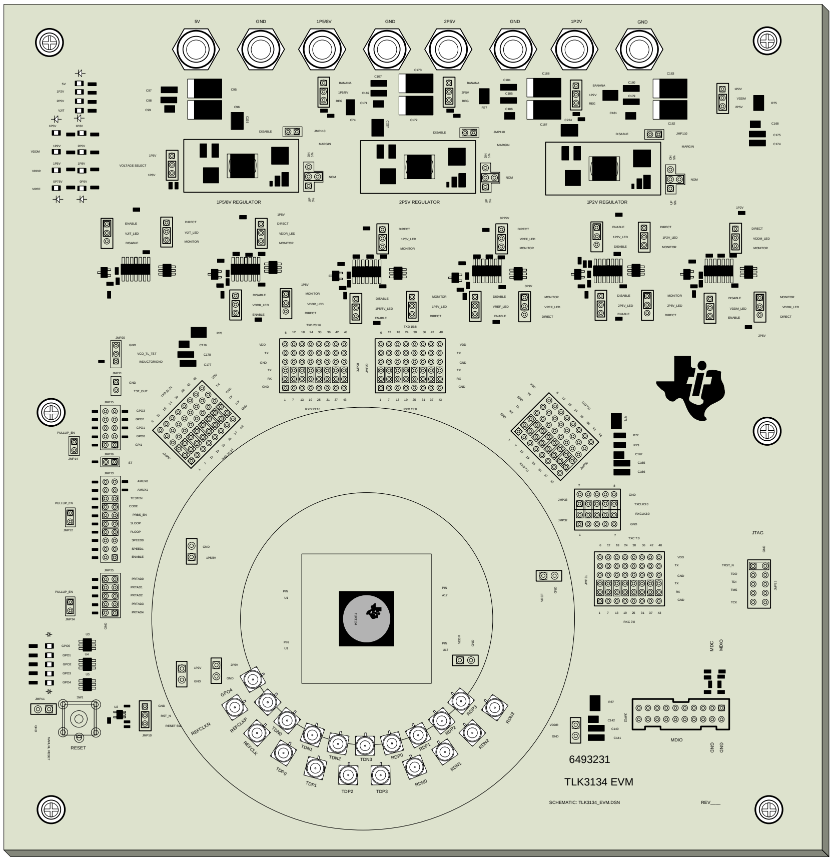
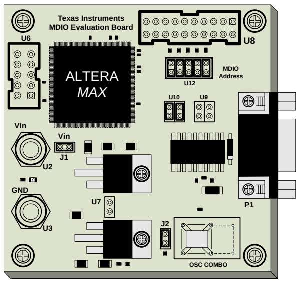
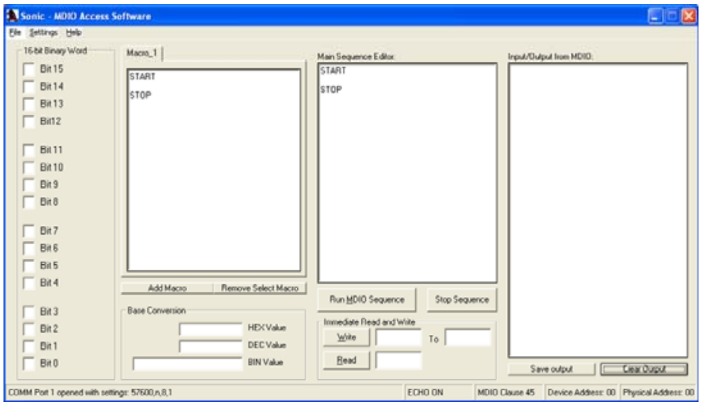
***September** 2008*
Rev 0.2

Sonic MDIO Software and TLK3134 EVM User's Guides Supplemental Information

Jonathan Nerger







START
STOP

START
STOP

COMM Port 1 opened with settings: 57600,n,8,1 | ECHO ON | MDIO Clause 45 | Device Address: 00 | Physical Address: 00

Data Sheet Register Information Syntax:

The TLK3134 Register space is divided into 3 register groups. The first addressable register group is accessible only through the Clause 45 protocol, the second is only accessible through the Clause 22 protocol, and the third is accessible Directly through the Clause 45 protocol but Indirectly through Clause 22 protocol.

Clause 45 Mode (ST = 0):

Registers accessible through the Clause 45 protocol appear in the datasheet and are discussed in a table similar to the following table which is the table for Register 0x0000.

Clause 45 register tables are distinguished by two features not found in the Clause 22 register tables. The first indication that this register responds to Clause 45 protocol is the 4 digit register address. The Clause 22 register address is only 2 digits. The second feature is the “4/5” in the “Bit(s)” column. This indicates that the information contained in the register is valid for both the devices addressed as “04” or “05.”

The information in the “Bit(s)” field of the table is broken up by decimal points according to the following convention.

<Clause 45 device address (4 and 5 if “4/5”)>.<Register Address in DEC>.<Bits in DEC with MSB:LSB>

The Default Value is the HEX value of the register following a Reset or power cycle. The binary value of the HEX Default Value for Register 0x0000 is 0010000001000000 (bit 15: bit 0).

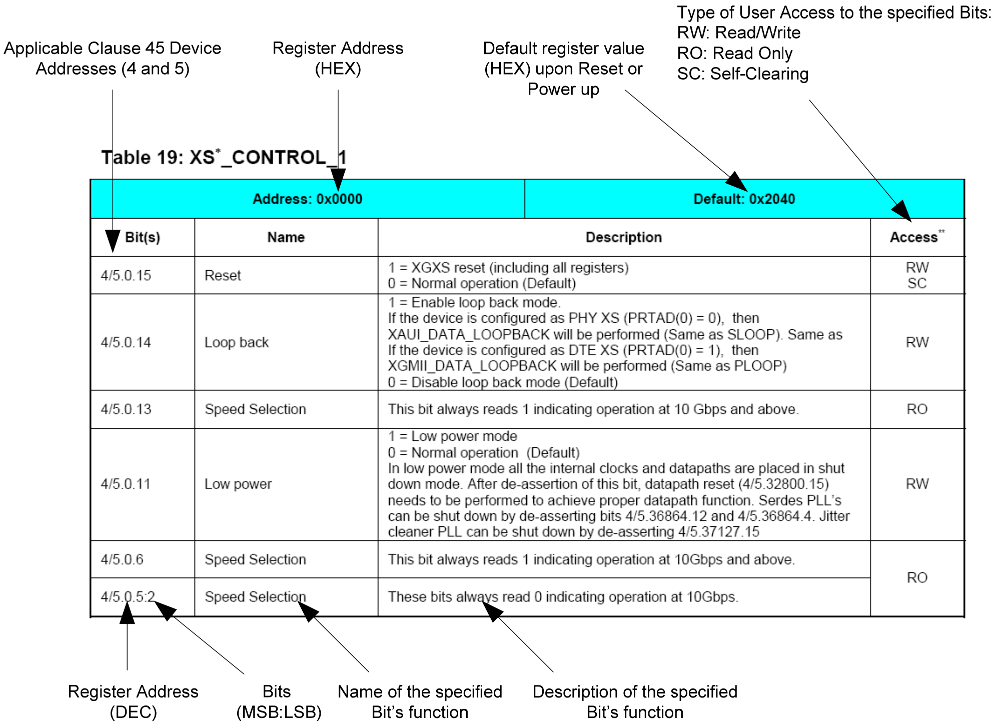

Applicable Clause 45 Device Addresses (4 and 5)
Register Address (HEX)
Default register value (HEX) upon Reset or Power up
Type of User Access to the specified Bits:
RW: Read/Write
RO: Read Only
SC: Self-Clearing
Register Address (DEC)
Bits (MSB:LSB)
Name of the specified Bit’s function
Description of the specified Bit’s function

Table 19: XS*_CONTROL_1

Address: 0x0000		Default: 0x2040	
Bit(s)	**Name**	**Description**	**Access****
4/5.0.15	Reset	1 = XGXS reset (including all registers) 0 = Normal operation (Default)	RW SC
4/5.0.14	Loop back	1 = Enable loop back mode. If the device is configured as PHY XS (PRTAD(0) = 0), then XAUI_DATA_LOOPBACK will be performed (Same as SLOOP). Same as If the device is configured as DTE XS (PRTAD(0) = 1), then XGMII_DATA_LOOPBACK will be performed (Same as PLOOP) 0 = Disable loop back mode (Default)	RW
4/5.0.13	Speed Selection	This bit always reads 1 indicating operation at 10 Gbps and above.	RO
4/5.0.11	Low power	1 = Low power mode 0 = Normal operation (Default) In low power mode all the internal clocks and datapaths are placed in shut down mode. After de-assertion of this bit, datapath reset (4/5.32800.15) needs to be performed to achieve proper datapath function. Serdes PLL’s can be shut down by de-asserting bits 4/5.36864.12 and 4/5.36864.4. Jitter cleaner PLL can be shut down by de-asserting 4/5.37127.15	RW
4/5.0.6	Speed Selection	This bit always reads 1 indicating operation at 10Gbps and above.	RO
4/5.0.5:2	Speed Selection	These bits always read 0 indicating operation at 10Gbps.	

Figure 1. TLK3134 Datasheet Clause 45 Mode Register Table Format

Clause 22 Mode (ST = 1):

Registers accessible through the Clause 22 protocol appear in the datasheet and are discussed in a table similar to the following table which is the table for Register 0x11.

Clause 22 register tables are distinguished from the Clause 45 register tables by not containing two features found in the Clause 45 register tables. The first indication that this register responds to Clause 22 protocol is the 2 digit register address. The Clause 22 register address is only 2 digits where as the Clause 45 register address is 4 digits. The second feature is the missing "4/5" in the "Bit(s)" column. This mode does not care about a "04" or "05" device address because that is only defined for Clause 45 Mode.

The information in the "Bit(s)" field of the table is broken up by decimal points according to the following convention.

<Register Address in DEC>.<Bits in DEC with MSB:LSB>

The Default Value is the HEX value of the register following a Reset or power cycle. The binary value of the HEX Default Value for Register 0x11 is 0011010110010000 (bit 15: bit 0).

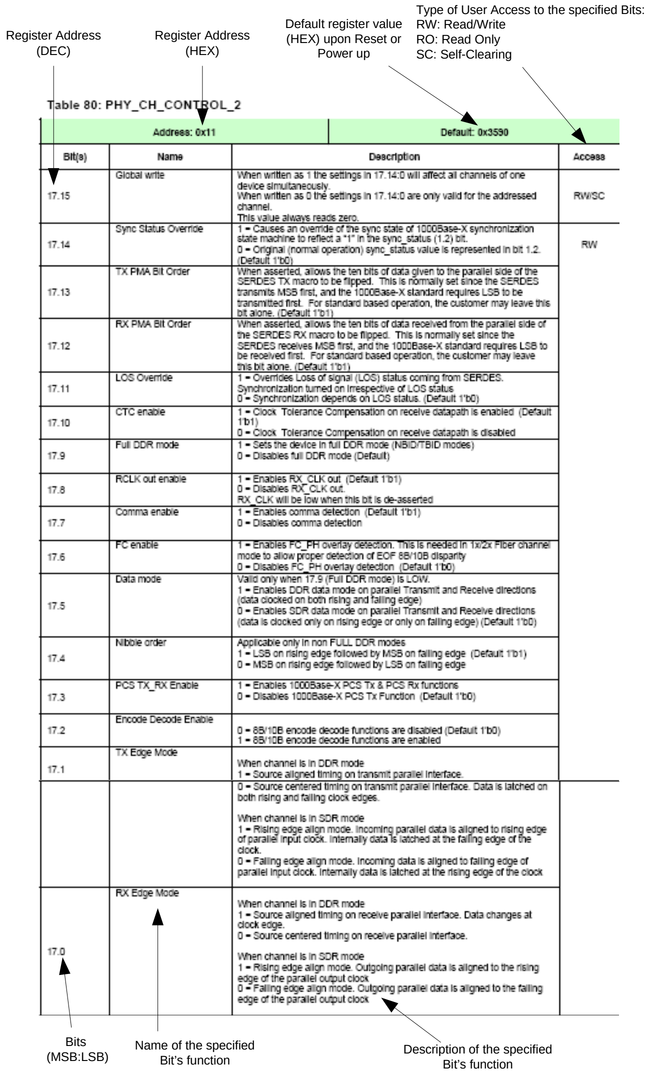

Register Address (DEC)
Register Address (HEX)
Default register value (HEX) upon Reset or Power up
Type of User Access to the specified Bits:
RW: Read/Write
RO: Read Only
SC: Self-Clearing
Bits (MSB:LSB)
Name of the specified Bit's function
Description of the specified Bit's function

