TI Designs：TIDM-1010
适用于 C2000™ MCU 的 BiSS-C 绝对编码器主接口参考设计

说明

C2000™微控制器 (MCU) 位置管理器技术可提供集成式解决方案来连接最常用的数字和模拟位置传感器，而无需外部现场可编程门阵列 (FPGA) 或专用集成电路 (ASIC)。位置管理器 BoosterPack™是一种用于评估不同编码器接口的灵活且具有成本效益的平台，旨在与多个 C2000 MCU LaunchPad™开发套件搭配使用。该参考设计的软件专门用于实施 BiSS-C（位置编码器的数字双向接口）。该参考设计中包括高度优化且易于使用的软件库和示例，支持使用位置管理器 BoosterPack 来运行 BiSS-C 位置编码器。

资源

TIDM-1010	设计文件夹
LAUNCHXL-F28379D	工具文件夹
SN65HVD78	产品文件夹
TLV702	产品文件夹
TPS22918-Q1	产品文件夹

TI E2E™ Community

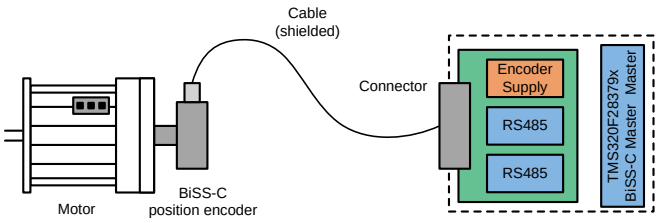
咨询我们的 E2E™ 专家

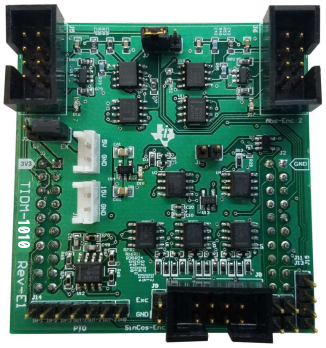
特性

- 用于位置编码器接口的灵活低电压 BoosterPack 评估平台
- 用于 BiSS-C 的集成 MCU 解决方案，无需额外的 FPGA
- 可通过库提供的驱动程序函数和数据结构轻松使用连接 BiSS-C 的诸项命令
- 用于解压缩接收数据的库支持和优化的循环冗余校验 (CRC) 算法
- 支持高达 10MHz 的时钟频率（在长达 100m 的电缆长度运行条件下进行了验证）
- 包括展示 BiSS-C 软件库的评估软件示例

应用

- 工业
- 电机驱动器





该 TI 参考设计末尾的重要声明表述了授权使用、知识产权问题和其他重要的免责声明和信息。

1 System Description

Industrial drives, like servo drives, require accurate, highly-reliable, and low-latency position feedback. BiSS is an open-source digital interface for sensors and actuators. BiSS-C is a pure serial, digital interface based on the RS-485 standard. The interface transmits position values or additional physical quantities from the encoder (BiSS-C Slave) to the MCU (BiSS-C Master). The interface also allows reading and writing of the internal memory of the encoder. The transmitted data types include: absolute position, turns, temperature, parameters, diagnostics, and so on. Mode commands that the subsequent electronics, often referred to as the BiSS-C master, send to the encoder select the transmitted data types. The TIDM-1010 device acts as an BiSS-C master and provides the subsequent electronics to interface an BiSS-C encoder with the F28379D LaunchPad. 图 1 shows the major hardware blocks used in this design.

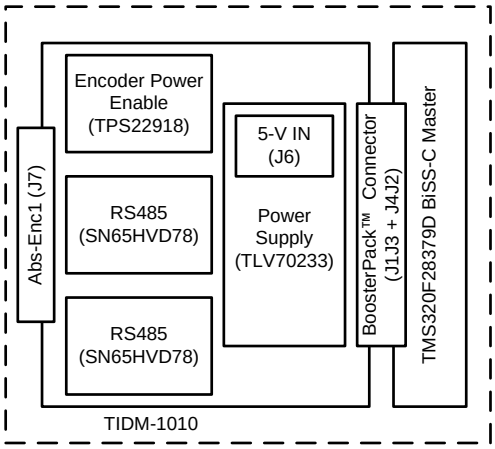


图 1. TIDM-1010 Hardware Blocks and Connectors

The position encoder with BiSS-C connects to the TIDM-1010 device through a 6-pin interface. A point-to-point configuration is typically used with the BiSS position or rotary encoders, as shown in 图 2. In the point-to-point configuration, only one device with one or more sensors is operated on the master. The MO line is eliminated, and the SL line is routed back directly from the slave. PM_bissc library only supports this configuration.

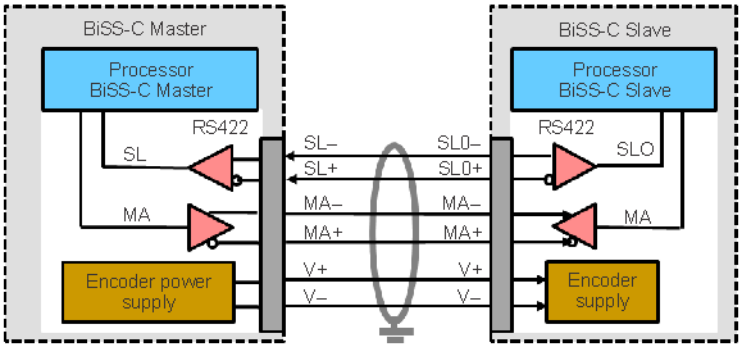


图 2. BiSS-C Point-to-Point Structure

In this configuration, two signals, MA (clock) and SL (data), are used for communication. These signals are differential in nature, resulting in the 4 wires, MA+, MA– and SL+, SL–, in a unidirectional, full-duplex mode. Two additional wires, V+ and V–, are for the encoder power supply, where V– is typically GND. The MA-clock frequency is variable and depends on the cable length (see 节 3.2.2.4). Depending on the encoder and the encoder cable, the maximum cable length or the maximum achievable clock frequency can vary. Encoder manufacturers define these limits in their data sheets and recommend an appropriate cable for use with their encoders, because the quality of the cable has an impact on the communication performance. More details of the protocol and point-to-point configuration are at BiSS-interface.com.

The C2000 Position Manager BissC (PM_bissc) library from Texas Instruments provides support for implementing the BiSS-C interface in subsequent electronics. The library comprises the software portion of the TIDM-1010 device. The BissC Library features an integrated-MCU solution for the BiSS-C interface that meets the digital-interface protocol requirements. The library can support up to a 10-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 allow 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.

Note these several key concepts while using the BiSS-C Library. The library only supports the basic-interface drivers for commands defined in the BiSS-C specification. All the higher-level application software must be developed by users using the basic interface provided by this library. Clock frequency for the BiSS-C clock is limited to a maximum of up to 10 MHz. This limitation applies irrespective of the cable length and encoder type. For any additional functionality or encoder use not specified in this reference design, contact the TI support team or refer to the TI E2E™ community.

1.1 Key System Specifications

表 1. Key System Specifications

PARAMETER	SPECIFICATIONS	DETAILS
Input voltage	5 V(1)	节 3.2.1.1
Output voltage (encoder)	5 V	节 3.2.1.1
Protocol supported	BiSS-C	BiSS
Frequency (encoder interface)	Approximately 10 MHz	节 2.3.2
Encoder bits	BiSS-C protocol standard	BiSS
CPU cycles	—	节 2.3.3.1
Maximum cable length (tested)	100 m	—

(1) The time of the encoder connected to the TIDM-1010 device determines the current limit of this supply. TI recommends a generic, bench-top, adjustable, power supply with an adjustable current limit.

