/**
*
* (c) Copyright 2010-13 Xilinx, Inc. All rights reserved.
*
* This file contains confidential and proprietary information of Xilinx, Inc.
* and is protected under U.S. and international copyright and other
* intellectual property laws.
*
* DISCLAIMER
* This disclaimer is not a license and does not grant any rights to the
* materials distributed herewith. Except as otherwise provided in a valid
* license issued to you by Xilinx, and to the maximum extent permitted by
* applicable law: (1) THESE MATERIALS ARE MADE AVAILABLE "AS IS" AND WITH ALL
* FAULTS, AND XILINX HEREBY DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS,
* IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF
* MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE;
* and (2) Xilinx shall not be liable (whether in contract or tort, including
* negligence, or under any other theory of liability) for any loss or damage
* of any kind or nature related to, arising under or in connection with these
* materials, including for any direct, or any indirect, special, incidental,
* or consequential loss or damage (including loss of data, profits, goodwill,
* or any type of loss or damage suffered as a result of any action brought by
* a third party) even if such damage or loss was reasonably foreseeable or
* Xilinx had been advised of the possibility of the same.
*
* CRITICAL APPLICATIONS
* Xilinx products are not designed or intended to be fail-safe, or for use in
* any application requiring fail-safe performance, such as life-support or
* safety devices or systems, Class III medical devices, nuclear facilities,
* applications related to the deployment of airbags, or any other applications
* that could lead to death, personal injury, or severe property or
* environmental damage (individually and collectively, "Critical
* Applications"). Customer assumes the sole risk and liability of any use of
* Xilinx products in Critical Applications, subject only to applicable laws
* and regulations governing limitations on product liability.
*
* THIS COPYRIGHT NOTICE AND DISCLAIMER MUST BE RETAINED AS PART OF THIS FILE
* AT ALL TIMES.
*
**/
/**/
/**
* @file xgpiops_intr_example.c
*
* This file contains a design example using the GPIO driver (XGpioPs) in an
* interrupt driven mode of operation.
*
* The example uses the interrupt capability of the GPIO to detect push button
* events and set the output LEDs based on the input . The user needs to press
* all the switches SW1-SW5 on the evaluation board to exit from this example.
*
* @note
* This example assumes that there is a Uart device in the HW design.
*
* <pre>
* MODIFICATION HISTORY:
*
* Ver   Who Date   Changes
* ----- ---- -------- ---
* 1.00a sv   01/18/10 First Release
*</pre>
*
**/

```
/************************** Include Files *****************************/

#include "xparameters.h"
#include "xgpiops.h"
#include "xscugic.h"
#include "xil_exception.h"
#include <xil_printf.h>

/************************** Constant Definitions **********************/

/*
 * The following constants map to the names of the hardware instances that
 * were created in the EDK XPS system.  They are defined here such that
 * the user can easily change all the needed device IDs in one place.
 */
#define GPIO_DEVICE_ID          XPAR_XGPIOPS_0_DEVICE_ID
#define INTC_DEVICE_ID          XPAR_SCUGIC_SINGLE_DEVICE_ID
#define GPIO_INTERRUPT_ID       XPAR_XGPIOPS_0_INTR

/*
 * The following constants define the GPIO banks that are used.
 */
#define INPUT_BANK      XGPIOPS_BANK0  /* Bank 0 of the GPIO Device */
#define OUTPUT_BANK     XGPIOPS_BANK1  /* Bank 1 of the GPIO Device */

/*
 * The following constants define the positions of the buttons of the GPIO.
 */
#define GPIO_ALL_BUTTONS        0xFFFF

/*
 * The following constant determines which buttons must be pressed to cause
 * interrupt processing to stop.
