A Functional Specification and Validation Model for Networks on Chip in the ACL2 Logic

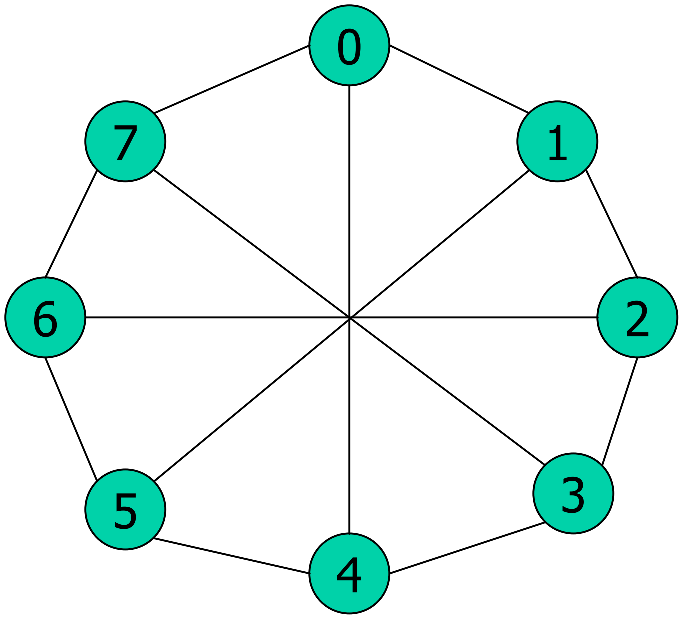
J. Schmaltz* and D. Borrione

TIMA Laboratory - VDS Group

Grenoble, France

*** Part of this work was done while visiting the CS department of UT**

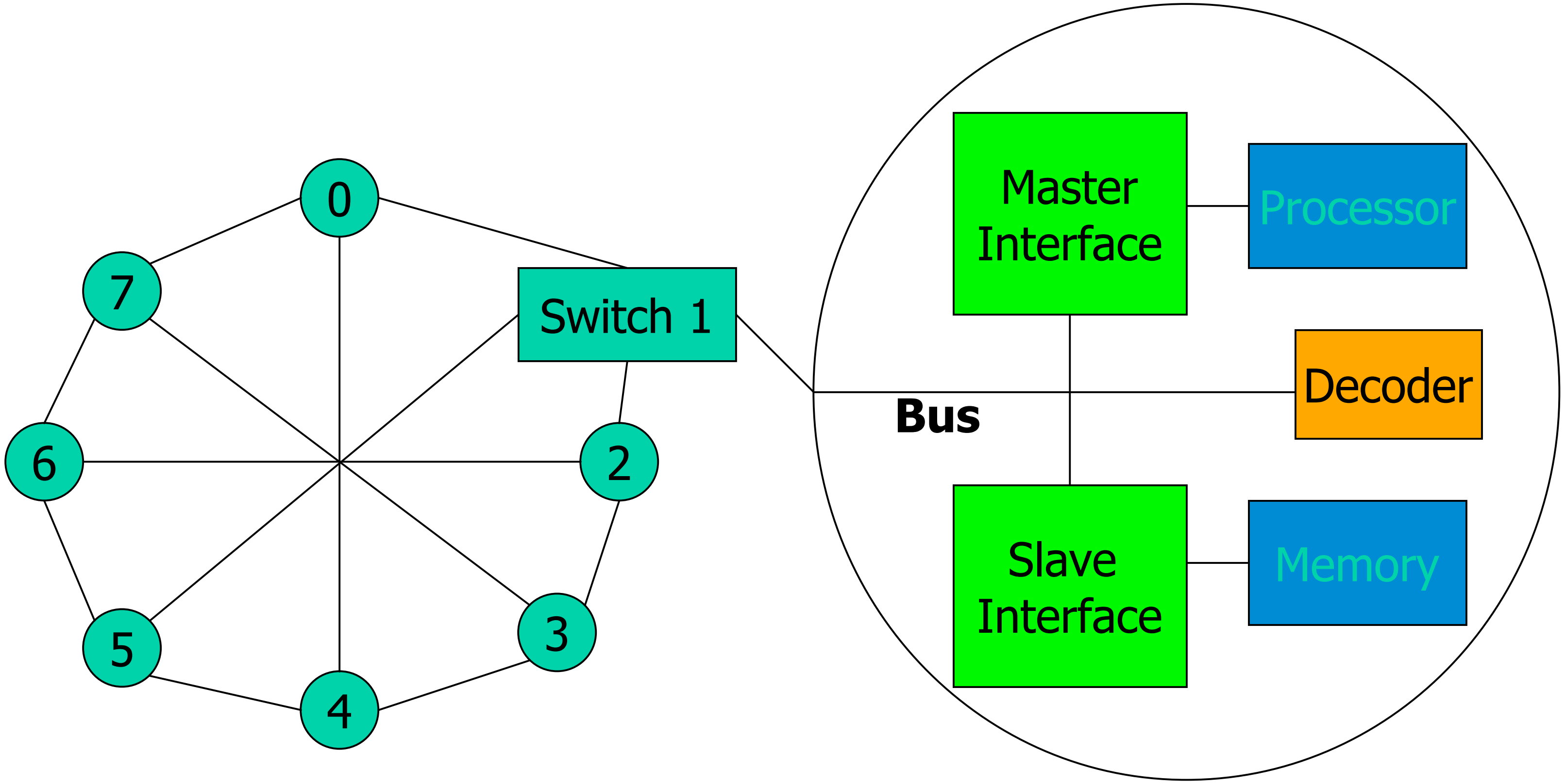
Initial Motivation: Octagon Network on Chip



- **Design by STmicroelectronics ref: DAC 2001 and IEEE MICRO 2002 by F. Karim *et al.***

- **8 Nodes**
- **Extensible to N = 4*i**
- **Bidirectional Links**
- **Simple Shortest Path Routing Algorithm**

