ARM Tunes Piccolo for DSP Performance

Full DSP Module to Add Signal-Processing Capacity to ARM7 Chips in 2H97

by Jim Turley

Addressing a growing chorus of users seeking better digital-signal-processing performance, ARM has composed its own signal processor. Named Piccolo (Italian for small; apparently no suitable human appendages were left), the new DSP core will become yet another optional piece of the modular ARM portfolio. Working in concert with an ARM7 core, Piccolo should boost ARM's fortunes with makers of wireless devices, PDAs, modems, and disk drives.

ARM architect Dave Jaggar described Piccolo at last month's Microprocessor Forum. Never a vendor to adhere blindly to convention, ARM developed an unusual new DSP core while preserving the traditional ARM merits of low transistor count, small die size, and modest power consumption. The Piccolo core should be available in 10–12 months, debuting in application-specific devices from an unnamed vendor, possibly in cellular phones or pagers.

Piccolo does not replace the ARM core but rather works in parallel with it, expanding a chip's DSP capabilities. The concept is not new and is reminiscent of Hitachi's SH-DSP *(see* **091603.PDF***)* and Motorola's 68356 *(see* **080802.PDF***)*. TI has also combined its TMS320C54x DSP with an ARM7 core.

Register File Backed with Reorder Buffer

Piccolo is an autonomous digital-signal processor sharing some resources with an ARM7-based chip. The DSP executes its own instruction set (which is incompatible with ARM code), uses its own registers, and follows its own control path. The ARM and Piccolo cores communicate solely through a pair of input and output buffers. The ARM core orchestrates overall chip control as well as accessing operands in memory, while Piccolo concentrates on signal-processing loops.

Piccolo has its own register set, independent of ARM's, as Figure 1 shows. The register set is nominally orthogonal, with sixteen 32-bit registers variously referred to either as d0–d15 or as a0–a3, x0–x3, y0–y3, and z0–z3. The upper and lower halves of all sixteen registers can alternatively be addressed as "half registers" for 16-bit operations. The first four registers, a0–a3, double as 48-bit accumulators for some instructions, notably multiply-accumulate.

Piccolo's register file is logically accessible from ARM code, although access is physically mediated by the input and output buffers. The registers are addressed with LDP (load to Piccolo) and STP (store from Piccolo) instructions, which are simple variations of existing ARM coprocessor instructions. These transfer up to 16 bytes of data between Piccolo registers and memory without passing through the ARM register file. Piccolo also has three special registers that hold ID information, status, and the DSP program counter.

The input buffer acts something like a data cache for the Piccolo registers. It is organized as eight 32-bit entries but can double as sixteen 16-bit entries as well.

Input, Output Buffers Decouple Two Cores

The input buffer is not simply a FIFO from ARM to Piccolo. Instead, it is managed more as a reorder buffer, or ROB. Each entry in the ROB is tagged with its destination register as the ARM core loads it. When a Piccolo register is freed, that register is automatically refilled with an entry from the ROB tagged for that register. If more than one ROB entry is tagged for the same register, Piccolo selects the oldest entry. (The oldest entry is identified by its position in the ROB.) If the same register becomes free again, Piccolo selects the next-oldest entry from the ROB destined for that register, and so on.

The ROB allows ARM code to fetch DSP data or coefficients from memory in whatever order is most convenient while allowing the DSP code to consume the items in what-

