

# Design Interface and Verification-I

Introduction to a model of interfacing components in a design; and its verification.

Mahapatra-A&M-Fall'00

1

## Interfacing components

- *Process* is an independent part of a computation done concurrently with other parts. (VHDL, UNIX)
- *Thread* is used in Java.
- *Component* in Codesign:

COMPONENT *name*

*declaration* ,local quantities of components used in computation  
*computation*

Mahapatra-A&M-Fall'00

2

# Telephone switch components

- ◆ *COMPONENT unit* , ( physical phone that detects the off-hook, sends id, and disconnects)

*p:phone\_connector*

....(\* computation \*)

*END unit*

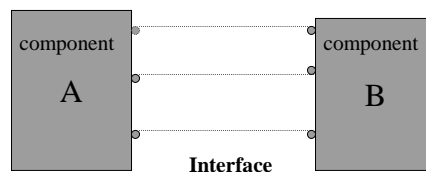
- *COMPONENT connect* , (check receiver not busy, free capacity : arbiter)
- *COMPONENT transfer* , (transmits digitized sound from common memory, data transmission constantly in both directions)

Mahapatra-A&M-Fall'00

3

# Interfacing components

- Interface determines the coordination of the components including their data transfer and synchronization.
- Codesign demands heterogeneous interface modeling.
  - Can not favor any one existing technology or signaling discipline alone.



Mahapatra-A&M-Fall'00

4

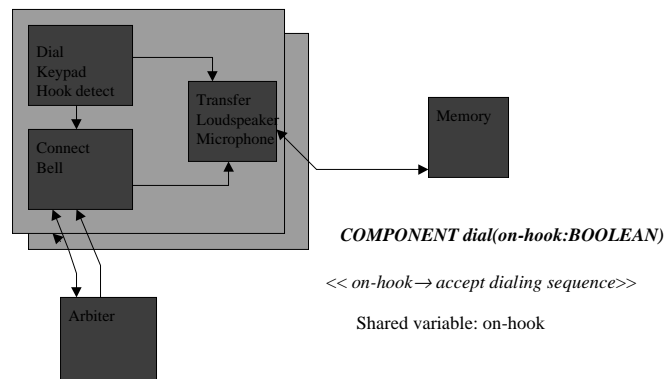
## Interfacing components (contd.)

- Interface model allows components to share one or more state variable. (i.e. simple Boolean, integer, buffer and other data type)
  - Sharing  $\Rightarrow$  value of state variable in interface may be changed by several different components.
  - Undisciplined used of shared variable can lead to time dependent design errors which can be difficult to locate.

Mahapatra-A&M-Fall'00

5

## Telephone switch



Mahapatra-A&M-Fall'00

6

## More Components of telephone switch

After receiving dialing sequence, the dialing component has the number until to be called, this number is exchanged with the transfer component and therefore part of the interface.

*COMPONENT dial(on-hook:BOOLEAN; rec\_number; ....)*

Together the *on-hook* and *rec-number* makes up the interface between the dial and connect components of one unit, they are grouped as record.

*TYPE InterfaceDialConn = (\*Interface: dial-connect component\*)*

*RECORD*

*on-hook: BOOLEAN (\*indicate receiver on hook\*)*

*rec\_number: number (\*Dialled phone\*)*

*END*

Mahapatra-A&M-Fall'00

7

## Physical Realization of State Variable

- Software: as program variable in memory
- hardware: wire (connecting sub-circuits)
  - ◆ wires contain current value of state variable, provision of refresh to remain current

A state variable  $x$  is interpreted as a function from *Time* to its range of values, e.g.,  $x$  is boolean, we have  $M_b$ ,  $x: \text{Time} \rightarrow B$

Example: Analog signal coming from microphone of a telephone. Signal can be represented by state variable (interface)  $M_v$ ,  $\text{voice}: \text{Time} \rightarrow \text{Freq.}$

The interface of an AD converter transforming analog input to discret 8-bit sample can be described as follows:

*COMPONENT adconvert(in: analog\_frequency; out: [0...255])*

Mahapatra-A&M-Fall'00

8

# Verification

- Verification is important for non-trivial design projects.
- Ideally, all exhaustive check is required but seldomly possible in practice.
- Formal methods are more recent.
  - Formality is a possibility but should not be mandatory.
- Three kinds of formal methods are used at different stages of design process:
  - Interface verification
  - design verification
  - Implementation verification

Mahapatra-A&M-Fall'00

9

# Interface verification

What is interface verification?

