New Arithmetic Extensions for LEON2

ESA contract 4000122242/17/NL/LF

Presenter Martin Daněk
daiteq
martin@daiteq.com

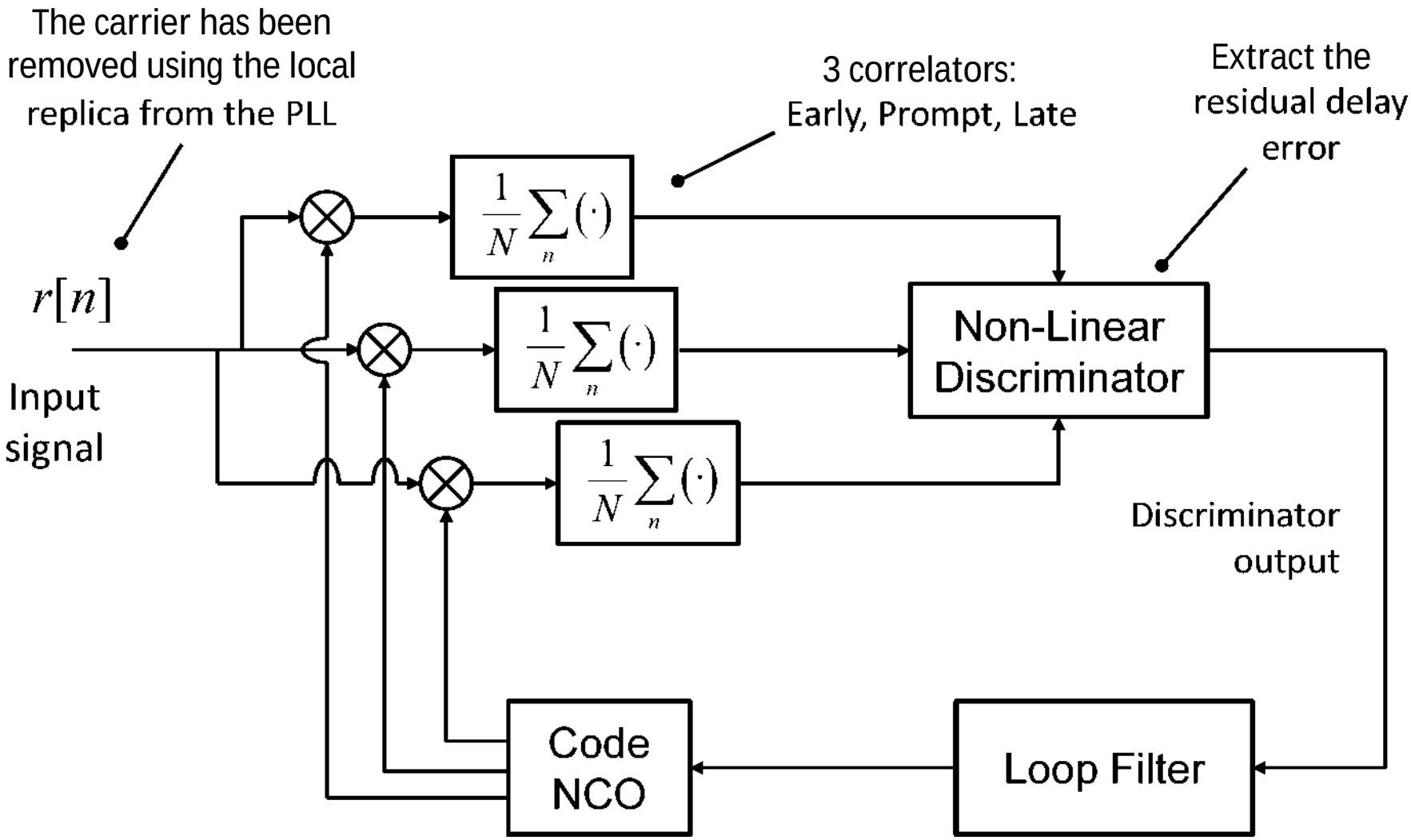
ADCSS2019
November 13, 2019

Problem

- Acceleration of SDR for GNSS in LEON2
  - Floating-point arithmetic w/ configurable precision
  - Fixed-point arithmetic w/ 2,3,4,…-bit packed integers

