User's Guide

C2000 Position Manager BISS-C Library

Subrahmanya Bharathi and Brett Larimore

Table of Contents

List of Figures

List of Tables

Trademarks

All trademarks are the property of their respective owners.

1 Introduction

1.1 BiSS-C interface

BiSS is an Open Source digital interface for sensors and actuators. BiSS is hardware-compatible to the industrial standard SSI (Serial Synchronous Interface), and also offers additional features and options:

- Serial synchronous data communication
- Unidirectional lines clock and data
  - Cyclic at high speed (up to 10 MHz with RS422 and 100 MHz with LVDS)
  - Line delay compensation for high speed data transfer
  - Request processing times for data generation at slaves
  - Safety capable: CRC, errors, warnings
  - Bus capability for multiple slaves and devices in a chain
- Unidirectional – BiSS C (unidirectional) protocol: unidirectional use of BiSS C
- Bidirectional – BiSS C protocol: continuous mode

More details can be found at http://www.biss-interface.com/.

1.2 System Description

Industrial drives, like servo drives, require accurate, highly reliable, and low-latency position feedback. A simplified system block diagram of a servo drive using an absolute position encoder with BiSS-C digital interface is shown in Figure 1-1. The interface transmits position values and additional information from Encoder (BiSS-C Slave) the MCU (BiSS-C Master). It also allows reading and writing of the internal memory of the encoder, and the type of data transmitted like absolute position, turns, temperature, parameters, diagnostics, and so on. The BiSS-C interface is a pure serial digital interface based on RS-485 standard.

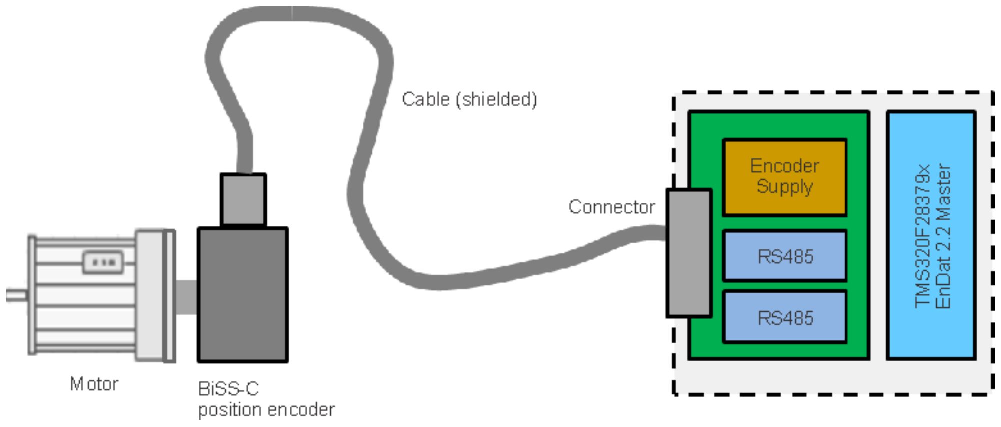


Figure 1-1. Industrial Servo Drive With BiSS-C Position Encoder Interface

1.2.1 BiSS-C Point to Point

The point-to-point configuration is typically used with BiSS position or rotary encoders, as shown in Figure 1-2. 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_Lib only supports this configuration.

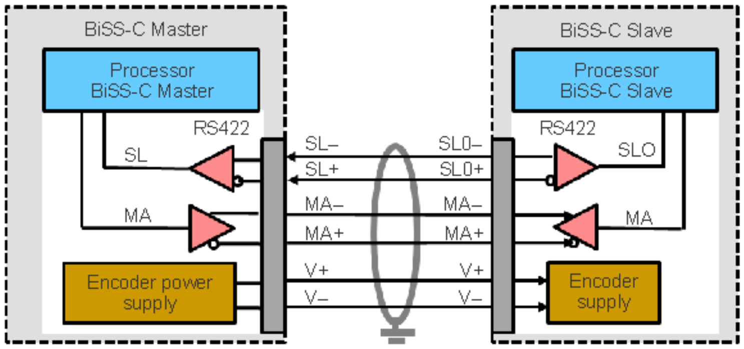


Figure 1-2. BiSS-C Point-to-Point Structure

In this on configuration two signals, MA (clock) and SL (data), are used for communication. These signals are differential in nature, resulting in the 4-wires, namely MA+, MA- and SL+, SL-, in unidirectional full-duplex mode. Two additional wires are for the encoder power supply V+ and V-, where V- is typically GND. The MA clock frequency is variable. The recommended MA clock frequency depends on the cable length, as outlined in BiSS Interface AN15: BiSS C MASTER OPERATION DETAILS Rev A2 with a maximum operating frequency of up to 10 MHz. 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, as the quality of the cable has an impact on the communication performance.

