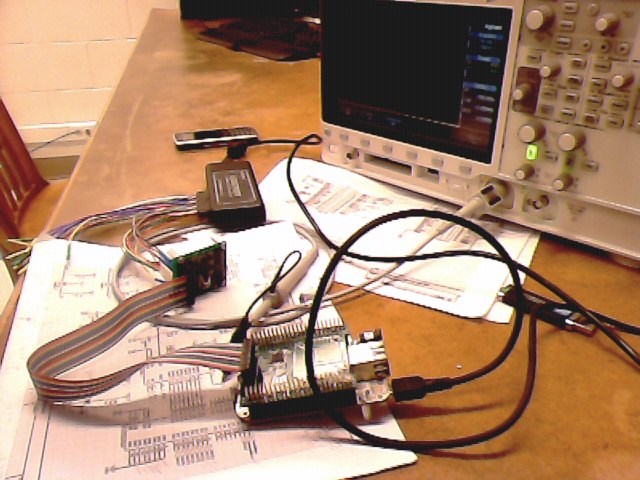
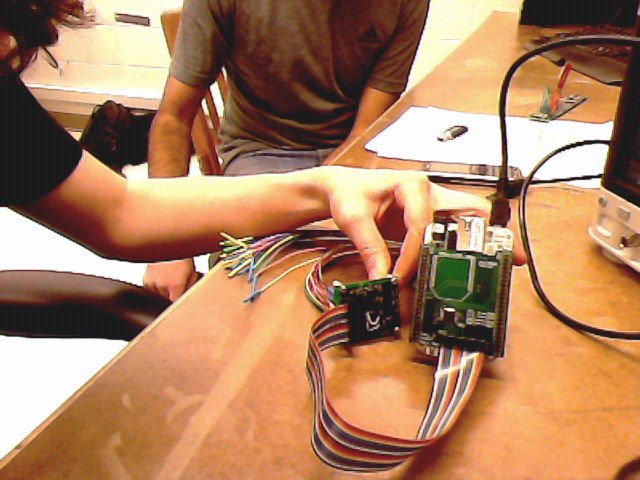


Camera Proto Board and Discussion

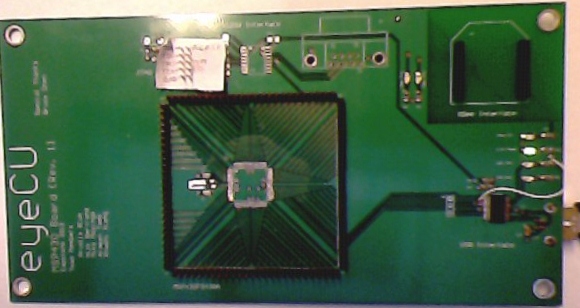




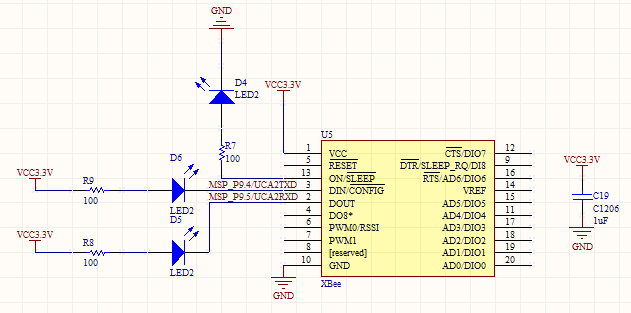
BeagleBone and custom camera testing



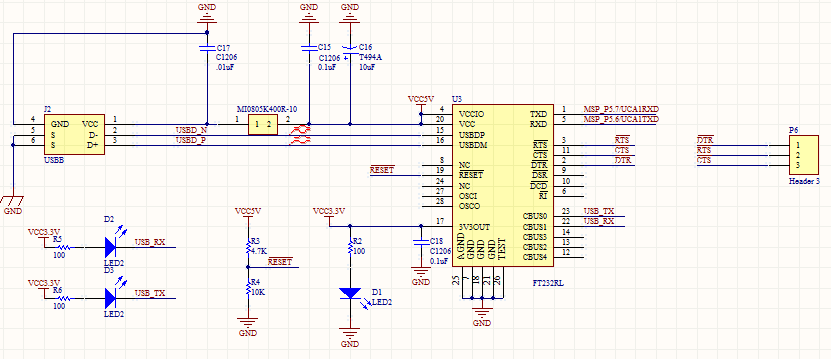
Glasses Prototype (get Khashi’s stuff too)