2 System Overview

2.1 Block Diagram

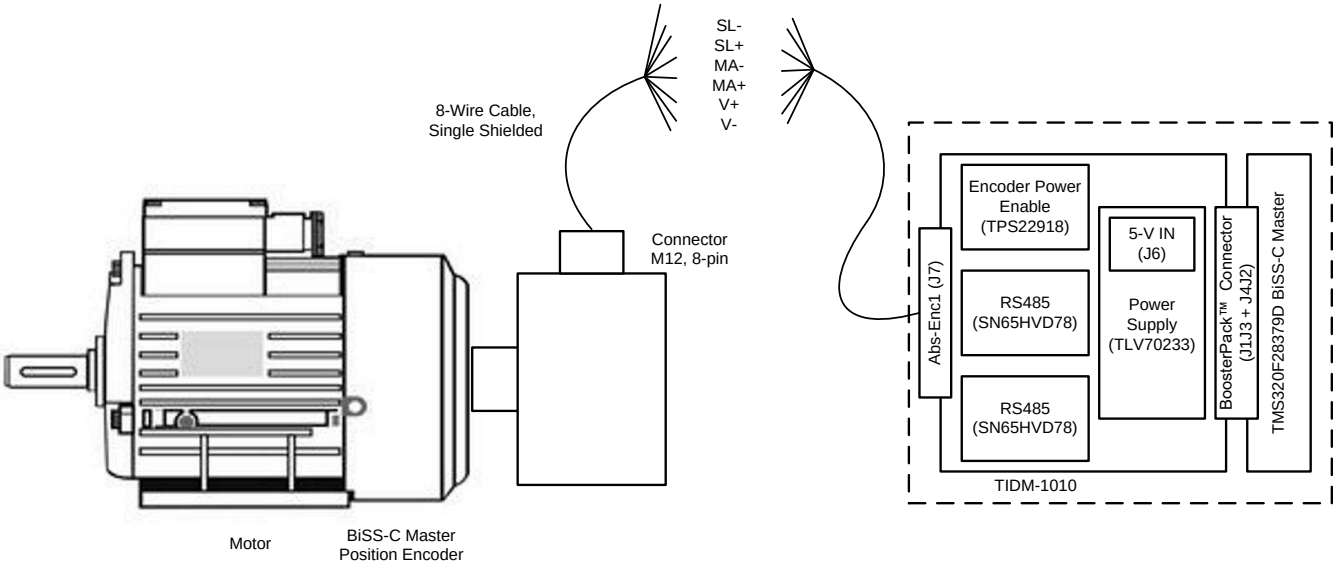


图 3. TIDM-1010 System Block Diagram

2.2 Highlighted Products

2.2.1 LAUNCHXL-F28379D

This development kit is based on the Delfino™ TMS320F28379D MCU, which provides 800 MIPS 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 device 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 internally connected 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 can interface directly with low-voltage control signals.

2.3 Design Considerations

2.3.1 TIDM-1010 Board Implementation

The TIDM-1010 board is identical to the Position Manager BoosterPack (BOOSTXL-POSMGR), which means the TIDM-1010 board can interface with several other position encoder types. The board is fully populated by default for future compatibility. This reference design focuses on the BiSS-C, and the hardware blocks not mentioned in this document can be ignored. Software support for the other types of position-encoder interfaces will be the subject of future reference designs. 表 2 lists the connectors on the TIDM-1010 and BOOSTXL-POSMGR and their functions.

表 2. TIDM-1010 Board and BOOSTXL-POSMGR Connectors

CONNECTOR	DESCRIPTION	RELEVANT TI DESIGNS AND HARDWARE
Abs-Enc-1 (J7)	BiSS-C and other absolute encoders	TIDM-1010, BOOSTXL-POSMGR
Abs-Enc-2 (J8)	BiSS-C and other absolute encoders	Future TID and BOOSTXL-POSMGR
Abs-Enc-2 Breakout (J10)	Allows two absolute encoders at site two using jumpers	Future TID and BOOSTXL-POSMGR
SinCos (J14)	SinCos encoder	Future TID and BOOSTXL-POSMGR
Resolver (J14 and J15)	Resolver interface with 15-V excitation circuitry	Future TID and BOOSTXL-POSMGR

表 2. TIDM-1010 Board and BOOSTXL-POSMGR Connectors (continued)

CONNECTOR	DESCRIPTION	RELEVANT TI DESIGNS AND HARDWARE
PTO (J17)	Pulse-train output	Future TID and BOOSTXL-POSMGR
J1, J3 and 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 and BOOSTXL-POSMGR

图 4 describes the encoder support at each site of the LaunchPad.

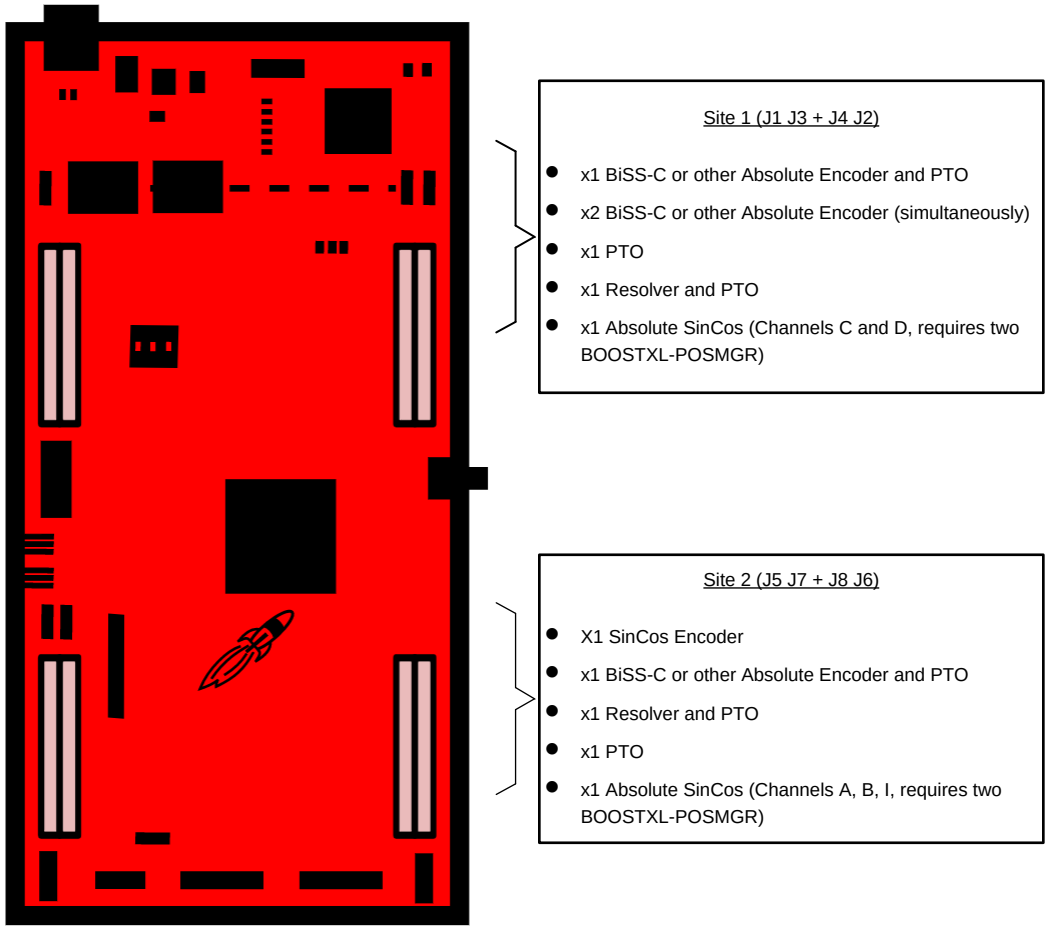


图 4. TIDM-1010 Board and BOOSTXL-POSMGR Encoder Support

2.3.2 PM Biss-C Master Implementation Details

This section gives a brief overview of how the BiSS-C interface is implemented on TMS320F28379D devices. By design the TIDM-1010 device works with multiple C2000 LaunchPad development kits. This reference design focuses on the F28379D LaunchPad as the main example.

Communication over the BiSS-C interface is primarily achieved by the following components:

- CPU (C28x)
- Configurable logic block (CLB)

- Serial peripheral interface (SPI)
- Device interconnects (XBARs)

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

- Generate two different clocks to the:
  - SPI on-chip (on GPIO26, looped back from SPI-1-CLK generated on GPIO7)
  - BISS-C Encoder (on GPIO6, BISSC-1-CLK)
- Adjust the delay between the two clocks
- Identify the critical delay between the clock edges sent to the encoder and the received data
- Monitor the data coming from the encoder through SPISIMO and poll for the start pulse
- Measure the propagation delay at a specific interval, as required by the interface
- Configure the block and adjust the propagation delay through software

图 5 shows how a BiSS-C transaction works in the system.

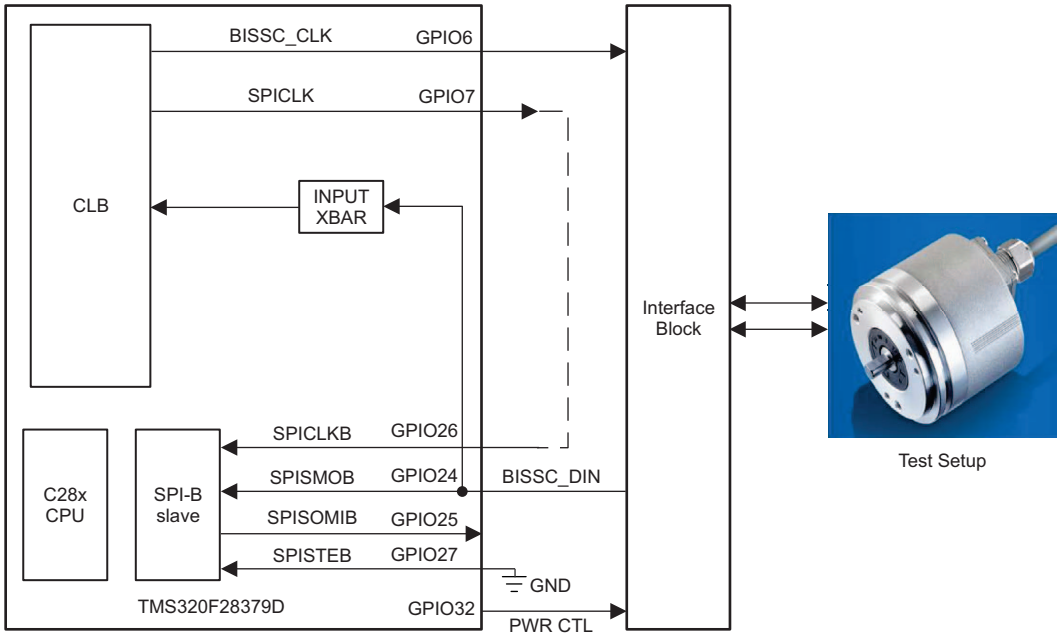


图 5. BiSS-C Implementation Diagram Inside TMS320F28379D Device

For every BiSS-C transaction initiated using the PM BissC Library command the:

- CPU configures the SPITXFIFO signal with the command and other data required for transmission to the encoder, as per the specific requirements of the current BiSS-C command.
- CPU sets up a 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 the CPU software for the current BiSS-C command.
- CLB also generates the direction-control signal for the data line transceiver. This signal is required to change the direction of the data line, to receive data from the encoder after sending the mode command
- CLB monitors the SPISIMO signal (as necessary) for detecting the start pulse and accordingly adjusts the phase of the receive clock.

