Synthesis from Waveform Transition Graphs

Alberto Moreno[1], Danil Sokolov[1], Jordi Cortadella[2]

[1] *School of Engineering, Newcastle University, UK*
[2] *Department of Computer Science, Universitat Politècnica de Catalunya, Spain*

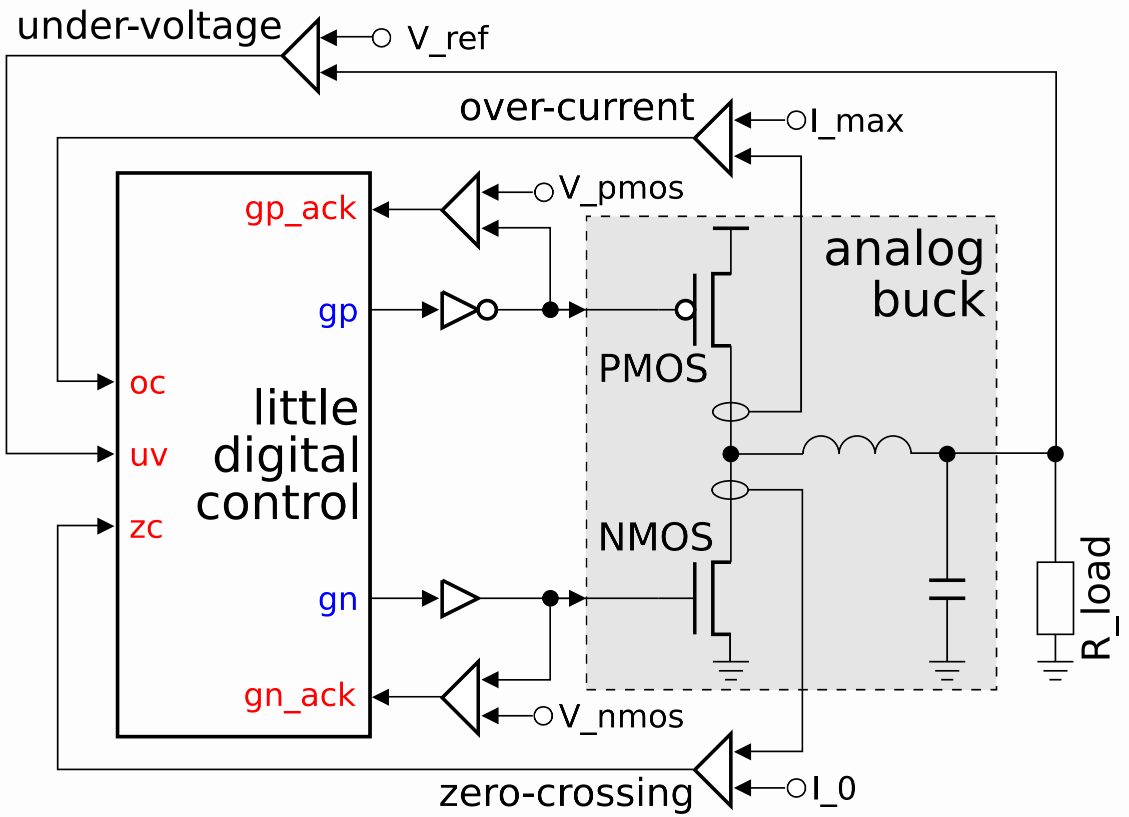
ASYNC 2019

Outline

- Motivation for yet another model
- Requirements from circuit designers
- Intuition for Waveform Transition Graphs
- Conversion to Signal Transition Graphs for synthesis and verification
- Design automation in Workcraft
- Examples and evaluation

Motivation: Application domain

- “Little digital” control – an ideal case for asynchronous design [1]
  - Relatively small controllers
  - Prompt reaction is paramount
  - Interface analog world
- Modelling aspects
  - Fine-grain control at the level of individual signals
  - Graph-based representation for causality, concurrency, and conflicts



[1] D. Sokolov et al. *“Automating the design of async. logic control for AMS electronics”* **IEEE TCAD**, 2019

Motivation: Limitations of existing models

- Signal Transition Graphs (STGs)

  ☺ Great expressive power and tool support

  ☹ Underlying Petri nets are unfamiliar to engineers

  ☹ Sophisticated modelling aspects (output persistency, input properness, non-commutativity, UCS/CSC conflicts, etc.)

- Burst Mode (BM) and eXtended BM (XBM) automata

  ☺ Engineers understand the underlying state machines

  ☹ Insufficient expressive power due to limited concurrency

- Generalized / Extended / Symbolic STGs

  ☹ Even more complex than STGs

  ☹ No mature tool support

Specification flow (industry perspective)

1. Sketch a waveform for intended circuit behaviour
2. Manually convert the waveform (or its fragment for one mode) to STG
3. Make sure that simulation of the STG resembles the sketch waveform
4. Repeat steps 2-3 for every distinctive mode of operation
5. Combine STGs for all modes in a state machine-like structure
6. Try hard to resolve all the STG implementability issues (inconsistency, irreducible encoding conflicts, non-persistency, etc.)

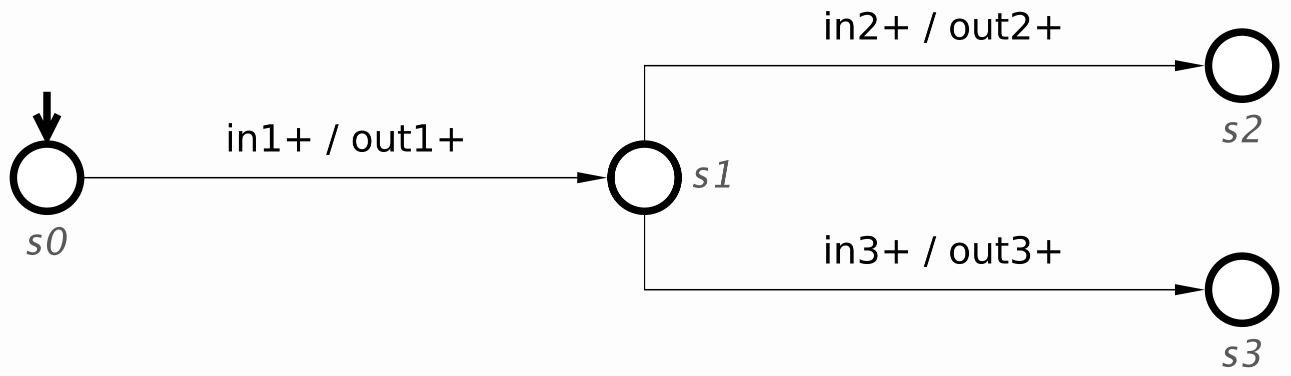
- How to express destabilisation/stabilisation of input signals?
- How to select the mode of operation based on signal levels?
- Can this flow be simplified and automated?

Usability requirements for a new model

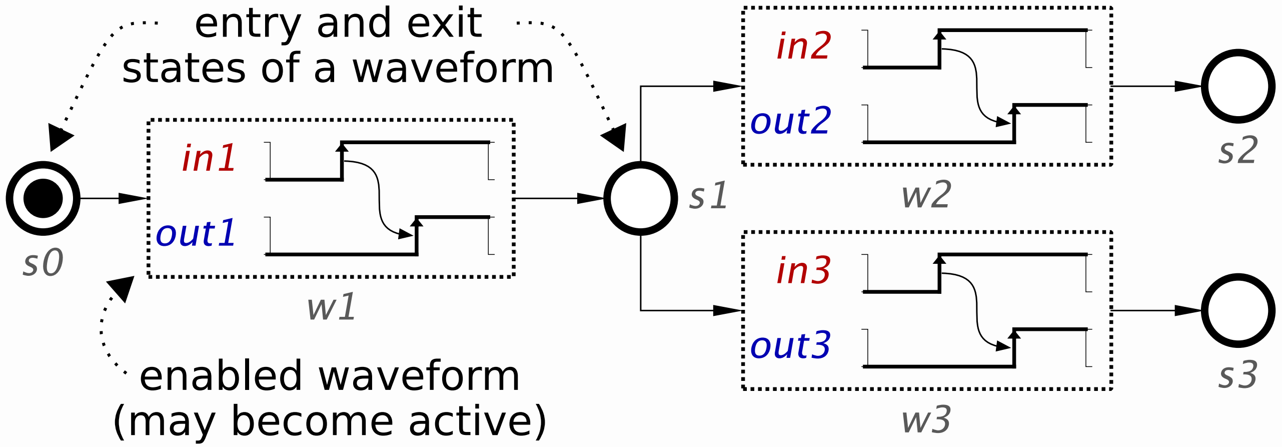
- State machine to express high-level modes of operation
  - Choice is restricted to state machine level
  - Current state is represented by a single token
- Waveforms to capture partial order of signals in each mode
  - Concurrency is contained within waveforms
  - At most one waveform is active at a time
- Advanced features for input signals
  - Unstable (don't care) and undefined (stable but unknown) states
- Flexibility in modelling of choice
  - Edge-sensitive and level-sensitive

Intuition for Waveform Transition Graphs

- Burst Mode automaton: state machine + input/output bursts



- WTG: state machine whose arcs are waveforms

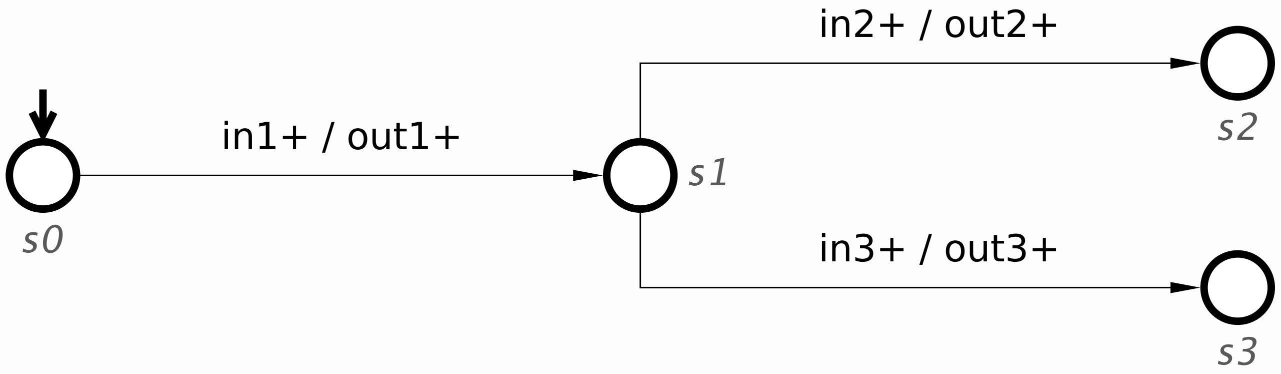


Enabled waveform activation:

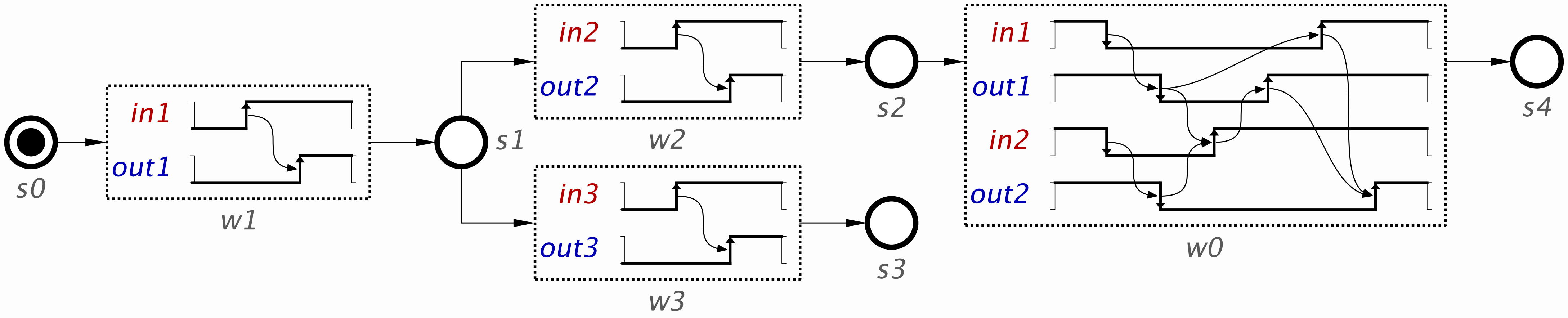
- Consume a token from the entry state
- Execute all its events
- Produce token at the exit state

Intuition for Waveform Transition Graphs

- Burst Mode automaton: state machine + input/output bursts

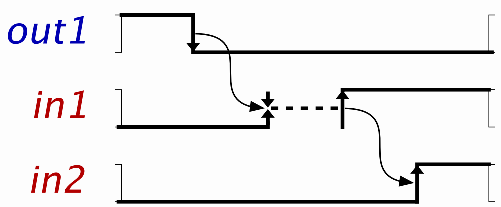


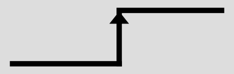
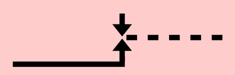
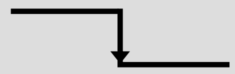
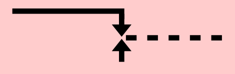
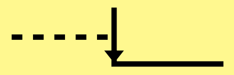
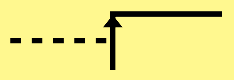
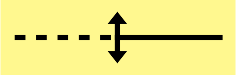
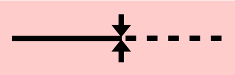
- WTG: state machine whose arcs are waveforms



Advanced features for signals

- Unstable inputs via destabilise/stabilise events
- Stabilise to low, high or unknown state



		to state			
		low	high	unstable	stable
from state	low				
	high				
	unstable				
	stable				

Legend:

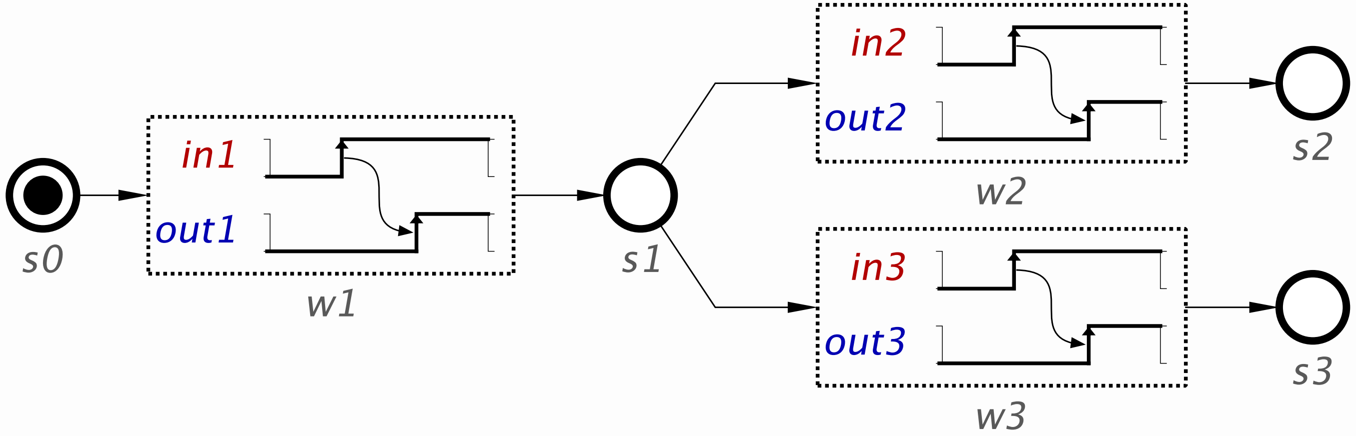
conventional rise/fall events

destabilise events

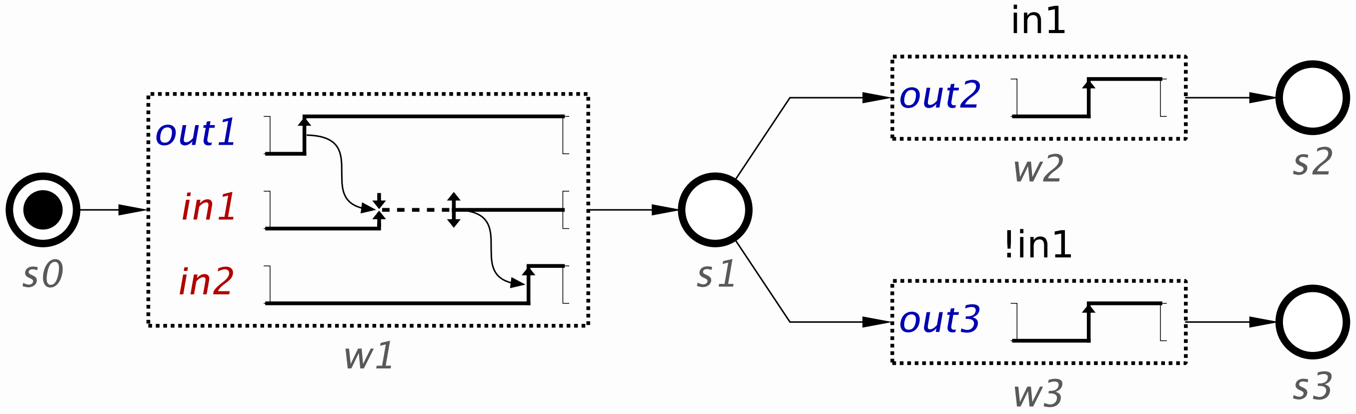
stabilise events

Flexibility in modelling of choice

- Edge-sensitive choice

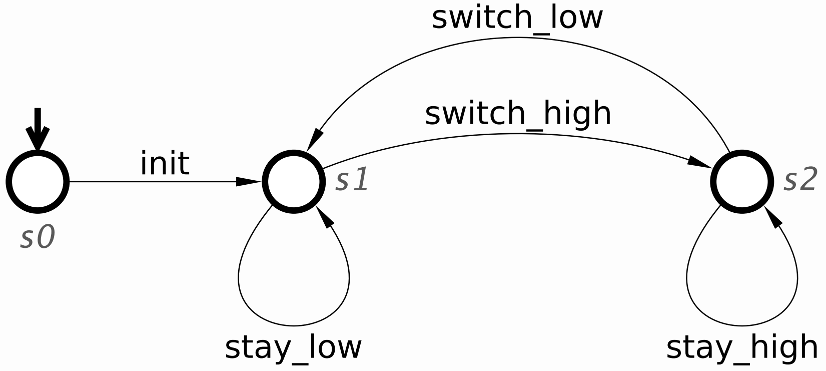


- Level-sensitive choice

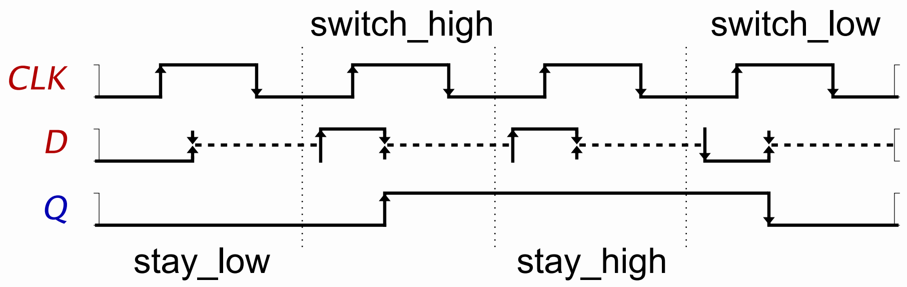


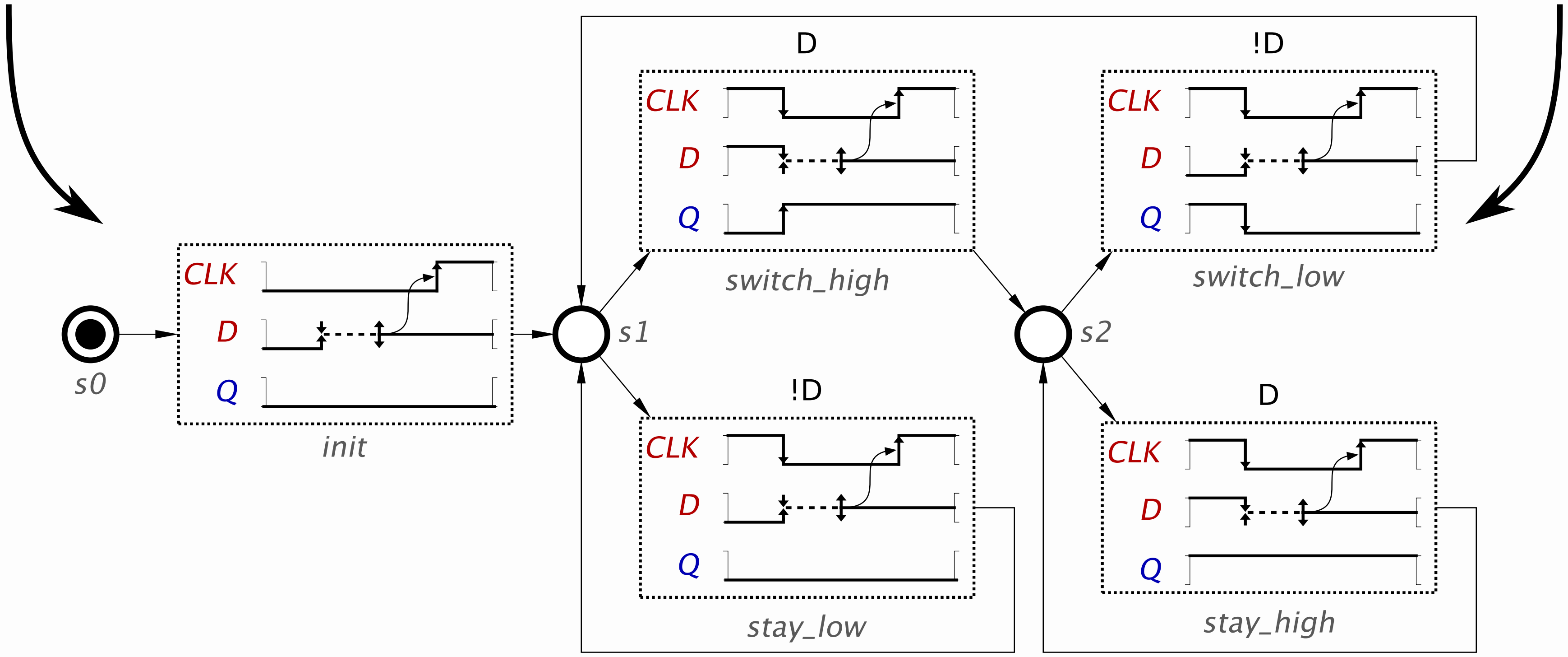
D flip-flop example

- High-level state machine



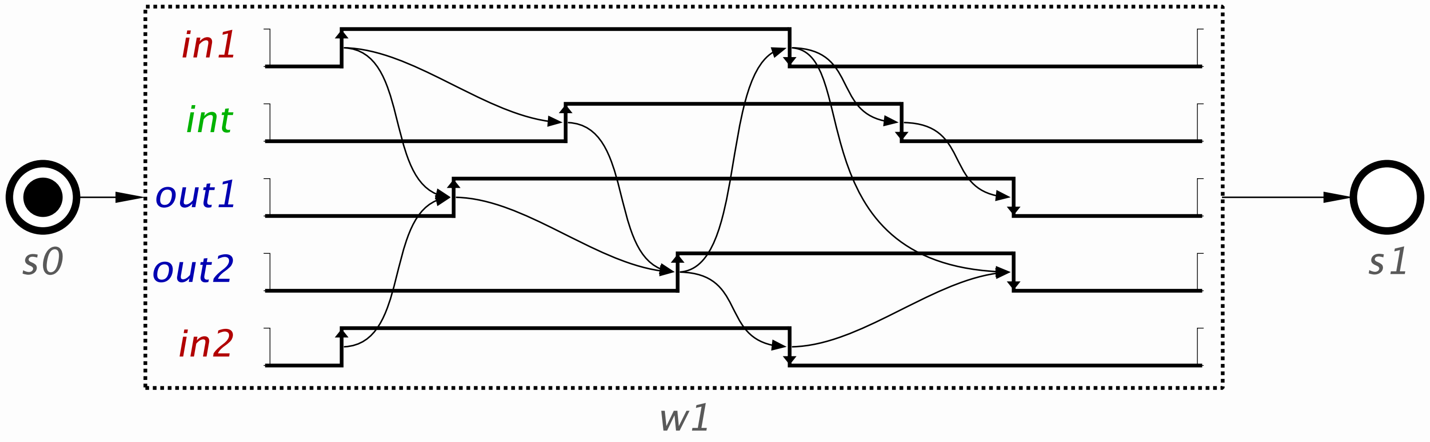
- Possible trace waveform



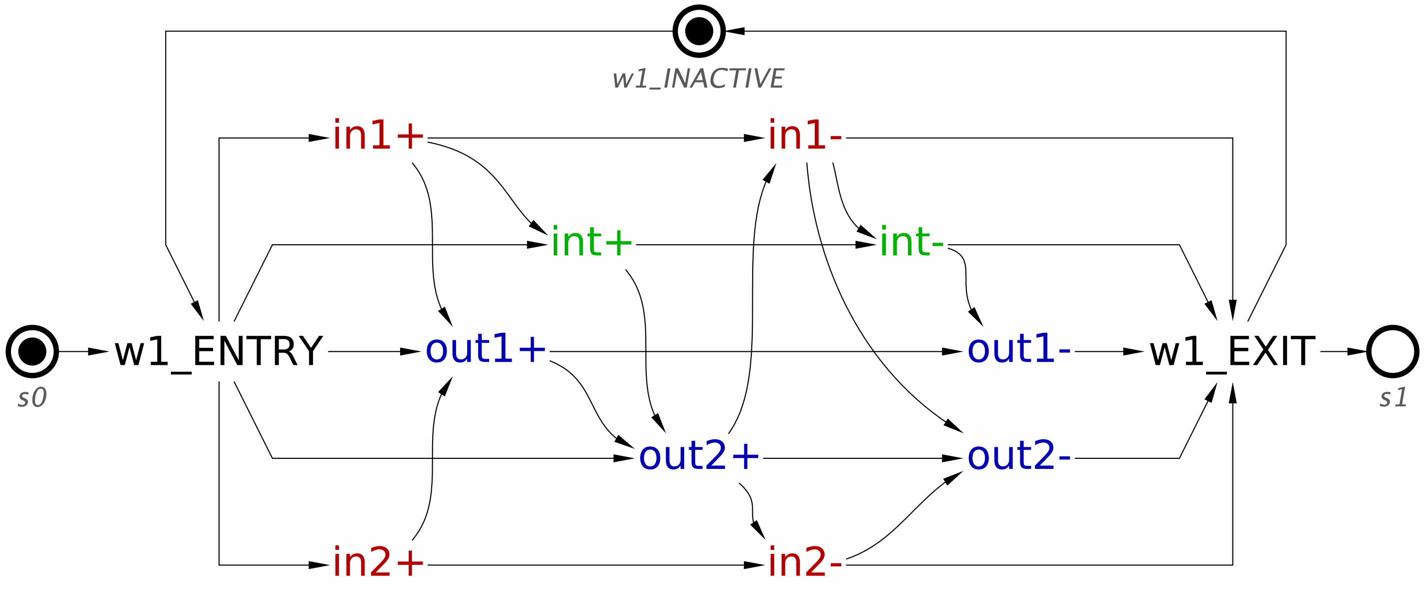


WTG to STG conversion: Simple waveform

- WTG fragment

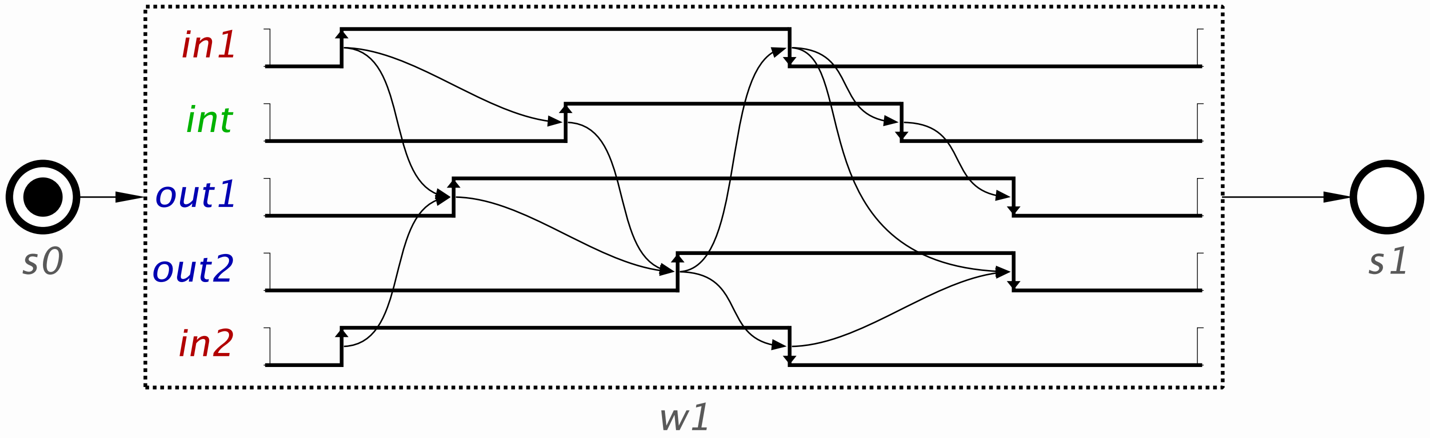


- STG fragment – one-to-one mapping

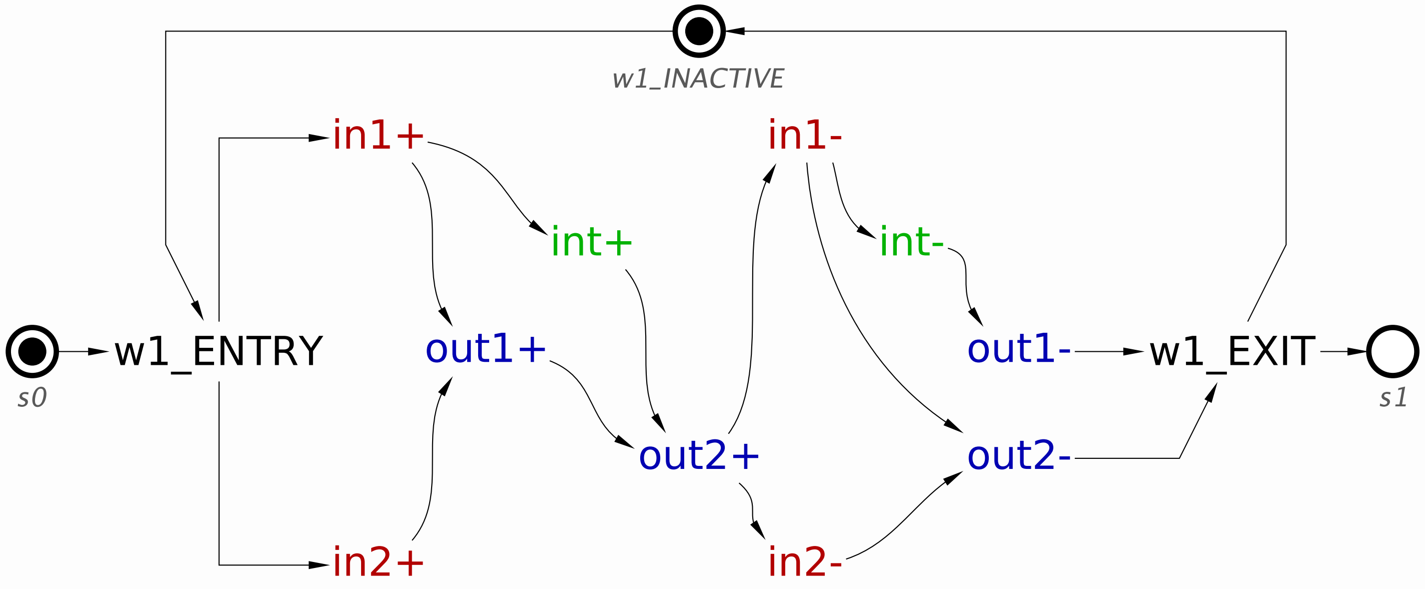


WTG to STG conversion: Simple waveform

