

Design Guide: TIDM-1011

Tamagawa T-Format Absolute-Encoder Master Interface Reference Design for C2000™ MCUs



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 circuits (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. The software of this reference design specifically targets implementation of the T-Format, which is a digital, bidirectional interface for position encoders. The highly optimized and easy-to-use software reference implementation and examples included in this reference design enable T-Format, position-encoder operation using the Position Manager BoosterPack.

Resources

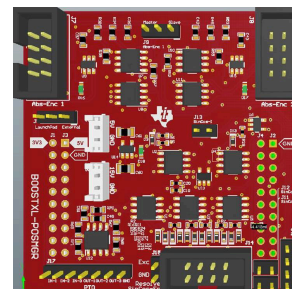
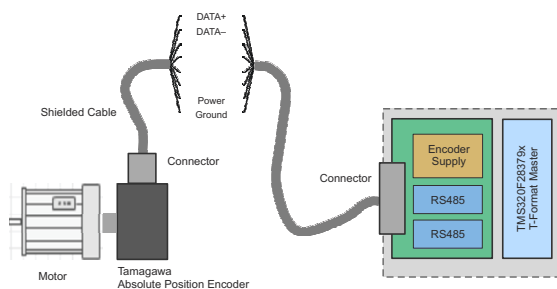
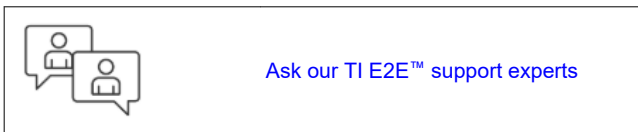
TIDM-1011	Design Folder
LAUNCHXL-F28379D	Tools Folder
SN65HVD78, TLV702, TPS22918-Q1	Product Folder

Features

- Flexible, low-voltage, boosterPack evaluation platform for position-encoder interfaces
- Integrated MCU solution for T-Format without additional FPGA requirements
- Easy interface T-Format commands through driver functions and data structure provided by interface function.
- Support for unpacking received data and optimized cyclic redundancy check (CRC) algorithm
- Supports clock frequency up to 2.5 MBPS and verified operation up to 100-m cable length
- Includes evaluation software example showcasing T-Format implementation

Applications

- [Industrial](#)
- [Motor Drives](#)



1 System Description

Industrial drives, like servo drives, require accurate, highly-reliable, and low-latency position feedback. T-Format, from Tamagawa, is designed for serial transfer of digital data between linear, rotary, or angle encoders, touch probes, accelerometers, and the subsequent electronics, such as numerical controls, servo amplifiers, and programmable-logic controllers. T-Format 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 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 T-Format master, send to the encoder select the transmitted-data types. The TIDM-1011 device acts as a T-Format master and provides the subsequent electronics to interface a T-Format encoder with the F28379D LaunchPad.

Figure 1-1 shows a T-format encoder interfaced to a F28379D LaunchPad.

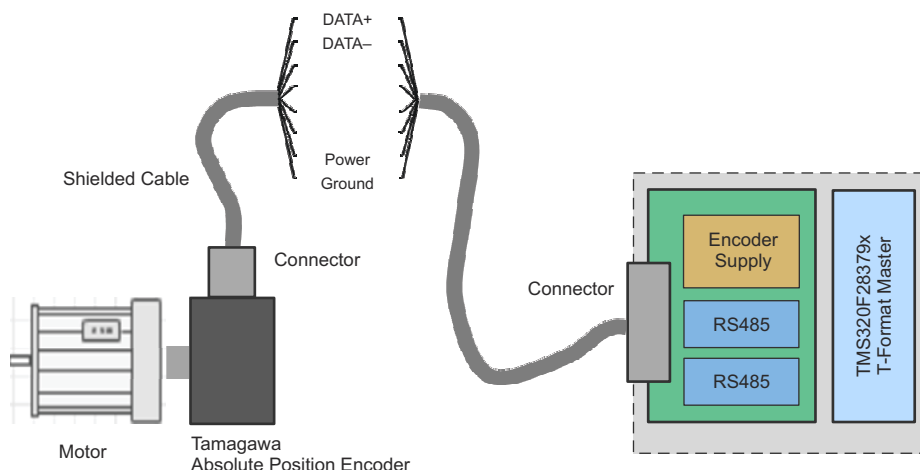


Figure 1-1. Industrial Servo Drive With T-Format Position Encoder Interface

The position encoder with T-Format connects to the TIDM-1011 device through a single, 4-wire, shielded cable. The four wires used for communication include two wires for DATA+ and DATA–, which are transmitted in differential format, and two wires for Up and Un, which are used for the encoder-power supply and ground.

The C2000, Position Manager, T-Format (PM_tformat) interface implementation from TI provides support for implementing the T-Format interface in subsequent electronics. The implementation comprises the software portion of the TIDM-1011 device. The T-Format implementation features an integrated MCU solution for the T-Format interface that meets the Tamagawa, T-Format, digital-interface protocol requirements. The reference implementation can support up to 2.5-MBPS clock frequency, independent of cable length – verified up to 100 m. This support is due to the integrated cable-propagation, delay-compensation algorithm, which is user configurable. The driver functions and data structure provided by the reference implementation allow other commands to easily be used. The reference implementation 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 functions. This reference implementation 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 T-Format reference implementation. This is a reference implementation and all the source code is made available to the user. Any changes needed for the implementation can be made by users as needed by their application. The implementation only supports the basic interface drivers for commands defined in the T-Format specification. All the higher-level application software must be developed by users using the basic interface provided by this implementation. Clock frequency for the T-Format Clock is limited to a fixed value of 2.5 MBPS. 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

Table 1-1. 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	T-Format	Tamagawa
Frequency (encoder interface)	Approximately 2.5 MBPS	Section 2.3.2
Encoder bits	T-Format protocol standard	Tamagawa
CPU cycles	—	Section 2.3.3.2

(1) The time of the encoder connected to the TIDM-1011 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

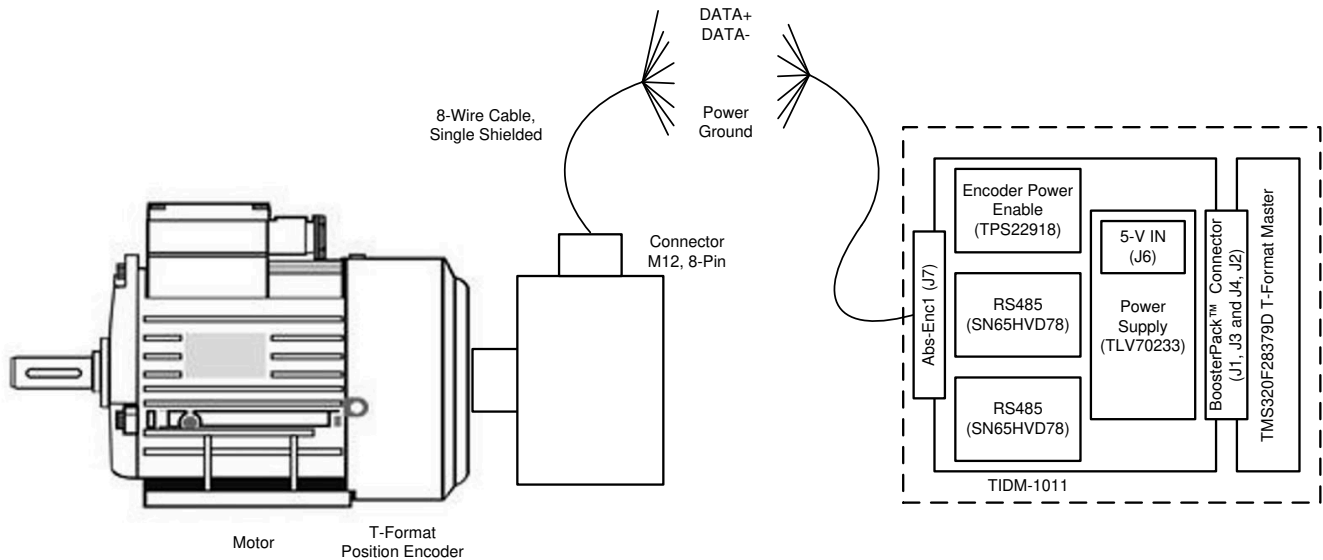


Figure 2-1. TIDM-1011 System Block Diagram

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 differential outputs of the driver and the differential inputs of the receiver 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 makes 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-1011 Board Implementation

The TIDM-1011 board is identical to the Position Manager BoosterPack (BOOSTXL-POSMGR), which means the TIDM-1011 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 T-Format, and the hardware blocks not mentioned can be ignored. Software support for the other types of position-encoder interfaces will be the subject of future reference designs. [Table 2-1](#) lists the connectors on the TIDM-1011 and BOOSTXL-POSMGR devices and their functions.

Table 2-1. TIDM-1011 Board and BOOSTXL-POSMGR Connectors

CONNECTOR	DESCRIPTION	RELEVANT TI DESIGNS AND HARDWARE
Abs-Enc-1 (J7)	T-Format and other absolute encoders	TIDM-1011, BOOSTXL-POSMGR
Abs-Enc-2 (J8)	T-Format 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
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

[Figure 2-2](#) shows the encoder support on each site of the LaunchPad.

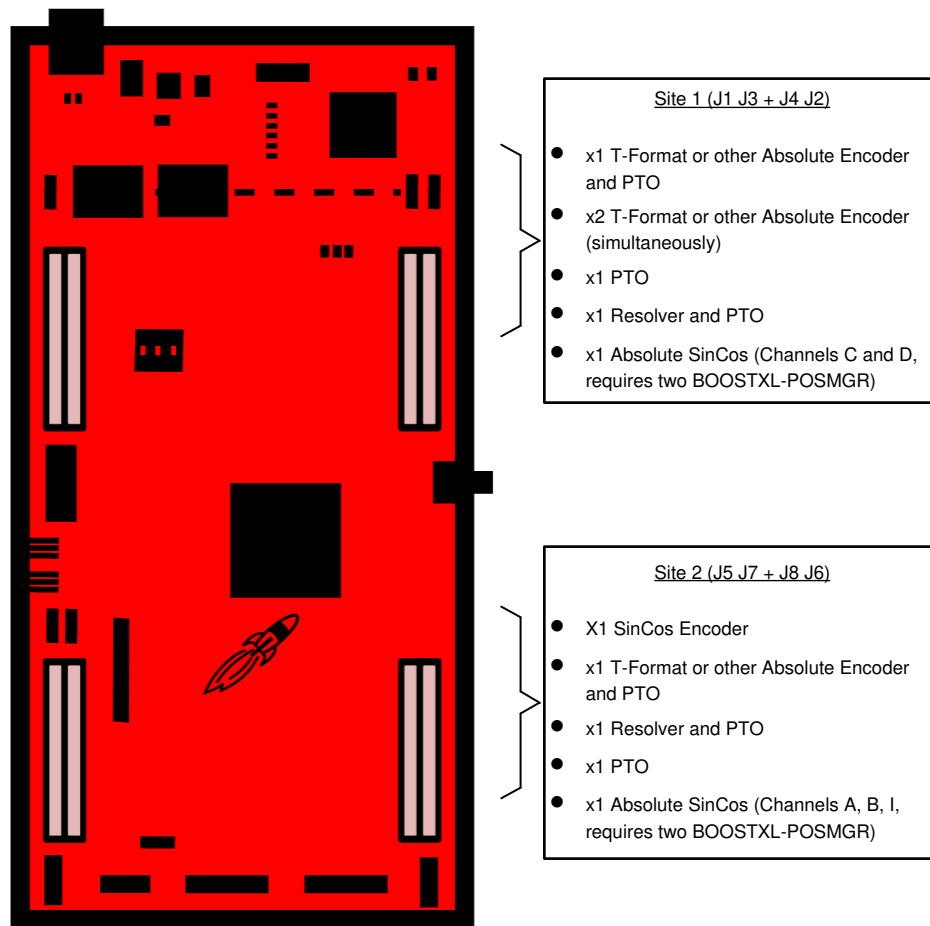


Figure 2-2. TIDM-1011 Board and BOOSTXL-POSMGR Encoder Support

2.3.2 PM T-Format Master Details

This section provides a brief overview of how the T-Format interface is implemented on TMS320F28379D devices. By design, the TIDM-1011 device works with multiple, C2000, LaunchPad development kits. This reference design focuses on the F28379D LaunchPad as the main example.

Communication over a T-Format 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 functions provided in the PM T-Format reference implementation and is not otherwise configurable by users. The following functions are implemented inside the CLB module:

- 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

Figure 2-3 shows how the T-Format transaction works in the system.

