Property-Driven Development of a RISC-V CPU

Tobias Ludwig

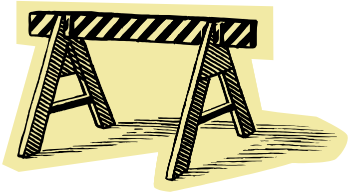
Technische Universität Kaiserslautern (TUK)

Status of Formal Verification

The biggest hurdle

black box verification

(simulation)



white box verification

(property checking)

Property Checking:
More like *design* than *verification* !

Vision

Formal RTL verification should:

- do more than *bug hunting*
- support new abstraction principles between electronic system level models (ESL) and low-level implementations (RTL)
- help to emancipate ESL models from prototypes to golden design models

Semantic Gap

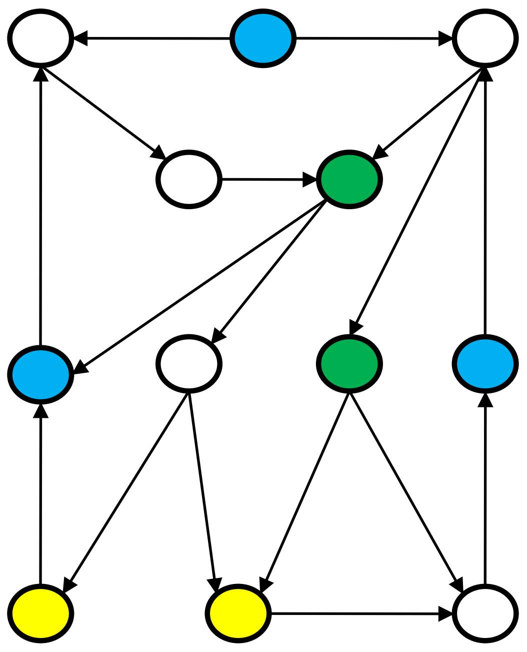
High trust in RT level as golden reference!

- Why do we trust the RT level?
- Equivalence proven through equivalence checking (EC)

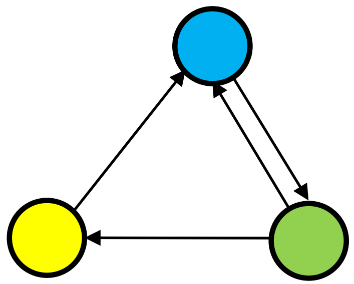
Trust in system-level models as golden reference?

- Almost no trust, except of HLS
- No notion of equivalence → EC not possible
- High-level synthesis is taking away too many RT-design decisions

Path Predicate Abstraction (PPA)



Operationally colored graph *G*

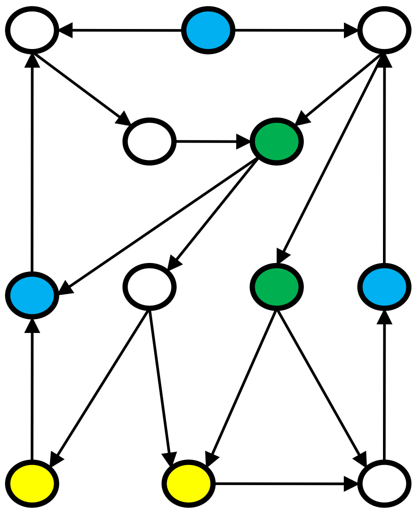


Path predicate abstracted graph *P*

Soundness Theorem

Let P be a path predicate abstraction of a graph G.
Then, for every (finite or infinite) path in G visiting a sequence of colored states (w_0, w_1, w_2, …) there exists an abstract path representing the same sequence of colors ($c(w_0)$, $c(w_1)$, $c(w_2)$, …) in P, and vice versa.

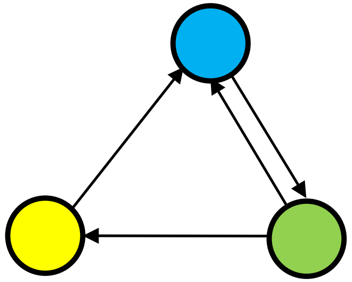
Path Predicate Abstraction (PPA)



Operationally colored graph

- Colored node: Important state
- Uncolored node: Unimportant state
- Operation: Transition between important states:
  - e.g., blue goes to green

