From Hardware Designs to TLA+ Specifications for Timing Analysis of Real-Time Systems

Samira Ait Bensaid,[1] Mihail Asavoae,[1] Mathieu Jan,[1] Farhat Thabet[1]

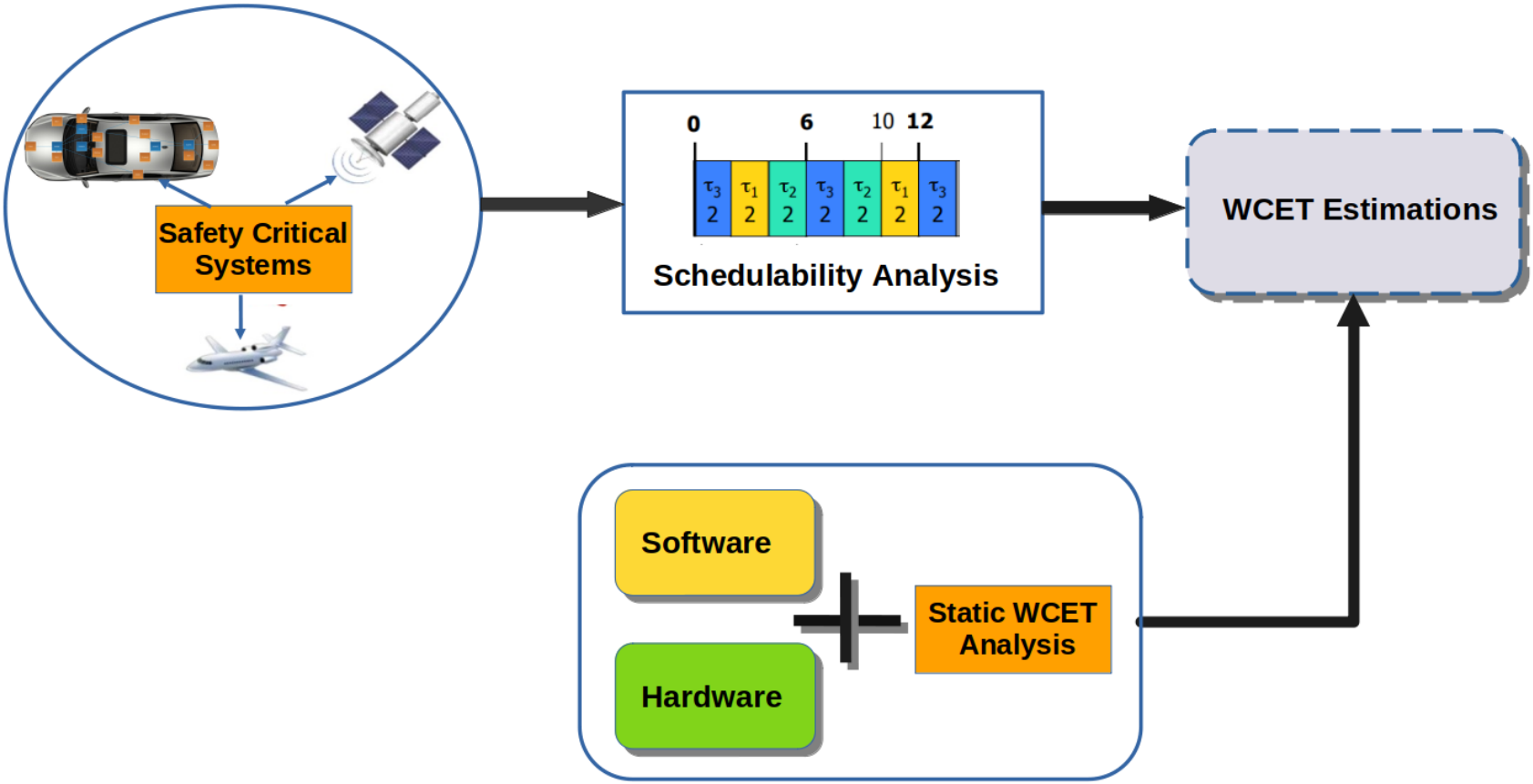
[1]Université Paris-Saclay, CEA-List, Palaiseau, France

TLA+ Community Event 2023
April 22

Outline

Context and Motivation

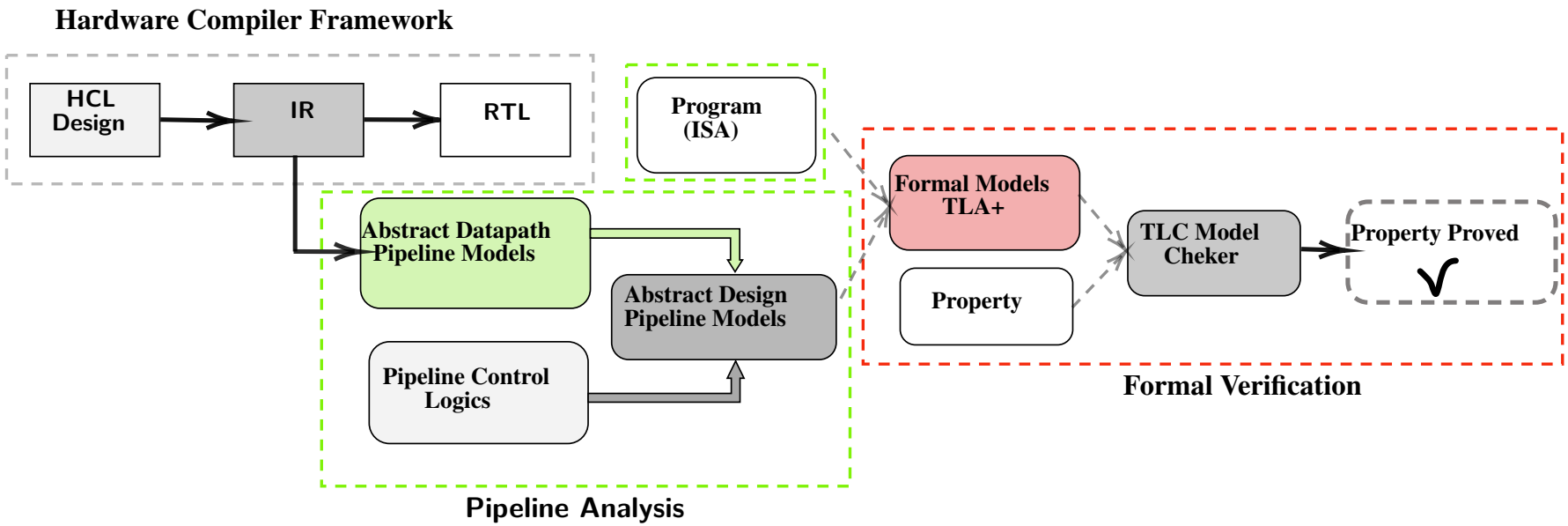
- Safety critical systems: ensure temporal correctness, estimate **Worst-Case Execution Time (WCET)**



