Debugging I^2C-Bus-based Systems

By Michael Wöstenfeld, INTRON GmbH

The I^2C bus, developed in the early 80s by Philips, has become an industry standard in consumer products and is also widely used in industrial applications. This test tip describes an easier way to measure and debug this serial bus than typical oscilloscope/logic analyzer methods.

History

The I^2C bus was developed in the early 1980s by Philips for their consumer products with multiple components that needed to be linked together. For example, a television set has many functions controlled by components, including: memory for storing channels, RTC, tuner, brightness, contrast and many additional functions invisible to users, such as the reference black and white levels of the CRT. Controlling these functions with parallel bus techniques would require high resource commitments and a lot of space.

Philips solved the problem with a 2-wire bus system. The speed requirements for their applications were not high, so they used transmission rates of (up to) 100 KHz. Today, the bus is specified up to 400 KHz. The I^2C bus is multimaster-capable. The exact specification can be found in “DATA HANDBOOK IC12", which is available on the web (www-eu.semiconductors.philips.com).

Problems

As microcontrollers became more complex, industrial users ran into similar problems. Today, a typical microcontroller can incorporate as many 40 I/O pins, 8- or 10-bit A/D converter channels, and timers, etc., so that there is often no need for a parallel communication bus for external peripherals. However, as always happens, some system functions are always missing. An RTC may be nice to have, and a small EEPROM may be necessary to store nonvolatile data for power-down phases. Many IC manufacturers recognized this need for additional peripherals, so a lot of industrial chips are available to meet various needs: ADC and DAC up to 8 channels/12 bit, volatile and non-volatile memory up to 8 Kbyte, sometimes including reset-generator and watchdog timers, LCD units, digital in- and outputs, potentiometers, RTCs and much more.

While connecting the devices together is easy, debugging them can be difficult. Parallel units can be debugged with a standard scope, —although not necessarily easily— a logic analyser, or, in some cases, a combination of an emulator and one of the above. Serial busses are often hard to trigger on, and there is no time-reference to the data because data is read bit-sequentially. In parallel units, logic analyzers give you the data and you have to look only for timing relationships among the parallel data. However, in serial units, you have to look for both sequential data and the sequential timing. Additonally, data serializing/deserializing is done via microcontroller software, in most cases, and there's no guarantee that the software will work properly at initial turn-on.

Solutions

Agilent's new 5462x-series oscilloscopes incorporate a powerful I^2C-triggering feature which makes debugging much easier from previous testing solutions. All you have to do is select SDA and SCL lines from the triggering menu. In addition, with the Agilent mixed-signal

oscilloscope (MSO) versions, you can use all 18 channels (2 analog scope channel plus 16 logic timing channels) to view additional I/O. After assigning the data (SDA) and clock (SCL) to specific scope channels, you can then select your I^{2}C serial triggering options. Triggering options includes start-bit, stop-bit, Frame Start - Addr – Read (Write) - Ack - Data and Not (Frame Start - Addr – Read (Write) - Ack - Data) as seen in Figure 1.

