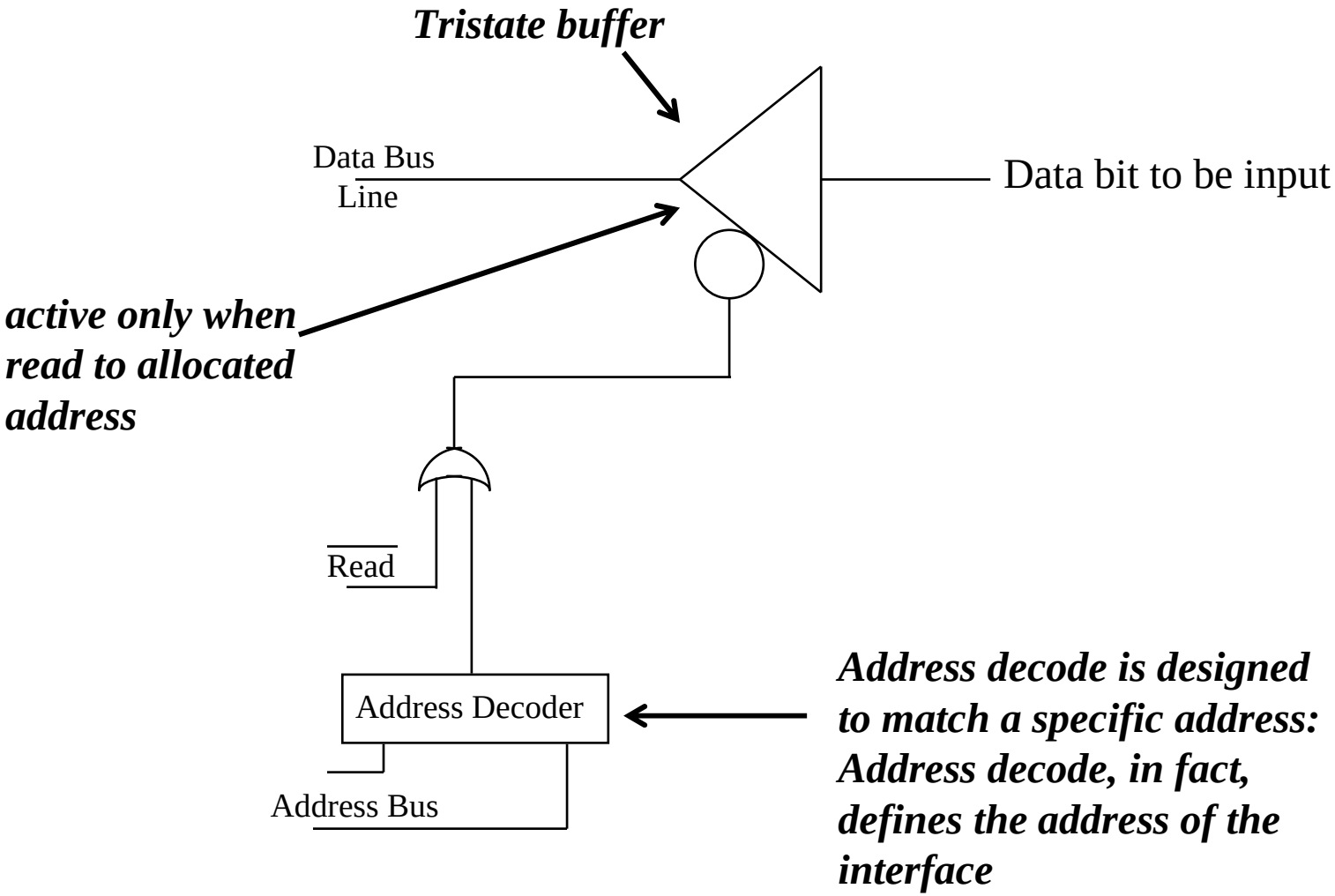
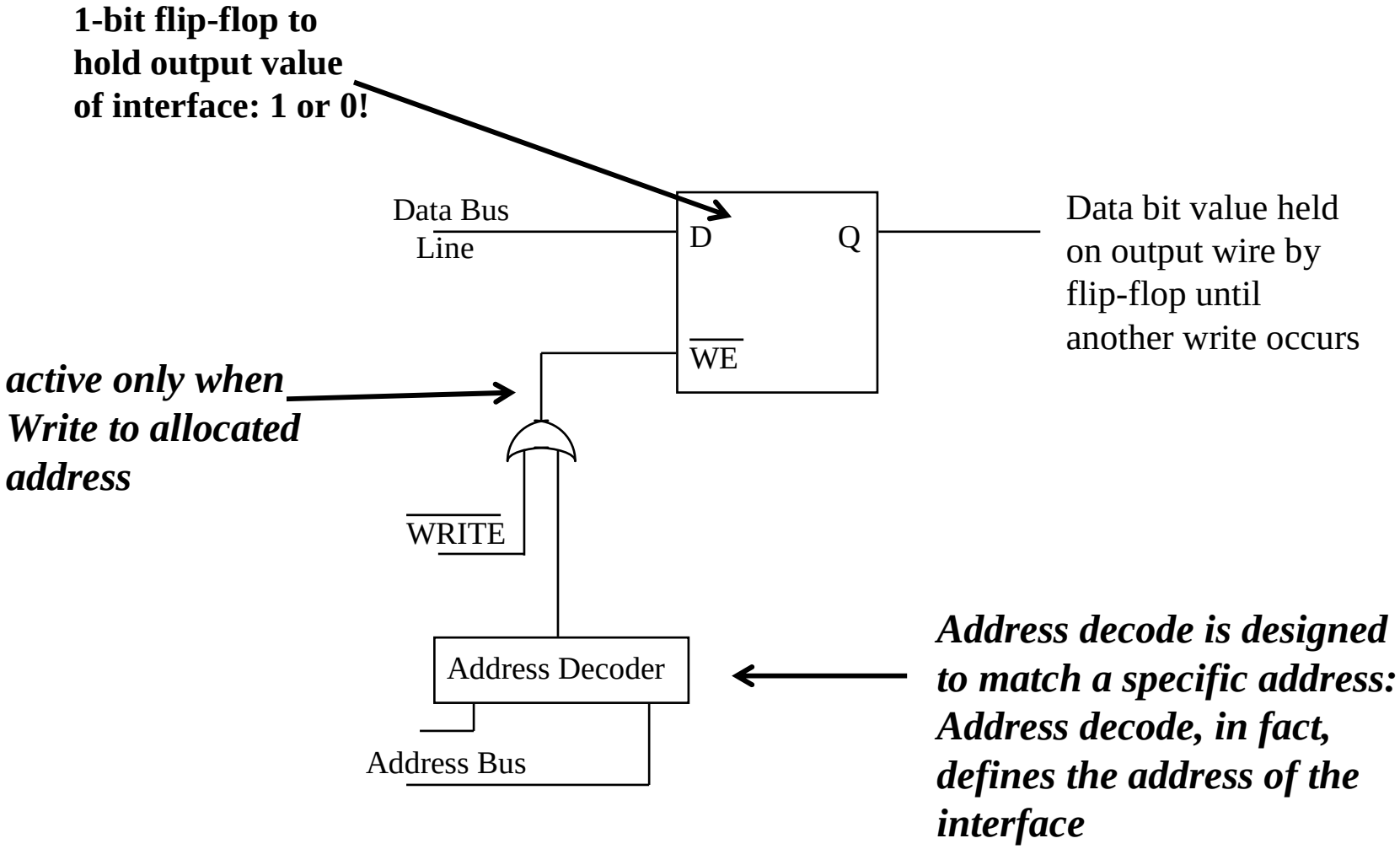
# *Input-Output (I/O)*

