

PRU-ICSS EtherCAT Slave Troubleshooting Guide

Garrett Ding, Paula Carrillo

ABSTRACT

TI's EtherCAT™ solution has been widely adopted in the Factory Automation, Automated Machinery and the Robots market sectors. This troubleshooting guide is intend to provide guidance on how to set up and debug the EtherCAT slave implemented on TI Sitara™ processors. Tips, tricks, logical and systematic debugging techniques are offered. This troubleshooting guide does not reiterate the general EtherCAT diagnostics for users and developers provided by EtherCAT Technology Group (ETG).

Contents

1	PRU-ICSS EtherCAT Slave Overview	2
2	EtherCAT Slave Demo Setup	3
3	Troubleshooting	4
4	References	14
Appendix A	PRU-ICSS Debug via CCS	15
Appendix B	Testing EtherCAT Slave With a Master on Sitara Processors	17
Appendix C	Adding/Changing PDO	18
Appendix D	FAQ	21

List of Figures

1	PRU-ICSS EtherCAT Software Architecture	3
2	First 16 Words of EEPROM	5
3	EEPROM Snapshot	5
4	ESC Register 0x140.....	5
5	EtherCAT FMMU0 Frame Trace.....	8
6	EtherCAT FMMU1 Frame Trace.....	9
7	EtherCAT DC Sync Jitter Capture	10
8	Malformed Packet Trace	11
9	LRW With Zero Data	11
10	EtherCAT FMMU/SM Configuration	12
11	14 Bytes Long Accessing Two Slaves	13
12	24 Bytes Long Accessing Two Slaves	13
13	LRD/LWR Accessing Two Slaves.....	13
14	PRU_0 and PRU_1 Cores in CCS.....	15
15	PRU_1 Disassembly Dode.....	15
16	Save Memory	15
17	EEPROM Flush Call Flow.....	21

List of Tables

1	EEPROM Header.....	5
2	Process Data Interfaces	6
3	Device Identity (hex)	6
4	Default PRU-ICSS Instances.....	6

5	EtherCAT DC Cycle Time	10
6	EtherCAT DC Sync Jitter	10
7	PRU-ICSS Global Memory Map	16

Trademarks

Sitara, Code Composer Studio are trademarks of Texas Instruments.
 Arm is a registered trademark of Arm Limited.
 EtherCAT is a trademark of Beckhoff Automation GmbH, Germany.
 All other trademarks are the property of their respective owners.

1 PRU-ICSS EtherCAT Slave Overview

TI's Sitara System-on-Chip (SoC) family offers the Programmable-Realtime Unit Industrial Communications Subsystem (PRU-ICSS), which enables the integration of real-time industrial communications protocols and eliminates the need for an external ASIC or FPGA.

TI has released several PRU-ICSS industrial software protocols including EtherCAT, which are adds-on to TI's processor SDK, a unified software platform for TI embedded processors.

EtherCAT, invented by Beckhoff Automation in Germany and later standardized by the ETG, is a realtime industrial Ethernet standard for industrial automation applications, such as input/output (I/O) devices, communication modules, sensors, and programmable logic controllers (PLCs). EtherCAT improves upon the traditional Ethernet by implementing on-the-fly processing, where the nodes in the EtherCAT network read the data from a frame as it passes through. All EtherCAT frames originate from the EtherCAT master, which sends commands and data to the slaves. Any data to be sent back to the master is written by the slave onto the frame as it passes through.

Three major software components comprise the EtherCAT slave implementation: the physical layer, data link layer, and application layer.

The first component is the physical layer, PHY, which provides all physical layer functions needed to transmit and receive data over standard twisted-pair cables. The Sitara processors with a TI Ethernet PHY device, such as DP83822, exhibits a low latency.

In EtherCAT layer 2, the data link, the PRU real-time cores execute the tasks of datagram processing, distributed clocking, address mapping, error detection and handling, and host interface. PRUs also emulate the EtherCAT register space in the internal shared memory. With their deterministic real-time processing capability, the PRUs handle EtherCAT datagrams with consistent and predictable processing latency.

The third component is the EtherCAT slave stack, which runs on the Arm® processor and industrial application that is dependent on the end equipment in which this solution is used. For the application layer connection, different process data interfaces (PDI) are available.

Additional supporting components, such as the protocol adaptation layer and device drivers, are provided in the Processor SDK from TI.

