Taking RISC-V® Mainstream

结合 Andes ACE 框架与 AndesCycle 加速 RISC-V 自订指令开发

顏敬哲, 陳枝懋, 吳奕緯

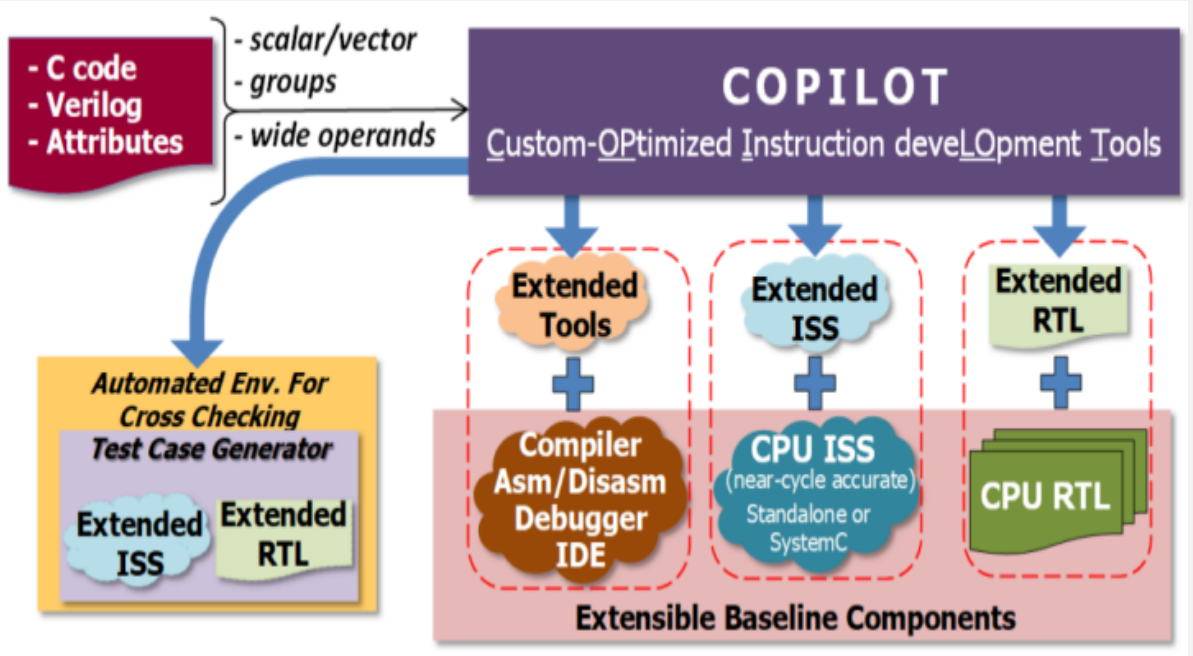
晶心科技 Andes Technology

Outline

- ACE Framework Overview
- AndesCycle Simulator
- Case Studies :
  - Video Codec Extension
  - Sigmoid Function Acceleration

Building Custom RISC-V Extensions with ACE

- Andes Custom Extension (**ACE**) framework
  - Enable designers to create custom CPU instructions on AndesCore™ processors.
- Key features
  - Support both custom scalar (ACE-Scalar) and vector (ACE-RVV) instructions
  - Provide a straightforward design flow
    - Describe instruction behavior in an ACE definition file
    - Implement the hardware logic in a concise Verilog file
  - Custom-OPtimized Instruction deveLOpment Tools (**COPILOT**)
    - Generate extended source code for compiler, debugger and instruction set simulator (ISS)
    - Produce Verilog code for ACE engine RTL integration



An Example of Designing Custom Instructions (1/2)

- **ACE definition** file includes
  - Instruction name, operands
  - C model block
  - Execution latency
- **Concise Verilog** file includes
  - Core logics
  - NOTE: Can be combinatorial or sequential

.ace

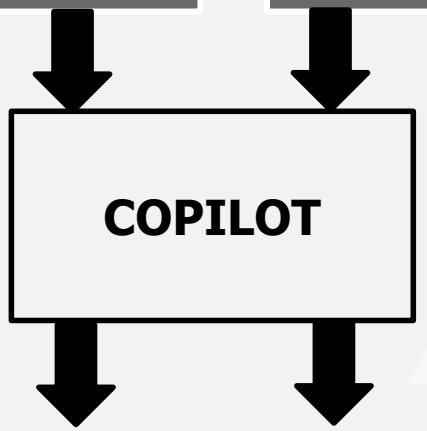
```
insn myadd {
  op = {out xrf dout, in xrf din, imm5 din2};
  csim = %{
    dout = din + din2; // C Model
  %};
  latency = 1;
};
```

