

### Administrative Stuff

- Homework 3 due Thursday 10/17
- Catchup lab
  - Lab due dates have been push back by one lab session
- Quiz 1 Today



### Interfacing

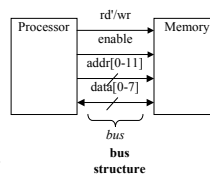
### Introduction

- Embedded system’s functionality aspects
  - Processing
    - transformation of data
    - processors
  - Storage
    - retention of data
    - memory
  - Communication
    - transfer of data between processors and memories
    - buses
- Interfacing

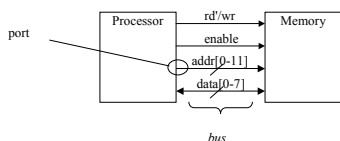


### A simple bus

- Wires:
  - Uni-directional or bi-directional
- Bus
  - Set of wires with a single function
    - Address bus, data bus
  - Or, entire collection of wires
    - Address, data and control
    - Associated protocol: rules for communication



### Ports

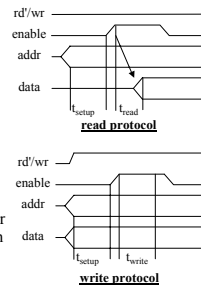


- Conducting device on periphery
- Connects bus to processor or memory
- Often referred to as a pin
  - Because of actual pins on periphery of IC package
    - that plug into socket on printed-circuit board
  - Today, use metallic balls instead of pins
  - Or, metal “pads” connecting processors and memories within single IC
- Single wire or set of wires with single function
  - 12-wire address port



### Timing Diagrams

- Most common hardware protocol
- Bus cycle
  - A portion of protocol, or 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  $t_{read}$



### Basic protocol concepts

- Actors: master initiates, servant (slave) responds
  - Independent of direction: sender, receiver
- Addresses:
  - Specifies a location in memory, a peripheral, or a register within a peripheral
- Time multiplexing
  - Share a single set of wires for multiple pieces of information
  - Saves wires

Time-multiplexed data transfer

data serializing

address/data muxing

7

CS122A: Embedded System Design, Fall 02

### Basic protocol concepts: control methods

strobe

- Master asserts req to receive data
- Servant puts data on bus **within time**  $t_{access}$
- Master receives data and deasserts req
- Servant ready for next request

handshake

- Master asserts req to receive data
- Servant puts data on bus **and asserts ack**
- Master receives data and deasserts req
- Servant ready for next request

8

CS122A: Embedded System Design, Fall 02

### Microprocessor interfacing: I/O addressing

- A microprocessor communicates with other devices
  - 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
    - A single instruction carries out the read or write protocol on the bus

9

CS122A: Embedded System Design, Fall 02

### Types of bus-based I/O: memory-mapped I/O and standard I/O

- Processor talks to memory and peripherals using same bus--
- Two ways to talk to peripherals
  - Memory-mapped I/O
    - Peripheral registers occupy addresses in same address space as memory
    - e.g., Bus has 16-bit address
    - lower 32K addresses may correspond to memory
    - upper 32k addresses may correspond to peripherals
  - Standard I/O (I/O-mapped I/O)
    - Additional pin (M/I/O) on bus indicates a memory or peripheral access
    - e.g., Bus has 16-bit address
    - all 64K addresses correspond to memory when M/I/O set to 0
    - all 64K addresses correspond to peripherals when M/I/O set to 1

10

CS122A: Embedded System Design, Fall 02

### Memory-mapped I/O vs. Standard I/O

- Advantage memory-mapped I/O:
  - Requires no special instructions
    - Assembly instructions involving memory like MOV and ADD work with peripherals as well
    - Standard I/O requires special instructions (e.g., IN, OUT) to move data between peripheral registers and memory
- Advantage standard I/O:
  - No loss of memory addresses to peripherals
  - Simpler address decoding logic in peripherals possible
    - when number of peripherals much smaller than address space
    - then high-order address bits can be ignored
    - smaller and/or faster comparators

11

CS122A: Embedded System Design, Fall 02

### Microprocessor interfacing: interrupts

- Suppose intermittent data -> peripheral -> processor
  - Processor can *poll* peripheral
    - wasteful
  - Peripheral can *interrupt* the processor
- Requires an extra pin or pins: Int
  - If Int is 1,
    - Processor suspends current program, jumps to an ISR
    - Known as interrupt-driven I/O
  - No extra cycle
    - "polling" is built-into the hardware