- 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 provides hooks to perform cable propagation-delay compensation using library functions.

表 3 lists the full, MCU resource uses.

表 3. TIDM-1010 MCU Resource Requirements

RESOURCE NAME	TYPE	PURPOSE	USE RESTRICTIONS
DEDICATED RESOURCES			
GPIO6	I/O	BiSS-C clock from master to encoder	I/O dedicated for BiSS-C
GPIO7	I/O	SPI clock generated by MCU	I/O dedicated for BiSS-C
EPWM4	I/O	Internal for clock generation	EPWM4 dedicated for BiSS-C
GPIO32	I/O	BiSS-C power control on LaunchPad	Dedicated I/O for encoder power enable
CONFIGURABLE RESOURCES			
SPI	Module and I/Os	One SPI instance to emulate the BiSS-C interface (SPIB on LaunchPad)	Any instance of SPI can be chosen — module and corresponding I/Os are dedicated for BiSS-C
SHARED RESOURCES			
CPU and memory	Module	Check CPU and memory use for various functions	Application to ensure enough CPU cycles and memory are allocated
Input XBAR	Module, I/O	Connected to SPISIMOB of the corresponding SPI instance dedicated for BiSS-C	INPUTXBAR1 is used for BiSS-C implementation, remaining inputs are available for application use

2.3.3 PM BissC Software Library

The PM BissC Software Library provides a host of commands and functions for interfacing C2000 devices with BiSS-C position encoders. This section provides some documentation on the library and describes the commands and functions it 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 the TIDM-1010 device uses the previously listed hardware resources, and the Position Manager BoosterPack is expected to be plugged on Site-2, as shown in 图 10.

The following subdirectory structure is used:

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

注: The software example included with the TIDM-1010 device handles properly configuring and including the BiSS-C Library in the Code Composer Studio™ (CCS) project. To learn how to use the library for other applications, see the *Position Manager BiSS-C Library Module User's Guide*.

2.3.3.1 PM BissC Library Functions

The BissC Library consists of the following functions, which enable the user to interface with BISS-C encoders. 表 4 lists the functions existing in the PM_bissc library and a summary of cycles taken for execution.

节 2.3.3.3 has detailed explanations of each function at the end.

表 4. BissC Library Functions

NAME	DESCRIPTION	CPU CYCLES	TYPE
PM_bissc_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.	24,467	Initialization time
PM_bissc_getCrc	Calculates the n-bit CRC of a message buffer by using the lookup table to get the CRC of each byte. This function can be used for both position and data-CRC checks with the corresponding CRC table and polynomial.	120	Run time
PM_bissc_getBits	This function is used to extract the bits of interest from the receive data buffer. After every transaction, data received from the encoder is stored in the receive buffer. This function can be used to extract position bits, CRC, CDS data, and so forth from the receive data buffer.	165	Run time
PM_bissc_setCDBit	This function is used to configure what the CDM bit is in the upcoming single-cycle data (SCD) transfer. Every BiSS frame ends with a timeout, and during this time no further clocks are transmitted by the BiSS Master. Clock-line MA is held to the state of the bit set by this function. The inverse of the same is interpreted as the CDM bit by the slave.	6	Run time
PM_bissc_startOperation	This function initiates the BiSS-C transfer, called after PM_bissc_setupNewSCDTransfer. This function starts the transaction set up earlier by the PM_bissc_setupNewSCDTransfer function. The setup and start operation are separate function calls. The user can set up the transfer when needed and start the actual transfer using this function call, as needed, at a different time.	54	Run time
PM_bissc_setupPeriph	The setup for SPI, CLB, and other interconnect XBARs for BiSS-C are performed with this function during system initialization. This function must be called after every system reset. No transactions are performed until the setup peripheral function is called.	4,742	Initialization time
PM_bissc_setFreq	Function to set the clock frequency. Clock Frequency = SYSCLK/(4 × BISSC_FREQ_DIVIDER). Example: set BISSC_FREQ_DIVIDER to 25 for 2-MHz operation. (1)	230	Initialization time
PM_bissc_setupNewSCDTransfer	Set up an SPI and other modules for a given transaction. All the transactions must start with this command. This function call sets up the peripherals for the upcoming BiSS-C transfer, but does not actually perform any transfer or activity on the interface. When the transfer is set up using this function, PM_bissc_startOperation can be called to start the transfer.	742	Run time

2.3.3.2 PM BissC Library Data Structures

The PM BissC Library defines the BiSS-C data structure handle as follows:

Object definition:

```
typedef struct {                            // bit descriptions
                    uint16_t cd_status;     // 0 - no cd; 1 - cd transfer ongoing;
                                            // 2 - cd rx'd; 3 – cd parsed & complete
                    int16_t remaining_cd_bits;
                    uint16_t cd_bits_to_send;
                    uint32_t cdm;
                    uint32_t cds_stream;
                    uint16_t cds_raw;        // cds without the crc
                    uint16_t cd_register_xfer_address;
                    uint16_t cd_register_xfer_rxdata;
                    uint16_t cd_register_xfer_txdata;
                    uint16_t cd_register_xfer_is_write;
                    uint32_t scd_raw;       // scd without the crc
                    uint16_t scd_crc;
                    uint32_t position;
                     uint16_t scd_error;
                    uint16_t scd_warning;
                    uint16_t crc_incorrect_count;
                    uint16_t dataReady;
                    uint32_t rdata[16];     // Receive data buffer
                    uint16_t fifo_level;
                    uint16_t xfer_address_withCTS;
                    volatile struct SPI_REGS *spi;
} BISSC_DATA_STRUCT;
```

表 5 lists the module interface definitions.

表 5. Module Interface Definitions

MODULE ELEMENT NAME	DESCRIPTION	TYPE
cd_status;	Status of the Control Data (CD) protocol • 0 – trigger new CD transfer cd_status; • 1 – CD transaction in progress • 2 – CD response received and is being parsed • 3 – CD transaction complete	uint16_t
remaining_cd_bits;	This variable shows how many more SCD transactions must occur before the CD data is complete.	int16_t
cd_bits_to_send;	Indicates the number of total bits in a CD transaction	uint16_t
cdm;	Stores the CD data that is sent out through the SCD bit-by-bit	uint32_t
cds_stream;	Contains the CDS (response from the encoder)	uint32_t
cds_raw;	Internal variables used by the library – for debug purposes	uint16_t
cd_register_xfer_address;	The address in the encoder to be read or written	uint16_t
cd_register_xfer_rxdata;	After a CD transaction, this variable shows the current value stored in cd_register_xfer_address.	uint16_t
cd_register_xfer_txdata;	If the CD transfer is a write, this is what is written in cd_register_xfer_address after a CD transfer.	uint16_t
cd_register_xfer_is_write;	Configures whether the next CD transmission is a read or write transaction	uint16_t
scd_raw;	Internal variables used by the library – for debug purposes	uint32_t
scd_crc;	The received SCD CRC value	uint16_t
position;	The current position of the encoder	uint32_t
scd_error;	The error status of the encoder	uint16_t
scd_warning;	The warning status of the encoder	uint16_t
crc_incorrect_count;	Counts the amount of CRC errors accumulated	uint16_t
dataReady;	Flag indicating when the data is ready – cleared by PM_bissc_startOperation;	uint16_t
rdata[16];	Internal variables used by the library – for debug purposes	uint32_t
fifo_level;	Internal variables used by the library – for debug purposes	uint16_t
xfer_address_withCTS	Internal variables used by the library – for debug purposes	uint16_t
spi	SPI instance used for BiSS-C implementation	Pointer to Spi*Regs

2.3.3.3 PM BissC Library Function Details

PM_bissc_setFreq

Directions:

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 the pTable.

