Computer Organization and Architecture

Chapter 7

Input/Output

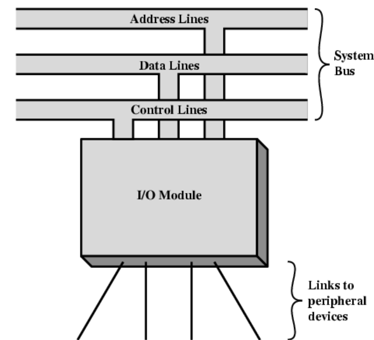
Input/Output Problems

- Computers have a wide variety of peripherals
  - Delivering different amounts of data, at different speeds, in different formats
- Many are not connected directly to system or expansion bus
- Most peripherals are slower than CPU and RAM; a few are faster
- Word length for peripherals may vary from the CPU
- Data format may vary (e.g., one word might include parity bits)

I/O Modules

- Peripheral communications are handled with I/O modules
- Two major functions:
  - Interface to processor or memory via bus or central link
  - Interface to one or more peripherals via tailored data links

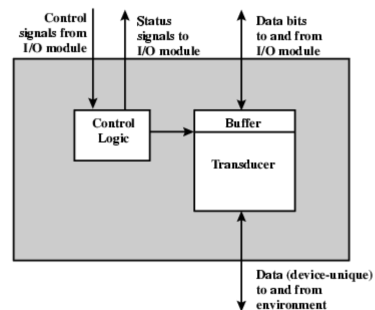
Generic Model of I/O Module



External Devices

- Human readable (human interface)
  - Screen, printer, keyboard, mouse
- Machine readable
  - Disks, tapes
    - Functional view of disks as part of memory hierarchy
    - Structurally part of I/O system
  - Sensors, actuators
  - Monitoring and control
- Communications
  - Modem
  - Network Interface Card (NIC)
  - Wireless Interface card

External Device Block Diagram



Keyboard and Monitor

- Most common HID (human interface device)
- In character based systems a monitor is acts as a “glass teletype”
- Basic unit of exchange is the character
- Most display adapters have text modes
  - characters are displayed by storing in adapter memory
  - Adapter contains hardware character generators that create bitmaps on the display

IRA (International Reference Alphabet)

- Usually referred to as ASCII
- 7-bit code
- Lower 31 chars plus last are control chars
- Developed before computers

ASCII Characters

- Text diagram is hard to read
- See assembler docs

bit position

			b_7	0	0	0	0	1	1	1	1
			b_6	0	0	1	1	0	0	1	1
			b_5	0	1	0	1	0	1	0	1
b_4	b_3	b_2	b_1								
0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	"	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	'	7	G	W	g	w
1	0	0	0	BS	CAN	(	8	H	X	h	x
1	0	0	1	HT	EM	)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[	k	{
1	1	0	0	FF	FS	,	<	L	\	l	|
1	1	0	1	CR	GS	-	=	M	]	m	}
1	1	1	0	SO	RS	.	>	N	^	n	~
1	1	1	1	SI	US	/	?	O	_	o	DEL

Some control characters

Format Control

BS (Backspace): Indicates movement of the printing mechanism or display cursor backward one position.
HT (Horizontal Tab): Indicates movement of the printing mechanism or display cursor forward to the next preassigned 'tab' or stopping position.
LF (Line Feed): Indicates movement of the printing mechanism or display cursor to the start of the next line.
VT (Vertical Tab): Indicates movement of the printing mechanism or display cursor to the next of a series preassigned printing lines.
FF (Form Feed): Indicates movement of the printing mechanism or display cursor to the starting position of the next page, form, or screen.
CR (Carriage Return): Indicates movement of the printing mechanism or display cursor to the starting position of the same line.

Transmission Control

SOH (Start of Heading): Used to indicate the start of a heading, which may contain address or routing information.
STX (Start of Text): Used to indicate the start of the text and so also indicates the end of the heading.
ETX (End of Text): Used to terminate the text that was started with STX.
EOT (End of Transmission): Indicates the end of a transmission, which may have included one or more 'texts' with their headings.
ENQ (Enquiry): A request for a response from a remote station. It may be used as a 'WHO ARE YOU' request for a station to identify itself.
ACK (Acknowledge): A character transmitted by a receiving device as an affirmation response to a sender. It is used as a positive response to polling messages.
NAK (Negative Acknowledgment): A character transmitted by a receiving device as an negative response to a sender. It is used as a negative response to polling messages.
SYN (Synchronous/Idle): Used by a synchronous transmission system to achieve synchronization. When no data is being sent a synchronous transmission system may send SYN characters continuously.
ETB (End of Transmission Block): Indicates the end of a block of data for communication purposes. It is used for blocking data where the block structure is not necessarily related to the processing format.

