Generalizing the ISA to the ILA: A Software/Hardware Interface for Accelerator-rich Platforms

Sharad Malik

Princeton University

DAC 2023

July 11, 2023

This work was supported in part by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA; by the DARPA POSH and SSITH programs; and by NSF XPS Grant No. 1628926.

ILA Verification Team + Collaborators

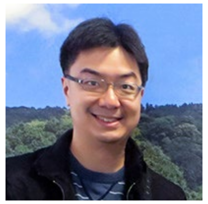
Sharad Malik

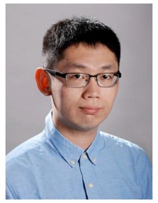
Aarti Gupta

Margaret Martonosi

Pramod Subramanyan

Bo-Yuan Huang

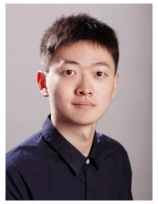

Jason Fung


Hongce Zhang

Yue Xing

Yi Li

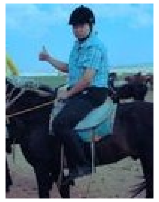
Huaixi Lu

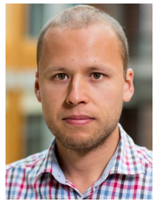

Yu Zeng

+

Sayak Ray

Yakir Vizel

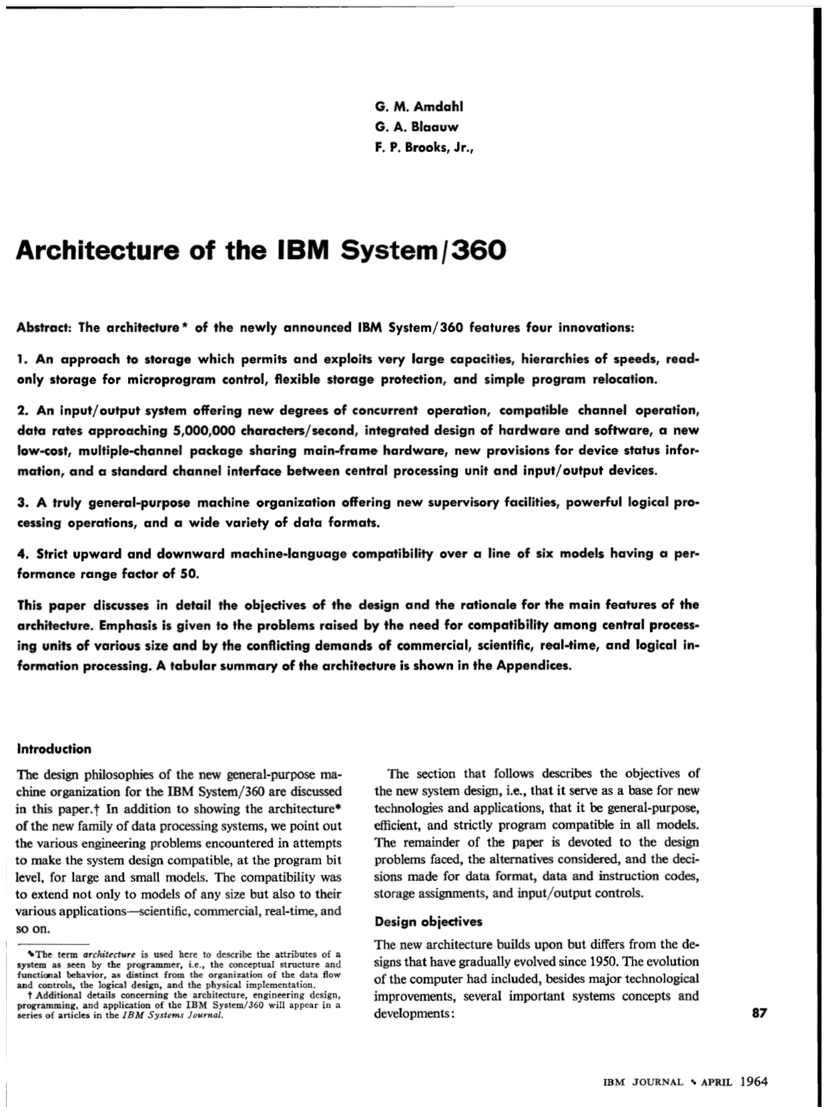

Weikun Yang


Grigory Fedyukovich

Caroline Trippel

Yatin A. Manerkar

Hardware-Software Interface



G. M. Amdahl
G. A. Blaauw
F. P. Brooks, Jr.,

Architecture of the IBM System/360

Abstract: The architecture* of the newly announced IBM System/360 features four innovations:

1. An approach to storage which permits and exploits very large capacities, hierarchies of speeds, read-only storage for microprogram control, flexible storage protection, and simple program relocation.

2. An input/output system offering new degrees of concurrent operation, compatible channel operation, data rates approaching 5,000,000 characters/second, integrated design of hardware and software, a new low-cost, multiple-channel package sharing main-frame hardware, new provisions for device status information, and a standard channel interface between central processing unit and input/output devices.

3. A truly general-purpose machine organization offering new supervisory facilities, powerful logical processing operations, and a wide variety of data formats.

4. Strict upward and downward machine-language compatibility over a line of six models having a performance range factor of 50.

This paper discusses in detail the objectives of the design and the rationale for the main features of the architecture. Emphasis is given to the problems raised by the need for compatibility among central processing units of various size and by the conflicting demands of commercial, scientific, real-time, and logical information processing. A tabular summary of the architecture is shown in the Appendices.

Introduction

The design philosophies of the new general-purpose machine organization for the IBM System/360 are discussed in this paper.† In addition to showing the architecture* of the new family of data processing systems, we point out the various engineering problems encountered in attempts to make the system design compatible, at the program bit level, for large and small models. The compatibility was to extend not only to models of any size but also to their various applications—scientific, commercial, real-time, and so on.

The section that follows describes the objectives of the new system design, i.e., that it serve as a base for new technologies and applications, that it be general-purpose, efficient, and strictly program compatible in all models. The remainder of the paper is devoted to the design problems faced, the alternatives considered, and the decisions made for data format, data and instruction codes, storage assignments, and input/output controls.

Design objectives

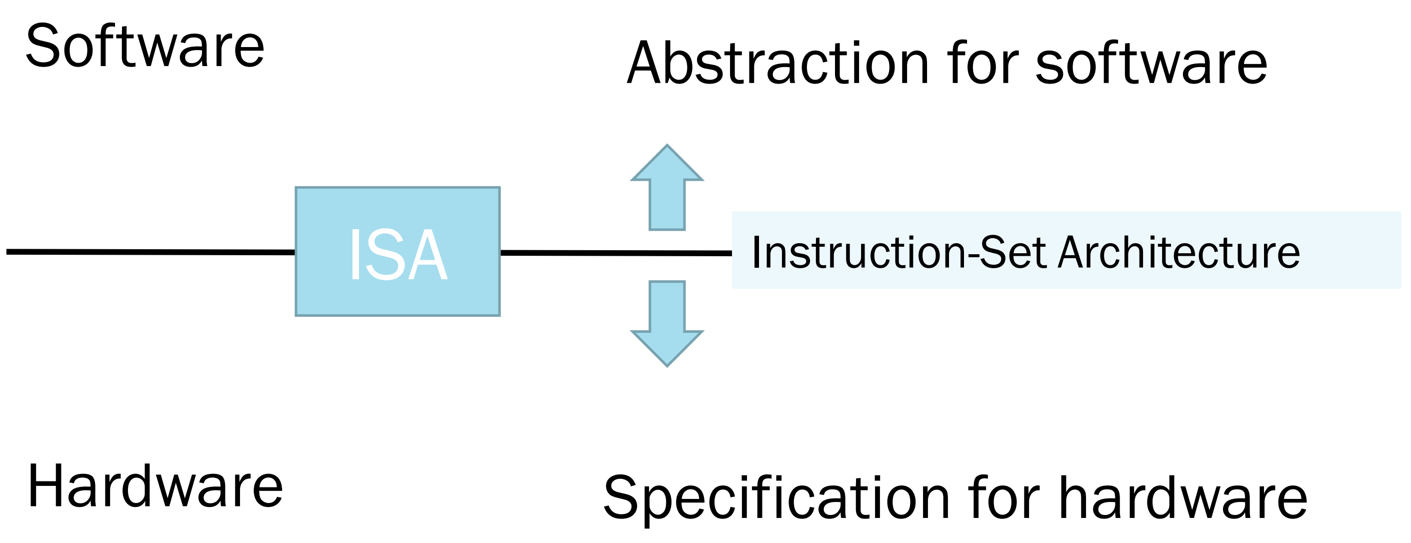
The new architecture builds upon but differs from the designs that have gradually evolved since 1950. The evolution of the computer had included, besides major technological improvements, several important systems concepts and developments:

*The term *architecture* is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flow and controls, the logical design, and the physical implementation.

