Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

ALTERA'S NEW CPU FOR FPGAS

Nios II Soft Processor Core Offers Alternative to Custom Silicon

By Tom R. Halfhill {6/28/04-02}

Some trends are hard to ignore, like coffee cups getting larger than coffee pots and telephones getting smaller than mouths. More to the point, the cost of spinning a custom chip in standard silicon is soaring, while the prices of FPGAs keep declining. FGPA vendors love this, because the argument for implementing a custom design in programmable logic becomes more compelling every year.

To lure designers away from the clutches of foundries, FPGA vendor Altera took the stage at the recent **Embedded Processor Forum** and introduced Nios II, a second-generation family of 32-bit synthesizable RISC processors. All Nios II cores are intended primarily for integration in system-on-programmable-chip (SoPC) devices—essentially, SoCs in FPGAs. However, they are also suitable for structured ASICs and regular SoCs, especially as a migration path from FPGAs if production volumes climb.

The first three members of the Nios II family are the Nios II/f (fast), Nios II/s (standard), and Nios II/e (economy). They are basically the same processor core with minor variations. Don't, however, confuse Nios II with Nios 2.0, the second version of Altera's original Nios soft processor for Excalibur FPGAs. (See *MPR 12/3/01-01*, "Excalibur Sharpened By Nios 2.0.") Nios II is an entirely new design that isn't binary-compatible with the older Nios architecture.

At the top of the Nios II family is the full-featured Nios II/f, which has a six-stage pipeline, dynamic branch prediction, a single-cycle 32-bit multiplier, a single-cycle barrel shifter, and optional instruction/data caches. To save gates, the other Nios II cores do without some of these features. The Nios II/s shortens the pipeline from six stages to five, substitutes static branch prediction for dynamic prediction, omits the data cache, and increases the latencies of the 32-bit multiplier and barrel shifter from one clock cycle to three.

The economy-model Nios II/e is even sparser. It drops the 32-bit multiplier, barrel shifter, branch predictor, and both the instruction cache and data cache. It also has a "pipeline" that is, in effect, only one stage long. Actually, the pipeline has six stages, but it can process only one instruction at a time, so the Nios II/e is effectively unpipelined. Table 1 compares the features of the Nios II family.

All three Nios II cores are small. At approximately 1,800 logic cells, the Nios II/f is the largest family member.

Feature	Nios II /f Fast	Nios II /s Standard	Nios II /e Economy
Pipeline	6 stages	5 stages	1 stage*
Multiplier**	1 cycle	3 cycles	—
Barrel Shifter**	1 cycle	3 cycles	—
Branch Predict	Dynamic	Static	—
Instr Cache	0–64KB	0–64KB	—
Data Cache	0–64KB	—	—
Custom ISA	Up to 256 custom instructions		
Logic Cells	~1,800	~1,150	~600

Table 1. Gate-conscious shoppers have three Nios II processor cores to choose from. Notably, all three retain the valuable feature of an extendable instruction set. "Logic Cells" are the number of FPGA gates required to implement the core. *The Nios II/e one-stage "pipeline" actually has six stages but can handle only one instruction at a time. **These features require an FPGA with DSP blocks, such as Altera's Stratix and new Stratix II.

The Nios II/s requires about 1,150 cells, and the Nios II/e requires only about 600 cells. Exact sizes will vary according to the core configuration and the FPGA.

To put these numbers in perspective: the economy-model Nios II/e would occupy only 20% of a Cyclone-family FPGA with 3,000 logic cells, which costs less than $2 in volume; the luxury-model Nios II/f would occupy only 1% of Altera's largest Stratix II, which will have up to 179,400 logic cells when it ships in 4Q04. Naturally, such a large FPGA is expensive. Altera is currently shipping the first member of the Stratix II family, the EP2S60, which has 60,400 logic cells and costs $125 in 25,000-unit volumes.

Translating FPGA cells into NAND-equivalent ASIC gates is always dicey, but Altera estimates that the Nios II/f is roughly equivalent to 25,000–30,000 gates. That size compares favorably with other 32-bit embedded-processor cores having similar capabilities.

