# Enhanced Symbolic Simulation of a Round-robin Arbiter


Yongjian Li Chinese Academy of Sciences China

Naiju Zeng Chinese Academy of Sciences China

William N. N. Hung Synopsys Inc. USA

Xiaoyu Song Portland State University USA

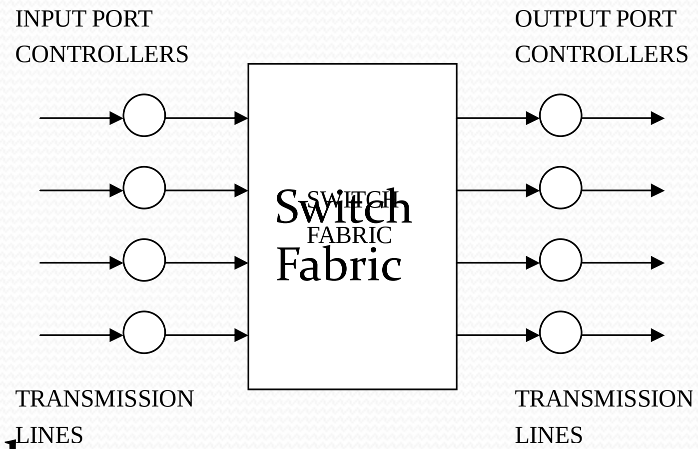
# Motivation

- A fast arbiter is one of the most dominant factors for high performance network switches
  - E.g. ATM (Asynchronous Transfer Mode) network
- Also used in Network-on-Chip
- Hardware implementation need to be verified
- NxN round-robin arbiter: $2^N$ x N cases

# Related Work

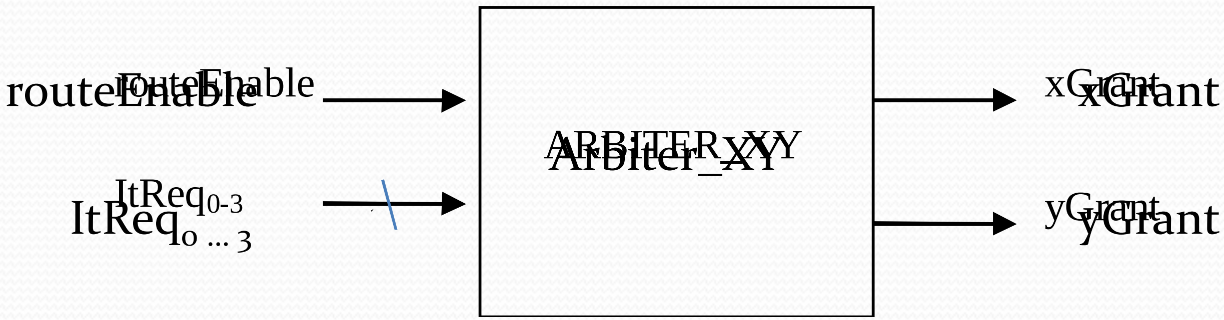
- Curzon's work using HOL theorem prover
  - Pros: exhaustive, interactive
  - Cons: not scalable (limited to 4x4 case), need fundamental modification to extend to 16x16 case
- Chen et al. using SMV
  - Pros: automatic through SMV
  - Cons: not exhaustive due to the way model was reduced
- Tahar et al. using MDG
  - Pros: MDG avoids model explosion
  - Cons: not exhaustive due to the way model was reduced

# NxN switch fabric




- Switches cells
  from N input controllers to N output controllers
- Different inputs destined for the same output?
  - only one will succeed
  - others are rejected, must retry later
- Arbitrates between these cells

# Arbiter




- There are N arbiters: one for each output port
- Each arbiter arbitrates the requests from N input ports
- ItReq: 1-bit per input port
- Output: vector encoding of arbitration result

# Round-robin Arbitration

- Request with highest priority in round-robin order is granted in each cycle
- Fairness (no starvation)
- Worst-case wait time: #requestors - 1

