LPC4350 Project Example

Jack Ganssle, February, 2012

In the last few years the industry has increasingly embraced the notion of using multiple processors, often in the form of multicore. Though symmetric multiprocessing – the use of two or more identical cores – has received a lot of media attention, many embedded systems are making use of heterogeneous cores. A recent example is ARM's big.LITTLE approach, which is specifically targeted to smart phones. A big Cortex-A15 processor does the heavy lifting, but when computational demands are slight it goes to sleep and a more power-frugal A7 runs identical code.

NXP's LPC43xx also has two ARM cores: a capable Cortex-M4 and a smaller M0. Since power constraints are hardly novel to phones, my question was: "if we mirror the big.LITTLE philosophy, what is the difference in performance between the M4 and the M0?"

It's challenging to measure the difference in power used by the cores as there's no way to isolate power lines going to the LPC4350 on the Hitex board. The board consumes about 0.25 amp at five volts, but most of that goes to the memories and peripherals. To isolate the LPC4350's changing power needs I put a 5 ohm resistor in the ground lead to the board, and built the circuit in figure 1. The pot nulls out the nominal 0.25 amp draw, and multiplies any difference from nominal by 50. The output is monitored on an oscilloscope.

The cores run a series of tests, each designed to examine one aspect of performance. The cores run the tests alternately, going to sleep when done. Thus, after initialization only one core is ever active at a time. When running a test the core sets a unique GPIO bit which is monitored on the scope to see which core is alive, and how long the test takes to run. One of those GPIO bits is assigned, by the board's design, to an LED. I removed that so its consumption would not affect the results. All of the tests use a compiler optimization level of –O3 (the highest). The tests are identical on each processor, with one minor exception noted later.

Figure 2 is an example of the data. The top, yellow, trace is the M4's GPIO bit, which is high when that processor is running. The middle, green, trace is the bit associated with the M0. Note how much faster the M4 runs. The lower, blue, trace is the amplified difference in consumed power. I attribute the odd waveform to distributed capacitance on the board, and it's clear that the results are less quantitative than one might wish. But it's also clear the M4, with all of its high-performance features, sucks more milliamps than the M0. So the current numbers I'll quote are indicative rather than precise, sort of like an impressionistic painting.

The first test put both CPUs to sleep, which reduced the board's power consumption by about 10 ma; that is, both CPUs running together consume somewhere around 10 ma. First impression: this part is very frugal with power.

In test 0 the processors take 300 integer square roots, using an algorithm from Math Toolkit for Real-Time Programming by Jack Crenshaw. Being integer, this algorithm is designed to examine the cores' behavior running generic C code. The M4 completes the roots in 1.842 msec, 21 times faster to the M0's 38.626 msec, but the M0 uses only a quarter of the current.

The next test ran the same algorithm using floating point. The M4 shined again, showing off its FPU, coming in 12 times faster than the M0 but with twice the power-supply load. There's considerable non-FPU activity in that code; software that uses floating point more aggressively will see even better numbers.

Test 3 also took 300 floating point square roots, and is the only one where the code varied slightly between cores. On the M4 it uses the __sqrtf() intrinsic instead of the M0's conventional C function sqrt(). The former invokes the FPU's VSQRT instruction, and that CPU just screamed with 174 times the performance of the M0. It was so fast the power measurements were completely swamped by the board's capacitance.

One of the Cortex-M4's important feature is its SIMD instructions. To give them a whirl I implemented an FIR algorithm that made use of the SMLAD SIMD instruction. Since the M0 doesn't have this I used the macro from the CMSIS that requires several lines of C. Not surprisingly, the M4 blew the M0 out of the water, completing 20 executions of the filter in 5.15 msec, 10 times faster than the M0 and for 9 times as many milliamps.

