Application Note

Using SMI of C2000 EtherCAT Slave Controller for Ethernet PHY Configuration

Bruce Liu, Chen Gao *Systems Engineering and Marketing*

ABSTRACT

The Ethernet physical layer (PHY) is a transceiver component for transmitting and receiving data of Ethernet frames and the PHY device implements the physical layer in the open systems interconnection (OSI) model. The PHY device acts as a bridge between the medium access controller (MAC - data link layer in OSI model) and a physical medium such as copper or fiber cable.

The serial management interface (SMI) provides access to the PHY device internal register space for status information and configuration. Proper PHY configuration using SMI is fundamental during the prototype stage, and also crucial to meeting the requirements of the lowest deterministic latency and fastest link detection in industrial Ethernet applications such as EtherCAT®. The SMI is compatible with IEEE 802.3 clause 22 and clause 45. The implemented register set consists of the registers required by the IEEE 802.3 plus several others to provide additional visibility and controllability of the PHY device.

This application note provides guidance on the Ethernet PHY configuration using SMI of the EtherCAT slave controller (ESC) in the C2000™ device for industrial applications.

Table of Contents

List of Figures

Trademarks

C2000™ and Code Composer Studio™ are trademarks of Texas Instruments.
EtherCAT® and Beckhoff® are registered trademarks of Beckhoff Automation GmbH.
All trademarks are the property of their respective owners.

1 Introduction of SMI

Ethernet for Control Automation Technology (EtherCAT®) is an Ethernet-based fieldbus system, invented by Beckhoff® Automation and is standardized in IEC 61158. All the slave nodes connected to the bus interpret, process, and modify the addressed data *on the fly* (when needed basis), without having to buffer the frame inside the node. This real-time behavior, frame processing, and forwarding requirements are implemented by the EtherCAT slave controller hardware. EtherCAT does not require software interaction for data transmission inside the slaves. EtherCAT only defines the MAC layer while the higher layer protocols and stack are implemented in software on the microcontrollers connected to the ESC. The SMI in C2000 ESC is called PHY management interface used for communication with the Ethernet PHYs. See Figure 1-1 for the connectivity of SMI.

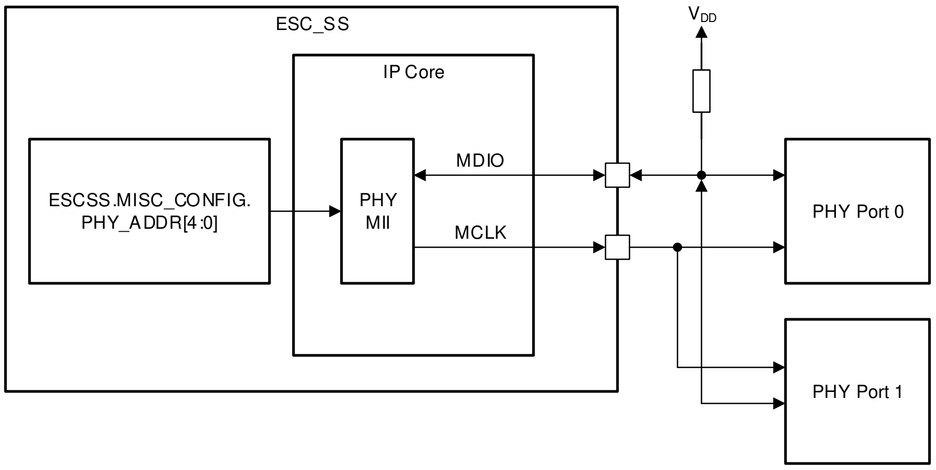


Figure 1-1. PHY Management Interface Connectivity

MDIO must have a pullup resistor (4.7 kΩ recommended) externally. MCLK is driven rail-to-rail, idle value is HIGH.

The SMI includes the management clock (MDC) and the management input/output data pin (MDIO). MDC is sourced by the ESC, and can run at a maximum clock rate of 25 MHz. MDC is not expected to be continuous, and can be turned off by the ESC when the bus is idle.

MDIO is sourced by the ESC and by the PHY. The data on the MDIO pin is latched on the rising edge of the MDC. MDIO pin requires a pullup resistor, which pulls MDIO high during IDLE and turnaround.