***Peter Rounce***
***P.Rounce@cs.ucl.ac.uk***

# Basic Input Interface Circuit:



# Basic Output Interface Circuit:



## Streamed Input-Output (I/O)

Sequence of data is sent

– each data item has to be seen **ONCE** & **ONLY ONCE**

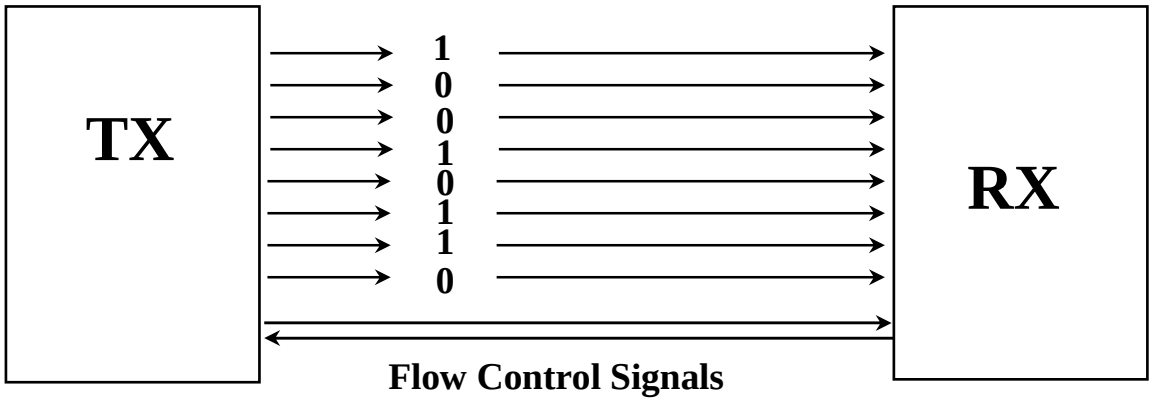
*Cool not col or ccooolll, 2002 not 202 or 20022*

Need mechanism to identify each new data item as it it sent

May need mechanism to identify the reception of each data item

## PARALLEL I/O

**Each bit of data item sent on different wire**




**Flow control signals operate at each transfer to ensure that each transmitted data item is received and read once and only once at the receiver.**

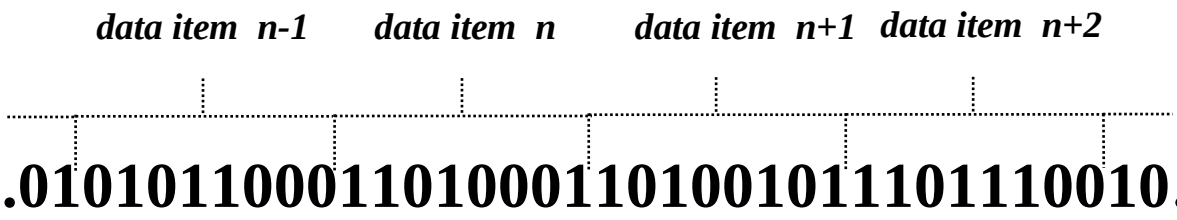
**Example:** **TX sends signal to RX to indicate each new data item sent.**

**Rx sends signal to TX to indicate when a data item has been read and the RX is *ready* for another transfer.**

Problems: expensive (lots of wire), race conditions between bits sent together limit data rate, cross-talk between bits – thus short distance to reduce these effects.

## Serial I/O

**Bits of data item are transmitted consecutively using same transmission pathway for each bit, e.g. the same wire or pair of wires.**




**This is more complicated than parallel, since each bit must seen once and only once, and the bits must be regrouped into the original data value.**

**Serial is used for the majority of I/O e.g. disk, ethernet, FAX**

- **Cheaper - less wiring**
- **more reliable - less noise, e.g. from cross-talk**
- **cheaper to shield serial than parallel**
- **no race conditions between bits of data items → faster transmission**

## Mechanisms for indicating a new data item

**Serial data transfer** systems

- **data sent one bit at a time** on a “single” wire

- **square wave (clock) signal** sent as well
  each clock cycle indicates a new data bit

- clock may be on separate wire or
  (cleverly) mixed in with data bits (ethernet) on a “single” wire

Note: the “single” wire may be a pair of wires twisted together (a “twisted pair”) that together carry a single signal stream. Telephone cables have 3 twisted pairs: two of these pairs are used in broadband – one pair for downstream transmission (into the client) and one pair for upstream (back to the telephone exchange).

# Parallel Data Transfer

**Bits of data are sent in parallel on multiple wires**
(usually 8-bit wide, occasionally 16-bit wide)

**Flow-control** (occasionally called Handshaking) **signals** are used to co-ordinate transfer of each data item

**Ready** signal **-** from Tx (transmitter) to Rx (receiver) signals each new data item sent

**Acknowledge** signal – from Rx to Tx indicates reading of last data item sent

**Exchange** of Ready & Acknowledge co-ordinates 1 data transfer

## Parallel Interfaces

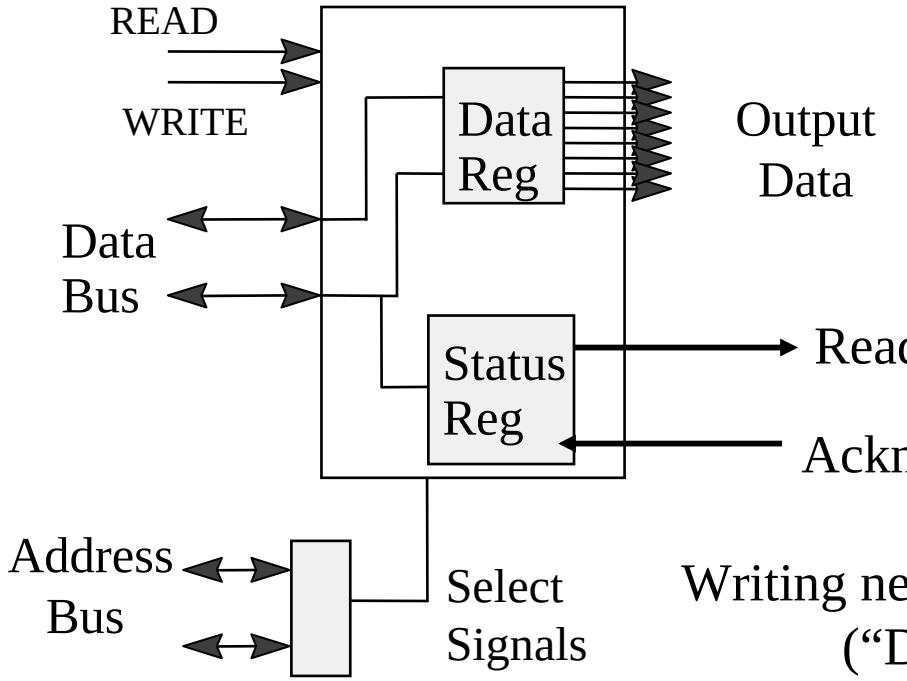
**Data Register:** data register (or data buffer) that holds outgoing or incoming data

**Status Register:** **indicates if data in data register**

1-bit latch that indicates status of interface's Data Register

1 indicates buffer full
0 indicates buffer empty

# 8-Bit Parallel Output Interface (TX – transmitter)




**Tx Status Register**

Data Reg full – data sent awaiting Acknowledge from Rx

Data Reg empty – ready to receive new data item for output.

***Flow Control Signals***

Writing new data into Data Register sets Ready ("Data Reg Full") output which signals Rx that a new data item has been sent; also sets Status Register to "Data Reg Full".

Acknowledge signal going active resets Status Register to"Data Reg Empty".