.v

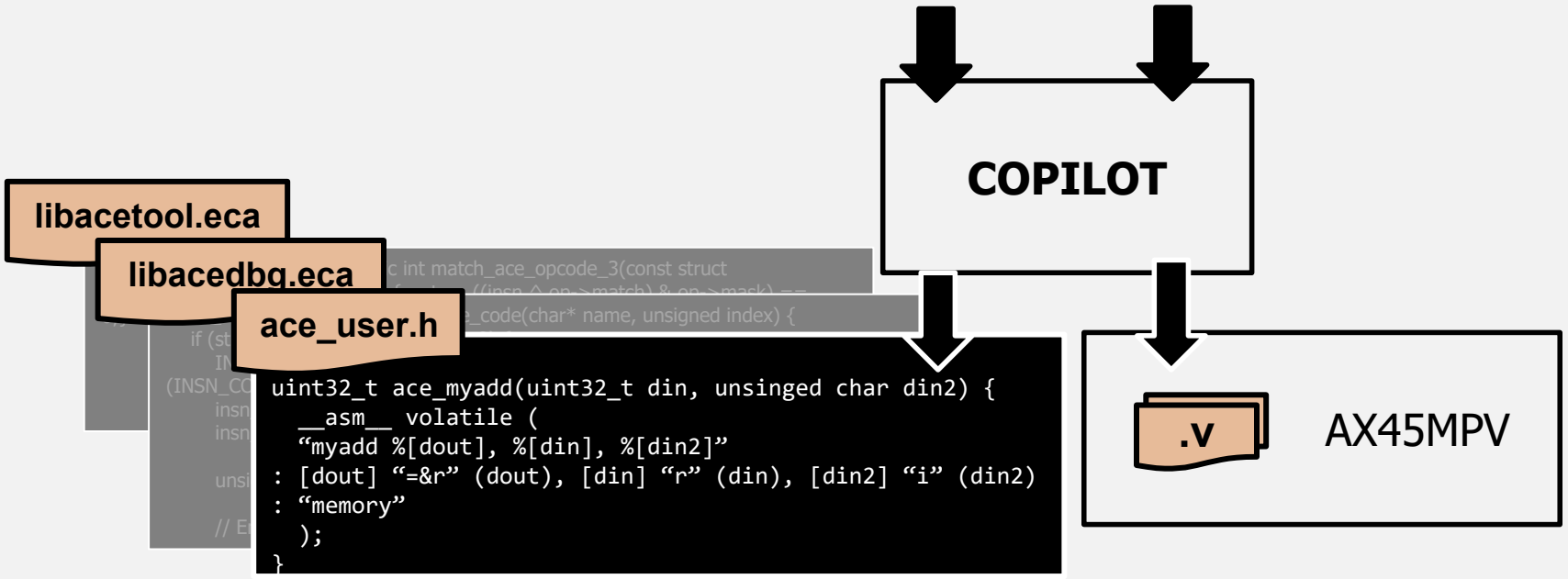
```
// ACE_BEGIN: myadd

assign  dout = din + {27'd0, din2};

// ACE_END
```



An Example of Designing Custom Instructions (2/2)



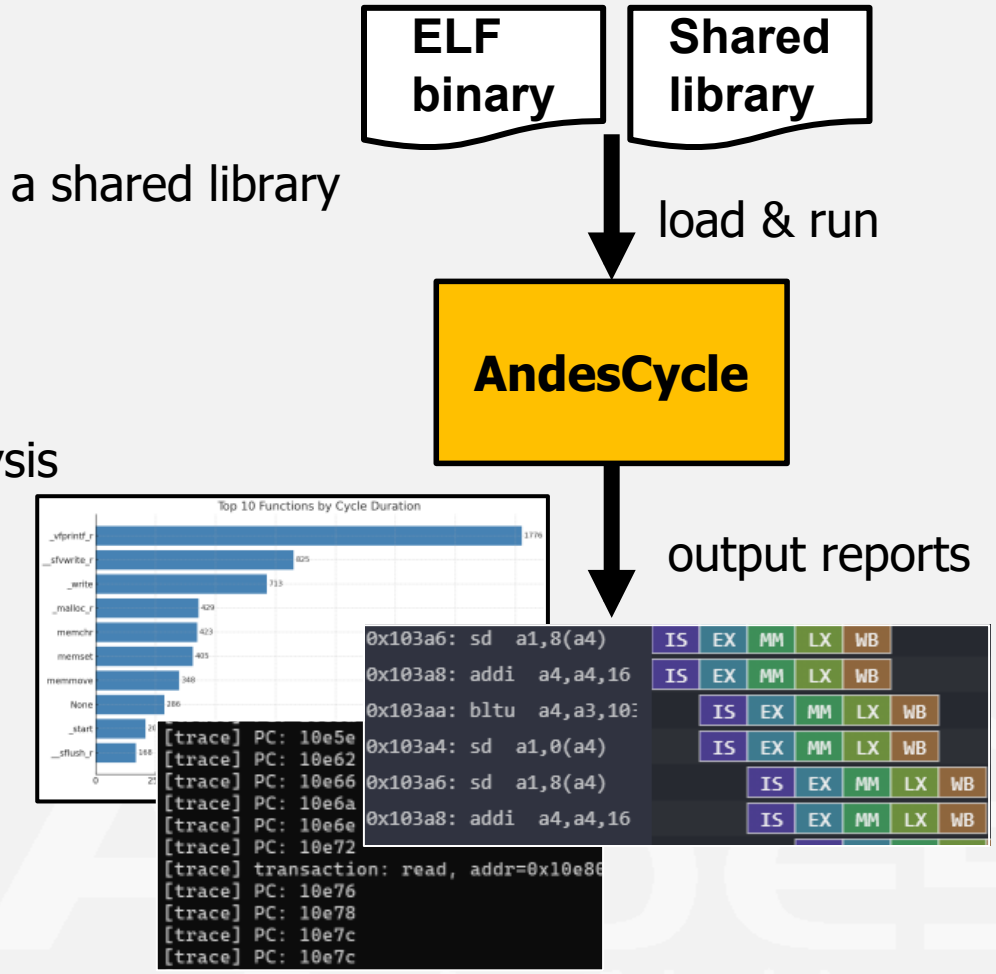
- COPILOT generates extended files :
  - ace_user.h : helper functions for C programming
  - libacetool.eca : for assembler
  - libacedbg.eca : for OpenOCD

