Application Report
Using the Fast Serial Interface (FSI) With Multiple Devices in an Application

TEXAS INSTRUMENTS

Aki Li, Kevin Allen, and Aditya Dholakia

ABSTRACT

In industrial applications, it is often necessary for multiple devices to communicate with each other in a fast, low latency, and synchronized manner. One example is in a decentralized / distributed control system architecture. A new communication peripheral created for C2000™ Real-Time Control Microcontrollers (MCU), the Fast Serial Interface (FSI) can expand its reliable high-speed communication features to multiple devices in a system. This application report demonstrates how to implement a daisy-chain or star network topology using FSI. Test results are provided to validate the high-speed communication capability of FSI with different configuration methods. You can quickly verify and design FSI in different applications using the provided source code, which can be downloaded from C2000WARE.

The target processors for the corresponding software include the TMS320F28002x, TMS320F28004x, and TMS320F2838x. The implementation methods and software can be applied and ported to future C2000 processors that include FSI. Example code discussed in this document can be found in the latest C2000WARE release, located within the following local directory after installation:

C:\ti\c2000\C2000Ware_<version_number>\driverlib\f28xxxx\examples\fsi

The available example projects are:

- fsi_ex_daisy_handshake_lead
- fsi_ex_daisy_handshake_node
- fsi_ex_star_broadcast

Table of Contents

List of Tables

Trademarks

C2000™ and Code Composer Studio™ are trademarks of Texas Instruments.

All trademarks are the property of their respective owners.

2   *Using the Fast Serial Interface (FSI) With Multiple Devices in an Application*   SPRACM3D – OCTOBER 2020 – REVISED AUGUST 2021

Submit Document Feedback

1 Introduction to the FSI Module

The FSI module is a serial communication peripheral capable of reliable and robust high-speed communications, up to 200 Mbps. Utilizing very few unidirectional signals, FSI provides a low cost way of communicating across an isolation barrier when leveraging digital isolation devices. Thus, FSI enables new ways of distributing the powerful sensing, processing, and actuation capabilities of C2000 MCUs in industrial applications, where real-time control with critical communication speed is required.

Generally, FSI can be implemented in two kinds of system conditions:

- Wired communications between MCUs that exist on the same voltage and ground planes.
- Wired communications across an isolation barrier, leveraging digital isolators (like ISO77xx), commonly used for MCUs placed on the hot-side needing to communicate with MCUs on the cold-side, or between boards with different voltage and ground planes.

There are a number of real-time systems that can benefit from the FSI peripheral. A multi-axis servo drive can be constructed with C2000 device nodes controlling each axis. Having FSI serve as the communication link, control loop information can be quickly transmitted and received between the devices to maintain precise motion control. For an example of this system see the *Distributed Multi-axis Servo Drive over Fast Serial Interface (FSI) reference design*.

Additionally, with increasing global power consumption, the need for higher efficiency power supplies, in conjunction with the availability of wide bandgap GaN and SiC products, is driving the use of more sophisticated power distribution architectures. Decentralized power control solutions using C2000 MCUs can be connected and made flexible with FSI to meet these requirements. For a discussion on such power related systems see the *Distributed Power Control Architecture with Multiple MCUs Over FSI* white paper.

The FSI peripheral offers a broad range of features, including programmable data length, hardware managed CRC, ECC support, and more. A PING watchdog and Frame watchdog can enable automatic line-break detection. The unique delay line control feature implemented within the FSI receive module can adjust for channel-to-channel skew introduced by trace-length mismatch, transceivers, or digital isolation ICs, allowing FSI to maintain high-speed and robust communication.

The FSI consists of the independent transmitter (FSITX) and receiver (FSIRX) cores, which are configured and operated independently. Because of this, the FSI protocol does not have a notion of master and slave, unlike some other synchronous communication protocols, and allows for simultaneous full speed communications in both directions. Figure 1-1 shows the CPU interface of each FSI module. Each module owns up to three signal lines: one clock and two data signals, where the second data lines, FSITXyD1 and FSIRXyD1, are optional, and can be enabled for multi-lane transmission and double the speed for data bits. Thus, at least four signal lines are needed to create 2-way point-to-point communication. Considering the timing spec for FSITX (see the device-specific data sheets referenced in Section 8), the maximum data rate of 200 Mbps can be achieved with the maximum clock of 50 MHz, using two data lines, since the data is transmitted on both edges of the clock signal. For a full overview of FSI including all features and functions available, see the device-specific Technical Reference Manual (TRM).
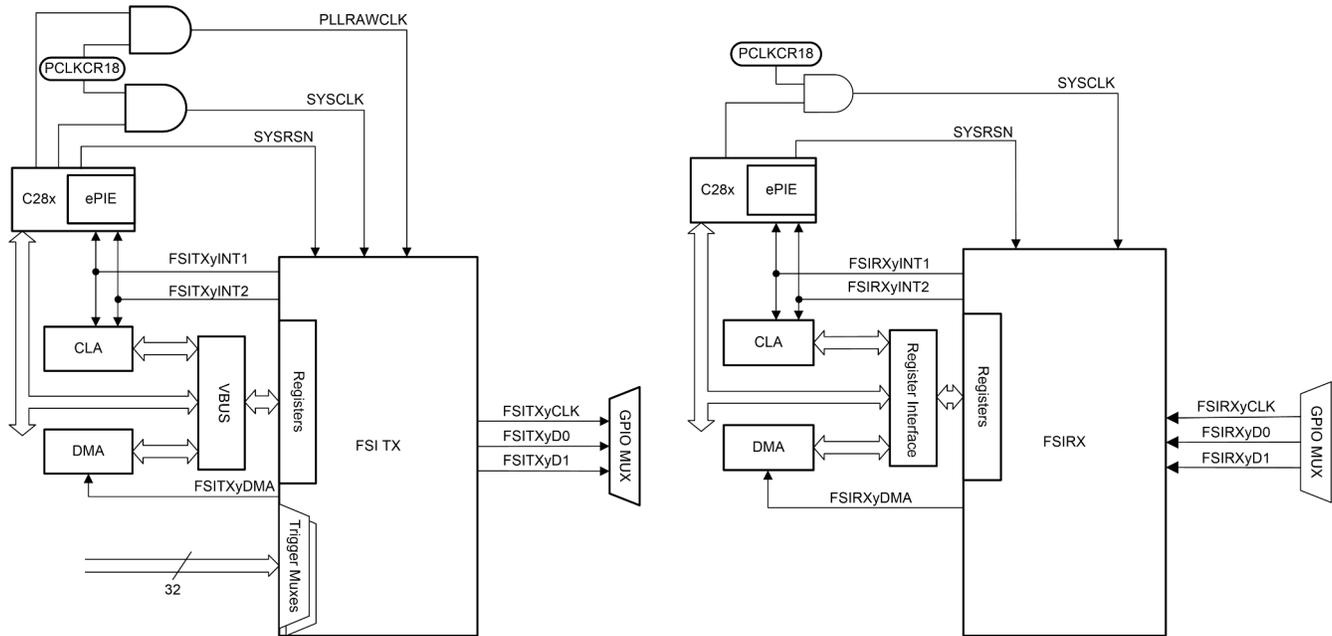
Figure 1-1. FSITX and FSIRX CPU Interface

2 FSI Applications

In terms of the trend in power electronic applications, the increasing demand for higher power levels makes multiple power modules in parallel much more popular. Examples of such applications include industrial drives, telecom rectifiers, server power supplies, on-board chargers, and so forth. Meanwhile, to achieve a complex system with high performance, multiple MCUs are commonly used and must operate in a synchronized fashion. Thus, critical data, including protection signals, sampling parameters, and even control loop data, needs to be transferred with the fastest speed and least amount of latency among multiple devices/modules. FSI will be more suitable to handle this when compared to the traditional Controller Area Network (CAN), Serial Peripheral Interface (SPI) or Universal Asynchronous Receiver/Transmitter (UART).

There are a number of communication network topologies for connecting multiple devices, each with their own benefits. A ring topology can be created by connecting multiple devices with FSI communication in a daisy-chain fashion. The advantages of a ring topology are that each device only needs one FSI transmitter and receiver and also the simplicity from a physical connection perspective. Figure 2-1 shows a daisy-chain connection system for N (N≥2) node devices, where each device (index i) connects with the FSITX of device i-1 and FSIRX of device i+1.
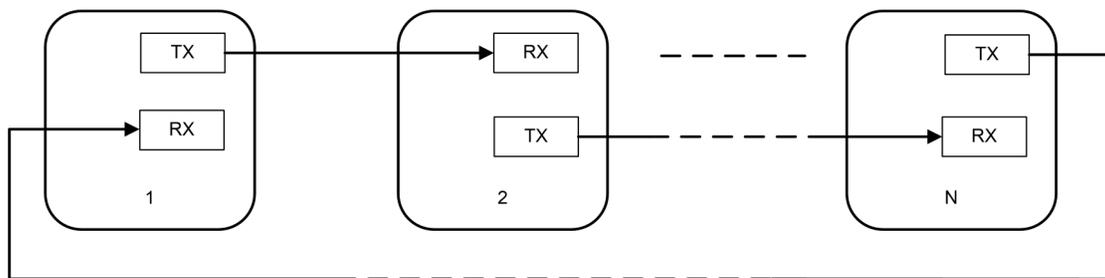
Figure 2-1. Daisy-Chain Connection Example

One disadvantage of the above daisy-chain topology is that if one device in the chain fails then the entire communication link is broken. Another downside is that devices must forward data along to the next device in the chain if the received data is intended for a subsequent device. This can add to the overall latency of when a data packet is transmitted and when the respective device in the chain receives the data.

One communication topology that solves the broken link issue and can reduce the device-to-device latency is a star topology, where several nodes connect directly to one central host device. Figure 2-2 shows a star topology system with N (N≥2) node devices.
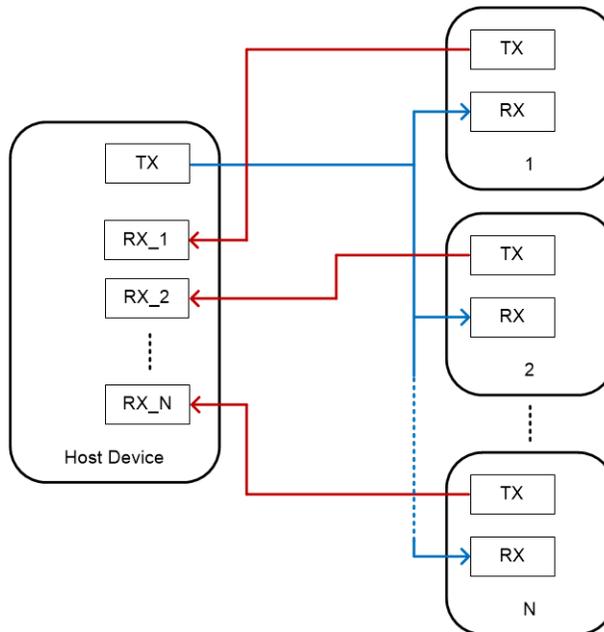

Figure 2-2. Star Topology Example

The host device's FSI transmitter is connected to the FSI receiver of each node device in order for the host to broadcast data packets to all nodes simultaneously. The node device transmitter's, on the other hand, are connected to independent receivers of the host device enabling them to send data directly back to the host at any time. This star implementation comes with a resource cost as the host needs N number of independent FSI receiver modules. The F2838x family of C2000 devices fit into the host socket with having two FSI transmitters and eight FSI receivers.

3 Handshake Mechanism

A handshake mechanism can be implemented in order to configure the devices and validate the link in an FSI communication topology. The different handshake sequences are discussed in the following sections.

3.1 Daisy-Chain Handshake Mechanism

Once the FSITX and FSIRX modules of each device have been configured, the handshake mechanism should be implemented to prepare each device in the chain before actual data transmission, since devices may power up in an arbitrary order in a real scenario.

