Application Note

FSI Bandwidth-Optimization for Multi-axis Servo Control

Chen Gao, Bruce Liu, Thomas Leyrer *System Engineering and Marketing*

ABSTRACT

TI Sitara™ microcontrollers offer two types of Programmable Real-time Unit (PRU) subsystems: PRU-Industrial Communication Subsystem (PRU-ICSS) and PRU-Industrial Communication Subsystem - Gigabit (PRU_ICSSG). PRU-ICSS is available for the AM335x, AM437x, AM57x series and K2G devices. PRU_ICSSG is available on AM243x, AM65x, and AM64x, and can be independently programmed to achieve some custom requirements with high real-time requirements to achieve product differentiation.

In industrial applications, multiple devices are often required to communicate with each other in a fast, low-latency, and synchronized manner. For example, this requirement exists in the multi-axis servo control system architecture. The Fast Serial Interface (FSI) is a new communication peripheral created for TI's real-time control microcontrollers (MCUs) to extend the reliable high-speed communication capabilities to multiple devices in a system. This application report describes how to use PRU to increase FSI communication bandwidth, provides test performance based on FSI high-speed communication in different configurations and modes, and applies these concepts in a multi-axis servo control system.

Table of Contents

List of Figures

List of Tables

Trademarks

Sitara™ and LaunchPad™ are trademarks of Texas Instruments.
Arm® is a registered trademark of Arm Limited.
All trademarks are the property of their respective owners.

1 Introduction

FSI bus topology is often used in multi-axis servo applications.

The key requirement for FSI handler is the *Low Latency* for the communication cycle. Motor drive applications normally use a 20-kHz cycle time and a maximum of eight axes.

Figure 1-1 shows the FSI bus connection schematic diagram.

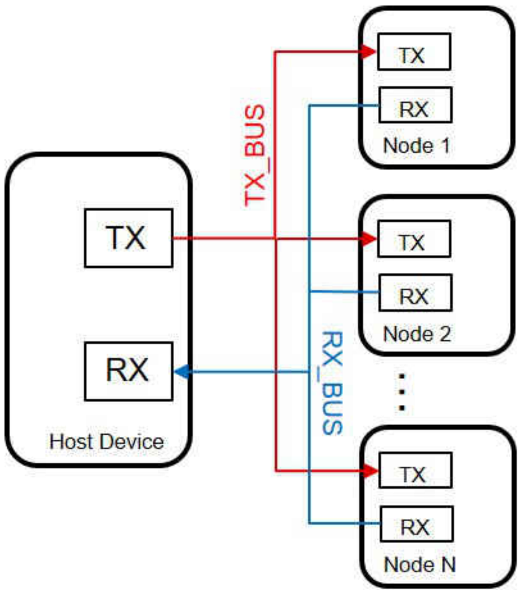


Figure 1-1. FSI Bus Connection Schematic Diagram

In terms of the amount of data sent and received by the FSI bus, the host device exchanges data with N target nodes in sequence according to the communication cycle. The Sitara AM243x is selected for the host device. For typical applications, the controller node sends 32 bytes and the device nodes reply with 32 bytes in one communication cycle. Considering the maximum of eight axes, the controller node needs to send 8 × 32 bytes through the FSI bus, and receive 8 × 32bytes of response data from the device nodes.

The known design challenges include the following:

- FSI module is *not* supported by AM243x DMA transfers
- Internal FSI benchmark to transmit 16-bit data (TX 32 bytes + RX 32 bytes) takes about 8.2µs without a data transfer
- Maximum of eight axes can require > 65.6µs to communicate in sequence which does not fulfill the requirements (maximum 50µs, 20 kHz)

Use PRU_ICSSG as the FSI handler (< 5µs with data transfer per axis) for communication to achieve low latency on the FSI data transfer and mapping of FSI data to multiple Arm® cores.

Figure 1-2 shows the communication path and latency estimation.

