Development of an RV64GC IP core for the GRLIB IP Library

Johan Klockars
Cobham Gaisler
info@gaisler.com

Agenda

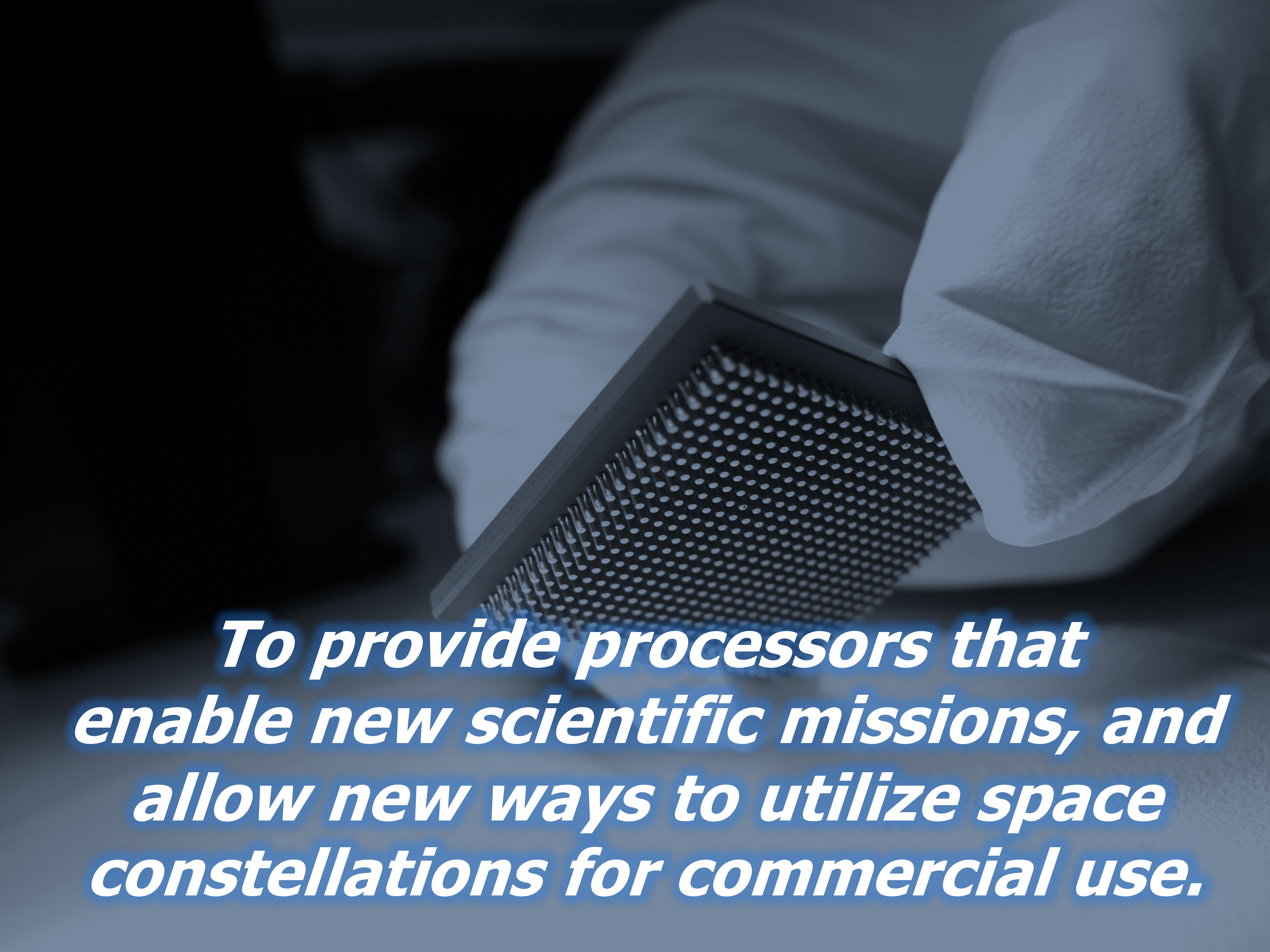
Background

Cobham Gaisler AB

Since 2 December 2014

- **Cobham Gaisler** is a world leader in processors for space applications like satellites & launchers
- Located in Gothenburg, Sweden
- Established in 2001 and acquired by Aeroflex in 2008
- Fully owned subsidiary of Cobham plc since 2014
- Management team with >100 years combined experience in the space sector:
- 34 employees with expertise within electronics, ASIC and software design
- Complete facilities in-house for ASIC and FPGA design
- Cobham has 15+ years experience designing open hardware

- RISC-V Foundation member 2019

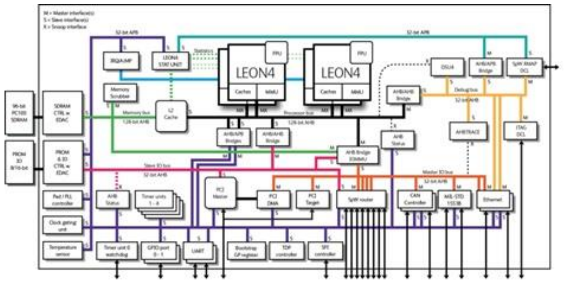


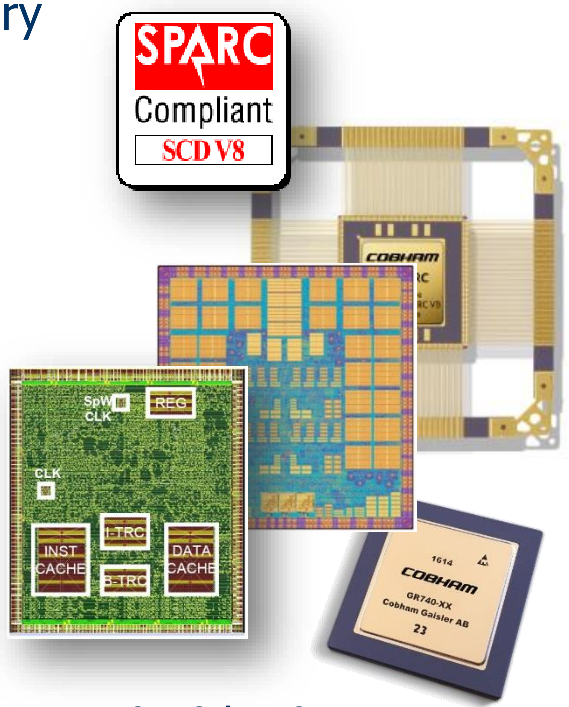
To provide processors that enable new scientific missions, and allow new ways to utilize space constellations for commercial use.

