

# Formal Deadlock Verification of On-Chip Communication Fabrics

Sebastiaan Joosten (TUE Eindhoven / Radboud Nijmegen)

Freek Verbeek (OU Heerlen)

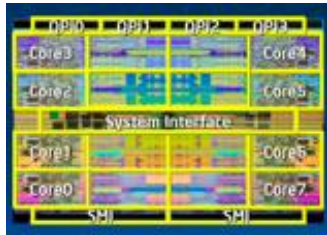
Julien Schmaltz (TUE Eindhoven)

Open Universiteit

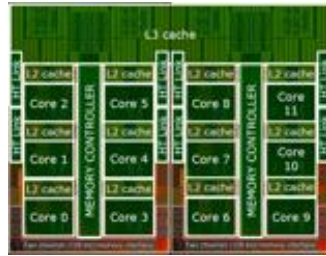
[www.ou.nl](http://www.ou.nl)



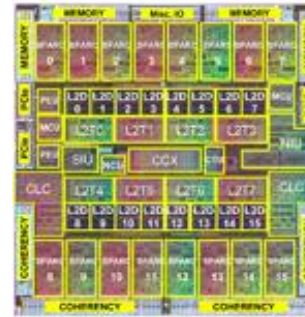
Intel 8 cores  
~2.3 Bill. T. on 6.8cm<sup>2</sup>



AMD Opteron 12 cores  
~1.8 Bill. T. on 2x3.46cm<sup>2</sup>



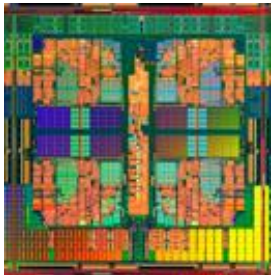
Sun Niagara3 16 cores  
~1 Bill. T. on 3.7cm<sup>2</sup>



Intel SCC 48 cores  
~1.3 Bill. T. on 5.6cm<sup>2</sup>



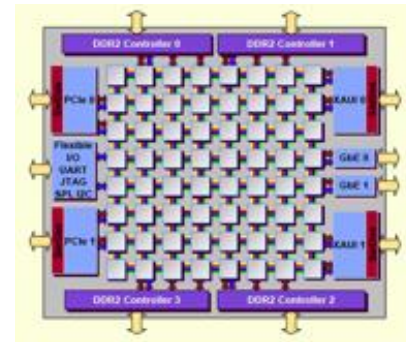
Intel 4 cores  
~582 Mio. T. on 2.86cm<sup>2</sup>



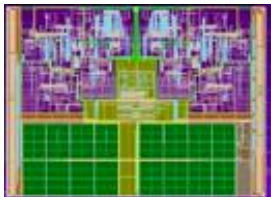
Intel Research 80 cores  
~100 Mio. T. on 2.75cm<sup>2</sup>



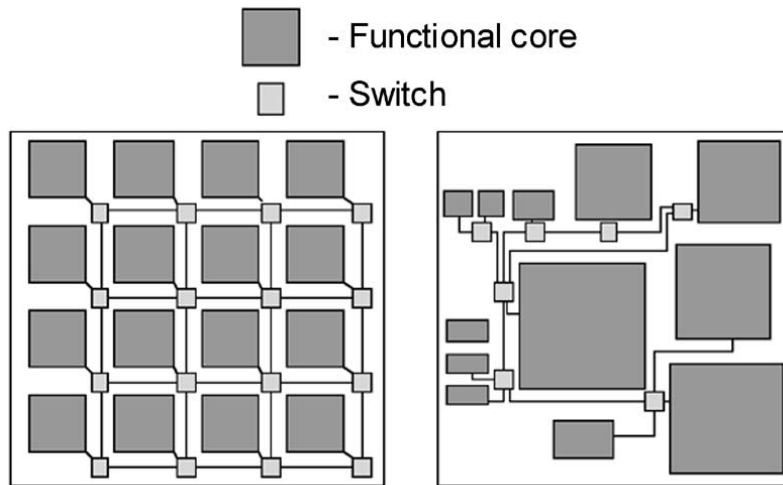
Tilera TILEPro64 64 cores



Intel 2 cores  
~167 Mio. T. on 1.1cm<sup>2</sup>



## 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

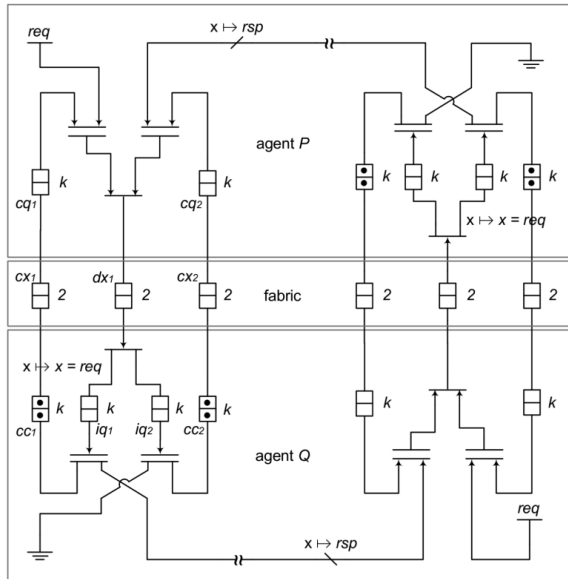
- in large systems
- cross-layer
- parametric
- quick

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!



