# EE-492-S02-SpTp-Advance Digital Hardware-SP2017

Project #2 - Report

Design and Verification of a   
SPI to JTAG Interface Adapter

Written by: [Jordan D. Ulmer](https://sites.google.com/site/whoisjdu/home)

Written by: [Nathan](https://sites.google.com/site/whoisjdu/home) Genetzky

Instructor: Dr. Robert Fourney

|  |  |
| --- | --- |
| Design Initiated: | **2017-03-25** |
| Date Demonstrated: | **2017-05-03** |
| Report Submission: | **2017-05-05** |
| Git Repository: | <https://bitbucket.org/adhd_digital_design/ee492> |

## 

## Goal and Motivation:

Almost all hardware devices have JTAG connections which can provide data from registers or provide low level control of pins via a boundary scan chain. Unfortunately, most microcontrollers do not have libraries or drivers for communicating using the JTAG protocol. The goal of this project is to provide a way to allow users to use existing libraries to access data that is typically accessed via JTAG protocol.

## Table of Contents:

[**EE-492-S02-SpTp-Advance Digital Hardware-SP2017**](#_wfz666irym1n) **1**

[Goal and Motivation:](#_egulo0f37j0h) 2

[Table of Contents:](#_f8ybkanwn921) 2

[Background:](#_p94gkpfnr745) 3

[System Components:](#_iymmskx97lcx) 5

[Hardware Design Components:](#_ko58vdlnwgpz) 6

[top\_spi2jtag & top\_hipe\_spi](#_vg455xpyqtr0) 6

[nios2\_spi\_pio](#_yvdcjdvx468e) 6

[hipe\_spi](#_yaj0yezajqwy) 8

[hipe\_jtag](#_ac209d2kusfz) 9

[spi2jtag](#_y37f2qknhti8) 10

[HDL Files](#_rqrh0j24mmw3) 11

[HDL Components for Top:](#_i32r1ejmcrdu) 11

[Software Design Components:](#_k9figaynmmh) 12

[Spi2JtagApp](#_k668yu9wtb5q) 12

[Software Files](#_xtlvas767szp) 12

[Verification:](#_35cb05mkgk4) 14

[Verified jtag\_tap\_fsm via dev. board](#_ulkikigig5tm) 14

[Verified TAP state transitions](#_9t0bk64i8zo3) 14

[Verified IR and DR shifts](#_7kwripk8pjvy) 15

[Demo Recap:](#_wppq6najmi19) 16

[Future Work:](#_e7x4s876rpwz) 16

[Appendix:](#_rw5yw2codraz) 17

## Background:

Four connections are required for JTAG and multiple devices can be supported by daisy chaining the Test Data In (TDI) & Test Data Out (TDO) connections between the devices, Fig. 1. Serial Peripheral Interface (SPI) is another serial protocol; one that is more likely to have a library provided for a particular micro controller. The similarities between SPI are readily apparent by looking at SPI interfacing with multiple devices in a daisy chain configuration, Fig 2. Note that both protocols have bidirectional flow by providing two data pins; the TDI of JTAG slave = Master Out Slave In (MOSI) of SPI master and TDI of JTAG slave = Master In Slave Out (MISO) of SPI master. The fourth signal is where these two protocols differ greatly. The JTAG Test Mode Select (TMS) signal requires more logic than Slave Select (SS), as seen in the timing diagram for a typical JTAG communication, Fig. 3. The TMS and TCK are used to control the TAP state machine, Fig. 4.

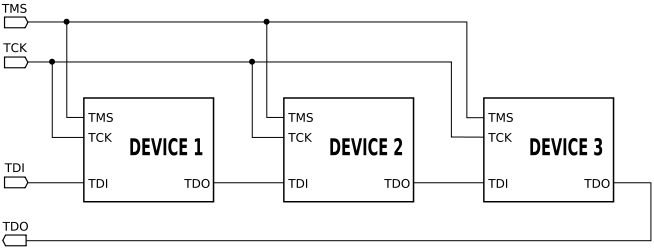


Figure 1: Multiple devices connected to a JTAG connection.

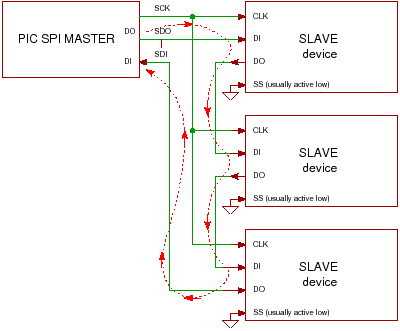


Figure 2: Multiple SPI slaves in daisy chain configuration.

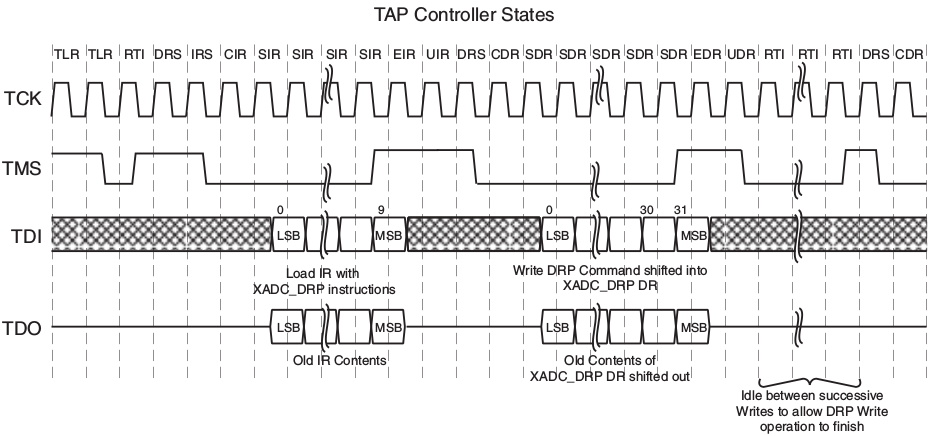


Figure 3: Timing diagram for a typical JTAG communication.

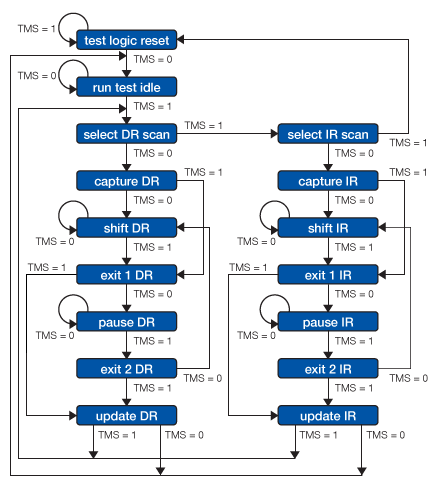


Figure: 4: JTAG finite state machine (IEEE 1149.1 JTAG)

## System Components:

Both software and hardware components were utilized to demonstrate the capability of the ***spi2jtag*** HDL module. A general overview of one particular application of the SPI to JTAG (spi2jtag) hdl module is shown below(See Fig. 6). Note that in the particular application demonstrated the “JTAG TDRs” is also a hdl module. A more general use case could use JTAG signals, provided by ***spi2jtag***, and be connected to a JTAG device that is external to the FPGA (See Fig. 5).



Figure 5: Block diagram for SPI Master driving a JTAG Device (Tap Controller and TDRs) through spi2jtag.



Figure 6: System Block Diagram for testing ***spi2jtag*** with a Nios2 processor as the SPI master.

## Hardware Design Components:

The important hardware modules are described in detail, and then a summary of each hdl file is provided. For the convenience of the reader components designed in EE-492 by Jordan and Nathan are listed in black bold italics (ie. ***top\_spi2jtag***). Individual Qsys components directly used from altera are listed in italics (ie. *altera\_avalon\_spi*). Components used from the Hardware Integrated Prototyping Environment (HIPE) Senior Design project, of which both Jordan and Nathan were team members are listed in bold red (ie. ***Senior Design****)* (See key in Fig. 6)

### top\_spi2jtag & top\_hipe\_spi

Several hdl modules were used to demonstrate SPI to JTAG communication; ***top\_hipe\_spi*** is equivalent in core functionality to ***top\_spi2jtag***, Fig.7. The *nios2\_spi\_pio* Qsys module uses Altera embedded IP. The SPI core in this Qsys module communicates with two SPI slaves in ***hipe\_spi*** via ***spi\_inf***.

