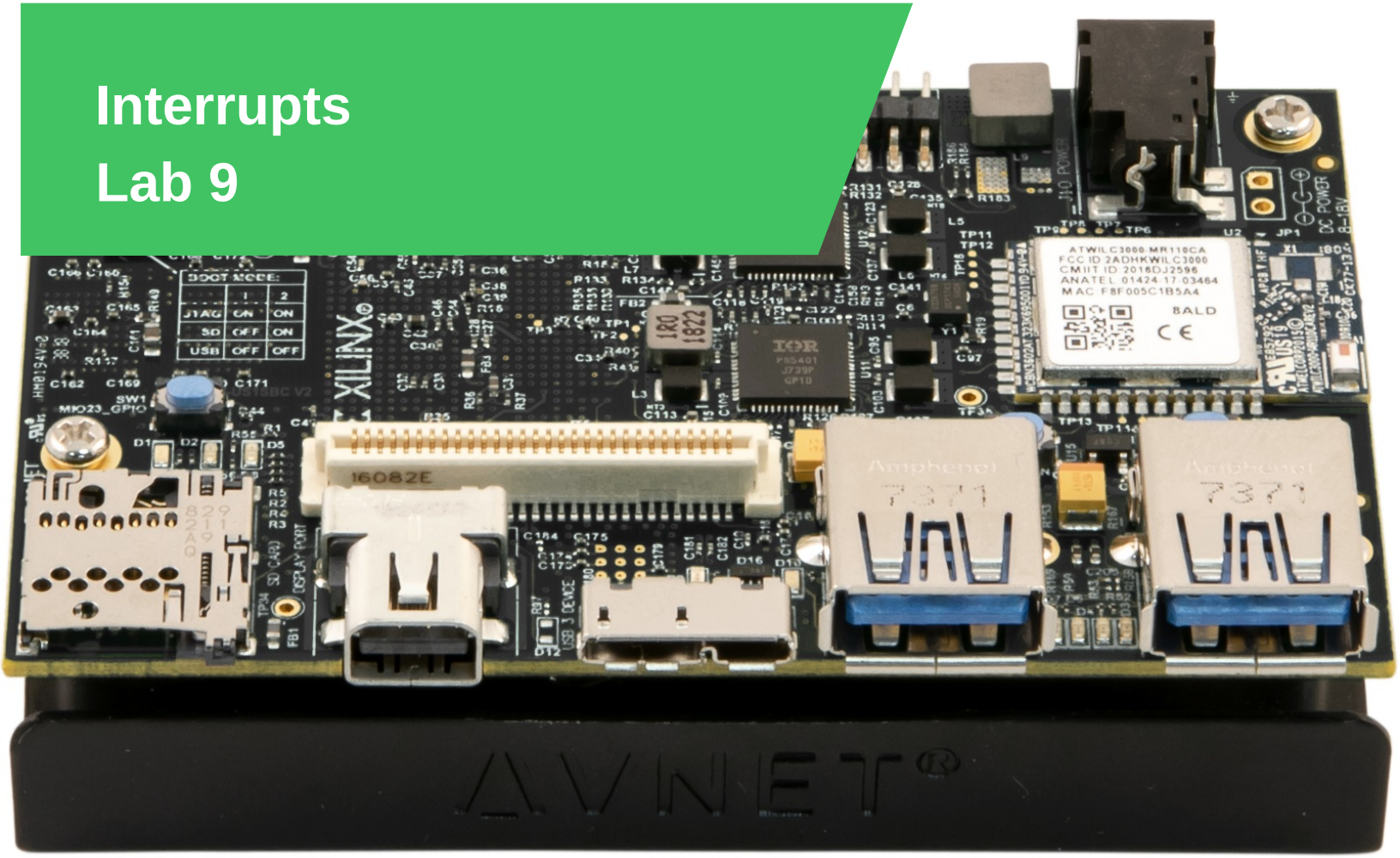
Avnet Technical Training Course

Interrupts
Lab 9



Tools:	2019.1
Training Version:	v12
Date:	July 2019

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Lab 09 Overview

An interrupt handler, also known as an Interrupt Service Routine (ISR), is a callback subroutine in microcontroller firmware, operating system, or device driver whose execution is triggered by the reception of a hardware interrupt. Interrupt handlers have a multitude of functions, which vary based on the reason the interrupt was generated and the speed at which the interrupt handler completes its task.

An interrupt handler is a low-level counterpart of event handlers. These handlers are typically initiated by hardware interrupts, and are used for servicing hardware devices and transitions between protected modes of operation such as system calls.