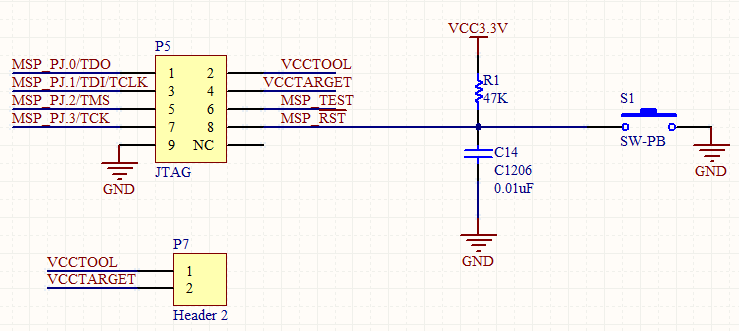
MSP430 Board discussion

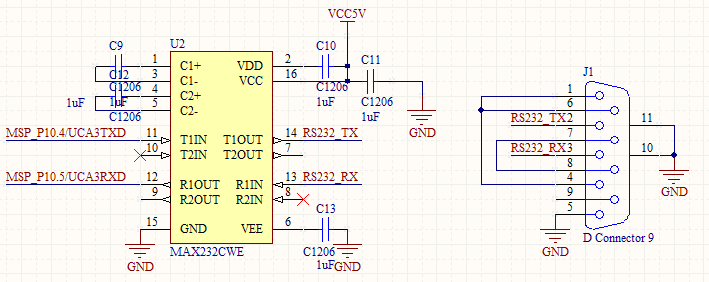


XBee Circuit



USB Circuit

\JTAG Connector



RS232 Interface

During the initial stage of development the Texas Instruments LM3S6965 Ethernet Evaluation Kit was used in order to begin development of the communication with the TCM8230MD camera.

Figure 1 shows the timing constraints from the TCM8230MD camera datasheet. The two parameters boxed in red will be examined.

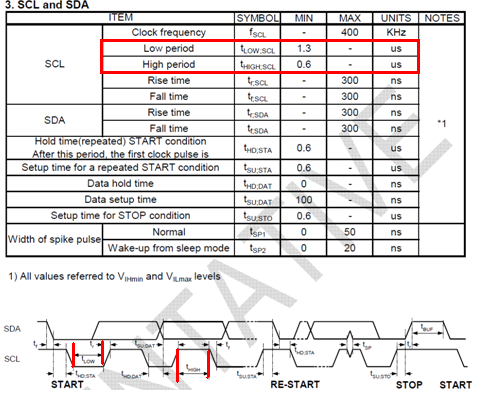


Figure 1: Camera I2C Timing Parameters

Figure 2 displays the logic analyzer capture of an I2C command generated by the Stellaris. The Stellaris, configured as the I2C Master successfully executed a write command, writing the data 0x30 to register 0xA0 to the slave address of 0x50.

