TI Designs: TIDM-1008

EnDat 2.2 Absolute Encoder Master Interface Reference Design for C2000™ MCUs

TEXAS INSTRUMENTS

Description

C2000™ microcontroller (MCU) Position Manager technology offers an integrated solution to interface to the most popular digital and analog position sensors, which eliminates the necessity for external field programmable gate arrays (FPGAs) or application specific integrated circuit (ASICs). The Position Manager BoosterPack™ is a flexible, cost-effective platform intended for evaluating various encoder interfaces and is designed to work with multiple C2000 MCU LaunchPad™ development kits. This reference design's software specifically targets implementation of EnDat 2.2, which is a digital, bidirectional interface for position encoders. The highly-optimized and easy-to-use software library and examples included in this reference design enable EnDat2.2 position encoder operation using the Position Manager BoosterPack.

Resources

TIDM-1008	Design Folder
LAUNCHXL-F28379D	Tools Folder
SN65HVD78	Product Folder
TLV702	Product Folder
TPS22918-Q1	Product Folder

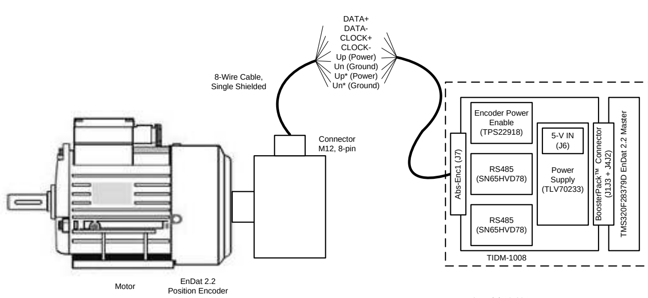
TI E2E™ Community — ASK Our E2E Experts

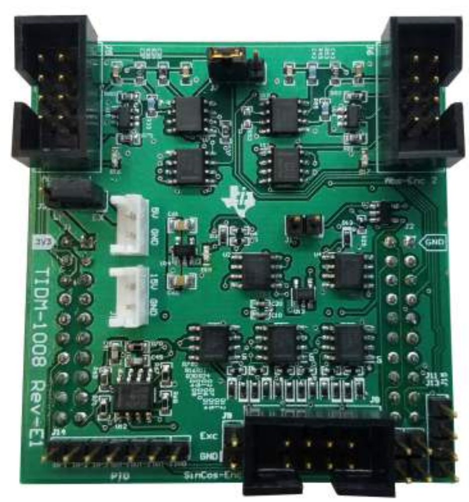
Features

- Flexible, Low-Voltage BoosterPack Evaluation Platform for Position Encoder Interfaces
- Integrated MCU Solution for EnDat 2.2 Without Additional FPGA Requirements
- Easy Interface-to-EnDat 2.2 Commands Through Driver Functions and Data Structure Provided by Library
- Library Support for Unpacking Received Data and Optimized Cyclic Redundancy Check (CRC) Algorithm
- Supports Clock Frequency up to 8 MHz and Verified Operation up to 100-m Cable Length
- Includes Evaluation Board and Software Example Showcasing EnDat22 Software Library

Applications

- Industrial
- Motor Drives

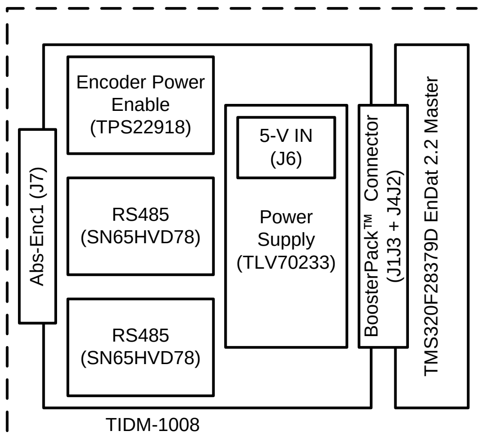




An IMPORTANT NOTICE at the end of this TI reference design addresses authorized use, intellectual property matters and other important disclaimers and information.

1 System Description

Industrial drives, like servo drives, require accurate, highly-reliable, and low-latency position feedback. The EnDat 2.2 interface from HEIDENHAIN™ is a digital, bidirectional interface standard for position or rotary encoders. The EnDat 2.2 is a pure serial, digital interface based on the RS-485 standard. The interface transmits position values or additional physical quantities and also allows reading and writing of the encoder's internal memory. The transmitted data types include absolute position, turns, temperature, parameters, diagnostics, and so on. Mode commands that the subsequent electronics, often referred to as EnDat 2.2 master, send to the encoder select the transmitted data types. TIDM-1008 acts as an EnDat 2.2 master and provides the subsequent electronics to interface an EnDat encoder with the F28379D LaunchPad. Figure 1 shows the major hardware blocks used in this design.



Copyright © 2017, Texas Instruments Incorporated

Figure 1. TIDM-1008 Hardware Blocks and Connectors

The position encoder with EnDat 2.2 connects to TIDM-1008 through a single, 8-wire, shielded cable. The eight wires used for communication include two wires for CLOCK+ and CLOCK- transmitted in differential format, two wires for DATA+ and DATA- that are transmitted in differential format, two wires Up and Un that are used for the encoder power supply and ground, and two wires Up* and Un* that are used for battery buffering or for parallel power supply lines to reduce the cables losses.

Texas Instruments' C2000 Position Manager EnDat22 (PM_endat22) library provides support for implementing the EnDat interface in subsequent electronics. The library makes up the software portion of TIDM-1008. The EnDat22 Library features an integrated MCU solution for the EnDat interface that meets HEIDENHAIN EnDat 2.1 and 2.2 digital interface protocol requirements. The library can support up to 8-MHz clock frequency independent of cable length—verified up to 100 m. This support is due to the integrated cable propagation delay compensation algorithm that is user configurable. The driver functions and data structure provided by the library allows other commands to be easily used. The library also uses an efficient and optimized CRC algorithm for both position and data CRC calculations with the capability of unpacking the received data and reversing the position data that is incorporated into library functions. This library solution is tuned for position control applications where position information is obtained from encoders every control cycle with better control of modular functions and timing.

There are several key concepts to note while using the EnDat22 Library. The library only supports the basic interface drivers for commands defined in EnDat22 specification. All the higher-level application software needs to be developed by users using 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. For any additional functionality or encoder usage not specified in this reference design, contact TI support team or refer to the TI E2E™community.

1.1 Key System Specifications

Table 1. Key System Specifications

PARAMETER	SPECIFICATIONS	DETAILS
Input voltage	5 V(1)	Section 3.2.1.1
Output voltage (encoder)	5 V	Section 3.2.1.1
Protocol supported	EnDat 2.2	Heidenhain
Frequency (encoder interface)	Approximately 8 MHz	Section 2.3.2
Encoder bits	EnDat 2.2 protocol standard	Heidenhain
CPU cycles	—	Section 2.3.3.2
Maximum cable length (tested)	100 m	—

(1) The time of the encoder connected to the TIDM-1008 determines the current limit of this supply. A generic, bench-top, adjustable power supply with an adjustable current limit is recommended.

