INTERFLOP - UNISIM-VP – ARM FRONT-END

October 6th 2021 – INTERFLOP Semi-Annual Meeting – Yves LHUILLIER, Franck VEDRINE

INTERFLOP
FP

HOW DID WE GET HERE ?

- **4 objectives from *sound* to … *less sound***
  - Enhance numerical analysis capabilities of our simulators (INTERFLOP)
  - Tackle numerical stability in artificial intelligence algorithms
  - Tackle numerical problems in representative embedded systems
  - Tackle GPU simulation and modeling

GAIA-X : LES RÊVES FOUS DU PROJET EUROPÉEN
NOUS AVONS POUR VOLONTÉ DE CRÉER UNE SURCOUCHE ADMINISTRATIVE COMPLEXE EN PARTENARIAT AVEC DES CENTRES DE RECHERCHE SOUS-ALIMENTÉS POUR RAISONS BUDGÉTAIRES ET DES STARTUPS COURT-TERMISTES QUI N'ONT PAS LES MOYENS DE LEURS AMBITIONS
ÇA VA DÉCHIRER
GAIA-X
data-infrastructure.eu
GAIA-X
NEXT INPACT

- **Means to do it**
  - Start a new ARMv8 simulator
  - Make it capable of running a full linux distro (+ Keras + TF + OpenCL)
  - Retain all previous UNISIM-VP instrumentation capabilities
  - Provide the optimal instruction floating-point description
    - Efficient Simulation
    - Flexible Instrumentation
    - Optimize Code factorization

OUTLINE

➡ UNISIM-VP Quick Overview

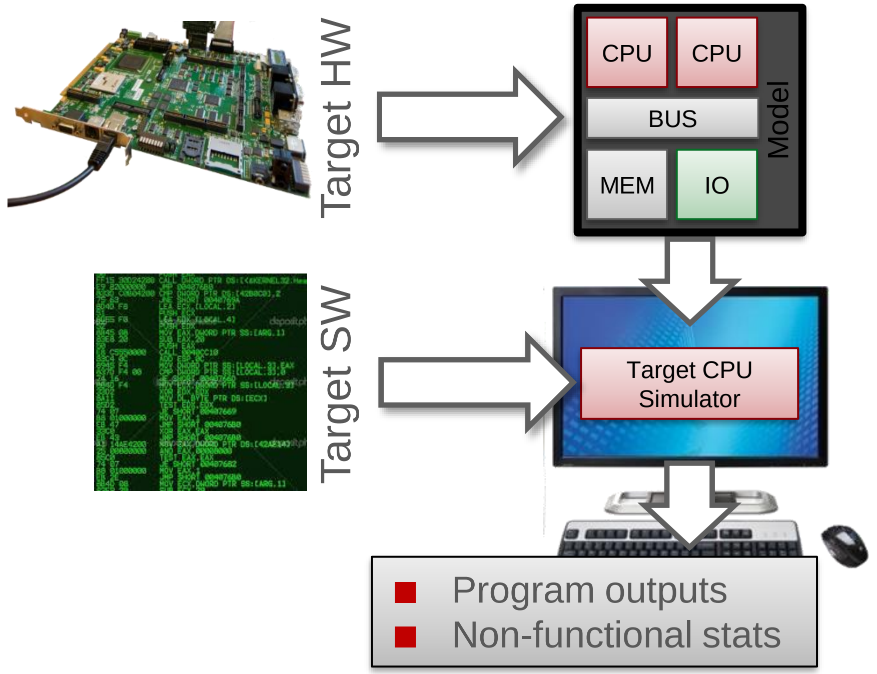
Pre-Existing instrumentation capabilities

On-going work & FP instrumentations

UNISIM-VP QUICK OVERVIEW

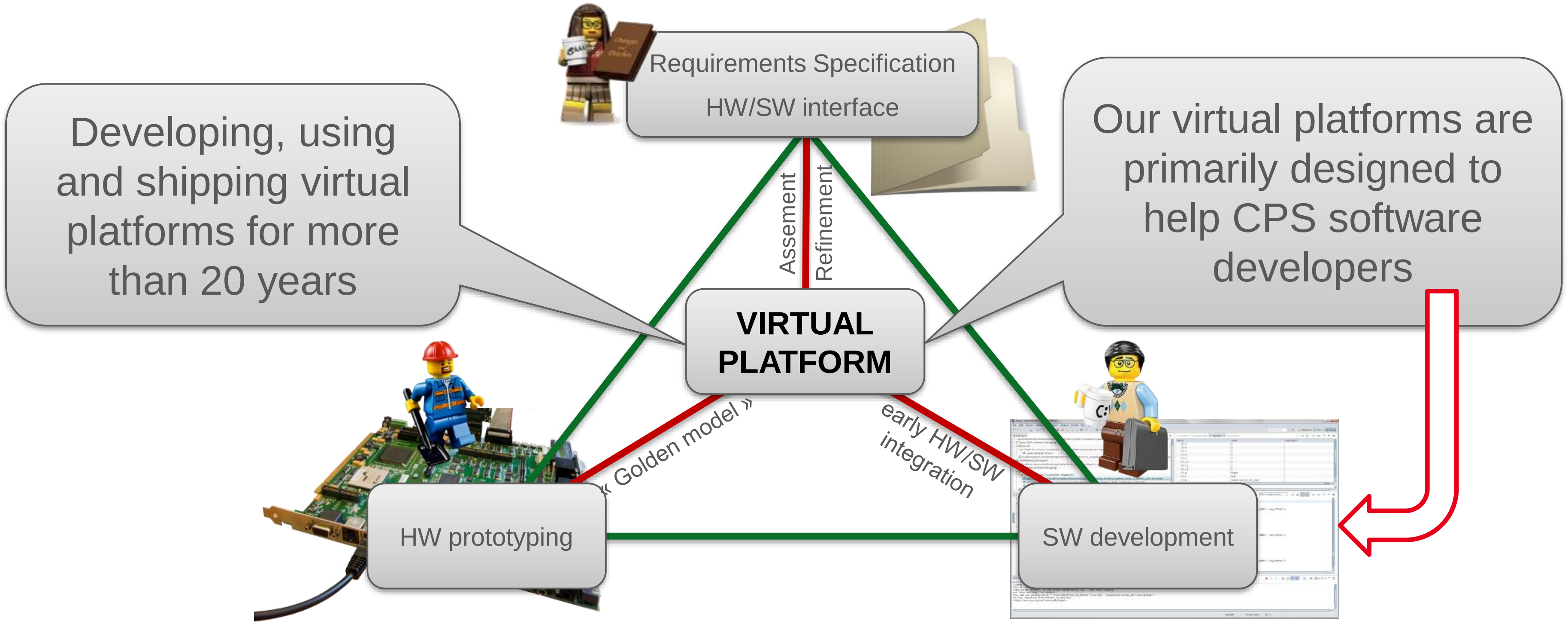
- **Electronic Systems Virtualization**
  - Hardware design exploration
  - Test bench virtualization
  - Interoperability
  - *Resource optimization*
  - Instrumentation
  - Sandboxing (cybersecurity)

- **Reusing common tools for V&V, safety and cybersecurity**
  - Decrease development costs
  - Increase confidence



UNISIM-VP QUICK OVERVIEW

Models and simulators are critical tools for designing and testing and validating electronic and cyber-physical systems.


