CloudBEAR

Применение P Расширения Системы Команд RISC-V для Алгоритмов Цифровой Обработки Сигналов

Захаров Дмитрий
15.04.2024

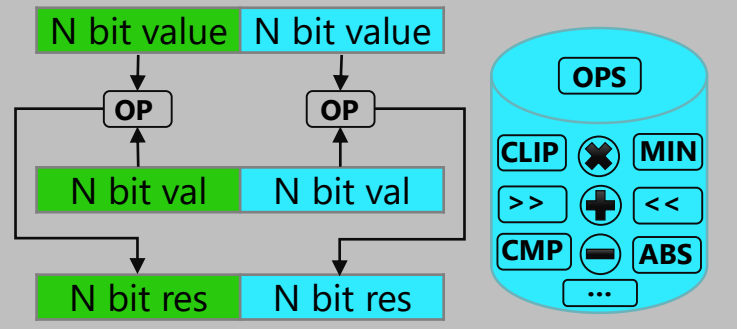
Ознакомление

Что из себя представляет P расширение?

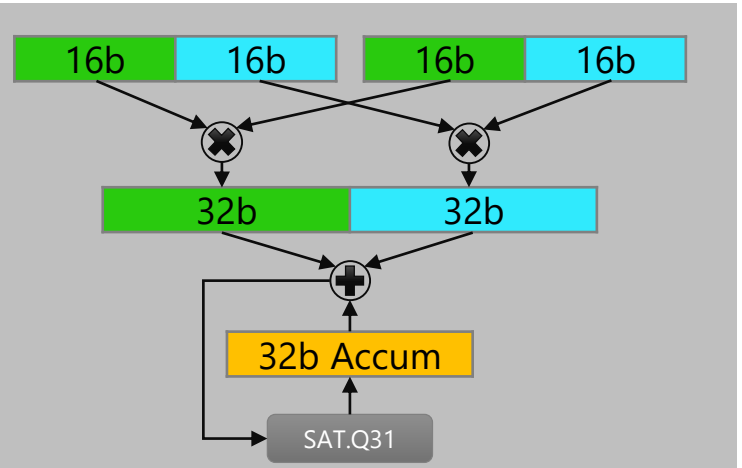
RISC-V [P]acked SIMD Расширение

CloudBEAR

- Набор инструкций для **ускорения fixed point и целочисленных вычислений** используют только регистры общего назначения (GPR).
- **~330 инструкций** разбитые на SIMD, Partial-SIMD, Non-SIMD группы операций.
- Поддерживается
  - RV32 и RV64 архитектура
  - [8, 16, 32, 64] битовые операнды
  - Данные в **Q нотации** (Q7, Q15, Q31)
  - **Комбинированные операции,** объединяющие умножение, накопление, сдвиг, округление, насыщение.
- Основное применение P расширения это **ускорение DSP алгоритмов** на конечных устройствах с ограниченными ресурсами.



До 4х значений параллельно (RV32)



6 операций за 1 инструкцию (RV32)

Модель Программирования

CloudBEAR

- Инструкции оперируют XLEN целочисленными регистрами общего назначения (GPRs).

- Регистры рассматриваются как небольшие вектора разбитые на N значений заданной битности.

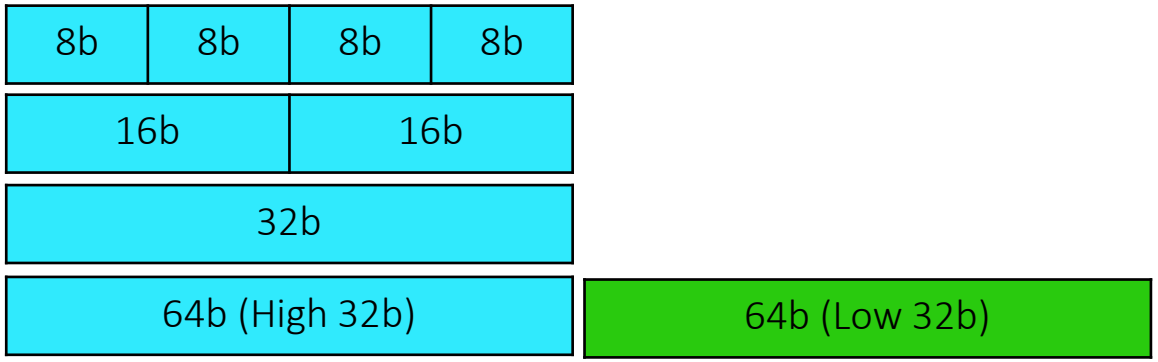
Конкретная интерпретация входных и выходных регистров определяется только самой инструкцией.

- Операция выполняется над всеми значениями векторов. Формат выходного вектора может сохраниться (N-to-N) или измениться (N-to-M или N-to-1).

- RV32 поддерживает 64-битные данные. Они образуются чет/нечет парой 32-битных регистров.

