Byte Paradigm
White Paper

byteparadigm

Solutions for SPI protocol testing and debugging in embedded system.

There are many different reasons to use serial protocols in embedded systems. Simplicity, low pin count and the ability to setup a kind of network of simple devices to implement a complex functionality are probably the most frequent reasons for using them in embedded systems. Focusing on the Serial Peripheral Interface (SPI), this paper explores the reasons to test and debug a SPI port. It then describes and compares the tools available on the market to do so, from general purpose oscilloscopes to specialised PC-based SPI exercisers and analysers.

Serial buses are everywhere in embedded design.

Among the existing protocols, SPI and I²C have established themselves as *de facto* standard. They are well suited for low-cost and (not always so) low-speed communication between ICs, between microprocessors and between microprocessors and peripherals. EEPROMs, real-time clocks (RTCs), ADCs, DACs, thermal management devices and LCD controllers are such peripherals. Table 1 briefly compares the 2 protocols.

	SPI	I²C
Clock frequency	free (n x MHz, 10n x MHz, ...)	10 kHz, 100 kHz, 400 kHz, 1 MHz and 3.4 MHz
Number of wires	4	2
Duplex	Full duplex	Half duplex
Multi-master	Yes[1]	Yes
Multi-slave	Yes	Yes
Built-in slave addressing	No	Yes
Access length defined	No	Yes
Flow control	No	Yes

Table 1 : SPI and I²C compared

SPI and I²C compete well at different levels for the usage mentioned above, but we can say that SPI is better suited than I²C for 'data stream' applications that use its full duplex capability, as opposed to reading and writing addressed locations in a slave device. As example of a 'data stream' application, we can think of transfers between a microprocessor and a DSP or sending data samples to a DAC.

Strictly speaking, there are many 'SPI protocols'. Designing a port with a protocol similar to SPI might seem quite straightforward. After all, it is all 'just' about producing a reference clock signal to generate and sample data onto 2 separate lines for in and out, let them play simultaneously and select the slave with a third slave select (should we say 'chip enable') line...

---

[1] Multi-master is possible but is not defined by the SPI standard and not often done. It may require additional signals.

Well, if you think to it, there are so many variations from the protocol first established by Motorola (4 wires, byte-by-byte scheme). To get an idea of the possible combinations, just ask yourself the following questions:

> ▸ *Will the protocol use a continuously generated clock or will the clock be active only when data are sampled and generated?*
> ▸ *In case of a non-continuous clock, what would be the clock line default level?*
> ▸ *Will the protocol allow any data length on the serial lines or be restricted to burst access of, say, 8 bits?*
> ▸ *Which clock edge will be used to sample the incoming data? Rising or falling?*
> ▸ *Same question for data generation: rising or falling clock edge?*
> ▸ *What is the polarity of the slave select lines?*

Clock rates, clock phase, clock continuity, sampling and toggling edges, signals polarities: actually, none of these parameters are really known in advance when using a 'SPI' device, or, should we say a 'SPI-like protocol device'[2].

This may be one of the reasons why SPI is nowadays so ubiquitous: in its principle, it is quite simple and nothing is really defined by any authority committee. Aside, because it is the peripheral protocol used for many popular processors from Motorola, then Freescale Semiconductor, such as the MPC8260 (communication processor) and microcontrollers such as the M68HC11 well there is a chance that you'll run into it one or other of these beautiful days.

Manufacturer	Device Types
Analog Devices	DSP, ADC, digital Pot., codec
Atmel	CPU, EEPROM, digital Pot.
Cirrus Logic	ADC, DAC, codec
EPSON	RTC
Fairchild	EEPROM
Freescale Semiconductor	DSP, MCU
Infineon	Pressure Sensor
Intel	CAN Controller
Linear Technology	ADC, DAC, Temperature Sensor + Voltage Monitor
Macronix	FLASH
Maxim	ADC, DAC, UART, Analog Switches
Microchip	Micro controller, EEPROM, ADC, CAN controller
National Semiconductor	LCD Controller, dig. temperature sensor, USB Controller
SanDisk	FLASH, MultiMediaCard
Texas Instruments	DSP, ADC, DAC
Zilog	MCU

Table 2 : Non-exhaustive list of SPI devices manufacturers

---

[2] In comparison, I²C has remained more 'pure' over the years.