Requirements Specification
HW/SW interface
Developing, using and shipping virtual platforms for more than 20 years
Our virtual platforms are primarily designed to help CPS software developers
Assement
Refinement
VIRTUAL PLATFORM
« Golden model »
early HW/SW integration
HW prototyping
SW development

http://unisim-vp.org

What we want

Tools for software development
- Early prototyping and debug
- Verification and validation
- Maintenance and Monitoring

Tools for code analysis
- Safety & Security
- Numerical stability
- Reverse engineering

What we develop

Hardware Components Models | **Virtualization Technologies** | **Software Validation Services** | **Third Party Tools Interfaces**

- SoCs & Boards

- Processor models
- Simulation standards

- OS emulation
- Binary Translation

AUTOSAR
RTOS*

- Debug & Profile
- Fuzzing, µExec
- Code Sanitizing

GNU GDB
Visual Studio Code
MATLAB
LAUTERBACH TRACE32
eclipse

Hightlights & Major achievements

•Full System Simulators

- Full software stacks

•ISA decoder front-ends for CyberSecurity

- Interfacing internal & external tools

•Numerical Stability Analyzers

- Numerical Stability in embedded systems (train)

•Support for rare processor brands

- High confidence level

« Validation with code introspection of a virtual platform for sandboxing and security analysis » In *C&ESAR 2019 Conference - Virtualization and Cybersecurity*, 2019

Roadmap

Scientific

- Embedded/Edge AI Code Analysis ToolSuite
- Enhanced tooling for HW/SW reverse engineering

•Technical

- GPU / TPU / NPU simulation
- Ease addition of Non-expert interface

OUTLINE

UNISIM-VP Quick Overview

➡ Pre-Existing instrumentation capabilities

On-going work & FP instrumentations

ADVANCED AND REPRESENTATIVE UNISIM INSTRUMENTATION

- **Connecting our concrete simulators to symbolic tools**
- **Using unmodified original simulator code**
- **Heavy use of C++ templates to instantiate different simulator instances (recompilation)**
- **Enables:**
  - Connection to symbolic and formal tools
  - Self-introspection, e.g. for simulation self-tests

EXTENDING UNISIM-VP FOR GENERIC CODE ANALYSIS

Connecting our concrete simulators to symbolic tools

How ? Re-write all hardware simulators to use flexible data types.

Curiously enough, this is often not that hard (thanks to C++)…

Ex: processor simulator

UNISIM-VP Instruction Set Description

Encodings – Disassembly – Behavior

```
add.execute( cpu )
{
  U32 operand1 = cpu.GetRegister( r1 );
  U32 operand2 = cpu.GetRegister( r2 );
  U32 result = operand1 + operand2;
  cpu.SetRegister( result );
}
```

Concrete

R1	R2	RD
10	5	0
⬇	⬇	⬇
10	5	15

Executing instruction on **concrete** values

Symbolic

R1	R2	RD
X	Y	Z
⬇	⬇	⬇
X	Y	X+Y

Executing instruction on **symbolic** values

Provides a uniform way to deploy a wide range of code analyzers

- **Numerical stability (floating point instrumentation)**
- **Code sanitizing (e.g. computation on uninitialized values)**
- **Taint analysis**

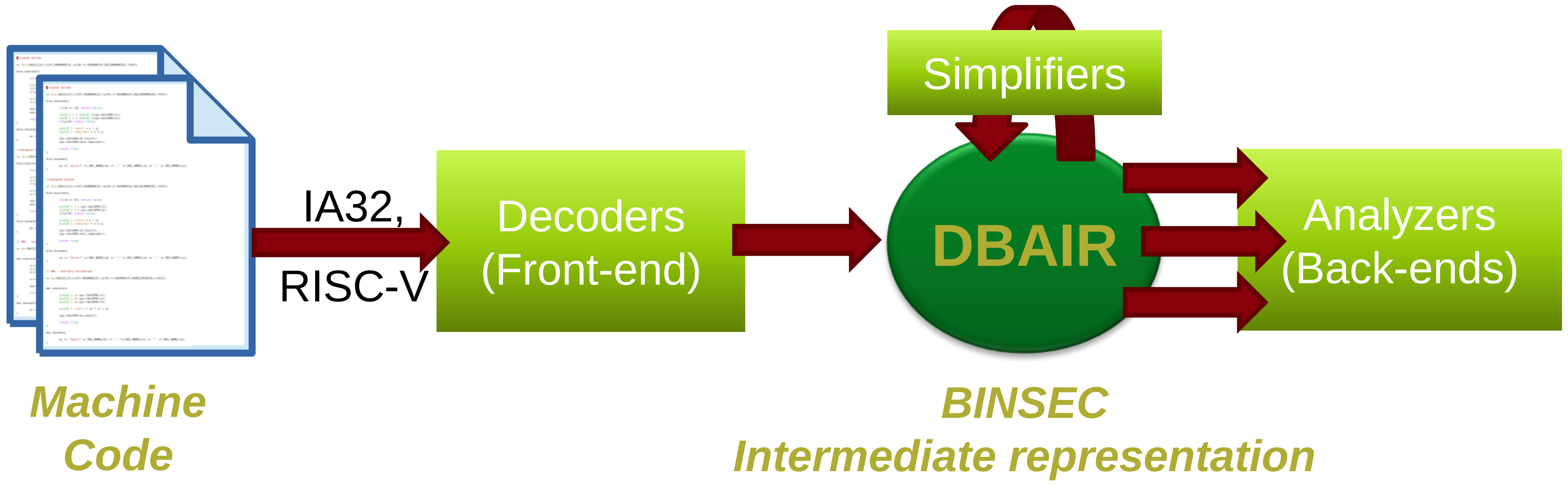
SECURITY USE-CASE: THE BINSEC TOOL

BINSEC is a binary code analyzer for security

leveraging formal methods & automatic code verification

BINSEC

https://binsec.github.io



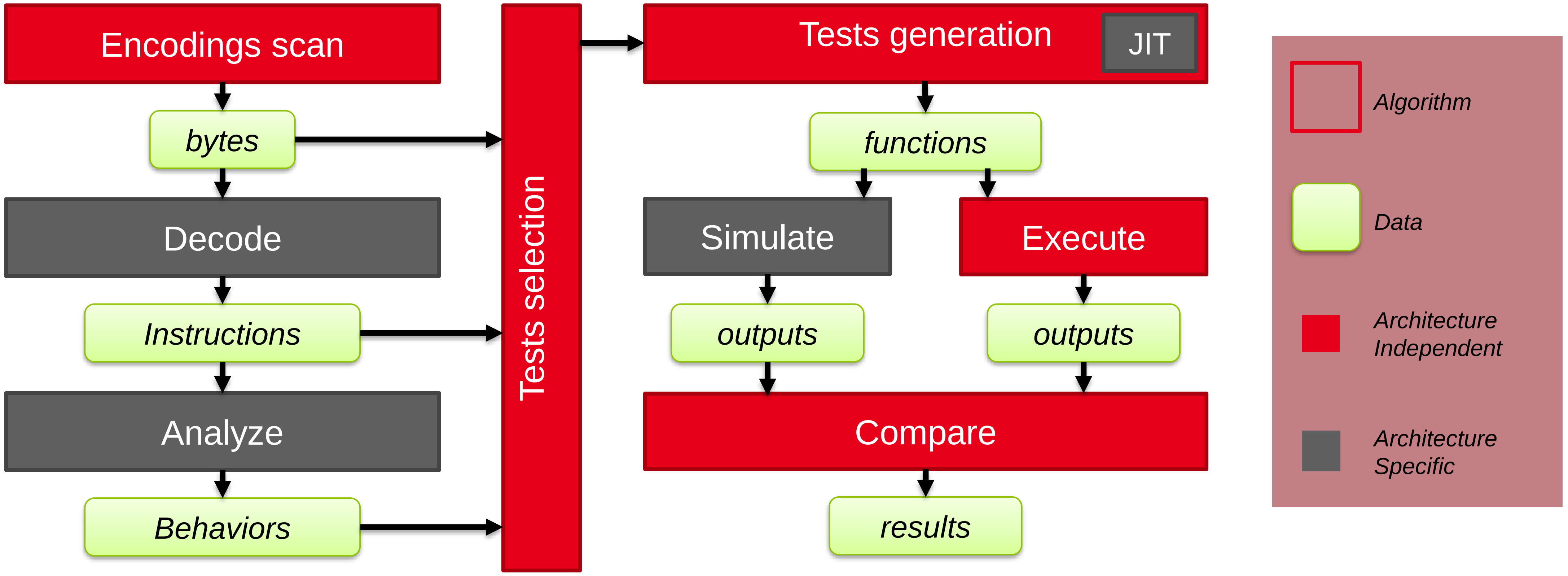
INSTRUCTION SET SIMULATOR SELF VERIFICATION

