# Tackling I²C Development Complexities Using Innovative Tools

## Expediting the Development of I²C Bus Enabled Products for Bus Compliance and Interoperability

## Introduction

The I²C bus has gained very broad acceptance in circuit applications and become virtually ubiquitous on digital boards. One can hardly find an embedded CPU, typically at the core of most digital applications these days, without at least one or more such built-in I²C bus interfaces. This enables interconnection between numerous available peripheral devices and adds a wide spectrum of capabilities and functionality to the processor. And all this may be accomplished with just two open collector / open drain signals (See Figure 1).

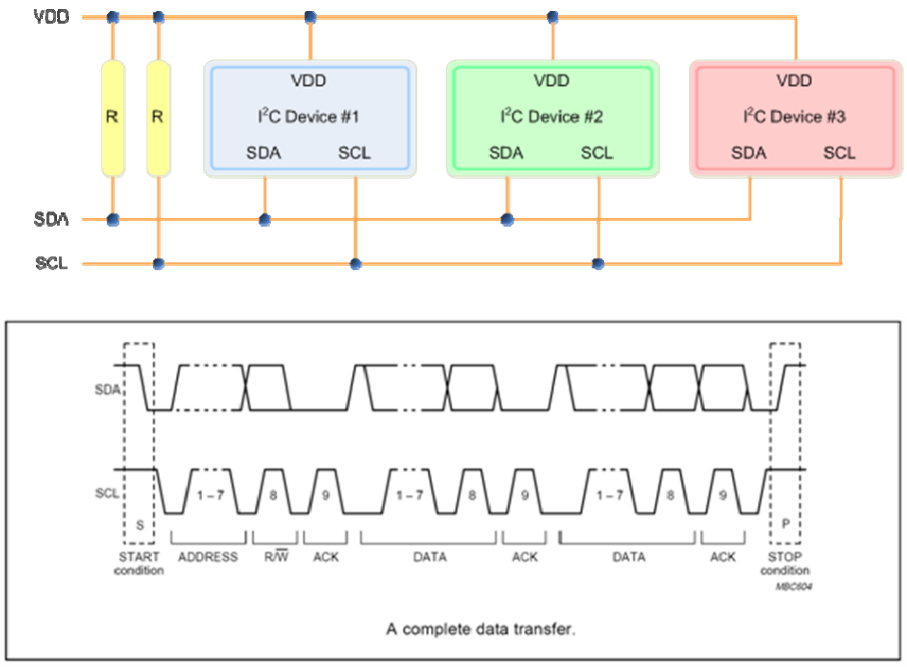


**Figure 1 – I²C Protocol Interconnection and Example Data Transfer**

Another key attraction of this bus is the very minimal hardware resources required to link a myriad of elements throughout the board or system. It ties together various

specialized slaves and/or additional arbitrating multiple masters. The protocol is ideal for moderate performance communications of board end-points, focused mainly on setup configuration, housekeeping, and general information exchange. It is not only used to connect devices on the board, but is also used in multi-board systems such as servers, linking boards and systems higher up in the hierarchy as well. The $I^2C$ bus is also used to carry higher level system protocol layers, having migrated to the domain of platform, server and rack-shelf availability and health monitoring networks. This includes IPMI and low level visibility of power sources and batteries (SMBus and PMBus).

Developers at all levels, ranging from devices up to complex systems, clearly need tools to expedite their $I^2C$ operational and compliance validation efforts, confirming conformance and reliable interactivity with 3rd party connections.

## Development Tools

Fortunately, there are readily available tools, providing the user needed visibility of $I^2C$ bus activity. These serve well when the focus is observing and/or recording traffic produced by a given bus installation. Some instruments also include user-directed or programmed drive capability to mimic a master or slave sitting on the bus, supporting a peek-and-poke function. Many also provide the ability to alter the bus electrical characteristics when needed, such as:

- Programmable pull-up resistor
- Selectable External/Internal pull-up reference voltage source
- Programmable pull-up reference voltage
- Programmable clock rate
- Adjustable timing skew between clock and data signals

Some instruments further depict timing and/or signal waveforms to aid in visualizing bus activity. Several also track protocol sequencing to identify violations in support of debugging target misbehavior. This often includes the ability to convey corrupt protocols to confirm appropriate target responses.

