Nexys3 Extended Peripheral Interface (N3IEF)
User Guide for ECE 544

Revision 1.0

Table of Contents

Revision History

1.0	23-March-2013	Roy Kravitz	First release based on s3eif User Guide

Related Documents

Digilent Nexys3™ Board Reference Manual. Digilent document 502-182 (28-Dec-2012)). Copyright Digilent, Inc.

Digilent PmodCLP™ Parallel LCD Reference Manual. Digilent document 502-142 (28-Apr-2008)). Copyright Digilent, Inc.

Digilent PmodENC™ Reference Manual. Digilent document 502-117 (31-Oct-2011)). Copyright Digilent, Inc.

Samsung KS0066 Data sheet. Copyright Samsung, Inc.

Chapman, Ken, *Initial Design for Spartan-3E Starter Kit (LCD Display Control);* Xilinx Application Note; 16-February-2006.

Chapman, Ken, *Rotary Encoder Interface for Spartan-3E Starter Kit*; Xilinx Application Note; 20-February- 2006

Required Hardware

The proper operation of the Nexys3 Extended Peripheral interface assumes the following hardware configuration:

- Digilent Nexys3™ Board
- Digilient PmodCLP™ connected to the JA and JB expansion headers on the Nexys3. The PmodCLP uses both rows of the JA connector and the bottom row of the JB connector.
- Digilient PmodENC™ rotary encoder connected to either the JC or JD expansion header on the Nexys3. The PmodENC only uses the bottom row of pins on the expansion header.

IMPORTANT: THE XILINX CONSTRAINTS FILE (TYPICALLY FILENAME.UCF) SHOULD INCLUDE SUITABLE PIN CONSTRAINTS FOR THE FOUR PMOD CONNECTORS ON THE NEXYS3 BOARD AND THE TOP LEVEL VERILOG (OR VHDL) MODULE MUST CONNECT THE PORTS FROM THE N3EIF INSTANTIATION TO EXTERNAL PORTS AT THE TOP LEVEL. A REFERENCE DESIGN SHOWING THIS IS INCLUDED IN THE N3EIF RELEASE.

Introduction

The Digilent Nexys3 board provides a prototyping platform and reference design for the Xilinx Spartan 6 family of FPGA's. This low cost board is built around a Spartan 6 XC6LX16-CS324 FPGA, 48Mbytes of external memory (including two non-volatile phase-change memories from Micron), and enough I/O devices and ports to host a wide variety of digital systems. The board has a 4 digit 7-segement display, eight (8) discrete LEDs, eight (8) slide switches and five (5) pushbuttons arranged in a joystick-type configuration. There are connectors for VGA, a USB device (such as a flash drive), Ethernet, and a USB serial connection. The on-board Adept™ high-speed USB2 port provides board power, FPGA programming, and user-data transfers at rates up to 38Mbytes/sec and interfaces well with Xilinx tools such as the SDK and ChipScope. The board also has four (4) Pmod connectors and a Digilent VHDC connector for expansion.

While the Nexys3 board has a good feature set for its price, it is lacking several functions that we use in ECE 544. There is no text display, there are no converters (Digital to Analog and Analog to Digital) and there are no human interface input devices except the pushbuttons and switches. Fortunately, Digilent provides the missing functionality through the Pmod connectors. For ECE 544 we will add key functionality with a PmodCLP (2 line by 16 character LCD) and a PmodENC (Rotary Encoder with pushbutton and slide switch).

The Nexy3 Extended Peripheral Interface N3EIF) is a package of custom IP, drivers, and Picoblaze Assembly code that provides a register-based interface to the peripherals used in ECE 544. Providing this interface package allows you to focus on your applications - we've done some of the hard work for you. Specifically, the N3EIF provides support for the following peripherals:

- Debounced push-buttons and switches, including the push-button shaft on the PmodENC
- Control of the 8-discrete LEDs on the Nexys3 board
- Count and control of the PmodENC rotary encoder
- Support for the PmodCLP LCD display including writing text and numbers to the display, clearing the display, positioning the cursor and shifting the characters on the display.

As a user of the N3EIF you will be able to read the push-buttons and switches, write the LEDs, get count information from the Rotary encoder, and control the LCD display from a Microblaze using C library calls.

Using a Microblaze-based System CPU with the N3EIF

Hardware Interface

Connecting a Microblaze to the N3EIF is not that difficult. N3EIF is implemented as a custom PLB slave peripheral and can be added to your embedded CPU through the Xilinx Platform Studio (XPS).

To add the N3EIF to an existing design select the **IP Catalog** tab and find the custom IP selection (may be called Project Repository or something similar). Select the N3EIF peripheral and drag it to the **System Assembly View** or right click on the N3EIF and select **Add IP**. Once the N3EIF IP is added to your system click on the **Ports** button in the **System Assembly View** and make the inputs and outputs of the xps_s3eif module "external" so they are brought to the top level of the embedded CPU module. You will also need to change the constraints file (typically found in data\system.ucf) to include the pin assignments and I/O levels for those pins. See Appendix A – N3EIF Sample Constraints File (.ucf) .

NOTE: DO NOT ADD THE DB_CLK SIGNAL AS AN EXTERNAL SIGNAL. LEAVE IT UNCONNECTED. IF YOU WISH TO USE DB_CLK YOU WILL NEED TO MAKE THE SIGNAL EXTERNAL AND THEN ADD A LINE TO DATA\SYSTEM.UCF FILE TO ASSIGN A PIN TO IT.

NOTE: IF YOU DO NOT SEE THE N3EIF IN THE IP CATALOG YOU NEED TO MAKE XPS AWARE OF THE LOCATION OF YOUR REPOSITORY. THIS CAN BE DONE BY ADDING THE REPOSITORY LOCATION TO YOUR PROJECT OPTIONS.

Application Program Interface (API)

The Xilinx EDK supports the full GNU tool chain – code development is expected to be done mainly in C or C++. Because of the support for C and C++ the API for the N3EIF is provided as a C source code driver. Several low level function calls are implemented as C macros using #define. The majority of the functions, however, are written in C.

