TI Designs

BiSS-C Interface Master Design Guide

TI Designs

TI Designs provide the foundation that you need including methodology, testing and design files to quickly evaluate and customize the system. TI Designs help *you* accelerate your time to market.

Design Resources

TIDEP0022	TI Design Files
AM437x	Product Folder
TMDXIDK437x	Product Folder
SN65HVD78D	Product Folder

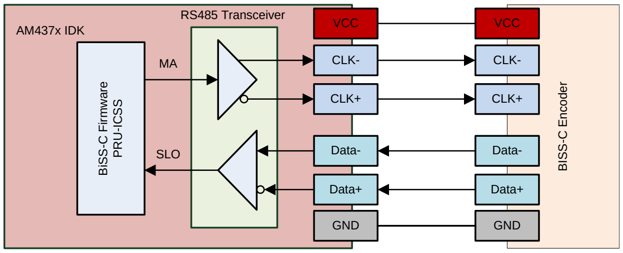
ASK Our E2E Experts
WEBENCH® Calculator Tools

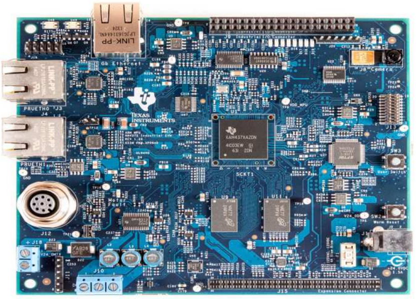
Design Features

- Offers a BiSS-C Interface Master For Point-To-Point Communication Running on PRU-ICSS
- Offers an Interface Speed of 1, 2, 5, and 10 MHz
- Offers an 8× Oversampled Input Capture
- Offers a Control Communication Interface
- Offers Line-Delay Compensation With Filtered-Sample Point
- Offers a Debouncing Filter on Oversampled Input
- Supports up to 100-m Cable
- Runs on AM437x With PRU-ICSS

Featured Applications

- Factory Automation and Process Control
- Sensors and Field Transmitters
- Motor Drives
- Position Control





1 Introduction

This design implements the BiSS-C interface master on the TI Sitara™ AM437x Industrial Development Kit (IDK).

BiSS is an open-source digital interface for sensors and actuators. BiSS stands for bidirectional serial synchronous. The BiSS interface was introduced by iC-Haus GmbH as an open-source protocol in 2002. This hardware is compatible with the industrial-standard serial synchronous interface (SSI) and uses two unidirectional lines for the clock and data, respectively.

BiSS-C mode is the continuous mode in which the BiSS-C interface master reads out the position data cyclically. Control communication is available for the master to send commands to the slaves and to read and write the slave local registers.

The BiSS interface is used in position-control applications. The interface enables a complete closed-loop position control system by providing the real-time position feedback to the master to control the motor.

The existing solutions on the market are based on Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). The implementation within this document provides an integrated solution that implements the BiSS-C interface master using programmable real-time unit on an industrial communication subsystem (PRU-ICSS).

2 System Overview

The position-feedback system is a position encoder attached to a motor with a cable up to 100 meters long. The cable provides power and serial communication and the master interface to the position encoder. For the Sitara AM437x processor, the master interface for position encoder is a function of a connected-drive controller. The AM437x provides the resources for industrial Ethernet and a motor-control application including on-chip analog-to-digital converters (ADCs) for measuring the current. The BiSS-C interface master on the Sitara AM437x processor uses one of four programmable real-time units (PRUs). The firmware is developed on the 32-bit RISC processor using register-mapped I/Os.

This design of the BiSS-C interface master interfaces the BiSS-C position encoder with a BiSS-C digital interface. This interface enables the simultaneous transmission of position data and the control data through the same wire. The SN65HVD78D RS485 transceiver interfaces the BiSS-C position encoder with the BiSS-C interface master.

Figure 1 shows the overview of the BiSS position feedback system. AM437x IDK with an onboard RS-485 transceiver is used for this design. The supply voltage for the encoder is available on the M12 connector.

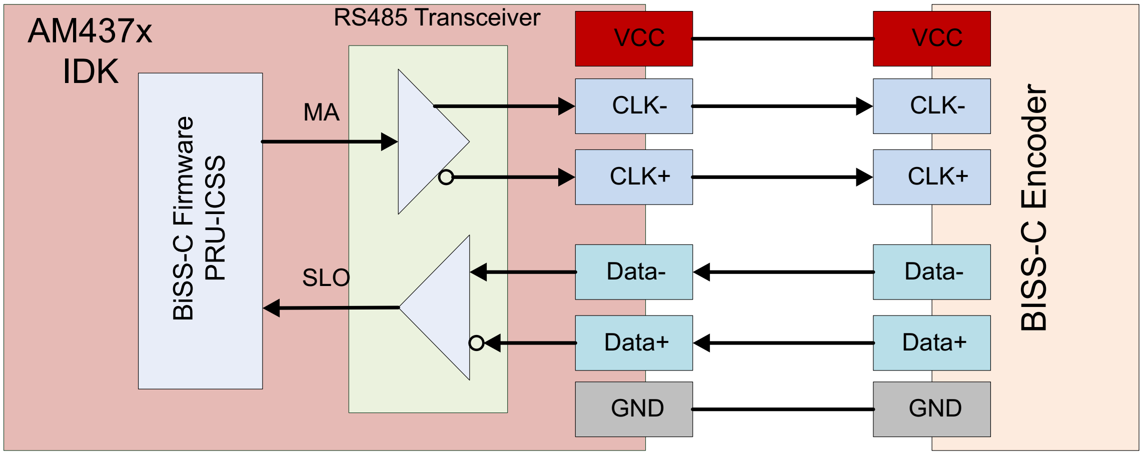


Figure 1. System Block Diagram

2.1 BiSS Protocol

BiSS-C allows the simultaneous transmission of position data and the control data over the same line. BiSS-C is hardware compatible with the SSI.

Though position encoders typically provide the feedback-position data of motors, this encoder allows closed-loop control of motors. This design implements the BiSS-C interface master for point-to-point communication.

Figure 2 shows a BiSS frame in point-to-point communication.

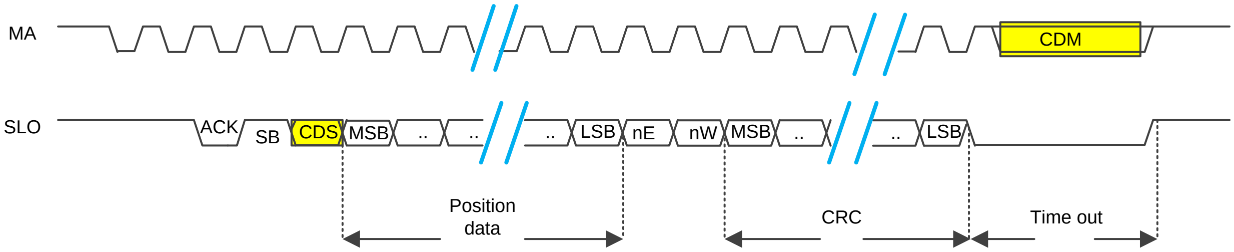


Figure 2. BiSS Frame

In the reset state, the master clock (MA) and slave data out (SLO) lines are active high. The BiSS master interface starts sending the clock over the MA line; on the second rising edge of the MA clock, the slave responds with a low signal, which is an acknowledge signal (ACK) of BiSS frame. For the next MA clock cycle, a start bit (SB) is asserted by the slave. Following the SB, the slave sends a control data slave (CDS) bit, which is the response of the control data master (CDM) bit. The CDM bit is the inverted state of the MA line during the BiSS time-out.

After the CDS bit, the slave sends the position data with the most significant bits (MSBs), followed by an error bit (nE) and a warning bit (nW). The slave sends the cyclic redundancy check (CRC) bits with MSB first. The CRC is sent inverted by the slave over the SLO line.

When the slave finishes sending all the bits, the slave goes to the BiSS time-out state driving SLO to a low signal level. The slave goes to high when it is ready for the next transmission or expiration of the BiSS time-out. The inverted state of the MA clock line during the BiSS time-out is the CDM bit for control communication. One control data bit is sent in each BiSS frame by the master. The slave responds with one CDS bit in each BiSS frame.

2.1.1 Line-Delay Compensation

In a real-world application environment, the encoder can be far from the BiSS-C interface master. A long-cable connection between the encoder and the BiSS-C interface master can delay transmission and physical noises.

Line delay is the delay due to the cable lengths in the BiSS-C transmission. When the BiSS-C interface master starts sending the clock, extra time is required for data to reach the encoder. When the slave receives the clock, it responds with slave data. The slave response also travels the reverse path to the BiSS-C interface master through the cable. The time delay in the transmission of data over the wire is proportional to the length of the cable. With a cable length up to 100 meters, a cable delay of 1 µs from the time the master sends the clock until it receives the slave response is possible.

The BiSS-C interface master has mechanism to compensate for line delay and avoid errors in the transmission of longer cables.

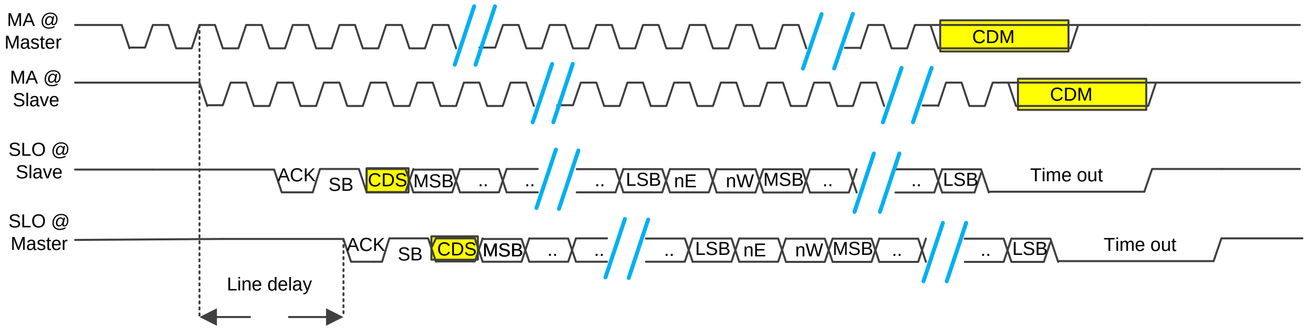


Figure 3. Line-Delay Compensation

Figure 3 shows the signals in two perspectives. The MA @ Master line shows how the clock looks at the BiSS-C interface master. Due to the line delay, the clock signal is delayed at the slave. The MA @ Slave line shows the delay. The slave responds to the second rising edge of delayed MA clock. The SLO @ Slave line shows the response of the slave to MA @ Slave. The response takes some time to travel to the master. Traveling to the master is also delayed as shown in the SLO @ Master signal. By measuring the time duration between second rising edge of the MA clock and the first falling edge of the SLO line, the total time delayed can be calculated. To avoid the transmission errors, BiSS-C master compensates for this line delay.

2.1.2 Processing Time Request by Slave

A slave can request processing time before sending its sensor data when it requires additional time. Extra time is required for operations like analog-to-digital conversion and memory access. The slave indicates the processing time by delaying the SB. The master must check whether slave is requesting processing time and provide additional clock cycles.

Figure 4 shows how the slave requests processing time by delaying the SB. If there is request for process time, the ACK bit is low for more than one BiSS clock cycle. Because this consumes extra clock cycles, the BiSS-C interface master must provide extra clock cycles to the slave.

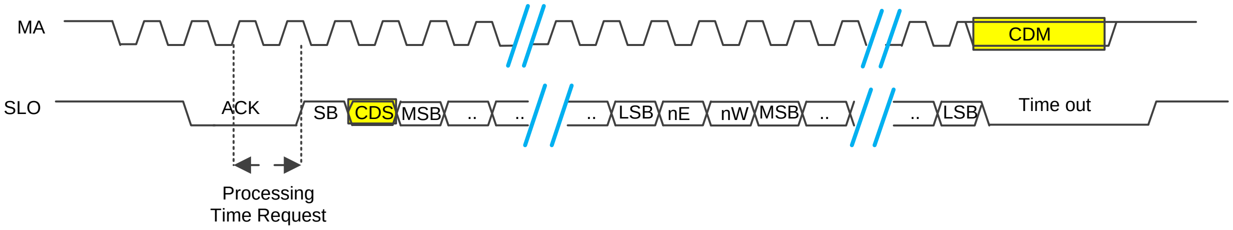


Figure 4. Processing a Time Request by Slave

2.1.3 Control Communication

This document describes some important concepts of the control communication. For more information on control communication, see www.biss-interface.com.

In BiSS-C communication, the master can send control data over the same line without interrupting the position-data communication. The master sends one control data bit per BiSS frame and the slave responds to these control data bits with one CDS per BiSS frame.

The control frame is for reading and writing to the slave registers. CRC also protects control communication. A control frame is the result of several BiSS frames.

Before starting a control frame, the BiSS-C interface master must transmit at least 14 numbers of CDM equaling 0. Any control frame can be cancelled by transmitting 14 numbers of CDM equaling 0.

In the control communication, addressing is crucial. For the addressing of the slaves, use a slave ID. The slave ID is assigned according to the sequence in the chain and carried out by setting the ID lock (IDL) bits for the first eight IDs.

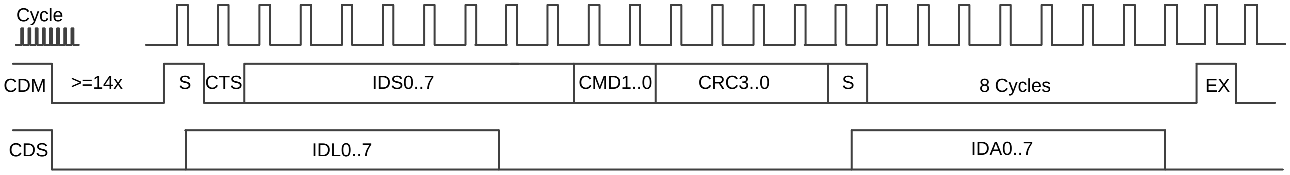


Figure 5. Command Frame

Control communication has the following two types:

- The slave register access (Register Read/Write)
- The command frame

Figure 5 shows the command frame. In the command frame, the control select bit (CTS) is zero (CTS = 0). Using the command frame to support a multipoint connection is beyond the scope of this design.

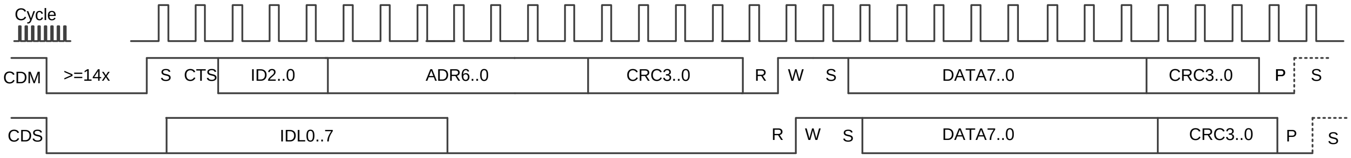


Figure 6. Register Write

In the register communication (register write or read), the CTS bit is set (CTS = 1). Figure 6 shows the register write frame.

The slave register access has only 3 ID select bits and 7 bits of register address. Because binary coding is used, eight slaves and 128 registers can be addressed. This communication is also protected by CRC. After the CRC has been transmitted, R bits, W bits, and S bits (the read bit, write bit, and start bit, respectively) are sent (see Figure 6). For a write, access RW equals 01. For a read, access RW equals 10. After the CRC is sent, the P and S bits are sent. A stop bit (P = 0) is at the end of the frame. Following the stop bit, there can be an optional start bit if the master must perform a sequential access of additional slave registers.

Figure 7 shows the register read access by the master. The RW combination is RW equals 10. After the start bit, the master sends 12 0 bits and 1 stop bit. The slave responds with 8 bits of data protected with 4 bits of CRC.

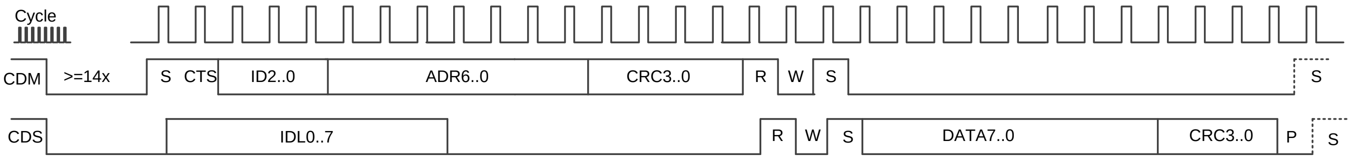


Figure 7. Register Read

For more information regarding the control communication, see www.biss-interface.com.

2.2 Sitara AM437x Processor

The Sitara AM437x processor family is based on a powerful ARM® Cortex®-A9 core and a rich peripheral set for industrial interfaces, graphics, security, and motor control. On-chip SRAM of 256KB is a shared memory between the industrial communication subsystem and the ARM CPU. There are four PRU cores with their own instruction and data memory.

The four PRUs have full access in the system and can communicate with all memories and I/Os except ARM CPU memory. The interrupt controller (INTC) in ICSS can send and receive events. Two ICSS blocks are available for industrial Ethernet and motor-side communication. The device can boot from various sources including QSPI. Industrial temperature range is supported up to 105°C/T_j. With a JTAG interface, all cores including PRUs can be debugged using Code Composer Studio™ (CCS). The processor supports parallel debugging ARM and PRUs. PRU code is developed using external assembler. The binaries can be loaded manually through CCS or using ARM side loader for PRU firmware. Before PRU code is executed, configure the subsystem and multiplex the pins. Choose pins for your register-mapped I/Os for the BiSS interface. The IDK connects to a fast RS-485 transceiver onboard for the BiSS implementation. See Figure 8 for more details.