à Precise orbit determination with Galileo or GPS

Case study: GNSS tracking loop

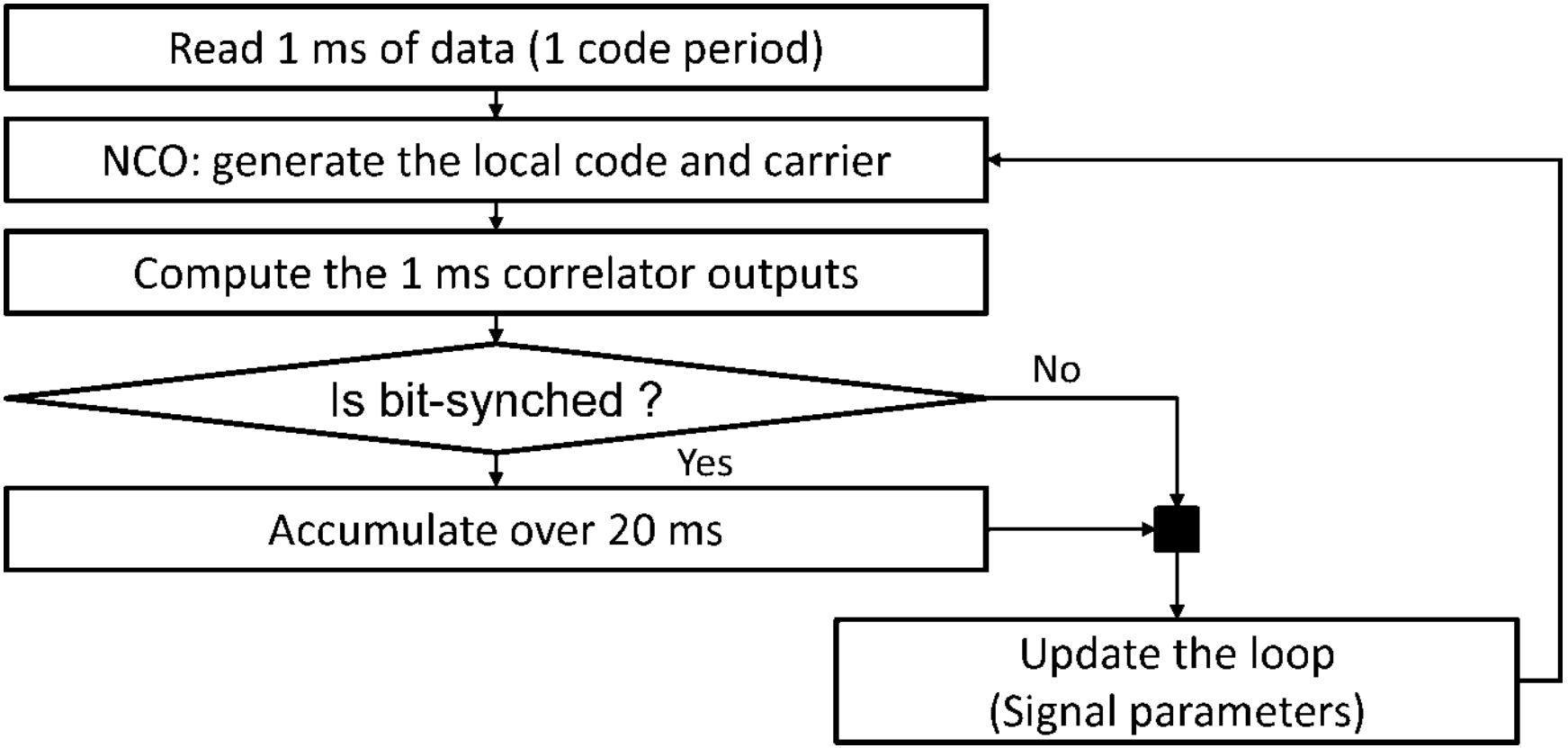


Structure of the code

Initialization

Main processing loop:

