Microprocessor Benchmark Results

Rabbit Semiconductor 5 Feb. 2000

1. Description of Benchmark Tests

A benchmark program was devised to test the floating point performance of the Rabbit 2000 against a number of popular 8-bit microprocessors. In order to ensure that the comparison was based on practical microprocessor systems the following ground rules were applied.

- The system must be able to generate standard baud rates. This restricts the clock frequencies that can be used.
- The system must be able to operate using 55 nS access time flash memory to store the code. This avoids system that must use impractical or overly expensive memory to store code.

In most cases the actual tests were run using available equipment and in one case a software simulator. The performance figures were scaled if the clock speed that the test board operated at was different than the ideal clock speed according the ground rules above.

The microprocessors tested are as follows

- Rabbit 2000 microprocessor using a Z-World Jackrabbit board operating at 14.73 MHz and scaled to 29.49 MHz. Z-World's Dynamic C for the Rabbit version 6.10 was used.
- AMD 188ES microprocessor using a Tern A104-40 board operating at 40 MHz and scaled to 36.8 MHz. Borland C version 3.31 was used.
- Zilog Z180 microprocessor using a Z-World model 2200 controller operating at 9.216 MHz and scaled to24.47 MHz. Z-World Dynamic C for the Z180 was used.
- Dallas DS80C320 high speed 8051 compatible microprocessor using a Systronix, Inc. HSM-KISS controller board operating at 25 MHz and scaled to 33.18 MHz. The Kiel 8051 compiler was used.
- Generic (e.g. Phillips) 8051 as tested using a JSIM-51 an 8051 simulator developed by Jens Altmann.

  See homepage: http://www.home.t-online.de/home/Jens.Altmann/jsim-e.htm

  The results were scaled to match an 8051 operating at 33.18 MHz.

The floating point operations benchmarked included add, multiply, divide, square root, sine, log, exponent and arc tangent. In addition one benchmark of integer arithmetic, the "sieve of Eratosthenes," was run on all of the processors. All of the tests were run with no wait states in the memory access. Corrections were made for the overhead of the clock interrupt when applicable. An interrupt driven clock was used to time the measurements except in the case of the AMD188ES where a stopwatch was used to time the interval between breakpoints in the debugger.

2. Results

The table below contains the results for the various microprocessors and benchmarks. All the results are given in microseconds for the particular operation except that the sieve is given in milliseconds.

Table 1. Floating Point Operation Benchmarks

	Rabbit 2000	8051	Dallas DS80C32 (fast 8051)	Z180	AMD188ES (x86)
Clock Speed MHz	29.49	33.18	33.18	24.58	36.86
floating add	9.6 uS	78 uS	32 uS	26 uS	194 uS
floating mul	12	85	34	42	184
floating div	27	239	156	109	202
square root	32	805	334	343	355
sine	94	1112	452	1238	804
atan	118	1327	551	1262	1195
log	134	1482	613	1465	1141
exp	93	1815	740	772	1360
sieve (int, ms)	90 mS	270 mS	123 mS	301 mS	130 mS

3. Analysis of Results

It should be obvious that the results depend on the quality of the floating point library, the speed of the microprocessor and the quality of the code generated by the compiler. In addition the time required for a particular operation may vary considerably depending on the values used. For example, some libraries may compute the sine of zero very quickly because the library routine contains a shortcut for this special case. An attempt was made to avoid non representative tests, but considerable variation may still exist due to this factor. In addition the results above may vary slightly from results of similar tests alluded to or published at other times.

The poor results for the AMD188ES for floating point operations is a typical characteristic associated with development software for x86 processors without hardware floating point. This is due to the use of a software emulator for the hardware floating point unit when the hardware unit is not present. Such an approach is very inefficient because the software must emulate the elaborate hardware and thus do work that goes far beyond what is necessary to accomplish the floating operation desired. The relative effect is worse for simple operations, such as add or multiply. It is not unfair to use these numbers in a comparison since the Borland compiler is widely used for embedded work with the x86 family and other available compilers appear to take the same approach.

Starting with version 6.10 of Dynamic C for the Rabbit the floating point library was enhanced and is highly optimized for speed. More efficient algorithms were adopted for divide and square root. Power series expansions were performed using integer arithmetic.

Both the Keil and Borland C compilers are considered (except for the Borland floating point on non-FPU systems) to be among the best and most highly optimized compilers available for embedded work and they have been in use for many years. Dynamic C for the Rabbit, being a new compiler, will probably improve its performance fairly rapidly, increasing the advantage of the Rabbit-Dynamic C combination.

4. Other Tests Performed

A few benchmark tests were performed on the Atmel AVR. The AVR is not an 8-bit microprocessor (although it is so advertised) since its instruction word is 16-bits wide. In addition it has a Harvard architecture and a RISC style register set. Harvard architecture uses independent code and data memories that may be accessed in parallel.

The tests were performed using the IAR demo AVR compiler, downloadable from the IAR web site, that includes a simulator that may be used to keep track of clock cycles. The times are for a 6 MHz clock, the maximum possible for the AVR. The floating add required between 14 and 50 microseconds depending on the numbers. The floating multiply required about 100 microseconds. The square root required about 500 microseconds and the other math functions tested above required about 1000 microseconds. The sieve program ran in 91 milliseconds which is an excellent time. Examination of the code showed that the sieve ran fast because the compiler placed virtually all the variables in the AVR's 32 internal registers. This approach does not necessarily scale up to larger programs since there may not be enough registers, or considerable overhead will be encountered in saving and restoring registers during subroutine calls. Harvard architecture does not scale up well either, since for efficient implementation all memory must be on board the microprocessor chip.

Tests were also performed on the AMD 186ES which is a 16-bit processor. Generally the AMD 186 ran about 1.6 times as fast as the AMD 188ES.