- **Why verify instruction set simulator ?**
  - Some critical systems are tested and validated on simulators
  - Formal methods based on semantic accuracy

- **Why validate using single instruction tests (unit tests) ?**
  - Simulator errors are hard to spot on full application runs
  - Only way to verify every machine instruction

- **Why a self-verifying simulator ?**
  - Writing unit tests is tedious
  - Running unit tests is tedious

PRINCIPLES OF THE SELF-VERIFYING SIMULATOR



INSTRUCTION DISCOVERY

2 usual techniques:

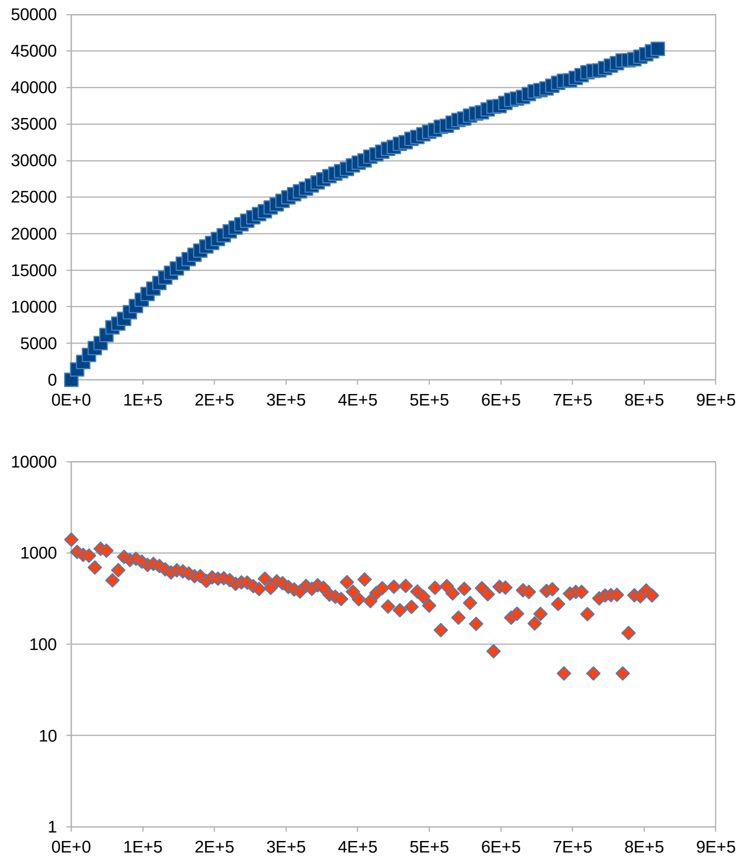
- Random generation of instruction bytes
- Derive random encodings of documented instructions

Unfortunately infeasible in x86-64

0-4	1-3	?1	?1	?1,2,4	?1,2,4
Prefixes	OpCode	ModR/M	SIB	Displacement	Immediate

- Up to 15 bytes long instructions
- (much) more than one way to encode an instruction

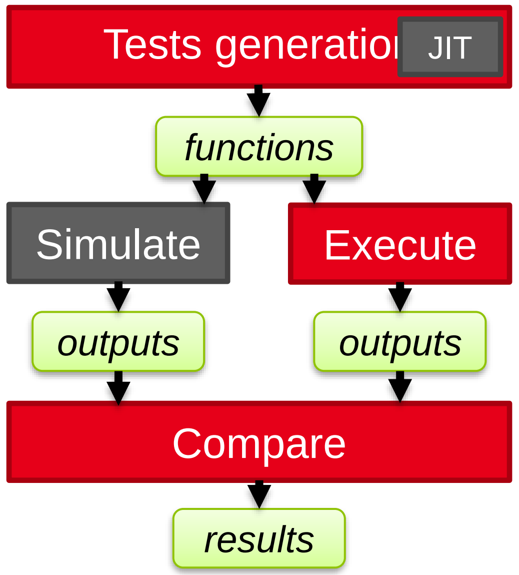
Forced to use a little of x86-64 knowledge



Rate of instruction discovery

INSTRUCTION COMPARISON

- **Monitoring instruction side effect is complex**
  - Looking at everything that may have changed is hard
  - Looking only at things that changed in simulation is biased
  - Some side effects may be hard to compare
- **Once again a little x86-64 knowledge is required**
  - Lookup simulation side-effects
  - Lookup common hardware feature: flag values
  - Transfer comparison values to memory
- **Some side effects may (still) be hard to compare**



Native execution of instruction tests raise issues

- Side effect may be unsafe for execution
- Random inputs may generate errors

Side effect	difficulty	Resolution
Memory access	May access memory randomly	a) Fix target address using input value control b) Run in an protected environment
Branch	May transfer control to any memory location	Step in debugger
Interrupts and Traps	Leave the program scope	Run in an hypervisor
Hypervisor ops	Hard to confine	Hardware debug ?
Access HW counters	Environment dependent values	No satisfying solution to our knowledge

A SANDBOX DETECTION BUG IN QEMU

For other simulators:

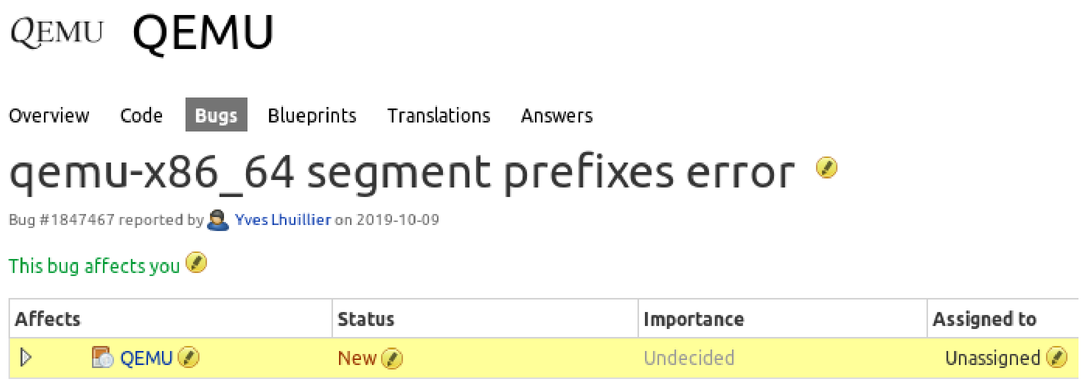
- Self-testing should work elsewhere
- Generated tests can also be reused

Among our own bugs

- We picked the trickiest
- Checked other simulators…

Incorrect simulation in QEMU

- Incorrect decoding of instruction prefixes
- Incorrect address computation
- Allows simple sandbox detection
- May confuse malware analyzers



