Interfacing to the On-board ADS7843 Touch Controller

Application Note 175

Andrew Wawra, Hans Rempel, Russell McGuire

Logic Product Development

Published: July 2004

Abstract

By using the procedures presented in this document, you can communicate to the touch chip via SPI. This document provides flowcharts, basic register settings, and special case scenarios.

This file contains source code, ideas, techniques, and information (the Information) which are Proprietary and Confidential Information of Logic Product Development, Inc. This information may not be used by or disclosed to any third party except under written license, and shall be subject to the limitations prescribed under license.

No warranties of any nature are extended by this document. Any product and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed license or agreement to purchase or lease equipments. The only warranties made by Logic Product Development, if any, with respect to the products described in this document are set forth in such license or agreement. Logic Product Development cannot accept any financial or other responsibility that may be the result of your use of the information in this document or software material, including direct, indirect, special or consequential damages.

Logic Product Development may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering the subject matter in this document. Except as expressly provided in any written agreement from Logic Product Development, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

© Copyright 2005, Logic Product Development, Inc. All Rights Reserved.

REVISION HISTORY

REV	EDITOR	DESCRIPTION	APPROVAL	DATE
A	Russell McGuire	Release	HAR	7/13/04
B	James Wicks	Removed SH7750R Reference	JAW	7/28/04
C	Bruce Rovner	Generalized document for all Sharp SOMs; Removed barrier function example; Removed reference to “Read Pen State Flow Chart”; General document editing	BR	1/3/06

1 Introduction

The following procedures provide examples for communicating to the ADS7843 over the SPI (Serial Peripheral Interface). For interfacing to the touch chip, see Texas Instruments (Burr-Brown) ADS7843 data sheet for the complete command set and registers. Also see Logic's card engine-specific IO Controller Specification documents for SPI Data and SPI Control Register bit definitions.

Card engine chipsets applicable to this document include the following:

- LH7A400-10
- LH7A404-11
- LH75401-11
- LH79520-10
- LH79524-10
- SH7760-10

Apart from those listed above, Logic's card engines use on-chip touch controllers or other controllers.

2 LH7XXXX: Communicating to the Touch Chip via SPI

This section only applies to the LH7XXXX series of processors from Sharp.

Logic's IO Controller Specification documents were written before we were aware of this Sharp application note: "Interfacing the Static Memory Controller with I/O Devices," available on the Sharp website http://www.sharpsma.com.

Since then, Logic has added a barrier function between each access to the SPI interface in the CPLD (Complex Programmable Logic Device) in order to make sure that the chip select line (/CS) toggles. Please refer to Logic's Application Note 303: *Interfacing to IO Devices via the Static Memory Controller on LH7xxxx Card Engines* for an example of the barrier function that must be used between accesses to the SPI interface registers in the CPLD. This document can be found at: http://www.logicpd.com/downloads/684/

3 Touch SPI Driver Flow Charts

These flow charts present the way Logic Product Development has implemented driver code. The flowcharts may be used as examples for creating custom code. The barrier functions referenced in section 2 of this document (Application Note 303: *Interfacing to IO Devices via the Static Memory Controller on LH7xxxx Card Engines*) are not displayed in the flow charts below.

3.1 Main State Machine: Main ISR Flow Chart

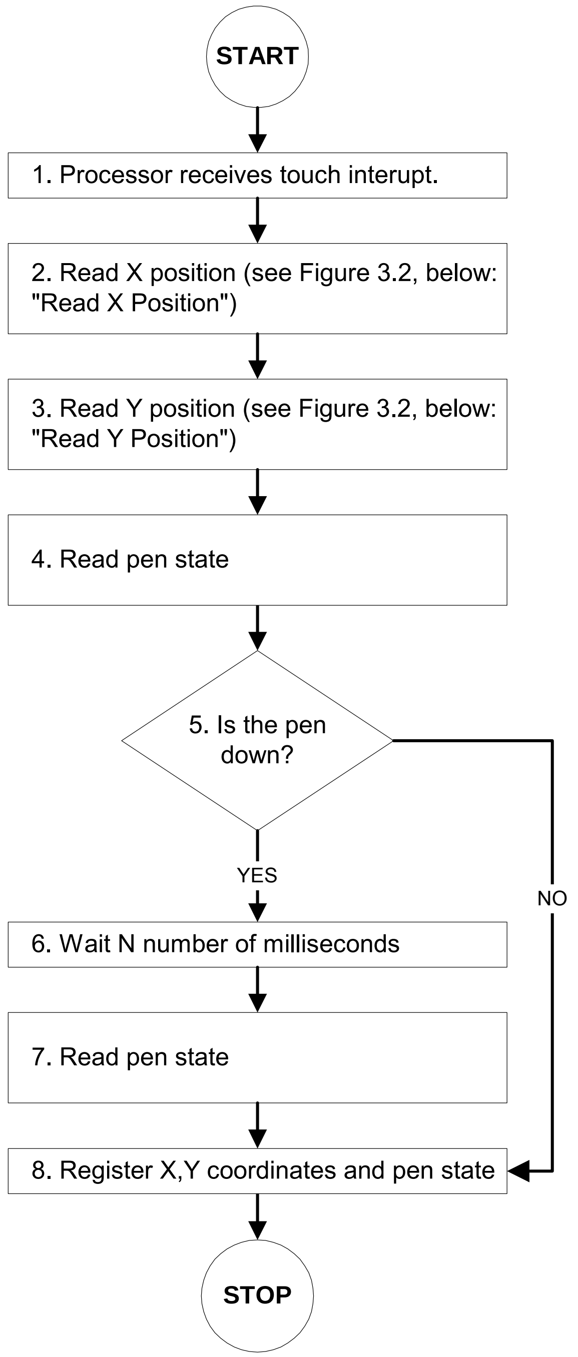
START
1. Processor receives touch interupt.
2. Read X position (see Figure 3.2, below: "Read X Position")
3. Read Y position (see Figure 3.2, below: "Read Y Position")
4. Read pen state
5. Is the pen down?
YES
NO
6. Wait N number of milliseconds
7. Read pen state
8. Register X,Y coordinates and pen state
STOP

Figure 3.1: Main ISR Flow Chart

3.2 Detail of Figure 3.1: Steps 2 - 3 in the Main State Machine Main ISR Flowchart

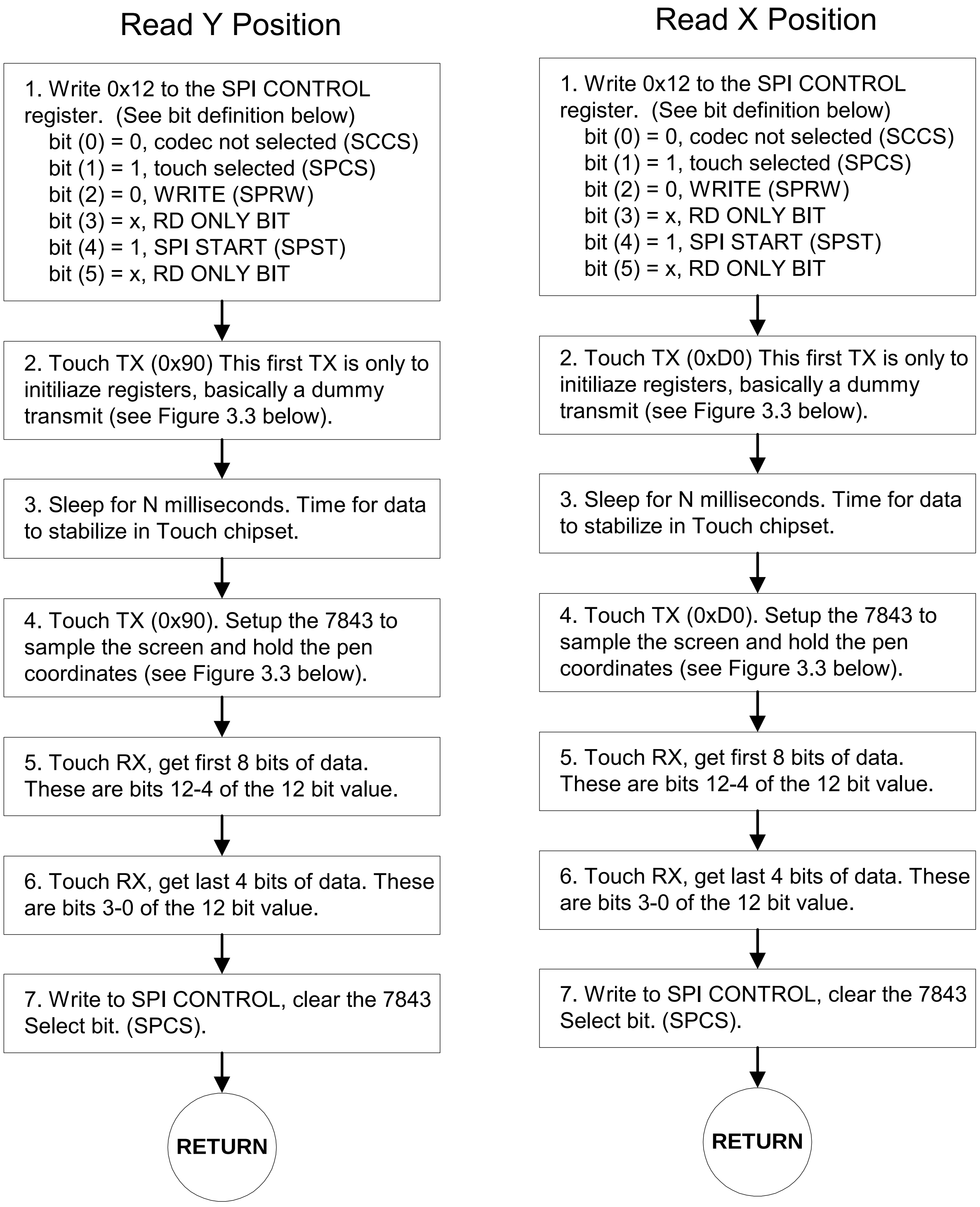
Read Y Position
1. Write 0x12 to the SPI CONTROL register. (See bit definition below)
bit (0) = 0, codec not selected (SCCS)
bit (1) = 1, touch selected (SPCS)
bit (2) = 0, WRITE (SPRW)
bit (3) = x, RD ONLY BIT
bit (4) = 1, SPI START (SPST)
bit (5) = x, RD ONLY BIT
2. Touch TX (0x90) This first TX is only to initiliaze registers, basically a dummy transmit (see Figure 3.3 below).
3. Sleep for N milliseconds. Time for data to stabilize in Touch chipset.
4. Touch TX (0x90). Setup the 7843 to sample the screen and hold the pen coordinates (see Figure 3.3 below).
5. Touch RX, get first 8 bits of data. These are bits 12-4 of the 12 bit value.
6. Touch RX, get last 4 bits of data. These are bits 3-0 of the 12 bit value.
7. Write to SPI CONTROL, clear the 7843 Select bit. (SPCS).
RETURN
Read X Position
1. Write 0x12 to the SPI CONTROL register. (See bit definition below)
bit (0) = 0, codec not selected (SCCS)
bit (1) = 1, touch selected (SPCS)
bit (2) = 0, WRITE (SPRW)
bit (3) = x, RD ONLY BIT
bit (4) = 1, SPI START (SPST)
bit (5) = x, RD ONLY BIT
2. Touch TX (0xD0) This first TX is only to initiliaze registers, basically a dummy transmit (see Figure 3.3 below).
3. Sleep for N milliseconds. Time for data to stabilize in Touch chipset.
4. Touch TX (0xD0). Setup the 7843 to sample the screen and hold the pen coordinates (see Figure 3.3 below).
5. Touch RX, get first 8 bits of data. These are bits 12-4 of the 12 bit value.
6. Touch RX, get last 4 bits of data. These are bits 3-0 of the 12 bit value.
7. Write to SPI CONTROL, clear the 7843 Select bit. (SPCS).
RETURN

