

Initialisation of Asynchronous Circuits

Danil Sokolov, Victor Khomenko, Alex Yakovlev

Newcastle University, UK

Introduction

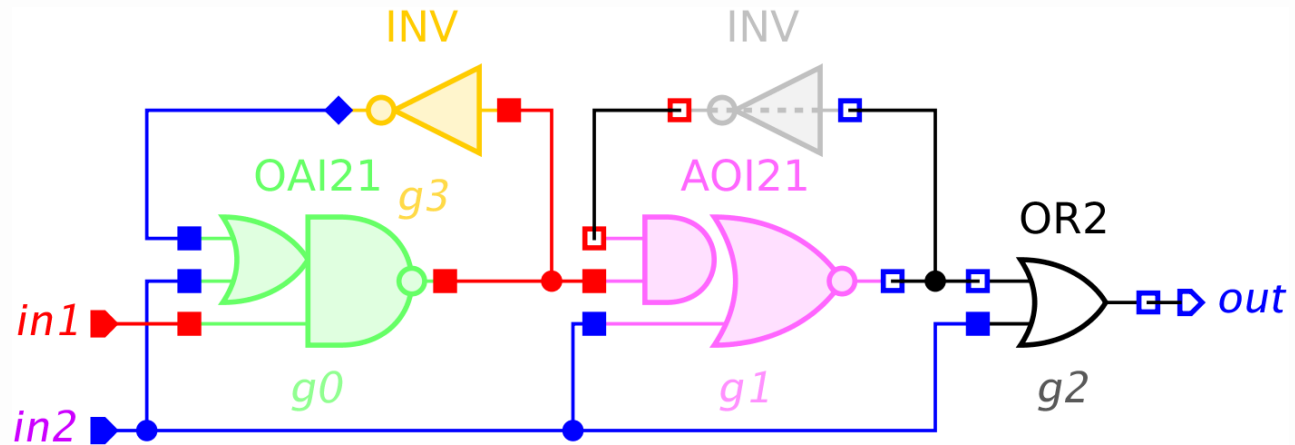
- Speed-independent (SI) synthesis does not insert reset logic
- Initialisation phase does not have to be SI
- Initialisation via an externally generated **reset** signal (e.g. active-low)
 - **reset** is initially low, sufficiently long to complete initialisation of all gates
 - **reset** eventually goes high and normal SI operation begins
 - **reset** stays high for the whole time of circuit normal operation
- Ways to initialise a circuit (can be used in combination)
 - Rely on the initial state of some of the inputs
 - Substitute some gates with “resetable” alternatives
 - Insert additional gates to explicitly initialise the internal and output signals (they act as buffers during normal operation, so be careful with isochronic forks)
- Need for design automation

Circuit initialisation in WORKCRAFT

- **Init to one** property (Boolean flag)
 - Defines the expected initial state of the signal
 - Automatically assigned if a circuit is synthesised by one of the backend tools
 - Designer responsibility if the circuit is manually altered
- **Force init** property (Boolean flag)
 - Defines if the signal is known to be in a correct initial state
 - Primary input – environment takes care of initialising it to the expected state
 - Component output – necessary circuitry will be added to properly initialise that pin
- Propagation of the initialisation state
 - Signals whose **Forced init** property is set are *initialised* (*others are uninitialised*)
 - Try to evaluate uninitialised signals using **Init to one** property of initialised signals
 - If the Boolean value of a signal can be derived, then it has *propagated* initial state and the signal is also considered initialised
 - Repeat evaluation of uninitialised signals until no further progress can be made

Initialisation analyser tool

- Highlighting gates initialisation
- Indicates pins initial state



- Toggle **Force init** property by clicking input ports and output pins (or gates)
- Changing **Force init** for groups of signals
- Heuristic-based complete initialisation
- Automatic insertion of reset logic (active-low or active-high reset)



Tool controls

Gate highlight legend

- Unknown initial state
- Don't touch zero delay
- Problem of initialisation
- Forced initial state
- Propagated initial state

Pin initial state:

- Expected high / low
- Propagated high / low
- Forced high / low

ENV $f(i) \neq 0$  + - 

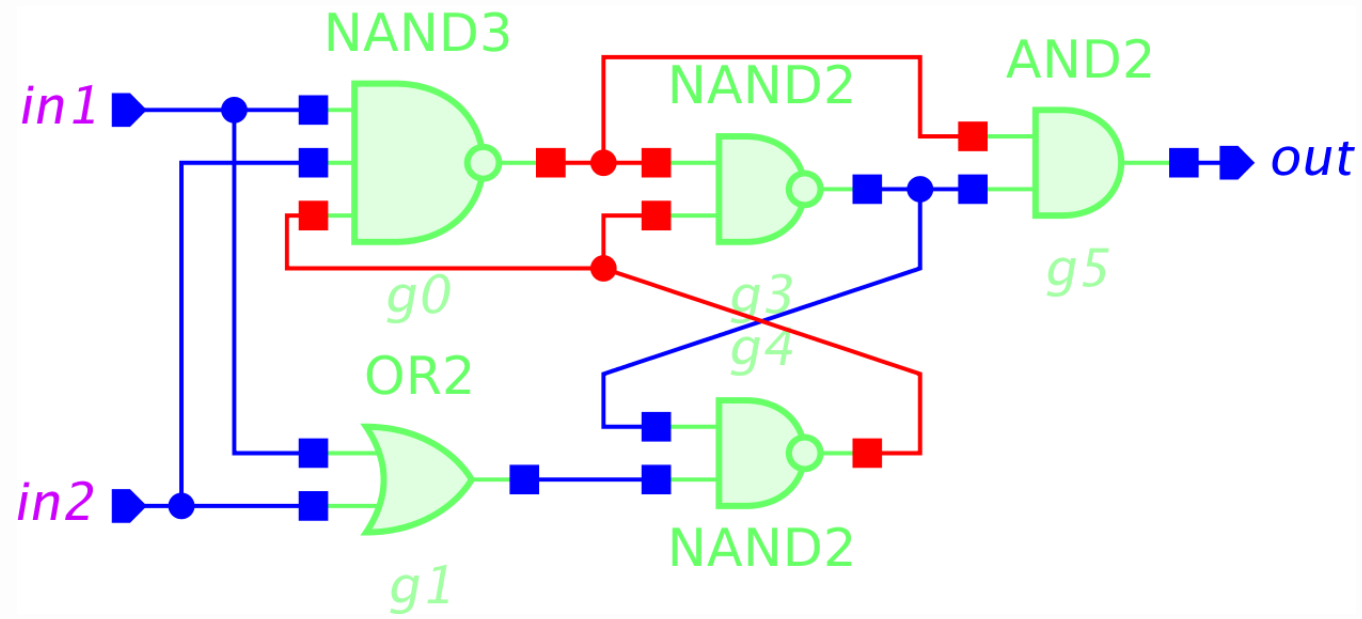
Force init pins

g3.ON

Insert reset (active-high) Insert reset (active-low)