Nios II Echoes Classic RISC

It's possible to implement almost any synthesizable processor in an FPGA, and some vendors of soft intellectual-property (IP) cores occasionally license their processors for this purpose. However, a processor core intended for fabrication in silicon isn't an optimal design for an SoPC. Altera's goal with Nios was to create a processor architecture optimized for the unique characteristics of its field-programmable logic, which consists of cells containing four-input lookup tables and a register. In an FPGA, implementing the functional logic of a microprocessor tends to be more expensive than hard-wiring the same functions in regular silicon, but some types of multipliers and memory elements can be less expensive to implement.

Those differences drove Altera to create a basic 32-bit processor architecture that harks back to the classic RISC architectures of the 1980s. Consequently, Nios II has a simple instruction set of 81 instructions and 32 general-purpose registers (GPR), as shown in Table 2. The instructions follow the classic RISC format: three-operand, nondestructive register-to-register operations; two-operand register-immediate operations; and separate load/store operations. There's even a hard-wired zero register: the lowest GPR is a read-only register that always returns zero, just like the zero registers in MIPS and SPARC, two seminal RISC architectures. Although the world may not seem to need yet another RISC architecture, Altera has produced a clean design that's well suited for embedded applications.

Other simplifications include nonvectored interrupts (all interrupts and exceptions jump to the same memory address, where a single handler must sort them out), no memory-management unit (MMU), no FPU, no DSP instructions, and no 16-bit instructions for code compression. However, Altera plans to add an MMU early next year, which would allow Nios II to run embedded operating systems that can manage virtual memory. By then, Nios II will run at least a half-dozen operating systems, including Nucleus Plus, MicroC/OS II, and the embedded Linux 2.6 kernel.

Few other embedded processors in the Nios II class have MMUs or FPUs, either, but DSP instructions are more common, and 16-bit instructions are nearly universal. For instance, the ARCompact, ARM Thumb, MIPS16e, and Tensilica Xtensa architectures all have 16-bit instructions in addition to their standard 24- or 32-bit instructions. Nios II omits 16-bit instructions to simplify decoding, but the trade-off is poorer code density. To some extent, Nios II designers can improve code density by defining custom instructions that do the work of several standard instructions.

Keep in mind that the up-front licensing fees for most 32-bit embedded-processor cores range from $100,000 to $1 million or more, whereas a Nios II license for FPGA integration is only $495—and the license is included with Altera's $995 development kit. That huge price difference makes the relative simplicity of Nios II easier to swallow. The money saved on licensing fees will pay for a lot of FPGA chips.

Despite its parsimony, Nios II is hardly the Yugo of processor cores. One outstanding feature of the architecture is an extendable instruction set, something more commonly associated with the configurable processors from ARC International, MIPS Technologies, and Tensilica. Designers can add up to 256 custom instructions to a Nios II processor, and programmers can use the instructions in assembly language or C/C++ functions. In contrast, the previous Nios 2.0 architecture allowed designers to add only five custom instructions.

A customizable instruction set is a potentially powerful feature that can reduce, if not completely eliminate, the performance gap between SoPCs and SoCs. Certified EEMBC benchmarks of other configurable processors show that a few well-chosen custom instructions can dramatically accelerate critical code in application programs. (Unfortunately, Altera hasn't published EEMBC benchmarks for Nios II.)

Nios II has a few other features not always found in similar processors: multiple privilege modes (system and user); optional branch prediction (dynamic or static); and a relatively deep pipeline (up to six stages). Performance is relatively peppy at 1.2 Dhrystone mips per megahertz, which yields 216 mips at its maximum clock frequency of about 180MHz in a Stratix II device. Table 3 compares the Nios II family with similar 32-bit embedded-processor cores.

Note that leading IP vendors ARM and MIPS aren't represented in Table 3—for interesting reasons. ARM declines to license processor cores for FPGA deployment (as opposed to prototyping), fearing the secrets of its IP will leak out. MIPS is willing to license a processor for FPGAs but doesn't encourage the practice, because its cores aren't optimized for programmable logic.

