Application Note

I²C Mechanisms in IVI FPD-Link IV Devices

Shelly Xie, Violet Lei, Esther Chang

ABSTRACT

This application note describes I²C communication fundamentals and addresses bus lockup issues in FPDLINK serializer-deserializer systems.

Table of Contents

Trademarks

1 I²C Introduction

I²C communication utilizes a controller-target architecture in which only one controller device controls the bus at any moment. The controller device generates the clock signal, initiates communication, and regulates the data transfer rate. Target devices are assigned unique addresses and engage in data exchange with the controller when called. In cases where multiple controllers concurrently request bus control, an arbitration mechanism maintains orderly communication. Figure 1-1 provides a classic example of I²C communication through FPD-Link devices.

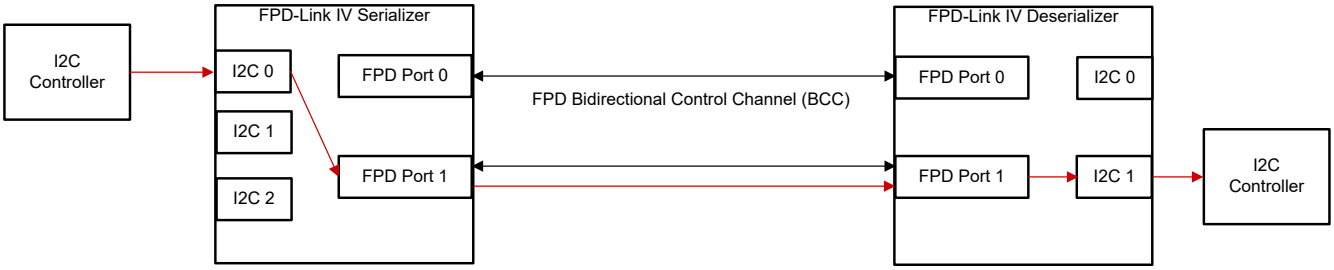

Figure 1-1. An Example of I²C Communication via FPD-Link Devices

2 I²C Communication Between Controller and Target Devices

The I²C serial control bus consists of SCL and SDA lines, where SCL serves as the clock line and SDA as the data line. The serial bus protocol operates through START, repeated START, and STOP conditions. As illustrated in Figure 2-1, a START condition occurs when SDA transitions from high to low while SCL remains high. Conversely, a STOP condition occurs when SDA transitions from low to high while SCL is held high. When the controller needs to initiate a new communication without releasing the bus to an idle state, a repeated START condition may be employed. A repeated START condition resembles a standard START condition and it differs in that it occurs before a STOP condition. Repeated STARTs are often employed in multi-controller systems to allow for multi-read/write transactions without another controller interrupting the transactions.

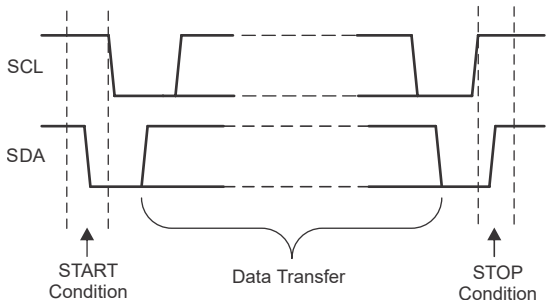

Figure 2-1. Example of Start and Stop Conditions

After the initial START condition, for every byte (8 bits) transferred, the transmitter pulls the SDA line low for a 9th clock cycle, in which the receiver can either respond with an ACK by pulling the SDA line low or NACK by leaving it high.

