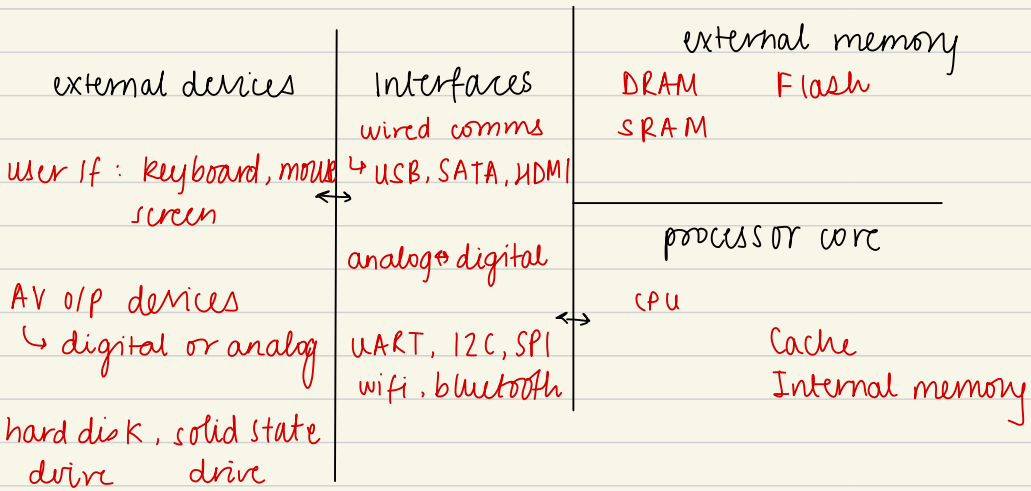
lecture notes

- major content : components of comp excluding CPU
- 10 mcq ques from Parts 2 in final exams

Computer Systems Overview



Sub-Systems

1. Processor core

2. Signal chain subsystem
   - deals w/ interfacing the comp to the real world

- analog to digital convertor, ADC in short, brings real world signal into comp
- digital to analog convertor, DAC, pushes information from comp to world

- parallel and serial Digital Interfaces
- Direct memory access controller
- Interrupt controller

3. Memory Subsystem

- semiconductor memories (SRAM, DRAM, FLASH)
- Solid state Drives
- Magnetic hard disk drives
- cache memory management
- virtual memory management.

4. Communication and User Interface Sub-system

- Wired
- Wireless
- Input devices
- output devices

Signal Chain Subsystem

1. ADC - DAC

Analog & Digital Signal

→ all real world signals are analog (continuous in voltage level, over time and space)

→ earlier most processing would be done in analog domain
→ digital processors are cheaper and more flexible in implementation and more tolerant to noise and component aging

→ digital signals are discrete time representation of analog signal
→ obtained by passing analog signals through ADC and "digitizing" them
→ digital signals have discrete voltage levels

→ analog signals can be reconstructed from digital signals via digital to analog conversion (DAC)

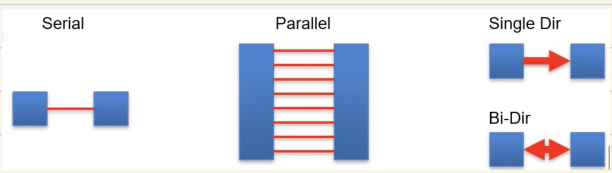
Analog $\xrightarrow{\text{ADC}}$ Digital $\xrightarrow{\text{DAC + filter}}$ Analog

- sample AS at discrete time (sampling interval)
- nyquist theorem: sampling freq $\geq$ 2 (signal freq)

- most interfaces in this course will be digital

2. Interface ?

- a boundary where I/f devices meet to exchange info
- modes of connection : single bit data transfer (serial)
  multiple bit data transfer (parallel)
- direction : single direction
  bi direction



- Semiconductors Interface via Pins
- 3 types : input , output , bidirection (both I & O)
- Interfaces are compatible if they share the electrical signal level and communication protocol