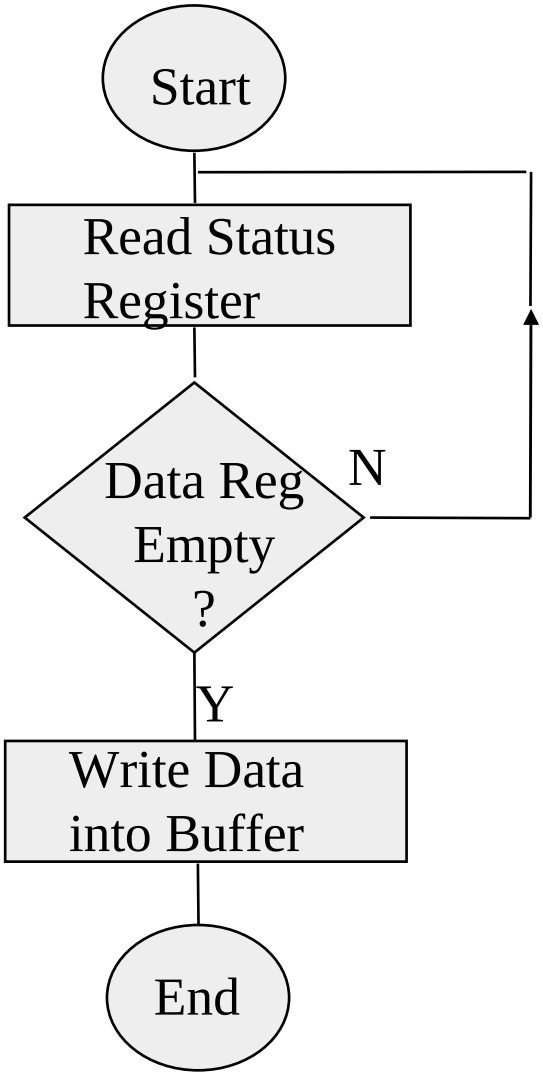
*Data and Status registers have different addresses.*

# Flowchart for putting data into Output Interface

## Polling Loop:

```
do {
        Read value from Status Register
} while (“Data Reg Full” == true ) ;
write data into Data Register ;
```

When (“Data full” == false)
write data into Data Register



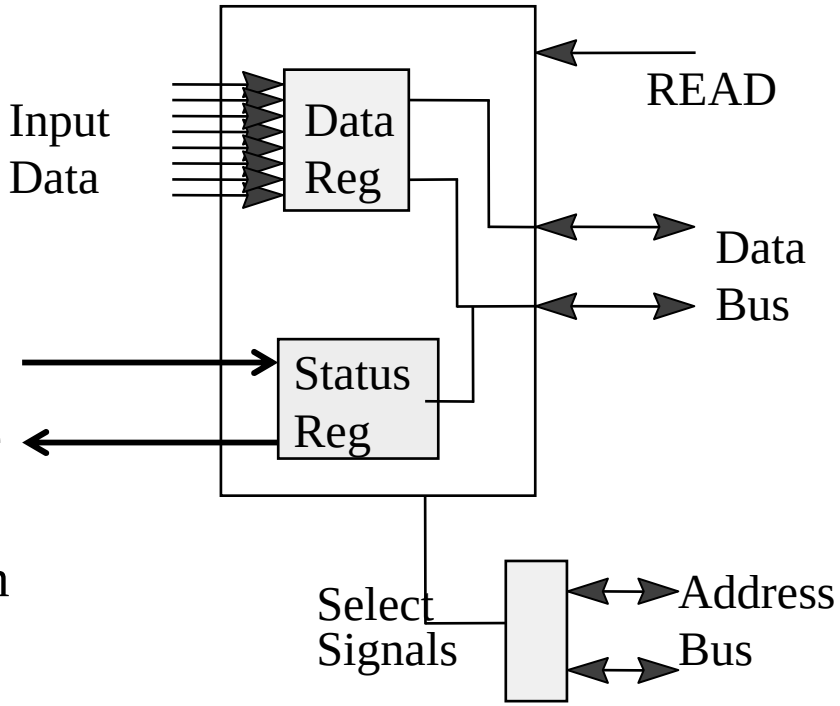
# 8-Bit Parallel Input Interface (Rx – Receiver)

**Rx Status Register**

Data Reg full – new data received needs to be read

Data Reg empty – no data, awaiting next Transmission.

***Flow Control Signals***




**Ready** signal from TX latches new data item into Rx Data Register and sets Status Register to “Data Reg Full”.

Reading data from Data Register sets Status Register to “Buffer Empty” and activates Acknowledge signal to Tx.

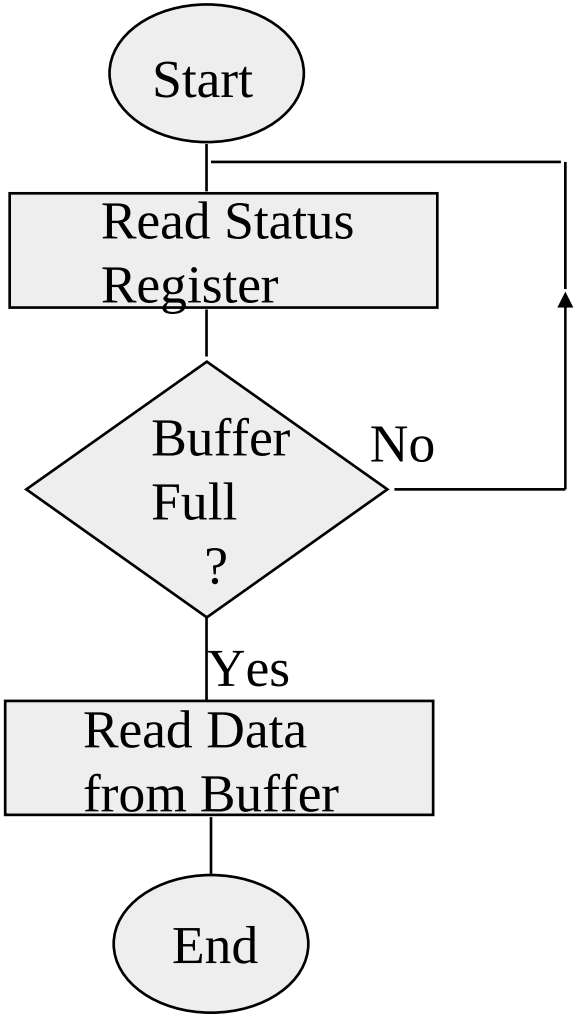
*Data and Status Registers have different addresses.*

# Flowchart for getting data from Input Interface

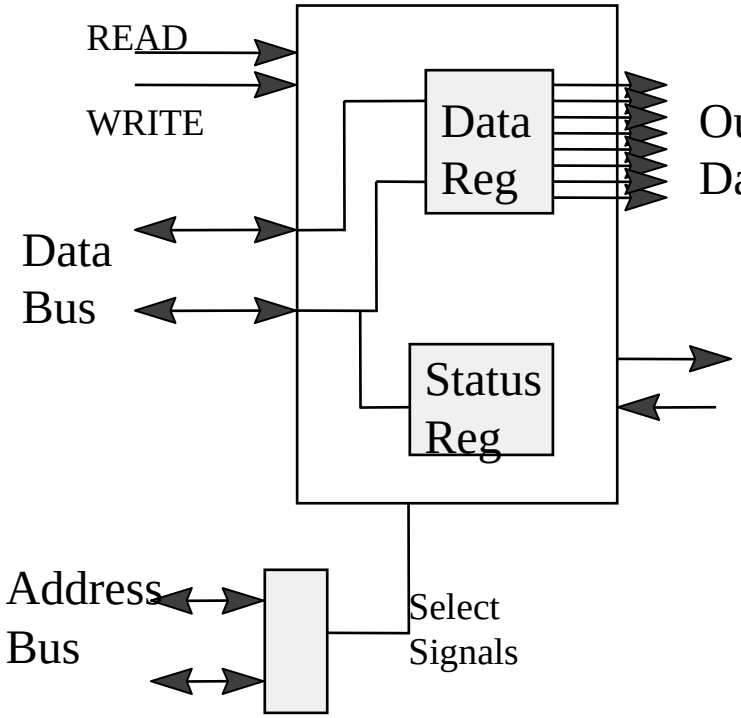
## Polling Loop:

```
do {
        Read value from Status Register
} while (“Data Reg Full” == false ) ;
```