N3EIF Register Offsets

Like most RISC CPU's, the Xilinx Microblaze uses memory-mapped I/O to reach its peripherals. In a memory-mapped I/O system each peripheral consumes a block of memory addresses that are assigned when you configure the system using XPS (Xilinx Platform System). As part of the software development process you export your design to the Xilinx SDK (software development tool chain) and build a board support package for your project. When the board support package is built the SDK generates a number of configuration files, one of them being *xparameters.h*. *xparameters.h* includes

#defines for the base address (beginning of the block of I/O register) and high address for each peripheral, along with a Device ID. These addresses are used by drivers like the N3EIF driver to reference the peripheral. For example, here are the N3EIF entries for a reference target system.

```
/* Definitions for peripheral N3EIF_0 */
#define XPAR_N3EIF_0_DEVICE_ID 0
#define XPAR_N3EIF_0_BASEADDR 0xCB000000
#define XPAR_N3EIF_0_HIGHADDR 0xCB00FFFF
```

As you can see, this instance of N3EIF has a base address of 0xCB000000 and a Device ID of 0. The I/O registers for N3EIF are referenced by adding an offset to `XPAR_N3EIF_0_BASEADDR`. The offsets for the N3EIF registers are as follows:

Register Name	Internal Name (Block Diagram)	Description	I/O	Offset
BTNSW_IN	btn_sw_out	Debounced buttons and switches	I	0x00
ROTSTS	rotary_sts	Rotary encoder Status	I	0x04
ROTCNTL	rotary_count_lo	Rotary count low byte	I	0x08
ROTCNTH	rotary_count_hi	Rotary count high byte	I	0x0C
LCDSTS	lcd_sts	LCD controller status	I	0x10
LEDS_DATA	leds_in	LED data (1 to turn on an LED)	O	0x14
ROTCTL	rotary_ctl	Rotary encoder control command	O	0x18
LCDCMD	lcd_cmd	LCD controller command	O	0x1C
LCDATA	lcd_data	LCD controller data	O	0x20
SPARE		*** RESERVED ***		0x24

While you don't need to understand the function of all these registers, they are used by the N3EIF to control the peripheral. For details refer to the section N3EIF Registers later in this document.

N3EIF Functions

N3EIF_init()

```
#include "n3eif.h"
XStatus N3EIF_init(u32 BaseAddress);
```

Initializes the Nexys 3 extended peripheral interface driver. The function waits until the N3EIF self-test is done and then the LCD display is cleared, the rotary encoder mode is set and the rotary encoder count is set to 0. *BaseAddress* is the memory-memory mapped address of the N3EIF I/O registers. An application program can get this base address from *xparameters.h* which is automatically generated by the Xilinx SDK (Software Development Kit). The function returns *XST_SUCCESS* if initialization is successful, *XST_FAILURE*, otherwise.

N3EIF_mReadReg()

```
#include "n3eif_l.h"
Xuint32 N3EIF_mReadReg(Xuint32 BaseAddress,
          unsigned RegOffset)
```

This function performs a 32-bit read of the value from *BaseAddress* + *RegOffset* in the I/O address space for the N3EIF. If the component is implemented in a smaller width, only the least significant data is read from the register. The most significant data will be read as 0. The function is implemented as a C macro (e.g. with a #define)

N3EIF_mWriteReg()

```
#include "n3eif_l.h"
void N3EIF_mWriteReg(Xuint32 BaseAddress,
         unsigned RegOffset, Xuint32 Data)
```

This function performs a 32-bit write of Data to *BaseAddress* + *RegOffset* in the I/O address space for the N3EIF. If the component is implemented in a smaller width, only the least significant data is written to the register. The function is implemented as a C macro (e.g. with a #define)

N3EIF_SelfTest()

```
#include "n3eif.h"
XStatus N3EIF_SelfTest(u32 baseaddr_p);
```

Runs a self-test on the N3EIF. This is not an exhaustive test, nor a hardware test, all it does is perform a write / read on several of the I/O registers, but it is enough to know that the peripheral is connected in your system and that the driver can communicate with it. . *BaseAddress* is the memory-memory mapped address of the N3EIF I/O registers. The function returns *XST_SUCCESS* if successful, *XST_FAILURE*, otherwise.

N3EIF_usleep()

```
#include "n3eif.h"
void N3EIF_usleep(u32 usec);
```

The N3EIFf_usleep() function adds a "usec" microseconds delay to your application program.

This function should be included in the Microblaze libc but it seems to be missing. This emulation assumes that the microblaze is running at 100MHz and takes 15 clocks per iteration through the delay loop - this may be totally bogus but it's a start.

Nexys3 Peripheral Functions

NX3_readBtnSw()

```
#include "n3eif.h"
XStatus NX3_readBtnSw(u32 *BtnSw_Data);
```

Samples the debounced push-buttons and switches from the Nexys3 board into the 32-bit variable pointed to by *BtnSw_Data*. Returns XST_SUCCESS. The calling function should pass the address (*&BtnSw_Data*) of the return location to the function. Returns XST_SUCCESS. You may use the following masks to select the buttons and/or switches that you are interested in (the values are in the rightmost 8-bits of the variable):

```
#define   msk_SWITCH0    (0x01)
#define   msk_SWITCH1    (0x02)
#define   msk_SWITCH2    (0x04)
#define   msk_SWITCH3    (0x08)
#define   msk_BTN_WEST   (0x10)
#define   msk_BTN_EAST   (0x20)
#define   msk_BTN_NORTH  (0x40)
#define   msk_BTN_ROT    (0x80)
```

NX3eif_writeleds()

```
#include "n3eif.h"
XStatus NX3_writeleds(u32 LED_Data);
```

Writes the LEDs on the Nexys3 board with the contents of *LED_Data*. The LED values should be placed in the rightmost 8 bits of *LEDS_Data*. The LEDs are addressed with LD7 (leftmost) on the Nexys3 in bit[7] of *LED_Data* and LD0 (rightmost) in bit[0]. Returns XST_SUCCESS.

Rotary Encoder Peripheral (PmodENC) Functions

ROT_clear()

```
#include "n3eif.h"
XStatus ROT_clear(void);
```

Clears the Rotary Encoder count. Does not affect the rotary encoder set-up. use ROT_init() to change the rotary encoder set-up. Returns XST_SUCCESS.

ROT_init()

```
#include "xps_s3eif_l.h"
XStatus ROT_init(int inc_dec_cnt, bool no_neg);
```

Initializes the Rotary Encoder control. *inc_dec_cnt* is the increment/decrement count. The count is truncated to 4 bits. *no_neg* TRUE causes the rotary encoder count to stop at 0 instead of going negative. Returns XST_SUCCESS.

NOTE : IT IS BEST TO CALL ROT_INIT() ONLY ONCE DURING YOUR INITIALIZATION PROCEDURE. CALLING THE FUNCTION REPEATEDLY IS NOT GUARANTEED TO PERFORM CORRECTLY.

ROT_readRotcnt

```
#include "n3eif.h"
XStatus ROT_readRotcnt(int* RotaryCnt);
```

Returns the rotary encoder count into the integer variable pointed to by *RotaryCnt*. The calling function should pass the address (*&RotaryCnt*) of the return location to the function. Returns XST_SUCCESS.

LCD Display (PmodCLP) Functions

LCD_clrd()

```
#include "n3eif.h"
XStatus LCD_clrd(void);
```

Writes blanks to the display and returns the cursor home. Returns XST_SUCCESS.

