

**NEW INDUSTRY-STANDARD,
REAL-WORLD BENCHMARKS—THE
CUMULATIVE EFFORTS OF 21 μ P AND
DSP VENDORS—GIVE YOU THE TOOLS
TO CONFIDENTLY SELECT A
PROCESSOR FOR YOUR
NEXT EMBEDDED-SYSTEM DESIGN.**



At last: benchmarks you can *believe*

WHEN WAS THE LAST TIME you believed a published benchmark result for an embedded processor? How much credence do you put on the Dhrystone MIPS benchmark when you're trying to evaluate a processor for an embedded application? If you're like most design engineers, you probably answered, "I don't remember" to the first question and "little to

none" to the second. But face it, it's difficult to design a benchmark for a processor in the context of an embedded application. The difficulty stems from the fact that one benchmark cannot effectively represent the variety of embedded applications. The EDN Embedded Microprocessor Benchmark Consortium (EEMBC, pronounced like "embassy") embarked on a mission in April 1997 to turn the art of embedded-processor benchmarks into a science. As a result, you can now more easily obtain an accurate, reliable, and application-based

metric to evaluate a processor's performance (see **sidebar** "The making of a benchmark").

EEMBC's suites of benchmarks reflect real-world applications. The consortium's general strategy is to extract the processor-intensive code segments of the most popular embedded applications. This method allows you to choose the benchmarks that are relevant to the application for which you are designing, rather than benchmarks that attempt to lump results into a single number. To accomplish this task, the members first established separate subcom-

At a glance.....	100
The making of a benchmark	100
The ultimate oxymoron: dependable benchmark scores	102

mittees that target each of the primary vertical-market segments: automotive/ industrial, consumer, networking, office automation, and telecommunications. Each subcommittee then selected the applications within its corresponding market segment; these applications include engine control, digital cameras, printers, cellular phones, modems, and more. EEMBC's members then dissected each of the applications and derived 35 algorithms to develop for the consortium's initial benchmark suites. (For a list of EEMBC's Phase I benchmarks, go to www.eembc.org.) These benchmarks allow you to pick the ones that most closely resemble your application. Unlike synthetic benchmarks, these are real algorithms in real embedded applications (Figure 1).

TESTDRIVING THE BENCHMARKS

The Automotive/Industrial Subcommittee developed benchmarks that encompass applications ranging from motor control to noise cancellation. As a result, EEMBC's members derived benchmark algorithms that range from table look-up and interpolation to FFTs. Regardless of the benchmark, the consortium implements each with a strong orientation to the application from which it is derived.

Use the Table Look-up and Interpolation Benchmark as an example to demonstrate the "real-world" nature of the automotive/industrial benchmarks. (You may find that this benchmark also applies to other types of embedded applications.) This algorithm explores a CPU's ability to handle basic arithmetic and pointer referencing and periodically performs a table look-up

AT A GLANCE

► The EDN Embedded Microprocessor Benchmark Consortium (EEMBC) benchmarks target real-world applications in the automotive/industrial, consumer, networking, office-automation, and telecommunications markets.

► EEMBC's Web site, www.eembc.org, will start in the second quarter of next year to post standard compiler-based benchmark scores and hand-tuned optimized benchmark scores.

► EEMBC has established the EEMBC Certification Lab, which will verify every publicly displayed benchmark score bearing the EEMBC trademark.

to derive an ignition-angle output value from the engine-load and -speed input variables. In a real engine-control application, external engine sensors would derive engine speed by measuring the period between pulses from magnetic pickup-sensing gear teeth on the crankshaft. Sensors measuring airflow through the throttle body derive engine load. For this benchmark, the CPU pulls these variables from test data stored in RAM. A map from Bosch's (www.bosch.com) automotive electric/electronic systems provides the model for the ignition-advance map's style.

In this algorithm, the CPU searches two linear arrays that contain the indices for the axes of a grid by adjacent points stored in the map. Next, the CPU computes the partial interpolations in each axis and uses the

results to adjust the four corner z values of the 3-D grid that the index pairs define. This process results in an interpolated z-axis value that represents the ignition-advance parameter that the program would pass to a subsequent part of the engine-control process and apply to the firing of the igniters or spark plugs. The CPU repeats the process for each pair of input values that it takes from the test data.

