F28P65X Launchpad REVA Bug

The first production run of Revision A F28P65X launchpads, which were all shipped in 2023, will not work out of the box when directly following the EtherCAT SubordinateDevice Controller Software User's Guide that is provided in C2000WARE. This is because the boards were designed to use REV1 of the DP83826E PHY, but these early REVA boards were mistakenly manufactured with REV0 of DP83826E. If users find that their board is affected by this issue, and they do not want to use the software workaround provided in this document, they can replace the REV0 PHY with a REV1 PHY to get the expected behavior. All DP83826E PHYs shipped by TI after 2023 should be REV1, and all LAUNCHXL-F28P65X boards shipped by TI after 2023 should have REV1 PHYs.

The resources used for this workaround can be found attached to the E2E FAQ at this link: https://e2e.ti.com/support/microcontrollers/c2000-microcontrollers-group/c2000/f/c2000-microcontrollers-forum/1325322/faq-launchxl-f28p65x-how-do-i-fix-the-ethercat-issue-where-the-launchpad-cannot-be-scanned-in-twincat

SOFTWARE WORKAROUND

1. The first step is to program the EEPROM that is used by the EtherCAT Subsystem. You will only need to do this once. For this step, please use the program_i2c_ESC_eeprom_F28P65x.zip project that is attached to the E2E FAQ linked above.
    a. Please unzip this project, then import it into your CCS workspace. You can simply click the debug button to build and load the project to CPU1 of your device.
        i. If you want to see the data that is written to and then read from the PHY, you can add a watch expression for `RX_MsgBuffer`
    b. This program will load only the minimum needed bytes into the EEPROM so that debug access can then be enabled to the F28P65x ESC registers. This is needed so that the device can read and write to the PHY.
2. The next step is to make sure that you can read and write the PHY correctly. After this step, you should be able to see the device in TwinCAT, and you will be able to write the EEPROM through TwinCAT by following the instructions in the EtherCAT User's Guide.
    a. Please import the F28p65x_cpu1_pdi_hal_test_app example from [YOUR_C2000WARE]/libraries/communications/Ethercat/f28p65x/examples into your CCS workspace
    b. Download the ethercat_slave_cpu1_hal_PHY_check.c file from the linked E2E FAQ. Paste it into your F28p65x_cpu1_pdi_hal_test_app project.
    c. Right click on the original ethercat_subdevice_cpu1_hal.c file and exclude it from build

d. If you are using C2000WARE 5.01, select the 'FLASH' build configuration since there is a bug in the LAUNCHXL_FLASH build configuration. You will need to define '_LAUNCHXL_F28P65X' at the top of your ethercat_slave_cpu1_hal_PHY_check.c file, or add it to your predefined symbols for your project.
e. Build and load the program to CPU1. There will be a few 'ESTOP0' commands, click resume until the code runs without stopping
f. If you follow the steps in section 4 of the EtherCAT User's Guide for 'scanning for EtherCAT Devices via TwinCAT' you should be able to see the box for the F28P65x device
    i. You can now write to the EEPROM