Initialisation via primary inputs

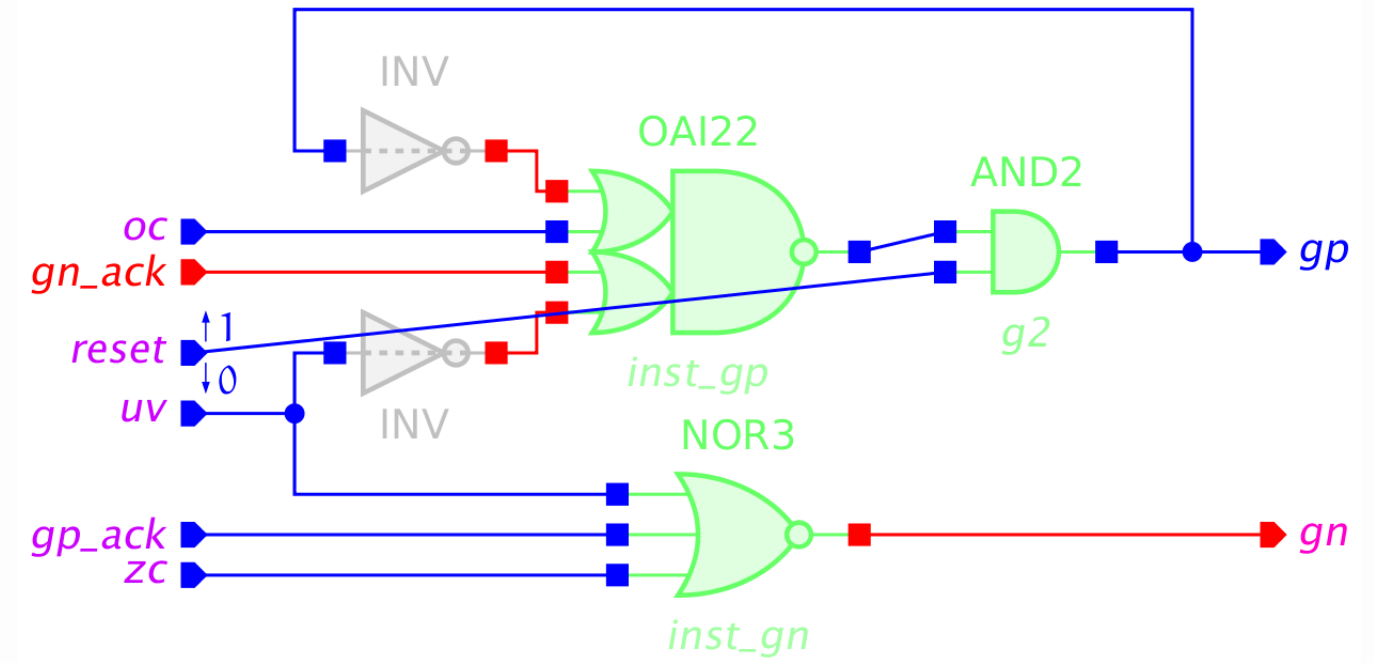
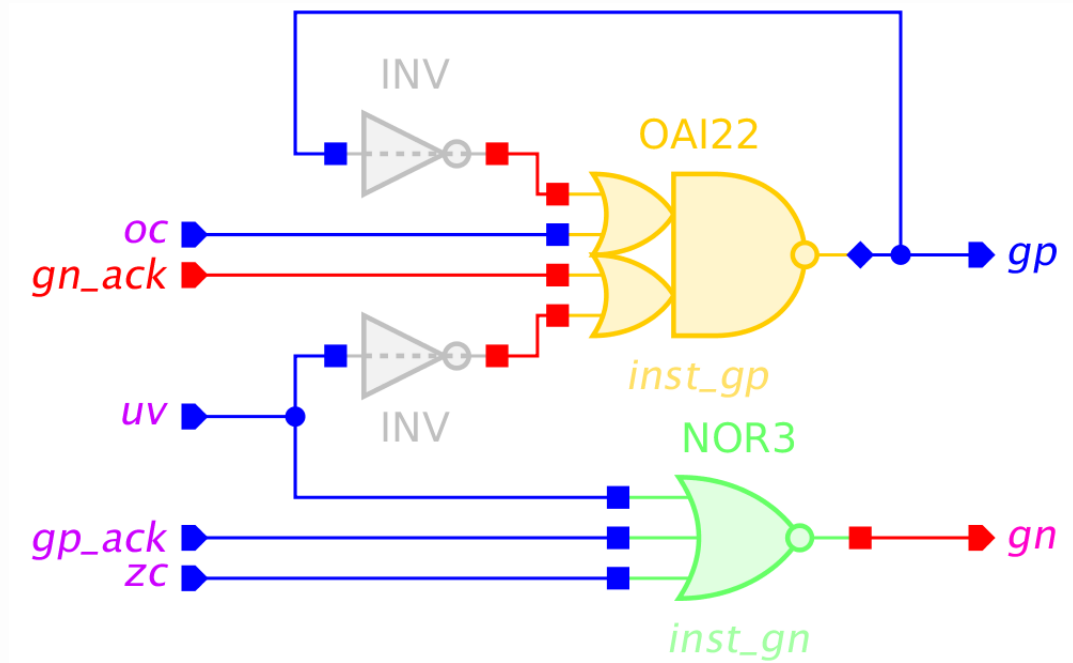
- Demo: *celement-decomposed.circuit.work*



- Sufficient to force the primary inputs to their initial state for complete initialisation

