2020-2021-IEFR020ESAIP0012-V1.2

ESA 4000122242/17/NL/LF
Deliverable: FR
Version: V1.2
Status: draft
Date: 2021/04/16

Final Report

ESA IP Core Extensions

ESA contract 4000122242

Martin Daněk, Roman Bartosiński
martin@daiteq.com

Contents

1 Executive Summary

This report describes work undertaken under ESA contract 4000122242 by daiteq s.r.o. The foundation for this work has been the existing LEON2-FT processor IP core that is owned and distributed by the ESA. The aim of this activity has been to extend the computing capacity of the LEON2-FT processor so as to support user-configurable integer and floating-point arithmetic. The rationale for the work is based on demand for efficient execution of algorithms (e.g. satellite navigation algorithms) that work with low-precision number representations. The approach taken has focussed on increasing the data throughput by decreasing operation latencies and by increasing data parallelism through using the SIMD approach. The activity has been structured as concurrent development of integer and floating-point instruction set extensions for LEON2-FT together with extending the existing compilation tools.

On the hardware side the activity has delivered a new highly-configurable FPU for LEON2-FT for floating-point computations, and a novel SIMD-within-a-register (SWAR) unit for integer computations.

The FPU can be configured at the synthesis time by the user to 21 distinct configurations so as to best fit the application needs and save implementation resources (FPGA slices, ASIC area). One of the 21 configurations is fully compatible with the Meiko FPU that is used in the AT697F ASIC. In this configuration the new FPU achieves slightly higher computation throughput than Meiko.

The SWAR unit can host a number of SWAR modules. These can implement user-defined custom computations on two 32-bit data words in full synchronization with the LEON2-FT integer pipeline. Six SWAR modules have been developed and delivered within the activity that target the domains of satellite navigation, audio and video processing. The design of the SWAR unit and SWAR modules is fully compatible with partial dynamic reconfiguration of FPGAs that may prove useful when the SWAR unit is implemented as an FPGA or eFPGA fabric configurable by the application at the run time. This design concept allows for introduction of a high number of additional SWAR modules that can be specified, designed (in VHDL) and implemented by the LEON2-FT end users.

On the software side the activity has delivered a new toolchain derived from the existing GNU binutils and LLVM compiler by implementing new assembly instructions in the binutils and support for new user-defined data types in the LLVM.

The binutils with daiteq extensions introduce new assembler instructions and machine operation codes for half precision floating-point operations, packed floating-point operations and SWAR operations. Emission of the new opcodes is controlled via new command-line switches that enable or disable the individual new features (e.g. floating-point opcodes for just one precision, or SWAR opcodes).

The LLVM compiler with daiteq extensions introduces a new half-precision (binary16) floating point type. In addition the LLVM compiler allows users to define half- or single-precision (binary32) data types that form two-element vectors, denoted also as the packed floating-point format, that are mapped to the packed floating-point opcodes that are supported in the packed-format daiFPU configurations (DAIFPU-PSP and DAIFPU-PHP).

For the SWAR types the LLVM compiler allows users to define new sub-32 bit integer types that can form vectors or arrays; these arrays are automatically partitioned and mapped to individual SWAR operations that work with SWAR values fitted inside standard 32-bit integer words. The SWAR operations are supported in the LEON2-FT SWAR unit (SWAR extensions).

On the application side the activity has delivered a set of benchmarks that can be used to evaluate floating-point performance of LEON-type processors, and a prototype implementation of a GNSS tracking loop that

has been used to evaluate the execution speed-up attributed to the SWAR instruction set extensions.

In addition, the activity has delivered four new technology mapping files for LEON2-FT that can be used to implement the processor in Xilinx Spartan6, Xilinx Virtex7, MicroSemi PolarFire and NanoXplore NG-Medium FPGAs.

The outputs have been documented in six project deliverables and a number of additional reports produced beyond the requirements of the contract.

Applicable documents

AD1 D05 - FPU Design Document, V1.9

AD2 D04 - ISE Design Document, V1.8

AD3 IEEE Std 754-2019 - IEEE Standard for Binary Floating-Point Arithmetic. June 13, 2019

AD4 SPARC, “The SPARC V8 specification”, 1992: http://www.sparc.com/standards/V8.pdf

AD5 D06 - SDE User Manual, V1.2

AD6 LEON2-FT daiFPU/SWAR User Manual, V1.4

Reference documents

RD1 The LEON2-FT Processor User’s Manual, Version 2014.1. July 15, 2014

Abbreviations

ASIC	application-specific integrated circuit
DP	double precision
GNSS	global navigation satellite system
HP	half precision
ISE	instruction set extensions
SIMD	single instruction, multiple data
SP	single precision
SWAR	SIMD within a register, or sub-word arithmetic
VHDL	VHSIC hardware description language

2 Motivation

This activity has been funded from the ESA GSTP programme. The developed IP core extensions – a configurable floating-point unit, and configurable instruction set extensions – focus on increasing the applicability of LEON2-FT to areas that are sensitive to power consumption as well as manufacturing cost of the device; these are namely space-related and consumer applications. Through an explicit support of flexibility in the manufactured device it is possible to further increase the efficiency of the user computation, visible through increased computation throughput, decreased power consumption and increased reuse value of the designs. This goal has been achieved namely through custom-designed processor extensions and extending the LLVM compiler. This approach has only recently been announced by manufacturers of other commercial embedded processors that are available on the market today, e.g. ARM and Synopsys ARC processor. Extensibility has also been one of the drivers for creating the quickly emerging RISC-V instruction set architecture [ris].

