Homework

- ☐ It's back by popular demand!
- ☐ Readings, exercises: Chapter 11, due 10/10 (later than usual!)
- ☐ Homework keys
- ☐ Study group(s)
- ☐ Prelim 1 – 10/12

COS 140: Foundations of Computer Science

Bus Arbitration: Daisy Chain Buses

Fall 2018

The problem

- Need to get information from one component to another
- Components can be:
  - functional units inside the CPU
  - CPU and memory
  - CPU/memory and I/O devices
- Maybe too expensive/impractical to have point-to-point communication
- How to have devices share communication pathways?

Use a communication *bus*

What is a Bus?

- The method of communication between components of the architecture.
- Multiple lines.
  - Each line sends a 1 or 0.
  - Lines are grouped together and bits are sent in parallel.
- Multiple devices can send and receive.
  - No privacy.
  - If all devices send, have a garbled message.

Communication with Memory

- words of data read or written to memory
- address of word
- control to specify read or write

Communication with the I/O Module

- ☐ data
- ☐ address
- ☐ read or write control
- ☐ select for a particular I/O device
- ☐ interrupt CPU

Communication with the CPU

- ☐ read instructions
- ☐ read and write data
- ☐ control signals to make system work
- ☐ handle interrupts

Interrupts

- Interrupts alert CPU to something important that has happened ⇒ CPU does not have to constantly check for each potential situation.
- Devices put a 1 on an interrupt line to signal interrupt
- CPU checks for interrupts at specific points in its functioning.
- If it finds an interrupt, it suspends what it is doing (e.g., the user's program) to handle the interrupt.
- When the interrupt has been handled, the CPU returns to what it was doing before the interrupt was discovered.

Some Bus Terminology

System bus: connects memory, CPU and I/O
Bus width: number of lines
Assert: put information on a line
Negate: remove information from a line

Bus Arbitration

- No physical reason why all devices on the bus can't use the bus at the same time
- If more than one device tries to send information on the bus at once: information garbled
- *Bus arbitration:* controls which device will get the bus.
- Many schemes for bus arbitration – for each
  - all devices have to follow the rules
  - do not send information unless they have permission

Centralized Arbitration

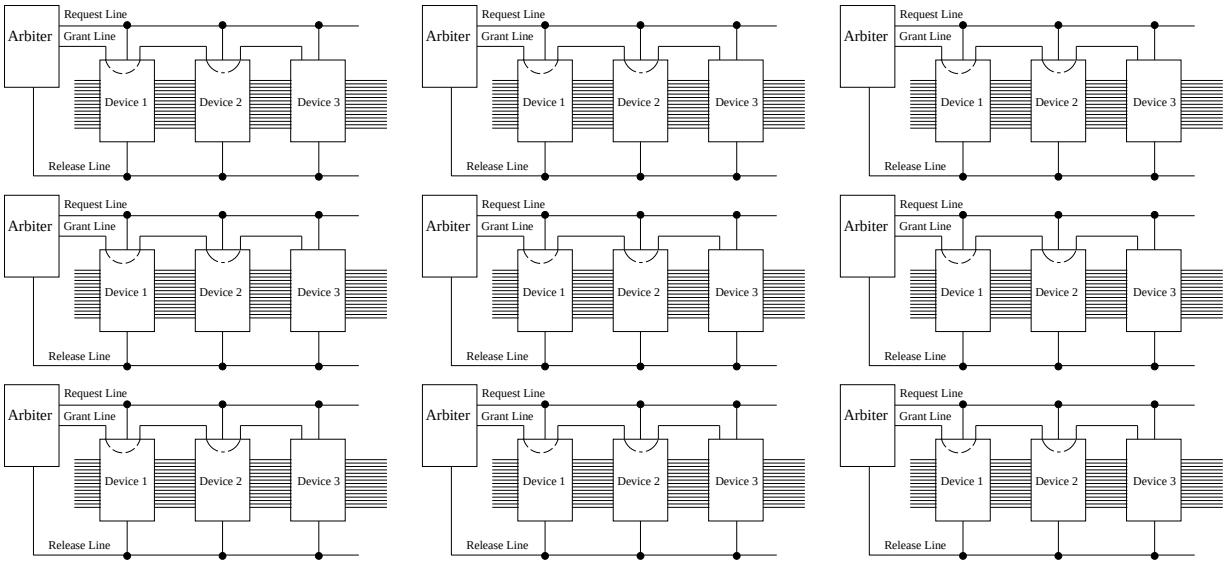
- One arbiter makes choice and informs devices on bus.
- Advantage:
  - Can use very simple scheme to select which device will have control.
  - Devices do not have to put any resources toward arbitration.
- Disadvantages:
  - Cost of arbiter.
  - *Single point of failure* at arbiter. If arbiter goes down, can't use the bus.