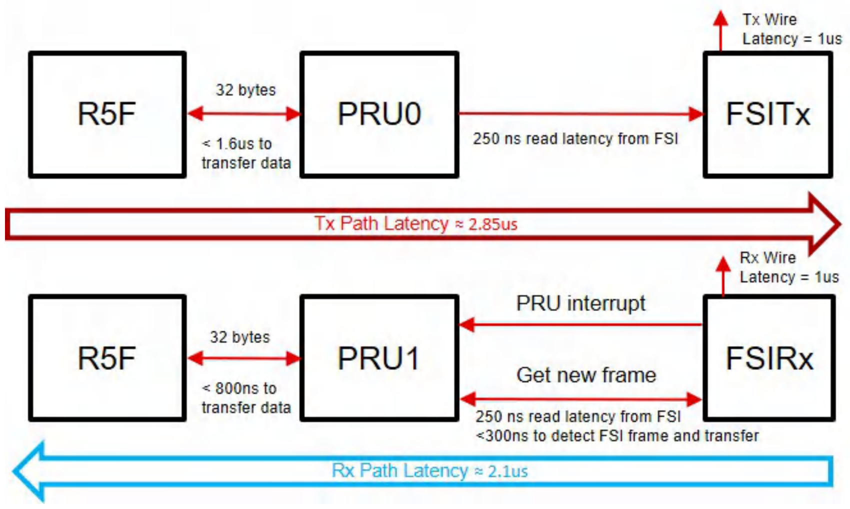


Figure 1-2. Communication Path and Latency Estimation

To reduce the overhead of manual data filtering of device nodes, the Frame Tag 4-bits that comes with the FSI bus is used to distinguish eight device nodes. The host device modifies the frame tags in turns in each communication cycle to achieve automatic filtering of the device axis data. Figure 1-3 shows the frame format.

Idle State	Preamble	Start of Frame	Frame Type	User Data	Data Words	CRC Byte	Frame Tag	End of Frame	Postamble	Idle State
	1111	1001	4 bits	8 bits	1-16 words	8 bits	4 bits	0110	1111	

Figure 1-3. Frame Format of FSI

To realize FSI bandwidth optimization, design a mechanism for FSI data transfer using the ICSSG module with multiple channels as shown in Figure 1-4.

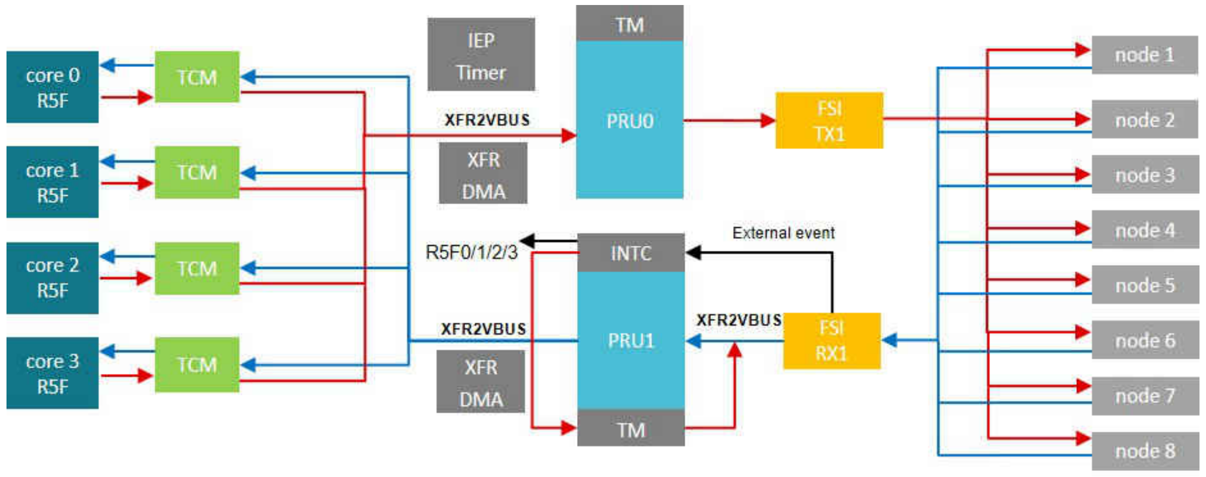


Figure 1-4. FSI Optimization Using Multiple Channels and PRU

2 Implementation

2.1 FSI TX module

In the FSI TX module, use the following example to setup and run FSI from Arm without interrupt:

fsi_loopback_polling (ti.com)

In addition, PRU executes the time trigger, sending data using the IEP timer and task manager. Figure 2-1 shows the design process of the FSI TX module.