## Outline

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



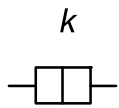
**xMAS**

**Open Universiteit**

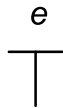
[www.ou.nl](http://www.ou.nl)



## xMAS: eXecutable Micro-Architectural Specifications



queue



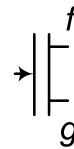
source



sink



function



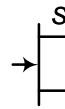
fork



join



merge



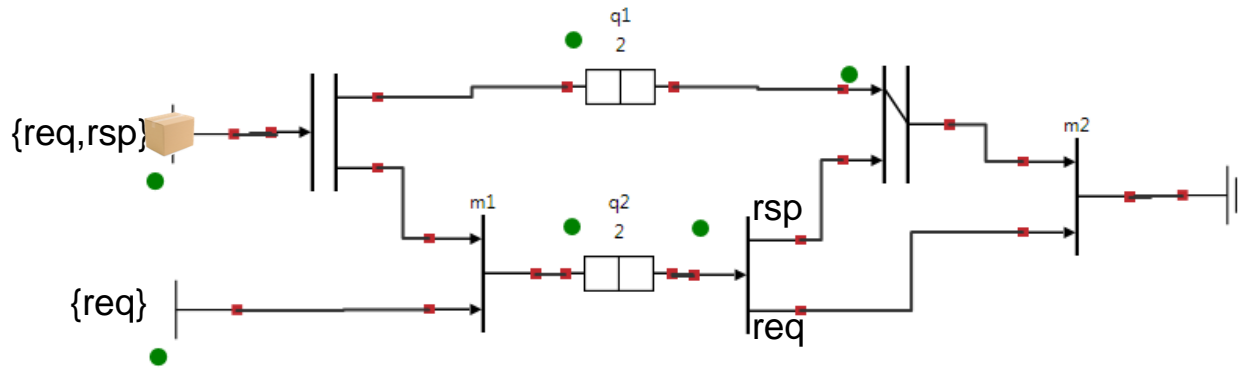
switch

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

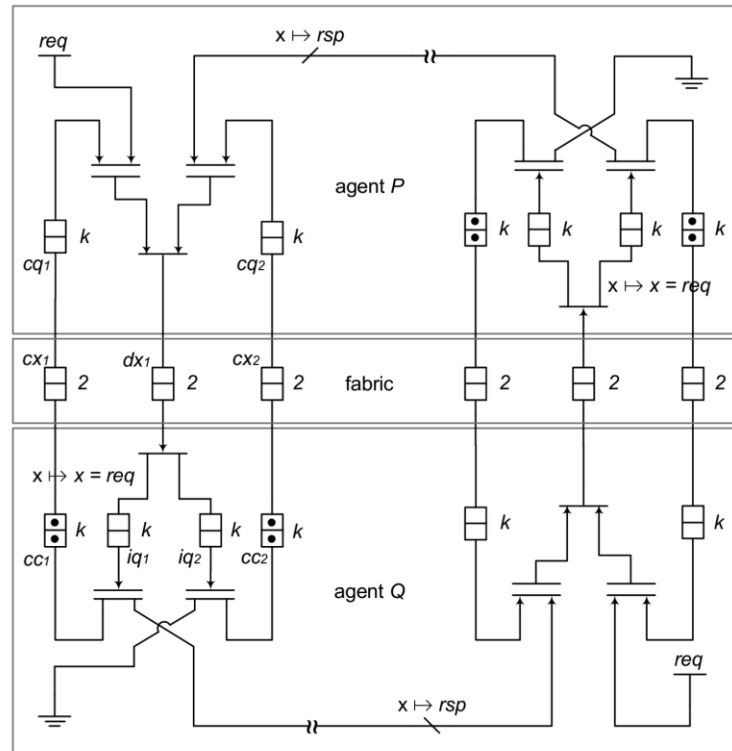




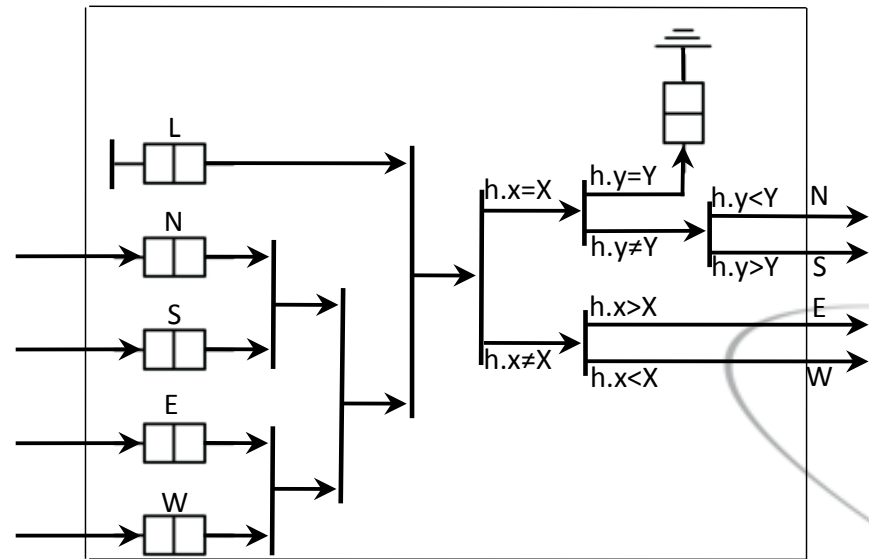
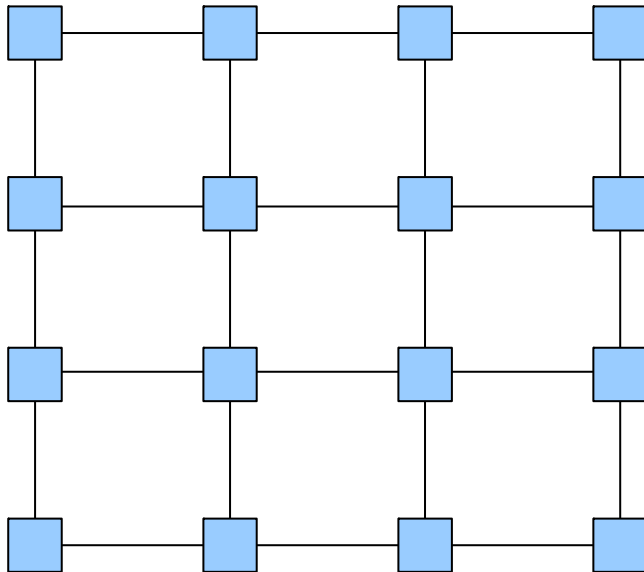
## xMAS: example



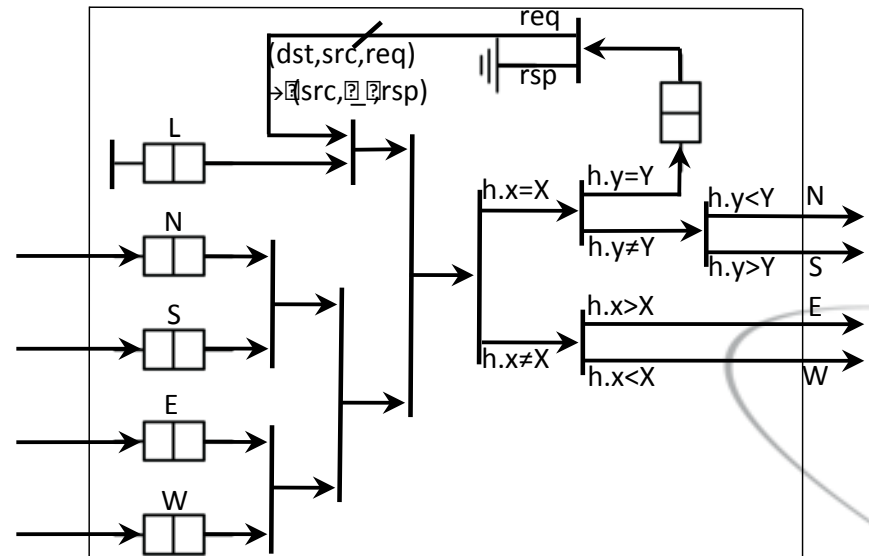
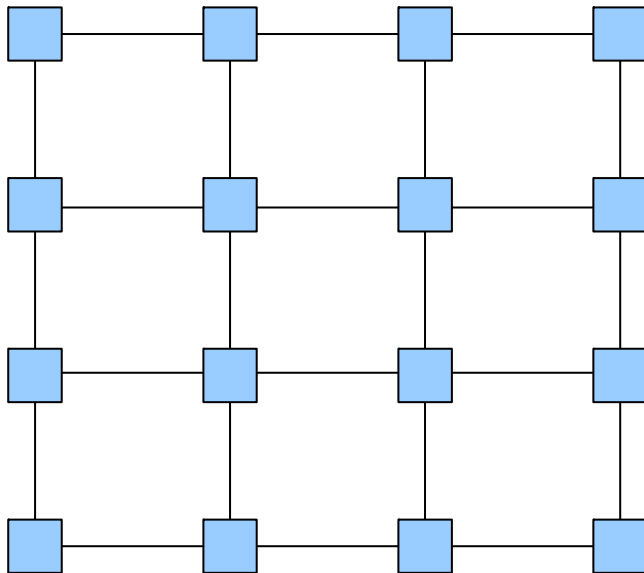
# Examples



