AUTHOR

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Department** |  | **Signature** |  | **Print Name** |  | **Date** |
| R&D |  |  |  |  |  |  |

SIGNATURES

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Department** |  | **Signature** |  | **Print Name** |  | **Date** |
| R&D |  |  |  |  |  |  |

Revision History

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Rev.** |  | **Date** |  | **Initials** |  | **Description** |
| 0.1 |  | 13-Apr-2009 |  | KW |  | Initial draft roughly based on Dyonics II SPI Protocol Verification rev 0.4 |
| 0.2 |  | 31-Aug-2009 |  | KW |  | Corrected expected Data for PowerMini with Hand Controls. (Was without Hand Controls). |
| A |  | 16-Oct-2009 |  | DAT |  | Initial Release |
| A.1 |  | 04-Sep-2019 |  | KW |  | Update to docx, integrate D25 changes |
| A.2 |  | 10-Sep-2019 |  | KW |  | Changes from dry run |
| A.3 |  | 10-Sep-2019 |  | KW |  | A few more missed corrections |
| B |  | 26-Sep-2019 |  | DAT |  | Update to Revision B |

Glossary

**DIIProtocolFull.tla** – A Tektronix TLA5202B Logic Analyzer configuration file setup for the DYONICS II EIP. A copy of **DIIProtocolFull.tla** can be obtained via VSS using the “Sustaining” database at “[\\Usashsmnas01\VSS\Controlsgroup](file:///\\Usashsmnas01\VSS\Controlsgroup)” under “**$/R\_D/DII-EIP/Software/Design Control Documents”**.

**Dsp application** – The application that runs under the Motor Controller. It can be obtained via VSS using the “Sustaining” database at “[\\Usashsmnas01\VSS\Controlsgroup](file:///\\Usashsmnas01\VSS\Controlsgroup)” under “**$/R\_D/DII-EIP/Dsp**”.

**Shaver application** – The application that runs under the System Controller. It can be obtained via VSS using the “Sustaining” database at “[\\Usashsmnas01\VSS\Controlsgroup](file:///\\Usashsmnas01\VSS\Controlsgroup)” under “**$/R\_D/DII-EIP/Applications/Shaver**”.

References

1. 15000721 – DYONICS II EIP Inter Controller Protocol
2. 11500060 – Schematic, Dyonics II EIP Printed Wiring Board

Table of Contents

[1. OVERVIEW 2](#_Toc18494797)

[2. VERIFICATION OF THE INTER-PROCESSOR COMMUNICATION 3](#_Toc18494798)

[2.1 Requirements 3](#_Toc18494799)

[2.2 Setup Instructions 4](#_Toc18494800)

[2.3 Test Script Instructions 5](#_Toc18494801)

[3. SUMMARY / REPORT 24](#_Toc18494802)

[3.1 NOTES 24](#_Toc18494803)

[3.2 OVERALL PASS / FAIL STATUS 24](#_Toc18494804)

Inter Controller Protocol Verification

# OVERVIEW

The purpose of this document is to:

* + Show the traceability of the verification procedures against the DYONICS II EIP Inter Controller Protocol
  + Provide a summary of the Verification procedures
  + Provide a summary of the Verification results
  + Provide a link to more detailed internal engineering verification reports when required

Verification is defined as the process by which the design output meets the design input requirements. The results of the design verification, including identification of the design, method(s), the date, and the individual(s) performing the verification, shall be documented in the DHF. Design verification shall be traceable to product specifications.

The verification is traceable to the following functional requirement and specification documents:

Dyonics II EIP Inter Controller Protocol – Document # 15000721

# VERIFICATION OF THE INTER-PROCESSOR COMMUNICATION

The Inter-Processor Communication Protocol is tested in the DYONICS POWER II Control System, which uses the protocol to communicate between the System Controller and the Motor Controller.

