Structured Hardware Design

Six lectures for CST Part Ia (50 percent).

Easter Term 2005.

(C) DJ Greaves.

Preface

There are a few more slides here than will be used in lectures. No Verilog is examinable: it is provided for reference use in part Ib. The first ten or so slides are revision of material from digital electronics.

At least 10 minutes or so of each lecture will be devoted to example material, including previous exam questions, for which there are no slides in this handout.

Books related to the course

Suggested books include:

Bignell & Donovan. *'Digital Electronics'* Delmar Publishers.

W.Ditch. *'Microelectronic Systems, A practical approach.'* Edward Arnold. The final chapters with details of the Z80 and 6502 are not relevant to this course.
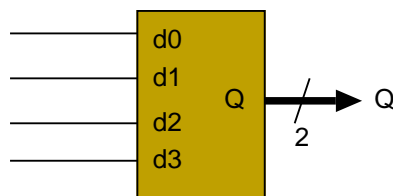
Floyd. *'Digital Fundamentals'* Prentice Hall International.

T.J. Stoneham. *'Digital Logic Techniques'* Chapman and Hall. This is a basic book and relates more to the previous course on Digital Electronics.

Randy H Katz. *'Contemporary logic design.'*
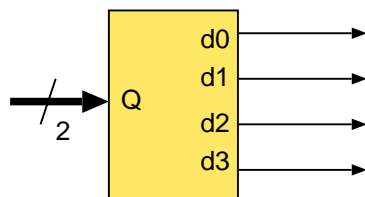
Encoder and Decoder
(Revision)

Priority Encoder

d0	d1	d2	d3	Q1	Q0
X	X	X	1	1	1
X	X	1	0	1	0
X	1	0	0	0	1
1	0	0	0	0	0

```
module priencoder(d, Q);
   output [1:0] Q;
   input [3:0] d;
   assign Q = d[3] ? 2'd3: d[2] ? 2'd2: d[1] ? 2'd1: 2'd0;
endmodule
```

Binary to Unary Decoder

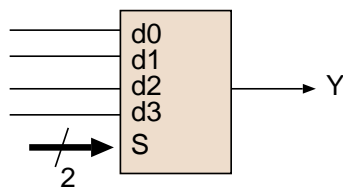| Q1	Q0	d0	d1	d2	d3
1	1	0	0	0	1
1	0	0	0	1	0
0	1	0	1	0	0
0	0	1	0	0	0

```
module decoder(Q, d);
   input [1:0] Q;
   output [3:0] d;
   assign d0 = (Q==2'd0);      assign d1 = (Q==2'd1);
   assign d2 = (Q==2'd2);      assign d3 = (Q==2'd3);
endmodule
```

4

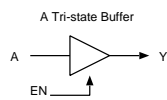# Multiplexor (Revision)

Multiplexor

d0	d1	d2	d3	S1	S0	Y
X	X	X	1	1	1	1
X	X	X	0	1	1	0
X	X	1	X	1	0	1
X	X	0	X	1	0	0
X	1	X	X	0	1	1
X	0	X	X	0	1	0
1	X	X	X	0	0	1
0	X	X	X	0	0	0

```
module multiplexor(d, S, y);
   input [1:0] S;
   input [3:0] d;
   output y;
   assign y = (S==2'd3) ? d[3]: (S==2'd2) ? d[2]:
              (S==2'd1) ? d[1]: d[0];
endmodule
```
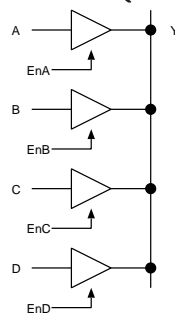
Distributed Multiplexor (Tri-State)

A Tri-state Buffer

Truth Table

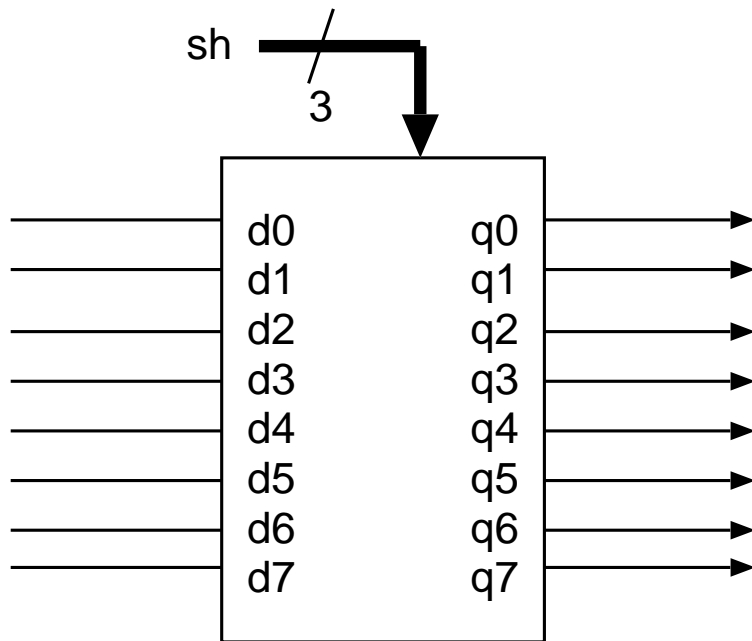| EN	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

Verilog: bufif(Y, A, en)
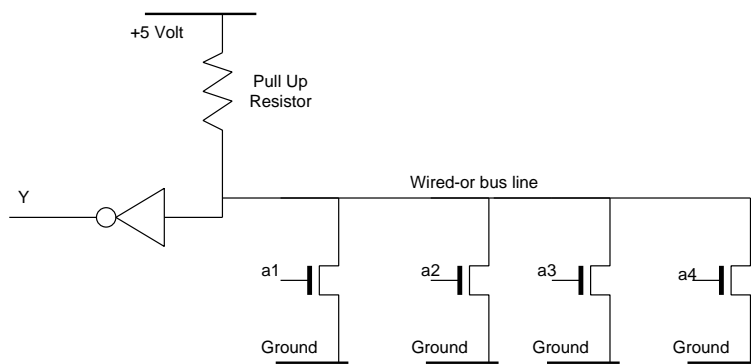
Tri-state wire must
be driven at one point
at a time only.

Makes a distribted multiplexor

Here only one bus wire is shown, but generally
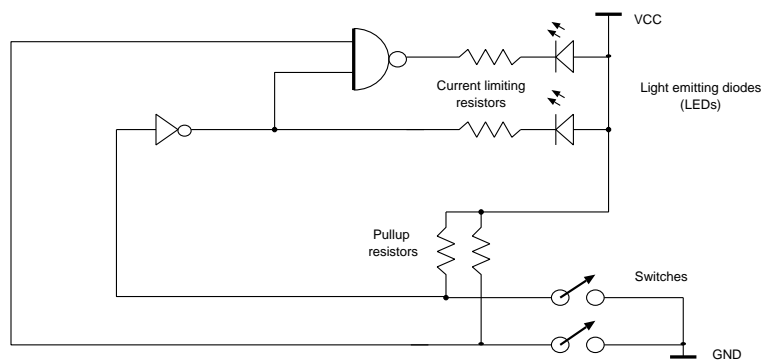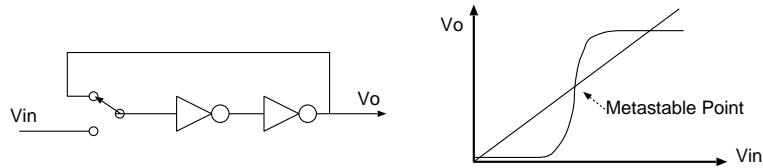32 or 64 wires are present in a tri-state bus

Barrel Shifter

sh —/— 3

d0	q0
d1	q1
d2	q2
d3	q3
d4	q4
d5	q5
d6	q6
d7	q7

Open Drain (open collector)

Distributed OR gate.

Leds and Switches Interfacing

VCC

Current limiting
resistors

Light emitting diodes
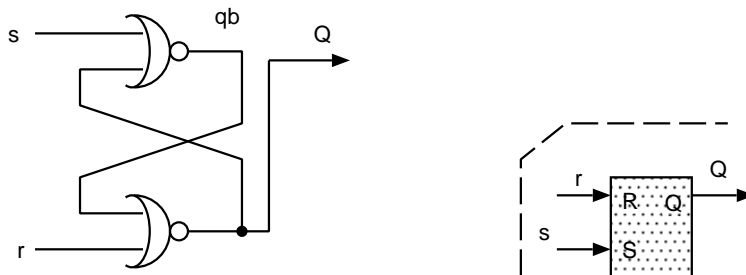(LEDs)

Pullup
resistors

Switches

GND

Bistable Revision

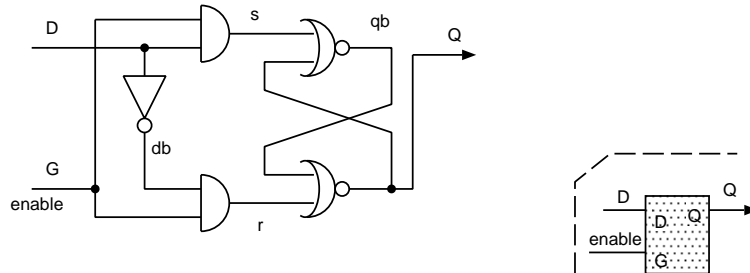The bistable is the most basic electronic store
for one bit.
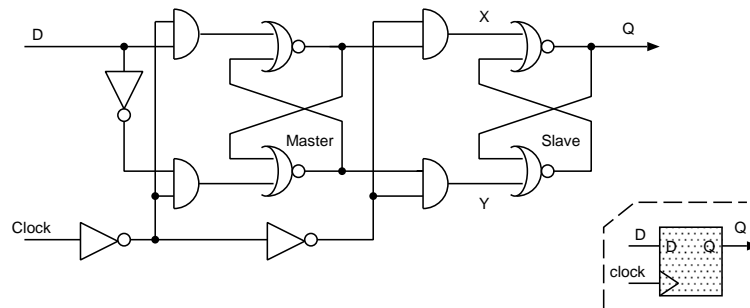

Adding a pair of inputs makes an RS latch
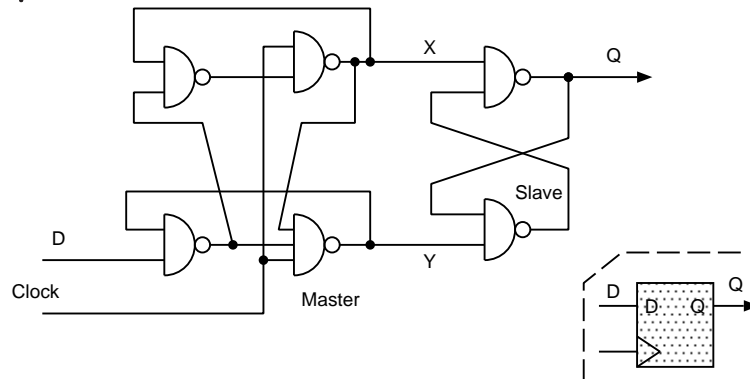
Flip–Flop Revision

Making a transparent latch from an RS latch:

Putting two together we get the D-type:

A more optimal circuit:

In this course, we go **upwards** from the D-type towards systems.

