The Spartan-3E
Tutorial 2:
Introduction to using the
PicoBlaze Microcontroller

Version 1.0

Author: Jasmine Banks

© 2012, Queensland University of Technology
Acknowledgements

Parts of this tutorial are based on an earlier version written for Project Navigator version 9.2, written by Michael Bakker, Matthew Grace and Warwick Kilroy, as part of ENB345 – Advanced Design in 2008.
Glossary

ALU   Arithmetic Logic Unit
DOS   Disk Operating System
FPGA  Field Programmable Gate Array
JTAG  Joint Test Action Group
LED   Light Emitting Diode
KCPSM3 (K)Constant Coded Programmable State Machine – a very simple 8-bit microcontroller optimised for Spartan-3 devices [2].
RISC  Reduced Instruction Set Computing
VHDL  VHSIC Hardware Description Language
VHSIC Very High Speed Integrated Circuit
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>3</td>
</tr>
<tr>
<td>Glossary</td>
<td>5</td>
</tr>
<tr>
<td>List of Figures</td>
<td>9</td>
</tr>
<tr>
<td>List of Tables</td>
<td>13</td>
</tr>
<tr>
<td><strong>1.0 Introduction</strong></td>
<td>15</td>
</tr>
<tr>
<td>1.1 Design Functionality</td>
<td>15</td>
</tr>
<tr>
<td>1.2 Relevant Documentation</td>
<td>15</td>
</tr>
<tr>
<td>1.3 Pre-requisite Knowledge</td>
<td>15</td>
</tr>
<tr>
<td>1.4 Scope</td>
<td>15</td>
</tr>
<tr>
<td><strong>2.0 Equipment</strong></td>
<td>17</td>
</tr>
<tr>
<td><strong>3.0 Background – The PicoBlaze Microcontroller</strong></td>
<td>19</td>
</tr>
<tr>
<td><strong>4.0 Procedure Part 1 – PicoBlaze</strong></td>
<td>21</td>
</tr>
<tr>
<td>4.1 PicoBlaze Download</td>
<td>21</td>
</tr>
<tr>
<td>4.2 Copy Files</td>
<td>21</td>
</tr>
<tr>
<td>4.3 Assembly Language Code</td>
<td>22</td>
</tr>
<tr>
<td>4.4 Running the Assembler</td>
<td>23</td>
</tr>
<tr>
<td><strong>5.0 Procedure Part 2 – Project Navigator</strong></td>
<td>31</td>
</tr>
<tr>
<td>5.1 Startup</td>
<td>31</td>
</tr>
<tr>
<td>5.2 Creating a New Project</td>
<td>32</td>
</tr>
<tr>
<td>5.3 Adding Source Files</td>
<td>35</td>
</tr>
<tr>
<td>5.4 tutorial.vhd and kcpsm3.vhd - Observations</td>
<td>38</td>
</tr>
<tr>
<td>5.5 Adding a top_level Entity</td>
<td>40</td>
</tr>
<tr>
<td>5.6 Editing the top_level Entity</td>
<td>45</td>
</tr>
<tr>
<td>5.7 top_level.vhd – Code</td>
<td>49</td>
</tr>
<tr>
<td>5.8 Syntax Checking</td>
<td>52</td>
</tr>
<tr>
<td>5.9 Pin Assignment</td>
<td>54</td>
</tr>
<tr>
<td>5.10 Synthesize, Translate, Map and Place &amp; Route</td>
<td>59</td>
</tr>
<tr>
<td>5.11 Download Design to Board</td>
<td>61</td>
</tr>
<tr>
<td><strong>6.0 Running the Program on the Spartan-3E Board</strong></td>
<td>75</td>
</tr>
<tr>
<td><strong>7.0 Further Information</strong></td>
<td>77</td>
</tr>
<tr>
<td><strong>8.0 References</strong></td>
<td>79</td>
</tr>
<tr>
<td>Appendix A – top_level.vhd</td>
<td>81</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 2.1</td>
<td>Spartan-3E Development Board</td>
<td>17</td>
</tr>
<tr>
<td>Figure 3.1</td>
<td>PicoBlaze components</td>
<td>19</td>
</tr>
<tr>
<td>Figure 3.2</td>
<td>KCPSM3 component declaration</td>
<td>20</td>
</tr>
<tr>
<td>Figure 3.3</td>
<td>Block Memory component declaration</td>
<td>20</td>
</tr>
<tr>
<td>Figure 4.1</td>
<td>KCPSM3 files after unzipping</td>
<td>21</td>
</tr>
<tr>
<td>Figure 4.2</td>
<td>Simple PicoBlaze program</td>
<td>22</td>
</tr>
<tr>
<td>Figure 4.3</td>
<td>Files in the working directory</td>
<td>22</td>
</tr>
<tr>
<td>Figure 4.4</td>
<td>KCPSM3 assembler files</td>
<td>23</td>
</tr>
<tr>
<td>Figure 4.5</td>
<td>DOS Command Prompt window, after changing to working directory</td>
<td>23</td>
</tr>
<tr>
<td>Figure 4.6</td>
<td>DOS Command Prompt window, with KCPSM3 command typed in</td>
<td>24</td>
</tr>
<tr>
<td>Figure 4.7</td>
<td>DOS Command Prompt window, after KCPSM3 successfully run</td>
<td>24</td>
</tr>
<tr>
<td>Figure 4.8</td>
<td>Error message which appears if KCPSM3 is run on a 64-bit machine</td>
<td>25</td>
</tr>
<tr>
<td>Figure 4.9</td>
<td>DOSBox window</td>
<td>26</td>
</tr>
<tr>
<td>Figure 4.10</td>
<td>DOSBox window, with KCPSM3 command typed in</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4.11</td>
<td>DOSBox window, after KCPSM3 successfully run</td>
<td>28</td>
</tr>
<tr>
<td>Figure 4.12</td>
<td>Files in the working directory after KCPSM3 successfully run</td>
<td>29</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Project Navigator Software Startup Window</td>
<td>31</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>New Project Wizard, Create New Project Page</td>
<td>32</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>New Project Wizard, Project Settings Page</td>
<td>33</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>New Project Wizard, Project Summary Page</td>
<td>34</td>
</tr>
<tr>
<td>Figure 5.5</td>
<td>Adding a source file to the project</td>
<td>35</td>
</tr>
<tr>
<td>Figure 5.6</td>
<td>Add Source file selection window</td>
<td>36</td>
</tr>
<tr>
<td>Figure 5.7</td>
<td>Adding Source Files window</td>
<td>36</td>
</tr>
<tr>
<td>Figure 5.8</td>
<td>kcpsm3 and tutorial in the Sources window</td>
<td>37</td>
</tr>
</tbody>
</table>
Figure 5.9: Source code for tutorial.vhd displayed in a tab .................................................. 38
Figure 5.10: tutorial entity ........................................................................................................... 38
Figure 5.11: kcpsm3 entity ......................................................................................................... 39
Figure 5.12: Adding a source file to the project ......................................................................... 40
Figure 5.13: New Source Wizard, Select Source Type ............................................................... 41
Figure 5.14: New Source Wizard, Define Module ....................................................................... 42
Figure 5.15: New Source Wizard, Summary .............................................................................. 43
Figure 5.16: top_level in the Sources window ......................................................................... 44
Figure 5.17: top_level.vhd, as displayed in Project Navigator, before editing .......................... 45
Figure 5.18(a): Architecture of top_level.vhd, part 1 ............................................................ 46
Figure 5.18(b): Architecture of top_level.vhd, part 2 ............................................................. 47
Figure 5.19: top_level in the Sources window ......................................................................... 48
Figure 5.20: Component declarations ....................................................................................... 49
Figure 5.21: Signal declarations ............................................................................................... 50
Figure 5.22: Component instantiations ...................................................................................... 50
Figure 5.23: Input ports .............................................................................................................. 51
Figure 5.24: Output ports .......................................................................................................... 51
Figure 5.25: Portion of Project Navigator screen with Synthesize – XST expanded ............... 52
Figure 5.26: A green tick next to Check Syntax shows that no errors were found .................. 53
Figure 5.27: Example where an error was purposely introduced .............................................. 53
Figure 5.28: Portion of Project Navigator screen, with User Constraints expanded ............... 55
Figure 5.29: Dialog Box asking if you wish to create an Implementation Constraint File ...... 55
Figure 5.30: Initial appearance of PlanAhead window ............................................................. 56
Figure 5.31: I/O Ports displayed in a separate window ............................................................. 56
List of Tables

