

# **C2000™ Position Manager EnDat22 Library Module**

## **User's Guide**



Literature Number: SPRUI35

November 2015

---



---



---

<b>1</b>	<b>Introduction</b> .....	<b>4</b>
1.1	EnDat interface .....	4
1.2	Abbreviations/Acronyms.....	4
1.3	System Description .....	4
1.4	C2000 EnDat Master Solution.....	5
1.5	EnDat22 Master Implementation Details.....	6
<b>2</b>	<b>PM EnDat22 Library</b> .....	<b>7</b>
2.1	PM EnDat22 Library Package Contents .....	7
<b>3</b>	<b>Module Summary</b> .....	<b>7</b>
3.1	PM EnDat22 Library Commands.....	7
3.2	PM EnDat22 Library Functions.....	8
3.3	Data Structures .....	9
3.4	Details of Function Usage .....	10
<b>4</b>	<b>Using PM_ENDAT22 Library</b> .....	<b>28</b>
4.1	Adding EnDat22 Lib to the Project .....	28
4.2	Steps for Initialization .....	29
4.3	Using the Library Functions .....	31
<b>5</b>	<b>Resource Requirements</b> .....	<b>33</b>
<b>6</b>	<b>Test Report</b> .....	<b>33</b>
<b>7</b>	<b>FAQs</b> .....	<b>34</b>
<b>8</b>	<b>References</b> .....	<b>34</b>

## List of Figures

1	Industrial Servo Drive With EnDat 2.2 Position Encoder Interface.....	5
2	EnDat Implementation Diagram Inside TMS320F28379D .....	6
3	Compiler Options for a Project Using PM EnDat22 Library .....	28
4	Adding PM EnDat22 Library to the Linker Options in CCS Project.....	29

## List of Tables

1	Abbreviations/Acronyms .....	4
2	Commands Supported .....	7
3	PM EnDat22 Library Functions .....	8
4	Module interface Definition .....	10
5	Summary of Function Details .....	10
6	Resource Requirements.....	33
7	Test Report.....	33

# C2000™ Position Manager EnDat22 Library Module

## 1 Introduction

### 1.1 EnDat interface

EnDat, from Heidenhain, is designed for serial transfer of digital data between linear, rotary or angle encoders, touch probes, accelerometers and the subsequent electronics, such as numerical controls, servo amplifiers and programmable logic controllers. For more details on the EnDat protocol and implementation aspects, see the [Heidenhain](#) website.

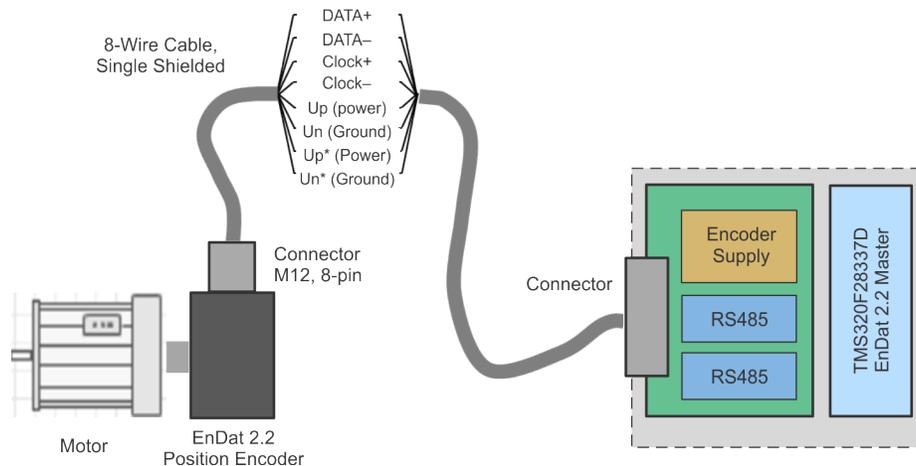
### 1.2 Abbreviations/Acronyms

**Table 1. Abbreviations/Acronyms**

Type	Description
C28x	Refers to devices with the C28x CPU core
CLB	Configurable Logic Block
CRC	Cyclic Redundancy Check
EnDat22	2.2 version of EnDat Position encoder interface protocol by <a href="#">Heidenhain</a>
EnDat21	2.1 version of EnDat Position encoder interface protocol by <a href="#">Heidenhain</a>
PM	Position Manager – Foundation hardware and software on C28x devices for position encoder interfaces
PM_endat22	Prefix used for all the library functions
SPI	Serial Peripheral Interface
Subsequent Electronics	EnDat Master implementation

### 1.3 System Description

Industrial drives (like servo drives), require accurate, highly reliable, and low-latency position feedback. A simplified system block diagram of a servo drive using an absolute position encoder with EnDat 2.2 bidirectional digital interface is shown in [Figure 1](#). The EnDat 2.2 interface from HEIDENHAIN is a digital, bidirectional interface standard for position or rotary encoders. The interface transmits position values or additional physical quantities. It also allows reading and writing of the encoder's internal memory. The type of data transmitted like absolute position, turns, temperature, parameters, diagnostics, and so on is selected through mode commands that the subsequent electronics, often referred to as EnDat 2.2 Master, sends to the encoder. The EnDat 2.2 interface is a pure serial digital interface based on RS-485 standard.



**Figure 1. Industrial Servo Drive With EnDat 2.2 Position Encoder Interface**

The position encoder with EnDat 2.2 is connected to the subsequent electronics, such as an EnDat Master implemented in an MCU, through a single, 8-wire shielded cable, as shown in [Figure 1](#). Below are details of each of the eight wires used for communication.

- Two wires for CLOCK+/CLOCK- transmitted in differential format
- Two wires for DATA+/DATA- transmitted in differential format
- Two wires (Up, Un) are used for the encoder power supply and ground
- Two wires (Up\*, Un\*) are used for battery buffering or for parallel power supply lines to reduce the cables losses.

#### 1.4 C2000 EnDat Master Solution

The Texas Instruments C2000 Position Manager EnDat22 (PM\_endat22) library is intended to provide support for implementing the EnDat interface in subsequent electronics.

Features offered by Endat22 library:

- Integrated MCU solution for EnDat interface
- Meets HEIDENHAIN EnDat 2.2/2.1 digital interface protocol requirements
- Clock frequency of up to 8 MHz supported irrespective of cable length
- Verified operation up to 100m cable length
- Easy interface to EnDat commands through driver functions and data structure provided by the library.
- Efficient and optimized CRC algorithm for both position and data CRC calculations
- Unpacking the received data and reversing the position data incorporated in library functions.
- Solution tuned for position control applications, where position information is obtained from encoders every control cycle, with better control of modular functions and timing.

Key things to note while using Endat22 library:

- This library supports only the basic interface drivers for commands defined in EnDat22 specification. All the higher level application software needs to be developed by utilizing the basic interface provided by this library.
- Clock frequency for the EnDat Clock is limited to a maximum of SYSCLOCK/24. This limitation applies irrespective of the cable length and encoder type.
- The functionality verified using this library, and this alone, is supported, see [Section 5](#). For any additional functionality or encoder usage not specified in this section, contact your TI support team.

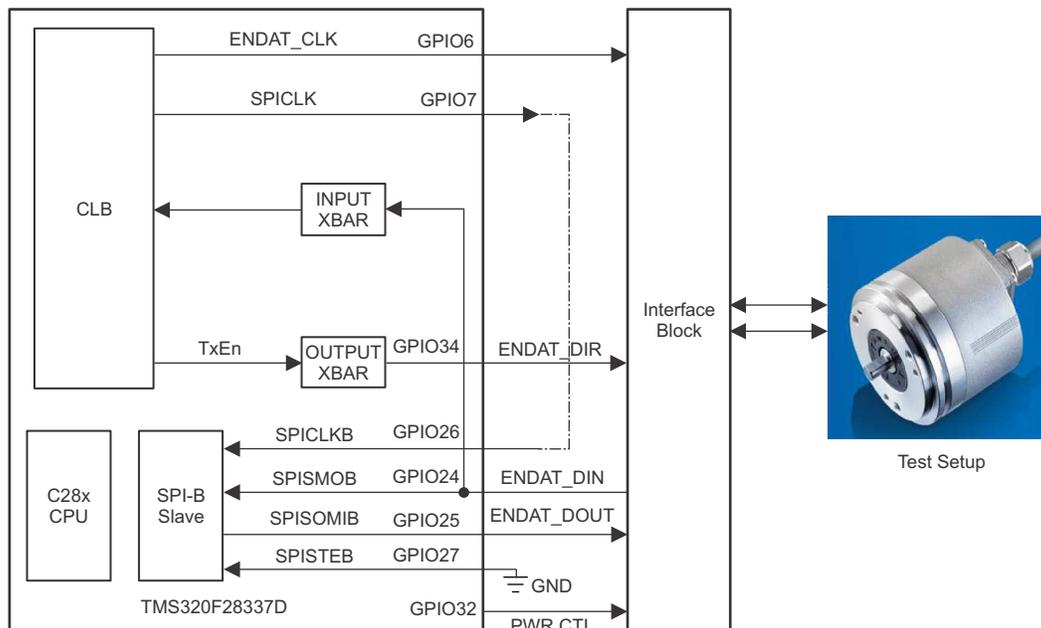
### 1.5 EnDat22 Master Implementation Details

This section gives a brief overview of how the EnDat interface is implemented on TMS320F28379D devices. Communication over EnDat interface is achieved primarily by the following components:

- CPU
- Configurable Logic Block (CLB)
- Serial Peripheral Interface (SPI)
- Device Interconnect (XBARSs)

While SPI performs the encoder data transmit and receive functions; clock generation is controlled by CLB. The following functions are implemented inside the CLB module. Note that the CLB module can only be accessed via library functions provided in the PM EnDat22 Library and not otherwise configurable by users.