Adding a Clock Enable and Synch Reset

Adding a clock enable
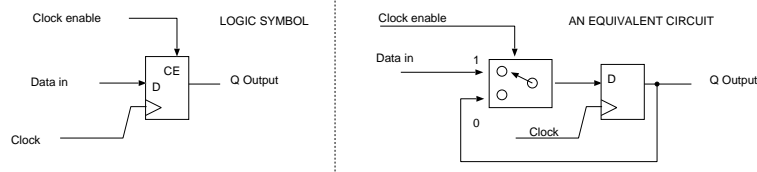

```
always @(posedge clk) q <= (clock_en) ? data_in: q;
```

alternatively

```
always @(posedge clk) begin
        if (clock_en) q <= data_in;
        ...
        end
```

Adding a Synchronous Reset
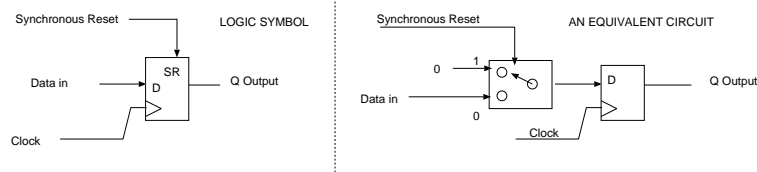

```
always @(posedge clk) q <= (sr) ? 0:data_in;
```

A Broadside Register
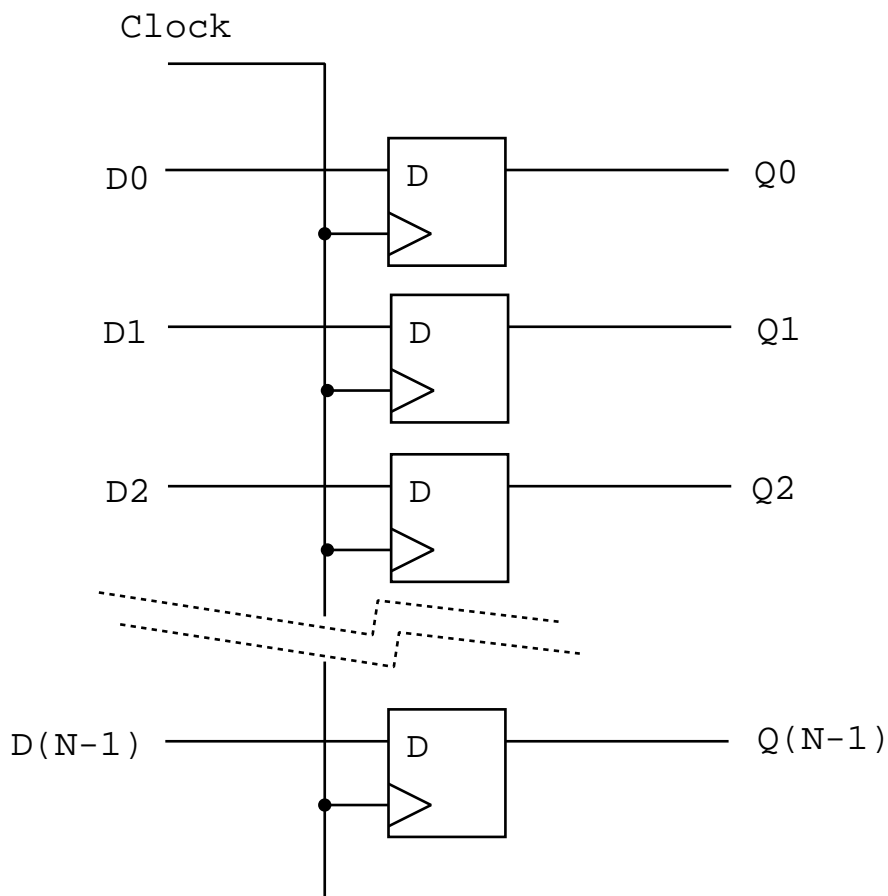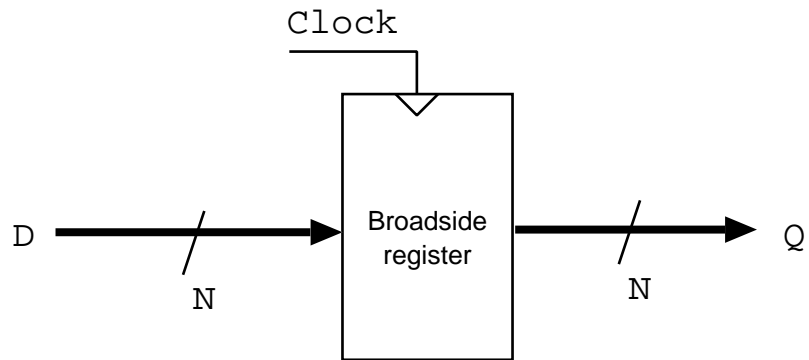
A broadside register of N bits is made out of N D-types with a commoned clock input. It can hold 2^N different values.

A Broadside Register – Verilog
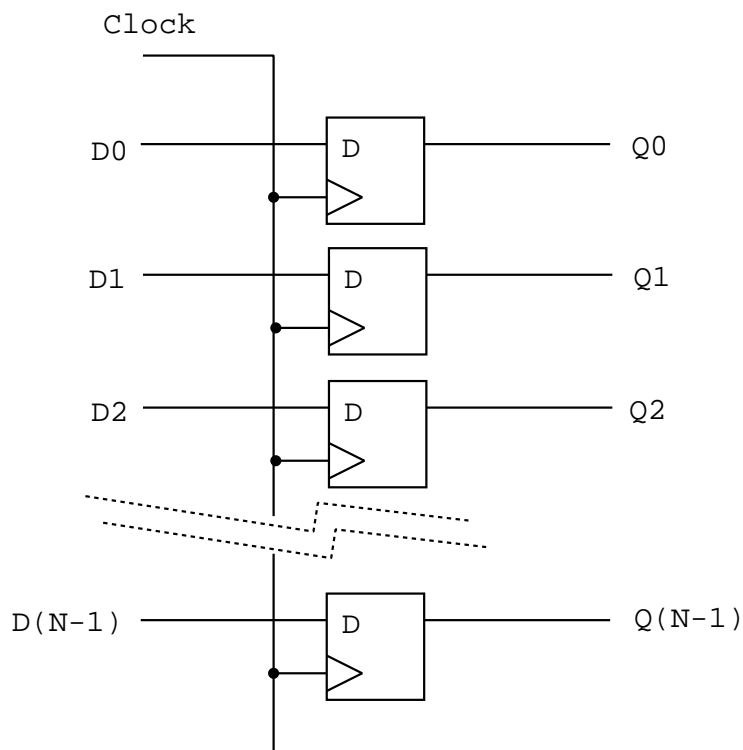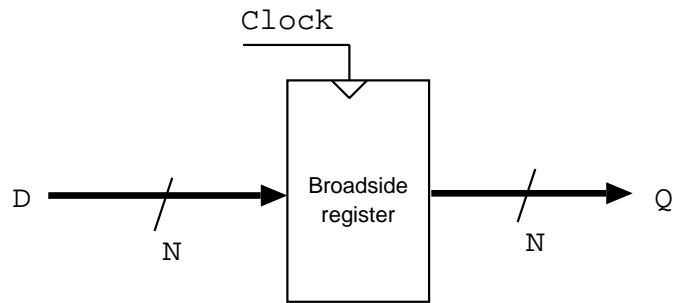

```
parameter N = 8;
reg [N-1:0] br_q;
always @(posedge clk) begin
     br_q <= data_in;
     end
```

A broadside two-to-one multiplexor
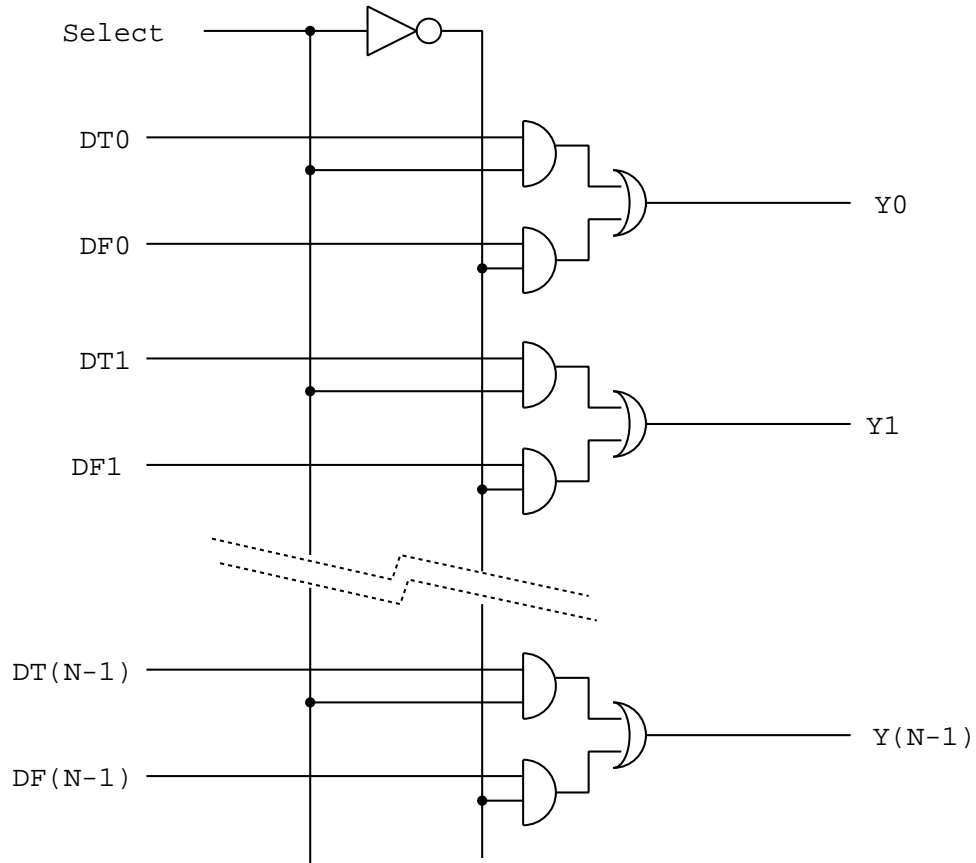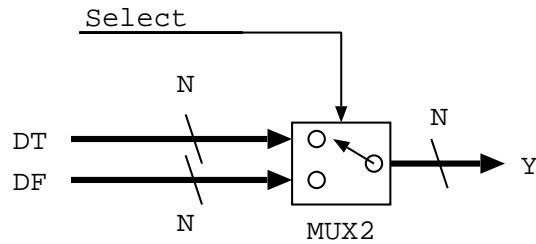

```
wire [N-1:0] Y, DT, DF;
assign Y = (Select) ? DT: DF;
```

Shift Registers

An n-bit shifter
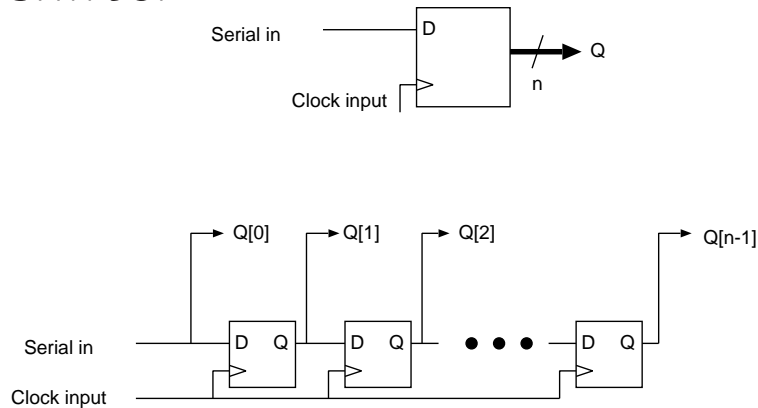
Adding a parallel load
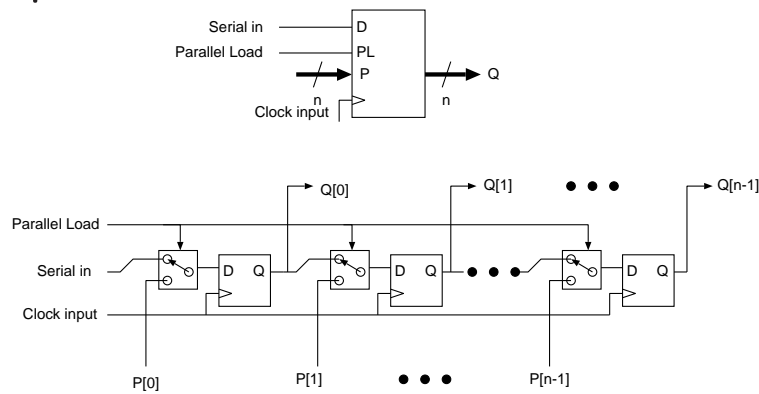

```
parameter N = 8;
reg [N-1:0] Q;
always @(posedge clk) begin
    Q <= (PL) ? P: (Q << 1) | D;
    end
```