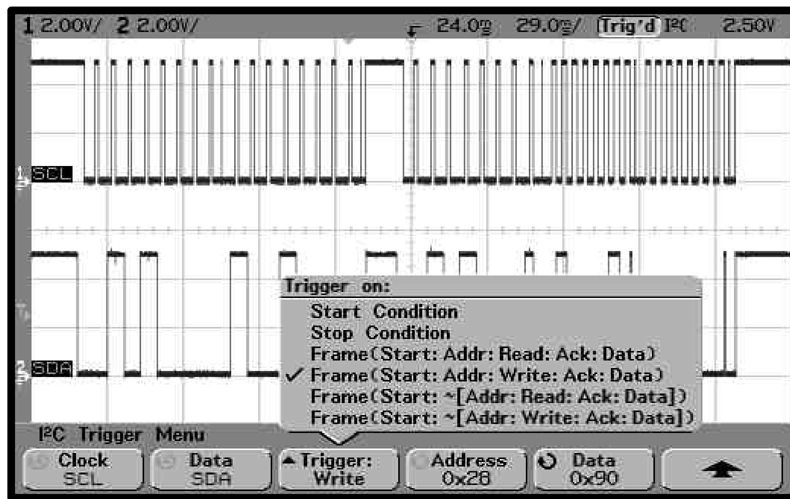

Figure 1: Setting up I^{2}C triggering on the Mixed Signal Oscilloscope

Demonstation

To demonstrate these features, I used a “Basic-Stamp II” (BS2) from Parallax (www.parallaxinc.com). This is a small PIC-based controller board with an integrated BASIC-Interpreter, housed in a 28-pin, 600-mil standard DIP case. An optional carrier board provides power supply, serial interface and a prototype/breadboarding area.

For the I^{2}C device under test, I selected a MAX127 A/D converter from Maxim (www.maxim-ic.com). This chip is best suitable for industrial analog measurements. It provides 8 protected input channels with 12-bit resolution and has a built-in reference. The input channels are software-selectable form 0 to 5, +-5, 0 to 10 and +-10 Volts.

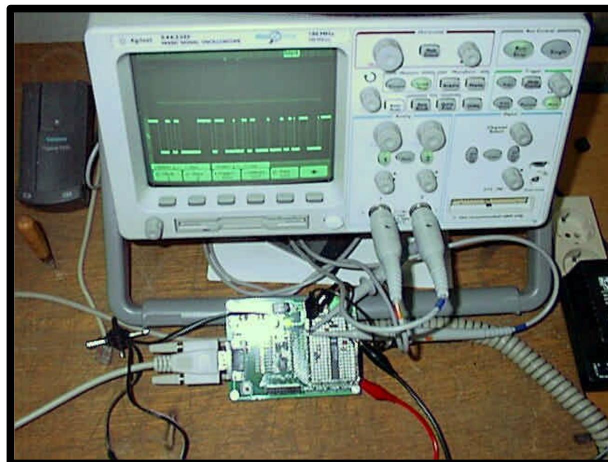

Figure 2: Test setup using the BASIC Stamp II and mixed signal scope

The whole test equipment setup can be seen in figure 2, which includes Agilent‘s MSO 54622D (www.agilent.com/find/MegaZoom). The schematic of the piggy-backed A/D converter is shown in Figure 3. The source code for the BS2 is shown in listing #1.

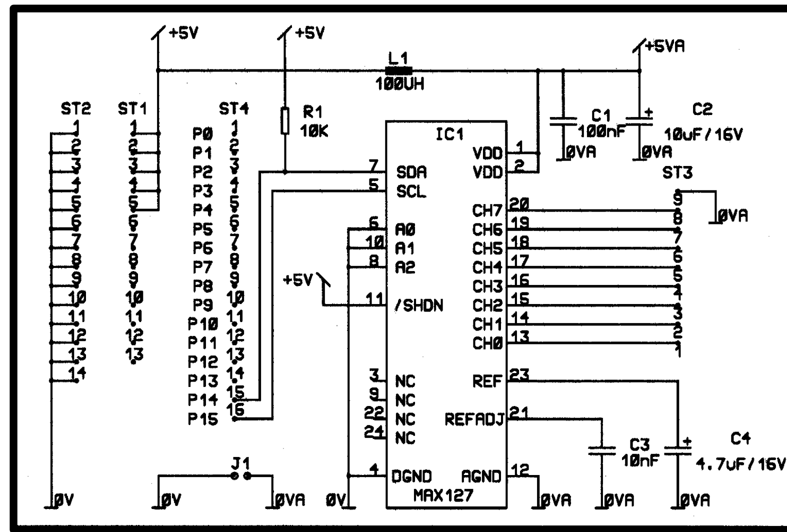

Figure 3: Piggybacked A/D converter schematic

The BASIC program (master) communicates with the MAX127 (slave) and outputs the received data in its debug-window (Figure 4).

```
-----------------------------------
Agilent 54622D, BS/2 and MAX127

cw ch  Range hex  dec   mv
--|---|-----|----|-----|-------
80 CH0  0-5V 41Fh 1055d 1298 mV
90 CH1  0-5V 41Fh 1055d
A8 CH2 0-10V 20Eh 0526d
B8 CH3 0-10V 20Eh 0526d
C4 CH4  +-5V 222h 0546d
D4 CH5  +-5V 222h 0546d
EC CH6 +-10V 11Dh 0285d
FC CH7 +-10V 11Dh 0285d
```