† Additional details concerning the architecture, engineering design, programming, and application of the IBM System/360 will appear in a series of articles in the *IBM Systems Journal*.

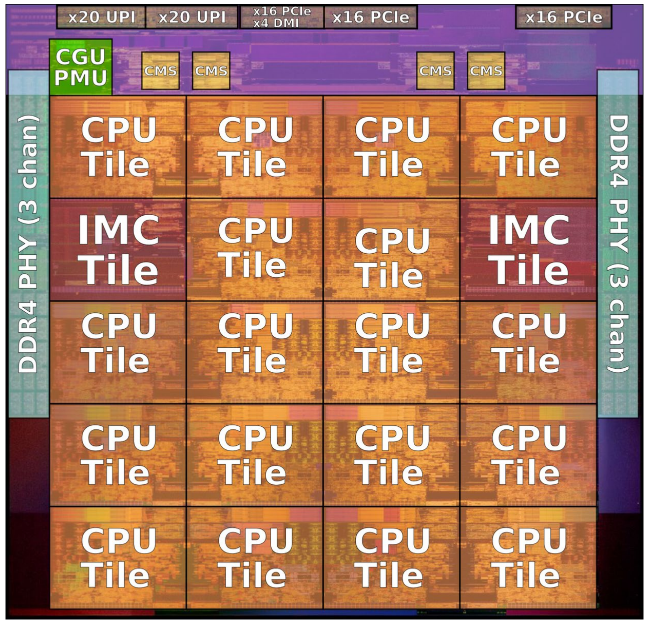
87

IBM JOURNAL • APRIL 1964



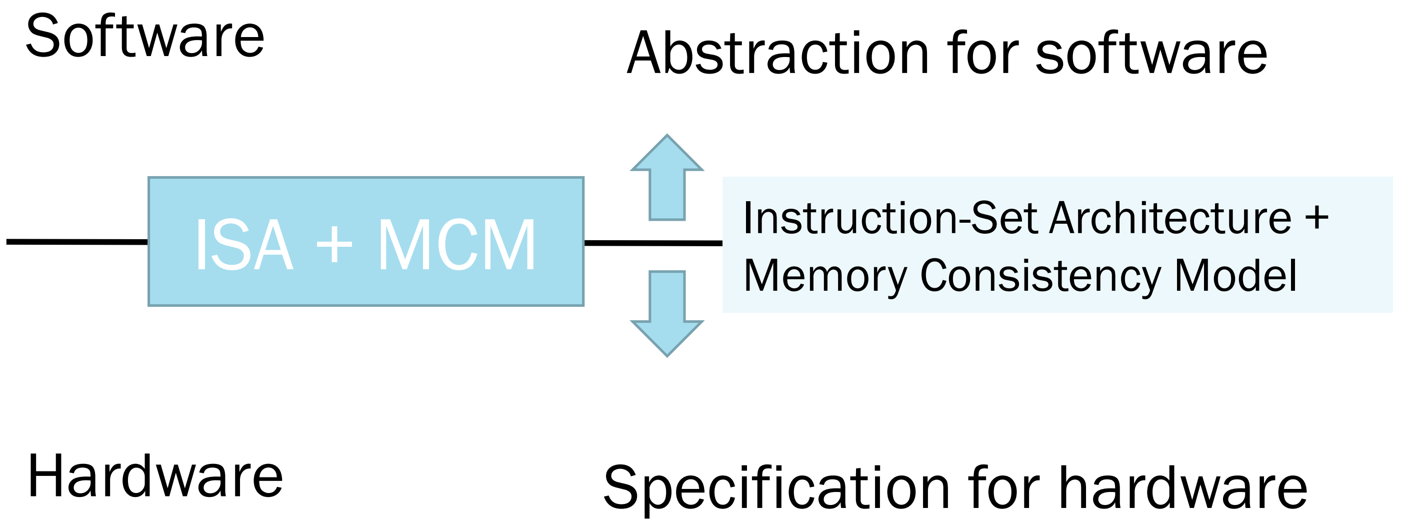
Single-core uniprocessor

Hardware-Software Interface



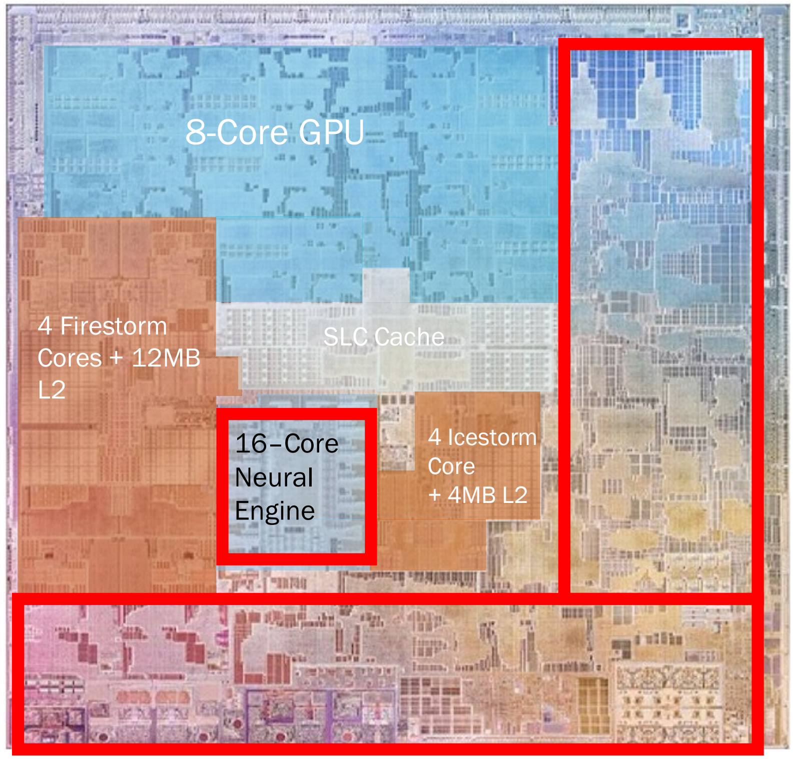
Intel Skylake

Source: wikichip.org



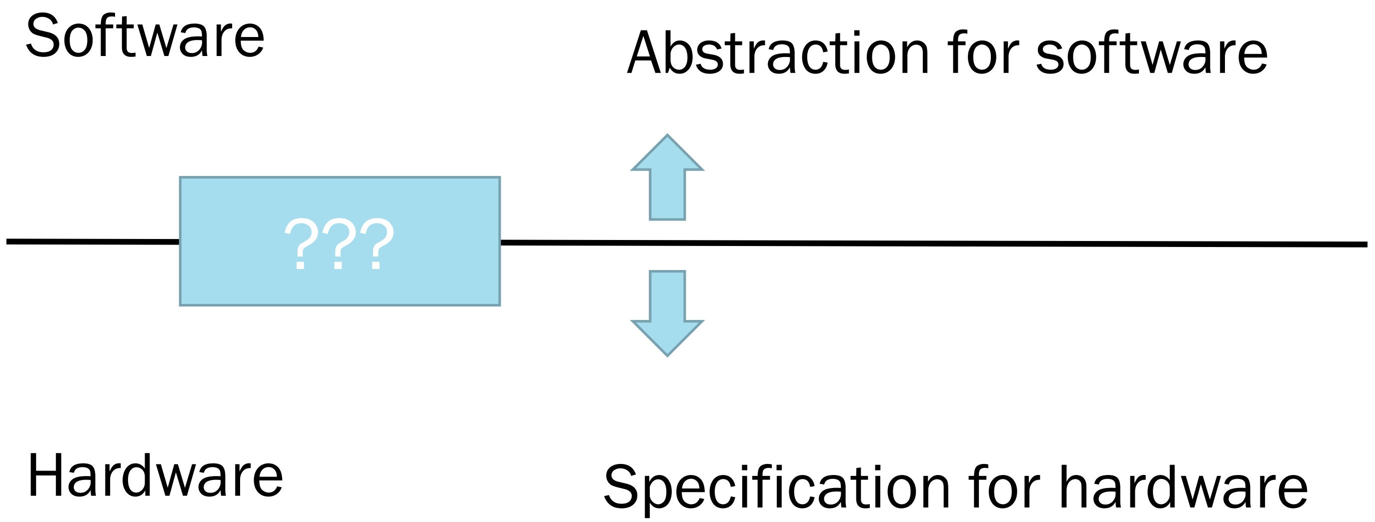
Chip Multiprocessor

Hardware-Software Interface



Apple M1 Die Photo
Source: AnandTech

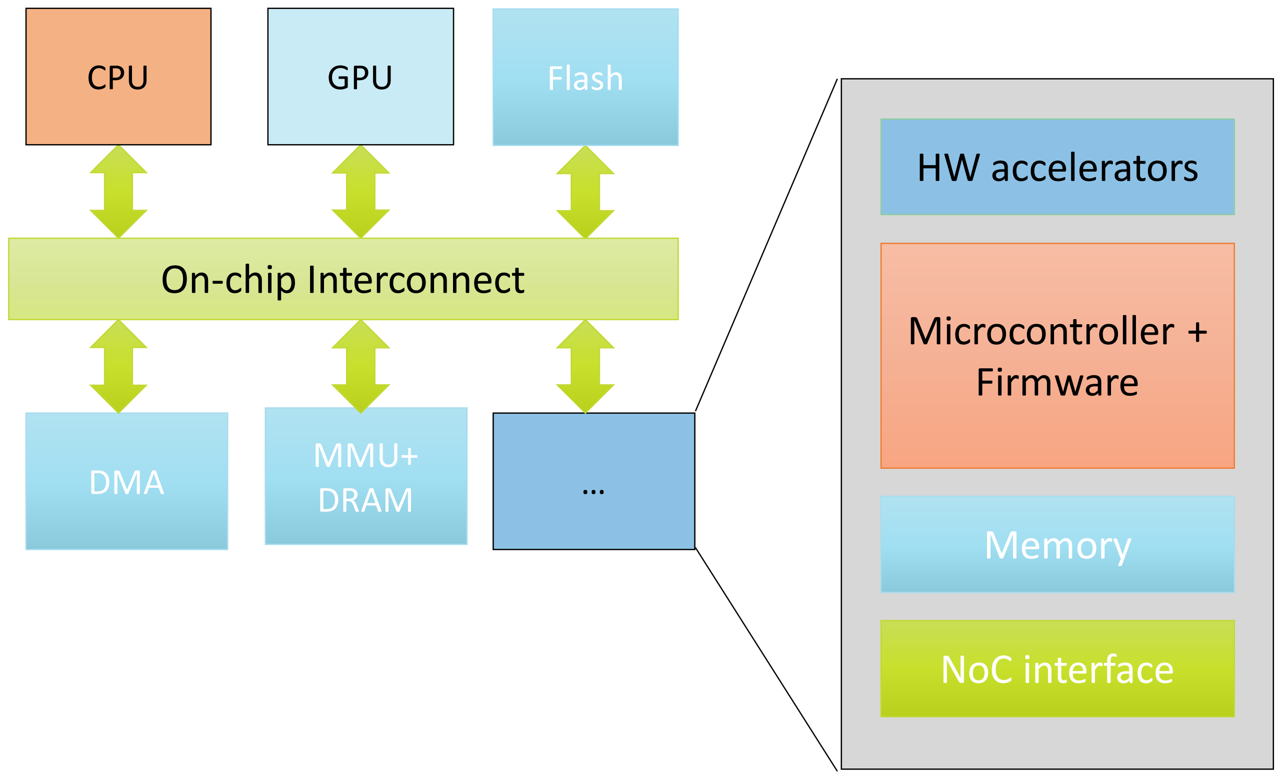
https://www.anandtech.com/show/16226/apple-silicon-m1-a14-deep-dive



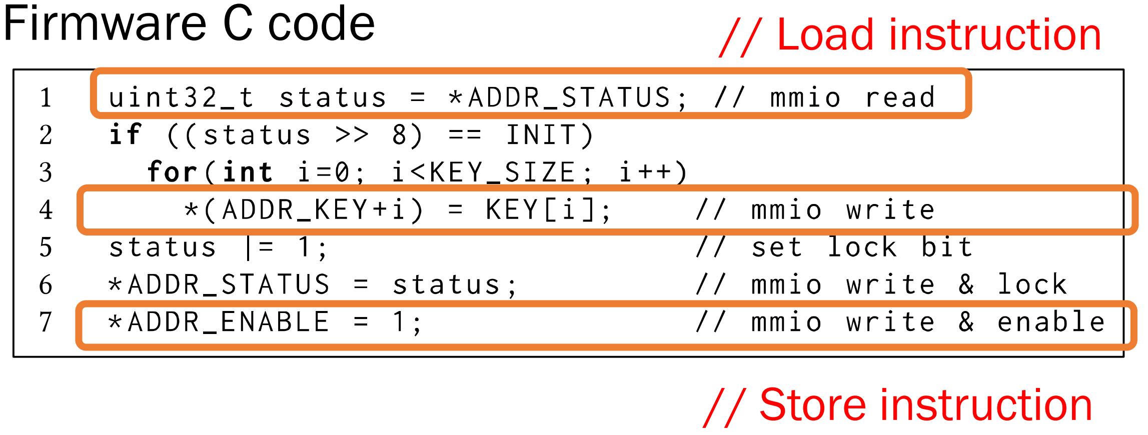
Heterogenous System-on-Chip

Specialized hardware units – aka accelerators
Software/firmware accessed/invoked

Accelerator Interface



Firmware C code



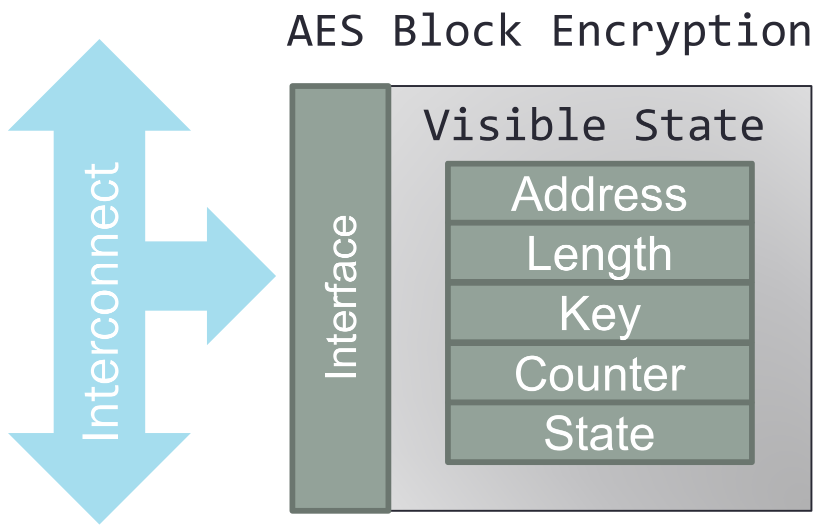
```
uint32_t status = *ADDR_STATUS; // mmio read
if ((status >> 8) == INIT)
  for(int i=0; i<KEY_SIZE; i++)
    *(ADDR_KEY+i) = KEY[i];     // mmio write
status |= 1;                    // set lock bit
*ADDR_STATUS = status;          // mmio write & lock
*ADDR_ENABLE = 1;               // mmio write & enable
```

MMIO Instructions have load/store semantics
Opaque to hardware accelerator semantics