Cobham Gaisler Processor Solutions

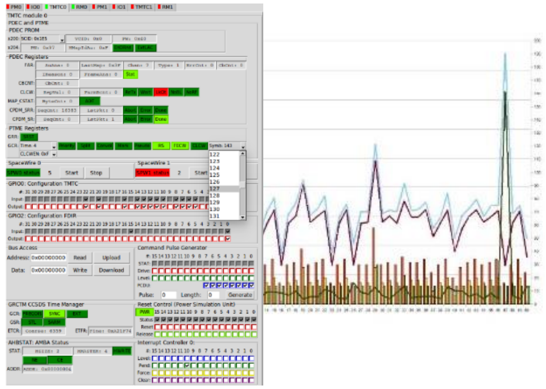
One-Stop-Shop

Synthesizable IP Core Library

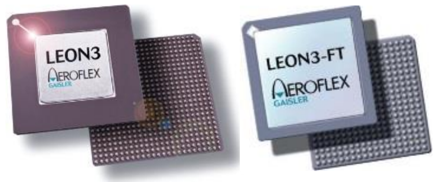




Simulators, Debuggers, Operating Systems, Compilers

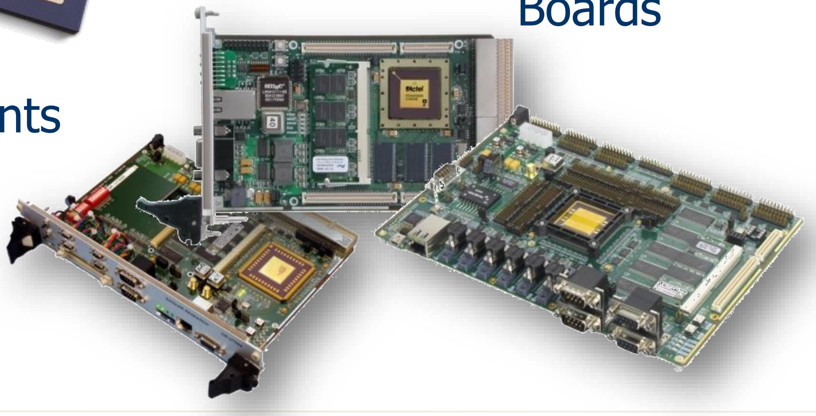


FT FPGA Processors



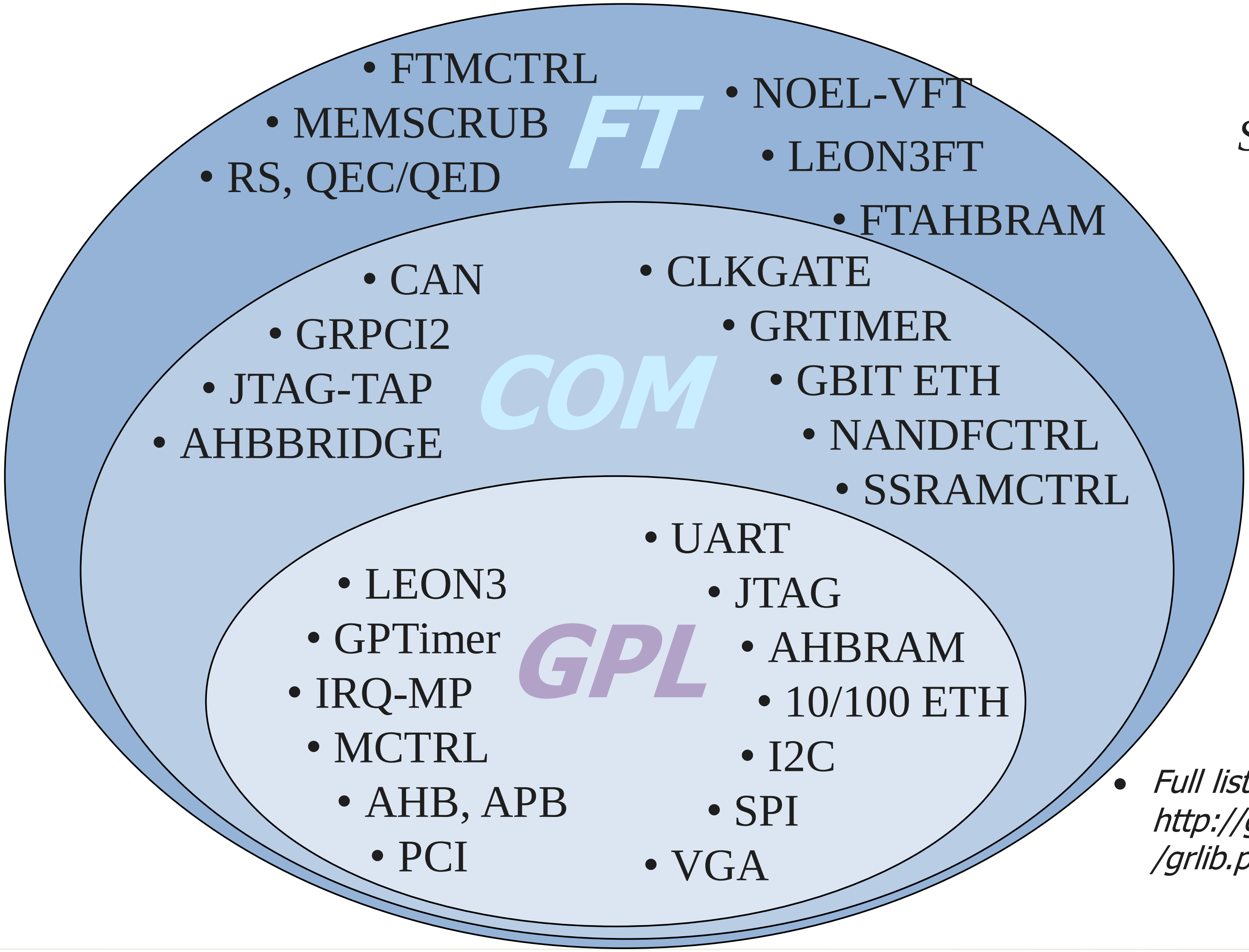
FT LEON3/LEON4 Processor Components

Development Boards



System Testbeds

GRLIB Distributions: GPL, Commercial, Fault-Tolerant

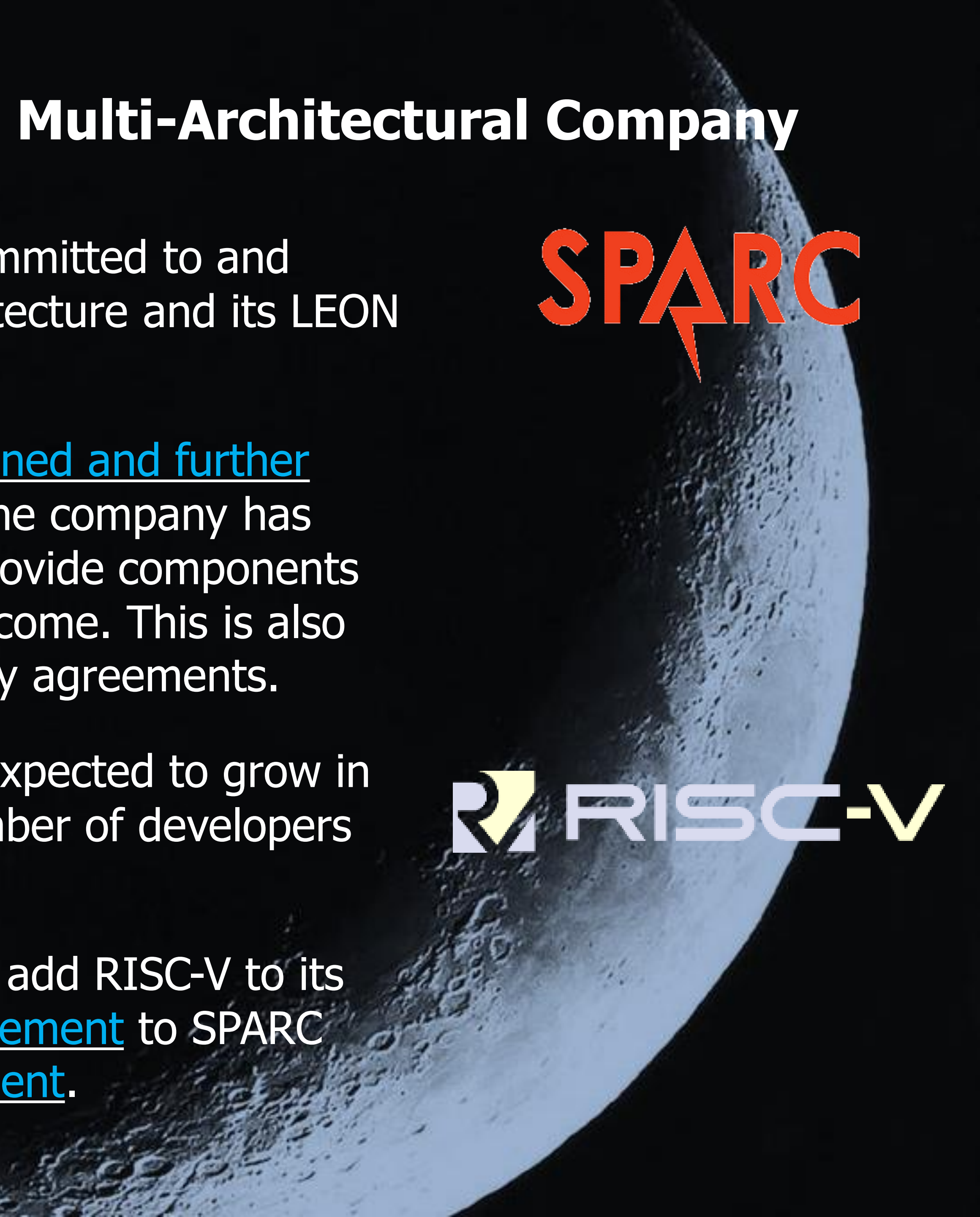


Separately licensed:

- FPU (-lite)
- SPW
- 1553
- USB
- IOMMU
- L2CACHE
- AES, ECC
- CAN-FD
- LEON4

- *Full list of cores available in http://gaisler.com/products/grlib/grlib.pdf*

Cobham is now a Multi-Architectural Company



Cobham continues to be committed to and invested in the SPARC architecture and its LEON implementations.

SPARC/LEON will be maintained and further developed going forward. The company has customers expecting it to provide components and support for decades to come. This is also ensured via long term supply agreements.

The RISC-V architecture is expected to grow in the future with a larger number of developers compared to SPARC V8.

Going forward, Cobham will add RISC-V to its product portfolio as a complement to SPARC and ARM, not as a replacement.