However, device or system developers soon learn of limitations when they need to fully validate their product's compliance with the $I^2C$ protocol, or operate beyond the normal bus ranges to assure sane behavior during unusual or stressful conditions. These are key operations needed to ship products with high confidence of robust conformance, but tend to be sorely lacking in the run-of-the-mill $I^2C$ bus analyzers.

Even the mere task of characterizing a device's timing compliance can be daunting. Such testing often requires a lab bench full of assorted instruments to confirm product protocol adherence and tolerance to deviant situations.

# The Problem: Difficult Classes of Bus Validation

One specific aspect of $I^2C$ behavior which is difficult to test is bus signal timing when driven by the user target.

Another challenging validation class is determining proper target response when another device drives the signals toward it. This case is especially difficult since, while bus behavior can be observed and measured, internal target events (such as the capture of a written byte) can only be inferred by that receiver's outcome. i.e. Are we sure a byte write has been correctly received and utilized in the target, regardless of the integrity of bus signals?

Furthermore, how does one test over a range of boundary conditions such as varying setup times? It would be useful to sweep from above to below nominal, for example, to capture the actual point of failure of the target.

Another concern is the sheer number of bus parameters which need to be validated without incurring a heavy up-front test design effort. Ideally, an instrument should automate this validation process as much as possible and be repeatable, with just the click of a mouse. This is especially true when performing production acceptance test procedures (ATP) for a run of products. Furthermore, an instrument cognizant of the $I^2C$ bus protocol and topology can clearly reduce test development time and costs.

# The Solution

## *Bus Parameter Measurements*

Fortunately, there is an instrument specifically designed to handle such issues: the CAS-1000 $I^2C$ Bus Analyzer from Corelis. This feature-rich instrument is loaded with capabilities to address the abovementioned complex test and measurement challenges while still including all the expected capabilities of more conventional $I^2C$ instruments, including:

- Monitoring/endless-logging visibility
- Bus interaction (peek-and-poke)
- Concurrently emulating up to one master and/or up to 10 slaves directed by text script programs
- Protocol corruption injection/detection
- Bus electrical characteristic alterations by adjustable pull-ups and reference voltage levels
- Built-in flash programming algorithms
- Automatic scripted bus testing

A key unique feature of this instrument is the Parameters Scope, which directly addresses the difficulties of measuring and validating the target's signal level conformance to the $I^2C$ standard. The Parameters Scope includes a large number of automatic bus characteristics measurement algorithms, which can automatically capture and validate virtually the entire set of observable bus requirements. These are measured and collected directly from the bus behavior produced by the target masters and/or slaves. Additional bus system parameters are generated and collected by the CAS-1000 itself, by directly stimulating the signals such as for signal capacitance or pull-up resistors.

For example, Figure 2 and Figure 3 screenshots display sets of system and master parameter measurements to easily view, correlate and validate information.