LCD_docmd()

```
#include "n3eif.h"
XStatus LCD_docmd(u32 lcdcmd, u32 lcddata);
```

Executes the LCD command in *lcdcmd* using the data in *lcdata*. The function controls the handshaking between the application and the N3EIF. "lcdcmd" and "lcddata" are 8-bit values located in the rightmost 8-bits of the parameter. Returns XST_SUCCESS.

LCD_docmd() is a low-level command to N3EIF that gives direct control of the display. The support commands are described in the Theory of Operations, Section LCD controller commands later in this document. For example, the command *Display On/Off* (function code 0x09) can be used to turn the cursor on or off or cause it to blink.

LCD_itoa()

```
#include "n3eif.h"
char* LCD_itoa(int value, char *string, int radix);
```

Converts the integer *value* to ASCII in base *radix*. **string* is a pointer to the string that the result is returned to. Returns the pointer to the string.

Notes:

- LCD_itoa() assumes string[] is long enough to hold the result plus the terminating null
- LCD_itoa() algorithm borrowed from ReactOS system libraries

LCD_setcgaddr()

```
#include "n3eif.h"
XStatus LCD_setcgadr(u32 addr);
```

Sets the character generator RAM Address to the value in the rightmost 6-bits of *addr*. The function also tells the LCD controller on the PmodCLP that the character data should be written to the character generator RAM instead of the data RAM. The character generator RAM contains 8 user defined custom characters. Returns XST_SUCCESS.

Refer to the Digilent PmodCLP Reference Manual and the Samsung KS0066 datasheet for details about the operation of the display.

LCD_setcursor()

```
#include "n3eif.h"
XStatus LCD_setcursor(u32 row, u32 col);
```

Sets the LCD cursor position to (*row*, *col*). The display is formed of 2 lines of 16 characters and each position has a corresponding address as indicated below:

Only the rightmost 4 bits of *row* and *col* are used with row being a value from 1..2 (1 being the top line of the display) and column being a value from 0..15 (0 being the leftmost position in the selected row). Returns XST_SUCCESS

LCD_setddaddr()

```
#include "n3eif.h"
XStatus LCD_setddadr(u32 addr);
```

Sets the data RAM address to the rightmost 7-bit value in *addr*. The function also tells the LCD controller that the character data should be written to the display RAM instead of the character generator RAM. Returns XST_SUCCESS.

Refer to the Digilent PmodCLP Reference Manual and the Samsung KS0066 datasheet for details about the operation of the display.

LCD_shiftl()

```
#include "n3eif.h"
XStatus LCD_shiftl(void);
```

Shifts the entire display left by one position without changing the display RAM contents. When the displayed data is shifted repeatedly, both lines move horizontally. The second display line does not shift into the first display line. Returns XST_SUCCESS

LCD_shiftr()

```
#include "n3eif.h"
XStatus LCD_shiftr(void);
```

Shifts the entire display right by one position without changing the display RAM contents. When the displayed data is shifted repeatedly, both lines move horizontally. The second display line does not shift into the first display line. Returns XST_SUCCESS

LCD_wrchar ()

```
#include "n3eif.h"
 XStatus LCD_wrchar(char ch);
```

Writes the character *ch* to the LCD display at the current cursor position. Returns XST_SUCCESS.

LCD_puthex()

```
#include "n3eif.h"
XStatus LCD_puthex(u32 num);
```

Writes the 32-bit unsigned number *num* to the LCD display at the current cursor position. Returns XST_SUCCESS.

LCD_putnum()

```
#include "n3eif.h"
XStatus LCD_putnum(int num, int radix);
```

Writes the 32-bit integer num to the LCD display at the current cursor position. radix is the base to output the number in. Returns XST_SUCCESS.

LCD_wrstring

#include "xps_s3eif.h"
XStatus LCD_wrstring(char* s);

The LCD_wrstring() function writes the null terminated string "s" to the LCD display starting at the current cursor position.

N3EIF Theory of Operation

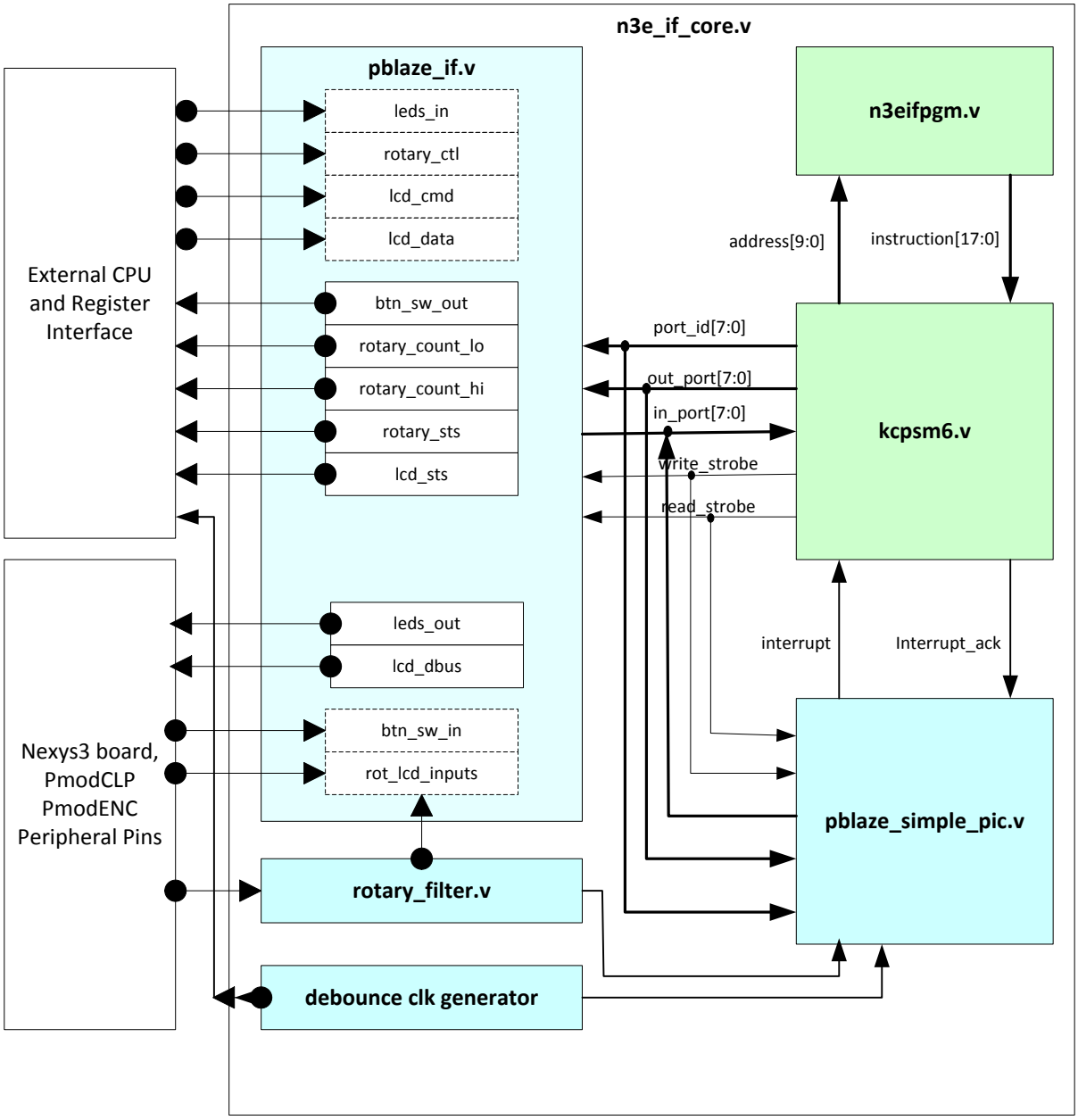
NOTE: YOU DO NOT NEED TO READ THE REMAINING SECTIONS OF THIS DOCUMENT TO USE THE N3EIF DRIVERS. IN FACT, IT MAY BE BETTER TO THINK OF THE N3EIF PERIPHERAL INTERFACE AS A "BLACK BOX" AND NOT DELVE INTO ITS IMPLEMENTATION UNLESS YOU ARE INTERESTED. THIS APPROACH WOULD BE TYPICAL OF IP THAT WAS PROCURED FROM A 3RD PARTY FOR YOUR PROJECT.