2 PHY Selection and Configuration for EtherCAT

When selecting the proper PHY for EtherCAT, first review the Application note – PHY selection guide on the EtherCAT home page. This application note describes the specifications and the recommendations of PHY performance from the EtherCAT perspective. The document also lists a variety of PHYs from TI, such as the DP836x, DP838x, TLK10x, and TLK11x.

Proper configuration of PHY to comply with IEEE 802.3 100BaseTX or 100BaseFX, includes:

- Support 100Mbps full-duplex links
- Provide an MII (or RMII, RGMII) interface
- Auto-negotiation in 100Base TX mode
- Support MII management interface
- MDI, MDI-X auto-crossover in 100BaseTX mode
- Receive and transmit delays must comply with the standard (the RX delay target is below about 320 ns, the TX delay is below about 140 ns)
- Must offer the RX_ER signal (MII, RMII) or RX_ER as part of the RX_CTL signal (RGMII)
- Link loss reaction time (link loss to link signal or LED output change) must be faster than 15 µs to enable redundancy operation

To setup the PHY in the correct mode to work in the EtherCAT environment, the serial management interface (SMI) can be used to program the PHY to be setup in a specific mode.

Up to 32 PHYs can share a common SMI bus. To distinguish between the PHYs, an x-bit address is used. During power up or hardware reset, the PHY device latches the PHY_AD[x:0] configuration pins to determine the address. The address can be changed by adding the required pullup or pulldown resistors defined in the bootstrap section of the PHY device data sheet. The bootstrap pin can also be used for PHY configuration. This application note only focuses on using the SMI. See Figure 2-1 for the hardware bootstrap diagram.

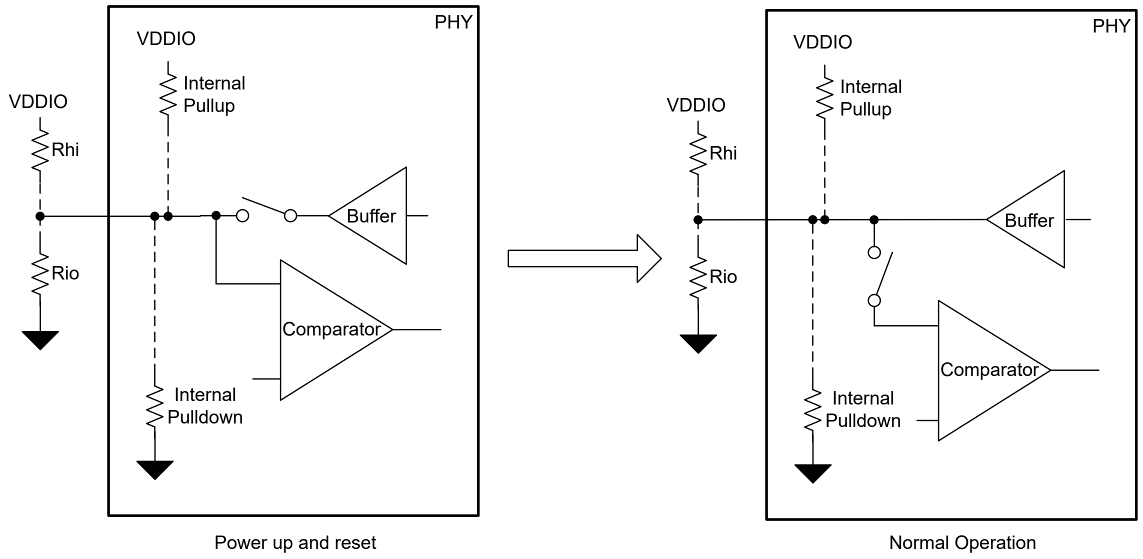


Figure 2-1. Hardware Bootstraps

The C2000 ESC addresses the Ethernet PHYs typically using the logical port number plus the PHY address offset. In the best situation, the Ethernet PHY addresses correspond with the logical port number, so PHY addresses 0 and 1 are used. A PHY address offset of 0 to 31 can be applied which moves the PHY addresses to any consecutive address range. The ESC module expects logical port 0 to have PHY address 0 plus the PHY address offset. The PHY address offset can be selected in register ESCSS_MISC_CONFIG.PHY_ADDR[4:0].

