By Markus Levy, Technical Editor

Embedded-system design: Exploring trade-offs

REAL-WORLD EMBEDDED-SYSTEM design requires real-world decisions; it's all about trade-offs. For many embedded-system designs involving processors, the goal isn't to pick the fastest processor but to pick the processor and I/O subsystem that can get the job done. Of course, there's nothing like experience to help you make those decisions. On the other hand, many processor vendors provide evaluation boards that let you test the theory and verify your decisions.

For this article, I stretched the capabilities of a handful of evaluation boards to figure out what design trade-offs are all about. However, one of the difficulties of a project like this is that many of the evaluation boards or processors provide only a limited set of features that you can evaluate. The main ingredients for this project included the IDT79-S465+IDT79S574 board with a 250-MHz IDT-79RC64575, the Mitsubishi M30620TB-RPD-E and MSA0600 kit with a 16-MHz M16C/62A, the NEC Midas RTE-V832-PC with a 143-MHz V832, the NEC DDB-VRC5074 Revision 4.0 with a 250-MHz VR5000, and the STMicroelectronics STi5500-DEMO with a 50-MHz ST20C2.

I worked with Integrated Device Technology's

EMBEDDED-SYSTEM DESIGN IS ABOUT TRADE-OFFS. *EDN* EXPLORED THE LABS OF FIVE LEADING SEMICONDUCTOR VENDORS TO EXAMINE A VARIETY OF PROCESSOR AND SYSTEM-LEVEL TRADE-OFFS. SEE WHAT WE DISCOVERED.

Illustration by Mike O'Leary

(IDT's) Upendra Kulkarni, Mitsubishi Electric's Ken Sheets, NEC Electronics' Dennis Han and Bin Takahashi, STMicroelectronics' Sam Jenkins, and Toshiba America Electronics Glenn Haley. We ran a variety of the *EDN* Embedded Microprocessor Benchmark Consortium (EEMBC) benchmarks through their processor boards, changing jumpers, turning off caches, and using different code-compilation options. I have normalized all the results that appear in this article to make it easier for you to assess the performance differences. The EEMBC Certification Labs (www.embedded-benchmarks.com) has certified these benchmarks, and they are available on the EEMBC Web site at www.eembc.org/benchmark. This Web site also contains data sheets describing each benchmark.

AT A GLANCE

▷ The *EDN* Embedded Microprocessor Benchmark Consortium benchmarks are useful tools for analyzing processor, system, and compiler performance.

▷ Memory speed makes a significant difference in overall system performance.

▷ A processor that includes a write-back cache mode yields big performance gains.

AUTOMOTIVE/INDUSTRIAL BENCHMARKS

Mitsubishi's Sheets and I used several EEMBC automotive/industrial benchmarks to experiment with Mitsubishi's M16C/62A running on the PC4701 emulator with the M30620TB-RPD-E pod with zero wait-state synchronous DRAM (SDRAM). As its name implies, the M16C/62A is a 16-bit CISC microcontroller. The general instruction format of the M16C allows you to transfer data between any two registers within the CPU or a 1-Mbyte address space. Other pertinent features of the M16C include a user-selectable 8- or 16-bit-wide data bus and bit-manipulation instructions. The compiler is the Mitsubishi NC30-Version 3.20 Release 1.