- Ability to generate two different clocks
  - To the serial peripheral interface on chip – on GPIO7 and looped back to SPICLK input
  - Clock to the Encoder on GPIO6
- Identification of the critical delay between the clock edges sent to the encoder and the received data
- Ability to adjust the delay between the two clocks mentioned above
- Monitoring the data coming from encoder, via SPISIMO, and poll for start pulse
- Ability to measure the propagation delay at a specific interval as needed by the interface
- Ability to configure the block and adjust delay the via software



**Figure 2. EnDat Implementation Diagram Inside TMS320F28379D**

GPIOs indicated in [Figure 2](#) are as implemented on TMDXIDDK379D.

[Figure 2](#) depicts how EnDat transaction works in the system. For every EnDat transaction initiated using the PM EnDat22 Library command:

- CPU configures the SPITXFIFO with the command and other data required for transmission to the Encoder as per the specific requirements of the current EnDat command.
- CPU sets up the configurable logic block to generate clocks for the Encoder and SPI.
- Number of clock pulses and edge placement for these two clocks is different and is precisely controlled by CLB, as configured by CPU software for the current EnDat command.

- CLB also generates the direction control signal for data line transceiver. This signal is needed to change the direction of the data line, after sending the mode command from subsequent electronics, to receive the data from the encoder.
- CLB also monitors the SPISIMO signal, as needed, for detecting the start pulse and adjusts the phase of the receive clock accordingly.
- CPU configures CLB to generate continuous clocking for the Encoder while waiting for the Start pulse from Encoder.
- CPU configures CLB to generate a predefined number of clock pulses needed for the SPI (as per the current command requirements, and continuous clocking for SPI is disabled while waiting for Start pulse from encoder).
- CLB also provides hooks to perform cable propagation delay compensation via library functions.

More details on various library functions provided and their usage can be found in the remainder of this document. For more details on usage and establishing basic communication with the Encoder, see the examples for using the PM EnDat22 library.

## 2 PM EnDat22 Library

### 2.1 PM EnDat22 Library Package Contents

The EnDat22 Library consists of the following components:

- Header files and software library for EnDat22 interface
- Documentation - *Using Position Manager EnDat22 Library on IDDK Hardware User's Guide* ([SPRUI34](#))
- Example project showcasing EnDat22 interface implementation on the TMDXIDDK379D hardware

Library contents are available at the following locations:

<base> install directory is:

```
C:\ti\controlSUITE\libs\app_libs\position_manager\endat22\vX.X
```

The following sub-directory structure is used:

```
<base>\Doc           Documentation
<base>\Float        Contains implementation of the library and corresponding include file
<base>\examples     Example using EnDat22 library
```

## 3 Module Summary

This section describes the contents of PM\_endat22\_Include.h, the include file for using the EnDat22 library.

### 3.1 PM EnDat22 Library Commands

Details of the endat protocol and commands supported in different modes can be obtained from [Heidenhain](#) EnDat documentation.

**Table 2. Commands Supported**

EnDat 2.1	
Encoder send position values	ENCODER_SEND_POSITION_VALUES
Selection of the memory area	SELECTION_OF_MEMORY_AREA
Encoder receive parameters	ENCODER_RECEIVE_PARAMETER
Encoder send parameter	ENCODER_SEND_PARAMETER
Encoder receive reset	ENCODER_RECEIVE_RESET
Encoder send test values	ENCODER_SEND_TEST_VALUES
Encoder receive test command	ENCODER_RECEIVE_TEST_COMMAND
EnDat 2.2	
Encoder send position value with additional information	ENCODER_SEND_POSITION_VALUES_WITH_ADDITIONAL_DATA

**Table 2. Commands Supported (continued)**

EnDat 2.1	
Encoder send position value and receive selection of memory area	ENCODER_SEND_POSITION_VALUES_AND_SELECTION_OF_THE_MEMORY_AREA
Encoder send position value and receive parameters	ENCODER_SEND_POSITION_VALUES_AND_RECEIVE_PARAMETER
Encoder send position value and send parameters	ENCODER_SEND_POSITION_VALUES_AND_SEND_PARAMETER
Encoder send position value and receive test command	ENCODER_SEND_POSITION_VALUES_AND_RECEIVE_TEST_COMMAND
Encoder send position value and receive error reset	ENCODER_SEND_POSITION_VALUES_AND_RECEIVE_ERROR_RESET
Encoder receive communication command	ENCODER_RECEIVE_COMMUNICATION_COMMAND

### 3.2 PM EnDat22 Library Functions

The EnDat22 Library consists of the following functions that enable the user to interface with EnDat encoders. [Table 3](#) lists the functions existing in the Endat22 library and a summary of cycles taken for execution.

More details of the functions and their usage in the following sections.

**Table 3. PM EnDat22 Library Functions**

Name	Description	CPU Cycles	Type
PM_endat22_generateCRCTable	This function generates a table of 256 entries for a given CRC polynomial (polynomial) with a specified number of bits (nBits). Generated tables are stored at the address specified by pTable.	30226 <sup>(1)</sup>	Initialization time
PM_endat22_getCrcPos	To get the CRC of each byte, calculate the n-bit CRC of a message buffer by using the lookup table. Use this function for calculating CRC of the position data. <ul style="list-style-type: none"> <li>Encoder send position values (EnDat 2.1)</li> <li>Encoder send position values (EnDat 2.2)</li> </ul>	220 <sup>(1)</sup>	Run time
PM_endat22_getCrcTest	To get the CRC of each byte, calculate the n-bit CRC of a message buffer by using the lookup table. Use this function for calculating CRC of the Test data. <ul style="list-style-type: none"> <li>Encoder send test values</li> </ul>	183 <sup>(1)</sup>	Run time
PM_endat22_getCrcNorm	To get the CRC of each byte, calculate the n-bit CRC of a message buffer by using the lookup table. Use this function for calculating CRC for below commands. <ul style="list-style-type: none"> <li>Selection of memory area</li> <li>Encoder receive parameter</li> <li>Encoder send parameter</li> <li>Encoder receive reset</li> <li>Encoder receive test command</li> <li>Additional data (EnDat 2.2)</li> </ul>	95 <sup>(1)</sup>	Run time
PM_endat22_setupCommand	Setup an SPI and other modules for a given command to be transmitted. All of the transactions should start with this command. This function call sets up the peripherals for upcoming EnDat transfer, but does not actually perform any transfer or activity on the EnDat interface. This function call populates the sdata array of ENDAT_DATA_STRUCT with the data to be transmitted to the Encoder.	1160 <sup>(1)</sup>	Run time
PM_endat22_startOperation	This function initiates the EnDat transfer to be called after the PM_endat22_setupCommand. it performs the EnDat transaction set up by previous commands. Note that the setup up and start operation are separate function calls. You can setup the EnDat transfer and start the actual transfer using this function call, as needed, at a different time.	46	Run time

<sup>(1)</sup> Implies that the CPU cycle data depends on the Encoder under test as well as the commands and data being used along with certain functions. These numbers could vary significantly depending on the command and corresponding data, additional data, and so forth.

**Table 3. PM EnDat22 Library Functions (continued)**

Name	Description	CPU Cycles	Type
PM_endat22_receiveData	Function for unpacking and populating the EnDat data structure with the data received from the Encoder. This function is called when the data from the Encoder is available in the SPI data buffer and transferred to the rdata array of ENDAT_DATA_STRUCT. Upon the function call, received data is unpacked as per the current command and unpacked results are stored accordingly.	500 <sup>(1)</sup>	Run time
PM_endat22_setupPeriph	Setup for SPI, CLB and other interconnect XBARs for EnDat are performed with this function during system initialization. This function needs to be called after every system reset. No EnDat transactions will be performed until the setup peripheral function is called.	8822	Initializati on time
PM_endat22_setFreq	This function sets the EnDat clock frequency. EnDat transfers, typically start with low frequency during initialization and switch to higher frequency during runtime. Endat Clock Frequency = SYSCLK/(4*ENDAT_FREQ_DIVIDER) <ul style="list-style-type: none"> <li>Used during initialization (~200 KHz)</li> <li>Used during application (~8 MHz)</li> </ul>	220	nitializati on time
PM_endat22_getDelayCompVal	This function is used while performing delay compensation when long cables are used. This function returns the measured delay from the rising edge of EnDat Clock to the start bit received (see the provided examples directory on usage and performing delay compensation). For cable delays and propagation requirements, see the EnDat documentation from <a href="#">Heidenhain</a> .	21	Initializati on time

### 3.3 Data Structures

The PM EnDat22 library defines the EnDat data structure handle as below:

Object Definition:

```
typedef struct { // bit descriptions
    uint32_t position_lo;
    uint32_t position_hi;
    uint16_t error1;
    uint16_t error2;
    uint16_t data_crc;
    uint16_t address;
    uint32_t additional_data1;
    uint32_t additional_data2;
    uint32_t additional_data1_crc;
    uint32_t additional_data2_crc;
    uint32_t test_lo;
    uint32_t test_hi;
    uint32_t position_clocks;
    volatile struct SPI_REGS *spi;
    uint32_t delay_comp;
    uint32_t sdata[16];
    uint32_t rdata[16];
    uint16_t dataReady;
    uint16_t fifo_level;
} ENDAT_DATA_STRUCT;
```

**Table 4. Module interface Definition**

Module Element Name	Description	Type
position_lo	Lower 32 bits of the position data	32 bits
position_hi	Upper 32 bits of the position data	Max 16 bits
error1	Error1 status received in EnDat21	0 or 1
error2	Error2 status received in EnDat22	0 or 1
data_crc	CRC for Position and other commands (see each command for details)	5-Bits CRC
address	Received address in multiple commands	8-bit address
additional_data1	Additional data 1 received in EnDat22	32-bit unsigned int
additional_data2	Additional data 2 received in EnDat22	32-bit unsigned int
additional_data1_crc	CRC for Additional data 1 received in EnDat22	5-Bits CRC
additional_data2_crc	CRC for Additional data 2 received in EnDat22	5-Bits CRC
test_lo	Lower 32 bits of the test data for encoder send test values command	32 bits
test_hi	Upper 32 bits of the test data for encoder send test values command – test data is 40bits only	Max 8 bits
position_clocks	Word 13 of the parameter area for the encoder manufacturer to be read and stored in this. Number of clock pulses for transfer of position value.	Max value 48
delay_comp	Measured cable propagation delay to be updated in this variable	Unsigned int
spi	SPI instance used for EnDat22 implementation	Pointer to Spi*Regs
dataReady	Flag indicating dataReady – set by PM_endat22_receiveData function, cleared by PM_endat22_setupCommand function	0 or 1
sdata	Internal variables used by library – for debug purposes	Array of 32-bit unsigned integers
rdata	Internal variables used by library – for debug purposes	Array of 32-bit unsigned integers
fifo_level	Internal variables used by library – for debug purposes	Max value 8

### 3.4 Details of Function Usage

Detailed description of various library functions in PM EnDat22 library and their usage can be found in the following sections.

**Table 5. Summary of Function Details**

Title	Page
<b>PM_endat22_generateCRCTable</b> — .....	<b>11</b>
<b>PM_endat22_getCrcPos</b> — .....	<b>12</b>
<b>PM_endat22_getCrcTest</b> — .....	<b>13</b>
<b>PM_endat22_getCrcNorm</b> — .....	<b>14</b>
<b>PM_endat22_setupCommand</b> — .....	<b>16</b>
<b>PM_endat22_receiveData</b> — .....	<b>19</b>
<b>PM_endat22_startOperation</b> — .....	<b>24</b>
<b>PM_endat22_setupPeriph</b> — .....	<b>25</b>
<b>PM_endat22_setFreq</b> — .....	<b>26</b>
<b>PM_endat22_getDelayCompVal</b> — .....	<b>27</b>

## PM\_endat22\_generateCRCTable

**Directions** This function generates table of 256 entries for a given CRC polynomial (polynomial) with specified number of bits (nBits). Generated tables are stored at the address specified by pTable.

**Definition**

```
void PM_endat22_generateCRCTable(uint16_t nBits, uint16_t polynomial, uint16_t *pTable)
```

### Parameters

Input	
nBits	Number of bits of the given polynomial
polynomial	Polynomial used for CRC calculations
pTable	Pointer to the table where the CRC table values are stored
Return	
None	

### Usage

```
#define NBITS_POLY1 5
#define POLY1 0x0B
#define SIZEOFTABLE 256
uint16_t table1[SIZEOFTABLE];

// Generate table for poly 1
PM_endat22_generateCRCTable(
    NBITS_POLY1,
    POLY1,
    table1);
```

## PM\_endat22\_getCrcPos

**Description** To get the CRC of each byte, calculate the 5-bit CRC of a message buffer by using the lookup table. This function should be used for calculating CRC of the position data.

- Encoder send position values (EnDat 2.1)
- Encoder send position values (EnDat 2.2)

**Definition**

```
uint32_t PM_endat22_getCrcPos(uint32_t total_clocks,uint32_t endat22,uint32_t
lowpos,uint32_t highpos,
uint32_t error1,uint32_t error2, uint16_t *crc_table);
```

**Parameters**

Input	
total_clocks	Word 13 of the parameter area of the encoder manufacturer. Number of clock pulses for transfer of position value.
endat22	1 for EnDat22, 0 for EnDat21 position CRC
lowpos	Lower 32 bits of the position data
highpos	Upper 32 bits of the position data
error1	Error1 status received in EnDat21
error2	Error2 status received in EnDat22
crc_table	Pointer to the table where the CRC table values are stored
Return	
crc	5-bit CRC value calculated

**Usage**

```
Define Endat Data structure during initialization.
(ENDAT_DATA_STRUCT endat22Data;)
```

**Function call in 2.1 mode:**

```
crc5_result = PM_endat22_getCrcPos (
    endat22Data.position_clocks,
    ENDAT21, //EnDat21 mode => ENDAT21=0
    endat22Data.position_lo,
    endat22Data.position_hi,
    endat22Data.error1,
    endat22Data.error2, // ignored in EnDat21 mode
    table1); //crc table
```

**Function call in 2.2 mode:**

```
crc5_result = PM_endat22_getCrcPos (
    endat22Data.position_clocks,
    ENDAT22, //EnDat22 mode => ENDAT22=1
    endat22Data.position_lo,
    endat22Data.position_hi,
    endat22Data.error1,
    endat22Data.error2,
    table1); //crc table
```

**Example Code:**

```
Val = PM_endat22_setupCommand(ENCODER_SEND_POSITION_VALUES, 0, 0, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_SEND_POSITION_VALUES, 0);
crc5_result1 = PM_endat22_getCrcPos(endat22Data.position_clocks, 0,
endat22Data.position_lo, endat22Data.position_hi, endat22Data.error1,
endat22Data.error2, table1);
```

## PM\_endat22\_getCrcTest

**Description** To get the CRC of each byte, calculate the 5-bit CRC of a message buffer by using the lookup table. This function should be used for calculating CRC of the Test data.

- Encoder send test values

**Definition**

```
uint32_t PM_endat22_getCrcTest(uint32_t lowtest, uint32_t hightest, uint32_t
error1, uint16_t *crc_table);
```

### Parameters

Input	
lowtest	Lower 32 bits of the test data
hightest	Upper 32 bits of the test data
error1	Error1 status received in EnDat21
crc_table	Pointer to the table where the CRC table values are stored
Return	
crc	5-bit CRC value calculated

### Usage

Define Endat Data structure during initialization.  
(ENDAT\_DATA\_STRUCT endat22Data;)

Function call in 2.1 mode:

```
crc5_result1 = PM_endat22_getCrcTest(
    endat22Data.test_lo,
    endat22Data.test_hi,
    endat22Data.error1,
    table1); //crc table
```

This function is exclusively used for calculating the CRC values for ENCODER\_SEND\_TEST\_VALUES command. This is EnDat2.1 mode command.

Example Code:

```
Val = PM_endat22_setupCommand(ENCODER_SEND_TEST_VALUES, 0x0, 0x0, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_SEND_TEST_VALUES, 0);
crc5_result1 = PM_endat22_getCrcTest(endat22Data.test_lo,
endat22Data.test_hi,
endat22Data.error1, table1);
```

## PM\_endat22\_getCrcNorm

**Description** To get the CRC of each byte, calculate the 5-bit CRC of a message buffer by using the lookup table. This function should be used for calculating CRC for the following commands:

- Selection of memory area
- Encoder receive parameter
- Encoder send parameter
- Encoder receive reset
- Encoder receive test command
- Additional data (EnDat 2.2)

**Definition**

```
uint32_t PM_endat22_getCrcNorm (uint32_t param8, uint32_t param16, uint16_t
*crc_table);
```

### Parameters

Input	
param8	Typically 8-bit Address or MRS code, and so forth, depending on the command
param16	Typically 16-bit Data or Acknowledgment, and so forth, depending on the command
crc_table	Pointer to the table where the CRC table values are stored
Return	
crc	5-bit CRC value calculated

### Usage

#### Example Code:

```
Val = PM_endat22_setupCommand(SELECTION_OF_MEMORY_AREA, 0xA1, 0x5555, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(SELECTION_OF_MEMORY_AREA, 0);
crc5_result1 = PM_endat22_getCrcNorm(endat22Data.address, endat22Data.data,
table1);
```

For the details on where the data received, for different EnDat commands, is unpacked and stored, see the PM\_endat22\_receiveData function. Below are few examples:

```
Define Endat Data structure during initialization.
ENDAT_DATA_STRUCT endat22Data;
```

While checking CRC for the data received by using:

- SELECTION\_OF\_MEMORY\_AREA
- ENCODER\_SEND\_PARAMETER
- ENCODER\_RECEIVE\_PARAMETER
- ENCODER\_RECEIVE\_TEST\_COMMAND

```
crc5_result = PM_endat22_getCrcNorm(
    endat22Data.address,
    endat22Data.data,
    table1); //crc table
```

While checking CRC for additional data1 received using:

- ENCODER\_SEND\_POSITION\_VALUES\_WITH\_ADDITIONAL\_DATA
- ENCODER\_SEND\_POSITION\_VALUES\_AND\_SELECTION\_OF\_THE\_MEMORY\_A  
REA
- ENCODER\_SEND\_POSITION\_VALUES\_AND\_RECEIVE\_PARAMETER
- ENCODER\_SEND\_POSITION\_VALUES\_AND\_SEND\_PARAMETER
- ENCODER\_SEND\_POSITION\_VALUES\_AND\_RECEIVE\_TEST\_COMMAND

- ENCODER\_SEND\_POSITION\_VALUES\_AND\_RECEIVE\_ERROR\_RESET

```
crc5_result = PM_endat22_getCrcNorm(
    endat22Data.additional_data1 >> 16, // top 8-
bits of additional data 1 as param8
    endat22Data.additional_data1,      // Uses lower 16-
bits of this field as param16
    table1); //crc table
```

While checking CRC for additional data2 received using:

- ENCODER\_SEND\_POSITION\_VALUES\_WITH\_ADDITIONAL\_DATA
- ENCODER\_SEND\_POSITION\_VALUES\_AND\_SELECTION\_OF\_THE\_MEMORY\_A  
REA
- ENCODER\_SEND\_POSITION\_VALUES\_AND\_SELECTION\_OF\_THE\_MEMORY\_A  
REA
- ENCODER\_SEND\_POSITION\_VALUES\_AND\_SEND\_PARAMETER
- ENCODER\_SEND\_POSITION\_VALUES\_AND\_RECEIVE\_TEST\_COMMAND
- ENCODER\_SEND\_POSITION\_VALUES\_AND\_RECEIVE\_ERROR\_RESET

```
crc5_result = PM_endat22_getCrcNorm(
    endat22Data.additional_data2 >> 16, // top 8-
bits of additional data 2 as param8
    endat22Data.additional_data2,      // Uses lower 16-
bits of this field as param16
    table1); //crc table
```

## PM\_endat22\_setupCommand

**Description** Setup a SPI and other modules for a given command to be transmitted. All the transactions should start with this command. This function call sets up the peripherals for upcoming EnDat transfer but does not actually perform any transfer or activity on the EnDat interface. This function call populates the sdata array of ENDAT\_DATA\_STRUCT with the data to be transmitted to the Encoder.

**Definition**

```
void Val = PM_endat22_setupCommand(uint16_t command, uint16_t data1, uint16_t data2, uint16_t nAddData);
```

### Parameters

Input	
command	Mode command for the EnDat transfer to be done
data1	Typically 18/6-bit Data or Address depending on the mode command
data2	Typically 18/6-bit Data or Address depending on the mode command
nAddData	Number of additional data enabled (0, 1 or 2 depending on the number of additional data enabled or not)
Return	
Val	If incorrect, command value is passed to this function, which would return zero. For all other cases function returns a value of one.

### Usage

#### Example Code:

```
Val = PM_endat22_setupCommand(SELECTION_OF_MEMORY_AREA, 0xA1, 0x5555, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(SELECTION_OF_MEMORY_AREA, 0);
crc5_result1 = PM_endat22_getCrcNorm(endat22Data.address, endat22Data.data,
table1);
```

Below are few examples of how the PM\_endat22\_setupCommand function is used with various mode commands. For further details, see the [Heidenhain](#) documentation.

Define Endat Data structure during initialization.

```
ENDAT_DATA_STRUCT endat22Data;
```

- **SELECTION\_OF\_MEMORY\_AREA**

In order to send or read parameters, the memory area must first be selected. This is done with the mode command, followed by a code for the memory area to be selected: the Memory Range Select (MRS) code. The encoder acknowledges the command.

```
Val = PM_endat22_setupCommand(
    SELECTION_OF_MEMORY_AREA,
    0xA1,          // MRS code
    0x5555,       // Any
    0);           // No. of additional data - 0 for EnDat21 commands
```

- **ENCODER\_SEND\_PARAMETER**

This mode command is required for reading parameters of encoder. It is read from the memory area that was last selected as being valid. The encoder acknowledges the command.

```
Val = PM_endat22_setupCommand(
    ENCODER_SEND_PARAMETER,
    0xD,          // Address
    0x5555,       // Any
    0);           // No. of additional data - 0 for EnDat21 commands
```

- **ENCODER\_RECEIVE\_PARAMETER**

This mode command is required for writing parameters of encoder. It is written to the memory area that was last selected as being valid. The encoder acknowledges the command.

```
Val = PM_endat22_setupCommand(
    ENCODER_RECEIVE_PARAMETER,
    0xA1,          // Address
    0x5555,       // Parameters
    0);           // No. of additional data - 0 for EnDat21 commands
```

- **ENCODER\_RECEIVE\_RESET**

This mode command is required for executing encoder reset.

```
Val = PM_endat22_setupCommand(
    ENCODER_RECEIVE_RESET,
    0xA1,          // Any
    0x5555,       // Any
    0);           // No. of additional data - 0 for EnDat21 commands
```

- **ENCODER\_SEND\_POSITION\_VALUES**

The following mode command requests position values without additional data.

```
Val = PM_endat22_setupCommand(
    ENCODER_SEND_POSITION_VALUES,
    0x0,          // Not applicable
    0x0,          // Not applicable
    0);           // No. of additional data - 0 for EnDat21 commands
```

- **ENCODER\_RECEIVE\_TEST\_COMMAND**

This command is used as first step in interrogating the test values. Encoder receive test command sent along with the port address will to be interrogated for test values.

```
Val = PM_endat22_setupCommand(
    ENCODER_RECEIVE_TEST_COMMAND,
    0x0,          // Port address
    0x0,          // Any
    0);           // No. of additional data - 0 for EnDat21 commands
```

- **ENCODER\_SEND\_TEST\_VALUES**

The following mode is necessary to interrogate test values.

```
Val = PM_endat22_setupCommand(
    ENCODER_SEND_TEST_VALUES,
    0x0,          // Not applicable
    0x0,          // Not applicable
    0);           // No. of additional data - 0 for EnDat21 commands
```

- **ENCODER\_SEND\_POSITION\_VALUES\_WITH\_ADDITIONAL\_DATA**

This mode command can be used to request additional data, such as diagnostic values, commutating values, and acceleration values etc. See the encoder specifications to determine which additional data are supported by the encoder. This information is also saved in the encoder memory for parameters according to EnDat 2.2 (word 0 and word 1).

```
Val = PM_endat22_setupCommand(
    ENCODER_SEND_POSITION_VALUES_WITH_ADDITIONAL_DATA,
    0x0,          // Not applicable
    0x0,          // Not applicable
    0);           // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

- **ENCODER\_SEND\_POSITION\_VALUES\_AND\_SELECTION\_OF\_THE\_MEMORY\_AREA**

This mode command is necessary in order to request a position value and to select the memory area or block address in the same cycle.

```
Val = PM_endat22_setupCommand(
    ENCODER_SEND_POSITION_VALUES_AND_SELECTION_OF_THE_MEMORY_AREA,
    0x0,          // MRS code
    0x0,          // Block address
    0);          // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

- **ENCODER\_SEND\_POSITION\_VALUES\_AND\_RECEIVE\_PARAMETER**

This mode command is necessary in order to request a position value and write parameters in the same cycle.

```
Val = PM_endat22_setupCommand(
    ENCODER_SEND_POSITION_VALUES_AND_RECEIVE_PARAMETER,
    0x0,          // Address
    0x0,          // Parameters
    0);          // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

- **ENCODER\_SEND\_POSITION\_VALUES\_AND\_SEND\_PARAMETER**

This mode command is necessary if you want to request a position value and in the same cycle send parameters necessary for read access.

```
Val = PM_endat22_setupCommand(
    ENCODER_SEND_POSITION_VALUES_AND_SEND_PARAMETER,
    0x0,          // Address
    0x0,          // Any
    0);          // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

- **ENCODER\_SEND\_POSITION\_VALUES\_AND\_RECEIVE\_TEST\_COMMAND**

This mode command is necessary in order to request position values and write a test command in the same cycle.

```
Val = PM_endat22_setupCommand(
    ENCODER_SEND_POSITION_VALUES_AND_RECEIVE_TEST_COMMAND,
    0x0,          // Port address
    0x0,          // Any
    0);          // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

- **ENCODER\_SEND\_POSITION\_VALUES\_AND\_RECEIVE\_ERROR\_RESET**

This mode command is necessary in order to request position values and reset errors in the same cycle.

```
Val = PM_endat22_setupCommand(
    ENCODER_SEND_POSITION_VALUES_AND_RECEIVE_ERROR_RESET,
    0x0,          // Any
    0x0,          // Any
    0);          // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

- **ENCODER\_RECEIVE\_COMMUNICATION\_COMMAND**

This mode command is necessary to send communication data. After the address has been assigned with the "Write parameters" mode command, all other mode commands for data exchange can be used. Only the encoder with the previously selected address reacts to the following mode commands, until a new address is given.

```
Val = PM_endat22_setupCommand(
    ENCODER_RECEIVE_COMMUNICATION_COMMAND,
    0x0,          // Address
    0x0,          // Instructions
    0);          // Zero
```

## PM\_endat22\_receiveData

### Description

Function for unpacking and populating the EnDat data structure with the data received from Encoder. This function will be called when the data from Encoder is available in the SPI data buffer and transferred to rdata array of ENDAT\_DATA\_STRUCT. Upon the function call, received data is unpacked as per the current command and unpacked results are stored accordingly.