Before entering into the desired operation mode, the PHY device must get out of the RESET condition by applying a high level to the RESET pin. This RESET signal is generated out of the ESC module. Since there are no pull devices active on the MCU during and after reset, a pulldown resistor must be added on this signal on the board level. In some cases, PHYs can be released from reset after releasing the ESC module. To generate a delay, the pin for nPHY_RESET can be used as an I/O and is switched later to the alternate output function. Moreover, a hardware reset can reinitialize all the PHY registers to default values by applying a low pulse, with a duration of at least 10 μs (T1) to the RESET pin (take DP83822 PHY for example, see Figure 2-2).

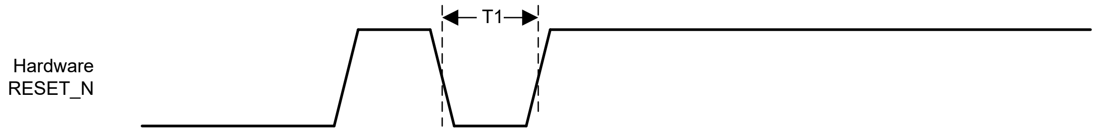


Figure 2-2. DP83822 PHY Hardware Reset Signal

The interface diagram between ESC and PHY device is shown in Figure 2-3. The PHY can be clocked using the ESCSS_PHY_CLK signal, if needed, otherwise provide an external 25-MHz source to the PHY and ESC (both must be clocked from the same source).

