



Queensland University of Technology
Brisbane Australia

This is the author's version of a work that was submitted/accepted for publication in the following source:

Banks, Jasmine (2013) *The Spartan-3E tutorial 3 : using the LCD display [version 1.0]*. Queensland University of Technology. (Unpublished)

This file was downloaded from: <http://eprints.qut.edu.au/58139/>

© Copyright 2012 Queensland University of Technology

Notice: *Changes introduced as a result of publishing processes such as copy-editing and formatting may not be reflected in this document. For a definitive version of this work, please refer to the published source:*

The Spartan-3E

Tutorial 3:

Using the LCD Display

Version 1.0

Author: Jasmine Banks



© 2013, Queensland University of Technology

Acknowledgements

Parts of this tutorial are based on an earlier version written for Project Navigator version 9.2, written by Matthew Dagg, Stephan Trauden and Matthew Watson, as part of ENB345 – Advanced Design in 2010.

Glossary

ALU	Arithmetic Logic Unit
DOS	Disk Operating System
FPGA	Field Programmable Gate Array
JTAG	Joint Test Action Group
LED	Light Emitting Diode
LCD	Liquid Crystal Display
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

	page
Acknowledgements	3
Glossary	5
List of Figures	9
List of Tables	13
1.0 Introduction	15
1.1 Relevant Documentation	15
1.2 Pre-requisite Knowledge	15
1.3 Scope and Further help.....	15
2.0 Equipment	17
3.0 The PicoBlaze Microcontroller	19
3.1 PicoBlaze Overview.....	19
3.2 VHDL Components and Design Process.....	20
3.3 PicoBlaze Interface Signals	22
3.4 The PicoBlaze Instruction Set.....	23
4.0 The LCD Display.....	25
4.1 The LCD Display Overview.....	25
4.2 LCD Memory Map.....	26
4.3 LCD Controller Command Set.....	28
4.4 PicoBlaze Code for LCD Control.....	31
5.0 Procedure – File Setup.....	39
5.1 PicoBlaze Download.....	39
5.2 Copy Files.....	39
5.3 Setup hello.psm.....	40
6.0 Procedure Part A – Hello World on the Spartan3E LCD.....	41
6.1 PicoBlaze Code for Hello World.....	41
6.2 Running the Assembler.....	42
6.3 Starting Project Navigator.....	49
6.4 Creating a New Project.....	50
6.5 Adding Source Files.....	54
6.6 hello.vhd and kcpsm3.vhd – Observations.....	57
6.7 Adding a top_level Entity.....	59
6.8 Editing the top_level Entity.....	64

6.9 top_level.vhd – Code	70
6.10 Syntax Checking	77
6.11 Pin Assignment	78
6.12 Synthesize, Translate, Map and Place & Route	85
6.13 Download Design to Board	87
6.14 Running the Program on the Spartan-3E Board	100
7.0 Procedure Part B – Creating Custom Characters.....	101
7.1 Edit the .psm file.....	101
7.2 Running the Assembler.....	105
7.3 Project Navigator.....	105
7.4 Running the Program on the Spartan-3E Board	106
8.0 Procedure Part C – Flashing and Shifting.....	107
8.1 Edit the .psm file.....	107
8.2 Running the Assembler.....	111
8.3 Project Navigator.....	111
8.4 Running the Program on the Spartan-3E Board	115
9.0 Further Information	117
10.0 References	119
Appendix A.....	121
A.1 hello.psm.....	121
A.2 top_level.vhd.....	130
Appendix B – hello.psm	135
Appendix C.....	145
C.1 hello.psm.....	145
C.2 top_level.vhd.....	156

List of Figures

	page
Figure 2.1: Spartan-3E Development Board	17
Figure 3.1: PicoBlaze microncontroller block diagram.....	20
Figure 3.2: PicoBlaze VHDL components	20
Figure 3.3: KCPSM3 component declaration	21
Figure 3.4: Block Memory component declaration	21
Figure 4.1: Character LCD interface.....	25
Figure 4.2: DD RAM Hexadecimal addresses.....	26
Figure 4.3: LCD Character set.....	27
Figure 4.4: PicoBlaze Assembler code for LCD control.....	32
Figure 5.1: KCPSM3 files after unzipping.....	39
Figure 6.1: Main program for hello.psm.....	41
Figure 6.2: KCPSM3 assembler files	42
Figure 6.3: DOS Command Prompt window.....	43
Figure 6.4: DOS Command Prompt window, with KCPSM3 command typed in	43
Figure 6.5: DOS Command Prompt window, after KCPSM3 successfully run	44
Figure 6.6: Error message which appears if KCPSM3 is run on a 64-bit machine	44
Figure 6.7: DOSBox window	45
Figure 6.8: DOSBox window, with KCPSM3 command typed in	46
Figure 6.9: DOSBox window, after KCPSM3 successfully run	47
Figure 6.10: Files in the working directory after KCPSM3 successfully run	48
Figure 6.11: Project Navigator Software Startup Window	49
Figure 6.12: New Project Wizard, Create New Project Page	50
Figure 6.13: New Project Wizard, Project Settings Page	52
Figure 6.14: New Project Wizard, Project Summary Page	53

Figure 6.15: Adding a source file to the project	54
Figure 6.16: Add Source file selection window	55
Figure 6.17: Adding Source Files window	55
Figure 6.18: kcpsm3 and hello in the Sources window	56
Figure 6.19: Source code for hello.vhd displayed in a tab	57
Figure 6.20: hello entity	57
Figure 6.21: kcpsm3 entity	58
Figure 6.22: Adding a source file to the project	59
Figure 6.23: New Source Wizard, Select Source Type	60
Figure 6.24: New Source Wizard, Define Module	61
Figure 6.25: New Source Wizard, Summary	62
Figure 6.26: top_level in the Sources window	63
Figure 6.27: top_level.vhd, as displayed in Project Navigator, before editing	64
Figure 6.28: entity and architecture for top_level.vhd.....	65
Figure 6.29: top_level in the Sources window	69
Figure 6.30: top_level entity.....	70
Figure 6.31: Component declarations	71
Figure 6.32: Signal declarations	72
Figure 6.33: LCD control.....	73
Figure 6.34: Component instantiations	74
Figure 6.35: Input ports	75
Figure 6.36: Output ports	76
Figure 6.37: Portion of Project Navigator screen with Synthesize – XST expanded	77
Figure 6.38: A green tick next to Check Syntax shows that no errors were found	78
Figure 6.39: Portion of Project Navigator screen, with User Constraints expanded	80
Figure 6.40: Dialog Box asking if you wish to create an Implementation Constraint File.....	80

Figure 6.41: Initial appearance of PlanAhead window	81
Figure 6.42: I/O Ports displayed in a separate window	81
Figure 6.43: I/O Ports window with individual ports expanded	82
Figure 6.44: I/O Ports window with values filled in	84
Figure 6.45: Portion of Project Navigator screen, with Implement Design expanded	85
Figure 6.46: Portion of Project Navigator screen, after Translate, Map and Place & Route have successfully been run	86
Figure 6.47: Portion of Project Navigator screen, with Implement Design expanded	87
Figure 6.48: Portion of Project Navigator screen, after Generate Programming File has successfully been run	88
Figure 6.49: The initial iMPACT window	89
Figure 6.50: iMPACT window, after double-clicking on Boundary Scan	90
Figure 6.51: iMPACT window, showing Initialize Chain selected	91
Figure 6.52: iMPACT window, assign configuration files	92
Figure 6.53: iMPACT window, assigning the configuration file for the xc3e500e	93
Figure 6.54: iMPACT window, dialog box asking if we wish to attach an SPI or BPI PROM .	94
Figure 6.55: : iMPACT window, bypassing the xcf04s	95
Figure 6.56: iMPACT window, bypassing the xc2c64a	96
Figure 6.57: iMPACT window, Device Programming Properties dialog box	97
Figure 6.58: iMPACT window, showing the device chain	98
Figure 6.59: iMPACT window, options which appear when right clicking on the xc3s500e ..	98
Figure 6.60: iMPACT window, after the program has been successfully downloaded to the Spartan-3E board	99
Figure 6.61: The Spartan-3E board with the program running for Part A.....	100
Figure 7.1: Main program loop for Part B.....	101
Figure 7.2: Disp_mesg subroutine for Part B.....	101
Figure 7.3: Disp_cust_chars subroutine for Part B.....	102
Figure 7.4: Setup_cust_chars subroutine for Part B.....	103

Figure 7.5: Locate Missing Source Files window.....	105
Figure 7.6: The Spartan-3E board with the program running for Part B.....	106
Figure 8.1: Delay constants for shifting.....	107
Figure 8.2: Main program loop for Part C.....	108
Figure 8.3: Disp_mesg subroutine for Part C.....	109
Figure 8.4: Interrupt service routine and ADDRESS directive.....	110
Figure 8.5: Library declarations to add.....	111
Figure 8.6: Signals used for interrupts.....	112
Figure 8.7: Processes associated with interrupt control.....	113
Figure 8.8: The Spartan-3E board with the program running for Part C.....	115

List of Tables

	Page
Table 3.1: PicoBlaze interface signals.....	22
Table 3.2: PicoBlaze instruction set.....	23
Table 4.1: Example custom character at CG RAM location 0x03.....	28
Table 4.2: LCD Character display command set.....	28
Table 4.3: Explanation of code for Figure 4.5.....	31
Table 6.1: Input/output ports of the top_level entity.....	79
Table 6.2: Values to enter in the I/O Ports window	83
Table 7.1: Example custom character stored in CG RAM address 0x00.....	104
Table 7.2: Example custom character stored in CG RAM address 0x01.....	104

1.0 Introduction

This tutorial is designed to assist users who wish to use the LCD screen on the Spartan-3E board. In this tutorial, the PicoBlaze microcontroller is used to control the LCD.

The tutorial is organised into three Parts. In Part A, code is written to display the message “Hello World” on the LCD. Part B demonstrates how to define and display custom characters. Finally, Part C shows how the display can be shifted and flashed. Shifting is done by using a delay in the main PicoBlaze program loop, while flashing is done using the PicoBlaze interrupt. The slider switches can be used to select the shifting direction, and to turn shifting and flashing on and off.

Each part of the tutorial steps through the following:

- Writing a PicoBlaze assembly language (.psm) file, and assembling the .psm file using KCPSM3.
- Writing a top level VHDL module to connect the PicoBlaze microcontroller (KCPSM3 component) and the program ROM, connect the required input and output ports, and implement any extra functionality that is required.
- Connecting the top level module inputs and outputs to the components on the Spartan-3E board.
- Downloading the program to the Spartan-3E board using the Project Navigator software.

1.1 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].

1.2 Pre-requisite Knowledge

Although this tutorial can be worked through on its own, it would be helpful if the user has already worked through “The Spartan-3E Tutorial 1: Introduction to FGPA Programming” [3], and “The Spartan-3E Tutorial 2: Introduction to Using the PicoBlaze Microcontroller” [4].

1.3 Scope and Further help

This tutorial is designed to help users who wish to use the LCD screen on the Spartan-3E FPGA board. For the purposes of understanding this tutorial, some background information is provided on the PicoBlaze, and the Spartan-3E LCD interface. Segments of KCPSM3 and VHDL code are also presented. However, this tutorial is not designed to be an exhaustive reference on the Spartan3E LCD, the PicoBlaze or VHDL. More detailed information on the Spartan3E and the PicoBlaze can be found in the documentation of [1,2,9]. For help with VHDL, the user can consult with a number of textbooks on the subject, such as [5,6], or find help online. The book by Chu [7] is also a useful reference for the Spartan-3 with many useful examples. Reference designs for the Spartan-3E can also be found here [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.

3.0 The PicoBlaze Micocontroller

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].

3.1 PicoBlaze Overview

A block diagram of the PicoBlaze microcontroller is shown in Figure 3.1. Some of the main features are described as follows:

Instruction PROM	Stores up to 1024 instructions consisting of the program to be run by the microcontroller.
Program Counter	Points to the next instruction to be executed.
CALL/RETURN Stack	The stack can contain up to 31 addresses for use in subroutine CALL and RETURN instructions. This means that up to 31 subroutine calls can be made before the stack overwrites itself, replacing the oldest value first.
Scratchpad RAM	64 bytes of internal memory that can be accessed using STORE and FETCH instructions.
Registers	16 byte-wide general purpose registers.
ALU	Used to calculate operations such as add, subtract, AND, OR, XOR and shift and rotate functions.
Ports	The PicoBlaze microcontroller supports up to 256 input and 256 output ports for connecting to peripheral devices or for use by other logic within the FPGA.
Flags	The ZERO and CARRY flags are set or cleared depending on the results of ALU operations.
Interrupts	An optional INTERRUPT input allows the PicoBlaze to handle asynchronous external events. The INTERRUPT_ENABLE flag can be set to enable the INTERRUPT input port.

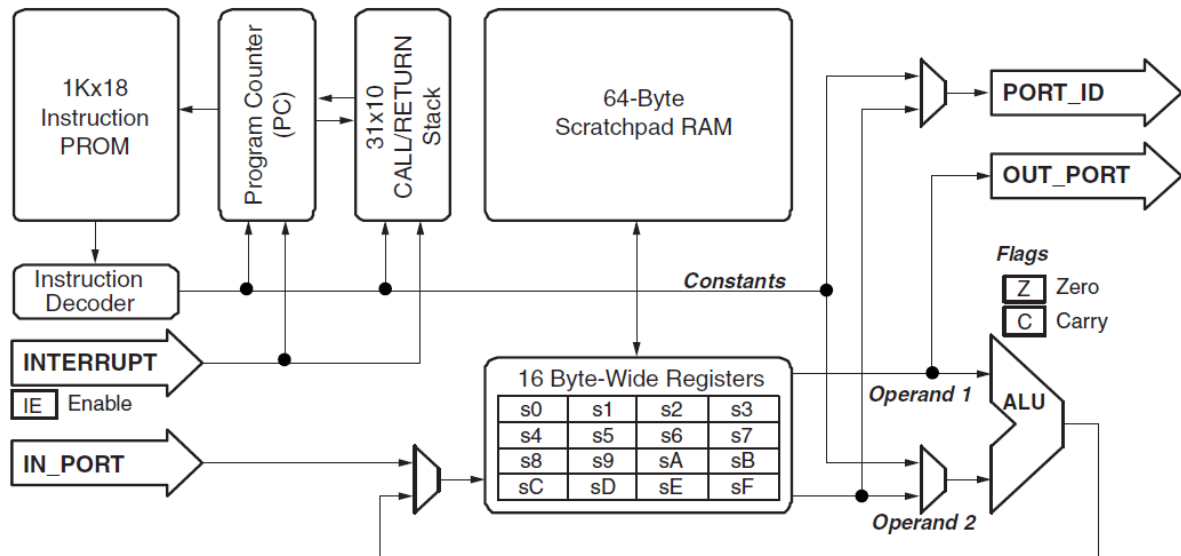


Figure 3.1: PicoBlaze microcontroller block diagram [2].

3.2 VHDL Components and Design Process

Figure 3.2 shows that the PicoBlaze consists of two VHDL components. The KCPSM3 component provides the ALU, registers, scratchpad RAM etc. The Block Memory (Program) component corresponds to the Instruction PROM in Figure 3.1 and contains the instructions to be executed.

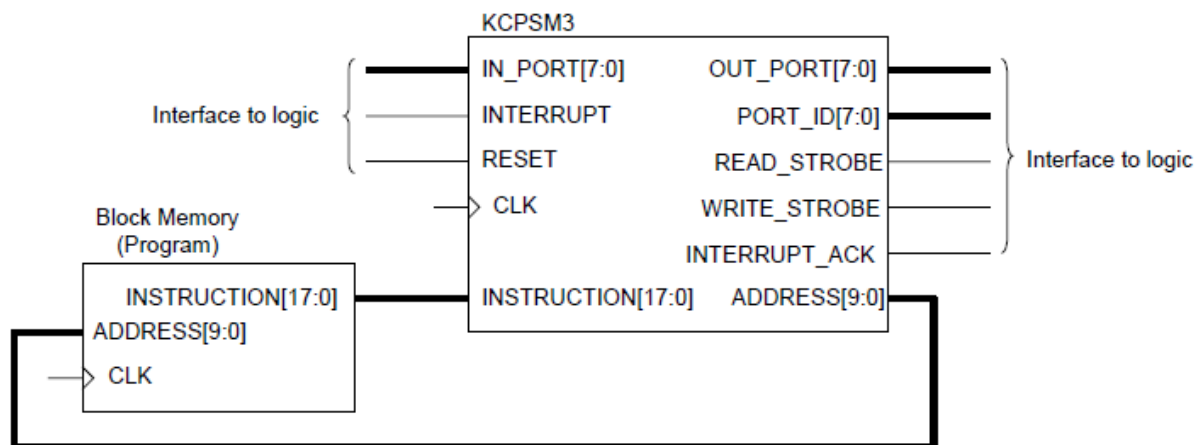


Figure 3.2: PicoBlaze VHDL components [9].

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 3.3 and 3.4 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**.

```
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;
```

Figure 3.3: KCPSM3 component declaration.

```
component myprog
  port (address      : in  std_logic_vector(9 downto 0);
        instruction  : out std_logic_vector(17 downto 0);
        clk          : in  std_logic);
end component;
```

Name of component derived
from name of .psm file

Figure 3.4: Block Memory component declaration.

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 tutorial, and the user can refer to documentation such as [2] for more information.

3.3 PicoBlaze Interface Signals

The Interface signals to the PicoBlaze are summarised in Table 3.1.

Signal	Direction	Description
IN_PORT[7:0]	Input	Input data is presented on this port during an INPUT instruction. The data is captured on the rising edge of CLK.
INTERRUPT	Input	If the INTERRUPT_ENABLE flag is set by the application code, generate an INTERRUPT Event. If the INTERRUPT_ENABLE flag is cleared, this input is ignored.
RESET	Input	Assert this input High for at least one CLK cycle, to reset the PicoBlaze microcontroller. A Reset Event is automatically generated immediately following FPGA configuration.
CLK	Input	Clock Input.
OUT_PORT[7:0]	Output	Output data appears on this port during an OUTPUT instruction. Output data is captured on the rising CLK edge when WRITE_STROBE is High.
PORT_ID[7:0]	Output	The I/O port address appears on this port for two CLK cycles during an INPUT or OUTPUT instruction.
READ_STROBE	Output	When asserted High, this signal indicates that input data on the IN_PORT[7:0] port was captured to the specified data register during an INPUT instruction.
WRITE_STROBE	Output	When asserted High, this signal validates the output data on the OUT_PORT[7:0] port during an OUTPUT instruction.
INTERRUPT_ACK	Output	When asserted High, this signal acknowledges that an INTERRUPT Event occurred.

Table 3.1: PicoBlaze interface Signals [2].

3.3 The PicoBlaze Instruction Set

The PicoBlaze Instruction Set is summarised in Table 3.2(a) and (b).

Instruction	Description
ADD sX, kk	Add register sX with literal kk.
ADD sX, sY	Add register sX with register sY.
ADDCY sX, kk	Add register sX with literal kk with CARRY bit.
ADDCY sX, sY	Add register sX with register sY with CARRY bit.
AND sX, kk	Bitwise AND register sX with literal kk.
AND sX, sY	Bitwise AND register sX with register sY.
CALL aaa	Unconditionally call subroutine at aaa.
CALL C, aaa	If CARRY flag set, call subroutine at aaa.
CALL NC, aaa	If CARRY flag not set, call subroutine at aaa.
CALL NZ, aaa	If ZERO flag not set, call subroutine at aaa.
CALL Z, aaa	If ZERO flag set, call subroutine at aaa.
COMPARE sX, kk	Compare register sX with literal kk. Set CARRY and ZERO flags as appropriate. Registers are unaffected.
COMPARE sX, sY	Compare register sX with register sY. Set CARRY and ZERO flags as appropriate. Registers are unaffected.
DISABLE INTERRUPT	Disable interrupts.
ENABLE INTERRUPT	Enable interrupts.
FETCH sX, sY	Read scratchpad RAM location pointed to by register sY into register sX.
FETCH sX, ss	Read scratchpad RAM location ss into register sX.
INPUT sX, sY	Read value on input port pointed to by register sY into register sX.
INPUT sX, pp	Read value on input port pp into register sX.
JUMP aaa	Unconditionally jump to aaa.
JUMP C, aaa	If CARRY flag set, jump to aa.
JUMP NC, aaa	If CARRY flag not set, jump to aa.
JUMP NZ, aaa	If ZERO flag not set, jump to aa.
JUMP Z, aaa	If ZERO flag set, jump to aa.
LOAD sX, kk	Load register sX with literal kk.
LOAD sX, sY	Load register sX with register sY.
OR sX, kk	Bitwise OR register sX with literal kk.

Table 3.2(a): PicoBlaze instruction set [2].

Instruction	Description
OR sX, sY	Bitwise OR register sX with register sY.
OUTPUT sX, sY	Write register sX to output port location pointed to by register sY.
OUTPUT sX, pp	Write register sX to output port location pp.
RETURN	Return unconditionally from subroutine.
RETURN C	If CARRY flag set, return from subroutine.
RETURN NC	If CARRY flag not set, return from subroutine.
RETURN NZ	If ZERO flag not set, return from subroutine.
RETURN Z	If ZERO flag set, return from subroutine.
RETURNI DISABLE	Return from interrupt service routine. Interrupt remains disabled.
RETURNI ENABLE	Return from interrupt service routine. Re-enable interrupt.
RL sX	Rotate sX register left.
RR sX	Rotate sX register right.
SL0 sX	Shift register sX left, zero fill.
SL1 sX	Shift register sX left, one fill.
SLA sX	Shift register sX left through all bits, including CARRY.
SLX sX	Shift register sX left. Bit sX[0] is unaffected.
SR0 sX	Shift register sX right, zero fill.
SR1 sX	Shift register sX right, one fill.
SRA sX	Shift register sX right through all bits, including CARRY.
SRX sX	Shift register sX right. Bit sX[7] is unaffected.
STORE sX, sY	Write register sX to scratchpad RAM location pointed to by register sY.
STORE sX, ss	Write register sX to scratchpad RAM location ss.
SUB sX, kk	Subtract literal kk from literal sX.
SUB sX, sY	Subtract register sY from register sX.
SUBCY sX, kk	Subtract literal kk from literal sX with CARRY (borrow).
SUBCY sX, sY	Subtract register sY from register sX with CARRY (borrow).
TEST sX, kk	Test bits in register sX against literal kk. Update CARRY and ZERO flags. Registers are unaffected.
TEST sX, sY	Test bits in register sX against register sY. Update CARRY and ZERO flags. Registers are unaffected.
XOR sX, kk	Bitwise XOR register sX with literal kk.
XOR sX, sY	Bitwise XOR register sX with register sY.

Table 3.2(b): PicoBlaze instruction set [2].

4.0 The LCD Display

4.1 LCD Display Overview

The Spartan-3E contains a Liquid Crystal Display (LCD) which displays 2 lines of 16 characters each. The FPGA controls the LCD through a 4-bit interface, as shown in Figure 4.1. The LCD actually supports an 8-bit interface, however the Spartan-3E uses a 4-bit interface to remain compatible with other Xilinx products, and to minimise pin count.

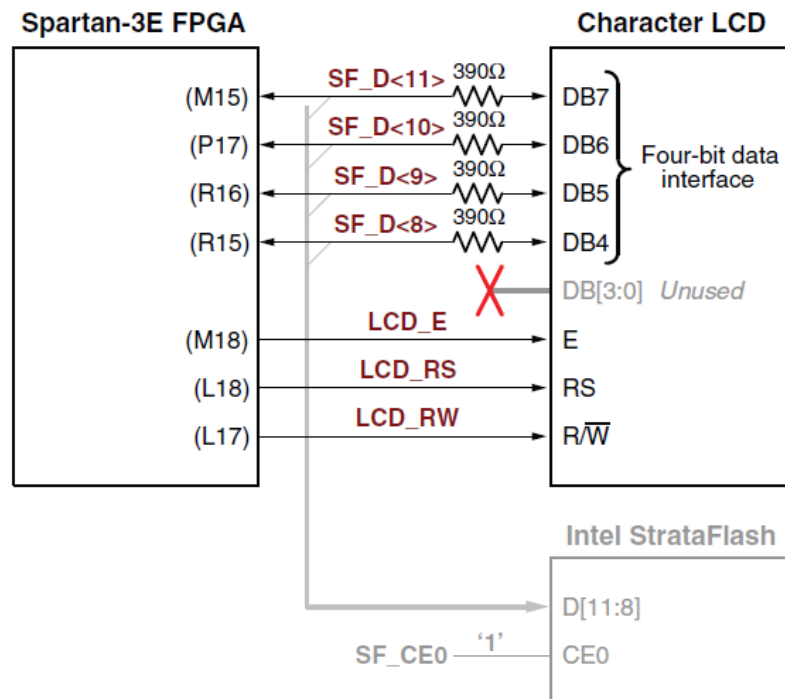


Figure 4.1: Character LCD interface [1].

The four LCD data signals are shared with the Intel Strataflash memory on the Spartan-3E. The interaction between the LCD and the StrataFlash will depend on the design [1]. When CE0 = '1', the the StrataFlash will be disabled and the FPGA has full read/write access to the LCD.

4.2 LCD Memory Map

The LCD Controller has three internal memory regions, which are outlined in this section. Before accessing any of these memory regions, the display must be initialised.

4.2.1 Display Data RAM (DD RAM)

The Display Data RAM (DD RAM) stores the character codes to be displayed on the screen. There are 80 physical locations in DD RAM, 40 per line. Only 32 characters can be displayed at a time (16 per line). Figure 4.2 shows the character display addresses for line 1 and line 2. Locations 0x00 to 0x0F in line 1 and 0x40 to 0x4F in line 2 are initially displayed. Addresses 0x10 to 0x27 and 0x50 to 0x67 can be used to store non-displayed data, or can be displayed by using the LCD's shifting functions.

Character Display Addresses																Undisplayed Addresses			
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	...	27
2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	...	67
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...	40

Figure 4.2: DD RAM Hexadecimal addresses [1].

4.2.2 Character Generator ROM (CG ROM)

The Character Generator ROM (CG ROM) contains the font bitmaps for the predefined characters that can be displayed on the LCD. The bytes stored in the DD RAM address a location in the CG ROM. As shown in Figure 4.3, the upper nibble indexes the CG ROM column while the lower nibble indexes the CG ROM row. The CG ROM contains ASCII characters and Japanese kana characters. For ASCII characters, the CG ROM address corresponds to the ASCII character code.

		Upper Data Nibble															
		DB7	DB6	DB5	DB4												
		0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
		0	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1
		0	1	1	0	0	1	1	1	1	0	0	1	1	1	1	1
		0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	1
Lower Data Nibble	xxxxx0000	CG RAM	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	xxxxx0001	CG RAM	!	1	A	Q	a	q	。	ア	チ	△	ä	q			
	xxxxx0010	CG RAM	"	2	B	R	b	r	「	イ	ツ	×	ß	θ			
	xxxxx0011	CG RAM	#	3	C	S	c	s	」	ウ	テ	ε	ε	ω			
	xxxxx0100	CG RAM	\$	4	D	T	d	t	、	エ	ト	†	μ	Ω			
	xxxxx0101	CG RAM	%	5	E	U	e	u	・	オ	ナ	1	σ	Ü			
	xxxxx0110	CG RAM	&	6	F	V	f	v	ヲ	カ	ニ	ヨ	ρ	Σ			
	xxxxx0111	CG RAM	'	7	G	W	g	w	フ	キ	ヌ	ラ	q	π			
	xxxxx1000	CG RAM	(8	H	X	h	x	イ	ク	ネ	リ	フ	×			
	xxxxx1001	CG RAM)	9	I	Y	i	y	ウ	ケ	ル	リ	フ	×			
	xxxxx1010	CG RAM	*	:	J	Z	j	z	エ	コ	ハ	レ	i	千			
	xxxxx1011	CG RAM	+	;	K	[k	[オ	サ	ヒ	ロ	*	万			
	xxxxx1100	CG RAM	,	<	L	¥	l	¥	ハ	シ	フ	ワ	¢	円			
	xxxxx1101	CG RAM	-	=	M]	m]	ユ	ズ	ハ	ン	も	÷			
	xxxxx1110	CG RAM	.	>	N	^	n	^	ヨ	セ	ホ	°	ñ				
	xxxxx1111	CG RAM	/	?	O	_	o	_	ッ	ソ	マ	°	ö	■			

Figure 4.3: LCD character set [1].

4.2.3 Character Generator RAM (CG RAM)

The Character Generator RAM (CG RAM) allows eight custom characters to be created and stored. The CG RAM occupies addresses 0x00 to 0x07 in the CG ROM. To write to the CG RAM, the *Set CG RAM Address* command is first used to set the address in the CGRAM and the row in the bitmap to be written to. Next, the *Write Data to CG RAM or DD RAM* command is used to write each row of the bitmap. Details of these LCD commands are given in Section 4.3. Table 4.1 shows an example where a checkerboard character bitmap is created in CG RAM location 0x03. The display will be lit where there is a '1' in the character bitmap, and unlit where there is a '0'. The bottom line is usually left unlit to allow for a cursor.

						Upper Nibble				Lower Nibble			
						Write Data to CG RAM of DD RAM							
A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
Character Address			Row Address			Don't Care			Character Bitmap				
0	1	1	0	0	0	-	-	-	0	1	0	1	0
0	1	1	0	0	1	-	-	-	1	0	1	0	1
0	1	1	1	1	0	-	-	-	0	1	0	1	0
0	1	1	0	1	1	-	-	-	1	0	1	0	1
0	1	1	1	0	0	-	-	-	0	1	0	1	0
0	1	1	1	0	1	-	-	-	1	0	1	0	1
0	1	1	1	1	0	-	-	-	0	1	0	1	0
0	1	1	1	1	1	-	-	-	0	0	0	0	0

Table 4.1: Example custom character at CG RAM location 0x03 [1].

4.3 LCD Controller Command Set

The commands to control the LCD are summarised in Table 4.2. Commands are sent to the LCD in two 4-bit nibbles, the upper nibble is sent first, followed by the lower nibble.

Function	LCD_RS	LCD_RW	Upper Nibble				Lower Nibble			
			DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Cursor Home	0	0	0	0	0	0	0	0	1	-
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display On/Off	0	0	0	0	0	0	1	D	C	B
Cursor and Display Shift	0	0	0	0	0	1	S/C	R/L	-	-
Function Set	0	0	0	0	1	0	1	0	-	-
Set CGRAM Address	0	0	0	1	A5	A4	A3	A2	A1	A0
Set DDRAM Address	0	0	1	A6	A5	A4	A3	A2	A1	A0
Read Busy Flag and Address	0	1	BF	A6	A5	A4	A3	A2	A1	A0
Write Data to CG RAM or DD RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0
Read Data from CG RAM or DD RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0

Table 4.2: LCD Character display command set [1].

The LCD commands in Table 4.2, and the values of the bits are detailed as follows:

Disable

If the LCD_E signal is low, all other inputs to the LCD are ignored and the LCD is disabled.

Clear Display

Clears the display and returns the cursor to the home position.

Return Cursor Home

Return the cursor to the home position, top left corner.

Entry Mode Set

Sets the cursor move direction and specifies whether or not to shift the display, during data reads and writes.

I/D (Increment/Decrement) bit – Sets the direction of cursor movement or display shift.

I/D	
0	Cursor or display moves to left.
1	Cursor or display moves to right.

S (Shift) bit – Turn display shift on or off.

S	
0	Shifting disabled.
1	Shifting enabled, and direction of shifting given by I/D bit. Appears as though the cursor position remains constant and the display moves.

Display On/Off

Display, cursor and cursor blinking are turned on or off.

D (Display) bit – Turns the display on or off.

I/D	
0	Display off.
1	Display on.

C (Cursor) bit – Turn cursor on or off.

S	
0	Cursor off.
1	Cursor on.

B (Cursor Blink) bit – Turn cursor blinking on or off.

S	
0	Cursor blinking off.
1	Cursor blinking on.

Cursor and Display Shift

Moves the cursor and shifts the display. The cursor automatically moves to the second line if it moves past the 40th character on the first line. When the display is shifted, both lines move at the same time.

D (Display) bit – Turns the display on or off.

S/C	R/L	
0	0	Shift the cursor to the left.
0	1	Shift the cursor to the right.
1	0	Shift the entire display to the left.
1	1	Shift the entire display to the right.

Function Set

Sets interface data length, number of display lines, and character font. The Spartan-3E supports a single function set with value 0x28.

Set CG RAM Address

Set the CG RAM address. All subsequent read/write commands will be to/from CG RAM.

Set DD RAM Address

Set the DD RAM address. All subsequent read/write commands will be to/from DD RAM.

Read Busy Flag and Address

Reads the busy flag (BF) to determine if an internal operation is in progress, and reads the current address counter. BF = 1 indicates that an internal operation is in progress.

Write Data to CG RAM or DD RAM

Write data into CG RAM if this command follows a *Set CG RAM Address* command, and write data into DD RAM if this command follows a *Set DD RAM Address* command.

Read Data from CG RAM or DD RAM

Read data from CG RAM if this command follows a *Set CG RAM Address* command, and read data from DD RAM if this command follows a *Set DD RAM Address* command.

4.4 PicoBlaze Code for LCD Control

In this tutorial, the PicoBlaze microcontroller is used to control the LCD display. Figure 4.5 shows a skeleton program in KCPSM3 Assembler code, which contains a number of subroutines for LCD control, and a placeholder main program. The LCD and delay subroutines presented here are from the *Initial Design for the Spartan-3E FPGA Starter Kit Board*, *control.psm* file, developed by Ken Chapman. This implements the initial design shipped with the board, and is one of the *Spartan-3E FPGA Starter Kit Board Design Examples*, Xilinx reference design resources that are available for download from [8].

In the code of Figure 4.5, comments have been printed in green, to distinguish them from assembler instructions and directives. To assist with explanation, the code has been split into colour-coded blocks. The purpose of each block and where applicable, each subroutine, are outlined in Table 4.2.

The blocks are colour-coded in Table 4.3 and Figure 4.5 for easier identification.

Block	subroutines	purpose
Block A	-	Definition of constants for the input/output ports for the LEDs, switches and LCD.
Block B	-	Main program – Calls LCD_reset, and then enters an infinite loop which reads in the values of the slider switches and push buttons, and then outputs these values to the LEDs. The main program can be thought of as a placeholder which later in this tutorial, will be replaced with code that calls the LCD control subroutines.
Block C		Software delay routines – Use a series of loops to delay for a given amount of time.
	delay_1us	Delay for 1 μ s.
	delay_40us	Delay for 40 μ s.
	delay_1ms	Delay for 1ms.
	delay_20ms	Delay for 20ms.
	delay_1s	Delay for 1s.
Block D		LCD Character module routines
	LCD_pulse_E	Send an LCD_E pulse
	LCD_write_inst4	Write a nibble (4 bits) to the LCD
	LCD_write_inst8	Write an 8 bit instruction to the LCD. Calls LCD_write_inst4 to send two nibbles.
	LCD_write_data	Write data to the display
	LCD_read_data8	Read data from the display
	LCD_reset	Perform initialisation and clear the display.

Table 4.3: Explanation of code for Figure 4.5.


```

;*****
;Port definitions
;*****
;
CONSTANT LED_port, 80          ;8 simple LEDs
CONSTANT LED0, 01              ; LED 0 - bit0
CONSTANT LED1, 02              ; 1 - bit1
CONSTANT LED2, 04              ; 2 - bit2
CONSTANT LED3, 08              ; 3 - bit3
CONSTANT LED4, 10              ; 4 - bit4
CONSTANT LED5, 20              ; 5 - bit5
CONSTANT LED6, 40              ; 6 - bit6
CONSTANT LED7, 80              ; 7 - bit7
;
CONSTANT switch_port, 00       ;Read switches and press buttons
CONSTANT switch0, 01           ; Switches SW0 - bit0
CONSTANT switch1, 02           ; SW1 - bit1
CONSTANT switch2, 04           ; SW2 - bit2
CONSTANT switch3, 08           ; SW3 - bit3
CONSTANT BTN_east, 10          ; Buttons East - bit4
CONSTANT BTN_south, 20         ; South - bit5
CONSTANT BTN_north, 40         ; North - bit6
CONSTANT BTN_west, 80          ; West - bit7
;
;
;LCD interface ports
;
;The master enable signal is not used by the LCD display itself
;but may be required to confirm that LCD communication is active.
;This is required on the Spartan-3E Starter Kit if the StrataFLASH
;is used because it shares the same data pins and conflicts must be
;avoided.
;
CONSTANT LCD_output_port, 40    ;LCD character module output
                                ;data and control
CONSTANT LCD_E, 01              ; active High Enable E - bit0
CONSTANT LCD_RW, 02             ; Read=1 Write=0 RW - bit1
CONSTANT LCD_RS, 04             ; Instruction=0 Data=1 RS - bit2
CONSTANT LCD_drive, 08          ; Master enable (active High)
                                ; - bit3
CONSTANT LCD_DB4, 10            ; 4-bit Data DB4 - bit4
CONSTANT LCD_DB5, 20            ; interface Data DB5 - bit5
CONSTANT LCD_DB6, 40            ; Data DB6 - bit6
CONSTANT LCD_DB7, 80            ; Data DB7 - bit7
;
;
CONSTANT LCD_input_port, 02      ;LCD character module input data
CONSTANT LCD_read_spare0, 01     ; Spare bits - bit0
CONSTANT LCD_read_spare1, 02     ; are zero - bit1
CONSTANT LCD_read_spare2, 04     ; - bit2
CONSTANT LCD_read_spare3, 08     ; - bit3
CONSTANT LCD_read_DB4, 10        ; 4-bit Data DB4 - bit4
CONSTANT LCD_read_DB5, 20        ; interface Data DB5 - bit5
CONSTANT LCD_read_DB6, 40        ; Data DB6 - bit6
CONSTANT LCD_read_DB7, 80        ; Data DB7 - bit7
;
;

```

BLOCK A

Figure 4.4(a): PicoBlaze Assembler code for LCD control.

