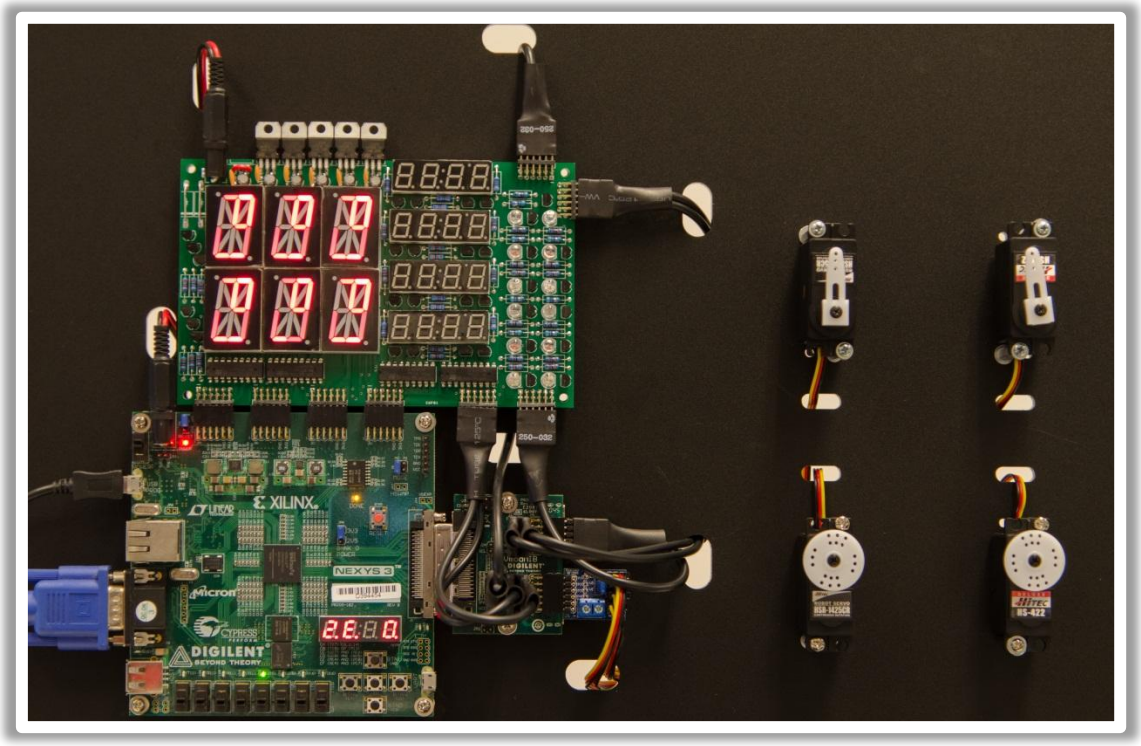
UNIVERSITY OF TECHNOLOGY SYDNEY

NEXYS FPGA RIG
LABORATORY USER GUIDE

VERSION 1.1



Partnered with…

labshare

UTS remotelabs

Table of Contents

Revision History

Rev	Date	Details	By
1.0	24/07/2012	Draft Created	LJC
1.1	27/08/2012	Initial Release	LJC
1.2	30/08/2012	Virtual Input information added	DA

1 Introduction

1.1 Remote Laboratories

Remote laboratories enable students to access physical laboratory apparatus through the internet, providing a supplement to their studies and existing hands-on experience. Students carry out experiments using real equipment, but with much greater flexibility since access can occur from anywhere and at any time. Their interaction with the remote equipment is assisted by the use of data acquisition instrumentation and cameras, providing direct feedback to students for better engagement.

Traditional engineering laboratories require students to be physically present in order to work with equipment, which may limit student flexibility. Conversely, remote laboratories let students work in their own time and even repeat experiments for better learning outcomes.

Of course students cannot actually touch and feel the equipment in a remote laboratory, but they can still perform most other tasks relevant to their learning. Sometimes, separation from potentially hazardous equipment is preferable from a safety point of view.

Due to the increased use of remote operation in industry, where machinery and entire plants are often controlled from a distant location, students may directly benefit from learning how to remotely control equipment. Furthermore, remote laboratories provide the opportunity to access a wider range of experiments as costly or highly specialised equipment may not be locally available. This presents the opportunity to share laboratory facilities between institutions.

Significant research and pilot studies have been undertaken in Australia and by several groups around the world into the educational effectiveness of using remote laboratories. These studies have consistently shown that, if used appropriately in a way that is cognizant of the intended educational outcomes of the laboratory experience, remote laboratories can provide significant benefits.

Indeed, multiple research studies have demonstrated that whilst there are some learning outcomes that are achieved more effectively through hands-on experimentation (e.g. identification of assumptions, specific haptic skills), there are other learning outcomes that are achieved more effectively through remotely accessed laboratories (e.g. processing of data, understanding of concepts).

Engineering students are able to access the Nexys FPGA rigs to help them record, analyse and compare data from a range of sensors and develop, simulate and test localisation and mapping algorithms.

1.2 NEXYS FPGA - The Rig Apparatus

The NEXYS FPGA (Field Programmable Gate Array) rig was designed to allow students to explore and understand modern re-programmable semiconductor devices and how to interface them with common displays, indicators (such as Seven Segment Displays & LEDs) and more complicated display systems that operate via a VGA signal.

Using a web-based interface, students are able to remotely upload bitstreams to the NEXYS FPGA rig and toggle switches and outputs. The FPGA is capable of controlling alpha-numeric displays, seven segment displays, red/orange/green LEDs and servo motors.

Each NEXYS FPGA Rig consists of the following main components:

- 1x Digilent NEXYS 3 Spartan-6 FPGA Board
- 1x UTS Remote Labs Display Board
  - *6x Alpha-Numeric Displays*
  - *4x Four-Character Seven-Segment Displays*
  - *4x Red LEDs*
  - *4x Amber LEDs*
  - *4x Green LEDs*
- 1x Dell 1707FP 17" LCD Monitor
- 1x Digilent Vmod Module Interface Board (VmodMIB)
  - 1x Digilent PModCON3 Sero Board
    - *2x Hitec HS-422 Deluxe Servo Motors*
    - *2x Hitec HSR-1425CR Continuous Rotation Servo Motors*

Additionally, each NEXYS FPGA rig is monitored by a web camera – providing an overview of the Spartan-6 board, display board, servo motors & LCD monitor.

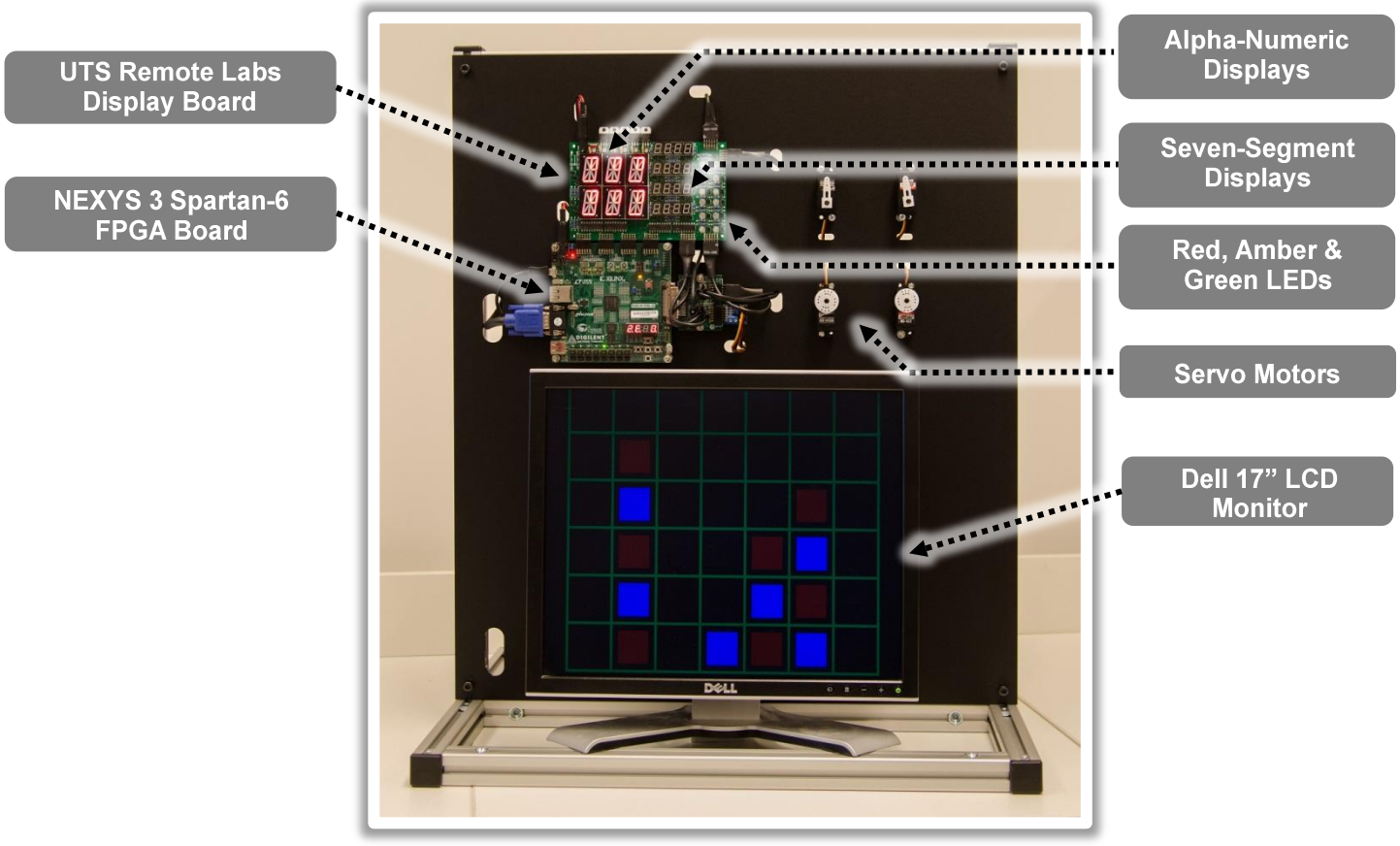


Figure 1: NEXYS FPGA Rig.

2 Rig Specifications

2.1 Digilent Nexys3 Spartan-6 FPGA Board

The heart of the rig is the Digilent Nexys3 Spartan-6 FPGA board. The Nexys3 is a well-featured platform that allows engineering students to develop and test designs ranging from simple timers and displays to complete digital systems such as controllers and signal processors.

The board has the following features:

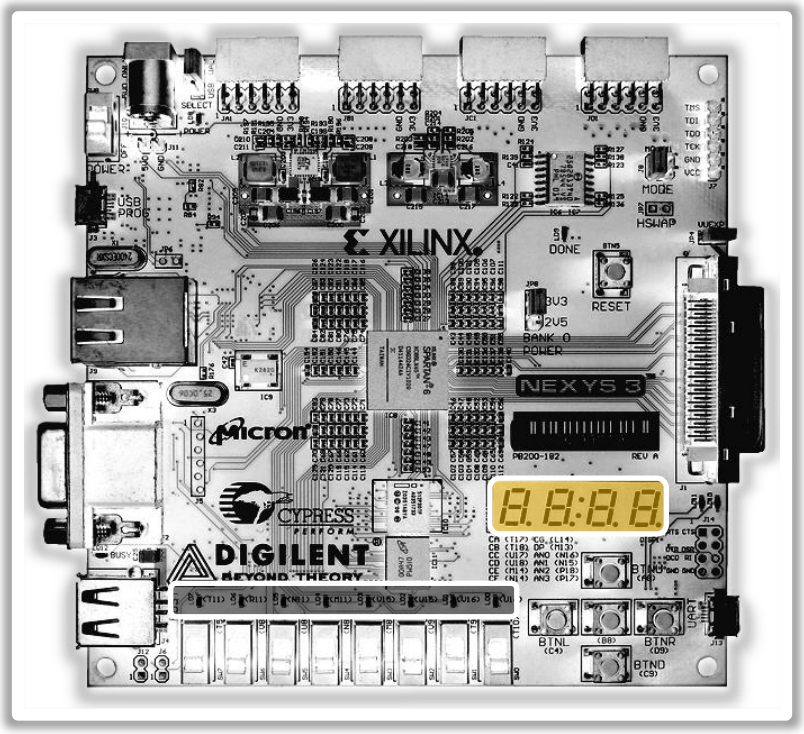
- Xilinx Spartan6 XC6LX16-CS324 FPGA
- 16Mbyte Micron Cellular RAM
- 16Mbyte Micron Parallel PCM
- 16Mbyte Micron Quad-mode SPI PCM
- 100MHz fixed-frequency oscillator
- 8 slide switches, 4 push buttons, 4-digit 7seg display, 8 LEDs
- 8-bit VGA Output
- 10/100 SMSC LAN8710 PHY
- USB-UART
- Digilent Adept USB port for power, programming & data transfers
- Four double-wide Pmod™ connectors, one VHDC connector

The board interfaces with a number of other boards in order to provide additional I/O and connectivity to a number of devices.

