Validation of a Parameterized Bus Architecture Model

SCHMALTZ Julien and BORRIONE Dominique
TIMA Laboratory -VDS Group, Grenoble, France

ACL2 Workshop 2003 Boulder, CO
Outline

• The point to point connection
• Functional modeling
• Validation of the model
• One step forward: the bus case
The direct point to point connection

\[ O = \text{operation (read or write)} \]

\[ L = \text{location} \]

\[ D = \text{the data to be written} \]

\[ SD = \text{the read data} \]

\[ ST = \text{slave status} \]

ORDERS

RESULTS

MASTER APPLICATION

SLAVE APPLICATION
Introduction of the interfaces

MASTER APPLICATION

SLAVE APPLICATION

MASTER INTERFACE

SLAVE INTERFACE

O  L  D  S-master  D-master

O-slave  L-slave  D-slave  SD  ST

R/W  ADDR  DATA

SD  ST

REQUESTS  RESPONSES
Functional modeling

• Each component is represented by a function

• Time is abstracted away

• Functional composition is used to express sequential communication events
Modeling the master interface

(defun master_interface (O L D SD ST)
  (if (equal O 'read)
      (list (list 1 L D) (list SD ST))
      (list (list 0 L D) (list SD ST)))))
Modeling the master interface

```
(defun master_interface (O L D SD ST)
  (if (equal O 'read)
      (list (list 1 L D) (list SD ST))
      (list (list 0 L D) (list SD ST))))
```

Master application

Slave interface
Modeling the master interface

(defun master_interface (O L D SD ST)
  (if (equal O 'read)
      (list (list 1 L D) (list SD ST))
      (list (list 0 L D) (list SD ST))))

(ADDR x)
Modeling the master interface

```
(defun master_interface (O L D SD ST)
  (if (equal O 'read)
      (list (list 1 L D) (list SD ST))
      (list (list 0 L D) (list SD ST))))
```

Master application

Master Interface

Slave interface
(defun master_interface (O L D SD ST)
  (if (equal O 'read)
      (list (list 1 L D) (list SD ST))
      (list (list 0 L D) (list SD ST))))
(D-master x)
Modeling the master interface

(defun master_interface (O L D SD ST)
  (if (equal O 'read)
    (list (list 1 L D) (list SD ST))
    (list (list 0 L D) (list SD ST))))

(S-master x)
Modeling the slave interface

(defun slave_interface (R/W ADDR DATA SD ST)
  (if (equal R/W 1)
      (list (list 'read ADDR DATA) (list SD ST))
      (list (list 'write ADDR DATA) (list SD ST))))
Transfers modeling

- Each transfer is modeled by a functional composition
  - \( \text{trans}_M\text{to}_S(\text{O L D}) = \text{slave}_{\text{interface}} \bullet \text{master}_{\text{interface}}(\text{O L D}) \)
  - \( \text{trans}_{S\text{to}M}(\text{SD ST}) = \text{master}_{\text{interface}} \bullet \text{slave}_{\text{interface}}(\text{SD ST}) \)
Transfer validation

\[(O, L, D) \equiv (O\text{-slave}, L\text{-slave}, D\text{-slave})\]

\[(D\text{-master}, S\text{-master}) \equiv (SD, ST)\]

• Correctness Criteria
  – The introduction of the interfaces does not modify the orders nor the results
One step forward: the bus case

Master Interface

Arbiter

Decoder

Slave Interface

HADDR

HSEL

HGRANT

Result

Order
The proof strategy

• 1\textsuperscript{st}: prove the decoder and the arbiter correct
  – Prove that the decoder selects the slave possessing the data required for the transfers
  – Prove that the arbiter grants the master with the highest priority

• 2\textsuperscript{nd}: prove the point to point connection correct
  – This takes place between a well chosen master interface and a well chosen slave interface
Modeling the address decoder

HADDR < Card_S x MEM_SIZE
UNADDR = HADDR mod MEM_SIZE
SEL = HADDR / MEM_SIZE

(defun select (Card_S SEL)
  (cond ((not (integerp Card_S)) nil)
        ((<= Card_S 0) nil)
        ((equal SEL 0)
         (cons 1
              (select (1- Card_S) (1- SEL))))
        (t
         (cons 0
              (select (1- Card_S) (1- SEL))))))