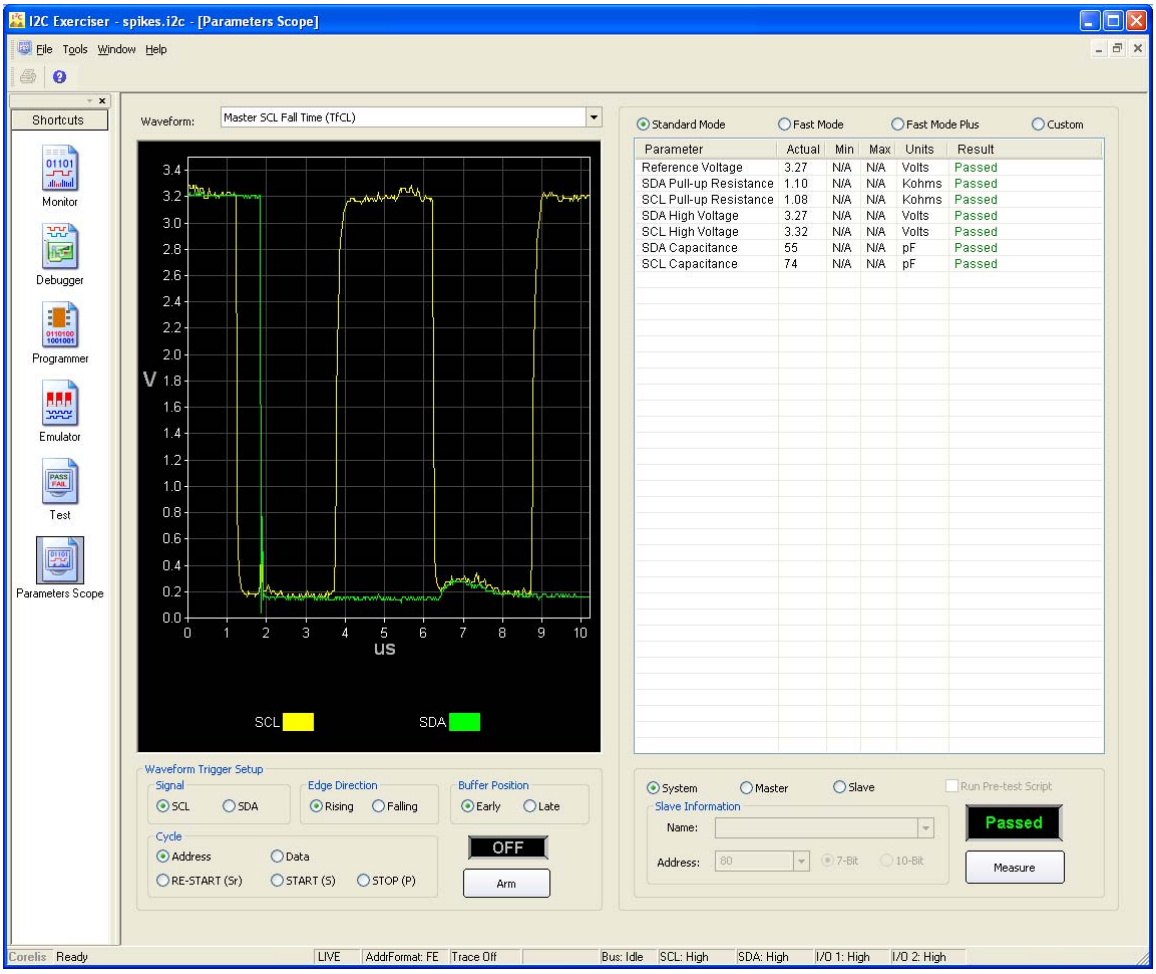


**Figure 2 – Measuring and Validating System Parameters**

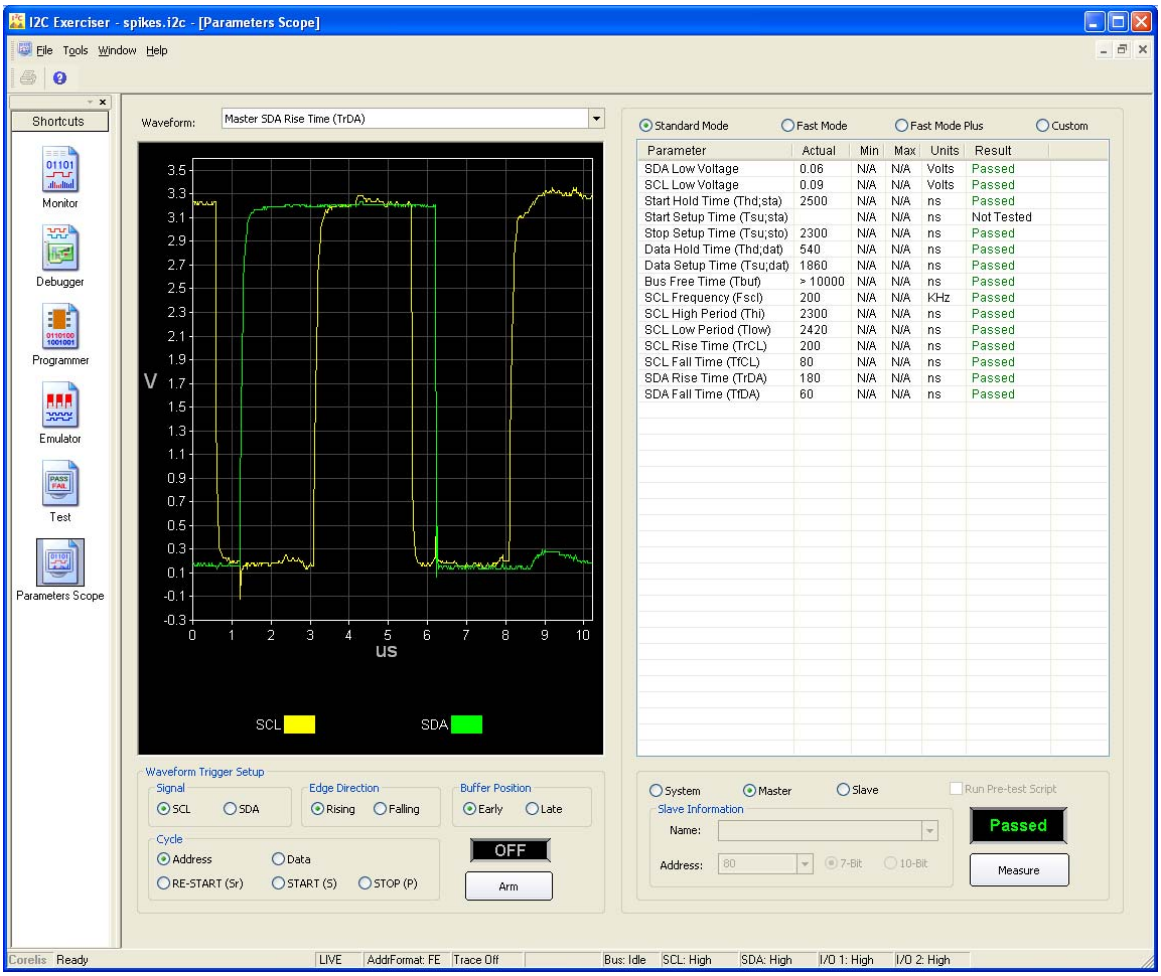


**Figure 3 – Measuring and Validating Master Parameters**

The complete list of $I^2C$ bus parameter measurements available as automatic tests in the CAS-1000 is listed in Table 1. Note that this list is virtually a duplication of the $I^2C$ bus specification parameters. Also, the min/max limits for pass/fail checking can be selected between Standard, Fast, Fast-plus, and custom modes.

| *I²C Parameter* | *Description* | *Unit* | *Measurement Type* |
|---|---|---|---|
| SDA | Current SDA Logical Level | - | SIGNAL_LEVEL |
| SCL | Current SCL Logical Level | - | SIGNAL_LEVEL |
| Discrete1 | Current Disrete1 Logical Level | - | SIGNAL_LEVEL |
| Discrete2 | Current Discrete2 Logical Level | - | SIGNAL_LEVEL |
| Vref | Reference Voltage | mV | SYSTEM |
| SDAPullUp | SDA Pull-up Resistance | Ohm | SYSTEM |
| SCLPullUp | SCL Pull-up Resistance | Ohm | SYSTEM |
| SDAHigh | SDA High Voltage | mV | SYSTEM |
| SCLHigh | SCL High Voltage | mV | SYSTEM |
| SDACap | SDA Capacitance | pF | SYSTEM |