**NOTE:** The format for transfer of position values varies in length depending on the encoder model. The number of clock pulses required for transferring the position value (without mode, start, error or CRC bits) must be read from the encoder manufacturer's memory area. This information should be stored in `ndat22Data.position_clocks`. Encoder transmits the position value with LSB first. The values stored in `endat22Data.position_hi` and `endat22Data.position_lo` however are already bit reversed and right justified. This is applicable to all the commands that receive position information in both EnDat21 and EnDat22 formats. For further details, see the [Heidenhain](#) documentation.

### Definition

```
void PM_endat22_receiveData (uint16_t command, uint16_t nAddData);
```

### Parameters

Input	
command	Mode command for the EnDat transfer done. This function should be called with the same mode command that was used to initiate the transfer.
nAddData	Number of additional data enabled (0, 1 or 2 depending on the number of additional data enabled or not)
Return	
val	If the incorrect command value is passed to this function it will return zero. For all other cases, function returns a value of one.

### Usage

#### Example Code:

```
Val = PM_endat22_setupCommand(SELECTION_OF_MEMORY_AREA, 0xA1, 0x5555, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(SELECTION_OF_MEMORY_AREA, 0);
crc5_result1 = PM_endat22_getCrcNorm(endat22Data.address, endat22Data.data,
table1);
```

Below are few examples of how the `PM_endat22_setupData` function is used with various mode commands. For further details, see the [Heidenhain](#) documentation.

Define Endat Data structure during initialization.  
`ENDAT_DATA_STRUCT endat22Data;`

- **SELECTION\_OF\_MEMORY\_AREA**

In order to send or read parameters, the memory area must first be selected. This is done with the mode command, followed by a code for the memory area to be selected: the Memory Range Select (MRS) code. The encoder acknowledges the command.

```
Val = PM_endat22_receiveData(
    SELECTION_OF_MEMORY_AREA,
    0); // No. of additional data - 0 for EnDat21 commands
```

Unpacked data stored in EnDat Data structure for this command:

```

endat22Data.address      = MRS code
endat22Data.data         = Any
endat22Data.data_crc    = CRC for the received data

```

- **ENCODER\_SEND\_PARAMETER**

This mode command is required for reading parameters of the encoder. It is read from the memory area that was last selected as being valid. The encoder acknowledges the command.

```

Val = PM_endat22_receiveData(
    ENCODER_SEND_PARAMETER,
    0);          // No. of additional data - 0 for EnDat21 commands

```

Unpacked data stored in EnDat Data structure for this command:

```

endat22Data.address      = Address Acknowledgment
endat22Data.data         = Parameters
endat22Data.data_crc    = CRC for the received data

```

- **ENCODER\_RECEIVE\_PARAMETER**

This mode command is required for writing parameters of the encoder. It is written to the memory area that was last selected as being valid. The encoder acknowledges the command.

```

Val = PM_endat22_receiveData(
    ENCODER_RECEIVE_PARAMETER,
    0);          // No. of additional data - 0 for EnDat21 commands

```

Unpacked data stored in EnDat Data structure for this command:

```

endat22Data.address      = Address Acknowledgment
endat22Data.data         = Parameter Acknowledgment
endat22Data.data_crc    = CRC for the received data

```

- **ENCODER\_RECEIVE\_RESET**

This mode command is required for executing encoder reset.

```

Val = PM_endat22_receiveData(
    ENCODER_RECEIVE_RESET,
    0);          // No. of additional data - 0 for EnDat21 commands

```

Unpacked data stored in EnDat Data structure for this command:

```

endat22Data.address      = Any
endat22Data.data         = Anyt
endat22Data.data_crc    = CRC for the received data

```

- **ENCODER\_SEND\_POSITION\_VALUES**

The following mode command requests position values without additional data.

```

Val = PM_endat22_receiveData(
    ENCODER_SEND_POSITION_VALUES,
    0);          // No. of additional data - 0 for EnDat21 commands

```

Unpacked data stored in EnDat Data structure for this command:

```

endat22Data.address      = Higher 32 bits of position
endat22Data.data         = Lower 32 bits of position
endat22Data.data_crc    = CRC for the received position data

```

- **ENCODER\_RECEIVE\_TEST\_COMMAND**

This command is used as first step in interrogating the test values. The Encoder receive test command sent along with the port address will to be interrogated for test values.

```
Val = PM_endat22_receiveData(
    ENCODER_RECEIVE_TEST_COMMAND,
    0);          // No. of additional data - 0 for EnDat21 commands
```

Unpacked data stored in EnDat Data structure for this command:

endat22Data.address           = Port address acknowledgment  
endat22Data.data               = Any  
endat22Data.data\_crc          = CRC for the received data

- **ENCODER\_SEND\_TEST\_VALUES**

The following mode is necessary to interrogate test values.

```
Val = PM_endat22_receiveData(
    ENCODER_SEND_TEST_VALUES,
    0);          // No. of additional data - 0 for EnDat21 commands
```

Unpacked data stored in EnDat Data structure for this command:

endat22Data.address           = Higher 8 bits of test data  
endat22Data.data               = Lower 32 bits of test data  
endat22Data.data\_crc          = CRC for the received test data

---

**NOTE:** Test values transmitted by the encoder are always 40 bits.

---

- **ENCODER\_SEND\_POSITION\_VALUES\_WITH\_ADDITIONAL\_DATA**

This mode command can be used to request additional data, such as diagnostic values, commutating values, and acceleration values etc. See the encoder specifications to determine which additional data are supported by the encoder. This information is also saved in the encoder memory for parameters according to EnDat 2.2 (word 0 and word 1).

```
Val = PM_endat22_receiveData(
    ENCODER_SEND_POSITION_VALUES_WITH_ADDITIONAL_DATA,
    0);          // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

Unpacked data stored in EnDat Data structure for this command:

endat22Data.address           = Higher 8 bits of test data  
endat22Data.data               = Lower 32 bits of test data  
endat22Data.data\_crc          = CRC for the received position data  
endat22Data.additional\_data1   = Additional data 1  
endat22Data.additional\_data1\_crc = CRC for additional data 1  
endat22Data.additional\_data2   = Additional data 2  
endat22Data.additional\_data2\_crc = CRC for additional data 2

- **ENCODER\_SEND\_POSITION\_VALUES\_AND\_SELECTION\_OF\_THE\_MEMORY\_AREA**

This mode command is necessary in order to request a position value and to select the memory area or block address in the same cycle.

```
Val = PM_endat22_receiveData(
    ENCODER_SEND_POSITION_VALUES_AND_SELECTION_OF_THE_MEMORY_AREA,
    0); // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

Unpacked data stored in EnDat Data structure for this command:

endat22Data.position_hi	= Higher 32 bits of position
endat22Data.position_lo	= Lower 32 bits of position
endat22Data.data_crc	= CRC for the received position data
endat22Data.additional_data1	= Additional data 1
endat22Data.additional_data1_crc	= CRC for additional data 1
endat22Data.additional_data2	= Additional data 2
endat22Data.additional_data2_crc	= CRC for additional data 2

- **ENCODER\_SEND\_POSITION\_VALUES\_AND\_RECEIVE\_PARAMETER**

This mode command is necessary in order to request a position value and write parameters in the same cycle.

```
Val = PM_endat22_receiveData(
    ENCODER_SEND_POSITION_VALUES_AND_RECEIVE_PARAMETER,
    0); // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

Unpacked data stored in EnDat Data structure for this command:

endat22Data.position_hi	= Higher 32 bits of position
endat22Data.position_lo	= Lower 32 bits of position
endat22Data.data_crc	= CRC for the received position data
endat22Data.additional_data1	= Additional data 1
endat22Data.additional_data1_crc	= CRC for additional data 1
endat22Data.additional_data2	= Additional data 2
endat22Data.additional_data2_crc	= CRC for additional data 2

- **ENCODER\_SEND\_POSITION\_VALUES\_AND\_SEND\_PARAMETER**

This mode command is necessary if you want to request a position value and in the same cycle send parameters necessary for read access.

```
Val = PM_endat22_receiveData(
    ENCODER_SEND_POSITION_VALUES_AND_SEND_PARAMETER,
    0); // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

Unpacked data stored in EnDat Data structure for this command:

endat22Data.position_hi	= Higher 32 bits of position
endat22Data.position_lo	= Lower 32 bits of position
endat22Data.data_crc	= CRC for the received position data
endat22Data.additional_data1	= Additional data 1
endat22Data.additional_data1_crc	= CRC for additional data 1
endat22Data.additional_data2	= Additional data 2
endat22Data.additional_data2_crc	= CRC for additional data 2

- **ENCODER\_SEND\_POSITION\_VALUES\_AND\_RECEIVE\_TEST\_COMMAND**

This mode command is necessary in order to request position values and write a test command in the same cycle.

```
Val = PM_endat22_receiveData(
    ENCODER_SEND_POSITION_VALUES_AND_RECEIVE_TEST_COMMAND,
    0);          // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

Unpacked data stored in EnDat Data structure for this command:

endat22Data.position_hi	= Higher 32 bits of position
endat22Data.position_lo	= Lower 32 bits of position
endat22Data.data_crc	= CRC for the received position data
endat22Data.additional_data1	= Additional data 1
endat22Data.additional_data1_crc	= CRC for additional data 1
endat22Data.additional_data2	= Additional data 2
endat22Data.additional_data2_crc	= CRC for additional data 2

- **ENCODER\_SEND\_POSITION\_VALUES\_AND\_RECEIVE\_ERROR\_RESET**

This mode command is necessary in order to request position values and reset errors in the same cycle.

