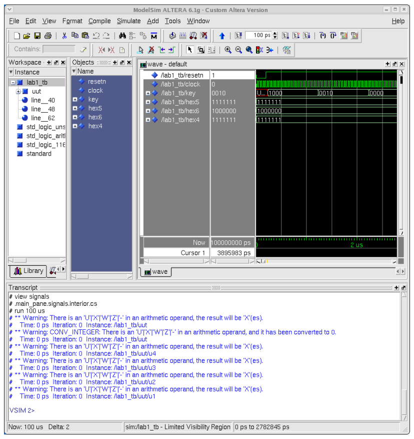
Specify options for generating output files for use with other EDA tools.

Tool name: ModelSim-Altera
☐ Run gate-level simulation automatically after compilation

EDA Netlist Writer options
Format for output netlist: VHDL   Time scale:
Output directory: simulation/modelsim
☐ Map illegal HDL characters   ☐ Enable glitch filtering
Options for Power Estimation
☐ Generate Value Change Dump (VCD) file script
Design instance name: i1

NativeLink settings
○ None
● Compile test bench: My testbench
☐ Use script to set up simulation:
○ Script to compile test bench:

Figure 15: Selecting ModelSim-Altera as the simulator and telling it about a testbench



```
/lab1_tb/resetn   1
/lab1_tb/clock    0
/lab1_tb/key      0010      U... 1000  0010  0000
/lab1_tb/hex5     1111111   1111111
/lab1_tb/hex6     1000000   1000000
/lab1_tb/hex4     1111111   1111111
Now       100000000 ps
Cursor 1  3895983 ps
```

```
# view signals
# .main_pane.signals.interior.cs
# run 100 us
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ps  Iteration: 0  Instance: /lab1_tb/uut
# ** Warning: CONV_INTEGER: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, and it has been converted to 0.
#    Time: 0 ps  Iteration: 0  Instance: /lab1_tb/uut
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ps  Iteration: 0  Instance: /lab1_tb/uut/u4
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ps  Iteration: 0  Instance: /lab1_tb/uut/u3
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ps  Iteration: 0  Instance: /lab1_tb/uut/u2
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 0 ps  Iteration: 0  Instance: /lab1_tb/uut/u1

VSIM 2>
```

```
Now: 100 us   Delta: 2   sim:/lab1_tb - Limited Visibility Region   0 ps to 2782845 ps
```

Figure 16: Running ModelSim and observing simulation results

```
process
begin
  wait for 100 ns;
  a   <= '0';
  b   <= '0';
  cin <= '0';
  wait for 20 ns;
  a   <= '1';
  b   <= '0';
  cin <= '0';
  wait for 20 ns;
  a   <= '1';
  b   <= '0';
  cin <= '1';
  wait;
end process;
```

You can test this lab by using this style of code to emulate buttons being pressed.

7.2 *Simulating your design*

Quartus can run an external VHDL simulator. We will use a version of Mentor Graphics's *ModelSim*. It is a hassle to run the simulator the first time, but it is much easier the second.

First, you probably need to tell Quartus where the simulator is. Go to Tools→Options, select "EDA Tool Options," double-click on the ModelSim-Altera line and enter the name of the directory in which the "vsim" executable resides. On our machines, this is /opt/altera/altera12.1/modelsim_ase/linuxaleom.

Now, tell Quartus that you want to use ModelSim-Altera as the "EDA simulator." With the project open, select Assignments→EDA tool settings and click on "Simulation." Set "Tool name" to "ModelSim-Altera."

The Assignments→EDA tool settings dialog is also where you must tell the simulator which testbench to use. Again under EDA Tool Settings→Simulation, specify a testbench in the "NativeLink settings" area by selecting "Compile test bench" and clicking on Test Benches.

In the Test Benches dialog, click New to create a new test bench. The name is arbitrary, but the entity name must match that in your VHDL test bench file and the instance should be the name of the instance of the design you are testing (e.g., "uut"). You must also specify an execution time for your testbench. This may be a number of μs. Finally, add the VHDL file for your testbench by selecting it and clicking "Add." See Figure 14.

Once you have created a new test bench, you can select it in the pulldown menu to the right of "Compile test bench." Figure 15 illustrates all of these settings.

Finally, you should be able to select Tools→EDA Simulation Tool→Run EDA RTL Simulation to start ModelSim. You need to have compiled your design before you start the simulation.

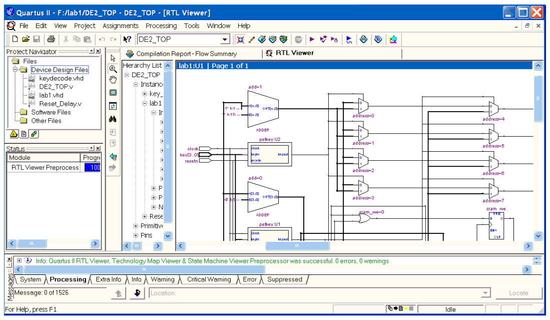
If all goes well, you should see the ModelSim window appear and a waveform viewer display the results of the simulation: Figure 16. Use the zoom tools to zoom in and out on this display and the scrollbars to move. By default, the display will show all the signals external to the unit under test (i.e., on the entity in your VHDL test bench file you specified earlier).

7.3 *The RTL Viewer*

We are designing a circuit but have been writing textual VHDL. Quartus includes an RTL viewer that displays your design as a schematic. Bring this up by selecting Tools→Netlist Viewers→RTL Viewer (Figure 17). Note that this is informative but not necessary for compilation.

8 What to turn in

Find an unsuspecting TA or instructor, show him/er your working memory reader/editor, your running simulation. Upload your working .vhd file on Courseworks.



Quartus II - F:/lab1/DE2_TOP - DE2_TOP - [RTL Viewer]

```
Info: Quartus II RTL Viewer, Technology Map Viewer & State Machine Viewer Preprocessor was successful. 0 errors, 0 warnings
```

Figure 17: Viewing your design as a schematic