ON THE OTHER HAND, UNDERSTANDING HOW THE N3EIF PERIPHERAL IS IMPLEMENTED COULD PROVIDE YOU WITH INSIGHT INTO THE DESIGN OF PERIPHERALS AND DRIVERS IN SOC IMPLEMENTATIONS SUCH AS THOSE WE STUDY IN ECE 544

N3EIF Hardware Description

The Nexys3 Extended Peripheral interface is implemented as a Picoblaze-based (kcpsm6 - an 8-bit CPU soft core provided by Xilinx) system. The peripheral implementation includes logic external to the Picoblaze that provides the debounce clock, a rotary encoder interface, a multi-source priority interrupt controller and the registers described in the N3EIF RegistersN3EIF Registers section of this document. The peripheral also includes logic to interface the core functionality with the Microblaze peripheral bus in your system. In the form that you use the N3EIF for this course, the N3EIF is implemented as a custom peripheral and instantiated into your design using the Xilinx XPS. Refer to the Xilinx documentation for information and instructions for constructing your own custom peripheral and including it in your system.

The I/O devices controlled by N3EIF (Nexys3 board, PmodCLP, and PmodENC) are controlled by the Picoblaze which is running a program that borrows heavily from Ken Chapman's *Design for Spartan-3E Starter Kit (LCD Display Control)* application note. The assembly language source code is available from the instructor. The block diagram for the N3EIF is shown below:



The N3EIF is comprised of a number of HDL and documentation file that are automatically added to your system when you add an instance of N3EIF to your system. The table below summarizes the files that make up the design. As you can see, the N3EIF is a complete embedded computer system of its own.

Verilog Source Files	
File Name	**Description**
n3e_if_core.v	Top level Verilog file for the N3EIF. Instantiates and connects the Verilog files that implement the interface. Also contains the debounce clock generation circuitry. The debounce clock circuitry generates a 1ms clock pulse that is used to debounce the pushbuttons and switches. The debounce clock output is also brought out of the peripheral as an external signal and is available as an interrupt source or as a capture clock for an external logic analyzer, scope and/or a Chipscope embedded logic analyzer.
pblaze_if.v	Implements the register-based interface to an external CPU. pblaze_if.v connects to the Picoblaze CPU embedded in the peripheral and is controlled using Picoblaze input/output instructions. In addition to the user-visible registers used by the driver, pblaze_if.v also receives the inputs and drives the outputs of the pins connected to the Nexys3 (pushbuttons, switches, LEDs) and the PmodCLP (LCD display).
rotary_filter.v	Implements the interface to the PmodENC rotary encoder. Receives the rotary A and rotary B inputs from the Pmod and generates *rotary_event* which signals the Picoblaze that the rotary encoder shaft has moved and the *rotary_left* signal which is used by the Picoblaze program to determine the direction the shaft turned. Ken Chapman's *Rotary Encoder Interface for Spartan-3E Starter Kit* application note explains the operation of this logic.
pblaze_simple_pic.v NOTE: This circuit was orignally	8-input priority interrupt controller for the Picoblaze. Expands the number of interrupt sources to a Picoblaze. The N3EIF relies on two interrupt sources to the

created by Richard Herveille and posted to opencores.org. His circuit was adapted to the Picoblaze by Roy Kravitz. This interrupt controller can be used in any Picoblaze-based design that needs more than one interrupt source. The Verilog source file includes a short "How To" for how to use and control the interrupt controller in your Picoblaze program.	Picoblaze. The 1ms debounce clock is used by the firmware to debounce the push-buttons and switches. The second interrupt source is the *rotary_event* signal from rotary_filter.v. *rotary_event* is asserted each time the rotary encoder shaft is advanced either to the left or the right.
kcpsm6.v	The Xilinx Picoblaze soft core for the Spartan 6 family The latest version of the Picoblaze should be downloaded from the Xilinx web site.
n3ifpgm.v	Program file for the the Picoblaze. Produced by the kcpsm6 Assembler from n3ifpgm.psm. The assembly language source code is available from the instructor.
user_logic.v	Verilog wrapper for the N3EIF custom peripheral. Connects the N3EIF core logic to the Microblaze registers used by the drivers. user_logic.v is instantiated by n3eif.vhd
n3eif.vhd	VHDL wrapper file that connects the N3EIF logic with the PLB bus and makes the peripheral available to the system builder. Refer to the Xilinx documentation on custom peripherals to understand the usage and implementation model.

Additional Files Included in the Distribution	
File Name	**Description**
N3EIF_UG.pdf	Nexys3 Expanded Peripheral Interface User Guide.

	This document. Describes register-based interface and API's for Microblaze-based systems
n3eif_test.c	C source code for the program used to test the N3EIF drivers and API. Gives examples for how to use some of the API function calls.
n3eif_l.h, n3eif.h	C header files for the N3EIF. n3eif_l.h is the header file for the low level driver calls and is automatically included by n3eif.h. n3eif.h is the header file for the N3EIF driver API.
n3eif.c	C source file for the N3EIF drivers API.