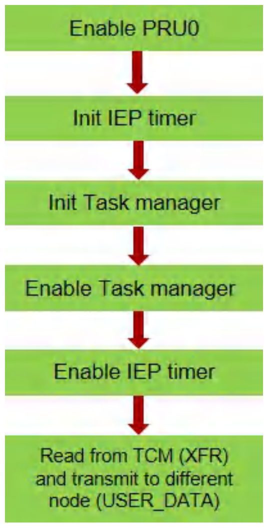
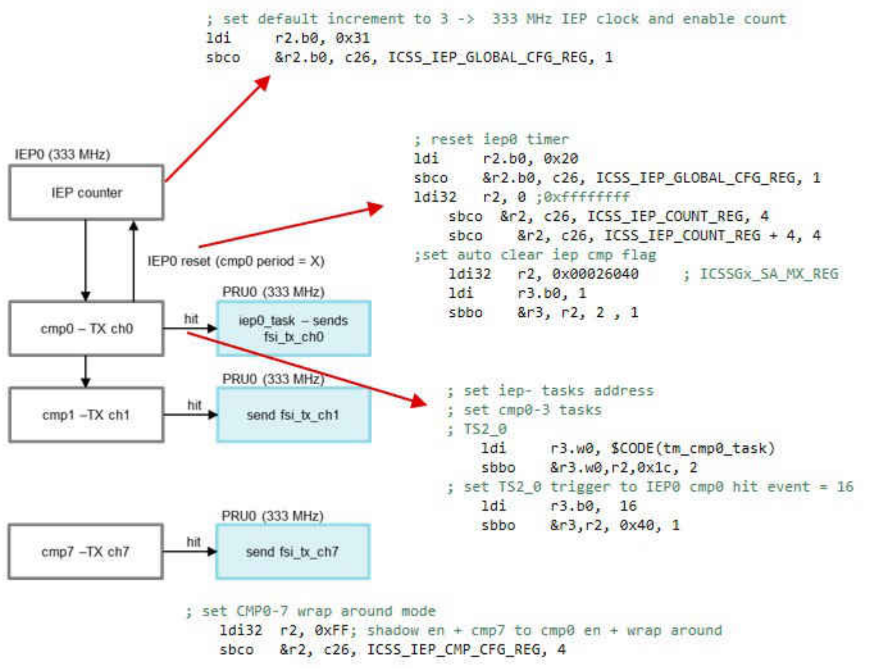


Figure 2-1. FSI TX Module Design Process

The specific steps to use the IEP to trigger FSI transmitting are listed in the following bullets:

1. Init IEP timer – set the clock to 333 MHz, cmp0 wrap around, the count value is set to 3 to achieve a count of 1 approximately equal to 1ns, so that cmp 0- 7 values can be set easily
2. Init task manager with cmpx_hit trigger
3. Enable task manager
4. Enable IEP timer
5. For debug use PRU_GPO to send a pulse

Figure 2-2 shows the IEP timer configuration.



```
; set default increment to 3 ->  333 MHz IEP clock and enable count
ldi     r2.b0, 0x31
sbco    &r2.b0, c26, ICSS_IEP_GLOBAL_CFG_REG, 1
```

```
; reset iep0 timer
ldi     r2.b0, 0x20
sbco    &r2.b0, c26, ICSS_IEP_GLOBAL_CFG_REG, 1
ldi32   r2, 0 ;0xffffffff
    sbco  &r2, c26, ICSS_IEP_COUNT_REG, 4
    sbco    &r2, c26, ICSS_IEP_COUNT_REG + 4, 4
;set auto clear iep cmp flag
    ldi32   r2, 0x00026040     ; ICSSGx_SA_MX_REG
    ldi     r3.b0, 1
    sbbo    &r3, r2, 2 , 1
```

```
; set iep- tasks address
; set cmp0-3 tasks
; TS2_0
    ldi     r3.w0, $CODE(tm_cmp0_task)
    sbbo    &r3.w0,r2,0x1c, 2
; set TS2_0 trigger to IEP0 cmp0 hit event = 16
    ldi     r3.b0,  16
    sbbo    &r3,r2, 0x40, 1
```

```
; set CMP0-7 wrap around mode
   ldi32  r2, 0xFF; shadow en + cmp7 to cmp0 en + wrap around
   sbco   &r2, c26, ICSS_IEP_CMP_CFG_REG, 4
```

Figure 2-2. IEP Timer Configuration

In terms of configuration of the FSI TX module, refer to the following paths to find the R5F source code:

`${MCU_PLUS_SDK_PATH}/source/drivers/fsi/vx/fsi_tx.c`

And rewrite the code with C or assembly language in ICSSG.

2.2 FSI RX Module

Use the following example in the FSI RX module to setup and run FSI from Arm with interrupt:

fsi_loopback_interrupt (ti.com)

The receiving flag is sent to the PRU_INTC module by an external event when the FSI RX buffer is written.

Use the interrupt controller (INTC), data processing accelerator (task manager) and data movement accelerator (XFR2VBUS) on PRU for the receive path.

Figure 2-3 shows the design process of the FSI RX module.