Revision 1.00 – 19-Jun-08   **2/9**
Byte Paradigm Products are available from:
The Debug Store, Berwyn House, Carrog, CORWEN, LL21 9AT, UK www.TheDebugStore.com

Why would you want to 'debug' a standard protocol?

After all, if SPI has been around for all this time, it is very unlikely that its current implementation would cause any trouble at all... All any engineer would have to do is to connect a network of SPI-compliant devices and let them play, right?

Actually, there are many cases where you would like to check what is going on at the protocol level. We pointed out 4 possible cases, summarised in the table below.

SPI Debug / Test Case	Description
CASE 1: Don't debug SPI, debug through SPI	A SPI functional port is used as an access port to gain visibility on an embedded system by analysing the traffic exchanged between the devices on a SPI network and by stimulating the embedded system through its SPI interconnects.
CASE 2: Command or higher level of the stack built on SPI	The SPI port is analysed and stimulated to test and debug the protocol stack built on top of the SPI signalling.
CASE 3: Detect which SPI protocol is used	The SPI port is debugged to check its parameters and how it derives from the standard 'pure' SPI interface.
CASE 4: Debug and test your own design of a SPI controller	A SPI port can be part of a custom design on CPLD, FPGA, ASIC or SoC. Like any design, this requires debug and test, which can be performed with SPI-oriented test and debug tools.

Table 3 : Overview of SPI test and debug cases

CASE 1: Don't debug SPI, debug through SPI

Organising testing and debugging of an embedded system requires choosing several input and output ports used to access and observe the behaviour of the system.
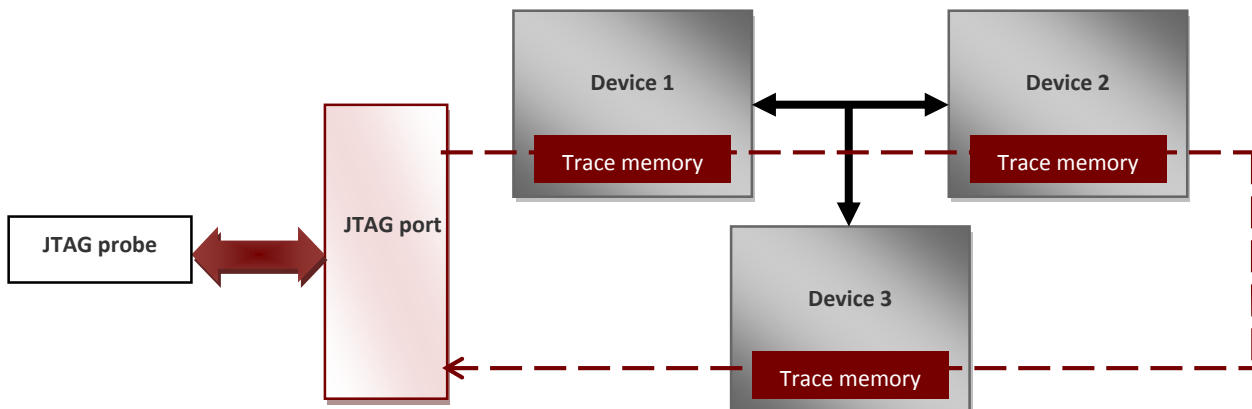
JTAG scheme

Figure 1: System debug done by collecting trace data from system memories through JTAG port

Byte Paradigm Products are available from:
The Debug Store, Berwyn House, Carrog, CORWEN, LL21 9AT, UK www.TheDebugStore.com

JTAG often comes in mind first, because it is used by emulators for software tracing. In many cases, it can be used to collect data from embedded memories and other chips like CPLD and FPGAs. However, while efficient for very specific uses (like single processor software debug) such a debug port offers a very low speed communication and can only be used like a 'side door' to access a whole system and collect previously stored trace data. Using JTAG to bring stimuli to a system and using the same JTAG port to collect results from the same system is a viable debug and test scheme as long as memory (for tracing storage) and speed are not an issue.

SPI scheme: intercept functional SPI traffic by placing an instrument on the bus

Now, let us imagine that your embedded system is composed of a network of devices like processors, peripherals, programmable devices (CPLD / FPGA) that use SPI protocol to communicate with each other.