2 System Overview

2.1 Block Diagram

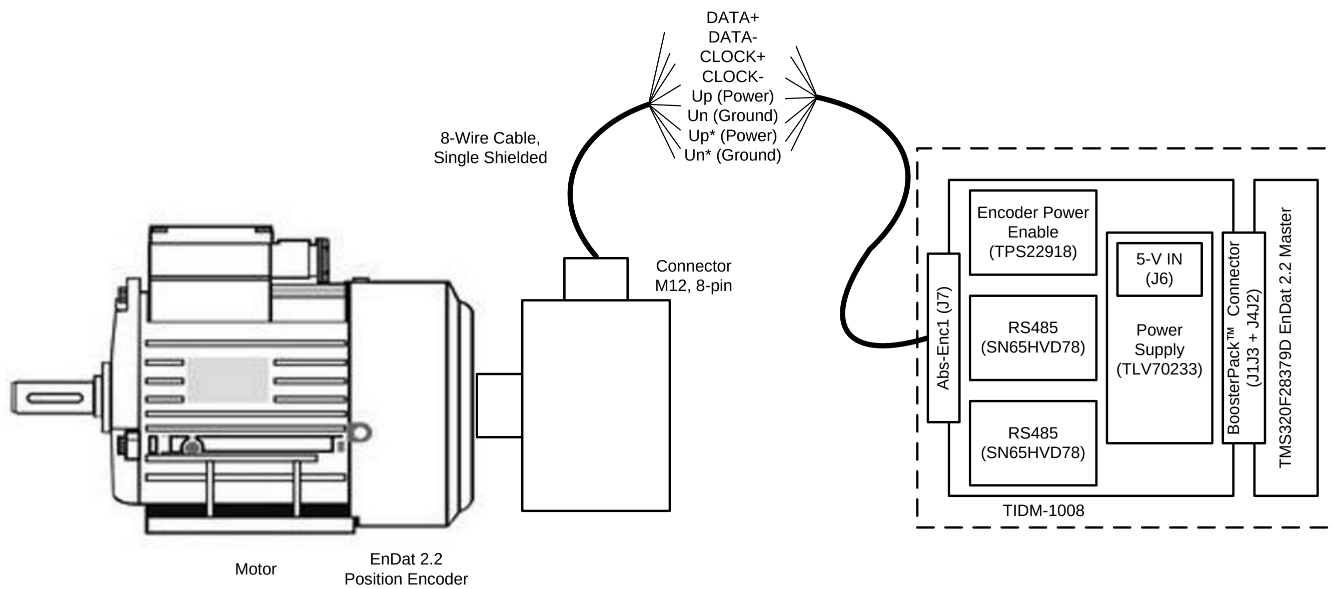


Figure 2. TIDM-1008 System Block Diagram

2.2 Highlighted Products

2.2.1 LAUNCHXL-F28379D

This development kit is based on the Delfino™ TMS320F28379D MCU, which provides 800MIPS of total system performance between dual 200-MHz, C28x CPUs and dual 200-MHz, real-time-control coprocessors (CLA). This powerful MCU contains 1MB of onboard flash and includes highly-differentiated peripherals, such as 16-bit or 12-bit analog-to-digital converters (ADCs), comparators, 12-bit digital-to-analog converters (DACs), delta-sigma sinc filters, HRPWMs, eCAPs, eQEPs, CANs, and more.

2.2.2 SN65HVD78

The SN65HVD78 combines a differential driver and a differential receiver, which operate from a single, 3.3-V power supply. The driver differential outputs and the receiver differential inputs are connected internally to form a bus port suitable for half-duplex (two-wire bus) communication. These devices feature a wide, common-mode voltage range, which make the devices suitable for multipoint applications over long cable runs.

2.2.3 TLV702

The TLV702 series of low-dropout (LDO) linear regulators are low-quiescent current devices with excellent line and load transient performance. All device versions have thermal shutdown and current limit for safety. The devices regulate to specified accuracy with no output load.

2.2.4 TPS22918-Q1

The TPS22918-Q1 is a single-channel load switch with configurable rise time and configurable quick output discharge. The device contains an N-channel MOSFET that can support a maximum continuous current of 2 A. The switch is controlled by an on and off input, which is capable of interfacing directly with low-voltage control signals.

2.3 Design Considerations

2.3.1 TIDM-1008 Board Implementation

The TIDM-1008 board is identical to the Position Manager BoosterPack (BOOSTXL-POSMGR), which means the TIDM-1008 board is capable of interfacing with several other position encoder types. The board is fully-populated by default for future compatability. This reference design focuses on EnDat 2.2 and the hardware blocks not mentioned in this document should be ignored. Software support for the other types of position encoder interfaces will be the subject of future reference designs. Table 2 describes the connectors on TIDM-1008 and BOOSTXL-POSMGR and their functions.

Table 2. TIDM-1008 Board and BOOSTXL-POSMGR Connectors

CONNECTOR	DESCRIPTION	RELEVANT TI DESIGNS AND HARDWARE
Abs-Enc-1 (J7)	EnDat 2.1, EnDat 2.2, other absolute encoders	TIDM-1008, BOOSTXL-POSMGR
Abs-Enc-2 (J8)	EnDat 2.1, EnDat 2.2, other absolute encoders	Future TID + BOOSTXL-POSMGR
Abs-Enc-2 Breakout (J10)	Allows x2 absolute encoders at site two using jumpers	Future TID + BOOSTXL-POSMGR
SinCos (J14)	SinCos encoder	Future TID + BOOSTXL-POSMGR
Resolver (J14 + J15)	Resolver interface with 15-V excitation circuitry	Future TID + BOOSTXL-POSMGR
PTO (J17)	Pulse train output	Future TID + BOOSTXL-POSMGR
J1 J3 + J4 J2	BoosterPack connector	All Designs, BOOSTXL-POSMGR
J6	5-V DC supply input	All Designs, BOOSTXL-POSMGR
J16	15-V DC resolver excitation input	Future TID + BOOSTXL-POSMGR

Figure 3 describes the encoder support on each site of the LaunchPad.