But I was surprised the results weren't even better, considering how much the M0 has to do to emulate the M4's single-cycle SMLAD. So I modified the program with a SIMD_ON #define. If TRUE, the code ran as described. If FALSE, the SMLADs were removed and replaced by simple assignment statements. The result: the M4 still ran in 5.15 msec. There was no difference, indicating that essentially all of the time was consumed in other parts of the FIR code. In other words, code making heavier use of the SIMD instructions will run vastly faster.

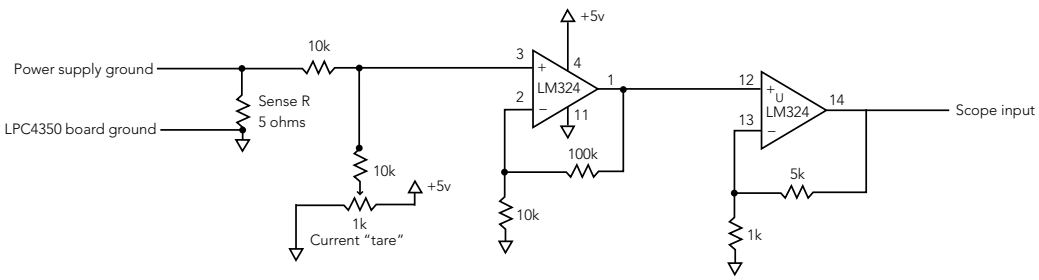

Figure 1: Current monitor

One note: in many cases the M4 consumed less power than the M0, despite the higher current consumption, since the M4 ran so much faster than the M0. It was asleep most of the time. However, in many systems a CPU has to be awake to take care of routine housekeeping functions. It makes little sense to use the M4 for these operations when the M0 can do them with a smaller power budget, and even handle some of the more complex tasks at the same time.

Though the LPC43xx family is positioned as a device that includes a fast processor with extensions for DSP-like applications, coupled with a smaller CPU for taking care of routine control needs, it's also a natural for deeply embedded big.LITTLE-like situations where a dynamic tradeoff between speed and power makes sense.

Thanks to NXP for their support during evaluation of this very impressive part

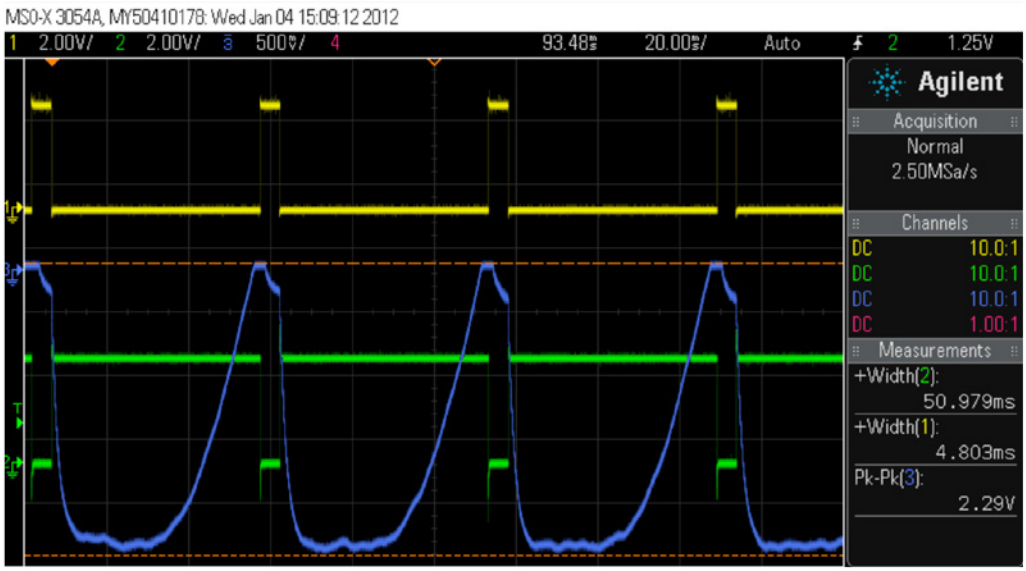

Figure 2: The FIR test results

Date of release: February 2012
Published in the USA