The benchmark performs the bilinear interpolation using the following equations:

$$\begin{aligned} dx &= (\text{loadValue} - \text{engLoad}[i]) \times \\ &\quad (\text{engLoad}[I+1] - \text{engLoad}[i]), \\ dy &= (\text{speedValue} - \text{engSpeed}[j]) \times \\ &\quad (\text{engSpeed}[j+1] - \text{engSpeed}[j]), \\ \text{and} \\ \text{outAngleValue} &= \\ &= (1.0 - dx)((1.0 - dy)(z[i][j])) \\ &+ dx((1.0 - dy)(z[I+1][j])) \\ &+ dx(dy(z[I+1][j+1])) \\ &+ (1.0 - dx)(dy(z[i][j+1])), \end{aligned}$$

where i and j are the x and y table indices, dx and dy are the interpolation ratios within the selected grid that contains the desired point that the engLoad and engSpeed input variables dictate and are in the range (0..1), and z[i][j] is an output angle in the table at x=i and y=j.

NETWORKING BENCHMARKS

Similar to the Automotive/Industrial Benchmark Suite, EEMBC's Networking Suite represents a collection of sophisticated real-world benchmarks. The Networking Suite characterizes some of the major functions in the course of servicing an In-

THE MAKING OF A BENCHMARK

When I became EDN's micro-processor and DSP editor more than four years ago, I soon realized that the industry lacked a comprehensive and accurate metric for evaluating processor performance. The Dhrystone benchmark was, and still is, the primary metric that processor vendors use for marketing their products. Although this benchmark provides some basic information about a processor, it provides no resemblance to real-world applications. Furthermore, processor

and compiler vendors have abused and misused this benchmark, rendering it almost meaningless.

So, about 18 months ago, I invited most of the embedded-processor vendors to meet to discuss my plan to develop a benchmark suite based on real applications. They accepted the plan, and we formed the EDN Embedded Microprocessor Benchmark Consortium (EEMBC), which now has 21 members: AMD (www.amd.com), Analog

Devices (www.analog.com), ARC (www.riscores.com), ARM (www.arm.com), Hitachi (www.hitachi.com/index95.html), IBM (www.ibm.com), IDT (www.idt.com), Lucent Technologies (www.lucent.com), Matshushita (www.masca.com), MIPS (www.mips.com), Mitsubishi Electric (www.mitsubishi.com), Motorola (www.motorola.com), National Semiconductor (www.national.com), NEC (www.nec.com), Philips (www.philips.com/home.html), QED (www.qedinc.com),

Siemens (www.siemens.com), STMicroelectronics (www.st.com), Sun Microelectronics (www.sun.com), Texas Instruments (www.ti.com), and Toshiba (www.toshiba.com).

In the first quarter of 1999, EEMBC will release its Phase 1 benchmarks, and EEMBC's members and the embedded industry will be able to judge how the various processor architectures perform with this new industry-standard benchmark suite.

ternet Protocol (IP) packet. The first function assesses the processor's performance in handling a basic IP header and Transmission Control Protocol (TCP) header. The benchmark computes checksums on packets that occupy multiple chained buffers in memory and involves accounting for the discontinuities of changing the current buffer in the chain. This process allows you to examine a processor's ability to perform pointer and numeric calculations in the same inner loop.

The packet flow in the reference design in this benchmark suite begins when the simulated input at the input-network inter-

faces receives packets generated by the code at the beginning of the test (Figure 2). The IP preprocessor scans the queue, finds a packet, dequeues it, and submits it to the main IP process. After the benchmark computes and verifies the IP-header checksum, the benchmark searches the routing table for a route for the destination address

that the header specifies. The destination is either the local machine or another router downstream. If the destination is a local machine, the benchmark enqueues the packet to the TCP routine, which verifies the packet's TCP header for errors, and the test repeats. For packets being routed through the virtual topology for this

(MTU derives from graph theory and is also known as maxflow-minicut in mathematics.) In the path of the packet from source to destination, a transit network may exist that cannot handle IP segments of this maximum size. The router then fragments the datagram according to the IP specification's requirements.