## Requirements

Inter-Processor Communication Protocol Unit testing was performed with following hardware:

|  |  |  |
| --- | --- | --- |
| Device | REF/Part Number | Serial Number/Lot Number |
| DYONICS POWER II Control System with TestApp 2.00.02 and  Motor Controller 2.00.00 | 72200873 |  |
| DYONICS POWERMINI MDU with Hand Controls | 72201500 |  |
| DYONICS POWERMAX Elite MDU with Hand Controls | 72200616 |  |
| Tektronix Logic Analyzer | TLA5202B |  |
| USB Key | 91000903 |  |
| C2G DB25 M/F Serial RS232 Cable (6 feet) | 02655 |  |

Addition equipment required for this testing:

* Dsp Parallel to 10 pin Debug Adapter
* Crossover Ethernet Cable
* Null modem 9 pin Serial Cable
* A Windows XP computer (PC) that has a free Ethernet and Parallel port and can run the following software:
  + - Platform Builder for Microsoft Windows CE 5.0
    - Microsoft eMbedded Visual C++ 4.0 software SP4
    - Freescale CodeWarrior for DSC56800E v8.2.3

## Setup Instructions

The Logic Analyzer needs to be wired up to the DYONICS POWER II Control System as follows:

| **System Controller** | | **Motor Controller (U2)** | | **Local Analyzer** | **Function** |
| --- | --- | --- | --- | --- | --- |
| **GPIO** | **Pin** | **GPIO** | **Pin** |  |  |
| PB22 | G17 | GPIOA8 | 154 | A0 - 0 | Master to Slave IRQ |
| PB23 | G15 | GPIOA9 | 10 | A0 - 1 | Slave to Master IRQ |
| PB24 | H16 | GPIOA0 | 19 | A0 - 2 | Master Xmt Toggle |
| PB25 | F15 | GPIOA1 | 20 | A0 - 3 | Master Rcv Toggle |
| PB26 | F14 | GPIOA2 | 21 | A0 - 4 | Master Output |
| PB27 | F17 | GPIOA3 | 22 | A0 - 5 | Reserved |
| PB28 | G16 | GPIOA4 | 23 | A0 - 6 | Slave Xmt Toggle |
| PB29 | F16 | GPIOA5 | 24 | A0 - 7 | Slave Rcv Toggle |
| PB30 | E15 | GPIOA6 | 25 | A1 - 0 | Slave Ouput |
| PB31 | E17 | GPIOA7 | 26 | A1 - 1 | Slave Ready |
| PE0 | K3 | GPIOF0 | 28 | A2 - 0 | Data Bit D0 |
| PE1 | K4 | GPIOF1 | 29 | A2 - 1 | Data Bit D1 |
| PE2 | L6 | GPIOF2 | 30 | A2 - 2 | Data Bit D2 |
| PE3 | L7 | GPIOF3 | 32 | A2 - 3 | Data Bit D3 |
| PE4 | L2 | GPIOF4 | 149 | A2 - 4 | Data Bit D4 |
| PE5 | L5 | GPIOF5 | 150 | A2 - 5 | Data Bit D5 |
| PE6 | K1 | GPIOF6 | 151 | A2 - 6 | Data Bit D6 |
| PE7 | L4 | GPIOF7 | 152 | A2 - 7 | Data Bit D7 |
| PE8 | M7 | GPIOF8 | 153 | A3 - 0 | Data Bit D8 |
| PE9 | L3 | GPIOF9 | 70 | A3 - 1 | Data Bit D9 |
| PE10 | L1 | GPIOF10 | 71 | A3 - 2 | Data Bit D10 |
| PE11 | M4 | GPIOF11 | 83 | A3 - 3 | Data Bit D11 |
| PE12 | M6 | GPIOF12 | 86 | A3 - 4 | Data Bit D12 |
| PE13 | M5 | GPIOF13 | 88 | A3 - 5 | Data Bit D13 |
| PE14 | M2 | GPIOF14 | 89 | A3 - 6 | Data Bit D14 |
| PE15 | M3 | GPIOF15 | 90 | A3 - 7 | Data Bit D15 |

Setup the rest of the cables and adapters as follows:

* Remove the cover of the DYONICS POWER II Control System
* Attach the Crossover Ethernet Cable between the PC and **J25** of the DYONICS POWER II Control System
* Attach one end of the C2G DB25 Pin Cable to the PC
* Attach the other end of the C2G DB25 Pin Cable to the Dsp Parallel to 10 pin Debug Adapter
* Plug the Adapter into the 10 pin connector on the DYONICS POWER II, (**J12**)
* Lift switch **SW1** #2 on the DYONICS POWER II up
* Remove all object files, download, start and stop the Dsp application
* Attach the Null modem 9 pin Serial Cable to the PC and the serial debug port on the DYONICS POWER II, (**P1**)