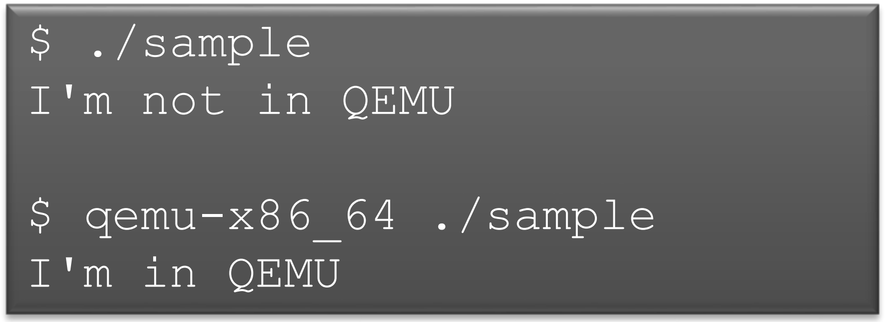
qemu-x86_64 segment prefixes error

Bug #1847467 reported by Yves Lhuillier on 2019-10-09

This bug affects you

Affects	Status	Importance	Assigned to
QEMU	New	Undecided	Unassigned

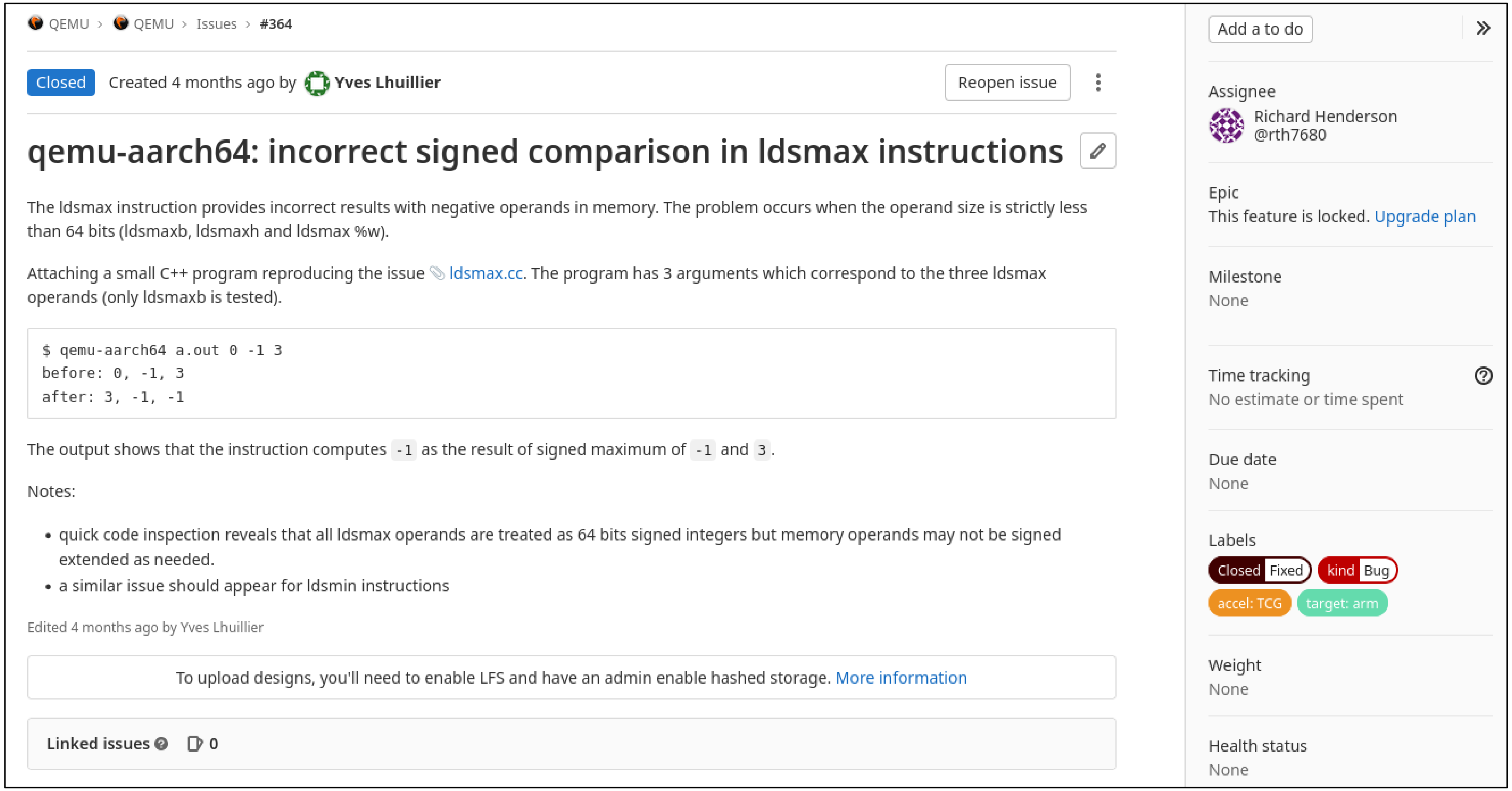
qemu-x86_64 segment prefixes error Bug #1847467



```
$ ./sample
I'm not in QEMU

$ qemu-x86_64 ./sample
I'm in QEMU
```

RECENT DISCOVERY OF A NEW FUNCTIONAL BUG



QEMU › QEMU › Issues › #364

Closed Created 4 months ago by Yves Lhuillier

Reopen issue

qemu-aarch64: incorrect signed comparison in ldsmax instructions

The ldsmax instruction provides incorrect results with negative operands in memory. The problem occurs when the operand size is strictly less than 64 bits (ldsmaxb, ldsmaxh and ldsmax %w).

Attaching a small C++ program reproducing the issue ldsmax.cc. The program has 3 arguments which correspond to the three ldsmax operands (only ldsmaxb is tested).

```
$ qemu-aarch64 a.out 0 -1 3
before: 0, -1, 3
after: 3, -1, -1
```

The output shows that the instruction computes `-1` as the result of signed maximum of `-1` and `3`.

Notes:

- quick code inspection reveals that all ldsmax operands are treated as 64 bits signed integers but memory operands may not be signed extended as needed.
- a similar issue should appear for ldsmin instructions

Edited 4 months ago by Yves Lhuillier

To upload designs, you'll need to enable LFS and have an admin enable hashed storage. More information

Linked issues 0

Add a to do

Assignee	Richard Henderson @rth7680
Epic	This feature is locked. Upgrade plan
Milestone	None
Time tracking	No estimate or time spent
Due date	None
Labels	Closed Fixed, kind Bug, accel: TCG, target: arm
Weight	None
Health status	None

