Tutorial 2
Learning EDK's “Create/Import Peripheral Wizard”

-Introduction

The “Create/Import Peripheral Wizard” in the Xilinx Embedded Development Kit (EDK) assists in creating a custom peripheral that is accessible from a PowerPC or Microblaze processor. When invoked, the user specifies several parameters that are used to create an ISE project with VHDL template files along with low-level software drivers. The user edits the generated files to implement the desired functionality. Once this is complete, the user must re-invoke the wizard to import the updates made to the peripheral. Finally, the peripheral is ready to be added to an EDK project.

This brief tutorial will use the “Create/Import Peripheral Wizard” to create a switch debouncer peripheral that will connect to the PowerPC over the Processor Local Bus (PLB). The peripheral will have one addressable register that will hold the state of the switches and the ability to interrupt the processor when the state of the switches changes. The peripheral will be added to an EDK project for implementation and testing on a V2Pro Evaluation Board. Please note that this tutorial barely covers the basics of creating peripheral in EDK. The reader is strongly advised to reference the EDK help topic on this subject for more information.

-Prerequisites

1. ISE 10.1.03 and EDK 10.1.03
2. V2Pro Evaluation Board
3. RS232 serial connection and terminal
4. Completed Tutorial 1 (Introduction to EDK 10.1 and the Digilent V2Pro Board)
5. An EDK base system for the V2Pro board with the following peripherals: RS232_UART, LEDs_4Bit, and Push.Buttons.Position (enable Use interrupt).

-Creating the Peripheral

1. From EDK, go to the Hardware menu and click on Create or Import Peripheral... Click Next at the Welcome screen for the Create and Import Peripheral Wizard.
2. At the Peripheral Flow screen, make sure the radio button for Create templates for a new peripheral is selected and click Next.

3. At the Repository or Project screen, choose where to store the peripheral and click Next. If you want to be able to re-use the peripheral in other EDK projects then store the peripheral in a repository. If you want to bind this peripheral to a specific EDK project then store the peripheral in that projects directory.

4. At the Name and Version screen, enter the name of the peripheral, the version number, and a description if desired. For this tutorial, set the name to switch_debouncer and the version number to 1.00.a. Click Next.
5. At the **Bus Interface** screen, select the **Processor Local Bus (PLB v4.6)** radio button and click **Next**. Please note that the Fast Simplex Link bus is only available for the Microblaze processor.

6. The next screen is the **IP Interface Services** screen. The IP Interface (IPIF) provides a variety of services for easily connecting your custom peripheral to a processor bus. A detailed description of each service can be found in the IPIF Features document. For this tutorial, we will use the **Interrupt control** and **User logic software register** services. The interrupt control service provides the means for our switch debouncer peripheral to interrupt the processor while the user logic software register service provides addressable registers. Click **Next**.
7. Keep the default options at the **Slave Interface** screen and click **Next**.

8. At the **Interrupt Service** screen, remove the check mark for the **Use Device ISC (interrupt source controller)** option. Keep the **Number of interrupts generated by user logic** at 1. Change the **Capture mode** to **Rising Edge Detect**. This means that the switch debouncer will have to generate a rising edge on the IPIF interrupt source line to generate an interrupt. Furthermore, software will have to clear the interrupt via the IP Interrupt Status Register.

9. Keep the **Number of software accessible registers** set to 1 at the **User S/W Register** screen. Most peripherals will probably need more than one register. The IPIF for the PLB v4.6 bus supports up to 4096 registers. Our switch debouncer peripheral needs only one register which will hold the state of the switches.

10. The **IP Interconnect (IPIC)** screen allows you to select additional interface signals for your peripheral. The signals that have been pre-selected are a result of the IPIF services chosen earlier. Most of the time, you will not need to add any additional signals. Click **Next**.

11. The **(OPTIONAL) Peripheral Simulation Support** screen provides support for simulating your entire peripheral. However, if you properly modularize your peripheral design you can simply simulate the core functionality. This tutorial will not use this support. Click **Next**.
12. The (OPTIONAL) Peripheral Implementation Support screen determines what kind of template files are generated for your peripheral. Keep the default options checked. This will create an ISE project with template source code in VHDL along with template driver files in C. Click Next.