- COPILOT generates **Verilog code** for control logics (ACE engine) and instruction modules
- Designers integrate these Verilog code into an existing AndesCore

AndesCycle

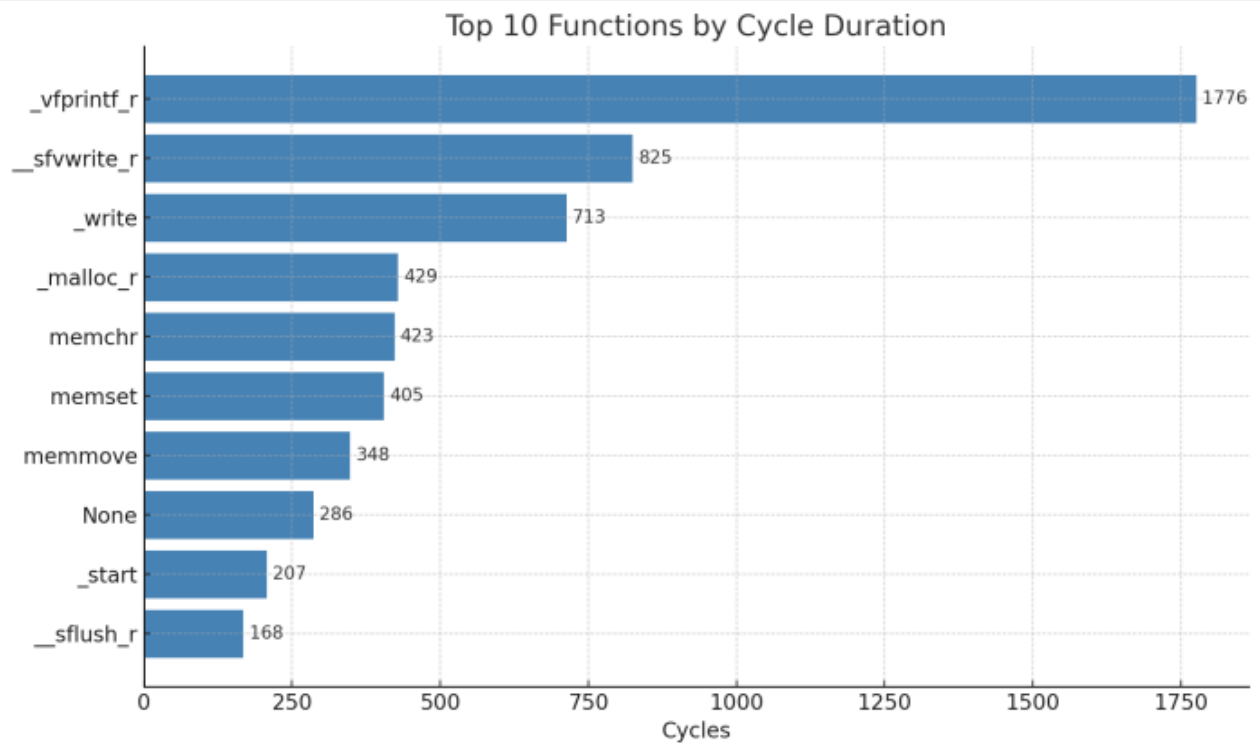
- AndesCycle is a cycle-accurate simulator generated from CPU RTL using Verilator