In this lab we will perform steps to import a known good code base to access a piece of custom hardware built into the system. From that point, we will modify the application code to respond to an interrupt generated by the custom hardware platform when an incorrect PWM value is written to the PWM controller.

Lab 09 Objectives

When you have completed Lab 09, you will know:

- How to enable the interrupt subsystem to allow hardware interrupts to interrupt software execution
- Create an interrupt service routine to handle the hardware interrupt
- The following is a software exercise in which we will enable LED D8 and change its brightness showcasing an interrupt

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Experiment 1: Create the Interrupt Application Project

Similar to the flow for importing existing application code in Lab 8, a new blank project is created and existing code is imported. The application code is then executed to verify the application code we were given functions as expected.

Experiment 1 General Instruction:

Create a new blank software application project. Import code from the following folder:

/home/training/AvnetTTC/ZynqSW/2017_4/Support_documents/LED_Dimmer/

Run the application on the target hardware and observe the behavior.

Experiment 1 Step-by-Step Instructions:

1. Create a new SDK software application project by selecting the **File→New→ Application Project** menu item.

2. In the **New Project** wizard, change the **Project name** field to the **LED_Dimmer_w_Int** name.

   Change the **Board Support Package** to the **Use existing** option and use the drop down menu to select the existing BSP **standalone_bsp_0** option.

   Leave the other settings to their default values. Click the **Next** button to continue.

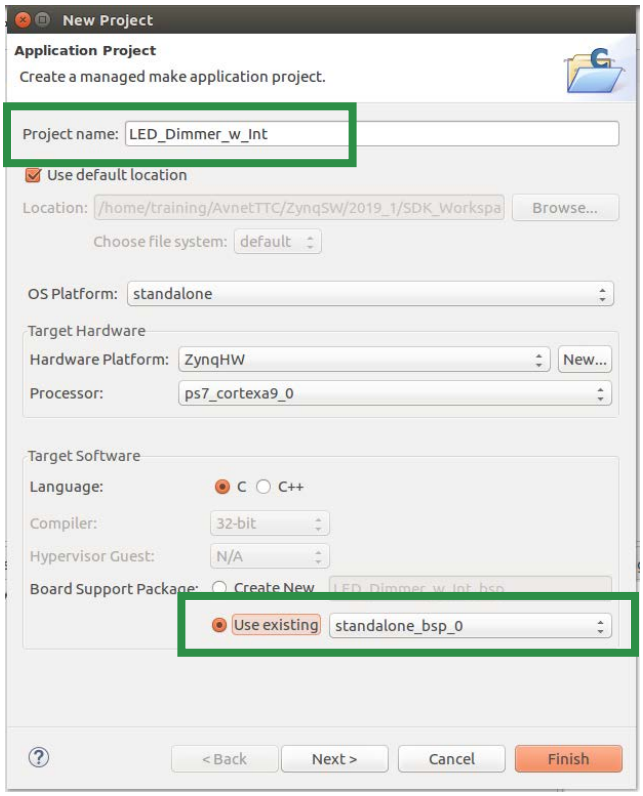


Figure 1 – Creating the LED_Dimmer_w_Int Application

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

3. Select the **Empty Application** project template and click the **Finish** button to complete the new project creation process using the **Empty Application** project template.

4. The empty **LED_Dimmer_w_Int** application project is created.

   In the **Project Explorer** tab, expand **LED_Dimmer_w_Int** and right-click on the **src** folder.

   Click on the **Import** option in the pop up menu.

5. In the **Import** window, expand the **General** item, select the **File System** option, and click the **Next** button.

6. Click on the **Browse** button and select the following folder which contains the application code that we wish to start from:

   /home/training/AvnetTTC/ZynqSW/2019_1/Support_documents/LED_Dimmer_w_Int

   After this folder is selected within the **Browse** dialog, click the **OK** button to search the folder for files to import into the application project.

   Select the **main.c** file and then click the **Finish** button to complete the **Import** operation.

