Application Note

F28P65 + DP83826 EtherCAT Frame Error Investigation Methods and Application Recommendations

Yunjing Wang, Zane Wei *North West China FAE team*

ABSTRACT

This document was translated from a simplified Chinese source. (ZHCAG56)

F28P65, integrated with Beckhoff's EtherCAT IP core, is TI's next generation of low-cost, high-performance C2000 chip for industrial Ethernet applications. Together with TI's new generation of low-latency 100-Megabit Ethernet PHY, DP83826, it delivers TI's latest cost-effective flagship EtherCAT solution, widely used in industrial real-time Ethernet communication scenarios. In real-world industrial Ethernet communication, a variety of communication issues often arise, such as the EtherCAT master failing to discover slaves, frame errors during communication, or even severe Link Down failures on the physical layer. For engineers who are new to EtherCAT communication, it is hard to investigate and locate issues one by one at the system level. Therefore, this application note begins with the OSI model for EtherCAT communication, describes some of the hardware mechanisms and methods for fault investigation provided in official EtherCAT specifications, and provides solutions for investigating, locating, and addressing common frame error issues in EtherCAT applications.

Table of Contents

Revision History

Version	Date	Author	Notes
1.0	Dec 14th, 2025	Yunjing Wang, Zane Wei	First version

1 Overview of EtherCAT and Communication Model

EtherCAT stands for Ethernet Control and Automation Technology. EtherCAT, a real-time industrial Ethernet technology, was first developed and launched by Beckhoff (Germany) in 2003. It became an international standard in 2007 and later became a national standard in China in 2014. Benefiting from its ultra-high speed, high real-time performance, and open architecture, it has become one of the most widely used industrial Ethernet technologies today.

Figure 1-1 illustrates the EtherCAT communication model, which is divided into the physical layer, the data link layer, and the application layer. The physical layer usually adopts RJ45 interfaces and PHY chips. In some cases, Low-Voltage Differential Signaling (LVDS) interfaces are also used to enable circuit-level protocol transmission. As the lowest layer in the EtherCAT communication model, the EtherCAT physical layer provides physical interfaces for network signal transmission, and also receives data units from the data link layer and decodes them accordingly. The data link layer converts data frames into raw bits for the physical layer and is responsible for packing data from the upper layers into frames, as well as flow control, error correction, and frame retransmission. The most basic function of this layer is to provide reliable data transmission between adjacent nodes through its protocol. The data link layer is implemented by an EtherCAT Slave Controller (ESC). Its primary functions include computing, comparing, and generating frame check sequences, and enabling communication by extracting data from or inserting data into Ethernet frames. The EtherCAT application layer supports multiple protocols such as VoE, FoE, EoE, and CoE.

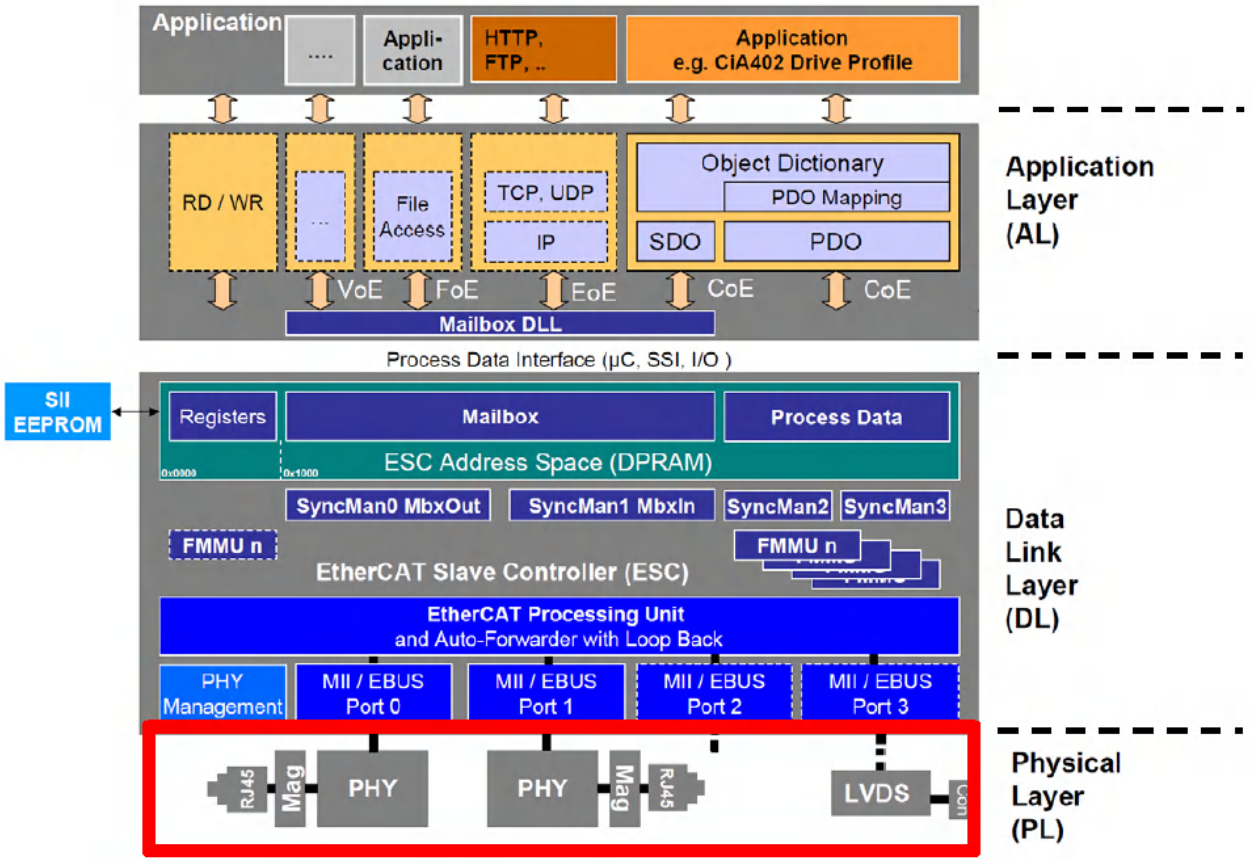

Figure 1-1. EtherCAT Communication Model

In legacy Ethernet communication, each node receives, processes, and forwards data sequentially, while EtherCAT enables simultaneous transmission and processing of EtherCAT data. Each slave node has a Fieldbus Memory Management Unit (FMMU), which is designed to analyze the addresses of all packets passing through the slave node, convert logical addresses into physical addresses, and read out the data in the packet

that is used by the current slave. It also forwards messages to the next slave node. Similarly, data can also be inserted into messages as they pass through, as shown in Figure 1-2.

Each EtherCAT slave extracts user data from, or inserts user data into, the received messages and sends the processed messages to the next EtherCAT slave. After being processed by all slaves in sequence, the messages are then looped back by the last EtherCAT slave and returned to the EtherCAT master control unit as response messages. The entire process is based on the full duplex mode of Ethernet, where messages are sent out over the TX line and return over the RX line. Therefore, any physical topology is logically always a loop in an EtherCAT communication system.