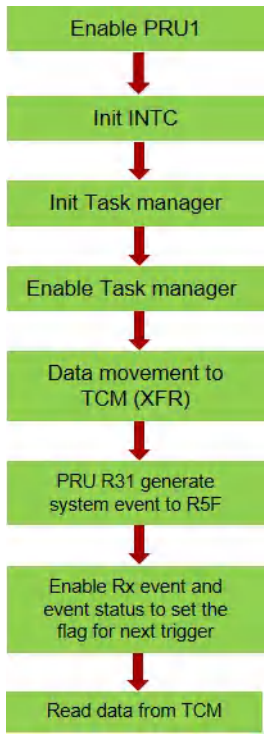

Enable PRU1
Init INTC
Init Task manager
Enable Task manager
Data movement to TCM (XFR)
PRU R31 generate system event to R5F
Enable Rx event and event status to set the flag for next trigger
Read data from TCM

Figure 2-3. FSI RX Module Design Process

Figure 2-4 shows the specific steps for the external event trigger INTC and hit task manager.

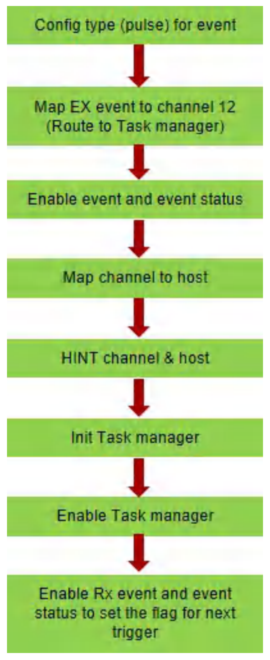

Config type (pulse) for event
Map EX event to channel 12 (Route to Task manager)
Enable event and event status
Map channel to host
HINT channel & host
Init Task manager
Enable Task manager
Enable Rx event and event status to set the flag for next trigger

Figure 2-4. INTC_TASK Manager Configuration Programming

Note

Make sure the event raw status bit is SET then the interrupt generates after INTC and Task Manager configuration is done.

The specific steps for system interrupt generate are as follows and the FSI receive data path is shown in Figure 2-5.

- Generate System Events (pulse) and mapping to R31
- Internal event 16 to 19 maps to host 2–5 (channel 2–5) which exported from PRU_ICSSG and mapped to device level-interrupt controllers
- PRU Interrupt requests to Arm

```
HOST 2 (Level) → R5FSS0_Core0_INTR_IN_122
…
HOST 5 (Level) → R5FSS0_Core0_INTR_IN_125
```

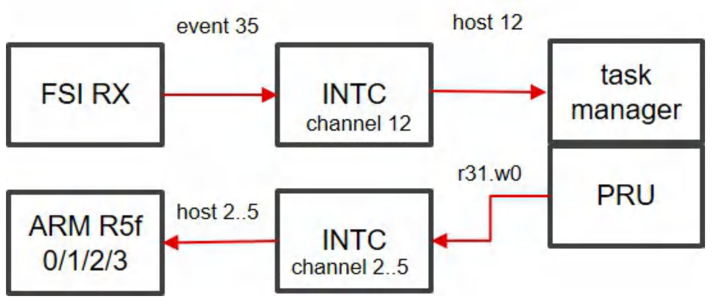


Figure 2-5. FSI Receive Data Path

2.3 Verification

To verify the mechanism for FSI data transferring using the ICSSG module, the code on the LP-AM243 is tested. Figure 2-6 shows the test set up.