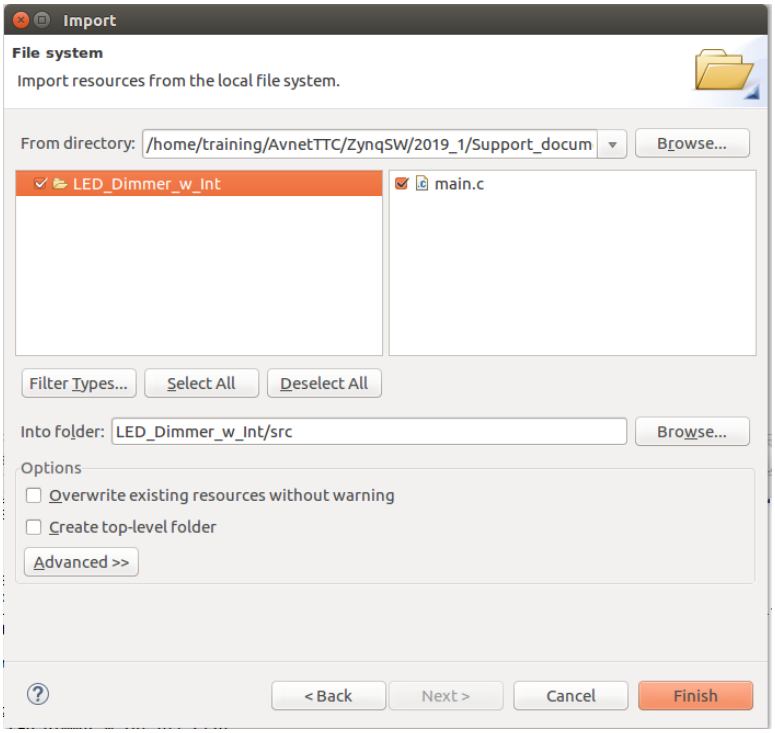


Figure 2 – Selecting Application Source Files

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

The SDK Console panel shows the results of the build. Make sure that the application is built without errors.

```
CDT Build Console [LED_Dimmer_w_Int]
Invoking: ARM v7 gcc compiler
arm-none-eabi-gcc -Wall -O0 -g3 -c -fmessage-length=0 -MT"src/main.o" -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard -I../../standalone_bsp_0/ps7_c
Finished building: ../src/main.c

Building target: LED_Dimmer_w_Int.elf
Invoking: ARM v7 gcc linker
arm-none-eabi-gcc -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard -Wl,-build-id=none -specs=Xilinx.spec -Wl,-T -Wl,../src/lscript.ld -L../../standal
Finished building target: LED_Dimmer_w_Int.elf

Invoking: ARM v7 Print Size
arm-none-eabi-size LED_Dimmer_w_Int.elf  |tee "LED_Dimmer_w_Int.elf.size"
   text    data     bss     dec     hex filename
  29556    1544   23256   54356    d454 LED_Dimmer_w_Int.elf
Finished building: LED_Dimmer_w_Int.elf.size

15:44:01 Build Finished (took 589ms)
```

Figure 3 – Application Build Console Window

7. After SDK finishes compiling the new application code, the ELF is available in the following location:

   /home/training/AvnetTTC/ZynqSW12019_1/SDK_Workspace/LED_Dimmer_w_Int/Debug/LED_Dimmer_w_Int.elf

   At this point the application is ready to be launched on the target hardware.