at each epoch 1 ms of data is read and processed

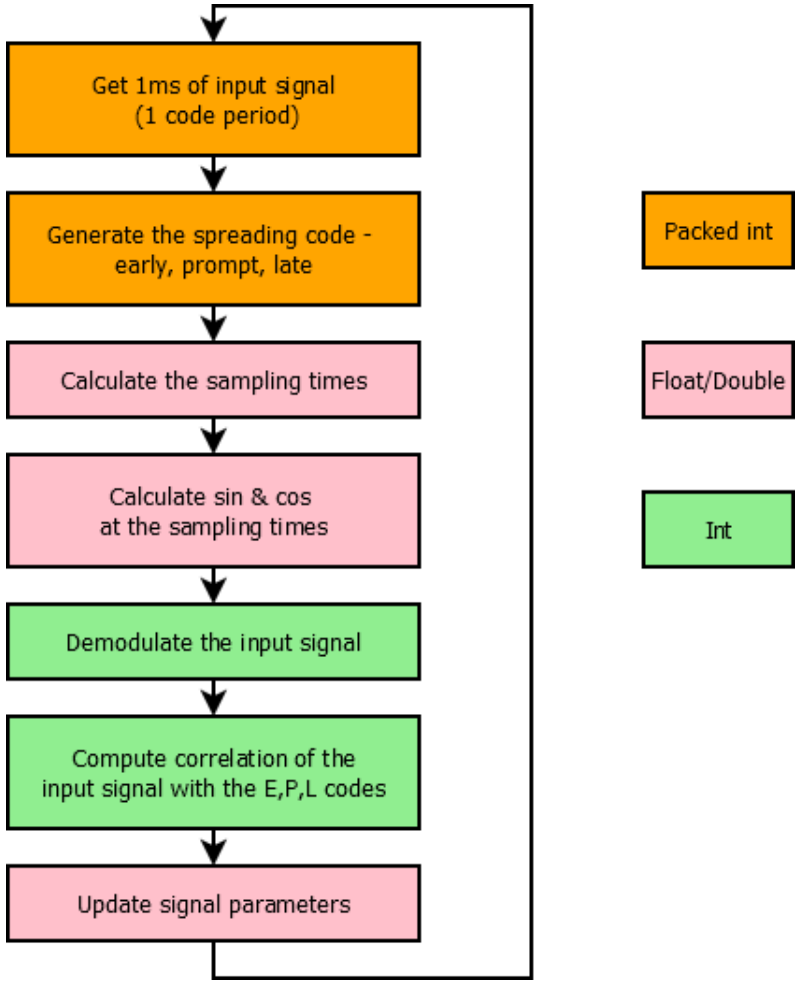


Modifications and optimizations

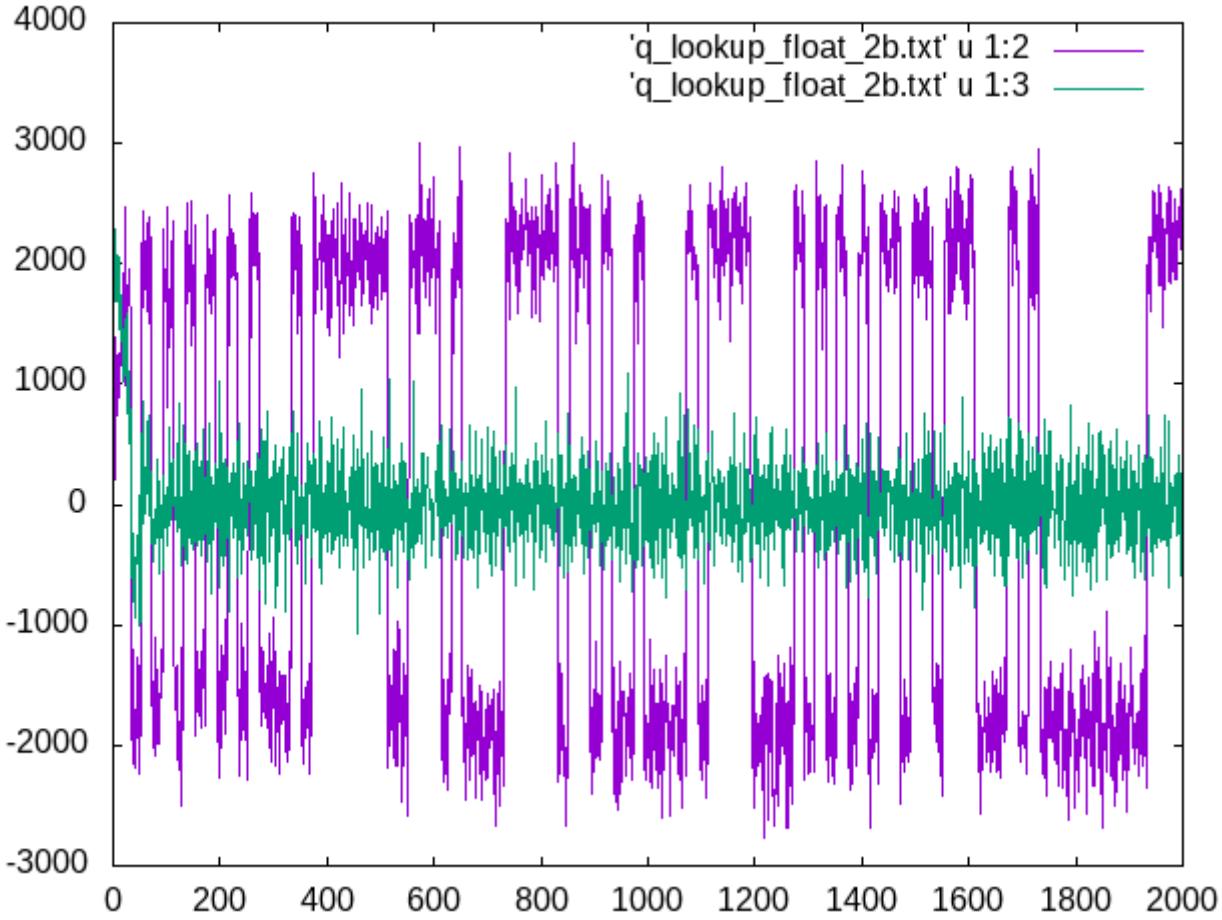
- Pre-processing: save memory and/or time
  - Input signal quantized and converted to packed int (memory)
  - Pre-generated spreading codes in packed int (memory & time)

- Processing: save time
  - All easy computations converted from double to int/unsigned
  - Sine / Cosine discretized using lookup w/ 8 thresholds

Structure of the code



Sample I/Q decoding



1 point is computed from approx. 25000 RF GNSS samples

Tracking loop profiling – execution time

Computation of 1 ms of data	Blksize – Inst count [s]	One value – Inst count [s]	One value – Exec cycles [1]
Signal input (FIFO)	0.001653400	6.6136e-8	1.65
Signal unpacking	0.06228932	2.49157e-6	62.29
One spreading code	1.40639172	5.62557e-5	1406.40
Sampling times	3.26883952	1.30754e-4	3268.85
All sine waves	6.488456840	2.59538e-4	6488.45
One sine wave point		1.29769e-4	3244.23
All demodulation	0.05603316	2.24133e-6	56.03
One signal value		1.12066e-6	28.02
All correlations	0.10813604	4.32544e-6	108.14
One correlation pt		7.20907e-7	18.03
TOTAL	14.86242812		219’106’131

