CVA6

Linux bootable 6-stage RISC-V

What, why, who, where?

- 6-stage, single-issue, in-order CPU core
- Implements 64-bit RISC-V instruction set with I, M, A and C extensions
- Also implements M, S, U privilege levels to support Unix-like OS (=able to boot Linux)
- Main design goal was to reduce critical path
- Open-source alternative to cores like ARM
- By OpenHW Group, non-profit global organization, HQ in Ottawa, Canada

HOW?

- **Languages:**
  - About 55% written in Assembly and about 25% written in SystemVerilog
  - Includes also Python scripts
- **Tools used / supported:**
  - Linux, GCC, core-V-verif (verification), riscv-isa-manual (instruction set)
- **License:**
  - Apache-2.0 license
    - You are allowed to:
      - Use the code
      - Modify it
      - Redistribute
      - Patent
    - Restrictions:
      - You need to mark any modifications
      - You must include Apache-2.0 license
      - You must include copyrights, patents, trademarks etc. from source work

Summary from Cloc-tool:

```
github.com/AlDanial/cloc v 1.98  T=4.61 s (310.3 files/s, 136673.7 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Assembly                       141          60231          78966         108046
YAML                           177           2810           1813          95569
Verilog-SystemVerilog          464          13274          21771          86307
Markdown                        71           9833             62          30796
AsciiDoc                        87           3233            218          14284
reStructuredText                88           6387          12567          10516
XML                              6             33              4          10288
Python                          55           1746           2073           9534
SVG                             37             11              2           9483
Tcl/Tk                          38            314            419           7742
C/C++ Header                    40            501            993           6666
C                               29            921           1088           4981
Bourne Shell                    80            694           1072           2328
make                            48            515            481           2091
Text                            11            219              0           1882
C++                              9            224            131           1383
diff                            13             66            519            849
Linker Script                    7             80            244            826
Stata                           10             27            354            629
CSV                              5              0              0            433
VHDL                             1             57            108            234
CSS                              3             54             54            222
HTML                             3              1              1             51
Mako                             1              2              0             51
DOS Batch                        1              8              1             26
Fortran 77                       1              2              0             23
Bourne Again Shell               1              7              3             21
INI                              1              1              0             12
Constraint Grammar               1              0              0              5
-------------------------------------------------------------------------------
SUM:                          1429         101251         122944         405278
-------------------------------------------------------------------------------
```

How easy is it to get started?

Are there instructions for "Hello world", are they clear?

- Yes there are, I tried… (2-3h)
- Steps to get it running:
  - Clone git
  - Install needed programs
  - Build the toolchain
  - Set environment variables
  - Run the simulation
- Given linux commands for every step
- Ready made installation scripts
- Possibility to use different simulators

```c
#include <stdint.h>
#include <stdio.h>

int main(int argc, char* arg[]) {

    printf("%d: Hello World !", 0);

    int a = 0;
    for (int i = 0; i < 5; i++)
    {
        a += i;
    }
    return 0;
}
```

```
git clone https://github.com/openhwgroup/cva6.git
sudo apt-get install autoconf automake autotools-dev curl git libmpc-dev libmpfr-dev
bash build-toolchain.sh $INSTALL_DIR
export DV_SIMULATORS=veri-testharness
```

```
python3 cva6.py --target cv32a60x --iss=$DV_SIMULATORS --iss_yaml=cva6.yaml \
--c_tests ../tests/custom/hello_world/hello_world.c \
--linker=../../config/gen_from_riscv_config/linker/link.ld \
--gcc_opts="-static -mcmodel=medany -fvisibility=hidden -nostdlib \
-nostartfiles -g ../tests/custom/common/syscalls.c \
../tests/custom/common/crt.S -lgcc \
-I../tests/custom/env -I../tests/custom/common"
```

- Instructions for ASIC and FPGA emulation also
- Problems with pip and naming of some scripts did not match instructions