Definition:

```
void PM_bissc_setFreq(uint32_t Freq_us);
```

Parameters:

INPUT		RETURN
Freq_us	Clock divider for the system clock sets BiSS-C Clock Frequency = SYSCLK/(4 × Freq_us)	None

Use:

```
//during initialization
PM_bissc_setFreq(FREQ_DIVIDER);
```

PM_bissc_getCRC

Directions:

Calculate the n-bit CRC of a message buffer by using the lookup table to get the CRC of each byte. This function can be used for both position and data-CRC checks, with the corresponding CRC table and polynomial. The CRC table must be initialized through PM_bissc_generateCRCTable before calling the PM_bissc_getCRC function.

Definition:

```
uint16_t PM_bissc_getCRC(uint16_t input_crc_accum,
uint16_t nBitsData, uint16_t nBitsPoly, uint16_t * msg,
uint16_t *crc_table, uint16_t rxLen);
```

Parameters:

INPUT		RETURN	
input_crc_accum	Initial CRC value (seed) for CRC calculation	crc	5-bit, CRC value is calculated
nBitsData	Number of bits of data for which the CRC is calculated		
nBitsPoly	Number of bits of polynomials used for CRC computations		
msg	Pointer to the data on which CRC is computed		
crc_table	Pointer to the table where the CRC table values are stored		
rxLen	Number of bytes of data for CRC calculation		

Use:

Define BiSS-C Data structure during initialization.

```
BISSC_DATA_STRUCT bissc_data_struct;
```

CRC computation of single-cycle data (SCD):

```
crcResult = PM_bissc_getCRC(0,          // Initial seed
positionBits+2,                         // positionBits + Error + Warning
BISS_SCD_CRC_NBITS_POLY1,               // SCD polynomial bits
(uint16_t *)&bissc_data_struct.scd_raw, // SCD data
bissCRCtableSCD,                        // CRC table with SCD polynomial
3);
```

CRC computation of control data (CD):

```
crcSelf = PM_bissc_getCRC(0,                          // Initial seed
11,                                                   // Number of Control Data bits for CRC
BISS_CD_CRC_NBITS_POLY1,                              // Control Data polynomial bits
(uint16_t *)&bissc_data_struct.xfer_address_withCTS, // CD Data
bissCRCtableCD,                                       // CRC table with CD polynomial
 2);
```

PM_biss_generateCRCTable

Directions:

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

Definition:

```
void PM_bissc_generateCRCTable(uint16_t nBits, uint16_t polynomial, uint16_t *pTable);
```

Parameters:

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

Use:

CRC table for SCD

```
#define BISS_SCD_CRC_NBITS_POLY1 6
#define BISS_SCD_CRC_POLY1 0x03
#define SIZEOFTABLE 256
uint16_t bissCRCtableSCD[SIZEOFTABLE];

// Generate table for poly 1
PM_bissc_generateCRCTable(BISS_SCD_CRC_NBITS_POLY1, BISS_SCD_CRC_POLY1, bissCRCtableSCD);
```

CRC table for CD

```
#define BISS_CD_CRC_NBITS_POLY1 4
#define BISS_CD_CRC_POLY1 0x03
#define SIZEOFTABLE 256
uint16_t bissCRCtableCD[SIZEOFTABLE];

// Generate table for poly 1
PM_bissc_generateCRCTable(BISS_CD_CRC_NBITS_POLY1, BISS_CD_CRC_POLY1, bissCRCtableCD);
```

PM_bissc_startOperation

Directions:

This function initiates a BiSS-C transfer, to be called after PM_bissc_setupNewSCDTransfer. This function starts the transaction set up earlier by the PM_bissc_setupNewSCDTransfer function. The setup and start operation are separate function calls. The user can set up the transfer when needed, and start the actual transfer using this function call, as needed, at a different time.

Definition:

```
void PM_bissc_startOperation(void);
```

Parameters:

INPUT	RETURN
None	None

Use:

Example code:

```
PM_bissc_setupNewSCDTransfer(BISS_DATA_CLOCKS, SPI_FIFO_WIDTH); PM_bissc_startOperation();
```

This function clears the bisscData.dataReady flag zero when called. This flag is subsequently set by the SPI interrupt service routine when the data is received from the encoder. The user can poll for this flag to know if the data from the encoder is successfully received after the PM_bissc_startOperation function call.

PM_bissc_getBits

Directions:

This function is used to extract the bits of interest from the receive data buffer. After every transaction, data received from the encoder is stored in the receive buffer. This function can be used to extract position bits, CRC, CDS data, and so forth from the receive data buffer.

Definition:

```
uint32_t PM_bissc_getBits (uint16_t len, uint16_t bitsParsed, uint16_t charBits);
```

Parameters:

INPUT		RETURN	
len	Length of the string to be extracted from the receive data buffer	—	Extracted bits of the array from specified bit index and length.
bitsParsed	Number of bits already parsed, or bit index to start extracting bits		
charBits	This field indicates the number of valid bits in each entry of the array. This input corresponds to the number of bits in each SPI FIFO entry or the receive data buffer.		

Use:

Example code:

```
//Extract position bits along with error and warning flags
bissc_data_struct.scd_raw = PM_bissc_getBits(positionBits+2, scdBitsParsed, SPI_FIFO_WIDTH);
scdBitsParsed = scdBitsParsed + positionBits + 2;         //positionBits + E + W

//Extract CRC bits for the SCD position data
bissc_data_struct.scd_crc = PM_bissc_getBits(crcBits, scdBitsParsed, SPI_FIFO_WIDTH);
```

PM_bissc_setupNewSCDTransfer

Directions:

This functions configures the library so that it can perform a new BiSS-C transaction. This function must be called each time a SCD must occur, and must happen before every PM_bissc_startOperation call. This function only sets up a transaction to occur; to actually start the transaction, call PM_bissc_startOperation.

Definition:

```
void PM_bissc_setupNewSCDTransfer(uint16_t nDataClks, uint16_t spi_fifo_width);
```

Parameters:

INPUT		RETURN	
nDataClks	nDataClks – Number of bits of data from (including) start bit • For Lika HS58S18/I7-P9-RM2 it is 34 → 32 (pos+E+W+posCRC) + CDS + Start • For Kuebler 8.F5863.1426.C423 it is 36 → 34 (pos+E+W+posCRC) + CDS + Start	—	None
spi_fifo_width	Number of bits in each SPI FIFO entry or the receive data buffer, such as SPI character length		

Use:

Example code:

```
#define SPI_FIFO_WIDTH 12            // SPI character length chosen
#define BISS_DATA_CLOCKS 34          // Lika HS58S18/I7-P9-RM2 it is 34 ==>
// 32 (pos+E+W+posCRC) + CDS + Start PM_bissc_setupNewSCDTransfer(BISS_DATA_CLOCKS,
SPI_FIFO_WIDTH);
```

PM_bissc_setCDBit

Directions:

This function is used to set up the CDM bit for the upcoming SCD transfer. Every BiSS frame ends with a timeout, and during this time no further clocks are transmitted by the BiSS Master. Clock-line MA is held to the state of the bit set by this function. The inverse of the same bit is interpreted as the CDM bit by the slave.

Definition:

```
void PM_bissc_setCDBit(uint32_t cdmBit);
```

Parameters:

INPUT		RETURN	
cdmBit	CDM bit to be transmitted in the upcoming BiSS Transaction. This bit is transmitted on the clock line MA.	—	None

Use:

Example code:

```
PM_bissc_setCDBit(temp32);
```

Set the CDM bit before starting the BiSS-C transfer using PM_bissc_startOperation. If no CDM bit is set for the current transfer, the module transmits a default value of 1.

PM_bissc_setupPeriph

Directions:

This function performs set up for SPI, CLB, and other interconnect XBARs for the BiSS-C during system initialization. This function must be called after every system reset. No transactions are performed until the setup peripheral function is called.

Definition:

```
void PM_bissc_setupPeriph (void);
```

Parameters:

INPUT	RETURN
None	None

Use:

Example code:

```
bissc_data_struct.spi = &SpibRegs;
PM_bissc_setupPeriph();
```

The PM_bissc library uses an instance of SPI for communication. For proper initialization, the SPI instance must be set in bissc_data_struct.spi before calling this function, as previously shown. When changing the SPI instance used, change the configuration for the Peripheral Interrupt Expansion (PIE) block as well.

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 the TIDM-1010 device and how to get started with the BiSS-C Library in CCS.

To experiment with the TIDM-1010 device, the following components are required:

- TIDM-1010 EVM board
- External, 5-V, DC power supply (see 表 1)
- F28379D LaunchPad development kit (LAUNCHXL-F28379D)
- USB-B to A cable
- BiSS-C Encoder (for example, Kuebler 8.F5863.1426.C423)
- Cable (for the Kuebler encoder, an 8-pole, M12 cable – length as determined by the application (maximum 100 m))
- Custom adapter to connect the female-terminated cable to the wire-leads adapter
- PC with CCS (CCS v6 or greater) installed

3.1.1.1 TIDM-1010 Jumper Configuration

图 6 shows the jumper configuration for the TIDM-1010/BOOSTXL-POSMGR board.