Table 80: PHY_CH_CONTROL_2

Address: 0x11		Default: 0x3590	
Bit(s)	Name	Description	Access
17.15	Global write	When written as 1 the settings in 17.14:0 will affect all channels of one device simultaneously. When written as 0 the settings in 17.14:0 are only valid for the addressed channel. This value always reads zero.	RW/SC
17.14	Sync Status Override	1 = Causes an override of the sync state of 1000Base-X synchronization state machine to reflect a "1" in the sync_status (1.2) bit. 0 = Original (normal operation) sync_status value is represented in bit 1.2. (Default 1'b0)	RW
17.13	TX PMA Bit Order	When asserted, allows the ten bits of data given to the parallel side of the SERDES TX macro to be flipped. This is normally set since the SERDES transmits MSB first, and the 1000Base-X standard requires LSB to be transmitted first. For standard based operation, the customer may leave this bit alone. (Default 1'b1)	
17.12	RX PMA Bit Order	When asserted, allows the ten bits of data received from the parallel side of the SERDES RX macro to be flipped. This is normally set since the SERDES receives MSB first, and the 1000Base-X standard requires LSB to be received first. For standard based operation, the customer may leave this bit alone. (Default 1'b1)	
17.11	LOS Override	1 = Overrides Loss of signal (LOS) status coming from SERDES. Synchronization turned on irrespective of LOS status 0 = Synchronization depends on LOS status. (Default 1'b0)	
17.10	CTC enable	1 = Clock Tolerance Compensation on receive datapath is enabled (Default 1'b1) 0 = Clock Tolerance Compensation on receive datapath is disabled	
17.9	Full DDR mode	1 = Sets the device in full DDR mode (NBID/TBID modes) 0 = Disables full DDR mode (Default)	
17.8	RCLK out enable	1 = Enables RX_CLK out (Default 1'b1) 0 = Disables RX_CLK out. RX_CLK will be low when this bit is de-asserted	
17.7	Comma enable	1 = Enables comma detection (Default 1'b1) 0 = Disables comma detection	
17.6	FC enable	1 = Enables FC_PH overlay detection. This is needed in 1x/2x Fiber channel mode to allow proper detection of EOF 8B/10B disparity 0 = Disables FC_PH overlay detection (Default 1'b0)	
17.5	Data mode	Valid only when 17.9 (Full DDR mode) is LOW. 1 = Enables DDR data mode on parallel Transmit and Receive directions (data clocked on both rising and falling edge) 0 = Enables SDR data mode on parallel Transmit and Receive directions (data is clocked only on rising edge or only on falling edge) (Default 1'b0)	
17.4	Nibble order	Applicable only in non FULL DDR modes 1 = LSB on rising edge followed by MSB on falling edge (Default 1'b1) 0 = MSB on rising edge followed by LSB on falling edge	
17.3	PCS TX_RX Enable	1 = Enables 1000Base-X PCS Tx & PCS Rx functions 0 = Disables 1000Base-X PCS Tx Function (Default 1'b0)	
17.2	Encode Decode Enable	0 = 8B/10B encode decode functions are disabled (Default 1'b0) 1 = 8B/10B encode decode functions are enabled	
17.1	TX Edge Mode	When channel is in DDR mode 1 = Source aligned timing on transmit parallel interface.	
		0 = Source centered timing on transmit parallel interface. Data is latched on both rising and falling clock edges. When channel is in SDR mode 1 = Rising edge align mode. Incoming parallel data is aligned to rising edge of parallel input clock. Internally data is latched at the falling edge of the clock. 0 = Falling edge align mode. Incoming data is aligned to falling edge of parallel input clock. Internally data is latched at the rising edge of the clock	
17.0	RX Edge Mode	When channel is in DDR mode 1 = Source aligned timing on receive parallel interface. Data changes at clock edge. 0 = Source centered timing on receive parallel interface. When channel is in SDR mode 1 = Rising edge align mode. Outgoing parallel data is aligned to the rising edge of the parallel output clock 0 = Falling edge align mode. Outgoing parallel data is aligned to the falling edge of the parallel output clock	

Figure 2. TLK3134 Datasheet Clause 22 Mode Register Table Format

Clause 45 and Clause 22 Mode:

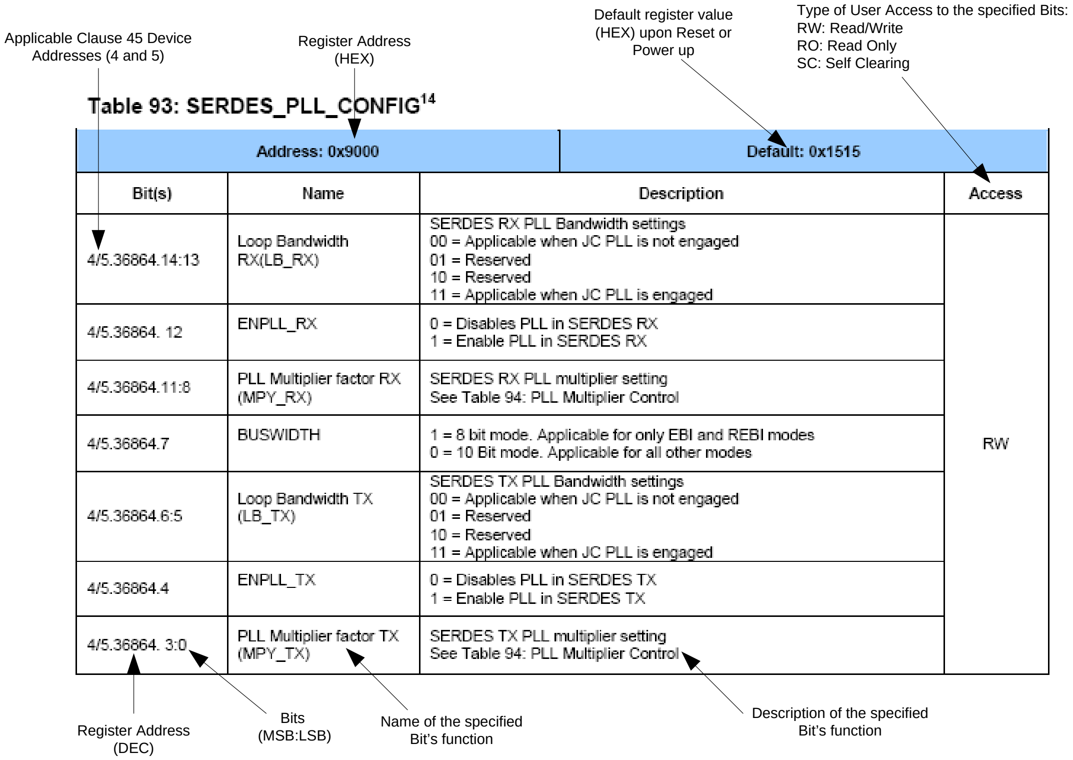
Registers accessible through both the Clause 45 and the Clause 22 protocol appear in the datasheet and are discussed in a table similar to the following table which is the table for Register 0x9000. This register is directly accessible if in Clause 45 Mode (ST = 0) and indirectly accessible if in Clause 22 Mode (ST = 1).

The first indication that this register responds to Clause 45 and Clause 22 protocol is the 4 digit register address begins with the HEX value "9." The second feature is the "4/5" in the "Bit(s)" column. This indicates that the information contained in the register is valid for both the devices addressed as "04" or "05" when the device is in Clause 45 mode, and ignored when the device is in Clause 22 mode.

The information in the "Bit(s)" field of the table is broken up by decimal points according to the following convention.

<Clause 45 device address (04 and 05 if "4/5")>.<Register Address in DEC>.<Bits in DEC with MSB:LSB>

The Default Value is the HEX value of the register following a Reset or power cycle. The binary value of the HEX Default Value for Register 0x9000 is 0001010100010101 (bit 15: bit 0).



Applicable Clause 45 Device Addresses (4 and 5)
Register Address (HEX)
Default register value (HEX) upon Reset or Power up
Type of User Access to the specified Bits:
RW: Read/Write
RO: Read Only
SC: Self Clearing
Register Address (DEC)
Bits (MSB:LSB)
Name of the specified Bit's function
Description of the specified Bit's function

Table 93: SERDES_PLL_CONFIG[14]

Address: 0x9000		Default: 0x1515	
Bit(s)	Name	Description	Access
4/5.36864.14:13	Loop Bandwidth RX(LB_RX)	SERDES RX PLL Bandwidth settings 00 = Applicable when JC PLL is not engaged 01 = Reserved 10 = Reserved 11 = Applicable when JC PLL is engaged	RW
4/5.36864. 12	ENPLL_RX	0 = Disables PLL in SERDES RX 1 = Enable PLL in SERDES RX	
4/5.36864.11:8	PLL Multiplier factor RX (MPY_RX)	SERDES RX PLL multiplier setting See Table 94: PLL Multiplier Control	
4/5.36864.7	BUSWIDTH	1 = 8 bit mode. Applicable for only EBI and REBI modes 0 = 10 Bit mode. Applicable for all other modes	
4/5.36864.6:5	Loop Bandwidth TX (LB_TX)	SERDES TX PLL Bandwidth settings 00 = Applicable when JC PLL is not engaged 01 = Reserved 10 = Reserved 11 = Applicable when JC PLL is engaged	
4/5.36864.4	ENPLL_TX	0 = Disables PLL in SERDES TX 1 = Enable PLL in SERDES TX	
4/5.36864. 3:0	PLL Multiplier factor TX (MPY_TX)	SERDES TX PLL multiplier setting See Table 94: PLL Multiplier Control	

Figure 3. TLK3134 Datasheet Clause 45 and Clause 22 Mode Register Table Format

Clause 22 Indirect Addressing:

The third register group which can be addressed through both clause 45 and clause 22 is implemented in vendor specific register space starting with the address of 16’h9000 (0x9000) and onwards. This address is composed of several parts which are:

<Number of significant binary bits counted from the LSB or Bit 0>’<h=HEX><Address in HEX>

This register space can be accessed directly using Clause 45 addressing method, but due to the Clause 22 register space limitations, and indirect addressing method is implemented so that this register space can be accessed through Clause 22 as well.

In order to access the register space 16’h9000 (0x9000) and onwards in Clause 22 mode, an “Address Control Register” Reg 30, 5’h1E (0x1E) and a “Content Register” Reg 31, 5’h1F (0x1F) have been added to the device. The register address of the register you would like to read/write should be written to the Address Control Register “0x1E.” If you are writing to this register, next write the value you would like to write to the register you just wrote to address “0x1E” to register “0x1F.” If you are reading from this register, simply read the register contents of register “0x1F” which has been set to the contents of the register you just wrote to address “0x1E.”

For example, if you are in Clause 22 mode and would like to read the value of register 0x9000, the following MDIO read/write transaction should occur.

```
WRITE(1E,9000)    //This loads the contents of register 0x9000 into Register 0x1F
READ(1F)          //This displays the contents of register 0x1F which are the same as Register 0x9000
```

The Default value of “1515” should be returned unless it has been changed since the last reset or power cycle.

Now if you would like to change the value of register 0x9000 and replace it with a value of “1510,” the following register read/write transactions should occur.

```
WRITE(1E,9000)    //This loads the contents of register 0x9000 into Register 0x1F
WRITE(1F,1510)    //This writes 1510 to the contents of Register 0x1F which are then transferred by the device to Register 0x9000
```

Now if you would like to read the contents of register 0x9000, the following register read/write transactions should occur.

```
WRITE(1E,9000)    //This loads the contents of register 0x9000 into Register 0x1F
READ(1F)          //This displays the contents of register 0x1F which are the same as Register 0x9000
```

The value 1510 should be returned instead of the previously read default value of 1515.

Hardware and Software Configuration:

MDIO Evaluation Board:

-Place Jumpers on all pins of Header Block U12 which is located next to the 20 pin Ribbon Cable Port U8.

-Place Jumpers on both pins of Header Block U10 located next to the Serial Port Connector.

-Ensure there are NO Jumpers on Header Block U9 located next to the Serial Port Connector.

-Place a Jumper on J2 located next to the Oscillator.

-Place a Jumper on J1 located next to the Banana Jacks.

-Ensure there are NO Jumpers on Header U7 located between the Regulators.

-Supply the board with 7V through the VIN (U2) and GND (U3) Banana Jacks.