The configurable floating-point unit (FPU) is targeted to providing flexibility for the FPGA and ASIC technology used in satellite applications. The key advantage is the ability to increase the actual functional density of the silicon used on board of satellites in the context of the actual on-board computations. This is done through allowing the user to parameterize the FPU at the synthesis time in a way to ensure the correct function of the application while not using more resources than necessary. Classical LEON2-FT FPUs are based on fixed data bus widths of 32 or 64 bits, often in situations where a reduced precision would be sufficient (e.g. 16 bits), also with operations that may not be used in their application. Primarily the configurable FPU can be used as:

1. LEON2-FT implemented in an ASIC or FPGA device, with the FPU parameters specified at the design time and hard-configured in the ASIC.

The configurable LEON2-FT instruction set extensions (ISE) are targeted namely towards fixed-point applications that require only a sub-word precision, e.g. 3 bits (in LEON2-FT 1 word consists of 32 bits), such as satellite navigation applications or data encryption; the performance of LEON2-FT can be increased through an implementation of SIMD-like operations on variables that are merged in one 32-bit word, thus sharing the data-path circuitry for two or more operations executed in one clock cycle. The extensions can be used in three set-ups:

2. LEON2-FT implemented in an ASIC device, with the ISE specified at design time and hard-configured in the ASIC.
3. LEON2-FT implemented in an ASIC device that also contains an embedded FPGA (eFPGA), with the ISE specified after ASIC fabrication at the application compile time and configurable at the runtime in the eFPGA part of the ASIC.
4. LEON2-FT implemented in an FPGA device, with the ISE specified and configured either at the FPGA implementation time (full FPGA configuration) or after that at the application compile time (partial FPGA reconfiguration if supported by the FPGA device used).

The partial runtime reconfiguration of the device mentioned in points 4 and 5 above can also be used in flight to provide an increased functional density of LEON2-FT for on-board applications.

3 Architecture

3.1 Packed floating-point formats

Packed floating-point formats are supported in some daiFPU configurations. They are defined for pairs of floating-point values that are stored in

- a single register for two half-precision values stored in one single-precision floating-point register, or
- a register pair of two consecutive registers for two single-precision values stored in a pair of even-odd single-precision registers.

For packed word operations the result is computed as the selected operation performed independently on the upper sub-words and lower sub-words. Exceptions and flags are computed as logical *OR* of the exceptions and flags generated for the upper and lower word.

3.2 daiFPU

The developed floating-point unit, denoted as daiFPU, is an IEEE Std.754 (2019) compliant floating-point unit that supports binary64, binary32, binary16 formats and their combinations. The unit consists of a floating-point datapath and a floating-point controller. The datapath executes all floating-point arithmetic operations and format conversions. The controller manages data exchange between the LEON2-FT integer pipeline and the daiFPU. The controller also executes floating-point comparisons.

The user can select seven major daiFPU configurations at the synthesis time that support individual floating-point formats, their combinations, or packed floating-point formats. For each major configuration the user can specify whether floating-point division and floating-point square root should be supported. This in total gives 21 distinct daiFPU configurations.

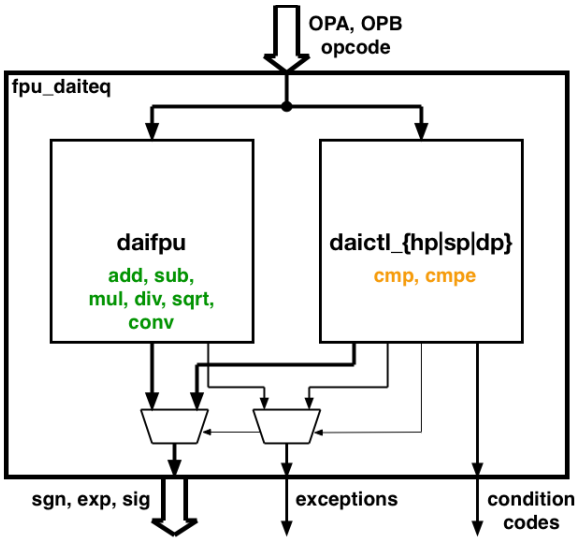

Figure 3.1: One-precision FPU - *fpu_daiteq.vhd* - a configuration that supports one precision.

The internal structure of the one-precision daiFPU is shown in Fig. 3.1. The internal structure of the *dual* and *packed* configurations is similar.

3.3 SWAR unit

The SWAR[1] instruction extensions are implemented as a new SWAR unit that is connected in parallel to the integer ALU in the LEON2FT integer pipeline (*iu.vhd*). The SWAR unit (see Fig. 3.2) contains one or more SWAR modules and an optional module with SWAR accumulators. The actual configuration of the SWAR unit can be selected by the user before LEON2-FT synthesis using the *make xconfig* mechanism (see [RD1]).

At present six SWAR modules have been developed that implement operations suitable for

- correlation of GNSS signals (sum of products for 1-4 bit data words),
- demodulation of GNSS signals (real and complex vector multiplication for 2-4 bit data words),
- sine / cosine lookup for GNSS demodulation (lookup of 1-4 bit values for 32 bit arguments)
- processing of audio signals (ADD, SUB, MUL with optional reduction for 16-bit words),
- processing of video signals (ADD, SUB, MUL with optional reduction for 8-bit words).
- generic ALU for sub 32 bit words (ADD, SUB, MUL with optional reduction for user-defined words).

The SWAR accumulator can be disabled or configured to fit the number of lanes and slice size of the SWAR modules:

- up to 16 independent accumulator registers,
- each accumulator register up to 64 bits wide.

The internal structure of the SWAR unit is shown in Fig. 3.2.