Intercepting and interpreting the SPI traffic exchanged on this network will bring valuable and 'real-time' information about the system, allowing you to actually *observe* the system. With a careful trigger generation and/or data sorting, you can narrow your search of a mighty bug. Moreover, if you can play arbitrary sequences or *replay recorded sequences* onto the SPI bus whenever you like, you'll be able to stimulate the system very precisely, at a very low level.
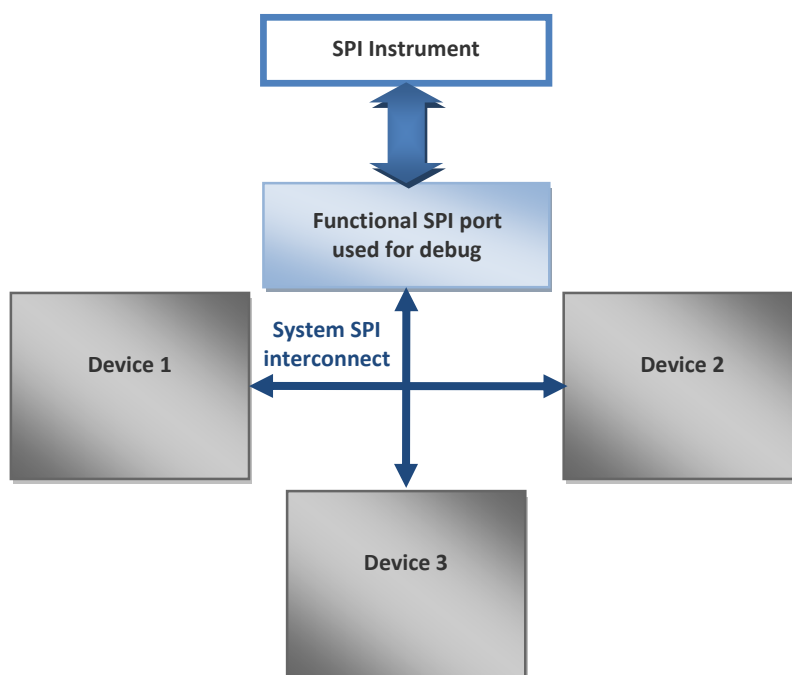

Figure 2: System debug through the functional SPI interconnect

Low-level stimulation and analysis is useful for embedded system test and debug

Processor-centric embedded systems offer a great deal of flexibility for test and debug, since (almost) everything is controlled from the embedded microcontroller and it is very easy to segment and write software to test any part of the system. It is very commonly thought that the embedded microcontroller is the one and only access point to the system for stimulation (through its serial ports and GPIO, for instance) and analysis (through simple register access, for instance).

Revision 1.00 – 19-Jun-08 **4/9**
Byte Paradigm Products are available from:
The Debug Store, Berwyn House, Carrog, CORWEN, LL21 9AT, UK www.TheDebugStore.com

However, even for such systems, low-level stimulation and analysis can reveal extremely useful. Tracking embedded systems bugs is of course about debug the software, but it is also all about having good ways to observe any given peripheral, co-processor or external device. Many debug strategies exist, but at the end, making sure that a whole system works starts by checking the behaviour of each of its individual parts independently.

Bringing stimuli directly to the I/O port of a given device allows early testing of device and hence, early validation of parts of the whole system functionality. This is especially useful during the design process, when not everything is available yet... In brief, there is a need to emulate non-existing parts of a system to test the parts that are already available and observe their response to specific stimuli. In other words, if you have carefully tested each component of a system before assembling them together, you'll increase your productivity if a bug occurs later because you'll be able to narrow your search for the problem.

Similarly, intercepting and observing traffic at the functional interfaces of the embedded system components and correlating them with any running software ran by the processor would increase your productivity. If you observe a system from more than one single access point, you'll increase the visibility you get from it. If you get a high visibility of the system, there is a good chance that you'll speed up the debug process.

Because SPI is very common as functional interface, and because it can reach quite interesting streaming speeds, using SPI as an access port to your system can definitely help test (stimulate) and debug it.

CASE 2: Debug command or higher level of the stack built on SPI

Standard SPI only defines the signal level, not how data is organised. Unlike I²C, SPI does not define any device addressing. It does not define any flow control, not any command[3]... Well, all you have is a few data and control lines and you are free to add any protocol layer to this.

This basically means that implementing a SPI system requires designing a protocol stack – simple or complex, this is up to you... As such, this development must be tested and debugged anyway. This is also a test and debug case where there is a need to plug yourself onto the SPI bus to analyse and exercise it.