<table>
<thead>
<tr>
<th>Table Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 5.1: Input/output ports of the top_level entity</td>
<td>54</td>
</tr>
<tr>
<td>Table 5.2: Values to enter in the I/O Ports window</td>
<td>57</td>
</tr>
</tbody>
</table>
1.0 Introduction

This tutorial is designed to help new users become familiar with using the PicoBlaze microcontroller with the Spartan-3E board. The tutorial gives a brief introduction to the PicoBlaze microcontroller, and then steps through the following:

- Writing a small PicoBlaze assembly language (.psm) file, and stepping through the process of assembling the .psm file using KCPSM3.
- Writing a top level VHDL module to connect the PicoBlaze microcontroller (KCPSM3 component) and the program ROM, and to connect the required input and output ports.
- Connecting the top level module inputs and outputs to the switches, buttons and LEDs on the Spartan-3E board.
- Downloading the program to the Spartan-3E board using the Project Navigator software.

1.1 Design Functionality

The code written in this tutorial reads the values of the four switches and the four push buttons, and displays the current values on the eight LEDs.

1.2 Relevant Documentation

Before commencing this tutorial, it would be helpful to download the Spartan-3E FPGA Starter Kit Board User Guide [1], and the PicoBlaze 8-bit Embedded Microcontroller User Guide [2].

1.3 Pre-requisite Knowledge

Before commencing this tutorial, the user should work through “The Spartan-3E Tutorial 1: Introduction to FPGA Programming” [3].

1.4 Scope

This tutorial is designed to help the user who is just starting to “get into” using the PicoBlaze with the Spartan-3E. It steps through the process of creating a very simple PicoBlaze program, running the assembler, putting the VHDL components together in Project Navigator, and downloading the final program to the board. It is not designed to be a tutorial on VHDL syntax or to provide detailed information on the PicoBlaze. For help with VHDL, the user can consult with a number of textbooks on the subject, such as [4,5], or find help online. The book by Chu [6] is also a useful reference for the Spartan-3 with many useful examples. Reference designs for the Spartan-3E can also be found here [7]. For detailed information about the features and instruction set of the PicoBlaze, the user can consult the documentation in [2,8].
2.0 Equipment

The following are required to work through this tutorial:

- The Xilinx ISE Project Navigator software. Version 14.3 was used in this tutorial, but older versions of the software can be used. The software can be downloaded with a free WebPack license from the Xilinx website, http://www.xilinx.com/. The user will need to register and log in.

- The Spartan-3E Starter Kit, including the Spartan-3E development board, power cable and USB cable for PC connection. The Spartan-3E development board is shown in Figure 2.1.

- The Picoblaze 8-bit Microcontroller software. The software can be downloaded for free from the Xilinx website, http://www.xilinx.com/. Again the user will need to register and log in.

- If a 64-bit machine is being used, software which can run 32-bit DOS programs, such as DOSBox, will be needed to run the KCPSM3 executable. DOSBox can be downloaded from http://www.dosbox.com/.

![Figure 2.1: Spartan-3E Development Board.](image)
3.0 Background – The PicoBlaze Microcontroller

The PicoBlaze is an 8-bit RISC microcontroller which is specifically designed and optimized for the Spartan-3 family. One of its main advantages is its small size, requiring only 96 FPGA slices. It is provided as a free, source-level VHDL file with royalty-free re-use within Xilinx FPGAs [2].

Figure 3.1 shows that the PicoBlaze consists of two components. The KCPSM3 component provides the ALU, registers, scratchpad RAM etc. The Block Memory (Program) component stores the instructions to be executed. This typically consists of a Block RAM, of 1024 bytes in size.

The basic design process using the PicoBlaze follows the steps below:

1. A PicoBlaze program is written in assembly language. This file is given the extension .psm.

2. The KCPSM3 assembler is run on the .psm file, and a VHDL file (extension .vhd) which embeds the instructions in the Block Memory component, is output. The name of the .vhd file will be derived from the name of the .psm file, i.e., if the .psm file is myprog.psm, then the .vhd file will be myprog.vhd.

3. The VHDL code for the Block Memory and KCPSM3 modules is loaded into Project Navigator. Further VHDL code will need to be written to connect the two modules and interface to the outside world.

4. The project is compiled using the Project Navigator Software, and ultimately downloaded to the Spartan-3E board (or other target hardware).

