Case Study: Formal Verification of an ATM Switch Fabric using VIS

	Page
Introduction	5.2
VIS Tool	5.3
Fairisle ATM Switch Fabric	5.5
Verification Strategy	5.7
Verification by Model Checking	5.9
Enhancement Techniques for Model Checking	5.18
Verification by Equivalence Checking	5.24
Conclusions	5.27
References	5.29

Introduction

The Problem:

- The need to produce high integrity communications systems
- ATM (Asynchronous Transfer Mode) switches are basic elements of state-of-the-art networks.
- Need a suitable technology to verify a whole ATM switch

Conventional Approaches:

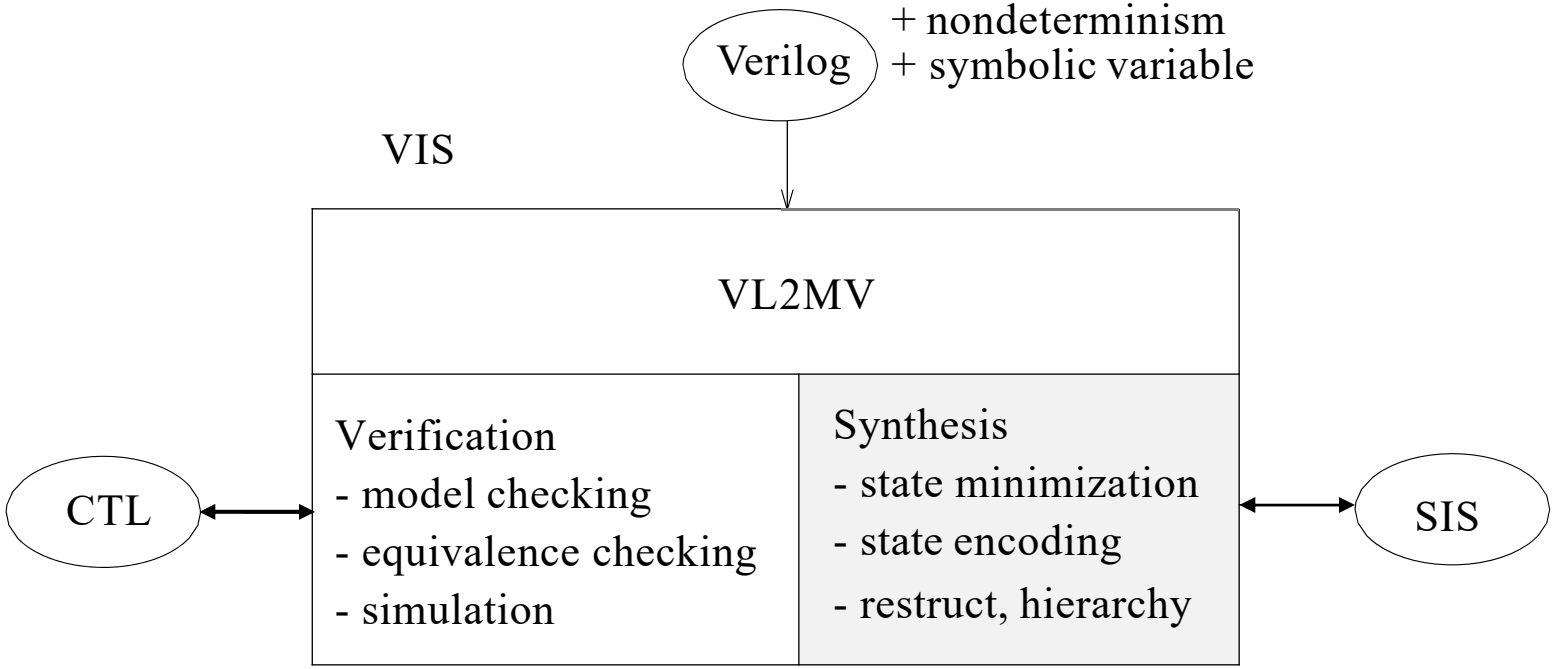
- Post-design Simulation, and Testing

$\Rightarrow$ Cannot guarantee complete correctness

The Proposed Approach:

- Use Formal Verification based on Model Checking and Equivalence Checking in the VIS tool.
- Develop techniques to avoid state space explosion