Comparing current SW-only execution to real-time

Tracking loop:

- 1 ms real-time = 15 s LEON2 (SoftFloat)
- LEON2 15000x slower than real signal

Context: GPS L1 C/A

Chip rate vs. symbol rate:

1 symbol = 1023 chips @ 1.023 Mcps = 1 ms

Navigation message (50 bps):

- 1 nav. message = 25 frames = 12.5 min
- 1 frame = 5 sub-frames = 30 s
- 1 sub-frame = 10 words = 6 s
- 1 word = 30 symbols = 0.6 s

Context: Galileo E5a-I

Chip rate vs. symbol rate:

1 symbol = 2x10230 chips @ 10.230 Mcps = 20 ms

Navigation message (F/NAV, 50 bps):

- 1 frame = 12 sub-frames = 600 s
- 1 sub-frame = 5 pages = 50 s
- 1 page = 500 symbols = 10 s

Tracking loop profiling – instructions

Computation	Blksize – Inst count [1]	One value – Inst count [1]	One value - Exec cycles [1]	CPI [1]
Signal input (FIFO)	31612	1.264	1.65	1.31
Signal unpacking	1291014	51.641	62.29	1.21
One spreading code	18606236	744.249	1406.40	1.89
Sampling times	63270490	2530.820	3268.85	1.29
All sine waves	89589974	3583.599	6488.45	
One sine wave point		1791.799	3244.23	1.81
All demodulation	500407	20.016	56.03	
One signal value		10.008	28.02	2.80
All correlations	1201717	48.069	108.14	
One correlation pt		8.011	18.03	2.25
TOTAL	219’106’131		371’560’703	1.70

Where to gain performance?

- SoftFloat à HW FPU
- Improved CPI
- Customized arithmetic instructions, e.g. convolution - SIMD-within-a-register (SWAR)
- HW support for signal packing/unpacking

Possible speed-up in LEON2

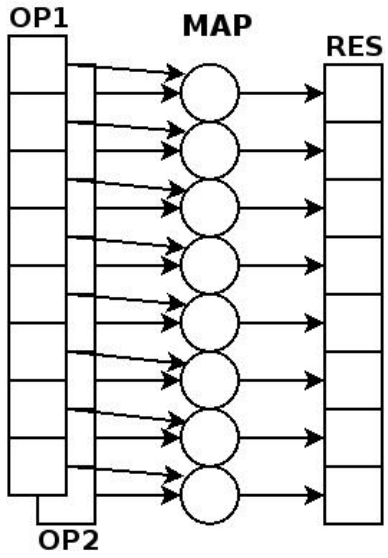
Feature	Reduction factor – Instructions	Reduction factor – cycles
HW FPU	10x	5x
HW unpacking – signal	50x	50x
HW unpacking – code	700x	1200x
Sub-word arithmetic (SWAR)	8x – 32x	32x – 128x
Half-precision FPU	20x	10x

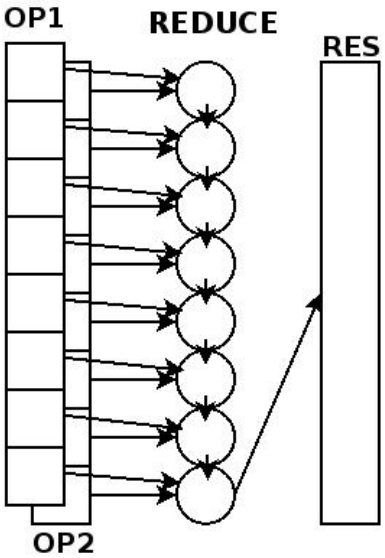
- è Speedup due to HW FPU vs SoftFloat: 4x (measured: AT697F)
- è Speedup due to HW FPU and packed integers: 11x (measured: ModelSim)
- è Execution time w/ new IP cores: ~1.39s (orig. was 14.862s)

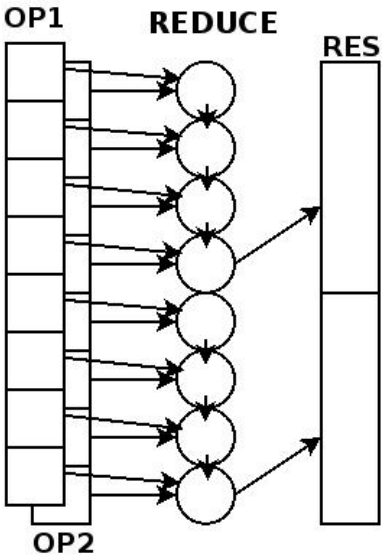
Improvements implemented

- Updated LEON2FT
  1. New packed integer processing - SIMD-within-a-register (SWAR)
  2. New family of blocking FPUs - daiFPU
     1. Two precisions, e.g. SP+DP=Meiko replacement.
     2. One precision only - DP, SP, HP
     3. Packed types in one precision - PHP=(HP,HP), PSP=(SP,SP)
  3. New tech maps for XC6S, XC7V, MPF, NG-Medium.
- Updated binutils
- C/C++ toolchain based on LLVM with new integer and floating-point types