The following figure contains a drawing of the MDIO Evaluation Board in its Default Jumper Configuration:

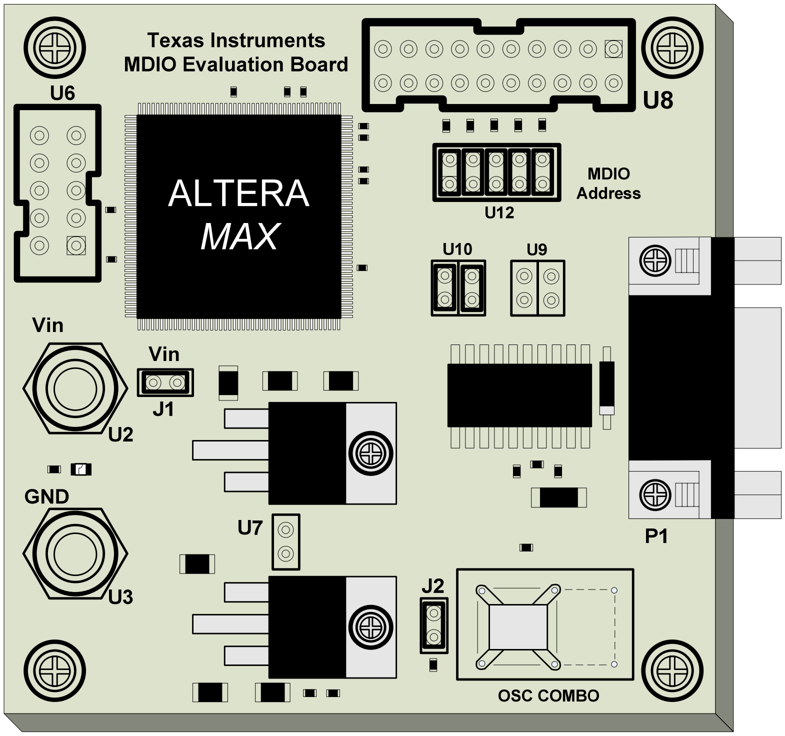
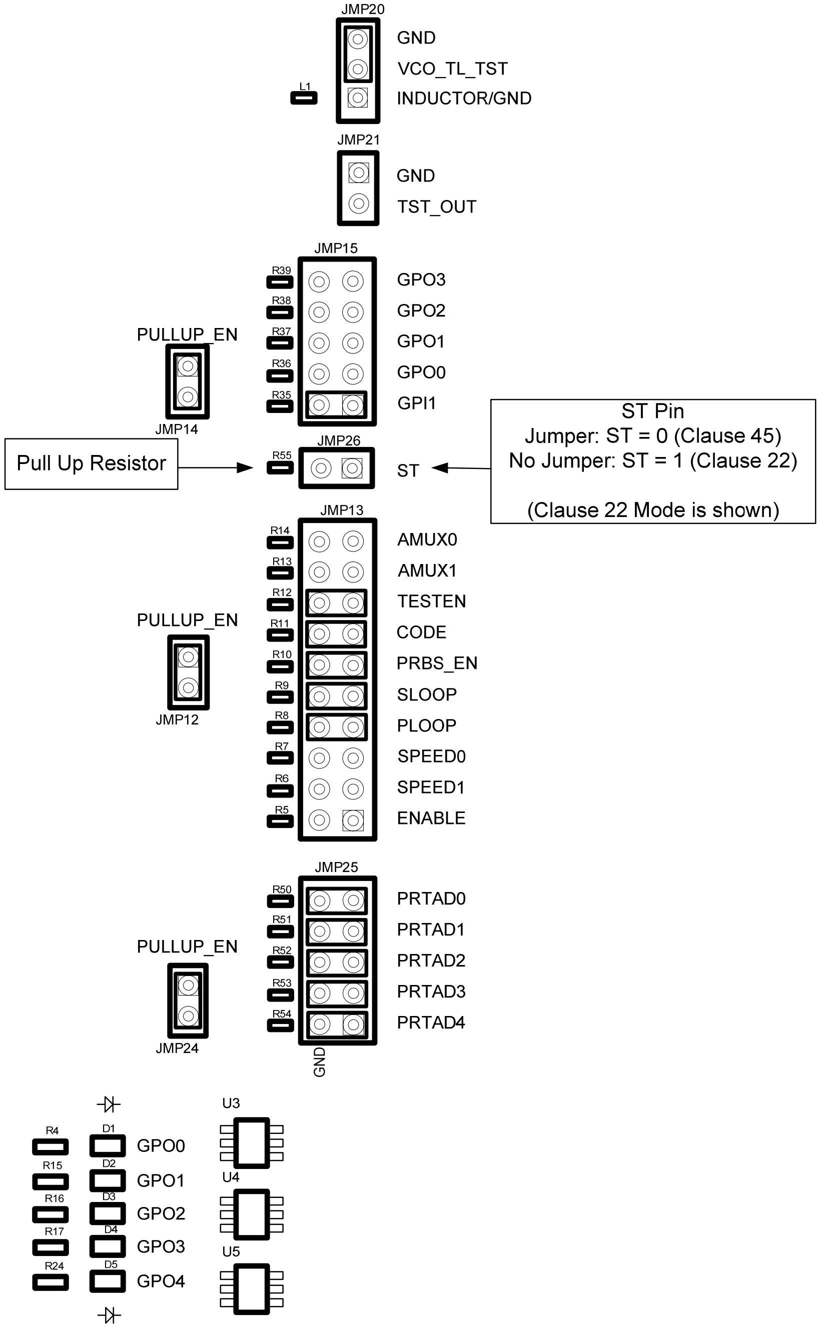


Figure 4. MDIO Evaluation Board Jumper Setting Diagram

TLK3134 Evaluation Board:

-Place Jumpers on all PRTAD(4:0) pins of Header Block JMP25.

-If in Clause 45 Mode, Place a Jumper on JMP26 (ST = 0). If in Clause 22 Mode, Ensure no Jumper is on JMP26 (ST = 1)



Pull Up Resistor

ST Pin
Jumper: ST = 0 (Clause 45)
No Jumper: ST = 1 (Clause 22)

(Clause 22 Mode is shown)

Figure 5. TLK3134 Evaluation Board Control Jumper Setting Diagram

-Make sure the VDDM Power Plane is set to, and powered with, 2.5V by placing the Jumper on JMP43 between the VDDM and 2P5V Pins.

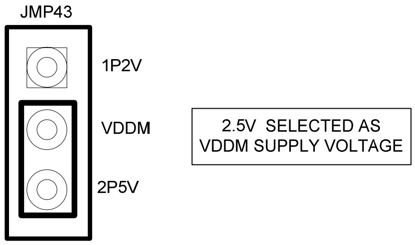


Figure 6. TLK3134 Evaluation Board VDDM Power Jumper Setting

-If the Voltage Monitor LED circuit is enabled with a Jumper between the VDDM_LED and MONITOR pins of JMP91, VDDM_LED and ENABLE pins of JMP92, and the VDDM_LED and MONITOR pins of JMP93, then the VDDM 2P5V LED should be on. If this circuit is enabled and the VDDM 2P5V LED is not on, then verify that you have 2.5V on the VDDM Power Plane which can be monitored with JMP9 located near the bottom left corner of the TLK3134 Device itself.

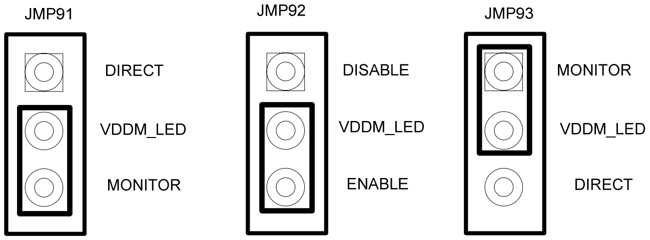
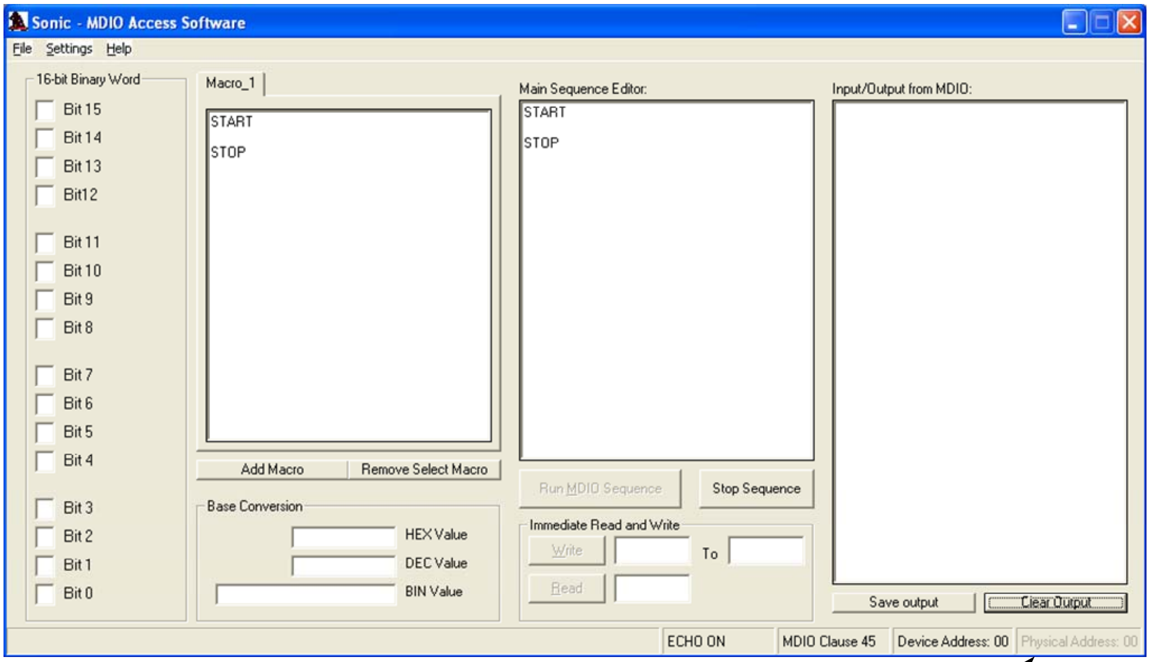


Figure 7. TLK3134 Evaluation Board VDDM Power LED Jumper Settings

Sonic MDIO Software:

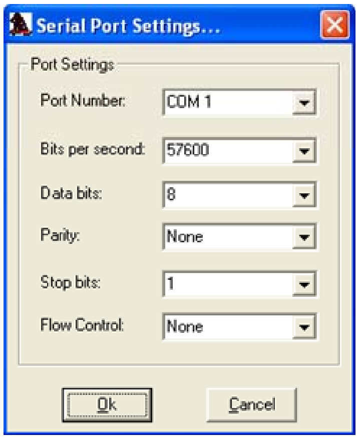
-Running the Sonic Software will generate a window with the Physical Address field in the bottom right corner of the window appearing in Grey and non-responsive to controls as seen in the following figure:



COM Port Connection is
NOT established

Figure 8. Sonic MDIO Access Software Initial Window

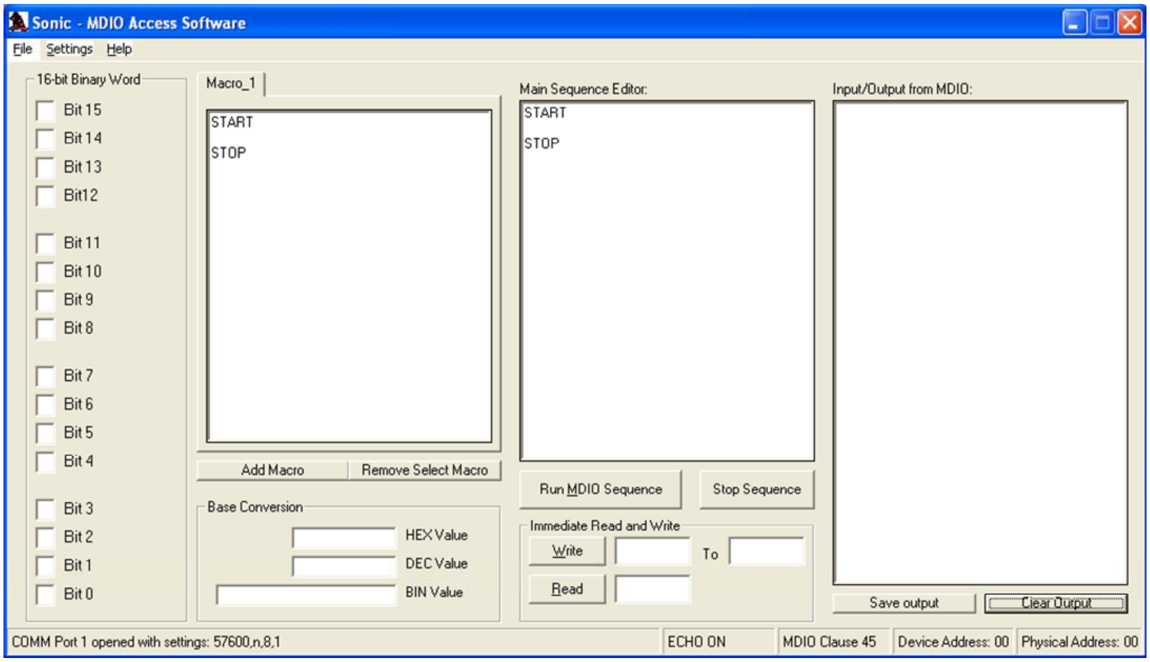
-Selecting the "Serial Port Settings…" under the "Settings" menu will bring up the following window. The Default settings should work and no changes should be needed unless your system is using multiple COM ports in which case the correct port should be selected. Simply click "OK" to enable the software and COM Port connection:



Port Number: COM 1
Bits per second: 57600
Data bits: 8
Parity: None
Stop bits: 1
Flow Control: None

Figure 9. Sonic MDIO Access Software "Serial Port Settings" Window

-The Physical Address box at the bottom right corner of the Sonic window should no longer be grey as seen in the following figure:



START

STOP

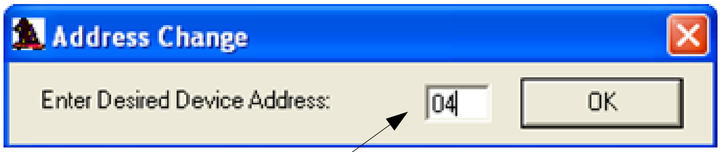
COMM Port 1 opened with settings: 57600,n,8,1 | ECHO ON | MDIO Clause 45 | Device Address: 00 | Physical Address: 00

COM Port Connection has been established

The Physical Address field is now responsive

Figure 10. Sonic MDIO Access Software Window after COM Port configuration

-Clicking on the “Device Address field in the bottom right of the window will bring up the following window where you should enter “04” for the Device Address which is needed when in Clause 45 Mode as seen in the following figure:



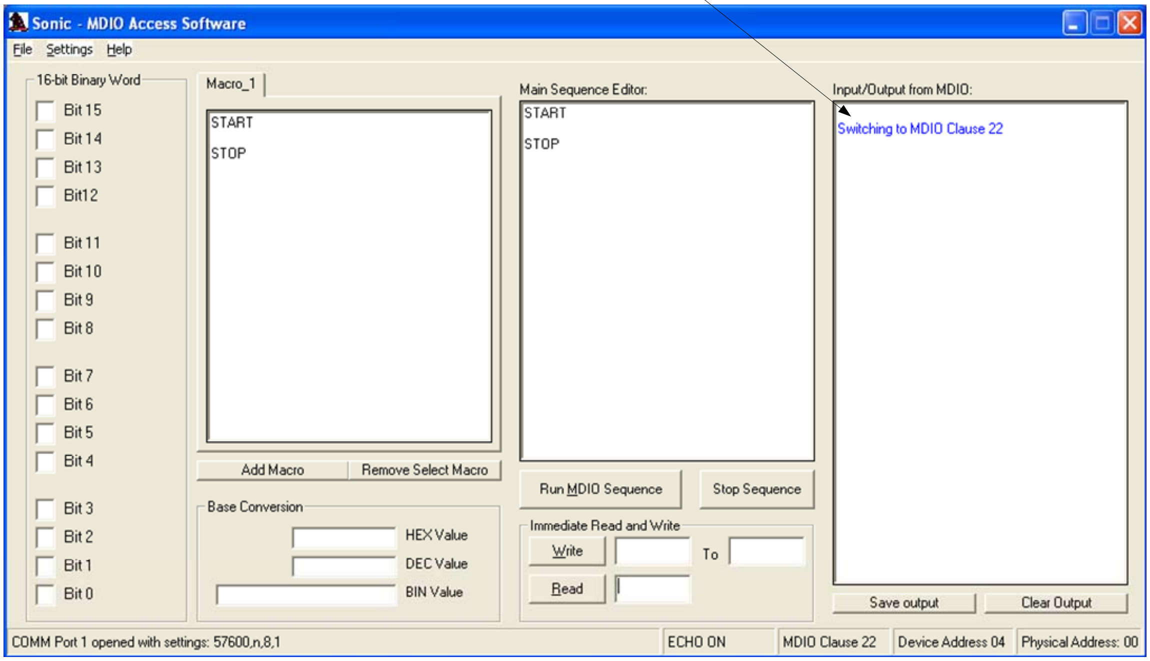
Enter Desired Device Address: 04

Enter “04”

Figure 11. Sonic MDIO Access Software “Device Address” Window

-Clicking on the “MDIO Clause” field will toggle between the Clause 45 and the Clause 22 Modes and create an entry in the log file as seen in the following figure. This should be done upon initial startup of the program to ensure the correct mode is selected and this is used primarily with the “Immediate Read and Write” functionality of the software.

Switching from Clause 45 to Clause 22 Mode
created the following Log File Entry:



```
START
STOP
```

```
START
STOP
```

```
Switching to MDIO Clause 22
```

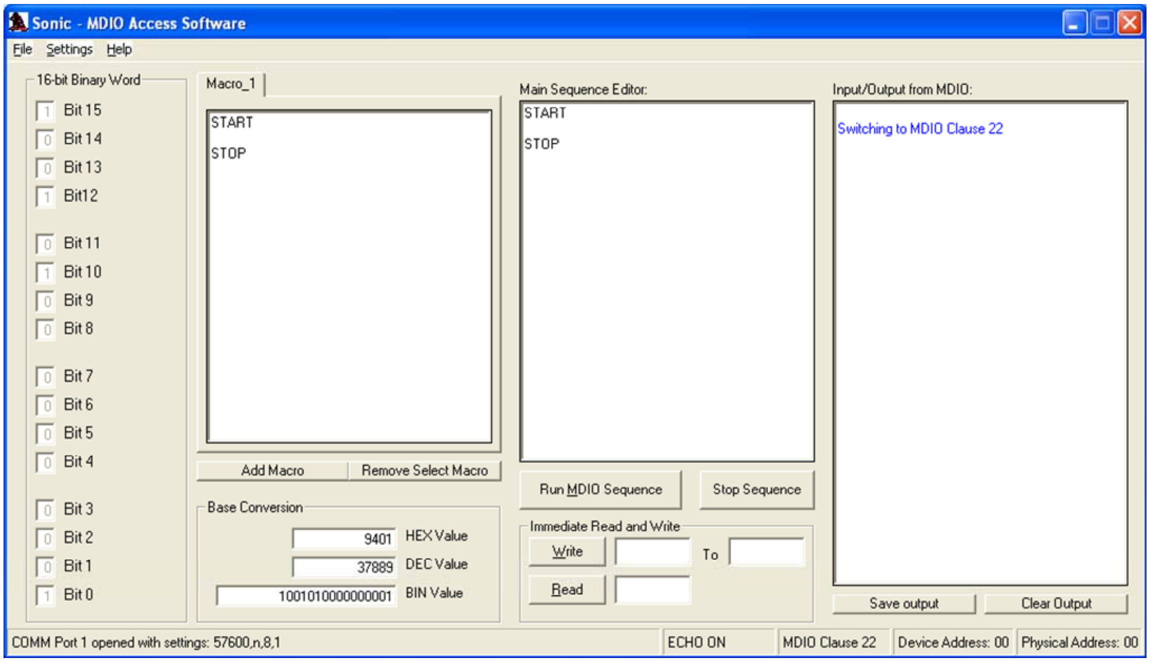
COMM Port 1 opened with settings: 57600,n,8,1 | ECHO ON | MDIO Clause 22 | Device Address 04 | Physical Address: 00

Clicking on the MDIO Clause Field will toggle
between Clause 45 and Clause 22 Modes

Figure 12. Sonic MDIO Access Software “MDIO Clause Toggle Button”

NOTE: When changing modes, double check the TLK3134 EVM’s ST pin to ensure it is properly set for the desired MDIO Clause. (Clause 45: ST = 0, Clause 22: ST = 1)

-When in Clause 22 mode, the registers for the 4 channels of the TLK3134 device are accessed on a per channel basis through the Physical Address field. The global registers only need to be set once however since they apply to all channels and device settings. When configuring a channel’s register settings, some settings can be applied to all 4 channels simultaneously by writing a “1” to bit 15 of that register (if applicable). Refer to the TLK3134 Datasheet’s 1G Programmers Reference register descriptions for more information on the registers. In order to access the channels individually simply set the “Physical Address” field in the SONIC software with the appropriate channel address.



1	Bit 15
0	Bit 14
0	Bit 13
1	Bit12
0	Bit 11
1	Bit 10
0	Bit 9
0	Bit 8
0	Bit 7
0	Bit 6
0	Bit 5
0	Bit 4
0	Bit 3
0	Bit 2
0	Bit 1
1	Bit 0

Macro_1:

```
START
STOP
```

Main Sequence Editor:

```
START
STOP
```

Input/Output from MDIO:

```
Switching to MDIO Clause 22
```

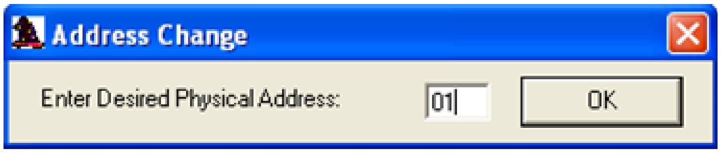
9401	HEX Value
37889	DEC Value
1001010000000001	BIN Value

COMM Port 1 opened with settings: 57600,n,8,1 | ECHO ON | MDIO Clause 22 | Device Address: 00 | Physical Address: 00

The Physical Address field is used to access the registers for the individual channels when in Clause 22 mode:

Figure 13. Sonic MDIO Access Software “Clause 22 Mode Physical Address Setting ”

-To change the Physical Address and access the different channels of the TLK3134 double click the “Physical Address” box in the SONIC software window to bring up the following Address Change Control Window. Then set the appropriate channel address based upon the following figure.



Enter Desired Physical Address: 01

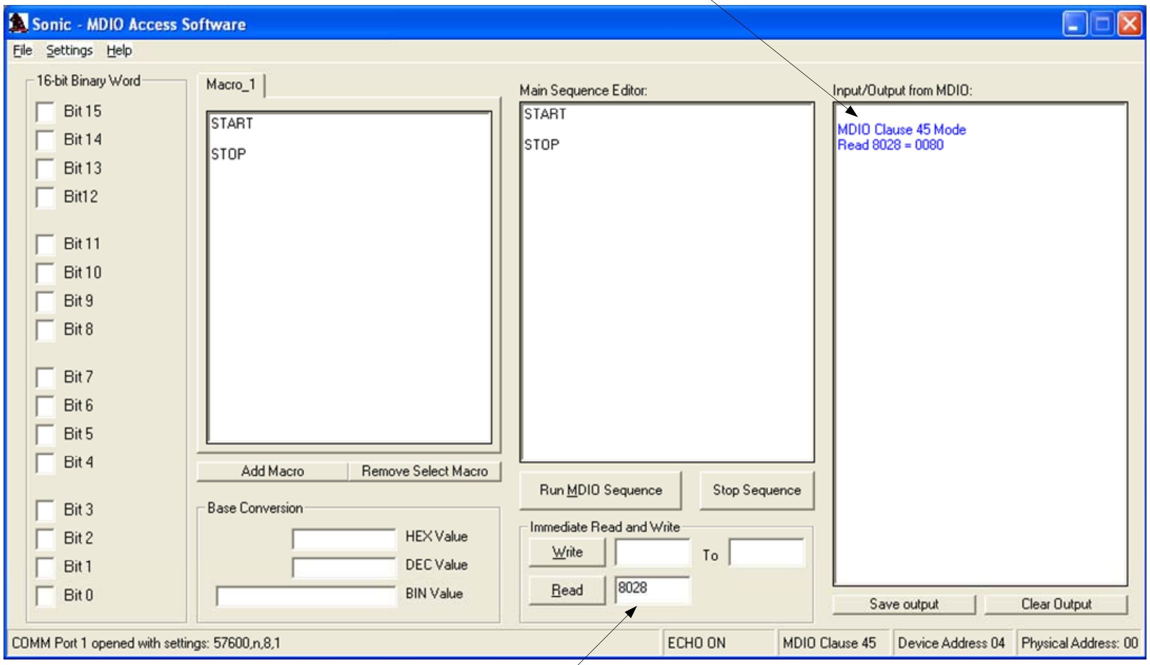
For CH0 set to: 00
For CH1 set to: 01
For CH2 set to: 10
For CH3 set to: 11

Figure 14. Sonic MDIO Access Software “Clause 22 Mode Physical Address Settings for Individual Channels of the TLK3134 Device”

-If the Serial Port Cable and the Ribbon cable are not already connected to the PC and TLK3134 EVM, make that connection at this time.

-Press the Reset button on the TLK3134 EVM to enable the default Register Values. If the TLK3134 is set up for Clause 45 communication, typing “8028” in the Read field, followed by a click on the “Read” button, should create the following entry in the log file if both the TLK3134 and MDIO Evaluation Boards and Sonic software are all configured correctly.