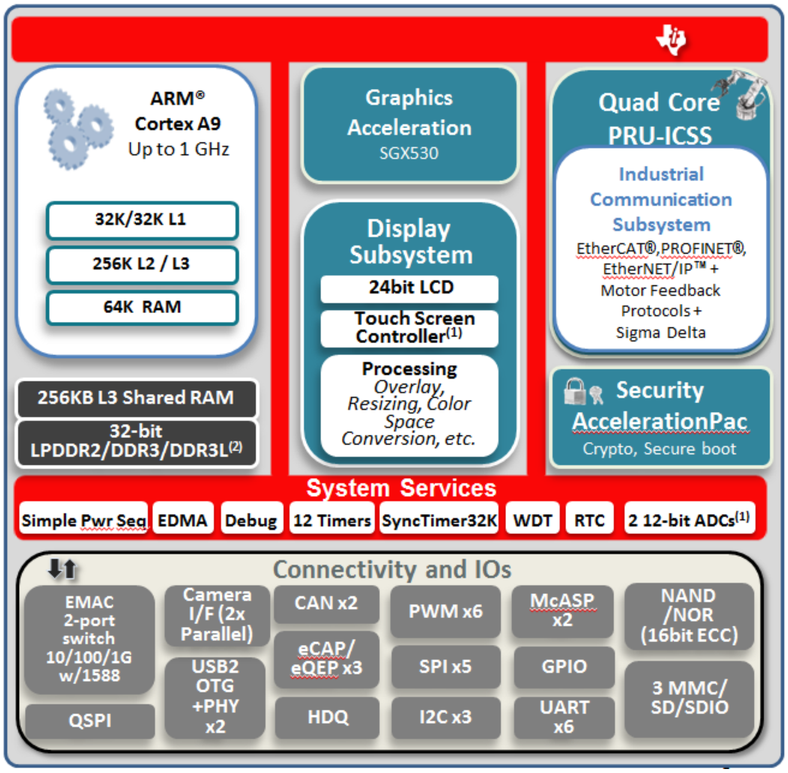


Figure 8. AM437x Functional Block Diagram

2.2.1 PRU-ICSS

PRU-ICSS is a separate processing unit from the ARM core that operates independently and clocks for greater efficiency and flexibility. PRU-ICSS enables additional peripheral interfaces and real-time protocols such as EtherCAT®, PROFINET®, EtherNet/IP™, PROFIBUS®, ETHERNET POWERLINK™, Sercos™, EnDat™, and others. The PRU core runs at 200 MHz and each PRU instruction takes 5 ns to execute, making it perfect for implementing real-time communication and control systems.

Figure 9 shows the functional block diagram of PRU-ICSS. The PRUs are used for interface functions in various modes. Real-time Ethernet protocols use the MII interface block which include translation from 4-bit to 16-bit data and crc32 verification. In GPIO mode, the PRU maps external I/O pins directly to registers R30 and R31. Serial-based protocols may also use the fast UART, which can be clocked for a 12-Mbaud data rate. BiSS protocol implementation uses direct GPIO mapping to PRU register. The AM437x processor has two PRU-ICSS subsystems. One subsystem is used for multiprotocol Industrial Ethernet. The second subsystem solves communication and control functions in power and sense applications such as servo drives. The memory configuration on application side is reduced to 4KB instructions. Industrial Ethernet peripheral (IEP) includes a time synchronization unit with many capture and compare registers. On the application side, this unit is used as the PLL application to synchronize PWM generation with current sensing and position sensing. For the BiSS-C interface master, the start of conversion for position measurement must compensate for line delay. The compare register in an IEP timer triggers an event with a 5-ns resolution. Basic signal processing functions are accelerated with a single-cycle hardware multiplier. Position data from the BiSS interface master interface can feed into the fast Fourier transform (FFT) to detect vibration of mechanical system.

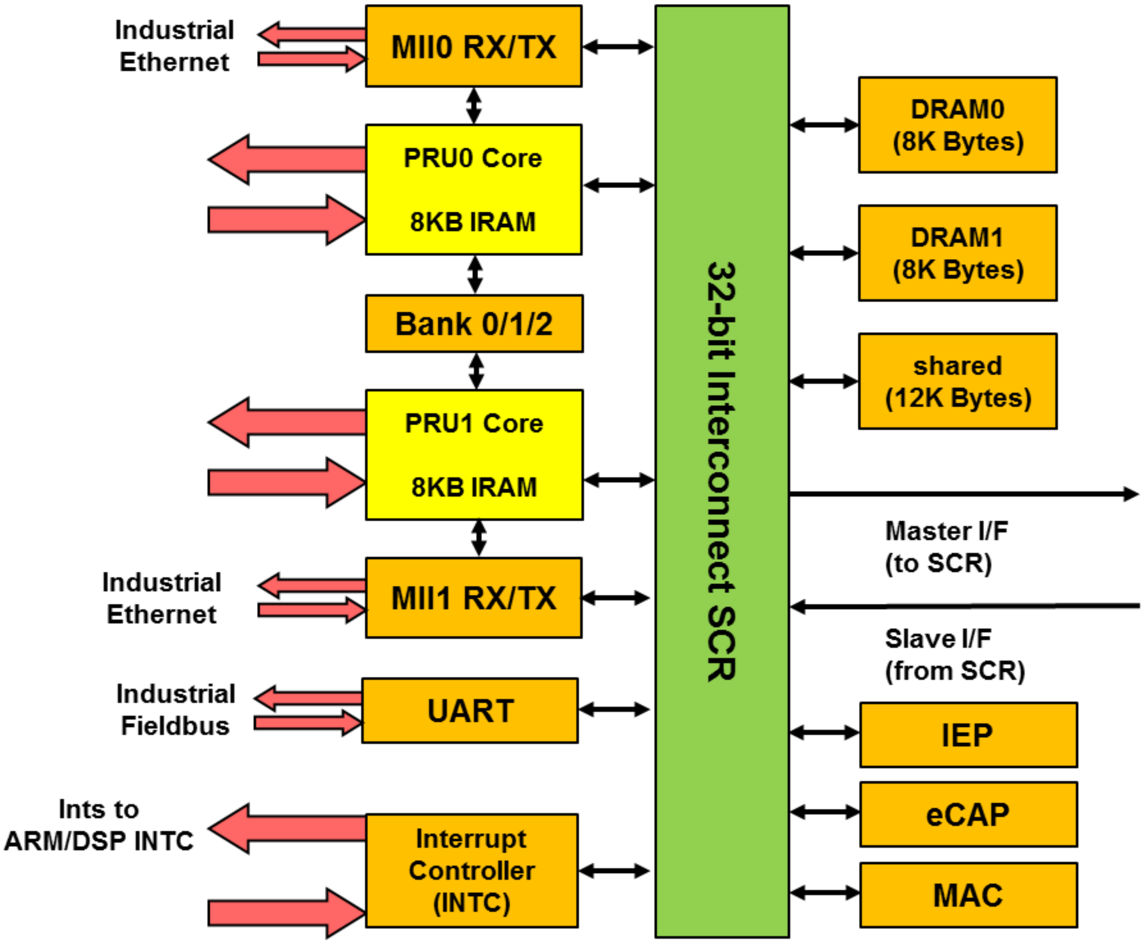


Figure 9. PRU-ICSS Block Diagram

2.3 RS485 Transceiver

The RS485 transceiver used in this design is the SN65HVD78. This device supports data rates up to 50 Mbps in a small package and a temperature range of –40°C to +125°C. The SN65HVD78 operates at 3.3 V with robust ESD and EMC protection. The device provides half-duplex communication interface for applications like factory automation, telecom infrastructure, and motion control. Figure 10 shows the package and logic diagram of the RS485 transceiver.

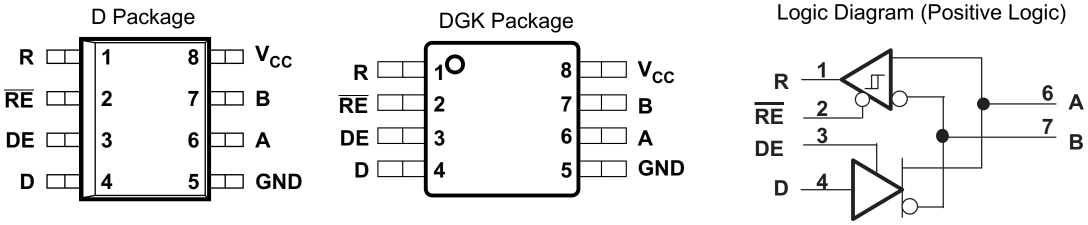


Figure 10. RS485 Transceiver – SN65HVD78

3 System Implementation

The BiSS-C master function is implemented on the TI Sitara AM437x IDK, which has an onboard RS485 transceiver for interfacing with the BiSS encoder. The design uses the PRU-ICSS on AM437x IDK for implementing the BiSS-C interface master firmware. The firmware is in PRU0 of ICSS0. The remaining PRUs in the AM437x IDK are available for industrial Ethernet communication or motor-control application.

Figure 11 outlines the system implementation. The ARM Cortex-A9 processor communicates over shared memory to the BiSS-C interface master on PRU-ICSS. A command interface in the shared memory is implemented and provides parameters on data, transmission, status, and errors for the communication.

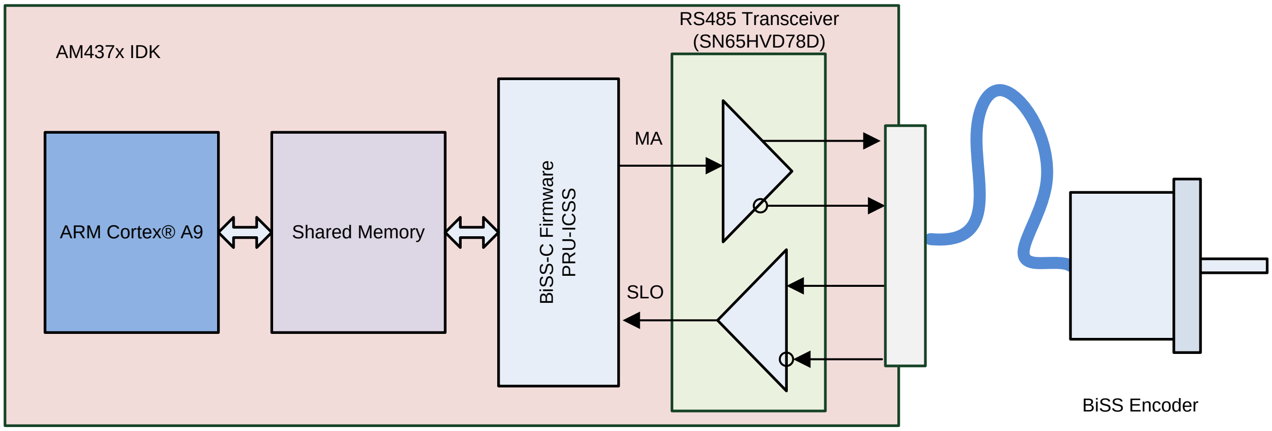


Figure 11. System Implementation

A data structure in the shared memory is used for the communication between the PRU_ICSS and the host.

PRU accesses shared memory through the PRU constant table entry C28. ICSS0 has no PRU shared memory at location 0x0001.0000, therefore C28 points to 0x0000.0000 (PRU0 local memory).

Table 1 shows the structure in the shared memory.

Table 1. Structure in Shared Memory

MEMORY LOCATION	ACCESS	NUMBER OF BITS	CONTENT
0x0000_0000	Write	8	Data length
0x0000_0001	Write	8	Number of bits CRC + nE +nW
0x0000_0002	Write	8	Frequency (supported frequencies: 1 = 1 MHz, 2 = 2 MHz, 5 = 5 MHz,10 = 10 MHz)
0x0000_0004	Write	32	Hex equivalent of control frame
0x0000_0008	Read	32	Position data higher word
0x0000_000C	Read	32	Position data lower word
0x0000_0010	Read	32	Control communication result
0x0000_0014	Read	16	Position data CRC error counts
0x0000_0016	Read	8	Error flags
0x0000_0018	Read	8	Position data CRC
0x0000_0019	Read	8	Control communication CRC
0x0000_001A	Read	8	Raw data byte with transition point
0x0000_001C	Read	16	Processing time requested by the slave
0x0000_001E	Read	8	Line delay

Table 2 shows the error flags at the memory location 0x0000_0016 that include various error and status flags.

Table 2. Status Flags

BIT POSITION	FLAG NAME	DESCRIPTION
0	SMA_FLAG	Set when simple moving average is available.
1	PROC_TIME_ERR	Set when process time exceeds the defined limit.
2	CMD_START_FLAG	Set when master starts sending control frame bits.
3	POS_CRC_ERR	Set when there is a position CRC error.
4	CMD_CRC_ERR	Set when there is a control frame CRC error.
5	NO_SLAVE_FLAG	Set when there is no slave connected.
6	EM_STOP_TEST	Set on emergency stop for testing.
7	Not used	

3.1 System Specifications

Table 3 lists the relevant system parameters. The hardware and software design support a maximum of 10-Mb data rate with a cable of up to 10 m. Lower-frequency steps allow for longer cables between master interface and encoder. A maximum length of 64 bits of data is supported in one communication cycle. Both position data and control data have a checksum. The complete BiSS-C interface master implementation takes 2.5KB out of 4KB instruction memory, leaving enough memory for customization and feature additions.

Table 3. System Specifications

PARAMETER	SPECIFICATION	DETAILS
Maximum frequency	10 MHz	Supported up to 10-m cable
Start-up/Initialization frequency	1 MHz	After power on or reset
Frequencies supported	1 MHz, 2 MHz, 5 MHz, 10 MHz	Can be switched during run time
Length of protocol	64 bits	Maximum number of bits
Position data CRC	6 bits	For position data verification
Position data CRC polynomial	0x43h	Polynomial used for position data verification
Control data CRC	4 bits	For control data verification
Control data CRC polynomial	0x13	Polynomial for control data verification
Firmware size	2.5KB	Total number of instructions 650

3.2 Firmware Implementation

The following section describes the firmware implementation of the BiSS-C interface master on PRU-ICSS. deterministic behavior of the 32-bit RISC core running at 200 MHz provides 5-ns resolution on sampling external signals and generating external signals. The firmware is written in modular block, reusing software components for different clock frequencies.

The high-speed data transmission with a maximum frequency of 10 MHz is implemented in the PRU-ICSS using a 28-bit shift mode for incoming data. 28-bit shift mode is used to sample a 1-data bit sent by the slave to multiple samples and select the most stable bit inside the oversampling window. Oversampling with a factor of 8 significantly improves the reliability of transmission with longer cables.

3.2.1 PRU 28-Bit Shift Mode

In the firmware, the GPI line for SLO is configured as the 28-bit shift mode. The GPI line is configured through the PRUSS_CFG registers. For more information, see Figure 12.

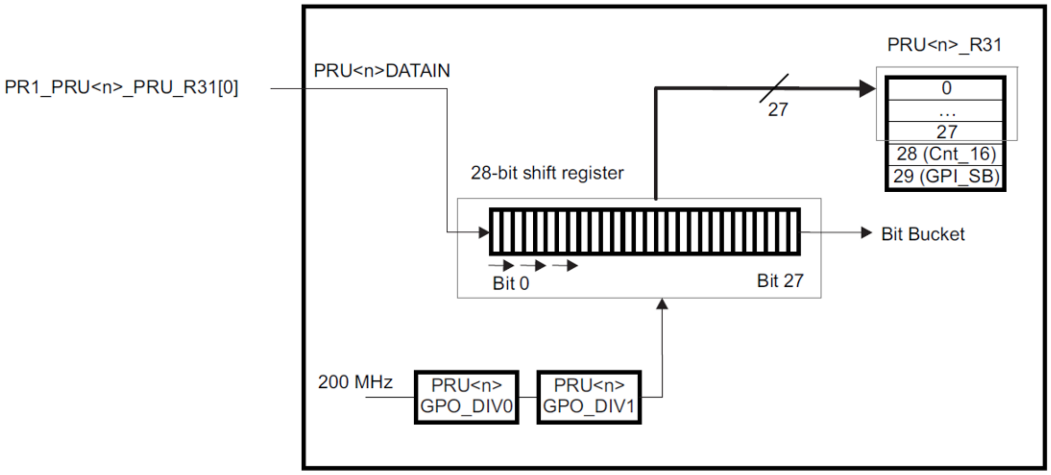


Figure 12. 28-Bit Shift Mode

In the 28-bit shift mode, pru<n>_r31_status [0] is sampled and shifted to the 28-bit shift register on an internal clock pulse. The register fills in LSB order (from bit 0 to 27) and then overflows into the bit bucket. The shift rate is controlled by the effective divisor of two cascaded dividers applied to the 200-MHz clock. These divisors are configured using PRUSS_CFG registers.

Figure 13 shows that the sampling frequency is calculated using the DIV0 and DIV1.

$$\text{Sampling Frequency} = \frac{200\text{ MHz} / \text{DIV0}}{\text{DIV1}}$$

Figure 13. Sampling Frequency

In this design, the 28-bit shift mode is configured to have eight samples for a single-slave data bit. Consequently, the fractional divisors cause the oversampling clock for the 28-bit shift mode to be eight times the BiSS-clock frequency. Table 4 lists the divisors and frequencies used for various BiSS clock frequencies using 8× oversampling. DIV0 and DIV1 can be set from 1 to 16 in increments of 0.5.