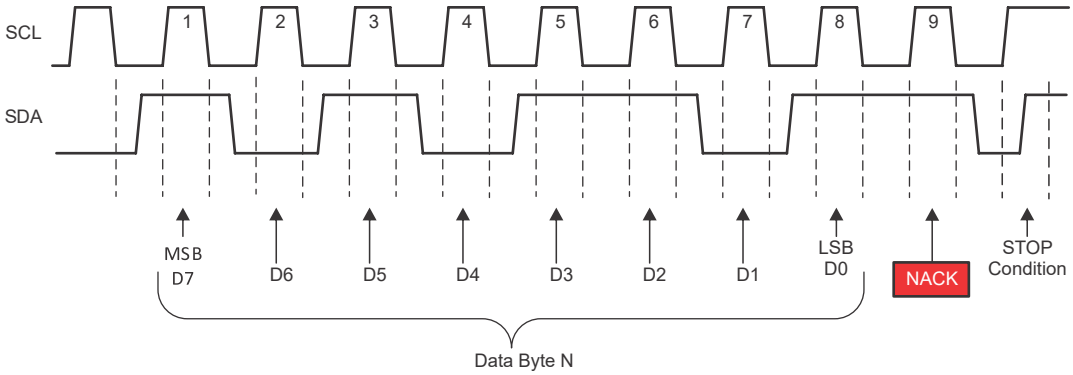

Figure 2-2. Example Data-Byte NACK Waveform

I²C Write

Figure 2-3 shows an I²C write transaction. The controller device first issues a START condition, followed by a 7-bit target address, with the last bit (R/) set to 0 to indicate a write operation. Upon receiving an ACK from the target, the controller transmits the 8-bit register address. After the target receives this data, it responds with an ACK to signal readiness for the command. The controller then sends the specific register data to the target until it is done and concludes the transmission by issuing a STOP condition to indicate completion. Figure 2-3 illustrates the flow of a write transaction.