What is NOEL-V?

NOEL-V Processor Core

Primary goals:

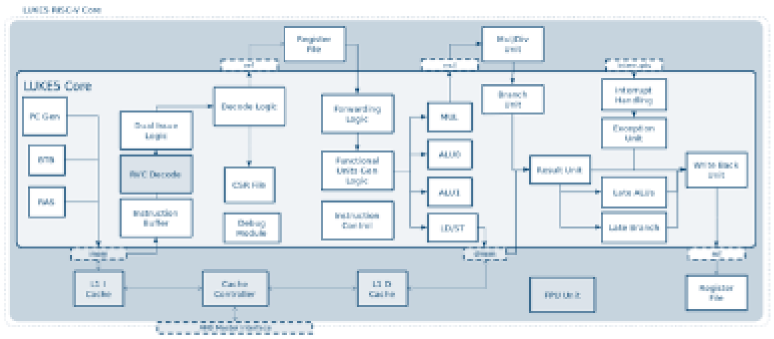
- RISC-V 64-bit compliant processor core
- Fault Tolerance - Error Correction Codes (ECC)
- Cybersecurity (proprietary solutions)
- Enable ISO 26262/FUSA certification (Road vehicles – Functional safety)
- Leverage foreseen uptake of RISC-V software and tool support in the commercial domain
- Compatible with GRLIB IP Core library

Target technologies:

- ASIC implementations for space applications
- High-end space FPGAs: Kintex Ultrascale

Target applications:

- General purpose payload processing
- Mixed platform and payload applications
- With future DDR4 SDRAM controller, specifically targeted for space applications



Cobham Gaisler GRLIB IP Library©

NOEL-V Features

COBHAM

- RISC-V RV64GCN

– M	mul/div	– A	atomics
– F	32 bit float	– D	64 bit float
– C	16 bit instructions	– N	user-level interrupts

- 7-stage dual-issue in-order
- Late ALUs and branch unit
- M/S/U with MMU
- SMP/AMP hardware coherency
- Dynamic branch prediction
- Blocking write-through L1
- PMP
- Hypervisor (pending standardization)
- RISC-V PLIC
- Multi-core
- RISC-V debug specification
- AMBA 2.0 AHB
  (subsystems with L2 cache and AXI4)