Decentralized Arbitration

- All devices participate in selecting which device will control the bus next.
- Advantages:
  - Less costly because do not have to have an arbiter.
  - If a device goes down, can continue to use bus.
- Disadvantage:
  - Schemes are more complicated than those for centralized arbitration.

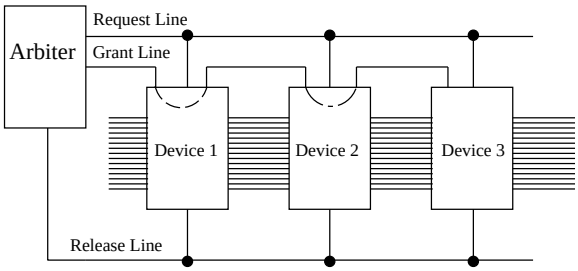
Daisy Chains

Daisy Chaining Architecture



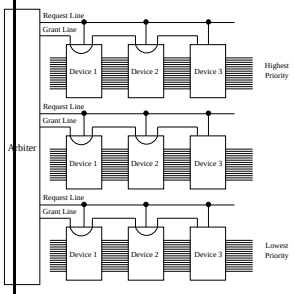
The bus has additional control lines: grant, request, and release When a device wants the bus it asserts its request line The arbiter senses the request, then it asserts the grant line If a node (a *bus master*) receives the grant line, but hasn't requested the bus, it passes it through When the node requesting the bus gets the grant line, it can start using the bus When the node is done with the bus, it releases the request line It then tells the arbiter it's done by raising the release line The arbiter then drops the grant line At this point, it's back to the initial state Now another device can request the bus

Daisy Chaining Issues



- How do devices know that the bus is busy and they can't take control?
- Which device most often gets control of the bus?
  - Do you think that's fair?
  - Can one device suffer from *starvation* – never getting any resources?