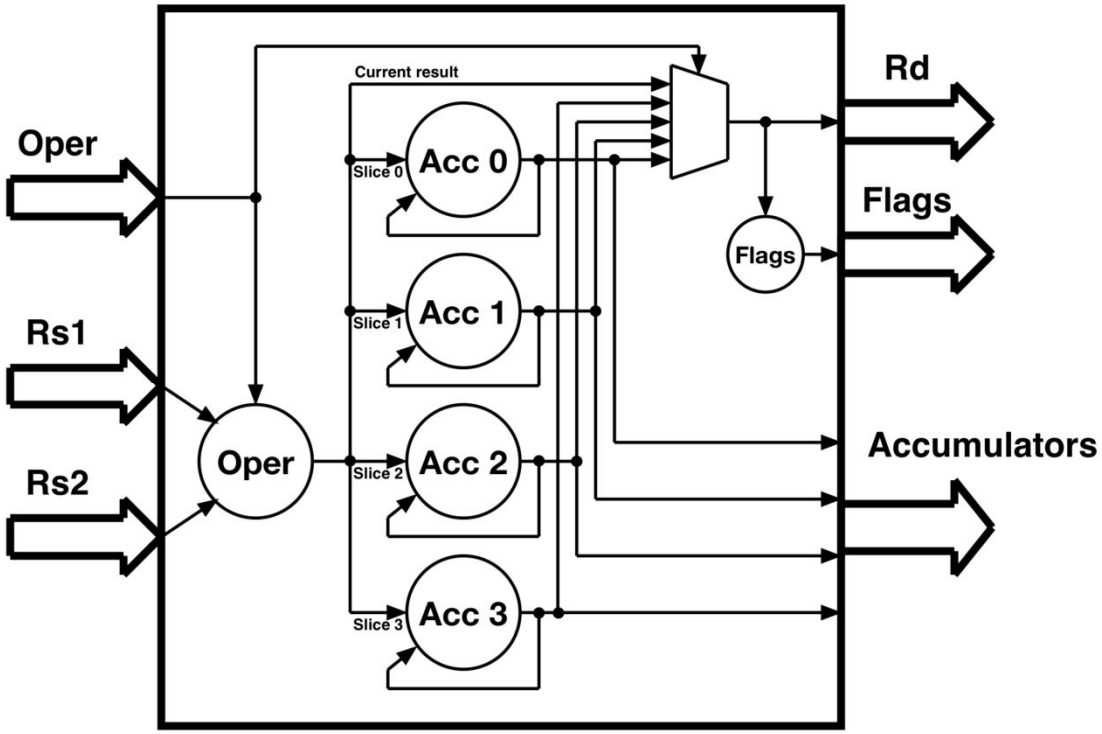
Application view: sub-word arithmetic







RTL view: generic SWAR module



Definition of SWAR types

```
/* cover swar types for systems without swar extension */
#if USE_SWAR
        typedef unsigned int subU4b  __attribute__((__subword(4)));
        typedef unsigned int sub8U4b  __attribute__((__subword__(4, 8)));
#else
        typedef unsigned int subU4b[1];
        typedef unsigned int sub8U4b[8];
#endif

#if USE_SWAR
        typedef unsigned int subU4b  __attribute__((__subword(4)));
        typedef unsigned int sub8U4b  __attribute__((__subword__(4, 8)));
#else
        typedef unsigned char subU4b[1];
        typedef unsigned char sub8U4b[8];
#endif
```

Definition of SWAR types (cont'd)

```
/* two ways how to define vector with 1 item (size = 2bit)  */
typedef unsigned int s1x2b __attribute__((subword(2)));
typedef unsigned int s1x2b_alter __attribute__((subword(2, 1)));

/* define type of swar vector of two-bits items in one U32 word  */
typedef unsigned int s16x2b __attribute__((subword(2, 16)));

/* define type of swar vector of one-bits items */
typedef unsigned int s16x1b __attribute__((subword(1, 16)));

/* define type of long swar vector (vector with 50 two-bits items organized in
 * unsigned int (U32) words - it occupies 4 U32 words  */
typedef unsigned int s50x2b __attribute__((subword(2, 50)));

/* define another swar type with one-bit items */
typedef unsigned int s50x1b __attribute__((subword(1, 50)));
```

Declaration of SWAR variables

```
/* vectors, each in one word */
s1x2b sitem;
s16x2b svecA, svecB;
s16x1b svecC;

/* long vectors */
s50x2b lvecL, lvecM, lvecN;
/* vector of four 32-bit words for reduction */
unsigned int rvec;
```