Synchronous Datapath – A Fragment
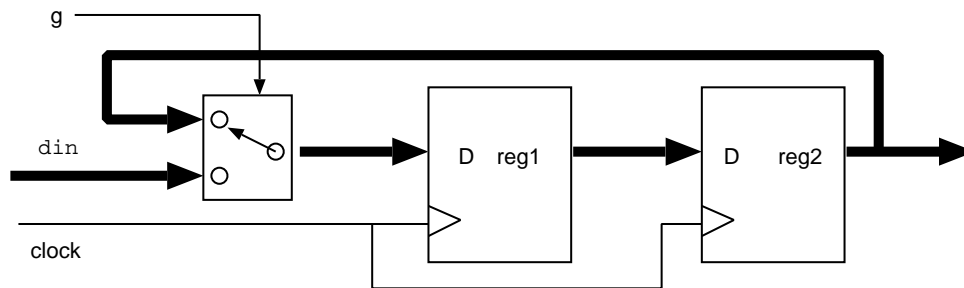
We swap the values between a pair of registers if the guard is false, but a broadside multiplexor introduces a new value into the loop when the guard is enabled.

```
reg [7:0] reg1, reg2;
  always @(posedge clock) begin
        reg1 <= (g) ? din: reg2;
        reg2 <= reg1;
        end
```

A Dual-port Register File

Example A=4, N=16.
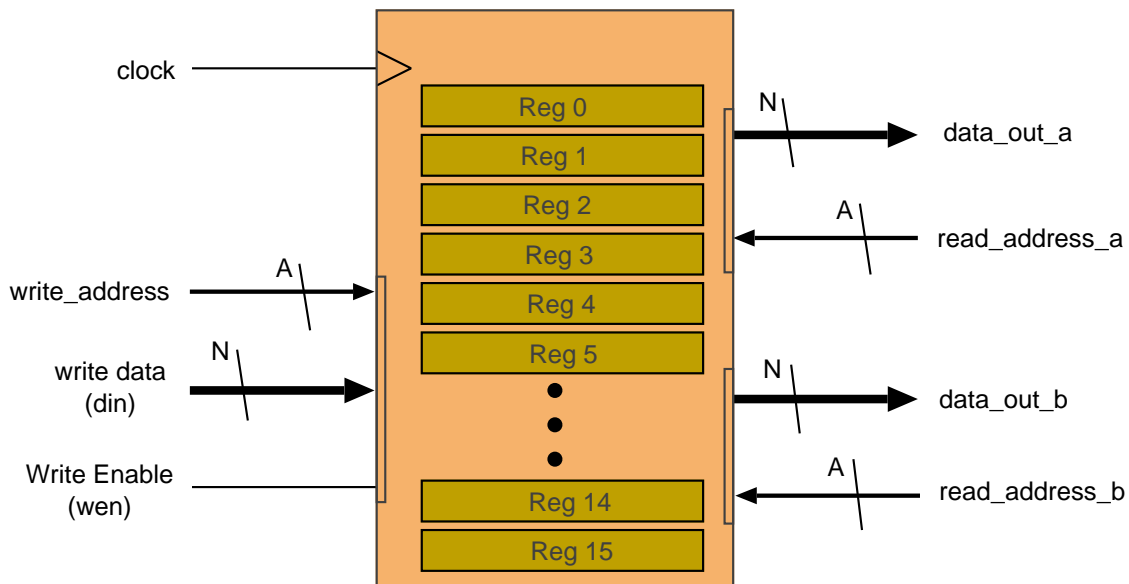

```verilog
// Verilog for a dual-read ported register file.
 input [3:0] write_address, read_address_a,
            read_address_b;
 reg [7:0] regfile [15:0]
 always @(posedge clk) begin
      if (wen) regfile[write_address] <= din;
      end

 // Combinational read (synchronous perhaps is more common).
 wire [7:0] data_out_a = regfile[read_address_a];
 wire [7:0] data_out_b = regfile[read_address_b];
```

Ex: Draw out the full circuit at the gate level!

Read/Write Memory (RAM)

Address In
Addr
A
Enable Input
(active low)
E
RAM
Data
Data In and Out
N
Read or write
mode select
R/Wb

Read Cycle - Like the ROM

Read or write
mode select

Enable Input
(active low)

Address In

Data Bus — High-Z Valid data High-Z
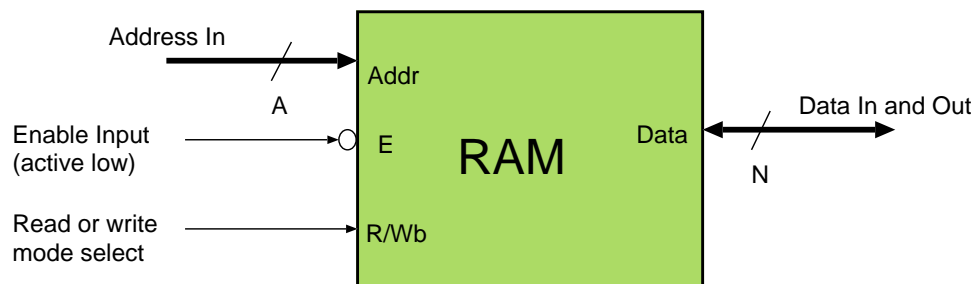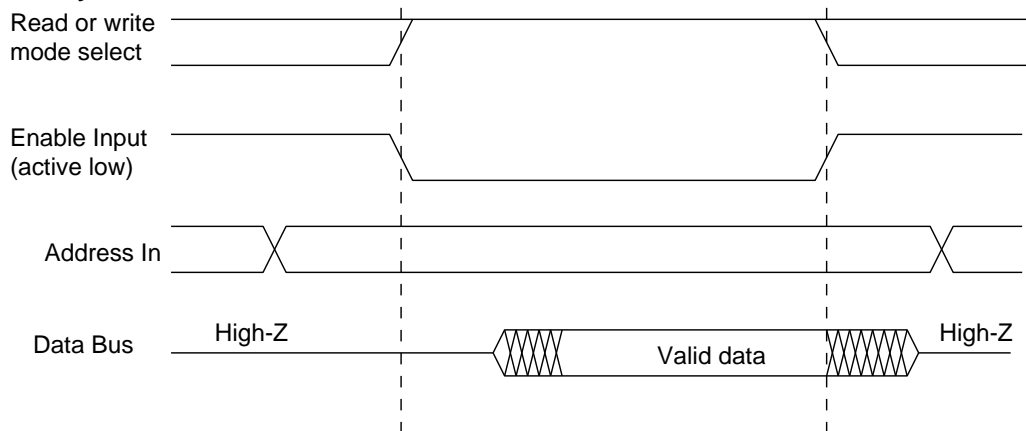
Write Cycle - Data stored internally

Read or write
mode select

Enable Input
(active low)

Address In

Data Bus — High-Z High-Z

Data must be valid
here to be stored.
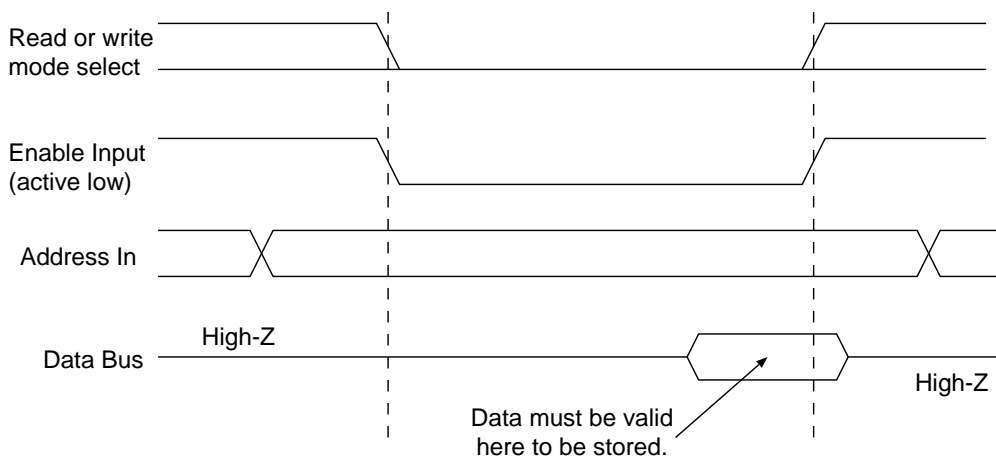
Each data bit internally stored in an RS latch.

Read Only Memory (ROM)

Address In —— A → Addr

Data —— N → Data Out

Enable Input (active low) ——○ E

ROM PROM or EPROM

The ROM contents are placed inside during manufacture or during field programming.

The ROM takes A address bits named A0 to A<A-1> and produces data words of N bits wide. For example, if A=5 and D=8 then the ROM contains 2**5 which is 32 locations of 8 bits each. The address lines are called A0, A1, A2, A3, A4 and the data lines D0, D1, ... D7

Access Time

Enable Input (active low)

Address In

Data Out   High-Z   Valid data   High-Z

Ouput Turnon Time

The ROM's outputs are high impedance unless the enable input is asserted (low). After the enable is low the output drivers turn on. When the address has been stable sufficiently long, valid data from that address comes out.

MASKED PROGRAMMED means contents inserted at time of manufacture.

FLASH PROM uses static electricity on floating transistor gates.

Non-volatile Technologies

Name	Persistence	Read Speed	Write Rate
RAM	Volatile	Same as SRAM	Same as SRAM
BB-RAM	Non-volatile	Same as SRAM	Same as SRAM
Mask PROM	Non-volatile	Same as SRAM	Not possible
EPROM	Non-volatile	Same as SRAM	10 us/byte
Sn-W PROM	Non-volatile	Same as SRAM	10 us/byte
EAROM	Non-volatile	Same as SRAM	10 us/byte

Name	Erase Time	Comment
RAM	not needed	
BB-RAM	not needed	Battery Life
Mask PROM	Not Possible	
EPROM	20 Mins	Needs UV window
Sn-W PROM	Not possible	
EAROM	100 ms/block	write cycle limit

Year 2000 onwards, Flash and memristive technology may start to dominate ...