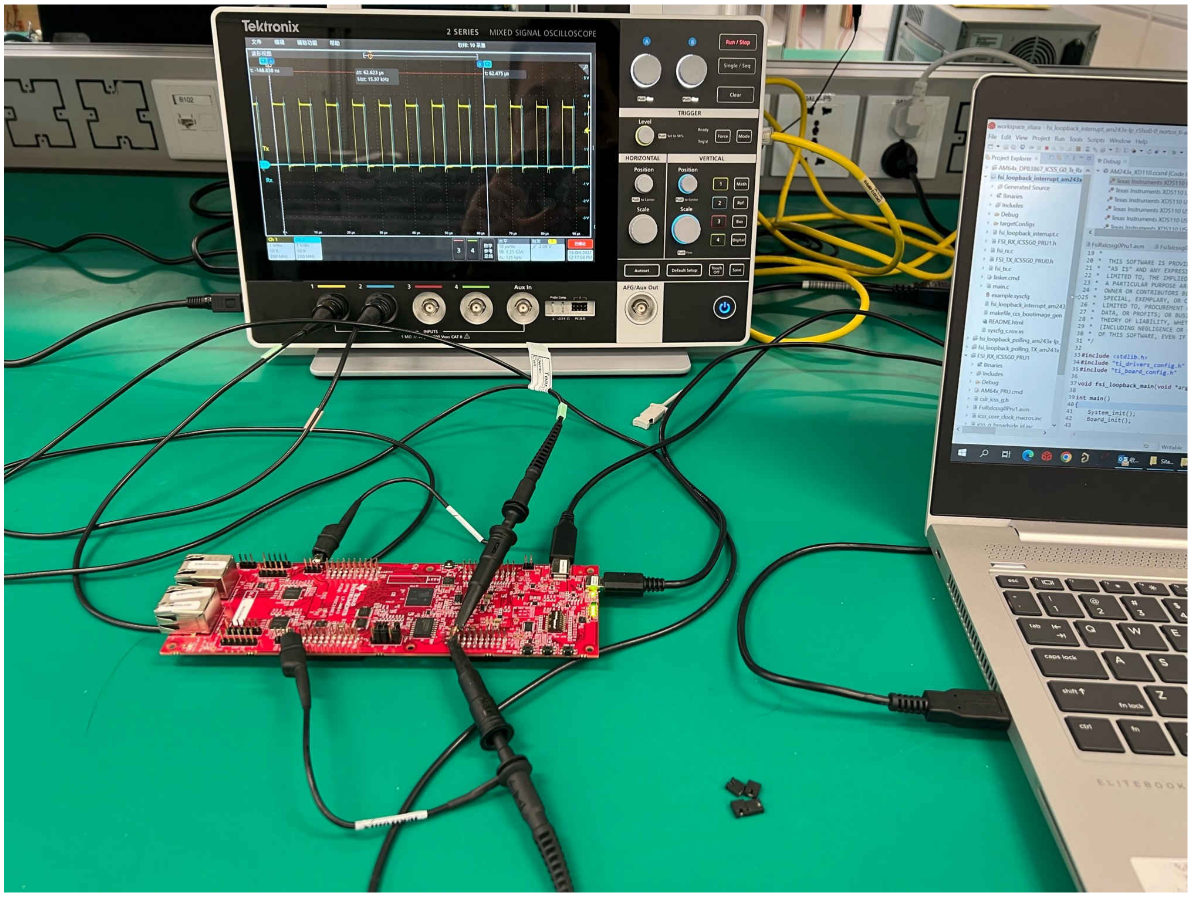


Figure 2-6. Test Set up

The AM243x LaunchPad™ provides an FSI header with FSI_TX0 and FSI_RX0 channels. To perform external loopback communication, jumper the two data lines and clock signals as Figure 2-7 shows.

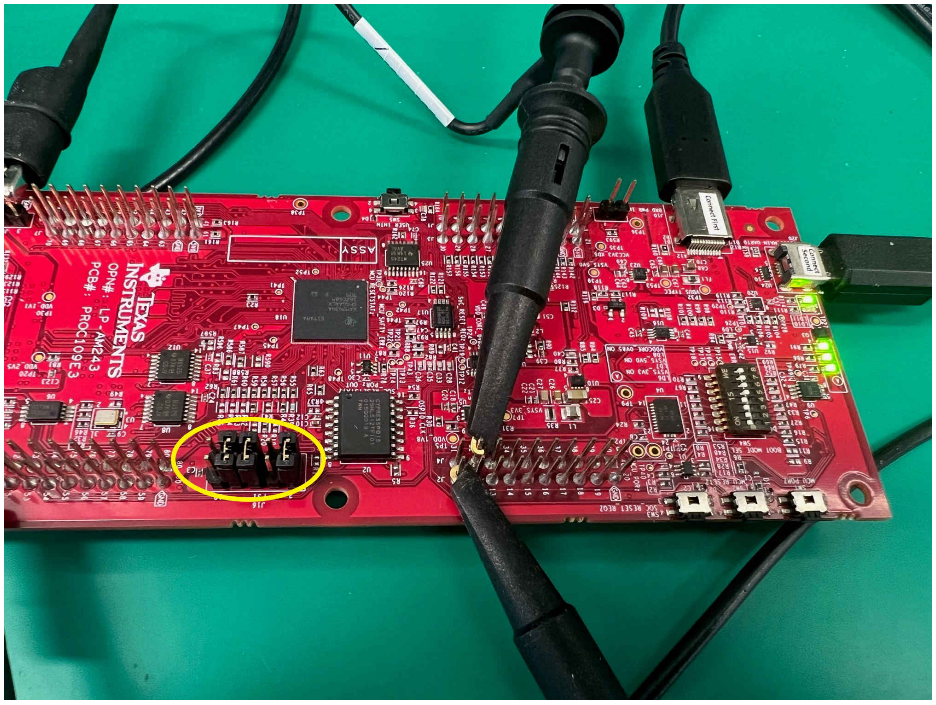


Figure 2-7. Jumper the Two Data Lines and Clock Signals

The test conditions for R5 core control:

→ TX, RX module configuration with handshake

```
#define FSI_APP_TXCLK_FREQ (50 * 1000 * 1000)
#define FSI_APP_CLK_FREQ (CONFIG_FSI_TX0_CLK)
#define FSI_APP_TX_PRESCALER_VAL (FSI_APP_CLK_FREQ / FSI_APP_TXCLK_FREQ / 2U)
#define FSI_APP_N_LANES (0x1U)
#define FSI_APP_FRAME_DATA_WORD_SIZE (16U)
FSI_executeTxFlushSequence(txBaseAddr, FSI_APP_TX_PRESCALER_VAL);
```

→ FSI TX0 transmitting data (16 Length 16-bit words) from MSRAM to TX buffer, clear TX event

→ CLK and data from TX to RX with jumper

→ Check if RX buffer frame done, write data from RX buffer to MSRAM, clear RX event

Two GPOs are configured accordingly for TX and RX.

For the TX part, GPO0 is set high when the TX buffer is going to set and set low after TX event is cleared and data transfer is done.

For the RX part, GPO1 is set high after the TX event is cleared and set low when data transfer is done.

The test conditions for PRU handler:

→ TX, RX module configuration with handshake

→ FSI TX0 transmitting data (16 Length 16-bit words) from TCM to TX buffer using PRU_XFR2VBUS, clear TX event

→ CLK and data from TX to RX with jumper

→ RX event trigger PRU INTC and data movement from the RX buffer to TCM, clear RX event

Two GPOs are configured accordingly for TX and RX.

For the TX part, GPO0 is set high when the TX buffer is going to set and set low after TX event is cleared and data transfer is done.

For the RX part, GPO1 is set high when receive interrupt flag is set in PRU INTC and set low when data transfer is done.

Test results:

The compare value of IEP timer to trigger transmitting are set with 8 × 7.8 µs = 62.5 µs (16-kHz communication cycle). Figure 2-8 shows the verification waveform.

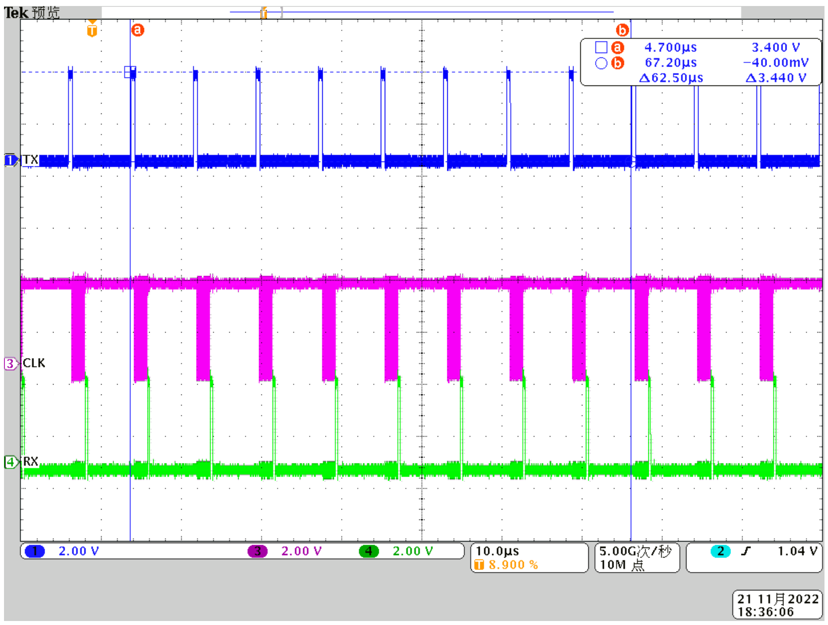


Figure 2-8. Communication Cycle

Table 2-1 shows the comparison results between the R5 core control and PRU handler on FSI communication. From the test results, the PRU handler helps to have 6 times reduction in FSI processing time (RX + TX) from 4.42 µs to 740 ns. And the time obtained for the data transmission is about 1.6 µs. To calculate the transmission speed, the consider the total data length. Figure 1-3 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 Epilogue fields

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.

Therefore, the best transmission time for 16 words can be derived theoretically as follows:

- Effective data bits = 16-bits × 16 (data words) + 8-bits (user data) + 8-bits (CRC) =272 bits
- Total Length bits = Effective data + Overhead = 272 + 24 = 296 bits
- FSITXCLK cycles for total Length = 272/4 + 24/2 = 80 cycles

Thus, with a total 80 FSITXCLK cycles for 16 data words, the transmission time can be calculated as follows:

(FSITXCLK cycles) / (FSITXCLK frequency) = 80 / 50 MHz = 1.6 µs (Conforms to the test results).

So, the transmission speed is 185Mbps (296 / 1.6 µs).

