Hard Real-Time Microcontroller For Embedded Applications

Keith Prettyjohns, CEO
Innovasic Semiconductor
3737 Princeton NE,
Albuquerque, NM 87107
Phone: 505-883-5263
Fax: 505-883-5477
Email: keith@innovasic.com
www.engineersbestfriend.com

Introduction

Many embedded applications have to deal with real-time control. Hard real-time control applications, where determinism, latency and jitter are critical to performance, are extremely challenging. In today's design environment, it is the combination of the microcontroller, Real-Time Operating System (RTOS) and the application firmware that determines the performance, reliability and ultimately the success of a new embedded product. Much emphasis has been placed on improving RTOS performance, but very few advances have been made in the microcontroller architecture to support hard real-time performance. The usual solution to improve performance has simply been to increase the clock speed with its resulting increase in power consumption and design complexity.

Innovasic Semiconductor has taken a radically new approach to this problem. By redefining boundaries between silicon and software, a new architecture has emerged that greatly eases the design and test of hard real-time control applications. This architecture brings the additional benefit of enabling higher reliability and even safety-critical applications.

Figure 1. The fido microcontroller packaged in a 208 PQFP

Hard Real-time Microcontroller Architecture

Innovasic has named this new microcontroller family, **fido** – an acronym for flexible input, deterministic output. fido's architectural design was driven by the fundamental fact that silicon is cheap, and embedded software development is expensive. The fido architecture redefines the boundary between hardware and software, shifting functions traditionally handled in software into the chip. This shift provides designers of embedded systems with a special-purpose tool configured to meet the latency, jitter and high-reliability demands of real-time applications.

Although the fido architecture is completely new, it has been designed to execute a modified CPU32+ (68000) compatible instruction set. The CPU32+ is a well-proven instruction set that has all the bugs worked out, leaving a very mature and reliable tool chain legacy. This instruction set not only provides a reliable tool chain for the fido family, but it has been optimized for real-time and safety-critical performance.

The fido1100 is the first product in the fido family of microcontrollers (Figure 2), providing deterministic processing, precise context switching and predictable interrupt latency for real-time tasks. The superior deterministic performance allows it to run at a clock rate of only 66 MHz, reducing power dissipation in the final system.

