PowerPC
Example Design

XILINX®

PowerPC™ Example Design

With Interrupt Controller

Target board: Xilinx® Virtex-II Pro™ ML310 Evaluation Platform

Development system: Embedded Development Kit (EDK), version 7.1i

This example design is provided with the EDK 7.1i development system. The design is completely implemented and ready to download and run on a ML310 Evaluation Board.

Hardware Requirements:

Xilinx Virtex-II Pro ML310 Evaluation Platform, Revision D (or later)

Serial cable: 9-pin female to 9-pin female “null modem” (cross-over)

Xilinx Parallel Cable 4 (with flat ribbon cable)

Software Requirements:

Xilinx Embedded Development Kit (EDK) 7.1i or later

Xilinx ISE™ 7.1i (or later)

Hyperterminal (or similar) for host communication with the embedded system

Design Description

This example design is originally created using the Base System Builder (BSB) in the EDK development system. BSB generates the customized embedded system hardware platform based on user selections from features available on the target board (ML310 Evaluation Board). BSB also generates a sample application program (TestApp_Memory.c) which exercises some of the selected hardware features. This example design contains an application program (TestInterrupt.c) which is based on TestApp_Memory.c, and to which an interrupt test routine and two interrupt service routines have been added. This design technique is typical of the way a development board, such as the ML310 Evaluation Board, can be used to quickly begin prototyping an embedded processor application.

The TestInterrupt.c application program begins by running the automatically generated tests from TestApp_Memory.c, which test the LEDs and OPB_BRAM memory. It then calls the InterruptTest routine which has been added. InterruptTest() initializes the interrupt system and the timer peripheral, then enters a loop waiting for interrupts from either the timer or the UART. When a timer interrupt occurs, the timer's interrupt service routine advances a pattern on the LEDs. When a character is received on the RS232 channel, the UART's ISR increases or decreases the timeout interval of the timer, causing the LED pattern to advance faster ("f") or slower ("s"). Receiving an "x" disables the interrupts and exits the program.

Design Contents

The embedded processor system contained in this example design consists of the following:

- PowerPC 405 32-bit hard processor core, running at 100 MHz, based on a 100 MHz system clock
- 16 KB of on-chip block RAM connected to the processor PLB bus, used for all instruction and data storage
- 16 KB of on-chip block RAM connected to the processor instruction-side OCM bus (used for interrupt vector storage)
- 8 KB of on-chip block RAM connected to the processor OPB bus (used only for test)
- RS232 serial channel on the ML310 Evaluation Board, connected to a UART peripheral (OPB_UARTlite) on the processor OPB bus, used for stdin (interrupt-driven) and stdout
- Eight LEDs on the ML310 Evaluation Board, connected to a General Purpose I/O peripheral (OPB_GPIO) on the processor OPB bus
- Timer/counter peripheral (OPB_Timer) on the processor OPB bus, used to generate interrupts at varying intervals
- Interrupt controller (OPB_Intc) on the processor OPB bus, used to manage multiple interrupts

The embedded processor system is implemented in a XC2VP30-FF896-6 Virtex-II Pro FPGA device on the ML310 Evaluation Board.

Project File Description

system.xmp: XPS project file (target device, project options, design file pointers)

system.mhs: Microprocessor Hardware Specification (processor, busses, peripherals)

system.mss: Microprocessor Software Specification (libraries, drivers, system software options)

data/system.ucf: Implementation constraint file (pinouts and clock frequency)

TestInterrupt/src/TestInterrupt.c: Application program

TestApp_Memory/src/TestApp_Memory_LinkScr: Linker script (generated by BSB for TestApp_Memory.c)