<pre> ;Constant to define a software delay of 1us. This must be adjusted ;to reflect the clock applied to KCPSM3. Every instruction executes ;in 2 clock cycles making the calculation highly predictable. The ;'6' in the following equation even allows for 'CALL delay_1us' ;instruction in the initiating code. ; ;delay_1us_constant = (clock_rate - 6)/4 Where 'clock_rate' ;is in MHz ; ;Example: For a 50MHz clock the constant value is (10-6)/4 = 11 ;(0B Hex). ;For clock rates below 10MHz the value of 1 must be used and the ;operation will become lower than intended. ; CONSTANT delay_1us_constant, 0B ; </pre>	BLOCK A
<pre> ; ;***** ;Initialise the system ;***** ; cold_start: CALL LCD_reset ;initialise LCD display ; Main: INPUT s7, switch_port ;read in switches OUTPUT s7, LED_port ;output to LEDs JUMP Main ; ; </pre>	BLOCK B
<pre> ; ;***** ;Software delay routines ;***** ; ; ;Delay of 1us. ; ;Constant value defines reflects the clock applied ;to KCPSM3. Every instruction executes in 2 clock ;cycles making the calculation highly predictable. ;The '6' in the following equation even allows for ;'CALL delay_1us' instruction in the initiating code. ; ; delay_1us_constant = (clock_rate - 6)/4 ; Where 'clock_rate' is in MHz ; ;Registers used s0 ; delay_1us: LOAD s0, delay_1us_constant wait_1us: SUB s0, 01 JUMP NZ, wait_1us RETURN ; ;Delay of 40us. ; ;Registers used s0, s1 ; </pre>	BLOCK C

Figure 4.4(b): PicoBlaze Assembler code for LCD control.

<pre> delay_40us: LOAD s1, 28 ;40 x 1us = 40us wait_40us: CALL delay_1us SUB s1, 01 JUMP NZ, wait_40us RETURN ; ; ;Delay of 1ms. ; ;Registers used s0, s1, s2 ; delay_1ms: LOAD s2, 19 ;25 x 40us = 1ms wait_1ms: CALL delay_40us SUB s2, 01 JUMP NZ, wait_1ms RETURN ; ;Delay of 20ms. ; ;Delay of 20ms used during initialisation. ; ;Registers used s0, s1, s2, s3 ; delay_20ms: LOAD s3, 14 ;20 x 1ms = 20ms wait_20ms: CALL delay_1ms SUB s3, 01 JUMP NZ, wait_20ms RETURN ; ;Delay of approximately 1 second. ; ;Registers used s0, s1, s2, s3, s4 ; delay_1s: LOAD s4, 32 ;50 x 20ms = 1000ms wait_1s: CALL delay_20ms SUB s4, 01 JUMP NZ, wait_1s RETURN ; </pre>	BLOCK C
<pre> ; ; ;***** ;LCD Character Module Routines ;***** ; ;LCD module is a 16 character by 2 line display but ;all displays are very similar The 4-wire data ;interface will be used (DB4 to DB7). ; ;The LCD modules are relatively slow and software ;delay loops are used to slow down KCPSM3 ;adequately for the LCD to communicate. The delay ;routines are provided in a different section (see ;above in this case). ; ; </pre>	BLOCK D

Figure 4.4(c): PicoBlaze Assembler code for LCD control.

<pre> ;Pulse LCD enable signal 'E' high for greater than ;230ns (1us is used). ; ;Register s4 should define the current state of the ;LCD output port. ; ;Registers used s0, s4 ; LCD_pulse_E: XOR s4, LCD_E ;E=1 OUTPUT s4, LCD_output_port CALL delay_1us XOR s4, LCD_E ;E=0 OUTPUT s4, LCD_output_port RETURN ; ;Write 4-bit instruction to LCD display. ; ;The 4-bit instruction should be provided in the ;upper 4-bits of register s4. ;Note that this routine does not release the master ;enable but as it is only ;used during initialisation and as part of the ;8-bit instruction write it should be acceptable. ; ;Registers used s4 ; LCD_write_inst4: AND s4, F8 ;Enable=1 RS=0 ;Instruction, RW=0 ;Write, E=0 ; OUTPUT s4, LCD_output_port ;set up RS and RW>40ns ;before enable pulse ; CALL LCD_pulse_E RETURN ; ; ;Write 8-bit instruction to LCD display. ; ;The 8-bit instruction should be provided in ; register s5. ;Instructions are written using the following ;sequence ; Upper nibble ; wait >1us ; Lower nibble ; wait >40us ; ;Registers used s0, s1, s4, s5 ; LCD_write_inst8: LOAD s4, s5 AND s4, F0 ;Enable=0 RS=0 ;Instruction, RW=0 ;Write, E=0 ; </pre>	BLOCK D
--	----------------

Figure 4.4(d): PicoBlaze Assembler code for LCD control.

```

OR s4, LCD_drive           ;Enable=1
CALL LCD_write_inst4       ;write upper nibble
CALL delay_1us             ;wait >1us
LOAD s4, s5                ;select lower nibble
                           ;with
SL1 s4                     ;Enable=1
SL0 s4                     ;RS=0 Instruction
SL0 s4                     ;RW=0 Write
SL0 s4                     ;E=0
CALL LCD_write_inst4       ;write lower nibble
CALL delay_40us            ;wait >40us
LOAD s4, F0                ;Enable=0 RS=0
                           ;Instruction, RW=0
                           ;Write, E=0

;
OUTPUT s4, LCD_output_port ;Release master enable
;
RETURN
;
;Write 8-bit data to LCD display.
;
;The 8-bit data should be provided in register s5.
;Data bytes are written using the following sequence
; Upper nibble
; wait >1us
; Lower nibble
; wait >40us
;
;Registers used s0, s1, s4, s5
;
LCD_write_data: LOAD s4, s5
AND s4, F0                ;Enable=0 RS=0
                           ;Instruction, RW=0
                           ;Write, E=0

;
OR s4, 0C                 ;Enable=1 RS=1 Data,
                           ;RW=0 Write, E=0

;
OUTPUT s4, LCD_output_port ;set up RS and RW>40ns
                           ;before enable pulse

;
CALL LCD_pulse_E           ;write upper nibble
CALL delay_1us             ;wait >1us
LOAD s4, s5                ;select lower nibble
                           ;with
SL1 s4                     ;Enable=1
SL1 s4                     ;RS=1 Data
SL0 s4                     ;RW=0 Write
SL0 s4                     ;E=0
OUTPUT s4, LCD_output_port ;set up RS and RW>40ns
                           ;before enable pulse

;
CALL LCD_pulse_E           ;write lower nibble
CALL delay_40us            ;wait >40us
LOAD s4, F0                ;Enable=0 RS=0
                           ;Instruction, RW=0
                           ;Write, E=0

```

BLOCK D

Figure 4.4(e): PicoBlaze Assembler code for LCD control.

```

;
OUTPUT s4, LCD_output_port      ;Release master enable
;
RETURN
;
;Read 8-bit data from LCD display.
;
;The 8-bit data will be read from the current LCD
;memory address and will be returned in register s5.
;It is advisable to set the LCD address (cursor
;position) before using the data read for the first
;time otherwise the display may generate invalid data
;on the first read.
;
;Data bytes are read using the following sequence
; Upper nibble
; wait >1us
; Lower nibble
; wait >40us
;
;Registers used s0, s1, s4, s5
;
LCD_read_data8: LOAD s4, OE      ;Enable=1 RS=1 Data,
                                ;RW=1 Read, E=0
;
OUTPUT s4, LCD_output_port      ;set up RS and RW>40ns
                                ;before enable pulse
;
XOR s4, LCD_E                   ;E=1
OUTPUT s4, LCD_output_port
CALL delay_1us                  ;wait >260ns to
                                ;access data
;
INPUT s5, LCD_input_port        ;read upper nibble
XOR s4, LCD_E                   ;E=0
OUTPUT s4, LCD_output_port
CALL delay_1us                  ;wait >1us
XOR s4, LCD_E                   ;E=1
OUTPUT s4, LCD_output_port
CALL delay_1us                  ;wait >260ns to
                                ;access data
;
INPUT s0, LCD_input_port        ;read lower nibble
XOR s4, LCD_E                   ;E=0
OUTPUT s4, LCD_output_port
AND s5, F0                      ;merge upper and
                                ;lower nibbles

SR0 s0
SR0 s0
SR0 s0
SR0 s0
OR s5, s0
LOAD s4, 04                    ;Enable=0 RS=1 Data,
                                ;RW=0 Write, E=0
;

```

BLOCK D

Figure 4.4(f): PicoBlaze Assembler code for LCD control.

```

        OUTPUT s4, LCD_output_port      ;Stop reading 5V
                                          ;device and release
                                          ;master enable
;
CALL delay_40us                        ;wait >40us
RETURN
;
;Reset and initialise display to communicate using
;4-bit data mode
;Includes routine to clear the display.
;
;Requires the 4-bit instructions 3,3,3,2 to be sent
;with suitable delays following by the 8-bit
;instructions to set up the display.
;
; 28 = '001' Function set, '0' 4-bit mode, '1' 2-line,
;      '0' 5x7 dot matrix, 'xx'
; 06 = '000001' Entry mode, '1' increment, '0'
;      no display shift
; 0C = '00001' Display control, '1' display on,
;      '0' cursor off, '0' cursor blink off
; 01 = '00000001' Display clear
;
;Registers used s0, s1, s2, s3, s4
;
LCD_reset: CALL delay_20ms              ;wait more that
                                          ;15ms for display
                                          ;to be ready
;
LOAD s4, 30
CALL LCD_write_inst4                  ;send '3'
CALL delay_20ms                      ;wait >4.1ms
CALL LCD_write_inst4                  ;send '3'
CALL delay_1ms                       ;wait >100us
CALL LCD_write_inst4                  ;send '3'
CALL delay_40us                      ;wait >40us
LOAD s4, 20
CALL LCD_write_inst4                  ;send '2'
CALL delay_40us                      ;wait >40us
LOAD s5, 28                          ;Function set
CALL LCD_write_inst8
LOAD s5, 06                          ;Entry mode
CALL LCD_write_inst8
LOAD s5, 0C                          ;Display control
CALL LCD_write_inst8
LCD_clear: LOAD s5, 01                ;Display clear
CALL LCD_write_inst8
CALL delay_1ms                       ;wait >1.64ms for
                                          ;display to clear

CALL delay_1ms
RETURN
;

```

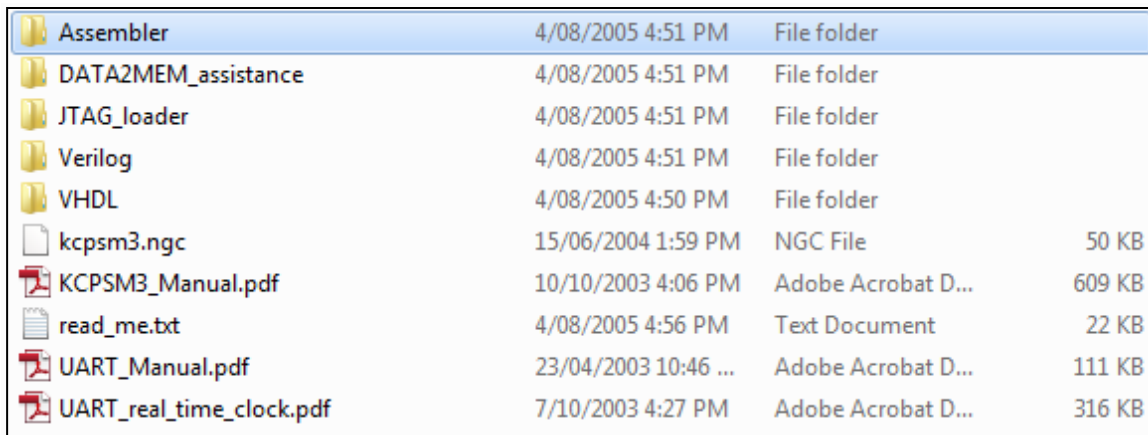
BLOCK D

Figure 4.4(g): PicoBlaze Assembler code for LCD control.

5.0 Procedure – File Setup

5.1 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 5.1.



Assembler	4/08/2005 4:51 PM	File folder	
DATA2MEM_assistance	4/08/2005 4:51 PM	File folder	
JTAG_loader	4/08/2005 4:51 PM	File folder	
Verilog	4/08/2005 4:51 PM	File folder	
VHDL	4/08/2005 4:50 PM	File folder	
kcpsm3.ngc	15/06/2004 1:59 PM	NGC File	50 KB
KCPSM3_Manual.pdf	10/10/2003 4:06 PM	Adobe Acrobat D...	609 KB
read_me.txt	4/08/2005 4:56 PM	Text Document	22 KB
UART_Manual.pdf	23/04/2003 10:46 ...	Adobe Acrobat D...	111 KB
UART_real_time_clock.pdf	7/10/2003 4:27 PM	Adobe Acrobat D...	316 KB

Figure 5.1: KCPSM3 files after unzipping.

The file **KCPSM3_Manual.pdf** is listed as reference [9] in this tutorial.

5.2 Copy Files

1. Create a directory called **tutorial_3** 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_3**:
 - KCPSM3.EXE
 - ROM_form.coe
 - ROM_form.v
 - ROM_form.vhd
3. Copy the following file in the **VHDL** directory into **tutorial_3**:
 - kcpsm3.vhd

5.3 Setup **hello.psm**

Copy the code from Figure 4.5 into a text file, and save it as **hello.psm**. Alternatively, download the file **initial.psm**, available for download with this tutorial, and rename it as **hello.psm**.

Note that **.psm** files can be edited using any text editor, including Notepad and Wordpad, or even the Xilinx Project Navigator software. If using Notepad, be careful not to save the file as **hello.psm.txt**. If using Project Navigator to edit **.psm** files, remember that the **.psm** file is not added to the VHDL project.

6.0 Procedure Part A – Hello World on the Spartan3E LCD

6.1 PicoBlaze Code for Hello World

1. Open your file **hello.psm** in a text editor (for example, Notepad or Wordpad).
2. Replace the main program (Block B) in Figure 4.5 with the code of Figure 6.1.

```

cold_start: CALL LCD_reset           ;initialise LCD display
            CALL Disp_mesg          ;display message

            Main: INPUT s7, switch_port ;read in switches
            OUTPUT s7, LED_port       ;output to LEDs
            JUMP Main

            Disp_mesg: LOAD s5, 80    ;Line 1 position 1
            CALL LCD_write_inst8
            CALL Disp_Hello_World

            LOAD s5, C0              ;Line 2 position 1
            CALL LCD_write_inst8
            CALL Disp_Hello_World
            RETURN

Disp_Hello_World: LOAD s5, 48        ;H
                  CALL LCD_write_data
                  LOAD s5, 65        ;e
                  CALL LCD_write_data
                  LOAD s5, 6C        ;l
                  CALL LCD_write_data
                  ;LOAD s5, 6C        ;l
                  CALL LCD_write_data
                  LOAD s5, 6F        ;o
                  CALL LCD_write_data
                  LOAD s5, 20        ;space
                  CALL LCD_write_data
                  LOAD s5, 57        ;W
                  CALL LCD_write_data
                  LOAD s5, 6F        ;o
                  CALL LCD_write_data
                  LOAD s5, 72        ;r
                  CALL LCD_write_data
                  LOAD s5, 6C        ;l
                  CALL LCD_write_data
                  LOAD s5, 64        ;d
                  CALL LCD_write_data
                  LOAD s5, 20        ;space
                  CALL LCD_write_data
                  RETURN

```

Figure 6.1: Main program for hello.psm.

For reference the modified **hello.psm** is listed in Appendix A. Note that in Figure 6.2 and in all PicoBlaze assembler code in this tutorial, comments are printed in green to distinguish them from KCPSM3 assembler instructions and directives.

As previously, the main program calls **LCD_reset**, and then enters an infinite loop which reads in the values of the slider switches and push buttons, and then outputs these values to the LEDs. However, subroutine **Disp_mesg** is now called before entering the loop. This subroutine positions the cursor at line 1, position 1, and then calls **Disp_Hello_World** to write the text “Hello World” to line 1 of the LCD. Next, it positions the cursor at line 2 position 1 and repeats the process to display the same message on line 2.

6.2 Running the Assembler

As shown in Figure 6.2, 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.

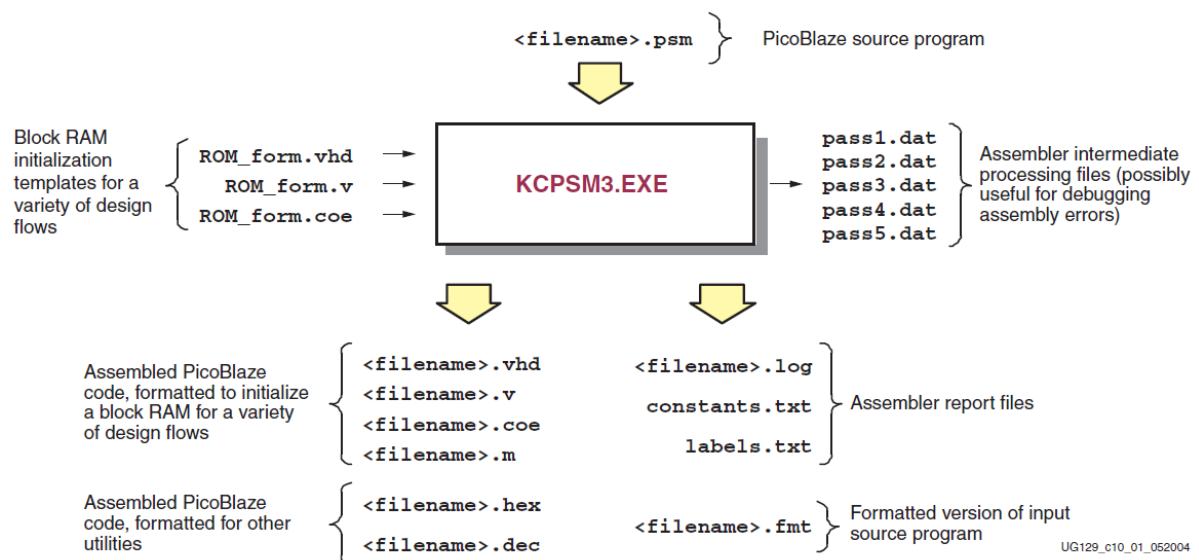


Figure 6.2: KCPSM3 assembler files [2].

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

6.2.1 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_3** working directory, as shown in Figure 6.3.

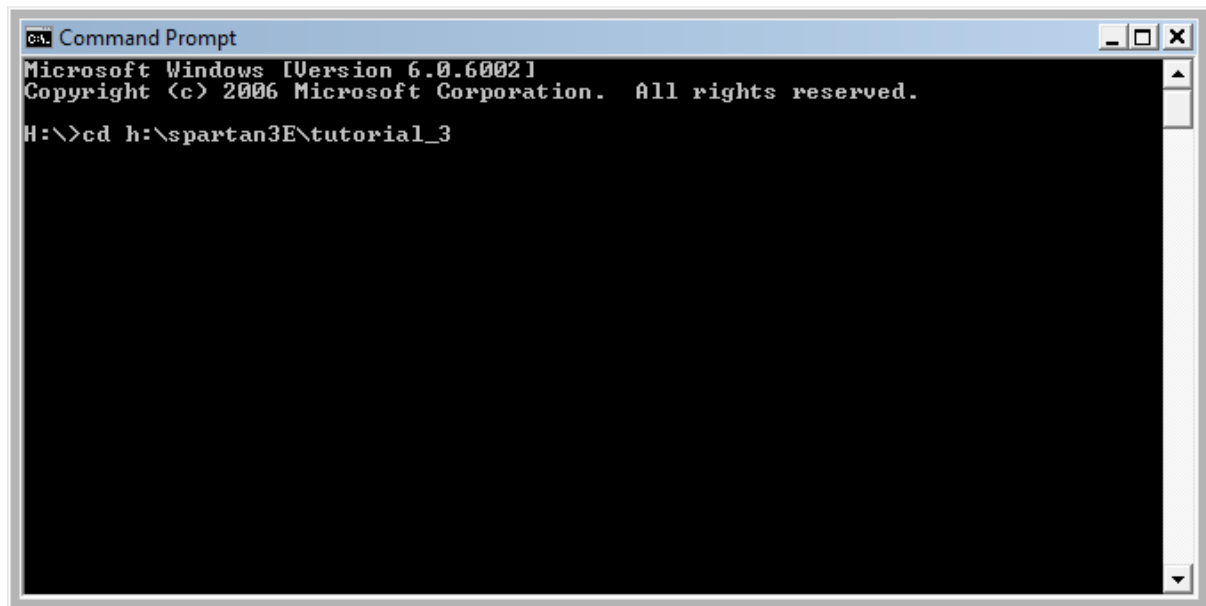


Figure 6.3: DOS Command Prompt window, after changing to working directory.

3. Now type the command **KCPSM3 hello.psm**, as shown in Figure 6.4.

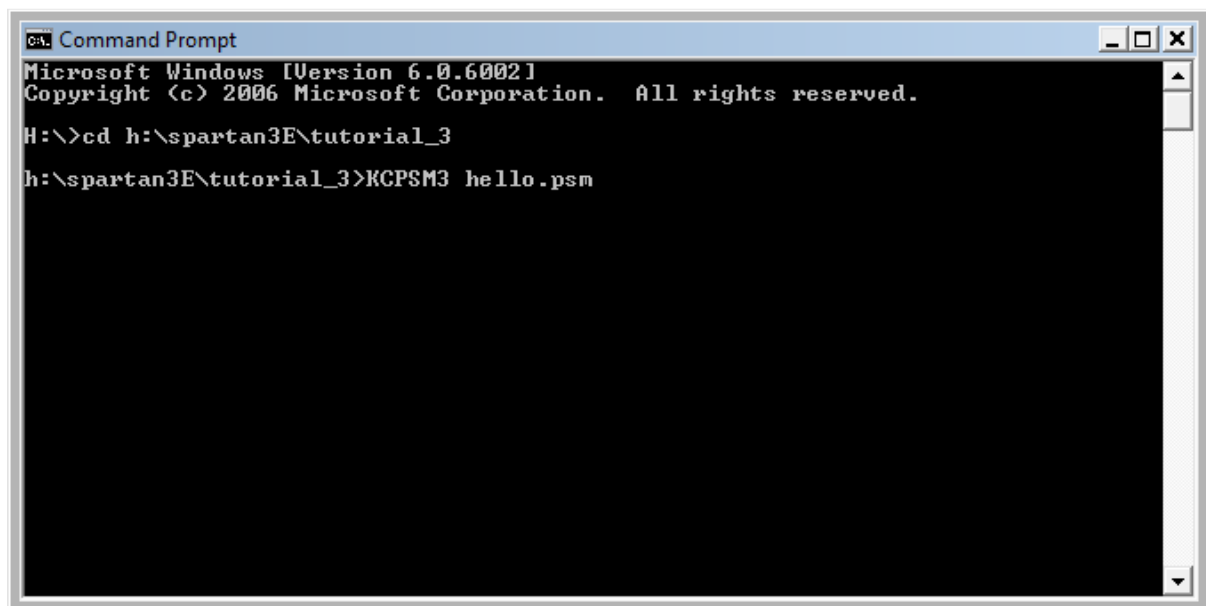
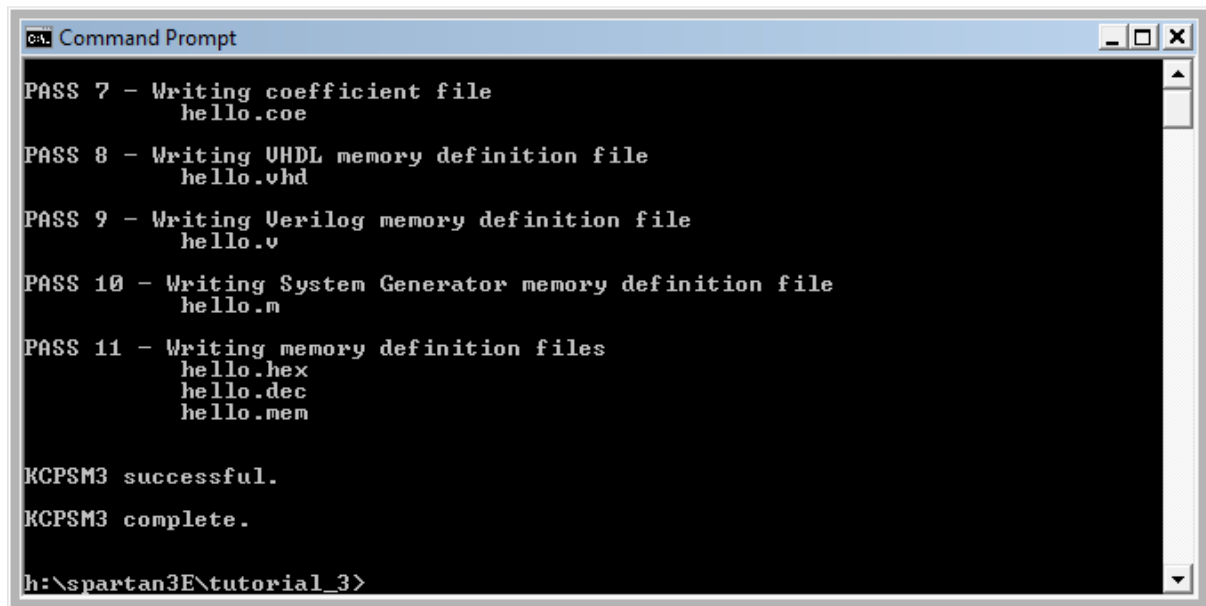


Figure 6.4: DOS Command Prompt window, with KCPSM3 command typed in.

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 6.5. After the assembler has successfully run, the working directory should contain many more files, as shown in Figure 6.10.



```
Command Prompt

PASS 7 - Writing coefficient file
        hello.coe

PASS 8 - Writing UHDL memory definition file
        hello.vhd

PASS 9 - Writing Verilog memory definition file
        hello.v

PASS 10 - Writing System Generator memory definition file
        hello.m

PASS 11 - Writing memory definition files
        hello.hex
        hello.dec
        hello.mem

KCPSM3 successful.
KCPSM3 complete.

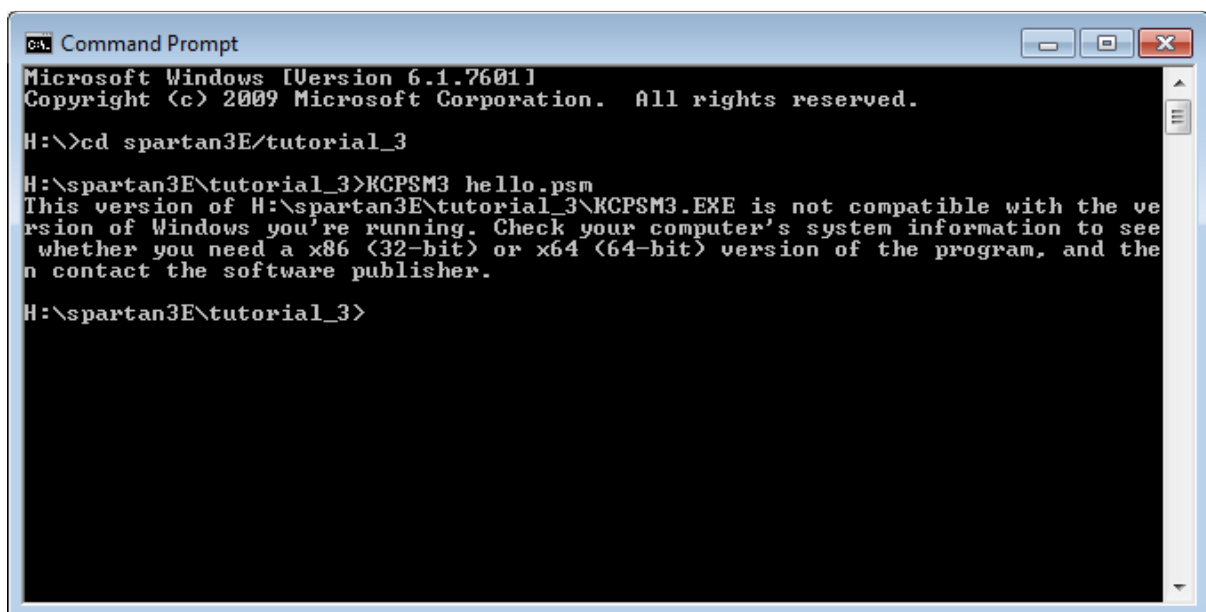
h:\spartan3E\tutorial_3>
```

Figure 6.5: DOS Command Prompt window, after KCPSM3 successfully run.

4. Type **exit** to close the Command Prompt window.

6.2.2 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 KCPMS3 in a DOS Command Prompt window, the error message shown in Figure 6.6 will appear.



```
Command Prompt

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

H:\>cd spartan3E\tutorial_3

H:\spartan3E\tutorial_3>KCPSM3 hello.psm
This version of H:\spartan3E\tutorial_3\KCPSM3.EXE is not compatible with the ve
rsion of Windows you're running. Check your computer's system information to see
whether you need a x86 (32-bit) or x64 (64-bit) version of the program, and the
n contact the software publisher.

H:\spartan3E\tutorial_3>
```

Figure 6.6: Error message which appears if it is attempted to run KCPSM3 on a 64-bit machine.

One way to work around this and run KCPSM3 is to use the DOSbox software, which can be downloaded from <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 6.7 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.

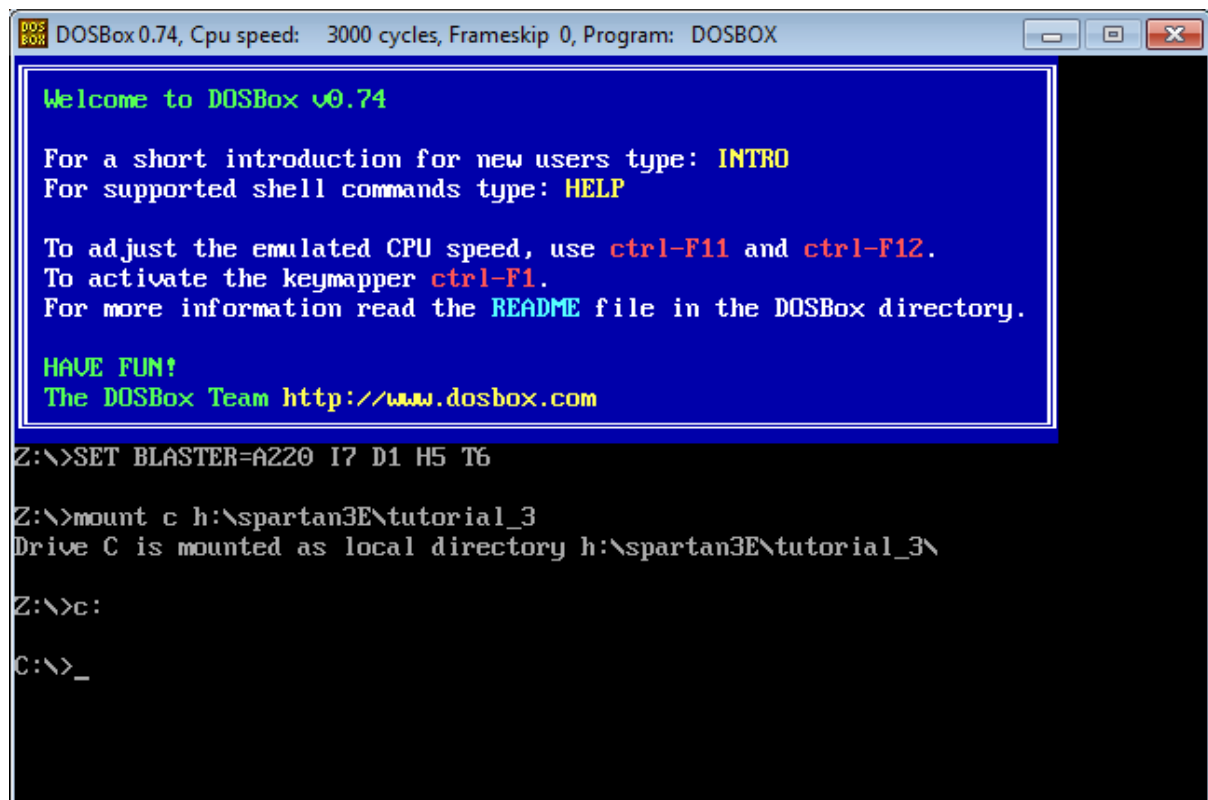


Figure 6.7: DOSBox window, commands entered to mount and change into the working directory.

3. Now type the command **KCPSM3 hello.psm**, as shown in Figure 6.8.

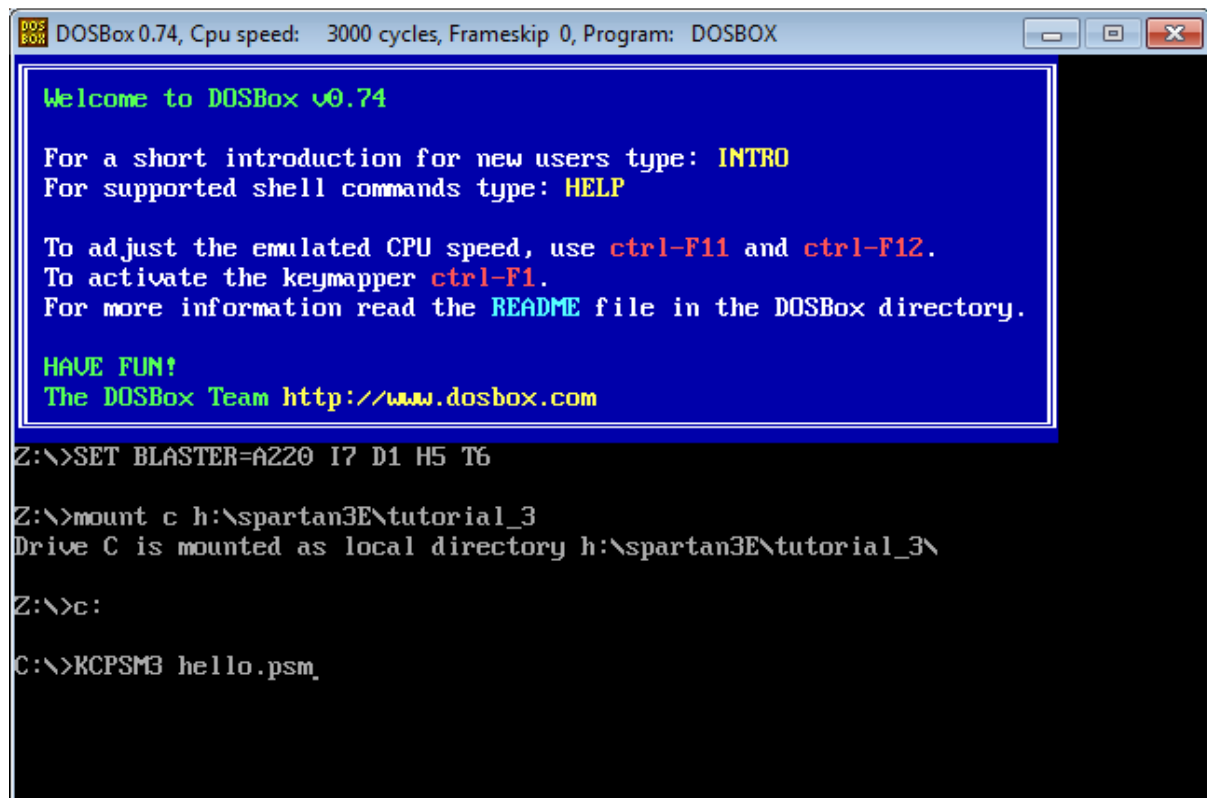
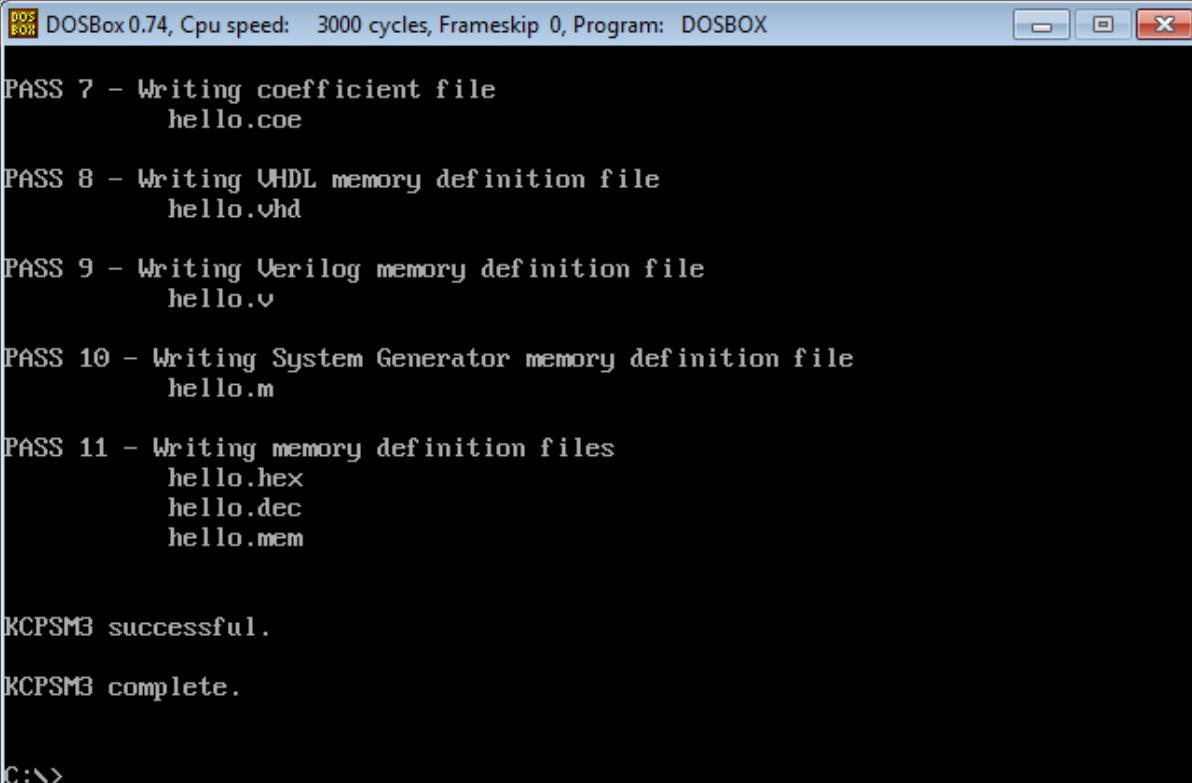


Figure 6.8: DOSBox window, with KCPSM3 command typed in.

