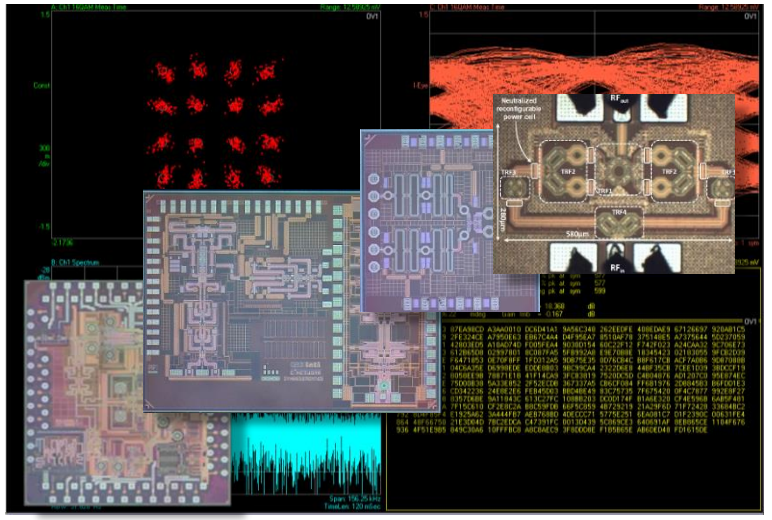
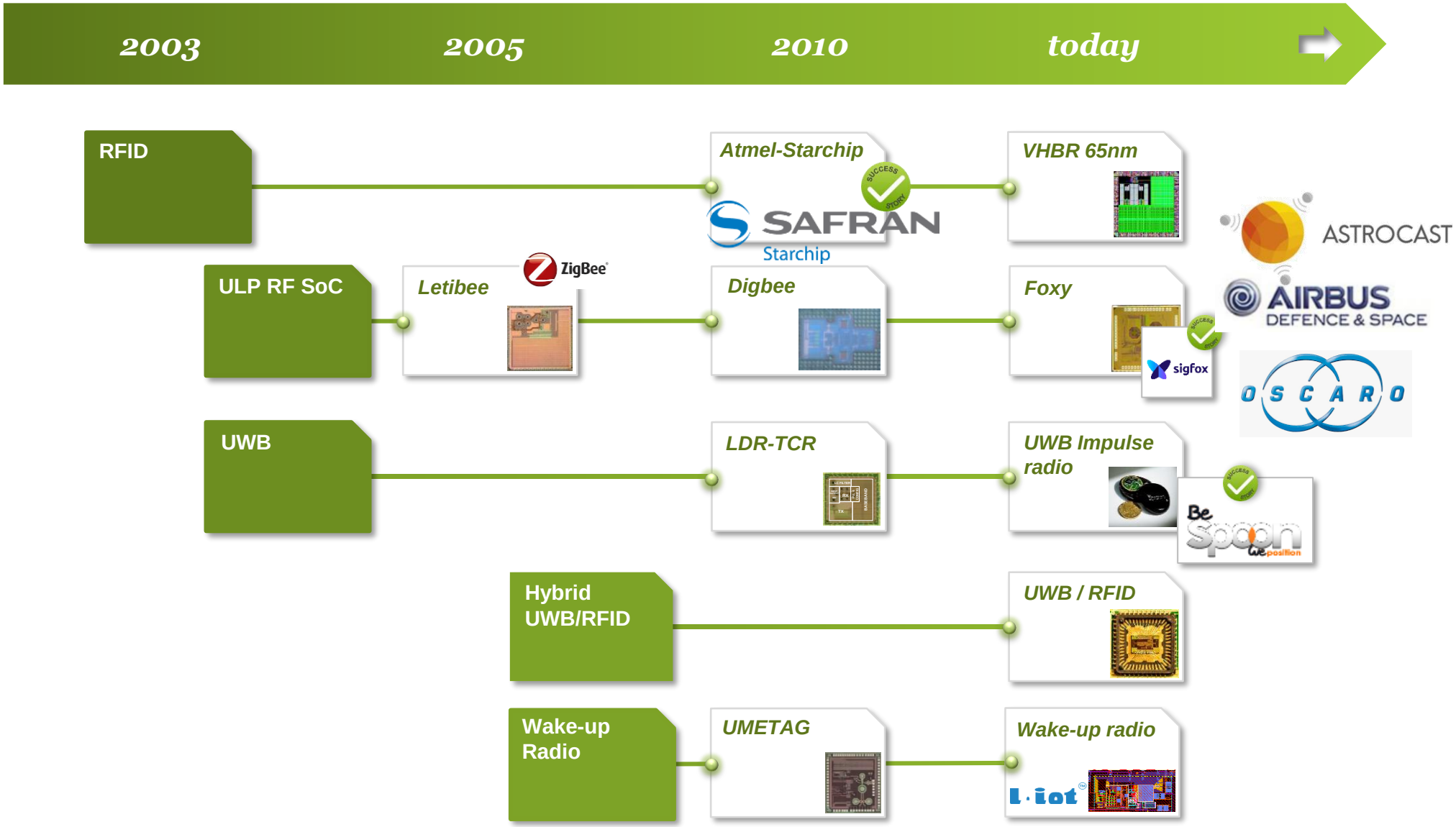


A RISC-V ISA EXTENSION FOR ULTRA-LOW POWER IOT WIRELESS SIGNAL PROCESSING

Carolynn Bernier, Hela Belhadj Amor, Zdeněk Přikryl

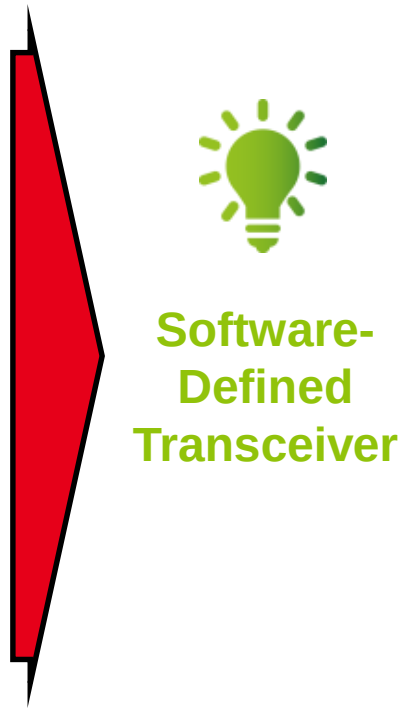
Oct 1, 2019

ULP WIRELESS DESIGN @ LETI



Motivation : A software-defined "Smart" wireless transceiver for IoT

- **PHY-agnostic solution for LPWA-IOT**
  - Address « multi-mode » markets and lower hardware bug fix costs
- **Offer future-proofed designs to our clients**
  - Our clients' advanced prototypes have evolving needs : satellite-IoT, Ultra-wide band localization, LPWA-IoT.
- **A new experimental platform**
  - Design new "RF software sensors"
  - Use light-weight ML algorithms to extract information from the RF signal