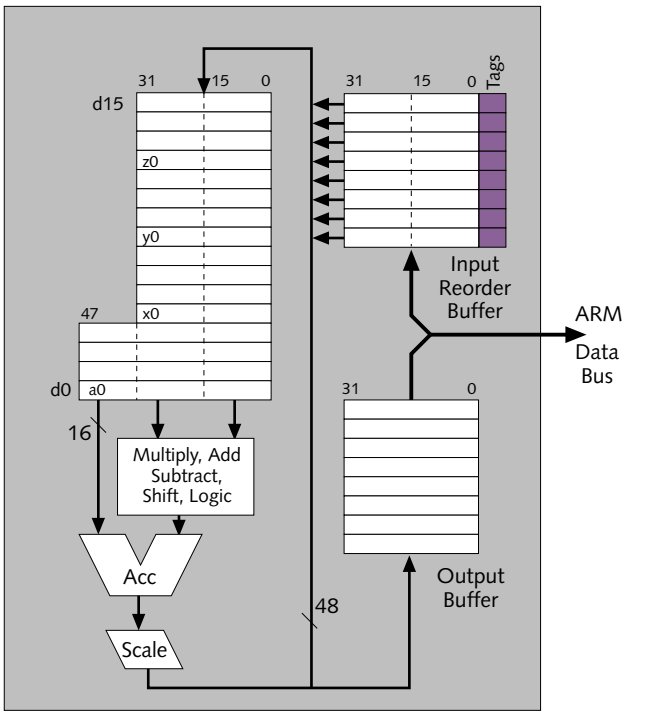


Figure 1. Piccolo includes its own register file and loads and stores its data from two buffers attached to the ARM data bus. The ARM core loads and empties the buffers with coprocessor instructions.

ever order they are required. For example, ARM code can load a block of four data points from memory with a single LDP instruction and write them to the ROB, then load a set of four coefficients from a different area of memory. Piccolo can then pull the data and coefficients from the ROB in pairs.

The output buffer is much simpler than the input ROB. It is a fairly straightforward 8 × 32-bit FIFO that queues results until ARM can retrieve them. The only special feature of Piccolo's output buffer is that it concatenates two consecutive 16-bit entries into a single 32-bit word. Thus, a single 16-bit result will not "fill" the first entry in the FIFO; a second 16-bit result is needed before ARM can remove the entry from the output buffer.

Input, Output Buffers Mediate Communications

If Piccolo tries to read from an empty input buffer, it will stall until ARM loads it. Conversely, if the input buffer is full when ARM tries to load it, ARM will stall until Piccolo removes at least one entry. The same is true of the output buffer: either Piccolo or ARM will stall if the buffer is full or empty, respectively. Jaggar optimistically described this condition as an opportunity to save power.

There is no signaling between ARM and Piccolo; the two cores are interlocked only through these two buffers. For instance, Piccolo cannot interrupt or otherwise indicate to ARM that the output buffer is full. It is the programmer's responsibility to synchronize integer and DSP code so that ARM services Piccolo's data-input and -output needs.

Piccolo is unaware of interrupts, exceptions, or faults that may occur while the ARM chip is running. Hardware interrupts, for example, do not stop Piccolo processing. If, during interrupt processing, the ARM CPU neglects to empty Piccolo's output buffer, Piccolo simply stalls until the output buffer is emptied again. Likewise, Piccolo stalls if its input buffer runs dry.

Piccolo fetches its own DSP code from on-chip or off-chip memory, which is cached in a private instruction cache. This cache is separate from any cache the ARM core might also have. Piccolo's cache is fully associative and holds at least 64 instructions, arranged in four or more 64-byte lines. Although this cache is small in size, ARM claims most common DSP inner loops fit easily into this space.

Programmers wishing to optimize performance can directly manipulate Piccolo's cache. By writing an address to a Piccolo coprocessor register, ARM code can force a Piccolo cache miss at that address, loading a line of 16 DSP instructions. Using this technique, control code can preload DSP code into some or all of Piccolo's cache. To launch DSP code, ARM writes to Piccolo's program counter.

Reordered Refill Reuses Registers

With Piccolo's limited resources and the data-hungry nature of most DSP algorithms, it is important to keep Piccolo's register file as busy as possible. After the contents of a register are used for the last time, Piccolo automatically reloads that register with new data from the ROB. Determining when a value is used for the last time is the programmer's responsibility.