3.4 Software toolchain

The *Software Development Environment* (SDE) is a set of tools for compiling and building binary executables for applications written in C. The SDE support for the LEON2-FT with the daiFPU and SWAR unit is based on two standard packages for the SPARC architecture. One package is *the GNU binutils* and the other is *the LLVM Compiler Infrastructure*. The implemented SDE consists of two stand-alone packages. The first package is referred to as *the daiteq binutils* to distinguish it from the common GNU binutils package. The second package is referred to as *the daiteq LLVM*; it is the classical LLVM framework with extensions implemented in the C compiler and the SPARC backend.

The daiteq binutils is a set of tools for working with binary object code (e.g. assembler, linker, objcopy, objdump). The recent version (v2.34) of the *GNU binutils* has been used in the project.

The daiteq binutils are used the same way as the classical GNU binutils, therefore the existing documents for GNU binutils are applicable but for the changed parts that are covered in this document.

From a user point of view the binutils have been changed in two parts:

- the assembler tool, and

[1] SWAR stands for *SIMD-within-a-register*. The SWAR data types usually represent integer numbers encoded on fewer than the usual 32 bits.

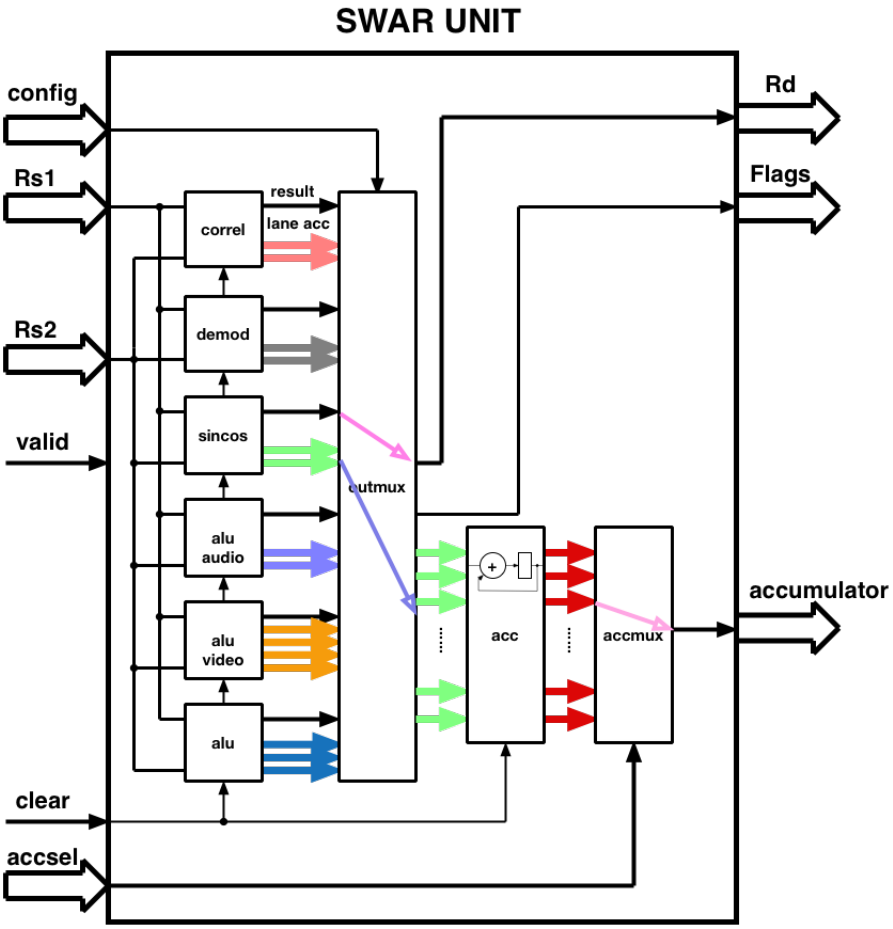

Figure 3.2: SWAR unit, configuration with up to six SWAR modules and a SWAR accumulator. *outmux* shows connections for the sincos operation selected in the SWAR configuration register, and *accmux* shows a connection for readout of SWAR accumulator #1 bit slice [31:0] selected in the SWAR accumulator selection register.

- the ELF file processing.

The assembler assembles a source code with CPU operations presented as a text description into an object code that contains sequences of a binary machine code which can be executed on the target CPU.

The daiteq binutils support new assembler instructions that have been added to the LEON2-FT processor in the 2020.1 release. These instructions are described in [AD1], [AD2] and [AD5]. In the binutils the new instructions have been assigned to all architectures that are based on the *SPARC V8* or *LEON* architectures.

The new instructions come in four groups:

- SWAR integer instructions[1] ,
- half-precision floating-point instructions,
- packed floating-point instructions, and
- complex floating-point instructions[2] .

Each group of inserted instructions has to be explicitly enabled using a corresponding command-line option when invoking the assembler tool.

The extended ELF file processing contains new hardware capability flags that correspond to new instruction groups. The hardware capability flags indicate if the generated binary code contains at least one instruction from the corresponding instruction group.

The daiteq LLVM extends the usual LLVM functionality in two distinct directions. The first adds a support for new, user-defined SWAR data types and the SWAR arithmetic operations. The SWAR operations are computed in the daiteq SWAR unit.

The second direction adds new floating-point data types and operations for the *binary16* precision and packed floating-point representations that are supported in certain configurations of the daiteq floating-point unit (*daiFPU*).

At the same time the new arithmetic extensions can also be used with legacy C compilers for SPARC/LEON (e.g. BCC1 or BCC2)[3] . In this case the user has to include the corresponding new assembly instructions in C sources by herself as needed. The selected legacy C compiler is then used to translate the C sources to assembler sources, and *the daiteq binutils* are used to generate binaries from the assembler sources.