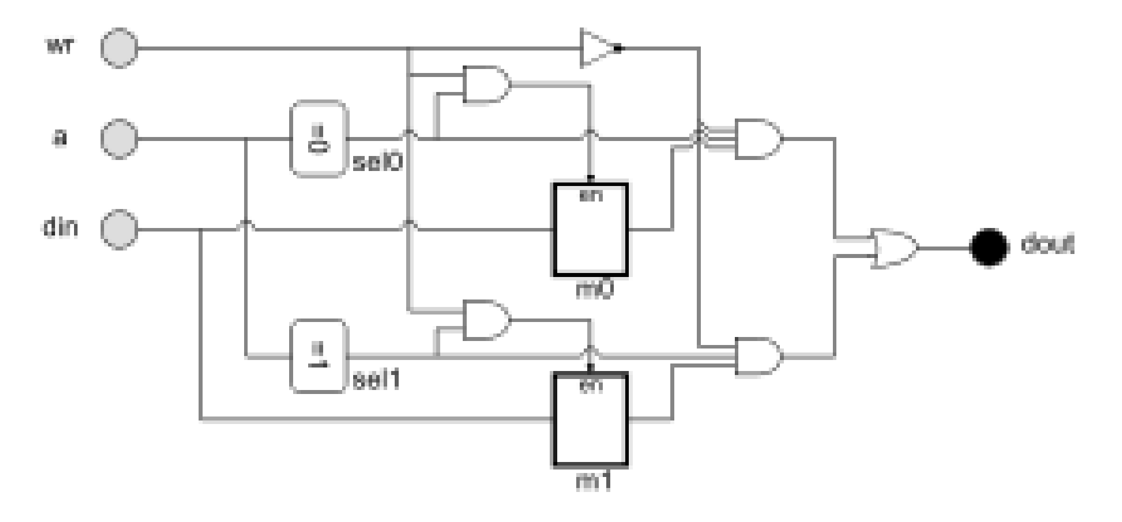
# Symbolic Trajectory Evaluation

- Formal verification technique based on ternary symbolic simulation
- Specification: *ant* ~> *cons*
  *ant*, *cons* are trajectory formula
- Verification: cktSat C *ant* ~> *cons*
- Circuit Model
  - $\mathbf{V} =_{df} \{ff, tt, X, T\}$ dual-rail encoding
  - A circuit state is an instantaneous snapshot of a circuit behavior given by an assignment of **V** to nodes of the circuit

# Generalized Symbolic Trajectory Evaluation

- STE can only deal with bounded time
- Generalized Symbolic Trajectory Evaluation (GSTE)
  - extension of STE
  - deal with properties ranging over unbounded time
  - Properties are specified by assertion graphs, which are ∀-automaton

# Example: Memory Cell




*write* = AndList [Is1 *wr*, *din* isB *bD*, *a* isB *bA*]

retain = AndList [*wr* isB *bWr*, a isB ($bWr \rightarrow \neg\, bA$)]

*read* = AndList [Is0 *wr*, *a* isB *bA*]

*outResult* = *dout* isB *bD*

# STE for Memory Cell

*ant* = AndList [*write*, Next *retain*, $\text{Next}^2$ *read*]

*cons* = $\text{Next}^2$ *outResult*

*ant* ~> *cons*

# Verify one round of arbitration

- Given:
  - last value selected
- Return:
  - the next highest value requested
  - with suitable wrap around from the highest possible request to the lowest

# One Round of Arbitration

Table 1: Ternary-valued Truth Table

| req0 | req1 | req2 | req3 | xGrant | yGrant | xGrant' | yGrant' |
|---|---|---|---|---|---|---|---|
| X | ff | ff | ff | ff | ff | ff | ff |
| X | tt | X | X | ff | ff | ff | tt |
| X | ff | tt | X | ff | ff | tt | ff |
| X | ff | ff | tt | ff | ff | tt | tt |
| ff | X | ff | ff | ff | tt | ff | tt |
| X | X | tt | X | ff | tt | tt | ff |
| X | X | ff | tt | ff | tt | tt | tt |
| tt | X | ff | ff | ff | tt | ff | ff |
| ff | ff | X | ff | tt | ff | tt | ff |
| X | X | X | tt | tt | ff | tt | tt |
| tt | X | X | ff | tt | ff | ff | ff |
| ff | tt | X | ff | tt | ff | ff | tt |
| ff | ff | ff | X | tt | tt | tt | tt |
| tt | X | X | X | tt | tt | ff | ff |
| ff | tt | X | X | tt | tt | ff | tt |
| ff | ff | tt | X | tt | tt | tt | ff |

# STE Specification

$\mathtt{constr}_0 = \neg\mathtt{yGrantV}\&\neg\mathtt{xGrantV}\&\neg\mathtt{reqV}_1\&\neg\mathtt{reqV}_2\&\neg\mathtt{reqV}_3,$

$\mathtt{constr}_1 = \neg\mathtt{yGrantV}\&\neg\mathtt{xGrantV}\&\mathtt{reqV}_1,$

$\mathtt{constr}_2 = \neg\mathtt{yGrantV}\&\neg\mathtt{xGrantV}\&\neg\mathtt{reqV}_1\&\mathtt{reqV}_2,$

$\mathtt{constr}_3 = \neg\mathtt{yGrantV}\&\neg\mathtt{xGrantV}\&\neg\mathtt{reqV}_1\&\neg\mathtt{reqV}_2\&\mathtt{reqV}_3,$

