Application Note

How to Debug I2C

Kenneth Montgomery, Duy (Bobby) Nguyen

ABSTRACT

This application note discusses the best practices for debugging a system that uses the I2C to communicate between devices. Suggested methods for dealing with NACKs are described in detail, and recommendations for using an oscilloscope to debug I2C are also provided. Issues potentially caused by I2C switches, buffers, and level translators are also discussed in this document.

Table of Contents

1 Introduction......2
2 General Checks for Dealing With NACKs......2
2.1 NACKs......2
2.2 Check the Schematic......2
2.3 Double Check SDA and SCL Between the Controller and Target......3
2.4 RESET Properly Biased......4
2.5 Device is Soldered Properly......4
3 Scopeshots......5
3.1 Why use Oscilloscopes for Debugging?......5
3.2 Setting up the Oscilloscope......5
3.3 Verify the I2C Address When a NACK is Received......5
3.4 Validate Start and Stop Conditions......6
3.5 Check the Byte Format......7
3.6 Are Rise Times Within I2C Standard?......7
3.7 Are the Sent Command Bytes Valid?......9
4 I2C Switches......10
4.1 Stop Conditions for TI I2C Switches......10
5 I2C Buffers......10
5.1 VoL versus ViLc of the Buffer......10
5.2 VoL of the Buffer Exceeds the ViL of the I2C Target......10
5.3 Static Offset of Buffers Cannot Connect to Other Static Offsets......10
6 Checklists......11
7 Conclusion......12

List of Figures

Figure 2-1. Example of NACK......2
Figure 2-2. TCA9555 With Error in Schematic......3
Figure 2-3. TCA9555 With Error in Schematic Revealed......3
Figure 2-4. Example of SDA and SCL Nets Swapped in Schematic......4
Figure 2-5. Example of Top Marking for Pin 1 on PCB......4
Figure 3-1. Example of a Debugging Scope Shot by Looking at the Data While the Clock Period is a Logic High......5
Figure 3-2. Example of Start Condition......6
Figure 3-3. Example of Stop Condition......6
Figure 3-4. Example of I2C Write Format With Multiple Register Addresses......7
Figure 3-5. Example of Rise Time......7
Figure 3-6. Out-of-Specification Rise Time Example......8
Figure 3-7. Rise Time Within Specification Example......8
Figure 3-8. Example of Invalid Register Receiving a NACK......9
Figure 3-9. Example of Acceptable Command Byte Receiving an ACK......9

List of Tables

Table 6-1. I2C IO Expander Schematic Checklist....................................11
Table 6-2. I2C switch or I2C MUX Schematic Checklist....................................11
Table 6-3. I2C Buffer Schematic Checklist....................................12

Trademarks

E2E™ is a trademark of Texas Instruments.
All trademarks are the property of their respective owners.

1 Introduction

Inter-IC (I2C) is a popular serial communication protocol that allows for multiple controller devices to communicate with multiple target devices. I2C consists of a bidirectional two-wire bus, where one line serves as a serial data line (SDA) and the other serves as a serial clock line (SCL). Both lines of the two-wire bus are generally connected to an open drain or open collector driver with an input buffer that supports bidirectional data transfer. When working with an open drain or open collector system, being aware of all the potential issues that can cause communication failures is important. The purpose of this article is to provide a comprehensive guide for recognizing and debugging these issues quickly and effectively.

2 General Checks for Dealing With NACKs

2.1 NACKs

In I2C communication, every transaction consists of 8 bits (1 byte) of information from the controller followed by one bit from the target device. The bit sent by the target device can either be a *0* (usually denoted as an ACK bit), or a *1* (usually denoted as a NACK bit). When an ACK bit is sent by the target, this indicates that the data transmission was correctly received without any errors. When a NACK bit is sent by the target, this indicates that the receiver did not correctly receive the data or address transmission. There are several general conditions that can lead to the generation of a NACK in an I2C system. These conditions are detailed in *ACK and NACK* section of the *Understanding the I2C Bus* application note.

Figure 2-1 is an example which showcases a NACK occurring when the I2C controller tries to write to the target device at address 0x55h and the I2C target NACKs (does not drive SDA low on the 9th clock pulse).