SOFTWARE RADIO FOR ULP IOT

- **Bottleneck : Existing software-defined radio (SDR) solutions are NOT ULP !**

TABLE IX
COMPARISON OF EXISTING SDR PLATFORMS

	Programm-ability	Flex-ability	Port-ability	Modul-arity	Computing Power	Energy Efficiency	Soft Core	FPGA	Cost (USD)
Imagine-based [151]	✓	×	×	×	Medium	Low	Imagine Stream Processor	N/A	N/A
USRP X300 [17]	✓	✓	×	✓	High	Low	PC	Xilinx Kintex-7	$\sim 4-5K$ Total
USRP E310 [17]	✓	✓	✓	✓	High	High	Dual-core ARM Cortex-A9	Xilinx Artix-4	$\sim 3K$ Total
KUAR [34]	✓	×	×	×	Medium	Low	Pc + 2× PowerPC cores	Xilinx Virtex II Pro	N/A
LimeSDR [146]	✓	✓	×	✓	High	Low	PC	Intel Cyclone IV	~ 300 Board Only
Ziria [147]	✓	✓	×	×	High	Low	PC	Depends on App	N/A
Sora [18]	✓	✓	×	×	High	Low	PC	Xilinx Virtex-5	~ 900 Board Only
SODA [67]	✓	✓	✓	×	High	High	ARM Cortex-M3 + Processing Elements	N/A	N/A
Iris [148]	✓	✓	✓	✓	High	High	Dual-core ARM Cortex-A9	Xilinx Kintex-4	$\sim 1.2K$ Total
Atomix [19]	✓	✓	✓	✓	High	Medium	TI 6670 DSP	N/A	~ 200 DSP Only
BeagleBoard-X15 [159]	✓	✓	✓	✓	High	Medium	2× TI C66x DSPs + 2× ARM Cortex-A15 & 2× M4	N/A	~ 270 Board Only
Airblue [20]	✓	✓	✓	✓	High	High	N/A	Intel Cyclone IV	$\sim 1.3K$ Board Only
WARP v3 [21]	✓	×	✓	✓	High	High	2× Xilinx MicroBlaze cores	Xilinx Virtex-6	$\sim 7K$ Total
PSoC 5LP [164]	✓	×	✓	✓	Low	High	ARM Cortex-M3	N/A	10 Board Only
Zynq-based [166]	✓	✓	✓	✓	High	High	Dual-core ARM Cortex-A9	Xilinx Kintex-4	$\sim 1.2K$ Total

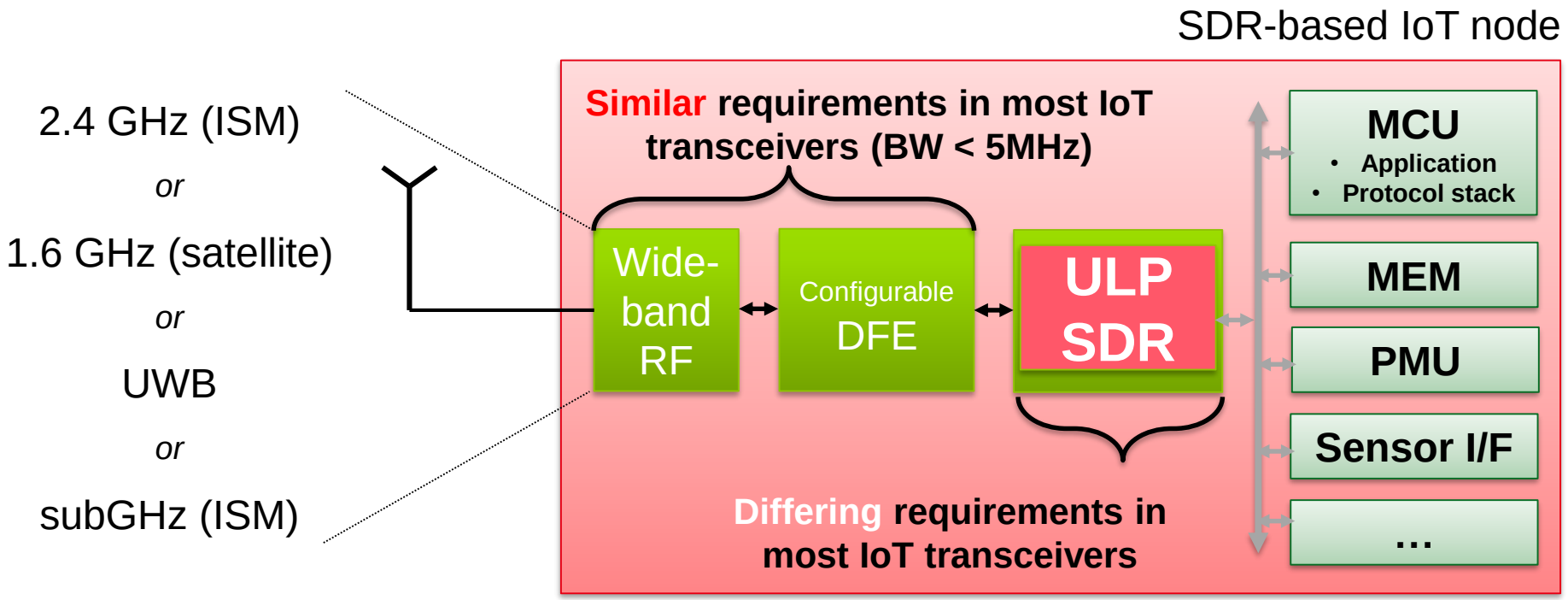
High cost (200 - 5K USD)

General purpose → High power

[Akeela, 2018]