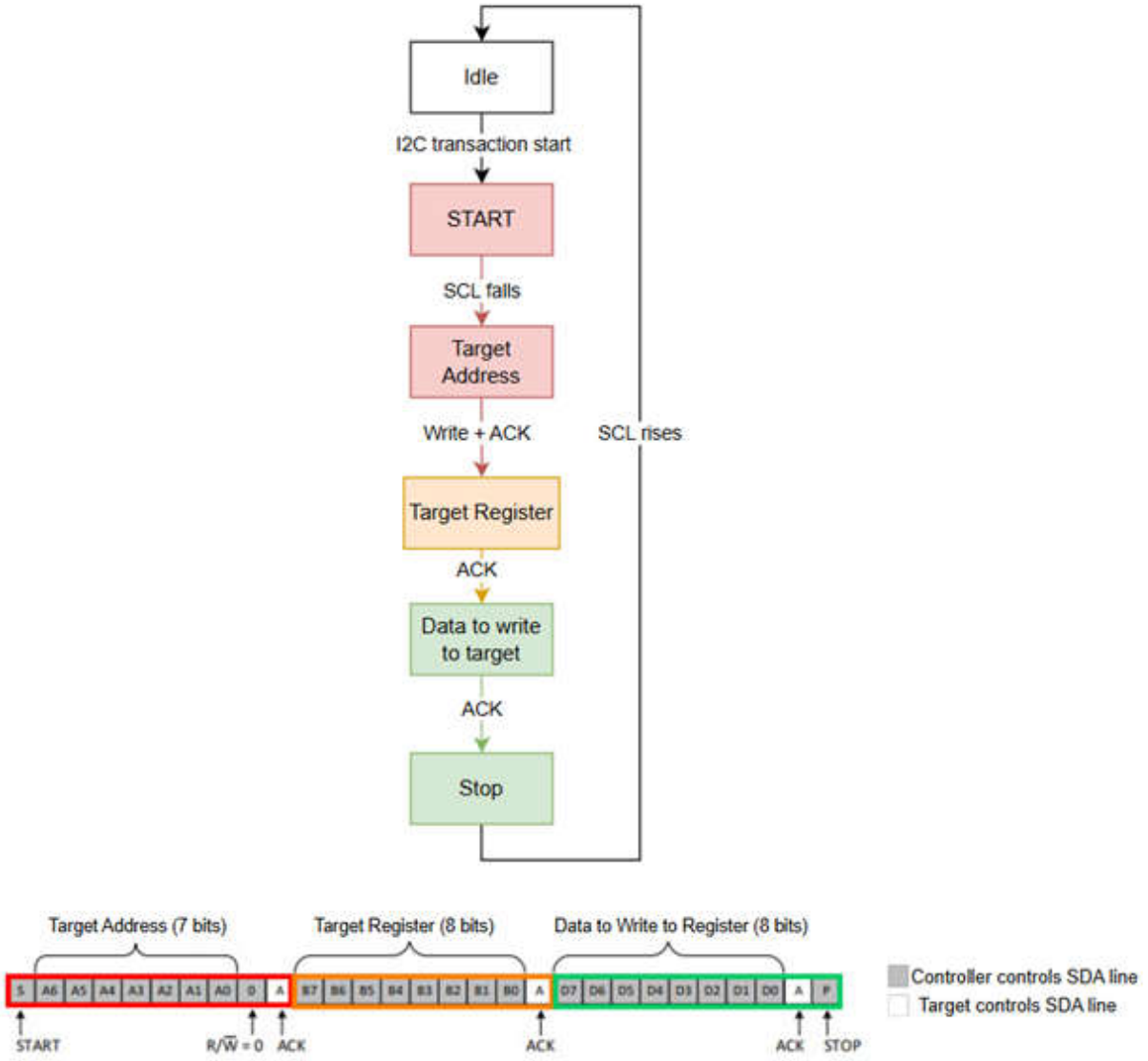

Figure 2-3. I2C Write Transaction

I2C Read

Similar to the write transaction, a read transaction starts when the controller device issues a START condition, followed by a 7-bit target address, with the last bit (R/)[EC1] set to 0 to indicate a write operation. The target issues an ACK, and the controller follows with the register address from which data is to be read.

After the target acknowledges the register address, the controller issues a Repeated START condition and transmits the device address with the R/ bit set to 1, placing the controller in read mode. The target ACKs this read request, whereupon the controller releases the SDA bus while continuing to provide the clock signal to the target and waits for the response.

The controller becomes the receiver, and the target becomes the transmitter. For every data byte the target sends back, the controller sends an ACK to indicate that it is ready for more data. The target continues transmitting data until the controller receives the expected number of bytes, at which point the controller sends a NACK to instruct the target to cease communication and release the bus. The controller then issues a STOP condition, completing the read operation from the target. Figure 2-4 illustrates the flow of a read transaction.