Octagon System



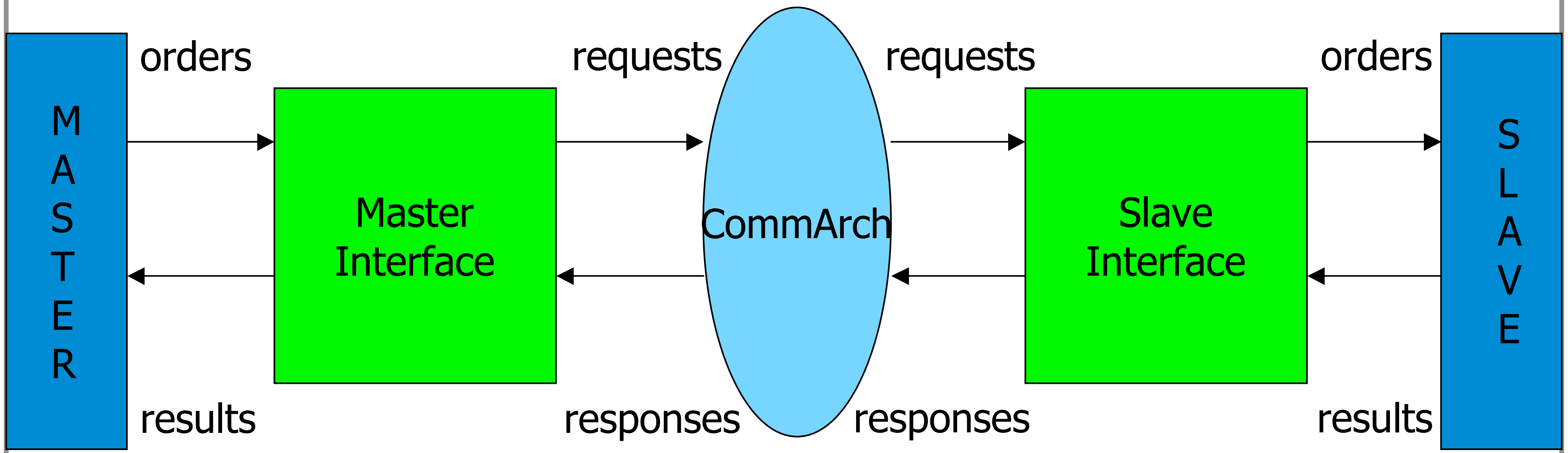
Parameterized Octagon Unit

Node System

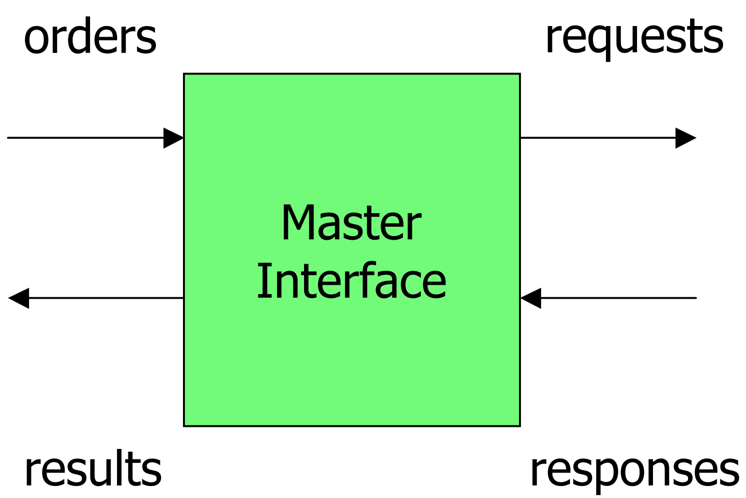
Outline

- Functional modeling of communications
  - Definition of transfers
  - Correctness Criteria
- Informal presentation of the Octagon
- ACL2 model of the Octagon