OUTLINE

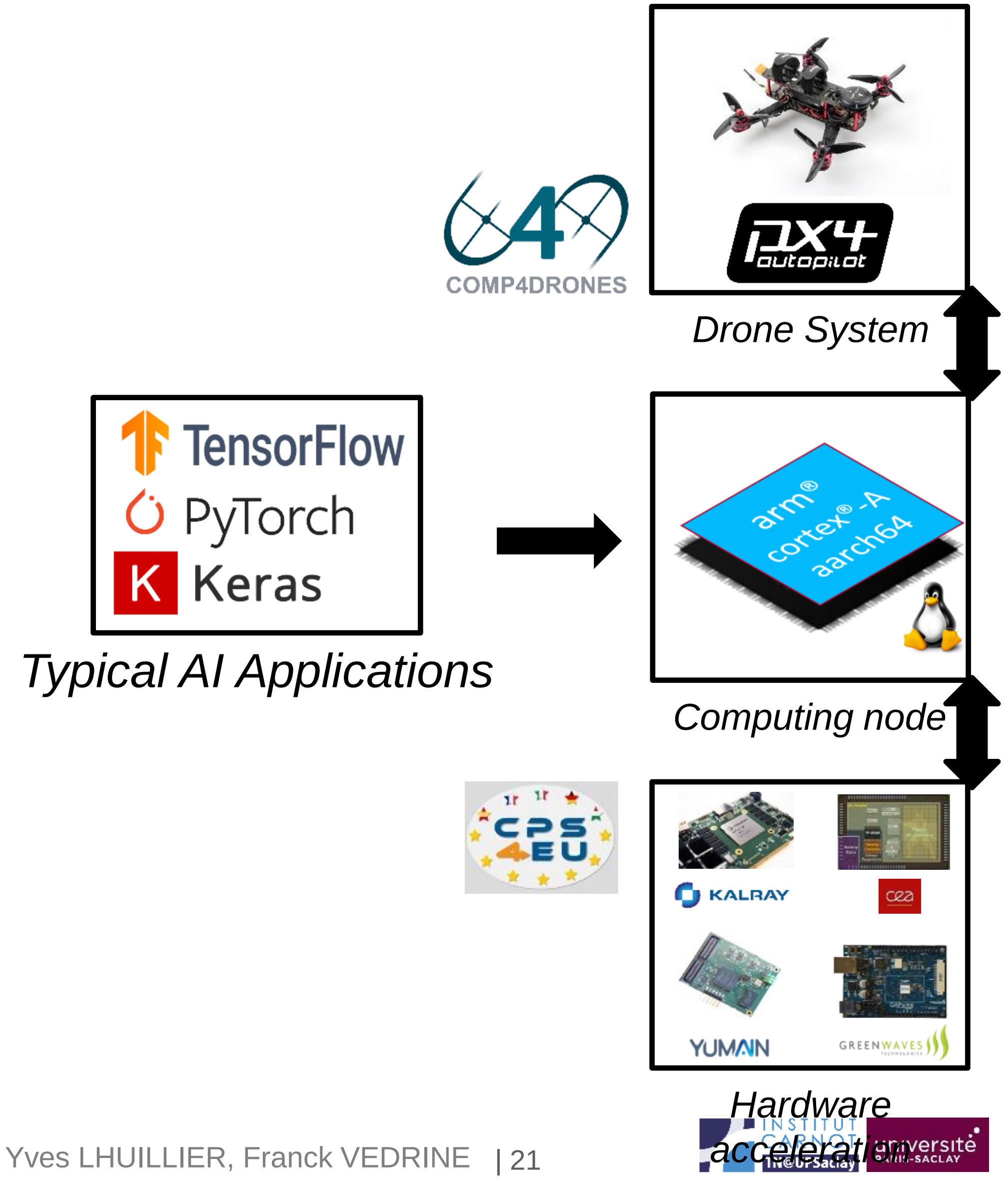
UNISIM-VP Quick Overview

Pre-Existing instrumentation capabilities

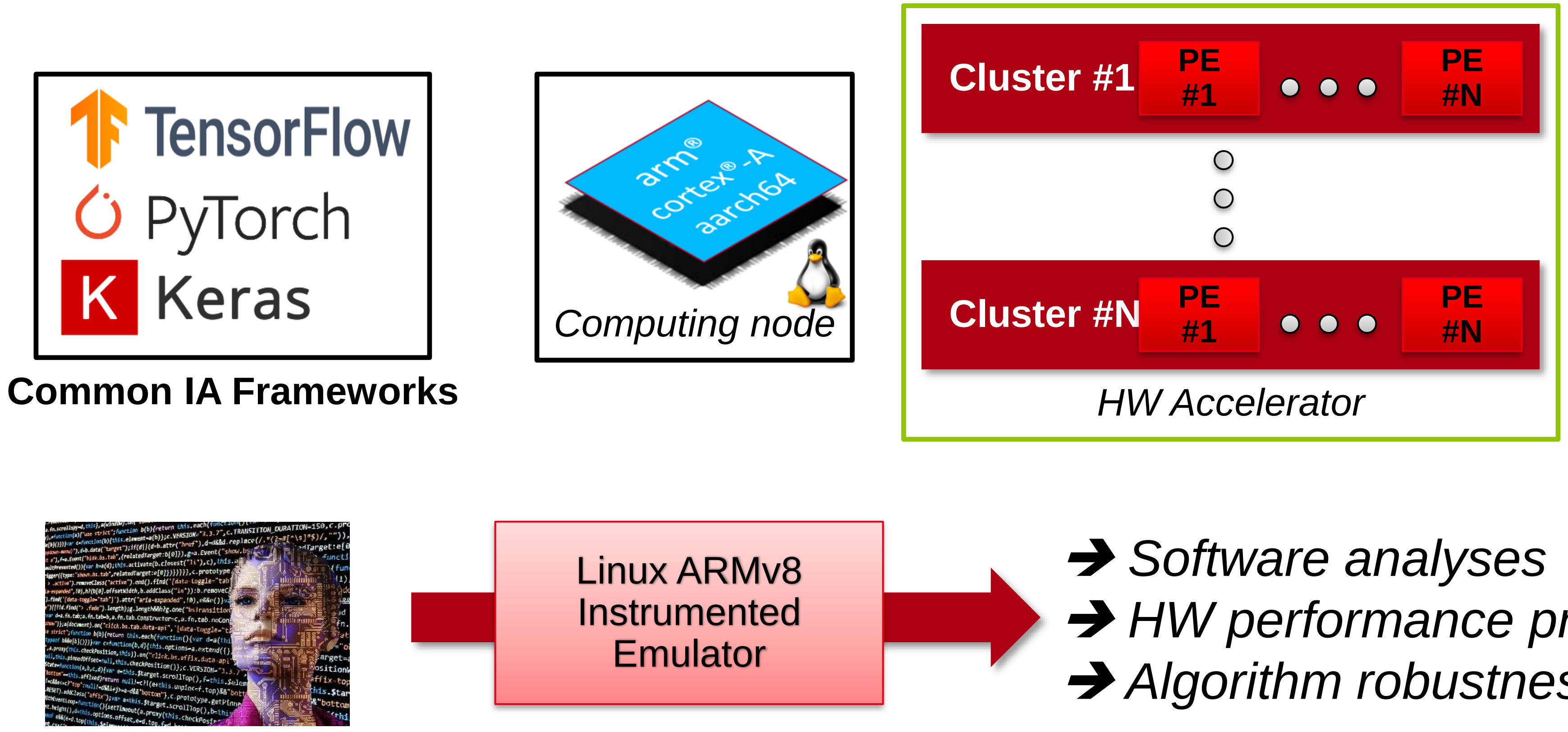
➡ On-going work & FP instrumentations

EVALUATION AND VALIDATION OF AI SOFTWARE IN EMBEDDED SYSTEMS

- **Moving AI software to Embedded/Edge**
  - For efficiency
  - For privacy
  - For responsiveness
- **Specialized Hardware selection**
  - Data type (FP, integer)
  - Precision
- **Validation on real hardware**
  - (re)validate robustness
  - Hardware failures (sensors, memory)
- **2-staged work (on-going)**
  - Full instrument-ready ARMv8/Linux Simulator
  - AI numerical analysis deployment



