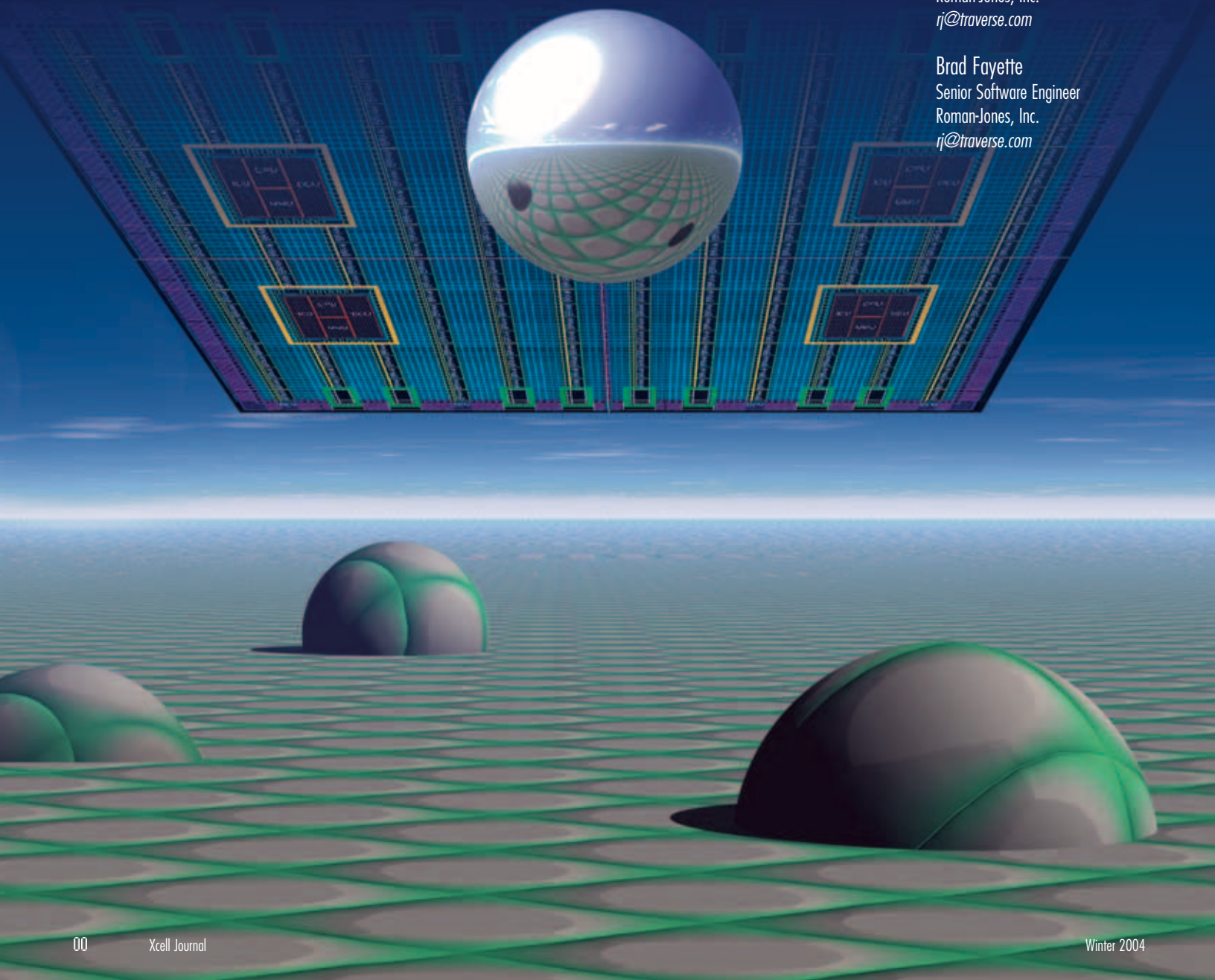# Emulate 8051 Microprocessor in PicoBlaze IP Core

Put the functions of a legacy microprocessor into a Xilinx FPGA.

by Lance Roman
President
Roman-Jones, Inc.
rj@traverse.com

Brad Fayette
Senior Software Engineer
Roman-Jones, Inc.
rj@traverse.com

How do you put a one-dollar Intel™ 8051 microprocessor into an FPGA without using 10 dollars' worth of FPGA fabric?

The answer is emulation. Using software emulation, Roman-Jones Inc. has developed a new type of 8051 processor core built on a Xilinx 8-bit, soft-core PicoBlaze™ (PB) processor. This "new" PB8051 is more than 70% smaller than competing soft-core implementations – without sacrificing any of the performance of this legacy part. The PB8051 is a Xilinx AllianceCORE™ microprocessor built through emulation.