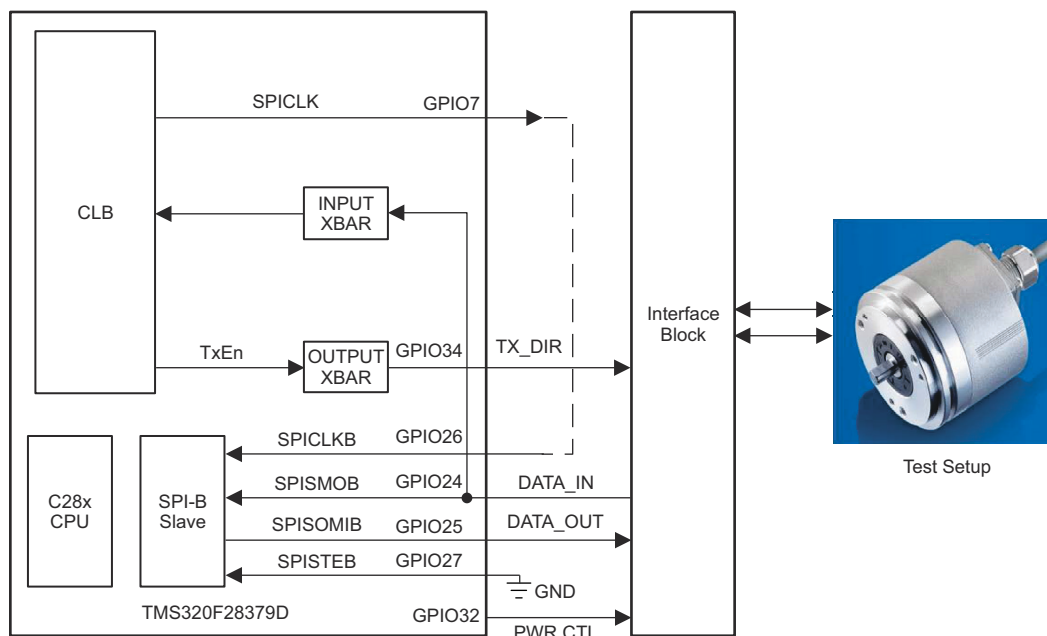


Figure 2-3. T-Format Implementation Diagram Inside TMS320F28379D

For every T-Format transaction initiated using the PM T-Format reference implementation command the following occurs.

- 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 T-Format command.
- The CPU sets up the CLB to generate clocks for the encoder and SPI.
- The number of clock pulses and edge placement for these two clocks is different and precisely controlled by the CLB, as configured by CPU software for the current T-Format command.
- The 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.
- The CLB monitors the SPISIMO signal (as necessary), which detects the start pulse and adjusts the phase of the receive clock accordingly.
- The CPU configures the CLB to generate continuous clocking for the encoder while waiting for the start pulse from the encoder.
- The CPU configures the CLB to generate a predefined number of clock pulses needed for the SPI (as per the current command requirements), and continuous clocking for the SPI is disabled while waiting for the start pulse from the encoder.
- The CLB also provides hooks to perform cable propagation-delay compensation using the functions in the reference implementation.

2.3.2.1 Implementation Details

T-Format reference implementation source files are available under:

<C2000WARE-MOTORCONTROL-SDK>\libraries\position_sensing\tformat\source

The project configuration files are available at:

<C2000WARE-MOTORCONTROL-SDK>\libraries\position_sensing\tformat\ccs\2837x

Source files can be edited and the project can be recompiled as needed. Compiling this project would create a .lib which is located under:

<C2000WARE-MOTORCONTROL-SDK>\libraries\position_sensing\tformat\lib

The generated library can be linked into the application project.

The following resources are used inside the CLB tile to achieve the desired function:

Figure 2-4 shows all CLB blocks used in this example, the associated logic equations and connections between these blocks.

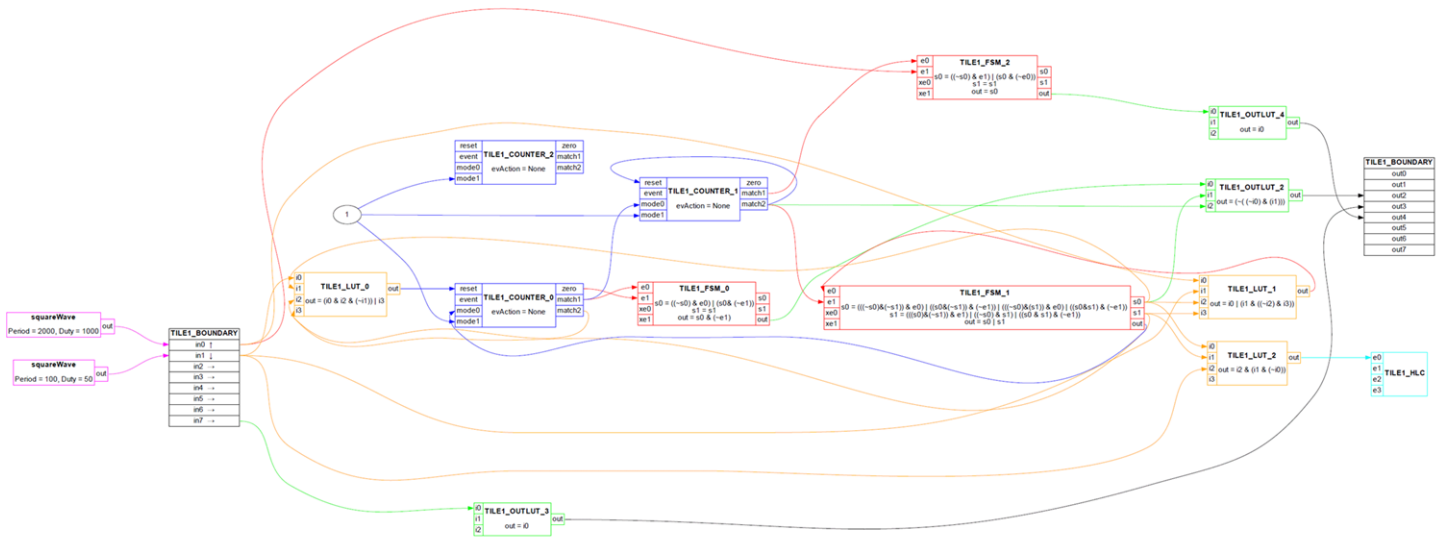


Figure 2-4. CLB Implementation Diagram

Figure 2-5, Figure 2-6, Figure 2-7, Figure 2-8, Figure 2-9, and Figure 2-10 show the same information as Figure 2-4 but through a logic schematic lens. Specifically, Figure 2-5 and Figure 2-6 show the contents of the CLB blocks using logic gates. Next, Figure 2-7 uses this logic to show main state machine controlling other blocks. Figure 2-9 traces a couple of simple CLB output signals starting from their inputs and passing through some associated logic. Figure 2-9 traces the Clock to SPI output starting from Input1 and passing through LUT_0, FSM_0, Counter_0 all the way to the Ouput_LUT_0 as controlled by 3 outputs from FSM_1. Figure 2-10 traces the Transmit Enable output starting from Input0 and Input1, and passing through LUT_0, Counter_0, Counter1 and FSM_2 as controlled by 3 outputs from FSM_1.