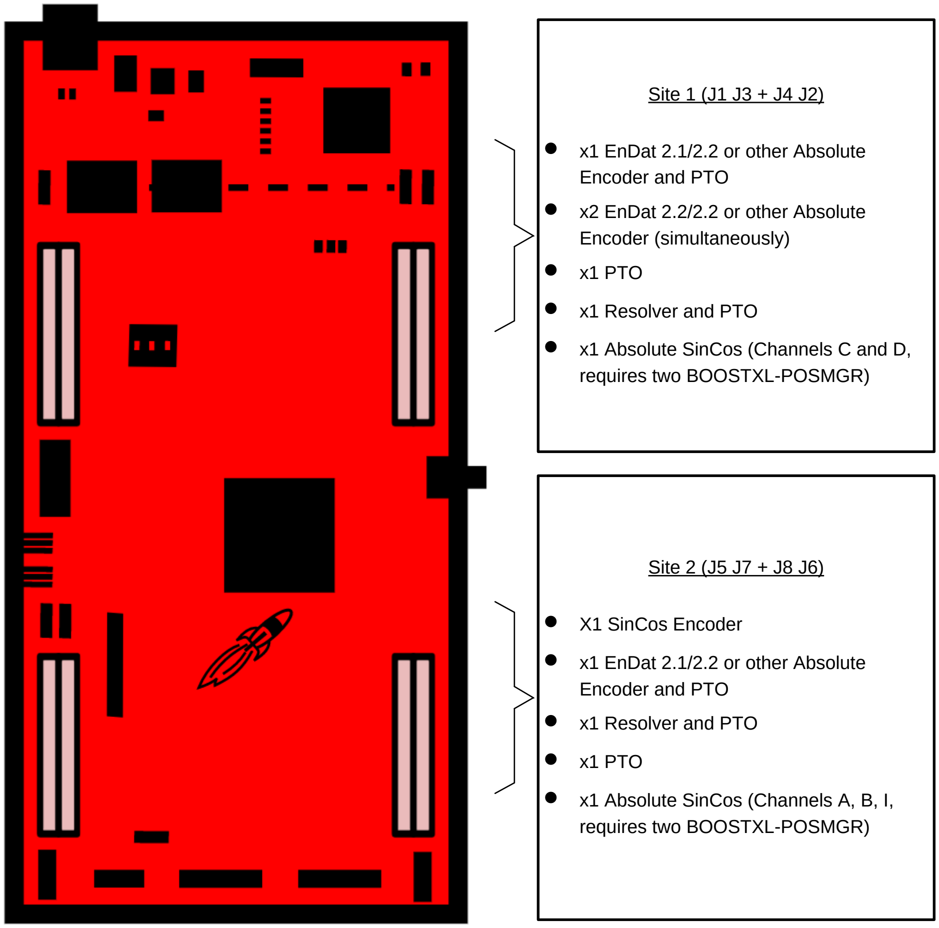


Figure 3. TIDM-1008 Board and BOOSTXL-POSMGR Encoder Support

2.3.2 PM EnDat22 Master Details

This section gives a brief overview of how the EnDat interface is implemented on TMS320F28379D devices. By design the TIDM-1008 works with multiple C2000 LaunchPad development kits. This reference design focuses on the F28379D LaunchPad as the main example.

Communication over EnDat interface is achieved primarily by the following components:

- CPU (C28x)
- Configurable logic block (CLB)
- Serial peripheral interface (SPI)
- Device interconnects (XBARs)

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

- Ability to generate two different clocks:
  - to the SPI on chip (on GPI065, looped back from SPI-1-CLK generated on GPIO7)
  - to the encoder (on GPIO6, ENC-1-CLK)

- Ability to adjust the delay between the two clocks
- Identification of the critical delay between the clock edges sent to the encoder and the received data
- Monitoring the data coming from encoder through SPISIMO and poll for start pulse
- Ability to measure the propagation delay at a specific interval as required by the interface
- Ability to configure the block and adjust the propagation delay through software

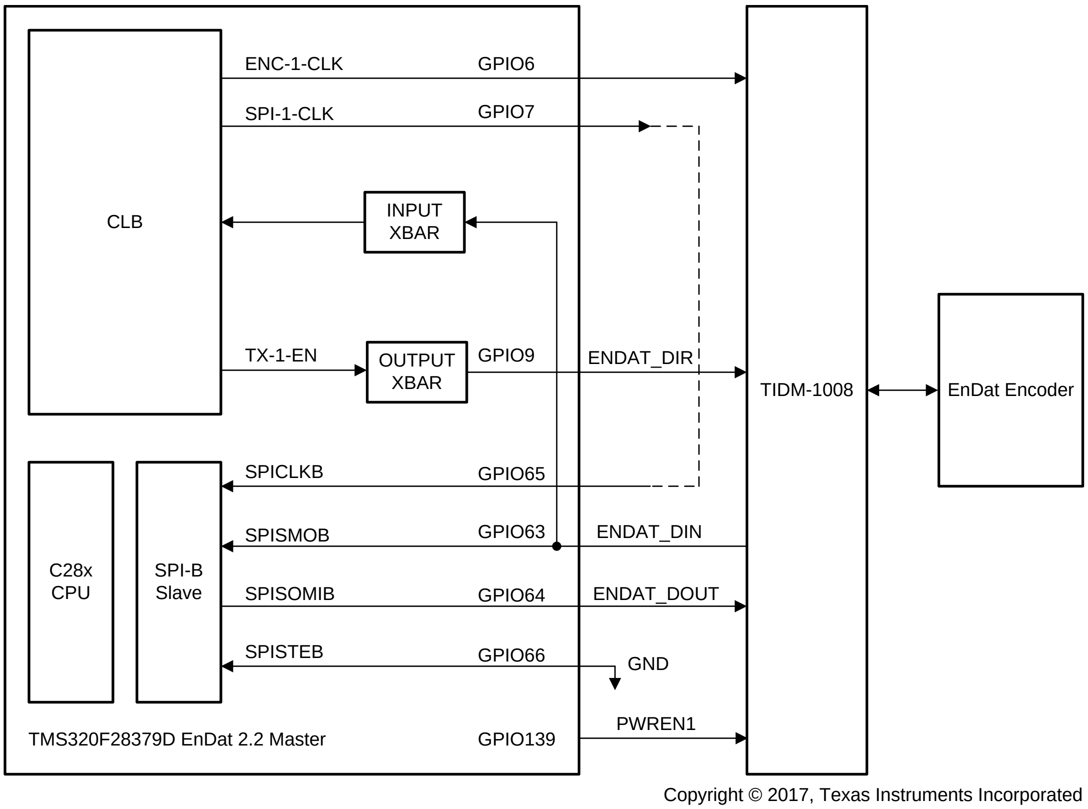


Figure 4. EnDat Implementation Diagram Inside TMS320F28379D

Figure 4 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 configurable logic block to generate clocks for the encoder and SPI.
- Number of clock pulses and edge placement for these two clocks are different and 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 required to change the direction of the data line in order to receive data from the encoder after sending the mode command
- CLB also monitor the SPISIMO signal (as necessary) 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 the encoder.
- CPU configures CLB to generate a predefined number of clock pulses needed for SPI (as per the current command requirements), and continuous clocking for SPI is disabled while waiting for the start pulse from the encoder.
- CLB also provides hooks to perform cable propagation delay compensation using library functions.

- The full MCU resource usage is highlighted in Table 3.

Table 3. TIDM-1008 MCU Resource Requirements

RESOURCE NAME	TYPE	PURPOSE	USAGE RESTRICTIONS
DEDICATED RESOURCES			
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
GPIO9	IO	EnDat direction control for data on LaunchPad	Dedicated IO for EnDat Direction control
GPIO139	IO	For EnDat power control on LaunchPad	Dedicated IO for encoder power enable
CONFIGURABLE RESOURCES			
SPI	Module and IOs	One SPI instance to emulate EnDat interface (SPIB on LaunchPad)	Any instance of SPI can be chosen—module and corresponding IOs will be dedicated for EnDat
SHARED RESOURCES			
CPU and memory	Module	Check CPU and memory utilization for various functions	Application to ensure enough CPU cycles and memory are allocated
Input XBAR	Module, IO	To be connected to SPISIMOB 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 GPIO9 using OUTPUT6 of output	OUTPUT6 is used for EnDat implementation, remaining outputs are available for application use

2.3.3 PM EnDat22 Software Library

The EnDat22 Library provides a host of commands and functions for interfacing C2000 devices with EnDat 2.2 position encoders. This section provides some documentation on the library and describes the commands and functions the library offers. If the latest version of controlSUITE is installed, the library is in the following directory:

```
C:\ti\controlSUITE\development_kits\BOOSTXL_POSMGR
```

Software delivered on controlSuite for TIDM-1008 uses the above hardware resources and the Position Manager BoosterPack is expected to be plugged on Site-2 as shown in Figure 9