## Test Script Instructions

The following is a list of all the tests. Each test uses one or more of the elements below:

|  |  |
| --- | --- |
| **Element** | **Description** |
| Shaver | Use eMbedded Visual C++ 4.0 to debug the Shaver application on the System Controller. |
| Platform Builder | Use Platform Builder 5.0 to create a custom build of the DYONICS POWER II software for testing. |
| Logic Analyzer | Use a Tektronix TLA5202B Logic Analyzer to record the transitions of the I/O lines used in the Protocol. |
| Motor Controller | Use a Freescale CodeWarrior DSC56800E v8.2.3 environment to debug Dsp application on the Motor Controller. |

The System Controller is the master and the Motor Board is the slave in all these tests.

| **Section** | **Verification Procedure Summary** | **Element** | **Summary Results**  **(may include links to other verification reports)** | **P / F** | **Initials / Date** |
| --- | --- | --- | --- | --- | --- |
| 2.1 a  2.1.b | Verify that transactions on the bus occur in units of 16-bit words.  Verify the sequence for sending of a command. | Shaver, Logic Analyzer | Shaver:  Turn on the Shaver and kill TestApp by touching the X or OK in the upper right hand corner. Modify the shaver code software by inserting the following code right before the line return TRUE; in the function InitDriver(void) from the file Driver.cpp:  {  SnWord wTmp;  SnBool yRet;    // GET + CRC + ACK  yRet = ReadWordFromDevice(1, &wTmp);  return yRet;  }  Set a breakpoint on the call to return yRet;  Shaver and Logic Analyzer:  Turn on the Logic Analyzer and load the system file **DIIProtocolFull.tla** via the  **File->Load System…** option on the Logic Analyzer.  Start the Logic Analyzer capture by pressing the RUN/STOP button. Press the F5 key on the Shaver Application to run to the breakpoint. Stop the Logic Analyzer capture by pressing the RUN/STOP button again.  Ignoring Sample #0, (which is not associated with a I/O transition), verify against the Full Listing window that:   * The size of the Data field is 16 bits. * Master sets the **Master to Slave IRQ** line (**S-IRQ**) low. * The Slave sets **Slave Output** line (**S-Out**) low. * Master sets the **Master Output** (**M-Out**)line high. * Master sets the **Data** to 0x1001, toggles **Master Xmt Toggle** line (**M-Xmt**). * The Slave toggles the **Slave Rcv Toggle** line (**S-Rcv**). * Master sets the **Data** to 0xC059, toggles **Master Xmt Toggle** line (**M-Xmt**). * The Slave toggles the **Slave Rcv Toggle** line (**S-Rcv**). * Master sets the **Master Output** (**M-Out**)line low. * The Slave sets **Slave Output** line (**S-Out**) high. * Slave sets the **Data** to 0x0402, toggles **Slave Xmt Toggle** line (**S-Xmt**). * The Master toggles the **Master Rcv Toggle** line (**M-Rcv**). * Slave sets the **Data** to 0xD0 F0, toggles **Slave Xmt Toggle** line (**S-Xmt**). * The Master toggles the **Master Rcv Toggle** line (**M-Rcv**). * Master sets the **Master to Slave IRQ** line (**S-IRQ**) high.   Verify that yRet is set to 1 meaning the function returned success.  Undo the code change to the Shaver application and turn off the Shaver and Logic Analyzer. |  |  |
| 2.1 b | Verify there is at least a 3ms timeout when the Master waits for the Slave to set its **Slave Output** line low. | Shaver,  Motor,  Logic Analyzer | Shaver:  Turn on the Logic Analyzer and load the system file **DIIProtocolFull.tla** via the  **File->Load System…** option on the Logic Analyzer.    Turn on the Shaver and kill TestApp by touching the X or OK in the upper right hand corner. Modify the shaver code software by inserting the following code right before the line return TRUE; in the function InitDriver(void) from the file Driver.cpp:  {  SnWord wTmp;  SnBool yRet;    // GET + CRC + ACK  yRet = ReadWordFromDevice(1, &wTmp);  return yRet;  }  Set a breakpoint on the call to return yRet;  Motor:  Set the initial value of g\_wHeartbeat to zero in Msg.c:  SnWord g\_wHeartbeat = 0;  Set a breakpoints in the Dsp code software in the function void Msg\_Interrupt(void) from the file Msg.c, (noted by the ● at the beginning of the line):  // Master will not start sending the command until we have  // disabled Slave Output  ● if (!SetSlaveDataOutput(FALSE))  goto ExitMsg;  Remove all object files, download and start the Motor application.  