EtherCAT Slave Controller Software

USER'S GUIDE

Texas Instruments

Copyright

Texas Instruments
13905 University Boulevard
Sugar Land, TX 77479
http://www.ti.com/c2000

Revision Information

This is version v2.00.00.00 of this document, last updated on April 3, 2019.

Table of Contents

1 Introduction

The EtherCAT Slave Controller (ESC) hardware abstraction layer (HAL) drivers and examples are designed to operate on the EtherCAT hardware peripheral on F2838x devices. The F2838x devices support EtherCAT on either CPU1 or the Connectivity Manager (CM). Either core can be setup to be an ESC.

This user guide details how to setup the EtherCAT master software (TwinCAT) on your computer, provide details on the HAL driver APIs, and how to run the HAL driver test applications.

Minimum Requirements:
Code Composer Studio v9, C2000 Compiler v18.12.1.LTS, ARM Compiler v18.12.1.LTS

Chapter Overview:

1.1 Terms and Abbreviations

Term	Definition
CCS	Code Composer Studio
EtherCAT	Ethernet for Control Automation Technology
CM	Connectivity Manager
HAL	Hardware Abstraction Layer
PDI	Processor Data Interface
TwinCAT	EtherCAT master software from Beckhoff
ESC	EtherCAT Slave Controller
ESCSS	EtherCAT Slave Controller Subsystem
ENI	EtherCAT Network Information
ESI	EtherCAT Slave Information
API	Application Programming Interface

Table 1.1: Terms and Abbreviations

1.2 References

1. Texas Instruments, F2838x Technical Reference Manual
2. Beckhoff, ET9300 EtherCAT Slave Stack Code
3. ETG, EtherCAT Slave Implementation Guide

2 Getting Started

This chapter details how to get setup and begin using the EtherCAT examples. The EtherCAT software includes the following:

2.1 CPU1 PDI HAL Test Example Sets up EtherCAT for CPU1 and performs a test of the PDI HAL APIs. Additionally, the example can be used to get started with basic communication between the EtherCAT master and ESC

2.2 CPU1 Echoback Demo Example A precompiled demo example that demonstrates usage of the EtherCAT slave stack code and loops back data to the master from the slave.

2.3 CPU1 Echoback Solution Example An example solution framework, which requires EtherCAT slave stack code to be integrated, that performs a loop back of data to the master from the slave.

2.4 CPU1 Allocate ECAT to CM Example Configures the EtherCAT clock and GPIOs before allocating EtherCAT ownership to the CM

2.5 CM PDI HAL Test Example Sets up EtherCAT for CM and performs a test of the PDI HAL APIs. Additionally, the example can be used to get started with basic communication between the EtherCAT master and ESC

2.6 CM Echoback Demo Example A precompiled demo example that demonstrates usage of the EtherCAT slave stack code and loops back data to the master from the slave.

2.7 CM Echoback Solution Example An example solution framework, which requires EtherCAT slave stack code to be integrated, that performs a loop back of data to the master from the slave.

2.1 CPU1 PDI HAL Test Example

- This example sets up EtherCAT to be allocated to CPU1 and configures the required EtherCAT GPIOs and clocking. Additionally, the example performs a series of reads and writes to the full range of EtherCAT RAM using the HAL APIs. These can be observed from the CCS memory browser or TwinCAT ESC memory browser.
- This example is self-checking when performing the reads and writes. The following details the pass and fail signals:
  - Pass Signal - Both controlCARD LEDs (D1,D2) are on (not flashing)
  - Fail Signal - Both controlCARD LEDs (D1,D2) are flashing

Note: The intent of this project is to demonstrate the usage of the PDI. Therefore, no EtherCAT stack is included in this demo.

1. First, TwinCAT must be installed and setup. Refer to Section 3.2.
2. Check your external connections: Section 3.1
3. Open CCS and import the example `f2838x_cpu1_pdi_hal_test_app`
4. Select the RAM build configuration and build the example
5. Load the example to the controlCARD and run the code
   (a) **Important**: If the EtherCAT HAL example for the CM was loaded previously and the controlCARD hasn't been power cycled since, make sure to power cycle the controlCARD before running this example

6. If this your first time running any EtherCAT code (CPU1 or CM) on the controlCARD, the LEDs should be indicating a fail signal.
   (a) This failure is occurring because the minimum required EtherCAT EEPROM locations aren't programmed yet.
   (b) Refer to Section 3.4 on how to program the EEPROM
7. Once EEPROM is programmed or re-programmed for the correct core, reset the CPU and restart the example.
8. Set a breakpoint on `ESC_debugUpdateESCRegLogs()` in `pdi_test_app.c` and the CPU should hit the breakpoint. The pass signal should be indicated by the controlCARD LEDs. If not, pause the execution and investigate further.
9. The `ESC_debugUpdateESCRegLogs()` will continually update the `ESC_escRegs` data structure with the EtherCAT register and RAM values added for monitoring as part of `ESC_setupPDITestInterface()`. This data structure can be viewed using the CCS Expressions window.
10. You can now restart the example and set various breakpoints within `ESC_setupPDITestInterface()` to observe the reads/writes from CCS as well as the TwinCAT Master memory window. Additionally, you can change values via either interface to introduce failures in the PDI test. Refer to Section 3.5 for information on using the TwinCAT Master memory window.

2.2 CPU1 Echoback Demo Example

- This demo example is a precompiled demonstration of the EtherCAT slave stack code.
- This demo example emulates a bank of switches (inputs) and LEDs (outputs). The EtherCAT master controls the LEDs' states and the EtherCAT slave loops back the virtual LED signals into the virtual switches so that the master can read back the LED output state.

Note: To view the source code and/or debug this project using CCS, refer to the CPU1 Echoback Solution Example 2.3.

1. First, TwinCAT must be installed and setup. Refer to Section 3.2.
2. Check your external connections: Section 3.1
3. Run `ethercat_slave_ssc_and_demo_setup.exe` installer to extract the demo files into the EtherCAT examples directory.
4. Within CCS, verify to be in the CCS Debug view and connect to CPU1.
5. Once connected to CPU1, go to `Run -> Load -> Load Program` and select `f2838x_cpu1_echoback_demo_FLASH.out`. Then click `Resume`.
6. Copy the ESI file (`F2838x CPU1 EtherCAT Slave.xml`) into the TwinCAT directory (Default location: `C:/TwinCAT/3.1/Config/Io/EtherCAT`)
7. Refer to Section 3.4 on how to program the EEPROM.
8. Once EEPROM is programmed, do the following:
   (a) Disconnect and power cycle the board
   (b) Reload CPU1 application
   (c) Reconnect to board to TwinCAT, rescan for devices, and restart TwinCAT in `config mode`

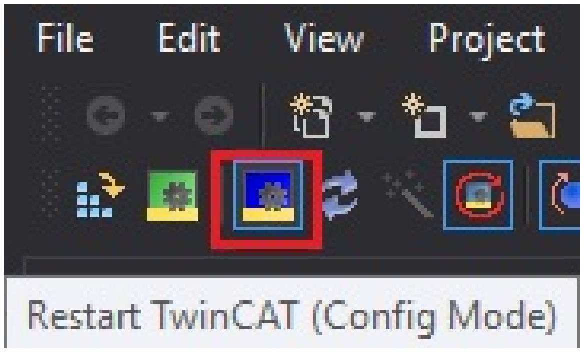

Figure 2.1: TwinCAT Restart in Config Mode Button

9. If you want to observe the variables in CCS, right-click within the CCS Expressions window and choose `Import`. Then select the `expressions_window_input_output_varaibles.txt` file.
10. Within TwinCAT, double-click on the discovered EtherCAT box and observe that the EtherCAT slave is running in OP mode.

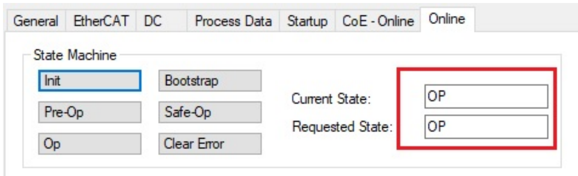

Figure 2.2: EtherCAT Slave in OP Mode

11. Within TwinCAT, expand the explorer to the EtherCAT box and find the various output/input mappings.

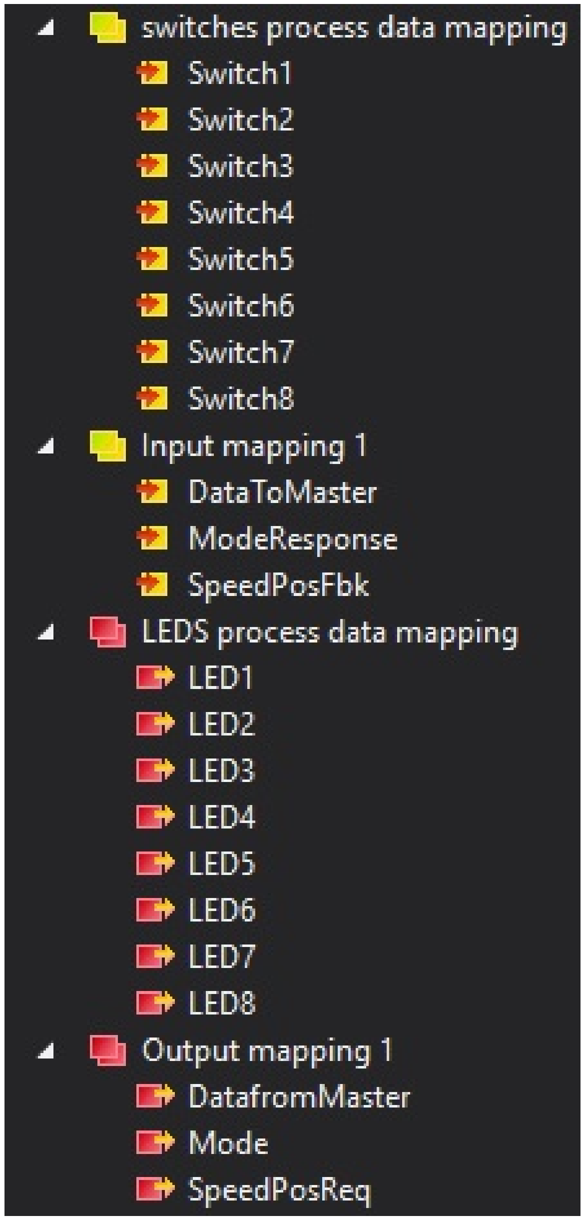

Figure 2.3: TwinCAT Solution Explorer Inputs and Outputs

(a) Select the `LEDS process data mapping` in the solution explorer and in the window on the right, you can change the value of any of the virtual LEDs. Switch to the `switches process data mapping` to see the looped back values. For example, if LED1 is set to 1, then Switch1 should also be 1.

(b) Select the `Output mapping 1` in the solution explorer and in the window on the right, you can change the values of the 3 data variables. Once set, the looped back value can be observed from the `Input mapping 1` variables.

2.3 CPU1 Echoback Solution Example

- This example requires application and slave stack files to be generated via the SSC tool before building/running.
- This example emulates a bank of switches (inputs) and LEDs (outputs). The EtherCAT master controls the LEDs' states and the EtherCAT slave loops back the virtual LED signals into the virtual switches so that the master can read back the LED output state.