VIS Verification Tool

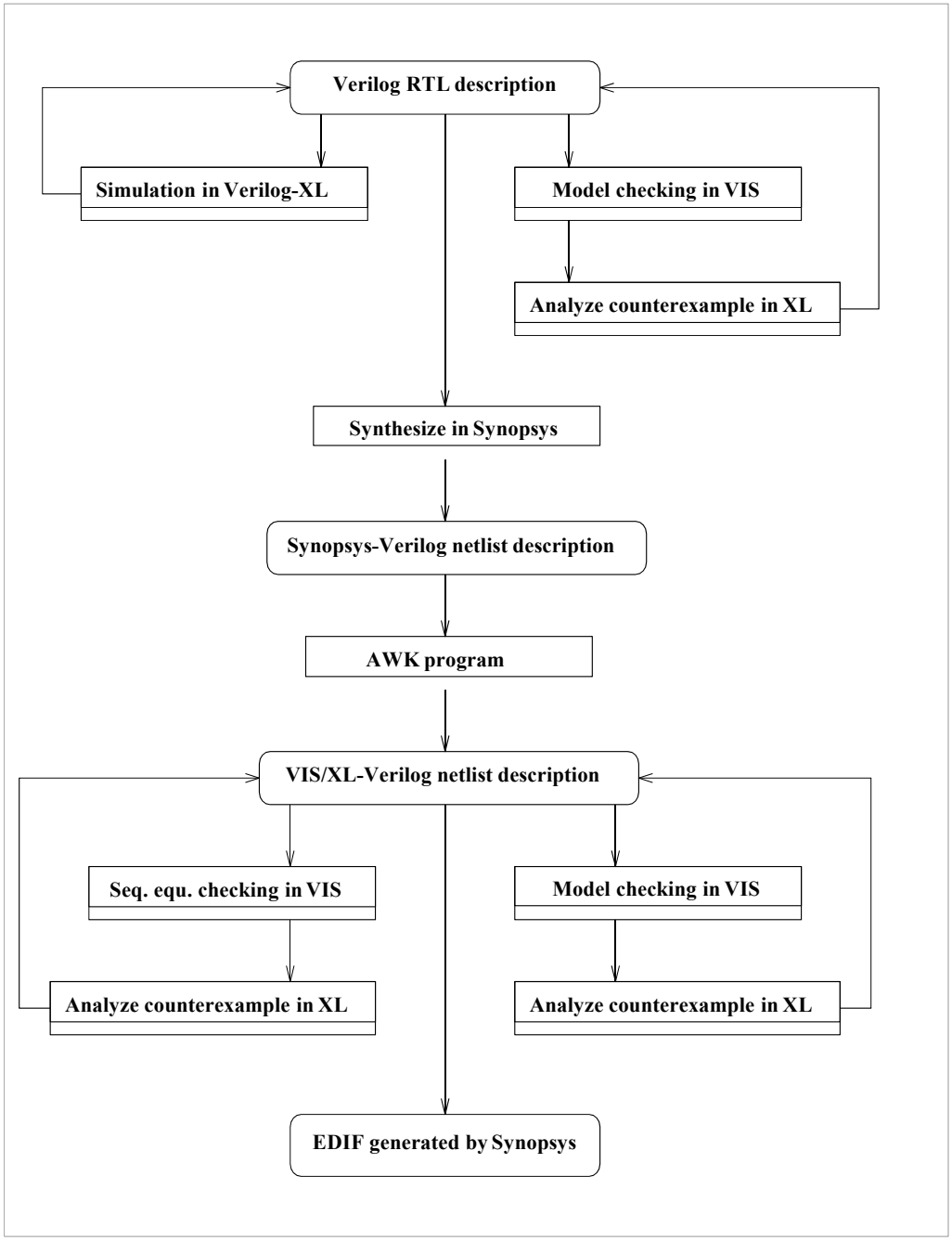


VIS: **V**erification **I**nteracting with **S**ynthesis

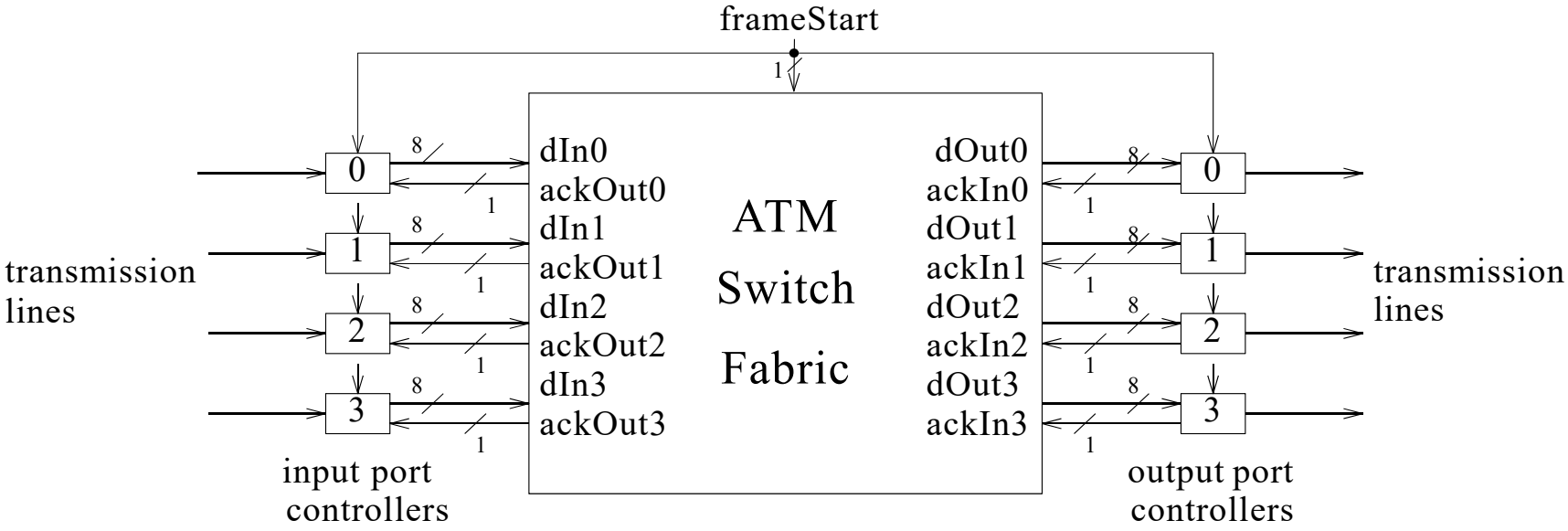
SIS: **S**equential **I**nteractive **S**ynthesis

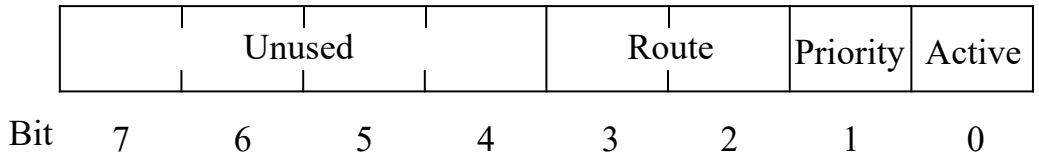
CTL: **C**omputational **T**ree **L**ogic

Design Flow and Formal Verification (VIS)