Shaver, Motor and Logic Analyzer:  Start the Logic Analyzer capture by pressing the RUN/STOP button. Press the F5 key on the Shaver Application to run to the breakpoint. Stop the Logic Analyzer capture by pressing the RUN/STOP button again.  Ignoring Sample #0, (which is not associated with an I/O transition), verify against the Full Listing window that there is at least a 3ms period of time between every transition of S-IRQ from 0 to 1.  Verify that there are 3 transitions of S-IRQ from 1 to 0 in the Full listing and yRet is set to 0, representing that the command is attempted 3 times before returning a failure.  Undo the code change to the Shaver application and Motor application then turn off the Shaver. |  |  |
| 2.1 b | Verify there is at least a 3ms timeout when the Master waits for the Slave to toggle its **Slave Rcv Toggle** line. | Shaver,  Motor,  Logic Analyzer | Shaver:  Turn on the Shaver and kill TestApp by touching the X or OK in the upper right hand corner. Modify the shaver code software by inserting the following code right before the line return TRUE; in the function InitDriver(void) from the file Driver.cpp:  {  SnWord wTmp;  SnBool yRet;    // GET + CRC + ACK  yRet = ReadWordFromDevice(1, &wTmp);  return yRet;  }  Set a breakpoint on the call to return yRet;  Motor:  Set the initial value of g\_wHeartbeat to zero in Msg.c:  SnWord g\_wHeartbeat = 0;  Set a breakpoints in the Dsp code software in the function void Msg\_Interrupt(void) from the file Msg.c, (noted by the ● at the beginning of the line):  // Read Command  ● if (!GetWordFromMaster(&wTmp))  goto ExitMsg;  Remove all object files, download and start the Motor application.  Shaver, Motor and Logic Analyzer:  Start the Logic Analyzer capture by pressing the RUN/STOP button. Press the F5 key on the Shaver Application to run to the breakpoint. Stop the Logic Analyzer capture by pressing the RUN/STOP button again.  Ignoring Sample #0, (which is not associated with an I/O transition), verify against the Full Listing window that there is at least a 3ms period of time between every transition of S-IRQ from 0 to 1.  Verify that there are 3 transitions of S-IRQ from 1 to 0 in the Full listing and yRet is set to 0, representing that the command is attempted 3 times before returning a failure.  Undo the code change to the Shaver application and Motor application then turn off the Shaver and Logic Analyzer. |  |  |
| 2.1 b | Verify there is at least a 3ms timeout when the Master waits for the Slave to toggle its **Slave Xmt Toggle** line. | Shaver,  Motor,  Logic Analyzer | Shaver:  Turn on the Logic Analyzer and load the system file **DIIProtocolFull.tla** via the  **File->Load System…** option on the Logic Analyzer.  Turn on the Shaver and kill TestApp by touching the X or OK in the upper right hand corner. Modify the shaver code software by inserting the following code right before the line return TRUE; in the function InitDriver(void) from the file Driver.cpp:  {  SnWord wTmp;  SnBool yRet = TRUE;    // GET + CRC + ACK  yRet = ReadWordFromDevice(1, &wTmp);  return yRet;  }  Set a breakpoint on the call to return yRet;  Motor:  Set the initial value of g\_wHeartbeat to zero in Msg.c:  SnWord g\_wHeartbeat = 0;  Set a breakpoints in the Dsp code software in the function void Msg\_Interrupt(void) from the file Msg.c, (noted by the ● at the beginning of the line):  ● SendWordToMaster((ACK << 12) | wCrc);  Remove all object files, download and start the Motor application.  Shaver, Motor and Logic Analyzer:  Start the Logic Analyzer capture by pressing the RUN/STOP button. Press the F5 key on the Shaver Application to run to the breakpoint. Stop the Logic Analyzer capture by pressing the RUN/STOP button again.  Ignoring Sample #0, (which is not associated with an I/O transition), verify against the Full Listing window that there is at least a 3ms period of time between every transition of S-IRQ from 0 to 1.  Verify that there are 3 transitions of S-IRQ from 1 to 0 in the Full listing and yRet is set to 0, representing that the command is attempted 3 times before returning a failure.  Undo the code change to the Shaver application and Motor application then turn off the Shaver and Logic Analyzer. |  |  |