Users are able to generate bitstreams using *Xilinx ISE Design Suite 13.x* and onwards – these can then be uploaded to the board via the Rig Control Software.

The following pages detail the on-board displays and indicators, installed add-on boards and attached devices. Users should be familiar with all of these components before programming the Nexys3 rig.

The image below shows the location of the on-board displays and indicators:

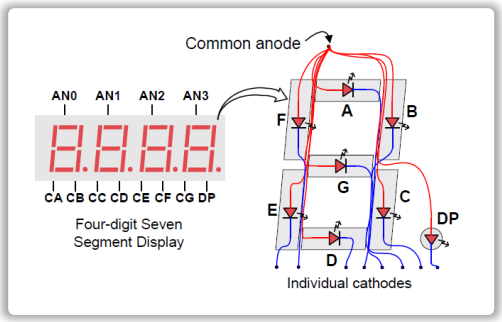


Key:

LED Bank

Seven Segment Display

2.1.1 On-board Seven Segment Displays



The Nexys3 board has a single four-digit seven segment display on-board. This display can be used just like the seven segment displays on the UTS Remote Labs Display Board. Illumination colour is red.

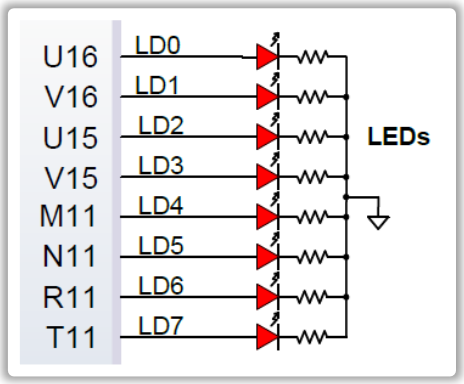
The *selection lines* operate on **inverse logic** – i.e. the output line controlling a display must be set low (0) in order to turn it on.

The *segments* operate on **inverse logic** – i.e. the output line controlling a segment must be set low (0) in order to turn it on.

The table below relates the FPGA Nets and I/O pins to the annotated segments on the left.

Segment	Net	Pin	Digit Selection	Net	Pin
A	SSEG_CA<0>	T17	0	SSEG_AN<0>	N16
B	SSEG_CA<1>	T18	1	SSEG_AN<1>	N15
C	SSEG_CA<2>	U17	2	SSEG_AN<2>	P18
D	SSEG_CA<3>	U18	3	SSEG_AN<3>	P17
E	SSEG_CA<4>	M14	-	-	-
F	SSEG_CA<5>	N14	-	-	-
G	SSEG_CA<6>	L14	-	-	-
DP	SSEG_CA<7>	M13	-	-	-

2.1.2 On-board LED Bank



The Nexys3 board has 8 surface-mount LEDs on-board. These LEDs can be used just like the LEDs on the UTS Remote Labs Display Board. Illumination colour is yellow.

The *LEDs* operate on **standard logic** – i.e. the output line controlling a segment must be set high (1) in order to turn it on.

The table below relates the FPGA Nets and I/O pins to the annotated segments on the left.

LED	Net	Pin	Digit Selection	Net	Pin
LED0	Led<0>	U16	LED4	Led<4>	M11
LED1	Led<1>	V16	LED5	Led<5>	N11
LED2	Led<2>	U15	LED6	Led<6>	R11
LED3	Led<3>	V15	LED7	Led<7>	T11

2.2 Dell 17" 1707FP LCD Monitor

Attached to the VGA port of the Nexys3 FPGA Board is a Dell 17" 1707FP LCD monitor.

Using VESA VGA signal timings, users are able to display images or graphics on the LCD monitor.

At the time of writing – the use of this device is beyond the scope of this user guide.

At a later stage, an updated user guide will be provided – indicating how to drive this display.

Channel	Net	Pin	Sync	Net	Pin
Red0	VGA_RED<0>	P8	Horizontal	VGA_HSYNC	N6
Red1	VGA_RED<1>	T6	Vertical	VGA_VSYNC	P7
Red2	VGA_RED<2>	V6	-	-	-
Green0	VGA_GREEN<0>	U7	-	-	-
Green1	VGA_GREEN<1>	V7	-	-	-
Green2	VGA_GREEN<2>	N7	-	-	-
Blue1	VGA_BLUE<0>	R7	-	-	-
Blue2	VGA_BLUE<1>	T7	-	-	-

2.3 Digilent VModMIB Interface Board

As noted above – a number of other boards interface with the Nexys3 FPGA board.

One such board is the Digilent VMod Module Interface Board (VModMIB). This is an additional peripheral board that connects via a 68-pin VHDI connector.

It provides 4x HDMI Type-D Micro connectors for driving displays and generating audio streams - as well as 5x 12-pin PMod connectors for additional I/O.

Four (4) of the PMod connectors are used to interface with the UTS Remote Labs I/O board and PModCON3 Servo Board.

2.4 Digilent PModCON3 Servo Board

The Digilent PModCON3 Servo Connector Module Board allows the Nexys FPGA rig to drive four servo motors. Power is selectable from either an external power supply or via Vcc from the Nexys3 FPGA board.

The rig has two types of servo motors attached to the PModCON3 board, standard (180° rotation) and continuous rotation (360° rotation).

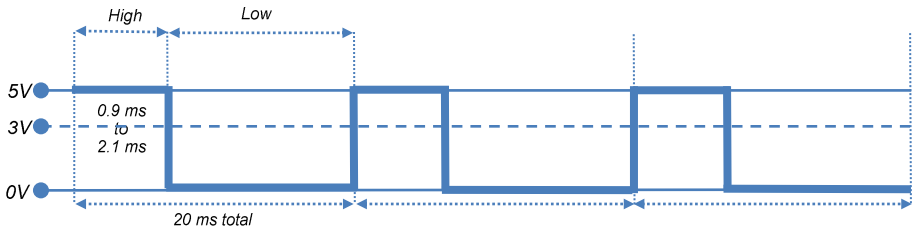
The table below relates the FPGA Nets and I/O pins to the servos attached to the PModCON3 board.

Servo	Net	Pin
HSR1425CR – Lower	Servo<0>	C8
HSR1425CR – Upper	Servo<1>	D8
HS422 – Upper	Servo<2>	F9
HS422 – Lower	Servo<3>	G9

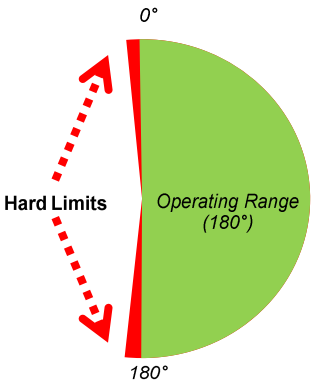
2.4.1 Servo Motor Waveform Specifications

In order to drive the servo motors, a square wave pulse must be sent to them. The high (3-5V peak) portion of the square wave can have a duration ranging from 0.9ms to 2.1ms with 1.5ms as the center. The pulse refreshes every 20ms (50Hz).

The diagram below illustrates the specifications of the pulse noted above:



2.4.2 Hitec HS-422 Deluxe Standard Servo Motor

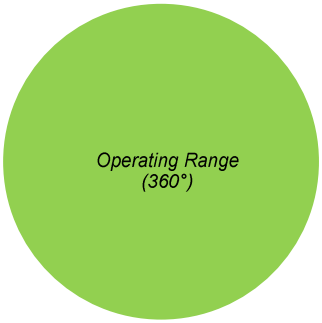


The Hitec HS-422 Deluxe is a durable 3 pole 180° rotation servo motor.

The motor is attached to the Digilent PModCON3 Servo Board and can be used by users for position indication. The upper and lower bounds of each motor's rotation on the rig has been aligned horizontally and vertically respectively. The table below lists the pulse width versus angle calibration values.

Angle	0°	45°	90°	135°	180°
High Pulse (ms)	0.55	1.00	1.45	1.90	2.35

2.4.3 Hitec HSR-1425CR Continuous Rotation Servo Motor



The Hitec HSR-1425CR is a 360° continuous rotation servo motor.

The motor is attached to the Digilent PModCON3 Servo Board and can be used by users as a means of timekeeping or providing feedback in the form of angular velocity. The table below lists the pulse width versus angle calibration values.

RPM	TBC
High Pulse (ms)	TBC

2.5 UTS Remote Labs Display Board

In order to make the most of the large number of I/O pins on the Nexys3 FPGA board, an additional board was developed in-house at UTS Remote Labs. This board holds six (6) alpha numeric displays, four (4) sets of four-digit seven segment displays and three (3) sets of red, amber and green LEDs.

Using these displays and indicators in conjunction with "virtual I/O" buttons and additional peripherals such as servo motors or the LCD display – users are able to create a rich variety of programs, from simple counters and clocks & simulation of traffic signals to simple games.

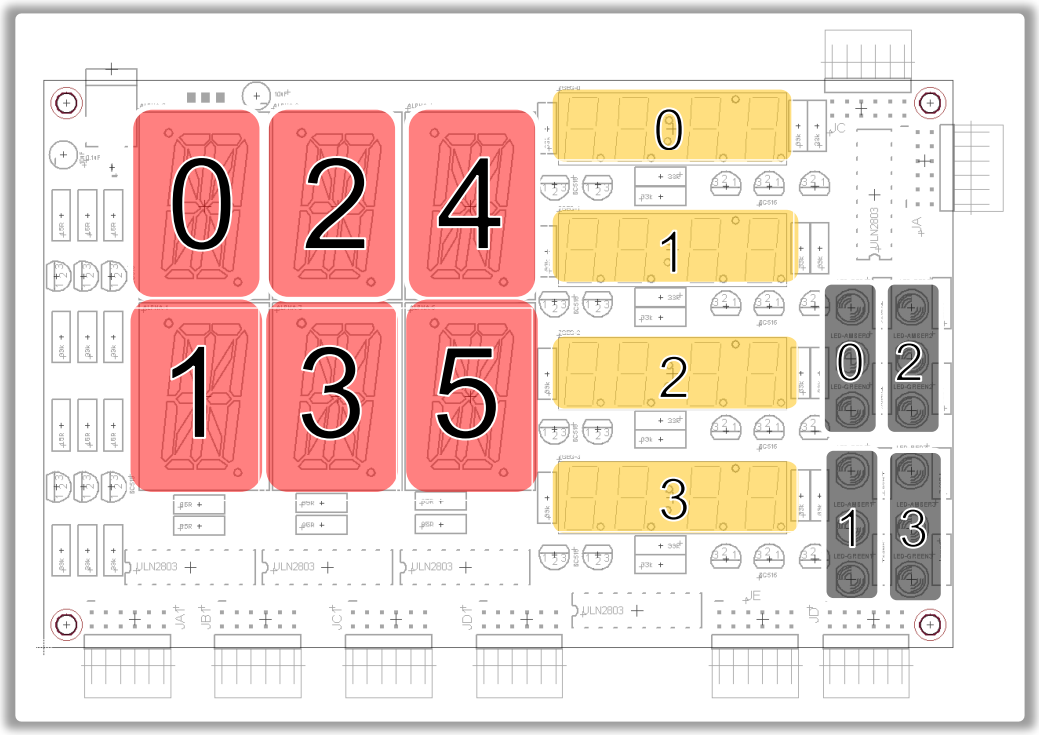


Figure 2: UTS Remote Labs Display Board Overview

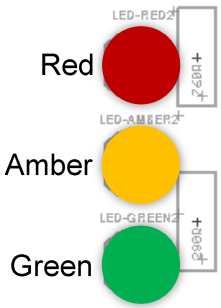
Key:

LED Bank

Alpha Numeric Display

Seven Segment Display

2.5.1 Red, Amber, Green LED Banks



The red, amber and green LEDs are China Young Sun LED Technology Co., LTD units. They are "super bright" T1 5mm packages with a rated output of 12,000 mcd.