ESL



Path predicate abstracted graph

Soundness

– Verification results obtained at ESL translate to the RTL

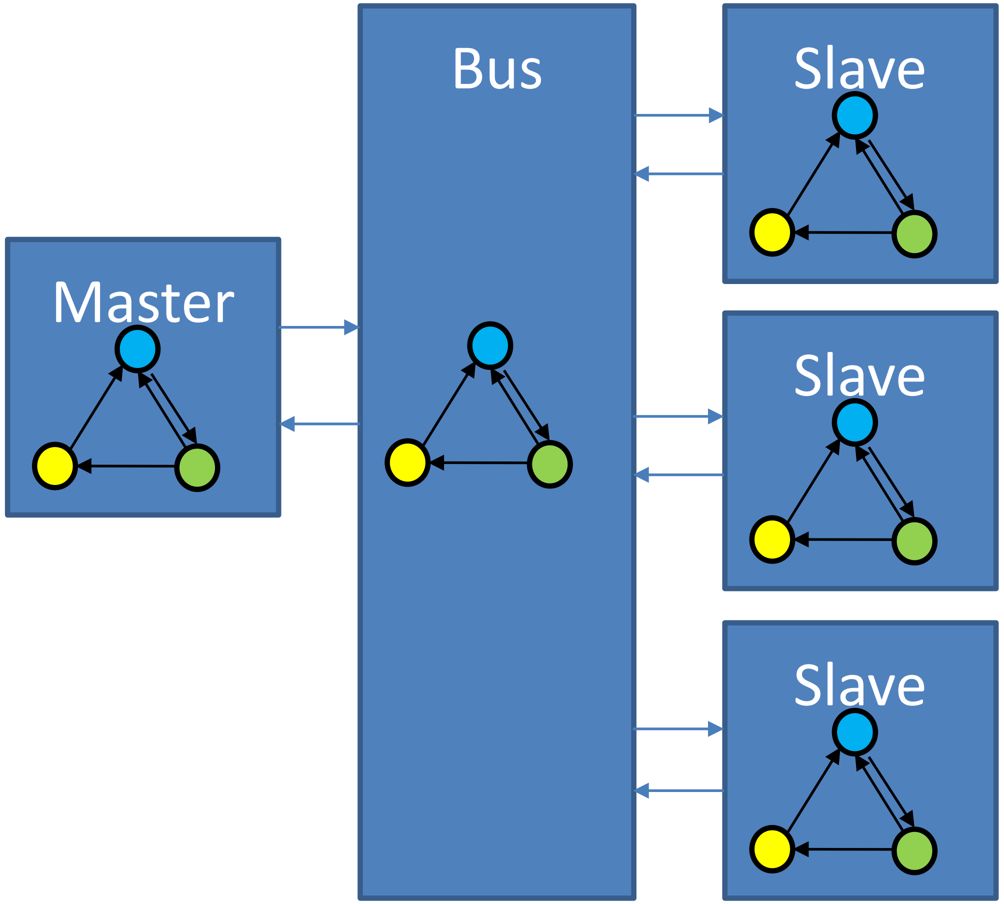
– Global verification tasks can be moved from the RTL to the ESL

– Significantly less chip-level simulation is required

Theorem shown for LTL properties

[J. Urdahl, D. Stoffel, W. Kunz: "Path Predicate Abstraction for Sound System-Level Models of RT-Level Circuit Designs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 33, No. 2, Feb. 2014, pp. 291-304.]