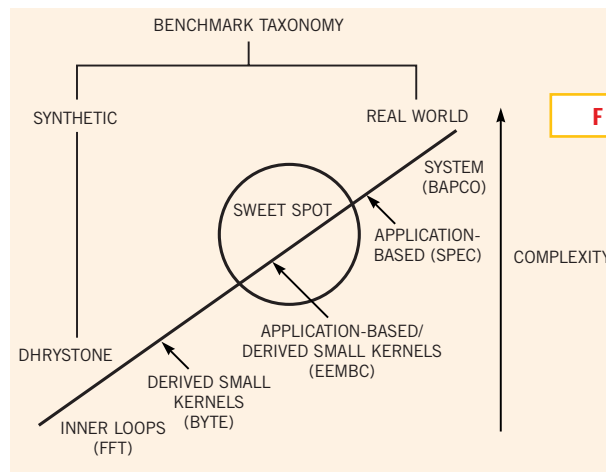


Figure 1

EEMBC bases its benchmarks on real embedded applications. The benchmarks have the properties of synthetic and real-world benchmarks (courtesy Alan Weiss, Motorola).

THE ULTIMATE OXYMORON: DEPENDABLE BENCHMARK SCORES

If you have ever run a benchmark, you know there's always a way to run it so that your product comes out as the winner. Furthermore, you can render a capable benchmark useless because no one believes the results. To overcome this age-old problem, the EDN Embedded Microprocessor Consortium (EEMBC) delivers not only the benchmarks, but also authenticated scores.

Analogous to Underwriters Laboratory approval or the Good Housekeeping Seal, EEMBC approval encompasses certification at the EEMBC Certification Lab (ECL), which carefully controls EEMBC's benchmarks. Any published benchmark score that bears the trademarked EEMBC name or logo must undergo a strict certification process at ECL.

This certification guarantees not only honest and accurate information, but also EEMBC's integrity.

But certification is technically challenging and painstaking. For "out-of-the-box" certification, the main challenge is to determine whether a vendor's compiler is generating "genuine" optimized code. Even moderately capable compiler vendors can "hide" an EEMBC optimizer within their compilers, as many do for the Dhrystone benchmark. Under normal compilations, the optimizer would be invisible to the user. When the compiler preprocesses a C program, it could start by searching for key words, such as "EEMBC"; the name of one of the benchmarks; or even variables. Then, if the compiler gets a hit, it would replace the out-of-the-box

code with highly optimized code that the compiler could never generate on its own. A visual inspection of the disassembled binary code would be both too time-consuming and somewhat impossible with complex processor architectures. One way to defeat benchmark-tuned compilers is for ECL to have a version of EEMBC code that includes no key words or even has different variable names. However, compiler vendors can be even smarter. For example, they can make their compilers search for patterns or other features. To succeed, ECL must outsmart the smartest compiler—and compiler vendor.

Result certification is also a difficult challenge for vendor-optimized EEMBC code. ECL must inspect all optimized algorithms to ensure that the vendors have

followed the benchmark's guidelines, which EEMBC will define for each benchmark. The biggest challenge is ensuring that the benchmark's guidelines are black-and-white. In other words, ECL must be able to clearly determine whether EEMBC allows an optimization.

Once ECL certifies benchmark scores, it posts the scores, along with a full disclosure of the benchmark environment, on EEMBC's Web site, www.eembc.org. This posting includes the processor architecture, device name, operating frequency, memory wait states, compiler version, compiler switches used, and more. Furthermore, ECL will certify and publish benchmark scores only from commercially available silicon and compilers.