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
```

NOTE: The software example included with TIDM-1008 takes care of properly configuring and including the EnDat22 Library in the CCS project. To learn how to use the library for other applications, refer to the *Position Manager EnDat22 Library Module User's Guide* [1].

2.3.3.1 PM EnDat22 Library Commands

Details of the EnDat protocol and commands supported in different modes can be obtained from Heidenhain. Table 4 and Table 5 show the commands supported by the EnDat22 Library.

Table 4. EnDat 2.1 Commands Supported

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

Table 5. EnDat 2.2 Commands Supported

Encoder send position value with additional information	ENCODER_SEND_POSITION_VALUES_WITH_ADDITIONAL_DATA
Encoder send position value and receive selection of memory	ENCODER_SEND_POSITION_VALUES_AND_SELECTION_OF_THE_ area 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

2.3.3.2 PM EnDat22 Library Functions

The EnDat22 Library consists of the following functions that enable the user to interface with EnDat encoders. Table 6 lists the functions existing in the EnDat22 Library and a summary of cycles taken for execution.

Detailed explanations of each function are explained at the end of Section 2.3.3.4.

Table 6. 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	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.	220 (1)	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.	183	Run time
PM_endat22_getCrcNormPM_endat22_getCrcNorm	To get the CRC of each byte, calculate the n-bit CRC of a message buffer by using the lookup table.	95	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 the call 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	Run time
PM_endat22_startOperation	This function initiates the EnDat transfer to be called after the PM_endat22_setupCommand. It performs the EnDat transaction setup by previous commands. Note that the setup and start operation are separate function calls. The user can setup the EnDat transfer and start the actual transfer using this function call, as necessary, at a different time.	46	Run time
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	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 must be called after every system reset. No EnDat transactions will be performed until the setup peripheral function is called.	8822	Initialization time
PM_endat22_setFreq	This function sets the EnDat clock frequency. EnDat transfers, typically start low frequency during initialization and switch to higher frequency during on runtime.	220	Initialization 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 Heidenhain.	21	Initialization time

2.3.3.3 PM EnDat22 Library Data Structures

The PM EnDat22 Library defines the EnDat data structure handle as:

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;
```

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	Maximum 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	Maximum 8 bits only
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.	Maximum 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
0 or 1	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	Maximum value 8

2.3.3.4 PM EnDat22 Library Function Details

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

Directions:

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:

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

Directions:

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:

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 CRC values for the ENCODER_SEND_TEST_VALUES command. This is an 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

Directions:

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	Typically16-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:

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_AREA
- 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 1as 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_AREA
- ENCODER_SEND_POSITION_VALUES_AND_SELECTION_OF_THE_MEMORY_AREA
- 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

Directions:

Setup an 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	Typically18/6-bit Data or Address depending on the mode command
data2	Typically18/6-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:

- SELECTION_OF_MEMORY_AREA In order to send or read parameters, the memory area must first be selected. This selection 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. The command 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. The command 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, acceleration values, and so forth. 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_A REA 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, // MRScode
    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 the user wants 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

Directions:

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 endat22Data.position_clocks. Encoder transmits the position value with LSB first. The values stored in endat22Data.position_hi and endat22Data.position_lo; however, the values 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 a few examples of how the PM_endat22_setupData function is used with various mode commands. For further details, see the Heidenhain documentation.

- 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 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. The command 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. The command is written to 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 = 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 = Any
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, acceleration values, and so on. 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_A REA 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 the user wants 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

Directions:

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 setup earlier by PM_endat22_setupCommand. Note that the setup up and start operation are separate function calls. User can setup the EnDat transfer when required and start the actual transfer using this function call, as necessary, at a different time.

Definition:

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

Parameters:

INPUT:	—
—	None
RETURN :	—
—	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. This flag can be polled to know if the data from the encoder is successfully received after the PM_endat22_startOperation function call.

PM_endat22_setupPeriph

Directions:

Setup for SPI, CLB, and other interconnect XBARs for EnDat are performed with this function during system initialization. This function must 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:

INPUT:	—
—	None
RETURN:	—
—	None

Usage:

Example code:

```
endat22Data.spi = &amp;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. This flag can be polled to know if the data from the encoder is successfully received after the PM_endat22_startOperation function call.

PM_endat22_setFreq

Directions:

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 (approximately 200 KHz)
- Used during application (approximately 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:

INPUT:	—
Freq_us	A clock divider for the system clock sets EnDat Clock Frequency = SYSCLK/(4* Freq_us)
RETURN:	—
—	None

Usage:

Example code:

```
endat22Data.spi = &amp;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. This flag can be polled to know if the data from the encoder is successfully received after the PM_endat22_startOperation function call.

PM_endat22_getDelayCompVal

Directions:

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. 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:

INPUT:	—
—	None
RETURN:	—
delay	Delay value to be set for endat22Data.delay_comp parameter

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 must be updated into endat22Data.delay_comp field before switching to higher-frequency operation. For delay compensation, see the TI provided examples on usage of this function.

3 Hardware, Software, Testing Requirements, and Test Results

3.1 Required Hardware and Software

3.1.1 Hardware

This section describes the hardware specifics of TIDM-1008 and how to get started with the EnDat22 Library in CCS.

To experiment with TIDM-1008, the following components are required:

- TIDM-1008 EVB
- External 5-V DC power supply (refer to key system specifications in Table 1)
- F28379D LaunchPad development kit (LAUNCHXL-F28379D)
- USB-B to A cable
- EnDat 2.2 encoder from Heidenhain (for example, ROC425 or ROC437)
- EnDat22 8-pin cable from Heidenhain—length as required by the application (maximum 100m)
- Custom adapter to connect HeidenhainCircular 8-position female terminated cable to wire leads adapter
- PC with CCS (CCSv6 or greater) installed

3.1.1.1 TIDM-1008 Jumper Configuration

Figure 5 shows the jumper configuration for TIDM-1008.

