Application Report

Ethernet PHY Configuration Using MDIO for Industrial Applications

Garrett Ding, Pratheesh Gangadhar TK, David Zaucha

ABSTRACT

As a bridge of the link layer device medium access controller (MAC) and physical medium such as copper cable, the Ethernet physical layer device (PHY) integrates all the physical-layer functions needed to transmit and receive data on standard twisted-pair cables. Proper PHY configuration using management data input/output (MDIO) is fundamental during the prototype stage, and also crucial to meeting the requirements of lowest deterministic latency and fastest link detection in industrial Ethernet applications such as EtherCAT®. This application report provides guidance on the Ethernet PHY configuration using the MDIO module within the Programmable-Realtime Unit Industrial Communications Sub-System (PRU-ICSS) in the Sitara™ device from TI, for industrial applications, by dissecting the PHY DP83822 configuration in EtherCAT on the AMIC110 Industrial Communications Engine (ICE). The goal of this application report is to expedite the development of industrial Ethernet applications on custom boards with migration and troubleshooting guides for the PHYs [1] [2] [3].

Table of Contents

List of Figures

List of Tables

Trademarks

1 PHY Selection and Connection

Many industrial Ethernet applications require PHY to comply with IEEE 802.3 100BaseTX or 100BaseFX, support 100-Mbps full-duplex links, use auto-negotiation, and support MDI/MDI-X auto-crossover in 100BaseTX mode. Moreover, link loss reaction time is also a key benchmark to ensure that link sensitivity is faster than 15 µs for redundancy operation. Although most PHYs support several media-independent interfaces with different pin counts and data rates to communicate with the MAC, the MII is recommended, because it reduces the additional forwarding delay caused by the TX FIFO in RMII. Figure 1-1 shows a typical Ethernet PHY connection with MAC and physical medium.

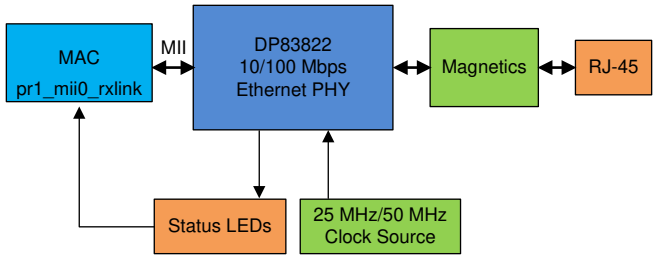

Figure 1-1. Ethernet PHY Connection With MAC and Physical Medium

The AMIC110 Industrial Communications Engine (ICE) is a development platform targeted for industrial communications and industrial Ethernet in particular. The AMIC110 ICE is the Sitara AMIC110 System-on-Chip (SoC) that features the ARM® Cortex™-A8 processor, with the PRU-ICSS, which enables the integration of real-time industrial protocols, without needing ASIC or FPGA. The power-saving DP83822 device (see Figure 1-2) was selected in the AMIC110 ICE. The DP83822 not only meets the requirements of IEEE 802.3u, but also maintains high margins in terms of cross-talk and alien noise.

The DP83822 device is a fully featured, single-port Physical Layer transceiver for 10BASE-Te, 100BASE-TX, and 100BASE-FX signaling, and its signals fall into the following categories:

- MAC interface
- Serial management interface
- Clock interface
- GPIO and LED interface
- Media-dependent interface
- Power and ground pins
- Other pins