13. Click Finish at the Congratulations screen.

Upon completion, the “Create or Import Peripheral” wizard creates two folders called pcores and drivers in the directory specified in step 3 for the peripheral. The pcores directory contains the ISE project, VHDL template files, and settings files for the peripheral. The drivers directory contains the C driver template files for the peripheral. It is recommended that the reader becomes more familiar with the files and directory structure generated by the peripheral wizard by referencing the help topic. Another great reference is the README.txt file found in \pcores\switch_debouncer\devl\.

-Customizing the Peripheral Template Files

1. Start ISE and open the switch_bouncer project found in pcores\switch_debouncer_v1_00_a\devl\projnav\. Expand the switch_debouncer top module in the Sources pane as shown below. You should now see the following four VHDL modules: switch_debouncer, PLBV46_SLAVE_SINGLE_I, INTERRUPT_CONTROL_I, and USER_LOGIC_I. The USER_LOGIC_I module is the primary module that you
will need to edit to add your custom functionality to the peripheral. You will also have to make minor changes to the top-level switch_debouncer module.

2. Right-click on the USER_LOGIC_I module in the Sources pane and select New Source. At the Select Source Type screen, set the Source Type to VHDL Module, set the File name to switch_debouncer_core, and set the Location to `\pcore\switch_debouncer_v1_00_a\hdl\vhdl\`. Click Next.

3. Click Next at the Define Module screen.

4. Click Finish at the Summary screen.

5. Delete all of the generated code in the switch_debouncer_core.vhd file. Copy the code provided below to implement the switch debouncer. Alternatively, you can write your own switch debouncer module.
Sample VHDL Code for the Switch Debouncer Code