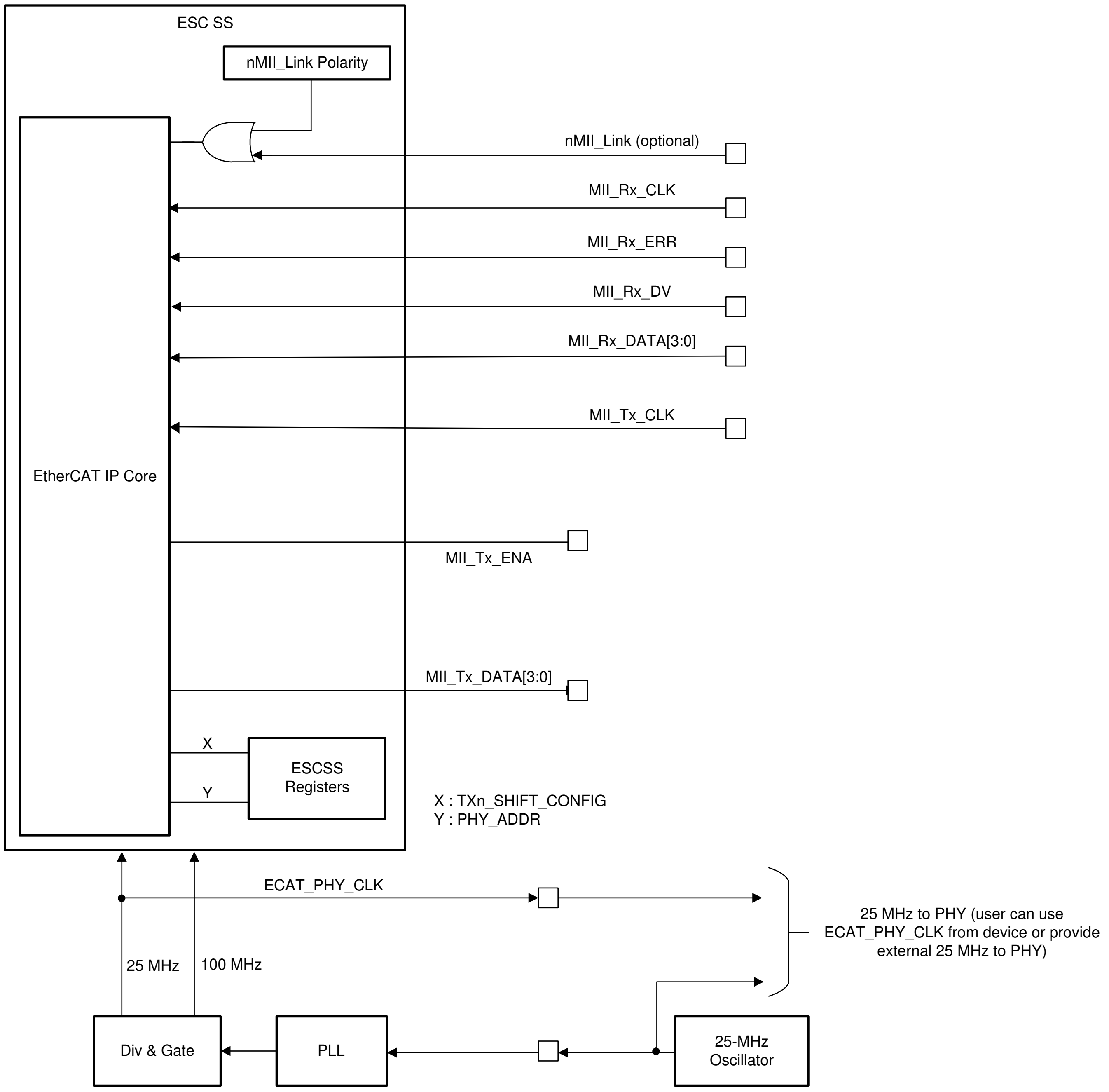


Figure 2-3. ESC PHY MII Interface Diagram

3 How to Read and Write to the PHY Register Using SMI of ESC

3.1 PHY Register Configuration for EtherCAT

Using the F28388D controlCARD (TMDSCNCD28388D) as the example, there are two EtherCAT ports on the control card with PHY address 0x00 and 0x01, respectively. Use the SMI to program the DP83822 PHY registers to work for EtherCAT 100BASE-TX as follows.

- *LED 0:*

  PHY register 0x19 value 0x8020 (Auto-MDIX enable and enable LED0 config)

  PHY register 0x18 value 0x0080 (Active High polarity)
- *LED 1:*

  PHY register 0x469 value 0x0440 (Active High polarity)

  Auto negotiate enable configuration:

  Make sure PHY register 0x04 value is 0x01E1 by default (Advertise which modes PHY support)

  PHY register 0x09 value 0x0020 (Enable Robust Auto MDIX)

  PHY register 0x00 value 0x3300 (Enable Auto negotiate and restart process)
- *Odd-nibble Detection Disable Configuration:*

  PHY register 0x0A value 0x0002 (Disable Odd-nibble detection)
- *Fast Link-Drop Enable:*

  PHY register 0x0B value 0x0008 (Enable FLD with correct FLD features RX Error count)

3.2 Steps to Read or Write PHY Register in C2000 ESC

1. *Give PDI access to MII management:*

   According to the ESC hardware data sheet, by default MII management control is set only to the ECAT master. PDI has to claim access to MII management via the EtherCAT IP register *MII Management PDI Access State (0x0517)*. This opens the MII management access to PDI. Once this is enabled, the PDI can access the PHY registers via the *MII Management Control* and *PHY Address* registers. An example is shown in the following code:

   ```
   #define ESC_MII_PDI_ACCESS_OFFSET 0x0517 //0x28B - High for C28x, 0x0517 for CM
   ESC_writeWordISR(0x0100, ESC_MII_ECAT_ACCESS_OFFSET); //0x0100 for C28x, 0x01 for CM
   ```

   The ESC base address and offset are different between C28 core and CM (M4) core due to the addressable register length (16-bit for C28 and 8-bit for CM). Since the ESC register is 8-bit, the offset and address using C28 core control needs to be halved.
2. *Set PHY address to read or write:*

   The ESC register 0x0512 defines the PHY address to be read or write. See the following code:

   ```
   #define ESC_PHY_ADDRESS_OFFSET 0x0512 //0x289 - low for C28x, 0x0512 for CM
   ESC_writeWordISR(PHY address, ESC_PHY_ADDRESS_OFFSET);
   ```

3. *Set the value to write to PHY register (write):*

   The ESC register 0x0514:0x0515 defines the PHY data to be read or write to the PHY register. See the following code:

   ```
   #define ESC_PHY_DATA_OFFSET 0x0514 //0x28A - low for C28x, 0x0514 for CM
   ESC_writeWordISR(PHY data, ESC_PHY_DATA_OFFSET);
   ```

4. *Initiate read or write command:*

   The ESC register 0x0510:0x0511 defines the MII management control and status. The bit 9:8 defines the commands to read or write the PHY register.