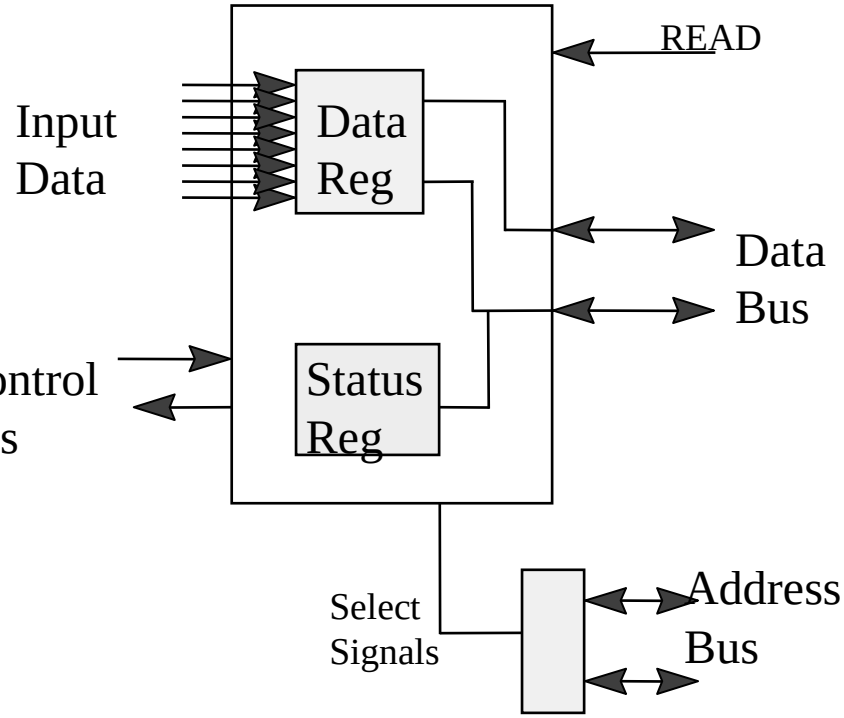
When “Data Reg full”
read data from data register



## 8-Bit Parallel Output Interface




## 8-Bit Parallel Input Interface

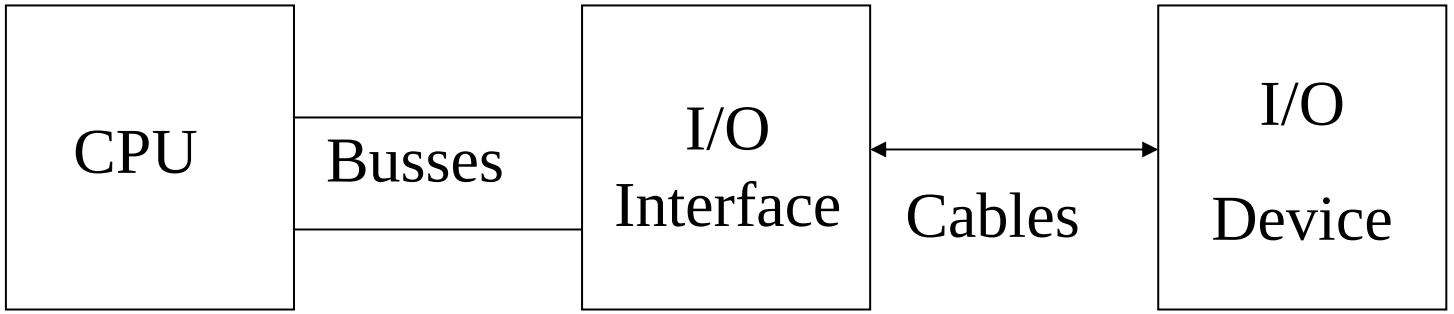



*Flow Control signals are designed so that output and input interfaces can be directly connected:*

Writing data into Tx: data goes to Input interface and sets Ready output
Ready going active latches data into Rx Data Reg & sets “Data Reg Full”

Reading data from Rx: data read from Rx Data Reg, resets Status Reg and sets Acknowledge
Acknowledge going active resets Tx Status Port to “Data Reg Empty”

## Role of Interface




I/O Devices can only be accessed by CPU via I/O Interface.

Communications: CPU send/receives data from interface.

Interface interacts with I/O device

These are distinct transfers:

Interface-device transfers are performed autonomously in a different time domain to CPU-Interface transfers.

## Role of Interface

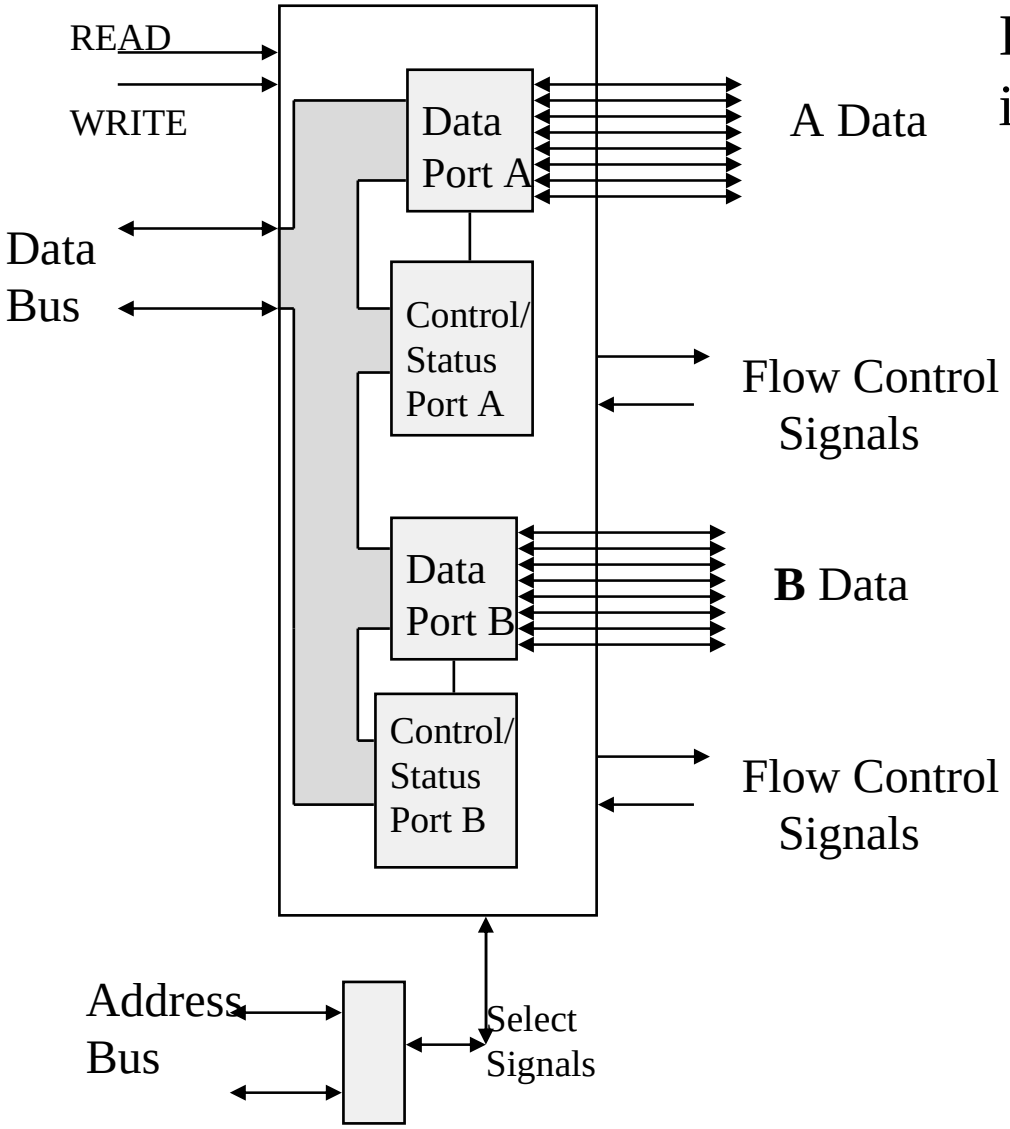
Acts as rendezvous (post box) between CPU and IO device
(CPU and IO device operate independently).