1. First, TwinCAT must be installed and setup. Refer to Section 3.2.
2. Install the SSC tool V5.12
   (a) **Important**: Only V5.12 is supported. Only download this version.
   (b) Download at ETG SSC ET9300
3. Check your external connections: Section 3.1
4. Run `ethercat_slave_ssc_and_demo_setup.exe` installer to extract the F2838x SSC configuration and echoback application files required by the SSC tool. These will be located in the newly created `ssc_configuration` directory.
5. Open the SSC tool and a `New Project` dialog box will open. Select `Import` and locate the `f2838x_ssc_config.xml`. Then click `Open`.
6. Use the Custom drop-down menu to select `TI F2838x CPU1 Sample (Includes Sample Application)` and click `OK`.

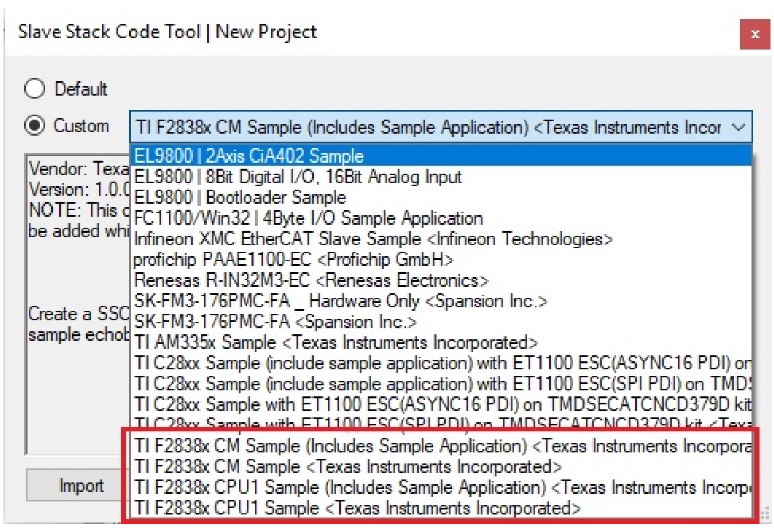

Figure 2.4: SSC Configuration Window

7. Click `Yes` when the pop up window asks about requiring external files to proceed.
8. Save the SSC project.
9. Within SSC tool, go to `Project -> Create new Slave Files`
   (a) Change the Source Folder directory to the `/examples/f2838x_cpu1_echoback_solution` directory.
   (b) Leave the ESI file directory location as is.
   (c) Click `Start` and then `OK`.

10. Import the example from /examples/f2838x_cpu1_echoback_solution into CCS and build it for RAM or FLASH.
11. Within CCS, verify to be in the CCS Debug view and connect to CPU1.
12. Once connected to CPU1, go to Run -> Load -> Load Program and select f2838x_cpu1_echoback_solution.out. Then click Resume.
13. Copy the ESI file (F2838x CPU1 EtherCAT Slave.xml) generated by the SSC tool into the TwinCAT directory (Default location: C:/TwinCAT/3.1/Config/Io/EtherCAT)
14. Refer to Section 3.4 on how to program the EEPROM.
15. Once EEPROM is programmed, do the following:
    (a) Disconnect and power cycle the board
    (b) Reload CPU1 application
    (c) Reconnect to board to TwinCAT, rescan for devices, and restart TwinCAT in config mode

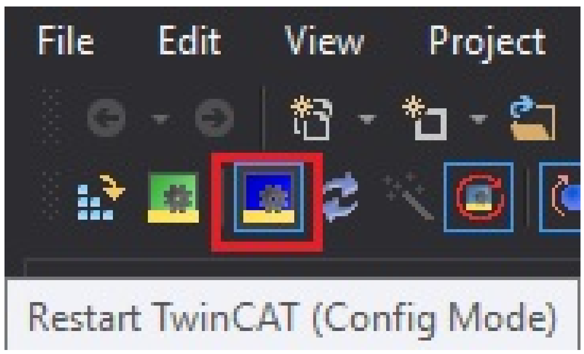

Figure 2.5: TwinCAT Restart in Config Mode Button

16. Within TwinCAT, double-click on the discovered EtherCAT box and observe that the EtherCAT slave is running in OP mode.

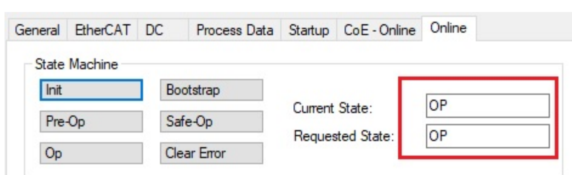

Figure 2.6: EtherCAT Slave in OP Mode

17. Within TwinCAT, expand the explorer to the EtherCAT box and find the various output/input mappings.

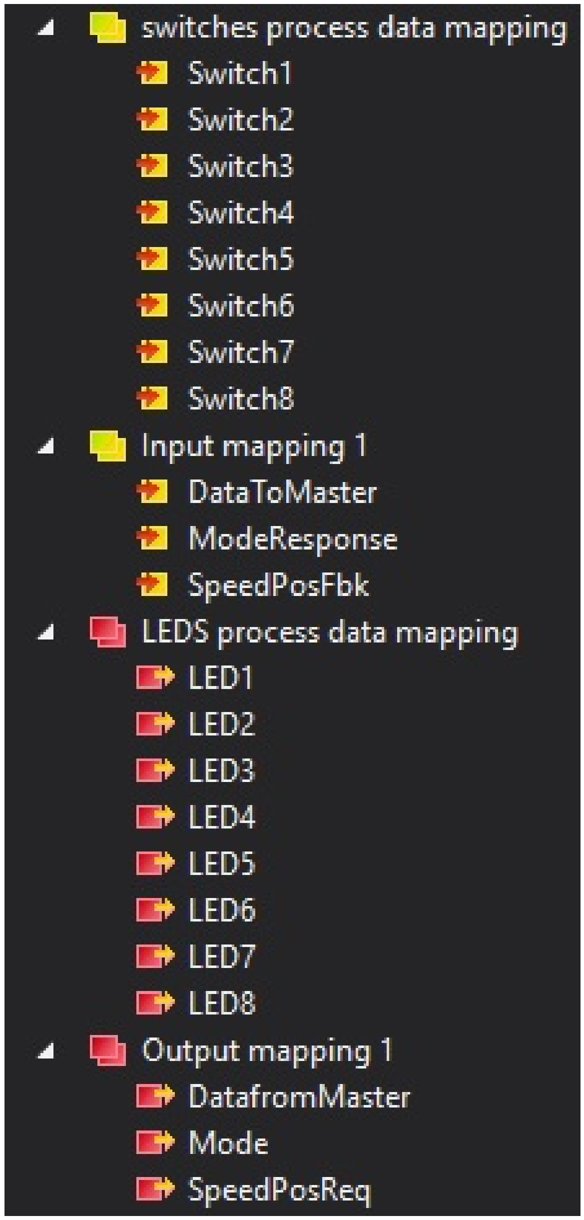

Figure 2.7: TwinCAT Solution Explorer Inputs and Outputs

(a) Select the `LEDS process data mapping` in the solution explorer and in the window on the right, you can change the value of any of the virtual LEDs. Switch to the `switches process data mapping` to see the looped back values. For example, if LED1 is set to 1, then Switch1 should also be 1.

(b) Select the `Output mapping 1` in the solution explorer and in the window on the right, you can change the values of the 3 data variables. Once set, the looped back value can be observed from the `Input mapping 1` variables.

2.4 CPU1 Allocate ECAT to CM Example

- This example sets up the EtherCAT required GPIOs and clocking then allocates the EtherCAT ownership to the Connectivity Manager (CM) core. Additionally, the example configures the CM clocks and releases the core to boot.

1. Check your external connections: Section 3.1
2. Open CCS and import the example `f2838x_cpu1_allocate_ecat_to_cm`
3. Select the RAM build configuration and build the example
4. Load the example to the controlCARD and run the code
   (a) **Important**: If the EtherCAT HAL example for CPU1 was loaded previously and the controlCARD hasn't been power cycled since, make sure to power cycle the controlCARD before running the CM HAL example
5. The example will perform the necessary setup and then enter an infinite loop.
6. The CM PDI HAL test example can now be run (if not already loaded) on the CM.

2.5 CM PDI HAL Test Example

- **Important**: The CPU1 example to allocate ECAT to the CM must be running before running this test example. (See Section 2.4)
- This example performs a series of reads and writes to full range of EtherCAT RAM using the HAL APIs. These can be observed from the CCS memory browser or TwinCAT ESC memory browser.
- This example is self-checking when performing the reads and writes. The following details the pass and fail signals:
  - Pass Signal - Both controlCARD LEDs (D1,D2) are on (not flashing)
  - Fail Signal - Both controlCARD LEDs (D1,D2) are flashing

Note: The intent of this project is to demonstrate the usage of the PDI. Therefore, no EtherCAT stack is included in this demo.

1. First, TwinCAT must be installed and setup. Refer to Section 3.2.
2. Check your external connections: Section 3.1
3. Open CCS and import the example `f2838x_cm_pdi_hal_test_app`
4. Select the RAM build configuration and build the example
5. Load the example to the controlCARD and run the code
6. If this your first time running any EtherCAT code (CPU1 or CM) on the controlCARD, the LEDs should be indicating a fail signal.
   (a) This failure is occurring because the minimum required EtherCAT EEPROM locations aren't programmed yet.
   (b) Refer to Section 3.4 on how to program the EEPROM
7. Once EEPROM is programmed or re-programmed for the correct core, reset the CPU and restart the example.
8. Set a breakpoint on `ESC_debugUpdateESCRegLogs()` in `pdi_test_app.c` and the CPU should hit the breakpoint. The pass signal should be indicated by the controlCARD LEDs. If not, pause the execution and investigate further.

9. The `ESC_debugUpdateESCRegLogs()` will continually update the `ESC_escRegs` data structure with the EtherCAT register and RAM values added for monitoring as part of `ESC_setupPDITestInterface()`. This data structure can be viewed using the CCS Expressions window.
10. You can now restart the example and set various breakpoints within `ESC_setupPDITestInterface()` to observe the reads/writes from CCS as well as the TwinCAT Master memory window. Additionally, you can change values via either interface to introduce failures in the PDI test. Refer to Section 3.5 for information on using the TwinCAT Master memory window.

2.6 CM Echoback Demo Example

- **Important**: The CPU1 example to allocate ECAT to the CM must be running before running this test example. (See Section 2.4)
- This demo example is a precompiled demonstration of the EtherCAT slave stack code.
- This demo example emulates a bank of switches (inputs) and LEDs (outputs). The EtherCAT master controls the LEDs' states and the EtherCAT slave loops back the virtual LED signals into the virtual switches so that the master can read back the LED output state.

Note: To view the source code and/or debug this project using CCS, refer to the CM Echoback Solution Example 2.7.