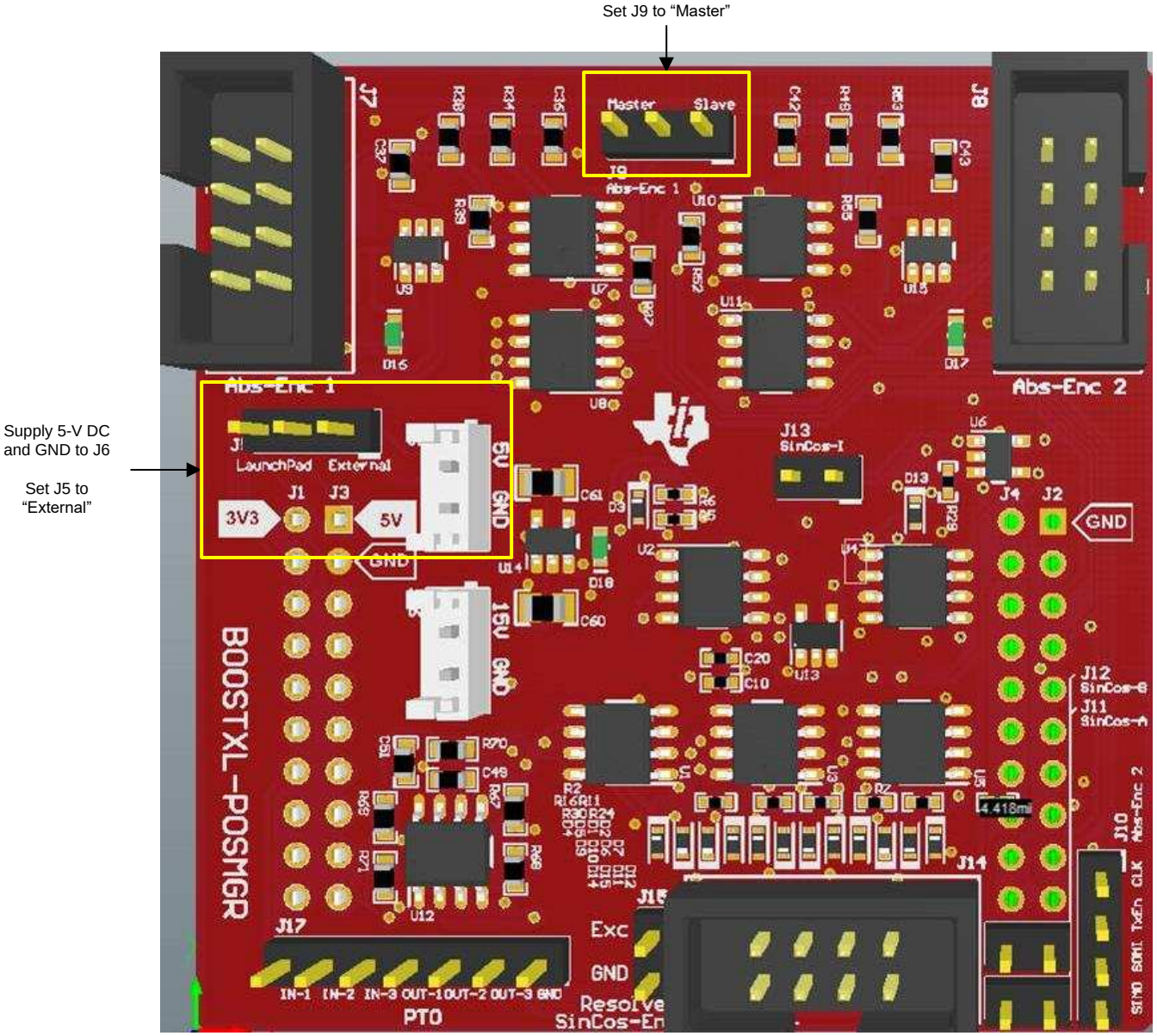


图 6. TIDM-1010 Board Jumper Configuration

表 6 lists the jumper configuration for the TIDM-1010 board.

表 6. TIDM-1010 Board Jumper Details

JUMPER	FUNCTION	POSITION
J5	TIDM-1010, 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 users provide an external power source to J6, as shown in 图 6.
(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 CCS™ and controlSUITE™

1. Install CCS v6.x or later, if it is not already on the PC.
2. Go to controlSUITE and run the controlSUITE installer. Allow the installer to download and update any automatically checked software for the C2000.
3. After installation, see 节 2.3.3 for more information on the BiSS-C Library.

3.1.2.2 Configuring CCS for F28379D LaunchPad™

1. Open CCS. This reference design assumes that version 6 or later is used.
2. When CCS opens, the workspace launcher asks the user to select a workspace location. The workspace is a location on the hard drive where all the user settings for the IDE (which projects are open), the selected configuration, and so forth are saved. This workspace can be anywhere on the disk, the location mentioned below is just for reference. If this is not the first time running CCS, the following dialog may not appear.
   1. Click the *Browse…* button.
   2. Create the following path by making new folders as necessary:
      `C:\c2000 projects\CCSv6_workspaces\PM_BiSS-C_eval_workspace`
   3. Uncheck the box that says *Use this as the default and do not ask again*.
   4. Click the *OK* button.

   A *Getting Started* tab opens with links to various tasks, from creating a new project, importing an existing project, and watching a tutorial on CCS. Users can close the *Getting Started* Tab, and go to the next step. CCS is configured to know which MCU the program connects to. This configuration is done by setting up the *Target Configuration*.
3. Set a new configuration file by clicking *View* → *Target Configuration*. This procedure opens the Target Configuration window.
4. In the Target Configuration window, click on the  icon.
5. Name the new configuration file depending on the target device. If the *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. The previous step opens up a new tab, as shown in 图 7. Select and enter the following options:
   1. Connection – select *Texas Instruments XDS100v2 USB Emulator* or *Texas Instruments XDS100v2 USB Debug Probe*.
   2. Device – select the C2000 MCU on the control card, TMS320F28379D, for example.
   3. Click the *Save* button and then close.

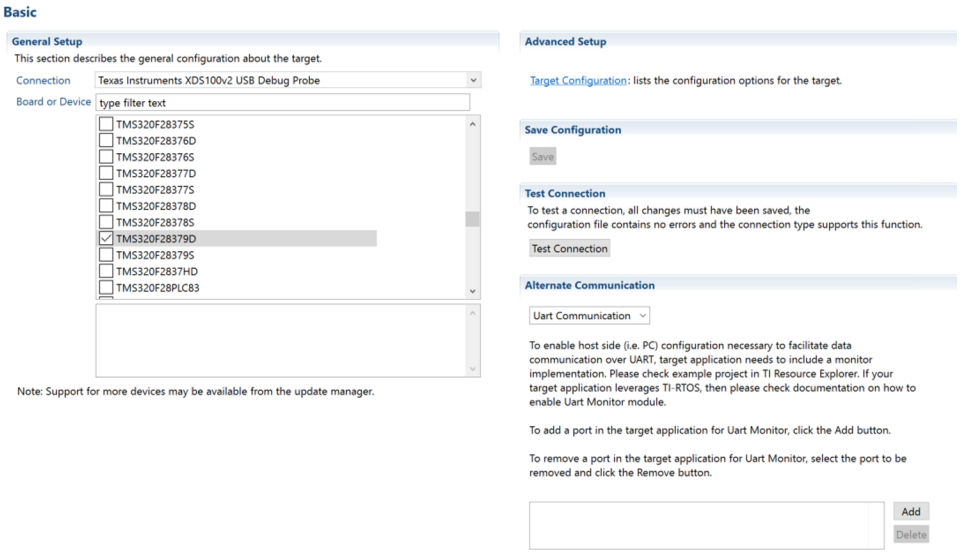


图 7. Configure New Target Configuration

7. Click *View* → *Target Configurations*. Then, in the *User Defined* section, find the file that was created in Steps 5 and 6.
8. 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_bissc_SystemTest*, 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 TIDM-1010 Example Project

1. Add the PM BiSS-C evaluation-example project into the current workspace by clicking *Project* → *Import CCS Project*.
   1. Select the project by browsing to:

      `C:\ti\controlSUITE\development_kits\BOOSTXL_POSMGR`
   2. A window similar to 图 8 appears. Click the *Finish* button.