Figures 3.2 and 3.3 show the VHDL component declarations for the KCPSM3 and Block Memory respectively. Note that the name of the Block Memory component is derived from the name of the original .psm file, i.e., if the .psm file was myprog.psm, the Block Memory component will be called myprog.
In addition, it is possible to download a new program into the Block Memory, using the JTAG port on the Spartan-3E board. This can provide a convenient means to update the program without having to recompile the VHDL code in Project Navigator. This is not covered by this introductory tutorial, and the user can refer to documentation such as [3] for more information.
4.0 Procedure Part 1 – PicoBlaze

4.1 PicoBlaze Download

1. Download the file KCPSM3.zip from http://www.xilinx.com/. The version of the software for the Spartan-3 family should be chosen.

2. Unzip the file. After unzipping, the files should appear as shown in Figure 4.1.

![Figure 4.1: KCPSM3 files after unzipping.](image)


4.2 Copy Files

1. Create a directory called tutorial_2 in an appropriate location. This will be the working directory for the rest of this tutorial.

2. Copy the following files in the Assembler directory into tutorial_2:

   - KCPSM3.EXE
   - ROM_form.coe
   - ROM_form.v
   - ROM_form.vhd

3. Copy the following file in the VHDL directory into tutorial_2:

   - kcpsm3.vhd
4.3 Assembly Language Code

1. Open a text editor (for example, Notepad or Wordpad), and enter the text shown in Figure 4.2. The text consists of a very simple program written in the KCPSM3 assembly language.

The program runs in an infinite loop, reading the contents of an input port at address 00h (connected to the switches) into a register, and writing the contents of this register to an output port at address 80h (connected to the LEDs). Characters which appear after a “;” in each line are comments.

```
; Simple loop that puts contents of input register
; into the output register
;
; switches DSIN $00
; LEDS DSOUT $80

start: INPUT s0, 00 ; read switches into register s0
       OUTPUT s0, 80 ; write contents of s0 to output port 80 - leds.
       JUMP start ; loop back to start
```

Figure 4.2: Simple PicoBlaze program.

2. Save your file to tutorial.psm, in the tutorial_2 directory. If using Notepad, be careful not to save the file as tutorial.psm.txt. The name of the file should be restricted to 8 characters. The tutorial_2 directory should now contain the files shown in Figure 4.3.

Figure 4.3: Files in the tutorial_2 working directory.
4.4 Running the Assembler

As shown in Figure 4.4, the assembler takes the .psm file as input, as well as three Block RAM initialisation templates. Fifteen different output files are produced. In this tutorial, we will be using the .vhd output file.

The assembler is a DOS executable file, KCPSM3.exe, which can be run in a DOS Command Prompt window.

4.4.1 32-bit Operating Systems

1. Open a DOS Command Prompt window by selecting:

Start→All Programs→Accessories→Command Prompt

2. Use the cd command to change into the tutorial_2 working directory, as shown in Figure 4.5.
3. Now type the command **KCPSM3 tutorial.psm**, as shown in Figure 4.6.

![Figure 4.6: DOS Command Prompt window, with KCPSM3 command typed in.](image)

After entering the command **KCPSM3 tutorial.psm**, numerous messages should fly past on the screen, ending with “KCPSM3 successful. KCPSM3 complete”, as shown in Figure 4.7. After the assembler has successfully run, the working directory should contain many more files, as shown in Figure 4.12.

![Figure 4.7: DOS Command Prompt window, after KCPSM3 successfully run.](image)

4. Type **exit** to close the Command Prompt window.
4.4.2 64-bit Operating Systems

The KCPSM3 executable will only work on 32-bit operating systems. If you are using a 64-bit machine and attempt to run KCPSM3 in a DOS Command Prompt window, the error message shown in Figure 4.8 will appear.

![Figure 4.8: Error message which appears if it is attempted to run KCPSM3 on a 64-bit machine.](image)

One way to work around this and run KCPSM3 is to use the DOSbox software, which can be downloaded from [http://www.dosbox.com/](http://www.dosbox.com/).

1. Download and run DOSBox.

2. Mount the working directory and change into this directory. When DOSbox is started up, a command window which resembles the DOS Command Prompt window appears. However, it is first necessary to mount the working directory to a drive letter before being able to enter this directory and run programs. This is done with the `mount` command:

   ```
   mount <drive_letter> <directory>
   ```

   Figure 4.9 shows the commands entered to mount and change into the working directory. In this case, the working directory is mounted as drive letter `c`. The command `c:` is then used to change into this directory.
3. Now type the command **KCPSM3 tutorial.psm**, as shown in Figure 4.10.
After entering the command **KCPSM3 tutorial.psm**, numerous messages should fly past on the screen, ending with “KCPSM3 successful. KCPSM3 complete”, as shown in Figure 4.11. After the assembler has successfully run, the working directory should contain many more files, as shown in Figure 4.12.

![DOSBox window, with KCPSM3 command typed in.](image-url)
4. Type `exit` to close DOSBox.
Figure 4.12: Files in the working directory after KCPSM3 successfully run.
5.0 Procedure Part 2 – Project Navigator

5.1 Startup

Start the Project Navigator software by selecting:

Start → All Programs → XILINX Design Tools → Xilinx ISE Design Suite 14.3 → ISE Design Tools → 32 bit Project Navigator

or

Start → All Programs → XILINX Design Tools → Xilinx ISE Design Suite 14.3 → ISE Design Tools → 64 bit Project Navigator

depending on your system. The Xilinx Project Navigator software should start. The initial window which appears on startup should appear as shown in Figure 5.1.

![Project Navigator Software Startup Window](image)

Figure 5.1: Project Navigator Software Startup Window.
5.2 Creating a New Project


2. Type tutorial_2 in the Name: field.

3. Choose Location: and Working Directory: as the tutorial_2 working directory.

4. Verify that Top-level source type: is selected as HDL.

5. The properties should now be set as shown in Figure 5.2. Click Next to move to the Project Settings page.

![Figure 5.2: New Project Wizard, Create New Project Page.](image)

6. Fill in the properties as follows:

   - Evaluation Development Board: None Specified or Spartan-3E Starter Board
   - Product Category: All
   - Family: Spartan3E
   - Device: XC3S500E
• Package: FG320
• Speed Grade: -4
• Top-Level Source Type: HDL
• Synthesis Tool: XST (VHDL/Verilog)
• Simulator: ISim (VHDL/Verilog)
• Preferred Language: VHDL
• Property Specification in Project File: Store All Values
• Manual Compile Order: unchecked
• VHDL Source Analysis Standard: VHDL-93
• Enable Message Filtering: unchecked

Note if you choose Evaluation Development Board as Spartan-3E Started Board, properties from Product Category through to Speed will be filled in automatically. However, you must make sure that Preferred Language is set to VHDL.

The properties should now be filled in as shown in Figure 5.3.