Instructions to Run the Example Design

1. Connect Parallel Cable 4 between your host computer and the ML310 Evaluation Board. Supply power to the Parallel Cable 4 using either the PS2 port of your host computer or external power supply.
2. Attach the RS232 Mini-cable (provided with board) to the FPGA UART connector (J4) on the ML310 board; the (speckled) red edge should be closest to pin 1. Connect the serial cable between your host computer and the RS232 Mini-cable.
3. Apply power to the ML310 Evaluation Board.
4. Start a hyperterminal (or similar) session on your host computer with the following settings:
   - Select the COM port corresponding to connected serial port on your host computer
   - Baud Rate = 9600
   - Data = 8 bits
   - Parity = none
   - Stop = 1 bit
   - Flow control = none
5. Invoke Xilinx Platform Studio (XPS).
6. If the Xilinx Platform Studio dialog box appears, choose Open a Recent Project and select "Browse for more Projects" (default); click OK. Otherwise, select File -> Open Project from the XPS main menu.
7. Navigate to the directory where you expanded the PPC_ML310_Tutorial_7_1 example design; select the system.xmp project file; and click Open.
8. In XPS, select Tools -> Download. The FPGA bitstream, including the application software, will be downloaded using the Parallel Cable 4 into the JTAG port of the FPGA on the ML310 Evaluation Board. When completed, the application will begin running immediately.

   The 8 LEDs should display a test pattern of alternating "walking ones" and "walking zeros", cycling 5 times total. (This may occur concurrently with the hyperterminal output.)

   The hyperterminal should display:

```
-- Entering main() --
Starting MemoryTest for opb_bram_if_cntlr_1:
  Running 32-bit test...
PASSED!
  Running 16-bit test...
PASSED!
  Running 8-bit test...
PASSED!
-- Entering InterruptTest() --
```

   The 8 LEDs should change to a Johnson-counter pattern (shift in all ones, shift in all zeros, repeating continuously).

9. Type “f” in the hyperterminal a few times. Each time, the rate of the LED pattern should double, and the hyperterminal should respond with

```
-- Reducing timer period by half --
```

10. Type “s” in the hyperterminal a few time. Each time, the rate of the LED pattern should slow by half, and the hyperterminal should respond with

```
-- Doubling timer period --
```

11. Type “x” in the hyperterminal to stop the LED pattern and exit the application.

Procedure Used to Create the Example Design

The XPS project presented in this example design was prepared using the following procedure:

Creating the Hardware Platform Using Base System Builder

1. Invoke XPS.
2. If the Xilinx Platform Studio dialog box appears, choose Base System Builder Wizard (default); click OK. Otherwise, select File -> New Project -> Base System Builder.
3. In “Create New Project...”, click Browse to select / create a project directory in which to write the system.xmp project file; click Open. Click OK.
4. In the BSB Welcome dialog, choose “I would like to create a new design” (default); click Next.
5. In BSB Select Board:
    a. Choose “I would like to create a system for the following development board” (default).
    b. For Board Vendor, select Xilinx.
    c. For Board Name, select Virtex-II Pro ML310 Evaluation Platform
    d. For Board Revision, select the revision of the ML310 Evaluation Board you are using.
    e. Click Next.
6. In BSB Select Processor, choose PowerPC (default); click Next.
7. In BSB Configure Processor:
    a. Under System Wide Settings, Reference Clock Frequency = 100 MHz (default), Processor Clock Frequency = 100 MHz (default), and Bus Clock Frequency = 100 MHz (default).
    b. Under Processor Configuration, for Debug Interface, choose FPGA JTAG (default).
    c. For On-Chip Memory, select Data = None (default) and Instruction = 16 KB.
    d. Click Next. (Do not enable the cache.)
8. In BSB Configure I / O Interfaces:
    a. Check RS232_Uart (default); Peripheral = OPB UARTLITE (default); Baud Rate = 9600 (default); Data Bits = 8 (default); Parity = None (default). Check “Use Interrupt”.
    b. Uncheck DDR_SDRAM_32Mx64.
    c. Uncheck SPI_EEPROM.

   d. Check LEDs_8Bit (default); Peripheral = OPB_GPIO (default). (Do not check Use Interrupts.)

   e. Click Next.

9. In BSB Configure Additional I / O Interfaces, uncheck all remaining peripherals (may span more than one screen); click Next.
10. In BSB Add Internal Peripherals:
    a. For PLB BRAM IF CNTLR, select Memory Size = 16 KB (default).
    b. Click Add Peripheral; select “OPB BRAM IF CNTLR”; click OK; for Memory Size, select 8 KB (default).
    c. Click Add Peripheral again; select OPB Timer; click OK; for Count Bit Width, select 32 (default); for Timer Mode, choose “One timer is present”; check “Use Interrupt”.
    d. Click Next.
11. In BSB Software Setup:
    a. For Standard Input and for Standard Output, select RS232_Uart (default).
    b. Under Sample Application Selection, check Memory Test (default).
    c. Click Next.
12. In BSB Configure Memory Test Application:
    a. For Instructions, Data and Stack / Heap, select “plb_bram_if_cntlr_1” (default).
    b. Click Next.
13. The BSB System Created screen will appear summarizing your selections; click Generate. Click Finish.

