```
#define TESTAPP_GEN

/* $Id: xgpio_intr_tapp_example.c,v 1.1.2.1 2009/11/25 07:38:15 svemula Exp $ */
/******************************************************************************
*
* (c) Copyright 2002-2009 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 xgpio_intr_tapp_example.c
*
* This file contains a design example using the GPIO driver (XGpio) in an
* interrupt driven mode of operation. This example does assume that there is
* an interrupt controller in the hardware system and the GPIO device is
* connected to the interrupt controller.
*
* This file is used by the TestAppGen utility to include a simplified test for
* gpio interrupts.

* The buttons and LEDs are on 2 seperate channels of the GPIO so that interrupts
* are not caused when the LEDs are turned on and off.
*
* <pre>
* MODIFICATION HISTORY:
*
* Ver   Who  Date        Changes
* ----- ---- -------- -----------------------------------------------------
* 2.01a sn   05/09/06 Modified to be used by TestAppGen to include test for
*                     interrupts.
* 3.00a ktn  11/21/09 Updated to use HAL Processor APIs and minior changes
*                     as per coding guidelines.
*</pre>
*
******************************************************************************/

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

#include "xparameters.h"
#include "xgpio.h"
#include "xil_exception.h"
#include "xintc.h"
```

```
/************************** Constant Definitions *****************************/
#ifndef TESTAPP_GEN
/*
 * The following constants map to the names of the hardware instances that
 * were created in the EDK XPS system.  They are only defined here such that
 * a user can easily change all the needed device IDs in one place.
 */
#define GPIO_DEVICE_ID          XPAR_PUSH_BUTTONS_5BIT_DEVICE_ID
#define INTC_DEVICE_ID          XPAR_INTC_0_DEVICE_ID
#define INTC_GPIO_INTERRUPT_ID  XPAR_INTC_0_GPIO_3_VEC_ID
#define GPIO_CHANNEL1 1
/*
 * The following constants define the positions of the buttons and LEDs each
 * channel of the GPIO
 */
#define GPIO_ALL_LEDS           0xFFFF
#define GPIO_ALL_BUTTONS        0xFFFF

/*
 * The following constants define the GPIO channel that is used for the buttons
 * and the LEDs. They allow the channels to be reversed easily.
 */
#define BUTTON_CHANNEL   1      /* Channel 1 of the GPIO Device */
#define LED_CHANNEL      2      /* Channel 2 of the GPIO Device */
#define BUTTON_INTERRUPT XGPIO_IR_CH1_MASK  /* Channel 1 Interrupt Mask */

/*
 * The following constant determines which buttons must be pressed at the same
 * time to cause interrupt processing to stop and start
 */
#define INTERRUPT_CONTROL_VALUE 0x7

/*
 * The following constant is used to wait after an LED is turned on to make
 * sure that it is visible to the human eye.  This constant might need to be
 * tuned for faster or slower processor speeds.
 */
#define LED_DELAY        1000000

#endif

#define INTR_DELAY      0x00FFFFFF


/************************** Function Prototypes ******************************/
void GpioDriverHandler(void *CallBackRef);

int GpioIntrExample(XIntc* IntcInstancePtr, XGpio* InstancePtr,
                        u16 DeviceId, u16 IntrId,
                        u16 IntrMask, u32 *DataRead);

int GpioSetupIntrSystem(XIntc* IntcInstancePtr, XGpio* InstancePtr,
                        u16 DeviceId, u16 IntrId, u16 IntrMask);

void GpioDisableIntr(XIntc* IntcInstancePtr, XGpio* InstancePtr,
                         u16 IntrId, u16 IntrMask);

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

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

XIntc Intc; /* The Instance of the Interrupt Controller Driver */


static u16 GlobalIntrMask; /* GPIO channel mask that is needed by
                            * the Interrupt Handler */

static volatile u32 IntrFlag; /* Interrupt Handler Flag */

/*****************************************************************************/
/**
* This function is the main function of the GPIO example.  It is responsible
```

```
* for initializing the GPIO device, setting up interrupts and providing a
* foreground loop such that interrupt can occur in the background.
*
* @param        None.
*
* @return
*               - XST_SUCCESS to indicate success.
*               - XST_FAILURE to indicate failure.
*
* @note         None.
*
******************************************************************************/
#ifndef TESTAPP_GEN
int main(void)
{
        int Status;
        u32 DataRead;

          print(" Press button to Generate Interrupt\r\n");

          Status = GpioIntrExample(&Intc, &Gpio,
                                   GPIO_DEVICE_ID,
                                   INTC_GPIO_INTERRUPT_ID,
                                   GPIO_CHANNEL1, &DataRead);

        if (Status == 0 ){
                if(DataRead == 0)
                        print("No button pressed. \r\n");
                else
                        print("Gpio Interrupt Test PASSED. \r\n");
        } else {
                 print("Gpio Interrupt Test FAILED.\r\n");
        }
}
#endif

/*****************************************************************************/
/**
*
* This is the entry function from the TestAppGen tool generated application
* which tests the interrupts when enabled in the GPIO
*
* @param        IntcInstancePtr is a reference to the Interrupt Controller
*               driver Instance
* @param        InstancePtr is a reference to the GPIO driver Instance
* @param        DeviceId is the XPAR_<GPIO_instance>_DEVICE_ID value from
*               xparameters.h
* @param        IntrId is XPAR_<INTC_instance>_<GPIO_instance>_IP2INTC_IRPT_INTR
*               value from xparameters.h
* @param        IntrMask is the GPIO channel mask
* @param        DataRead is the pointer where the data read from GPIO Input is
*               returned
*
* @return       XST_SUCCESS if the Test is successful, otherwise XST_FAILURE
*
* @note         None.