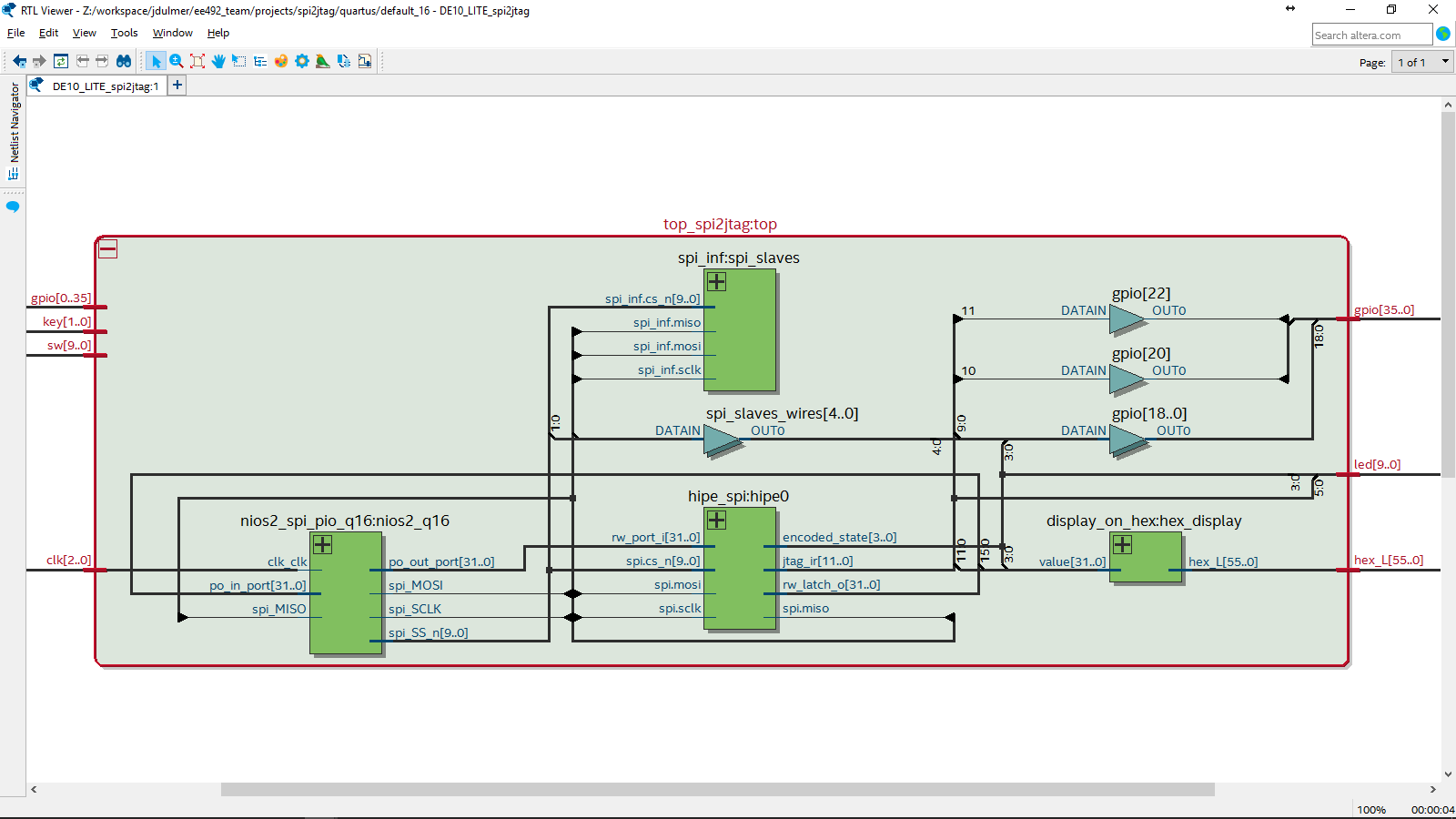


Figure 7: RTL for ***top\_spi2jtag.***

### nios2\_spi\_pio

Altera Qsys components, Fig. 8, were combined into a single Qsys module *nios2\_spi\_pio*. The SPI Qsys component was configured as a SPI master; the timing configuration for the SPI Qsys component, Fig. 9, was determined by considering [SPI-Clock Polarity and MODE](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus#Clock_polarity_and_phase) with the the IEEE 1149.1 JTAG specification.

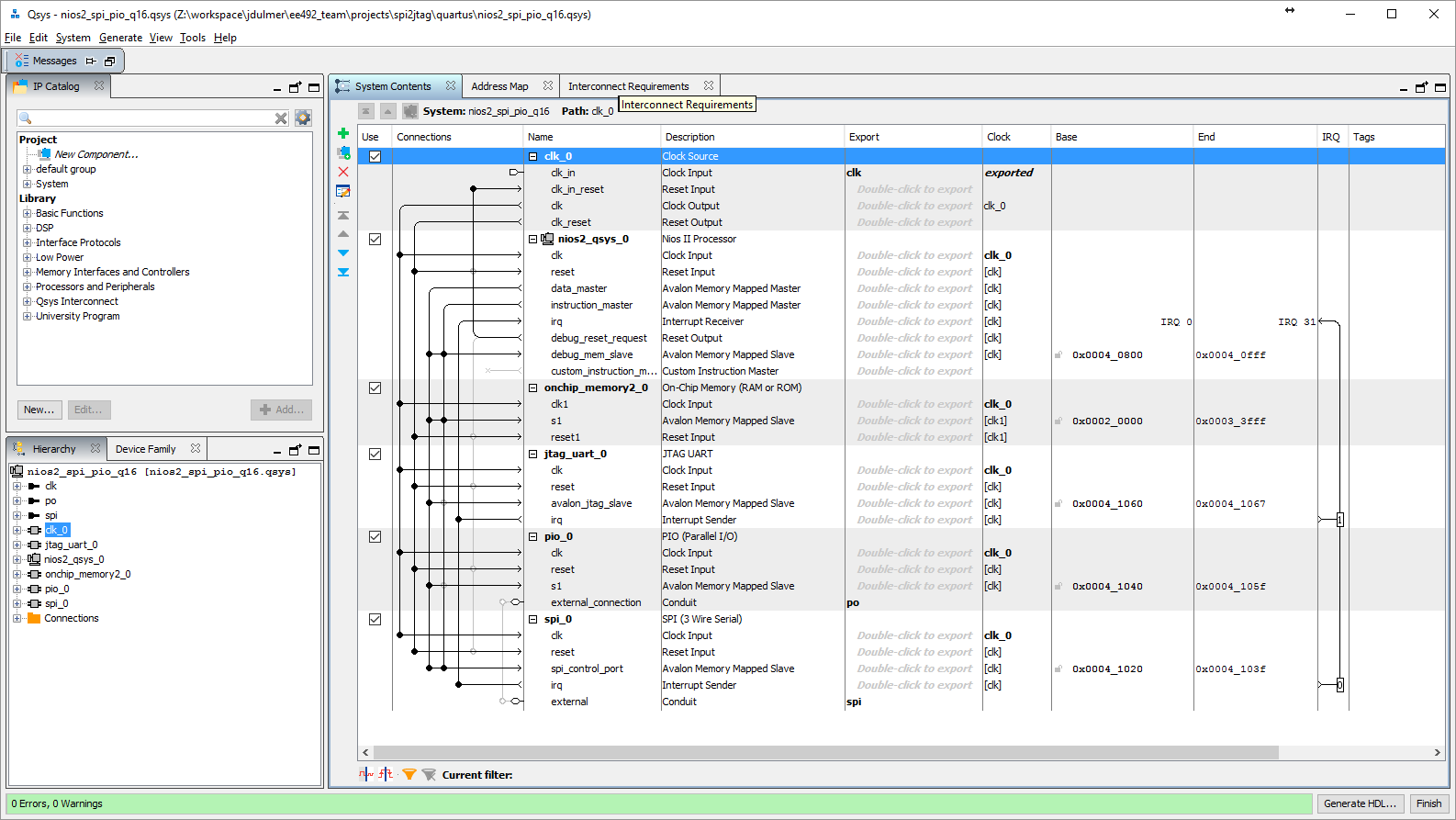


Figure 8: System contents for *nios2\_spi\_pio\_q16*.