8. Set the Boot Mode switch to JTAG Mode

   a. MiniZed

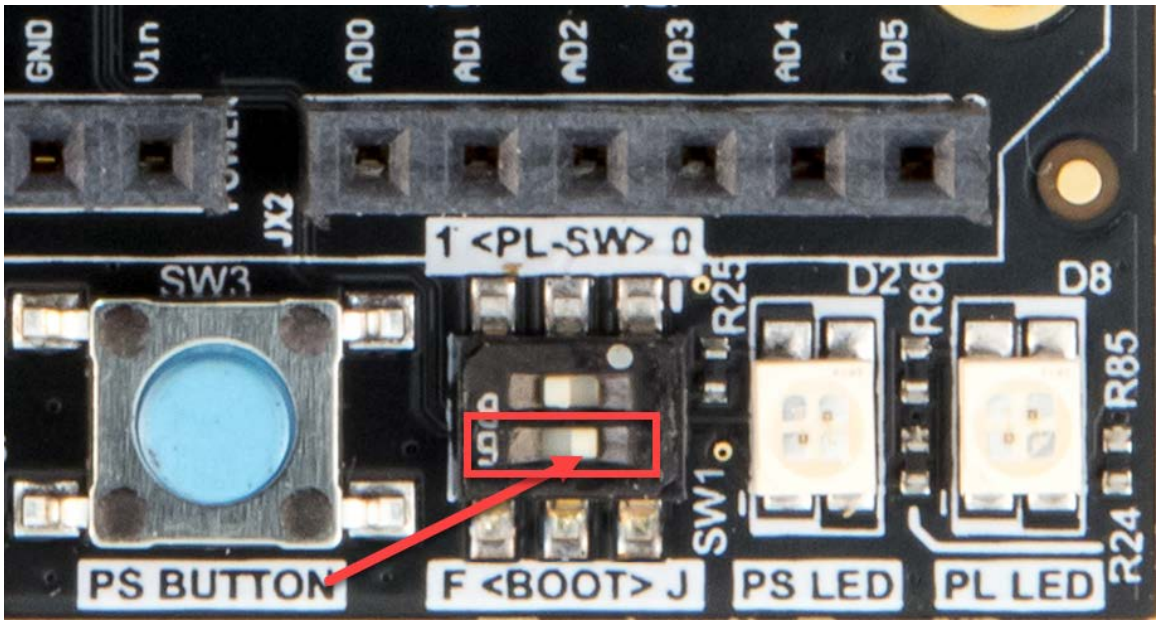


Figure 4 – MiniZed – JTAG Boot

9. Connect to JTAG as follows:

   a. MiniZed – Connect microUSB to MiniZed USB-UART-JTAG (J2) port and your Host machine.

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

10. In SDK, select **Xilinx Tools → Program FPGA** or click the icon.

11. SDK will already know the correct .bit file since this was imported with the hardware platform. Click the **Program** button.

    When D3 lights blue, the PL has configured successfully. Look for the message "`FPGA configured successfully with bitstream`" in the SDK Log window.

12. Launch a terminal program (GTK Term) with the 115200/8/n/1/n settings.

13. In the **Project Explorer** tab, right-click on the **LED_Dimmer_w_Int** project folder.

    Click on the **Run As→ Launch on Hardware (System Debugger)** option in the pop up menu.

14. Using the Launch on Hardware option will automatically create a run configuration using default settings and launch the application on the target platform. Once the application is launched and is running on the target hardware, a prompt should appear in the terminal window asking the user to **Select a Brightness between 0 and 9**.

15. **If you are using the built in SDK terminal you will see odd character returns. – This is due to the nature of the SDK Terminal. When sending a character you first type it in and then click the send or return key on your keyboard. Your application first see the character you entered and then it sees the return key character you entered. This can all be viewed with breakpoints. – Suggested work around it to use GTK Term.**

16. Experiment with various inputs at the terminal to determine if the existing software application is sufficient for a user to assign a new brightness value to the PWM block. Please take a look at LED D8 on your MiniZed while doing so.

    One good check to make is to see how the application responds to an invalid input. Since we cannot enter more than one character at a time and the application is looking for an ASCII value from 0x30 to 0x39 to be entered, try entering in one of the alphabetical keys from A-Z to trigger an input outside of the valid range. Does the application respond the way you think it should to invalid input?

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

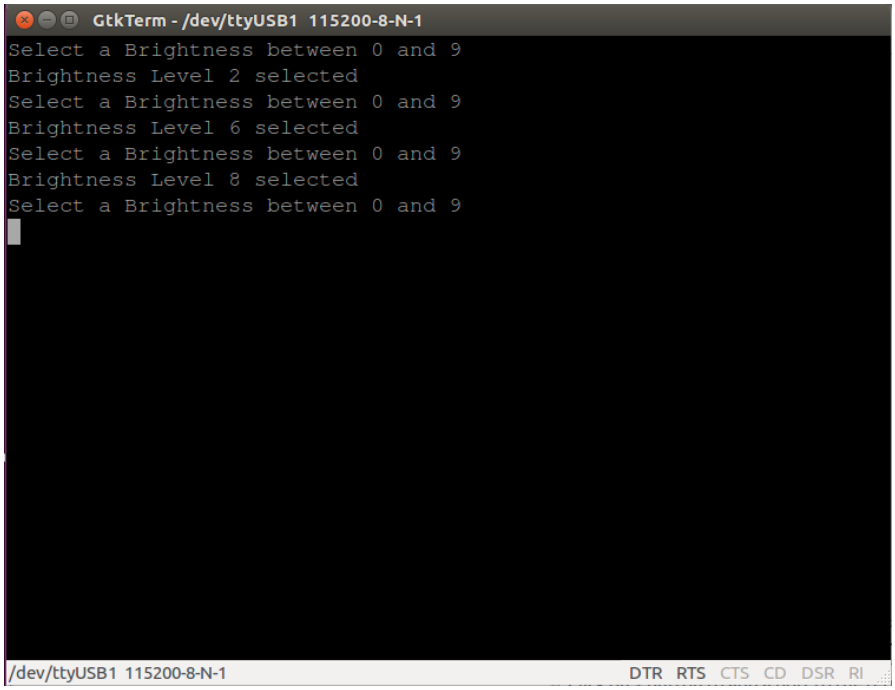


Figure 5 – Terminal Output from LED_Dimmer_w_Int Application

Questions:

Answer the following questions:

- *What happens when a value outside of the requested range of 0-9 is entered?*

_____

- *Are interrupts currently enabled in the example application provided?*

_____

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Experiment 2: Add Interrupt Support to the Application

You may have noticed that the application code that we have been given does not do a very good job of handling incorrect values being passed in. In fact, it blindly passes incorrect values on to the PWM hardware block. This problem could be solved in software by first checking the values before writing the value to the PWM hardware block.

What if the application requires us to pass a value that could be invalid to the PWM hardware block?

In this experiment, we will enable the GIC to pass the PWM hardware block interrupt to the processor and interrupt execution flow. We will also add an Interrupt Service Routine (ISR) which will handle the interrupt from the PWM hardware block and assign a known ***safe*** value to the PWM hardware block before continuing application execution.

This is accomplished by the use of the BSP hardware driver **scugic** API. The following API calls will be used to add interrupt support:

```
int    XScuGic_CfgInitialize
       (XScuGic *InstancePtr,
       XScuGic_Config *ConfigPtr,
       u32 EffectiveAddr)
```

- CfgInitialize a specific interrupt controller instance/driver.
- The function initializes the field of XscuGic structure and the initial vector table with stub function calls.
- All interrupt sources are disabled when initialization occurs**.**

```
int    XScuGic_Connect
       (XScuGic *InstancePtr,
       u32 Int_Id,
       Xil_InterruptHandler Handler,
       void *CallBackRef)
```

- Makes the connection between the Int_Id of the interrupt source and the associated handler that is to run when the interrupt is recognized.

```
void XScuGic_Enable
       (XScuGic *InstancePtr,
       u32 Int_Id)
```

- Enables the interrupt source provided as the argument Int_Id.
- Any pending interrupt condition for the specified Int_Id will occur after this function is called.

```
void XScuGic_SetPriorityTriggerType
       (XScuGic *InstancePtr,
       u32 Int_Id,
       u8 Priority,
       u8 Trigger)
```

- Sets the interrupt priority and trigger type for the specified IRQ source.

```
XScuGic_Config *  XScuGic_LookupConfig (u16 DeviceId)
```

- Looks up the device configuration based on the unique device ID.
- A table contains the configuration info for each device in the system.

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

void XScuGic_InterruptHandler (XScuGic *InstancePtr)

- This function is the primary interrupt handler for the driver.
- It must be connected to the interrupt source such that it is called when an interrupt of the interrupt controller is active.
- It will resolve which interrupts are active and enabled and call the appropriate interrupt handler.

The following API calls are not used in our application but could be useful for use in a future application:

```
void                                                    XScuGic_Disable
     (XScuGic                                             *InstancePtr,
     u32 Int_Id)
```

- Disables the interrupt source provided as the argument Int_Id.

```
void                                          XScuGic_GetPriorityTriggerType
     (XScuGic                                                 *InstancePtr,
     u32                                                            Int_Id,
     u8                                                           *Priority,
     u8 *Trigger)
```

- Gets the interrupt priority and trigger type for the specified IRQ source.

int XScuGic_SelfTest (XScuGic *InstancePtr)

- Runs a self-test on the driver and the associated hardware device.
- This function clears all of the interrupt enable bits.
- All interrupts will be disabled when this function is complete.

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Experiment 2 General Instruction:

Open the **LED_Dimmer_w_Int** application source code. Create an interrupt service routine (ISR) to handle interrupts from the PWM hardware block. Add application code to enable interrupts and register the ISR.

Experiment 2 Step-by-Step Instructions:

1. Expand the **LED_Dimmer_w_Int → src** folder in the **Project Explorer** panel. Double click the application source file **main.c** to open it in the code editor.