ARC and Tensilica are more open to licensing their processor cores for FPGAs, but both companies say they perceive insufficient interest from customers to justify creating

specially optimized versions of their latest cores. In 2000, ARC did introduce some variations of its ARC3 processor for integration in Xilinx FPGAs, and ARC says it has sold some licenses for that purpose. ARC says it will license its latest ARC 600 and ARC 700 cores for FPGA integration on demand.

Instruction	Description	Instruction	Description
Arithmetic & Logical		**Data Transfer**	
and	32b logical AND	**ldw**	Load 32b word from memory
or	32b logical OR	**stw**	Store 32b word in memory
xor	32b logical XOR	ldwio	Load 32b word from peripheral
nor	32b logical NOR	stwio	Store 32b word to peripheral
andi	16b immed, zero-extend to 32b	**ldb**	Load byte, sign-extend
ori	16b immed, zero-extend to 32b	**ldbu**	Load byte, zero-extend
xori	16b immed, zero-extend to 32b	stb	Store byte, sign-extend
andhi	16b immed, logical shift left to 32b	ldh	Load half-word, sign-extend
orhi	16b immed, logical shift left to 32b	**ldhu**	Load half-word, zero-extend
xorhi	16b immed, logical shift left to 32b	**sth**	Store half-word, sign-extend
add	32b add	ldbio	Load byte from peripheral
sub	32b substract	ldbuio	Load byte from peripheral, zero-extend
mul	32b multiply	**stbio**	Store byte to peripheral
div	32b divide	**ldhio**	Load half-word from peripheral
divu	32b unsigned divide	ldhuio	Load half-word from peripheral, zero-extend
addi	16b signed immed add	**sthio**	Store half-word to peripheral
subi*	16b signed immed subtract	**Shift & Rotate**	
muli	16b signed immed multiply	rol	Rotate bits left
mulxss	Upper 32b multiply, signed	ror	Rotate bits right
mulxuu	Upper 32b multiply, unsigned	roli	Rotate bits left, based on immed
mulxsu	Upper 32b multiply, signed x unsigned	sll	Logical shift left (<<)
Comparison		slli	Logical shift left immed (<<)
cmpeq	Equal (==)	sra	Arithmetic shift right
cmpne	Not equal (!=)	**srl**	Logical shift right (>>)
cmpge	Signed greater-than or equal	**srai**	Arithmetic shift right immed
cmpgeu	Unsigned greater-than or equal	srli	Logical shift right immed (>>)
cmpgt*	Signed greater-than	**Conditional Branches**	
cmpgtu*	Unsigned greater-than	bge	Comp two regs, branch relative
cmple*	Signed less-than or equal	**bgeu**	Comp two regs, branch relative
cmpleu*	Unsigned less-than or equal	bgt*	Comp two regs, branch relative
cmplt	Signed less-than	**bgtu***	Comp two regs, branch relative
cmpltu	Unsigned less-than	ble*	Comp two regs, branch relative
cmpeqi	Comp register, 16b immed, ==	**bleu***	Comp two regs, branch relative
cmpnei	Comp register, 16b immed, !=	blt	Comp two regs, branch relative
cmpgei	Comp register, 16b immed, >=	**bltu**	Comp two regs, branch relative
cmpgeui	Unsigned cmpgei	beq	Comp two regs, branch relative
cmpgti*	Comp register, 16b immed, >	**bne**	Comp two regs, branch relative
cmpgtui*	Unsigned cmpgti	**Control & Miscellaneous**	
cmplei*	Comp register, 16b immed, <=	trap	Generate exception
cmpleui*	Unsigned cmplei	**eret**	Return from exception
cmplti	Comp register, 16b immed, <	break	Breakpoint for debugger
cmpltui	Unsigned cmplti	**bret**	Return from break (debug)
Moves		rdctl	Read control register
mov*	Register-to-register move	**wrctl**	Write control register
movhi*	Move 16b signed immed upper	flushd	Flush data cache
movi*	Move 16b signed immed, sign-extend	**flushi**	Flush instruction cache
movui*	Move 16 signed immed lower	initd	Prime data cache
movia*	Move address into register	**initi**	Prime instruction cache
Unconditional Jumps & Calls		flushp	Flush prefetched instr from pipeline
call	Call subroutine at immed address	**sync**	Complete all instr before load/store
callr	Call subroutine at address in register	nextpc	Store address of next instr in register
ret	Return from call or callr	**nop***	No operation
jmp	Jump to absolute address in register		
br	Branch relative		

Table 2. The Nios II instruction set is fairly straightforward, but it includes a few extras, such as special data-transfer instructions for unbuffered I/O with peripherals. Several instructions were designed to support specific functions in the GNU C/C++ compiler. *Denotes pseudoinstructions derived from other instructions: for example, NOP is an ADD with read-only r0 as the destination register.