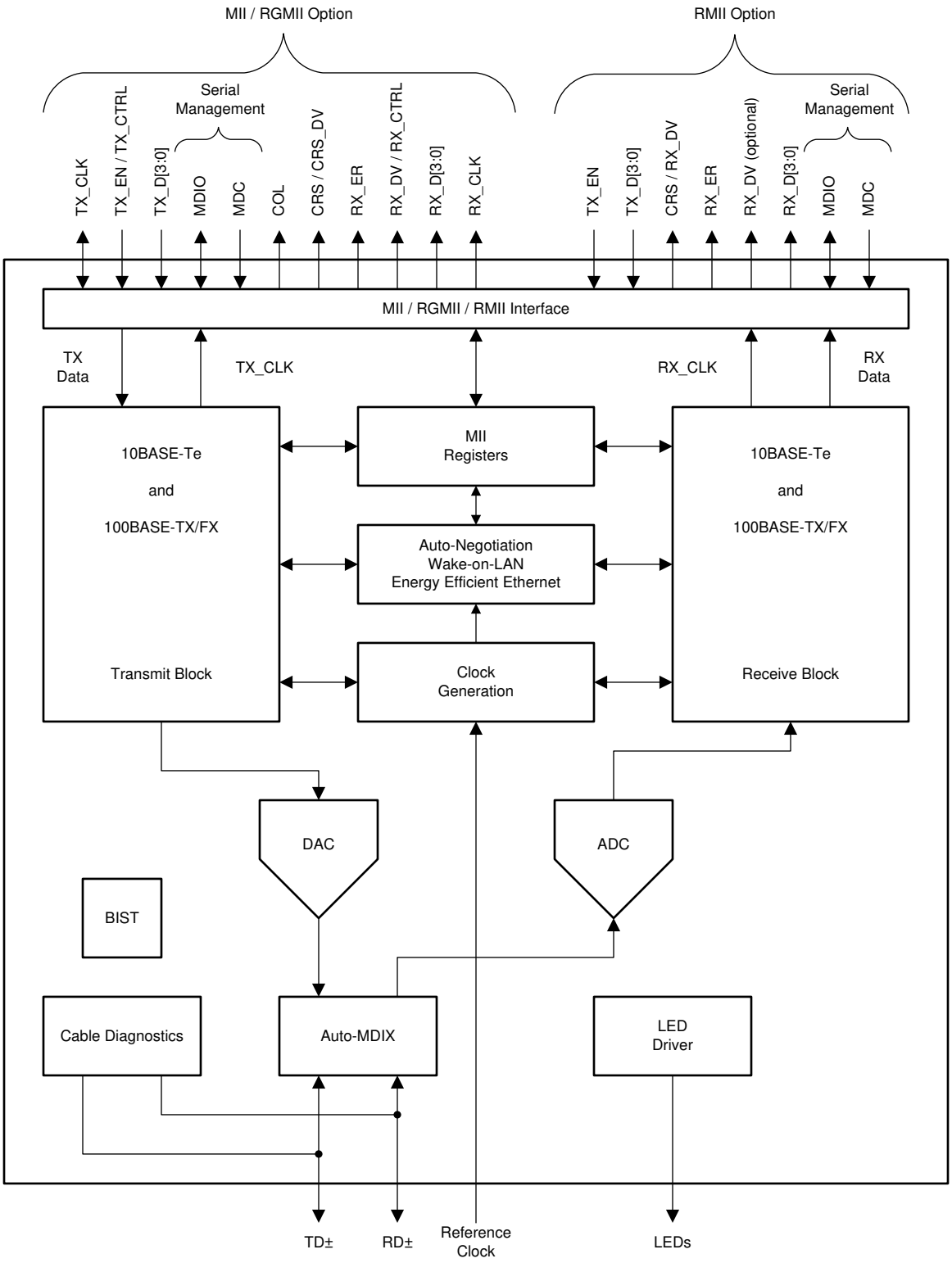

Figure 1-2. DP83822 Functional Block Diagram

Beckhoff provides a comprehensive PHY selection guide to assist in PHY selection and configuration[1]. This document lists a variety of PHYs from TI, such as the DP836x, DP838x, TLK10x, and TLK11x.

2 PHY Reset and Address

PHY bootstrap configurations, which place the device into the desired operation mode, are performed at power up or hardware reset. A hardware reset is accomplished by applying a low pulse, with a duration of at least 10 µs (T1), to the RESET pin in the DP83822 PHY. This pulse resets the device so that all registers are reinitialized to default values, and the hardware configuration values are relatched into the device (see Figure 2-1).

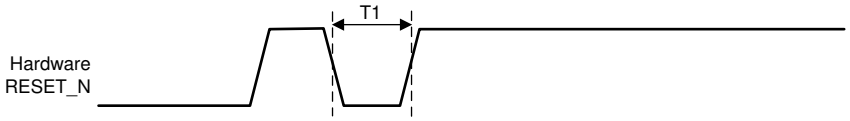

Figure 2-1. PHY Reset Signal

Because the bootstrap pins have alternative functions after the reset is deasserted, to minimize interference during bootstrap pin signal sampling the PRU-ICSS MII multiplexing should not occur until after the PHY RESET is released on the AMIC110 ICE. If these bootstrap pins are not configured, they will be GPIO and High-Z as soon as the reset is released. The hardware reset sequence implementation in the TI EtherCAT looks as follows:

```
Board_init(BOARD_INIT_MODULE_CLOCK);
Board_phyReset(2);
/* mux PRU MII after PHY reset in case PRU drives signals
* on the pins and interfere with the PHY bootstrap configurations
*/
#ifdef ENABLE_UART_PRINT
Board_init(BOARD_INIT_UART_STDIO|BOARD_INIT_ICSS_PINMUX);
#else
Board_init(BOARD_INIT_ICSS_PINMUX);
```

The Board_init( ) API is from the TI Processor SDK board library. PRU-ICSS EtherCAT is built on top of the Processor SDK from TI, which is a unified software platform for TI embedded processors, providing easy setup and fast out-of-the-box access to benchmarks and demos. The TI PRU-ICSS industrial software package supports EtherCAT, PROFINET, PROFIBUS, EtherNet/IP, HSR/PRP, and more.

Board_phyReset() is implemented as follows:

```
/* drive phy1 (& phy2) reset to high; Both PHY resets are sourced from one GPIO */
    GPIOModuleEnable(PhyResetInfo[1].baseAddr);
    GPIODirModeSet(PhyResetInfo[1].baseAddr, PhyResetInfo[1].pin, GPIO_DIR_OUTPUT);
    GPIOPinWrite(PhyResetInfo[1].baseAddr, PhyResetInfo[1].pin, GPIO_PIN_HIGH);
    delay_us(20);
    GPIOPinWrite(PhyResetInfo[1].baseAddr, PhyResetInfo[1].pin, GPIO_PIN_LOW);
/* T1 - RESET pulse width, min. 10us, see DP83822 data sheet */
    delay_us(20);
    GPIOPinWrite(PhyResetInfo[1].baseAddr, PhyResetInfo[1].pin, GPIO_PIN_HIGH);
```

Where PhyResetInfo[1] has the GPIO pin configured for the reset.

```
/* phy1 reset - drive high */
    PhyResetInfo[1].pin = 13;
    PhyResetInfo[1].baseAddr = SOC_GPIO_1_REGS;
```

Note

PHY can also be reset through the 0x001F PHY Reset Control Register (PHYRCR), see Table 2-1.