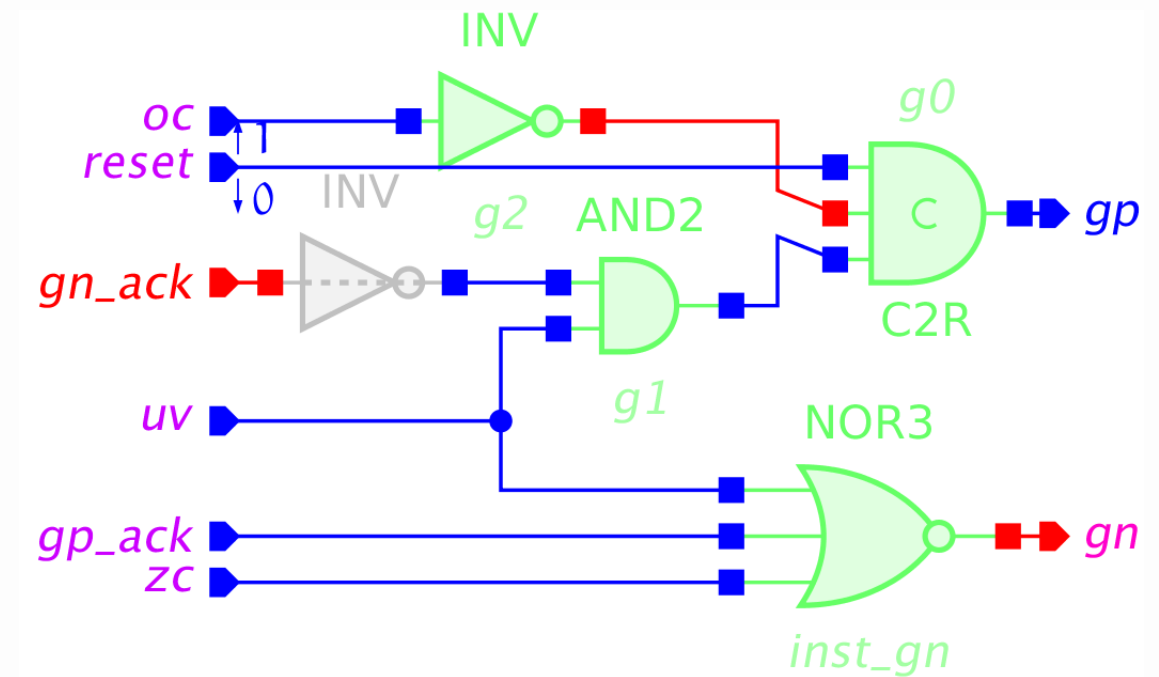
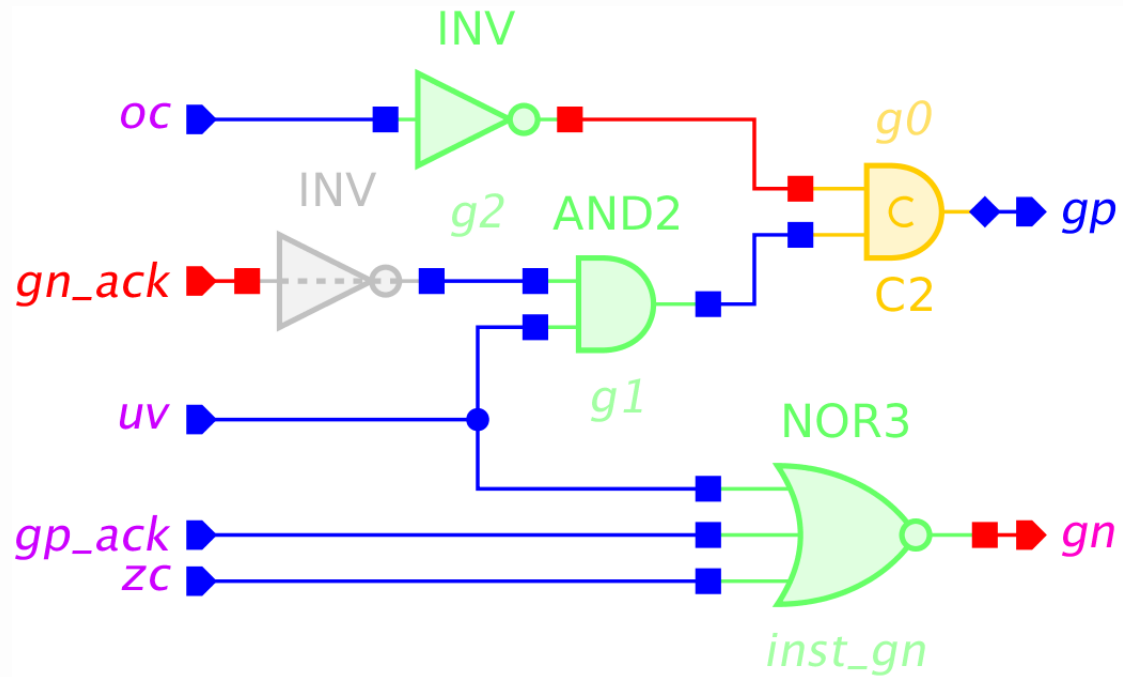
Initialising combinational loops