12

CS122A: Embedded System Design, Fall 02

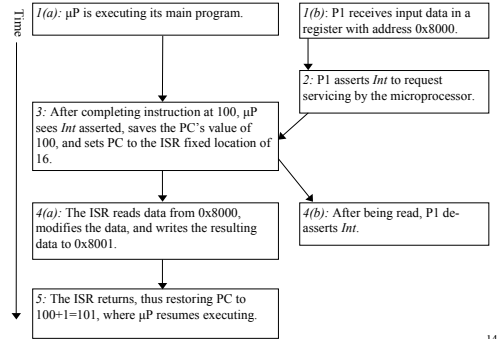
### Microprocessor interfacing: interrupts

- Address (interrupt address vector) of the ISR?
  - Fixed interrupt
    - Address built into microprocessor, cannot be changed
    - ISR stored at address
    - or a jump to actual ISR stored if not enough bytes available
  - Vectored interrupt
    - Peripheral must provide the address
    - Common when multiple peripherals connected by a system bus
  - Compromise: interrupt address table

13

CS122A: Embedded System Design, Fall 02

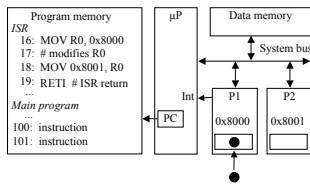
### Interrupt-driven I/O using fixed ISR location



14

CS122A: Embedded System Design, Fall 02

### Interrupt-driven I/O using fixed ISR location (cont')

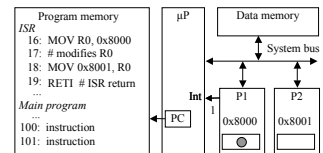


- 1(a): μP is executing its main program
- 1(b): P1 receives input data in a register with address 0x8000.

15

CS122A: Embedded System Design, Fall 02

### Interrupt-driven I/O using fixed ISR location (cont')

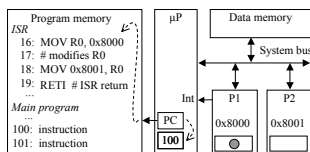


- 2: P1 asserts *Int* to request servicing by the microprocessor

16

CS122A: Embedded System Design, Fall 02

### Interrupt-driven I/O using fixed ISR location (cont')

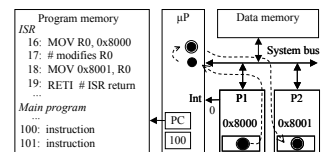


- 3: After completing instruction at 100, μP sees *Int* asserted, saves the PC's value of 100, and sets PC to the ISR fixed location of 16.

17

CS122A: Embedded System Design, Fall 02

### Interrupt-driven I/O using fixed ISR location (cont')

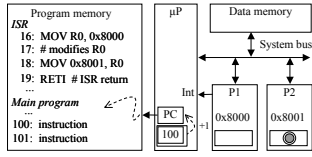


- 4(a): The ISR reads data from 0x8000, modifies the data, and writes the resulting data to 0x8001.
- 4(b): After being read, P1 deasserts *Int*.

18

CS122A: Embedded System Design, Fall 02

### Interrupt-driven I/O using fixed ISR location (cont')

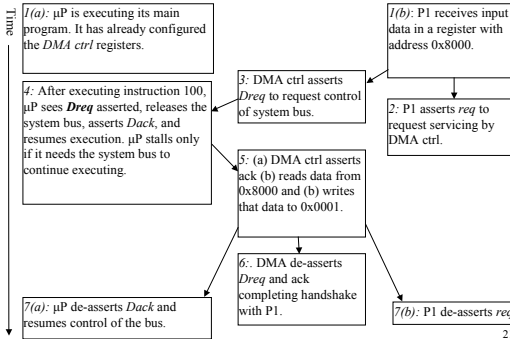


5: The ISR returns, thus restoring PC to 100+1=101, where μP resumes executing.

