System Benchmark Texas Instruments platforms

By Valter Minute, Windows Embedded MVP and Senior Consultant at Adeneo Embedded

Rev 1.0 – June 2012

Using the BMQ benchmark built using the ARMv4i and the ARMv7 compiler, we tested the performance of the code generated by the two compiler starting from the same source code.

The BMQ benchmark provides five measurements:

1. Integer operations – Number of 1000 function calls (testing different integer operations) which can be made during about 1 second.
2. Floating point operations – Number of 1000 function calls (testing different float operations) which can be made during about 1 second.
3. Drawing operations – Number of some drawing operations made during about 4 seconds.
4. Windowing operations – Number of some windows operations made during about 10 seconds.
5. Memory operations – Number of read/write in the memory made during about 1 second.

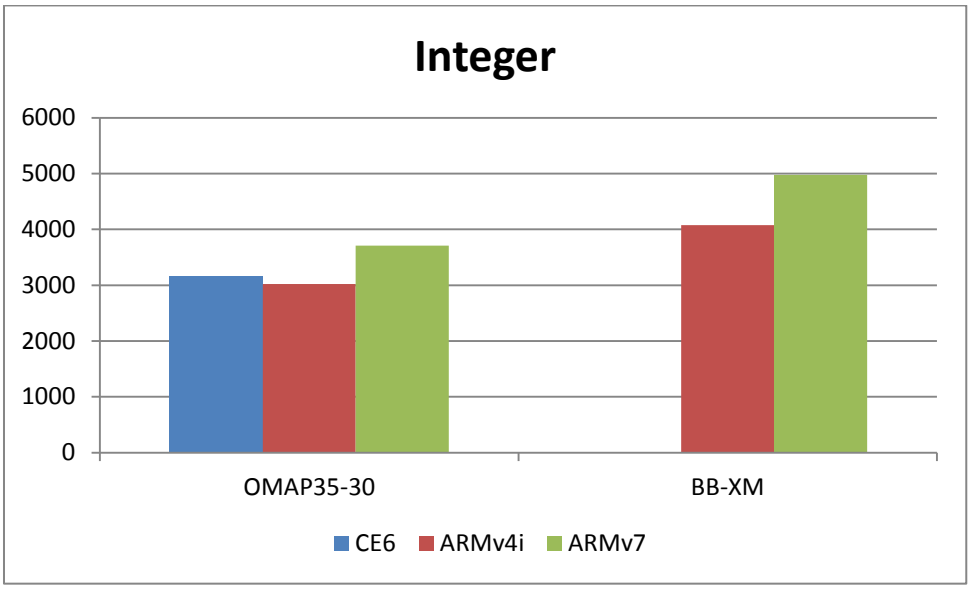
The benchmark showed a 23% performance improvement in integer operations and a 19% improvement in floating point operations. The results of the drawing, windowing and memory operations were the same for the two versions. This led to an overall improvement (the “Total” index provided by BMQ) of 8%.

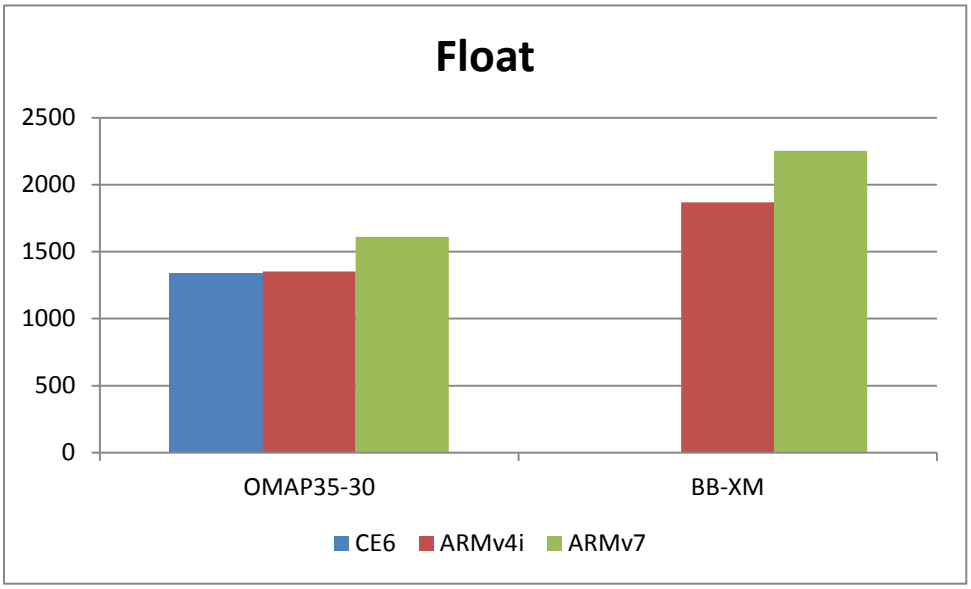
Integer and floating point operations benefits from the use of the new instruction set and, for floating point, use of the new VFP3 unit. This explains a good performance gain in those two benchmarks.