![Figure 5.3: New Project Wizard, Project Settings Page.](image-url)
7. Click **Next** to move to the **Project Summary** page, which will appear as shown in Figure 5.4.

![Figure 5.4: New Project Wizard, Project Summary Page.](image)

8. Click **Finish** to exit the New Project Wizard.
5.3 Adding Source Files

1. Select **Project→Add Source** as shown in Figure 5.5. A window will appear allowing you to choose one or more files.

![Figure 5.5: Adding a source file to the project.](image)

2. Select **TUTORIAL.VHD** and **kcpsm3.vhd** as shown in Figure 5.6. Both files can be selected at once by clicking on the first filename, holding down the CTRL key and clicking the second filename. Alternatively, one file can be selected and steps 1-3 repeated for the second file.
3. The **Adding Source Files** window will now appear as shown in Figure 5.7, showing the two files selected to be added to the project. Click OK.

![Figure 5.6: Add Source file selection window.](image)

![Figure 5.7: Adding Source Files window.](image)
As shown in Figure 5.8, kcpsm3 and tutorial will now appear in the Sources window. Double-clicking on either filename in the Sources window will display the file in a tab.

![Image of Sources Window]

Figure 5.8: kcpsm3 and tutorial in the Sources window.
5.4 tutorial.vhd and kcpsm3.vhd – Observations

1. Double-click on tutorial in the Sources window. This will display the source code in a tab, as shown in Figure 5.9. It can be seen that Project Navigator colour codes the text of VHDL files, to make them easier to read. Comment lines, which start with “--” are displayed in green. Reserved words of the VHDL language are displayed in blue, while VHDL types are displayed in red. Everything else is left as black.

A close up of the code for the tutorial entity is shown in Figure 4.10. Note that this corresponds to the Block Memory (Program) component of Figures 3.1 and 3.3.

```vhdl
entity tutorial is
  Port ( address : in std_logic_vector(9 downto 0);
      instruction : out std_logic_vector(17 downto 0);
      clk : in std_logic);
end tutorial;
```

Figure 5.9: Source code for tutorial.vhd is displayed in a tab.

Figure 5.10: tutorial entity.
2. Double-click on `kcpsm3` in the Sources window, to display the source code for kcpsm3.vhd. A close up of the code for the `kspsm3` entity is shown in Figure 5.11. Note that this corresponds to the KCPSM3 block of Figures 3.1 and 3.2.

```
77 entity kcpsm3 is
78   Port (    address : out std_logic_vector(9 downto 0);
79                     instruction : in std_logic_vector(17 downto 0);
80                     port_id : out std_logic_vector(7 downto 0);
81                     write_strobe : out std_logic;
82                     read_strobe : out std_logic;
83                     in_port : in std_logic_vector(7 downto 0);
84                     interrupt : in std_logic;
85                     interrupt_ack : out std_logic;
86                     reset : in std_logic;
87                     clk : in std_logic);
89 end kcpsm3;
```

Figure 5.11: kcpsm3 entity.
5.5 Adding a top_level Entity

VHDL code still needs to be written to tie together the \texttt{kcpasm3} and \texttt{tutorial} entities, and also to interface with the Spartan-3E board. We will create a file called \texttt{top_level.vhd} for this purpose.

1. Select \textbf{Project}→\textbf{New Source} as shown in Figure 5.12. The \textbf{New Source Wizard} will appear.

![Figure 5.12: Adding a source file to the project.](image)

2. Select Source Type as \texttt{VHDL Module}.

3. Enter the file name as \texttt{top_level}, and enter the location of the file (same as the project location entered earlier).

4. Verify that the \textbf{Add to project} box is checked. The New Source Wizard should now appear as shown in Figure 5.13.
5. Click **Next** to go to the **Define Module** window.

6. Define the ports (inputs and outputs of the design) by entering the information as shown in Figure 5.14. These ports are described as follows:

   - switches will be an input consisting of 8 bits, and will be connected with the 4 slide switches and 4 push buttons on the Spartan-3E.
   - clk will be an input consisting of 1 bit, and will be connected to the clock input.
   - LEDs will be an output consisting of 8 bits, and will be connected with the LEDs on the Spartan-3E.
7. Click **Next** to move to the **Summary** page, as shown in Figure 5.15.

![Figure 5.14: New Source Wizard, Define Module.](image)

---

**Figure 5.14: New Source Wizard, Define Module.**

<table>
<thead>
<tr>
<th>Port Name</th>
<th>Direction</th>
<th>Bus</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>switches</td>
<td>in</td>
<td>7</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>clk</td>
<td>in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LEDs</td>
<td>out</td>
<td>7</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>in</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>in</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

7. Click **Next** to move to the **Summary** page, as shown in Figure 5.15.
9. Click **Finish** to exit the New Source Wizard.

As shown in Figure 5.16, **top_level** will now appear in the Sources window. Double-clicking on **top_level** in the Sources window will display the file, **top_level.vhd** in a tab.
Figure 5.16: top_level in the Sources window.
5.6 Editing the top_level Entity

1. Double-click on top_level in the Sources window to display the file, top_level.vhd in a tab. The code for top_level.vhd is shown in Figure 5.17.

```
library IEEE;
useIEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity top_level is
  Port ( switches : in STD_LOGIC_VECTOR (7 downto 0);
         clk : in STD_LOGIC;
         LEDs : out STD_LOGIC_VECTOR (7 downto 0));
end top_level;

architecture Behavioral of top_level is
begin
end Behavioral;
```

Figure 5.17: top_level.vhd, as displayed in Project Navigator, before editing.

The code in Figure 5.17 contains an entity and an architecture section. The entity section defines the inputs and outputs of this hardware block. In this case these have been automatically added using the New Source Wizard.

The architecture section still needs to be written for this module.

2. Replace the architecture block in Figure 5.17 with the code in Figure 5.18(a) and (b). This code is a trimmed down version of the Initial Design for the Spartan-3E FPGA Starter Kit Board (the original design shipped with the board), downloaded from [7].

For reference the complete code for top_level.vhd is listed in Appendix A. Note that where VDHL code is listed in this tutorial, the same colour coding as Project Navigator is used, to assist with readability.