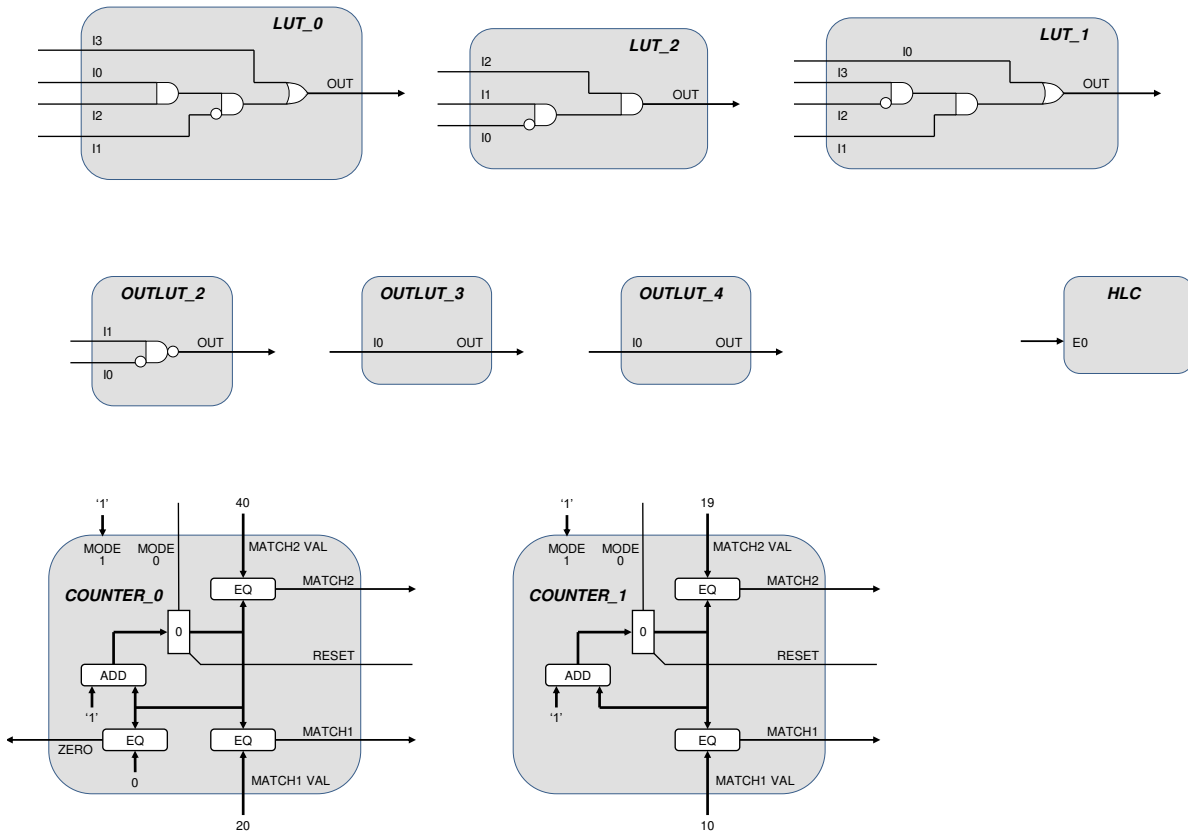


Figure 2-5. LUTs, OUTLUTs, and Counters

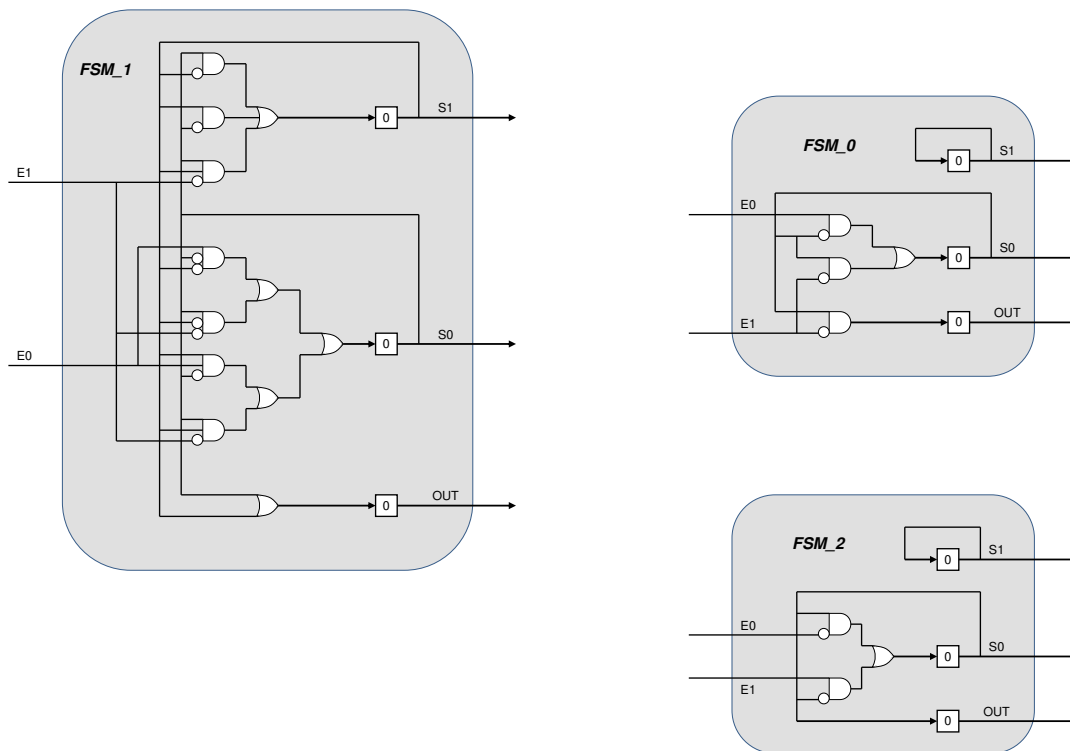


Figure 2-6. Finite State Machines

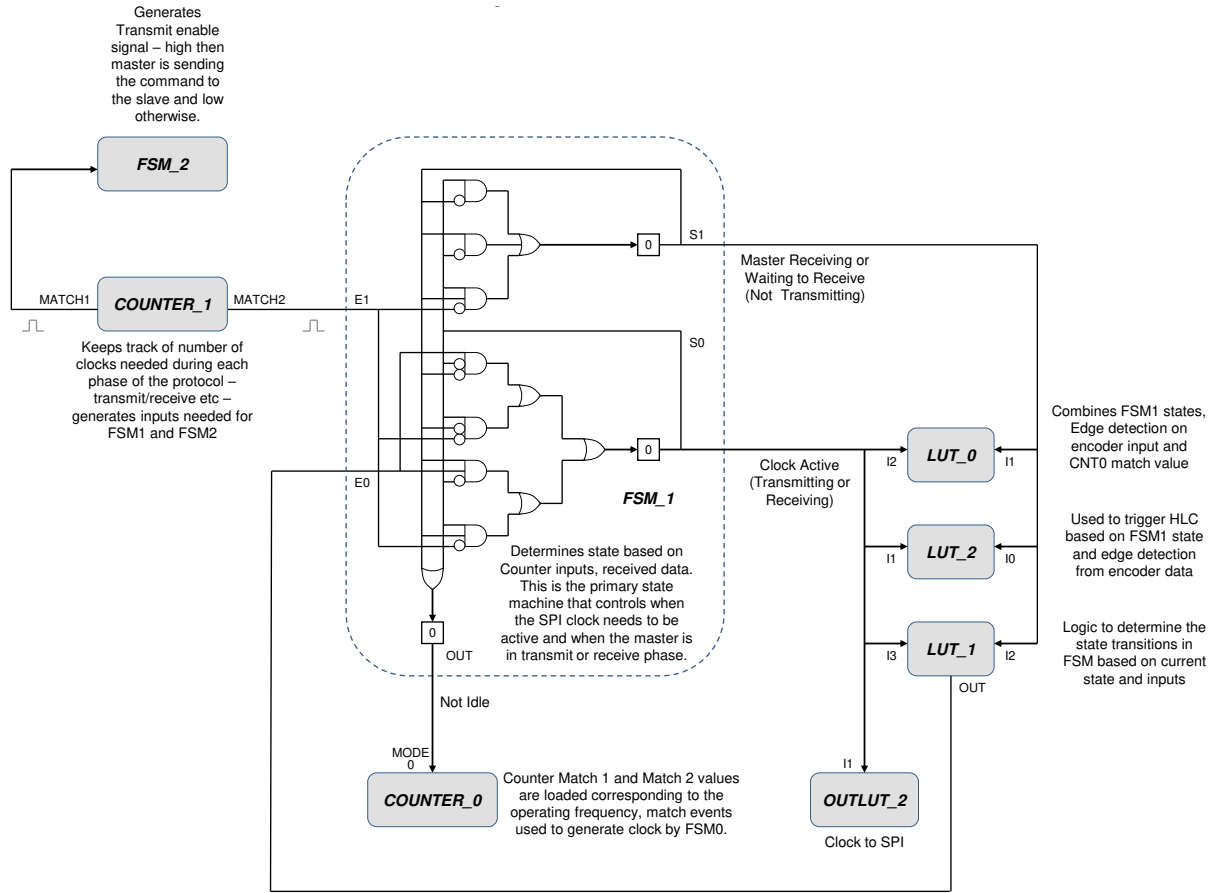


Figure 2-7. The Main State Machine

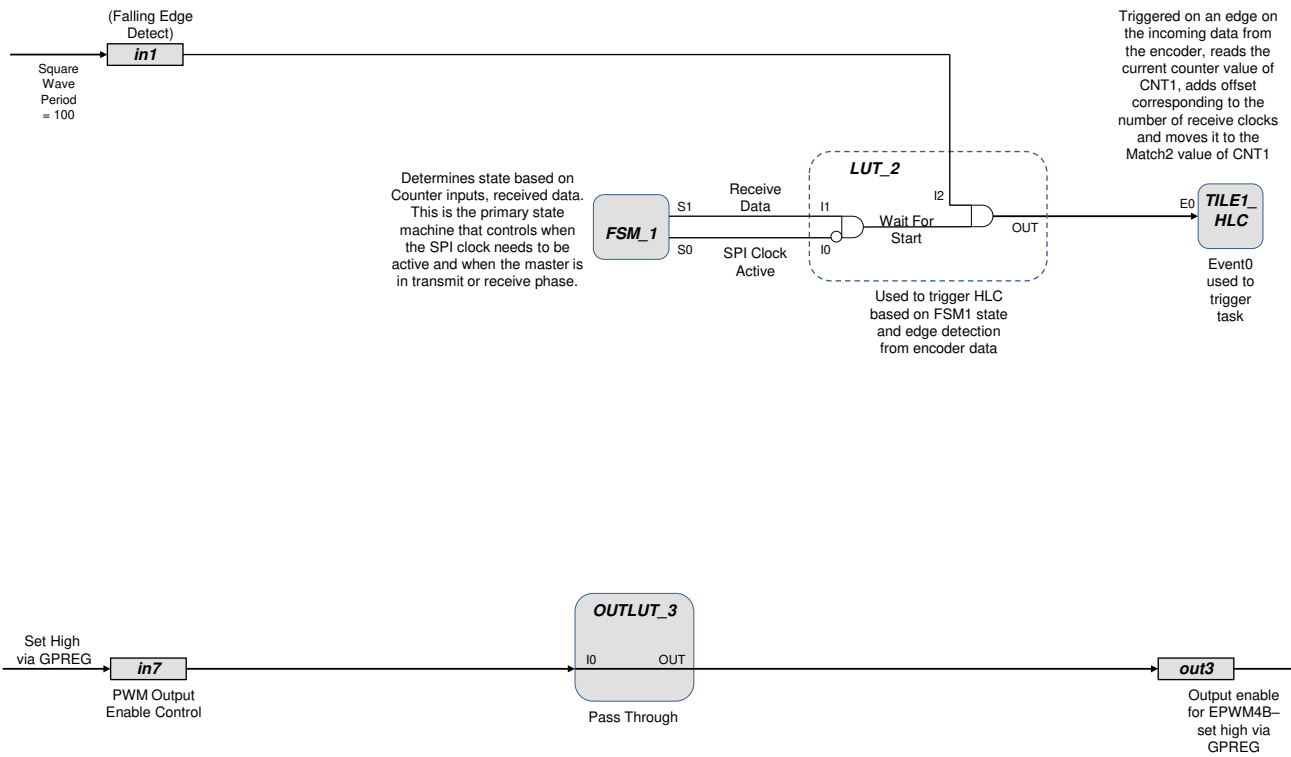


Figure 2-8. CLB Outputs – HLC Event0 and EPWM Output Enable

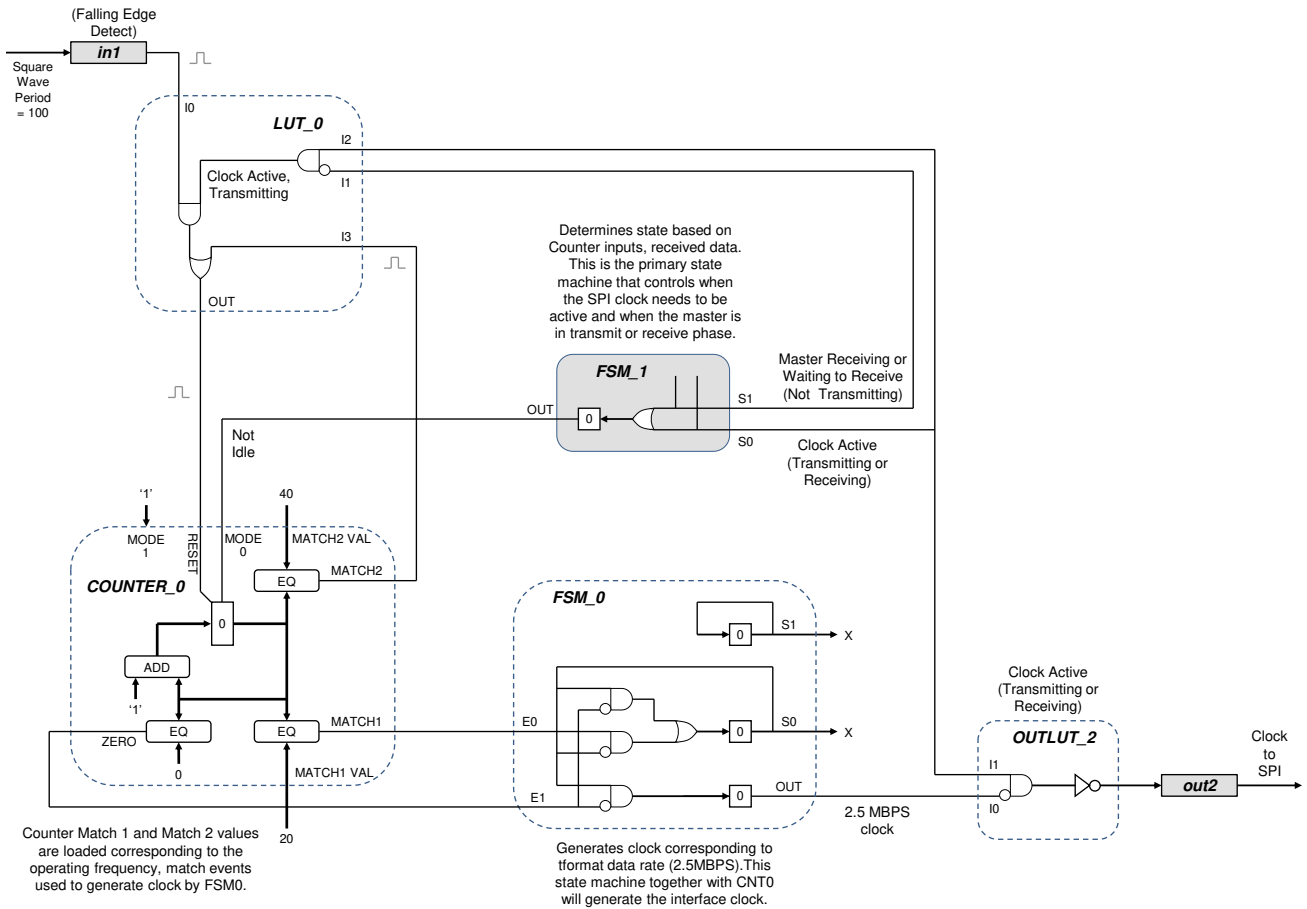


Figure 2-9. CLB Outputs – Clock to SPI

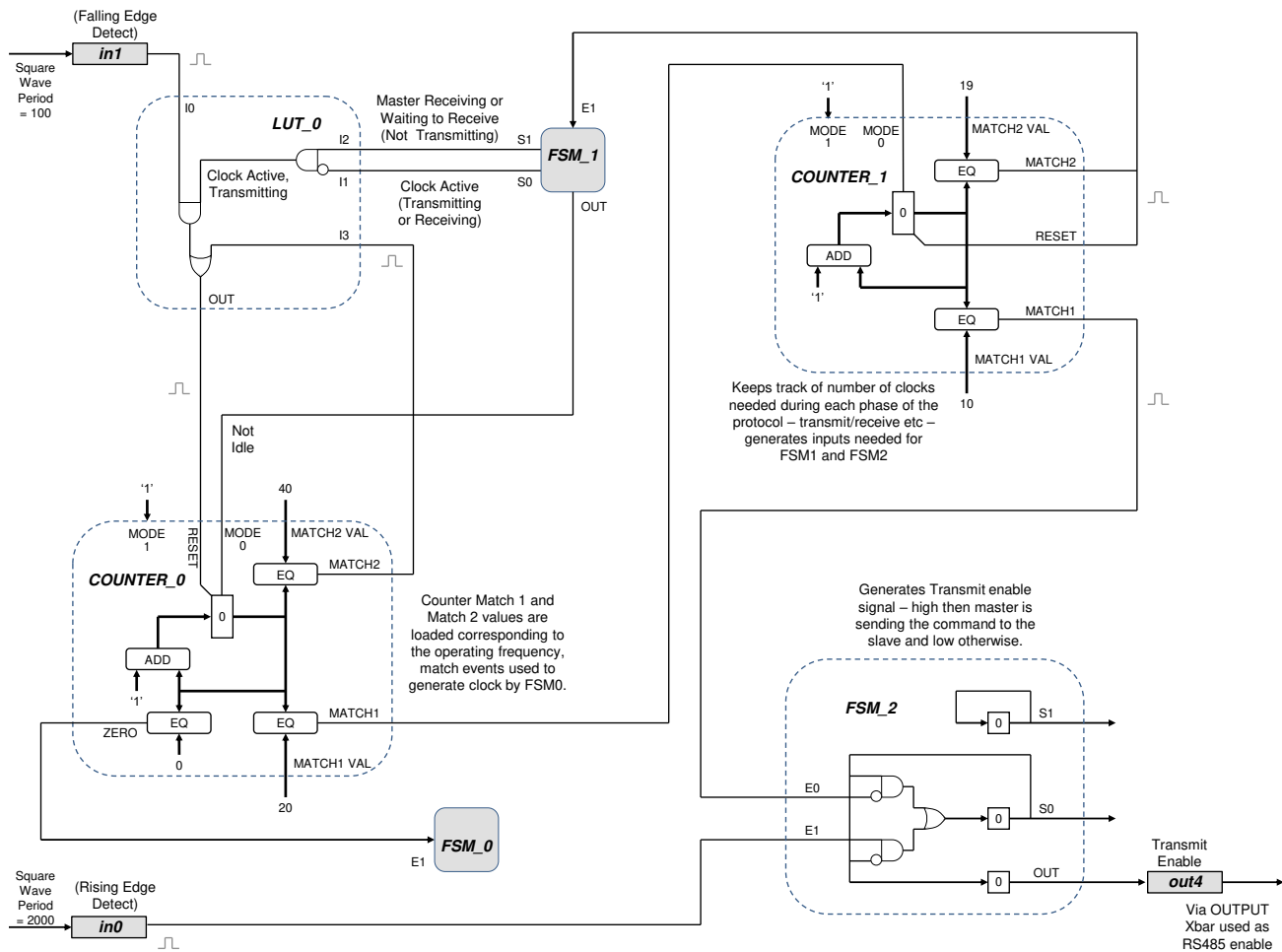


Figure 2-10. CLB Outputs – RS485 Enable

RESOURCE	FUNCTION	NOTES
INPUTS		
In0	On/Off Control via GPREG	
In1	Falling Edge Detect	SPISIMO Via Input and CLB XBar
In2	Not used	
In3	Not used	
In4	Not used	
In5	Not used	
In6	Not used	
In7	Set High via GPREG	PWM Output Enable Control
OUTPUTS		
Out0	Not used	Not used
Out1	Not used	Not used
Out2	Clock to SPI	GPIO7 via EPWM4B
Out3	Connected to In7	Output enable for EPWM4B – set high via GPREG
Out4	Transmit Enable	Via OUTPUT XBarUsed as RS485 enable
Out5	Not used	Not used
Out6	Not used	Not used

RESOURCE	FUNCTION	NOTES
INPUTS		
Out7	Not used	Not used
LOGIC RESOURCES		
LUT0	Reset Signal for CNT0	Combines FSM1 states, Edge detection on encoder input and CNT0 match value
LUT1	State transition input for FSM1	Logic to determine the state transitions in FSM based on current state and inputs
LUT2	HLC event input	Used to trigger HLC based on FSM1 state and edge detection from encoder data
FSM0	Clock generator	Generates clock corresponding to tformat data rate (2.5MBPS). This state machine together with CNT0 will generate the interface clock.
FSM1	Primary state machine	Determines state based on Counter inputs, received data. This is the primary state machine that controls when the SPI clock needs to be active and when the master is in transmit or receive phase.
FSM2	TxEN control	Generates Transmit enable signal – high then master is sending the command to the slave and low otherwise.
CNT0	Clock generation	Counter Match1 and Match2 values are loaded corresponding to the operating frequency, match events used to generate clock by FSM0.
CNT1	Transaction clocks	Keeps track of number of clocks needed during each phase of the protocol – transmit/ receive etc – generates inputs needed for FSM1 and FSM2.
CNT2	Not used	Not used
High Level Controller		
HLC	Event0 used to trigger task	Triggered on an edge on the incoming data from the encoder, reads the current counter value of CNT1, adds offset corresponding to the number of receive clocks and moves it to the Match2 value of CNT1

State transitions occur as per the timing shown in [Figure 2-11](#). Initially, FSM1 is in idle state. When the `PM_startOperation()` command is issued by the master (via write to `GPREG[0]`), FSM1 moves to transmit the data phase. In this phase, CLB generates the clocks corresponding to the transmit bits needed by the encoder to receive the `DATAID` command along with start and stop bits.