1. First, TwinCAT must be installed and setup. Refer to Section 3.2.
2. Check your external connections: Section 3.1
3. Run `ethercat_slave_ssc_and_demo_setup.exe` installer to extract the demo files into the EtherCAT examples directory.
4. Within CCS, verify to be in the CCS Debug view and connect to the CM core.
5. Once connected to the CM core, go to `Run -> Load -> Load Program` and select `f2838x_cm_echoback_demo_FLASH.out`. Then click `Resume`.
6. Copy the ESI file (`F2838x CM EtherCAT Slave.xml`) into the TwinCAT directory (Default location: `C:/TwinCAT/3.1/Config/Io/EtherCAT`)
7. Refer to Section 3.4 on how to program the EEPROM.
8. Once EEPROM is programmed, do the following:
   (a) Disconnect and power cycle the board
   (b) Reload CPU1 and CM applications
   (c) Reconnect to board to TwinCAT, rescan for devices, and restart TwinCAT in `config mode`

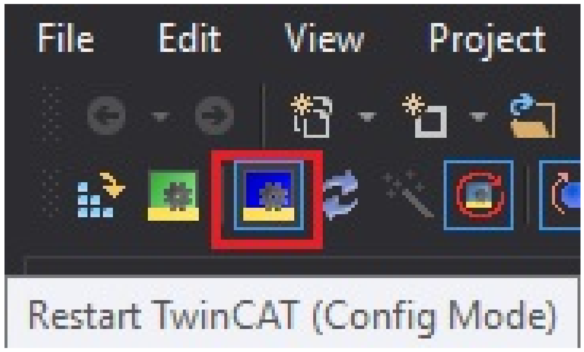

Figure 2.8: TwinCAT Restart in Config Mode Button

9. If you want to observe the variables in CCS, right-click within the CCS Expressions window and choose `Import`. Then select the `expressions_window_input_output_varaibles.txt` file.
10. Within TwinCAT, double-click on the discovered EtherCAT box and observe that the EtherCAT slave is running in OP mode.

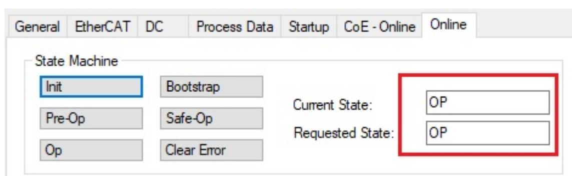

Figure 2.9: EtherCAT Slave in OP Mode

11. Within TwinCAT, expand the explorer to the EtherCAT box and find the various output/input mappings.

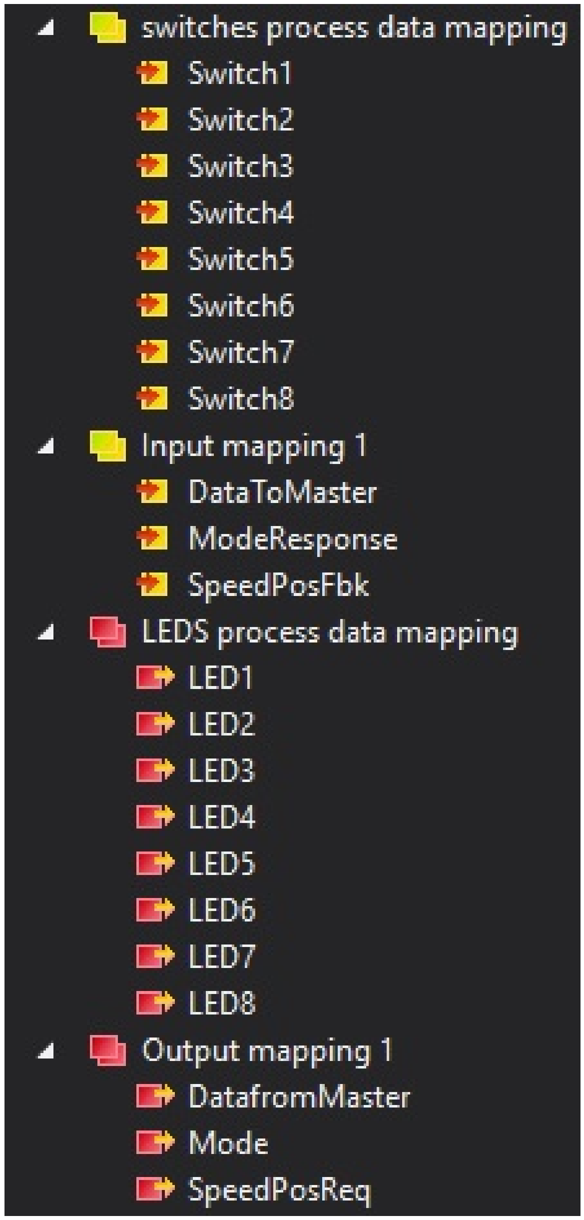

Figure 2.10: TwinCAT Solution Explorer Inputs and Outputs

(a) Select the `LEDS process data mapping` in the solution explorer and in the window on the right, you can change the value of any of the virtual LEDs. Switch to the `switches process data mapping` to see the looped back values. For example, if LED1 is set to 1, then Switch1 should also be 1.

(b) Select the `Output mapping 1` in the solution explorer and in the window on the right, you can change the values of the 3 data variables. Once set, the looped back value can be observed from the `Input mapping 1` variables.

2.7 CM Echoback Solution Example

- **Important**: The CPU1 example to allocate ECAT to the CM must be running before running this test example. (See Section 2.4)
- This example requires application and slave stack files to be generated via the SSC tool before building/running.
- This example emulates a bank of switches (inputs) and LEDs (outputs). The EtherCAT master controls the LEDs' states and the EtherCAT slave loops back the virtual LED signals into the virtual switches so that the master can read back the LED output state.

1. First, TwinCAT must be installed and setup. Refer to Section 3.2.
2. Install the SSC tool V5.12
   (a) **Important**: Only V5.12 is supported. Only download this version.
   (b) Download at ETG SSC ET9300
3. Check your external connections: Section 3.1
4. Run `ethercat_slave_ssc_and_demo_setup.exe` installer to extract the F2838x SSC configuration and echoback application files required by the SSC tool. These will be located in the newly created `ssc_configuration` directory.
5. Open the SSC tool and a `New Project` dialog box will open. Select `Import` and locate the `f2838x_ssc_config.xml`. Then click `Open`.
6. Use the Custom drop-down menu to select `TI F2838x CM Sample (Includes Sample Application)` and click `OK`.

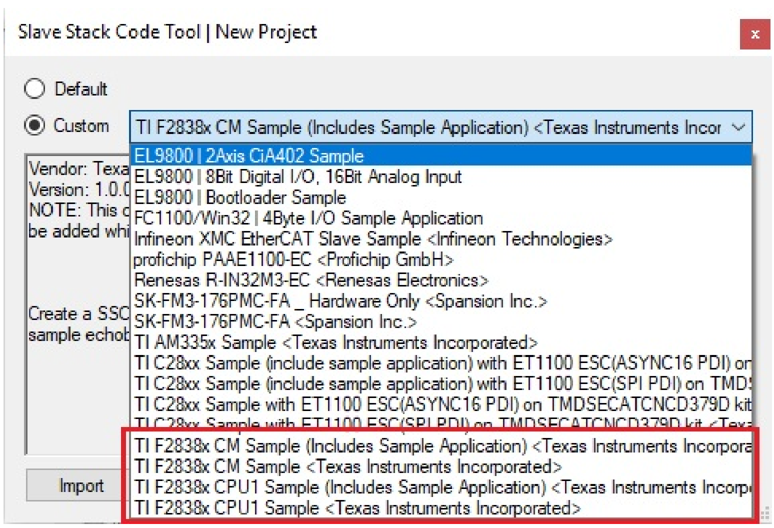

Figure 2.11: SSC Configuration Window

7. Click `Yes` when the pop up window asks about requiring external files to proceed.
8. Save the SSC project.
9. Within SSC tool, go to `Project -> Create new Slave Files`
   (a) Change the Source Folder directory to the `/examples/f2838x_cm_echoback_solution` directory.

   (b) Leave the ESI file directory location as is.
   (c) Click `Start` and then `OK`.

10. Import the example from `/examples/f2838x_cm_echoback_solution` into CCS and build it for `RAM` or `FLASH`.
11. Within CCS, verify to be in the CCS Debug view and connect to the CM core.
12. Once connected to the CM core, go to `Run -> Load -> Load Program` and select `f2838x_cm_echoback_solution.out`. Then click `Resume`.
13. Copy the ESI file (`F2838x CM EtherCAT Slave.xml`) generated by the SSC tool into the TwinCAT directory (Default location: `C:/TwinCAT/3.1/Config/Io/EtherCAT`)
14. Refer to Section 3.4 on how to program the EEPROM.
15. Once EEPROM is programmed, do the following:
    (a) Disconnect and power cycle the board
    (b) Reload CPU1 and CM applications
    (c) Reconnect to board to TwinCAT, rescan for devices, and restart TwinCAT in `config mode`

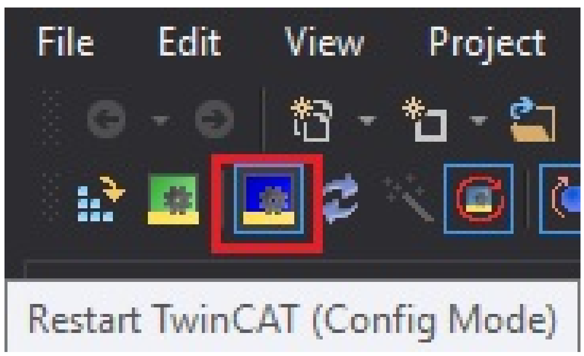

Figure 2.12: TwinCAT Restart in Config Mode Button

16. Within TwinCAT, double-click on the discovered EtherCAT box and observe that the EtherCAT slave is running in OP mode.