The new data types (the floating-point *half*, *packed half*, *packed single* data types, and the integer SWAR data types) that are supported in the LEON2-FT with the daiteq FPU and the SWAR unit cannot be declared directly when using a legacy GCC compiler; the user has to map the new data types to the corresponding existing data types, usually *int* or a union of *float/int* or *double/int*[4] .

[2] The complex instructions are not supported in daiFPU. Efficient complex operations can be implemented in software using the packed floating-point instructions.

[3] The daiteq FPU in the configuration *DAIFPU-DUAL-DPSP-DIVSQRT* is fully compatible with the Meiko FPU, and can be used with legacy software toolchains, that is with the common GNU binutils *without the daiteq extensions*.

[4] Using unions of floating-point and integer values leads to lower performance due to increased use of load and store instructions in the compiled binary since in SPARC there are no instructions for direct data transfer between integer and floating-point registers.

4 Validation

4.1 daiFPU

Validation of the developed FPU has been performed in these steps:

1. Validation of individual FPU modules and operations in self-checking stand-alone testbenches. Test vectors were generated using the TestFloat tool that has been developed and distributed by John Hauser.
2. Validation of the LEON2-FT / FPU integration using a simple C program that applies a limited number of TestFloat vectors on the FPU inputs and compares the result with a reference result stored in the TestFloat vectors.
3. Validation of the LEON2-FT / FPU integration using the paranoia program originally developed by Prof. Kahan.
4. Validation of correct floating-point computation in LEON2-FT with the new FPU using the C-Ray benchmark, and comparing the computed results with results computed by C-Ray on a desktop PC.

4.2 SWAR unit

The SWAR unit has been validated in several independent ways:

1. By hand-transforming the GNSS tracking loop code to work with 2-, 3- and 4-bit values, and by transforming elementary computation kernels into SWAR modules while verifying that the tracking loop can still decode bits of the navigation message. First the SWAR modules were represented by their functional models in C. The functional models were then replaced by actual SWAR modules implemented in VHDL.
2. By validating SWAR modules developed in VHDL against their functional models writted in C.
3. By validating LEON2-FT execution of the tracking loop, using the SWAR functional models in C or the actual SWAR modules in VHDL against desktop execution of the tracking loop using the SWAR functional models in C.

4.3 Software toolchain

The daiteq binutils and the daiteq LLVM were validated in a functional simulator of the LEON2-FT with the new extensions, ModelSim and hardware. The testing was carried out in the following test phases:

1. Simple assembler programs were used to validate correct translation of assembler instructions to binaries and their dissassembling. The programs used all the new assembler instructions
2. Legacy C programs with inline assembler instructions were compiled with a legacy C compiler (e.g. BCC1) to assembler programs. The assembler programs were translated to program binaries with the daiteq binutils. The C programs included new assembler instructions through inline assembler statements. Both the new floating-point types and SWAR types were covered. The programs checked correct execution of the new instructions for a number of input operands, or test vectors, using either

golden reference output vectors or golden reference functional models written in C to verify the correct function of each instruction.

3. C programs with the new data types were compiled with the daiteq LLVM and the program binaries generated with the daiteq binutils. The C programs included the new floating-point and integer data types supported in the daiFPU and the SWAR unit. The new assembler instructions were generated by the LLVM compiler. Both the new floating-point types and SWAR types were covered. The programs checked correct execution of the new instructions for a number of input operands, or test vectors, using either golden reference output vectors or golden reference functional models written in C to verify the correct function of each instruction.
4. More complex C programs with the new floating-point and SWAR data types, like prime number generation, FIR filtering or Mandelbrot set generation, were compiled with the daiteq LLVM and the program binaries generated with the daiteq binutils.

5 Evaluation

5.1 Tools used

The tools used for compilation and generation of the binary files used in this section consisted of

- the daiteq binutils, derived from GPU binutils 2.34,
- the daiteq LLVM compiler, derived from LLVM 10.0,
- the SoftFloat library, version 3e, written by John Hauser,
- the daiteq math library, derived from OpenLibm.

More details about the toolchain are provided in [AD5].

5.2 daiFPU

The daiFPU achieves slightly better performance than the Meiko FPU, which can be observed in Tables 5.1 to 5.4.

Table 5.1: Whetstone, normalized performance for daiFPU and Meiko.

Whetstone - normalized performance kWIPS/MHz				
Optim.	daiFPU		Meiko	
flag	DP	SP	DP	SP
daifpu_dual_dpsp_divsqrt				
-O0	163.52	222.72	147.02	221.3
-O1	210.09	323.35	196.73	332.21
-O2	285.85	427.58	261.33	444.67
-O3	286.47	424.82	261.68	445.71
daifpu_dual_dpsp_divonly				
-O0	158.45	212.86	134.21	198.14
-O1	201.11	298.47	172.37	285.87
-O2	269.25	385.44	223.46	353.02
-O3	267.8	382.52	223.72	353.67
daifpu_dual_dpsp_none				
-O0	88.95	124.61	44.56	74.05
-O1	103.35	148.79	48.91	78.92
-O2	187.48	267.14	108.66	165.94
-O3	187.85	264.92	103.92	166.08

Table 5.2: Linpack, normalized average performance for daiFPU and Meiko.