Table 2-1. PHY Reset Control Register (PHYRCR)

Bit	Field	Type	Reset	Description
15	Software Reset	RW. SC	0	Software Reset: • 1 = Reset PHY • 0 = Normal Operation This bit is self cleared and has the same effect as the hardware reset pin.
14	Digital Restart	RW, SC	0	Digital Restart: • 1 = Reset PHY • 0 = Normal Operation This bit is self cleared and resets all PHY digital circuitry, except the registers.
13:0	Reserved	RW	0	Reserved

In the DP83822 device, the PHY address pins PHY_AD[4:1] are multiplexed with RX_D[3:0] and pulled down. PHY_AD[0] (LSB of the address) is multiplexed with COL on pin 29 and pulled up. If no external pullup or pulldown is present, the default PHY address is 0x01. The DP83822 device can be configured for any of the 32 possible PHY addresses available through bootstrap configuration. The PHY address is latched into the device at power up or hardware reset, and is hard coded in the software.

```
#define AM335X_ICSS1_PORT1_PHY_ADDR 1
#ifndef iceAMIC11x
#define AM335X_ICSS1_PORT2_PHY_ADDR 3
#else
#define AM335X_ICSS1_PORT2_PHY_ADDR 13
#endif
```

The PHY addresses are written into shared data RAM with the offset defined as follows, to notify PRU firmware.

```
#define ESC_ADDR_TI_PORT0_PHYADDR   0xE08
#define ESC_ADDR_TI_PORT1_PHYADDR   0xE09
    //Indicate PHY address to firmware via vendor specific registers
bsp_write_byte(pruIcssHandle, pmdio_params->addr0, ESC_ADDR_TI_PORT0_PHYADDR);
bsp_write_byte(pruIcssHandle, pmdio_params->addr1, ESC_ADDR_TI_PORT1_PHYADDR);
```

For the AMIC110, the shared data RAM address is 0x4A31_0000.

When the MDIO fails to access PHY_ID1_REG (register 0x02) with host API, for example, Board_getPhyIdentifyStat(), it usually implies that the PHY is not reset correctly or the PHY address is not configured correctly. The reset method may vary between TI and customer boards as a result of using different GPIOs.

3 PHY Speed, Duplex, and More

After the PHY is reset, it can be configured using the MDIO for the desired operation mode. The MDIO within the PRU-ICSS in AMIC110 implements the 802.3 serial management interface (SMI) to interrogate and control two Ethernet PHYs simultaneously using a shared 2-wire bus. The SMI in the DP83822 device, compatible with IEEE 802.3 clause 22 and clause 45, provides access to its internal register space including the standard register set 0 to 31, the extended register set using the Register Control Register (REGCR, address 0x000D), and the Data Register (ADDAR, address 0x000E), for status information and configuration.

The Ethernet auto-negotiation provides a mechanism for exchanging configuration information, such as speed and duplex, between the two ends of a link segment through burst pulses. Auto-negotiation ensures that the highest performance protocol is selected based on the advertised abilities of the link partner and local device. Auto-negotiation can be enabled or disabled in hardware using the AN_EN bootstrap, or by register configuration using bit [12] in the Basic Mode Control Register (BMCR, address 0x0000).

In the PRU-ICSS EtherCAT software, the call flow of PHY initialization using the MDIO is as follows:

```
Phy_reset( )
|
Task_create(task1,&taskParams, NULL);
|
BIOS_start();
```

Where MDIO and PHY initialization is performed in task1:

```
task1() -> Hw_init( ) --> bsp_init( ) -> bsp_pruss_mdio_init( ) -> bsp_ethphy_init( )
```

Hw_init( ) – EtherCAT Slave Interface initialization

bsp_init() – PRU initialization

The MDIO is initialized with bsp_pruss_mdio_init( ) using the CSL function, CSL_MDIO_init(), where it takes input and output clock as arguments.

```
/** ============================================================================
 *   @n@b CSL_MDIO_init
 *
 * \brief   This API initializes the MDIO peripheral. This enables the MDIO state
 *          machine, uses standard pre-amble and set the clock divider value.
 *
 * \param   baseAddr       Base Address of the MDIO module.
 * \param   mdioInputFreq  The clock input to the MDIO module.
 * \param   mdioOutputFreq The clock output required on the MDIO bus.
 * ============================================================================
 */
static inline void CSL_MDIO_init(uint32_t baseAddr,
              uint32_t mdioInputFreq,
              uint32_t mdioOutputFreq);
```

The initialization of the PRU-ICSS, which is needed for the MDIO PRU firmware to communicate with PHYs, must be performed after powering on the PRU-ICSS domain and before the PRU firmware is loaded and executed on both PRUs.

When the MDIO is initialized, it can access the PHY with the CSL functions CSL_MDIO_phyRegRead() and CSL_MDIO_phyRegWrite() to configure the following:

- Connection speed, duplex, and auto-negotiation
- Auto-MDIX, which determines if a straight or crossover cable is used to connect to the link partner.
- Extended full-duplex mode. In extended full-duplex mode, when the PHY is set to auto-negotiation or Force 100Base-TX, and the link partner is operated in Force 100Base-TX, the link is always full duplex. When disabled, the decision to work in full-duplex or half-duplex mode follows IEEE specification – half duplex.
- Detection of transmit error in odd-nibble boundary, which extends TX_EN by one additional TX_CLK cycle and behaves as if TX_ER were asserted during that additional cycle.

- Additional features like odd nibble insertion and fast link down detection, described as follows:

```
/** ============================================================================
 *   @n@b CSL_MDIO_phyRegRead
 *
 * \param   baseAddr Base Address of the MDIO module.
 * \param   phyAddr  PHY Address.
 * \param   regNum   Register Number to be read.
 * \param   pData    Pointer where the read value shall be written.
 *
 * \retval  TRUE     Read is successful.
 * \retval  FALSE    Read is not acknowledged properly.
 * ============================================================================
 */
static inline Uint32  CSL_MDIO_phyRegRead(uint32_t baseAddr,
                    Uint32 phyAddr,
                    Uint32 regNum,
                    Uint16 *pData)
/** ============================================================================
 *   @n@b CSL_MDIO_phyRegWrite
 *
 * \brief   This API writes a PHY register using MDIO.
 *
 * \param   baseAddr      Base Address of the MDIO module.
 * \param   phyAddr       PHY Address.
 * \param   regNum        Register Number to be written.
 * \param   wrVal         Value to be written.
 * ============================================================================
 */
static inline void CSL_MDIO_phyRegWrite(uint32_t baseAddr,
                     uint32_t phyAddr,
                     uint32_t regNum,
                     uint16_t wrVal);
```

Most MDIO PHY configuration functions are integrated in the board library of the Processor SDK.