More details of the protocol and point-to-point configuration can be found at http://www.biss-interface.com/.

1.3 C2000 BiSS-C Master Solution

The Texas Instruments C2000 Position Manager BissC (PM_bissc) library is intended to provide support for implementing the BiSS-C interface in microcontroller (master).

Features offered by Bissc library:

- Integrated MCU solution for BiSS-C interface
- Meets BiSS-C point-to-point digital interface protocol requirements
- Clock frequency of up to 8 MHz supported irrespective of cable length – achievable frequency depends on the encoder and cables used.
- Verified operation up to 100m cable length using RS485/RS422 transceivers
- Easy interface to commands through driver functions and data structure provided by the library
- Integrated cable propagation delay compensation algorithm configurable through drivers provided in library.
- Efficient and optimized CRC algorithm for both position and data CRC calculations
- Example projects illustrating the CDM/CDS register access interface
- Solution-tuned for position control applications, where position information is obtained from encoders every control cycle, and with better control of modular functions and timing.

Key things to note while using BiSS-C library:

- This library supports only the basic interface drivers for BiSS-C operation. All higher level application software must be developed by users utilizing the basic interface provided by this library.
- Single-Cycle-Data transactions supported
- The Control CD transactions are partially supported. Only register communication has been implemented.
- Clock frequency for the BiSS-C Clock is limited to a maximum of SYSCLOCK/24. This limitation applies irrespective of the cable length and encoder type.
- For the functionality verified using this library, see Section 5. For any additional functionality or encoder usage not specified in this section, contact the TI support team.

1.4 BiSS-C Master Implementation Details

This section gives a brief overview of how the BiSS-C interface is implemented on TMS320F28379D devices.