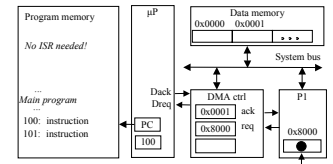
### Direct memory access

- Buffering
  - Temporarily storing data in memory before processing
  - Data from peripheral commonly buffered in memory
- Microprocessor could handle this with ISR, but
  - Storing and restoring microprocessor state inefficient
    - Regular program must wait
- DMA controller more efficient
  - Separate single-purpose processor
  - Microprocessor relinquishes control of system bus to DMA controller
  - Microprocessor can resume its regular program
    - no inefficient storing and restoring state
    - regular program need not wait unless it requires the system bus

### Peripheral to memory transfer with DMA



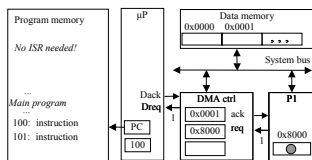
### Peripheral to memory transfer with DMA (cont')



1(a): μP is executing its main program. It has already configured the DMA ctrl registers

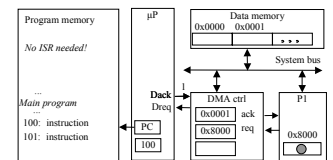
1(b): P1 receives input data in a register with address 0x8000.

### Peripheral to memory transfer with DMA (cont')



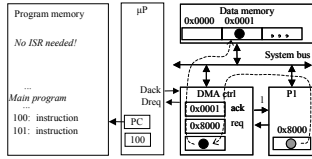
- 2: P1 asserts req to request servicing by DMA ctrl.
- 3: DMA ctrl asserts Dreq to request control of system bus

### Peripheral to memory transfer with DMA (cont')



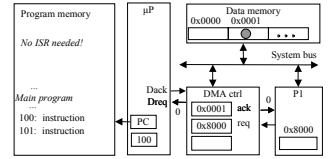
4: After executing instruction 100, μP sees Dreq asserted, releases the system bus, asserts Dack, and resumes execution, μP stalls only if it needs the system bus to continue executing.

### Peripheral to memory transfer with DMA (cont')



5: DMA ctrl (a) asserts ack, (b) reads data from 0x8000, and (c) writes that data to 0x0001.

### Peripheral to memory transfer with DMA (cont')



6: DMA de-asserts *Dreq* and *ack* completing the handshake with P1.

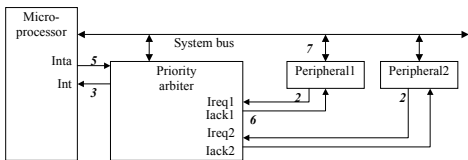
### Arbitration

- Arbitration:
  - Resolves competing requests from multiple peripherals
  - E.g. CD player and keyboard requesting services from CPU
- Many different methods
  - Priority arbiter
    - Fixed priority
    - Rotating priority
  - Daisy-chain arbiter

### Priority arbiter

- Single-purpose processor
- Connect to system bus for configuration
- Sit in between peripherals and resources
  - Peripherals make requests to arbiter, arbiter makes requests to resource
  - Resource send ack to arbiter, arbiter send ack to peripherals
- Fixed priority
  - Each peripheral has unique rank
  - Highest rank chosen first with simultaneous requests
  - Preferred when clear difference in rank between peripherals
- Rotating priority (E.g. round-robin)
  - Priority changed based on history of servicing
  - Better distribution of servicing
    - Especially among peripherals with similar priority demands

### Arbitration using a priority arbiter

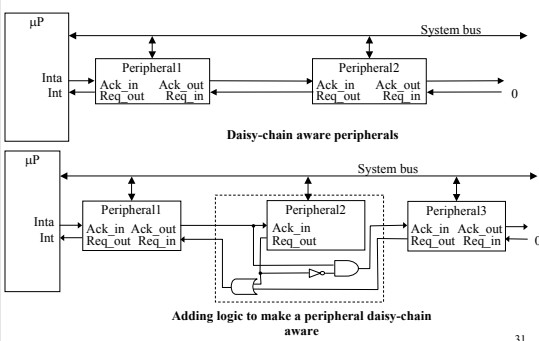


1. Microprocessor is executing its program.
2. Peripheral1 needs servicing so asserts *Ireq1*. Peripheral2 also asserts *Ireq2*.
3. Priority arbiter sees at least one *Ireq* input asserted, so asserts *Int*.
4. Microprocessor stops executing its program and stores its state.
5. Microprocessor asserts *Inta*.
6. Priority arbiter asserts *Iack1* to acknowledge Peripheral1.
7. Peripheral1 puts its interrupt address vector on the system bus
8. Microprocessor jumps to address of ISR, ISR executes and returns
9. Microprocessor resumes executing its program, but not for long...