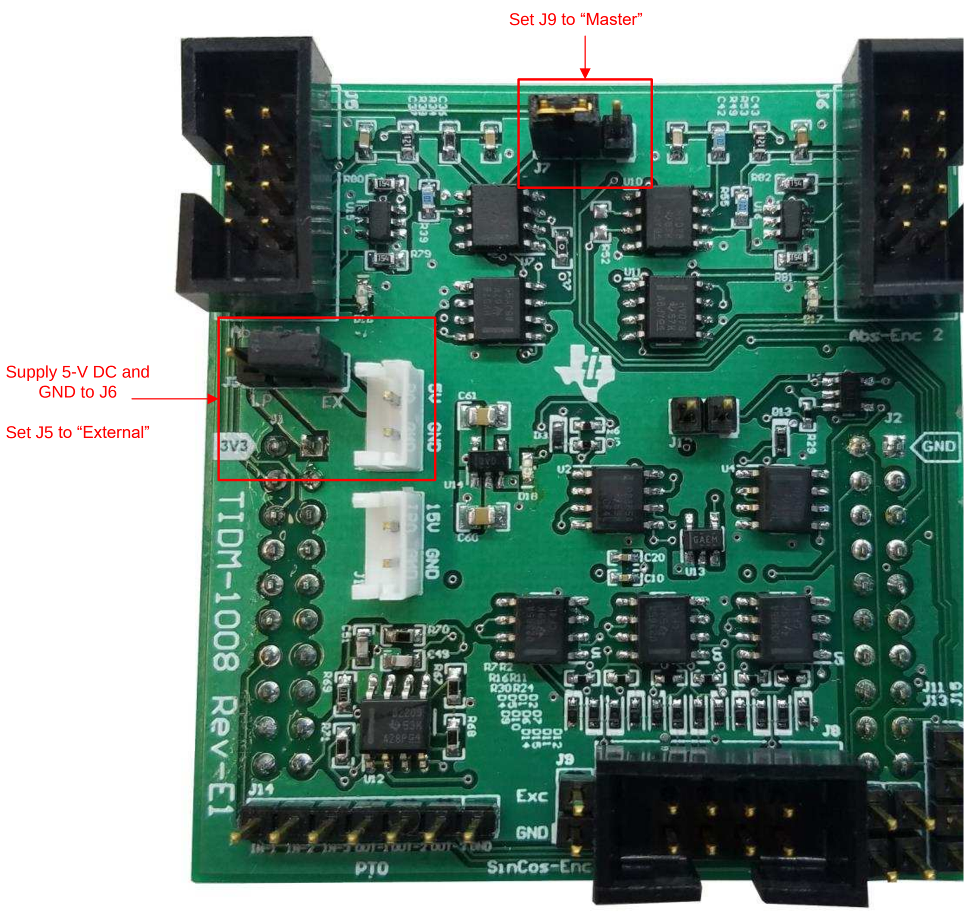


Figure 5. TIDM-1008 Board Jumper Configuration

Table 7 lists the jumper configuration for the TIDM-1008 board.

Table 7. TIDM-1008 Board Jumper Details

JUMPER	FUNCTION	POSITION
J5	TIDM-1008 5-V power plane source selection	*External* (1)
J9	Abs-Enc-1 master slave mode selection	*Master* (2)
J11	Sine-Cosine encoder A signal enable	Open
J12	Sine-Cosine encoder B signal enable	Open[3]
J13	Sine-Cosine encoder index signal enable	Open[3]

(1) This configuration requires providing an external power source to J6 as shown in Figure 5.
(2) This jumper is for a future reference design.

3.1.2 Software

This section describes how to configure the software environment for the F28379D LaunchPad.

3.1.2.1 Installing Code Composer Studio™ and controlSUITE™

1. Install CCS v6.x or later if it is not already on the PC.
2. Go to http://www.ti.com/controlsuite and run the controlSUITE™ installer. Allow the installer to download and update any automatically checked software for C2000.
3. After installation, refer to Section 2.3.3 for more information on the EnDat22 Library.

3.1.2.2 Configure CCS for F28379D LaunchPad™

1. Open CCS. Note that this document assumes that version 6 or later is used.
2. Once CCS opens, the workspace launcher may appear that would ask to select a workspace location. Note that workspace is a location on the hard drive where all the user settings for the IDE (which projects are open), what configuration is selected, and so forth are saved. This workspace can be anywhere on the disk, the location mentioned below is just for reference. Note that if this is not the first-time running CCS, the dialog below may not appear.
   a. Click the *Browse…* button.
   b. Create the following path by making new folders as necessary:
   ```
   C:\c2000 projects\CCSv6_workspaces\PM_endat22_eval_workspace
   ```
   c. Uncheck the box that says *Use this as the default and do not ask again*.
   d. Click *OK*.
3. A *Getting Started* tab will open with links to various tasks from creating a new project, importing an existing project, and watching a tutorial on CCS. User can close the *Getting Started* Tab, and go to next step.
4. CCS is configured in order to know which MCU the program will be connecting to. This configuration is done by setting up the *Target Configuration*.
5. A new configuration file can be set by clicking *View → Target Configuration*. This procedure will open the Target Configuration window. In this window, click on the  icon. Give a name to the new configuration file depending on the target device. If *Use shared location* checkbox is checked then this configuration file can be stored in a common location by CCS for use by other projects as well. Then, click *Finish*.

6. This step should open up a new tab as shown in Figure 6. Select and enter the following options:
   a. Connection— *Texas Instruments XDS100v2 USB Emulator* or *Texas Instruments XDS100v2 USB Debug Probe*
   b. Device—the C2000 MCU on the control card, TMS320F28379D, for example
   c. Click *Save* and close.

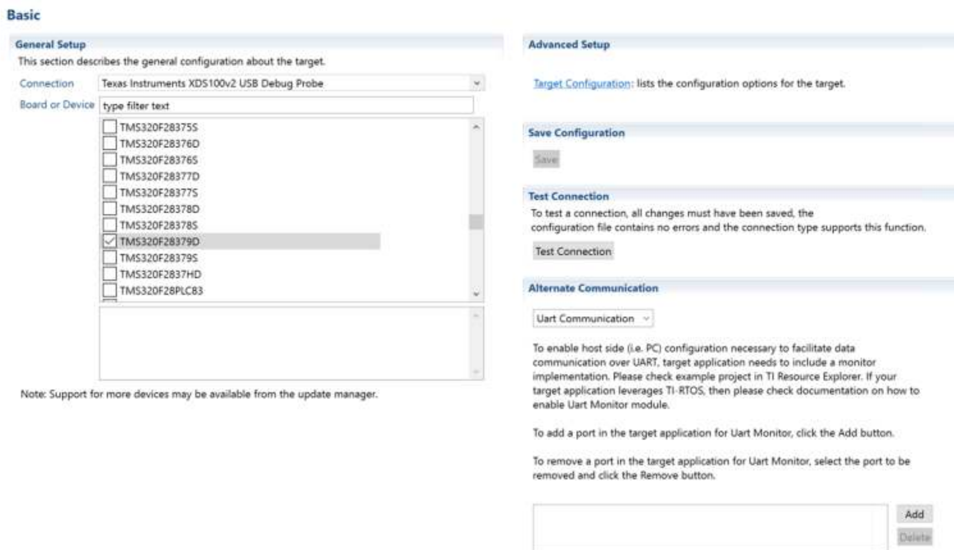


Figure 6. Configuring a New Target Configuration

7. Click *View → Target Configurations*. In the *User Defined* section, find the file that was created in steps 6 and 7. Right-click on this file, and select *Set as Default*. To use the configuration file supplied with the project, click *View → Target Configurations*, expand *Projects → PM_endat22_BasicPosAcc_DelComp*, and right-click on the *xds100v2_F2837x.ccxml* and *Set as Default* files. This tab also allows the user to reuse the existing target configurations and links them to specific projects.

3.1.2.3 Configuring the TIDM-1008 Example Project

1. Add the PM EnDat22 evaluation example project into the current workspace by clicking *Project → Import CCS Project*.
   a. Select the project by browsing to:
      `C:\ti\controlSUITE\development_kits\BOOSTXL_POSMGR`
   b. Something similar to Figure 7 import will appear, and click *Finish*.