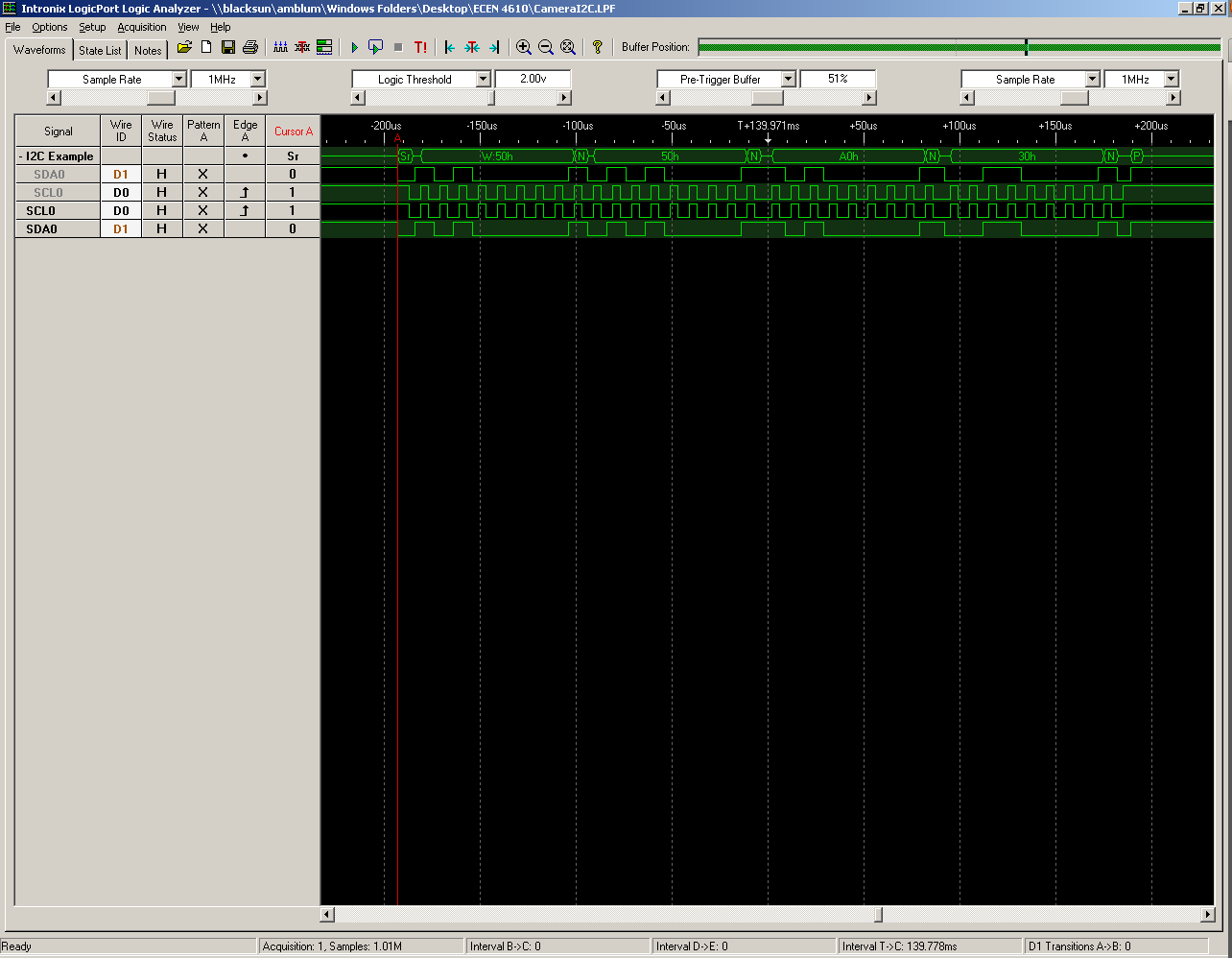


Figure 2: Logic Analyzer Capture of Stellaris I2C

The Figure 3 displays the I2C protocol used by the camera.

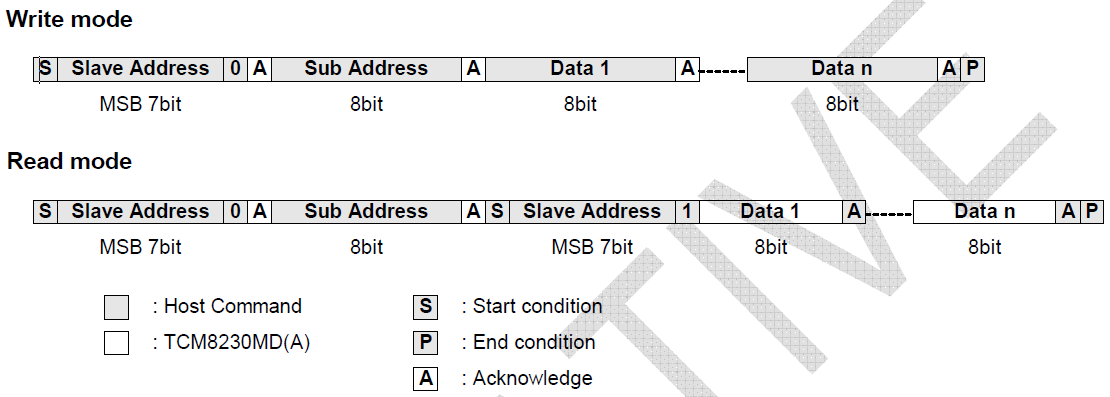


Figure 3: TCM8230 I2C Protocol

As shown by Figure 2, I2C communication was initialized with a START command followed by the write command. The Stellaris should wait for an ACK here but since it received no response, the I2C interpreter translated it as a NACK. Next the slave address, 0x50 was sent out, again with no ACK response. Next the subaddress or register address, 0xA0 was sent with no ACK received. Finally the data, 0x30, with a NACK. To end the transmission, the Stellaris generated a STOP command.

Figure 4 displays the timing analysis conducted on SCL.