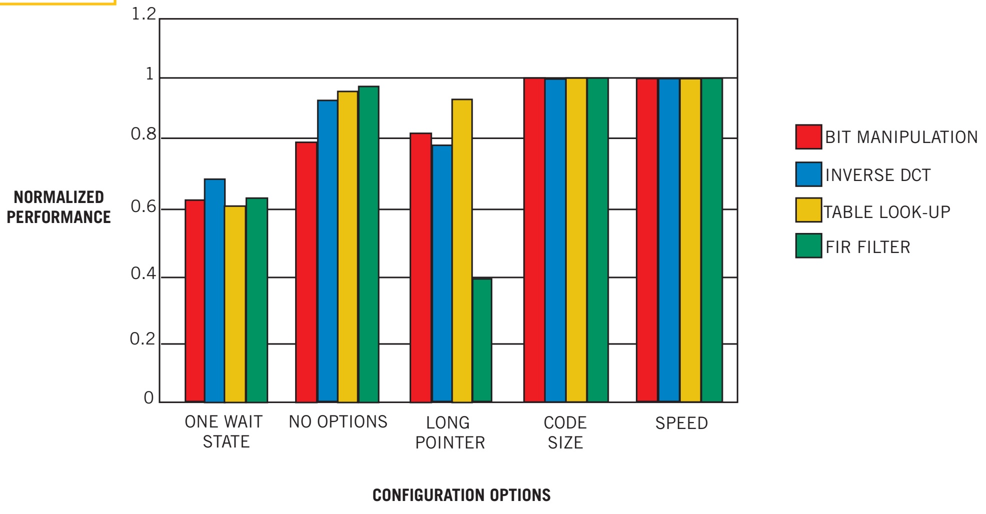
For the M16C, it was straightforward to try compilation options, including the use of the far-memory model, optimizing for code size and without optimizations. We also imposed one wait state on the processor's memory access by setting the processor-configuration register, Processor Mode Register 1, in the initialization file. One useful option that we didn't perform was demonstrating the performance difference between an 8- and a 16-bit memory bus. This task would have required us to put the M16C/62 in microprocessor mode to use external memory, change hardware settings on the emulator pod, and change the EEMBC code.

I normalized the data for the M16C/62 to the fastest execution for each benchmark. So, for example, the fastest run for the bit-manipulation benchmark is a 1; any of the other results for the bit-manipulation benchmark is some fraction of 1. Compiling for optimal code size had an insignificant effect on performance. However, this compiling also minimally affects code size. For example, on the inverse-discrete-cosine-transform benchmark, the saving in code size is approximately 1%.

Compiling with the far-memory model significantly affects performance. The far-memory model shows the effect of using the memory space greater than 64 kbytes; this technique requires the use of 32-bit pointers, which adds the execution overhead of manipulating 32-bit values using 16-bit registers. For example, if the address requires more than a 16-bit pointer (as it would in a far-memory model), the CPU must perform separate operations on the lower and upper 2 bytes. However, if the CPU reads or writes to data higher than 64 kbytes, but the address pointer doesn't change, no decrease in performance occurs. The far-memory model hits the FIR filter harder than for the other three benchmarks tested, decreasing performance by more than 60%. This decrease is the result of a large amount of data processing (both reads and writes) for the FIR filter and the need for frequent pointer manipulation. However, this information indicates that you should evaluate the use of data memory beyond 64 kbytes on a case-by-case basis, because the far-memory model does not as significantly affect the other benchmark scores.

Turning off compiler optimizations also yielded interesting results (**Figure 1**). Except for the bit-manipulation benchmark, performance dropped less than 7%. I have insufficient data to explain this finding, but several potential theories exist. First, the compiler/processor combination could inherently work well together; the compiler automatically im-

Figure 1



Results from Mitsubishi's M16C/62A microcontroller shows a variety of options with EEMBC's automotive/industrial benchmarks.

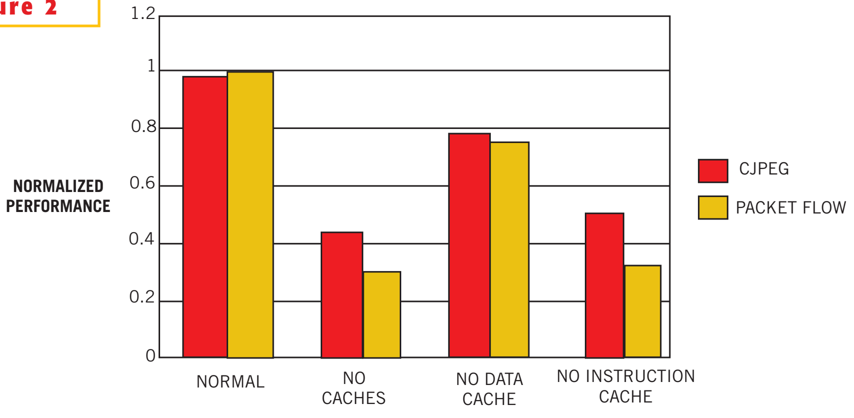
plements many optimizations. Alternatively, because we derived these results using the EEMBC out-of-the-box rules, the lack of intrinsics or other compiler hints impedes the compiler's optimizing capability. If these impediments are what occurred, the M16C/62 should demonstrate significant performance improvements when Mitsubishi engineers implement EEMBC's optimized format. On the other hand, processor performance dropped 21% when we turned off optimizations for the bit-manipulation benchmark. Mitsubishi claims that this decrease is due to the processor's and compiler's abilities to handle bit-masking and -shifting operations.