Formal Verification of Timing Properties Workflow

Goal: Generating hardware **architecture models** for **formal verification** of timing properties

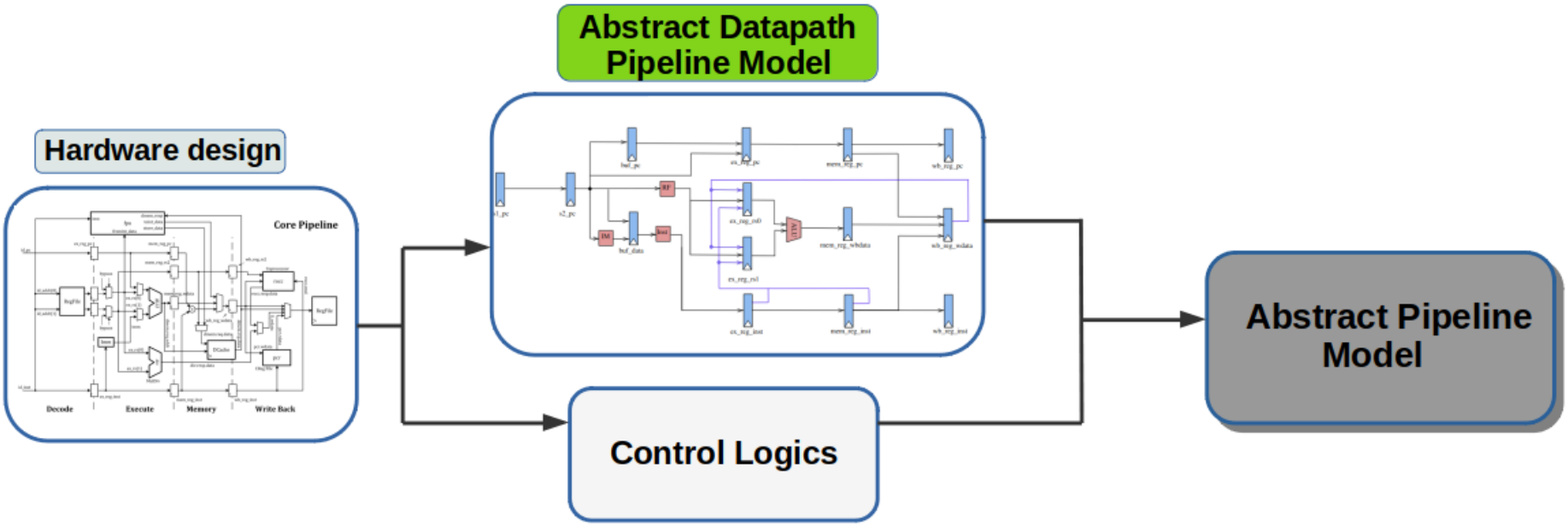
- Automation construction of pipeline models
- Timing properties: timing predictability



- **Challenges**
  - How to construct the formal pipeline models
  - How to formally verify timing properties [AHJ18]

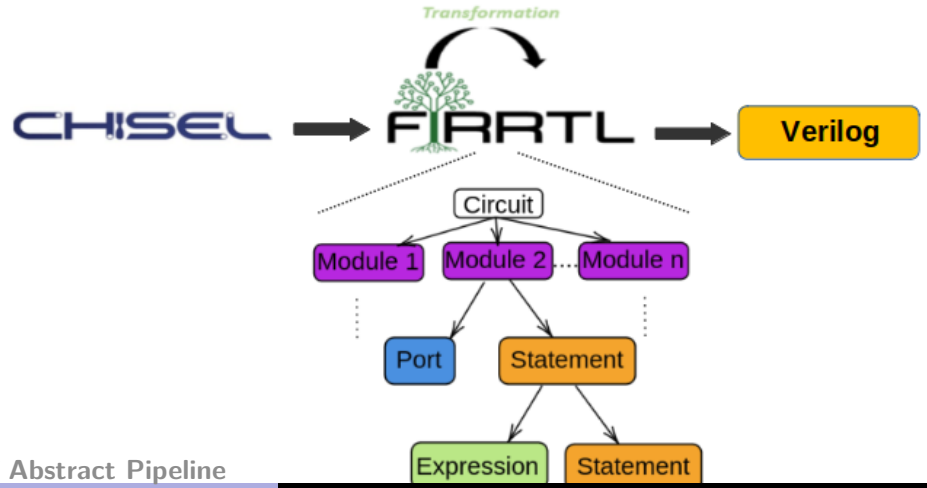
Outline

Automated Construction of Abstract Pipeline Models for Timing Analysis



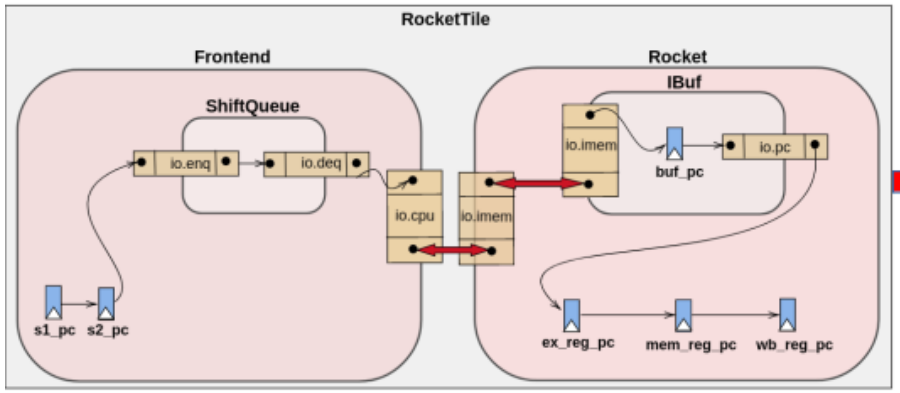
How

- Open source high-level hardware designs for code analysis
- Hardware compilation framework Chisel/FIRRTL to integrate custom passes