Figure 4: BASIC program debug-window

First, a start condition issued by the BS/2 — which is a high to low-transition on SDA while SCL is high — has to appear on the bus (Figure 5). Next, 8 bits are clocked into the MAX127. The first seven bits (7 to 1) are the 7-bit slave address defined by the manufacturer of the device, in our case 0x28. Because bit 0 dictates whether it is a read or write command, we first have to write the control word,"0," for write. To acknowledge the data, the MAX127 pulls SDA low for one clock cycle (See “A” on Figure 5).

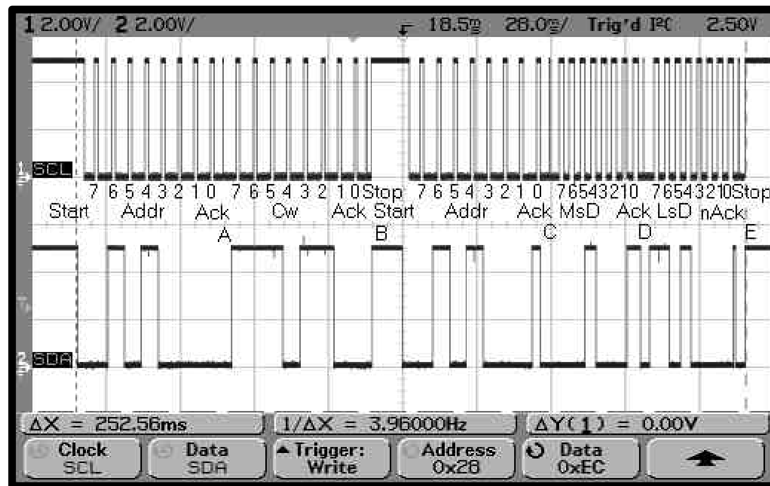

Figure 5: Clock and data timing as shown on the mixed signal oscilloscope.

The next 8 bits (“A” to “B”) are the control byte, used to select channel number (bit 6-4), range (bit 3,2) and power-down modes (bit 1,0). Bit 7 must be one. The slave acknowledges this data. Then the BS/2 ends the write cycle by issuing a stop condition, which is a low-to-high transition on SDA while SCL is high (See “B”).

The sample data is read back in a similar way: first a start condition, then the slave address, (with bit 0 set to 1 for a read), and an acknowledge from the slave (“B” to “C”).
The first data byte — in this case defined as D11 to D4 of the A/D data — follows immediately (“C” to “D”). When the master has issued an acknowledge, the slave transmits the second byte (““D” to “E”), defined as D3 to D0 on bits 7 to 4. Bit 3 to 0 are always zero. To end the cycle, the master issues a "Not" acknowledge condition, followed by a stop condition.

The two bytes then have to be shifted a little and sorted by software to get an integer result. Some chips can generate additional wait states on the bus, but this is not supported by this program.

In the debug windows, first, the written control word appears to make things easier. Next, channel number and the requested range appear. Readback data are shown as hex, decimal, and, for channel 0, as mV, which is rounded slightly due to the limited math capabilities of the BASIC Stamp II.

The complete cycle for reading the 8 channels and displaying them onto the debug-window (40 bytes to transfer) takes about 2.4 seconds. This would send some scopes with low memory depth into a state of confusion.In addition, with most scopes you would have to set up an I/O port bit as a trigger marker, because of the lack of ability to trigger on bit streams. With Agilent´s 54600-Series scopes with 2MB of acquisition memory, it is possible to take a single-shot of the whole cycle as an overview and then zoom into every bit.