SOFTWARE RADIO FOR ULP IOT

Solution : Design of ULP-SDR



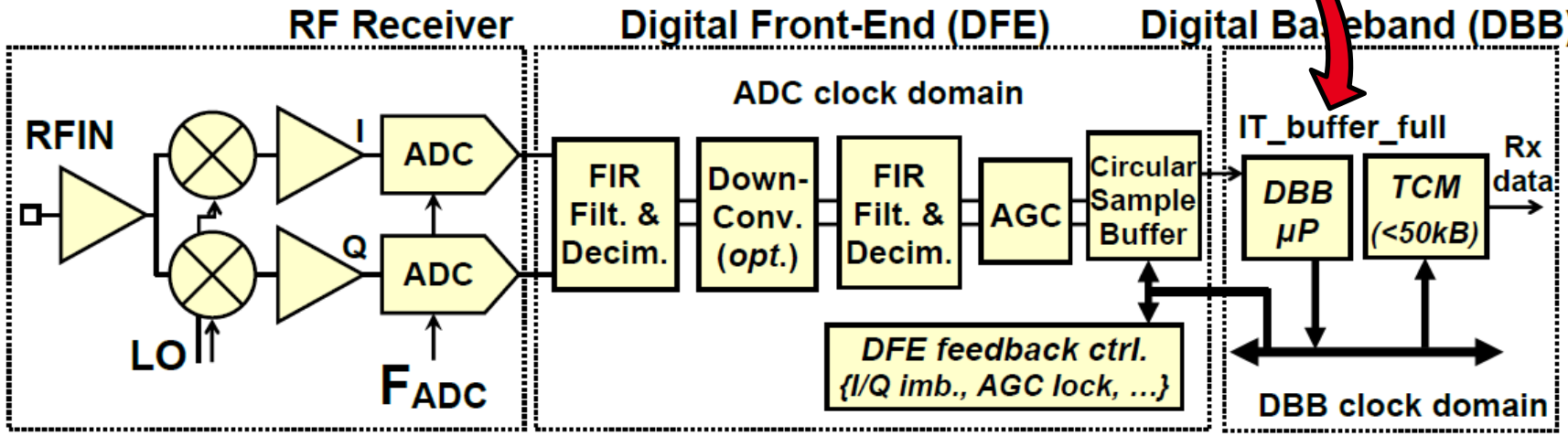
Heterogeneous multi-core platform

Challenge:

Target mW-level power consumption

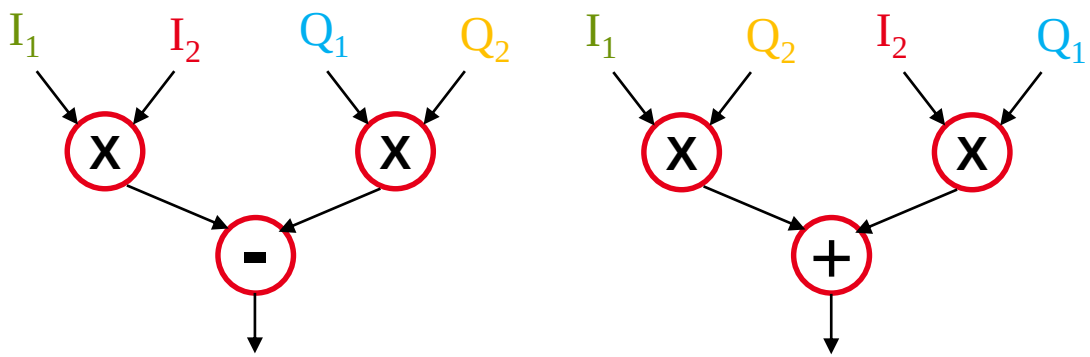
SYSTEM REQUIREMENTS

- **Target Architecture**
  - A very small and fast core (signoff ~300 MHz) associated to a TCPM and TCDM
- **Software DSP limited to decimated sample streams**
  - DFE includes easily configurable and common HW operators : FIR filters, down-converters, AGC…
- **Real-time processing of complex samples**
  - Samples are temporarily stored in sample buffer and processed in blocks
  - Integer processing only
- **Limit size of memory → big impact on power → configurable in size**
  - TCPM (high speed non volatile)
  - TCDM (stack usage !)
  - Sample buffer
  - Limit read/write to TCM
- **Single-cycle sleep**
  - Wait for next block of samples
  - Radio = OFF/ON



COMPUTING FOR WIRELESS DSP

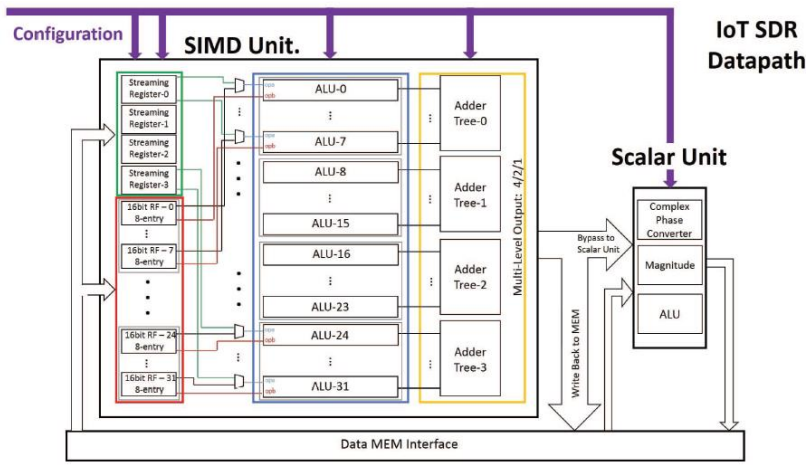
- **Wireless DSP requires linearity and low distortion**
  - Operatiors MUST NOT saturate
  - Operators MUST NOT overflow → but checking for overflows is too costly
- **Wireless DSP must conserve dynamic range (DR)**
  - The useful signal is often contained in the least significant bits
  - Beware of quantification noise → take care when rescaling the signal !
- **Most wireless signals are complex *: i(t) + j*q(t)***
  - Frequent use of MUL, ADD, SUB, MAG, SHIFT, … instructions on 8/16/32 bit **complex** data
- **Demodulation/compensation algorithms are mostly based on correlations**
  → i.e. multiplication
- **Input signal stream is typically <= 8 bits**
  - i.e. data streams are typically 8 / 16 / 32 bits
  - → fits well on a 32-bit machine



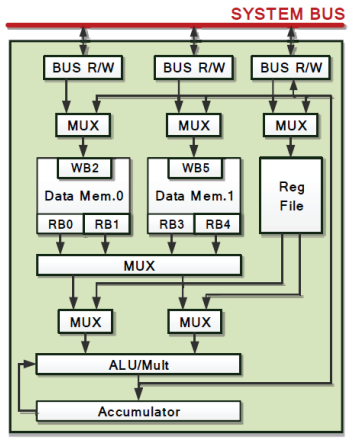
WHICH PROCESSOR FOR OUR SDR ?