Case Study: RISC-V Rocket Processor

Chisel



FIRRTL

```
module RocketTile:
 inst frontend of Frontend
 inst core of Rocket
 core.io.imem.resp.bits.pc <=
       frontend.io.cpu.resp.bits.pc
---------------
module Rocket:
 inst ibuf of IBuf
 reg ex_reg_pc, mem_reg_pc, wb_reg_pc,
     ↪ ex_reg_inst: UInt<size>
 when cond1 :
   ex_reg_pc <=ibuf.io.pc
   ex_reg_inst <=id_inst
   /*updates: mem_reg_pc and wb_reg_pc*/
 ibuf.io.imem.bits.pc <=
       io.imem.resp.bits.pc
 ibuf.io.imem.bits.data <=
       io.imem.resp.bits.data
---------------
module IBuf:
 output io : {/*imem and pc fields*/}
 reg buf.pc : UInt<40>
 buf.pc <-io.imem.bits.pc
 node _io_pc_T_1 =
    mux(cond2, buf.pc, io.imem.bits.pc)
 io.pc <=_io_pc_T_1
```

Interface IO

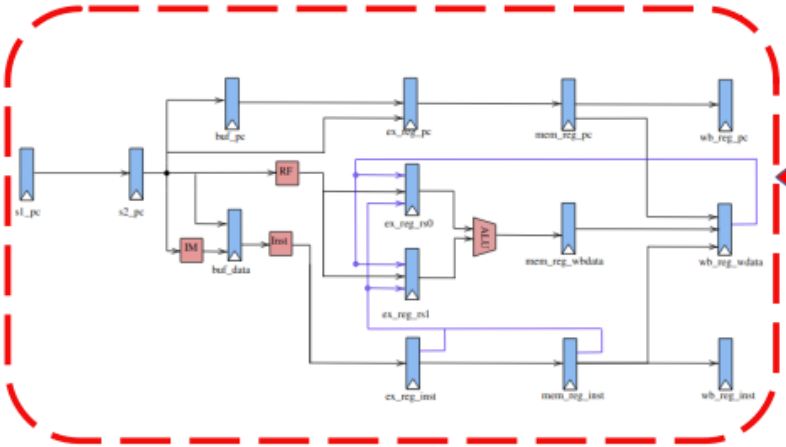
Declaration

Update of sequential logic under "when" context

Output port