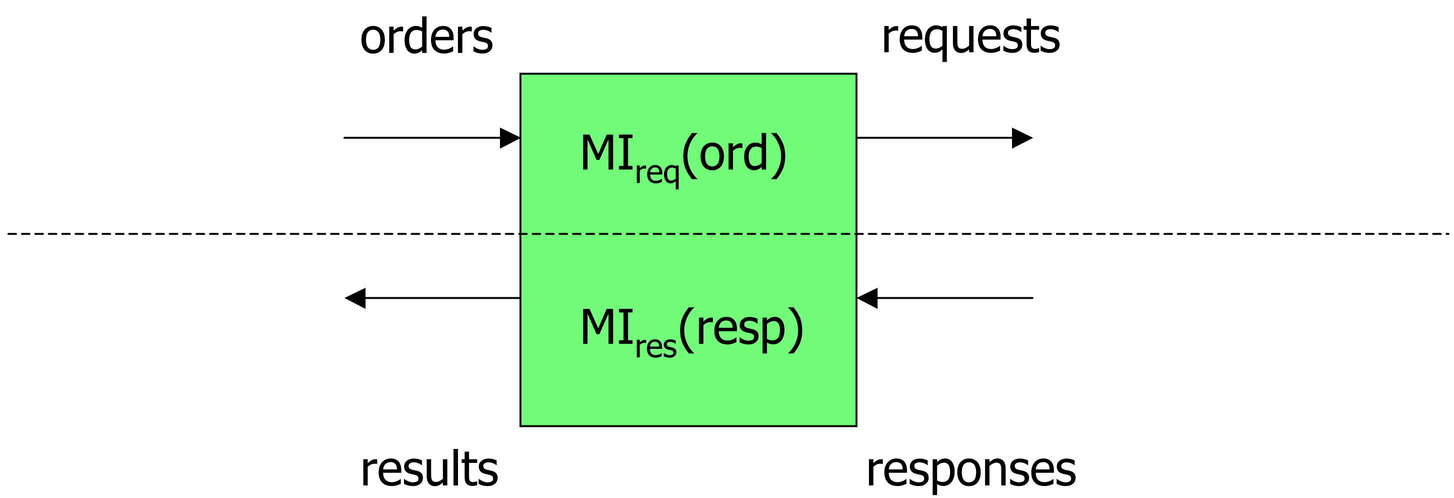
Communication Scheme



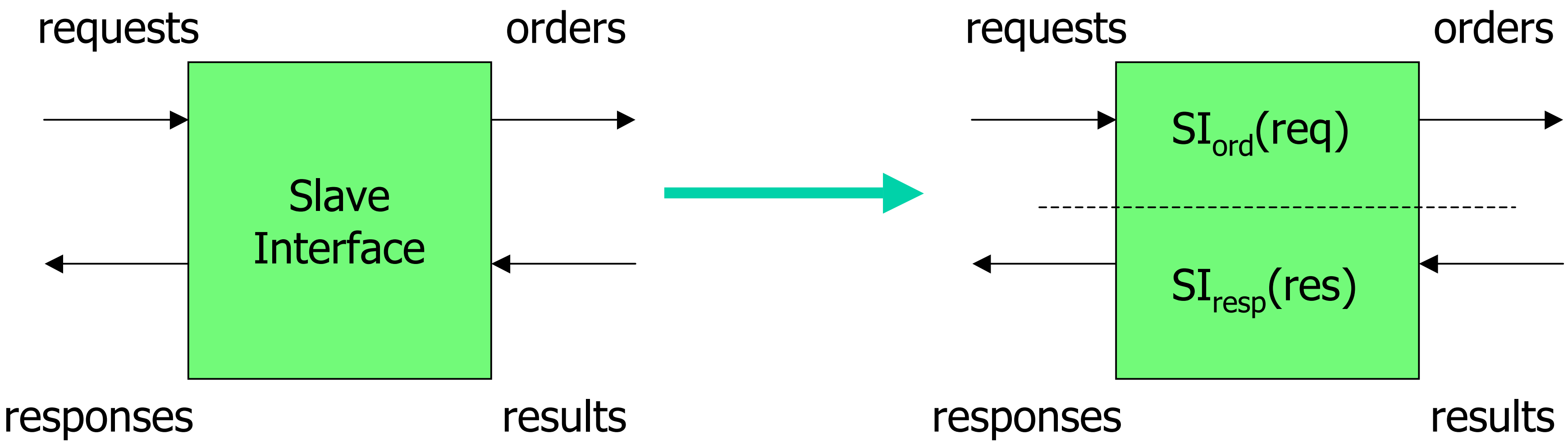
Functional Modeling



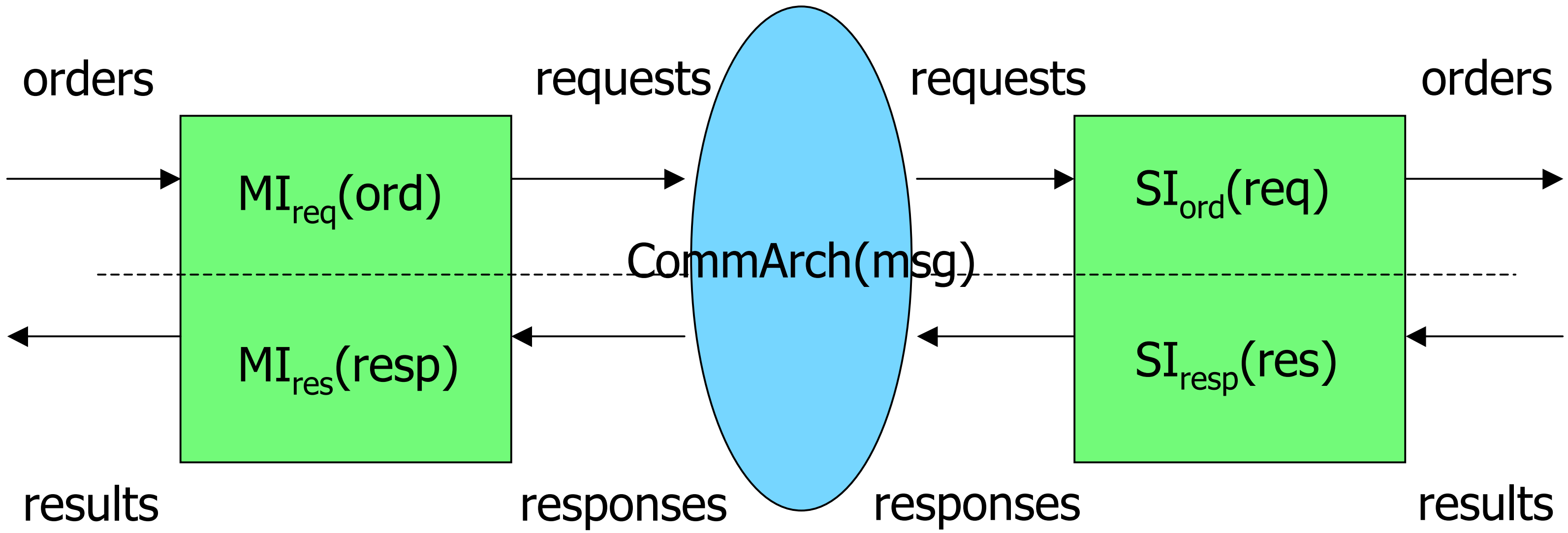
Functional Modeling



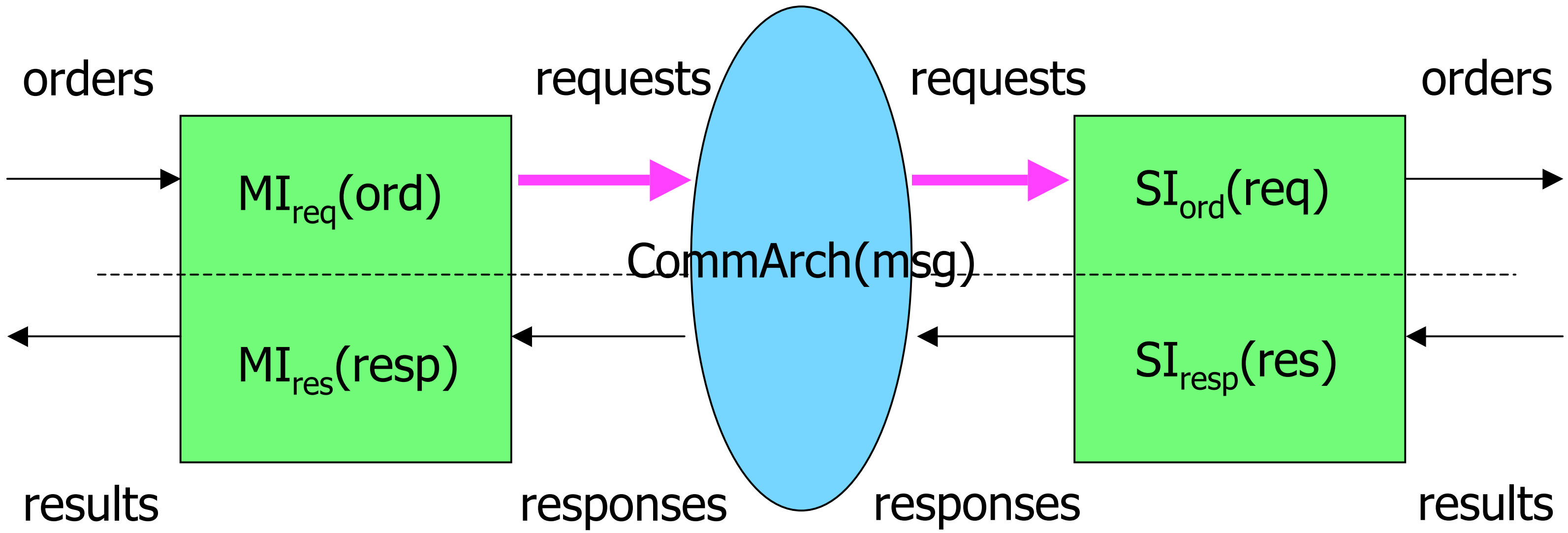
Functional Modeling



Definition of Transfers