If the system is functioning correctly, the value of “0080” should be returned as the contents of register 0x8028



Macro_1:

```
START

STOP
```

Main Sequence Editor:

```
START

STOP
```

Input/Output from MDIO:

```
MDIO Clause 45 Mode
Read 8028 = 0080
```

Read: 8028

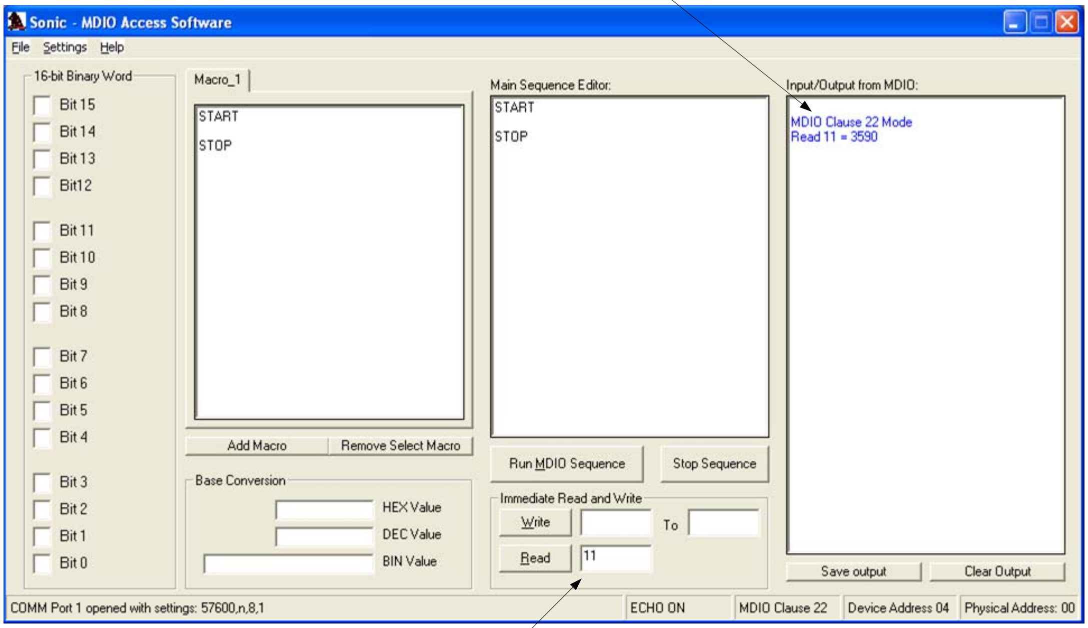
```
COMM Port 1 opened with settings: 57600,n,8,1 | ECHO ON | MDIO Clause 45 | Device Address 04 | Physical Address: 00
```

Enter the Clause 45 Four-Digit HEX address of the TLK3134 Register you would like to read and click “Read”

Figure 15. Sonic MDIO Access Software – Immediate Read of a Clause 45 Register

-Press the Reset button on the TLK3134 EVM to enable the default Register Values. If the TLK3134 is set up for Clause 22 communication, typing “11” in the Read field, followed by a click on the “Read” button should create the following entry in the log file if the TLK3134 and MDIO Evaluation Boards and Sonic software are all configured correctly.

If the system is functioning correctly, the value of “3590” should be returned as the contents of register 0x11



Macro_1:

```
START

STOP
```

Main Sequence Editor:

```
START

STOP
```

Input/Output from MDIO:

```
MDIO Clause 22 Mode
Read 11 = 3590
```

Read: 11

COMM Port 1 opened with settings: 57600,n,8,1 | ECHO ON | MDIO Clause 22 | Device Address 04 | Physical Address: 00

Enter the Clause 22 Two-Digit HEX address of the TLK3134 Register you would like to read and click “Read”

Figure 16. Sonic MDIO Access Software – Immediate Read of a Clause 22 Register

*************SONIC SOFTWARE ERRATA************

Sonic supports both Clause 45 and Clause 22 MDIO Access Protocols.

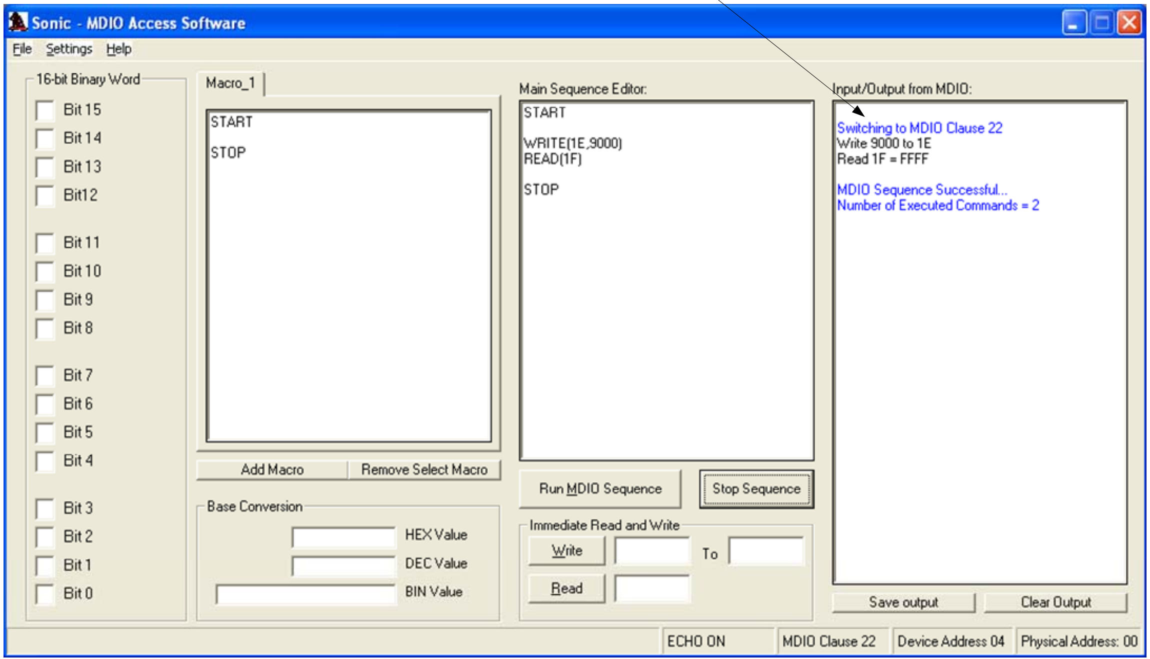
When using the “Immediate Read and Write” portion of the program the “MDIO Clause” toggle button in the bottom left corner of the window should be used to select the desired Clause.

If the “Main Sequence Editor:” portion of the program is used, Sonic ignores the Clause setting at the bottom of the window and changes to the default Clause 45 Mode of operation. However, the MDIO Clause Button at the bottom of the window remains unchanged despite the fact that the supported mode has changed.

For example, if the MDIO Clause button is manually set to Clause 22, and the “Run MDIO Sequence” button is pressed, the Program will change itself to Clause 45 mode and the MDIO Clause button will still show “MDIO Clause 22.” Subsequent Clause 22 Register reads and writes will result in “FFFF” errors until the mode is reset.

If you desire to run a sequence in the “Main Sequence Editor:” in Clause 22 Mode, the command “CLAUSE 22” should be placed before all register read and write commands.

Running the script in the Main Sequence Editor resulted in an Error of “FFFF” because the line “CLAUSE 22” was not included at the beginning of the sequence.



Macro_1:

```
START

STOP
```

Main Sequence Editor:

```
START

WRITE(1E,9000)
READ(1F)

STOP
```

Input/Output from MDIO:

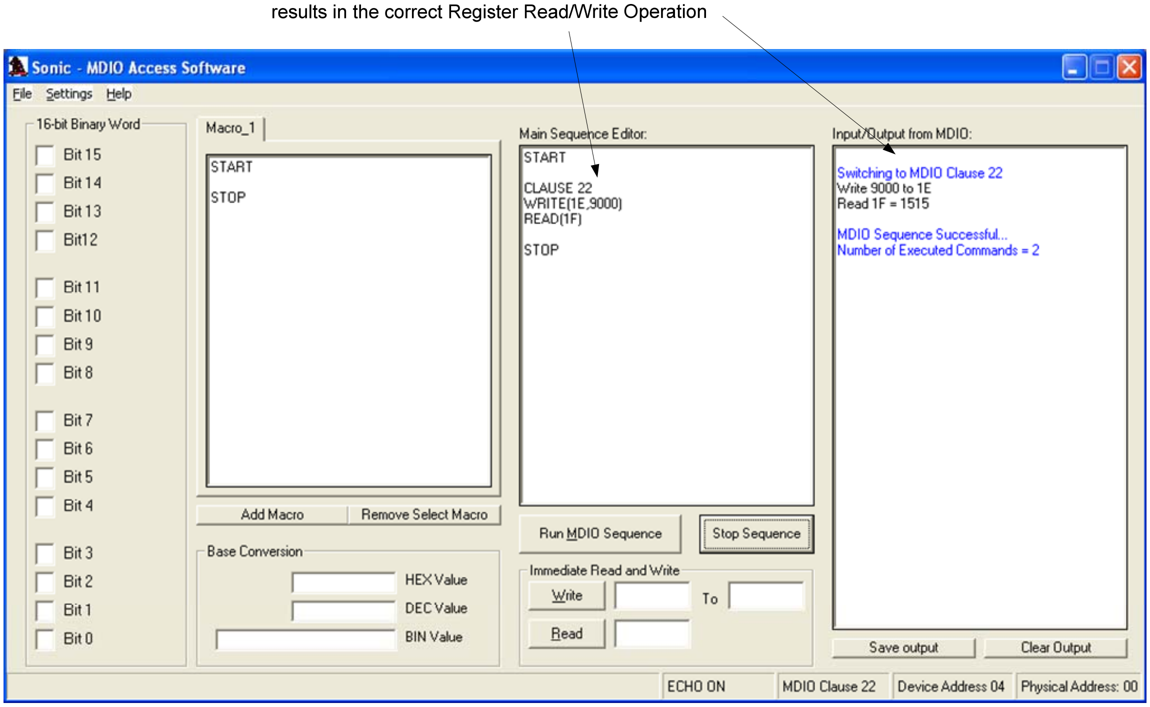
```
Switching to MDIO Clause 22
Write 9000 to 1E
Read 1F = FFFF

MDIO Sequence Successful...
Number of Executed Commands = 2
```

ECHO ON | MDIO Clause 22 | Device Address 04 | Physical Address: 00

The MDIO Clause Toggle Button is set to Clause 22 Mode even though the Main Sequence Editor changes to Clause 45 mode.

Figure 17. Sonic MDIO Access Software – Incorrect Clause 22 “Indirect Access”


Adding the command “CLAUSE 22” to the sequence results in the correct Register Read/Write Operation

Macro_1:

```
START

STOP
```

Main Sequence Editor:

```
START

CLAUSE 22
WRITE(1E,9000)
READ(1F)

STOP
```

Input/Output from MDIO:

```
Switching to MDIO Clause 22
Write 9000 to 1E
Read 1F = 1515

MDIO Sequence Successful...
Number of Executed Commands = 2
```

Figure 18. Sonic MDIO Access Software – Correct Clause 22 “Indirect Access”

Sonic MDIO Software Example:

The following will be an example of how to use the Sonic MDIO software to implement the "Gigabit Ethernet Mode (RGMII) Test Setup Configuration" found in the TLK3134 Datasheet and EVM User's Guide.

The device reset requirements and setup procedure to configure the TLK3134 for Gigabit Ethernet Mode (RGMII) is as follows:

****NOTE: All global registers must be accessed indirectly through Clause 22.**

REFCLK frequency = 125MHz, Serdes Data Rate = Half Rate, Mode = Transceiver, Edge Mode = Source Centered Mode, RX_CLK[n] out = TXBLK[n], Jitter Cleaner PLL Multiplier Ratio = 1X or Off.

The Device Pin Settings for the TLK3134 EVM are as follows and allow for the maximum software configurability.
(No Jumper = High, Jumper = Low)

Ensure: ST = High
CODE = Low
PLOOP = Low
SLOOP = Low
SPEED [1:0] = Both pins are High
ENABLE = High
PRBS_EN = Low

Reset the Device by pressing the RESET button or write 1'b1 to 0.15. The (1'b1) is read "One bit of binary value 1."

Generically this is interpreted:
<Number of binary bits to change>'<Data Type: h=HEX, b=BIN><Value of those bits in the specified HEX or BIN Format>

The (0.15) is read "Bit 15 of Register Address (DEC) 0."

