Formal Deadlock Verification of On-Chip Communication Fabrics

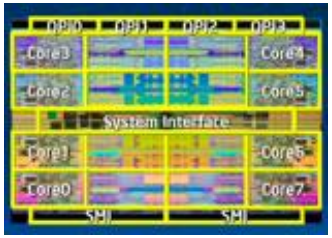
Sebastiaan Joosten (TUE Eindhoven / Radboud Nijmegen)

Freek Verbeek (OU Heerlen)

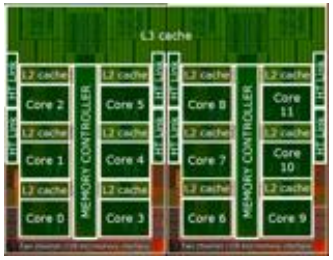
Julien Schmaltz (TUE Eindhoven)

Open Universiteit
www.ou.nl

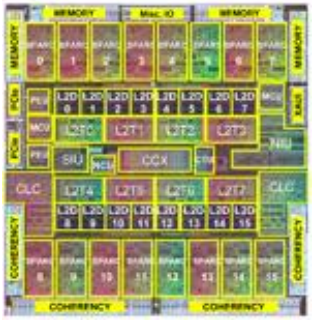
Intel 8 cores
~2.3 Bill. T. on 6.8cm^2



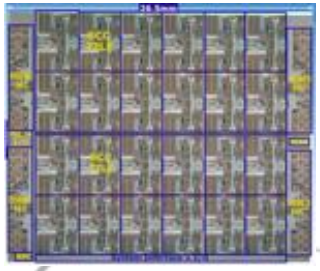
AMD Opteron 12 cores
~1.8 Bill. T. on 2x3.46cm^2



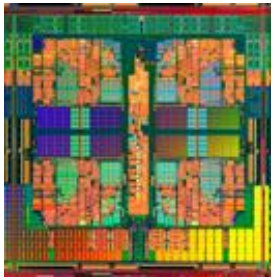
Sun Niagara3 16 cores
~1 Bill. T. on 3.7cm^2



Intel SCC 48 cores
~1.3 Bill. T. on 5.6cm^2

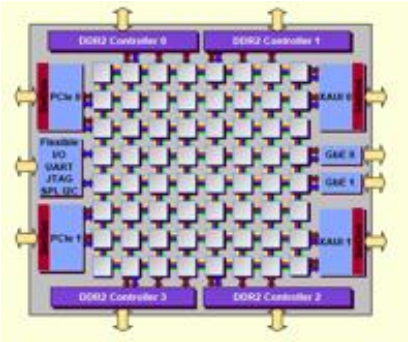


Intel 4 cores
~582 Mio. T. on 2.86cm^2

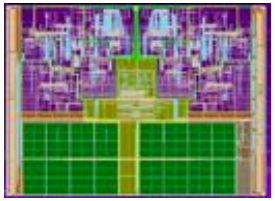


Intel Research 80 cores
~100 Mio. T. on 2.75cm^2

Tilera TILEPro64 64 cores

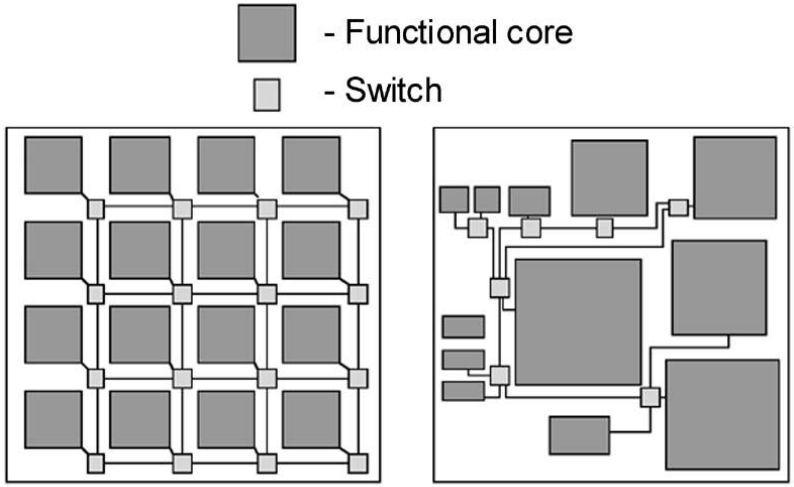


Intel 2 cores
~167 Mio. T. on 1.1cm^2



Open Universiteit
www.ou.nl

Growing complexity of communication fabric



- Heterogeneous topology
- Flow control mechanisms
- Advanced routing function
- Protocol level
  message dependencies
- Virtual channels

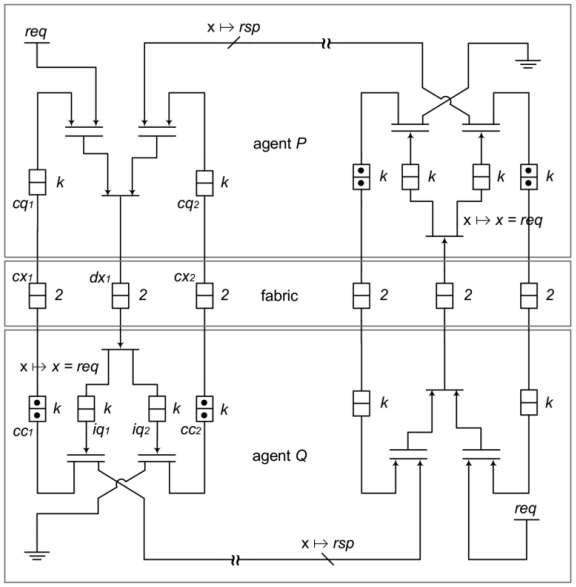
Formal Verification Challenge

We need tools to check for deadlocks

- in large systems
- cross-layer
- parametric
- quick

> Formal Verification should not be a side-salad!
>
> Mike Muller - ARM, Inc. (DAC'12 Keynote)

WickedxMAS: a new way for deadlock verification



Deadlock configuration

No deadlock!