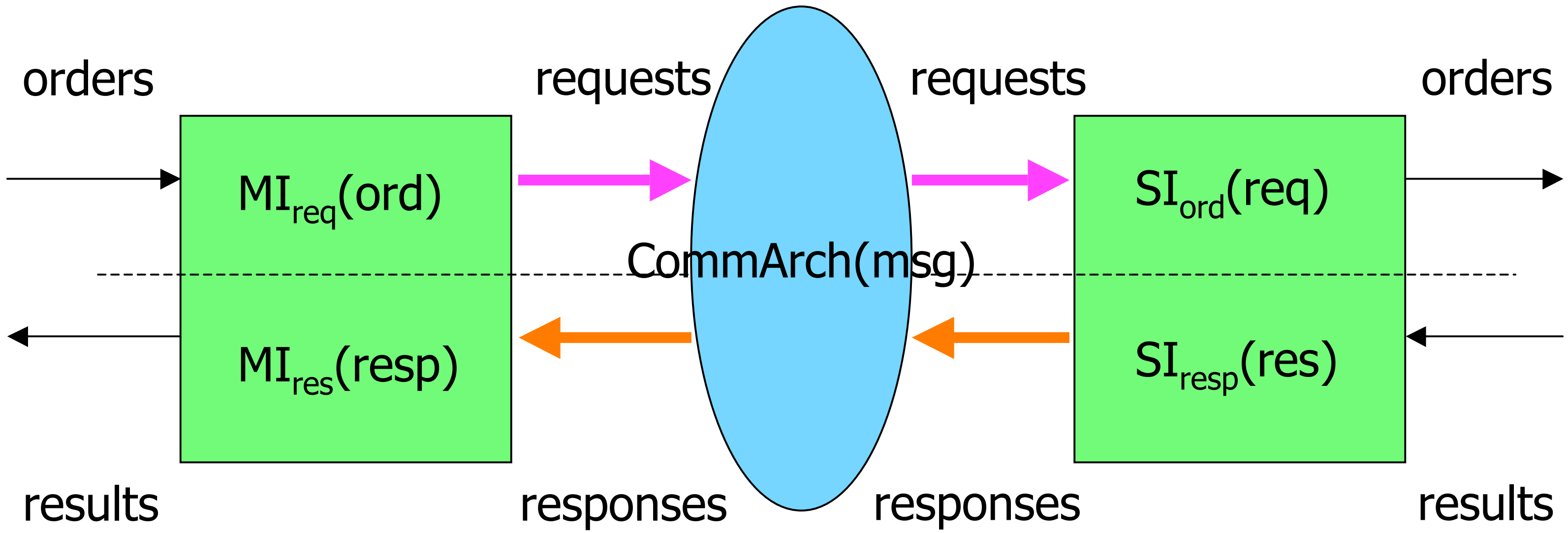


Definition of Transfers



$Trans\_ord(ord) = SI_{ord} \circ CommArch \circ MI_{req}(ord)$

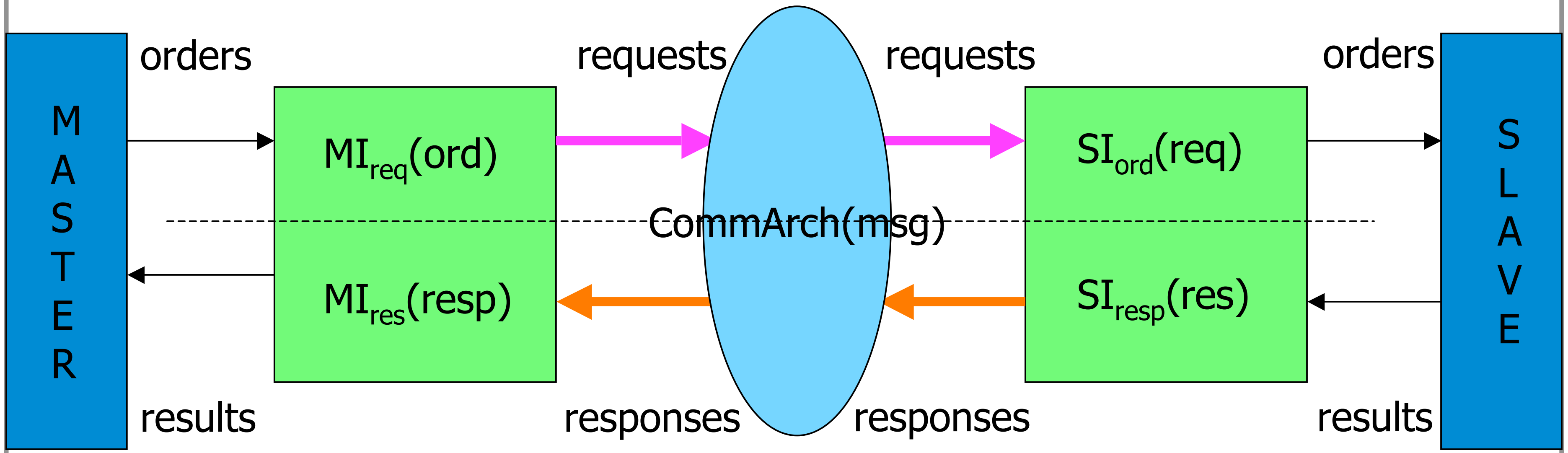
Definition of Transfers



Trans_ord(ord) = SI_{ord} o **CommArch** o MI_{req}(ord)