Figure 9: Configuration for *altera\_avalon\_spi* (spi\_0) component of *nios2\_spi\_pio\_q16*.

### hipe\_spi

In order to use this module as a JTAG device the TAP controller was created (***jtag\_tap\_ctrl***, ***jtag\_tap\_fsm***). The JTAG device in hipe\_spi, Fig. 10, was created by combining the TAP controller, Fig. 11, with ***hipe\_jtag***, Fig. 14. Instead of requiring JTAG connections on the top of the module, the ***spi2jtag*** module, Fig. 15, was used to connect the JTAG device via SPI.



Figure 10: Annotated RTL for ***hipe\_spi.***

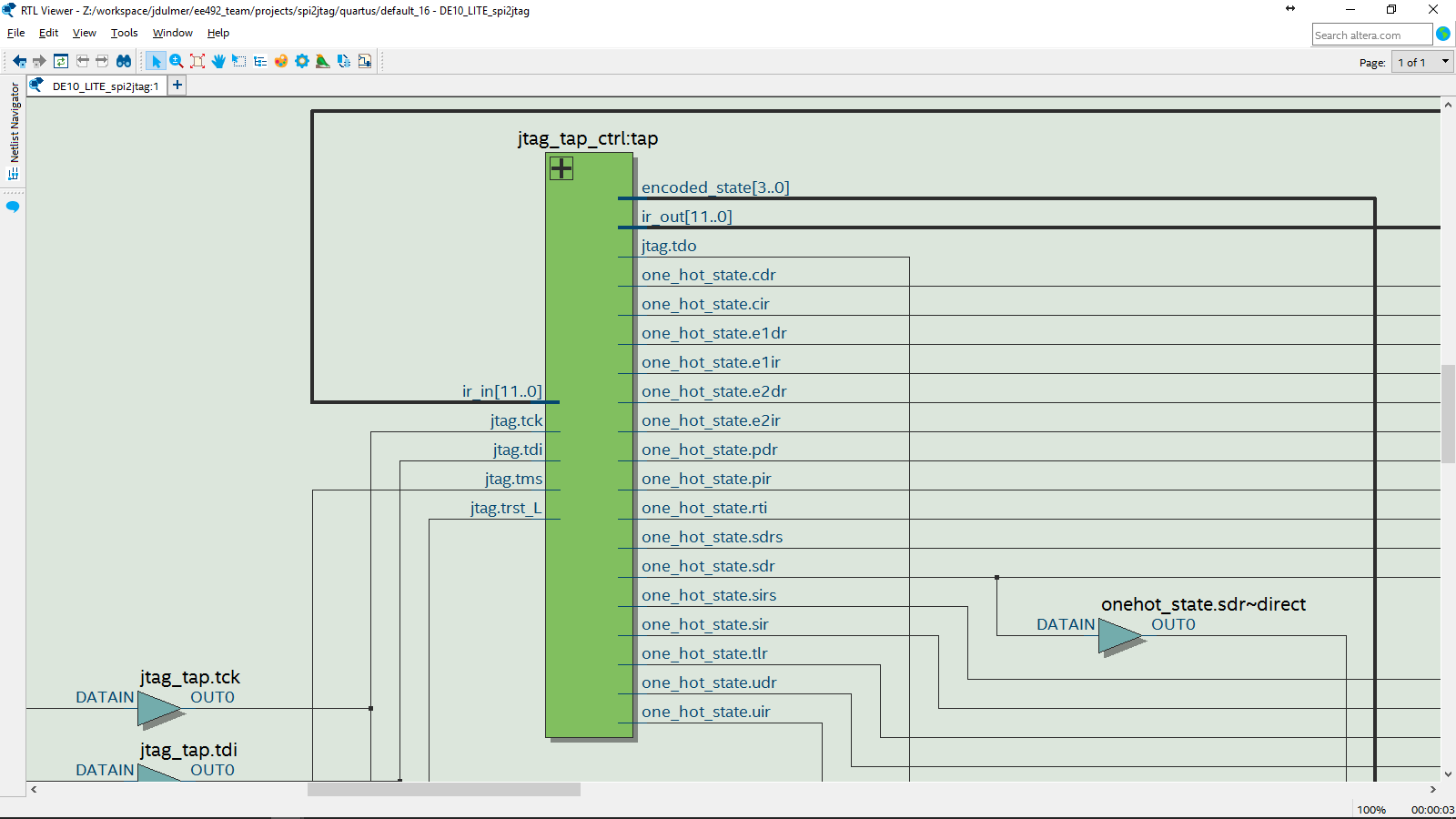


Figure 11: RTL for ***jtag\_tap\_ctrl*** instantiated in ***hipe\_spi.***

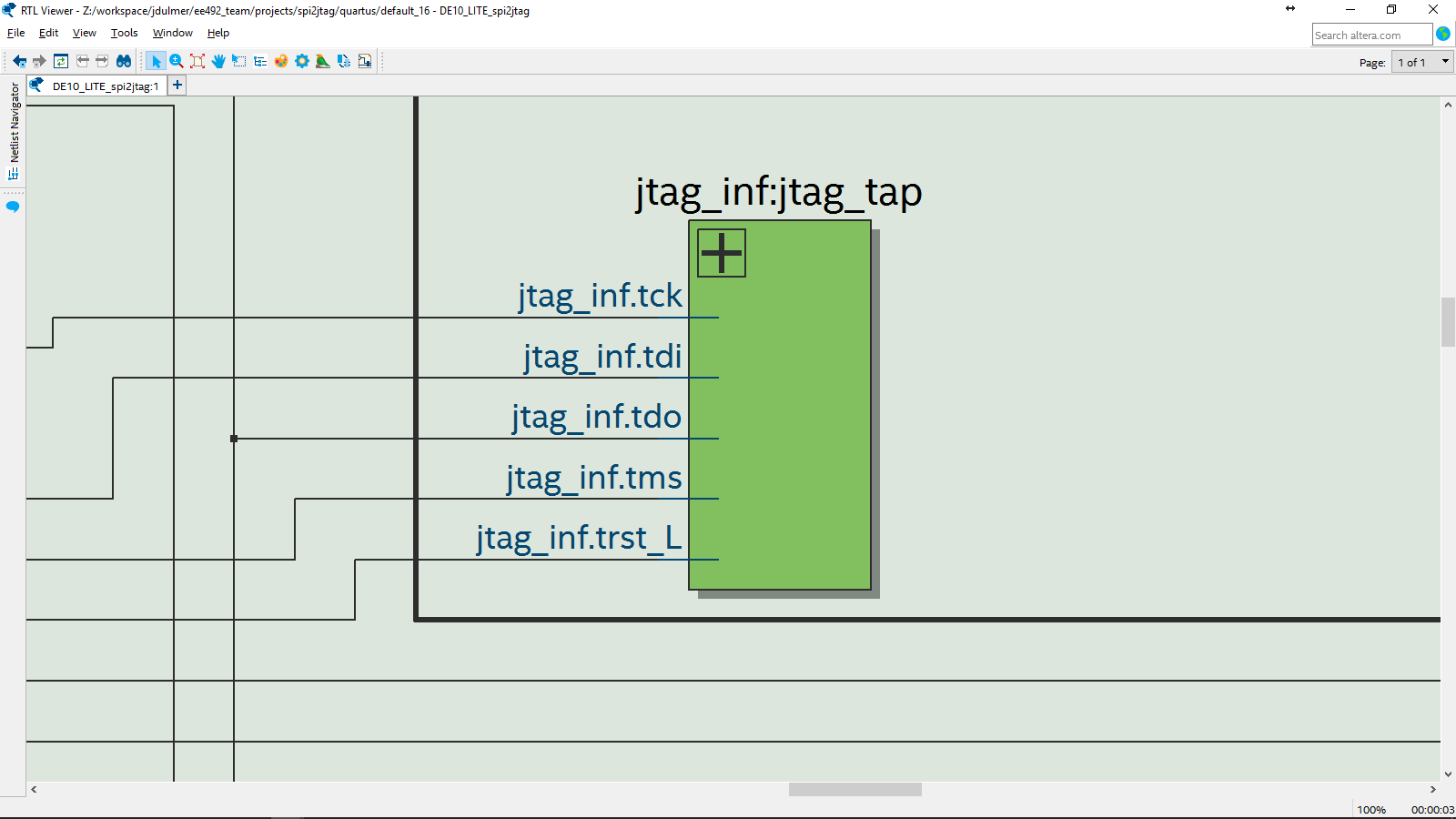


Figure 12: JTAG Interface instantiated for tap in ***hipe\_spi.***

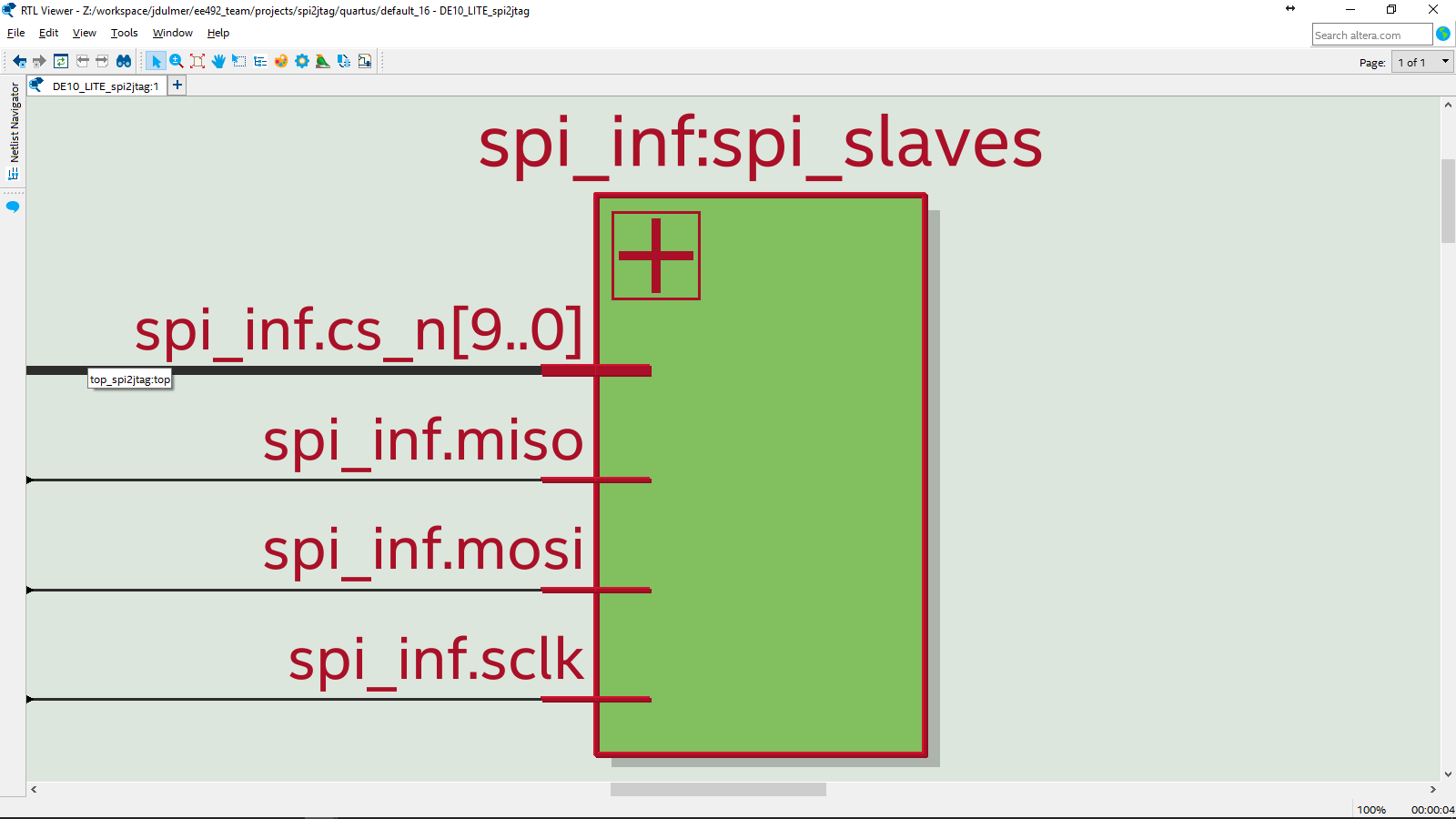


Figure 13: SPI Interface instantiated in ***hipe\_spi.***