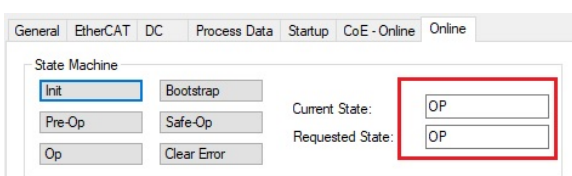

Figure 2.13: EtherCAT Slave in OP Mode

17. Within TwinCAT, expand the explorer to the EtherCAT box and find the various output/input mappings.

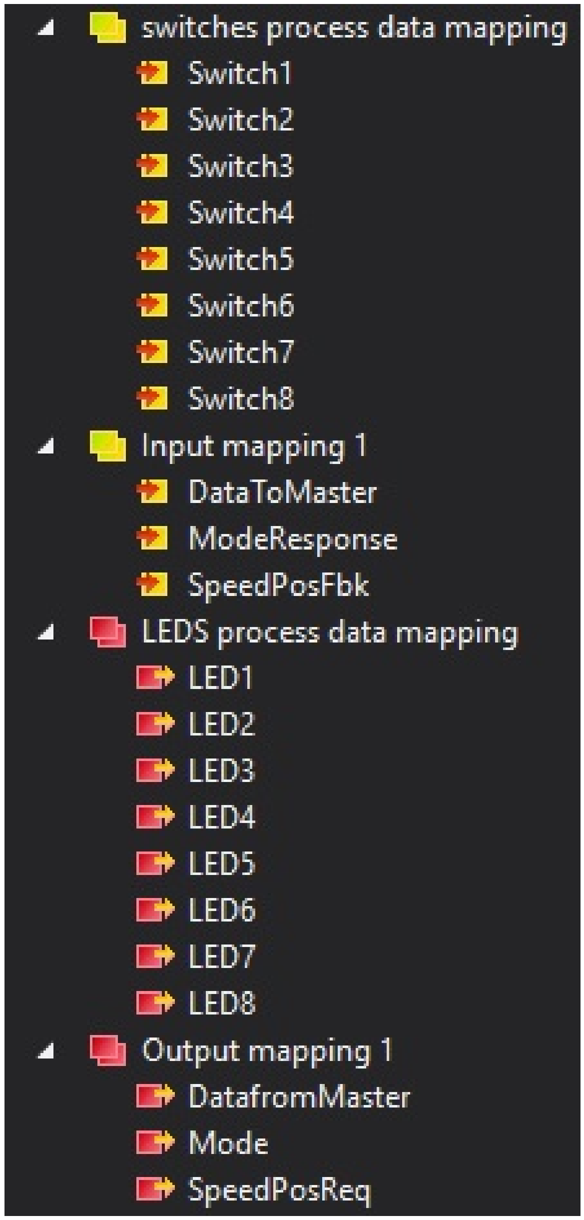

Figure 2.14: TwinCAT Solution Explorer Inputs and Outputs

(a) Select the `LEDS process data mapping` in the solution explorer and in the window on the right, you can change the value of any of the virtual LEDs. Switch to the `switches process data mapping` to see the looped back values. For example, if LED1 is set to 1, then Switch1 should also be 1.

(b) Select the `Output mapping 1` in the solution explorer and in the window on the right, you can change the values of the 3 data variables. Once set, the looped back value can be observed from the `Input mapping 1` variables.

3 How-To Procedures

3.1 Example External Connections

Example	External Connections
CPU1 PDI HAL Test	Mini USB connection between controlCARD and computer Ethernet cable connected to controlCARD RJ45 Port 0 and to computer
CPU1 Echoback Demo	Mini USB connection between controlCARD and computer Ethernet cable connected to controlCARD RJ45 Port 0 and to computer
CPU1 Echoback Solution	Mini USB connection between controlCARD and computer Ethernet cable connected to controlCARD RJ45 Port 0 and to computer
CPU1 Allocate to CM	Mini USB connection between controlCARD and computer
CM PDI HAL Test	Mini USB connection between controlCARD and computer Ethernet cable connected to controlCARD RJ45 Port 0 and to computer
CM Echoback Demo	Mini USB connection between controlCARD and computer Ethernet cable connected to controlCARD RJ45 Port 0 and to computer
CM Echoback Solution	Mini USB connection between controlCARD and computer Ethernet cable connected to controlCARD RJ45 Port 0 and to computer

Table 3.1: Example External Connections

3.2 Setup TwinCAT

1. Optional: Install Microsoft Visual Studio. This isn't required since TwinCAT will install a Visual Studio shell if no Visual Studio installation is found.
   (a) Download and install Microsoft Visual Studio
   (b) TwinCAT supports integration into Visual Studio 2010/2012/2013/2015/2017
2. Download and install TwinCAT3 from the Beckhoff
   (a) Follow the left sidebar to `Download->Software->TwinCAT 3->TE1xxx | Engineering` and select the software product `TwinCAT 3.1 eXtended Automation Engineering (XAE)`
3. Once installation is complete, verify that the TwinCAT Runtime is active
   (a) Check that the TwinCAT Config Mode icon is shown in the Windows notification panel. Right click on this icon and select `Tools->TwinCAT Switch Runtime`. From the Tc-SwitchRuntime window, verify that it is `active`. When active, it will only provide the option to `Deactivate`. Don't Deactivate!

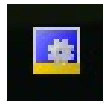

Figure 3.1: TwinCAT Config Mode Icon

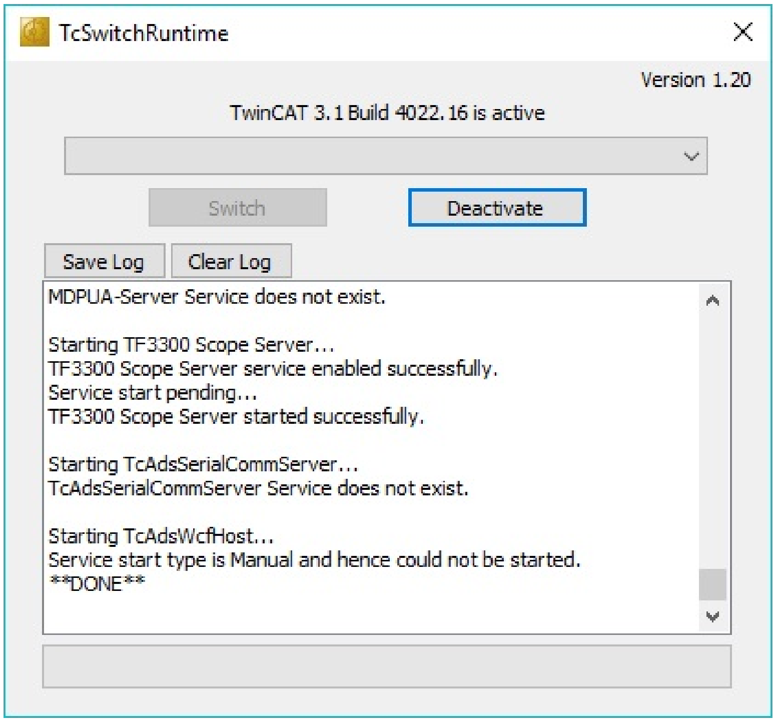

Figure 3.2: TcSwitchRuntime Window Activated

   (b) If the icon isn't present, then locate the TwinCAT Runtime executable from the file system. (Default installation location is typically: `C:/TwinCAT/TcSwitchRuntime`)

4. Start up Visual Studio with TwinCAT using one of the following methods:
   (a) Recommended: Right click the TwinCAT Config Mode icon from the Windows notification panel and select `TwinCAT XAE`
   (b) Use installed desktop icon: `TwinCAT XAE`
   (c) Use installed Start Menu icon under Beckhoff folder: `TwinCAT XAE`
5. Once Visual Studio running, verify that the main toolbar has options `TwinCAT` and `PLC` shown. If these aren't present, then the TwinCAT Switch Runtime isn't active.
6. Within Visual Studio, create a new EtherCAT project. Select `File -> New -> Project` and under templates select `TwinCAT Projects` then `TwinCAT XAE Project (XML format)`. Fill in a name and click `OK`.
7. Now that the project is created, verify that a realtime Ethernet adapter is installed.
   (a) In Visual Studio, select the `TwinCAT` menu from the main toolbar and select `Show Realtime Ethernet Compatible Devices`
   (b) In the popup window, under `Installed and ready to use devices(realtime capable)` category, if no connections are shown, select one from the list of Compatible devices and click `Install`.

8. TwinCAT setup is complete.

3.3 Scanning for EtherCAT Devices via TwinCAT

1. Open the TwinCAT project created via Section 3.2
2. Verify that the controlCARD is running the HAL example code and that the development computer (running TwinCAT) is connected via an Ethernet cable to the port 0 connection on the controlCARD.
   (a) Port 0 is the top Ethernet port on the side of the controlCARD with two Ethernet connections.
3. In Visual Studio on the left side solution explorer, expand the `Project`, then expand `I/O`
4. Right click on `Devices` and select `Scan`
   (a) A dialog will popup stating that `Not all types of devices can be found automatically`. Click `OK`.
5. Once scanning is complete, a popup window will appear. The following options may appear:
   (a) A popup stating that `1 new I/O devices found` where the device is `Device 2 (EtherCAT Automation Protocol)`. This or any other device numbers besides `Device 1` is correct, click `OK`.
   (b) A popup stating that no devices have been found or stating that `1 new I/O devices found` where the device is `Device 1 (EtherCAT Automation Protocol)`. This means some setup is incorrect. Verify that the example is running on the device (or at least has gone through the GPIO setup and reset of the EtherCAT IP). Then check the Section 4 for troubleshooting.
6. After clicking `OK`, another popup will ask to `Scan for boxes`. Click `Yes`.
7. After clicking `YES`, another popup will ask to `Activate Free Run`. Click `Yes`.
8. In the solution explorer on the left, under devices you should see `Device 2 (EtherCAT)`. Under that, there will be a `Box #`. This `Box` is the controlCARD ESC.

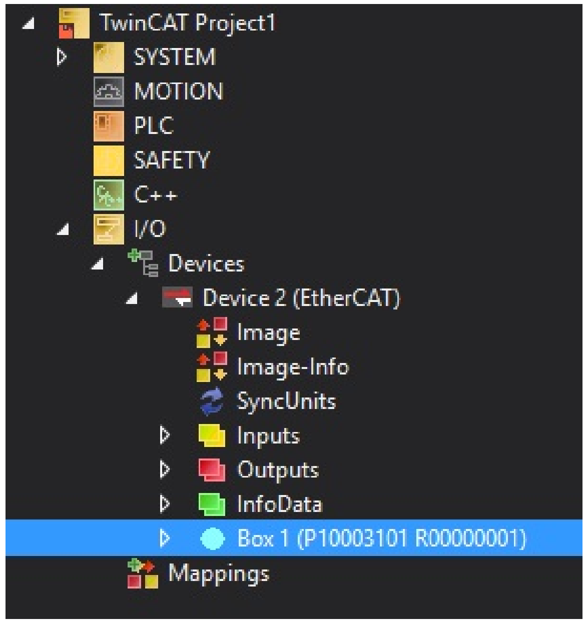

Figure 3.3: TwinCAT Solution Explorer

9. The EtherCAT master communication is now setup with the slave device.

3.4 Program ControlCard EEPROM

Verify first that TwinCAT has discovered the ESC (See Section 3.3 for steps)

1. In the Visual Studio solution explorer, double click on `Box #` under `Device 2 (EtherCAT).`
2. The TwinCAT project window should be open to the right of the solution explorer and have some tabs such as `General`, `EtherCAT`, etc
3. Select the `EtherCAT` tab and then click on `Advanced Settings`
4. In the new window, expand the `ESC Access` menu, then expand the `E2PROM` menu. Click on `Smart View`
5. If running a HAL PDI test Example, follow these steps to begin programming the EEPROM. Otherwise, skip to the next step.
   (a) Click on `Write E2PROM` and select `Browse`. Browse to `/C2000ware_X_XX_XX_XX/libraries/communications/ethercat/c28/eeprom` and select `f2838x_cpu1_pdi_test_app.bin` if running the CPU1 example or select `f2838x_cm_pdi_test_app.bin` if running the CM example. Click `OK`.
      i. Note that these BIN files only program the required first 15 bytes of EEPROM and should only be used with the HAL examples.
      ii. Additionally, either CPU1 or CM HAL example will work with either of the EEPROM files provided but for identification purposes two are provided so from the TwinCAT master, the user can identify which core is controlling the EtherCAT IP.
6. If running an Echoback example, follow these steps to begin programming the EEPROM.
   (a) Click on `Write E2PROM` and expand the `Texas Instruments Incorporated` menu within the `Available EEPROM Descriptions` window.