-- SWITCH_DEBOUNCER_CORE.VHD
-- This VHDL module "debounces" an array of digital input switches. The
-- INPUT_SWITCH_ARRAY must be stable for [DELAY] CLOCK cycles before it is
-- assigned to the OUTPUT_SWITCH_ARRAY.
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;
use ieee.math_real.all;
entity SWITCH_DEBOUNCER_CORE is
generic
(
 NUM_SWITCHES : integer := 8; -- Width of input & output
 DELAY : integer := 8
);
port
(
 RESET : in std_logic;
 CLOCK : in std_logic;
 INPUT_SWITCH_ARRAY : in std_logic_vector((NUM_SWITCHES-1) downto 0);
 OUTPUT_SWITCH_ARRAY : out std_logic_vector((NUM_SWITCHES-1) downto 0);
 NEW_SWITCH_STATE : out std_logic;
 ACKNOWLEDGE : in std_logic
);
end SWITCH_DEBOUNCER_CORE;
architecture Behavioral of SWITCH_DEBOUNCER_CORE is
-- Determine the minimum counter size
constant SIZE : integer := integer(ceil(log2(real(DELAY))));
-- Convert (DELAY-1) to an unsigned bit vector
constant DELAY_U : unsigned((SIZE-1) downto 0) := to_unsigned((DELAY-1),SIZE);
-- The counter for the debouncer
signal DEBOUNCE_COUNTER : unsigned((SIZE-1) downto 0);
signal INPUT_REGISTER : std_logic_vector((NUM_SWITCHES-1) downto 0);
signal OUTPUT_REGISTER : std_logic_vector((NUM_SWITCHES-1) downto 0);
signal INT_NEW_SWITCH_STATE : std_logic;
-- The debouncer state machine
type DEBOUNCE_SM is (IDLE, DEBOUNCE_IN_PROGRESS);
signal DEBOUNCE_STATE : DEBOUNCE_SM;
begin
DEBOUNCE_INPUT_SWITCH_ARRAY : process(CLOCK) is
begin
  if rising_edge(CLOCK) then
    if (RESET = '1') then
      DEBOUNCE_COUNTER <= (others => '0');
    else
      if INT_NEW_SWITCH_STATE = '1' then
        DEBOUNCE_STATE <= DEBOUNCE_IN_PROGRESS;
        DEBOUNCE_COUNTER <= DEBOUNCE_COUNTER + 1;
      else
        DEBOUNCE_STATE <= IDLE;
      end if;
    end if;
  end if;
end process;
end Behavioral;
Sample VHDL Code for the Switch Debouncer Code

```vhdl
INPUT_REGISTER <= (others => '0');
OUTPUT_REGISTER <= (others => '0');
INT_NEW_SWITCH_STATE <= '0';
DEBOUNCE_STATE <= IDLE;
else
  case DEBOUNCE_STATE is
    when IDLE =>
      -- If a new value is detected on the switch array then it is
      -- registered and the DEBOUNCE_STATE goes to DEBOUNCE_IN_PROGRESS.
      -- The DEBOUNCE_COUNTER gets initialized to DELAY.
      if(INPUT_SWITCH_ARRAY /= INPUT_REGISTER) then
        INPUT_REGISTER <= INPUT_SWITCH_ARRAY;
        DEBOUNCE_STATE <= DEBOUNCE_IN_PROGRESS;
        DEBOUNCE_COUNTER <= DELAY_U;
      end if;
    when DEBOUNCE_IN_PROGRESS =>
      -- The value of the switch array must be checked to determine that
      -- it remains constant for [DELAY] clock cycles. Once this occurs
      -- the value is registered again and outputted while the
      -- NEW_SWITCH_STATE is set to 1. If the value of the switches
      -- changes before [DELAY] clock cycles then the DEBOUNCE_STATE
      -- returns to IDLE.
      if(INPUT_SWITCH_ARRAY = INPUT_REGISTER) then
        DEBOUNCE_COUNTER <= DEBOUNCE_COUNTER - 1;
        if(DEBOUNCE_COUNTER = 0) then
          OUTPUT_REGISTER <= INPUT_REGISTER;
          INT_NEW_SWITCH_STATE <= '1';
          DEBOUNCE_STATE <= IDLE;
        end if;
      end if;
    else
      DEBOUNCE_STATE <= IDLE;
    end case;
  end case;
  -- Clear NEW_SWITCH_STATE when ACKNOWLEDGE is asserted
  if((INT_NEW_SWITCH_STATE = '1') and (ACKNOWLEDGE = '1')) then
    INT_NEW_SWITCH_STATE <= '0';
  end if;
end if; -- if(RESET = '1')
end process DEBOUNCE_INPUT_SWITCH_ARRAY;
OUTPUT_SWITCH_ARRAY <= OUTPUT_REGISTER;
NEW_SWITCH_STATE <= INT_NEW_SWITCH_STATE;
end architecture;
```

6. Copy the `NUM SWITCHES` and `DELAY` generics, and
`INPUT SWITCH ARRAY` port from the `SWITCH DEBOUNCER CORE`
module to the generic and port sections of the `USER LOGIC I` module
respectively, as shown below.
7. Add the component declaration for the SWITCH_DEBOUNCER_CORE to the architecture section of the USER_LOGIC_I module as shown below.

```
architecture IMP of user_logic is

component SWITCH_DEBOUNCER_CORE
  generic
  (  
    NUM_SWITCHES : integer := 8; -- Width of input & output
    DELAY : integer := 8;
  );
  port
  (  
    RESET : in std_logic;
    CLOCK : in std_logic;
    INPUT_SWITCH_ARRAY : in std_logic_vector((NUM_SWITCHES-1) downto 0);
    OUTPUT_SWITCH_ARRAY : out std_logic_vector((NUM_SWITCHES-1) downto 0);
    NEW_SWITCH_STATE : out std_logic;
    ACKNOWLEDGE : in std_logic
  );
end component;
```

8. Add the signal declarations for OUTPUT_SWITCH_ARRAY and NEW_SWITCH_STATE to the signal declaration section of the USER_LOGIC_I module as shown below.

```
--USER signal declarations added here, as needed for user logic
signal OUTPUT_SWITCH_ARRAY : std_logic_vector((NUM_SWITCHES-1) downto 0);
signal NEW_SWITCH_STATE : std_logic;
```
9. Delete the signal declarations for `slv_reg0` and `intr_counter` in the USER_LOGIC_I module.