Can we convert MMIO loads/stores to meaningful instruction-level semantics?

Instruction-Level Abstraction (ILA)

Interface Commands $\overset{\text{def}}{=}$ Instructions



MMIO accesses	Commands
Write, 0xff00, 0x1	START_ENCRYPT
Write, 0xff02, data	WRITE_ADDRESS
…	…

- Merits (similar to ISA)
  - Software-visible "architectural" state variables
  - Modular: set of instructions
    - Per-instruction state-update
- Abstraction of the HW as seen at the interface
  - A <u>disciplined</u> lifting of the RTL to the level of software

Generalizes the ISA to include processors and accelerators

Instruction-Level Abstraction (ILA)

```
module Top_rtl (
  clk, rst, interrupt,
  if_axi_rd_r_msg, if_axi_rd_r_rdy,
  if_axi_wr_w_msg, if_axi_wr_w_rdy
  // other AXI interface ports
);

assign and_192 = and_6& (fsm_out[2]);
assign or_117 = (fsm_out[2]) | (fsm_out[5]);

always @(posedge clk or negedge rst_bar) begin
  if (~rst_bar) begin
    addrBound_4_1_equal <= 1’b0;
  end
  else if (run_w_wen & (~while_case_0)) begin
    addrBound_4_1_equal <= addrBound_1_1_tmp_7;
  end
end
```

(a) Accelerator RTL implementation snippet.

```
{
  WR 0x33000100 val (SetWeightAddr val),
  WR 0x33000120 val (SetTensorSize val),
  WR 0x33000130 val (SetLSTMConfig val),
  WR 0x33000200 0x1 (StartLSTM), ...
}
```