In order to simplify the data flow, one device is assigned as the lead, working as the driver of the handshake sequence, and the other N-1 devices, within the daisy-chain loop, are assigned as nodes. Following the example in Figure 2-1, Device 1 will be the lead device. It should be noted that the other N-1 node devices will share the same handshake configuration.

The handshake process can be described as follows:

1. For all devices, configure the Frame Type of FSITX as Ping Frame, and enable the receiver interrupts for Ping Frame Received event on the FSI INT1 vector to detect the incoming transmission.
2. Begin the ping loop 0:
   a. The lead device sends the flush sequence to the second device followed by a ping frame with Tag0(0000); wait for some time. If the lead device receives a valid ping frame tag Tag0, continue to the second loop; otherwise iterate the ping loop 0 again.
   b. The node devices enter a wait loop for a receiver interrupt. If a valid ping frame tag of Tag0 is received from the previous device, continue to the loop 1; otherwise iterate the ping loop 0 again.

3. Begin the ping loop 1:
   a. The lead device sends a ping frame with Tag1(0001); wait for some time. If the lead device receives a valid ping frame Tag1 the handshake sequence is complete and the application can continue; otherwise iterate the ping loop 1 again.
   b. The node devices send the flush sequence followed by a ping frame Tag0 and wait for a receiver interrupt. If a valid ping frame Tag1 is received send a ping frame Tag1 to signal the completion of the handshake sequence; or else iterate the ping loop 1 again.
4. Handshake completed.
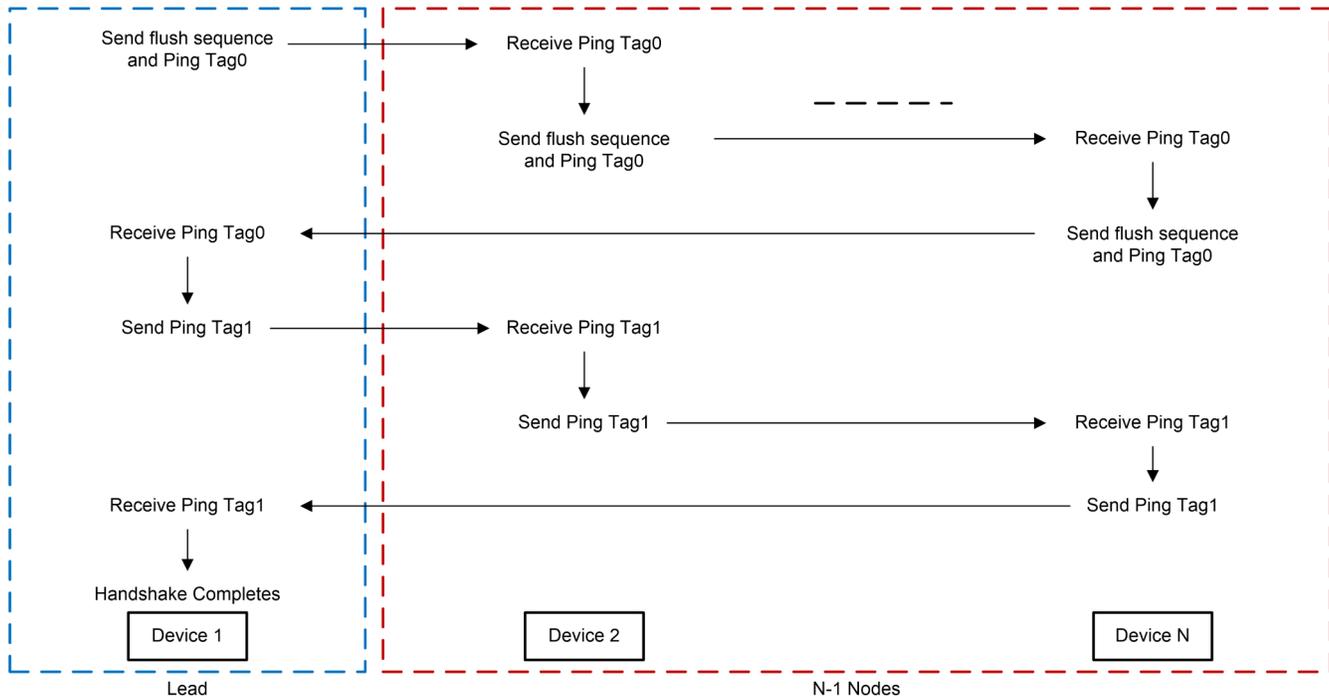
Figure 3-1. Daisy-Chain Handshake Sequence

The simplified data flow is shown in Figure 3-1. Two ping loops are necessary for the daisy-chain connection handshake mechanism. Ping loop 0 has the purpose of establishing the communication path along the chain of devices and ping loop 1 acts as the acknowledgment to the nodes that the communication path is good. In ping loop 0, the node devices wait to receive a Ping Tag0 from the previous device in the chain. Once a Ping Tag0 is successfully received, it will be forwarded on to the next device in the chain. The ping loop 0 will fail if a device in the chain has not powered up or is not ready for the reception. Once ping loop 0 has succeeded, in which ping tag0 has made its way back to the lead device, ping loop 1 is initiated to inform the node devices that the handshake sequence has completed and to begin expecting actual data.

The handshake function can be found in the tested projects, with handshake_lead() for the lead device and handshake_node() for the other N-1 devices in the daisy-chain loop.

3.2 Star Handshake Mechanism

A handshake sequence very similar to what is described in Section 3.1 can be applied for the star topology case. In this implementation the host, or lead, device sends broadcast pings to all of the node devices and they each respond along their independent TX paths. The lead device waits to receive ping frames from all nodes before moving on to the next step.

The sequence is shown in Figure 3-2.
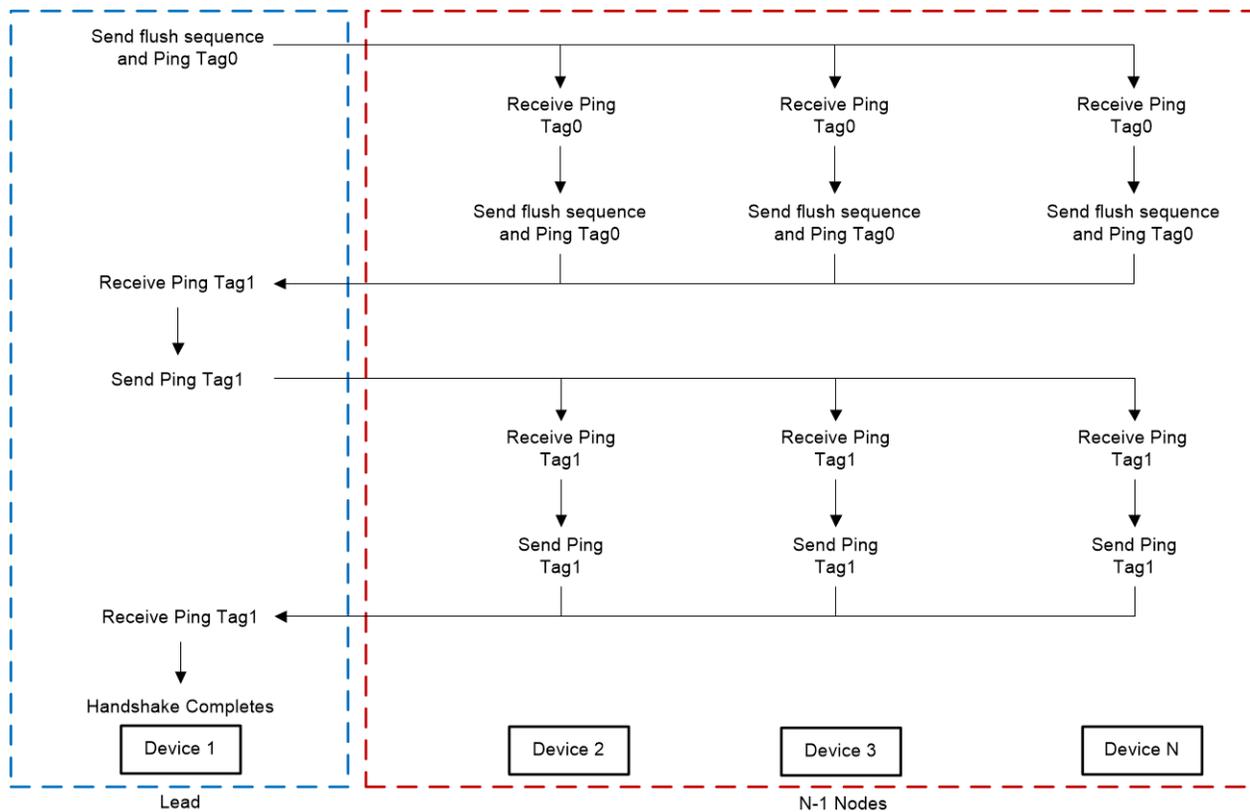
Figure 3-2. Star Handshake Sequence

4 Sending and Receiving FSI Data Frames

4.1 FSI Data Frame Configuration APIs

Several configurations are needed for data frames to be sent and received properly, including the frame type, frame tag, user data, word length, number of data lines, and writing to or reading from the data buffer. The configuration example code uses driverLib API functions for FSITX and FSIRX, defined in the fsi.h driverLib header file in C2000WARE, which are shown below. Note that the content of frame tag and user data is fully user-configurable, which can be used to differentiate which device the data received is sent from or which device it is meant for.

```
// TX setting part
   FSI_setTxFrameType(FSITXA_BASE, FSI_FRAME_TYPE_NWORD_DATA);
   FSI_setTxSoftwareFrameSize(FSITXA_BASE, nWords);
   FSI_setTxDataWidth(FSITXA_BASE, nLanes);
   FSI_setTxUserDefinedData(FSITXA_BASE, txUserData);
   FSI_setTxFrameTag(FSITXA_BASE, txDataFrameTag);
// RX setting part
   FSI_setRxSoftwareFrameSize(FSIRXA_BASE, nWords);
   FSI_setRxDataWidth(FSIRXA_BASE, nLanes);
```

4.2 Start Transmitting Data Frames

There are three methods to trigger the data transmission, including software triggered, externally triggered (EPWMx-SOCA/B), or using the DMA. For the software triggering method, writing 1 to the TX_FRAME_CTRL.START register bit, or using the driverLib function "FSI_startTxTransmit()", will start the transmission. If using an external trigger, like EPWMx-SOCA, once the external trigger signal occurs, the data will be sent.

Since the DMA trigger can be generated every time a data frame transmission or receiving is completed from the FSITX or FSIRX module, it provides a convenient method to transfer and store data, especially with a mass amount of data. Here a configuration example is given for the FSI communication using DMA.

TX_OPER_CTRL_LO.START_MODE must be set to 0x2, which means writing to frame tag/user data fields can trigger the transmission and then enables a DMA event on FSITX:

```
FSI_setTxStartMode(FSITXA_BASE, FSI_TX_START_FRAME_CTRL_OR_UDATA_TAG);
FSI_enableTxDMAEvent(FSITXA_BASE);
```

Two consecutive DMA channels are needed to fill the transmit buffer and frame tag/user data fields, respectively. Using the two channels in sequence allows for the transmission to start right after the frame tag and user data are set, as configured in the TX_OPER_CTRL_LO.START_MODE register bits. In the example code, DMA CH1 and DMA CH2 are used. Another important point is that the wrap control must be enabled for data of more than 16 words, since the FSI transmit buffer is a 16-word circular buffer.

```
DMA_configWrap(DMA_CH1_BASE, DMA_TRANSFER_SIZE_IN_BURSTS, 0, dest_WrapSize, 0);
```

Here, dest_WrapSize represents the number of bursts to be transferred before a wrap of the destination address, so dest_WrapSize should be 16/ nWords. This can be implemented such that the transmit buffer is continuously fed by the DMA, which is triggered by FSITX in return, with DMA Continuous Mode enabled.