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

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

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

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

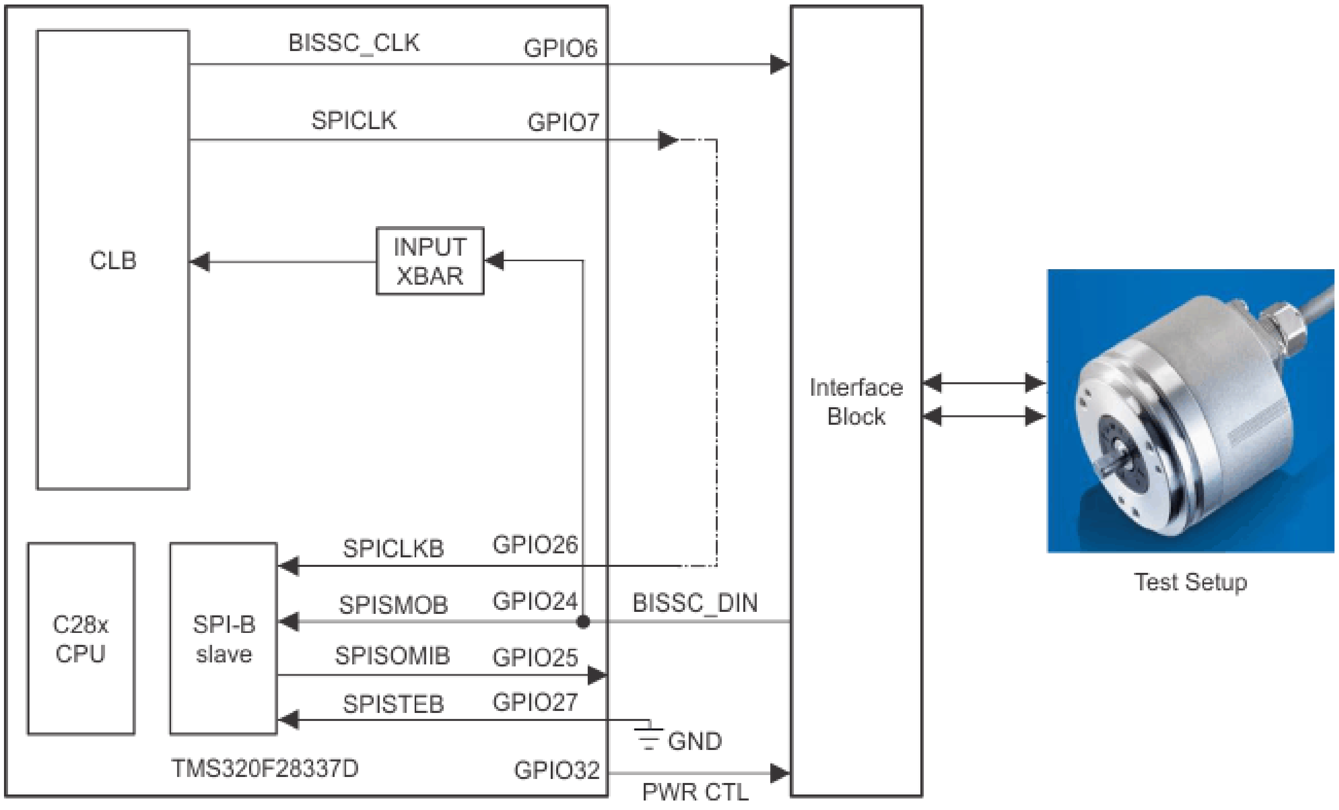


Figure 1-3. BiSS-C Implementation Diagram Inside TMS320F28379D

GPIOs indicated in Figure 1-3 are as implemented on TMDXIDDK379D.

Figure 1-3 depicts how BiSS-C transaction works in the system. For every BiSS-C transaction initiated using the PM_bissc Library command:

- The CPU configures the SPITXFIFO with the command and other data required for transmission to the encoder, as per the specific requirements of the current BiSS-C transaction.
- The CPU sets up configurable logic block to generate clocks for the encoder and SPI.
- The number of clock pulses and edge placement for these two clocks is different and is precisely controlled by CLB – as configured by CPU software for the current transaction.
- The CLB monitors the SPISIMO signal, as needed, for detecting the start pulse, and adjusts the phase of the receive clock accordingly.
- The CPU configures CLB to generate continuous clocking for the encoder while waiting for a Start pulse from the encoder.
- The 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 a Start pulse from the encoder.
- The CLB also provides hooks to perform cable propagation delay compensation through the library functions.

More details on various library functions provided and their usage can be found in the reminder of this document. Users can also refer to the examples for using the PM_bissc library, for more details on usage and establishing basic communication with the encoder.

2 PM_bissc Library

2.1 PM_bissc Library Package Contents

The PM_bissc Library consists of the following components:

- Header files and software library for BiSS-C interface
- Documentation – PM_bissc Library User Guide
- Example project showcasing BiSS-C interface implementation on TMDXIDDK379D hardware

Library contents are available at the following location:

<base> install directory is

C:\ti\controlSUITE\libs\app_libs\position_manager\bissc\vX.X

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

3 Module Summary

This section describes the contents of PM_bissc_Include.h – the include file for using PM_bissc library.

3.1 PM_bissc Library Functions

The PM_bissc Library consists of the following functions that enable the user to interface with the encoders. Table 3-1 lists the functions existing in the PM_bissc library and a summary of cycles taken for execution.

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

Table 3-1. Functions and Cycles

Name	Description	CPU Cycles	Type
PM_bissc_generateCRCTable	This function generates 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	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.	120	Run time
PM_bissc_getBits	This function is used for extracting 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 will be in the upcoming SCD transfer. Every BiSS frame ends with a timeout, and during this time no further clocks are transmitted by 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 up and start operation are separate function calls. The user can setup 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	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	Setup a SPI and other modules for a given transaction. All the transactions should start with this command. This function call sets up the peripherals for upcoming BiSS-C transfer but does not actually perform any transfer or activity on the interface. Once the transfer is setup using this function, PM_bissc_startOperation can be called to start the transfer.	742	Run time

(1) Implies the CPU cycle data depends on the encoder under test, and the commands and data being used along with a certain function. These numbers could vary significantly, depending on the command and corresponding data, additional data, and so forth.