Memory Banks

A17..16

A15..1 A D ce
A15..1 A D ce
A15..1 A D ce
A15..1 A D ce

A15..1 A D ce
A15..1 A D ce
A15..1 A D ce
A15..1 A D ce

D7..0
8
D15..8

8 ROM DEVICES

EACH ROM DEVICE IS
32768 BYTES CAPACITY

BANK ORGANISATION
128K locations of 16 bits
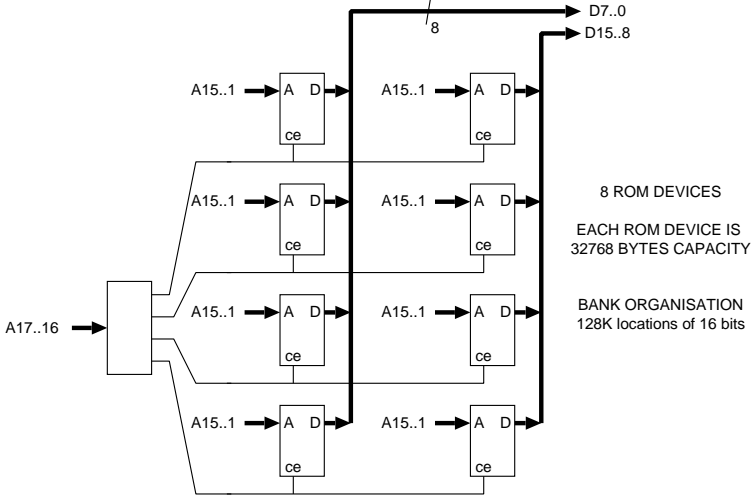
Unlike the edge-triggered flip-flop, the transparent latch
passes data through in a transparent way when its enable
input is high. When its enable input is low, the output stays
at the current value.
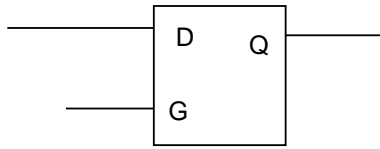
Transparent latch
schematic symbol

Transparent latch implemented from gates.

output
enable

Data

WE*

CE*

Binary to unary decoder

Address Input

Synchronous FIFO Memory

DRAM

Multiplexed Address In → MAddr

A

Row Address Strobe (RAS) → RAS   **DRAM**

Col Address Strobe (CAS) → CAS   Data → Data In and Out

N

Read or write mode select → R/Wb
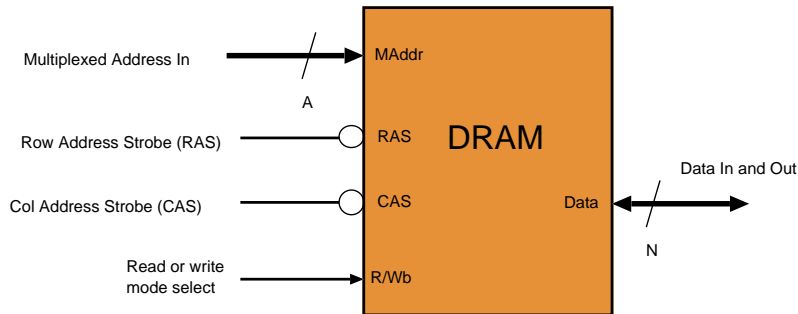
Read Cycle (write is similar)

Read or write mode select

Row Address Strobe (RAS)

Col Address Strobe (CAS)

Multiplexed Address   Row Address   Col Address

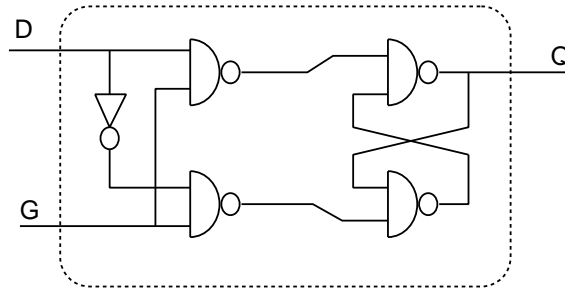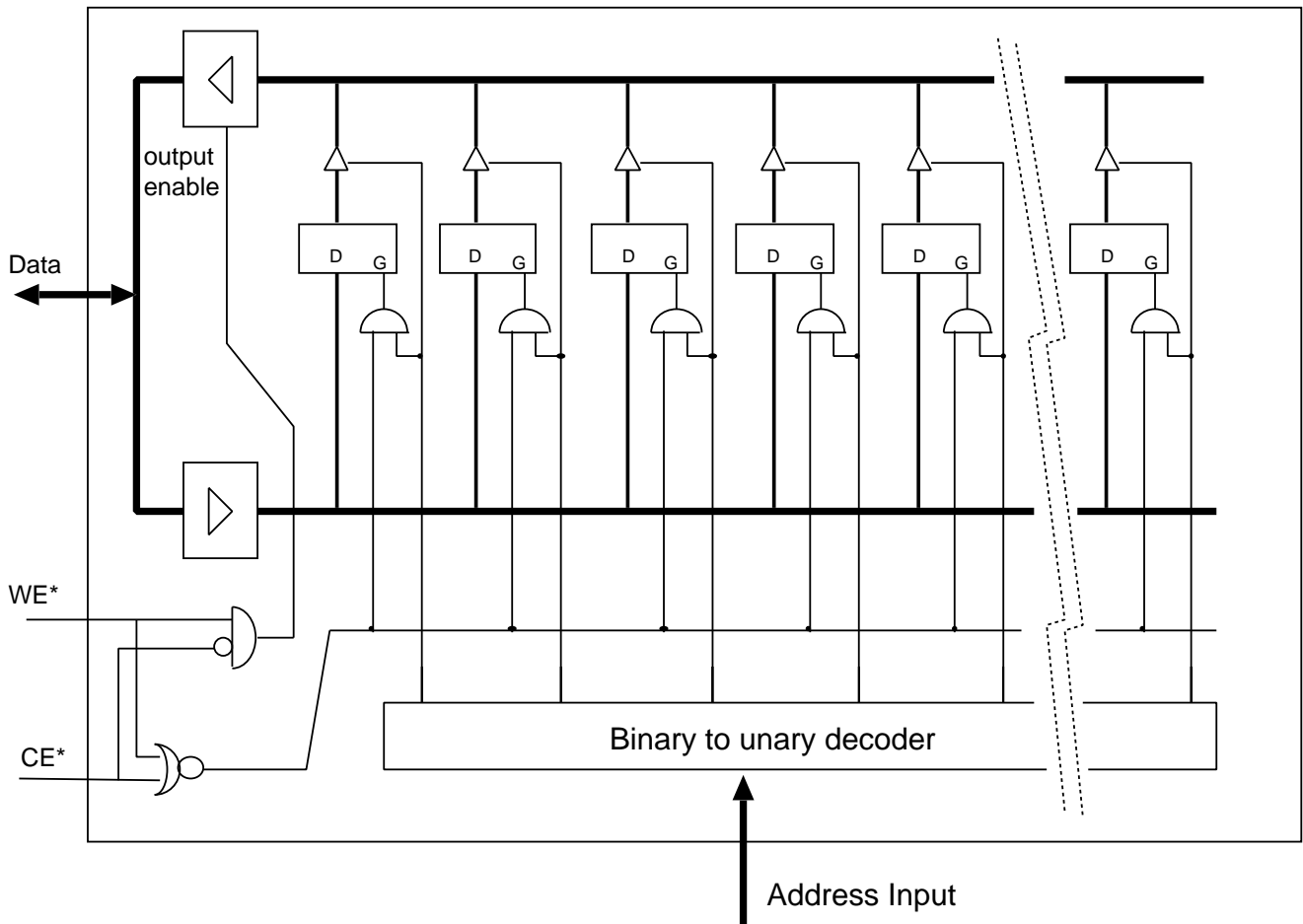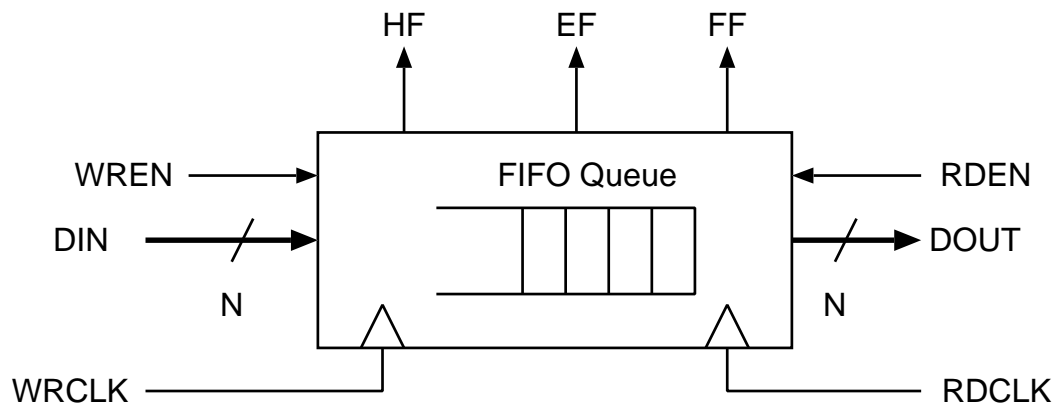Data Bus   High-Z   Valid data   High-Z
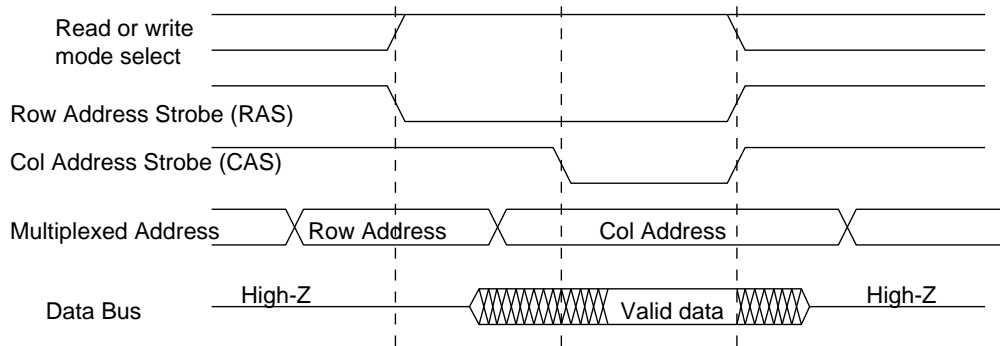
A DRAM has a multiplexed address bus and the address is presented in two halves, known as row and column addresses. So the capacity is 4**A x D. A 4 Mbit DRAM might have A=10 and D=4.

When a processor (or its cache) wishes to read many locations in sequence, only one row address needs be given and multiple col addresses can be given quickly to access data in the same row. This is known as 'page mode' access.

EDO (extended data out) DRAM is now quite common. This guarantees data to be valid for an exteneded period after CAS, thus helping system timing design at high CAS rates.

Refresh Cycle - must happen sufficiently often!

Row Address Strobe (RAS)

Col Address Strobe (CAS)
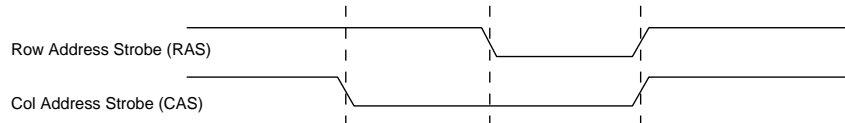
No data enters or leaves the DRAM during refresh, so it 'eats memory bandwidth'.
Typically 512 cycles of refresh must be done every 8 milliseconds.

Modern DRAM has a clock input at 200 MHz and transfers data on both edges.