Converts data organisation, e.g. from parallel in computer to serial to device

Converts data representation,

e.g. RS232C 1: 5V in computer ←→ -12V to device
0: 0V in computer ←→ +12V to device

## Programmable 8-Bit Parallel Interface




Interfaces vary, but all similar interfaces on computer side:

Data registers for data transfers in and out

Status registers for indicating state of data registers full or empty

Control registers for setting up interface
e.g. whether Ports A and B are input or output

# Serial Data Transmission

**Most data transmission is serial – one bit at a time on a single "wire".**

**Serial is:-**

- **Cheaper – fewer wires**
- **More noise tolerant -**
  **less wires for cross-talk, fewer wires to make noise tolerant**

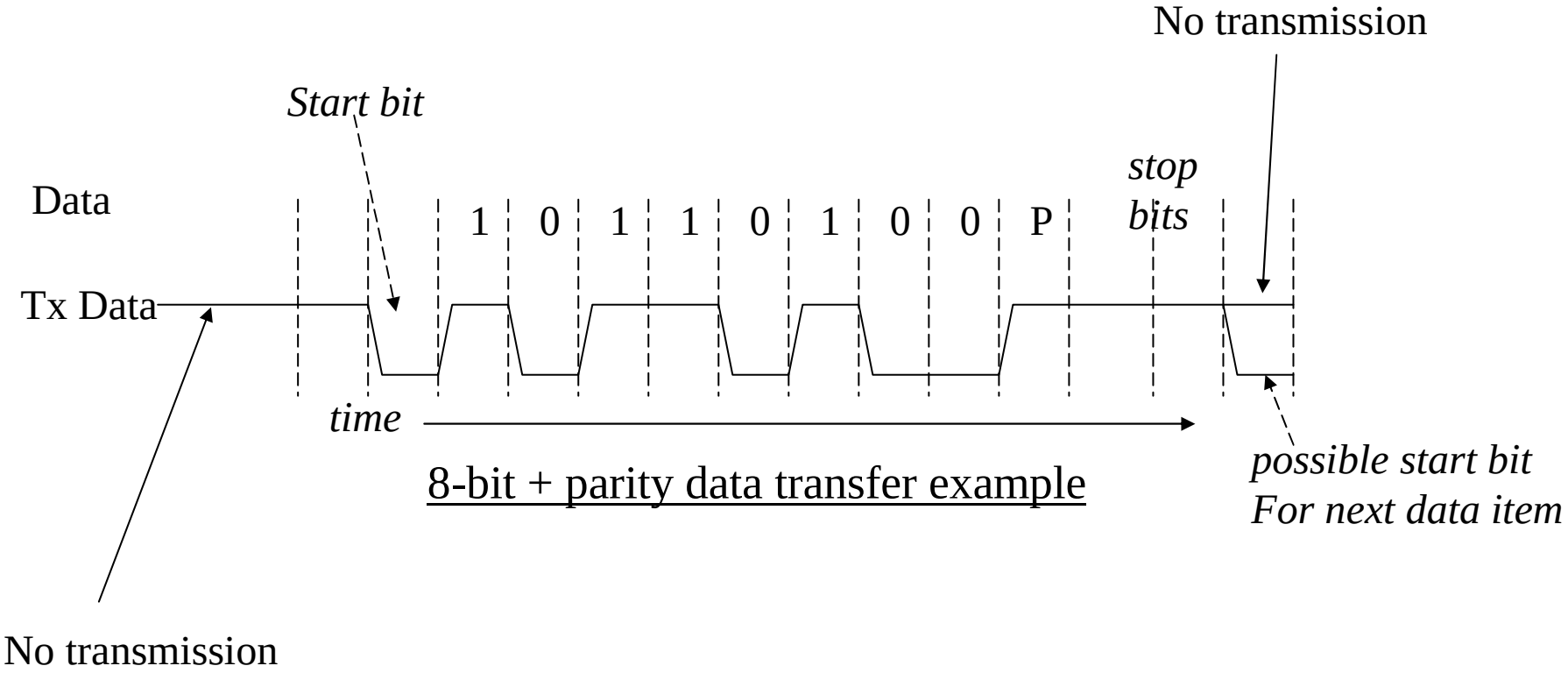
**Parallel transmission is used only over short distances < 3-6 feet**
**parallel connection to printers and scanners**
**backplane buses, I/O buses – SCSI, GPIB.**

**Serial is used for lots of things and everything > 2-3 feet**
**modems, ethernet, USB, mouse, keyboard**
**screen (1 serial connection for each colour – blue, red, green)**
**serial printers**

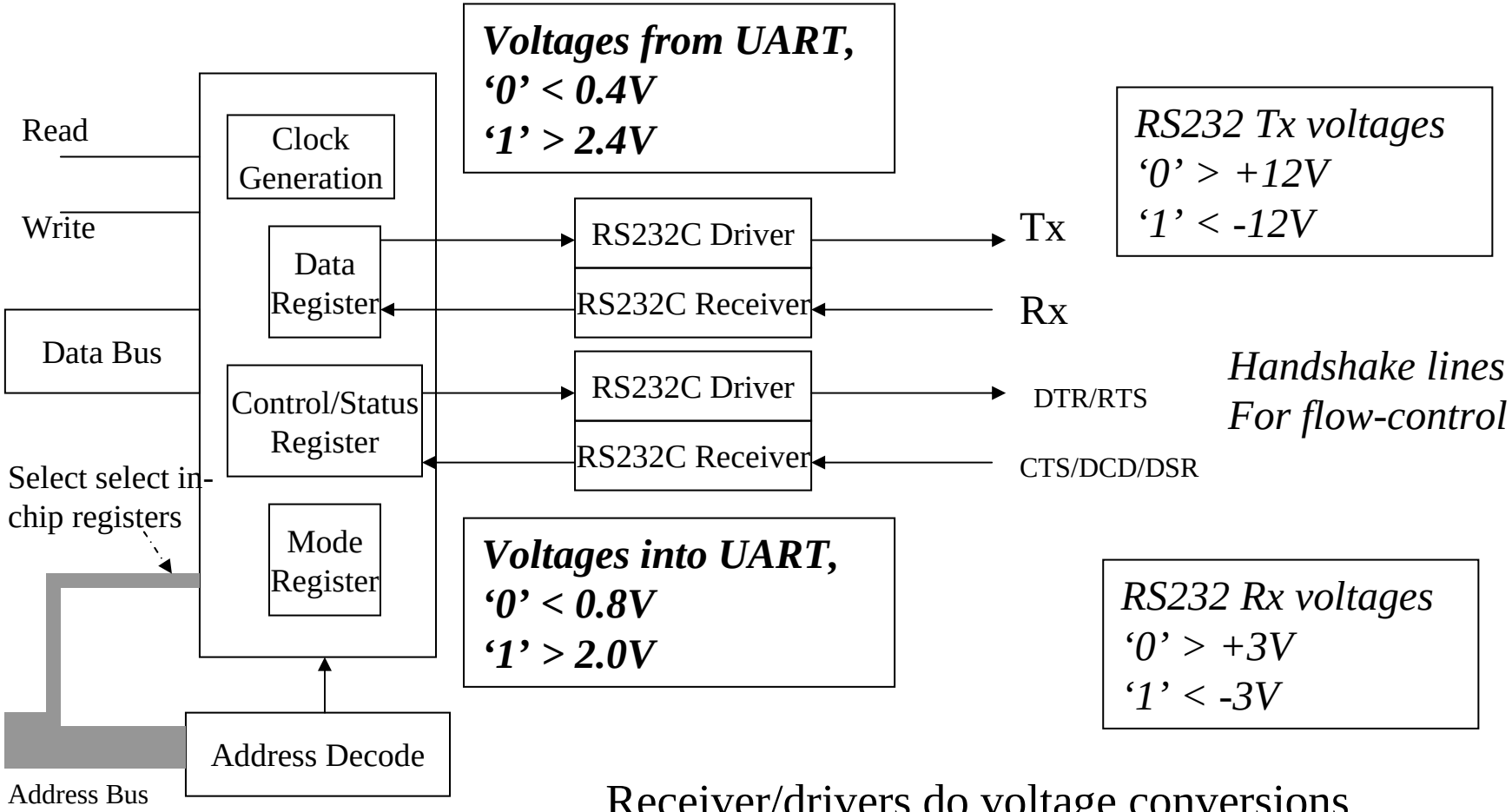
**Synchronous serial protocols are used for high frequency serial transmission with data usually sent in blocks.**