Piccolo's instruction format includes a "refill" bit for each source operand. Programmers must flag the last use of

Mnemonic	Description	Mnemonic	Description	Mnemonic	Description
Arithmetic		**Arithmetic**		**Multiplication**	
ADD	Add	ADDADD	Parallel add, add	MUL	Multiply
ADC	Add with carry	ADDSUB	Parallel add, subtract	SMUL	Multiply, saturating
ADDA	Add and accumulate	SUBSUB	Parallel subtract, subtract	MULA	Multiply-accumulate
CAS	Add conditional	SUBADD	Parallel subtract, add	MULS	Multiply-subtract
SADD	Add, saturating	CMNCMN	Parallel compare (add/add)	SMULA	Multiply-accumulate, saturating
SUB	Subtract	CMNCMP	Parallel compare (add/sub)	SMULS	Multiply-subtract, saturating
SBC	Subtract with carry (borrow)	CMPCMN	Parallel compare (sub/add)	SMLDA	Multiply-accumulate, doubling
SUBA	Subtract and accumulate	CMPCMP	Parallel compare (sub/sub)	SMLDS	Multiply-accumulate, dbl, saturate
SSUB	Subtract, saturating	**Logical**		**Special**	
RSB	Reverse subtract	AND	Logical AND	EMPTY	Mark register for refill
SRSB	Reverse subtract, saturating	ORR	Logical OR	ZERO	Clear selected registers
RSC	Reverse subtract with carry	BIC	Bit clear (Logical AND NOT)	OUTPUT	Force registers to output FIFO
CAS	Conditional add/subtract	EOR	Logical exclusive-OR	**Loop Constructs**	
CASC	Conditional add/subtract w/carry	TST	Logical AND, no register write	RMOV	Set register-mapping parameters
CMP	Subtract w/o register write	TEQ	Logical EOR, no register write	REPEAT	Initiate loop
CMN	Add w/o register write	ASL	Shift left, arithmetic	NEXT	Terminate loop
SABS	Absolute value, saturating	ASR	Shift right, arithmetic	**Branch & Miscellaneous**	
MIN	Find minimum absolute value	LSR	Shift right, logical	Bcc	Branch conditional
MAX	Find maximum absolute value	ROR	Rotate right	SELcc	Select (conditional move)
MINMIN	Parallel minimum, minimum	MOV	Copy 16-bit immediate	SELTTcc	Parallel select, true/true
MAXMAX	Parallel maximum, maximum	CLB	Count leading bits	SELTFcc	Parallel select, true/false

Table 1. The Piccolo DSP core adds an entire new instruction set to ARM processors. A number of parallel instructions can operate on two 16-bit values simultaneously.

each register so a new value can be obtained from the ROB. The last use is specified in the assembler syntax by affixing a caret (^) to the register name. After the last use, hardware moves the oldest pending entry in the ROB to that register in the next cycle. A one-cycle load-use penalty is exacted if the immediately subsequent DSP instruction depends on the new value.

Instruction Set Parallels ARM

Piccolo instructions are encoded differently from ARM instructions; because they are executed in different pipelines, they don't have to be compatible. Like ARM, Piccolo instructions offer in-line operand scaling and optional flag updating. Unlike ARM, conditional execution is not supported; Piccolo uses conditional-branch instructions instead. Taken branches face a three-cycle penalty.

The current core implementation has four pipeline stages (fetch, decode and register read, execute, and write-back). At 40 MHz, a Piccolo-equipped ARM processor can execute 40 MOPS. Counting multiply-accumulate as two operations, or using parallel "split" instructions, Piccolo reaches 80 MOPS.

The complete Piccolo instruction set is listed in Table 1. The instruction set includes the usual add, subtract, and multiply operations (no divide), with both saturating and nonsaturating versions. Piccolo also has a number of "split" functions that work on two 16-bit operands at once, similar to MMX and other new media extensions. Just as with ARM, integer division is a stepped, iterative process that takes 3–19 cycles for 32÷16-bit division.

Multiply-accumulate, add-accumulate, and subtract-accumulate are also supported and use one of the 48-bit accumulators, a0–a3. The large accumulators avoid any possibility of overflow on 16 × 16-bit operations. With a 48-bit accumulator, even repeated multiply-accumulates would not overflow in less than 65,535 iterations, an ample number for any practical application.

The CLB (count leading bits) instruction reports the number of bits by which the source must be shifted left until its two most significant bits differ. That count can then be used for normalizing other numbers.

Every instruction has, conceptually, two possible destinations. Depending on the assembly syntax, Piccolo can store results to the register file, the output buffer, both, or neither. (This last combination, while seemingly pointless, is useful for creating nondestructive comparison or test instructions.)