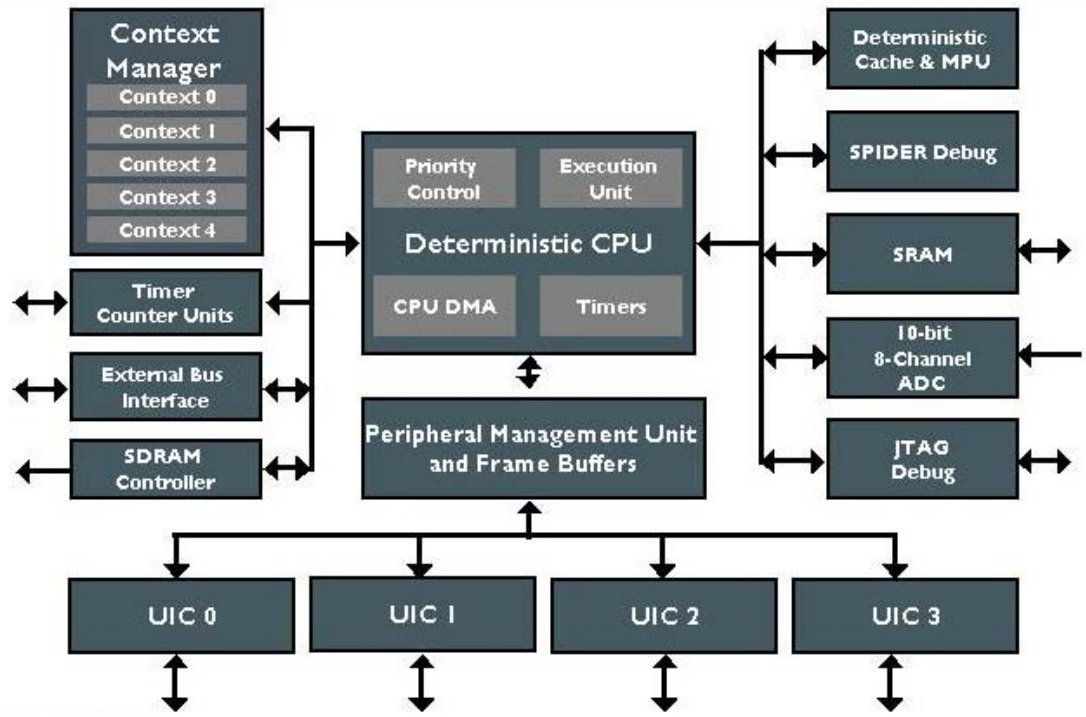

Figure 2. The fido1100 real-time microcontroller is based on a 32-bit architecture

The fido1100 incorporates a context manager for five space- and time-partitioned hardware contexts. The Timer Counter Units consist of two, 32-bit timer control units and a Watchdog and Context Timer. The External Bus Interface is 8-, 16- or 32-bits wide, offering programmable chip selects and a memory controller. Additionally, the chip includes a Deterministic Cache™, a Memory Protection Unit (MPU) and the SPIDER™ Debug interface (described below). The chip also includes a SDRAM Controller, a 6k x 32 High-Speed SRAM and a 10-bit, 8 -channel A/D converter. A JTAG Debug interface is also integrated for emulation and debugging.

The dedicated Peripheral Management Unit (PMU) and integrated data Frame Buffers manage the four UICs™ (Universal I/O Controllers), a flexible peripheral solution that supports numerous interface requirements.

The fido1100 works at a supply voltage of 3.3 V with 5 V tolerant I/Os. The microcontroller is available in industrial temperature grade 208 PQFP and FBGA packaging.

Communication Protocols

The fido1100 incorporates four unique UICs (Universal I/O Controllers), each of which can be programmed to support various communication protocols across multiple platforms. When working with the on board Peripheral Management Unit (PMU) and data buffers, the operation of the interfaces requires little core processor intervention. This feature allows the processor to utilize its bus bandwidth for more important functions rather than managing data traffic.

With the UIC the designer has the flexibility to find the set of peripherals that most closely match the system interface demands. This feature is very important for industrial systems where designers traditionally need to select from a fixed set of peripherals to meet their application requirements.

Each UIC can be programmed to support the following protocol formats.

- Half and full duplex 10/100 Ethernet
    - With MAC address filtering
- Up to 72 Smart GPIO
- 16550 UART
    - Each UIC can be programmed to be a 2-channel, 16550 compatible UART.
- Rx and Tx buffer size can be scaled relative to the interface speed
- CAN 2.0B
- I2C
- Serial Peripheral Interface (SPI)
- Customer proprietary protocols

Each interface supported by the architecture is provided as a standard library element that can be compiled right along with the remaining application code. The UIC approach allows the user the re-use fido for many applications with widely differing I/O requirements.

The Real -Time Functionality of the Microcontroller

Fast Context Switching

In hard real-time applications, a microcontroller must complete tasks within a precise amount of time. For instance, an industrial controller design might require that a set of inputs be read and that the outputs are then set based on the inputs as well as other factors. These tasks must happen on a predictable basis. If not, the entire factory operation is at risk. Unfortunately, most microcontrollers are not designed to focus on time constraints. These are tasks typically handled by the Real-Time Operating System (RTOS).

In a typical microcontroller, there is no inherent difference between real-time, safety-critical, user interface or user application code. The difference is determined by additional software – the RTOS. The RTOS manages when code is run, tasks are switched, the relative priorities and the entire associated overhead. The problem is that the system designer has little control over how the RTOS manages these tasks.

In fido's architecture many of the real-time tasks are handled in silicon, giving the designer better control over them. Figure 3 shows the differences between a typical microcontroller and the fido device.