   Bit 9:8 commands:

   00: No command/MI idle (clear error bits)

   01: Read

   10: Write

   11: Reserved/invalid command (do not issue)

   See the following code:

```
#define ESC_MII_CTRL_STATUS_1_OFFSET 0x0510 //0x288 - low for C28x, 0x0510 for CM
#define ESC_MII_CTRL_STATUS_2_OFFSET 0x0511 //0x288 - high for C28x, 0x0511 for CM
ESC_writeWordISR(0x0200, ESC_MII_CTRL_STATUS_1_OFFSET); //Write command for C28x
ESC_writeWordISR(0x0200, ESC_MII_CTRL_STATUS_2_OFFSET); // Write command for CM
ESC_writeWord(0x0100, ESC_MII_CTRL_STATUS_1_OFFSET); //Read command for C28x
ESC_writeWord(0x0100, ESC_MII_CTRL_STATUS_2_OFFSET); //Read command for CM
```

The previously-described approach is to access standard registers 0 to 31 defined in IEEE 802.3. For accessing the clause 45 extended register set, the Register Control Register (REGCR, address 0x000D) and Data Register (ADDAR, address 0x000E) need to be set. Register REGCR [4:0] is the device address DEVAD that directs any accesses of the ADDAR register to the appropriate MDIO manageable device. The following example demonstrates a write operation with no post increment according to the example write operation section of the *DP83822 Robust, Low Power 10/100 Mbps Ethernet Physical Layer Transceiver* data sheet. In this example, the MAC impedance is adjusted to 99.25 Ω using the IO MUX GPIO Control Register (IOCTRL, address 0x0461).

- Write the value 0x001F to register 0x000D
- Write the value 0x0461 to register 0x000E (Sets desired register to the IOCTRL)
- Write the value 0x401F to register 0x000D
- Write the value 0x0400 to register 0x000E (Sets MAC impedance to 99.25 Ω)

See the code in CCS.

```
#define ESC_PHY_REG_ADDRESS_OFFSET 0x0513 //0x289 - High for C28x, 0x0513 for CM
#define ESC_PHY_DATA_OFFSET 0x0514 //0x28A - low for C28x, 0x0514 for CM
#define ESC_MII_CTRL_STATUS_1_OFFSET 0x0510 //0x288 - low for C28x, 0x0510 for CM
#define ESC_MII_CTRL_STATUS_2_OFFSET 0x0511 //0x288 - high for C28x, 0x0511 for CM
ESC_writeWordISR(0x0D00, ESC_PHY_REG_ADDRESS_OFFSET); //0x0D for CM, set extended PHY register
control
ESC_writeWordISR(0x001F, ESC_PHY_DATA_OFFSET); // DEVAD for MMD
ESC_writeWord(0x0200, ESC_MII_CTRL_STATUS_1_OFFSET); //write command for C28x, status_2_offset
register for CM
ESC_writeWordISR(0x0E00, ESC_PHY_REG_ADDRESS_OFFSET); //0x0E for CM, set extended PHY Data register
ESC_writeWordISR(0x0461, ESC_PHY_DATA_OFFSET); // PHY extended register address
ESC_writeWord(0x0200, ESC_MII_CTRL_STATUS_1_OFFSET); //write command for C28x, status_2_offset
register for CM
ESC_writeWordISR(0x0D00, ESC_PHY_REG_ADDRESS_OFFSET); //0x0D for CM, set extended PHY register
control
ESC_writeWordISR(0x401F, ESC_PHY_DATA_OFFSET); // change to Data in REGCR Bit 15:14
ESC_writeWord(0x0200, ESC_MII_CTRL_STATUS_1_OFFSET); //write command for C28x, status_2_offset
register for CM
ESC_writeWordISR(0x0E00, ESC_PHY_REG_ADDRESS_OFFSET); //0x0E for CM, set extended PHY Data register
ESC_writeWordISR(0x0400, ESC_PHY_DATA_OFFSET); // PHY extended register value to be written
ESC_writeWord(0x0200, ESC_MII_CTRL_STATUS_1_OFFSET); //write command for C28x, status_2_offset
register for CM
```

To read the extended PHY register, the following example demonstrates a read operation. In this example, the MMD7 Energy Efficient Ethernet Link Partner Ability Register (MMD7_EEE_LP_ABILITY, address 0x703D) is read.