### hipe\_jtag

The jtag device that was chosen to demonstrate ***spi2jtag*** was from our Senior Design project, HIPE (Hardware Integrated Prototyping Environment). Understanding exactly how this module works is out of the scope of this document; however, a brief description will be provided. Basically the module provides two parallel vectors of ports, one in and one out, to the hardware designer; by default all signals on the input are driven at the output but by changing the state of the module the output can be changed. The state of the module can be changed by control signals from a JTAG TAP controller, see Fig. 11.

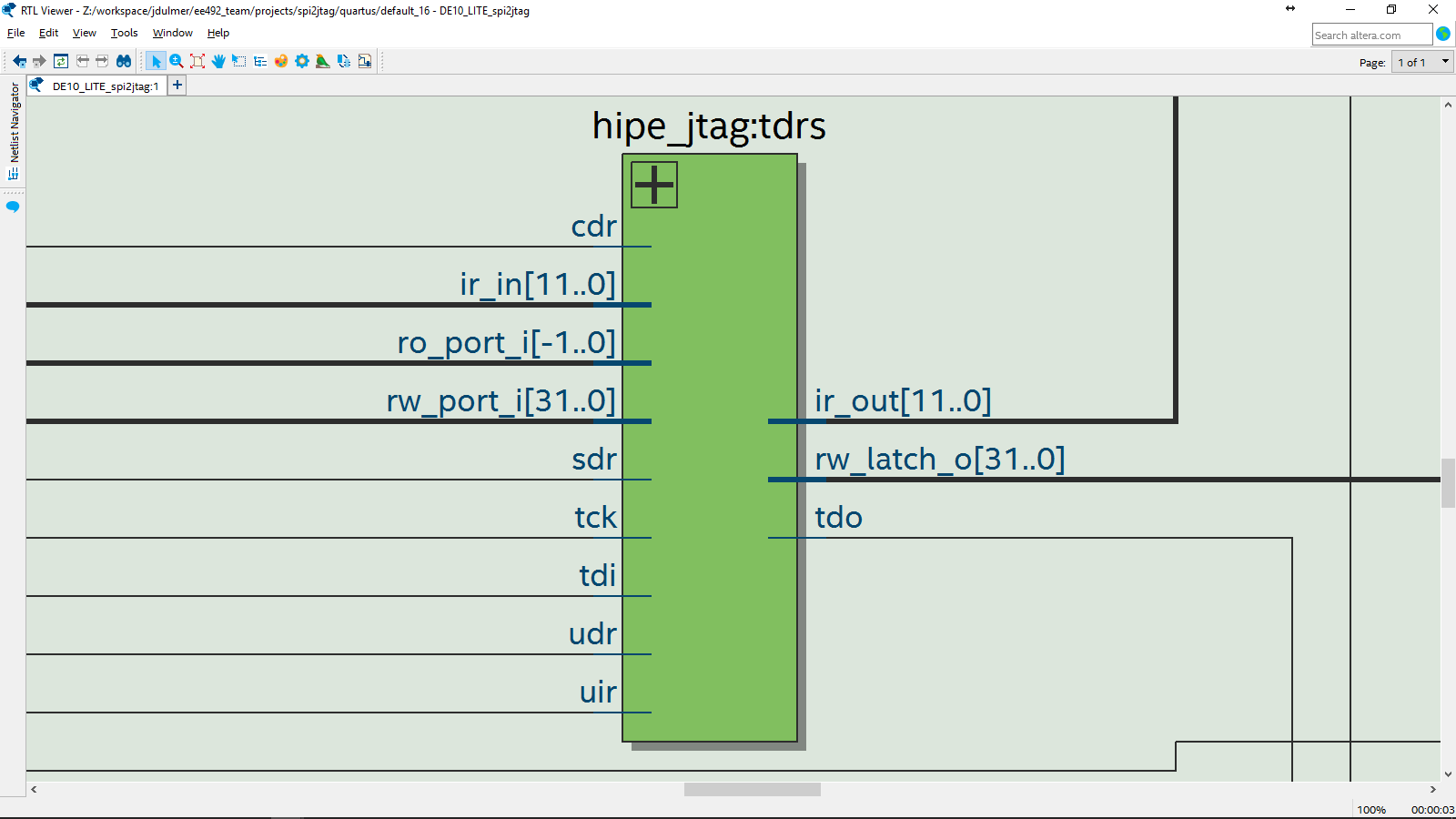


Figure 14: RTL for ***hipe\_jtag*** instantiated in ***hipe\_spi.***

### spi2jtag

Provides two SPI slaves. One of these slaves should be selected to communicate with the TAP controller, and the other selected to communicate with the IR or DR. The TAP controller can be navigated by sending the TMS signal for each TCK pulse as the MOSI for that SCLK pulse. The second slave has MOSI connected to TDI and MISO to TDO (See Fig. 15).