3.2 Data Structures

The PM Bissc library defines the BiSS-C data structure handle as below:

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;
```

Module Interface Definition:

Table 3-2. Module Interface Definitions

Module Element Name	Description	Type
cd_status;	Status of the Control Data protocol. 0 – trigger new CD transfer 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 CD data is complete.	int16_t
cd_bits_to_send;	Tells the number of total bits that are in a CD transaction	uint16_t
cdm;	Stores the CD data that will be 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 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 will be a read or write transaction	uint16_t
scd_raw;	Internal variables used by 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 library – for debug purposes	uint32_t
fifo_level	Internal variables used by library – for debug purposes	uint16_t
xfer_address_withCTS	Internal variables used by library – for debug purposes	uint16_t

Table 3-2. Module Interface Definitions (continued)

Module Element Name	Description	Type
spi	SPI instance used for BiSS-C implementation	Pointer to Spi*Regs

3.3 Instructions

3.3.1 Summary of Instructions

3.4 Details of Function Usage

A detailed description of various library functions in PM_bissc library and their usage can be found in the following sections.

PM_bissc_setFreq

Description

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: Freq_us: Clock divider for the system clock sets BiSS-C Clock Frequency = SYSCLK/(4* Freq_us)

Return

None

BiSS-C Clock Frequency = SYSCLK/(4* Freq_us)

Usage

Example Code:

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

PM_bissc_getCRC

Description

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 should be initialized through PM_bissc_generateCRCTable prior to 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

Table 3-3. PM_bissc_getCRC Input

input_crc_accum	Initial CRC value (seed) for CRC calculation
nBitsData	Number of bits of data for which the CRC needs will be calculated
nBitsPoly	Number of bits of polynomial used for CRC computations
msg	pointer to the data on which CRC will be computed
crc_table	Pointer to the table where the CRC table values are stored
rxLen	Number of bytes of the data for CRC calculation

Return

crc – n-bit CRC value calculated

Usage

Define BiSS-C Data structure during initialization.

BISSC_DATA_STRUCT bissc_data_struct;

CRC Computation of Single Cycle Data

```
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

```
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);                                                   // Number
```

PM_bissc_generateCRCTable

Description

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 the pTable.

Definition

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

Parameters

Table 3-4. PM_bissc_generateCRCTable 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

None

CRC Table for SCD (single-cycle data)

```
#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 (control data)

```
#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

Description

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 setup 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);
```

Input

None

Return

None

Usage

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 should subsequently be 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

Description

This function is used for extracting 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

Table 3-5. PM_bissc_getBits Inputs

len	Length of the string to be extracted from the receive data buffer
bitsParsed	Number of bits already parsed i.e. bit index to start extracting bits
charBits	This field indicates the number of valid bits in each entry of the array. This corresponds to the number of bits in each SPI FIFO entry or the receive data buffer.

Return

Extracted bits of the array from specified bit index and length.

Usage

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

Description

This functions configures the library so that it can do a new BiSS-C transaction. This function must be called each time a SCD must occur, and must happen prior to 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);

Parmeters

Table 3-6. PM_bissc_setupNewSCDTransfer Inputs

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
spi_fifo_width	Number of bits in each SPI FIFO entry or the receive data buffer, such as SPI character length

Return

None

Usage

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

Description

This function is used to setup 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 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 CDM bit by the slave.

Definition

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

Parameters

Table 3-7. PM_bissc_setCDBit Inputs

cdmBit	CDM bit to be transmitted in the upcoming BiSS Transaction. This bit is transmitted on the clock line MA.

Return

None

Usage

```
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

Description

Setup for SPI, CLB, and other interconnect XBARs is performed for BiSS-C 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.

Definition

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

Input

None

Return

None

Usage

Example Code:

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

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

4 Using PM_BISSC Library

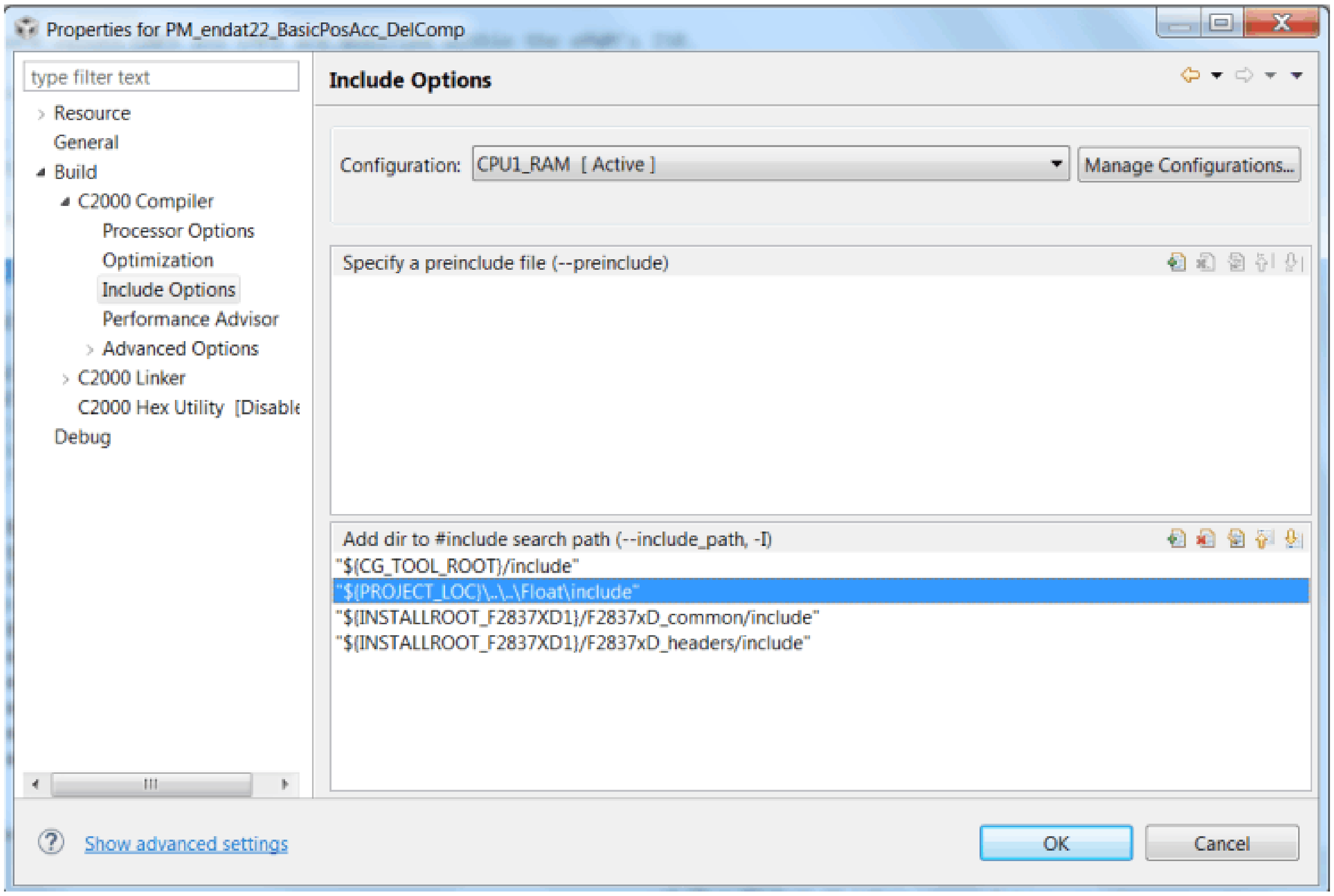
4.1 Adding BiSS-C Lib to the Project

Include library in {ProjectName}-Includes.h.

```
#include "PM_bissc_Include.h"
```

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

Path for the library: *C:\ti\controlSUITE\libs\app_libs\position_manager\v01_00_00_00\bissc\Float\lib*.



Configuration: CPU1_RAM [Active]

Specify a preinclude file (--preinclude)

Add dir to #include search path (--include_path, -I)

