MicroBlaze Example Design

XILINX®

MicroBlaze Example Design

with Interrupt Controller

Target board: Spartan-3 Starter Board

Development system: Embedded Development Kit (EDK), version 6.3i

This example design is provided with the EDK 6.3i development system. The design is completely implemented and ready to download and run on a Spartan-3 Starter Board.

Hardware Requirements:

Xilinx Spartan-3 Starter Board, Revision E (or later)

Serial cable (9-pin male to 9-pin female)

Xilinx Parallel Cable 4 (with flat ribbon cable)

Software Requirements:

Xilinx Embedded Development Kit (EDK) 6.3i or later

Xilinx ISE 6.3i (or later): Foundation, Alliance or BaseX configuration

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 custom embedded system hardware platform based on user selections from features available on the target board (Spartan-3 Starter Board). BSB also generates a sample application program (TestApp.c) which exercises some of the selected hardware features. This example design contains an application program (TestInterrupt.c) which is based on TestApp.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 Spartan-3 Starter Board, can be used to quickly begin prototyping an embedded processor application.

The TestInterrupt.c application program begins with the automatically generated tests from TestApp.c, which test the LEDs, pushbuttons, SRAM memory and on-chip OPB 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 get faster ("f") or slower ("s"). Receiving an "x" stops the timer and exits the program.

Design Contents

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

- MicroBlaze 32-bit soft processor core (v3.00.a), running on a 50MHz system clock
- 16KB of on-chip block RAM connected to the processor Local Memory Bus (LMB), used for all instruction and data storage
- 8KB of on-chip block RAM connected to the processor OPB bus (used only for test)
- 256K x 32 SRAM on the Spartan-3 Starter Board, connected to an External Memory Controller (OPB_EMC v1.10.b) on the processor OPB bus (used only for test)
- RS232 serial channel on the Spartan-3 Starter Board, connected to a UART peripheral (OPB_UARTlite v1.00.b) on the processor OPB bus, used for stdin (interrupt-driven) and stdout
- 8 LEDs on the Spartan-3 Starter Board, connected to a General Purpose I/O peripheral (GPIO v3.01.a) on the processor OPB bus
- 4 pushbuttons on the Spartan-3 Starter Board, connected to a General Purpose I/O peripheral (GPIO v3.01.a) on the processor OPB bus (used only for test)
- Timer/counter peripheral (OPB_Timer v1.00.b) on the processor OPB bus, used to generate interrupts at varying intervals
- Interrupt controller (OPB_Intc v1.00.c) on the processor OPB bus, used to manage multiple interrupts

The embedded processor system is implemented in a XC3S200-FT256-4 Spartan-3 FPGA device on the Spartan-3 Starter 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

Instructions to Run the Example Design

1. Connect Parallel Cable 4 between your host computer and the Spartan-3 Starter Board. Supply power to the Parallel Cable 4 using either the PS2 port of your host computer or external power supply.
2. Connect serial cable between your host computer and the Spartan-3 Starter Board.
3. Apply power to the Spartan-3 Starter 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.
7. Navigate to the directory where you expanded the MB_Spartan3_Tutorial_6_3 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 Spartan-3 Starter 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 SRAM_256Kx32:
  Running 32-bit test...
PASSED!
  Running 16-bit test...PASSED!
PASSED!
  Running 8-bit test...
PASSED!
Starting MemoryTest for opb_bram_if_cntlr_1:
  Running 32-bit test...
PASSED!
  Running 16-bit test...PASSED!
PASSED!
  Running 8-bit test...
PASSED!
```

