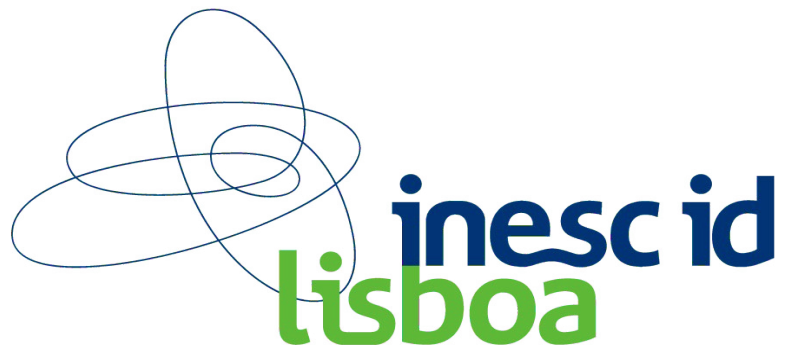


RISC-V/CGRA-based open source SoC

José T. de Sousa



Outline

- Introduction: Motivation & Objectives
- Open source FPGA vs CGRA
- Open source RISC-V processors and toolchains
- The open source Versat CGRA
- IOB-soc: an open source SoC using RISC-V processors and Versat CGRAs
- Conclusion

Intro: motivation

- Open source software has brought tremendous benefits to mankind: OS, toolchain, browser, text and media, internet, communication, etc
- HDLs turned hardware into software, so the model should apply to hardware... but we are not quite there... we're working on it!
- Hardware is... hard: synthesis, place and route, timing closure, verification, testing... What about Analog?!
- Hardware is... attractive: small size, low power, high performance – IoT, Machine Learning, Artificial Intelligence, ...

Intro: accelerating software

- Workaround: build and reuse hardware blocks to accelerate software: DSP (limited performance), GPU and FPGA (limited size and energy efficiency)
 - FPGAs are more efficient but harder to program compared to GPUs
- Open source perspective:
 - DSP: good, it is software!
 - GPUs: good, it is software!
 - FPGA: not good, it is more like hardware!!!

Intro: objectives

- Analyze open source FPGA possibilities
- Analyze open source CGRA possibilities
 - Conclude on the advantages vs FPGA
- Discuss open source RISC-V processor
- Describe open source Versat CGRA
 - Single layer architecture
 - Multi-layer architecture
- Describe open source RISC-V/Versat SoC
- Highlight achievements

Open source FPGA

- Imagine a universal open source FPGA toolchain like gcc, gdb, etc... (!)
- Nice but difficult to achieve:
 - No universal FPGA architecture (unlike CPU standard ISA)
 - Difficult to get device details from FPGA vendors (pre and post silicon)
- FPGA open source tools
 - Example: Verilogtorouting.org
 - Need to use an FPGA architecture description file – difficult for commercial FPGAs
- Open source FPGA architecture
 - Easy to get device details available for several technologies
 - Difficult to design: FPGAs are full-custom circuits
 - Interesting work by U. Toronto on synthesizable FPGA targeting standard cells (2x lower performance)

Open source CGRA

- Open source Coarse Grained Reconfigurable Array (CGRA) alternative
 - Array of ALUs instead of Fine-Grained Look Up Tables (LUTs)
 - Less flexible compared to FPGA
 - Used to accelerate program loops rather than generic code
- Easier to reach open source status for architecture and tools
 - CGRA generally described using HDL and target FPGA or ASIC
 - Can be built to avoid complex design software tools, as shown next!

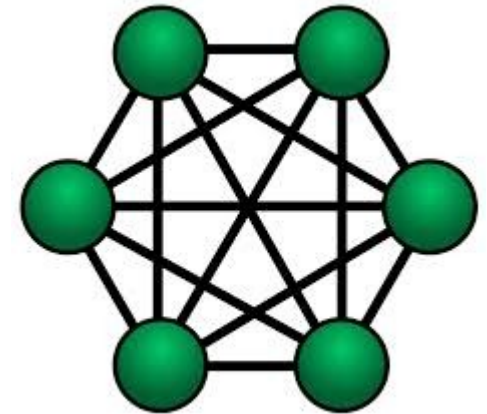
Open source RISC-V processors

- Praise predecessor OpenRISC: visionary resource-deprived community
 - Not the best ISA
 - Lots of problems with the hardware descriptions
 - Only recently supported in GCC 9
- RISC-V: doing it properly
 - Patterson, Berkeley, Silicon-Valley, Money, etc: good ISA, supported since GCC 7.1
 - Lots of problems with the hardware descriptions: need to pay to get support
 - Fairly easy to run the canned demos provided in different projects but difficult to reuse
 - Like OpenRISC, only works for the provided demos!