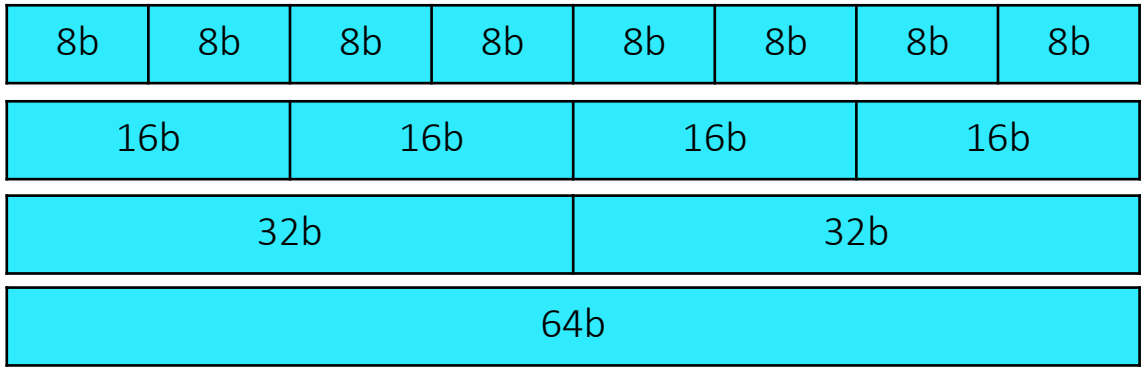
Только чётный GPR передаётся в такие инструкции как операнд.

- Насыщение/переполнение отражается в CSR регистре VXSAT. Других конфигурационных регистров расширение не добавляет.



RV32: GPR xN

RV32: x(N+1) GPR



RV64: xN GPR

Ключевые События

[2016.12]	Решение о разработке Packed SIMD расширение ISA[1]
[2017]	Andes предложили собственное Packed SIMD расширение как базу для RISC-V P
[2018]	Официально создана Task Group по P
[2021.12]	Последние опубликованные изменение спецификации P (0.9.11)[2]
[2022.09]	Начался активный процесс пересмотра спецификации (John Hauser)[3]
[2024.04]	Спецификация ещё в разработке

- От исходных ~150 инструкций спецификация разрослась до ~330.
- После продолжительной стагнации работа над спецификацией снова идёт. К работе привлечён John Hauser (автор SoftFloat/TestFloat).
- Первоочередная цель на данный момент - выделить небольшое base P подмножество и ратифицировать. Остальные инструкции - к (возможному) рассмотрению на будущие Zp*.

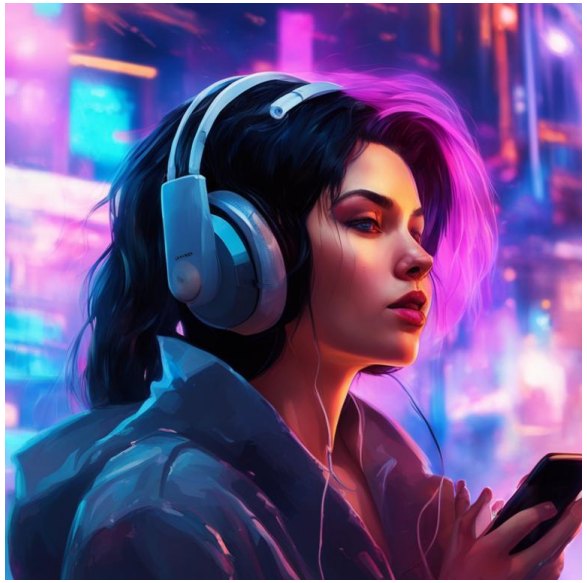
[1] 5th RISC-V Workshop Dec 2016.
[2] https://github.com/riscv/riscv-p-spec
[3] http://www.jhauser.us/RISCV/ext-P/RISCV-20220911-P-extension.pdf

Применения

Где это используется?

Возможные Приложения

- Audio/Speech Processing
- Motor Control
- Tiny ML
- Codec
- Sensors Signal Processing