(b) Expand `TI C28xx Slave Devices` and select `F2838x CM EtherCAT Slave`. Click `OK`.

7. Visual Studio will indicate that the EEPROM is being programmed. When it completes, if the Smart View doesn't automatically update with the new contents, you can select `Read E2PROM` to read back the newly programmed values.
8. The Product Code for CPU1 is `0x10003201` and the Product Code for CM is `0x10003101`

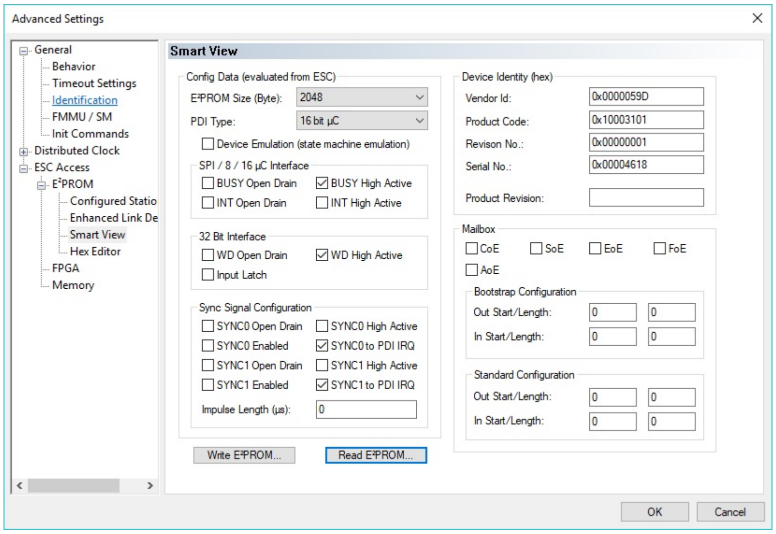

Figure 3.4: TwinCAT EEPROM Window

3.5 Use TwinCAT Memory Window

Verify first that TwinCAT has discovered the ESC (See Section 3.3 for steps)

1. In the Visual Studio solution explorer, double click on `Box #`.
2. The TwinCAT project window should be open and have some tabs such as `General`, `EtherCAT`, etc
3. Select the `EtherCAT` tab and then click on `Advanced Settings`
4. In the new window, expand the `ESC Access` menu, then select `Memory`. This is the connected ESC memory.

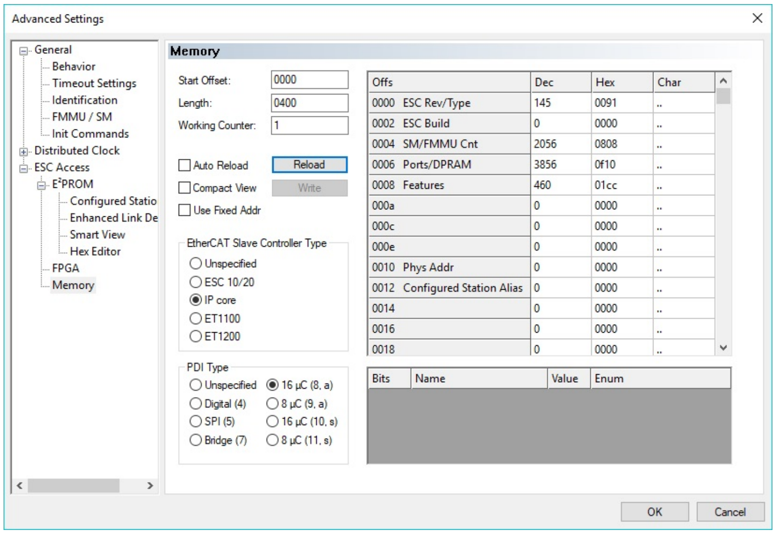

Figure 3.5: TwinCAT Memory Window

5. Adjust the `Start Offset` and `Length` as necessary to view the ESC registers or RAM. Note that these are byte offsets.
   (a) ESC registers are `0x0` to `0xFFF`
   (b) ESC RAM is `0x1000` to `0x4FFF`
6. You can select `Reload` once the offsets are changed or if the ESC is changing memory that needs to be reflected here on the Master side.
7. Additionally, the memory values can be manipulated through this window and can be applied once the `Write` button is selected. Such changes can be confirmed by viewing the same memory through the CCS memory browser.

3.6 Generate Slave Stack Code

These are steps to generate slave stack code without an application. If you are looking for instructions on generating for a specific F2838x EtherCAT example, refer to Getting Started Chapter 2.

1. Install the SSC tool V5.12
   (a) **Important**: Only V5.12 is supported. Only download this version.
   (b) Download at ETG SSC ET9300
2. Run `ethercat_slave_ssc_and_demo_setup.exe` installer to extract the F2838x SSC configuration and device system files required by the SSC tool. These will be located in the newly created `ssc_configuration` directory.
3. Open the SSC tool and a `New Project` dialog box will open. Select `Import` and locate the `f2838x_ssc_config.xml`. Then click `Open`.
4. Use the Custom drop-down menu to select `TI F2838x CPU1 Sample` or `TI F2838x CM Sample` and click `OK`.

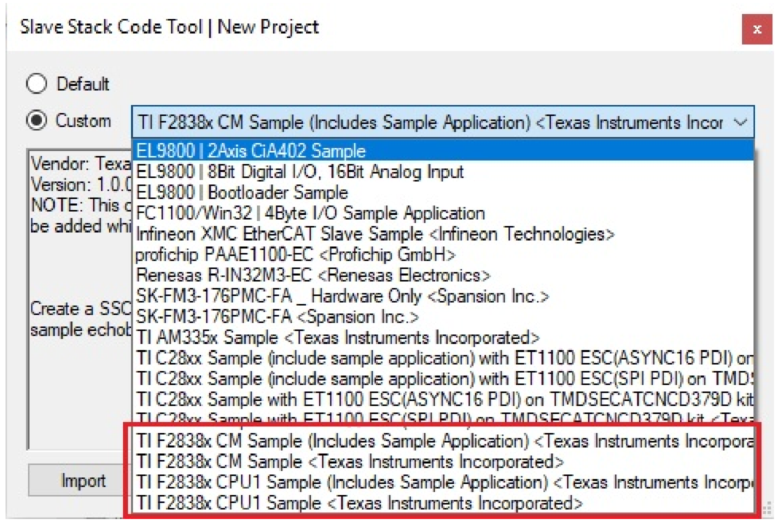

Figure 3.6: SSC Configuration Window

5. Click `Yes` when the pop up window asks about requiring external files to proceed.
6. Save the SSC project.
7. Within SSC tool, go to `Project -> Create new Slave Files`
   (a) You can leave both source and ESI directory paths as default
   (b) Click `Start` and then `OK`.

4 Troubleshooting

This chapter details some common issues that can cause the user trouble when using TwinCAT or the examples.

Problem: "Example won't import into Code Composer Studio (CCS)"
Solutions:

- Verify that you have the latest `C2000 Device Support Package` installed within CCS. In CCS go `Help` -> `Check for Updates`.
- Verify that you have the minimum required compiler versions installed for both C2000 and ARM. In CCS go `Help` -> `Install Code Generation Compiler Tools`.
- If you've previously imported an example into CCS, deleted it, and can't import it again, verify that the example is completely deleted from the CCS workspace (not just the CCS Project Explorer).

Problem: "EtherCAT network fails to initialize when running TwinCAT"
(This can include: "Reload Devices" fails, "Scan" for devices fails, "Restart EtherCAT in config mode" fails)
Solutions:

- Power-cycle the controlCARD
- Confirm that the EtherCAT example is loaded and running
- Verify that a realtime Ethernet adapter is installed
  - In Visual Studio, select the `TwinCAT` menu from the main toolbar and select `Show Realtime Ethernet Compatible Devices`
  - In the popup window, under `Installed and ready to use devices(realtime capable)` category, if no connections are shown, select one from the list of Compatible devices and click `Install`.

Problem: "Example is getting stuck when attempting to enable ESCSS debug access"
Solutions:

- Power-cycle the controlCARD. This problem will occur when previously running EtherCAT from one core and then trying to run EtherCAT from another core without power-cycle.

Problem: "The EEPROM and slave stack examples are loaded but device won't go to OP mode"
Solutions:

- Restart TwinCAT in config mode

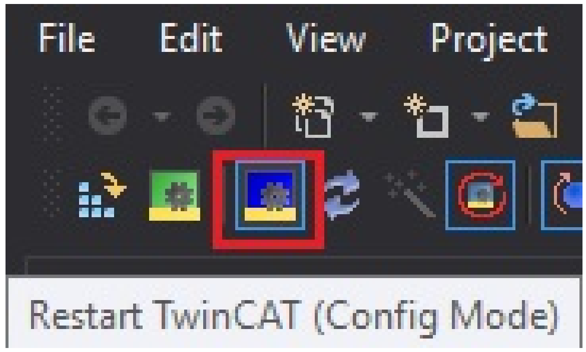

Figure 4.1: TwinCAT Restart in Config Mode Button

- Power-cycle the controlCARD, restart TwinCAT, and re-scan

5 ESC HAL APIs

This chapter details the CPU1 and CM ESC HAL APIs.

5.1 CPU1 HAL APIs

Functions

- __interrupt void ESC_applicationLayerHandler (void)
- __interrupt void ESC_applicationSync0Handler (void)
- __interrupt void ESC_applicationSync1Handler (void)
- void ESC_clearTimer (void)
- void ESC_debugAddESCRegsAddress (uint16_t address)
- void ESC_debugInitESCRegLogs (void)
- void ESC_debugUpdateESCRegLogs (void)
- uint32_t ESC_getTimer (void)
- void ESC_holdESCInReset (void)
- uint16_t ESC_initHW (void)
- uint16_t ESC_loadedCheckEEPROM (void)
- void ESC_passFailSignalSetup (void)
- void ESC_readBlock (ESCMEM_ADDR ∗pData, uint16_t address, uint16_t len)
- void ESC_readBlockISR (ESCMEM_ADDR ∗pData, uint16_t address, uint16_t len)
- uint32_t ESC_readDWord (uint16_t address)
- uint32_t ESC_readDWordISR (uint16_t address)
- uint16_t ESC_readWord (uint16_t address)
- uint16_t ESC_readWordISR (uint16_t address)
- void ESC_releaseESCReset (void)
- void ESC_releaseHW (void)
- void ESC_resetESC (void)
- void ESC_setLed (uint16_t runLed, uint16_t errLed)
- void ESC_setupPDITestInterface (void)
- void ESC_signalFail (void)
- void ESC_signalPass (void)
- uint32_t ESC_timerIncPerMilliSec (void)
- void ESC_writeBlock (ESCMEM_ADDR ∗pData, uint16_t address, uint16_t len)
- void ESC_writeBlockISR (ESCMEM_ADDR ∗pData, uint16_t address, uint16_t len)
- void ESC_writeDWord (uint32_t dWordValue, uint16_t address)
- void ESC_writeDWordISR (uint32_t dWordValue, uint16_t address)
- void ESC_writeWord (uint16_t wordValue, uint16_t address)
- void ESC_writeWordISR (uint16_t wordValue, uint16_t address)

5.1.1 Function Documentation

5.1.1.1 ESC_applicationLayerHandler

Application Layer Handler

Prototype:

```
__interrupt void
ESC_applicationLayerHandler(void)
```

Description:
This function is the interrupt handler for EtherCAT application/PDI interrupts.