The FSIRX is configured very similarly to the FSITX, except for that there is no order requirement for the DMA channels for RX buffer and tag and user data. In the example projects, DMA CH3 and DMA CH4 are used.

5 Daisy-Chain Topology Tests

In order to demonstrate the communication speed and different configurations for FSI, daisy-chain connections for two device and three device configurations have been tested and validated. The test hardware utilized is composed of multiple F280025C ControlCARD Evaluation Modules and TMDSFSIADAPEVMs.

Note

These same tests can be performed with LAUNCHXL-F280049Cs and TMDSFSIADAPEVMs as well with similar hardware setups.

The tested example projects can be found within the C2000WARE download. All test results were collected using optimization level 2, configured within Code Composer Studio™ (CCS). Changing the optimization level may yield different results. An overall description of the tested projects is shown in Table 5-1. For better understanding, a general software flow chart with different project settings is shown in Figure 5-1.

Table 5-1. Example Projects Description - Daisy-Chain

Project	Description	Settings
fsi_ex_daisy_handshake_lead	Project for the lead device in the daisy-chain loop.	① [#define FSI_DMA_ENABLE 0] represents FSI communication using CPU control.
fsi_ex_daisy_handshake_node	Project for the N-1 other devices (N>=2) in the daisy-chain loop.	② [#define FSI_DMA_ENABLE 1 && #define TX_DMA_TRIGGER_ENABLE 0] represents FSI communication using DMA control, and using software to trigger DMA for the transmitted data (manually). ③ [#define FSI_DMA_ENABLE 1 && #define TX_DMA_TRIGGER_ENABLE 1] represents FSI communication using DMA control, and enabling FSITX to trigger DMA for the transmitted data.
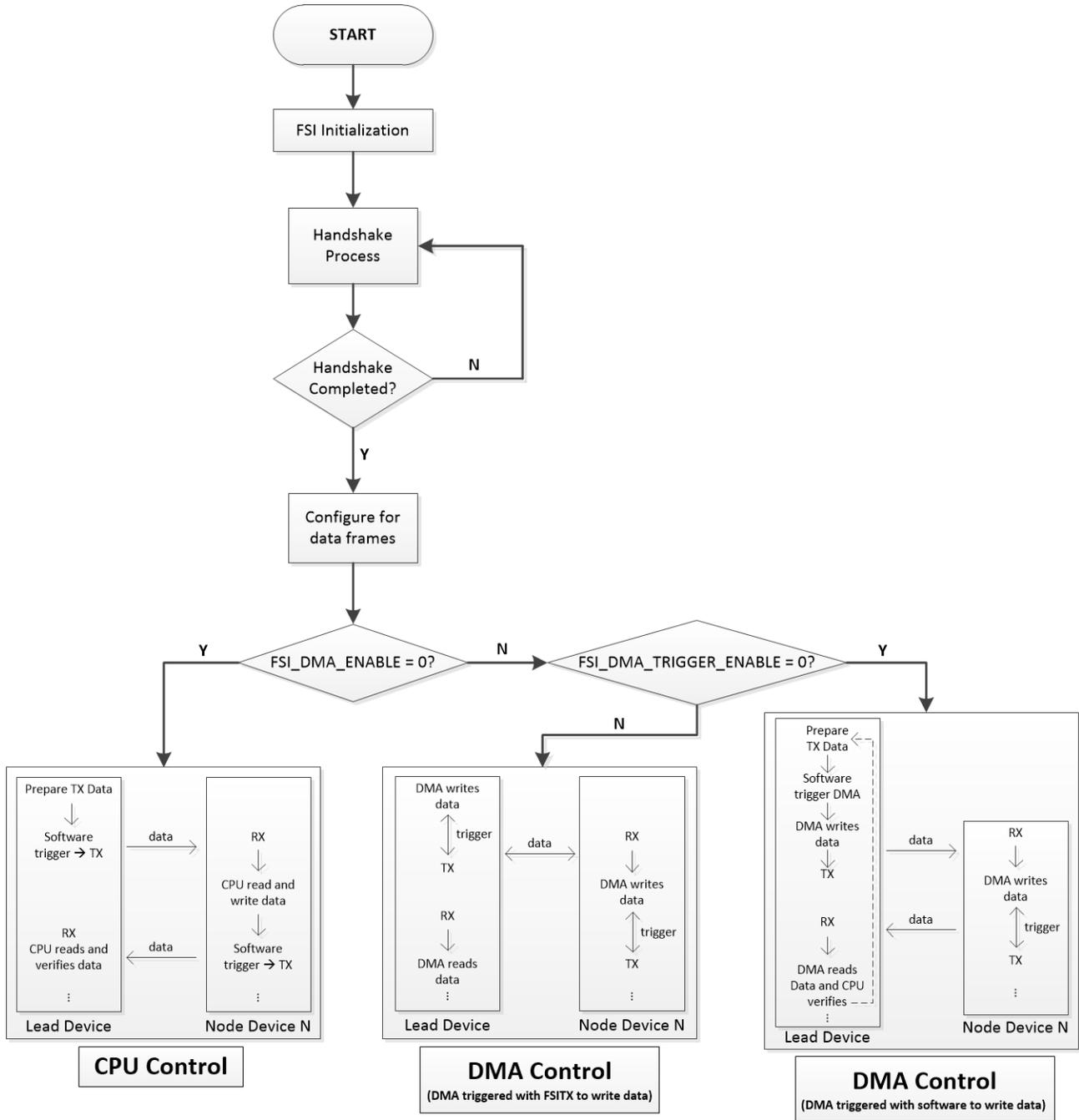
Figure 5-1. Software Flow Chart With Different Project Settings

5.1 Two Device FSI Communication

For a minimal daisy-chain connection test, a system of two F280025C ControlCARD Evaluation Modules and TMDSFSIADAPEVMs are used as shown in Figure 5-2. The comparison of the communication speeds for both CPU control and DMA control is provided in the following sub-sections.

Figure 5-2. Test Platform for Two Device Communication

5.1.1 CPU Control

• **Test condition:**

Device 1 sends data -> Device 2 receives data -> Device 2 CPU moves RX data to TX buffer and registers -> Device 2 triggers FSI TX with SW which forwards the received data back to Device 1 -> Device1 receives data back and the CPU verifies it matches the originally sent TX data.

• **Test case:**

Data length of 8 words, two data lines, TXCLK = 50 MHz, with Setting ① (Table 5-1) enabled.

In the test, GPIOs are toggled within software when specific events occur during the communication and measured using an oscilloscope to obtain the respective timing data. In Figure 5-3, the green signal represents the GPIO toggling of Device 1 (Lead device) and the magenta signal represents the GPIO toggling of Device 2 (Node device).

Figure 5-3. Data Transmission Test Using CPU Control

10   *Using the Fast Serial Interface (FSI) With Multiple Devices in an Application*   SPRACM3D – OCTOBER 2020 – REVISED AUGUST 2021

Submit Document Feedback

From the results shown in Figure 5-3, the time obtained for the data transmission is ~1.4 µs. In order to calculate the transmission speed, the total data length should be considered. Table 5-2 shows the general structure of a data frame, which can be divided into two parts: effective data bits and overhead bits.

- **Effective Data Bits:** Includes the 8-bit User Data, 16-bit Data Words, and 8-bit CRC fields
- **Overhead Bits:** Includes the Preamble, SOF, Frame Type, EOF, and Postamble fields

Therefore, the ideal transmission time for 8 words can be derived theoretically, as shown in Table 5-3.

It should be noted that since two data lines only work for effective data bits, one FSITXCLK cycle delivers 4 effective data bits, while one FSITXCLK cycle only delivers 2 overhead bits. Thus, with a total 48 FSITXCLK cycles for 8 data words, the transmission time can be calculated as shown in Equation 1.

$$\text{(FSITXCLK cycles)/(FSITXCLK frequency)} = 48/50\text{MHz} = 0.96 \text{ µs} \tag{1}$$

Therefore, the theoretical transmission speed is 175 Mbps (168/0.96 µs), while the speed from the test is 120 Mbps with 1.4 µs transmission time, due to the fact that the tested transmission time includes entering the ISR (to toggle an IO pin), delay introduced by isolators, transceivers, cables, and so forth. If changing to one data line, the theoretical transmission speed is 100 Mbps, while the test speed is 80 Mbps with a transmission time of 2.1 µs.

Another finding from Figure 5-3 is that moving data from the FSIRX buffers to the FSITX buffers in the node device takes some time, approximately 4.9 µs using the FSI driverLib functions. This will be a key factor that distinguishes DMA control as shown in the next section.

Note

The time to move data between the FSI buffers and registers can be optimized by writing to and reading from the FSI registers directly instead of using the provided driverLib functions.

Table 5-2. Data Frame Structure

IDLE	Preamble	SOF	Frame Type	User Data	Data Words	CRC	Frame Tag	EOF	Postamble	IDLE
	1111	1001	0011	8 bits	N words	8 bits	4 bits	0110	1111	

Table 5-3. Calculated Transmission Time for 8 Words

Effective Data Bits (bits)	Overhead Bits (bits)	Total Length (bits)	FSITXCLK Cycles for Effective Data Bits (cycles)	FSITXCLK Cycles for Overhead Bits (cycles)	Total Data Transmission Time (us)
144	24	168	36	12	0.96

5.1.2 DMA Control

- **Test condition**

  Device 1 sends data -> Device 2 receives data -> Device 2 DMA moves RX data to TX buffer and registers -> Device 2 triggers TX when TX_FRAME_TAG_UDATA FSI register is written to which forwards the received data back to Device 1 -> Device 1 receives data back -> Device 1 DMA moves RX data to memory -> CPU verifies data in memory matches the originally sent TX data.
- **Test case**

Data length of 8 words (8 words per burst, 1 burst per transfer), two data lines, TXCLK = 50 MHz, with Setting ② (Table 5-1) enabled.

In this test, DMA interrupts from CH2 and CH4 are enabled to trigger at the end of a transfer in the lead device, which means that interrupts occur every time data has been copied from memory to the FSITX buffer (CH2) or data has been transferred out of the FSRX buffer to a location in memory (CH4). In the node device, the DMA channels are configured to transfer received data from the RX buffers and registers to the TX buffers and registers anytime an FSI data frame is received, ultimately forwarding the data back to the lead device. Therefore the node device only has one DMA interrupt enabled while the lead device has two. With GPIOs toggling in the DMA ISRs, Figure 5-4 shows the test results of FSI communication using DMA control.

Figure 5-4. FSI Communication Using DMA Control

It should be noted that the time of 1.86 µs shown includes the time for the lead device to transmit the data frame, the node device to move RX data to the TX buffer / registers, entering of the ISR, and toggling the GPIO. According to the DMA pipeline timing requirements (See device specific TRM referenced in Section 8 for more info), the time for moving data of 9 words (8 words data + 1 word of user data and frame tag) using 2 channels can be calculated as shown in Equation 2.

$$(9 \times 3 \; cycles/word + 2 \; cycles) \div (100 MHz) = 0.29 \mu s \tag{2}$$

Thus, considering other delay times, the actual transmission time is almost aligned with the former test result (1.4 µs) using CPU control. Also, it should be highlighted that utilizing the DMA in this case drastically saves the time for transferring received data, especially in an application with mass data transmissions..

Further test results are given in Table 5-4, for the comparison of using CPU control and DMA control of FSI. With overhead bits being fixed in the FSI data frame structure, it is beneficial to use a longer data length to maximize the effective data throughput.