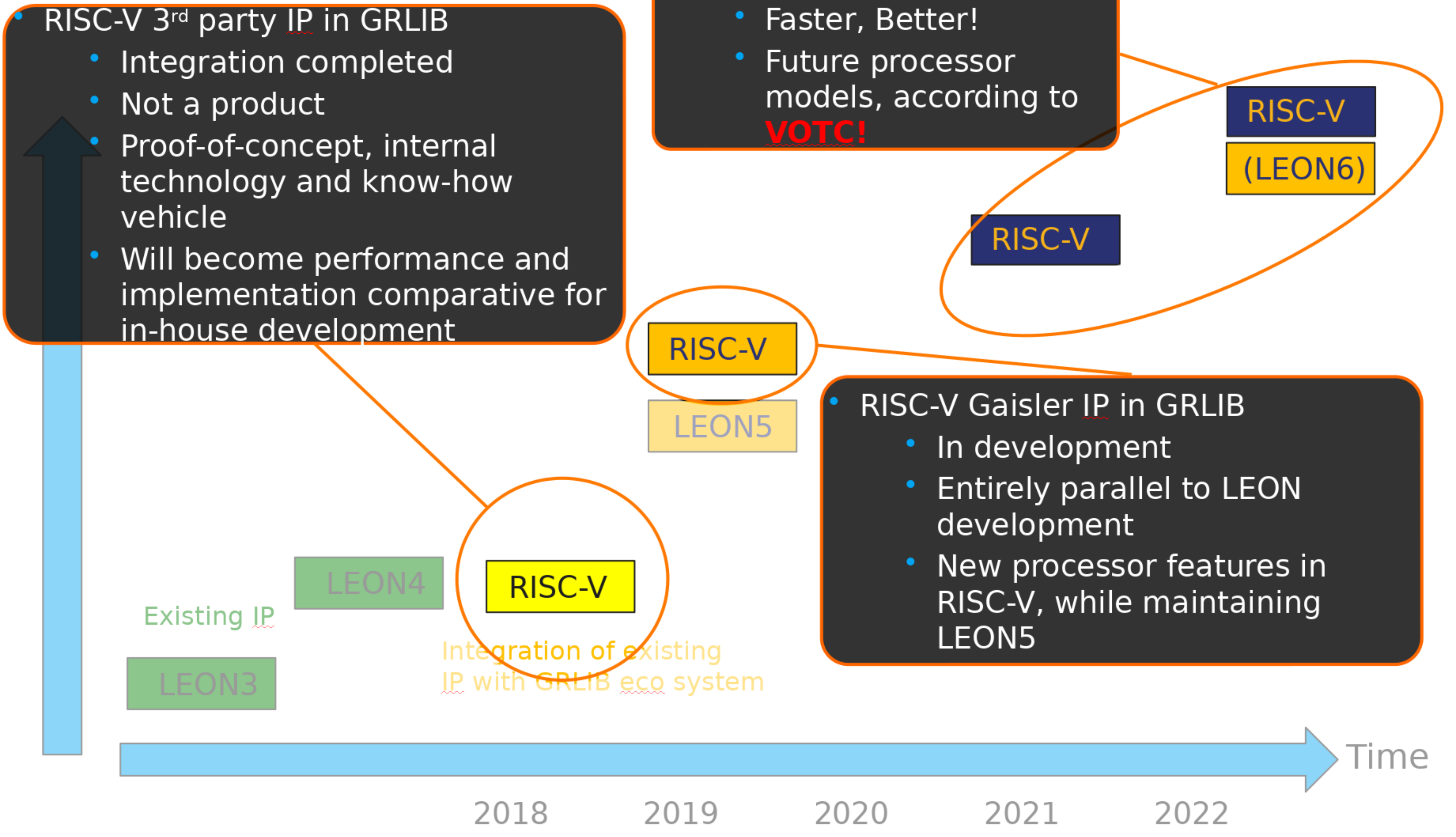
Fault tolerance

- For space, fault tolerance is necessary.
  - Various choices regarding ECC, parity etc.

- What to do when non-correctable RAM errors are detected?
- Requirement:
  - Deterministic and safe behavior.
- Wish:
  - Be able to log the fault, on best-effort-basis, to external storage.
- RISC-V leaves CPU response on HW fault to the implementation
  - no dedicated exception number assigned for bus access fault, IU register error, FPU register error, etc.
  - no semantic on CPU response to the above events
  - mtval may not be enough (SW-writable, nesting faults?)
- NOEL-V approach to be determined.

Combined processor roadmap

~~RISC-V roadmap~~



- RISC-V 3rd party IP in GRLIB
  - Integration completed
  - Not a product
  - Proof-of-concept, internal technology and know-how vehicle
  - Will become performance and implementation comparative for in-house development

- Future Gaisler IP
  - Faster, Better!
  - Future processor models, according to **VOTC!**

- RISC-V Gaisler IP in GRLIB
  - In development
  - Entirely parallel to LEON development
  - New processor features in RISC-V, while maintaining LEON5

NOEL-V Performance

- No work on compiler/libraries for NOEL-V yet.
- Testing on Kintex Ultrascale (KCU105) at 100 MHz.

- LEON5 has been measured at (Cobham gcc):
  - 3.14 DMIPS/MHz
    (-O3 and all files are combined during compilation)
  - 4.57 Coremark/MHz
    (-O3 -mcpu=leon5 -msoft-float -DPERFORMANCE_RUN=1 -funroll-all-loops -finline-functions -finline-limit=1000)

- Very preliminary, NOEL-V simulated (standard toolchain)
  - 4.36 CoreMark/MHz (ee_u32 as signed)

NOEL-V Relation to LEON5

- Related micro-architectures, but separate teams.
- NOEL-V development started later, so reuse.

- Partial reuse
  - Principles of the integer pipeline
  - Similar branch prediction

- Complete (more or less) reuse
  - FPU
  - Instruction and data cache
  - MMU and cache controller