# Asynchronous Serial Data Transmission – *no clock sent!*

- Low rate, short distance transfers
- transmission protocol used by UARTs
- UART – **U**niversal **A**synchronous **R**eceiver/**T**ransmitter

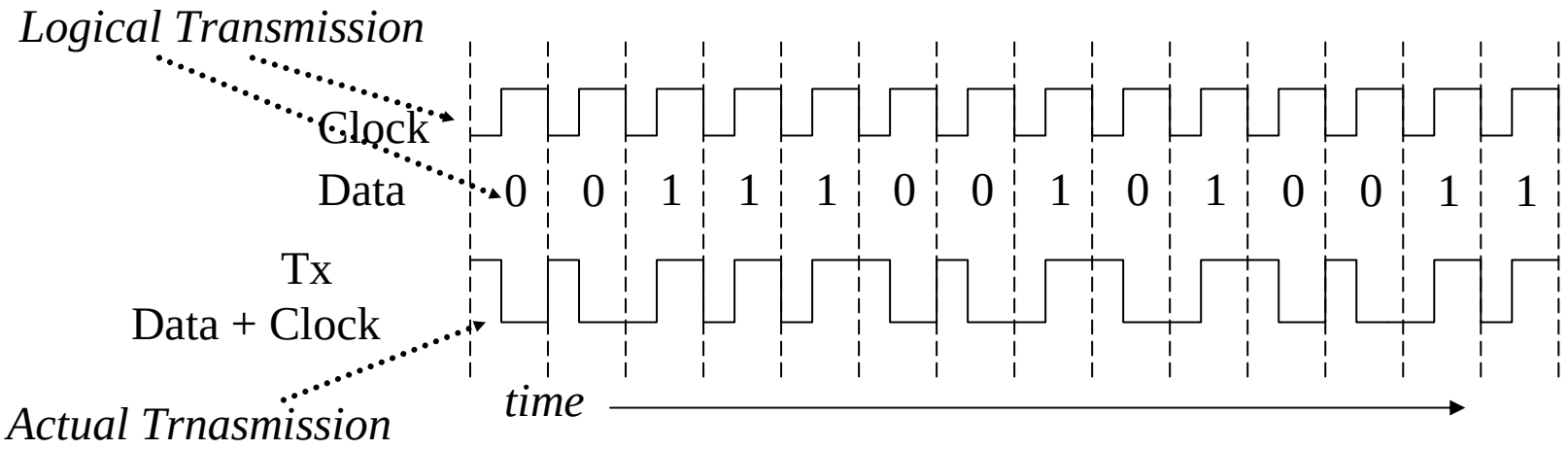


# UART System




Receiver/drivers do voltage conversions
Interface does serial to parallel, parallel to serial.

# Synchronous Serial Data Transfer



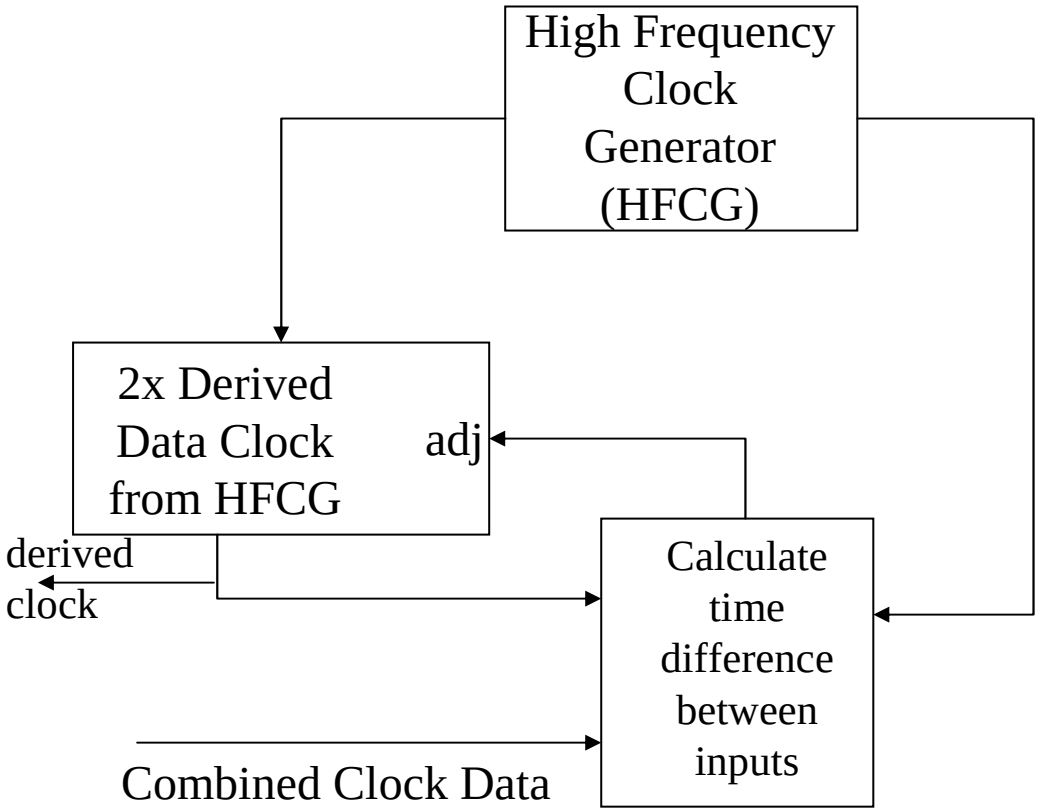

‘Manchester Encoding’ example – used on Ethernet

Data and clock sent combined on same wire
Multiple data items sent in long blocks (1000s of bits)
Fixed pattern of date (Pre-amble) at start of block allows identification of which rising edge marks the ‘beginning’ of a clock period from that which marks the ‘middle’ of clock period.

## **Phase Locked Loop** – digital example

**For interest only**

1. The time difference between the rising edges on the 2x derived clock and edges on the incoming data& clock is calculated on each incoming edge. The time difference is measured as the number of clock periods of the high frequency clock.