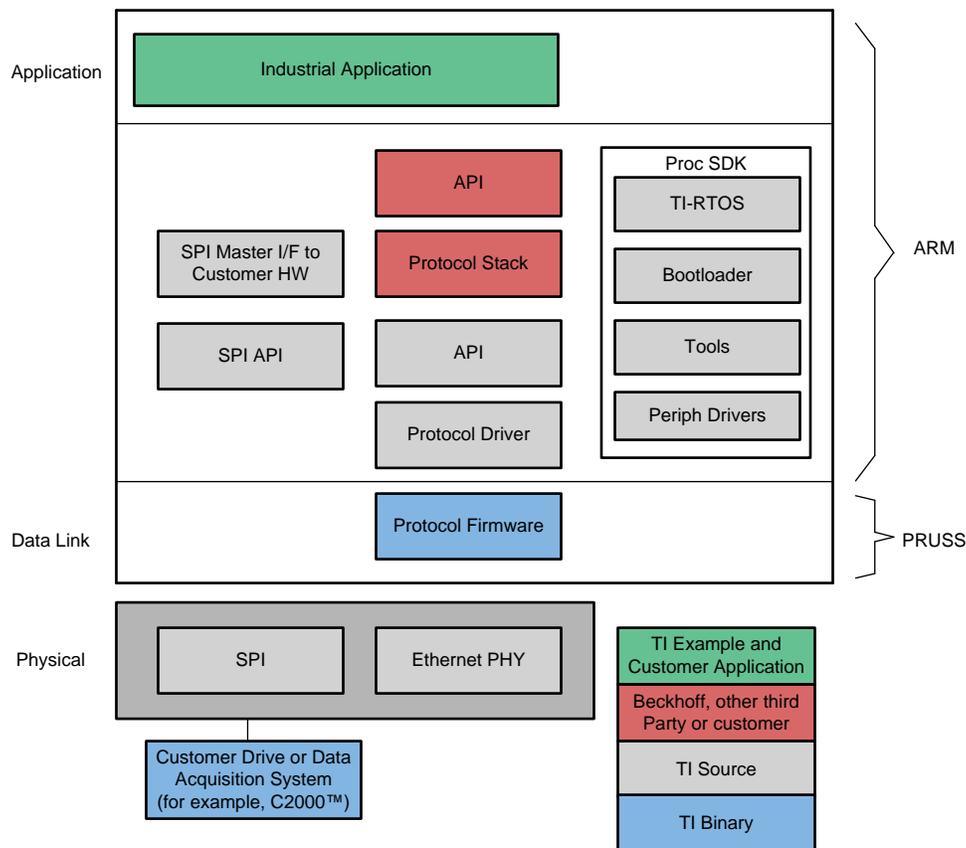


Figure 1. PRU-ICSS EtherCAT Software Architecture

For more details on TI's EtherCAT implementation, see the [EtherCAT on Sitara processors white paper](#).

2 EtherCAT Slave Demo Setup

PRU-ICSS EtherCAT slave is an add-on package to the TI-RTOS Processor SDK (PSDK). Download the TI-RTOS PSDK version recommended on EtherCAT system requirements in order to assure correct functionality. Mix and match between TI-RTOS PSDK and PRU-ICSS EtherCAT slave version could lead to incorrect behavior.

2.1 Platform Setup

PRU-ICSS EtherCAT slave is supported in the following platforms:

- AM572x IDK 1.3A/B
- AM571x IDK 1.3A
- AM437x IDK
- AM335x ICEV2
- K2G ICE
- AMIC11x ICE
- AMIC12x ICE

Note that the AM335x ICE Rev. 2.1 EVM has two 10/100 Ethernet transceivers (TLK110) interfaced to connectors J1 and J2. The Ethernet ports are connected from the Gig Switch and the PRU-ICSS units of the AM335x to the transceivers through a muxing/ORing logic. Jumpers J18 and J19 need to be set to control the Ethernet ports using CPSW (Gig Switch) or PRU-ICSS mode. For PRU-ICSS mode used for EtherCAT, connect Pin2 and Pin3.

2.2 How to Build and Run PRU-ICSS EtherCAT Slave

These links provide instructions on how to build and run EtherCAT slave demo:

- Code Composer Studio™ project creation: [Industrial Protocol Package Getting Started Guide](#) wiki
- [Running and Configuring EtherCAT Slave](#) wiki

These are links to the application report and how-to videos:

- [DDR-less EtherCAT® Slave on AMIC110 Reference Design](#)
- [Demonstrating DDR-less EtherCAT® Slave on AMIC110](#)
- [Demonstrating TI ESC SPI Mode DDR-less AMIC110 with C2000 EtherCAT Slave](#)

3 Troubleshooting

To isolate and resolve the problems identified from its symptoms, the troubleshooting process is categorized into the following sections:

- Sanity check
 - CCS project build/run
 - ESI file and EEPROM
 - EtherCAT master configuration
- Hardware check
 - PRU-ICSS Ethernet ports
 - PHY
 - Bootloader
 - EEPROM
- Status and error counters
- Frame traces and working counter
- DC sync cycle time and jitter
- LRW access to non-interleaved input and output process data
- DDR-less and ET1100 mode

3.1 Sanity Check

3.1.1 EtherCAT Demo Project Build Failure or Run Exception

Prior to EtherCAT release 1.0.6, the EtherCAT demo project may fail to build in Code Composer Studio or run on platforms due to incompatible TI-RTOS Processor SDK releases.

- Use the Processor SDK components listed in the [System Requirement](#) section of the [PRU-ICSS EtherCAT user's guide](#).
- The folder [INSTALL-DIR]/protocols/pdk_patches of EtherCAT software package may contain the patches for:
 - PDK drivers
 - Thumb vs. ARM mode
 - Secondary bootloader

Except the thumb mode patch for EtherCAT DDR-less on AMIC11x, which is not planned to be added into Processor SDK. The other patches, if any, will be merged in later releases.

For the AMIC110 DDRless application to build, it is critical that the processor SDK is built in Thumb Mode. This patch file can be found at [INSTALL-DIR]/protocols/pdk_patches/04.03.00/AM335x_PDK_thumb_mode.patch. After applying this package, the Processor SDK needs to be cleaned and rebuilt.

3.1.2 ESI File and EEPROM

EtherCAT EEPROM stores slave ID and information about the slave's functionality. EEPROM information can be derived from the Slave Information (ESI) file in XML format.

The EtherCAT master uses the EtherCAT Network Information file (ENI) created using the configuration tool and based on the ESI file, which is provided by the vendors for every device. The ENI file describes the network topology, the initialization commands for each device and the commands that have to be sent cyclically.

The EtherCAT demo project works with the ESI xml file provided in the ESI folder. For the application to work with another ESI xml file, you need to generate a corresponding ESI header file (tiesc_eeeprom.h) and re-build the ethercat_slave_full application project with the generated .h file. To generate the binary file equivalent to the ESI xml file, see [Generating EEPROM binary](#). The binary file to header file can be converted using the bin2header.exe utility available from [INSTALL-DIR]/examples/tools/bin2header.

Figure 2 shows the first 16 words of EEPROM contents.

