Designing an Asynchronous Bus Interface

Joep Kessels & Ad Peeters (Philips Research)

Torsten Kramer (Kramer Consulting)

Markus Feuser & Klaus Ully (Philips Semiconductors)

Contents

• Asynchronous design SFR-bus 80C51

• Problems in the design of an asynchronous bus

• Communication means for modular and efficient design

• Transformational design (3 steps)
80C51 in System Chips

- 80C51 used in pager, telephone and smart card chips
- to control peripheral units: UART, timers, DES, ..
- Each system chip has different set of peripheral units
- Final bus design evolved gradually

SFR-Bus

```
80C51 Controller

\[\text{SFR-bus}\]

\[\text{Peripheral Units}\]

\[\text{UART}\]

\[\text{DES}\]
```
Required properties SFR bus

- Plug-and-play compositional
- Software errors should not lead to deadlocks
- Atomic read-modify-write accesses

Bus design with point-to-point channels

Acc<sub>0</sub>, ..., Acc<sub>n</sub>: chan <SFR address, write>
Read<sub>0</sub>, ..., Read<sub>n</sub>: chan byte
Write<sub>0</sub>, ..., Write<sub>n</sub>: chan byte

Central routing by CPU
Bus design with Tangram channels

GlbAcc: chan broad <[0..127],bool>
{<global address,write>}
GlbWrite: chan narrow narb byte
GlbRead: chan multi narb byte

Distributed routing by peripheral units and channels

First design

Master (CPU)

GlbAcc
GlbWrite
GlbRead

Slave Unit

command: <start, Task>
status: <interrupt, …>

GlbAcc: chan broad <[0..127],bool>
{<global address,write>}
GlbWrite: chan narrow narb byte
GlbRead: chan multi narb byte
Programs first design

```
forever
  do GlbAcc?<GlbAdd,write>;
     if MyAddress(GlbAdd) then if write
       then GlbWrite?SFRs[locadd(GlbAdd)];
          if start
            then Execute Task
          fi
       else GlbRead!SFRs[locadd(GlbAdd)]
      fi
  od

{No interference by slave unit}
```

Second design: enabling re-use

```
LocAcc: chan [<0..3>,bool>
{point-to-point <local address, write>}
```
Programs second design

```
forever
do GlbAcc?<GlbAdd,write>;
    if MyAddress(GlbAdd)
        then
            LocAcc!<locadd(GlbAdd),write>
        fi
    od

forever
do LocAcc?<LocAdd,write>;
    if write
        then GlbWrite? SFRs[LocAdd];
            if start
                then Execute Task
            fi
        else GlbRead! SFRs[LocAdd]
            fi
    od
```

bus unit

peripheral unit

Communication through variables

```
Active side                      Passive side

 Session ->
            Vs ———— Vr

 write Vs;
 Session!;

 wait probe(Session);
 (read Vs || Write Vr);
 Session?;

 read Vr
```

Philips Research

PHILIPS
Third design: saving area & preventing deadlocks

**Programs third design**

```
<GlbaAdd,write>:= <globadd,false>;   GlbAcc!
{Rdata contains information read}
<Wdata,write>:= <f(Rdata),true>;   GlbAcc!
forever
  do wait probe(GlbAcc);
  if MyAddress(GlbAdd)
    then LocAcc<locadd(GlbAdd),write>;
        if write
        then LocWrite!Wdata
        else LocRead?Rdata
    fi
  fi;
GlbAcc?
od
```

read-modify-write by master

bus unit
Peripheral unit

forever
do LocAcc?<LocAdd,write>;
  if write
    then GlobWrite?SFRs[LocAdd];
      if start then Execute Task fi
  else GlobRead!SFRs[LocAdd]
  fi
od

Fourth design: eliminating unacceptable delays

forever
do LocAcc?<LocAdd,read,write>;
  if read
    then LocRead!SFRs[LocAdd]
  fi;
  if write
    then LocWrite?SFRs[LocAdd]
      STA:= start
  fi
od

LocAcc: chan multi arb <[0..3],bool,bool>
  (locadd,read,write)
LocWrite: chan multi narb byte
LocRead: chan narrow narb byte
Fourth design: eliminating unacceptable delays

forever
do wait STA:
  Execute Task
od

**Co-processor**

LocAcc: chan multi arb <[0..3],bool,bool>
  {locadd,read,write}

LocWrite: chan multi narb byte
LocRead: chan narrow narb byte

---

Final design

**Master**

**Bus Unit**

**SFR Unit**

**Co-proc**

---

Philips Research
Conclusions

- Problems in bus design using conventional channels:
  - obtaining plug-and-play compositionality
  - avoiding deadlocks
  - avoiding busy waiting
  - avoiding redundant registers (broadcast)
- Problems are solved using:
  - safe communication through variables in sessions
  - (self-routing) narrowcast and multiple-sender channels

Design allowing multiple masters

Mutex: 
```plaintext
 forever do  P!;  V?  od
```

Read-modify-write by Master: 
```plaintext
 P?;  read-modify-write access:  V!
```
Protocol conversion

Master

\[<\text{GlbAdd}, \text{rmw}, \text{write}> := <\text{globadd}, \text{true}, \text{false}>;\]
\[\text{GlbAcct};\]

\[<\text{Wdata}, \text{write}> := \langle f(\text{Rdata}), \text{true} \rangle;\]
\[\text{GlbAcct};\]

Bus unit

wait probe(\text{GlbAcc});

\[\text{LocAcc}!\langle \text{LocAdd}, \text{true}, \text{true} \rangle;\]

wait probe(\text{GlbAcc});

SFR unit

\[\text{LocRead}?\text{Rdata};\]

\[\text{LocWrite}\text{!Wdata}\]

\[\text{GlbAcc}?;\]

wait probe(\text{GlbAcc});

\[\text{GlbAcc}?;\]