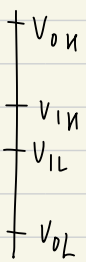
↓

O/P level of o/p device ≤ max input voltage level

3. Electrical Signal Level for Digital Data Transfer

- V_{OH} : min o/p voltage for logic 1
- V_{OL} : max o/p voltage for logic 0
- V_{IH} : min i/p voltage for logic 1
- V_{IL} : max i/p voltage for logic 0

when transmiting $V_{OH} \geq V_{IH}$
$V_{OL} \leq V_{IL}$



① higher margin between V+ & V- to differentiate between logic 0 and 1.

② external interference has the same effect on V+ and V- so the voltage difference b/w V+ and V- remains approximately the same

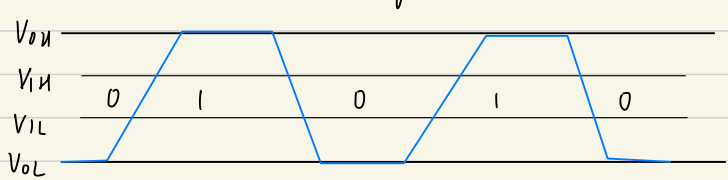
4 Differential Signals

larger margin

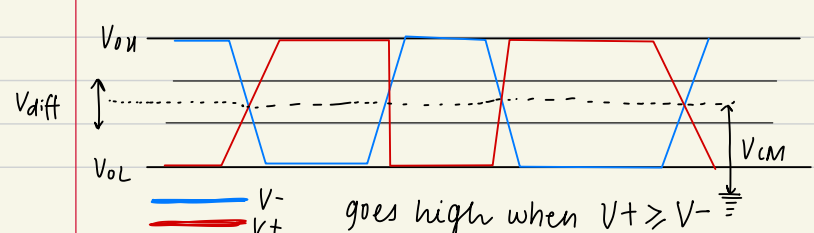
also allow for external interference cancelation

have better noise tolerance .. can be clocked at a higher frequency

V_{CM} = common mode ground



Single-ended digital signals (TTL, CMOS, RS-232)



goes high when $V+ \geq V-$

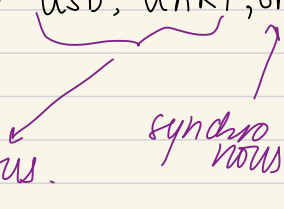
Differential digital signals (LVDS, ECL, PECL)

6. Communication Protocols

- refers to how data is formatted during transmission

3 parts of a format
- → number of bits in transmission frame
- → what synchronization to use
- → types of data and its formatting

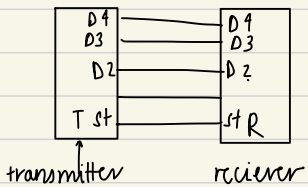
- examples of communication protocol : USB, UART, SPI



synchronous

preferred when sustained high speed is needed

7. Parallel Data Transfer



- st : strobe
  - ↑ signal that informs receiver when to latch in the data
- able to achieve higher transfer rate than serial interface on the same clock.
- more prone to signal skew and crosstalk
  - limit max clocking frequency

a. Signal Skew

- signal in a particular data line (one or more) took different amounts of time to reach the receiver, then there's a 'skew' b/w signals in the parallel data bus
- due to variation in propagation delay b/w signals in the same data bus
  - capacitance and resistance of data line affects pd.
    - like loading data lines within the same data with capacitors
    - $T = RC$
    - basically $pd \propto R$ & $pd \propto C$
    - affected by printed circuit board
      - PCB length / width
      - connecting active components
        - and bad soldering

"well behaved" signals ⇒ signal propagation delay in the system has been tweaked so that there is no signal skew among signals

b. Crosstalk

- interference b/w signals from diff data lines
- crosstalk can be transmitted electrically or via electromagnetic radiation
- ↑ w/ increase in frequency, number of lines
- ground plane helps absorb circuit noise