Returns:
None

5.1.1.2 ESC_applicationSync0Handler

Application Sync 0 Handler

Prototype:

```
__interrupt void
ESC_applicationSync0Handler(void)
```

Description:
This function is the interrupt handler for EtherCAT SYNC0 interrupts.

Returns:
None

5.1.1.3 ESC_applicationSync1Handler

Application Sync 1 Handler

Prototype:

```
__interrupt void
ESC_applicationSync1Handler(void)
```

Description:
This function is the interrupt handler for EtherCAT SYNC1 interrupts.

Returns:
None

5.1.1.4 ESC_clearTimer

Clears the Timer Value

Prototype:

```
void
ESC_clearTimer(void)
```

Description:
This function resets the timer counter.

Returns:
None

5.1.1.5 ESC_debugAddESCRegsAddress

Adds ESC Register Address to be read to RAM for Debug

Prototype:
```
void
ESC_debugAddESCRegsAddress(uint16_t address)
```

Parameters:
address is the ESC register or memory byte address that needs to be read

Description:
This function is optional for the non-HAL API Test application use cases and is available for applications or users to add a register to the pre-set array of registers that are ready by the ESC_debugUpdateESCRegLogs() function using the PDI interface.

Note:
Only 16-bit reads are provided by the reference code.

This function is only relevant for the HAL API Test application.

Returns:
None.

5.1.1.6 ESC_debugInitESCRegLogs

Initializes ESC Register Read Log Array

Prototype:
```
void
ESC_debugInitESCRegLogs(void)
```

Description:
This function is optional for the non-HAL API Test application use cases and is available for applications or users to and initializes the registers read log array to default 0xFFFF. This is called once during init time or user can call it after every update to reset the previous read values in the array.

Note:
This function is only relevant for the HAL API Test application.

Returns:
None.

5.1.1.7 ESC_debugUpdateESCRegLogs

Reloads local RAM with ESC register Values

Prototype:
```
void
ESC_debugUpdateESCRegLogs(void)
```

Description:
This function is optional for the non-HAL API Test application use cases and is available for applications or users to perform a load of pre-set ESC registers in a loop by using PDI interface.

Note:
This function is only relevant for the HAL API Test application.

Returns:
None.

5.1.1.8 ESC_getTimer

Gets the Current Timer Value

Prototype:
```
uint32_t
ESC_getTimer(void)
```

Description:
This function returns the current timer counter value from the CPU timer.

Returns:
Returns the 1's compliment of the timer counter register value

5.1.1.9 ESC_holdESCInReset

Hold ESC in Reset

Prototype:
```
void
ESC_holdESCInReset(void)
```

Description:
This function holds the ESC peripheral in reset.

Returns:
None.

5.1.1.10 ESC_initHW

Initializes the Device for EtherCAT

Prototype:

```
uint16_t
ESC_initHW(void)
```

Description:

This function initializes the host controller, interrupts, SYNC signals, PDI, and other necessary peripherals.

Returns:

Returns **ESC_HW_INIT_SUCCESS** if initialization was successful and **ESC_HW_INIT_FAIL** if an error occurred during initialization

5.1.1.11 ESC_loadedCheckEEPROM

Checks if EEPROM was Loaded

Prototype:

```
uint16_t
ESC_loadedCheckEEPROM(void)
```

Description:

This function checks if the EEPROM load happened properly or not. The function reads the EEPROM LOADED register bit in the DL register as no proper EEPROM loaded IO signal is available. Recommended to be called by applications during start up and after an EEPROM reload happens.

Note:

ESC RAM access via PDI is blocked until EEPROM happens correctly.

Returns:

Returns **ESC_EEPROM_SUCCESS** if EEPROM loaded successfully, **ESC_EEPROM_NOT_LOADED** if EEPROM not loaded as per the ESC DL register status, and **ESC_EEPROM_LOAD_ERROR** if EEPROM ESC control status register indicates that EEPROM is not loaded and device information not available.

5.1.1.12 ESC_passFailSignalSetup

Sets up the ControlCARD GPIOs for LEDs

Prototype:

```
void
ESC_passFailSignalSetup(void)
```

Description:

This function sets up the LED GPIOs that are used to signal the PASS/FAIL conditions.

Note:

This function is tied to the controlCARD as in the GPIOs used as per the controlCARD hardware design.

This function is only relevant for the HAL API Test application.

Returns:

None.

5.1.1.13 ESC_readBlock

Reads the ESC Data into Local Buffer with Interrupts Disabled

Prototype:

```
void
ESC_readBlock(ESCMEM_ADDR *pData,
              uint16_t address,
              uint16_t len)
```

Parameters:

pData is the pointer to the local destination buffer. (Type of pointer depends on the host controller architecture, detailed in ecat_def.h or the Slave Stack Code Tool)

address is the EtherCAT slave controller offset address which specifies the offset within the ESC memory area in bytes. (Only valid addresses are used depending on ESC 8 bit, 16 bit, or 32 bit access specified in ecat_def.h or the Slave Stack Code Tool)

len is the access size in bytes

Description:

This function is used to access the ESC registers and the DPRAM area with interrupts disabled. The function disables interrupts, reads the requested number of bytes from the ESC address, copies the data into the data buffer specified, and re-enables interrupts.

Returns:

None

5.1.1.14 ESC_readBlockISR

Reads the ESC Data into Local Buffer

Prototype:

```
void
ESC_readBlockISR(ESCMEM_ADDR *pData,
                 uint16_t address,
                 uint16_t len)
```

Parameters:

pData is the pointer to the local destination buffer. (Type of pointer depends on the host controller architecture, detailed in ecat_def.h or the Slave Stack Code Tool)

address is the EtherCAT slave controller offset address which specifies the offset within the ESC memory area in bytes. (Only valid addresses are used depending on ESC 8 bit, 16 bit, or 32 bit access specified in ecat_def.h or the Slave Stack Code Tool)

len is the access size in bytes

Description:

This function is used to access the ESC registers and the DPRAM area. The function reads the requested number of bytes from the ESC address and copies the data into the data buffer specified.

Returns:

None

5.1.1.15 ESC_readDWord

Reads two 16-bit words from ESC Memory with interrupts disabled

Prototype:

```
uint32_t
ESC_readDWord(uint16_t address)
```

Parameters:

address is the EtherCAT slave controller offset address in bytes. This must be a valid 32-bit aligned address boundary.

Description:

This function disables interrupts, then reads two 16-bit words from the specified ESC address, and re-enables interrupts.

Returns:

Returns two 16-bit words

5.1.1.16 ESC_readDWordISR

Reads two 16-bit words from ESC Memory

Prototype:

```
uint32_t
ESC_readDWordISR(uint16_t address)
```

Parameters:

address is the EtherCAT slave controller offset address in bytes. This must be a valid 32-bit aligned address boundary.

Description:

This function reads two 16-bit words from the specified ESC address.

Returns:

Returns two 16-bit words

5.1.1.17 ESC_readWord

Reads one 16-bit word from ESC Memory with interrupts disabled

Prototype:

```
uint16_t
ESC_readWord(uint16_t address)
```

Parameters:

address is the EtherCAT slave controller offset address in bytes. This must be a valid 16-bit aligned address boundary.

Description:

This function disables interrupts, reads one 16-bit word from the specified ESC address, and re-enables interrupts.

Returns:
Returns 16-bit word value

5.1.1.18 ESC_readWordISR

Reads one 16-bit word from ESC Memory

Prototype:
```
uint16_t
ESC_readWordISR(uint16_t address)
```

Parameters:
address is the EtherCAT slave controller offset address in bytes. This must be a valid 16-bit aligned address boundary.

Description:
This function reads one 16-bit word from the specified ESC address..

Returns:
Returns 16-bit word value

5.1.1.19 ESC_releaseESCReset

Release ESC from Reset

Prototype:
```
void
ESC_releaseESCReset(void)
```

Description:
This function de-activates the ESC peripheral reset signal and brings ESC out of reset.

Returns:
None.

5.1.1.20 ESC_releaseHW

Releases the Device Resources

Prototype:
```
void
ESC_releaseHW(void)
```

Description:
This function releases the allocated device resources.

Note:
Implementation of this function is left to the end user and currently performs no action.

Returns:
None

5.1.1.21 ESC_resetESC

Reset the ESC

Prototype:

```
void
ESC_resetESC(void)
```

Description:
This function resets the ESC peripheral.

Returns:
None.

5.1.1.22 ESC_setLed

Updates the EtherCAT Run and Error LEDs

Prototype:

```
void
ESC_setLed(uint16_t runLed,
           uint16_t errLed)
```

Parameters:
runLed is the EtherCAT run LED state
errLed is the EtherCAT error LED state

Description:
This function updates the EtherCAT run and error LEDS (or EtherCAT status LED).

Note:
This is configured to use the LED GPIOs for the controlCARD.

Returns:
None

5.1.1.23 ESC_setupPDITestInterface

Setup and run tests on the PDI Interface

Prototype:

```
void
ESC_setupPDITestInterface(void)
```

Description:
This function is optional for the non-HAL API Test application use cases and is available for applications or users to perform a test of the PDI interface. The function reads the PDI control registers, initializes an array of registers that needs to be read from ESC and also performs read write tests on all of the RAM in ESC using the HAL API.

Note:
This function is only relevant for the HAL API Test application.

Returns:
None.

5.1.1.24 ESC_signalFail

Signal Fail Status on ControlCARD LEDs

Prototype:
```
void
ESC_signalFail(void)
```

Description:
This function provides a FAIL signature on the LED GPIOs when the tests complete successfully.

Note:
This function is tied to the controlCARD as in the GPIOs used as per the controlCARD hardware design.

This function is only relevant for the HAL API Test application.

Returns:
None.

5.1.1.25 ESC_signalPass

Signal Pass Status on ControlCARD LEDs

Prototype:
```
void
ESC_signalPass(void)
```

Description:
This function provides a PASS signature on the LED GPIOs when the tests complete successfully.

Note:
This function is tied to the controlCARD as in the GPIOs used as per the controlCARD hardware design.

This function is only relevant for the HAL API Test application.

Returns:
None.

5.1.1.26 ESC_timerIncPerMilliSec

Get the Timer Increment Value

Prototype:
```
uint32_t
ESC_timerIncPerMilliSec(void)
```

Description:
This function returns a constant value of 200000UL for the timer increment value as the CPU timer is configured to run at 200MHz.