Adding Priorities to Daisy Chaining

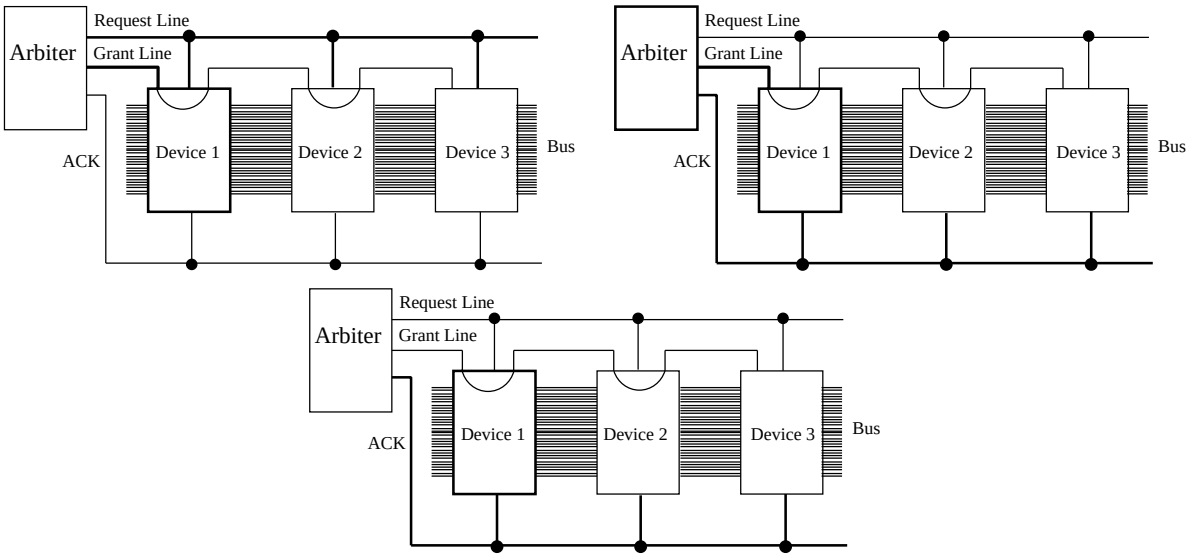


- Can add request and grant lines to create priorities.
  - Devices are assigned priorities and make requests and seek grants of the bus from the lines for their priorities.
  - Can have several priority levels.
- The arbiter has to assert the grant line corresponding to the highest priority request line that is asserted.
- For each priority level, arbitration works the same way for the device.

Hidden Arbitration

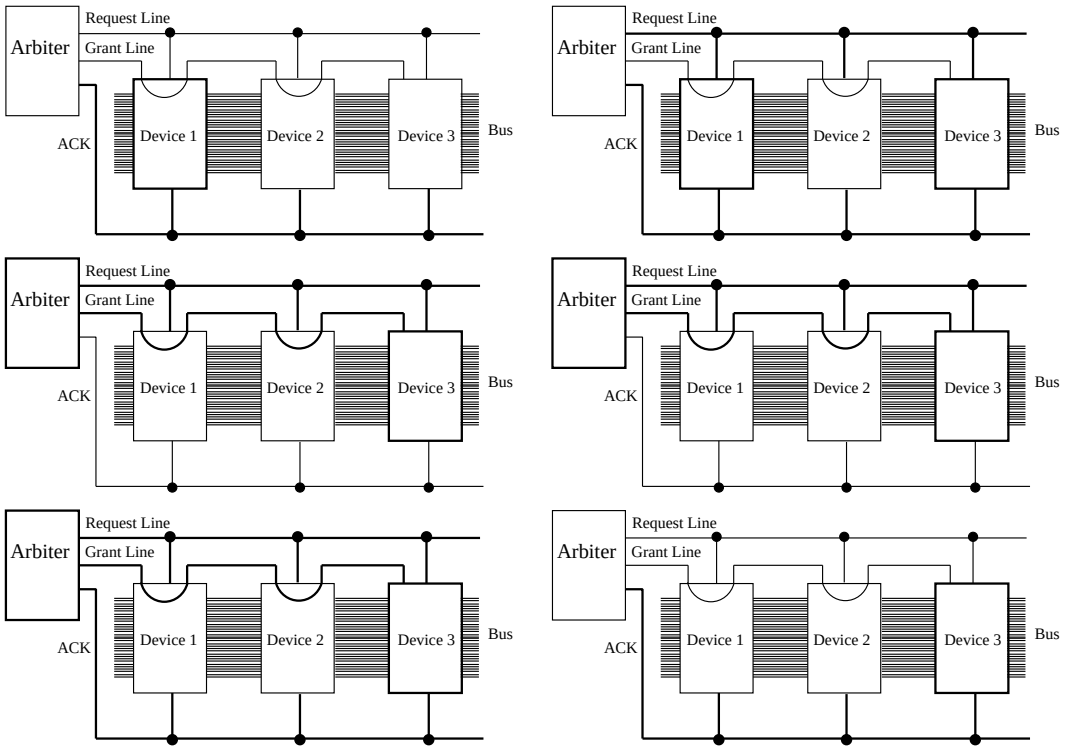
- In current scheme, arbitration occurs when the bus is available.
  - Data is not sent when arbitration is going on.
  - Less data sent over the bus – waste time on arbitration.
- *Hidden arbitration* means that arbitration takes place while the bus is being used, so arbitration is hidden from the bus usage.