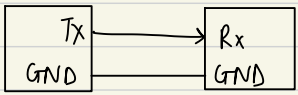
c. Pros vs Cons

- Advantages
  - → fast (more bits / sec) (er than serial at same clock rate)
  - → HW interface is simpler as only strobe signal is needed

- Disadvantages
  - → signal skew and cross talk limit max clocking speed and transfer distance
  - → HW cable can be bulky if data width is large
  - → needs more space for routing
  - → higher HW cost than serial data implementation

- signal skew and crosstalk place limitations on:
  - → maximum number of lines on data bus
  - → maximum clock rate
  - → max number of bytes transfered in one block
  - → minimum delay b/w each transfer

9. Serial Data Transfer



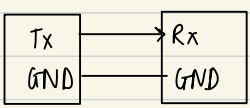
- one bit per cycle
- less affected by signal skew and crosstalk, can support higher freq. clocking

- Data transfer rate is lower comp. to parallel given same clock rate

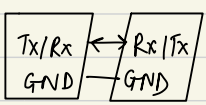
- reliable over longer distances
- lower cost since only one wire is needed

a. Types.

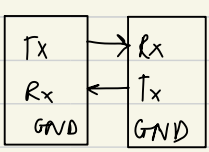
- Simplex : one direction



- Half Duplex : both directions
  Tx, Rx are mutually exclusive



- full Duplex : both at the same time



Synchronous	Asynchronous
• common clk signal b/w T & R	• no common clk signal ∴ pre-fixed clock frequency must be used for data transfer

b. Synchronous Serial Data Transfer

- common clk
- master-slave configuration : master provides the clk
  - DATA is latched on the falling edge
  - MSB first
  - master recieves at posedge, passes on at negedge

I2C, SPI → serial per

- a synchronous signal bus
- MOSI : master out, slave in
- MISO : master in, slave out
- start transfer : slave select (SS) has to be pulled low
- Data transfer : data on MOSI, MISO latched in on rising / falling clock edge
  - allows for use of multiple slaves connected to 1 master

C. Asynchronous Serial Transfer

- no common clock
- reciever must know transmitting clock rate and number of bits to be sent.
- SYNC words : indicate START STOP condition
- after recieving "start" sync the reciever starts its clock to track timing

- potential skew as 2 separate clocks are used.

- SYNC word bits help provide occasional timestamp so that the clocks get synced. (r's syncs to t's)

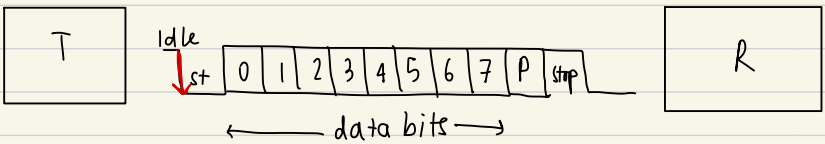
UART : universal asynchronous receiver transmitter
↓ a commonly used serial interface

communication b/w PC and processor dev boards

↑ COM port, it uses UART protocol

now, USB have replaced COM ports but they use virtual COM ports which still uses UART protocol

UART Transmit



- in IDLE state, data line is logic 1
- transmitter sends a start pattern (logic 0) to alert receiver (resulting in falling edge ↓idle start)
- data is then transfered at baud rate (a known freq)

  unit of transmission speed = number of signal state changes per second.
- at the end of data transfer, parity bit could be output
- then a STOP bit is sent (logic 1)
- every word transfer needs its own start and stop

IDLE : 1
START : 0
STOP : 1

Parity Bit

- if parity scheme is enabled, receiver checks for the parity bit & error
- even → even no of ones in total → (data + parity field)
  odd → odd no of ones ...
- Framing error : if logic 0 is present where STOP bit is expected