Table 5-4. Comparison of Using CPU Control and DMA Control in FSI for Two Devices

	FSITXCLK (MHz)	# of Data Lines	Data Length (16-bit words)	Transmission Time (µs) [1]	Buffer Data Move Time (µs) [1]	Theoretical Transmission Speed (Mbps)[2]	Test Transmission Speed (Mbps)
CPU Control	50	2	8	1.4	4.9	175	120
	50	2	16	2.1	8.3	185	141
	50	1	8	2.1	4.9	100	80
	10	1	8	8.9	4.9	20	18.9
DMA Control	50	2	8	1.9		/	/
	50	2	16	3.0		/	/
	50	1	8	2.6		/	/
	10	1	8	9.3		/	/

(1)   Measured times are rounded to the nearest 0.1 us.
(2)   Accounts for FSI frame overhead bits being transmitted on both lines in the two data lines cases.

There may be cases where FSI communication may need some additional robustness and noise immunity and for that reason a lower clock frequency has also been tested. The FSI protocol is designed to communicate only when there is data exchange. This helps to reduce power and over all EMI in the system. Additionally, lower FSI clock frequencies and half-duplex communication could improve overall system level EMI performance while continuing to provide higher throughput than generic serial ports at the same operating frequencies. Generally, it is best to use a twisted pair or shielded wire per line for board-to-board connections, while on board FSI signal trace lengths should match and have special care taken in the layout to enhance noise immunity.

In the tests performed there are isolation and differential transceiver devices being used on the TMDSFSIADAPEVM boards, which could introduce channel-to-channel skew. In a real world application that utilizes these same or similar devices, and/or varying signal trace lengths, the integrated skew compensation block within the FSI receiver module can be used to manage the skew that may occur between the clock and data signals. For more information on this topic, see *Fast Serial Interface (FSI) Skew Compensation*.

5.2 Three Device FSI Communication

Based on the test and comparison results given in Section 5.1, FSI communication for three devices has also been investigated. The test platform used is shown in Figure 5-5.

Figure 5-5. Test Platform for Three Devices Communication

Due to the nature of a daisy-chain connection, data will need to pass through a number of devices for the transmission from the first device to reach the last device. Therefore, to reduce latency it is important to make the data handling and forwarding time in each device as short as possible, especially when there are a number of devices in a connection loop. From the conclusion drawn in Section 5.1, to avoid having the CPU spending too much bandwidth moving data, it is recommended to use the DMA to serve FSI communication. The following tests utilize the same software discussed in Section 5 for the lead and node devices.

- **Test condition**

  Device 1 sends data -> Device 2 receives data -> Device 2 moves RX data to TX buffer and sends data to Device 3->…..-> Device 3 moves RX data to TX buffer and sends data to Device 1 -> Device1 receives data and verifies the data matches the originally sent TX data.
- **Test case**

  Data length of 8 words, 1 data line, TXCLK = 50 MHz, with setting ① for CPU control case and ② for DMA control case (Table 5-1) enabled.

In the tests, GPIOs are toggled within software when specific events occur during the communication and measured using an oscilloscope to obtain the respective timing data. In the figures below, the green signal represents the GPIO toggling of Device 1 (Lead device), the blue signal represents the GPIO toggling of Device 2 (Node device), and the magenta signal represents the GPIO toggling of Device 3 (Node device).
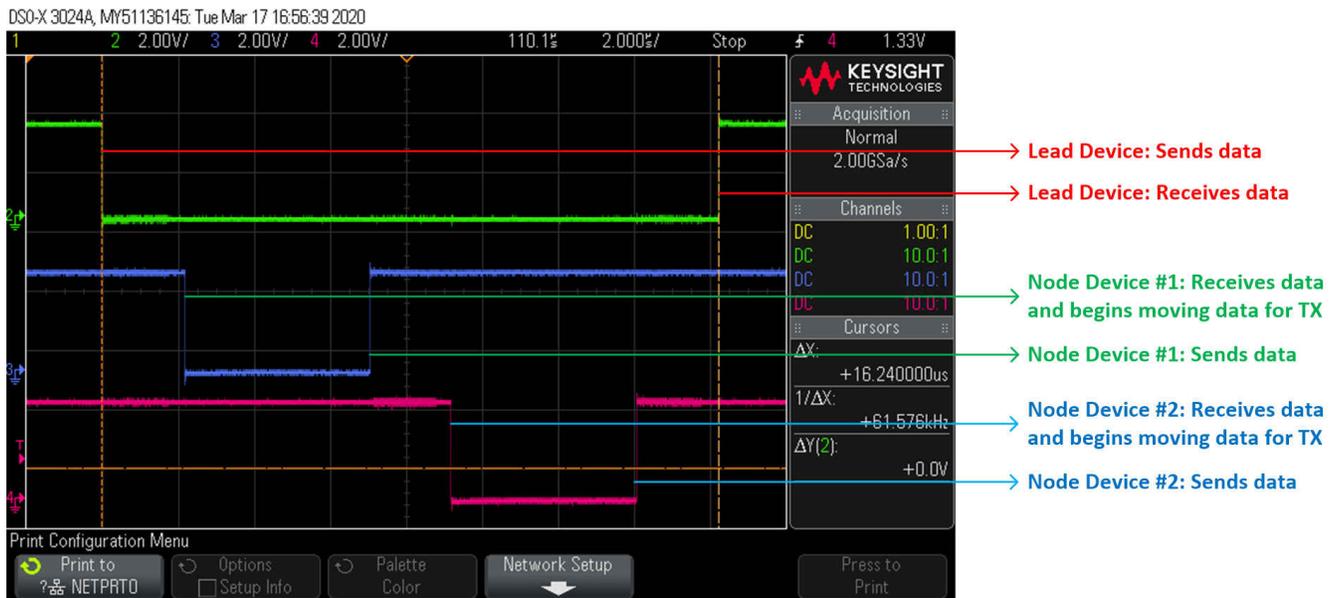
Figure 5-6. FSI Communication With CPU Control Among Three Devices

Figure 5-7. Time of Data Going Through One Device - CPU Control

For the CPU control case, the time needed for the data transmission to complete the three device daisy-chain loop is 16.2 μs, as shown in Figure 5-6. This time will increase by 7.1 us for each device added to the daisy-chain connection system, as shown in Figure 5-7. The 7.1 μs increased time per device includes the time for transmission and moving RX data to the TX buffer and registers.

14   *Using the Fast Serial Interface (FSI) With Multiple Devices in an Application*   SPRACM3D – OCTOBER 2020 – REVISED AUGUST 2021

Submit Document Feedback

Figure 5-8. FSI Communication with DMA Control Among Three Devices

Figure 5-9. Time of Data Going Through One Device - DMA Control

For the DMA control case, the time needed for the data transmission to complete the three device daisy-chain loop is 6.5 µs, as shown in Figure 5-8. This time will increase by 2.3 us for each device added to the daisy-chain connection system, as shown in Figure 5-9. The 2.3 µs increased time per device includes the time for transmission and moving RX data to the TX buffer and registers.
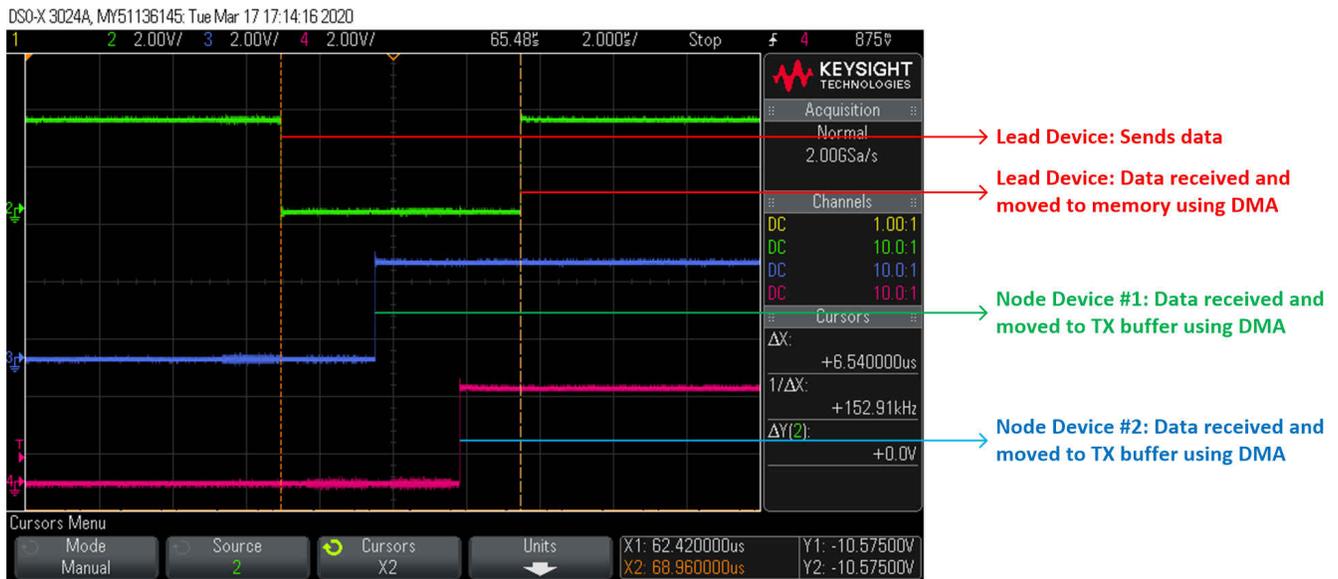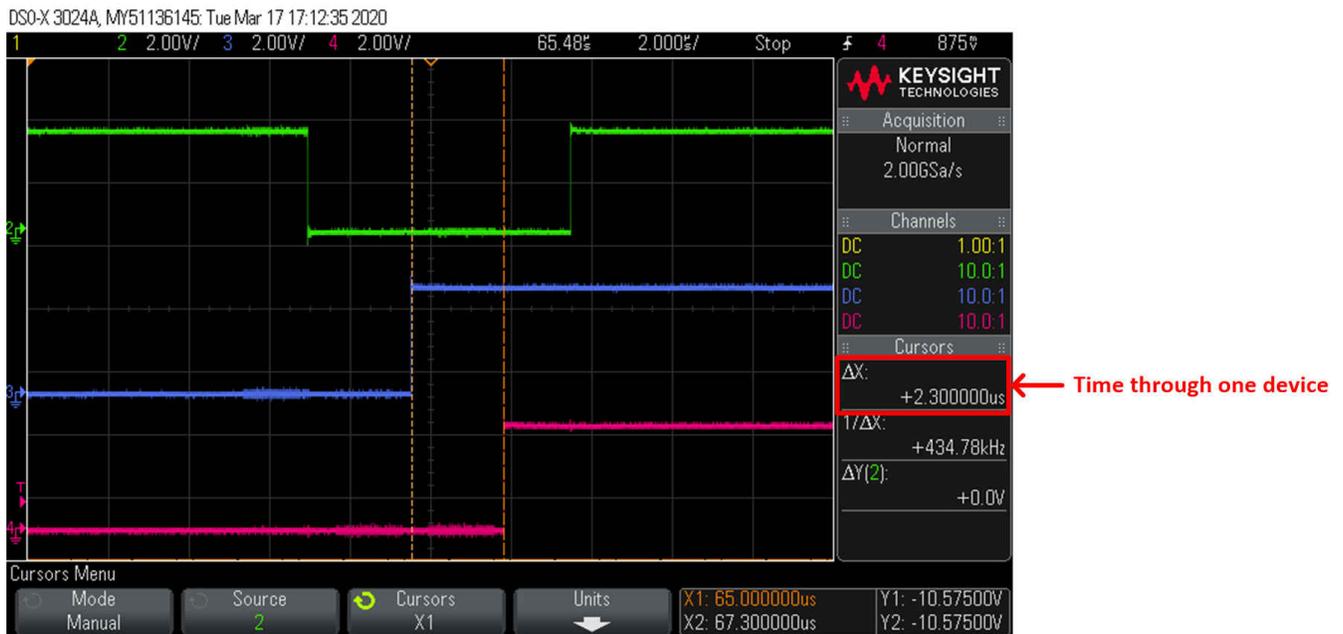
Further test results can be found in Table 5-5.