| SCLCap | SCL Capacitance | pF | SYSTEM |
| SlaveSDALow | Slave SDA Low Voltage | mV | SLAVE |
| SlaveThdDAT | Slave Data Hold Time | ns | SLAVE |
| SlaveTsuDAT | Slave Data Setup Time | ns | SLAVE |
| SlaveTrDA | Slave SDA Rise Time | ns | SLAVE |
| SlaveTfDA | Slave SDA Fall Time | ns | SLAVE |
| MasterSDALow | Master SDA Low Voltage | mV | MASTER |
| MasterSCLLow | Master SCL Low Voltage | mV | MASTER |
| MasterThdSTA | Master Start Hold Time | ns | MASTER |
| MasterTsuSTA | Master Start Setup Time | ns | MASTER |
| MasterTsuSTO | Master Stop Setup Time | ns | MASTER |
| MasterThdDAT | Master Data Hold Time | ns | MASTER |
| MasterTsuDAT | Master Data Setup Time | ns | MASTER |
| MasterTbuf | Master Bus Free Time | ns | MASTER |
| MasterFscl | Master SCL Frequency | Hz | MASTER |
| MasterThi | Master SCL High Period | ns | MASTER |
| MasterTLo | Master SCL Low Period | ns | MASTER |
| MasterTrCL | Master SCL Rise Time | ns | MASTER |
| MasterTfCL | Master SCL Fall Time | ns | MASTER |
| MasterTrDA | Master SDA Rise Time | ns | MASTER |
| MasterTfDA | Master SDA Fall Time | ns | MASTER |