$\mathtt{constr}_4 = \mathtt{yGrant}\&\neg\mathtt{xGrantV}\&\neg\mathtt{reqV}_2\&\neg\mathtt{reqV}_3\&\neg\mathtt{reqV}_0,$

$\mathtt{constr}_5 = \mathtt{yGrant}\&\neg\mathtt{xGrantV}\&\mathtt{reqV}_2),$

$\mathtt{constr}_6 = \mathtt{yGrant}\&\neg\mathtt{xGrantV}\&\neg\mathtt{reqV}_2\&\mathtt{reqV}_3,$

$\mathtt{constr}_7 = \mathtt{yGrant}\&\neg\mathtt{xGrantV}\&\neg\mathtt{reqV}_2\&\neg\mathtt{reqV}_3\&\mathtt{reqV}_0,$

$\mathtt{constr}_8 = \neg\mathtt{yGrantV}\&\mathtt{xGrant}\&\neg\mathtt{reqV}_1\&\neg\mathtt{reqV}_3\&\neg\mathtt{reqV}_0,$

$\mathtt{constr}_9 = \neg\mathtt{yGrantV}\&\mathtt{xGrant}\&\mathtt{reqV}_3,$

$\mathtt{constr}_{10} = \neg\mathtt{yGrantV}\&\mathtt{xGrant}\&\neg\mathtt{reqV}_3\&\mathtt{reqV}_0,$

$\mathtt{constr}_{11} = \neg\mathtt{yGrantV}\&\mathtt{xGrant}\&\mathtt{reqV}_1\&\neg\mathtt{reqV}_3\&\neg\mathtt{reqV}_0,$

$\mathtt{constr}_{12} = \mathtt{yGrant}\&\mathtt{xGrant}\&\neg\mathtt{reqV}_1\&\neg\mathtt{reqV}_2\&\neg\mathtt{reqV}_0,$

$\mathtt{constr}_{13} = \mathtt{yGrant}\&\mathtt{xGrant}\&\mathtt{reqV}_0,$

$\mathtt{constr}_{14} = \mathtt{yGrant}\&\mathtt{xGrant}\&\mathtt{reqV}_1\&\neg\mathtt{reqV}_0,$

$\mathtt{constr}_{15} = \mathtt{yGrant}\&\mathtt{xGrant}\&\neg\mathtt{reqV}_1\&\mathtt{reqV}_2\&\neg\mathtt{reqV}_0$

$\mathtt{cons}_0 = \mathtt{When}\ \mathtt{constr}_0\ \mathtt{AndList}[\mathtt{Is0\ xGrant}, \mathtt{Is0\ yGrant}],$

$\mathtt{cons}_1 = \mathtt{When}\ \mathtt{constr}_1\ \mathtt{AndList}[\mathtt{Is0\ xGrant}, \mathtt{Is1\ yGrant}],$

$\mathtt{cons}_2 = \mathtt{When}\ \mathtt{constr}_2\ \mathtt{AndList}[\mathtt{Is1\ xGrant}, \mathtt{Is0\ yGrant}],$

$\mathtt{cons}_3 = \mathtt{When}\ \mathtt{constr}_3\ \mathtt{AndList}[\mathtt{Is1\ xGrant}, \mathtt{Is1\ yGrant}],$

$\mathtt{cons}_4 = \mathtt{When}\ \mathtt{constr}_4\ \mathtt{AndList}[\mathtt{Is0\ xGrant}, \mathtt{Is1\ yGrant}],$

$\mathtt{cons}_5 = \mathtt{When}\ \mathtt{constr}_5\ \mathtt{AndList}[\mathtt{Is1\ xGrant}, \mathtt{Is0\ yGrant}],$

$\mathtt{cons}_6 = \mathtt{When}\ \mathtt{constr}_6\ \mathtt{AndList}[\mathtt{Is1\ xGrant}, \mathtt{Is1\ yGrant}],$

$\mathtt{cons}_7 = \mathtt{When}\ \mathtt{constr}_7\ \mathtt{AndList}[\mathtt{Is0\ xGrant}, \mathtt{Is0\ yGrant}],$

$\mathtt{cons}_8 = \mathtt{When}\ \mathtt{constr}_8\ \mathtt{AndList}[\mathtt{Is1\ xGrant}, \mathtt{Is0\ yGrant}],$

$\mathtt{cons}_9 = \mathtt{When}\ \mathtt{constr}_9\ \mathtt{AndList}[\mathtt{Is1\ xGrant}, \mathtt{Is1\ yGrant}],$

$\mathtt{cons}_{10} = \mathtt{When}\ \mathtt{constr}_{10}\ \mathtt{AndList}[\mathtt{Is0\ xGrant}, \mathtt{Is0\ yGrant}],$