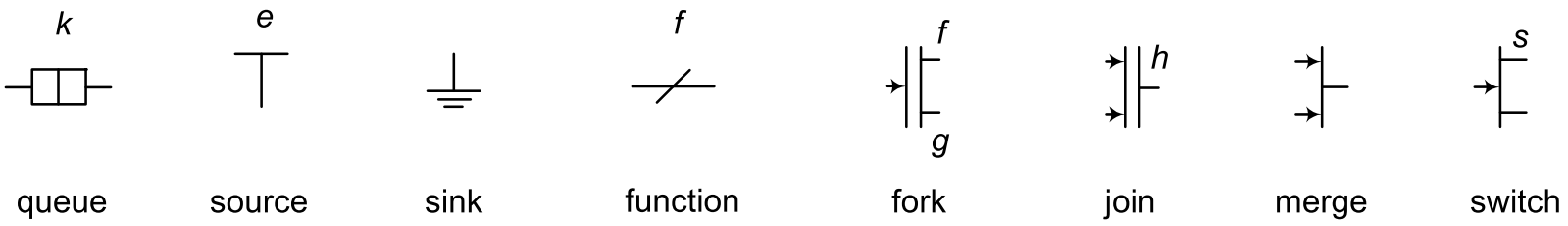
Open Universiteit
www.ou.nl

Outline

- xMAS
- Deadlocks
- Deadlock Detection
  - In the tool
  - Behind the scenes (short)

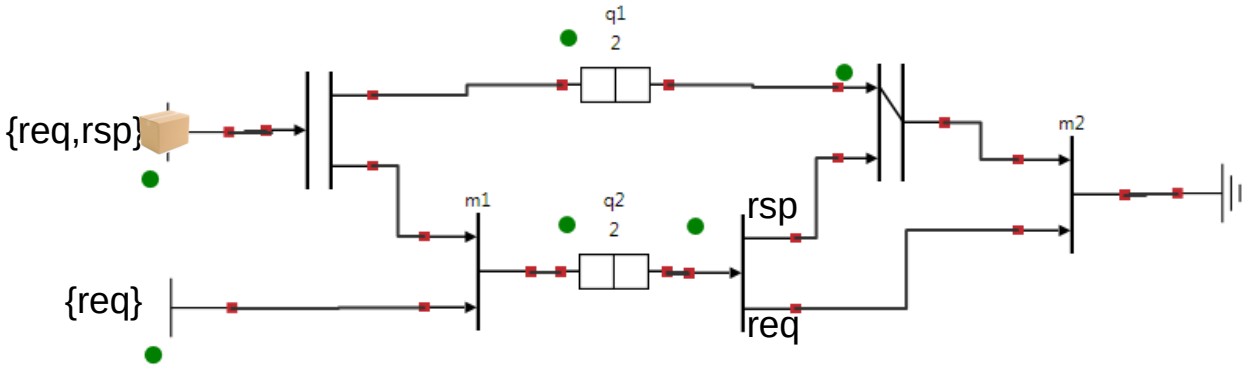
xMAS

xMAS: eXecutable Micro-Architectural Specifications

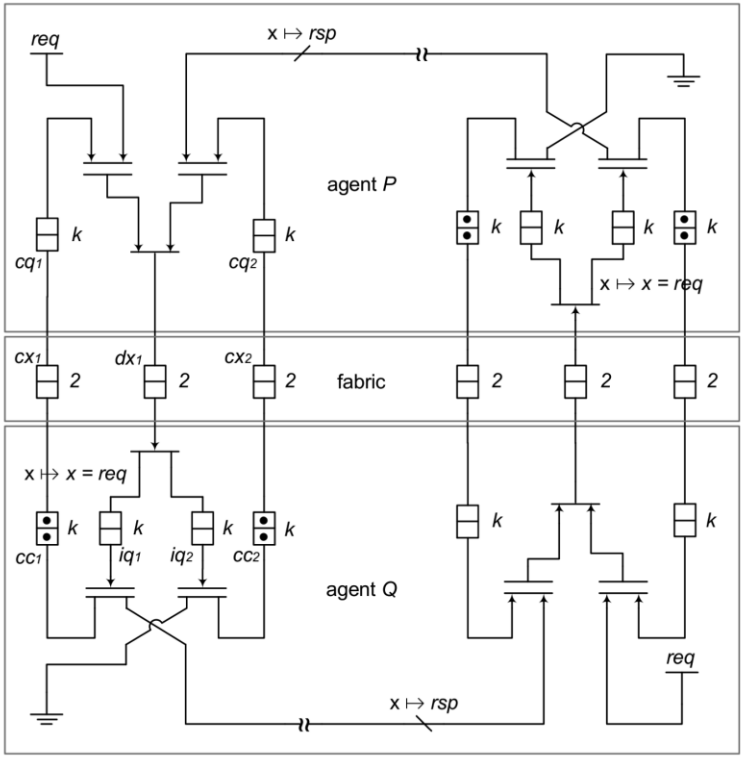


- Fair sinks and sources
- Fair merges (no starvation)
- Restricted joins
- Synchronous execution semantics

