Using eFPGA for LEON2-FT Instruction Set Extensions

SEFUW2023

March 16, 2023

Martin Daněk, Roman Bartosiński, daiteq

Overview

- Concept of custom instructions with the SWAR unit
- SWAR in eFPGA
- SWAR for CCSDS121
  - Speedup due to SWAR
  - Implementation in Menta eFPGA
- Conclusions

This work has been performed under an ESA contract 4000122242/17/NL/LF.

SWAR history

SWAR = SIMD-within-a-register

- Term introduced by R.J.Fisher in 2003
- Initial study of SWAR for GNSS in LEON2-FT performed by H.A.Bridonneau at ESTEC in 2014
- SWAR implemented in LEON2-FT (2019) and NOEL-V (2022) by daiteq, including support in LLVM

Motivation for SWAR

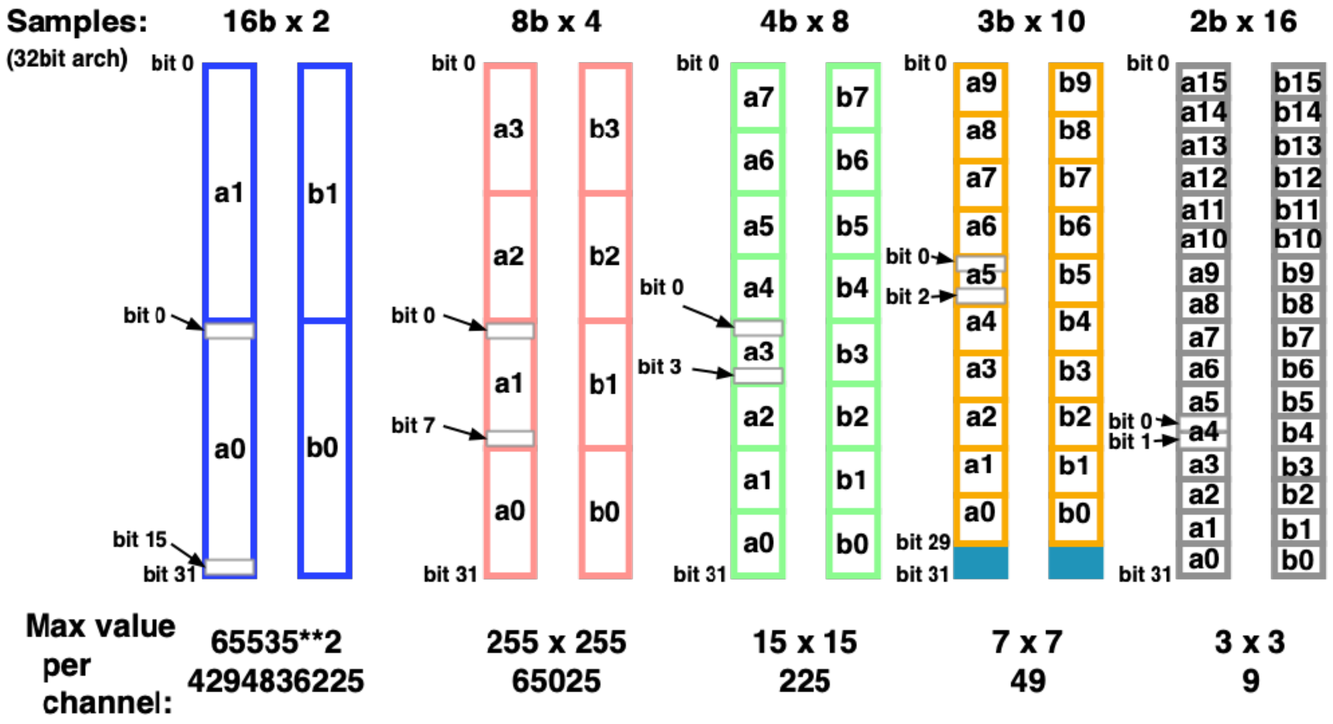
- Consider low-precision integer numbers, e.g. 2-bit GNSS samples
- Find ways to increase processing with minimal changes to the processor microarchitecture
- E.g. for LEON2-FT:

  \+

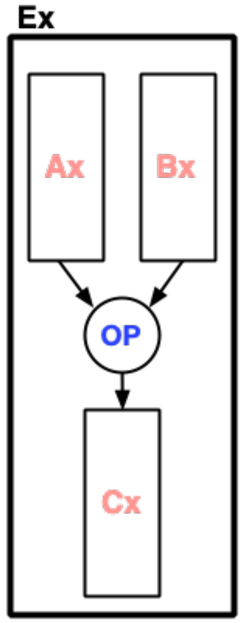
  \-

  *

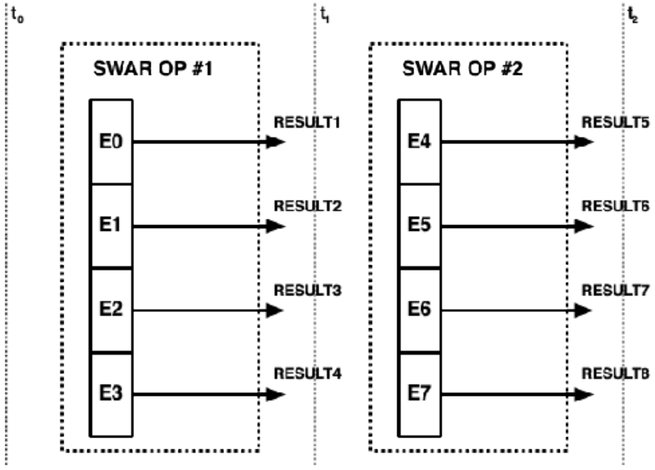
  >>



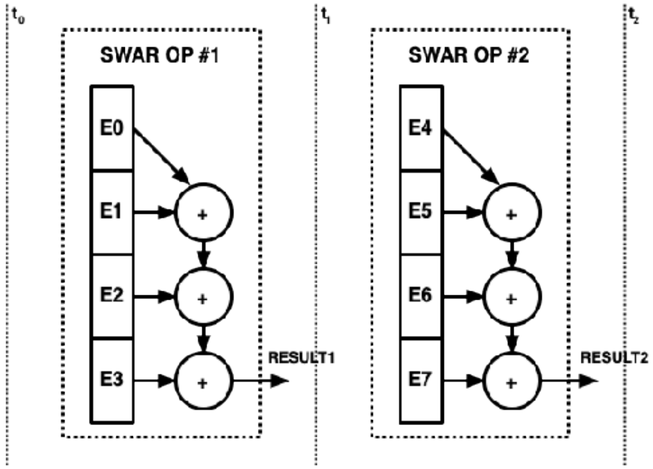
SWAR “skeletons"



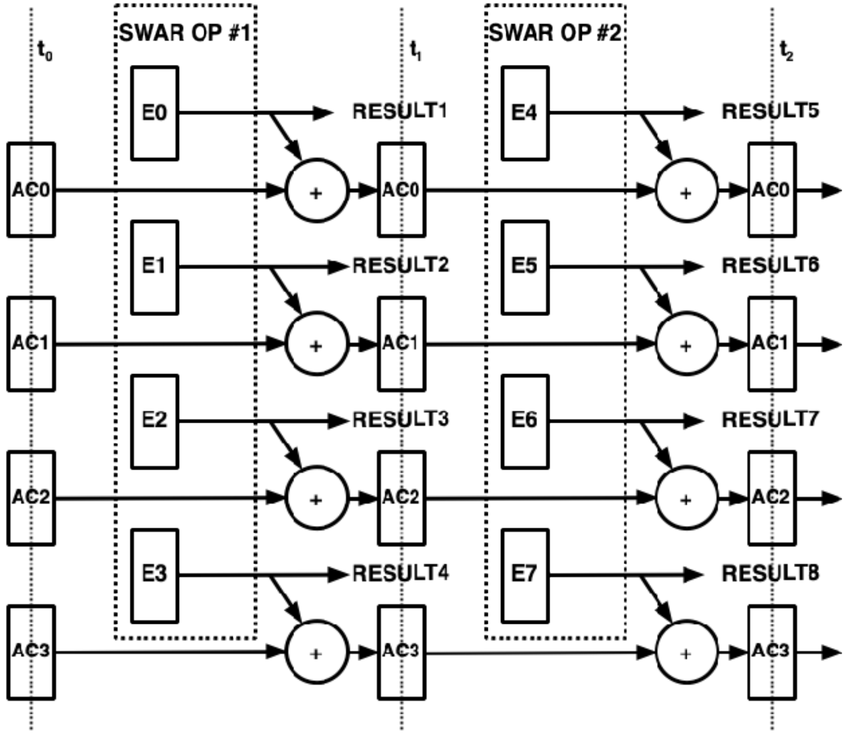
Ex denotes a value pair together with an operation
ACx denotes an accumulator



MAP



REDUCE



MAP & ACCUMULATE

SWAR C-level API

- Type definition

```
typedef unsigned int u16x2b __attribute__((subword(2, 16)));

u16x2b lvecL[50], lvecM[50], lvecN[50];
```

- Either inferring SWAR operations

```
for (int i=0;i<50;i++) {
 lvecN[i] = lvecL[i] * lvecM[i];
}

unsigned int accum = 0;
for (int i=0;i<50;i++) {
  accum += lvecL[i];
}
```

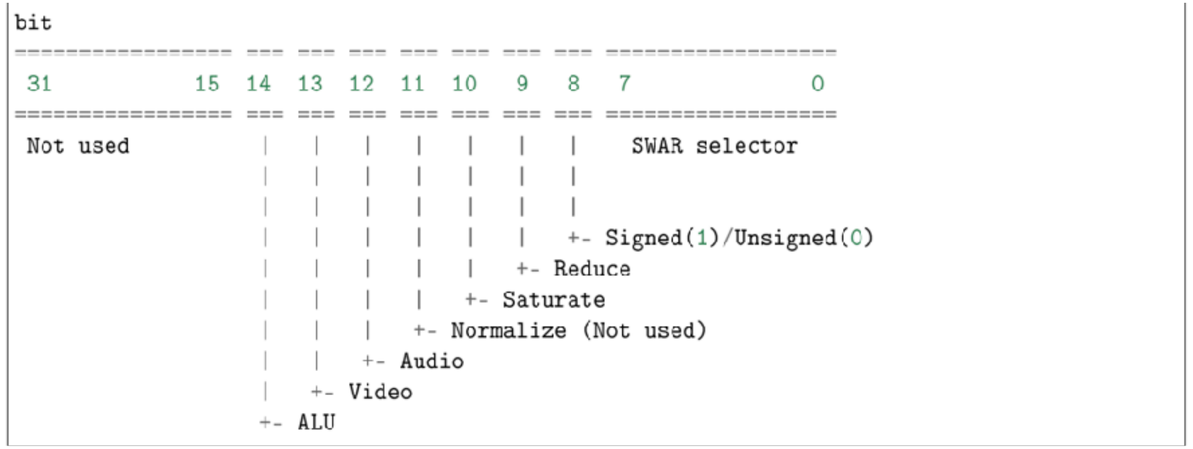
- Or explicitly configuring SWAR operations

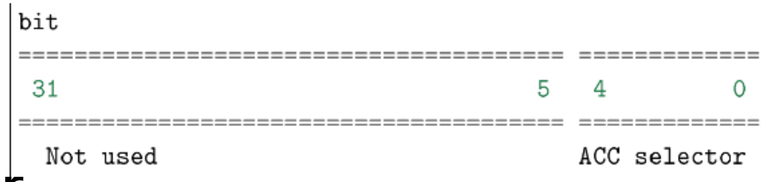
```
set_swar_op(0x4);
for (int i=0;i<50;i++) {
  op_swar(lvecL[i], lvecM[i], lvecN[i]);
}

set_swar_op(0x10);
for (int i=0;i<50;i++) {
  op_swar(lvecL[i], 0x0, tmp);
  accum += tmp;
}
```

SWAR machine-level API

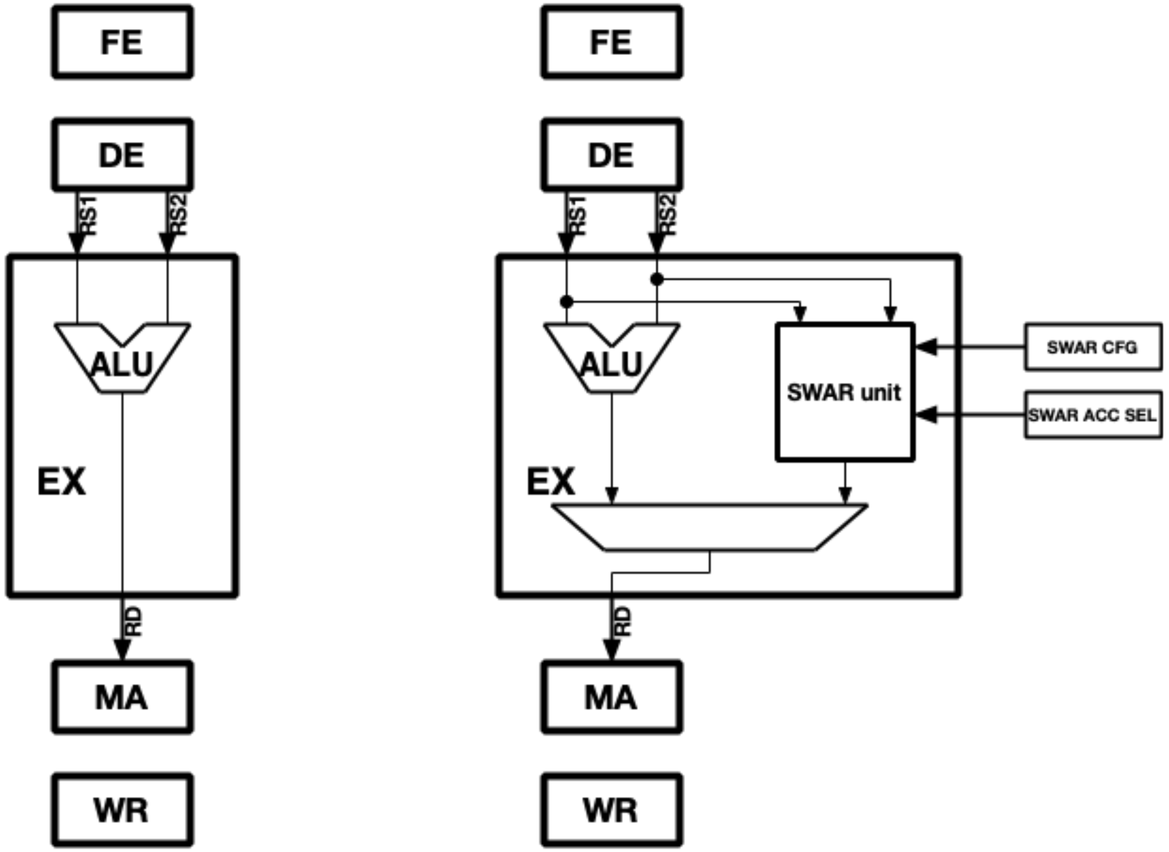
A minimalist approach using generic instructions:

- SWAR
  - exec SWAR operation
- SWARcc
  - exec SWAR operation –> flags
- WRASR
  - set SWAR operation
  - set SWAR accumulator for readback
- RDASR
  - read the selected SWAR accumulator





LEON2-FT + SWAR



ARITH	op3(5-4)			
op3(3-0)	00	01	10	11
0000	ADD	ADDcc	TADDcc	*WRASR*/WRY
0001	AND	ANDcc	TSUBcc	WRPSR
0010	OR	ORcc	TADDccTV	WRWIM
0011	XOR	XORcc	TSUBccTV	WRTBR
0100	SUB	SUBcc	MULScc	FPop1
0101	ANDN	ANDNcc	SLL	FPop2
0110	ORN	ORNcc	SRL	CPop1
0111	XNOR	XNORcc	SRA	CPop2
1000	ADDX	ADDXcc	*RDASR*/RDY/STBAR	JMPL
1001	**SWAR**	**SWARCC**	RDPSR	RETT
1010	UMUL	UMULcc	RDWIM	TICC
1011	SMUL	SMULcc	RDTBR	FLUSH
1100	SUBX	SUBXcc		SAVE
1101				RESTORE
1110	UDIV	UDIVcc		
1111	SDIV	SDIVcc		

True user-defined instructions with SWAR

The idea: enable users to define their own application-specific machine-level instructions in LEON2-FT ASIC (or NOEL-V ASIC).

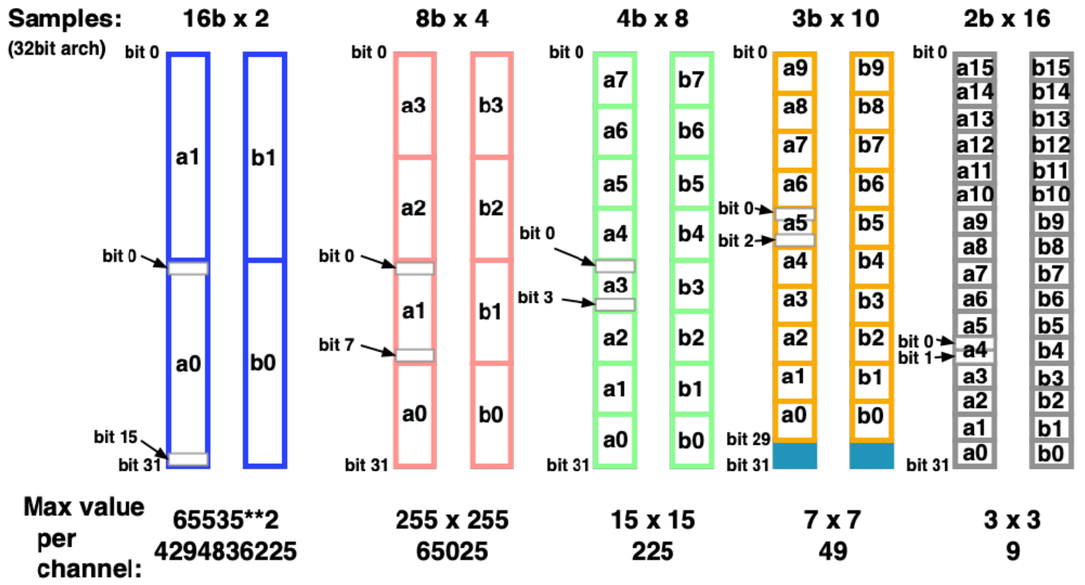
How: implement the SWAR unit as an eFPGA.

Current evaluation: main focus on satellite applications

- GNSS - tracking loop
- GPP - FIR filter
- Image compression - CCSDS121

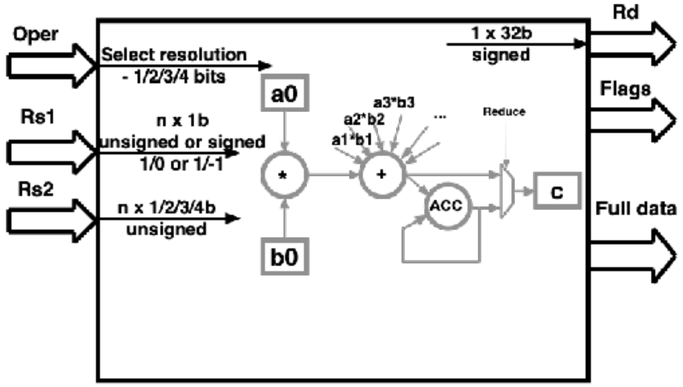
SWAR modules

- Correlation - 2b, 3b, 4b
- Demodulation (multiplication) - 2b, 3b, 4b
- Sine/cosine lookup - 32b argument –> 2b, 3b, 4b value
- ALU - add, sub, mul, shr - 8b, 16b
- Entropy coding - 8b, 16b, 32b

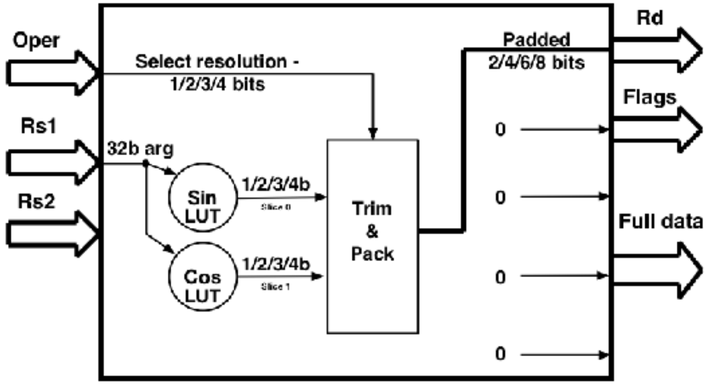


The concept: eFPGA = SWAR unit

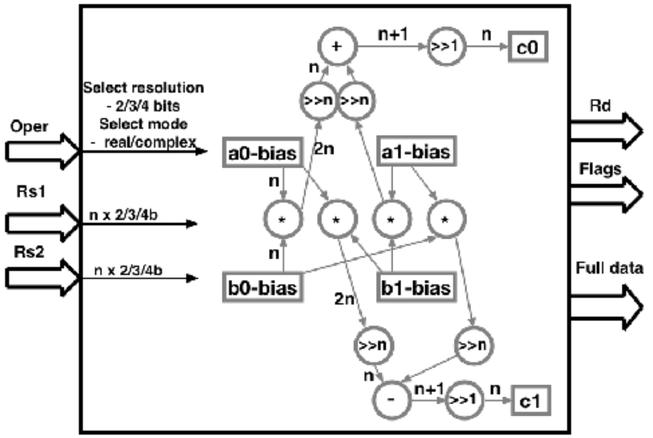
GNSS



Correlation (dot product)



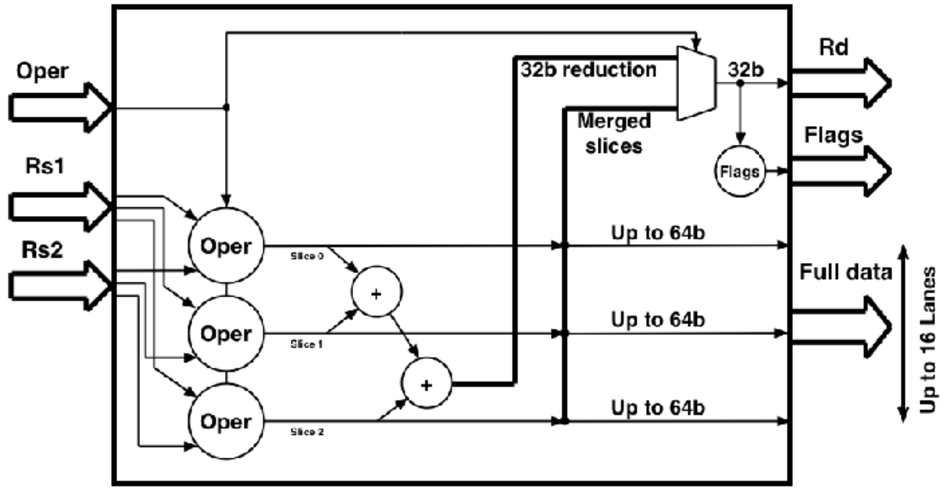
Sine/cosine lookup



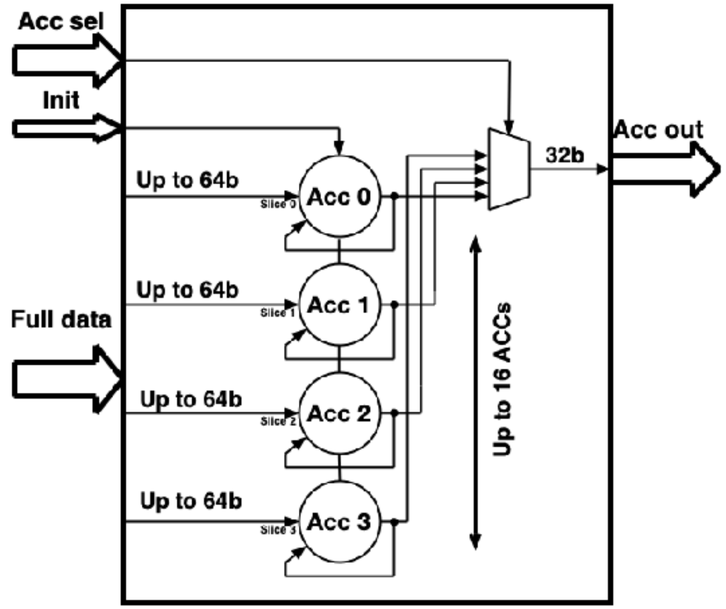
Demodulation
(real/complex multiplication)

The concept: eFPGA = SWAR unit

GPP

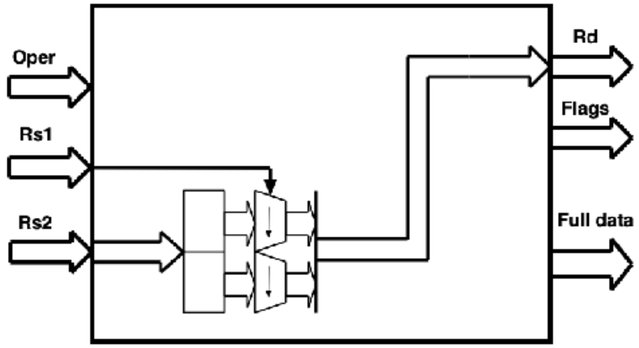


SWAR ALU (add, sub, mul, shr)

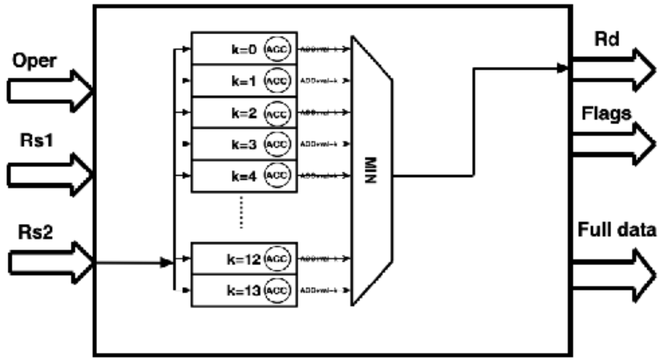


SWAR accumulators

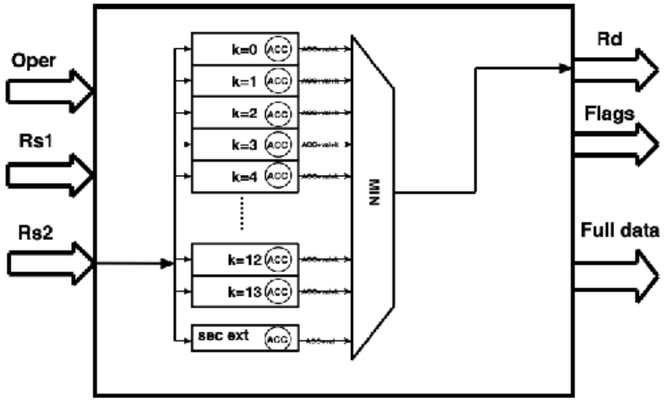
The concept: eFPGA = SWAR unit



shr - shift right

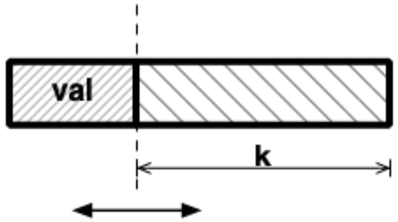


k-split



Extended k-split

CCSDS121



1. shr
2. val
3. add k

Optimal Rice code

SWAR for CCSDS121

SHyLoC case 1
  SWAR: CCSDS121 input: 32b words
        block size: 64 words
        k-split (shift right): 1,2,...,29

SHyLoC case 2
  SWAR: CCSDS121 input: 16b words
        block size: 64 words
        k-split (shift right): 1,2,...,13

SHyLoC case 3
  SWAR: CCSDS121 input: 8b words
        block size: 64 words
        k-split (shift right): 1,2,…,5

CCSDS121 speedup w/ SWAR (shr)

TestID	Nx	Ny	Nz	D	J	ICount_orig	ICount_swar	Improvement [%]
img1_14_16_0	512	512	32	14	16	10208501256	8740487560	14.38
img1_14_32_0	512	512	32	14	32	9696999386	7964748186	17.86
img1_14_64_0	512	512	32	14	64	9439838250	7575468298	19.75
img1_14_8_1	512	512	32	14	8	12081073145	11141437689	7.78
img1_14_16_1	512	512	32	14	16	10974968387	9506848963	13.38
img1_14_32_1	512	512	32	14	32	10418597848	8686236440	16.63
img1_14_64_1	512	512	32	14	64	10141518742	8277036342	18.38
img2_14_8_0	512	512	32	14	8	11127286428	10187747740	8.44
img2_14_16_0	512	512	32	14	16	10107851497	8639837801	14.52
img2_14_32_0	512	512	32	14	32	9593914977	7861663777	18.06
img2_14_64_0	512	512	32	14	64	9333788568	7469418616	19.97
img2_14_8_1	512	512	32	14	8	11973732857	11034055740	7.85
img2_14_16_1	512	512	32	14	16	10886353712	9418234288	13.49
img2_14_32_1	512	512	32	14	32	10340108590	8607747182	16.75
img2_14_64_1	512	512	32	14	64	10067925373	8203442973	18.52

Execution in QEMU:

- ICount_orig - instruction count for SHyLoC w/o SWAR
- ICount_swar - instruction count for SHyLoC w/ SWAR

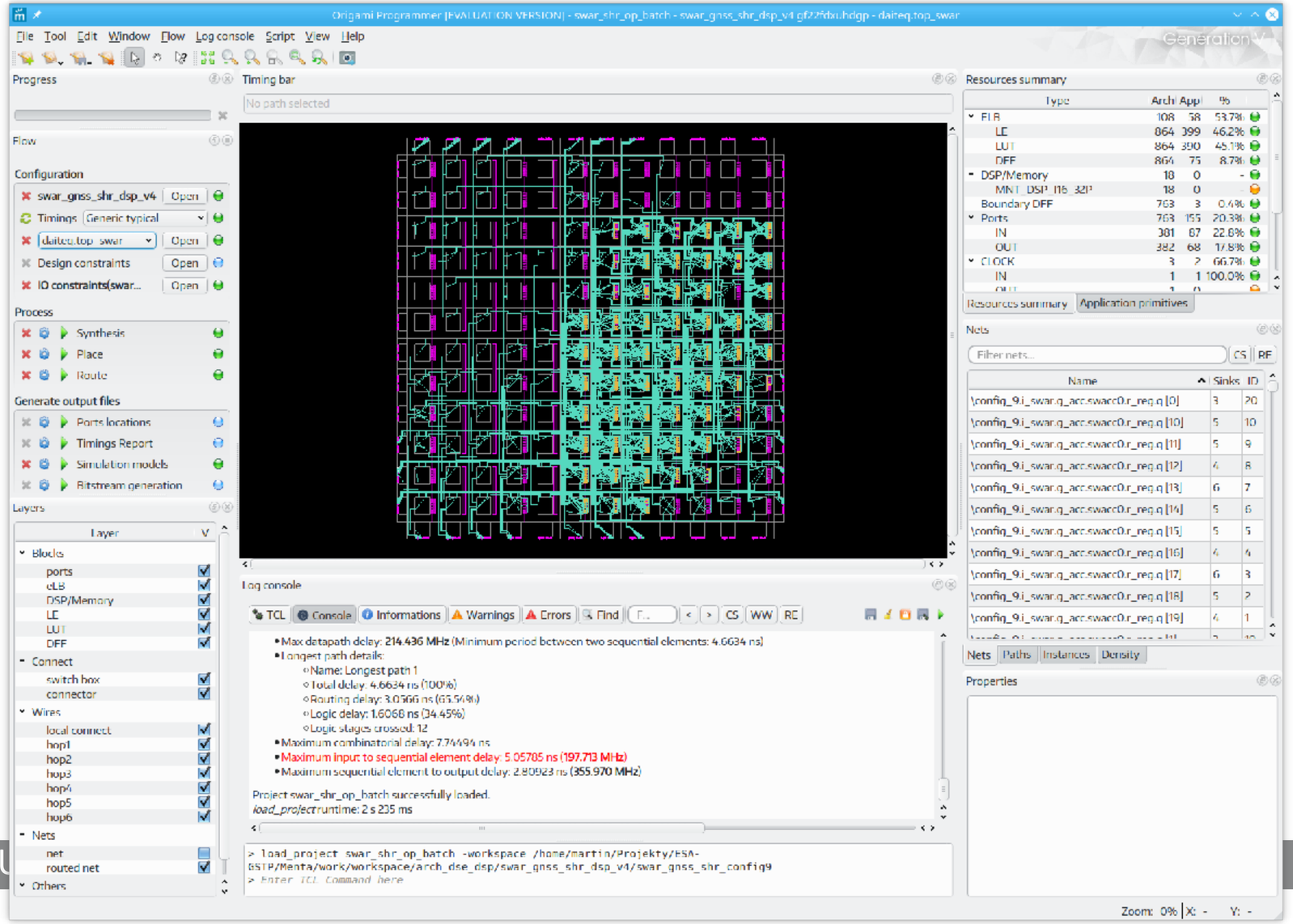
eFPGA for SWAR - sampled SWAR configurations

- Generate a common architecture for all configurations
- Achieve frequency at least 100MHz in GF22FDX UHDGP

config	swar_unit = description	OPS	SinCos	Accumulators
0	hand-optimized for GNSS, all bitwidths	CDL	Y	No
1	hand-optimized for GNSS, 2b only	CDL	Y	No
2	hand-optimized for GNSS, 3b only	CDL	Y	No
3	hand-optimized for GNSS, 4b only	CDL	Y	No
4	swar_alu w/ swar_acc video - 2x16b	AMS	N	No
5	swar_alu w/ swar_acc audio - 4x8b	AMS	N	No
6	swar_alu w/ swar_acc gen 3x10b	AMS	N	No
7	swar_alu w/ swar_acc video - 2x16b	AMS	N	2x38b
8	swar_alu w/ swar_acc audio - 4x8b	AMS	N	4x22b
9	swar_alu w/ swar_acc gen 3x10b	AMS	N	3x20b
10	ALU SHR 1x 32b	S	N	1x38b
11	ALU SHR 2x 16b	S	N	1x22b
12	ALU SHR 4x 8b	S	N	1x15b

OPS:
C - correlation
D - demodulation
L - sine/cosine lookup
A - addition
M - multiplication
S - shift right

Menta Origami Programmer



SWAR configurations are described in VHDL/Verilog, and implemented in Origami Programmer.

The tool generates simulation netlists (Verilog), configuration bitstream, and implementation reports.

eFPGA for SWAR - operating frequencies for implemented configurations

Parameter	Architecture				
.	base	dsp_v1	dsp_v2	dsp_v3	dsp_v4
CLK/S/R	1	1	1	1	1
#LUT6	1152	1296	1008	1440	864
DSP type	I24_48_F32P	I16_32P	I16_32P	I8_16P	I16_32P
DSP	24	27	21	18	18
INs	477	477	413	477	381
OUTs	478	478	414	478	382
GCLK	1	1	1	1	1
GS/GR	1	1	1	1	1
Area [%]	100	66	52	63	44
Pstat [%]	100	112	87	99	99
SWAR implementations					
config1 [MHz]	126.378	117.691	137.039	111.696	128.925
config2 [MHz]	117.681	119.252	129.957	108.532	128.155
config3 [MHz]	118.010	126.364	126.921	136.267	117.446
config4 [MHz]	91.387	90.573	93.188	82.643	103.406
config5 [MHz]	85.159	91.886	86.579	86.100	93.158
config6 [MHz]	79.218	85.303	81.046	80.302	84.098
config7 [MHz]	85.333	97.586	94.341	85.010	94.730
config8 [MHz]	88.389	96.574	101.720	90.525	94.287
config9 [MHz]	85.022	93.203	97.269	90.707	93.675
config10 [MHz]	199.848	207.524	204.413	175.645	203.886
config11 [MHz]	146.017	161.346	156.018	131.403	155.925
config12 [MHz]	157.286	152.539	167.408	157.389	156.068

Target tech - gf22fdxuhdgp

All configurations used identical pinout.

Architecture:
Base - auto-generated
dsp_vx - hand-tuned

Conclusions

- Using custom instructions in LEON2-FT can significantly improve performance, e.g., ~1.25x fewer executed instructions for CCSDS121 (compiled, swar_shr version), ~2x faster FIR filter (compiled), >2x faster GNSS tracking loop (hand-crafted).
- For the considered SWAR configurations realistically achievable eFPGA frequencies are slightly below 100MHz (gf22fdxuhdgp).
- Certain SWAR configurations may need to execute in 2 or more pipeline cycles not to impose low frequency on the whole LEON2-FT => introduce support for SWAR idle cycle insertion.
  - The impact on the performance is expected to be low since the intensity of SWAR instructions is unlikely to be close to 100% in real applications. Key factor - (clock cycles per SWAR instruction) / (clock cycles per replaced kernel)
- Reconfiguration time for the dsp_v4 fabric is ~106us (10580 cycles @100MHz over SPI-1).
  - Consider multi-context configuration, e.g. using two multiplexed eFPGAs.

THANK YOU