- Demo: *buck-feedback.circuit.work*



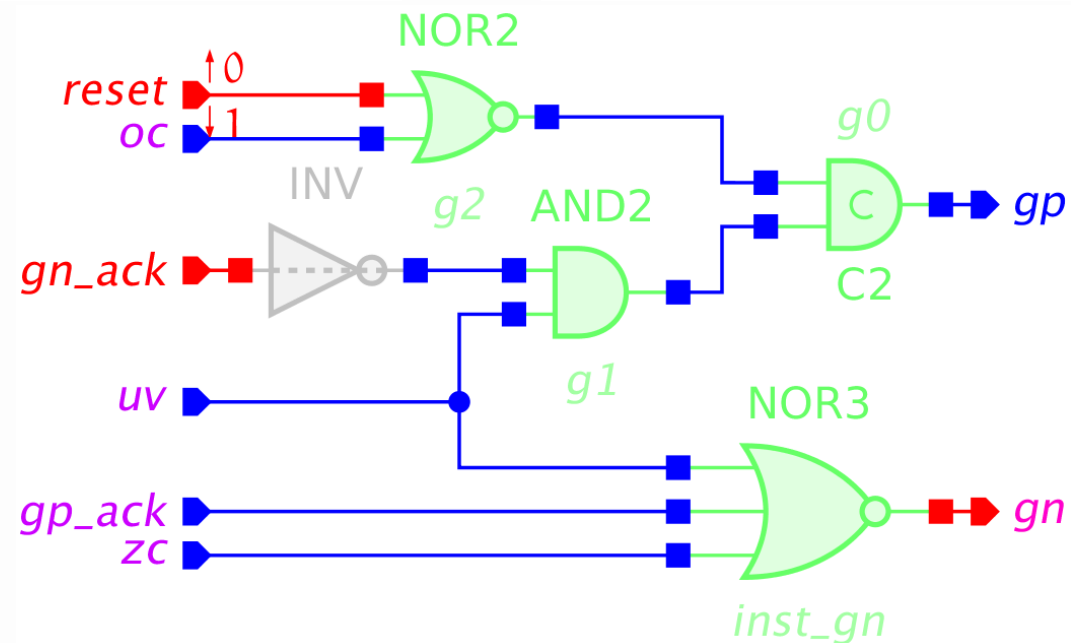
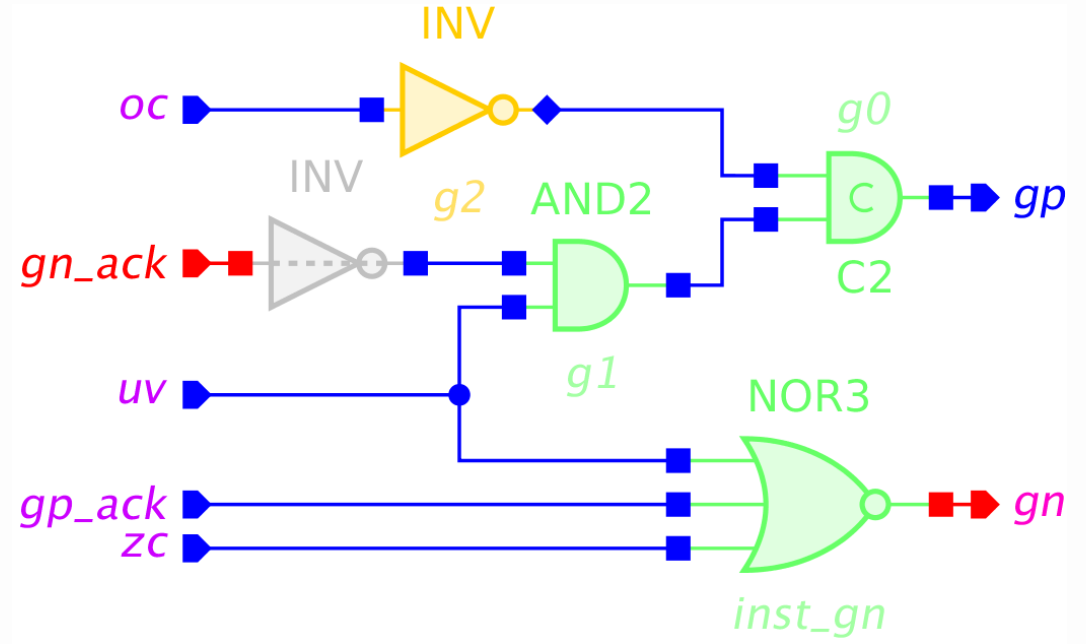