N3EIF Registers

An external Microblaze communicates with the N3EIF peripheral through ten (10) 8-bit I/O. Two of these registers are used to drive the LEDS and read the debounced push-buttons and switches. Four registers are used to control and query the PmodENC. The Rotary Encoder on the PmodENC returns a 16-bit count and status and has several parameters that can be set through its control register. The remaining registers are used to manage the 2-line by 16-character LCD screen on the PmodCLP. Commands are sent to the N3EIF by writing function codes to the LCD command register. Data such as the character to display are sent to the LCD by writing to the LCD data. LCD status is returned to the Microblaze by reading the LCD status register. There is also one spare register which is reserved for future use. The spare register is written and read by the N3EIF self-test to confirm communication between the driver and the peripheral.

The N3EIF produces a 1ms clock (called *db_clk*) that can be used as an interrupt source or timing signal to the external CPU. The same clock is used to debounce the push-buttons and switches.

The N3EIF register interface can be summarized as follows:

Register Name	Internal Name (Block Diagram)	Description	I/O	Offset from Base Address of Peripheral
BTNSW_IN	btn_sw_out	Debounced buttons and switches	I	0x00
ROTSTS	rotary_sts	Rotary encoder Status	I	0x04
ROTCNTL	rotary_count_lo	Rotary count low byte	I	0x08
ROTCNTH	rotary_count_hi	Rotary count high byte	I	0x0C
LCDSTS	lcd_sts	LCD controller status	I	0x10
LEDS_DATA	leds_in	LED data (1 to turn on an LED)	O	0x14
ROTCTL	rotary_ctl	Rotary encoder control command	O	0x18
LCDCMD	lcd_cmd	LCD controller command	O	0x1C
LCDATA	lcd_data	LCD controller data	O	0x20
SPARE		*** RESERVED ***		0x24

* Direction is relative to the Microblaze. For example, LEDS_DATA is an output (port write) from the Microblaze. BTNSW_IN is an input (port read) to the Microblaze

LEDS AND PUSH-BUTTONS/SWITCHES

The two simplest functions of the n3eif are the LED output and push-button input. The LEDs are lit (or not lit) by writing an 8-bit value to the *LEDS_DATA* register. The debounced push-button and switches

are read from the 8-bit *BTNSW_IN register*.

BTNSW_IN

B_ROT	B_NORTH	B_EAST	B_WEST	SW[3]	SW[2]	SW[1]	SW[0]

NOTE: B_SOUTH IS NOT IMPLEMENTED AS AN INPUT TO THIS REGISTER. B_SOUTH IS USED AS A GLOBAL SYSTEM RESET SIGNAL.

NOTE: THE NEXYS3 BOARD HAS 8 SLIDE SWITCHES. ONLY THE LOW ORDER 4 SWITCHES ARE AVAILABLE FROM N3EIF. THIS WAS DONE TO MAINTAIN SOFTWARE COMPATIBILITY WITH THE S3E STARTER BOARD PERIPHERAL. THE REMAINING SWITCHES AND THE SLIDE SWITCH ON THE PMODENC CAN BE MADE AVAILABLE BY ADDING A GPIO INSTANCE TO YOUR SYSTEM AND CONNECTING THE PINS TO IT.

B_ROT Debounced Rotary Encoder pushbutton. A value of 1 indicates that the button is pressed. A value of 0 indicates that the button is not pressed.

B_NORTH Debounced push-button North. A value of 1 indicates that the button is pressed. A value of 0 indicates that the button is not pressed. B_North is located above the Rotary Encoder.

B_EAST Debounced push-button East. A value of 1 indicates that the button is pressed. A value of 0 indicates that the button is not pressed. B_East is located to the right of the Rotary Encoder.

B_WEST Debounced push-button West. A value of 1 indicates that the button is pressed. A value of 0 indicates that the button is not pressed. B_West is located to the left of the Rotary Encoder.

SW[3:0] Debounced slide switch inputs. A value of 1 indicates that the switch is on (up). A value of 0 indicates that the switch is off (down). SW3 is the leftmost switch. SW0 is the rightmost switch.

LEDS_IN

LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0

LED[7:0] LED bits 7:0. Bit 7 is the leftmost bit in the LED bank (LD7); bit 0 is the rightmost bit (LD0). A LED is lit by writing a 1 to the appropriate bit.

THE ROTARY ENCODER

The rotary encoder on the PmodENC integrates two different functions. The switch shaft rotates and outputs a count whenever the shaft turns. The shaft can also be pressed, acting as a push-button switch. The push-button switch is debounced and returned in bit[7] of the *BTNSW_IN* register as described earlier. Rotary shaft movements are returned in the *ROTCNTL* and *ROTCNTH* registers described below.

ROTARY COUNT REGISTER

The Rotary Count register consists of two 8-bit registers that are combined into a single 16-bit integer by the N3EIF driver. The Rotary Count register is incremented each time the Rotary Encoder shaft is turned one direction and decremented each time the shaft is turned the other direction. Since it is possible for the count to go negative (if the shaft is turned to the left more than to the right) the Rotary Count register is implemented as a two's-complement counter.

C[15]	C[14]	C[13]	C[12]	C[11]	C[10]	C[9]	C[8]

C[7]	C[6]	C[5]	C[4]	C[3]	C[2]	C[1]	C[0]

C[15:8] Most significant byte of the 16-bit rotary count. These bits are located in *the ROTCNTH* register.

C[7:0] Least significant byte of the 16-bit rotary count. These bits are located in *the ROTCNTL* register.

ROTARY ENCODER CONTROL REGISTER

The information returned by Rotary Encoder can be controlled, to a certain extent, by reading and writing to the *ROTCTL* register. Specifically, the Rotary Encoder control register can be used to clear the rotary count (return the value to 0) and to set the increment/decrement count (the amount the count is changed by each shaft rotation) and the operating mode of the count (whether the count stops at 0 or is allowed to go negative). The rotary count can be cleared at any time but it is suggested that the increment/decrement count and operating mode be set once during the initialization phase of your application.

The Rotary Encoder defaults to an increment/decrement count of 1 with negative counts enabled.

CLR	LDCFG	RES	NONEG	I/D CNT[3:0]

CLR Clear the rotary count. Toggling the CLR bit (setting it to 0 -> 1 -> 0) will set the value of

the Rotary Count register to 0.

LDCFG Load Rotary Encoder configuration. Toggling the LDCFG bit (setting it to 0 -> 1 -> 0) sets the operation of the rotary encoder counter to the values of the NONEG and I/D CNT fields in the register. It is recommended that this only be done during initialization and not dynamically during the operation of your application.

NONEG Allow/don't allow negative counts. Setting this bit to 1 limits the rotary count to numbers greater than or equal to 0. Once the count reaches 0 further shaft that would decrement the count will not cause the count to go negative. Setting the bit to 0 allows negative counts. Changes to this bit will only take effect after the LDCFG bit in the register is toggled. The default configuration is to allow negative counts (NONEG = 0).