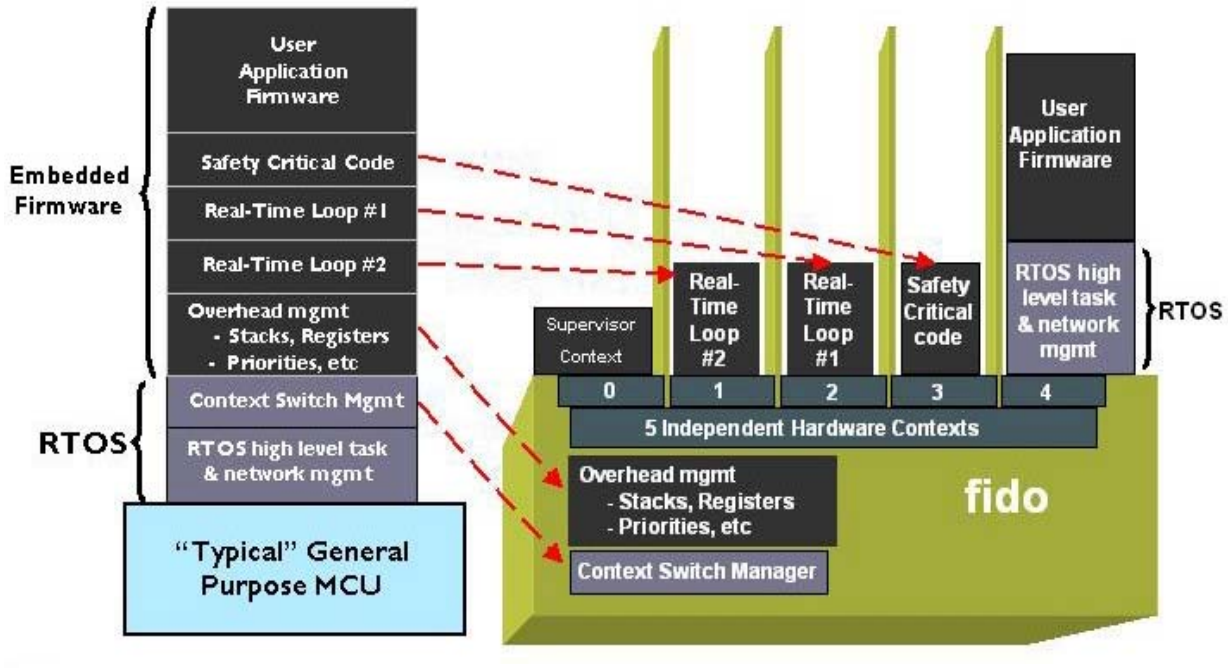

Figure 3. In fido1100, many of the real-time tasks are handled in hardware.

fido has several features that are specifically designed to reduce the software overhead associated with switching and managing tasks. The architecture allocates priorities to one of five space- and-time partitioned hardware contexts. These contexts act like "virtual CPUs" and take only one clock cycle to be switched (Figure 4). This precise control over task switching creates a predictable mechanism for low latency, deterministic performance in the final product.