The data being transmitted by this command is initialized by the `PM_tformat_setupCommand` function. Data that must be transmitted is written into the SPI TXFIFO upfront before calling the start operation function. During this phase, TxEN (transmit enable) for the RS485 transceiver data line is held high, indicating that the data transfer is from the master.

Once the data is transmitted from the master, TxEN is held low giving the control of the data line to the slave, for example, the encoder. The master then waits for the encoder to respond with data. This phase is denoted as the *wait for start* phase in [Figure 2-4](#). This stage can be any number of clock cycles depending on the state of encoder, cable length, and so forth. Once the start bit is received from the encoder, the receiver circuitry is activated, and the state machine then moves to receive the data state. In this state, CLB generates a number of clock cycles corresponding to the data to be sent by encoder – which in turn is dependent on the command transmitted. Once the number of SPI CLKs generated matches the data bits transmitted by encoder, the transaction is complete and FSM1 moves back to IDLE state waiting for next operation to be triggered.

Sometimes an additional number of clocks are generated to match the SPI fifo buffer requirements, depending on the command, even though the transaction itself may not require those additional clocks. A SPI interrupt is generated when the fifo level is reached and the received data is available in the SPI RXFIFO and PM_tformat_receiveData when invoked with the appropriate command would unpack the fifo buffer and populate the data structure tformatData with the received data.

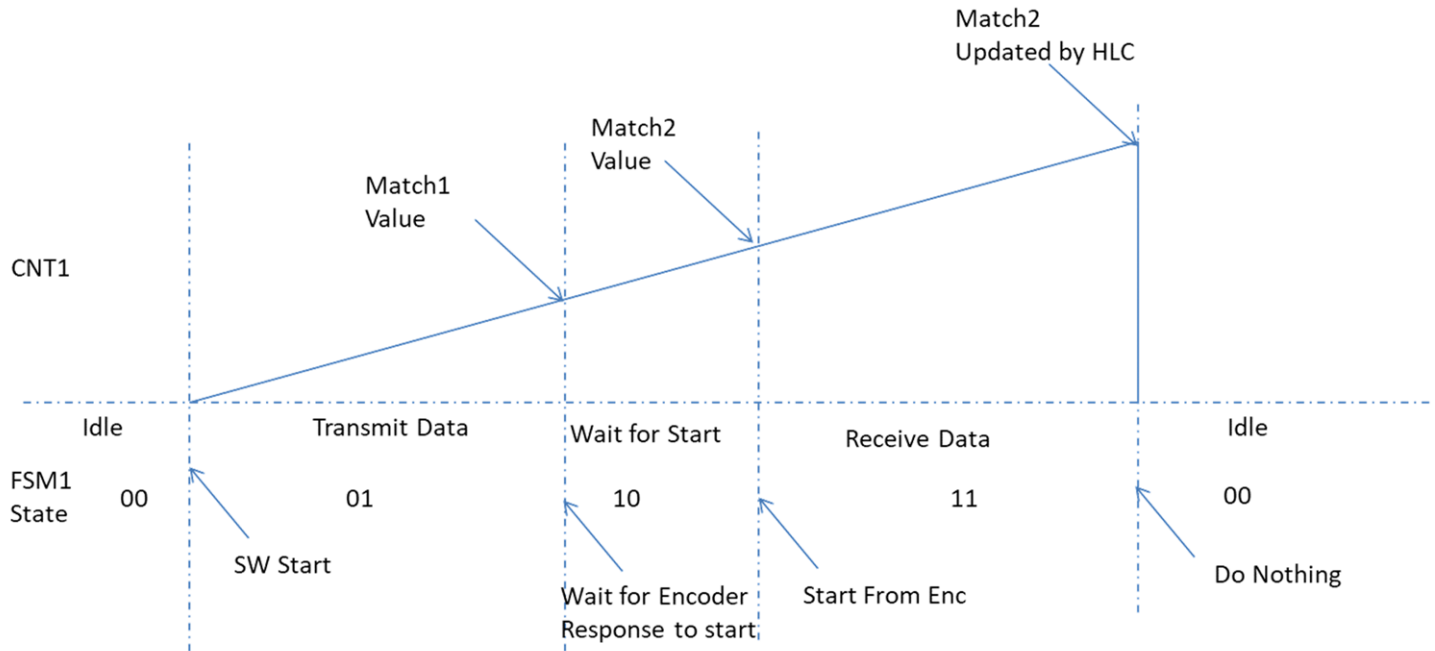


Figure 2-11. Signal Timing and State Transitions

Table 2-2 lists the full MCU resource use.

Table 2-2. TIDM-1011 MCU Resource Requirements

RESOURCE NAME	TYPE	PURPOSE	USE RESTRICTIONS
RESOURCES (Can be changed in reference implementation)			
GPIO7	I/O	SPI clock generated by the MCU	I/O dedicated for T-Format implementation
EPWM4	I/O	Internal for clock generation	EPWM4 dedicated for T-Format implementation
Input XBAR (INPUTXBAR1)	Module, I/O	Connected to SPISIMO of the corresponding SPI instance	INPUTXBAR1 is used for implementation. The remaining inputs are available for application use.
Output XBAR (OUTPUTXBAR6)	Module, I/O	Bring out T-Format TxEn (direction control) signal on GPIO9 using OUTPUT6 of Output XBAR	OUTPUT6 is used for implementation. The remaining outputs are available for application use.
CONFIGURABLE RESOURCES (Can be changed in application project - refer to example)			
SPI	Module and I/Os	One SPI instance to emulate T-Format interface (SPIB on LaunchPad)	Any instance of SPI can be chosen – module and corresponding I/Os are dedicated for T-Format
PwrCtl	I/O	For encoder power control	Can choose any I/O power control. GPIO139 is used for T-Format implementation and example projects.
TxEn	I/O	Direction control on data	GPIO9 is used for T-Format implementation and example projects. Users can choose any GPIO with an OUTPUTXBAR6 mux option.
SHARED RESOURCES			
CPU and Memory	Module	Check CPU and memory use for various functions	Application to ensure enough CPU cycles and memory are allocated

2.3.2.2 Configuration of Inputs to CLB

This configuration's details can be found in the source file PM_tformat_source.c. This file, in addition to configuring CLB, also configures the input and output selection.

2.3.2.2.1 Input Selection

IN0 Selection: Activated as GPREG

```
CLB_configGlobalInputMux(CLB4_BASE, CLB_IN0,
CLB_GLOBAL_IN_MUX_EPWM1A);CLB_configGPIInputMux(CLB4_BASE, CLB_IN0, CLB_GP_IN_MUX_GP_REG);
```

Rising edge detect turned ON:

```
CLB_selectInputFilter(CLB4_BASE, CLB_IN0, CLB_FILTER_RISING_EDGE);
```

IN1 Selection: Activated as CLB X-Bar AUXSIG0

```
CLB_configGlobalInputMux(CLB4_BASE, CLB_IN1, CLB_GLOBAL_IN_MUX_CLB_AUXSIG0);
;;;Configuration of CLB X-Bar AUXSIG0->INPUTXBAR1 XBAR_setCLBMuxConfig(XBAR_AUXSIG0,
XBAR_CLB_MUX01_INPUTXBAR1); XBAR_enableCLBMux(XBAR_AUXSIG0, XBAR_MUX01);
```

2.3.2.2.2 Activating Outputs

Enabling the Output enable to activate CLB output:

```
CLB_setOutputEnableMask(CLB4_BASE, 0x3C);
```

2.3.2.3 Function Description

2.3.2.3.1 tformat_initCLB4

Initializes the CLB instance 4 with the code generated from SysConfig configuration file clb_config.h.

2.3.2.3.2 tformat_initCLBXBAR

Initializes the CLB X-Bar for input and output configuration.

2.3.2.3.3 tformat_initSPIFIFO

Initializes the SPI FIFO for transmit and receive data to and from the encoder – SPI is configured in Slave mode and clock is generated by CLB for SPI operation.

2.3.2.3.4 tformat_resetCLB

Used to reset CLB initially before setting up the peripheral for tformat operation – clears any residual counter values, toggles the global enable to clear the states of FSMs, and so forth.

2.3.2.3.5 tformat_configureSPILen

This function configures the SPI FIFO level and word length. This is set for every command based on the transmit data and receive data lengths that are corresponding to the command being issued to the encoder.

All other functions (PM_*) are described in subsequent sections in detail.

2.3.3 PM T-Format Interface Example

The PM T-Format implementation provides a host of commands and functions for interfacing C2000 devices with T-Format position encoders. This section provides some documentation on the implementation, example and describes the commands and functions the provided. If the latest version of C2000WARE-MOTORCONTROL-SDK (MotorControl software development kit (SDK) for C2000™ MCUs) is installed, the source is in the following directory:

```
CLB based reference implementation is located at:
<C2000WARE-MOTORCONTROL-SDK>\libraries\position_sensing\tformat
Example implementation using BOOSTXL_POSMGR + F28379D LaunchPad development kit (LAUNCHXL-F28379D)
is located at:
<C2000WARE-MOTORCONTROL-SDK>\solutions\boostxl_posmgr\f2837xd\ccs\tformat
```

Software delivered on C2000WARE-MOTORCONTROL-SDK for the TIDM-1011 device uses the previously mentioned hardware resources. The Position Manager BoosterPack is expected to be plugged onto site 2, as shown in [Figure 3-7](#).

The following subdirectory structure is used:

```
Documentation:
<C2000WARE-MOTORCONTROL-SDK>\solutions\boostxl_posmgr\Docs
Example project files:
<C2000WARE-MOTORCONTROL-SDK>\solutions\boostxl_posmgr\f2837xd\ccs\tformat
Example source files:
<C2000WARE-MOTORCONTROL-SDK>\solutions\boostxl_posmgr\f2837xd\source
```

Note

The software example included with the TIDM-1011 device handles proper configuration and includes the T-Format implementation in the application project. To learn how to use the reference implementation for other applications .

2.3.3.1 PM T-Format Reference Implementation Commands

Details of the T-Format protocol and commands supported in different modes can be obtained from [Tamagawa](#). [Table 2-3](#) lists the commands supported by the T-Format reference implementation.

Table 2-3. T-Format Commands Supported

COMMAND	DESCRIPTION
DATAID0	Data readout
DATAID1	Data readout
DATAID2	Data readout
DATAID3	Data readout
DATAID6	Writing to EEPROM
DATAID7	Reset
DATAID8	Reset
DATAIDD	Reading from EEPROM
DATAIDC	Reset

2.3.3.2 Functions Supported in PM T-Format Reference Implementation

The PM T-Format implementation consists of the following functions, which enable the user to interface with T-Format encoders. [Table 2-4](#) lists the functions existing in the T-Format implementation and a summary of cycles taken for execution.

Detailed explanations of each function are available at the end of [Section 2.3.3.4](#).