Table 5-5. Comparison of Using CPU Control and DMA Control in FSI Among Three Devices

	FSITXCLK (MHz)	# of Data Lines	Data Length (16-bit words)	Time of Data Going Through One Device (µs)	Time of the Full Connection Loop - 3 Devices (us)
CPU control	50	1	8	7.1	16.2
	50	1	16	11.8	26.8
DMA control	50	1	8	2.3	6.5
	50	1	16	4.0	11.8

6 Star Topology Tests

The star topology over FSI application example demonstrates a different kind of communication topology, showcasing how a central host device can receive information from multiple node devices at the same time rather than wait for packets to be forwarded through subsequent devices, like in the daisy-chain example. The advantages and disadvantages of the star topology are discussed in Section 2.

The defining requirements of the star implementation provided are hardware related, i.e. host device TX needing to have multi-drop functionality to each node, and MCU resource related, i.e. host device needing N number of RX instances. From a software perspective the central host device uses a new *star_broadcast* project while the N node devices use the same node device software utilized in the daisy-chain tests, details provided in Table 6-1.

Table 6-1. Software Example Projects - Star Topology

Project	Description	Supported Devices
fsi_ex_star_broadcast	Project for central host device in the star network	F2838x
fsi_ex_daisy_handshake_node	Project for N number of node devices in the star network	F28002x, F28004x, F2838x

The software flow of the *star_broadcast* project is similar to that of the lead device CPU Control daisy-chain project discussed in Section 5. The handshake mechanism will be slightly different as shown in Section 3.2. Upon completion of the handshake, the central host device will transmit a broadcast data frame to all of the node devices connected to its FSITX. The host will then wait to receive the data frame back from all connected node devices and then validate that each received frame matches the originally transmitted one, after which it will prepare and send a new data frame.

By default, the *star_broadcast* project has pre-made configurations for FSI RX instances A, B and C of the device. Each instance can be configured by setting the individual pre-processor directives below to "1". Additional FSI RX instances can be added if they are available on the host device.

```
//
// Enable FSI RX Instances
//
#define FSI_RXA_ENABLE          1
#define FSI_RXB_ENABLE          0
#define FSI_RXC_ENABLE          1
```

Timing measurements of the star topology will be very similar, if not the same, as those collected in the previous daisy-chain tests. Therefore the data provided in Table 5-4 can be utilized for this purpose.

7 Event Synchronization Over FSI

This section describes event synchronization using the Fast Serial Interface module for daisy-chain and star configuration using multiple devices.

7.1 Introduction

7.1.1 Requirement of Event Sync for Distributed Systems

In any real-time control system where more than one device operates together to perform a task, the devices send and receive critical information based on the control loop functions. The time to transmit the data from one device to the other can be different from device to device because of multiple factors including the distance between the devices, slight deviation in device clock during operation due to manufacturing uncertainties, thermal effects, ageing etc. The time-critical control loop may fail if the data received at the any of the device in the network is delayed or incorrect. It becomes important for the user to synchronize the operation of devices at regular intervals of time to correct the asynchronization due to uncertainties. The synchronization can be necessary for any of the events such as PWM signals, ADC Start of Conversions (ADC SoC), external triggers through GPIOs, etc.

In a real-time application, multiple configurations with lead and node devices can be arranged such as star network or daisy-chain network as described in Section 2. The star and daisy-chain configuration can be briefly understood from Figure 7-1 and Figure 7-2. All the node devices can be operating at different voltage levels and can be at different distances from the controller. Because of this, there may be significant difference in the time of any control signal sent from the lead device and the signal being received at each node device. The signal at each node device will arrive at different instances in time resulting in asynchronous operation of all the operating devices at every node. If the signals are fed to respective devices at different intervals of time, it can generate spurious operation which can cause the control loop application to breakdown. It becomes very important that the devices operate in synchronism to avoid the failure of system.

The aim is to synchronize all devices in the network topology with minimum amount of trigger latency and event jitter and also without using any additional wires other than the existing communication channel between the devices. This event synchronization has been achieved using the Fast Serial Interface (FSI) communication protocol.
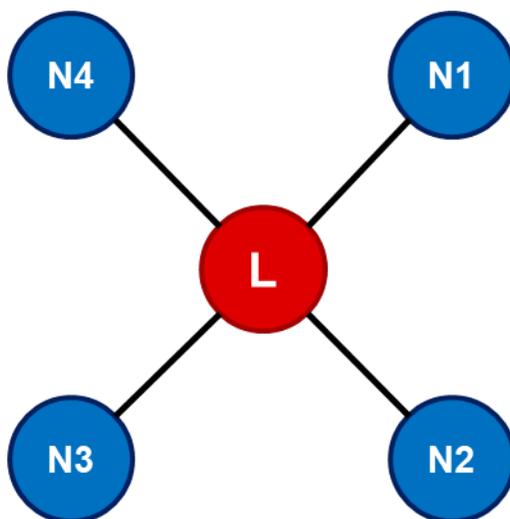

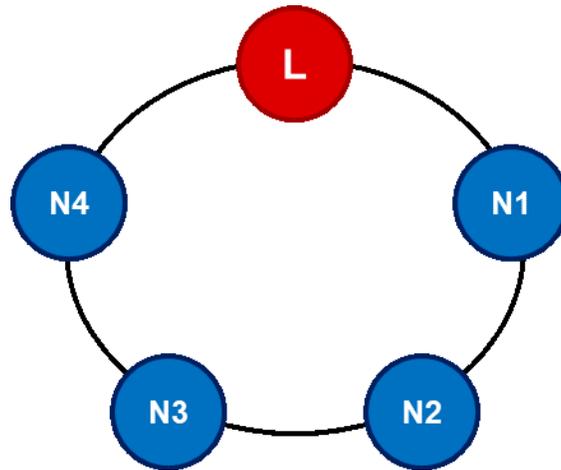**Figure 7-1. Star Network Configuration**

Figure 7-2. Daisy-Chain Network Configuration

7.1.2 Solution using FSI Event Sync Mechanism

Fast Serial Interface (FSI) is capable to send data at a rate up to 200Mbps across different operating voltage levels over the isolation barriers. FSI can achieve sub 100 nanoseconds time synchronized event triggering in multi-device network topologies. This eliminates the need for any additional synchronism signal between devices saving device resource and additional design cost.

To realize event sync mechanism over FSI, the lead device should trigger the FSITX module at regular time intervals, using operation similar to a timer, to send a synchronization request packet to all devices in the network. Receiving the packet from the lead device, each node device can initiate a local synchronization mechanism.

The sync packet itself may arrive at different time intervals for each node device. The interpretation of the sync packet on the individual node devices have to be adjusted locally.

Using 1 clock and 1 data line of the FSI module, the synchronization packet is sent at frequent time intervals which will ensure that all the node devices remain in synchronism. The regular application specific data is sent over these same lines, hence it reduces the need of extra wires to ensure synchronism amongst all the node devices.

7.1.3 Functional Overview of FSI Event Sync Mechanism

To showcase the event sync mechanism, a particular case of EPWM sync in a daisy chain network has been taken in the example provided. The task is to ensure that all the EPWM signals of respective node devices remain synchronized with the EPWM signal of lead device. In general, any event can be synchronized using the FSI event sync configuration as per the application. Since all the control applications use EPWM time base, it is sufficient to keep the EPWM of all the node devices synchronized. Typically, activities relating to ADC, comparator or any event function in the node device will be based on EPWM event or triggers.

The functioning of the PWM Sync over FSI can be understood considering one lead device and 2 node devices as shown in Figure 7-3. In the daisy chain network, FSI communication link is first established using the handshake mechanism as described in Section 3.1.
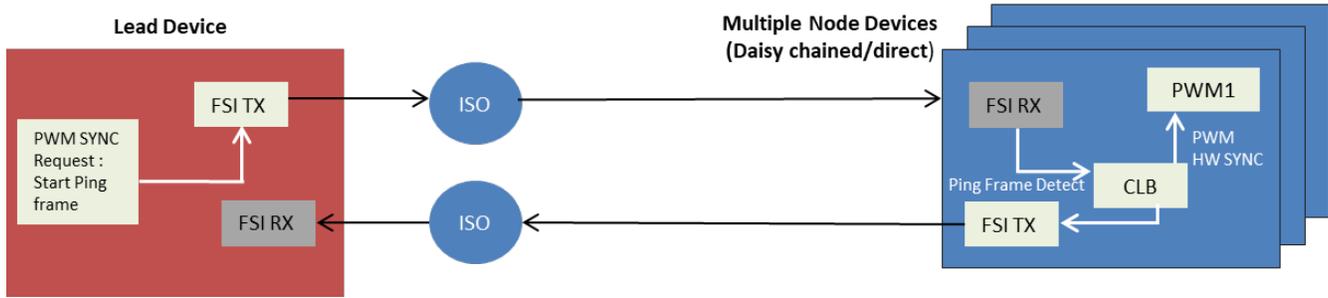

Figure 7-3. Daisy-Chain Communication Block Diagram

The ping packet, being the shortest possible FSI packet in terms of length, is sent by the lead device to all node devices at regular time intervals to verify the communication link and signal the node devices to synchronize with the lead device. The lead device FSITX trigger is controlled by local EPWM compare event configured by the user. On rising or falling edge of the EPWM as configured, FSITX is triggered to send the ping packet to node devices. The counter compare value for the EPWM signal depends on the number of devices in the chain, distance between devices, etc. which is to be configured as per the application by user.

As the ping packet sent by the lead device is received at a node, the FSI ping frame received signal (RX_PING_FRAME) generated at the node is connected internally to the Configurable Logic Block (CLB) module of the same node, as seen in Figure 7-3. The CLB routes the RX_PING_FRAME signal to immediately trigger the FSI TX ping forwarding to the next device in the chain. A configurable delay counter implemented inside CLB acts as a timer with period as *'match'* value fed to the CLB and restarts counting on reaching the *'match'* value. When CLB receives the RX_PING_FRAME signal, the counter starts counting and on reaching the *'match'* value, it generates an EPWM sync-in signal for the EPWM module of that node. In other words, the *'match'* value helps to provide a calculated delay from the time a ping packet signal is received at the node to the time when the PWM sync-in signal is generated at the local node device. Ideally, EPWM sync-in signal for every node will be generated at the same time and aligned with the lead device EPWM counter equals to zero event. A functional representation of process flow within a node device is showcased in Figure 7-4.
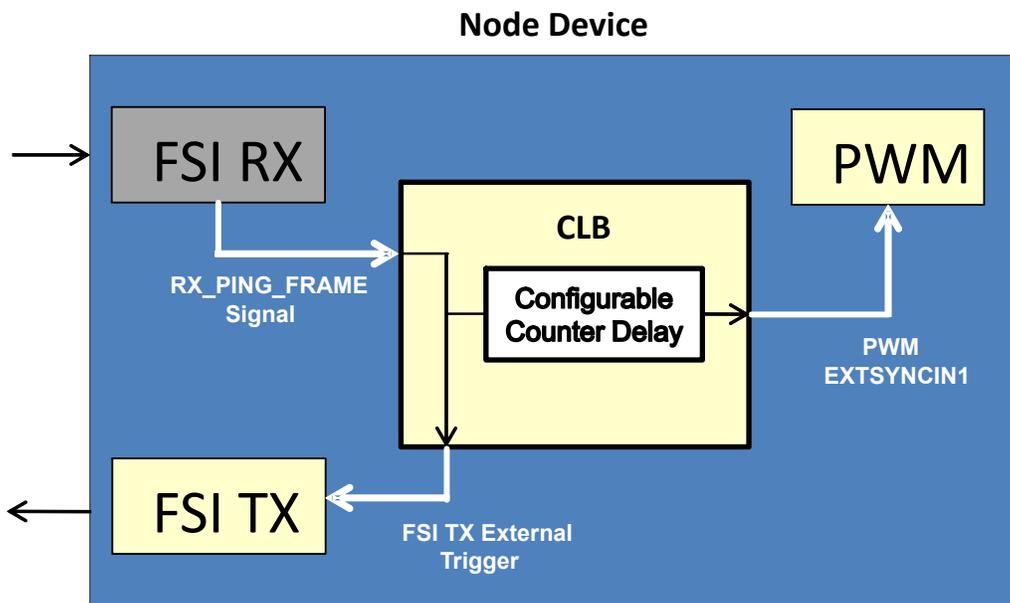

Figure 7-4. Node Device Implementation

The PWM signals wouldn't have remain synchronized over time since the oscillator clock cannot be identical for every device. The node devices in the chain experience the uncertainties in the form of jitter present in their respective EPWM signals as shown in Figure 7-5. The last device in the chain will experience the highest amount of jitter for a daisy-chain configuration because of the additive jitter from packet forwarding and synchronizer deviations at each device, discussed in Section 7.2.5. This way of utilizing the ping packet to generate the EPWM sync-in signal would keep the EPWMs synchronized for all devices in the chain.