$\mathtt{cons}_{11} = \mathtt{When}\ \mathtt{constr}_{11}\ \mathtt{AndList}[\mathtt{Is0\ xGrant}, \mathtt{Is1\ yGrant}],$

$\mathtt{cons}_{12} = \mathtt{When}\ \mathtt{constr}_{12}\ \mathtt{AndList}[\mathtt{Is1\ xGrant}, \mathtt{Is1\ yGrant}],$

$\mathtt{cons}_{13} = \mathtt{When}\ \mathtt{constr}_{13}\ \mathtt{AndList}[\mathtt{Is0\ xGrant}, \mathtt{Is0\ yGrant}],$

$\mathtt{cons}_{14} = \mathtt{When}\ \mathtt{constr}_{14}\ \mathtt{AndList}[\mathtt{Is0\ xGrant}, \mathtt{Is1\ yGrant}],$

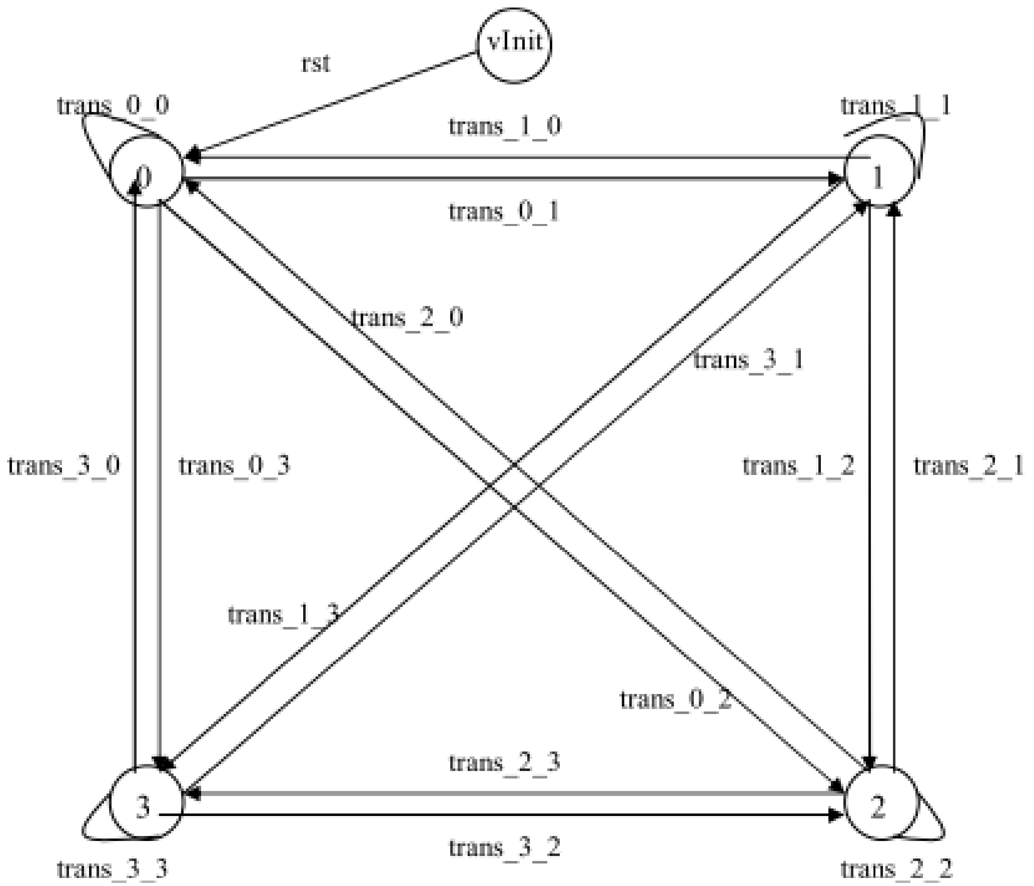
$\mathtt{cons}_{15} = \mathtt{When}\ \mathtt{constr}_{15}\ \mathtt{AndList}[\mathtt{Is1\ xGrant}, \mathtt{Is0\ yGrant}],$

$\mathtt{ant} = \mathtt{AndList}[\mathtt{grantbvAregrantV}, \mathtt{reqbvArereqV}],$

$\mathtt{cons} = \mathtt{Next}\ \mathtt{AndList}[\mathtt{cons}_0, \mathtt{cons}_1, \ldots, \mathtt{cons}_{15}],$