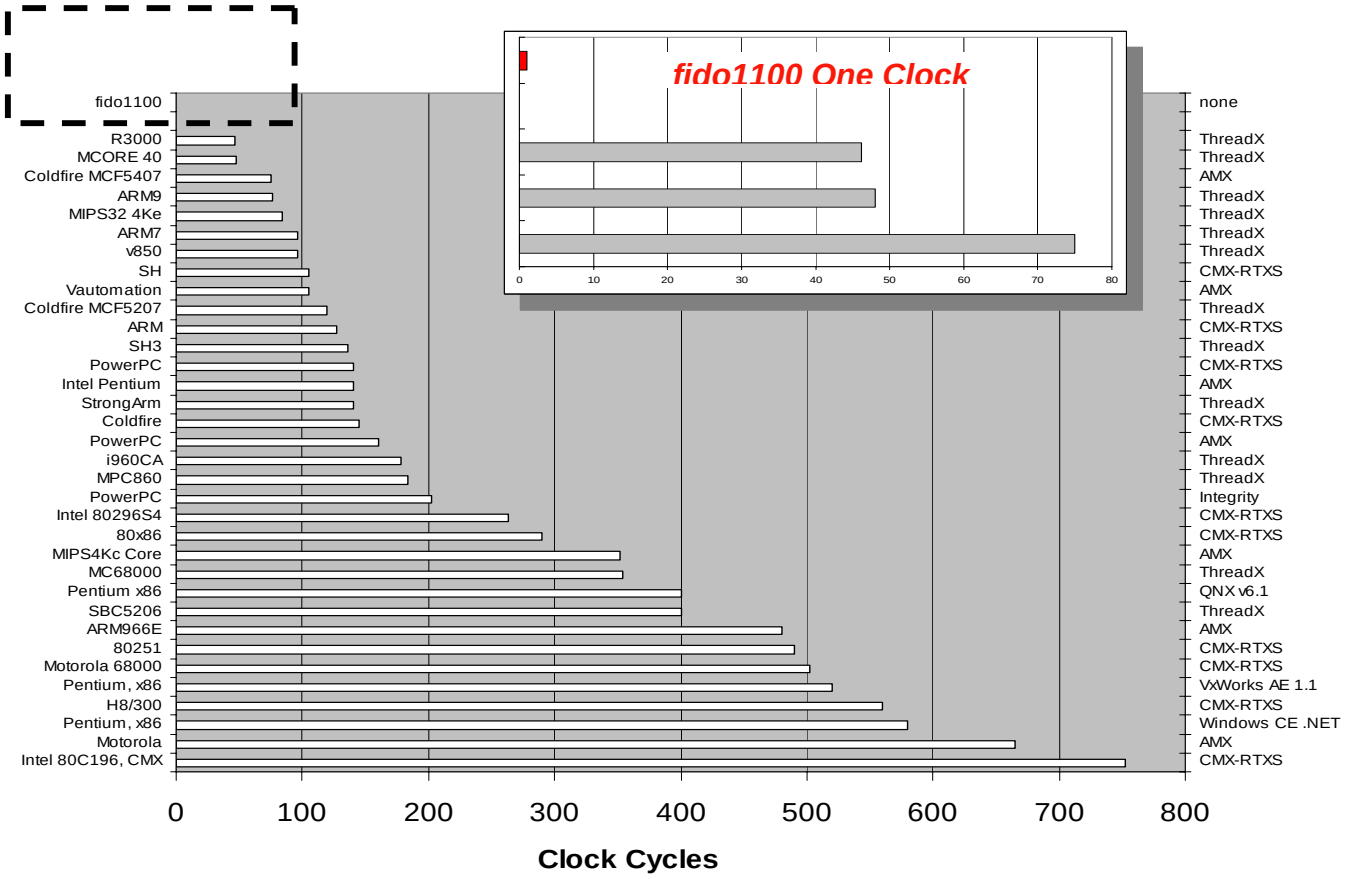

Figure 4. The fido1100 can switch contexts in one clock cycle

The space- and time-partitioning within the chip ensures that one context cannot "step on" another and that no context can ever freeze. This means that the user interface and low-priority code cannot interfere with real-time or safety-critical code. The chip keeps track of which context is being executed, greatly simplifying the debug efforts.

Each context includes its own set of registers that keeps track of the context being processed. When interrupted, the context simply stops where it was processing, switches over to the new context and handles the interrupt (Figure 5). After the interrupt execution, the original context just picks up where it left off. No pushing or popping of registers onto a stack is needed.