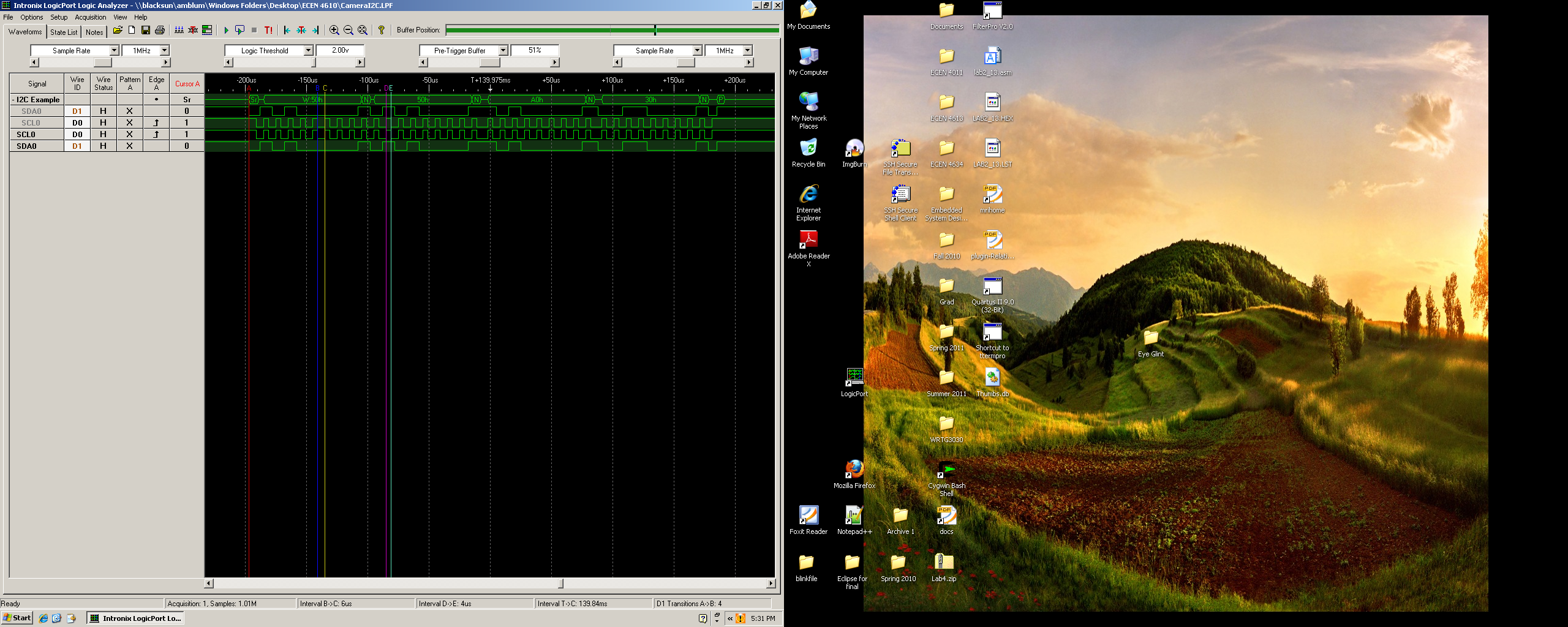


Figure 2: Logic Analyzer Capture of Stellaris I2C

The following table compares the required timing for SCL I2C communication with the measured SCL timing for the Stellaris.

|  |  |  |
| --- | --- | --- |
|  | Measured | TCM8230MD Required Minimum |
| SCL Low Period | 6us (Interval B -> C) | 1.3us |
| SCL High Period | 4us (Interval D -> E) | 0.6us |

Table 1: SCL Timing for Measured and Required

From the table it is clear that the SCL generated by the Stellaris satisfies the camera’s timing constraints.

**Timing Analysis of JTAG Communication (MSP430F5438A and MSP-FET430UIF)**

According to the datasheet for the MSP430F5438A, the maximum TCK frequency is 10MHz as shown by the following table.

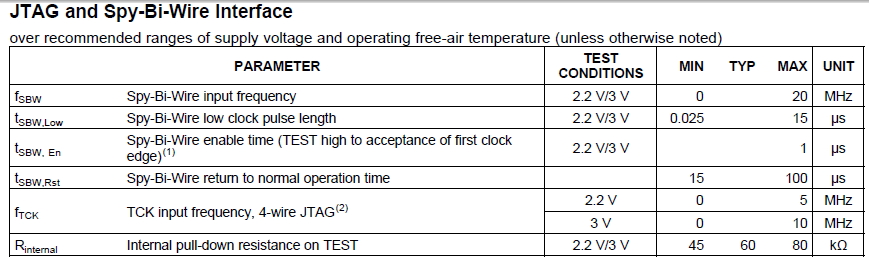


Figure 1: MSP430F5438A Datasheet JTAG Timing Requirements

When using the logic analyzer to sample the JTAG signals, we must sample at least at twice the highest frequency of the signals. Therefore I set the sample rate at 50MHz. Figure 2 displays a screen capture of the JTAG signals: TCK, TMS, TDI, TDO. Note that the highest TCK frequency is 3.847MHz which is less than the maximum TCK frequency specified by the datasheet.