Table 2-4. Functions in T-Format Reference Implementation

NAME	DESCRIPTION	TYPE
PM_tformat_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.	Initialization time
PM_tformat_getCrc	To get the CRC of each byte, calculate the n-bit CRC of a message buffer using the lookup table. Use this function to calculate CRC of the position data.	Run time
PM_tformat_setupCommand	Set up an SPI and other modules, for a given command to be transmitted. All of the transactions must start with this command. This function call sets up the peripherals for the upcoming T-Format transfer, but the call does not actually perform any transfer or activity on the T-Format interface. This function call populates the sdata array of TFORMAT_DATA_STRUCT with the data to be transmitted to the encoder.	Run time
PM_tformat_receiveData	Function for unpacking and populating the T-Format 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 TFORMAT_DATA_STRUCT. Upon the function call, received data is unpacked, as per the current command, and unpacked results are stored accordingly.	Run time
PM_tformat_startOperation	This function initiates the T-Format transfer to be called after PM_tformat_setupCommand. This function performs the T-Format transaction set up by previous commands. The setup and start operation are separate function calls. The user can set up the T-Format transfer and start the actual transfer using this function call, as necessary, at a different time.	Run time
PM_tformat_setupPeriph	Setup for SPI, CLB, and other interconnect XBARS for T-Format are performed with this function during system initialization. This function must be called after every system reset. No T-Format transactions is performed until the setup peripheral function is called.	Initialization time
PM_tformat_setFreq	This function sets the T-Format clock frequency. T-Format transfers typically start low frequency during initialization and switch to higher frequency during on runtime.	Initialization time

2.3.3.3 PM T-Format Implementation Data Structures

The PM T-Format reference implementation defines the T-Format data structure handle as follows:

Object definition:

```
typedef struct {                                // bit descriptions
    uint32_t controlField;
    uint32_t statusField;
    uint16_t dataField0;
    uint16_t dataField1;
    uint16_t dataField2;
    uint16_t dataField3;
    uint16_t dataField4;
    uint16_t dataField5;
    uint16_t dataField6;
    uint16_t dataField7;
    uint16_t crc;
    uint16_t eepromAddress;
    uint16_t eepromWrDtata;
    uint16_t eepromRdDtata;
    volatile struct SPI_REGS *spi;
    uint32_t sdata[16]; //send data buffer
    uint32_t rdata[16]; //receive data buffer
    uint16_t dataReady;
    uint16_t fifo_level;
    uint32_t rxPkts[3];
} TFORMAT_DATA_STRUCT;
```

MODULE ELEMENT NAME	DESCRIPTION	TYPE
controlField	Control Field (CF) received from encoder	16 bits
statusField	Status Field (SF) received from encoder	16 bits
dataField0	Data Field 0 (DF0) received from encoder	16 bits
dataField1	Data Field 1 (DF1) received from encoder	16 bits
dataField2	Data Field 2 (DF2) received from encoder	16 bits
dataField3	Data Field 3 (DF3) received from encoder	16 bits
dataField4	Data Field 4 (DF4) received from encoder	16 bits
dataField5	Data Field 5 (DF5) received from encoder	16 bits
dataField6	Data Field 6 (DF6) received from encoder	16 bits
dataField7	Data Field 7 (DF7) received from encoder	16 bits
crc	CRC Field 7 received from encoder (CRC)	16 bits
eepromAddress	EEPROM address for read write accesses (ADF)	16 bits
eepromWrDtata	EEPROM write data for write accesses (EDF)	16 bits
eepromRdDtata	EEPROM read data received in read accesses (EDF)	16 bits
spi	SPI instance used for T-Format implementation	Pointer to Spi*Regs
sdata	Internal variables used for debug purposes	Array of 32-bit, unsigned integers
rdata	Internal variables used for debug purposes	Array of 32-bit, unsigned integers
dataReady	Flag indicating dataReady – set by the PM_tformat_receiveData function, cleared by the PM_tformat_setupCommand function	0 or 1
fifoLevel	Internal variables used for debug purposes	Maximum value 8
rxPkts	Received data in packed 32-bit packets	32 bits

To set the SPI used for tfotmat communication:

```
#define PM_TFORMAT_SPI SPIB_BASE
```

2.3.3.4 PM T-Format Function Details

PM_tformat_generateCRCTable

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 pTable.

Definition:

```
void PM_tformat_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:

```
#define NBITS_POLY1 8
#define POLY1 0x01
#define SIZEOFTABLE 256
uint16_t tformatCRCTable[SIZEOFTABLE];
// Generate table for poly 1
PM_tformat_generateCRCTable(
    NBITS_POLY1,
    POLY1,
    tformatCRCTable);
```

PM_tformat_getCrc

Description:

Calculate CRC of a message buffer by using the lookup table, to get the CRC of each byte.

Definition:

```
uint32_t PM_tformat_getCrc (uint16_t input_crc_accum, uint16_t nBitsData, uint16_t nBitsPoly,
uint16_t * msg, uint16_t *crc_table, uint16_t rxLen)
```

Parameters:

INPUT	
inputCRCaccum	Initial CRC value (seed) for CRC calculation
nBitsData	Number of bits of data for which the CRC needs are calculated
nBitsPoly	Number of bits of polynomial used for CRC computations
msg	Pointer to the data on which CRC is computed
crcTable	Pointer to the table where the CRC table values are stored
rxLen	Number of bytes of data for CRC calculation
crc	n-bit CRC value calculated

Use:

Define t-format data structure during initialization.

```
TFORMAT_DATA_STRUCT tformatData;
```

Example code:

```
crcResult = PM_tformat_getCRC( 0, // Initial seed
nBits, // number of bits of data
NBITS_POLY1, // polynomial bits
(uint16_t *)& tformatData.rxPkts, // Pointer to data array
tformatCRCTable, // CRC table with polynomial
4); // number of bytes
```

PM_tformat_setupCommand

Description:

Set up an SPI and other modules for a given command to be transmitted. All the transactions must start with this command. This function call sets up the peripherals for the upcoming transfer, but does not actually perform any transfer or activity on the interface. This function call populates the sdata array of TFORMAT_DATA_STRUCT, with the data to be transmitted to the encoder.

Definition:

```
void Val = PM_tformat_setupCommand (uint16_t dataID, uint16_t eepromAddr, uint16_t eepromData,
uint16_t crc);
```

Parameters:

INPUT		RETURN	
dataID	Mode command for the transfer to be done	Val	If incorrect, command value is passed to this function, which would return zero. For all other cases function returns a value of one.
eepromAddr	EEPROM address, depending on the mode command for the read or write address		
eepromData	EEPROM data, depending on the mode command for write access		
crc	CRC value to be sent to the encoder in case of EEPROM accesses		

Use:

Example code:

```
Val = PM_tformat_setupCommand (DATAIDD, address, 0, crcResult); PM_tformat_startOperation();
while (tformatData.dataReady != 1) {}
retvall = PM_tformat_receiveData(DATAIDD);
crcResult = PM_tformat_getCRC(0, 32, 8, (uint16_t *)&tformatData.rxPkts, tformatCRCTable, 4);
```

PM_tformat_receiveData

Description:

Function for unpacking and populating the T-Format 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 TFORMAT_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 encoder transmits the position value with the LSB first. For further details, see the encoder documentation provided by the manufacturer.

Definition:

```
uint16_t PM_tformat_receiveData (uint16_t dataID);
```

Parameters:

INPUT		RETURN	
dataID	Mode command for the T-Format transfer is done. This function must be called with the same mode command that was used to initiate the transfer.	val	If the incorrect command value is passed to this function, it returns zero. For all other cases, the function returns a value of one.

Use:

Example code:

```
retvall = PM_tformat_setupCommand (DATAID3, 0, 0, 0); PM_tformat_startOperation();
while (tformatData.dataReady != 1) {}
retvall = PM_tformat_receiveData(DATAID3);
```

PM_tformat_startOperation

Description:

This function initiates the transfer to the encoder. This function must only be called after PM_tformat_setupCommand. Therefore, the PM_tformat_startOperation function starts the transaction that was set up earlier by PM_tformat_setupCommand. The setup and start operation are separate function calls. Users 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_tformat_startOperation(void);
```

Parameters:

INPUT	RETURN
None	None

Use:

Example code:

```
retvall = PM_tformat_setupCommand (DATAIDD, address, 0, crcResult); PM_tformat_startOperation();
while (tformatData.dataReady != 1) {}
retvall = PM_tformat_receiveData(DATAIDD);
```

This function clears the tformatData.dataReady flag zero when called. This flag must subsequently be set by the SPI interrupt service routine when the data is received from the encoder. Users can poll for this flag to know if the data from the encoder is successfully received after the PM_tformat_startOperation function call.

PM_tformat_setupPeriph

Description:

Setup for SPI, CLB, and other interconnect XBARs for T-Format are performed with this function during system initialization. This function must be called after every system reset. No T-Format transactions are performed until the set-up peripheral function is called.

Definition:

```
void PM_tformat_setupPeriph (void);
```

Parameters:

INPUT	RETURN
None	None

Use:

Example code:

```
PM_tformat_setupPeriph();
```

The PM T-Format reference implementation uses an instance of SPI for communication. For proper initialization, the SPI instance must be set in `tformatData.spi` before calling this function, SPI instance must be set in `PM_tformat_include.h` (`#define PM_TFORMAT_SPI SPIB_BASE`) before calling this function, as previously shown.

PM_tformat_setFreq

Description:

This function sets the T-Format clock frequency. The T-Format transactions occur at a fixed data rate of 2.5 MBPS.

Definition:

```
void PM_tformat_setFreq(uint32_t Freq_us);
Data Rate = SYSCLK/(4* Freq_us)
```

Parameters:

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

Use:

Example code:

```
PM_tformat_setFreq(TFORMAT_FREQ_DIVIDER);
```

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-1011 device and how to get started with the T-Format Library in CCS.

To experiment with the TIDM-1011 device, the following components are required:

- TIDM-1011 EVM board
- External, 5-V, DC power supply (see [Table 1-1](#))
- F28379D LaunchPad development kit (LAUNCHXL-F28379D)
- USB-B to A cable
- T-Format Encoder from Tamagawa (for example, TS5700N8501)
- 4-pin cable from Tamagawa – length as required by the application (maximum 100 m)
- Custom adapter to connect Tamagawa, 4-position, female-terminated cable to wire leads adapter
- PC with CCS (CCS v6 or greater) installed

3.1.1.1 TIDM-1011 Jumper Configuration

Figure 3-1 shows the jumper configuration for TIDM-1011 board.

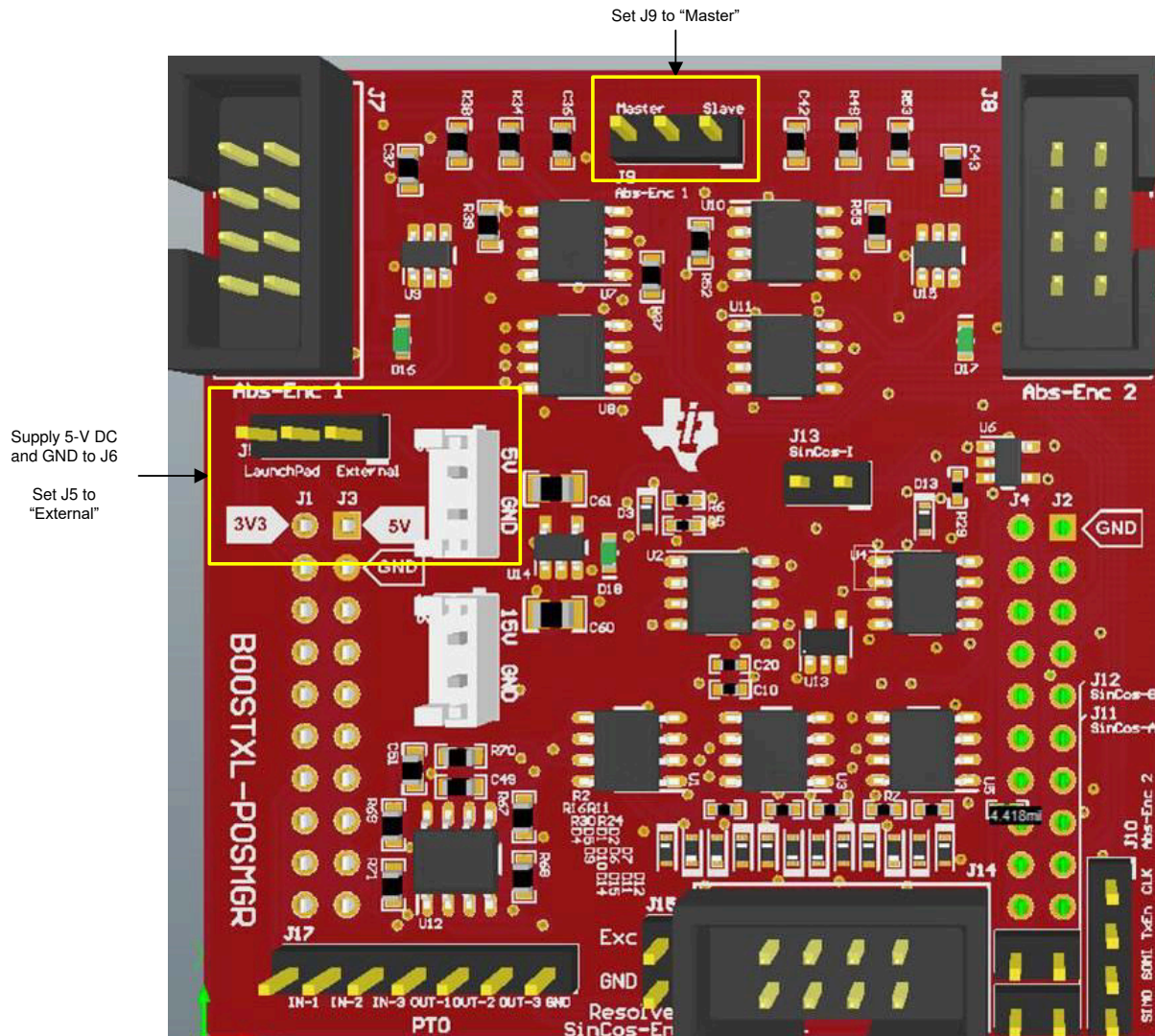


Figure 3-1. TIDM-1011/Position Manager BoosterPack Jumper Configuration

Table 3-1 lists the jumper configuration for the TIDM-1011 board.

Table 3-1. TIDM-1011 Board Jumper Details

JUMPER	FUNCTION	POSITION
J5	TIDM-1011, 5-V, power-plane source selection	<i>External</i> ⁽¹⁾
J9	Abs-Enc-1, master-slave mode selection	<i>Master</i> ⁽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 3-1.

(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 C2000WARE-MOTORCONTROL-SDK

1. Install [CCS v9.2](#) or later, if it is not already on the PC.
2. Go to <http://www.ti.com/tool/C2000WARE-MOTORCONTROL-SDK> and run the SDK installer. Allow the installer to download and update any automatically-checked software for the C2000.
3. After installation, see [Section 2.3.3](#) for more information on the T-Format reference example.