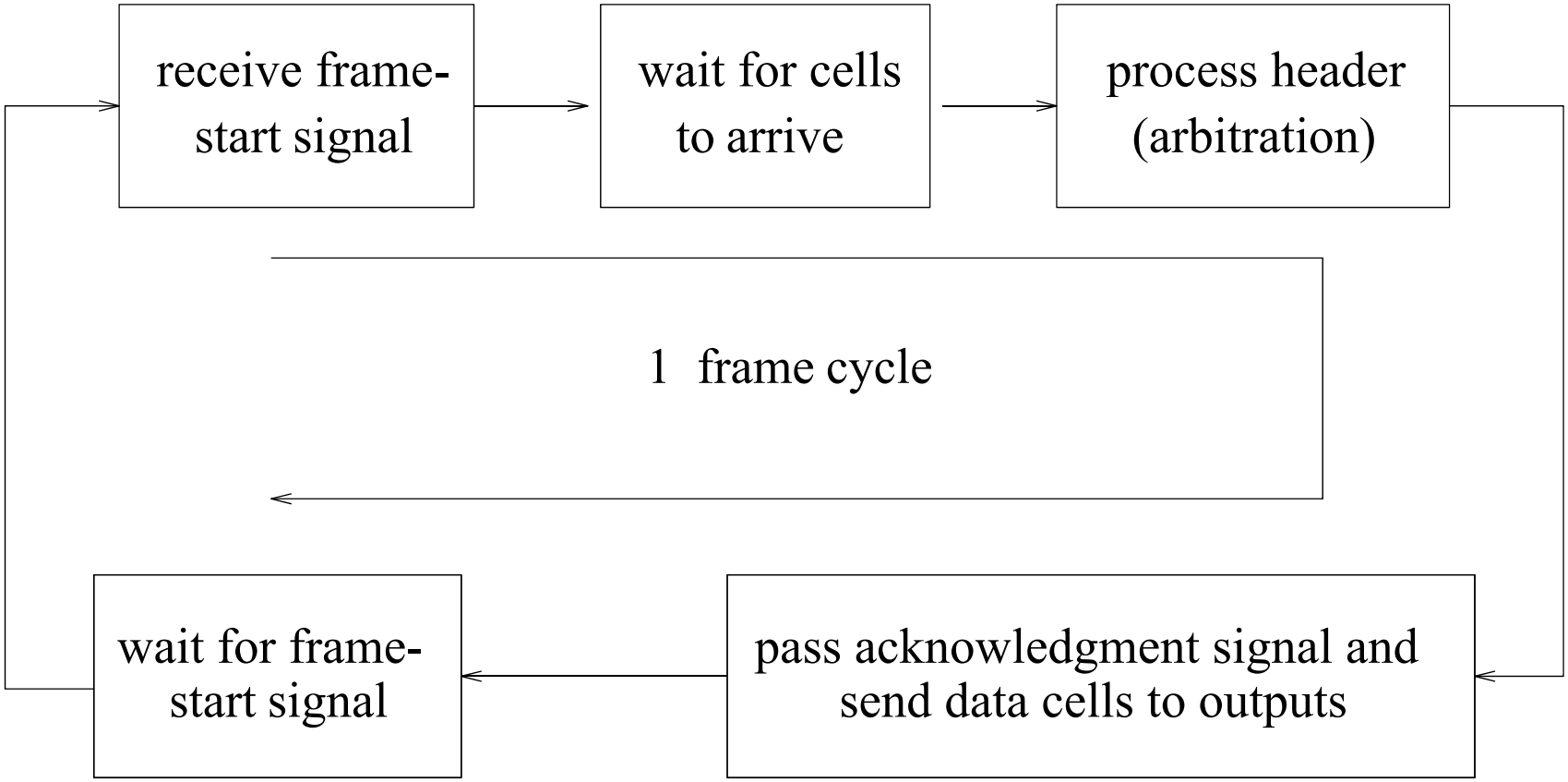


Fairisle ATM Switch Fabric

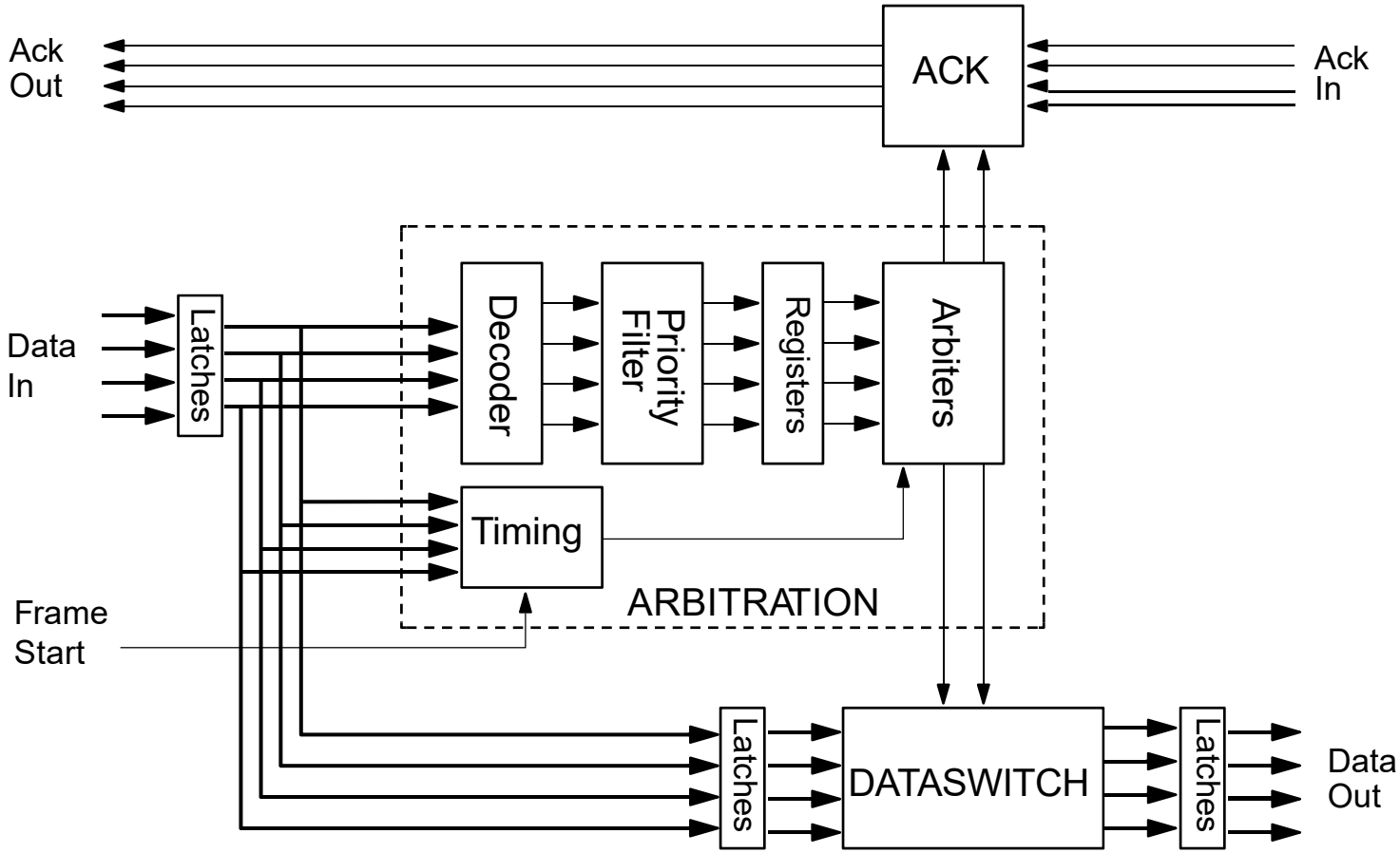




Switch Fabric Behavior



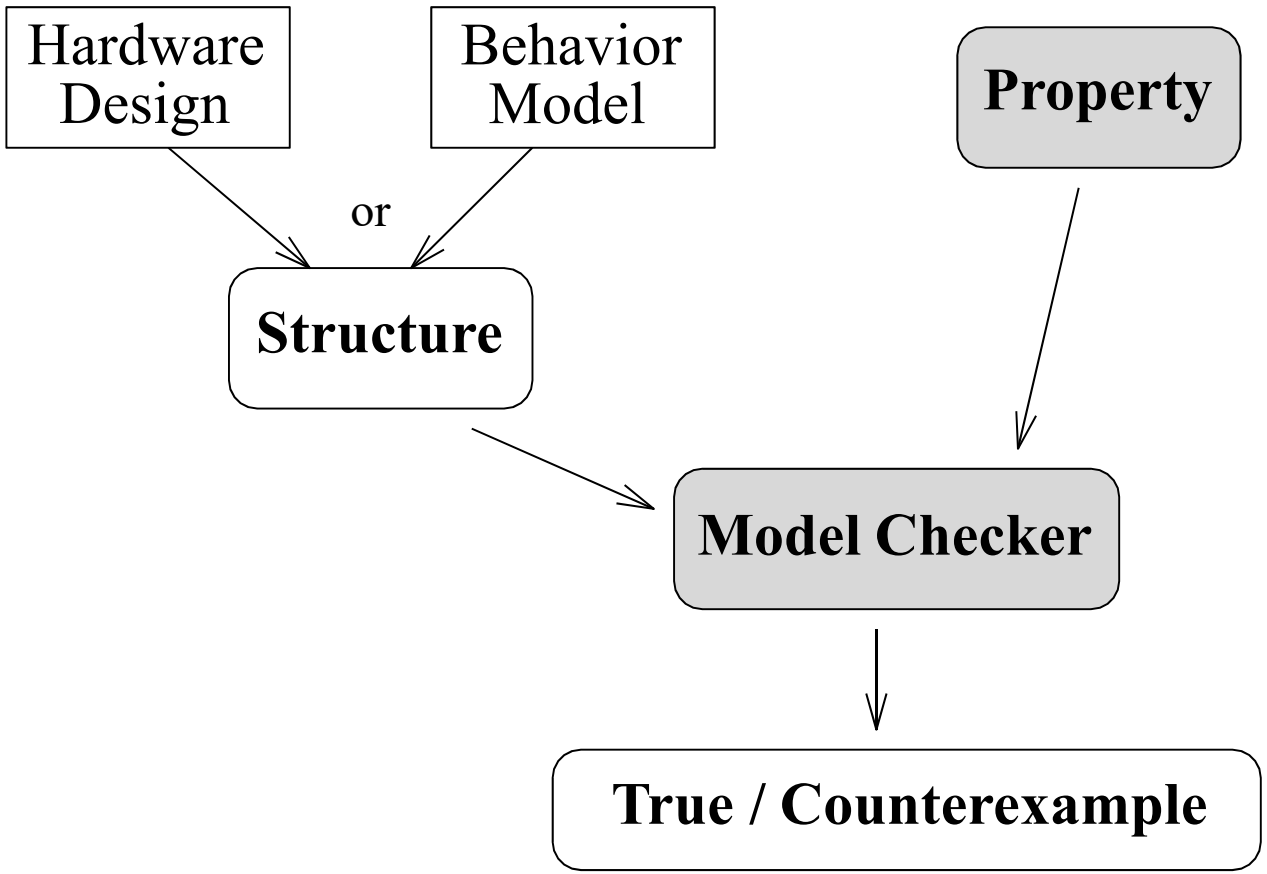
Switch Fabric Implementation



Verification Strategy

- **Objective**: Impl. satisfies Spec. (property and equ. checking)
  - RTL description in Verilog (Specification)
  - Netlist description in Verilog (Implementation)
  - Properties in CTL (safety & liveness properties)
- **Problems**:
  - Verification cannot handle large circuits
  - Hard to consider all initial states of a circuit
- **Strategy**:
  - Model checking on abstracted models augmented with several enhancement techniques
  - Equivalence checking of submodules of the circuit

Model Checking — Basic Idea



Model Checking on the Fabric

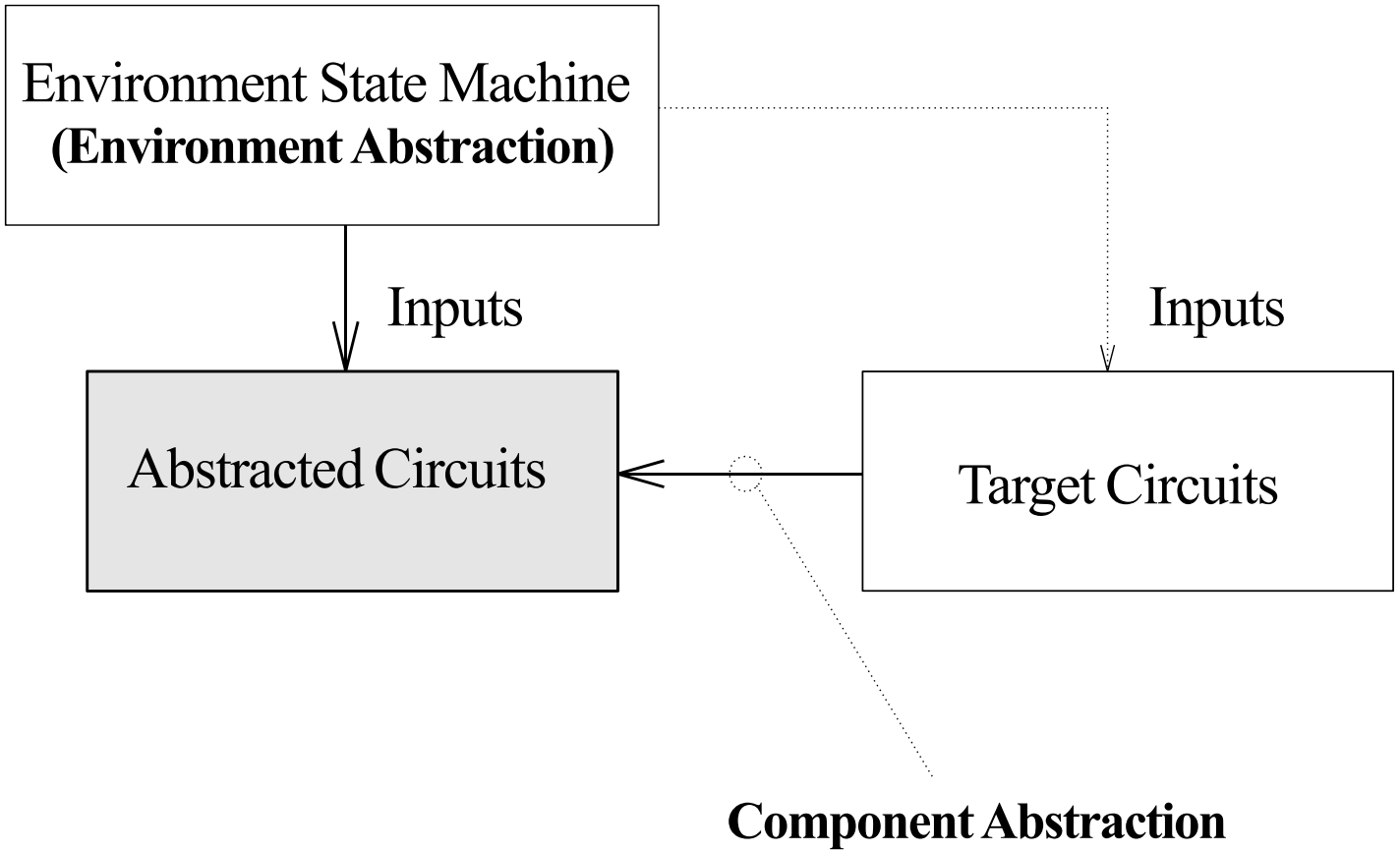
⇒ Problem in property checking: *state space explosion*

- 210 latches in the design ⇒ 2^{210} states (assuming no state reduction skill by the tool)