I/O Module Functions

- Major requirements or functions of an I/O module are
  - Control & Timing
  - CPU Communication
  - Device Communication
  - Data Buffering
  - Error Detection

Control and Timing

- Coordination of traffic between internal resources and external devices
- Example transaction:
  - Processor interrogates status of I/O module
  - Module returns device status
  - Device indicates ready to transmit; processor requests data transfer by means of a command to the module
  - I/O module obtains a byte of data from the device
  - Data are transferred to the processor
    - Typically requires one or more bus arbitrations

Processor Communication

- Processor communication involves:
  - Command decoding
    - Commands sent as signals on control bus with parameters on data bus
      - E.g. disk: Read Sector, Write Sector, Seek, …
  - Data exchange with processor
  - Status reporting
    - Peripherals are very slow compared to processor
    - May take some time after a READ command before data is ready
    - Typical signals: BUSY, READY
  - Address decoding
    - Module recognizes unique address for each device it controls

Device Communication

- On the other side the I/O module has to communicate with the device
  - Commands
  - Status information
  - Data
- Buffering is often essential
- Handles the speed mismatch between memory and the device
  - Low speed devices need to have data from memory buffered
  - High speed devices need to have data going to memory buffered
  - With any interrupt-driven device, data may be buffered pending interrupt handler servicing

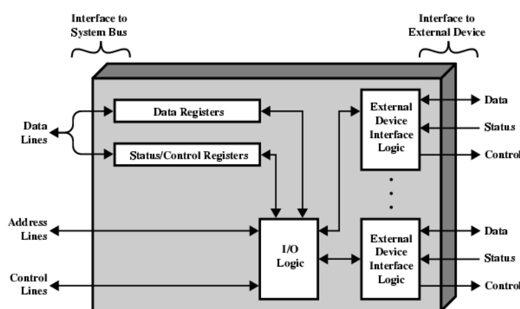
Error Detection and Reporting

- Mechanical and Electrical malfunction
  - Ex. Out of paper, paper jam, bad disk sector
- Data communication errors
  - Typically detected with parity bits

Typical I/O Control Steps

- Communication goes across the bus
  - CPU checks I/O module device status
  - I/O module returns status
  - If ready, CPU requests data transfer
  - I/O module gets data from device
  - I/O module transfers data to CPU
  - Variations for output, DMA, etc.

Typical I/O Module Structure



I/O Module Decisions

- Hide or reveal device properties to CPU
  - Ex. Disks: LBA (logical block addressing) physical address (CHS) is hidden from CPU
  - But older disks expose CHS addressing
- Support multiple or single device
  - Most disk controllers handle 2 devices
- Control device functions or leave for CPU
  - Ex: Video adapters with Direct Draw interface
  - But tape drives expose direct control to cpu
- Also O/S decisions
  - e.g. Unix treats everything it can as a file

Terminology

- Device or I/O Controller
  - Relatively simple, detailed control left to CPU
- I/O Processor or I/O Channel
  - Presents high-level interface to CPU
  - Often controls multiple devices
  - Has processing capability

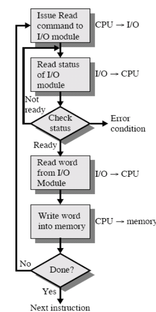
Input Output Techniques

- Programmed
  - CPU controls the entire process
  - Can waste CPU time
- Interrupt driven
  - Processor issues command
  - Device proceeds and leaves processor free
- Direct Memory Access (DMA)
  - Device exchanges data directly with memory

Programmed I/O

- CPU has direct control over I/O
  - Sensing status
  - Read/write commands
  - Transferring data
- CPU waits for I/O module to complete operation
- Wastes CPU time

Programmed I/O flowchart



Programmed I/O - detail

- CPU requests I/O operation
- I/O module performs operation
- I/O module sets status bits
- CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU
- CPU may wait or come back later

Types of I/O Commands

- CPU issues address
  - Identifies module (& device if >1 per module)
- CPU issues command
  - Control - telling module what to do
    - e.g. spin up disk
  - Test - check status
    - e.g. power? Error?
  - Read/Write
    - Module transfers data via buffer from/to device

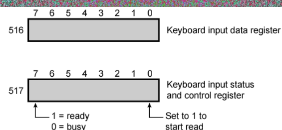
Addressing I/O Devices

- Under programmed I/O data transfer is very much like memory access (CPU viewpoint)
- Each device given unique identifier
- CPU commands contain identifier (address)

I/O Mapping

- Memory mapped I/O
  - Devices and memory share an address space
  - I/O looks just like memory read/write
  - No special commands for I/O
    - Large selection of memory access commands available
  - Ex: Motorola 68000 family
- Isolated I/O
  - Separate address spaces
  - Need I/O or memory select lines
  - Special commands for I/O
    - Limited set of commands
    - Ex: Intel 80x86 family has IN and OUT commands

Memory Mapped and Isolated I/O



ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load AC	"1"	Load accumulator
	Store AC	517	Initiate keyboard read
202	Load AC	517	Get status byte
	Branch if Sign = 0	202	Loop until ready
	Load AC	516	Load data byte

(a) Memory-mapped I/O

ADDRESS	INSTRUCTION	OPERAND	COMMENT
200	Load I/O	5	Initiate keyboard rea
201	Test I/O	5	Check for completion
	Branch Not Ready	201	Loop until complete
	In	5	Load data byte

(b) Isolated I/O

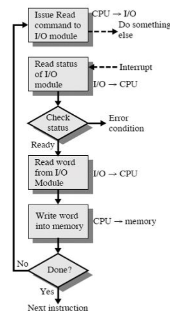
Interrupt Driven I/O

- Overcomes CPU waiting
- Avoids repeated checking of device by CPU (polling)
- I/O module interrupts when ready