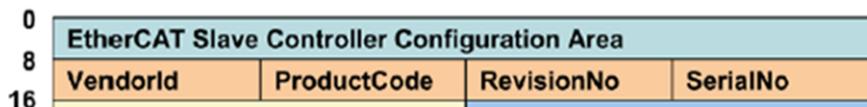


Figure 2. First 16 Words of EEPROM

The first 8 words (16-bit) configuration area contains the PDI control, configuration, sync signal, station alias information as shown in Table 1.

Table 1. EEPROM Header

Word	Register Description	Register Address
0	PDI control	0x0140:0x0141
1	PDI configuration	0x0150:0x0151
2	Pulse length sync signals	0x0982:0x0983
3	Extended PDF configuration	0x0152:0x0153
4	Configured station alias	0x0012:0x0013
5	Reserved	-
6	Reserved	-
7	CRC	-

Figure 3 indicates the PDI type 0x80 - on-chip bus.

```
0000 80 0C E0 88 E8 03 00 00 01 00 00 00 00 00 A2 00
0010 9D 05 00 00 01 00 49 54 11 00 00 00 00 00 00 00
```

Figure 3. EEPROM Snapshot

Figure 4 shows the ESC register 0x140.

PDI Control	0x0140					PDI access is exception here for TI ESC
		0-7	R/-	RW-	0x80	PDI type / EEPROM 0x0000 On-chip bus

Figure 4. ESC Register 0x140

The EtherCAT full feature demo application polls the PDI control in a while loop.

```
while(ul6PdiCtrl != 0x80); //see HW_Init() of tieschw.c
```

Similarly, while the PDI interface is SPI (in TI's C2000 + AMIC110 EtherCAT demo application), it will poll the PDI control word 0x05.

Table 2. Process Data Interfaces

PDI Number (PDI Control Register 0x0140[7:0])	PDI Name
0	Interface deactivated
4	Digital I/O
5	SPI slave
7	EtherCAT Bridge (port 3)
8	16-bit asynchronous μ C
9	8-bit asynchronous μ C
10	16-bit asynchronous μ C
11	8-bit asynchronous μ C

The second 8 words contain the vendor ID, product code, revision number and serial number (see [Table 3](#)):

Table 3. Device Identity (hex)

Vendor Id	0x00000059D
Product Code	0x54490001
Revision Number	0x00000011
Serial Number	0x00000000

3.1.3 EtherCAT master configuration

TI EtherCAT slave does not support non-overlapping PDOs. TwinCAT by default offers optimal overlapping PDO configuration. However some EtherCAT masters like IGH, SOEM, requires a special patch for TI ESC to reach OP state. The details are described in the section 4.6 LRW access to non-interleaved input and output process data.

3.2 Hardware Check

3.2.1 PRU-ICSS Ethernet Ports

The RJ45 ports on TI EVMs are connected to PRU-ICSS and the CPSW Gig Switch on TI EVMs. Caution should be used when connecting the Ethernet cable from the master to the PRU-ICSS instance ports.

[Table 4](#) provides details on the default PRU-ICSS instance on all the supported ICE and IDK Boards.

Table 4. Default PRU-ICSS Instances

SoC/Board	PRU-ICSS Instance Available	Default PRU-ICSS Instance
AM335x / AM335x ICEv2	PRU-ICSS	PRU-ICSS
AMIC11x / AMIC11x ICE		
AM437x / AM437x IDK	PRU-ICSS0, PRU-ICSS1	PRU-ICSS1
AMIC12x / AM437x IDK		
AM572x / AM572x IDK	PRU-ICSS1, PRU-ICSS2	PRU-ICSS2
AM571x / AM571x IDK		
K2G / K2G ICE	PRU-ICSS0, PRU-ICSS1	PRU-ICSS1

For example, on AM572x IDK, the Ethernet cable should be connected to the EtherCAT IN/Port0 of PRU-ICSS 2 on AM572x IDK. If there are multiple IDKs in the chain, you need a connect from the EtherCAT OUT/Port1 to the IN/Port0 of the next IDK. For the last IDK in chain, OUT/Port1 will be left open (if NOT using redundancy mode).

To use PRU-ICSS1 on the AM571x IDK or K2G ICE, the #define PRUICSS_INSTANCE needs to be set to PRUICSS_INSTANCE_ONE in the [INSTALL-DIR]/protocols/ethercat_slave/include/tiesc_soc.h file.

3.2.2 PHY

Proper Ethernet physical layer device (PHY) configuration using data input/output (MDIO) is an essential step to bring up EtherCAT on a custom board, and crucial to meeting the requirements of lowest deterministic latency and fastest link detection in industrial Ethernet applications such as EtherCAT.

When the MDIO fails to access PHY_ID1_REG (register 0x02) with the 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 a different general-purpose input/output (GPIO).

[Ethernet PHY Configuration Using MDIO for Industrial Applications](#) provides guidance on the Ethernet PHY configuration using the MDIO module within PRUICSS for industrial applications, and dissects the PHY DP83822 configuration in EtherCAT on the AMIC110 Industrial Communications Engine (ICE).

3.2.3 Bootloader

By default, bootloader supports power-on-reset bootstraps for the board. It initializes the board, loads the application from the memory device to DDR and transfers control to the application. The processor SDK RTOS bootloader uses a two stage boot process: ROM Bootloader (RBL) and Secondary Bootloader (SBL).

For EtherCAT on AMIC11x and AMIC12x DDRless execution, it requires a special SBL that loads the firmware and EEPROM data along with the application without DDR.

The EtherCAT Slave 1.0.6 release is on top of the processor SDK 4.3 and compatible with the processor SDK 5.0. The SBL for AMIC110 DDRless execution is only available from the processor SDK 5.0 or from the processor SDK 4.2 after the update to <PDK_INSTALL_DIR>\packages\ti\starterware\.