After entering the command **KCPSM3 hello.psm**, numerous messages should fly past on the screen, ending with “KCPSM3 successful. KCPSM3 complete”, as shown in Figure 6.9. After the assembler has successfully run, the working directory should contain many more files, as shown in Figure 6.10.



The image shows a DOSBox window with a blue title bar. The title bar text is "DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX". The window contains a black terminal area with white text. The text shows the progression of the KCPSM3 process through several passes, writing various files, and finally reaching a successful completion state. The prompt "C:\>_" is visible at the bottom left of the terminal area.

```
DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: DOSBOX

PASS 7 - Writing coefficient file
        hello.coe

PASS 8 - Writing VHDL memory definition file
        hello.vhd

PASS 9 - Writing Verilog memory definition file
        hello.v

PASS 10 - Writing System Generator memory definition file
        hello.m

PASS 11 - Writing memory definition files
        hello.hex
        hello.dec
        hello.mem

KCPSM3 successful.

KCPSM3 complete.

C:\>_
```

Figure 6.9: DOSBox window, after KCPSM3 successfully run.

4. Type **exit** to close DOSBox.























 CONSTANT.TXT	4/02/2013 12:15 PM	Text Document	1 KB
 HELLO.COE	4/02/2013 12:15 PM	COE File	8 KB
 HELLO.DEC	4/02/2013 12:15 PM	DEC File	0 KB
 HELLO.FMT	4/02/2013 12:15 PM	FMT File	19 KB
 HELLO.HEX	4/02/2013 12:15 PM	HEX File	0 KB
 HELLO.LOG	4/02/2013 12:15 PM	Text Document	24 KB
 HELLO.M	4/02/2013 12:15 PM	MATLAB Code	5 KB
 HELLO.MEM	4/02/2013 12:15 PM	MEM File	0 KB
 hello.psm	21/01/2013 6:56 AM	PSM File	19 KB
 HELLO.V	4/02/2013 12:15 PM	V File	23 KB
 HELLO.VHD	4/02/2013 12:15 PM	VHD File	0 KB
 KCPSM3.EXE	5/07/2005 9:33 AM	Application	89 KB
 kcpsm3.vhd	20/07/2005 8:50 AM	VHD File	67 KB
 LABELS.TXT	4/02/2013 12:15 PM	Text Document	1 KB
 PASS1.DAT	4/02/2013 12:15 PM	DAT File	34 KB
 PASS2.DAT	4/02/2013 12:15 PM	DAT File	34 KB
 PASS3.DAT	4/02/2013 12:15 PM	DAT File	12 KB
 PASS4.DAT	4/02/2013 12:15 PM	DAT File	50 KB
 PASS5.DAT	4/02/2013 12:15 PM	DAT File	66 KB
 ROM_form.coe	25/01/2002 4:17 PM	COE File	1 KB
 ROM_form.v	4/07/2005 6:05 PM	V File	15 KB
 ROM_form.vhd	5/07/2005 9:39 AM	VHD File	13 KB

Figure 6.10: Files in the working directory after KCPSM3 successfully run.

6.3 Starting Project Navigator

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 6.11.

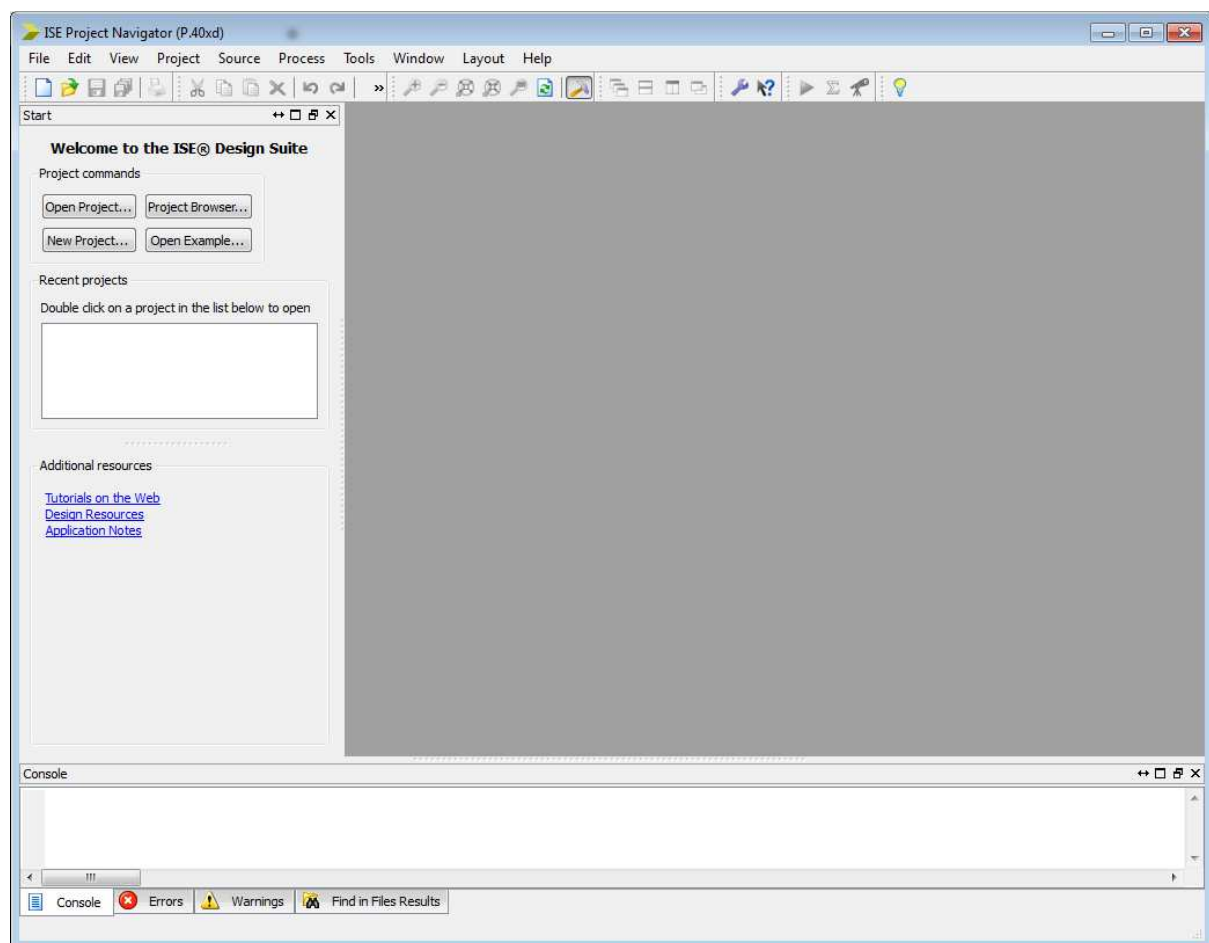


Figure 6.11: Project Navigator Software Startup Window.

6.4 Creating a New Project

1. Select **File→New Project**. The **New Project Wizard** will appear.
2. Type **tutorial_3** in the **Name:** field.
3. Choose **Location:** and **Working Directory:** as the **tutorial_3** working directory.
4. Verify that **Top-level source type:** is selected as **HDL**.
5. The properties should now be set as shown in Figure 6.12. Click **Next** to move to the **Project Settings** page.

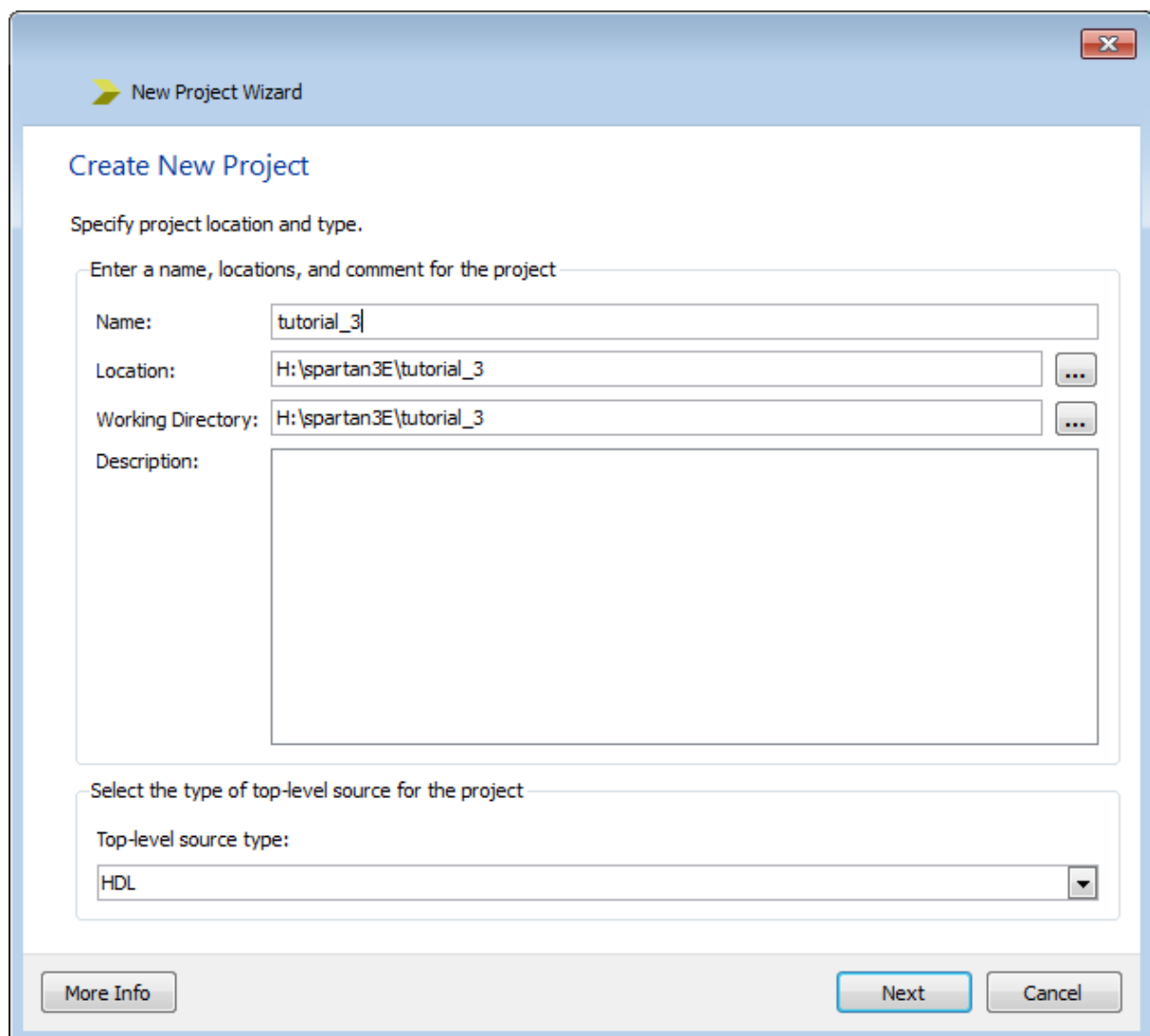


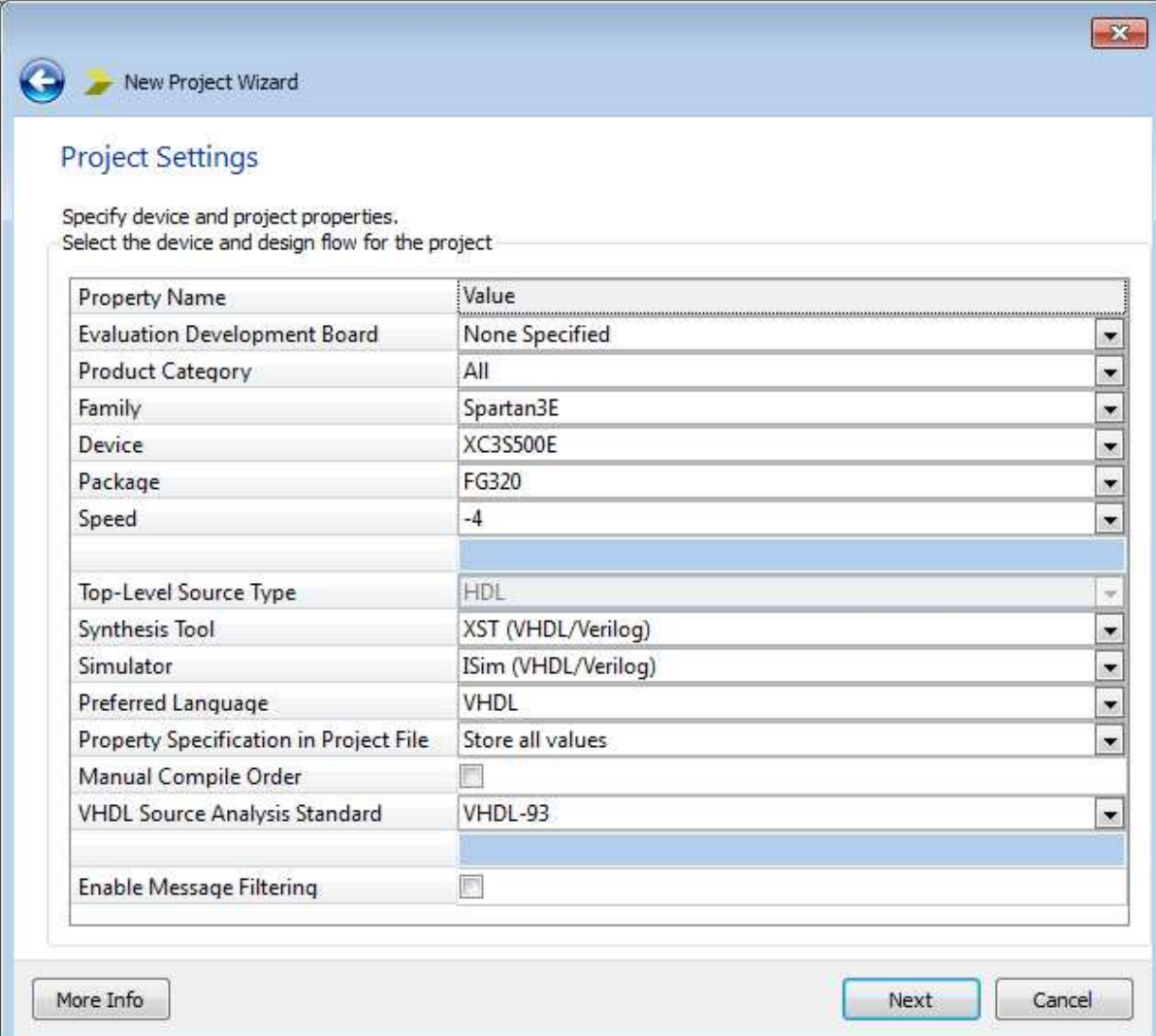
Figure 6.12: New Project Wizard, Create New Project Page.

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 6.13.



Project Settings

Specify device and project properties.
Select the device and design flow for the project

Property Name	Value
Evaluation Development Board	None Specified
Product Category	All
Family	Spartan3E
Device	XC3S500E
Package	FG320
Speed	-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	<input type="checkbox"/>
VHDL Source Analysis Standard	VHDL-93
Enable Message Filtering	<input type="checkbox"/>

More Info Next Cancel

Figure 6.13: New Project Wizard, Project Settings Page.

7. Click **Next** to move to the **Project Summary** page, which will appear as shown in Figure 6.14.

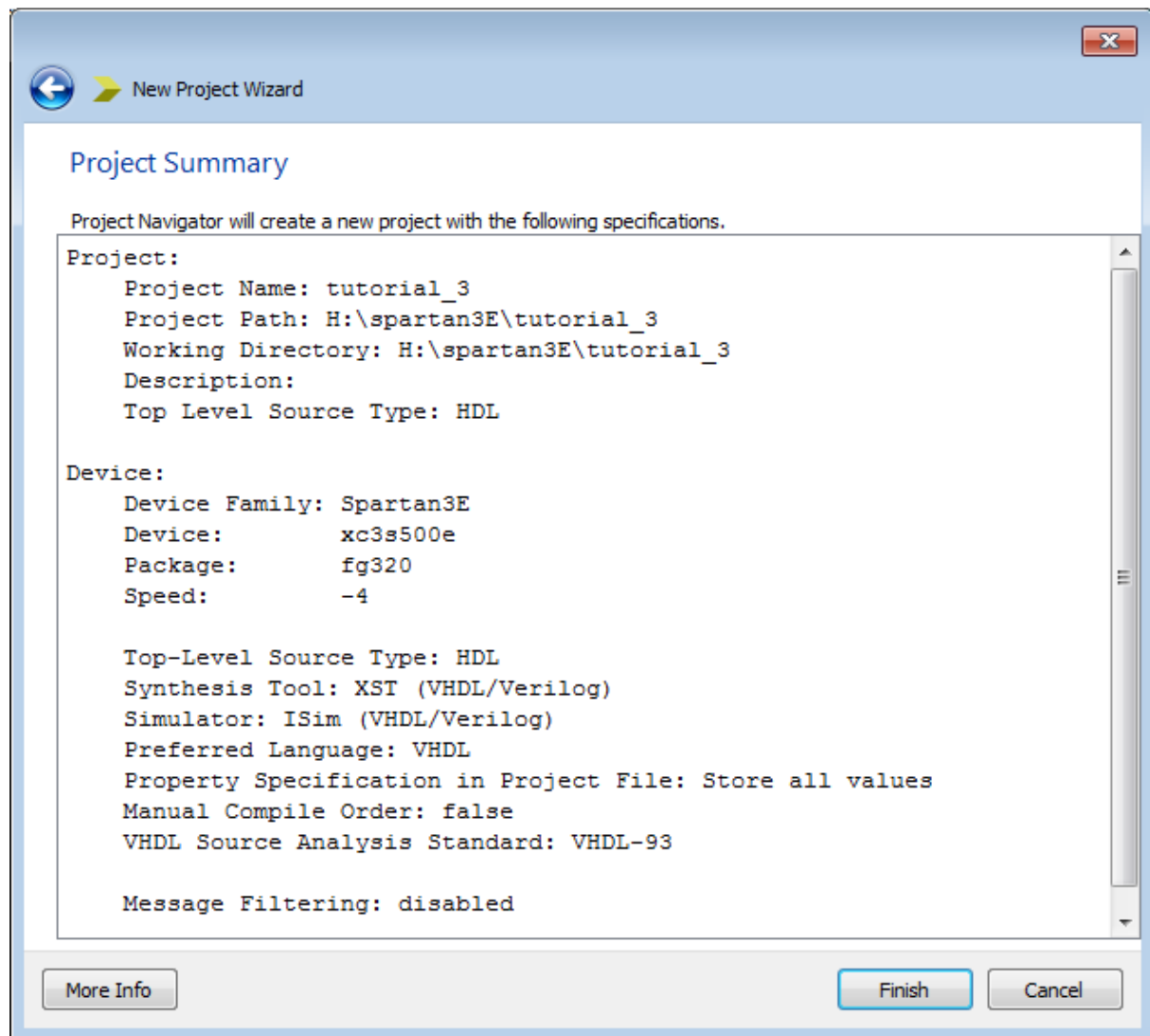


Figure 6.14: New Project Wizard, Project Summary Page.

8. Click **Finish** to exit the New Project Wizard.

6.5 Adding Source Files

1. Select **Project→Add Source** as shown in Figure 6.15. A window will appear allowing you to choose one or more files.

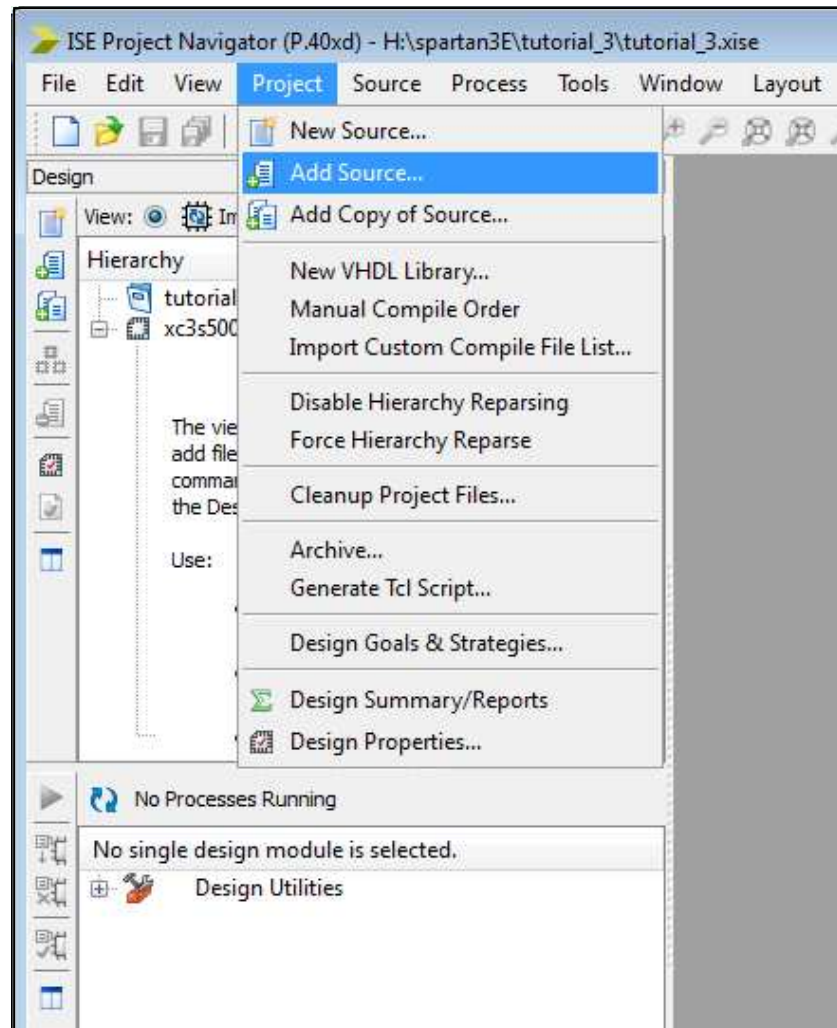


Figure 6.15: Adding a source file to the project.

2. Select **HELLO.VHD** and **kcpsm3.vhd** as shown in Figure 6.16. 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.

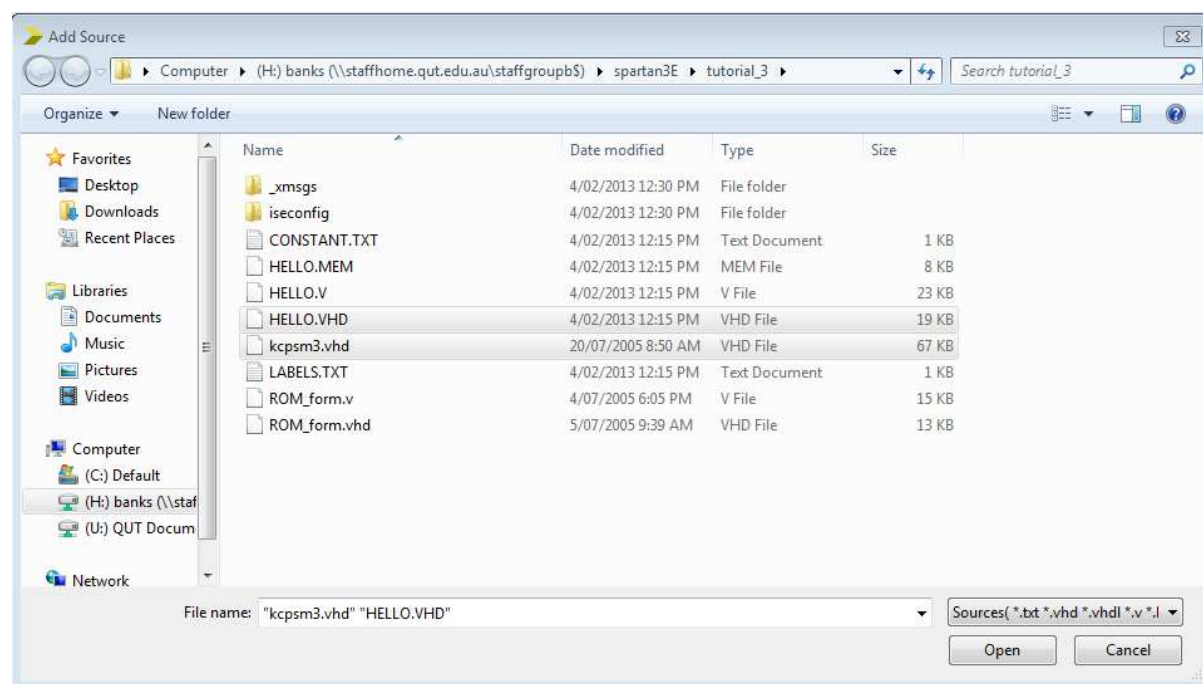


Figure 6.16: Add Source file selection window.

3. The **Adding Source Files** window will now appear as shown in Figure 6.17, showing the two files selected to be added to the project. Click OK.

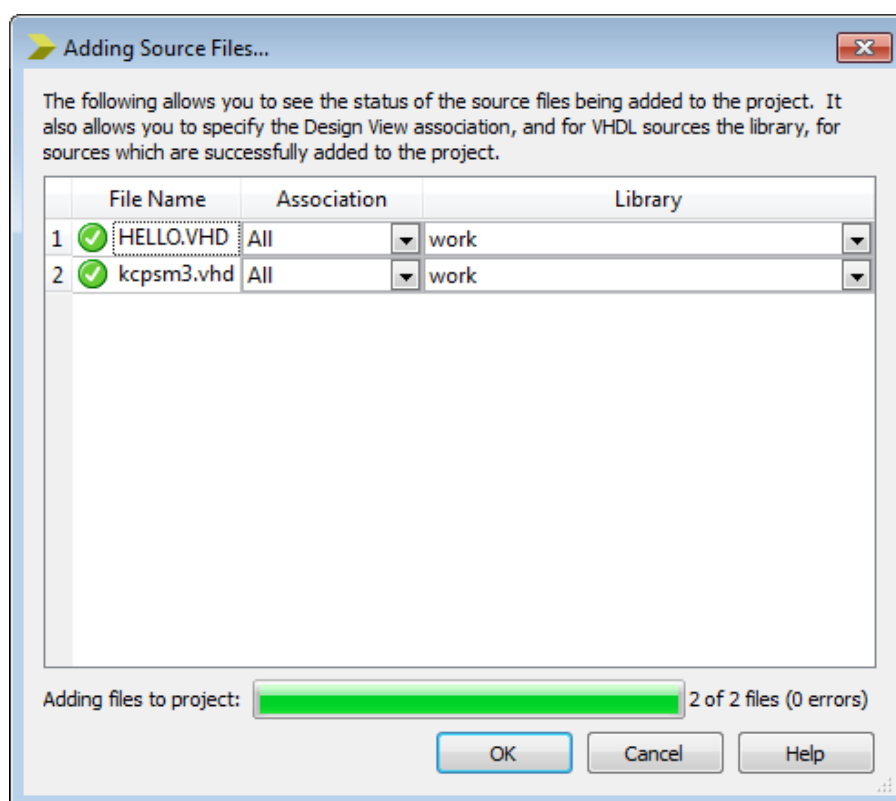


Figure 6.17: Adding Source Files window.

As shown in Figure 6.18, **kcpsm3** and **hello** will now appear in the Sources window. Double-clicking on either filename in the Sources window will display the file in a tab.

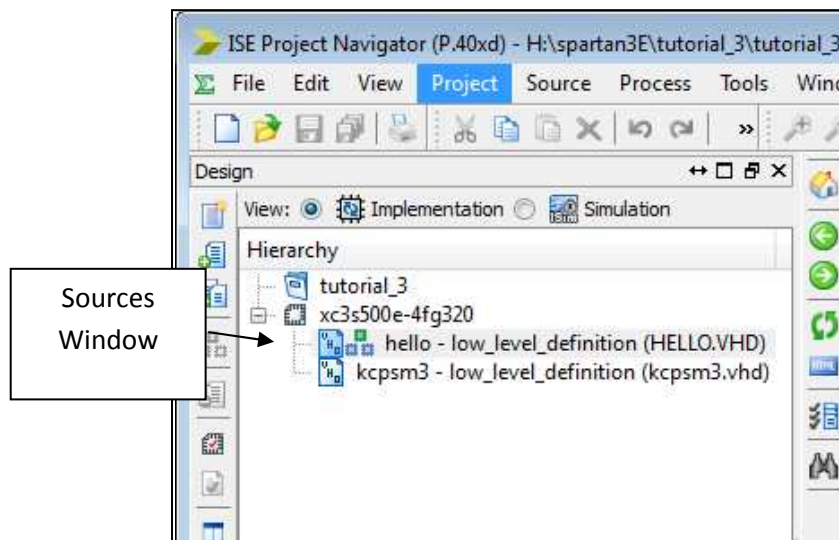


Figure 6.18: kcpsm3 and hello in the Sources window.

6.6 hello.vhd and kcpsm3.vhd – Observations

1. Double-click on **hello** in the Sources window. This will display the source code in a tab, as shown in Figure 6.19. 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.

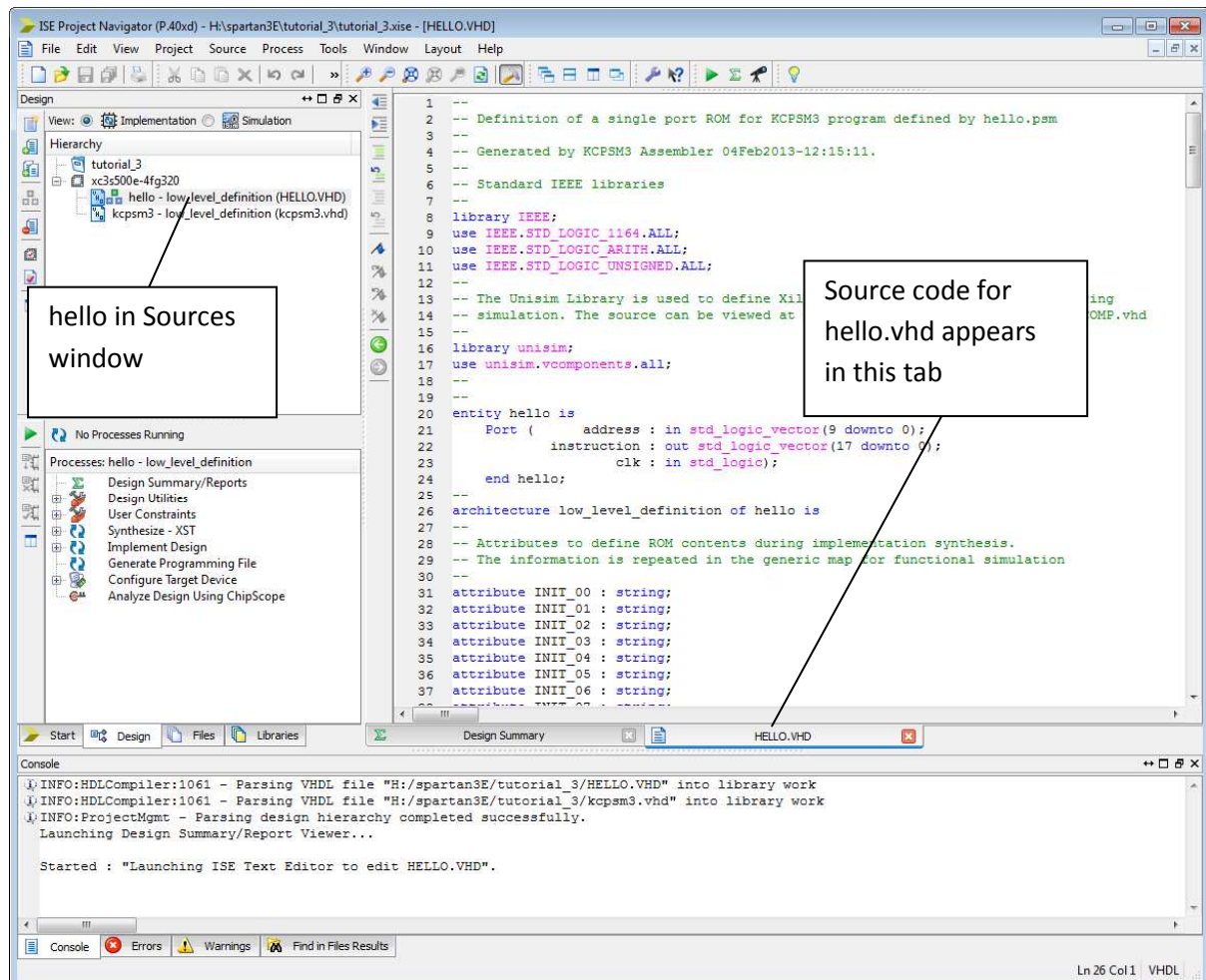


Figure 6.19: Source code for hello.vhd is displayed in a tab.

A close up of the code for the **hello** entity is shown in Figure 6.20. Note that this corresponds to the Block Memory (Program) component of Figures 3.1 and 3.3.

```

20 entity hello is
21     Port (      address : in std_logic_vector(9 downto 0);
22             instruction : out std_logic_vector(17 downto 0);
23             clk : in std_logic);
24 end hello;
```

Figure 6.20: hello 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 **kpsm3** entity is shown in Figure 6.21. Note that this corresponds to the KCPSM3 block of Figures 3.2 and 3.3.

```
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              out_port    : out std_logic_vector(7  downto 0);
83              read_strobe : out std_logic;
84              in_port     : in  std_logic_vector(7  downto 0);
85              interrupt   : in  std_logic;
86              interrupt_ack : out std_logic;
87              reset       : in  std_logic;
88              clk         : in  std_logic);
89 end kcpsm3;
```

Figure 6.21: kcpsm3 entity.

6.7 Adding a top_level Entity

VHDL code still needs to be written to tie together the **kcpsm3** and **hello** entities, and also to interface with the Spartan-3E board. We will create a file called `top_level.vhd` for this purpose.

1. Select **Project**→**New Source** as shown in Figure 6.22. The **New Source Wizard** will appear.

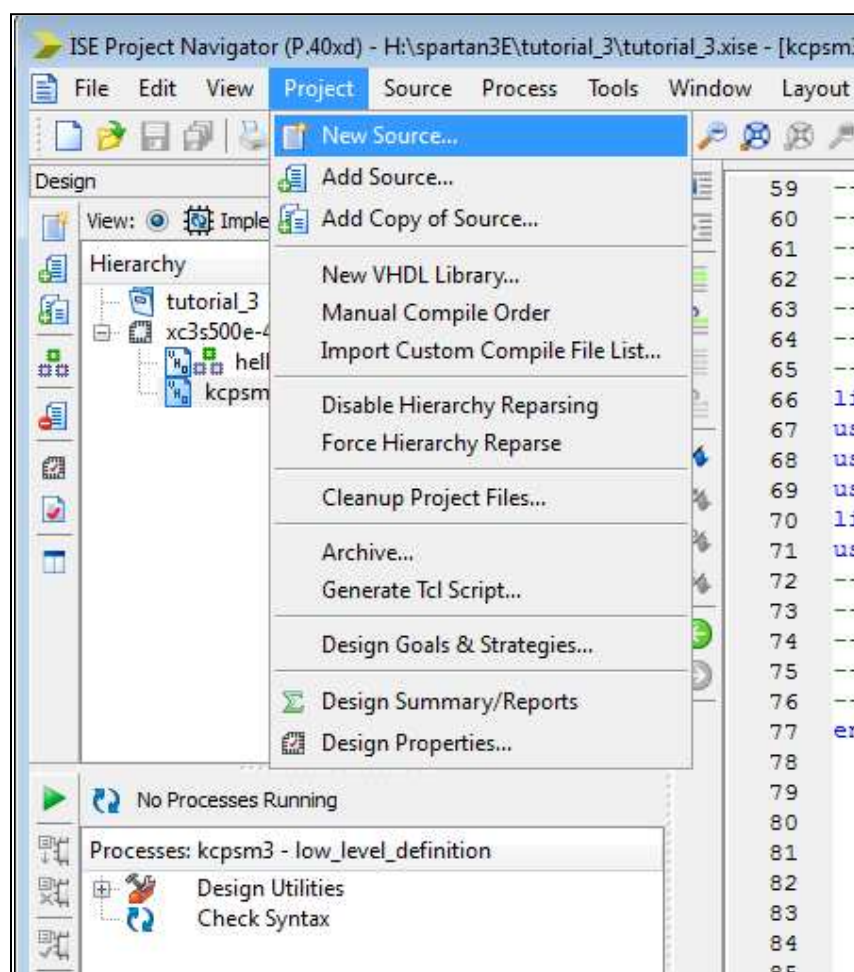


Figure 6.22: Adding a source file to the project.

2. Select Source Type as **VHDL Module**.
3. Enter the file name as **top_level**, and enter the location of the file (same as the project location entered earlier).
4. Verify that the **Add to project** box is checked. The New Source Wizard should now appear as shown in Figure 6.23.