Процессоры с P Расширением

CloudBEAR

Telink TLSR9 Series

CPU: Andes D25F (RV32IMAFDCPB) @ 96 MHz

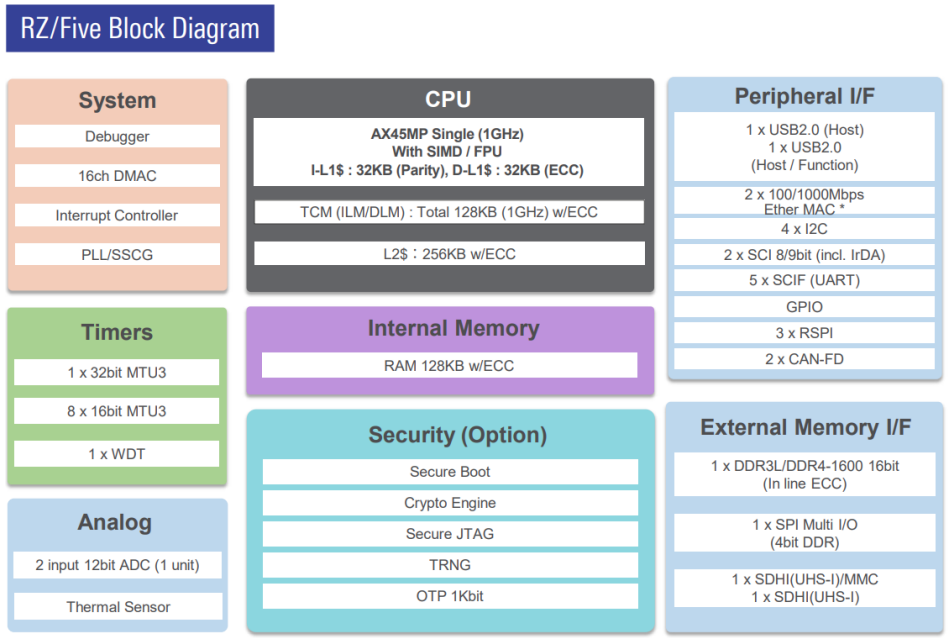
Audio & IoT Chip

Gigadevice GD32VW553

CPU: Nuclei N307 (RV32IMAFDCPB) @ 160 MHz

IoT chip with wide range of wireless interfaces

Renesas RZ/Five

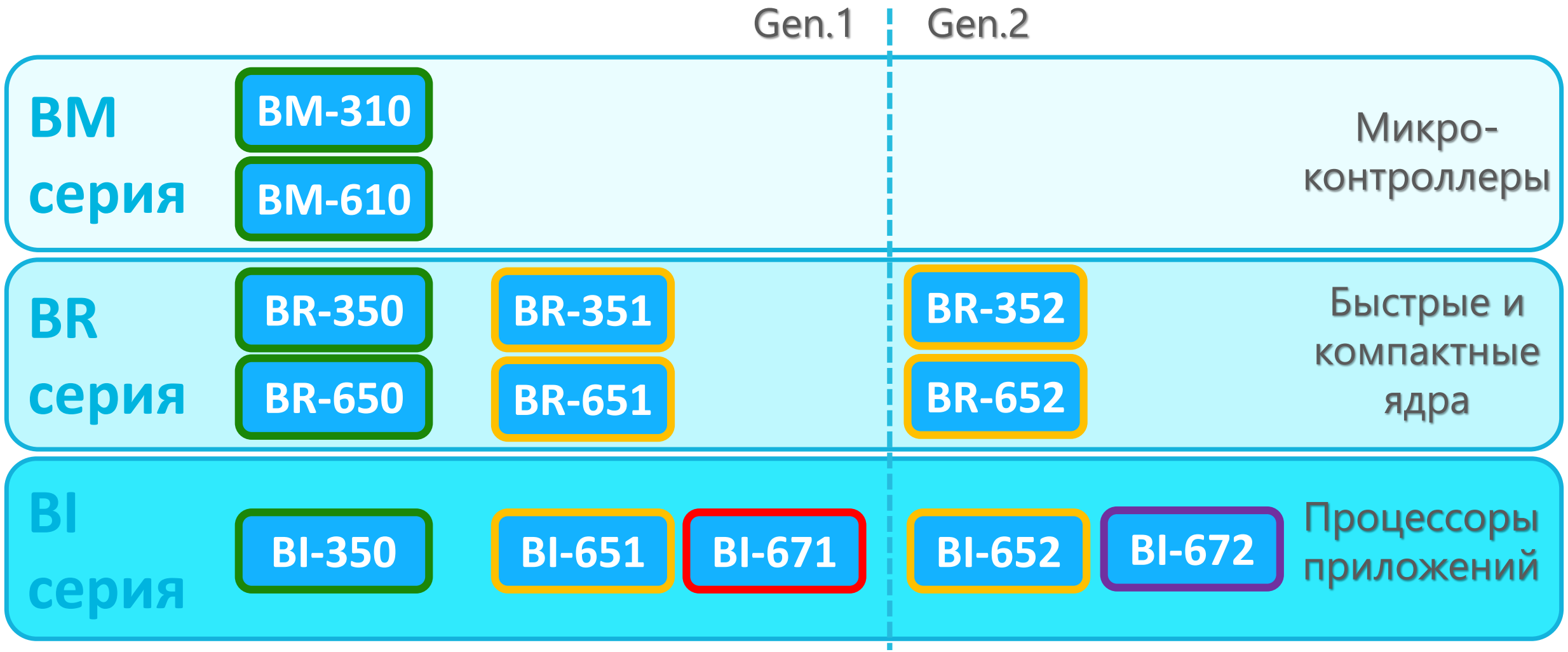


CPU: Andes AX45MP (RV64GCPN) @ 1 GHz

Civil Infrastructure Platform (CIP) Linux core for industrial applications

Процессорные Ядра RISC-V

CloudBEAR



Маркировка:

- Bx-3xx: RV32
- Bx-6xx: RV64
- 1-wide P extension
- 2-wide P extension
- 2-wide Out-of-Order
- 3-wide Out-of-Order

CMSIS-DSP

Библиотека[1] распространённых вычислительных функций для Cortex-M и Cortex-A ядер.

13 групп функций, в том числе:

- Поэлементные базовые операции: add, sub, mul, bitwise, etc.
- Спектральные преобразования: FFT, DCT, MFCC
- Фильтрация: FIR, Biquad IIR, Convolution, Correlation, etc.
- Контроллер: PID, Park/Clark Transforms, Sin/Cos
- Классический ML: SVM, Bayes classifier
- .. 8 других групп ..

Поддерживаемые типы данных:

- [q7, q15, q31] для целочисленной арифметики
- [f16, f32, f64] для арифметики с плавающей точкой

CMSIS-DSP совместимые библиотеки предоставляются многими вендорами Embedded процессорных ядер. Для RISC-V наиболее известны **AndeSoft DSP Library (Andes)** и **NMSIS (Nuclei)**, где используются **P и V расширения**.

[1] https://github.com/ARM-software/CMSIS-DSP

BEAR-DSP

BEAR-DSP - коллекция DSP примитивов с C интерфейсом и интенсивным использование операций P расширения.