The important portion of ***spi2jtag.sv*** is very terse in hdl and easy to understand by a knowledgeable hardware designer, and therefore is included below.

|  |
| --- |
| ***ee492\projects\spi2jtag\hdl\spi2jtag.sv***  51: assign cs\_n\_tap = spi.cs\_n[0]; // Shift out jtag\_encoded\_state  52: assign cs\_n\_jtag = spi.cs\_n[1]; // Shift out Instruction Register or Data Register  53: assign cs\_n\_loopback = spi.cs\_n[2]; // Loopback  54:  55: assign jtag.trst\_L = 1'b1; // Unused Active low reset  56: assign jtag.tck = spi.sclk; // Test Data Clock  57: assign jtag.tdi = spi.mosi; // Test Data (Shift) in  58:  59: // Allow tdi to control Tap FSM when tap is enabled.  60: // When disabled the Tap will move to and stay at nearest stable state. (on tck)  61: // Will steady out to one of these states: RTI, SDR, PDR, SIR, PIR.  62: assign jtag.tms = ( (1'b0==cs\_n\_tap ) ? spi.mosi : // Test Mode Select  63: 1'b0 ); // Default  64:  65: assign spi.miso = ( (1'b0==cs\_n\_jtag ) ? jtag.tdo : // Test Data (Shift) out  66: (1'b0==cs\_n\_tap ) ? tapp\_state\_so\_delayed :  67: (1'b0==cs\_n\_loopback) ? spi.mosi :  68: 1'b1 ); // Default |

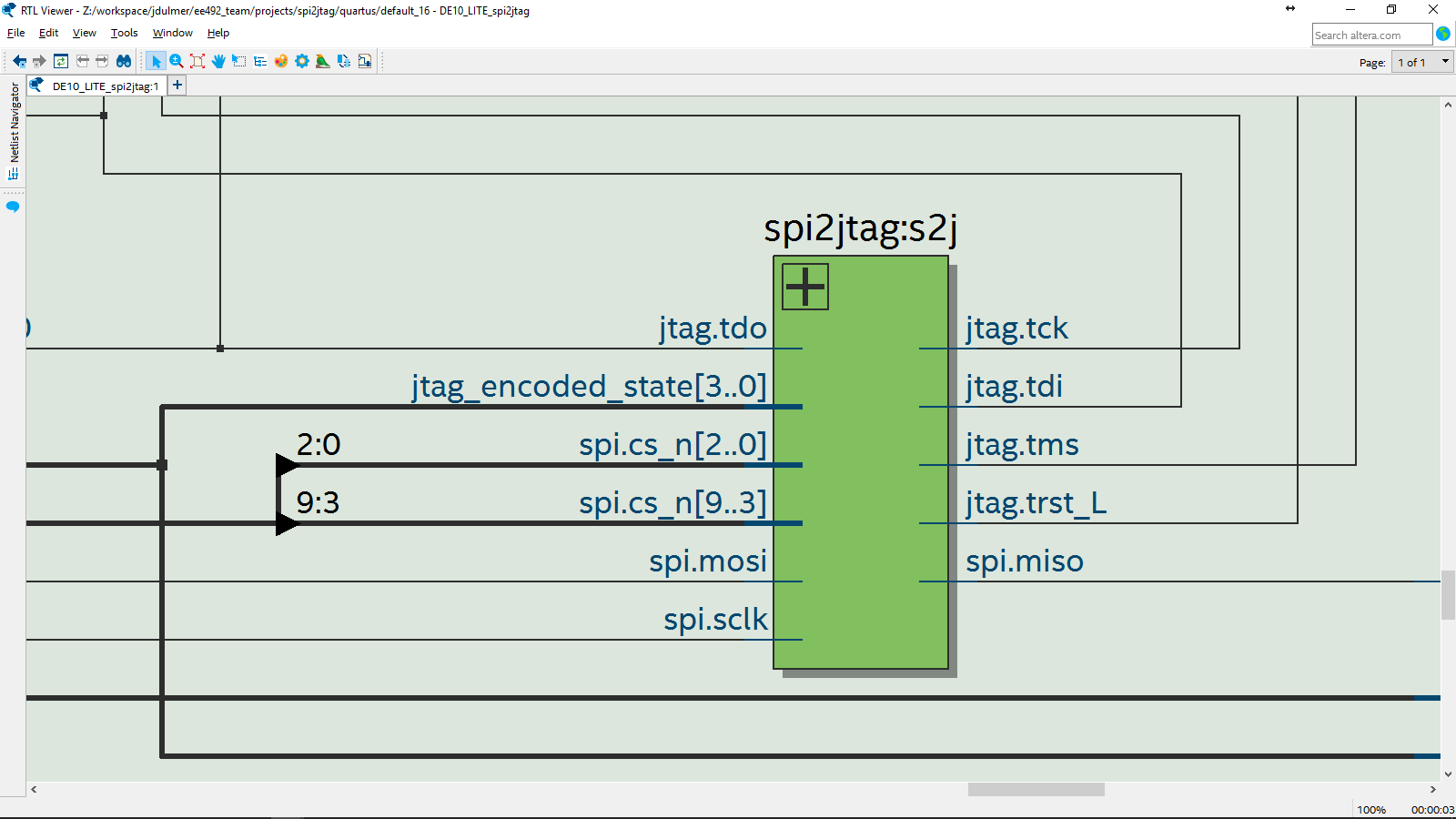


Figure 15: RTL for ***spi2jtag*** instantiated in ***hipe\_spi.***

### HDL Files

Each hardware module is listed with a brief summary of its purpose below.

1. ***hipe\_spi.sv***
   1. Instantiates hipe\_rw and ***jtag\_tap\_ctrl*** as a complete JTAG Device.
   2. Provides ***spi\_inf*** which connects JTAG signals via ***spi2jtag***.
2. ***spi2jtag.sv***
   1. Provides two SPI slaves, the first controls the TAP controller via TMS and the second accesses the JTAG IR/DR registers via TDI/TDO, see Fig. 15.
3. ***spi\_inf.sv***
   1. A SPI interface with a configurable number of active low chip selects, see Fig. 13.
4. ***jtag\_inf.sv***
   1. A JTAG interface utilized by JTAG devices, see Fig. 12.
5. ***shift\_reg\_async\_ld.sv***
   1. A shift register with async load, which is utilized for shifting out current jtag\_tap\_state.
6. ***jtag\_tap\_ctrl.sv***
   1. Provides control signals for TDR, controlled via *jtag\_inf*, and provides either internal IR or external DR between TDI/TDO based on TAP state, Fig. 11.
   2. ***jtag\_tap\_fsm.sv***
      1. Implements JTAG finite state machine, Fig. 4.
   3. ***jtag\_tap\_fsm\_state\_t.svh***
      1. 4-bit enum reg for JTAG states (IEEE 1149.1 JTAG), equivalent to ***JtagTap.h****.*
   4. ***jtag\_tap\_fsm\_one\_hot\_state\_inf.sv***
      1. Provides one-hot state signals for JTAG TAP states (IEEE 1149.1 JTAG), which can be used to control JTAG test data registers (TDRs).
7. ***hipe\_jtag.sv***
   1. Provide test data registers (***hipe\_rw.sv***), which can be controlled by a TAP controller and accessed as a shift register, Fig. 14.
8. ***dev\_board\_hiper.sv***
   1. instantiation of hipe\_spi which connect various development board signals (GPIO, push buttons, switches, leds, and 7-Segment displays) through HIPE to allow any signals from an FPGA board to be controlled via software and provide a very visual demo. Not verified.