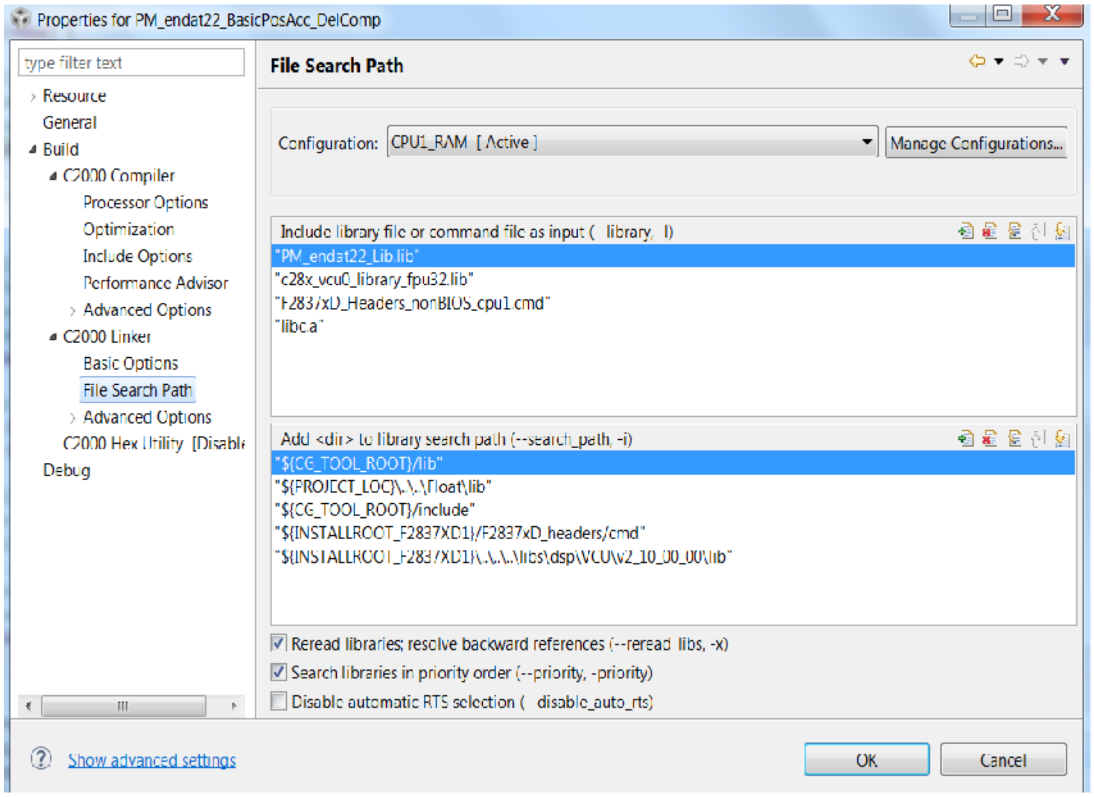
```
"${CG_TOOL_ROOT}/include"
"${PROJECT_LOC}\..\..\Float\include"
"${INSTALLROOT_F2837XD1}/F2837xD_common/include"
"${INSTALLROOT_F2837XD1}/F2837xD_headers/include"
```

Figure 4-1. Compiler Options for a Project Using PM Bissc Library

The exact location may vary, depending on where controlSUITE is installed and which other libraries the project is using.

Link Bissc Library: (PM_bissc_lib.lib) to the project, located at: *controlSUITE\libs\app_libs\position_manager\v01_00_00_00\bissc\Float\lib*.

Figure 4-2 shows the changes to the linker options, which are required to include the BiSS-C library:


Properties for PM_endat22_BasicPosAcc_DelComp

type filter text
- Resource
- General
- Build
  - C2000 Compiler
    - Processor Options
    - Optimization
    - Include Options
    - Performance Advisor
    - Advanced Options
  - C2000 Linker
    - Basic Options
    - File Search Path
    - Advanced Options
  - C2000 Hex Utility [Disable
- Debug

File Search Path

Configuration: CPU1_RAM [Active] — Manage Configurations...

Include library file or command file as input (--library, -l)
"PM_endat22_Lib.lib"
"c28x_vcu0_library_fpu32.lib"
"F2837xD_Headers_nonBIOS_cpu1.cmd"
"libc.a"

Add <dir> to library search path (--search_path, -i)
"${CG_TOOL_ROOT}/lib"
"${PROJECT_LOC}\..\..\float\lib"
"${CG_TOOL_ROOT}/include"
"${INSTALLROOT_F2837XD1}/F2837xD_headers/cmd"
"${INSTALLROOT_F2837XD1}\..\..\libs\dsp\VCU\v2_10_00_00\lib"

[✓] Reread libraries; resolve backward references (--reread_libs, -x)
[✓] Search libraries in priority order (--priority, -priority)
[ ] Disable automatic RTS selection (--disable_auto_rts)

Show advanced settings — OK — Cancel

Figure 4-2. Adding PM_bissc Library to the Linker Options in CCS Project

4.2 Steps for Initialization

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

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

```
BISSC_DATA_STRUCT bissc_data_struct;
```

