Advancing the P extension

2022/9/11 Meeting

P-Extension Background

- Originally intended as packed-SIMD alternative to full vectors
- At Google RISC-V workshop Dec 2016, decision made to keep P for small SIMD in integer registers and only have single V ISA
- Andes contributed P extension proposal end of 2017, based on previous ISA design
- TG officially started in 2018
- Very large monolithic “flat” detailed proposal with no explanatory structure/rationale has been difficult to digest, so moved slowly at RVIA
- A few F2F meetings, almost no official TG meetings, and a little email traffic
- Instructions added by others over time (roughly doubled 150->300?)
- Several P-like RISC-V extensions in existence (Andes, Pulp, …)
- Overlap with Bit manip extensions removed in 2020/2021 (but still in spec)
- Arch committee put at high priority after 2021 ratifications, but has taken considerable effort to get to today’s proposed path forward
- *This is only an initial proposal to get discussion started*

Exec Summary of Proposal

- Prioritize development of a relatively comprehensive "base P" extension (could also be named "Zpa"), which contains a subset of current P proposal plus some new instructions not in current proposal
- Defer more complex and/or niche instructions to later Zp* extensions
  - Some complex instructions in current P draft replaced with two or more simpler instructions, could be added as single complex instruction in later extensions.

Goals for base P extension:

- Select instructions to optimize dense linear algebra, convolutions, and transforms that occur in DSP applications with 8b/16b/32b fixed-point datatypes (+ 64b accumulators)
- Reduce number/complexity of instructions based on common model/rationale for fixed-point arithmetic operations
- Prioritize RV32, but also support RV64
- Provide intrinsics that are portable between RV32 and RV64 P extensions
- Avoid being gratuitously incompatible with "*ISO Technical Report 18037, Programming languages - C - Extensions to support embedded processors*"
- Be competitive/superior to other architectures' DSP extensions

Notes from 7/21 Meeting

Agree on philosophy behind P to help guide decisions
Prioritize RV32 first over RV64 (but consider RV64 implications)
Create spreadsheet of all proposed instructions to track status
Find more volunteers to help with spec writing

Fixed-Point Multiply-Accumulate

- Cornerstone of most DSP operations
- Different forms of multiply and add occupy the most opcode space in P
  - data widths, signed/unsigned, widening versus promoting versus single-width, saturating versus wraparound, operand locations, …
  - “promoting” when one source operand is extended to match size of other source
  - “widening” when result is wider than source operand
- Base P extension instructions have XLEN bits of total multiplier input and accumulate in either XLEN or 2*XLEN total bits (single register or register pair)
- Individual element multiplier source operands are ≤ 32b, accumulators ≤ 64b
- Provide packed add/sub, packed multiply, and packed multiply-accumulate (MACC)
- Multiply-accumulates (MACCs) do not have saturating forms in base P
  - Often not desired
  - Adds many instructions to ISA to have both, saturating and non-saturating
  - Saturation lengthens critical path
  - Possible in an extra Zp* extension

Add & Multiply Categories, Base P

Packed Adds (rd = rs1 + rs2)

- Element operands up to 64b
- Arithmetic variants: integer, saturating (signed/unsigned), averaging (signed/unsigned)

Packed Multiplies (rd = rs1 * rs2)

- Element source operands up to 32b
- Arithmetic variants: integer, fractional-saturating (signed/unsigned/signed-unsigned)

Packed Multiply-Accumulates (rd += rs1 * rs2)

- Element source operands up to 32b
- Arithmetic variants: integer, fractional (signed for all sizes; unsigned and signed-unsigned for 8b only), NO SATURATION

MACC Instruction Widening Patterns

- Widening integer (2xb += xb * xb) x=8,16,32
- Widening fractional (2xb += (xb*xb)>>(x-1)), x=(8?),16, 32
- Promoting fractional (2xb += (2xb*xb)>>x), x=8,16
- Quad-widening integer (4xb += xb * xb) x=8,16

- RV32 element-wise widening instructions use register-pair for wider accumulator
- Dot-product instructions sum across elements to get wider accumulator
  - e.g., x[rd][31:0] += x[rs1][15:0]*x[rs2][15:0] + x[rs1][31:16]*x[rs2][31:16]
  - But also, x[rd+1][31:0],x[rd][31:0] += x[rs1][15:0]*x[rs2][15:0] + x[rs1][31:16]*x[rs2][31:16]

- Separate packed multiply and packed add instructions can support other multiply-accumulate patterns, including saturating accumulation

Register-Pair Operands in Base P

- Register pair operands are aligned even-odd only (e.g. x6 & x7, not x5 & x6).
- For RV32, no saturating or averaging (halving) adds/subtracts for doublewords; only plain ADDD, SUBD.
- For RV64, no register pairs in base P, but maybe in a future Zp* extension (packed-SIMD only, not quadword scalars).

- RV32 should have maximal packed-SIMD adds/subtracts and related ops in register pairs (each operand is 2 × XLEN).
- But packed-SIMD multiply operands should not exceed equivalent size of a single register (XLEN each). [Right?] Products for some instructions will be double-wide (2 × XLEN).
  - For RV64, more multiplier hardware possible with future register-pair Zp* extension. (But same not expected for RV32, because better to move to RV64 first.)

Support in Base P, but Deprioritize

- Multiply-accumulate with saturation of addition
  - Usually best to avoid.
  - Use two instructions: multiply, then add with saturation.
- Multiply-subtract-accumulate
  - Rarely occurs?
  - Use two instructions: multiply, then subtract.
- Unsigned × unsigned multiply-accumulate, for > 8b
  - Use two instructions: unsigned multiply, then add.