## HDL Components for Top:

Top files connect hdl modules with the development board (DE10\_LITE). Support was also provided for DE1-SOC, and the DE2, but all verification was performed on the DE10\_LITE:

1. ***top\_hipe\_spi.sv***
   1. Verified and used for final demo.
   2. Instantiates *nios2\_spi\_pio* as SPI Master which interfaces with hipe\_spi.
   3. PIO of *nios2\_spi\_pio* is connected in loopback through ***hipe\_spi*** to verify functionality.
2. ***top\_spi2jtag.sv***
   1. Copy of ***top\_hipe\_spi*** before code cleanup. Outputs IR, tap state and SPI via GPIO. Verified.
3. ***top\_spi\_master.sv***
   1. Instantiates *nios2\_spi\_pio* and connects SPI through GPIO.
   2. Not verified; however, SPI transactions appeared to be correct on the OSCOPE.
4. ***top\_dev\_board\_gpio.sv***
   1. Instantiates ***dev\_board\_hiper.sv*** and connects this SPI slave via GPIO. Not verified.
5. ***top\_dev\_board\_nios2.sv***
   1. Instantiates ***dev\_board\_hiper*** as slave and ***nios2\_spi\_master*** as master*.* Not verified.
6. ***nios2\_spi\_master.sv***
   1. Minimal version of a SPI master via GPIO using *nios2\_spi\_pio*. Not verified.
7. ***top\_spi\_slave.sv***
   1. Instantiates hipe\_spi which intercepts Switches that drive LEDs by default; hipe\_spi is controlled by a spi\_inf ; the spi\_inf was connected to GPIO.
   2. Not verified; were not able to communicate with slave via GPIO from ***top\_spi\_master***.

## Software Design Components:

Notably the software has been written as a library in (C++-03) to support jtag device interactions (ie. IR and DR shifts) using 8-bit SPI transactions through the Altera SPI core (3 wire serial) IP Component. All SPI communication is performed through ***altera\_spi\_transfer()*** in ***altera\_spi.h***; to use the library with a different SPI Master/Driver, this function must be replaced.

### Spi2JtagApp

Provides an example usage of the spi2jtag C++ Library with access to functionality via stdin/stdout. Allows the libraries core functionality to be used in a debug or demo scenario without having to recompile. By calling ***Spi2JtagApp::interactive()*** the user is able specify an action by entering a value 0x0-0xF, Fig. 16. There are four “registers” (a,b,c,d) that can be set and used as arguments for the various function calls. All input is accepted from the user as hex values via functions in ***scanf\_x.h***.

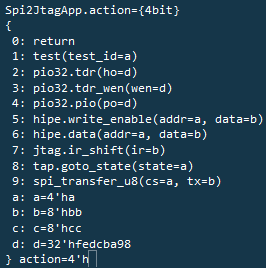
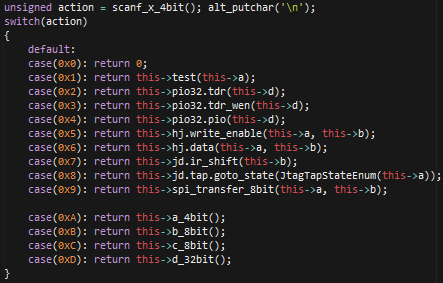


Figure 16: Functionality (left) available to user via terminal (right) connected to stdin/stdout.

### Software Files

Each software module is listed with a brief summary of its purpose below; paths are mentioned relative to the following directory from the repo: **ee492/projects/spi2jtag/quartus/software/AlteraNios2/include**.

1. ***spi2jtag***
   1. Library for interfacing with spi2jtag module and hipe\_spi.
   2. ***spi2jtag.h***
      1. Combines all required includes for spi2jtag software components.
   3. ***spi2jtag/Spi2JtagApp.h***
      1. Provides an interactive UI for the spi2jtag C++ Library via stdin/stdout.
   4. ***spi2jtag/HipeJtag.h***
      1. Struct for accessing registers in ***hipe\_jtag.sv*** via SPI.
   5. ***spi2jtag/HipeRw.h***
      1. Intended to model ***hipe\_rw.sv***; however it was found easier to create ***HipeJtag.***
   6. ***spi2jtag/JtagDevice.h***
      1. Allows a ir\_shift or dr\_shift (see JtagTdr) to be performed with JTAG device via SPI.
   7. ***spi2jtag/JtagTap.h***
      1. Models the JTAG TAP Controller and allows the state to be changed via SPI.
   8. ***spi2jtag/PioTdr.h***
      1. Closed loop verification, expects pio\_out to be connected through HIPE and back into pio\_in which allows closed loop testing from software.
   9. ***spi2jtag/spi\_loopback\_test.h***
      1. Sends SPI transactions with CS\_LOOPBACK therefore MISO = MSOI in hardware.
2. ***altera*/ & *bsp*/**
   1. Files in these directories are specific to interfacing with Altera provided libraries and drivers. This concept increases the portability of the AlteraNios2 library created for this project.
   2. ***altera/altera\_stdint.h***
      1. Macros to convert altera datatypes (ie. alt\_u8 ) to standard types (ie. uint8\_t)
   3. ***altera/altera\_stdio.h***
      1. Macros isolate use of alt\_stdio.h from altera, and make it easy to transition the code to use <stdio.h> instead.
   4. ***altera/scanf\_x.h***
      1. Mimics basic behavior of scanf for command line input using alt\_getchar.
   5. ***bsp/pio/altera\_pio.h***
      1. Macros used to simply use of Quartus Qsys PIO component.
   6. ***bsp/spi/altera\_spi.h***
      1. Macros and functions used to simply use of Quartus Qsys SPI core component.
3. ***jtag/JtagTapState.h***
   1. Enum for JTAG states (IEEE 1149.1 JTAG), equivalent to ***jtag\_tap\_fsm\_state\_t.***
4. ***jtag/JtagTdr.h***
   1. Represents data register in Test Data Register, must be used with other modules to actually shift the data into the TDR.
5. ***lib*/**
   1. Directory contains code provided by other developers and used in this project.
   2. ***lib/BinaryLiteral.h***
      1. Macro magic to convert binary string to its unsigned integer representation.
      2. source: [http://stackoverflow.com](http://stackoverflow.com/)
   3. ***lib/bithacks.h***
      1. Macros for performing basic binary operations on unsigned integers.
      2. source: <http://www.catonmat.net>
6. ***IF\_DEBUG.h***
   1. Macro to wrap debug statements.
7. ***pio2jtag/pio.h***
   1. Unused module where the *altera\_avalon\_pio* Qsys component was modeled.
8. ***pio2jtag/pio2spi.h***
   1. Unused module which was intended to bit-bang SPI over PIO, but was not verified.
9. ***UintUnion.h***
   1. Unused unions for representing an unsigned integer in different ways.

## 

## 

## Verification:

The hardware components for the JTAG TAP controller (***jtag\_tap\_ctrl, jtag\_tap\_fsm)*** have been verified through both hardware and software based drivers and monitors.

### Verified **jtag\_tap\_fsm** via dev. board

Initially, a top file was created to isolate the jtag\_tap\_fsm design, switches were used to drive the Test Mode Select (TMS) and Test Reset (TRST\_L); ***jtag\_tap\_fsm*** was driven by hand through all of the state transitions specified by the IEEE 1149.1 JTAG standard and the results were visually verified by observing the state on the HEX display.

### Verified TAP state transitions

The hardware modules (***spi2jtag, jtag\_tap\_ctrl***) were verified with ***JtagTap::test\_transitions()***; the success of this test also verified **altera\_spi\_transfer()** from **altera\_spi.h** which was used for the SPI communication. The test performed SPI transactions with the ***spi2jtag*** module to transition through states, Fig. 17, in ***jtag\_tap\_fsm****.* These transactions were observed on the OSCOPE, Fig. 18, and the expected IR and tap state observed on the 7-segment hex displays.

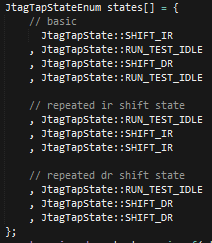
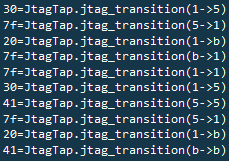


Figure 17: Terminal output (left) from transitioning through a number of states (right).