- Key features
  - **ACE support**
    - Support both ACE-Scalar and ACE-RVV instructions via a shared library
  - **Performance analyzer**
    - Perform hotspot function analysis
    - Create pipeline view for detailed instruction-level analysis
  - **Debug tools**
    - Support OpenOCD, GDB
    - Dump instruction and address trace



Feature : Hotspot Function Analysis

- The chart highlights potential target functions for optimization
- Program to read hpmcounters (CSR) to get precise statistics
  - IPC, branch miss, cache miss, hotspot analysis, etc.

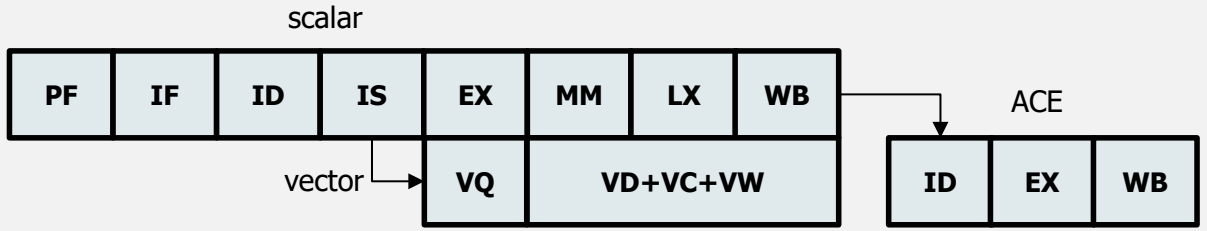


Name	Value
Number of cycles	6879
Number of instructions	3577
IPC	0.52
I$ miss rate	0.051
D$ miss rate	0.062
I$ miss waiting cycles	151
D$ miss waiting cycles	1523
Conditional branch miss rate	0.31

Feature : Pipeline View (AX45MPV)

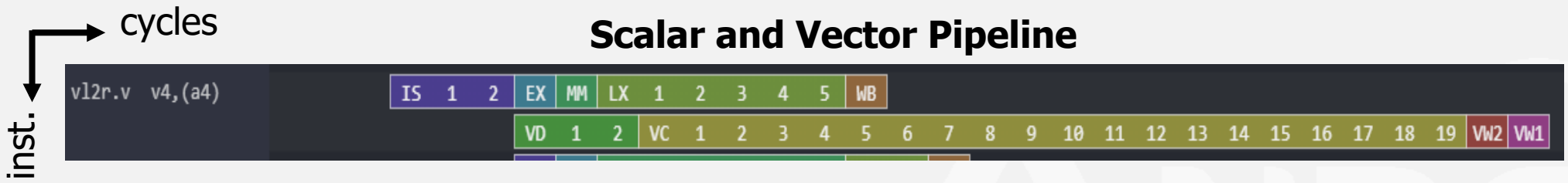
AX45MPV microarchitecture :

- Dual issue, 8-stage in-order scalar pipeline
- Dual issue, out-of-order execution vector pipeline
- Support ACE