Linpack - normalized average performance [kFLOPS/MHz]					
Optim.	Linpack	daiFPU		Meiko	
flag	version	DP	SP	DP	SP
daifpu_dual_dpsp_divsqrt					
-O0	ROLLED	31.93	36.63	28.12	39.57
-O1	ROLLED	49.57	57.97	48.11	65.89
-O2	ROLLED	51.58	66.68	49.12	83.03
-O3	ROLLED	51.24	66.49	49.25	83.49
-O0	UNROLLED	38.58	46.34	33.77	50.89
-O1	UNROLLED	55.07	65.67	48.36	69.39
-O2	UNROLLED	55.9	72.56	49.37	83.58
-O3	UNROLLED	55.51	72.32	49.51	84.07
daifpu_dual_dpsp_divonly					
-O0	ROLLED	31.93	36.63	28.12	39.57
-O1	ROLLED	49.57	57.97	48.11	65.89
-O2	ROLLED	51.58	66.68	49.12	83.03
-O3	ROLLED	51.24	66.49	49.25	83.49
-O0	UNROLLED	38.58	46.34	33.77	50.89
-O1	UNROLLED	55.07	65.67	48.36	69.39
-O2	UNROLLED	55.9	72.56	49.37	83.58
-O3	UNROLLED	55.51	72.32	49.51	84.07
daifpu_dual_dpsp_none					
-O0	ROLLED	31.7	36.29	27.83	39.21
-O1	ROLLED	49.25	57.68	47.22	64.94
-O2	ROLLED	51.21	66.32	1.59	81.61
-O3	ROLLED	50.85	66.11	48.37	82.05
-O0	UNROLLED	38.28	46.03	33.35	50.3
-O1	UNROLLED	54.7	65.28	47.44	68.29
-O2	UNROLLED	55.51	72.74	2.11	82.08
-O3	UNROLLED	55.06	71.76	48.61	82.5

Table 5.3: Stanford composite results for daiFPU and Meiko.

Stanford	daiFPU				Meiko			
time[ms]	DP		SP		DP		SP	
Opt.flags	non FP	FP	non FP	FP	non FP	FP	non FP	FP
divsqrt								
-O0	86	132	86	126	95	147	95	138
-O1	45	69	45	70	52	81	52	81
-O2	28	50	27	52	27	55	27	55
-O3	27	48	27	50	26	53	26	53
divonly								
-O0	86	131	86	127	94	147	95	138
-O1	45	69	45	70	52	81	52	81
-O2	28	50	27	52	27	55	27	55
-O3	27	48	27	50	26	53	26	54
none								
-O0	86	132	86	127	95	150	95	139
-O1	45	69	45	71	52	82	52	81
-O2	27	50	28	52	27	56	27	55
-O3	26	48	27	50	26	54	26	53

Table 5.4: C-ray performance for daiFPU and Meiko.

C-Ray - time to render one pixel [ms]					
Optim.	Image	daiFPU		Meiko	
flag	resolution	DP	SP	DP	SP
daifpu_dual_dpsp_divsqrt					
-O0	100x75	4.4779	1.6984	4.9841	1.6787
-O1	100x75	1.8585	0.9907	1.8325	0.8788
-O2	100x75	1.2369	0.7847	1.3237	0.6952
-O3	100x75	1.2379	0.7844	1.3237	0.6952
-O0	200x150	4.4496	1.6667	4.9840	1.6788
-O1	200x150	1.8269	0.9942	1.8326	0.8798
-O2	200x150	1.2370	0.7846	1.3239	0.6952
-O3	200x150	1.2379	0.7843	1.3239	0.6952
-O0	400x300	4.4727	1.6908	.	.
-O1	400x300	1.8517	0.9816	.	.
-O2	400x300	1.2295	0.7767	.	.
-O3	400x300	.	.	.	.
daifpu_dual_dpsp_divonly					
-O0	100x75	4.6732	1.8291	5.5491	1.9360
-O1	100x75	2.0204	1.1255	2.2579	1.1801
-O2	100x75	1.3921	0.9343	1.7364	0.9897
-O3	100x75	1.3896	0.9192	1.7439	0.9897
-O0	200x150	4.6449	1.8283	5.5554	1.9361
-O1	200x150	1.9934	1.1255	2.2580	1.1802
-O2	200x150	1.3922	0.9233	1.7491	0.9898

Continued on next page

Table 5.4 – continued from previous page

C-Ray - time to render one pixel [ms]					
Optim.	Image	daiFPU		Meiko	
flag	resolution	DP	SP	DP	SP
-O3	200x150	1.3897	0.9192	1.7368	0.9898
-O0	400x300	4.6678	1.7969	.	.
-O1	400x300	2.0461	1.1180	.	.
-O2	400x300	1.3847	0.9105	.	.
-O3	400x300	.	.	.	.
daifpu_dual_dpsp_none					
-O0	100x75	5.4147	2.2395	7.4891	2.9645
-O1	100x75	2.5708	1.5233	3.9139	2.1657
-O2	100x75	1.9607	1.3233	3.4047	2.0065
-O3	100x75	1.9479	1.3249	3.4052	1.9836
-O0	200x150	5.3868	2.2081	7.4899	2.9650
-O1	200x150	2.5395	1.4919	3.9147	2.1662
-O2	200x150	1.9247	1.3235	3.4056	2.0129
-O3	200x150	1.9166	1.3252	3.4060	1.9841
-O0	400x300	5.4100	2.2328	.	.
-O1	400x300	2.5649	1.5162	.	.
-O2	400x300	1.9415	1.3158	.	.
-O3	400x300	.	.	.	.