xMAS: example

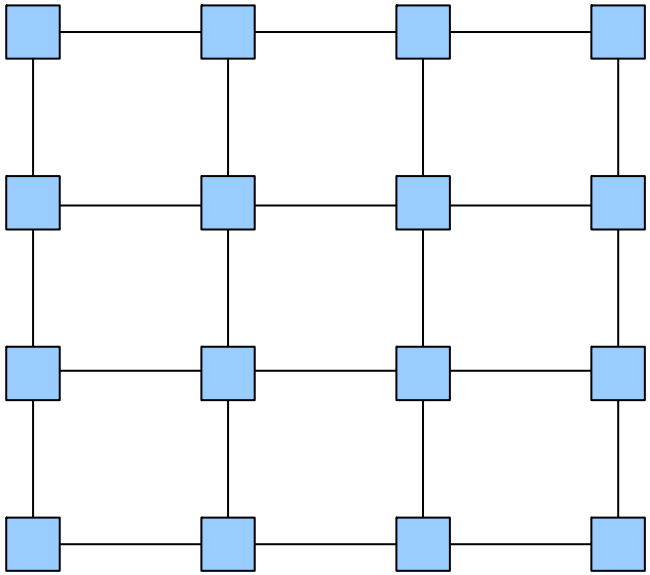


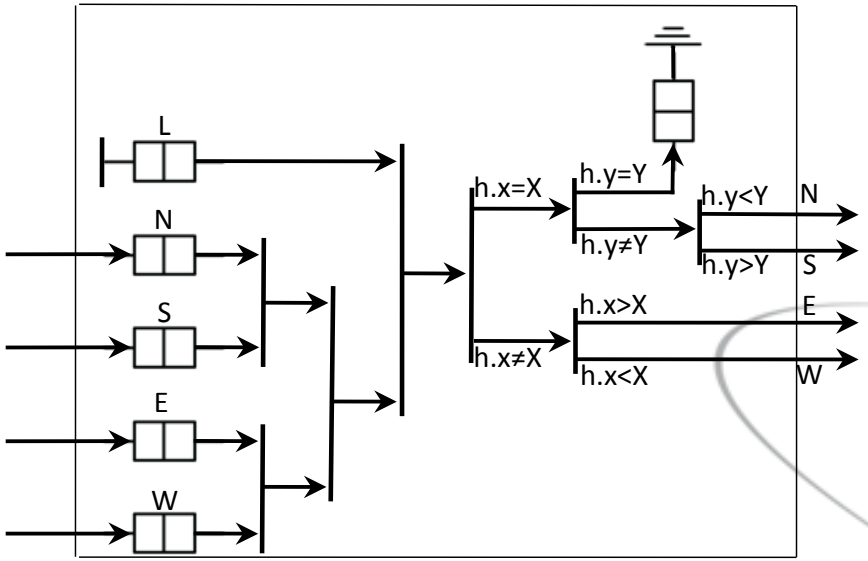
Examples



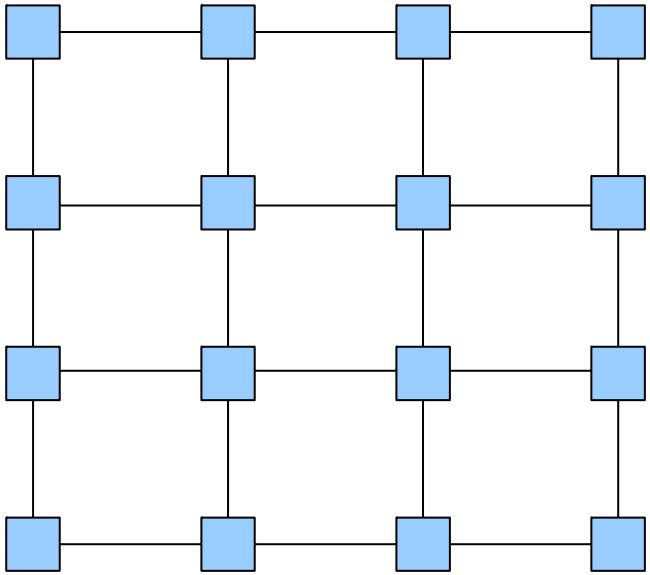
Open Universiteit
www.ou.nl

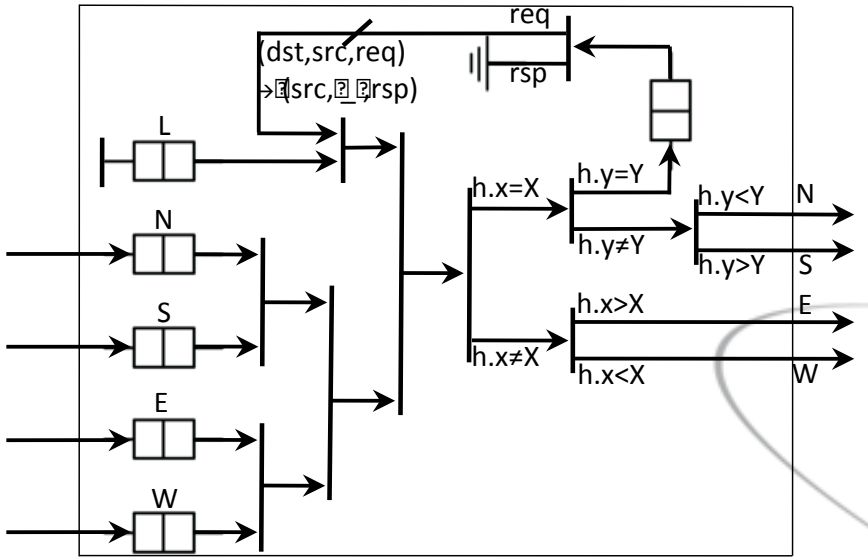
Examples