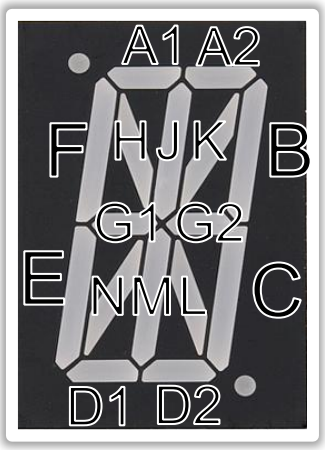
Their layout on the display board means they can be used to simulate the operation of traffic lights, or merely as status indicators.

The *LEDs* operate on **standard logic** – i.e. the output line controlling an LED must be set high (1) in order to turn it on.

The table below relates the FPGA Nets and I/O pins to the LEDs on the left, laid out as per the display board diagram.

LED	Net	Pin	LED	Net	Pin	LED	Net	Pin
Red 0	RedLed<0>	C7	Amber 0	AmberLed<0>	A7	Green 0	GreenLed<0>	B6
Red 1	RedLed<1>	C14	Amber 1	AmberLed<1>	F13	Green 1	GreenLed<1>	E13
Red 2	RedLed<2>	A6	Amber 2	AmberLed<2>	D11	Green 2	GreenLed<2>	D6
Red 3	RedLed<3>	C5	Amber 3	AmberLed<3>	A5	Green 3	GreenLed<3>	D14

2.5.2 Alpha Numeric Displays



The alpha numeric displays are Yetda Industry units. They are 1" high with black faces and white segments. Illumination colour is red.

The *selection lines* operate on **inverse logic** – i.e. the output line controlling a display must be set low (0) in order to turn it on.

The *segments* operate on **standard logic** – i.e. the output line controlling a segment must be set high (1) in order to turn it on.

The table below relates the FPGA Nets and I/O pins to the annotated segments on the left.

Segment	Net	Pin	Display Selection	Net	Pin
A1	AlphaHex<0>	V11	Alpha - Display 0	Alpha<0>	G11
A2	AlphaHex<1>	U11	Alpha - Display 1	Alpha<1>	F10
B	AlphaHex<2>	N9	Alpha - Display 2	Alpha<2>	F11
C	AlphaHex<3>	M10	Alpha - Display 3	Alpha<3>	E11
D1	AlphaHex<4>	P11	Alpha - Display 4	Alpha<4>	A4
D2	AlphaHex<5>	N10	Alpha - Display 5	Alpha<5>	B4
E	AlphaHex<6>	V12	-	-	-
F	AlphaHex<7>	T1	-	-	-
G1	AlphaHex<8>	K5	-	-	-
G2	AlphaHex<9>	K3	-	-	-
H	AlphaHex<10>	J1	-	-	-
J	AlphaHex<11>	J3	-	-	-
K	AlphaHex<12>	L3	-	-	-
L	AlphaHex<13>	L4	-	-	-
M	AlphaHex<14>	K1	-	-	-
N	AlphaHex<15>	K2	-	-	-

2.5.3 Seven Segment Displays

The seven segment displays are China Young Sun LED Technology Co., LTD units. They are ½" high with black faces and white segments. Illumination colour is red.

The *selection lines* operate on **inverse logic** – i.e. the output line controlling a segment must be set low (0) in order to turn it on.

The *segments* operate on **standard logic** – i.e. the output line controlling a segment must be set high (1) in order to turn it on.

The table below relates the FPGA Nets and I/O pins to the annotated segments on the left.

Segment	Net	Pin	Display Selection	Net	Pin
A	SSEGHex<0>	C13	COL – Display 0	SSEGCL<0>	A2
B	SSEGHex<1>	A13	COL – Display 1	SSEGCL<1>	B2
C	SSEGHex<2>	B11	COL – Display 2	SSEGCL<2>	A3
D	SSEGHex<3>	A11	COL – Display 3	SSEGCL<3>	B3
E	SSEGHex<4>	B14	Digit 0 – Display 0	SSEGD0<0>	D12
F	SSEGHex<5>	A14	Digit 0 – Display 1	SSEGD0<1>	C12
G	SSEGHex<6>	B12	Digit 0 – Display 2	SSEGD0<2>	F12
DP	SSEGHex<7>	A12	Digit 0 – Display 3	SSEGD0<3>	E12
COL	SSEGHex<8>	C6	Digit 1 – Display 0	SSEGD1<0>	H3
-	-	-	Digit 1 – Display 1	SSEGD1<1>	L7
-	-	-	Digit 1 – Display 2	SSEGD1<2>	K6
-	-	-	Digit 1 – Display 3	SSEGD1<3>	G3
-	-	-	Digit 2 – Display 0	SSEGD2<0>	A15
-	-	-	Digit 2 – Display 1	SSEGD2<1>	C15
-	-	-	Digit 2 – Display 2	SSEGD2<2>	A16
-	-	-	Digit 2 – Display 3	SSEGD2<3>	B16
-	-	-	Digit 3 – Display 0	SSEGD3<0>	H1
-	-	-	Digit 3 – Display 1	SSEGD3<1>	J7
-	-	-	Digit 3 – Display 2	SSEGD3<2>	J6
-	-	-	Digit 3 – Display 3	SSEGD3<3>	F2

2.6 More Information

A specification pack containing more information on the hardware used will be released at a later date as a downloadable archive (.zip) file located on the session page for each rig.

3 Rig Control Software

Once you have been allocated to a rig – you should be presented with the rig control software. This is an HTML5 web interface that allows you to upload a bitstream, toggle virtual buttons (and input lines) and view a video feed of the rig.

Note: If you have issues using the Rig Control Software interface, please update to the latest version of your browser of choice. The latest version of each major browser is recommended (e.g. Firefox, Chrome, Internet Explorer).

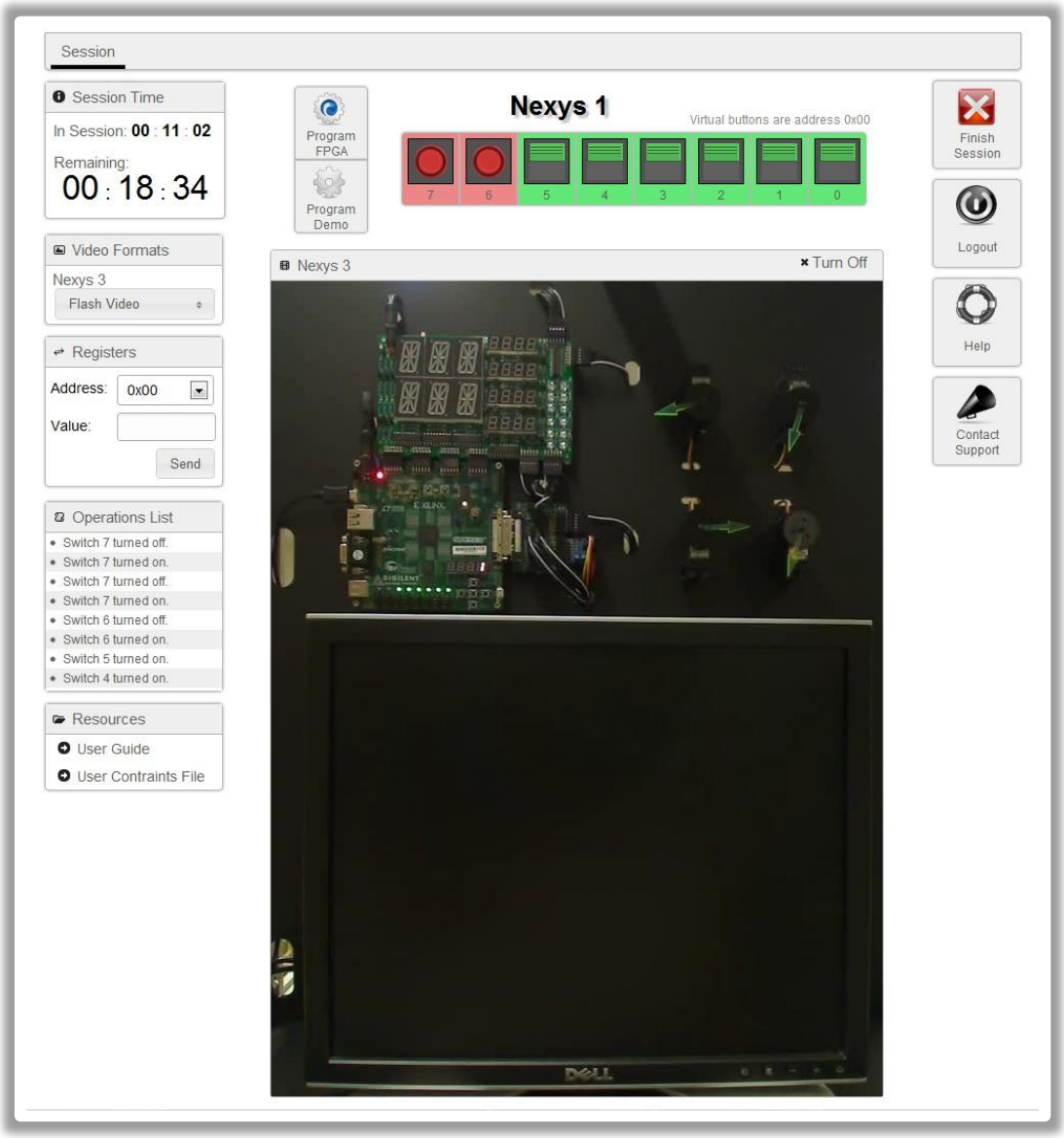


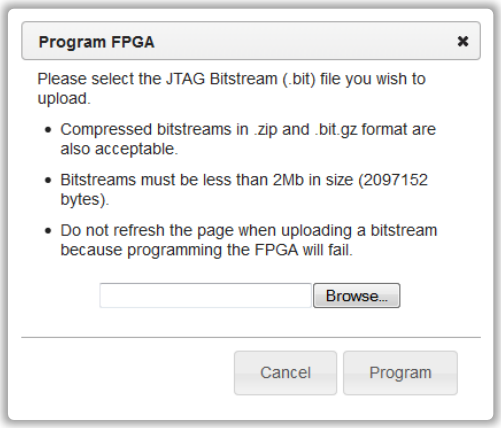
Figure 3: *Rig Control Software with the rig in its default state.*

3.1 Programming the FPGA

To program the FPGA, you will need a bistream file (.bit) generated using *Xilinx ISE Design Suite 13.2* or higher. See **Section 5 Programming in Xilinx for Remote Labs** for more information on how to do this.

Once you have your bitstream file – simply click the "**Program FPGA**" button at the top left of the page.

This will then bring up a file selection dialogue box. Click "**Browse**" and navigate to the location of your bistream file (typically in the same folder as your Xilinx project). Select the file, click **"OK"** and ensure that the file path is displayed in the dialogue box.