10. Add the component instantiation for the SWITCH_DEBOUNCER_CORE in the user logic section of the USER_LOGIC_I module as shown below. Please note the clock and reset signals from the PLB is connected to the SWITCH_DEBOUNCER_CORE component.

```
--USER logic implementation added here
SWITCH_DEBOUNCER_CORE_INSTANCE: SWITCH_DEBOUNCER_CORE
generic map
(
NUM_SWITCHES => NUM_SWITCHES,
DELAY => DELAY
)
port map
(
RESET => Bus2IP_Reset,
CLOCK => Bus2IP_Clk,
INPUT_SWITCH_ARRAY => INPUT_SWITCH_ARRAY,
OUTPUT_SWITCH_ARRAY => OUTPUT_SWITCH_ARRAY,
NEW_SWITCH_STATE => NEW_SWITCH_STATE,
ACKNOWLEDGE => NEW_SWITCH_STATE
);
```

11. Delete the SLAVE_REG_WRITE_PROC process from the USER_LOGIC_I module since we do not need to write to the switch debouncer peripheral. Remove `slv_reg0` from the SLAVE_REG_READ_PROC process sensitivity list and replace it with OUTPUT_SWITCH_ARRAY. Edit the SLAVE_REG_READ_PROC process such that it drives the OUTPUT_SWITCH_ARRAY onto slv_ip2bus_data when slv_reg_read_sel(0) is high as shown below.

```
-- implement slave model software accessible register(s) read mux
SLAVE_REG_READ_PROC : process (slv_reg_read_sel, OUTPUT_SWITCH_ARRAY) is
begin
  case slv_reg_read_sel is
    when "1" => slv_ip2bus_data(0 to (C_SLV_WIDTH-NUM_SWITCHES-1)) <= (others => '0');
    when others => slv_ip2bus_data(0 to (C_SLV_WIDTH-1)) <= OUTPUT_SWITCH_ARRAY;
  end case;
end process SLAVE_REG_READ_PROC;
```

12. Delete the example interrupt code in the USER_LOGIC_I module. In its place, add code to drive the NEW_SWITCH_STATE signal from the SWITCH_DEBOUNCER_CORE to the IP2Bus_IntrEvent(0) signal as shown below.

```
IP2Bus_IntrEvent(0) <= NEW_SWITCH_STATE;
```
13. Open the top-level SWITCH_DEBOUNCER module. Once again, take some time to study the structure of the code. Copy the `NUM_SWITCHES` and `DELAY` generics from the USER_LOGIC_I module to the generics section of the SWITCH_DEBOUNCER module. Copy the `INPUT_SWITCH_ARRAY` port from the USER_LOGIC_I module to the ports section of the SWITCH_DEBOUNCER module.

```vhdl
entity switch_debouncer is
    generic
    (
        -- ADD USER GENERICS BELOW THIS LINE -----------------------
        NUM_SWITCHES : integer := 0; -- Width of input & output
        DELAY : integer := 0;
        -- ADD USER GENERICS ABOVE THIS LINE -----------------------
    
        -- DO NOT EDIT BELOW THIS LINE -----------------------------
        -- Bus protocol parameters, do not add to or delete
        C_BASEADDR  : std_logic_vector := X"00000000"
        C_HIGHADDR  : std_logic_vector := X"00000000"
        C_SPLB_AWIDTH : integer := 32;
        C_SPLB_DWIDTH : integer := 128;
        C_SPLB_NUM_MASTERS : integer := 8;
        C_SPLB_MID_WIDTH : integer := 4;
        C_SPLB_NATIVE_DWIDTH : integer := 32;
        C_SPLB_P2P : integer := 0;
        C_SPLB_SUPPORT_BURST : integer := 0;
        C_SPLB_SMALLEST_MASTER : integer := 32;
        C_SPLB_CLK_PERIOD_US : integer := 10000;
        C_INCLUDE_DPHASE_TIMER : integer := 0;
        C_FAMILY : string := "virtex5"
        -- DO NOT EDIT ABOVE THIS LINE -----------------------------
    );
    port
    (
        -- ADD USER PORTS BELOW THIS LINE --------------------------
        INPUT_SWITCH_ARRAY : in std_logic_vector((NUM_SWITCHES-1) downto 0);
        -- ADD USER PORTS ABOVE THIS LINE --------------------------
    );
```