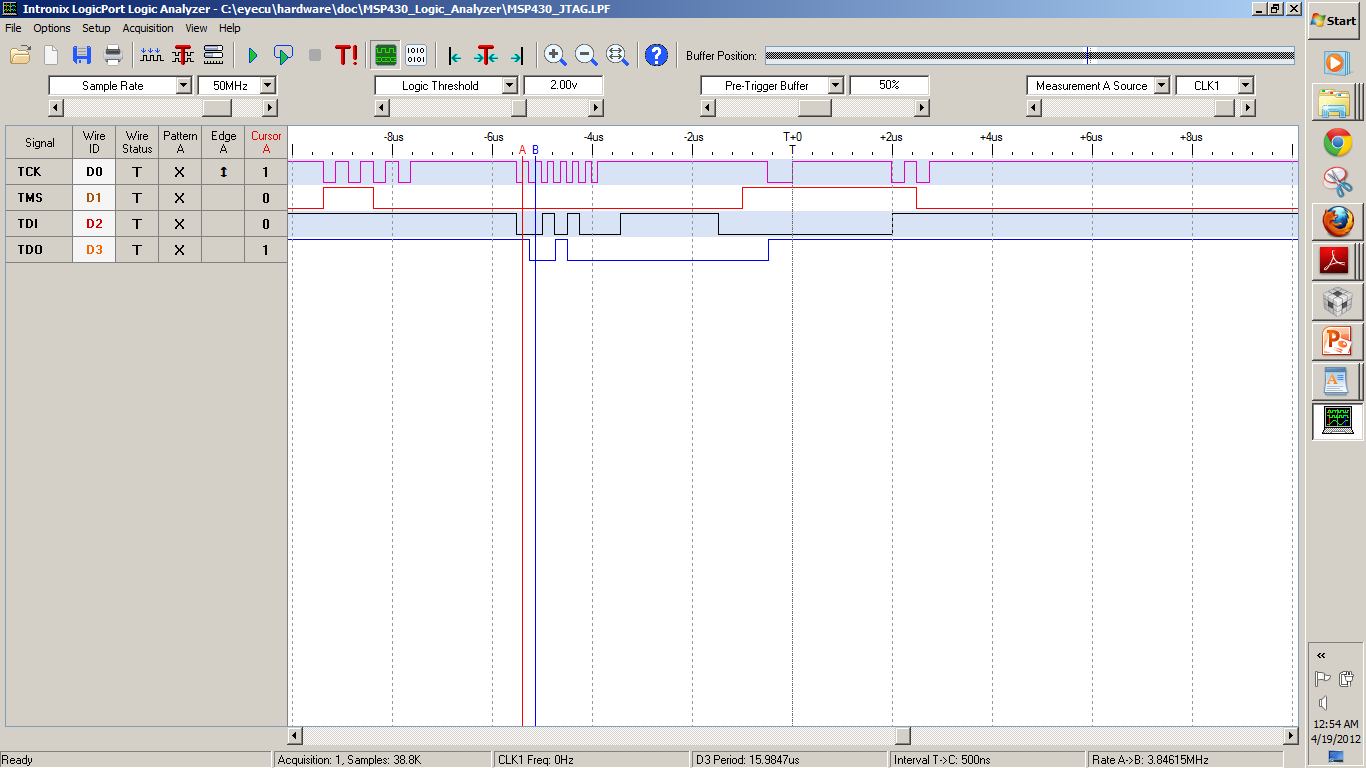


Figure 2: Logic Analyzer Capture of JTAG Communication

**MSP430F5438A Programming and Verification**

Adapting some sample code provided by Texas Instruments, I was able to verify that the MSP430F5438A was successfully programmed.

According to the datasheet for the MSP430F5438A, there is an internal low frequency oscillator onboard. The frequency of this oscillator is 32kHz as shown in the following excerpt from the datasheet.

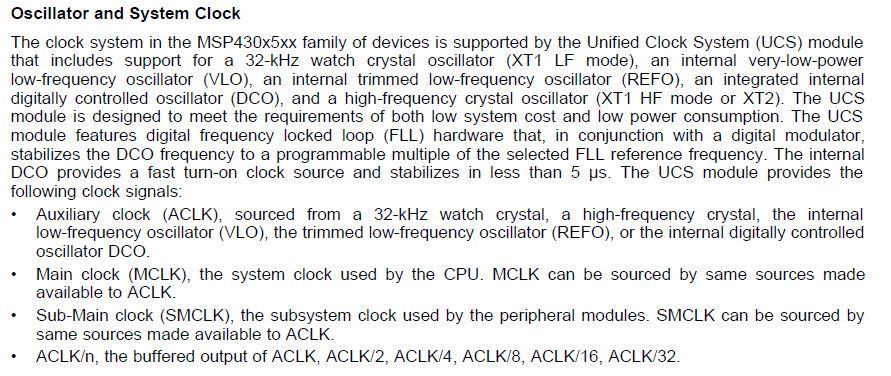


Figure 3: MSP430F5438A Internal Oscillator

To verify the proper programming of the MSP, I set the internal clock to output to pin P11.0 and probed the waveform generated at this pin. The following is the code I used to program the MSP