NOEL-V Synthesis configurability

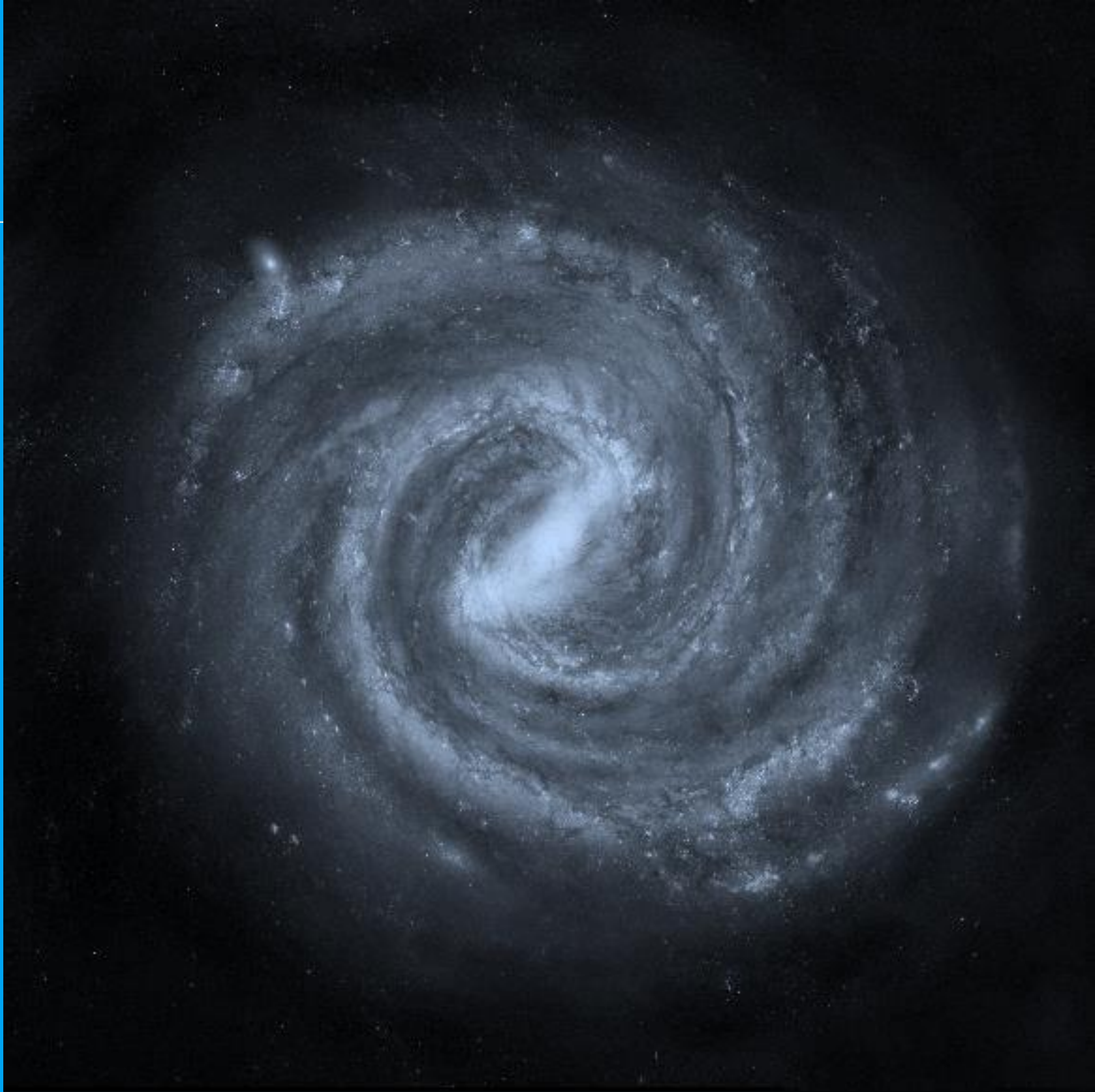
Planned

- MCADFN
- Virtual address space
- Late ALU
- Caches
- TLB
- Branch prediction
- U/S (MMU)
- Physical address space
- Late branch
- FPU
- PMP
- Single-issue

Possibilities for later

- 32 bit
- Non-standard instructions
- B / P / V
- ...

For who is NOEL-V?



Why another RISC-V implementation?

- Cobham is developing its own RISC-V implementation:
  - As opposed to licensing from 3rd party
- Full control of the design means short path to new custom features
  - I.e. not dependent on external IP

- Experienced processor team in-house
- GRLIB based implementation - existing infrastructure
- Allows for flexible license options
  - Flight
  - Commercial
  - Educational
  - Hobbyist

Licensing model

- Parts of GRLIB are under an open license
- The intention is to do the same with NOEL-V
  - GPL, CC, CERN OHW, solderpad,...
  - Any user can evaluate on FPGA development board
  - Academic use without complicated license setup
  - Hobbyists

- Fault-tolerant functionality in the flight license
  - Netlist, encrypted RTL

- NOEL-V will be distributed from Q1 2020

Development

NOEL-V Development

- Not Chisel, SpinalHDL, Lava, MyHDL, Migen,...
  - Few developers familiar with them
  - HW engineers often not computer scientists
  - No support from tool vendors
- Not HLS
  - Mostly as above
  - Questionable performance

NOEL-V Development

VHDL

- Really nice, when used "the right way"
- Very common, in Europe at least
  - We can find developers
  - Our users can understand
- Well established among customers in the space domain
- Good tool support
  - Including free simulator
  - Logical equivalence checkers
- GRLIB and LEON3-5

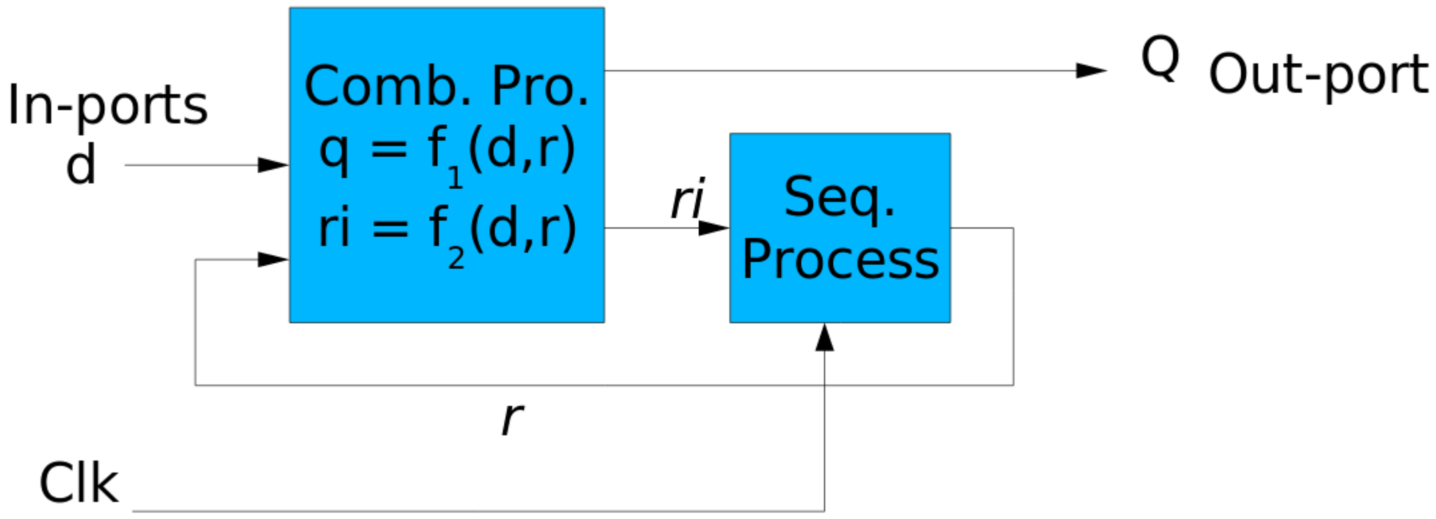
NOEL-V Development

- “Classic” VHDL, to maximize tool support
- Strive for code clarity, and rely on the tools!
  - Gaisler two-process implementations (www.gaisler.com/doc/structdesign.pdf)
    - Combinational
      with a few record output signals,
      one of which is total internal state
    - Clocked
      generally only registers the above
      internal state, and handles reset
  - Small number of processes
  - Few signals, mostly in/out/state records
  - Variables
  - Functions / procedures

NOEL-V Development

- From “A Structured VHDL Design Method” by Jiri Gaisler.

Two-process VHDL entity



NOEL-V Development

- Algorithms easily extracted
- Easy to extend

- Readability = Maintainability
- Fast simulation
- Easier debugging and verification
- No simulation/synthesis discrepancies

NOEL-V Development

- **Example:** Current NOEL-V integer pipeline
  - 2 processes
    - Combinational, 2200 lines
    - Clocked, 60 lines
    - 53/22 procedures/functions, ~5000 lines
      (not counting generic ones from other files)
  - 17 in port signals
  - 13 out port signals
  - 4 local signals (+12 for disassembler)
- The in/out ports connect to separate modules for:
  caches, register file, branch prediction, IRQ, debug, mul/div.