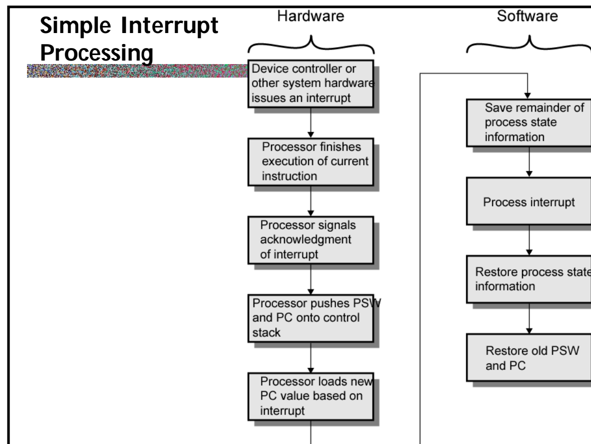
Interrupt Driven I/O Basic Operation

- CPU issues read command
- I/O module gets data from peripheral while CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

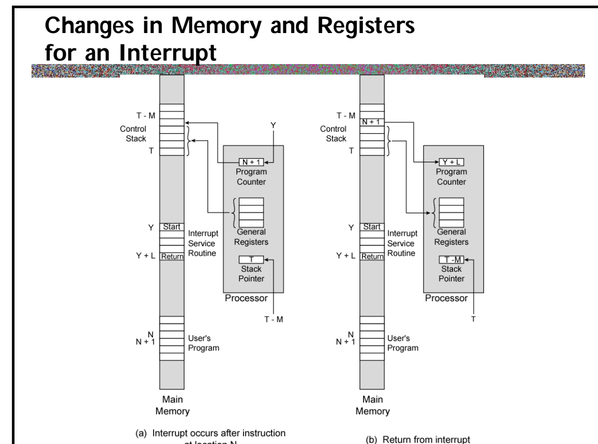
Interrupt-Driven I/O Flowchart



Simple Interrupt Processing



Changes in Memory and Registers for an Interrupt



Design Issues

- How do you identify the module issuing the interrupt?
- How do you deal with multiple interrupts?
  - i.e. an interrupt handler being interrupted

Identifying Interrupting Module (1)

- Different line for each module
  - Don't want to devote a lot of bus or cpu pins to interrupt lines
  - Limits number of devices
  - But lines can be shared between devices, and these will use one of the following techniques
- Software poll
  - CPU asks each module in turn, or checks status register in each module
  - Slow

Identifying Interrupting Module (2)

- Daisy Chain or Hardware poll
  - Interrupt Acknowledge sent down a chain
  - Module responsible places a word of data (the vector) on bus
  - CPU uses vector to identify handler routine
- Bus Arbitration
  - Module must claim the bus before it can raise interrupt
  - e.g. PCI & SCSI
  - Processor responds with Interrupt Acknowledge
  - Module can then place vector on bus

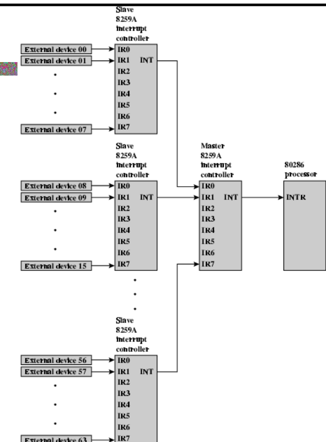
Multiple Interrupts

- Each interrupt line has a priority
- Higher priority lines can interrupt lower priority lines
- If bus mastering only current master can interrupt

Example - PC Bus

- 80x86 CPU has one interrupt line INTR and one interrupt acknowledge INTA
- 8086 based systems used one 8259A programmable interrupt controller
  - Each 8259A has 8 interrupt lines
- Current x86 processors typically use 2 8259A's (master and slave)
  - This provides 15 IRQs because the slave is attached to one of the master's IRQ pins
- Up to 8 controllers can be linked to a master controller

82C59A Interrupt Controller



Sequence of Events

- 8259A accepts interrupts
- 8259A determines priority
- 8259A signals 80x86 (raises INTR line)
- CPU Acknowledges when ready (interrupts can be disabled)
- 8259A puts correct vector (ISR address) on data bus
- CPU processes interrupt

ISA Bus Interrupt System

- ISA bus chains two 8259As together
- Link is via interrupt 2
- Gives 15 lines
  - 16 lines less one for link
- IRQ 9 is used to re-route anything trying to use IRQ 2
  - Backwards compatibility
- Incorporated in chip set – you will not see a chip labeled "8259A" on a motherboard

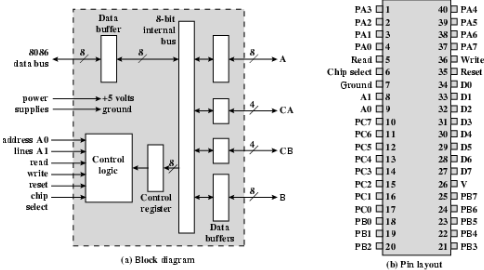
PIC

- This chip is called a "programmable interrupt controller" because it can be set up by the OS to use different operating modes
  - Fully nested: IRQs are prioritized (settable by OS)
  - Rotating: round-robin of equal priority interrupts
  - Masks can inhibit or enable interrupts
  - Interrupts can be vectored to a different INT from the IRQ