The Legacy of the 8051

The Intel 8051 family of microprocessors – probably one of the most popular architectures around – is still the core of many embedded applications. This processor just refuses to retire. Many designers are using legacy code from previous projects, while others are actually writing new code.

The 8051 architecture was designed for ASIC fabric. It is not efficient in an FPGA, resulting in excess logic usage with marginal performance.

FPGA microprocessor integration is a solution for older 8051 products undergoing redesign to eliminate obsolesce, lower costs, decrease component count, and increase overall performance.

The new FPGA-embedded PB8051 designs allow you to take advantage of existing in-house software tools and your own architecture familiarity to quickly implement a finished design. The integrated PB8051 can be customized on the FPGA to exact requirements.

Processor Emulation

Programmers have used microprocessor emulation for many years as a software development vehicle. It allows programmers to write and test code on a development platform before testing on target hardware.

This same concept can be practical when the target microprocessor architecture does not lend itself to efficient implementation and use of FPGA resources.

The features of our PB8051 emulated processor include:

- Smaller Size – Traditional 8051 implementations range from 1,100 to 1,600 slices of FPGA logic. The PicoBlaze 8051 processor requires just 76 slices. Emulation hardware requires 77 slices. Add another 158 slices for two timers and a four-mode serial port, and you have a total of a 311 slices. This is a reduction of more than two-thirds of the FPGA fabric of competing products.

- Faster – At 1.3 million instructions per second (MIPS), the PB8051 is faster than a legacy 8051 (1 MIPS) running at 12 MHz. Compare this to 5 MIPS with a 40 MHz Dallas version or 8 MIPS with a traditional FPGA

This "new" PB8051 is more than 70% smaller than competing soft-core implementations – without sacrificing any of the performance of this legacy part.

implementation – which takes up more than three times as much FPGA fabric as the PB8051. The PicoBlaze processor itself runs at a remarkable 40 MIPS using an 80 MHz clock.

- Software Friendly – You can write C code or assembler code with your present software development tools to generate programs. You can also run legacy objects out of a 27C512 EPROM.

- Easy Hardware – You can use VHDL or Verilog™ hardware description languages to instantiate the 8051-

type core. Hook up block RAM or off-chip program ROM, and you're ready to go.

- Low Cost – The PB8051 is $495 with an easy Xilinx SignOnce IP license.

Architecture of Emulated 8051

As shown in Figure 1, the architecture of an emulated processor has several elements. Each element is designed independently, but together, they act as a whole.

PicoBlaze Platform

The PicoBlaze host processor is the heart of the emulated system and defines the architecture of the PB8051 emulated processor core. It comprises:

- PicoBlaze Code ROM – Block ROM contains the software code to emulate 8051 instructions.

- Internal Address/Bus – The PicoBlaze peripheral bus is a 256-byte address space accessed by PicoBlaze I/O instructions. It allows the PicoBlaze processor to interface with RAM, emulation peripherals, timers, and the serial port. This bus is internal to the core and thus hidden from the user.

- Emulation Peripherals – Not all emulation tasks can be done in software and still meet the performance requirements for your particular application. Specialized hardware assists the PicoBlaze code to perform tasks that are time-consuming, on a critical execution path, or both.

A good example is the parity bit in the PSW (program status word) register that reflects 8051 accumulator parity. This function must be performed for every 8051 instruction executed, and would take several PicoBlaze instruction cycles to perform. It is, therefore, done in hardware.