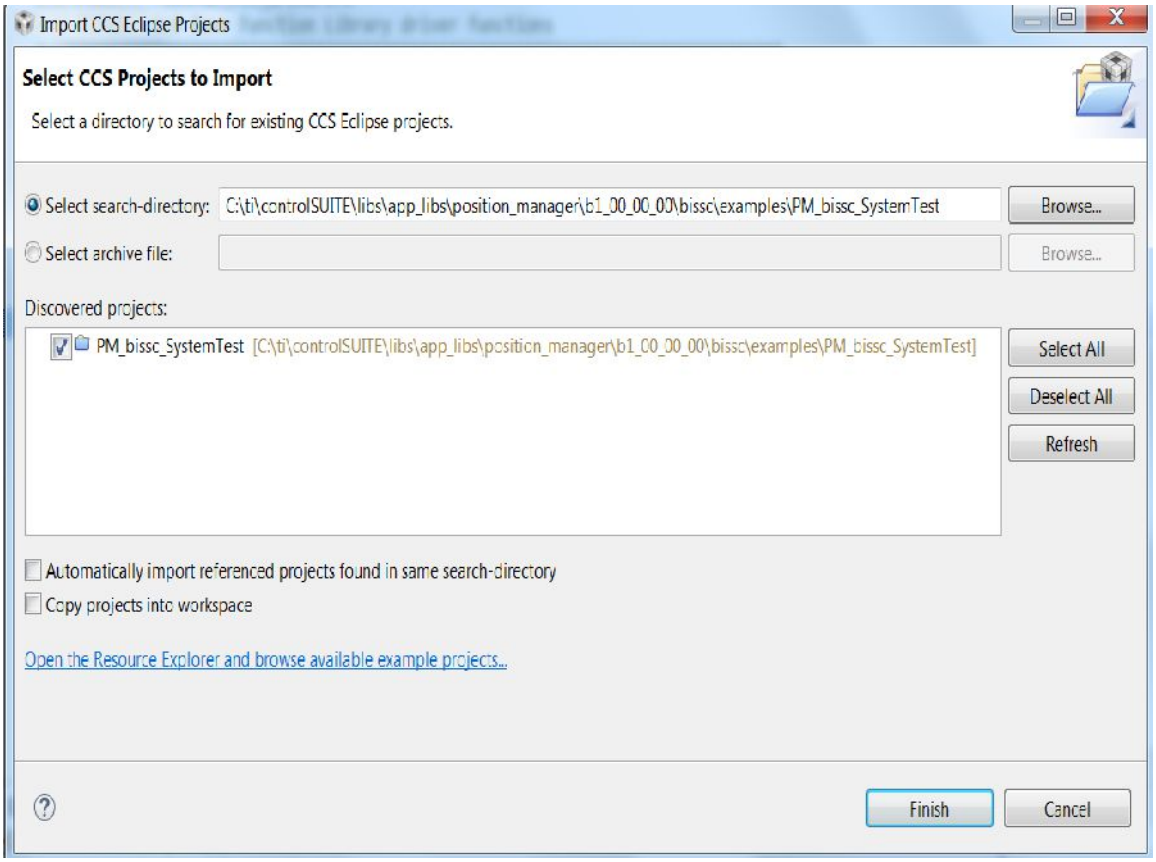


图 8. Adding PM Biss-C Example Project to Workspace

注: By default, CCS does not copy the project into the workspace. Any changes made to files within CCS are thus reflected in the files stored in the controlSUITE installation. If users desire to preserve the original files stored in controlSUITE, check the box *Copy projects into workspace*, as shown in 图 8.

2. Assuming this is the first use of CCS, xds100v2-F2837x should be set as the default target configuration. Verify this setting by viewing the xds100v2-f2837x.ccxml file in the expanded project structure and observing an *Active* or *Default* status written next to the 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*.

   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.

3. As shown in 图 9, right-click on an individual project and select *Active Build Configuration* → *CPU1_RAM configuration*.

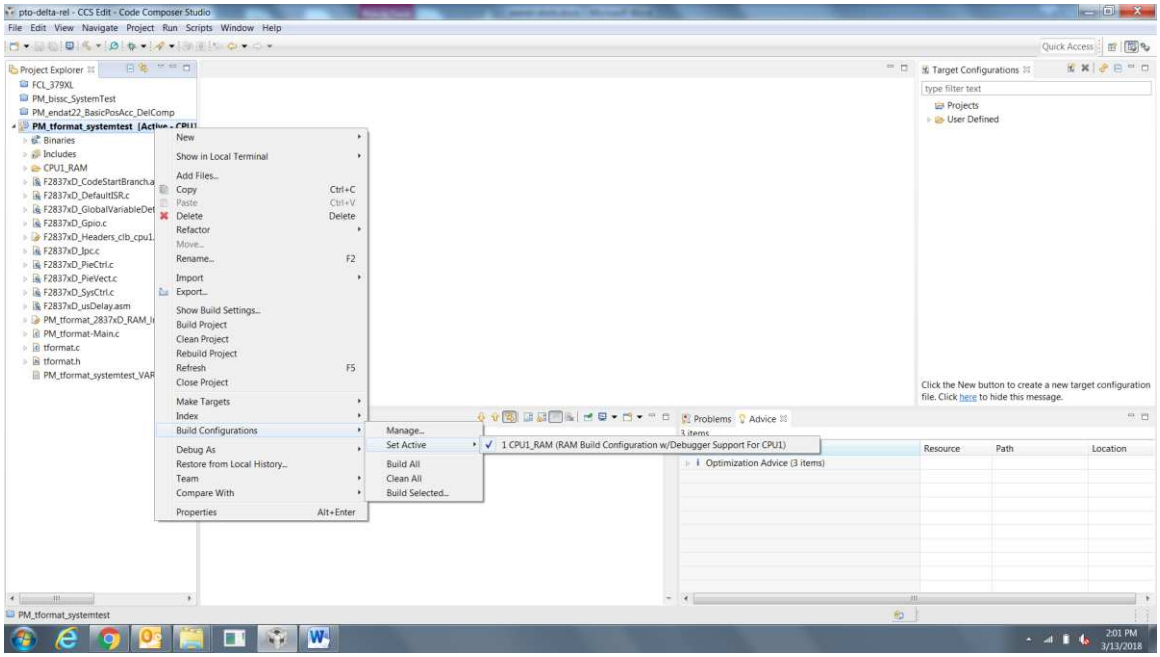


图 9. Selecting F2837x_RAM Configuration

3.2 Testing and Results

3.2.1 Test Setup

3.2.1.1 Hardware Configuration

1. Ensure that the jumper configuration of the TIDM-1010 device is as described in 表 6.
2. Connect the TIDM-1010 device to the LaunchPad using the BoosterPack connector (J1, J3 and J4, J2). Ensure the TIDM-1010 device is connected to site two of the LaunchPad, as shown in 图 10.

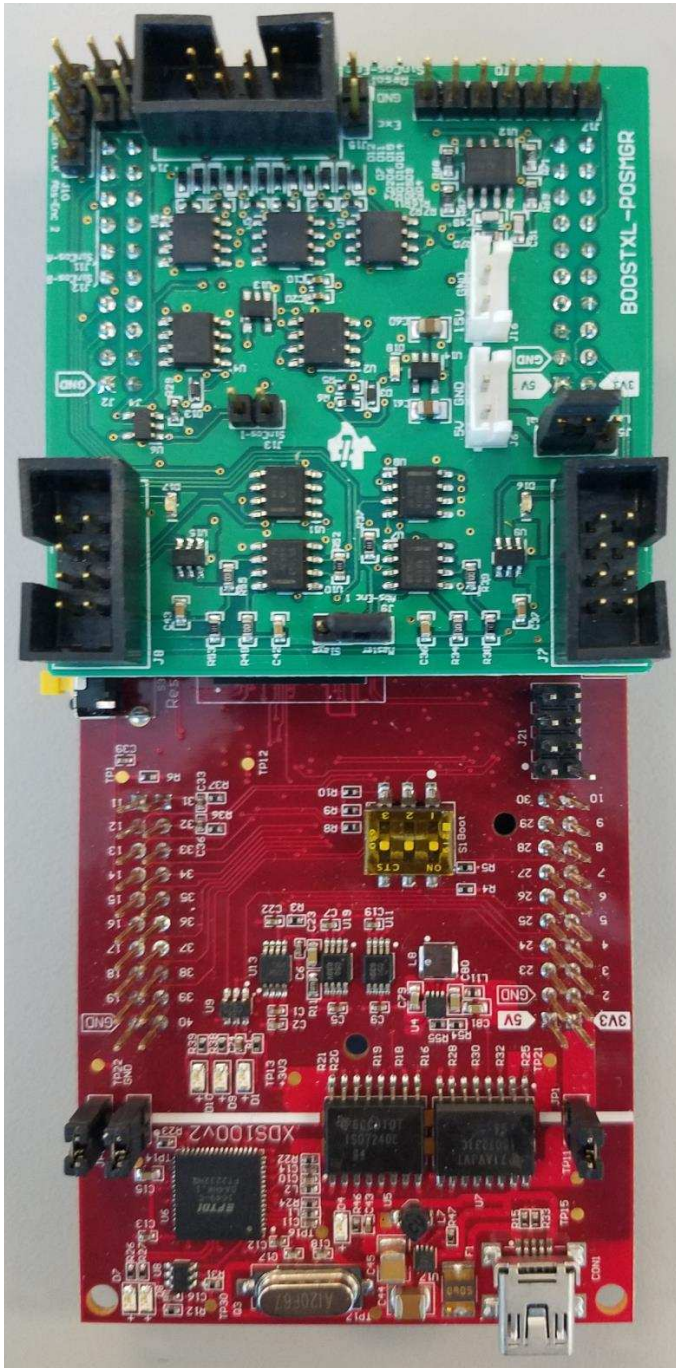


图 10. TIDM-1010 Board Connected to Site Two of LaunchPad™

3. Connect the USB cable to the LaunchPad.

4. Connect to the encoder:
   1. Prepare an adapter to connect the cable to the BiSS-C interface using the circular female to wire the leads adapter (see the BOM for the header used for the encoder connector, J7).
   2. Insert the header of the adaptor created in the previous step to connect to Abs-Enc-1 (J7). The female end of the cable connects to the encoder. 图 11 shows the pinout of J7.