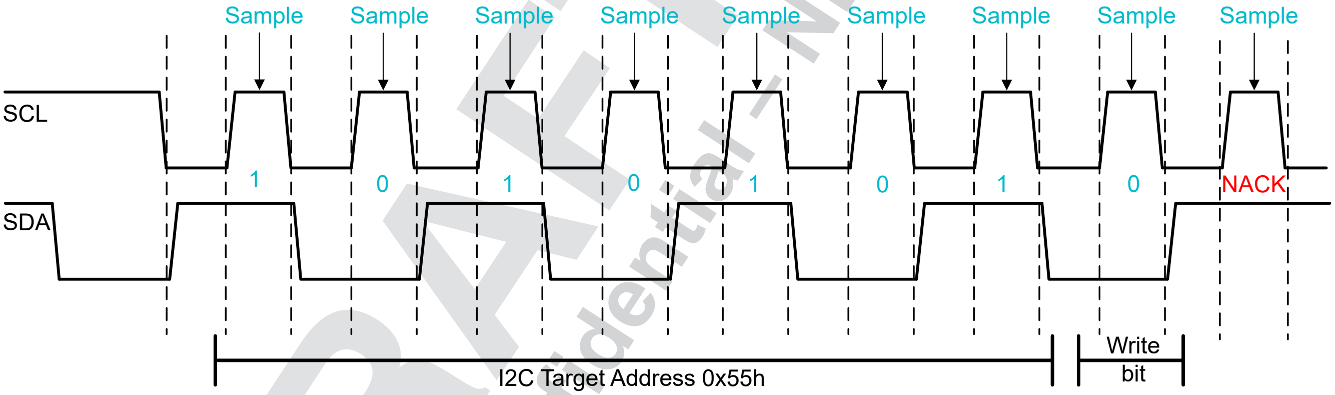


Figure 2-1. Example of NACK

2.2 Check the Schematic

When debugging an I2C device, always check and see if the data sheet pinout matches the schematic pinout. Sometimes the pins on a schematic are arranged differently from how the pins are presented on a device data sheet. If this check was not performed, there is a chance that the device can be incorrectly connected on the circuit. Leaving a device incorrectly connected can cause the device to become damaged whenever the circuit is energized. Therefore, to avoid damaging the device unintentionally, always check to see if there are any differences between the pinout of the data sheet and the pinout of the schematic.

Figure 2-2 shows an example of TCA9555 in which a schematic error can cause a NACK to occur.

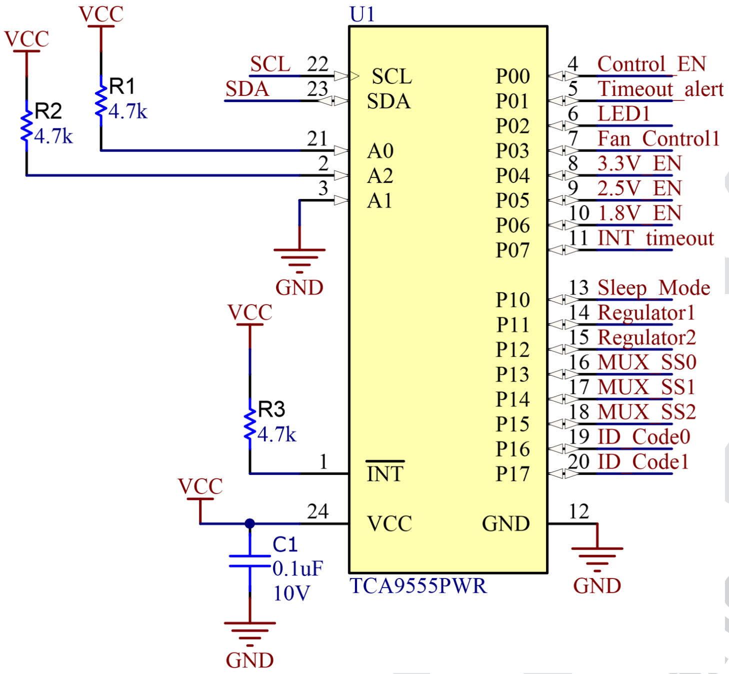


Figure 2-2. TCA9555 With Error in Schematic

The error illustrated in Figure 2-2 involves pin 2 and pin 3 being swapped (A2 and A1) which indicates the I2C target address is 0x25h. Figure 2-3 reveals the schematic error side by side with the TCA9555 data sheet pinout. With the A1 and A2 swap, the schematic address indicates 0x25h while the correct I2C target address based on the data sheet pinout is 0x23h. Anyone viewing the schematic incorrectly tries to communicate with the I2C target at 0x25h and always receive a NACK.