CASE 3: Detect which SPI protocol is used

SPI variants are numerous. When something goes wrong with an interconnect of SPI devices, you must first check which SPI configuration is used (SPI mode, clock, ...) and even, if the device 'so-called' SPI protocol really complies with what you think.

CASE 4: Debug and test your own design of a SPI controller

What if you designed a SPI master or slave yourself? Whether you need an I/O port for a CPLD, FPGA, ASIC or SoC, a SPI port has got many advantages:

---

[3] Actually, the original SPI protocol defined serial data grouped by bytes, but there are many devices today that use other arbitrary – and even variable-length bit grouping.

Revision 1.00 – 19-Jun-08   **5/9**
Byte Paradigm Products are available from:
The Debug Store, Berwyn House, Carrog, CORWEN, LL21 9AT, UK www.TheDebugStore.com

- If performance requirements allow it, a serial interface is better than parallel memory-like interfaces for general purpose or register access because it spares costly pins.
- It allows arbitrary clock frequencies, up to several 10th of MHz (even more, if you do a careful routing on the board).
- It is 'self-contained' on 4 (or less) wires: clock, slave select and data lines.

Given its relatively low complexity and because any additional protocol layer will likely be custom (since nothing is defined with the SPI standard), you'll probably end up designing instead of integrating a ready-to-use IP. During and after the development of your own SPI interface, it will be necessary to test and debug it... This is another case where you'll need the proper tools and environment.

What is available for SPI debug and test?

Now, let's have a look at the solutions currently available on the market for SPI test and debug. Table 4 provides an overview of the existing techniques and tools.

Device / tool	Purpose
Oscilloscope and Logic Analysers with SPI protocol analysis support	Probing and analysing SPI port at signal level and provide visualisation and decoding support. **Analyse only.**
Processor with a SPI interface	Provide a 'software' access point onto the SPI interconnect. **Stimulate and Analyse.**
Arbitrary digital pattern generator	Stimulate SPI interconnect at signal level. **Stimulate only.**
PC-based SPI analyser	Probing and analysing SPI port at signal level and provide visualisation and decoding support. **Analyse only.**
PC-based SPI exerciser	Stimulate SPI interconnect at signal level – act as a master onto the SPI interconnect. **Stimulate only.**
Combined PC-based SPI exerciser / analyser	Combination of the 2 tools above: **Stimulate and Analyse.**

Table 4: Overview of existing techniques for SPI debug and test

Oscilloscope and Logic Analysers with SPI protocol support.

This can look as the most obvious solution for any engineer having experience with hardware design. Scopes and logic analysers will let you visualise and record information at signal level from the SPI port and provide enough triggering capabilities to track virtually anything from your SPI traffic.

Many oscilloscope and logic analyser vendors provide specific protocol support (generally: SPI, I²C, CAN and so on...) which are actually software add-ons for your scope or logic analyser. Once you have triggered something from the SPI traffic, these software add-ons work on the sampled data to provide you with protocol-specific information to help visualise and decode it.

On many scopes, you'll be able to see the SPI signals superimposed with the decoding of the protocol according to the settings you would have defined (see Figure 3). This enhances your readability of what is going on at the protocol level. If you want to analyse data on your PC, you'll be able to download what you

Byte Paradigm Products are available from:
The Debug Store, Berwyn House, Carrog, CORWEN, LL21 9AT, UK www.TheDebugStore.com