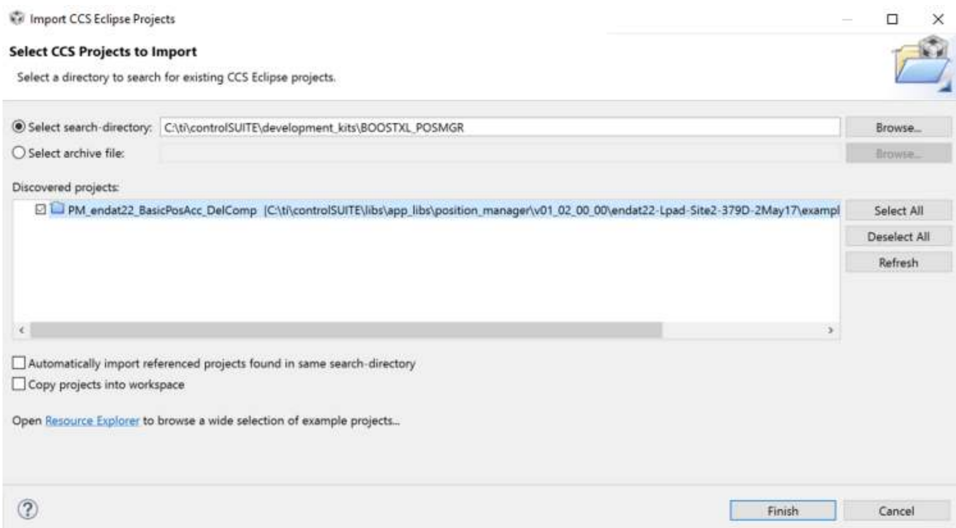


Figure 7. Adding PM EndDat22 Example Project to Workspace

NOTE: By default, CCS will not copy the project into the workspace. Any changes made to files within CCS will thus be reflected in the files stored in the controlSUITE installation. If desired to preserve the original files stored in controlSUITE, check the box *Copy projects into workspace*, as seen in Figure 7.

---

2. Assuming this is the first time using CCS, the xds100v2-F2837x should have been set as the default target configuration. Verify this by viewing the xds100v2-f2837x.ccxml file in the expanded project structure and an *Active* or *Default* status written next to file. By going to *View → Target Configurations*, the user can edit existing target configurations or change the default or active configuration. The user can also link a target configuration to a project in the workspace by right-clicking on the target configuration name and selecting *Link to Project*.

3. The project can be configured to create code and run in either Flash or RAM. Either of the two can be selected, however, RAM configuration is used most of the time for lab experiments and Flash configuration for production. As shown in Figure 8, right-click on an individual project and select *Active Build Configuration → CPU1_RAM configuration*.

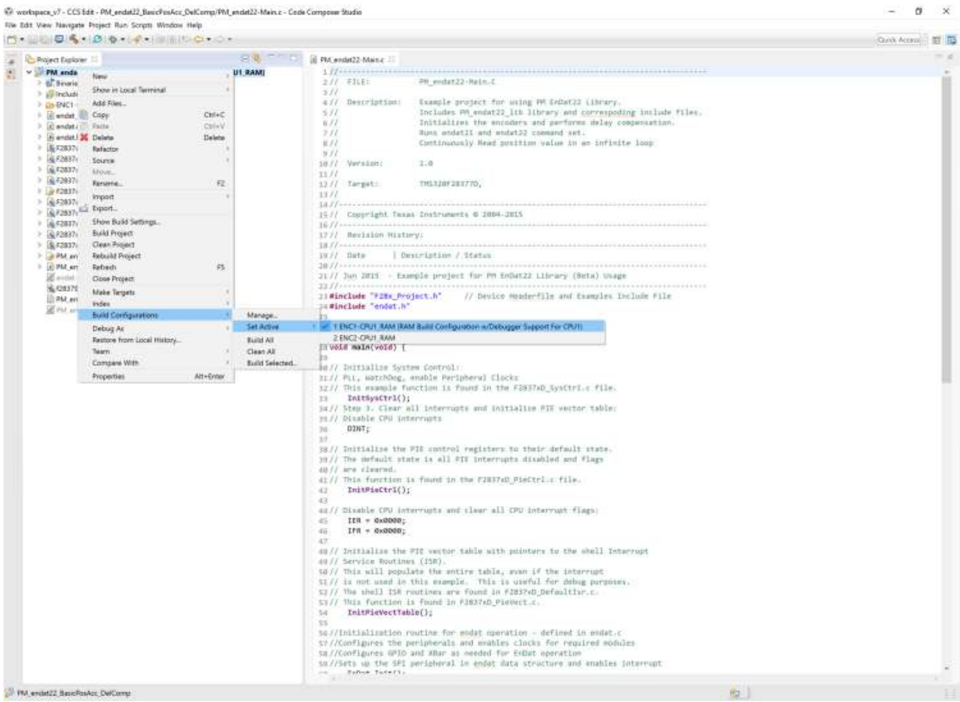


Figure 8. Selecting F2837x_RAM Configuration

3.2 Testing and Results

3.2.1 Test Setup

3.2.1.1 Hardware Configuration

1. Ensure that the jumper configuration of TIDM-1008 is as described in Table 7.
2. Connect TIDM-1008 to the LaunchPad using the BoosterPack connector (J1-J3 and J4-J2). Make sure TIDM-1008 is connected to site two of the LaunchPad as shown in Figure 9.

Figure 9. TIDM-1008 Board Connected to Site Two of LaunchPad™

3. Connect the USB cable to the LaunchPad.
4. Connection to the encoder:
   a. Prepare an adapter to connect the Heidenhain cable to the IDDK EnDat interface using the circular 8-position female to wire leads adapter (refer to the BOM for the header used for the encoder connector—J7). Refer to Figure 10.

Figure 10. Adapter to Connect Heidenhain Cable to TIDM-1008 Board

   b. Insert the header of the adaptor created in the previous step to connect to Abs-Enc-1 (J7). The female end of the Heidenhain cable connect to the encoder. The pinout of J7 is shown Figure 11.

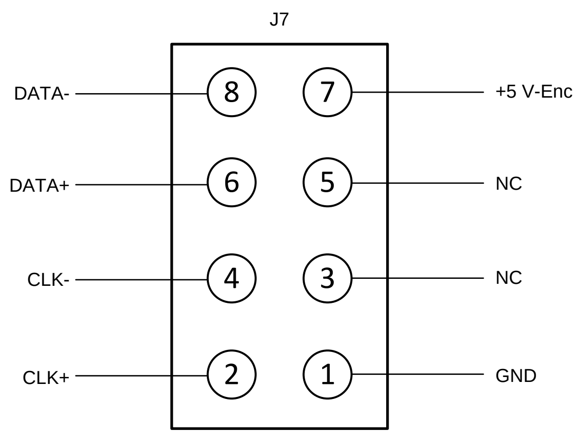


Figure 11. Abs-Enc-1 (J7) Pinout on TIDM-1008 Board

5. Supply 5-V DC and GND to J6 as shown in Figure 5. The board should now look like Figure 12. LED D18 should light, which shows that the board has power.