3. Save the file by selecting File → Save from the main menu.
architecture Behavioral of top_level is
-- declaration of KCPSM3 (always use this declaration to call
-- up PicoBlaze core)
component kcpsm3
  port (address : out std_logic_vector(9 downto 0);
  instruction : in std_logic_vector(17 downto 0);
  port_id : out std_logic_vector(7 downto 0);
  write_strobe : out std_logic;
  out_port : out std_logic_vector(7 downto 0);
  read_strobe : out std_logic;
  in_port : in std_logic_vector(7 downto 0);
  interrupt : in std_logic;
  interrupt_ack : out std_logic;
  reset : in std_logic;
  clk : in std_logic);
end component;
--------------------------------------------------- ----------------------
-- declaration of program memory (here you will specify the entity name
-- as your .psm prefix name)
component tutorial
  port (address : in std_logic_vector(9 downto 0);
  instruction : out std_logic_vector(17 downto 0);
  clk : in std_logic);
end component;
--------------------------------------------------- ----------------------
-- Signals used to connect PicoBlaze core to program memory and I/O logic
signal address : std_logic_vector(9 downto 0);
signal instruction : std_logic_vector(17 downto 0);
signal port_id : std_logic_vector(7 downto 0);
signal out_port : std_logic_vector(7 downto 0);
signal in_port : std_logic_vector(7 downto 0);
signal write_strobe : std_logic;
signal read_strobe : std_logic;
signal interrupt_ack : std_logic;
signal reset : std_logic;
-- the following input is assigned an inactive value since it is
-- unused in this example
signal interrupt : std_logic := '0';
--------------------------------------------------- ----------------------
-- Start of circuit description
begin
-- Instantiating the PicoBlaze core
processor: kcpsm3
  port map (address => address,
  instruction => instruction,
  port_id => port_id,
  write_strobe => write_strobe,

Figure 5.18(a): Architecture of top_level.vhd, part 1.
```vhdl
out_port => out_port,
read_strobe => read_strobe,
in_port => in_port,
interrupt => interrupt,
interrupt_ack => interrupt_ack,
reset => reset,
clk => clk);

-- Instantiating the program memory
program: tutorial
port map (address => address,
instruction => instruction,
clk => clk);

-- Connect I/O of PicoBlaze

----- KCPSM3 Define input ports

-- The inputs connect via a pipelined multiplexer
input_ports: process(clk)
begin
if clk'event and clk='1' then
  case port_id(1 downto 0) is
  when "00" =>
    in_port <= switches;
  when others =>
    in_port <= "XXXXXXXX";
  end case;
end if;
end process input_ports;

----- KCPSM3 Define output ports

-- adding the output registers to the processor at address 80 hex
output_ports: process(clk)
begin
if clk'event and clk='1' then
  if port_id(7)='1' then
    LEADS <= out_port;
  end if;
end if;
end process output_ports;

end Behavioral;
```

Figure 5.18(b): Architecture of top_level.vhd, part 2.
After **top_level.vhd** has been edited and saved, it will now appear in the Sources window as shown in Figure 5.19. Since the **top_level** module uses the **kcpsm3** and **tutorial** modules as components, it has now moved above these modules in the hierarchy.

![Figure 5.19: top_level in the Sources window.](image)
5.7 top_level.vhd – Code

This section briefly outlines different parts of the code of the architecture block of top_level.vhd, and their functions.

At the start of the architecture block, the kcpsm3 and tutorial components that will be used are declared, as shown in Figure 5.20.

```vhdl
-- declaration of KCPSM3 (always use this declaration to call
-- up PicoBlaze core)
component kcpsm3
    port (address       : out std_logic_vector(9 downto 0);
          instruction   : in std_logic_vector(17 downto 0);
          port_id       : out std_logic_vector(7 downto 0);
          write_strobe  : out std_logic;
          out_port      : out std_logic_vector(7 downto 0);
          read_strobe   : out std_logic;
          in_port       : in std_logic_vector(7 downto 0);
          interrupt     : in std_logic;
          interrupt_ack : out std_logic;
          reset         : in std_logic;
          clk           : in std_logic);
end component;

---------------------------------------------------

-- declaration of program memory (here you will specify the entity name
-- as your .psm prefix name)
component tutorial
    port (address     : in std_logic_vector(9 downto 0);
          instruction : out std_logic_vector(17 downto 0);
          clk         : in std_logic);
end component;
```

Figure 5.20: Component declarations.

Next, the signals which connect the components of top_level, as well as connecting the inputs and outputs to the outside world, are declared in Figure 5.21.
The code in Figures 5.22 to 5.24 appears between the `begin` and `end` statements in the architecture block. First, the `kcpsm3` and `tutorial` components are instantiated, with ports connected to signals, as shown in Figure 5.22.

```
-- Signals used to connect PicoBlaze core to program memory and I/O logic
signal address : std_logic_vector(9 downto 0);
signal instruction : std_logic_vector(17 downto 0);
signal port_id : std_logic_vector(7 downto 0);
signal out_port : std_logic_vector(7 downto 0);
signal in_port : std_logic_vector(7 downto 0);
signal write_strobe : std_logic;
signal read_strobe : std_logic;
signal interrupt_ack : std_logic;
signal reset : std_logic;
signal interrupt : std_logic := '0';
```

Figure 5.21: Signal declarations.

The code in Figure 5.23 consists of a VHDL process. On each rising edge of the clock, if the current `port_id` is 00h, it copies `switches` into the `in_port` of the kcpsm3.

```
-- Instantiating the PicoBlaze core
processor: kcpsm3
    port map (address => address,
             instruction => instruction,
             port_id => port_id,
             write_strobe => write_strobe,
             out_port => out_port,
             read_strobe => read_strobe,
             in_port => in_port,
             interrupt => interrupt,
             interrupt_ack => interrupt_ack,
             reset => reset,
             clk => clk);

-- Instantiating the program memory
program: tutorial
    port map (address => address,
             instruction => instruction,
             clk => clk);
```

Figure 5.22: Component instantiations.
The code in Figure 5.24 consists of a VHDL process. On each rising edge of the clock, if the current `port_id` is 80h, it reads the `out_port` of the kcpsm3 into `LEDS`.

---

--- KCPSM3 Define output ports

---

```vhdl
-- adding the output registers to the processor at address 80 hex
output_ports: process(clk)
begin
  if clk'event and clk='1' then
    if port_id(7)='1' then
      LEDs <= out_port;
    end if;
  end if;
end process output_ports;
end Behavioral;
```

---

Figure 5.24: Output ports.

---

--- KCPSM3 Define input ports

---

```vhdl
-- The inputs connect via a pipelined multiplexer
input_ports: process(clk)
begin
  if clk'event and clk='1' then
    case port_id(1 downto 0) is
    when "00" =>
      in_port <= switches;
    when others =>
      in_port <= "XXXXXXXX";
    end case;
  end if;
end process input_ports;
```

---

Figure 5.23: Input ports.