—x—

- configuration . 7 0 1 → 1 stop bit
  - 7 → 7 data bits
  - 0 → odd parity

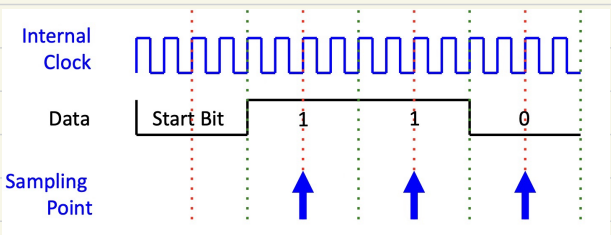
eg J : 0x4A : 100 1010

LSB is sent first ∴ waveform will show reverse binary

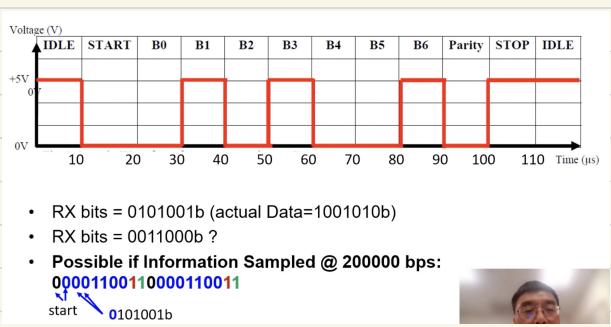
sent at earlier time

UART Receive

- falling edge due to start bit triggers an interupt and starts receiving process
- needs to know baud rate to sample data bits correctly
- internal clock runs at a multiple times of the baud rate so that sampling time is close to the middle of each data bit



← 4 x baud rate

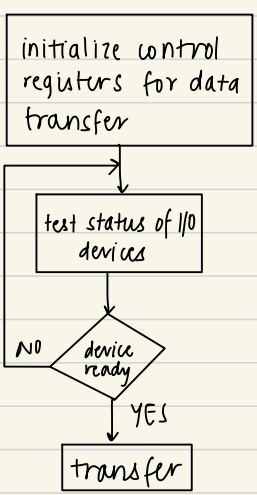


Data Transfer Mechanism

- CPU, memory, IO are connected by a system bus which is used to transfer data
- 2 controllers initiate this transfer : CPU core & DMA cont.
- 3 transfer mechanisms are used
  - → polling
  - → interrupt triggered
  - → DMA (direct memory access)

a. Polling / Programmed IO

- CPU polls a certain I/O port continuously for data
- it uses 100% of its resources to do the polling, so nothing else can be done

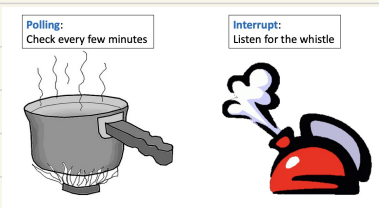


advantages :
purely sw : programmer has complete control ∴ testing & debugging is easy

disadvantage :
- inefficient use of CPU resources since nothing else can be done at that time

b. Interrupts

- a signalling mechanism that alerts CPU that its attention is needed
  - electrical pulse / change in register status
- CPU can choose whether or not to address the request
  - if yes: CPU branches off to "interrupt service routine" ISR



- interrupt latency delay: time b/w receiving IR & entering ISR
- to know where to go (which ISR) for each IR, theres a vector table which stores the starting address of each ISR in the system.
  the indexes correspond to IRs

status of key registers

receive request ✓ → check vector table → save current context → go to ISR & execute → restore saved → continue on w/ routine

- in case of multiple simultaneous IR's an arbitration scheme is set up to decide which request is served first

ad.	dis.
· efficient use of CPU resources · allows prioritization & pre-emption	· more hw interface circuitry required b/w I/O device and processor · prog is more complex and hard to debug

C. DMA: Direct Memory Access

- if CPU keeps transferring data, it has less time to perform algorithm processing & data manipulation
- DMA controller (DMAC) has dedicated hardware to move data more efficiently than CPU when complex address manipulation is req.
  ↓
  eg. de-interleaving left and right channel audio data.
- DMAC and CPU share the system bus (and other resources)
  if there are no conflicts, data transfer via DMAC & CPU execution can occur simultaneously
  else, access is given to the one with higher priority

no common resources being used

↑ this is one way of resolving the conflict.