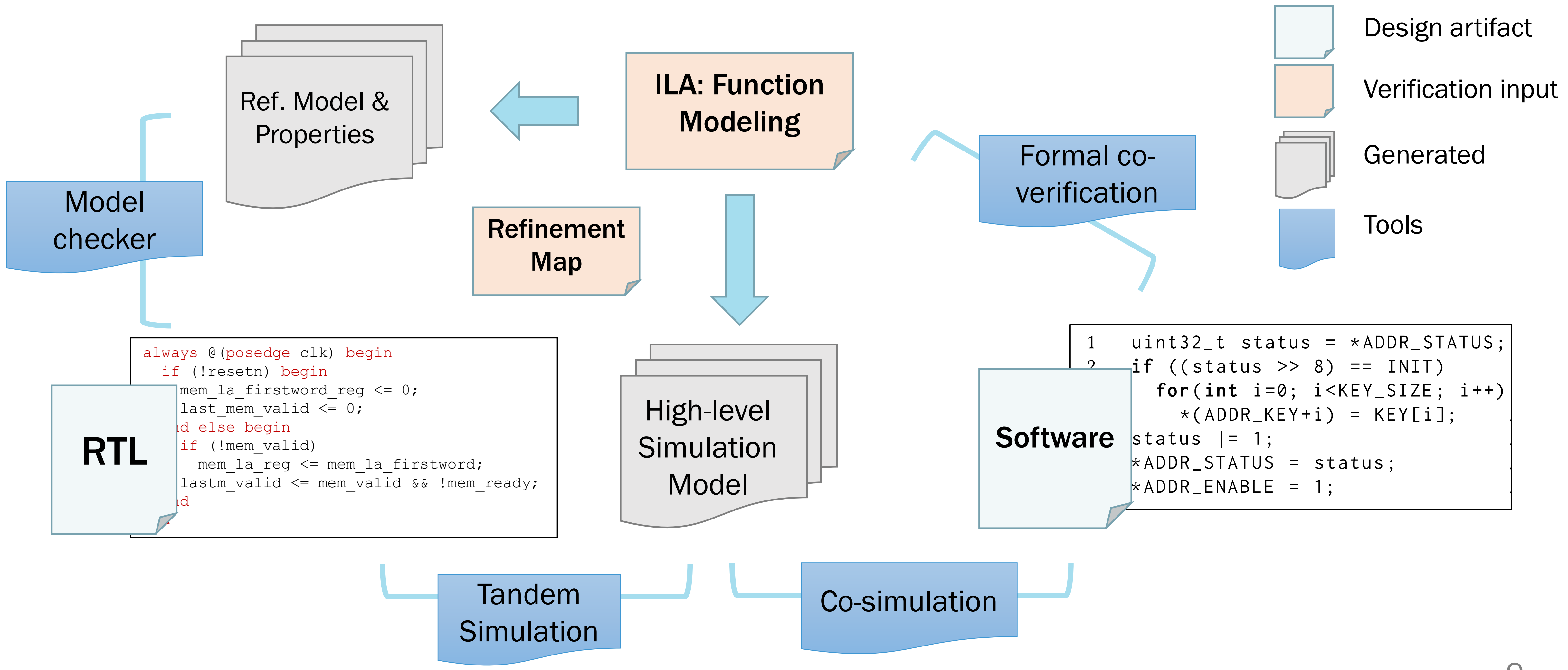
(b) Accelerator instruction set (partial). Each HW operation triggered by an MMIO access is modeled as an individual instruction.

```
SetWeightAddr   0xff100100
SetDataAddr     0xff100200
SetOutputAddr   0xff100300
SetTensorSize   0x20
SetNumTimestep  0x10
SetLSTMConfig   0x02cd87f9
StartLSTM
```

(c) A sequence of accelerator instructions performing an LSTM layer.

FlexASR Accelerator (Speech Recognition)
Harvard Wei Group

ILA Verification Framework



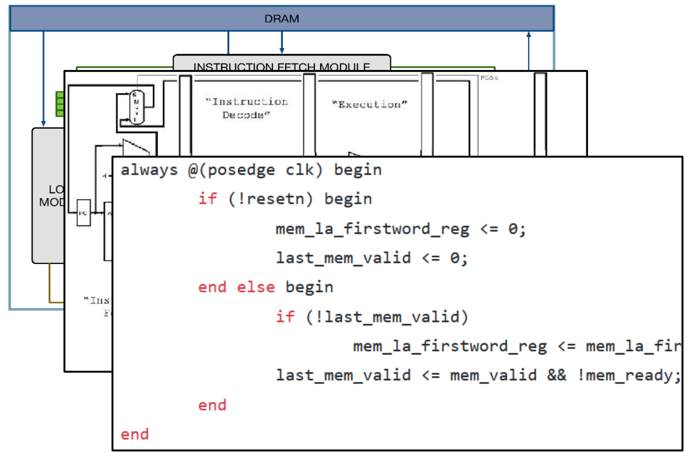
```
always @(posedge clk) begin
  if (!resetn) begin
    mem_la_firstword_reg <= 0;
    last_mem_valid <= 0;
   d else begin
    if (!mem_valid)
      mem_la_reg <= mem_la_firstword;
    lastm_valid <= mem_valid && !mem_ready;
   d
```

```
uint32_t status = *ADDR_STATUS;
if ((status >> 8) == INIT)
  for(int i=0; i<KEY_SIZE; i++)
    *(ADDR_KEY+i) = KEY[i];
status |= 1;
*ADDR_STATUS = status;
*ADDR_ENABLE = 1;
```

Application of ILA: Hardware Verification

- Formal verification of RTL implementation

Processor or accelerator RTL



```
always @(posedge clk) begin
        if (!resetn) begin
                mem_la_firstword_reg <= 0;
                last_mem_valid <= 0;
        end else begin
                if (!last_mem_valid)
                        mem_la_firstword_reg <= mem_la_fir
                last_mem_valid <= mem_valid && !mem_ready;
        end
end
```

Refinement Map

ILA specification

Add: $r_d = r_{s1} + r_{s2}$ pc+=4, …	SetLength: if !busy: length=axi_wdata
Jump: pc= r_{s1}+imm, …	SetKey: if !busy: key=axi_wdata

(for processors or accelerators)

What to compare

```
"state_mapping": {
  "aes_address" : "RTL.aes_reg_opaddr_i.reg_out",
  "aes_length"  : "RTL.aes_reg_oplen_i.reg_out",
  … }
```

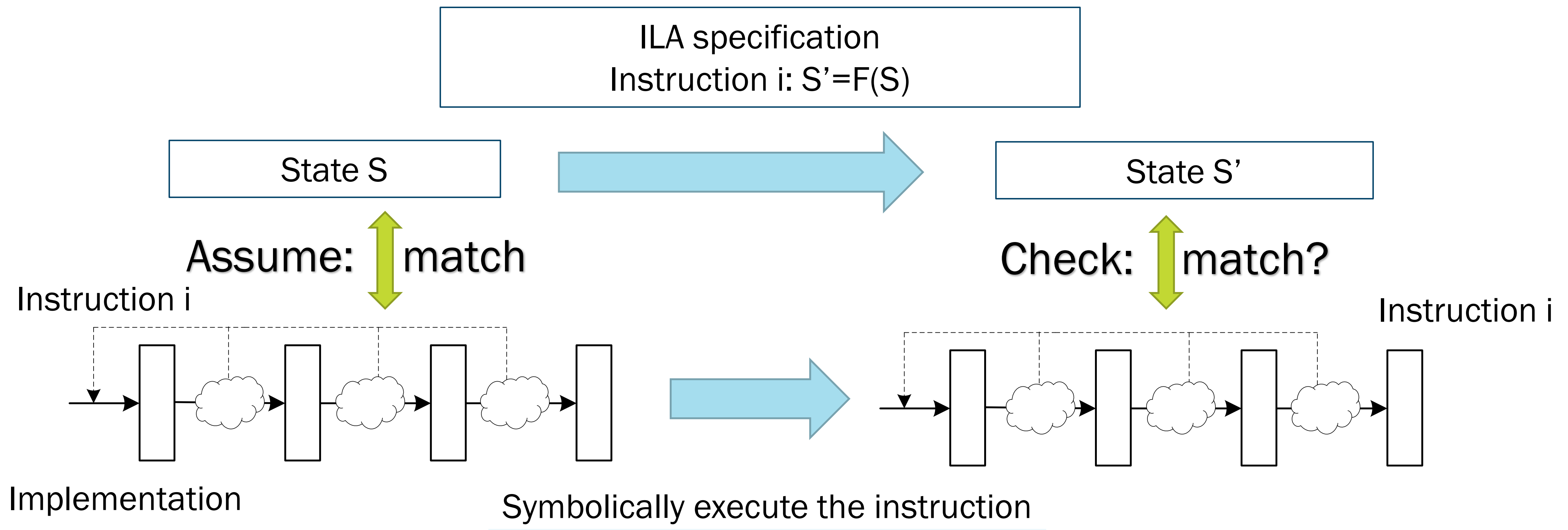
When to compare

```
"instructions": [{
    "instruction"  : "WR_ADDR",
    "ready_signal" : "RTL.xram_ack_delay_1",
    "max_bound"    : 20  }, …]
```