Returns:
Returns a constant value depending on the max frequency of the CPU timer

5.1.1.27 ESC_writeBlock

Writes the Local Buffer Data into the ESC Memory with interrupts disabled

Prototype:
```
void
ESC_writeBlock(ESCMEM_ADDR *pData,
               uint16_t address,
               uint16_t len)
```

Parameters:
- ***pData*** is the pointer to the local source buffer. (Type of pointer depends on the host controller architecture, detailed in ecat_def.h or the Slave Stack Code Tool)
- ***address*** is the EtherCAT slave controller offset address which specifies the offset within the ESC memory area in bytes. (Only valid addresses are used depending on ESC 8 bit, 16 bit, or 32 bit access specified in ecat_def.h or the Slave Stack Code Tool)
- ***len*** is the access size in bytes

Description:
This function disables interrupts, writes the requested number of bytes from the data buffer into the specified ESC addresses, and re-enables interrupts.

Returns:
None

5.1.1.28 ESC_writeBlockISR

Writes the Local Buffer Data into the ESC Memory

Prototype:
```
void
ESC_writeBlockISR(ESCMEM_ADDR *pData,
                  uint16_t address,
                  uint16_t len)
```

Parameters:
- ***pData*** is the pointer to the local source buffer. (Type of pointer depends on the host controller architecture, detailed in ecat_def.h or the Slave Stack Code Tool)
- ***address*** is the EtherCAT slave controller offset address which specifies the offset within the ESC memory area in bytes. (Only valid addresses are used depending on ESC 8 bit, 16 bit, or 32 bit access specified in ecat_def.h or the Slave Stack Code Tool)
- ***len*** is the access size in bytes

Description:
This function writes the requested number of bytes from the data buffer and into the specified ESC addresses.

Returns:
None

5.1.1.29 ESC_writeDWord

Writes two 16-bit words into ESC Memory with interrupts disabled

Prototype:
```
void
ESC_writeDWord(uint32_t dWordValue,
               uint16_t address)
```

Parameters:
dWordValue is the local 32-bit variable which contains the value that needs to be written.
address is the EtherCAT slave controller offset address in bytes. This must be a valid 32-bit aligned address boundary.

Description:
This function disables interrupts, writes two 16-bit words from *DWordValue* to the ESC address, and re-enables interrupts.

Returns:
None

5.1.1.30 ESC_writeDWordISR

Writes two 16-bit words into ESC Memory

Prototype:
```
void
ESC_writeDWordISR(uint32_t dWordValue,
                  uint16_t address)
```

Parameters:
dWordValue is the local 32-bit variable which contains the value that needs to be written.
address is the EtherCAT slave controller offset address in bytes. This must be a valid 32-bit aligned address boundary.

Description:
This function writes two 16-bit words from *DWordValue* to the ESC address.

Returns:
None

5.1.1.31 ESC_writeWord

Writes one 16-bit word into ESC Memory with interrupts disabled

Prototype:

```
void
ESC_writeWord(uint16_t wordValue,
              uint16_t address)
```

Parameters:

wordValue is the local 16-bit variable which contains the value to be written.

address is the EtherCAT slave controller offset address in bytes. This must be a valid 16-bit aligned address boundary.

Description:

This function disables interrupts, writes one 16-bit word from *WordValue* into the specified ESC address, and re-enables interrupts.

Returns:

None

5.1.1.32 ESC_writeWordISR

Writes one 16-bit word into ESC Memory

Prototype:

```
void
ESC_writeWordISR(uint16_t wordValue,
                 uint16_t address)
```

Parameters:

wordValue is the local 16-bit variable which contains the value to be written.

address is the EtherCAT slave controller offset address in bytes. This must be a valid 16-bit aligned address boundary.

Description:

This function writes one 16-bit word from *WordValue* into the specified ESC address.

Returns:

None

5.2 CM HAL APIs

Functions

- __interrupt void ESC_applicationLayerHandler (void)
- __interrupt void ESC_applicationSync0Handler (void)
- __interrupt void ESC_applicationSync1Handler (void)
- void ESC_clearTimer (void)
- void ESC_debugAddESCRegsAddress (uint16_t address)

- void ESC_debugInitESCRegLogs (void)
- void ESC_debugUpdateESCRegLogs (void)
- uint32_t ESC_getTimer (void)
- void ESC_holdESCInReset (void)
- uint16_t ESC_initHW (void)
- uint16_t ESC_loadedCheckEEPROM (void)
- void ESC_passFailSignalSetup (void)
- void ESC_readBlock (ESCMEM_ADDR ∗pData, uint16_t address, uint16_t len)
- void ESC_readBlockISR (ESCMEM_ADDR ∗pData, uint16_t address, uint16_t len)
- uint8_t ESC_readByte (uint16_t address)
- uint8_t ESC_readByteISR (uint16_t address)
- uint32_t ESC_readDWord (uint16_t address)
- uint32_t ESC_readDWordISR (uint16_t address)
- uint16_t ESC_readWord (uint16_t address)
- uint16_t ESC_readWordISR (uint16_t address)
- void ESC_releaseESCReset (void)
- void ESC_releaseHW (void)
- void ESC_resetESC (void)
- void ESC_setLed (uint8_t runLed, uint8_t errLed)
- void ESC_setupPDITestInterface (void)
- void ESC_signalFail (void)
- void ESC_signalPass (void)
- uint32_t ESC_timerIncPerMilliSec (void)
- void ESC_writeBlock (ESCMEM_ADDR ∗pData, uint16_t address, uint16_t len)
- void ESC_writeBlockISR (ESCMEM_ADDR ∗pData, uint16_t address, uint16_t len)
- void ESC_writeByte (uint8_t byteValue, uint16_t address)
- void ESC_writeByteISR (uint8_t byteValue, uint16_t address)
- void ESC_writeDWord (uint32_t dWordValue, uint16_t address)
- void ESC_writeDWordISR (uint32_t dWordValue, uint16_t address)
- void ESC_writeWord (uint16_t wordValue, uint16_t address)
- void ESC_writeWordISR (uint16_t wordValue, uint16_t address)

5.2.1 Function Documentation

5.2.1.1 ESC_applicationLayerHandler

Application Layer Handler

Prototype:

```
__interrupt void
ESC_applicationLayerHandler(void)
```

Description:

This function is the interrupt handler for EtherCAT application/PDI interrupts.

Returns:

None

5.2.1.2 ESC_applicationSync0Handler

Application Sync 0 Handler

Prototype:
```
__interrupt void
ESC_applicationSync0Handler(void)
```

Description:
This function is the interrupt handler for EtherCAT SYNC0 interrupts.

Returns:
None

5.2.1.3 ESC_applicationSync1Handler

Application Sync 1 Handler

Prototype:
```
__interrupt void
ESC_applicationSync1Handler(void)
```

Description:
This function is the interrupt handler for EtherCAT SYNC1 interrupts.

Returns:
None

5.2.1.4 ESC_clearTimer

Clears the Timer Value

Prototype:
```
void
ESC_clearTimer(void)
```

Description:
This function resets the timer counter.

Returns:
None

5.2.1.5 ESC_debugAddESCRegsAddress

Adds ESC Register Address to be read to RAM for Debug

Prototype:
```
void
ESC_debugAddESCRegsAddress(uint16_t address)
```

Parameters:
address is the ESC register or memory address that needs to be read

Description:
This function is optional for the non-HAL API Test application use cases and is available for applications or users to add a register to the pre-set array of registers that are ready by the ESC_debugUpdateESCRegLogs() function using the PDI interface.

Note:
Only 16-bit reads are provided by the reference code.

This function is only relevant for the HAL API Test application.

Returns:
None.

5.2.1.6 ESC_debugInitESCRegLogs

Initializes ESC Register Read Log Array

Prototype:
```
void
ESC_debugInitESCRegLogs(void)
```

Description:
This function is optional for the non-HAL API Test application use cases and is available for applications or users to and initializes the registers read log array to default 0xFFFF. This is called once during init time or user can call it after every update to reset the previous read values in the array.

Note:
This function is only relevant for the HAL API Test application.

Returns:
None.

5.2.1.7 ESC_debugUpdateESCRegLogs

Reloads local RAM with ESC register Values

Prototype:
```
void
ESC_debugUpdateESCRegLogs(void)
```

Description:
This function is optional for the non-HAL API Test application use cases and is available for applications or users to perform a load of pre-set ESC registers in a loop by using PDI interface.

Note:
This function is only relevant for the HAL API Test application.

Returns:
None.

5.2.1.8 ESC_getTimer

Gets the Current Timer Value

Prototype:
```
uint32_t
ESC_getTimer(void)
```

Description:
This function returns the current timer counter value from the CPU timer.

Returns:
Returns the 1's compliment of the timer counter register value

5.2.1.9 ESC_holdESCInReset

Hold ESC in Reset

Prototype:
```
void
ESC_holdESCInReset(void)
```

Description:
This function holds the ESC peripheral in reset.

Returns:
None.

5.2.1.10 ESC_initHW

Initializes the Device for EtherCAT

Prototype:
```
uint16_t
ESC_initHW(void)
```

Description:
This function initializes the host controller, interrupts, SYNC signals, PDI, and other necessary peripherals.

Returns:
Returns **ESC_HW_INIT_SUCCESS** if initialization was successful and **ESC_HW_INIT_FAIL** if an error occurred during initialization

5.2.1.11 ESC_loadedCheckEEPROM

Checks if EEPROM was Loaded

Prototype:
```
uint16_t
ESC_loadedCheckEEPROM(void)
```

Description:

This function checks if the EEPROM load happened properly or not. The function reads the EEPROM LOADED register bit in the DL register as no proper EEPROM loaded IO signal is available. Recommended to be called by applications during start up and after an EEPROM reload happens.

Note:

ESC RAM access via PDI is blocked until EEPROM happens correctly.

Returns:

Returns **ESC_EEPROM_SUCCESS** if EEPROM loaded successfully, **ESC_EEPROM_NOT_LOADED** if EEPROM not loaded as per the ESC DL register status, and **ESC_EEPROM_LOAD_ERROR** if EEPROM ESC control status register indicates that EEPROM is not loaded and device information not available.

5.2.1.12 ESC_passFailSignalSetup

Sets up the ControlCARD GPIOs for LEDs

Prototype:

```
void
ESC_passFailSignalSetup(void)
```

Description:

This function sets up the LED GPIOs that are used to signal the PASS/FAIL conditions.

Note:

This function is tied to the controlCARD as in the GPIOs used as per the controlCARD hardware design.

This function is only relevant for the HAL API Test application.

Returns:

None.

5.2.1.13 ESC_readBlock

Reads the ESC Data into Local Buffer with Interrupts Disabled

Prototype:

```
void
ESC_readBlock(ESCMEM_ADDR *pData,
              uint16_t address,
              uint16_t len)
```

Parameters:

pData is the pointer to the local destination buffer. (Type of pointer depends on the host controller architecture, detailed in ecat_def.h or the Slave Stack Code Tool)

address is the EtherCAT slave controller offset address which specifies the offset within the ESC memory area in bytes. (Only valid addresses are used depending on ESC 8 bit, 16 bit, or 32 bit access specified in ecat_def.h or the Slave Stack Code Tool)

len is the access size in bytes

Description:

This function is used to access the ESC registers and the DPRAM area with interrupts disabled. The function disables interrupts, reads the requested number of bytes from the ESC address, copies the data into the data buffer specified, and re-enables interrupts.