- **Academic: Dedicated processors**

Custom SIMD [Chen, HPCA16]



Custom MCU [Wu, GlobalSIP16]



- Promising power consumption
- Dedicated architectures → difficult to program
- No software tool-chains
- Low frequency clock → Large surface overheads
- Inefficient use of advanced CMOS nodes

Commercial: GP processors, DSP

Previous work:

M3/M0+ vs. RISCY

[Belhadj, DATE19]

→ **Lessons learned** : GP processor can rival dedicated SoA processor architectures (with additional benefits)

→ **Lessons learned** : size of register file has huge impact on cycle count

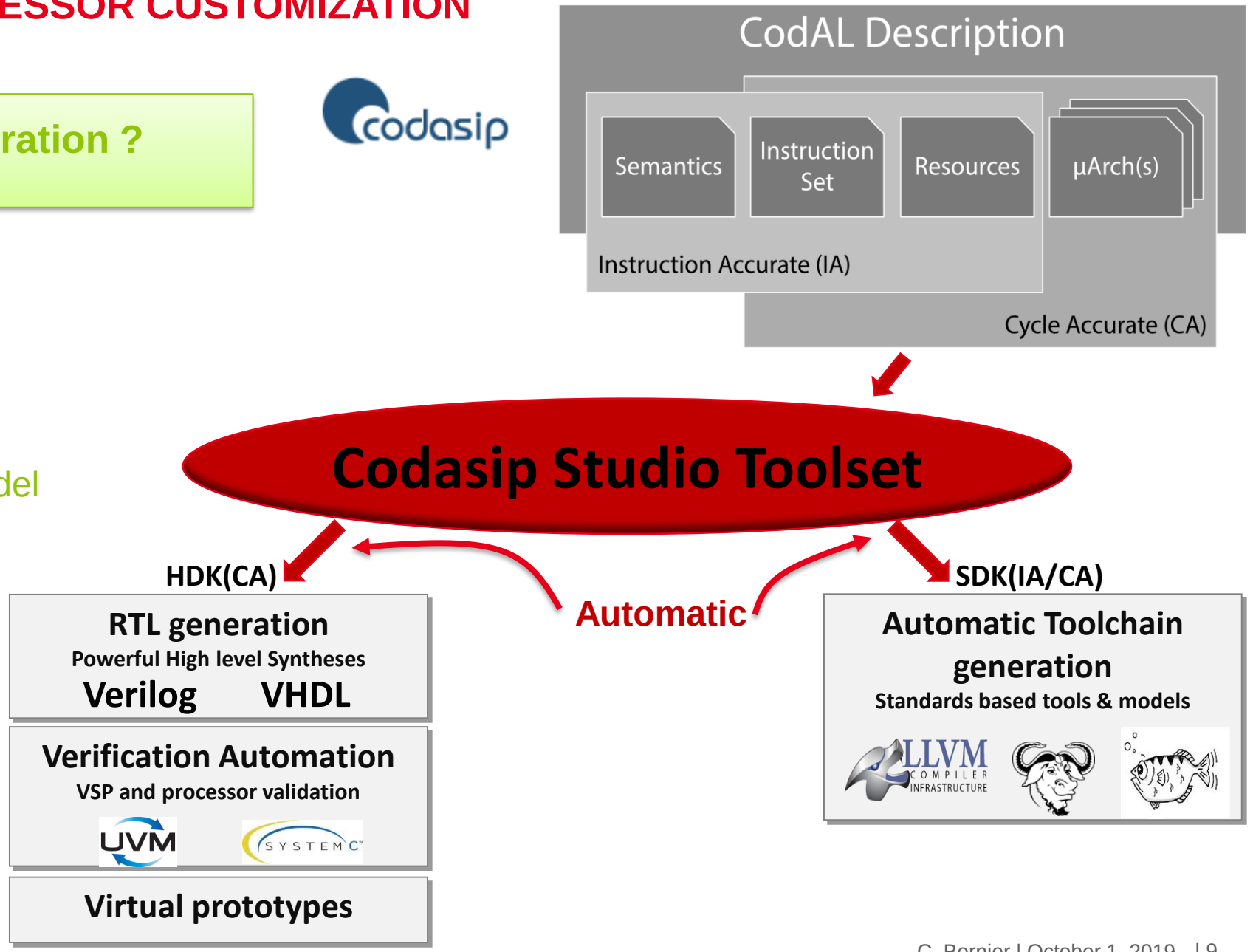
RISC-V advantage !

→ **Lessons learned** : post-increment, HW loop, SIMD → not important in our test benches (mix of DSP computing and control)

PROCESSOR CUSTOMIZATION

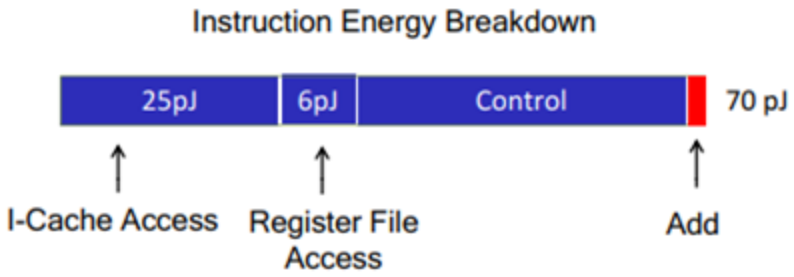
- **RISC-V-based acceleration ?**

- **Extend RISC-V ISA using dedicated instructions**
  - Codasip Studio : → An easy task ?
  - Instruction Accurate (IA) model of new instructions
  - Dedicated to RF DSP computing “zero cost” hardware implementation



EXPLORING THE INSTRUCTION JUNGLE

- **Wanted**
  - Minimal set of USEFUL instructions.
  - Only 32-bit opcodes for low decoding complexity.
- **Opportunities**
  - Wide opcodes means up to 5 operands !
  - First operation on 8-bit data is ALWAYS a complex multiplication
  - Advanced CMOS allows single-cycle operators
  - Tiny relative cost of ALU operators