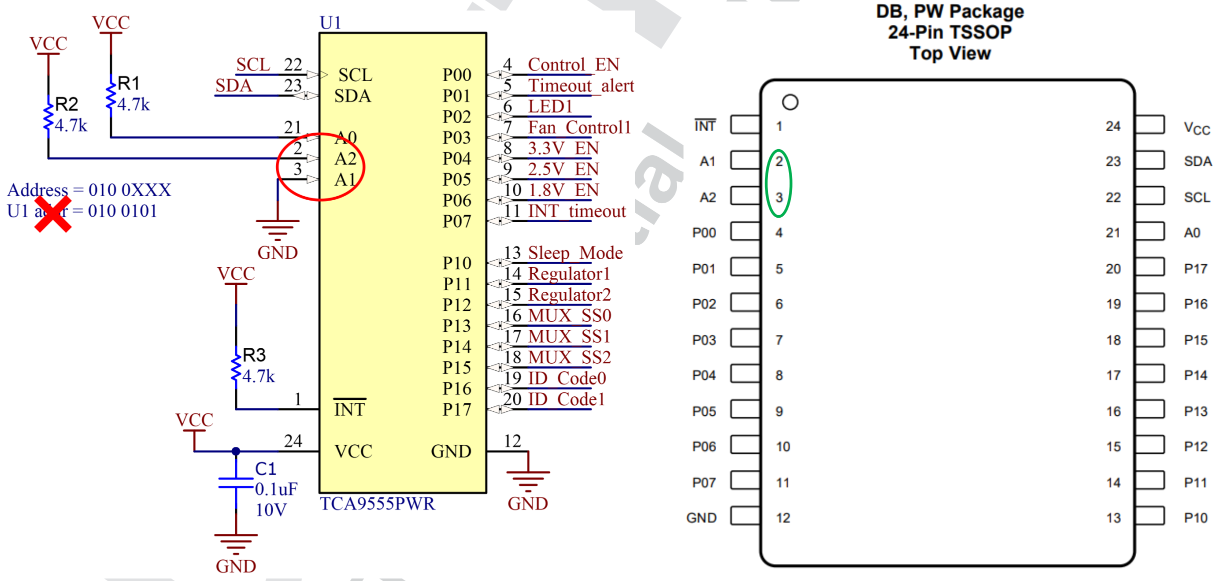


Figure 2-3. TCA9555 With Error in Schematic Revealed

2.3 Double Check SDA and SCL Between the Controller and Target

If NACKs are being received from an I2C device the design is communicating with, check to make sure that the SDA and SCL lines are properly connected between the I2C controller and target. Sometimes users accidentally swap the SDA and SCL connections between a controller and target device. If this happens, the target device always sends NACKs back to the controller, even if the correct bits are being transmitted. To prevent this from happening, always verify that the SDA and SCL connections between the controller and the target are correct. Figure 2-4 shows an example where the SDA and SCL nets on the schematic are swapped with the schematic pinout SDA and SCL resulting in the I2C target device always NACKing the address.

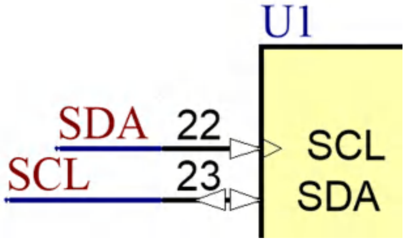


Figure 2-4. Example of SDA and SCL Nets Swapped in Schematic

2.4 RESET Properly Biased

In an I2C circuit, every component in the system is exposed to some level of electrical noise. Sometimes electrical noise can become large enough to unintentionally influence the voltage levels on floating pins of I2C devices. This high noise level can have a negative impact on the operation of a device, especially if the pin that is left floating is the device reset pin. If the reset pin of an I2C device is left floating, electrical noise can cause the device to unintentionally enter a reset state. To avoid unintentionally entering a reset state, bias the reset pin of all I2C target devices to a defined logic level with either a pullup or pulldown resistor. Biasing the reset pin keeps the reset pin in a defined state until the pin is intentionally changed by a controller. If high level of noise is expected in the circuit, a capacitor can be added onto the reset pin to further help reduce the effect of noise. Generally, 1-µF and 0.1-µF capacitors are both used for this type of application.

If the reset pin of an I2C target device is active low, use a pullup resistor to bias the reset pin to V_{CC}. In this configuration, the device is only reset if the device is intentionally driven low by a controller. If the reset pin of an I2C target device is active high, use a pulldown resistor to bias the target device to GND. In this configuration, the device is only reset if the device is intentionally driven high by a controller.

2.5 Device is Soldered Properly

Before soldering an I2C device to a board, always verify that the device is placed in the correct orientation. An I2C device continuously sends NACKs back to the controller if the device is incorrectly soldered onto a PCB. Soldering a device in an incorrect orientation can also damage the device whenever power is applied to the circuit. When verifying if a device has been soldered onto a PCB properly, look for two things: a package marking, and a footprint marking. These markings on the package and footprint are created to help the user match the pin locations of the package to the pin locations of the footprint.

Generally, a package marking is placed directly over pin 1 of the package (see Figure 2-5). The shape of the marking can vary from a small circle to a diagonal line. An easy way to find out the shape of this marking is to look at the pinout diagram on the data sheet of the device.

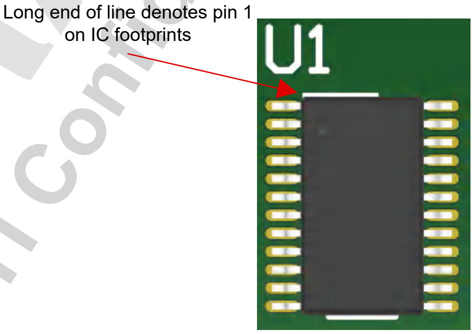


Figure 2-5. Example of Top Marking for Pin 1 on PCB

Figure 2-5 illustrates the small line that is usually located near pin 1 of the footprint. Use both package marking and footprint marking to match the location of pin 1 on the footprint to pin 1 on the device. If both the package marking and the footprint marking are located on top of each other, it is likely that the device was placed onto the PCB in the correct orientation.