Table 4. Divisor Selection for 28-Bit Shift Mode

SERIAL NUMBER	BiSS CLOCK FREQUENCY	OVERSAMPLING FREQUENCY	DIVISOR0 (DIV0)	DIVISOR1 (DIV1)
1	1 MHz	8 MHz	0x17	0x2
2	2 MHz	16 MHz	0x17	0x0
3	5 MHz	40 MHz	0x8	0x0
4	10 MHz	80 MHz	0x3	0x0

The 28-bit shift mode fills register R31 from R31.t0 to R31.t27. In the configuration of eight times oversampling the first byte of the register R31, R31.b0 carries the samples for 1 slave data bit.

Figure 14 shows the oversampled data which resides in R31 byte 0. If the zoomed-in data bit is 1, then R31.b0 contains 1s.

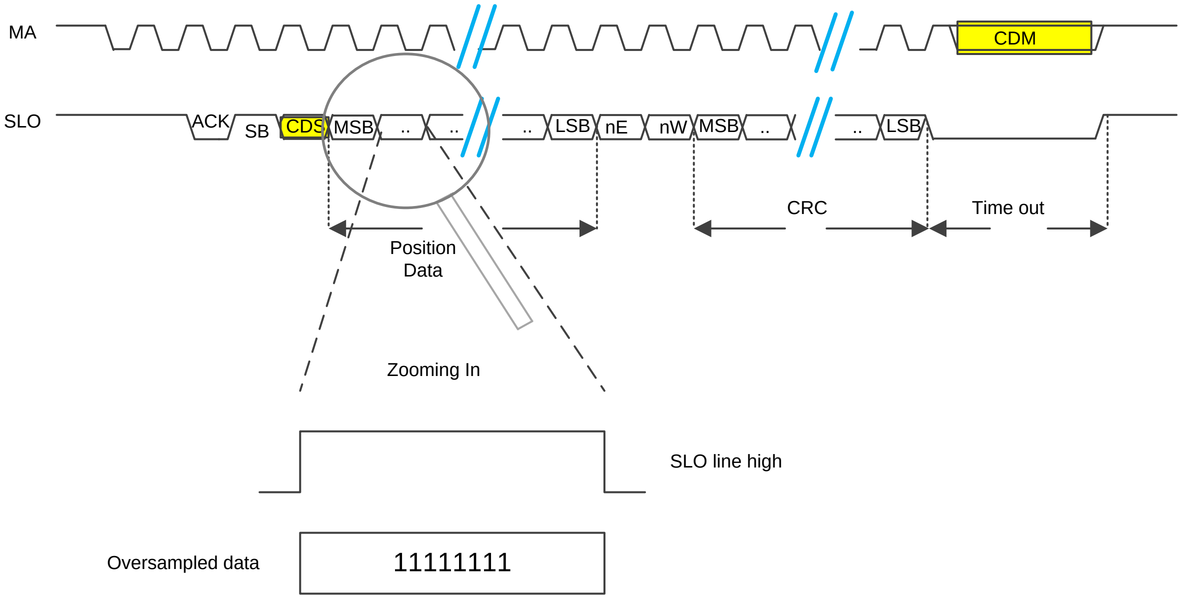


Figure 14. R31.b0 During Transmission

These eight samples are used to find the best sample point for data in the SLO line (slave-data bit). The first low-to-high transition on new communication cycle determines the sampling point in the middle of a bit window.

3.2.2 Supported Frequencies

To comply with BiSS-C interface protocol standards, the frequency for transmission varies with the length of the cable. The BiSS-C master firmware is configured to support different frequencies. The various frequencies that the firmware can support are given in the following table.

Table 5 lists the TI-recommended frequency values of the length of the cable. The host or user must select one of these frequencies before starting the transmission. The firmware then starts the transmission at the selected frequency.

Table 5. Supported Frequencies

SERIAL NUMBER	CABLE LENGTH	FREQUENCY
1	Up to 10 m	10 MHz
2	Up to 25 m	5 MHz
3	Up to 60 m	2 MHz
4	Up to 100 m	1 MHz

3.2.3 Software Files

This design limits the software components required to run the BiSS-C interface. There is no ARM side code needed to operate the interface. There are three types of files in this design.

- Gel scripts
- BiSS-C master firmware file
- Include files

The AM437x_BiSS_Gel folder contains the gel scripts that initialize the system. The AM43xx_BiSSConfig.gel file multiplexes the pins for the design and the receive line of the RS485 transceiver. Section 4 explains how to use the gel scripts.

The pru0_BiSS.p file is the BiSS-C master firmware file. The main functions and the subfunctions are implemented in this file.

The Include folder contains the header files and the macros in the design.

3.2.4 Firmware Architecture

The firmware has the following sections:

- Start-up section
- Multiple frequency section
- Optimized section for 10 MHz
- Post processing

The start-up section initializes some variables and does some start-up configuring. This section is only executed once during start-up.

The multiple frequency section includes several small blocks of codes. The timing of these blocks is scalable and configurable according to the selected frequencies. This section supports BiSS clock frequencies of 1 MHz, 2 MHz, and 5 MHz.

The optimized section for 10 MHz is a special section that is only for 10-MHz BiSS clock frequency. Whenever the host requests for transmission in 10-MHz BiSS clock frequency, the BiSS-C interface master executes this section of code.

The post processing section is the code where the BiSS-C interface master executes the post-transmission functions like control communication, CRC verification, and writing to the shared memory.

Figure 15 shows the program flow of the BiSS-C interface master firmware.

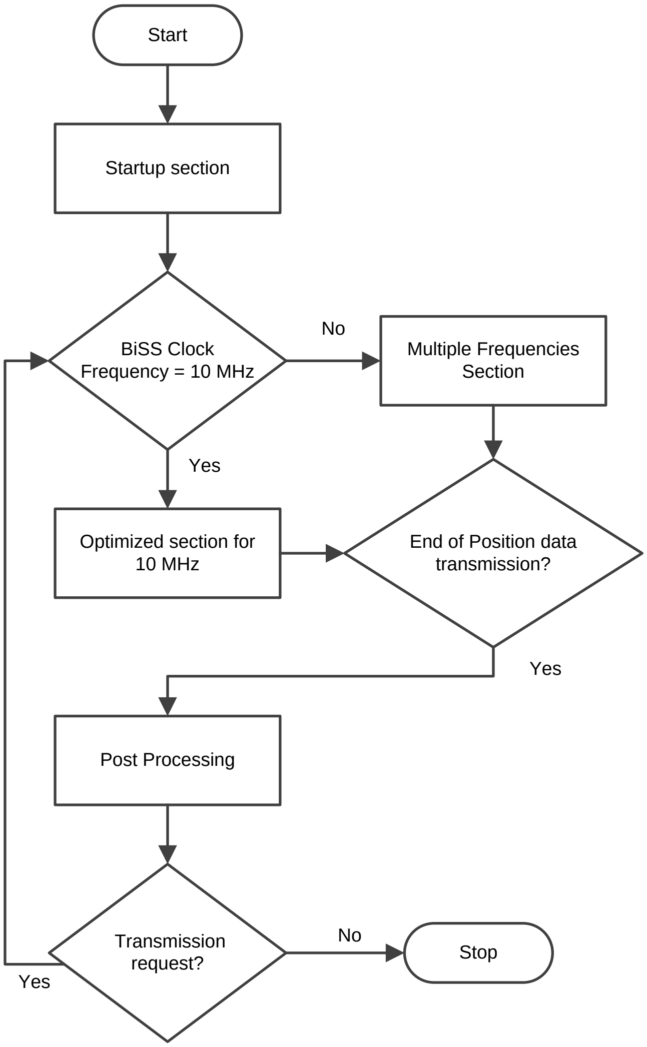

Start
Startup section
BiSS Clock Frequency = 10 MHz
No
Multiple Frequencies Section
Yes
Optimized section for 10 MHz
End of Position data transmission?
Yes
Post Processing
Transmission request?
No
Stop
Yes

Figure 15. Firmware Block Diagram

3.2.4.1 Start-Up Section

The main functionality of the start-up section is the initialization of variables and configuration of the PRU-ICSS. This section is executed only once after loading and starting the code.

The firmware checks whether the slave is connected to the master. The firmware detects the slave by checking whether the receive line is high after setting the clock line to high because the BiSS (SLO) line is high during reset. The firmware informs the host if the slave is disconnected with an error signal.

The first step in the start-up section is the configuration of industrial Ethernet peripheral (IEP) timer for triggering the transmission. The IEP timer is configured to trigger the transmission every 100 µs.

The start-up section also loads the data length (variable DATA_LENGTH), number of CRC, nE bits, and nW bits (variable CRC_ERR_WAR) from the shared memory. These values are configured by the host. The firmware uses the values to calculate the clock length (variable CLOCK_LENGTH) and the CDS bit number (variable CDS_BIT_NUMBER). Some of the other variables are also initialized in the start-up section like the bit number for the clock in the R30 register (variable CLK_BIT_NUMBER).

The start-up section executes a reset or start-up delay of 40 µs, which helps the slave maintain a break between the power on and the transmission.

The Figure 16 shows the flow chart of the start-up section.

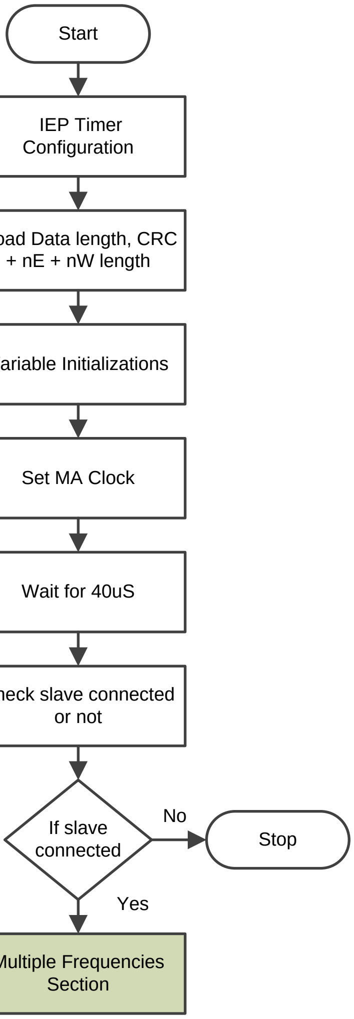


Figure 16. Start-up Section

When the start-up delay is completed, the firmware checks whether the slave is connected or not. If the slave is disconnected with the BiSS-C interface master, firmware sets a NO_SLAVE_FLAG error flag.

If the slave is connected, it executes the multiple frequency section once and provides a line delay to the host. In the start-up section, the BiSS-C interface master configures the multiple frequency section with an operating frequency of 1 MHz. When the start-up section completes, the master is ready for the position data and control data transmission.

After the start-up section, the host can use the line-delay value to select an appropriate operating frequency to execute the position data transmission.

3.2.4.2 Multiple Frequency Section

This section is executed whenever the BiSS clock frequency is less than 10 MHz. A TIME_DELAY variable scales the time duration and lets this section support multiple BiSS clock frequencies. This section is also built with a number of small blocks of codes which perform different tasks. These blocks are named based on their functionalities. These names can be found in the firmware in the same name. The 10-MHz BiSS clock frequency is handled by an optimized firmware section, which is similar to the multiple frequency section.

The multiple frequency section and the optimized section for 10 MHz generate the BiSS clock signal for the selected frequency. The clock signal is generated by the deterministic behavior of the PRU core.

In the firmware, the SET instruction sets the MA clock line and CLR instruction for clearing the MA clock line. Both instructions support bit-wise addressing and do not affect bits in the register. Each PRU instruction takes 5-ns execution time. To program a fixed frequency, certain numbers of instructions are executed between every SET and CLR instruction. In the multiple frequencies section, a TIME_DELAY variable makes the time delay between each SET and CLR scalable.

Figure 17 shows the tasks in the multiple frequency section.

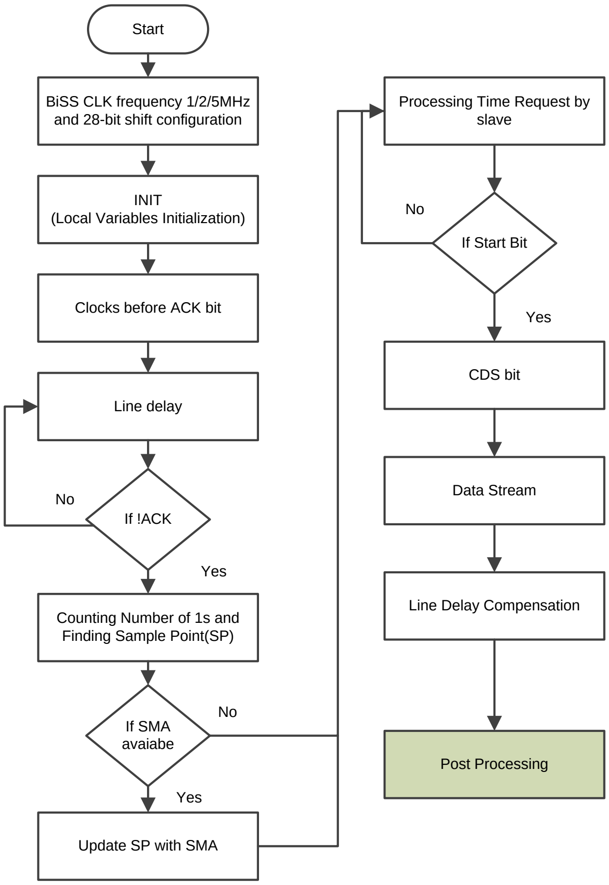

Start
BiSS CLK frequency 1/2/5MHz and 28-bit shift configuration
INIT (Local Variables Initialization)
Clocks before ACK bit
Line delay
No
If !ACK
Yes
Counting Number of 1s and Finding Sample Point(SP)
If SMA avaiabe
No
Yes
Update SP with SMA
Processing Time Request by slave
No
If Start Bit
Yes
CDS bit
Data Stream
Line Delay Compensation
Post Processing

Figure 17. Multiple Frequency Section

The following section describes tasks in the multiple frequency section.

3.2.4.2.1 BiSS CLK Frequency Configuration

The multiple frequency section starts with the configuration of BiSS clock frequency. The corresponding oversampling rate is also configured by the firmware. The oversampling rate (28-bit shift rate) is configured to be eight (×2) times the BiSS clock frequency. For one BiSS clock cycle, the data bits sent by the slave is sampled eight (×2) times and shifted into the R31 bit 0–8.

The following four code sections in the firmware configure the four different frequencies:

- BiSS_CLK frequency 1-MHz configuration
- BiSS_CLK frequency 2-MHz configuration
- BiSS_CLK frequency 5-MHz configuration
- BiSS_CLK frequency 10-MHz configuration

The two main functionalities of these blocks are to configure the TIME_DELAY variable based on the selected frequency and to configure the 28-bit shift mode.

By varying the TIME_DELAY variable, the firmware can automatically switch to a new frequency. This variable is used before every SET/CLR BiSS clock instruction that makes the time duration between BiSS clocks scalable.

When the host selects a new BiSS clock frequency, the 28-bit shift mode must be reconfigured. The 28-bit shift rate must always be eight times the BiSS clock frequency. The two cascaded divisors select an appropriate shift rate for the 28-bit shift mode. The DIV0 and DIV1 values for selecting shift rates, which are suitable for the BiSS clock frequencies in the implementation, are given in the 28-bit shift mode section of the document.

Table 6. 28-Bit Shift Mode Divisors

SERIAL NUMBER	DIV0	DIV1	VALUES TO BE WRITTEN IN HEX
1	1	1	0x0
2	1.5	1.5	0x1
3	2	2	0x2
4	2.5	2.5	0x3
5	3	3	0x4
6	3.5	3.5	0x5
7	4	4	0x6
8	4.5	4.5	0x7
9	5	5	0x8
10	5.5	5.5	0x9
11	6	6	0xA
12	6.5	6.5	0xB
13	7	7	0xC
14	7.5	7.5	0xD
15	8	8	0xE
16	8.5	8.5	0xF
17	9	9	0x10
18	9.5	9.5	0x11
19	10	10	0x12
20	10.5	10.5	0x13
21	11	11	0x14
22	11.5	11.5	0x15
23	12	12	0x16
24	12.5	12.5	0x17

Table 6. 28-Bit Shift Mode Divisors (continued)

SERIAL NUMBER	DIV0	DIV1	VALUES TO BE WRITTEN IN HEX
25	13	13	0x18
26	13.5	13.5	0x19
27	14	14	0x1A
28	14.5	14.5	0x1B
29	15	15	0x1C
30	15.5	15.5	0x1D
31	16	16	0x1E

For supporting additional BiSS frequencies and 28-bit shift rates, use Table 6. Table 6 lists various possibilities of the DIV0 and DIV1 and the corresponding hex values to write to the GPCFG_n (GPCFG0 or GPCFG1) registers.

3.2.4.2.2 Local Variables Initialization

The multiple frequency section has a local initialization block that initializes all the local variables that must be initialized before every position-data transmission. These variables are CLK_LENGTH, RESULT_REG, SHIFT_INDEX, PROCESS_TIME, and TEMP_REG to find the line delay.

The CLK_LENGTH variable is calculated using variables and constants like DATA_LENGTH, CRC_ERR_WAR, ACK, SB, and CDS. As the constants, the ACK, SB, and CDS bits require one BiSS clock per BiSS frame. The sum of these three constants is 3. CRC_ERR_WAR stands for the sum of the number of CRC bits, nE bits, and nW bits. This value is provided by the host before the start-up. The DATA_LENGTH variable is also provided by the host before start-up. This variable is the position-data length or the accuracy of the encoder. The sum of all these variables and constants gives the CLK_LENGTH value. The clocks before the ACK bit are not considered while calculating the value of the CLK_LENGTH. The calculated value of CLK_LENGTH is copied into a temporary register for the calculation of the line delay.