14. Scroll down to the bottom of the SWITCH_DEBOUNCER module to where the instantiation of the USER_LOGIC_I module is. Add code to map the `NUM_SWITCHES` and `DELAY` generics and the `INPUT_SWITCH_ARRAY` port as shown below.
15. At this point, it is a good idea to synthesize the design and fix any errors.

-Importing the Changes to the Peripheral

1. From EDK, go to the Hardware menu and click on Create or Import Peripheral... Click Next at the Welcome screen for the Create and Import Peripheral Wizard.

2. At the Peripheral Flow screen, click the radio button for Import existing peripheral and click Next.
3. At the **Repository or Project** screen, select the directory where the switch debouncer peripheral is stored.

4. At the **Name and Version** screen, set the name to `switch_debouncer` and the version number to **1.00.a**. Click **Next**. When the **Overwrite Existing Peripheral** popup screen appears, click **Yes**.

5. At the **Source File Types** screen, select the **HDL source files** option. Note that you can attach documentation to your peripheral. This is good thing to do if you are going to redistribute a peripheral.
6. At the **HDL Source Files** screen, select the option for **Use data (*.mpd)** collected during a previous invocation of this tool and make sure the radio button for **Use an XST project file (*.prj)** is selected. Set the data file to `switch_debouncer_v2_1_0.mpd` from the `\pcores\switch_debouncer_v1_0\data\` directory. Set the XST project file to `switch_debouncer.prj` from the `\pcores\switch_debouncer_v1_0\dev\projnav\` directory. Click **Next**.

7. The **HDL Analysis Information** screen shows all the HDL source files that comprise the peripheral. Click **Next**.
8. Make sure that the **PLBV46 Slave (SPLB)** option is the only select option at the **Bus Interface** screen and click **Next**.

9. Click **Next** at the **SPLB : Port** screen.

10. Click **Next** at the **SPLB : Parameters** screen.

11. Click **Next** at the **Identify Interrupt Signal** screen.

12. Click **Next** at the **Parameters Attributes** screen. Please note the **NUM SWITCHES** and **DELAY** parameters that we added to the peripheral.

13. Click **Next** at the **Port Attributes** screen. Please note the **INPUT SWITCH ARRAY** port that we added to the peripheral.
14. Click **Finish** at the **Congratulations** screen.
-Adding the Peripheral to an EDK Project

1. From EDK, click on the IP Catalog tab in the Project Information Area pane. This tab shows all the peripherals that are available to add to the project. If you stored the peripheral in a repository then it will be listed under the Project Peripheral Repository item. If you stored the peripheral in your EDK project directory then it will be listed under the Project Local pcores item. Expand the appropriate item. Right-click the SWITCH_DEBOUNCE peripheral and select Add IP. You should notice the new peripheral in the Bus Interfaces tab in the main pane with the instance name of switch_debouncer_0.

2. If necessary, click on the Bus Interfaces tab in the main pane. Expand the switch_debouncer_0 peripheral. Select plb from the Bus Connection drop-down list.
3. Right-click the **switch_debouncer_0** peripheral and select **Configure IP**. Set the **NUM_SWITCHES** parameter to 5 in the **All** pane of the **switch_debouncer_v1_00_a** screen. Click **OK**.

4. Click on the **Ports** tab in the main pane. Expand the **switch_debouncer_0** peripheral. Select **New Connection** under the **Net** drop-down list for the **IP2INTC_Irpt** port.