- Write the value 0x0007 to register 0x000D
- Write the value 0x003D to register 0x000E (Sets desired register to the MMD7_EEE_LP_ABILITY)
- Write the value 0x4007 to register 0x000D
- Read the value of register 0x000E (Data read is the value contained within the MMD7_EEE_LP_ABILITY)

See the code in CCS.

```
#define ESC_PHY_REG_ADDRESS_OFFSET 0x0513 //0x289 - High for C28x, 0x0513 for CM
#define ESC_PHY_DATA_OFFSET 0x0514 //0x28A - low for C28x, 0x0514 for CM
#define ESC_MII_CTRL_STATUS_1_OFFSET 0x0510 //0x288 - low for C28x, 0x0510 for CM
#define ESC_MII_CTRL_STATUS_2_OFFSET 0x0511 //0x288 - high for C28x, 0x0511 for CM
ESC_writeWordISR(0x0D00, ESC_PHY_REG_ADDRESS_OFFSET); //0x0D for CM, set extended PHY register
control
ESC_writeWordISR(0x0007, ESC_PHY_DATA_OFFSET); // DEVAD for MMD7
ESC_writeWord(0x0200, ESC_MII_CTRL_STATUS_1_OFFSET); //write command for C28x, status_2_offset
register for CM
ESC_writeWordISR(0x0E00, ESC_PHY_REG_ADDRESS_OFFSET); //0x0E for CM, set extended PHY Data register
ESC_writeWordISR(0x003D, ESC_PHY_DATA_OFFSET); // PHY extended register address
ESC_writeWord(0x0200, ESC_MII_CTRL_STATUS_1_OFFSET); //write command for C28x, status_2_offset
register for CM
ESC_writeWordISR(0x0D00, ESC_PHY_REG_ADDRESS_OFFSET); //0x0D for CM, set extended PHY register
control
ESC_writeWordISR(0x4007, ESC_PHY_DATA_OFFSET); // change to Data in REGCR Bit 15:14
ESC_writeWord(0x0200, ESC_MII_CTRL_STATUS_1_OFFSET); //write command for C28x, status_2_offset
register for CM
ESC_writeWordISR(0x0E00, ESC_PHY_REG_ADDRESS_OFFSET); //0x0E for CM, set extended PHY Data register
ESC_writeWord(0x0100, ESC_MII_CTRL_STATUS_1_OFFSET); //Read command for C28x, status_2_offset
register for CM
```

3.3 Using the Script to Debug Ethernet PHY Register in CCS

Section 3.2 described the method to read and write the PHY register with C2000 ESC. To simplify the approach, a script can be used to read or write the PHY register dump and print in the console window in CCS during the debug phase.

The General Extension Language (GEL) can be used to configure the Code Composer Studio™ development environment and to initialize the target CPU. GEL is an interpreted language, and the syntax is similar to that of C. A rich set of built-in GEL functions are available, or custom GEL functions can be written. This document describes ways to create well-written GEL start-up files. For a summary of recommendations, see GEL Guidelines.

The step-by-step guide is detailed in the following list:

1. Config the correct PHY address and cores (c28x or cm) ready to use in C2000_ESC_PHY_READ.gel
2. Add GEL file using Tools > GEL Files > Load GEL option
3. After initializing PHYs and ESC SMI (for example; run f2838x_cpu1_pdi_hal_test_app), then run the script using Scripts > ESC SMI configuration > C2000_ESC_PHY_Reg_Dump