Application of ILA: Hardware Verification (cont'd)

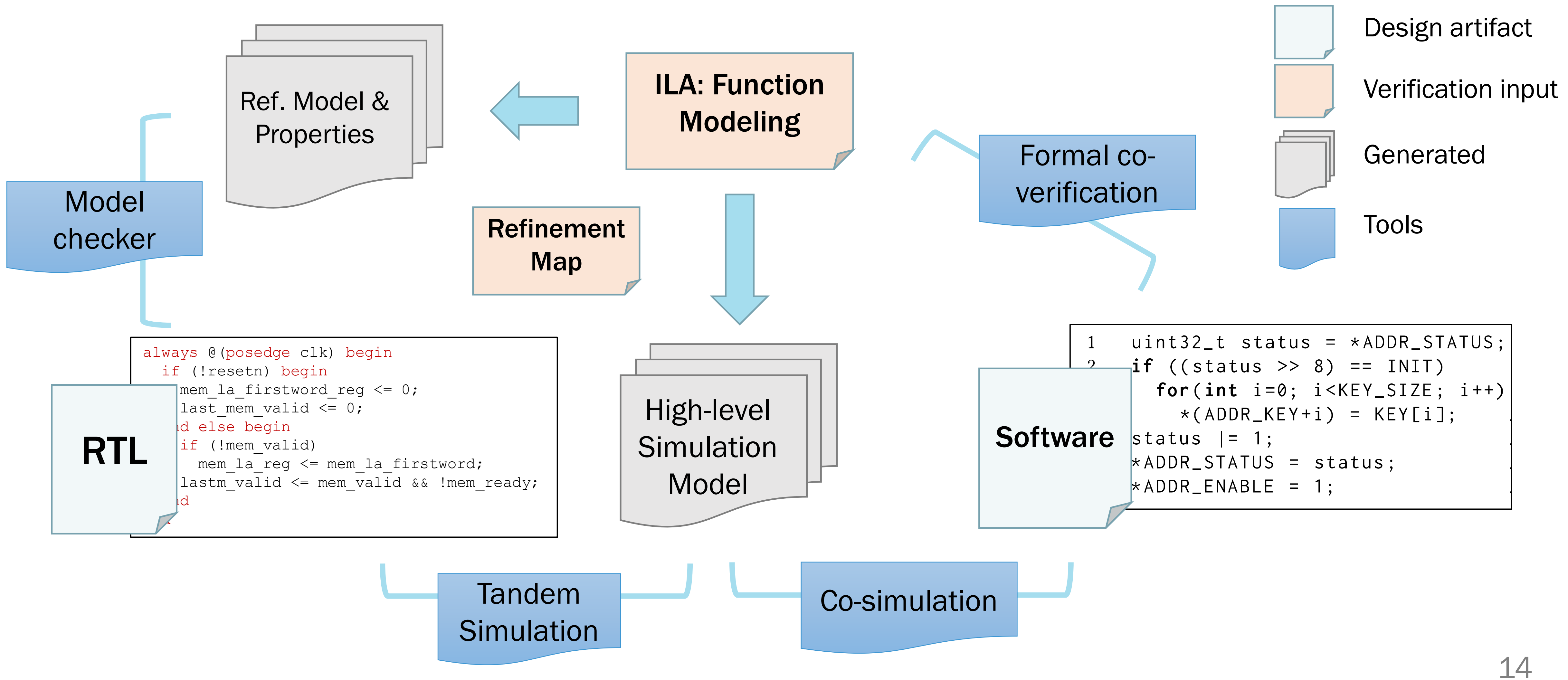
- Formal verification of RTL implementation
  - Auto-generate complete formal properties for each instruction

Contrast with ad-hoc set of properties



Use standard property checkers

ILA Verification Framework

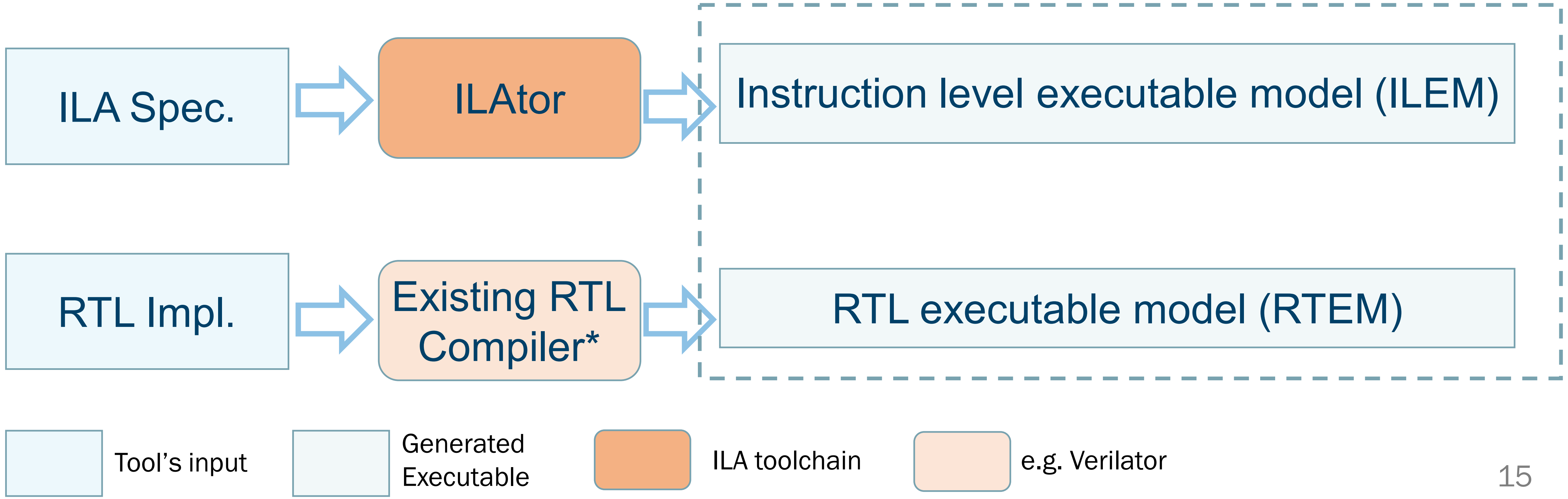


```
always @(posedge clk) begin
  if (!resetn) begin
    mem_la_firstword_reg <= 0;
    last_mem_valid <= 0;
  d else begin
    if (!mem_valid)
      mem_la_reg <= mem_la_firstword;
    lastm_valid <= mem_valid && !mem_ready;
  d
```

```
uint32_t status = *ADDR_STATUS;
if ((status >> 8) == INIT)
  for(int i=0; i<KEY_SIZE; i++)
    *(ADDR_KEY+i) = KEY[i];
status |= 1;
*ADDR_STATUS = status;
*ADDR_ENABLE = 1;
```

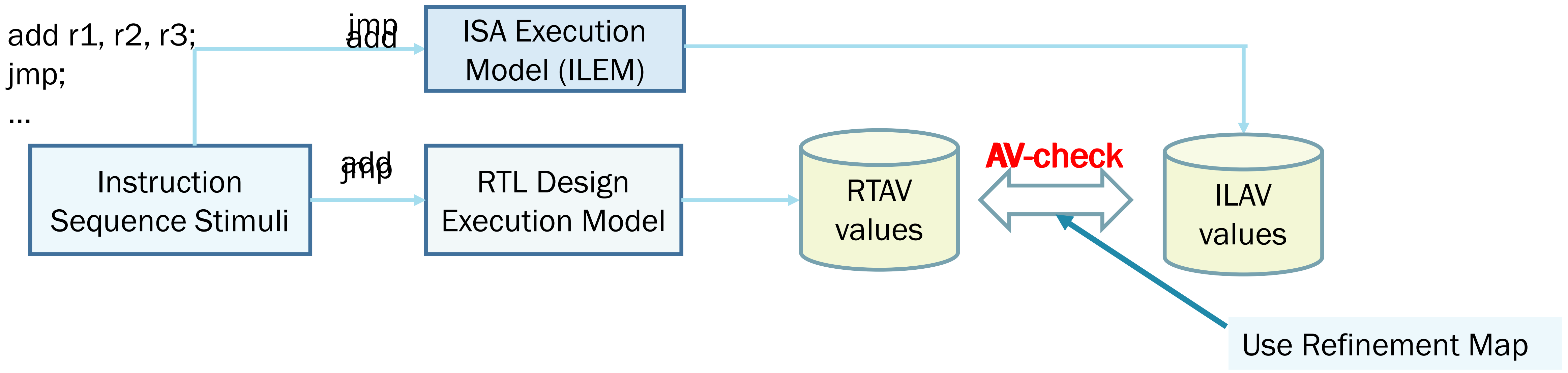
Application of ILA: Hardware Verification (cont'd)