Open source Versat CGRA (github)

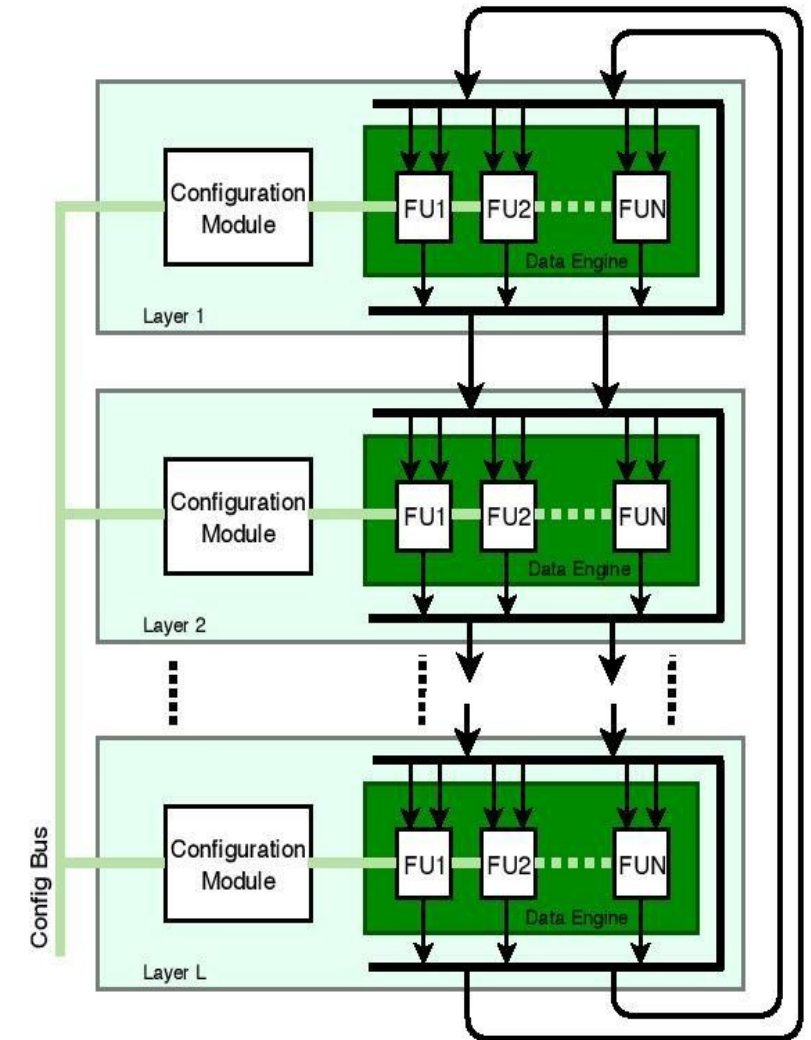


- Research project started at INESC-ID in 2014 to build a CGRA IP core
 - Currently 1 professor/researcher, 1 PhD student and 4 master's dissertations
 - Main idea: simplicity and effectiveness → Full Mesh of Heterogeneous Functional Units
 - Single clock, fully synchronous design with all functional unit outputs registered (fitting-independent delays)
- Advantages of full mesh topology
 - Trivial fitting (no place and route)
 - JIT: fitting is incorporated in application using small software lib
 - Scales well in time: reconfigure dynamically and partially for almost anything
- Disadvantages of full mesh topology
 - Spatial scaling leads to wire congestion
- Startup created in 2018 (IObundle, Lda), first customer in 2019



Multi-layer Versat CGRA

- Addresses spatial scalability
- Ring of full meshes (1-D super structure)
- Each Versat node is called a layer
- Each node as easy to program as originally
- Nodes organized in 1-D structure add little complexity to overall programming
 - Easily accomplished by loop unrolling