I/D CNT[3:0] Increment/decrement count. The Rotary Encoder count is incremented and decremented by a fixed amount on each operation. The amount that the counter is incremented or decremented can be set by placing a 4-bit positive number in the I/D CNT field. Changes to the increment/decrement count will only take effect after the LDCFG bit in the register is toggled. The default increment/decrement count is 1.

ROTARY ENCODER STATUS REGISTER

The Rotary Encoder controller in the N3EIF returns two status bits that can be used by an application or driver to synchronize its activity with that of the N3EIF firmware. The bits in this status register indicate whether it is safe to perform operations. While it is not absolutely necessary for an application to test (and busy-wait on this status), doing so will ensure that the application receives a consistent view of the registers in the interface. For example, since the N3EIF is operating asynchronously tothe application it is possible for the value of one of the Rotary Encoder count registers to be updated in a way that provides an incorrect value when the registers are taken as a pair. Polling the BUSY bit in this register and waiting until the bit is cleared will insure that the rotary count registers are consistent.

BUSY	SLFTST	x	x	x	x	x	x

BUSY Rotary Encoder Busy flag. A value of 1 indicates that the N3EIF is operating on the Rotary Encoder register set and may be in the process of updating the user visible registers. A value of 0 indicates that it is safe for the application to perform an operation.

SLFTST Self-test in progress flag. A value of 1 indicates that the N3EIF is executing its hardware self-test and is unavailable for operations. The self-test may be executed once by the N3EIF firmware at startup (configurable in the firmware source code). A value of 0 indicates that the hardware self test is complete and the N3EIF is ready to accept commands.

THE LCD DISPLAY CONTROLLER

The LCD display controller in the N3EIF is capable of executing commands that position the cursor and place characters on the screen. Commands are sent to the LCD display controller in the PmodCLP in the *lcd_cmd* register. Data (ex: the ASCII code of a character to write to the display) for the command is sent to the LCD display controller in the *lcd_data* register. Like the Rotary Encoder controller, the LCD display controller returns "busy" status in the *lcd_status* register.

LCD COMMAND REGISTER

The LCDCMD register contains a 5-bit field specifying the command to be executed by the controller. The command codes are summarized in the LCD controller commands section of this user guide. A command is initiated by toggling the DOCMD bit in the register. The safest way to initiate a new command to the LCD controller is to:

1. Loop on the BUSY bit in the *LCDSTS* register until it is 0. This insures that the LCD controller is free to accept a new command.
2. Write the data needed by the new command to the *LCDATA* register.
3. Write the LCD command to be executed to the *LCDCMD* register. Keep the DOCMD bit at 0.
4. Write a 1 to the DOCMD bit in the *LCDCMD* register preserving the CMD bits in the register. This initiates the new command.
5. Loop on the BUSY bit in the *LCDSTS* register until it is 0. The BUSY bit will be set to 1 for the entire time the command is being executed.
6. Write a 0 to the DOCMD bit in the *LCDCMD* register to complete the command and prepare the *LCDCMD* register for the next command.
7. Insert a delay (2 msecs is plenty) before sending a new command to the N3EIF. This delay insures that the LCD controller has completed the operation on the display.

DOCMD	x	x	CMD[4]	CMD[3]	CMD[2]	CMD[1]	CMD[0]

DOCMD — Do command. Toggling the DOCMD bit (setting it to 0 -> 1) will initiate the LCD command specified in CMD[4:0]. The bit should be set back to 0 when the command is complete.

CMD[4:0] — LCD controller command. The LCD controller in the N3EIF is capable of executing most of the commands described in the PmodCLP reference manual. This field should be set to the appropriate command code as specified in the LCD controller commands section.

LCD DATA REGISTER

The *LCDATA* register should be loaded with any data needed for an LCD controller command before the

command is initiated in the *LCDCMD* register.

D[7]	D[6]	D[5]	D[4]	D[3]	D[2]]	D[1]	D[0]

D[7:0] LCD command data. The LCD controller in the N3EIF is capable of executing commands that require data (ex: write character). The data for the command should be loaded into D[7:0] before the command is initiated.

LCD CONTROLLER STATUS REGISTER

The LCD display controller in the N3EIF returns a status bit that can be used by an application running on an external CPU to synchronize its activity with that of the N3EIF firmware. While it is not absolutely necessary for an application to test (and busy-wait on) this status, doing so will ensure that the LCD firmware in the N3EIF is ready to accept a new command.

The LCD controller status register is used by the "read character" command to return the low order 7 bits of the ASCII code of the character at the current cursor position. The most significant bit of the character code is truncated because *lcd_sts*[7] is the BUSY bit. Truncating the high order bit is not a problem if the character is alphanumeric or a printable special character (such as : or .). Bit 7 of all of those characters is set to 0. However, characters with bit 7 set to 1 (such as the Japanese kana characters in the LCD character ROM) will not be correct. Figure 5-4 in the *Spartan-3E Starter Kit Board User Guide* contains a table of the LCD character set for the LCD display.

BUSY	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD	RSVD

BUSY LCD controller Busy flag. A value of 1 indicates that the N3EIF is executing an LCD command. A value of 0 indicates that it is safe for the application to initiate a new LCD command.

RSVD Reserved for future use.

LCD CONTROLLER COMMANDS

NOTE: READ THE PMODCLP AND SAMSUNG KS006U DATASHEET TO BETTER UNDERSTAND THESE COMMAND DESCRIPTIONS.

The LCD display controller in the N3EIF is capable of executing the following commands:

CMD Code	Command Name	LCD Data	Description
0x00	NOP	NONE	No Operation
0x01	Set Cursor Position	D[7:4] – Line # D[3:0] – Char #	Position the cursor to (line, char). The display is formed of 2 lines(1-2) of 16 characters (0 – 15) each.
0x02	Write Character	D[7:0] – ASCII character code for character to be written (if address mode is set to DD). D[7:0] – Bit field for user generated characters (if address mode is set to CG)	Writes the character in *LCDATA* to the LCD display at the current cursor position if the address mode is set to DD. Writes the bit field in LCDATA to the LCD character generator ROM at the current CG address if the address mode is set to CG
0x03	Read Character NOTE: THIS COMMAND IS NOT IMPLEMENTED BY N3EIF	*lcd_sts*[6:0] – low order 7 bits of the ASCII character at the current cursor position. The Read Character Command does not use the *lcd_data* register	Returns the ASCII character code for the character at the current cursor position of the LCD display if the address is set to DD Returns the bit field at the current Character ROM address if the address mode is set to CG
0x04	Clear Display	NONE	Clears the display and returns the cursor position to line 1, position 0 (top left character position). Writes spaces (ASCII 0x20) to all of the character positions.
0x05	Return Cursor Home	NONE	Return the cursor to the home position (top-left corner). DD RAM contents are unaffected. Also returns the display being shifted to the original position.
0x06	Set CG Address	D[7:6] – Not used D[5:0] – New CG Address	Sets the initial CG RAM address. After this command all subsequent read or write operations to the display are to or from CG RAM (e.g. the display is in CG mode)
0x07	Set DD Address	D[7] – Not used D[6:0] – New DD	Sets the initial DD RAM address. After this command all subsequent read or write operations to the display are to or from DD

