# Interface Synthesis

Kris Kuchcinski

Krzysztof.Kuchcinski@cs.lth.se



## Communication Synthesis

- After system partitioning we got a set of tasks assigned to system components (processors executing software + hardware components); these processes are communicating through *abstract channels*.
- Tasks are also interacting with peripheral devices and other external interfaces.
- Communication synthesis has to generate hardware and software which interconnects the system components and enables processes to communicate with each other, with peripheral devices and other interfaces.



## Communication Synthesis (cont'd)

- Communication synthesis, as a top-down design task, is performed in three main steps:
  - Channel binding,
  - Communication refinement,
  - Interface generation.
- After communication synthesis, the initial system specification results in a specification which can be directly synthesised to a physical implementation.



## Channel Binding

- Abstract channels have to be implemented using physical communication components:
  - resources have to be allocated which support communication throughout the system,
  - abstract channels have to be partitioned and the resulted groups are bound to the allocated resources,
  - messages corresponding to channels in one group are multiplexed on a shared communication component.

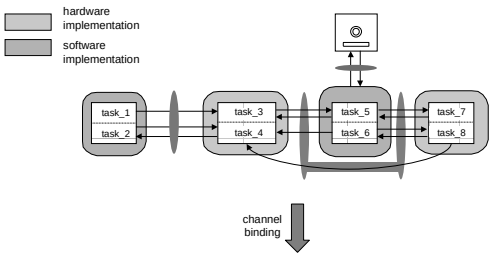


## Channel Binding (cont'd)

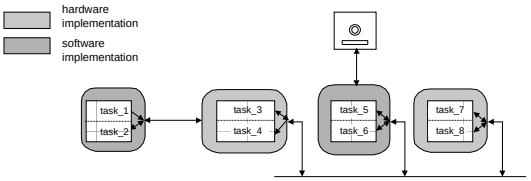
- The main criteria used for channel grouping is to avoid bus conflicts and to reduce the total number of connecting wires:
  - group together channels which don't access concurrently the bus,
  - group together channels which are accessed by the same processes,
  - depending on its features, a communication unit can support a certain number of channels to be multiplexed on it, without reducing the communication rate below a required minimum.



## Channel Binding (cont'd)



## Channel Binding (cont'd)






## Communication Refinement

- After *Channel binding* the interconnection topology of the system is known and it is determined which channels are bound to a given communication support.
- Communication is still quite abstract and has to be refined with several implementation details:
  - the *width of the communication lines* has to be determined, depending on constraints concerning data transfer rates, number of available pins, cost,
  - if communication buses are shared, an adequate *control strategy* has to be decided,
  - a *communication protocol* has to be defined for each communication link.



## Interface Generation

- The interfaces needed for a correct functionality of the system can be generated; both software and hardware components have to be generated:
  - *access routines* inside the communicating tasks (expanded as executable code in software or as hardware specification in hardware components),
  - *controllers* (buffers, FIFO queues, arbitration logic) implement correct access to communication support,

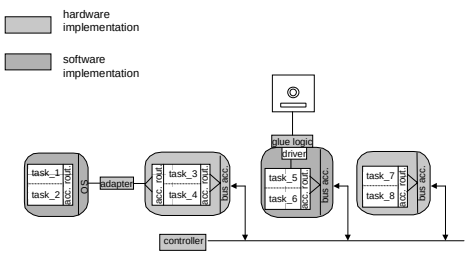


## Interface Generation (cont'd)

- *adapters* needed to interface components which use incompatible protocols,
- *device drivers* to support access to peripheral devices and application specific interfaces,
- *low level support for communication related tasks* (interrupt control, DMA, memory mapped I/O).



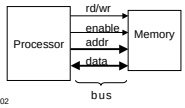
## Interface Generation






## Buses and Protocols

- A bus consists of wires connecting two or more processors or memories.
- Each wire in a bus may be *uni-directional* (e.g. rd/wr, enable or addr), or *bi-directional* (e.g., data).
- A bus has an associated *protocol* describing the rules for transferring data over the wires.




G. Vahid and T. Givargis, "*Embedded System Design: A Unified Hardware/Software Approach*", Draft of the book, 1999.

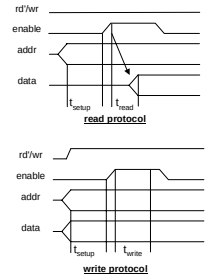
## Buses and Protocols

- Bus protocols usually described using *timing diagrams*.
- For a purpose of automatic synthesis other methods are also proposed:
  - HDL descriptions,
  - grammar based descriptions.
- Verification of protocols:
  - timing properties,
  - deadlock,
  - ...