A more complex network-benchmark algorithm deals with the creation of the routing table using the “Practical Algorithm To Retrieve Information Coded in Alphanumeric” (PATRICIA), a radix-search algorithm. PATRICIA, which finds widespread use for moderate to large tables indexed by long keys, features a worst-case $O(\log N)$ execution time in searching the table for a matching entry. $O(\log N)$, a mathematical notation from the theory of algorithms, is the minimum power of two that results in a number larger than the number of routers that the algorithm sorts. For example, a table of 40,000 router entries would arrive at the matching entry in no more than 16 steps. Another benchmark, the topological database for the reference network using the Dijkstra Shortest Path First (SPF) Algorithm, complements the PATRICIA table. The algorithm begins by scanning the database, creating a “fringe” priority queue, and then forming the SPF tree itself. After the algorithm forms the tree, you know the optimum virtual links for forwarded packets. The algorithm enters those links when the virtual hardware inserts each route entry in the PATRICIA routing table. The execution of the Dijkstra algorithm is within $O(N \cdot \log N)$. The number of operations is proportional to linear time because every insertion into the fringe queue requires a priority sort of all the elements.

FLEXING A PROCESSOR’S MUSCLE

The Bezier Curve Interpolation Algorithm from the Office Automation Subcommittee is another example of the complexity and lifelike quality of EEMBC’s benchmarks. You form a Bezier curve by repeatedly connecting the midpoints of the lines formed by four defining points. In other words, you use three lines to connect four points. Then you can draw two more lines connecting the midpoints of the previous three line segments. You now have four inner lines and can draw more line segments to connect their midpoints, and so on. You could theoretically repeat this process infinitely and form a curve. Printer manufacturers use Bezier curves in scalable-font technology, because you can define four general points and calculate a smooth curve at any size. Because you can calculate the midpoints an infinite number of times, a font of Bezier curves would be infinitely scalable.



THE BEZIER CURVE INTERPOLATION ALGORITHM IS ANOTHER EXAMPLE OF THE COMPLEXITY AND LIFELIKE QUALITY OF EEMBC’S BENCHMARKS.

The equations you use to calculate a Bezier curve are third-degree equations, meaning they have variables raised to the third power. EEMBC’s algorithm uses a parametric equation in t to interpolate the points along a Bezier curve, where t is either zero or one. The algorithm alternately calculates the X and Y values of the points using the X and Y values of the endpoints and control points. The equation is: $P(t) = p_0(1-t)^3 + 3p_1t(1-t)^2 + 3p_2t^2(1-t) + p_3t^3$, where $p_0 - p_3$ is either the X or the Y value

of the four points, and $P(t)$ is the interpolated X or Y value of the intermediate point. The innermost loop calculates a number of values for the parametric variable (given by a defined constant in the header file), going from zero to one. The X and Y coordinates for each intermediate point are calculated for each of these parametric values. Another outer loop around this calculation iterates through all the collected Bezier-curve structures.

Under normal operation, the algorithm compiles in the data as an array of curve structures, eliminating the intermediate steps of converting ASCII text lines to floating point, to (X,Y) point structure, and, finally, to curve structures. The header file for the curve data includes a defined constant for the size of the array, and the algorithm uses this value to count the number of iterations through the curve array. (The array size is greater than 16 kbytes to exceed the limit of most caches.) EEMBC generated the compiled-in array using a random function with a uniform distribution of values between zero and 10 and rearranged the resulting points so that the two control points fall between the endpoints.

A BEHIND-THE-SCENES LOOK

EEMBC considered many technical factors during the development of its benchmark algorithms. The algorithm’s size plays a significant role in the design of a bench-