$\mathtt{assert} = \mathtt{ant} \leadsto \mathtt{cons}$

# Sequential Behavior of Arbiter



# GSTE specification

```
let rst = Is1 reset;
let nRst = Is0 reset;
let rtE = Is1 routeEnable;
let trans_0_1 = AndList[nRst, rtE, Is1 req1];
let trans_0_2 = AndList[nRst, rtE, Is0 req1, Is1 req2];
let trans_0_3 = AndList[nRst, rtE, Is0 req1, Is0 req2, Is1 req3];
let trans_0_0 = AndList[nRst, rtE, Is0 req1, Is0 req2, Is0 req3];
let trans_1_2 = AndList[nRst, rtE, Is1 req2];
let trans_1_3 = AndList[nRst, rtE, Is0 req2, Is1 req3];
let trans_1_0 = AndList[nRst, rtE, Is0 req2, Is0 req3, Is1 req0];
let trans_1_1 = AndList[nRst, rtE, Is0 req2, Is0 req3, Is0 req0];
let trans_2_3 = AndList[nRst, rtE, Is1 req3];
let trans_2_0 = AndList[nRst, rtE, Is0 req3, Is1 req0];
let trans_2_1 = AndList[nRst, rtE, Is0 req3, Is0 req0, Is1 req1];
let trans_2_2 = AndList[nRst, rtE, Is0 req3, Is0 req0, Is0 req1];
let trans_3_0 = AndList[nRst, rtE, Is1 req0];
let trans_3_1 = AndList[nRst, rtE, Is0 req0, Is1 req1];
let trans_3_2 = AndList[nRst, rtE, Is0 req0, Is0 req1, Is1 req2];
let trans_3_3 = AndList[nRst, rtE, Is0 req0, Is0 req1, Is0 req2];
```

# Parameterized Verification

# NxN Round-robin Arbiter

```
let otherAntsL = [Is0 "reset", Is1 "routeEnable"];
let transIJ i 0 width N =
   (i, i,
   AndList ((map Is0 (req subtract [req!i]))@otherAntsL),
   (grantOut bvAre (encode i width)))
/\ transIJ i j widthN =
   let j' = (i + j)%N in
   let negReqs = 1 upto (j - 1) in
   let negReqs = map (\k.(req!((k + i)%N))) negReqs in
   let ant = AndList (((map Is0 negReqs)
     union [Is1 (req!j')])@otherAntsL) in
   let newLast = (encode i width ) in
   let cons = (grantOut bvAre newLast) in
   (i, j', ant, cons);
let transFromI WIDTH i =
   let NUM_PORTS = 2 * *WIDTH in
   map (\j.transIJ i j WIDTH NUM_PORTS)
(0 upto (NUM_PORTS - 1));
let transFrom WIDTH =
   let NUM_PORTS = 2 * *WIDTH in
   flat (map transFromI (0 upto (NUM_PORTS - 1)));
let ag WIDTH =
   let initEdge = (0, 1, Is1"reset") in
   initEdge@
   (map (\(from, to, ant, cons).
   (from + 1, to + 1, ant, cons)) transForm);
```