- Separate design to distinct components is common
  - components interact through interface using coordination mechanism following the protocols
    - Designers might treat details on signaling differently across various components. Leads to inconsistency.
- ⇒ Interface verification checks this inconsistencies.

Mahapatra-A&M-Fall'00

10

## Design verification

What is design verification?

- Consists of verifying selected key requirements of incomplete models.
- Example: Arbiter
  - At most one device has access to common resources (mutual exclusion). Once the interface to bus has been designed, it is possible to verify that it does not violate the mutual exclusion property, *even if other components of the design are still missing.*

## Implementation verification

- To construct efficient product, it is necessary to refine initial design into concrete realization - typically includes a number of restrictions. (e.g. restrict integer values to a certain range so that you use fixed number of bits)
- Clearly, it requires that the concrete realization has been done and hence implementation verification is relevant rather late in design process.
- Now available in commercial design system.

## Simplified Arbiter as an example

*COMPONENT* arbiter(*reql*, *grl*, *reqr*, *grr*: *BOOLEAN*)

*INITIALLY* *grl* = *FALSE* *grr* = *FALSE*

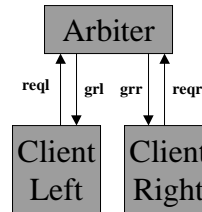
<< *reql*  $\wedge$   $\neg$ *grr*  $\rightarrow$  *grl* := *TRUE*>> //

<<  $\neg$  *reql*  $\wedge$  *grl*  $\rightarrow$  *grl* := *FALSE*>> //

<< *reqr*  $\wedge$   $\neg$ *grl*  $\rightarrow$  *grr* := *TRUE*>> //

<<  $\neg$  *reqr*  $\wedge$  *grr*  $\rightarrow$  *grr* := *FALSE*>>

*END* arbiter



Would check *grr* and *grl* are never both true. This is *mutual exclusion*, to be asserted,  $\neg$  (*grl*  $\wedge$  *grr*).

Reading assignment: Section 6.5.1 of Text

Mahapatra-A&M-Fall'00

13

## Interface Verification

- How is it possible to verify that different components have a consistent view of their interface?
  - Allow them to have different views as long as these are not in conflict.
- EX: packet in communication protocol.
  - An uninterrupted collection of bits to be transmitted vs. structured packet with different fields indicating addresses, control and checksum.

In codesign computational model, interface of component consists of “set of state variable” and a “protocol”.

Mahapatra-A&M-Fall'00

14

## Interface verification contd.

- Simple arbiter protocol: four-phase
  - client requests the privilege by setting  $req$  to true
  - when  $gr$  becomes true, the client enter its critical section
  - when leaving it,  $req$  is set false
  - $gr$  becomes false following  $req$  is set false.

- This can be expressed formally as follows:

$req.post \neq req.pre \Rightarrow grant.post \neq req.post$

$grant.post \neq grant.pre \Rightarrow grant.post = req.post$

Both the arbiter and the client may assume that the other components follow this protocol. To verify that the changes made to  $gr$  by arbiter follow the protocol, it must be shown that all its transitions obey:

$gr.post \neq gr.pre \Rightarrow req.post = gr.post$ ;

Similarly, for client:  $req.post \neq req.pre \Rightarrow req.post \neq gr.post$

Mahapatra-A&M-Fall'00

15

## Design verification

- Requirements of a design are formalized as predicates constraining the computation. E.g. two grant signals never given simultaneous access.
- Example: Let us verify invariant in a design.

Invariant: Sub set of state space containing initial state.  
Further, there must not be any transition from a state with in the subset to a state.  
Hence, invariant describes properties that hold through the computation.

- $I(s)$  on a state  $S$  is invariant
- $I(pre) \wedge t(pre, post) \Rightarrow I(post)$
- To show that  $\neg (grr \wedge grr)$  is an invariant for arbiter, four implications must be shown. I.e.  $reql \wedge \neg grr \wedge grr \Rightarrow \neg (grr \wedge grr)$

Mahapatra-A&M-Fall'00

16

# Implementation verification

- Refine initial abstract design into concrete realization.
  - This includes number of restrictions
- Both abstraction and realization are described formally. (abstract design, concrete design)
- Abstract design consists of many possible scheduling algorithms vrs. concrete design is one of the possible realization.
  - Thus, concrete design wont exhibit all the behavior of abstract design
- Refer pp226-227 for examples of concrete design of arbiter and the abstract arbiter.
  - To show that the concrete design is a refinement, one must show that any behavior exhibited by concrete design is also a possible behavior of abstract design.