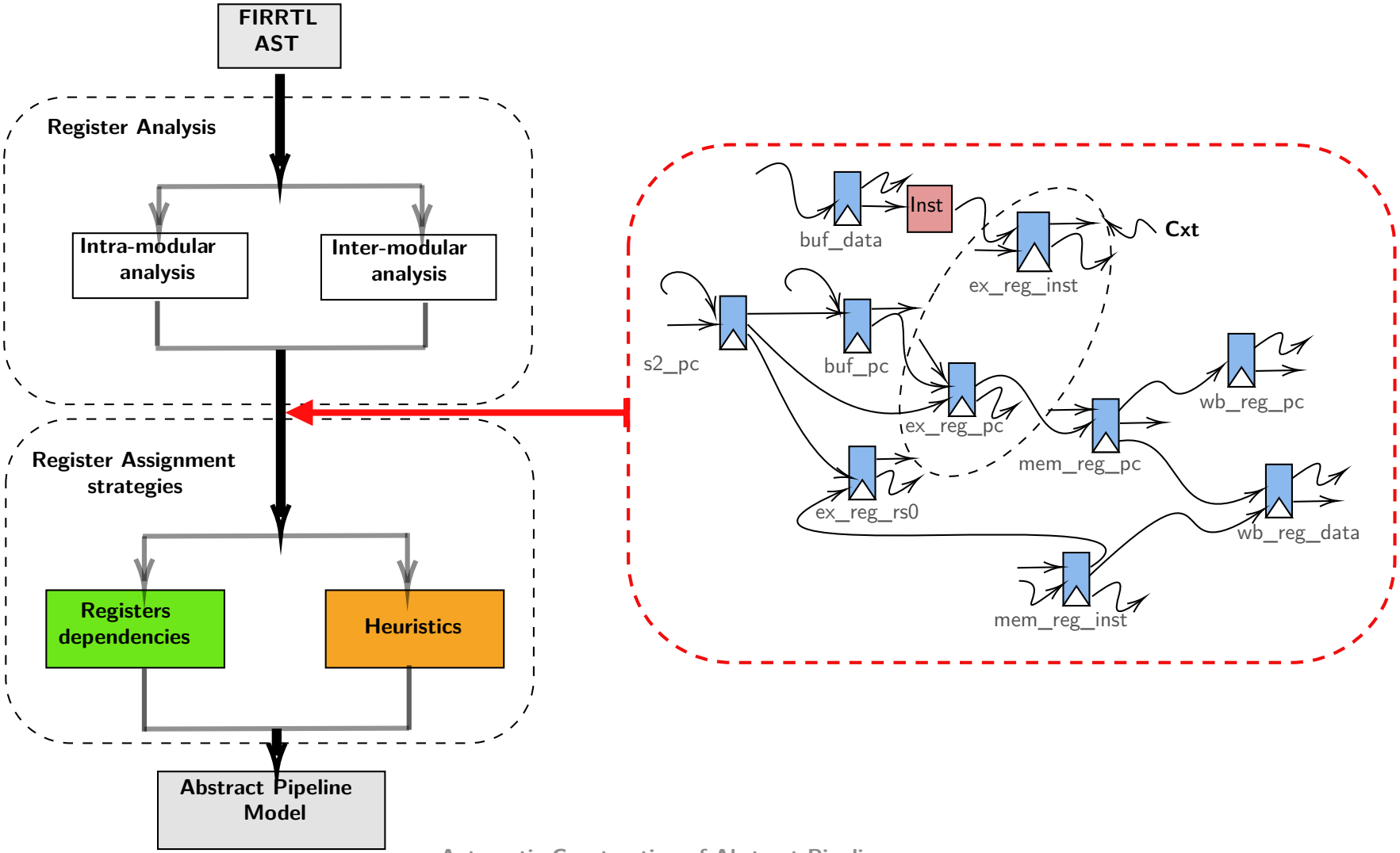
Combinatorial logic

Abstract pipeline model



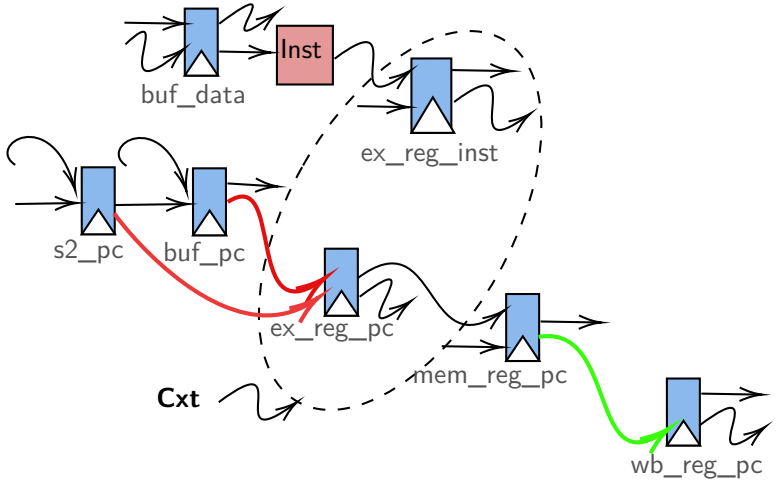
Register Analysis over Rocket Processor

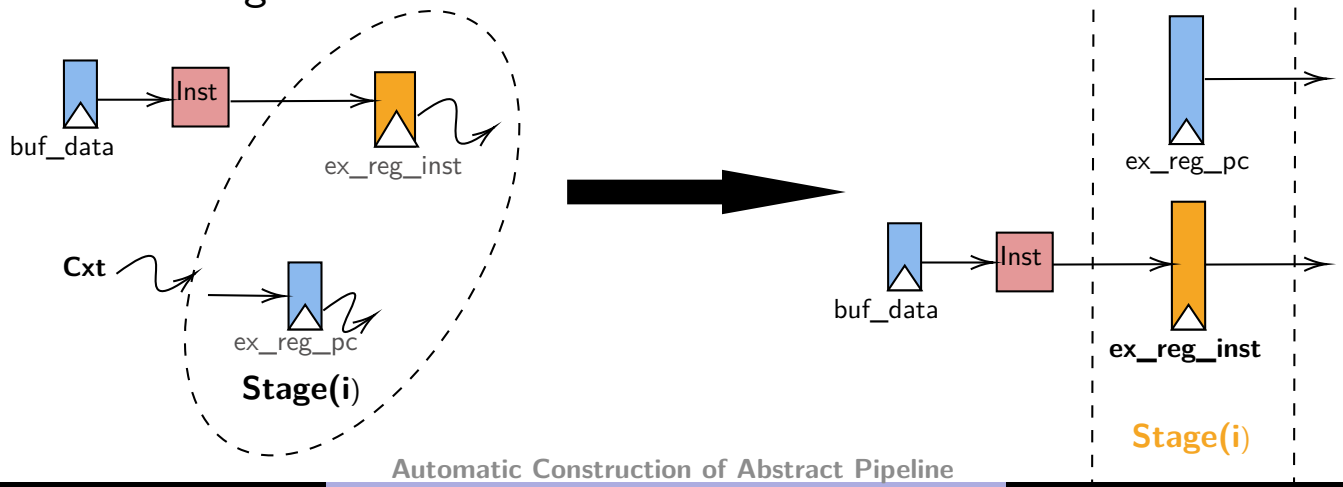
- Register analysis result: non-connected graph with the set of nodes as registers and edges as their dependencies



Register Assignment over Rocket Processor

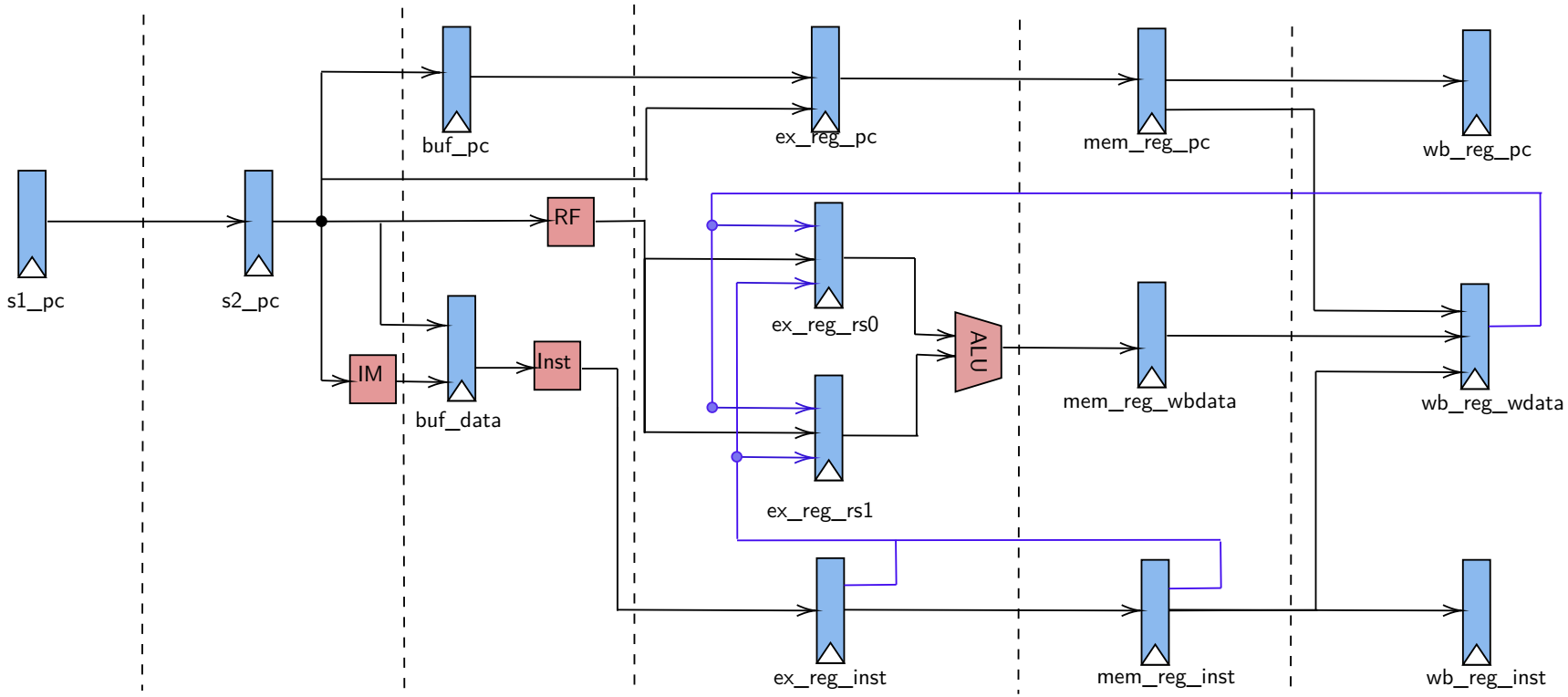
- Register assignment: assign registers to their pipeline stages
  - Assign fetch stage to PC counter register: $\texttt{to\_stage}(\texttt{PC}) = 1$
  - Assign other registers:
    - Case 1: Register dependencies
    - Linear case: one source connection for the register to assign (`wb_reg_pc`)
    - Max: several sources connected to the register to assign (`ex_reg_pc`)
    - Case 2: Heuristic «When condition», assign registers of the same condition context in the same stage