Depending on the node in the daisy-chain network, the delay needed in generating the EPWM signal will be different. For example, for node 1, the ping packet will be received in a short time and for 8th node, the time taken for the ping packet signal to reach it will be higher due to forwarding and propagation delay, so the CLB counter count (*'match'* value) for node 1 has to be higher to ensure more delay in generation of EPWM sync-in signal than the count for node 8 to adjust for the transmission delay. The *'match'* value to be fed to the counter has to be set by the user and configured in the CLB configuration block, which will be dependent on the isolation barrier used, the distance between the nodes and the device in operation. Approximate value for *'match'* count to start has been provided in the project source file introduction section.
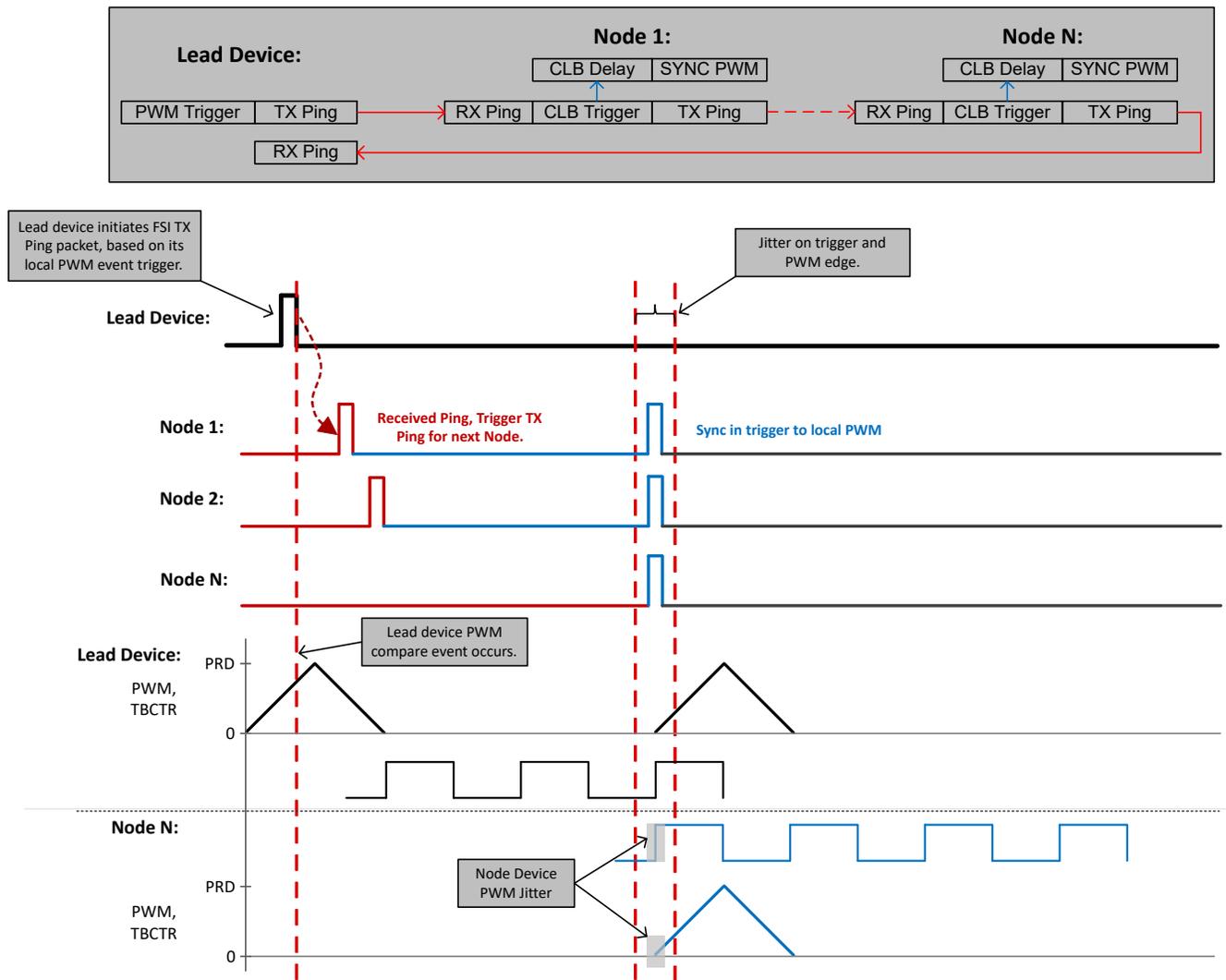

Figure 7-5. Daisy-Chain Sync Timing Diagram

20   *Using the Fast Serial Interface (FSI) With Multiple Devices in an Application*   SPRACM3D – OCTOBER 2020 – REVISED AUGUST 2021

Submit Document Feedback

7.2 C2000Ware FSI EPWM Sync Examples

This section provides technical and some additional details about the FSI event synchronization example provided in the C2000Ware.

7.2.1 Location of the C2000Ware Example Project

The example displaying the PWM Sync capability over the Fast Serial Interface module is showcased in the C2000Ware 3.04 version and the versions thereafter. The location for the same can be found at <C2000Ware_ROOT_FOLDER\examples\demos> folder, named *f28004x_fsi_daisychain_epwmsync* and *f28002x_fsi_daisychain_epwmsync*. The capability is showcased for two families of devices, f28004x and f28002x. Each device folder contains 2 projects, one for the lead device and other for the node device. Apart from the lead and node device folders, the *'common'* folder contains necessary target configuration file. The details about configuring the target configuration file is mentioned in Section 7.3.2. The projects can be run on either the controlCARD or the LaunchPad.

7.2.2 Summary of Software Configurations

Lead Device Configuration

The lead device frame transmission trigger to send the ping packet at regular intervals is done using EPWM1C module. The lead device is configured to send ping packets based on EPWM compare events. User needs to configure EPWM counter compare C register, EPWM_CMPC_VALUE, in the code as per the hardware configuration. This value will depend on external factors such as distance between two nodes, device oscillator clock, isolation barriers, etc. The frequency to send the ping packet can be in fractions of the EPWM clock frequency and can be further adjusted by configuring the EPWM_CMPC_EVENT_COUNT parameter. The EPWM_CMPC_EVENT_COUNT parameter sets the EPWM event trigger prescalar value, which configures the frame transmission trigger to occur on every Nth compare event. The maximum value of the prescalar is 15.

The duty cycle and frequency of EPWM can be adjusted in the code by updating EPWM_CMPA_VALUE for EPWM1A duty cycle, EPWM_CMPB_VALUE for EPWM1B duty cycle and EPWM_TIMER_TBPRD for frequency. Default values are 50% duty cycle at 20kHz EPWM frequency.

Node Device Configuration

The node devices receive the ping packet and they have to generate the EPWM sync signals after a certain delay depending on the order of device in the node. The delay configuration is done in the *fsi_daisy_epwmsync_node.syscfg* file provided along with the project. The configuration has to be done using the counter value *'match'* in the Configurable Logic Block (CLB) under the Counter0 tab as shown in Figure 7-6. FSM block contains the logic to generate the EPWM sync-in signal after the match count delay once it gets the ping packet received signal. The CLB configuration done using the sysconfig file can be seen in the clb.html file under the syscfg tab in the project when the project is built using the CPU1_CLB build configuration as is shown in Figure 7-7.
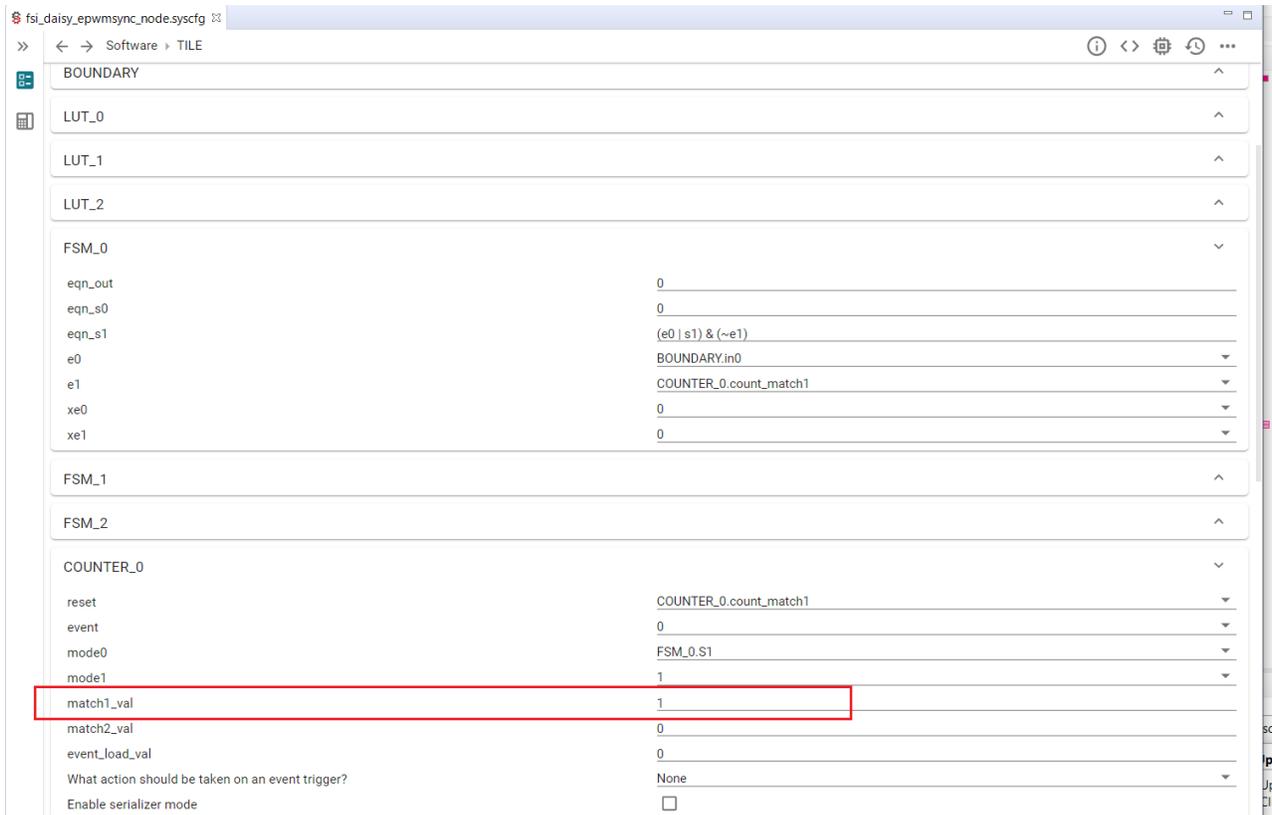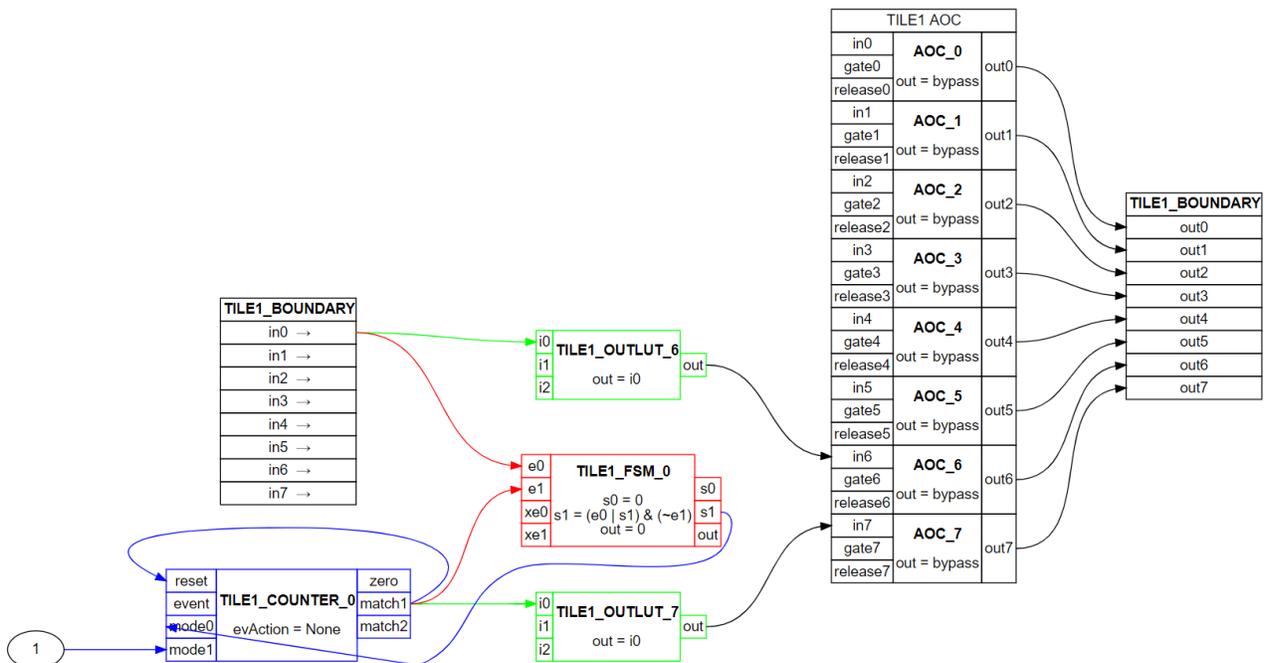
Figure 7-6. CLB Block Configuration