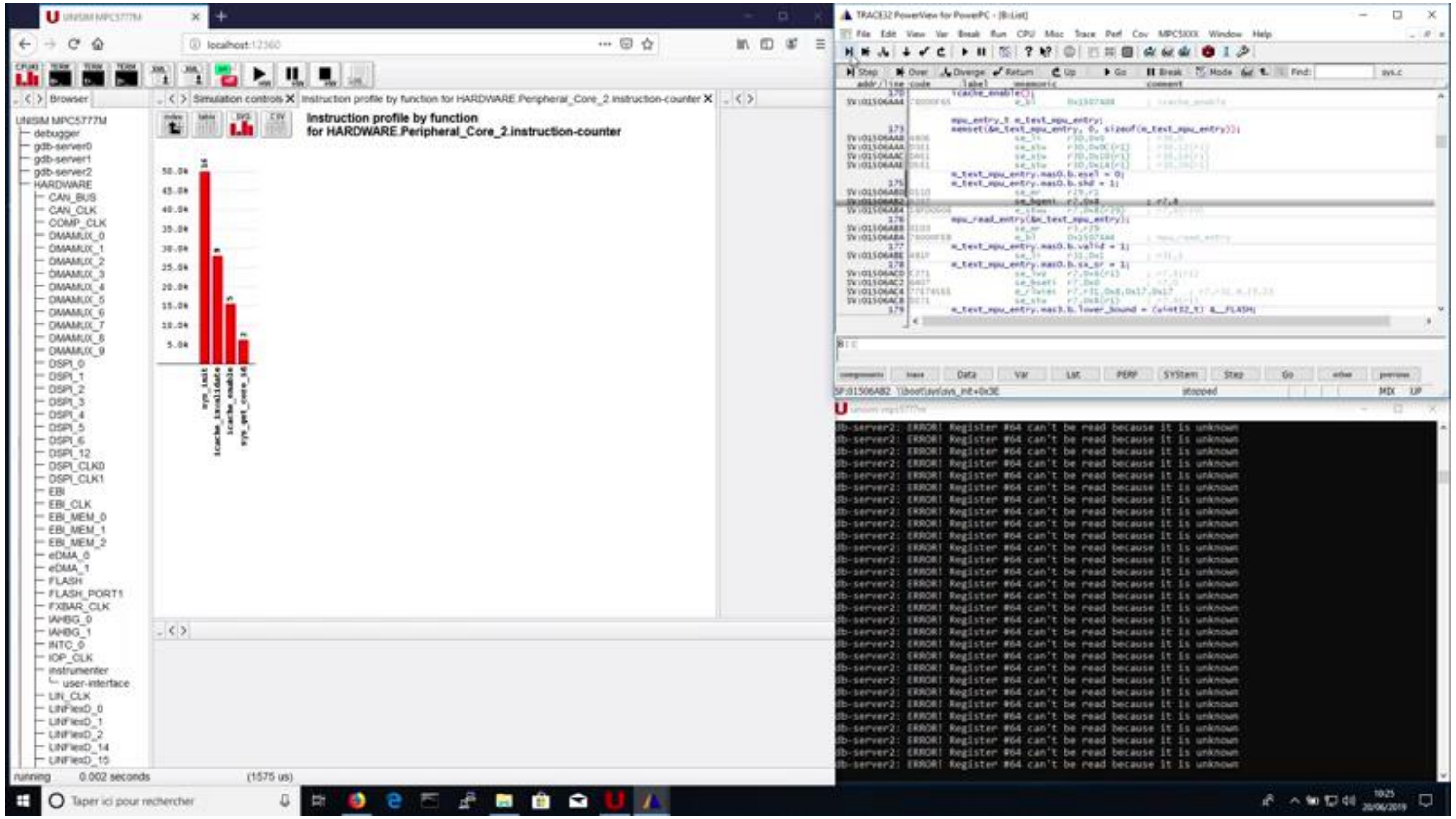
EVALUATION AND VALIDATION OF AI SOFTWARE IN EMBEDDED SYSTEMS



➔ *Software analyses*
➔ *HW performance predictions*
➔ *Algorithm robustness*

- Demonstrating an emulator capable of
  - Running full CPS software stacks (e.g. QEMU)
  - Performing numerical analysis & code sanitizing (e.g. Verrou – Valgrind)

FULL INSTRUMENT-READY ARMV8/LINUX SIMULATOR



Instruction profile by function
for HARDWARE.Peripheral_Core_2.instruction-counter

```
UNISIM MPC5777M
├ debugger
├ gdb-server0
├ gdb-server1
├ gdb-server2
├ HARDWARE
│ ├ CAN_BUS
│ ├ CAN_CLK
│ ├ COMP_CLK
│ ├ DMAMUX_0
│ ├ DMAMUX_1
│ ├ DMAMUX_2
│ ├ DMAMUX_3
│ ├ DMAMUX_4
│ ├ DMAMUX_5
│ ├ DMAMUX_6
│ ├ DMAMUX_7
│ ├ DMAMUX_8
│ ├ DMAMUX_9
│ ├ DSPI_0
│ ├ DSPI_1
│ ├ DSPI_2
│ ├ DSPI_3
│ ├ DSPI_4
│ ├ DSPI_5
│ ├ DSPI_6
│ ├ DSPI_12
│ ├ DSPI_CLK0
│ ├ DSPI_CLK1
│ ├ EBI
│ ├ EBI_CLK
│ ├ EBI_MEM_0
│ ├ EBI_MEM_1
│ ├ EBI_MEM_2
│ ├ eDMA_0
│ ├ eDMA_1
│ ├ FLASH
│ ├ FLASH_PORT1
│ ├ FXBAR_CLK
│ ├ IAHBG_0
│ ├ IAHBG_1
│ ├ INTC_0
│ ├ IOP_CLK
│ ├ instrumenter
│ │ └ user-interface
│ ├ LIN_CLK
│ ├ LINFlexD_0
│ ├ LINFlexD_1
│ ├ LINFlexD_2
│ ├ LINFlexD_14
│ ├ LINFlexD_15
```

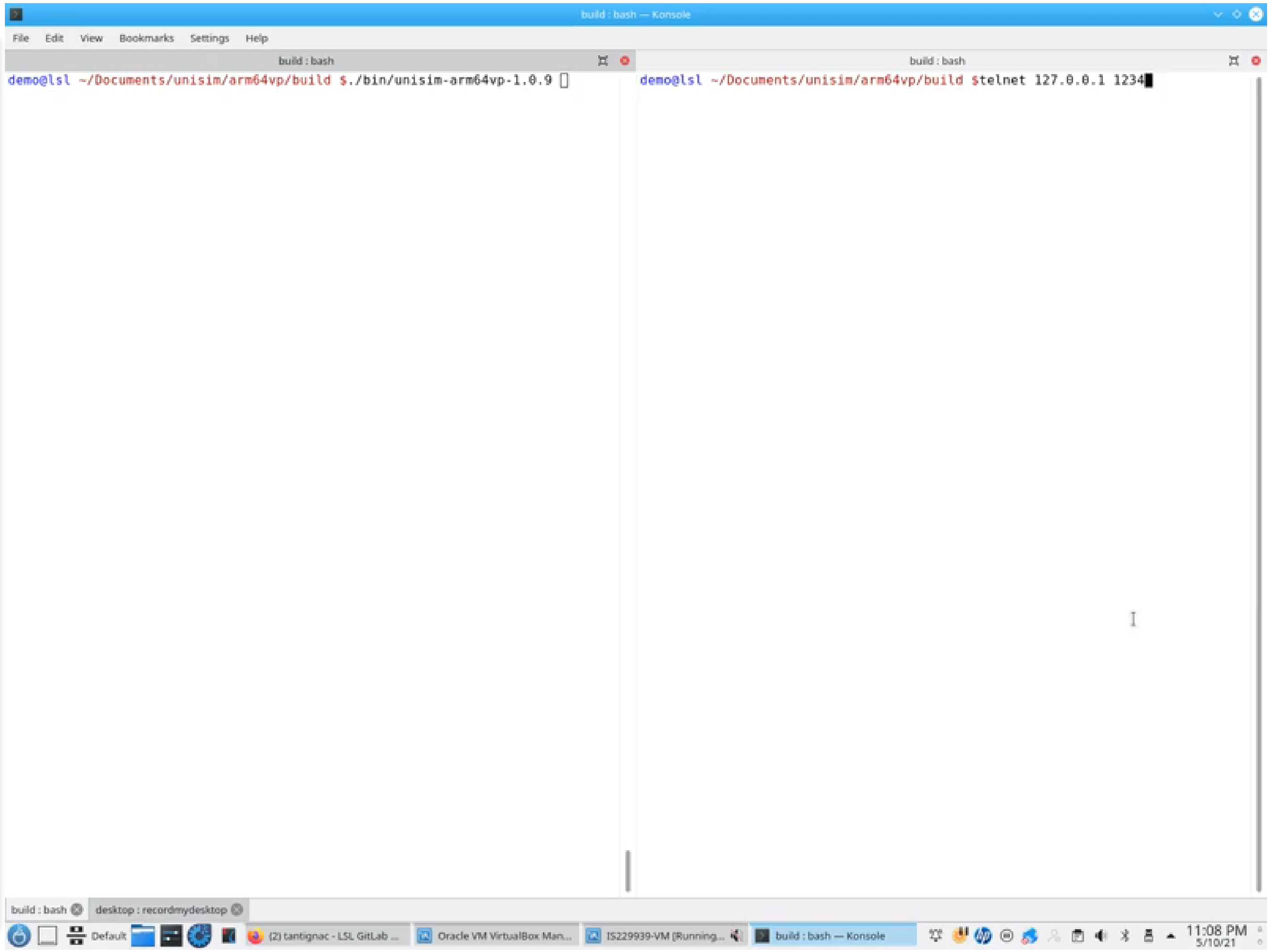
Function	Count
sys_init	56
icache_invalidate	[illegible]
icache_enable	[illegible]
sys_get_core_id	[illegible]

```
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
db-server2: ERROR! Register #64 can't be read because it is unknown
```