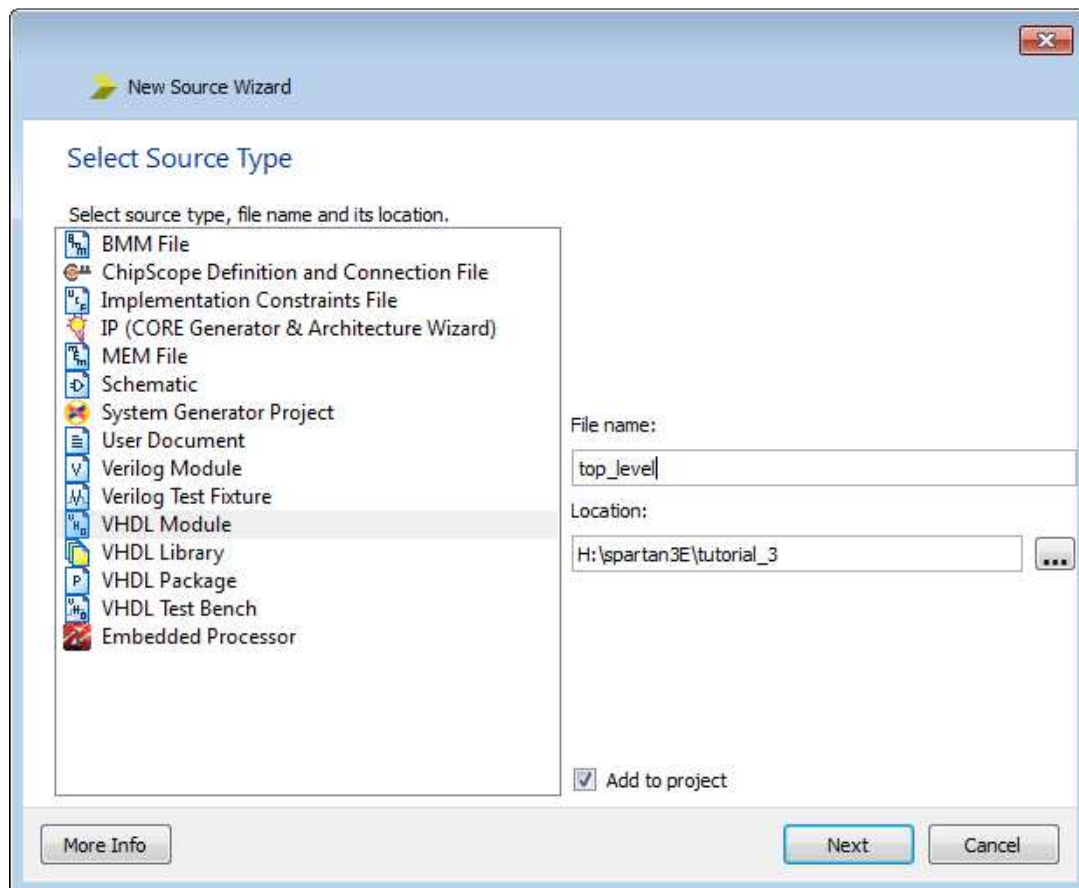
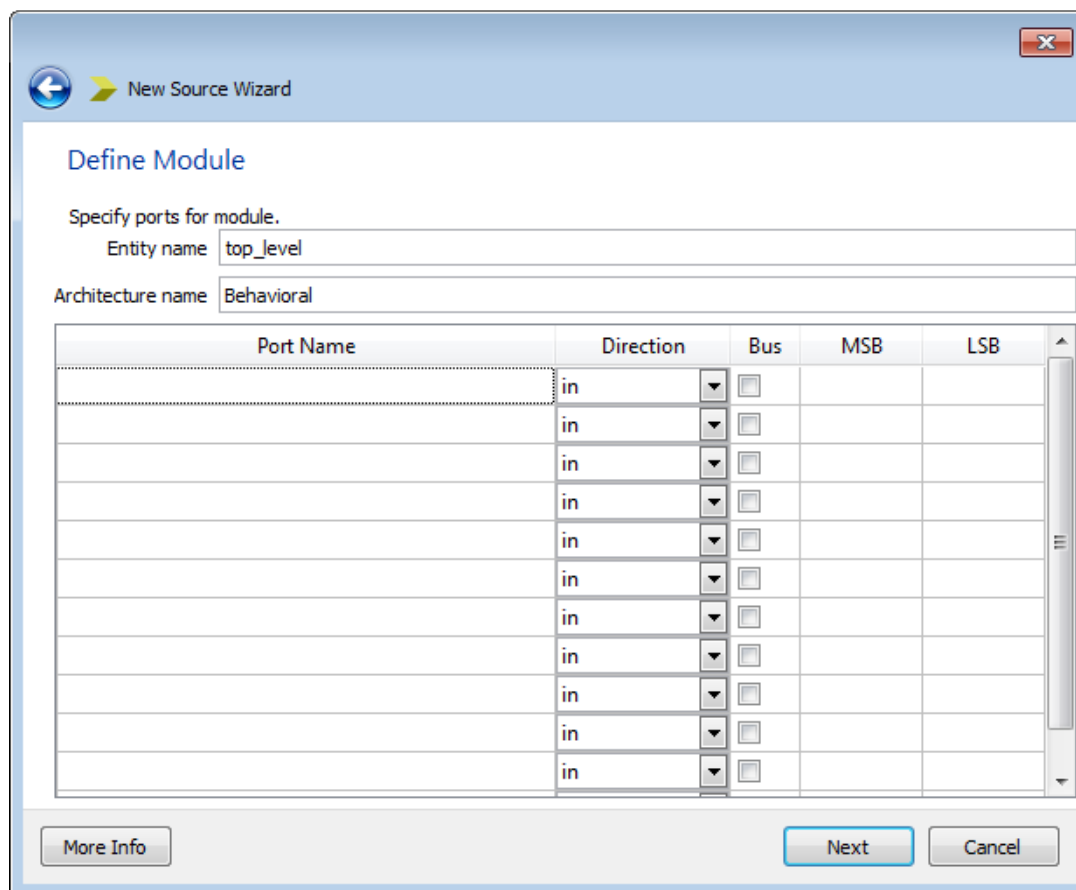


Figure 6.23: New Source Wizard, Select Source Type.

5. Click **Next** to go to the **Define Module** window.

6. The Define Module window appears as shown in Figure 6.24. We will define the ports (inputs and outputs of the design) later. Click **Next** to move to the **Summary** page, as shown in Figure 6.25.



The image shows the 'New Source Wizard' dialog box, specifically the 'Define Module' step. The window has a title bar with a back arrow, a yellow arrow icon, and the text 'New Source Wizard'. Below the title bar, the text 'Define Module' is displayed. Underneath, it says 'Specify ports for module.' followed by two input fields: 'Entity name' with the value 'top_level' and 'Architecture name' with the value 'Behavioral'. Below these fields is a table with five columns: 'Port Name', 'Direction', 'Bus', 'MSB', and 'LSB'. The table contains 12 rows, each with an empty 'Port Name' field, a dropdown menu set to 'in', an unchecked 'Bus' checkbox, and empty 'MSB' and 'LSB' fields. At the bottom of the dialog, there are three buttons: 'More Info', 'Next', and 'Cancel'.

New Source Wizard

Define Module

Specify ports for module.

Entity name

Architecture name

Port Name	Direction	Bus	MSB	LSB
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		
	in	<input type="checkbox"/>		

More Info Next Cancel

Figure 6.24: New Source Wizard, Define Module.

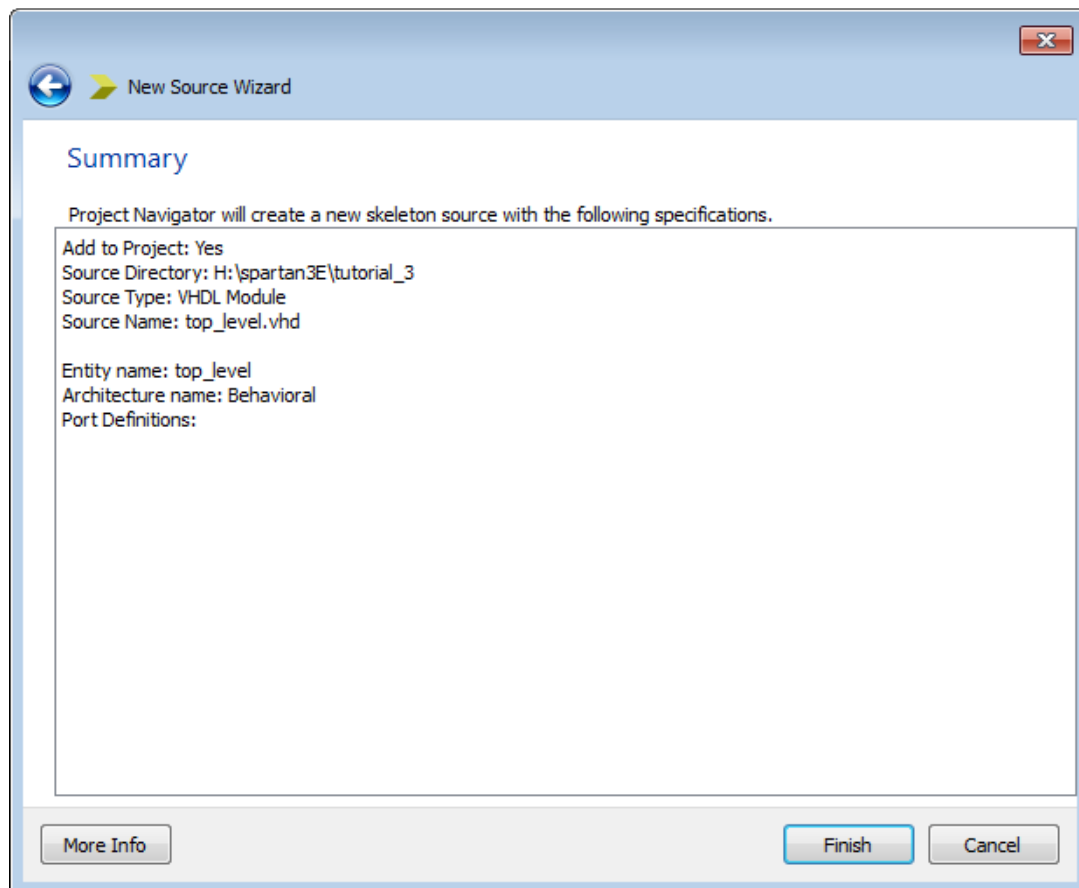


Figure 6.25: New Source Wizard, Summary.

8. Click **Finish** to exit the New Source Wizard.

As shown in Figure 6.26, **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.

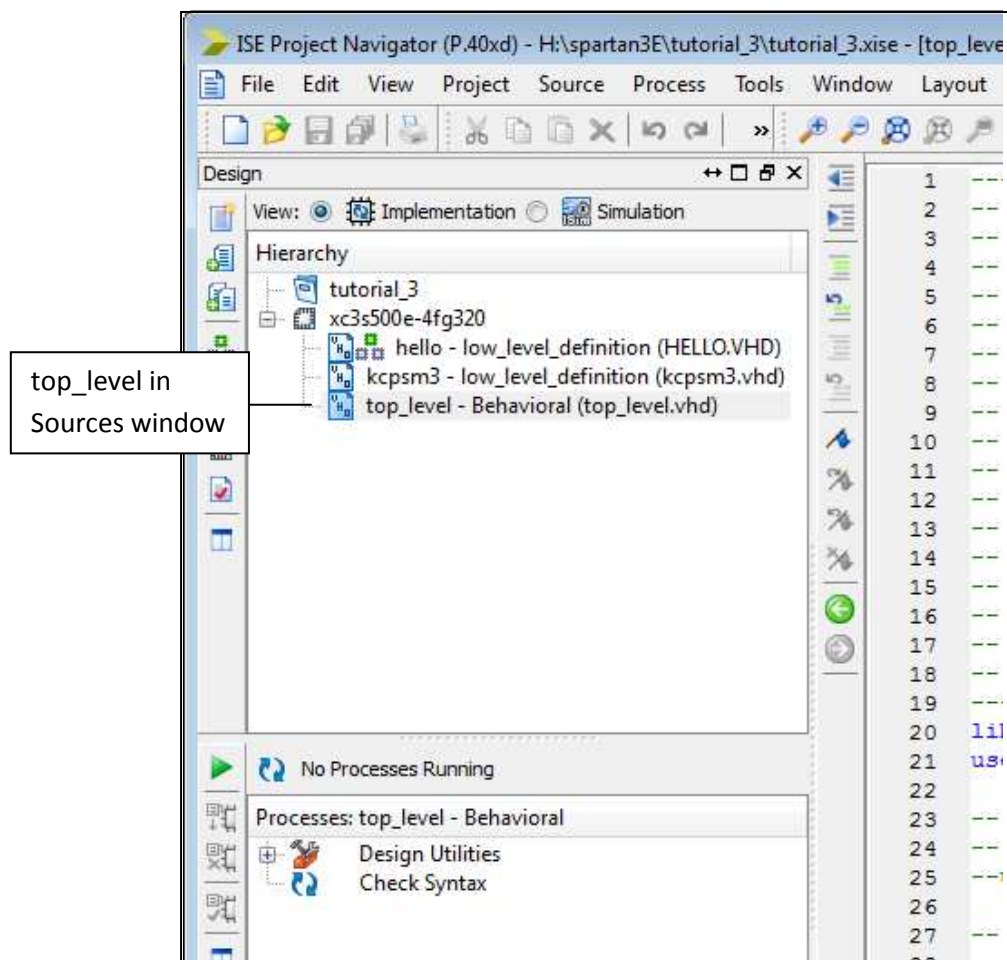


Figure 6.26: top_level in the Sources window.

6.8 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 6.27.

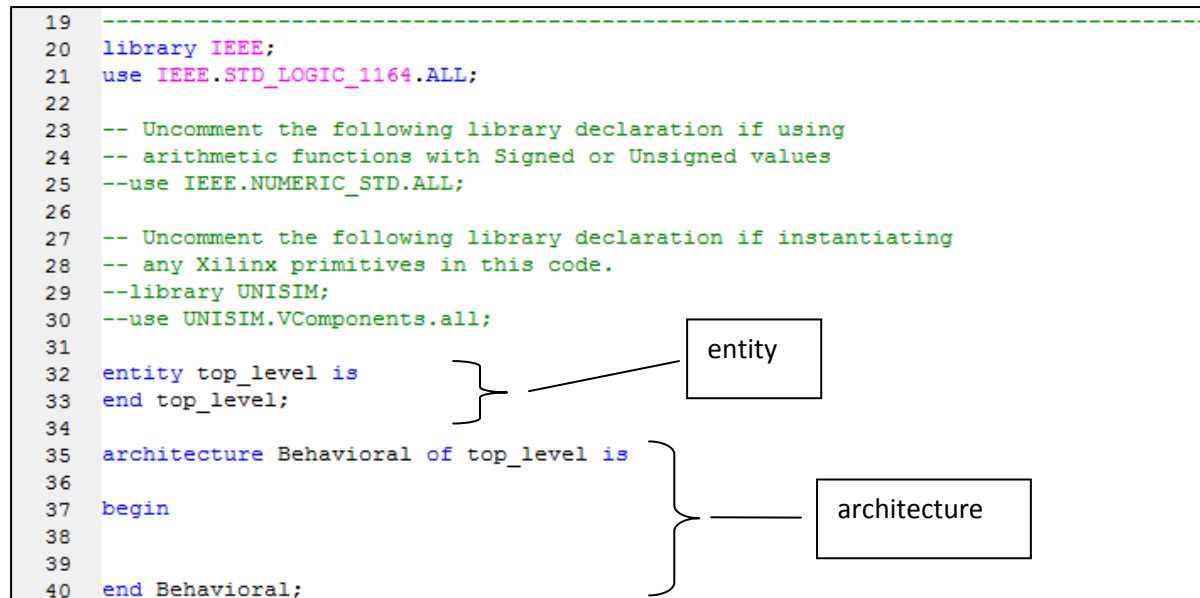


Figure 6.27: top_level.vhd, as displayed in Project Navigator, before editing.

The code in Figure 6.27 contains an **entity** and an **architecture** section. The **entity** section defines the inputs and outputs of this hardware block. The **entity and architecture** sections still need to be written for this module.

2. Replace the **entity and architecture** blocks in Figure 6.27 with the code in Figure 6.28(a)-(d). Alternatively, download the file **top_level_init.vhd**, available for download with this tutorial, and copy the code from it. The code in Figure 6.28 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 [8].

For reference the complete code for **top_level.vhd** is listed in Appendix A. Note that where VHDL 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.

```

entity top_level is
    Port (
        led : out std_logic_vector(7 downto 0);
        strataflash_oe : out std_logic;
        strataflash_ce : out std_logic;
        strataflash_we : out std_logic;
        switch : in std_logic_vector(3 downto 0);
        btn_north : in std_logic;
        btn_east : in std_logic;
        btn_south : in std_logic;
        btn_west : in std_logic;
        lcd_d : inout std_logic_vector(7 downto 4);
        lcd_rs : out std_logic;
        lcd_rw : out std_logic;
        lcd_e : out std_logic;
        clk : in std_logic);

    end top_level;
--
--
-- Start of test architecture
--
architecture Behavioral of top_level is
--
-- declaration of KCPSM3
--
    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 ROM
--
    component hello
        Port (
            address : in std_logic_vector(9 downto 0);
            instruction : out std_logic_vector(17 downto 0);
            --proc_reset : out std_logic; --JTAG Loader version
            clk : in std_logic);

        end component;
--
--
-- Signals used to connect KCPSM3 to program ROM and I/O logic
--
    signal address : std_logic_vector(9 downto 0);
    signal instruction : std_logic_vector(17 downto 0);

```

Figure 6.28(a): entity and architecture for top_level.vhd.

```

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        : std_logic := '0';
signal interrupt_ack     : std_logic;
signal kcpsm3_reset     : std_logic;
--
--
-- Signals for LCD operation
--
-- Tri-state output requires internal signals
-- 'lcd_drive' is used to differentiate between LCD and
-- StrataFLASH communications which share the same data bits.
--
signal lcd_rw_control : std_logic;
signal lcd_output_data : std_logic_vector(7 downto 4);
signal lcd_drive : std_logic;
--
-----
begin
  --
  -----
  -- LCD interface
  -----
  --
  -- The 4-bit data port is bidirectional.
  -- lcd_rw is '1' for read and '0' for write
  -- lcd_drive is like a master enable signal which prevents either the
  -- FPGA outputs or the LCD display driving the data lines.
  --
  --Control of read and write signal
  lcd_rw <= lcd_rw_control and lcd_drive;

  --use read/write control to enable output buffers.
  lcd_d <= lcd_output_data when (lcd_rw_control='0' and lcd_drive='1')
    else "ZZZZ";
  --
  -----
  -- Disable StrataFLASH
  -----
  --
  strataflash_oe <= '1';
  strataflash_ce <= '1';
  strataflash_we <= '1';
  --
  -----
  -- KCPSM3 and the program memory

```

Figure 6.28(b): entity and architecture for top_level.vhd.

```

-----
--
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 => kcpsm3_reset,
    clk => clk);

program_rom: hello
  port map(
    address => address,
    instruction => instruction,
    --proc_reset => kcpsm3_reset,    --JTAG Loader version
    clk => clk);

--
-----
-- KCPSM3 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
      -- read simple toggle switches and buttons at address 00 hex
      when "00" =>
        in_port <= btn_west & btn_north & btn_south & btn_east & switch;

      -- read LCD data at address 02 hex
      when "10" =>    in_port <= lcd_d & "0000";

      -- 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 output ports

```

Figure 6.28(c): entity and architecture for top_level.vhd.

```
-----  
--  
-- adding the output registers to the processor  
--  
output_ports: process(clk)  
begin  
  
    if clk'event and clk='1' then  
        if write_strobe='1' then  
  
            -- Write to LEDs at address 80 hex.  
            if port_id(7)='1' then  
                led <= out_port;  
            end if;  
  
            -- LCD data output and controls at address 40 hex.  
            if port_id(6)='1' then  
                lcd_output_data <= out_port(7 downto 4);  
                lcd_drive <= out_port(3);  
                lcd_rs <= out_port(2);  
                lcd_rw_control <= out_port(1);  
                lcd_e <= out_port(0);  
            end if;  
        end if;  
    end if;  
end process output_ports;  
  
end Behavioral;
```

Figure 6.28(d): entity and architecture for top_level.vhd.

After **top_level.vhd** has been edited and saved, it will now appear in the Sources window as shown in Figure 6.29. Since the **top_level** module uses the **kcp3m3** and **hello** modules as components, it has now moved above these modules in the hierarchy.

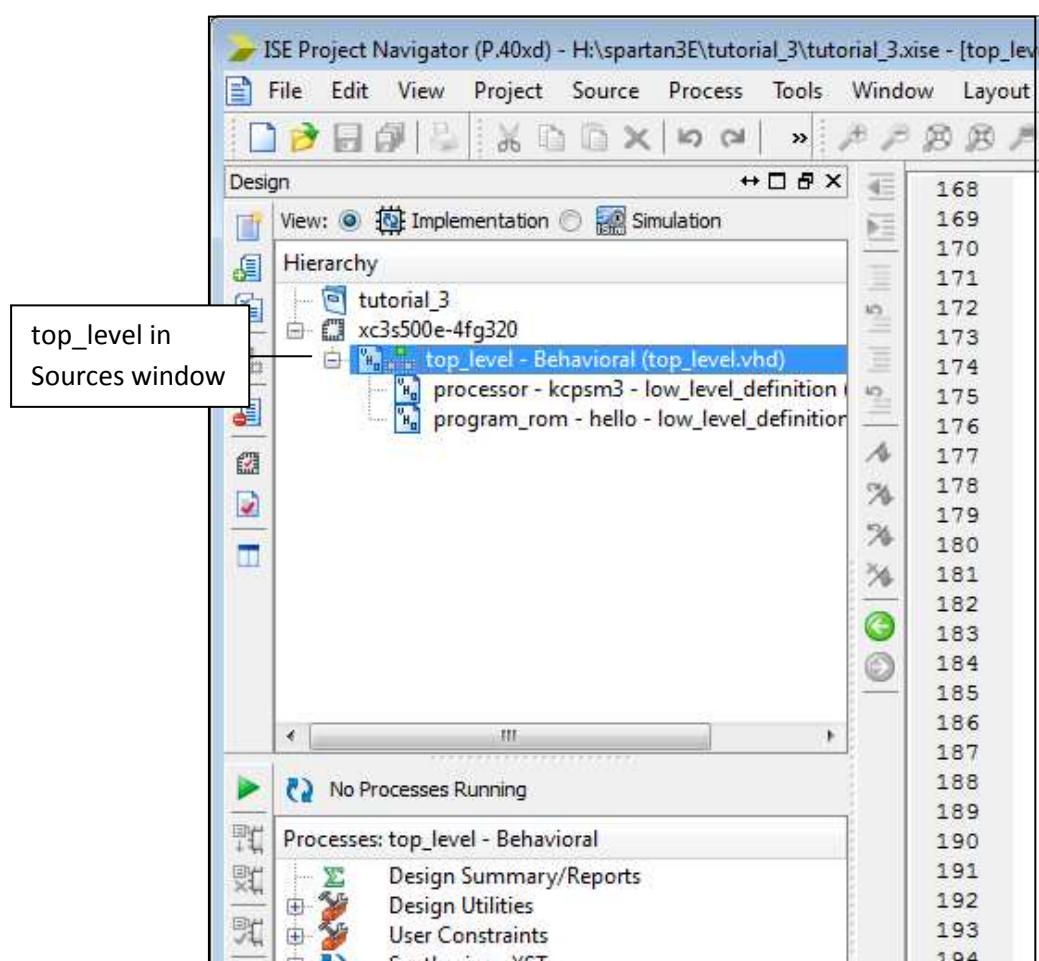


Figure 6.29: top_level in the Sources window.

6.9 top_level.vhd – Code

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

The **top_level** entity defines the inputs and outputs of the module, as shown in Figure 6.30.

```
entity top_level is
  Port (
    led : out std_logic_vector(7 downto 0);
    strataflash_oe : out std_logic;
    strataflash_ce : out std_logic;
    strataflash_we : out std_logic;
    switch : in std_logic_vector(3 downto 0);
    btn_north : in std_logic;
    btn_east : in std_logic;
    btn_south : in std_logic;
    btn_west : in std_logic;
    lcd_d : inout std_logic_vector(7 downto 4);
    lcd_rs : out std_logic;
    lcd_rw : out std_logic;
    lcd_e : out std_logic;
    clk : in std_logic);
end top_level;
```

Figure 6.30: top_level entity.

At the start of the architecture block, the **kcpsm3** and **hello** components that will be used are declared, as shown in Figure 6.31.

```
-- declaration of KCPSM3
--
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 ROM
--
component hello
  Port (
    address : in std_logic_vector(9 downto 0);
    instruction : out std_logic_vector(17 downto 0);
    --proc_reset : out std_logic; --JTAG Loader version
    clk : in std_logic);
  end component;
--
```

Figure 6.31: Component declarations.

Next, signals which connect the components of top_level to each other and the outside world, as well as some signals required for LCD operation, are declared in Figure 6.32.


```

-----
--
-- Signals used to connect KCPSM3 to program ROM 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     : std_logic := '0';
signal interrupt_ack : std_logic;
signal kcpsm3_reset  : std_logic;
--
--
-- Signals for LCD operation
--
-- Tri-state output requires internal signals
-- 'lcd_drive' is used to differentiate between LCD and
-- StrataFLASH communications which share the same data bits.
--
signal  lcd_rw_control : std_logic;
signal  lcd_output_data : std_logic_vector(7 downto 4);
signal      lcd_drive : std_logic;
--

```

Figure 6.32: Signal declarations.

The code in Figures 6.33 to 6.36 appears between the **begin** and **end** statements in the architecture block. In Figure 6.33, signal control for the LCD interface is shown. Signals are also sent to disable the StrataFlash.

```
-----  
-- LCD interface  
-----  
--  
-- The 4-bit data port is bidirectional.  
-- lcd_rw is '1' for read and '0' for write  
-- lcd_drive is like a master enable signal which prevents either the  
-- FPGA outputs or the LCD display driving the data lines.  
--  
--Control of read and write signal  
lcd_rw <= lcd_rw_control and lcd_drive;  
  
--use read/write control to enable output buffers.  
lcd_d <= lcd_output_data when (lcd_rw_control='0' and lcd_drive='1')  
    else "ZZZZ";  
--  
-----  
-- Disable StrataFLASH  
-----  
--  
strataflash_oe <= '1';  
strataflash_ce <= '1';  
strataflash_we <= '1';  
--
```

Figure 6.33: LCD control.

Next, the **kcpsm3** and **hello** components are instantiated, with ports connected to signals, as shown in Figure 6.34.

```
--  
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 => kcpsm3_reset,  
    clk => clk);  
  
program_rom: hello  
  port map(  
    address => address,  
    instruction => instruction,  
    --proc_reset => kcpsm3_reset,    --JTAG Loader version  
    clk => clk);  
--
```

Figure 6.34: Component instantiations.

The code in Figure 7.25 consists of a VHDL process. On each rising edge of the clock, depending on the **port_id**, different data will be copied into the **in_port** of the **kcpsm3**. If the **port_id** is 0x00, the values of the push buttons and switches will be copied into the **in_port**, while if the **port_id** is 0x10, the LCD data will be copied into the **in_port**.

```

-----
-- KCPSM3 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
            -- read simple toggle switches and buttons at address 00 hex
            when "00" =>
                in_port <= btn_west & btn_north & btn_south & btn_east & switch;

            -- read LCD data at address 02 hex
            when "10" =>    in_port <= lcd_d & "0000";

            -- 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;
--

```

Figure 6.35: Input ports.

The code in Figure 6.36 consists of a VHDL process. On each rising edge of the clock, depending on the **port_id**, different data will be read from the **out_port** of the **kcpsm3**. If the **port_id** is 0x80, then **out_port** will be read into **led**, while if the **port_id** is 0x40, then various bits of **out_port** will be read into LCD control signals.

```
-----  
--  
-- adding the output registers to the processor  
--  
output_ports: process(clk)  
begin  
  
    if clk'event and clk='1' then  
        if write_strobe='1' then  
  
            -- Write to LEDs at address 80 hex.  
            if port_id(7)='1' then  
                led <= out_port;  
            end if;  
  
            -- LCD data output and controls at address 40 hex.  
            if port_id(6)='1' then  
                lcd_output_data <= out_port(7 downto 4);  
                lcd_drive <= out_port(3);  
                lcd_rs <= out_port(2);  
                lcd_rw_control <= out_port(1);  
                lcd_e <= out_port(0);  
            end if;  
        end if;  
    end if;  
end process output_ports;
```

Figure 6.36: Output ports.

6.10 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 6.37.

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**.

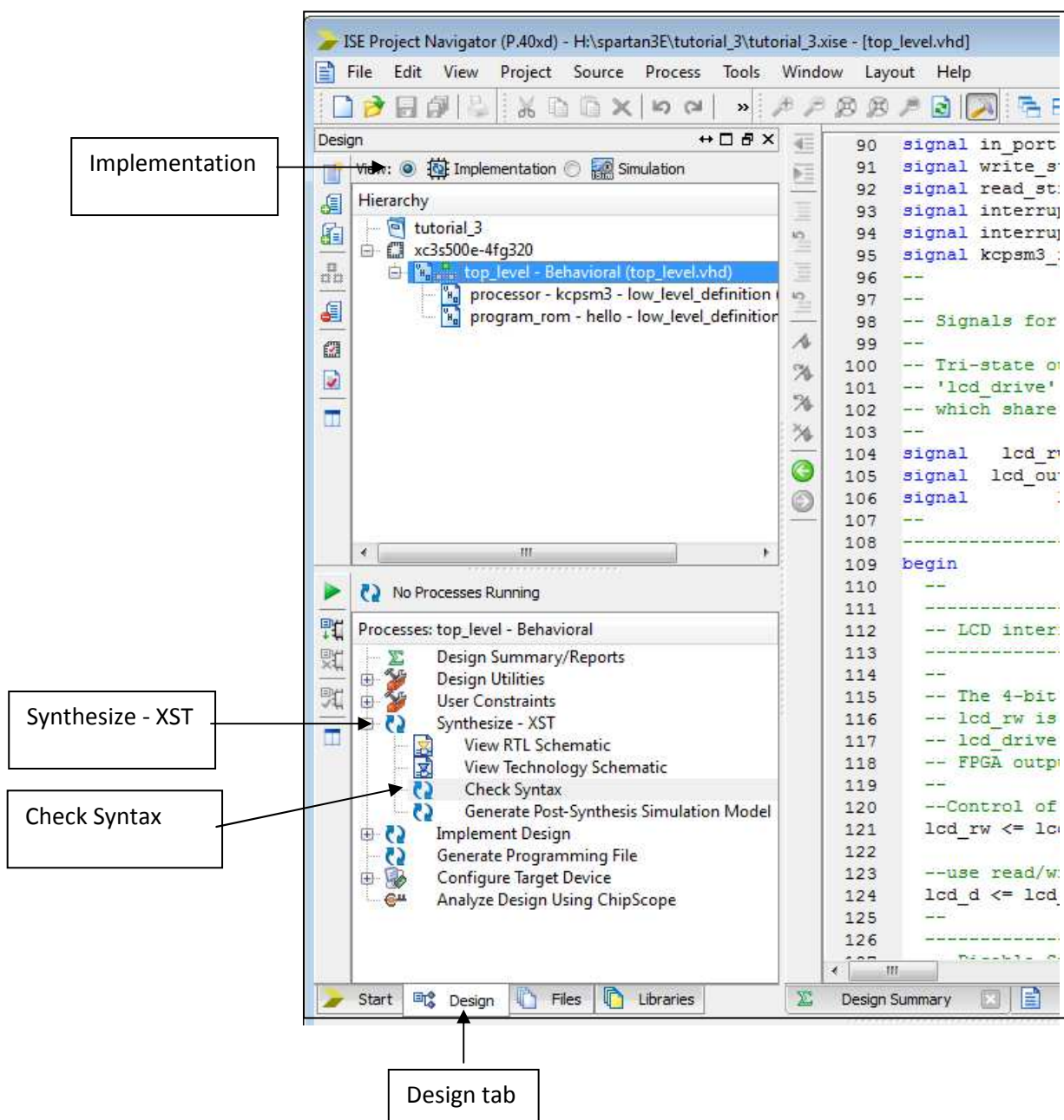


Figure 6.37: Portion of Project Navigator screen with Synthesize – XST expanded.

4. Double-click on **Check Syntax**. After some time, a green tick should appear beside **Check Syntax**, as shown in Figure 6.38. If instead, a red cross appears, this means a syntax error has been found. Any errors should be fixed before proceeding.

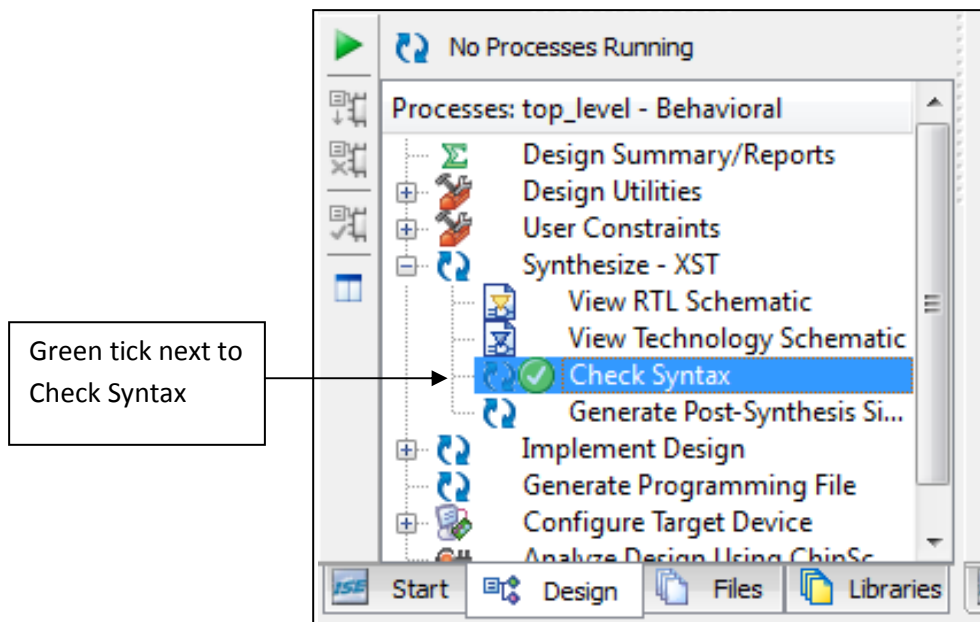


Figure 6.38: A green tick next to Check Syntax shows that no errors were found.

6.11 Pin Assignment

Recall that the VHDL code for the **top_level** entity is:

```
entity top_level is
  Port (
    led : out std_logic_vector(7 downto 0);
    strataflash_oe : out std_logic;
    strataflash_ce : out std_logic;
    strataflash_we : out std_logic;
    switch : in std_logic_vector(3 downto 0);
    btn_north : in std_logic;
    btn_east : in std_logic;
    btn_south : in std_logic;
    btn_west : in std_logic;
    lcd_d : inout std_logic_vector(7 downto 4);
    lcd_rs : out std_logic;
    lcd_rw : out std_logic;
    lcd_e : out std_logic;
    clk : in std_logic);
end top_level;
```

We wish to connect the inputs and outputs of the **top_level** entity to the switches, buttons, LEDs, clock and LCD interface on the Spartan-3E board.

For each input and output port of the top_level entity, Table 6.1 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].

Port name	Spartan-3E signal name	Description	FPGA pin
led(7)	LD7	LED	F9
led(6)	LD6	LED	E9
led(5)	LD5	LED	D11
led(4)	LD4	LED	C11
led(3)	LD3	LED	F11
led(2)	LD2	LED	E11
led(1)	LD1	LED	E12
led(0)	LD0	LED	F12
strataflash_oe	SF_OE	Strataflash chip enable	C18
strataflash_ce	SF_CE0	Strataflash chip enable	D16
strataflash_we	SF_WE	Strataflash write enable	D17
switch(3)	SW3	Slider switch	N17
switch(2)	SW2	Slider switch	H18
switch(1)	SW1	Slider switch	L14
switch(0)	SW0	Slider switch	L13
btn_north	BTN_NORTH	Push button	V4
btn_east	BTN_EAST	Push button	H13
btn_south	BTN_SOUTH	Push button	K17
btn_west	BTN_WEST	Push button	D18
lcd_d(7)	SF_D<11>	LCD data bit DB7	M15
lcd_d(6)	SF_D<10>	LCD data bit DB6	P17
lcd_d(5)	SF_D<9>	LCD data bit DB5	R16
lcd_d(4)	SF_D<8>	LCD data bit DB4	R15
lcd_rs	LCD_RS	LCD Register Select	L18
lcd_rw	LCD_RW	LCD Read/Write Control	L17
lcd_e	LCD_E	LCD Read/Write Enable	M18
clk	CLK_50MHZ	Clock	C9

Table 6.1: Input/output ports of the top_level entity; and the name, description and FPGA pin no. that the ports will be connected to.

The following steps are used to connect the inputs and outputs to the switches, buttons, LEDs, clock and LCD interface on the Spartan-3E board:

1. As shown in Figure 6.39, click on the '+' next to **User Constraints**. This will expand out to show various items, including **I/O Pin Planning (PlanAhead) – Pre-Synthesis**.