Figure 7-7. Configured CLB Block

22   *Using the Fast Serial Interface (FSI) With Multiple Devices in an Application*   SPRACM3D – OCTOBER 2020 – REVISED AUGUST 2021

Submit Document Feedback

Similar to the lead device, the user can configure the EPWM frequency and duty cycle for the respective node using EPWM_TIMER_TBPRD for frequency, EPWM_CMPA_VALUE for EPWM1A duty cycle and EPWM_CMPB_VALUE for EPWM1B duty cycle.

The CLB configuration differs slightly from F28004x to F28002x. F28004x devices have to route CLBx_OUT to FSITX external trigger connection through the EPWM XBAR while the F28002x devices have been updated to directly connect FSITX external connection to CLBx_OUT. User can refer to the device reference manual which explains the specific information.

7.2.3 1 Lead and 2 Node F28002x Device Daisy-Chain Tests

Hardware Setup and Configurations

The provided daisy-chain example is run on 1 lead LAUNCHXL-F280025C device with 2 node LAUNCHXL-F280025C devices along with TMDSFSIADAPEVM boards with each lead and node devices. The devices are connected using a cable with length of 1ft. The operating PWM frequency has been considered at 20kHz with 50% duty ratio. The FSI clock is configured to operate at 50MHz. The CLB counter '*match*' values as per the calculation of Node 1 having 43*(N-1) will have '*match*' value as 43 and Node 2 will have value as 1.
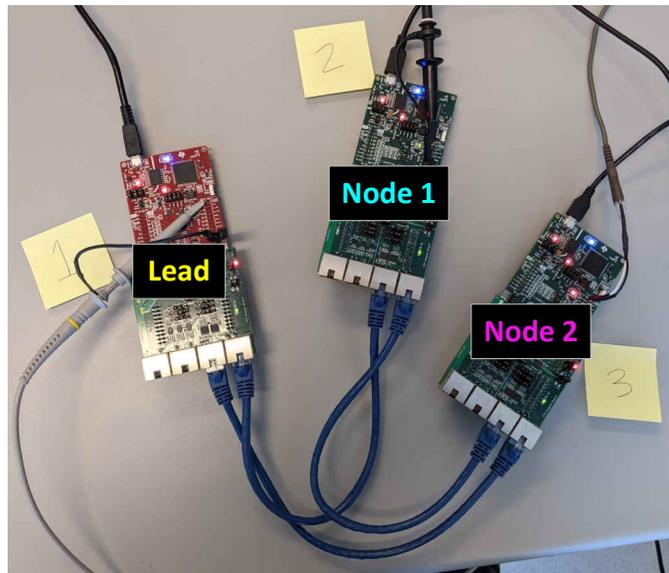

Figure 7-8. 1 Lead - 2 Node Device Setup

Experimental Results

The EPWM signals are first captured without enabling the EPWM sync capability over FSI, as shown in Figure 7-9. Without the FSI event sync solution enabled, the node device EPWM signals will gradually move out of phase of the lead device's EPWM signal due to small differences of each device's oscillator clock.

Figure 7-9. 1 Lead - 2 Node Device without EPWM Synchronization

The EPWM signals after being synchronized have been captured and showcased in Figure 7-10.

The EPWM rising edge jitter at lead device and both node devices are captured after running the example for a week to capture all possible uncertainties and shown in Figure 7-10. Ideally, the jitter measured at the node devices should be as small as possible. It is the noise in the chain present because of inconsistency between the devices, isolation barriers, the communication link, etc. CLB also brings in certain delay causing jitter, called as synchronizer delay discussed in Section 7.2.5. Higher number of devices and longer distance between nodes will result in worsening of the PWM jitter.
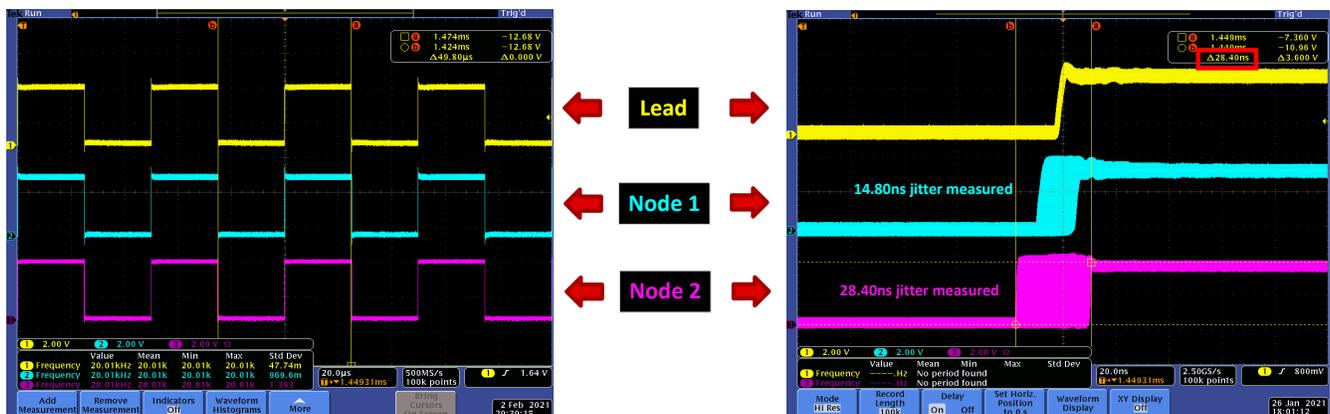

Figure 7-10. 1 Lead - 2 Node Device In-Sync EPWM

24   *Using the Fast Serial Interface (FSI) With Multiple Devices in an Application*   SPRACM3D – OCTOBER 2020 – REVISED AUGUST 2021

Submit Document Feedback

7.2.4 1 Lead and 8 Node F28002x Device Daisy-Chain Tests

Hardware Setup and Configurations

This example showcases the operation of one lead device and 8 node devices with a cable length of 1ft between each node, except last two nodes having a longer cable length distance of 5m as shown in Figure 7-11.

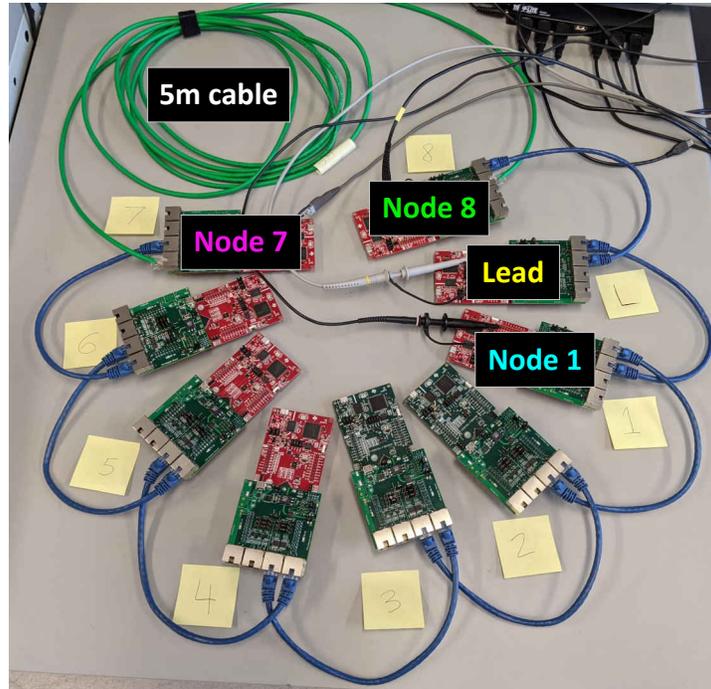The duty cycle of PWM output at every node is kept 50% with the frequency of 20 kHz.

Figure 7-11. 1 Lead - 8 Node Device Setup

Experimental Results

The EPWM waveform of node 1, node 7 and node 8 are captured in the oscilloscope to verify the synchronous operation. The results captured without EPWM synchronization enabled and with the EPWMs synchronized are compared to showcase the capability of EPWM synchronization over FSI. The difference in the jitter from the 2-node configuration can be observed clearly.