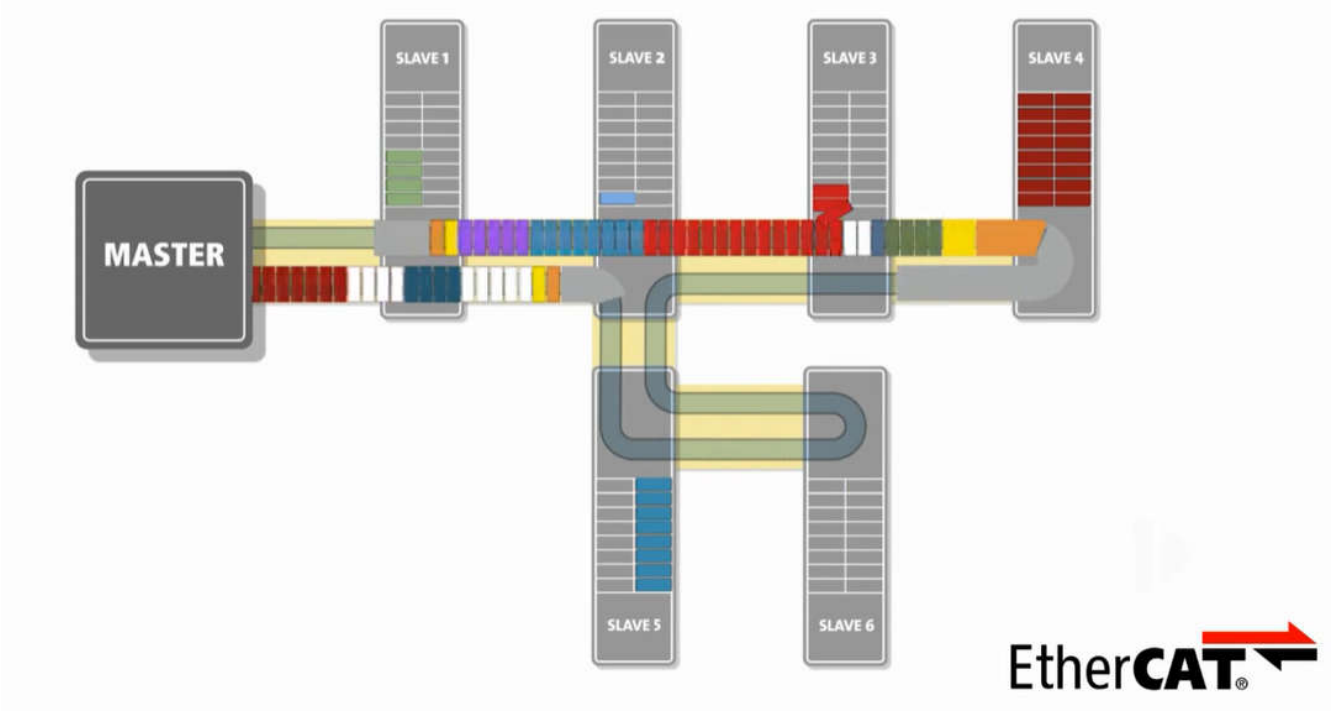

Figure 1-2. EtherCAT Communication Process Diagram

2 EtherCAT Slave Frame Transmission and Diagnostics

2.1 Frame Processing at EtherCAT Slaves

Figure 2-1 illustrates the frame processing process explained in the official EtherCAT specifications. F28P65 supports only two ports; therefore, Port2 and Port3 are disabled. A frame enters through Port0, passes through the Auto-Forwarder into the EtherCAT processing unit, and then proceeds to Port1. This process continues until the last ESC, and finally, the frame is looped back from Port1 to the Master.