5.3 SWAR unit

The SWAR unit can increase the performance of the GNSS tracking loop by about **5x**. A joint benefit of using the daiFPU with the SWAR unit over using a LEON2-FT without any FPU is about **382x**. This is documented in Table 5.5. The table shows results for iteration 0 and iteration 1 of given configurations. Iteration 0 corresponds to a cold instruction cache, while Iteration 1 in a way corresponds to a warmed instruction cache. Due to the dynamic nature of the processed data nothing can be said about the data cache.

The following configurations have been analysed (not all are listed in the table):

---

1. **Octave-equivalent** [BAB+07], [Roj11], i.e. all samples and computations in floating point, original USRP file, buffered execution (store all intermediate results in arrays like in Octave). Computed correlation results are identical to those computed in Octave.

---

2. Like the previous step, but navigation samples and sine/cosine **values quantized to 2 bits**.

---

3. Samples quantized to 2b values, **integer arguments** for spreading code expansion and carrier generation, expand separate E, P, L codes, do not use SWAR instructions, buffered execution
4. Like the previous step, but use **SWAR for demodulation**
5. Like the previous step, but use **SWAR also for demodulation and correlation**
6. Like the previous step, but use **SWAR also for sine/cosine lookup**

7. Samples quantized to 2b values, integer arguments for spreading code expansion and carrier generation, expand separate E, P, L codes, use SWAR for sine/cosine lookup, demodulation and correlation, **fused carrier generation and demodulation**. Navigation samples are read from the FIFO and stored in a buffer at the beginning of the processing.
8. Like the previous step, but **all processing steps fused**, i.e. carrier generation, demodulation and correlation. Navigation samples are read from the FIFO when needed (without buffering).
9. Like the previous step, but expand **just one spreading code** and use it for all E, P and L codes
10. Like the previous step, but use **HW support for spreading code expansion**

Table 5.5: GNSS tracking loop - execution times for the main processing steps.

Config	FIFO	Codes	Carrier	Demodulation	Correlation	Total
/Iter	[us]	[us]	[us]	[us]	[us]	[us]
A - Original reference code, 2-bit values, SoftFloat						
2 / 0	63‘942	4‘874‘978	9‘624‘254	56‘033	108‘136	14‘530‘587
2 / 1	63‘943	4‘874‘998	9‘756‘074	56‘033	108‘136	14‘862‘450
B - like A plus SWAR instructions and daiFPU						
5 / 0	2‘343	82‘868	105‘190	9‘186	5‘771	208‘005
5 / 1	2‘125	82‘849	105‘185	9‘190	6‘077	205‘427
6 / 0	2‘430	82‘689	20‘601	9‘182	6‘204	121‘286
6 / 1	2‘125	82‘935	20‘595	9‘187	6‘074	120‘915
C - like B plus fused steps						
7 / 0	2‘413	83‘874		20‘753	5‘812	113‘396
7 / 1	2‘207	83‘844		20‘753	5‘809	112‘914
8 / 0		83‘873		20‘129	6‘116	110‘118
8 / 1		84‘847		20‘131	5‘943	109‘920
D - like C plus just one expanded code						
9 / 0		27‘772		18‘245	4‘366	50‘802
9 / 1		27‘751		18‘234	4‘366	50‘625
10 / 0		15‘286		18‘300	4‘501	38‘502
10 / 1		15‘269		18‘286	4‘502	38‘451

5.4 Eficiency of the software toolchain

Simple programs that used the SWAR data types showed that the SWAR data types are useful for computations with repetitive operations on a data vector rather than for single operations, at least because the SWAR unit must be configured before each different operation[5] . The performance advantage of the SWAR data types and operations for an online FIR filter is shown in Fig. 5.1.

The SWAR accumulators and the optional SWAR computation with saturation or reduction provide a big performance advantage that should be used whenever possible. Results from all the evaluated examples show that computations with very short data vectors stored in SWAR variables can be worse than using

[5] Setting the configuration requires at least two processor instructions.

classical data types. In the cases that use processor intrinsic data types and compiler optimization the difference can be even more significant.

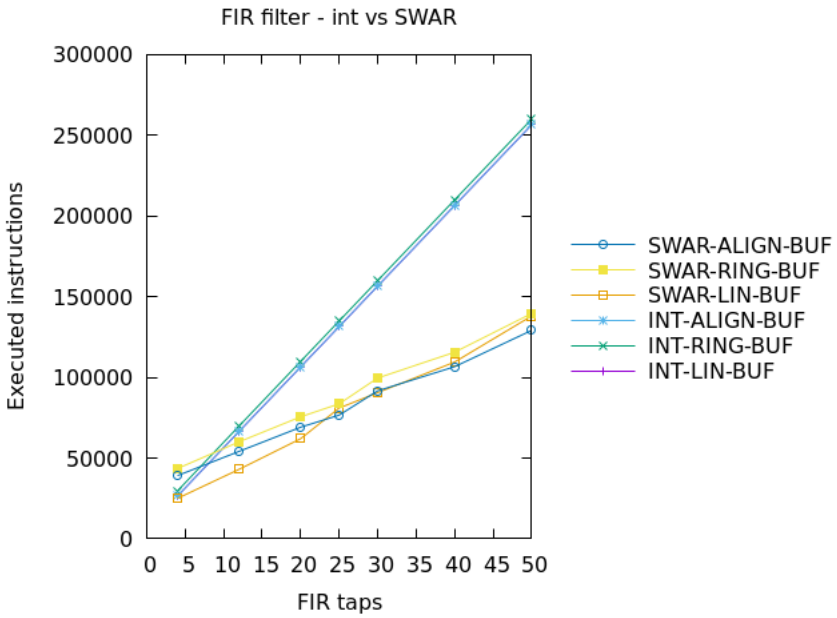

Figure 5.1: Online FIR filter - number of executed instructions for integer vs. SWAR implementations with a linear buffer, ring buffer and ring buffer aligned to 2^n. Sources compiled with the daiteq LLVM toolchain, optimization level *-O2*.