Maximum jitter observed at the 8th node, with test performed over a week to capture all uncertainties, is in sub 100 nanoseconds range which does not have a large affect on the edge of the EPWM frequency.
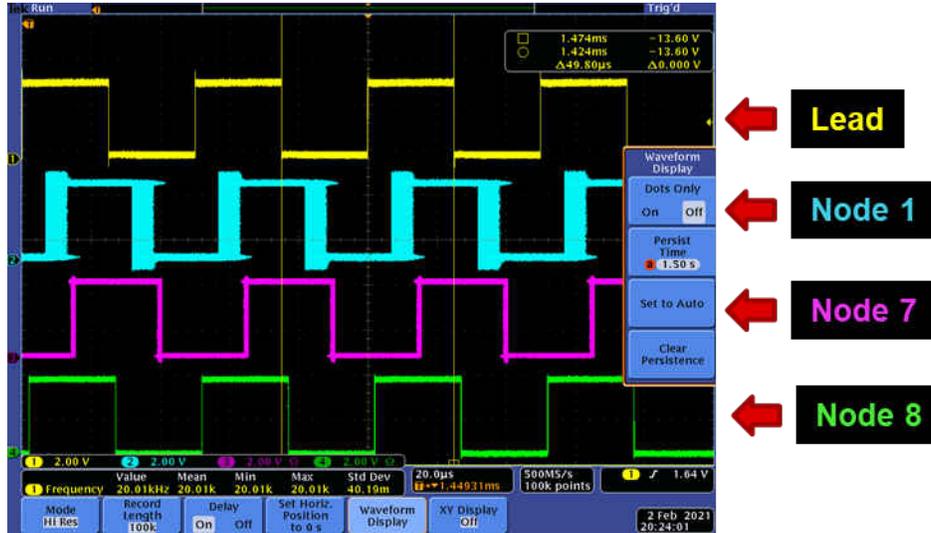

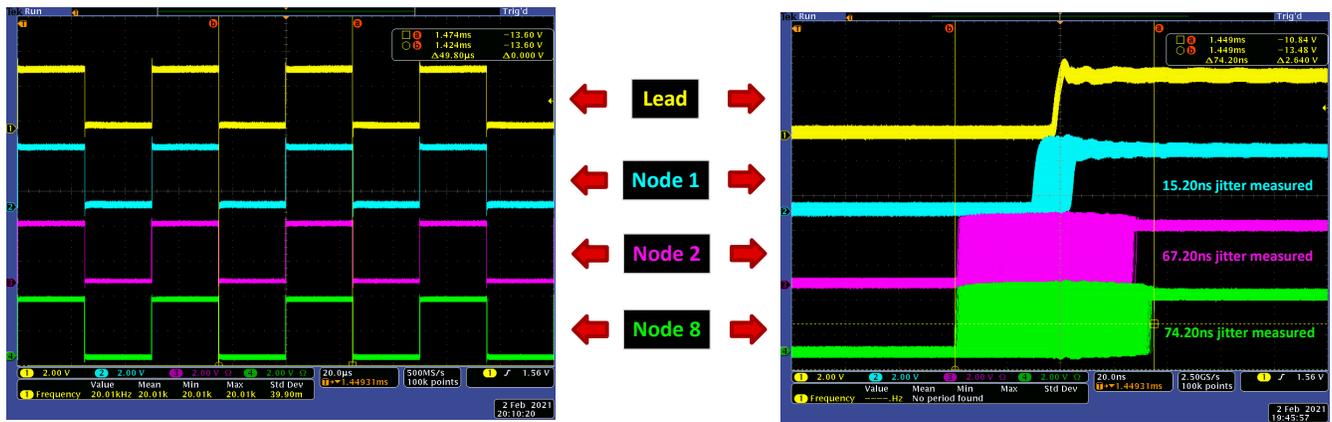**Figure 7-12. 1 Lead - 8 Node Device without EPWM Synchronization**

Figure 7-13. 1 Lead - 8 Node Device In-Sync EPWM

7.2.5 Theoretical C2000 Uncertainties

The event jitter as described in the earlier sections is a result of the uncertainties in the oscillator clocks from device to device, transmission distance between the devices, digital isolators and/or differential device disturbance, and so forth. Additional uncertainty is also induced by the synchronism from FSIRX to the CLB module to FSITX signal path configured inside the device, shown in Figure 7-14.
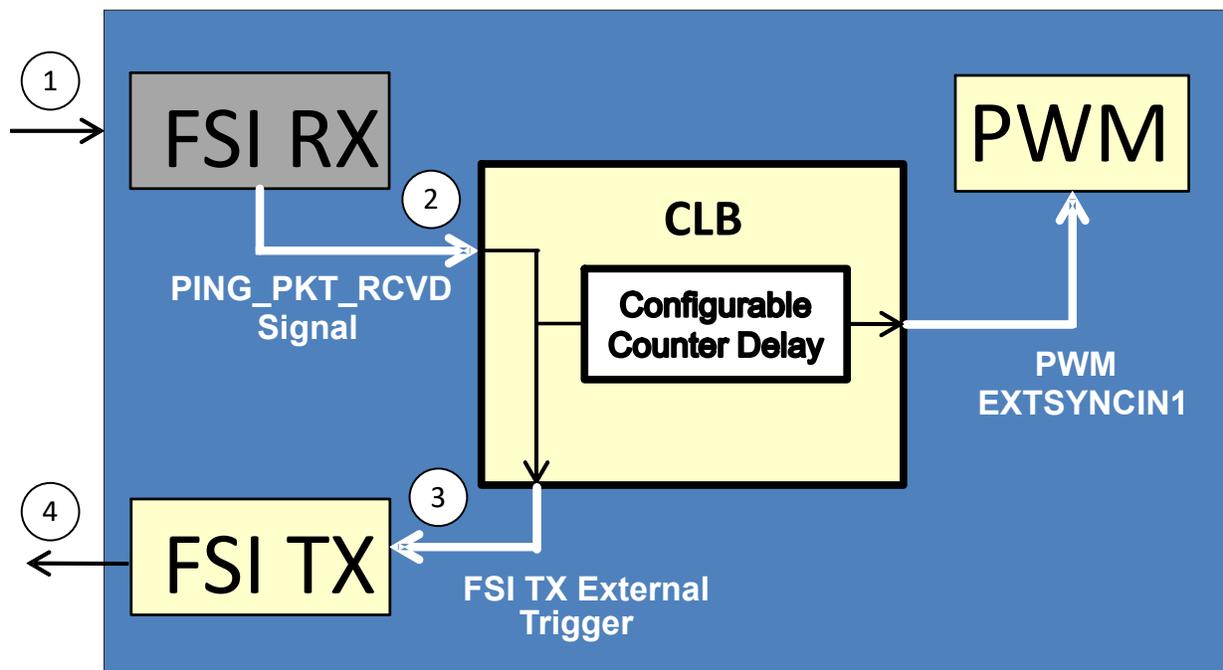

Figure 7-14. Internal C2000 FSI Event Trigger Path

Along the internal C2000 FSI event trigger path the clock domain changes between FSI RX clock, device SYSCLK (CLB clock), and the FSI TX clock. Each time the signal moves from one clock domain to another it encounters a synchronizer that adds some amount of variable cycle delay (not the same number each time). The theoretical cycle count uncertainty of the event trigger path from section 1 to 4 shown in Figure 7-14 is described below.

Path 1-2: FSI Ping frame being received at RX module to generation of the PING_PKT_RCVD signal. Clock domain change from FSI RX CLK to SYSCLK is made resulting in 0-2 SYSCLK cycles of uncertainty.

Path 2-3: Ping_PKT_RCVD signal passing through the CLB module to externally trigger the FSI TX module to send a Ping frame. Clock domain change from SYSCLK to FSI TX CLK is made resulting in 0-2 SYSCLK cycles of uncertainty.

Path 3-4: FSI TX external trigger to FSI TX ping frame generation. Clock domain change from SYSCLK (CLB Clock) to FSI TX CLK is made resulting in 0-2 FSI TX CLK cycles of uncertainty.

From the above the total worst-case theoretical cycle uncertainty can be calculated:

Note
Total Worst-Case Theoretical Cycle Uncertainty = 4 SYSCLK cycles + 2 FSI TXCLK cycles.

For a device operating at a SYSCLK frequency of 100 MHz, the theoretically calculated amount of jitter induced will be around 40 nanoseconds, assuming TXCLK is the same as SYSCLK. The practical value achieved will be a small fraction of the calculated theoretical value because this is the worst-case theoretical consideration. The possibility of observing this worst-case value is statistically low for any device in a network topology. This can be seen in the experimental results shown earlier, where the actual jitter measured for 8 node devices is measured to be 75 nanoseconds while the theoretical worst case will be 320 nanoseconds, considering 40 nanoseconds of jitter induced per device in the chain.

7.3 Additional Tips and Usage of FSI Event Sync

7.3.1 Running the Example

To run the example, 1 lead with 'n' number of node devices, F28004x or F28002x, have to be connected in the daisy-chain configuration. The lead and node examples for the respective devices have to be imported in CCS. Once the projects are successfully imported, open the target configuration block available under the 'View' tab. Right clicking anywhere in the empty space within the target configuration block allows the user to import any existing target configuration. The target configuration file has to be imported from the *'common'* folder available with the main project folder named *as <fsi_daisychain_1lead_2node.ccxml>.* The given file is configured for 1 lead and 2 node devices, which can be configured by the user as per the hardware requirement. The details on how to add a new device to the existing target configuration file are provided below in Section 7.3.2 .

After successfully adding and testing the device connection using the target configuration file, the target configuration has to be launched by right clicking on the target configuration file. Note that the lead and node project have to be successfully built before launching the target configuration file. After the target configuration file is launched, connecting to the device and loading the project to the device has to be done using the connect and load icons provided in the menu bar on the top.

Another way to run the project is to open multiple windows of CCS, equal to the number of devices, using different workspaces and individually build and load the project in each device through its dedicated CCS window.

The variables *'data_frame_cntr'*, *'error'* and *'FSIRxintCount'* can be added in watch variables of node project to observe the data frames being transmitted and received through the CCS window. The user may also want to use the *'epwm_cmpc_value_var'* and *'epwm_sync_event_count_var'* to dynamically change the duty cycle and the prescalar value in run-time environment in the infinite loop.

7.3.2 Target Configuration File

The target configuration file (.ccxml) helps to connect with one or more target hardware. With multiple target hardware for the given example, it becomes important to set the target configuration file appropriately to ensure successful connection to each device. Detailed documentation for the connection to multiple device and updating the serial number for the device has been provided in *TI Documentation*. Additional documentation on developing your own custom target configuration file and understanding the debug probe for the device can be found in the links provided, *[1]* and *[2]*. Steps to find and update existing target configuration are provided in *Custom Target Configuration* document.

Note

The target configuration file provided in the *'common'* folder already has serial number configured for the devices, which will not be the case for the user. For the use case, the user can either reprogram the serial number in the target configuration file or update the serial number of the debug probe through the steps mentioned in the provided links.

7.3.3 Usage of Event Sync for Star Configuration

The star configuration is expected to be much simpler and provide better performance over the daisy-chain case since the delay from device-to-device transmission won't be present. The factors which will affect the synchronism of EPWM signals for multiple devices will be the manufacturing uncertainty of the oscillator clock, distance between the devices and the isolators separating the devices. Daisy-chain example, in addition to the factors in Star connection, contains additive delay from device-to-device communication and the device uncertainties will add up for the last device in the chain, invoking higher complexity.

The user will need to make some minor changes in the existing code to ensure EPWM synchronization for the star configuration. All links between the lead and node devices will be direct connections. For the C2000Ware example code, the star configuration will have all the CLB counter *'match'* values in the similar range dependent on the distances between lead and node devices and uncertainties in the communication path. The *'match'* values will not add up from device to device as seen in the daisy-chain configuration. Therefore, the star event sync will offer lower event jitter. The lead device will have to be chosen such that it contains the number of FSIRX modules equal to or higher than the number of nodes so that one lead device is capable to connect with multiple node devices. For detailed information on the star configuration, see Section 6.

8 References

- Texas Instruments: *TMS320F28002x Microcontrollers Data Sheet*
- Texas Instruments: *TMS320F28002x Microcontrollers Technical Reference Manual*
- Texas Instruments: *TMS320F2838x Microcontrollers Data Sheet*
- Texas Instruments: *TMS320F2838x Microcontrollers Technical Reference Manual*
- Texas Instruments: *TMS320F28004x Microcontrollers Data Sheet*
- Texas Instruments: *TMS320F28004x Microcontrollers Technical Reference Manual*
- Texas Instruments: *TIDM-02006 Distributed multi-axis servo drive over fast serial interface (FSI) reference design*
- Texas Instruments: Distributed Power Control Architecture With Multiple MCUs Over FSI
- TMDSFSIADAPEVM: FSI Adapter Board
- F280025C ControlCARD Evaluation Modules
- LAUNCHXL-F280025C Evaluation Module

9 Revision History

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

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 (https: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.