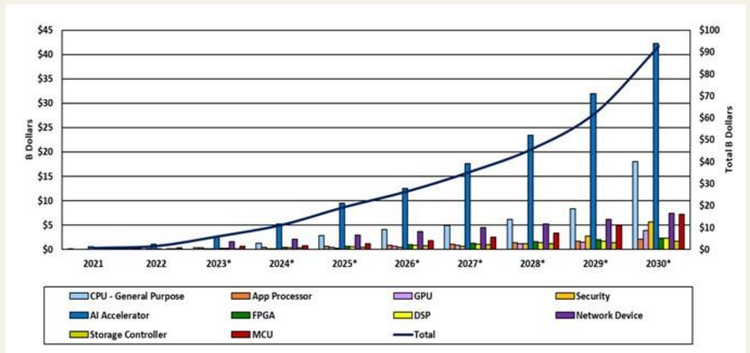
uFC – Functional Coverage on RISC-V microcode

Ajeetha Kumari Venkatesan
AsFigo Technologies

1 Introduction

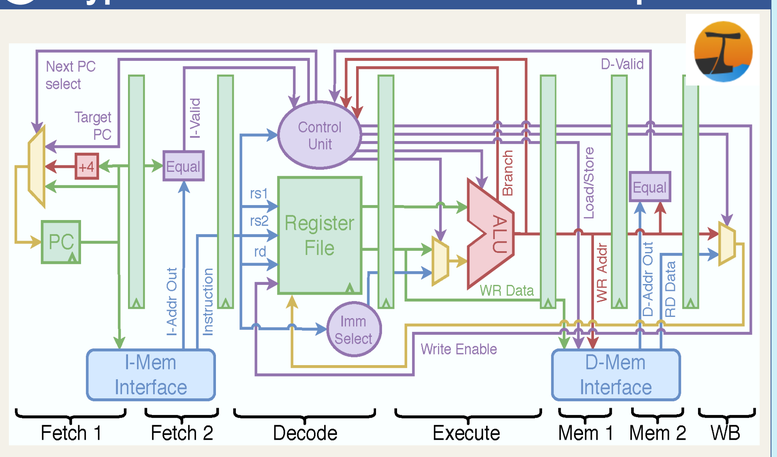
RISC-V – a name that sparks innovation all across the globe! Recent market survey indicates RISC-V industry to touch $92B by 2030. We at AsFigo are driven by opensource and help our customer adopt the same in chip design. In this work, we share our experience of building custom flows based on opensource ecosystem to build reliable, secure systems using RISC-V.



Source: SHD Group

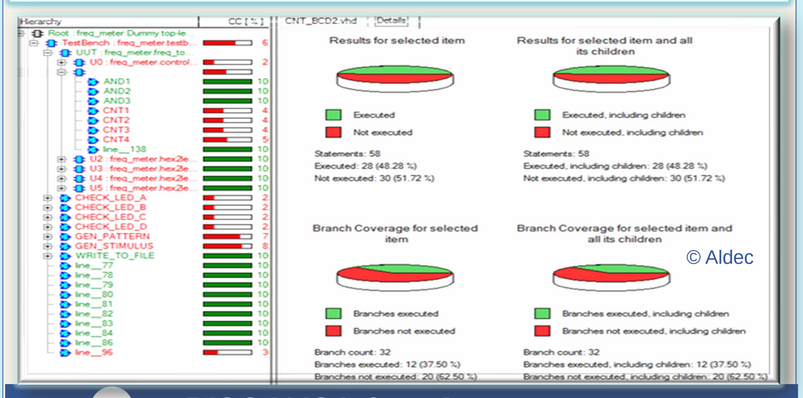
Specifically, we have built flows around uCode to measure quality of designs that save customers weeks in their chip design schedules. It build functional coverage metrics quickly from uCode and aptly named as uFC.

2 Typical RISC-V uArch – Trireme example



Trireme RISC-V Design Platform

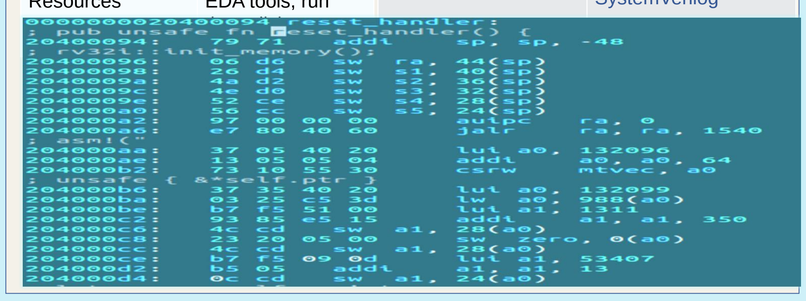
Extension	Description
I	Integer
M	Integer Multiplication and Division
A	Atomics
F	Single-Precision Floating Point
D	Double-Precision Floating Point
G	General Purpose = IMAFD
C	16-bit Compressed Instructions
Non-Standard User-Level Extensions	
Xext	Non-standard extension "ext"