*
******************************************************************************/
int GpioIntrExample(XIntc* IntcInstancePtr, XGpio* InstancePtr, u16 DeviceId,
                        u16 IntrId, u16 IntrMask, u32 *DataRead)
{
        int Status;
        u32 delay;

        /* Initialize the GPIO driver. If an error occurs then exit */

        Status = XGpio_Initialize(InstancePtr, DeviceId);
        if (Status != XST_SUCCESS) {
                return XST_FAILURE;
        }

        Status = GpioSetupIntrSystem(IntcInstancePtr,
                                InstancePtr,
                                DeviceId,
                                IntrId,
                                IntrMask);
        if (Status != XST_SUCCESS) {
```

```
                return XST_FAILURE;
        }

        IntrFlag = 0;
        delay = 0;

        while(!IntrFlag && (delay < INTR_DELAY)) {
                delay++;
        }

        GpioDisableIntr(IntcInstancePtr,
                        InstancePtr,
                        IntrId,
                        IntrMask);

        *DataRead = IntrFlag;

        return Status;

}


/*****************************************************************************/
/**
*
* This function performs the GPIO set up for Interrupts
*
* @param        IntcInstancePtr is a reference to the Interrupt Controller
*               driver Instance
* @param        InstancePtr is a reference to the GPIO driver Instance
* @param        DeviceId is the XPAR_<GPIO_instance>_DEVICE_ID value from
*               xparameters.h
* @param        IntrId is XPAR_<INTC_instance>_<GPIO_instance>_IP2INTC_IRPT_INTR
*               value from xparameters.h
* @param        IntrMask is the GPIO channel mask
*
* @return       XST_SUCCESS if the Test is successful, otherwise XST_FAILURE
*
* @note         None.
*
******************************************************************************/
int GpioSetupIntrSystem(XIntc* IntcInstancePtr, XGpio* InstancePtr,
                        u16 DeviceId, u16 IntrId, u16 IntrMask)

{
        int Result;

        GlobalIntrMask = IntrMask;

#ifndef TESTAPP_GEN
        /*
         * Initialize the interrupt controller driver so that it's ready to use.
         * specify the device ID that was generated in xparameters.h
         */
        Result = XIntc_Initialize(IntcInstancePtr, INTC_DEVICE_ID);
        if (Result != XST_SUCCESS) {
                return Result;
        }
#endif

        /* Hook up simple interrupt service routine for TestApp */

        Result = XIntc_Connect(IntcInstancePtr, IntrId,
                               (XInterruptHandler)GpioDriverHandler,
                               InstancePtr);

        /*
         * Enable the GPIO channel interrupts so that push button can be detected
         * and enable interrupts for the GPIO device
         */
        XGpio_InterruptEnable(InstancePtr, IntrMask);
        XGpio_InterruptGlobalEnable(InstancePtr);

        /* Enable the interrupt vector at the interrupt controller */
        XIntc_Enable(IntcInstancePtr, IntrId);

#ifndef TESTAPP_GEN
```

```
        /*
         * Initialize the exception table and register the interrupt
         * controller handler with the exception table
         */
        Xil_ExceptionInit();
        Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_INT,
                                (Xil_ExceptionHandler)XIntc_InterruptHandler,
                                IntcInstancePtr);

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

        /*
         * Start the interrupt controller such that interrupts are recognized
         * and handled by the processor
         */
        Result = XIntc_Start(IntcInstancePtr, XIN_REAL_MODE);
#endif

        if (Result != XST_SUCCESS) {
                return Result;
        }

        return XST_SUCCESS;
}


/****************************************************************************/
/**
*
* This is the interrupt handler routine for the GPIO for this example.
*
* @param        CallbackRef is the Callback reference for the handler.
*
* @return       None.
*
* @note         None.
*
*****************************************************************************/
void GpioDriverHandler(void *CallbackRef)
{
        XGpio *GpioPtr = (XGpio *)CallbackRef;

        IntrFlag = 1;
        /*
         * Clear the Interrupt
         */
        XGpio_InterruptClear(GpioPtr, GlobalIntrMask);

}

/****************************************************************************/
/**
*
* This function disables the interrupts for the GPIO
*
* @param        IntcInstancePtr is a pointer to the Interrupt Controller
*               driver Instance
* @param        InstancePtr is a pointer to the GPIO driver Instance
* @param        IntrId is XPAR_<INTC_instance>_<GPIO_instance>_IP2INTC_IRPT_INTR
*               value from xparameters.h
* @param        IntrMask is the GPIO channel mask
*
* @return       None
*
* @note         None.
*
*****************************************************************************/
void GpioDisableIntr(XIntc* IntcInstancePtr, XGpio* InstancePtr,
                     u16 IntrId, u16 IntrMask)
{
        XGpio_InterruptDisable(InstancePtr, IntrMask);
        XIntc_Disable(IntcInstancePtr, IntrId);
        return;
}
```