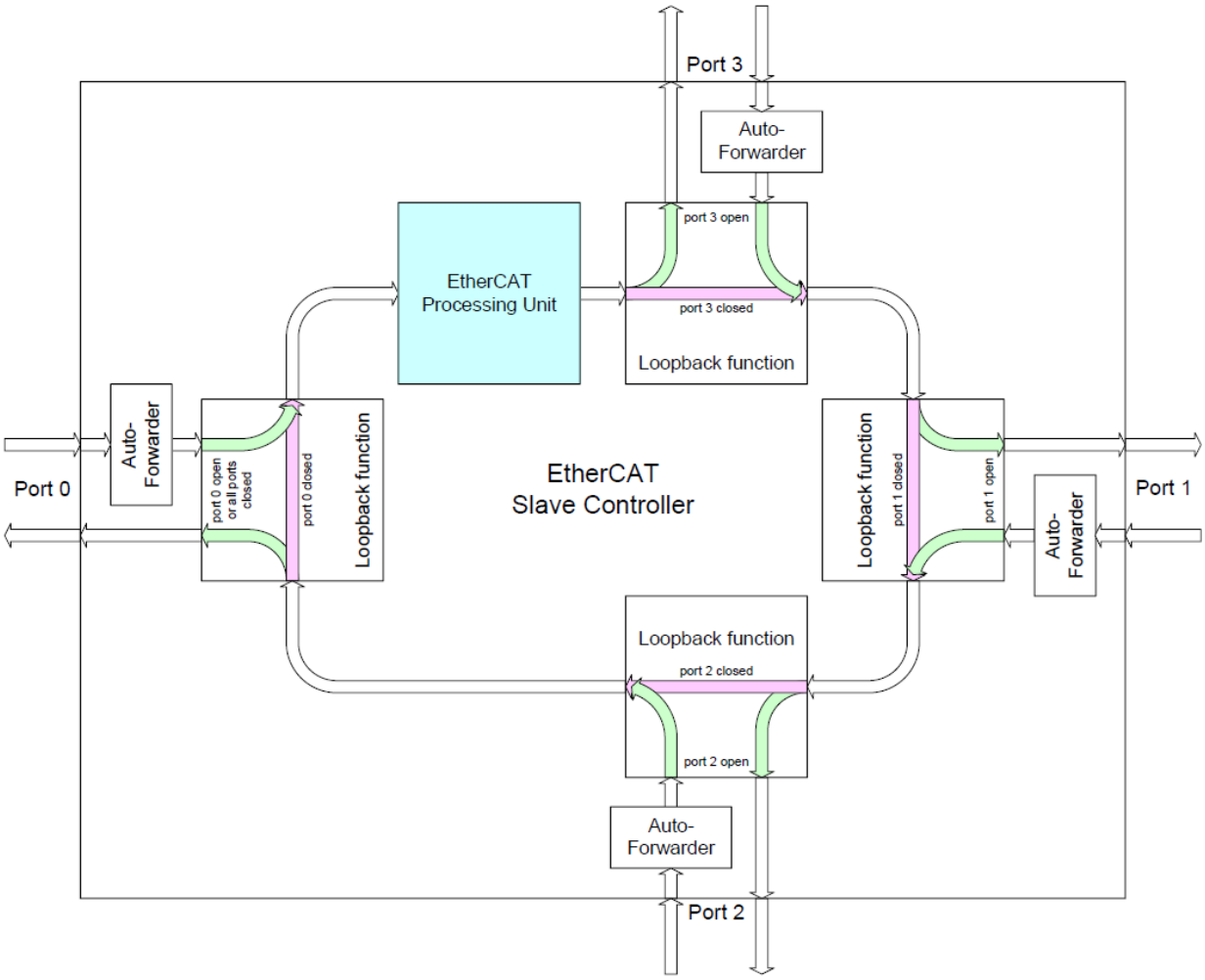

Figure 2-1. EtherCAT Frame Transmission Process

2.2 ESC Diagnostics

Each EtherCAT message ends with a 16-bit Working Counter (WKC), indicating the number of successful memory accesses requested by the command. As shown in Figure 2-2, when a message is sent from the EtherCAT Master, the WKC starts at 0. As each Slave (ESC) accesses the memory requested by the command according to appropriate rules, the WKC count increases accordingly. By comparing the difference between the WKC count received and the expected value, the Master can determine how well an EtherCAT message is being processed. If the WKC count received by the Master does not correspond to the expected value, then one or more ESCs addressed by the message have failed to access the memory. ESC physical layer connection faults, ESC OP state exceptions, frame errors, and some other conditions all possibly cause a WKC count to not match the expected value. The user can further identify the cause and localize the fault using various error counters supported by ESCs.

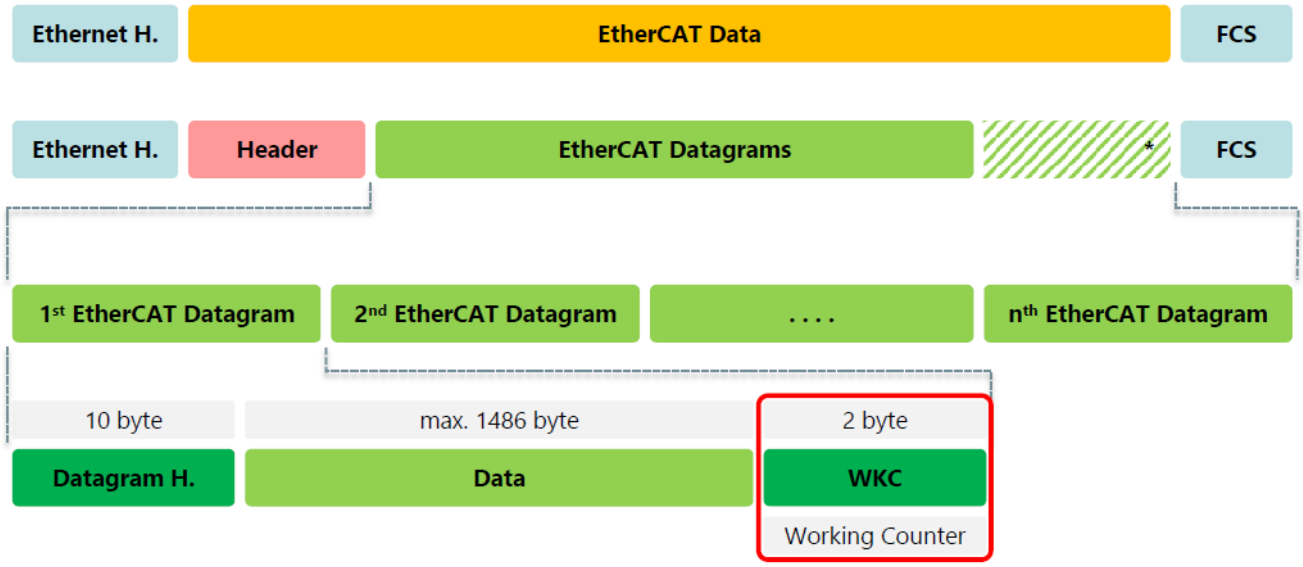

Figure 2-2. EtherCAT Communication Frame Structure With WKC

In the event of any communication issue with an EtherCAT loop, the first thing to do is to verify if the physical layer connection status is abnormal. The user can verify the link status of each port through the ESC 0x0110 register. Because of space constraints, this article focuses on the analysis and localization of Frame Error faults. Fault analysis methods for EtherCAT physical layer link status exceptions will be further discussed with case studies in other articles in this series.

Even if there are no issues with the link between EtherCAT devices, EtherCAT frames can be corrupted in transmission because of EMC interference, ESC faults, and so forth. An ESC provides various frame error-related counters to help the user identify the cause of a frame error and localize the fault. Refer to Figure 2-3 to learn more about specific error counters and their functions.

Error Counter	Register	Description
Port Error Counters	0x0300:0x0307	Errors counted at the Auto-Forwarder (per port):
Invalid Frame Counter	0x0300/2/4/6	Invalid frame initially detected (includes RX Errors)
RX Error Counter	0x0301/3/5/7	Physical layer RX Errors (inside/outside frame): MII: RX_ER EBUS: Manchester violations
Forwarded RX Error Counter	0x0308:0x030B	Invalid frame with marking from previous ESC detected (per port)
ECAT Processing Unit Error Counter	0x030C	Invalid frame passing the EtherCAT Processing Unit (additional checks by processing unit)
PDI Error Counter	0x030D	Physical Errors detected by the PDI. Refer to PDI description in Section III for details.
Lost Link Counter	0x0310:0x0313	Link lost events (per port), counts only if port is in Auto or Auto close mode
Watchdog Counter Process Data	0x0442	Watchdog timeout events
Watchdog Counter PDI	0x0443	Watchdog timeout events

Figure 2-3. ESC Error Counters

The following error registers only start counting when the port is in the Open state, so the port status of the ESC must be considered for fault analysis:

RX Error Counter: RX_ERR refers to any error that the physical layer chip can detect, such as a symbol error in 4B or 5B encoding or a voltage level error for the hardware MLT-3 signal. These errors are reported to the ESC via the RX_ER pin of the PHY on the MII interface and are counted by the ESC RX Error register.

Invalid Frame Counter: Each port automatically performs a cyclic redundancy check when it receives a frame. Both physical layer errors and in-frame errors are considered frame errors.

Forward Error Counter: When the ESC first detects a frame error, this data frame is marked with an incorrect CRC sequence and an additional termination byte at the end when it is sent to subsequent ESCs. This condition will be detected by subsequent ESCs and is counted by the Forward Error Counter.

3 EtherCAT Frame Error Fault Cases and Investigation Methods

Frame errors are common in EtherCAT applications, which can lead to severe consequences such as communication interruption, device disconnection, and device instability. In general, an EtherCAT frame error occurs when the master detects that a message returned by a slave is not as expected, or that there is data corruption or loss during communication, or even when the master fails to discover any slaves because the entire communication link is interrupted because of the failure of a device. Frame error faults are caused by physical layer data errors, MDI clock offset, ESC faults, and so forth. Therefore, fault localization requires investigation using diagnostic information such as physical-layer and ESC fault isolation, EtherCAT Master state machine, error counters, and more.