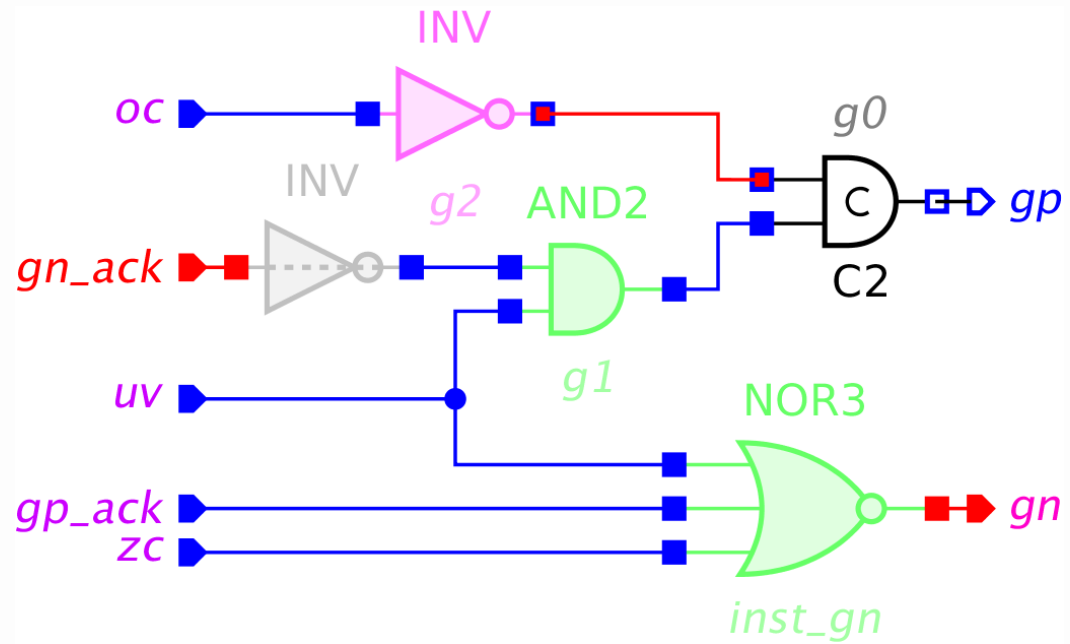
Resetting C-elements via SET/CLEAR pins