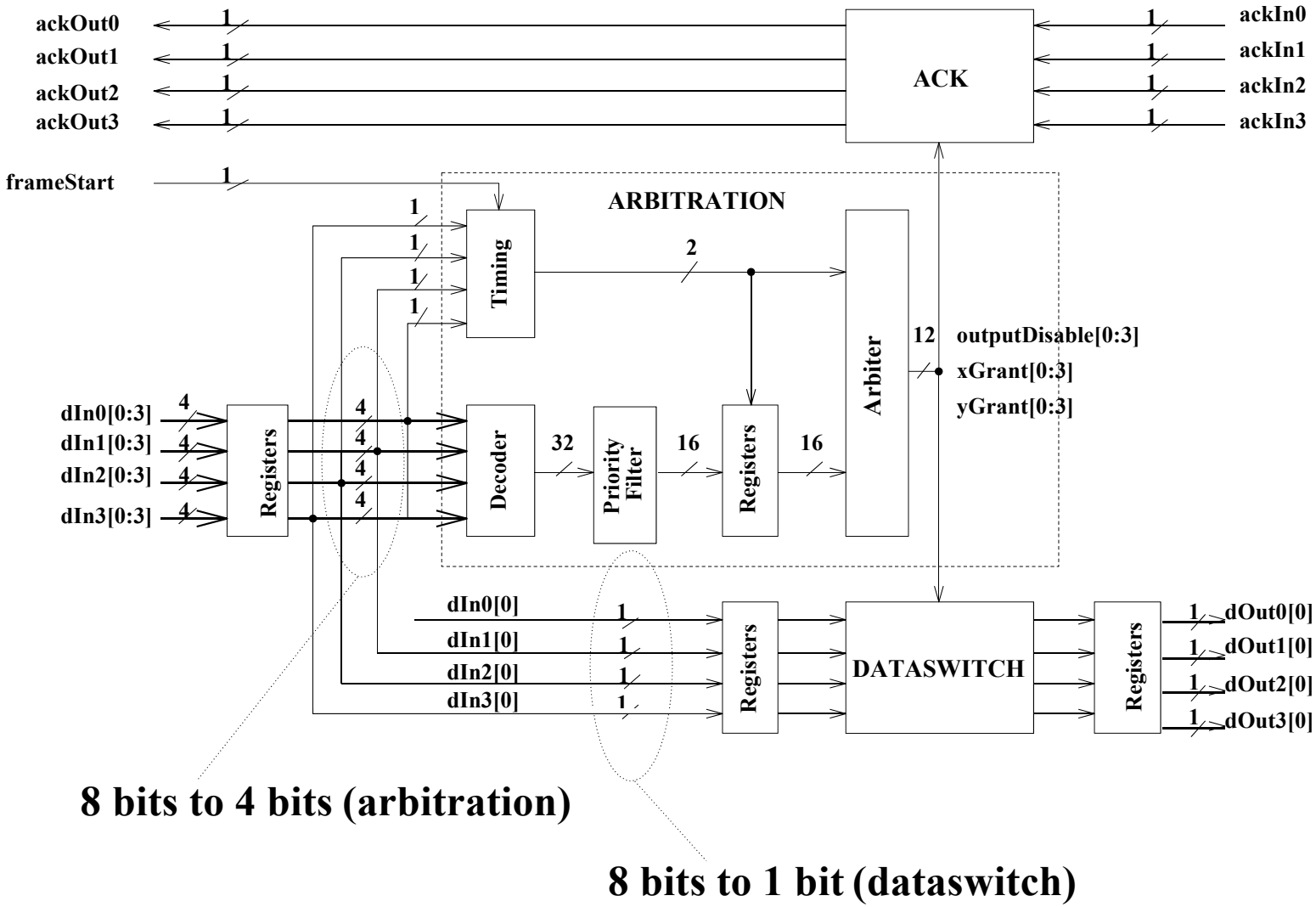
⇒ Solutions:

- **Environment abstraction:** restrict the possible inputs according to the expected behavior (e.g. frames of 64 clock cycles)
- **Component abstraction:** abstract the target component by some rules (e.g. reduce the dataswitch paths from 8 bits to 1 bit)

Abstraction Techniques



Abstracted Fabric



Why do we need an environment machine?

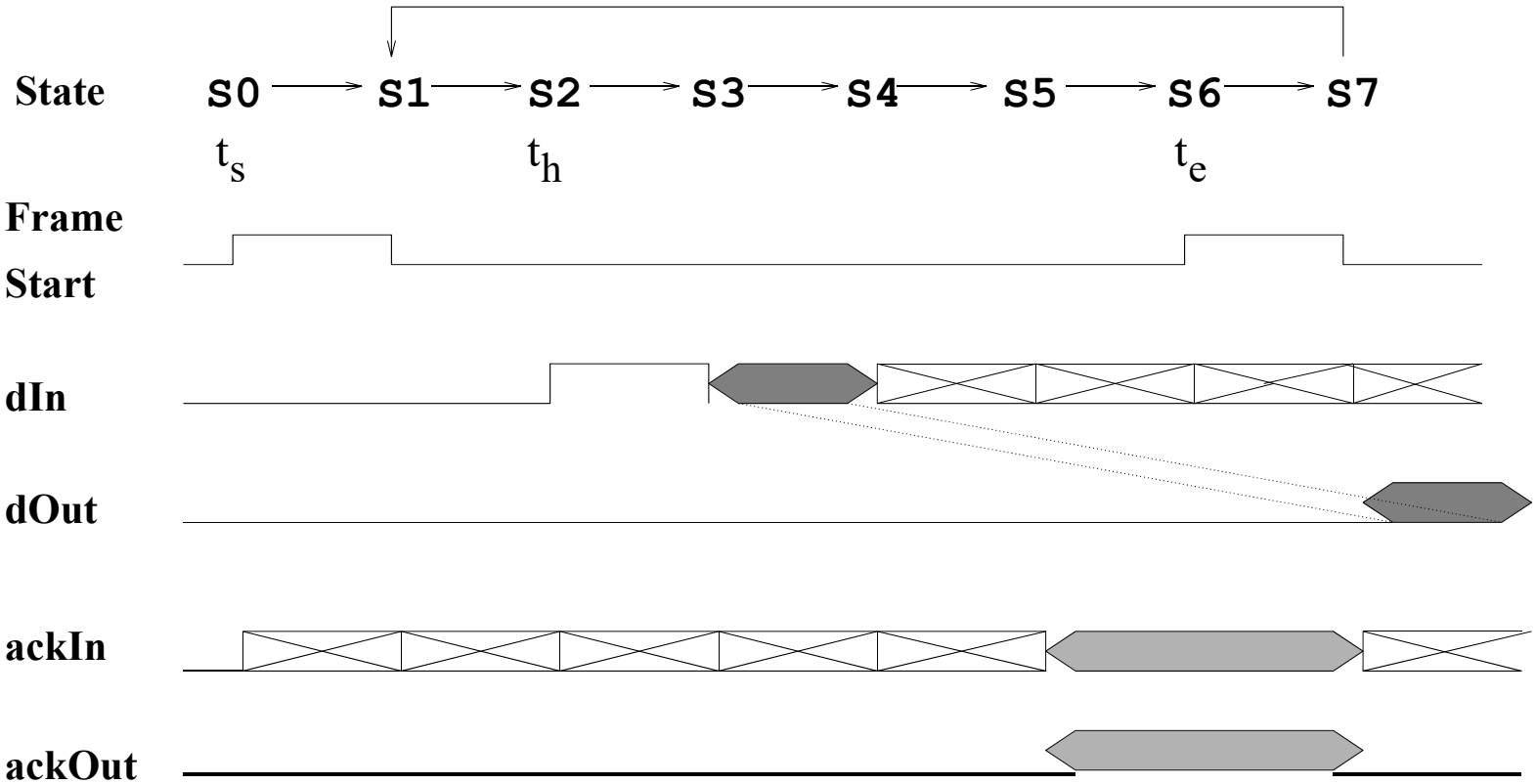
1. Explicit input values required in CTL $\Rightarrow$ we use nondeterministic register variables to express inputs