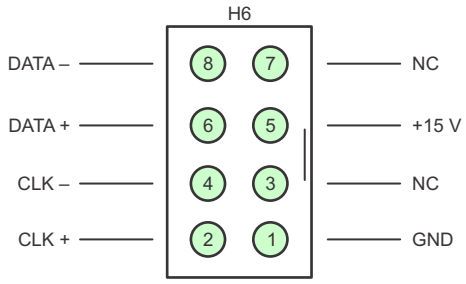


图 11. Abs-Enc-1 (J7) Pinout on TIDM-1010 Board

5. Supply 5-V DC and GND to J6, as shown in 图 6. The board should now look like 图 12. LED D18 should be lit, which shows that the board has power.

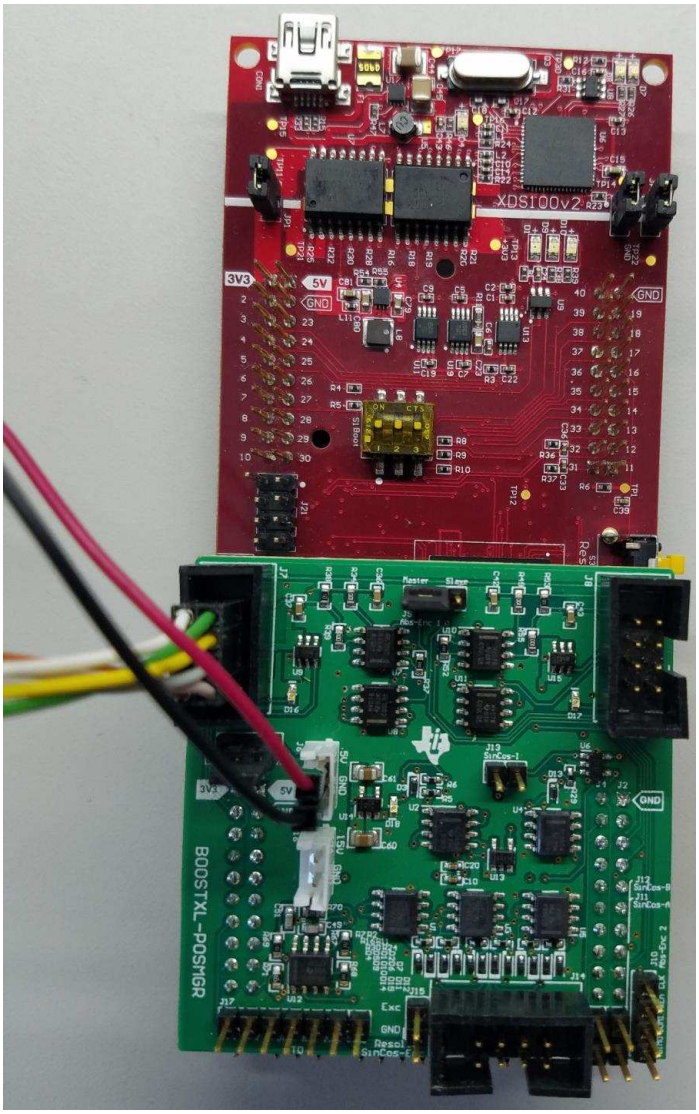


图 12. TIDM-1010 Board Connected to BiSS-C Encoder

3.2.2 Test Results

This section describes how to run the software example and detail the results in CCS. 图 13 shows the software flow diagram for this project.

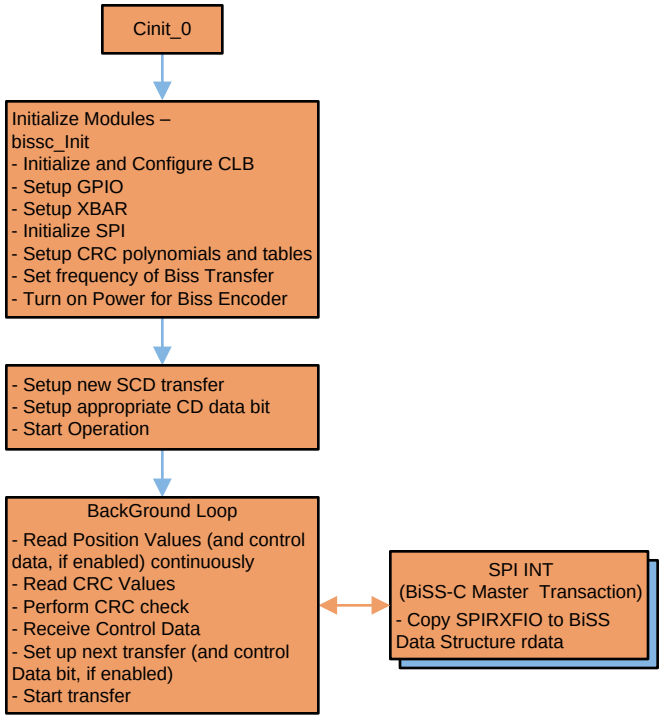


图 13. Software Flow Diagram for PM_bissc_SystemTest Example Project

3.2.2.1 Building and Loading Project

1. Complete the software setup as described in 节 3.1.2 .
2. Open the bissc.h file and ensure BISSC_FREQ_DIVIDER is set as needed. Save this file. For more information see the *C2000 Position Manager BISS-C Library User's Guide*.
3. Right-click on the project name and select *Rebuild Project*, then watch the console window. Any errors in the project are displayed in the console window.
4. Upon successful completion of the build, click the Debug button , on the top-left side of the CCS window. If a window appears prompting to select a CPU, ensure CPU1 is selected, and click the *OK* button (see 图 14).

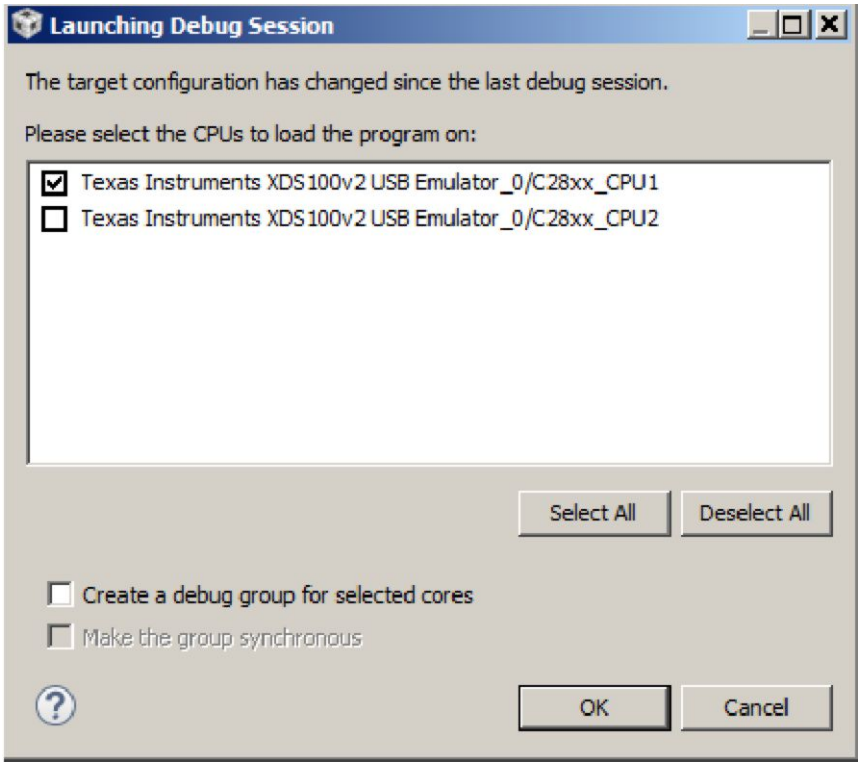


图 **14. Selecting CPUs to Connect**

5. The IDE automatically connects to the target, loads the output file into the device, and changes to the debug perspective.
6. Click the *Tools → Debugger Options → Program / Memory Load Options* option. The debugger can be enabled to reset the processor each time the debugger reloads the program by checking *Reset the target on program load or restart*, and clicking *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 sets bit 1 (DGBM bit) of the status register 1 (ST1) to a 0. 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.

注: 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*.
2. Browse to the .txt file containing these variables.
3. For this project, browse to the root directory and select *Pm_bissc_SystemTest_VAR.txt*, and then click the *OK* button to import the variables, as shown in 图 15.
4. 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.