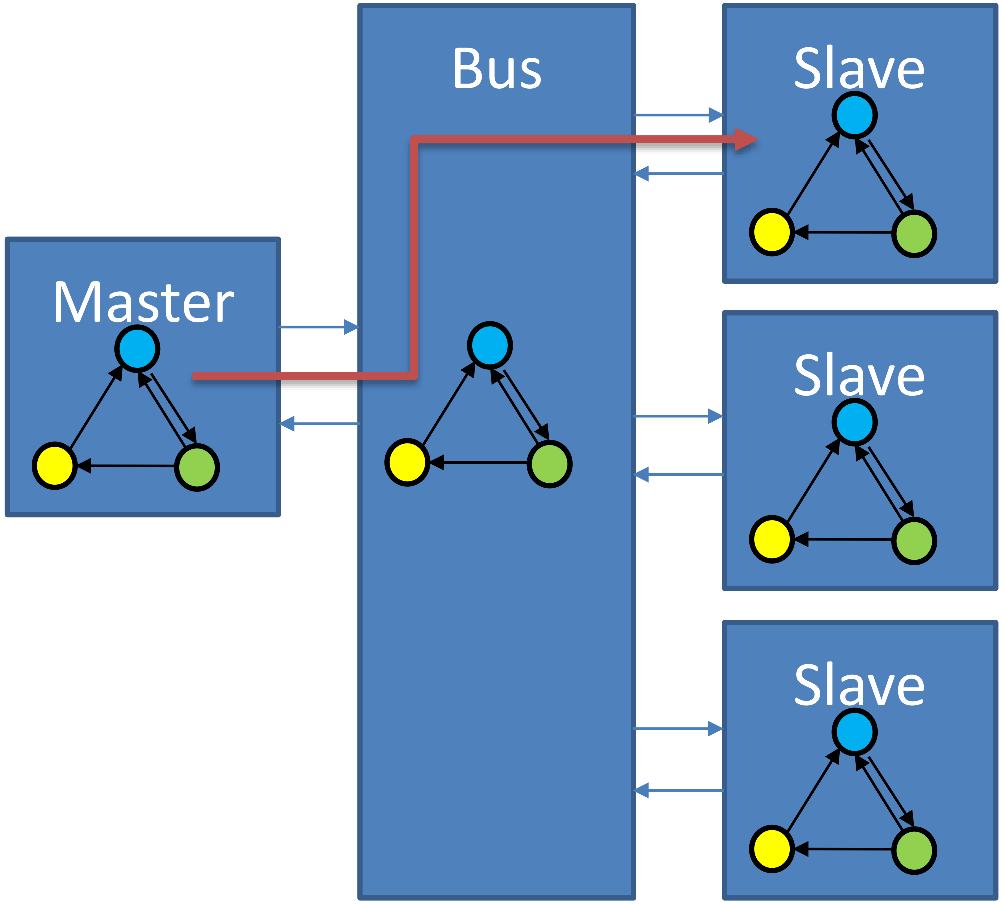
PPA in practice?



Setup:

- ESL model of a bus
- FSM of each module is a PPA
- Implement sound RTL for each PPA

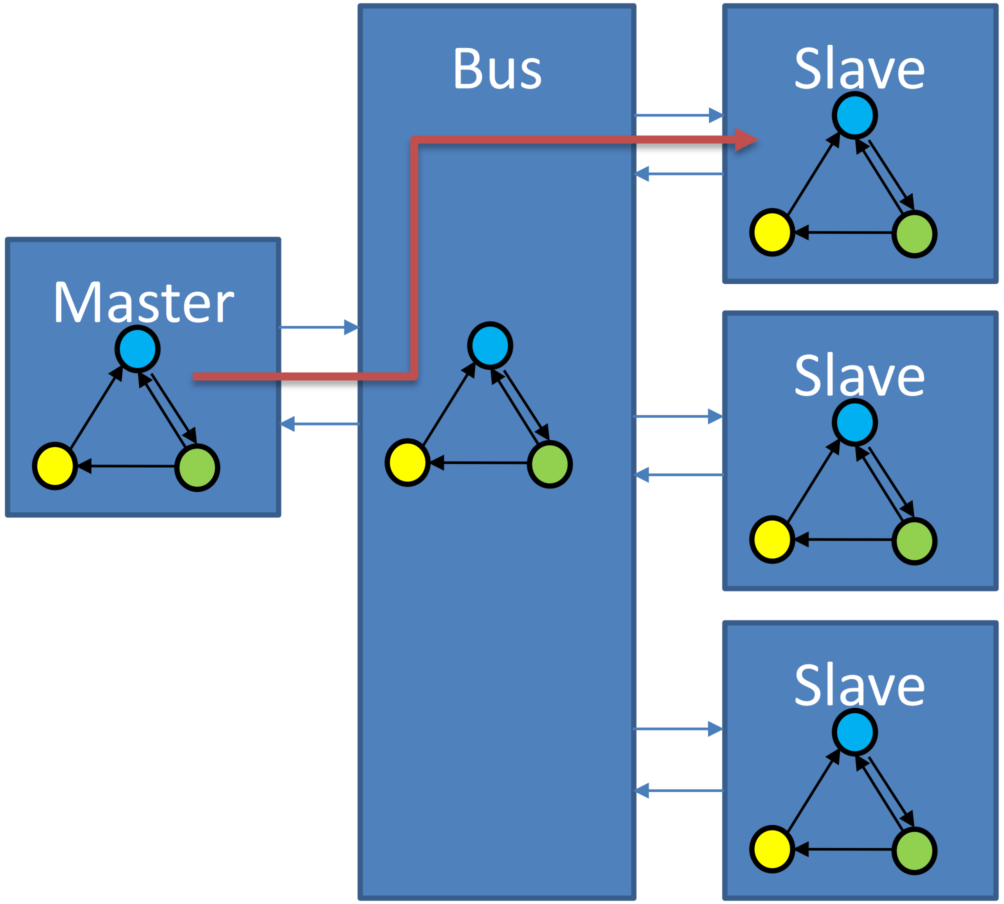
PPA in practice?