Trans_res(res) = MI_{res} o CommArch o SI_{resp}(res)

Correctness Criteria

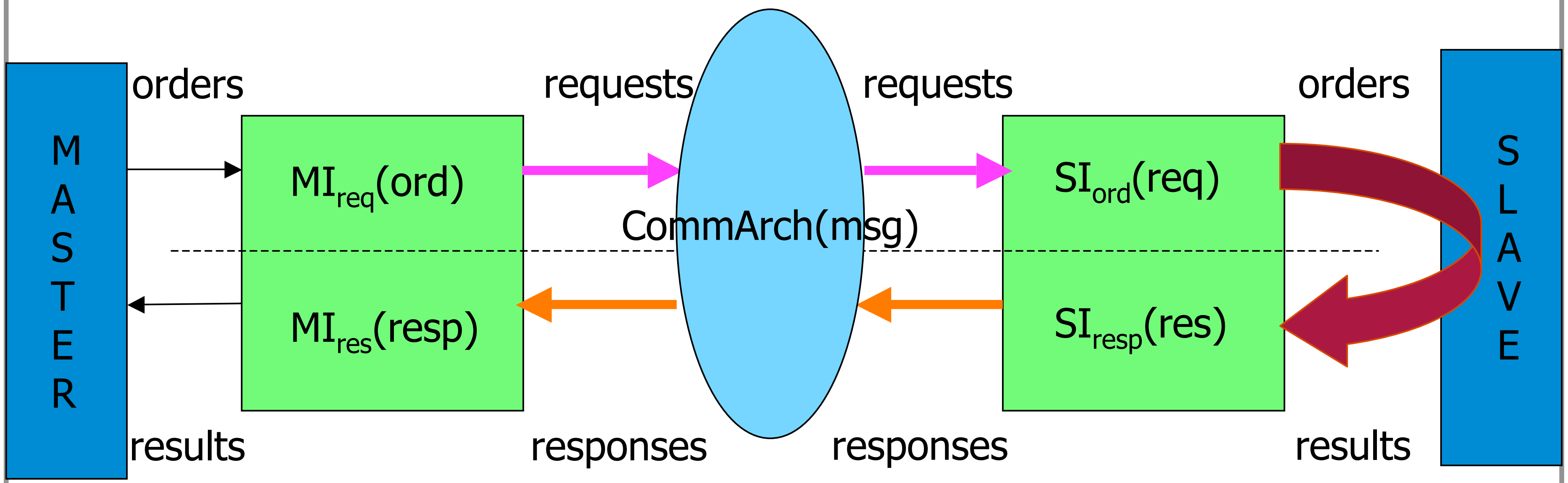


THM

Trans_ord(ord) = SI_{ord} o CommArch o MI_{req}(ord) = ord

Trans_res(res) = MI_{res} o CommArch o SI_{resp}(res) = res

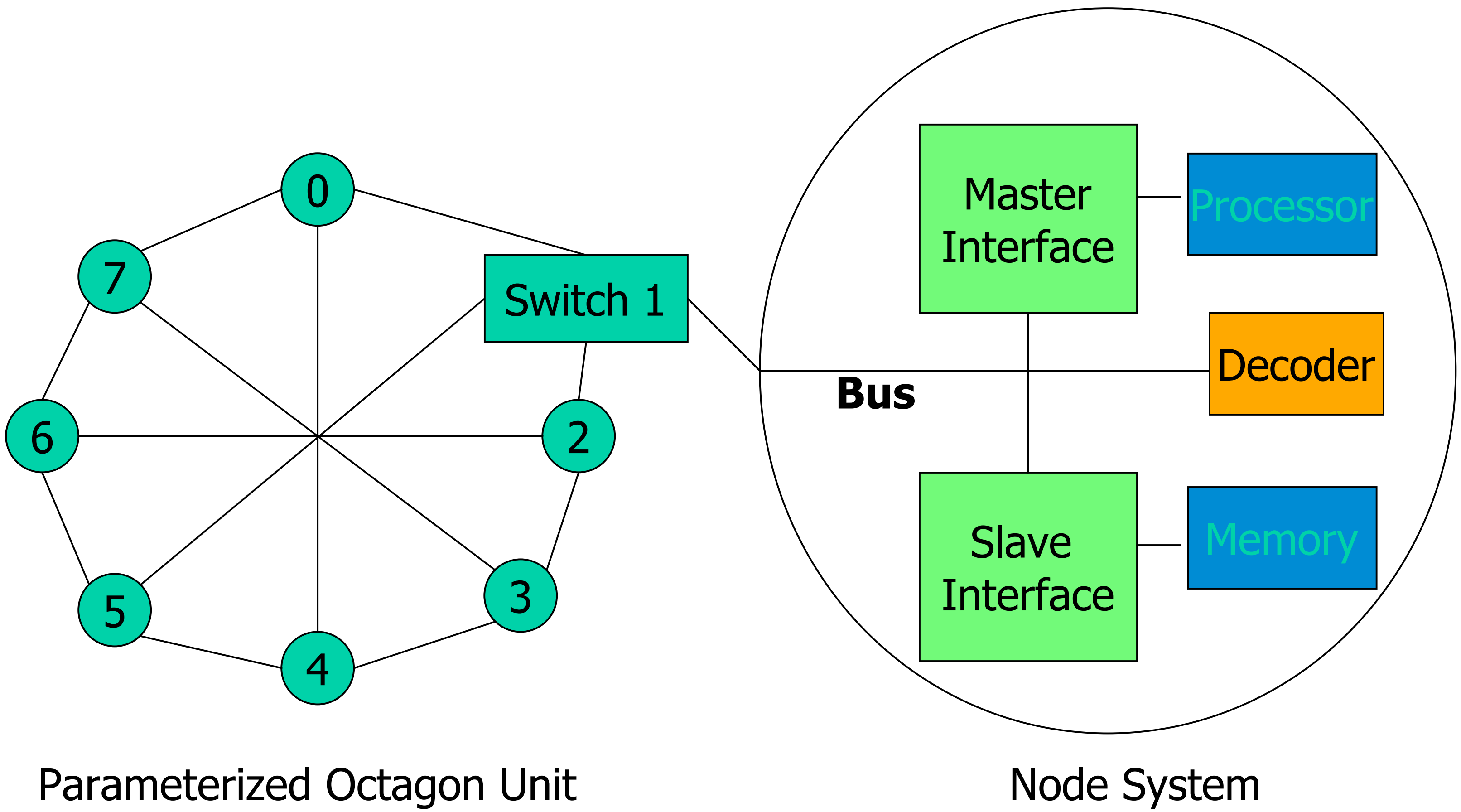
Correctness Criteria



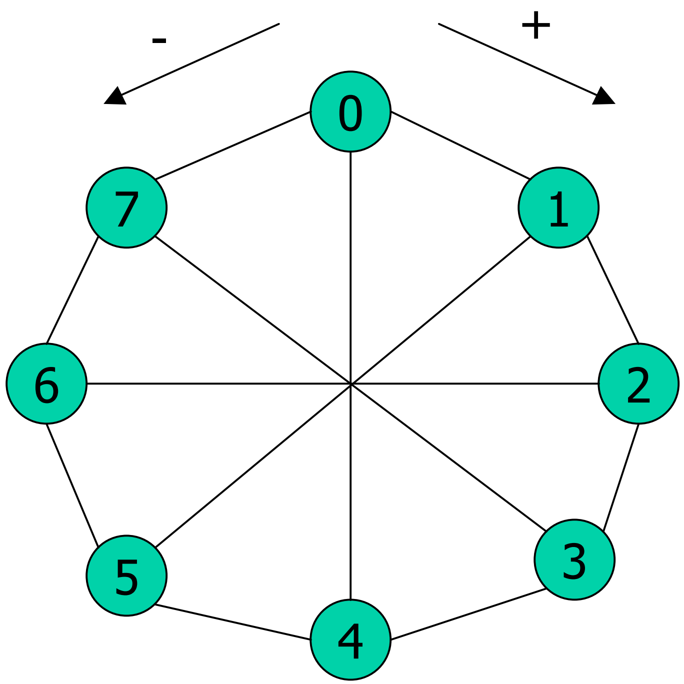
THM: Transfer(ord) = Slave(ord)

Transfer(ord) = Trans_res o Slave o Trans_ord(ord)

Octagon System



Routing Algorithm



If (dest – from) mod n ≤ n/4

then go clockwise (+)

If 3n/4 ≤ (dest-from) mod n

then go counter clockwise (–)

Else go across

Routing Definition

- Formals
  - Origin and destination
  - Number of nodes
- Result
  - A path which is a list of nodes
- Role
  - Compute the path between origin and destination

Routing Definition

```
(defun route (from dest n)
  (declare (xargs :measure (min (nfix (mod (- dest from) (* 4 n)))
                                (nfix (mod (- from dest) (* 4 n)))))))
 (cond ((or
           (not (integerp dest)) (< dest 0) (< (- (* 4 n) 1) dest)
           (not (integerp from)) (< from 0) (< (- (* 4 n) 1) from)
           (not (integerp n)) (<= n 0))
         nil)
        ((equal (- dest from) 0)
         (cons from nil))
        ((and   (< 0 (mod (- dest from) (* 4 n)))
                (<= (mod (- dest from) (* 4 n)) n))
         (cons from (route (n_clockwise from n) dest n)))
        ((<= (* 3 n) (mod (- dest from) (* 4 n)))
         (cons from (route (n_counter_clockwise from n) dest n)))
        (t
         (cons from (route (n_across from n) dest n)))))
```

Routing Theorem

```
(defthm CORRECTNESS_OF_ROUTE
   (implies (and (integerp from) (<= 0 from) (< from (* 4 n))
                 (integerp to) (<= 0 to) (< to (* 4 n)) (integerp n)
                 (< 0 n))
            (and ;; a route is a consp
                 (consp (route from to n))
                 ;; every node is an integer
                 (all_intp (route from to n))
                 ;; every node number is positive
                 (all_pos_intp (route from to n))
                 ;; every route contains no duplicate
                 (no-duplicatesp (route from to n))
                 ;; every node is less than the maximum of nodes
                 (all_inf_np (route from to n) (* 4 n))
                 ;; a route is made of available moves
                 (AvailableMovep (route from to n) n)
                 ;; the first node is the starting node
                 (equal (car (route from to n)) from)
                 ;; the last node is the final node
                 (equal (car (last (route from to n))) to))))
```

Messages not lost in the medium: (AvailableMovep lines)

Connection of a well-formed Master/Slave pair: (first/last node lines)

Scheduling Policy

- Circuit switched mode
  - A path is allocated during the complete transfer
- Formals
  - Travel list and two “working” variables
- Result
  - A set of non overlapping communications