Preparing the Software Application

1. In XPS, in the Applications tab, right-click “Project: TestApp_Memory”; uncheck “Mark to Initialize BRAMs”.
2. Under “Project: TestApp_Memory”, expand Sources; double-click “... / TestApp_Memory.c”:
   a. Edit the program to add the InterruptTest() routine, the UART ISR and the Timer ISR, as shown in the listing below.
   b. Before the end of the main() routine, insert a call to InterruptTest(), as shown in the listing below.
   c. Select File -> Save As; navigate up 2 levels to your main project directory; create a new directory named “TestInterrupt” and open it; create a new directory named “src” and open it; replace the File Name with “TestInterrupt.c” and Save it.

   As a short-cut, you may copy the TestInterrupt.c program file from this example design.
3. In XPS, select Project -> Add SW Application Project; for Project Name, type “TestInterrupt”; click OK.
4. In the Applications tab, under “Project: TestInterrupt”, right-click Sources; select Add File; browse to the TestInterrupt / src directory; select TestInterrupt.c; click Open.
5. Under “Project: TestInterrupt”, double-click Compiler Options. In the Set Compiler Settings dialog, in the Directories tab, click the browse button (“...”) for Linker Script, navigate to the TestApp_Memory / src directory, select TestApp_Memory_LinkScr and click Open; click OK. As this application uses interrupts, it is necessary to map the

.vectors section of the compiled program to a different physical memory region than the remainder of the instruction and data sections, because .vectors must be aligned on a 64 KB boundary.

6. In XPS, select Project -> Software Platform Settings. In the “Processor, Driver Parameters and Interrupt Handlers” tab, in the Global Interrupt Handlers section, under “tmrctr: opb_timer_1”, for interrupt_handler, type “timer_int_handler” in the Current Value column; click OK. This statically registers the function timer_int_handler() in TestInterrupt.c as the ISR for interrupts received from this peripheral. Note: To demonstrate dynamic registration, the UART’s ISR (uart_int_handler) is registered by a call to XIntc_RegisterHandler in TestInterrupt.c; therefore, do not enter a value for the UARTlite interrupt_handler in the Software Platform Settings dialog.

Implementing the Embedded System

1. In XPS, select Tools -> Update Bitstream. This implements the embedded processor hardware platform, compiles the system and application software, and merges them into a bitstream ready for download to the board.