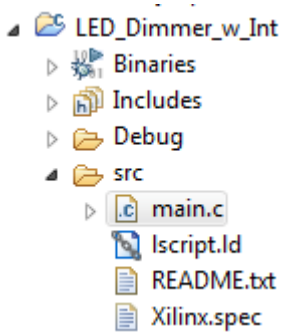


Figure 6 – LED_Dimmer_w_Int Source Code

2. If user feels comfortable with skipping the section were we modify code to enable the interrupt in C, please delete the current main.c file and reimport the main.c file from C:\Speedway\ZynqSW\2017_4\Support_documents\LED_Dimmer_w_Int into the LED_Dimmer_w_int SRC folder

3. In the source file we need to include the drivers for the **SCUGIC** and the **Exception Handler**. Place these lines below the other **#include** statements by inserting near line 49.

```
#include                                                    "xscugic.h"
#include                                                  "xil_exception.h"
```

4. Copy these definitions into the source code by inserting them near line 59 just below the existing definition for **PWM_BASE_ADDRESS**. These are static definitions that must be added to map our interrupt setup routine to the **SCUGIC** driver calls.

```
/* The following definitions are related to handling interrupts from the
 * PWM controller. */
#define XPAR_PS7_SCUGIC_0_DEVICE_ID    0
#define INTC_PWM_INTERRUPT_ID          XPAR_FABRIC_PWM_W_INT_0_INTERRUPT_OUT_INTR
#define INTC                           XScuGic
#define INTC_HANDLER                   XScuGic_InterruptHandler
#define  INTC_DEVICE_ID                                  XPAR_PS7_SCUGIC_0_DEVICE_ID
```

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

5. Copy these variable declarations into the source code variable declarations section by inserting them near line 72 just within the **Variable Definitions** section. The **brightness** variable is now declared globally to make it visible to the ISR that will be added later. The **Intc** variable is a static definition used to setup the **SCUGIC** driver.

```
/* LED brightness level is now global to make is visible to the ISR. */
volatile u32 brightness;
/* The Instance of the Interrupt Controller Driver */
static INTC Intc;
```

6. Add this ISR into the source code following the variable declarations section by inserting it at line 77 just below the variable declarations inserted during Step 5 above. This will serve as the callback function that is called when the hardware interrupt is serviced.

```
void PWMIsr(void *InstancePtr)
{
    /* Inform the user that an invalid value was detected by the PWM
     * controller. */
    print("PWM Value exceeded, brightness reset to zero. Enter new value: \r\n");

    /* Set the brightness value to a safe value and write it to the
     * PWM controller in order to clear the pending interrupt. */
    brightness = 0;
    Xil_Out32(PWM_BASE_ADDRESS, brightness);
}
```

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

7. Add this setup function into the source code following the ISR section by inserting it near line 89 just below the ISR code inserted during Step 5 above. This code provides the setup needed to enable the PWM interrupt to be recognized and attaches the **PWMIsr()** as the interrupt handler.

```
/*****************************************************************************/
/**
* This function sets up the interrupt system for the PWM dimmer controller.
* The processing contained in this function assumes the hardware system was
* built with an interrupt controller.
*
* @param       None.
*
* @return      A status indicating XST_SUCCESS or a value that is contained in
*              xstatus.h.
*
* @note        None.
*
*****************************************************************************/
int SetupInterruptSystem()
{
    int result;
    INTC *IntcInstancePtr = &Intc;

    XScuGic_Config *IntcConfig;

    /* Initialize the interrupt controller driver so that it is ready to
     * use. */
    IntcConfig = XScuGic_LookupConfig(INTC_DEVICE_ID);
    if (IntcConfig == NULL)
    {
        return XST_FAILURE;
    }

    /* Initialize the SCU and GIC to enable the desired interrupt
     * configuration. */
    result = XScuGic_CfgInitialize(IntcInstancePtr, IntcConfig,
                                   IntcConfig->CpuBaseAddress);
    if (result != XST_SUCCESS)
    {
        return XST_FAILURE;
    }

    XScuGic_SetPriorityTriggerType(IntcInstancePtr, INTC_PWM_INTERRUPT_ID,
                                   0xA0, 0x3);

    /* Connect the interrupt handler that will be called when an
     * interrupt occurs for the device. */
    result = XScuGic_Connect(IntcInstancePtr, INTC_PWM_INTERRUPT_ID,
                             (Xil_ExceptionHandler) PWMIsr, 0);
    if (result != XST_SUCCESS)
    {
        return result;
    }

    /* Enable the interrupt for the PWM controller device. */
    XScuGic_Enable(IntcInstancePtr, INTC_PWM_INTERRUPT_ID);

    /* Initialize the exception table and register the interrupt controller
```

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

```
     * handler with the exception table. */
    Xil_ExceptionInit();
    Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
                                 (Xil_ExceptionHandler)INTC_HANDLER, IntcInstancePtr);

    /* Enable non-critical exceptions */
    Xil_ExceptionEnable();

    return XST_SUCCESS;
}
```