This section analyzes a frame error fault instance, describing how to determine the frame error, methods for isolating the fault location, and application recommendations for the ESC with F28P65.

3.1 Frame Error Fault Case

Stably reproducible communication faults are usually located quickly. However, for faults that recur randomly, it is more difficult to locate them because of the challenge of reproduction and the random location of the faulty ESC. This fault case involves occasional failures of the Master to discover its Slaves after power-on and initialization during repeated and consecutive power-on reset (POR) tests, which results in the EtherCAT loop not communicating properly, although the physical layer link has been verified to be in good condition beforehand. Seven ESC Slaves underwent repeated and consecutive POR tests, and the Master could not discover any of them when the problem occurred.

However, after ESC5 was removed from the system, the Master successfully discovered ESC1–ESC4, ESC6, and ESC7. Therefore, ESC5 was preliminarily identified as the faulty ESC. Performing an F28P65 chip reset on ESC5 recovered ESC5 from the fault, but performing only an internal ESC reset did not allow ESC5 to recover from the fault. The records of each ESC error register when the issue reoccurred are shown in Relevant Information on ESC Error Registers in Fault State. By analyzing the ESC error registers of each device, it was found that ESC5 first counted a Port fault during transmission.

Table 3-1. Relevant Information on ESC Error Registers in Fault State

Register	ESC1	ESC2	ESC3	ESC4	ESC5	ESC6	ESC7
0x300 PORT0 bit[15:8]-RX ERR bit[7:0]-Invalid frame	0	0	0	0	00FF	00FF	00FF
0x302 PORT1 bit[15:8]-RX ERR bit[7:0]-Invalid frame	00FF	00FF	00FF	00FF	00FF	00FF	0
0x308 Forward Error Port 0-bit[7:0], Port1-bit[15:8]	FF00	FF00	FF00	FF00	FF00	FFFF	00FF
0x30C EPU Err	0	0	0	0	00FF	00FF	00FF
0x30D PDI ERR	0	0	0	0	00FF	00FF	00FF

Based on the internal architecture of the ESC with F28P65 and the looped data transmission process, the error accumulation and propagation process of the entire loop is analyzed as follows.

As shown in Figure 3-1, ESC5 is the source of the fault. ESC5 Port0 first receives an invalid frame. The 0x300 Invalid Frame, 0x308[7:0] Forward Error, and 0x30C EPU errors are accumulated and propagated to ESC6 and ESC7. When the loop is closed at ESC7 Port1 and the data frame begins to loop back, all Invalid Frame (0x302) and Forward Error (0x308[15:8]) on Port1 of all subsequent ESCs, including ESC6 through ESC1, start to accumulate and propagate, except for ESC7 Port1, which is at the end of the loop and in a closed state, thus its Port1 0x302 Invalid Frame error is not accumulated.

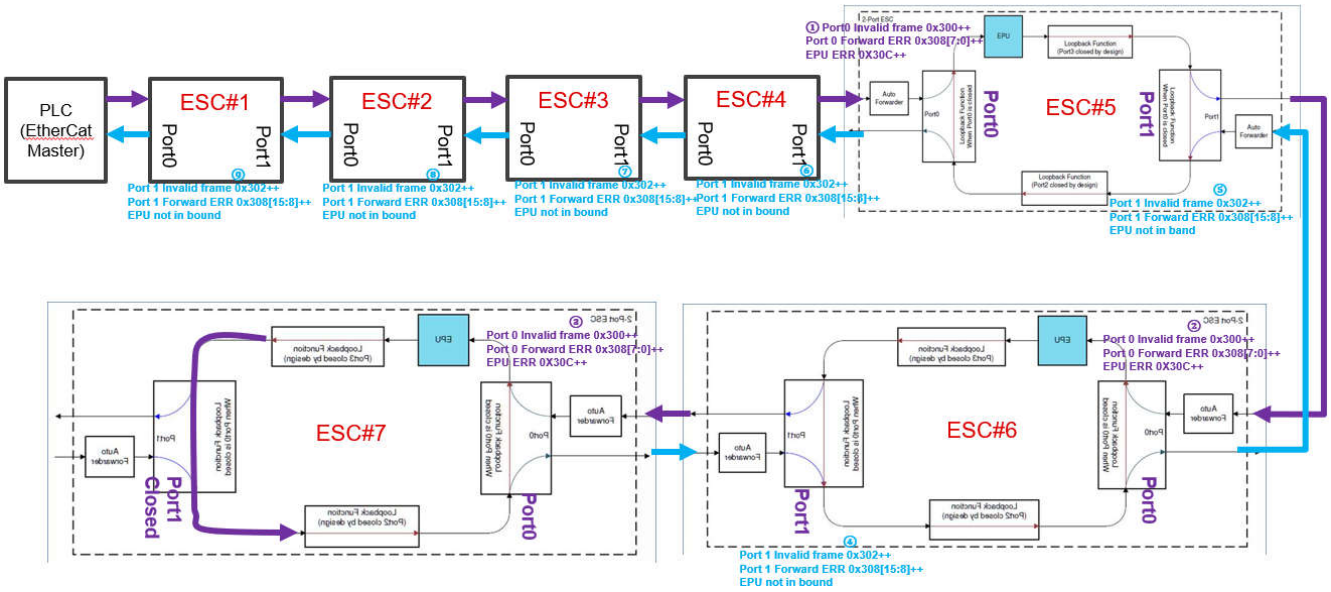

Figure 3-1. Frame Error Fault Case – Error Accumulation and Propagation Process Analysis

The fault counts on ESC5 include Lost Frame for the Master, and CRC Error, Forward Error, and EPU Error for the Slave. According to the ESC error register-based fault localization guide shown in Figure 3-2, the fault register information in this case points to several possible causes, including PHY faults, MII interface timing faults, F28P65 internal timing faults, and so forth. To quickly locate the fault, Section 3.2 describes how to use the PHY Loopback mode for fault isolation and localization.