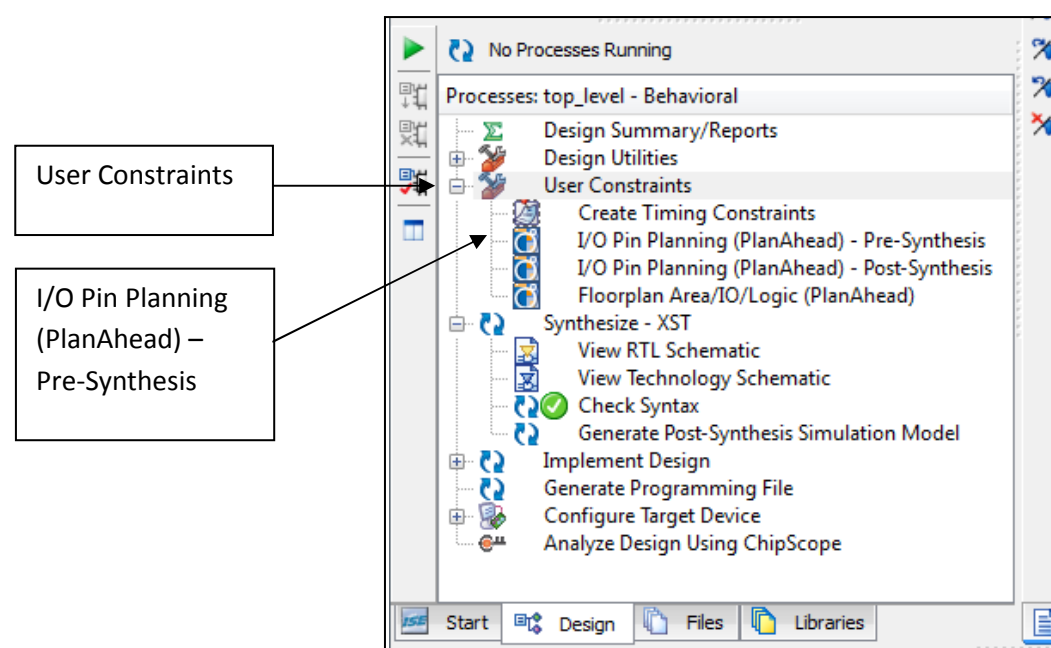


Figure 6.39: Portion of Project Navigator screen, with User Constraints expanded.

2. Double-click on **I/O Pin Planning (PlanAhead) – Pre-Synthesis**. The first time this is done, the window of Figure 6.40 will appear, asking whether it is OK to create a UCF file. Click **Yes**.

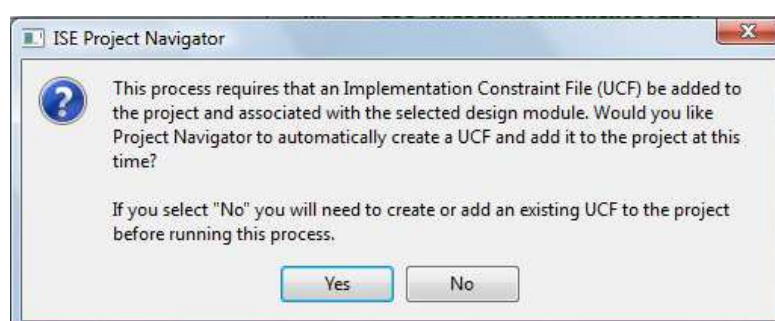


Figure 6.40: Dialog Box asking if you wish to create an Implementation Constraint File.

3. After clicking **Yes** in Figure 6.40, the **PlanAhead** window of Figure 6.41 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 6.42.

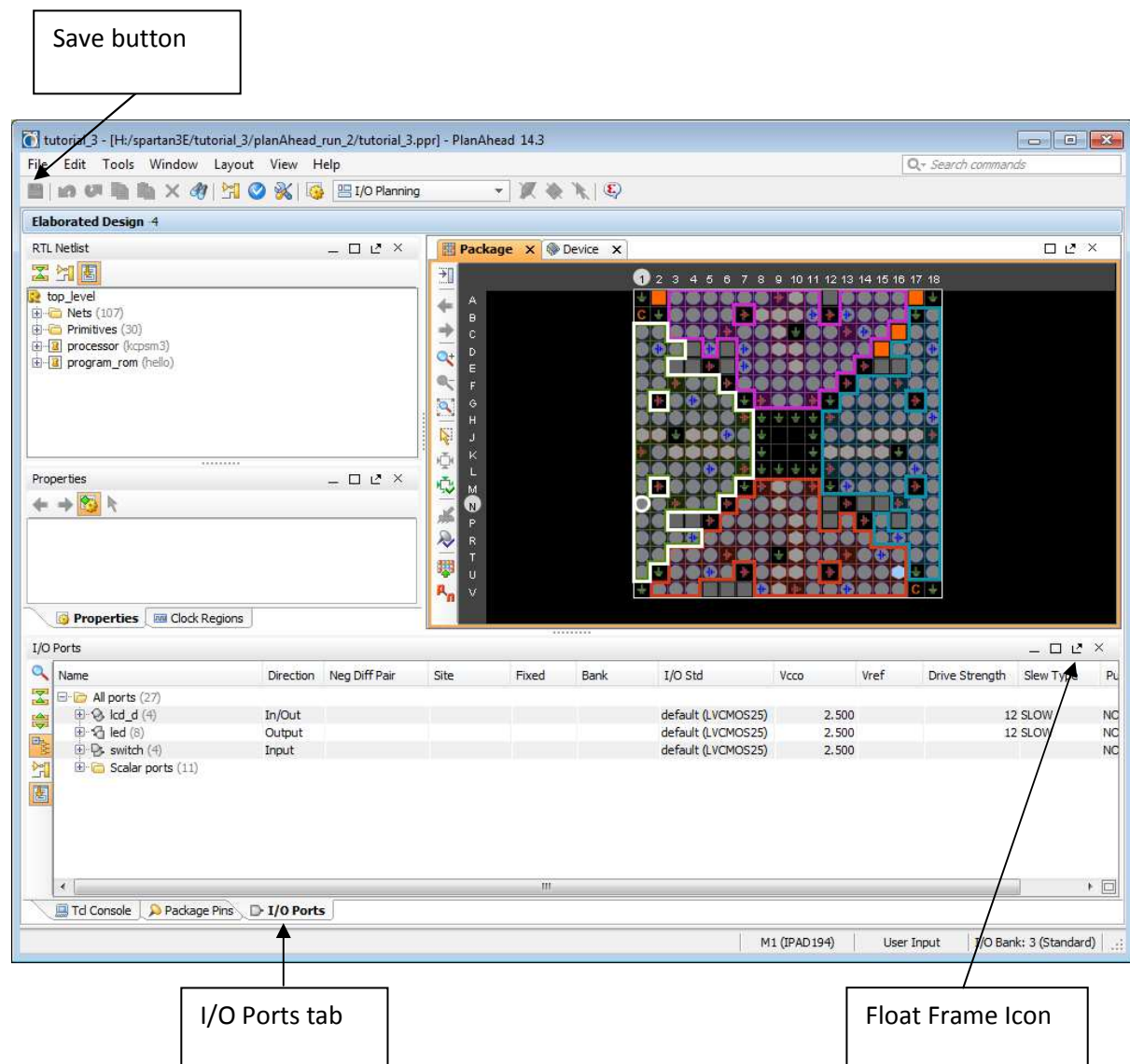


Figure 6.41: Initial appearance of PlanAhead window.

- Click on the '+' next to **lcd_d(4)**, **led(8)**, **switch(4)** and **Scalar ports(11)** in Figure 6.42. This will display all individual input/outputs as shown in Figure 5.33.

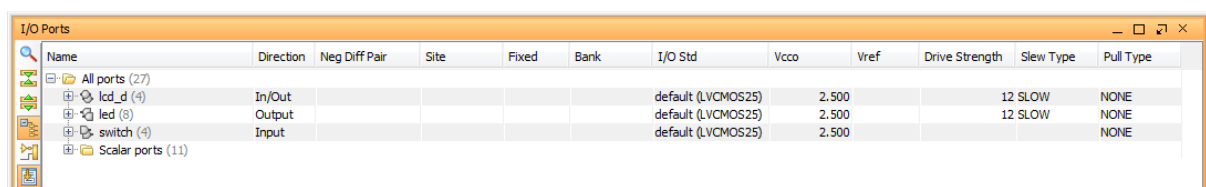


Figure 6.42: I/O Ports displayed in a separate window.

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type
All ports (27)											
lcd_d[7]	In/Out					default (LVCMOS25)	2,500		12 SLOW	NONE	
lcd_d[6]	In/Out					default (LVCMOS25)	2,500		12 SLOW	NONE	
lcd_d[5]	In/Out					default (LVCMOS25)	2,500		12 SLOW	NONE	
lcd_d[4]	In/Out					default (LVCMOS25)	2,500		12 SLOW	NONE	
led (8)	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
led[7]	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
led[6]	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
led[5]	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
led[4]	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
led[3]	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
led[2]	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
led[1]	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
led[0]	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
switch (4)	Input					default (LVCMOS25)	2,500				NONE
switch[3]	Input					default (LVCMOS25)	2,500				NONE
switch[2]	Input					default (LVCMOS25)	2,500				NONE
switch[1]	Input					default (LVCMOS25)	2,500				NONE
switch[0]	Input					default (LVCMOS25)	2,500				NONE
Scalar ports (11)											
btn_east	Input					default (LVCMOS25)	2,500				NONE
btn_north	Input					default (LVCMOS25)	2,500				NONE
btn_south	Input					default (LVCMOS25)	2,500				NONE
btn_west	Input					default (LVCMOS25)	2,500				NONE
clk	Input					default (LVCMOS25)	2,500				NONE
lcd_e	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
lcd_rs	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
lcd_rw	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
strataflash_ce	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
strataflash_oe	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	
strataflash_we	Output					default (LVCMOS25)	2,500		12 SLOW	NONE	

Figure 6.43: I/O Ports window with individual ports expanded.

5. Enter the **Site**, **I/O Std**, **Drive Strength**, **Slew Type** and **Pull Type** columns, with the values given in Table 6.2. The **Bank** and **Vcco** columns will be filled in automatically when the **Site** and **I/O Std** columns respectively are filled in.

Port	Site	I/O Std	Drive Strength	Slew Type	Pull Type
lcd_d(7)	M15	LVC MOS33	4	SLOW	NONE
lcd_d(6)	P17	LVC MOS33	4	SLOW	NONE
lcd_d(5)	R16	LVC MOS33	4	SLOW	NONE
lcd_d(4)	R15	LVC MOS33	4	SLOW	NONE
led(7)	F9	LVTTL	8	SLOW	NONE
led(6)	E9	LVTTL	8	SLOW	NONE
led(5)	D11	LVTTL	8	SLOW	NONE
led(4)	C11	LVTTL	8	SLOW	NONE
led(3)	F11	LVTTL	8	SLOW	NONE
led(2)	E11	LVTTL	8	SLOW	NONE
led(1)	E12	LVTTL	8	SLOW	NONE
led(0)	F12	LVTTL	8	SLOW	NONE
switch(3)	N17	LVTTL			PULLUP
switch(2)	H18	LVTTL			PULLUP
switch(1)	L14	LVTTL			PULLUP
switch(0)	L13	LVTTL			PULLUP
btn_east	H13	LVTTL			PULLDOWN
btn_north	V4	LVTTL			PULLDOWN
btn_south	K17	LVTTL			PULLDOWN
btn_west	D18	LVTTL			PULLDOWN
clk	C9	LVC MOS33			NONE
lcd_e	M18	LVC MOS33	4	SLOW	NONE
lcd_rs	L18	LVC MOS33	4	SLOW	NONE
lcd_rw	L17	LVC MOS33	4	SLOW	NONE
strataflash_ce	D16	LVC MOS33	4	SLOW	NONE
strataflash_oe	C18	LVC MOS33	4	SLOW	NONE
strataflash_we	D17	LVC MOS33	4	SLOW	NONE

Table 6.2: Values to enter in the I/O Ports window.

The I/O Ports window should now appear as shown in Figure 6.44.

Name	Direction	Neg Diff Pair	Site	Fixed	Bank	I/O Std	Vcco	Vref	Drive Strength	Slew Type	Pull Type
All ports (27)											
lcd_d [4]	In/Out					1 LVCMOS33*	3.300		4* SLOW	NONE	
lcd_d[7]	In/Out		M15	<input checked="" type="checkbox"/>		1 LVCMOS33*	3.300		4* SLOW	NONE	
lcd_d[6]	In/Out		P17	<input checked="" type="checkbox"/>		1 LVCMOS33*	3.300		4* SLOW	NONE	
lcd_d[5]	In/Out		R16	<input checked="" type="checkbox"/>		1 LVCMOS33*	3.300		4* SLOW	NONE	
lcd_d[4]	In/Out		R15	<input checked="" type="checkbox"/>		1 LVCMOS33*	3.300		4* SLOW	NONE	
led (8)	Output					0 LVTTTL*	3.300		8* SLOW	NONE	
led[7]	Output		F9	<input checked="" type="checkbox"/>		0 LVTTTL*	3.300		8* SLOW	NONE	
led[6]	Output		E9	<input checked="" type="checkbox"/>		0 LVTTTL*	3.300		8* SLOW	NONE	
led[5]	Output		D11	<input checked="" type="checkbox"/>		0 LVTTTL*	3.300		8* SLOW	NONE	
led[4]	Output		C11	<input checked="" type="checkbox"/>		0 LVTTTL*	3.300		8* SLOW	NONE	
led[3]	Output		F11	<input checked="" type="checkbox"/>		0 LVTTTL*	3.300		8* SLOW	NONE	
led[2]	Output		E11	<input checked="" type="checkbox"/>		0 LVTTTL*	3.300		8* SLOW	NONE	
led[1]	Output		E12	<input checked="" type="checkbox"/>		0 LVTTTL*	3.300		8* SLOW	NONE	
led[0]	Output		F12	<input checked="" type="checkbox"/>		0 LVTTTL*	3.300		8* SLOW	NONE	
switch (4)	Input					1 LVTTTL*	3.300				PULLUP*
switch[3]	Input		N17	<input checked="" type="checkbox"/>		1 LVTTTL*	3.300				PULLUP*
switch[2]	Input		H18	<input checked="" type="checkbox"/>		1 LVTTTL*	3.300				PULLUP*
switch[1]	Input		L14	<input checked="" type="checkbox"/>		1 LVTTTL*	3.300				PULLUP*
switch[0]	Input		L13	<input checked="" type="checkbox"/>		1 LVTTTL*	3.300				PULLUP*
Scalar ports (11)											
btn_east	Input		H13	<input checked="" type="checkbox"/>		1 LVTTTL*	3.300				PULLDOWN*
btn_north	Input		V4	<input checked="" type="checkbox"/>		2 LVTTTL*	3.300				PULLDOWN*
btn_south	Input		K17	<input checked="" type="checkbox"/>		1 LVTTTL*	3.300				PULLDOWN*
btn_west	Input		D18	<input checked="" type="checkbox"/>		1 LVTTTL*	3.300				PULLDOWN*
clk	Input		C9	<input checked="" type="checkbox"/>		0 LVCMOS33*	3.300				NONE
lcd_e	Output		M18	<input checked="" type="checkbox"/>		1 LVCMOS33*	3.300		4* SLOW	NONE	
lcd_rs	Output		L18	<input checked="" type="checkbox"/>		1 LVCMOS33*	3.300		4* SLOW	NONE	
lcd_rw	Output		L17	<input checked="" type="checkbox"/>		1 LVCMOS33*	3.300		4* SLOW	NONE	
strataflash_ce	Output		D16	<input checked="" type="checkbox"/>		1 LVCMOS33*	3.300		4* SLOW	NONE	
strataflash_oe	Output		C18	<input checked="" type="checkbox"/>		1 LVCMOS33*	3.300		4* SLOW	NONE	
strataflash_we	Output		D17	<input checked="" type="checkbox"/>		1 LVCMOS33*	3.300		4* SLOW	NONE	

Figure 6.44: I/O Ports window with values filled in.

- Click the **Save** button in the PlanAhead window (location of Save button is shown in Figure 6.41), to save the entered pins. The PlanAhead window can be closed at this stage.

6.12 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 6.45, click on the '+' next to **Implement Design**. This will expand out to show the **Translate**, **Map** and **Place & Route** stages.

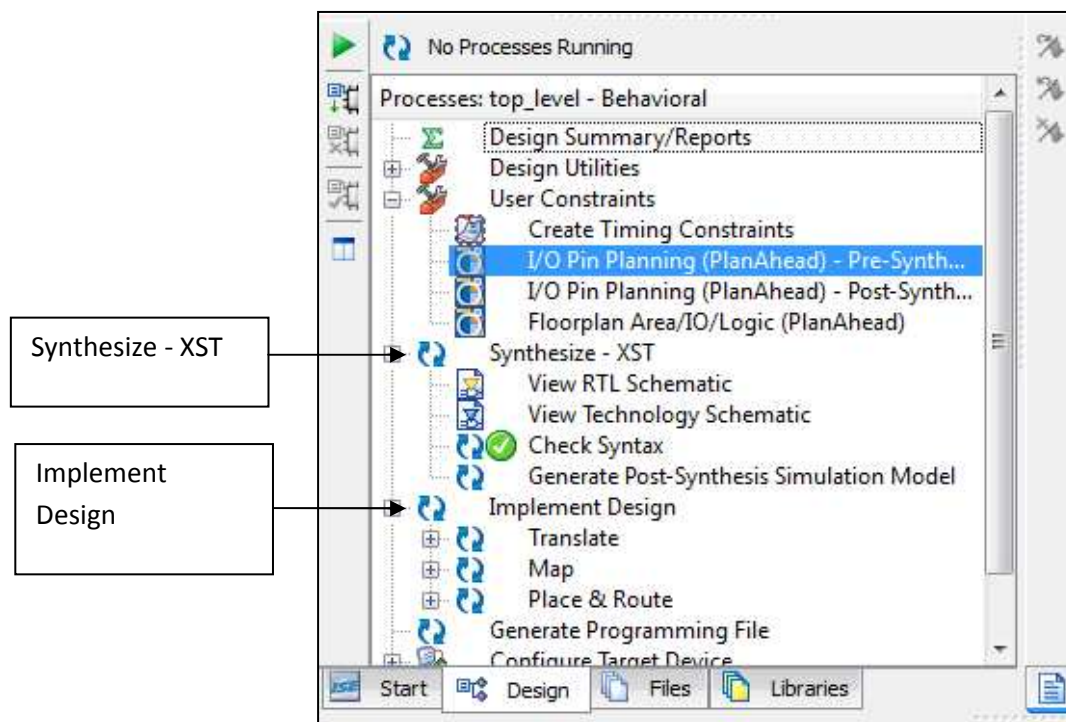


Figure 6.45: Portion of Project Navigator screen, with Implement Design expanded.

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 6.46.

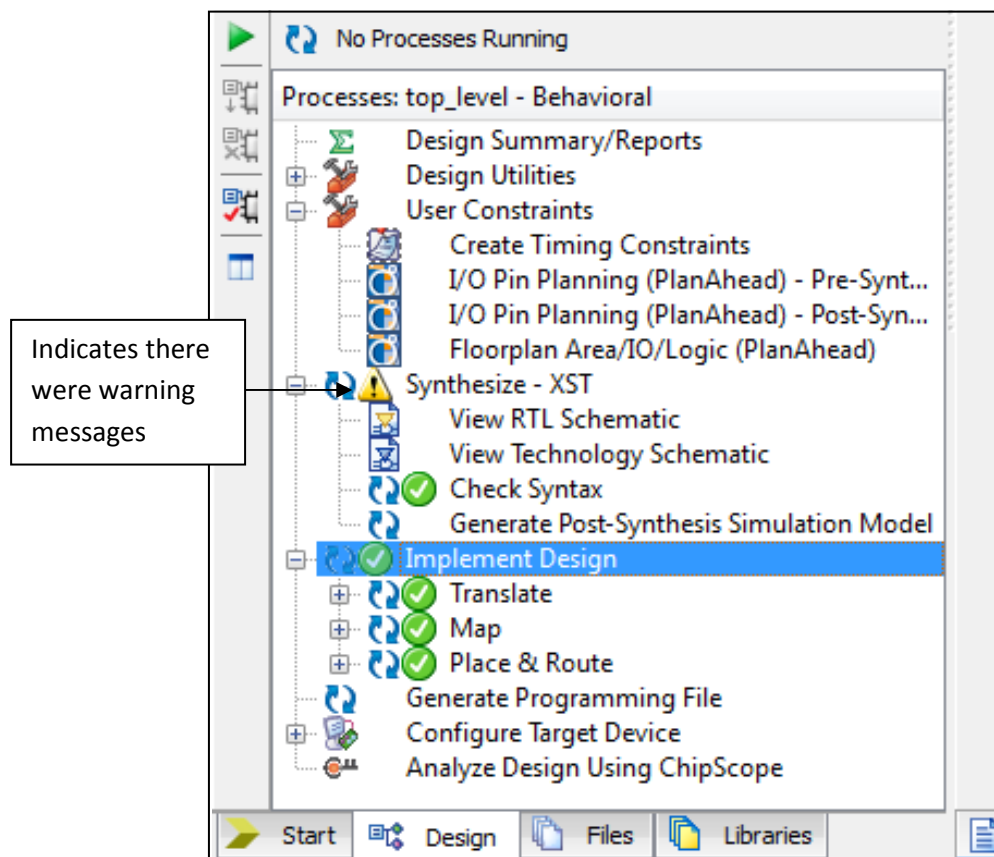


Figure 6.46: Portion of Project Navigator screen, with Implement Design expanded, after Translate, Map and Place & Route have successfully been run.

6.13 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 6.47, click on the '+' next to **Configure Target Device**. This will expand out to show the **Manage Configuration Project (iMPACT)** option.

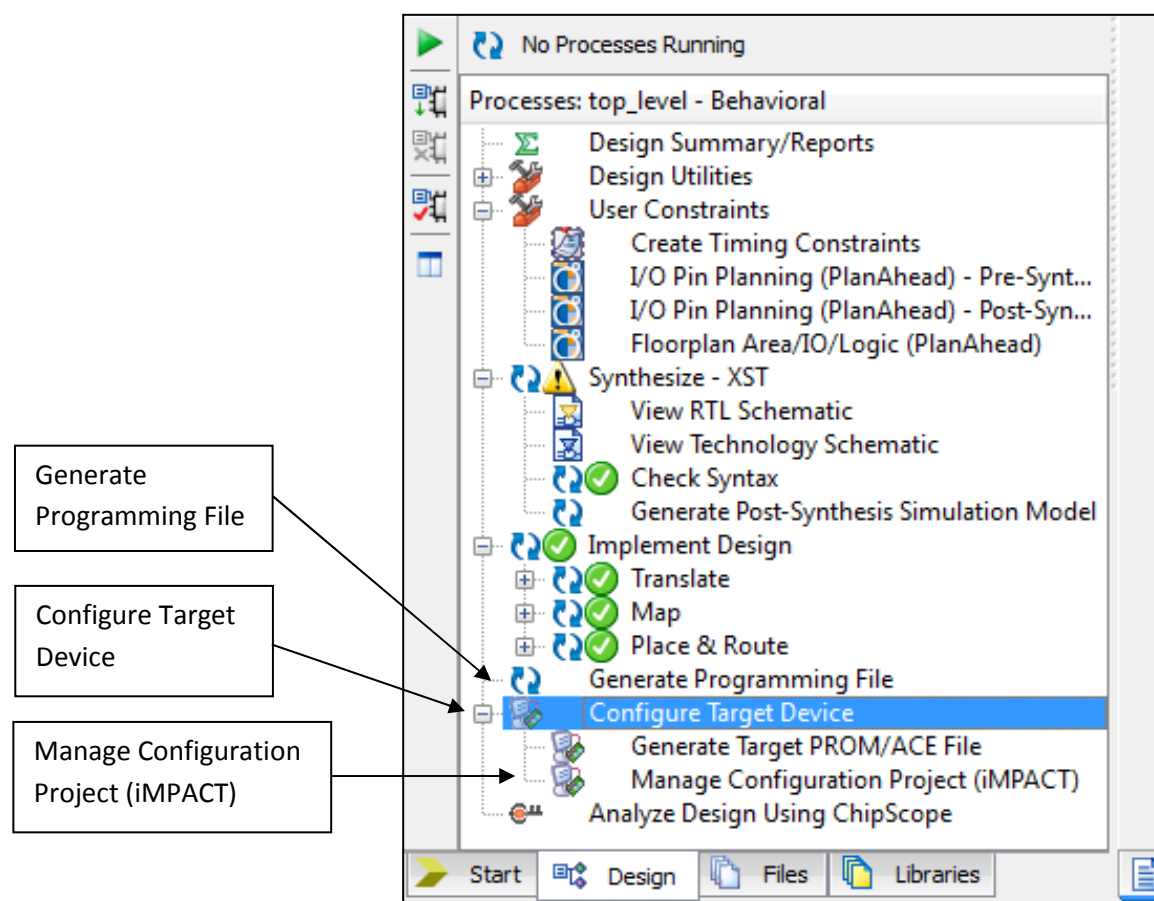


Figure 6.47: Portion of Project Navigator screen, with Implement Design expanded.

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 6.48.
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.

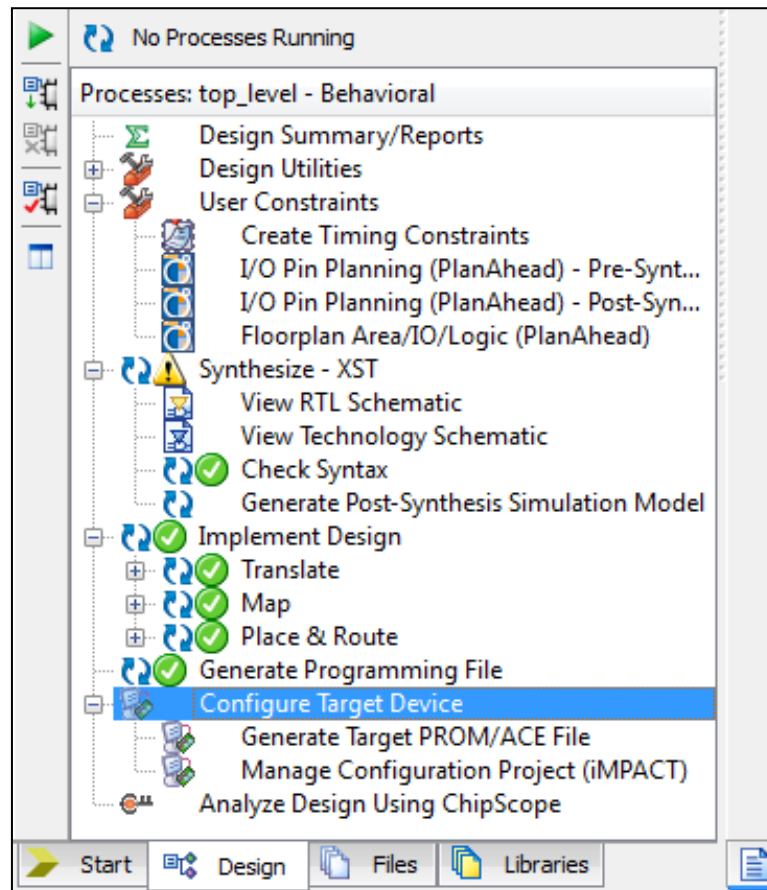


Figure 6.48: Portion of Project Navigator screen, after Generate Programming File has successfully been run.

4. Double-click on **Manage Configuration Project (iMPACT)**. The iMPACT window should appear as shown in Figure 6.49.

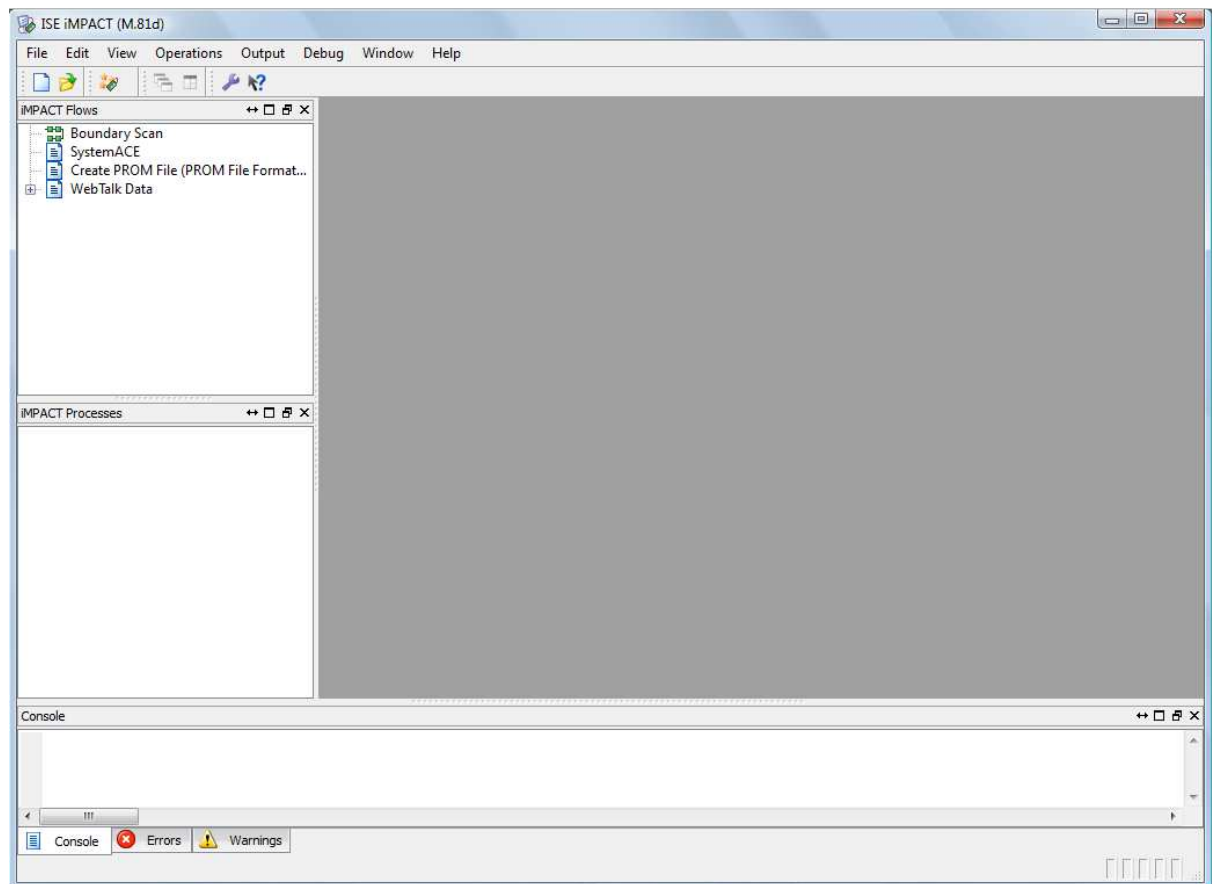


Figure 6.49: The initial iMPACT window.

5. Double-click on Boundary Scan as shown in Figure 6.50. The message “Right click to Add Device or Initialize JTAG Chain” should appear to the right.

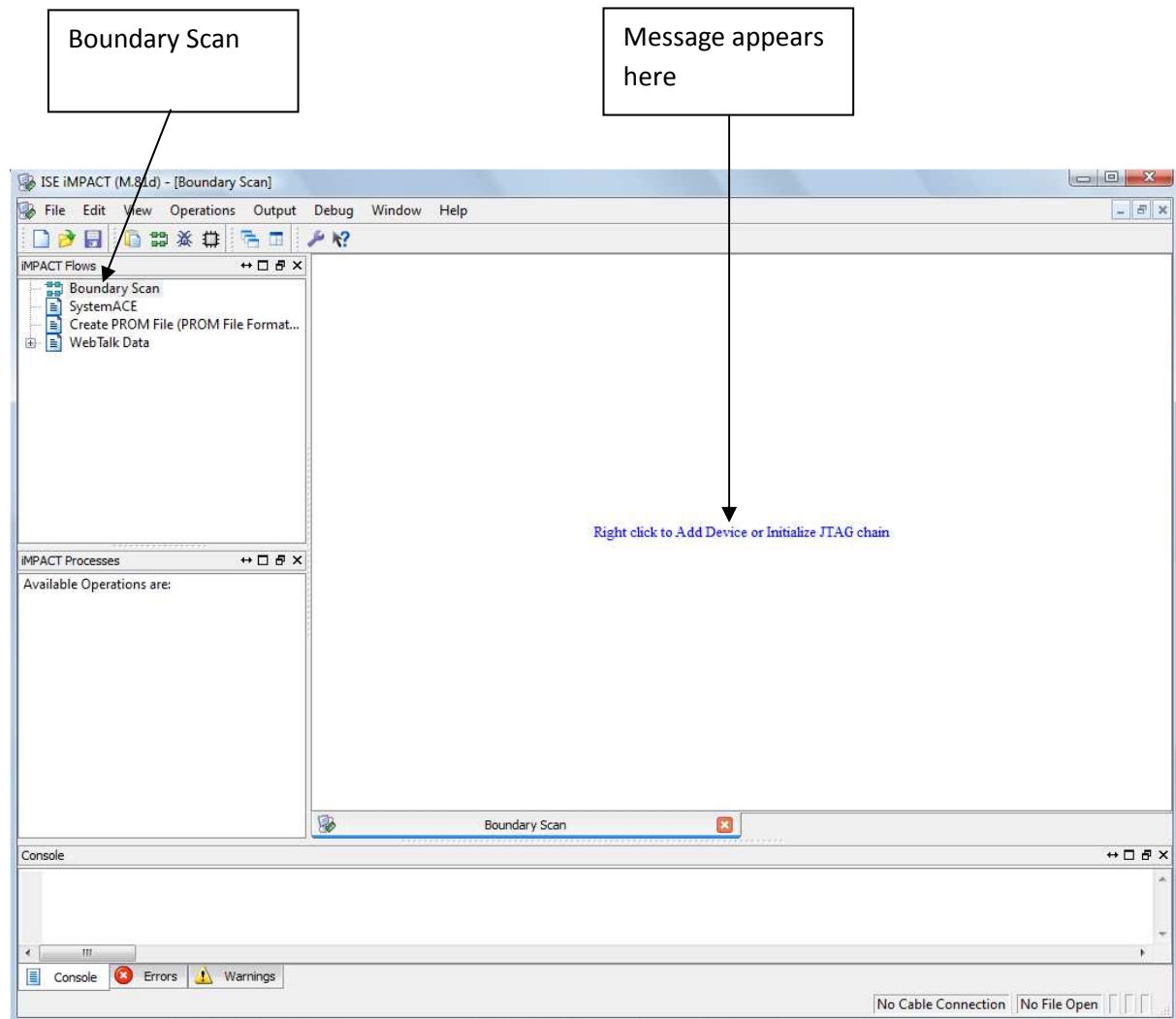


Figure 6.50: iMPACT window, after double-clicking on Boundary Scan.

6. Right click on the text “Right click to Add Device or Initialize JTAG Chain”, and select **Initialise Chain**, as shown in Figure 6.51.

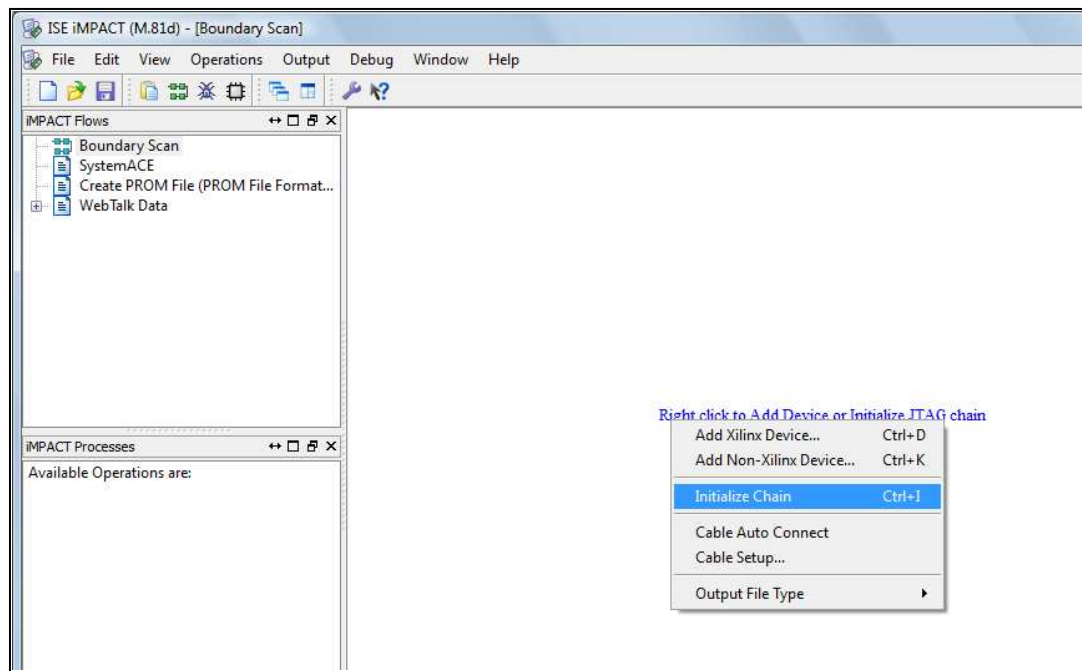


Figure 6.51: iMPACT window, showing Initialize Chain selected.

- After a while, a picture of a “chain” should appear, along with the message **Identify Succeeded** in a blue box (Figure 6.52). The first chip, the **xc3s500e**, is the FPGA chip that we wish to program. The other two, **xc404s** 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 6.52. Click **Yes**.

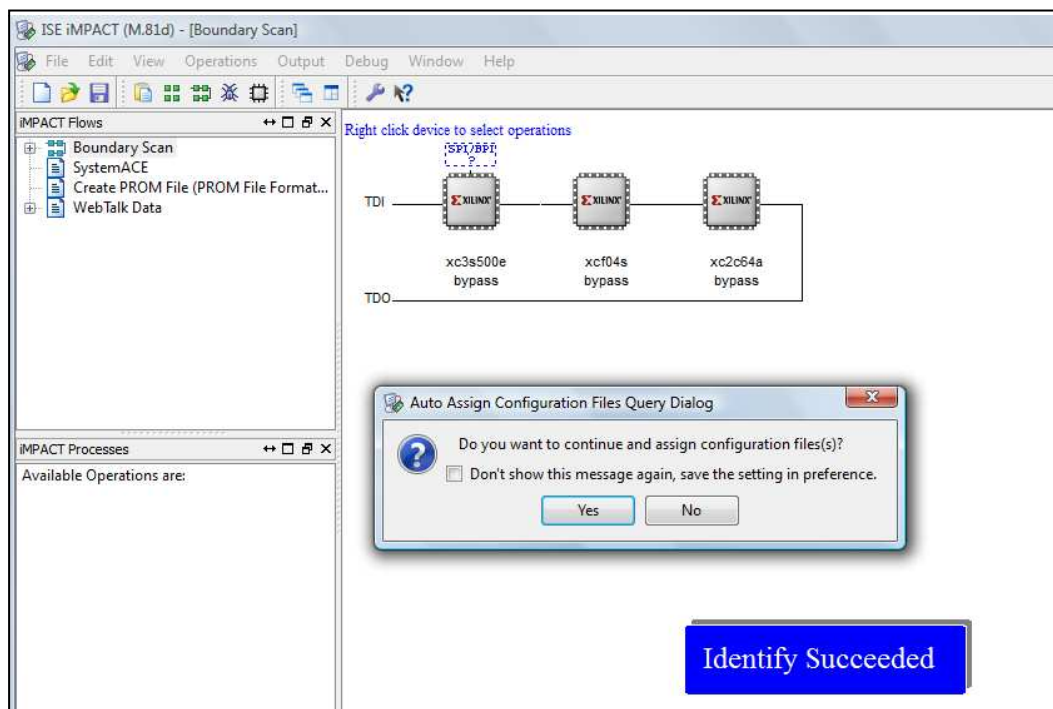


Figure 6.52: iMPACT window, assign configuration files.