There are other helpful features, as well. With the I^2C triggering feature, you can “pick out” every single transaction on the bus. Let's take channel 6 as an example. The debug-window (Figure 4) shows a control-word of 0xEC for this channel. Setting up the scope is easy: You press “more” on the trigger section of the front panel. With the softkeys you select “I^2C” and then “settings”. Once you have selected clock and data lines, and perhaps named them, all you have to do is select “Trigger: Write,” set up the address to 0x28 (general for MAX127 in

this application) and the data to 0xEC. As seen in the description of the bus-cycles, Figure 5 shows the expected result. As the signal is repeated sequentially, it is easy to look at single bits or clocks by zooming and panning the timebase. But even with a “single shot” it is possible to spot unexpected cycles. Additionally, different cursor measurements and a delayed timebase help you verify the proper setup and hold times, for example. Also, various automatic measurements are available to check signal quality.

Speeding things up

In the above solution, it takes about 250 ms to digitize one channel; optimizing the software could reduce the time to 200 ms. But the BS2, unlike the older BS1 with which the above program also works, has powerful features to avoid toggling bits manually for a serial data stream. Shiftin/Shiftout commands send out data and clock signals in various configurations. Listing #2 shows a possible solution which speeds up a conversion to 18 ms.

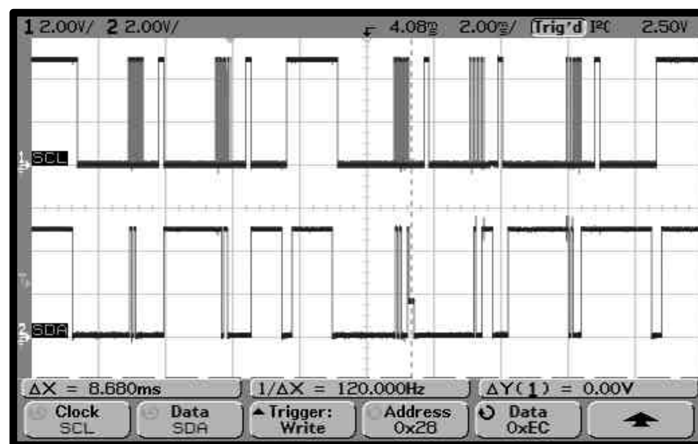

Figure 6: Clock and Data signals in the sped-up mode.

As we see in Figure 6 (channel 2, cursor position), this solution has a side effect. Because the Shiftout command is designed to do unidirectional communication it leaves SDA in an active state to spare an otherwise needed pullup or pulldown resistor. So until the next command “input SDA_PIN” is executed, the line is driven because of the “read-bit” in the comand. But directly after the clock, the MAX127 issues an acknowledge (SDA=low) condition, and for this time there is a bus conflict between the two devices.

To avoid this problem, shift out only seven bits and then generate the eighth bit by toggling the ports respectively using the pullup on SDA. If the subroutine "out_byte_s_ack_ok“ in routine getad is executed instead of "out_byte_s_ack," the program runs a little slower but avoids the conflict, see Figure 7.

Observing and resolving bus conflicts are good reasons to use a deep-memory scope with advanced triggering features instead of a logic analyzer, where the captured results would not have been nearly as easy and intuitive to visually interpret.

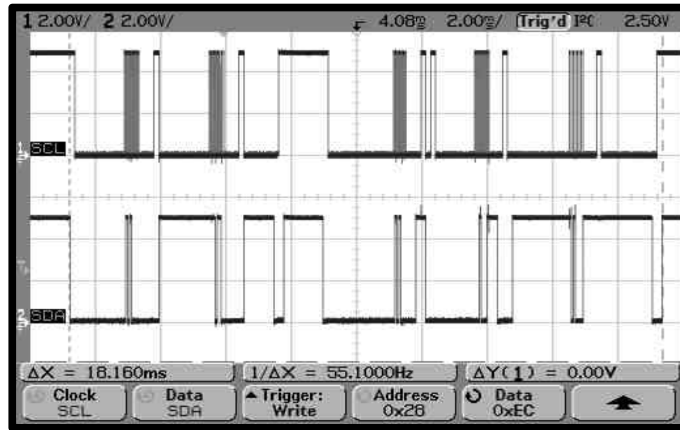

Figure 7: Avoiding conflict.