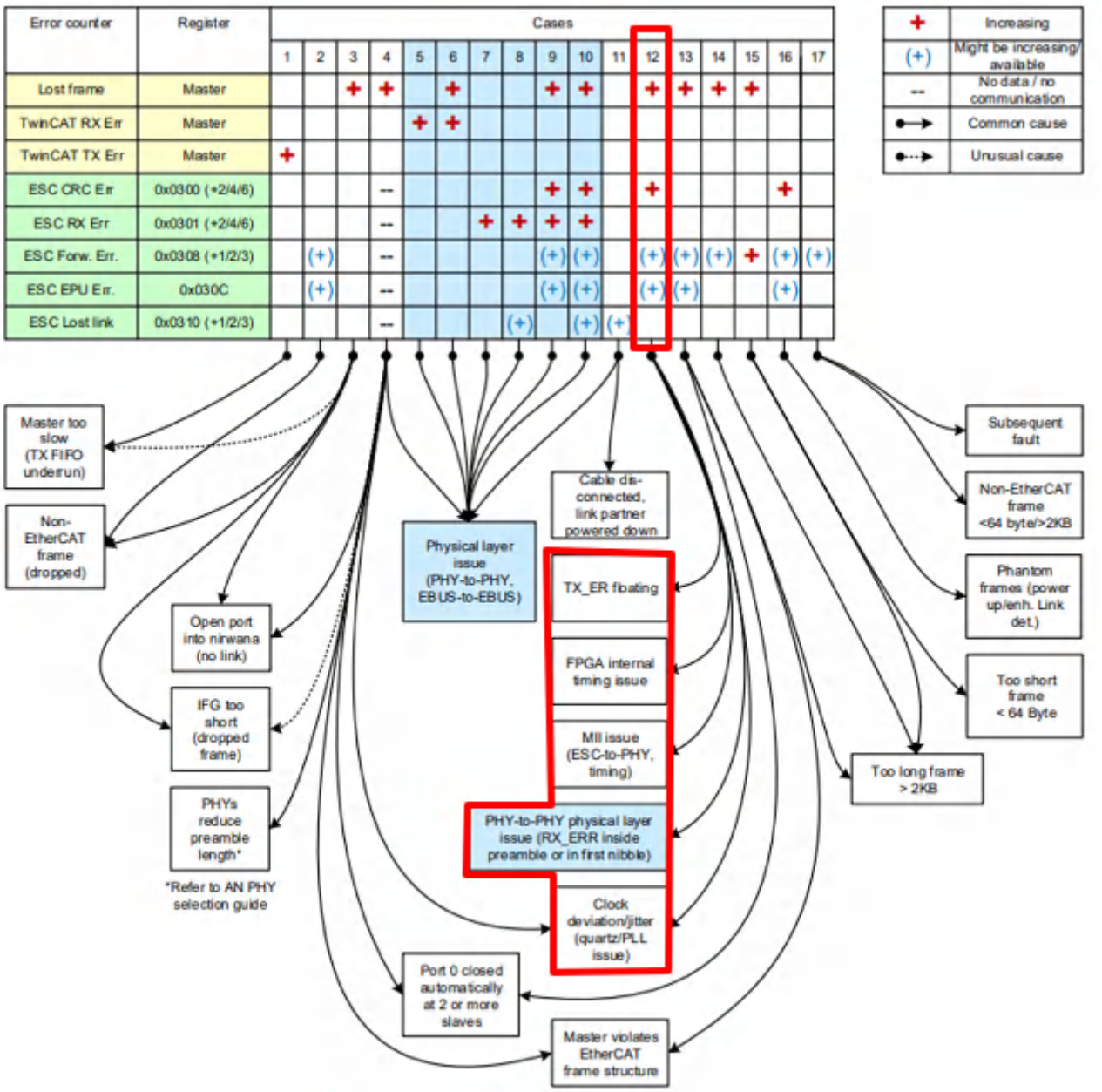

Figure 2: Error counters interpretation guide

Figure 3-2. Guide for Fault Localization Based on ESC Error Registers

3.2 Frame Error Investigation Case – Fault Isolation Analysis

Leveraging the Loopback mode of the PHY chip enables the effective isolation of the EtherCAT fault location, allowing for identifying the fault within the PHY, MII, and F28P65 to aid further analysis. To reproduce the fault, the loop can be configured according to sections 3.2.1 and 3.2.2 below, to verify whether PHY0 or PHY1, MII, or F28P65 on the faulty device ESC5 is the source of the fault.

3.2.1 ESC PHY0 or PHY1 Fault Isolation Analysis

As shown in Figure 3-3, to reproduce the fault, PHY Reverse Loopback can be used to verify the operational status of ESC5 PHY0 and PHY1 individually.

Connect the Master to the leftmost ESC1 and configure PHY0 of the faulty device ESC5 to operate in the Reverse Loopback mode using the following DP83826 register configuration. Data will be looped back at the MII interface of ESC5 PHY0, enabling the verification of whether ESC5 PHY0 is working properly.

The DP83826 PHY Reverse Loopback mode is configured as follows:

```
0000 2100 //Disables Auto-Neg, Selects 100 Mbps
0016 0010 //Select Reverse Loopback
001F 4000 //Soft Reset
```

In this fault case, the Reverse Loopback test for ESC5 PHY0 shows that the Master discovers ESC1 to ESC4 normally under this configuration, and the ESC4 0x0110 register displays Port1 as Open, indicating a good link between ESC4 PHY1 and ESC5 PHY0. Additionally, the data is successfully looped back by the MII interface after passing through ESC5 PHY0; therefore, ESC5 PHY0 is not faulty.

Connect the master to the rightmost ESC7, reverse the loop direction from right to left, and apply the same Reverse Loopback configuration to PHY1 of the faulty device ESC5. Using the same method and criteria, ESC5 PHY1 can also be verified as fault-free.

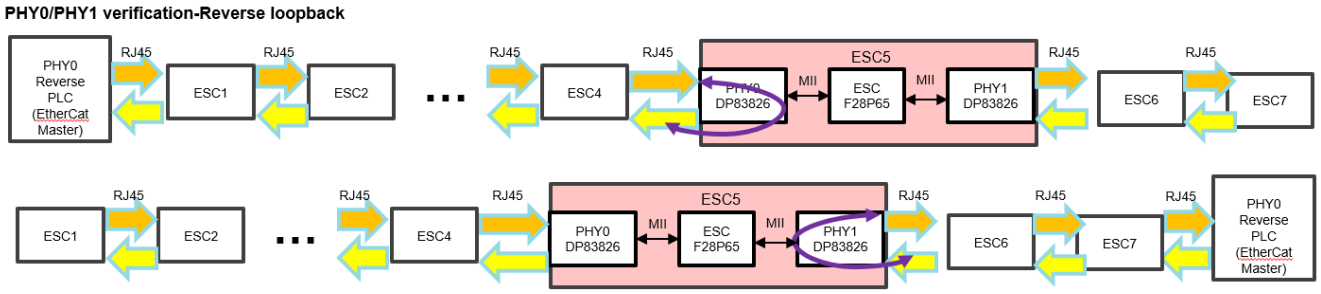

Figure 3-3. Fault Isolation of PHY0 or PHY1 on Faulty Device ESC5

3.2.2 ESC F28P65 and MII Fault Isolation Analysis

After excluding PHY-related faults as described in 3.2.1, comparative testing using PHY MII Loopback and F28P65 Port Force Close can be conducted to analyze and isolate the fault between the internal ports and external MII interfaces of F28P65 on ESC5, as shown in Figure 3-4.

Test 1 (Purple – PHYx MII Loopback Path): If the Master is connected to ESC1 on the far left, configure PHY1 on the faulty device ESC5 to operate in MII Loopback mode, and the data that arrives at ESC5 from the Master is then looped back inside ESC5 PHY1 via the MII interface, without passing through other functional blocks inside PHY1 except for MII.