2. No description of explicit time points in CTL $\Rightarrow$ we use explicit states to express timing information

3. Imitates the behavior of port controllers and hence constraints the number of possible inputs of the fabric

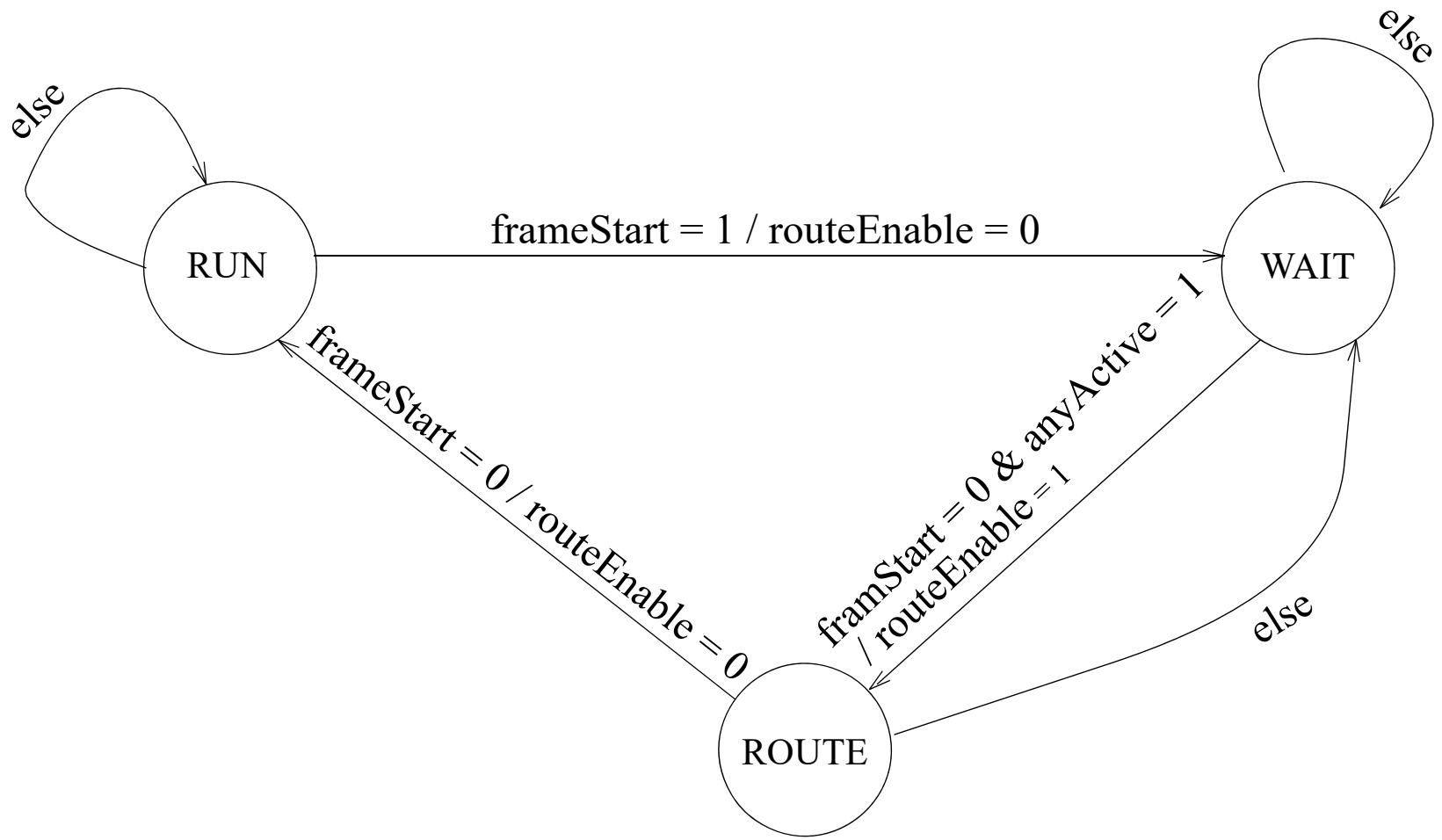
$\Rightarrow$ We use 8 state environment machine to ease the CTL expressions

8-state Environment Machine



How do we extract properties from Spec?

Example: *Timing block*



How do we extract properties from Spec?

$\Rightarrow$ Generally, properties are based on FSM specification

- **Liveness** (something good will eventually happen)

  Example: *If framStart = 0 in "wait" state, the fabric will eventually move to "route" state.*

  CTL: `AG (A (frameStart = 0 ∧ state = wait) U (state = route))`

- **Safety** (nothing bad will ever happen)

  Example: *If (frameStart = 0 and anyActive = 1) in "wait" state, its next state must be "route".*

  CTL: `AG (state = wait ∧ frameStart = 0 ∧ anyActive = 1`
  $\Rightarrow$ `AX (state = route))`

Results of Model Checking

Properties	CPU time (seconds)	Memory usage (MB)	Nodes allocated
Property 1	3933.9	40.3	84,199,139
Property 2	4550.7	4.3	90,371,031
Property 3	14.8	2.8	368,749
Property 4	3593.4	32.4	93,073,140
Property 5	833.0	4.5	28,560,871
Property 6	3679.7	40.9	79,687,784
Property 7	414.8	5.3	4,180,124
Property 8	1037.9	11.6	29,755,252

$\Rightarrow$ Unreasonable CPU time (1 to 2 hours machine time)

$\Rightarrow$ Develop enhancement techniques

Enhancement Techniques of Model Checking

1. **Cascade Property Division**
   - Divide a property into several seq. related sub-properties
   - Penalty: Environment machines are required
2. **Parallel Property Division**
   - Split a property into several parallel sub-properties checked on abstracted models stripped from the design
   - Penalty: Disassemble circuits at some specific locations
3. **Latch Reduction**
   - Reduce the primary inputs and outputs of state holding elements (i.e. latches)
   - Penalty: Re-evaluate the timing behavior of circuits

Cascade Property Division (Example)

Property 7: *If input port 0 chooses output port 0 with priority, the value on ackOut0 will be the input of ackIn0.*

CTL: `AG ((dIn0[3:0] = 0011 ∧ dIn1[1] = 0 ∧ dIn2[1] = 0 ∧ dIn3[3]= 0 ∧ state = S2) ⇒ AX AX AX (ackOut0 == ackIn0));`

- **sub-prop. 1:** `AG ((.....) ⇒ AX AX AX (state = S5 ∧ xGrant[0] = 0 ∧ yGrant[0] = 0 ∧ outputDisable[0] = 0));`
- **sub-prop. 2:** `AG ((....) ⇒ ackOut0 == ackIn0);`

⇒ Easy proof: (sub-property 1 ∧ sub-property 2) ⇒ Property 7

(+) enhance model checking by 41 times

(-) environment machine with outputs for intermediate signals (i.e., xGrant, yGrant, outputDisable)

Parallel Property Division (Example)

Property 3: *From t_h+1 (state S3) to t_h+4 (state S6), the default value (zero) is put on the data output ports.*

CTL: `AG (state = S3 ∨ state = S4 ∨ state = S5 ∨ state = S6)`
`⇒ dOut0 = 0 ∧ dOut1 = 0 ∧ dOut2 = 0 ∧ dOut3 = 0 );`