## Timing Diagrams

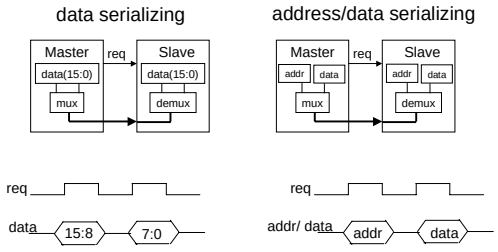
- Most common hardware protocol
- Bus cycle
  - Possible subprotocol
    - e.g., read protocol or write protocol
  - Complete transfer
  - May be several clock cycles
- Active high vs. active low
  - "Assert" means active
- Read protocol example:
  - *rd'/wr* set low
  - address placed on *addr* by processor for at least $t_{setup}$ time before *enable* set high
  - high *enable* triggers memory to place data on *data* wires by time $t_{read}$






*Embedded Systems Design: A Unified Hardware/Software Introduction,* (c) 2000 Vahid/Givargis

## Protocol Basics — Time-multiplexing



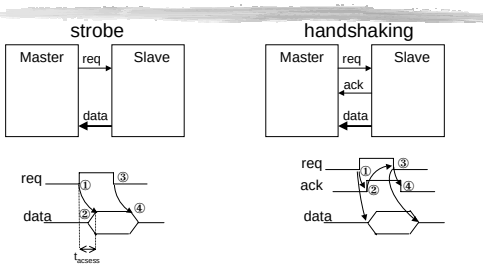



## Protocol Control Methods

- *Strobe* protocol — the master uses one control line, often called the *request* line, to initiate the data transfer, and the transfer is considered to be complete after some fixed time interval after the initiation.
- *Handshake* protocol — master uses a *request* line to initiate the transfer, and the slave uses an *acknowledge* line to inform the master when the data is ready.



## Protocol Control Methods



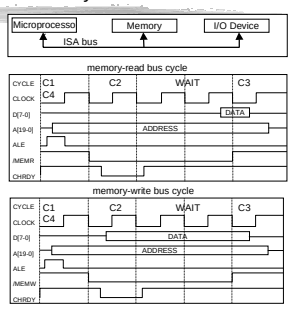

1. Master asserts *req* to receive data
2. Slave puts data on bus **within time $t_{access}$**
3. Master receives data and deasserts *req*
4. Slave ready for next request

1. Master asserts *req* to receive data
2. Slave puts data on bus **and asserts *ack***
3. Master receives data and deasserts *req*
4. Slave ready for next request



## ISA bus protocol – memory access

- ISA: Industry Standard Architecture
  - Common in 80x86's
- Features
  - 20-bit address
  - Compromise strobe/handshake control
    - 4 cycles default
    - Unless CHRDY deasserted – resulting in additional wait cycles (up to 6)






*Embedded Systems Design: A Unified Hardware/Software Introduction,* (c) 2000 Vahid/Givargis

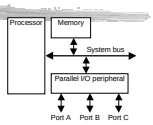
## Microprocessor interfacing: I/O addressing

- A microprocessor communicates with other devices using some of its pins
  - Port-based I/O (parallel I/O)
    - Processor has one or more N-bit ports
    - Processor's software reads and writes a port just like a register
    - E.g., P0 = 0xFF; v = P1.2; -- P0 and P1 are 8-bit ports
  - Bus-based I/O
    - Processor has address, data and control ports that form a single bus
    - Communication protocol is built into the processor
    - A single instruction carries out the read or write protocol on the bus

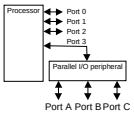


## Compromises/extensions

- Parallel I/O peripheral
  - When processor only supports bus-based I/O but parallel I/O needed
  - Each port on peripheral connected to a register within peripheral that is read/written by the processor
- Extended parallel I/O
  - When processor supports port-based I/O but more ports needed
  - One or more processor ports interface with parallel I/O peripheral extending total number of ports available for I/O
  - e.g., extending 4 ports to 6 ports in figure



Adding parallel I/O to a bus-based I/O processor



Extended parallel I/O

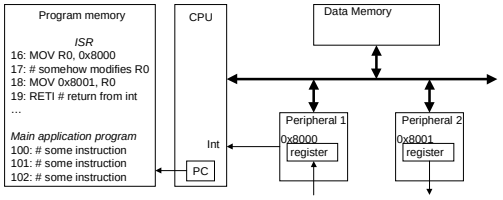


## Interrupts

- The communication with a device which produces data asynchronously requires a new communication paradigm.
- *Polling* — repeatedly check whether data is available at the port (waste of time).
- *Interrupts* — the microprocessor checks for the presence of a particular condition (similar to events in some languages).
- If interrupt then a *Interrupt Service Routin*e (ISR) is called automatically.