Register Assignment Result

- Results of a subset of registers linked to the PC register: registers assigned in their pipeline stages

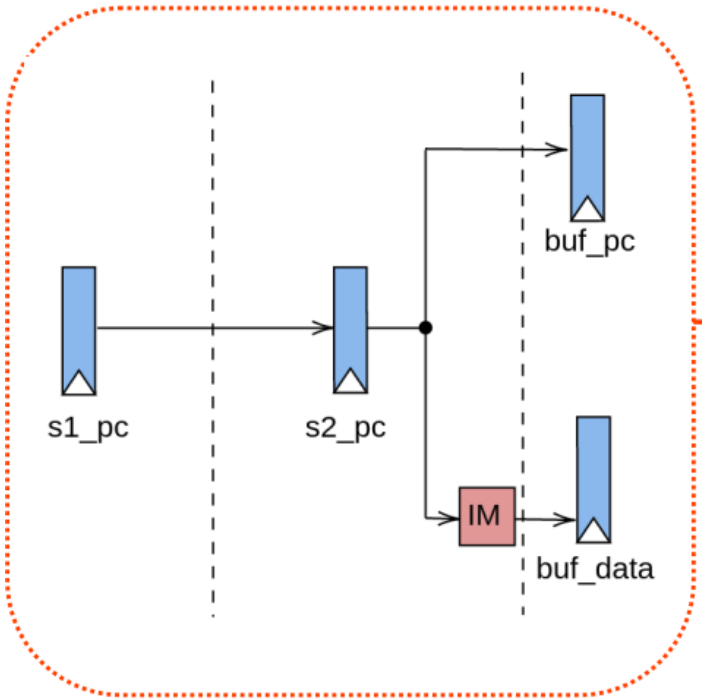


- Pipeline depth: 6 stages

Outline

Specification TLA+: Datapath Formal Model

- **Spec** $\triangleq$ **Init** $\wedge$ [][**Next**]_ << **Variables** >>



Rocket Datapath TLA+ Formal Model

MODULE Rocket

Extends Sequences, instructions, FiniteSets, Integers, TLC

State Variables ← **VARIABLES** currCycle, stage_1, stage_2, stage_3, stage_4, stage_5, stage_6

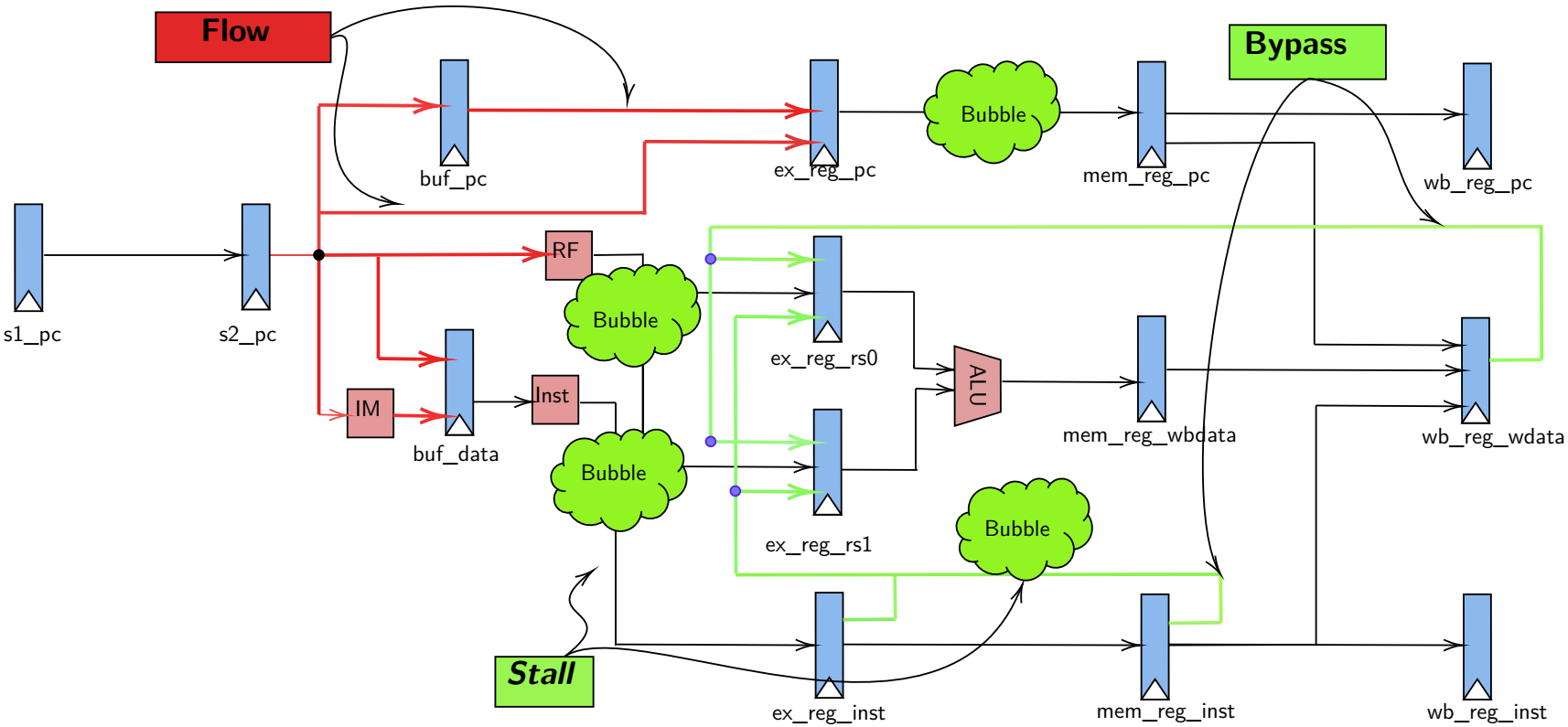
Init $\triangleq$
$\wedge stage\_1 = [s1\_pc |-> empty]$
$\wedge stage\_2 = [s2\_pc |-> empty]$
$\wedge stage\_3 = [buf\_pc |-> empty,$
$\quad buf\_data |-> empty]$
$\wedge currCycle = 0$
$\wedge prog = [rest |-> Program, exec |-> <<>>]$