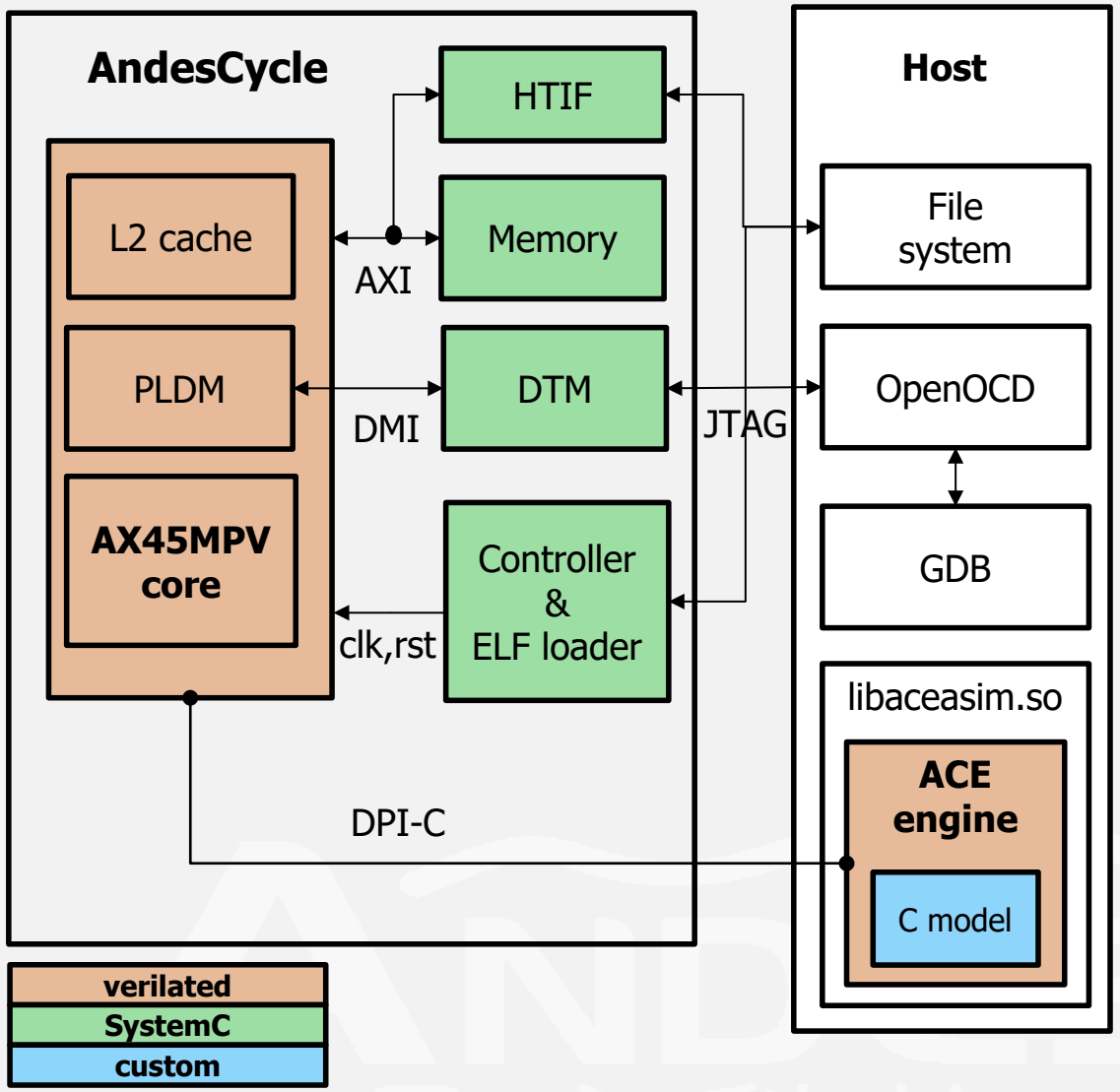
Pipeline view capabilities :

