XPT2046 Touch Controller Interface Manual

Michael Smith

December, 2024

Contents

1 Introduction

This document serves is a comprehensive reference manual for interfacing the XPT2046 touch controller with the Nexys A7 FPGA board. It includes detailed descriptions of hardware setup, software design, and implementation considerations based on the XPT2046 datasheet.

2 Hardware Overview

2.1 XPT2046 Touch Controller

The XPT2046 is a 4-wire resistive touch screen controller capable of providing precise X and Y touch coordinates over SPI communication. Key features include:

Relevant Datasheet Information:

- 12-bit resolution for X, Y, and pressure measurements.
- SPI communication with a maximum clock frequency of 2 MHz.
- Typical DCLK cycle time of 200 ns.
- Single supply voltage of 2.7V to 5.5V.

Outside the Datasheet:

- **Commands:** `0x90` (X-axis), `0xD0` (Y-axis), `0xB0` (Z1), and `0xC0` (Z2) initiate ADC conversions for the respective coordinates.
- **Timing:** Ensure the SPI and the LCD Display clocks respects the 200 ns minimum DCLK period and can run properly at 2 MHz when paired with the XPT2046.
- **Power:** Operates at 3.3V, compatible with the Nexys A7's voltage levels.

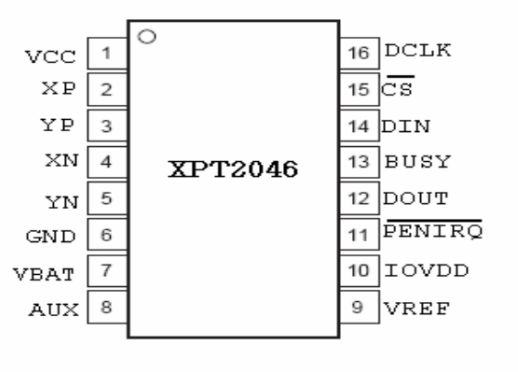


Figure 1: XPT2046 Pins

2.2 Nexys A7 FPGA Board

The Nexys A7, powered by an Artix-7 FPGA, provides the necessary GPIOs and AXI Quad SPI block for interfacing with the XPT2046. The FPGA's flexibility allows for precise timing control and efficient processing of touch data.

3 Hardware Connections

3.1 Vivado Setup

This project utilizes the AXI Quad SPI IP block to handle the communication bus with the XPT2046. There are four pins of the SPI block that need to map to the touch controller:

- **sck_o:** The system clock output pin.
- **ss_o[0:0]** The slave select pins.
- **io0_o:** The Master Out Slave In output pin.
- **io1_i** The Master In Slave Out input pin.

Refer to Figure 2 to find these on the Quad SPI block and make them external. By doing this, the external pins of the SPI can be mapped to the Pmod using the constraints file. Additionally, a Clocking Wizard is needed to maintain a 2 MHz output clock. Do this by adding the Clocking Wizard IP shown in Figure 3 and connecting its input(clk_in1) to the main system clock(ui_clk) and its output clock(clk_out1) to the SPI's ext_spi_clk. Customizing the Clocking Wizards output clock frequency to 8 MHz and the SPI's frequency ratio to 4 will produce the maximum required 2 MHz clock for the XPT2046 to run.

3.2 XPT2046 Connections

- **T_CLK or DCLK(Clock):** Connected to SPI sck_o.
- **T_CS or CS(Chip Select):** Connected to the SPI ss_o[0:1] pins.
- **T_DIN or DIN(MOSI):** Connected to SPI MOSI which is the io0_o pin.
- **T_DO or DO(MISO):** Connected to SPI MISO which is the io1_i pin.

- **T_IRQ or PENIRQ(Interrupt):** Connected to a GPIO for touch detection (active-low) which would be the gpio_io_i pin.

Figure 4 below shows a proper wiring schematic for the touch controller pins paired with an LCD display. Use one of the Pmods on the Nexys boards to connect these wires depending on your hardware constraints files. If the SPI is handling multiple slave for both the LCD screen and the touch, they will share the same MOSI, MISO, and Clock lines. It is possible to have separate SPI's for each controller, in that case the wiring for MOSI, MISO, and Clock would be mapped seperately.