The code in Figure 5.24 consists of a VHDL process. On each rising edge of the clock, if the current `port_id` is 80h, it reads the `out_port` of the kcpsm3 into `LEDS`.
5.8 Syntax Checking

Syntax checking can be done at this stage, to check that the VHDL code has been entered correctly. The following steps refer to the Project Navigator screen of Figure 5.25.

1. Verify that the **Implementation** check box toward the top left of the screen has been selected.
2. Verify that the **Design** tab has been selected.
3. Click on the ‘+’ next to **Synthesize – XST**. This will expand out to show various items, including **Check Syntax**.

![Figure 5.25: Portion of Project Navigator screen with Synthesize – XST expanded.](image)
4. Double-click on **Check Syntax**. After some time, a green tick should appear beside **Check Syntax**, as shown in Figure 5.26. If instead, a red cross appears, this means a syntax error has been found. Any errors should be fixed before proceeding. For example, in Figure 5.27, a syntax error has been purposely made, in that a space has been left between “LED” and “S” in “LEDS”. If a syntax error has been found, an error message should appear in the **Console Window**, which should assist with diagnosing and fixing the problem (see Figure 5.27).

Figure 5.26: A green tick next to Check Syntax shows that no errors were found.

Figure 5.27: Example where an error was purposely introduced. A red cross next to Check Syntax indicates that an error was found.
5.9 Pin Assignment

Recall that the VHDL code for the top_level entity is:

```vhdl
entity top_level is
  Port ( switches : in STD_LOGIC_VECTOR (7 downto 0);
        clk : in STD_LOGIC;
        LEDs : out STD_LOGIC_VECTOR (7 downto 0));
end top_level;
```

We wish to connect the inputs and outputs of the top_level entity to the switches, buttons, LEDs and clock on the Spartan-3E board. For example, we wish to read inputs switches(0-3) from the four slider switches. These are connected to pins L13, L14, H18 and N17 of the FPGA chip. Similarly, we wish to read inputs switches(4-7) from the four push buttons on the board. The North, South, East and West push buttons are connected to pins V4, K17, H13 and D18 of the FPGA. Also, we wish to connect the output LEDs(0-7) to the eight LEDs on the board, corresponding to pins F12, E12, E11, F11, C11, D11, E9 and F9 respectively. Finally, we wish to connect the clk input to the 50MHz on board clock, which is pin C9 of the FPGA.

For each input and output port of the top_level entity, Table 5.1 lists the name of the device on the Spartan-3E board that we wish to connect the port to, a description of what it physically corresponds to (clock, slider switch, push button or LED), and the FPGA pin number that it is connected to. This information comes from the Spartan-3E FPGA Starter Kit Board User Guide [1].

<table>
<thead>
<tr>
<th>Port name</th>
<th>Spartan-3E board device name</th>
<th>Description</th>
<th>FPGA pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>clk</td>
<td>CLK_50MHz</td>
<td>Clock</td>
<td>C9</td>
</tr>
<tr>
<td>switches(0)</td>
<td>SW0</td>
<td>Slider switch</td>
<td>L13</td>
</tr>
<tr>
<td>switches(1)</td>
<td>SW1</td>
<td>Slider switch</td>
<td>L14</td>
</tr>
<tr>
<td>switches(2)</td>
<td>SW2</td>
<td>Slider switch</td>
<td>H18</td>
</tr>
<tr>
<td>switches(3)</td>
<td>SW3</td>
<td>Slider switch</td>
<td>N17</td>
</tr>
<tr>
<td>switches(4)</td>
<td>BTN_North</td>
<td>Push button</td>
<td>V4</td>
</tr>
<tr>
<td>switches(5)</td>
<td>BTN_South</td>
<td>Push button</td>
<td>K17</td>
</tr>
<tr>
<td>switches(6)</td>
<td>BTN_East</td>
<td>Push button</td>
<td>H13</td>
</tr>
<tr>
<td>switches(7)</td>
<td>BTN_West</td>
<td>Push button</td>
<td>D18</td>
</tr>
<tr>
<td>LEDs(0)</td>
<td>LD0</td>
<td>LED</td>
<td>F12</td>
</tr>
<tr>
<td>LEDs(1)</td>
<td>LD1</td>
<td>LED</td>
<td>E12</td>
</tr>
<tr>
<td>LEDs(2)</td>
<td>LD2</td>
<td>LED</td>
<td>E11</td>
</tr>
<tr>
<td>LEDs(3)</td>
<td>LD3</td>
<td>LED</td>
<td>F11</td>
</tr>
<tr>
<td>LEDs(4)</td>
<td>LD4</td>
<td>LED</td>
<td>C11</td>
</tr>
<tr>
<td>LEDs(5)</td>
<td>LD5</td>
<td>LED</td>
<td>D11</td>
</tr>
<tr>
<td>LEDs(6)</td>
<td>LD6</td>
<td>LED</td>
<td>E9</td>
</tr>
<tr>
<td>LEDs(7)</td>
<td>LD7</td>
<td>LED</td>
<td>F9</td>
</tr>
</tbody>
</table>

Table 5.1: Input/output ports of the top_level entity; and the name, description and FPGA pin no. of the devices on the Spartan-3E board that the ports will be connected to.

The following steps are used to connect the inputs and outputs to the switches, buttons and LEDs on the Spartan-3E board:
1. As shown in Figure 5.28, click on the ‘+’ next to **User Constraints**. This will expand out to show various items, including **I/O Pin Planning (PlanAhead) – Pre-Synthesis**.

![Figure 5.28: Portion of Project Navigator screen, with User Constraints expanded.](image)

2. Double-click on **I/O Pin Planning (PlanAhead) – Pre-Synthesis**. The first time this is done, the window of Figure 5.29 will appear, asking whether it is OK to create a UCF file. Click **Yes**.

![Figure 5.29: Dialog Box asking if you wish to create an Implementation Constraint File.](image)

3. After clicking **Yes** in Figure 5.29, the **PlanAhead** window of Figure 5.30 will appear. Click on the **I/O Ports** tab, and then the **float frame** icon. This displays the I/O Ports in a separate window, as shown in Figure 5.31.
4. Click on the ‘+’ next to **LEDs(8)**, **switches(8)** and **Scalar ports(1)** in Figure 5.31. This will display all individual input/outputs as shown in Figure 5.32.

**Figure 5.30: Initial appearance of PlanAhead window.**

**Figure 5.31: I/O Ports displayed in a separate window.**
5. Enter the Site, I/O Std, Drive Strength, Slew Type and Pull Type columns, with the values given in Table 5.2. The Bank and Vcco columns will be filled in automatically when the Site and I/O Std columns respectively are filled in.