**Table 1 - Set of Automatically Measured $I^2C$ Bus Parameters.**

Not included in the above list are parameters which depend on the internal actions of the device, so they are hidden inside the device and are not directly visible on the $I^2C$ bus. These include:

- Timing margins
- Glitch tolerance
- Proper capture of the SDA data stream by the receiving device

This brings up the second major class of validation, which involves the verification of proper target responses to verify these internal device actions. The CAS-1000 feature we will discuss next will address these issues as well.

## *Target Timing Stressing*

Another major unique capability of the CAS-1000 is the Glitch Pattern Injector. Besides its implied usage to force glitch spikes onto the bus as a means of testing target reaction, this tool is very useful to stress $I^2C$ timing. For example, the user can program it to inject a complex signal pattern on SDA and/or SCL at some trigger point in order to alter various timing parameters. This easily and rapidly allows the creation of arbitrary and re-usable sequences on the bus to stress connected devices by:

- Adjusting signal setup/hold times over a wide range to the target
- Applying corrupted patterns of virtually any pattern
- Forcing SCL clock stretching of varying amounts on the bus at specified points
- Forcing multi-master collisions on the bus
- Forcing protocol violations of target bus drivers
- Generating  many more possible anomalous signal patterns

Some examples of actual target testing where the Glitch Pattern Injector tool was used for various target stress testing can be seen in Figure 4, Figure 5, and Figure 6.