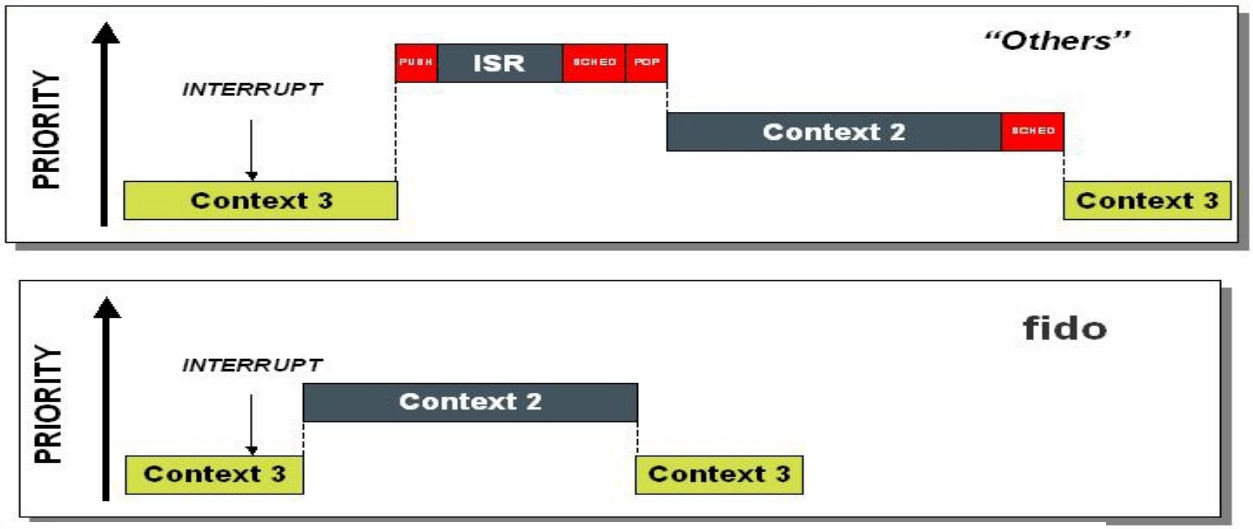

Figure 5. fido1100 offers optimized context switching within one clock cycle.

The hardware also knows when nothing needs to be executed. The processor then goes instantly into auto-sleep mode, again with no code intervention. The programmer only has to worry about the important tasks, and the microprocessor handles the low-power sleep mode. The microprocessor wakes up immediately upon receipt of an interrupt.

The context-aware Memory Protection Unit (MPU) manages the memory requirements of the different contexts without software overhead. The MPU is programmed at boot time so that certain blocks of memory are associated with specific contexts. A block of memory may be read-write to one context, read-only to another, and unavailable to a third, ensuring that blocks of code cannot interfere with each other. This feature is very important in safety-critical applications.

Each context can be allocated a maximum amount of execution time. A context timer is then reset upon entrance to a new interrupt. If a context overruns its allotted time, a fault is generated and the context can be switched. Scheduling can then easily be generated and uncontrolled tasks are protected.

User Programmable "Deterministic Cache™"
Unlike a PC system, tasks in a real-time application must be executed in a predictable manner. Tasks such as measuring control parameters, controlling outputs and motion control have to start at exact regular intervals. The time it takes to begin a task once it has been requested, latency, must be within a tight tolerance. When this latency becomes unpredictable it is called "jitter."

In a typical microcontroller one of the tactics to deal with jitter is to increase the clock speed, reducing the latency time and bringing the jitter to an acceptable level. The side affect of this method is that the increased clock speed produces unnecessary heat. In industrial systems, this additional power is important, given that many of the modules are sealed to prevent the intrusion of foreign substances.

In the fido family, the User Programmable "Deterministic" Cache reduces jitter. Under software control, typically at boot time, the critical pieces of code are stored in the deterministic cache and remain there. When the code needs to be executed, it is in the cache each and every time. No cache misses, and no unnecessary increases in the clock speed are required. As with many other features of the fido architecture, the deterministic cache is also context-aware, producing a consistent response time (Figure 6).