- Instruction stall analysis
- Pipeline utilization analysis



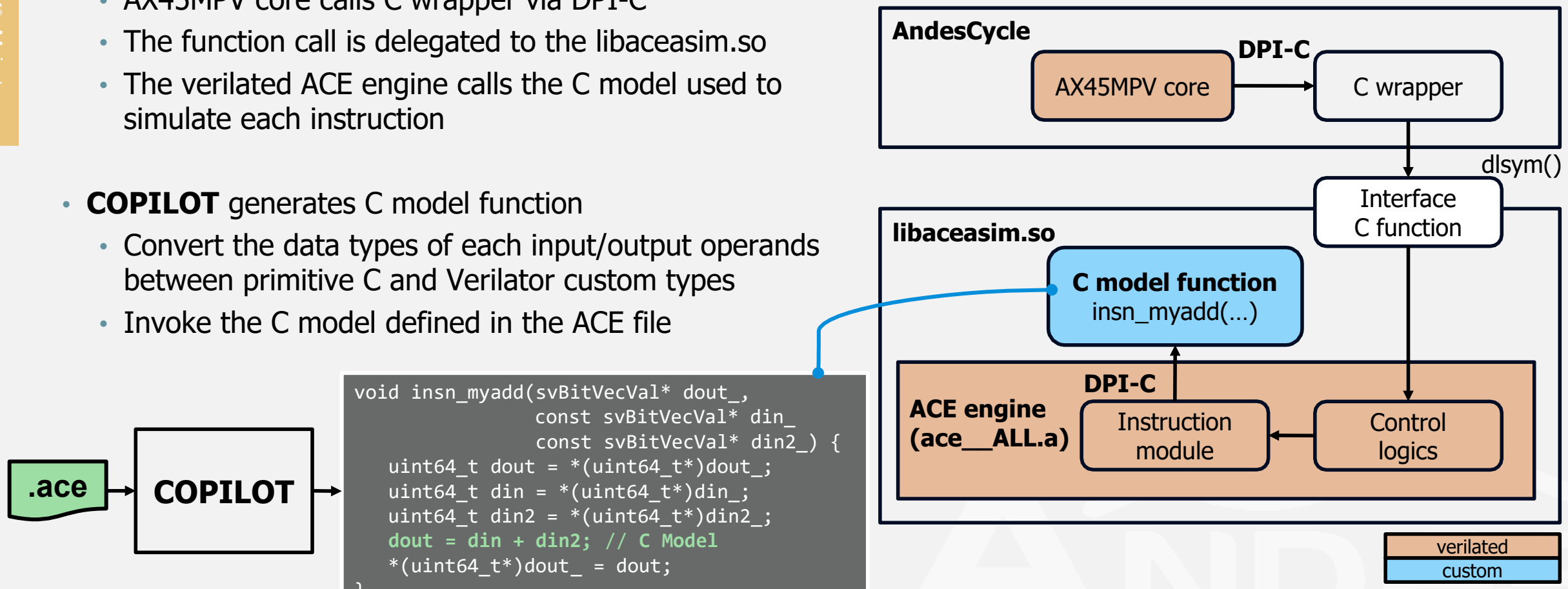
AndesCycle Architecture

- **CPU module**
  - AX45MPV core
  - L2 cache
  - Platform Debug Module (PLDM)
  - Signal interfaces (AXI, AHB, DMI) in SystemC TLM2.0
- **Platform peripherals**
  - HTIF : host-target interface, handle system call
  - Memory : simple memory modeling
  - DTM : debug transport module
  - ELF loader : load binary to memory
- **ACE engine**
  - Verilated from RTL
  - Communicated with ACE core via DPI-C



ACE Instruction Support in AndesCycle

- Steps to support **ACE** instructions
  - AX45MPV core calls C wrapper via DPI-C
  - The function call is delegated to the libaceasim.so
  - The verilated ACE engine calls the C model used to simulate each instruction
- **COPILOT** generates C model function
  - Convert the data types of each input/output operands between primitive C and Verilator custom types
  - Invoke the C model defined in the ACE file



Case Study : Video Codec Extension

- Custom RVV extension used to accelerate video codec proposed by ByteDance : (https://code.videolan.org/BD-qjy/codec-kernel-bench-riscv)

- Implement two of them in ACE-RVV :
  - Designed in C model, no need for Verilog
  - Assumed 1 cycle execution latency for LMUL=1
  - Achieved **4.5%** speedup

ACE file

```
rvv_insn vwabdu_vv {
    operand = { out vr:2uint vd, in  vr:uint vs2, in  vr:uint vs1 };
    vector_mask = out;
    vector_unit = elen;

    csim = %{
        unsigned csr_vtype_vsew = (csr_vtype >> 3) & 0x7;
        switch (csr_vtype_vsew) {
            case 0: {
                for (int i = 0; i < 4; i++)
                    vd.u16[i] = abs(vs2.u8[i] - vs1.u8[i]);
                break;
            }
            case 1: {
                vd.u32[0] = abs(vs2.u16[0] - vs1.u16[0]);
                vd.u32[1] = abs(vs2.u16[1] - vs1.u16[1]);
                break;
            }
            case 2: {
                vd.u64 = abs(vs2.u32 - vs1.u32);
                break;
            }
            case 3: {
                break;
            }
        }
    %};

    latency = 1;
};
```

Verilog

```
// ACE_BEGIN: vwabdu_vv

wire [3:0] sew_onehot;
assign sew_onehot = {vace_vtype_sew64, vace_vtype_sew32, vace_vtype_sew16,
vace_vtype_sew8};
reg [63:0] rsl

always @* begin
  casez (sew_onehot)
    4'b1??: begin // SEW64
      rslt64 = {{32'b0, vs1[63:32]} - {32'b0, vs2[63:32]} ? {32'b0, vs1} - {32'b0, vs2}) : {32'b0, vs2}
- {32'b0, vs1}};
    end
    4'b01?: begin // SEW16
      rslt64[32  -1:0]  = {{16'b0, vs1[(32  -1):16]} - {16'b0, vs2[(32  -1):16]}) ?
                           {{16'b0, vs1[(32  -1):16]} - {16'b0, vs2[(32  -1):16]}) :
                           {{16'b0, vs2[(32  -1):16]} - {16'b0, vs1[(32  -1):16]});
      rslt64[(32*2 -1):32] = {{16'b0, vs1[(64  -1):(32*2 -1)]} - {16'b0, vs2[(64  -1):(32*2 -1)]}} > {16'b0, vs2[(16*2 -1):16]}) ?
                           {{16'b0, vs1[(16*2 -1):16]} - {16'b0, vs2[(16*2 -1):16]}) :
                           {{16'b0, vs2[(16*2 -1):16]} - {16'b0, vs1[(16*2 -1):16]});
    end
    default: begin // SEW8
      rslt64[(16  -1):0]    = {{8'b0, vs1[(16  -1):8]}    - {8'b0, vs2[(8   -1):0]}) ?
                               {{8'b0, vs1[(16  -1):8]}    - {8'b0, vs2[(8   -1):0]}) :
                               {{8'b0, vs2[(16  -1):8]}    - {8'b0, vs1[(8   -1):0]});
      rslt64[(16*2 -1):16]  = {{8'b0, vs1[(8*2 -1):8]}    - {8'b0, vs2[(8*2 -1):8]}) ?
                               {{8'b0, vs1[(8*2 -1):8]}    - {8'b0, vs2[(8*2 -1):8]}) :
                               {{8'b0, vs2[(8*2 -1):8]}    - {8'b0, vs1[(8*2 -1):8]});
      rslt64[(16*3 -1):16*2] = {{8'b0, vs1[(8*3 -1):8*2]}    - {8'b0, vs2[(8*3 -1):8*2]})
?
                               {{8'b0, vs1[(8*3 -1):8*2]}    - {8'b0, vs2[(8*3 -1):8*2]}) :
                               {{8'b0, vs2[(8*3 -1):8*2]}    - {8'b0, vs1[(8*3 -1):8*2]});
      rslt64[(16*4 -1):16*3] = {{8'b0, vs1[(8*4 -1):8*3]}    - {8'b0, vs2[(8*4 -1):8*3]})
?
                               {{8'b0, vs1[(8*4 -1):8*3]}    - {8'b0, vs2[(8*4 -1):8*3]}) :
                               {{8'b0, vs2[(8*4 -1):8*3]}    - {8'b0, vs1[(8*4 -1):8*3]});
    end
  endcase
end
```