For example, to enable the auto-MDIX bit in the 0x0019 PHY Control Register (PHYCR), see the following:

```
Board_enablePhyAutoMDIX((((PRUICSS_HwAttrs *)(pruIcssHandle->hwAttrs))->prussMiiMdioRegBase),
pmdio_params->addr0);
```

To enable the extended full-duplex mode bit in the 0x000A Control Register 2 (CR2), see the following:

```
Board_phyExtFDEnable((((PRUICSS_HwAttrs *)(pruIcssHandle->hwAttrs))->prussMiiMdioRegBase), phy0addr);
```

To enable the odd nibble insertion bit in the 0x000A Control Register 2 (CR2), see the following:

```
Board_phyODDNibbleDetEnable((((PRUICSS_HwAttrs *)(pruIcssHandle->hwAttrs))->prussMiiMdioRegBase),
phy0addr);
```

Users can also directly call the CSL functions to manage the PHY, for example to turn off RMII mode and select MII mode, see the following:

```
phyregval = 0;
CSL_MDIO_phyRegWrite((((PRUICSS_HwAttrs *)(pruIcssHandle->hwAttrs))->prussMiiMdioRegBase),
pmdio_params->addr0, TLKPHY_RCSR_REG, phyregval);
```

Reading PHY and writing to PHY through the USERACCESS0/1 register using memory browser or registers window in Code Composer Studio™ is a effective field-debug feature to troubleshoot PHY issues (see Table 3-1).

Table 3-1. MDIOUSERACCESS0 Register Field Descriptions

Bit	Field	Type	Reset	Description
31	GO	R/W/S	0	Go. Writing 1 to this bit causes the MDIO state machine to perform an MDIO access when it is convenient (this is not an instantaneous process). Writing 0 to this bit has no effect. This bit is write-able only if the MDIO state machine is enabled. This bit self clears when the requested access completes. Any writes to the MDIOUSERACCESS0 register are blocked when the GO bit is 1. If byte access is being used, then the GO bit should be written last.
30	WRITE	R/W	0	Write enable. Setting this bit to 1 causes the MDIO transaction to be a register write, otherwise it is a register read.
29	ACK	R/W	0	Acknowledge. This bit is set if the PHY acknowledged the read transaction.
28:26	Reserved	R	0	Reserved
25:21	REGADR	R/W	0	Register address. Specifies the PHY register to be accessed for this transaction.
20:16	PHYADR	R/W	0	PHY address. Specifies the PHY to be accessed for this transaction.
15:0	DATA	R/W	0	User data. The data value read from or to be written to the specified PHY register.

To read PHY in the memory browser or registers window, follow these steps:

1. Ensure the GO bit in the MDIO user access register (MDIOUSERACCESSn) is cleared.
2. Write to the GO, REGADR, and PHYADR bits in MDIOUSERACCESSn corresponding to the PHY and PHY register to be read.
3. The read data value is available in the DATA bits in MDIOUSERACCESSn after the module completes the read operation on the serial bus. Completion of the read operation can be determined by polling the GO and ACK bits in MDIOUSERACCESSn. When the GO bit clears, the ACK bit is set on a successful read.

To write PHY in the memory browser or registers window, follow these steps:

1. Ensure the GO bit in the MDIO user access register (MDIOUSERACCESSn) is cleared.
2. Write to the GO, WRITE, REGADR, PHYADR, and DATA bits in MDIOUSERACCESSn corresponding to the PHY and PHY register you want to write.
3. The write operation to the PHY is scheduled and completed by the MDIO module. Completion of the write operation can be determined by polling the GO bit in MDIOUSERACCESSn for a 0.

To verify the PHY is successfully connected, users can look into the MDIOALIVE and MDIOLINK registers at the base address offset, 0x04 and 0x08.

Each of the 32 bits of the MDIOALIVE register are set if the PHY acknowledged the most recent access to the PHY, with the address corresponding to the register bit number. The bit is reset if the PHY fails to acknowledge the access. Both the user and polling accesses to a PHY cause the corresponding alive bit to be updated. The alive bits are only meant to be used to indicate the presence of a PHY, or lack thereof, with the corresponding address. Writing 1 to any bit clears it; writing 0 has no effect.

The MDIOLINK register is updated after a read of the Status Register of a PHY. The bit is set if the link of PHY with the corresponding address is up and the PHY acknowledges the read transaction. The bit is reset if the PHY indicates it does not have the link or fails to acknowledge the read transaction. Writes to the register have no effect. In addition, the status of the two PHYs specified in the MDIOUSERPHYSELn registers can be determined using the MLINK input pins. This selection is determined by the LINKSEL bit in the MDIOUSERPHYSELn register.

Figure 3-1 shows a snapshot from the Code Composer Studio PRU register window captured on the AMIC110 ICE. This snapshot indicates two PHYs are present (MDIOALIVE = 0x0002002) and the PHY with address 0x1 link is up (MDIOLINK = 0x00000002).