| 2.1 b  2.1.2 | Verify there is at least a 1 second timeout when the Master waits for the Slave to toggle the **Slave to Master IRQ** line. | Shaver,  Motor,  Logic Analyzer | Shaver:  Turn on the Logic Analyzer and load the system file **DIIProtocolFull.tla** via the  **File->Load System…** option on the Logic Analyzer.  Turn on the Shaver and kill TestApp by touching the X or OK in the upper right hand corner. Modify the shaver code software by inserting the following code right before the line return TRUE; in the function InitDriver(void) from the file Driver.cpp:  {  volatile SnQByte qData;  SnWord pwData[512];  SnBool yRet = TRUE;    // FLASH + CRC + ACK + READY  for (qData = 0; qData < 512; qData++)  pwData[qData] = (SnWord)qData;  yRet = FlashBlockToDevice(0, pwData);  // GET + CRC + ACK  ReadWordFromDevice(1, pwData);  return yRet;  }  Set a breakpoint on the call to return yRet;  Motor:  Set the initial value of g\_wHeartbeat to zero in Msg.c:  SnWord g\_wHeartbeat = 0;  Set a breakpoints in the Dsp code software in the function void Msg\_Interrupt(void) from the file Msg.c, (noted by the ● at the beginning of the line):  case CMD\_WR\_FLASH\_BLK:  ● sg\_eMsgWait = MSG\_WAIT\_FOR\_FLASH;  break;  Remove all object files, download and start the Motor application.  Shaver, Motor and Logic Analyzer:  Start the Logic Analyzer capture by pressing the RUN/STOP button. Press the F5 key on the Shaver Application to run to the breakpoint. Stop the Logic Analyzer capture by pressing the RUN/STOP button again.  Ignoring Sample #0, (which is not associated with a I/O transition), verify against the Full Listing window that there is at least a 1 second period of time between the end of the FlashBlockToDevice command, (S-IRQ goes from 0 to 1), and the start of the first attempt of the ReadWordFromDevice command, (S-IRQ goes from 1 to 0).  Verify that the FlashBlockToDevice command is attempted only once, ReadWordFromDevice is attempted 3 times and yRet is set to 0, (FlashBlockToDevice returned failure).  Undo the code change to the Shaver application and Motor application then turn off the Shaver and Logic Analyzer. |  |  |
| 2.1 b  2.1 c  2.1.1 | Verify the NAK response. | Shaver, Logic Analyzer | Shaver:  Turn on the Logic Analyzer and load the system file **DIIProtocolFull.tla** via the  **File->Load System…** option on the Logic Analyzer.  Turn on the Shaver and kill TestApp by touching the X or OK in the upper right hand corner. Modify the shaver code software by inserting the following code right before the line return TRUE; in the function InitDriver(void) from the file Driver.cpp:  {  SnWord wTmp;  SnBool yRet = TRUE;    // GET + CRC + ACK  yRet = ReadWordFromDevice(1, &wTmp);  return yRet;  }  Set a breakpoint on the call to return yRet;  Shaver and Logic Analyzer:  Insert a jumper into the Logic Analyzer connector board next to the A3 connector that ties D0 to ground. Start the Logic Analyzer capture by pressing the RUN/STOP button. Press the F5 key on the Shaver Application to run to the breakpoint. Stop the Logic Analyzer capture by pressing the RUN/STOP button again. Remove the jumper.  Ignoring Sample #0, (which is not associated with a I/O transition), verify against the Full Listing window that a NAK response was returned because of the bad CRC 3 times and then the function return a status of 0, (yRet is set to 0).  Undo the code change to the Shaver application and turn off the Shaver and Logic Analyzer. |  |  |
|  | Verify that NACK can be used as a response to a command. |  | The data above shows an example of the NACK used as a response to a command. | N/A | N/A |
|  | Verify that the response to the command received an NAK with the computed CRC over all data sent by the slave occupying the lower 8 bits of that word. |  | The data above shows the response to the command received an NAK that has the correct CRC value of all the slave data sent in the lower 8 bits of the word. | N/A | N/A |