↓ depends on processor design. is processor specific

- DMAC is a Data Bus Controller module that performs data transfer b/w memory ⇄, I/O peripheral ⇄, memory
  - peripheral → int or external
  - memory ⇄ I/O peripheral
- it generates address and initiates read/write operations
- uses Fetch and Deposit mechanism ↘

  data from source is first taken to the internal buffer of DMAC and then to the destination

the process

- it needs parameters which are configured by the CPU to the DMAC
  - → source address of data
  - → destination address of data
  - → number of words to transfer
  - → trigger signal to tell it when to transfer
    - ↓ stays in IDLE until triggered
- after data is transferred DMAC releases system bus and issues DMA Interrupt signal to CPU, informing it that the data is ready to be processed.
- DMAC's are processor specific and use a combo of 3 modes to transfer data: Burst, Cycle Stealing, Transparent

a. Burst Mode

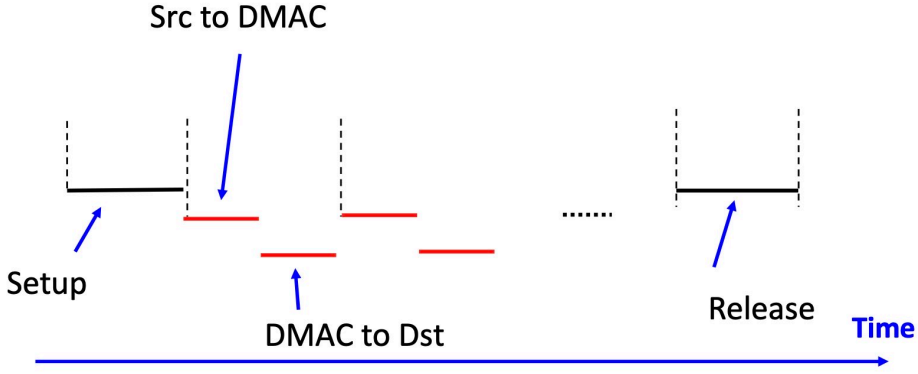
- transfers multiple units of data before returning control of the bus to the CPU
- burst could be the whole block or a subset of the block of data the DMAC has to transfer
- CPU may be suspended if it needs to access system bus while DMAC is working.
- its faster but may causse CPU to be suspended for a long period of time

b. Cycle Stealing

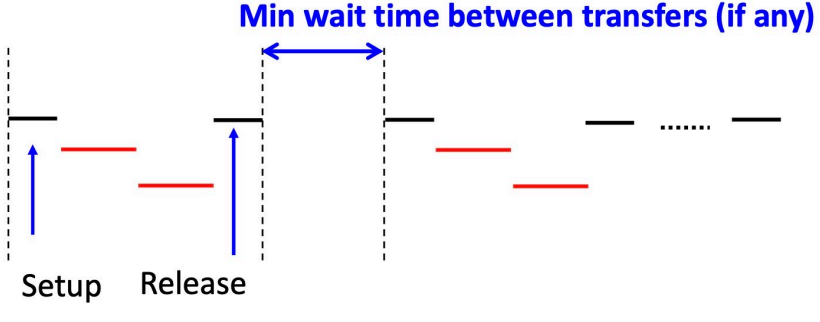
- DMAC releases system after transferring one unit of data
- depending on processor design, transfer occurs b/w ① CPU instructions, ② Pipeline stages
- again, CPU maybe suspended if it needs system bus access, but for a much shorter time
- transfer rate is slow, but CPU suspension time is short
- favoured when CPU must be responsive eg. security sys

Fetch and Deposit (Timings)

Burst



Cycle Stealing



c. Transparent mode

- special case of cycle stealing
- DMAC transfers data only when CPU is not using the databus
- slowest one, least CPU disruption
- complex hardware required to figure out when CPU is not using
- CPU activity detection can be used as a cue to trigger a burst transfer