```
Val = PM_endat22_receiveData(
    ENCODER_SEND_POSITION_VALUES_AND_RECEIVE_ERROR_RESET,
    0);          // No. of additional data (0, 1 or 2 depending on no.of
additional data enabled)
```

Unpacked data stored in EnDat Data structure for this command:

endat22Data.position_hi	= Higher 32 bits of position
endat22Data.position_lo	= Lower 32 bits of position
endat22Data.data_crc	= CRC for the received position data
endat22Data.additional_data1	= Additional data 1
endat22Data.additional_data1_crc	= CRC for additional data 1
endat22Data.additional_data2	= Additional data 2
endat22Data.additional_data2_crc	= CRC for additional data 2

- **ENCODER\_RECEIVE\_COMMUNICATION\_COMMAND**

This mode command is necessary to send communication data. After the address has been assigned with the "Write parameters" mode command, all other mode commands for data exchange can be used. Only the encoder with the previously selected address reacts to the following mode commands, until a new address is given.

```
Val = PM_endat22_receiveData(
    ENCODER_RECEIVE_COMMUNICATION_COMMAND,
    0);          // Zero
```

Unpacked data stored in EnDat Data structure for this command:

endat22Data.position_hi	= Address Acknowledgment
endat22Data.position_lo	= Instructions Acknowledgment
endat22Data.data_crc	= CRC for the received data

## PM\_endat22\_startOperation

**Description**

This function initiates the EnDat transfer. This function should only be called after PM\_endat22\_setupCommand. Hence the PM\_endat22\_startOperation function kick starts the EnDat transaction that was set up earlier by PM\_endat22\_setupCommand. Note that the setup up and start operation are separate function calls. User can setup the EnDat transfer when needed and start the actual transfer using this function call, as needed, at a different time.

**Definition**

```
void PM_endat22_startOperation(void);
```

**Parameters**

<b>Input</b>	
	None
<b>Return</b>	
	None

**Usage**
**Example Code:**

```
Val = PM_endat22_setupCommand(SELECTION_OF_MEMORY_AREA, 0xA1, 0x5555, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(SELECTION_OF_MEMORY_AREA, 0);
crc5_result1 = PM_endat22_getCrcNorm(endat22Data.address, endat22Data.data,
table1);
```

This function clears the endat22Data.dataReady flag zero when called. This flag should subsequently be set by the SPI Interrupt service routine when the data is received from the encoder. You can poll for this flag to know if the data from the encoder is successfully received after the PM\_endat22\_startOperation function call.

## PM\_endat22\_setupPeriph

### Description

Setup for SPI, CLB and other interconnect XBARs for EnDat are performed with this function during system initialization. This function needed to be called after every system reset. No EnDat transactions will be performed until the setup peripheral function is called.

Definition `void PM_endat22_setupPeriph (void);`

### Parameters

<b>Input</b>	
	None
<b>Return</b>	
	None

### Usage

#### Example Code:

```
endat22Data.spi = &SpibRegs
  PM_endat22_setupPeriph();
```

This function clears the endat22Data.dataReady flag zero when called. This flag should subsequently be set by the SPI Interrupt service routine when the data is received from the encoder. You can poll for this flag to know if the data from the encoder is successfully received after the PM\_endat22\_startOperation function call.

## PM\_endat22\_setFreq

**Description**

Function to set the EnDat clock frequency. EnDat transfers typically start with low frequency during initialization and switch to higher frequency during runtime.

Typical frequencies used during initialization and runtime:

- Used during initialization (~200 KHz)
- Used during application (~8 MHz) C2000 EnDat implementation currently supports 8 MHz only, irrespective of cable length.

**Definition**

```
void PM_endat22_setFreq(uint32_t Freq_us);
Endat Clock Frequency = SYSCLK/(4* Freq_us
```

**Parameters**

<b>Input</b>	
	Freq_us: A clock divider for the system clock sets Endat Clock Frequency = SYSCLK/(4* Freq_us)
<b>Return</b>	
	None

**Usage**
**Example Code:**

```
//during initialization and delay compensation
PM_endat22_setFreq(ENDAT_INIT_FREQ_DIVIDER);

//during runtime
PM_endat22_setFreq(ENDAT_RUNTIME_FREQ_DIVIDER);
```

## PM\_endat22\_getDelayCompVal

**Description** This function is used while performing delay compensation when long cables are used. This function returns the measured delay from rising edge of EnDat Clock to the start bit received. Please refer to examples provided on usage and performing delay compensation. Refer to EnDat documentation from Heidenhain for cable delays and propagation requirements.

**Definition** `uint16_t PM_endat22_getDelayCompVal(void);`

### Parameters

<b>Input</b>	
	Freq_us: A clock divider for the system clock sets Endat Clock Frequency = SYSCLK/(4* Freq_us)
<b>Return</b>	
	None

### Usage

#### Example Code:

```
//during initialization and delay compensation
delay = PM_endat22_getDelayCompVal();

endat22Data.delay_comp = delay;
```

---

**NOTE:** Propagation delay should be measured using this function multiple times and the average value needs to be update into endat22Data.delay\_comp field before switching to higher frequency operation. For delay compensation, see the TI provided examples on usage of this function.

---

## 4 Using PM\_ENDAT22 Library

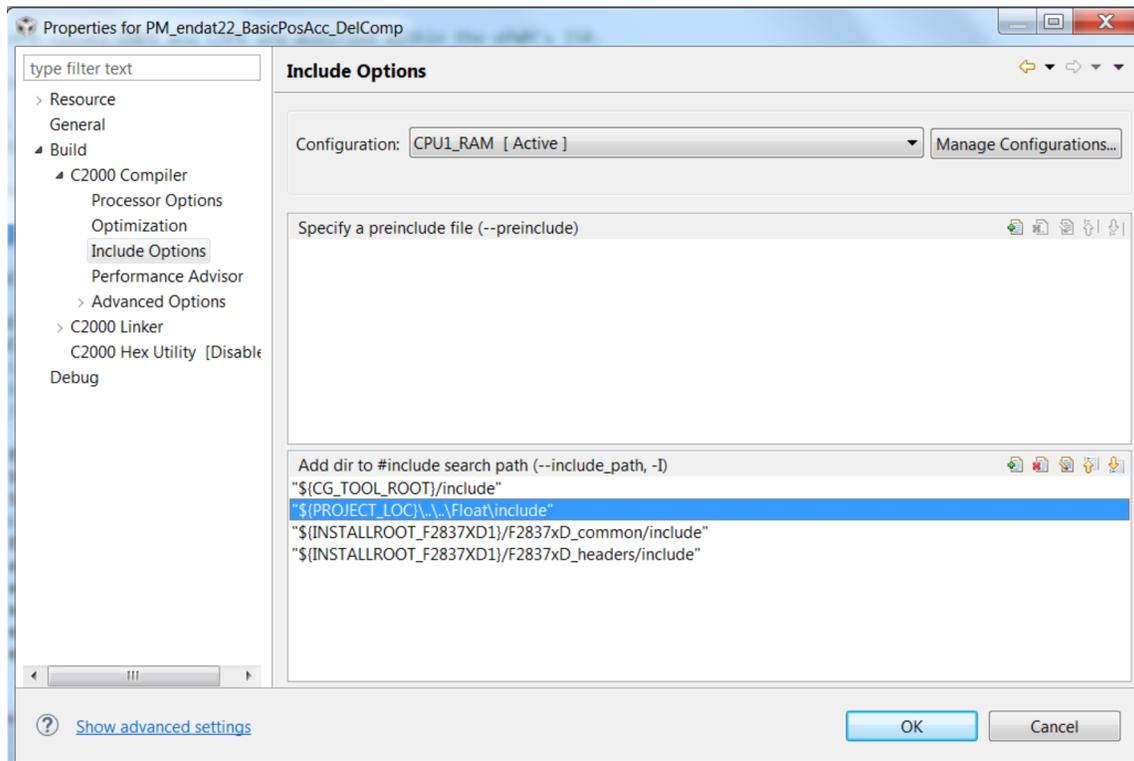
### 4.1 Adding EnDat22 Lib to the Project

1. Include the library in *{ProjectName}-Includes.h*.