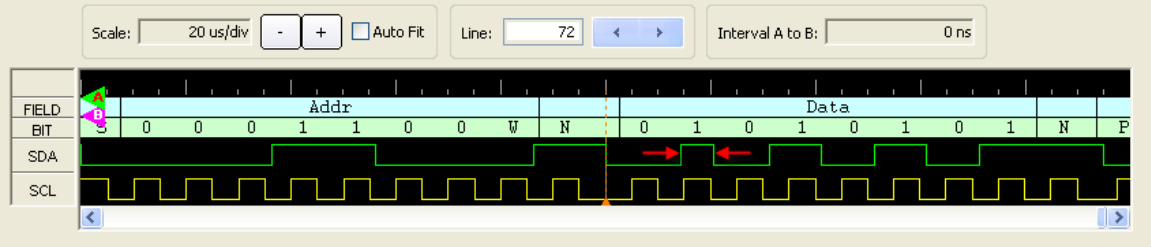


**Figure 4 – Glitch Pattern injected to alter setup & hold times of 2nd data bit**

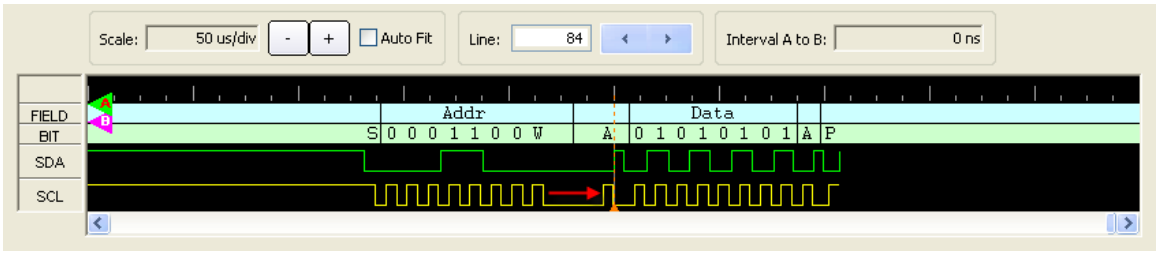


**Figure 5 – Glitch Pattern injected to force clock stretching on target**

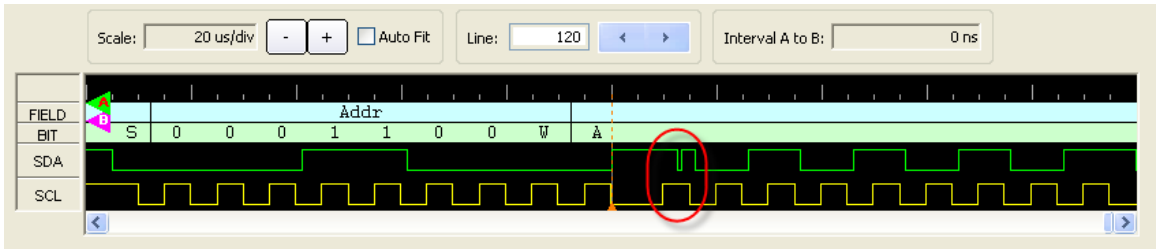


**Figure 6 – Glitch spike injected onto target traffic**

The Glitch Pattern Injector not only can inject intentional noise/glitch/errors on the bus, but it can also be used as an arbitrary pattern generator. This pattern generating capability can be extremely useful for testing the target's responses in boundary conditions for the purpose of validating the target's $I^2C$ compliancy. By creating and generating $I^2C$ signals with various hold, setup, high, and low timings within the $I^2C$ specification, you can push your devices to the limit and verify whether your devices are behaving properly at those minimum or maximum conditions. You can also produce waveforms beyond the specification boundaries (e.g., setup time or hold time less than minimum) to validate robustness and tolerance limits of your target devices, or determine actual failure points.

The following is a simple example of stressing a target slave device with the minimum data setup time. In order to verify whether the slave device being tested is responding correctly to the minimum setup time, you can create an address transaction pattern with a desired setup time and inject it to the bus to observe the target slave's response. According to the $I^2C$ specification, in the Fast mode for example, the minimum setup time is 100 ns. Therefore, you need to create the pattern with 100 ns setup time to verify the compliance. Figure 7 is an example of such a pattern, which will generate the slave address of 0xA0.


**Figure 7 – Creating a Pattern with Minimum Setup Times**

Figure 8 shows the timing display of the signals captured after the above pattern was injected. As shown here, the target slave device was responding with an ACK even with the setup time of 100-ns. This verifies that the target slave is compliant to the $I^2C$ Fast mode specification in regards to the SDA data setup time.

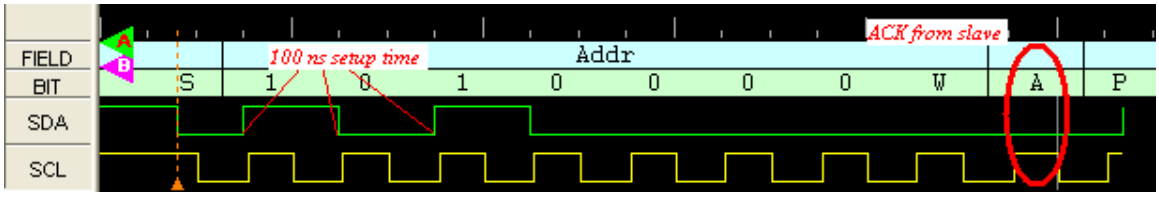


**Figure 8 – Timing Display Showing Acknowledgement of Slave to min Setup Time**

Using similar methods as shown above, you can easily adjust various timing parameters such as data hold time, stop setup time, start hold/setup time, etc., for the purpose of stressing your targets and observing their behaviors. These various patterns can be saved to files and used repeatedly at later times. You can also organize and automate your test cases by creating test scripts with calls to various glitch pattern injections.

# More Common Standard Features

As stated earlier, the CAS-1000 naturally includes an in-depth array of more conventional standard bus visibility tools. A typical trace monitor screenshot, shown in Figure 9, depicts a target observation session subset. The figure shows the trace listing and the perpetual precision signal timing waveforms which accompany all collected data. Also shown is the Debugger tool enabling user interactive peek-and-poke communications with the bus, also recorded by the monitor.

Not seen here are there numerous agile options and display formats available for user customization.