Generically this is interpreted:
<Register Address in Decimal>.<Bit(s) MSB:LSB>

A Software RESET would be implemented using the Sonic MDIO software as follows:

```
WRITE(00,8000)          //This is binary 1000000000000000 where bit 15 =1
```

The value of HEX "8" gives us a binary 1000 for bits 15:12 placing a BIN "1" in the position we care about in order to create a Device Reset. Writing every other bit with a "0" does not do anything for us except make it easy to determine HEX values for the bits 11:0 since a 1 in bit 15 will create a device reset and all registers will be set to their default values irregardless of the values being written to bits 14:0. However, a Software RESET could also be implemented using the mask function of the Sonic MDIO Software as follows:

```
WRITE(00,FFFF,8000)  //This writes binary 1111111111111111 to register 00 but only allows bit 15
                     //to be changed since it is the only bit that has an associated "1" in the Mask
                     //setting of HEX 8000 or BIN 1000000000000000
```

Generically this write command is:

WRITE(<HEX Register Address>,<HEX Data to write to the register>,<Optional bit mask to set only the register bits that have a 1 in the mask field>)

Configure the clock:

If using the JCPLL (JCPLL 1X)

Select REFCLK Input (Default = Differential)

If the Single Ended REFCLK is used – write 2’b01 to 4/5.37120.15:14
This is read “write 2 bits of binary value 01 to register 0x9100 bits 15:14”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)        //Indirectly address register 0x9100
WRITE(1F,4000,C000)   //Write a “01” to bits 15:14 through “4” and mask
                      //off the rest of the bits through the “Mask” of
                      //C000 where only bits 15:14 are set to 1 in “C”
```

If the Differential REFCLK is used – write 2’b00 to 4/5.37120.15:14 OR don’t write anything since the register’s default value implements the Differential REFCLK.
This is read “write 2 bits of binary value 00 to register 0x9100 bits 15:14”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)        //Indirectly address register 0x9100
WRITE(1F,0000,C000)   //Write a “00” to bits 15:14 through “0” and mask
                      //off the rest of the bits through the “Mask” of
                      //C000 where only bits 15:14 are set to 1 in “C”
```

Write 2’b11 to 4/5.37120.13:12 to select differential REFCLKP/N as RXBYTECLK
This is read “write 2 bits of binary value 11 to register 0x9100 bits 13:12”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)        //Indirectly address register 0x9100
WRITE(1F,3000,3000)   //Write a “11” to bits 13:12 through “3” and mask
                      //off the rest of the bits through the “Mask” of
                      //3000 where only bits 13:12 are set to 1 in “3”
```

Write 4’b0000 to 4/5.37120.11:8 to select jitter cleaned clock for SERDES TX/RX
This is read “write 4 bits of binary value 0000 to register 0x9100 bits 11:8”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)        //Indirectly address register 0x9100
WRITE(1F,0000,0F00)   //Write a “0000” to bits 11:8 through “0” and mask
                      //off the rest of the bits through the “Mask” of
                      //0F00 where only bits 11:8 are set to 1 in “F”
```

Write 2’b11 to 4/5.37120.7:6 to select differential REFCLKP/N as Delay Stopwatch clock input
This is read “write 2 bits of binary value 11 to register 0x9100 bits 7:6”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)        //Indirectly address register 0x9100
WRITE(1F,00C0,00C0)   //Write a “11” to bits 7:6 through “C” and mask
                      //off the rest of the bits through the “Mask” of
                      //00C0 where only bits 7:6 are set to 1 in “C”
```

Write 2’b00 to 4/5.37120.5:4 to select jitter cleaned clock for HSTL VTP 2x

This is read “write 2 bits of binary value 00 to register 0x9100 bits 5:4”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)          //Indirectly address register 0x9100
WRITE(1F,0000,0030)     //Write a “00” to bits 5:4 through “0” and mask
                        //off the rest of the bits through the “Mask” of
                        //0030 where only bits 5:4 are set to 1 in “3”
```

****NOTE: Up until this point, Register 0x9100 has been used and individual bits of that register have been changed according to their function. However, for faster execution and cleaner code, all of the bits set to this point can be combined and implemented in a single transaction such as:**

If using a Single Ended REFCLK:

```
WRITE(1E,9100)          //Indirectly address register 0x9100
WRITE(1F,70C0,FFF0)     //Write a “011100001100” to bits 15:4 and mask
                        //off the rest of the bits 3:0
```

The default value for bits 3:0 of register 0x9100 are 0000 so in the previous case the mask can be omitted because we are re-writing the default values, but it was left in for demonstrative purposes.

If using a Differential REFCLK:

```
WRITE(1E,9100)          //Indirectly address register 0x9100
WRITE(1F,30C0,FFF0)     //Write a “001100001100” to bits 15:4 and mask
                        //off the rest of the bits 3:0
```

The default value for bits 3:0 of register 0x9100 are 0000 so in the previous case the mask can be omitted because we are re-writing the default values, but it was left in for demonstrative purposes.

Write 2’b00 to 16.10:9 to select SERDES TX clock as RX_CLK output (per channel)

This is read “write 2 bits of binary value 00 to register 0x10 bits 10:9”
In Sonic this is accomplished by the following commands:

```
WRITE(10,0000,0600)     //Write a “00” to bits 10:9 through “0” and mask
                        //off the rest of the bits through the “Mask” of
                        //0600 where only bits 10:9 are set to 1 in “6”
```

****NOTE: Register 0x10 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 16’h0081 to 4/5.37126 to set Charge Pump Control

This is read “write 16 bits of hexadecimal value 0081 to register 0x9106 bits 15:0”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9106)          //Indirectly address register 0x9106
WRITE(1F,0081)          //Write a “0081” to bits 15:0
```

****NOTE: No Mask is required because all bits of register 0x9106 are being set**

Write 16’h00A0 to 4/5.37128 to set the TXRX output divider

This is read “write 16 bits of hexadecimal value 00A1 to register 0x9108 bits 15:0”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9108)          //Indirectly address register 0x9108
WRITE(1F,00A0)          //Write a “00A0” to bits 15:0
```

****NOTE: No Mask is required because all bits of register 0x9108 are being set**

Set the Clock Divide Settings

Write 7’b1000000 to 4/5.37124.14:8 to set REF_DIV to the value of 1

This is read “write 7 bits of binary value 1000000 to register 0x9104 bits 14:8”

Write 1’b1 to 4/5.37124.15 to set REFDIV_EN to enable the reference clock divider

This is read “write 1 bit of binary value 1 to register 0x9104 bit 15”

Write 7’h18 to 4/5.37124.6:0 to set FB_DIV to the value of 24

This is read “write 7 bits of hexadecimal value 18 to register 0x9104 bits 6:0”

Write 1’b1 to 4/5.37124.7 to set FBDIV_EN to enable the feedback divider

This is read “write 1 bit of binary value 1 to register 0x9104 bit 7”

In Sonic these last four instructions can be combined and are accomplished by the following commands:

```
WRITE(1E,9104)          //Indirectly address register 0x9104
WRITE(1F,C098)          //Write a “C098” to bits 15:0
```

****NOTE: No Mask is required because all bits of register 0x9104 are being set**

Write 7’h18 to 4/5.37125.6:0 to set RXTX_DIV to the value of 24

This is read “write 7 bits of hexadecimal value 18 to register 0x9105 bits 6:0”

Write 1’b1 to 4/5.37125.7 to set OUTDIV_EN to enable the RXTX_DIV output divider

This is read “write 1 bit of binary value 1 to register 0x9105 bit 7”

In Sonic these last two instructions can be combined and are accomplished by the following commands:

```
WRITE(1E,9105)          //Indirectly address register 0x9105
WRITE(1F,0098,00FF)     //Write a “10011000” to bits 7:0 and mask
                        //off the rest of the bits through the “Mask” of
                        //00FF where only bits 7:0 are set to 1
```

Write 7’h0D to 4/5.37121.14:8 to set HSTL_DIV to the value of 13

This is read “write 7 bits of hexadecimal value 0D to register 0x9101 bits 14:8”

Write 7’h06 to 4/5.37121.6:0 to set HSTL_DIV2 to the value of 6

This is read “write 7 bits of hexadecimal value 06 to register 0x9101 bits 6:0”

In Sonic these last two instructions can be combined and are accomplished by the following commands:

```
WRITE(1E,9101)         //Indirectly address register 0x9101
WRITE(1F,0D06,7F7F)    //Write a “0001101” to bits 14:8 and “0000110” to bits 6:0
                       //and mask off the rest of the bits through the “Mask” of
                       //7F7F where only bit 15 and bit 8 are set to 0 and ignored
```

Write 2’b11 to 4/5.36864.14:13 to set RX Loop Bandwidth

This is read “write 2 bits of binary value 11 to register 0x9000 bits 14:13”

Write 2’b11 to 4/5.36864.6:5 to set TX Loop Bandwidth

This is read “write 2 bits of binary value 11 to register 0x9000 bits 6:5”

In Sonic these last two instructions can be combined and are accomplished by the following commands:

```
WRITE(1E,9000)         //Indirectly address register 0x9000
WRITE(1F,6060,6060)    //Write a “11” to bits 14:13 and “11” to bits 6:5
                       // mask off the rest of the bits through the “Mask” of
                       //6060 where only the bits 14:13 and 6:5 are
                       // set to 1 and the rest of the bits are ignored
```

Write 4’b0101 to 4/5.36864.11:8 to set the MPY RX multiplier factor to 10

This is read “write 4 bits of binary value 0101 to register 0x9000 bits 11:8”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9000)         //Indirectly address register 0x9000
WRITE(1F,0500,0F00)    //Write a “0101” 11:8 and mask off the rest of the bits
                       //through the “Mask” of 0F00 where only bits 11:8
                       // are set to 1 and the rest of the bits are ignored
```

Write 4’b0101 to 4/5.36864.3:0 to set the MPY TX multiplier factor to 10

This is read “write 4 bits of binary value 0101 to register 0x9000 bits 3:0”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9000)         //Indirectly address register 0x9000
WRITE(1F,0005,000F)    //Write a “0101” 3:0 and mask off the rest of the bits
                       //through the “Mask” of 000F where only bits 3:0
                       // are set to 1 and the rest of the bits are ignored
```

Write 16’h5555 to 4/5.36865 SERDES_RATE_CONFIG_TX_RX to set Half Rate Mode

This is read “write 16 bits of hexadecimal value 5555 to register 0x9001 bits 15:0”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9001)         //Indirectly address register 0x9001
WRITE(1F,5555)         //Write a “0101010101010101” to bits 15:0
```

****NOTE: No Mask is required because all bits of register 0x9001 are being set**

Write 3’b000 t0 4/5.37127.14:12 to set the control bits for VCO tail current to 0

This is read “write 3 bits of binary value 000 to register 0x9107 bits 14:12”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9107)          //Indirectly address register 0x9107
WRITE(1F,0000,7000)     //Write a “000” to bits 14:12 and mask off the rest of the
                        // bits through the “Mask” of 7000 where only bits 14:12
                        //are set to 1 and the rest of the bits are ignored
```

Write 1’b1 t0 4/5.37127.15 to enable the Jitter Cleaner

This is read “write 1 bit of binary value 1 to register 0x9107 bit 15”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9107)          //Indirectly address register 0x9107
WRITE(1F,8000,8000)     //Write a “1” to bit 15
                        //and mask off the rest of the bits through the “Mask” of
                        //8000 where only bit 15 is set to 1 and the rest are ignored
```

Wait 50 ms in order to allow the JCPLL to lock

Else If using the clock bypass mode (JCPLL Off):

Select REFCLK Input (Default = Differential)

If the Single Ended REFCLK is used – write 2’b01 to 4/5.37120.15:14

This is read “write 2 bits of binary value 01 to register 0x9100 bits 15:14”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)          //Indirectly address register 0x9100
WRITE(1F,4000,C000)     //Write a “01’ to bits 15:14 through “4” and mask
                        //off the rest of the bits through the “Mask” of
                        //C000 where only bits 15:14 are set to 1 in “C”
```

If the Differential REFCLK is used – write 2’b00 to 4/5.37120.15:14 OR don’t write anything since the register’s default value implements the Differential REFCLK.

This is read “write 2 bits of binary value 00 to register 0x9100 bits 15:14”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)          //Indirectly address register 0x9100
WRITE(1F,0000,C000)     //Write a “00’ to bits 15:14 through “0” and mask
                        //off the rest of the bits through the “Mask” of
                        //C000 where only bits 15:14 are set to 1 in “C”
```

Select RXBYTE_CLK (Default = Differential)

If the Single Ended REFCLK is used – write 2’b10 to 4/5.37120.13:12

This is read “write 2 bits of binary value 10 to register 0x9100 bits 13:12”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)          //Indirectly address register 0x9100
WRITE(1F,2000,3000)     //Write a “10” to bits 13:12 through “2” and mask
                        //off the rest of the bits through the “Mask” of
                        //3000 where only bits 13:12 are set to 1 in “3”
```

If the Differential REFCLK is used – write 2’b11 to 4/5.37120.13:12 OR don’t write anything since the register’s default value implements the Differential REFCLK.

This is read “write 2 bits of binary value 11 to register 0x9100 bits 13:12”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)        //Indirectly address register 0x9100
WRITE(1F,3000,3000)   //Write a “11” to bits 13:12 through “3” and mask
                      //off the rest of the bits through the “Mask” of
                      //3000 where only bits 13:12 are set to 1 in “3”
```

Select SERDES TX Reference Clock Input (Default = Differential)

If the Single Ended REFCLK is used – write 2’b10 to 4/5.37120.11:10

This is read “write 2 bits of binary value 10 to register 0x9100 bits 11:10”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)        //Indirectly address register 0x9100
WRITE(1F,0800,0C00)   //Write a “10” to bits 11:10 through “8” and mask
                      //off the rest of the bits through the “Mask” of
                      //0C00 where only bits 11:10 are set to 1 in “C”
```

If the Differential REFCLK is used – write 2’b11 to 4/5.37120.11:10 OR don’t write anything since the register’s default value implements the Differential REFCLK.

This is read “write 2 bits of binary value 11 to register 0x9100 bits 11:10”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)        //Indirectly address register 0x9100
WRITE(1F,0300,0300)   //Write a “11” to bits 11:10 through “3” and mask
                      //off the rest of the bits through the “Mask” of
                      //0300 where only bits 11:10 are set to 1 in “3”
```

Select SERDES RX Reference Clock Input (Default = Differential)

If the Single Ended REFCLK is used – write 2’b10 to 4/5.37120.9:8

This is read “write 2 bits of binary value 10 to register 0x9100 bits 9:8”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)        //Indirectly address register 0x9100
WRITE(1F,0200,0300)   //Write a “10” to bits 9:8 through “2” and mask
                      //off the rest of the bits through the “Mask” of
                      //0300 where only bits 9:8 are set to 1 in “3”
```

If the Differential REFCLK is used – write 2’b11 to 4/5.37120.9:8 OR don’t write anything since the register’s default value implements the Differential REFCLK.