Setup:
- ESL model of a bus
- FSM of each module is a PPA
- Implement sound RTL for each PPA

Verify on system-level:
- E.g.: Simulate sending a message across the bus at the ESL
- Working correctly? Due to soundness RTL works correct, too
- No more verification required for system-level behavior

PPA in practice?



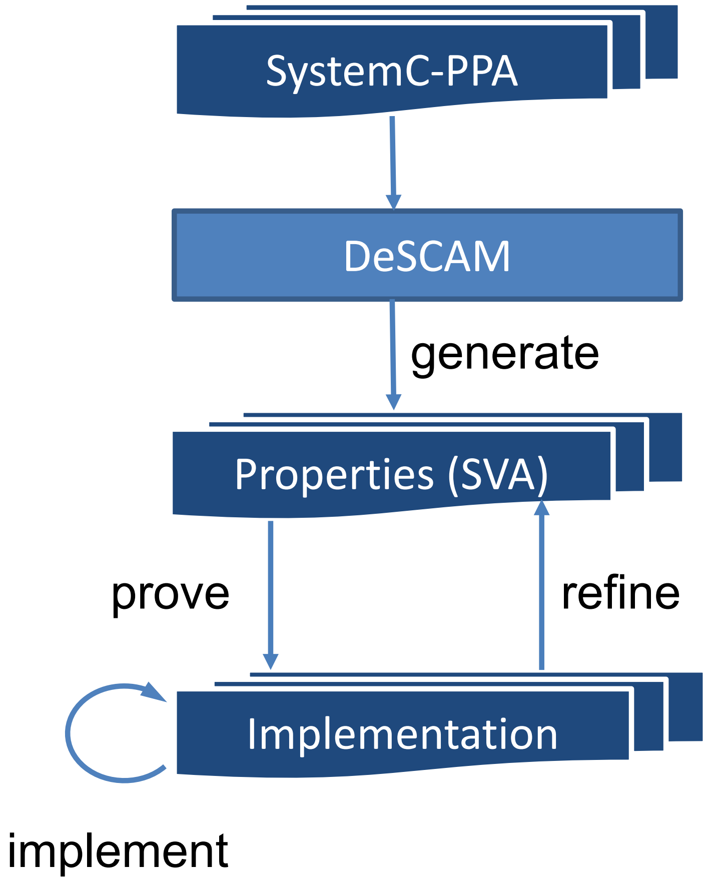
Setup:
- ESL model of a bus
- FSM of each module is a PPA
- **Implement sound RTL for each PPA**

Verify on system-level:
- E.g.: Simulate sending a message across the bus at the ESL
- Working correctly? Due to soundness RTL works correct, too
- No more verification required for system-level behavior

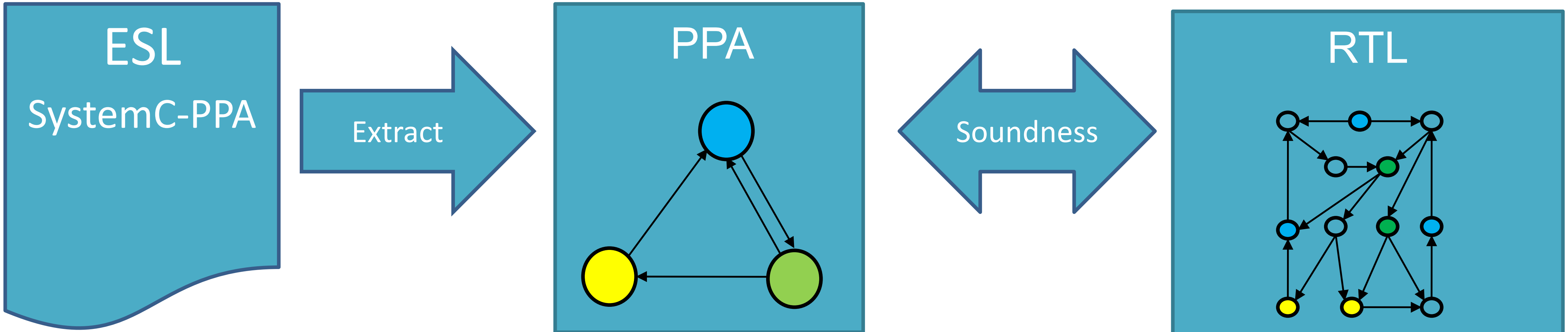
Property-Driven Development (PDD)