Integer	
Add	
8 bit	0.03pJ
32 bit	0.1pJ
Mult	
8 bit	0.2pJ
32 bit	3.1pJ

45 nm, 0.9 V [M. Horowitz, ISSCC 2014]

REJECTED

- **More general solution prefered :**
  - Halving variants (e.g. RADD)
- **Not clearly indispensable :**
  - CSMUL (complex-scalar multiply)
- **Useless :**
  - saturating instructions, MIN/MAX, 8 bit SIMD, CONJ

PROPOSED EXTENSION

- **15 instructions using 3 major opcodes**

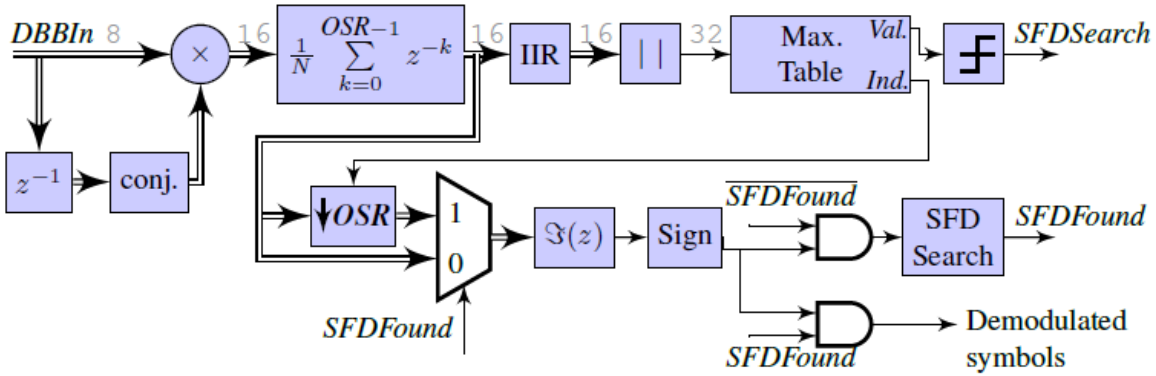
Mnemonic	Instruction	Operation
Zero-Cost Instructions		
ADDC16 rd, rs1, rs2, imm	16-bit Addition & Shift Right Arithmetic Immediate	rd.L = (rs1.L + rs2.L)>>imm rd.H = (rs1.H + rs2.H)>>imm
SUBC16 rd, rs1, rs2, imm	16-bit Subtraction	rd.L = (rs1.L - rs2.L)>>imm rd.H =(rs1.H - rs2.H)>>imm
MUL2ADD16-32 rd, rs1, rs2, imm	Two "16x16" and Signed Addition	rd = [(rs1.L * rs2.L) + (rs1.H * rs2.H)]>>imm
SRAC16 rd, rs1, imm	16-bit Shift Right Arithmetic Immediate	rd.L = rs1.L>>imm rd.H = rs1.H>>imm
SLLC16 rd, rs1, imm	16-bit Shift Left Logical Immediate	rd.L = rs1.L<<imm rd.H = rs1.H<<imm
CRASC16 rd, rs1, rs2, imm	16-bit Cross Add & Sub	rd.L = (rs1.L + rs2.H)>>imm rd.H = (rs1.H - rs2.L)>>imm
CRSAC16 rd, rs1, rs2, imm	16-bit Cross Sub & Add	rd.L = (rs1.L - rs2.H)>>imm rd.H = (rs1.H + rs2.L)>>imm
MULC8-16 rd, rs1, rs2, H1, H2, imm	Two "8x8" and Signed Subtraction Two Crossed "8x8" and Signed Addition	if Hx = 1, {ix,qx} = {rsx.B2,rsx.B3} if Hx = 0, {ix,qx} = {rsx.B0,rsx.B1} rd.L = ((i1 * i2) - (q1 * q2))>>imm rd.H = ((i1 * q2) + (i2 * q1))>>imm
MULC16 rd, rs1, rs2	Two "16x16" and Signed Subtraction Two Crossed "16x16" and Signed Addition	rd.L = (rs1.L * rs2.L)>>16 - (rs1.H * rs2.H)>>16 rd.H = (rs1.H * rs2.L)>>16 + (rs1.L * rs2.H)>>16
Low-Cost Instructions		
ADDC32 rd, rs1, rs2, imm	32-bit Addition & Shift Right Arithmetic Immediate	rd = (rs1 + rs2)>>imm rd+1 = ((rs1+1) + (rs2+1))>>imm
SUBC32 rd, rs1, rs2, imm	32-bit Subtraction	rd = (rs1 - rs2)>>imm rd+1 = ((rs1+1) - (rs2+1))>>imm
CRASC32 rd, rs1, rs2, imm	32-bit Cross Add & Sub	rd = (rs1 + (rs2+1))>>imm rd+1 = ((rs1+1) - rs2)>>imm
CRSAC32 rd, rs1, rs2, imm	32-bit Cross Sub & Add	rd = (rs1 - (rs2+1))>>imm rd+1 = ((rs1+1) + rs2)>>imm
MULC16-32 rd1, rd2, rs1, rs2, imm	Two "16x16" and Signed Subtraction Two Crossed "16x16" and Signed Addition	rd1 = ((rs1.L * rs2.L) - (rs1.H * rs2.H))>>imm rd2 = ((rs1.H * rs2.L) + (rs1.L * rs2.H))>>imm
Higher-Cost Instructions		
MULC32 rd, rs1, rs2, rs3, rs4	Two "32x32" and Signed Subtraction Two Crossed "32x32" and Signed Addition	rd = (rs1 * rs3)>>32 - (rs2 * rs4)>>32 rd+1 = (rs2 * rs3)>>32 + (rs1 * rs4)>>32

Shorthand definitions: r.B3→r[31:24], r.B2→r[23:16], r.B1→r[15:8], r.B0→r[7:0], r.H → r[31:16], r.L→ r[15:0]

- **« Zero-cost »**
  →Reconfigurable HW
  →Systematic output DR adjust
- **« Low-cost »**
  → 4 output / 2 input port register file
  → Duplicated ALU
- **« Higher-cost »**
  → 3 more 32-bit multipliers