Versat CGRA Pre-Silicon Configure

- Node structure decided pre-silicon
- Set parameters in Verilog header file. ALU array example:

```
// Instantiate the ALUs
generate
    for ( i = 0; i < `nALU ; i = i + 1) begin : add array
xalu alu (
    . clk ( clk ),
    . rst ( run_reg ),
    //Data IO
    . data_bus ( data_bus ),
    . result ( data_bus [ 'DATA_ALU0_B - i * 'DATA_W -: 'DATA_W ] ),
    // Configuration data
    . configdata ( config_reg_shadow [ 'CONF_ALU0_B - i * 'ALU_CONF_BITS -:
'ALU_CONF_BITS ] )
```

Versat CGRA Pre-Silicon Configure

- Ring size decided pre-silicon
- Set nVERSAT macro in Verilog header file

```
// Instantiate the Versats
genvar i;
generate
    for (i=0; i < `nVERSAT; i=i+1) begin :
versat_array
    xversat versat (
        .clk(clk),
        .rst(rst),

        //Control Bus Interface
        .ctr_req(versat_ctr_req[i]),
        .run_req(run_req),
        .ctr_rnw(ctr_rnw),
        .ctr_addr(ctr_addr[`ADDR_W-1:0]),
        .ctr_data_to_wr(ctr_data_to_wr),

        .ctr_data_to_rd(versat_ctr_data_to_rd[i]),
```