		(display data memory) Address. The DD RAM addresses are shown in Figure 5.3 of the *Spartan-3E Starter Kit Board User Guide*	RAM (e.g. the display is in DD mode)
0x08	Set Display Entry Mode	D[7:2] – Not used D[1] – Auto Increment (1) or decrement (0) address counter D[0] – Shift entire display in the direction controlled by D[1] (1) or disable shifting (0)	Sets the cursor move direction and specifies whether or not to shift the display. These operations are performed during data reads and writes
0x09	Display On/Off	D[7:3] – Not used D[2] – Display characters (1) or don't display characters (0). D[1] – Display cursor (1) or do not display cursor (0) D[0] – Blink the cursor (1) or do not blink cursor (0)	Turns the display on or off, controlling all characters, cursor and cursor position character (underscore) blink.
0x0A	Shift Display Left	NONE	Shifts the display left one position without changing DD RAM contents. When the displayed data is shifted repeatedly, both lines move horizontally.The second display line does not shift into the first display line
0x0B	Shift Display Right	NONE	Shifts the display right one position without changing DD RAM contents. When the displayed data is shifted repeatedly, both lines

			move horizontally. The second display line does not shift into the first display line
0x0C	Move Cursor Left	NONE	Positions the cursor one position to the left in order to modify an individual character, or to scroll the display window left to reveal additional data stored in the DD RAM beyond the 16th character on a line. The cursor automatically moves to the first line when it moves beyond the 0th location on the second line (I think)
0x0D	Move Cursor Right	NONE	Positions the cursor one position to the right in order to modify an individual character, or to scroll the display window right to reveal additional data stored in the DD RAM beyond the 16th character on a line. The cursor automatically moves to the second line when it moves beyond the 40th location on the first line.

Appendix A – N3EIF Sample Constraints File (.ucf)

NOTE: ASSUMES THAT THE PMODCLP IS CONNECTED TO THE JA AND JB PMOD CONNECTORS ON THE NEXYS3 BOARD.

```
## This file is an edited version of the master  .ucf for Nexys3 rev B board
## Only the signals used in the test_n3if system are listed.

##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;

## Leds
Net "Led<0>" LOC = U16 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L2P_CMPCLK, Sch name =
                    LD0
Net "Led<1>" LOC = V16 |  IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L2N_CMPMOSI, Sch name
                    = LD1
Net "Led<2>" LOC = U15 |  IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L5P, Sch name = LD2
Net "Led<3>" LOC = V15 |  IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L5N, Sch name = LD3
Net "Led<4>" LOC = M11 |  IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L15P, Sch name = LD4
Net "Led<5>" LOC = N11 |  IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L15N, Sch name = LD5
Net "Led<6>" LOC = R11 |  IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L16P, Sch name = LD6
Net "Led<7>" LOC = T11 |  IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L16N_VREF, Sch name =
                    LD7

## Switches
Net "sw<0>" LOC = T10 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L29N_GCLK2, Sch name =
                    SW0
Net "sw<1>" LOC = T9 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L32P_GCLK29, Sch name =
                    SW1
Net "sw<2>" LOC = V9 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L32N_GCLK28, Sch name =
                    SW2
Net "sw<3>" LOC = M8 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L40P, Sch name = SW3
Net "sw<4>" LOC = N8 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L40N, Sch name = SW4
Net "sw<5>" LOC = U8 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L41P, Sch name = SW5
Net "sw<6>" LOC = V8 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L41N_VREF, Sch name = SW6
Net "sw<7>" LOC = T5 | IOSTANDARD = LVCMOS33; #Bank = MISC, pin name = IO_L48N_RDWR_B_VREF_2,
                    Sch name = SW7

## Buttons
Net "btns" LOC = B8 | IOSTANDARD = LVCMOS33; #Bank = 0, pin name = IO_L33P, Sch name = BTNS
Net "btnu" LOC = A8 | IOSTANDARD = LVCMOS33; #Bank = 0, pin name = IO_L33N, Sch name = BTNU
Net "btnl" LOC = C4 | IOSTANDARD = LVCMOS33; #Bank = 0, pin name = IO_L1N_VREF, Sch name = BTNL
Net "btnd" LOC = C9 | IOSTANDARD = LVCMOS33; #Bank = 0, pin name = IO_L34N_GCLK18, Sch name =
                    BTND
Net "btnr" LOC = D9 | IOSTANDARD = LVCMOS33; # Bank = 0, pin name = IO_L34P_GCLK19, Sch name =
                    BTNR

## 12 pin Expanasion (Pmod) connectors

##JA - PmodCLP data bus for this system
```

```
Net "JA<0>" LOC = T12 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L19P, Sch name = JA1
Net "JA<1>" LOC = V12 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L19N, Sch name = JA2
Net "JA<2>" LOC = N10 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L20P, Sch name = JA3
Net "JA<3>" LOC = P11 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L20N, Sch name = JA4
Net "JA<4>" LOC = M10 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L22P, Sch name = JA7
Net "JA<5>" LOC = N9 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L22N, Sch name = JA8
Net "JA<6>" LOC = U11 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L23P, Sch name = JA9
Net "JA<7>" LOC = V11 | IOSTANDARD = LVCMOS33; #Bank = 2, pin name = IO_L23N, Sch name = JA10

##JB - PmodCLP control signals for this system
Net "JB<0>" LOC = K2 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L38P_M3DQ2, Sch name = JB1
Net "JB<1>" LOC = K1 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L38N_M3DQ3, Sch name = JB2
Net "JB<2>" LOC = L4 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L39P_M3LDQS, Sch name =
                    JB3
Net "JB<3>" LOC = L3 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L39N_M3LDQSN, Sch name =
                    JB4
Net "JB<4>" LOC = J3 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L40P_M3DQ6, Sch name = JB7
Net "JB<5>" LOC = J1 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L40N_M3DQ7, Sch name = JB8
Net "JB<6>" LOC = K3 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L42N_GCLK24_M3LDM, Sch
                    name = JB9
Net "JB<7>" LOC = K5 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name =
                    IO_L43N_GCLK22_IRDY2_M3CASN, Sch name = JB10

##JC - Debug signals for this system
Net "JC<0>" LOC = H3 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L44N_GCLK20_M3A6, Sch
                    name = JC1
Net "JC<1>" LOC = L7 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L45P_M3A3, Sch name = JC2
Net "JC<2>" LOC = K6 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L45N_M3ODT, Sch name = JC3
Net "JC<3>" LOC = G3 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L46P_M3CLK, Sch name = JC4
Net "JC<4>" LOC = G1 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L46N_M3CLKN, Sch name =
                    JC7
Net "JC<5>" LOC = J7 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L47P_M3A0, Sch name = JC8
Net "JC<6>" LOC = J6 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L47N_M3A1, Sch name = JC9
Net "JC<7>" LOC = F2 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L48P_M3BA0, Sch name = JC10

##JD,- PmodENC signals for this system
Net "JD<0>" LOC = G11 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L40P, Sch name = JD1
Net "JD<1>" LOC = F10 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L40N, Sch name = JD2
Net "JD<2>" LOC = F11 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L42P, Sch name = JD3
Net "JD<3>" LOC = E11 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L42N, Sch name = JD4
Net "JD<4>" LOC = D12 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L47P, Sch name = JD7
Net "JD<5>" LOC = C12 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L47N, Sch name = JD8
Net "JD<6>" LOC = F12 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L51P, Sch name = JD9
Net "JD<7>" LOC = E12 | IOSTANDARD = LVCMOS33; #Bank = 3, pin name = IO_L51N, Sch name = JD10
```