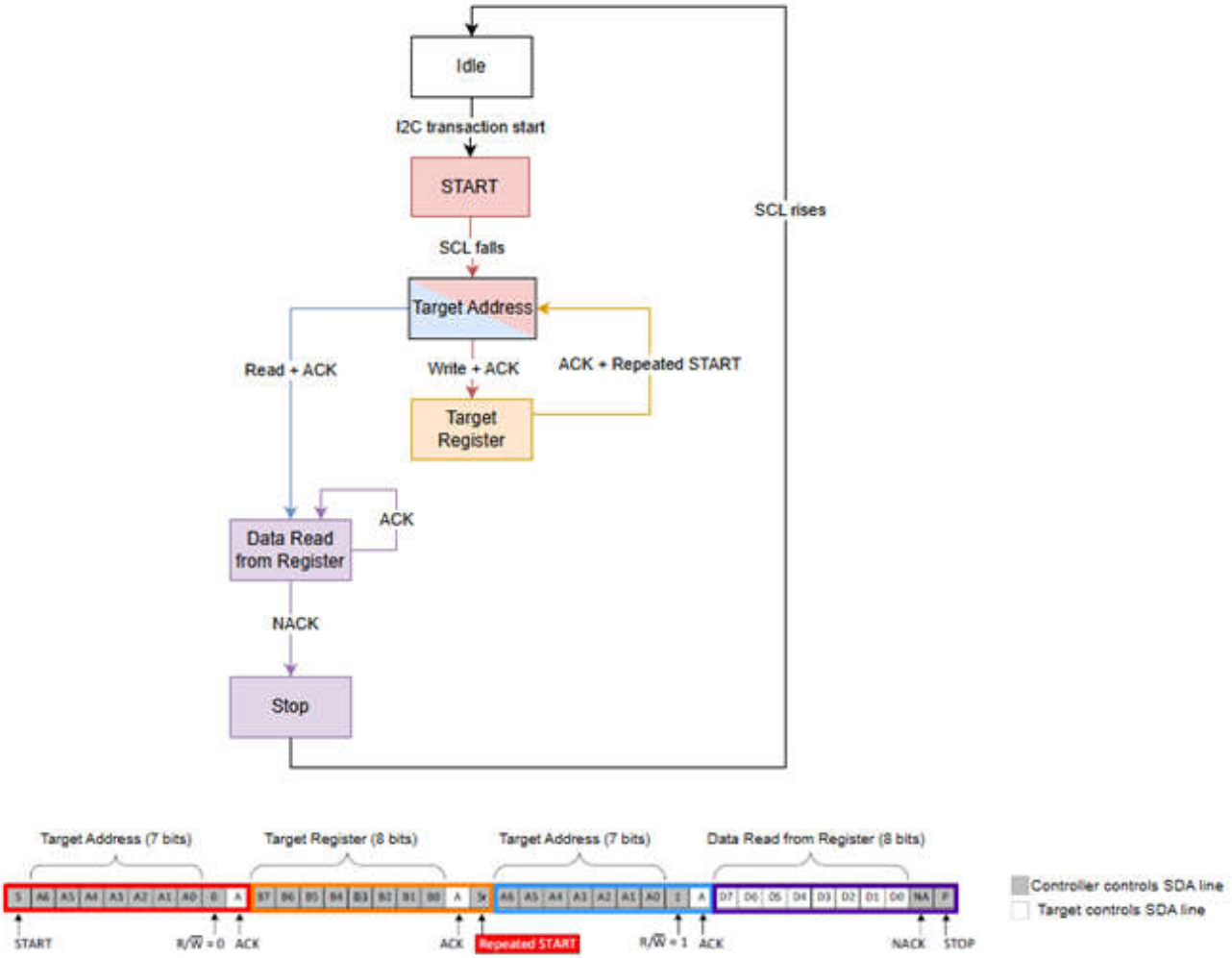

Figure 2-4. I²C Read Transaction

Clock Stretching

In system-level I2C implementation, target devices utilize clock stretching situations where it is not ready to respond to the controller. During all I2C transactions, the controller controls the SCL speed; if a target device needs longer to respond, it *stretches the clock* by holding down the SCL line until it is ready to respond.

3 I2C Mechanisms in FPD-IV Devices

FPD-Link IV devices incorporate three mechanisms to monitor the I²C bus. The following descriptions are based off this system implementation:

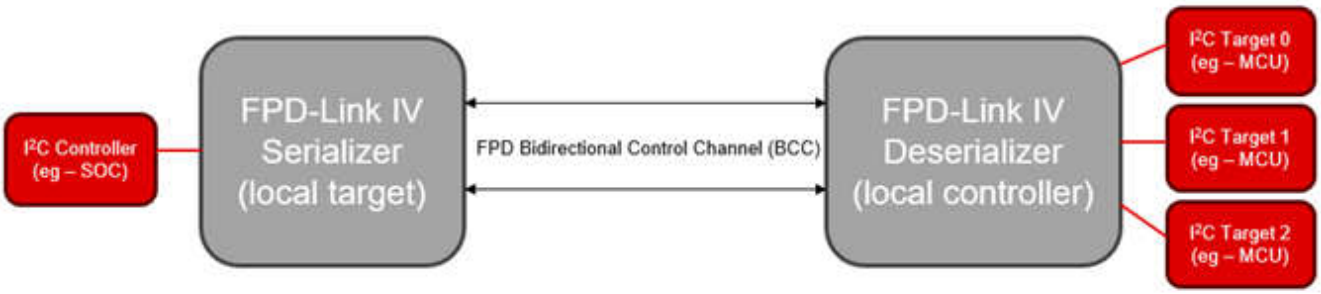

Figure 3-1. Example of I2C Stuck Issue in Clock Stretching

In this system, the FPD-Link IV serializer is denoted as the local target, and the deserializer is denoted as the local controller. The I²C controller sends commands to the serializer *locally*, and sends commands to the deserializer and I²C target *remotely* (through the FPD-Link BCC).

I²C Arbiter

The I²C bus arbiter function prevents one I²C target device from blocking other I²C targets from accessing the FPD-Link bi-directional control channel (BCC) system. The timer starts after the I²C target receives a START command and expires after the timeout period, which is set in an ARB_TIMEOUT register, if the I²C target does not receive a STOP command. When ARB_TIMEOUT occurs, the FPD-Link device internally generates a STOP signal and transmits it to the remote device through the BCC, thereby freeing the arbiter for use by other I²C targets on the local device.

I2C Bus Timeout

The I²C bus timer is devoted to the local bus between either the controller and local target or the local controller and target. This timer initiates when a START condition is initiated by the respective controller and times out based on a I2C_BUS_TIMER_SPEEDUP register setting if there is no toggling of the SDA line when SCL is high (there is no response from the target). Upon timeout, the controller generates a NACK signal internally and transmits it through the local bus to release it.

BCC Watchdog timer

The BCC watchdog timer maintains the release of the BCC if a remote target takes too long to respond during remote transactions. The timer starts after the SER receives a START command from the controller and expires after the timeout period set by BCC_WATCHDOG if the SER does not receive a response from the remote deserializer through the BCC. When timeout occurs, a NACK is sent to the local I²C controller attached to the SER to cancel the transaction and release the BCC.