5. Select **Make External** under the **Net** drop-down list for the **INPUT_SWITCH_ARRAY** port. This allows the **INPUT_SWITCH_ARRAY** port to be connected to external pins on the FPGA. If you expand the **External Ports** item then you will notice **INPUT_SWITCH_ARRAY** port listed there with a range of [4:0].
6. Expand the **xps_intc_0** interrupt controller peripheral. Click on the **Net** field for the **Intr** port. This will bring up the **Interrupt Connection Dialog** screen. Select the **switch_debouncer_0_IP2INTC_Irpt** port under the list of **Potential Interrupt Connections** and click the **+** button to add it to the list of **Connected Interrupts**. Click **OK**.

7. Click on the **Addresses** tab in the main pane. Select the **switch_debouncer_0** peripheral. Click on the **Base Address** field and change it to **0xC0000000**.
8. Click on the **Project** tab in the **Project Information Area** pane and double-click on the `system.ucf` file under **Project Files**. Comment all of the constraints for the **Push.Buttons_Position** peripheral.

```plaintext
### Module PushButtons_5Bit constraints

# Net fpga_0_PushButtons_5Bit_GPIO_IO_pin<0> LOC=AG5;
# Net fpga_0_PushButtons_5Bit_GPIO_IO_pin<1> IOSTANDARD = LVTTL;
# Net fpga_0_PushButtons_5Bit_GPIO_IO_pin<2> LOC=AH4;
# Net fpga_0_PushButtons_5Bit_GPIO_IO_pin<3> IOSTANDARD = LVTTL;
# Net fpga_0_PushButtons_5Bit_GPIO_IO_pin<4> LOC=AG3;
# Net fpga_0_PushButtons_5Bit_GPIO_IO_pin<5> IOSTANDARD = LVTTL;
# Net fpga_0_PushButtons_5Bit_GPIO_IO_pin<6> LOC=AH1;
# Net fpga_0_PushButtons_5Bit_GPIO_IO_pin<7> IOSTANDARD = LVTTL;
# Net fpga_0_PushButtons_5Bit_GPIO_IO_pin<8> LOC=AH2;
# Net fpga_0_PushButtons_5Bit_GPIO_IO_pin<9> IOSTANDARD = LVTTL;
```

9. Add the following constraints to the `system.ucf` file. These constraints connect the `INPUT_SWITCH ARRAY` port to the buttons on the V2Pro.

**Constraints for the switch_debouncer_0 peripheral**

```plaintext
### Module switch_debouncer_0 constraints

Net switch_debouncer_0_INPUT_SWITCH_ARRAY_pin<0> LOC=AH4; # UP
Net switch_debouncer_0_INPUT_SWITCH_ARRAY_pin<1> LOC=AH2; # RIGHT
Net switch_debouncer_0_INPUT_SWITCH_ARRAY_pin<2> LOC=AG3; # DOWN
Net switch_debouncer_0_INPUT_SWITCH_ARRAY_pin<3> LOC=AH1; # LEFT
Net switch_debouncer_0_INPUT_SWITCH_ARRAY_pin<4> LOC=AG5; # ENTER
Net switch_debouncer_0_INPUT_SWITCH_ARRAY_pin<*> IOSTANDARD = LVTTL;
```

10. Finally, build the project by going to the **Hardware** menu and selecting **Generate Bitstream**. Correct any errors that are found.
1. From EDK, click on the Applications tab in the Project Information Area pane. Double-click on Add Software Application Project... Type Test_Switch_Debouncer in the Project Name field of the Add Software Application Project screen. Click OK.

2. Right-click Sources under Project: Test_Switch_Debouncer in the Applications pane. Select Add New File... If necessary, browse to the Test_Switch_Debouncer\src\ folder in your project. Type Test_Switch_Debouncer.c in the File name field. Click Save.
3. Open the Test_Switch_Debouncer.c file under the Sources item in the Test_Switch_Debouncer application project. Copy in the code below and save the file.