For the on-chip memory (DDRless) execution on AMIC12x ICE, the application utilizes the L2_cache as SRAM. By default, the SBL configures the L2_cache as a cache memory and not SRAM. As the memory needs to be configured as SRAM before starting the application, the SBL must be modified for the same. A patch is available in the EtherCAT Slave Package for modifying the SBL source at this path [INSTALL-DIR]/protocols/pdk_patches/04.03.00/AMIC12x_DDR-less_MLO.patch. The SBL needs to be rebuilt after applying the patch. This patch modifies both the mmcsd and the QSPI bootloaders.

3.3 Status and Error Counters

EtherCAT provides a variety of status and error counter registers that helps identify issues quickly. Below is the type of registers implemented in TI ESC:

- ESC DL Status: 0x0110-0x0111 Provides the status of enhanced link detection, physical link on port 0/1, communication on port 0/1.
- AL Status: 0x0130-0x0131 Provides current ESC status: 1: INIT, 3: BOOTSTRAP, 2: PREOP, 4: SAFEOP, 8: OP
- AL Status Code: 0x0134
- RX Error Counter: 0x0300-0x0307 Provides the numbers of invalid frame and Rx errors. Counting is stopped when 0xFF is reached. Clear if any of the counters is written. The invalid frame counters are incremented if there is an error in the frame format (Preamble, SFD – Start of Frame Delimiter, FCS – Checksum, invalid length).
- Forwarded RX Error Counter: 0x0308-0x030B

3.5 DC Sync Cycle Time and Jitter

With three boards running full mode applications connected with the distributed clocks (SYNC0) enabled and CoE data in “Auto-update” for all objects, the lowest cycle time is collected (see Table 5).

Table 5. EtherCAT DC Cycle Time

SoC/Board	Arm CPU Frequency	Lowest Cycle Time (tested)	Remarks
AMIC11x/iceAMIC110	300 MHz	62.5 μ s	Tested in DC mode with CoE update enabled
AM335x/AM335x ICEv2	600 MHz	62.5 μ s	Tested in DC mode with CoE update enabled
AM437x/AM437x IDK	600 MHz	50 μ s	Tested in DC mode with CoE update enabled
AM57xx/AM57xx IDK	1 GHz	31.25 μ s	Tested in DC mode with CoE update enabled
K2G/K2G ICE	600 MHz	50 μ s	Tested in DC mode with CoE update enabled

Typical PCs with the EtherCAT master have trouble going below 1000 μ s cycle time. It might generate the AL status code 0x1a error and the slave goes back to safeop. Using a EtherCAT master PLC is recommended.

To reduce the DC sync jitter, ensure the optimized SoC PLL configuration and the appropriate TX_START_DELAY, which defines the minimum time interval (delay) between receiving the receive data valid (RXDV) signal for the current frame and the start of the transmit interface sending data to the MII interface are set.

The jitter measurement shown in Table 6 was performed in the topology Master \leftrightarrow AM3 \leftrightarrow AM4 \leftrightarrow K2G \leftrightarrow AM572x \leftrightarrow AM571x \leftrightarrow AM3, running in 100 μ s cycle time for 2 day.

Table 6. EtherCAT DC Sync Jitter

Slave Number	Signal Color	Jitter in ns
1 (AM335x)	Channel 1 (Yellow)	NA
6 (AM335x)	Channel 2 (Green)	23

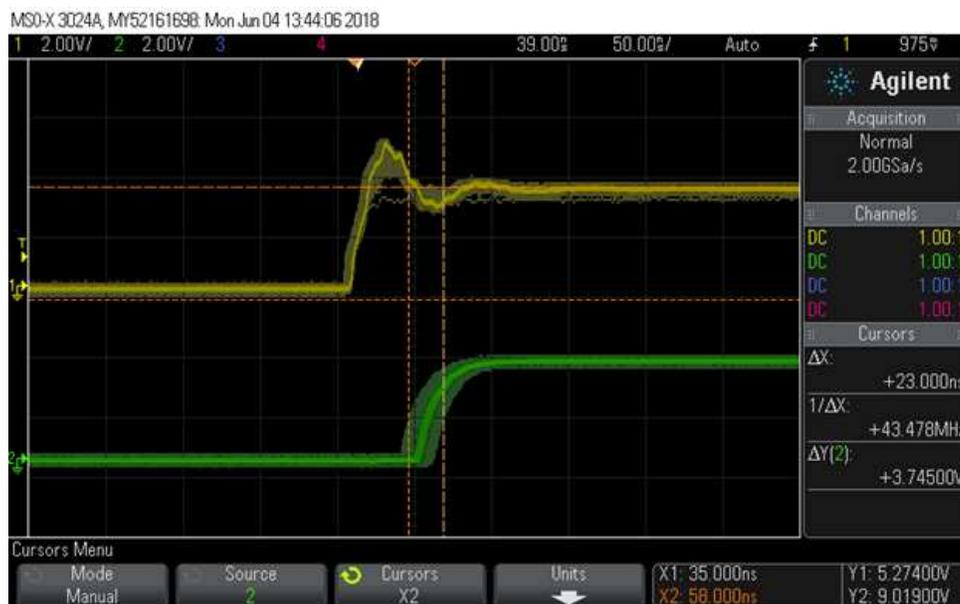


Figure 7. EtherCAT DC Sync Jitter Capture

3.6.1 Failing Cases and Workaround Covered in PINDSW-141 Errata

In the *Conditions in which failures occur* section of the [EtherCAT slave errata PDINSW-141](#), it lists:

- A single LRW datagram accessing FMMU mapped areas in multiple slaves and PD out is mapped:
 - FMMU0(0x1000:0x1007)->SM2 #1(Write SM) FMMU1(0x1008:0x100F)->SM2#2(Write SM)
 - FMMU2(0x1010:0x1017)->SM3 #1(Read SM) FMMU3(0x1018:0x101F)->SM3#2(Read SM)
- A single LRW access from (0x1000:101F)

Three workarounds are provided:

1. Workaround 1: Use the LRD/LWR datagram to access process data. This workaround is suboptimal as more overhead from the packet header and the working counter are introduced. For more details, see [Section 3.6.2](#).
2. Workaround 2: Use the LRW datagram to access process data, input and output overlaid on the same logical address range (TwinCAT usage). This is the optimal configuration and TI highly recommends this workaround. [Figure 10](#) demonstrates that the FMMU0/SM2 and FMMU1/SM3 (physical address 0x1100/0x1400) are mapped to the same logic address 0x01000000.