3.2.4.2.3 Clocks Before ACK Bit

The Clocks before ACK bit block sends the clocks before the ACK bit. The BiSS slave acknowledges at the second rising edge of the MA clock because of protocol. The firmware returns from this block after giving the second rising edge.

3.2.4.2.4 Line Delay

The Line-Delay block calculates the line delay of cable. The block checks whether the slave has asserted the ACK bit. If the BiSS-C interface master detects the ACK bit immediately after the clocks before ACK bit block, the firmware determines that there is no line delay due to the longer cables. Otherwise, the firmware executes Line Delay block and decrements the CLK_LENGTH for each BiSS clock cycle. As a result, the Line delay is calculated by subtracting the CLK_LENGTH value when the master detects the ACK bit from the temporary register whose value is initialized as the total number of clocks.

```
//Finding the Line delay in number of cycles
SUB                 TEMP_REG_1.b0, TEMP_REG_1.b2, CLK_LENGTH
```

The previous code is for finding the line delay.

Figure 18 shows how the line delay affects the transmission.

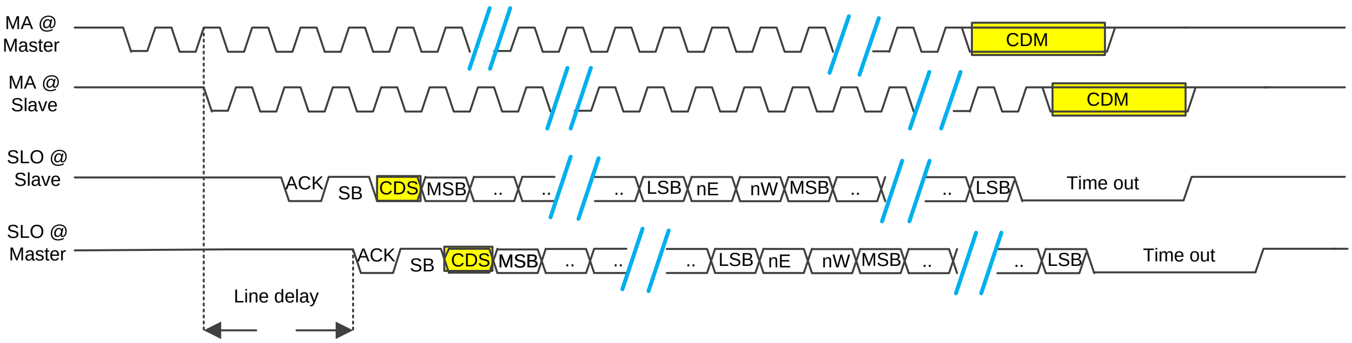


Figure 18. Line Delay

Until the slave asserts ACK signal low. all oversampled data bits in R30.b0 remain 1. When there is a transition from high to low, the oversampled data changes from 255 to a value less than 255. The firmware detects an ACK bit by checking if R31.b0 is equal to 255. When R31.b0 is unequal to 255, the firmware detects a transition from high to low in the SLO line.

After detecting the ACK bit, the block must to find the sample point from the last received byte in the Counting Number of 1s and Finding Sample Point block.

3.2.4.2.5 Counting the Number of 1s and Finding the Sampling Point

As per the BiSS clock frequency configuration block, every bit that the encoder sends is sampled eight times and saved in R31.b0. The most stable bit from the samples must be selected as the sample point. An algorithm is used to find the sample point.

The R31.b0 value is copied into a variable RAW_DATA after setting the BiSS clock line. Copying the R31.b0 value occurs at certain time intervals proportional to BiSS clock frequency. To find the sample point, firmware uses the RAW_DATA byte, which includes the transition from high to low (ACK bit).

To find the sample point, calculate the number of 1s in the RAW_DATA byte.

3.2.4.2.5.1 Counting the Number of 1s in a Byte

The number of PRU assembly instructions must always be the same between the SET and CLR instructions to maintain the clock frequency in the BiSS-C firmware. Firmware uses an algorithm that can efficiently calculate the number of 1s in a byte. The execution time for this algorithm is always the same and independent of bit combinations.

Figure 19 shows how this algorithm is implemented in the BiSS-C interface master firmware.

Nine blocks are used to find the number of 1s. For any bit combination, the algorithm can find the number of 1s using 10-PRU instructions.

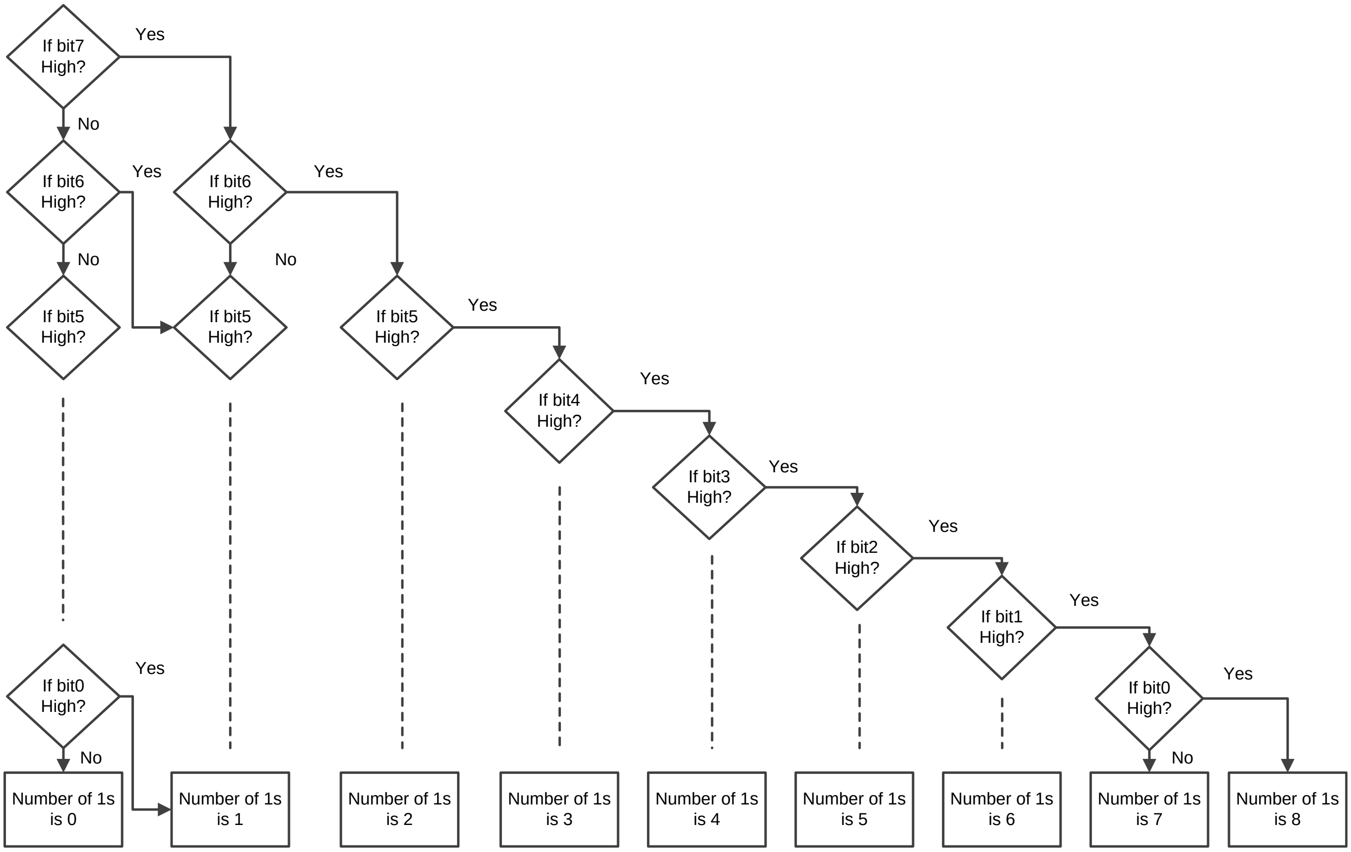


Figure 19. Counting Number of Ones in a Byte

The algorithm starts with checking bit 7. If bit 7 is high, the firmware skips the first block and jumps to the second block where firmware checks whether the bit 6 is high. And if the bit 6 is high, the firmware skips the second block and jumps to the third and so forth. The following code shows a snapshot of the PRU code.

```
no_of_ones:
QBBS count1, RAW_DATA.t7
QBBS count1.1, RAW_DATA.t6
QBBS count1.2, RAW_DATA.t5
QBBS count1.3, RAW_DATA.t4
QBBS count1.4, RAW_DATA.t3
QBBS count1.5, RAW_DATA.t2
QBBS count1.6, RAW_DATA.t1
QBBS count1.7, RAW_DATA.t0
MOV TEMP_REG_1.b1, 0
QBA sample_point
```

If the byte contains all ones, firmware executes only the first instruction of each block and then jump to the next and so forth to give the number of 1s as 8. Firmware exits from the algorithm. As a result, the algorithm executes exactly 10 instructions for this operation.

If the byte contains all 0s, the firmware does not return from the first block to any other blocks in the algorithm because all the conditions are false. Consequently, the result is 0. In this case, the algorithm also executes in 10 instructions.

The number of instructions executed is the same for other combinations of 1s and 0s.

The number of 1s in the byte is used to find the sample point. This sample point is a bit number or bit position, which is used for every data bit sent by the encoder.

3.2.4.2.5.2 *Finding the Sample Point*

Figure 20 shows the algorithm used to find the sample point.

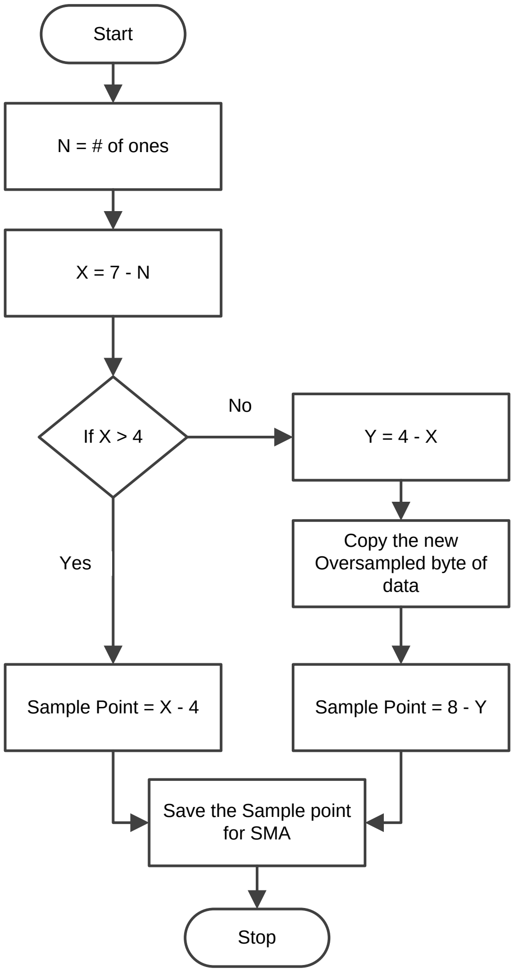


Figure 20. Algorithm to Find a Sample Point

NOTE: In the flow chart, variables like X, Y, and N are used. In the firmware, temporary registers are used.

To find X and Y, do the following:

1. Find X by subtracting the number of ones N in the RAW_DATA byte from 7.
2. Compare X with 4. If X is greater than 4, the sample point is the result of X – 4.

NOTE: If X is less than 4, the sample point is in the new oversampled byte.

3. Find Y by subtracting X from 4.

4. Copy the new oversampled byte from R31.b0 to RAW_DATA.
5. Determine the sample point by subtracting Y from 8.

NOTE: The sample point defines the bit that is representing the ACK bit.

6. Copy the sample point into a temporary register to find the simple moving average (SMA).

The following example shows how to calculate the SMA to help you understand the algorithm.

Consider the last 2 bytes are the following:

- Old byte = 11110101 (This byte contains the transition.)
- Latest byte = 00000000 (This byte is the latest byte.)

1. As old byte is not equal to 255, find the number of ones in it. The result is N = 6.
2. To find X, perform equation 1.
   $X = 7 – N$
   $X = 7 – 6$
   $X = 1$
3. Compare X with 4 to see whether X is less than 4.
4. To find Y, perform equation 2.
   $Y = 4 – X$
   $Y = 4 – 1$
   $Y = 3$
5. Copy the latest byte.

NOTE: The sample point is in the latest byte in this case.

6. Subtract Y from 8 to find the sample point.
   $SP = 8 – Y$
   $SP = 8 – 3$
   $SP = 5$

Figure 21 shows the result.

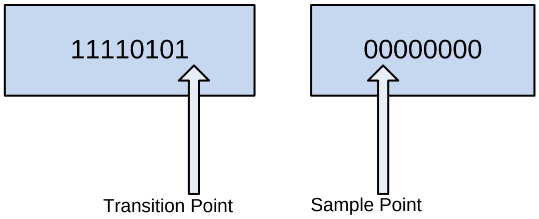


Figure 21. Example for Sample Point

Figure 21 shows the sample point is at bit position 5. After finding the sample point, the firmware checks only the bit at the bit position number SP from the 8 samples for every slave data bit. The firmware uses QBBS to find whether the bit at the sample point is high. The simple moving average (SMA) is a method for filtering the sample point that improves its accuracy.

3.2.4.2.6 Updating SP With SMA

This block checks whether the SMA is available and (if available) if the sample point is updated with the SMA.

A simple moving average filters the sample point in the BiSS-C interface master firmware. The subset size is 4. The last four sample points are averaged to find the SMA. When a new sample point is available, the oldest sample point is omitted and the newest sample point is considered by the SMA algorithm to find the SMA value.

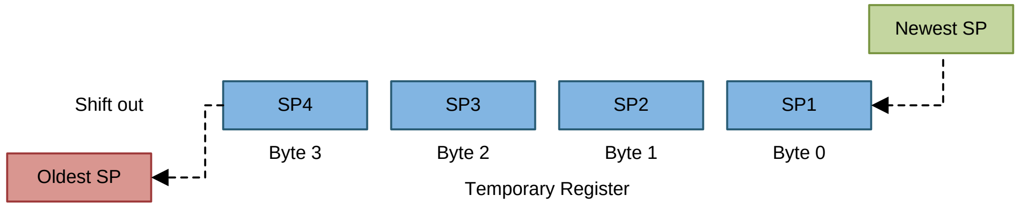


Figure 22. Simple Moving Average Algorithm

Figure 22 shows how the newest sample point is added to the temporary register and the oldest sample point is shifted out.

The following equation calculates the SMA.

$$SMA = \frac{(SP1 + SP2 + SP3 + SP4)}{4} \tag{1}$$

When the host selects a new BiSS clock frequency, the SMA is disabled for four cycles to collect the first four sample points. When enough numbers of sample point values are collected in a temporary register, firmware sets a SMA_FALG flag. The post processing section performs SMA calculation, enabling, and disabling.

The Update SP with SMA block checks whether the SMA is available. If the SMA is available, the block updates the sample point with SMA value.

3.2.4.2.7 Processing Time Request by Slave

The encoders can request additional processing time before sending data, for example, for A/D conversion or accessing memory. Slaves do this by delaying the start bit. If there is a process time request by the slave, ACK signal takes more than one BiSS clock. If the slave requests the processing time, the BiSS-C master must provide extra clock cycles.

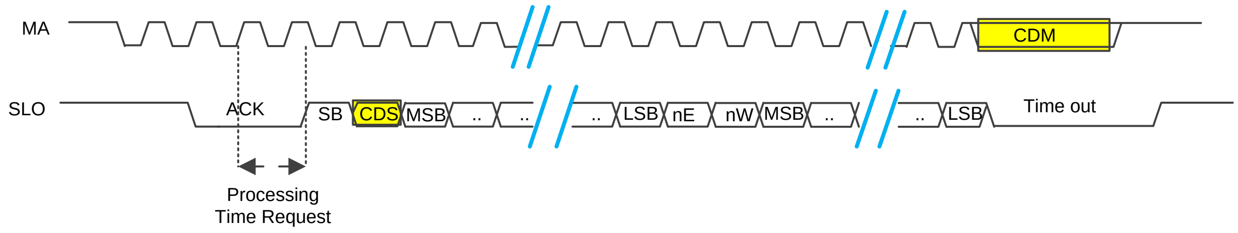


Figure 23. Processing Time Request by Slave

Figure 23 shows how the processing time is requested by delaying the start bit (SB). The maximum required processing time should be configured as the timeout period in the master. If the processing time exceeds this limit, the frame is cancelled.

The Processing Time Request by Slave block detects the processing time by checking the SB. After executing the Update SP with SMA block, firmware checks whether the bit at the filtered sample point is high or how. As per the protocol, SB is in high state. If the bit at the sample point is high, firmware detects the SB and it jumps to the CDS bit block. Otherwise, the Processing Time Request by Slave block is repeated until the firmware detects the SB. The PROCESS_TIME variable is incremented in every cycle to count the number of clock cycles.

If the slave requests additional processing time, the firmware provides extra clocks. The value of the PROCESS_TIME after detecting the SB is the total processing time requested by the slave. This value is counted as BiSS clock cycles is equal to the extra number of clock cycles from the firmware. If this value is greater than the value in the PROC_TIME_MAX variable, the BiSS frame is dropped.