## Interrupt Service Routine




1. Microprocessor is executing its program, say current program counter (PC)=100.
2. Peripheral 1's input device stores new data in register at 0x8000.
3. Peripheral 1 asserts Int to request servicing by microprocessor of the new data.
4. Microprocessor stops executing its program and stores state, including PC.
5. Microprocessor jumps to fixed location of ISR at program memory location 16 (sets PC=16)
6. ISR reads data from 0x8000, modifies the data, and writes to 0x8001.
7. ISR returns, causing restoration of the microprocessor state, thus setting PC= 100+1 = 101
8. Microprocessor resumes executing its program.



## Interrupts (cont'd)

- *Fixed* address for ISR.
- *Vectored interrupt* make it possible to determine the address at which the ISR resides
  - peripheral asserts *IntReq*,
  - processor acknowledges *IntAc*k that the interrupt has been detected,
  - the peripheral provides the address on the data bus, and the microprocessor jumps to the ISR.
- *Interrupt address table* — the peripheral provides *int* and the interrupt number and the processor decodes it using the table.



## Additional interrupt issues

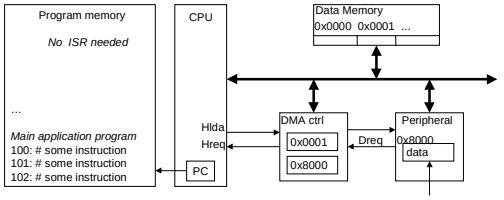
- Maskable vs. non-maskable interrupts
  - Maskable: programmer can set bit that causes processor to ignore interrupt
    - Important when in the middle of time-critical code
  - Non-maskable: a separate interrupt pin that can't be masked
    - Typically reserved for drastic situations, like power failure requiring immediate backup of data to non-volatile memory
- Jump to ISR
  - Some microprocessors treat jump same as call of any subroutine
    - Complete state saved (PC, registers)
  - Others only save partial state, like PC only
    - Thus, ISR must not modify registers, or else must save them first
    - Assembly-language programmer must be aware of which registers stored

## Direct Memory Access

- DMA controller — single-purpose processor which transfers data between memories and peripherals "outside" the processor.
- Steels memory access cycles from processor while needed.
- Does not require interventions from processor and overhead of interrupted program.
- DMA usually transfers block of data.



## Direct Memory Access




1. Microprocessor is executing its program, say current program counter (PC)=100.
2. Peripheral's input device stores new data in register at 0x8000.
3. Peripheral asserts *Dreq* to request servicing of the new data.
4. DMA controller asserts *Hreq* to request control of the system bus.
5. Microprocessor relinquishes system bus, perhaps stopping after executing statement 100.
6. **DMA controller reads data from 0x8000 and writes that data to 0x0001 in data memory.**
7. DMA controller deasserts Hreq and completes handshake with peripheral.
8. Microprocessor resumes executing its program, perhaps starting with statement 101.

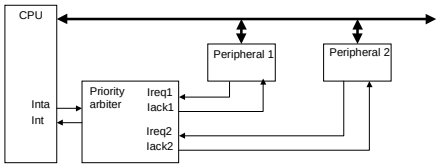


## Bus Arbitration

- Multiple peripherals might request service from a single resource, for example a bus.
- Priority arbiter — decides to whom the resource is granted.
- Arbitration method:
  - *fixed priorities* between peripherals,
  - *rotating priorities* (round-robin) — the arbiter changes priority of peripherals based on the history of servicing of those peripherals.



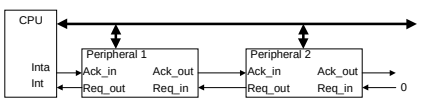
## Bus Arbitration




1. Microprocessor is executing its program.
2. Peripheral1 needs servicing so asserts *Ireq*1. Peripheral2 also needs servicing so asserts *Ireq*2.
3. Priority arbiter sees at least one *Ireq* input asserted, so asserts *Int*.
4. Microprocessor stops executing its program and stores its state.
5. **Microprocessor asserts *Inta*.**
6. **Priority arbiter asserts *Iack1* to acknowledge Peripheral1.**
7. **Peripheral1 puts its interrupt address vector on the system bus**
8. **Microprocessor jumps to the address of ISR read from data bus, ISR executes and returns (and completes handshake with arbiter).**
9. Microprocessor resumes executing its program.