- Simulation-based Validation
  - ILA supports auto-generation of simulation model



Application of ILA: Hardware Verification (cont'd)

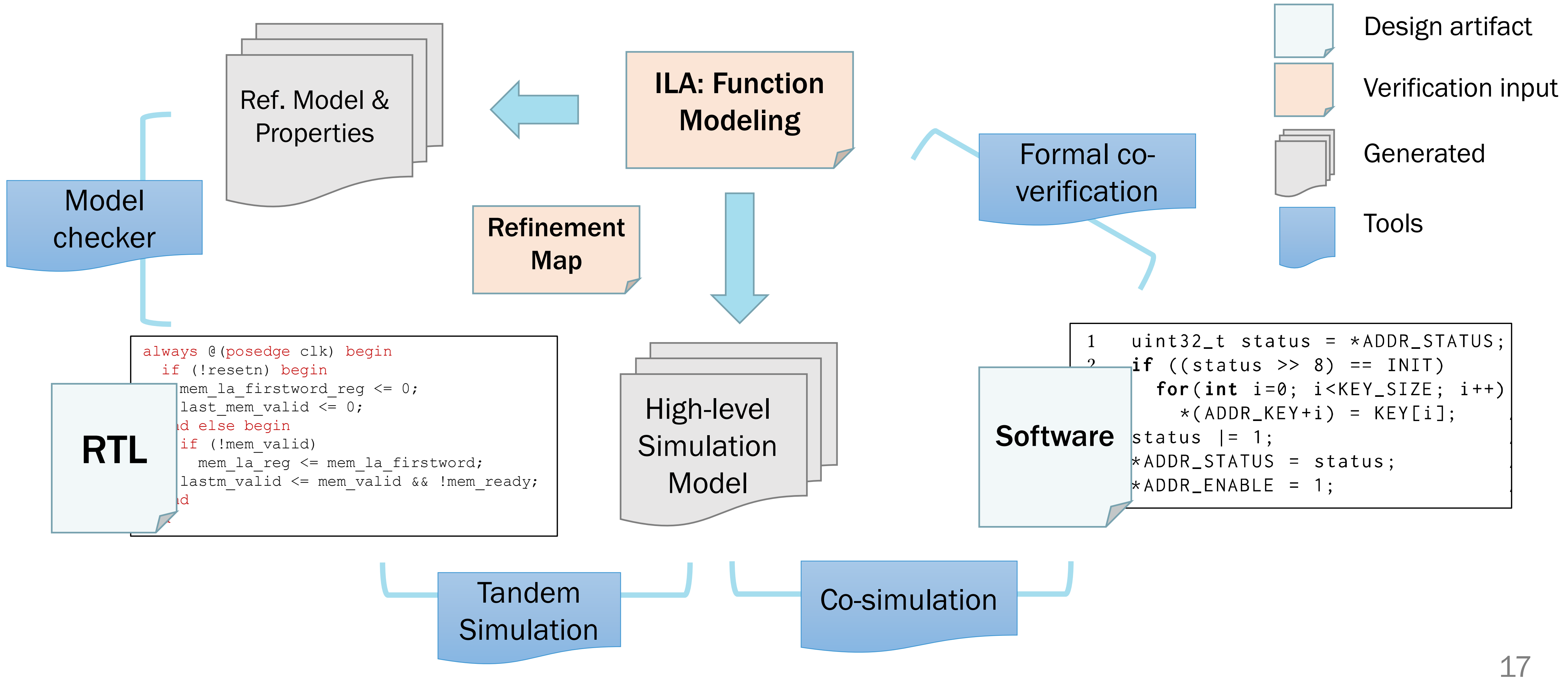
- Simulation-based validation (tandem simulation)
  - After simulating each instruction, check if the RTEM Architectural Variables (RTAV) match ILEM Architectural Variables (ILAV)


add r1, r2, r3;
jmp;
…
jmp
add
ISA Execution Model (ILEM)
add
jmp
Instruction Sequence Stimuli
RTL Design Execution Model
RTAV values
AV-check
ILAV values
Use Refinement Map

Identify bugs right at the instruction that causes AV deviations

Automated - Generalized to Processors + Accelerators

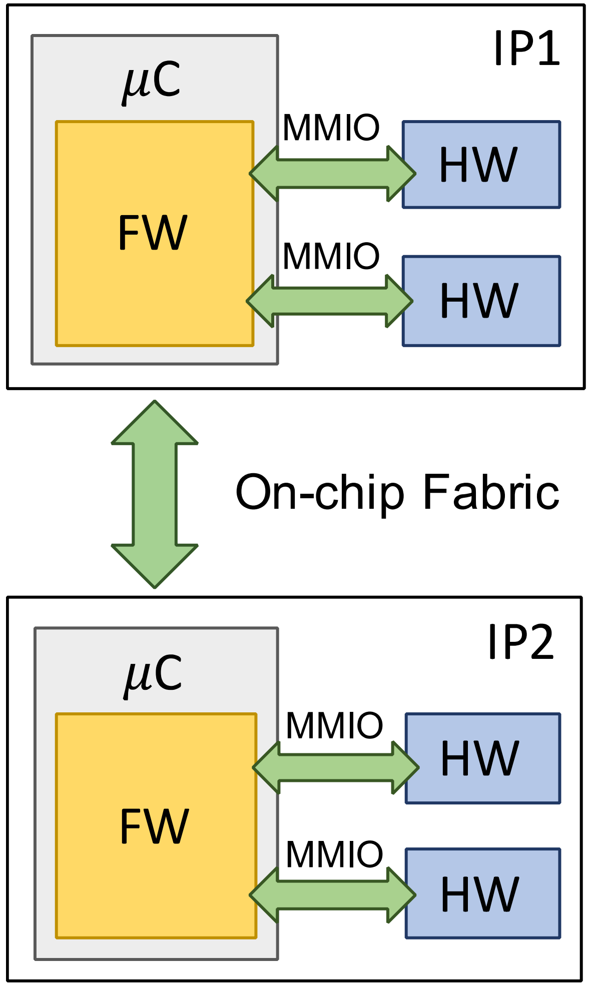
ILA Verification Framework



```
always @(posedge clk) begin
  if (!resetn) begin
    mem_la_firstword_reg <= 0;
    last_mem_valid <= 0;
  d else begin
    if (!mem_valid)
      mem_la_reg <= mem_la_firstword;
    lastm_valid <= mem_valid && !mem_ready;
  d
```

```
uint32_t status = *ADDR_STATUS;
if ((status >> 8) == INIT)
  for(int i=0; i<KEY_SIZE; i++)
    *(ADDR_KEY+i) = KEY[i];
status |= 1;
*ADDR_STATUS = status;
*ADDR_ENABLE = 1;
```

FW/HW co-verification in SoCs



- Communicating (heterogeneous) IPs
  - Processor
  - Firmware
  - Specialized accelerators