sampled as a file through the scope PC interface (GPIB, USB, Ethernet, ...) and most often, data will be presented as 'SPI decoded data'.
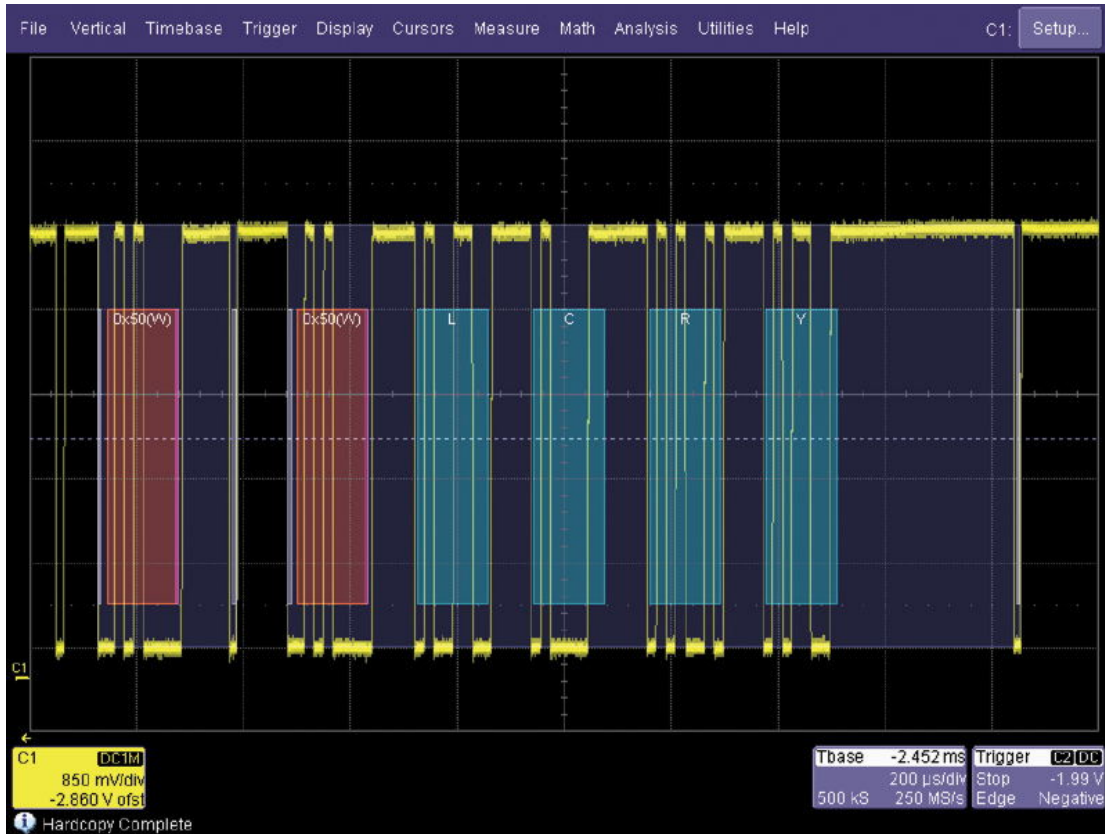

Figure 3: Example of protocol visualisation enhancement in oscilloscope

Processor with a SPI interface

This approach is in fact about using the system resources for self-testing and self-debugging. In this case, the engineer uses embedded software specifically developed for testing and debugging. To get an insight of what is going on in the embedded system, it is all about defining a program that will send the adequate stimuli to the SPI interconnect and use the software itself to read back results stored in the embedded system memory mapped registers or any other storage location. Because this technique only indirectly accesses the SPI interconnect, it requires a great deal of interpretation. It might be your preferred choice if you are skilled with software development and reluctant to dig into the hardware at signal level.

Actually, this approach may be considered as a 'first-line test and debug strategy' and must be used if possible before implementing any other more invading technique. It is very useful to check if the system *'does what it must do'*... and is often the way you'll detect that there is a problem with it, when the results that you receive are not as you expected. According to the problem, you'll need to investigate and perhaps go at a much lower-level, for which you'll need other complementary techniques than just using the embedded system processor as single debug resource.

However, please bear in mind that using a processor for SPI stimulation won't always allow you to send *any* stimulus at clock cycle resolution: by their very nature, it is sometimes very tricky to have a processor produce digital stimuli with a cycle-accurate timing. Moreover, due to software execution latencies, you

Revision 1.00 – 19-Jun-08   7/9
Byte Paradigm Products are available from:
The Debug Store, Berwyn House, Carrog, CORWEN, LL21 9AT, UK www.TheDebugStore.com

won't always be able to stimulate the SPI interface at maximum speed, which may be desirable if you wish to characterise your SPI interface.

Arbitrary digital pattern generator

Arbitrary digital pattern generators are sometimes referred to as digital I/O and have the ability to produce arbitrary digital stimuli. As opposed to the previous 'processor' solution, this is a very low-level solution acting at signal level. Basically, it consists in defining how each of your port signal lines will toggle.

Good digital pattern generator should provide efficient ways to define your pattern, like a programming interface, that would allow you to emulate the SPI protocol.

Digital pattern generator efficiently complement any analysis tool, since they provide a very low-level way to send stimuli to your embedded system and hence, lots of control about what is *exactly* sent onto the SPI interconnect. Arbitrary pattern generator allows overcoming the limitations of the use of a processor (see above) for stimuli generation.