```
#include "PM_endat22_Include.h"
```

Add the PM\_endat22 library path in the include paths under Project Properties → Build → C2000 Compiler → Include Options.

Path for the library: C:\ti\controlSUITE\libs\app\_libs\position\_manager\v1\_00\_00\_00\endat22\Float\lib

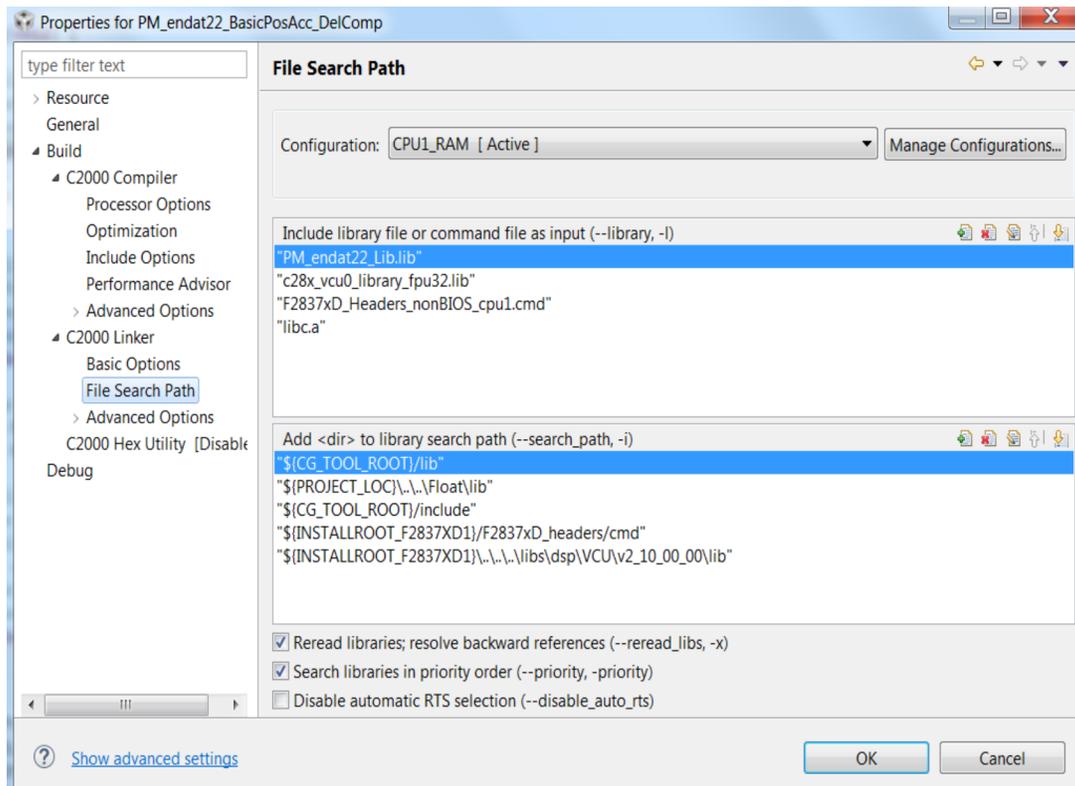


**Figure 3. Compiler Options for a Project Using PM EnDat22 Library**

**NOTE:** Note that the exact location may vary depending on where controlSUITE is installed and which other libraries the project is using.

Link EnDat22 Library: (PM\_endat22\_lib.lib) to the project, it is located at:  
controlSUITE\libs\app\_libs\position\_manager\v01\_00\_00\_00\endat22\Float\lib

2. **Figure 4** shows the changes to the linker options that are required to include the EnDat22 library.



**Figure 4. Adding PM EnDat22 Library to the Linker Options in CCS Project**

**NOTE:** Note that the exact location may vary depending on where controlSUITE is installed and which other libraries the project is using.

## 4.2 Steps for Initialization

The following steps are needed for initialization and proper functioning of the EnDat22 library functions. For more details, see the examples provided along with the library.

1. Create and add module structure to {ProjectName}-Main.c for EnDat interface.

```
ENDAT_DATA_STRUCT endat22Data;
```

2. Define the type of the encoder. If EnDat21, this should be set to 21. If EnDat22, this should be set to 22.

```
#define ENCODER_TYPE      22
```

3. Define the frequency of the EnDat clock during initialization and run.

```
#define ENDAT_RUNTIME_FREQ_DIVIDER      6
#define ENDAT_INIT_FREQ_DIVIDER        250
```

```
//Endat Initial Clock Frequency = SYSCLK/(4*ENDAT_INIT_FREQ_DIVIDER) -
used during initialization (~200KHz)
//Endat Clock Frequency = SYSCLK/(4*ENDAT_FREQ_DIVIDER) -
used during applicaiton (~8 MHz)
//Set ENDAT_FREQ_DIVIDER, ENDAT_INIT_FREQ_DIVIDER accordingly. Only even values greater than 6
are supported.
```

4. Set the SPI instance to be used for EnDat communication. For usage on TMDXIDDK379D, the SPI instance has to be set to SpiB and enable the SpiB receive FIFO interrupt.

```

endat22Data.spi = &SpiBRegs;
PieVectTable.SPIB_RX_INT = &spiRxFifoIsr;
PieCtrlRegs.PIECTRL.bit.ENPIE = 1;    // Enable the PIE block
PieCtrlRegs.PIEIER6.bit.INTx3 = 1;    // Enable PIE Group 6, INT 9
IER = 0x20;                            // Enable CPU INT6
EINT;
    
```

---

**NOTE:** Alternatively, users can also poll for the Interrupt Flag and not necessarily use interrupt. Make sure the SPIRXFIFO contents are copied into endat22Data.rdata after the flag is set. This is required to be executed before calling PM\_endat22\_receiveData function. Also, interrupt flag needs to be cleared to get the SPI ready for next EnDat transaction.

---

```

for (i=0;i<=endat22Data.fifo_level;i++){endat22Data.rdata[i]= endat22Data.spi->SPIRXBUF;}
endat22Data.spi->SPIFFRX.bit.RXFFINTCLR=1; // Clear Interrupt flag
    
```

5. Enable clocks to EPWM Instances 1, 2, 3 and 4.

```

CpuSysRegs.PCLKCR2.bit.EPWM1 = 1;
CpuSysRegs.PCLKCR2.bit.EPWM2 = 1;
CpuSysRegs.PCLKCR2.bit.EPWM3 = 1;
CpuSysRegs.PCLKCR2.bit.EPWM4 = 1;
    
```

6. Initialize and setup EnDat peripheral configuration by calling PM\_endat22\_setupPeriph function.

```

endat22Data.spi = &SpiBRegs;
PM_endat22_setupPeriph();
    
```

7. Setup GPIOs needed for EnDat configuration, which is required for TMDXIDDK379D. The GPIOs used for SPI has to be changed based on the chosen SPI instance. GPIO6, GPIO7 and GPIO34 have to be configured always.

```

GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1;    // Configure GPIO6 as EnDat Clk master
GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 1;    // Configure GPIO7 as SPI Clk slave
    
```

```

GpioCtrlRegs.GPAGMUX2.bit.GPIO24 = 1;
GpioCtrlRegs.GPAGMUX2.bit.GPIO25 = 1;
GpioCtrlRegs.GPAGMUX2.bit.GPIO26 = 1;
GpioCtrlRegs.GPAGMUX2.bit.GPIO27 = 1;
    
```

```

GpioCtrlRegs.GPAMUX2.bit.GPIO24 = 2;    // Configure GPIO24 as SPISIMOB
GpioCtrlRegs.GPAMUX2.bit.GPIO25 = 2;    // Configure GPIO25 as SPISOMIB
GpioCtrlRegs.GPAMUX2.bit.GPIO26 = 2;    // Configure GPIO26 as SPICLKB
GpioCtrlRegs.GPAMUX2.bit.GPIO27 = 2;    // Configure GPIO27 as SPISTEB
    
```

```

GpioCtrlRegs.GPAQSEL2.bit.GPIO24 = 3;    // Asynch input GPIO24 (SPISIMOB)
GpioCtrlRegs.GPAQSEL2.bit.GPIO25 = 3;    // Asynch input GPIO25 (SPISOMIB)
GpioCtrlRegs.GPAQSEL2.bit.GPIO26 = 3;    // Asynch input GPIO26 (SPICLKB)
GpioCtrlRegs.GPAQSEL2.bit.GPIO27 = 3;    // Asynch input GPIO27 (SPISTEB)
    
```

```

GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 1;    // Configure GPIO34 as EnDat TxEN
    
```

```

GpioCtrlRegs.GPBDIR.bit.GPIO32 = 1;    // Configure GPIO32 as EnDat Pwr Ctl
    
```

8. Setup XBAR as shown below. The GPIOs used for SPI has to be changed based on the chosen SPI instance.

```

//SPISIMOB on GPIO24 connected to CLB4 i/p 1 via XTRIP and CLB X-Bars
// XTRIP1 is used inside the CLB for monitoring received data from Encoder.
InputXbarRegs.INPUT1SELECT = 24;        // GPTRIP XBAR TRIP1 -> GPIO24
    
```

### 9. Initialization for CRC related object and table generation.

```
//CRC tables to be used for CRC calculations
uint16_t table1[SIZEOFTABLE];

// Generate table for poly 1 and generated tables are placed in table1
PM_endat22_generateCRCTable(NBITS_POLY1, POLY1, table1);

-- Constants are defined in PM_endat22_Include.h as below.
#define NBITS_POLY1 5
#define POLY1 0x0B
#define SIZEOFTABLE 256
```

### 10. Turn the power ON. GPIO used for Power control can be changed based on the hardware. GPIO32 is used for power control on TMDXIDDK379D.

```
// Power up EnDat 5v supply through GPIO32
GpioDataRegs.GPBDAT.bit.GPIO32 = 1;
```

## 4.3 Using the Library Functions

Below are some examples for commonly used library functions:

#### 1. Read the position values using EnDat21 command.

```
Val = PM_endat22_setupCommand(ENCODER_SEND_POSITION_VALUES, 0, 0, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_SEND_POSITION_VALUES, 0);
```

#### 2. Read the position values using EnDat22 command.

