```
/* $Id: xintc_selftest.c,v 1.1.2.1 2010/09/17 05:26:04 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 xintc_selftest.c
*
* Contains diagnostic self-test functions for the XIntc component. This file
* requires other files of the component to be linked in also.
*
* <pre>
* MODIFICATION HISTORY:
*
* Ver   Who  Date     Changes
* ----- ---- -------- -----------------------------------------------
* 1.00b jhl  02/21/02 First release
* 1.10c mta  03/21/07 Updated to new coding style
* 2.00a ktn  10/20/09 Updated to use HAL Processor APIs
* </pre>
*
******************************************************************************/

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

#include "xil_types.h"
#include "xil_assert.h"
#include "xintc.h"
#include "xintc_i.h"

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

#define XIN_TEST_MASK 1

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


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

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


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


/*****************************************************************************/
/**
* Run a self-test on the driver/device. This is a destructive test.
*
* This involves forcing interrupts into the controller and verifying that they
* are recognized and can be acknowledged. This test will not succeed if the
* interrupt controller has been started in real mode such that interrupts
* cannot be forced.
* @param        InstancePtr is a pointer to the XIntc instance to be worked on.
* @return
*               - XST_SUCCESS if self-test is successful.
*               - XST_INTC_FAIL_SELFTEST if the Interrupt controller fails the
*               self-test. It will fail the self test if the device has
*               previously been started in real mode.
* @note         None.
*
******************************************************************************/
int XIntc_SelfTest(XIntc * InstancePtr)
   {
   u32 CurrentMIE;
   u32 CurrentISR;
   u32 Temp;

/* Assert the arguments */
   Xil_AssertNonvoid(InstancePtr != NULL);
   Xil_AssertNonvoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);

   CurrentMIE = XIntc_In32(InstancePtr->BaseAddress + XIN_MER_OFFSET);

/* Acknowledge all pending interrupts by reading the interrupt status register and writing
   the value to the acknowledge register */
   Temp = XIntc_In32(InstancePtr->BaseAddress + XIN_ISR_OFFSET);

   XIntc_Out32(InstancePtr->BaseAddress + XIN_IAR_OFFSET, Temp);

/* Verify that there are no interrupts by reading the interrupt status */
   CurrentISR = XIntc_In32(InstancePtr->BaseAddress + XIN_ISR_OFFSET);

/* ISR should be zero after all interrupts are acknowledged */
   if (CurrentISR != 0)
      { return XST_INTC_FAIL_SELFTEST; }

/* Set a bit in the ISR which simulates an interrupt */
   XIntc_Out32(InstancePtr->BaseAddress + XIN_ISR_OFFSET, XIN_TEST_MASK);

/* Verify that it was set */
   CurrentISR = XIntc_In32(InstancePtr->BaseAddress + XIN_ISR_OFFSET);

   if (CurrentISR != XIN_TEST_MASK)
      { return XST_INTC_FAIL_SELFTEST; }

/* Acknowledge the interrupt */
   XIntc_Out32(InstancePtr->BaseAddress + XIN_IAR_OFFSET, XIN_TEST_MASK);

/* Read back the ISR to verify that the interrupt is gone */
   CurrentISR = XIntc_In32(InstancePtr->BaseAddress + XIN_ISR_OFFSET);

   if (CurrentISR != 0)
      { return XST_INTC_FAIL_SELFTEST; }

   return XST_SUCCESS;
   }

/*****************************************************************************/
/**
* Allows software to simulate an interrupt in the interrupt controller. This
* function will only be successful when the interrupt controller has been
* started in simulation mode. Once it has been started in real mode,
* interrupts cannot be simulated. A simulated interrupt allows the interrupt
* controller to be tested without any device to drive an interrupt input
```

```
* signal into it.
* @param        InstancePtr is a pointer to the XIntc instance to be worked on.
* @param        Id is the interrupt ID for which to simulate an interrupt.
* @return
*               - XST_SUCCESS if successful
*               - XST_FAILURE if the interrupt could not be
*               simulated because the interrupt controller is or
*               has previously been in real mode.
* @note         None.
*
*****************************************************************************/
int XIntc_SimulateIntr(XIntc * InstancePtr, u8 Id)
   {
   u32 Mask;
   u32 MasterEnable;

/* Assert the arguments */
   Xil_AssertNonvoid(InstancePtr != NULL);
   Xil_AssertNonvoid(InstancePtr->IsReady == XIL_COMPONENT_IS_READY);
   Xil_AssertNonvoid(Id < XPAR_INTC_MAX_NUM_INTR_INPUTS);

/* Get the contents of the master enable register and determine if hardware interrupts have
   already been enabled, if so, this is a write once bit such that simulation can't be done
   at this point because the ISR register is no longer writable by software */
   MasterEnable = XIntc_In32(InstancePtr->BaseAddress + XIN_MER_OFFSET);
   if (MasterEnable & XIN_INT_HARDWARE_ENABLE_MASK)
      { return XST_FAILURE; }

/* The Id is used to create the appropriate mask for the desired bit position. Id currently
   limited to 0 - 31 */
   Mask = XIntc_BitPosMask[Id];

/* Enable the selected interrupt source by reading the interrupt enable register and then
   modifying only the specified interrupt id enable */
   XIntc_Out32(InstancePtr->BaseAddress + XIN_ISR_OFFSET, Mask);

/* Indicate the interrupt was successfully simulated */
   return XST_SUCCESS;
   }
```