This is read “write 2 bits of binary value 11 to register 0x9100 bits 9:8”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)        //Indirectly address register 0x9100
WRITE(1F,0300,0300)   //Write a “11” to bits 9:8 through “3” and mask
                      //off the rest of the bits through the “Mask” of
                      //0300 where only bits 9:8 are set to 1 in “3”
```

Select DELAY_CLK (Default = Differential)

If the Single Ended REFCLK is used – write 2’b10 to 4/5.37120.7:6
This is read “write 2 bits of binary value 10 to register 0x9100 bits 7:6”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)          //Indirectly address register 0x9100
WRITE(1F,0080,00C0)     //Write a “10” to bits 7:6 through “2” and mask
                        //off the rest of the bits through the “Mask” of
                        //00C0 where only bits 7:6 are set to 1 in “C”
```

If the Differential REFCLK is used – write 2’b11 to 4/5.37120.7:6 OR don’t write anything since the register’s default value implements the Differential REFCLK.
This is read “write 2 bits of binary value 11 to register 0x9100 bits 7:6”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)          //Indirectly address register 0x9100
WRITE(1F,00C0,00C0)     //Write a “11” to bits 7:6 through “C” and mask
                        //off the rest of the bits through the “Mask” of
                        //00C0 where only bits 7:6 are set to 1 in “C”
```

Select HSTL_2X_CLK (Default = Differential)

If the Single Ended REFCLK is used – write 2’b10 to 4/5.37120.5:4
This is read “write 2 bits of binary value 10 to register 0x9100 bits 5:4”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)          //Indirectly address register 0x9100
WRITE(1F,0020,0030)     //Write a “10” to bits 5:4 through “2” and mask
                        //off the rest of the bits through the “Mask” of
                        //0030 where only bits 5:4 are set to 1 in “3”
```

If the Differential REFCLK is used – write 2’b11 to 4/5.37120.5:4 OR don’t write anything since the register’s default value implements the Differential REFCLK.
This is read “write 2 bits of binary value 11 to register 0x9100 bits 5:4”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9100)          //Indirectly address register 0x9100
WRITE(1F,0030,0030)     //Write a “11” to bits 5:4 through “3” and mask
                        //off the rest of the bits through the “Mask” of
                        //0030 where only bits 5:4 are set to 1 in “3”
```

****NOTE: Up until this point, Register 0x9100 has been used and individual bits of that register have been changed according to their function. However, for faster execution and cleaner code, all of the bits set to this point can be combined and implemented in a single transaction such as:**

If using a Single Ended REFCLK:

```
WRITE(1E,9100)         //Indirectly address register 0x9100
WRITE(1F,6AA0,FFF0) //Write a "011010101010" to bits 15:4 and mask
                       //off the rest of the bits 3:0
```

The default value for bits 3:0 of register 0x9100 are 0000 so in the previous case the mask can be omitted because we are re-writing the default values, but it was left in for demonstrative purposes.

If using a Differential REFCLK the registers are already set due to the default conditions or you can re-write those registers:

```
WRITE(1E,9100)         //Indirectly address register 0x9100
WRITE(1F,3FF0,FFF0)  //Write a "001111111111" to bits 15:4 and mask
                       //off the rest of the bits 3:0
```

The default value for bits 3:0 of register 0x9100 are 0000 so in the previous case the mask can be omitted because we are re-writing the default values, but it was left in for demonstrative purposes.

Write 2'b00 to 16.10:9 to select SERDES TX clock as RX_CLK output (per channel)

This is read "write 2 bits of binary value 00 to register 0x10 bits 10:9"

In Sonic this is accomplished by the following commands:

```
WRITE(10,0000,0600)    //Write a "00" to bits 10:9 through "0" and mask
                       //off the rest of the bits through the "Mask" of
                       //0600 where only bits 10:9 are set to 1 in "6"
```

****NOTE: Register 0x10 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 7'h04 to 4/5.37121.6:0 to set HSTL_DIV2 to the value of 4

This is read "write 6 bits of hexadecimal value 04 to register 0x9101 bits 6:0"

In Sonic this is accomplished by the following commands:

```
WRITE(1E,9101)         //Indirectly address register 0x9101
WRITE(1F,0004,007F)    //Write a "0000100" to bits 6:0 and mask off
                       // the rest of the bits through the "Mask" off 007F
                       //where only bits 6:0 are set to 1 through "7F"
```

Write 15'h1515 to 4/5.36864.14:0 SERDES_PLL_CONFIG to set the MPY RX/TX multiplier factor to 10.

This is read "write 15 bits of hexadecimal value 1515 to register 0x9000 bits 14:0"

In Sonic this is accomplished by the following commands:

```
WRITE(1E,9000)         //Indirectly address register 0x9000
WRITE(1F,1515,7FFF)    //Write a "0010101 14:8 and "0000110" to bits 6:0
                       //and mask off the rest of the bits through the "Mask" of
                       //7FFF where only bit 15 is set to 0 and ignored
```

Write 16'h5555 to 4/5.36865 SERDES_RATE_CONFIG_TX_RX to set Half Rate Mode

This is read "write 16 bits of hexadecimal value 5555 to register 0x9001 bits 15:0"

In Sonic this is accomplished by the following commands:

```
WRITE(1E,9001)         //Indirectly address register 0x9001
WRITE(1F,5555)         //Write a "0101010101010101" to bits 15:0
```

****NOTE: No Mask is required because all bits of register 0x9001 are being set**

Configure the Mode:

Write 1’b0 to 17.0 to select RX Source Centered Mode (per channel)
This is read “write 1 bit of binary value 0 to register 0x11 bit 0”
In Sonic this is accomplished by the following commands:

```
WRITE(11,0000,0001)    //Write a “0” to bit 0 through “0” and mask
                       //off the rest of the bits through the “Mask” of
                       //0001 where only bit 0 is set to 1 in “1”
```

****NOTE: Register 0x11 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 1’b0 to 17.1 to select TX Source Centered Mode (per channel)
This is read “write 1 bit of binary value 0 to register 0x11 bit 1”
In Sonic this is accomplished by the following commands:

```
WRITE(11,0000,0002)    //Write a “0” to bit 1 through “0” and mask
                       //off the rest of the bits through the “Mask” of
                       //0002 where only bit 1 is set to 1 in “2”
```

****NOTE: Register 0x11 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 1’b1 to 17.2 to enable 8B/10B encode and decode functions (per channel)
This is read “write 1 bit of binary value 1 to register 0x11 bit 2”
In Sonic this is accomplished by the following commands:

```
WRITE(11,0004,0004)    //Write a “1” to bit 2 through “4” and mask
                       //off the rest of the bits through the “Mask” of
                       //0004 where only bit 3 is set to 1 in “4”
```

****NOTE: Register 0x11 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 1’b1 to 17.3 to enable 1000Base-X PCS TX & PCS RX functions (per channel)
This is read “write 1 bit of binary value 1 to register 0x11 bit 3”
In Sonic this is accomplished by the following commands:

```
WRITE(11,0008,0008)    //Write a “1” to bit 3 through “8” and mask
                       //off the rest of the bits through the “Mask” of
                       //0008 where only bit 3 is set to 1 in “8”
```

****NOTE: Register 0x11 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 1’b1 to 17.4 to set the nibble order, LSB on rising edge, MSB on falling edge (per channel)
This is read “write 1 bit of binary value 1 to register 0x11 bit 4”
In Sonic this is accomplished by the following commands:

```
WRITE(11,0010,0010)    //Write a “1” to bit 4 through “1” and mask
                       //off the rest of the bits through the “Mask” of
                       //0010 where only bit 4 is set to 1 in “1”
```

****NOTE: Register 0x11 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 1’b1 to 17.5 to enable DDR data for both TX/RX directions (per channel)
This is read “write 1 bit of binary value 1 to register 0x11 bit 5”
In Sonic this is accomplished by the following commands:

```
WRITE(11,0020,0020)    //Write a “1” to bit 5 through “2” and mask
                       //off the rest of the bits through the “Mask” of
                       //0020 where only bit 5 is set to 1 in “2”
```

****NOTE: Register 0x11 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 1’b0 to 17.6 to disable FC_PH overlay detection (per channel)
This is read “write 1 bit of binary value 0 to register 0x11 bit 6”
In Sonic this is accomplished by the following commands:

```
WRITE(11,0000,0040)    //Write a “0” to bit 6 through “0” and mask
                       //off the rest of the bits through the “Mask” of
                       //0040 where only bit 6 is set to 1 in “4”
```

****NOTE: Register 0x11 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 1’b1 to 17.7 to enable comma detection (per channel)
This is read “write 1 bit of binary value 1 to register 0x11 bit 7”
In Sonic this is accomplished by the following commands:

```
WRITE(11,0080,0080)    //Write a “1” to bit 7 through “8” and mask
                       //off the rest of the bits through the “Mask” of
                       //0080 where only bit 7 is set to 1 in “8”
```

****NOTE: Register 0x11 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 1’b0 to 17.9 to disable full DDR mode (per channel)
This is read “write 1 bit of binary value 0 to register 0x11 bit 9”
In Sonic this is accomplished by the following commands:

```
WRITE(11,0000,0100)    //Write a “0” to bit 9 through “0” and mask
                       //off the rest of the bits through the “Mask” of
                       //0100 where only bit 9 is set to 1 in “1”
```

****NOTE: Register 0x11 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

****NOTE: Up until this point, Register 0x11 has been used and individual bits of that register have been changed according to their function. However, for faster execution and cleaner code, all of the bits set to this point can be combined and implemented in a single transaction such as:**

```
WRITE(11,00BC,02FF)  //Write a "0" to bit 9 and "10111100" to bits 7:0 and mask
                     //off the rest of the bits through the "Mask" of
                     //02FF where only bit 9 and bits 7:0 are set to 1 in "2FF"
```

Write 1'b0 to 16.8 to disable Farend Loop Back (per channel)
This is read "write 1 bit of binary value 0 to register 0x10 bit 8"
In Sonic this is accomplished by the following commands:

```
WRITE(10,0000,0100)  //Write a "0" to bit 8 through "0" and mask
                     //off the rest of the bits through the "Mask" of
                     //0100 where only bit 8 is set to 1 in "1"
```

****NOTE: Register 0x10 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 1'b0 to 0.14 to disable loop back mode (per channel)
This is read "write 1 bit of binary value 0 to register 0x00 bit 14"
In Sonic this is accomplished by the following commands:

```
WRITE(00,0000,4000)  //Write a "0" to bit 14 through "0" and mask
                     //off the rest of the bits through the "Mask" of
                     //4000 where only bit 14 is set to 1 in "4"
```

****NOTE: Register 0x00 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Write 3'b111 to 4/5.36874.11:9 to set channel 0 TX swing setting amplitude to 1375mVdfpp
This is read "write 3 bits of binary value 111 to register 0x900A bits 11:9"
In Sonic this is accomplished by the following commands:

```
WRITE(1E,900A)       //Indirectly address register 0x900A
WRITE(1F,0E00,0E00)  //Write a "111" to bits 11:9
                     //and mask off the rest of the bits through the "Mask" of
                     //0E00 where only bits 11:9 are set to 1 in "E"
```

Write 1'b1 to 4/5.36874.8 to set channel 0 TX CM bit
This is read "write 1 bit of binary value 1 to register 0x900A bit 8"
In Sonic this is accomplished by the following commands:

```
WRITE(1E,900A)       //Indirectly address register 0x900A
WRITE(1F,0100,0100)  //Write a "1" to bit 8
                     //and mask off the rest of the bits through the "Mask" of
                     //0100 where only bit 8 is set to 1 in "1"
```

Write 3’b111 to 4/5.36876.11:9 to set channel 1 TX swing setting amplitude to 1375mVdfpp
This is read “write 3 bits of binary value 111 to register 0x900C bits 11:9”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,900C)        //Indirectly address register 0x900C
WRITE(1F,0E00,0E00)   //Write a “111” to bits 11:9
                      //and mask off the rest of the bits through the “Mask” of
                      //0E00 where only bits 11:9 are set to 1 in “E”
```

Write 1’b1 to 4/5.36876.8 to set channel 1 TX CM bit
This is read “write 1 bit of binary value 1 to register 0x900C bit 8”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,900C)        //Indirectly address register 0x900C
WRITE(1F,0100,0100)   //Write a “1” to bit 8
                      //and mask off the rest of the bits through the “Mask” of
                      //0100 where only bit 8 is set to 1 in “1”
```

Write 3’b111 to 4/5.36878.11:9 to set channel 2 TX swing setting amplitude to 1375mVdfpp
This is read “write 3 bits of binary value 111 to register 0x900E bits 11:9”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,900E)        //Indirectly address register 0x900E
WRITE(1F,0E00,0E00)   //Write a “111” to bits 11:9
                      //and mask off the rest of the bits through the “Mask” of
                      //0E00 where only bits 11:9 are set to 1 in “E”
```