8. Replace the **brightness** variable declaration near line 160 with this assignment to initialize **brightness** to a zero value(u32 brightness = 0;" to just "brightness = 0). This variable is already declared globally so declaring it again in main would cause a compiler error.

```
    brightness = 0;
```

9. Add this call to **SetupInterruptSystem()** during our initialization steps within **main()** by inserting it near line 164 just below the following code:

```
Xil_Out32(PWM_BASE_ADDRESS, 0);
```

   This call will enable interrupts and register the ISR before any inputs from the user are processed.

```
    /* Setup the interrupts such that interrupt processing can occur.  If an
     * error occurs while setting up interrupts, then exit the application. */
    status = SetupInterruptSystem();
    if (status != XST_SUCCESS)
    {
        return XST_FAILURE;
    }
```

10. Save the updated source file. SDK will detect the source code change and automatically rebuild the application project and there should be no errors reported in the SDK console.

```
Finished building: ../src/main.c

Building target: LED_Dimmer_w_Int.elf
Invoking: ARM v7 gcc linker
arm-none-eabi-gcc -mcpu=cortex-a9 -mfpu=vfpv3 -mfloat-abi=hard -Wl,-build-id=none
Finished building target: LED_Dimmer_w_Int.elf

Invoking: ARM v7 Print Size
arm-none-eabi-size LED_Dimmer_w_Int.elf  |tee "LED_Dimmer_w_Int.elf.size"
   text    data     bss     dec     hex filename
  33116    2320   23288   58724    e564 LED_Dimmer_w_Int.elf
Finished building: LED_Dimmer_w_Int.elf.size


15:50:52 Build Finished (took 411ms)
```

Figure 7 – Rebuilding the Updated LED_Dimmer_w_Int Application

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

10. In the **Project Explorer** tab, right-click on the **LED_Dimmer_w_Int** project folder.

    Click on the **Run As→ Launch on Hardware (System Debugger)** option in the pop up menu.

11. If the application is still running on the target hardware from the previous experiment, a Processor in use warning will appear. Click on the **OK** button to terminate the previous launch and run the updated application on the target hardware.

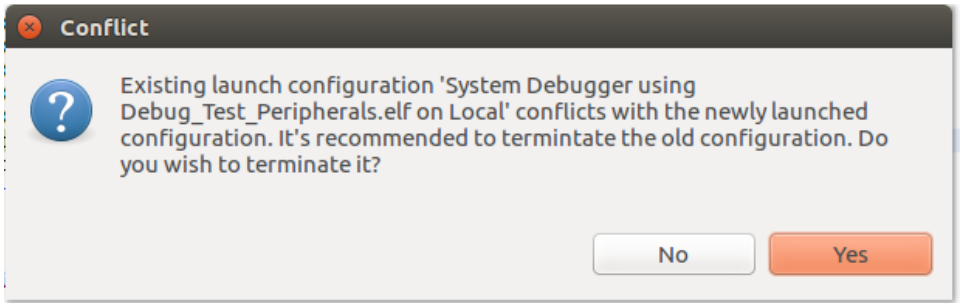


Figure 8 – Terminate Previous Launch

12. Once the application is launched and is running on the target hardware, a prompt should appear in the terminal window asking the user to **Select a Brightness between 0 and 9**.

    Experiment with various inputs at the terminal to determine if the existing software application is any better suited for a user to assign a new brightness value to the PWM block. Please keep an eye on LED D8 on your MiniZed

    One good check to make is to see how the application responds to an invalid input. Since we cannot enter more than one character at a time and the application is looking for an ASCII value from 0x30 to 0x39 to be entered, try entering in one of the alphabetical keys from A-Z to trigger an input outside of the valid range. Does the application respond the way you think it should to invalid input?