Classical Full System Simulator Features: debugging, profiling, coverage

FULL INSTRUMENT-READY ARMV8/LINUX SIMULATOR

Simulator console



```
demo@lsl ~/Documents/unisim/arm64vp/build $./bin/unisim-arm64vp-1.0.9
```

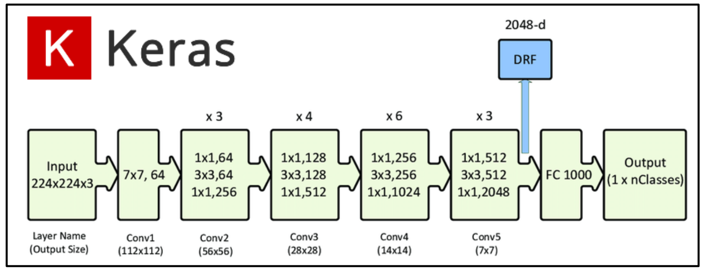
```
demo@lsl ~/Documents/unisim/arm64vp/build $telnet 127.0.0.1 1234
```

Linux boot console

Uninitialized value detector in full Linux environment

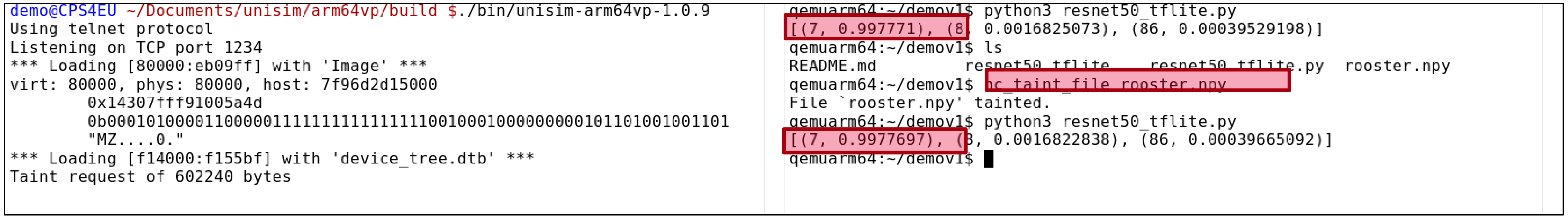
AI NUMERICAL ANALYSIS

Input image



= rooster (99.77) %

Tests realized on a state-of-the-art resnet50 object classifier



```
demo@CPS4EU ~/Documents/unisim/arm64vp/build $./bin/unisim-arm64vp-1.0.9
Using telnet protocol
Listening on TCP port 1234
*** Loading [80000:eb09ff] with 'Image' ***
virt: 80000, phys: 80000, host: 7f96d2d15000
        0x14307fff91005a4d
        0b0001010000110000011111111111111110010001000000000101101001001101
        "MZ....0."
*** Loading [f14000:f155bf] with 'device_tree.dtb' ***
Taint request of 602240 bytes
```

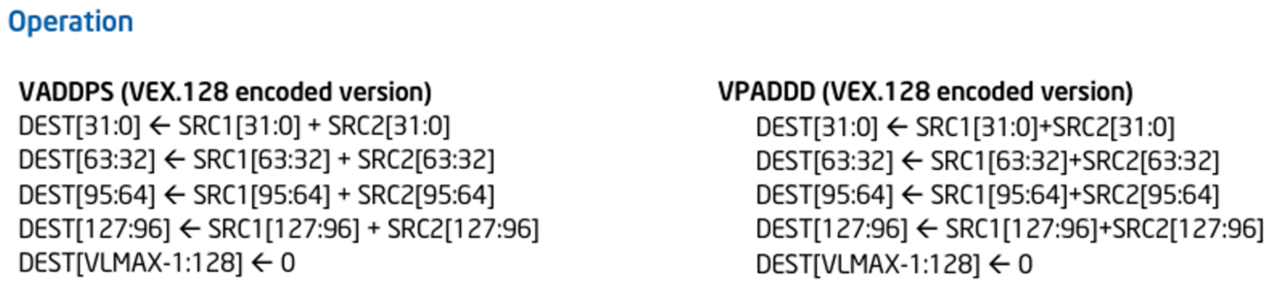
```
qemuarm64:~/demov1$ python3 resnet50_tflite.py
[(7, 0.997771), (8, 0.0016825073), (86, 0.00039529198)]
qemuarm64:~/demov1$ ls
README.md         resnet50_tflite     resnet50_tflite.py   rooster.npy
qemuarm64:~/demov1$ nc_taint_file rooster.npy
File `rooster.npy' tainted.
qemuarm64:~/demov1$ python3 resnet50_tflite.py
[(7, 0.9977697), (8, 0.0016822838), (86, 0.00039665092)]
qemuarm64:~/demov1$ 
```

Simple Numerical Analysis ➔ Direct impact observation of degraded hardware accuracy

PREPARING AN EFFICIENT FP-OPS DESCRIPTION IN UNISIM-VP

- We like our instruction descriptions to be intuitive while retaining a specification aspect

- Intel

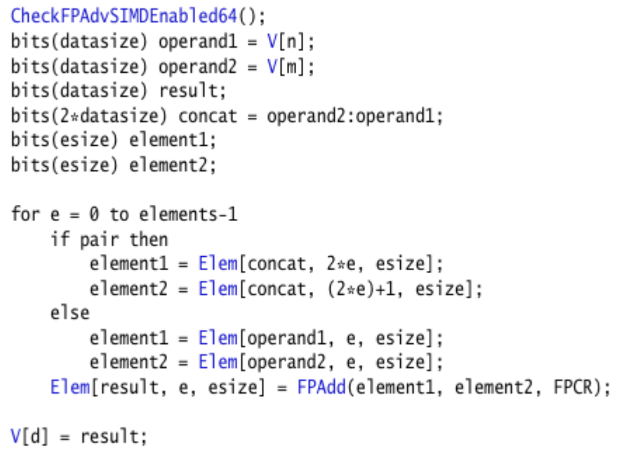


Operation

```
VADDPS (VEX.128 encoded version)
DEST[31:0] ← SRC1[31:0] + SRC2[31:0]
DEST[63:32] ← SRC1[63:32] + SRC2[63:32]
DEST[95:64] ← SRC1[95:64] + SRC2[95:64]
DEST[127:96] ← SRC1[127:96] + SRC2[127:96]
DEST[VLMAX-1:128] ← 0
```

```
VPADDD (VEX.128 encoded version)
    DEST[31:0] ← SRC1[31:0]+SRC2[31:0]
    DEST[63:32] ← SRC1[63:32]+SRC2[63:32]
    DEST[95:64] ← SRC1[95:64]+SRC2[95:64]
    DEST[127:96] ← SRC1[127:96]+SRC2[127:96]
    DEST[VLMAX-1:128] ← 0
```