Модель библиотеки

- **Набор C функций**, каждая из которых реализует один оптимизированный DSP примитив
- **Набор intrinsic функций** для прямого доступа к P инструкциям в пользовательском алгоритме
- **Код** библиотеки **как набор примеров** использования P инструкций и подходов к оптимизации

Статус библиотеки

Первая версия (1.0.0) готовится к релизу. Версия охватывает:

- Подмножество наиболее перспективных и запрашиваемых функций совместимых с **CMSIS-DSP**
- Типы с фиксированной точкой и плавающей точкой одинарной точности **[q7, q15, q31, fp32]**
- **98 функций,** из которых **49 оптимизированы** с P

BEAR-DSP v1.0.0

P extension intrinsics
Controller Q31
Fast Math Q31
Filtering (IIR Biquad) Q31
Filtering (IIR Biquad) FP32
Controller FP32
FastMath FP32
Basic Math Q31|Q15|Q7
FastMath Q15
Transform (FFT) Q15|Q31
Filtering (FIR) Q15|Q31
Matrix FP32

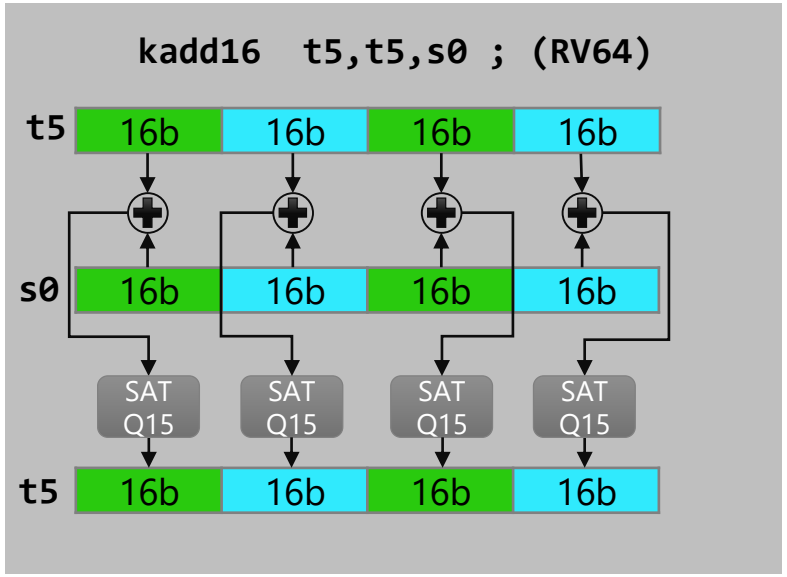
Практика

Как это используется?

SIMD Операция RV64

$$x_i = a_i + b_i$$
$$y_i = MAX(-32768, MIN(x_i, 32767))$$

```
<.L42>:
ld      t5,0(a4)  ; load {a0, a1} (v4int16)
ld      s0,0(a3)  ; load {b0, b1} (v4int16)
kadd16  t5,t5,s0  ; 4x add & sat
sd      t5,0(a7)  ; store {y0, y1} (v4int16)
addi    a4,a4,8
addi    a3,a3,8
addi    a7,a7,8
bltu    a4,a5,b6 <.L42>
```



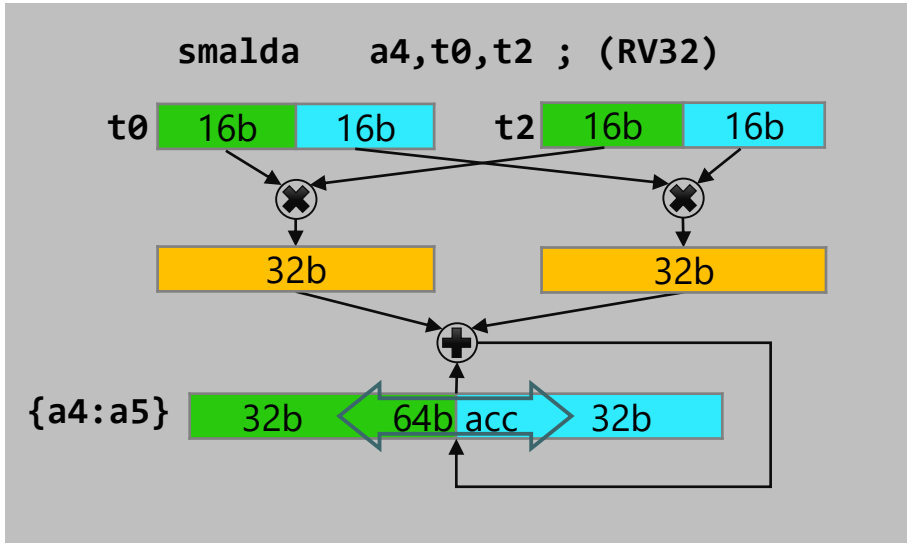
Цветовая маркировка отрывка программы

- Загрузка 64х битных вектора (4 операнда в каждом) используя стандартный ld (RV64I)
- Поэлементное сложение двух векторов с насыщением результата (RV64P)
- Сохранение результатов

Dot Product RV32

$$y = \sum_i a_i * b_i$$

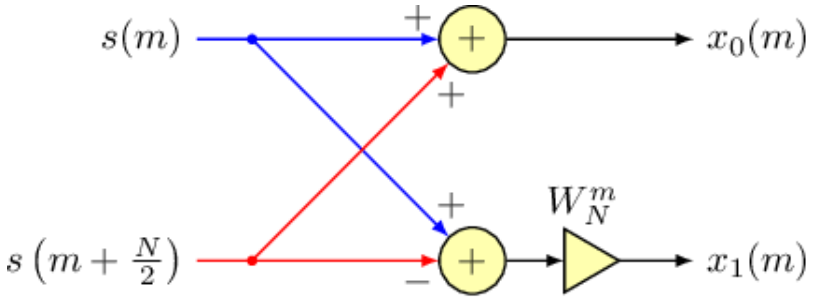
```
<.L32>:
lw      s0,0(a2) ; load {a0, a1} (v2int16)
lw      s1,0(t1) ; load {b0, b1} (v2int16)
lw      t0,4(a2)
lw      t2,4(t1)
smalda  a4,s0,s1 ; 2x MAC
smalda  a4,t0,t2
addi    a2,a2,8
addi    t1,t1,8
bltu    a2,a6,be <.L32>
sw      a4,0(a3) ; store result (int64) Hi word
sw      a5,4(a3) ; store result (int64) Lo word
```