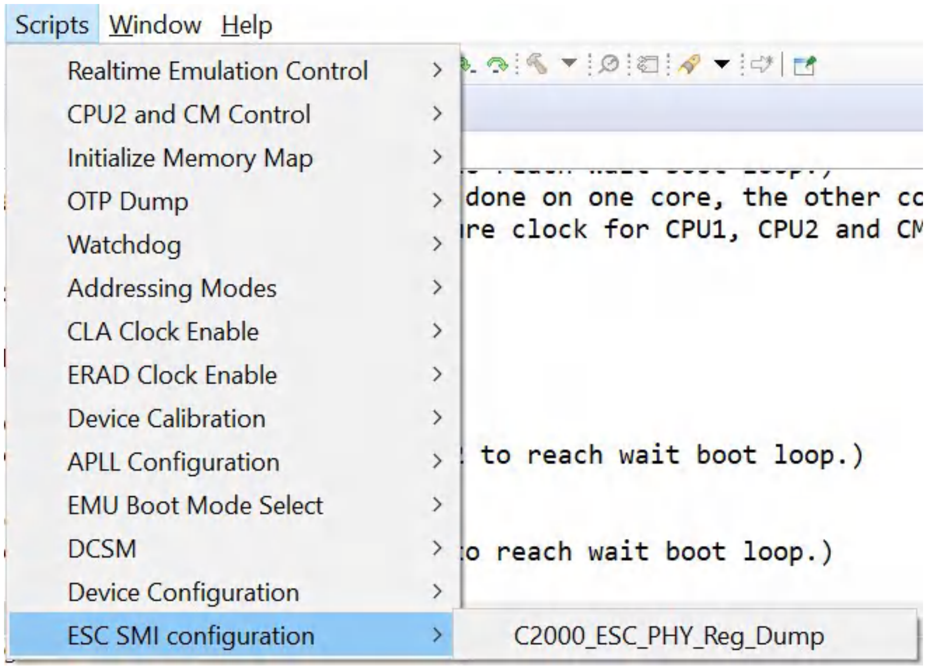


Figure 3-1. Script Interface

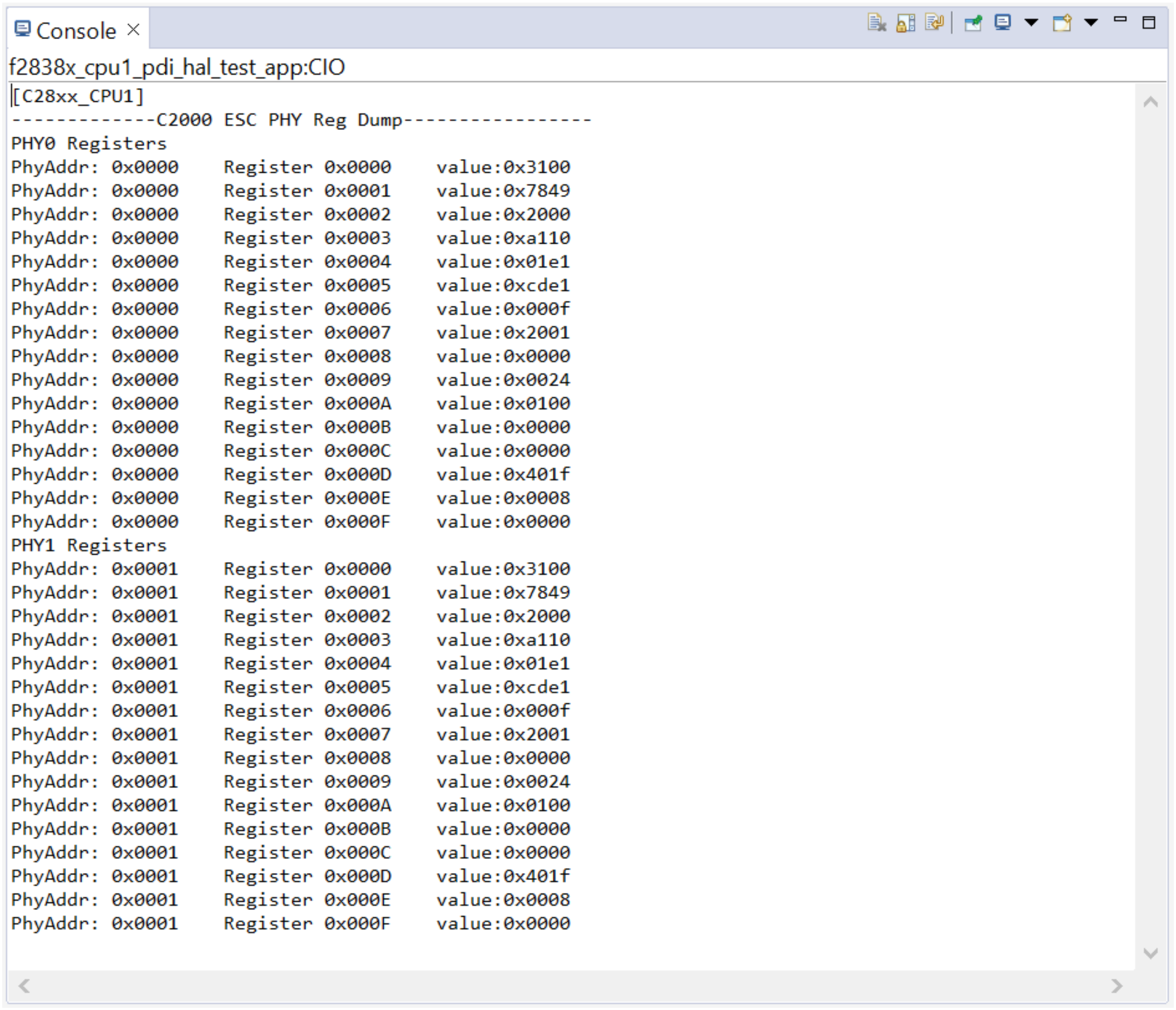
Frequently accessed GEL functions can be added to the GEL menu of the Code Composer Studio menu bar as shown in Figure 3-1. To do this, use the "menuitem" keyword to create a new drop-down list of menu items under the GEL menu. Then use the keywords "hotmenu", "dialog", or "slider" to add new menu items in the most recent drop-down list. When the user-defined menu item is selected (under the GEL menu), a dialog box or slider object appears. The example code follows:

```
menuitem "ESC SMI configuration";
hotmenu C2000_ESC_PHY_Reg_Dump()
{}
```