### Arbitration: Daisy-chain arbitration

- Arbitration done by peripherals
  - Built into peripheral
  - External logic added for each peripheral
    - *req* and *ack*
- Peripherals connected to each other in daisy-chain
  - One peripheral connected to resource
  - All others connected "upstream"
  - Peripheral's *req* flows "downstream" to resource,
  - Resource's *ack* flows "upstream" to requesting peripheral
- Pros/cons
  - Easy to add/remove peripheral - no system redesign needed
  - Does not support rotating priority
  - One broken peripheral can cause loss of access to other peripherals
    - Especially the upstream peripherals

## Arbitration using a daisy-chain configuration



CS122A: Embedded System Design, Fall 02

31

## Advanced communication principles

- Layering
  - Break complexity of communication protocol into pieces
  - Lower levels provide services to higher level
    - Lower level might work with bits
    - higher level might work with packets of data
  - Physical layer
    - Lowest level in hierarchy
    - Medium to carry data from one actor (device or node) to another
- Parallel communication
  - Physical layer capable of transporting multiple bits of data
- Serial communication
  - Physical layer transports one bit of data at a time
- Wireless communication
  - No wire needed for transport at physical layer

CS122A: Embedded System Design, Fall 02

32

## Parallel communication

- Multiple data, control, and possibly power wires
  - One bit per wire
- High data throughput with short distances
- Good for devices on same IC or same circuit board
- Bus must be kept short
  - long parallel wires result in high capacitance values
    - requires more time to charge/discharge
  - Data misalignment between wires increases as length increases
- Higher cost, bulky

CS122A: Embedded System Design, Fall 02

33

## Serial communication

- Single data wire, possibly also control and power wires
- Words transmitted one bit at a time
- Higher data throughput with long distances
  - Less average capacitance, so more bits per unit of time
- Cheaper, less bulky
- More complex interfacing logic and communication protocol
  - Sender needs to decompose word into bits
  - Receiver needs to recombine bits into word
- Control signals often sent on same wire as data
  - Start-bit to “wake up” receiver
  - Transmit at predetermined speed
  - End-bit to signal end of transmission

CS122A: Embedded System Design, Fall 02

34

## Wireless communication

- Infrared (IR)
  - Electronic wave frequencies just below visible light spectrum
  - Diode emits infrared light to generate signal
  - Infrared transistor detects signal, conducts when exposed to infrared light
  - Cheap to build
  - Need line of sight, limited range
- Radio frequency (RF)
  - Electromagnetic wave frequencies in radio spectrum
  - Analog circuitry and antenna needed on both sides of transmission
  - Line of sight not needed, transmitter power determines range
  - Frequency hopping
    - Communicate while constantly changing frequency
    - Need common understanding of the sequence by transmitter and receiver
    - Allow many devices to share a fixed set of frequency

CS122A: Embedded System Design, Fall 02

35

## Frequency Hopping

- Interference occurs only at same frequency and time
- FH radio:
  - Hop between 2.4 and 2.483 GHz
- Hopping code
  - Determine the transmitter and receiver sequence
  - FCC regulation
    - 75 or more frequencies per band
    - Maximum dwell time of 400ms
- Retransmission at different frequency when interfered
- 2Mbps
- Low cost, low power, low interference, low data rates, high capacities for multiple users, and lower range (than direct sequencing)

CS122A: Embedded System Design, Fall 02

36

## Error detection and correction

- **Often part of bus protocol**
- **Error detection:**
  - ability of receiver to detect errors during transmission
- **Error correction:**
  - ability of receiver and transmitter to cooperate to correct problem
  - Typically done by acknowledgement/retransmission protocol
- **Bit error:** single bit is inverted
- **Burst of bit error:** consecutive bits received incorrectly
- **Parity:** extra bit sent with word used for error detection
  - Odd parity: data word plus parity bit contains odd number of 1's
  - Even parity: data word plus parity bit contains even number of 1's
  - Always detects single bit errors, but not all burst bit errors
- **Checksum:** extra word sent with data packet of multiple words
  - e.g., extra word contains XOR sum of all data words in packet