Appendix B- N3EIF Sample Top-Level Verilog File

NOTE: THIS IS A SAMPLE. THE INSTANTIATION OF THE EMBEDDED SYSTEM (EMBSYS) WILL VARY DEPENDING ON HOW YOU CONFIGURED THE EXTERNAL PORTS IN THE SYSTEM BUILDER

```
// n3fpga.v - Top level module for test_n3if
//
// Copyright Roy Kravitz, Portland State University 2013, 2014, 2015
//
// Created By: Roy Kravitz
// Date:                    14-March-2013
// Version:          1.0
//
// Description:
// ------------
// This module provides the top level for testing the Nexys3 interface
// peripheral.  It assume that a PmodCLP is plugged into the JA and JB
// expansion ports and that a PmodENC is plugged into the JD expansion
// port.
//////////////////////////////////////////////////////////////////////
module n3fpga(
   input                clk,          // 100Mhz clock input
   input                btns,         // center pushbutton
   input                btnu,         // north pusbhbutton
   input                btnl,         // west pushbutton
   input                btnd,         // south pushbutton  - used for system reset
   input                btnr,         // east pushbutton
   input       [7:0]    sw,           // slide switches on Nexys 3
   output      [7:0]    Led,          // Leds on Nexys 3
   input                RxD,          // USB UART Rx on Nexys 3
   output               TxD,          // USB UART Tx on Nexys 3
   inout       [7:0]    JA,           // JA Pmod connector -
                                      // PmodCLP data bus
                                      // both rows are used
   inout       [7:0]    JB,           // JB Pmod connector - PmodCLP control
                                      // signals
                                      // only the bottom row is used
   inout       [7:0]    JD,           // JD Pmod connector - PmodENC signals
                                      // only the bottom row is used
```

```
   inout       [7:0]       JC          // JC used as debug header
);

// internal variables
wire                    sysclk;
wire                    sysreset;
wire    [31:0]          gpio;
wire                    rotary_a, rotary_b, rotary_press, rotary_sw;
wire    [7:0]           lcd_d;
wire                    lcd_rs, lcd_rw, lcd_e;

reg                     clk_1ms;

// make the connections

// system-wide signals
assign sysclk = clk;
assign sysreset = btnd;

// PmodCLP signals
// JA - top and bottom rows
// JB - bottom row only
assign JA = lcd_d[7:0];
assign JB = {1'b0, lcd_e, lcd_rw, lcd_rs, 4'b0000};

// PmodENC signals
// JD - bottom row only
assign rotary_a = JD[4];
assign rotary_b = JD[5];
assign rotary_press = JD[6];
assign rotary_sw = JD[7];

// GPIO signals (extra slide switches)
assign gpio = {24'h000000, 1'b0, rotary_sw, btns, sw[7:4]};

// Debug signals are on top row of JC
assign JC = {4'b0000, clk_1ms, rotary_a, rotary_b, rotary_press};

// create 50% duty cycle version of debounce clock (1KHz/2 = 500Hz)
always @(posedge dbclk)
```

```
        clk_1ms = ~clk_1ms;


// instantiate the embedded system
// Instantiate the module
(* BOX_TYPE = "user_black_box" *)
system EMBSYS (
   .fpga_0_RS232_Uart_1_RX_pin(RxD),
   .fpga_0_RS232_Uart_1_TX_pin(TxD),
   .fpga_0_mem_bus_mux_0_MEM_ADDR_pin(),
   .fpga_0_mem_bus_mux_0_DQ_pin(),
   .fpga_0_mem_bus_mux_0_MEM_OEN_pin(),
   .fpga_0_mem_bus_mux_0_MEM_WEN_pin(),
   .fpga_0_mem_bus_mux_0_RAM_CEN_O_pin(),
   .fpga_0_mem_bus_mux_0_RAM_BEN_O_pin(),
   .fpga_0_mem_bus_mux_0_FLASH_ADDR_pin(),
   .fpga_0_mem_bus_mux_0_FLASH_CEN_O_pin(),
   .fpga_0_mem_bus_mux_0_FLASH_RPN_O_pin(),
   .fpga_0_mem_bus_mux_0_QUAD_SPI_C_O_pin(),
   .fpga_0_mem_bus_mux_0_QUAD_SPI_S_O_pin(),
   .fpga_0_mem_bus_mux_0_MOSI_QUAD_SPI_pin(),
   .fpga_0_clk_1_sys_clk_pin(sysclk),
   .fpga_0_rst_1_sys_rst_pin(sysreset),
   .n3eif_0_rotary_press_pin(rotary_press),
   .n3eif_0_btn_north_pin(btnu),
   .n3eif_0_rotary_a_pin(rotary_b),   // flip to make right turn increment
   .n3eif_0_btn_west_pin(btnl),
   .n3eif_0_leds_out_pin(Led),
   .n3eif_0_rotary_b_pin(rotary_a),  // flip to make right turn increment
   .n3eif_0_sw_pin(sw[3:0]),
   .n3eif_0_lcd_rs_pin(lcd_rs),
   .n3eif_0_lcd_rw_pin(lcd_rw),
   .n3eif_0_lcd_e_pin(lcd_e),
   .n3eif_0_lcd_d_pin(lcd_d),
   .n3eif_0_db_clk_pin(dbclk),
   .n3eif_0_btn_east_pin(btnr),
         .xps_gpio_0_GPIO_IO_I_pin(gpio)
);

endmodule
```