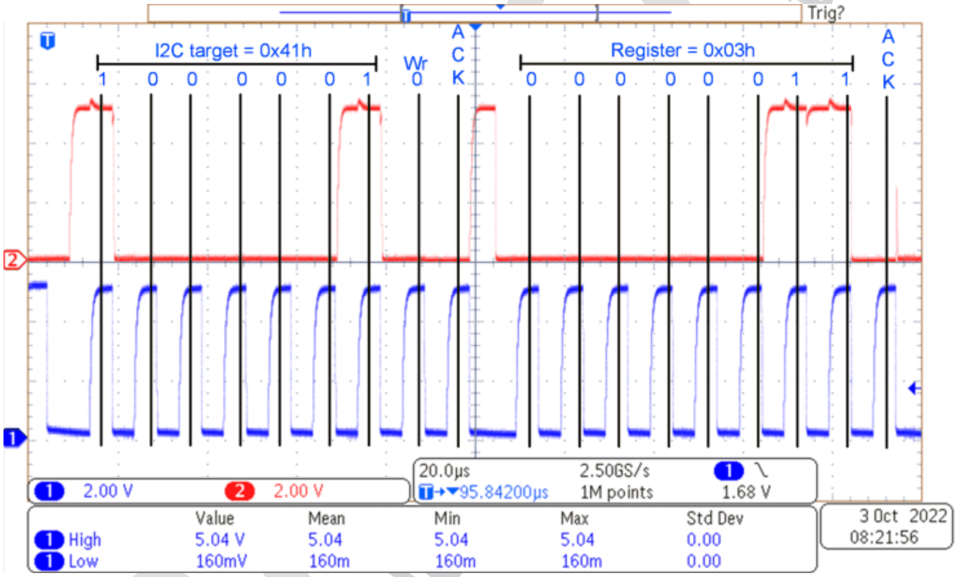
3 Scopeshots

3.1 Why use Oscilloscopes for Debugging?

When debugging an I2C bus, an oscilloscope is better to use over a logic analyzer. This is because logic analyzers hide signal details that are critical to the I2C debugging process. Logic analyzers are designed to only display measured data as being either a logic high or a logic low. This is a feature that makes debugging I2C very difficult because I2C signals can hide certain properties of the signal in an I2C frame. Logic analyzers do not capture signal rise times, overshoot voltages, undershoot voltages, and voltage output low (V_{oL}) values. All of these properties provide information that is necessary to debug an I2C system. Conversely, an oscilloscope, provides the ability to view the rise times, overshoots, undershoots, and V_{oL} levels making debugging easier.

3.2 Setting up the Oscilloscope

Use an oscilloscope to view entire frames of transferred data when debugging an I2C bus. Adjust the y-axis of an oscilloscope window such that the entire voltage range of the measured SDA and SCL signals are both viewable within the same scope window. Set up the x-axis of the oscilloscope window so that the following is clearly visible: the start condition, stop condition, and all address and data bits of a single frame. If possible, place the measured SDA and SCL signals directly on top of each other with a voltage offset in the oscilloscope window (doing this makes verifying the individual data bits easier to match to their corresponding clock pulses). Figure 3-1 shows an example of how an oscilloscope window screenshot looks.



	Value	Mean	Min	Max	Std Dev
1 High	5.04 V	5.04	5.04	5.04	0.00
1 Low	160mV	160m	160m	160m	0.00

Figure 3-1. Example of a Debugging Scope Shot by Looking at the Data While the Clock Period is a Logic High

3.3 Verify the I2C Address When a NACK is Received

If NACKs are received from a device that the design is attempting to communicate with, sending the wrong address can cause these errors.

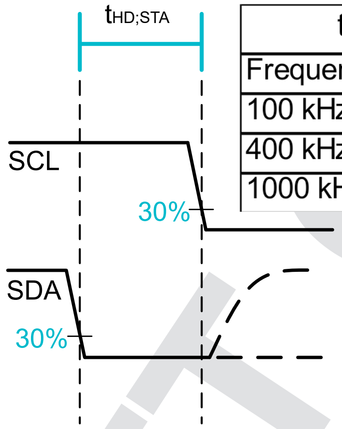
Communicating with a device via I2C requires sending out the address of the specific device being communicated with after a start condition is initiated. If NACKs are received from a device that the design is attempting to communicate with, these NACKs can be a result of the wrong address being sent to the device. To verify that the correct address is being sent, use the oscilloscope window to view the individual bits being sent to the device after a start condition is initiated. Use clock pulses on the SCL line to mark the individual bits of a single frame (remember, there should be 8 clock pulses between each ACK or NACK bit). NACKs are created when the wrong target address is sent to the device that you are trying to communicate with, which is why verifying that the controller is sending out the correct address is important.

Once the oscilloscope is used to determine the target address that is being sent out, next check the data sheet of the I2C device for the target address value. For I2C communications to be successful, the target address that is being sent out must match the address in the data sheet of the target. If a target is sending back NACKs for every I2C frame that is sent, these NACKs can be because the sent target address does not match the actual address of the target (which is always specified in the data sheet of the target device).

Some I2C devices can also have a hardware-addressable target address that allows for the target address to be changed by the user. If this is the case, check these hardware addressable target bits and make sure that these bits are correctly biased to either a logic high or a logic low value. If these hardware-addressable target bits are left floating, the target address can easily be changed by electrical noise in the system. If the target address is changed to a value that is different than the address being sent by the controller, the target sends back a NACK.