NOEL-V Development

- **Example:** Current NOEL-V cache controller and MMU
  - 3 processes
    - Combinational, 3500 lines
    - Two clocked, one assignment each (+debug)
    - 10/45 procedures/functions, ~1500 lines
      (not counting generic ones from other files)
  - 12 in port signals
  - 4 out port signals
  - 4 local signals (+2 for debug)
- The in/out ports connect to:
  AHB bus, caches, integer pipeline.
- Both LEON5 (Sparc) and NOEL-V (RISC-V)!

NOEL-V Development

- **Example:** First half of the execute stage.

```
ex_flush    := '0';
if wb_fence_i = '1' or v.wb.flushall = '1' or x_branch = '1' then
  ex_flush  := '1';
end if;

ex_branch_flush     := '0';
if wb_fence_i = '1' or v.wb.flushall = '1' then
  ex_branch_flush   := '1';
end if;

ex_forwarding(...);         -- Lane 0
ex_forwarding(...);         -- Lane 1

branch_unit(...);

jump_ex_forwarding(...);

jump_unit(...);

alu_execute(...);           -- ALU0
alu_execute(...);           -- ALU1

ex_stdata_forwarding(…);

mul_gen(…);

for i in 0 to ISSUEWAYS-1 loop
  ex_xc(i)                  := r.e.ctrl(i).xc;
  ex_xc_cause(i)            := r.e.ctrl(i).cause;
  ex_xc_tval(i)             := r.e.ctrl(i).tval;
end loop;
...
```

NOEL-V Development

- **Example:** Detail from the execute stage.

```
-- Forwarding Lane 1 ---------------------------------------------------
ex_forwarding(r,                                   -- in  : Registers
              1,                                   -- in  : Lane 1
              r.e.forw(1),                         -- in  : Forwarded from Memory
              ex_alu_op1(1),                       -- out : Output op1 from Mux
              ex_alu_op2(1)                        -- out : Output op2 from Mux
              );


-- Branch Unit ---------------------------------------------------------
branch_unit(ex_alu_op1(1),                         -- in  : Forwarded Op1
            ex_alu_op2(1),                         -- in  : Forwarded Op2
            r.e.ctrl(1).valid,                     -- in  : Enable/Valid Signal
            r.e.ctrl(1).branch.valid,              -- in  : Branch Valid Signal
            r.e.ctrl(1).inst(14 downto 12),        -- in  : Inst funct3
            r.e.ctrl(1).branch.addr,               -- in  : Branch Target Address
            r.e.ctrl(1).branch.naddr,              -- in  : Branch Next Address
            r.e.ctrl(1).branch.taken,              -- in  : Prediction
            r.e.ctrl(1).pc,                        -- in  : PC In
            ex_branch_valid,                       -- out : Branch Valid
            ex_branch_mis,                         -- out : Branch Outcome
            ex_branch_addr,                        -- out : Branch Address
            ex_branch_xc,                          -- out : Branch Exception
            ex_branch_cause,                       -- out : Exception Cause
            ex_branch_tval                         -- out : Exception Value
            );
```

NOEL-V Development

- **Example:** Extract from the MMU/cache controller.

```
entity mmu_cache5v2rv is
  generic (…);
  port (
    rst         : in  std_ulogic;
    clk         : in  std_ulogic;
    ici         : in  icache_in_type4;      -- I$ requests from iu5
    ico         : out icache_out_type4;     --    replies
    dci         : in  dcache_in_type4;      -- D$ requests from iu5
    dco         : out dcache_out_type4;     --    replies
    ahbi        : in  ahb_mst_in_type;      -- AHB replies
    ahbo        : out ahb_mst_out_type;     --    requests
    ahbsi       : in  ahb_slv_in_type;      -- AHB snoop address
    ahbso       : in  ahb_slv_out_vector;   -- AHB config data
    crami       : out cram_in_type4;        -- tags and data to cache
    cramo       : in  cram_out_type4;       -- tags and data from cache
    csr         : in  csrtype;              -- MMU and PMP configuration
    sclk        : in std_ulogic;            -- sclk for snoop (not gated)
    );
end;
…
  comb: process(r, rs, rst, ici, dci, ahbi, ahbsi, ahbso, cramo, csr)
…
  regs: process(clk)
…
  sregs: process(sclk)
…
```

Verification

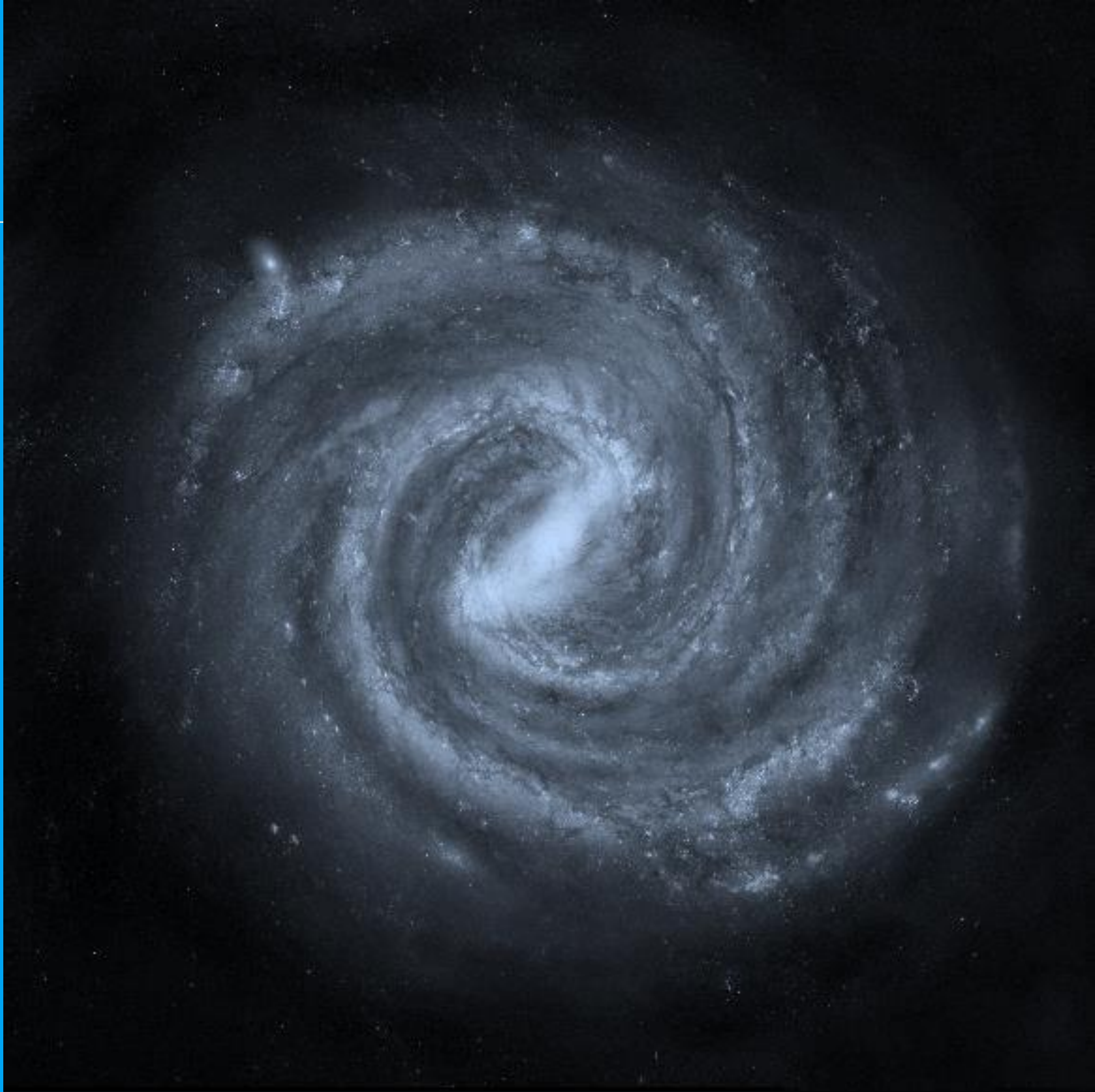
NOEL-V Verification – how?