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

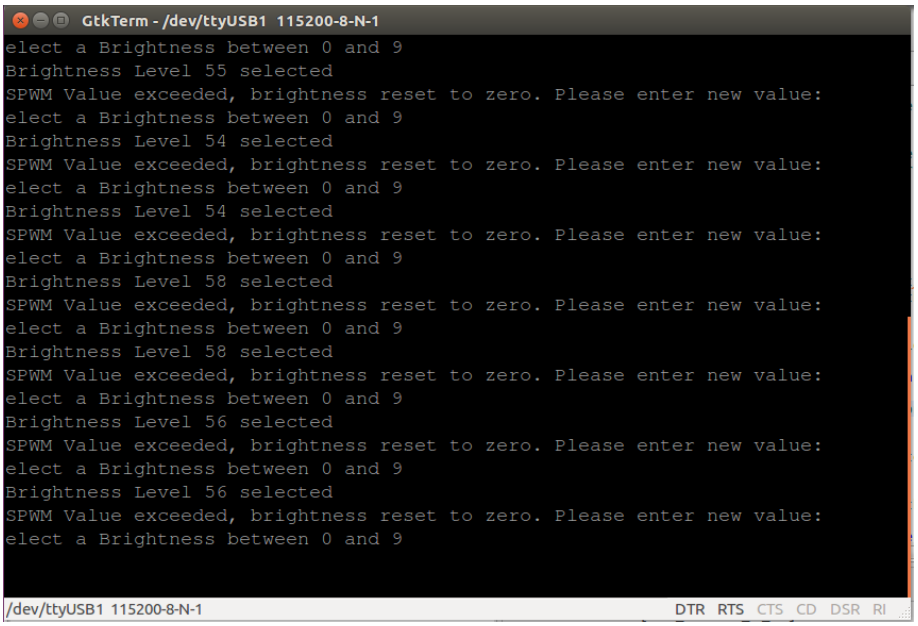


Figure 9 – Terminal Output from LED_Dimmer_w_Int Application

Questions:

Answer the following questions:

- *When does the function PWMIsr() get called?*

_____

- *GROUP DISCUSSION QUESTION – Think about this and we will discuss when all participants are done with the labs.*

  We handled the ISR functionality inside the ISR, what is another way of doing this?

_____

_____

Exploring Further

If you have more time and would like to investigate more…

- Add a global flag which gets set by the interrupt service routine and have the interrupt cleared by the main application loop once the flag is read.

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

- Remove the call to **SetupInterruptSystem()** and see if the interrupt service routine still gets called.

This concludes Lab 09.

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Revision History

Date	Version	Revision
12 Nov 13	01	Initial Release
22 Nov 13	02	Revisions after pilot
01 May 14	03	ZedBoard.org Training Course Release
15 Dec 14	04	Update to Vivado 2014.3
07 Jan 15	05	Update to Vivado 2014.4
09 Mar 15	06	Merge MicroZed and ZedBoard instructions
18 Mar 15	07	Finalize SDK 2014.4
Oct 15	08	Updated to SDK 2015.2
Aug 16	09	Updated to SDK 2016.2
Jun 17	10	Updated to SDK 2017.1 for MiniZed + Rebranding
Feb 18	11	Updated to Vivado/SDK 2017.4
July 19	12	Updated to Vivado/SDK 2019.1

Resources

www.minized.org

www.microzed.org

www.picozed.org

www.zedboard.org

www.em.avnet.com/drc

www.xilinx.com/zynq

www.xilinx.com/sdk

www.xilinx.com/vivado

www.xilinx.com/products/silicon-devices/soc/zynq-7000/ecosystem/index.htm

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.

Answers

Experiment 1

- *What happens when a value outside of the requested range of 0-9 is entered?*

The value is accepted, multiplied by the brightness clock multiplier and assigned to the PWM hardware block control registers. However, the custom IP block detects the out of range value and triggers an interrupt..

- *Are interrupts currently enabled in the example application provided?*

Hardware interrupts are currently provided from the PWM hardware block which are then routed to the GIC in the hardware platform. However, the software application at this point does not configure the GIC to allow the application execution to be interrupted so the software end of the interrupt is not currently enabled.

Experiment 2

- *When does the function PWMIsr() get called?*

The **PWMIsr()** is the interrupt service routine callback function. It is registered with the interrupt subsystem as the callback interrupt handler through the call to **XScuGic_Connect()** during our interrupt setup.

- *GROUP DISCUSSION QUESTION – Think about this and we will discuss when all participants are done with the labs.*

  We handled the ISR functionality inside the ISR, what is another way of doing this?

Alternatively, we could have the ISR just set a flag and then wait for the main function process the flag and write a message to the console.

© 2019 Avnet. All rights reserved. All trademarks and registered trademarks are the property of their respective owners. All specifications are subject to change without notice.

NOTICE OF DISCLAIMER: Avnet is providing this design, code, or information "as is." By providing the design, code, or information as one possible implementation of this feature, application, or standard, Avnet makes no representation that this implementation is free from any claims of infringement. You are responsible for obtaining any rights you may require for your implementation. Avnet expressly disclaims any warranty whatsoever with respect to the adequacy of the implementation, including but not limited to any warranties or representations that this implementation is free from claims of infringement and any implied warranties of merchantability or fitness for a particular purpose.