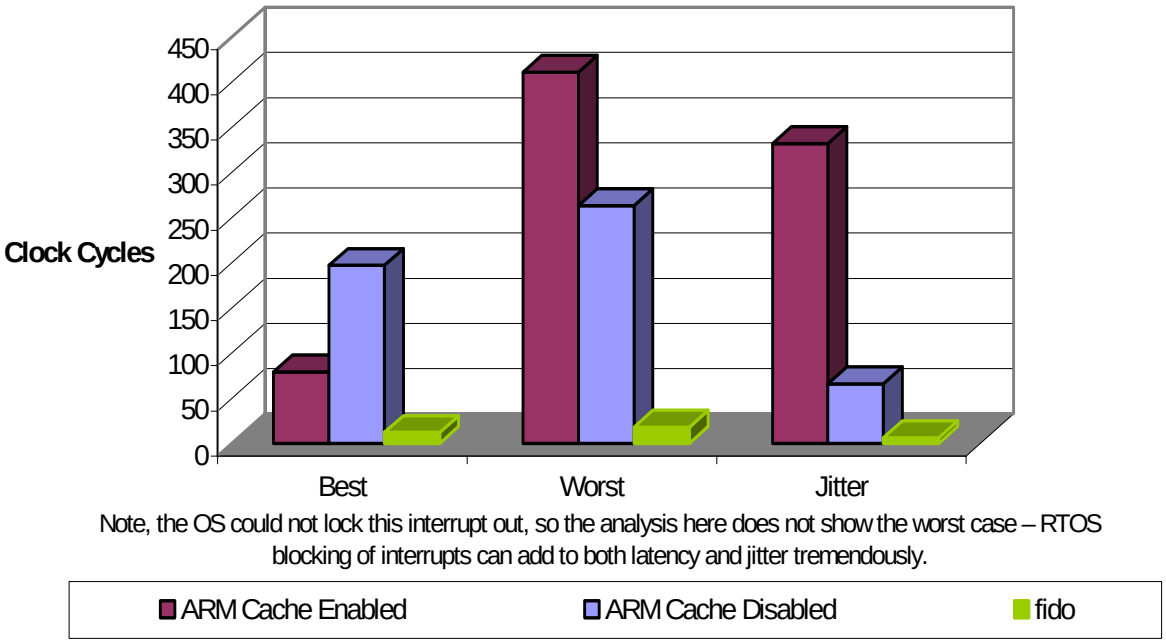

Figure 6. fido1100 Deterministic Cache.

The Challenge of Debugging Hard Real-time Systems

In a recent survey carried out in the United States, 75% of embedded system designers state that they require complex real-time control within their system. Also, 63% of these developers state that debugging the system is their biggest problem.

System debug in real-time applications is very challenging since even one undetected firmware bug could cause intermittent failures or loss of real-time control. In certain applications this could be extremely costly to the end customer, or could even compromise the safety of personnel. For this reason, exhaustive testing and debugging must be conducted.

The increased use of Ethernet connectivity in embedded systems can further impede the task of system debug since an unpredictable flow of TCP/IP traffic may have to be handled by the system, creating an almost infinite number of testing scenarios.

Innovasic has created SPIDER to help solve these problems. With SPIDER, hardware features have been designed into the chip so that debug is greatly simplified. Its rich features clearly and predictably point the designer to the bug, speeding the debug time and reducing the time to market (Figure 7). SPIDER provides a debug environment with a level of observability and controllability that is unmatched by any other commercial solution.