Hidden Arbitration in Daisy Chaining



- The device that controls the bus is chosen in the same way as with regular daisy chaining.
- When a device takes control of the bus, it asserts an acknowledgment (ACK) line, which is noticed by arbiter
- The arbiter then negates the grant line

Hidden Arbitration in Daisy Chaining



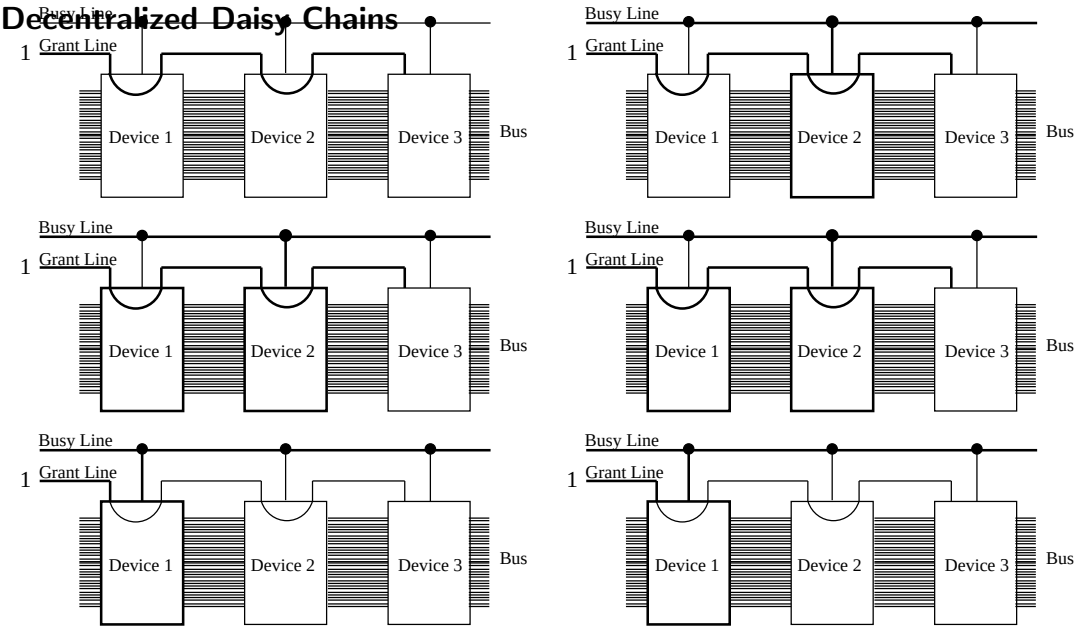
- ☐ Once grant line is negated...
- ☐ Others may request bus...and are selected in the usual way
- ☐ When the first device is finished, it negates ACK line
- ☐ Next device starts using bus as soon as ACK negated

Decentralized Daisy Chains 19 / 21

Decentralized Daisy Chain

- ☐ The arbiter is replaced with an asserted grant line.
- ☐ Add a busy line to show when bus is busy and cannot change grant line.
- ☐ Take control of bus as with centralized daisy chain (with addition of busy line).

Decentralized Daisy Chains



1. Initially, grant line passed throughout chain
2. When device wants bus and busy negated, negate grant line, assert busy
3. If another wants bus, senses busy, waits
4. When finished, negate busy, pass asserted grant
5. Device wanting bus now notices busy negated, asserts it and negates grant downstream
6. Device begins using bus

So...

- □ Which is best?
- □ Pros, cons?
- □ Real buses can be *much* more complicated than this!
- □ Many different kinds