Crystal oscillator clock source

1M

33pF

33pF

Ground

RC oscillator clock source

R

Vo

Shmitt Inverter

C

Ground

Vin

Schematic
Symbol
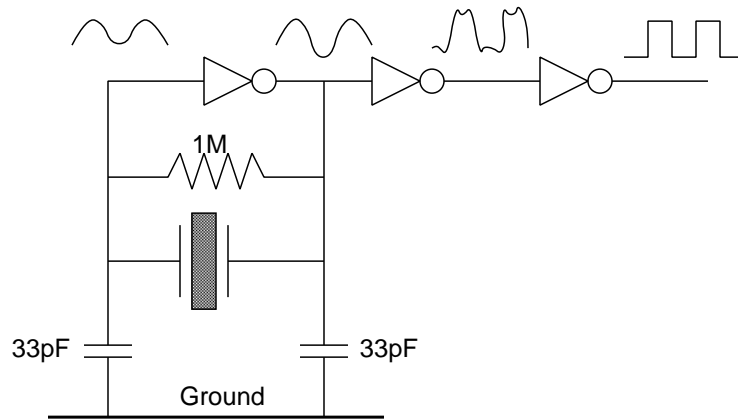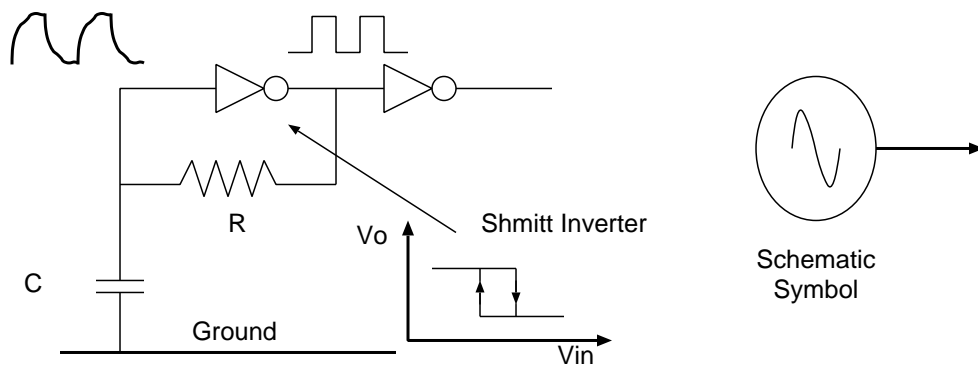
Clock multiplication and distribution

Outside the chip
Inside the chip
PLL Circuit

VCO 1000MHz

External clock input

100 MHz

Divide 10

Clock distribution H tree

H tree layout

Power-on reset

Supply

R

Active low
Reset output

Vo

Vi

C

Ground
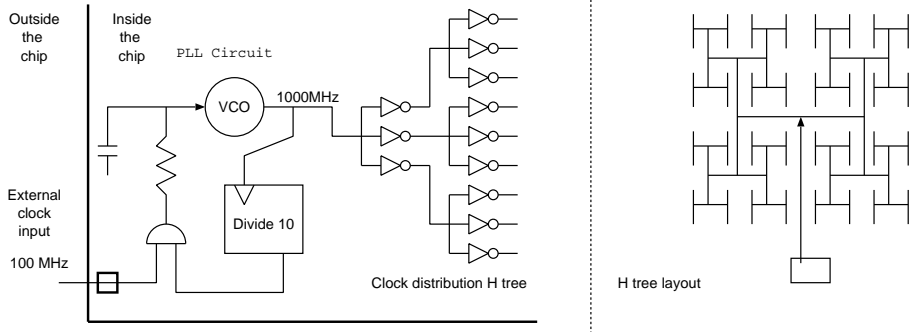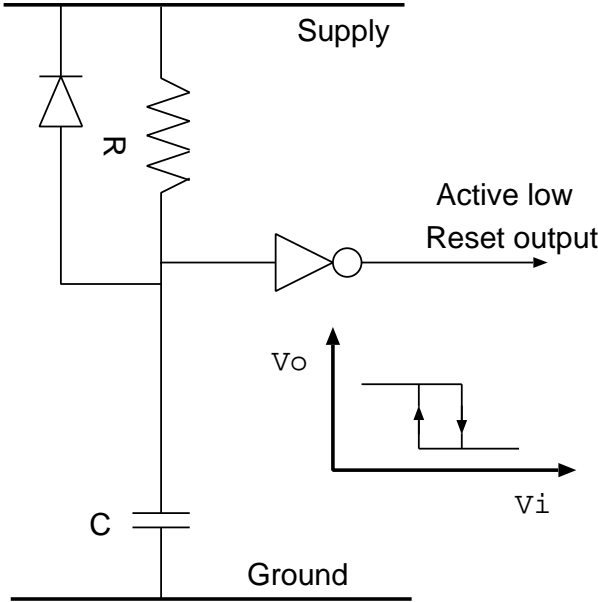
Driving a heavy current or high-voltage load

High Voltage Supply

Back
EMF
protection
diode

Load may be
directly connected
or driven through a
mechanical relay

Control input

Power
MOSFT
transistor

Ground
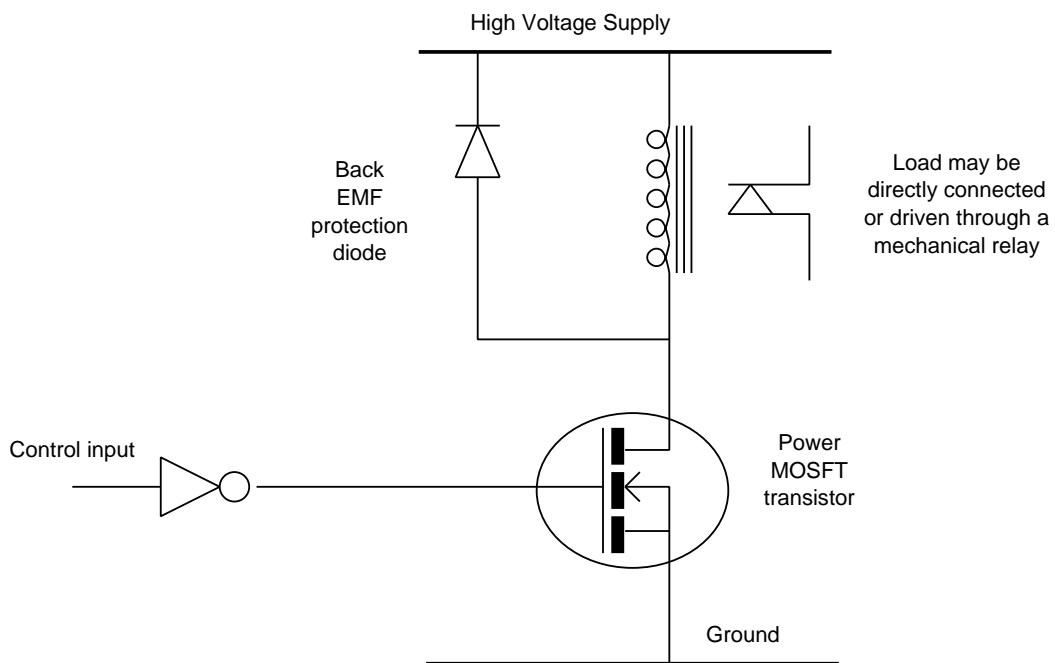
Transistor active area could be 1 square centimeter.

Debouncer circuit for a double-throw switch

+5Volt supply rail

Pullup
Resistors

Switch

A

B

Gnd

Output

Bounces

A

B
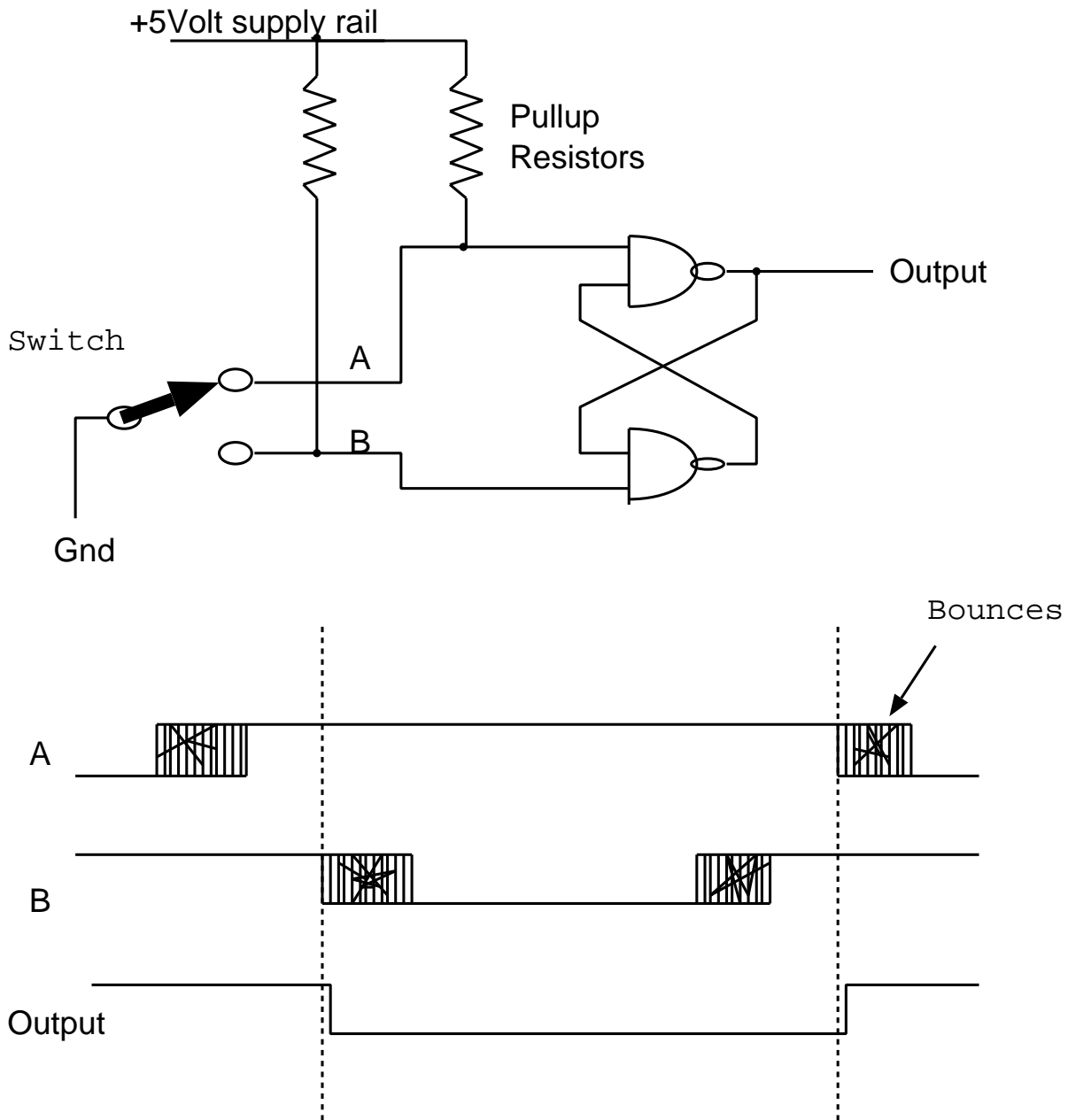
Output

ALU and Flags Register
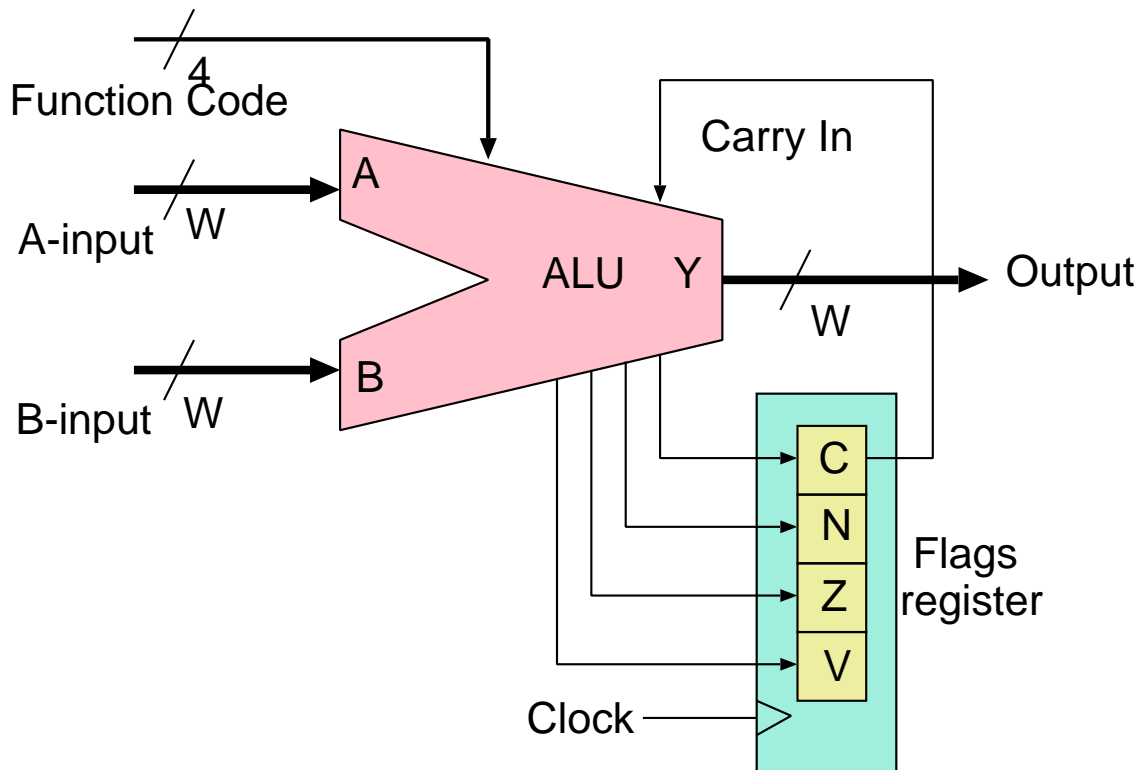

```
input [7:0] A, B, fc;
output [7:0] Y;
output C, V, N, Z;

always @(A or B or fc)
    case (fc)
     0:  { C, Y } = { 1'b0, A }; // A
     1:  { C, Y } = { 1'b0, B }; // B
     2:  { C, Y } = A+B;         // A+B
     3:  { C, Y } = A+B;         // A+B
     4:  { C, Y } = A+B+cin;     // A+B+Carry in
     5:  { C, Y } = A-B          // and so on
     ...
    endcase

assign Z = (Y == 0);   assign N = y[W-1];
```