Sample code with SWAR operations

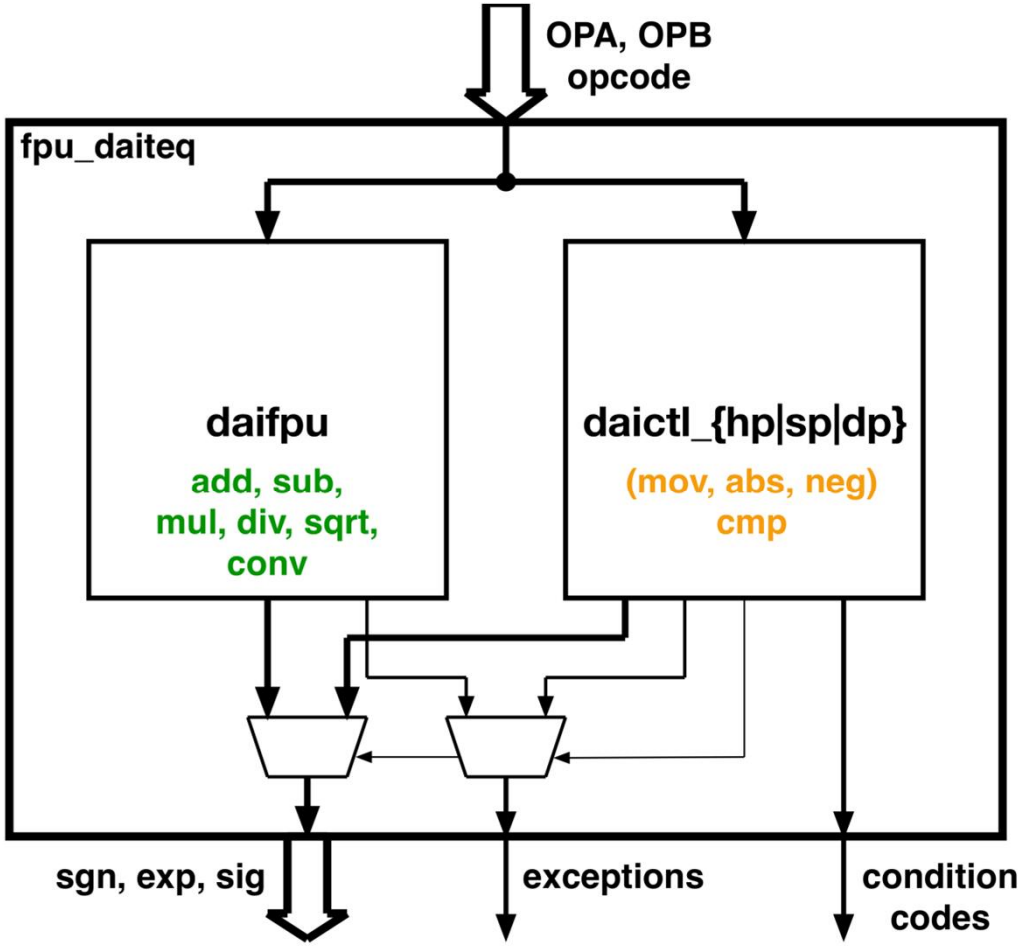
```
int main(void){
        /* set item in a vector */
        svecA[2] = 3;
        // svecA[3] = 4; /* compiler returns compilation error - too big value */
        /* direct operations - SWADD, SWSUB, SWMUL - variables have to contain the same number of items */
        //svecA = svecB + sitem; /* compiler returns compilation error - operands do not have the save number of elements */
        svecA = svecB + svecC;
        svecA = svecB - svecC;
        svecA = svecB * svecC;  /* element-wise multiplication */
```

... (listing continues on the following slide)

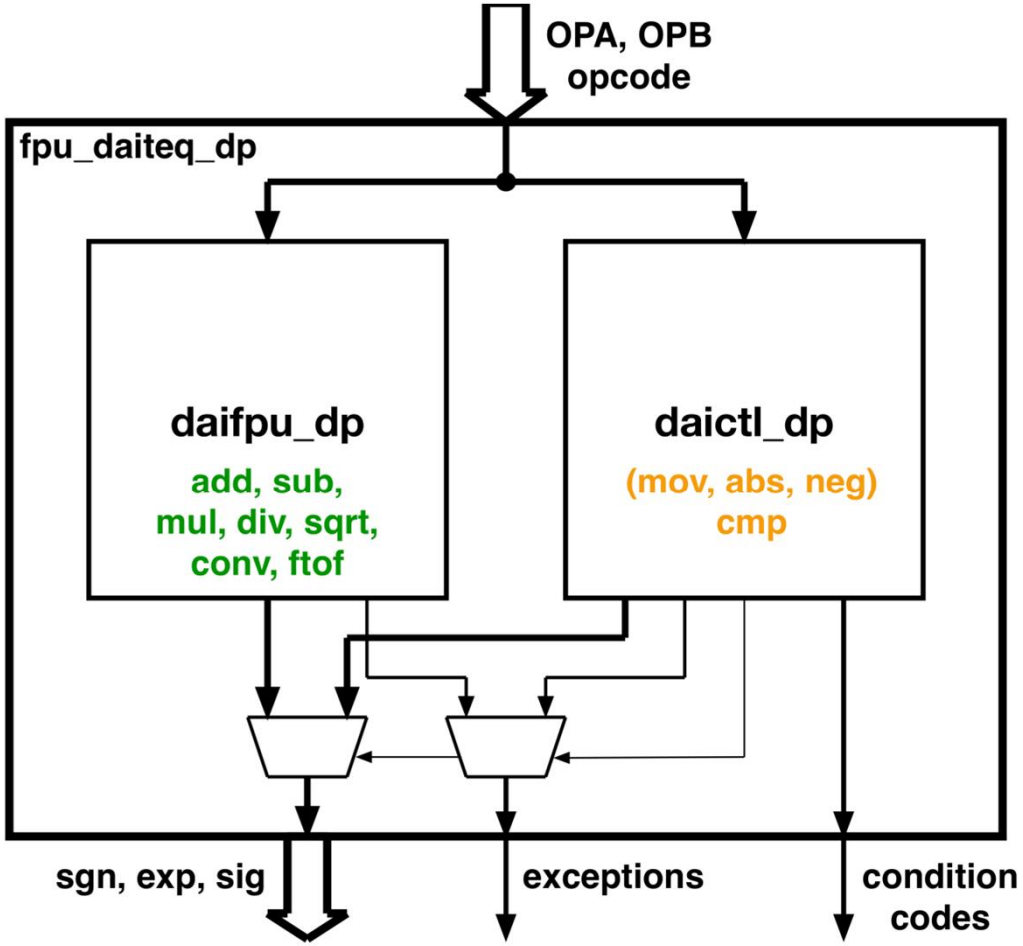
Sample code with SWAR operations (cont'd)

```
        /* for loop is used for long vectors and vectors with different lengths */
        unsigned int accum = 0;
        for (int i=0;i<50;i++)
                accum += lvecL[i];
        /* element-wise product - operands have the same width */
        for (int i=0;i<16;i++)
                svecA[i] = svecA[i]*svecB[i];
        /* reduction */
        rvec = 0;
        for (int j=0;j<50;j++)
                rvec += lvecM[j]*lvecN[j];
        /* get item from a vector */
        int o = svecA[5];
        return 0;
}
```