The performance of drawing and windowing benchmarks depends on the operating system and, in particular, on the degree of optimization of video drivers. The OS was the same for both tests so the performance was not influenced by the compiler used. For the memory benchmark there is no difference.

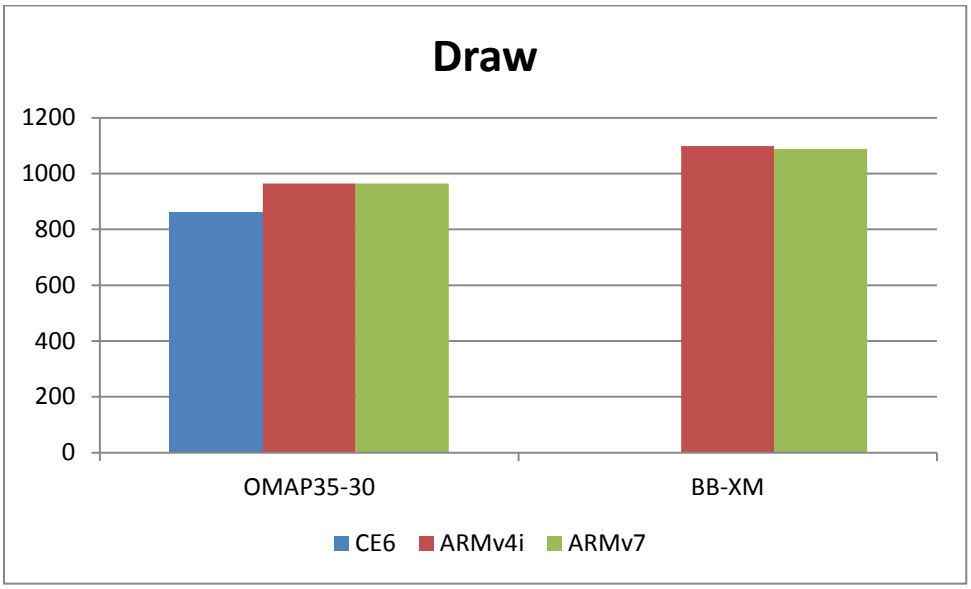
The Cortex-A8 core supports the NEON extension (support in the Cortex-A9 core is optional), which provides instruction that can manipulate multiple data in a single instruction, but this extension is not used in the code generated by the Microsoft compiler. It is possible to use those instructions for optimized functions by linking assembly modules compiled by the ARMAsm tool provided in the Platform Builder toolchain. The version of this tool included in the Windows Embedded Compact 7 toolkit can compile code that uses NEON opcodes.

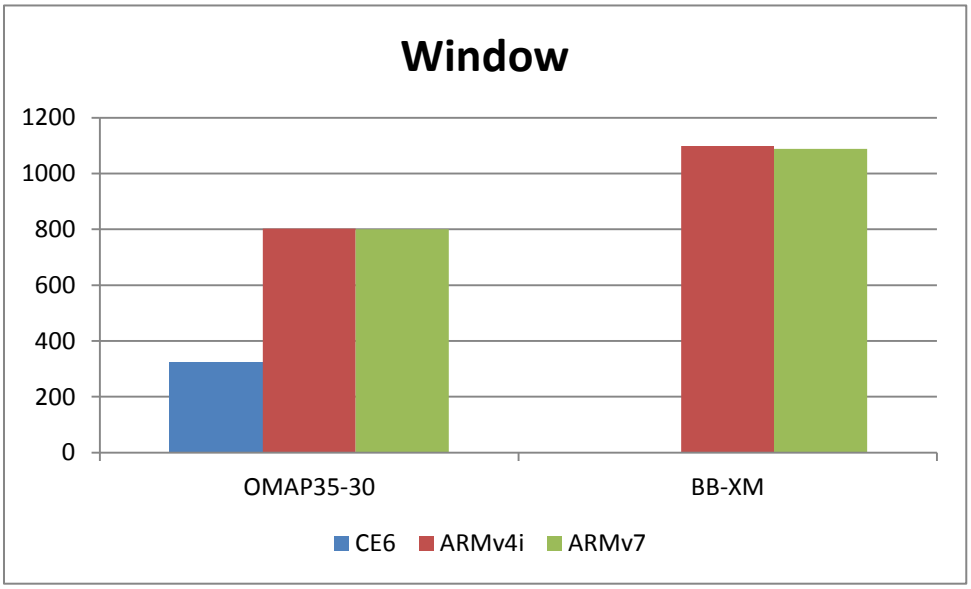
Below are the raw results of the benchmarking tests.