Table 2-1. FSI External Loopback Communication Using R5 Core Control or PRU handler

Controller	FSITXCLK (MHz)	Data Lines	Data Length (Bit)	TX Module Data Processing Time (µs)	Transmission Time (µs)	RX Module Data Processing Time (µs)	Total Time (µs)
R5 Core Control	50	2	16	1.5	1.6	2.92	6.5(1)
PRU Handler	50	2	16	0.46	1.6	0.28	2.396(1)

(1) TX event clear time is included.

Figure 2-9–Figure 2-16 show the test waveforms including total cycle time, transmit latency, data transfer, and receive latency comparison between R5 core control and PRU handler.

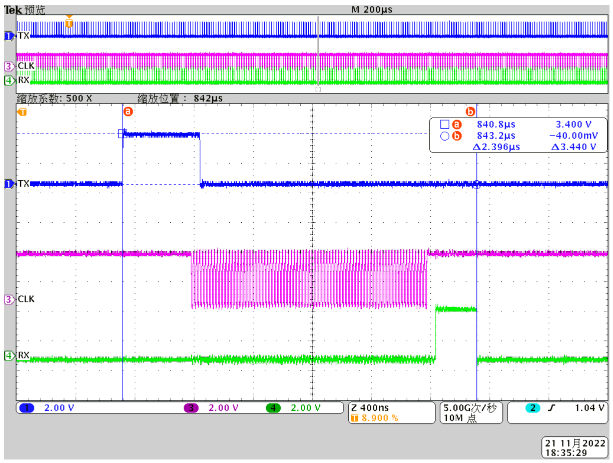


Figure 2-9. PRU handler_total Cycle Time

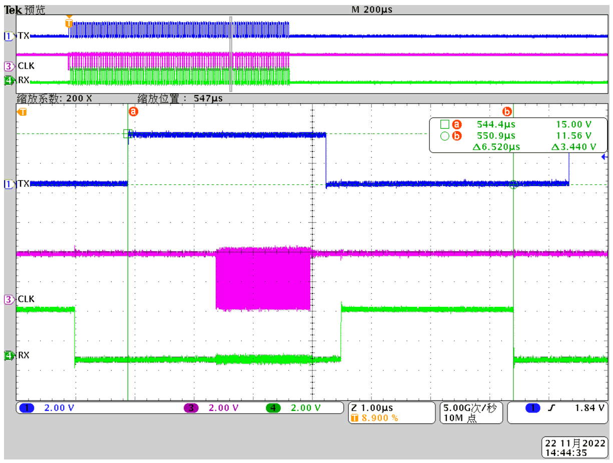


Figure 2-10. R5 Core control_total Cycle Time

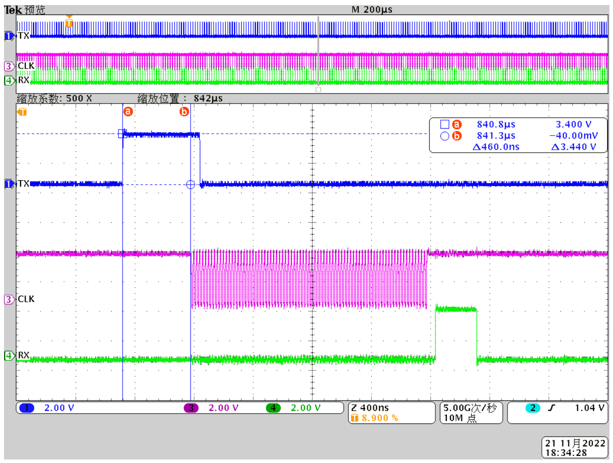


Figure 2-11. PRU handler_Transmit Latency

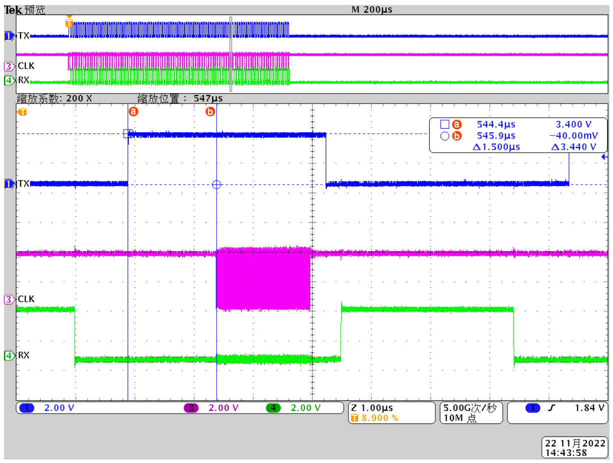


Figure 2-12. R5 Core control_Transmit Latency

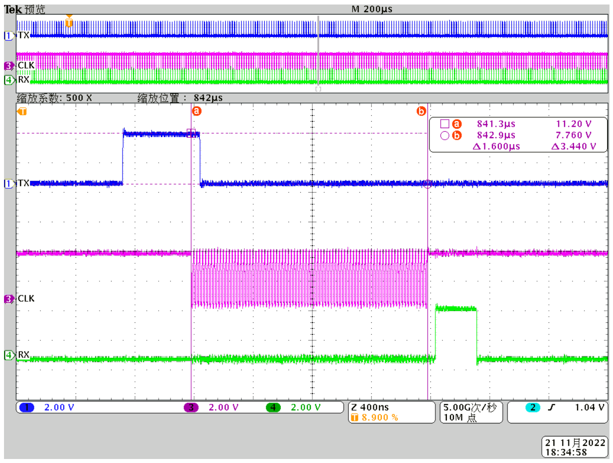


Figure 2-13. PRU handler_Data Transfer