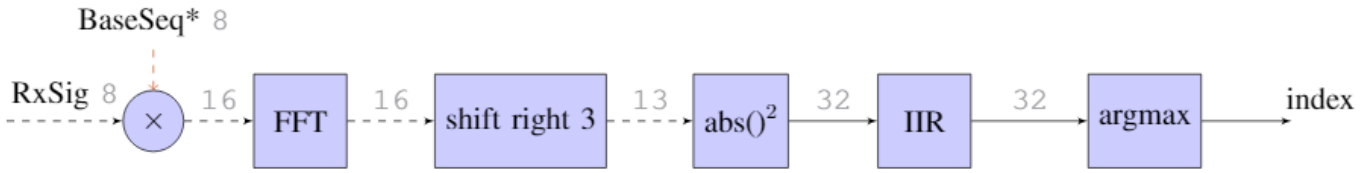
WIRELESS DSP TESTBENCHES

Testbench 1: FSK demodulation



Testbench 2: LoRa preamble synchronization

- Spreading Factor (SF) = 7, 11



Testbench 3: 16 and 32-bit FFT

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} nk} = \sum_{n=0}^{N-1} x_n W_N^{kn}$$

- Radix-4 decimation-infrequency, complex FFT with bit-reversed outputs, N = 128, 2048
- Based on source code from a port of the ARM CMSIS DSP library to RISC-V

Testbench 4: CORDIC algorithm

$$x_{i+1} = x_i - s_i.y_i.2^{-i}$$

$$y_{i+1} = y_i + s_i.x_i.2^{-i}$$

$$z_{i+1} = z_i - s_i.atan(2^{-i})$$

- 10 iteration CORDIC algorithm applied to 32-bit complex input data.

RESULTS

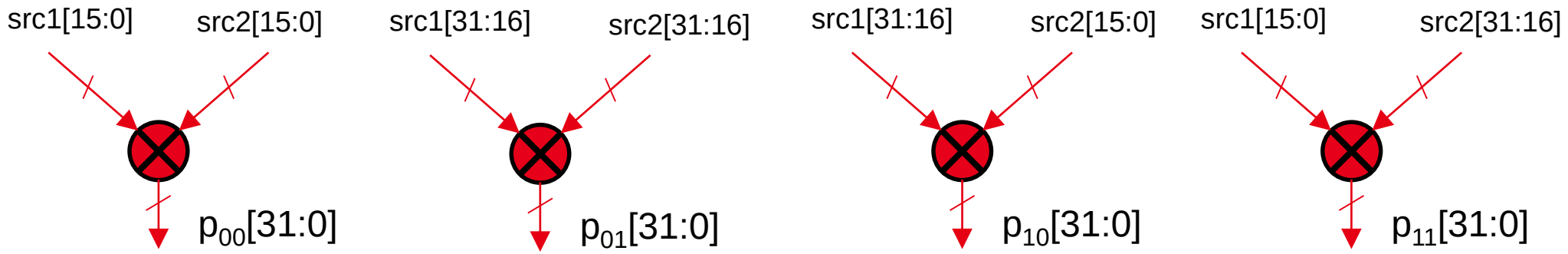
Power Model	Baseline	+Extensions
All instr. except NOP and MUL	1	*1.05*
MUL	1.14	*1.14*
MULC16-32 / MULC16	-	*1.3*
MULC32	-	*1.59*

- Expect at least ~50% power reductions with reduced clock and VDD.

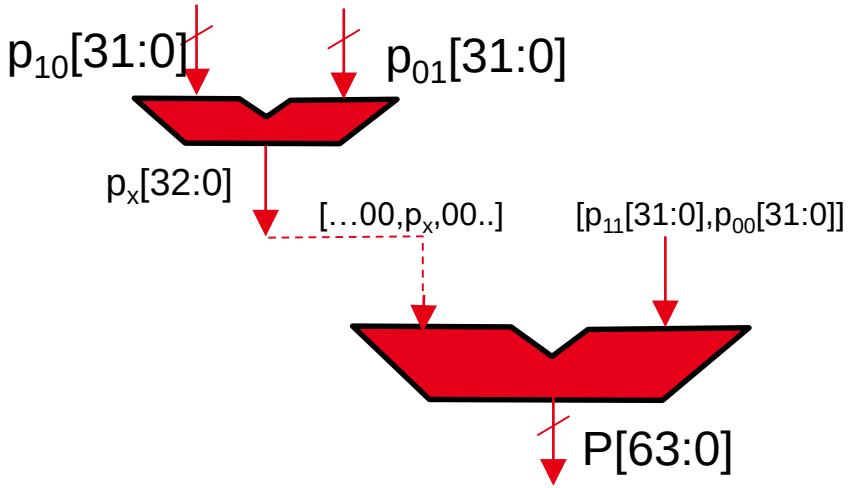
Testbench	Cycle count improvement (IA model)	*Energy improvement (est.)*
FSK Demod	22 %	
LoRa, SF=7	49 %	*46 %*
LoRa, SF=11	52 %	*50 %*
16-bit FFT, N=128	55 %	*53 %*
16-bit FFT, N=2048	57 %	*55 %*
32-bit FFT, N=128	34 %	*32 %*
32-bit FFT, N=2048	34 %	*30 %*
32-bit CORDIC, 10 iteration	28 %	

FUTURE WORK

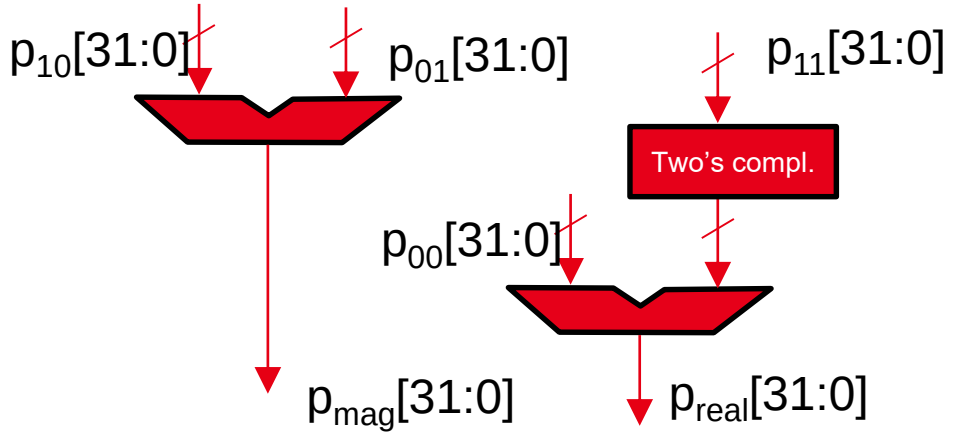
- **Finish CA model & run Power/Area analysis in 22 nm**
  - Reconfigurable hardware blocks designed in CodAL. Ex: 32-bit multiplication