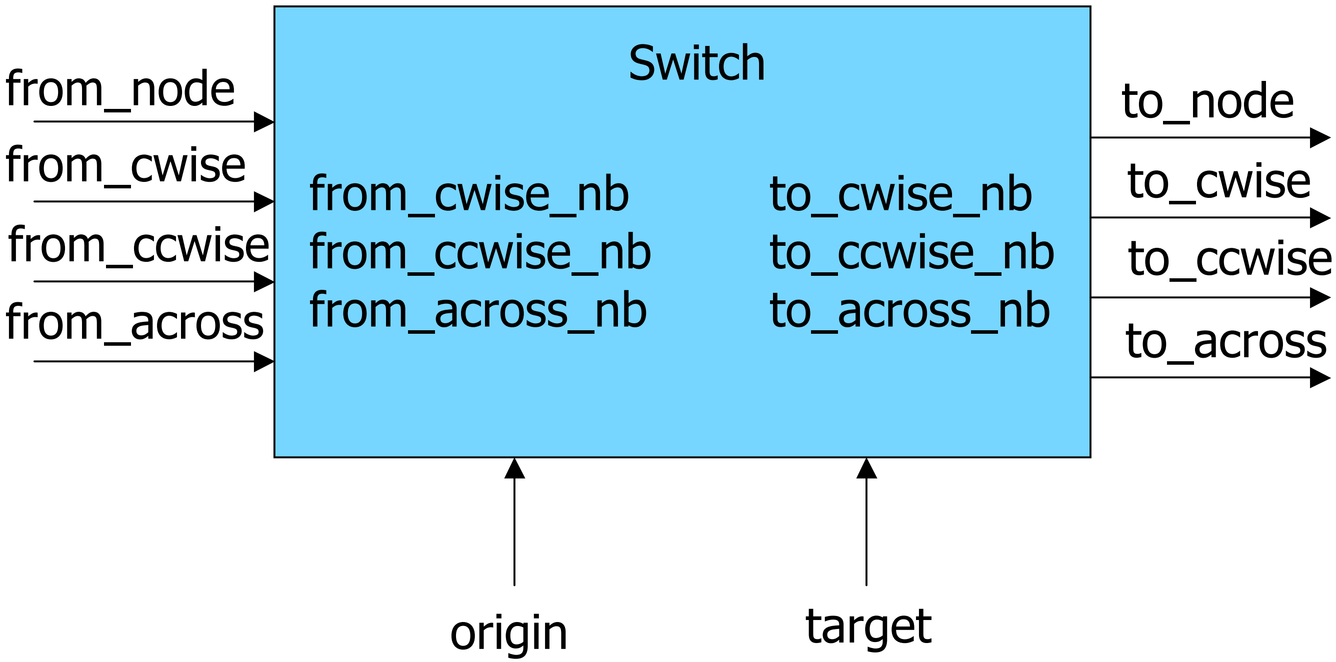
Scheduling Policy

```
(defun scheduler (tl non_ovlp_r prev)
  ;; extracts non overlapping communications of tl
  (if (endp tl)
      (rev non_ovlp_r)
    (let ((route_i (cdr (car tl))))
      (if (no_intersectp route_i prev)
          (scheduler (cdr tl)
                     (cons (car tl) non_ovlp_r)
                     (append route_i prev))
        (scheduler (cdr tl) non_ovlp_r prev)))))

(defthm all_no_duplicatesp_scheduler
  ;; if tl contains routes with no duplicate, then grab_nodes
  ;; of scheduler contains no duplicate
  (implies (all_no_duplicatesp tl)
           (no-duplicatesp (grab_nodes (scheduler tl nil prev)))))
```

Scheduler correctness $\wedge$ Routing Correctness } ⟹ Every scheduled transfer is a well-formed non-overlapping Master/Slave communication

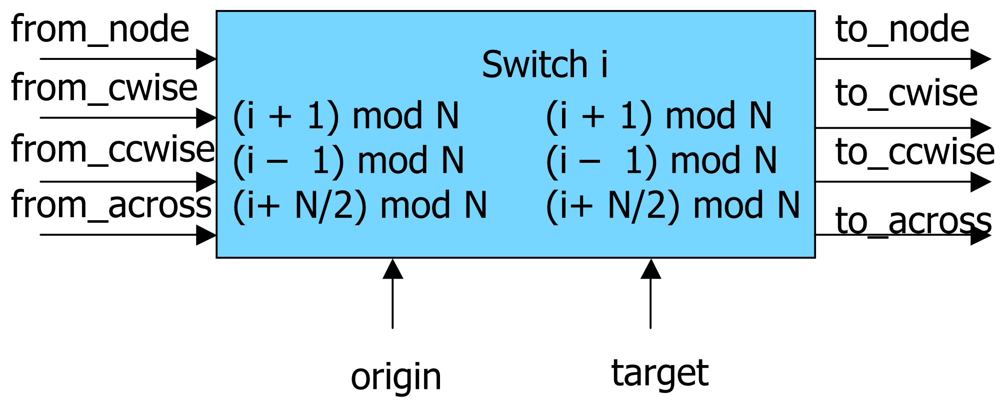
Switch



Switch
from_node
from_cwise
from_ccwise
from_across
from_cwise_nb
from_ccwise_nb
from_across_nb
to_cwise_nb
to_ccwise_nb
to_across_nb
to_node
to_cwise
to_ccwise
to_across
origin
target

Traveling Functions

- Model the interconnection structure
  - **Trip: Travel List x NumNode → Travel List**
  - Connection of Switches
  - Iterative calls to the switch function



Traveling Correctness

- We do not lose a message if its route is made of correct moves
  - **Trip(tl, Num_Node) = tl**
- We do lose a message if its route is not made of correct moves
  - **Trip(tl, Num_Node) ≠ tl**

Traveling Correctness

- We do not lose a message if its route is made of correct moves
  - **Trip(tl, Num_Node) = tl**
- We do lose a message if its route is not made of correct moves
  - **Trip(tl, Num_Node) ≠ tl**

```
(AvailableMovep (route from to n) n) -> Trip(tl, Num_Node) = tl
```

Memory Model

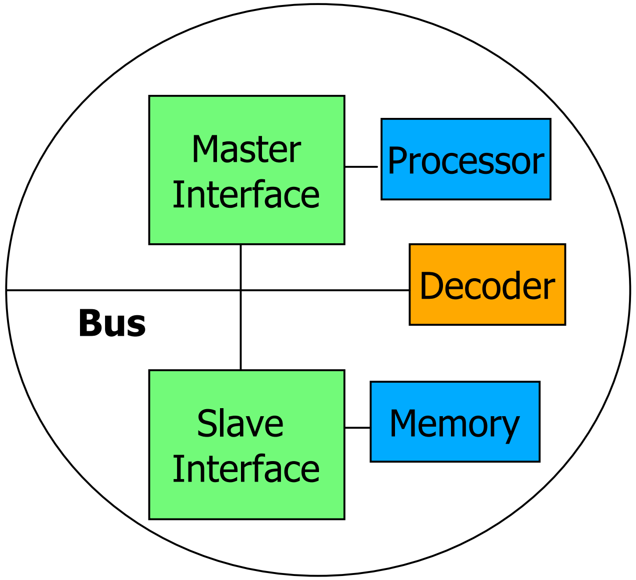
Local Memory =(d_0 d_1 ….d_{ms-1}) ;; manipulated using nth and put-nth