Basic Idea

- System model as formal specification:
  - Designable/implementable subset of SystemC (*SystemC-PPA*)
  - Generate complete set of properties
- Refine generated properties (templates)
- Prove properties on final implementation



DeSCAM: from ESL to RTL



DeSCAM

- De*SCAM* (“**De**sign from **S**ystemC **A**bstract **M**odels”):
  - Analyzes a given SystemC model for compliance with the designable subset
  - Supports by refining the model into a SystemC-PPA
  - Automatically generates the properties
- A manual explains how to use DeSCAM in Property-Driven Development (from a practical point of view)

Available on GitHub: github.com/ludwig247/DeSCAM

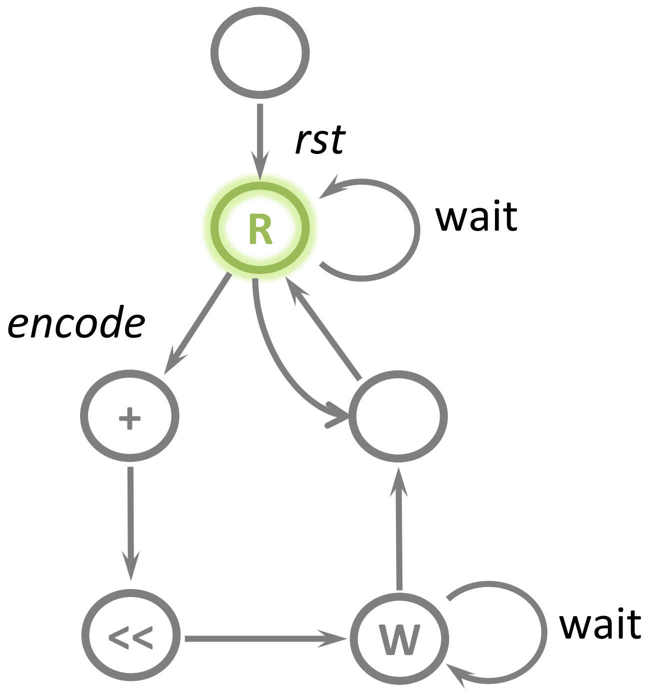
Designable Subset: SystemC-PPA

- Models an FSM in a time-abstract fashion
- Single thread executing infinitely
- Only blocks if a communication interface is called
- No cyclic path without a blocking communication
- No dynamic memory allocation

Extracting the PPA from the SystemC CFG

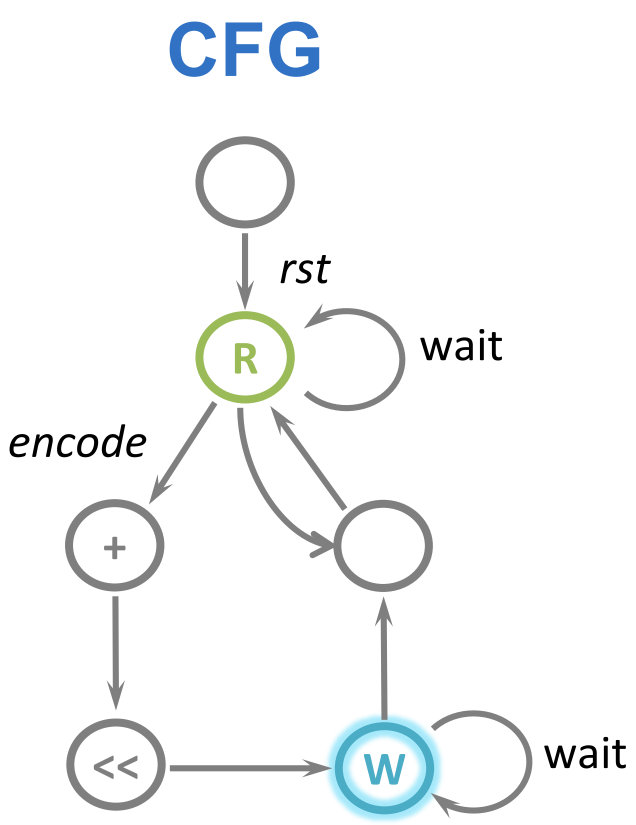
```
SC_MODULE(encoder) {
    // constructor and declarations
    void fsm() {
        while(true){
          bus→read(status,data)
          if (status==encode) {
            // encode(data)
            bus →write(data);
          }
        //some code
     } } };
```