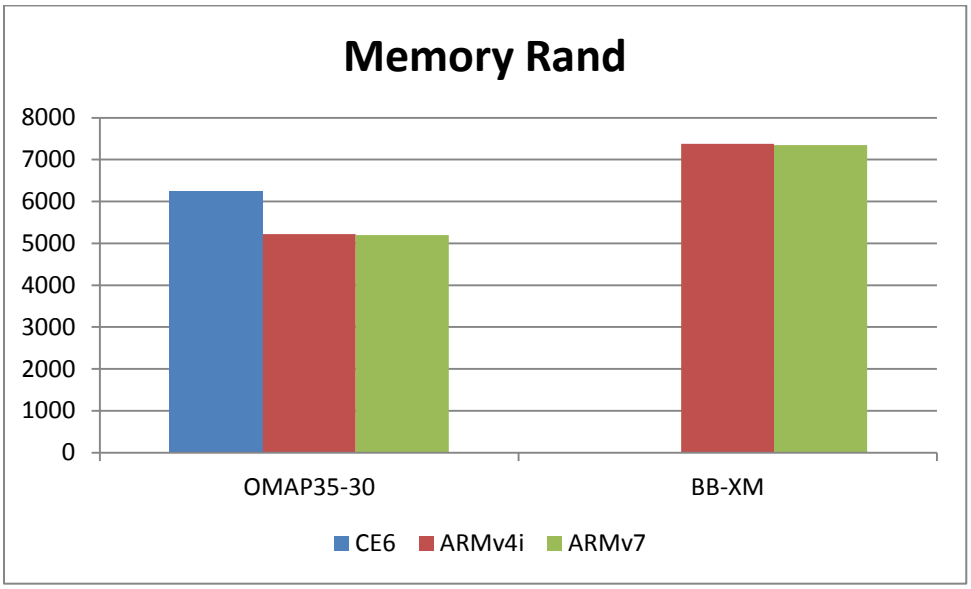


The ARMv7 compiler improves the performances of the Integer and Floating point benchmarks of around 20% on both platforms. This improvement is due to the usage of the new instructions set and, for floating point, to the support of the new VFP3 floating point unit that is part of the Cortex-A8, A9 and A15 processor cores. Applications that perform heavy calculations should get a good performance improvement using the new compiler.





The results of the drawing and window benchmarks is not impacted by the use of the new compiler. This can be explained by the fact that those benchmarks execute a sequence of API calls and the actual application code runs for a very small fraction of the benchmark time. On the other hand we notice a good improvement in draw and window results moving from CE 6 to Compact 7 on the OMAP 3530 platform. In this case BSP optimization and the improvements granted by Windows Embedded Compact 7 can grant a better performance results for UI and graphics intensive applications.



The memory benchmark results are not improved by the new compiler. The new instructions provided by the ARMv7 instruction set does not influence access to random memory locations.

This benchmark is also calling an API (through the "rand" library function). The Windows Embedded Compact 7 implementation of this API seems to be less efficient of the CE 6.0 one and this may explain the better performances on CE 6. On the other side, rand() main focus is to generate random numbers that are not predictable, not in generating them quickly.

Conclusion

In conclusion we could notice that using Windows Embedded Compact 7 and the ARMv7 compiler also for building application code can improve performances of a device.

The improvement rate depends on many factors, including also the optimization and quality of the BSP that can improve graphics operation, increase the responsiveness of the system and decrease the overhead for the CPU using hardware acceleration when available.

About the author

Valter Minute is a Windows Embedded MVP since 2009 and he has been working on Windows CE, Windows CE.NET, Windows Embedded CE and Windows Embedded Compact since 1999. He's part of the Adeneo Embedded expert team and likes to "hack" any device that seems to have a microprocessor. He has a blog about embedded software and Italian cooking at http://geekswithblogs.net/WindowsEmbeddedCookbook.