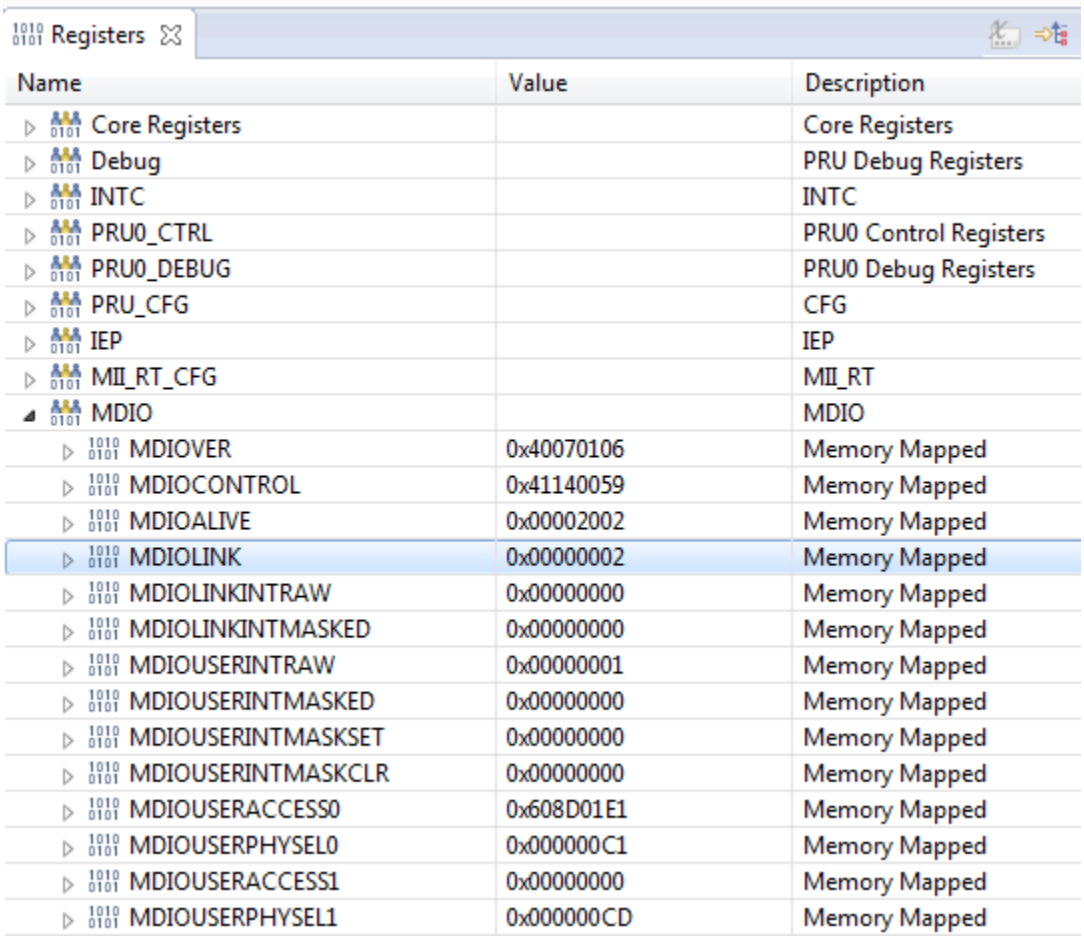

Figure 3-1. MDIO Registers in Code Composer Studio

In the EtherCAT implementation from TI, multiple PRU-ICSS MDIO host APIs are provided and can be used to read/write PHY registers and query the PHY link status. For instance, see the following:

```
Int16 bsp_pruss_mdio_phy_read (Uint8 phyaddr, Uint8 regoffset, Uint16 *regval);
```

Parameters:

- phyaddr – Selects the PHY to read using the PHY address
- regoffset – Register offset in the PHY to read
- regval – Pointer to the variable to hold the register value read
- Return value
  - 0: Success
  - –1: MDIO access error

```
Int16 bsp_pruss_mdio_phy_write (Uint8 phyaddr, Uint8 regoffset, Uint16 regval);
```

Parameters:

- phyaddr – Selects the PHY to write using the PHY address
- regoffset – Register offset in the PHY to read
- regval – Value to write to the PHY register
- Return value
  - 0: Success
  - –1: MDIO access error

```
Uint32 bsp_pruss_mdio_phy_link_state (Uint8 phyaddr);
```

Parameters:

- phyaddr – Select the PHY for link status
- Return value
  - 0: Link Down
  - Otherwise: Link Up

Note

The bsp_pruss_mdio_phy_link_state( ) API considers MII_LINK signal polarity differences and is recommended when TIESC_MDIO_RX_LINK_ENABLE is enabled for enhanced link detection.

4 Enhanced Link Detection

the MDIO internal state machine can be used to update the link status in the MDIOLINK Register after a read of the PHY Status Register. In addition, the link status can also be determined by using the MLINK input pin, depending on the LINKSEL bit in the MDIOUSERPHYSELn register. The MDIO state machine-based detection is slow due to a serial link for messaging from the MDIO controller to the PHYs, which typically takes from 200 to 250 µs. MLINK/mii_rxlink detection occurs as fast as the PHY can toggle the link and typically within 10 µs, provided by the advanced link-down capability of the DP83822 device, which uses criteria such as RX error count, MLT3 error count, low SNR threashold, and signal and energy loss to quickly detect link loss.

Fast link loss detection is mandatory to support cable redundancy, which requires 10 to 15 µs of link loss detection time. This duration is two to three minimum-sized EtherCAT frames, which is the maximum that can be dropped during a link failure or cable break event.

The link status determination select (LINKSEL) bit in the MDIOUSERPHYSELn register determines the link status using the MLINK pin by setting it to 1. The default value is 0, which implies that the link status is determined by the MDIO state machine.

The pr1_mii0_rxlink and pr1_mii1_rxlink pins of PRU-ICSS, which connect to the PHY LED_LINK or LED_SPEED pins, respectively, are connected as the MLINK signal to MDIO. Depending on the PHY strap settings, link polarity varies on different boards and must be adjusted for each board by reading the MDIOLINK

register. When connecting to the LED_LINK pin, set the LED_LINK control mode as *LINK_OK* instead of *RX/TX Activity* to prevent link detection failure due to intermittent RX/TX traffic (see Table 4-1).