4 Clock Stretching Example in FPD-Link IV Device

Often, when a MCU (I2C target connected locally to a deserializer) is handling multiple processes, it can clock stretch on the local I^{2}C bus to request a wait state from the SOC.

Figure 4-1 shows an example of this clock stretching. D0 and D1 represent the deserializer – MCU's SDA and SCL buses respectively, while D2 and D3 represent the SOC – serializer's SCL and SDA buses respectively. If the MCU's clock stretching duration exceeds the ARB_TIMEOUT threshold configured in the serializer, the arbiter timeout is triggered. At this point, the serializer internally generates a NACK and sends it to the deserializer to release the deserializer – MCU bus.

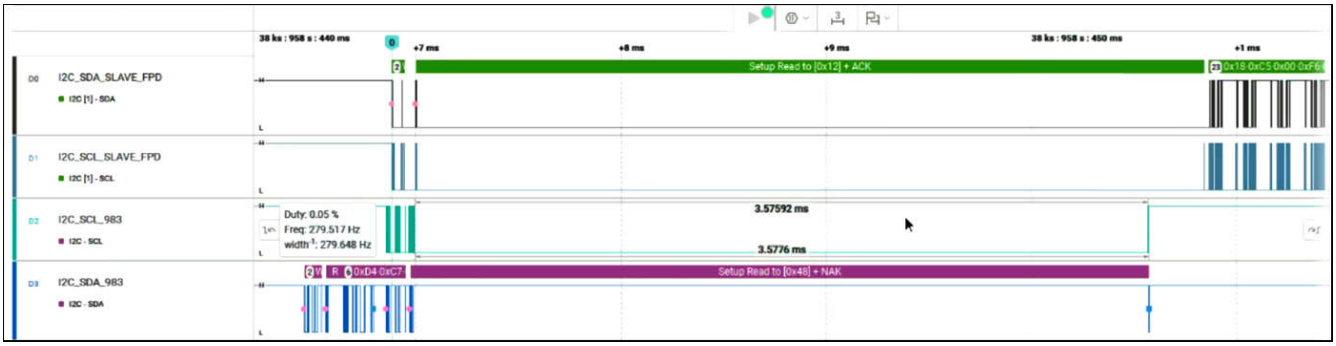

Figure 4-1. Clock Stretching Results in ARB_TIMEOUT Condition

This clock stretching occurs during a read transaction to a target remotely through the FPD-Link. SCL is held low during the target address portion of the read command, and the transaction cannot proceed until it receives an ACK. However, when the serializer-side ARB_TIMEOUT is triggered and sends a NACK, this is not sufficient for the deserializer to proceed onto the next state of the read transaction, causing the deserializer to be unable to exit this read state after the ARB_TIMEOUT is triggered.

If the clock stretching duration is less than the arbiter timeout of both serializer and deserializer, this issue can be prevented. However, if the clock stretching duration is larger than what is programmed in the ARB_TIMEOUT register of the serializer, the serializer-side I^{2}C arbiter must be disabled to prevent premature bus release. In the case that this is a transaction across the FPD-Link, the deserializer also has an arbiter that would trigger, and must be disabled if the MCU is expected to clock stretch more than the appropriated time in the ARB_TIMEOUT register. Therefore, disabling the I^{2}C bus arbiters on both the serializer and deserializer would be appropriate in scenarios where the target is expected to clock stretch longer than the allocated time in the ARB_TIMEOUT function of the serializer and deserializer. This approach introduces no additional risk, as the BCC watchdog timer and I^{2}C bus timeout function remain capable of handling timeout conditions and releasing the I^{2}C bus if necessary.

5 Summary

This application note describes basic I^{2}C mechanisms, specific I^{2}C mechanisms introduced in the FPD-Link IV IVI devices, and a system-level implementation clock stretching example in which the arbiter timer is disabled.

6 References

- Texas Instruments, *DS90HB983W-Q1 8K DP/eDP to FPD-Link IV Serializer with DSC* datasheet
- Texas Instruments, Understanding the I^2C Bus application report

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, regulatory 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 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, TI’s General Quality Guidelines, 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. Unless TI explicitly designates a product as custom or customer-specified, TI products are standard, catalog, general purpose devices.

TI objects to and rejects any additional or different terms you may propose.

Last updated 10/2025