ALU and Register File
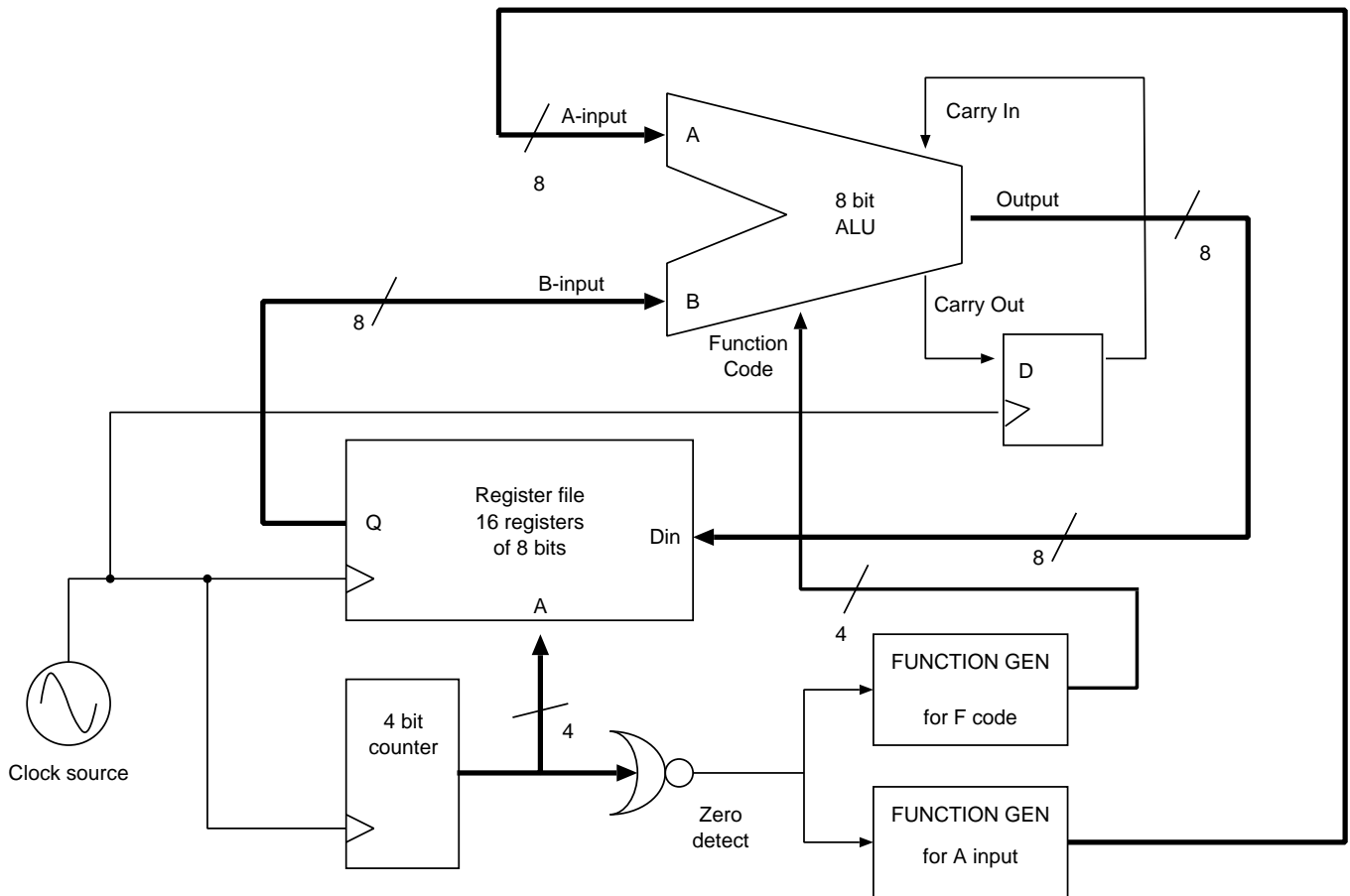
An example structure using an ALU and register file.

Ex: Program the ROM function generators to make one large counter out of the whole register file.

Multiplier

Flash multiplier - combinatorial implementation (e.g. a Wallace Tree).
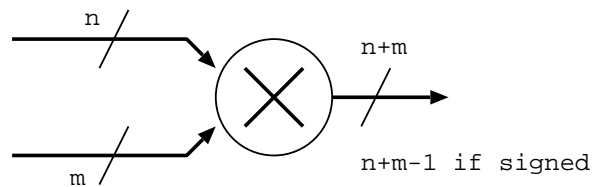
Sequential Long Multiplication

```
RA=A
RB=B
RC=0
while(RA>0)
{
  if odd(RA) RC=RC+RB;
  RA = RA >> 1;
  RB = RB << 1;
}
```

Microarchitecture for a Long Multiplier

MSI Schematic Symbol
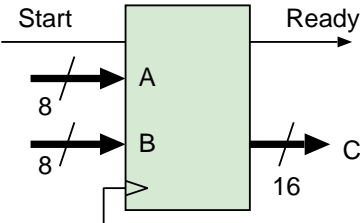
One possible internal structure (showing datapath/microarchitecture)
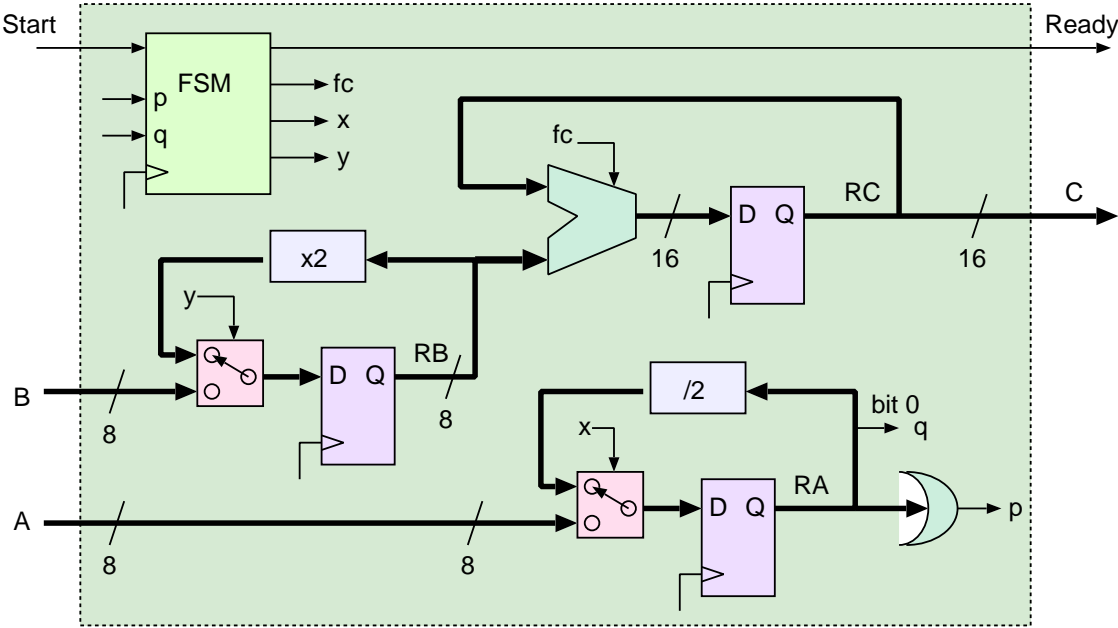
Booth's multiplier

Booth does two bits per clock cycle:

```
(* Call this function with c=0 and carry=0 to multiply
   x by y. *)
fun booth(x, y, c, carry) =
    if(x=0 andalso carry=0) then c else
let val x' = x div 4
    val y' = y * 4
    val n  = (x mod 4) + carry
    val (carry', c') = case (n) of
      (0) => (0, c)
     |(1) => (0, c+y)
     |(2) => (0, c+2*y)
     |(3) => (1, c-y)
     |(4) => (1, c)
    in booth(x', y', c', carry')
    end
```

Ex: Design a micro-architecture consisting of an ALU and register file to implement Booth. Design the sequencer too.

Logic Symbol

System Clock

Reset Input

Interrupt Request

R

I

Microprocessor

Data

Address

Opreq

R/Wb

W

N

A

Operation
Request

Read/Notwrite

Wait
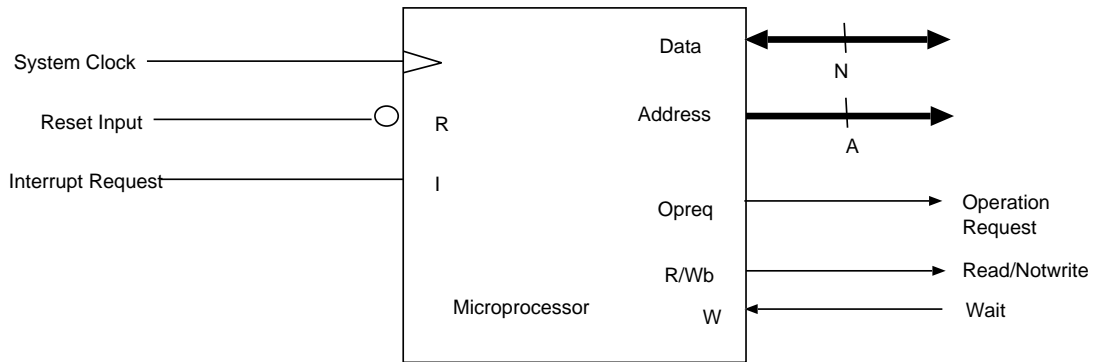
Internal Structure Block Diagram

Addresses   Write

System
Clock

Clock

Dual Port
Register
File

Function code

Load or Store

ALU

MUX

Operation Request

Read/notwrite

Data Bus

Bus Control

Execution Unit

Control Unit

Instruction
Register

IR   Clock

Clock

OPERAND EA

Execution address
incrementor

Instruction
Decoder

Mux 2

Reset

PC   Clock

MUX2

Reset

Control Wires To
All Other Sections

Program
Counter

Address Bus

34

Example of memory address decode and simple LED and switch interfacing for programmed IO (PIO) to a microprocessor.

A D8/A16 Microcomputer

Reset

Clock

Execution Unit
(Register File + ALU)

Data bus
(8 bits)

D0-7

Control Unit
(including PC)

Microprocessor

Address bus
(16 bits)

A15

A14

A13

Memory Map
decoder circuit.

The 'glue' logic

Often a PAL
single chip device.