- WTG fragment

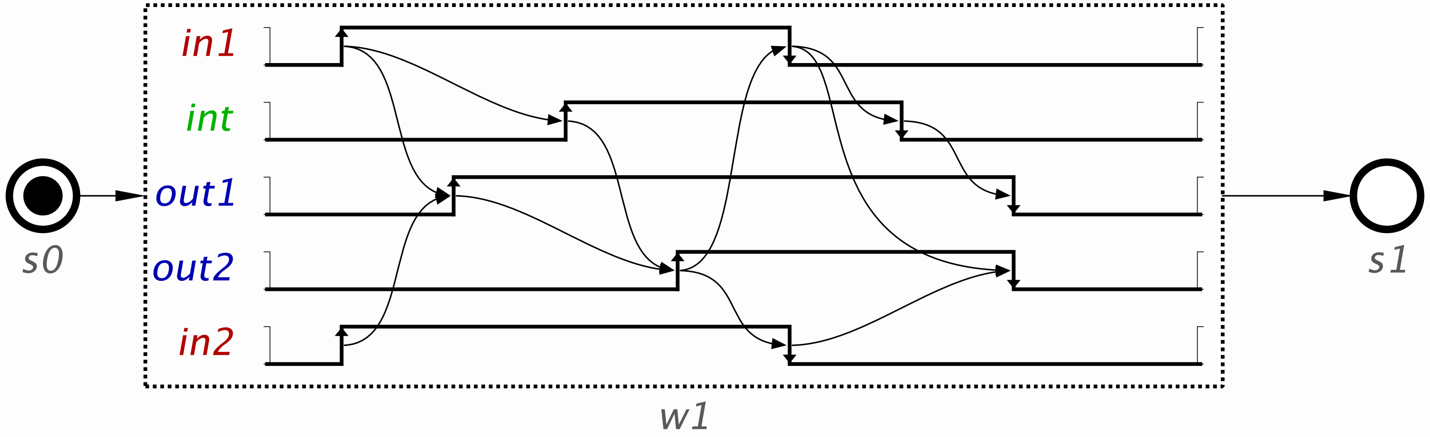


- STG fragment – redundant arcs removed

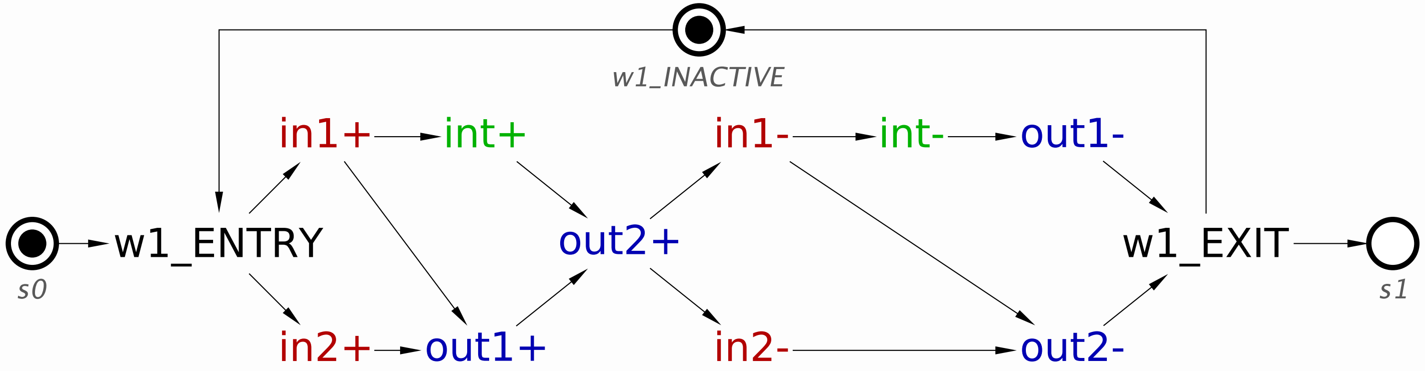


WTG to STG conversion: Simple waveform

- WTG fragment

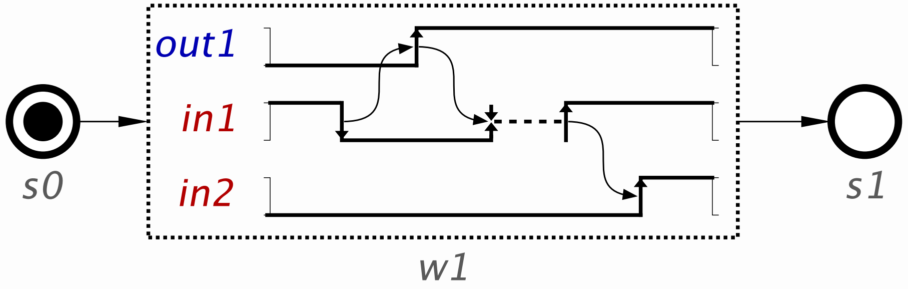


- STG fragment – rearranged layout

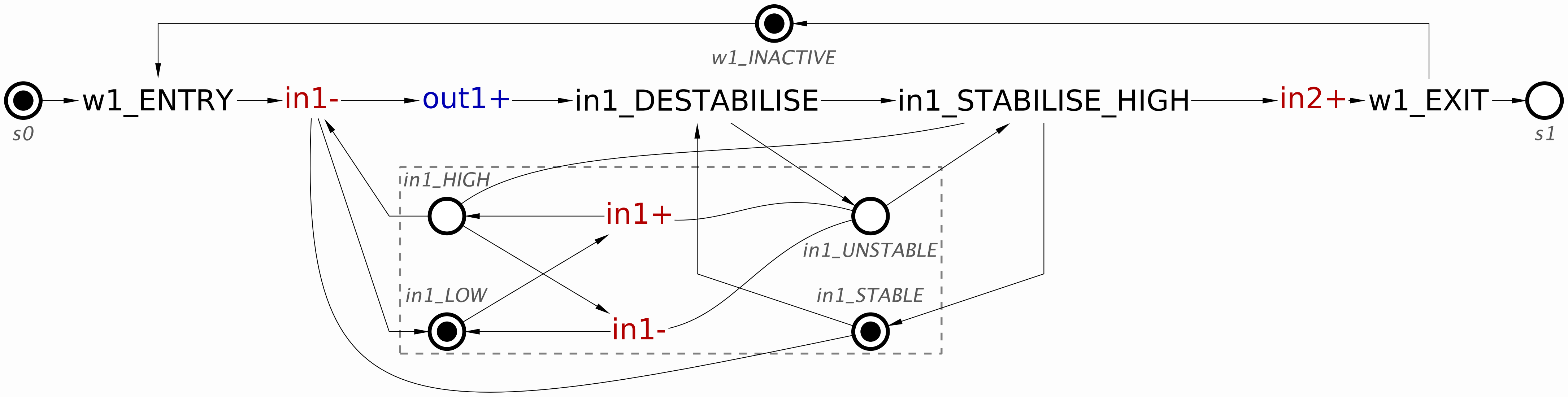


WTG to STG conversion: Stabilise at HIGH/LOW state

- WTG fragment

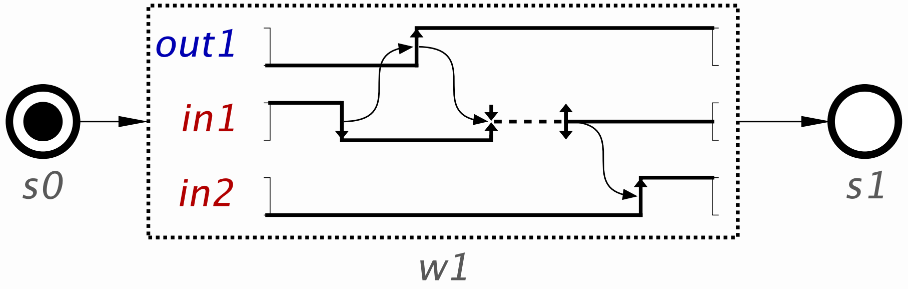


- STG fragment

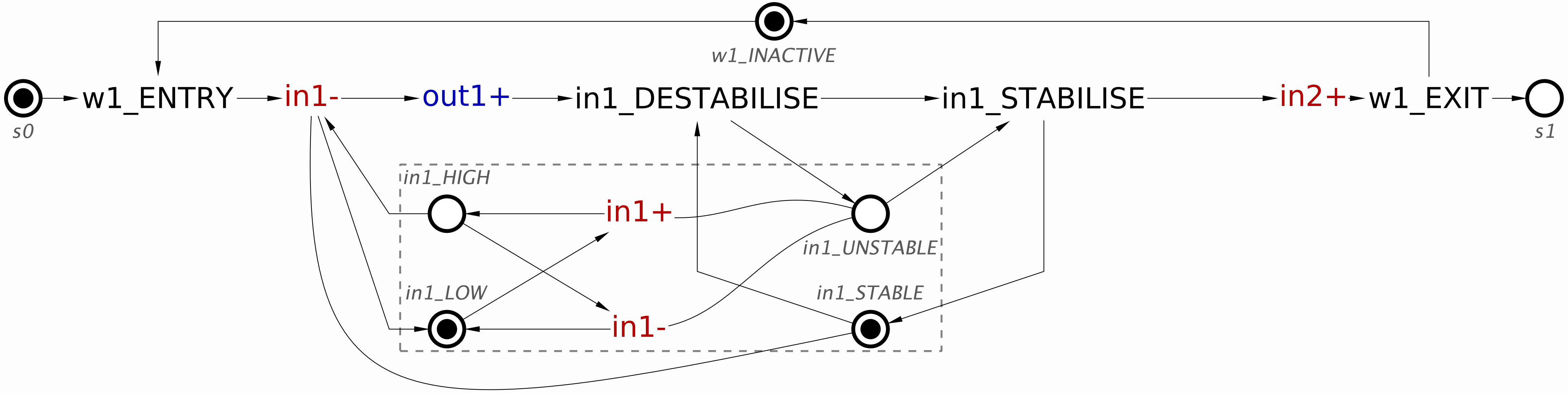


WTG to STG conversion: Stabilise at unknown state

- WTG fragment

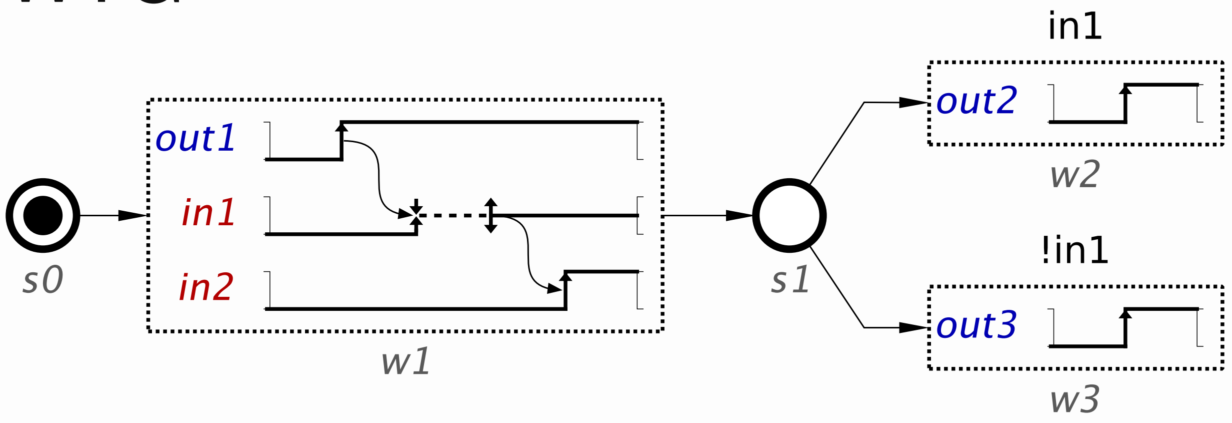


- STG fragment

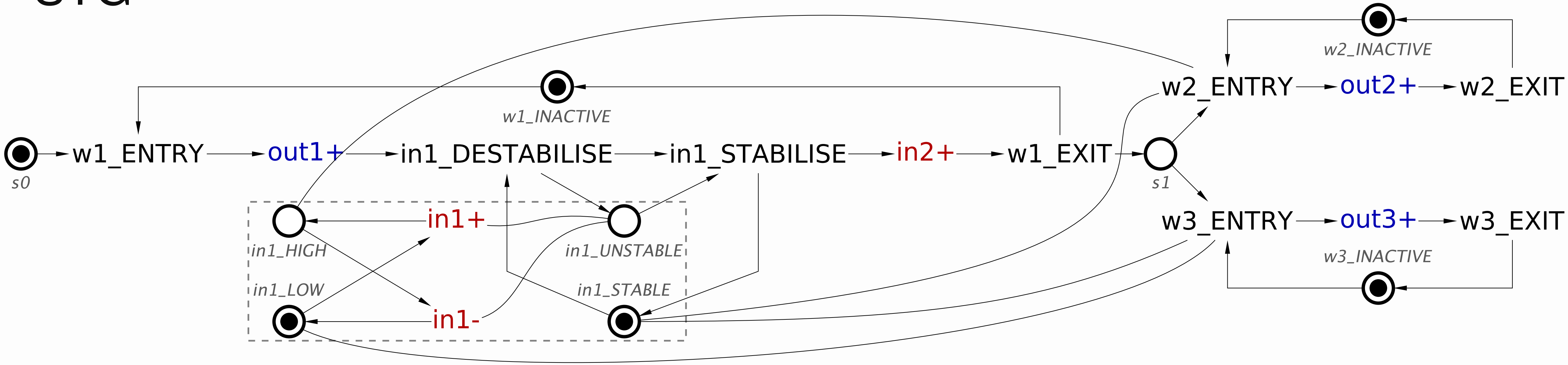


WTG to STG conversion: Guards in level-sensitive choice

- WTG



- STG

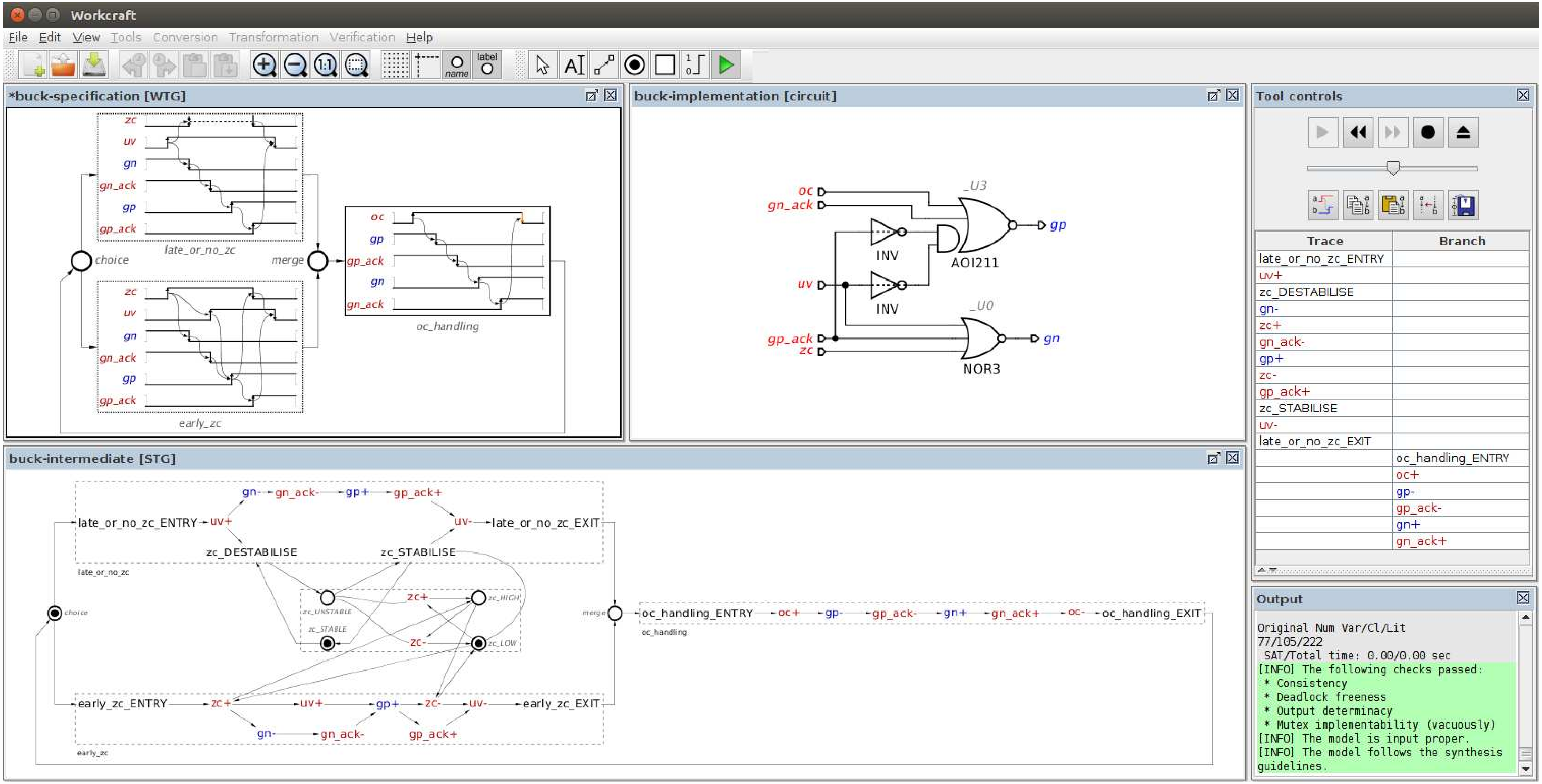


Design automation in WORKCRAFT

- Support for capturing and simulating WTGs
- Local structural checks to ensure implementability
  - Consistency of signals between waveforms
  - Output-persistency and output-determinacy at choice states
  - See the paper for more details
- Automatic conversion to STGs as backend representation
- Reuse existing methods and tools
  - Formal verification of specification (Punf + MPSat)
  - Logic synthesis of circuit implementation (Petrify, MPSat, ATACS)
- Backtracking for communication of problems

Output-persistency: enabled output must not be disabled by another signal
Output-determinacy: if an output is enabled by a sequence of events then all executions
of this trace must enable the same output