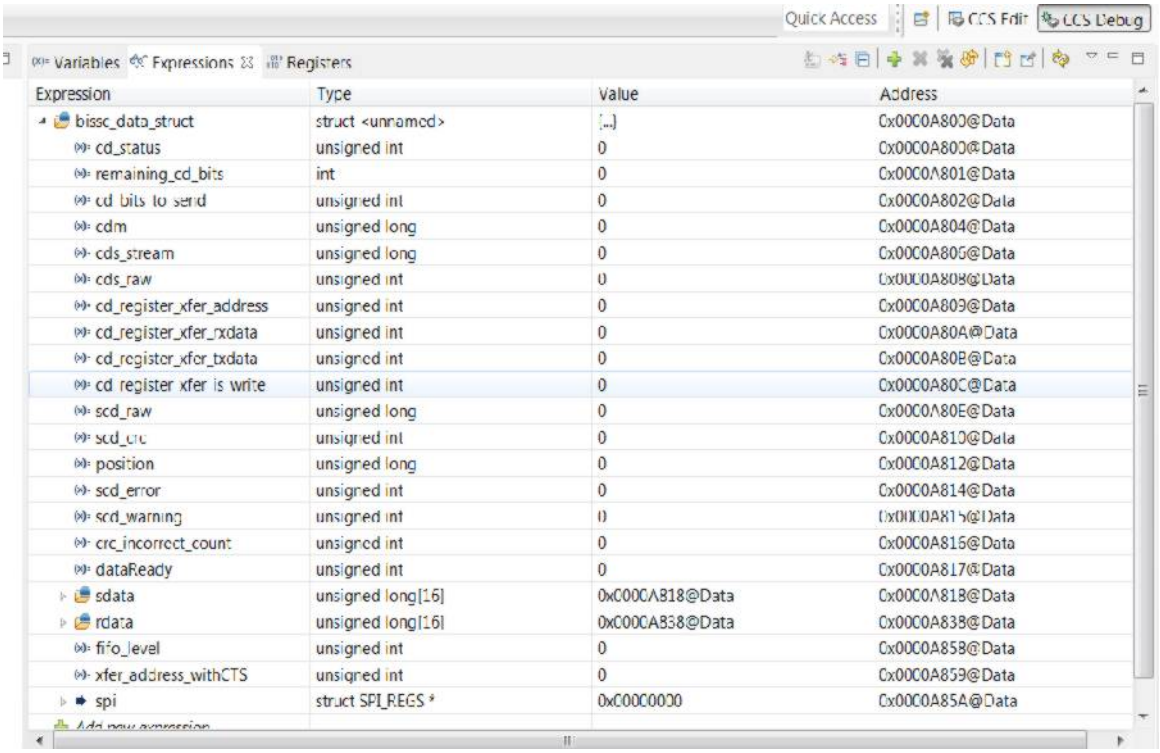


图 15. Properly Configured Watch Window

5. Each variable uses the number format that the variable is associated with during declaration. The variable can be changed to another number format by right-clicking on it. 图 15 shows a typical expressions window.
6. Click on the *Continuous Refresh* button , in the watch window. This button 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.

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

3.2.2.3 Running Example Code

1. Run the code by pressing the *Run* button , in the Debug tab.

   The project runs and the values in the watch window continuously update. If the encoder is not mounted on a spinning motor, the user can manually rotate the encoder shaft and observe the variables accordingly change:

   - As the encoder sends the position information, observe the same in the watch window as the *biss_data_struct.position* variable.
   - The *biss_data_struct.position* variable reflects the latest position received by the encoder.

2. When 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. It is not necessary to terminate the debug session each time the user changes or runs the code. Instead, after rebuilding the project, use the following procedure (Run → Reset → CPU Reset) then (Run → Restart), and enable real-time options.
4. Once complete, disable real-time options and reset the CPU.
5. Terminate the project if the target device or the configuration is changed (RAM to flash or flash to RAM) and before shutting down CCS.
6. Customize the project to meet your encoder and application requirements. Change the encoder type in bissc.h. Users can also change the BiSS-C Clock frequency as instructed in the *C2000 Position Manager BISS-C Library User's Guide* .
7. Modify the example code as needed, and perform tests with different cable lengths.

3.2.2.4 Cable Length Validation

表 7 lists tests with various types of encoders; cable length tests were performed at Texas Instruments. The tests include basic command-set exercising and reading position values with additional data if applicable.

表 7. Cable Length Test Report

ENCODER MANUFACTURER	ENCODER NAME	TYPE	RESOLUTION (BITS)	CABLE LENGTH(1) (m)	MAXIMUM BiSS CLOCK (MHz)	CD INTERFACE	TEST RESULTS
Lika	HS58S18/I7	Rotary	18 bits (padded to 24 bits)	100 m	3.33 MHz	Yes	Pass
Kuebler	8.F5863.1426 .C423	Rotary	26 bits (12 plus 14)	100 m	5 MHz	No	Pass

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

4 Design Files

To download the design files, see the product page at TIDM-1010.

5 Software Files

To download the software files, see the product page at TIDM-1010.

6 Related Documentation

1. Texas Instruments, *Position Manager BiSS-C Library User's Guide*
2. Texas Instruments, C2000 DesignDRIVE, software for Industrial Drives and Motor Control
3. Texas Instruments, *C2000 Position Manager SinCos Library*, user's guide

6.1 商标

7 Terminology

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-1010 board is identical to the Position Manager BoosterPack EVM (see 节 2.3.1)

CRC— Cyclic redundancy check

BiSS-C— An open-source digital interface for sensors and actuators

PM— Position Manager – foundational hardware and software on C28x devices for position encoder interfaces

PM_bissc— Prefix used for all the library functions

SPI— Serial peripheral interface

8 About the Authors

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

SHEENA PATEL works on the Industrial Drives team of the C2000 MCU group as a Product Marketing Engineer.

有关 TI 设计信息和资源的重要通知

德州仪器 (TI) 公司提供的技术、应用或其他设计建议、服务或信息，包括但不限于与评估模块有关的参考设计和材料（总称“TI 资源”），旨在帮助设计人员开发整合了 TI 产品的 应用； 如果您（个人，或如果是代表贵公司，则为贵公司）以任何方式下载、访问或使用了任何特定的 TI 资源，即表示贵方同意仅为该等目标，按照本通知的条款进行使用。

TI 所提供的 TI 资源，并未扩大或以其他方式修改 TI 对 TI 产品的公开适用的质保及质保免责声明；也未导致 TI 承担任何额外的义务或责任。TI 有权对其 TI 资源进行纠正、增强、改进和其他修改。

您理解并同意，在设计应用时应自行实施独立的分析、评价和 判断， 且应全权负责并确保 应用的安全性， 以及您的 应用 （包括应用中使用的所有 TI 产品）） 应符合所有适用的法律法规及其他相关要求。你就您的 应用声明，您具备制订和实施下列保障措施所需的一切必要专业知识，能够 (1) 预见故障的危险后果，(2) 监视故障及其后果，以及 (3) 降低可能导致危险的故障几率并采取适当措施。您同意，在使用或分发包含 TI 产品的任何 应用前， 您将彻底测试该等 应用 和该等应用所用 TI 产品的 功能而设计。除特定 TI 资源的公开文档中明确列出的测试外，TI 未进行任何其他测试。

您只有在为开发包含该等 TI 资源所列 TI 产品的 应用时， 才被授权使用、复制和修改任何相关单项 TI 资源。但并未依据禁止反言原则或其他法理授予您任何TI知识产权的任何其他明示或默示的许可，也未授予您 TI 或第三方的任何技术或知识产权的许可，该等产权包括但不限于任何专利权、版权、屏蔽作品权或与使用TI产品或服务的任何整合、机器制作、流程相关的其他知识产权。涉及或参考了第三方产品或服务的信息不构成使用此类产品或服务的许可或与其相关的保证或认可。使用 TI 资源可能需要您向第三方获得对该等第三方专利或其他知识产权的许可。

TI 资源系“按原样”提供。TI 兹免除对 TI 资源及其使用作出所有其他明确或默认的保证或陈述，包括但不限于对准确性或完整性、产权保证、无屡发故障保证，以及适销性、适合特定用途和不侵犯任何第三方知识产权的任何默认保证。

TI 不负责任何申索，包括但不限于因组合产品所致或与之有关的申索，也不为您辩护或赔偿，即使该等产品组合已列于 TI 资源或其他地方。对因 TI 资源或其使用引起或与之有关的任何实际的、直接的、特殊的、附带的、间接的、惩罚性的、偶发的、从属或惩戒性损害赔偿，不管 TI 是否获悉可能会产生上述损害赔偿，TI 概不负责。

您同意向 TI 及其代表全额赔偿因您不遵守本通知条款和条件而引起的任何损害、费用、损失和/或责任。

本通知适用于 TI 资源。另有其他条款适用于某些类型的材料、TI 产品和服务的使用和采购。这些条款包括但不限于适用于 TI 的半导体产品 (http://www.ti.com/sc/docs/stdterms.htm)、评估模块和样品 (http://www.ti.com/sc/docs/sampterms.htm) 的标准条款。

邮寄地址：上海市浦东新区世纪大道 1568 号中建大厦 32 楼，邮政编码：200122