PHY MII Loopback is configured as follows:

```
0000 6100 //Disables Auto-Neg, Selects 100 Mbps, MII Loopback enabled.
0016 0104 //Select MII Loopback in BISCR
001F 4000 //Soft Reset
```

Specific code configuration can be found below for reference:

```
/* MII Management and PHY Addressing defines */
#define ESC_MII_CTRL_STATUS_1_OFFSET          0x0510
#define ESC_MII_CTRL_STATUS_2_OFFSET          0x0511
#define ESC_PHY_ADDRESS_OFFSET                0x0512
#define ESC_PHY_REG_ADDRESS_OFFSET            0x0513
#define ESC_PHY_DATA_OFFSET                   0x0514
#define ESC_MII_ECAT_ACCESS_OFFSET            0x0516
#define ESC_MII_PDI_ACCESS_OFFSET             0x0517

// Adding PHY configuration code here
HW_EscWriteByte(0x01,ESC_MII_PDI_ACCESS_OFFSET); // Give PDI access to MII management
HW_EscWriteByte(0x00,ESC_PHY_REG_ADDRESS_OFFSET); // Set PHY register to read/write
HW_EscWriteByte(0x01,ESC_MII_CTRL_STATUS_2_OFFSET); // Read PHY reg
HW_EscWriteWord(0x6100,ESC_PHY_DATA_OFFSET); // Set the value to write to register
HW_EscWriteByte(0x02,ESC_MII_CTRL_STATUS_2_OFFSET); // Write PHY reg
```

```
HW_EscWriteByte(0x16,ESC_PHY_REG_ADDRESS_OFFSET); // Set PHY register to read/write
HW_EscWriteByte(0x01,ESC_MII_CTRL_STATUS_2_OFFSET); // Read PHY reg
HW_EscWriteWord(0x0104,ESC_PHY_DATA_OFFSET); // Set the value to write to register
HW_EscWriteByte(0x02,ESC_MII_CTRL_STATUS_2_OFFSET); // Write PHY reg

HW_EscWriteByte(0x1F,ESC_PHY_REG_ADDRESS_OFFSET); // Set PHY register to read/write
HW_EscWriteByte(0x01,ESC_MII_CTRL_STATUS_2_OFFSET); // Read PHY reg
HW_EscWriteWord(0x4000,ESC_PHY_DATA_OFFSET); // Set the value to write to register
HW_EscWriteByte(0x02,ESC_MII_CTRL_STATUS_2_OFFSET); // Write PHY reg
```

Under this configuration, if the Master successfully discovers Slaves ESC1 to ESC5, it indicates that there are no exceptions to internal Port0 or Port1 and external MII interfaces of F28P65 on ESC5, and ESC5 is functioning normally. No Test 2 is required. Conversely, if the Master fails to discover ESC5, it indicates that there is an exception to internal Port0 or Port1 or external MII interfaces of F28P65 on ESC5. Test 2 is then conducted to verify the MII interface in the next step.

Test 2 (Blue – ESC Controller PortX Internal Loopback Path): If the Master is connected to ESC1 on the far left, force close Port1 of F28P65 on the faulty device ESC5 (or force close Port1 by flipping its link state if ESC5 is not discovered by the Master). The data is then transmitted from the Master through ESC1 to ESC4, forwarded internally from ESC5 Port0 to ESC5 Port1, and finally looped back inside ESC5 Port1, without being sent out by ESC5 Port1.

If the Master fails to discover ESC5 in Test 1 and also fails to discover ESC5 in Test 2, it indicates that ESC5 is indeed faulty and the fault source is ESC5 F28P65; if the Master fails to discover ESC5 in Test 1 but successfully discovers ESC5 in Test 2, it indicates that there is an exception to the external MII interface for ESC5 Port1. For example, an external MII trace mismatch causes timing anomalies after the MII signal passes through the PCB traces.

The above case involves a Master connected to ESC1 on the far left, which allows for verifying whether Port1 on the faulty device ESC5, and its external MII interface, are abnormal. To verify whether Port0 and its external MII interface are abnormal, simply connect the Master to the right-most ESC7 and perform analysis accordingly.

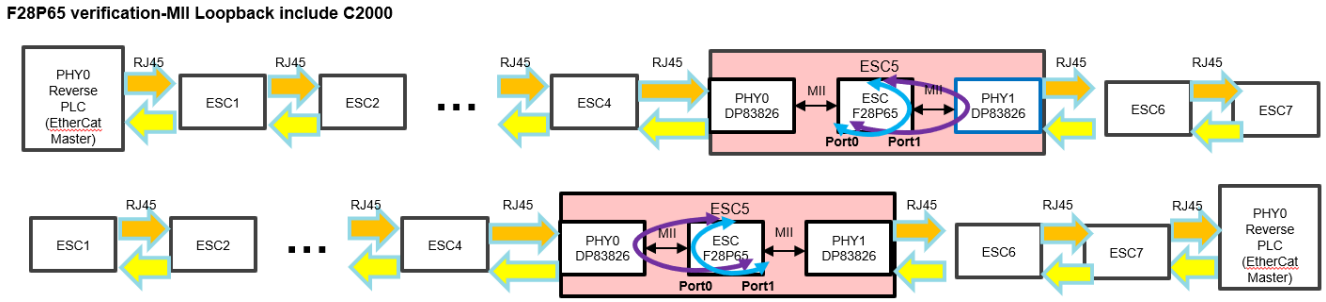

Figure 3-4. Fault Isolation for MII Interfaces of Faulty Device ESC5 and F28P65

Fault Case Measured Results: Connect the Master to the left-most ESC1 and conduct the MII Loopback test on ESC5 PHY1 (Test 1). The Master successfully discovered ESC1 to ESC4 but failed to discover ESC5. In the Port1 Force Close test (Test 2) on ESC5 Port1, the Master still failed to discover ESC5. Connect the Master to the far right ESC7 and conduct the MII Loopback test on ESC5 PHY0 (Test 1). The Master successfully discovered ESC1 to ESC4 but failed to discover ESC5. The Master still failed to discover ESC5 in the Port1 Force Close test (Test 2) on ESC5 Port0. The test results indicate that the fault source of the faulty device ESC5 is internal to the ESC controller, F28P65.

Section 3.2 describes the fault location isolation analysis of an EtherCAT Frame Error fault when the error register information points to multiple possible fault sources, using a real-world frame error fault case as an example of fault isolation analysis. Section 3.3 below will describe the F28P65 internal investigation analysis and application recommendations based on the fault location identified in this case study.

3.3 Frame Error Investigation Case – ESC Fault Investigation Within F28P65

Based on the fault isolation analysis result in Section 3.2, the PHY and MII interfaces have been excluded as the cause of the fault. Therefore, this section will investigate the ESC internal timing.