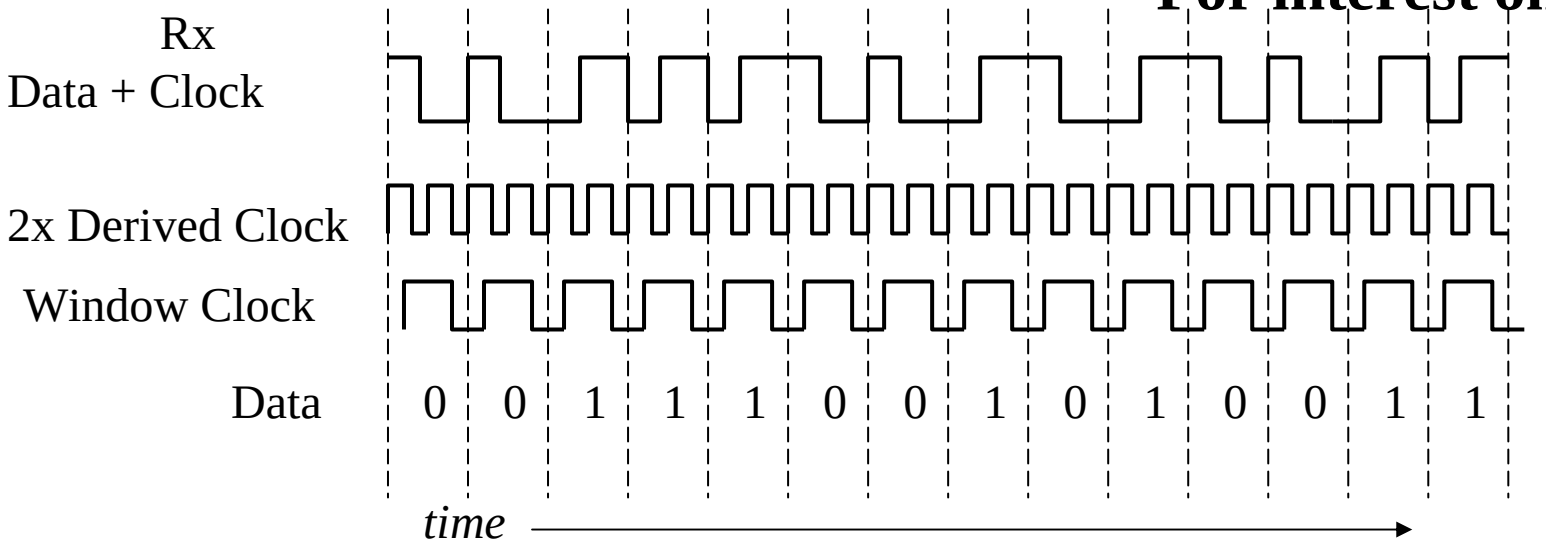


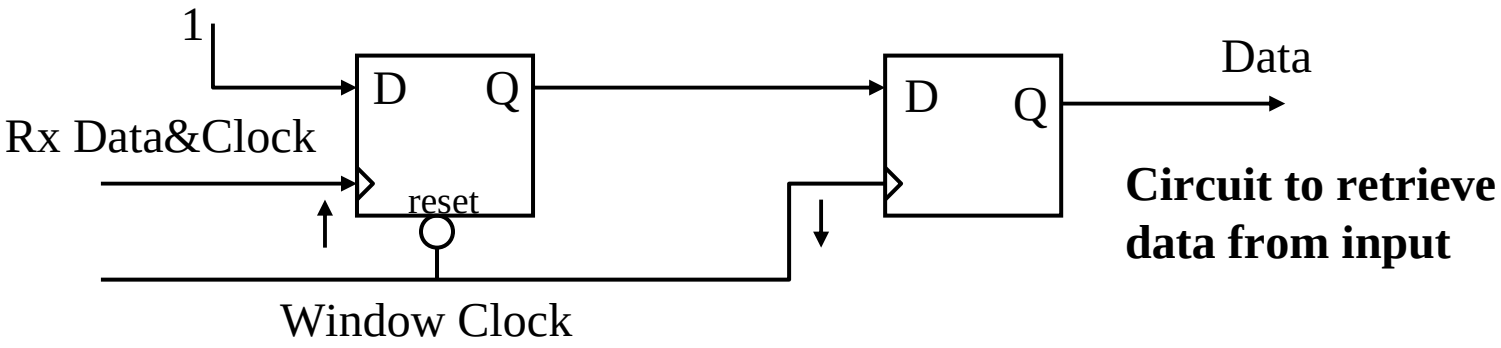

2. The calculated time difference is passed to the 2x clock unit to adjust the timing of the next clock edge to bring in line with the incoming edges.

Therefore, the 2x derived clock adjust its edges, and thus its frequency, to match the edges of the incoming data.

The 2x derived clock has to be approximately twice the incoming clock frequency to begin with. It is derived by dividing down from the High Frequency Clock