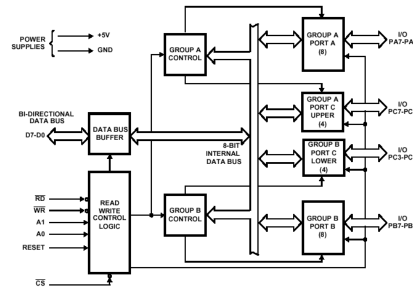
Intel 82C55A PPI

- Intel 82C55A Programmable Peripheral Interface is a simple I/O controller used to handle the keyboard and the internal speaker
- The PPI has 24 I/O lines designated as 8-bit registers or ports A, B, and C
  - C is nibble-addressable and can be used to control A and B
  - Another 8 bit register is the control register
- There are 8 bidirectional data lines
- Two address lines are used to select A,B,C or Control
- A transfer takes place when Chip Select is enabled along with Read or Write line

Intel 82C55A Programmable Peripheral Interface



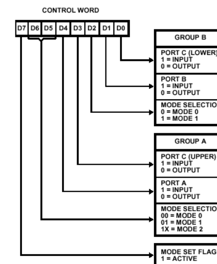
Another view



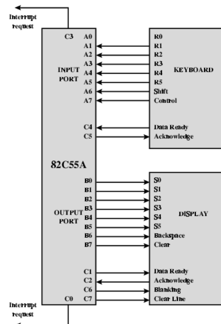
Control Register

- Controls mode of operation and defines signals
  - In mode 0 lines function as 3 I/O ports. Each port can be designated as input or output
  - In other modes A and B are I/O ports, and the lines of C serve as control signals
- Two types of control signals: handshaking and Interrupt request
  - Handshaking is a simple sync mechanism; one control line is used as DATA READY and another is used as receiver as ACK (data has been read)
  - Or a control line can be used as as IRQ line

Control Word



Keyboard/Display Interfaces to 82C55A



Assembler example

- This example is a code fragment from a keyboard interrupt handler for a screen saver.
- This code is executed when a keystroke has been detected
- The keystroke is read from the controller and discarded prior to restoring the screen

X86 I/O Instructions: IN (INput from port)

- **Purpose:**
  Input a byte, word, or dword from an I/O port.
- **Syntax:**
  IN AL,port OR: IN AX,port OR IN eax, port
- **Semantics:**
  AL <- port OR AX <- port OR eax <- port
  Flags: ODITSZAPC unchanged
- **Operands:**
  – port can be immediate or DX
- Notes:
  – Ports are numbered 0000H through FFFFH. Port can be specified either as an immediate operand or can be specified in DX. No other register may be used for I/O addressing

X86 I/O Instructions: OUT (Output to port)

- **Purpose:**
  Output a byte, word, or dword to an I/O port.
- **Syntax:**
  OUT port,al OR: OUT port,ax OR OUT port,eax
- **Semantics:**
  Port <- al OR Port <- ax OR Port <- eax
  Flags: ODITSZAPC unchanged
- **Operands:**
  – port can be immediate or DX
- Notes:
  – Ports are numbered 0000H through FFFFH. Port can be specified either as an immediate operand or can be specified in DX. No other register may be used for I/O addressing

Assembler Example

```
vec91:
  push ax
  in al, 60h       ; read scan code from controller
  in al,61h        ; keyboard control port
  or al,80h        ; set ack bit
  jmp $+2          ; delay a little bit
  out 61h,al       ; send affirmation
  and al,7Fh       ; clear affirmation bit
  jmp $+2          ; delay a little bit
  out 61h,al       ; send to port  B
  mov al,20h       ; I/O Address of 8259 PIC
  out 20h,al       ; Code 20H = EOI
  pop ax
  IRET
```

Another assembler example: the speaker

```
jmp start
  tones dw 523, 659, 784, 1026 ; notes C,E,G,C on standard
                               ; equal-tempered chromatic scale
start:
  mov cx, 2             ; play 4 tones twice
L0:
  mov si, offset tones
  push cx               ; save outer loop counter
  mov cx, 4             ; inner loop counter: number of tones
L1:
  call speakerOn        ; turn on the speaker
  mov bx, [si]          ; param for beep: frequency
  mov dx, 6             ; sound for 1/3 second
  call Beep
  call speakerOff       ; turn off speaker
  mov dx, 3             ; silent for 1/6 second
  call Waits
  add si, 2             ; get next tone
  Loop L1               ; inner loop
  pop cx                ; outer loop counter
  loop L0               ; outer loop
  mov ax, 4c00h         ; terminate program
  int 21h
```

Speaker On/Off

```
speakerOn:
; enable the speaker by setting bits 0-1 in the PPI
; (programmable peripheral interface) control register
  push ax
  in al, 61h            ; get ppi control register
  or al, 3              ; enable speaker
  out 61h, al
  pop ax
  ret

speakerOff:
; disable the speaker by clearing bits 0-1 in the PPI
  push ax
  in al, 61h            ; get ppi control byte
  and al, 0FCh          ; disable speaker
  out 61h, al
  pop ax
  ret
```