Versat CGRA Post-Silicon Configure

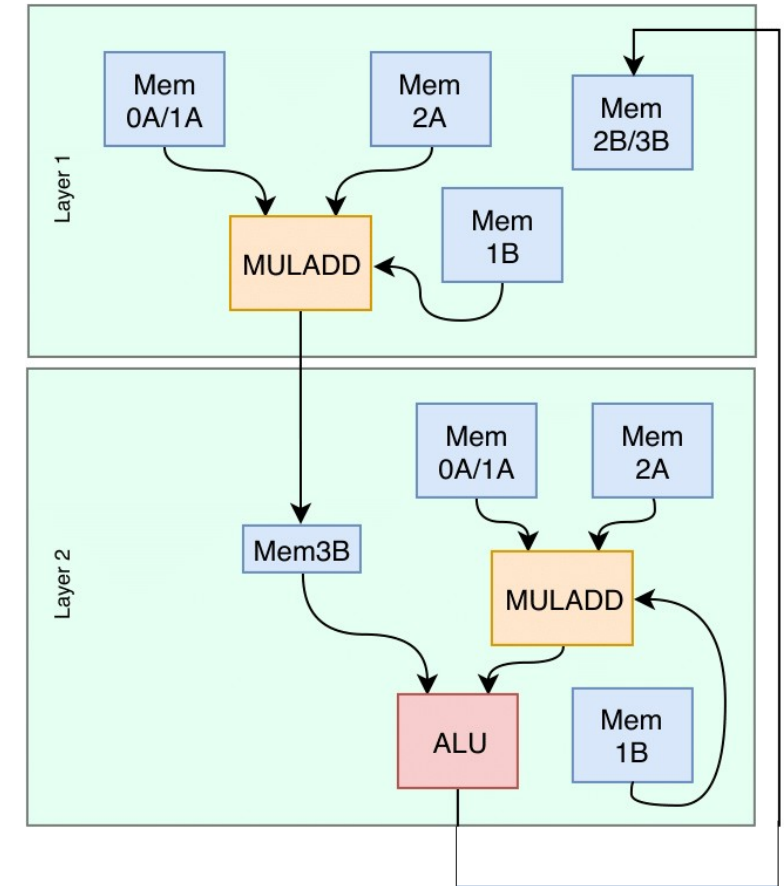
```
//Statically configure aluLite layer2 (in a, in b, function)
v2.alulite[0].setConf(sMEMB[3], sMULADD[0], ALULITE_ADD);
v2.alulite[0].writeConf();

//Statically configure result memory in layer1 (22 * 24 * 24 = 12672 elements)
//Save 1st part of the result(6336) in MEM2 of 1st Versat layer (pos 1856-8191)
v1.memPort[m2B].setConf(1856, 22, 1, 20, 5, 1, sALULITE_p[0]);
v1.memPort[m2B].writeConf();

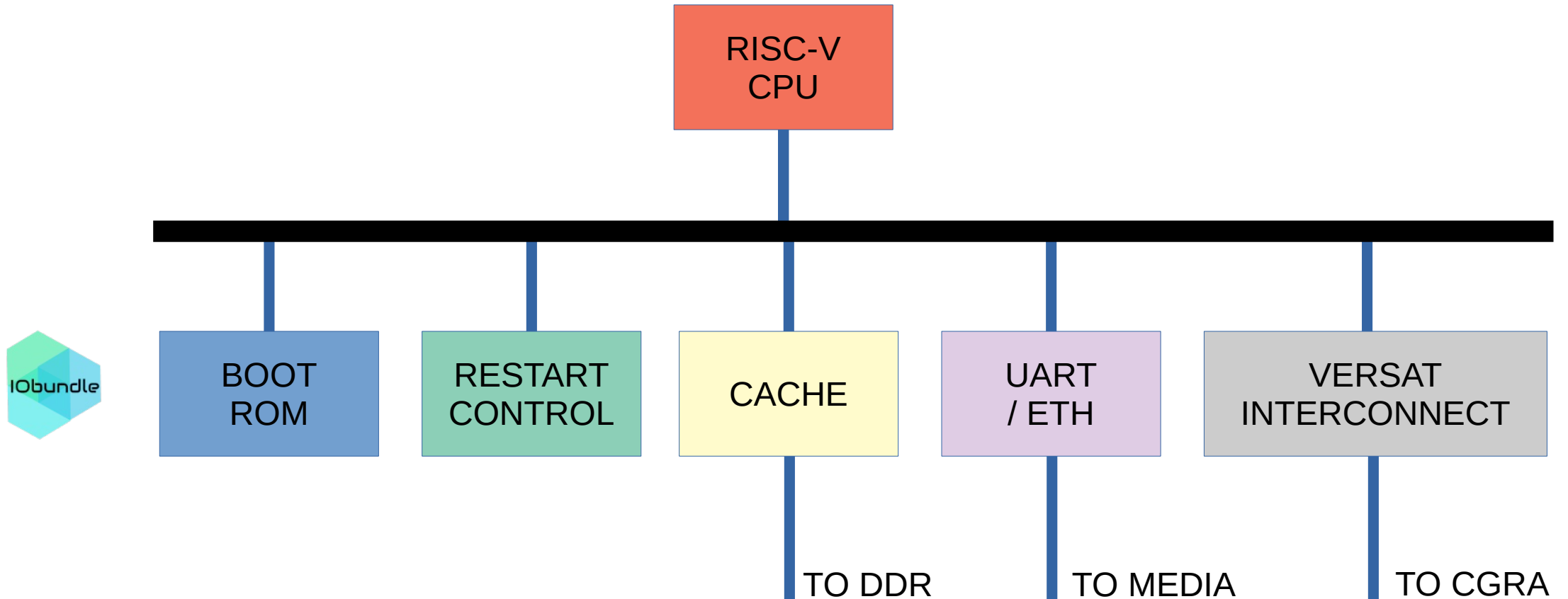
//Run with dynamic and partial reconfiguration (change start addresses only)
for (i=0; i<rowsA/2; i++) {
    v1.memPort[m0A].setStart(i*colsA+0);
    v2.memPort[m0A].setStart(i*colsA+5);

    //We get 22 results in each run
    v1.memPort[m2B].setStart(1856+colsBT*i);

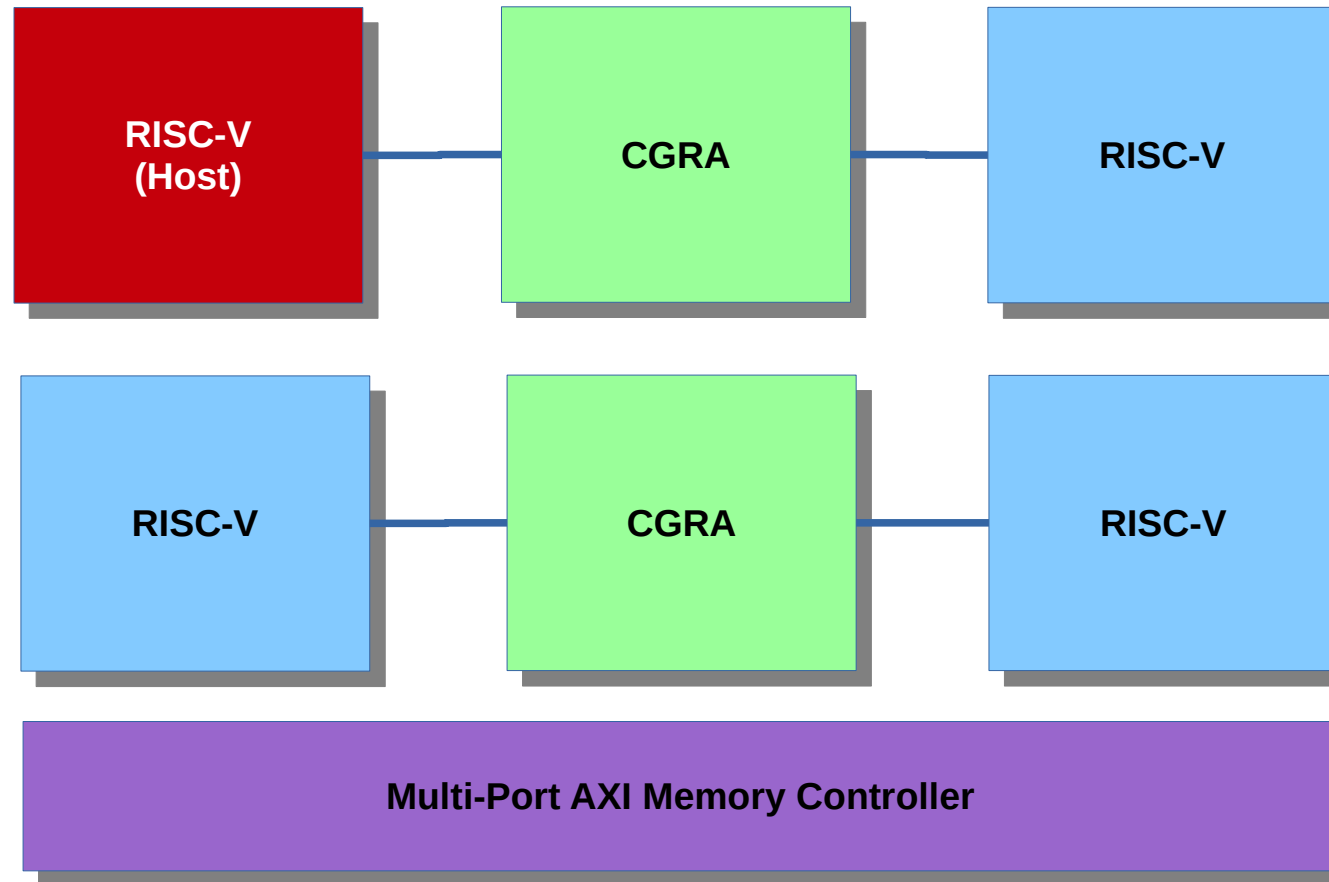
    versatRun();
}
}
```