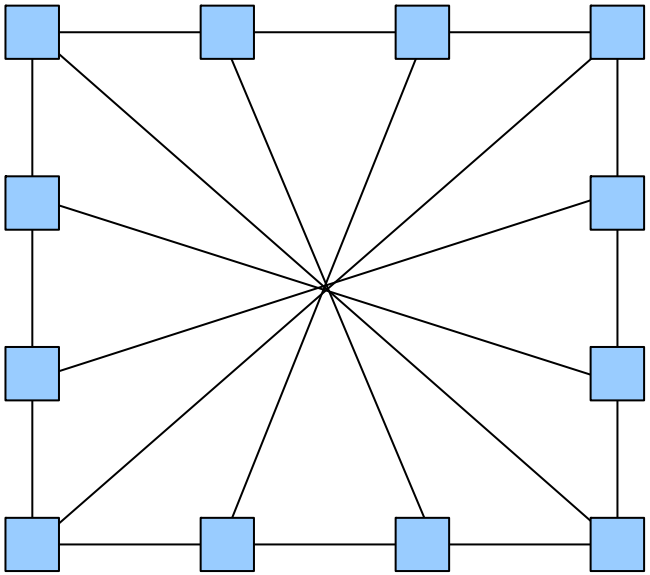


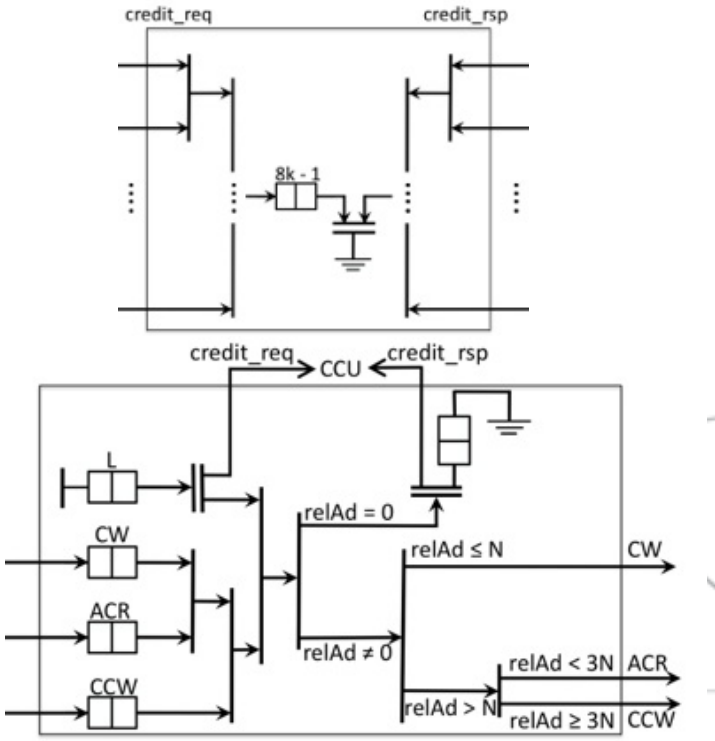
Examples





Examples

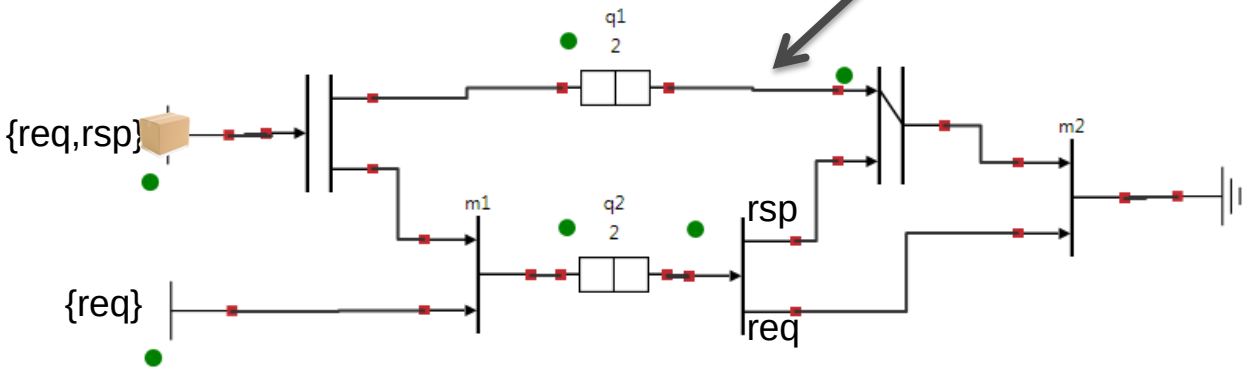




Deadlocks

Deadlocks

$$\Diamond(c.irdy \wedge \Box\neg c.trdy)$$



The challenge:

Find a configuration in which:

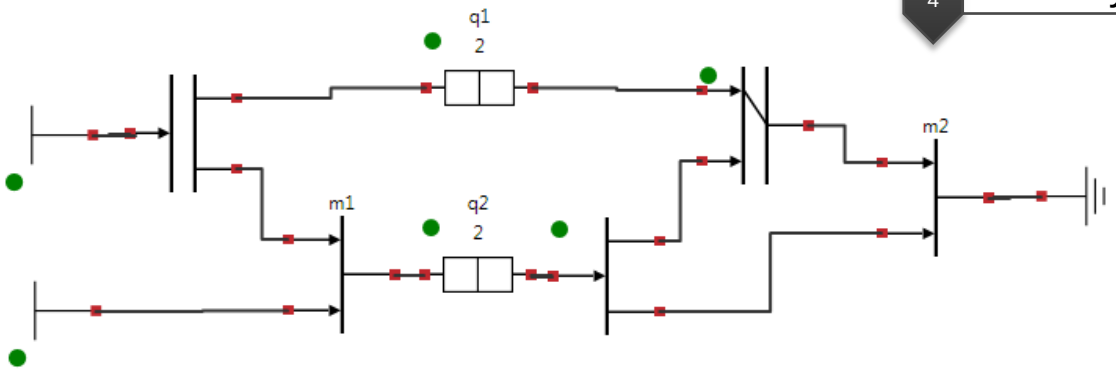
- a packet is permanently stuck
- that is reachable

Deadlock detection

In the tool

WickedxMAS

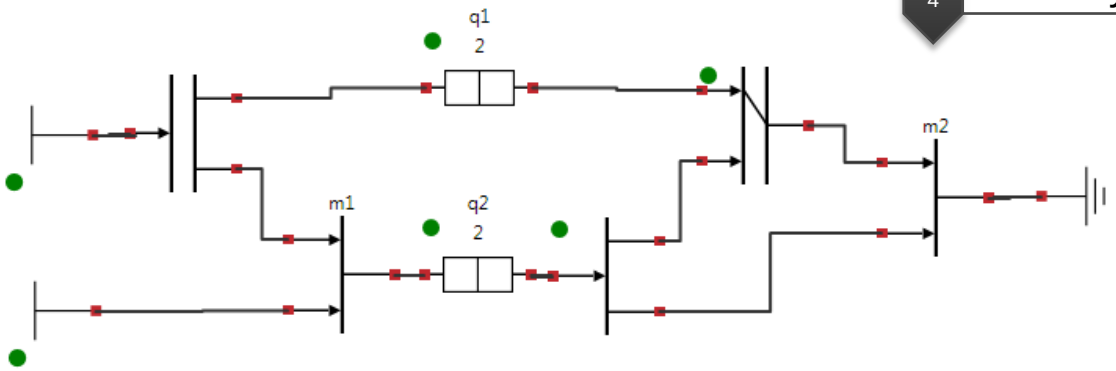
1. Define packet types
2. Add components
3. Add wires
4. Verify design

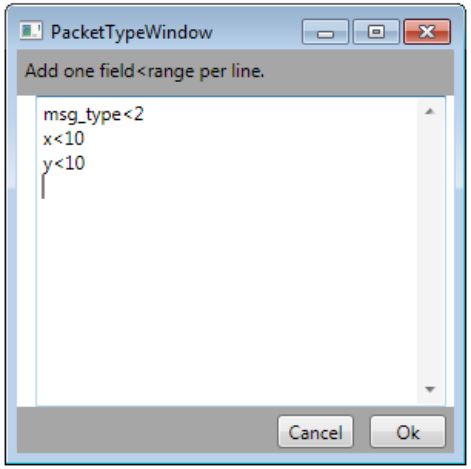


Open Universiteit
www.ou.nl

WickedxMAS

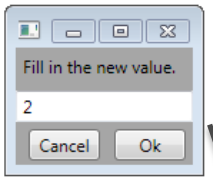
1. Define packet types
2. Add components
3. Add wires
4. Verify design