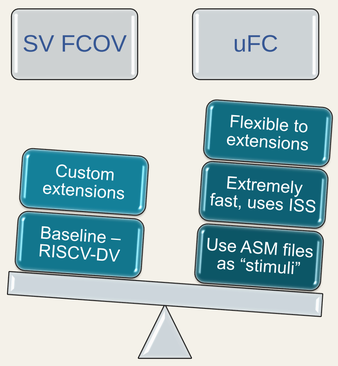
3 RISC-V ISA & code coverage

Code Coverage	
Importance	Necessary but not sufficient
Ease of use	Fully automated
Waivers	Manual/semi-automated exclusion flows
Block-by-block view	An instruction "fetched" may not be "executed"
Combinatorial	Lacks temporal coverage
Resources	EDA tools, run

uFC – uCode Functional Coverage	
Importance	Necessary but not sufficient
Ease of use	Semi automated
Flexible views	end-to-end
Data centric	Data dependency across stages
Combinatorial , Sequential	Choose appropriate sampling
Resources	Python, ISS, SystemVerilog



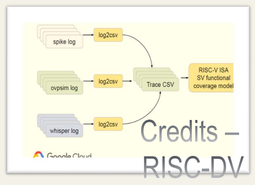
4 Mining microcode for metrics



- all opcodes
- all instruction operands
- positive/negative immediate values
- exception and interrupt handling
- corner cases (overflow, underflow, DIV-by-0)
- aligned/unaligned load/store operations
- forward/backward branches
- branch hit history
- Illegal, hint instructions
- privileged CSRs
- hazard conditions
  - RD port conflict

5 Integrating uFC into simulation Framework

- uFC
- Python script to generate Trace-CSV
- Supports both UVM and non-UVM TBs
- Wider applications – Emulation, FPGA, Post-Si etc.



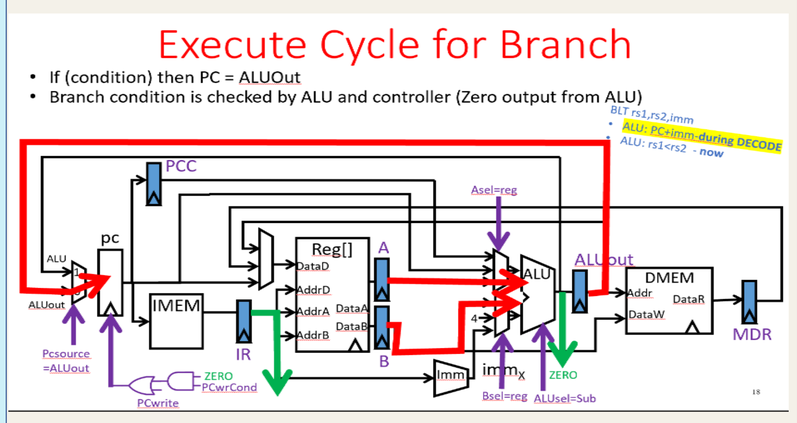
- Pick intelligent tests
- Optimize regression suites

Applications

- Machine Learning to improve test quality
- Floating Point support

Roadmap

6 uFC- Branch instructions




```
    beqz rs2, clear
    mv   rd, rs1
    j    next
clear:
    mv   rd, zero
next:
```

branch

```
sequence af_rv_no_br_seq;
   (imem_rd && imem_addr == CUR_PC) ##1
   (imem_rd && imem_addr == CUR_PC + 4) [->1];
endsequence : af_rv_no_br_seq

sequence af_rv_br_seq;
   (imem_rd && imem_addr == CUR_PC) ##1
   (imem_rd && imem_addr == CUR_PC + (cur_imm * 4)) [->1];
endsequence : af_rv_br_seq
```

7 Results

We ran this flow at single core level and a subsystem of 8 cores. We generated 412 coverpoints in a single IP setup and with existing regression of 1200+ tests (each running with 10+ random seeds) we achieved 76% functional coverage at first. We then added specific tests to focus on and ended adding 20 focused tests (constrained random) and ran them with 50 seeds each and got 99.2% coverage.

We have also deployed similar uFV flow on non-RSIC-V processors and hence are confident that this flow suits various classes of designs.

Configuration	Coverpoints	Tests	Seeds per Test	Initial Coverage	Focused Tests	Seeds per Focused Test	Final Coverage
Single Core	412	1200+	50	76%	20	100	99.2%
Subsystem (8 cores)	1800	280+	10+	48%	40	50	72.8%

8 Conclusion

In summary, our paper contributes to the growing body of research on RISC-V verification methodologies by introducing a novel approach to address the unique challenges associated with verifying microcode. We believe that our custom flow development for microcode functional coverage will serve as a valuable resource for *RISC-V Summit Europe*

9 References

[1] RISC-V, The RISC-V Instruction Set Manual Volume I: User-Level ISA.

[2] IEEE, Standard for SystemVerilog Unified Hardware Design, Specification, and Verification Language (IEEE Std 1800™-2023)

[3] IEEE UVM 1800.2

[4] https://pypi.org/project/riscv-assembler

Acknowledgements

Would like to acknowledge the contributions by my colleagues

- Deepa Palaniappan
- Hemamalini Sundaram

Contact Information

Ajeetha Kumari Venkatesan
Director of Verification
AsFigo Technologies
+44 7308 024 192
ajeethak@asfigo.com
www.asfigo.com