<table>
<thead>
<tr>
<th>Port</th>
<th>Site</th>
<th>I/O Std</th>
<th>Drive Strength</th>
<th>Slew Type</th>
<th>Pull Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEDs(7)</td>
<td>F9</td>
<td>LVTTL</td>
<td>8</td>
<td>SLOW</td>
<td></td>
</tr>
<tr>
<td>LEDs(6)</td>
<td>E9</td>
<td>LVTTL</td>
<td>8</td>
<td>SLOW</td>
<td></td>
</tr>
<tr>
<td>LEDs(5)</td>
<td>D11</td>
<td>LVTTL</td>
<td>8</td>
<td>SLOW</td>
<td></td>
</tr>
<tr>
<td>LEDs(4)</td>
<td>C11</td>
<td>LVTTL</td>
<td>8</td>
<td>SLOW</td>
<td></td>
</tr>
<tr>
<td>LEDs(3)</td>
<td>F11</td>
<td>LVTTL</td>
<td>8</td>
<td>SLOW</td>
<td></td>
</tr>
<tr>
<td>LEDs(2)</td>
<td>E11</td>
<td>LVTTL</td>
<td>8</td>
<td>SLOW</td>
<td></td>
</tr>
<tr>
<td>LEDs(1)</td>
<td>E12</td>
<td>LVTTL</td>
<td>8</td>
<td>SLOW</td>
<td></td>
</tr>
<tr>
<td>LEDs(0)</td>
<td>F12</td>
<td>LVTTL</td>
<td>8</td>
<td>SLOW</td>
<td></td>
</tr>
<tr>
<td>switches(7)</td>
<td>D18</td>
<td>LVTTL</td>
<td></td>
<td>PULLDOWN</td>
<td></td>
</tr>
<tr>
<td>switches(6)</td>
<td>H13</td>
<td>LVTTL</td>
<td></td>
<td>PULLDOWN</td>
<td></td>
</tr>
<tr>
<td>switches(5)</td>
<td>K17</td>
<td>LVTTL</td>
<td></td>
<td>PULLDOWN</td>
<td></td>
</tr>
<tr>
<td>switches(4)</td>
<td>V4</td>
<td>LVTTL</td>
<td></td>
<td>PULLDOWN</td>
<td></td>
</tr>
<tr>
<td>switches(3)</td>
<td>N17</td>
<td>LVTTL</td>
<td></td>
<td>PULLUP</td>
<td></td>
</tr>
<tr>
<td>switches(2)</td>
<td>H18</td>
<td>LVTTL</td>
<td></td>
<td>PULLUP</td>
<td></td>
</tr>
<tr>
<td>switches(1)</td>
<td>L14</td>
<td>LVTTL</td>
<td></td>
<td>PULLUP</td>
<td></td>
</tr>
<tr>
<td>switches(0)</td>
<td>L13</td>
<td>LVTTL</td>
<td></td>
<td>PULLUP</td>
<td></td>
</tr>
<tr>
<td>clk</td>
<td>C9</td>
<td>LVCMOS33</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 5.2: Values to enter in the I/O Ports window.

The I/O Ports window should now appear as shown in Figure 5.33.
6. Click the **Save** button in the PlanAhead window (location of Save button is shown in Figure 5.30), to save the entered pins. The PlanAhead window can be closed at this stage.
5.10 Synthesize, Translate, Map, and Place & Route

The next stage involves going through the Synthesize, Translate, Map and Place and Route Steps. These steps are carried out by the Project Navigator software, and are briefly described as follows:

- Synthesize: generates netlists for each source file.
- Translate: merges multiple files into a single netlist.
- Map: the design is mapped to slices and I/O blocks.
- Place and Route: works out how the design is to be placed on the chip and components connected.

1. As shown in Figure 5.34, click on the ‘+’ next to Implement Design. This will expand out to show the Translate, Map and Place & Route stages.

![Figure 5.34: Portion of Project Navigator screen, with Implement Design expanded.](image-url)
2. Double-click on Implement Design. This will first cause Synthesize – XST to run. Next, Translate, Map and Place & Route will run in turn. As each stage is completed, a green tick will appear next to it. If there are warning messages at any stage, a yellow icon with an exclamation mark will appear. The user can inspect these warning messages and decide whether they are critical to the design. After all three stages are complete, the Project Navigator screen will appear as shown in Figure 5.35.

![Figure 5.35: Portion of Project Navigator screen, with Implement Design expanded, after Translate, Map and Place & Route have successfully been run.](image-url)
5.11 Download Design to Board

The next steps involve generating the program file, and downloading it to the Spartan-3E board using iMPACT.

1. As shown in Figure 5.36, click on the ‘+’ next to **Configure Target Device**. This will expand out to show the **Manage Configuration Project (iMPACT)** option.

![Figure 5.36: Portion of Project Navigator screen, with Implement Design expanded.](image-url)
2. Double-click on **Generate Programming File**. When this has successfully run, a green tick will appear next to **Generate Programming File** as shown in Figure 5.37.

3. Connect the power cable and the USB cable to the Spartan-3E board (refer to Figure 2.1 for locations of the power and USB plugs). Plug the USB cable from the Spartan-3E into the PC, and make sure the Spartan-3E board is switched on.

![Diagram of Project Navigator](image)

**Figure 5.37: Portion of Project Navigator screen, after Generate Programming File has successfully been run.**
4. Double-click on **Manage Configuration Project (iMPACT)**. The iMPACT window should appear as shown in Figure 5.38.

![iMPACT Window](image)

*Figure 5.38: The initial iMPACT window.*
5. Double-click on Boundary Scan as shown in Figure 5.39. The message “Right click to Add Device or Initialize JTAG Chain” should appear to the right.

Figure 5.39: iMPACT window, after double-clicking on Boundary Scan.
6. Right click on the text “Right click to Add Device or Initialize JTAG Chain”, and select **Initialise Chain**, as shown in Figure 5.40.

---

Figure 5.40: iMPACT window, showing Initialize Chain selected.
7. After a while, a picture of a “chain” should appear, along with the message **Identify Succeeded** in a blue box (Figure 5.41). The first chip, the **xc3s500e**, is the FPGA chip that we wish to program. The other two, **xcf04s** and **xc2c64a**, are other chips on the board that will be bypassed.

A dialog box, asking “Do you wish to continue and select configuration file(s)?” will appear, as shown in Figure 5.41. Click **Yes**.

![Figure 5.41: iMPACT window, assign configuration files.](image)
8. The **Assign New Configuration File** window will appear (Figure 5.42). Select the file “top_level.bit”, and click **Open**. This associates the file top_level.bit with the xc3s500e.