CASE : 32-bit integer multiplication



CASE : 16-bit complex multiplication



Special thanks to :

Hela Belhadj Amor
Zdeněk Přikryl
Jerry Ardizzone

And

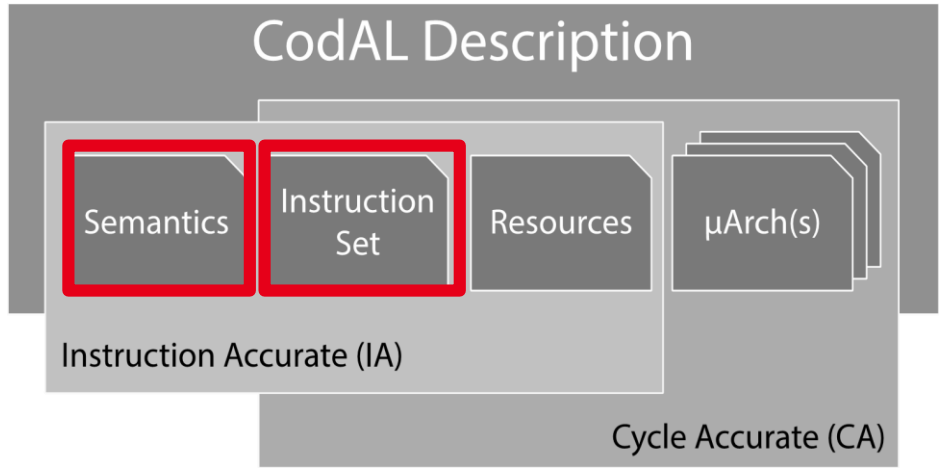
Ivan Miro Panades
Yves Durand
Henri-Pierre Charles
Simone Bacles-Min
Romain Lemaire
… and all of LISAN !

Leti, technology research institute
Commissariat à l'énergie atomique et aux énergies alternatives
Minatec Campus | 17 rue des Martyrs | 38054 Grenoble Cedex | France
www.leti.fr

INSTITUT CARNOT CEA LETI

PROCESSOR CUSTOMIZATION

- **Step 1 : ISA exploration using IA model**




```
element opc_name
{
    use instance_data_type as name of instances;

    assembler {textual form of the instruction};
    binary {The instructions's binary coding};

    semantics
    {
        The instruction's behavior is described using a subset of the ANSI C language.
    };
};
```

Used by IA and CA models

```
element i_load
{
    use opc_load as opc;
    use gpr_all as gpr_dst, gpr_src1;
    use simm12;

    assembler { opc gpr_dst "," simm12 "(" gpr_src1 ")" };
    binary { simm12 gpr_src1 opc[OPC_FRAG1] gpr_dst opc[OPC_FRAG0]  };

    semantics
    {
        ADDR_TYPE address;
        WORD_TYPE result;

        codasip_compiler_schedule_class(sc_load);

        address = rf_gpr_read(gpr_src1) + simm12;
        result = load(opc, address);
        rf_gpr_write(result, gpr_dst);
    };
};
```

Used by IA model

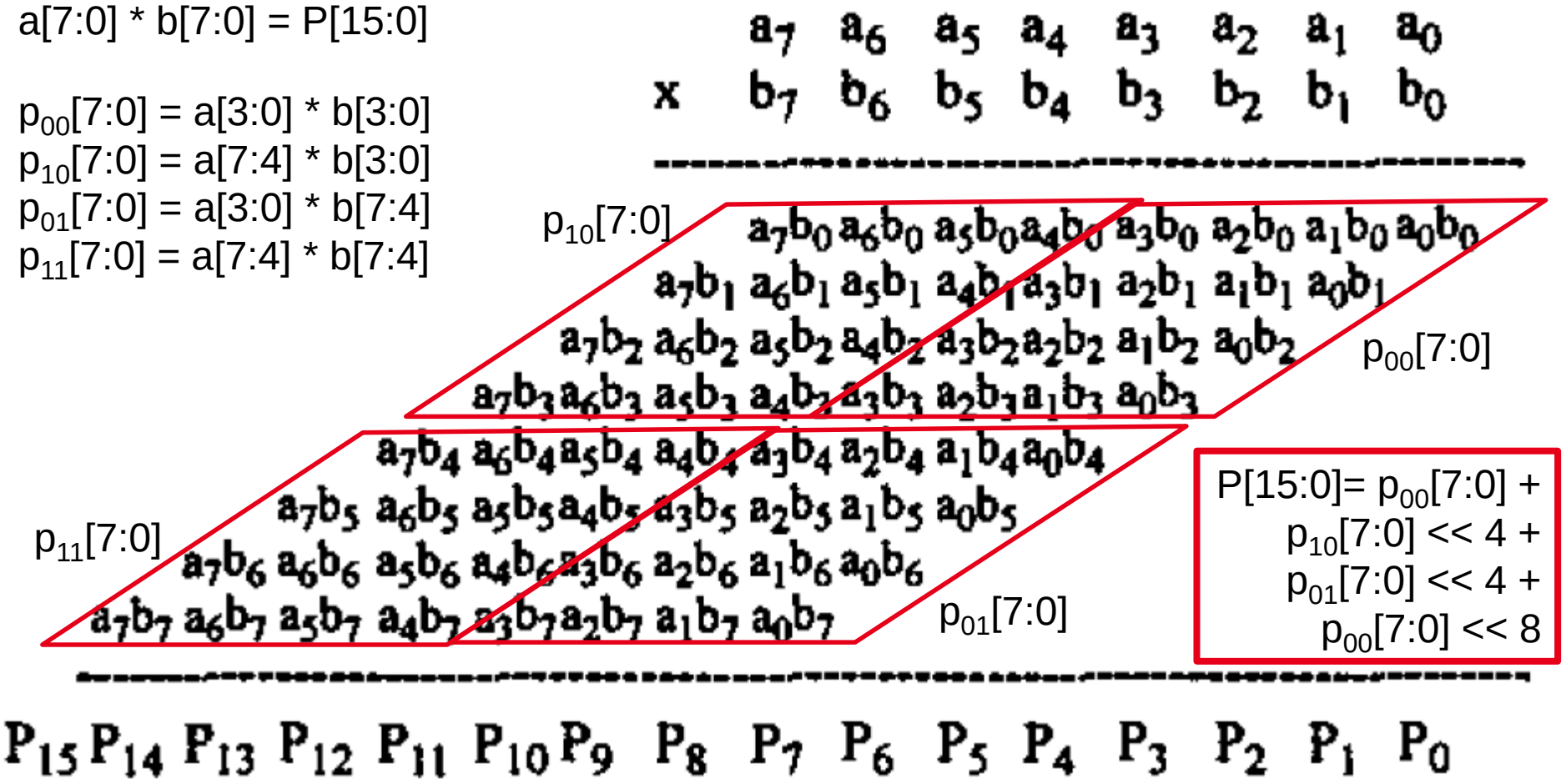
Call to memory interface if_ldst

RECONFIGURABLE MULTIPLIER (8 BIT EXAMPLE HERE)