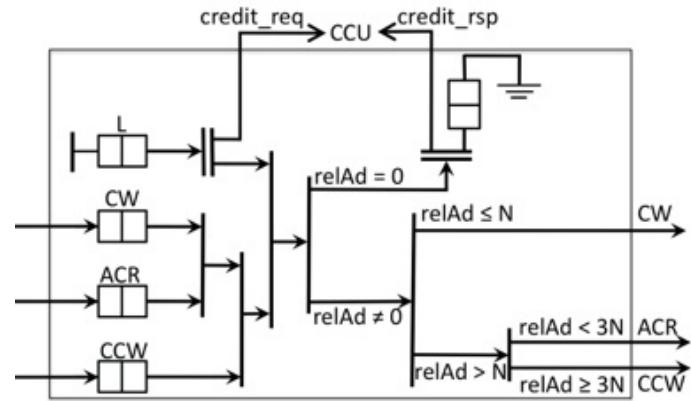
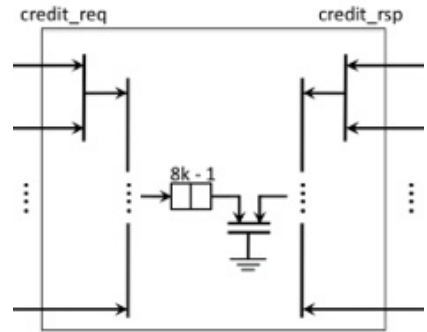
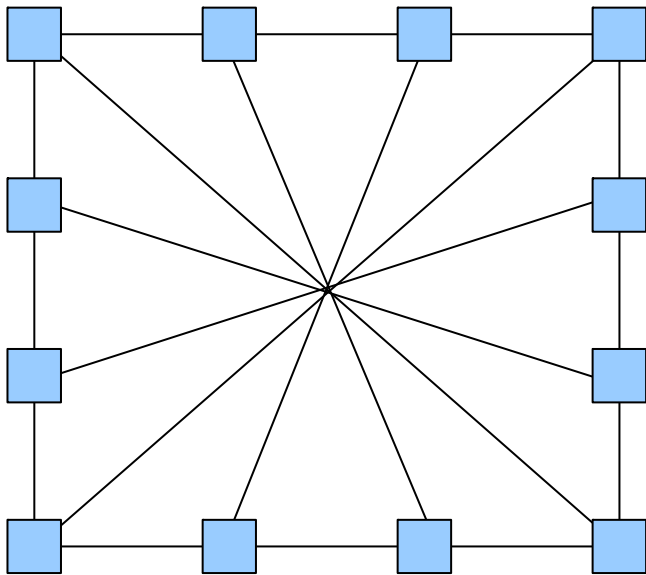
## Examples



## Examples



# Examples



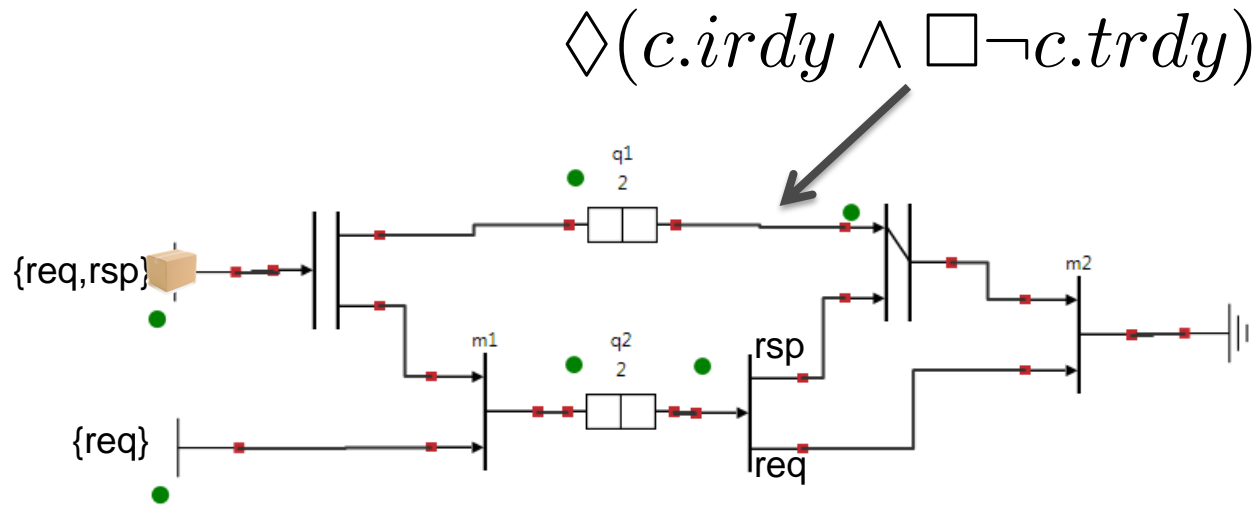
# Deadlocks

Open Universiteit

[www.ou.nl](http://www.ou.nl)