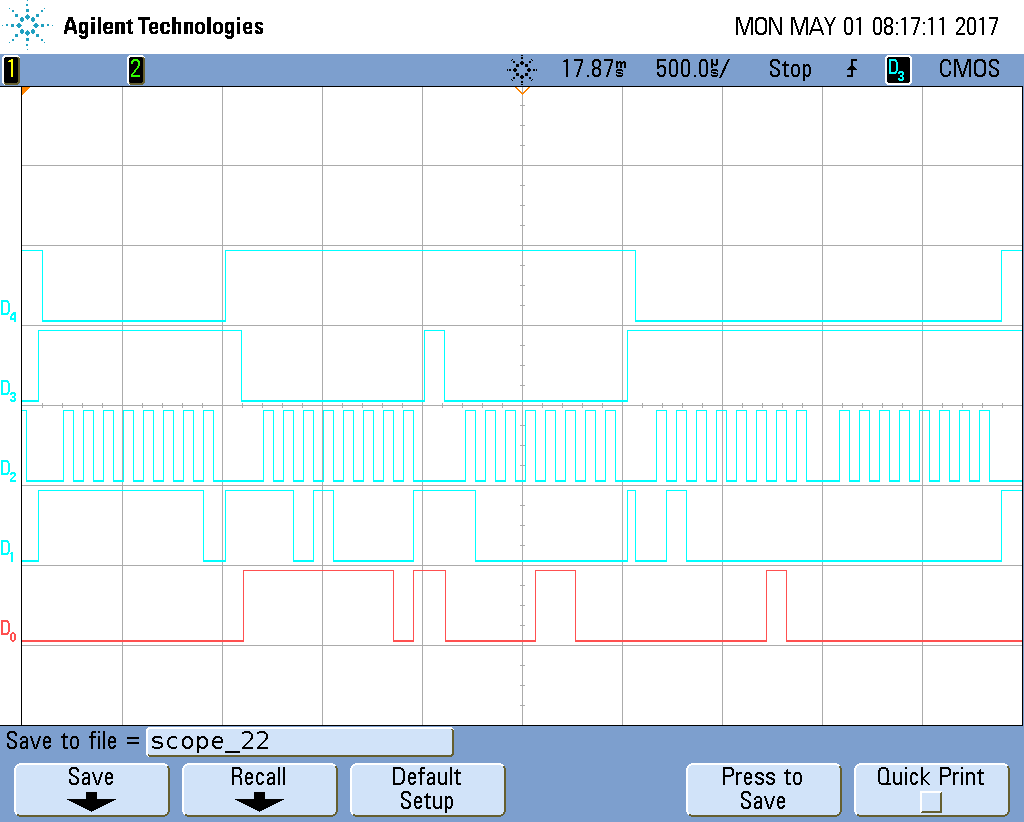


Figure 18: Oscope[[1]](#footnote-0) capture of SPI transactions with JTAG TAP for transitions: {RTI->SIR, SIR->RTI}.

### Verified IR and DR shifts

The hardware module (hipe\_spi) was verified with PioTdr.h; which expected PIO to be connected in loopback through ***hipe\_spi***. After writing a value to the PIO’s output via ***PioTdr::pio\_out()*** the value on the PIO’s input can be read via ***PioTdr::pio\_in()*** to determine the effect of ***hipe\_spi***. Specific IR and DR shifts were performed in ***PioTdr::tdr()*** and ***PioTdr::tdr\_wen()*** to change the behavior of the ***hipe\_spi*** instance and then the expected behavior was checked with ***PioTdr::check\_pi()***. Captures of specific SPI transactions show how the IR and DR value changed via SPI rather than JTAG, Fig. 19 and Fig. 20. The Instruction Register (IR) was also verified visually on the 7-segment hex displays.

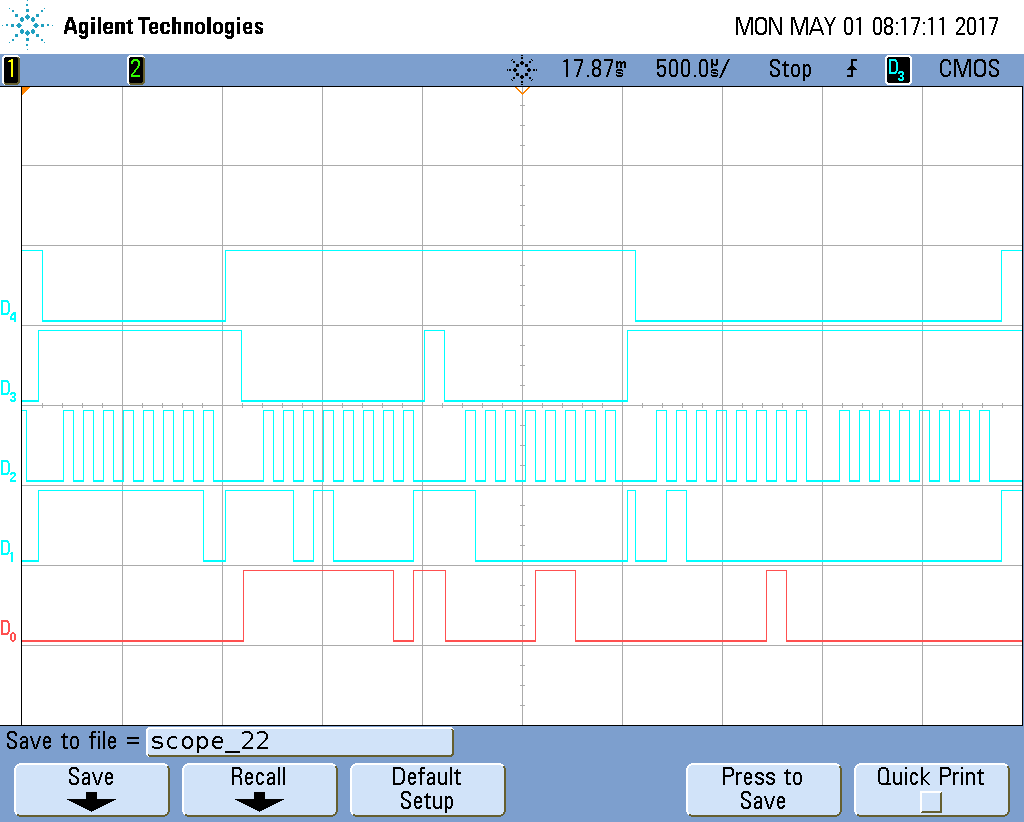


Figure 19: Oscope[[2]](#footnote-1) capture of the following transactions: {tap=SIR, IR[4:0]=4’h2, IR[12:4]=0}.