- Instruction Decode ROM – This is a key emulation peripheral that is used to decode 8051 instructions to set PicoBlaze routine locations and emulation parameters.
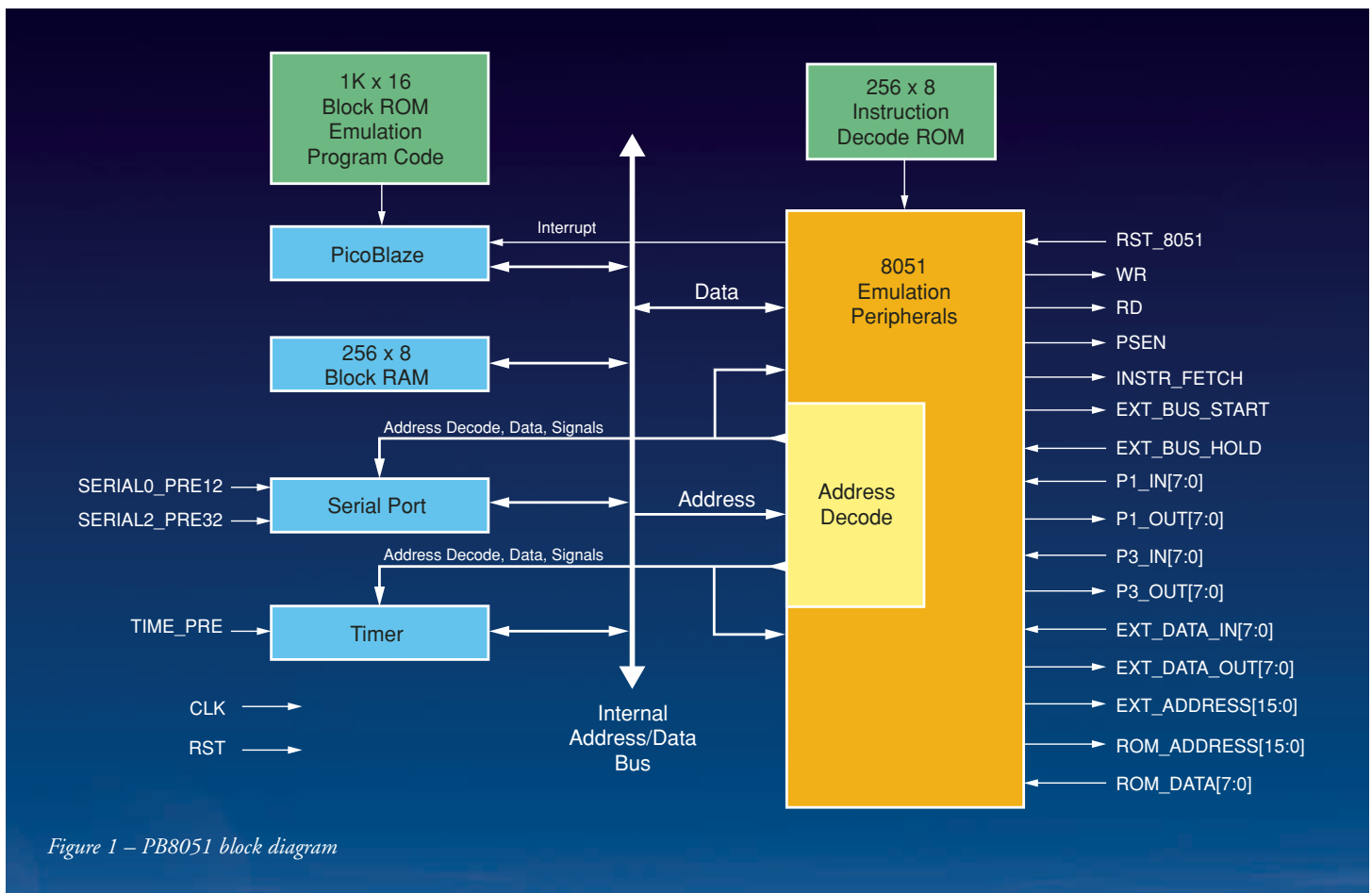
Figure 1 – PB8051 block diagram

- **Address Decode** – A significant amount of the emulation peripheral hardware is dedicated to simple address decode of the PicoBlaze peripheral bus. This address space is for the PicoBlaze processor only and is insulated from the 8051 application.

- **Block RAM** – 256 bytes are available to 8051 internal RAM and some 8051 registers. You can access this block RAM via 8051 instructions.

- **Serial Port and Timer** – The actual 8051 timer and multi-mode serial port were best done in hardware instead of trying to implement these functions in software. Clock prescaling inputs are provided so that these functions can run at a clock rate independent of the emulated system.

PicoBlaze Emulation Software
A 1K x 16 block ROM holds the PicoBlaze code that performs the actual emulation. The emulation program is carefully constructed

in very tight PicoBlaze assembler code, optimized for speed and efficiency.

The emulation program is divided into several segments:

- **Instruction Fetch** – An 8051 bus cycle is simulated to fetch the next instruction from 8051 program memory (64 Kb size), which may be on-chip block RAM or off-chip EPROM (such as the 27C256).

- **Instruction Decode** – Fetched instructions are decoded to determine the addressing mode and operation.

- **Fetch Operands** – Depending upon the addressing mode, additional operands are fetched for the instruction from program memory, internal RAM, or external RAM.

- **Instruction Execution** – An instruction performs the desired operation and updates emulated register contents, including affected 8051 PSW flags.

- **Scan Interrupts** – This function determines if an interrupt is pending, and if so, services it. An interrupt window is generated at the end of every emulated instruction when required.

In addition to PicoBlaze code, Java™ software utilities process symbols taken from PicoBlaze listings (.LOG files) into a ROM table. These .LOG files are used to decode 8051 opcodes.