**#include** "msp430x54xA.h"

**void** **main**(**void**)

{

WDTCTL = WDTPW + WDTHOLD; // Stop WDT

P11DIR = BIT0; // P11.0 to output direction

P11SEL = BIT0; // P11.0 to output ACLK

UCSCTL4 |= SELS\_5; // Select ACLK source

**while**(1); // Loop in place

}

Figure 4 displays the waveform captured by the oscilloscope. As we expect, the signal has a frequency of roughly 32.768kHz and a voltage of ~3.3V.

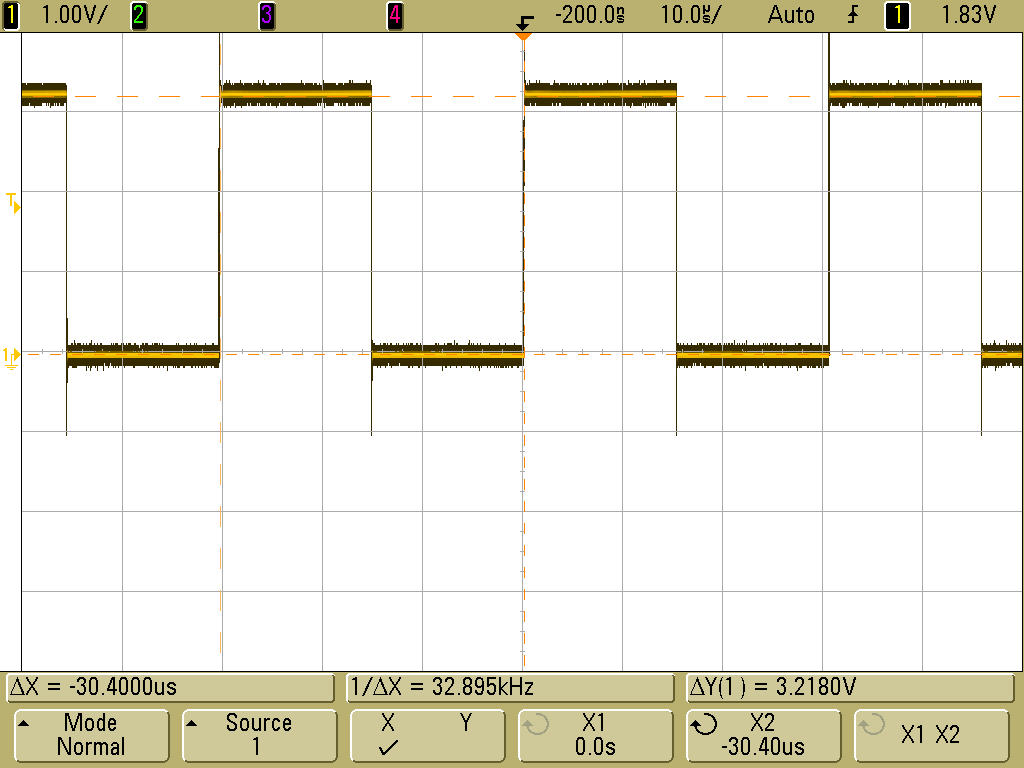


Figure 4: Oscilloscope Capture of ACLK

Furthermore, the debugger in Code Composer Studio version 5 is operational as shown by Figure 5.