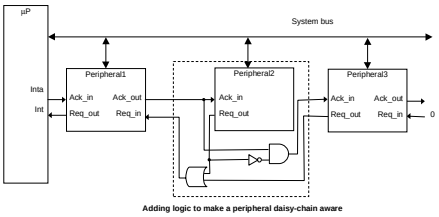
## Daisy-chain arbitration




- A peripheral asserts its request output if it requires servicing, OR if its request input is asserted.
- The microprocessor gets only one request.
- When Ack_in arrives the peripheral which requested the service proceeds to put its interrupt vector address on the system bus; if it did not requested the service, it passes the Ack_out upstream to the next peripheral.



## Daisy-chain arbitration logic



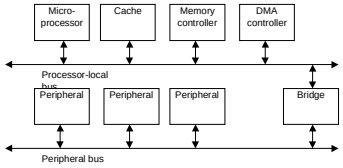

**Adding logic to make a peripheral daisy-chain aware**

## Multilevel bus architectures

- If single bus used for all communication
  - Peripherals required to have high-speed, processor-specific bus interface
    - excess gates, power consumption, cost
    - less portable
  - Too many peripherals slows down bus
- Processor-local bus
  - High speed, wide, most frequent communication
  - Connects microprocessor, cache, memory controllers, etc.
- Peripheral bus
  - Lower speed, narrower, less frequent communication
  - Typically industry standard bus (ISA, PCI) for portability
- Bridge
  - Single-purpose processor converts communication between busses



## A two-level bus architecture






## Protocol examples

- $I^2C$ (Inter-IC) - two-wire serial bus protocol developed by Philips Semiconductors nearly 20 years ago
  - Data transfer rates up to 100 kbits/s and 7-bit addressing possible in normal mode
  - 3.4 Mbits/s and 10-bit addressing in fast-mode
- CAN (Controller area network)
  - Protocol for real-time applications
  - Developed by Robert Bosch GmbH
  - Originally for communication among components of cars
  - Data transfer rates up to 1 Mbit/s and 11-bit addressing



## Protocol examples (cont'd)

- FireWire (a.k.a. I-Link, Lynx, IEEE 1394)
  - High-performance serial bus developed by Apple Computer Inc.
  - Designed for interfacing independent electronic components (e.g., desktop, scanner, cameras)
  - Data transfer rates from 12.5 to 400 Mbits/s, 64-bit addressing
- USB (Universal Serial Bus)
  - Easier connection between PC and monitors, printers, digital speakers, modems, scanners, digital cameras, joysticks, multimedia game equipment
  - 2 data rates:
    - 12 Mbps for increased bandwidth devices
    - 1.5 Mbps for lower-speed devices (joysticks, game pads)
- PCI Bus (Peripheral Component Interconnect) - high performance bus originated at Intel in the early 1990's
  - Data transfer rates of 127.2 to 508.6 Mbits/s and 32-bit addressing
    - Later extended to 64-bit while maintaining compatibility with 32-bit schemes



## Protocol examples (cont'd)

- ARM Bus
  - Designed and used internally by ARM Corporation
  - Interfaces with ARM line of processors
  - Data transfer rate is a function of clock speed
    - If clock speed of bus is X, transfer rate = 16 x X bits/s
- IrDA
  - Protocol suite that supports short-range point-to-point infrared data transmission
  - Created and promoted by the Infrared Data Association (IrDA)
  - Data transfer rate of 9.6 kbps and 4 Mbps



## Protocol examples (cont'd)

- Bluetooth
  - New, global standard for wireless connectivity
  - Based on low-cost, short-range radio link
- IEEE 802.11
  - Proposed standard for wireless LANs
  - provisions for data transfer rates of 1 or 2 Mbps

## Summary

- Interface design is a challenging design step which requires a deep knowledge on underlying hardware communication devices.
- On the top level we distinguish
  - Channel binding,
  - Communication refinement,
  - Interface generation.
- The deep knowledge hardware properties, such as communication protocols, interrupts, DMA, arbiters is, however, needed to make these steps efficient.



## Literature


- Course book - Chapters 3.2, (3.3), 4.2, 4.3, 4.5
- P. Eles, K. Kuchcinski and Z. Peng, *System Synthesis with VHDL*, Kluwer Academic Publisher, 1998.
- F. Vahid and T. Givargis, *Embedded System Design: A Unified Hardware Software Approach*, John Wiley & Sons; ISBN: 0471386782. Copyright (c) 2002.