CFG



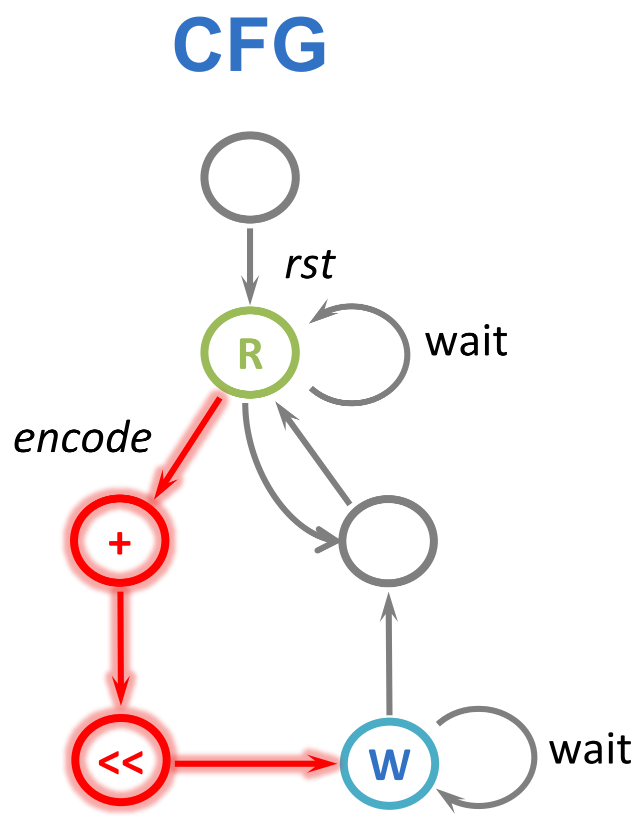
Extracting the PPA from the SystemC CFG

```
SC_MODULE(encoder) {
    // constructor and declarations
    void fsm() {
        while(true){
         bus→read(status,data)
         if (status==encode) {
           // encode(data)
           bus →write(data);
         }
       //some code
    } } };
```

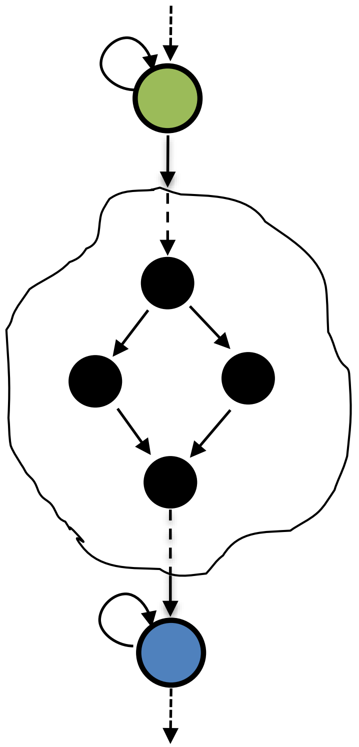


Extracting the PPA from the SystemC CFG

```
SC_MODULE(encoder) {
    // constructor and declarations
    void fsm() {
        while(true){
          bus→read(status,data)
          if (status==encode) {
            // encode(data)
            bus →write(data);
          }
        //some code
    } } };
```



Operation properties



- Operations start and end in important states
- Important states subsume millions of concrete states
- Operations have an arbitrary, (but fixed) length of n cycles
- Objects of the system-level are referenced by macros/functions

```
property encode;
assume:
   at t:    start_state;
   at t:    status == encode;
prove:
   at t+n: out(encoded(data@t));
   at t+n: end_state;
endproperty;
```