RISC-V System (picoRV32, github)



The IOB-soc Architecture



Results for digital audio

Audio Format	#CPUs	#CGRAs
MPEG1/2 Layers I/II stereo encoder/decoder	1	0
AAC-LC 6-channel decoder	1	0
AAC-LC stereo encoder	1	0
AAC-LC 6-channel encoder	3	0
HE-AAC stereo encoder/decoder	1	1
HE-AAC 6-channel encoder/decoder	3	1
AC3 stereo encoder/decoder	1	1
6-channel encoder/decoder	3	1

Core (40nm)	Area (mm)	RAM (kB)	Freq. (MHz)	Power (mW)
ARM decoder	4.6	64	50	31.25
ARM Encoder	4.6	64	70	43.75
This Decoder	1.75	164	80	20.76
This Encoder	1.95	172	80	23.31

ARM = ARM Cortex A9 with NEON SIMD unit

This decoder = 3xAdept5 + 1xVersat4

This encoder = 3xAdept5 + 1xVersat2

Conclusion

- Open source reconfigurable computing is beneficial
- Evidence abounds from open source software
- Need to open source architecture and tools indissociably
- Full custom FPGA is difficult, consider synthesizable designs
- FPGA needs map, place, route and timing analysis tools!
- CGRA easier to open source compared to FPGA (simple place, route and timing)
- FPGA and CGRA need control CPUs: open source control RISC-V processors and toolchains
- The open source Versat CGRA (github): pre-silicon configurable, runtime partially reconfigurable
- IOB-soc: an open source SoC using RISC-V processors and Versat CGRAs