It is worth noting that, in addition to the Invalid Frame, Forward Error, and EPU Error involved in the error accumulation and propagation analysis in Section 3.1, there are PDI Error counts from ESC5 through ESC7. The PDI is an important interface for interaction between the ESC and the F28P65 CPU, and the official ESC manual also states that the PDI error count should be set to zero after power-on. Therefore, based on the fault isolation analysis result in Section 3.2 and the PDI Error-related information, it can be further concluded that there is a serious issue with the interaction between the SC IP core within the F28P65 chip and the F28P65 CPU for ESC5. This section will analyze this issue in detail and provide solutions.

The ESC is a peripheral function module of F28P65. Figure 3-5 shows how the ESC is connected to F28P65. The 16-bit asynchronous interface (PDI) and interrupt requests are the main channels for interactions between the F28P65 CPU and the ESC. Interrupt requests can be used to trigger actions based on ESC internal events, exceptions, or time-synchronized events. At the same time, the initialization of the ESC is controlled by the F28P65 CPU, so every step of the initialization process must be checked and analyzed for possible ESC dysfunction.

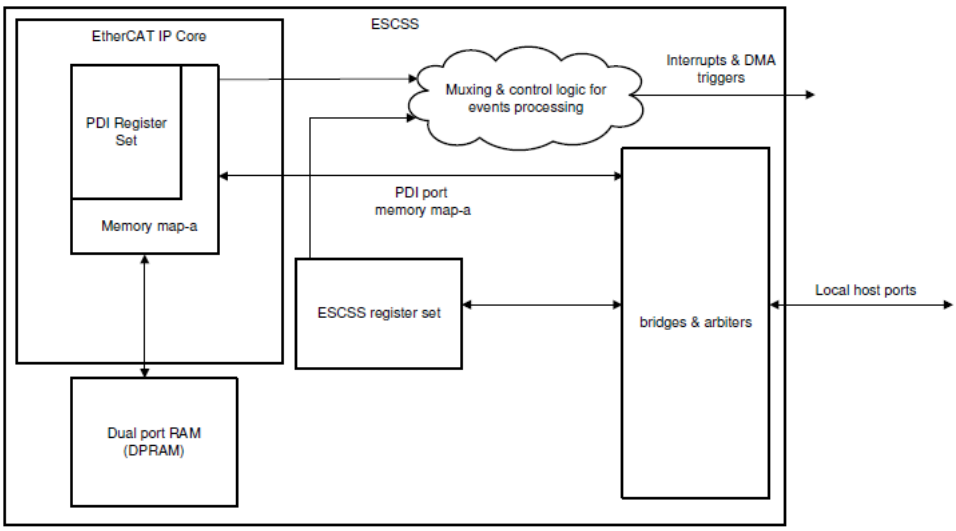

Figure 3-5. EtherCAT IP Core to F28P65 Communication Model

Figure 3-6 shows the process of initializing the ESC IP core of F28P65 CPU1. By studying the previous case, it can be determined that CPU1 of ESC4 is working normally, and the ESC4 EtherCAT IP core is also operational; otherwise, no error counts would be recorded. Perform analysis following the ESC initialization process step by step. Step 5 is a critical point to investigate, where the EtherCAT clock source and division factor are set. Issues with the system clock configuration, if any, will result in serious communication faults, which also correspond to what we have observed. Combined with the limitations of the system control register configuration in the F28P65 TRM, the system control register memory operates on the 10MHz INTOSC1 clock, while the CPU runs at 200MHz. Therefore, a certain delay should be incorporated for writes to associated registers; otherwise, a second write fails. The ETHERCATCLKCTL register in step 5, which sets the clock source and division factor, operates exactly on this low-frequency 10MHz clock. By checking the TI-provided library functions, it is true that two consecutive writes to ETHERCATCLKCTL are executed, with the second write configuring the required clock source and division factor. However, as previously described, the second correct configuration is most likely not written correctly, which will lead to a serious communication error. After this issue was identified, tests were conducted for different delays, and it was found that the probability of fault reproduction varied with the delay time.

Step	Action
1	General device initialization (configure clock, enable PLL, enable peripheral clocks except EtherCAT)
2	Configure Aux Clock for EtherCAT (if using Aux clock as source)
3	Configure GPIOs for EtherCAT (set pin configurations, set GPIO qualification mode, set pad configuration)
4	Initialize interrupts and register ISR handlers
5	Set EtherCAT clock source and divider. Then configure if EtherCAT PHY is clocked from device or external PHY clock.
6	Configure the EEPROM size
7	Bring ESC out of reset using system control register
8	Perform EtherCAT memory initialization and wait until memory initialization is complete
9	(Optional) Enable debug access to the EtherCAT registers
10	(Optional) Check that EEPROM loaded successfully
11	EtherCAT subsystem configurations for interrupt masking, SYNCx connections, and so on(1)

(1) Applications must make sure that ESC outputs are in a safe state until the EEPROM is loaded and that SYNC and LATCH are configured only after the EEPROM is loaded.

Figure 3-6. F28P65 CPU1 ESC Initialization Process

```
static inline void
SysCtl_setECatClk(SysCtl_ECatClkDivider divider, SysCtl_PLLClockSource source,
                        uint16_t enable)
{
    //
    // Clears the divider & the source, then configures it.
    //
    EALLOW;
    HWREGH(CLKCFG_BASE + SYSCTL_O_ETHERCATCLKCTL) =
                    (HWREGH(CLKCFG_BASE + SYSCTL_O_ETHERCATCLKCTL) &
                     ~(SYSCTL_ETHERCATCLKCTL_PHYCLKEN  |
                       SYSCTL_ETHERCATCLKCTL_ECATDIV_M |
                       SYSCTL_ETHERCATCLKCTL_DIVSRCSEL));


    HWREGH(CLKCFG_BASE + SYSCTL_O_ETHERCATCLKCTL) |=
                ((uint16_t)divider << SYSCTL_ETHERCATCLKCTL_ECATDIV_S) |
                ((uint16_t)source << SYSCTL_ETHERCATCLKCTL_DIVSRCSEL_S) |
                (enable << SYSCTL_ETHERCATCLKCTL_PHYCLKEN_S);

    EDIS;
}
```

Figure 3-7. F28P65 EtherCAT Clock Setup Library Function

3.3.1 EtherCAT Clock Configuration Failure Analysis

To analyze the root cause of any EtherCAT clock configuration failure, the design team created a test environment and performed simulations, as shown in Figure 3-8, where two writes to the ETHERCATCLKCTL register were executed within approximately 40 cycles. As shown in Figure 3-8, the first and second writes are marked in red (VBUS32_REQ: Yellow waveform). This write operation must configure the value of the EtherCAT clock divider, which operates in a slower clock domain (INTOSC1). Therefore, a pulse transmission from 200MHz to 10MHz is required. Under the pulse transmission signal “Pulse sync”, signals i_src_clk (source clock), i_src_pulse (source pulse), i_des_clk (destination clock), o_des_pulse (destination pulse), and des_ack (acknowledge signal from destination domain) are displayed. Any other write to this register is missed as long as the ACK signal is high. In this case, a second write to the register occurs when the des_ack signal (magenta or pink signal) is high. However, the divider never receives this value; it is only configured with the old value.