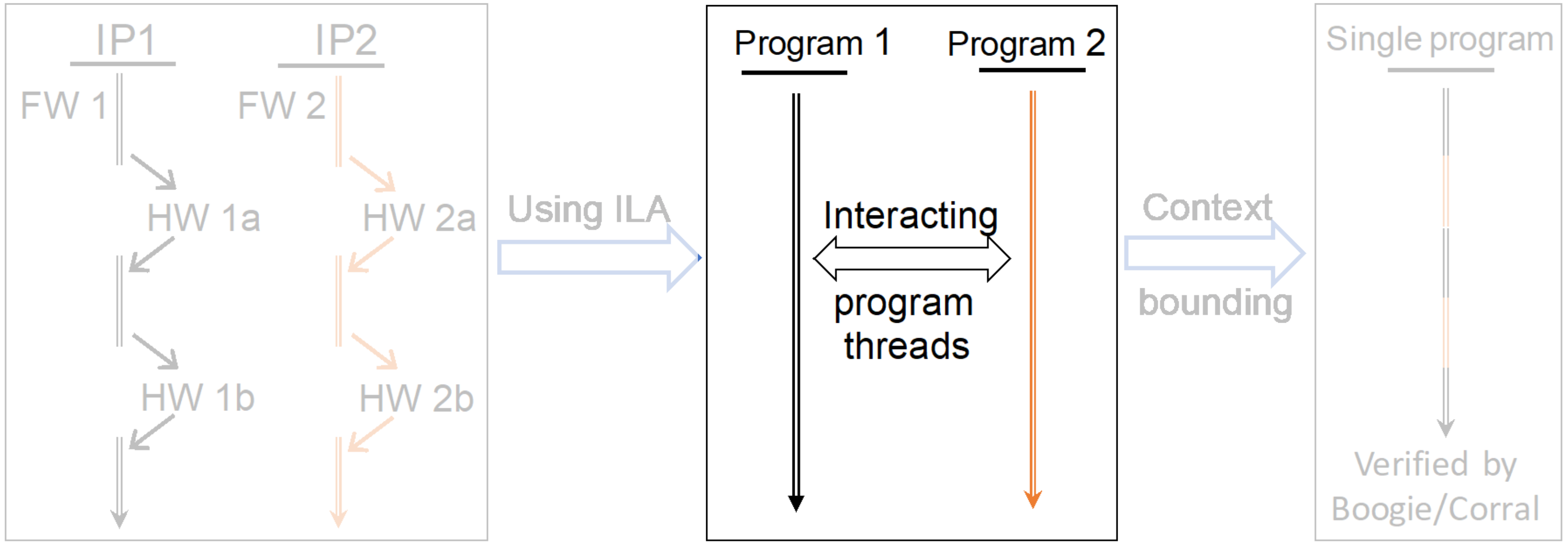
```
uint32_t status = *ADDR_STATUS; // mmio read
if ((status >> 8) == INIT)
  for(int i=0; i<KEY_SIZE; i++)
    *(ADDR_KEY+i) = KEY[i];     // mmio write
status |= 1;                    // set lock bit
*ADDR_STATUS = status;          // mmio write & lock
*ADDR_ENABLE = 1;               // mmio write & enable
```

- FW/HW interaction
  - Hardware functions not captured
  - RTL models not practical (too complex)
  - SW/HW level of abstraction gap

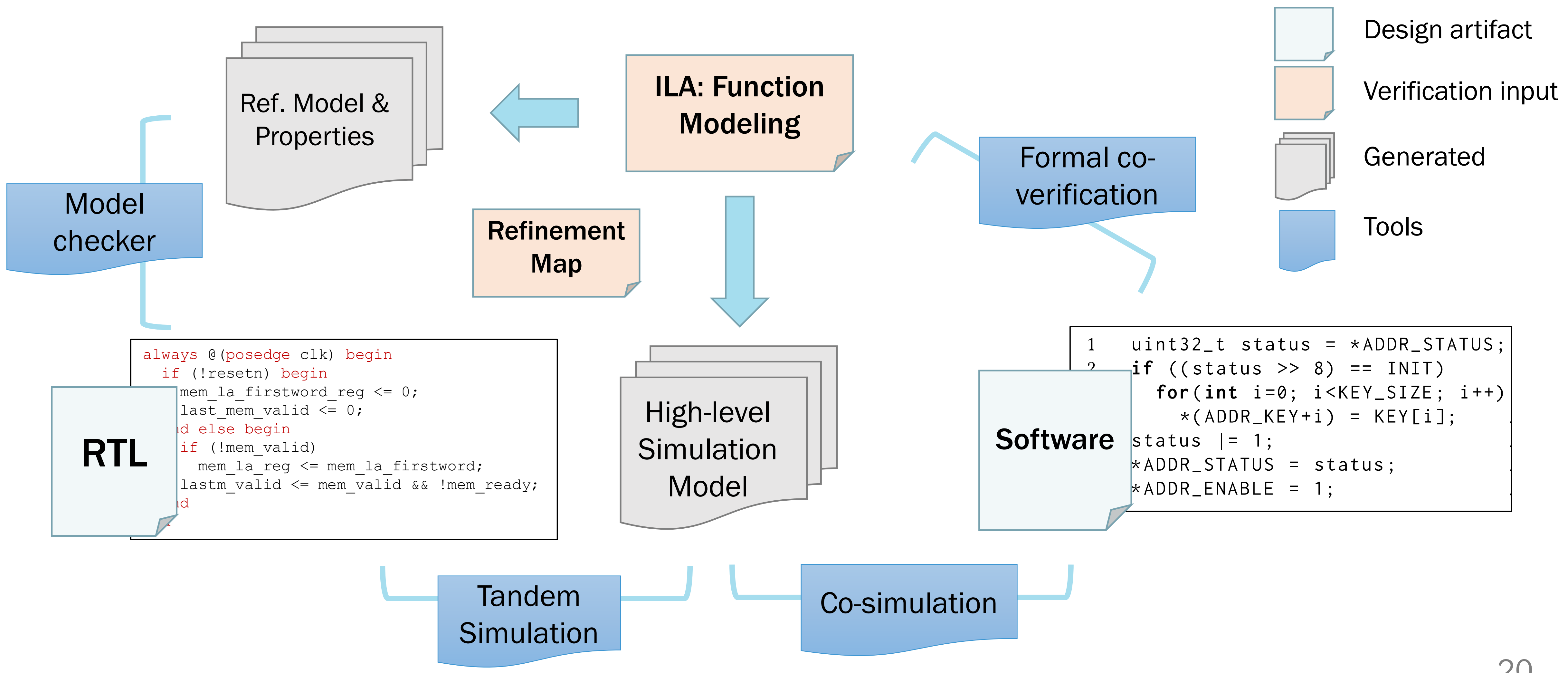
Co-verification methodology

- Modeling
  - ILA for specialized HW
  - Source-level (LLVM) modeling of SW

- Verification
  - Software verification techniques



ILA Verification Framework

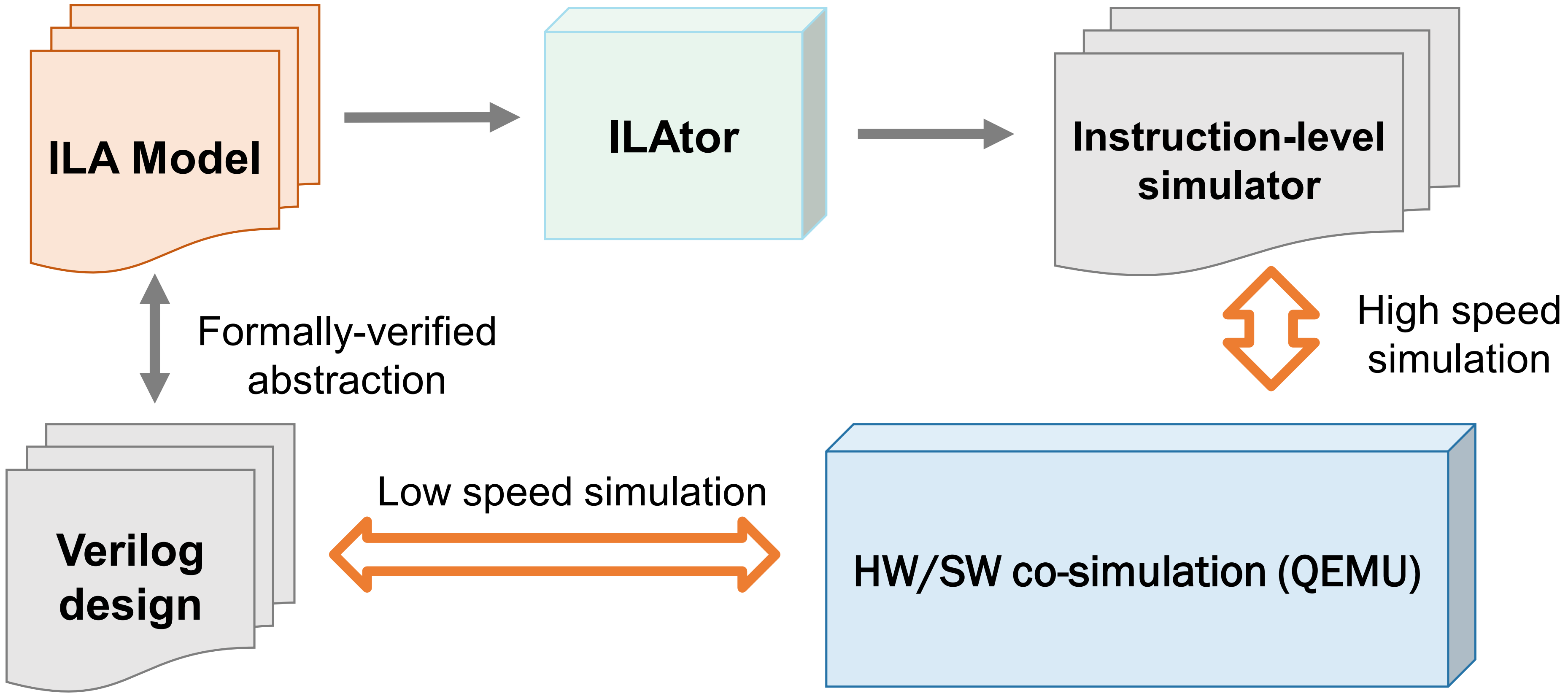


```
always @(posedge clk) begin
  if (!resetn) begin
    mem_la_firstword_reg <= 0;
    last_mem_valid <= 0;
  nd else begin
    if (!mem_valid)
      mem_la_reg <= mem_la_firstword;
    lastm_valid <= mem_valid && !mem_ready;
  nd
```