ACE-RVV instruction	Operations
vwabdu.vv vd, vs2, vs1, vm	vd[i] = abs(vs2[i]-vs1[i])
vwabdau.vv vd, vs2, vs1, vm	vd[i] = vd[i] + abs(vs2[i]-vs1[i])

16x16 kernel in ACE-RVV

```
...
vwabdu_vv    v16,v0,v8
vwabdau_vv   v16,v4,v12
...
```

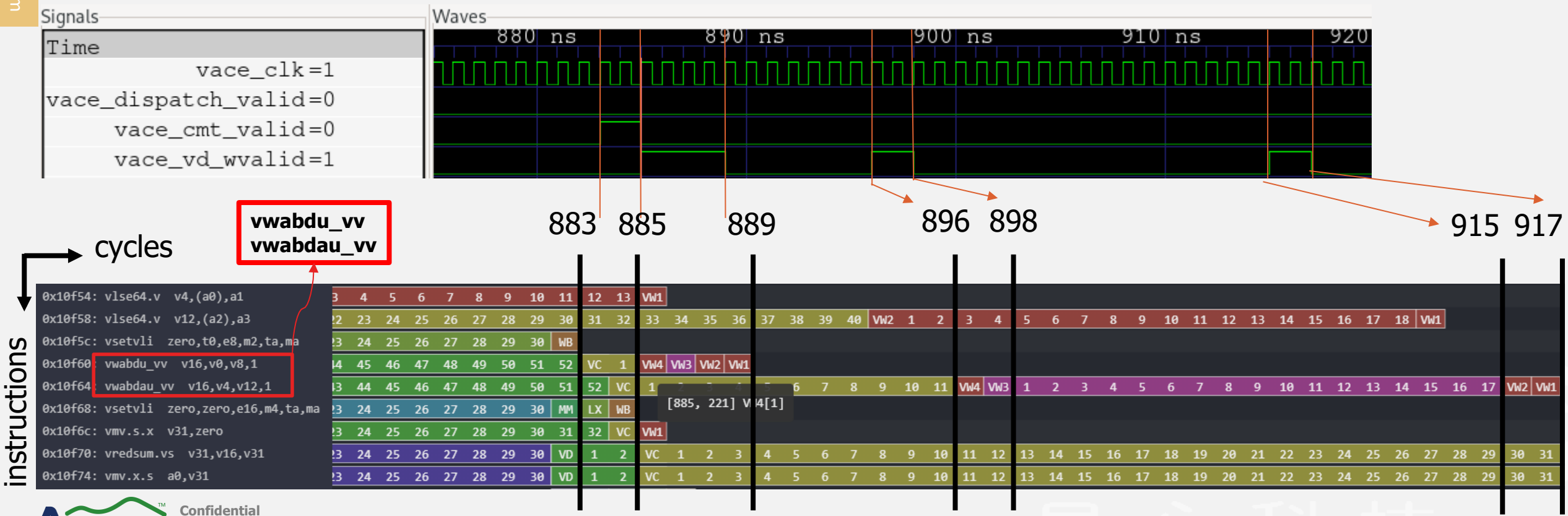
16x16 kernel in RVV

```
...
vmaxu.vv     v28,v0,v8
vminu.vv      v24,v0,v8
vwsubu.vv    v16,v28,v24
vmaxu.vv     v28,v4,v12
vminu.vv     v24,v4,v12
vwsubu.vv    v8,v28,v24
...
```

6 inst.