8. The **Assign New Configuration File** window will appear (Figure 6.53). Select the file “top_level.bit”, and click **Open**. This associates the file top_level.bit with the xc3s500e.

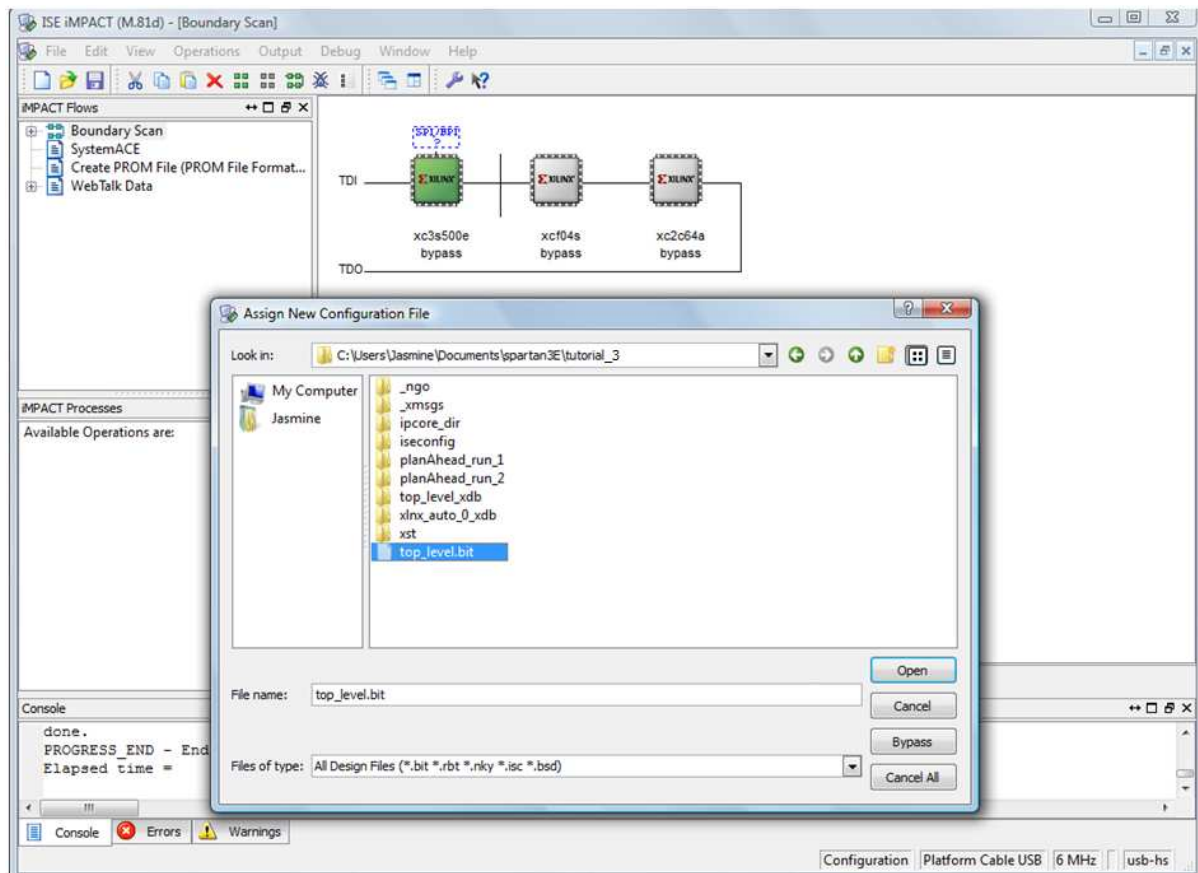


Figure 6.53: iMPACT window, assigning the configuration file for the xc3e500e.

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 6.54). This is not needed for this design. Click **No**.

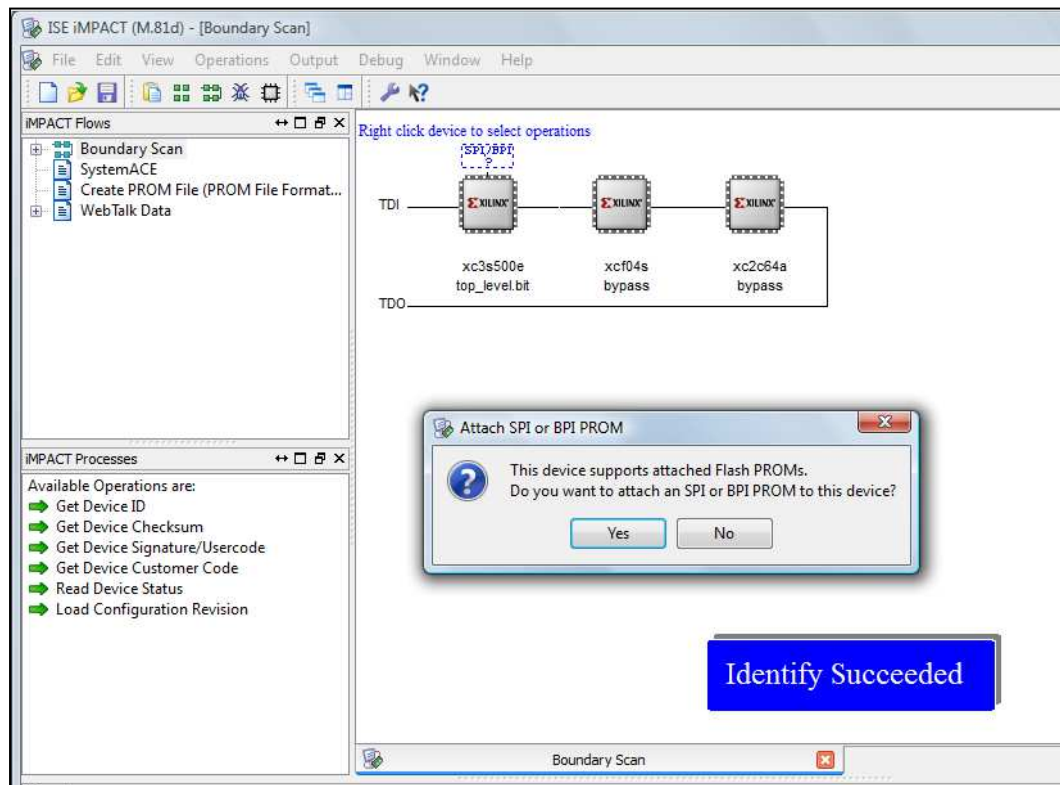


Figure 6.54: iMPACT window, dialog box asking if we wish to attach an SPI or BPI PROM.

10. The **Assign New Configuration File** window will appear again (Figure 6.55). In this case click **Bypass**. This ensures that the xcf04s is bypassed.

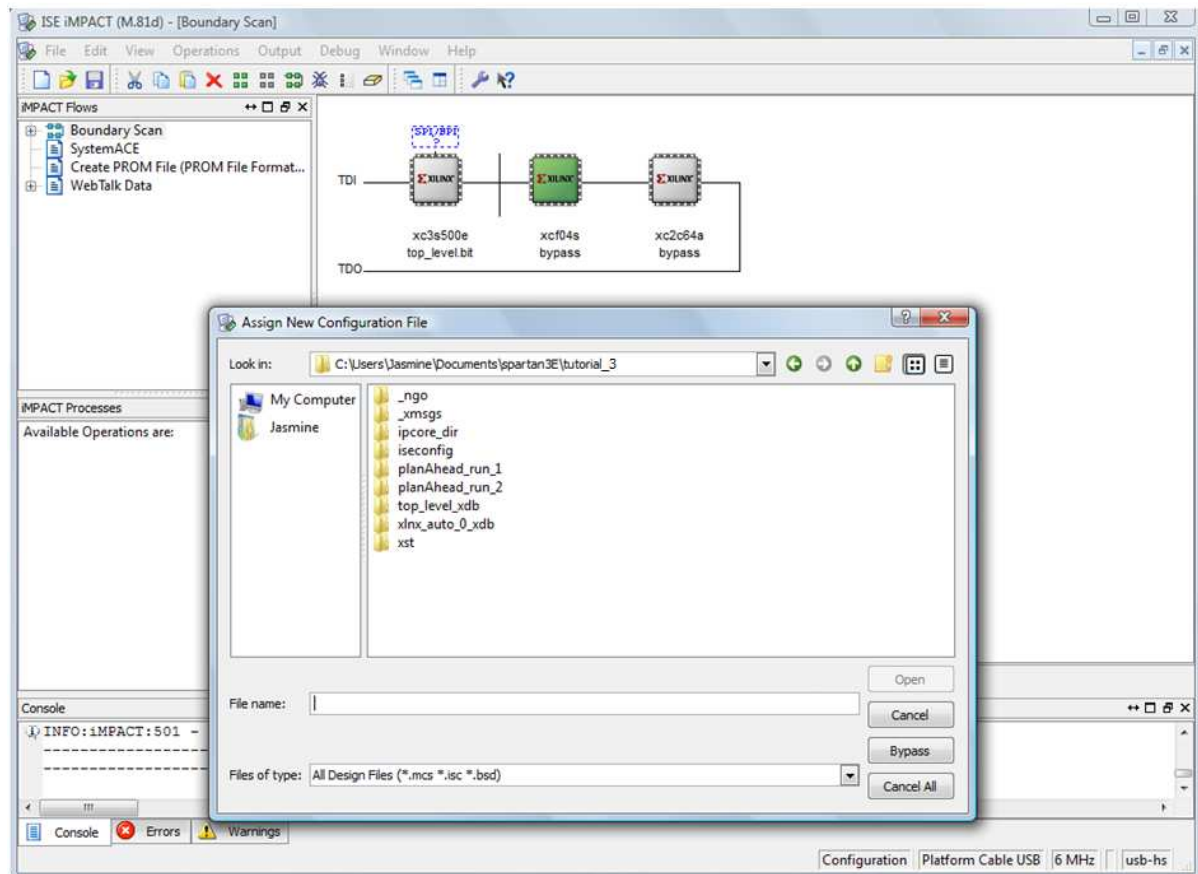


Figure 6.55: iMPACT window, bypassing the xcf04s.

11. The **Assign New Configuration File** window will appear yet again (Figure 6.56). Again click **Bypass**. This ensures that the xc2c64a is bypassed.

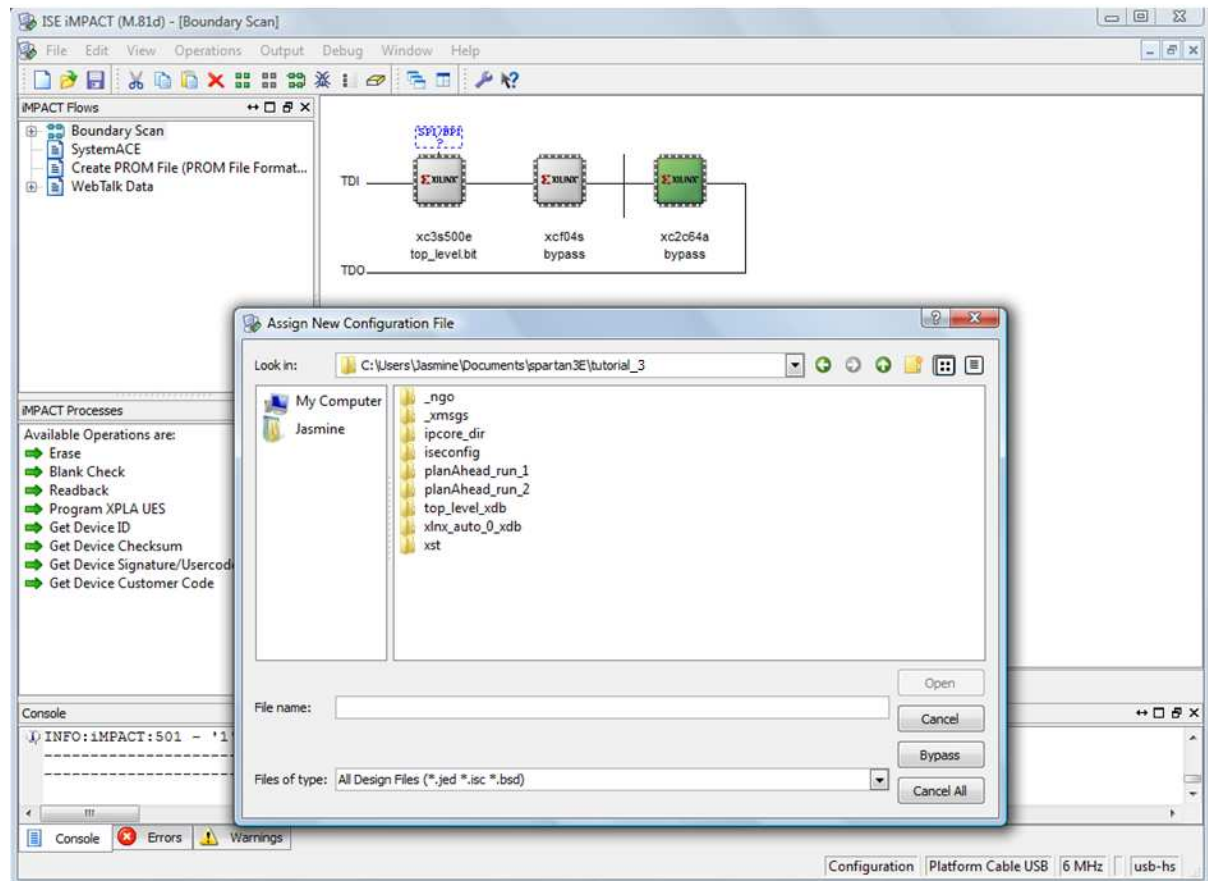


Figure 6.56: iMPACT window, bypassing the xc2c64a.

12. A window entitled **Device Programming Properties – Device 1 Programming Properties** will appear (Figure 6.57). Click **OK**.

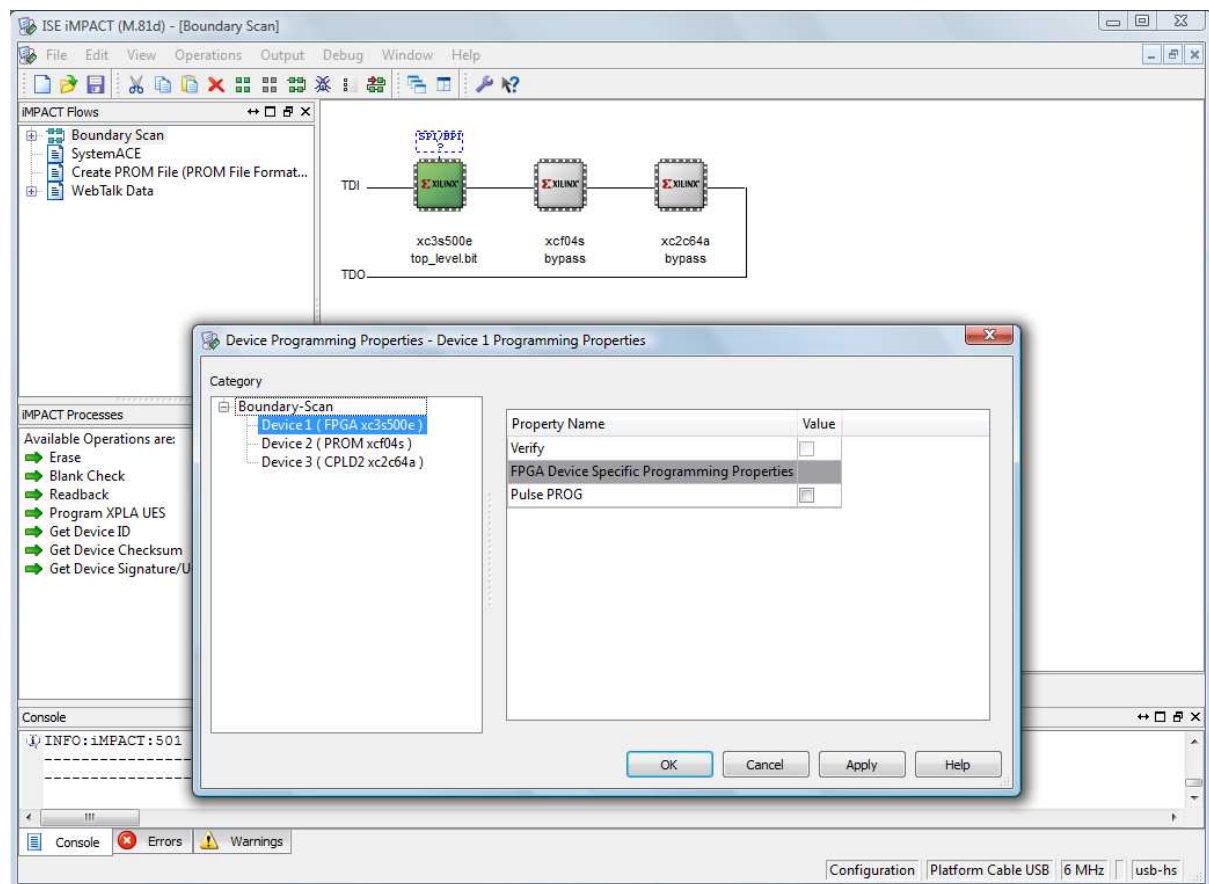


Figure 6.57: iMPACT window, Device Programming Properties dialog box.

13. The iMPACT window should now appear as shown in Figure 6.58. Right click on the xc3e500e chip (Figure 6.59) and select **Program**.

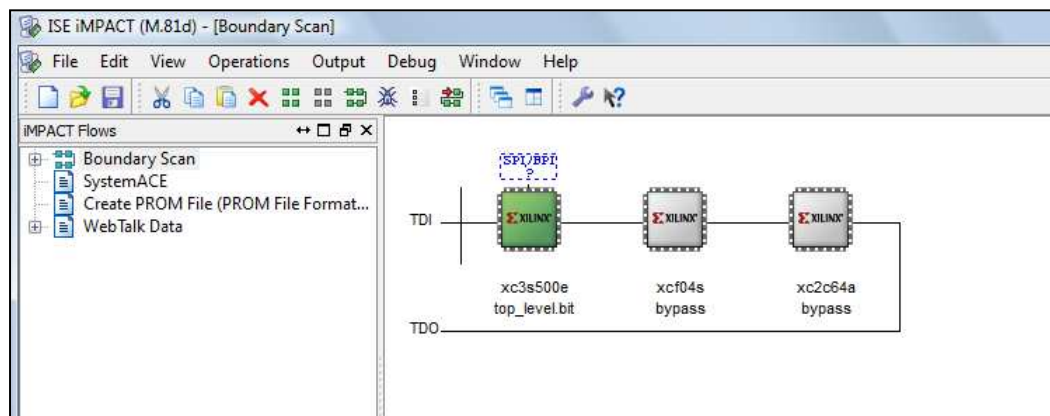


Figure 6.58: iMPACT window, showing the device chain.

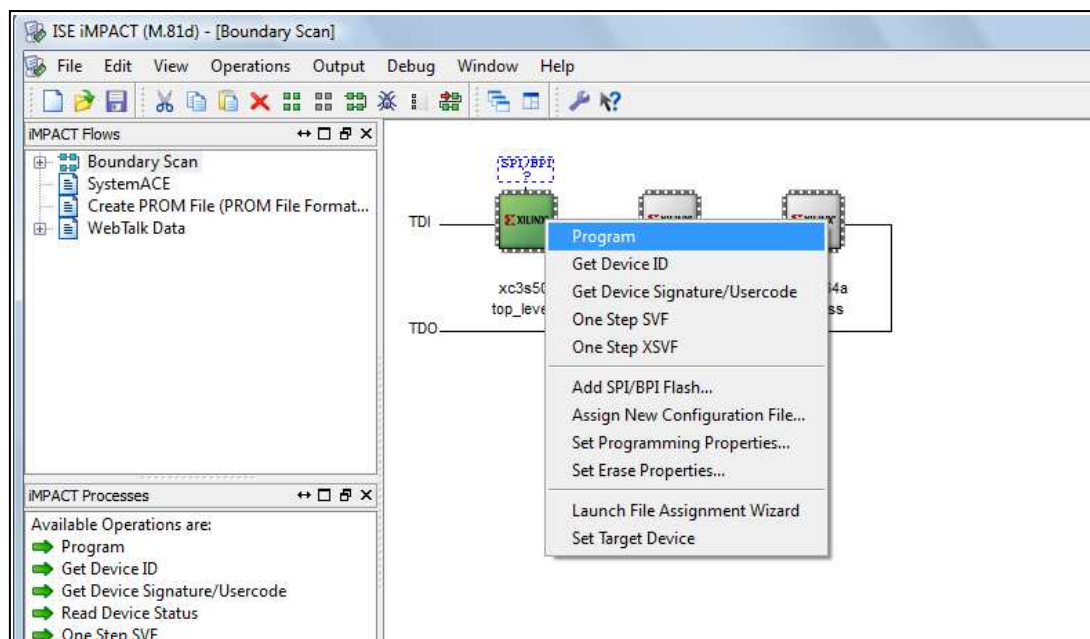


Figure 6.59: iMPACT window, options which appear when right clicking on the xc3s500e.

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 6.60).

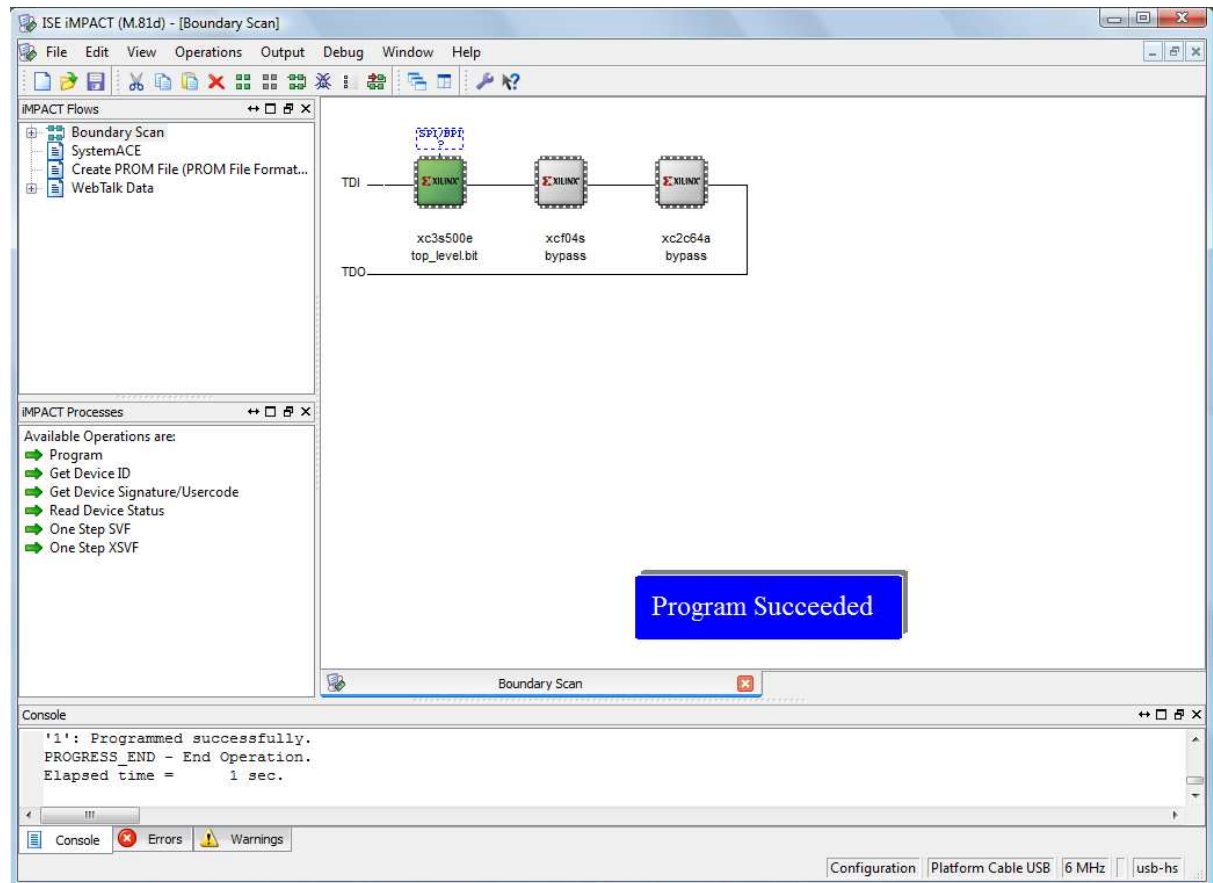


Figure 6.60: iMPACT window, after the program has been successfully downloaded to the Spartan-3E board.

6.14 Running the Program on the Spartan-3E Board

The Spartan-3E board after downloading the program is shown in Figure 6.61. The LCD screen displays the words “Hello World” on the first and second line. Also, corresponding LEDs will be lit when slider switches are in the ON position or push buttons are pressed.

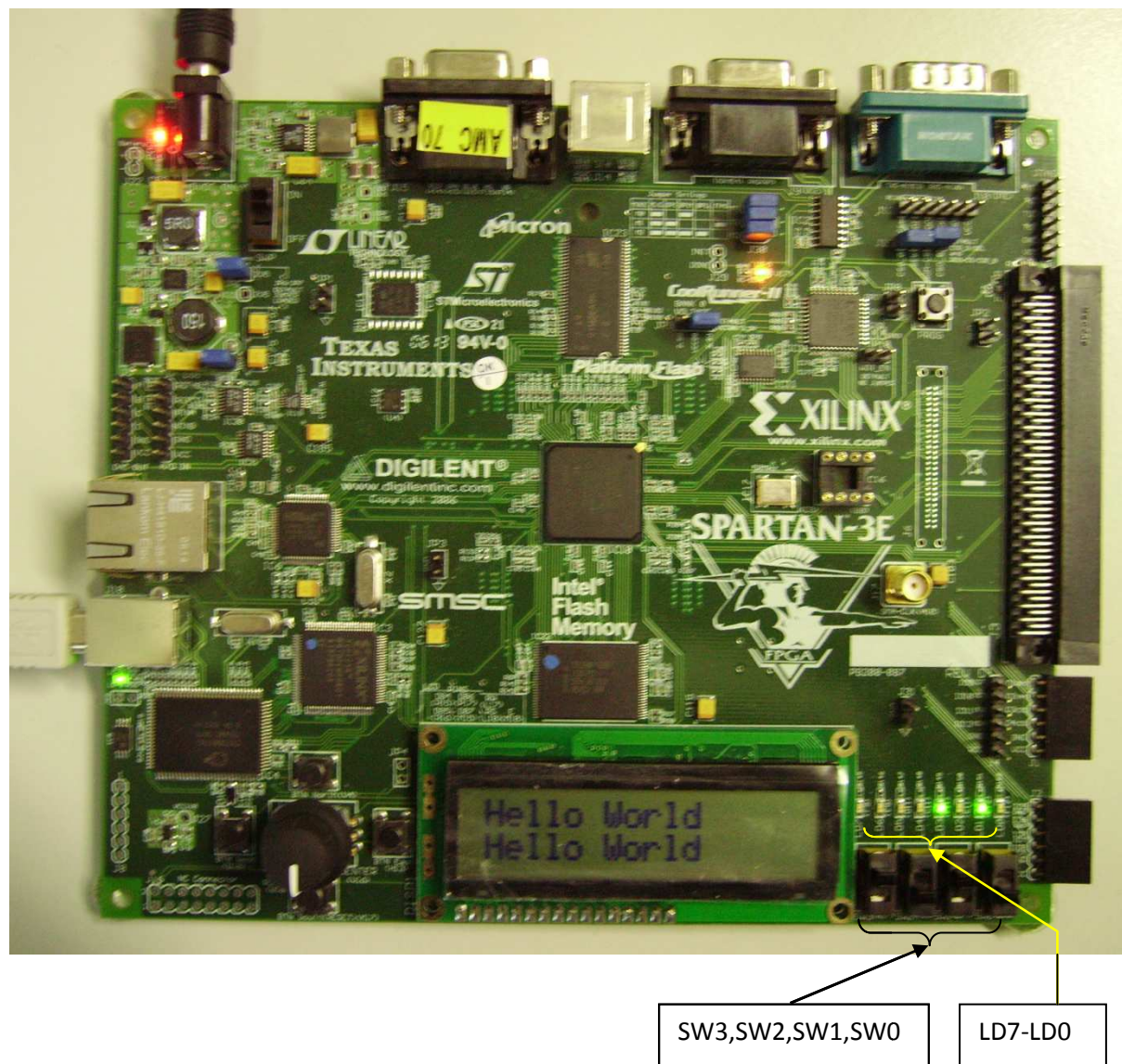


Figure 6.61: The Spartan-3E board with the program running for Part A.

7.0 Procedure Part B – Creating Custom Characters

7.1 Edit the .psm file

1. Open your file **hello.psm** in a text editor.
2. Replace the statements between **cold_start:** and **JUMP Main** in Figure 6.1 with the code of Figure 7.1.

```
cold_start: CALL LCD_reset           ;initialise LCD display
            CALL Setup_cust_chars    ;set up customised characters
            CALL Disp_mesg           ;display message
            ;
Main: INPUT s7, switch_port          ;read in switches
      OUTPUT s7, LED_port            ;output to LEDs
      JUMP Main
```

Figure 7.1: Main program loop for Part B.

2. Replace the **Disp_mesg** subroutine in Figure 6.1 with the code of Figure 7.2.

```
Disp_mesg: LOAD s5, 80               ;Line 1 position 1
            CALL LCD_write_inst8
            CALL Disp_Hello_World    ;12 chars
            CALL Disp_cust_chars     ; 4 chars

            LOAD s5, C0              ;Line 2 position 1
            CALL LCD_write_inst8
            CALL Disp_Hello_World    ;12 chars
            CALL Disp_cust_chars     ; 4 chars

            RETURN
```

Figure 7.2: Disp_mesg subroutine for Part B.

3. Add the **Disp_cust_chars** and **Setup_cust_chars** subroutines, shown in Figures 7.3 and 7.4. The **Disp_Hello_World** subroutine is unchanged.

```
Disp_cust_chars:  LOAD s5, 00                ;CG RAM address 00, cust char
                  CALL LCD_write_data
                  CALL LCD_write_data

                  LOAD s5, 01                ;CG RAM address 01, cust char
                  CALL LCD_write_data

                  LOAD s5, 20                ;space
                  CALL LCD_write_data
                  RETURN
                  ;
```

Figure 7.3: Disp_cust_chars subroutine for Part B.

```

Setup_cust_chars: LOAD s5, 40                ;set CG RAM address 00
                  CALL LCD_write_inst8

                  LOAD s5, 00                ;line 1
                  CALL LCD_write_data
                  LOAD s5, 0E                ;line 2
                  CALL LCD_write_data
                  LOAD s5, 1F                ;line 3
                  CALL LCD_write_data
                  LOAD s5, 15                ;line 4
                  CALL LCD_write_data
                  LOAD s5, 1F                ;line 5
                  CALL LCD_write_data
                  LOAD s5, 1F                ;line 6
                  CALL LCD_write_data
                  LOAD s5, 15                ;line 7
                  CALL LCD_write_data
                  LOAD s5, 00                ;line 8
                  CALL LCD_write_data

                  LOAD s5, 48                ;set CG RAM address 01
                  CALL LCD_write_inst8

                  LOAD s5, 0E                ;line 1
                  CALL LCD_write_data
                  LOAD s5, 1F                ;line 2
                  CALL LCD_write_data
                  LOAD s5, 0F                ;line 3
                  CALL LCD_write_data
                  LOAD s5, 07                ;line 4
                  CALL LCD_write_data
                  LOAD s5, 0F                ;line 5
                  CALL LCD_write_data
                  LOAD s5, 1F                ;line 6
                  CALL LCD_write_data
                  LOAD s5, 0E                ;line 7
                  CALL LCD_write_data
                  LOAD s5, 00                ;line 8
                  CALL LCD_write_data
                  RETURN

```

Figure 7.4: Setup_cust_chars subroutine for Part B.

For reference the modified **hello.psm** is listed in Appendix B.

The subroutine **Setup_cust_chars** initialises two customised characters, at addresses 0x00 and 0x01 in the CG RAM. Tables 7.1 and 7.2 show the two customised characters. The subroutine **Disp_cust_chars** displays two of the character at 0x00, followed by the character at 0x01, followed by a space.

Disp_mesg positions the cursor at line 1, position 1, and then writes the text “Hello World” followed by the custom characters to line 1 of the LCD. Next, it positions the cursor at line 2 position 1 and repeats the process to display the same message on line 2.

						Upper Nibble			Lower Nibble				
						Write Data to CG RAM of DD RAM							
A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
Character Address			Row Address			Don't Care			Character Bitmap				
0	0	0	0	0	0	-	-	-	0	0	0	0	0
0	0	0	0	0	1	-	-	-	0	1	1	1	0
0	0	0	1	1	0	-	-	-	1	1	1	1	1
0	0	0	0	1	1	-	-	-	1	0	1	0	1
0	0	0	1	0	0	-	-	-	1	1	1	1	1
0	0	0	1	0	1	-	-	-	1	1	1	1	1
0	0	0	1	1	0	-	-	-	1	0	1	0	1
0	0	0	1	1	1	-	-	-	0	0	0	0	0

Table 7.1: Example custom character stored in CG RAM address 0x00.

						Upper Nibble				Lower Nibble			
						Write Data to CG RAM of DD RAM							
A5	A4	A3	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
Character Address			Row Address			Don't Care			Character Bitmap				
0	0	1	0	0	0	-	-	-	0	1	1	1	0
0	0	1	0	0	1	-	-	-	1	1	1	1	1
0	0	1	1	1	0	-	-	-	0	1	1	1	1
0	0	1	0	1	1	-	-	-	0	0	1	1	1
0	0	1	1	0	0	-	-	-	0	1	1	1	1
0	0	1	1	0	1	-	-	-	1	1	1	1	1
0	0	1	1	1	0	-	-	-	0	1	1	1	0
0	0	1	1	1	1	-	-	-	0	0	0	0	0

Table 7.2: Example custom character stored in CG RAM address 0x01.

7.2 Running the Assembler

Run the KCPSM3 assembler on the modified **hello.psm** as done in Section 6.2. After the assembler has successfully been run a new output file **hello.vhd** should be produced.

7.3 Project Navigator

1. Return to Project Navigator. The new **hello.vhd** will be automatically loaded. If you click on the Project Navigator window before KCPSM3 has finished running, the **Locate Missing Source Files** window shown in Figure 7.5 will appear. Simply click **Cancel** after KCPSM3 has finished running, and the new **hello.vhd** will be automatically loaded.

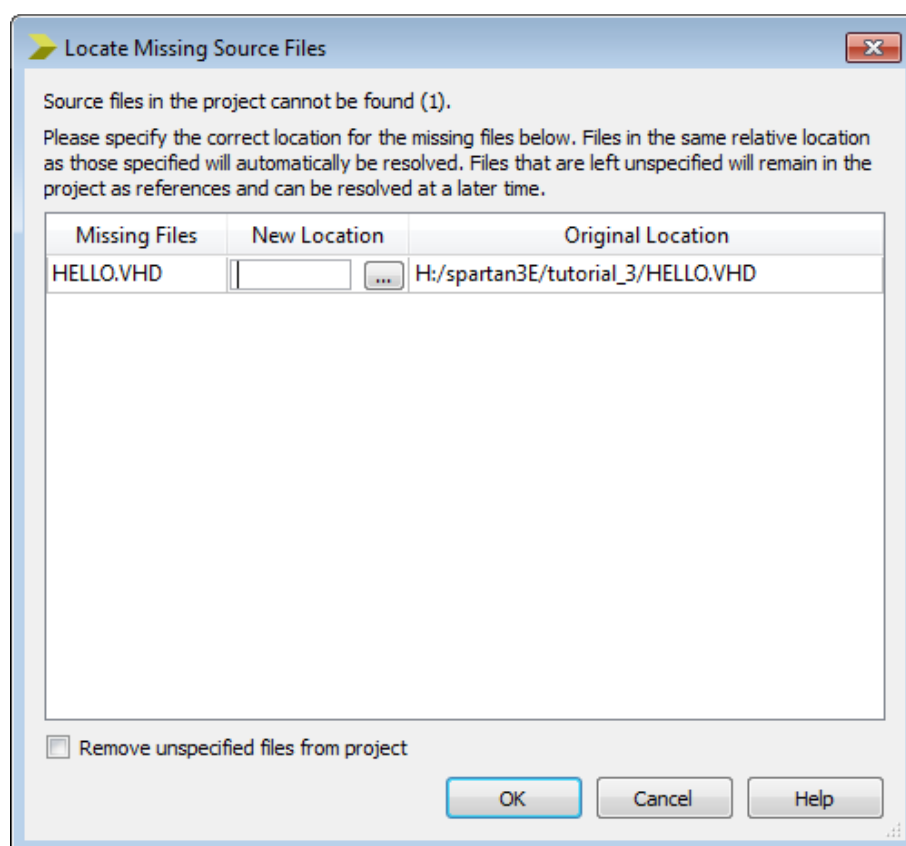


Figure 7.5: Locate Missing Source Files window.