2. Define the following based on the encoder properties:

```
//If Control data interface is being used this bit should be set to 1 else 0.
#define BISS_ENCODER_HAS_CD_INTERFACE    1
//Number of position bits for the encoder in use
#define BISS_POSITION_BITS        24
//Number of CRC bits for the encoder in use - this is defined as 6 for BiSS-C
#define BISS_POSITION_CRC_BITS 6
//Number of bits of Single Cycle Data Polynomial
#define BISS_SCD_CRC_NBITS_POLY1       6
// Single Cycle Data Polynomial
#define BISS_SCD_CRC_POLY1             0x03   //x^6 + x + 1 (inverted output) 1000011
//Number of bits of Control Data Polynomial
#define BISS_CD_CRC_NBITS_POLY1        4
//Control Data Polynomial
#define BISS_CD_CRC_POLY1              0x03   //x^4 + x + 1 (inverted output) 10011
//Size of CRC tables for Control Data and Single Cycle Data - should be set to 256
#define BISS_SCD_CRC_SIZEOF_TABLE      256
#define BISS_CD_CRC_SIZEOF_TABLE       256
```

3. Define and run the frequency of the BiSS-C clock during initialization.

```
#define    BISSC_FREQ_DIVIDER 25
//BiSS Clock Frequency = SYSCLK/(4*BISSC_FREQ_DIVIDER)
//Set BISSC_FREQ_DIVIDER accordingly. Only even values greater than 6 are supported.
```

4. Set the SPI instance to be used for communication. For usage on TMDXIDDK379D, the SPI instance must be set to SpiB and the SpiB receive fifo interrupt must be enabled.

```
    bissc_data_struct.spi = &SpibRegs;
    PieVectTable.SPIB_RX_INT = &bissc_spiRxFifoIsr;
    PieCtrlRegs.PIECTRL.bit.ENPIE = 1;     // Enable the PIE block
    PieCtrlRegs.PIEIER6.bit.INTx3 = 1;     // Enable PIE Group 6, INT 9
    IER  |= M_INT6;                        // Enable CPU INT6
    EINT;
```

Alternatively, users can also poll for the interrupt flag and not necessarily use an interrupt. Copy the SPIRXFIFO contents into bissc_data_struct.rdata after the flag is set. This must be executed before calling the PM_bissc_getBits function. Also, the interrupt flag must be cleared to get the SPI ready for the next BiSS-C transaction.

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

5. Set SPI character width:

```
#define    SPI_FIFO_WIDTH 12
//Use optimal character length based on the encoder and number of bits needed for transfer.
```

6. Enable clocks to EPWM Instances 3 and 4:

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

7. Initialize and setup the peripheral configuration by calling the *PM_bissc_setupPeriph* function:

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

8. Setup the GPIOs needed for the BiSS-C operation (required for TMDXIDDK379D). The GPIOs used for SPI must be changed based on the chosen SPI instance. GPIO6, GPIO7, and GPIO34 must always be configured.

```
    GpioCtrlRegs.GPAMUX1.bit.GPIO6 = 1; // Configure GPIO6 as bissC Clk master
    GpioCtrlRegs.GPAMUX1.bit.GPIO7 = 1; // Configure GPIO7 as SPI Clk slave
    GpioCtrlRegs.GPAGMUX2.bit.GPIO24 = 1;
    GpioCtrlRegs.GPAGMUX2.bit.GPIO25 = 1;
    GpioCtrlRegs.GPAGMUX2.bit.GPIO26 = 1;
    GpioCtrlRegs.GPAGMUX2.bit.GPIO27 = 1;
    GpioCtrlRegs.GPAMUX2.bit.GPIO24 = 2; // Configure GPIO24 as SPISIMOB
    GpioCtrlRegs.GPAMUX2.bit.GPIO25 = 2; // Configure GPIO25 as SPISOMIB
    GpioCtrlRegs.GPAMUX2.bit.GPIO26 = 2; // Configure GPIO26 as SPICLKB
    GpioCtrlRegs.GPAMUX2.bit.GPIO27 = 2; // Configure GPIO27 as SPISTEB
    GpioCtrlRegs.GPAQSEL2.bit.GPIO24 = 3; // Asynch input GPIO24 (SPISIMOB)
    GpioCtrlRegs.GPAQSEL2.bit.GPIO25 = 3; // Asynch input GPIO25 (SPISOMIB)
    GpioCtrlRegs.GPAQSEL2.bit.GPIO26 = 3; // Asynch input GPIO26 (SPICLKB)
    GpioCtrlRegs.GPAQSEL2.bit.GPIO27 = 3; // Asynch input GPIO27 (SPISTEB)
    GpioCtrlRegs.GPBMUX1.bit.GPIO34 = 0; // Configure GPIO34 as bissC TxEN - drive low
    GpioCtrlRegs.GPBDIR.bit.GPIO32 = 1; // Configure GPIO32 as bissC Pwr Ctl
```