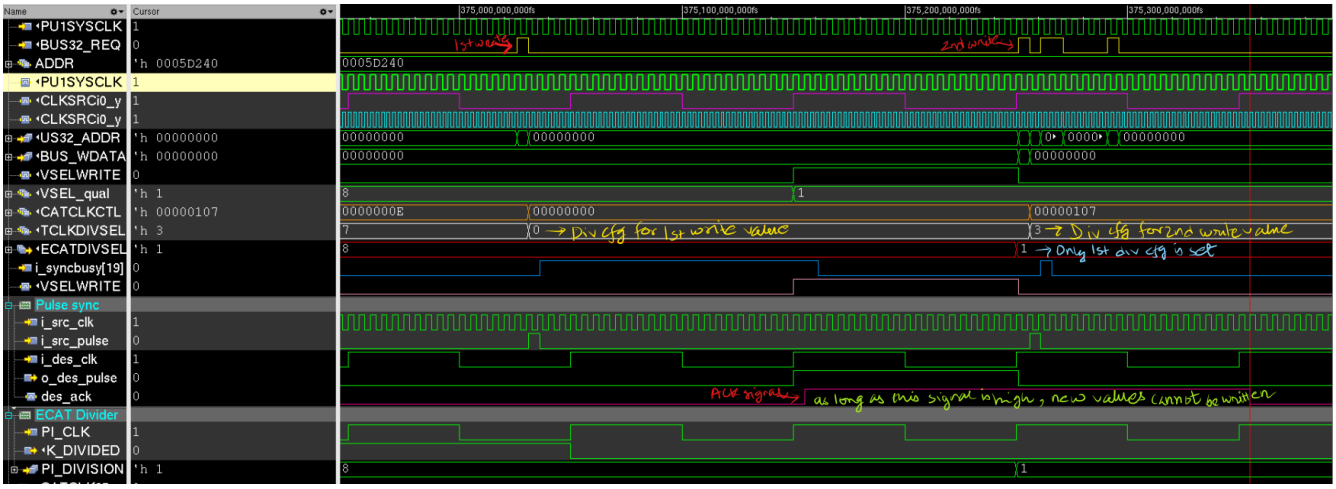

Figure 3-8. RTL Simulation for ESC Clock Configuration

3.3.2 Recommendations for EtherCAT Clock Configuration

The failure issue described above is essentially caused by metastability during clock domain crossing. This issue occurs when a signal is transferred from one clock domain to another, and the phase relationship between the two clocks is uncertain, or their frequencies differ. Metastability is an indeterminate state of a flip-flop in a digital circuit, which occurs when the clock sampling signal arrives while the data input signal is still changing. To ensure the second write is successful and to prevent metastability issues during clock domain crossing, it is recommended to calculate the delay for two register writes using the following equation:

Delay = 6 x (F_{SYSCLK} / $F_{INTOSC1}$) + 9

Therefore, for the division value of the EtherCAT clock to be written correctly, there should be a delay of at least 129 cycles. As shown in Figure 3-9:

#define SYSCTL_REGWRITE_DELAY **asm**(" RPT #129 || NOP")

```
static inline void
SysCtl_setECatClk(SysCtl_ECatClkDivider divider, SysCtl_PLLClockSource source,
                  uint16_t enable)
{
    //
    // Clears the divider & the source, then configures it.
    //
    EALLOW;
    HWREGH(CLKCFG_BASE + SYSCTL_O_ETHERCATCLKCTL) =
                   (HWREGH(CLKCFG_BASE + SYSCTL_O_ETHERCATCLKCTL) &
                    ~(SYSCTL_ETHERCATCLKCTL_PHYCLKEN  |
                      SYSCTL_ETHERCATCLKCTL_ECATDIV_M |
                      SYSCTL_ETHERCATCLKCTL_DIVSRCSEL));
    SYSCTL_REGWRITE_DELAY;

    HWREGH(CLKCFG_BASE + SYSCTL_O_ETHERCATCLKCTL) |=
                ((uint16_t)divider << SYSCTL_ETHERCATCLKCTL_ECATDIV_S) |
                ((uint16_t)source << SYSCTL_ETHERCATCLKCTL_DIVSRCSEL_S) |
                (enable << SYSCTL_ETHERCATCLKCTL_PHYCLKEN_S);
    SYSCTL_REGWRITE_DELAY;
    EDIS;
}
```

Figure 3-9. EtherCAT Clock Library Functions With Delay Included

F28P65 library functions often clear a register before writing to it, which can cause the second write to fail because of insufficient delay. Another simple approach is to execute only one normal value write to the register when configuring the EtherCAT clock, thereby avoiding configuration failures caused by a second write.

3.3.3 Addressing Metastability Issue During Clock Domain Crossing Using DCC

The Dual Clock Comparator (DCC) is a safety feature module in the F28P65 that monitors the clock frequency and detects clock faults. Its core function is to compare the frequencies of two independent clock sources to ensure they are operating within the expected range.

Although incorporating enough delay between two writes guarantees the success of the second write, the metastability issue introduced by clock domain crossing is inevitable. There is a very low probability of metastability occurring, which could still potentially lead to incorrect ESC clock configuration and subsequently cause communication errors. Refer to Figure 3-6 to use the C2000 DCC peripheral to monitor whether the ESC clock is configured properly.

```
// Verify the frequency of ECATPHYCLK clock using the SYSCLK as reference clock
// FClk1 = eCAT PHY Clock frequency = 100MHz
// FClk0 = SYSCLK frequency = 200MHz
// Tolerance = 1%
// Allowable Frequency Tolerance = 0% (update as per the error in the ECATPHYCLK frequency)
// SysClk Freq = 200MHz
//
// Note: Update the parameters in case you are using different PLL or XTAL
// frequencies,
//
status = DCC_verifyClockFrequency(DCC0_BASE,
                                  DCC_COUNT1SRC_ECATPHYCLK, 100.0F,
                                  DCC_COUNT0SRC_SYSCLK, 200.0F,
                                  1.0F, 0.0F, 200.0F);
```

Figure 3-10. Monitoring EtherCAT Clock With DCC Module

4 Summary

This article begins by introducing the basic concepts of EtherCAT communication, followed by its characteristics, principles, communication model, and the diagnostic capabilities of ESCs. Starting with a real-world frame error fault case, it presents a methodology for analyzing EtherCAT frame errors, which includes diagnostic information investigation, error accumulation and propagation analysis, and fault location isolation analysis. Furthermore, it provides recommendations for using F28P65 in EtherCAT applications, sharing analysis and debugging experience for scenarios implementing EtherCAT applications with F28P65 + DP83826.

5 References

1. EtherCAT_Diagnosis_For_Developers
2. ethercat_esc_datasheet_sec1_technology_2i3
3. ethercat_esc_datasheet_sec2_registers_3i0
4. ethercat_et1100_datasheet_v2i1
5. F28P65 TRM
6. DP83826 Troubleshooting Guidance

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, regulatory 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 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, TI’s General Quality Guidelines, 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. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025