Returns:

None

5.2.1.14 ESC_readBlockISR

Reads the ESC Data into Local Buffer

Prototype:

```
void
ESC_readBlockISR(ESCMEM_ADDR *pData,
                 uint16_t address,
                 uint16_t len)
```

Parameters:

pData is the pointer to the local destination buffer. (Type of pointer depends on the host controller architecture, detailed in ecat_def.h or the Slave Stack Code Tool)

address is the EtherCAT slave controller offset address which specifies the offset within the ESC memory area in bytes. (Only valid addresses are used depending on ESC 8 bit, 16 bit, or 32 bit access specified in ecat_def.h or the Slave Stack Code Tool)

len is the access size in bytes

Description:

This function is used to access the ESC registers and the DPRAM area. The function reads the requested number of bytes from the ESC address and copies the data into the data buffer specified.

Returns:

None

5.2.1.15 ESC_readByte

Reads one byte from ESC Memory with interrupts disabled

Prototype:

```
uint8_t
ESC_readByte(uint16_t address)
```

Parameters:

address is the EtherCAT slave controller offset address in bytes.

Description:

This function disables interrupts, reads one byte from the specified ESC address, and re-enables interrupts.

Returns:

Returns byte value

5.2.1.16 ESC_readByteISR

Reads one byte from ESC Memory

Prototype:
```
uint8_t
ESC_readByteISR(uint16_t address)
```

Parameters:
address is the EtherCAT slave controller offset address in bytes.

Description:
This function reads one byte from the specified ESC address.

Returns:
Returns byte value

5.2.1.17 ESC_readDWord

Reads two 16-bit words from ESC Memory with interrupts disabled

Prototype:
```
uint32_t
ESC_readDWord(uint16_t address)
```

Parameters:
address is the EtherCAT slave controller offset address in bytes. This must be a valid 32-bit aligned address boundary.

Description:
This function disables interrupts, then reads two 16-bit words from the specified ESC address, and re-enables interrupts.

Returns:
Returns two 16-bit words

5.2.1.18 ESC_readDWordISR

Reads two 16-bit words from ESC Memory

Prototype:
```
uint32_t
ESC_readDWordISR(uint16_t address)
```

Parameters:
address is the EtherCAT slave controller offset address in bytes. This must be a valid 32-bit aligned address boundary.

Description:
This function reads two 16-bit words from the specified ESC address.

Returns:
Returns two 16-bit words

5.2.1.19 ESC_readWord

Reads one 16-bit word from ESC Memory with interrupts disabled

Prototype:

```
uint16_t
ESC_readWord(uint16_t address)
```

Parameters:

address is the EtherCAT slave controller offset address in bytes. This must be a valid 16-bit aligned address boundary.

Description:

This function disables interrupts, reads one 16-bit word from the specified ESC address, and re-enables interrupts.

Returns:

Returns 16-bit word value

5.2.1.20 ESC_readWordISR

Reads one 16-bit word from ESC Memory

Prototype:

```
uint16_t
ESC_readWordISR(uint16_t address)
```

Parameters:

address is the EtherCAT slave controller offset address in bytes. This must be a valid 16-bit aligned address boundary.

Description:

This function reads one 16-bit word from the specified ESC address.

Returns:

Returns 16-bit word value

5.2.1.21 ESC_releaseESCReset

Release ESC from Reset

Prototype:

```
void
ESC_releaseESCReset(void)
```

Description:

This function de-activates the ESC peripheral reset signal and brings ESC out of reset.

Returns:

None.

5.2.1.22 ESC_releaseHW

Releases the Device Resources

Prototype:

```
void
ESC_releaseHW(void)
```

Description:
This function releases the allocated device resources.

Note:
Implementation of this function is left to the end user and currently performs no action.

Returns:
None

5.2.1.23 ESC_resetESC

Reset the ESC

Prototype:

```
void
ESC_resetESC(void)
```

Description:
This function resets the ESC peripheral.

Returns:
None.

5.2.1.24 ESC_setLed

Updates the EtherCAT Run and Error LEDs

Prototype:

```
void
ESC_setLed(uint8_t runLed,
           uint8_t errLed)
```

Parameters:
runLed is the EtherCAT run LED state
errLed is the EtherCAT error LED state

Description:
This function updates the EtherCAT run and error LEDS (or EtherCAT status LED).

Note:
This is configured to use the LED GPIOs for the controlCARD.

Returns:
None

5.2.1.25 ESC_setupPDITestInterface

Setup and run tests on the PDI Interface

Prototype:

```
void
ESC_setupPDITestInterface(void)
```

Description:

This function is optional for the non-HAL API Test application use cases and is available for applications or users to perform a test of the PDI interface. The function reads the PDI control registers, initializes an array of registers that needs to be read from ESC and also performs read write tests on all of the RAM in ESC using the HAL API.

Note:

This function is only relevant for the HAL API Test application.

Returns:

None.

5.2.1.26 ESC_signalFail

Signal Fail Status on ControlCARD LEDs

Prototype:

```
void
ESC_signalFail(void)
```

Description:

This function provides a FAIL signature on the LED GPIOs when the tests complete successfully.

Note:

This function is tied to the controlCARD as in the GPIOs used as per the controlCARD hardware design.

This function is only relevant for the HAL API Test application.

Returns:

None.

5.2.1.27 ESC_signalPass

Signal Pass Status on ControlCARD LEDs

Prototype:

```
void
ESC_signalPass(void)
```

Description:

This function provides a PASS signature on the LED GPIOs when the tests complete successfully.

Note:

This function is tied to the controlCARD as in the GPIOs used as per the controlCARD hardware design.

This function is only relevant for the HAL API Test application.

Returns:

None.

5.2.1.28 ESC_timerIncPerMilliSec

Get the Timer Increment Value

Prototype:

```
uint32_t
ESC_timerIncPerMilliSec(void)
```

Description:

This function returns a constant value of 125000UL for the timer increment value as the CPU timer is configured to run at 125MHz.

Returns:

Returns a constant value depending on the max frequency of the CPU timer

5.2.1.29 ESC_writeBlock

Writes the Local Buffer Data into the ESC Memory with interrupts disabled

Prototype:

```
void
ESC_writeBlock(ESCMEM_ADDR *pData,
               uint16_t address,
               uint16_t len)
```

Parameters:

pData is the pointer to the local source buffer. (Type of pointer depends on the host controller architecture, detailed in ecat_def.h or the Slave Stack Code Tool)

address is the EtherCAT slave controller offset address which specifies the offset within the ESC memory area in bytes. (Only valid addresses are used depending on ESC 8 bit, 16 bit, or 32 bit access specified in ecat_def.h or the Slave Stack Code Tool)

len is the access size in bytes

Description:

This function disables interrupts, writes the requested number of bytes from the data buffer into the specified ESC addresses, and re-enables interrupts.

Returns:

None

5.2.1.30 ESC_writeBlockISR

Writes the Local Buffer Data into the ESC Memory

Prototype:

```
void
ESC_writeBlockISR(ESCMEM_ADDR *pData,
                  uint16_t address,
                  uint16_t len)
```

Parameters:

pData is the pointer to the local source buffer. (Type of pointer depends on the host controller architecture, detailed in ecat_def.h or the Slave Stack Code Tool)

address is the EtherCAT slave controller offset address which specifies the offset within the ESC memory area in bytes. (Only valid addresses are used depending on ESC 8 bit, 16 bit, or 32 bit access specified in ecat_def.h or the Slave Stack Code Tool)

len is the access size in bytes

Description:

This function writes the requested number of bytes from the data buffer and into the specified ESC addresses.

Returns:

None

5.2.1.31 ESC_writeByte

Write one byte into ESC Memory with interrupts disabled

Prototype:

```
void
ESC_writeByte(uint8_t byteValue,
              uint16_t address)
```

Parameters:

byteValue is the local 8-bit variable which contains the value to be written.

address is the EtherCAT slave controller offset address in bytes.

Description:

This function disables interrupts, writes one byte from *byteValue* into the specified ESC address, and re-enables interrupts.

Returns:

None

5.2.1.32 ESC_writeByteISR

Write one byte into ESC Memory

Prototype:

```
void
ESC_writeByteISR(uint8_t byteValue,
                 uint16_t address)
```

Parameters:

byteValue is the local 8-bit variable which contains the value to be written.

address is the EtherCAT slave controller offset address in bytes.

Description:

This function writes one byte from *byteValue* into the specified ESC address.

Returns:

None

5.2.1.33 ESC_writeDWord

Writes two 16-bit words into ESC Memory with interrupts disabled

Prototype:

```
void
ESC_writeDWord(uint32_t dWordValue,
               uint16_t address)
```

Parameters:

dWordValue is the local 32-bit variable which contains the value that needs to be written.

address is the EtherCAT slave controller offset address in bytes. This must be a valid 32-bit aligned address boundary.

Description:

This function disables interrupts, writes two 16-bit words from *dWordValue* to the ESC address, and re-enables interrupts.

Returns:

None

5.2.1.34 ESC_writeDWordISR

Writes two 16-bit words into ESC Memory

Prototype:

```
void
ESC_writeDWordISR(uint32_t dWordValue,
                  uint16_t address)
```

Parameters:

dWordValue is the local 32-bit variable which contains the value that needs to be written.

address is the EtherCAT slave controller offset address in bytes. This must be a valid 32-bit aligned address boundary.

Description:

This function writes two 16-bit words from *dWordValue* to the ESC address.

Returns:

None

5.2.1.35 ESC_writeWord

Writes one 16-bit word into ESC Memory with interrupts disabled

Prototype:

```
void
ESC_writeWord(uint16_t wordValue,
              uint16_t address)
```

Parameters:

wordValue is the local 16-bit variable which contains the value to be written.

address is the EtherCAT slave controller offset address in bytes. This must be a valid 16-bit aligned address boundary.

Description:

This function disables interrupts, writes one 16-bit word from *wordValue* into the specified ESC address, and re-enables interrupts.

Returns:

None

5.2.1.36 ESC_writeWordISR

Writes one 16-bit word into ESC Memory

Prototype:

```
void
ESC_writeWordISR(uint16_t wordValue,
                 uint16_t address)
```

Parameters:

wordValue is the local 16-bit variable which contains the value to be written.

address is the EtherCAT slave controller offset address in bytes. This must be a valid 16-bit aligned address boundary.

Description:

This function writes one 16-bit word from *wordValue* into the specified ESC address.

Returns:

None

6 Revision History

v2.00.00.00: Slave Stack Examples

- CPU1 and CM HAL Driver API Updates to support Stack
- CPU1 Allocate ECAT to CM Example Updated
- CPU1 Echoback Demo Example
- CPU1 Echoback Solution Example
- CM Echoback Demo Example
- CM Echoback Solution Example
- SSC Configuration and application stack files

v1.00.00.00: First Release

- CPU1 and CM HAL Driver APIs
- CPU1 PDI HAL Test Example
- CPU1 Allocate ECAT to CM Example
- CM PDI HAL Test Example
- Master mode initialization is supported

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

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