Local memory of node i

Global Memory =(d_0 d_1 d_2 … $d_{i.ms-1}$ $d_{i.ms}$ ….$d_{(i+1).ms-1}$ $d_{(i+1).ms}$ … $d_{n.ms-1}$)

Node System

- If a request arrives at a node
  - Function **Node** ≈ **SI_resp o Memory o SI_ord**
- If a response is coming
  - Function **Node** ≈ **MI_res**
- If a request is emitted
  - Function **Node** ≈ **MI_req**



Node System

Node System

```
(defun node (op loc dat Glob_Mem
            nw_stat nw_r_dat ;; stat and dat coming from the network
            nw_r/w nw_addr nw_dat ;; msg coming from the network
            IncomingResponse IncomingRequest ;; scheduler commands
            node_nb ;; node number
            ms) ;; size of local memory
  (if (equal op 'NO_OP) ;; node master is doing nothing
      ;; if there is a command from the scheduler, we do it
      (if (equal IncomingRequest 1) ;; a request is coming
          (let ((dec (decoder nw_addr ms node_nb)))
            (mv-let (st dat memo)
                    (nw_transfer nw_r/w nw_addr nw_dat Glob_Mem
                                 dec node_nb ms)
                    (mv 'NO_OP 'NO_DATA ;; to the master
                        st dat 'NO_MSG_DATA ;; res sent to th nw
                        memo))) ;; new memory
        (if (equal IncomingResponse 1)
            ;; a response is coming -> call MI_res
```

Node System

```
;; else the node master is doing a write or read operation
    (let ((dec (decoder loc ms node_nb)))
      (if (equal dec 1)
          ;; local transfer
        ;; else the node sends a msg to the nw
        (mv-let (r/w addr data)
                (mi_req op loc dat)
                (mv 'NO_OP 'NO_DATA ;; nothing to the master
                    r/w addr data ;; msg sent to the nw
                    Glob_Mem))))))
```

Node System

```
(defun nw_transfer (r/w addr data Glob_Mem sl_select node_nb ms)
  (mv-let (op loc dat)
          (si_ord r/w addr data sl_select ms)
          (mv-let (stat dat memo)
                  (memory op loc dat
                                  (get_local_mem Glob_Mem node_nb ms))
                  (mv-let (st d)
                          (si_resp stat dat sl_select)
                          (mv st d
                              (update_local_mem Glob_Mem memo
                                                node_nb ms))))))
```

Octagon Definition

```
(defun Octagon (op_lst N ms Glob_Mem)
  ;; model of the complete network, i.e. nodes connected to Octagon
  ;; runs the Network once and returns loc_done nw_done and the memory
  (mv-let (loc_done nw_op Glob_Mem1)
          ;; first we collect the messages and execute the local ops
          ;; we also get the non local requests
          (collect_msg op_lst nil nil Glob_MEM ms) ;; node function
          ;; then we compute the travel list
          (let* ((tl (make_travel_list nw_op nil ms N)) ;; Routing
                 ;; we extract the set of non-overlapping communications
                 (novlp (scheduler tl nil nil))
                 ;; we move every request to their destination
                 (tl_at_dest (trip novlp N)))
            (mv-let (cr_lst Glob_Mem2)
                    ;; we compute the response of every request (node)
                    (ComputeResponses tl_at_dest Glob_Mem1 ms nil)
                    ;; we move the responses back to their source node
                    (let ((tl_back (trip cr_lst N)))
                      ;; we compute the final result
                      (mv-let (nw_done Glob_Mem3)
                              (ComputeRes tl_back Glob_Mem2 ms nil)
                              (mv loc_done nw_done Glob_Mem3)))))))
```

Final Theorems

```
(defthm mem_ok_read_Octagon
  ;; we prove that in case of read transfers the memory is not
  ;; changed
  (implies (and (all_read_op_lstp op_lst)
                (all_non_loc_op_lstp op_lst ms)
                (all_node_nb_validp op_lst (* 4 N))
                (all_address_validp op_lst (* 4 N) ms)
                (equal (len Glob_Mem) (* (* 4 N) ms))
                (integerp N) (< 0 N)
                (true-listp Glob_Mem)
                (NODE_MEM_SIZEp ms))
           (equal (mv-nth 2 (Octagon op_lst N ms Glob_Mem))
                  Glob_Mem)))
```

Final Theorems

```
(defthm mem_ok_write_octagon
  (implies (and (all_write_op_lstp op_lst)
                (all_non_loc_op_lstp op_lst ms)
                (all_node_nb_validp op_lst (* 4 N))
                (all_address_validp op_lst (* 4 N) ms)
                (equal (len Glob_Mem) (* (* 4 N) ms))
                (integerp N) (< 0 N)
                (true-listp Glob_Mem)
                (NODE_MEM_SIZEp ms))
           (equal (mv-nth 2 (Octagon op_lst N MS Glob_Mem))
                  (good_mem_write
                   (scheduler
                    (make_travel_list
                     (mv-nth 1 (collect_msg op_lst nil nil
                                            Glob_Mem ms))
                     NIL ms N)
                    nil nil)
                   Glob_Mem ms)))
```

req_lst = the (scheduler …) argument above

(good_mem_write …) = (put-nth (addr req_lst_i) (data req_lst_i) Glob_Mem)

Conclusions

- Specification and validation of a parameterized Octagon system
  - Industrial case study
  - Functional framework
- First sketch of a methodology
  - 1^{st} design steps of generic NoC
- Original work
  - New application of ACL2
  - New application of theorem proving

Prospective Applications

- Packet and Wormhole Routing
- Protocols
  - Ethernet
- Buses
  - AMBA AHB (extension of ACL2'03)
  - Crossbars

Thank you !!