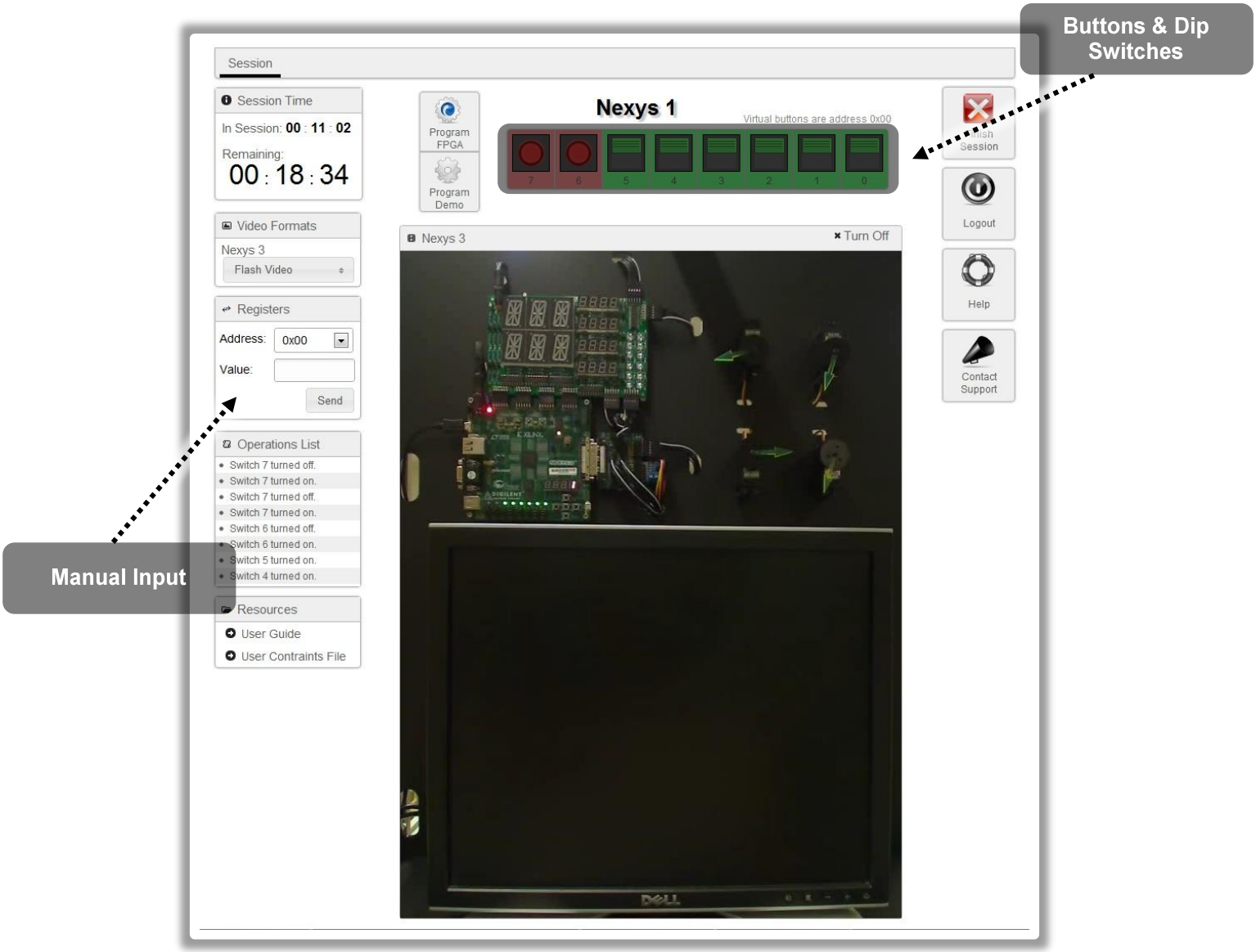


When ready, click **"Program"**. If all goes well, your bistream should now be uploaded and running on the Nexys FPGA rig. The status of the programming operation will be displayed on the left hand side under the "Operations List" heading.

If it didn't work or if unexpected things are happening – refer to **Section 7 FAQ & Troubleshooting**.

3.2 Virtual Inputs

The Rig Control Software allows users to interact with the Nexys FPGA board through virtual inputs. These virtual inputs have a specific protocol that must be implemented in the design that is uploaded to the FPGA. Example VHDL code is provided in **Section 4 Virtual Inputs**.



There are two methods of virtual input for the Nexys FPGA board:

- Buttons and Dip Switches – These virtual buttons flip individual bits on the register address 0x00. They simulate the function of a push button (holds a high value until depressed) or a dip switch (maintains either a high or a low signal).
- Manual Input – The manual input allows you to write directly to a register in either hex (0xFF), decimal (255) or binary (0b11111111).

For these virtual inputs to function properly, the FPGA design must have the virtual input protocol designed into it in VHDL (see **Section 4 Virtual Inputs** for info on how to do this).

4 Virtual Inputs

To interact with the Remote Labs FPGA boards, you will have to use the virtual inputs that are provided through the web interface. These inputs are treated as *registers* created on the FPGA and follow a transmission protocol described below. The Rig Control Software can handle up to 16 8-bit registers.

4.1 How does they work?

The virtual interface to the FPGA board consists of 12 signal lines:

- *EppAstb* – Address strobe: Indicates that an address packet was written to the data bus.
- *EppDstb* – Data strobe: Indicates that a data packet was written to the data bus.
- *EppWr* – Port Write Signal: Indicates that a packet is being written.
- *EppDB* – 8-port data bus: The actual data being transferred in 8-bit segments.
- *EppWait* – Wait Signal: Indicates that the FPGA is waiting for another transfer.

A write process follows these steps:

1. The server writes an address to the FPGA board. To do this, it sets *EppWr* and *EppAstb* to high and sets the data port with the address.
2. The FPGA should be designed to recognise the address write and store the value.
3. The server will then write the data packet to the FPGA board. It does this by setting EppWr and EppDstb to high and then sets the data port with the address.
4. The FPGA should then write that data to the address that was previously stored.

4.2 Explaining the sample code

This section will explain the sample code provided for the FPGA to use with the virtual inputs.

4.2.1 Address Receive

```
process (EppAstb)
      begin
            if rising_edge(EppAstb) then  -- Astb end edge
                  if EppWr = '0' then -- EPP Addr write cycle
                        regEppAdr <= EppDB; -- Epp Address register update
                  end if;
            end if;
      end process;
```

This section of code handles an incoming address packet. On the end of the address transmission (indicated by a rising edge in *EppAstb*), the FPGA will then store the address value in the register *regEppAdr* for use later.

The addresses are a simple binary number (such as 0b00000001 or 0x01), with the Rig Control Interface handling up to 16 addresses.

4.2.2 Data Receive

```
process (EppDstb)
      begin
            if rising_edge(EppDstb) then
                  if EppWr = '0' then
                        if regEppAdr = X"00" then
                              reg0 <= EppDB;
                        elsif regEppAdr = X"01" then
                              reg1 <= EppDB;
                        end if;
                  end if;
            end if;
      end process;
```

This section of code handles the data packet and sorts it into either register 0x00 or 0x01 (which are created within the FPGA as signals). On the end of a data transmission (indicated by the rising edge in *EppDstb*) the FPGA will write data to the register stored in the address register *regEppAdr*.

To use the data, just read it from the appropriate register like so:

```
Led <= reg0;
```

The code can also easily be extended to handle the 16 registers that the Rig Control Software can handle. This is done by extending the `if` statement (or re-writing it into a case selection) and continuing the addresses down to 0x15.

4.3 Sample Code

```
-- Wait signal

 EppWait <= '1' when EppAstb = '0' or EppDstb = '0' else '0';


-- EPP Address receive

process (EppAstb)

      begin

            if rising_edge(EppAstb) then  -- Astb end edge

                  if EppWr = '0' then -- EPP Addr write cycle

                        regEppAdr <= EppDB; -- Epp Address register update

                  end if;

            end if;

      end process;


-- EPP Data receive

process (EppDstb)

      begin

            if rising_edge(EppDstb) then

                  if EppWr = '0' then

                        if regEppAdr = X"00" then

                              reg0 <= EppDB;

                        elsif regEppAdr = X"01" then

                              reg1 <= EppDB;

                        end if;

                  end if;

            end if;

      end process;
```

5 Programming in Xilinx for Remote Labs

5.1 Overview

To program the Nexys FPGA board, you will need to create a project in the *Xilinx ISE Design Suite* and output a bitstream file (.bit) for uploading to the FPGA board. The *Xilinx ISE Design Suite* is a software tool produced by Xilinx that enables the synthesis and analysis of HDL (Hardware Description Language) designs. It supports both VHDL and Verilog HDL languages, but for the purposes of this tutorial, VHDL will be used.

The requirements for programming in Xilinx for the UTS Remote Labs FPGA Rig are:

- *Xilinx ISE Design Suite 13.2* or higher (This should be installed in the FEIT computer labs - otherwise it is available for download at http://www.xilinx.com/products/design-tools/ise-design-suite/ise-webpack.htm).
- A UTS Remote Labs account with access to the FPGA2

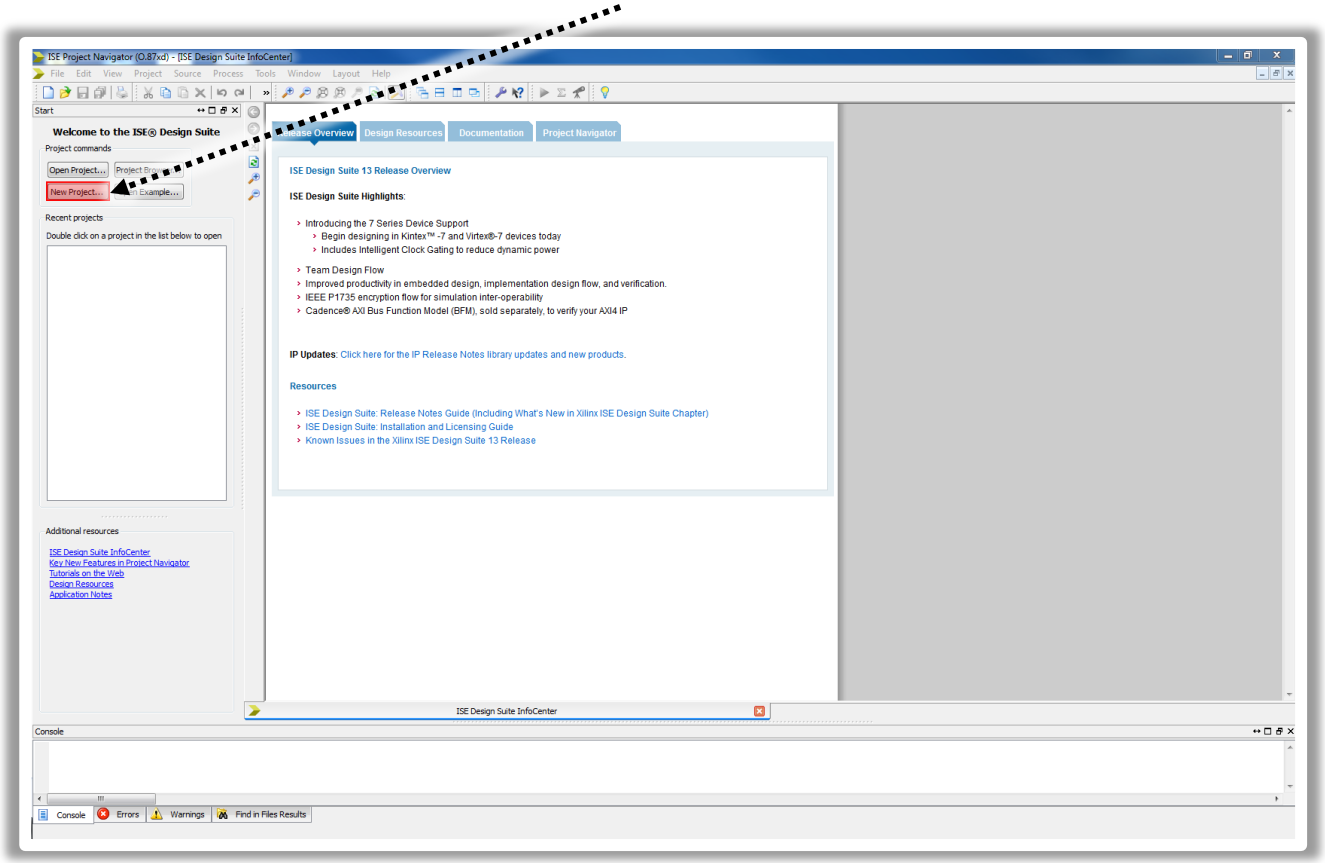
5.2 Programming in Xilinx

The following sections will provide a quick guide in using Xilinx to program in VHDL and output a bitstream file for upload to the FPGA. The program will simply toggle an LED on the board. It will introduce basic concepts of programming in VHDL and also how to generate a bitstream for upload to Remote Labs.