Write 1’b1 to 4/5.36878.8 to set channel 2 TX CM bit
This is read “write 1 bit of binary value 1 to register 0x900E bit 8”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,900E)        //Indirectly address register 0x900E
WRITE(1F,0100,0100)   //Write a “1” to bit 8
                      //and mask off the rest of the bits through the “Mask” of
                      //0100 where only bit 8 is set to 1 in “1”
```

Write 3’b111 to 4/5.36880.11:9 to set channel 3 TX swing setting amplitude to 1375mVdfpp
This is read “write 3 bits of binary value 111 to register 0x9010 bits 11:9”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9010)        //Indirectly address register 0x9010
WRITE(1F,0E00,0E00)   //Write a “111” to bits 11:9
                      //and mask off the rest of the bits through the “Mask” of
                      //0E00 where only bits 11:9 are set to 1 in “E”
```

Write 1’b1 to 4/5.36880.8 to set channel 3 TX CM bit
This is read “write 1 bit of binary value 1 to register 0x9010 bit 8”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9010)        //Indirectly address register 0x9010
WRITE(1F,0100,0100)   //Write a “1” to bit 8
                      //and mask off the rest of the bits through the “Mask” of
                      //0100 where only bit 8 is set to 1 in “1”
```

Set RX Equalization Settings:

Write 4’b0001 to 4/5.36866.15:12 to turn on adaptive equalization (4’b0000 is off)
This is read “write 4 bits of binary value 0001 to register 0x9002 bits 15:12”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9002)        //Indirectly address register 0x9002
WRITE(1F,1000,F000)   //Write a “0001” to bits 15:12
                      //and mask off the rest of the bits through the “Mask” of
                      //F000 where only bits 15:12 are set to 1 in “F”
```

Write 4’b0001 to 4/5.36868.15:12 to turn on adaptive equalization (4’b0000 is off)
This is read “write 4 bits of binary value 0001 to register 0x9004 bits 15:12”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9004)        //Indirectly address register 0x9004
WRITE(1F,1000,F000)   //Write a “0001” to bits 15:12
                      //and mask off the rest of the bits through the “Mask” of
                      //F000 where only bits 15:12 are set to 1 in “F”
```

Write 4’b0001 to 4/5.36870.15:12 to turn on adaptive equalization (4’b0000 is off)
This is read “write 4 bits of binary value 0001 to register 0x9006 bits 15:12”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9006)        //Indirectly address register 0x9006
WRITE(1F,1000,F000)   //Write a “0001” to bits 15:12
                      //and mask off the rest of the bits through the “Mask” of
                      //F000 where only bits 15:12 are set to 1 in “F”
```

Write 4’b0001 to 4/5.36872.15:12 to turn on adaptive equalization (4’b0000 is off)
This is read “write 4 bits of binary value 0001 to register 0x9008 bits 15:12”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9008)        //Indirectly address register 0x9008
WRITE(1F,1000,F000)   //Write a “0001” to bits 15:12
                      //and mask off the rest of the bits through the “Mask” of
                      //F000 where only bits 15:12 are set to 1 in “F”
```

Write 2’b01 to 4/5.36866.3:2 for AC coupled mode (2’b00 is DC coupled mode)
This is read “write 2 bits of binary value 01 to register 0x9002 bits 3:2”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9002)        //Indirectly address register 0x9002
WRITE(1F,0004,000C)   //Write a “01” to bits 3:2
                      //and mask off the rest of the bits through the “Mask” of
                      //000C where only bits 3:2 are set to 1 in “C”
```

Write 2’b01 to 4/5.36868.3:2 for AC coupled mode (2’b00 is DC coupled mode)
This is read “write 2 bits of binary value 01 to register 0x9004 bits 3:2”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9004)        //Indirectly address register 0x9004
WRITE(1F,0004,000C)   //Write a “01” to bits 3:2
                      //and mask off the rest of the bits through the “Mask” of
                      //000C where only bits 3:2 are set to 1 in “C”
```

Write 2’b01 to 4/5.36870.3:2 for AC coupled mode (2’b00 is DC coupled mode)
This is read “write 2 bits of binary value 01 to register 0x9006 bits 3:2”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9006)          //Indirectly address register 0x9006
WRITE(1F,0004,000C)     //Write a “01” to bits 3:2
                        //and mask off the rest of the bits through the “Mask” of
                        //000C where only bits 3:2 are set to 1 in “C”
```

Write 2’b01 to 4/5.36872.3:2 for AC coupled mode (2’b00 is DC coupled mode)
This is read “write 2 bits of binary value 01 to register 0x9008 bits 3:2”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9008)          //Indirectly address register 0x9008
WRITE(1F,0004,000C)     //Write a “01” to bits 3:2
                        //and mask off the rest of the bits through the “Mask” of
                        //000C where only bits 3:2 are set to 1 in “C”
```

Set TX DLL Offset:

Write 16’h0028 to 4/5.37888 TX0_DLL_CONTROL
This is read “write 16 bits of hexadecimal value 0028 to register 0x9400 bits 15:0”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9400)          //Indirectly address register 0x9400
WRITE(1F,0028)          //Write a “0000000000101000” to bits 15:0
```

****NOTE: No Mask is required because all bits of register 0x9400 are being set**

Write 16’h0028 to 4/5.37889 TX1_DLL_CONTROL
This is read “write 16 bits of hexadecimal value 0028 to register 0x9401 bits 15:0”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9401)          //Indirectly address register 0x9401
WRITE(1F,0028)          //Write a “0000000000101000” to bits 15:0
```

****NOTE: No Mask is required because all bits of register 0x9401 are being set**

Write 16’h0028 to 4/5.37890 TX2_DLL_CONTROL
This is read “write 16 bits of hexadecimal value 0028 to register 0x9402 bits 15:0”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9402)          //Indirectly address register 0x9402
WRITE(1F,0028)          //Write a “0000000000101000” to bits 15:0
```

****NOTE: No Mask is required because all bits of register 0x9402 are being set**

Write 16’h0028 to 4/5.37891 TX3_DLL_CONTROL

This is read “write 16 bits of hexadecimal value 0028 to register 0x9403 bits 15:0”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,9403)        //Indirectly address register 0x9403
WRITE(1F,0028)        //Write a “0000000000101000” to bits 15:0
```

****NOTE: No Mask is required because all bits of register 0x9403 are being set**

Poll Serdes PLL Status for Locked State:

Read 4/5.36891.4,0 SERDES_PLL_STATUS-PLL_LOCK_TX/RX

This is read “read register 0x901B bits bits 4 and 0”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,901B)        //Indirectly address register 0x901B
READ(1F,0011,0011)    //Read bit 4 and bit 0 of register 901B and compare the
                      //the expected value of “0011” with the bits masked off the
                      // through the “Mask” of 0011 where only bit 4 and bit 0
                      //are set to 1 in “11”
```

If the bits read are equal to the bits in the expected value field then the text color in Sonic will be Green, otherwise the text color will be red. Repeat this register read until bit 4 and bit 0 of register 0x901B are equal to “1.”

Issue a Data Path Reset:

Write 1’b1 to 16.11 to issue a TX and RX Data Path Reset (per channel)

This is read “write 1 bit of binary value 1 to register 0x10 bit 11”
In Sonic this is accomplished by the following commands:

```
WRITE(10,0800,0800)   //Write a “1” to bit 11 through “8” and mask
                      //off the rest of the bits through the “Mask” of
                      //0800 where only bit 11 is set to 1 in “8”
```

****NOTE: Register 0x10 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Clear Latched Registers:

Read 1 PHY_STATUS_1 to clear (per channel)

This is read “read register 0x01”
In Sonic this is accomplished by the following commands:

```
READ(01)        //Read all bits of register 0x01
```

****NOTE: Register 0x01 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Read 18 PHY_RX_CTC_FIFO_STATUS to clear (per channel)
This is read “read register 0x12”
In Sonic this is accomplished by the following commands:

```
READ(12)        //Read all bits of register 0x12
```

****NOTE: Register 0x12 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Read 19 PHY_TX_CTC_FIFO_STATUS to clear (per channel)
This is read “read register 0x13”
In Sonic this is accomplished by the following commands:

```
READ(13)        //Read all bits of register 0x13
```

****NOTE: Register 0x13 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Read 28 PHY_CHANNEL_STATUS to clear (per channel)
This is read “read register 0x1C”
In Sonic this is accomplished by the following commands:

```
READ(1C)        //Read all bits of register 0x1C
```

****NOTE: Register 0x1C is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Read 4/5.36891 SERDES_PLL_STATUS to clear
This is read “read all bits of register 0x901B”
In Sonic this is accomplished by the following commands:

```
WRITE(1E,901B)          //Indirectly address register 0x901B
READ(1F)                //Read all bits of register 901B
```

Operational Mode Status:

Read Verify 1.2 PHY_STATUS_1 – Link Status (1’b1) (per channel)
This is read “read register 0x01 bit 2”
In Sonic this is accomplished by the following commands:

```
READ(01,0105,0004)      //Read all bits of register 01 and compare the
                        //the expected value of “0105” with the bits masked off the
                        // through the “Mask” of 0004 where only bit 2 is set to 1 in
                        //”4” since that is the only bit we care about
```

****NOTE: Register 0x01 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

If bit 2 is read as a “1” the link is up and it will be equal to the bits in the expected value field of “0105” and the text color in Sonic will be Green, otherwise the text color will be red.

Read Verify 18.15 PHY_RX_CTC_FIFO_STATUS – RX_CTC_Reset (1’b0) (per channel)
This is read “read register 0x12 bit 15”
In Sonic this is accomplished by the following commands:

```
READ(12,0000,8000)    //Read all bits of register 0x12 and compare the expected
                      //value of “0000” with bit 15 masked off through the
                      //”Mask” of 8000 where only bit 15 is set to 1 in “8” since
                      //that is the only bit we care about
```

If bit 15 is read as a “0” the FIFO has not overflowed or been reset and it will be equal to the bit in the expected value field of “0000” and the text color in Sonic will be Green, otherwise the text color will be red.

****NOTE: Register 0x12 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

Read Verify 19.15 PHY_TX_CTC_FIFO_STATUS – TX_FIFO_Reset_1Gx (1’b0) (per channel)
This is read “read register 0x13 bit 15”
In Sonic this is accomplished by the following commands:

```
READ(13,0000,8000)    //Read all bits of register 0x13 and compare the expected
                      //value of “0000” with bit 15 masked off through the
                      //”Mask” of 8000 where only bit 15 is set to 1 in “8” since
                      //that is the only bit we care about
```

****NOTE: Register 0x13 is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

If bit 15 is read as a “0” the TX FIFO and FIFO have not had a collision or been reset and it will be equal to the bit in the expected value field of “0000” and the text color in Sonic will be Green, otherwise the text color will be red.

Read Verify 28.13:12 PHY_CHANNEL_STATUS – Enc/Dec Invalid Code Word (2’b00) (per channel)
This is read “read register 0x1C bits 13:12”
In Sonic this is accomplished by the following commands:

```
READ(1C,0000,3000)    //Read all bits of register 0x1C and compare the expected
                      //value of “0000” with bits 13:12 masked off through the
                      //”Mask” of 3000 where only bits 13:12 are set to 1 in “8”
                      //since they are the only bits we care about
```

****NOTE: Register 0x1C is accessible directly through Clause 22, therefore Indirect Addressing is not required.**

If bits 13:12 are read as “00” the encoder and decoders have not received invalid control words and it will be equal to the bit in the expected value field of “0000” and the text color in Sonic will be Green, otherwise the text color will be red.

Read Verify 4/5.36891.4 SERDES_PLL_STATUS – PLL_LOCK_RX (1'b1)

This is read "read register 0x901B bit 4"
In Sonic this is accomplished by the following commands:

```
WRITE(1E,901B)        //Indirectly address register 0x901B
READ(1F,0010,0010)    //Read all bits of register 901B and compare the expected
                      //value of "0010" with bit 4 masked off through the
                      //"Mask" of 0010 where only bit 4 is set to 1 in "1"
                      //since this is the only bit we care about
```

If bit 4 is read as a "1" the PLL is locked within 10ppm of REFCLKP/N in the SERDES RX macro and it will be equal to the bit in the expected value field of "0000" and the text color in Sonic will be Green, otherwise the text color will be red.

Read Verify 4/5.36891.0 SERDES_PLL_STATUS – PLL_LOCK_TX (1'b1)

This is read "read register 0x901B bit 0"
In Sonic this is accomplished by the following commands:

```
WRITE(1E,901B)        //Indirectly address register 0x901B
READ(1F,0011,0001)    //Read all bits of register 901B and compare the expected
                      //value of "0011" with bit 0 masked off through the
                      //"Mask" of 0001 where only bit 0 is set to 1 in "1"
                      //since this is the only bit we care about
```

If bit 0 is read as a "1" the PLL is locked within 10ppm of REFCLKP/N in the SERDES TX macro and it will be equal to the bit in the expected value field of "0000" and the text color in Sonic will be Green, otherwise the text color will be red.

Revision History:

June 2008 – Initial Creation (JN)
July 2008 – Rev 0.1 (JN)
September 2008 – Rev 0.2 (JN)