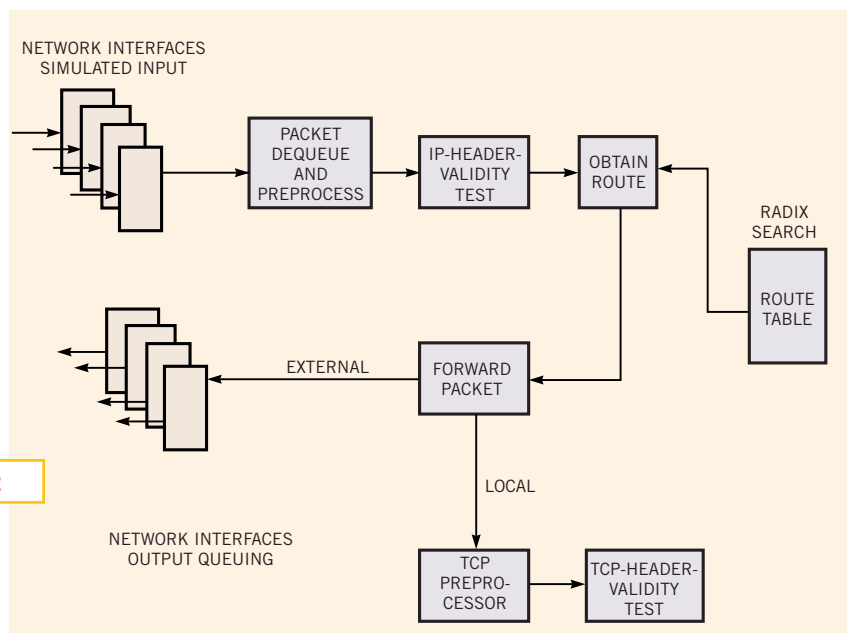


Figure 2

In this networking benchmark, packets flow from the simulated virtual-network interface to the output queue (courtesy Excelsior Research, www.dffx.com).

mark, primarily from a cache perspective. Many of the real-world algorithms in the EEMBC benchmark suite are relatively small. The procedure for running a small algorithm may require it to loop many hundreds or thousands of times to get an accurate result. This looping could potentially deliver one of the drawbacks of the Dhrystone program: The code fits into the cache and doesn't test the processor's external-bus structure. To resolve this drawback, EEMBC studied how its benchmark algorithms behave within the context of the application. In some cases, an algorithm apparently runs repeatedly, but not continuously, during the operation of the application. Therefore, depending on a processor's cache size or on whether you use cache locking, the processor would repeatedly have to reload the algorithm into its cache. So, when appropriate, EEMBC wrapped "cache-thrashing" code around the benchmark algorithm to simulate the real-world effect and exercise a processor's bus-interface unit as it loads code and data from external memory into the on-chip cache.

The algorithm's code and data sizes are



**STANDARDIZING THE INPUT AND
OUTPUT DATA SETS IS JUST AS
IMPORTANT AS THE BENCHMARK
ALGORITHM ITSELF.**

also important factors for developing benchmarks to run on processors ranging from 8-bit microcontrollers to DSPs to 64-bit μ Ps. For example, the JPEG Compression Algorithm of the Consumer Suite may require a large input data set that represents

a bit map that the processor must compress. In a real-world implementation, the original bit map could exceed 1 Mbyte, clearly overtaxing the address range of most 8-bit processors. EEMBC resolved this issue by defining data sets as small (320×240 pixels), medium (640×480 pixels), and large (800×600 pixels), with an option to provide benchmark scores for each of these data sets.

EEMBC provides more than just the benchmark algorithms. For an unbiased comparison, standardizing the input and output data sets is just as important as the benchmark algorithm itself. Furthermore, it is not enough just to measure the raw performance of a processor: The processor running the benchmark must also generate the "correct" result. This fact presented EEMBC with the challenge of determining the signal-to-noise ratio (SNR). Some benchmark algorithms, such as a convolutional encoder or Viterbi Add-Compare-Select (ACS), must produce bit-exact results because they require precise digital translation. But algorithms that generate data (that is, those related to audio or

video) for humans to interpret need not be bit-exact. ADSL, for example, which uses the FFT and autocorrelation algorithms, has a degree of error tolerance. To determine the error tolerance of the 512-point real FFT using 16-bit data and eight stages, you can assume the rule of thumb that you lose about 1 bit of accuracy per stage. Therefore, 50 dB would be the lower bound of the FFT benchmark with a theoretical maximum of 95 dB. For each of the benchmarks that don't require a bit-exact result, EEMBC determines the acceptable limit for successfully performing the benchmark. Typically, a balance exists between high performance and error rate.

THE COMPILER SCRIMMAGE

Keeping score is the most controversial aspect of any benchmark and involves determining which benchmark-coding optimizations a vendor can make while maintaining the essence of the benchmark and ensuring that the vendor's scores are honest and accurate (see **sidebar** "The ultimate oxymoron: dependable benchmark scores").