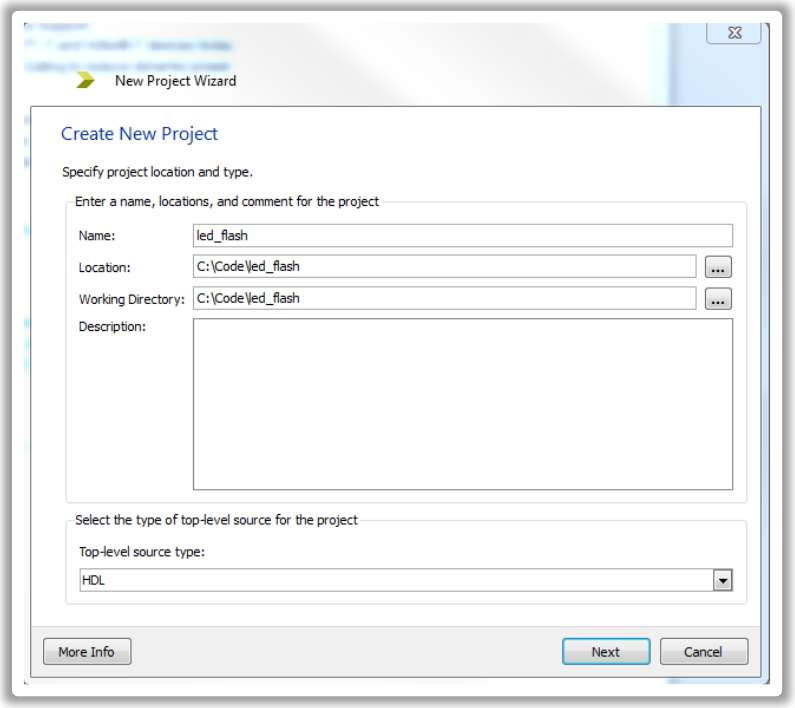
5.2.1 Creating a new Project

Creating a new project in Xilinx involves specifying the hardware that will be targeted by the project.

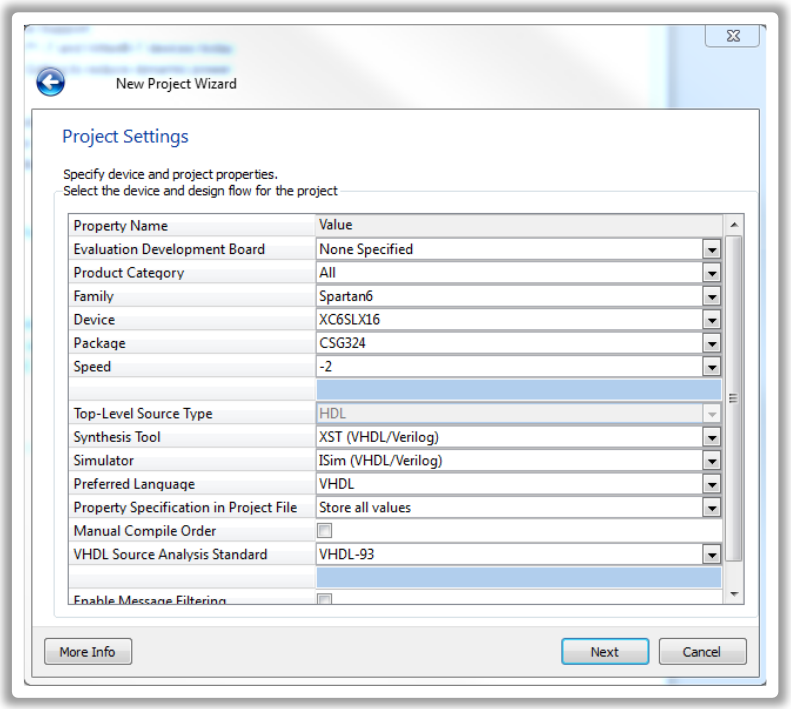
1. Open **Xilinx ISE Design Suite** and click **New Project** (or alternatively, go to **File→New Project**).



2. Enter a name for your project (in this case, we'll call it *led_flash*) and find a location to save it to. Leave the **Top-level source type** as *HDL*. Click next after you have completed this.



3. This screen will prompt you to specify the device and project properties. The board you will be using has a Spartan6 XC6SLX16 FPGA. Enter the following values into the fields:



Once completed, click through by pressing Next.

4. Review the Project Summary, ensure all the values are correct and then press Finish.

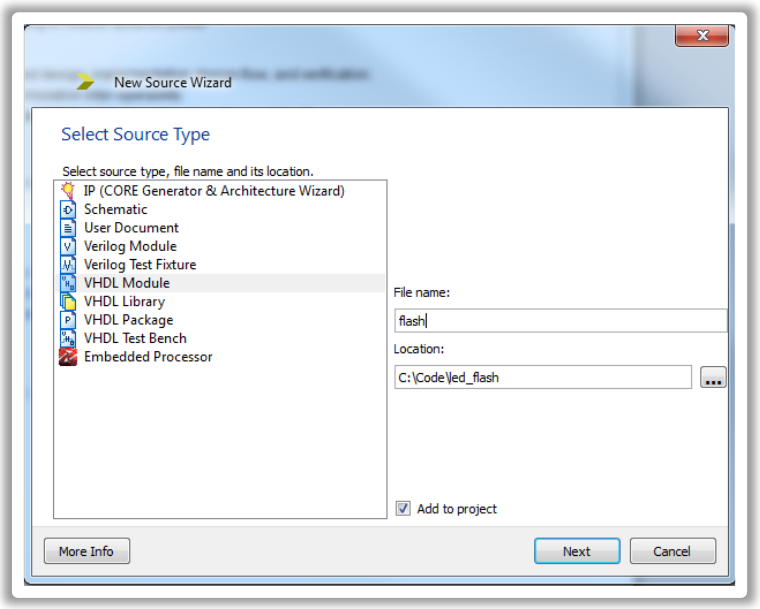
You have just created a Xilinx Project. This will enable you to add or create source files and synthesise them for the specific FPGAs used in Remote Labs. If you get any of the settings wrong, you can easily change them by going to **Project→Design Properties**.

5.2.2 Adding new source file

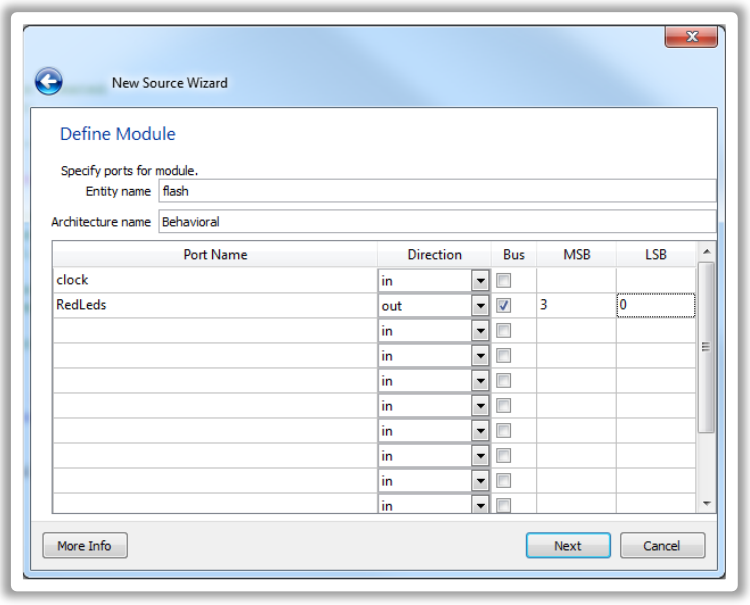
This next part of the guide will show you how to create a new VHDL source file for the project created above. A VHDL file contains an *entity*, which is a high level "component". An entity is made up of two parts: an Entity Declaration which defines the entity and its input/outputs, and an architecture which defines the sequential and combinational logic.

To add a source file to a Xilinx Project:

1. Click on **Project→New Source** in the menu bar.



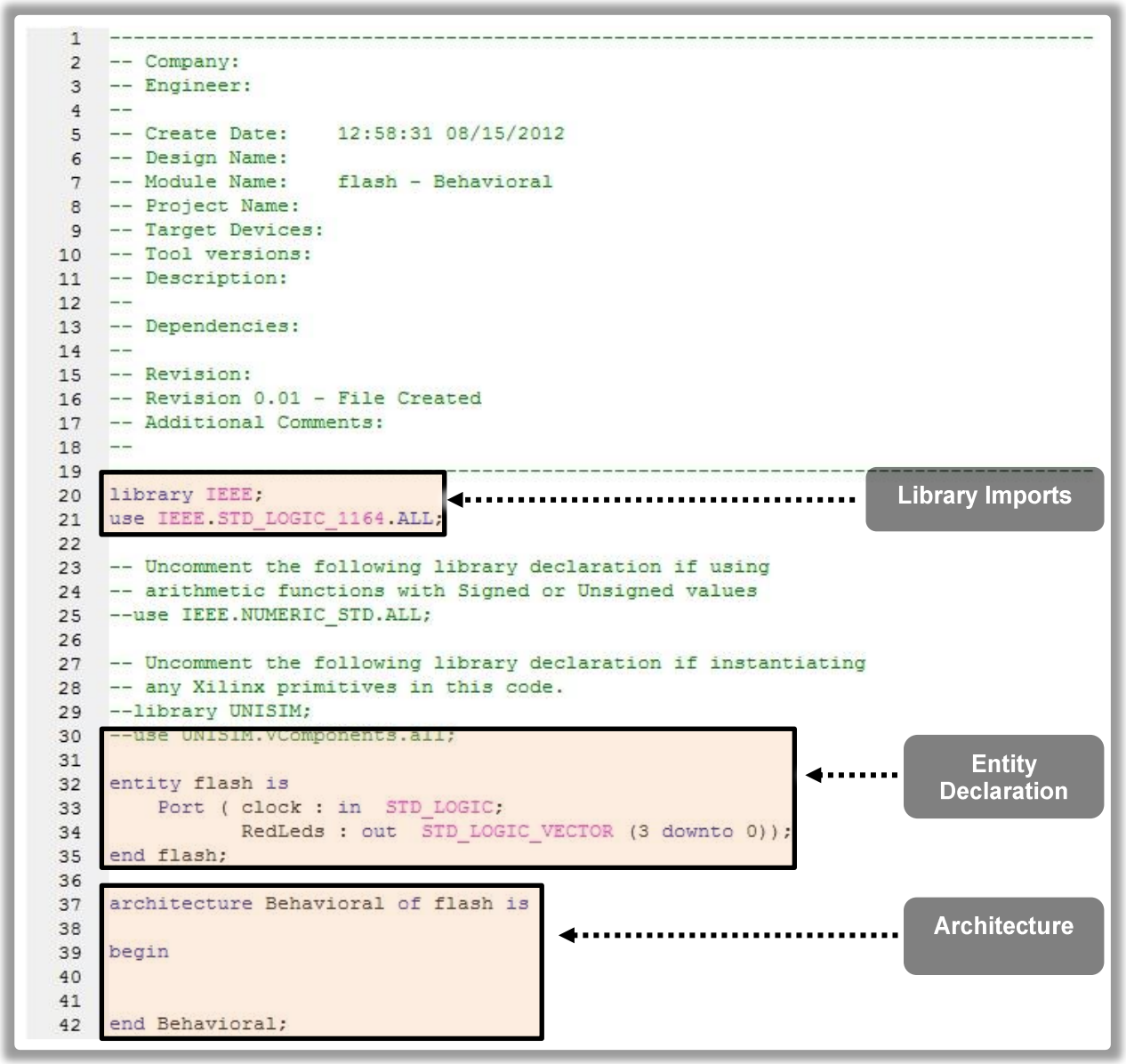
2. Select *VHDL Module* as the Source Type and name it *flash*. Click next once done.
3. Next you will be shown the Define Module window. Here you will be able to name the entity and define ports (inputs & outputs) for the entity. Xilinx will then convert these values into a VHDL entity declaration. In the Define Module window, add two ports as shown below:



Here we have defined two ports, one input for the clock to drive the logic and an output to turn on an LED. Click next once you have added these.