The use of the packed-half and packed-float data types in C sources compiled with the daiteq LLVM toolchain and executed in daiFPU provides a performance advantage of **23-36%** (also demonstrated in Fig. 5.2).

More details can be found in [AD5].

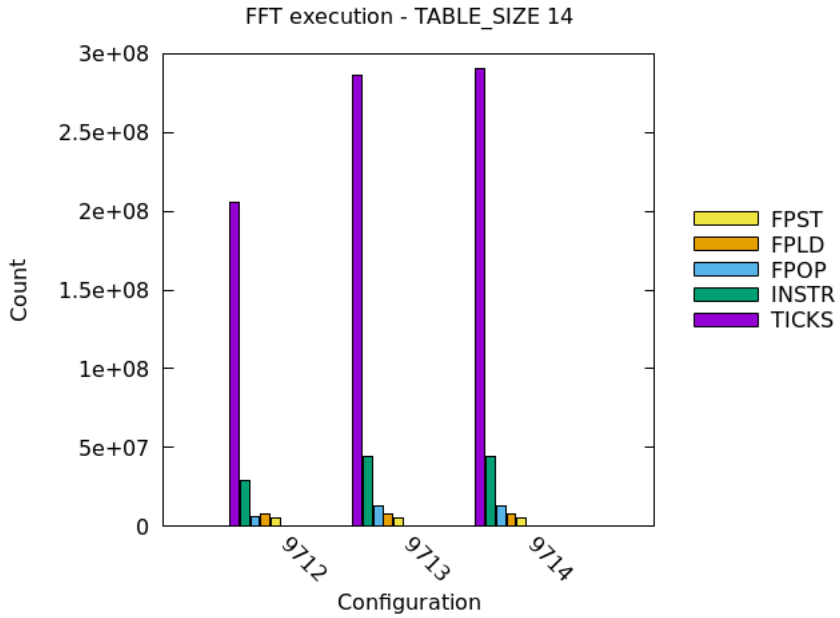

Figure 5.2: FFT kernels - execution profiles for hard-float, TABLE_SIZE=14, float (9713, 9714) vs. packed-float (9712) implementation. Shown are processor ticks, executed instructions, floating-point operations, floating-point loads and stores to compute FFT kernels.

6 Implementation characteristics

6.1 daiFPU

Resource requirements of the daiFPU are demonstrated for the Xilinx Virtex7 FPGA. Table 6.1 lists resource requirements for all the supported daiFPU configurations, while Table 6.2 lists the maximal operating frequency for LEON2-FT with the daiFPU.

Table 6.1: daiFPU - XC7V - FPU-only, resources used.

Flavour	Slices	Slice regs	LUTs	LUTRAM	DSP48E1
daifpu-dual-dpsp					
divsqrt	3777	3402	9446	385	15
divonly	3222	2920	8119	362	15
none	2237	2587	6719	279	15
daifpu-dual-sphp					
divsqrt	2197	2197	5239	150	2
divonly	1800	1988	4516	148	2
none	1629	1824	3700	121	2
daifpu-dp					
divsqrt	2523	2585	6202	324	15
divonly	2073	2259	5257	296	15
none	1719	1917	4205	233	15
daifpu-sp					
divsqrt	1381	1534	3277	164	2
divonly	1109	1393	2816	106	2
none	920	1189	2331	109	2
daifpu-hp					
divsqrt	786	947	1816	73	1
divonly	655	875	1526	60	1
none	546	783	1301	57	1
daifpu-psp					
divsqrt	2867	2921	6604	303	4
divonly	2233	2732	5742	213	4
none	1531	2332	4678	211	4
daifpu-php					
divsqrt	1439	1791	3636	159	2
divonly	1182	1740	3047	116	2
none	1099	1445	2611	121	2

Table 6.2: daiFPU - XC7V - maximal frequency.

Flavour	Target[MHz]	Final[MHz]
nofpu		
.	120	120
daifpu-dual-dpsp		
divsqrt	120	102
divonly	120	105
none	120	106
daifpu-dual-sphp		
divsqrt	120	118
divonly	120	120
none	120	120
daifpu-dp		
divsqrt	120	115
divonly	120	116
none	120	98
daifpu-sp		
divsqrt	120	114
divonly	120	118
none	120	116
daifpu-hp		
divsqrt	120	111
divonly	120	114
none	120	120
daifpu-psp		
divsqrt	120	112
divonly	120	105
none	120	110
daifpu-php		
divsqrt	120	118
divonly	120	120
none	120	111

6.2 SWAR unit

Resource requirements of the daiFPU are demonstrated for the Xilinx Virtex7 FPGA. Table 6.3 lists resource requirements for representative SWAR configurations, while Table 6.4 lists the maximal operating frequency for LEON2-FT with the SWAR unit.

Table 6.3: SWAR unit - XC7V - SWAR unit, resources used.

Flavour	Slices	Slice regs	LUTs	LUTRAM	DSP48E1
swarall	870	253	2941	0	9
swaralu	91	95	331	0	3
swaraudio	92	98	310	0	2
swargnss	736	85	1816	0	0
swargnss-2b	261	83	696	0	0
swargnss-3b	131	78	760	0	0
swargnss-4b	308	82	897	0	0
swarvideo	136	139	456	0	4

Table 6.4: SWAR w/ daiFPU configuration DAIFPU-DUAL-DPSP w/ FDIV and FSQRT - XC7V - processor core, maximal frequency.