Figure 3.2

3.3 Detail of Figure 3.2: Single Data Transmit or Receive From the Touch Chip

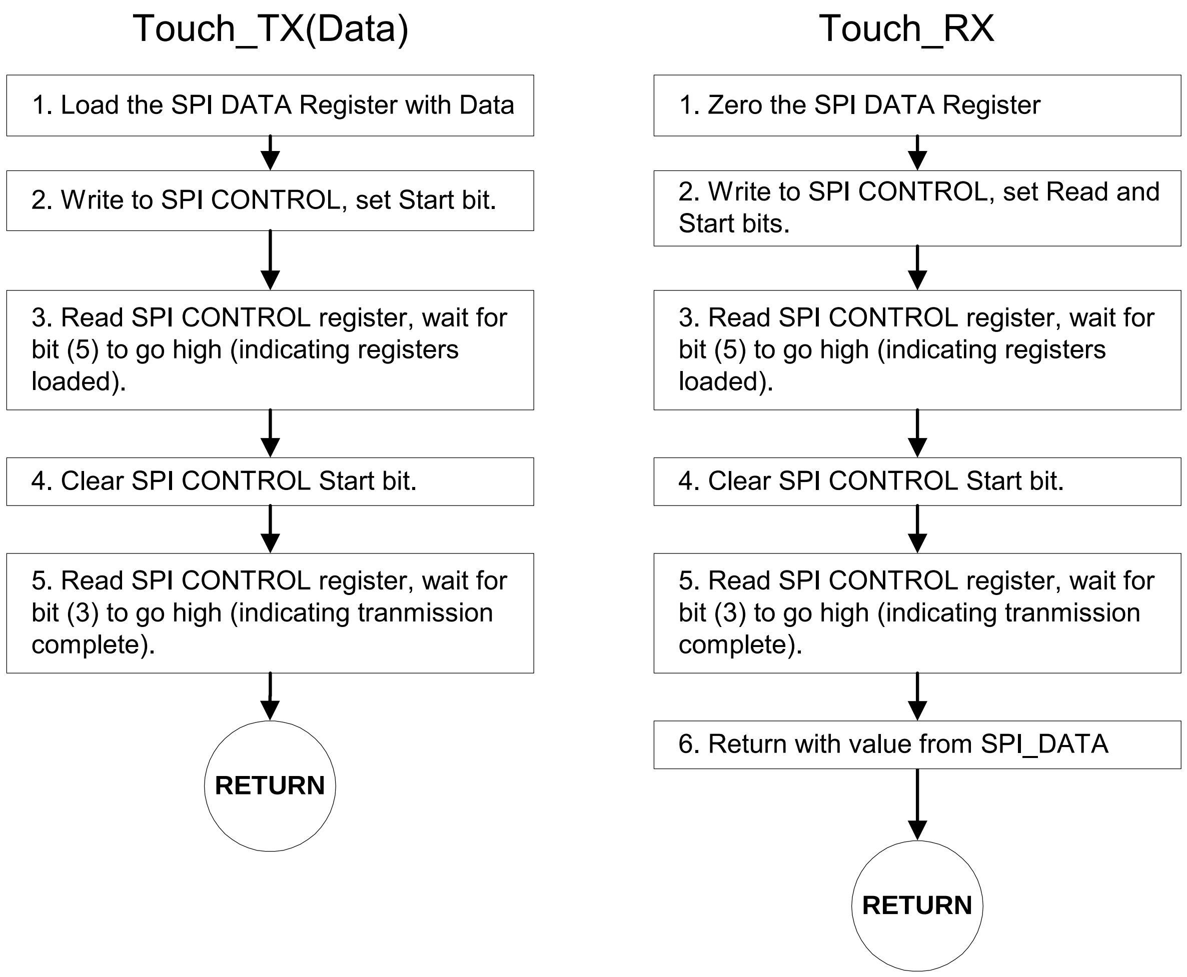
Touch_TX(Data)
1. Load the SPI DATA Register with Data
2. Write to SPI CONTROL, set Start bit.
3. Read SPI CONTROL register, wait for bit (5) to go high (indicating registers loaded).
4. Clear SPI CONTROL Start bit.
5. Read SPI CONTROL register, wait for bit (3) to go high (indicating tranmission complete).
RETURN
Touch_RX
1. Zero the SPI DATA Register
2. Write to SPI CONTROL, set Read and Start bits.
3. Read SPI CONTROL register, wait for bit (5) to go high (indicating registers loaded).
4. Clear SPI CONTROL Start bit.
5. Read SPI CONTROL register, wait for bit (3) to go high (indicating tranmission complete).
6. Return with value from SPI_DATA
RETURN

