~ Sli.do Q&A ~
ESA IP core extensions for LEON2FT: a new FPU with configurable (half/full/double) precision, and SWAR (SIMD Within A Register) instruction extensions

Have you done any benchmarking between your implementation vs more generic SW implementations, in CPUs with and without FPU?

The most complex test we have executed is the tracking loop code for GNSS. We have started with a version of LEON2FT that used no hardware FPU and no SWAR operations. We have used this as the baseline to evaluate our more optimized implementations of the tracking loop that use the daiFPU as well as the SWAR operations. The baseline execution times are detailed on slide 12, and a comparison to several optimized versions is provided on slide 14; the times stated for config 2 iter 0 and 1 (i.e. the first two lines in the table) correspond to execution in LEON2FT without hardware FPU and without SWAR operations at 25MHz processor clock.

How about benchmarking vs hardware implementations (if available) e.g. in FPGAs?

We have not evaluated our implementation against pure hardware implementations of the GNSS tracking loop. We can assume that (semi-)pipelined versions probably exist that have acceptable latencies. The core of the question then is probably about resource requirements for the pure hardware version and the LEON2 version, or more generally about the functional density of the used silicon resources (area, gates, LUTs, FFs) and its reuse. My guess is that resource consumption for both the pure hardware version and the LEON2-based version of the tracking loop would be more or less the same, i.e. it would not differ by more than one order of magnitude. The important difference is that while the pure hardware version could not be used for any other computing workload than the tracking loop, the LEON2-based version can be used to compute other problems at the same time without changing the circuit design.

Are the SWAR resources included or without the "normal" LEON2?

The new LEON2FT distribution, which will be made available in the ESA IP portfolio, will come with both the daiFPU floating-point unit and the SWAR unit options. Both daiFPU and SWAR can be disabled in the LEON2FT configuration tool; without them the resources used for LEON2FT will be the same as for the previous version of the LEON2FT package. Just the extra resources required by the new features are shown on slide 21 for daiFPU and slide 34 for the SWAR unit.

How did you validate the FPU to guarantee exact results from all inputs? Validating a FPU is usually way more complex than designing it.

We have used a structured, 2-level approach. On the first level we have used test vectors generated by the TestFloat tool by John Hauser for all supported operations in half-, single- and double-precision format, including integer conversions for 32-bit and 64-bit integer formats. Each floating-point operation for each supported rounding mode in each of the supported precision has used its own independent test set generated by the TestFloat tool; each such test set consists of 10 million

unique test vectors. In total we have used 108 different test sets, each with 10 million unique test vectors. The validation objectives on this level were first to get results identical to the reference results computed by the TestFloat tool, and second to achieve 100% code and branch coverage. These objectives have been achieved.

On the second level we have used common floating-point benchmarks, such as the Paranoia program by prof. Kahan and the Whetstone benchmark, together with our configurable implementation of the GNSS tracking loop. The validation objective was to get the best floating-point report in paranoia, and to get identical results for programs executed in LEON2 and in a desktop PC. These objectives have been achieved.

From the presentation it seems that 1 single channel requires several ms to process 1 real time ms, this is too slow for real time applications

First, we need to say that the foreseen use of our implementation in in off-line processing of GNSS navigation samples in connection with inertial sensors and precise time reference that would allow to establish the current position of a space vehicle based on slightly outdated data samples.

The performance figures that we have presented indicate a limit value in that respect that our implementation always performs full tracking for all data samples; this is not always necessary, as suggested on slide 9 (our implementation always takes the “not bit-synced” path). The processing times stated on slide 15 correspond to a case where all signal samples are processed (check values for configuration 10 on slide 14) for a relatively low processor frequency of 25MHz; further reduction of the processing time can be achieved through down-sampling, and also simply by increasing the LEON2FT frequency from 25MHz to 100MHz or more (both suggested on the last two lines of slide 15).