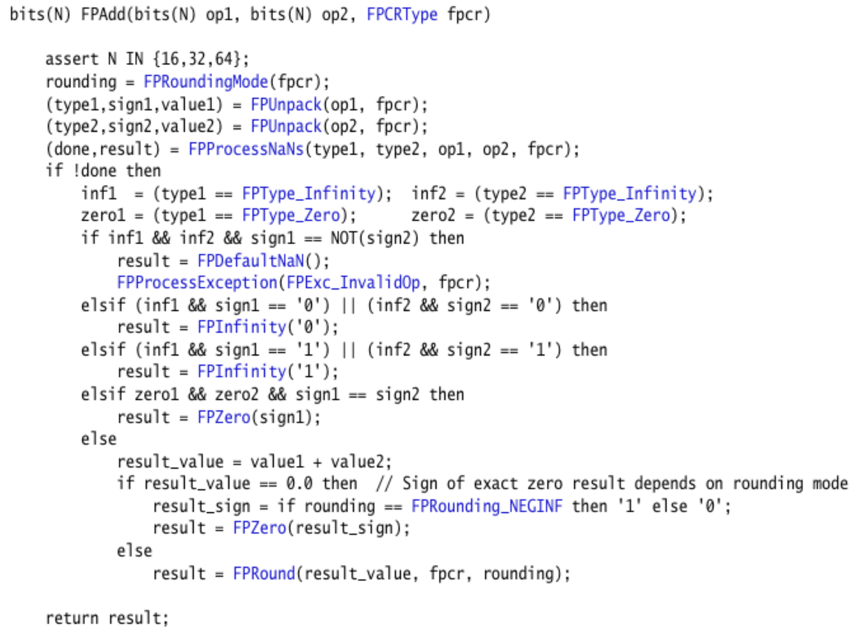
- ARM



```
CheckFPAdvSIMDEnabled64();
bits(datasize) operand1 = V[n];
bits(datasize) operand2 = V[m];
bits(datasize) result;
bits(2*datasize) concat = operand2:operand1;
bits(esize) element1;
bits(esize) element2;

for e = 0 to elements-1
    if pair then
        element1 = Elem[concat, 2*e, esize];
        element2 = Elem[concat, (2*e)+1, esize];
    else
        element1 = Elem[operand1, e, esize];
        element2 = Elem[operand2, e, esize];
    Elem[result, e, esize] = FPAdd(element1, element2, FPCR);

V[d] = result;
```

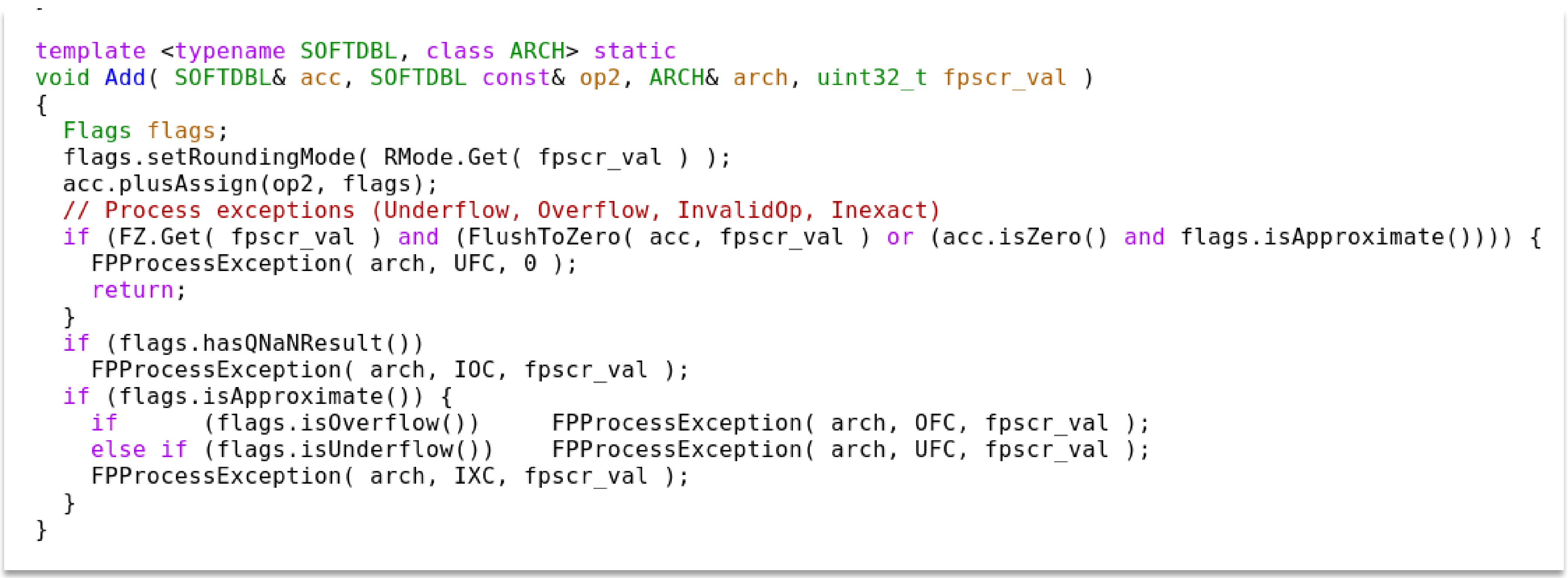


```
bits(N) FPAdd(bits(N) op1, bits(N) op2, FPCRType fpcr)

    assert N IN {16,32,64};
    rounding = FPRoundingMode(fpcr);
    (type1,sign1,value1) = FPUnpack(op1, fpcr);
    (type2,sign2,value2) = FPUnpack(op2, fpcr);
    (done,result) = FPProcessNaNs(type1, type2, op1, op2, fpcr);
    if !done then
        inf1  = (type1 == FPType_Infinity);  inf2 = (type2 == FPType_Infinity);
        zero1 = (type1 == FPType_Zero);      zero2 = (type2 == FPType_Zero);
        if inf1 && inf2 && sign1 == NOT(sign2) then
            result = FPDefaultNaN();
            FPProcessException(FPExc_InvalidOp, fpcr);
        elsif (inf1 && sign1 == '0') || (inf2 && sign2 == '0') then
            result = FPInfinity('0');
        elsif (inf1 && sign1 == '1') || (inf2 && sign2 == '1') then
            result = FPInfinity('1');
        elsif zero1 && zero2 && sign1 == sign2 then
            result = FPZero(sign1);
        else
            result_value = value1 + value2;
            if result_value == 0.0 then  // Sign of exact zero result depends on rounding mode
                result_sign = if rounding == FPRounding_NEGINF then '1' else '0';
                result = FPZero(result_sign);
            else
                result = FPRound(result_value, fpcr, rounding);

    return result;
```

PREPARING AN EFFICIENT FP-OPS DESCRIPTION IN UNISIM-VP



```
template <typename SOFTDBL, class ARCH> static
void Add( SOFTDBL& acc, SOFTDBL const& op2, ARCH& arch, uint32_t fpscr_val )
{
  Flags flags;
  flags.setRoundingMode( RMode.Get( fpscr_val ) );
  acc.plusAssign(op2, flags);
  // Process exceptions (Underflow, Overflow, InvalidOp, Inexact)
  if (FZ.Get( fpscr_val ) and (FlushToZero( acc, fpscr_val ) or (acc.isZero() and flags.isApproximate()))) {
    FPProcessException( arch, UFC, 0 );
    return;
  }
  if (flags.hasQNaNResult())
    FPProcessException( arch, IOC, fpscr_val );
  if (flags.isApproximate()) {
    if      (flags.isOverflow())     FPProcessException( arch, OFC, fpscr_val );
    else if (flags.isUnderflow())    FPProcessException( arch, UFC, fpscr_val );
    FPProcessException( arch, IXC, fpscr_val );
  }
}
```

TODO:

- **Clearly separate hardware specificities (exceptions, flags)**
- **Maximize use of legible operators (+, -, *, abs,…)**
- **(re)Connect and maintain connections to InterFlop APIs**

Thank You!

Commissariat à l’énergie atomique et aux énergies alternatives
Institut List | CEA SACLAY NANO-INNOV | BAT. 861 – PC142
91191 Gif-sur-Yvette Cedex - FRANCE
www-list.cea.fr