JPEG COMPRESSION AND PACKET FLOW

In the next set of tests, I examined a variety of processor configurations for EEMBC's JPEG-compression and networking-packet-flow benchmarks. The JPEG-compression benchmark divides an image into 8×8-pixel blocks and calculates the discrete cosine transform (DCT) for each block. A quantizer then rounds off the DCT coefficients according to the quantization matrix. The compression technique uses a variable-length code on these coefficients and writes the compressed data stream to memory. The packet-flow benchmark operates on 2 Mbytes of data. It receives and processes data packets. Both benchmarks handle large amounts of nonrepeating data; this feature minimizes the benefits of a data cache.

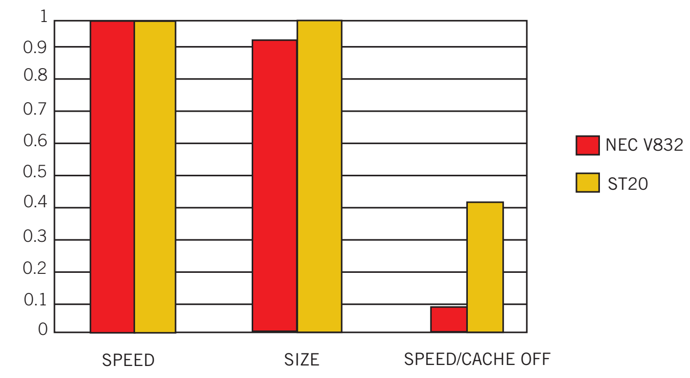
Jenkins and I started off with STMicroelectronics' ST20C2 running at 50 MHz. This low-end, 32-bit RISC processor has 2-kbyte instruction and data

Figure 2



Results from STMicro's ST20 running the JPEG-compression and networking-packet-flow benchmarks use different cache options.

Figure 3



Comparing NEC's V832 results with STMicro's ST20 using different compilation and cache options with the JPEG-compression benchmark demonstrates that the faster the processor, the more beneficial the caches.

FOR MORE INFORMATION...

For more information on products such as those discussed in this article, enter the appropriate numbers at www.ednmag.com/infoaccess.asp. When you contact any of the following manufacturers directly, please let them know you read about their products in *EDN*.

EEMBC
www.eembc.org
Enter No. 301

Green Hills Software
www.ghs.com
Enter No. 302

Integrated Device Technology
www.idt.com
Enter No. 303

Mitsubishi Electric Corp
www.mitsubishichips.com
Enter No. 304

NEC Electronics Inc
www.necel.com
Enter No. 305

STMicroelectronics
www.st.com
Enter No. 306

Toshiba America Electronics Components Inc
www.toshiba.com/taec
Enter No. 307

SUPER CIRCLE NUMBER
For more information on the products available from all of the vendors listed in this box, enter No. 308 at www.ednmag.com/infoaccess.asp.

caches and a 16-bit external-memory interface. For this benchmark configuration, the memory bus is 50 MHz with an access rate of 4-3-3-3. We used STMicro's ST20 ANSI C Compiler Version 1.8 to compile the EEMBC benchmarks. Turning off the instruction cache caused performance to drop 50 to 70% depending on the benchmark, whereas the absence of a data cache caused only a 21% drop in performance (**Figure 2**). (Incidentally, I normalized the CJPEG and packet-flow scores independently from one another.)

Another interesting perspective on this analysis, using NEC's V832, shows that performance drops off more dramatically on a faster processor with larger caches. With both processors' scores normalized independently, the V832 performance dropped 92% when we turned off the caches (**Figure 3**). The V832 is a 143-MHz, 32-bit RISC processor with 4-kbyte instruction and data caches. (Takahasi and I used Green Hills Software's NEC V800 Version 1.8.9 compiler to compile this EEMBC benchmark.) With caches not enabled, this 143-MHz processor basically runs at the external bus speed. For the V832, we also compiled the CJPEG code optimized for size and saw an almost-12-kbyte drop in code size—from 36,118 bytes to 24,842 bytes—and an 8% drop in performance. The V832 board, the Midas RTE-V832-PC, allowed us to easily switch the external bus width between 16 and 32 bits. We ran the CJPEG and packet-flow benchmarks with these two configurations (**Figure 4**). (We normalized the fastest performance for each benchmark to 1.) The resultant drop in performance for both benchmarks was 12 to 14%; depending on the needs of your application, the reduced memory cost may be worth the performance sacrifice.