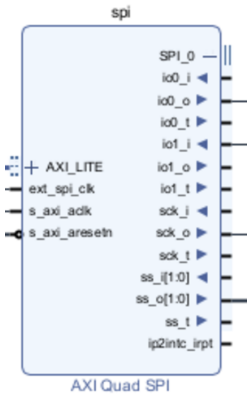


Figure 2: Quad SPI block

Figure 3: Quad SPI block

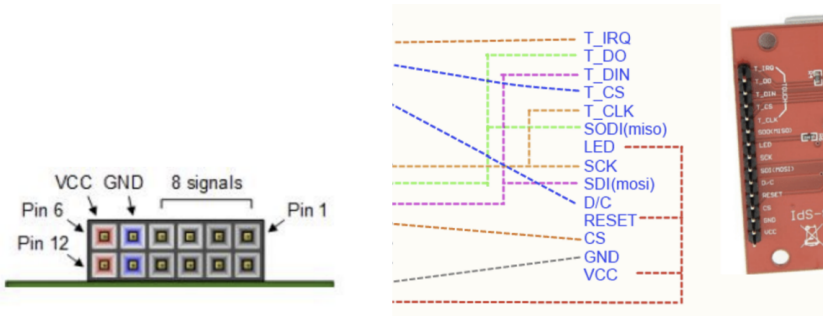


Figure 4: HiLetGo LCD ili9342 and xpt2046 pin wiring to Nexys Pmod

4 Software Implementation

The systems software initializes the SPI instance and then starts polling to detect touches. The polling is done using a while loop and monitors the Z pressure data from the XPT2046. To grab data from the touch, the XPT2046 slave is detected and then a transfer is performed using a control byte. When a touch is detected, additional transfers are sent to grab the X and Y coordinates.

4.1 Initialization

The software initializes the AXI Quad SPI block for master mode with manual slave select. Key Initialization Steps:

- Initialize the SPI:

Listing 1: SPI Initializationl

```
XSpi_Config *spiConfig; /* Pointer to Configuration data */
u32 status;

spiConfig = XSpi_LookupConfig(XPAR_SPI_DEVICE_ID);
if (spiConfig == NULL) {
xil_printf("Can't find spi device!\n");
return XST_DEVICE_NOT_FOUND;
}

status = XSpi_CfgInitialize(&spi, spiConfig, spiConfig->BaseAddress);
if (status != XST_SUCCESS) {
xil_printf("Initialize spi fail!\n");
return XST_FAILURE;
}
```

- Configure the SPI control register to enable master mode and slave select using either XSpi_SetOption() or XSpi_getControlReg():

Listing 2: SPI Initializationl

```
XSpi_SetOptions(&spi, XSP_MASTER_OPTION | XSP_MANUAL_SSELECT_OPTION);
```

Listing 3: SPI Initializationl

```
controlReg = XSpi_GetControlReg(&spi);
XSpi_SetControlReg(&spi,
(controlReg | XSP_CR_ENABLE_MASK | XSP_CR_MASTER_MODE_MASK) &
(~XSP_CR_TRANS_INHIBIT_MASK));
```

- Start the SPI and disable any global interrupts that may interfere with the communication bus:

Listing 4: SPI Initializationl

```
XSpi_Start(&spi);
XSpi_IntrGlobalDisable(&spi);
```

With these steps, the SPI driver should be properly configured to handle the touch controller communication. If not, SPI library functions like XSpi_GetOptions(), XSpi_GetControlReg(), and XSpi_GetStatusReg() can be used to check the driver status. Detecting touch is possible through polling either the Z pressure data or the T_IRQ/PENIRQ interrupt. If using the interrupt, make sure the PENIRQ doesn't effect the clock or otherwise it will disrupt the transfer. Below is how this project used polling to detect touches. The is_touched() and get_touch_coordinates() functions simply initialize a transfer using a control byte according to the data expected to recieve ie. X, Y, or Z.

Listing 5: SPI Initializationl

```
//Polling loop for touches
while (1) {
//Check z data for touches
if (is_touched(&z1, &z2)) {
get_touch_coordinates(&x, &y);//get coordinates when touched
printf("Touch detected: X=%d, Y=%d, Z1=%d, Z2=%d\n", x, y, z1, z2);
printToScr(x, y);//Print coordinates to the display
} else {
printf("No touch detected.\n");
}
}
```

4.2 XPT2046 Touch Controller Commands

The following commands are used to retrieve touch coordinates:

- `0x90`: Reads the X-coordinate.
- `0xD0`: Reads the Y-coordinate.
- `0xB0`: Reads the Z1-pressure.
- `0xC0`: Reads the Z2-pressure.

A command represents a specific control byte for the touch controller to read from. Each control byte initiates an ADC conversion, use the XPT2046 data sheet to varify these. The results are retrieved as 12-bit values, which requires two SPI transfers. The entire transaction is performed using XSpi_Transfer(), passing both the command and an empty buffer for the received data. Make sure the 12 bits returned are converted corrected.