## Deadlocks



### The challenge:

Find a configuration in which:

- a packet is permanently stuck
- that is reachable

# Deadlock detection

In the tool

Open Universiteit

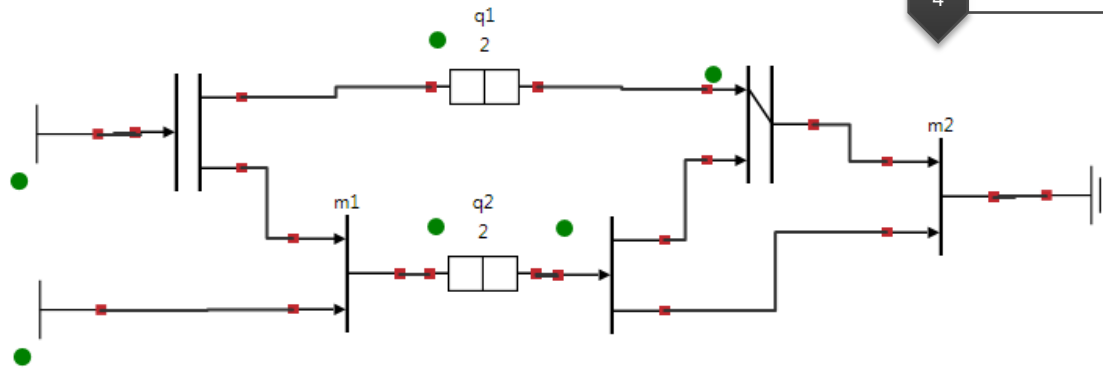
[www.ou.nl](http://www.ou.nl)





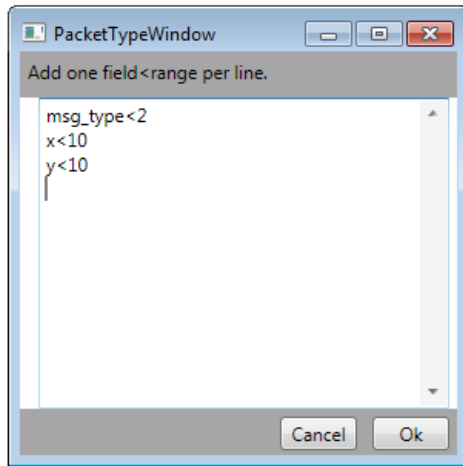
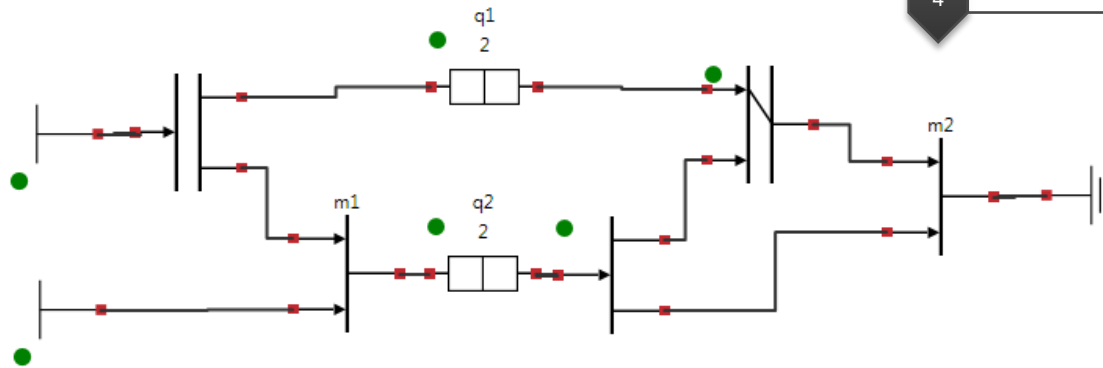
# WickedxMAS