 */
#define GPIO_EXIT_CONTROL_VALUE 0x1F

#define printf                  xil_printf      /* Smalller foot-print printf */

/************************** Type Definitions **************************/

/***************** Macros (Inline Functions) Definitions ******************/

/************************** Function Prototypes ***********************/

static int GpioIntrExample(XScuGic *Intc, XGpioPs *Gpio, u16 DeviceId,
                           u16 GpioIntrId);
static void IntrHandler(void *CallBackRef, int Bank, u32 Status);
static int SetupInterruptSystem(XScuGic *Intc, XGpioPs *Gpio, u16 GpioIntrId);

/************************** Variable Definitions **********************/

/*
 * The following are declared globally so they are zeroed and so they are
 * easily accessible from a debugger.
 */
static XGpioPs Gpio; /* The Instance of the GPIO Driver */

static XScuGic Intc; /* The Instance of the Interrupt Controller Driver */

static u32 AllButtonsPressed; /* Intr status of the bank */

/*****************************************************************************/
/**
*
* Main function that invokes the GPIO Interrupt example.
```

```
*
* @param        None.
*
* @return
*               - XST_SUCCESS if the example has completed successfully.
*               - XST_FAILURE if the example has failed.
*
* @note         None.
*
*****************************************************************************/
int main(void)
{
        int Status;

        xil_printf("GPIO Interrupt Example Test \r\n");

        /*
         * Run the GPIO interrupt example, specify the parameters that
         * are generated in xparameters.h.
         */
        Status = GpioIntrExample(&Intc, &Gpio, GPIO_DEVICE_ID,
                                 GPIO_INTERRUPT_ID);

        if (Status != XST_SUCCESS) {
                xil_printf("GPIO Interrupt Example Test Failed\r\n");
                return XST_FAILURE;
        }

        xil_printf("Successfully ran GPIO Interrupt Example Test\r\n");
        return XST_SUCCESS;
}

/*****************************************************************************/
/**
* This function shows the usage of interrupt fucntionality of the GPIO device.
* It is responsible for initializing the GPIO device, setting up interrupts and
* providing a foreground loop such that interrupts can occur in the background.
*
* @param        Intc is a pointer to the XScuGic driver Instance.
* @param        Gpio is a pointer to the XGpioPs driver Instance.
* @param        DeviceId is the XPAR_<Gpio_Instance>_PS_DEVICE_ID value
*               from xparameters.h.
* @param        GpioIntrId is XPAR_<GIC>_<GPIO_Instance>_VEC_ID value
*               from xparameters.h
*
* @return       - XST_SUCCESS if the example has completed successfully.
*               - XST_FAILURE if the example has failed.
*
* @note         None
*
*****************************************************************************/
int GpioIntrExample(XScuGic *Intc, XGpioPs *Gpio, u16 DeviceId, u16 GpioIntrId)
{
        XGpioPs_Config *ConfigPtr;
        int Status;

        /*
         * Initialize the Gpio driver.
         */
        ConfigPtr = XGpioPs_LookupConfig(DeviceId);
        if (ConfigPtr == NULL) {
                return XST_FAILURE;
        }
        XGpioPs_CfgInitialize(Gpio, ConfigPtr, ConfigPtr->BaseAddr);
```

```
	/*
	 * Run a self-test on the GPIO device.
	 */
	Status = XGpioPs_SelfTest(Gpio);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}

	/*
	 * Setup direction register of bank0, so
	 * that all the pins are configured as inputs.
	 */
	XGpioPs_SetDirection(Gpio, INPUT_BANK, ~GPIO_ALL_BUTTONS);


	/*
	 * Set the direction for all signals to be
	 * outputs and Enable the Output enable for the LED Pins.
	 */
	XGpioPs_SetDirection(Gpio, OUTPUT_BANK, GPIO_ALL_BUTTONS);
	XGpioPs_SetOutputEnable(Gpio, OUTPUT_BANK, GPIO_ALL_BUTTONS);

	/*
	 * Setup the interrupts such that interrupt processing can occur. If
	 * an error occurs then exit.
	 */
	Status = SetupInterruptSystem(Intc, Gpio, GPIO_INTERRUPT_ID);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}

	printf("\n\rPush each of the 5 buttons once to exit\n\r");
	AllButtonsPressed = FALSE;

	/*
	 * Loop forever while the button changes are handled by the interrupt
	 * level processing.