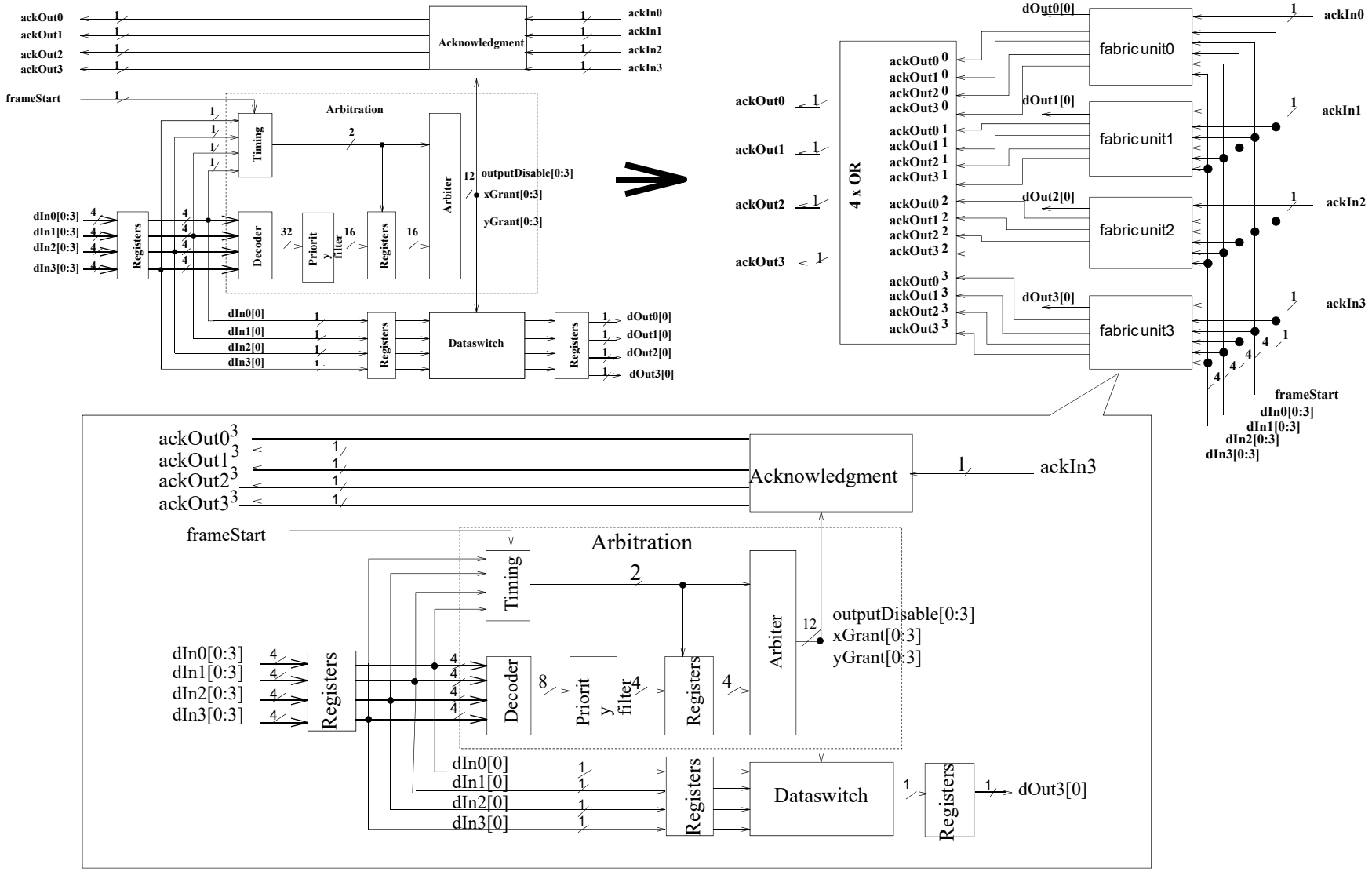
- **sub-prop. i :** `AG (state = S3 ∧ state = S4 ∧ state = S5 ∧ state = S6)`
  (i=1,2,3,4) `⇒ dOut0[i] = 0);`

⇒ Easy to prove: (sub-property1 ∧ sub-property2 ∧ sub-property 3 ∧ sub-property 4) ⇒ Property 3

(+) enhance model checking by 73 times

(−) decompose the fabric circuit in 4 units

Parallel Property Division (cont'd)



Latch Reduction

- Influence of latches on model checking
  - Original fabric (210 latches): impossible to check
  - Abstracted fabric (85 latches): up to 4000 seconds
  - Abstracted fabric unit (54 latches): ≤ 50 seconds
- Example — **Property 2:** *Data bytes in a cell are transferred from input port 0 to output port 0 sequentially with 4 clock cycle delay.*

CTL: `EG (state = S3 -> AX AX AX AX (dOut0 = dIn0`^{S3}`));`

⇒ Reduce a set of primary output latches and check the property:

CTL: `EG (state = S3 -> AX AX AX (dOut0 = dIn0`^{S3}`));`

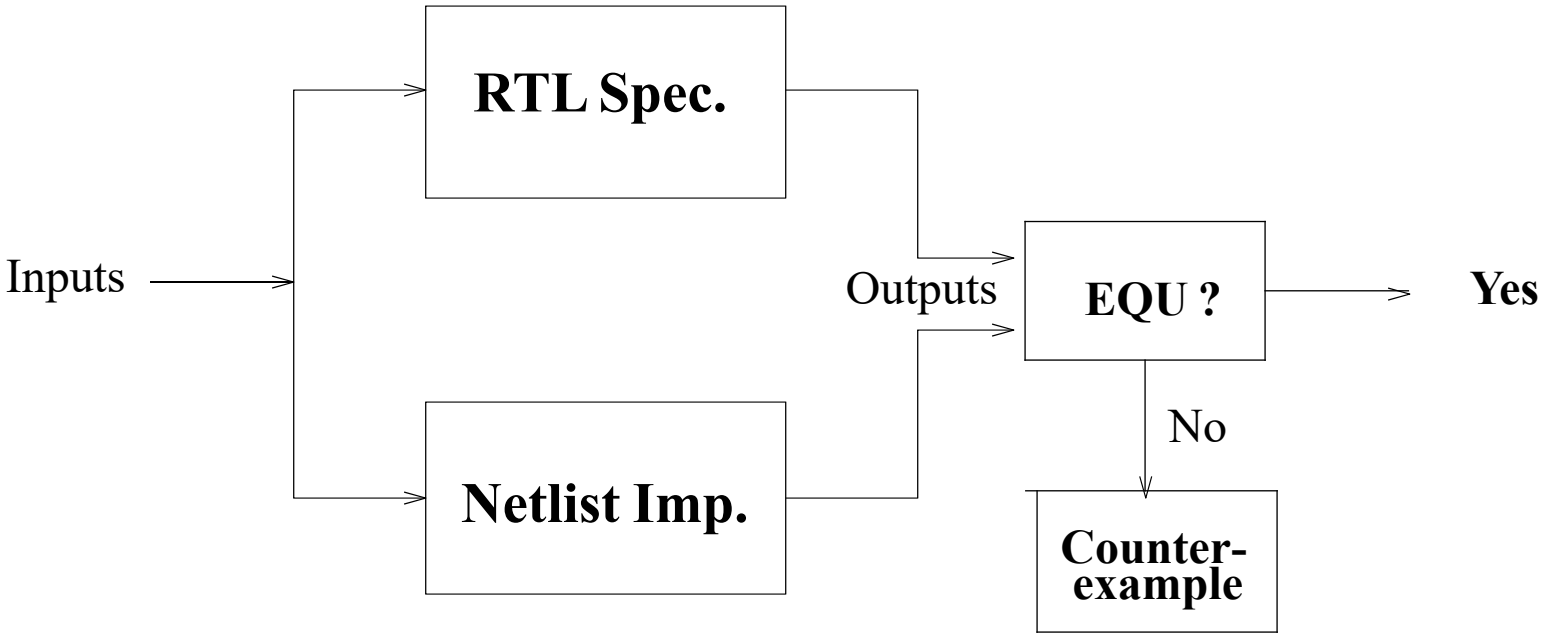
(+) Enhance model checking by 200 times using latch reduction