- Internally developed SystemVerilog framework
  - Self-checking tests
  - Match against golden model (spike)
    - Instruction by instruction
      (some special handling, especially regarding time)
  - Regression tests script
- Mainly Modelsim
  - Mixed language
  - Snap-shot for faster simulation

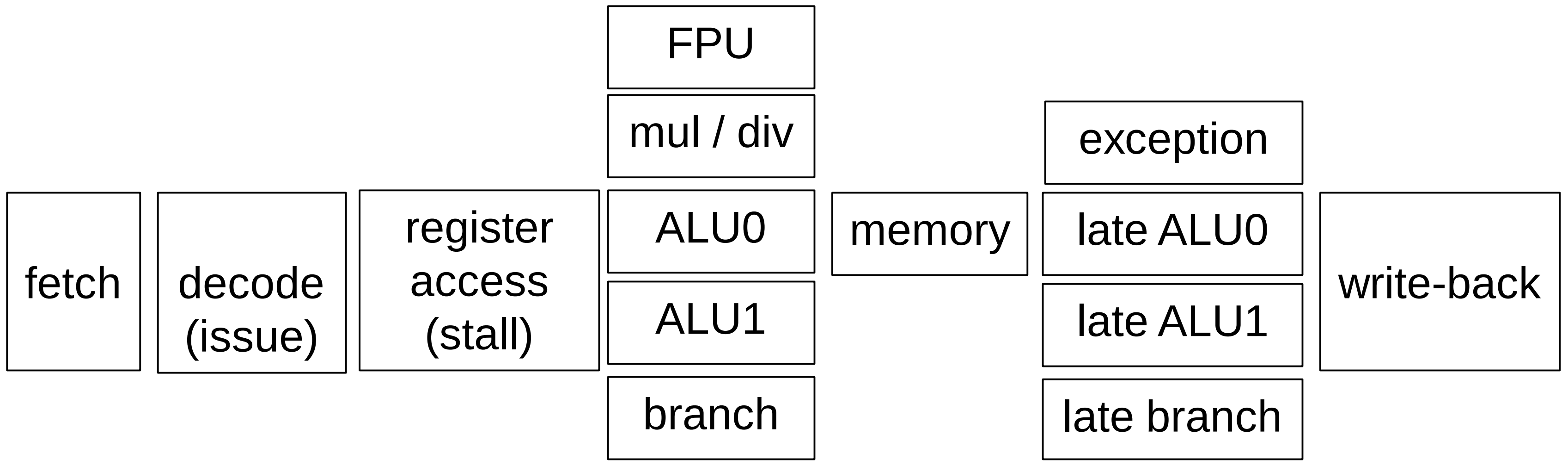
NOEL-V Verification – what?

- Publicly available test suites, such as
  - riscv-compliance
  - riscv-tests
  - riscv-dv
  - riscv-torture
- Internal random generator
- OS kernels
  - Zephyr
  - RISC-V Proxy Kernel
  - Rvirt
  - Linux
  - RTEMS
- Applications
- Coverage
- Targeted tests

Pipeline

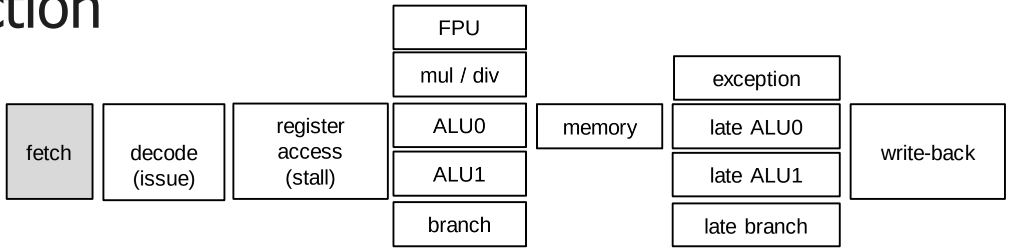


NOEL-V Integer pipeline



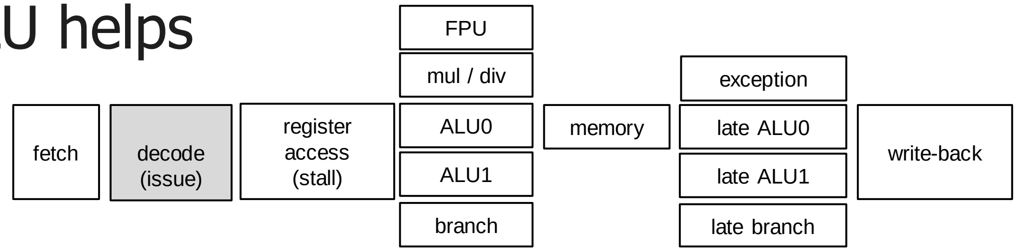
NOEL-V Fetch stage

- Configurable branch prediction
- Branch target buffer
- Branch history table
  - Bimodal
  - Two-level dynamic

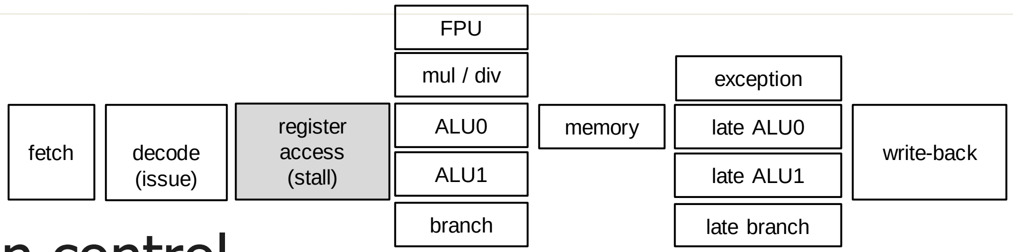


NOEL-V Decode stage

- Expands compressed instructions
- Checks for dual-issue conflicts
  - One unit: Memory, branch, mul/div, CSR
  - CSR write first
  - A few more, but late ALU helps
- Swap instructions if needed
  - Memory in 0
  - Branch in 1
- Check for illegal or privileged instruction
- Check for RAS hit and early branch