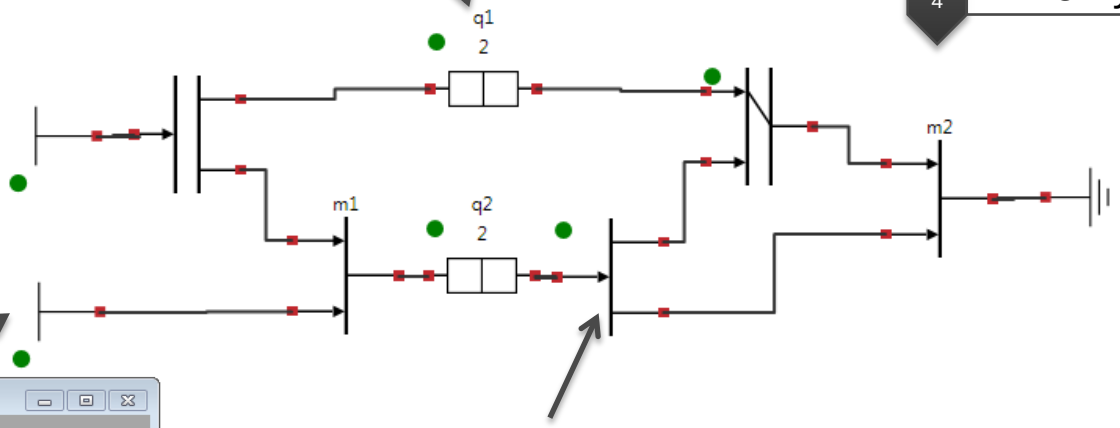


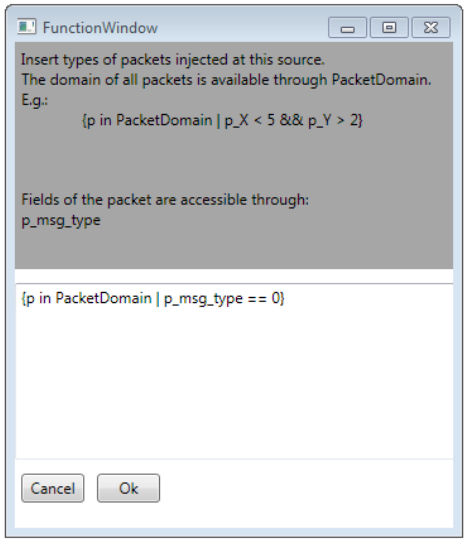


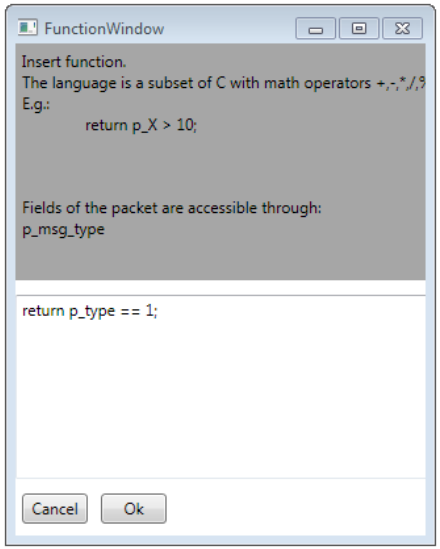
Open Universiteit
www.ou.nl

WickedxMAS







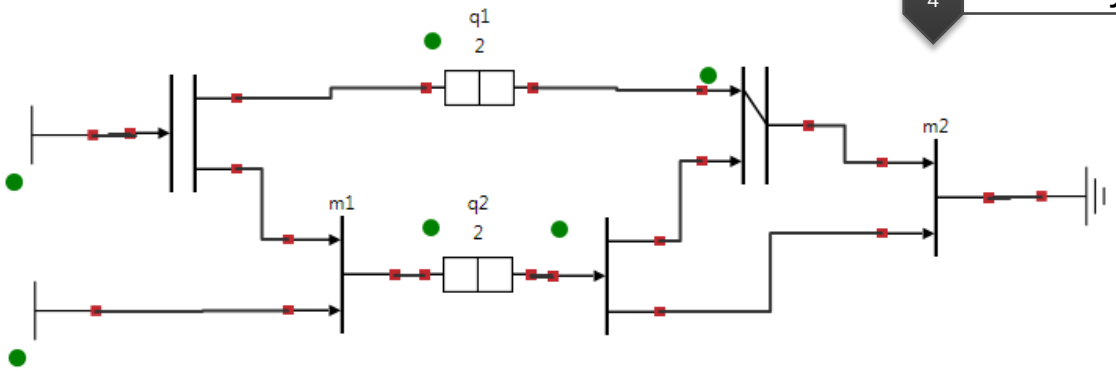


1. Define packet types
2. Add components
3. Add wires
4. Verify design

Open Universiteit
www.ou.nl

WickedxMAS

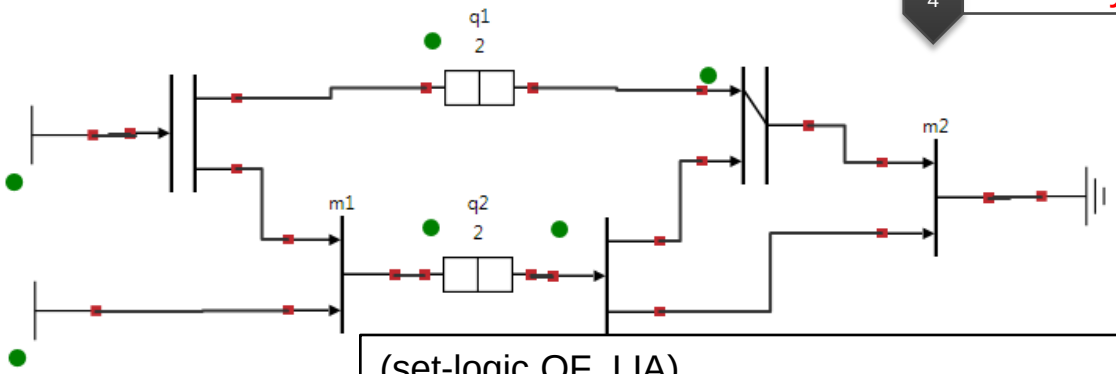
1. Define packet types
2. Add components
3. Add wires
4. Verify design