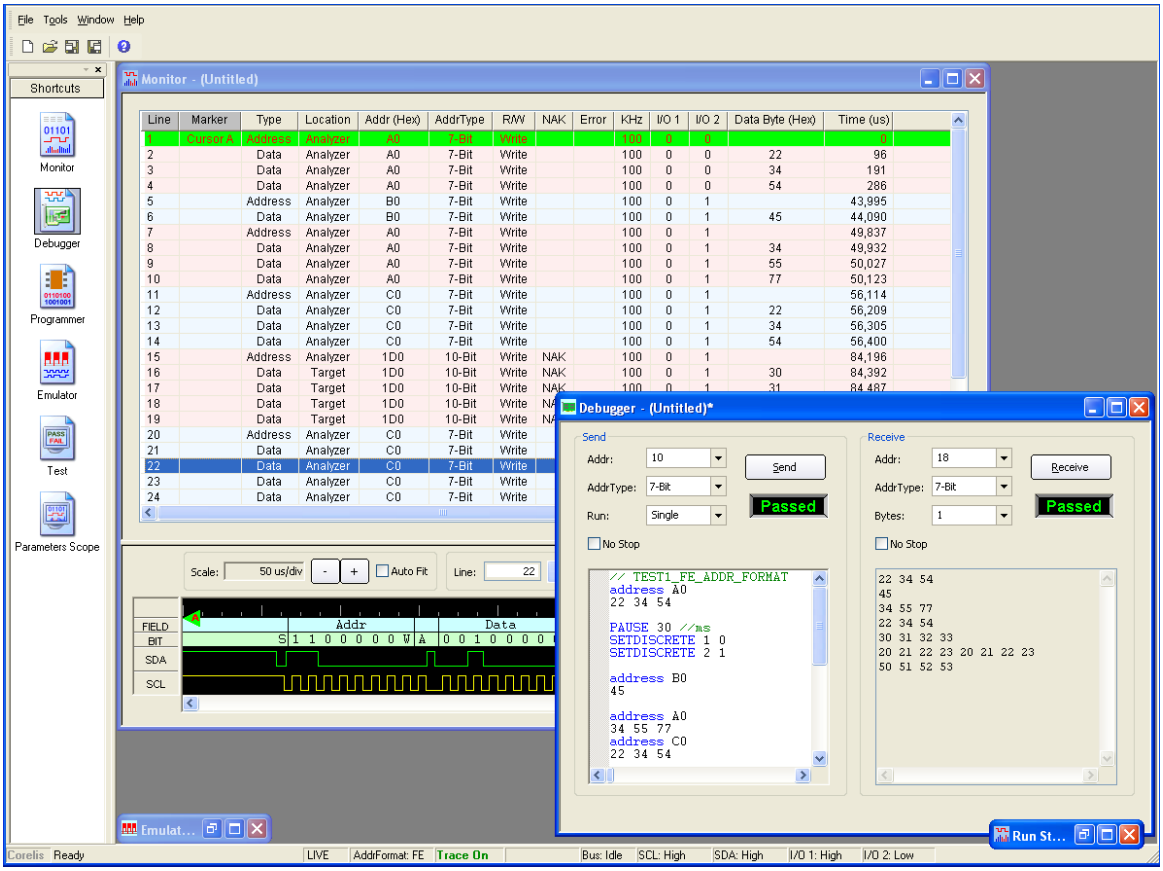


**Figure 9 – Bus Observation Session with Debug Interactive portal Screenshot**

# Conclusion

Developing devices, boards or systems to use the exclusive benefits of the $I^2C$ bus requires care to assure that interoperability will result with complete integrity. This is greatly facilitated with a precise instrument such as the Corelis CAS-1000, which is focused on providing comprehensive and automatic measurements specific to target bus behavior. In addition, methods to stress the target timing or protocol sequences can further validate and qualify target responses during extreme or deviant conditions. Useful indeed is the CAS-1000's ability to determine accurately the actual failure points of the target.

While simple monitoring or interactive I/O tools are adequate for basic visibility of such $I^2C$ bus traffic, they are typically lacking in the advanced capabilities necessary to completely assure compliance, especially when high quality product confirmation is a must. The case of the unprecedented Corelis CAS-1000 analyzer is presented here as a valuable solution toward satisfying such critical target validation requirements.

# About Corelis

Corelis, Inc., a subsidiary of Electronic Warfare Associates, Inc., offers bus analysis tools, embedded test tools and the industry's broadest line of JTAG/boundary-scan software and hardware products combining exceptional ease-of-use with advanced technical innovation and unmatched customer service. Corelis' development and test tools are used by companies such as Agilent, Dell, IBM, Jabil, Intel, Microsoft, Lockheed Martin, Rockwell Collins, Hewlett-Packard, Motorola, Qualcomm, Nokia, Panasonic, TI, Ford, Plexus, Broadcom, Ericsson, Flextronics, and many others. Corelis products are found globally in every industry developing or manufacturing electronic products.