SVA Properties

```
property encode(length);
//freeze variables
        int data;
        status_t status;
// freeze values @t
        t ##0 hold(data_0, port_in_data()) and
// triggers
        t ##0 start_state() and
        t ##0 sync() and
        t ##0 port_in_status() == encode
implies
        t_end(length) ##0 end_state() and
        t_end(length) ##0 port_out == encoded(data_0)
endproperty;
```

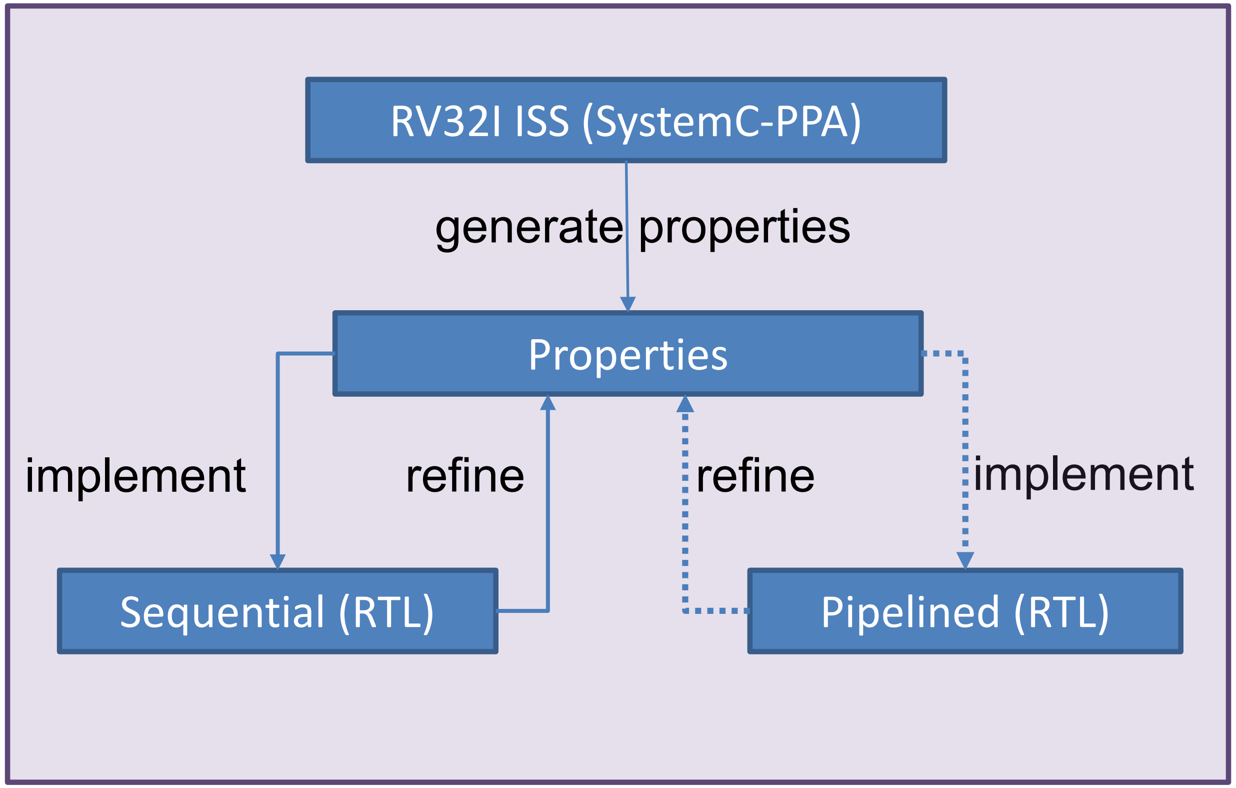
```
function int port_in_data()
    return {
        $past(unit/data_in,7),
        $past(unit/data_in,6),
        $past(unit/data_in,5),
        $past(unit/data_in,4),
        $past(unit/data_in,3),
        $past(unit/data_in,2),
        $past(unit/data_in,1),
        unit/data_in
        };
endfunction;
```

What is the *promise?*

- Correct-by-construction design
- Increased design productivity
- No RTL simulation
- Support for aggressive optimization techniques