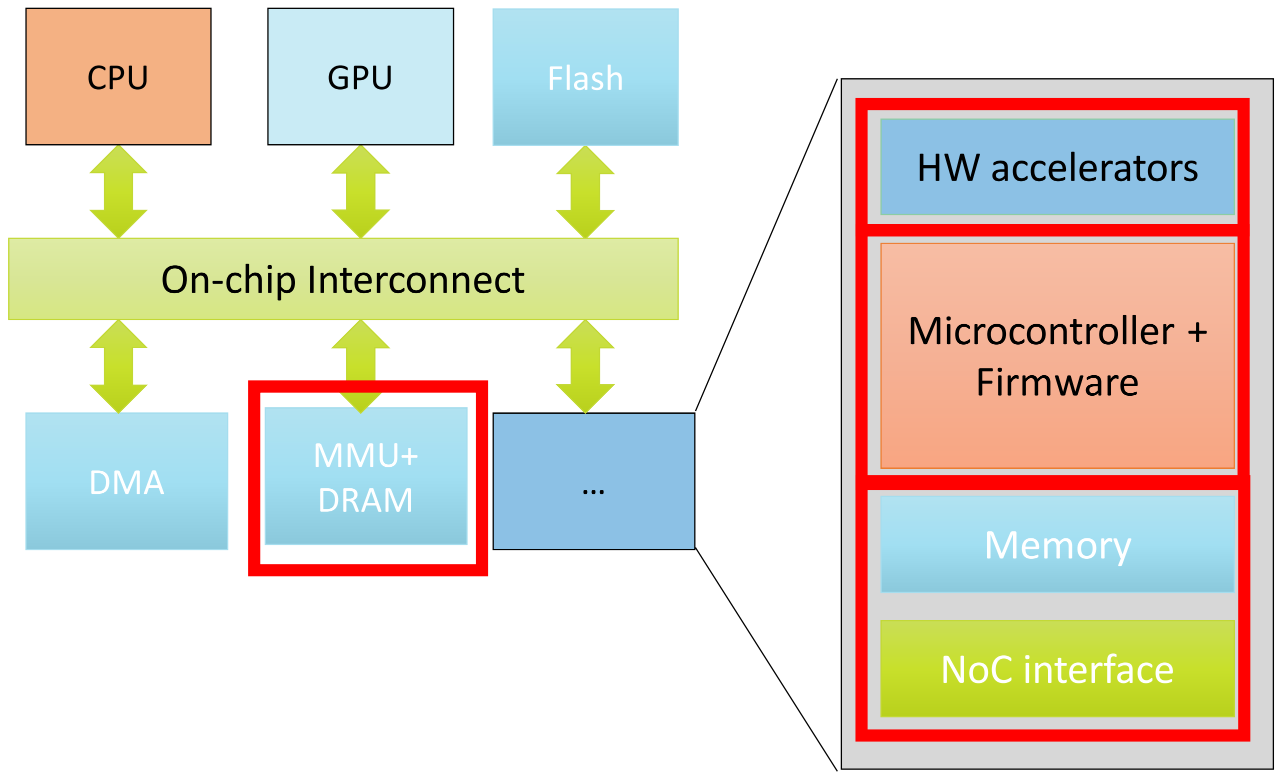
```
uint32_t status = *ADDR_STATUS;
if ((status >> 8) == INIT)
  for(int i=0; i<KEY_SIZE; i++)
    *(ADDR_KEY+i) = KEY[i];
status |= 1;
*ADDR_STATUS = status;
*ADDR_ENABLE = 1;
```

Hardware-Software Co-Verification (cont'd)

- Simulation-based
  - hardware-software co-simulation using ILA as verified abstraction


ILA Model
ILAtor
Instruction-level simulator
Formally-verified abstraction
High speed simulation
Verilog design
Low speed simulation
HW/SW co-simulation (QEMU)

Summary: ILA Based SoC Verification

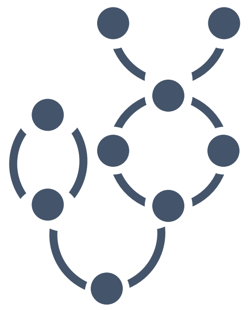


- Accelerator implementation verification
  - Formal
  - Simulation-based
- Firmware hardware co-verification
  - Formal
  - Simulation-based
- Shared memory accesses and memory consistency

ILA-MCM: Memory Consistency Models for Acclerator-rich SoC Platforms

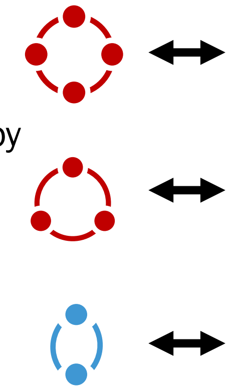
ILA-based Compilation for Accelerators

Applications provided in DSLs → Compiler IR (compute graph) → Verified mapping using ILA Pattern matching & code generation → Programs exploiting custom accelerators → Heterogeneous backends



Relay

1. Provide compiler IR-accelerator mapping by using ILA as a verified lifting.



```
Accel 1 func A:
STR r2, 0xffff0000
LDR r3, 0xffff0010
```

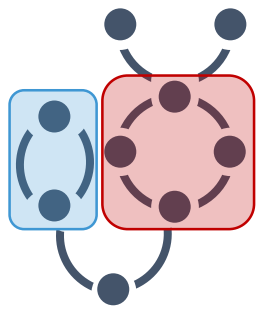
```
Accel 1 func B:
STR r2, 0xffffaa00
LDR r3, 0xffffaabb
```

```
Accel 2 func C:
STR r4, 0xffff0100
LDR r5, 0xffff0110
```

2. Verify the mapping correctness.

ILAng

- Simulation-based
- Proof-based



3. Pattern match the compiler IR pattern provided in the mapping.
4. Rewrite compute graph and lower to the MMIO accesses during code generation.

```
; CPU instructions
CMP    r0, r1
SUBGT  r0, r0, r1
BNE    loop
; Invoke accel 1 (MMIO)
STR    r2, 0xffff0000
LDR    r3, 0xffff0010
; Invoke accel 2 (MMIO)
STR    r4, 0xffff0100
LDR    r5, 0xffff0110
; CPU instructions
MOV    r3, r2
SUBGT  r0, r0, r1
B      lr
```

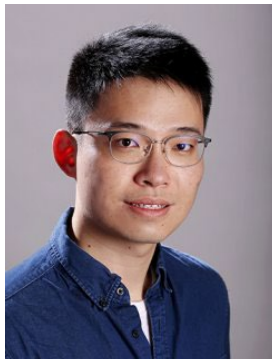
General purpose:
- RISC-V
- ARM
- x86

Application specific:
- VTA
- HLSCNN
- FlexASR
- NVDLA

Compiler Team (Princeton, UW, Harvard)

Bo-Yuan Huang

Thierry Tambe


Yi Li

Mike He

Gus Smith

Gu-Yeon Wei

Aarti Gupta

Sharad Malik

Zachary Tatlock

The ILAng Framework

GitHub:	https://github.com/PrincetonUniversity/ILAng
Wiki:	https://bo-yuan-huang.gitbook.io/ilang/
Docker:	https://hub.docker.com/r/byhuang/ilang/

Selected Bibliography

Primary Papers

- Generalizing Tandem Simulation: Connecting High-level and RTL Simulation Models [ASPDAC 21]
- ILAng: A Modeling and Verification **Platform** for SoCs using Instruction-Level Abstractions. [TACAS19]
- Integrating **Memory Consistency Models** with Instruction-Level Abstractions for Heterogeneous System-on-Chip Verification. [FMCAD18]
- Formal Security **Verification of Concurrent Firmware** in SoCs using Instruction-Level Abstraction for Hardware. [DAC18]
- **Instruction-Level Abstraction (ILA):** A Uniform Specification for System-on-Chip (SoC) Verification. [TODAES18] (Best Paper Award)

Additional Papers

- Leveraging Processor Modeling and Verification for **General Hardware Modules.** [DATE21] (Best Paper Award)
- Generating Architecture-Level Abstractions from RTL Designs for Processors and Accelerators, Part I: **Determining Architectural State Variables** [ICCAD 20]
- **Automatic Generation of Architecture-Level Models from RTL** Designs for Processors and Accelerators [DATE 22]