4. View the Summary and click Finish.

Xilinx will then add the source file to the project. You can view it by double clicking on the new source file in the **Hierarchy** on the top left.




```
1  ----------------------------------------------------------------------------------
2  -- Company:
3  -- Engineer:
4  --
5  -- Create Date:    12:58:31 08/15/2012
6  -- Design Name:
7  -- Module Name:    flash - Behavioral
8  -- Project Name:
9  -- Target Devices:
10 -- Tool versions:
11 -- Description:
12 --
13 -- Dependencies:
14 --
15 -- Revision:
16 -- Revision 0.01 - File Created
17 -- Additional Comments:
18 --
19 ----------------------------------------------------------------------------------
20 library IEEE;
21 use IEEE.STD_LOGIC_1164.ALL;
22
23 -- Uncomment the following library declaration if using
24 -- arithmetic functions with Signed or Unsigned values
25 --use IEEE.NUMERIC_STD.ALL;
26
27 -- Uncomment the following library declaration if instantiating
28 -- any Xilinx primitives in this code.
29 --library UNISIM;
30 --use UNISIM.VComponents.all;
31
32 entity flash is
33     Port ( clock : in  STD_LOGIC;
34            RedLeds : out  STD_LOGIC_VECTOR (3 downto 0));
35 end flash;
36
37 architecture Behavioral of flash is
38
39 begin
40
41
42 end Behavioral;
```

You'll notice that the Entity Declaration already has the ports that we entered into the Define Module window declared.

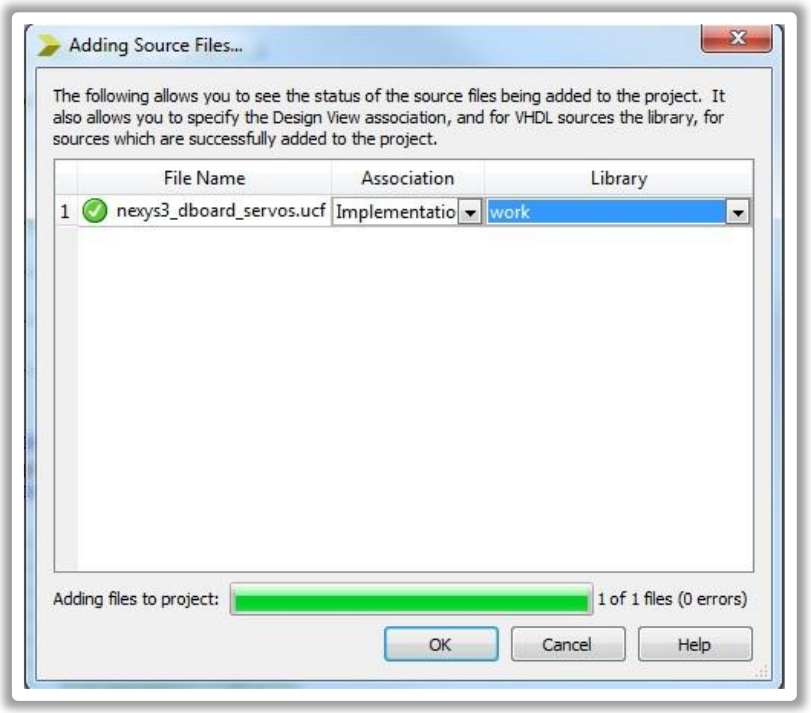
Before you begin writing any VHDL code, you will need to add an Implementation Constraints file (.ucf) to define the pins in the hardware of the Nexys board.

5.2.3 Adding a constraints file

In this step we'll be adding a User Constraints File (.ucf) to the design. UCF files are ASCII files specifying constraints on the logical design. These constraints affect how the logical design is implemented in the target device.

To add the constraint file:

1. Download the .ucf file from the ***Rig Control Software*** page.
2. In Xilinx ISE, go to **Project→Add Source**, add the .ucf file and then click OK.



3. You'll notice it has been added to the **Hierachy** on the top left, double click the constraints file to open it.

Take a few moments to look through the constraints file. Each of the components that can be operated by the FPGA have been assigned pins here, from the various LEDs and SSDs to the servo motors. You'll notice most of the pins are commented out using #. Xilinx will **not** synthesize unless all the declared pins are used in the VHDL design. **Uncomment ONLY the pins you will be using in your design.**

```
Net "RedLed<0>" LOC = C7 | IOSTANDARD=LVCMOS33;
```

To use one of these pins, you must declare it as a **Port** in the **Entity Declaration** of your VHDL source. The name must correspond to the Net value (e.g "clk"). If the Net value indicates it is part of a bus (e.g RedLed<2> is the third LSB of the RedLed vector), the Port should be declared a vector of the same size.

5.2.4 Creating a sample bitstream

In this guide, we will create a sample program in VHDL, synthesise it and generate a programming file for upload to the Remote Lab.

1. Open the VHDL source file you created before (flash.vhd).
2. In the library imports, add these lines of code:

```
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
```

   This will add various math functions and number functions to the code.

3. Under the start of the Architecture, add two signals *clkdiv* and *red_led_buffer* to the code:

```
signal clkdiv : std_logic_vector(30 downto 0)  :=
"0000000000000000000000000000000";
signal red_led_buffer   : std_logic_vector(3 downto 0) := "0000";
```

   All signal and variable declarations happen outside of the process.

4. After the `begin` in the architecture, add the following logic:

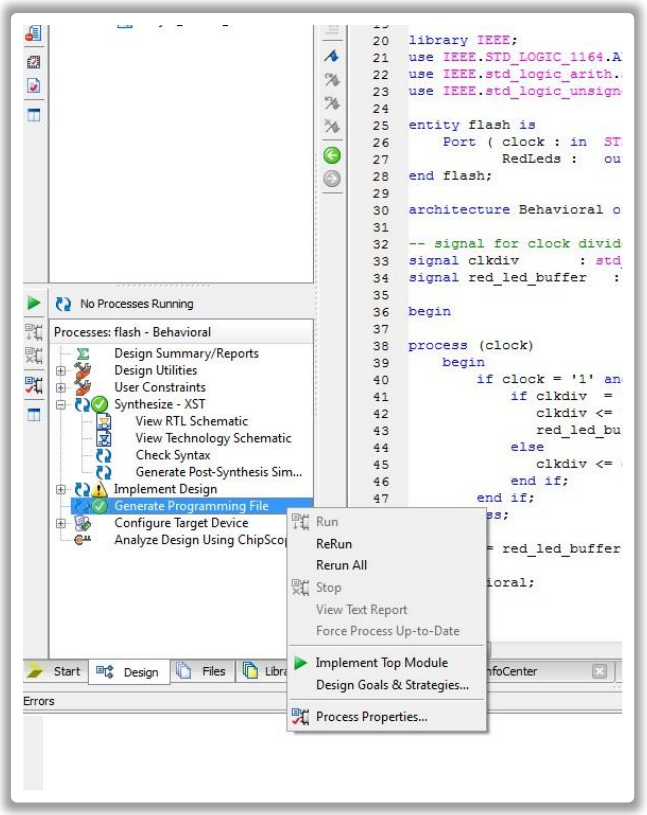
```
process (clock)
    begin
        if clock = '1' and clock'Event then
            if clkdiv  = "11111111111111111111111111" then
               clkdiv <= "00000000000000000000000000";
               red_led_buffer <= not red_led_buffer;
            else
               clkdiv <= clkdiv + 1;
            end if;
        end if;
end process;

RedLeds <= red_led_buffer;
```

   This code will begin a process that uses a clock divider (the normal clock speed is 100MHz) to toggle the red LEDs on the board.

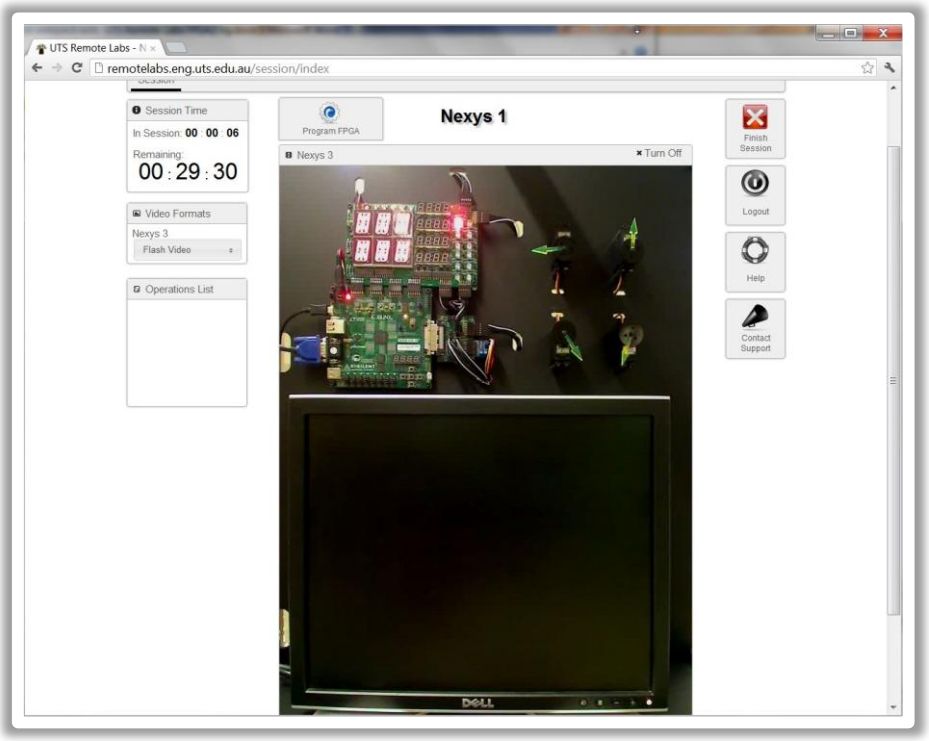
```
20  library IEEE;
21  use IEEE.STD_LOGIC_1164.ALL;
22  use IEEE.std_logic_arith.all;
23  use IEEE.std_logic_unsigned.all;
24
25  entity flash is
26      Port ( clock : in  STD_LOGIC;
27             RedLeds :   out  STD_LOGIC_VECTOR (3 downto 0));
28  end flash;
29
30  architecture Behavioral of flash is
31
32  -- signal for clock divider to divide board clock to usable frequency
33  signal clkdiv       : std_logic_vector(25 downto 0)  := "00000000000000000000000000";
34  signal red_led_buffer   : std_logic_vector(3 downto 0) := "0000";
35
36  begin
37
38  process (clock)
39      begin
40          if clock = '1' and clock'Event then
41              if clkdiv  = "11111111111111111111111111" then
42                 clkdiv <= "00000000000000000000000000";
43                 red_led_buffer <= not red_led_buffer;
44              else
45                 clkdiv <= clkdiv + 1;
46              end if;
47          end if;
48  end process;
49
50  RedLeds <= red_led_buffer;
51
52  end Behavioral;
```

5. Once the code has been entered, it's time to **Synthesise** and **Generate a Programming File**. While in the VHDL source file, look to the left of the screen and right click **Generating Programming File** and then click on **Rerun All**.