Figure 3.3

4 Windows CE Driver Theory of Operation

Software structure of the touch system is as follows. The application layer interfaces with the GWES (Graphics, Windowing, and Events Subsystem) layer in Windows CE, which in turn loads the touch driver provided by a third party, in this case, Logic Product Development. This DLL sits on top of the OAL (OEM Adaptation Layer), also provided by Logic. The DLL talks directly with the hardware, utilizing the assistance of the OAL only for loading and unloading.

The hardware structure is completed with the card engine CPU communicating with the Logic CPLD IO device over the local bus. The CPLD handles the SPI communication with the touch device via the SPI bus.

The touch driver has the responsibility to read the data from the 7843 touch chip; higher layers in the software model calibrate the data and do conversion from the raw data into screen coordinates.

Logic's touch controller ADC has a 12 bit sampling register. Therefore, the range from low end to high end would ideally be 0 to 4096. Because of offset error introduced by the inherent resistance of the measuring circuitry, the values will actually be something more like 0 to 4000 or more generically X to X + Y.

The touch Interrupt Service Thread waits for a touch interrupt to occur, i.e. it waits for a pen-down event. Once this is detected, control of the SPI bus is sought since it is shared between two devices. This is done by asserting the touch chip select signal. When ownership of the SPI bus is obtained, X and Y read commands are sent to the touch chip to read the X and Y point that have been touched. The state machine, which takes the X and Y readings, ignores the first few readings to avoid response to spurious touches. This state machine then returns raw data to the GWES layer in the operating system.

5 Further Reading

For a more in depth understanding of the individual chipsets please refer to the technical documents furnished by Texas Instruments on the ADS7843, as well as Logic's documents applicable to the specific card engine that is being used with the ADS7843.

- Logic's up-to-date card engine-specific manuals can be found online at:

http://www.logicpd.com

- Need additional help? Please contact us through our website, and refer to our technical discussion group and FAQ's available online at:

http://www.logicpd.com/support

6 Appendix A: Example Code from Logic's SH7760-10 Driver

```
/*!--------------------------------------------------------------------------
 * \file       ADS7843_macros.c
 * \brief      Memory Utilities
 */
/* © Copyright 2002, Logic Product Development, Inc. All Rights Reserved.
 *
 * NOTICE:
 *  This file contains source code, ideas, techniques, and information
 *  (the Information) which are Proprietary and Confidential Information
 *  of Logic Product Development, Inc.  This Information may not be used
 *  by or disclosed to any third party except under written license, and
 *  shall be subject to the limitations prescribed under license.
 *
 *--------------------------------------------------------------------------*/

unsigned short spi_ctrl_reg_shadow, touch_sample, tmp;

#define CPLD_SPI_CONTROL_REG          (MEMORY MAPPED ADDRESS OF THE CPLD SPI CONTROL REGISTER)
#define CPLD_SPI_DATA_REG             (MEMORY MAPPED ADDRESS OF THE CPLD SPI DATA REGISTER)
#define CPLD_INTERRUPT_REG            (MEMORY MAPPED ADDRESS OF THE CPLD INTERRUPT/MASK REGISTER)

#define CPLD_TOUCH_CS         0x0002
#define CPLD_XFER_DONE        0x0008
#define CPLD_SPI_READ         0x0004
#define CPLD_SPI_START        0x0010
#define CPLD_SPI_LOADED       0x0020
#define CPLD_TOUCH_INT        0x0002
#define CPLD_TOUCH_PIRQ       0x0010

#define TOUCH_START           0x80
#define TOUCH_Y                       0x50
#define TOUCH_X                       0x10
#define TOUCH_8BIT            0x08
#define TOUCH_SERDF                   0x04
#define TOUCH_DIS_INT         0x01
#define TOUCH_POWERED         0x03

//#define TOUCH_X_SAMPLE      (TOUCH_START | TOUCH_X | TOUCH_SERDF | TOUCH_POWERED)
#define TOUCH_X_SAMPLE        (TOUCH_START | TOUCH_X)// | TOUCH_SERDF)
//#define TOUCH_Y_SAMPLE      (TOUCH_START | TOUCH_Y | TOUCH_SERDF | TOUCH_POWERED)
#define TOUCH_Y_SAMPLE        (TOUCH_START | TOUCH_Y)// | TOUCH_SERDF)


#define TCH_READ_AD_X(v) {

                spi_ctrl_reg_shadow = CPLD_TOUCH_CS;

                WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);
                touch_tx(TOUCH_X_SAMPLE);

                Sleep(1);

                touch_tx(TOUCH_X_SAMPLE);

                touch_sample = touch_rx();

                touch_sample <<= 5;

                touch_sample |= (touch_rx() >> 3);

                touch_sample &= 0x0FFF;

                spi_ctrl_reg_shadow &= ~CPLD_TOUCH_CS;

                WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);
                v = touch_sample;
```

```
}


#define TCH_READ_AD_Y(v) {

                    spi_ctrl_reg_shadow = CPLD_TOUCH_CS;

                    WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);
                    touch_tx(TOUCH_Y_SAMPLE);

                    Sleep(1);

                    touch_tx(TOUCH_Y_SAMPLE);

                    touch_sample = touch_rx();
                              touch_sample <<= 5;

                    touch_sample |= (touch_rx() >> 3);

                    touch_sample &= 0x0FFF;

                    spi_ctrl_reg_shadow &= ~CPLD_TOUCH_CS;

                    WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);
                    v = touch_sample;

}



#define TCH_READ_PEN_STATE(v)         { /* v is 1 if pen is up, else 0 */
          if (!(READ_REGISTER_USHORT(CPLD_INTERRUPT_REG) & CPLD_TOUCH_INT)) v=0;
                    else v=1;

          }



void
touch_tx(unsigned char xmit_char)
{

                    // write data to data reg
                    WRITE_REGISTER_USHORT(CPLD_SPI_DATA_REG,(unsigned short)xmit_char);

//                  xprintf("Wrote 0x%X to data register\r\n",xmit_char);

//                  spi_ctrl_reg_shadow |= CPLD_TOUCH_CS;
//                  WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);

                    // set the start bit so the data will load
                    spi_ctrl_reg_shadow |= CPLD_SPI_START;
                    WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);

//                  xprintf("Set start bit and touch chip select\r\n");

                    // poll load bit (tells us when the data has been loaded in the spi data register)