State 1 : the block performs 8-bit integer multiplication

$a[7:0] * b[7:0] = P[15:0]$

$p_{00}[7:0] = a[3:0] * b[3:0]$
$p_{10}[7:0] = a[7:4] * b[3:0]$
$p_{01}[7:0] = a[3:0] * b[7:4]$
$p_{11}[7:0] = a[7:4] * b[7:4]$



$P[15:0] = p_{00}[7:0] + p_{10}[7:0] << 4 + p_{01}[7:0] << 4 + p_{00}[7:0] << 8$

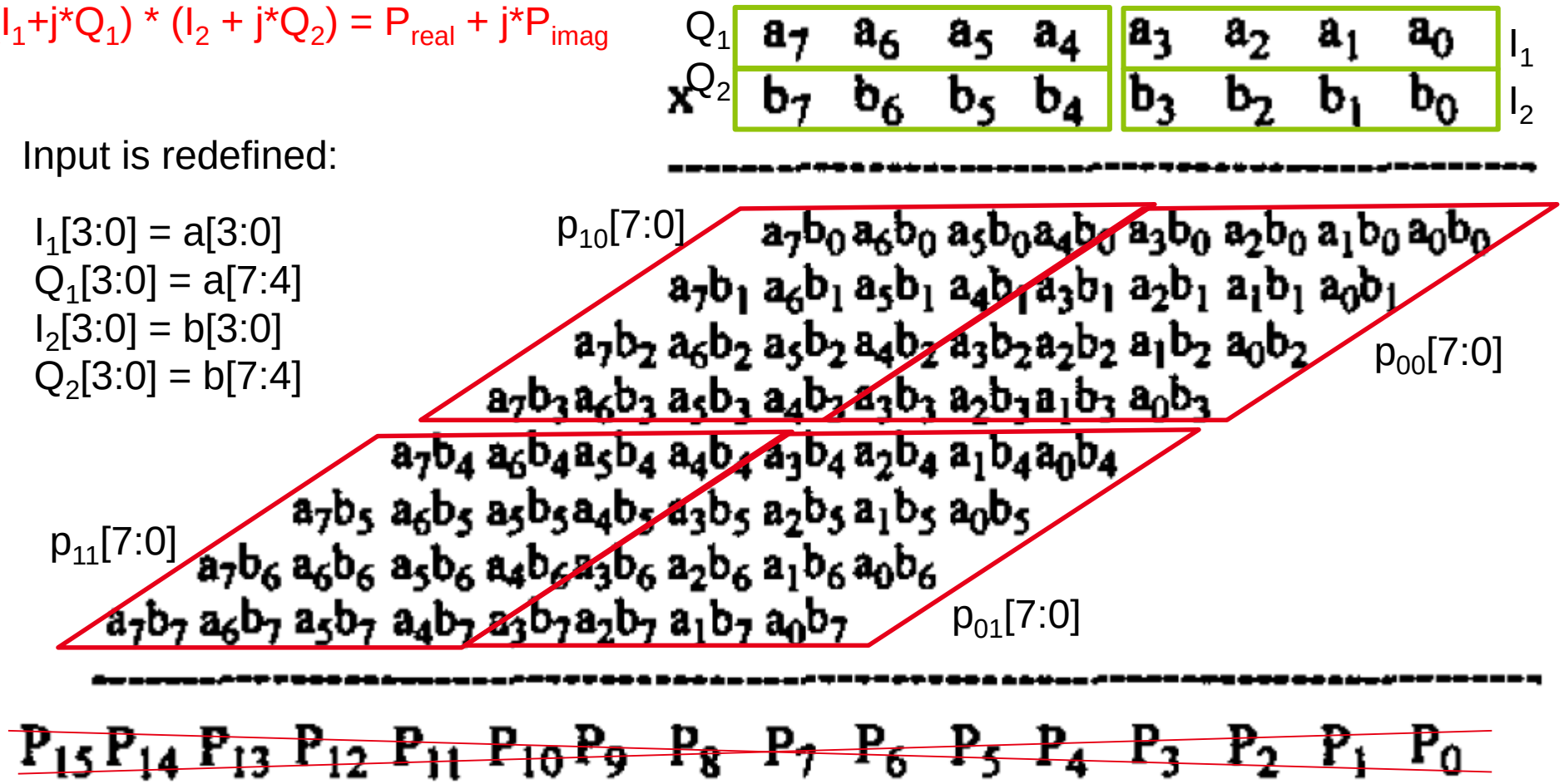
RECONFIGURABLE MULTIPLIER (8 BIT EXAMPLE HERE)

State 2 : the block performs a 4-bit **complex** integer multiplication :

$(I_1+j*Q_1) * (I_2 + j*Q_2) = P_{real} + j*P_{imag}$

Input is redefined:

$I_1[3:0] = a[3:0]$
$Q_1[3:0] = a[7:4]$
$I_2[3:0] = b[3:0]$
$Q_2[3:0] = b[7:4]$



RECONFIGURABLE MULTIPLIER (8 BIT EXAMPLE HERE)

State 2 : the block performs a 4-bit **complex** integer multiplication :

$(I_1+j*Q_1) * (I_2 + j*Q_2) = P_{real} + j*P_{imag}$

$P_{real} = I_1*I_2 - Q_1 * Q_2$
$P_{real} = p_{00}[7:0] - p_{11}[7:0]$

$P_{imag} = I_1*Q_2 + Q_1 * I_2$
$P_{imag} = p_{01}[7:0] + p_{10}[7:0]$