3.4 Validate Start and Stop Conditions

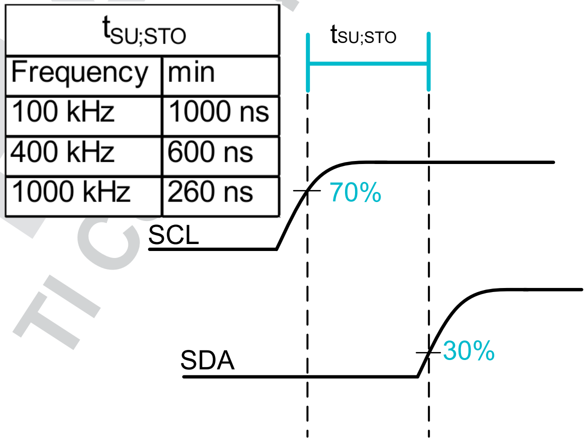
Before any address or data bits are sent over an I2C bus, a start condition must be sent. A high-to-low transition on the SDA line while SCL is high defines a start condition (SCL must go low after this transition occurs for the frame to officially begin). Use an oscilloscope to verify that a start condition is properly initiated before any data or address bits are sent on the I2C bus. Figure 3-2 shows what a start condition looks like and the minimum hold time required according to the frequency of operation.



$t_{HD;STA}$	
Frequency	min
100 kHz	1000 ns
400 kHz	600 ns
1000 kHz	260 ns

Figure 3-2. Example of Start Condition

After the address and data bits are sent, a stop condition is initiated so the controller can let the bus go idle (assuming there is not another controller on the bus, in which case a restart condition can be valid). A low-to-high transition on the SDA line while SCL is already high defines a stop condition. Use an oscilloscope to verify that a stop condition is properly initiated once the controller is ready to let the I2C bus go idle. Figure 3-3 shows a stop condition and the minimum setup time required according to the frequency of operation.

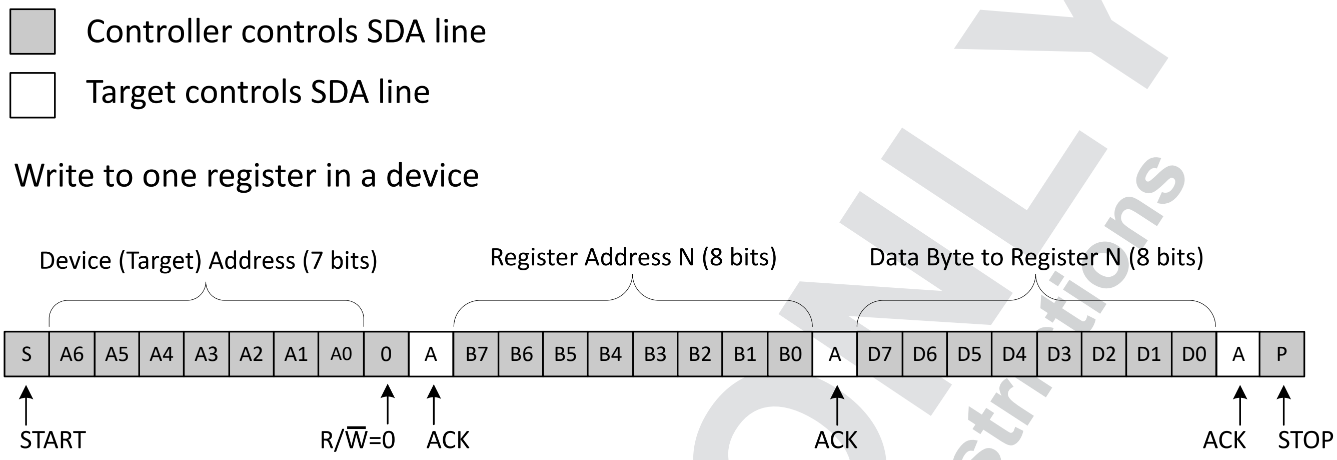


$t_{SU;STO}$	
Frequency	min
100 kHz	1000 ns
400 kHz	600 ns
1000 kHz	260 ns

Figure 3-3. Example of Stop Condition

3.5 Check the Byte Format

Between the first start and the final stop condition, the controller sends bytes of data to the target. The bytes of data can be sent to the target in the following order: target byte address, command byte, then data byte (assuming the target device has multiple registers). Data sent to the target in a different order can cause NACKs to occur. To prevent this, use an oscilloscope to verify that data is being sent to the target device in the correct order (target byte address, command byte, data byte). Remember that clock pulses on the SCL signal line are used to segment off the individual bits being sent on the SDA line. Every frame needs to have 8 clock pulses for the data bits and 1 clock pulse for the ACK or NACK bit.

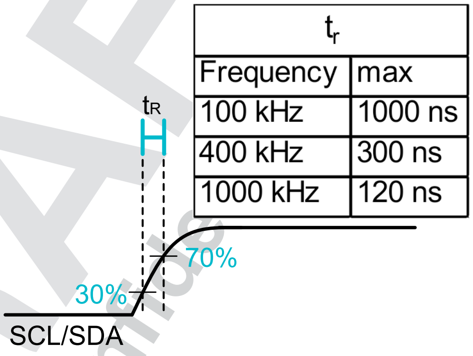


Controller controls SDA line
Target controls SDA line
Write to one register in a device
Device (Target) Address (7 bits)
Register Address N (8 bits)
Data Byte to Register N (8 bits)
S
A6
A5
A4
A3
A2
A1
A0
0
A
B7
B6
B5
B4
B3
B2
B1
B0
A
D7
D6
D5
D4
D3
D2
D1
D0
A
P
START
R/W̅=0
ACK
ACK
ACK
STOP