# Experimental Results

Table 2: Experiments

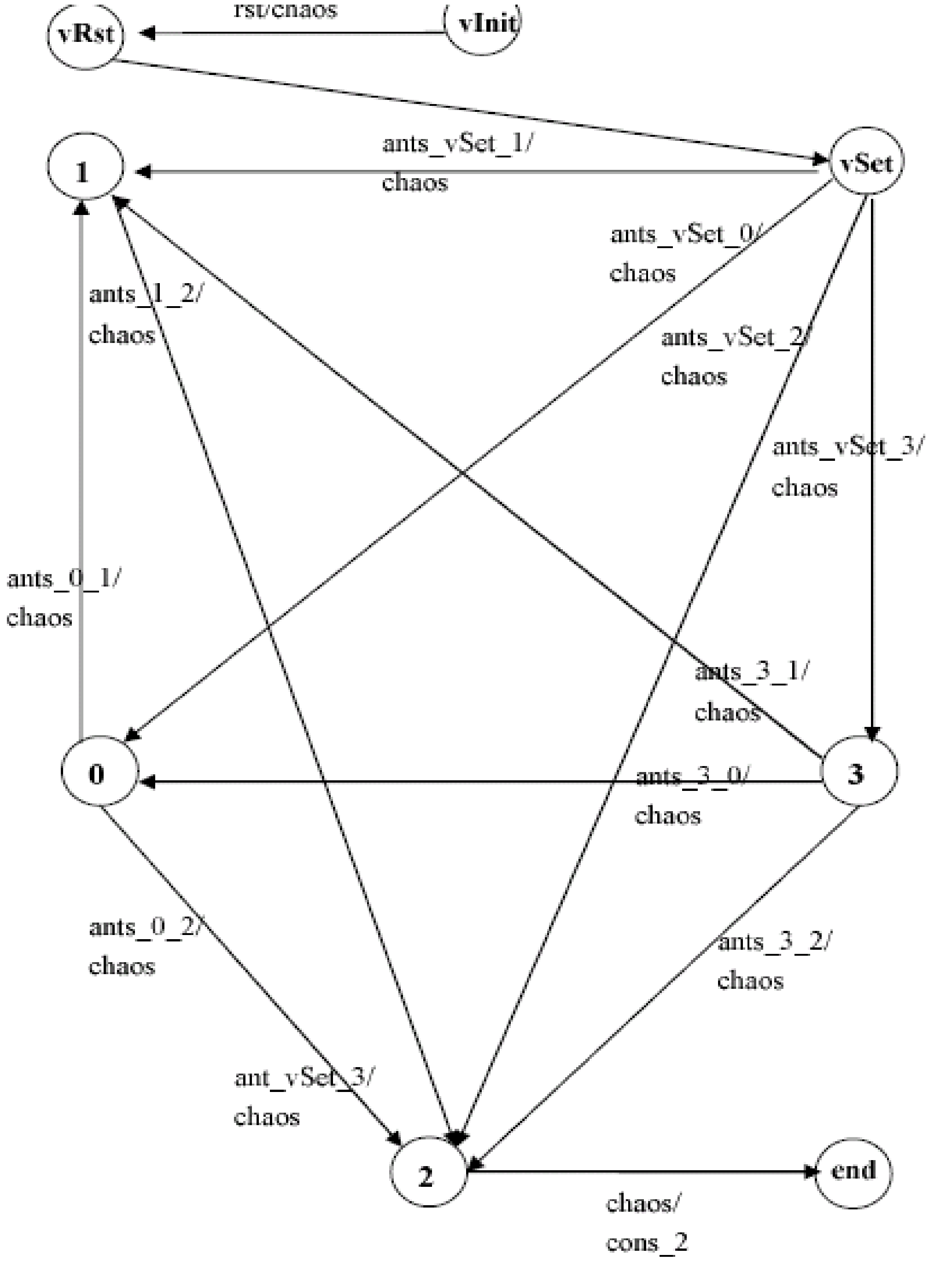
| ARBITER | | | GSTE | | STE | |
|---|---|---|---|---|---|---|
| N | ♯lats | ♯gates | time sec. | mem MB | time sec. | mem MB |
| 8 | 3 | 280 | 0.1 | 12.3 | 0.02 | 8.2 |
| 16 | 4 | 1310 | 1.2 | 19.2 | 0.05 | 10.3 |
| 32 | 5 | 6180 | 34.2 | 50.5 | 0.34 | 20.8 |
| 64 | 6 | 28714 | 1217.9 | 361.3 | 4.43 | 73.0 |

# STE for response property

Response property specifies that once a request $req_i$ *is set high from a state and kept high, then* the request will be granted after several cycles. The worst case waiting time for the request to be granted can also be predicted.

.

*Example : Consider a state where the value of grant is [ff, tt], which means that the last request granted is req2, if the* request *req2 is set high again and kept high, then the request* will be granted (or the value of *grant is set [ff,tt] again) after* at most 4 cycles. Namely, the worst-case waiting time for this request to be granted is 4.



vRst
rst/chaos
vInit
1
vSet
ants_vSet_1/
chaos
ants_vSet_0/
chaos
ants_1_2/
chaos
ants_vSet_2/
chaos
ants_vSet_3/
chaos
ants_0_1/
chaos
ants_3_1/
chaos
0
3
ants_3_0/
chaos
ants_0_2/
chaos
ants_3_2/
chaos
ant_vSet_3/
chaos
2
end
chaos/
cons_2