PC-based SPI analyser, SPI exerciser and combined PC-based SPI analyser/exerciser

PC-based instruments are connected to a PC through one of its ports (USB, PCI, Ethernet…) and are provided with a software running on the PC used to control the instrument and process the data generated for the instrument or collected from it.

PC-based SPI analyser and PC-based SPI exerciser are categories of devices specialised for SPI analysis and SPI stimulation. Schematically:

▸ A PC-based SPI analyser samples the embedded system SPI traffic and sends it to the PC memory. Good PC-based analysers provide SPI protocol decoding support, and tools for data visualisation on the PC, like a waveform viewer.

▸ A PC-based SPI exerciser lets act as a master on the SPI bus directly from the PC. It configures the used SPI port (clock characteristics, number of slaves, single access length …), offers interfaces from the PC to define the data to be sent onto the SPI interconnect, and stimulate the SPI interconnect according to the defined timing and other protocol characteristics.

▸ A combined tool can function both as a SPI exerciser and as a SPI analyser.

In comparison with more classical instruments, PC-based SPI tools present the following advantages:

▸ **Interfaces:** good PC-based SPI tools benefit from the flexibility of the PC and present multiple interfaces like GUI and other programming interfaces (for example: C/C++ or scripting interface). Having programming interfaces to control a SPI instrument is especially valuable because you don't quit your PC environment for low-level SPI test and debug. It offers a great deal of flexibility for test automation and data processing and, with careful programming, to develop a fully integrated test and debug solution that couples both software debug (software execution with the embedded processor) and low-level hardware debug, through the embedded system SPI port. Data generation is greatly simplified too, since the SPI exerciser can be coupled to any software or simulation used as a stimulus source.

▸ **Memory:** when lots of data must be analysed or when a tool must generate long data runs (e.g. stream samples to a DAC, a DSP or a CODEC), PC memory is largely available and inexpensive.

▸ **Compactness:** during embedded system development, more expensive and encumbering equipment like logic analysers are not always available since they are most often shared by

several engineers. It is also very handy to be able to place a single small device next to your laptop in the design room...

▸ **Price:** because they limit the 'specialised' hardware and their performance to the strict necessary and benefit from your standard PC for memory and data processing power, you'll often get a PC instrument for a better price than its bench-top general purpose equivalent.

Productivity matters

The right solution for debugging and testing SPI must make the most of your time during test and debug – in other words, help you validate your embedded system faster and help you find and correct bugs faster.

Embedded system development involves a wide variety of skills and very often a team of software and hardware engineers with their own specialties and own habits. Being really productive during test and debug does not necessarily mean choosing the most powerful oscilloscope because it has got the highest sampling rate. In many cases, you'll end up with costly equipment that engineers won't use because it is not available or because it does not really fit the test and debug case or because the engineers *are not familiar with it.*

Embedded system development *is difficult* because it mixes so much potential software and hardware issues. Even an embedded software developer can end up checking the system at signal level *because it impacts the software he is writing.*

PC-based instruments offer a good match for embedded system designer who are not used to playing with oscilloscope or logic analysers or any other bench-top instrument located in the lab.

▸ They usually provide a familiar programming interface, in C/C++, Visual Basic or any other standard programming language.

▸ They offer a good continuity between the design environment and the test and debug environment.

▸ They can better interact with software debug, since performed from the same, familiar PC environment.

▸ They usually focus on specific tasks ("SPI debug").

▸ They are priced so that each engineer can afford one on his/her desk, next to his/her laptop. So, the embedded software engineer won't need to go to the lab, export his files or wait for a scope or a LA to be unoccupied to start debugging at low level.

▸ Combined SPI exerciser / analyser offer multiple functions and many options that fit **both** embedded system **stimulation** AND embedded system **analysis**.

Testing and debugging times are increasingly important in the total design process, because of the growing complexity of embedded systems. Efficient test and debug strategy heavily depends on the skills and intelligence of the engineers who perform it. If your engineer is a software specialist, he'll do wonders with a PC. PC instruments will just allow him/her keep on using its favourite PC environment, even for low-level tasks like SPI test and debug...

About the author
Frédéric Leens is Sales and Marketing Manager at Byte Paradigm.
He can be reached at: frederic.leens@byteparadigm.com

Revision 1.00 – 19-Jun-08   **9/9**
Byte Paradigm Products are available from:
The Debug Store, Berwyn House, Carrog, CORWEN, LL21 9AT, UK www.TheDebugStore.com