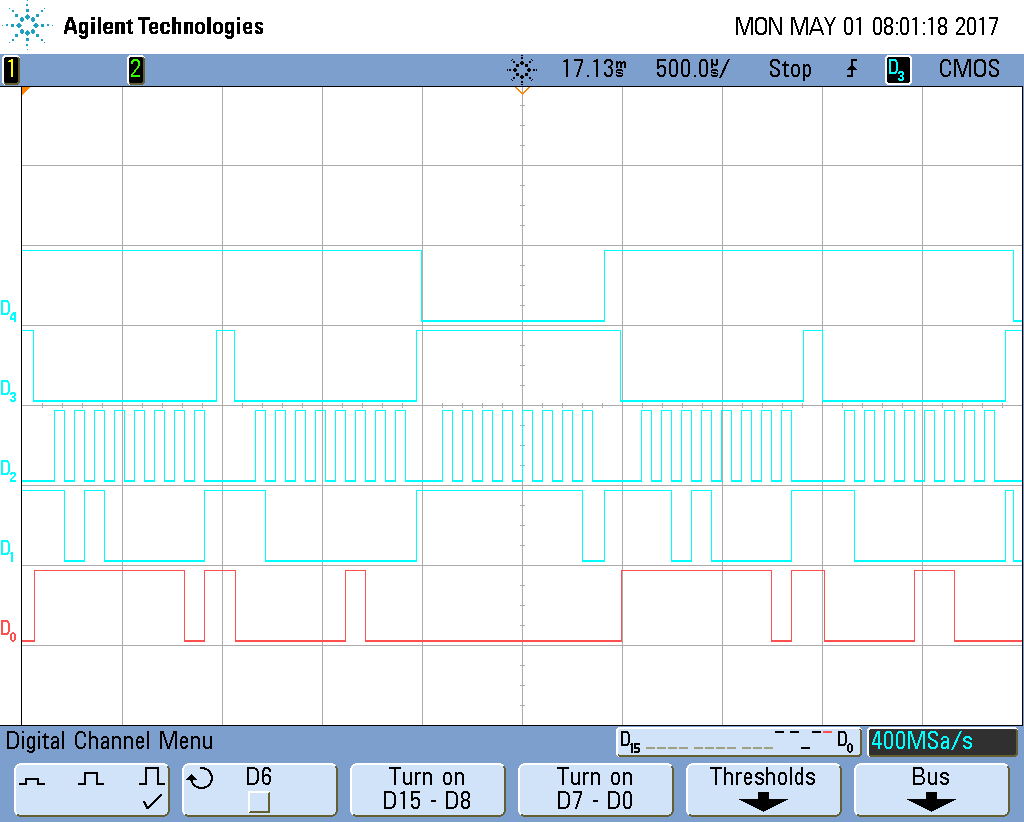


Figure 20: Oscope[[3]](#footnote-2) capture of the following transactions: {tap=SDR, DR[8:0]=0, tap=RTI}

## Demo Recap:

A demonstration of the ***spi2jtag***  embedded system in (Fig. 6)was performed by Nathan and Jordan on 2017-05-03, in the presence of Dr. Fourney. This presentation was video recorded and will be producible upon request. The block diagram presented at this demonstration is viewable in Fig. 21.

* Live demonstration
  + Showed messages from NIOS2 via JTAG UART that printed data from the IR and a TDR.
  + Development board displayed the values of a TDR which were send from NIOS2.
  + Development board displayed the state of the JTAG TAP FSM TDR which were changed from NIOS2.
* Data collected from OScope
  + The SPI signals for transactions of various JTAG TAP FSM transitions were observed on an O-SCOPE from GPIO pins on the FPGA dev board (Also described above).
  + The JTAG signals were observed on an on the FPGA dev board Hex displays.

## Future Work:

Use library with a different SPI driver by replacing ***altera\_spi\_transfer()*** in ***altera\_spi.h***.

***jtag\_tap\_ctrl*** should provide ports to connect to the TDR’s TDO and TDI. The multiplexing performed in hipe\_spi should be performed internally to this module.

The issue the causes an extra shift during SDR should be fixed in a different way. One possible solution could be for the JtagDevice to transition to CDR before shifting data rather than SDR. The current code essentially leaves the SDR or CDR state immediately when jtag\_inf.tms goes low. The particular lines of interest are shown below:

|  |
| --- |
| ***ee492\projects\spi2jtag\hdl\jtag\_tap\jtag\_tap\_ctrl.sv***  37: assign ir\_inf.sdr = (one\_hot\_state.sir **& (~jtag.tms)**); // Shift Instruction Register |

|  |
| --- |
| ***ee492\projects\spi2jtag\hdl\spi2jtag\hipe\_spi.sv***  092: hipe\_jtag #(  097: ) tdrs (  102: .sdr({onehot\_state.sdr **& (~jtag.tms)**}),  114: ); |

Currently JtagTap is limited to transitioning only to stable states. This limitation could be removed with a little bit of trivial work. The following functions and their uses should be considered.

|  |
| --- |
| ***ee492\projects\spi2jtag\quartus\software\AlteraNios2\include\spi2jtag\JtagTap.h***  23: inline uint8\_t **jtag\_tms**(JtagTapStateEnum a, JtagTapStateEnum b){  64: inline uint8\_t **jtag\_transition**(JtagTapStateEnum a, JtagTapStateEnum b){ |

## Appendix:

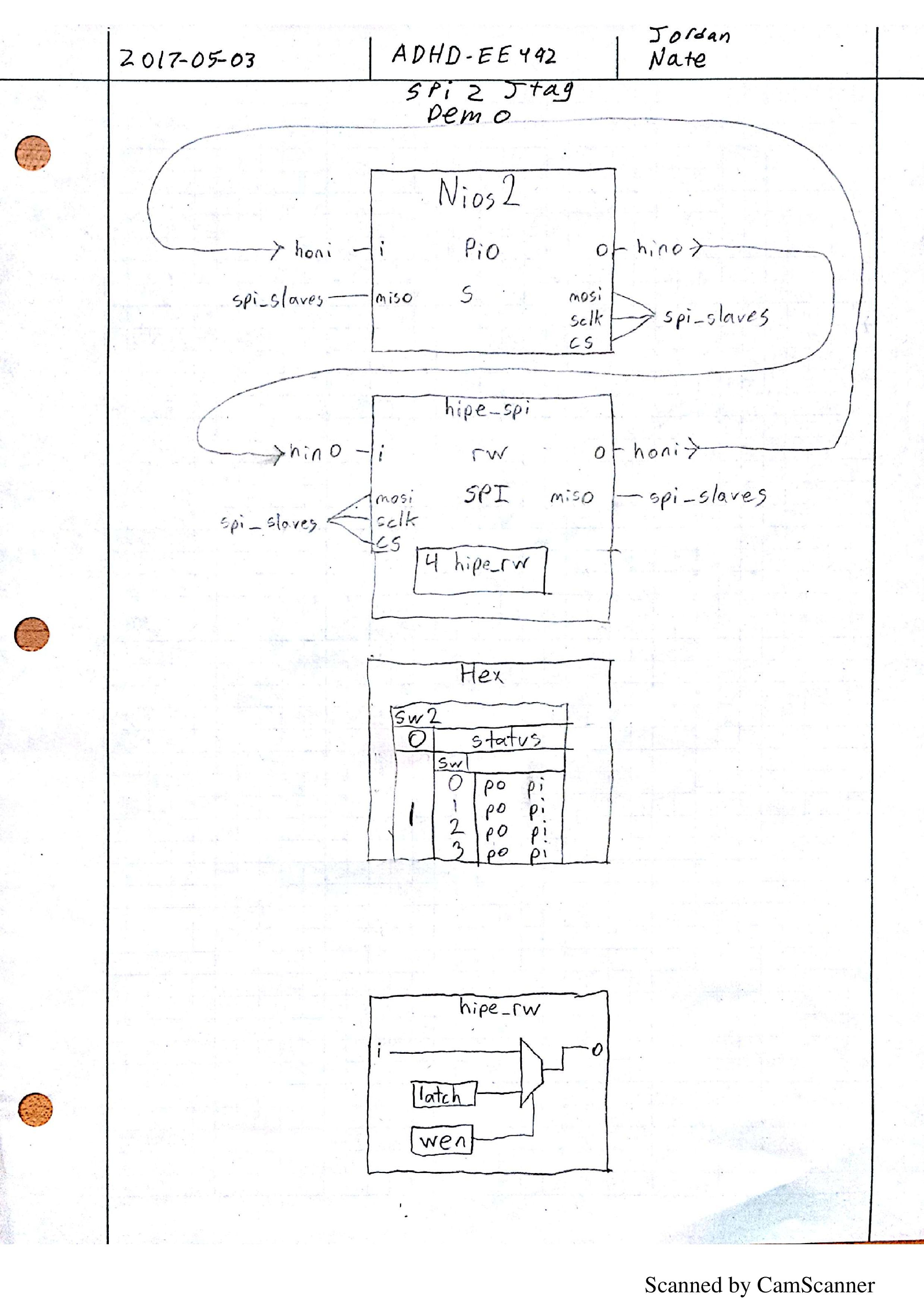


Figure 21: 2017-05-03-EE492-SPI2JTAG-DEMO-Figure-Engr-paper

1. Signals from bottom to top: MOSI, MISO, SCLK, CS\_n[0] (TAP), CS\_n[1] (DR/IR). [↑](#footnote-ref-0)
2. Signals from bottom to top: MOSI, MISO, SCLK, CS\_n[0] (TAP), CS\_n[1] (DR/IR). [↑](#footnote-ref-1)
3. Signals from bottom to top: MOSI, MISO, SCLK, CS\_n[0] (TAP), CS\_n[1] (DR/IR). [↑](#footnote-ref-2)