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



# WickedxMAS

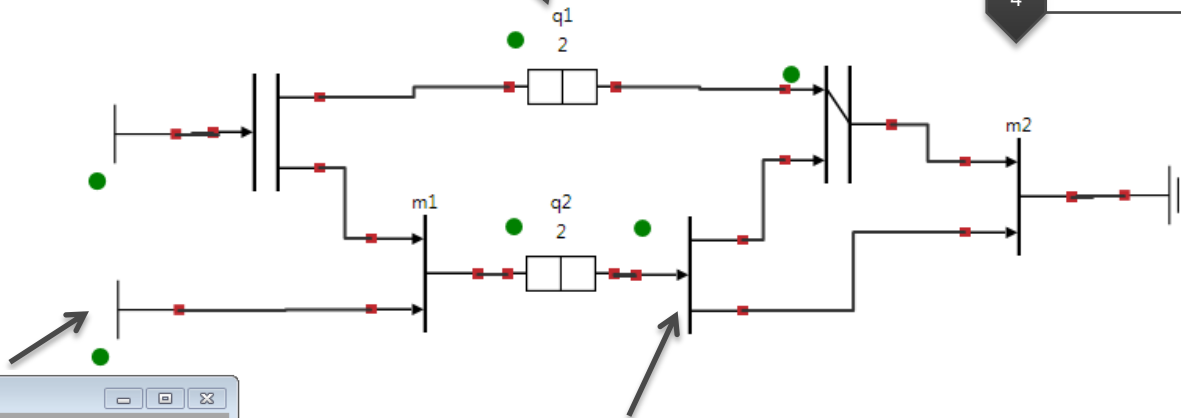
- 1 • Define packet types
- 2 • Add components
- 3 • Add wires
- 4 • Verify design



# WickedxMAS

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

Fill in the new value.  
2  
Cancel Ok



FunctionWindow

Insert types of packets injected at this source.  
The domain of all packets is available through PacketDomain.  
E.g.:  
`{p in PacketDomain | p_X < 5 && p_Y > 2}`

Fields of the packet are accessible through:  
p\_msg\_type

`{p in PacketDomain | p_msg_type == 0}`

Cancel Ok

FunctionWindow

Insert function.  
The language is a subset of C with math operators +, -, \*, /, ?  
E.g.:  
`return p_X > 10;`

Fields of the packet are accessible through:  
p\_msg\_type

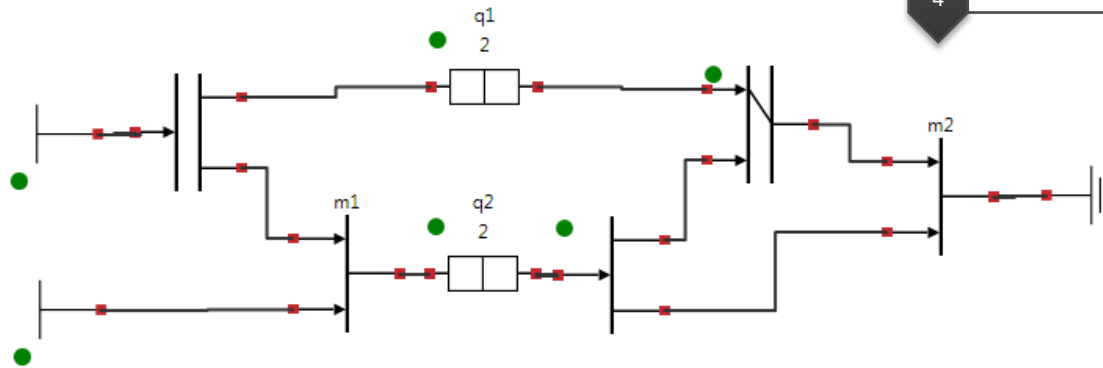
`return p_type == 1;`

Cancel Ok



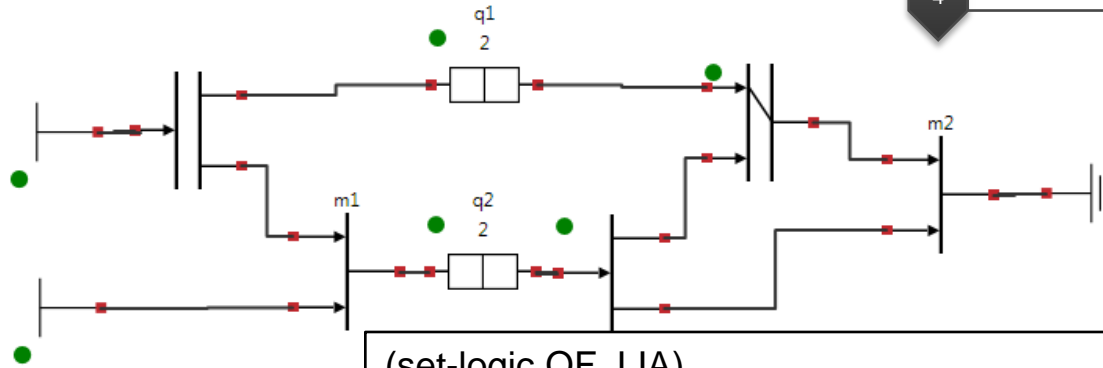
# WickedxMAS

- 1 • Define packet types
- 2 • Add components
- 3 • **Add wires**
- 4 • Verify design



# WickedxMAS

- 1 • Define packet types
- 2 • Add components
- 3 • Add wires
- 4 • **Verify design**



```
(set-logic QF_LIA)
```