![Figure 5.42: iMPACT window, assigning the configuration file for the xc3e500e.](image-url)
9. A message stating “This device supports attached flash PROMs. Do you want to attach an SPI or BPI PROM to this device?” will appear (Figure 5.43). This is not needed for this design. Click No.

![Image](image-url)

Figure 5.43: iMPACT window, dialog box asking if we wish to attach an SPI or BPI PROM.
10. The **Assign New Configuration File** window will appear again (Figure 5.44). In this case click **Bypass**. This ensures that the xcf04s is bypassed.

![Figure 5.44: iMPACT window, bypassing the xcf04s.](image)

**Figure 5.44:** iMPACT window, bypassing the xcf04s.
11. The **Assign New Configuration File** window will appear yet again (Figure 5.45). Again click **Bypass**. This ensures that the xc2c64a is bypassed.

![Figure 5.45: iMPACT window, bypassing the xc2c64a.](image)
12. A window entitled **Device Programming Properties – Device 1 Programming Properties** will appear (Figure 5.46). Click **OK**.

![Figure 5.46: iMPACT window, Device Programming Properties dialog box.](image-url)
13. The iMPACT window should now appear as shown in Figure 5.47. Right click on the xc3e500e chip (Figure 5.48) and select **Program**.

![Figure 5.47: iMPACT window, showing the device chain.](image1)

**Figure 5.47: iMPACT window, showing the device chain.**

![Figure 5.48: iMPACT window, options which appear when right clicking on the xc3s500e.](image2)

**Figure 5.48: iMPACT window, options which appear when right clicking on the xc3s500e.**
14. The program should now be downloaded to the Spartan-3E board. After the download is complete, the message “Program Succeeded” will appear in a blue box (Figure 5.49).

Figure 5.49: iMPACT window, after the program has been successfully downloaded to the Spartan-3E board.
6.0 Running the Program on the Spartan-3E Board

The Spartan-3E board after downloading the program is shown in Figure 6.1. Note that the appearance of the LCD screen may differ from what is shown. The LCD screen will normally continue to display whatever was being displayed at the instant the new program was downloaded. The LCD screen is not used in this tutorial.

![Image of Spartan-3E board with program running]

Figure 6.1: The Spartan-3E board with the program running.

Now the program can be tested on the Spartan-3E. For the slider switches, SW0, SW1, SW2 and SW3, the switch in the UP position indicates that the switch is on. LD0-LD3 should switch on when SW0-SW3 are on. LD4-LD7 should switch on when BTN_North, BTN_South, BTN_East and BTN_West are pressed.
7.0 Further Information

For further information about this tutorial, please contact:

Dr. Jasmine Banks  
School of Electrical Engineering and Computer Science  
Queensland University of Technology  
GPO Box 2434, Brisbane 4001,  
AUSTRALIA  
Email: j.banks@qut.edu.au or jbanks@ieee.org.
7.0 References

[1] Spartan-3E FPGA Starter Kit Board User Guide, Online: 


[7] Spartan-3E FPGA Starter Kit Board Design Examples, Online: 

Appendix A – top_level.vhd

-- Company:
-- Engineer:
--
-- Create Date: 15:42:06 12/03/2012
-- Design Name:
-- Module Name: top_level - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
-- use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
-- library UNISIM;
-- use UNISIM.VComponents.all;

entity top_level is
  port ( switches : in  STD_LOGIC_VECTOR (7 downto 0);
         clk       : in   STD_LOGIC;
         LEDs      : out  STD_LOGIC_VECTOR (7 downto 0));
end top_level;

architecture Behavioral of top_level is

-- declaration of KCPSM3 (always use this declaration to call up PicoBlaze
-- core)
component kcpsm3
  port ( address           : out std_logic_vector(9 downto 0);
         instruction      : in std_logic_vector(17 downto 0);
         port_id           : out std_logic_vector(7 downto 0);
         write_strobe      : out std_logic;
         out_port          : out std_logic_vector(7 downto 0);
         read_strobe       : out std_logic;
         in_port           : in std_logic_vector(7 downto 0));
end kcpsm3;
interrupt : in std_logic;
interrupt_ack : out std_logic;
reset : in std_logic;
clk : in std_logic);
end component;

-- declaration of program memory (here you will specify the entity name
-- as your .psm prefix name)
component tutorial
port (address : in std_logic_vector(9 downto 0);
instruction : out std_logic_vector(17 downto 0);
--proc_reset : out std_logic; --Used by JTAG Loader
clk : in std_logic);
end component;

-- Signals used to connect PicoBlaze core to program memory and I/O logic
signal address : std_logic_vector(9 downto 0);
signal instruction : std_logic_vector(17 downto 0);
signal port_id : std_logic_vector(7 downto 0);
signal out_port : std_logic_vector(7 downto 0);
signal in_port : std_logic_vector(7 downto 0);
signal write_strobe : std_logic;
signal read_strobe : std_logic;
signal interrupt_ack : std_logic;
signal reset : std_logic;
-- the following input is assigned an inactive value since it is
-- unused in this example
signal interrupt : std_logic := '0';

-- Start of circuit description
begin
-- Instantiating the PicoBlaze core
processor: kcpsm3
port map (address => address,
instruction => instruction,
port_id => port_id,
write_strobe => write_strobe,
out_port => out_port,
read_strobe => read_strobe,
in_port => in_port,
interrupt => interrupt,
interrupt_ack => interrupt_ack,
reset => reset,
clk => clk);

-- Instantiating the program memory
program: tutorial
port map (address => address,
instruction => instruction,
--proc_reset => reset, --Used by JTAG Loader
clk => clk);

-- Connect I/O of PicoBlaze
--------------------------------------------------- ------------------------
-- KCPSM3 Define input ports
---------------------------------------------------------------------------
-- The inputs connect via a pipelined multiplexer
input_ports: process(clk)
beg
if clk'event and clk='1' then
case port_id(1 downto 0) is
  -- read simple toggle switches and buttons at address 00 hex
  when "00" =>
in_port <= switches;
  -- Don't care used for all other addresses to ensure minimum
  -- logic implementation
  when others =>
in_port <= "XXXXXXXX";
end case;
end if;
end process input_ports;

-- KCPSM3 Define output ports
---------------------------------------------------------------------------
-- adding the output registers to the processor at address 80 hex
output_ports: process(clk)
beg
if clk'event and clk='1' then
  if port_id(7)='1' then
    LEDS <= out_port;
  end if;
end if;
end process output_ports;

end Behavioral;