Writing the results to the register file is the usual case when output values will be reused as input in a subsequent calculation. When the output value will not be reused, it can be written directly to the output buffer for later removal by ARM. Storing results to both the register file and the output buffer allows ARM code to retrieve intermediate results. To specify that the result should be stored to the output buffer, the assembly syntax adds a caret (^) to the destination.

Price & Availability

The Piccolo DSP core has been licensed to a number of semiconductor companies. The first Piccolo-equipped processor is expected to ship in 2H97. For information, contact ARM (Cambridge, U.K.) at 44.1223.400.400, fax 44.1223.400.410 or in the U.S. at 408.399.5190, fax 408.399.8854 or visit the Web at *www.armltd.com.*

Register Remapping Rotates Resources

Like most DSPs, Piccolo has a loop-construct primitive. The REPEAT instruction takes two arguments: the size of the loop and the intended number of iterations, which can be a register value. REPEAT instructions can be nested four deep.

The looping construct also supports a feature whereby Piccolo's registers are remapped after each pass through the loop. The purpose of this unusual feature is to allow DSP loops to operate on a series of coefficients without explicitly addressing different registers. Using the loop-remapping option, up to eight registers shift one, two, or four apparent positions in the register file, modulo the number of registers being remapped. After the final pass through the loop, the registers are back in their original positions.

The code fragment in Figure 2 illustrates how both the register remapping and the ROB refill are used on an FIR filter. The code multiplies a series of data points (d[*n*]) with an eight-element vector (c[0]–c[7]). Because each data point is multiplied with each of the vector elements, a loop with modulo-8 remapping will automatically access each element in turn. Also, once a data point has been fully accumulated, its register is no longer needed and can be refilled with new data from the ROB. The net result is that Piccolo needs to

```
REPEAT #N/4, x++ n4, y++ n4

  MULA    A0, X0.L^, Y0.L,  A0
  MULA    A1, X0.H,  Y0.L,  A1
  MULA    A2, X1.L,  Y0.L,  A2
  MULA    A3, X1.H,  Y0.L^, A3
```

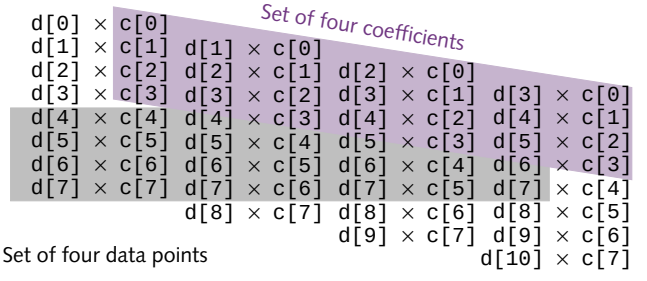


Figure 2. In this example of an FIR filter, a sequence of data points is multiplied with each of eight coefficients in turn. For every fourth pass through the loop, one data point and one coefficient can be replaced with new values. Piccolo's register rotation and register reloading handle both these tasks without explicit code, while ARM handles memory accesses.

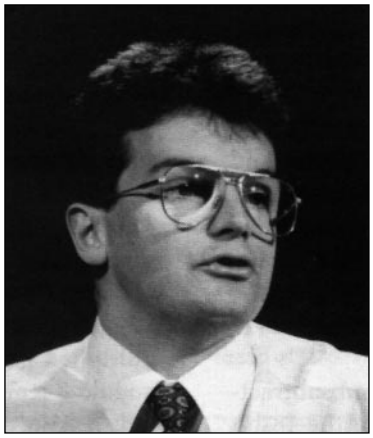
load only one new data value on each pass through the loop, which will probably be waiting in the ROB.

Piccolo Debut Set for Mid-1997

During his presentation, Jaggar confirmed that Piccolo development is complete and that some number of both current and unannounced ARM licensees have signed on. Those companies have not been identified, but at least one is said to be working on a Piccolo-based chip for release in mid-1997. The first implementation will be with an ARM7TDMI (that is, an ARM7 core with the Thumb, hardware multiplier, and emulator/debugger modules).