| 2.1 b  2.1 c  2.1 d  2.1.1  2.1.2  2.2  2.2.1 | Verify commands. | Shaver, Logic Analyzer | Shaver:  Turn on the Logic Analyzer and load the system file **DIIProtocolData.tla** via the  **File->Load System…** option on the Logic Analyzer.  Plug a DYONICS POWERMINI MDU in Port A of the Shaver.  Modify the shaver code software by inserting the following code right before the line return TRUE; in the function InitDriver(void) from the file Driver.cpp:  {  #include "ControllerTypes.h"  volatile SnQByte qData;  SnWord pwData[512];  SnBool yGet, yPut, yRead, yWrite, yFlash;  SnBool yGoodSerial, yBadSerial;    // GET + CRC + ACK  yGet = ReadWordFromDevice(1, pwData);  // PUT + CRC + ACK  yPut = WriteWordToDevice(0, 0x5555);  // READ + CRC + ACK  yRead = ReadWordsFromDevice(0, 2, pwData);  // WRITE + CRC + ACK  pwData[0] = 0x1234;  pwData[1] = 0x5678;  yWrite = WriteWordsToDevice(0, 2, pwData);  // FLASH + CRC + ACK + READY  for (qData = 0; qData < 512; qData++)  pwData[qData] = (SnWord)qData;  yFlash = FlashBlockToDevice(0, pwData);  // SERIAL + CRC + ACK + READY + READ + ACK  pwData[0] = 0x0db1;  yGoodSerial = SerialCmdsToDevice((offsetof(Status\_Control,  tSerial.wCmdResult))/2, 1, pwData, 4);  // SERIAL + CRC + ACK + FAULT (No MDU connected to Port A)  pwData[0] = 0x0db1;  yBadSerial = SerialCmdsToDevice((offsetof(Status\_Control,  tSerial.wCmdResult))/2, 1, pwData, 4);  return yBadSerial;  }  Set a breakpoint on the second call to SerialCmdsToDevice() and the return yBadSerial; line of the new code segment.  Shaver and Logic Analyzer:  Turn on the Shaver and kill TestApp by touching the X or OK in the upper right hand corner. Start the Logic Analyzer capture by pressing the RUN/STOP button. Press the F5 key on the Shaver Application to run to the first breakpoint. Remove the DYONICS POWERMINI MDU from Port A and continue execution of the Shaver application till the second breakpoint. Stop the Logic Analyzer capture by pressing the RUN/STOP button again.  Verify the commands below from the Data Listing capture with the following rules:   * Ignore sample #0. * When M-Out is 1 the Master is sending Data * When M-Out is 0 the Slave is sending Data * When M-IRQ toggles and S-Rdy is 1, the Slave sent a Ready to the Master * When M-IRQ toggles and S-Rdy is 0, the Slave sent a Fault to the Master   Command: GET + CRC + ACK  Master: 1001 C059  Slave: 0402 D0F0  Command: PUT + CRC + ACK  Master: 2000 5555 C0F8  Slave: D000    Command: READ + CRC + ACK  Master: 3002 0000 C0D9  Slave: 5555 0402 D0E7  Command: WRITE + CRC + ACK  Master: 4002 0000 1234 5678 C0C4  Slave: D000  Command: FLASH + CRC + ACK + READY  Master: 5000 0000 0001 0002 .... 01FF C05C  Slave: D000  Slave sends Ready, (S-Rdy is 1)  Command: SERIAL + CRC + ACK + READY  Master: 6301 0DB1 C0F5  Slave: D000  Slave sends Ready, (S-Rdy is 1)  Command: READ + CRC + ACK (Called from SerialCmdsToDevice function).  Master: 3001 0293 C07B  Slave: 5402 D02B  Command: SERIAL + CRC + ACK + FAULT  Master: 6301 0DB1 C0F5  Slave: D000  Slave sends Fault, (S-Rdy is 0)  Verify that yGet, yPut, yRead, yWrite, yFlash and yGoodSerial are all set to 1 indicating the command was executed successfully.  Verify that yBadSerial is set to 0 indicating the command failed.  Undo the code change to the Shaver application and remove all wires and cables from the Shaver. Turn off the Shaver and Logic Analyzer. |  |  |
|  | Verify that the command and response values are in the upper 4 bits (MSB) of the word. |  | The upper 4 bits of the command and response words in the data above match table 2.1.c of the protocol. |  |  |
|  | Verify that GET, PUT, READ, WRITE, FLASH, and SERIAL are sent as commands to the slave and acknowledged. |  | The data above shows that GET, PUT, READ, WRITE, FLASH and SERIAL commands all get acknowledged by the slave sending an ACK (0xD0XX). |  |  |
|  | Verify that ACK can be used as a response to a command. |  | The data above shows examples of the ACK used as a response to a command. |  |  |