Pipeline and Waveform Alignment

- Comparing the pipeline view (from C model simulation) with waveform (from RTL design) shows they align precisely, demonstrating the accuracy of C model simulation approach.



Case Study : Activation Function in ML

- The non-linear element-wise sigmoid function is implemented in RVV with various approximation methods :
  - Polynomial approximation
  - Hard sigmoid approximation

- Implement an ACE-RVV instruction (**sigmoid.vv**) for the sigmoid function
  - Designed in C model with berkeley-softfloat-3 library
  - Assumed 1 cycle execution latency for LMUL=1
  - Achieved **39%** speedup

Implementation	Operation	Cycles
Polynomial	0.5 + x*(0.25 - x^2/48)	66
Hard sigmoid	clip(0.2·x+0.5, 0, 1)	53
ACE-RVV **sigmoid.vv**	1 / (1+ e^(-x))	**38**

1.39x

ACE file

```
rvv_insn sigmoid_vv {
    operand = { out vr:fp vd, in vr:fp vs1 };
    vector_mask = out;
    vector_unit = elen;

    csim = %{
        unsigned csr_vtype_vsew = (csr_vtype >> 3) & 0x7;
        if (csr_vtype_vsew == 1)
            for (int i = 0; i < 4; i++)
                vd.f16[i] = softfloat_op16(vs1.f16[i], sigmoid);
        else if (csr_vtype_vsew == 2)
            for (int i = 0; i < 2; i++)
                vd.f32[i] = softfloat_op32(vs1.f32[i], sigmoid);
        else if (csr_vtype_vsew == 3)
            vd.f64 = softfloat_op64(vs1.f64, sigmoid);
    %};

    latency = 1;
};
```

ACE-RVV

```
...
 sigmoid.vv  v16, v0, 1
...
```

Hard sigmoid (RVV)

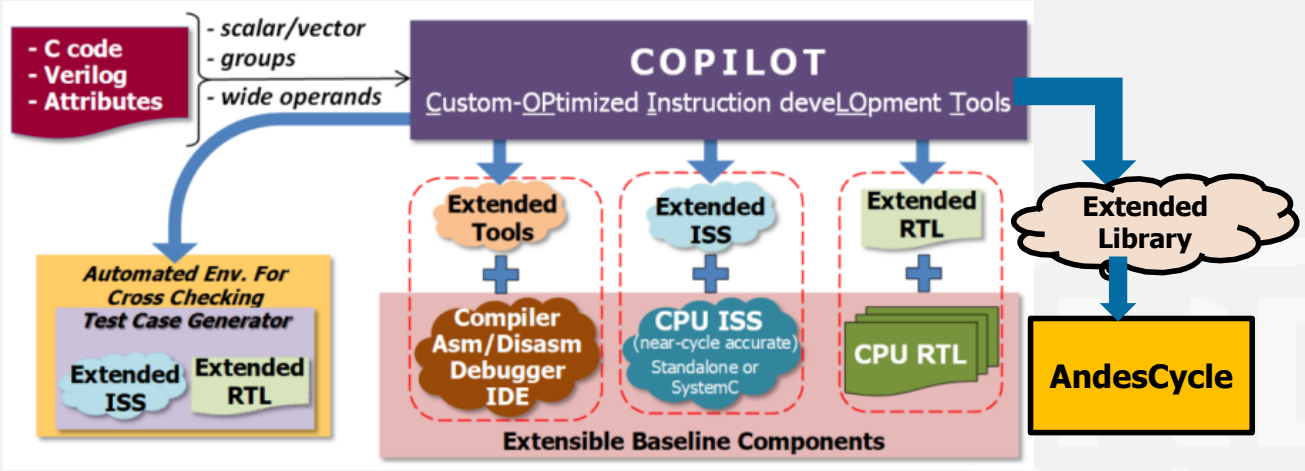
```
...
flw     fa0, coef020, t1
fmv.w.x f1, zero
flw     fa2, coef1, t1
vfmul.vf v4, v0, fa0
vfadd.vf v4, v4, fa2
vfmv.v.f v8, fa1
vfmv.v.f v12, fa3
vfmax.vv v16, v4, v8
vfmax.vv v20, v16, v12
...
```

9 inst.

Conclusions and Future works

- Conclusions
  - **ACE** framework simplifies custom instruction design and tool integration.
  - **AndesCycle** delivers detailed performance analysis for ACE instructions.
  - These tools provide a powerful solution to **accelerate custom instruction development.**

- Future works
  - Support variable latency control in the C/C++ model
  - Support pipeline custom logic design in the C/C++ model
  - Support platform builder to allows for custom hardware modules



Thank you