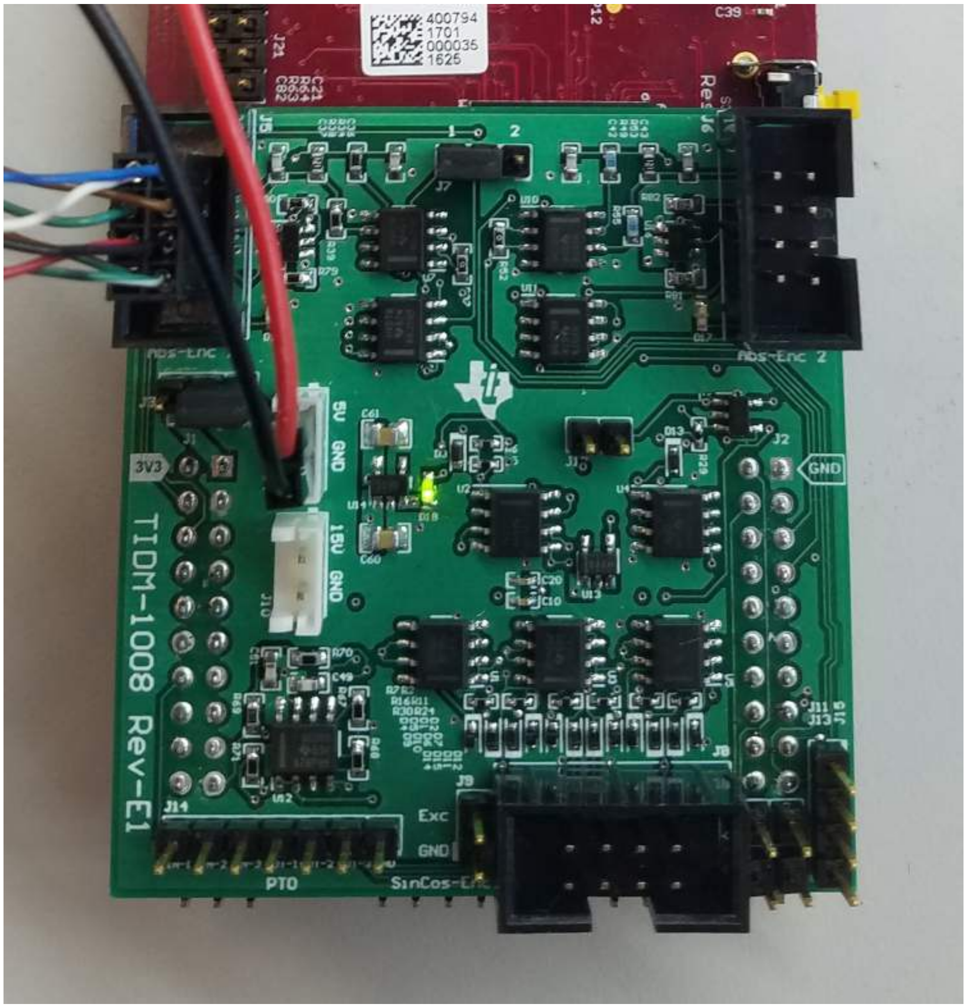


Figure 12. TIDM-1008 Board Powered on and Connected to Heidenehain Encoder

3.2.2 Test Results

This section will describe how to run the software example and detail the results in CCS. The software flow diagram for this project is shown in Figure 13.

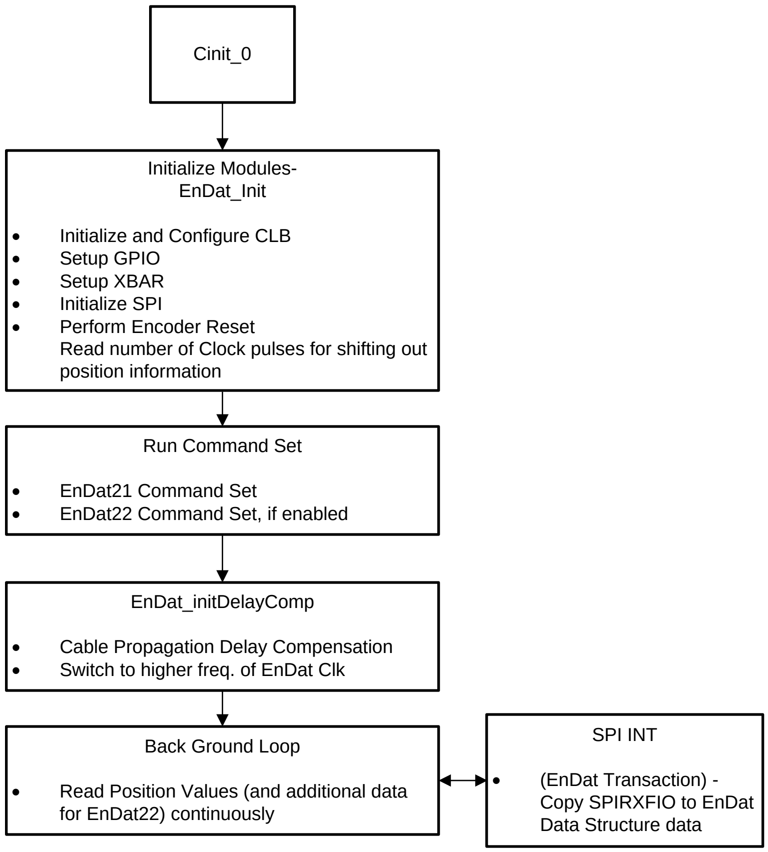


Figure 13. Software Flow Diagram for Example Project PM_endat22_BasicPosAcc_DelComp

3.2.2.1 Build and Load Project

1. Complete the software setup as described in Section 3.1.2 .
2. Open the endat.h file and make sure that ENDAT_RUNTIME_FREQ_DIVIDER and ENDAT_INIT_FREQ_DIVIDER are set as necessary. Save this file. For more information refer to Section 4.2 of *Position Manager EnDat22 Library Module*.

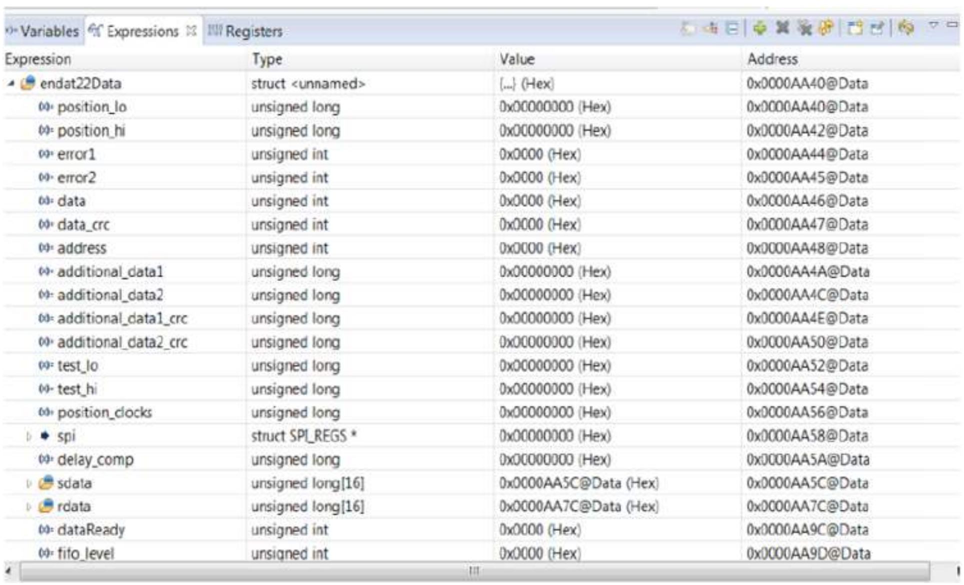
   Right-click on the project name and select *Rebuild Project*, and watch the console window. Any errors in the project will be displayed in the console window.
3. Right-click on the project name and select *Rebuild Project*,and watch the console window.
4. Upon successful completion of the build, click the Debug button on the top-left of the CCS window. If a window appears prompting to select a CPU, make sure CPU1 is selected, and click *OK*.
5. The IDE will automatically connect to the target, load the output file into the device, and change to the debug perspective.
6. Click the *Tools → Debugger Options → Program / Memory Load Options*. The debugger can be enabled to reset the processor each time the debugtger reloads the program by checking *Reset the target on program load or restart*, and click *Remember My Settings* to make this setting permanent.
7. Click on the *Enable silicon real-time mode* button , which autoselects the *Enable polite real-time mode* button . This button allows the user to edit and view variables in real time.
8. Select *YES* to enable debug events, if a message box appears. This action will set bit 1 (DGBM bit) of the status register 1 (ST1) to a 0. The DGBM is the debug enable mask bit. When the DGBM bit is set

to 0, memory and register values can be passed to the host processor for updating the debugger windows.