PipNext $\triangleq$ $LET nxt == next\_instr(prog) IN$
$\wedge stage\_1' = [s1\_pc |-> nxt]$
$\wedge stage\_2' = [s2\_pc |-> stage\_1.s1\_pc]$
$\wedge stage\_3' = [buf\_pc |-> stage\_2.s2\_pc,$
$\quad buf\_data |-> stage\_2.s2\_pc]$
$\wedge currCycle' = currCycle + 1$
$\wedge prog' = [exec |-> nxt, rest |-> rest\_instr(prog)]$

Next $\triangleq$ **PipNext_** << **Variables** >>

Specification TLA+: Formal Control Logics

- Control logics address signals that ensure
  - **Flow** in pipeline stages: instructions patterns
  - **Pipeline status**: stall, bypass ...



Rocket Processor: TLA+ Specification Model

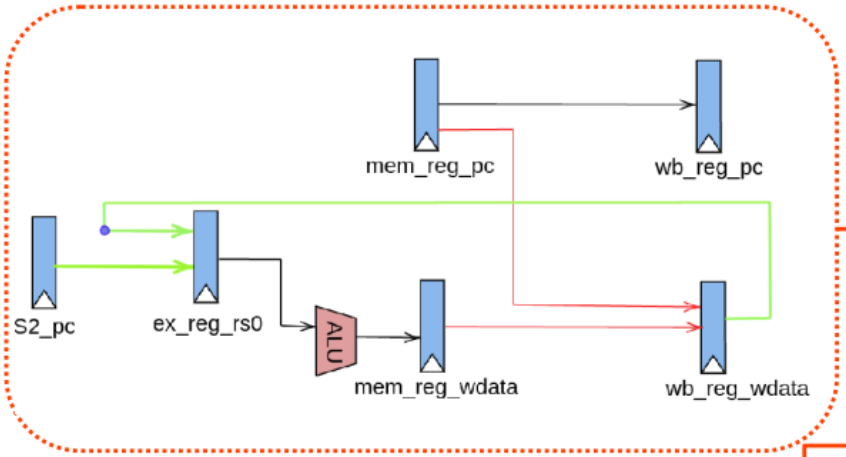
- **Next $\triangleq$ PipNext $\vee$ PipStall**

Instructions Representation

MODULE Instruction

VARIABLES prog

instrs == {[**type**|− > "**lw** − **RR**",
RD|− > "*RD*",
*RS*1|− > "*rs*1",
*RS*2|− > "*rs*2",
conds|− > {}],
[**type**|− > "**bne** − **RR**",
RD|− > "*empty*",
*RS*1|− > "*rs*1",
*RS*2|− > "*rs*2",
conds|− > {"**cond**"}]}



Pipeline status : Stalling logics for memory access

Rocket Hardware TLA+ Formal Model

MODULE Rocket

Extends Sequences, **instructions**, FiniteSets, Integers, TLC
VARIABLES currCycle, **stall_delay**, stage_1, stage_2, stage_3, stage_4, stage_5, stage_6

-bypass condition

bypass == *stage_3.buf_pc.inst.RS1* = *stage_6.wb_reg_pc.inst.RD*

-control condition

stall == *stage_5.mem_reg_pc.inst.type* = "**lw** − **RR**"
∨*stage_5.mem_reg_pc.inst.type* = "**sw** − **RR**"
stall_cond == ∧*stall*
∧*stall_delay* < 5