|  | Verify that the **Slave Ready** line (S-Rdy) is used to report success or failure of a slow command. |  | The data above shows examples of the Slave Ready line is used to report success or failure of a slow command. The data above shows that the slave set the **Slave Ready** line high in response to a valid slow command and low for the SERIAL command when the DYONICS POWERMINI MDU was removed from Port A. |  |  |
|  | Verify that all commands are terminated with the correct CRC value in the lower 8 bits of the word. |  | The data above shows all commands are either terminated with a CRC value in the lower 8 bits. The CRC value is the correct value for each command. |  |  |
|  | Verify that all responses to a command receive an ACK with the computed CRC over all data sent by the slave occupying the lower 8 bits of that word. |  | The data above shows all commands are responded to with and ACK that has the correct CRC value of all the slave data sent in the lower 8 bits of the word. |  |  |
|  | Verify that for the FLASH and SERIAL commands a READY/FAULT response is sent back from the slave. |  | The data above shows that for the FLASH and SERIAL commands a READY/FAULT response is sent back from the slave. |  |  |
|  | Verify that no commands are sent from master while waiting for a READY/FAULT response. |  | The data above shows that no words were sent from master before receiving the handshake interrupt from the slave and the READY/FAULT response was queried. |  |  |
|  | Verify that commands can be sent from master after the READY/FAULT response. |  | The data above shows that commands can be sent from master after the READY/FAULT response. |  |  |
|  | Verify that the GET command has the offset in the lower 12 bits of the command word. |  | The data above shows the offset of 1 passed into the function ReadWordFromDevice() appears in the lower 12 bits of the GET command word (0x1001). |  |  |
|  | Verify that the GET command returns the read value as the second word from the slave. |  | The data above shows that the GET command returns the read value for the memory offset given as the second word (0x0402) from the slave. |  |  |
|  | Verify that the PUT command has the offset in the lower 12 bits of the command word. |  | The data above shows the offset of 0 passed into the function WriteWordToDevice() appears in the lower 12 bits of the PUT command word (0x2000). |  |  |
|  | Verify that the PUT command sends the write value as the second word to the slave. |  | The data above shows that write value passed into the function WriteWordToDevice() appears as the second word (0x5555) sent from the master to the slave for the PUT command. |  |  |
|  | Verify that the READ command has the number of words to read in the lower 12 bits of the command word. |  | The data above shows that the number of words to read (2) is passed into the function ReadWordsFromDevice() and appears in the lower 12 bits of the command word (0x3002). |  |  |
|  | Verify that the READ command sends the offset as the second word to the slave. |  | The data above shows that offset passed into the function ReadWordsFromDevice() appears as the second word (0x0000) sent from the master to the slave for the READ command. |  |  |
|  | Verify that the READ command returns the read values as the third and fourth words from the slave. |  | The data above shows that the READ command returns the read values for the memory offset given as the third and fourth words (0x5555 0x0402) from the slave. |  |  |
|  | Verify that the WRITE command has the number of words to write in the lower 12 bits of the command word. |  | The data above shows that the number of words to write (2) is passed into the function WriteWordsToDevice() and appears in the lower 12 bits of the command word (0x4002). |  |  |
|  | Verify that the WRITE command sends the offset as the second word to the slave. |  | The data above shows that offset passed into the function WriteWordsToDevice() appears as the second word (0x0000) sent from the master to the slave for the WRITE command. |  |  |