   It will prompt you to save the file if you haven't already and then will proceed to convert your code into a hardware schematic and produce a programming file for it.



6. After it has completed, open Windows Explorer and navigate to the directory where you saved your Project. Inside there should be a **.bit** file (in our case, **flash.bit**). This is the bitstream file that you will need to upload to the remote FPGA.
7. Open an internet browser and go to http://remotelabs.uts.edu.au. Choose the FEIT option and then enter your login details. Click on the Nexys tab and queue up access for a rig.
8. Once you have access to a rig, you will see this screen:



Click the **Program FPGA button** at the top and navigate to your .bit file. Once you've found it, click Program. The bit file will then upload and be programmed onto the FPGA and your program should begin to run. If all went right, you should see the red LEDs flashing on and off.

5.2.5 Code

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity flash is
    Port ( clock : in  STD_LOGIC;
           RedLed :   out  STD_LOGIC_VECTOR (3 downto 0));
end flash;

architecture Behavioral of flash is

signal clkdiv       : std_logic_vector(25 downto 0)  :=
"00000000000000000000000000";
signal red_led_buffer   : std_logic_vector(3 downto 0) := "0000";

begin

process (clock)
    begin
        if clock = '1' and clock'Event then
            if clkdiv  = "11111111111111111111111111" then
               clkdiv <= "00000000000000000000000000";
                              red_led_buffer <= not red_led_buffer;
            else
               clkdiv <= clkdiv + 1;
            end if;
        end if;
end process;

RedLed <= red_led_buffer;

end Behavioral;
```

6 UCF (Universal Communications Format) File

6.1 Inputs

6.1.1 Clock Signal

```
Net "clk" LOC=V10 | IOSTANDARD=LVCMOS33;
Net "clk" TNM_NET = sys_clk_pin;
TIMESPEC TS_sys_clk_pin = PERIOD sys_clk_pin 100000 kHz;
```

6.1.2 Physical Switches

```
Net "switches<0>" LOC=T10  |  IOSTANDARD=LVCMOS33;
Net "switches<1>" LOC=T9   |  IOSTANDARD=LVCMOS33;
Net "switches<2>" LOC=V9   |  IOSTANDARD=LVCMOS33;
Net "switches<3>" LOC=M8   |  IOSTANDARD=LVCMOS33;
Net "switches<4>" LOC=N8   |  IOSTANDARD=LVCMOS33;
Net "switches<5>" LOC=U8   |  IOSTANDARD=LVCMOS33;
Net "switches<6>" LOC=V8   |  IOSTANDARD=LVCMOS33;
Net "switches<7>" LOC=T5   |  IOSTANDARD=LVCMOS33;
```

6.1.3 Virtual I/O

```
Net "EppAstb"   LOC = H1 | IOSTANDARD = LVCMOS33;
Net "EppDstb"   LOC = K4 | IOSTANDARD = LVCMOS33;
Net "EppWait"   LOC = C2 | IOSTANDARD = LVCMOS33;
Net "EppDB<0>"  LOC = E1 | IOSTANDARD = LVCMOS33;
Net "EppDB<1>"  LOC = F4 | IOSTANDARD = LVCMOS33;
Net "EppDB<2>"  LOC = F3 | IOSTANDARD = LVCMOS33;
Net "EppDB<3>"  LOC = D2 | IOSTANDARD = LVCMOS33;
Net "EppDB<4>"  LOC = D1 | IOSTANDARD = LVCMOS33;
Net "EppDB<5>"  LOC = H7 | IOSTANDARD = LVCMOS33
Net "EppDB<6>"  LOC = G6 | IOSTANDARD = LVCMOS33;
Net "EppDB<7>"  LOC = E4 | IOSTANDARD = LVCMOS33;
Net "EppWr"    LOC = H2 | IOSTANDARD = LVCMOS33;
```

6.2 Outputs

6.2.1 LEDs

6.2.1.1 Nexys3 Board LEDs

```
Net "Led<0>" LOC = U16 | IOSTANDARD=LVCMOS33;
Net "Led<1>" LOC = V16 | IOSTANDARD=LVCMOS33;
Net "Led<2>" LOC = U15 | IOSTANDARD=LVCMOS33;
Net "Led<3>" LOC = V15 | IOSTANDARD=LVCMOS33;
Net "Led<4>" LOC = M11 | IOSTANDARD=LVCMOS33;
Net "Led<5>" LOC = N11 | IOSTANDARD=LVCMOS33;
Net "Led<6>" LOC = R11 | IOSTANDARD=LVCMOS33;
Net "Led<7>" LOC = T11 | IOSTANDARD=LVCMOS33;
```

6.2.1.2 Display Board LEDs

6.2.1.2.1 Red LEDs

```
Net "RedLed<0>" LOC = C7  | IOSTANDARD=LVCMOS33;
Net "RedLed<1>" LOC = C14 | IOSTANDARD=LVCMOS33;
Net "RedLed<2>" LOC = A6  | IOSTANDARD=LVCMOS33;
Net "RedLed<3>" LOC = C5  | IOSTANDARD=LVCMOS33;
```

6.2.1.2.2 Amber LEDs

```
Net "AmberLed<0>" LOC = A7  | IOSTANDARD=LVCMOS33;
Net "AmberLed<1>" LOC = F13 | IOSTANDARD=LVCMOS33;
Net "AmberLed<2>" LOC = D11 | IOSTANDARD=LVCMOS33;
Net "AmberLed<3>" LOC = A5  | IOSTANDARD=LVCMOS33;
```

6.2.1.2.3 Green LEDs

```
Net "GreenLed<0>" LOC = B6  | IOSTANDARD=LVCMOS33;
Net "GreenLed<1>" LOC = E13 | IOSTANDARD=LVCMOS33;
Net "GreenLed<2>" LOC = D6  | IOSTANDARD=LVCMOS33;
Net "GreenLed<3>" LOC = D14 | IOSTANDARD=LVCMOS33;
```

6.2.2 Alpha Numeric Displays

6.2.2.1 Display Board Alpha Numeric Displays

6.2.2.1.1 Display Selection

```
Net "Alpha<0>" LOC = G11 | IOSTANDARD=LVCMOS33;
Net "Alpha<1>" LOC = F10 | IOSTANDARD=LVCMOS33;
Net "Alpha<2>" LOC = F11 | IOSTANDARD=LVCMOS33;
Net "Alpha<3>" LOC = E11 | IOSTANDARD=LVCMOS33;
Net "Alpha<4>" LOC = A4 | IOSTANDARD=LVCMOS33;
Net "Alpha<5>" LOC = B4 | IOSTANDARD=LVCMOS33;
```

6.2.2.1.2 Segment Selection

```
Net "AlphaHex<0>"  LOC = V11 | IOSTANDARD=LVCMOS33; #A1
Net "AlphaHex<1>"  LOC = U11 | IOSTANDARD=LVCMOS33; #A2
Net "AlphaHex<2>"  LOC = N9  | IOSTANDARD=LVCMOS33; #B
Net "AlphaHex<3>"  LOC = M10 | IOSTANDARD=LVCMOS33; #C
Net "AlphaHex<4>"  LOC = P11 | IOSTANDARD=LVCMOS33; #D1
Net "AlphaHex<5>"  LOC = N10 | IOSTANDARD=LVCMOS33; #D2
Net "AlphaHex<6>"  LOC = V12 | IOSTANDARD=LVCMOS33; #E
Net "AlphaHex<7>"  LOC = T12 | IOSTANDARD=LVCMOS33; #F
Net "AlphaHex<8>"  LOC = K5  | IOSTANDARD=LVCMOS33; #G1
Net "AlphaHex<9>"  LOC = K3  | IOSTANDARD=LVCMOS33; #G2
Net "AlphaHex<10>" LOC = J1  | IOSTANDARD=LVCMOS33; #H
Net "AlphaHex<11>" LOC = J3  | IOSTANDARD=LVCMOS33; #J
Net "AlphaHex<12>" LOC = L3  | IOSTANDARD=LVCMOS33; #K
Net "AlphaHex<13>" LOC = L4  | IOSTANDARD=LVCMOS33; #L
Net "AlphaHex<14>" LOC = K1  | IOSTANDARD=LVCMOS33; #M
Net "AlphaHex<15>" LOC = K2  | IOSTANDARD=LVCMOS33; #N
```

6.2.3 Seven Segment Displays

6.2.3.1 Nexys3 Board Seven Segment Displays

6.2.3.1.1 Display Selection

```
Net "SSEG_AN<0>" LOC=N16 | IOSTANDARD=LVCMOS33;
Net "SSEG_AN<1>" LOC=N15 | IOSTANDARD=LVCMOS33;
Net "SSEG_AN<2>" LOC=P18 | IOSTANDARD=LVCMOS33;
Net "SSEG_AN<3>" LOC=P17 | IOSTANDARD=LVCMOS33;
```

6.2.3.1.2 Segment Selection

```
Net "SSEG_CA<0>" LOC=T17 | IOSTANDARD=LVCMOS33; #A
Net "SSEG_CA<1>" LOC=T18 | IOSTANDARD=LVCMOS33; #B
Net "SSEG_CA<2>" LOC=U17 | IOSTANDARD=LVCMOS33; #C
Net "SSEG_CA<3>" LOC=U18 | IOSTANDARD=LVCMOS33; #D
Net "SSEG_CA<4>" LOC=M14 | IOSTANDARD=LVCMOS33; #E
Net "SSEG_CA<5>" LOC=N14 | IOSTANDARD=LVCMOS33; #F
Net "SSEG_CA<6>" LOC=L14 | IOSTANDARD=LVCMOS33; #G
Net "SSEG_CA<7>" LOC=M13 | IOSTANDARD=LVCMOS33; #DP
```

6.2.3.2 Display Board Seven Segment Displays

6.2.3.2.1 Display Selection

```
#----SSEGD0 (Digit 0) on Display <n>
Net "SSEGD0<0>" LOC = D12 | IOSTANDARD=LVCMOS33;
Net "SSEGD0<1>" LOC = C12 | IOSTANDARD=LVCMOS33;
Net "SSEGD0<2>" LOC = F12 | IOSTANDARD=LVCMOS33;
Net "SSEGD0<3>" LOC = E12 | IOSTANDARD=LVCMOS33;

#----SSDG1 (Digit 1) on Display <n>
Net "SSEGD1<0>" LOC = H3 | IOSTANDARD=LVCMOS33;
Net "SSEGD1<1>" LOC = L7 | IOSTANDARD=LVCMOS33;
Net "SSEGD1<2>" LOC = K6 | IOSTANDARD=LVCMOS33;
Net "SSEGD1<3>" LOC = G3 | IOSTANDARD=LVCMOS33;

#----SSDG2 (Digit 2) on Display <n>
Net "SSEGD2<0>" LOC = A15 | IOSTANDARD=LVCMOS33;
Net "SSEGD2<1>" LOC = C15 | IOSTANDARD=LVCMOS33;
Net "SSEGD2<2>" LOC = A16 | IOSTANDARD=LVCMOS33;
Net "SSEGD2<3>" LOC = B16 | IOSTANDARD=LVCMOS33;

#----SSDG3 (Digit 3) on Display <n>
Net "SSEGD3<0>" LOC = G1 | IOSTANDARD=LVCMOS33;
Net "SSEGD3<1>" LOC = J7 | IOSTANDARD=LVCMOS33;
Net "SSEGD3<2>" LOC = J6 | IOSTANDARD=LVCMOS33;
Net "SSEGD3<3>" LOC = F2 | IOSTANDARD=LVCMOS33;

#----SSCOL (Colon) on Display <n>
Net "SSEGCL<0>" LOC = A2 | IOSTANDARD=LVCMOS33;
Net "SSEGCL<1>" LOC = B2 | IOSTANDARD=LVCMOS33;
Net "SSEGCL<2>" LOC = A3 | IOSTANDARD=LVCMOS33;
Net "SSEGCL<3>" LOC = B3 | IOSTANDARD=LVCMOS33;
```

6.2.3.2.2 Segment Selection