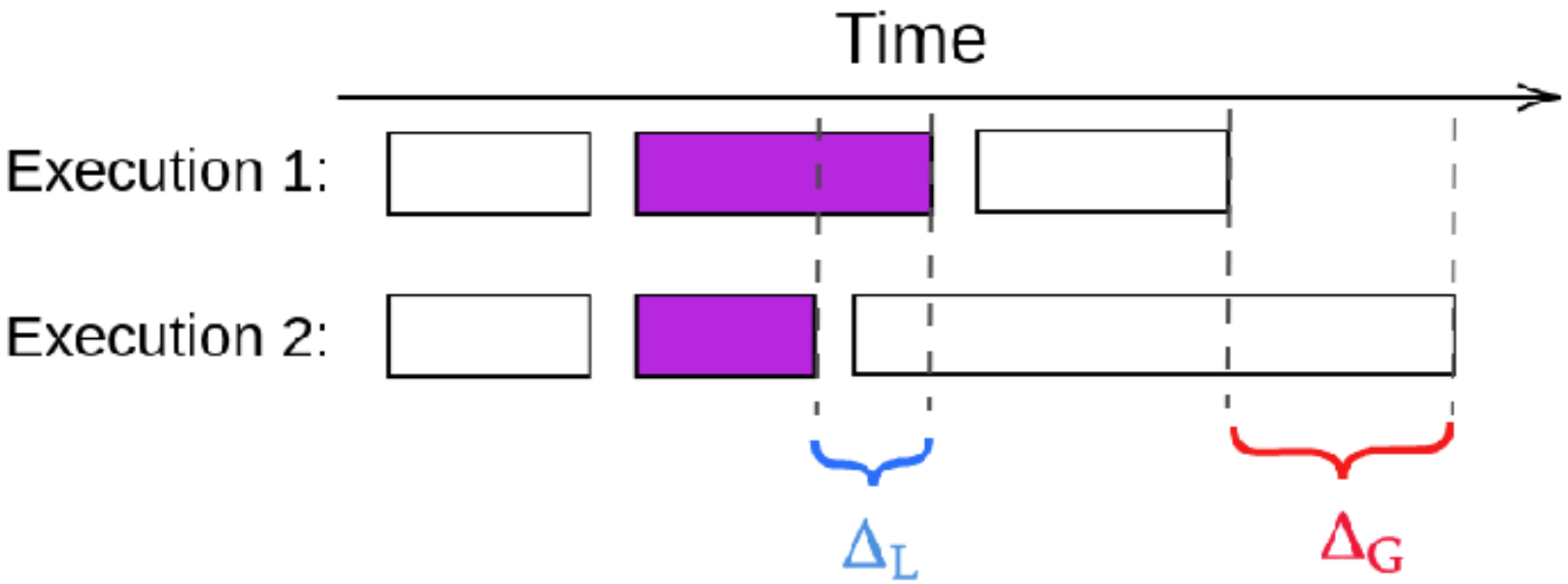
PipNext $\triangleq$ ∧∼ **stall_cond**
∧.....
∧*stage_4′* = [*ex_reg_pc*|− > *stage_3.buf_pc*,
ex_reg_rs0|− >**IF bypass**
THEN *stage_6.wb_reg_wdata*
ELSE *stage_2.s2_pc*]
∧*stage_6′* = [*wb_reg_wdata*|− >**IF cond**
THEN *stage_5.mem_reg_pc*,
ELSE *stage_5.mem_reg_wdata*]
∧*currCycle′* = *currCycle* + 1
∧**stall_delay′** = **0**

PipStall $\triangleq$ **stall_cond**
∧*stage_6′* = [*wb_reg_pc*|− > *empty*,
wb_reg_wdata|− > *empty*]
∧*currCycle′* = *currCycle* + 1
∧**stall_delay′** = **stall_delay** + **1**
∧**UNCHANGED** << *stage_1*, *stage_2*, *stage_3*, *stage_4*, *stage_5*, *prog* >>

Outline

Timing Anomalies Intuition

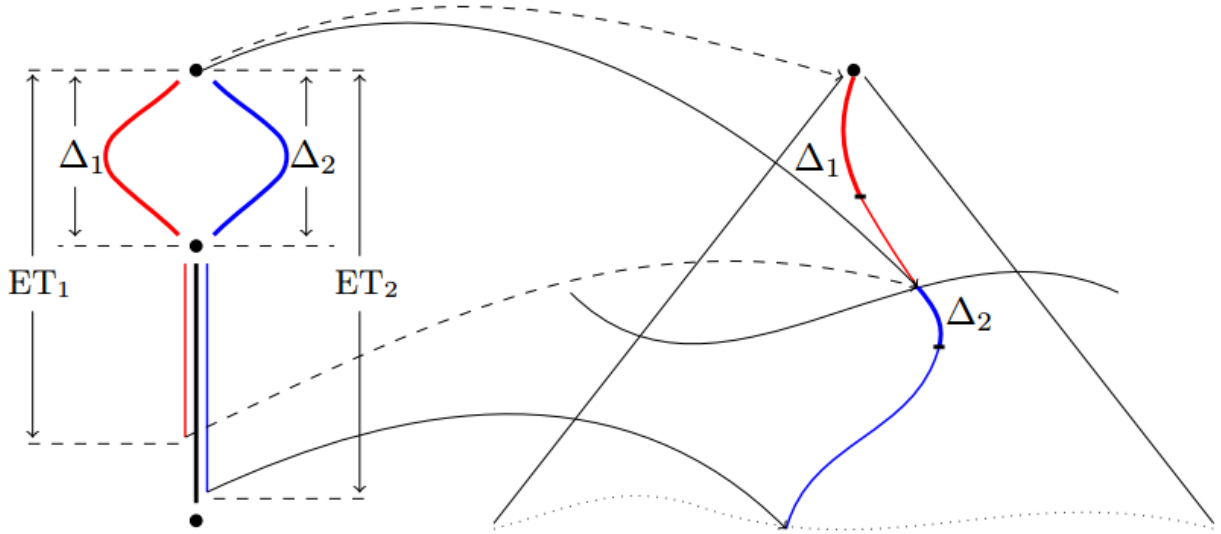
- Timing anomalies complicates the **WCET** analysis
- Timing anomalies detection needs two variations of the same program execution (same sequence of instructions)



➔ Formal TLA+ timing anomalies modeling requires:

- Formal Model: HW/SW
- Variability Notions: **local/global**
- Two **Program** Traces

Timing Anomalies TLA+ Specification



- State Variable: **asi** =
  [**delta**|− > [**n** ∈ **1..2**|− > **0**],
  ET|− > [**n** ∈ **1..2**|− > **0**],
- Extended Pipeline Specification:
  Next ≜ **PipNext** ∨ **PipStall** ∨ **Reload** ∨ **Other**
- Property:
  ∼ (**asi.delta**[1] < **asi.delta**[2] ∧ **asi.ET**[1] > **asi.ET**[2])

TLA+ Program Specification

- Local variations: set of instruction latencies in execute stage (**latency**)
- Extended asi variable: **asi =**
  pcid$|->$ **Program.did**,
  pathid$|->$ **Program.fst**