Flavour	Target[MHz]	Final[MHz]
no swar	120	102
swarall	120	94
swaralu	120	103
swaraudio	120	106
swargnss	120	105
swargnss-2b	120	103
swargnss-3b	120	107
swargnss-4b	120	96
swarvideo	120	103

7 Tools

7.1 daiteq binutils

The daiteq binutils can be downloaded using the following commands:

```
$ git clone https://devsrv.daiteq.com/martin/daiteq-binutils.git
$ cd daiteq-binutils
$ ./run.sh all
```

The compiled binary files will be located in *daiteq-binutils/install/bin*. Add this path to the $PATH variable.

```
$ export PATH=`pwd`/install/bin:$PATH
$ cd ..
```

Modify your Makefiles to use the prefix *sparc-daiteq-elf-* instead of, for example, the common *sparc-elf-*.

Test if the toolchain is accessible

```
$ sparc-daiteq-elf-as --version

GNU assembler (GNU Binutils) 2.34
Copyright (C) 2020 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or later.
This program has absolutely no warranty.
This assembler was configured for a target of `sparc-daiteq-elf'.
```

7.2 daiteq LLVM

C programs with the new integer and floating-point types for the daiFPU and the SWAR unit can be compiled using the daiteq LLVM compiler. Download and install the daiteq LLVM using the following commands:

```
$ git clone https://devsrv.daiteq.com/martin/daiteq-llvm10.git
$ cd daiteq-llvm10
$ ./run.sh all
```

The compiled binary files will be located in *daiteq-llvm10/install/bin*. Add this path to the $PATH variable.

```
$ export PATH=`pwd`/install/bin:$PATH
$ cd ..
```

7.3 daiteq demo examples

The new toolchain is accompanied by example programs written in C and assembler to demonstrate the use of the new floating-point and integer data types and operations. The examples can be downloaded as a git repository by executing

```
$ git clone https://devsrv.daiteq.com/martin/daiteq-demo.git
```

Instructions for building and executing the examples can be found in the *daiteq-demo/README* file, e.g. by typing

```
$ cd daiteq-demo
$ less README
```

The simplest way to build and execute the examples is by running the shell script `test.sh` as follows:

```
$ cd examples
$ ./test.sh build ./test_list.txt
```

The compiled binaries will be stored in the directory `BUILD`.

8 Deliverables and reports

The following deliverables and reports have been produced within this contract.

Deliverables

- ISE Specification (D01)
- FPU Specification (D02)
- FPU Survey Report (D03)
- ISE Design Document (D04)
- FPU Design Document (D05)
- SDE User Manual (D06)
- Final Report (FR)

Databases

- Benchmarking framework (DB1)
- leon2ft_2020.1_daifpu_swar (DB2, DB3)
- standalone_swar_leon2ft_2020.1_daifpu_swar (DB2)
- standalone_daifpu_leon2ft_2020.1_daifpu_swar (DB3)
- Extended GNU binutils (DB4)
- Extended LLVM (DB4)

CCN deliverables

- NG-Medium Techmap Report (CCN/D3)

Additional items

- LEON2-FT daiFPU/SWAR User Manual (IEUM020ESAIP0010)
- Tracking Loop Document (IETN018ESAIP004)
- BMFWK Document (IETN018ESAIP005)
- Prototype GNSS tracking loop (SW)
- tests-leon2ft (SW for leon2ft_2020.1_daifpu_swar)
- test-set (DATA for standalone_daifpu_leon2ft_2020.1_daifpu_swar)
- codes (DATA for the Prototype GNSS tracking loop)
- datasets (DATA for the Prototype GNSS tracking loop)

9 Conclusions

This activity has produced extensions to the LEON2-FT IP Core that significantly increase the potential of the LEON2-FT processor for on-board data processing, e.g. for the GNSS domain the demonstrated speedup is **382x**[6]. The use of the packed floating-point operations has demonstrated a speedup in the range of **23-35%**[7]. The developed toolchain provides means for programmers to fine-tune integer and floating-point data types used in applications, and ensures efficient processing of these data types in the daiFPU and in the SWAR unit.

The development of the prototype GNSS tracking loop, derived from [BAB+07] and [Roj11], has provided a valuable insight into proper identification of computing kernels in this class of algorithms, and also an insight into efficient mapping of floating-point values to integer ranges for range reduction of function arguments.

The developed IP cores are advertised on daiteq's website [dai].

The developed IP cores will be ported to the NOEL RISC-V processor.

The new software development tools can be downloaded from https://www.daiteq.com/en/software .

[6] speedup related to leon2ft_2015.3_nomeiko
[7] speedup related to implementations that use standard SPARC V8 floating-point operations

10 List of tables

11 List of figures

Bibliography

[ris] RISC-V: The Free and Open RISC Instruction Set Architecture. accessed on 2020-09-27. URL: http://riscv.org.

[dai] http://www.daiteq.com. Accessed: 2020-09-02.

[BAB+07] K. Borre, D.M. Akos, N. Bertelsen, P. Rinder, and S.H. Jensen. *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach*. Applied and Numerical Harmonic Analysis. Birkhäuser Boston, 2007. ISBN 9780817645403.

[Roj11] Cristian Paul Peñaranda Rojas. SoftGNSS - octave version. 2011. accessed on 2018-06-13. URL: https://github.com/kristianpaul/SoftGNSS/.

Revision

Date	Author	Description
2020/09/02	M.Daněk	Initial version
2020/09/27	M.Daněk	Document updated based on feedback from the FR
2021/04/16	M.Daněk	Updated information on the SDE toolchain

Disclaimer: