Micromint, Inc	**AN812**
	MicroBolt
The MicroBolt as an I²C Slave	10/9/2005

Introduction:
This application notes demonstrates how to use the MicroBolt as an I²C Slave on the I²C serial bus.

Background:
The Philips 2 wire I²C Serial bus is a very popular 2 wire network. Many devices are available that support connection to this serial bus. Since the MicroBolt is based off from a Philips LPC2106 controller, its I²C implementation is highly integrated. For this reason, the MicroBolt can live on the I²C bus with very little user written code since most of the I²C functionality is built into the LPC2106 hardware.

For more information on the Philips I²C Serial bus visit the Philips site: http://www.semiconductors.philips.com/buses/i2c

How it works:
This ImageCraft ICCARM demo project utilizes the MicroBolt's I²C serial channel. The MicroBolt is setup as a slave on the I²C bus and its address is set to 10 (0x0A) for this example. All I²C parsing occurs in the I²C interrupt handler. The example code contains an I²C buffer (array of 20 bytes) that can be used for transmitting and receiving. An index points to the buffer allowing the buffer to be incremented or decremented at will.

For I²C slave receive data, 2 packets are used as an example; Packet 'A' and Packet 'B'. In this example, the first byte of an I²C packet will determine which type of packet it is ('A' or 'B'). Each packet has a certain length, so data is ACK'd or NAK'd accordingly.

For I²C slave transmit data, 1 packet is used as an example; Packet 'C'. The first byte sent out to the master will be a 'C'. This packet was defined to be 6 bytes long, so 6 bytes will be sent to the master upon request.

The I²C slave framework of this project allows the user to configure the I²C slave as needed. The I²C buffer, index pointer, and valid states have been properly setup and example packets show a simple I²C slave implementation.

This I²C slave framework has been used and tested on a few different I²C masters. One example master was a 32-bit microprocessor, and the other a serial to I²C adapter connected to a PC. This serial to I²C adapter is readily available from Emicros and can be found here: http://www.emicros.com/i2c232.htm. A visual basic driver from Pride Embedded is available for use with this adapter: http://www.prideembedded.com/, or you can use a simple terminal emulator session for testing with it.

Program Listing:

```
/*
|-------------------------------------------------------------------------------
|-------------------------------------------------------------------------------
| File Name                     : MicroBolt_I2C_Slave.c
| Author                        : Micromint, Inc.
| Copyright                     : Copyright © 2005, Micromint, Inc.
| Creation Date                 : 10/9/05
| Version                       : 1.00
| Spaces per tab                : 2
| Description                   : Main C file
| Revision                      : Initial
|-------------------------------------------------------------------------------
```

```
|-------------------------------------------------------------------------------------------
*/

/*
|-----------------------------------------------
|     Includes
|-----------------------------------------------
*/

#include <ARM/philips/lpc210x.h>
#include <arm_macros.h>
#include "MicroBolt_I2C_Slave.h"

static unsigned char I2cBuffer[20];
static unsigned char I2cBufferIndex = 0;

/*
|-------------------------------------------------------------------------------------------
|     Function        :   main
|     Inputs          :   None
|     Outputs         :   None
|     Purpose         :   Main function for system
|     Author          :   Micromint, Inc.
|-------------------------------------------------------------------------------------------
*/

void main(void)
  {
  __DISABLE_INTERRUPT();                        // Disable all interrupts

/*
|-------------------------
| Config MAM
|-------------------------
*/
  MAM_CR = 0x00;                                // Turn MAM off (default)
  MAM_TIM = 0x04;                               // Set flash timing to 4 clock cycles

  MAM_CR = 0x02;                                // Fully enable the Memory Accleration Module

/*
|-------------------------
| Config PLL and CCLK
|-------------------------
*/
  SCB_PLLCFG |= 0x23;                           // Set to 59 MHz (0x03 is multiply value of 4)
  SCB_PLLCON |= 0x01;                           // Enable the PLL
  SCB_PLLFEED = 0xAA;                           // Shadow register copy to enable changes
  SCB_PLLFEED = 0x55;                           // in PLLCON and PLLCFG

/*
|-------------------------
| Config PCLK
|-------------------------
*/
  SCB_VPBDIV = 0;                   // Peripheral clock is 1/4th Processor clock which equals 14.7456 MHz

/*
|-------------------------
| Configure VIC
|-------------------------
*/
  VICVectAddr0 = (unsigned)pll_isr;             // Assign the PLL lock ISR vector address
  VICVectCntl0 = INTERRUPT_CHANNEL_FOR_PLL;     // Assign the VIC address to the actual interrupt
  VICIntEnable = INTERRUPT_ENABLE_FOR_PLL;      // Enable the interrupt

  VICVectAddr1 = (unsigned)I2c_ISR;              // Assign the I2C ISR vector address
  VICVectCntl1 = INTERRUPT_CHANNEL_FOR_I2C;     // Assign the VIC address to the actual interrupt
  VICIntEnable = INTERRUPT_ENABLE_FOR_I2C;      // Enable the interrupt

/*
|-------------------------
| Config GPIO
|-------------------------
```

```
*/
  PCB_PINSEL0=0x00000050;                       // Setup with ICCARM App builder - MicroBolt_I2C.bcf (I2C)
  PCB_PINSEL1=0x55400000;                       //                                  (Secondary JTAG pins)

  GPIO_IODIR |= MICROBOLT_LED;                  // Setup MicroBolt LED as output

  GPIO_IOCLR=0xffffffff;                        // Clear all pins to start with

/*
|-------------------------
| Config I2C
|-------------------------
*/
  I2C_I2SCLH=100;                               // I2C clock
  I2C_I2SCLL=100;
  I2C_I2ADR= 0x00000014;                        // Address 10 (0x0A << 1)
  I2C_I2CONSET=0x00000044;                      // Setup flags

  __ENABLE_INTERRUPT();                         // Enable all interrupts

/*
|-----------------------------------------------------
|     Start of application
|-----------------------------------------------------
*/

  while(1)                                      // Do this forever
    {
    }
  }

/*
|-------------------------------------------------------------------------------------------
|     Function        :   pll_isr
|     Inputs          :   None
|     Outputs         :   None
|     Purpose         :   Once PLL has locked, connect it and use for system clock
|     Author          :   Micromint, Inc.
|-------------------------------------------------------------------------------------------
*/

#pragma interrupt_handler pll_isr

void pll_isr(void)
  {
  SCB_PLLCON |= 0x02;                           // Connect the PLL
  SCB_PLLFEED = 0xAA;                           // Shadow register copy to enable changes
  SCB_PLLFEED = 0x55;                           // in PLLCON and PLLCFG
  VICIntEnClear = PLL_CLR;                      // Disable the PLL interrupt, not needed anymore
  VICVectAddr = VIC_ACK;                        // Acknowledge Interrupt
  }

/*
|-------------------------------------------------------------------------------------------
|     Function        :   I2c_ISR
|     Inputs          :   None
|     Outputs         :   None
|     Purpose         :   I2C interrupt and command processing
|     Author          :   Micromint, Inc.
|-------------------------------------------------------------------------------------------
*/

#pragma interrupt_handler I2c_ISR

void I2c_ISR(void)
  {

  See ImageCraft ICCARM demo project for the rest of the code

  }
```