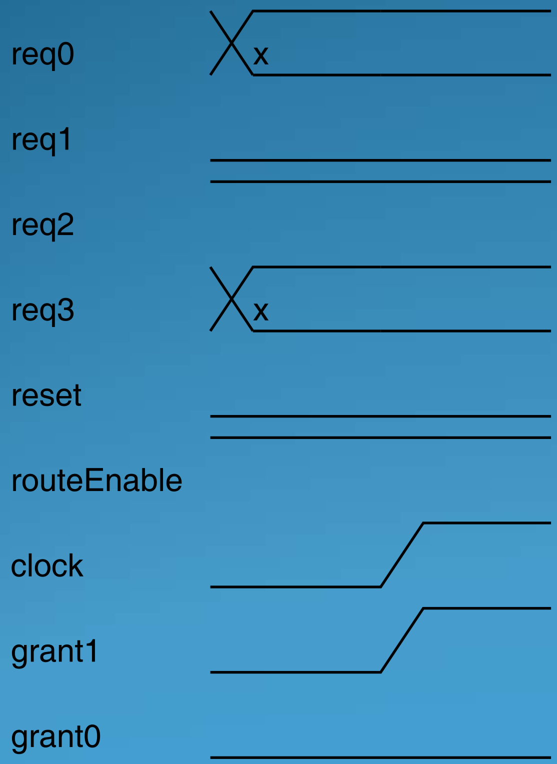
# GSTE Specifications

```
let rst = Is1 reset;
let antSet = AndList [Is0 req0, Is0 req1, Is1 req2, Is0 req3]
let ants_vSet_3 = AndList[Is1 req2, Is1 req3];
let ants_vSet_0 = AndList[Is1 req2, Is0 req3, Is1 req0];
let ants_vSet_1 = AndList[Is1 req2, Is0 req3, Is0 req0,
                          Is1 req1];
let ants_vSet_2 = AndList[Is1 req2, Is0 req3, Is0 req0,
                           Is0 req1];
let ants_3_0 = AndList[Is1 req2, Is1 req0];
let ants_3_1 = AndList[Is1 req2, Is0 req0, Is1 req1];
let ants_3_2 = AndList[Is1 req2, Is0 req0, Is0 req1, ];
let ants_0_1 = AndList[Is1 req2, Is1 req1];
let ants_0_2 = AndList[Is1 req2, Is0 req1];
let ants_1_2 = AndList[Is1 req2];
let cons_2 = AndList[Is0 grant0, Is1 grant1];
```