(−) Re-evaluate the timing behavior

Enhanced Results of Model Checking

Properties	CPU time w/o enhancement (s)	Enhancement techniques	CPU time with enhancement (s)	Speed- up
Property 1	3933.9	latch red.	27.8	142
Property 2	4550.7	latch red.	23.1	197
Property 3	14.8	-	-	-
Property 4	3593.4	parallel div.	48.9	74
Property 5	833.0	parallel div.	72.5	11
Property 6	3679.7	latch red.	34.1	51
Property 7	414.8	cascade div.	10.0	41
Property 8	1037.9	latch red.	46.1	23

Principle of Sequential Equivalence Checking



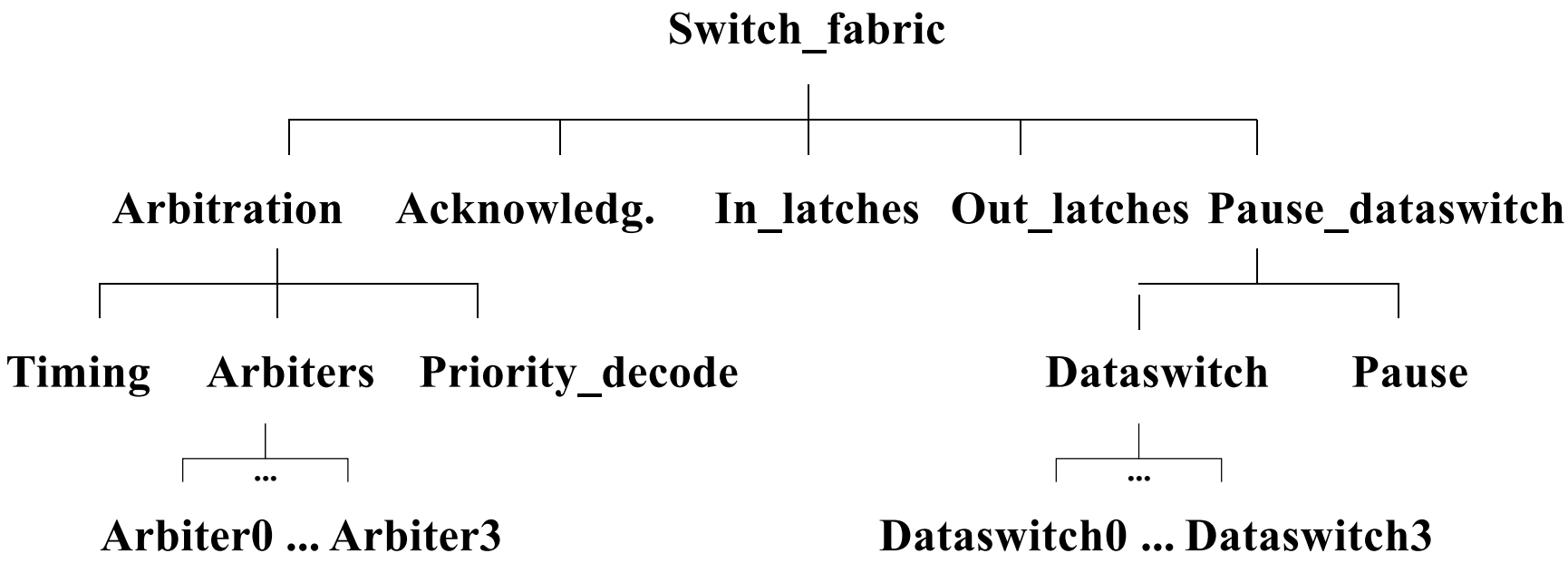
- Combinational circuit: straightforward
- Sequential circuit: must consider all initial states

$\Rightarrow$ Hard to handle large circuits

Equivalence Checking

- Objective: Impl. (Netlist) *equivalent* to Spec. (RTL)

⇒ Apply equivalence checking hierarchically on submodules of the switch fabric in a bottom-up fashion



Results of Equivalence Checking

Component	CPU time (seconds)	Number of latches
Acknowledgment	1.4	0
In_latches	4.2	32
Out_latches	4.2	32
Pause	4.0	32
Arbiter_i	1.4	3
Arbiters	13.3	12
Priority_decode	26.9	16
Timing	0.3	2
Dataswitch_i	1855.8	16
Arbitration	**67860.0**	**30**
Dataswitch	**failed**	**64**
Pause_dataswitch	**failed**	**96**
Switch_fabric	**failed**	**190**

Conclusions

- Simulation is still a powerful verification tool, but it is not sufficient
- Model checking efficiently used in the verification of high-level RTL design and control circuity
- Environment abstraction and component abstraction play an important role in easing model checking
- Property cascade division, property parallel division and latch reduction are efficient enhancement techniques to model checking
- Equivalence checking applied efficiently in the verification of synthesized submodules

Conclusions (cont‘d)

Human effort (not including time for learning the tool and development of abstraction/enhancement techniques):

Project phases	**Time (man-days)**	**Code (# lines)**
RTL description	10	580 (Verilog)
Netslist description	3	647 (Verilog)
Simulation	3	102 (testbench)
Model checking	3	200 (env. mach.)
Equivalence checking	1	0
Total	20	1529

$\Rightarrow$ The human time for formal verification is almost the same as that for simulation in a design.

References

1. J. Lu and S. Tahar: Practical Approaches to the Automatic Verification of an ATM Switch Fabric using VIS; Proc. IEEE 8th Great Lakes Symposium on VLSI (GLS-VLSI'98), Lafayette, Louisiana, USA, February 1998, IEEE Computer Society Press, pp. 368-373.
2. J. Lu and S. Tahar: On the Formal Verification and Reimplementation of an ATM Switch Fabric Using VIS; Technical Report No. 401, Concordia University, Department of Electrical and Computer Engineering, September 1997.
3. Jianping Lu: On the Formal Verification of ATM Switches; MaSc Thesis, Concordia University, Department of Electrical and Computer Engineering, May 1999.

Technical report with source code available on-line at:

http://hvg.ece.concordia.ca/Publications/TECH_REP/VIS_TR97/VIS_TR97.html

other papers on ATM switch verification can be found at:

http://hvg.ece.concordia.ca/Research/APPL/atmsv.php