                    while (!(READ_REGISTER_USHORT(CPLD_SPI_CONTROL_REG) & CPLD_SPI_LOADED));

//                  xprintf("Load bit is set\r\n");

                    // clear start bit (will enable transmission/reception)
                    spi_ctrl_reg_shadow &= ~CPLD_SPI_START;
                    WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);

//                  xprintf("Cleared start bit, tx should go!\r\n");

                    // poll done bit
                    while (!(READ_REGISTER_USHORT(CPLD_SPI_CONTROL_REG) & CPLD_XFER_DONE));
```

```
//                  xprintf("Finished polling done bit.\r\n");

//                  spi_ctrl_reg_shadow &= ~CPLD_TOUCH_CS;
//                  WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);

}



unsigned char
touch_rx(void)
{
unsigned char RetVal;

                    // write data to data reg
                    WRITE_REGISTER_USHORT(CPLD_SPI_DATA_REG,(unsigned short)0x0000);

//                  spi_ctrl_reg_shadow |= CPLD_TOUCH_CS;
//                  WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);

                    // set touch CS to 1 and set the start bit so the data will load
                    spi_ctrl_reg_shadow |= CPLD_SPI_START | CPLD_SPI_READ;
                    WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);

//                  xprintf("Set start bit\r\n");

                    // poll load bit (tells us when the data has been loaded in the spi data register)
                    while (!(READ_REGISTER_USHORT(CPLD_SPI_CONTROL_REG) & CPLD_SPI_LOADED));

//                  xprintf("Load bit is set\r\n");

                    // clear start bit (will enable transmission/reception)
                    spi_ctrl_reg_shadow &= ~CPLD_SPI_START;
                    WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);

//                  xprintf("Cleared start bit, rx should go!\r\n");

                    // poll done bit
                    while (!(READ_REGISTER_USHORT(CPLD_SPI_CONTROL_REG) & CPLD_XFER_DONE));

//                  xprintf("Finished polling done bit.\r\n");

                    RetVal = (unsigned char)READ_REGISTER_USHORT(CPLD_SPI_DATA_REG);

//                  spi_ctrl_reg_shadow &= ~CPLD_TOUCH_CS;
//                  WRITE_REGISTER_USHORT(CPLD_SPI_CONTROL_REG,spi_ctrl_reg_shadow);

                    return RetVal;

}
```

```
/******************************************************************************
*   FUNCTION :              tchStateMachine
*   DESCRIPTION :   Returns whether we have a valid sample
*   INPUTS :                none, assume we're called because an interrupt happened
*   OUTPUTS :                               Returns whether a valid {x,y} sample has been read.
*   DESIGN NOTES :
*   CAUTIONS :
******************************************************************************/
static bool
tchStateMachine(void)
{

        unsigned int tmp;

                /* read the X position */
                TCH_READ_AD_X(g_xpos);

                Sleep(3); /* sleep for around 3mS */

                /* read the Y position */
                TCH_READ_AD_Y(g_ypos);

                /* get the pen state */
                TCH_READ_PEN_STATE(tmp);

                /* if the pen is down, we put a delay here in-between samples */
                if (!tmp) {
                        status = TCH_PEN_DOWN;
                        Sleep(20);

                /* adding this here because sometimes the pen goes up after reading the value */
                        TCH_READ_PEN_STATE(tmp);

                        if (tmp) {
                                status = TCH_PEN_UP;
//                              TCHMSG(1,(TEXT("Xpos - 0x%X, Ypos - 0x%X\r\n"),g_xpos,g_ypos));
                        }


                } else {
                        status = TCH_PEN_UP;
//                      TCHMSG(1,(TEXT("Xpos - 0x%X, Ypos - 0x%X\r\n"),g_xpos,g_ypos));
                }


                /* set up to get another sample */
                state = WAIT_INTR;

                /* return */
                return 1;

}
```