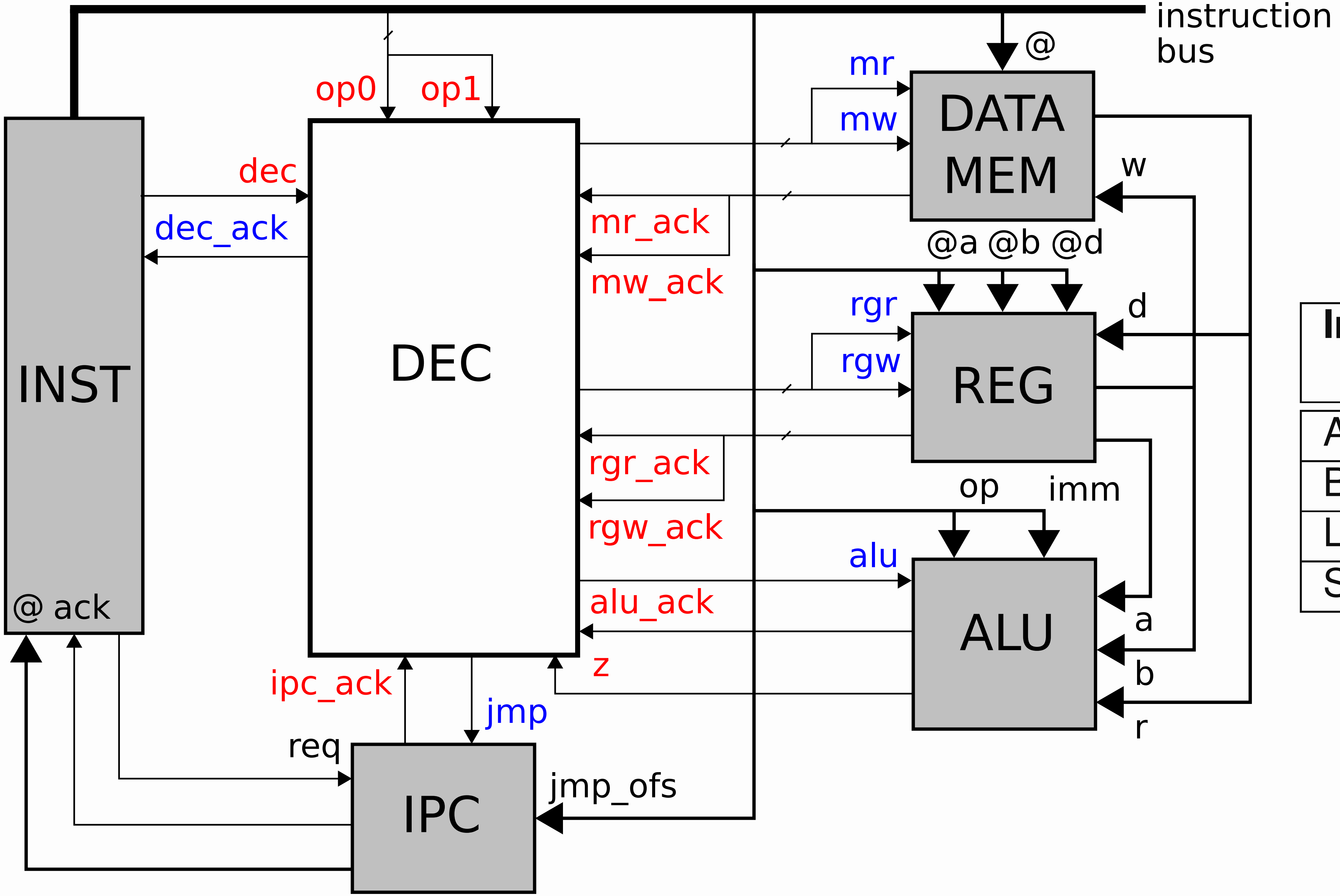
Design automation in WORKCRAFT



Trace	Branch
late_or_no_zc_ENTRY	
uv+	
zc_DESTABILISE	
gn-	
zc+	
gn_ack-	
gp+	
zc-	
gp_ack+	
zc_STABILISE	
uv-	
late_or_no_zc_EXIT	
	oc_handling_ENTRY
	oc+
	gp-
	gp_ack-
	gn+
	gn_ack+

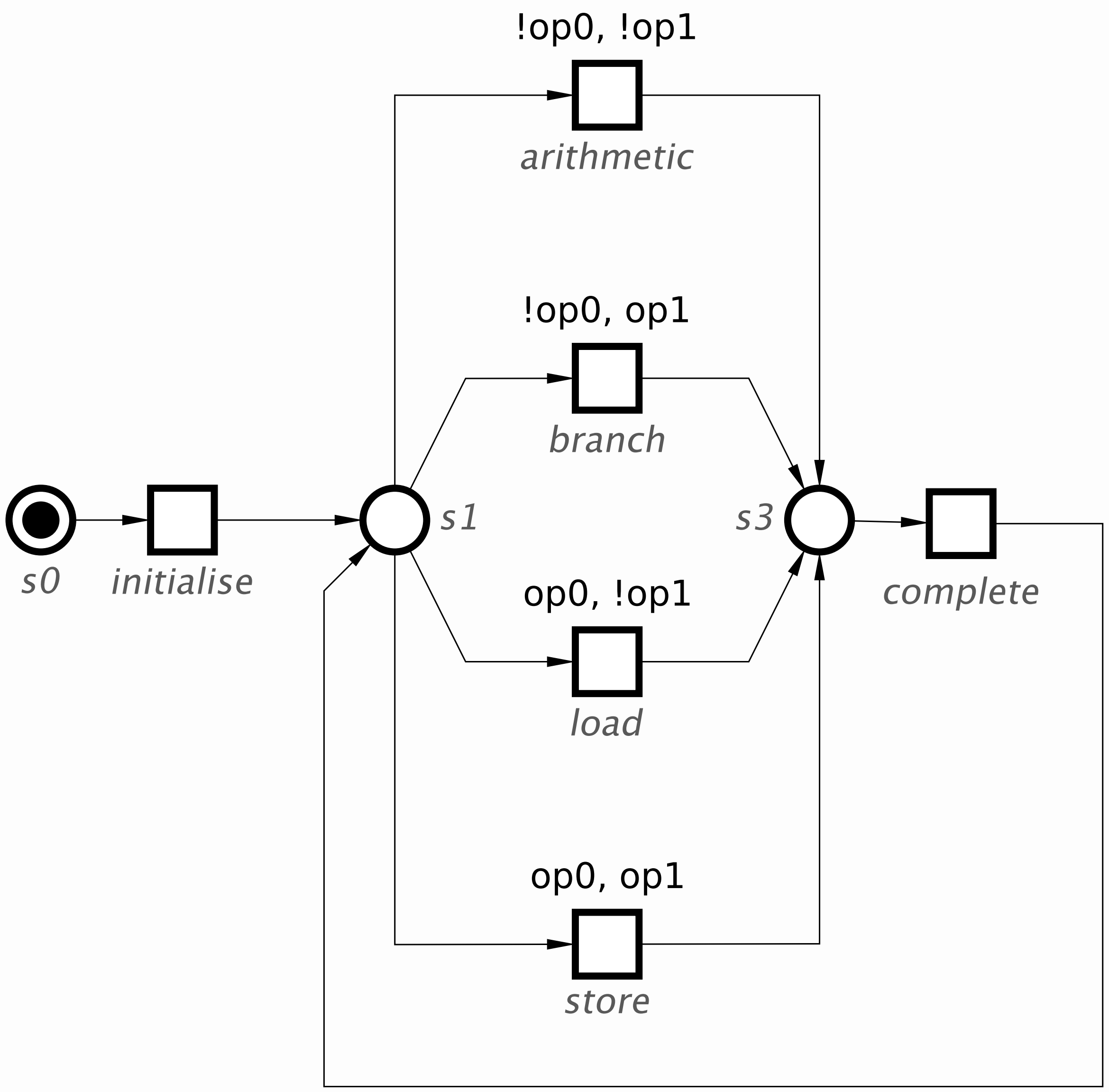
```
Original Num Var/Cl/Lit
77/105/222
 SAT/Total time: 0.00/0.00 sec
[INFO] The following checks passed:
 * Consistency
 * Deadlock freeness
 * Output determinacy
 * Mutex implementability (vacuously)
[INFO] The model is input proper.
[INFO] The model follows the synthesis
guidelines.
```

Instruction decoder example: Block diagram



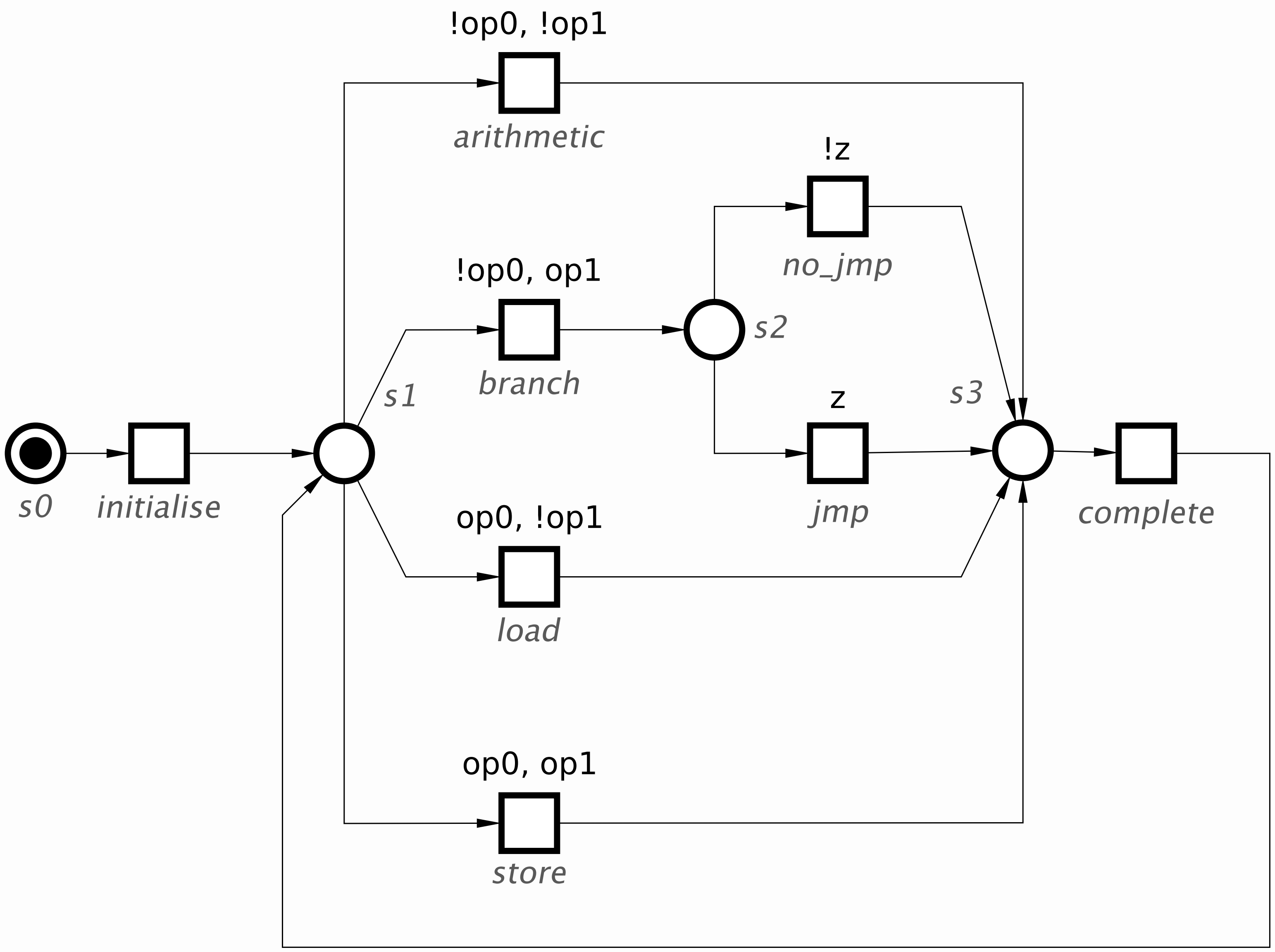
Instruction class	Opcode op0,op1
Arithmetic	0,0
Branch	0,1
Load	1,0
Store	1,1