(defun decoder (MEM_SIZE Card_S HADDR)
  (select Card_S (floor HADDR MEM_SIZE)))
Validation of the decoder function

Selection of the right slave

(defthm ith_select_=1
  (implies (and (integerp i) (integerp Card_S)
    (>= i 0) (> Card_S i))
    (equal (nth i (select Card_S i)) 1)))

Uniqueness of the selection

(defthm pth_select_=0
  (implies (and (integerp p) (integerp Card_S)
    (<= 0 p) (< p Card_S)
    (not (equal p i)))
    (equal (nth p (select Card_S i)) 0))
  :hints ("GOAL"
    :induct (function_hint_th2_select p Card_S i))))
Modeling the bus arbiter

MREQ is a matrix with P lines and N columns

HGRANT is a list of bits computed according to a priority scheme

MREQ

HGRANT

Last_Granted
N, P

<table>
<thead>
<tr>
<th></th>
<th>N0</th>
<th>N1</th>
<th>N2</th>
<th>N3</th>
</tr>
</thead>
<tbody>
<tr>
<td>P0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>P1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
First Step: RLINE search

(defun stage_P (L)  ; returns the line number
  (cond ((endp L) 0)  ; for the highest priority request
    ((no_requestp_matrix L) 0)  ; if empty list or no request returns 0
    ((not (no_requestp (car L))) 0)  ; we count the number of stages containing
    (t  ; no request until we meet a stage with
      (+ 1 (stage_P (cdr L))))))  ; at least one request

(deffm prior_scheme  ; we prove that each stage j prior to the
  (implies (and (equal (stage_P L) i)  ; returned one i contains no request
                (< j i) (<= 0 j)  ; returned one i contains no request
                (no_requestp (nth j L))))
2\textsuperscript{nd} Step: who is the next owner?

\begin{verbatim}
(defun round_robin (RLINE Last_Granted)
  (cond ((no_requestp RLINE) 0)
        ((no_requestp (lastn (1+ Last_Granted) RLINE))
         (find_next_1 (firstn (1+ Last_Granted) RLINE)))
        (t
         (+ (1+ Last_Granted)
             (find_next_1 (lastn (1+ Last_Granted) RLINE)))))
\end{verbatim}
2\textsuperscript{nd} Step: validation

\begin{verbatim}
(defthm no_deadlock
  (implies (and (integerp i)
                (<= 0 i)
                (equal (nth Last_Granted RLINE) 1)
                (list_of_1_and_0 RLINE)
                (not (equal Last_granted i)))
  (implies (equal (nth i RLINE) 1)
            (not (equal (round_robin RLINE Last_Granted) Last_Granted))))

:hints ("GOAL" :use lemma1_no_deadlock
            :in-theory (disable lemma1_no_deadlock firstn)))

:rule-classes ((:rewrite :match-free :all))
\end{verbatim}
3\textsuperscript{rd} Step: compute and build

Compute the number of the granted master

\begin{align*}
\text{(defun master_num} (\text{MREQ N Last\_Granted}) \\
+ (* (\text{stage\_P MREQ}) N) \\
(\text{round\_robin} (\text{nth} (\text{stage\_P MREQ}) \text{MREQ}) \text{Last\_Granted})))
\end{align*}

Build the output list HGRANT

\begin{align*}
\text{(defun arbiter} (\text{N P MREQ Last\_Granted}) \\
(\text{select} (* \text{N P}) (\text{master\_num} \text{MREQ N Last\_Granted})))
\end{align*}
Proof of transfers

• 1\textsuperscript{st}: decoder and arbiter, OK

• 2\textsuperscript{nd}: Prove the point to point connection correct
  – $\text{Trans}_M\_t\_o\_S(0 \ L \ D) = (O \ (\text{mod} \ L \ \text{MEM\_SIZE}) \ D)$
  – $\text{Trans}_S\_t\_o\_M(SD \ ST) = (SD \ ST)$
Conclusion and Future Work

• Conclusion
  – 20 functions, 60 theorems, proof time about 30 seconds
  – Protocol proven correct for an arbitrary number of masters and slaves

• Future Work
  – Modeling on chip networks
  – Test implementations against the formal specification