Interrupt

Main Memory

RAM

D0-7

Memory

Static RAM

16 kByte

A0-13

wen

ren

enb

RAM_ENABLE_BAR

ROM
for booting

D0-7

1 K Byte ROM
Read Only Memory

A0-9

ren

enb

ROM_ENABLE_BAR

I/O Device

D0-7

UART
Serial Port

wen

ren

A0-2

enb

UART_ENABLE_BAR

RS232 serial connection

Memory Address Mapping


```
-------   -----     --------------------
Start     End       Resource
-------   -----     --------------------
0000      03FF      EPROM
0400      3FFF      Unused images of EPROM
4000      7FFF      RAM
8000      BFFF      Unused
C000      C001      Registers in the UART
C002      FFFF      Unused images of the UART
-------   -----     --------------------
```

```verilog
module address_decode(abus, rom_cs, ram_cs, uart_cs);
   input [15:14] abus;
   output rom_cs, ram_cs, uart_cs);

   assign rom_cs = (abus == 2'b00);  // 0x0000
   assign ram_cs = (abus == 2'b01);  // 0x4000
   assign uart_cs = !(abus == 2'b11);// 0xC000
endmodule
```

PC Motherboard, 1997 vintage

PSU

KYBD

Main memory DRAM

SIMM 4

SIMM 3

SIMM 2

SIMM 1

CACHE RAM

USB

COM1

COM2

PRINTER

PCI1

PCI2

PCI3

IDE-1

IDE-2

Floppy

BIOS ROM

IDE & Floppy

Cache Control

General glue

Pentium CPU

ISA
16 BIT
SLOTS

BATTERY

Clock

Regulator

Parallel Port

Parallel Port Interface Logic

CPU BUS SIDE

Address

Data

r/wbar

/cs

Read/Writebar
device select

Strobe

Acknowledge

Busy

Parallel Data

D25 Parallel (Centronix) Port

Parallel Data — Valid Data For Transfer To Peripheral Device

Strobe_bar

Acknowledge

Busy

Ready for next data

Flow control: New data is not sent while
the busy wire is high.

Serial Port (UART)

Address

Data

Int

Read/Writebar

chip select

Interrupt

r/wbar

/cs

Serial Output

Serial Input

Voltage convertors

Baud Rate Generator

25-Way D connector for Serial Port.

Most computers just use a 9 way connector these days.

LOGIC 1

LOGIC 0

DO D1 D2 D3 D4 D5 D6 D7

Start Bit (zero)

Stop Bit (one)

Keyboard and/or PS/2 port

+5 Volt

Fuse

Power wire

Clock wire

Data Wire

Ground wires

PS/2
Connector

Ground

4   3

5   2

6   1

PS/2 Keyboard/Mouse Cable
1. Clock
2. Ground
3. Data
4. Spare
5. Power +5Volts
6. Spare
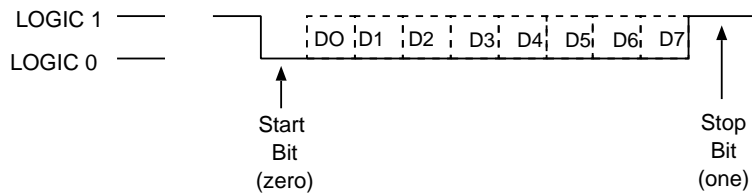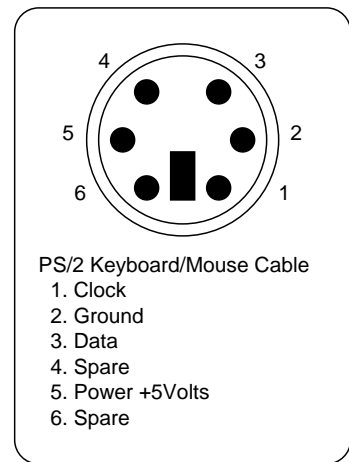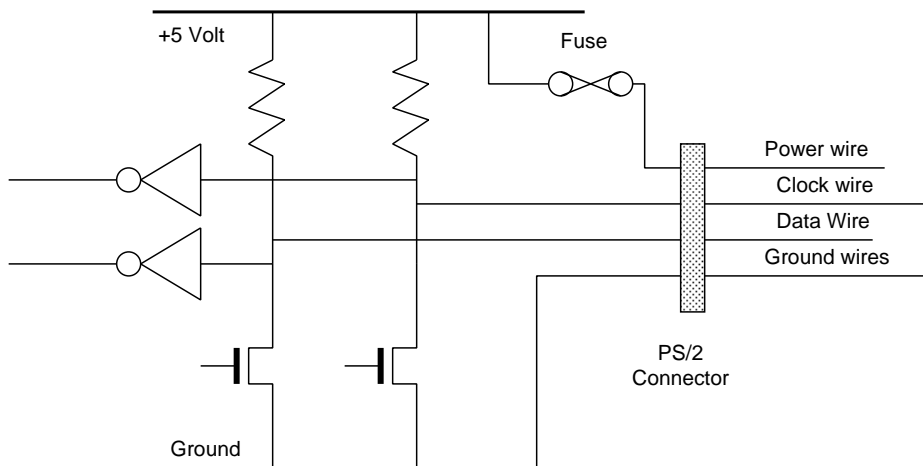
Open drain/collector wiring using two signalling wires.

The 1394 Firewire and USB ports are essentially the same as PS2 at the physical layer.

Ethernet

Processor Bus

Address — MAC — RX QUEUE

Data

interrupt ← IRQ

r/wbar → Read/Writebar — TX QUEUE

/cs — device select

TX-DATA
TX-CLK

RX-DATA
RX-CLK
CS/COL

PHY

Transformers

RJ45 Socket
(4 of 8 pins used)

42

Canonical Synchronous FSM

FSM = < Set of Inputs, Set of states Q, Transiton function D>

An initial state can be jumped to by designating one of the inputs a (synchronous) reset.

An automata theory "accepting state" would be indicated by a single Moore output, but in general hardware, we have multiple outputs, mostly of Moore style (to avoid combinational delay chains).

m inputs
n state bits

Schematic symbol
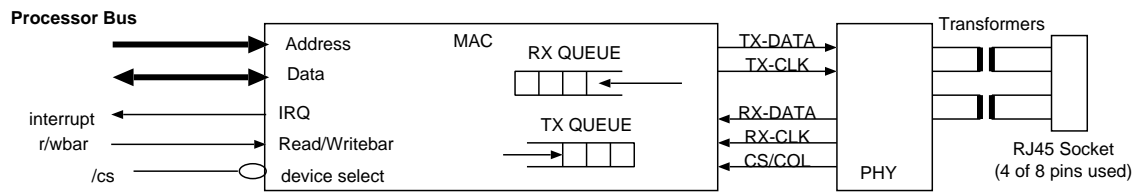
Inputs

Clock

Moore Outputs

Mealy Outputs

m

FSM

Internal structure

Inputs

Clock

I0
I1
I2
I(m-1)

Flip-flops for current state

D Q0
D Q1
D Q2
D Q(n-1)

Loop-free, combinational next state function

Current state feedback

Mealy output feedforward

Loop-free, combinational logic function (state labels)

Moore Outputs

Mealy Outputs

Loop-free, combinational logic function

43

Canonical Logic Array

AND
(product)
Array

OR
(sum)
array

Inputs

Outputs

Combinational Logic Minimisation

There are numerous combinatorial logic circuits that implement the same truth table.

Where two min-terms differ in one literal, they can alway be combined:

```
(A & ~B & C) + (A & ~B)        -->  (A & ~B)

(A & ~B & C) + (A & ~B & ~C)  -->  (A & ~B)
```

Lookup 'Kline-McClusky' for more information.

Karnaugh Maps are convenient to allow the human brain to perform minimisation by pattern recognition.

```
(A & ~C) + (A & B) + (B & C)  -->
                        (A & ~C) + (B & C)
```
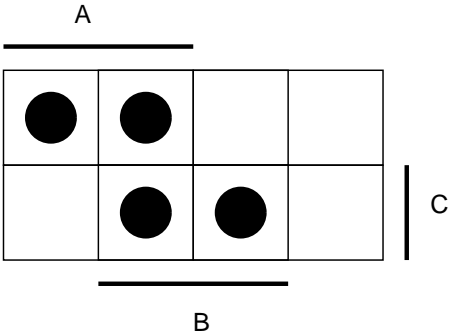

Often, there are don't care conditions, that allow further minimisation. Denote with an X on the K-map:
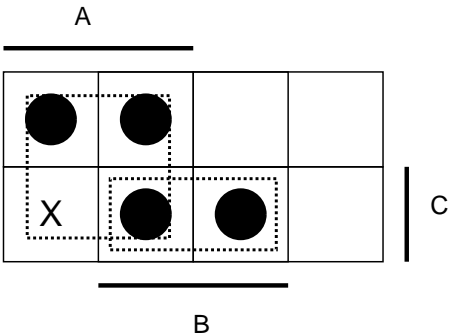

```
(A & ~C) + (A & B) + (B & C)  -->
                        A  + (B & C)
```

Lookup 'ESPRESSO' for more information.

Sequential Logic Minimisation

A finite state machine may have more states than it needs to perform its observable function.
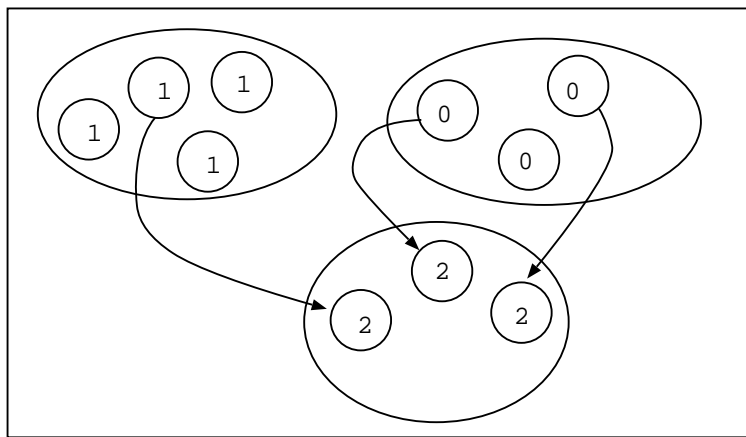
A Moore machine can be simplified by the following procedure

1. Partition all of the state space into blocks of
   states where the observable outputs are the same
   for all members of a block.

2. Repeat until nothing changes (i.e. until it closes)
     For each input setting:
        2a. Chose two blocks, B1 and B2.
        2b. Split B1 into two blocks consisting of those
            states with and without a transition from B2.
        2c. Discard any empty blocks.