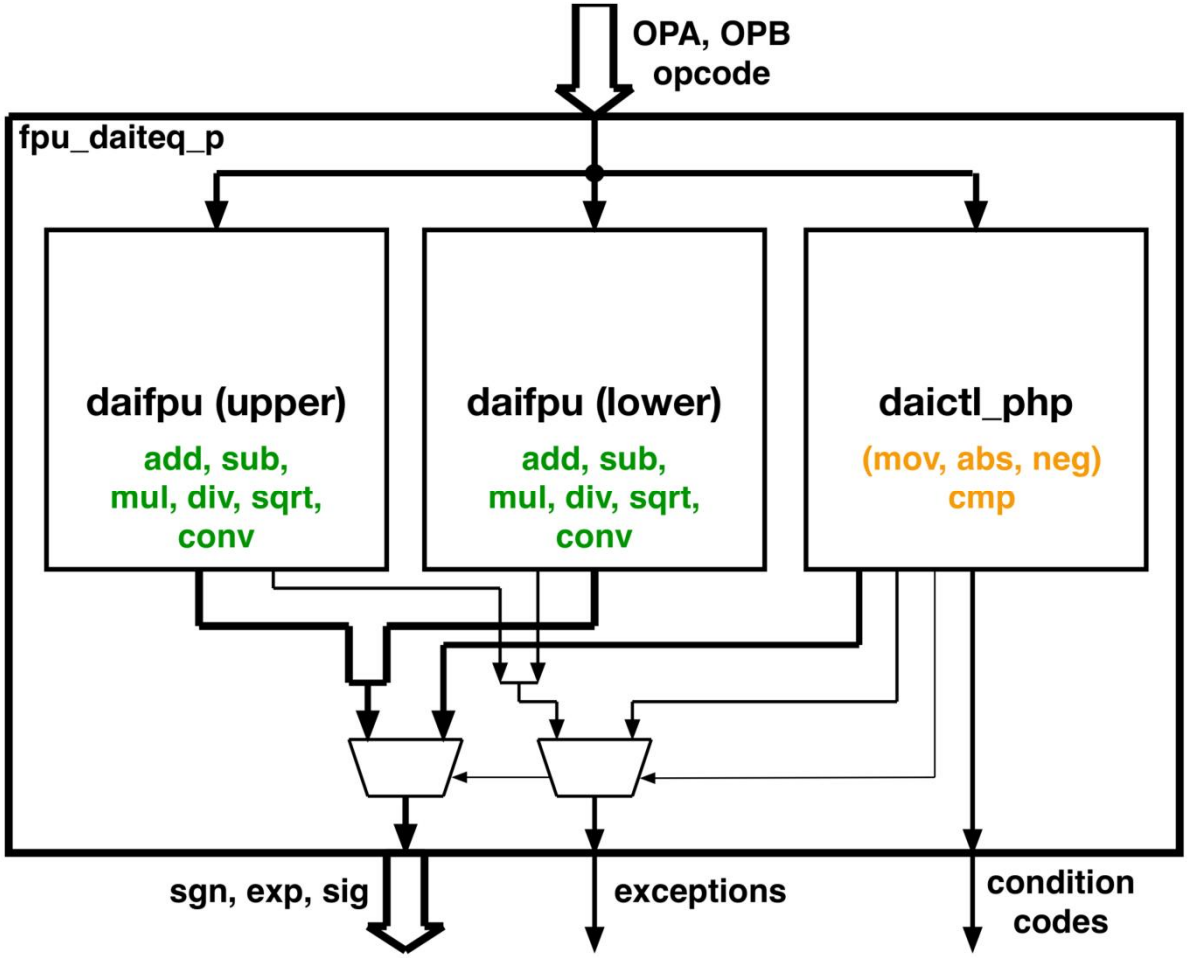
RTL view: fpu (one precision)



RTL view: fpu_dp (Meiko)