3.1.2.2 Configuring CCS for F28379D LaunchPad™

1. Open CCS. This document assumes that version 9.2 or later is used.
2. When CCS opens, the workspace launcher may ask users 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), what configuration is selected, and so forth are saved. This workspace can be anywhere on the disk. The following location mentioned is just for reference.
 - a. Click the *Browse...* button.
 - b. Create the following path by making new folders as necessary:

```
C:\c2000_projects\CCS_workspaces\PM_tformat_eval_workspace
```

- c. Uncheck the box that says *Use this as the default and do not ask again*.
- d. 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 next step. CCS is configured to know which MCU the program is connecting to. This configuration is done by setting up the *Target Configuration*.

3. A new configuration file can be set by clicking *View* → *Target Configuration*. This procedure opens the Target Configuration window (see [Figure 3-2](#)).
4. In this window, click on the  icon. 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*.

This step opens a new tab, as shown in [Figure 3-2](#).

5. Select and enter the following options:
 - a. Connection – *Texas Instruments XDS100v2 USB Emulator* or *Texas Instruments XDS100v2 USB Debug Probe*
 - b. Device – C2000 MCU on the control card, for example, TMS320F28379D
 - c. Click the Save button and close.

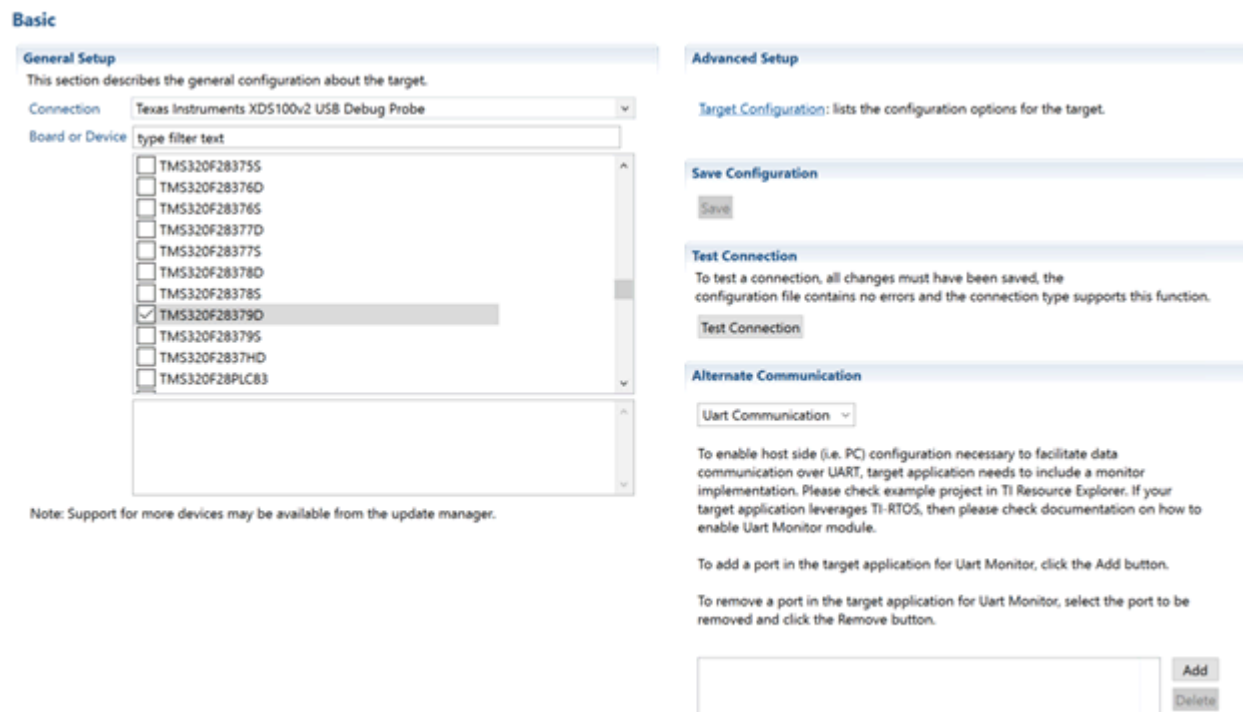


Figure 3-2. Configuring New Target Configuration

6. Click *View* → *Target Configurations*.
7. 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:
 - a. Click *View* → *Target Configurations*
 - b. Expand *Projects* → *PM_tformat_systemtest*.
 - c. Right-click the *xds100v2_F2837x.ccxml* and *Set as Default* files.

This tab also allows users to reuse the existing target configurations and links them to specific projects.

8. Add the PM T-Format evaluation example project into the current workspace by clicking *Project* → *Import CCS Project*.
9. Select the project by browsing to <C2000WARE-MOTORCONTROL-SDK> solutions\boostxl_posmgr \f2837xd\ccs (see [Figure 3-3](#)).

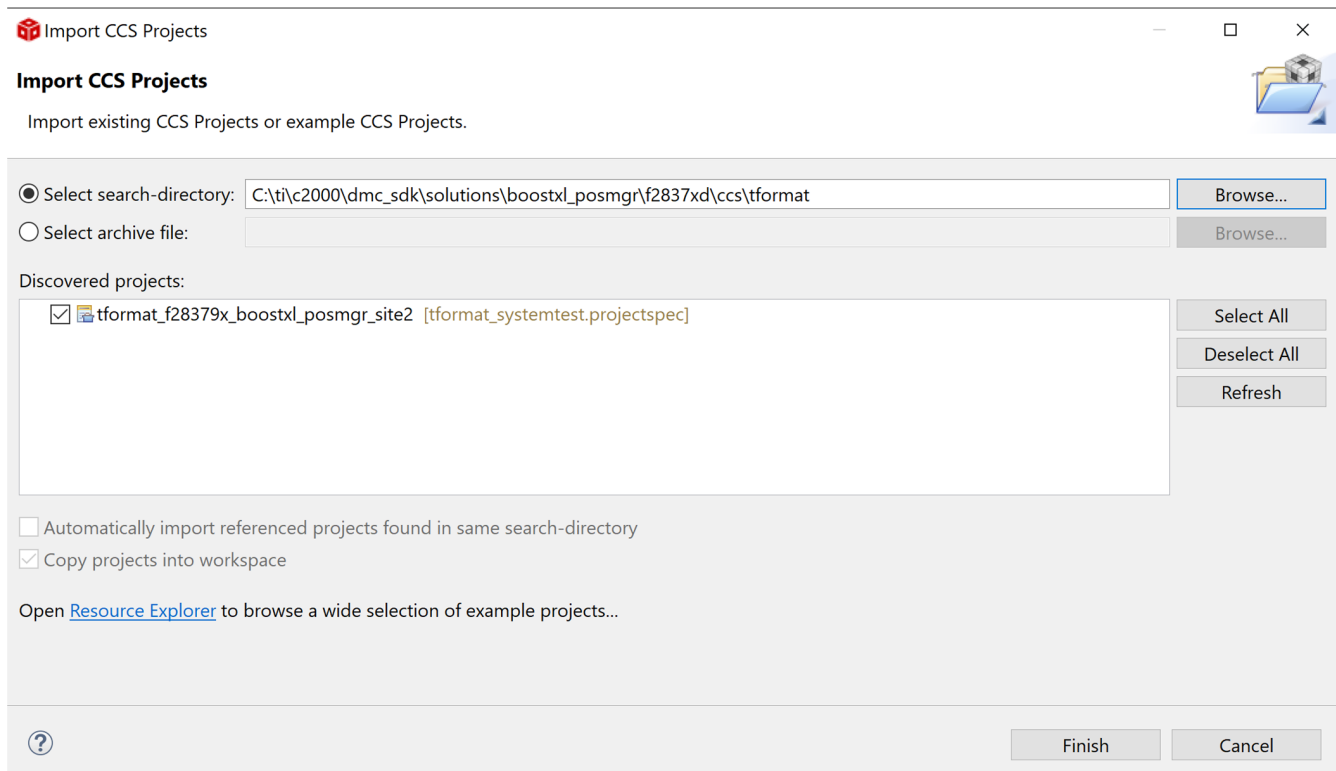


Figure 3-3. Adding PM T-Format Example Project to Workspace

10. If there are multiple projects in this directory, choose the projects to import and then click the Finish button. This action copies all of the selected projects into the workspace.
11. Assuming this is the first-use of CCS, xds100v2-F2837x should have been set as the default target configuration. Verify this setting by viewing the xds100v2-f2837x.ccxml file in the expanded project structure and observe 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.
12. 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 3-4, right-click on an individual project and select Active Build Configuration → CPU1_RAM configuration.

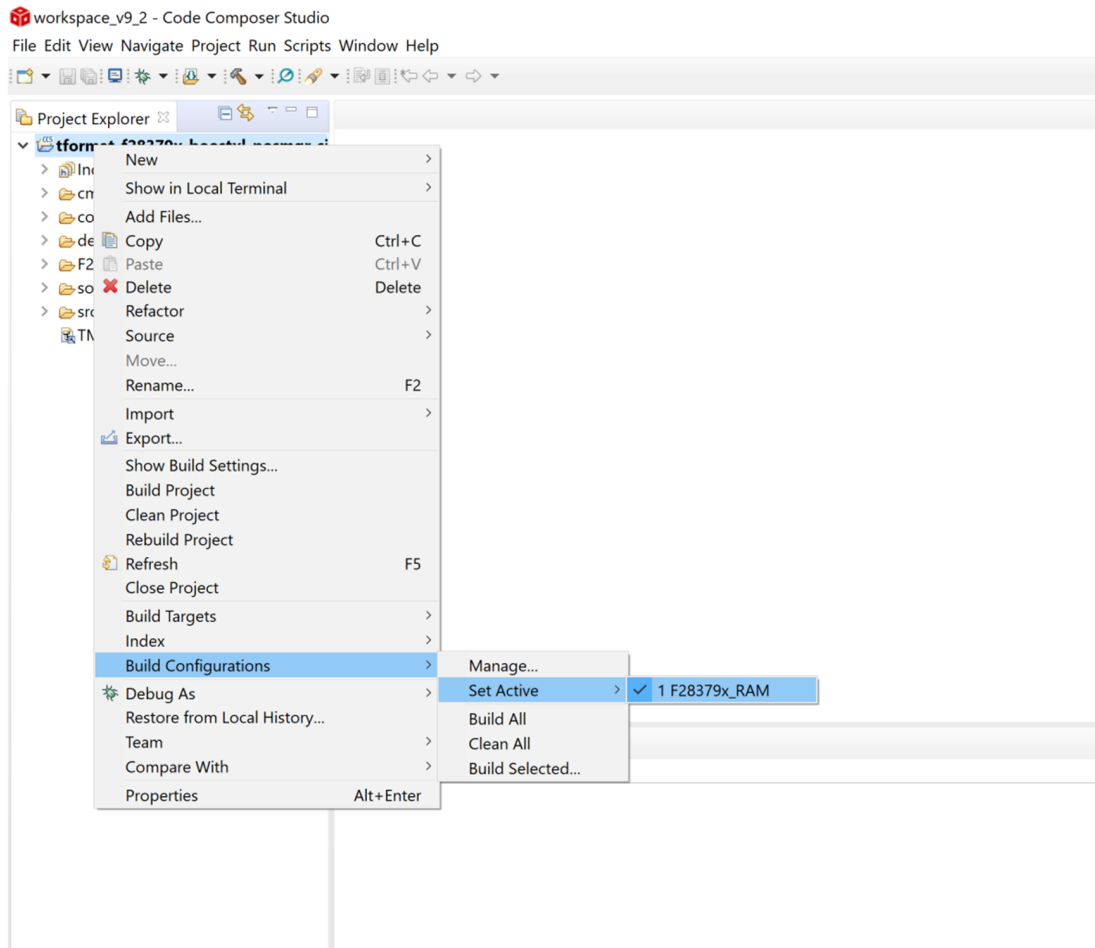


Figure 3-4. Selecting F28379x_ RAM Configuration

3.1.2.3 Using PM_tformat Reference Implementation in New or Existing Projects

Note

Users with an existing project in CCS can skip Step 1.

1. To create a new project go to **File > New > CCS Project**. Below target, choose the device TMS320F28379D. Name the project and select a project template. Then, click **Finish**.
2. Include the T-Format library in {ProjectName}-Includes.h.

```
#include "PM_tformat_Include.h"
```

3. Add the PM_tformat library path in the include paths below **Project Properties > Build > C2000 Compiler > Include Options**. Path for the library:

<C2000WARE-MOTORCONTROL-SDK>\libraries\position_sensing\tformat\lib (see [Figure 3-5](#)).