3.2.4.2.8 Control Data Slave Bit

The next block is the Control Data Slave (CDS) bit. According to the protocol description, the CDS bit comes after the SB. The CDS bit is saved to the result register in this block.

This bit is also used in the control communication. When BiSS-C interface master firmware initiates a control communication, the CDS bits are extracted from the RESULT_REGs to CDS_RESULT register.

3.2.4.2.9 Data Stream

In the Data Stream block, the BiSS-C interface master firmware starts saving the position data, the nE bit, the nW bit, and CRC bits into the result registers.

For each BiSS clock cycle, 1 bit is saved and the CLK_LENGTH variable is decremented by 1. In this block, the firmware checks the bit at the SP is high or low. If the bit is high, the bit0 of the RESULT_REG1 register is set by the firmware and shifted one position left. If the bit is low, the bit0 is cleared by the firmware and shifted left.

A SHIFT_INDEX variable tracks the number of bits shifted into the result registers. If the variable is 31, the firmware copies the value of RESULT_REG1 to the RESULT_REG2 register .

This block is repeated until the value of the CLK_LENGTH variable is 0. When the value of the CLK_LENGTH variable is 0, the firmware returns from this block to the final block in the Line Delay Compensation block of the multiple frequency section.

3.2.4.2.10 Line Delay Compensation

The Line Delay Compensation block is the final block in the multiple frequency section. In this block, the firmware compensates the line delay. As per the Line Delay Compensation block, the calculated line delay value is stored in a temporary register. In the Line Delay Compensation block the temporary register is checked to determine if it is a line delay. If there is no line delay, the firmware exits the block to the post processing section. If there is line delay, the firmware repeats the Line Delay Compensation block for n times, where n is equal to the value of line delay. The firmware fails to provide any extra clocks to the slave in the Line Delay Compensation block. As there is a time delay in receiving data bits, the firmware only must read in the data bits in correct intervals without giving any extra BiSS clocks. Copying the R31.b0 to the RAW_DATA byte must be in the same intervals of time as the DATA_STREAM block. With this sequence, the delayed bits can be read without missing any data bits sent by the slave.

3.2.4.2.10.1 Optimized Section for 10 MHz

This section of code is similar to the multiple frequency section. This section is optimized to support only 10 MHz. Because each instruction in PRU takes 5 ns to generate 10-MHz clock using PRU, the number of instructions between each SET and CLR instructions must be 9. Because some of the blocks in the multiple frequency section have more than 10 instructions, a separate block of code is required to support 10-MHz BiSS clock frequency.

The main difference between the multiple frequency section and optimized section for 10 MHz is that the optimized section for 10 MHz is optimized only for 10 MHz and it does not use scalable TIME_DELAY variable for supporting any other BiSS clock frequencies. Because there are not enough spare cycles to check whether the SMA is available in the optimized section for 10 MHz block, SMA is disabled for the 10 MHz. This is due to the Counting Number of 1s and Finding Sample Point block. It can be resolved using a different method to optimize the counting number of 1s. A look-up table can be used to find the number of 1s in the byte. The shared memory can be used to precalculate the lookup table with 256 entries (256 bytes). Each byte includes the number of 1-s value. An INDEX variable can be used to point to the shared memory location.

Example code follows.

```
        //Finding the Line delay in number of cycles
            SUB         TEMP_REG_1.b0, TEMP_REG_1.b2, CLK_LENGTH
```

This method can reduce the number of instructions required for the calculation of the number of ones in a byte with the expense of 256 bytes of memory from the shared memory. Figure 24 shows the flow chart for the optimized section for 10-MHz.

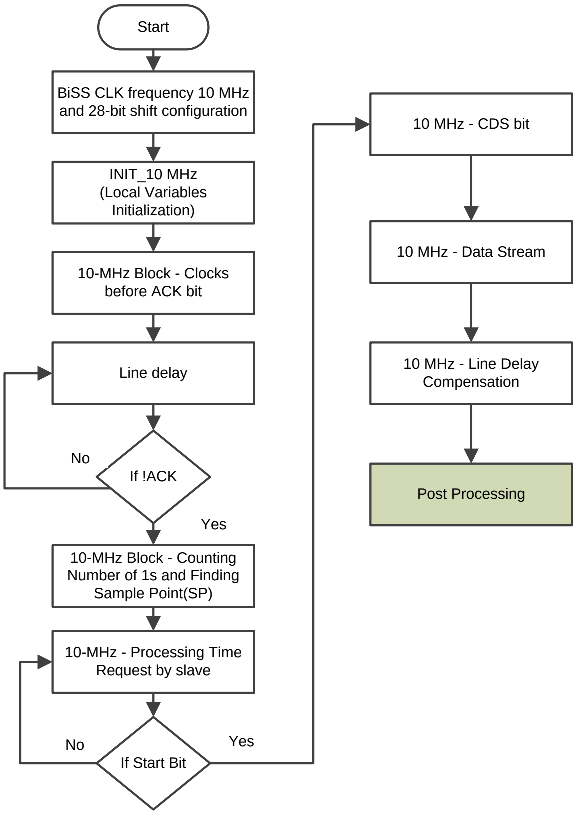

Start
BiSS CLK frequency 10 MHz and 28-bit shift configuration
INIT_10 MHz (Local Variables Initialization)
10-MHz Block - Clocks before ACK bit
Line delay
No
If !ACK
Yes
10-MHz Block - Counting Number of 1s and Finding Sample Point(SP)
10-MHz - Processing Time Request by slave
No
If Start Bit
Yes
10 MHz - CDS bit
10 MHz - Data Stream
10 MHz - Line Delay Compensation
Post Processing

Figure 24. Optimized Section for 10 MHz

The optimized section for 10 MHz also jumps to the post processing section after receiving all the data bits from the slave.

3.2.4.2.10.1.1 Post Processing

The main tasks of post processing include the following:

- Control communication
- 4-bit CRC verification
- Frequency selection
- SMA calculation
- Position data extraction
- 6-bit CRC verification
- Time-out
- Error signaling

Figure 25 shows the post processing section flow chart.

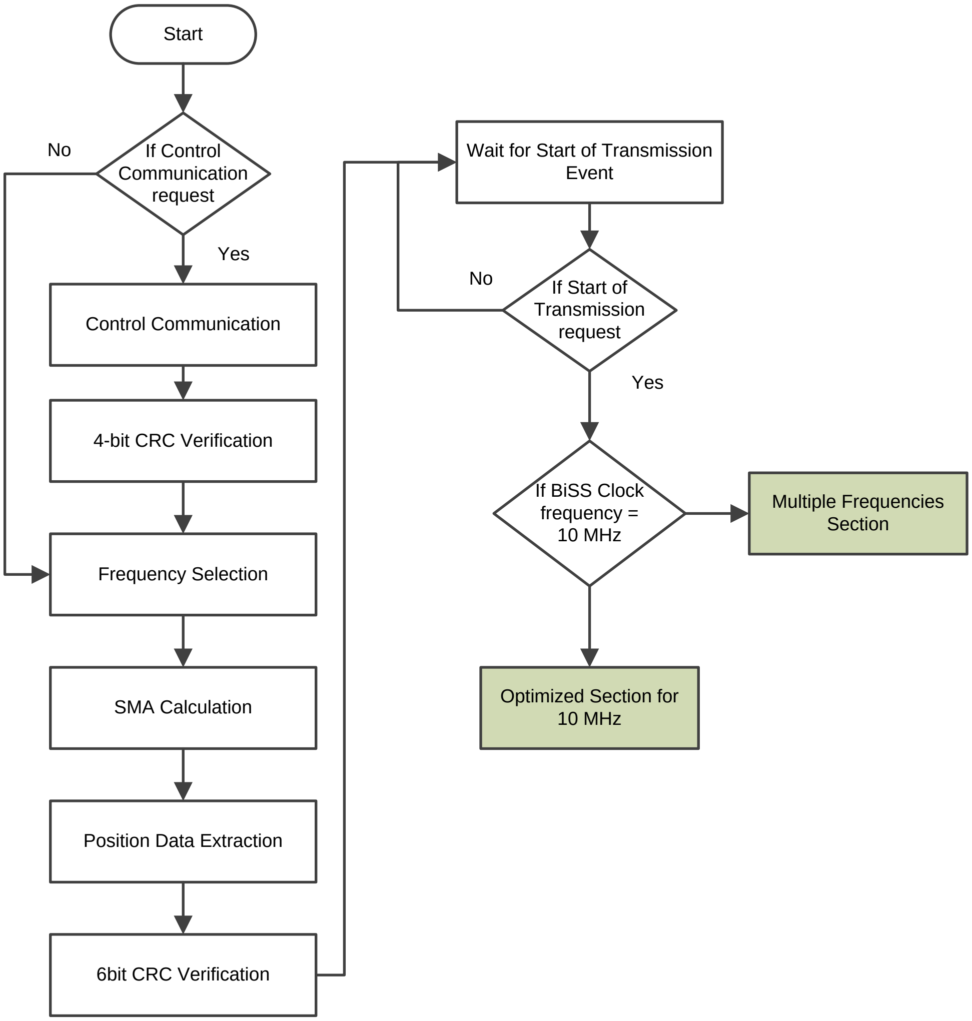

Start
If Control Communication request
No
Yes
Control Communication
4-bit CRC Verification
Frequency Selection
SMA Calculation
Position Data Extraction
6bit CRC Verification
Wait for Start of Transmission Event
If Start of Transmission request
No
Yes
If BiSS Clock frequency = 10 MHz
Multiple Frequencies Section
Optimized Section for 10 MHz

Figure 25. Post Processing Section

3.2.4.2.11 Control Communication

For the control communication, CDM bits must be sent during the time out period of each BiSS frame. Control communication occurs in the post processing section. Before starting to send a control frame, a series of at least 14 CDM equals 0 bits must be sent. The first CDM equals 1 bit after the 14 CDM equals 0 bits is the SB of the control frame. The CTS equals 1 bit is sent for slave register read/write. The CDM bit is sent inverted on the MA clock line. In the BiSS-C interface master firmware implementation, the control frame (CDM bits) is precalculated in equivalent hex value. A memory location is defined for the hex equivalent of control frame in the shared memory location. When the host provides a hex equivalent of a control frame in the defined memory location, firmware executes control communication and puts the result in the defined location for the control communication result. Table 7 lists hex equivalents for various control frames and the meaning of each control frame.

Table 7. Hex Equivalent of Control Frame

Serial No.	Operation	Hex Equivalent	Meaning
1	Read at 0x44	0x889B4000	Read 32-bit serial number
2	Read at 0x45	0x88BD4000	Read 32-bit serial number
3	Read at 0x46	0x88D74000	Read 32-bit serial number
4	Read at 0x47	0x88F14000	Read 32-bit serial number
5	Read at 0x60	0x8C174000	Read status bits of encoder
6	Write to 0x60	0x8C13C194	Send commands to the encoder
7	Read at 0x67	0x8CE54000	Read internal temperature
8	Read at 0x68	0x8D014000	Read error/warning messages
9	Read at 0x78	0x8F0B4000	Read 48-bit device ID
10	Read at 0x79	0x8F2D4000	Read 48-bit device ID
11	Read at 0x7A	0x8F474000	Read 48-bit device ID
12	Read at 0x7B	0x8F614000	Read 48-bit device ID
13	Read at 0x7C	0x8F934000	Read 48-bit device ID
14	Read at 0x7D	0x8FB54000	Read 48-bit device ID

Figure 26 is an example control frame calculation for reading from register at address 0x67 to understand the method of how to calculate the control frame hex equivalent.

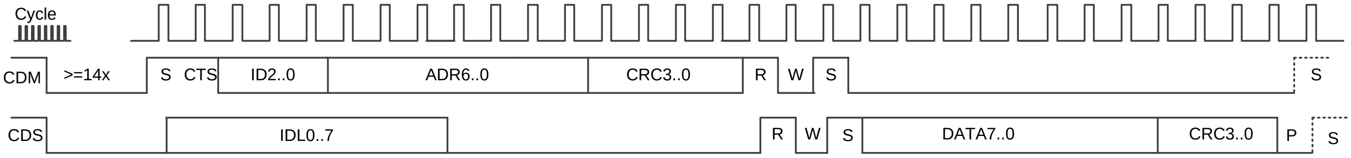


Figure 26. Register Read Access

Figure 26 shows the register read access. The first CDM equals 1 bit after a series of at least fourteen instances of CDM equals 0 is the start bit of the control communication. This start bit is not included in the calculation of the hex equivalent of the control frame. The CDM bits starting from the CTS bit are considered for the calculation of the hex equivalent. For register Read/Write operation, bit CTS equals 1. The following three bits are the IDS bits. As the BiSS-C interface master firmware implements only point-to-point communication, only one slave is connected to the master. So the ID2..0 equals 000. Following the IDS bits, the register must be addressed. This addressing occurs using the 7-bit addressing. The register at 0x67 can be written in binary as 1100111. Excluding the start bit, all other bits are covered with CRC. The bits from CTS to the 7-bit register address are used for calculating the 4-bit CRC. The polynomial used for the CRC calculation is 0x13. For CRC calculation, the bits can be written as 10001100111. 4-bit CRC can be determined manually using the algorithm used to verify the 4-bit CRC verification, which is described later in this document. After finding the CRC, the inverted CRC bits should be appended with the other bits. As a result, the bit sequence 10001100111 the 4-bit CRC is 1101. As the CRC bits are sent inverted on the line, the inverted CRC bits 0010. The inverted CRC bits are appended with the previous bits that result in 100011001110010. The next three bits are R, W, and S bits. R bit is high for the register read access. RWS equals 101. W bit is low and S stands for the start bit and is high.

The bit stream is 100011001110010101. Thirteen zeroes are appended including a stop bit. Another zero is added to these bits to disable the sequential register access. The final bit sequence is 10001100111001010100000000000000. The hex equivalent of these bits 1000 1100 1110 0101 0100 0000 0000 0000 = 0x8CE54000.\ Read access at 0x67in hex = 0x8CE54000 is how the hex equivalents for all the register accesses are determined.

3.2.4.2.12 4-Bit CRC Verification

The received control data is also protected with a 4-bit CRC. The received control data must be verified using the algorithm used for finding the 4-bit CRC. For the verification of the control data, Figure 27 shows three 32-bit temporary registers. Consider TEMP_REG1 for the data, TEMP_REG2 for the bit number of left most 1 in the TEMP_REG1 register, and TEMP_REG3 for CRC polynomial. The CRC polynomial for the control communication is 0x13.

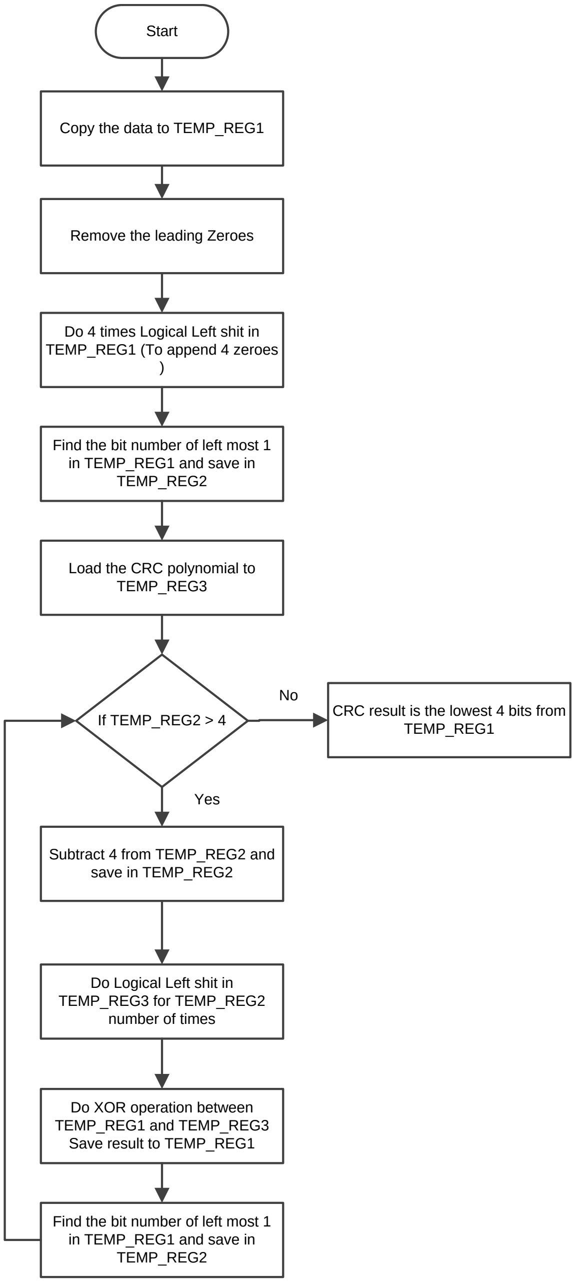

Start
Copy the data to TEMP_REG1
Remove the leading Zeroes
Do 4 times Logical Left shit in TEMP_REG1 (To append 4 zeroes)
Find the bit number of left most 1 in TEMP_REG1 and save in TEMP_REG2
Load the CRC polynomial to TEMP_REG3
If TEMP_REG2 > 4
No
CRC result is the lowest 4 bits from TEMP_REG1
Yes
Subtract 4 from TEMP_REG2 and save in TEMP_REG2
Do Logical Left shit in TEMP_REG3 for TEMP_REG2 number of times
Do XOR operation between TEMP_REG1 and TEMP_REG3 Save result to TEMP_REG1
Find the bit number of left most 1 in TEMP_REG1 and save in TEMP_REG2

Figure 27. Algorithm for 4-Bit CRC

Consider 8 bits 01010111. After removing the leading zeroes, the 1010111 appends four 0s. The resulting data is 10101110000. Perform the XOR operation between data and the CRC polynomial 0x13 (10011).

```
10011)      10101110000
            10011
            .................
            00110110000
              10011
            .................
              010000000
               10011
            .................
               00011000
                  10011
            .................
                  01011
```

The CRC result is 1011. The slave sends the CRC bits inverted on the line. For verifying the CRC, the CRC bits must be inverted and compared with the CRC bits sent by the slave. The inverted CRC bits for the previous example are 0100. The BiSS-C interface master firmware checks whether the CRC bits received by the master are equal to the CRC bits calculated. If both are not the same, firmware sets an error flag CMD_CRC_ERR to inform the host.