RTL view: fpu_p (packed)



Sample code with the new half type

```
#include <stdint.h>
volatile half a,b,r;
volatile int i,j;
void test_half(void)
{
        r = a + b;
        r = a - b;
        r = a * b;
        r = a / b;
        a = (half)i;
        j = (int)r;
}
```

```
int main(void)
{
        volatile half cf = 1.234567H;
        volatile half df = 3.1415926H;
        volatile half ef = cf + df;
        test_half();
        return 0;
}
```

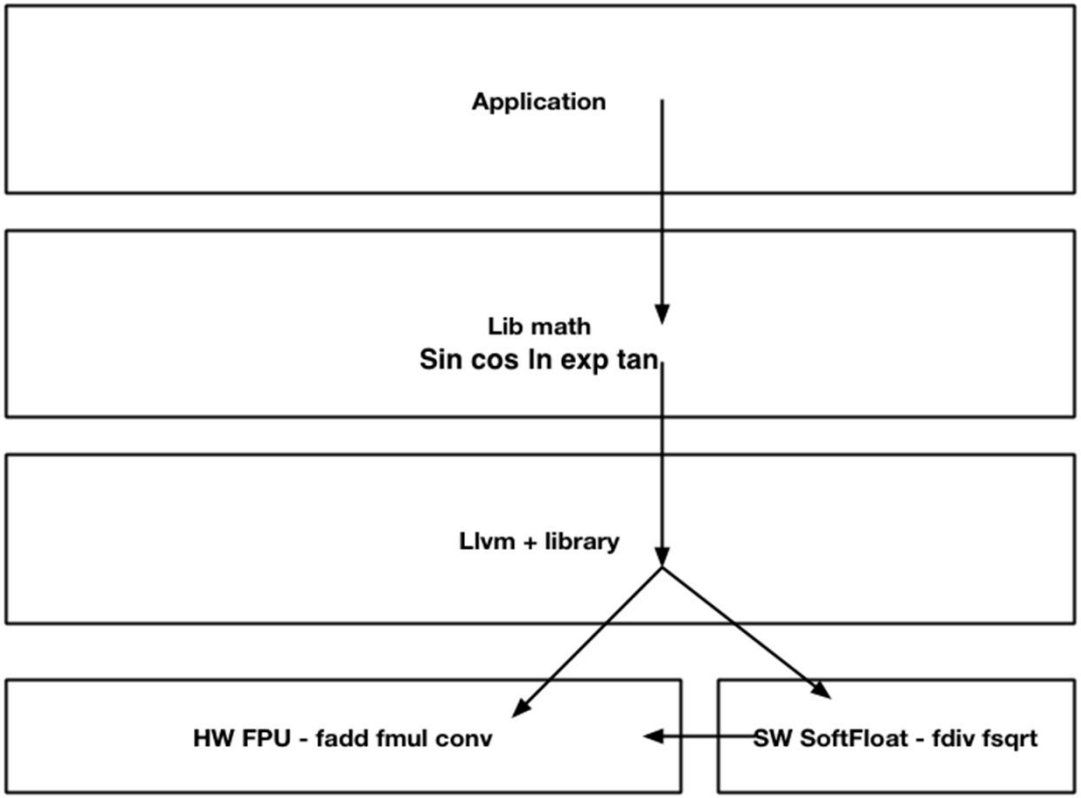
Sample code with the new packed half type

```
typedef half half2 __attribute__((ext_vector_type(2)));

half2 p = {3.1415h, 2.718h};
volatile half2 q,r;

void test(void)
{
        r = p + q;
        r = p - q;
        r = p * q;
        r = p ^ q;
        r = p / q;
}
```

```
int main(void)
{
        p.x = 1.234h;
        test();
        return p.y;
}
```

Compilation and linking: fpu (one precision)

E.g. HW support for one selected precision only, and/or selected operations.



Instruction set extensions

- In this talk the focus is on space, but the applicability is wider
  - HPC – weather modelling (reduced floating-point precision)
  - Consumer – mobile computing

- Areas to increase processor efficiency
  - Decrease processor idle time
  - Increase generation of useful information per instruction/operation
  - Increase silicon functional density – implementation-time configuration of hardware parameters (FPU ops & precision, closely coupled HW accelerators)

Whetstone: fpu_dp vs. Meiko

SP605 ddr

- 50MHz, SP: 13.99 MWIPS, DP: 12.22 MWIPS

VC707: ddr

- 50MHz, SP: 14.74 MWIPS, DP: 12.43 MWIPS
- 100MHz, SP: 27.40 MWIPS, DP: 23.30 MWIPS

AT697F sdram

- 40MHz WS0: SP: 11.73 MWIPS, DP: 9.95 MWIPS
- 60MHz WS0: SP: 17.65 MWIPS, DP: 14.95 MWIPS

AT697F sram

- 40MHz WS1: SP: 11.65 MWIPS, DP: 9.65 MWIPS
- 60MHz WS1: SP: 17.54 MWIPS, DP: 14.51 MWIPS
- 100MHz WS2: SP: 27.43 MWIPS, DP: 22.32 MWIPS
- 100MHz WS3: SP: 25.89 MWIPS, DP: 21.62 MWIPS

Thank you

Martin Daněk
martin@daiteq.com