Feature	Altera Nios II /f	Altera Nios II /s	Altera Nios II /e	ARC ARC 600	Tensilica Xtensa LX	Xilinx MicroBlaze
Architecture	Nios II	Nios II	Nios II	ARCompact	Xtensa	MicroBlaze
Instr Length	32 bits	32 bits	32 bits	16–32 bits	16–24 bits [1]	32 bits
Configurability	High	High	High	High	High	Low
Pipeline Depth	6 stages	5 stages	1 stage [2]	5 stages	5–7 stages [3]	3 stages
I-Cache	0–64KB	0–64KB	—	0–32KB	0–32KB	0–64KB
D-Cache	0–64KB	—	—	0–32KB	0–32KB	0–64KB
32-bit Multiplier	Yes	Yes	—	Optional	Optional	Yes
Barrel Shifter	Yes	Yes	—	Optional	Optional	Optional
DSP Extensions	—	—	—	Optional	Optional	—
MMU	—	—	—	—	—	—
FPU	—	—	—	—	Optional	—
Branch Predict	Dynamic	Static	—	Static	—	—
Privilege Levels	2	2	2	1	2	1
Core Freq (Max) [4]	140–180MHz	140–180MHz	140–180MHz	n/a	n/a	150MHz
Logic Cells	~1,800	~1,150	~600	n/a	n/a	950
Introduction	May 2004	May 2004	May 2004	2003	June 2004	2001

Table 3. All these 32-bit RISC processors are synthesizable cores suitable for FPGA integration, but the Nios II and MicroBlaze cores are the only ones specifically designed for that purpose. (Xilinx also offers a PicoBlaze synthesizable core, but it's an 8-bit processor.) [1]Tensilica's Xtensa LX has an optional VLIW-like format that allows custom instructions up to 64 bits long. [2]The Nios II/e actually has a six-stage pipeline, but it works like a one-stage pipeline (see text). [3]Xtensa LX pipeline is configurable at design time. [4]Core clock frequencies are for programmable-logic implementations, not standard silicon, and will vary according to the FPGA. The lower range for Nios II assumes a Stratix device; the upper range for Nios II assumes a Stratix II. n/a: not available.

Custom Instructions Boost Performance

No 32-bit processor core synthesized for an FPGA will run faster than 200MHz with today's technology. Therefore, performance is a concern when targeting programmable logic instead of regular silicon, even if the anticipated production volumes are low enough to otherwise justify the strategy. Altera's answer is to let designers optimize the software's critical routines by using custom instructions. If custom instructions aren't fast enough, hardware accelerators implemented in programmable logic can boost performance still further.

Designers can write custom instructions in Verilog or VHDL, then use Altera's User Logic Import Wizard to bring the module into the project. The wizard is part of Altera's graphical processor-configuration tool, *SoPC Builder*. The Import Wizard can automatically define I/O datapaths for any operands and results associated with the custom instruction, and it automatically creates assembly-language macros and intrinsic functions for the GNU C compiler. Programmers can use a custom instruction in a macro or call it as a function in a C/C++ program, passing any operands as parameters. Figure 1 shows how the imported logic for a custom instruction interfaces with the Nios II core.

For even greater performance, designers can implement hardware accelerators in the programmable logic surrounding the Nios II core, which plays to the strengths of an FPGA. Unlike custom instructions, which share the Nios II execution pipeline with standard instructions, hardware accelerators work in parallel with the processor. They can have their own direct-memory access to data memory and their own register files.