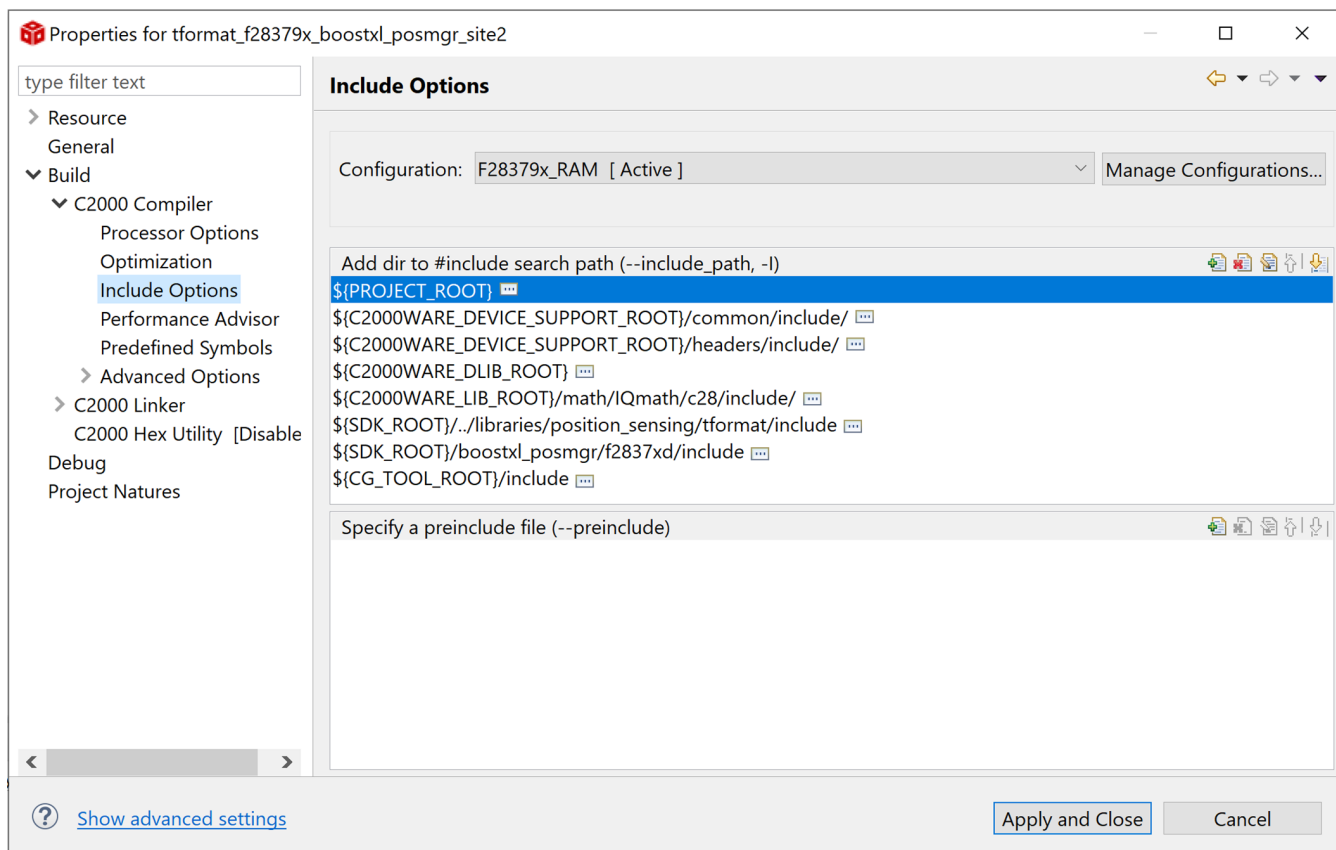


Figure 3-5. Compiler Options for Project Using PM T-Format Library

Note

The exact location may vary, depending on where C2000WARE-MOTORCONTROL-SDK is installed and which other libraries the project is using.

4. Figure 3-6 shows how to add the PM T-Format Library to Linker Options in a CCS Project

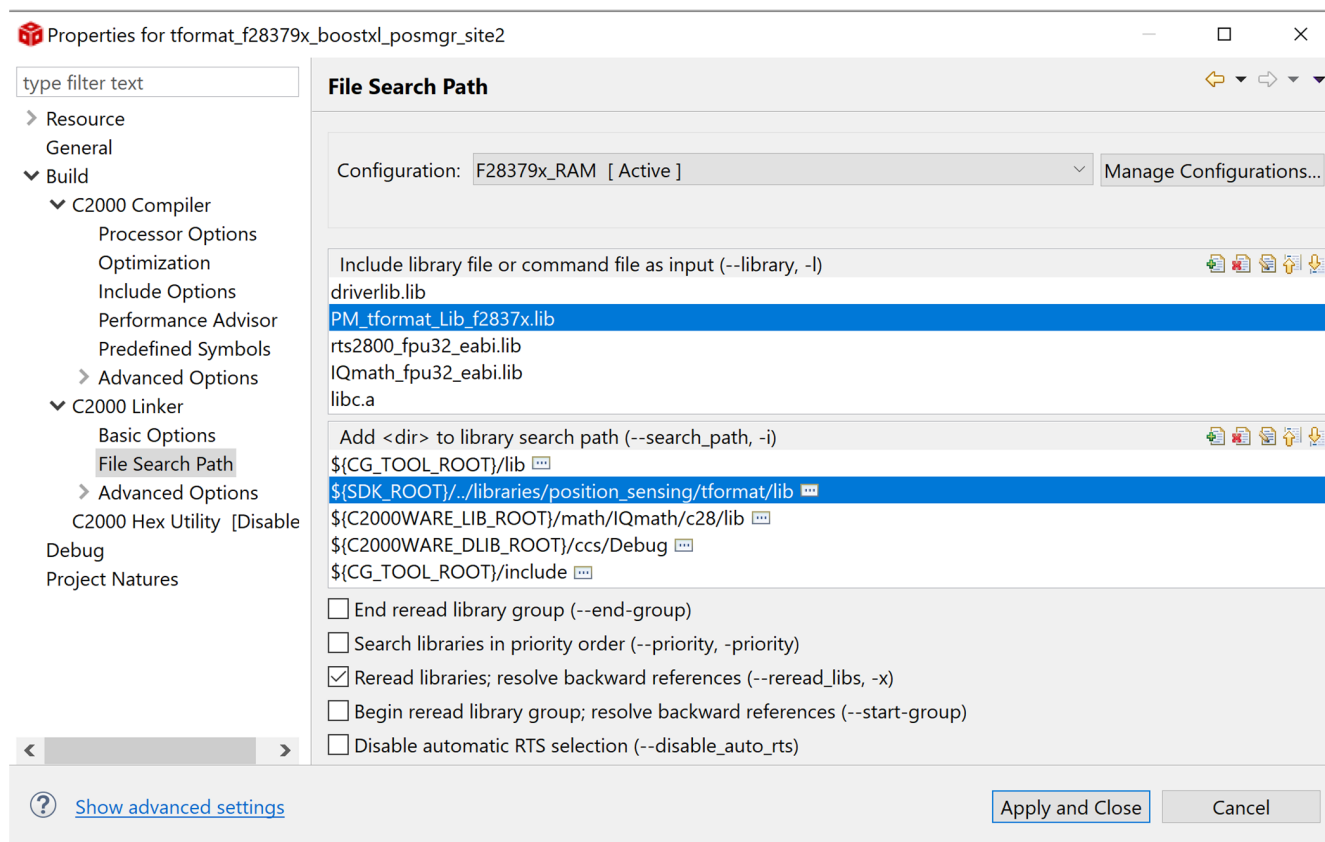


Figure 3-6. Adding PM T-Format Library to Linker Options in CCS Project

5. Initialization steps: The following steps are needed for initialization and proper functioning of the T-Format reference implementation functions. For more details, see the examples provided with the library.
- Create and add the module structure to {ProjectName}-Main.c.

```
TFORMAT_DATA_STRUCT tformatData; //PM tformat data structure
```

- Define the frequency of the clock divider.

```
#define TFORMAT_FREQ_DIVIDER 20
```

- Set the SPI instance to be used for communication. For use on the F28379D LaunchPad, the SPI instance must be set to SpiB, and the SpiB receive FIFO interrupt must be enabled.

```
SPI_enableInterrupt(SPIB_BASE, SPI_INT_RXFF);
// Configuration complete. Enable the module. // SPI_enableModule(SPIB_BASE);
```

Note

Alternatively, users can also poll for the Interrupt Flag and not necessarily use interrupt. Ensure the SPIRXFIFO contents are copied into tformatData.rdata after the flag is set. This step must be executed before calling the PM_tformat_receiveData function. Also, the interrupt flag must be cleared to get the SPI ready for the next T-Format transaction.

```
for (i = 0; i <= tformatData.fifoLevel; i++) { tformatData.rdata[i] =
SPI_readDataNonBlocking(PM_TFORMAT_SPI); } SPI_clearInterruptStatus(PM_TFORMAT_SPI,
SPI_INT_RXFF_OVERFLOW);
```

- d. Enable clocks to EPWM instance 4.

```
SysCtl_enablePeripheral(SYSCTL_PERIPH_CLK_EPWM4);
```

- e. Initialize and set up peripheral configuration by calling the PM_tformat_setupPeriph function.

```
PM_tformat_setupPeriph();
```

- f. Set up GPIOs needed for configuration, which is required for the F28379D LaunchPad. The GPIOs used for SPI must be changed based on the chosen SPI instance. GPIO7, and GPIO34 must always be configured.

```
// GPIO7 is SPI Clk slave
GPIO_setMasterCore(7, GPIO_CORE_CPU1);    GPIO_setPinConfig(GPIO_7_EPWM4B);
// GPIO63 is the SPISIMOB //
GPIO_setMasterCore(63, GPIO_CORE_CPU1);    GPIO_setPinConfig(GPIO_63_SPISIMOB);
GPIO_setQualificationMode(63, GPIO_QUAL_ASYNC);
// GPIO64 is the SPISOMIB //
GPIO_setMasterCore(64, GPIO_CORE_CPU1);    GPIO_setPinConfig(GPIO_64_SPISOMIB);
GPIO_setQualificationMode(64, GPIO_QUAL_ASYNC);
// GPIO65 is the SPICLKB //
GPIO_setMasterCore(65, GPIO_CORE_CPU1);    GPIO_setPinConfig(GPIO_65_SPICLKB);
GPIO_setQualificationMode(65, GPIO_QUAL_ASYNC);
// GPIO66 is the SPISTEB //
GPIO_setMasterCore(66, GPIO_CORE_CPU1);    GPIO_setPinConfig(GPIO_66_SPISTEB);
GPIO_setQualificationMode(66, GPIO_QUAL_ASYNC);
// GPIO9 is tformat TxEN //
GPIO_setMasterCore(9, GPIO_CORE_CPU1);    GPIO_setPinConfig(GPIO_9_OUTPUTXBAR6);
// GPIO139 is PwrEN //
GPIO_setMasterCore(139, GPIO_CORE_CPU1);
GPIO_setDirectionMode(139, GPIO_DIR_MODE_OUT);
```

- g. Set up XBAR, shown as follows. The GPIOs used for the SPI must be changed based on the chosen SPI instance.

```
//SPISIMOB on GPIO63 connected to CLB4 i/p 1 via XTRIP and CLB X-Bars
//XTRIP1 is used inside the CLB for monitoring received data from Encoder.

XBAR_setInputPin(XBAR_INPUT1, 63);
```

- h. Initialize CRC-related object and table generation:

```
//CRC tables to be used for CRC calculations
uint16_t table1[SIZEOFTABLE];
// Generate table for poly 1 and generated tables are placed in tformatCRtable(NBITS_POLY1,
POLY1, tformatCRtable);
-- Constants are defined in PM_tformat_Include.h as below.
#define NBITS_POLY1 8
#define POLY1 0x01
#define SIZEOFTABLE 256
```

- i. Turn the power ON. The GPIO used for power control can be changed based on the hardware. GPIO32 is used for power control on the F28379D LaunchPad.

```
// Power up 5v supply through GPIO139
GPIO_writePin(139, 1);
```

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-1011 device is as described in [Table 3-1](#).
2. Connect the TIDM-1011 device to the LaunchPad using the BoosterPack connector (J1 to J3 and J4 to J2). Ensure the TIDM-1011 device is connected to site two of the LaunchPad, as shown in [Figure 3-7](#).

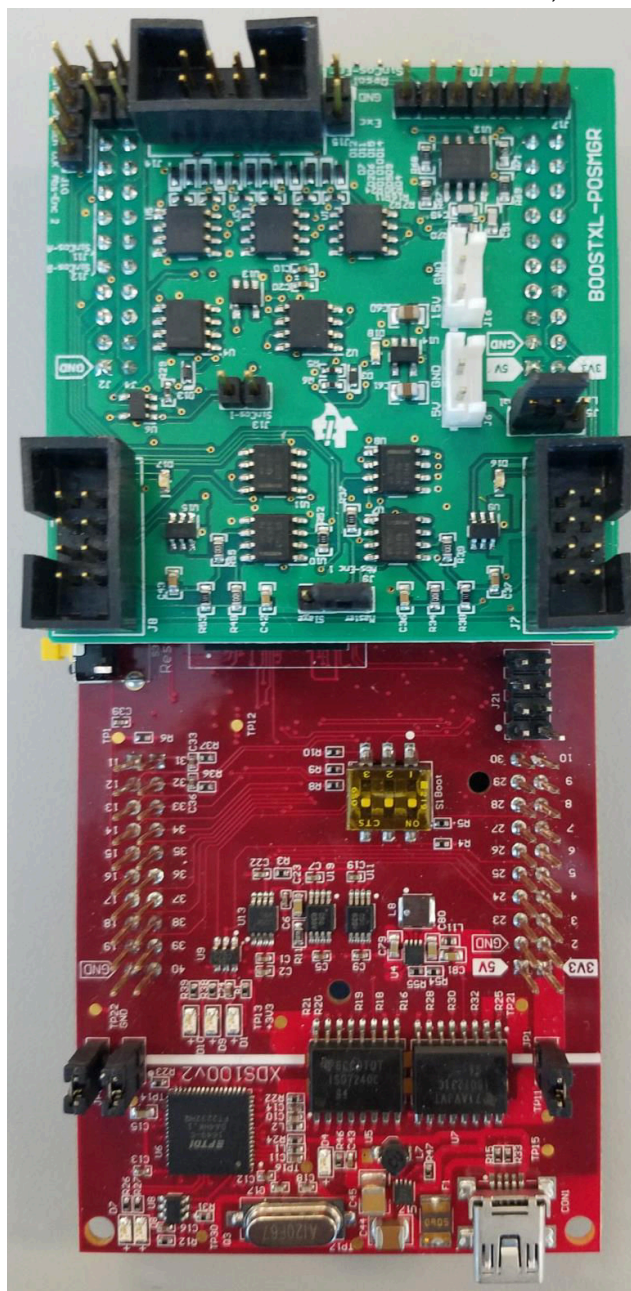


Figure 3-7. Position Manager BoosterPack Connected to Site Two of LaunchPad™

3. Connect the USB cable to the LaunchPad.
4. Set up the connection to the encoder.
 - a. Prepare an adapter to connect the Tamagawa cable to the T-Format interface using a 8-position female to wire the leads adapter (see the BOM for the header used for the encoder connector, J7).