3. Run Synthesise, Translate, Map, and Place and Route as done in Section 6.12. Note that no changes need to be made to `top_level.vhd`, or the pin assignments.
4. Download the design to the board, as done in Section 6.13.

7.4 Running the Program on the Spartan-3E Board

The Spartan-3E board after downloading the program is shown in Figure 7.5. The LCD screen displays the words “Hello World” followed by the customised characters, on the first and second line. Also, corresponding LEDs will be lit when slider switches are in the ON position or push buttons are pressed.

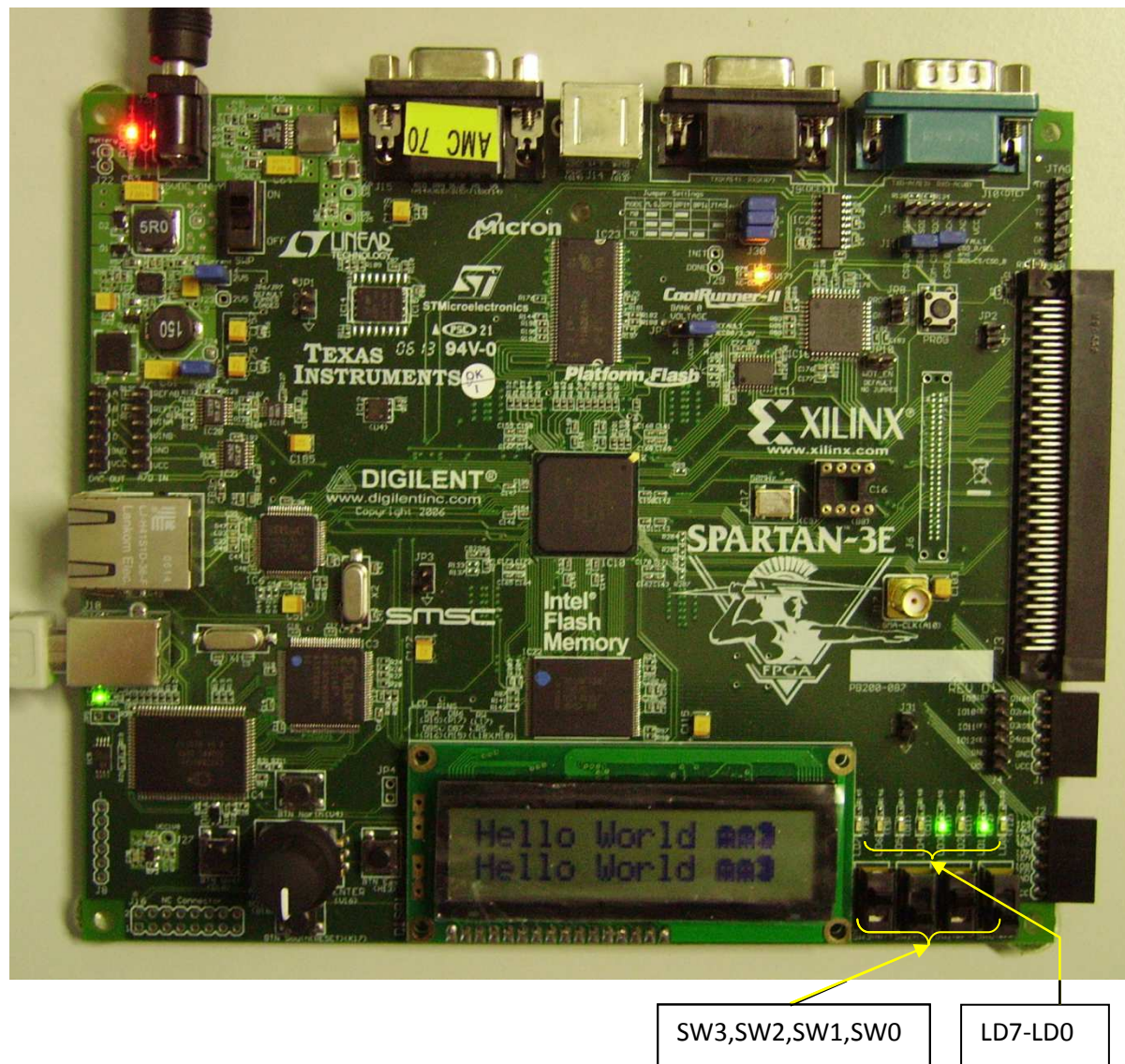


Figure 7.6: The Spartan-3E board with the program running for Part B.

8.0 Procedure Part C – Flashing and Shifting

8.1 Edit the .psm file

1. Open your file **hello.psm** in a text editor.
2. Place the constants **shift_delay_msb** and **shift_delay_lsb** shown in Figure 8.1, in the constants section header of **hello.psm** (Block A in Figure 4.5).

```
;The main operation of the program uses lms delays to set the  
;shift rate of the LCD display. A 16-bit value determines how  
;many milliseconds there are between shifts  
;  
;Tests indicate that the fastest shift rate that the LCD  
;display supports is 500ms. Faster than this and the display  
;becomes less clear to read.  
;  
CONSTANT shift_delay_msb, 01          ;delay is 500ms (01F4 hex)  
CONSTANT shift_delay_lsb, F4  
;
```

Figure 8.1: Delay constants for shifting.

3. Replace the statements between **cold_start:** and **JUMP Main** in Figure 7.1 with the code of Figure 8.2.

```

cold_start: CALL LCD_reset           ;initialise LCD display
            CALL Setup_cust_chars    ;set up customised characters
            CALL Disp_mesg           ;display message
            ;
            ENABLE INTERRUPT         ;enable interrupts
            LOAD s9, 01              ;initialise display flag to on
            ;
Main:       LOAD sF, shift_delay_msb  ;[sF,sE]=loop delay in ms
            LOAD sE, shift_delay_lsb

LCD_delay_loop: CALL delay_1ms        ;1ms delay
            SUB sE, 01               ;decrement delay counter
            SUBCY sF, 00
            JUMP NC, LCD_delay_loop

            INPUT s7, switch_port    ;read in switches
            OUTPUT s7, LED_port      ;output to LEDs

            TEST s7, switch3         ;check for switch3 (left)
            JUMP Z, Check_right

            LOAD s5, 18              ;shift display left
            CALL LCD_write_inst8

Check_right: TEST s7, switch2         ;check for switch2 (right)
            JUMP Z, End_loop

            LOAD s5, 1C              ;shift display right
            CALL LCD_write_inst8

End_loop:   JUMP Main

```

Figure 8.2: Main program loop for Part C.

3. Replace the **Disp_mesg** subroutine in Figure 7.2 with the code of Figure 8.3.

```
Disp_mesg: LOAD s5, 80                                ;Line 1 position 1
            CALL LCD_write_inst8

            ;total 40 chars written to line 1
            CALL Disp_Hello_World                      ;12 chars
            CALL Disp_cust_chars                      ; 4 chars
            CALL Disp_Hello_World                      ;12 chars
            CALL Disp_cust_chars                      ; 4 chars
            CALL Disp_cust_chars                      ; 4 chars

            CALL Disp_cust_chars                      ; 4 chars

            LOAD s5, C0                                ;Line 2 position 1
            CALL LCD_write_inst8

            ;total 40 chars written to line 2
            CALL Disp_Hello_World                      ;12 chars
            CALL Disp_cust_chars                      ; 4 chars
            CALL Disp_Hello_World                      ;12 chars
            CALL Disp_cust_chars                      ; 4 chars
            CALL Disp_cust_chars                      ; 4 chars

            CALL Disp_cust_chars                      ; 4 chars

            RETURN
```

Figure 8.3: Disp_mesg subroutine for Part C.

4. Place the interrupt service routine and ADDRESS directive, shown in Figure 8.4, at the end of the program.

```

;*****
;Interrupt Service Routine
;*****
;
ISR: TEST s7, switch1           ;check for switch1 (flash)
    JUMP Z, Test_disp_on
;
    XOR s9, 01                 ;toggle display flag
    TEST s9, 01                ;check if display should
                                ;be turned on or off

    JUMP NZ, Disp_on
;
    LOAD s5, 08                ;turn display off
    JUMP End_disp
;
Disp_on: LOAD s5, 0C            ;turn display on
    JUMP End_disp
;
Test_disp_on: TEST s9, 01       ;if flash off, make sure
                                ;the display is on

    JUMP NZ, End_ISR
;
    LOAD s9, 01                ;set display flag on
    LOAD s5, 0C                ;turn display on
;
End_disp: CALL LCD_write_inst8
;
End_ISR: RETURNI ENABLE
;
;-----
;
ADDRESS 3FF
JUMP ISR
;
;

```

Figure 8.4: Interrupt service routine and ADDRESS directive.

For reference the modified **hello.psm** is listed in Appendix C.

The **Disp_mesg** subroutine in Figure 8.3 has been modified to display a longer message, which takes up all 40 characters of both lines.

Shifting of the display is controlled by the main program loop in Figure 8.2, between the label **Main:** and the instruction **JUMP Main**. Registers **sF** and **sE** are loaded with 0x01F4 (500 in decimal). The code at label **LCD_delay_loop:** will cause the subroutine **delay_1ms** to be called 500 times, creating a delay of 500ms. The switches are then read in from the **switch_port**. If SW3 is on, a command is sent to the LCD to shift the display left. If SW2 is on, a command is sent to shift the

display right. If both switches are on, the code will cause both commands to be sent, but the effect is that the display will not shift. The code then jumps back to the **Main:** label and the loop is repeated.

Flashing is controlled by the interrupt service routine in Figure 8.4. In the initialisation code after **cold_start:** in Figure 8.2, interrupts are enabled using the **ENABLE INTERRUPT**. The register **s9** is used to keep track of whether the display is on or off, and is initialised to 0x01 indicating that the display is initially on.

The interrupt service routine first checks if SW1 is on. If so, the display needs to be flashed. The display flag, **s9** is toggled. Then, if **s9** is equal to 0x01, a command is sent to the LCD to turn it on, and if **s9** is equal to 0x00, a command is sent to the LCD to turn it off. Finally, the code also has to check if SW1 was switched off while the display was off. In this case, the display should be turned back on. Therefore, if SW1 is off, the code jumps to label **Test_disp_on:**. If the display is off, then a command is sent to the LCD to turn it on, and **s9** is set to 0x01.

The **RETURNI ENABLE** instruction returns from the interrupt service routine and re-enables interrupts.

ADDRESS is an assembler directive, indicating that the code that follows needs to be assigned to a specific address. In Figure 8.4, it indicates that the **JUMP ISR** instruction has to be placed at the interrupt vector, 0x3FF.

8.2 Running the Assembler

Run the KCPSM3 assembler on the modified **hello.psm** as described in Section 6.2. After the assembler has successfully been run a new output file **hello.vhd** should be produced.

8.3 Project Navigator

1. Return to Project Navigator. The new **hello.vhd** will be automatically loaded.
2. Add the library declarations of Figure 8.5 to **top_level.vhd**.

```
use IEEE.STD_LOGIC_ARITH.ALL;  
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

Figure 8.5: Library declarations to add.

2. Add the signal definitions of Figure 8.6, to the **architecture** of **top_level**.


```
-- Interrupt signals
signal count_event      : std_logic;
signal count_ticks      : std_logic_vector(25 downto 0) :=
"00000000000000000000000000000000";

-- 12.5e6 ticks = 0.25 seconds, 50MHz clock
--constant count_limit_min: std_logic_vector(25 downto 0) :=
"001101111010111100001000000";
--
-- 25e6 ticks = 0.5 seconds, 50MHz clock
constant count_limit     : std_logic_vector(25 downto 0) :=
"011011110101111000010000000";
--
-- 50e6 ticks = 1 second, 50MHz clock
--constant count_limit_max: std_logic_vector(25 downto 0) :=
"101111101011110000100000000";
--
--
```

Figure 8.6: Signals used for interrupts.

3. Add the **processes** of Figure 8.7, to the **architecture** of top_level.

```

-----
-- interrupt
-----
--
-- sets an interrupt flag based on a counter
--
counter: process(clk)
begin
    if clk'event and clk='1' then
        count_ticks <= count_ticks + 1;
        if count_ticks > count_limit then
            count_event <= '1';
            count_ticks <= (others => '0');
        else
            count_event <= '0';
        end if;
    end if;
end process counter;
--
-----
-- interrupt handler
-----
--
interrupt_control: process(clk)
begin
    if clk'event and clk='1' then
        if interrupt_ack = '1' then
            interrupt <= '0';
        elsif count_event = '1' then
            interrupt <= '1';
        else
            interrupt <= interrupt;
        end if;
    end if;
end process interrupt_control;

```

Figure 8.7: Processes associated with interrupt control.

5. Run Synthesise, Translate, Map, and Place and Route as done in Section 6.12. Note that no changes need to be made to the pin assignments.
6. Download the design to the board, as done in Section 6.13.

For reference the modified **top_level.vhd** is listed in Appendix C.

Code has been added to **top_level.vhd** to generate interrupts to cause the display to flash at a pre-defined time interval. The interrupt service routine code of Figure 8.4 will be called when the **interrupt** signal input to the KCPSM3 becomes '1'. The **interrupt** signal is managed by the **interrupt_control** process of Figure 8.7. If the **count_event** signal is '1', the **interrupt** signal is set to '1'. If an **interrupt_ack** signal comes from the KCPSM3, the **interrupt** signal is cancelled (set to '0').

Signal **count_event** is set by the **counter** process of Figure 8.7, when a counter reaches **count_limit**. In Figure 8.6, **count_limit** has been set to a value of "01101111010111100001000000", which is the number of clock ticks to result in a delay of 500ms, with a 50MHz clock. For experimentation purposes, values of **count_limit** for delays of 250ms and 1s have been included in the code but commented out.

8.4 Running the Program on the Spartan-3E Board

The Spartan-3E board after downloading the program is shown in Figure 8.8. The LCD screen displays the words “Hello World” followed by the customised characters, on the first and second line. If slider switches SW3 or SW2 are in the ON position, the display will shift to the left or right respectively, with a time interval of 0.5s. If slider switch SW1 is in the ON position, the display will flash, also with a time interval of 0.5s. Corresponding LEDs will also be lit when slider switches are in the ON position or push buttons are pressed.

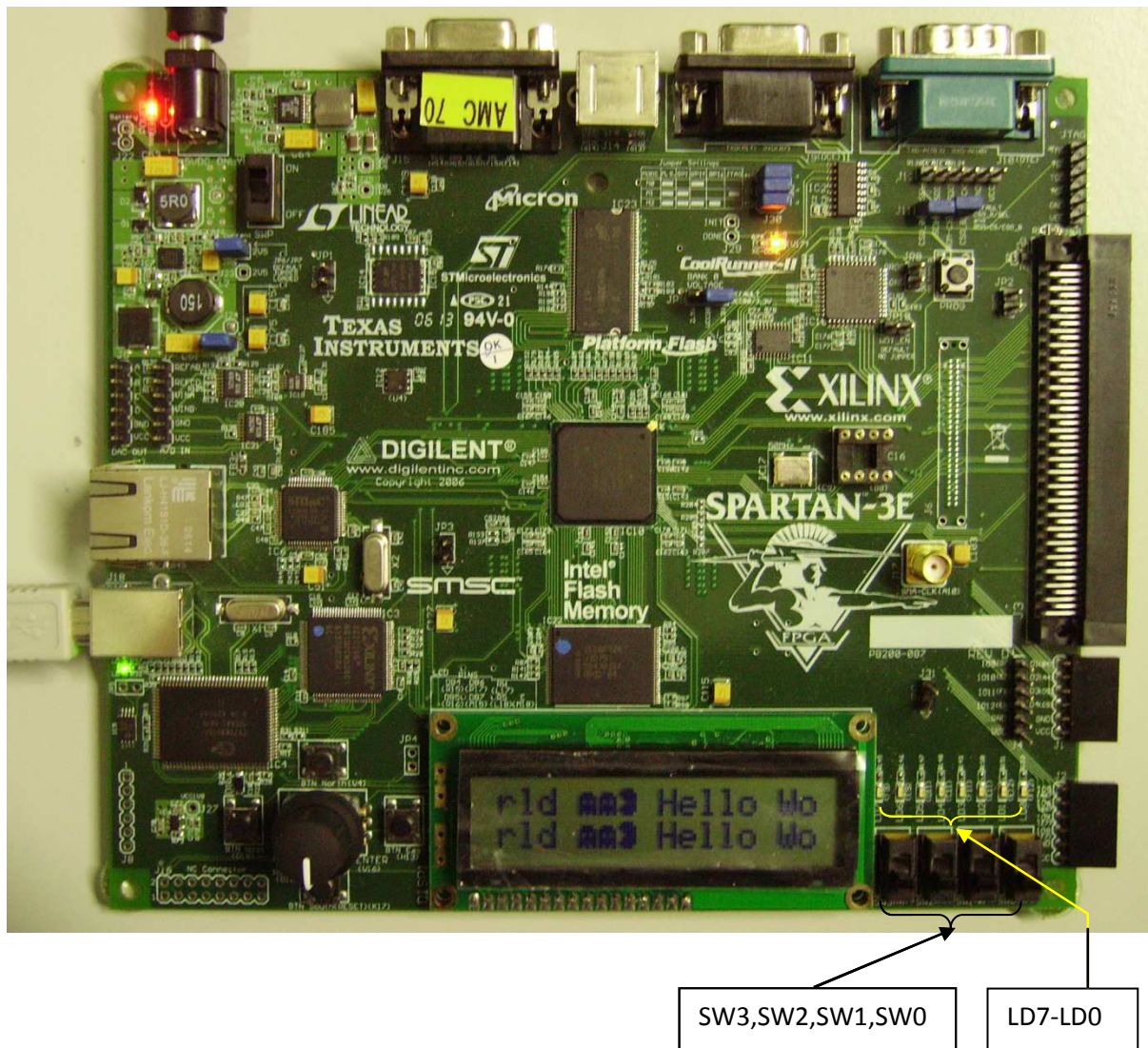


Figure 8.8: The Spartan-3E board with the program running for Part C.

9.0 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:
http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf, accessed 3 Dec 2012.
- [2] PicoBlaze 8-bit Embedded Microcontroller User Guide, Online:
http://www.xilinx.com/support/documentation/ip_documentation/ug129.pdf, accessed 3 Dec 2012.
- [3] Banks, J., The Spartan-3E Tutorial 1: Introduction to FPGA Programming, Queensland University of Technology, 2011.
- [4] Banks, J., The Spartan-3E Tutorial 2: Introduction to Using the PicoBlaze Microcontroller, Queensland University of Technology, 2012.
- [5] Roth, C. H., Digital Systems Design Using VHDL, PWS Publishing Company, 1998.
- [6] Roth, C. H. And Kinney, L. L., Fundamentals of Logic Design, 6th edition, CENGAGE Learning, 2010.
- [7] Chu, P., FPGA Prototyping by VHDL Examples: Xilinx Spartan-3 Edition, Wiley-Interscience, 2008.
- [8] Spartan-3E FPGA Starter Kit Board Design Examples, Online:
http://www.xilinx.com/products/boards/s3estarter/reference_designs.htm, accessed 3 Dec 2012.
- [9] PicoBlaze KCPSM3 8-bit Microcontroller for Spartan-3, Virtex-II and Virtex_IIPro, Ken Chapman, Xilinx, 2003, downloaded from <http://www.xilinx.com/> as part of KCPSM3.zip, accessed 3 Dec 2012.

Appendix A – Code for Part A

A.1 hello.psm

```

;*****
;Port definitions
;*****
;
CONSTANT LED_port, 80          ;8 simple LEDs
CONSTANT LED0, 01              ;    LED 0 - bit0
CONSTANT LED1, 02              ;    1 - bit1
CONSTANT LED2, 04              ;    2 - bit2
CONSTANT LED3, 08              ;    3 - bit3
CONSTANT LED4, 10              ;    4 - bit4
CONSTANT LED5, 20              ;    5 - bit5
CONSTANT LED6, 40              ;    6 - bit6
CONSTANT LED7, 80              ;    7 - bit7
;
CONSTANT switch_port, 00       ;Read switches and
                                ;press buttons
;
CONSTANT switch0, 01           ; Switches SW0 - bit0
CONSTANT switch1, 02           ;    SW1 - bit1
CONSTANT switch2, 04           ;    SW2 - bit2
CONSTANT switch3, 08           ;    SW3 - bit3
CONSTANT BTN_east, 10          ; Buttons East - bit4
CONSTANT BTN_south, 20         ;    South - bit5
CONSTANT BTN_north, 40         ;    North - bit6
CONSTANT BTN_west, 80          ;    West - bit7
;
;
;LCD interface ports
;
;The master enable signal is not used by the LCD display
;itself but may be required to confirm that LCD
;communication is active.
;This is required on the Spartan-3E Starter Kit if the
;StrataFLASH is used because it shares the same data pins
;and conflicts must be avoided.
;
CONSTANT LCD_output_port, 40    ;LCD character module
                                ;output data and control
CONSTANT LCD_E, 01             ;    active High Enable E - bit0
CONSTANT LCD_RW, 02            ;    Read=1 Write=0    RW - bit1
CONSTANT LCD_RS, 04            ;    Instruction=0 Data=1 RS - bit2
CONSTANT LCD_drive, 08         ;Master enable(active High)- bit3
CONSTANT LCD_DB4, 10           ; 4-bit                Data DB4 - bit4
CONSTANT LCD_DB5, 20           ; interface            Data DB5 - bit5
CONSTANT LCD_DB6, 40           ;                      Data DB6 - bit6
CONSTANT LCD_DB7, 80           ;                      Data DB7 - bit7
;

```

```

;
CONSTANT LCD_input_port, 02      ;LCD character module
                                ;input data
CONSTANT LCD_read_spare0, 01     ; Spare bits          - bit0
CONSTANT LCD_read_spare1, 02     ; are zero             - bit1
CONSTANT LCD_read_spare2, 04     ;                      - bit2
CONSTANT LCD_read_spare3, 08     ;                      - bit3
CONSTANT LCD_read_DB4, 10        ; 4-bit Data DB4 - bit4
CONSTANT LCD_read_DB5, 20        ;interface Data DB5 - bit5
CONSTANT LCD_read_DB6, 40        ; Data DB6 - bit6
CONSTANT LCD_read_DB7, 80        ; Data DB7 - bit7
;
;
;Constant to define a software delay of 1us. This must be
;adjusted to reflect the clock applied to KCPSM3. Every
;instruction executes in 2 clock cycles making the
;calculation highly predictable. The '6' in the following
;equation even allows for 'CALL delay_1us' instruction in
;the initiating code.
;
;delay_1us_constant = (clock_rate - 6)/4
;Where 'clock_rate' is in MHz
;
;Example: For a 50MHz clock the constant value is
;      (10-6)/4 = 11 (0B Hex).
;For clock rates below 10MHz the value of 1 must be used
;and the operation will become lower than intended.
;
CONSTANT delay_1us_constant, 0B
;
;*****
;Initialise the system
;*****
;
cold_start: CALL LCD_reset          ;initialise LCD display
            CALL Disp_mesg         ;display message
;
Main: INPUT s7, switch_port        ;read in switches
      OUTPUT s7, LED_port         ;output to LEDs
      JUMP Main
;
;
Disp_mesg: LOAD s5, 80              ;Line 1 position 1
            CALL LCD_write_inst8
            CALL Disp_Hello_World
;
            LOAD s5, C0             ;Line 2 position 1
            CALL LCD_write_inst8
            CALL Disp_Hello_World
;
RETURN
;

```

```

Disp_Hello_World: LOAD s5, 48                                ;H
                  CALL LCD_write_data
                  LOAD s5, 65                                ;e
                  CALL LCD_write_data
                  LOAD s5, 6C                                ;l
                  CALL LCD_write_data
                  ;LOAD s5, 6C                                ;l
                  CALL LCD_write_data
                  LOAD s5, 6F                                ;o
                  CALL LCD_write_data
                  LOAD s5, 20                                ;space
                  CALL LCD_write_data
                  LOAD s5, 57                                ;W
                  CALL LCD_write_data
                  LOAD s5, 6F                                ;o
                  CALL LCD_write_data
                  LOAD s5, 72                                ;r
                  CALL LCD_write_data
                  LOAD s5, 6C                                ;l
                  CALL LCD_write_data
                  LOAD s5, 64                                ;d
                  CALL LCD_write_data
                  LOAD s5, 20                                ;space
                  CALL LCD_write_data
                  RETURN
;
;
;*****
;Software delay routines
;*****
;
;Delay of 1us.
;
;Constant value defines reflects the clock applied to
;KCP3M3. Every instruction executes in 2 clock cycles
;making the calculation highly predictable. The '6' in
;the following equation even allows for 'CALL delay_1us'
;instruction in the initiating code.
;
; delay_1us_constant = (clock_rate - 6)/4
; Where 'clock_rate' is in MHz
;
;Registers used s0
;
delay_1us: LOAD s0, delay_1us_constant
wait_1us:  SUB s0, 01
          JUMP NZ, wait_1us
          RETURN
;
;Delay of 40us.
;
;Registers used s0, s1

```

```

;
delay_40us: LOAD s1, 28 ;40 x 1us = 40us
wait_40us: CALL delay_1us
          SUB s1, 01
          JUMP NZ, wait_40us
          RETURN
;
;
;Delay of 1ms.
;
;Registers used s0, s1, s2
;
delay_1ms: LOAD s2, 19 ;25 x 40us = 1ms
wait_1ms: CALL delay_40us
          SUB s2, 01
          JUMP NZ, wait_1ms
          RETURN
;
;Delay of 20ms.
;
;Delay of 20ms used during initialisation.
;
;Registers used s0, s1, s2, s3
;
delay_20ms: LOAD s3, 14 ;20 x 1ms = 20ms
wait_20ms: CALL delay_1ms
          SUB s3, 01
          JUMP NZ, wait_20ms
          RETURN
;
;Delay of approximately 1 second.
;
;Registers used s0, s1, s2, s3, s4
;
delay_1s: LOAD s4, 32 ;50 x 20ms = 1000ms
wait_1s: CALL delay_20ms
          SUB s4, 01
          JUMP NZ, wait_1s
          RETURN
;
;
;
;*****
;LCD Character Module Routines
;*****
;
;LCD module is a 16 character by 2 line display but all
;displays are very similar
;The 4-wire data interface will be used (DB4 to DB7).
;
;The LCD modules are relatively slow and software delay
;loops are used to slow down KCPSM3 adequately for the

```

```

;LCD to communicate. The delay routines are provided in
;a different section (see above in this case).
;
;
;Pulse LCD enable signal 'E' high for greater than 230ns
;(1us is used).
;
;Register s4 should define the current state of the LCD
;output port.
;
;Registers used s0, s4
;
LCD_pulse_E: XOR s4, LCD_E                      ;E=1
              OUTPUT s4, LCD_output_port
              CALL delay_1us
              XOR s4, LCD_E                      ;E=0
              OUTPUT s4, LCD_output_port
              RETURN
;
;Write 4-bit instruction to LCD display.
;
;The 4-bit instruction should be provided in the upper
;4-bits of register s4. Note that this routine does not
;release the master enable but as it is only used during
;initialisation and as part of the 8-bit instruction
;write it should be acceptable.
;
;Registers used s4
;
LCD_write_inst4: AND s4, F8                      ;Enable=1 RS=0
                                                         ;Instruction, RW=0
                                                         ;Write, E=0
              OUTPUT s4, LCD_output_port        ;set up RS and RW >40ns
                                                         ;before enable pulse

              CALL LCD_pulse_E
              RETURN
;
;
;Write 8-bit instruction to LCD display.
;
;The 8-bit instruction should be provided in register s5.
;Instructions are written using the following sequence
; Upper nibble
; wait >1us
; Lower nibble
; wait >40us
;
;Registers used s0, s1, s4, s5
;
LCD_write_inst8: LOAD s4, s5
                AND s4, F0                      ;Enable=0 RS=0
                                                         ;Instruction, RW=0

```

```

;
OR s4, LCD_drive
CALL LCD_write_inst4
CALL delay_1us
LOAD s4, s5

SL1 s4
SL0 s4
SL0 s4
SL0 s4
CALL LCD_write_inst4
CALL delay_40us
LOAD s4, F0

OUTPUT s4, LCD_output_port
RETURN

;
;Write 8-bit data to LCD display.
;
;The 8-bit data should be provided in register s5.
;Data bytes are written using the following sequence
; Upper nibble
; wait >1us
; Lower nibble
; wait >40us
;
;Registers used s0, s1, s4, s5
;
LCD_write_data: LOAD s4, s5
AND s4, F0

;
OR s4, 0C

;
OUTPUT s4, LCD_output_port

;
CALL LCD_pulse_E
CALL delay_1us
LOAD s4, s5

SL1 s4
SL1 s4
SL0 s4
SL0 s4
OUTPUT s4, LCD_output_port

;
;Write, E=0
;Enable=1
;write upper nibble
;wait >1us
;select lower nibble
;with
;Enable=1
;RS=0 Instruction
;RW=0 Write
;E=0
;write lower nibble
;wait >40us
;Enable=0 RS=0
;Instruction, RW=0
;Write, E=0
;Release master enable

;Enable=0 RS=0
;Instruction, RW=0
;Write, E=0

;Enable=1 RS=1 Data,
;RW=0 Write, E=0

;set up RS and RW >40ns
;before enable pulse

;write upper nibble
;wait >1us
;select lower nibble
;with
;Enable=1
;RS=1 Data
;RW=0 Write
;E=0
;set up RS and RW >40ns
;before enable pulse

```

```

CALL LCD_pulse_E           ;write lower nibble
CALL delay_40us            ;wait >40us
LOAD s4, F0                ;Enable=0 RS=0
                            ;Instruction, RW=0
                            ;Write, E=0

;
OUTPUT s4, LCD_output_port ;Release master enable
RETURN
;
;Read 8-bit data from LCD display.
;
;The 8-bit data will be read from the current LCD memory
;address and will be returned in register s5.
;It is advisable to set the LCD address (cursor position)
;before using the data read for the first time otherwise
;the display may generate invalid data on the first read.
;
;Data bytes are read using the following sequence
; Upper nibble
; wait >1us
; Lower nibble
; wait >40us
;
;Registers used s0, s1, s4, s5
;
LCD_read_data8: LOAD s4, 0E           ;Enable=1 RS=1 Data,
                                      ;RW=1 Read, E=0

;
OUTPUT s4, LCD_output_port           ;set up RS and RW >40ns
                                      ;before enable pulse

;
XOR s4, LCD_E                         ;E=1
OUTPUT s4, LCD_output_port
CALL delay_1us                        ;wait >260ns to
                                      ;access data

;
INPUT s5, LCD_input_port              ;read upper nibble
XOR s4, LCD_E                         ;E=0
OUTPUT s4, LCD_output_port
CALL delay_1us                        ;wait >1us
XOR s4, LCD_E                         ;E=1
OUTPUT s4, LCD_output_port
CALL delay_1us                        ;wait >260ns to
                                      ;access data

;
INPUT s0, LCD_input_port              ;read lower nibble
XOR s4, LCD_E                         ;E=0
OUTPUT s4, LCD_output_port
AND s5, F0                            ;merge upper and
                                      ;lower nibbles

SR0 s0
SR0 s0

```

```

        SR0 s0
        SR0 s0
        OR s5, s0
        LOAD s4, 04                                ;Enable=0 RS=1 Data,
                                                    ;RW=0 Write, E=0

        ;
        OUTPUT s4, LCD_output_port                ;Stop reading 5V
                                                    ;device and release
                                                    ;master enable

        ;
        CALL delay_40us                            ;wait >40us
        RETURN

        ;
        ;Reset and initialise display to communicate using 4-bit
        ;data mode
        ;Includes routine to clear the display.
        ;
        ;Requires the 4-bit instructions 3,3,3,2 to be sent with
        ;suitable delays following by the 8-bit instructions to
        ;set up the display.
        ;
        ; 28 = '001' Function set, '0' 4-bit mode, '1' 2-line,
        ;      '0' 5x7 dot matrix, 'xx'
        ; 06 = '000001' Entry mode, '1' increment, '0'
        ;      no display shift
        ; 0C = '00001' Display control, '1' display on,
        ;      '0' cursor off, '0' cursor blink off
        ; 01 = '00000001' Display clear
        ;
        ;Registers used s0, s1, s2, s3, s4
        ;
LCD_reset: CALL delay_20ms                        ;wait more than
                                                    ;15ms for display
                                                    ;to be ready

        ;
        LOAD s4, 30
        CALL LCD_write_inst4                       ;send '3'
        CALL delay_20ms                            ;wait >4.1ms
        CALL LCD_write_inst4                       ;send '3'
        CALL delay_1ms                             ;wait >100us
        CALL LCD_write_inst4                       ;send '3'
        CALL delay_40us                            ;wait >40us
        LOAD s4, 20
        CALL LCD_write_inst4                       ;send '2'
        CALL delay_40us                            ;wait >40us
        LOAD s5, 28                                ;Function set
        CALL LCD_write_inst8
        LOAD s5, 06                                ;Entry mode
        CALL LCD_write_inst8
        LOAD s5, 0C                                ;Display control
        CALL LCD_write_inst8
LCD_clear: LOAD s5, 01                            ;Display clear

```

```
CALL LCD_write_inst8
CALL delay_1ms           ;wait >1.64ms for
                          ;display to clear

CALL delay_1ms
RETURN
;
```

A.2 top_level.vhd

```

-----
-- Company:
-- Engineer:
--
-- Create Date:      13:20:51 02/04/2013
-- 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 (
        led : out std_logic_vector (7 downto 0);
        strataflash_oe : out std_logic;
        strataflash_ce : out std_logic;
        strataflash_we : out std_logic;
        switch : in std_logic_vector (3 downto 0);
        btn_north : in std_logic;
        btn_east : in std_logic;
        btn_south : in std_logic;
        btn_west : in std_logic;
        lcd_d : inout std_logic_vector (7 downto 4);
        lcd_rs : out std_logic;
        lcd_rw : out std_logic;
        lcd_e : out std_logic;
        clk : in std_logic);
end top_level;

```

```

--
-----
--
-- Start of test architecture
--
architecture Behavioral of top_level is
--
-----
-- declaration of KCPSM3
--
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 ROM
--
component hello
    Port (
        address : in std_logic_vector (9 downto 0);
        instruction : out std_logic_vector (17 downto 0);
        --proc_reset : out std_logic;           --JTAG Loader version
        clk : in std_logic);
    end component;
--
-----
-- Signals used to connect KCPSM3 to program ROM 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         : std_logic := '0';
signal interrupt_ack     : std_logic;
signal kcpsm3_reset     : std_logic;
--
--
-- Signals for LCD operation
--

```

```

-- Tri-state output requires internal signals
-- 'lcd_drive' is used to differentiate between LCD and StrataFLASH
-- communications which share the same data bits.
--
signal    lcd_rw_control : std_logic;
signal    lcd_output_data : std_logic_vector (7 downto 4);
signal    lcd_drive : std_logic;
--
-----
begin
    --
    -----
    -- LCD interface
    -----
    --
    -- The 4-bit data port is bidirectional.
    -- lcd_rw is '1' for read and '0' for write
    -- lcd_drive is like a master enable signal which prevents either the
    -- FPGA outputs or the LCD display driving the data lines.
    --
    -- Control of read and write signal
    lcd_rw <= lcd_rw_control and lcd_drive;

    -- use read/write control to enable output buffers.
    lcd_d <= lcd_output_data when (lcd_rw_control='0' and lcd_drive='1')
        else "ZZZZ";
    --
    -----
    -- Disable StrataFLASH
    -----
    --
    strataflash_oe <= '1';
    strataflash_ce <= '1';
    strataflash_we <= '1';
    --
    -----
    -- KCPSM3 and the program memory
    -----
    --
    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 => kcpsm3_reset,
            clk => clk);

```

```

program_rom: hello
  port map(
    address => address,
    instruction => instruction,
    --proc_reset => kcpsm3_reset,    --JTAG Loader version
    clk => clk);

--
-----
-- KCPSM3 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
      -- read simple toggle switches and buttons at address 00 hex
      when "00" =>    in_port <= btn_west & btn_north & btn_south
                        & btn_east & switch;

      -- read LCD data at address 02 hex
      when "10" =>    in_port <= lcd_d & "0000";

      -- 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 output ports
-----
--
-- adding the output registers to the processor
--
output_ports: process(clk)
begin
  if clk'event and clk='1' then
    if write_strobe='1' then

      -- Write to LEDs at address 80 hex.
      if port_id(7)='1' then
        led <= out_port;
      end if;

      -- LCD data output and controls at address 40 hex.
      if port_id(6)='1' then
        lcd_output_data <= out_port(7 downto 4);
        lcd_drive <= out_port(3);

```

```
        lcd_rs <= out_port(2);
        lcd_rw_control <= out_port(1);
        lcd_e <= out_port(0);
    end if;
end if;
end if;
end process output_ports;

end Behavioral;
```

Appendix B – Code for Part B

```

;*****
;Port definitions
;*****
;
CONSTANT LED_port, 80           ;8 simple LEDs
CONSTANT LED0, 01               ; LED 0 - bit0
CONSTANT LED1, 02               ; LED 1 - bit1
CONSTANT LED2, 04               ; LED 2 - bit2
CONSTANT LED3, 08               ; LED 3 - bit3
CONSTANT LED4, 10               ; LED 4 - bit4
CONSTANT LED5, 20               ; LED 5 - bit5
CONSTANT LED6, 40               ; LED 6 - bit6
CONSTANT LED7, 80               ; LED 7 - bit7
;
CONSTANT switch_port, 00        ;Read switches and
                                ;press buttons
;
CONSTANT switch0, 01            ; Switches SW0 - bit0
CONSTANT switch1, 02            ; SW1 - bit1
CONSTANT switch2, 04            ; SW2 - bit2
CONSTANT switch3, 08            ; SW3 - bit3
CONSTANT BTN_east, 10           ; Buttons East - bit4
CONSTANT BTN_south, 20          ; South - bit5
CONSTANT BTN_north, 40          ; North - bit6
CONSTANT BTN_west, 80           ; West - bit7
;
;
;LCD interface ports
;
;The master enable signal is not used by the LCD display
;itself but may be required to confirm that LCD
;communication is active.
;This is required on the Spartan-3E Starter Kit if the
;StrataFLASH is used because it shares the same data pins
;and conflicts must be avoided.
;
CONSTANT LCD_output_port, 40     ;LCD character module
                                ;output data and control
CONSTANT LCD_E, 01              ; active High Enable E - bit0
CONSTANT LCD_RW, 02              ; Read=1 Write=0 RW - bit1
CONSTANT LCD_RS, 04              ; Instruction=0 Data=1 RS - bit2
CONSTANT LCD_drive, 08           ;Master enable(active High)- bit3
CONSTANT LCD_DB4, 10             ; 4-bit Data DB4 - bit4
CONSTANT LCD_DB5, 20             ; interface Data DB5 - bit5
CONSTANT LCD_DB6, 40             ; Data DB6 - bit6
CONSTANT LCD_DB7, 80             ; Data DB7 - bit7
;
;
CONSTANT LCD_input_port, 02      ;LCD character module

```