Figure 3-4. Example of I2C Write Format With Multiple Register Addresses

3.6 Are Rise Times Within I2C Standard?

Rise time in I2C is defined as the time taken for the I2C signal to transition from a logic low to a logic high. The minimum value for a logic high in the I2C standard is defined as 70% of V_{CC}; the maximum value for a logic low in the I2C standard is defined as 30% of V_{CC}. Therefore, measure rise time as the time for the I2C signal to transition from 30% of V_{CC} to 70% of V_{CC}. Figure 3-5 shows an example of SDA, SCL during the rise transition. Use an oscilloscope to measure rise times for both the SDA and SCL lines on the I2C bus.



t_r	
Frequency	max
100 kHz	1000 ns
400 kHz	300 ns
1000 kHz	120 ns

Figure 3-5. Example of Rise Time

When debugging an I2C bus, verifying that the rise time does not exceed the maximum rise time requirement for the frequency that the I2C bus is operating at (see the table in Figure 3-5) is important. Rise times over the limits outlined by the I2C standards can cause bits of data to be unintentionally removed from a data transfer frame (for each ACK bit from the target, there should be 8 bits sent from the controller). Bus capacitance and pullup resistance are both factors that can affect rise times, so check these two parameters whenever you run into issues with your rise times being over the allowed limits (see the *I2C Bus Pullup Resistor Calculation* application note for more information on how pullup resistance and capacitance can be used to calculate rise times).

Figure 3-6 shows an example where the rise time exceeds the maximum specified by the I2C standard.

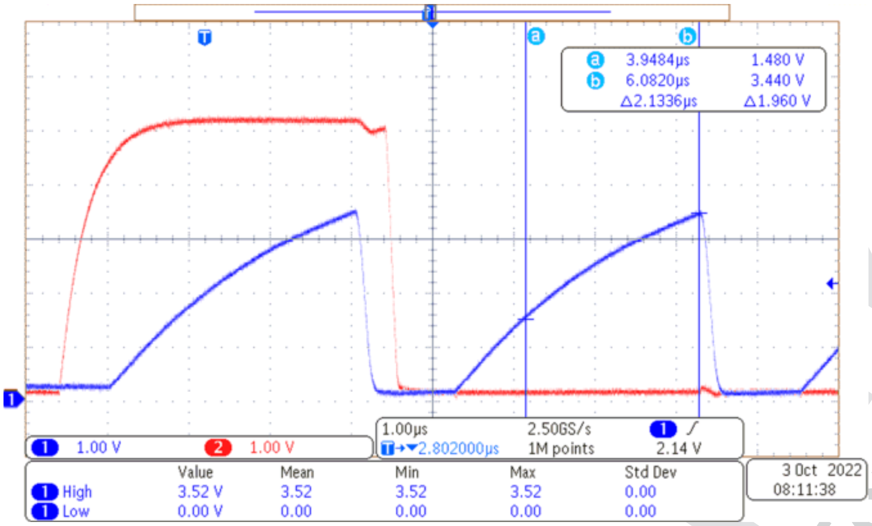


Figure 3-6. Out-of-Specification Rise Time Example

Figure 3-7 shows an example where the rise time is compliant with the I2C standard.

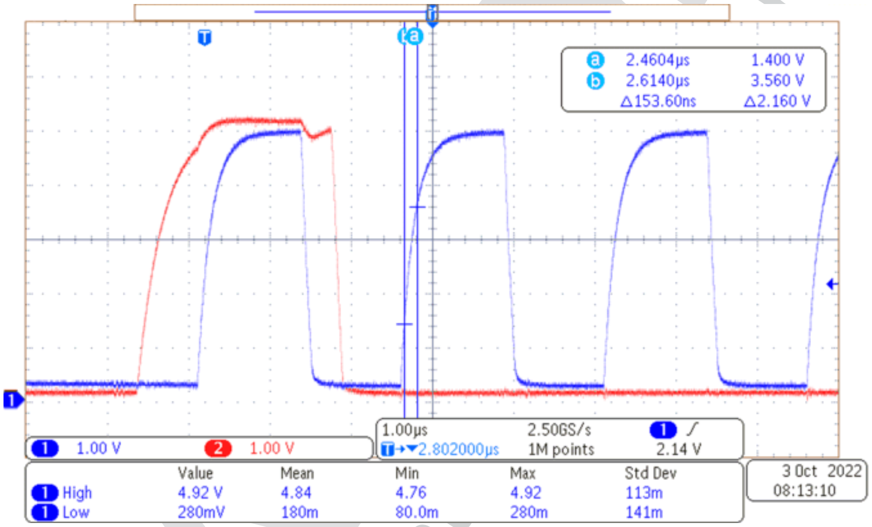
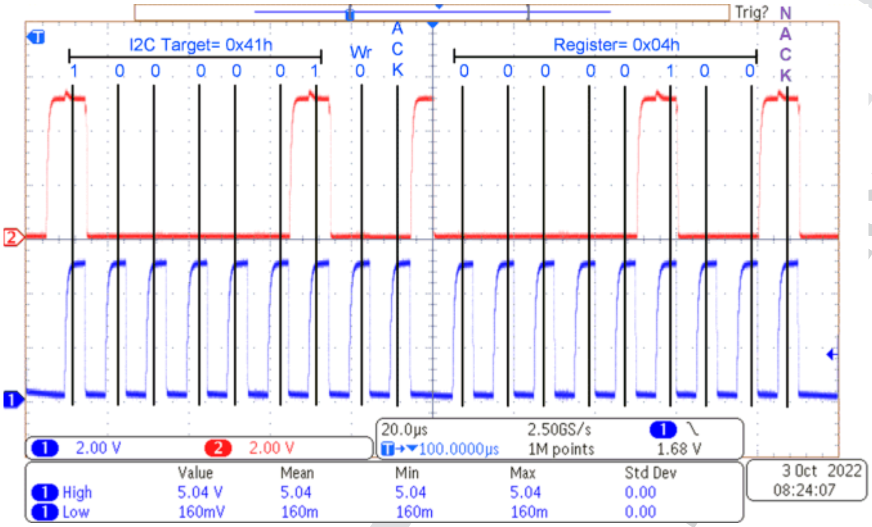