Цветовая маркировка отрывка программы

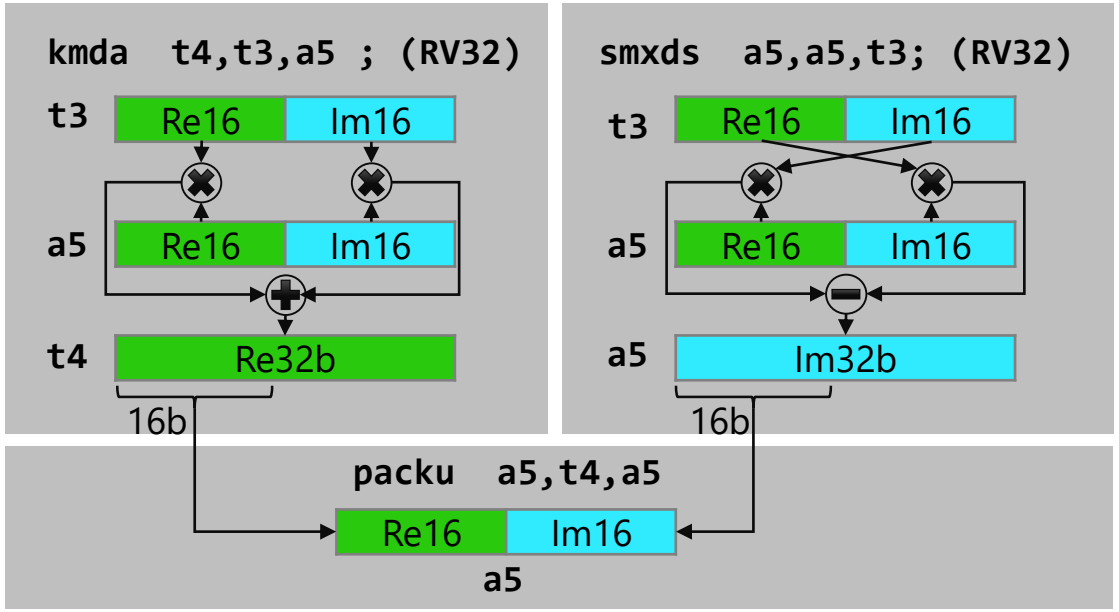
- Загрузка 32х битных векторов (2 операнда в каждом) используя стандартный lw (RV32I).
- Поэлементное умножение с накоплением результатов в 64х битном аккумуляторе, составленным из пары регистров {a4; a5} (RV32P)
- Сохранение 64х битного результата (пары регистров) по адресу назначения

FFT Radix2 Butterfly RV32



$$\begin{pmatrix} a'_{re} = a_{re} + b_{re} \\ a'_{im} = a_{im} + b_{im} \end{pmatrix} \qquad \begin{pmatrix} b'_{re} = (a_{re} - b_{re}) * W_{re} + (a_{im} - b_{im}) * W_{im} \\ b'_{im} = (a_{im} - b_{im}) * W_{re} - (a_{re} - b_{re}) * W_{im} \end{pmatrix}$$

```
lw       a5,0(a1)    ; load A (v2int16)
lw       t4,0(a4)    ; load B (v2int16)
lw       t3,0(t1)    ; load W (v2int16)
radd16   a0,a5,t4    ; A + B >> 1
rsub16   a5,a5,t4    ; A - B >> 1
kmda     t4,t3,a5    ; straight mul & add
smxds    a5,a5,t3    ; crossed mul & sub
srai16.u a0,a0,0x1   ; A >> 1 with roundUp
packu    a5,t4,a5    ; Pack Hi Halves into Word
sw       a0,0(a1)
sw       a5,0(a4)
```



Цветовая маркировка отрывка программы

- Загрузка действительной и мнимой части комплексного числа как единого 32х битного вектора
- Выполнение операций над комплексными числами (RV32P)
- Упаковка старшей части результатов умножения в вектор с комплексным числом и сохранение результата

Производительность

Насколько это эффективно?

Benchmark Окружение

CMSIS-DSP Test Framework[1]

Тестовый фреймворк CMSIS-DSP из репозитория с исходным кодом библиотеки.

```
Group : Root  (1)
  Group : DSP Benchmarks  (1)
    Group : Spectral Transformations  (9)
      Suite : Mel Frequency Cepstral Coefficients Q15  (1)
        Profile MFCC (test_mfcc_q15 - 1) : PASSED (cycles = 49051)
         256,20,13    <- 256 точек входной вектор, 20 fbank и 13 выходных коэффициентов
        Profile MFCC (test_mfcc_q15 - 1) : PASSED (cycles = 50347)
         256,20,20
        ...
```

CloudBEAR ядра для профилирования

BM-310: Компактное single issue ядро с 3х стадийным конвейером

BR-351: Быстрое 32х битное Gen1 Dual Issue ядро с длинным конвейером

BR-651: RV64 версия BR-351

[1] *https://github.com/ARM-software/CMSIS-DSP/blob/main/Testing/README.md*

BEAR-DSP: Выборочные Результаты

CloudBEAR

Ускорение - во сколько раз код с использованием P эффективнее по циклам платформа-независимого кода, исполненного на том же ядре.