    [Or should support, because easy?]
- Complex numbers, FFTs for 8b
  - Compute in 16b instead.

Proposed to Exclude from Base P (1)

Multiply-accumulate with saturation:

KDMABB, KDMABT, KDMATT, KMABB, KMABT, KMADA, KMADRS, KMADS, KMAR64, KMATT, KMAXDA, KMAXDS, KMMAC, KMMAC.U, KMMAWB, KMMAWB.U, KMMAWB2, KMMAWB2.U, KMMAWT, KMMAWT.U, KMMAWT2, KMMAWT2.U, KMMSB, KMMSB.U, KMSDA, KMSR64, KMSXDA, KMXDA, UKMAR64, UKMSR64

Multiply and subtract-accumulate:

KMMSB, KMMSB.U, KMSDA, KMSR64, KMSXDA, MSUBR32, SMSLDA, SMSLXDA, SMSR64, UKMSR64, UMSR64

64b saturation:

KADD64, KMAR64, KMSR64, KSUB64, UKADD64, UKMAR64, UKMSR64, UKSUB64

64b averaging (halving) add:

RADD64, RSUB64, URADD64, URSUB64

Proposed to Exclude from Base P (2)

Uncommon subtract direction (but other direction is supported):

SMALDS, SMDS

Unsigned interleaved add-subtract, subtract-add:

UKCRAS16, UKCRSA16, UKSTAS16, UKSTSA16, URCRAS16, URCRSA16, URSTAS16, URSTSA16

“Straight” interleaved add-subtract, subtract-add:

KSTAS16, KSTSA16, RSTAS16, RSTSA16, STAS16, STSA16, UKSTAS16, UKSTSA16, URSTAS16, URSTSA16

Absolute value with saturated signed result [maybe?]:

KABS8, KABS16, KABSW

Proposed to Exclude from Base P (3)

Has four operands (must support differently):

BPICK

Others, not useful enough [?]:

KADDH, KDMABB, KDMABT, KDMATT, KDMBB, KDMBT, KDMTT, KHM8, KHMBB, KHMBT, KHMTT, KHMX8, KHMX16, KMMWB2, KMMWB2.U, KMMWT2, KMMWT2.U, KSUBH, MADDR32, MSUBR32, SMAL, SMALBB, SMALBT, SMALTT, SMMWB.U, SMMWT.U, SMULX8, SMULX16, UKADDH, UKSUBH, UMULX8, UMULX16

This list is mostly multiply/add. Architecture review continues for most other instruction categories.

Rename P Instructions, Examples

ADD8, ADD16, SUB8, SUB16	PADD.B, PADD.H, PSUB.B, PSUB.H
CRAS16, CRSA16	PAS.HX, PSA.HX
KHM16	PMULQ.H
PBSAD, PBSADA	PDIFSUMU.B, PDIFSUMAU.B
RADD8, RADD16, RSUB8, RSUB16	PAADD.B, PAADD.H, PASUB.B, PASUB.H
KCRAS16, KCRSA16	PSAS.HX, PSSA.HX
SMALDA, SMALXDA	PM2WADDA.H, PM2WADDA.HX
SMDRS, SMXDS	PM2SUB.H, PM2SUB.HX
SMAQA, SMAQA.SU	PM4ADDA.B, PM4ADDASU.B
SMUL8, SMUL16	PWMUL.B, PWMUL.H

For RV32:

KADDW, KSUBW, RADDW, RSUBW	SADD, SSUB, AADD, ASUB
KWMMUL, KWMMUL.U	MULQ, MULQR
MULR64, SMAR64, SMMUL.U	WMULU, WMACC, MULHR
SMBB16, SMBT16, SMTT16	MUL.H00, MUL.H01, MUL.H11

Equivalent Instructions Without Saturating Addition

KMADA, KMAXDA	PM2ADDA.H, PM2ADDA.HX
KMADRS, KMAXDS	PM2SUBA.H, PM2SUBA.HX
KMDA, KMXDA	PM2ADD.H, PM2ADD.HX

For RV32:

KMABB, KMABT, KMATT	MACC.H00, MACC.H01, MACC.H11
KMAR64, UKMAR64	WMACC
KMMAC, KMMAC.U	MHACC, MHRACC
KMMAWB, KMMAWT	MHACC.H0, MHACC.H1

Proposed New Instructions in Base P

- RV32 register-pair packed-SIMD add/sub, others
  - Cheap when already have register-pair 64b add/sub
- Fill in some missing combinations
  - Signed × unsigned multiplies, for 8b, 16b data
  - Widening mul-accums (current P has widening multiply, but without accum)
  - For most instructions with accum $\longrightarrow$ version without accum
- Widening add, add-accumulate?
  - Useful operations, because adds are so common
- Some others
  - Replacements for four-operand BPICK
  - PREDSUM (reduction sum, like VREDSUM)
  - SSH1SADD (saturating shift, saturating add) ?
  - … to be discussed later

Define Intrinsics Identical for RV32, RV64

Examples:

Exact intrinsics and names to be decided

-
  ```
  int16x4_t a, b, c;
  c = rvp_add_s16(a, b);
  ```

  RV32: `padd.dh rd,rs1,rs2` Register-pair operands ('D' for *doubleword*)

  RV64: `padd.h rd,rs1,rs2` Single register operands

-
  ```
  int16x4_t a, b;
  int64_t sum;
  sum = rvp_dotadd_s16(a, b, sum);
  ```

  RV32: `pm2adda.h rd,rs1,rs2` Two elements per instruction
  `pm2adda.h rd,rs1,rs2`

  RV64: `pm4adda.h rd,rs2,rs2` All four elements in one instruction

Coordinate with Other Extensions

Some instructions (e.g. bit merge) useful also outside P.