The company claims Piccolo requires somewhat less than 1.5 mm^2 of silicon in a 0.35-micron three-layer-metal process. A basic ARM7 core needs about 2.2 mm^2 in the same process, making Piccolo about two-thirds as large as ARM7.

In a 0.6-micron process, parts are expected to reach 40 MHz, about the same speed as ARM7 in the same process. In 0.35-micron technology, which more aggressive ARM vendors like VLSI are now delivering, Piccolo should hit 66–80 MHz. Jaggar suggested an updated Piccolo with a longer pipeline will reach 120 MHz. This version might be mated to an ARM8 or StrongArm core.



MICHAEL MUSTACCHI

Dave Jaggar of ARM conducts the first disclosure of the Piccolo DSP at the Microprocessor Forum.

DSP-Like Performance

Piccolo has not been fabricated, so no definite performance figures are available. ARM has simulated Piccolo running at a variety of clock speeds and compared its performance with that of more conventional, albeit aging, parts from AT&T, Motorola, and TI. A 66-MHz Piccolo edged out the 1627, the 56002, and the 320C52 running at 40–70 MHz.

Granted, the performance of Piccolo was simulated and the three competitors chosen don't represent the state of the art, but ARM's version seems to hold its own. On the other hand, TI, Motorola, AT&T, and other vendors are all shipping parts now, with 100-MHz and faster DSPs in the works. Before Piccolo ships, a performance gap may open up.

ARM's results indicate that customers with a moderate requirement for DSP speed need not give up performance to use Piccolo. Many embedded DSP applications also require a conventional integer processor for control flow, user interface, protocol handling, and other features. ARM's CPU/DSP combination may be more convenient and more economical than separate parts. It may be more accurate to characterize a Piccolo chip as a DSP with CPU capabilities rather than a CPU with DSP capabilities.

Two Cores Need to Cooperate

Although DSP performance looks good, ARM performance suffers terribly when the DSP is active. According to ARM's own figures, from 33% (during GSM loops) to 87% (during FFTs) of ARM's bandwidth is used to feed Piccolo. In either case, there's little time remaining for a task switch or for meaningful processing.

In an ARM/Piccolo duet, the two cores essentially take turns running. The CPU can run at full bandwidth when the DSP is idle, or vice versa. This characteristic is a good match with a large subset of application requirements. A cellular telephone, for example, might depend on the ARM for the user interface and for number recall while dialing, then switch to heavy DSP usage while the call is in progress. Disk drives provide another applicable example: interface protocols (SCSI, IDE, etc.) can be handled by ARM code while data recovery is handled by the DSP.

Piccolo is less well suited for a handheld organizer or PDA; while Piccolo will make a fine modem or wireless controller, the remaining ARM bandwidth might frustrate users. Consumers will generally want to continue using their system while the modem or wireless link is active, and Piccolo would sap most of the ARM's strength in such circumstances.

Different Bottlenecks

Piccolo bears similarities to both the 68356 and the SH-DSP. Like the Motorola part, Piccolo-equipped chips will execute two separate instruction streams on two different cores at the same time. Unlike the 68356, however, the CPU and DSP cores share an address and data bus, necessitating a lot of cooperation between the two.

The SH-DSP binds the two cores more tightly, as does ARM's approach. In fact, in the Hitachi design, the CPU and DSP execute from a single instruction stream and are not independent at all. Each 32-bit instruction is split, with half going to the DSP and half to the CPU for address generation and flow control.

Both the 68356 and the SH-DSP have separate X and Y memories typical of DSPs but not found in Piccolo. ARM chose instead to be clever with its register file, supplying operands from DSP registers, not local memories. Even though Piccolo can access a pair of registers per cycle, it can reload only one from the ROB. A data-intensive algorithm can starve the register file. In the end, Piccolo is cursed with lower operand bandwidth than other DSPs, and this may prove to be its ultimate weakness.

Like other companies, ARM has responded to a growing need to merge signal processing with arithmetic or control-flow processing. As Hitachi and Motorola have found, the combination can be a popular one, as the nature of embedded designs changes and the demand for wireless appliances increases. With its announcement of Piccolo, ARM can end 1996 on a high note. MR