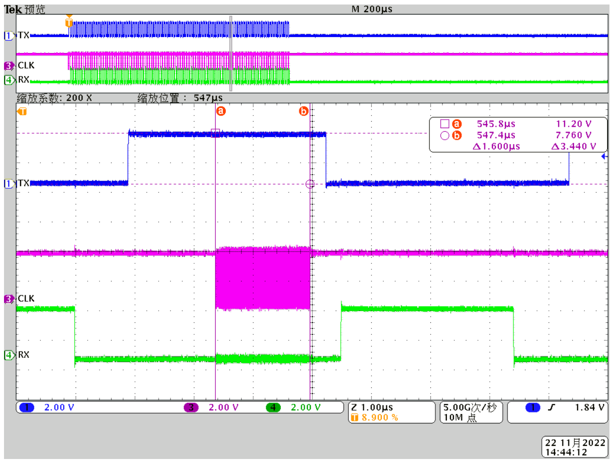


Figure 2-14. R5 Core control_Data Transfer

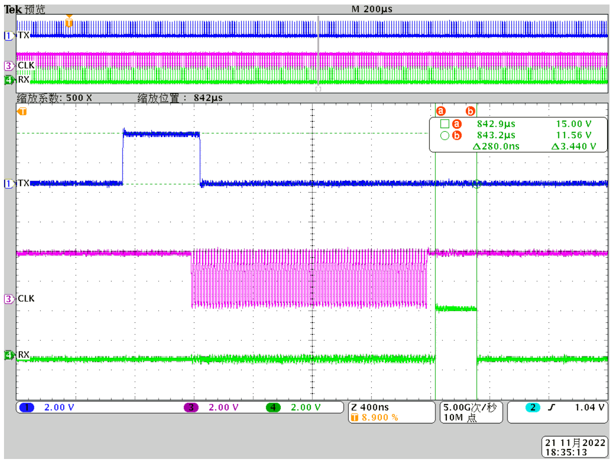


Figure 2-15. PRU handler_Receive Latency

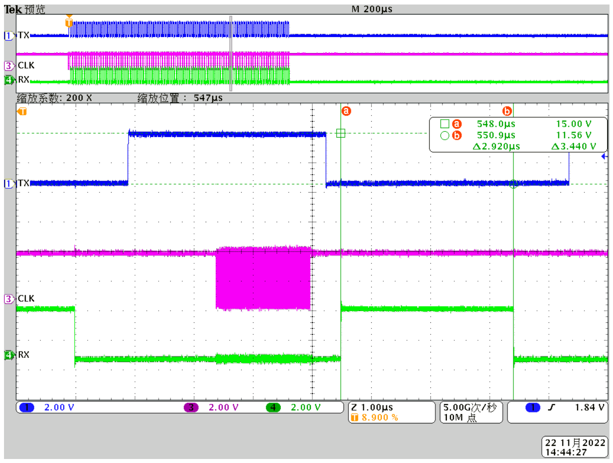


Figure 2-16. R5 Core control_Receive Latency

3 Summary

This application note provided a software-optimized method using PRU to increase FSI communication bandwidth and decrease the latency of data processing time. The verification results proved the effect of this method that the PRU handler helps to have six times reduction in FSI processing time (RX + TX) from 4.42 μs to 740 ns compares to R5 control. This can be useful for multi-axis servo control system.

4 References

1. Texas Instruments, *AM243x Sitara™ Microcontrollers*, data sheet
2. Texas Instruments, *AM64x / AM243x Processors Silicon*, technical reference manual
3. Texas Instruments, *PRU Assembly Instruction User Guide*
4. Texas Instruments, *Using the Fast Serial Interface (FSI) With Multiple Devices in an Application*, application report