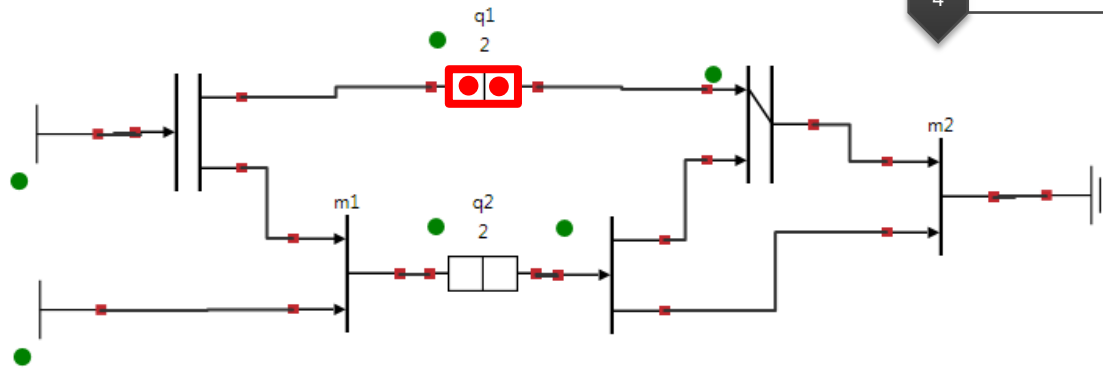
```
(declare-fun q1 () Int) (assert (<= 0 q1))  
(declare-fun q1.type0 () Int) (assert (<= 0 q1.type0))  
(declare-fun q1.type1 () Int) (assert (<= 0 q1.type1))  
(declare-fun q2 () Int) (assert (<= 0 q2))  
(declare-fun q2.type0 () Int) (assert (<= 0 q2.type0))  
(declare-fun q2.type1 () Int) (assert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