To deal with the first issue, EEMBC requires unoptimized and optimized scoring. EEMBC refers to the unoptimized type of score as "out-of-the-box" benchmarking; anyone running the benchmarks must use the unmodified source code that the EEMBC delivers. In EEMBC's Phase 1 benchmark release, all benchmark source code uses standard ANSI C programming. Although this code "theoretically" ensures portability across all processors and their corresponding compilers, strict ANSI C code doesn't always yield the highest performance code (even with all compiler optimizations active). The term "theoretically" applies because EEMBC discovered that some tested "ANSI C-compliant" compilers require custom modifications to the ANSI C-compliant EEMBC code.

The compiler is inseparable from most designs in today's embedded-development environment. This fact is especially true as more and more designs involve the use of the C language. Running out-of-the-box benchmarks provides an excellent means for comparing one compiler with another, especially when several compilers support

one processor architecture. The EEMBC benchmarks will help elevate the overall quality of compilers for embedded processors, because they are the first industry-standard benchmarks that test compilers in a real-world application.

EEMBC's rules also allow you to optimize the benchmarks to take advantage of a processor's features, as well as any special compiler optimizations. An obvious example would be for you to replace a multiply-and-add sequence with a single multiply-accumulate instruction if applicable. Other more complex optimizations may depend on the processor and the application. You may need to profile the execution of the benchmark using the appropriate tools to understand where and how to make optimizations.

Although each benchmark has specific optimization guidelines to preserve its essence, EEMBC's main goal is to provide a real-world scenario. In other words, optimizations will allow you to make the benchmark code similar to what you would see in a real application. It will be interesting to compare the out-of-the-box and op-

timized benchmark results and study the amount of code tweaking that was necessary to achieve better performance (Reference 1).

AUTOMATING THE TESTING

Whether you're using out-of-the-box or optimized benchmark code, you'll find it helpful to have a mechanism that automates the execution of these benchmarks. EEMBC members Richard Russell (AMD) and Alan Weiss (Motorola) jointly developed an automated test mechanism that supports deterministic control of the target under test. This mechanism, code named the Test Authenticator, uses a command-line-based program (not a graphical-user-interface program) on the host computer that communicates to the test mechanism running on the target device (that is, the embedded platform). Communication occurs via a simple protocol. The Test Authenticator uses the Perl scripting programs to drive

the host control program to run fully automated tests.

The Test Authenticator also isolates system dependencies from the host computer. This approach means that all benchmark target platforms should act the same, as far as the host is concerned. The basic operations of loading, running a benchmark, and gathering results are identical for every target. You can port the Test Authenticator to your target platforms in a way that requires no target-specific host-side work.

In the second quarter of 1999, EEMBC's Web site, www.eembc.org, will begin posting EEMBC Certification Lab-approved benchmark scores. Also within that time frame, you will be able to license the benchmarks; licensing information is posted on EEMBC's Web site. Until now, the embedded industry has never had a comprehensive benchmark suite that could analyze a processor's behavior in the context of a given application. Furthermore,

there has never been an official benchmark program that could provide you a guarantee of the accuracy of the scores. EEMBC represents the efforts of 21 semiconductor companies. Together, they have created the industry-standard embedded-processor benchmark suite that will serve the needs of every engineer facing the arduous task of selecting a processor for that next design. □

Reference

1. Levy, Markus, "C compilers for DSPs flex their muscles," *EDN*, June 5, 1997, pg 93 or ednmag.com/reg/1997/060597/12df_02.cfm.

Acknowledgments

In addition to thanking all the members for making this consortium a reality, I would like to give special thanks to Paul Cobb and Rick Kepple (QED), Kamesh Kothuri (Sun), David Lamar (NEC), Angel O'Campo and Alan Weiss (Motorola), Russ Rivin (Analog Devices), Richard Russell (AMD), and Lyle Supp (Hitachi) for their outstanding contributions to EEMBC.

You can reach EDN
Technical Editor
Markus Levy, who is
also President of
EEMBC, at 1-916-
939-1642, fax 1-916-
939-1650,
[markus.levy@
worldnet.att.net](mailto:markus.levy@worldnet.att.net).