4.3 Code Highlights

Below is an example function that implements the transfer. The command parameter represents the control byte mentioned earlier to retrieve specific values from the touch:

Listing 6: Touch Coordinate Retrieval

```
uint16_t TouchTransfer(uint8_t command) {
    uint8_t txBuffer[3] = {command, 0x00, 0x00};
    uint8_t rxBuffer[3] = {0x00, 0x00, 0x00};
    uint16_t coordinate;

    XSpi_SetSlaveSelect(&SpiInstance, Touch_Slave_Pin); // Select the touch
    XSpi_Transfer(&SpiInstance, txBuffer, rxBuffer, 3);
    XSpi_SetSlaveSelect(&SpiInstance, 0x00); // Deselect the touch

    coordinate = ((rxBuffer[0] << 8) | rxBuffer[1]) >> 4;
    return coordinate;
}
```

The function utilizes the XSpi_SetSlaveSelect() to select and deselect the touch slave. If the SPI is setup to handle multiple slaves, it is important to properly select and deselect when talking to different slaves. The transfer using a transmission buffer to send the control byte and a receiving buffer that the touch fills with the requested data. Because the XPT2046 returns 12 bits of data, make sure it is properly shifted so that it is accurate.

4.4 Interrupt and Polling

Polling of T_IRQ ensures efficient use of resources. If interrupts are desired, the GPIO interrupt handler must be configured to detect low signals on T_IRQ. Without interrupts, touches can be detected using a polling method while checking Z1 and Z2 pressure data. The below function returns true if a press has occurred based on the Z1 and Z2 data received after a transfer. Polling this inside a while loop will allow you to request and receive X and Y coordinates when a touch has occurred.

Listing 7: Touch Coordinate Retrieval

```
int is_touched(uint16_t *z1, uint16_t *z2) {
    *z1 = TouchTransfer(0xB0);
    *z2 = TouchTransfer(0xC0);

    // Simple pressure check
    return (*z1 > 10 && *z2 < 2000);
}
```

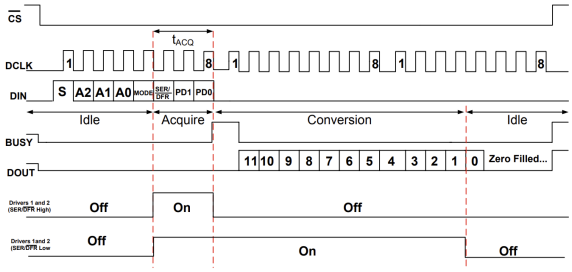
5 Test and Validation

5.1 Loopback Tests

To verify SPI communication, a loopback test was performed by shorting MISO and MOSI. The successful data echo confirms the SPI io0_o and io1_i pins setup. Replicating this involves directly connecting the MISO and MOSI pins and observing the bytes sent and received. If the MOSI and MISO of the SPI is set up correctly, the data sent will be the same as the data received.

5.2 Logic Analyzer Outputs

When testing with a Logic Analyzer, the following should be observed over the lines when data is being successfully transferred.



- **MOSI or DIN:** Command bytes (e.g., `0x90` and `0xD0`).
- **MISO or DO:** 12-bit ADC results for X and Y coordinates.
- **SCK or DCLK:** Stable clock signal matching the configured frequency of 2 MHz.

5.3 Software Testing

The best way to test the software is to monitor the SPI driver before, after and during a transfer. There are many ways to do this in the Xilinx spi driver library. This project has a function test_driver() that displays the mode and status of the SPI. When pairing the XPT2046 with another slave like an LCD, checking the FIFO status is helpful because it can disrupt tranfers when it is full. The code below is used to check the FIFO of the SPI.

Listing 8: Touch Coordinate Retrieval

```
u32 statusReg = XSpi_GetStatusReg(&spi);
xil_printf("SPI Status Register: 0x%08X\n", statusReg);

if (statusReg & XSP_SR_TX_FULL_MASK) {
xil_printf("SPI transmit FIFO is full.\n");
}

if (statusReg & XSP_SR_RX_EMPTY_MASK) {
xil_printf("SPI receive FIFO is empty.\n");
}
```

The SPI library also allows checks to the SPI control register and options. This can help determine if your SPI initialize and configuration correct.

6 Conclusion

The integration of the XPT2046 touch controller with the Nexys A7 FPGA was successfully achieved. The entire software can be found in the touch.c file in project github repository under References. If you are using Xilinx Vitis, downloading the Final_Demo.ide.zip and importing it into a Vitis workspace will provide a functioning software. Challenges such as SPI timing mismatches and touch inconsistencies were resolved through careful debugging and adherence to datasheet specifications. Future improvements include interrupt-driven touch detection and multiple touch recognition.

References & Manuals

- Github Project Repository
- PaulStroffregen's XPT2046 Touchscreen Project
- Xilinx SPI Driver
- Austinlightguy's Blog
- ILI9341 Datasheet
- XPT2046 Datasheet
- HiLetGo 240x320 SPI TFT ILI9342 LCD Display
- Nexys A7 FPGA Trainer Board