**For interest only**

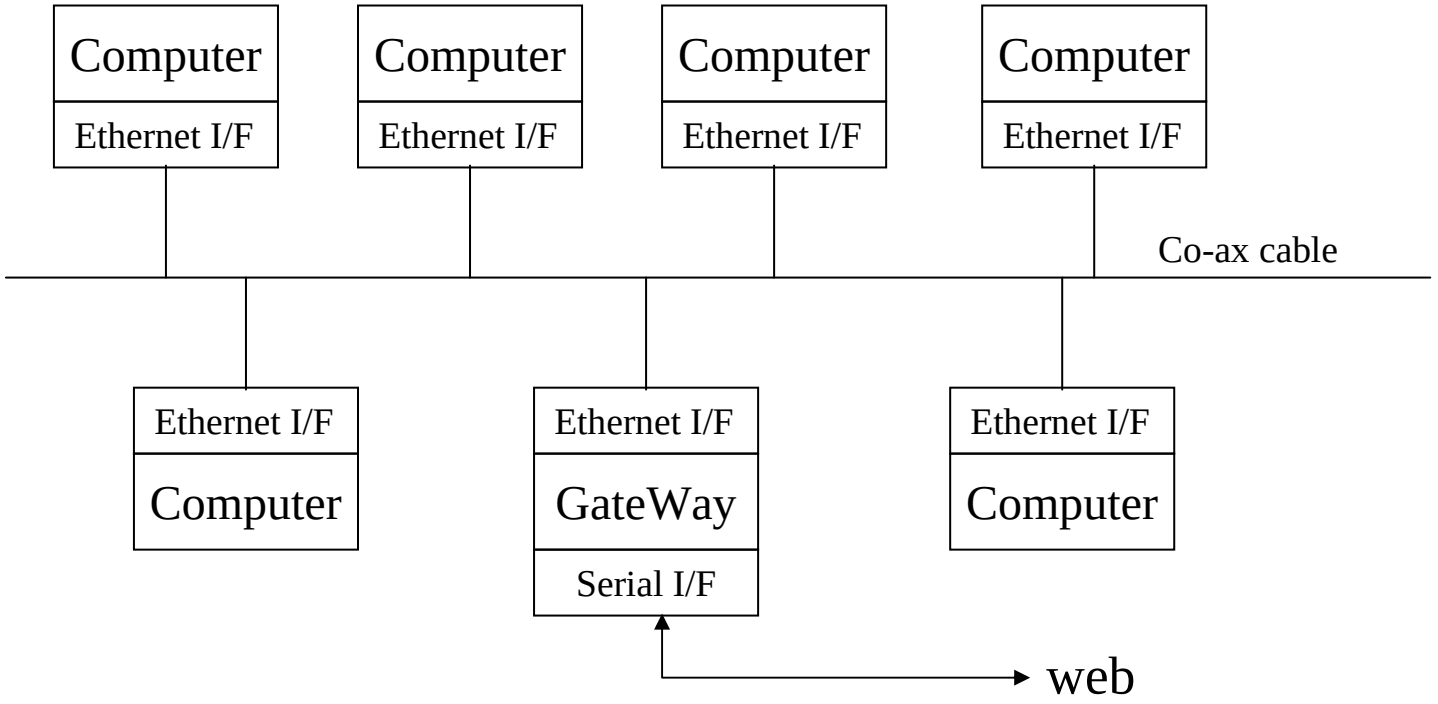


# Ethernet

## Synchronous Serial Transmission using Manchester encoding

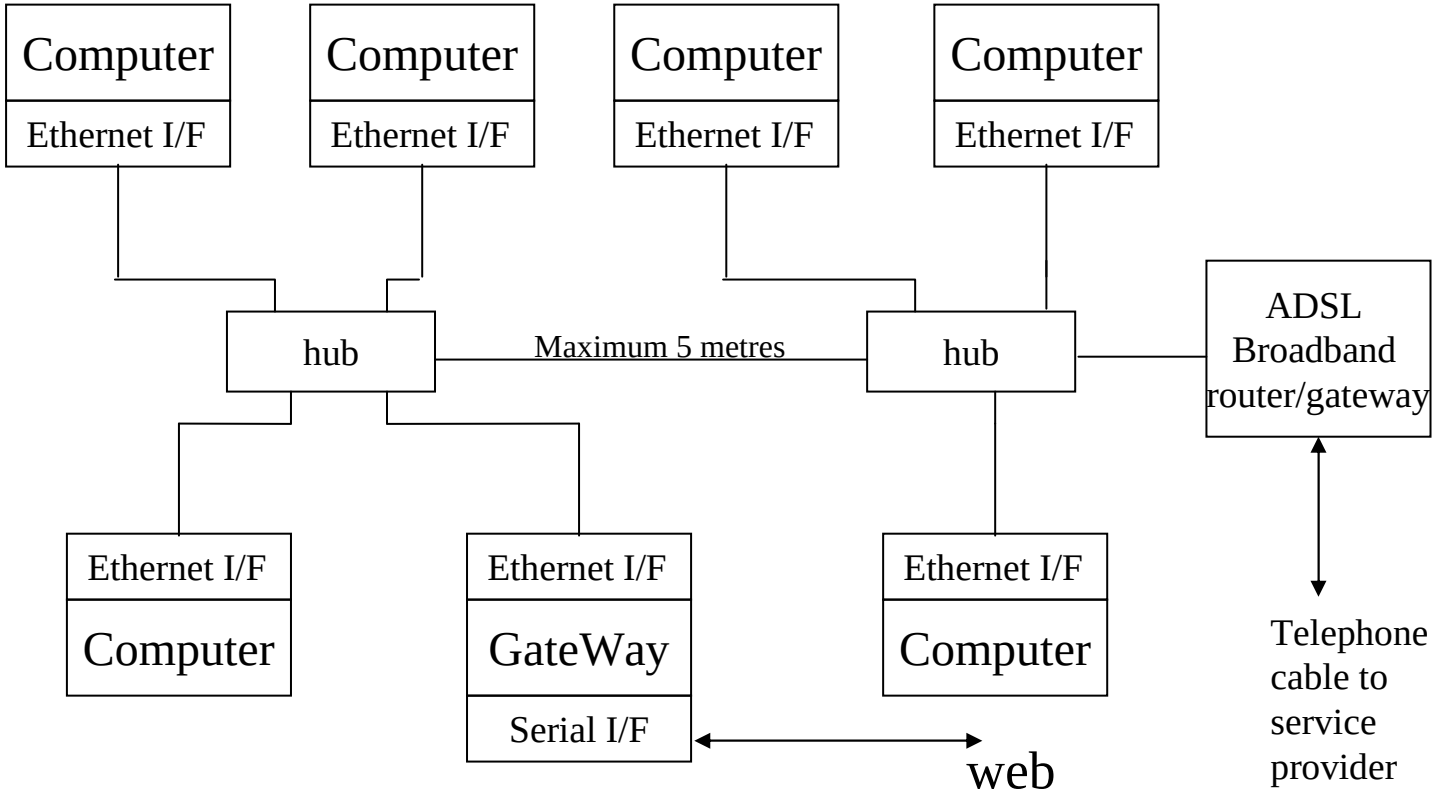
Original co-axial system – clock frequency 10MHz
Now twisted pair cable – 10Base5 (10 MHz over 500m)



# (Fast) Ethernet

**For interest only**

clock frequency 100MHz – hub based network – still CSMA/CD –100BaseT

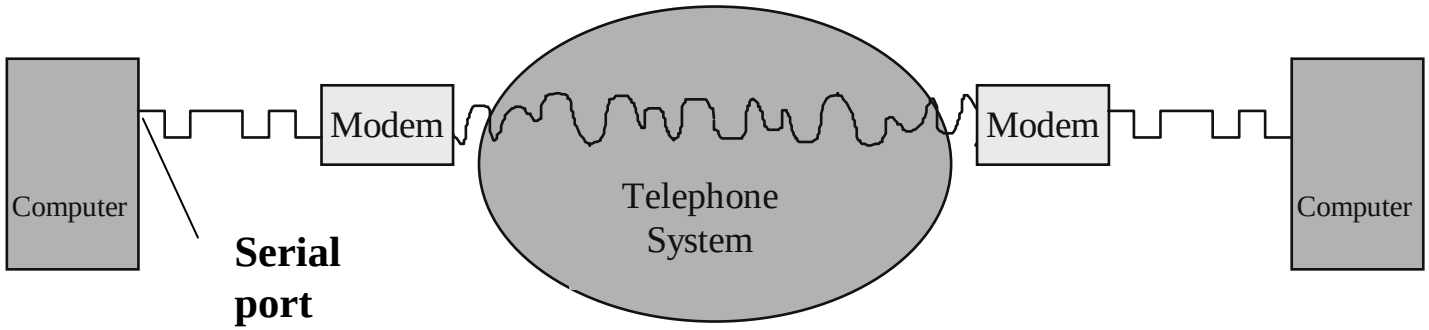


# Modems

- Long distances => wires must be hired from a telecommunications company
  - Simplest way - use an ordinary telephone connection.
- Telephone connections are optimised for voice
  - Range of frequencies carried is artificially restricted
    - Capacity (bits/sec) is proportional to range of frequencies, the “bandwidth”
  - Sharp transitions are distorted 
- Need equipment to encode bits to give a signal which has similar characteristics as voice.

# Modems [2]

- Modulation
  - –E.g. frequency modulation – one pitch for 1, another for 0
- “Modulator/Demodulator” or “Modem”
- Interface between computer and modem is standardised. - “RS-232C”.
  - –Available form PC “serial port”



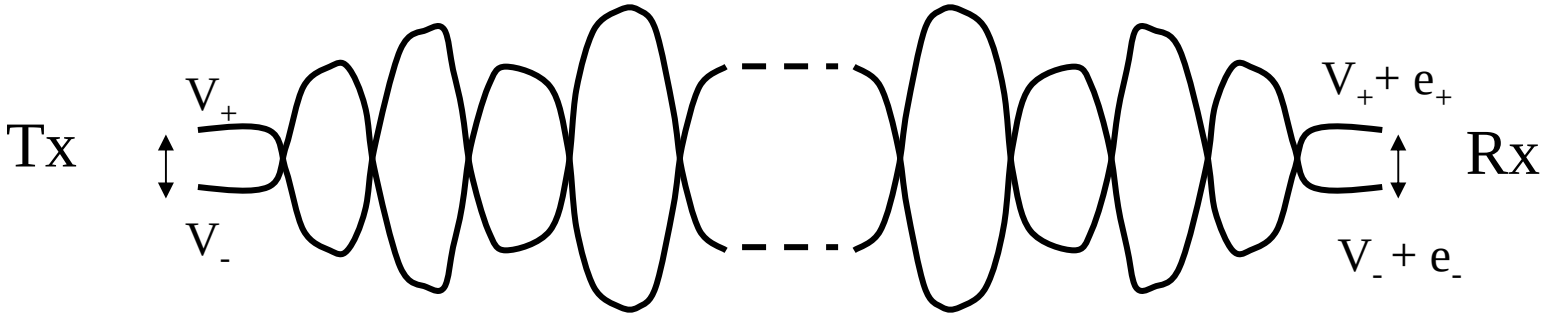
# Digital Transmission

- Purpose-built computer networks - digital throughout
  - No need for modems.
  - E.g. “Local Area Networks” (LANs) like Ethernet
- Modern telephone networks - digital internally;
  - Voice represented digitally - CODEC at the exchange
  - “Local loop” is analogue
  - Can be upgraded to a digital broadband by removing filters at receiver that cuts out high frequemcy
- ADSL (Broadband)
  - Digital over local loop –analogue representation of digital
  - Then direct to Internet

# How many bits per second?

| Network type | Bit rate | Time to transmit 100K bytes |
|---|---|---|
| Mobile phone + modem | 1.2 Kbps | 667 sec |
| Typical telephone connection + modems | 34.4 Kbps | 23.5 sec |
| ISDN channel | 64 Kbps | 12.5 sec |
| ADSL downlink | 512 Kbps | 1.6 sec |
| Ethernet LAN | 10 Mbps | .08 sec (= 80 ms) |
| “Fast” Ethernet (as per CS Dept.) | 100 Mbps | .008 sec (= 8 ms) |
| “Gigabit” Ethernet | 1 Gbps | .0008 sec (= 1.2 ms) |

Twisted Pair Cables used for high frequency signalling – ethernet, broadband




2 wires driven to opposite voltages at TX, e.g. +0.6V, -0.6V
Direction of voltage difference carries binary value.
Thus, Tx sets wires to $V_+$ and $V_-$

Rx measures voltage difference received.
Rx receives $V_+ + e_+$ and $V_- + e_-$ : *$e_+$, $e_-$ are noise signals*

Voltage difference = $(V_+ - V_-) + (e_+ - e_-)$ = applied voltage

For twisted pair, $e_+ == e_-$ to first order so they cancel on subtraction