Функция	Ускорение		
	BM-310	BR-351	BR-651
cfft_q15	4,0	3,3	8,0
cfft_q31	1,2	1,2	3,0
biquad_cascade_q15	4,7	2,9	1,5
biquad_cascade_q31	3,9	3,2	1,7
dot_prod_q7	5,3	5,4	7,7
pid_q31	1,5	1,3	1,5
sincos_q31	3,5	3,8	1,5

Цветовая маркировка значений:

- **≤ Квартиль 1 (1,5)**
- **≥ Квартиль 3 (4,0)**

Motor Control

CloudBEAR

Прототип двух модулей системы управления мотором (вычисления с 32 битными значениями) включает алгоритмы:

- Sin и Cos
- Преобразование Кларка и Парка
- ПИД регулятор
- Space Vector Modulation (SVM)
- CORDIC алгоритм для нелинейных функций

	Тактов CPU на итерацию	Ускорение к Baseline	Уменьшение числа тактов [%]
BM-310	539	x1.47	32%
BR-351	404	x1.66	40%

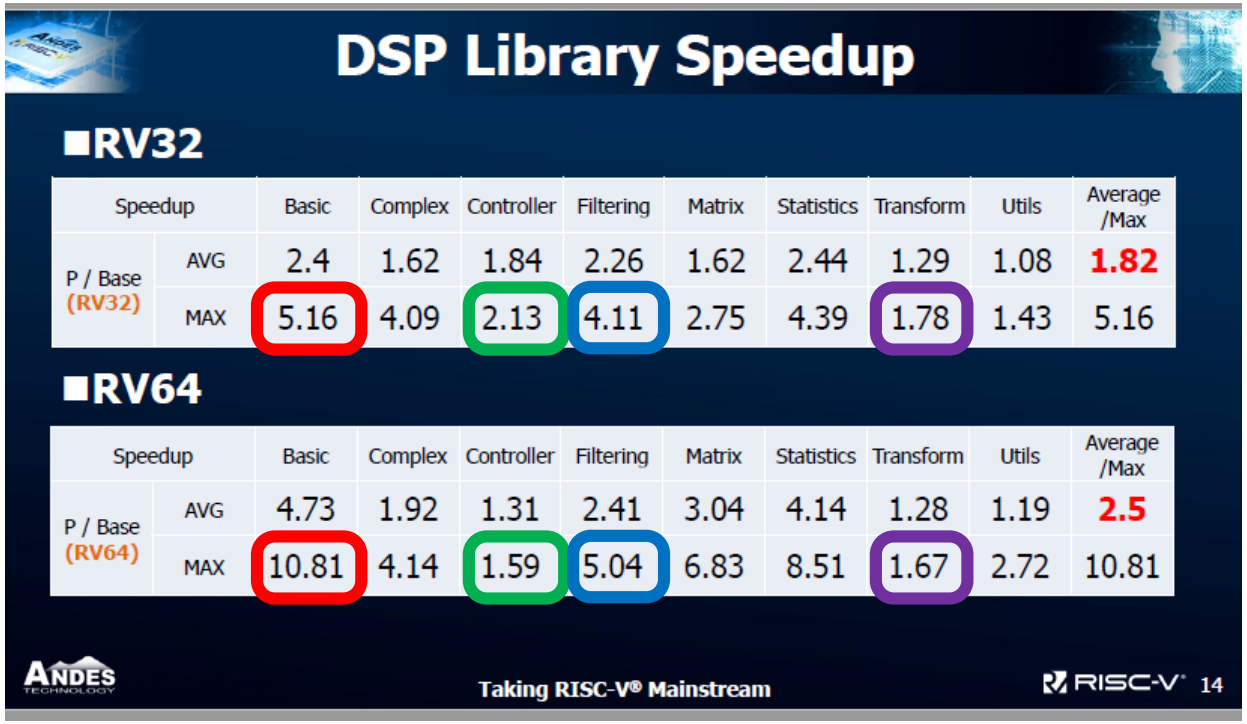
Сравнение с Конкурентами: Andes

CloudBEAR

Результаты Andes опубликованы[1] в 2019 для ядер AndeStar™ V3 в рамках ранних этапов работы над спецификацией P расширения.

MAX Speedup

Функции	BM-310	BR-351	BR-651
	RV32	RV32	RV64
BasicMath	8,05	8,77	11,05
Controller	5,81	3,16	1,71
Filtering	4,69	4,78	3,46
Transform	3,99	3,36	8,04