|  | Verify that the WRITE command sends the write values as the third and fourth words to the slave. |  | The data above shows that write values passed into the function WriteWordsToDevice() appears as the third and fourth words (0x1234 0x5678) sent from the master to the slave for the WRITE command. |  |  |
|  | Verify that the FLASH command has the 24 bit address of flash memory to write in the lower 8 bits of the command word and the second word sent to the slave. |  | The data above shows that the address of the flash memory to write (0) is passed into the function FlashBlockToDevice() and the MSB appears in the lower 8 bits of the command word (0x5000) and the LSB is the second word sent to the slave. |  |  |
|  | Verify that the FLASH command sends 512 words of flash data following the first two words sent to the slave. |  | The data above shows that the 512 words of flash data (a ramp starting at 0 and going 0x1ff) passed into the function FlashBlockToDevice() is sent to the slave following the first two words. |  |  |
|  | Verify that the SERIAL command has the Serial Command Timeout - 1 in bits 8-11 of the command word. |  | The data above shows that the serial command timeout (4) is passed into the function SerialCmdsToDevice() and appears as (serial command timeout – 1) in bits 8-11 of the command word (0x6301). |  |  |
|  | Verify that the SERIAL command has the number of words to read in the lower 8 bits of the command word. |  | The data above shows that the number of words to write (1) is passed into the function SerialCmdsToDevice() and appears in the lower 8 bits of the command word (0x6301). |  |  |
|  | Verify that the SERIAL command sends the write value as the second word to the slave. |  | The data above shows that write value passed into the function SerialCmdsToDevice() appears as the second word (0x0DB1) sent from the master to the slave for the SERIAL command. |  |  |
| 2.1 b | Fault insertion. | Platform Builder, Shaver | Platform Builder:  Using Platform Builder 5.0, open up the TestApp platform.  Modify the Msg.c by inserting the following code right before the function GetTickDelta():  SnQByte qSendWordErr = 0;  SnQByte qGetWordErr = 0;  SnQByte qSetOuputErr = 0;  SnBool PeriodicForcedError(SnQByte \*pqError, SnWord \*pwStr)  {  if (\*pqError >= 200) {  \*pqError = 0;  NKDbgPrintfW(pwStr);  return TRUE;  }  (\*pqError)++;  return FALSE;  }  Modify the Msg.c by inserting the following code right before the line g\_qSlaveRcvToggle ^= SLAVE\_RCV\_TOGGLE; in the function SendWordToSlave().  if (PeriodicForcedError(&qSendWordErr,  TEXT("SendWordToSlave - Forced Error\n")))  return FALSE;  Modify the Msg.c by inserting the following code right before the line g\_qSlaveXmtToggle ^= SLAVE\_XMT\_TOGGLE; in the function GetWordFromSlave().  if (PeriodicForcedError(&qGetWordErr,  TEXT("GetWordFromSlave - Forced Error\n")))  return FALSE;  Modify the Msg.c by inserting the following code right before the line // Enable Xmt in the function SetMasterDataOuput().  if (PeriodicForcedError(&qSetOuputErr,  TEXT("SetMasterDataOuput - Forced Error\n")))  return FALSE;  Re-build the SneDriver Project, creating a new NK.bin image.  Plug a USB key into the PC and use TestBasic.bat to build a new software update key for the DII EIP. Upgrade the DII EIP with the new software key.  Plug a DYONICS POWERMINI MDU in Port A of the Shaver.  Plug a DYONICS POWERMAX Elite into Port B of the Shaver.  Shaver:  Turn on the Shaver and kill TestApp by touching the X or OK in the upper right hand corner. Press the F5 key on the Shaver Application.  Using a Serial Debug program on the PC like Tera Term, (115200 Baud, 8 bit, 1 Stop), verify that there is constant stream of Forced Errors demonstrating that the Fault Insertion code is working.  Using the Shaver Application verify that the DII EIP recovers from all the Forced Errors and operates normally.  Stop the Shaver Application Debug, undo the code change to the Shaver application and remove all wires and cables from the Shaver and PC. Turn off the Shaver and Logic Analyzer. |  |  |

# SUMMARY / REPORT

## NOTES

## OVERALL PASS / FAIL STATUS

|  |  |
| --- | --- |
| Overall Pass / Fail Status |  |
| Date |  |
| Signature |  |
| Printed Name |  |
| Department |  |
| Title |  |