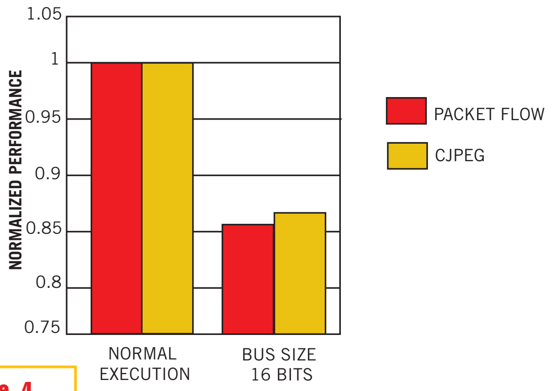


Figure 4

Bus width plays a big role in performance, as NEC's V832, running the packet-flow and JPEG-compression benchmarks, shows.

FREQUENCY VERSUS MEMORY

Kulkarni and I did a similar test using IDT's 79RC64575 processor with the company's IDT79S465+IDT79S574 motherboard/daughterboard combination. The 79RC64575 is a MIPS-based, 64-bit RISC processor. The implementation we used ran at 250 MHz with a 50-MHz memory bus and contained 32-kbyte instruction and data caches. The compiler was IDT's homegrown version of GNU. The data for the packet-flow benchmark indicates that performance of this processor dropped 20% when we halved the 64-bit external-bus width. Similarly, performance dropped 94% when we shut off the caches. (Incidentally, for the three MIPS processors that we used in this project, we didn't turn off the caches. Instead, we modified the link files so that the ROM monitor loaded the code and data into noncacheable regions.)

IDT's processor board comes with both SDRAM and SRAM with access rates of 7-1-1-1 and 3-1-1-1 for reads, respectively. By modifying the link file, we located the benchmark code and data in either SDRAM or SRAM. A jumper on the board also allowed us to change the processor frequency, although the memory-bus speed remained constant (**Figure 5**). Note that performance scales almost linearly for both the SDRAM and the SRAM implementations. But memory speed becomes more critical at higher frequencies, as the 21% performance delta at 250 MHz indicates. Compare this figure with the 9% performance delta at 150 MHz. Recall that this 20% drop in performance is equivalent to the drop associated with running with a 32-bit memory bus; this figure gives you an idea of your design-tradeoff options.

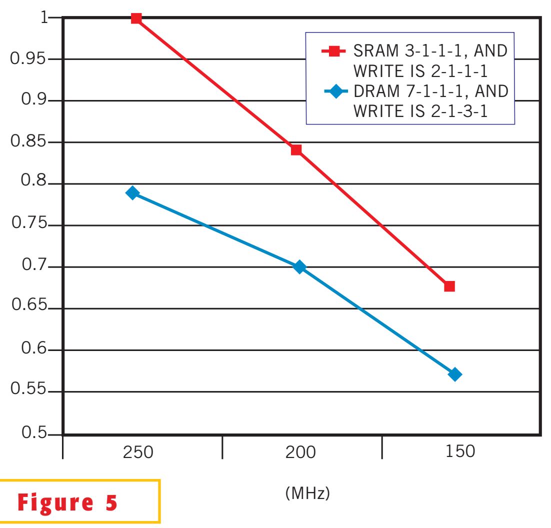


Figure 5

Memory type, access latency, and processor frequency play big roles in performance, as IDT's IDT79RC64575 running the packet-flow benchmark shows.