3.2.4.2.13 Frequency Selection

After the completion of BiSS fame, the BiSS-C interface master reads the defined memory location, which specifies the operational BiSS clock frequency provided by the host. When the cable length is short, the host selects a higher frequency. When the cable is long, the host selects a lower frequency. In the current firmware implementation, the host can chose from the list of available frequencies listed in Table 2.

3.2.4.2.14 SMA Calculation

The next task of the post processing section is to calculate the SMA. The firmware first checks the availability of the SMA using a flag called SMA_FLAG. When the host or user selects a new frequency, the SMA is disabled for the first four BiSS cycles. The SMA is enabled starting with the fifth BiSS cycle. After the fourth BiSS cycle, enough sample points exist to calculate SMA. When the SMA is available, this SMA calculation block sets the SMA_FLAG. The SMA_FLAG flag is used to decide whether to update SMA in the Update SP with SMA block in the multiple frequency section. For a detailed description about the calculation of SMA, see the Update SP with SMA block in the multiple frequency section.

3.2.4.2.15 Position Data Extraction

An important task of the firmware is extracting the position data. This extraction occurs in the post processing section. After executing the multiple frequency section or optimized section for 10 MHz, the position data bits, CDS bit, nE, nWn and 6 bits of CRC are stored in the result registers. The task of the position data extraction block is to extract only the position data bits from the result registers and provide the host with the position data result and verify it. The CDS bit is cleared using the CDS_BIT_NUMBER variable. This CDS_BIT_NUMBER is the variable that includes the CDS bit position in the result register. This variable is calculated in the start-up section. In the next step, the 6 bits of CRC and the nE, nW bits are shifted out from the result register and the remaining bits are saved in a different result register. The new result register contains only the position data. This data is stored into the defined memory location in the shared memory by the firmware for informing host. The position data received by the BiSSC interface master must be verified using the 6-bit CRC. For the calculation of 6-bit CRC, SB is not considered.

3.2.4.2.16 6-Bit CRC Verification

The 6-bit CRC is used for protecting the position data transmission. The CRC polynomial used is 0x43. The algorithm used for the 6-bit CRC is similar to the 4-bit CRC verification. Figure 28 shows the 6-bit CRC verification flow chart.

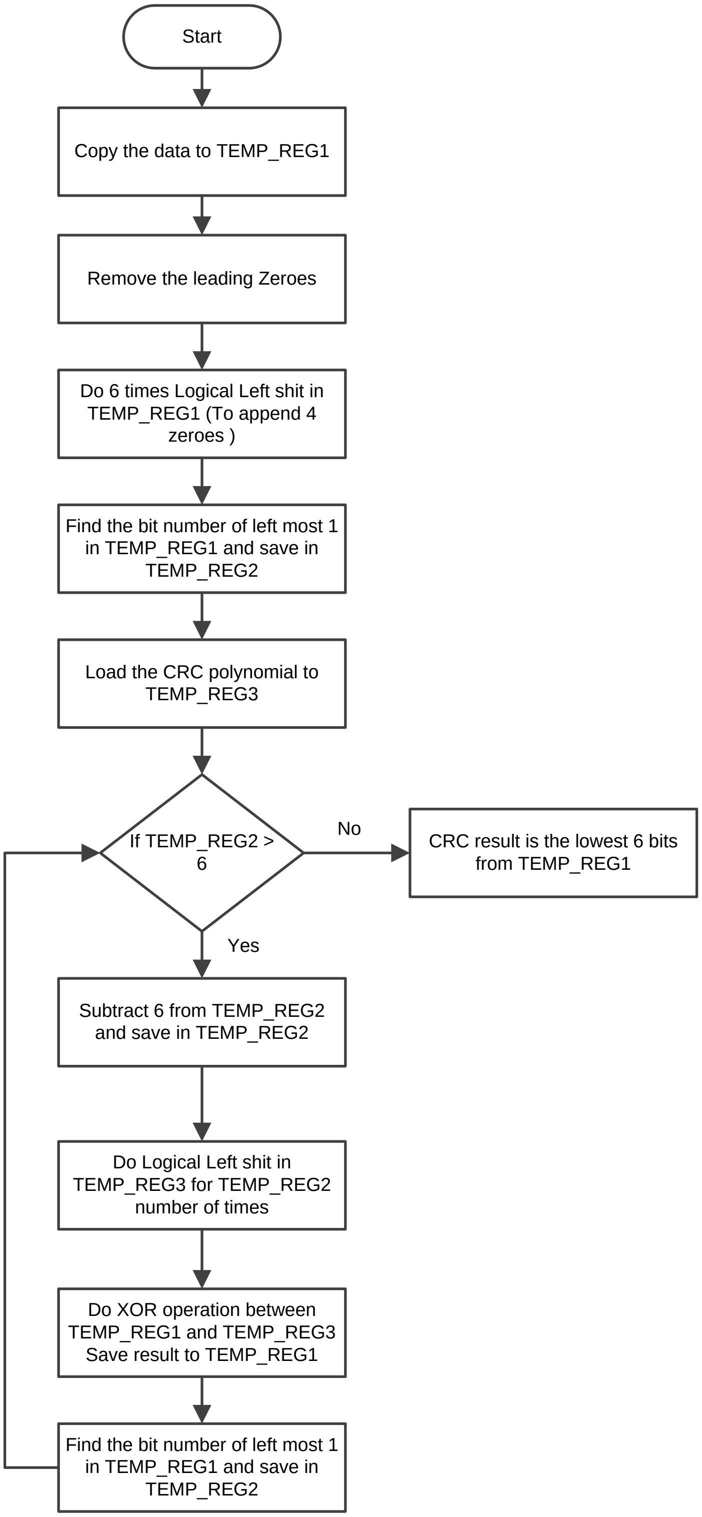

Start
Copy the data to TEMP_REG1
Remove the leading Zeroes
Do 6 times Logical Left shit in TEMP_REG1 (To append 4 zeroes)
Find the bit number of left most 1 in TEMP_REG1 and save in TEMP_REG2
Load the CRC polynomial to TEMP_REG3
If TEMP_REG2 > 6
No
CRC result is the lowest 6 bits from TEMP_REG1
Yes
Subtract 6 from TEMP_REG2 and save in TEMP_REG2
Do Logical Left shit in TEMP_REG3 for TEMP_REG2 number of times
Do XOR operation between TEMP_REG1 and TEMP_REG3 Save result to TEMP_REG1
Find the bit number of left most 1 in TEMP_REG1 and save in TEMP_REG2

Figure 28. 6-Bit CRC Verification

Consider 00000000000101. Remove the leading 0s so the data equals 101. Append six 0s. The data equals 101000000. Perform an XOR operation between data and the CRC polynomial 0x43 (1000011).

```
1000011)    10101110000
            1000011
            ………………..
            00101000000
              1000011
               …………….
              001001100
                1000011
                ………….
                0001111
```

The CRC result is 001111. The slave sends the CRC bits inverted on the line. For verifying the CRC, the CRC bits are inverted and then compared with the CRC bits sent by the slave. The resulting inverted CRC bits for the example are 110000.

The BiSS-C interface master checks whether the CRC bits received by the master is equal to the CRC bits calculated. If the master and CRC bits are not the same, the firmware sets an POS_CRC_ERR error flag and also increments the CRC error counter CRC_ERR_COUNTER by 1.

3.2.4.2.17 Wait for Event

This block is used for waiting the event from the IEP timer. At this time, the event is not mapped to the host processor. The IEP_CMP_STATUS_REG is read and the event status bit is used to check the counter overflow. If the status bit is set, the firmware clears the status bit and starts the next position data transmission.

3.2.5 Register Definition

Register definition shows the PRU register map. This register map has a header file that includes all the register definitions used for the firmware. Table 8shows the register map used for the project.

Table 8. PRU Register Map

<table>
<tr><td colspan="2">Status</td><td colspan="6">Oversampled Data</td><td>R31</td></tr>
<tr><td colspan="7">Not used</td><td colspan="1">GPO</td><td>R30</td></tr>
<tr><td colspan="5">Not used</td><td colspan="3">Call Register</td><td>R29</td></tr>
<tr><td colspan="8" rowspan="4">Reserved for Multiplier with Optional Accumulation</td><td>R28</td></tr>
<tr><td>R27</td></tr>
<tr><td>R26</td></tr>
<tr><td>R25</td></tr>
<tr><td colspan="8">Not used</td><td>R24</td></tr>
<tr><td colspan="8">NO_OF_CYCLES</td><td>R23</td></tr>
<tr><td colspan="4">PROCESS_TIME</td><td colspan="2">Not used</td><td colspan="2">SP_COUNTER</td><td>R22</td></tr>
<tr><td colspan="2">JUMP_TO_FREQ</td><td colspan="2">FREQ_SELECT</td><td colspan="4">CRC_ERR_COUNTER</td><td>R21</td></tr>
<tr><td colspan="8">CMD_INIT_COUNT</td><td>R20</td></tr>
<tr><td colspan="2">TIMEOUT_DURATION</td><td colspan="2">CDS_BIT_NUMBER</td><td colspan="2">CRC_ERR_WAR</td><td colspan="2">DATA_LENGTH</td><td>R19</td></tr>
<tr><td colspan="8">CDS_RESULT</td><td>R18</td></tr>
<tr><td colspan="8">CMD_REG</td><td>R17</td></tr>
<tr><td colspan="8">CRC_POLY</td><td>R16</td></tr>
<tr><td colspan="8">MASK_CRC</td><td>R15</td></tr>
<tr><td colspan="8">RESULT_REG3</td><td>R14</td></tr>
<tr><td colspan="8">RESULT_REG2</td><td>R13</td></tr>
<tr><td colspan="8">RESULT_REG1</td><td>R12</td></tr>
<tr><td colspan="8">TEMP_REG_6</td><td>R11</td></tr>
<tr><td colspan="8">TEMP_REG_5</td><td>R10</td></tr>
<tr><td colspan="8">TEMP_REG_4</td><td>R9</td></tr>
<tr><td colspan="8">TEMP_REG_3</td><td>R8</td></tr>
<tr><td colspan="8">TEMP_REG_2</td><td>R7</td></tr>
<tr><td colspan="8">TEMP_REG_1</td><td>R6</td></tr>
<tr><td colspan="2">Not used</td><td colspan="2">BiSS_CLK_BIT</td><td colspan="4">PROCESS_TIME_MAX</td><td>R5</td></tr>
<tr><td colspan="2">CMD_BIT_PTR</td><td colspan="2">CRC_LENGTH</td><td colspan="2">RAW_DATA</td><td colspan="2">Error flags</td><td>R4</td></tr>
<tr><td colspan="8">TIME_DELAY</td><td>R3</td></tr>
<tr><td colspan="2">SMA</td><td colspan="2">Not used</td><td colspan="2">SHIFT_INDEX</td><td colspan="2">CLK_LENGTH</td><td>R2</td></tr>
<tr><td colspan="8">Reserved for MVI</td><td>R1</td></tr>
<tr><td colspan="8">Reserved</td><td>R0</td></tr>
</table>

3.2.6 BiSS-C Hardware Interface

The physical data transmission in BiSS-C is done using the RS-485 standard. The data is transmitted as differential signals using the RS-485 transceiver between the BiSS-C master and the encoder. The master sends the clock to the BiSS-C encoder and the slave responds to this clock. The design uses two differential signals for each of the lines (clock and data).

3.2.6.1 RS-485 Transceiver

BiSS-C interface master and the BiSS encoder is connected using the RS-485 transceiver. According to the BiSS protocol description, the RS-422 device can also be used for the physical layer. The RS-485 transceiver allows transmission up to a frequency of 10 MHz. Data is transmitted differentially over RS-485. This transceiver has high noise immunity and long distance transmission capabilities. In the BiSS-C interface master implementation, onboard SN65HVD78 RS-485 transceiver is used. The receiver line in the RS-485 transceiver is enabled using the GPIO (GPIO5 [12]).

3.2.6.2 AM437x Pin-Multiplexing for BiSS-C Interface

The pin-multiplexing occurs through the gel script. Mainly two pins are used for the transmission of BiSS-C frame. One pin is used for the clock (CLK) and the other for the slave data output (SLO). mcasp0_aclkx is used for the SLO input, which is configured as the 28-bit shift register mode in PRU-ICSS. mcasp0_fsx is used for the clock output, which is configured as the GPO in PRU-ICSS. Two more pins are used for enabling the RS485 transceivers. Table 9 lists the pin-mux for BiSS-C interface.

Table 9. AM437x Pin Multiplexing for BiSS-C Interface

Pin Name	Signal Name	Mode	Offset	Function
mcasp0_aclkx	pr0_pru0_gpi[0]	6	0x0990	SLO input to AM437x
mcasp0_fsx	pr0_pru0_gpo[1]	5	0x0994	MA clock output to the encoder
endat_en	gpio5[12]	7	0x0a48	For enabling receive line in RS485 transceiver

3.2.6.3 AM437x IDK Hardware Modifications

Hardware changes on the AM437x board are required to support the BiSS-C interface. For the receive line, 28-bit shift register mode of PRU is used. For the 28-bit shift mode, PRU register R31 pin 0 must be used [R31.0]. As the default, the R31.0 pin is used for ENDAT_CLK in AM437x. It must be modified to support the 28-bit shift mode for the BiSS-C firmware. Perform the following modifications:

1. Connect AM437X_PRU0_ENDAT0_CLK (U11, N24) to U57, pin 1 through a blue wire.
2. Remove R428, R427, and R514.
3. Mount 0 Ω for R422.

Figure 29 shows a snapshot of these resistors from the schematics.

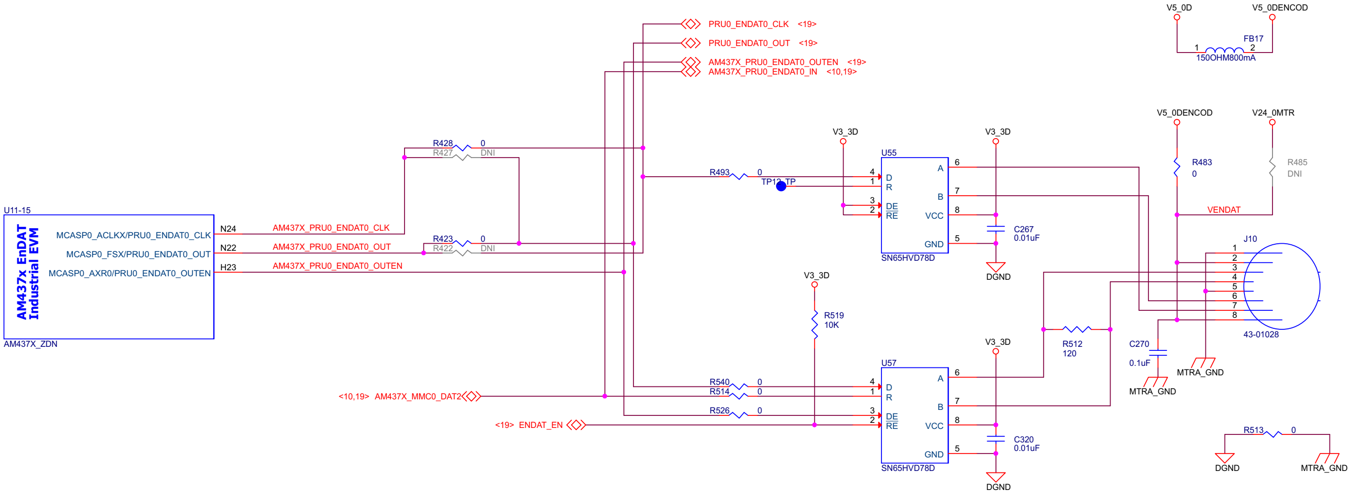


Figure 29. Hardware Modification for AM437x IDK

4 Getting Started Firmware

To use the firmware, do as follows:

1. Assemble the PRU code using the free pasm tool at https://github.com/beagleboard/am335x_pru_package.

NOTE: The pasm tool is built using Microsoft Visual Studio® and the pasm.exe file generates in the am335x_pru_package-master/pru_sw/utils directory.

---

2. Copy the pasm.exe file into BiSS firmware folder.
3. Enter pasm.exe --V3 --L --b pru0_BiSS.p on the Windows® terminal to generate the PRU binary file for BiSS firmware.

NOTE: The output of the assembler is a binary file that is used to download the code through CCS.

---

4. Install the latest version of CCS with the support for ARM platform from http://processors.wiki.ti.com/index.php/Download_CCS.
5. Connect the BiSS encoder to the AM437x IDK.
6. Power on the board with 24-V supply.
7. Connect the TI XDS100v2 USB Emulator_0 to the board.
8. Open CCS.
9. Navigate to File→ New→ Target Configuration File (see Figure 30).