```
Val = PM_endat22_setupCommand(ENCODER_SEND_POSITION_VALUES_AND_SELECTION_OF_THE_MEMORY_AREA,
0xA1, 0, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_SEND_POSITION_VALUES_AND_SELECTION_OF_THE_MEMORY_AREA, 0)
```

#### 3. Execute the Encoder send parameter EnDat21 command.

```
Val = PM_endat22_setupCommand(ENCODER_SEND_PARAMETER, 0xD, 0xAAAA, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_SEND_PARAMETER, 0);
```

```
crc5_result1 = PM_endat22_getCrcNorm(endat22Data.address, endat22Data.data, table1);
```

#### 4. Execute the Encoder receive parameter EnDat21 command.

```
Val = PM_endat22_setupCommand(ENCODER_RECEIVE_PARAMETER, 0x0, 0x0, 0); //data1=MRS; data2=any
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_RECEIVE_PARAMETER, 0);
```

```
crc5_result1 = PM_endat22_getCrcNorm(endat22Data.address, endat22Data.data, table1);
```

#### 5. Execute the Encoder receive test command - EnDat21 command.

```
Val = PM_endat22_setupCommand(ENCODER_RECEIVE_TEST_COMMAND, 0x1, 0x0, 0); //data1=MRS;
data2=any
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_RECEIVE_TEST_COMMAND, 0);
```

```
crc5_result1 = PM_endat22_getCrcNorm(endat22Data.address, endat22Data.data, table1);
```

#### 6. Execute the Encoder send test values EnDat21 command.

```
Val = PM_endat22_setupCommand(ENCODER_SEND_TEST_VALUES, 0x0, 0x0, 0); //data1=MRS; data2=any
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_SEND_TEST_VALUES, 0);
```

```
crc5_result1 = PM_endat22_getCrcTest(endat22Data.test_lo, endat22Data.test_hi,
endat22Data.error1, table1);
```

**7. Read the Encoder position values with additional data and selection of memory area - EnDat22 command,**

```
Val = PM_endat22_setupCommand(ENCODER_SEND_POSITION_VALUES_AND_SELECTION_OF_THE_MEMORY_AREA,
0xA1, 0, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_SEND_POSITION_VALUES_AND_SELECTION_OF_THE_MEMORY_AREA, 0);

crc5_result1 = PM_endat22_getCrcPos(endat22Data.position_clocks, 1, endat22Data.position_lo,
endat22Data.position_hi, endat22Data.error1, endat22Data.error2, table1);
```

**8. Read the Encoder position values with additional data and receive parameter - EnDat22 command.**

```
al = PM_endat22_setupCommand(ENCODER_SEND_POSITION_VALUES_AND_RECEIVE_PARAMETER, 0x0, 0, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_SEND_POSITION_VALUES_AND_RECEIVE_PARAMETER, 0);

crc5_result1 = PM_endat22_getCrcPos(endat22Data.position_clocks, 1, endat22Data.position_lo,
endat22Data.position_hi, endat22Data.error1, endat22Data.error2, table1);
```

**9. Read the Encoder position values with additional data and send parameter - EnDat22 command.**

```
Val = PM_endat22_setupCommand(ENCODER_SEND_POSITION_VALUES_AND_SEND_PARAMETER, 0xD, 0, 0);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_SEND_POSITION_VALUES_AND_SEND_PARAMETER, 0);

crc5_result1 = PM_endat22_getCrcPos(endat22Data.position_clocks, 1,
endat22Data.position_lo, endat22Data.position_hi, endat22Data.error1, endat22Data.error2,
table1);
```

**10. Read the Encoder position values with additional data EnDat22 command.**

```
Val = PM_endat22_setupCommand(ENCODER_SEND_POSITION_VALUES_WITH_ADDITIONAL_DATA, 0, 0, 2);
PM_endat22_startOperation();
while (endat22Data.dataReady != 1) {}
Val = PM_endat22_receiveData(ENCODER_SEND_POSITION_VALUES_WITH_ADDITIONAL_DATA, 2);

crc5_result1 = PM_endat22_getCrcPos(endat22Data.position_clocks, 1,
endat22Data.position_lo, endat22Data.position_hi, endat22Data.error1, endat22Data.error2,
table1);
```

## 5 Resource Requirements

The resources of the MCU (see [Table 6](#)) are consumed by the PM EnDat22 Library for implementing EnDat interface.

**Table 6. Resource Requirements**

Resource Name	Type	Purpose	Usage Restrictions
<b>Dedicated Resources</b>			
GPIO6	IO	EnDat Clock from master to Encoder	IO dedicated for EnDat
GPIO7	IO	SPI clock generated by MCU	IO dedicated for EnDat
EPWM4	IO	Internally for Clock generation	EPWM4 dedicated for EnDat
GPIO34	IO	EnDat Direction control for Data on IDDK	Dedicated IO for EnDat Direction control
<b>Configurable Resources</b>			
SPI	Module and IOs	One SPI instance to emulate EnDat interface (SPIB on IDDK)	Any instance of SPI can be chosen – Module and corresponding IOs will be dedicated for EnDat
GPIO32	IO	For EnDat Power control on IDDK	Can choose any IO power control
<b>Shared Resources</b>			
CPU and Memory	Module	Check CPU and Memory utilization for various functions	Application to ensure enough CPU cycles and Memory are allocated
ECAP2	Module/IO	ECAP2 output	ECAP2 o/p path dedicated for Endat. Can only be used as Input in application.
Input XBAR	Module/IO	To be connected to SPISIMO of the corresponding SPI instance dedicated for EnDat	INPUTXBAR1 is used for EnDat implementation. Remaining inputs are available for application use
Output XBAR	Module/IO	Bringing out EnDat TxEn (Direction Control) signal on GPIO32 via OUTPUT1 of Output XBAR	OUTPUT1 is used for EnDat implementation. Remaining outputs are available for application use

## 6 Test Report

[Table 7](#) lists tests with various types of encoders; cable lengths are performed at [Heidenhain Labs](#). Tests include basic command set exercising and reading position values, with additional data if applicable.

**Table 7. Test Report**

Encoder Name	Type	Resolution	Cable Length in Meters <sup>(1)</sup>	Max EnDat Clock in MHz	Test Results
ROC425	Rotary	25 bits	70m	8 MHz	Pass
LC415	Linear	35 bits	70m	8 MHz	Pass
RCN8310	Rotary	29 bits	70m	8 MHz	Pass
ROQ437	Multi Turn	25 bits, 12 bits(Turns)	70m	8 MHz	Pass
LIC211	Linear	32 bits	70m	8 MHz	Pass
ROC413	Rotary	13 bits	70m	8 MHz	Pass

<sup>(1)</sup> Cable lengths up to 100m have also been tested with some of the encoders. Users can deploy longer cable lengths, perform delay compensation, switch to higher EnDat clock frequencies and perform tests.

## 7 FAQs

1. **Question:** Why is the position information not received or received incorrectly?

**Answer:** Perform or cross check the following:

- Make sure the Power-On procedure, specific to the Encoder, is used properly.
- Check for Encoder error messages and warnings
- Clear the Errors and Warnings if any, and perform Encoder Reset
- Check CRC of the received position data

For further help, see the [Heidenhain](#) documentation.

2. **Question:** What are the limitations of the EnDat22 implementation with this library?

**Answer:** While using the Endat22 library, see the *Key things to note ...* bulleted list in [Section 1.4](#).

3. **Question:** I have encountered a CRC failure. What could be the reason for this?

**Answer:** For details, see the Error Type I, II, III from [Heidenhain](#) Endat documentation. CRC errors can also occur due to noise in transmission path.

4. **Question:** How is the EnDat22 interface implemented on the TMS320F28379D devices?

**Answer:** For details, see [Section 1.4](#).

5. **Question:** Does TI share the source for the PM EnDat22 library to customers?

**Answer:** TI does not share the source code with customers. For specific requests, contact the TI sales team.

6. **Question:** Does TI provide application level interface functions for EnDat22?

**Answer:** Basic usage examples are provided along with the library. The example has high-level application layer functions for initialization, running full command set, setting and reading parameters, performing cable propagation delay compensation, enabling additional data, checking CRC for various types of received data, and so forth, which are sufficient for most of the applications. Any additional application layer functionality should be developed by using the basic driver interface commands provided in the PM EnDat22 Library.

## 8 References

- Power Supply Reference design for EnDat 2.2 encoder interfaces can be obtained from Texas Instruments (TIDA-00172): <http://www.ti.com/tool/TIDA-00172>
- Documentation from [Heidenhain](#):
  - Bidirectional Synchronous-Serial Interface for Position Encoders
  - EnDat Application notes
- C2000 DesignDRIVE Development Kit for Industrial Motor Control - TMDXIDDK379D:
  - *DesignDRIVE Development Kit IDDK Hardware Reference Guide* ([SPRUI23](#))
  - *DesignDRIVE Development Kit IDDK User's Guide* ([SPRUI24](#))
  - *Using Position Manager EnDat22 Library on IDDK Hardware User's Guide* ([SPRUI34](#))

## IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ("TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources 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.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products (<http://www.ti.com/sc/docs/stdterms.htm>), [evaluation modules](#), and [samples](http://www.ti.com/sc/docs/sampterm.htm) (<http://www.ti.com/sc/docs/sampterm.htm>).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © 2017, Texas Instruments Incorporated