```
Net "SSEGHex<0>" LOC = C13  |  IOSTANDARD=LVCMOS33; #A
Net "SSEGHex<1>" LOC = A13  |  IOSTANDARD=LVCMOS33; #B
Net "SSEGHex<2>" LOC = B11  |  IOSTANDARD=LVCMOS33; #C
Net "SSEGHex<3>" LOC = A11  |  IOSTANDARD=LVCMOS33; #D
Net "SSEGHex<4>" LOC = B14  |  IOSTANDARD=LVCMOS33; #E
Net "SSEGHex<5>" LOC = A14  |  IOSTANDARD=LVCMOS33; #F
Net "SSEGHex<6>" LOC = B12  |  IOSTANDARD=LVCMOS33; #G
Net "SSEGHex<7>" LOC = A12  |  IOSTANDARD=LVCMOS33; #DP
Net "SSEGHex<8>" LOC = C6   |  IOSTANDARD=LVCMOS33; #COL
```

6.2.4 Servo Motors

6.2.4.1 Continuous Rotation Servos

```
Net "servo<0>" LOC = C8 | IOSTANDARD=LVCMOS33;
Net "servo<1>" LOC = D8 | IOSTANDARD=LVCMOS33;
```

6.2.4.2 180 Degree Positioning Servos

```
Net "servo<2>" LOC = F9 | IOSTANDARD=LVCMOS33;
Net "servo<3>" LOC = G9 | IOSTANDARD=LVCMOS33;
```

6.2.5 VGA Output

```
#----Green
#NET "VGA_GREEN<0>" LOC=P8 | IOSTANDARD=LVCMOS33;
#NET "VGA_GREEN<1>" LOC=T6 | IOSTANDARD=LVCMOS33;
#NET "VGA_GREEN<2>" LOC=V6 | IOSTANDARD=LVCMOS33;

#----Red
#NET "VGA_RED<0>" LOC=U7 | IOSTANDARD=LVCMOS33;
#NET "VGA_RED<1>" LOC=V7 | IOSTANDARD=LVCMOS33;
#NET "VGA_RED<2>" LOC=N7 | IOSTANDARD=LVCMOS33;

#----Blue
#NET "VGA_BLUE<0>" LOC=R7 | IOSTANDARD=LVCMOS33;
#NET "VGA_BLUE<1>" LOC=T7 | IOSTANDARD=LVCMOS33;

#----Sync
#NET "VGA_VSYNC" LOC=N6 | IOSTANDARD=LVCMOS33;
#NET "VGA_HSYNC" LOC=P7 | IOSTANDARD=LVCMOS33;
```

6.2.6 Board Memory

```
#---Onboard Cellular RAM, Numonyx StrataFlash and Numonyx Quad Flash
#Net "MemOE"     LOC = L18 | IOSTANDARD=LVCMOS33;
#Net "MemWR"     LOC = M16 | IOSTANDARD=LVCMOS33;
#Net "MemAdv"    LOC = H18 | IOSTANDARD=LVCMOS33;
#Net "MemWait"   LOC = V4  | IOSTANDARD=LVCMOS33;
#Net "MemClk"    LOC = R10 | IOSTANDARD=LVCMOS33;

#Net "RamCS"     LOC = L15 | IOSTANDARD=LVCMOS33;
#Net "RamCRE"    LOC = M18 | IOSTANDARD=LVCMOS33;
#Net "RamUB"     LOC = K15 | IOSTANDARD=LVCMOS33;
#Net "RamLB"     LOC = K16 | IOSTANDARD=LVCMOS33;

#Net "FlashCS" LOC = L17 | IOSTANDARD=LVCMOS33;
#Net "FlashRp" LOC = T4  | IOSTANDARD=LVCMOS33;

#Net "QuadSpiFlashCS"  LOC = V3  | IOSTANDARD=LVCMOS33;
#Net "QuadSpiFlashSck" LOC = R15 | IOSTANDARD=LVCMOS33;

#Net "MemAdr<1>" LOC = K18  | IOSTANDARD=LVCMOS33;
#Net "MemAdr<2>" LOC = K17  | IOSTANDARD=LVCMOS33;
#Net "MemAdr<3>" LOC = J18  | IOSTANDARD=LVCMOS33;
#Net "MemAdr<4>" LOC = J16  | IOSTANDARD=LVCMOS33;
#Net "MemAdr<5>" LOC = G18  | IOSTANDARD=LVCMOS33;
#Net "MemAdr<6>" LOC = G16  | IOSTANDARD=LVCMOS33;
#Net "MemAdr<7>" LOC = H16  | IOSTANDARD=LVCMOS33;
#Net "MemAdr<8>" LOC = H15  | IOSTANDARD=LVCMOS33;
#Net "MemAdr<9>" LOC = H14  | IOSTANDARD=LVCMOS33;
#Net "MemAdr<10>" LOC = H13 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<11>" LOC = F18 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<12>" LOC = F17 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<13>" LOC = K13 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<14>" LOC = K12 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<15>" LOC = E18 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<16>" LOC = E16 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<17>" LOC = G13 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<18>" LOC = H12 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<19>" LOC = D18 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<20>" LOC = D17 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<21>" LOC = G14 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<22>" LOC = F14 | IOSTANDARD=LVCMOS33;
```

```
#Net "MemAdr<23>" LOC = C18 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<24>" LOC = C17 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<25>" LOC = F16 | IOSTANDARD=LVCMOS33;
#Net "MemAdr<26>" LOC = F15 | IOSTANDARD=LVCMOS33;

#Net "QuadSpiFlashDB<0>" LOC = T13 | IOSTANDARD=LVCMOS33;
#Net "MemDB<0>"  LOC = R13 | IOSTANDARD=LVCMOS33;
#Net "MemDB<1>"  LOC = T14 | IOSTANDARD=LVCMOS33;
#Net "MemDB<2>"  LOC = V14 | IOSTANDARD=LVCMOS33;
#Net "MemDB<3>"  LOC = U5  | IOSTANDARD=LVCMOS33;
#Net "MemDB<4>"  LOC = V5  | IOSTANDARD=LVCMOS33;
#Net "MemDB<5>"  LOC = R3  | IOSTANDARD=LVCMOS33;
#Net "MemDB<6>"  LOC = T3  | IOSTANDARD=LVCMOS33;
#Net "MemDB<7>"  LOC = R5  | IOSTANDARD=LVCMOS33;
#Net "MemDB<8>"  LOC = N5  | IOSTANDARD=LVCMOS33;
#Net "MemDB<9>"  LOC = P6  | IOSTANDARD=LVCMOS33;
#Net "MemDB<10>" LOC = P12 | IOSTANDARD=LVCMOS33;
#Net "MemDB<11>" LOC = U13 | IOSTANDARD=LVCMOS33;
#Net "MemDB<12>" LOC = V13 | IOSTANDARD=LVCMOS33;
#Net "MemDB<13>" LOC = U10 | IOSTANDARD=LVCMOS33;
#Net "MemDB<14>" LOC = R8  | IOSTANDARD=LVCMOS33;
#Net "MemDB<15>" LOC = T8  | IOSTANDARD=LVCMOS33;
```

7 FAQ & Troubleshooting

7.1 My bistream upload failed?

Ensure that your bistream file meets the requirements. These are:

- The file must be upload as a raw bitstream file (.bit) or archived using .zip or .bit.gz formats.
- The file size must be less than 2MiB (2,097,152 bytes).

Also please ensure that you do not refresh the page during upload or else programming will fail.

7.2 My bitstream fails to work as expected?

Ensure that your .ucf file is correct. You can check your nets and pins against the version provided here in **Section 6 UCF (Universal Communications Format) File.** The .ucf file is downloadable from the ***Rig Control Software*** page.

7.3 The servo motors do not work as expected?

The servo motors require correct timing of the pulse window in order to work as intended. The FPGA clock-speed is 100MHz, so ensure you use the correct calculations to generate the 20ms pulse window for the servo motors.

7.4 Hardware Limitations

The following hardware limitations apply to the rig – care should be taken to avoid mistaking real phenomena as faults and the limitations should be observed when analysing the output obtained from the rig.

7.4.1 Servo Motor Rotation RPM Consistency

The servos in use are analogue in nature – thus they do not operate in a discrete manner. It is possible for two or more servos, receiving the same signal, to rotate at slightly different RPM. Thus servos that should be “in sync” can appear to be “out of sync”.

Care should be taken when developing a program using the servo motors – they should not be relied upon as accurate rotation/timing devices.

7.4.2 Servo Motor Positioning Consistency

The position indicator arrows used on the servo motors are made up of laser-cut parts that are glued together. Due to this design the glued assemblies may vary in their alignment.

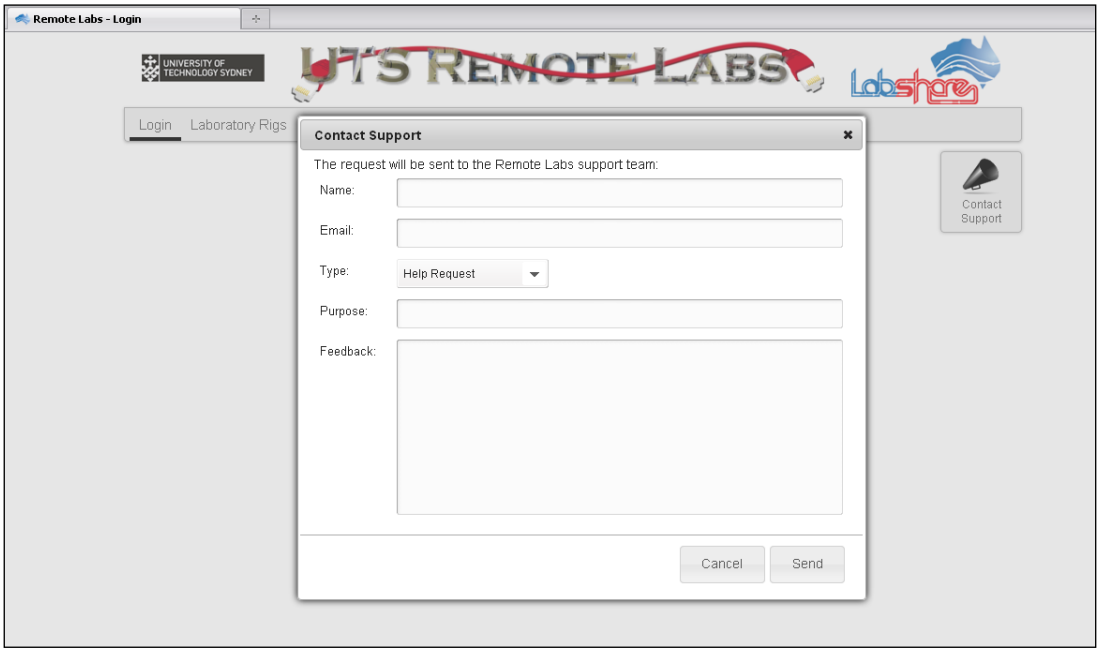
Whilst the best efforts are made to ensure that the position indicator arrows are aligned between servos – they may be out of alignment by a few degrees.

Care should be taken when developing a program using the servo motors – they should not be relied upon as accurate positioning devices.

7.5 Contacting Support

Any questions regarding the nature of assessment tasks should initially be directed to the relevant academic. If the user encounters any difficulties during the course of using the rigs, the **"Contact Support"** button should be used to request assistance and report an incident.

The following popup will appear – please enter your name and a valid email address, followed by a category from the "Type" drop down list.



You may then enter a brief statement regarding the nature of the request in the "Purpose" field. Be sure to enter as detailed a description as possible of the incident in the "Feedback" field.

7.5.1 Providing Feedback

Users are strongly encouraged to leave feedback and comments of their experience with the rigs to help improve the system, as well as any suggestions for additional features to be included in the future. Feedback can be left by clicking the **"Contact Support"** button and selecting "General comment" from the "Type" drop down list.