NOTE: Do not reset the CPU without first disabling these real-time options

3.2.2.2 Using Watch Window

1. The best way to import all of the useful variables in the example is by right-clicking in the expressions window and then clicking *Import*. Browse to the .txt file containing these variables.
2. For this project, browse to the root directory and select *Pm_endat22_BasicPosAcc_DelComp_VAR.txt*, and click *OK* to import the variables as shown in Figure 14.
3. Click *View* → *Expressions* on the menu bar to open a watch window to view the variables being used in the project. Additional variables can be added to the watch window as shown if desired.



Expression	Type	Value	Address
endat22Data	struct <unnamed>	{...} (Hex)	0x0000AA40@Data
position_lo	unsigned long	0x00000000 (Hex)	0x0000AA40@Data
position_hi	unsigned long	0x00000000 (Hex)	0x0000AA42@Data
error1	unsigned int	0x0000 (Hex)	0x0000AA44@Data
error2	unsigned int	0x0000 (Hex)	0x0000AA45@Data
data	unsigned int	0x0000 (Hex)	0x0000AA46@Data
data_crc	unsigned int	0x0000 (Hex)	0x0000AA47@Data
address	unsigned int	0x0000 (Hex)	0x0000AA48@Data
additional_data1	unsigned long	0x00000000 (Hex)	0x0000AA4A@Data
additional_data2	unsigned long	0x00000000 (Hex)	0x0000AA4C@Data
additional_data1_crc	unsigned long	0x00000000 (Hex)	0x0000AA4E@Data
additional_data2_crc	unsigned long	0x00000000 (Hex)	0x0000AA50@Data
test_lo	unsigned long	0x00000000 (Hex)	0x0000AA52@Data
test_hi	unsigned long	0x00000000 (Hex)	0x0000AA54@Data
position_clocks	unsigned long	0x00000000 (Hex)	0x0000AA56@Data
spi	struct SPI_REGS *	0x00000000 (Hex)	0x0000AA58@Data
delay_comp	unsigned long	0x00000000 (Hex)	0x0000AA5A@Data
sdata	unsigned long[16]	0x0000AA5C@Data (Hex)	0x0000AA5C@Data
rdata	unsigned long[16]	0x0000AA7C@Data (Hex)	0x0000AA7C@Data
dataReady	unsigned int	0x0000 (Hex)	0x0000AA9C@Data
fifo_level	unsigned int	0x0000 (Hex)	0x0000AA9D@Data

Figure 14. Properly Configured Watch Window

4. Each variable uses the number format that the variable is associated with during declaration. The variable can be changed to another number format right-clicking on it. Figure 14 shows a typical expressions window.
5. Click on the *Continuous Refresh* button in the watch window. This enables the window to run with real-time mode. By clicking the down arrow in this watch window, the user can select *Customize Continuous Refresh Interval* and edit the refresh rate of the watch window.

NOTE: Choosing too fast of an interval may adversely effect performance.

3.2.2.3 Run the Example Code

1. Run the code by pressing the *Run*button in the Debug tab.
2. The project should run and the values in the watch window should continuously update. If the encoder is not mounted on a spinning motor, the user can manually rotate the encoder shaft and observe the variables change accordingly:
   a. As the encoder sends the position information, observe the same in the watch window as endat22Data.position_hi, endat22Data.position_lo variables.
   b. The variable endat22Data.position_hi would only change if the encoder sends more than 32 bits of position information.

3. Once complete, reset the processor (*Run → Reset → CPU Reset*) and terminate the debug session by clicking (*Run → Terminate*) . This action halts the program and disconnects CCS from the MCU.

3.2.2.4 Cable Length Validation

Table 8 lists tests with various types of encoders; cable length tests are performed at Heidenhain Labs. Tests include basic command set exercising and reading position values with additional data if applicable.

Table 8. Cable Length Test Report

ENCODER NAME	TYPE	RESOLUTION (BITS)	CABLE LENGTH[1] (m)	MAX EnDat CLOCK (MHz)	TEST RESULTS
ROC425	Rotary	25	70	8	Pass
LC415	Linear	35	70	8	Pass
RCN8310	Rotary	29	70	8	Pass
ROQ437	Multi-turn	25, 12 (Turns)	70	8	Pass
LIC211	Linear	32	70	8	Pass
ROC413	Rotary	13	70	8	Pass

(1) 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.

4 Design Files

To download the design files, see the product page TIDM-1008 .

5 Software Files

Refer to Section 3.1.2.3.

6 Related Documentation

1. Texas Instruments, *Position Manager EnDat22 Library Module* , User's Guide (SPRUI35)
2. EnDat 2.2 – Bidirectional Interface for Position Encoders, Heidenhain EnDat 2.2 Documentation
3. Texas Instruments, C2000 DesignDRIVE, Software for Industrial Drives and Motor Control
4. Texas Instruments, *C2000 Position Manager SinCos Library* , User's Guide (SPRUI54)

6.1 Trademarks

C2000, BoosterPack, LaunchPad, E2E, Delfino, controlSUITE are trademarks of Texas Instruments Incorporated.
HEIDENHAIN is a trademark of DR. JOHANNES HEIDENHAIN GmbH.
All other trademarks are the property of their respective owners.

7 Terminology

TYPE	DESCRIPTION
C28x	Refers to devices with the C28x CPU core
CLB	Configurable logic block
Position Manager BoosterPack	Future EVM for interfacing with various position encoders. The TIDM-1008 board is identical to the Position Manager BoosterPack EVM (refer to Section 2.3.1)
CRC	Cyclic redundancy check
EnDat22	2.2 version of EnDat position encoder interface protocol by Heidenhain
EnDat21	2.1 version of EnDat position encoder interface protocol by Heidenhain
PM	Position Manager—foundational hardware and software on C28x devices for position encoder interfaces
PM_endat22	Prefix used for all the library functions
SPI	Serial peripheral interface

8 About the Author

SUBRAHMANYA BHARATHI AKONDY has worked on architecture definition and design of several C2000 MCU products and control peripherals. His interests include MCU architecture, applications, and design aspects.

WILLIAM STEVERS has worked on the Industrial Drives Systems team of the C2000 MCU group for several summers—first in Application Engineering role and more recently in a Systems Engineering role. William is currently completing a Master's degree in Electrical Engineering at Michigan State University.

Revision History

NOTE: Page numbers for previous revisions may differ from page numbers in the current version.

Changes from Original (September 2017) to A Revision	**Page**
• Changed part number TLV70233 to TLV702	1
• Changed part number TPS22918 to TPS22918-Q1	1

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/sampterms.htm).