Figure 3-7. Rise Time Within Specification Example

3.7 Are the Sent Command Bytes Valid?

If a command byte is not defined for a particular target device, a NACK is sent to the controller. To make sure that the correct command byte for a device is sent, use an oscilloscope to check if all of the 8 bits sent to the target match the bits of the command defined in the data sheet of the device. Remember that the command byte needs to be sent after the address byte and before the data byte in a frame.

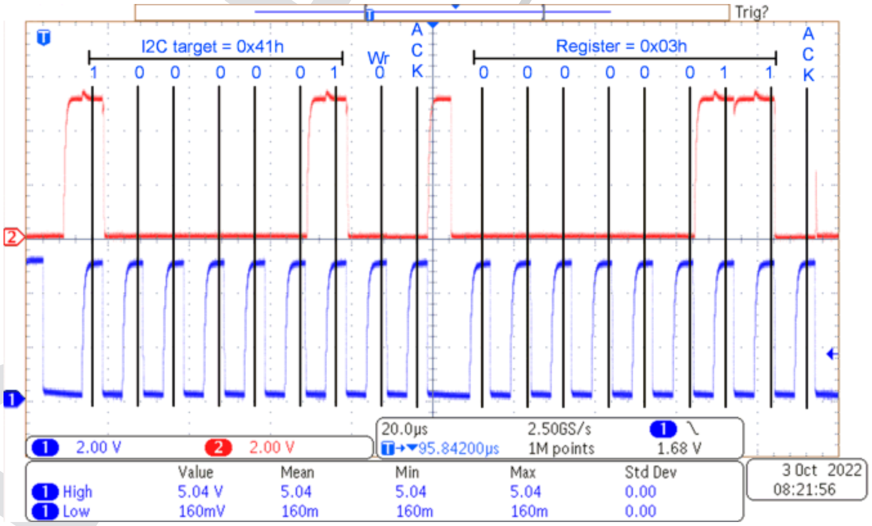
Figure 3-9 shows an example where the I2C target address and write bit is sent followed by a bad (outside of the available registers of the I2C target) command byte resulting in a NACK. The target device in this example only has four registers (0x00h to 0x03h). If a command byte with the register value of 0x04h is sent, the result expected is a NACK, since 0x04h is outside of the acceptable bytes of 0x00h to 0x03h.



	Value	Mean	Min	Max	Std Dev
1 High	5.04 V	5.04	5.04	5.04	0.00
1 Low	160mV	160m	160m	160m	0.00

Figure 3-8. Example of Invalid Register Receiving a NACK

Figure 3-9 shows an example using the same I2C target address and write bit from Figure 3-8 but a known good command byte (a command byte recognized by the I2C target) is sent resulting in an ACK. The target device in this example accepts the 0x03h register byte because the target device has a register between values of 0x00h and 0x03h, all of which are registers that can be written to.



	Value	Mean	Min	Max	Std Dev
1 High	5.04 V	5.04	5.04	5.04	0.00
1 Low	160mV	160m	160m	160m	0.00

Figure 3-9. Example of Acceptable Command Byte Receiving an ACK

4 I2C Switches

4.1 Stop Conditions for TI I2C Switches

Texas Instruments I2C switch devices require a stop condition to properly read in information from I2C writes. Stop conditions must be sent at the end of an I2C data transfer frame for the information to be executed by the receiving device (a repeated start condition does not work). In the case where the user does a write to an I2C switch and implements a restart condition followed by an I2C read transaction, the I2C switch data read shows the same data written from the last write transaction; however, the device does not actually enable the channels that were intended. In short, if a repeated start is used to read data from a TI I2C switch, the data read looks correct but is not actually implemented. See this E2E™ forum thread for more details about this special feature.

5 I2C Buffers

5.1 VoL versus ViLc of the Buffer

Many I2C buffers use an internal static voltage offset to determine the side of an I2C bus that is generating a low signal. While this feature is very effective at avoiding a bus lock condition, the feature can also create additional problems for the user if the buffer is connected incorrectly. If a buffer is being connected to a target device at the location of the static voltage offset (this is commonly known as the side of the device), the VoL value at that location must be lower than the ViLc value of the device. ViLc is an internal voltage value (generally specified in the device data sheet) that is used by the internal logic circuit of the buffer to propagate a signal through the device. For the internal logic of the buffer to correctly propagate a low signal from the buffer side, VoL of the target must be lower than the provided ViLc value of the buffer. If the value is exceeded, the buffer enters a state where the output is constantly alternating between a low and a high value. To avoid this error from taking place, always verify that the VoL value of the device connected to the buffered side is lower than the ViLc value provided by the data sheet of the buffer.