Instruction decoder example: High-level state machine

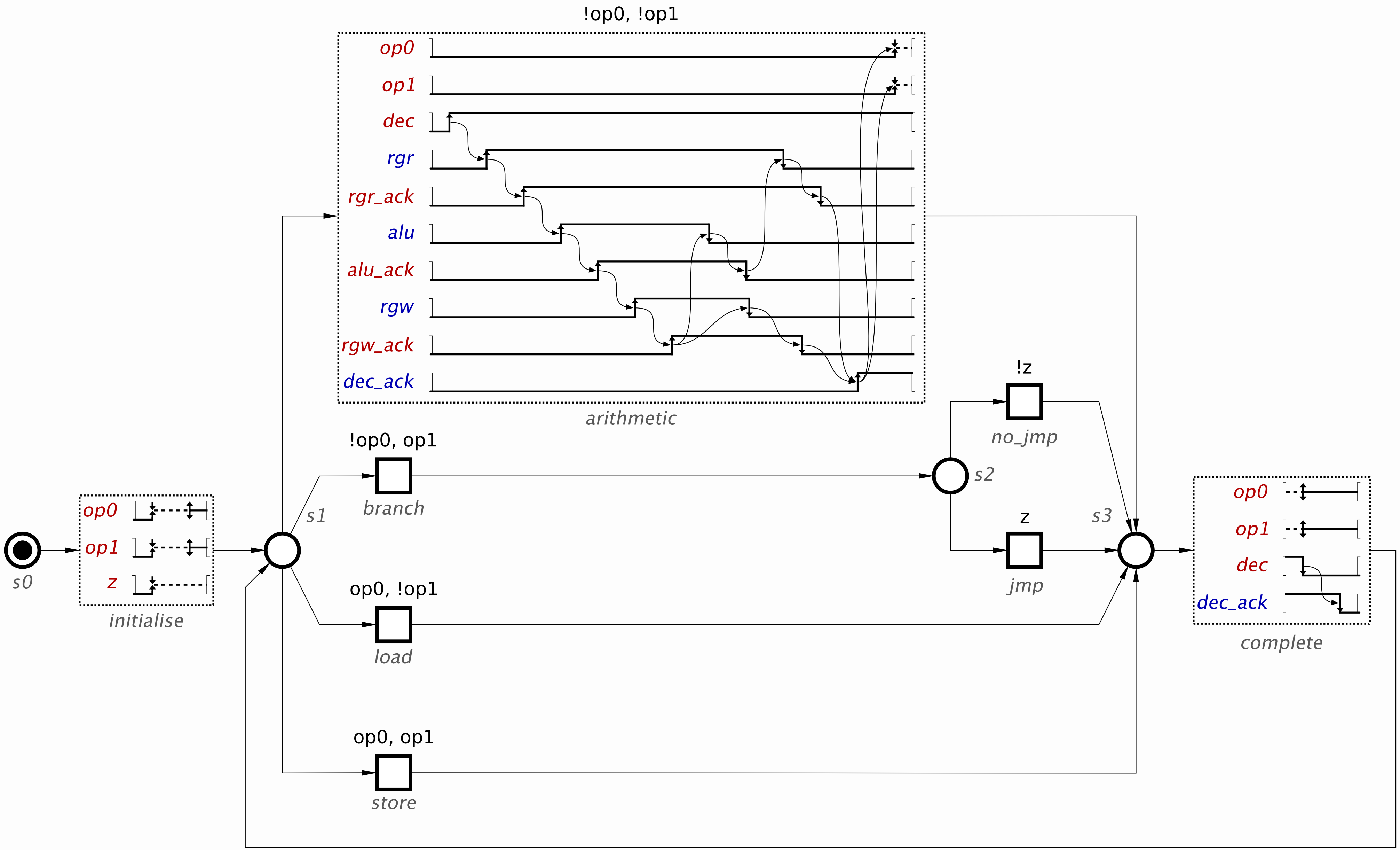


Instruction class	**Opcode op0,op1**
Arithmetic	0,0
Branch	0,1
Load	1,0
Store	1,1

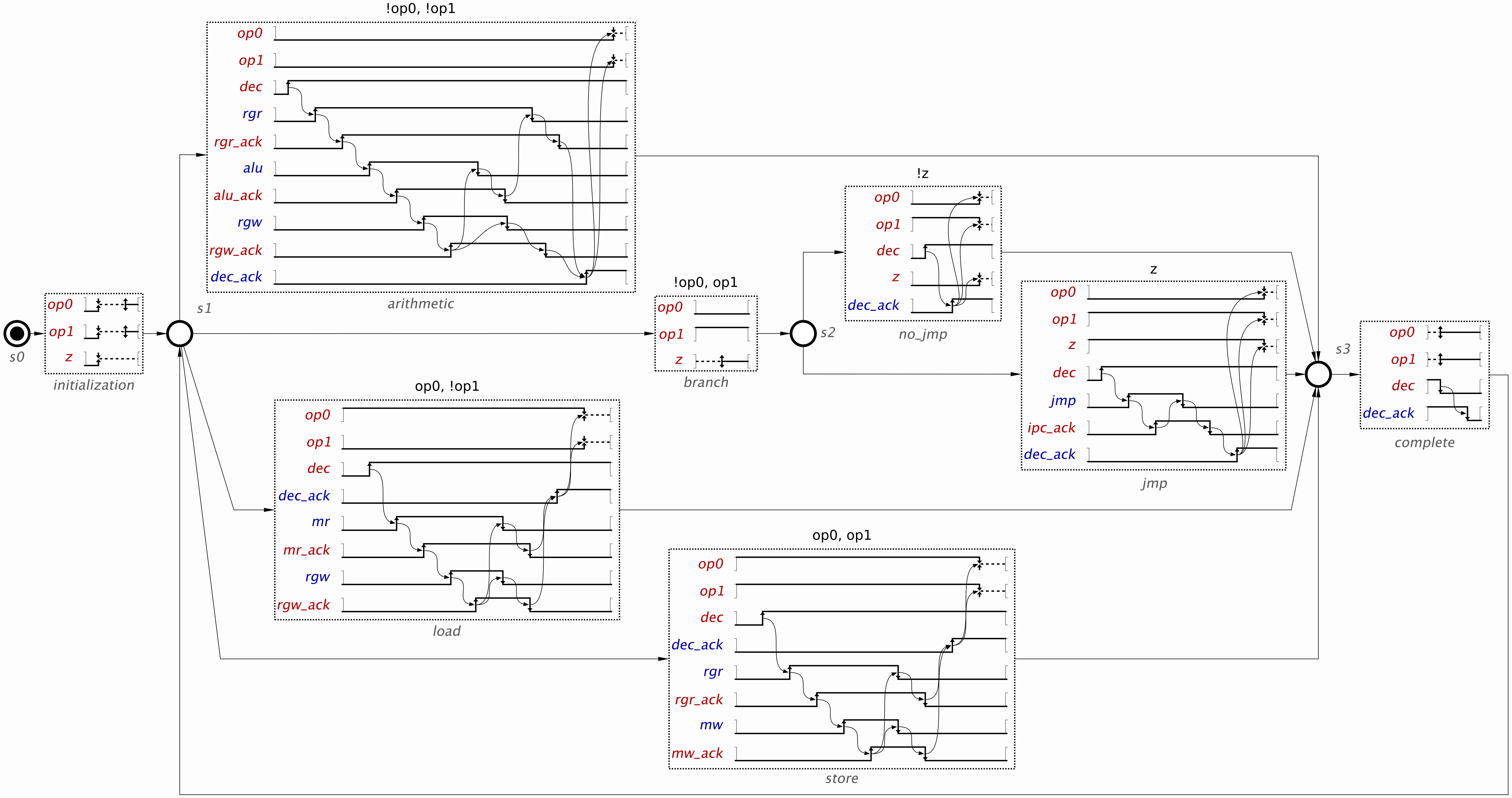
Instruction decoder example: High-level state machine