To demonstrate what's possible with these options, Altera cites the example of a cyclic-redundancy-check (CRC) algorithm implemented three ways: in software with standard instructions, in software with a custom instruction, and in logic with a hardware accelerator. The standard-instruction software version requires 23.2 million clock cycles to check a 64KB data block on a Nios II. With a single custom instruction, the processor can execute the algorithm in 860,000 cycles, or 27 times faster than the standard-instruction version. With a hardware accelerator, the algorithm executes in fewer than 44,000 cycles, or 529 times faster than the standard-instruction version. Of course, it's a safe bet that Altera has chosen a favorable example for this illustration, but there's no doubt that custom instructions or hardware acceleration can significantly boost the performance of critical code.

Altera's extendable instruction set is a major feature that distinguishes Nios II from the MicroBlaze soft-processor core that archrival Xilinx offers for its FPGAs. (See *MPR 12/5/01-03*, "FPGAs Catch Fire At MPF.") Unlike MicroBlaze, Nios II also has branch prediction and multiple privilege modes for protecting supervisor processes from user-level tasks. MicroBlaze and Nios II aren't really head-to-head competitors, though, because each vendor licenses its core only for integration in its own programmable-logic devices. Customers choosing between Altera and Xilinx on the basis of the capacity and cost of their FPGAs must use the processor core that vendor offers, unless the customer licenses a processor from a third party or rolls a custom processor.

New Alternatives Keep Sprouting

Synthesizing a soft processor in an FPGA isn't the only way to leverage the advantages of programmable logic. Alternatives that provide different degrees of performance and flexibility include hard processor cores integrated in FPGAs and new standard-part processors that include some reconfigurable logic.

For instance, both Altera and Xilinx sell FPGAs with 32-bit RISC processors integrated as prehardened cores, although Altera appears less committed to this strategy than Xilinx is. Altera's hard-core option is the ARM922T, available only in older-generation Excalibur FPGAs, not the latest Stratix and Stratix II devices. Xilinx offers a hardened PowerPC 405 core integrated in the Virtex-II Pro. Both the ARM9

and PowerPC architectures outclass the simple Nios II. As prehardened cores, they can deliver higher performance than a soft core synthesized in programmable logic, and they don't occupy any logic cells on the chip. On the downside, they aren't configurable, either at design time or in the field.

Another alternative was recently introduced by Stretch, a startup with a fresh idea. Stretch has integrated a Tensilica Xtensa V processor core with some programmable logic on a family of chips sold as standard products. These hybrid devices combine the high performance of a 32-bit RISC processor on standard silicon with the flexibility of programmable logic and a configurable CPU architecture. On the downside, Stretch's chips are priced more like mid-size FPGAs than high-volume ASICs ($35 to $100), and they contain less programmable logic than is typically found in a similarly priced FPGA. Even so, Stretch's S5000-family chips present an option not previously available. (See *MPR 4/26/04-01*, "Stretching Performance.")

A waning option is the unique chips sold by Triscend. Since 1998, Triscend had been selling two families of standard-product chips that integrated an embedded-processor core with some programmable logic, primarily for the purpose of adding application-specific function units and peripherals. Triscend's chips used ARM7 or 8051-compatible processor cores, were priced under $15, and competed mainly against microcontrollers. A few months ago, Xilinx acquired Triscend after outbidding ARM. (See *MPR 3/15/04-02*, "Xilinx Reconfigures Triscend.") Since then, Xilinx has engaged a third party to continue supplying and supporting the parts, but only after startling Triscend's customers with an end-of-life notification that required them to place their final orders by September. Xilinx has told *MPR* it acquired Triscend mainly for its proprietary IP, not for its chips.

One drawback of implementing a low-volume SoC design as an SoPC is that rising demand could make the programmable-logic solution too costly. At some point, depending on numerous factors, it becomes cheaper to spin an ASIC or SoC. Prices of FPGAs are dropping about 35% per year (for devices having the same amount of logic), but that may not be a steep enough decline if demand rises too quickly. Altera has two answers for that quandary: its HardCopy conversion program and conventional IP licensing.

Under the HardCopy program, introduced in 2001, the customer pays Altera a nonrecurring-engineering (NRE) fee to convert the FPGA-based design into a smaller, hard-mask version of the chip—a structured ASIC. The converted chip is guaranteed to run at least as fast as the FPGA version, and perhaps faster if the customer targeted the HardCopy model to begin with. The HardCopy conversion process takes about eight weeks and can cut costs by up to 70%, according to Altera. Customers buy the finished chips from Altera.