37

CS122A: Embedded System Design, Fall 02

## Examples for error detection

- Want to transmit 7 bit data 0010110
  - Odd parity: actually transmit 00101100
  - Even parity: actually transmit 00101101
  - Can detect single bit error like 0010010
  - Can't detect error if the 7 bit data got flip to 0110010
- Want to transmit 4 words:
  - 00101100, 11010001, 00101110, 11010001
  - Transmit a checksum: 00000000
  - Can detect packet error like
    - 00100100, 11010001, 00101110, 11010001
    - XOR is 00001010
  - Can't detect packet error like
    - 00100100, 11010001, 00100100, 11010001

38

CS122A: Embedded System Design, Fall 02

## Serial protocols: FireWire

- 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
  - Data transfer rates from 12.5 to 400 Mbits/s, 64-bit addressing
  - Plug-and-play capabilities
  - Capable of supporting a LAN similar to Ethernet
    - 64-bit address:
    - 10 bits for network ids, 1023 subnetworks
    - 6 bits for node ids, each subnetwork can have 63 nodes
    - 48 bits for memory address, each node can have 281 terabytes of distinct locations

39

CS122A: Embedded System Design, Fall 02

## Serial protocols: USB

- USB (Universal Serial Bus)
  - Easier connection between PC and:
    - monitors, printers, digital speakers, modems, digital cameras, joysticks
  - 2 data rates:
    - 12 Mbps for increased bandwidth devices
    - 1.5 Mbps for lower-speed devices (joysticks, game pads)
  - Tiered star topology can be used
    - One USB device (hub) connected to PC
    - hub can be embedded in devices like monitor or keyboard or can standalone
    - Multiple USB devices can be connected to hub
    - Up to 127 devices can be connected like this
  - USB host controller
    - Manages and controls bandwidth and driver software
    - Dynamically allocates power downstream

40

CS122A: Embedded System Design, Fall 02

## Wireless protocols: IrDA

- IrDA
  - “Beaming”
  - Protocol suite that supports short-range infrared data transmission
  - Created and promoted by the Infrared Data Association (IrDA)
  - Data transfer rate of 9.6 kbps and 4 Mbps
  - IrDA hardware deployed in notebook computers, printers, PDAs, digital cameras, public phones, cell phones
  - Lack of suitable (software) drivers has slowed use by applications
  - Windows 2000/98 now include support
  - Becoming available on popular embedded OS's

41

CS122A: Embedded System Design, Fall 02

## Wireless Protocols: IEEE 802.11

- IEEE 802.11
  - Proposed standard for wireless LANs
  - Ad-hoc or infrastructure network
  - Specifies parameters for PHY and MAC layers of network
  - PHY layer
    - physical layer
    - handles transmission of data between nodes
    - provisions for data transfer rates of 1 or 2 Mbps
    - operates in 2.4 to 2.4835 GHz frequency band (RF) or 300 to 428,000 GHz (IR)
  - MAC layer
    - medium access control layer
    - protocol responsible for maintaining order in shared medium
    - collision avoidance/detection
    - Random back-off factor
    - RTS and CTS

42

CS122A: Embedded System Design, Fall 02

## 802.11b

- Do for wireless networking what 802.3 do for Ethernet
- Wi-Fi (Wireless Fidelity)
  - Industrial group certification
- 11Mbps
- Operate at 2.4Ghz band
- First product: Apple AirPort (1999)
  
- Cheap!
  - Wireless router ~ \$200
  - Wireless access point ~ \$150
  - Wireless PC Cards ~\$75



43

CS122A: Embedded System Design, Fall 02

## 802.11a

- 5GHz Unlicensed National Information Infrastructure
  - 5.15GHz-5.35GHz and 5.725GHz-5.825GHz
  - Power rating (50mW, 250mW, 1W)
  - Unaffected by wireless phone, microwave, or bluetooth
    - Unlike 802.11b
  - But not internationally available (vs HiperLAN/2)
- Orthogonal Frequency division multiplexing
  - As oppose to spread spectrum (DS or FH)
  - Superior for indoor
- Support data rate up to 54Mbps
  - Actual rate will be about 35Mbps
- Not compatible with 802.11b
  - Unlike 802.11g



44

CS122A: Embedded System Design, Fall 02