Open Universiteit
www.ou.nl

WickedxMAS

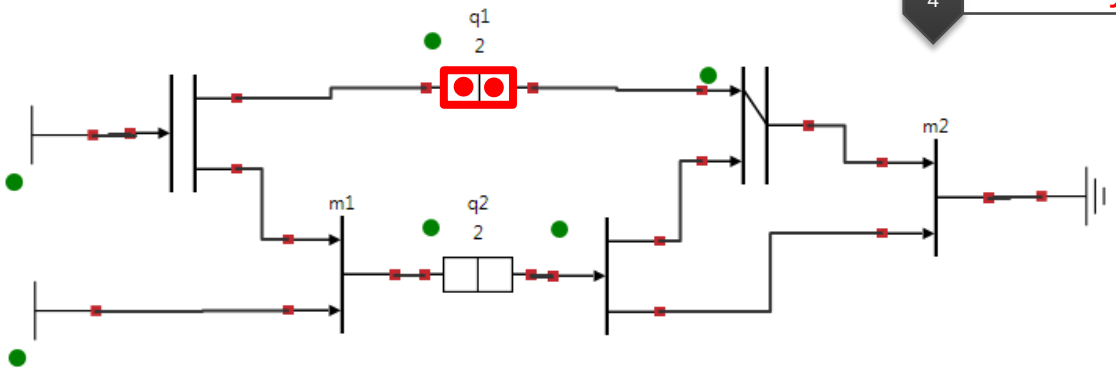
1. Define packet types
2. Add components
3. Add wires
4. Verify design

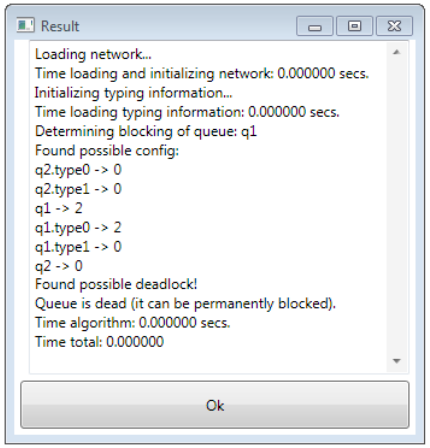



```
(set-logic QF_LIA)

(declare-fun q1 () Int) (assert (<= 0 q1))
(declare-fun q1.type0 () Int) (assert (<= 0 q1.type0))
(declare-fun q1.type1 () Int) (assert (<= 0 q1.type1))
(declare-fun q2 () Int) (assert (<= 0 q2))
(declare-fun q2.type0 () Int) (assert (<= 0 q2.type0))
(declare-fun q2.type1 () Int) (assert (<= 0 q2.type1))

(assert (= 0 (+ (~ q1) q1.type0 q1.type1)))
(assert (= 0 (+ (~ q2) q2.type0 q2.type1)))

(assert (<= 0 (+ q1 (~ q2.type1))))
(assert (<= 0 (+ q1.type1 (~ q2.type1))))
```

WickedxMAS

1. Define packet types
2. Add components
3. Add wires
4. Verify design





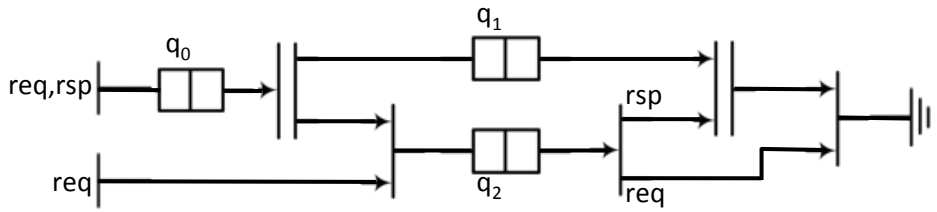
Open Universiteit
www.ou.nl

Deadlock detection

Behind the scenes

Deadlock Detection Algorithm

From network...



... to ILP instance ...

Structural Deadlocks

$\#q_1.req > 0$
$\#q_2.rsp = 0$
$\#q_{1\_} = \#q_1.size$

or

$\#q_2.rsp > 0$
$\#q_{1\_} = 0$
$\#q_{2\_} = \#q_2.size$

and

Invariants

$\#q_1.rsp + \#q_2.rsp$

and

Sanity Constraints

$\#q_{1\_} \leq \#q_1.size$
$\#q_{2\_} \leq \#q_2.size$
$\#q_{1\_} \geq 0$
$\#q_{2\_} \geq 0$

and

Typing Constraints

$\#q_1 = \#q_1.req + \#q_1.rsp$
$\#q_2 = \#q_2.req + \#q_2.rsp$

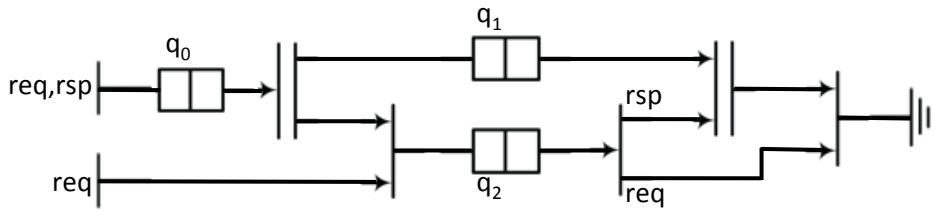
... to solution:

$\#q_1.req = \#q_1.size$
$\#q_2.rsp = 0$

Deadlock Detection Algorithm

Structural Deadlocks | **Invariants** | **Sanity Constraints** | **Typing Constraints**

From network...