Program Listing

File: TestInterrupt.c

All code added to the original TestApp_Memory.c shown in **bold** (except comments).

```
/*
 *   * Copyright (c) 2005 Xilinx, Inc.  All rights reserved.
 *
 * Xilinx, Inc.
 * XILINX IS PROVIDING THIS DESIGN, CODE, OR INFORMATION "AS IS" AS A
 * COURTESY TO YOU.  BY PROVIDING THIS DESIGN, CODE, OR INFORMATION AS
 * ONE POSSIBLE   IMPLEMENTATION OF THIS FEATURE, APPLICATION OR
* STANDARD, XILINX IS MAKING NO REPRESENTATION THAT THIS IMPLEMENTATION
 * IS FREE FROM ANY CLAIMS OF INFRINGEMENT, AND YOU ARE RESPONSIBLE
 * FOR OBTAINING ANY RIGHTS YOU MAY REQUIRE FOR YOUR IMPLEMENTATION
 * XILINX 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, IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE.
 */

/*
 * Xilinx EDK 7.1
 *
 * This file is a sample test application
 *
 * This application is intended to test and/or illustrate some
 * functionality of your system.  The contents of this file may
 * vary depending on the IP in your system and may use existing
 * IP driver functions.  These drivers will be generated in your
 * XPS project when you run the "Generate Libraries" menu item
 * in XPS.
 */
```

```
// Located in: ppc405_0/include/xparameters.h
#include "xparameters.h"

#include "xgpio_l.h"
#include "xutil.h"

/* Begin user-supplied interrupt test routine */

/* This example demonstrates how to use an interrupt controller
 * that responds to interrupts from two peripherals (UART and OPB_timer)
 * in a PowerPC based system.
 * This interrupt test routine has been added to the test application
 * (TestApp_Memory) generated by the Base System Builder.
 */

#include “xtmrctr_l.h” /* timer/counter peripheral control functions */
#include “xuartlite_l.h” /* uartlite peripheral control functions */
#include “xintc_l.h” /* interrupt controller peripheral control
   functions */
#include “xexception_l.h” /* PPC exception handler control functions */

#define LED_MSB 0x00000080 /* mask for position of left-most LED */

/* Global variables */
unsigned int led_data = 0; /* initial LED pattern when interrupt test
  begins */
unsigned int timer_count = 33554432; /* initial timer period in OPB
  cycles ~= 0.3 sec */
volatile unsigned int exit_command = 0; /* flag from UART ISR to exit
  InterruptTest routine */

/* UART interrupt service routine */
void uart_int_handler(void *baseaddr_p) {
   char c;
   /* While UART receive FIFO has data */
   while (!XUartLite_mIsReceiveEmpty(XPAR_RS232_UART_BASEADDR)) {
      /* Read a character */
      c = XUartLite_RecvByte(XPAR_RS232_UART_BASEADDR);
      switch (c) {
         case ’f’: /* FASTER command */
            if (timer_count > 1) {
               timer_count >>= 1;
               print("-- Reducing timer period by half --\r\n");}
            break;
         case ’s’: /* SLOWER command */
            if (timer_count < 1073741824) {
               timer_count <<= 1;
               print("-- Doubling timer period --\r\n");}
            break;
         case ’x’: /* EXIT command */
            exit_command = 1;
      }
      /* Update timer period with new value of timer_count */
      XTmrCtr_mSetLoadReg(XPAR_OPB_TIMER_1_BASEADDR, 0, timer_count);
   }
}

/* Timer interrupt service routine */
/* Note: This ISR was registered statically in the Software Platform
```

```
   Settings dialog */
void timer_int_handler(void * baseaddr_p) {
   unsigned int csr;
   /* Read timer 0 CSR to see if it requested the interrupt */
   csr = XTmrCtr_mGetControlStatusReg(XPAR_OPB_TIMER_1_BASEADDR, 0);
   if (csr & XTC_CSR_INT_OCCURED_MASK) {
      /* Increment led_count in a johnson-counter pattern */
      if (led_data & LED_MSB)
         led_data = led_data << 1;
      else
         led_data = (led_data << 1) + 1;
      /* Update LED output pattern */
      XGpio_mSetDataReg(XPAR_LEDS_8BIT_BASEADDR, 1, led_data);
   }
   /* Clear the timer interrupt */
   XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_1_BASEADDR, 0, csr);
}

/* Interrupt test routine */
void InterruptTest(void) {
   print("-- Entering InterruptTest() --\r\n");
   /* Initialize exception handling */
   XExc_Init();
   /* Register external interrupt handler */
   XExc_RegisterHandler(XEXC_ID_NON_CRITICAL_INT,
      (XExceptionHandler)XIntc_DeviceInterruptHandler,
    (void *)XPAR_OPB_INTC_0_DEVICE_ID);
   /* Register the UART interrupt handler in the vector table */