9. Setup XBAR as shown below. The GPIOs used for SPI must be changed based on the chosen SPI instance.

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

10. Initialization for CRC-related object and table generation:

```
    // Generate CRC tables for BiSS-C as defined in bissc.h
    PM_bissc_generateCRCTable(BISS_SCD_CRC_NBITS_POLY1, BISS_SCD_CRC_POLY1, bissCRCtableSCD);
    #if BISS_ENCODER_HAS_CD_INTERFACE
    PM_bissc_generateCRCTable(BISS_CD_CRC_NBITS_POLY1, BISS_CD_CRC_POLY1, bissCRCtableCD);
    #endif
```

11. Turn the power ON. The GPIO used for power control can be changed based on the hardware. GPIO32 is used for power control on TMDXIDDK379D.

```
    // Power up Biss-C 15v supply through GPIO32
    GpioDataRegs.GPBDAT.bit.GPIO32 = 1;
```

4.3 Resource Requirements

The following resources of the MCU are consumed by the PM_bissc Library in implementing the BiSS-C interface.

Table 4-1. Resource Requirements

Resource Name	Type	Purpose	Usage Restrictions
Dedicated Resources			
GPIO6	IO	Clock from master to encoder	IO dedicated for Biss-C
GPIO7	IO	SPI clock generated by MCU	IO dedicated for Biss-C
EPWM4	IO	Internally for clock generation	EPWM4 dedicated for Biss-C
Input XBAR (NPUTXBAR1)	Module/IO	Connected to SPISIMO of the corresponding SPI instance dedicated for Biss-C	
Configurable Resources			
SPI	Module/IO	One SPI instance to emulate Biss-C interface (SPIB on IDDK)	Any instance of SPI can be chosen – Module and corresponding IOs will then be dedicated for Biss-C
Biss-PwrCtl	IO	For power control	Can choose any IO power control. GPIO32 is used on IDDK and example projects
Biss-TxEn	IO	For Transmit Enable control of RS485	Can choose any IO. GPIO34 is used on IDDK and example projects
Shared Resources			
CPU and Memory	Module	Check CPU and Memory utilization for various functions	Application to ensure enough CPU cycles and memory are allocated

5 Test Report

The following tests, using various types of encoders and cable lengths, have been performed at Texas Instruments' laboratories. Tests include basic command set exercising and reading position values with additional data, if applicable.

Table 5-1. Test Report

Encoder Manufacturer	Encoder Name	Type	Resolution	Cable Length	Max BiSS Clock	CD Interface	Test Result
Lika	HS58S18/I7	Rotary	18-bits (padded to 24bits)	100m	3.33 MHz	Yes	Pass
Kuebler	8.F5863. 1426.C423	Rotary	26-bits (12+14)	100m	5 MHz	No	Pass

6 FAQ

1. What BiSS-C protocol options are supported using the PM_bissc library?
   - Fully digital BiSS-C interface support
   - Point-to-point communication only is supported
   - Single-Cycle Data (SCD) and Control Data (CD) operations supported. Only the CD register communication portion is implemented.
   - Software functions for checking data integrity (CRC)

   For further help, see the BiSS-C documentation at http://www.biss-interface.com/.
2. What are the limitations of the BiSS-C implementation with this library?

   Refer to Section 1.3 while using PM_bissc library.

3. How is the BiSS-C interface implemented on TMS320F28379D devices?

   Refer to Section 1.3.
4. Does TI share the source for the PM_bissc library to customers?

   TI does not share this source code with customers. For any specific requests, contact the TI sales team.
5. Does TI provide application-level interface functions for BiSS-C?

   Basic usage examples are provided along with the library. The example has higher level application layer functions for initialization, setting and reading parameters, SCD and CD transfers, checking CRC for various types of received data, and so forth. This should be sufficient for most applications. Any additional application layer functionality should be developed by users using the basic driver interface functions provided in the PM_bissc Library.

7 References

- Power Supply Reference design for BiSS-C encoder interfaces can be obtained from Texas Instruments (TIDA-00175)
- Documentation from http://www.biss-interface.com/
- C2000 DesignDRIVE Development Kit for Industrial Motor Control - TMDXIDDK379D:
  – *DesignDRIVE Development Kit IDDK Hardware Reference Guide* (SPRUI23)
  – *DesignDRIVE Development Kit IDDK User's Guide* (SPRUI24)