```

                                ;input data
CONSTANT LCD_read_spare0, 01    ; Spare bits           - bit0
CONSTANT LCD_read_spare1, 02    ; are zero            - bit1
CONSTANT LCD_read_spare2, 04    ;                      - bit2
CONSTANT LCD_read_spare3, 08    ;                      - bit3
CONSTANT LCD_read_DB4, 10       ; 4-bit Data DB4 - bit4
CONSTANT LCD_read_DB5, 20       ;interface Data DB5 - bit5
CONSTANT LCD_read_DB6, 40       ; Data DB6 - bit6
CONSTANT LCD_read_DB7, 80       ; Data DB7 - bit7
;
;
;Constant to define a software delay of 1us. This must be
;adjusted to reflect the clock applied to KCPSM3. Every
;instruction executes in 2 clock cycles making the
;calculation highly predictable. The '6' in the following
;equation even allows for 'CALL delay_1us' instruction in
;the initiating code.
;
;delay_1us_constant = (clock_rate - 6)/4
;Where 'clock_rate' is in MHz
;
;Example: For a 50MHz clock the constant value is
;      (10-6)/4 = 11 (0B Hex).
;For clock rates below 10MHz the value of 1 must be used
;and the operation will become lower than intended.
;
CONSTANT delay_1us_constant, 0B
;
;*****
;Initialise the system
;*****
;
cold_start: CALL LCD_reset           ;initialise LCD display
            CALL Setup_cust_chars    ;set up customised characters
            CALL Disp_mesg           ;display message
            ;
Main: INPUT s7, switch_port          ;read in switches
      OUTPUT s7, LED_port            ;output to LEDs
      JUMP Main
      ;
      ;
Disp_mesg: LOAD s5, 80               ;Line 1 position 1
            CALL LCD_write_inst8
            CALL Disp_Hello_World
            CALL Disp_cust_chars

            LOAD s5, C0              ;Line 2 position 1
            CALL LCD_write_inst8
            CALL Disp_Hello_World
            CALL Disp_cust_chars

RETURN

```

```

;
Disp_Hello_World: LOAD s5, 48 ;H
                  CALL LCD_write_data
                  LOAD s5, 65 ;e
                  CALL LCD_write_data
                  LOAD s5, 6C ;l
                  CALL LCD_write_data
                  ;LOAD s5, 6C ;l
                  CALL LCD_write_data
                  LOAD s5, 6F ;o
                  CALL LCD_write_data
                  LOAD s5, 20 ;space
                  CALL LCD_write_data
                  LOAD s5, 57 ;W
                  CALL LCD_write_data
                  LOAD s5, 6F ;o
                  CALL LCD_write_data
                  LOAD s5, 72 ;r
                  CALL LCD_write_data
                  LOAD s5, 6C ;l
                  CALL LCD_write_data
                  LOAD s5, 64 ;d
                  CALL LCD_write_data
                  LOAD s5, 20 ;space
                  CALL LCD_write_data
                  RETURN

;
Disp_cust_chars: LOAD s5, 00 ;CG RAM address 00, cust char
                 CALL LCD_write_data
                 CALL LCD_write_data

                 LOAD s5, 01 ;CG RAM address 01, cust char
                 CALL LCD_write_data

                 LOAD s5, 20 ;space
                 CALL LCD_write_data
                 RETURN

;
Setup_cust_chars: LOAD s5, 40 ;set CG RAM address 00
                  CALL LCD_write_inst8

                  LOAD s5, 00 ;line 1
                  CALL LCD_write_data
                  LOAD s5, 0E ;line 2
                  CALL LCD_write_data
                  LOAD s5, 1F ;line 3
                  CALL LCD_write_data
                  LOAD s5, 15 ;line 4
                  CALL LCD_write_data
                  LOAD s5, 1F ;line 5
                  CALL LCD_write_data
                  LOAD s5, 1F ;line 6

```

```

CALL LCD_write_data
LOAD s5, 15                                ;line 7
CALL LCD_write_data
LOAD s5, 00                                ;line 8
CALL LCD_write_data

LOAD s5, 48                                ;set CG RAM address 01
CALL LCD_write_inst8

LOAD s5, 0E                                ;line 1
CALL LCD_write_data
LOAD s5, 1F                                ;line 2
CALL LCD_write_data
LOAD s5, 0F                                ;line 3
CALL LCD_write_data
LOAD s5, 07                                ;line 4
CALL LCD_write_data
LOAD s5, 0F                                ;line 5
CALL LCD_write_data
LOAD s5, 1F                                ;line 6
CALL LCD_write_data
LOAD s5, 0E                                ;line 7
CALL LCD_write_data
LOAD s5, 00                                ;line 8
CALL LCD_write_data
RETURN
;
;
;*****
;Software delay routines
;*****
;
;Delay of 1us.
;
;Constant value defines reflects the clock applied to
;KCP3M3. Every instruction executes in 2 clock cycles
;making the calculation highly predictable. The '6' in
;the following equation even allows for 'CALL delay_1us'
;instruction in the initiating code.
;
; delay_1us_constant = (clock_rate - 6)/4
; Where 'clock_rate' is in MHz
;
;Registers used s0
;
delay_1us: LOAD s0, delay_1us_constant
wait_1us: SUB s0, 01
JUMP NZ, wait_1us
RETURN
;
;Delay of 40us.
;

```

```

;Registers used s0, s1
;
delay_40us: LOAD s1, 28 ;40 x 1us = 40us
wait_40us: CALL delay_1us
SUB s1, 01
JUMP NZ, wait_40us
RETURN
;
;
;Delay of 1ms.
;
;Registers used s0, s1, s2
;
delay_1ms: LOAD s2, 19 ;25 x 40us = 1ms
wait_1ms: CALL delay_40us
SUB s2, 01
JUMP NZ, wait_1ms
RETURN
;
;Delay of 20ms.
;
;Delay of 20ms used during initialisation.
;
;Registers used s0, s1, s2, s3
;
delay_20ms: LOAD s3, 14 ;20 x 1ms = 20ms
wait_20ms: CALL delay_1ms
SUB s3, 01
JUMP NZ, wait_20ms
RETURN
;
;Delay of approximately 1 second.
;
;Registers used s0, s1, s2, s3, s4
;
delay_1s: LOAD s4, 32 ;50 x 20ms = 1000ms
wait_1s: CALL delay_20ms
SUB s4, 01
JUMP NZ, wait_1s
RETURN
;
;
;
;*****
;LCD Character Module Routines
;*****
;
;LCD module is a 16 character by 2 line display but all
;displays are very similar
;The 4-wire data interface will be used (DB4 to DB7).
;
;The LCD modules are relatively slow and software delay

```

```

;loops are used to slow down KCPSM3 adequately for the
;LCD to communicate. The delay routines are provided in
;a different section (see above in this case).
;
;
;Pulse LCD enable signal 'E' high for greater than 230ns
;(1us is used).
;
;Register s4 should define the current state of the LCD
;output port.
;
;Registers used s0, s4
;
LCD_pulse_E: XOR s4, LCD_E                      ;E=1
             OUTPUT s4, LCD_output_port
             CALL delay_1us
             XOR s4, LCD_E                      ;E=0
             OUTPUT s4, LCD_output_port
             RETURN
;
;Write 4-bit instruction to LCD display.
;
;The 4-bit instruction should be provided in the upper
;4-bits of register s4. Note that this routine does not
;release the master enable but as it is only used during
;initialisation and as part of the 8-bit instruction
;write it should be acceptable.
;
;Registers used s4
;
LCD_write_inst4: AND s4, F8                      ;Enable=1 RS=0
                                                         ;Instruction, RW=0
                                                         ;Write, E=0
             OUTPUT s4, LCD_output_port          ;set up RS and RW >40ns
                                                         ;before enable pulse

             CALL LCD_pulse_E
             RETURN
;
;
;Write 8-bit instruction to LCD display.
;
;The 8-bit instruction should be provided in register s5.
;Instructions are written using the following sequence
; Upper nibble
; wait >1us
; Lower nibble
; wait >40us
;
;Registers used s0, s1, s4, s5
;
LCD_write_inst8: LOAD s4, s5
               AND s4, F0                      ;Enable=0 RS=0

```

```

;Instruction, RW=0
;Write, E=0

;
OR s4, LCD_drive           ;Enable=1
CALL LCD_write_inst4       ;write upper nibble
CALL delay_1us             ;wait >1us
LOAD s4, s5                ;select lower nibble
                             ;with
SL1 s4                     ;Enable=1
SL0 s4                     ;RS=0 Instruction
SL0 s4                     ;RW=0 Write
SL0 s4                     ;E=0
CALL LCD_write_inst4       ;write lower nibble
CALL delay_40us            ;wait >40us
LOAD s4, F0                ;Enable=0 RS=0
                             ;Instruction, RW=0
                             ;Write, E=0
OUTPUT s4, LCD_output_port ;Release master enable
RETURN

;
;Write 8-bit data to LCD display.
;
;The 8-bit data should be provided in register s5.
;Data bytes are written using the following sequence
; Upper nibble
; wait >1us
; Lower nibble
; wait >40us
;
;Registers used s0, s1, s4, s5
;
LCD_write_data: LOAD s4, s5
AND s4, F0                ;Enable=0 RS=0
                             ;Instruction, RW=0
                             ;Write, E=0

;
OR s4, 0C                 ;Enable=1 RS=1 Data,
                             ;RW=0 Write, E=0

;
OUTPUT s4, LCD_output_port ;set up RS and RW >40ns
                             ;before enable pulse

;
CALL LCD_pulse_E           ;write upper nibble
CALL delay_1us             ;wait >1us
LOAD s4, s5                ;select lower nibble
                             ;with
SL1 s4                     ;Enable=1
SL1 s4                     ;RS=1 Data
SL0 s4                     ;RW=0 Write
SL0 s4                     ;E=0
OUTPUT s4, LCD_output_port ;set up RS and RW >40ns
                             ;before enable pulse

```

```

;
CALL LCD_pulse_E           ;write lower nibble
CALL delay_40us           ;wait >40us
LOAD s4, F0               ;Enable=0 RS=0
                           ;Instruction, RW=0
                           ;Write, E=0

;
OUTPUT s4, LCD_output_port ;Release master enable
RETURN

;
;Read 8-bit data from LCD display.
;
;The 8-bit data will be read from the current LCD memory
;address and will be returned in register s5.
;It is advisable to set the LCD address (cursor position)
;before using the data read for the first time otherwise
;the display may generate invalid data on the first read.
;
;Data bytes are read using the following sequence
; Upper nibble
; wait >1us
; Lower nibble
; wait >40us
;
;Registers used s0, s1, s4, s5
;
LCD_read_data8: LOAD s4, 0E           ;Enable=1 RS=1 Data,
                                       ;RW=1 Read, E=0

;
OUTPUT s4, LCD_output_port           ;set up RS and RW >40ns
                                       ;before enable pulse

;
XOR s4, LCD_E                         ;E=1
OUTPUT s4, LCD_output_port
CALL delay_1us                        ;wait >260ns to
                                       ;access data

;
INPUT s5, LCD_input_port              ;read upper nibble
XOR s4, LCD_E                         ;E=0
OUTPUT s4, LCD_output_port
CALL delay_1us                        ;wait >1us
XOR s4, LCD_E                         ;E=1
OUTPUT s4, LCD_output_port
CALL delay_1us                        ;wait >260ns to
                                       ;access data

;
INPUT s0, LCD_input_port              ;read lower nibble
XOR s4, LCD_E                         ;E=0
OUTPUT s4, LCD_output_port
AND s5, F0                            ;merge upper and
                                       ;lower nibbles

SRO s0

```

```

        SR0 s0
        SR0 s0
        SR0 s0
        OR s5, s0
        LOAD s4, 04                                ;Enable=0 RS=1 Data,
                                                    ;RW=0 Write, E=0

        ;
        OUTPUT s4, LCD_output_port                ;Stop reading 5V
                                                    ;device and release
                                                    ;master enable

        ;
        CALL delay_40us                            ;wait >40us
        RETURN

        ;
        ;Reset and initialise display to communicate using 4-bit
        ;data mode
        ;Includes routine to clear the display.
        ;
        ;Requires the 4-bit instructions 3,3,3,2 to be sent with
        ;suitable delays following by the 8-bit instructions to
        ;set up the display.
        ;
        ; 28 = '001' Function set, '0' 4-bit mode, '1' 2-line,
        ;      '0' 5x7 dot matrix, 'xx'
        ; 06 = '000001' Entry mode, '1' increment, '0'
        ;      no display shift
        ; 0C = '00001' Display control, '1' display on,
        ;      '0' cursor off, '0' cursor blink off
        ; 01 = '00000001' Display clear
        ;
        ;Registers used s0, s1, s2, s3, s4
        ;
LCD_reset: CALL delay_20ms                        ;wait more than
                                                    ;15ms for display
                                                    ;to be ready

        ;
        LOAD s4, 30
        CALL LCD_write_inst4                       ;send '3'
        CALL delay_20ms                            ;wait >4.1ms
        CALL LCD_write_inst4                       ;send '3'
        CALL delay_1ms                             ;wait >100us
        CALL LCD_write_inst4                       ;send '3'
        CALL delay_40us                            ;wait >40us
        LOAD s4, 20
        CALL LCD_write_inst4                       ;send '2'
        CALL delay_40us                            ;wait >40us
        LOAD s5, 28                                ;Function set
        CALL LCD_write_inst8
        LOAD s5, 06                                ;Entry mode
        CALL LCD_write_inst8
        LOAD s5, 0C                                ;Display control
        CALL LCD_write_inst8

```

```
LCD_clear: LOAD s5, 01                ;Display clear
          CALL LCD_write_inst8
          CALL delay_1ms              ;wait >1.64ms for
                                     ;display to clear
          CALL delay_1ms
          RETURN
          ;
```

Appendix C – Code for Part C

C.1 hello.psm

```

;*****
;Port definitions
;*****
;
CONSTANT LED_port, 80          ;8 simple LEDs
CONSTANT LED0, 01              ;    LED 0 - bit0
CONSTANT LED1, 02              ;    1 - bit1
CONSTANT LED2, 04              ;    2 - bit2
CONSTANT LED3, 08              ;    3 - bit3
CONSTANT LED4, 10              ;    4 - bit4
CONSTANT LED5, 20              ;    5 - bit5
CONSTANT LED6, 40              ;    6 - bit6
CONSTANT LED7, 80              ;    7 - bit7
;
CONSTANT switch_port, 00      ;Read switches and
                               ;press buttons
;
CONSTANT switch0, 01          ; Switches SW0 - bit0
CONSTANT switch1, 02          ;    SW1 - bit1
CONSTANT switch2, 04          ;    SW2 - bit2
CONSTANT switch3, 08          ;    SW3 - bit3
CONSTANT BTN_east, 10         ; Buttons East - bit4
CONSTANT BTN_south, 20        ;    South - bit5
CONSTANT BTN_north, 40        ;    North - bit6
CONSTANT BTN_west, 80         ;    West - bit7
;
;
;LCD interface ports
;
;The master enable signal is not used by the LCD display
;itself but may be required to confirm that LCD
;communication is active.
;This is required on the Spartan-3E Starter Kit if the
;StrataFLASH is used because it shares the same data pins
;and conflicts must be avoided.
;
CONSTANT LCD_output_port, 40   ;LCD character module
                               ;output data and control
CONSTANT LCD_E, 01            ;    active High Enable E - bit0
CONSTANT LCD_RW, 02           ;    Read=1 Write=0    RW - bit1
CONSTANT LCD_RS, 04           ;    Instruction=0 Data=1 RS - bit2
CONSTANT LCD_drive, 08        ;Master enable(active High)- bit3
CONSTANT LCD_DB4, 10          ; 4-bit                Data DB4 - bit4
CONSTANT LCD_DB5, 20          ; interface            Data DB5 - bit5
CONSTANT LCD_DB6, 40          ;                    Data DB6 - bit6
CONSTANT LCD_DB7, 80          ;                    Data DB7 - bit7
;

```

```

;
CONSTANT LCD_input_port, 02      ;LCD character module
                                ;input data
CONSTANT LCD_read_spare0, 01     ; Spare bits           - bit0
CONSTANT LCD_read_spare1, 02     ; are zero             - bit1
CONSTANT LCD_read_spare2, 04     ;                      - bit2
CONSTANT LCD_read_spare3, 08     ;                      - bit3
CONSTANT LCD_read_DB4, 10        ;    4-bit Data DB4 - bit4
CONSTANT LCD_read_DB5, 20        ;interface Data DB5 - bit5
CONSTANT LCD_read_DB6, 40        ;          Data DB6 - bit6
CONSTANT LCD_read_DB7, 80        ;          Data DB7 - bit7
;
;*****
;Useful data constants
;*****
;
;The main operation of the program uses lms delays to
;set the shift rate of the LCD display. A 16-bit value
;determines how many milliseconds there are between
;shifts
;
;Tests indicate that the fastest shift rate that the LCD
;display supports is 500ms. Faster than this and the
;display becomes less clear to read.
;
CONSTANT shift_delay_msb, 01      ;delay is 500ms (01F4 hex)
CONSTANT shift_delay_lsb, F4
;
;Constant to define a software delay of lus. This must be
;adjusted to reflect the clock applied to KCPSM3. Every
;instruction executes in 2 clock cycles making the
;calculation highly predictable. The '6' in the following
;equation even allows for 'CALL delay_lus' instruction in
;the initiating code.
;
;delay_lus_constant = (clock_rate - 6)/4
;Where 'clock_rate' is in MHz
;
;Example: For a 50MHz clock the constant value is
;          (10-6)/4 = 11 (0B Hex).
;For clock rates below 10MHz the value of 1 must be used
;and the operation will become lower than intended.
;
CONSTANT delay_lus_constant, 0B
;
;*****
;Initialise the system
;*****
;
cold_start: CALL LCD_reset          ;initialise LCD display
            CALL Setup_cust_chars   ;set up customised characters
            CALL Disp_mesg          ;display message

```

```

;
cold_start: CALL LCD_reset           ;initialise LCD display
            CALL Setup_cust_chars    ;set up customised characters
            CALL Disp_mesg           ;display message
;
            ENABLE INTERRUPT         ;enable interrupts
            LOAD s9, 01              ;initialise display flag to on
;
Main: LOAD sF, shift_delay_msb       ;[sF,sE]=loop delay in ms
      LOAD sE, shift_delay_lsb

LCD_delay_loop: CALL delay_lms        ;lms delay
               SUB sE, 01            ;decrement delay counter
               SUBCY sF, 00
               JUMP NC, LCD_delay_loop

               INPUT s7, switch_port ;read in switches
               OUTPUT s7, LED_port   ;output to LEDs

               TEST s7, switch3      ;check for switch3 (left)
               JUMP Z, Check_right

               LOAD s5, 18            ;shift display left
               CALL LCD_write_inst8

Check_right: TEST s7, switch2         ;check for switch2 (right)
              JUMP Z, End_loop

              LOAD s5, 1C             ;shift display right
              CALL LCD_write_inst8

End_loop: JUMP Main
;
;
Disp_mesg: LOAD s5, 80                ;Line 1 position 1
          CALL LCD_write_inst8

          ;total 40 chars written to line 1
          CALL Disp_Hello_World       ;12 chars
          CALL Disp_cust_chars        ; 4 chars
          CALL Disp_Hello_World       ;12 chars
          CALL Disp_cust_chars        ; 4 chars
          CALL Disp_cust_chars        ; 4 chars

          CALL Disp_cust_chars        ; 4 chars

          LOAD s5, C0                 ;Line 2 position 1
          CALL LCD_write_inst8

          ;total 40 chars written to line 2
          CALL Disp_Hello_World       ;12 chars
          CALL Disp_cust_chars        ; 4 chars

```

```

        CALL Disp_Hello_World          ;12 chars
        CALL Disp_cust_chars           ; 4 chars
        CALL Disp_cust_chars           ; 4 chars

        CALL Disp_cust_chars           ; 4 chars

        RETURN
;
Disp_Hello_World: LOAD s5, 48           ;H
        CALL LCD_write_data
        LOAD s5, 65                     ;e
        CALL LCD_write_data
        LOAD s5, 6C                     ;l
        CALL LCD_write_data
        ;LOAD s5, 6C                     ;l
        CALL LCD_write_data
        LOAD s5, 6F                     ;o
        CALL LCD_write_data
        LOAD s5, 20                     ;space
        CALL LCD_write_data
        LOAD s5, 57                     ;W
        CALL LCD_write_data
        LOAD s5, 6F                     ;o
        CALL LCD_write_data
        LOAD s5, 72                     ;r
        CALL LCD_write_data
        LOAD s5, 6C                     ;l
        CALL LCD_write_data
        LOAD s5, 64                     ;d
        CALL LCD_write_data
        LOAD s5, 20                     ;space
        CALL LCD_write_data
        RETURN
;
Disp_cust_chars: LOAD s5, 00             ;CG RAM address 00, cust char
        CALL LCD_write_data
        CALL LCD_write_data

        LOAD s5, 01                     ;CG RAM address 01, cust char
        CALL LCD_write_data

        LOAD s5, 20                     ;space
        CALL LCD_write_data
        RETURN
;
Setup_cust_chars: LOAD s5, 40            ;set CG RAM address 00
        CALL LCD_write_inst8

        LOAD s5, 00                     ;line 1
        CALL LCD_write_data
        LOAD s5, 0E                     ;line 2
        CALL LCD_write_data

```

```

LOAD s5, 1F                                ;line 3
CALL LCD_write_data
LOAD s5, 15                                ;line 4
CALL LCD_write_data
LOAD s5, 1F                                ;line 5
CALL LCD_write_data
LOAD s5, 1F                                ;line 6
CALL LCD_write_data
LOAD s5, 15                                ;line 7
CALL LCD_write_data
LOAD s5, 00                                ;line 8
CALL LCD_write_data

LOAD s5, 48                                ;set CG RAM address 01
CALL LCD_write_inst8

LOAD s5, 0E                                ;line 1
CALL LCD_write_data
LOAD s5, 1F                                ;line 2
CALL LCD_write_data
LOAD s5, 0F                                ;line 3
CALL LCD_write_data
LOAD s5, 07                                ;line 4
CALL LCD_write_data
LOAD s5, 0F                                ;line 5
CALL LCD_write_data
LOAD s5, 1F                                ;line 6
CALL LCD_write_data
LOAD s5, 0E                                ;line 7
CALL LCD_write_data
LOAD s5, 00                                ;line 8
CALL LCD_write_data
RETURN
;
;
;*****
;Software delay routines
;*****
;
;Delay of 1us.
;
;Constant value defines reflects the clock applied to
;KCPSM3. Every instruction executes in 2 clock cycles
;making the calculation highly predictable. The '6' in
;the following equation even allows for 'CALL delay_1us'
;instruction in the initiating code.
;
; delay_1us_constant = (clock_rate - 6)/4
; Where 'clock_rate' is in MHz
;
;Registers used s0
;

```

```

delay_1us: LOAD s0, delay_1us_constant
wait_1us: SUB s0, 01
          JUMP NZ, wait_1us
          RETURN
          ;
          ;Delay of 40us.
          ;
          ;Registers used s0, s1
          ;
delay_40us: LOAD s1, 28                                ;40 x 1us = 40us
wait_40us: CALL delay_1us
          SUB s1, 01
          JUMP NZ, wait_40us
          RETURN
          ;
          ;
          ;Delay of 1ms.
          ;
          ;Registers used s0, s1, s2
          ;
delay_1ms: LOAD s2, 19                                ;25 x 40us = 1ms
wait_1ms: CALL delay_40us
          SUB s2, 01
          JUMP NZ, wait_1ms
          RETURN
          ;
          ;Delay of 20ms.
          ;
          ;Delay of 20ms used during initialisation.
          ;
          ;Registers used s0, s1, s2, s3
          ;
delay_20ms: LOAD s3, 14                                ;20 x 1ms = 20ms
wait_20ms: CALL delay_1ms
          SUB s3, 01
          JUMP NZ, wait_20ms
          RETURN
          ;
          ;Delay of approximately 1 second.
          ;
          ;Registers used s0, s1, s2, s3, s4
          ;
delay_1s: LOAD s4, 32                                  ;50 x 20ms = 1000ms
wait_1s: CALL delay_20ms
          SUB s4, 01
          JUMP NZ, wait_1s
          RETURN
          ;
          ;
          ;
          ;*****
          ;LCD Character Module Routines

```

```

;*****
;
;LCD module is a 16 character by 2 line display but all
;displays are very similar
;The 4-wire data interface will be used (DB4 to DB7).
;
;The LCD modules are relatively slow and software delay
;loops are used to slow down KCPSM3 adequately for the
;LCD to communicate. The delay routines are provided in
;a different section (see above in this case).
;
;
;Pulse LCD enable signal 'E' high for greater than 230ns
;(1us is used).
;
;Register s4 should define the current state of the LCD
;output port.
;
;Registers used s0, s4
;
LCD_pulse_E: XOR s4, LCD_E                ;E=1
              OUTPUT s4, LCD_output_port
              CALL delay_1us
              XOR s4, LCD_E                ;E=0
              OUTPUT s4, LCD_output_port
              RETURN
;
;Write 4-bit instruction to LCD display.
;
;The 4-bit instruction should be provided in the upper
;4-bits of register s4. Note that this routine does not
;release the master enable but as it is only used during
;initialisation and as part of the 8-bit instruction
;write it should be acceptable.
;
;Registers used s4
;
LCD_write_inst4: AND s4, F8                ;Enable=1 RS=0
                                                         ;Instruction, RW=0
                                                         ;Write, E=0
              OUTPUT s4, LCD_output_port    ;set up RS and RW >40ns
                                                         ;before enable pulse
              CALL LCD_pulse_E
              RETURN
;
;
;Write 8-bit instruction to LCD display.
;
;The 8-bit instruction should be provided in register s5.
;Instructions are written using the following sequence
; Upper nibble
; wait >1us

```



```

; Lower nibble
; wait >40us
;
;Registers used s0, s1, s4, s5
;
LCD_write_inst8: LOAD s4, s5
AND s4, F0
;Enable=0 RS=0
;Instruction, RW=0
;Write, E=0

;
OR s4, LCD_drive
;Enable=1
CALL LCD_write_inst4
;write upper nibble
CALL delay_1us
;wait >1us
LOAD s4, s5
;select lower nibble
;with
SL1 s4
;Enable=1
SL0 s4
;RS=0 Instruction
SL0 s4
;RW=0 Write
SL0 s4
;E=0
CALL LCD_write_inst4
;write lower nibble
CALL delay_40us
;wait >40us
LOAD s4, F0
;Enable=0 RS=0
;Instruction, RW=0
;Write, E=0

OUTPUT s4, LCD_output_port
;Release master enable
RETURN

;
;Write 8-bit data to LCD display.
;
;The 8-bit data should be provided in register s5.
;Data bytes are written using the following sequence
; Upper nibble
; wait >1us
; Lower nibble
; wait >40us
;
;Registers used s0, s1, s4, s5
;
LCD_write_data: LOAD s4, s5
AND s4, F0
;Enable=0 RS=0
;Instruction, RW=0
;Write, E=0

;
OR s4, 0C
;Enable=1 RS=1 Data,
;RW=0 Write, E=0

;
OUTPUT s4, LCD_output_port
;set up RS and RW >40ns
;before enable pulse

;
CALL LCD_pulse_E
;write upper nibble
CALL delay_1us
;wait >1us
LOAD s4, s5
;select lower nibble

```

```

SL1 s4
SL1 s4
SL0 s4
SL0 s4
OUTPUT s4, LCD_output_port

;
CALL LCD_pulse_E
CALL delay_40us
LOAD s4, F0

;
OUTPUT s4, LCD_output_port
RETURN

;
;Read 8-bit data from LCD display.
;
;The 8-bit data will be read from the current LCD memory
;address and will be returned in register s5.
;It is advisable to set the LCD address (cursor position)
;before using the data read for the first time otherwise
;the display may generate invalid data on the first read.
;
;Data bytes are read using the following sequence
; Upper nibble
; wait >1us
; Lower nibble
; wait >40us
;
;Registers used s0, s1, s4, s5
;
LCD_read_data8: LOAD s4, 0E

;
OUTPUT s4, LCD_output_port

;
XOR s4, LCD_E
OUTPUT s4, LCD_output_port
CALL delay_1us

;
INPUT s5, LCD_input_port
XOR s4, LCD_E
OUTPUT s4, LCD_output_port
CALL delay_1us
XOR s4, LCD_E
OUTPUT s4, LCD_output_port
CALL delay_1us

;with
;Enable=1
;RS=1 Data
;RW=0 Write
;E=0
;set up RS and RW >40ns
;before enable pulse

;write lower nibble
;wait >40us
;Enable=0 RS=0
;Instruction, RW=0
;Write, E=0

;Release master enable
;Enable=1 RS=1 Data,
;RW=1 Read, E=0

;set up RS and RW >40ns
;before enable pulse

;E=1

;wait >260ns to
;access data

;read upper nibble
;E=0

;wait >1us
;E=1

;wait >260ns to
;access data

```

```

;
INPUT s0, LCD_input_port           ;read lower nibble
XOR s4, LCD_E                      ;E=0
OUTPUT s4, LCD_output_port
AND s5, F0                         ;merge upper and
                                   ;lower nibbles

SR0 s0
SR0 s0
SR0 s0
SR0 s0
OR s5, s0
LOAD s4, 04                        ;Enable=0 RS=1 Data,
                                   ;RW=0 Write, E=0

;
OUTPUT s4, LCD_output_port         ;Stop reading 5V
                                   ;device and release
                                   ;master enable

;
CALL delay_40us                    ;wait >40us
RETURN

;
;Reset and initialise display to communicate using 4-bit
;data mode
;Includes routine to clear the display.
;
;Requires the 4-bit instructions 3,3,3,2 to be sent with
;suitable delays following by the 8-bit instructions to
;set up the display.
;
; 28 = '001' Function set, '0' 4-bit mode, '1' 2-line,
;      '0' 5x7 dot matrix, 'xx'
; 06 = '000001' Entry mode, '1' increment, '0'
;      no display shift
; 0C = '00001' Display control, '1' display on,
;      '0' cursor off, '0' cursor blink off
; 01 = '00000001' Display clear
;
;Registers used s0, s1, s2, s3, s4
;
LCD_reset: CALL delay_20ms          ;wait more than
                                   ;15ms for display
                                   ;to be ready

;
LOAD s4, 30
CALL LCD_write_inst4               ;send '3'
CALL delay_20ms                    ;wait >4.1ms
CALL LCD_write_inst4               ;send '3'
CALL delay_1ms                     ;wait >100us
CALL LCD_write_inst4               ;send '3'
CALL delay_40us                    ;wait >40us
LOAD s4, 20
CALL LCD_write_inst4               ;send '2'

```

```

        CALL delay_40us                ;wait >40us
        LOAD s5, 28                    ;Function set
        CALL LCD_write_inst8
        LOAD s5, 06                    ;Entry mode
        CALL LCD_write_inst8
        LOAD s5, 0C                    ;Display control
        CALL LCD_write_inst8
LCD_clear: LOAD s5, 01                  ;Display clear
        CALL LCD_write_inst8
        CALL delay_1ms                ;wait >1.64ms for
                                       ;display to clear

        CALL delay_1ms
        RETURN

;
;*****
;Interrupt Service Routine
;*****
;
ISR: TEST s7, switch1                ;check for switch1 (flash)
    JUMP Z, Test_disp_on
;
    XOR s9, 01                       ;toggle display flag
    TEST s9, 01                      ;check if display should
                                       ;be turned on or off

    JUMP NZ, Disp_on
;
    LOAD s5, 08                      ;turn display off
    JUMP End_disp
;
Disp_on: LOAD s5, 0C                  ;turn display on
        JUMP End_disp
;
Test_disp_on: TEST s9, 01             ;if flash off, make sure
                                       ;the display is on

        JUMP NZ, End_ISR
;
        LOAD s9, 01                  ;set display flag on
        LOAD s5, 0C                  ;turn display on
;
End_disp: CALL LCD_write_inst8
;
End_ISR: RETURNI ENABLE
;
;-----
;
ADDRESS 3FF
JUMP ISR
;
;

```

C.2 top_level.vhd

```

-----
-- Company:
-- Engineer:
--
-- Create Date:      13:20:51 02/04/2013
-- 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;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.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 (
        led : out std_logic_vector (7 downto 0);
        strataflash_oe : out std_logic;
        strataflash_ce : out std_logic;
        strataflash_we : out std_logic;
        switch : in std_logic_vector (3 downto 0);
        btn_north : in std_logic;
        btn_east : in std_logic;
        btn_south : in std_logic;
        btn_west : in std_logic;
        lcd_d : inout std_logic_vector (7 downto 4);
        lcd_rs : out std_logic;
        lcd_rw : out std_logic;
        lcd_e : out std_logic;
        clk : in std_logic);
end top_level;

```

```

--
-----
--
-- Start of test architecture
--
architecture Behavioral of top_level is
--
-----
--
-- declaration of KCPSM3
--
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 ROM
--
component hello
    Port (
        address : in std_logic_vector (9 downto 0);
        instruction : out std_logic_vector (17 downto 0);
        --proc_reset : out std_logic;           --JTAG Loader version
        clk : in std_logic);
    end component;
--
-----
--
-- Signals used to connect KCPSM3 to program ROM 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         : std_logic := '0';
signal interrupt_ack     : std_logic;
signal kcpsm3_reset     : std_logic;
--
--
-- Signals for LCD operation

```

```

--
-- Tri-state output requires internal signals
-- 'lcd_drive' is used to differentiate between LCD and StrataFLASH
-- communications which share the same data bits.
--
signal    lcd_rw_control : std_logic;
signal    lcd_output_data : std_logic_vector (7 downto 4);
signal    lcd_drive : std_logic;
--
-- Interrupt signals
signal count_event      : std_logic;
signal count_ticks      : std_logic_vector(25 downto 0) :=
"00000000000000000000000000000000";

-- 12.5e6 ticks = 0.25 seconds, 50MHz clock
--constant count_limit_min: std_logic_vector(25 downto 0) :=
"00110111101011110000100000";
--
-- 25e6 ticks = 0.5 seconds, 50MHz clock
constant count_limit      : std_logic_vector(25 downto 0) :=
"0110111101011111000010000000";
--
-- 50e6 ticks = 1 second, 50MHz clock
--constant count_limit_max: std_logic_vector(25 downto 0) :=
"1011111010111110000100000000";
--
--
-----
-----
begin
    --
    -----
    -- LCD interface
    -----
    --
    -- The 4-bit data port is bidirectional.
    -- lcd_rw is '1' for read and '0' for write
    -- lcd_drive is like a master enable signal which prevents either the
    -- FPGA outputs or the LCD display driving the data lines.
    --
    -- Control of read and write signal
    lcd_rw <= lcd_rw_control and lcd_drive;

    -- use read/write control to enable output buffers.
    lcd_d <= lcd_output_data when (lcd_rw_control='0' and lcd_drive='1')
        else "ZZZZ";
    --
    -----
    -- Disable StrataFLASH
    -----
    --
    strataflash_oe <= '1';
    strataflash_ce <= '1';
    strataflash_we <= '1';

```

```

--
-----
-- KCPSM3 and the program memory
-----
--
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 => kcpsm3_reset,
    clk => clk);

program_rom: hello
  port map(
    address => address,
    instruction => instruction,
    --proc_reset => kcpsm3_reset,    --JTAG Loader version
    clk => clk);

--
-----
-- KCPSM3 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
      -- read simple toggle switches and buttons at address 00 hex
      when "00" =>    in_port <= btn_west & btn_north & btn_south
                        & btn_east & switch;

      -- read LCD data at address 02 hex
      when "10" =>    in_port <= lcd_d & "0000";

      -- 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 output ports
-----

```

```

--
-- adding the output registers to the processor
--
output_ports: process(clk)
begin
    if clk'event and clk='1' then
        if write_strobe='1' then

            -- Write to LEDs at address 80 hex.
            if port_id(7)='1' then
                led <= out_port;
            end if;

            -- LCD data output and controls at address 40 hex.
            if port_id(6)='1' then
                lcd_output_data <= out_port(7 downto 4);
                lcd_drive <= out_port(3);
                lcd_rs <= out_port(2);
                lcd_rw_control <= out_port(1);
                lcd_e <= out_port(0);
            end if;
        end if;
    end if;
end process output_ports;
--
-----
-- interrupt
-----
--
-- sets an interrupt flag based on a counter
--
counter: process(clk)
begin
    if clk'event and clk='1' then
        count_ticks <= count_ticks + 1;
        if count_ticks > count_limit then
            count_event <= '1';
            count_ticks <= (others => '0');
        else
            count_event <= '0';
        end if;
    end if;
end process counter;
--
-----
-- interrupt handler
-----
--
interrupt_control: process(clk)
begin
    if clk'event and clk='1' then
        if interrupt_ack = '1' then

```

```
        interrupt <= '0';
    elsif count_event = '1' then
        interrupt <= '1';
    else
        interrupt <= interrupt;
    end if;
end if;
end process interrupt_control;

end Behavioral;
```