Sample C Code for testing the Switch Deboouncer Peripheral

```c
#include "xparameters.h" // System definitions
#include "stdio.h"
#include "xbasic_types.h"
// Low-level driver to access the GPIO peripherals
#include "xgpio_l.h"
// Low-level driver to configure the Interrupt Controller
#include "xintc_l.h"
// Low-level driver to access the switch_debouncer peripheral
#include "switch_debouncer.h"

volatile Xuint8 interrupt_flag;
volatile Xuint32 button_state;

// Interrupt handler for the Switch Deboouncer
// This handler clears the interrupt flag and increments the 4-bit LEDs
void Switch_Debouncer_Interrupt_Handler (void * baseaddr_p)
{
    Xuint32 baseaddr = (Xuint32) baseaddr_p;
    Xuint32 Status;
    static Xuint8 led_counter = 0;
    // Read status from Interrupt Status Register.
    Status = SWITCH_DEBOUNCER_mReadReg(baseaddr, SWITCH_DEBOUNCER_INTR_IPISR_OFFSET);
    // Write back status to Interrupt Status Register to clear interrupt
    ```
Sample C Code for testing the Switch Debouncer Peripheral

```c
SWITCH_DEBOUNCER_mWriteReg(baseaddr,
SWITCH_DEBOUNCER_INTR_IPISR_OFFSET, Status);