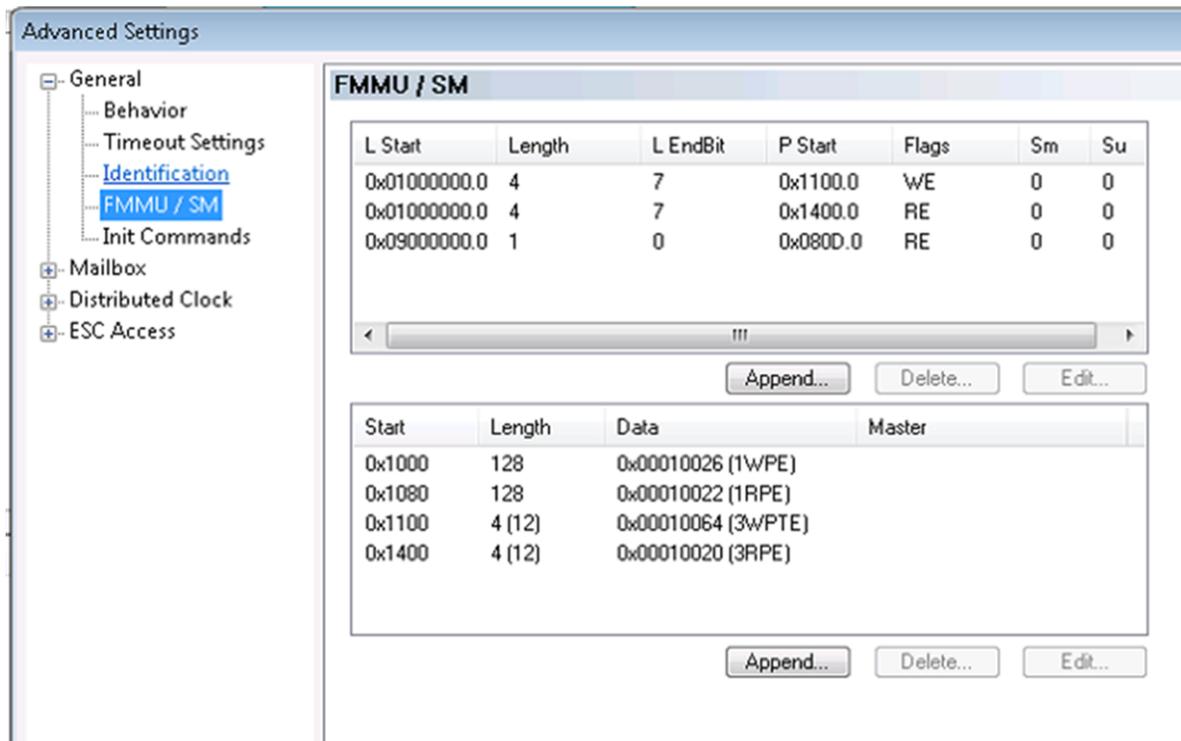


Figure 10. EtherCAT FMMU/SM Configuration

3. Workaround 3: Use the LRW datagram to access process data, input and output of a given slave back to back in a logical address space.
 - a. FMMU0(0x1000:0x1007)->SM2 #1(Write SM) FMMU1(0x1008:0x100F)->SM3#1 (Read SM)
 - b. FMMU2(0x1010:0x1017)->SM2 #2 (Write SM) FMMU3(0x1018:0x101F)->SM3#2 (Read SM)

This workaround is suboptimal, as it requires an increased TX_START_DELAY. Increasing in the process path latency may impact cycle time as the number of slaves in the network increases. Also, with TX_START_DELAY increases, the minimum IPG is maintained at 90 ns more than the TX_START_DELAY, which might be challenging in a large network of slaves. Thus, the workaround 2 overlapping FMMU/SM configuration is highly recommended.

3.6.2 Root Cause of PINDSW-141

When firmware needs to switch from one FMMU/SM to second FMMU/SM, there is significant overhead in terms of the cycle's requirement for PRUs. Such an overhead does not apply in overlapped use case as both FMMU/Read SM and FMMU/Write SM information is loaded during processing of datagram header.

Consider the case of two slaves with TwinCAT that uses 5 bytes output and 7 bytes input data (TI ESC default configuration).

The master creates one LRW datagram of size 14 bytes accessing two slaves (see [Figure 11](#)).



Figure 11. 14 Bytes Long Accessing Two Slaves

The original SOEM master default LRW command is 24 bytes long accessing two slaves (see [Figure 12](#)).

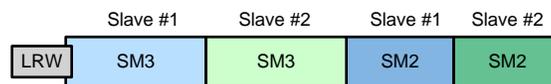


Figure 12. 24 Bytes Long Accessing Two Slaves

As the number of slaves increases, this becomes suboptimal and is not significantly different from the LRD/LWR mode. LRD/LWR takes the 12 bytes packet header with the WC overhead compared to original SOEM default independent of number of slaves in the network.