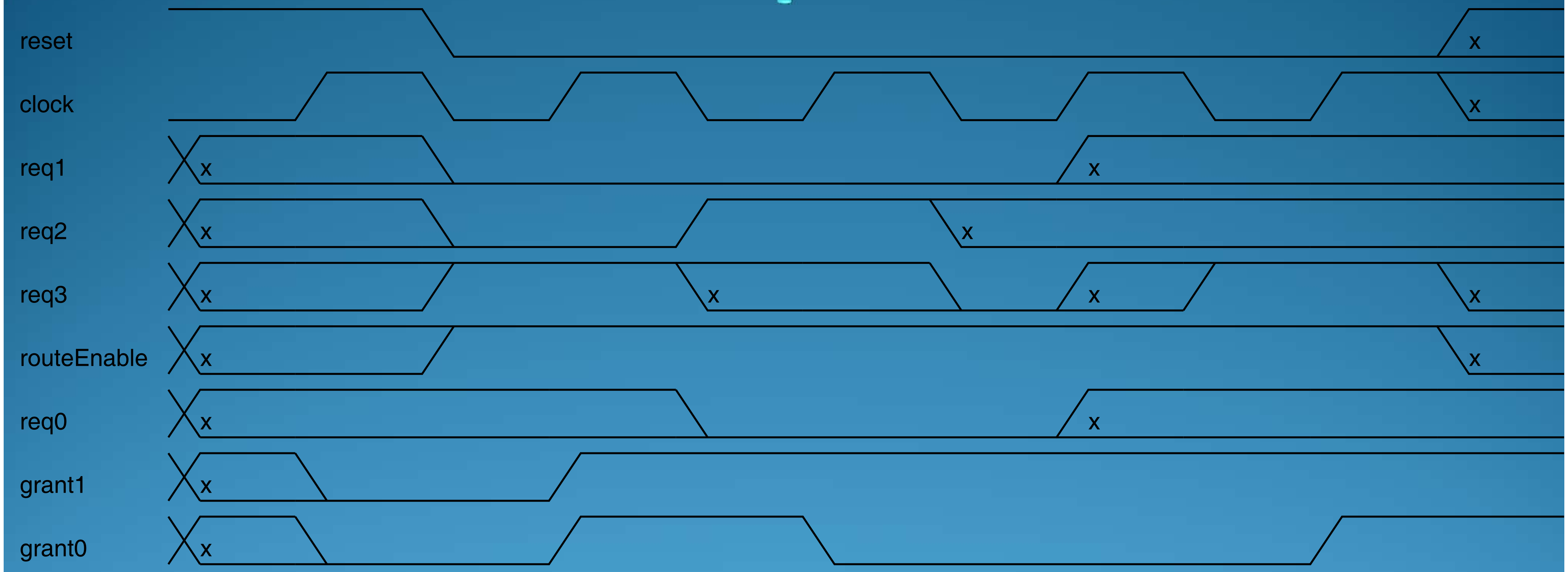
# From (G)STE specs to test patterns

1. Constraint-based
2. Randomly walking around the edges of GSTE graph

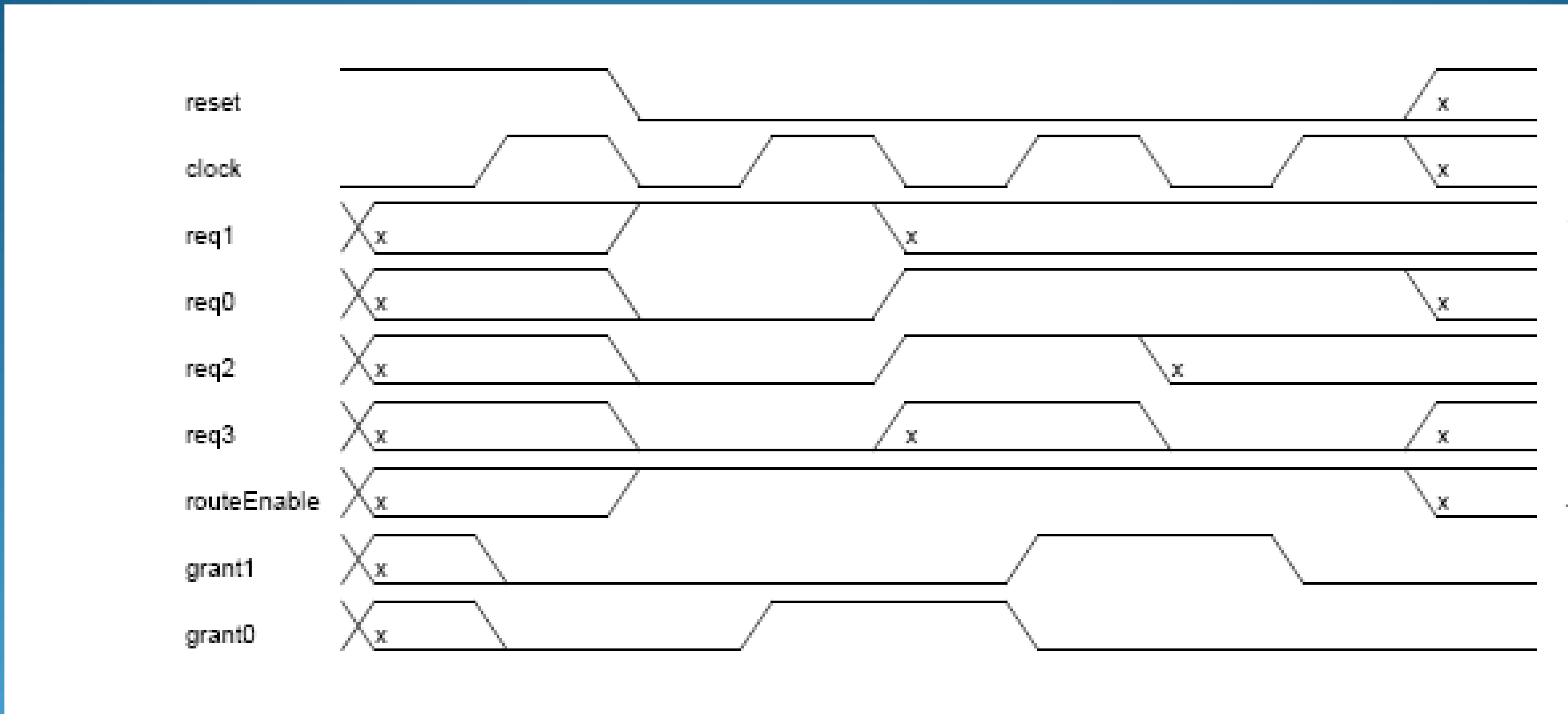
# Generated test patterns



# Generated test patterns



# Generated test patterns



# Any Questions?

# HOL specification

```
let SUC_MODN N last =
    (last + 1 = len) => 0 | (last + 1);

letrec RoundRobin 0 requestSet last N = 0
    /\ RoundRobin n requestSet last N =
            let tryNext = SUC_MOD N last in
            ((tryNext mem requestSet) => tryNext
            | RoundRobin (n – 1) requestSet tryNext);

let RoundRobinArbiter N requestSet last =
    (requestSet = []) => NORESULT |
    (RESULT (RoundRobin N requestSet last));
```

# HOL Specification (part 2)

```
let RequestsToArbitrate 0 reqVect = []
    /\ RequestsToArbitrate n reqVect =
            (requests ! (n-1)) =>
            ([n] union (RequestsToArbitrate (n-1) requests)
            |RequestsToArbitrate (n-1) requests;

let SuccessFullInput last reqVect =
    let requestSet =
    RequestsToArbitrate (length reqVect) reqVect in
    (RoundRobinArbiter (length reqVect) requestSet last);

let GrantForOut reqVect grantVect =
    let sucInp = SuccessFullInput (BNVAL grantVect) reqVect in
    (suc_inp = NORESULT) => grantVect
      | (val (RESULT result = sucInp) in
            VAL2VEC (length grantVect) result));
```