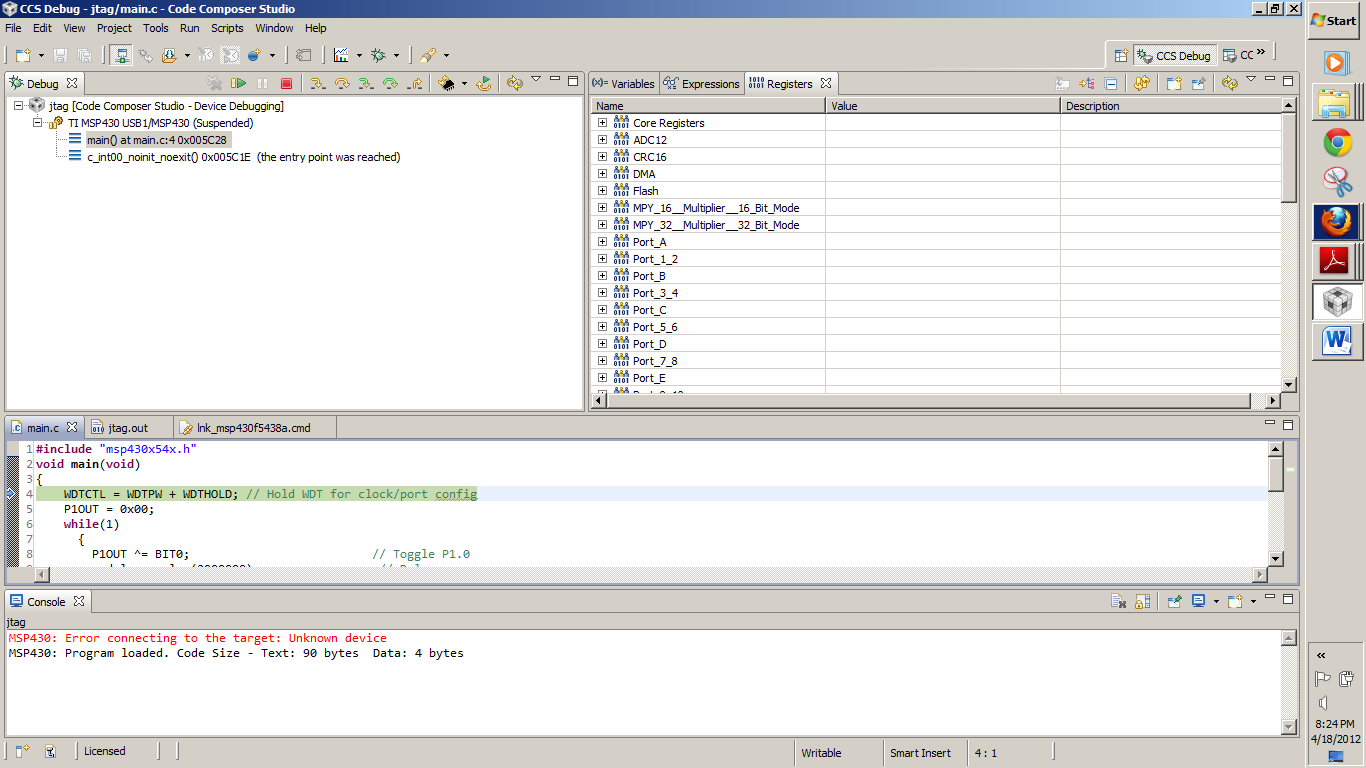
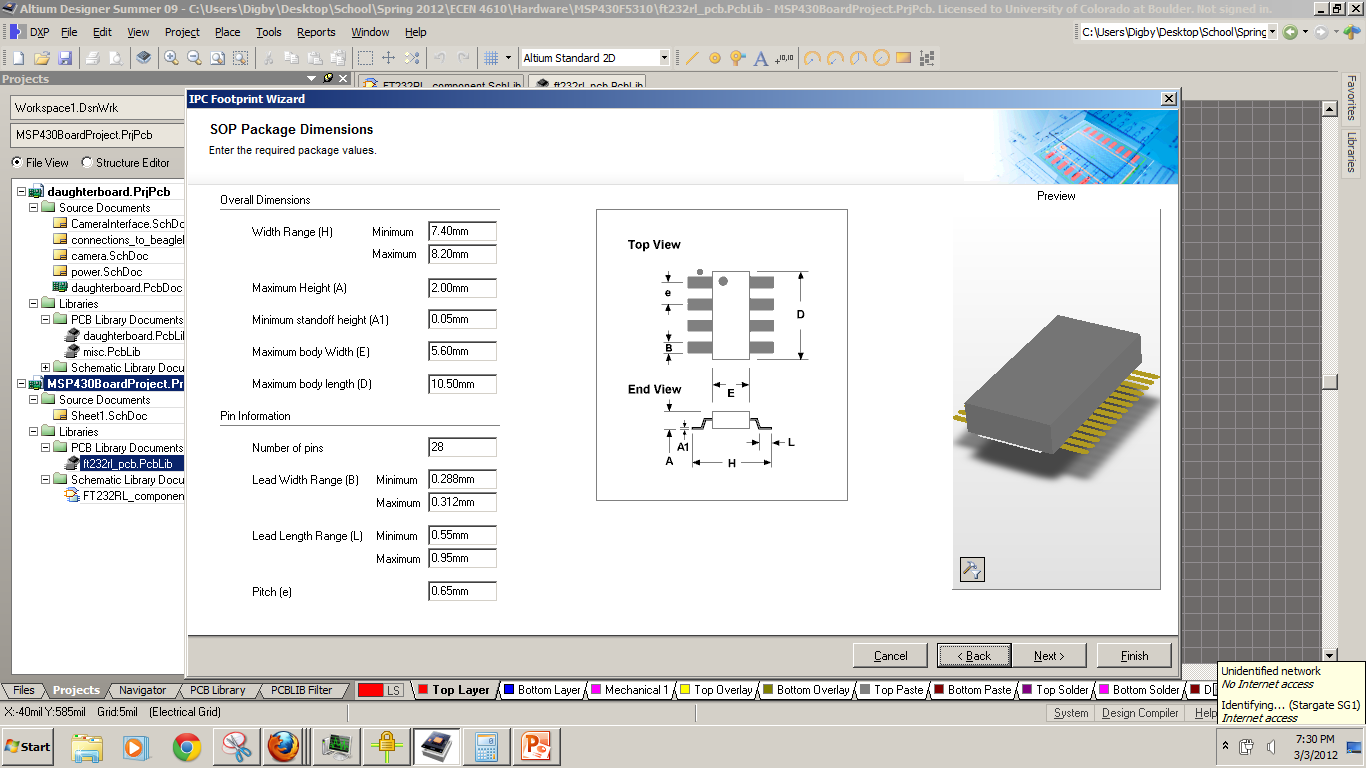