The method of reading and writing registers in the gel file needs to be rewritten because there is no intervention of library functions. The example code follows:

```
ESC_readWordISR(unsigned short wordValue, unsigned short address)
{
unsigned short *p_address;
p_address = (unsigned short *) address;
wordValue = *p_address;
}
ESC_writeWordISR(unsigned short wordValue, unsigned short address)
{
unsigned short *p_address;
p_address = (unsigned short *) address;
wordValue = *p_address;
}
```

Finally, find the PHY register dump in the console window as Figure 3-2 shows.



```
[C28xx_CPU1]
-------------C2000 ESC PHY Reg Dump----------------
PHY0 Registers
PhyAddr: 0x0000    Register 0x0000    value:0x3100
PhyAddr: 0x0000    Register 0x0001    value:0x7849
PhyAddr: 0x0000    Register 0x0002    value:0x2000
PhyAddr: 0x0000    Register 0x0003    value:0xa110
PhyAddr: 0x0000    Register 0x0004    value:0x01e1
PhyAddr: 0x0000    Register 0x0005    value:0xcde1
PhyAddr: 0x0000    Register 0x0006    value:0x000f
PhyAddr: 0x0000    Register 0x0007    value:0x2001
PhyAddr: 0x0000    Register 0x0008    value:0x0000
PhyAddr: 0x0000    Register 0x0009    value:0x0024
PhyAddr: 0x0000    Register 0x000A    value:0x0100
PhyAddr: 0x0000    Register 0x000B    value:0x0000
PhyAddr: 0x0000    Register 0x000C    value:0x0000
PhyAddr: 0x0000    Register 0x000D    value:0x401f
PhyAddr: 0x0000    Register 0x000E    value:0x0008
PhyAddr: 0x0000    Register 0x000F    value:0x0000
PHY1 Registers
PhyAddr: 0x0001    Register 0x0000    value:0x3100
PhyAddr: 0x0001    Register 0x0001    value:0x7849
PhyAddr: 0x0001    Register 0x0002    value:0x2000
PhyAddr: 0x0001    Register 0x0003    value:0xa110
PhyAddr: 0x0001    Register 0x0004    value:0x01e1
PhyAddr: 0x0001    Register 0x0005    value:0xcde1
PhyAddr: 0x0001    Register 0x0006    value:0x000f
PhyAddr: 0x0001    Register 0x0007    value:0x2001
PhyAddr: 0x0001    Register 0x0008    value:0x0000
PhyAddr: 0x0001    Register 0x0009    value:0x0024
PhyAddr: 0x0001    Register 0x000A    value:0x0100
PhyAddr: 0x0001    Register 0x000B    value:0x0000
PhyAddr: 0x0001    Register 0x000C    value:0x0000
PhyAddr: 0x0001    Register 0x000D    value:0x401f
PhyAddr: 0x0001    Register 0x000E    value:0x0008
PhyAddr: 0x0001    Register 0x000F    value:0x0000
```

Figure 3-2. F28388 Control Card With DP83822 Register Dump

4 Summary

This application note provided guidance on the Ethernet PHY configuration using SMI of the EtherCAT slave controller in C2000 device for industrial applications. And also provides a simply script tool in code composer to read/write PHY register.

5 References

1. Texas Instruments, *DP83822 Robust, Low Power 10/100 Mbps Ethernet Physical Layer Transceiver* data sheet
2. Texas Instruments, *TMS320F2838x Real-Time Microcontrollers with Connectivity Manager* technical reference manual
3. Texas Instruments, *TMS320F28388D controlCARD Information Guide* user guide
4. Beckhoff, *EtherCAT Slave Controller* Hardware Data Sheet Section I
5. Beckhoff, *EtherCAT Slave Controller – PHY selection guide* application note