Case study: RISC-V CPU

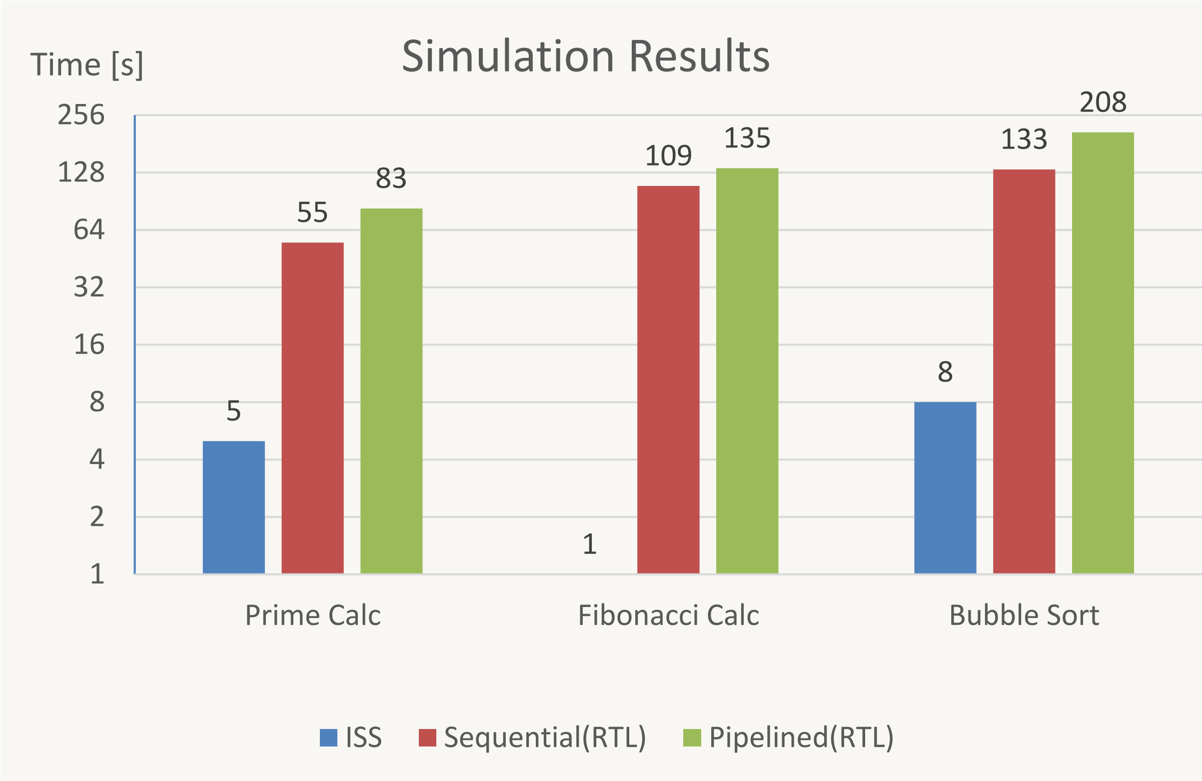
- PDD design of a RV32I processor
- Instruction set simulator (ISS) as a designable SystemC-PPA
- Two RTL implementations
- Implementations are sound refinements of the same ESL



Experimental Results RISC-V

- Work effort for RTL design:
  - Sequential: 2 weeks
  - Pipelined: 4 weeks
- Work effort for verification:
  - Sequential: 3 person days
  - Pipelined: 4 weeks

Design Size	RTL States	LoC	PPA /Oper./Var.
Sequential	1881	1430	6 states/ 21 op/ 7 var
Pipelined	2502	2020	6 states/ 21 op/ 7 var



Industrial Case Study

- Provided industrial Framer
- Extracting SystemC-PPA: 6 PM*(person month)*
- Top-down redesign: 1PM
- 22 properties proven in 22min
- FF reduction 10%
- Power saving: up to 50%

Module	RTL States	LoC RTL
Framer(or.)	4.2k – 47k	27k
Monitor(or.)	30	850
Framer(re.)	3.9k – 42k	12k
Monitor(re.)	92	92

Module	PPA States	PPA Oper.	PPA Variables
Framer	4	13	4
Monitor	2	9	2

Conclusion

- **Property Driven Development (PDD):**
  - Results in a formally sound correct-by-construction design
  - No formal verification knowledge required
- In practice, PDD is based on:
  - The provided open-source tool SCAM
  - State-of-the-art property checker

→ Shifting global design and verifications to the ESL!