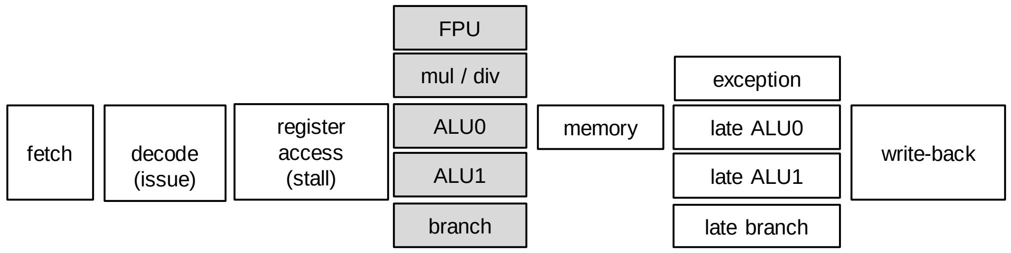
NOEL-V Register access stage



- Read register file and CSR
  - RF is 4R/2W
- Generate ALU and instruction control
- Decide on early/late ALU/BU
- Pipeline bubbles only here
  - Dependence on late ALU
  - Non-commited CSR write to read CSR
  - Memory access following MMU/PMP CSR write
  - ...

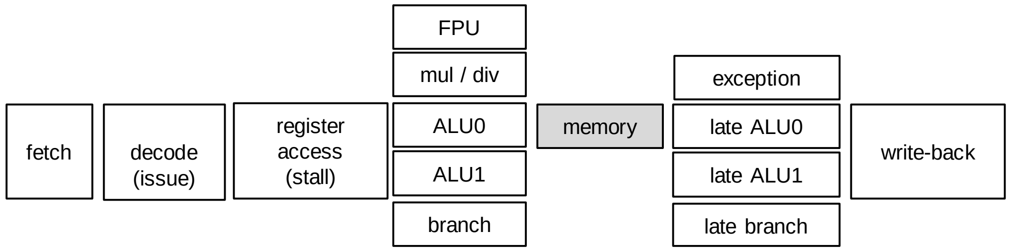
NOEL-V Execution stage

- Two equal ALUs
- Branch unit
- Combinational virtual address to data cache interface
- FPU and mul/div start here



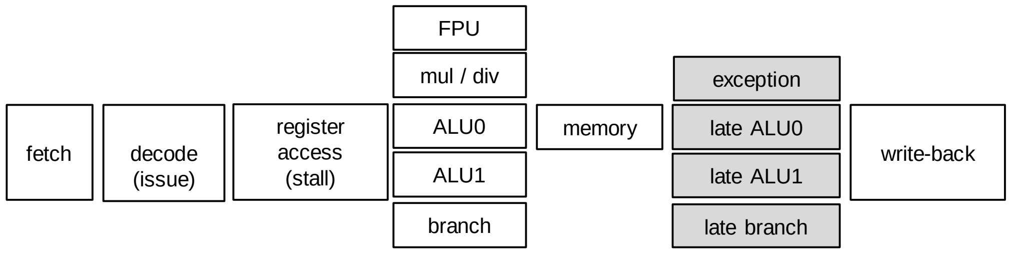
NOEL-V Memory stage

- Align data from cache
- Check TLB
- Check tags
  - Virtually indexed, physically tagged
- L1 write-through, blocking
  - Separate instruction and data caches
  - Up to 4-way associative, LRU
- FSM to deal with cache/TLB miss, store buffer full
  - Hardware page table walk
- Snooping for coherence
- Full PMP support, configurable



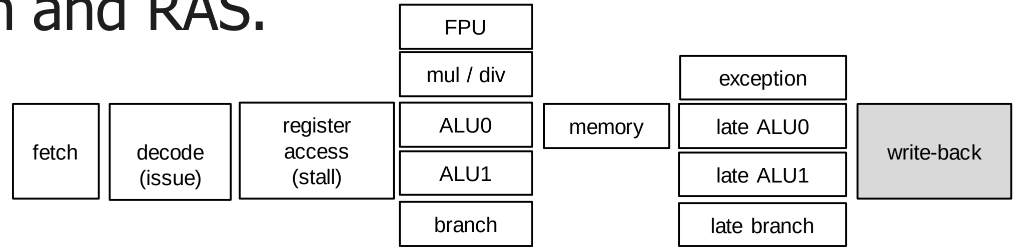
NOEL-V Exception stage

- Two more full ALUs
- Another branch unit
- Write to CSR
- Collect exceptions
- External interrupts



NOEL-V Write-back stage

- Write to register file
- Update branch prediction and RAS.



NOEL-V Current pipeline in use

```
uintptr_t a[LENGTH], b[LENGTH], c[LENGTH];
...
for(int i = 0; i < LENGTH; i++) {
  a[i] = b[i] + c[i];
}

loop:
  ld     a1, 0(a2)
  addi   a2, a2, 8
  ld     a4, 0(a5)
  addi   a3, a3, 8
  addi   a5, a5, 8
  add    a4, a4, a1
  sd     a4, -8(a3)
  bne    a5, a0, loop
```

NOEL-V Current pipeline in use

Pairing when first instruction is not 8-byte aligned.

```
loop:
  ld      a1, 0(a2)

  addi    a2, a2, 8         swapped
  ld      a4, 0(a5)           since this must be in lane 0

  addi    a3, a3, 8
  addi    a5, a5, 8

  add     a4, a4, a1        swapped
  sd      a4, -8(a3)          since this must be in lane 0

  bne     a5, a0, loop
```

8 instructions in 5 cycles!

NOEL-V Current pipeline in use

Pairing when first instruction is 8-byte aligned.

```
loop:
  ld      a1, 0(a2)
  addi    a2, a2, 8

  ld      a4, 0(a5)
  addi    a3, a3, 8

  addi    a5, a5, 8
  add     a4, a4, a1        late ALU

  sd      a4, -8(a3)        wait...
  bne     a5, a0, loop
```

8 instructions in 7 cycles.

NOEL-V Current pipeline in use

Different code generation, also 8-byte aligned.

```
loop:
  ld      a4, 0(a5)          swapped, paired with branch at end

  ld      a1, 0(a2)
  addi    a5, a5, 8

  addi    a2, a2, 8
  add     a4, a4, a1         late ALU

  sd      a4, 0(a3)          wait...
  addi    a3, a3, 8

  bne     a5, a0, loop
  ld      a4, 0(a5)
```

Again, 8 instructions in 7 cycles.

Thanks for listening!

Shameless plug:

Looking for talent!

https://www.gaisler.com/career

?