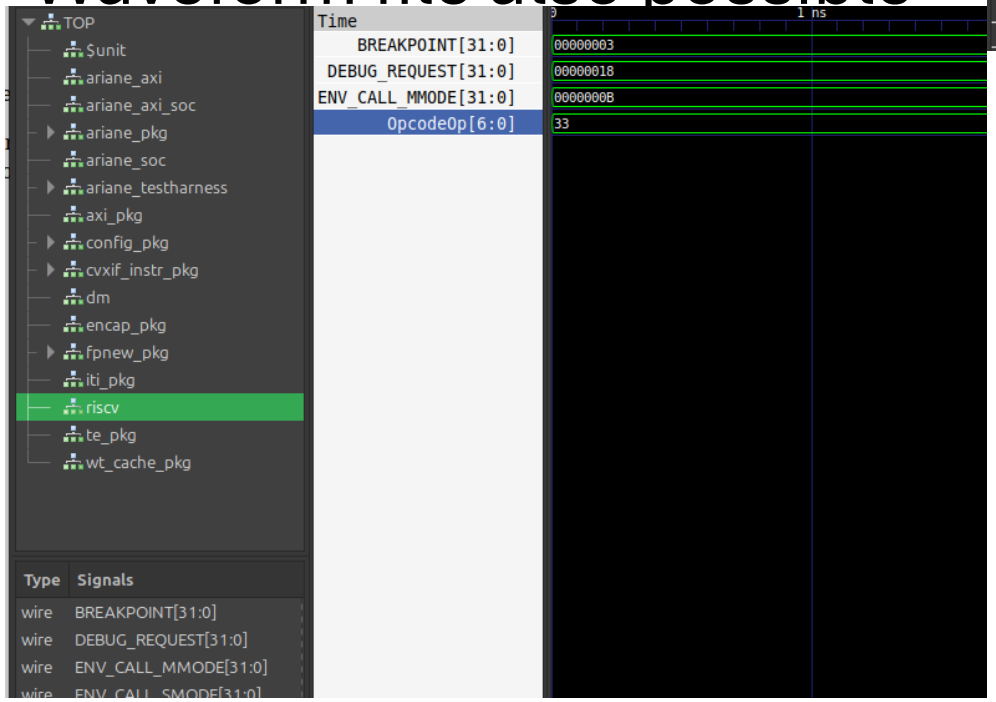
Results from simulation

- Log files
  - Assembly instructions
  - Terminal print
- Waveform file also possible

```
/home/topi/Koulu/cva6/verif/sim/out_2026-01-26/directed_tests/hello_world.o *** SUCCESS *** (tohost = 0) after 3334 cycles
*** [rvfi_tracer] INFO: Simulation terminated after       3322 cycles!

CPU time used: 7938.25 ms
Wall clock time passed: 7940.53 ms
```

	pc	instr	gpr	csr	binary	mode	instr_str	operand	pad
1	0000000080000000		ra:00000000		00004081	3	c.li ra, 0		
2	0000000080000002		sp:00000000		00004101	3	c.li sp, 0		
3	0000000080000004		gp:00000000		00004181	3	c.li gp, 0		
4	0000000080000006		tp:00000000		00004201	3	c.li tp, 0		
5	0000000080000008		t0:00000000		00004281	3	c.li t0, 0		
6	000000008000000a		t1:00000000		00004301	3	c.li t1, 0		
7	000000008000000c		t2:00000000		00004381	3	c.li t2, 0		
8	000000008000000e		s0:00000000		00004401	3	c.li s0, 0		
9	0000000080000010		s1:00000000		00004481	3	c.li s1, 0		
10	0000000080000012		a0:00000000		00004501	3	c.li a0, 0		
11	0000000080000014		a1:00000000		00004581	3	c.li a1, 0		



Did you understand the content in this time?

- Some parts from running the simulation
- Repository is large and there are a lot of content