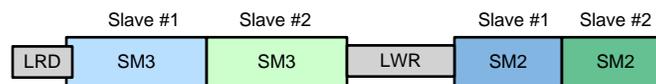


Figure 13. LRD/LWR Accessing Two Slaves

3.6.3 Fix of PINDSW-141 Errata by the SOEM Master and IgH Master

The PINDSW-141 errata is understood by the SOEM society and fixed by the API `ecx_config_overlap_map()`, which internally calls the `ecx_config_overlap_map_group()` to map all PDOs in one group of slaves to IOmap with Outputs/Inputs overlapping to emulate the TwinCAT configuration. This is the workaround 2 – the optimal configuration and highly recommended.

```
/** Map all PDOs in one group of slaves to IOmap with Outputs/Inputs
 * overlapping. NOTE: Must use this for TI ESC when using LRW.
 *
 * @param[in] context = context struct
 * @param[out] pIOmap = pointer to IOmap
 * @param[in] group = group to map, 0 = all groups
 * @return IOmap size
 */
int ecx_config_overlap_map_group(ecx_contextt *context, void *pIOmap, uint8 group)
```

The overlapping PDO support has been added in a branch of the IgH EtherCAT master as well.

3.6.4 LRW Access to One Slave With Non-Overlapping FMMU/SM

Although the one slave use case is not explicitly listed in the PINDSW-141, switching from one FMMU/SM to a second FMMU/SM adds significant overhead in terms of the cycle's requirement for PRUs, as explained in [Section 3.4](#). Thus, the overlapping FMMU/SM for PDO is still required. With non-overlapping FMMU/SM as shown in the Ethernet trace of topology #1 and default `TX_START_DELAY` value, malformed packets and zero data still occur.

3.7 **DDRless and ASIC (ET1100 Compatible) Mode**

The EtherCAT slave package provides support for application execution using only on-chip memory (DDRless on AMIC11x and AMIC12x SoCs). Also, the TI ESC SPI Slave mode (ASIC, ET1100 compatible mode) on AMIC11x supports on-chip memory execution.

Several binaries need to be flashed to SPI flash properly in order to bring up DDRless EtherCAT on AMIC11x. If failure to do so, the program will likely be running into dead loop/data abort exception default handler in ROM and 0x0002008C will be shown in the PC register of ARM core.

The how-to videos of [Demonstrating DDR-less EtherCAT Slave on AMIC110](#) and [Demonstrating DDR-less EtherCAT Slave on AMIC110 with C2000](#) describe detailed instructions.

4 **References**

- [Ethernet PHY Configuration Using MDIO for Industrial Applications](#)
- [DDR-less EtherCAT® Slave on AMIC110 Reference Design](#)
- [Demonstrating DDR-less EtherCAT® Slave on AMIC110](#)
- [Running and Configuring EtherCAT Slave](#) wiki
- [Demonstrating TI ESC SPI Mode DDR-less AMIC110 with C2000 EtherCAT Slave](#)
- [PRU_ICSS EtherCAT Firmware API Guide](#) wiki
- [EtherCAT Slave Stack Code Application Note ET9300](#)
- [Running AM335x EtherCAT Application in DC Mode](#) wiki

PRU-ICSS Debug via CCS

A.1 PRU-ICSS Debug via CCS

Instructions on how to load and debug and ARM application in CCS are shared in the *CCS Debug* section of the [Industrial Protocol Package Getting Started Guide](#) wiki.

The PRU-ICSS EtherCAT slave firmware is distributed as a binary. No source code is included in PRU-ICSS EtherCAT slave software package. Debugging the PRU-ICSS EtherCAT slave firmware would need TI's experts help. The E2E forum is a great place to get help. However, in some cases, you might need to gather PRU's information in order for TI to analyze. Below steps show how to connect and debug PRU cores.

1. Halt ARM core, right click on the PRU_0 or PRU_1 and select "Connect Target". Then, suspend and view Disassembly. [Figure 14](#) shows PRU_0 and PRU_1 cores in CCS.

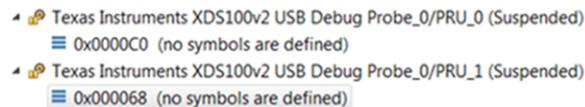


Figure 14. PRU_0 and PRU_1 Cores in CCS

2. Disassembly view in CCS. Step into and Step over can be used. [Figure 15](#) shows example of PRU_1 Disassembly code.



Figure 15. PRU_1 Disassembly Dode

A.1.1 PRU-ICSS Memory Dump

If required for debugging, example for comparing between releases or comparing between a working and a non-working version, users can get a memory dump via CCS. Steps below:

1. In CCS go to "View" and "Memory Browser".
2. For dumping memory, in memory browser window go to save memory.

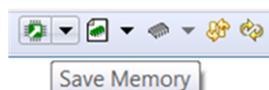


Figure 16. Save Memory

3. Select a file name and location. Save as *.dat. Go to “next”.
4. In the start address, use the PRU-ICSS base address (for example, 0x4A300000 for AM335x) + desired offset. Type the size as “Number of words to read”.
5. For memory map details, see the device-specific Technical Reference Manual. [Table 7](#) was taken from the [AM335x and AMIC 110 Sitara™ Processors Technical Reference Manual](#).

Table 7. PRU-ICSS Global Memory Map

Offset Address	PRU-ICSS
0x0000-0000	Data 8KB RAM 0
0x0000-2000	Data 8KB RAM 1
0x0001-0000	Shared Data 12KB RAM 2
0x0002-0000	INTC
0x0002-2000	PRU0 Control
0x0002-2400	PRU0 Debug
0x0002-4000	PRU1 Control
0x0002_4400	PRU1 Debug
0x0002_6000	CFG
0x0002_8000	UART 0
0x0002_A000	Reserved
0x0002_C000	Reserved
0x0002_E000	IEP
0x0003_0000	eCAP 0
0x0003_2000	MII_RT_CFG
0x0003_2400	MII_MDIO
0x0003_4000	PRU0 8KB IRAAM
0x0003_8000	PRU1 8KB IRAM
0x0004_0000	Reserved

Testing EtherCAT Slave With a Master on Sitara Processors

B.1 Testing EtherCAT Slave With a Master on Sitara Processors

B.1.1 TwinCAT

TwinCAT is PC-based EtherCAT controller from Beckhoff. Instructions on how to test the EtherCAT slave with TwiCAT can be found in TI's [PRU-ICSS EtherCAT slave User Guide](#) wiki.