If the customer wants to port the design to standard silicon for manufacturing at a foundry, Altera offers the Nios II processor core as licensed IP. Fees are negotiable, starting at $25,000 for the license and a $25,000 lifetime royalty cap. The customer is responsible for the NRE costs of porting the design to silicon and for the usual manufacturing costs. Xilinx offers a similar option, in which customers can license the MicroBlaze processor core for ASIC integration, royalty free. The IP-licensing options from Altera and Xilinx are similar to the licensing models of other processor-IP vendors, except they're much less expensive.

Altera's strategy is smart. By offering two ways to go beyond an FPGA-based design, Altera hopes customers will feel more secure about trying a programmable-logic implementation first, rather than spinning an expensive custom chip. With the costs of ASIC/SoC projects soaring into tens of millions of dollars, anything that reduces risk (technical risk and business risk) looks attractive.

For customers, one potential disadvantage of Altera's strategy is that a HardCopy or standard-silicon conversion from a Nios II FPGA-based implementation will be stuck

Price & Availability

Nios II soft processor cores and development kits are available for licensing now. The license fee for FPGA integration is $495 and is included with the $995 development kit. License fees for ASIC integration are negotiable, starting at $25,000, with a $25,000 royalty cap. For more information, see *www.altera.com/niosii*.

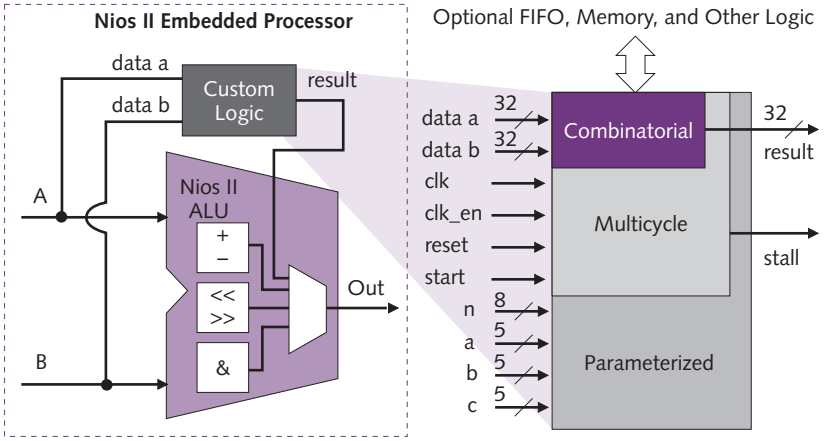

Figure 1. Logic blocks for custom instructions bypass the processor's ALU. The custom block uses combinatorial logic to generate results from the input operands, which can originate from any of the processor's 32 GPRs. Instructions can be single- or multicycle operations; the latter type will stall the execution pipeline until completion.

with the Nios II processor, unless a total redesign substitutes a different core. Although Nios II is an efficient architecture for FPGAs, a chip design targeting standard silicon from the outset would probably use a different processor with more features. On the other hand, a Nios II processor customized with application-specific instructions could conceivably match the performance of an ARC, ARM, MIPS, or Tensilica processor. We encourage Altera to prove this point by submitting the Nios II to EEMBC benchmark tests.

In any case, Nios II is a significant improvement over Nios 2.0, and Altera's HardCopy and IP-licensing options will comfort designers who turn to FPGAs to remedy the rising costs of standard silicon. When vendors like ARM and MIPS perceive enough demand to offer competitive versions of their processors optimized for programmable logic, it will be a sure sign the market is tilting in favor of SoPCs—but it's a sign we haven't seen yet.

Perhaps ARM and MIPS are intimidated by the low licensing fees of Altera and Xilinx, which charge hundreds of thousands of dollars less for their processor IP than other soft-IP vendors do. Altera and Xilinx have the luxury of subsidizing their soft IP with sales of FPGAs. Indeed, their processors are really loss leaders to encourage sales of more FPGAs. That strategy gives Altera and Xilinx an economic advantage that's difficult for other soft-IP vendors to beat.