NETWORKING BENCHMARKS' RESULTS

Toshiba's Haley and I tried all of EEMBC's networking benchmarks on the company's BMU3927JMR board with the MIPS-based, 32-bit TX3927 processor. The TX3927 implementation we used ran at 133 MHz, contained a 4-kbyte instruction cache and a 2-kbyte data cache, and used a 66-MHz external bus with a 5-1-1-1 access configuration. The compiler was the Green Hills Multi2000 Version 2.0 (**Figure 6**). In addition to the packet-flow benchmark, this testing included the route-look-up and open-shortest-path-first (OSPF) benchmarks. The most interesting finding was that the OSPF performance increased when we optimized the benchmark for code size. One logical explanation is that the compiler, to save code size, achieved tighter loops within the code. The route-look-up benchmark benefited most from compiler-speed optimizations, dropping

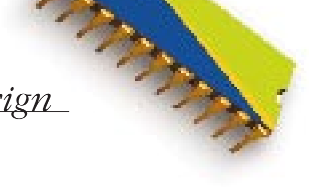
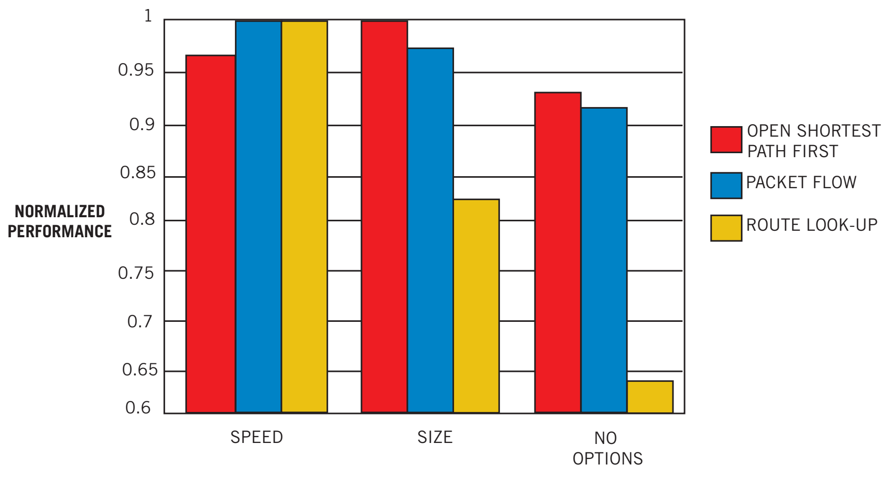


Figure 6



Toshiba's TX3927 demonstrates that the effect of compiler optimizations depends on the algorithm.

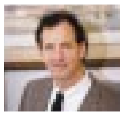
18 and 35%, respectively, when we optimized for size and turned optimizations off. Similar to the V832 and other MIPS processors, the TX3927 performance dropped greater than 90% when we did not use the caches.

Speaking of caches, NEC's Han and I checked out the cache protocols for NEC's VR5000. We measured the performance differences from using the processor's write-back and write-through protocols. The VR5000 setup we used was similar to IDT's 79RC64575. It ran at 250 MHz and contained 32-kbyte instruction and data caches, but the system had a 100-MHz memory bus. Similar to Toshiba's implementation, this implementation used Green Hills' Multi2000 Version 2.0 compiler. When we used the write-through mode for the packet-flow and JPEG-compression benchmarks, processor performance dropped 42 and 69%, respectively. JPEG performance suffered more because of the greater number of sequential writes. Also, the JPEG algorithm deals with 16-bit data, which is more inefficient on this 64-bit processor. (Benchmark optimizations would help with this issue.) Similar results for EEMBC's DCT benchmark showed that performance dropped almost 60% when using the write-through mode. In write-through mode, cache writes pass through to memory, raising bandwidth requirements. On the other hand, the write-back mode reduces the number of memory transactions from the core but requires more processor logic to implement. This use of the write-back mode is the reason that many processors do not implement a write-back mode, but it creates a big performance penalty.

THINGS TO TRY ON A RAINY DAY

Although many of the results we obtained in this project seem obvious, it is beneficial to have quantitative data to rely on. Furthermore, the standardization of the EEMBC benchmarks ensured that all processors were running the same tests. In a perfect world, we could have run these benchmarks with many more processor, compiler, and system configurations. But many configurations were impossible, given the available tools. For example, it would have been useful to change a processor's cache size to determine the "knee" in the curve where performance drops off. But in real hardware, changing the cache size requires that you lock down regions of the cache with invalid data, creating a virtually smaller cache. It would also have been helpful to vary the number of wait states or the external bus widths for all the processors we used. Using a simulator instead of real hardware would have made many more experiments possible, but I think I'll save those for a rainy day.□

You can reach Technical Editor Markus Levy at 1-530-672-9113, fax 1-530-672-9103, e-mail markus@embedded-benchmarks.com.



Enter No. 4 at www.ednmag.com/infoaccess.asp