	 */
	while(AllButtonsPressed == FALSE);

	printf("\n\r The GPIO Interrupt example has passed Successfully.\n\r");

	return XST_SUCCESS;
}

/****************************************************************************/
/**
* This function is the user layer callback function for the bank 0 interrupts of
* the GPIO device. It checks if all the switches have been pressed to stop the
* interrupt processing and exit from the example.
*
* @param	CallBackRef is a pointer to the upper layer callback reference.
* @param	Status is the Interrupt status of the GPIO bank.
*
* @return	None.
*
* @note		None.
*
****************************************************************************/
static void IntrHandler(void *CallBackRef, int Bank, u32 Status)
{
	XGpioPs *Gpio = (XGpioPs *)CallBackRef;
	static u32 ButtonsChanged;

	/*
```

```
         * Do nothing if the intr is generated for a different bank.
         */
        if (Bank != INPUT_BANK) {
                return;
        }

        ButtonsChanged |= Status;

        /*
         * Set the LEDs.
         */
        XGpioPs_Write(Gpio, OUTPUT_BANK, ButtonsChanged);

        if (ButtonsChanged == GPIO_EXIT_CONTROL_VALUE) {
                /*
                 * Five buttons are pressed to mark the completion of the test.
                 */
                AllButtonsPressed = TRUE;
                ButtonsChanged = 0;
        }
}


/****************************************************************************/
/**
*
* This function sets up the interrupt system for the example. It enables falling
* edge interrupts for all the pins of bank 0 in the GPIO device.
*
* @param        GicInstancePtr is a pointer to the XScuGic driver Instance.
* @param        GpioInstancePtr contains a pointer to the instance of the GPIO
*               component which is going to be connected to the interrupt
*               controller.
* @param        GpioIntrId is the interrupt Id and is typically
*               XPAR_<GICPS>_<GPIOPS_instance>_VEC_ID value from
*               xparameters.h.
*
* @return       XST_SUCCESS if successful, otherwise XST_FAILURE.
*
* @note         None.
*
****************************************************************************/
static int SetupInterruptSystem(XScuGic *GicInstancePtr, XGpioPs *Gpio,
                                u16 GpioIntrId)
{
        int Status;

        XScuGic_Config *IntcConfig; /* Instance of the interrupt controller */

        Xil_ExceptionInit();

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

        Status = XScuGic_CfgInitialize(GicInstancePtr, IntcConfig,
                                       IntcConfig->CpuBaseAddress);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }
```

```
	/*
	 * Connect the interrupt controller interrupt handler to the hardware
	 * interrupt handling logic in the processor.
	 */
	Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
				     (Xil_ExceptionHandler)XScuGic_InterruptHandler,
				     GicInstancePtr);

	/*
	 * Connect the device driver handler that will be called when an
	 * interrupt for the device occurs, the handler defined above performs
	 * the specific interrupt processing for the device.
	 */
	Status = XScuGic_Connect(GicInstancePtr, GpioIntrId,
				     (Xil_ExceptionHandler)XGpioPs_IntrHandler,
				     (void *)Gpio);
	if (Status != XST_SUCCESS) {
		return Status;
	}

	/*
	 * Enable falling edge interrupts for all the pins in bank 0.
	 */
	XGpioPs_SetIntrType(Gpio, INPUT_BANK, 0x00, 0x00, 0x00);

	/*
	 * Set the handler for gpio interrupts.
	 */
	XGpioPs_SetCallbackHandler(Gpio, (void *)Gpio, IntrHandler);


	/*
	 * Enable the GPIO interrupts of Bank 0.
	 */
	XGpioPs_IntrEnable(Gpio, INPUT_BANK, 0xFFFFFFFF);


	/*
	 * Enable the interrupt for the GPIO device.
	 */
	XScuGic_Enable(GicInstancePtr, GpioIntrId);


	/*
	 * Enable interrupts in the Processor.
	 */
	Xil_ExceptionEnableMask(XIL_EXCEPTION_IRQ);

	return XST_SUCCESS;
}
```