... to ILP instance ...

Structural Deadlocks

$\#q_1.req > 0$
$\#q_2.rsp = 0$
$\#q_1 = \#q_1.size$

or

$\#q_2.rsp > 0$
$\#q_1 = 0$
$\#q_2 = \#q_2.size$

and

Invariants

$\#q_1.rsp = \#q_2.rsp$

and

Sanity Constraints

$\#q_1 \geq \#q_1.size$
$\#q_2 \geq \#q_2.size$
$\#q_1 \geq 0$
$\#q_2 \geq 0$

and

Typing Constraints

$\#q_1 = \#q_1.req + \#q_1.rsp$
$\#q_2 = \#q_2.req + \#q_2.rsp$

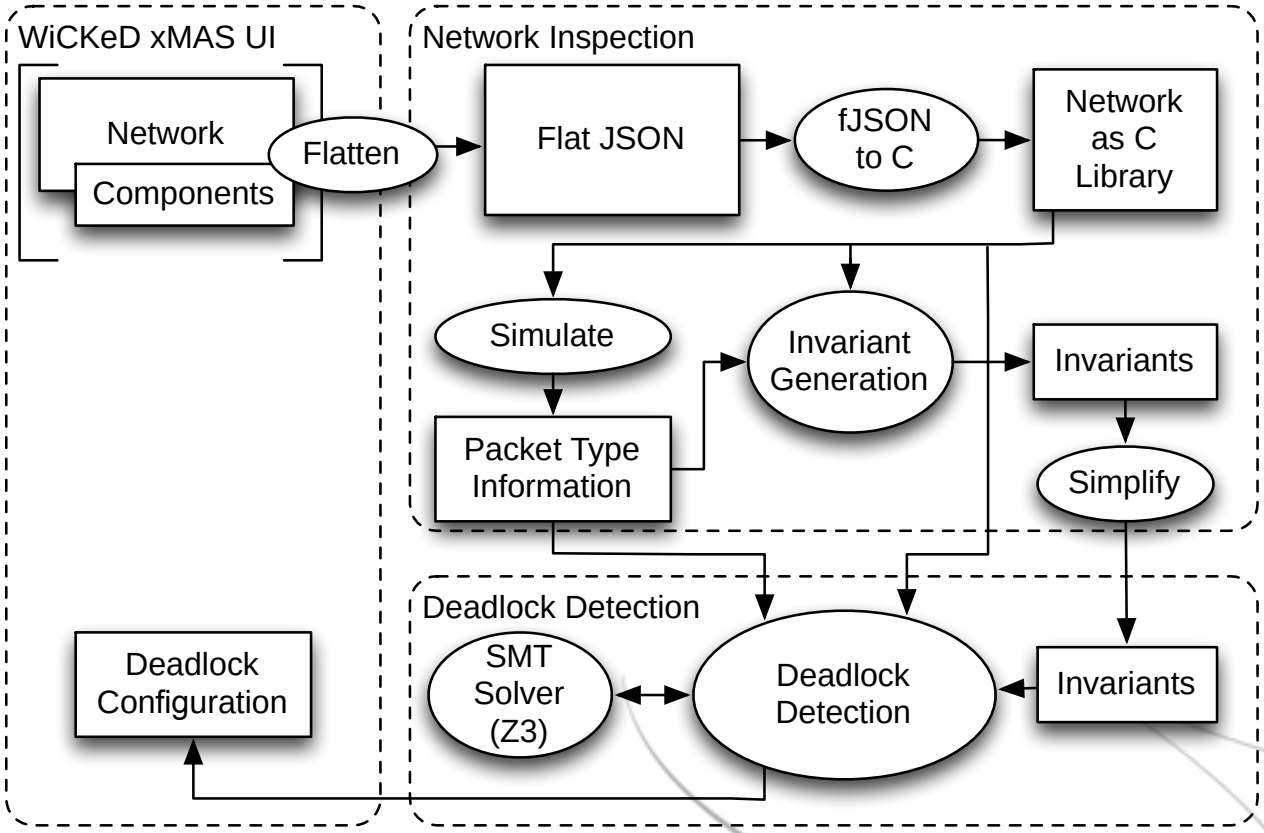
... to solution:

$\#q_1.req = \#q_1.size$
$\#q_2.rsp = 0$

Open Universiteit
www.ou.nl

Deadlock Detection Algorithm

Structural Deadlocks | **Invariants** | **Sanity Constraints** | **Typing Constraints**



Conclusion

- WickedxMAS: design & verification tool
  - Automatic deadlock detection
  - xMAS: language on high-level of abstraction

Future Work

- Many small extensions & improvements
  - Enumerations in packet types
  - Better deadlock visualization
  - Better layout of wires
  - Feedback when something goes wrong
- Dealing with unrestricted joins
- Allowing unfair merges
- Adding custom components
- Hierarchical design & verification: networks-of-networks
- Verified compilers from xMAS to Verilog
- Apply to real examples