This program also produces the .COE files used by the Xilinx CORE Generator™ system to create PicoBlaze code ROM. All of this is transparent to designers integrating with the PB8051.

User Back-End Interface
What gives the PB8051 its "hardware flavor" is the user back-end interface, where you interface your logic design with the emulated 8051 processor. The back-end interface is part of the emulation peripherals, controlled by the PicoBlaze platform.

What gives the PB8051 its "hardware flavor" is the user back-end interface, where you interface your logic design with the emulated 8051 processor.

Just as the 8051 processor family has many derivatives to define port, functionality, features, and pinout, the back-end interface serves the same function. The PB8051 has a back-end interface that resembles the generic 8031, the ROM-less version of the 8051.

Roman-Jones Inc. customizes back-end interfaces to meet your exact 8051 needs, such as removing an unused serial port or adding an I²C port to emulate the 80C652 derivative.

Designing with the PB8051
Incorporating the PB8051 into the rest of your design is easy, because it comes with reference designs and examples of Xilinx integrated software design (ISE) projects.

Hardware Considerations
Figure 1 illustrates the signal names available on the user back-end interface. A VHDL or Verilog template provides the exact signal names – many of which are already familiar to 8051 designers. A few new types of signals exist, including:

- Pre-scales used by the timer and serial port to set counting and baud rates.

- One-clock-wide read/write strobes to read and write external memory space. There is also a program store enable (PSEN) strobe for the 8051 code memory.

- Bus start and hold signals used to insert wait states for external memory cycles that cannot be completed in one system clock cycle.

The PB8051 is instantiated as a component into your top-level design. You determine if the 8051 program resides in on-chip block RAM or off-chip EPROM. Peripherals can be hooked up to either the P1/P3 port lines or to the external address and data buses. For convenience, the address and data lines for program and data memory spaces are separated, so conven-

tional multiplexer circuitry is not needed.

If your entire design, including the 8051 program, resides on the FPGA, simply set the EXT_BUS_HOLD to "low" to take full advantage of running at clock speed. If you elect to use an off-chip EPROM or have slow peripherals, wait states can be inserted by asserting EXT_BUS_HOLD at "high." One of our reference designs illustrates wait state generation.

Xilinx Implementation Considerations
There are few implementation considerations other than the need to place the PB8051 design netlist into your project directory and instantiate it as a component in your VHDL or Verilog design; ISE software will do the rest. ISE schematic capture is also supported.

Simulation Considerations
We've included a behavior simulation model of the PB8051 for adding (along with the rest of your design) to your favorite simulator. Modeltech and Aldec™ simulators have been tested for correct operation. Post place-and-route or timing simulations follow a conventional design flow.

You will enjoy watching your 8051 instruction execution flow go by on the simulation waveforms. This makes behavioral debugging straightforward, fast, and easy. We've provided a reference test bench with example waveform files.

8051 Software Considerations
To generate your 8051 programs the way you always have, use your favorite C compiler or assembler and linker (if necessary) to produce the same Intel hex format file that you would use to burn an EPROM. You can even use an existing hex file, because the PB8051 looks like a regular 8051 as far as software code is concerned. An Intel hex to .COE utility is included for those designs that put the 8051 software into on-chip block RAM. On-chip program storage provides maximum speed performance.

Test and Debug Considerations
We recommend you use design tools to quickly and easily test and debug your design. The most useful tool will be an HDL simulator, such as Modeltech or Aldec programs. Most problems and bugs can be solved at the behavioral level. For interactive debugging, the Xilinx ChipScope™ integrated logic analyzer has proved to be the tool of choice. At the current time, no source code debugger tools are available for the PB8051.

Designer's Learning Curve
Designers should have some experience in 8051 hardware/software and FPGA design before attempting to consolidate the two. The PB8051 core is designed for ease of use and integration.

Your 8051 hardware and software expertise should include hardware understanding of the part and experience in writing 8051 code using software development tools. The basic design flow using the PB8051 is identical to the normal packaged processor flow.

Integrating the PB8051 onto the Xilinx part is much the same as instantiating a core using the Xilinx CORE Generator tool. If you are familiar with VHDL or Verilog language, and have a couple of Xilinx designs under your belt, you're good to go.

Conclusion
Integrating microprocessors onto FPGAs through emulation, as illustrated with the PB8051, is a viable alternative to a full hardware functional design. The advantage of FPGA fabric savings correlates to reduced parts cost.

Integrating the PB8051 processor into your design yields lower component count, easier board debugging, less noise, and optimum performance of the peripheral/8051 micro-interface, because both are in an FPGA. For more information about the PB8051 microcontroller, visit *www.roman-jones.com/rj2/ PB8051Microcontroller.htm*. ∑