```
Result
Loading network..
Time loading and initializing network: 0.000000 secs.
Initializing typing information..
Time loading typing information: 0.000000 secs.
Determining blocking of queue: q1
Found possible config:
q2.type0 -> 0
q2.type1 -> 0
q1 -> 2
q1.type0 -> 2
q1.type1 -> 0
q2 -> 0
Found possible deadlock!
Queue is dead (it can be permanently blocked).
Time algorithm: 0.000000 secs.
Time total: 0.000000
```

Ok



# Deadlock detection

Behind the scenes

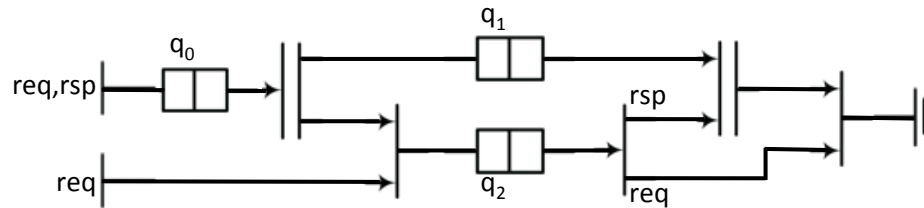
Open Universiteit

[www.ou.nl](http://www.ou.nl)



# Deadlock Detection Algorithm

From network...



... to ILP instance ...

### Structural Deadlocks

#q<sub>1</sub>.req ≥ 1  
 #q<sub>2</sub>.rsp = 1  
 #q<sub>1</sub> = #q<sub>1</sub>.size

or

#q<sub>2</sub>.rsp > 1  
 #q<sub>1</sub> = 1  
 #q<sub>2</sub> = #q<sub>2</sub>.size

### Invariants

and #q<sub>1</sub>.rsp + #q<sub>2</sub>.rsp

### Sanity Constraints

#q<sub>1</sub> ≤ #q<sub>1</sub>.size  
 #q<sub>2</sub> ≤ #q<sub>2</sub>.size  
 #q<sub>1</sub> ≥ 0  
 #q<sub>2</sub> ≥ 0

### Typing Constraints

#q<sub>1</sub> = #q<sub>1</sub>.req + #q<sub>1</sub>.rsp  
 #q<sub>2</sub> = #q<sub>2</sub>.req + #q<sub>2</sub>.rsp

... to solution:

#q<sub>1</sub>.req = #q<sub>1</sub>.size  
 #q<sub>2</sub>.rsp = 1





# Deadlock Detection Algorithm

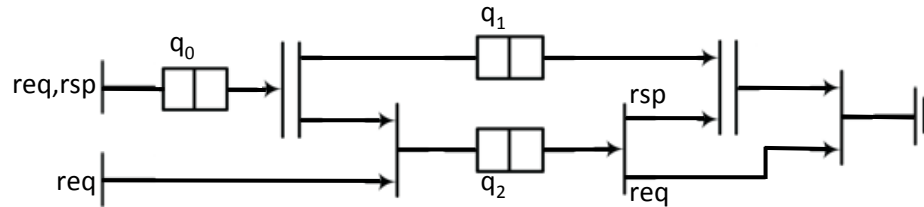
Structural  
Deadlocks

Invariants

Sanity Constraints

Typing  
Constraints

From network...



... to ILP instance ...

Structural  
Deadlocks

#q<sub>1</sub>.req ≥ 0  
#q<sub>2</sub>.rsp = 0  
#q<sub>1</sub> = #q<sub>1</sub>.size

or

#q<sub>2</sub>.rsp > 0  
#q<sub>1</sub> = 0  
#q<sub>2</sub> = #q<sub>2</sub>.size

Invariants

and #q<sub>1</sub>.rsp + #q<sub>2</sub>.rsp

Sanity  
Constraints

#q<sub>1</sub> ≤ #q<sub>1</sub>.size  
#q<sub>2</sub> ≤ #q<sub>2</sub>.size  
#q<sub>1</sub> ≥ 0  
#q<sub>2</sub> ≥ 0

Typing  
Constraints

#q<sub>1</sub> = #q<sub>1</sub>.req + #q<sub>1</sub>.rsp  
#q<sub>2</sub> = #q<sub>2</sub>.req + #q<sub>2</sub>.rsp

... to solution:

#q<sub>1</sub>.req = #q<sub>1</sub>.size  
#q<sub>2</sub>.rsp = 0



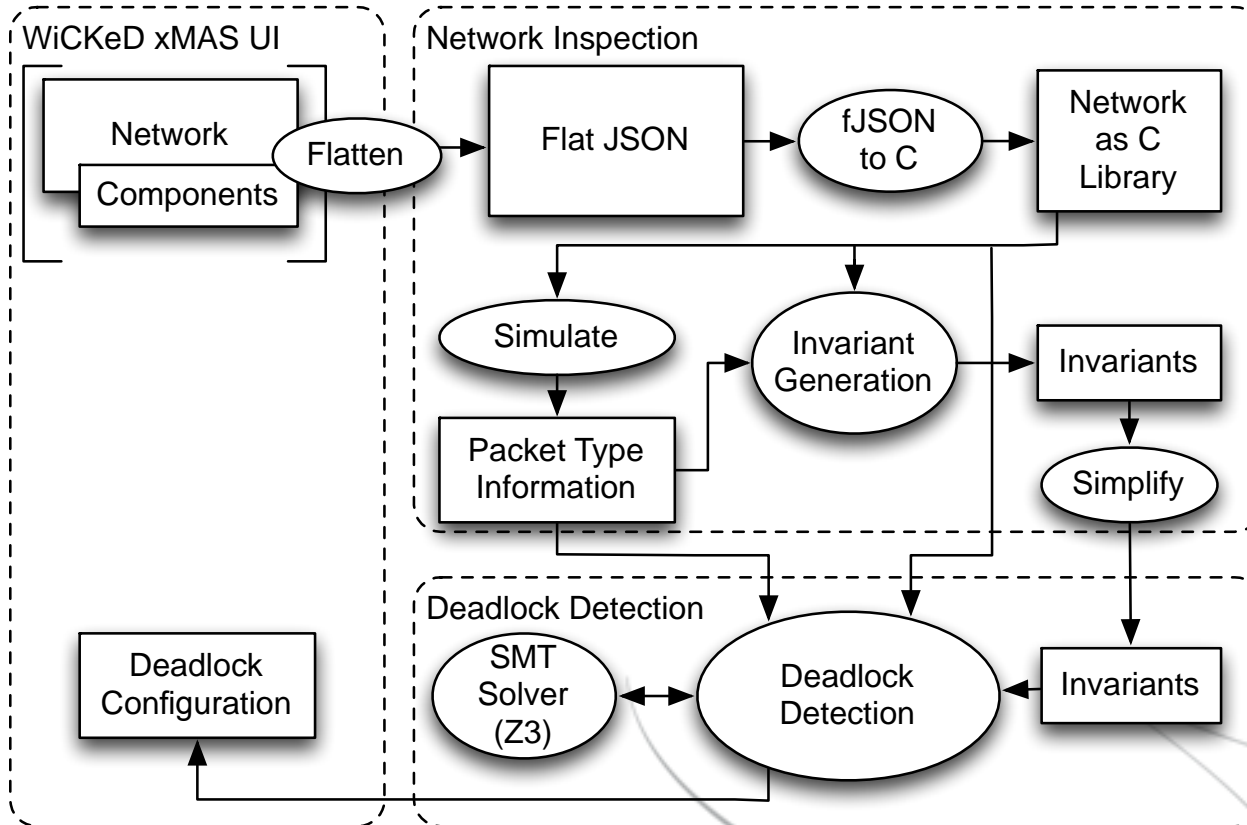
# Deadlock Detection Algorithm

**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