Table 4-1. LEDs Configuration Register 1 (LEDCFG1)

Bit	Field	Type	Reset	Description
15:12	Reserved	R/O	0	Reserved
11:8	LED_1 Control	R/W	0101	LED_1 Control: Selects the source for LED_1. • 0000 = LINK OK • 0001 = RX/TX Activity • 0010 = TX Activity • 0011 = RX Activity • 0100 = Collision • 0101 = Speed, high for 100Base-TX • 0110 = Speed, high for 10Base-Te • 0111 = Full-duplex • 1000 = LINK OK, blink on TX/RX Activity • 1001 = Active stretch signal • 1010 = MII LINK (100BT+FD) • 1011 = LPI mode • 1100 = TX/RX MII error • 1101 = Link Lost • 1110 = Blink for PRBS errors, LED remains on until the BMCR register (address 0x0001) is cleared. • 1111 = Reserved Link Lost, LED remains on until the BMCR register (address 0x0001) is read. Blink for PRBS Errors, LED remains on for a single error and remains on until the BICSR1 register (address 0x001B) is cleared.
7 :4	LED_3 Control (RX_D3)	R/W	0101	LED_3 Control: Selects the source for RX_D3. Reference bits [11:8] for a list of sources.
3:0	Reserved	R/W	0001	Reserved

When the enhanced link detection pin is connected to PHY_LED1 on the AMIC110 ICE and the PHY configuration is set so that LED1 blinks on activity, whenever there is activity it appears to the MDIO that the link is turning on and off, which results in enhanced link detection failure. This is not the correct LED1 activity configuration for enhanced link detection. Instead, LED1 should be configured to indicate LINK OK.

To update the LED configuration for the source – LINK OK instead of RX/TX Activity, use the API:

```
Board_phyLED1Config((((PRUICSS_HwAttrs *)(pruIcssHandle->hwAttrs))->prussMiiMdioRegBase), phy0addr,
LED_CFG_MODE0);
Board_phyLED1Config((((PRUICSS_HwAttrs *)(pruIcssHandle->hwAttrs))->prussMiiMdioRegBase), phy1addr,
LED_CFG_MODE0);
```

Where LED_CFG_MODE0 represents 0000 = LINK OK in LED_1 Control bits.

To enable the enhanced link detection, use the Board_phyFastLinkDownDetEnable( ) API to configure PHY control register 3 with any combination of the following macros as its argument val.

```
#define FAST_LINKDOWN_SIGENERGY        1u
#define FAST_LINKDOWN_LOWSNR           (1u<<1)
#define FAST_LINKDOWN_MLT3ERR          (1u<<2)
#define FAST_LINKDOWN_RXERR            (1u<<3)
void Board_phyFastLinkDownDetEnable(uint32_t mdioBaseAddress, uint32_t phyNum, uint8_t val);
```

Enhanced link detection can also be disabled during MDIO initialization using the DISABLE macro instead of ENABLE when the PHY does not have the LED_LINK or LED_SPEED signal, and the pr1_mii0_rxlink and pr1_mii1_rxlink signals can be left floating.

```
Int16 bsp_pruss_mdio_init (t_mdio_params *pmdio_params);
```

Parameters:

- pmdio_params – Pointer to parameter structure of PRU-ICSS MDIO initialization
- pmdio_params->clkdiv – MDIO clkdiv
- pmdio_params->addr0 – Address of the PHY hooked to PRU-ICSS MII0
- pmdio_params->addr1 – Address of the PHY hooked to PRU-ICSS MII1
- pmdio_params->link0pol – LINK_MII signal polarity of the PHY hooked to PRU-ICSS MII0
- pmdio_params->link1pol – LINK_MII signal polarity of the PHY hooked to PRU-ICSS MII1
- mdio_params->enhancedlink_enable – Enable enhanced link detection

Struct for MDIO initialization parameters:

```
typedef struct {
 Uint16 clkdiv;
 Uint8 addr0;
 Uint8 addr1;
 Uint8 link0pol; //1: Active Low 0: Active High
 Uint8 link1pol; //1: Active Low 0: Active High
 Uint8 enhancedlink_enable;
} t_mdio_params;
```

Link polarity is of important for the EtherCAT if enhanced link detection is enabled. The setting of link polarity parameters, link0pol and link1pol, is determined by the MDIOLINK Register when enhanced link detection is enabled. To check the link polarity:

1. Use the MDIO link register.
2. insert the Ethernet cable from the PC to the DUT port.
3. observe any change, then make adjustments in the link polarity field in bsp_pruss_mdio_init( ) accordingly.

For instance, to set polarity HIGH for both the PHYs, see the following:

```
    #define TIESC_LINK0_POL   TIESC_LINK_POL_ACTIVE_HIGH
    #define TIESC_LINK1_POL   TIESC_LINK_POL_ACTIVE_HIGH
    PRUICSS_V1_Object *object;
    object = (PRUICSS_V1_Object *)pruIcssHandle->object;
mdioParamsInit.addr0 = Board_getPhyAddress(object->instance, 1);
    mdioParamsInit.addr1 = Board_getPhyAddress(object->instance, 2);
    mdioParamsInit.enhancedlink_enable = TIESC_MDIO_RX_LINK_ENABLE;
    if(TIESC_MDIO_RX_LINK_ENABLE == mdioParamsInit.enhancedlink_enable)
    {
        //Enhanced link detection enabled
        mdioParamsInit.link0pol = TIESC_LINK0_POL;
        mdioParamsInit.link1pol = TIESC_LINK1_POL;
    }
    else
    {
        //Enhanced link detection disabled
        mdioParamsInit.link0pol = TIESC_LINK_POL_ACTIVE_HIGH;
        mdioParamsInit.link1pol = TIESC_LINK_POL_ACTIVE_HIGH;
    }
    bsp_pruss_mdio_init(pruIcssHandle, &mdioParamsInit);
```

The TI EtherCAT implementation also provides a few PHY-related registers that can be helpful when debugging the PHY configuration, for example, see Table 4-2.