```
-- Entering InterruptTest() --
```

The 8 LEDs should change to a repeating 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:

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 Spartan-3 Starter Board.
   d. For Board Revision, select the revision of the Spartan-3 Starter Board you are using.
   e. Click Next.
6. In BSB Select Processor, choose MicroBlaze (default); click Next.
7. In BSB Configure Processor:
   a. Under System Wide Settings, for Processor Bus Clock Frequency, select 50 MHz.
   b. Under Processor Configuration, for Debug Interface, choose On-chip HW Debug Module (default).
   c. For Local Data and Instruction Memory, select 8 KB (default).
   d. Click Next. (Do not enable the cache.)
8. In BSB Configure I / O Interfaces:
   a. Check RS232 (default); Peripheral = OPB UARTLITE (default); Baud Rate = 9600 (default); Data Bits = 8 (default); Parity = None (default). Check “Use Interrupt”.
   b. Check LEDs_8Bit (default); Peripheral = OPB_GPIO (default). (Do not check Use Interrupts.)
   c. Uncheck LED_7SEGMENT.

d. Check Push_Buttons_3Bit (default); Peripheral = OPB_GPIO (default). (Do not check Use Interrupts.)

e. Uncheck DIP_Switches_8Bit.

f. Click Next.

9. In BSB Configure Additional I/O Interfaces: check SRAM_256Kx32 (default); Peripheral = OPB EMC (default); click Next.

10. In BSB Add Internal Peripherals:

    a. Click Add Peripheral; select “OPB BRAM IF CNTLR”; click OK.

    b. For Memory Size, select 8 KB (default).

    c. Click Add Peripheral again; select OPB Timer; click OK.

    d. For Count Bit Width, select 32 (default).

    e. For Timer Mode, choose “One timer is present”.

    f. Check Use Interrupts.

    g. Click Next.

11. In BSB Software Configuration:

    a. Check Generate sample application and Linker Script (default).

    b. For Standard Input and for Standard Output, select RS232 (default).

    c. For Instructions, select “ilmb_cntlr” (default).

    d. For Data and for Stack/Heap, select “dlmb_cntlr” (default).

    e. Click Next.

12. The BSB System Created screen will appear summarizing your selections; click Generate. Click Finish.

13. In XPS, select Project -> Add SW Application Project; for Project Name, type “TestInterrupt”; click OK.

14. In XPS, in the Applications tab, right-click “Project: TestApp”; uncheck “Mark to Initialize BRAMs”.

15. Under “Project: TestApp”, expand Sources; double-click “.../TestApp.c”:

    a. Edit the program to add the InterruptTest() routine, the UART ISR and the Timer ISR, as shown in the listing below.

    b. 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, copy the TestInterrupt.c program file from this example design.

16. Under “Project: TestInterrupt”, right-click Sources; select Add File; browse to the TestInterrupt/src directory; select TestInterrupt.c; click Open.

    Note: This application runs using the default linker script.

17. 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;

do not enter a value for the UARTlite interrupt_handler in the Software Platform Settings dialog.

18. 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.c shown in **bold** (except comments).

```
/*
 * Xilinx EDK 6.3
 *
 * 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: microblaze_0/include/xparameters.h
#include "xparameters.h"

#include "xutil.h"
#include "xgpio_l.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 MicroBlaze based system.
 * This interrupt test routine has been added to the test application
(TestApp)
 * 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 */

#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 = 16777216; /* 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_BASEADDR)) {
      /* Read a character */
      c = XUartLite_RecvByte(XPAR_RS232_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");
   /* Enable MicroBlaze interrupts */
   microblaze_enable_interrupts();
   /* Register the UART interrupt handler in the vector table */
   XIntc_RegisterHandler(XPAR_OPB_INTC_0_BASEADDR,
      XPAR_OPB_INTC_0_RS232_INTERRUPT_INTR,
     (XInterruptHandler)uart_int_handler,(void *)XPAR_RS232_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_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 UART interrupts */
   XUartLite_mEnableIntr(XPAR_RS232_BASEADDR);

  /* Wait for interrupts to occur until exit_command received from UART
ISR */
   while (!exit_command)
   ;

   /* Disable MicroBlaze interrupts */
   microblaze_disable_interrupts();
   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--;
   }
```

```
}

/*
 * Routine to read data from a GPIO
 * which is configured as an input
 *   PARAMETER C_ALL_INPUTS = 1
 */
Xuint32 ReadFromGPInput(Xuint32 BaseAddress) {
   Xuint32 data = XGpio_mGetDataReg(BaseAddress, 1);
   return data;
}

//====================================================
int main (void) {
   print("-- Entering main() --\r\n");

   /*
    * MemoryTest routine will not be run for the memory at
    * dlmb_cntlr.C_BASEADDR
    * because it is being used to hold a part of this application program
    */

   /*
    * MemoryTest routine will not be run for the memory at
    * ilmb_cntlr.C_BASEADDR
    * because it is being used to hold a part of this application program
    */

   WriteToGPOutput(XPAR_LEDS_8BIT_BASEADDR, 8);

   {
      Xuint32 data = ReadFromGPInput(XPAR_PUSH_BUTTONS_3BIT_BASEADDR);
      //xil_printf("Data read from Push_Buttons_3Bit: 0x%x\n", data);
   }

   /* Testing EMC Memory (SRAM_256Kx32)*/
   {
      XStatus status;
      print("Starting MemoryTest for SRAM_256Kx32:\r\n");
      print("  Running 32-bit test...");
      status =
      XUtil_MemoryTest32((Xuint32*)XPAR_SRAM_256KX32_MEM0_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_SRAM_256KX32_MEM0_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_SRAM_256KX32_MEM0_BASEADDR,
        4096, 0xA5, XUT_ALLMEMTESTS);
        if (status == XST_SUCCESS) {
           print("PASSED!\r\n");
        }
        else {
           print("FAILED!\r\n");
        }
     }

     /* 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 MicroBlaze 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*.