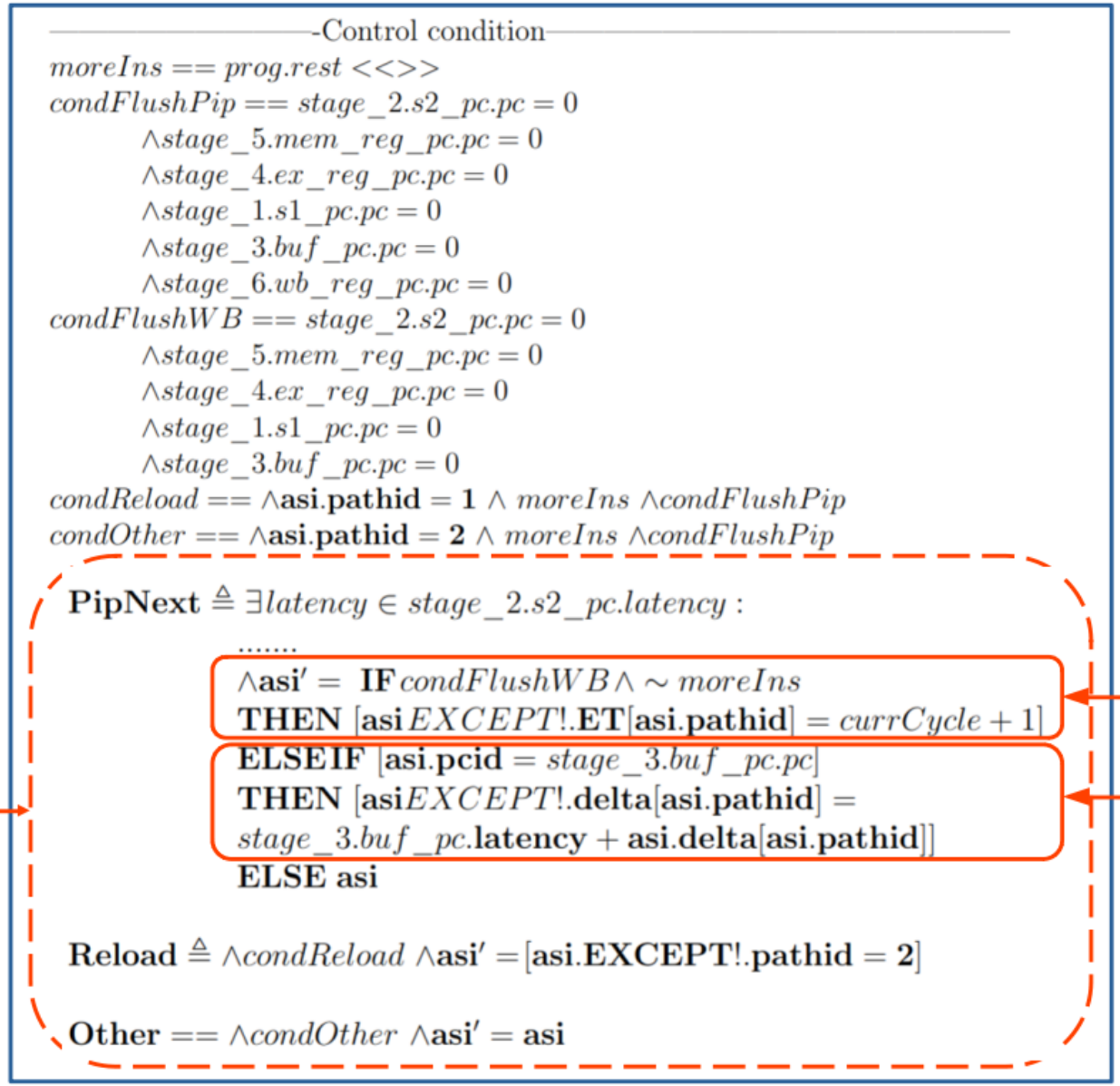
Rocket Timing Anomalies : TLA+ Program

```
Program <- [ fst |-> « [pc |-> 1, type |-> "lw-RR", latency |-> {8,10}],
                       [pc |-> 2, type |-> "add-RR", latency |-> {1,2} ],
                       [pc |-> 3, type |-> "bne-RR", latency |-> {5,8}] »,

             snd |-> « [pc |-> 1, type |-> "lw-RR", latency |-> {8,10}],
                       [pc |-> 2, type |-> "add-RR", latency |-> {4,3}],
                       [pc |-> 3, type |-> "bne-RR", latency |-> {5,8}] »,

             did |-> 2 ]
```

Rocket Processor: Encoding of Timing Anomalies



Rocket Processor: Timing Anomalies Verification Results

- Cycle = 21
- **asi = [delta |-> «1, 3», ET |-> «25, 21»]**

Outline

Conclusion and Future Work

- **Contributions**
  - Workflow to formally verify timing properties
  - Automatic generation of datapath pipeline model of RISC-V Chisel-based processors
  - Generate the TLA+ formal model from abstract pipeline model
  - Integrate the TLA+ formal model into the formal detection of timing anomalies procedure with TLC model checker
- **On going work**
  - Apply the workflow on other RISC-V processors for timing analysis
  - Add several functional units on pipeline models to extend the detection of timing anomalies analysis

References I

[AHJ18] Mihail Asavoae, Belgacem Ben Hedia, and Mathieu Jan. "Formal Executable Models for Automatic Detection of Timing Anomalies". In: *WCET 2018*. Vol. 63. 2018, 2:1–2:13.

[Ben+22a] Samira Ait Bensaid et al. "Deriving Pipeline Models for Timing Analysis from High-Level HDL Processor Designs". In: *MEMOCODE*. Submitted. 2022.

[Ben+22b] Samira Ait Bensaid et al. "Work in Progress: Automatic Construction of Pipeline Datapaths from High-Level HDL Code". In: *RTAS-BP*. To appear. 2022.

[Bin+20] Benjamin Binder et al. "Scalable Detection of Amplification Timing Anomalies for the Superscalar TriCore Architecture". In: *FMICS*. 2020.

[Sam+22] Benjamin Binder Samira Ait Bensaid et al. "Formal Processor Modeling for Analyzing Safety and Security Properties". In: *ERTS*. To appear. 2022.