Table 4-2. PHY Registers in EtherCAT

Bit	Field	Permissions		Register Offset	Reset	Description
		ECAT	PDI			
31:0	PRU MII RX LINK polarity	R/-	R/W	0x0E0C	0b	Link LED signal polarity PHY address N (bit N)
7:0	Port0 PHY address	R/-	R/W	0x0E08	0	Specifies the PHY address of PHY connected to the physical port 0 to PRU firmware
7:0	Port1 PHY address	R/-	R/W	0x0E09	0	Specifies the PHY address of PHY connected to the physical port 1 to PRU firmware

The PHY link status and port status is reflected in the standard EtherCAT data link(DL) status register (see Table 4-3).

Table 4-3. Data Link Status Register in EtherCAT

Bit	Field	Permissions		Register Offset	Reset	Description
		ECAT	PDI			
15		R/-	R/-			Communication on port 3 – not available in TI ESC
14		R/-	R/-			Loop port 3 – not available in TI ESC
13		R/-	R/-			Communication on port 2 – not available in TI ESC
12		R/-	R/-			Loop port 2
11		R/-	R/-			Communication on port 1
10		R/-	R/-			Loop port 1
9		R/-	R/-			Communication on port 0 • 0 = no stable communication • 1 = communication established
8		R/-	R/-			Loop port 0 • 0 = opened • 1 = closed
7		R/-	R/-			Physical link on port 3 – not available in TI ESC
6		R/-	R/-			Physical link on port 2 – not available in TI ESC
5		R/-	R/-			Physical link on port 1
4		R/-	R/-			Physical link on port 0 • 0 = no link • 1 = link detected
3		R/-	R/-			Reserved
2		R/-	R/-			Enhanced link detection • 0 = deactivated on all ports • 1 = activated on at least one port
1		R/-	R/-			• 0 = PDI watchdog expired • 1 = PDI watchdog reloaded
0		R/-	R/-			• 0 = EEPROM not loaded, PDI not operational (no access to PD RAM) • 1 = EEPROM loaded correctly, PDI operational (access to PD RAM)

Note

The EtherCAT definition of the port OPEN/CLOSE definition is if there is no link, the port cannot be OPEN (allows traffic) and must be CLOSE (no traffic).

5 Add PHYs in Processor SDK

Several PHY devices have been included in the Platform Development Kit (PDK) of the Processor SDK from TI. The PDK is a single, scalable, software driver package that offers streamlined development across different processors and platforms. The PDK package contains Device Abstraction Layer libraries and peripheral/board level sample/demo examples that demonstrate the capabilities of the peripherals on platforms for development, deployment, and execution of applications.

The PHY configurations in the PDK are supported in the board library as shown in the following software structure (see Figure 5-1).

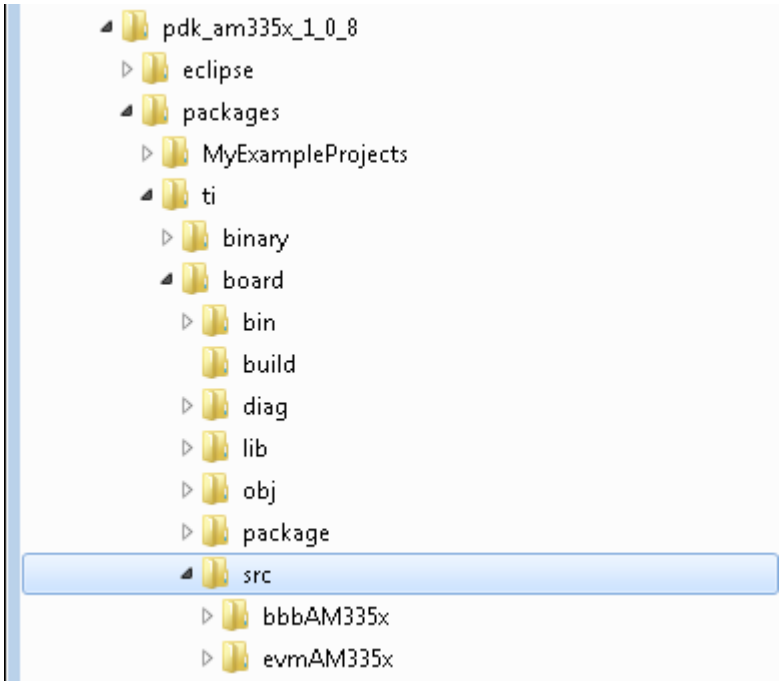

Figure 5-1. Board Library in PDK

The board library provides a high-level abstraction for the following:

- Pinmux
- clock tree
- Configuration
- Board devices
- Memory map
- Board level mux controls
- Board power
- I/O expanders

The board library APIs perform auto-detection of board type and definition, and then develop the abstraction based on the board configuration and the onboard device descriptions.

In some platforms, the configuration functions of the PHY are supported in board\src\<BOARD>\device\enet_phy.c, while other platforms such as the AMIC110 ICE and its PHY functions are supported in src\<BOARD>\<PROCESSOR>_ethernet_config.c, as shown in Figure 5-2.

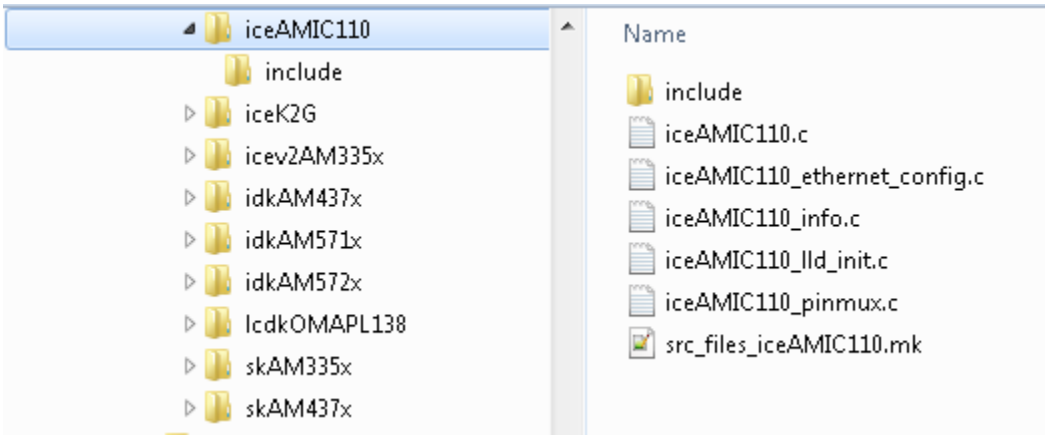

Figure 5-2. AMIC110 ICE Board Library

To add a new board and PHY (see Figure 5-3), migrate the following primary items from one platform:

- Pinmux settings
- Clock and timer configuration changes.
- Memory configuration
- Interrupt changes
- I/O changes (MII, MDIO, PHY, GPIO, and UART)
- Board component substitutions
- Board initialization (drawing from all of the previous)

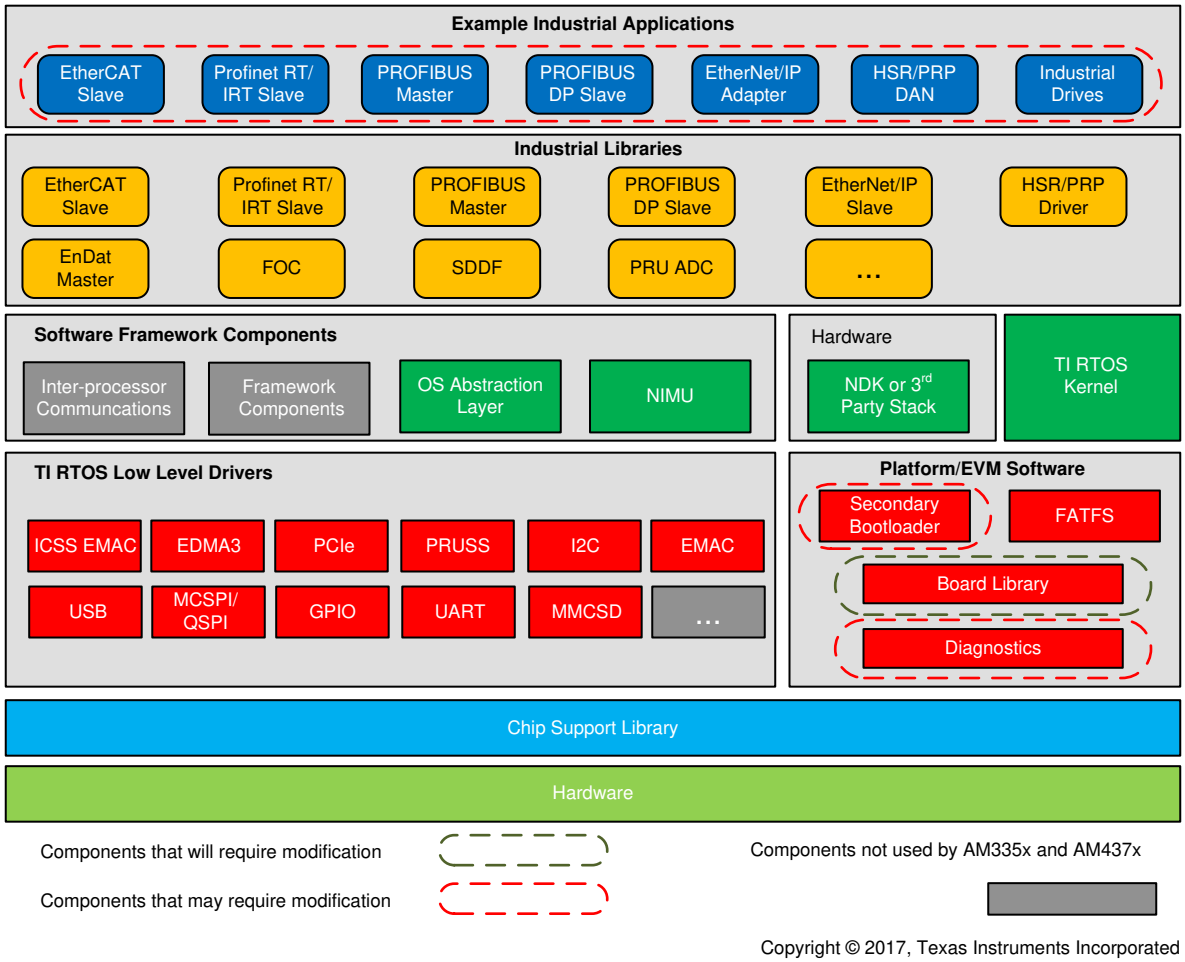

Figure 5-3. Adding New Board and PHY

6 Conclusion

It is a challenge to bring up Ethernet on a custom board, so proper PHY configuration using MDIO is an essential step in the path. This application report summarizes the flow of Ethernet PHY configuration using MDIO including PHY selection and connection, reset and address, speed/duplex/auto-MDIX, and also explains industrial specific features – enhanced link detection, and lastly discusses how to add new PHYs in the unified software Processor SDK from TI.

7 References

1. Texas Instruments, DP83865 and DP83864 Gigabit Physical Layer Device Trouble Shooting Guide, application report
2. Texas Instruments, DP83867 Troubleshooting Guide, application report
3. Texas Instruments, DP83822 Hardware Rollover, application report
4. Beckhoff, PHY Selection Guide, application note
5. Texas Instruments, DP83822 Robust, Low Power 10/100 Mbps Ethernet Physical Layer Transceiver, data sheet
6. Texas Instruments, PRU-ICSS Industrial Software for Sitara™ Processors
7. Texas Instruments, PRU ICSS EtherCAT Slave Controller Register List, TI Wiki
8. Texas Instruments, Industrial SDK EMAC Porting Guide, TI Wiki

8 Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Revision * (December 2017) to Revision A (May 2021)	Page
• Updated the numbering format for tables, figures and cross-references throughout the document	2

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (https:www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265