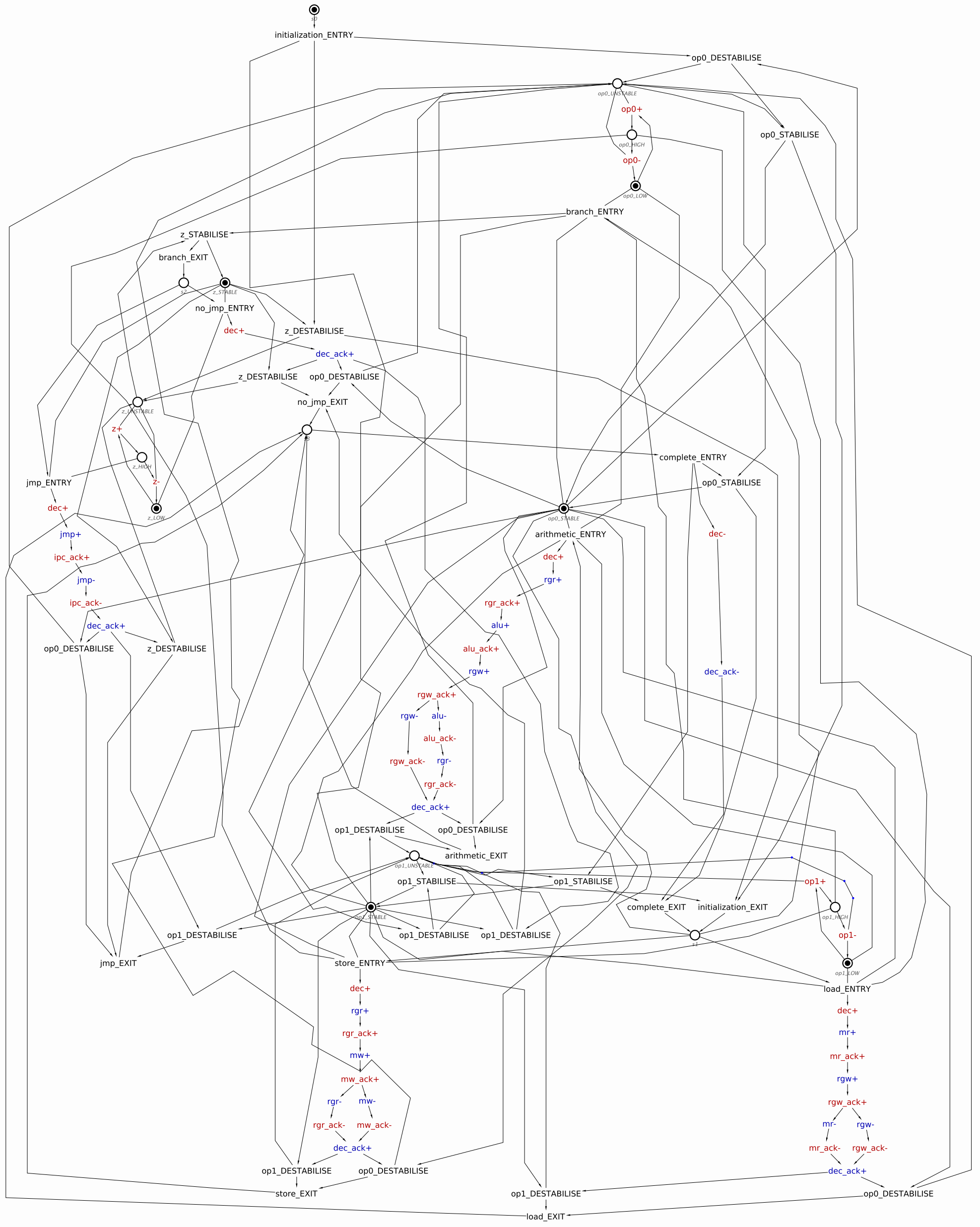
Instruction decoder example: High-level state machine

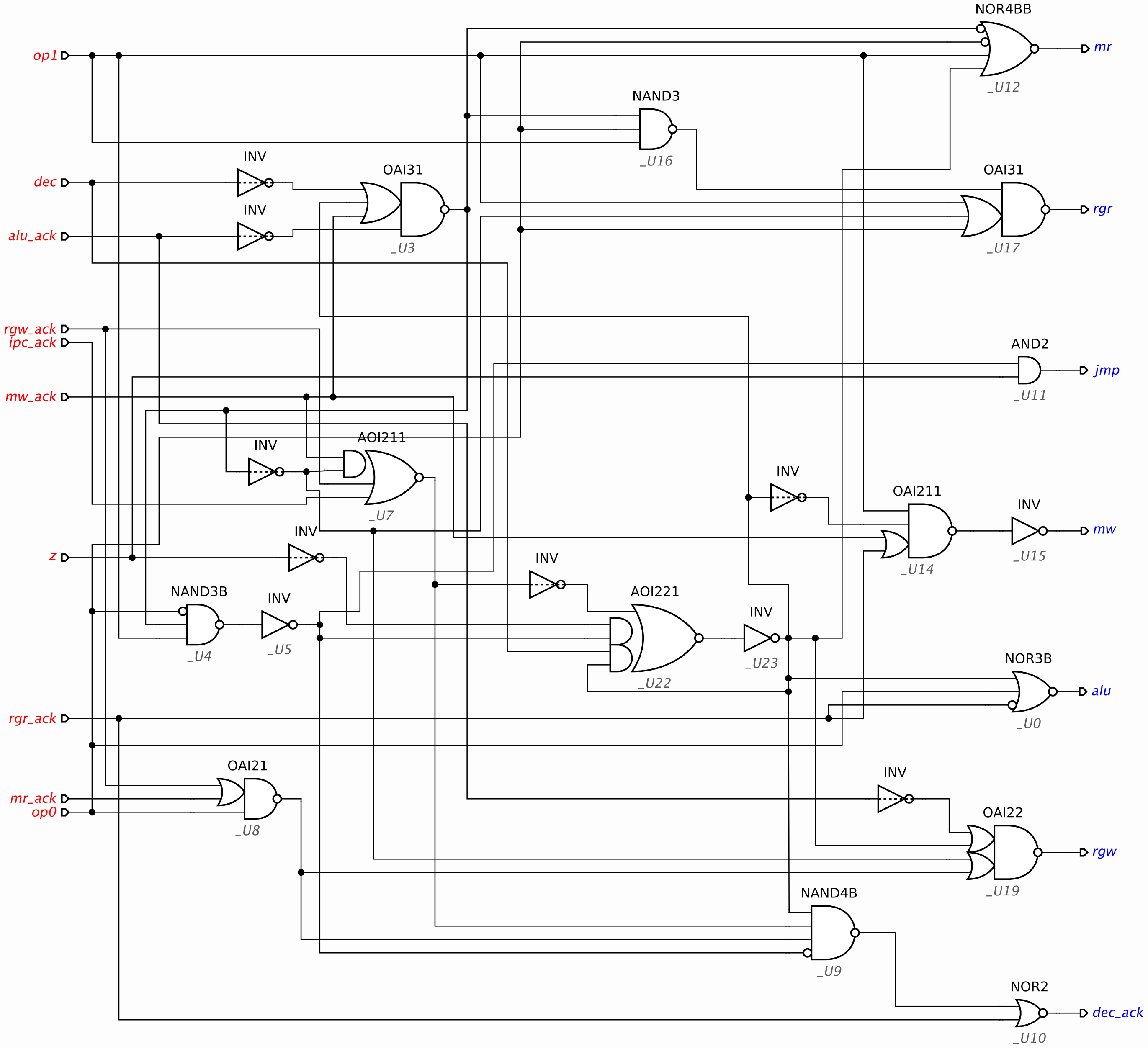


Instruction decoder example: Complete WTG



Instruction decoder example: STG and SI circuit



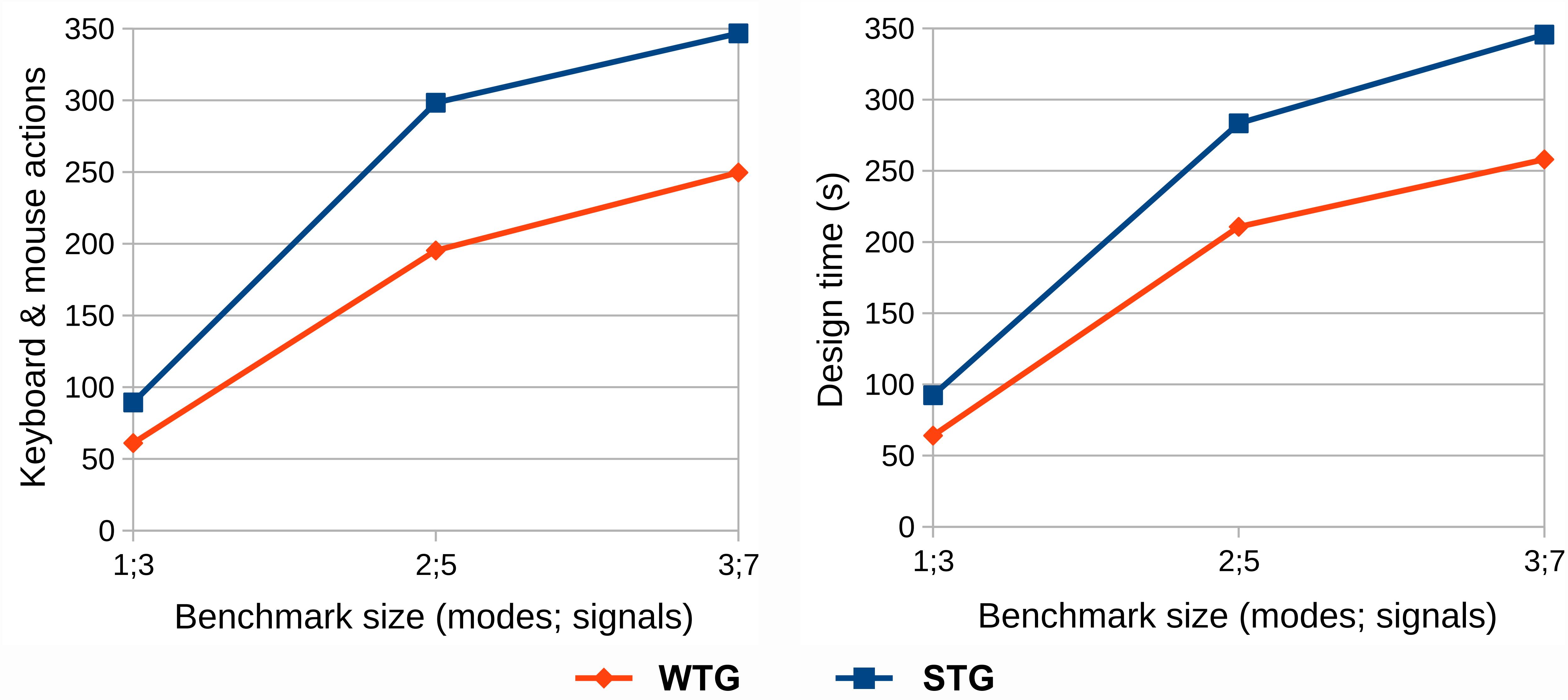


Productivity: WTG vs STG

Benchmark	Size		User	Input actions			Design time (s)		
	mode	signal		STG	WTG	Impr.	STG	WTG	Impr.
C-element	1	3	A	104	73	32%	118	86	31%
			B	96	61		112	71	
			C	68	49		47	35	
VME bus controller	2	5	A	262	199	35%	262	238	26%
			B	302	205		320	257	
			C	331	182		268	137	
Buck controller	3	7	A	338	227	28%	295	260	25%
			B	320	279		462	320	
			C	382	243		280	194	
Total				2,203	1,518	31%	2,164	1,598	26%

Productivity: WTG vs STG

Average data for 3 users with different experience: **>25%** productivity improvement



Conclusions

- WTGs model
  - Based on familiar modelling abstractions
  - Explicit separation of choice and concurrency aspects
  - Simpler than STGs and more expressive than XBM automata
  - Support for unstable signals via destabilise/stabilise events
  - Edge-sensitive and level-sensitive choice
- WTGs design automation
  - Design flow supported in Workcraft (`https://workcraft.org/`)
  - 25% productivity improvement compared to STGs
  - STG translation for reuse of synthesis and verification tools