B.1.2 acontis EC-Master

EC-Master can run on PRU-ICSS/CPSW ports on AM335x and AM57x boards. EC-Master can be download from [EtherCAT master Evaluation for TI Processors](#). Additional details can be found in [EEtherCAT® Master Reference Design for AM57x Gigabit Ethernet and PRU-ICSS Design Guide](#) for AM57x and [EtherCAT Master Stack for TI Sitara CPU Family Design Guide Design Guide](#) for AM335x. Also, the [how-to video](#) shows how to build, run and test the EC-Master on TI's devices.

Adding/Changing PDO

C.1 Adding/Changing PDO

Two possible methods can be used.

- Option 1: Modify .xml and use SSC OD tool to create SSC based application files. This method is described on Beckhoff's Application Note ET9300
- Option 2: Manually modify tiescappl.h and tiescappl.c. Below shows an example of changes required for adding TxPDO with two entries. For this example we mapped 0x6020 -> 0x1A02 -> 0x1C13
 - Add a new TxPDO object in tiescappl.h, by creating the entry description, the object name and a struct variable for process data.

```

/*****
*                               Object 0x1A02: My new input TxPDO
*****
#ifdef _OBJD_
/* Entry description */
OBJCONST TSDOINFORMATIONDESC OBJMEM asEntryDesc0x1A02[] = {
  {DEFTYPE_UNSIGNED8, 0x8, ACCESS_READ }, /* SubIndex 000 */
  {DEFTYPE_UNSIGNED32, 0x20, ACCESS_READ}, /* SubIndex 001: SubIndex 001 */
  {DEFTYPE_UNSIGNED32, 0x20, ACCESS_READ}}; /* SubIndex 002: SubIndex 002 */

/* Name of the object and the entries */
OBJCONST UCHAR OBJMEM aName0x1A02[] = "TxPDO1-Map\000SubIndex 1\000SubIndex 2\000\377";
#endif //ifdef _OBJD_

/* Structure to handle the object data*/
typedef struct STRUCT_PACKED_START {
  UINT16  u16SubIndex0;
  UINT32  aEntries[2];
} STRUCT_PACKED_END
TOBJ1A02;

/* Variable to handle the object data */
PROTO TOBJ1A02 sNewTxPDO1Map
#ifdef _TIESC_HW_
= {0x02, {0x60200120, 0x60200210}}
#endif
;

```

- Add a new TxPDO object in tiescappl.h for the Sync Management assignment.

```

/*****
*                               Object 0x1C13: TxPDO assignment
*****
#ifdef _OBJD_
OBJCONST TSDOINFORMATIONDESC OBJMEM asEntryDesc0x1C13[] =
{
  {DEFTYPE_UNSIGNED8, 0x08, ACCESS_READ | ACCESS_WRITE_PREOP},
  {DEFTYPE_UNSIGNED16, 0x10, ACCESS_READ | ACCESS_WRITE_PREOP},
  {DEFTYPE_UNSIGNED16, 0x10, ACCESS_READ | ACCESS_WRITE_PREOP},
  {DEFTYPE_UNSIGNED16, 0x10, ACCESS_READ | ACCESS_WRITE_PREOP}
};
OBJCONST UCHAR OBJMEM aName0x1C13[] = "TxPDO assign";
#endif //ifdef _OBJD_

```

```
typedef struct STRUCT_PACKED_START
{
    UINT16    ul6SubIndex0;
    UINT16    aEntries[3];
} STRUCT_PACKED_END
TOBJ1C13;
```

```
PROTO TOBJ1C13 sTxPDOassign
#ifdef _TIESC_HW_
    = {0x03, {0x1A00, 0x1A02, 0x1A03}}
#endif
```

– Add a new Input Data object in tiescappl.h.

```

/*****
 *
 *          Object 0x6020: New input object
 *****/
#ifdef _OBJD_
OBJCONST TSDOINFORMENTRYDESC  OBJMEM asEntryDesc0x6020[] = {
    {DEFTYPE_UNSIGNED8, 0x8, ACCESS_READ }, /* Subindex 000 */
    {DEFTYPE_INTEGER32, 0x20, ACCESS_READ | OBJACCESS_TXPDOMAPPING}, /* SubIndex 001: Info 1 */
    {DEFTYPE_INTEGER16, 0x10, ACCESS_READ | OBJACCESS_TXPDOMAPPING}}; //, /* SubIndex 002: Info
2 */

OBJCONST UCHAR OBJMEM aName0x6020[] = "New Inputs\000NewInput\000NewTest\000\377";
#endif //ifdef _OBJD_

typedef struct STRUCT_PACKED_START {
    UINT16    ul6SubIndex0;
    INT32     new_input;
    INT16     new_test;
} STRUCT_PACKED_END
TOBJ6020;

PROTO TOBJ6020 sNewInputs
#ifdef _TIESC_HW_
    = {0x02, 0x00000000, 0x0000}
#endif
;
```