- Demo: *buck-monot.circuit.work*



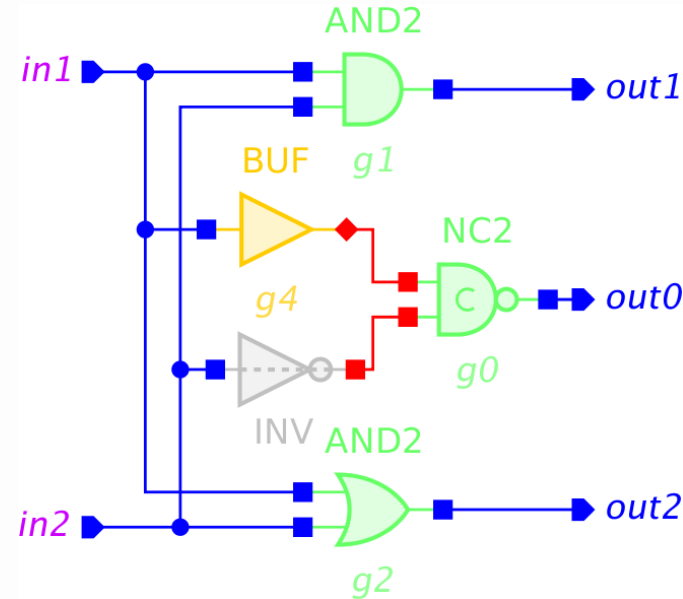
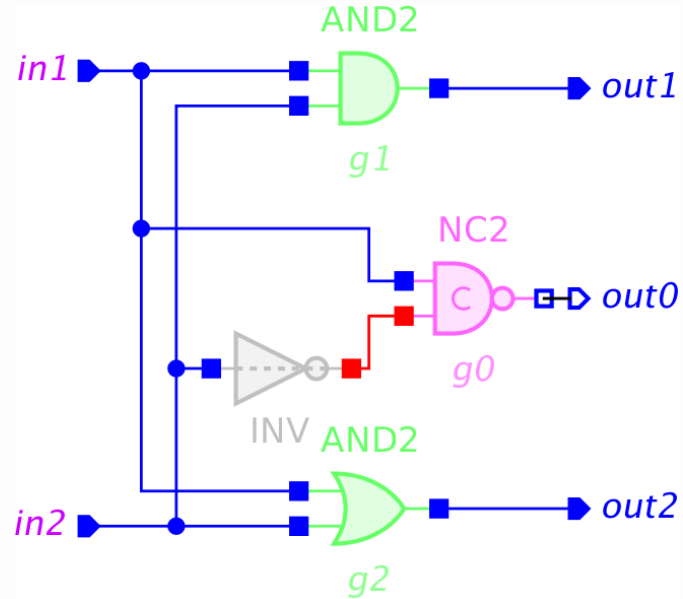
Forcing C-element inputs

- Demo: *buck-monot-inv.circuit.work*



Careful with the forks though!

- Demo: *example-forks.circuit.work*



Verification results

Conformation is violated.

Trace(s) leading to the problematic state(s):

Unexpected change of output 'out0'

g4-, in1+, out2+, in2+, out0+

Play

Close

Verification

- Reset insertion *should not break* the circuit (unless you experiment with the forks)
- Still, always verify the circuit after modification
- Use the original STG as the environment for the modified circuit
- Automatic setup for active-low reset (active-high is symmetrical)
 - **Init to one** property is unset (reset signal is initially low)
 - **Set function** is assigned to 1 (reset signal is allowed to go high)
 - **Reset function** is assigned to 0 (once high reset signal never goes low again)

Practical: Initialisation of speed-independent circuits

- Tutorials section at *workcraft.org*

Modelling causality and concurrency

- Modelling with Finite State Machines: Vending machine
- Petri net synthesis: Concurrent vending machine
- Modelling with Petri nets: Dining philosophers
- Modelling with STGs: Distributed Mutual Exclusion
- Modelling with STGs: Writer-biased read/write lock
- Modelling Genetic Regulatory Networks with STGs: Lysis-Lysogeny switch in Phage λ
- Optimising asynchronous pipelines using wagging



Synthesis and verification of asynchronous circuits

- Design of C-element (basic, detailed instructions)
- Design of basic buck controller (medium, some hints)
- Design of VME bus controller (medium, individual)
- Hierarchical design of a realistic buck controller
- Initialisation of speed-independent circuits
- Loop breaking and offline testing
- Resolution of encoding (CSC) conflicts
- Logic decomposition and technology mapping
- Verification and synthesis of hierarchical designs

All training materials...

- Direct link: <https://workcraft.org/tutorial/synthesis/initialisation/start>