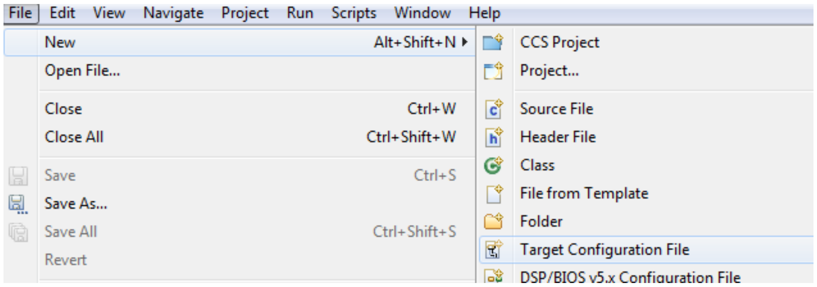
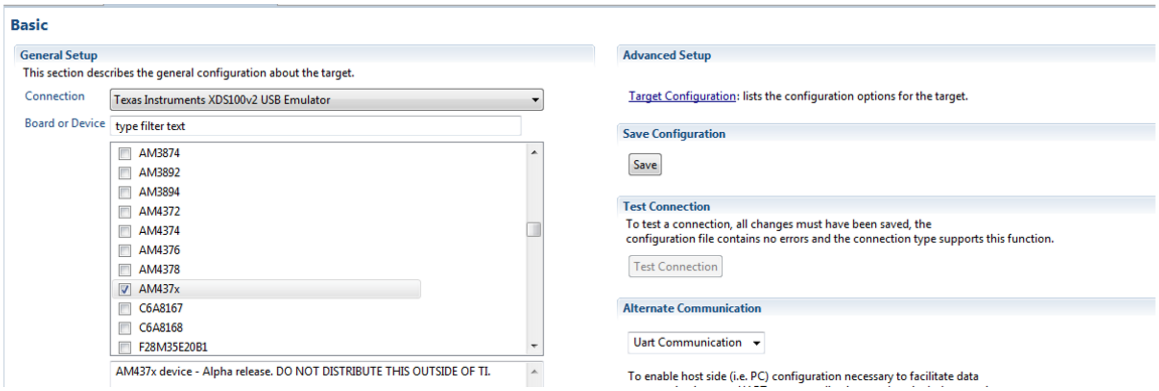


Figure 30. Creating the Target Configuration

10. Select the TI XDS100v2 USB Emulator_0 (see Figure 31).
11. Select the board or device as AM437x.
12. Save the device selection.
13. Click *Target Configuration* in the Advanced Setup tab.



Basic

General Setup

This section describes the general configuration about the target.

Connection: Texas Instruments XDS100v2 USB Emulator

Board or Device: type filter text

☐ AM3874
☐ AM3892
☐ AM3894
☐ AM4372
☐ AM4374
☐ AM4376
☐ AM4378
☑ AM437x
☐ C6A8167
☐ C6A8168
☐ F28M35E20B1

AM437x device - Alpha release. DO NOT DISTRIBUTE THIS OUTSIDE OF TI.

Advanced Setup

Target Configuration: lists the configuration options for the target.

Save Configuration

Test Connection

To test a connection, all changes must have been saved, the configuration file contains no errors and the connection type supports this function.

Alternate Communication

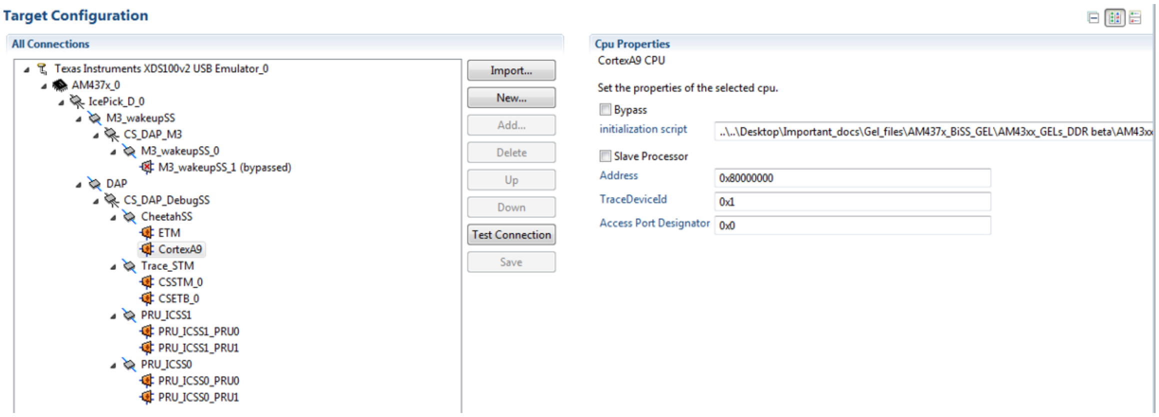
Uart Communication

To enable host side (i.e. PC) configuration necessary to facilitate data

Figure 31. Selecting the Device

14. Click *M3_WakeupSS_1* from the window on the left side.

15. Click the *Bypass* box.
16. Navigate to where you have saved the gel script.
17. Select the initialization script.
18. Click Cortex-A9 (see Figure 32).



```
Texas Instruments XDS100v2 USB Emulator_0
  AM437x_0
    IcePick_D_0
      M3_wakeupSS
        CS_DAP_M3
          M3_wakeupSS_0
            M3_wakeupSS_1 (bypassed)
      DAP
        CS_DAP_DebugSS
          CheetahSS
            ETM
            CortexA9
          Trace_STM
            CSSTM_0
            CSETB_0
          PRU_ICSS1
            PRU_ICSS1_PRU0
            PRU_ICSS1_PRU1
          PRU_ICSS0
            PRU_ICSS0_PRU0
            PRU_ICSS0_PRU1
```

CortexA9 CPU

Set the properties of the selected cpu.

☐ Bypass

initialization script ..\..\Desktop\Important_docs\Gel_files\AM437x_BiSS_GEL\AM43xx_GELs_DDR beta\AM43xx

☐ Slave Processor

Address 0x80000000

TraceDeviceId 0x1

Access Port Designator 0x0

Figure 32. Loading the Gel Script

19. Click the TI XDS100v2 USB Emulator_0.
20. Select JTAG TCLK Frequency (MHz) as *Adaptive with user specified value*.
21. Enter 15 MHz in the field *Enter a value from 1.0 MHz to 30 MHz*.
22. Save the target configuration file.

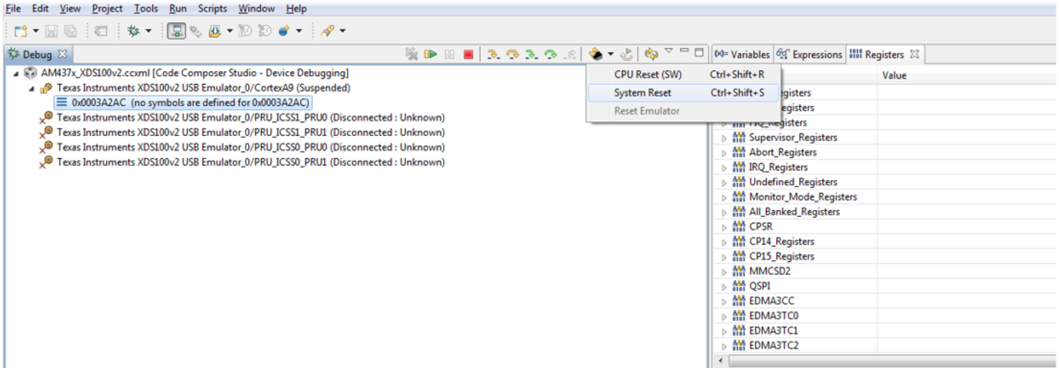
To connect the target, do as follows:

1. Right-click the target configuration file.
2. Select *Launch Selected Configuration*.

NOTE: You can view the Cortex-A9 core and four PRU cores in the target window

---

3. Right-click the Cortex-A9 core.
4. Click *Connect Target*.
5. Click *System Reset* (see Figure 33).
6. Click *Pause*.
7. Click *CPU Reset (SW)*.



```
AM437x_XDS100v2.ccxml [Code Composer Studio - Device Debugging]
  Texas Instruments XDS100v2 USB Emulator_0/CortexA9 (Suspended)
    0x0003A2AC  (no symbols are defined for 0x0003A2AC)
  Texas Instruments XDS100v2 USB Emulator_0/PRU_ICSS1_PRU0 (Disconnected : Unknown)
  Texas Instruments XDS100v2 USB Emulator_0/PRU_ICSS1_PRU1 (Disconnected : Unknown)
  Texas Instruments XDS100v2 USB Emulator_0/PRU_ICSS0_PRU0 (Disconnected : Unknown)
  Texas Instruments XDS100v2 USB Emulator_0/PRU_ICSS0_PRU1 (Disconnected : Unknown)
```

Figure 33. Connecting to the Cortex-A9 Processor

To run the gel scripts, do as follows:

1. Click *AM43xx System Initialization* (see Figure 34).

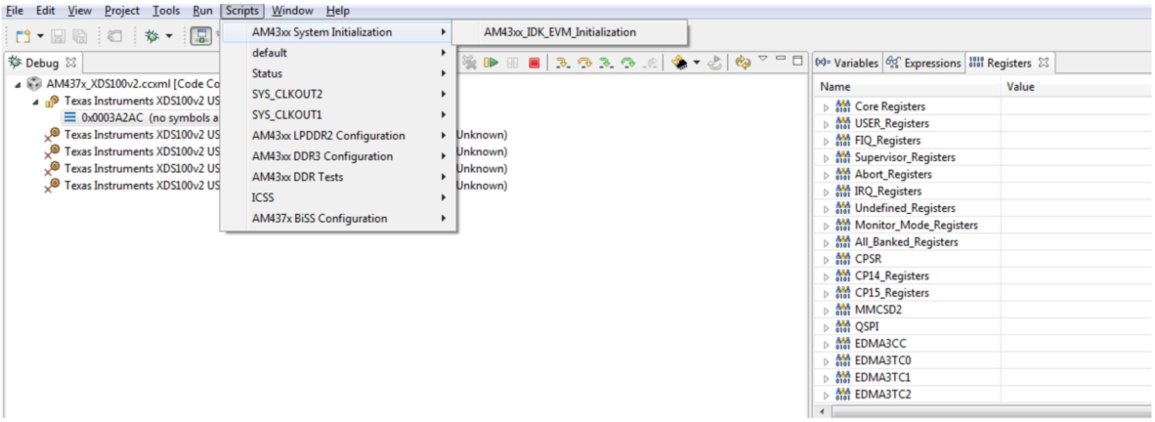
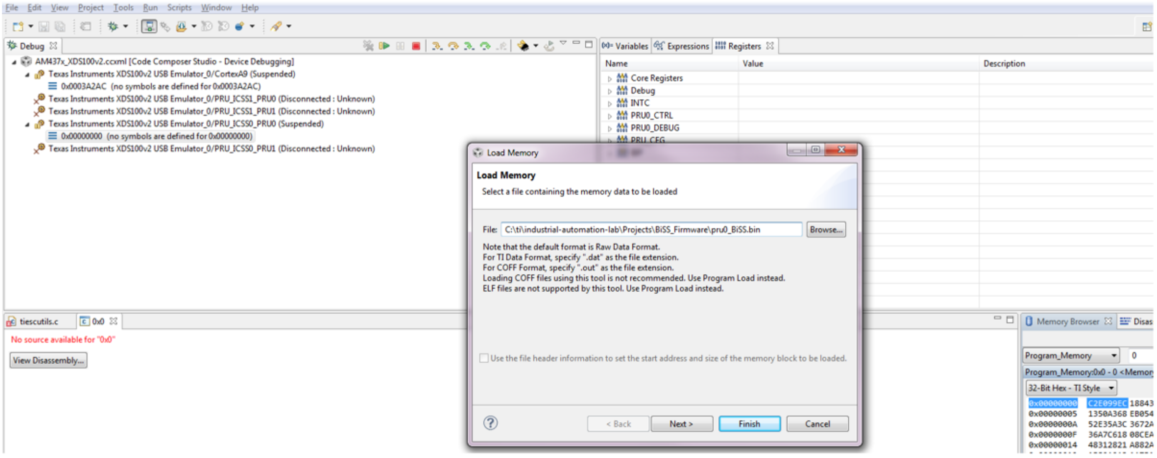


Figure 34. Executing Gel Scripts

2. Click *ICSS*.

To load the PRU firmware to the PRU-ICSS_0, do as follows:

1. Right-click *PRU_ICSS0_PRU0*.
2. Select *Connect Target*.
3. Right-click *Program_Memory*.
4. Load the BiSS-C interface master firmware binary file (see Figure 35).



Load Memory

Select a file containing the memory data to be loaded

File: C:\ti\industrial-automation-lab\Projects\BiSS_Firmware\pru0_BiSS.bin

Note that the default format is Raw Data Format.
For TI Data Format, specify ".dat" as the file extension.
For COFF Format, specify ".out" as the file extension.
Loading COFF files using this tool is not recommended. Use Program Load instead.
ELF files are not supported by this tool. Use Program Load instead.

☐ Use the file header information to set the start address and size of the memory block to be loaded.

Figure 35. Loading Code into PRU Program Memory

5. Click *Program_Memory*.
6. Select *PRU_Device _Memory*.
7. Click *Finish*.

NOTE: Clicking *Finish* shows the shared memory view from the PRU

---

8. See Table 1 to find the length of the BiSS encoder data, the length of CRC_ERR_WAR, and frequency.
9. Click *Start* to start the BiSS-C interface master firmware.
10. See Table 1 to find the results and status.

5 Test Setup

The test setup included an AM437x IDK, BiSS encoders, and cable. For debugging, a PC with the JTAG was used. An oscilloscope was used for measurements and screen shots. The encoder in the functional test and performance test was produced by Wachendorff and has the following order number: WDGF58M 10 12 00 11 BI A B 2 1 T3. This 12-bit single-turn digital encoder has a BiSS interface. The supply voltage can range from 3.5 V to 30 V. The encoder includes 2 meters of cable. For a cable length greater than 2 m, use Baumer IVO PVC cable (5 × 2 LiY 0.14 mm^2). Figure 36 shows the test setup for the BiSS-C interface master firmware.

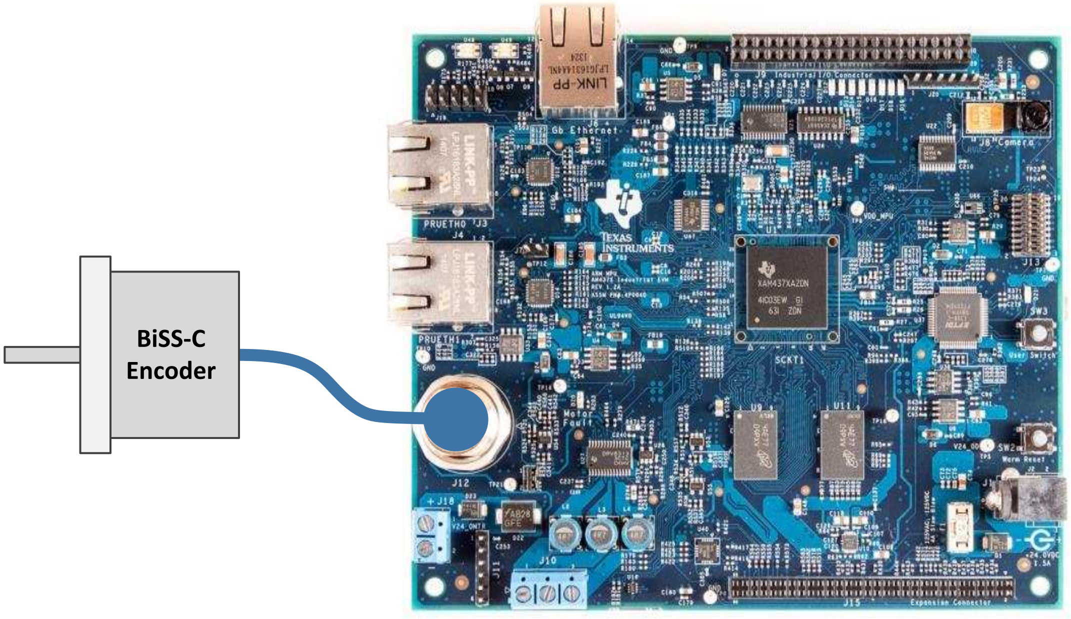


Figure 36. Test Setup for BiSS-C Interface Master

NOTE: Not optimized for maximum reach.

6 Test Data

Functional tests are performed with various cable lengths and supported frequencies. The design does not support maximum cable length but supports all functions including line delay compensation, which is required to support longer cables. To support maximum cable length, the design of the RS-485 transceiver must be optimized. A higher-quality cable is required for maximum cable length. The following sections contain screen shots from oscilloscope of the tests.

6.1 Screen Shots

Figure 37 shows a screen shot of transmission at 1 MHz in continuous mode.