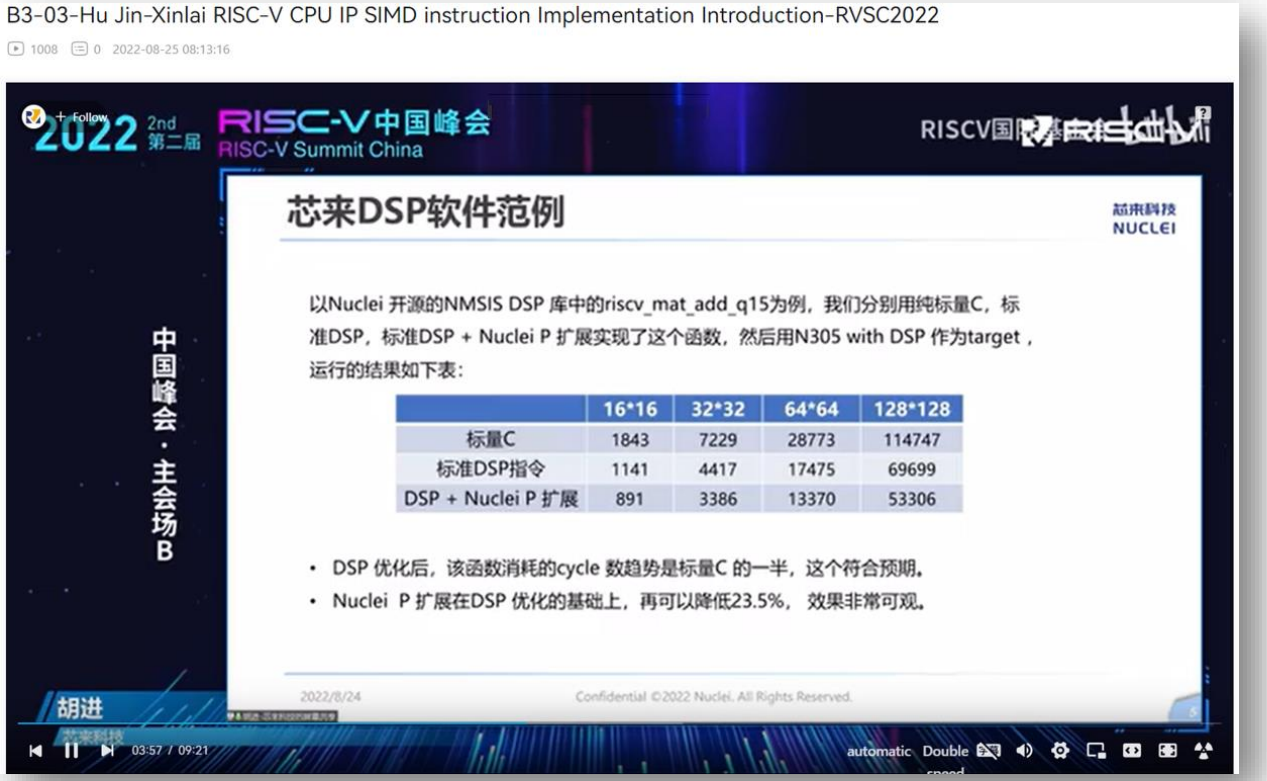


[1] *Status update of RISC-V P extension task group (Chuan-Hua Chang @ RISC-V Workshop Zurich 2019)*

Сравнение с Конкурентами: Nuclei

- Подходящие для сравнения результаты Nuclei опубликованы на RISC-V Summit China 2022.
- *`riscv_mat_add_q15`* функция практически идентична *`riscv_add_q15`*. Для последней имеется портированный на P аналог в BEAR-DSP.

Matrix Size	Basic Ops	Nuclei N305[1] [cycles]	BM-310 [cycles]	BR-351 [cycles]
16x16	256	891	757 (-15%)	704 (-21%)
32x32	1024	3386	2773 (-18%)	2220 (-34%)
64x64	4096	13370	10836 (-19%)	8301 (-38%)
128x128	16384	53306	43092 (-19%)	32859 (-38%)



[1] *Nuclei N305: 32x битное single issue ядро с 3х стадийным конвейером. Корректно сравнивать с BM-310.*

Сравнение с Конкурентами: ARM

CloudBEAR

Производительность поэлементных операций в сравнении с Arm Cycle Models[1]

Eltwise Case [Type: size]	Cortex -M7 [cycles]	BR-351 [cycles (to M7)]		Cortex M55 [cycles]	BR-651 [cycles (to M55)]
Add Q31: 16	266	217 (-18%)		168	191 (+14%)
Add Q31: 256	3557	1219 (-66%)		1848	666 (-64%)
Mult Q31: 16	364	260 (-29%)		172	198 (+15%)
Mult Q31: 256	5052	1479 (-71%)		1912	803 (-58%)

Производительность Комплексного FFT в сравнении с STM32[2] MCUs

CFFT [Type: Size]	Cortex-M4 [cycles]	BM-310 [cycles (to M4)]		Cortex-M7 [cycles]	BR-351 [cycles (to M7)]	BR-651 [cycles (to M7)]
Q15: 64	3509	2140 (-39%)		2994	1729(-42%)	1198 (-60%)
Q15: 1024	77371	43171 (-44%)		56898	37871(-33%)	22861 (-60%)
Q31: 64	6007	3369 (-44%)		4537	2842 (-37%)	1684 (-63%)
Q31: 1024	144214	81147 (-44%)		93725	66769 (-29%)	37875 (-60%)

[1]Jason Andrews. How to use the Arm Cortex-M55 Processor with the open-source CMSIS library. ARM Community Blogs (https://community.arm.com) March 11, 2020
[2]AN4841: Digital signal processing for STM32 microcontrollers using CMSIS . Application note (https://www.st.com) Feb 23, 2018

Заключение

Что в итоге?

Ключевые Выводы

CloudBEAR

- Компактное расширение для **целочисленной арифметики на основном регистровом файле** и (почти) без CSR.
- Ускорение за счёт комбинированных **(fused) операций, применяемых над множеством данных** за одну инструкцию.
- **Прирост** производительности BEAR-DSP с P от +**30% до +1000%**.
- На рынке **уже есть процессоры с P** в нератифицированной версии спецификации.
- В текущем виде спецификация маловероятно, что будет принята. **Слишком много opcodes резервируется**.
- Ключевое направление изменений – **разбиение и сокращение числа операций**.
- Производительность процессорных ядер **CloudBEAR** с P **лучше,** чем у конкурентов по открытым данным.