3. You will need to add the below code snippet to the initialization all of your future EtherCAT projects (assuming CPU1 ownership).
    a. ADD TO TOP OF FILE:

```
//
//ADDED INCLUDES
//
#include "escss.h"
#include "device.h"

//
// Definitions for reading / writing to DP838xx PHY devices
//
#define ESC_MII_CTRL_STATUS_OFFSET        0x0510 // 0x288 - Low
#define ESC_PHY_ADDRESS_OFFSET            0x0512 // 0x289 - Low
#define ESC_PHY_REG_ADDRESS_OFFSET        0x0513 // 0x289 - High
#define ESC_PHY_DATA_OFFSET               0x0514 // 0x28A - Low
#define ESC_MII_ECAT_ACCESS_OFFSET        0x0516 // 0x28B - Low
#define ESC_MII_PDI_ACCESS_OFFSET         0x0517 // 0x28B – High

//
//  FUNCTIONS NEEDED FOR REV0 SILICON
//
//

//
// Function for reading from an ESC PHY register using CPU1 -
//      Assumes Word and address are 16-bit values
//
uint16_t mii_readPhyRegister(uint16_t address, uint16_t target_phy_address)
{
   // Set PHY address to read/write, (0x0512//0x289 low)
   ESC_writeWordISR((address<<8) | target_phy_address, ESC_PHY_ADDRESS_OFFSET);

   // Write PHY reg, (0x0510//0x288 low)
   ESC_writeWordISR(0x0100,ESC_MII_CTRL_STATUS_OFFSET);
   ESC_writeWordISR(0x0103,ESC_MII_CTRL_STATUS_OFFSET);


   while(ESC_readWordISR(ESC_MII_CTRL_STATUS_OFFSET) != 0x3); // wait till read is completed

   return ESC_readWordISR(ESC_PHY_DATA_OFFSET);
}

//
// Function for writing to an ESC PHY register using CPU1 -
//      Assumes Word and address are 16-bit values
//
void mii_writePhyRegister(uint16_t address, uint16_t target_phy_address, uint16_t data)
{
   ESC_writeWordISR((address<<8) | target_phy_address, ESC_PHY_ADDRESS_OFFSET);
   ESC_writeWordISR(data, ESC_PHY_DATA_OFFSET);
   ESC_writeWordISR(0x203, ESC_MII_CTRL_STATUS_OFFSET);

   //DEVICE_DELAY_US(100);
   while(ESC_readWordISR(ESC_MII_CTRL_STATUS_OFFSET) != 0x3); // wait till write is
completed and successful
}


//
//  Function for reading extended PHY Register, different from reading first 0x1F address
//
//

void mii_writePhyExtendedRegister(uint16_t address, uint16_t target_phy_address, uint16_t
data)
{
   mii_writePhyRegister(0x0D,target_phy_address,0x001F);
   mii_writePhyRegister(0x0E,target_phy_address,address);
   mii_writePhyRegister(0x0D,target_phy_address,0x401F);
   mii_writePhyRegister(0x0E,target_phy_address,data);

}


//
// Function for reading from an Extended ESC PHY register using CPU1 -
//      Assumes Word and address are 16-bit values
//
uint16_t mii_readPhyExtendedRegister(uint16_t address, uint16_t target_phy_address)
{
   mii_writePhyRegister(0xD, target_phy_address, 0x1F); //Phy reg 0xD = 0x1F
   mii_writePhyRegister(0xE, target_phy_address, address); //Phy Reg 0xE = *Address of register
outside of 0-1F
   mii_writePhyRegister(0xD, target_phy_address, 0x401F); //Phy reg 0xD = 0x401F

   return(mii_readPhyRegister(0xE, target_phy_address)); // Extended read data
}
```

b. You will need to add the actual register writes sometime after your hardware initialization, possibly inside of your main function:

```
   //
   // works after including "escss.h"
   //


   ESCSS_enableDebugAccess(0x00057E00U);
   ESCSS_initMemory(0x00057E00U);
   ESC_writeWordISR(0x0100,0x0516);

   DEVICE_DELAY_US(100);
   mii_writePhyRegister(0x19, 0x00 , 0x8020);
   mii_writePhyRegister(0x18, 0x00 , 0x0080);
   mii_writePhyExtendedRegister(0x460, 0x00, 0x0005);  //CHANGE LED1 LED2 function to LINK
   mii_writePhyExtendedRegister(0x469, 0x00, 0x0004);  // FLIPPED POLARITY OF LEDs
   mii_writePhyExtendedRegister(0x304, 0x00, 0x0008);
   mii_writePhyRegister(0x04, 0x00, 0x01E1);
   mii_writePhyRegister(0x09, 0x00, 0x0020);
   mii_writePhyRegister(0x00, 0x00, 0x3300);
   mii_writePhyRegister(0x0B, 0x00, 0x0008);
   mii_writePhyRegister(0x17, 0x00, 0x49);
   mii_writePhyRegister(0xA, 0x00, 0x102);

   mii_writePhyRegister(0x1F, 0x00, 0x4000);

   DEVICE_DELAY_US(100);

   mii_writePhyRegister(0x19, 0x01 , 0x8020);
   mii_writePhyRegister(0x18, 0x01 , 0x0080);
   mii_writePhyExtendedRegister(0x460, 0x01, 0x0005);  //CHANGE LED1 LED2 function to LINK
   mii_writePhyExtendedRegister(0x469, 0x01, 0x0004);  // FLIPPED POLARITY OF LEDs
   mii_writePhyExtendedRegister(0x304, 0x01, 0x0008);
   mii_writePhyRegister(0x04, 0x01, 0x01E1);
   mii_writePhyRegister(0x09, 0x01, 0x0020);
   mii_writePhyRegister(0x00, 0x01, 0x3300);
   mii_writePhyRegister(0x0B, 0x01, 0x0008);
   mii_writePhyRegister(0x17, 0x01, 0x49);
   mii_writePhyRegister(0xA, 0x01, 0x102);

   mii_writePhyRegister(0x1F, 0x01, 0x4000);
   DEVICE_DELAY_US(100);
```