– Add/update the references in the object dictionary.

```
TOBJECT OBJMEM ApplicationObjDic[] =
{
    /* Object 0x1603 */
    {NULL, NULL, 0x1603, {DEFTYPE_PDOMAPPING, 2 | (OBJCODE_REC << 8)}, asEntryDesc0x1603,
aName0x1603, &RxMyPDO1Map, NULL, NULL, 0x0000 },
    /* Object 0x1A00 */
    {NULL, NULL, 0x1A00, {DEFTYPE_PDOMAPPING, 1 | (OBJCODE_REC << 8)}, asEntryDesc0x1A00,
aName0x1A00, &TxPDOmap, NULL, NULL, 0x0000 },
    /* Object 0x1A02 */
    {NULL, NULL, 0x1A02, {DEFTYPE_PDOMAPPING, 2 | (OBJCODE_REC << 8)}, asEntryDesc0x1A02,
aName0x1A02, &sNewTxPDO1Map, NULL, NULL, 0x0000 },
    /* Object 0x1A03 */
    {NULL, NULL, 0x1A03, {DEFTYPE_PDOMAPPING, 2 | (OBJCODE_REC << 8)}, asEntryDesc0x1A03,
aName0x1A03, &sAITxPDO1Map, NULL, NULL, 0x0000 },
    /* Object 0x1C12 */
    {NULL, NULL, 0x1C12, {DEFTYPE_UNSIGNED16, 3 | (OBJCODE_ARR << 8)}, asEntryDesc0x1C12,
aName0x1C12, &sRxPDOassign, NULL, NULL, 0x0000 },
    /* Object 0x1C13 */
    {NULL, NULL, 0x1C13, {DEFTYPE_UNSIGNED16, 3 | (OBJCODE_ARR << 8)}, asEntryDesc0x1C13,
aName0x1C13, &sTxPDOassign, NULL, NULL, 0x0000 },
    /* Object 0x6000 */
    {NULL, NULL, 0x6000, {DEFTYPE_RECORD, 1 | (OBJCODE_REC << 8)}, asEntryDesc0x6000,
aName0x6000, &sDIInputs, NULL, NULL, 0x0000 },
    /* Object 0x6020 */

```

```
{NULL, NULL, 0x6020, {DEFTYPE_RECORD, 2 | (OBJCODE_REC << 8)}, asEntryDesc0x6020,  
aName0x6020, &sNewInputs, NULL, NULL, 0x0000 },
```

– Modify test application code in tiescappl.c.

```
for(j = 0; j < sTxPDOassign.ul6SubIndex0; j++)  
{  
    switch(sTxPDOassign.aEntries[j])  
    {  
        case 0x1A00:  
            *pTmpData++ = sDIInputs.switches;  
            break;  
        case 0x1A02:  
            *pTmpData++ = sNewInputs.new_input & 0xFF;  
            *pTmpData++ = (sNewInputs.new_input & 0xFF00) >> 8;  
            *pTmpData++ = (sNewInputs.new_input & 0xFF0000) >> 16;  
            *pTmpData++ = (sNewInputs.new_input & 0xFF000000) >> 24;  
            *pTmpData++ = sNewInputs.new_test & 0xFF;  
            *pTmpData++ = (sNewInputs.new_test & 0xFF00) >> 8;  
            break;  
    }  
}
```

FAQ

D.1 FAQ

1. **What are the ESC functions supported by PRU-ICSS in the current solution (for example, FMMU, sync-manager, distributed clock, and so forth)?**

This is covered in the EtherCAT slave data sheet ([EtherCAT_SDK]docs). For more details, see the [PRU_ICSS EtherCAT Firmware API Guide](#) wiki.

2. **Is it possible to modify virtual EEPROM with master? Can the configuration data be modified by master without going through other procedures?**

Yes, the EEPROM modification is supported by the API function:

```
/* @brief Flush the EEPROM cache to non-volatile storage (say SPI flash) if available */
PROTO void bsp_eeprom_emulation_flush(void);
```

And the call flow looks like

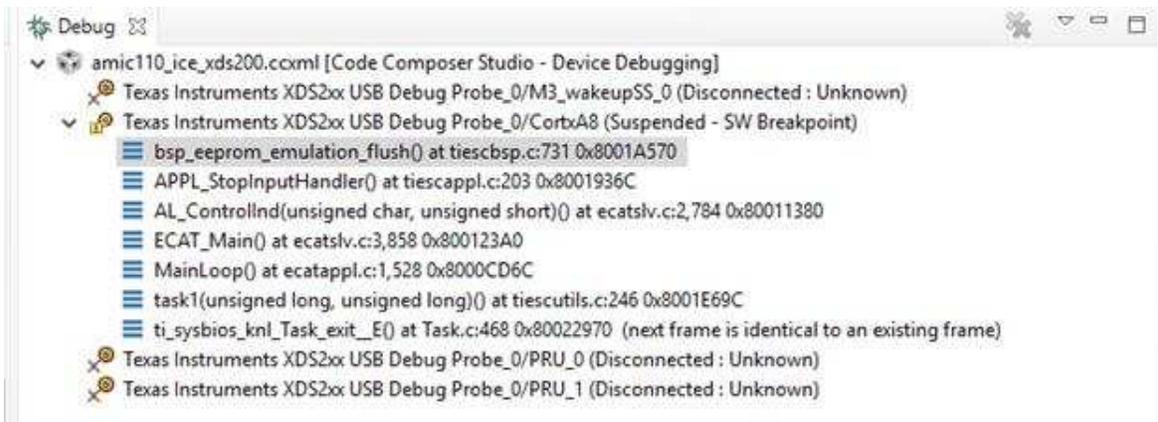


Figure 17. EEPROM Flush Call Flow

3. **What is the DC sync timing jitter for AMIC110? How do I measure the jitter correctly?**

[Section 3.5](#) shows 23 ns jitter from AM335x ICEv2. AMIC110 uses the same PRU-ICSS firmware as AM335x. The procedure of jitter measurement is described in the [Running AM335x EtherCAT Application in DC Mode](#) wiki.

4. **To add more objects to the object dictionary in C2000 EtherCAT demo project and xml file, does any document help accelerate the process?**

For more information, see the [EtherCAT Slave Stack Code Application Note ET9300](#).

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 (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
Copyright © 2018, Texas Instruments Incorporated