CloudBEAR

BackUp

BEAR-DSP: Выборочные Результаты

CloudBEAR

Функция	Сложность[1]	Метрика	BM-310	BR-351	BR-651
cfft_q15	**N*log2(N)**	**Op/cycle**	0.24	0.31	0.56
		SpeedUp	x4.0	x3.3	x8.0
cfft_q31	**N*log2(N)**	**Op/cycle**	0.14	0.17	0.31
		SpeedUp	x1.2	x1.2	x3.0
biquad_ cascade_q15	**5M*N**	**Op/cycle**	0.48	0.53	0.44
		SpeedUp	x4.7	x2.9	x1.5
biquad_ cascade_q31	**5M*N**	**Op/cycle**	0.44	0.54	0.65
		SpeedUp	x3.9	x3.2	x1.7
dot_prod_q7	**N**	**Op/cycle**	0.65	0.75	1.1
		SpeedUp	x5.3	x5.4	x7.7
pid_q31	**4N**	**Op/cycle**	0.13	0.16	0.25
		SpeedUp	x1,5	x1,3	x1.5
sincos_q31	**N**	**Op/cycle**	0.03	0.05	0.04
		SpeedUp	x3.5	x3.8	x1.5

[1] *В формулах сложности N – размер вектора данных, M – количество стадий для каскада biquad фильтров*

- *op/cycle* - Базовые операции на цикл. Нормализованная по вычислительной сложности метрика. Показывает сколько базовых операций выполняется на 1 цикл ядра.

$$\frac{basic\_ops\_num}{spent\_cycles}$$

- *SpeedUp* - во сколько раз код с использование P эффективнее платформа-независимого кода функции по op/cycle на том-же ядре.

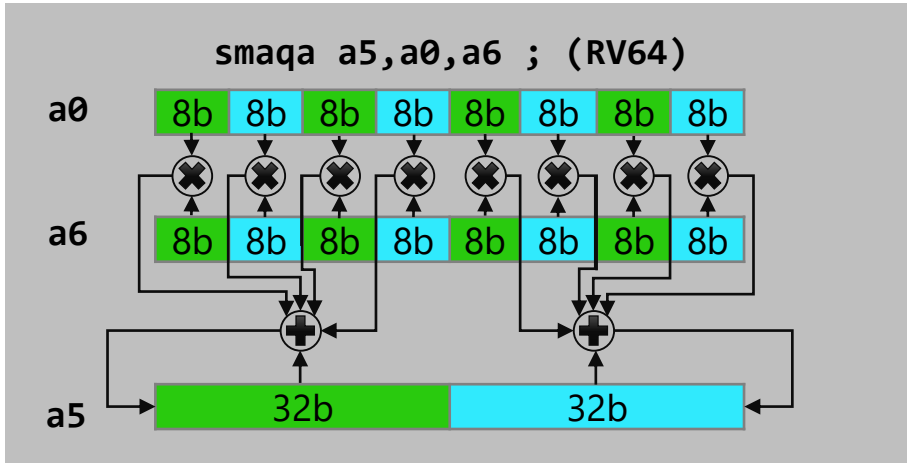
- В таблице показан максимальный результат из множества тестов заданной функции

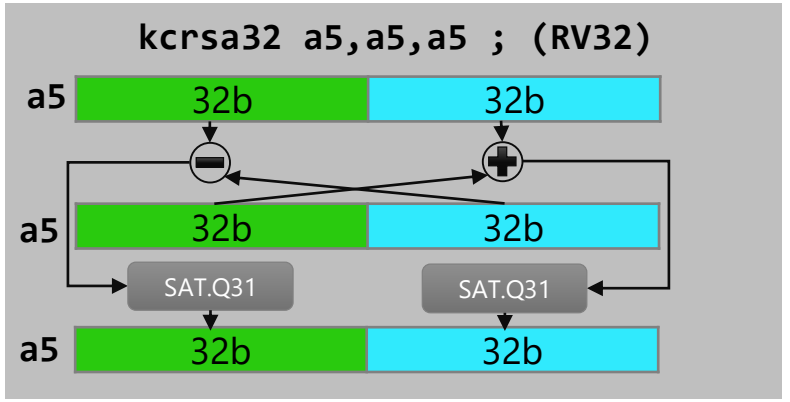
Dot Product RV64

$$y = \sum_i a_i * b_i$$

```
<.L10>:
ld      a0,0(a2) ; load {a0 to a7} (v8int8)
ld      a6,0(a1) ; load {b0 to b7} (v8int8)
addi    a2,a2,8
addi    a1,a1,8
smaqa   a5,a0,a6 ; {4x MAC, 4x Mac}
bltu    a2,a4,8a <.L10>

kcrsa32 a5,a5,a5 ; cros sub & add
sw      a5,0(a3)
```





Цветовая маркировка отрывка программы

- Загрузка 64х битных векторов (8 int8 операндов в каждом) используя стандартный ld (RV64I).
- Поэлементное умножение с накоплением результатов в двух 32b аккумуляторах внутри одного 64b регистра (RV64P)
- Чтобы сложить два 32битных аккумулятора внутри 64х битного регистра, используем cross add & sub инструкцию (RV64P). Младшее слово с результатом “add” сохраняем