int main()
{
    interrupt_flag = 0;
    print("Welcome to the Test_Switch_Debouncer program\r\n\n");
    // Configure GPIO for 4-bit LEDs as output
    XGpio_mSetDataDirection(XPAR_LEDS_4BIT_BASEADDR, 1, 0x00);
    // Turn off 4-Bit LEDs
    XGpio_mSetDataReg(XPAR_LEDS_4BIT_BASEADDR, 1, 0x00);
    // Initialize Interrupts on PowerPC
    XExc_Init();
    // Register the interrupt handler of the XPS Interrupt Controller
    // with the PowerPC's external interrupt.
    XExc_RegisterHandler(XEXC_ID_NON_CRITICAL_INT, (XExceptionHandler)XIntc_DeviceInterruptHandler,
    (void *) XPAR_INTC_0_DEVICE_ID);
    // Register the Switch Debouncer interrupt handler in
    // the vector table of the XPS Interrupt Controller
    XIntc_RegisterHandler(XPAR_INTC_0_BASEADDR,
    XPAR_XPS_INTC_0_SWITCH_DEBOUNCER_0_IP2INTC_IRPT_INTR, (XInterruptHandler)Switch_Debouncer_Interrupt_Handler,
    (void *) XPAR_SWITCH_DEBOUNCER_0_BASEADDR);
    // Start the XPS Interrupt Controller
    XIntc_mMasterEnable(XPAR_INTC_0_BASEADDR);
    // Enable Switch Debouncer interrupt requests in the XPS Interrupt Controller
    XIntc_mEnableIntr(XPAR_INTC_0_BASEADDR,
    XPAR_SWITCH_DEBOUNCER_0_IP2INTC_IRPT_MASK);
    // Local Interrupt enable for the Switch Debouncer peripheral
    SWITCH_DEBOUNCER_mWriteReg(XPAR_SWITCH_DEBOUNCER_0_BASEADDR,
    SWITCH_DEBOUNCER_INTR_IPISR_OFFSET, 1);
    // Global interrupt enable for the Switch Debouncer peripheral
    SWITCH_DEBOUNCER_mWriteReg(XPAR_SWITCH_DEBOUNCER_0_BASEADDR,
    SWITCH_DEBOUNCER_INTR_DGIER_OFFSET, INTR_GIE_MASK);
    // Enable PowerPC non-critical (external) interrupts
    XExc_mEnableExceptions(XEXC_NON_CRITICAL);
    // Wait for the state of the Buttons to change
    while(1)
    {
        // When the state of the buttons changes...
        // Print which button is pressed (one button at a time)
```
if(interrupt_flag)
{
    
    **Sample C Code for testing the Switch Debouncer Peripheral**

    switch(button_state)
    {
        case 0x1E:
            print("North Button Pressed\r\n");
            break;
        case 0x1D:
            print("East Button Pressed\r\n");
            break;
        case 0x1B:
            print("South Button Pressed\r\n");
            break;
        case 0x17:
            print("West Button Pressed\r\n");
            break;
        case 0x0F:
            print("Center Button Pressed\r\n");
            break;
    }
    interrupt_flag = 0; // Clear the interrupt flag
}

4. Right-click on **Project: Test_Switch_Debouncer** in the **Applications** pane. Select **Generate Linker Script**. Keep all of the options in the **Generate Linker Script** screen except change the **Output Linker Script** field to **Test_Switch_Debouncer_linker_script.ld**. Click **OK**.
### Sections View:

<table>
<thead>
<tr>
<th>Section</th>
<th>Size (Bytes)</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>vectors</td>
<td></td>
<td>xps_bram_if_cnt1</td>
</tr>
<tr>
<td>test</td>
<td></td>
<td>xps_bram_if_cnt1</td>
</tr>
<tr>
<td>rodata</td>
<td></td>
<td>xps_bram_if_cnt1</td>
</tr>
<tr>
<td>rodata1</td>
<td></td>
<td>xps_bram_if_cnt1</td>
</tr>
<tr>
<td>sdata2</td>
<td></td>
<td>xps_bram_if_cnt1</td>
</tr>
<tr>
<td>data</td>
<td></td>
<td>xps_bram_if_cnt1</td>
</tr>
<tr>
<td>data1</td>
<td></td>
<td>xps_bram_if_cnt1</td>
</tr>
<tr>
<td>files</td>
<td></td>
<td>xps_bram_if_cnt1</td>
</tr>
<tr>
<td>sdata</td>
<td></td>
<td>xps_bram_if_cnt1</td>
</tr>
<tr>
<td>sbss</td>
<td></td>
<td>xps_bram_if_cnt1</td>
</tr>
<tr>
<td>bss</td>
<td></td>
<td>xps_bram_if_cnt1</td>
</tr>
</tbody>
</table>

### Heap and Stack View:

<table>
<thead>
<tr>
<th>Section</th>
<th>Size (Bytes)</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heap</td>
<td>0x400</td>
<td>xps_bram_if_cnt1</td>
</tr>
<tr>
<td>Stack</td>
<td>0x400</td>
<td>xps_bram_if_cnt1</td>
</tr>
</tbody>
</table>

### Memory View:

<table>
<thead>
<tr>
<th>Memory</th>
<th>Start Address</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>xps_bram_if_cnt1</td>
<td>0xFFFFF0000</td>
<td>64K</td>
</tr>
</tbody>
</table>

### ELF file used to populate section information:

C:\ECE595_SPRING_2009\Tutorial_2\Test_Switch_Debouncer\executable.elf

### Output Linker Script:

Test_Switch_Debouncer linker script
5. Right-click the Test_Switch_Debouncer project in the Applications tab of the Project Information Area pane. Select Mark to Initialize BRAMs and then select Build Project.

6. Go to the Device Configuration menu and select Update Bitstream.

7. Turn on your V2Pro board and connect the USB cable and serial cable from your computer. Go to the Device Configuration menu and select Download Bitstream.
8. You should notice text printed to a terminal application similar to that shown below when you press the buttons on the V2Pro. Additionally, the 4-LEDs will count the number of state changes on the buttons.

```
Welcome to the Test_Switch_Debouncer program
Center Button Pressed
South Button Pressed
South Button Pressed
West Button Pressed
East Button Pressed
North Button Pressed
Center Button Pressed
West Button Pressed
South Button Pressed
Center Button Pressed
East Button Pressed
North Button Pressed
West Button Pressed
```

-Author Bio

Name: Vallabh Srikanth Devarapalli  
Graduate student at UNM in ECE department.  
E-mail: vsdevar@unm.edu.

We thank Steve Wasson for authoring the tutorial from which this present tutorial has been derived.