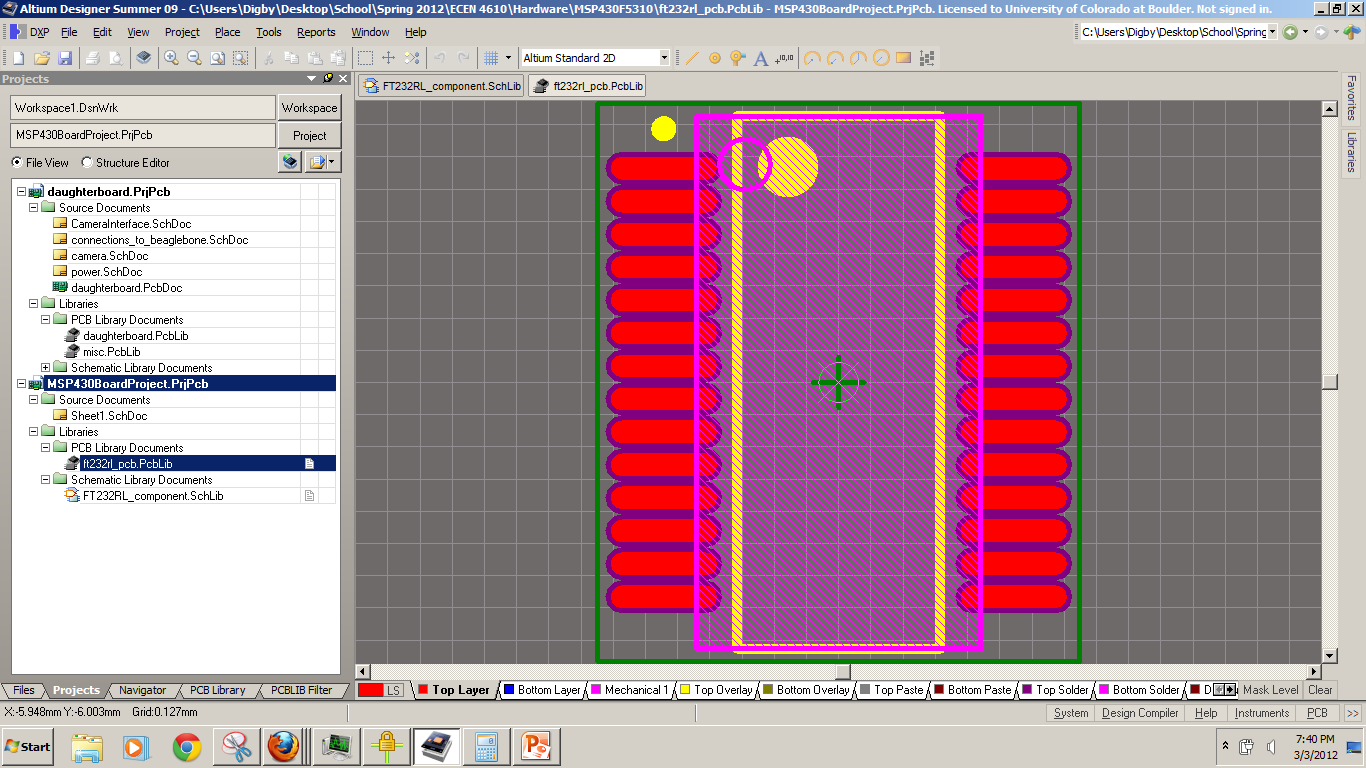
Figure 5: CCSv5 Debugger Operational

**Making of Custom Component in Altium: FT232RL**

IPC Wizard



Footprint after Wizard



Adding the footprint to the new component