Beep

```
Beep:
; play a tone on the speaker by programming timer channel 2
; the input oscillator operates at 1.19318 MHz.
; To program X Hz, divide 1,193,180 by X and load
; the quotient into timer channel 2 countdown register
; Parameters: BX   Desired frequency in Hz
;    CX   Period to wait in ticks
  push ax
  push bx
  push dx
  mov dx, 12h           ; 1234DCh = 1,193,180
  mov ax, 34DCh
  div bx
  out 42h, al           ; send low byte to channel 2
  mov al, ah            ; prepare to send high byte
  out 42h, al           ; send high byte
  pop dx                ; restore time count
  call waits
  pop bx
  pop ax
  ret
```

Wait

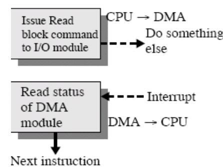
```
Waits:
; wait for specified ticks in dx. Calculate ending time by adding
; ticks to current time, then sit in a tight loop, subtracting stop
; time from ending time until there is no longer a carry.
  pusha                ; we use a lot of registers!
  push es
  mov ax, 0040h        ; address bios data area
  mov es, ax
  sub bx, bx           ; we'll use bx:cx as 32-bit integer
  add dx, es:[6Ch]     ; add in the low word to get stopping time
  adc bx, es:[6Eh]     ; now bx:cx has the stopping time
Wait1:
  mov bp, es:[6Ch]     ; get low word of current time
  mov ax, es:[6Eh]     ; and the high word
  sub bp, dx           ; subtract from stop time
  sbb ax, bx           ;
  jc wait1             ; keep waiting if non-zero
  pop es
  popa
  ret
```

Direct Memory Access

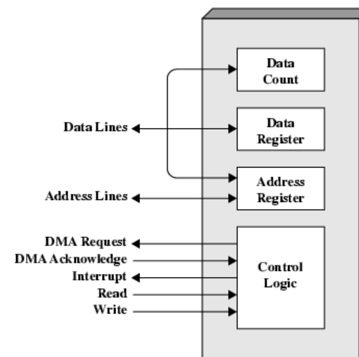
- Both interrupt driven and programmed I/O require active CPU intervention
  - Processor must transfer data
    - Read from device or memory
    - Write to memory or device
  - Transfer rate is limited
  - CPU is tied up
- DMA allows devices to communicate directly with memory without passing data through the CPU

DMA Function

- Additional Module (hardware) on bus
- DMA controller takes over from CPU for I/O



Typical DMA Module Diagram



DMA and the System Bus

- In order to access memory the DMA module must pass data over the bus
- DMA and the CPU cannot share the bus
  - But cpu may have cached instructions/data
  - DMA can suspend processor operations temporarily (cycle stealing)
- On-board level 2 cache may keep the CPU busy without accessing the system bus
  - But cache coherency can be a problem

DMA Transfer
Cycle Stealing

- DMA controller takes over bus for a cycle
- Transfer of one word of data
- Not an interrupt
  - CPU does not switch context
- CPU suspended just before it accesses bus
  - i.e. before an operand or data fetch or a data write
- Slows down CPU but not as much as CPU doing transfer

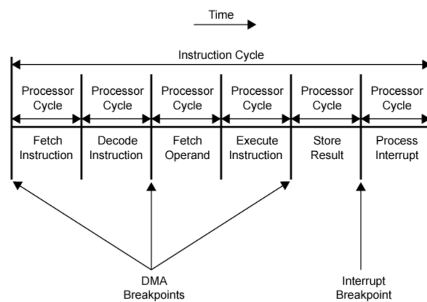
DMA Operation

- CPU tells DMA controller:-
  - Read/Write
  - Device address
  - Starting address of memory block for data
  - Amount of data to be transferred
- CPU carries on with other work
- DMA controller deals with transfer
- DMA controller sends interrupt when finished

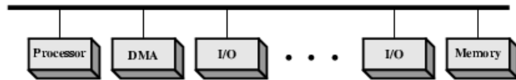
First Party and Third Party DMA

- The DMA discussed here is sometimes called "third party DMA" because a third party (the DMA controller) performs the transfer
  - It does not perform as well as "first-party" DMA where DMA circuitry is part of the IO device
  - First-party DMA is supported by PCI and newer buses
  - The device itself becomes a bus master
    - Example: ATA drives typically use "Ultra DMA"
    - Terms such ULTRA ATA/100, Ultra ATA/66 etc are marketing terms, not standards

DMA and Interrupt Breakpoints During an Instruction Cycle

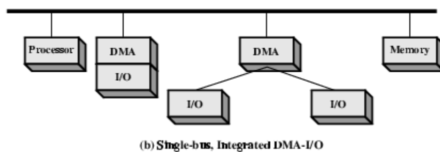


DMA Configurations (1)



- Single Bus, Detached DMA controller
- Each transfer uses bus twice
  - I/O to DMA then DMA to memory
- CPU is suspended twice (2 bus cycles)