   XIntc_RegisterHandler(XPAR_OPB_INTC_0_BASEADDR,
      XPAR_OPB_INTC_0_RS232_UART_INTERRUPT_INTR,
      (XInterruptHandler)uart_int_handler,
      (void *)XPAR_RS232_UART_BASEADDR);
   /* Start the interrupt controller */
   XIntc_mMasterEnable(XPAR_OPB_INTC_0_BASEADDR);
   /* Set the gpio for LEDs as output */
   XGpio_mSetDataDirection(XPAR_LEDS_8BIT_BASEADDR, 1, 0x00000000);
   /* Set the number of cycles the timer counts before interrupting */
   XTmrCtr_mSetLoadReg(XPAR_OPB_TIMER_1_BASEADDR, 0, timer_count);
   /* Reset the timers, and clear interrupts */
   XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_1_BASEADDR, 0,
      XTC_CSR_INT_OCCURED_MASK | XTC_CSR_LOAD_MASK );
   /* Enable timer and uart interrupt requests in the interrupt
      controller */
   XIntc_mEnableIntr(XPAR_OPB_INTC_0_BASEADDR,
      XPAR_OPB_TIMER_1_INTERRUPT_MASK |
      XPAR_RS232_UART_INTERRUPT_MASK);
   /* Start the timers */
   XTmrCtr_mSetControlStatusReg(XPAR_OPB_TIMER_1_BASEADDR, 0,
      XTC_CSR_ENABLE_TMR_MASK | XTC_CSR_ENABLE_INT_MASK |
      XTC_CSR_AUTO_RELOAD_MASK | XTC_CSR_DOWN_COUNT_MASK);
   /* Enable PPC non-critical interrupts */
   XExc_mEnableExceptions(XEXC_NON_CRITICAL);
   /* Enable UART interrupts */
   XUartLite_mEnableIntr(XPAR_RS232_UART_BASEADDR);

   /* Wait for interrupts to occur until exit_command received from
      UART ISR */
   while (!exit_command)
   ;
   /* Disable PPC non-critical interrupts */
   XExc_mDisableExceptions(XEXC_NON_CRITICAL);
   print("-- Exiting InterruptTest() --\r\n");
}

/* End user-supplied interrupt test routine */

/*
 * Routine to write a pattern out to a GPIO
 * which is configured as an output
 *   PARAMETER C_ALL_INPUTS = 0
 */
void WriteToGPOutput(Xuint32 BaseAddress, int gpio_width) {
   int i=0, j=0, k=0;
   int numTimes = 5;

   XGpio_mSetDataDirection(BaseAddress, 1, 0x00000000);
     /* Set as outputs */
   while (numTimes > 0) {
      j = 1;
      for(i=0; i<(gpio_width-1); i++) {
         XGpio_mSetDataReg(BaseAddress, 1, j);
         j = j << 1;
         for (k=0; k<1000000; k++) {
            ; //wait
         }
      }
      j = 1;
      j = ~j;
      for(i=0; i<(gpio_width-1); i++) {
         XGpio_mSetDataReg(BaseAddress, 1, j);
         j = j << 1;
         for (k=0; k<1000000; k++) {
            ; //wait
         }
      }
      numTimes--;
   }
}

//====================================================

int main (void) {

   print("-- Entering main() --\r\n");

   WriteToGPOutput(XPAR_LEDS_8BIT_BASEADDR, 8);

   /*
    * MemoryTest routine will not be run for the memory at
    * 0xffffc000 (plb_bram_if_cntlr_1)
   * because it is being used to hold a part of this application program
    */

   /* Testing BRAM Memory (opb_bram_if_cntlr_1)*/
   {
      XStatus status;
```

```
         print("Starting MemoryTest for opb_bram_if_cntlr_1:\r\n");
         print("  Running 32-bit test...");
         status =
          XUtil_MemoryTest32((Xuint32*)XPAR_OPB_BRAM_IF_CNTLR_1_BASEADDR,
           1024, 0xAAAA5555, XUT_ALLMEMTESTS);
         if (status == XST_SUCCESS) {
            print("PASSED!\r\n");
         }
         else {
            print("FAILED!\r\n");
         }
         print("  Running 16-bit test...");
         status =
          XUtil_MemoryTest16((Xuint16*)XPAR_OPB_BRAM_IF_CNTLR_1_BASEADDR,
           2048, 0xAA55, XUT_ALLMEMTESTS);
         if (status == XST_SUCCESS) {
            print("PASSED!\r\n");
         }
         else {
            print("FAILED!\r\n");
         }
         print("  Running 8-bit test...");
         status =
           XUtil_MemoryTest8((Xuint8*)XPAR_OPB_BRAM_IF_CNTLR_1_BASEADDR,
           4096, 0xA5, XUT_ALLMEMTESTS);
         if (status == XST_SUCCESS) {
            print("PASSED!\r\n");
         }
         else {
            print("FAILED!\r\n");
         }
      }

      /* Run user-supplied interrupt test routine */
      InterruptTest();

      print("-- Exiting main() --\r\n");
      return 0;
   }
```

Further Reading

A complete description of design techniques and function calls required to implement interrupts in a PowerPC design can be found in the Interrupt Management chapter of the *Platform Studio User Guide*. Details on the driver calls used to control the UARTlite, OPB_Timer, OPB_GPIO and OPB_Intc peripherals can be found in the *Driver Reference Guide*.