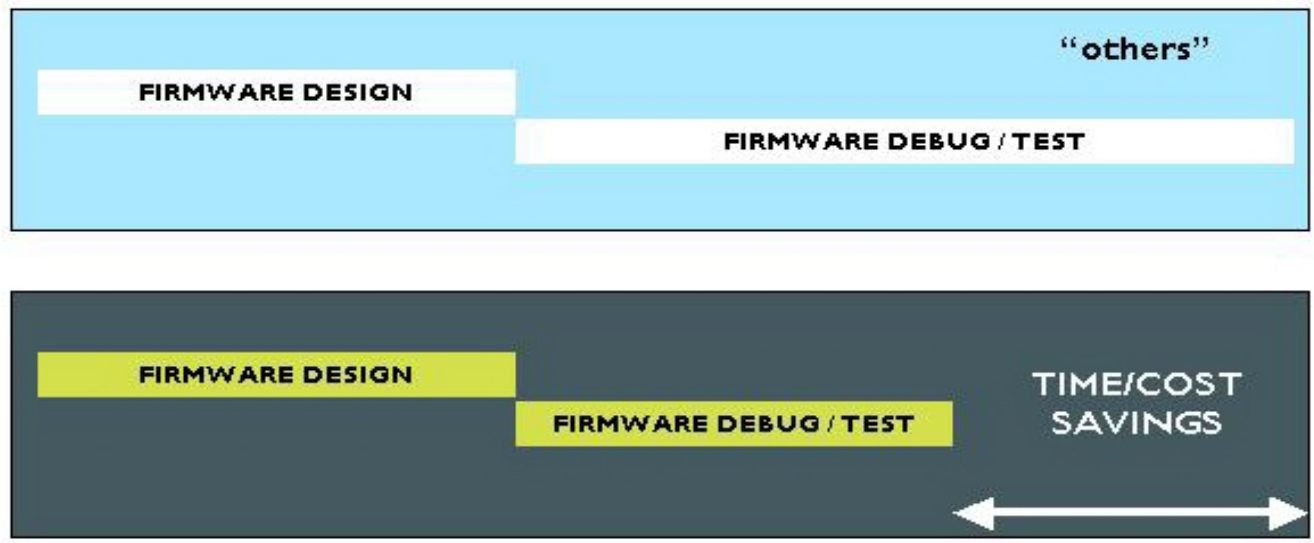

Figure 7. SPIDER reduces debug time and test costs.

In a typical microcontroller, software tools insert trace operations into the code, set up break- and watch-points or even emulate single-step debug. The problem is that the instrumentation of code substantially changes its behavior, making it difficult to identify a bug. The fido1100's SPIDER uses silicon gates within the chip to generate trace data, set break- and watch-points and perform true single-step debug, thereby eliminating the need for code instrumentation.

SPIDER's trace can be written out to system memory. This feature provides the engineer with several options. For example, a logic analyzer can be hooked up to capture the data and timing information for later analysis. Furthermore, some blocks of internal or external memory can be dedicated to capturing the data.

SPIDER also allows the user to set breakpoints to trigger a trace. For example, the designer realizes that something goes wrong in a certain set of circumstances. He can set a breakpoint that will trigger a trace under these circumstances. The trace data can then later be evaluated.

When enabled, the breakpoints can be set to chain, allowing the user to build complex trigger conditions. Through the JTAG port the CPU can be single-stepped from any breakpoint. The user is able to debug code one instruction at a time – all without interfering with the operation of other contexts, or halting processor execution altogether. If desired, the designer can change the state of a register prior to clocking the system, or change the state of a peripheral, one bit at a time, with the easy-to-use SPIDER tools. And, as with all features in fido, the JTAG debug interface is context-aware.

Furthermore, the debug environment provides statistical software profiling that helps to identify critical pieces of code, which can then be moved into the Deterministic Cache.

Development Platform

For a quick evaluation of the revolutionary fido architecture, Innovasic offers a dedicated Evaluation Kit (Figure 8), which includes an industry-standard Eclipse-based toolchain.

Figure 8. The fido1100 Evaluation Kit.

Conclusion

Innovasic has produced a new microcontroller architecture, arguably the first completely new embedded microcontroller architecture for more than ten years. This new architecture redefines the boundaries between the silicon and conventional RTOS/application firmware. By taking this new approach, significant benefits can be realized in fast, deterministic context switching, reduced jitter through on-chip context management and a novel "deterministic cache". Finally, with on-chip debug features tied to the architecture, the challenge of debug and system test can be greatly simplified.