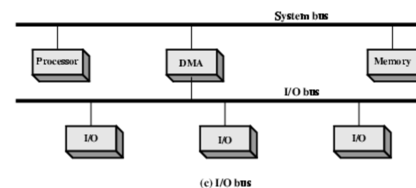
DMA Configurations (2)



(b) Single-bus, Integrated DMA-I/O

- Single Bus, Integrated DMA controller
  - Direct connection between DMA and I/O module
- Controller may support >1 device
- Each transfer uses bus once
  - DMA to memory
- CPU is suspended once

DMA Configurations (3)



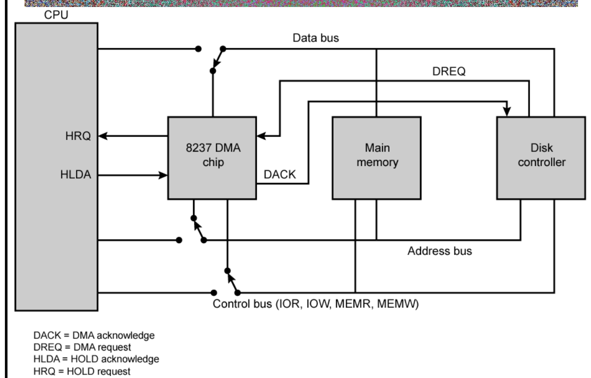
(c) I/O bus

- Separate I/O Bus
- Bus supports all DMA enabled devices
- Each transfer uses bus once
  - DMA to memory
- CPU is suspended once

Intel 8237A DMA Controller

- Interfaces to 80x86 family and DRAM
- When DMA module needs buses it sends HOLD signal to processor
- CPU responds HLDA (hold acknowledge)
  - DMA module can use buses
- E.g. transfer data to/from memory to/from disk
  1. Device requests service of DMA by pulling DREQ (DMA request) high
  2. DMA puts high on HRQ (hold request),
  3. CPU finishes present bus cycle (not necessarily present instruction) and puts high on HDLA (hold acknowledge). HOLD remains active for duration of DMA
  4. DMA activates DACK (DMA acknowledge), telling device to start transfer
  5. DMA starts transfer by putting address of first byte on address bus and activating MEMR; it then activates IOW to write to peripheral. DMA decrements counter and increments address pointer. Repeat until count reaches zero
  6. DMA deactivates HRQ, giving bus back to CPU

8237 DMA Usage of Systems Bus



Fly-By

- While DMA using the bus the processor is idle unless operating from cache, and when the processor uses bus, DMA is idle
- The 8237 is a “fly-by” DMA controller
- Data does not pass through and is not stored in DMA chip
  - DMA only between I/O port and memory
  - Not between two I/O ports or two memory locations
- Can do “non fly-by” memory to memory via register (2 bus cycles)
- 8237 contains four DMA channels
  - Programmed independently
  - Any one active
  - Numbered 0, 1, 2, and 3

8237 Registers

Bit	Command	Status	Mode	Single Mask	All Mask
D0	Memory-to-memory E/D	Channel 0 has reached TC	Channel select	Select channal mask bit	Clear/set channel 0 mask bit
D1	Channel 0 address hold E/D	Channel 1 has reached TC			Clear/set channel 1 mask bit
D2	Controller E/D	Channel 2 has reached TC	Verify/write/read transfer	Clear/set mask bit	Clear/set channel 2 mask bit
D3	Normal/compressed timing	Channel 3 has reached TC		Not used	Clear/set channel 3 mask bit
D4	Fixed/rotating priority	Channel 0 request	Autoinitialization E/D		Not used
D5	Late/extended write selection	Channel 0 request	Address increment/decrement select		
D6	DREQ sense active high/low	Channel 0 request	Demand/single/block/ cascade mode select		
D7	DACK sense active high/low	Channel 0 request			

8237 Register Functions

- Command register is loaded to control the operation of the DMA. Can perform operations such as memory block fill by disabling auto-increment
- Status register indicates for each channel if Terminal Count has been reached and if any requests are pending
- Mask registers can used to enable/disable individual channels or all 4 channels

Mode Register

- Determines mode of operation
  - D0 and D1 are channel select
  - D2 and D3 determine I/O->mem, mem-> I/O or a verify operation
  - D4 set will reset memory addr register and count to original values at termination of operation
  - D5 controls address autoincrement/decrement
  - D6 and D7 determine single mode (single byte of data), block mode, demand mode (allows premature termination) and cascade mode

The Evolution of the I/O Function

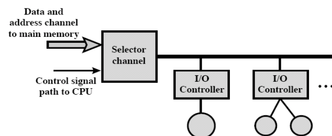
1. CPU directly controls device
2. Controller or I/O module; simple programmed transfer by CPU. Device details transparent to CPU
3. Same as above with interrupts; more efficient
4. I/O module accesses memory through DMA
5. I/O module becomes processor
   - Executes stored program
   - CPU directs module to program in memory
   - I/O processor executes program and generates interrupt on completion
6. I/O module has local memory and is a computer in its own right.