- b. Insert the header of the adaptor, created in the previous step, to connect to Abs-Enc-1 (J7). The female end of the Tamagawa cable connects to the encoder. [Figure 3-8](#) shows the pinout of J7.

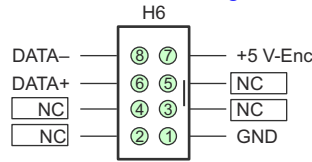


Figure 3-8. Abs-Enc-1 (J7) Pinout on TIDM-1011 Board

- 5. Supply 5-V DC and GND to J6, as shown in [Figure 3-1](#). The board should now look like [Figure 3-9](#). LED D18 should be lit, which shows that the board has power.

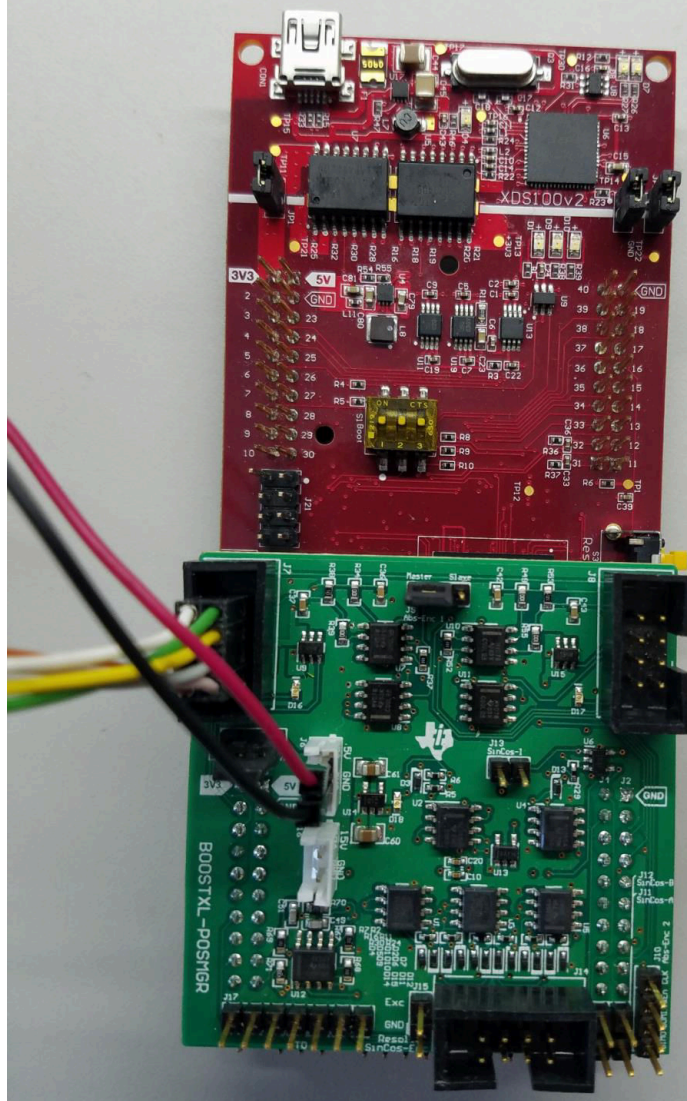


Figure 3-9. Position Manager BoosterPack Powered On and Connected to Tamagawa Encoder

3.2.2 Test Results

This section describes how to run the software example and detail the results in CCS. Figure 3-10 shows the software flow diagram for this project.

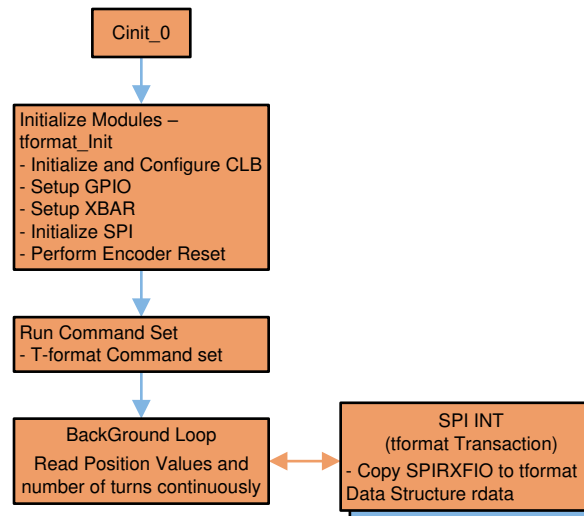



Figure 3-10. Software Flow Diagram for Example Project PM_tformat_systemtest

3.2.2.1 Building and Loading Project

1. Open the tformat.h file and ensure that TFORMAT_FREQ_DIVIDER is set as needed; save this file.
2. 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.
3. On successful completion of the build, click the Debug button , (located in the top-left side of the screen).
4. The following window may pop up, requesting that users select which of the two CPUs (in F28379x) to connect to (see [Figure 3-11](#)), if it is the first time that the project is used. Select CPU1 by clicking the box next to it.

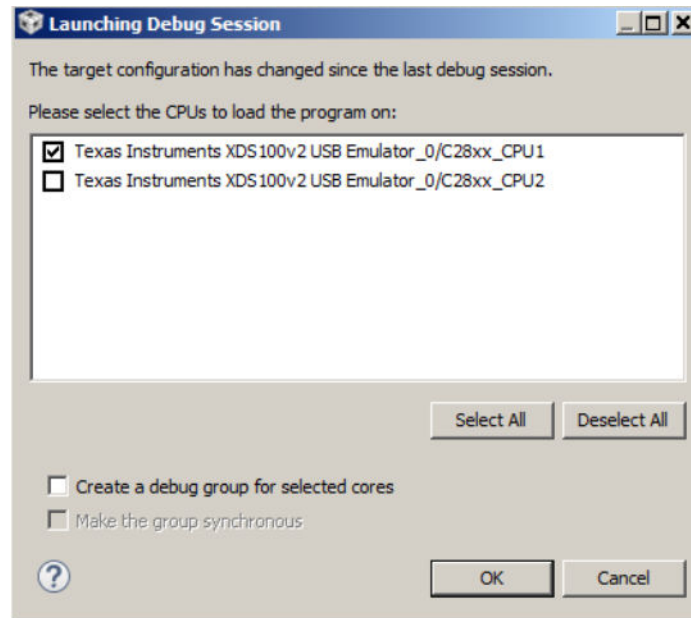




Figure 3-11. Selecting the CPU(s) to Connect

The IDE automatically connects to the target, loads the output file into the device, and changes to the Debug perspective.

5. Click Tools → Debugger Options → Program / Memory Load Options. Users can enable the debugger to reset the processor each time it reloads the program by checking Reset the target on program load or restart and clicking Remember My Settings, to make this setting permanent.
6. Click on the Enable silicon real-time mode button , which auto selects the Enable polite real-time mode button .

This option allows users to edit and view variables in real time. Do not reset the CPU without disabling these real-time options.

7. Select YES to enable debug events if a message box appears. This action sets bit 1, the debug enable mask (DGBM) bit, of the status register 1 (ST1) to 0. When the DGBM bit is set to 0, memory and register values can be passed to the host processor for updating the debugger windows.

3.2.2.2 Setting Up Watch Window and Graphs

1. Click View → Expressions on the menu bar to open a watch window, to view the variables being used in the project.
2. Add variables to the watch window, as shown in [Figure 3-12](#). The window uses the number format associated with the variables during declaration. Users can select a desired number format for the variable by right-clicking on it and choosing. Configure the expressions window as shown in [Figure 3-12](#).







Expression	Type	Value
 tformatData	struct <unnamed>	{...} (Hex)
(x) controlField	unsigned int	0x0058 (Hex)
(x) statusField	unsigned int	0x0004 (Hex)
(x) dataField0	unsigned int	0x007C (Hex)
(x) dataField1	unsigned int	0x003E (Hex)
(x) dataField2	unsigned int	0x00EA (Hex)
(x) dataField3	unsigned int	0x00E8 (Hex)
(x) dataField4	unsigned int	0x0000 (Hex)
(x) dataField5	unsigned int	0x0000 (Hex)
(x) dataField6	unsigned int	0x0000 (Hex)
(x) dataField7	unsigned int	0x0003 (Hex)
(x) crc	unsigned int	0x0061 (Hex)
(x) eepromAddress	unsigned int	0x0081 (Hex)
(x) eepromWrDdata	unsigned int	0x0000 (Hex)
(x) eepromRdDdata	unsigned int	0x0000 (Hex)
▶  spi	struct SPI_REGS *	0x00006110 (Hex)
▶  sdata	unsigned long[16]	0x0000AA10@Dat...
▶  rdata	unsigned long[16]	0x0000AA30@Dat...
(x) dataReady	unsigned int	0x0001 (Hex)
(x) fifo_level	unsigned int	0x0008 (Hex)
▶  rxPkts	unsigned long[3]	0x0000AA52@Dat...
(x) crcResult	unsigned long	0x000000DF (Hex)
(x) position	unsigned long	0x00577C3F (Hex)
(x) turns	unsigned long	0
 Add new expression		

Figure 3-12. Configuring Expressions Window





1. Click the Continuous Refresh button , in the watch window.

This action enables the window to run in real-time mode. By clicking the down arrow in this watch window, users can select Customize Continuous Refresh Interval and edit the refresh rate of the watch window. Choosing too fast an interval may affect performance.

3.2.2.3 Running Code

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

The project should run and the values in the watch window should continuously update. The following are screen captures of typical CCS perspectives while using this project. Users may want to resize the windows according to their preference.

2. Once complete, reset the processor (Run → Reset → CPU Reset) , and terminate the debug session by clicking (Run → Terminate) . This halts the program and disconnects Code Composer Studio from the MCU.
3. As the encoder sends the position information, observe the same in the watch window, turns variables.
4. If the encoder is not mounted on a spinning Motor, users can manually rotate the encoder shaft and observe the position changing in the watch window.
5. It is not necessary to terminate the debug session each time code is change or ran. Instead, follow this procedure:
 - a. After rebuilding the project, (Run → Reset → CPU Reset) , (Run → Restart) , and enable real-time options.
 - b. Once complete, disable real-time options and reset the CPU.
 - c. 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 the tformat.h file.
7. Modify the example code as needed and perform tests with different cable lengths.

3.2.2.4 Cable Length Validation

[Table 3-2](#) lists tests with various types of encoders; cable length tests are performed. Tests include basic command-set exercising and reading-position values, with additional data if applicable.

Table 3-2. Cable Length Test Report

ENCODER NAME	TYPE	RESOLUTION (BITS)	CABLE LENGTH ⁽¹⁾ (m)	MAXIMUM T-FORMAT CLOCK	TEST RESULT
TS5702N40	Rotary	17 bits	70 m	2.5 MBPS	Pass
TS5700N8501	Rotary	24 bits	70 m	2.5 MBPS	Pass

- (1) Cable lengths up to 100 m have also been tested with some of the encoders.

4 Design Files

To download the design files, see the product page at [TIDM-1011](#).

5 Software Files

See [Section 3.1.2.3](#).

6 Related Documentation

1. Texas Instruments, [DesignDRIVE Development Kit IDDK v2.2.1 - User's Guide](#)
2. Texas Instruments, [DesignDRIVE Development Kit IDDK v2.2.1 - Hardware Reference Guide](#)
3. Texas Instruments, [C2000 DesignDRIVE](#), software for Industrial Drives and Motor Control
4. Texas Instruments, [C2000 Position Manager SinCos Library](#), User's Guide

6.1 Trademarks

C2000™, BoosterPack™, TI E2E™, E2E™, and Delfino™ are trademarks of Texas Instruments. LaunchPad™ is a trademark of Texas Instruments Incorporated. All other trademarks are the property of their respective owners.

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-1011 board is identical to the Position Manager BoosterPack EVM (see Section 2.3.1)
CRC	Cyclic redundancy check
T-Format	T-Format specification of the encoder interface protocol by Tamagawa
Subsequent Electronics	T-Format Master implementation
PM	Position Manager – foundational hardware and software on C28x devices for position encoder interfaces
PM_tformat	Prefix used for all the reference implementation 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.

9 Revision History

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

Changes from Revision B (January 2020) to Revision C (September 2020)	Page
• Updated the numbering format for tables, figures and cross-references throughout the document.....	2
Changes from Revision A (November 2019) to Revision B (January 2020)	Page
• Added paragraph describing figures 6-11.....	6
• Added figures 6-11 to show a logic schematic lens.....	6
Changes from Revision * (April 2018) to Revision A (November 2019)	Page
• Added <i>Implementation Details</i> section.....	6
• Added <i>Configuration of Inputs to CLB</i> section.....	15
• Added <i>Function Description</i> section.....	15
• Changed example code for PM_tformat_setupPeriph.....	20
• Changed information in <i>Software</i> section.....	29
• Changed encoder number in <i>Cable Length Test Report</i> table.....	41

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES "AS IS" AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI's products are provided subject to TI's Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI's provision of these resources does not expand or otherwise alter TI's applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2020, Texas Instruments Incorporated