5.2 VoL of the Buffer Exceeds the ViL of the I2C Target

VoL on the buffer side cannot exceed the ViL level of the connected devices. If VoL exceeds ViL, the device is unable to recognize a logic low when the I2C bus is pulled low. To prevent this from happening, always verify that VoL is less than ViL by checking the data sheets of the buffers and connected devices.

5.3 Static Offset of Buffers Cannot Connect to Other Static Offsets

When two buffers are connected together, the buffers never be placed in a configuration such that their static voltage offsets are connected together. If connected together, the static voltage offsets of each buffer influences the internal logic structures of both devices, and prevents a low from propagating to the controller when the target device pulls low. To avoid this issue, never connect two buffers together by their buffered sides. See the *Why, When, and How to use I2C Buffers* application note for an overview of the different acceptable connection configurations of buffers.

6 Checklists

Table 6-1 through Table 6-3 show schematic checklists for the I2C IO expander, I2C switch, I2C MUX, and I2C buffer.

Table 6-1. I2C IO Expander Schematic Checklist

Check	Comments (if Needed)
Local decoupling capacitors	Generally, a 0.1-µF capacitor is placed on V_{CC}, as close to device as possible.
Verify the schematic pinout matches the data sheet pinout.	
Verify that the SDA, and SCL net names match SDA and SCL pinout.	
Check that the pullup resistors are present on the SDA and SCL net within the schematic.	
Unused GPIO pins are biased to either V_{CC} or GND via resistor.	Most of TI's IO expander portfolio do not include internal pullup resistors on the p-port pins; the exceptions are PCF8575, PCF8574, PCF8574A, TCA9555|PCA9555, PCA9554|TCA9554, and TCA9554A|PCA9554A and therefore can be left floating. An alternative approach is after powering up the device, any unused p-port pins can be set as an output (does not matter if set high or low).
Device address is unique on the bus unless using an I2C switch or I2C MUX to resolve conflicts.	
If the device has a #RESET pin, bias the pin high (preferably with a pullup resistor) after powering up.	
If the device has a #INT pin and the #INT pin is used, tie this pin to a pullup resistor.	If unused, the #INT pin can be left floating since this pin is an open drain output.

Table 6-2. I2C switch or I2C MUX Schematic Checklist

Check	Comments (if Needed)
Local decoupling capacitors	Generally a 0.1-µF capacitor is placed on V_{CC} as close to device as possible.
Verify that the schematic pinout matches the data sheet pinout.	
Check that the SDA and SCL net names match SDA and SCL pinout.	
Check that the pullup resistors are present on the SDA and SCL net within the schematic.	
Device address is unique on the bus unless the device sits downstream of an I2C switch or I2C MUX to resolve conflicts.	
If the device has a #RESET pin, bias the pin high (preferably with a pullup resistor) after powering up.	
If the device has an #INT input pin, the pin needs to be biased (usually with a resistor to VCC or GND).	Even if unused, bias this pin to GND or V_{CC}, preferably with a resistor.
If the device has an #INT output pin and the pin is used, tie the pin to a pullup resistor.	If unused, the #INT pin can be left floating since the pin is an open drain output.

Table 6-3. I2C Buffer Schematic Checklist

Check	Comments (if Needed)
Local decoupling capacitors	Generally a 0.1-µF capacitor is placed on V_{CC} as close to device as possible.
Verify that the schematic pinout matches the data sheet pinout.	
Check that the SDA and SCL net names match SDA and SCL pinout.	
Verify that pullup resistors are present on the SDA and SCL net within the schematic.	
Check that V_{CC} rules are followed.	
Verify the static voltage offset is not connected to any other I2C buffer on the offset side.	For TCA9517, TCA9517A, TCA9617A, and TCA9617B, the offset is on the B side. P82B96 has the offset on the Sx or Sy side. TCA9509 has the offset on the A side. TCA980x has a current offset on the B side and cannot connect to any offset on the B side either. TCA9515A and TCA9515B have their offset on both sides and cannot connect to any static voltage offset device on either side.
If device has an enable pin, make sure the pin is biased properly after powering up.	Some devices include an internal pullup resistor. An external pullup resistor is recommended when external noise can couple onto the enable pin from a noisy environment.

7 Conclusion

To effectively debug I2C systems and devices, a sound understanding of the different ways communication errors can occur is necessary. Knowing all the possible ways that NACKs occur can help expedite the process of identifying and correcting errors in a system. Using correct tools and analysis techniques is also an essential component of the debugging process. Communication errors are easier to recognize if the correct tools and analysis techniques are used when capturing and inspecting I2C data transfer frames. With sound background knowledge and correct analysis techniques, errors in I2C systems and devices can be efficiently debugged in a manner that is both quick and effective. For additional I2C debugging assistance, see TI's E2E™ forums.

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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 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 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.

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

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

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATA SHEETS), 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 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 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.

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

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