I/O Channels and Processors

- Steps 5 and 6 (processing capability, local memory) we talk about an I/O channel or I/O processor instead of I/O module
  - e.g. mainframe terminal controllers, 3D graphics cards, USB hubs
- CPU instructs I/O controller to do transfer or execute stored program
- Improves speed
  - Takes load off CPU
  - Dedicated processor is faster

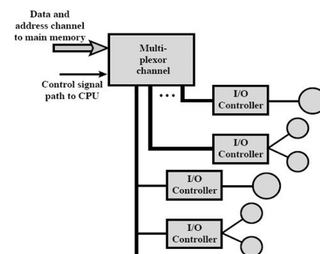
I/O Channel Architecture: Selector Channel

- Controls multiple high-speed devices
- Dedicated to one device at a time



I/O Channel Architecture: Multiplexor Channel

- Used with low or high speed devices
- Low speed: byte multiplexor
- High speed: block multiplexor



I/O Channel Architectures

- I/O channel architectures have traditionally only been used on mainframe computers
- They are now being used in high-performance file servers and in storage networks

Interfacing

- Connecting devices together
- Serial and Parallel
  - Traditional usage parallel for high speed, serial for low speed
  - Recent develop in high-speed serial interfaces have given serial devices the advantage
    - USB
    - Gigabit ethernet
    - SATA drives
  - Serial cables are smaller and easier to manage electrically
    - Interference between lines limits speed of parallel interfaces
    - Synchronization of lines poses a problem at high speed and limits cable length
    - Cables are bulky and require lots of shielding

Point-to-point and multipoint interfaces

- Point-to-point interfaces provide a dedicated line to a device
  - Ex: keyboard, rs-232, parallel printer interface
- Multipoint external interfaces support multiple devices
- Effectively behave as external buses
  - Ex: USB, FireWire, InfiniBand

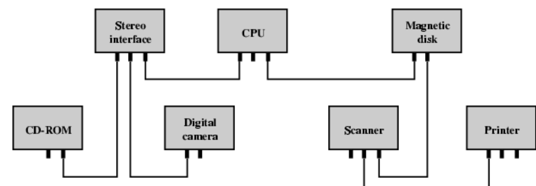
IEEE 1394 FireWire

- High performance serial bus
- Fast, low cost, easy to implement
- Also being used in digital cameras, VCRs and TV as well as computer systems

FireWire Configuration

- Daisy chain configuration
  - Not strict, can be tree-structured
- Up to 63 devices on single port
  - Really 64 of which one is the interface itself
- Up to 1022 buses can be connected with bridges
- Hot plugging—connect and disconnect peripherals while powered up
- Automatic configuration
- No bus terminators needed

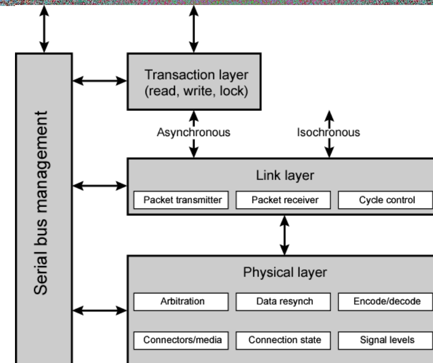
Simple FireWire Configuration



FireWire 3 Layer Protocol Stack

- Physical Layer
  - Defines transmission medium, electrical and signaling characteristics
- Link Layer
  - Describes transmission of data in packets
- Transaction
  - Defines a request-response protocol; hides low level details from applications

FireWire Protocol Stack



FireWire - Physical Layer

- Data rates from 25 to 400Mbps
- Converts bits to electrical signals
- Provides arbitration service
- Two forms of arbitration
  - Based on tree structure, daisy chain is a special case
  - Devices in a tree configure themselves so that one node is designated as root
  - Root acts as bus arbiter; process requests in first come first served order
  - Natural priority controls simultaneous requests
    - i.e. who is nearest to root

Firewire Arbitration

- The tree structure arbitration is supplemented by two additional functions:
  1. Fairness arbitration
     - Time is organized into "fairness intervals"
     - At beginning of interval each node sets an arb_enable flag
     - Once access is gained, flag is cleared and device may not compete again during the interval
     - Prevents high-priority devices from monopolizing the bus
  2. Urgent arbitration
     - Some devices can be configued with *urgent* priority
     - These devices can get the bus more than once during a fairness interval
     - Three urgent packets can be transmitted for every one non-urgent packet

FireWire - Link Layer

- Two transmission types
  - Asynchronous
    - Variable amount of data and several bytes of transaction data transferred as a packet
    - Packet is transferred to an explicit address
    - Acknowledgement returned
  - Isochronous
    - Variable amount of data in sequence of fixed size packets at regular intervals
    - Simplified addressing
    - No acknowledgement

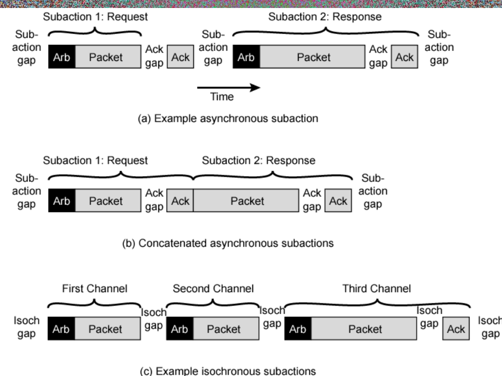
Asynchronous and Isochronous Data

- Asynchronous transmission for devices without fixed data rate requirements
  - Ex. Printer, scanner, CD, DVD
  - Default aribtration is fair arbitration
- Isochronous transmission for devices the require a substantial portion of bus capacity or have stringent latency requirements
  - Real-time data collection, digital sound or video inputs
  - Use urgent arbitration method
  - Guarantees data rate / latency