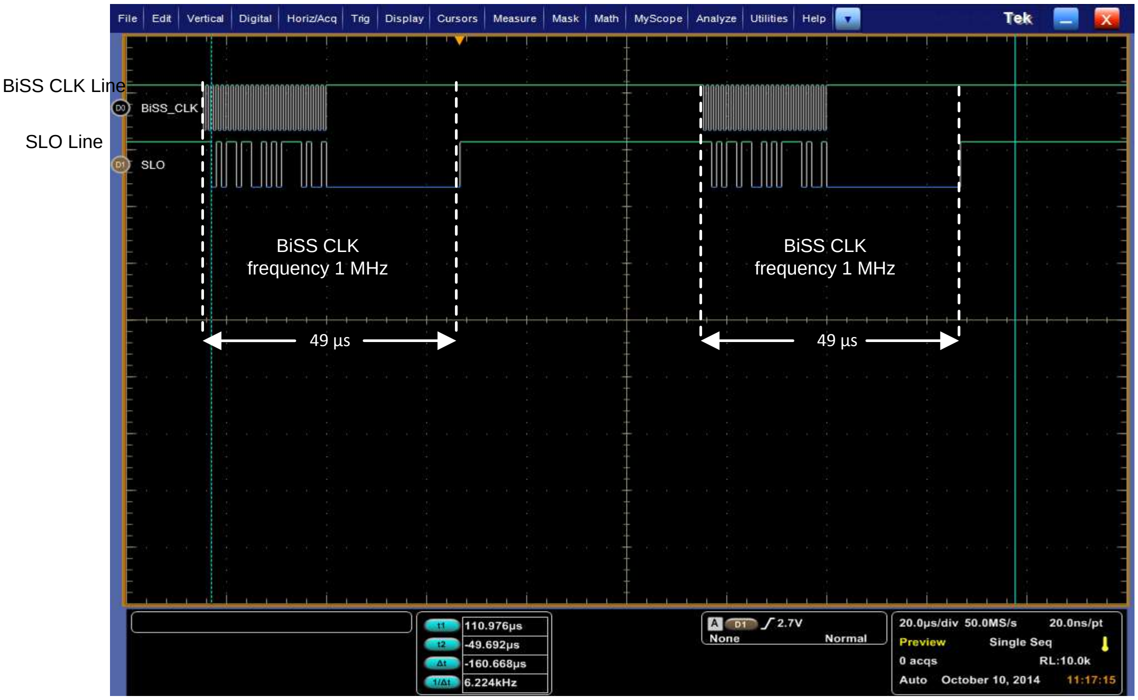
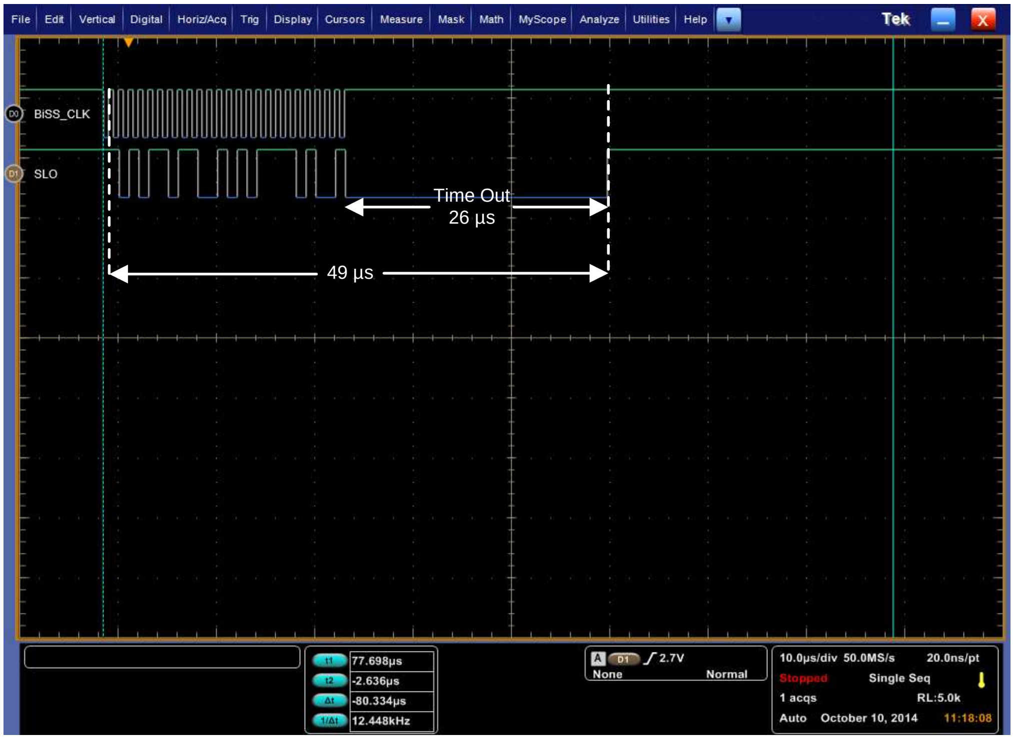


Figure 37. BiSS Continuous Mode

Figure 38 shows a screen shot of transmission at 1-MHz BiSS clock frequency. The minimum time required for the transmission is 49 µs.



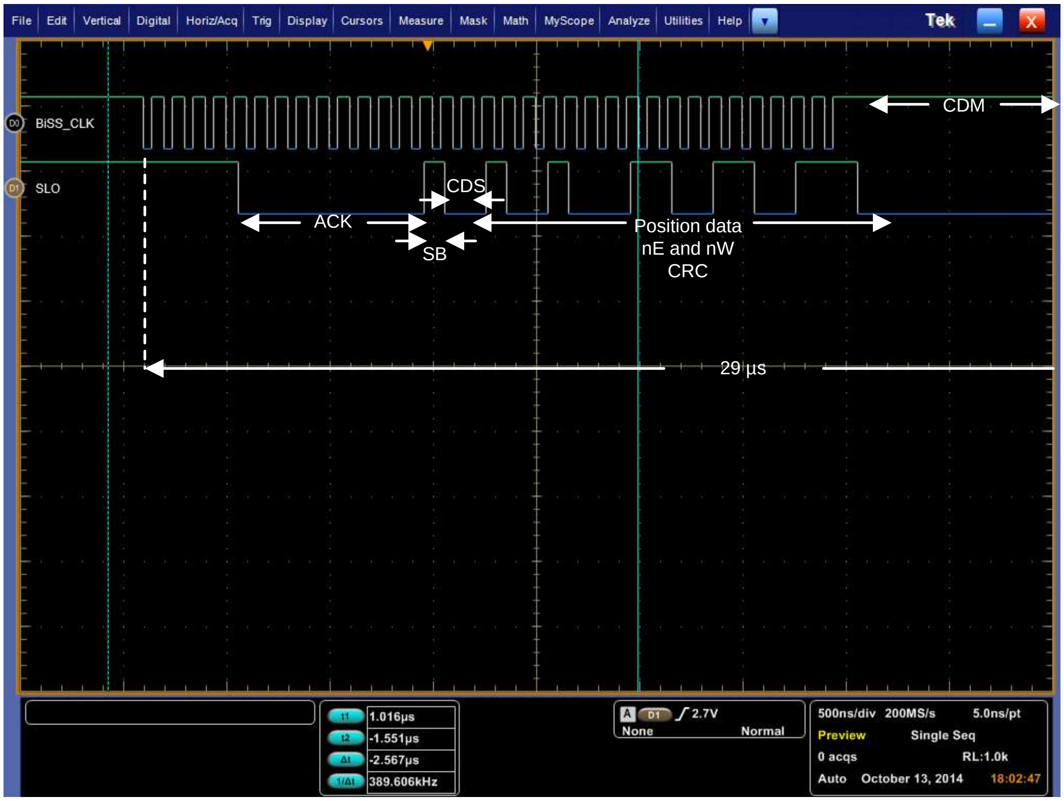
t1 77.698µs
t2 -2.636µs
Δt -80.334µs
1/Δt 12.448kHz

A D1 2.7V
None Normal

10.0µs/div 50.0MS/s 20.0ns/pt
Stopped Single Seq
1 acqs RL:5.0k
Auto October 10, 2014 11:18:08

Figure 38. Transmission at 1 MHz

Figure 39 shows a screen shot of transmission at maximum BiSS clock frequency (10 MHz). The minimum time required for the transmission in the maximum frequency is 29 µs.



t1 1.016µs
t2 -1.551µs
Δt -2.567µs
1/Δt 389.606kHz

A D1 2.7V
None Normal

500ns/div 200MS/s 5.0ns/pt
Preview Single Seq
0 acqs RL:1.0k
Auto October 13, 2014 18:02:47

Figure 39. Transmission at 10 MHz

6.2 Position Data Transmission Tests

Position data transmission tests were performed using various cable lengths with supported frequencies. For testing, BiSS cycles were repeated 1000 times and the number of CRC errors and other parameters, like line delay, were recorded. Table 10 lists the test cases and results.

Table 10. Position Data Transmission Test

No.	Frequency (MHz)	Cable Length (mk)	No. of BiSS Cycles	CRC Errors	Line Delay (No. of Clock Cycles)	Remarks
1	1	2	1000	0	1	BiSS cycle time = 49 µs
2	1	15 + 2	1000	0	1	BiSS cycle time = 51.83 µs
3	1	25 + 2	1000	0	1	BiSS cycle time = 51.73 µs
4	1	40 + 2	1000	0	1	BiSS cycle time = 52.90 µs
5	1	65 + 2	1000	29	2	BiSS cycle time = 54.10 µs
6	2	2	1000	0	1	BiSS cycle time = 39.73 µs
7	2	15 + 2	1000	0	1	BiSS cycle time = 39.33 µs
8	2	25 + 2	1000	0	2	BiSS cycle time = 39.53 µs
9	2	40 + 2	1000	0		BiSS cycle time = 39.80 µs
10	2	65 + 2	1000	0	3	BiSS cycle time = 42.10 µs
11	5	2	1000	0	1	BiSS cycle time = 31.18 µs
12	5	15 + 2	1000	0	3	BiSS cycle time = 32.32 µs
13	5	25 + 2	1000	6	3	BiSS cycle time = 32.82 µs
14	10	2	1000	1	4	BiSS cycle time = 29.87 µs

6.3 Control Communication Tests

Control communication tests were performed with various frequencies. Table 11 lists the results of the control communication tests. The following functional tests represent register definitions of specific Wachendorff encoders. All tests are passed with different communication frequencies.

Table 11. Control Communication Test

No.	Control Frame	Read/Write	Frequency (MHz)	Control Data Result (in Hex)	Remarks
1	Read at 0x44	Read	1	0	32-bit serial number
2	Read at 0x44	Read	2	0	32-bit serial number
3	Read at 0x44	Read	5	0	32-bit serial number
4	Read at 0x44	Read	10	0	32-bit serial number
5	Read at 0x45	Read	1	D9	32-bit serial number
6	Read at 0x45	Read	2	D9	32-bit serial number
7	Read at 0x45	Read	5	D9	32-bit serial number
8	Read at 0x45	Read	10	D9	32-bit serial number
9	Read at 0x46	Read	1	1	32-bit serial number
10	Read at 0x46	Read	2	1	32-bit serial number
11	Read at 0x46	Read	5	1	32-bit serial number
12	Read at 0x46	Read	10	1	32-bit serial number
13	Read at 0x47	Read	1	32	32-bit serial number
14	Read at 0x47	Read	2	32	32-bit serial number
15	Read at 0x47	Read	5	32	32-bit serial number
16	Read at 0x47	Read	10	32	32-bit serial number
17	Read at 0x60	Read	1	B	8-bit status
18	Read at 0x60	Read	2	B	8-bit status
19	Read at 0x60	Read	5	B	8-bit status
20	Read at 0x60	Read	10	B	8-bit status
21	Read at 0x67	Read	1	5F	Internal temperature
22	Read at 0x67	Read	2	5F	Internal temperature
23	Read at 0x67	Read	5	5F	Internal temperature
24	Read at 0x67	Read	10	5F	Internal temperature
25	Read at 0x68	Read	1	0	Error and warning messages
26	Read at 0x68	Read	2	0	Error and warning messages
27	Read at 0x68	Read	5	0	Error and warning messages
28	Read at 0x68	Read	10	0	Error and warning messages
29	Read at 0x78	Read	1	46	48-bit device ID
30	Read at 0x78	Read	2	46	48-bit device ID
31	Read at 0x78	Read	5	46	48-bit device ID
32	Read at 0x78	Read	10	46	48-bit device ID
33	Read at 0x79	Read	1	3A	48-bit device ID
34	Read at 0x79	Read	2	3A	48-bit device ID
35	Read at 0x79	Read	5	3A	48-bit device ID
36	Read at 0x79	Read	10	3A	48-bit device ID
37	Read at 0x7A	Read	1	C0	48-bit device ID
38	Read at 0x7A	Read	2	C0	48-bit device ID
39	Read at 0x7A	Read	5	C0	48-bit device ID

Table 11. Control Communication Test (continued)

No.	Control Frame	Read/Write	Frequency (MHz)	Control Data Result (in Hex)	Remarks
40	Read at 0x7A	Read	10	C0	48-bit device ID
41	Read at 0x7B	Read	1	0C	48-bit device ID
42	Read at 0x7B	Read	2	0C	48-bit device ID
43	Read at 0x7B	Read	5	0C	48-bit device ID
44	Read at 0x7B	Read	10	0C	48-bit device ID
45	Read at 0x7C	Read	1	0	48-bit device ID
46	Read at 0x7C	Read	2	0	48-bit device ID
47	Read at 0x7C	Read	5	0	48-bit device ID
48	Read at 0x7C	Read	10	0	48-bit device ID
49	Read at 0x7D	Read	1	1	48-bit device ID
50	Read at 0x7D	Read	2	1	48-bit device ID
51	Read at 0x7D	Read	5	1	48-bit device ID
52	Read at 0x7D	Read	10	1	48-bit device ID
53	Read at 0x7E	Read	1	57	Manufacturer ID
54	Read at 0x7E	Read	2	57	Manufacturer ID
55	Read at 0x7E	Read	5	57	Manufacturer ID
56	Read at 0x7E	Read	10	57	Manufacturer ID
57	Read at 0x7F	Read	1	41	Manufacturer ID
58	Read at 0x7F	Read	2	41	Manufacturer ID
59	Read at 0x7F	Read	5	41	Manufacturer ID
60	Read at 0x7F	Read	10	41	Manufacturer ID

7 Unsupported Features

Limitations in the current design are described in the following sections.

7.1 CRC Supports Data Lengths Only Less Than 23 Bits

CRC is calculated only for the BiSS-C encoders with an accuracy of less than 23 bits. If the accuracy of the encoder is greater than this limit, the value of CRC is invalid.

7.2 SMA is Not Available for 10 MHz

SMA is not available for 10 MHz but is available for all other frequencies supported by the BiSS-C interface master firmware. In the 10-MHz block, SMA cannot be used due to the timing limitations. This issue can be resolved using a look-up table with 256 entries in the shared memory. See the optimized section for 10 MHz for the details for implementing this method.

7.3 Sequential Register Access in Control Communication

In the control communication, the sequential register access is not implemented. In the list of hex equivalents for the control communication, the SB for the sequential control communication is not given (that is, each register access stops with a stop bit).

8 Software Files

To download the software files for this reference design, see http://www.ti.com/tool/TIDEP-0022.

For the CCS gel script (GPIO, example configuration of the encoder) and the PRU-ICSS firmware, see http://www.ti.com/lit/zip/tidc920.

9 References

1. *AM4379 Sitara™ Processor product page* (http://www.ti.com/product/am4379)
2. *iCHaus* home page (http://www.ichaus.de/)
3. *BiSS Interface* home page (http://www.ti.com/lit/pdf/http://www.biss-interface.com)
4. *Moving Average* wiki page (http://en.wikipedia.org/wiki/Moving_average)
5. *Cycle redundancy check* wiki page (http://en.wikipedia.org/wiki/Cyclic_redundancy_check)
6. *Programmable Real-time Unit (PRU) Software Support Package* tool page (http://www.ti.com/tool/pru-swpkg)
7. *CCS Download* page (http://processors.wiki.ti.com/index.php/Download_CCS)
8. *Interface to a 5-V BiSS Position Encoder Reference Design* (http://www.ti.com/tool/TIDA-00175)

Revision History

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

Changes from Original (March 2015) to A Revision	Page
• Updated Figure 11.	8
• Added "PRU accesses shared memory through the PRU constant table entry C28. ICSS0 has no PRU shared memory at location 0x0001.0000, therefore C28 points to 0x0000.0000 (PRU0 local memory)."	8
• Updated Memory Location column and Content column in Table 1. Structure in Shared Memory.	9
• Updated steps following "Perform the following modifications:"	33
• Updated Step 7. Changed GO to FINISH.	36

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Texas Instruments Incorporated ("TI") reference designs are solely intended to assist designers (“Buyers”) who are developing systems that incorporate TI semiconductor products (also referred to herein as “components”). Buyer understands and agrees that Buyer remains responsible for using its independent analysis, evaluation and judgment in designing Buyer’s systems and products.

TI reference designs have been created using standard laboratory conditions and engineering practices. **TI has not conducted any testing other than that specifically described in the published documentation for a particular reference design.** TI may make corrections, enhancements, improvements and other changes to its reference designs.

Buyers are authorized to use TI reference designs with the TI component(s) identified in each particular reference design and to modify the reference design in the development of their end products. HOWEVER, NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY THIRD PARTY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT, IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI REFERENCE DESIGNS ARE PROVIDED "AS IS". TI MAKES NO WARRANTIES OR REPRESENTATIONS WITH REGARD TO THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING ACCURACY OR COMPLETENESS. TI DISCLAIMS ANY WARRANTY OF TITLE AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT, QUIET POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO TI REFERENCE DESIGNS OR USE THEREOF. TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY BUYERS AGAINST ANY THIRD PARTY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON A COMBINATION OF COMPONENTS PROVIDED IN A TI REFERENCE DESIGN. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT OF TI REFERENCE DESIGNS OR BUYER’S USE OF TI REFERENCE DESIGNS.

TI reserves the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques for TI components are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

Reproduction of significant portions of TI information in TI data books, data sheets or reference designs is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards that anticipate dangerous failures, monitor failures and their consequences, lessen the likelihood of dangerous failures and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in Buyer’s safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed an agreement specifically governing such use.

Only those TI components that TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components that have ***not*** been so designated is solely at Buyer's risk, and Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265