Transactions

- The process of delivering a single packet is called a subaction
- An asynchronous subaction has 5 time periods:
  1. Activation sequence
     - The exchange of signals required to give a device control of the bus
  2. Packet Transmission
     - Packet header has source and destination IDs, packet type, CRC checksum, and type-specific parameters
     - Data block has user data and another CRC
  3. Acknowledgement Gap
     - Time delay for for dest to receive and decode a packet, generate acknowledgement
  4. Acknowledgement
     - Recipient returns an ACK packet indicating action taken
  5. Subaction Gap
     - Enforced idle period to ensure that other nodes do not begin arbitration before ACK packet has been transmitted

Typical FireWire Transactions



(a) Example asynchronous subaction

(b) Concatenated asynchronous subactions

(c) Example isochronous subactions

Concatenated Asynchronous Subactions

- At the time that the acknowledgement is sent the acknowledging node has the bus
- If the exchange is a request/response transaction then the responding node can immediately transmit the response packet without an arbitration sequence

Isochronous Transactions

- To accommodate mixed traffic of isochronous and asynchronous transactions one node is designated as *cycle master*
- Periodically the cycle master issues a cycle start packet signaling to all devices that an isochronous cycle has begun
- During an isochronous cycle only isochronous packets may be sent
- Each device arbitrates for bus access; winning node transmits immediately
- No ack packets, so other devices resume arbitration immediately after packet transmission
  - Note that isochronous packets are smaller than asynch packets. Packet header includes negotiated 8 bit channel number, data length field and CRC

Mixed Traffic

- Isochronous traffic requires only a small gap between the end of a packet and the start of arbitration (smaller than a subaction gap)
- After all isochronous sources have transmitted a subaction gap will occur
- Signals to asynchronous devices that they may now compete for access

InfiniBand

- I/O specification aimed at high end servers
  - Merger of Future I/O (Cisco, HP, Compaq, IBM) and Next Generation I/O (Intel)
- Version 1 released early 2001
- Architecture and spec. for data flow between processor and intelligent I/O devices
- Intended to replace PCI in servers
  - Enables servers, remote storage, network devices etc to be connected in a central fabric of switches and links
  - Can accommodate up to 64,000 devices
- Provides increased capacity, expandability, flexibility in server design

Infiniband compared to PCI

- PCI has to exist on the machine chassis; Infiniband is designed for external use
- Devices are attached to a central fabric of switches and links
- Allows greater server density (in a farm) by removing I/O from the server chassis
- PCI measures distances in centimeters
- Infiniband allows distances of 17 m. (copper); 300 m. (multimode optical fiber) and 10km. (single mode optical fiber)
- Transmission rates up to 30 Gbs compared to 1Gbs PCI

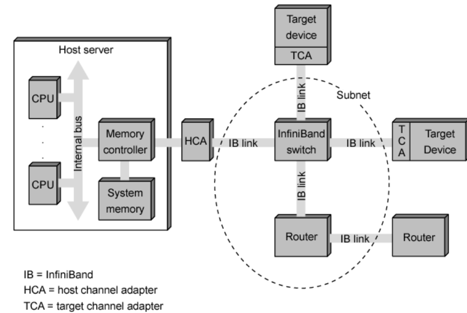
Key Elements

- Host Channel Adapter (HCA)
  - A typical server has a single interface to an HCA that links the server to an Infiniband switch
  - HCA attaches at a memory controller that controls traffic between CPUs, memory, and HCA
  - HCA uses DMA to access memory
- Target Channel Adapter (TCA)
  - TCAs connect storage systems, routers and other peripheral devices to an Infiniband network
- InfiniBand Switch
  - Provides point-to-point physical connections to devices and switches traffic from one link to another
  - Servers and devices communicate through HCAs via the switch
  - Switch manages linkage without interrupting servers' operation

Key Elements

- Link
  - The link between a switch and a channel adapter or between two switches
- Subnet
  - One or more interconnected switches plus the links that connect devices to switches
  - Allow confinement (if desired) of broadcast and multicast transmissions
- Router
  - Connects InfiniBand subnets, or connects a switch to a LAN, WAN, or storage network

InfiniBand Switch Fabric



InfiniBand Operation

- 16 logical channels (virtual lanes) per physical link
- One lane for fabric management, rest for data transmission
- Data in stream of packets
- A virtual lane is dedicated temporarily to end to end transfer
- Switch maps traffic from incoming to outgoing lane
- Protocol stack provides queues to accommodate speed differences

InfiniBand Stack Layers

- Physical
  - 1x, 4x, 12x link speeds = 2.5, 10, 30 GBs
- Link
  - Defines packet structure, addressing scheme, virtual lane logic, error detection
- Network
  - Routes packets between subnets
- Transport
  - End-to-end management

InfiniBand Protocol Stack