3. The final blocks are the new states.

Timing Specifications

Data in   D   Q   Q oiutput

Clock

Clock

Hold time

Data in

Setup time

Q oiutput

Propagation delay
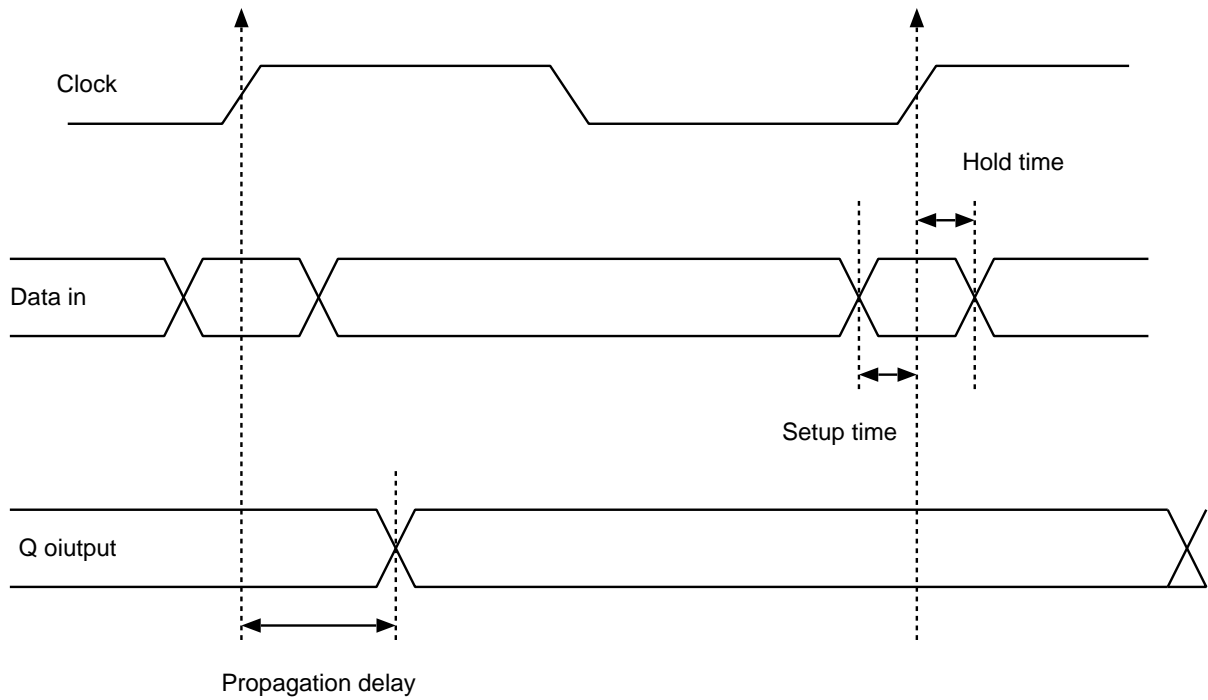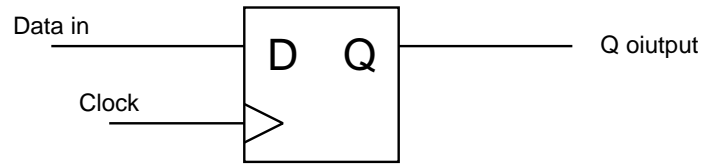
Typical Nature of a Critical Path
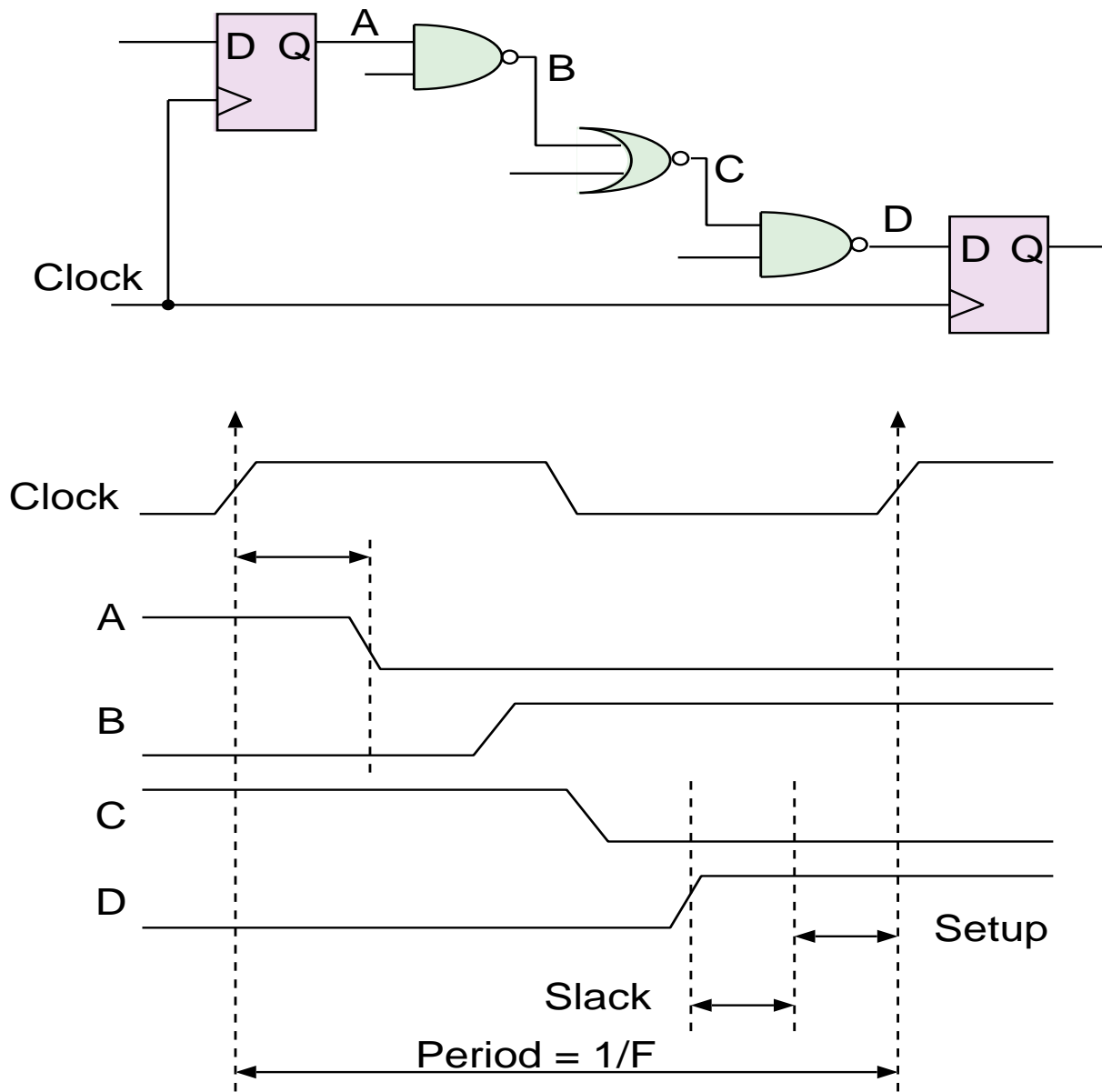
Clock speed can be increased while slack margin is positive.

Johnson counters

Q1	Q2	Q3

Clock

Pipelining

Data in

Another input

Yet another input

Domain clock

Large loop-free combinatorial logic function

An output

Yet another output

Another output still

Desired logic function

Data in

Another input

Yet another input

Domain clock

Loop-free combinatorial logic function - first half

... second half

An output

Yet another output

Another output still
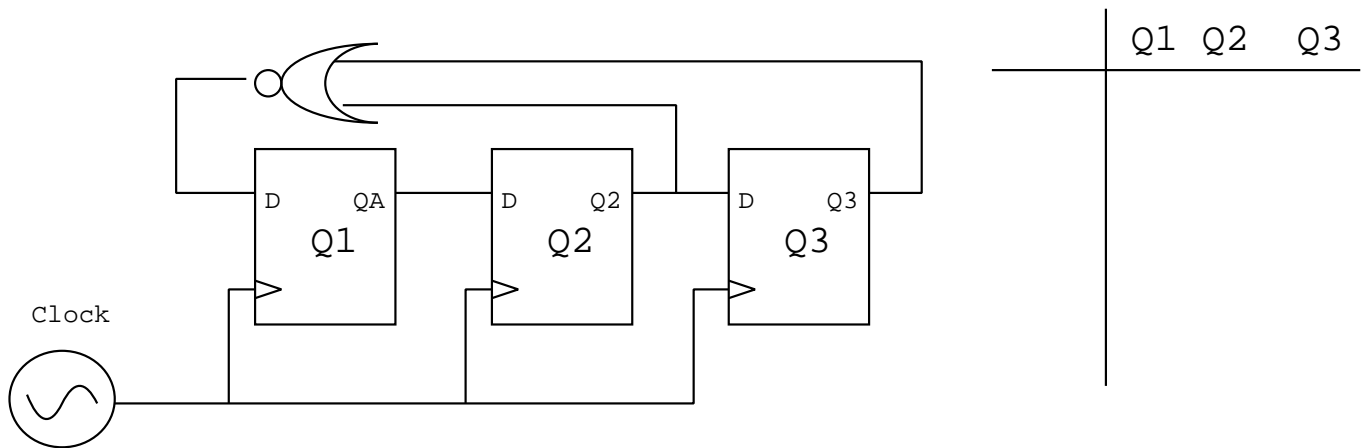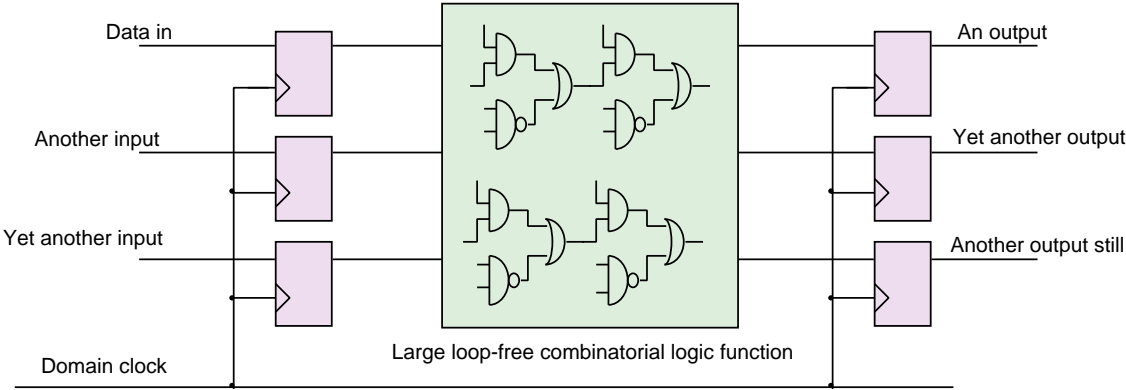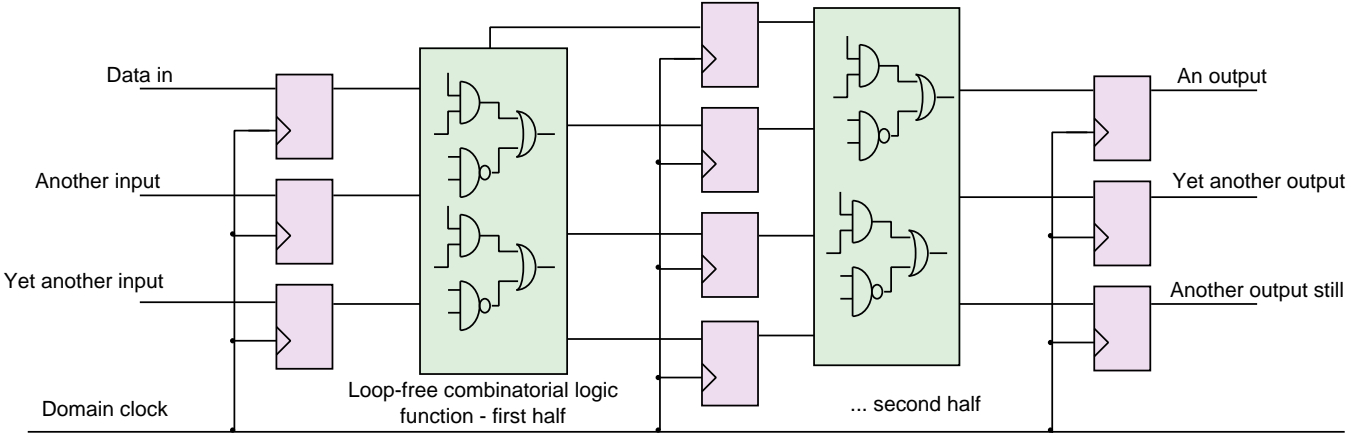
Same logic function - pipelined version.

Cascading FSMs

Clock

Inputs

FSM

Moore

Mealy

Inputs

FSM

Moore Outputs

Mealy Outputs

Inputs

FSM

Moore Outputs

Mealy Outputs

How Not To Do It

Five Bit BroadsideRegister   Parallel data out

Divide by 5 counter

D Q   D Q   D Q   D Q   D Q

Serial in

D Q   D Q   D Q   D Q   Shift Register
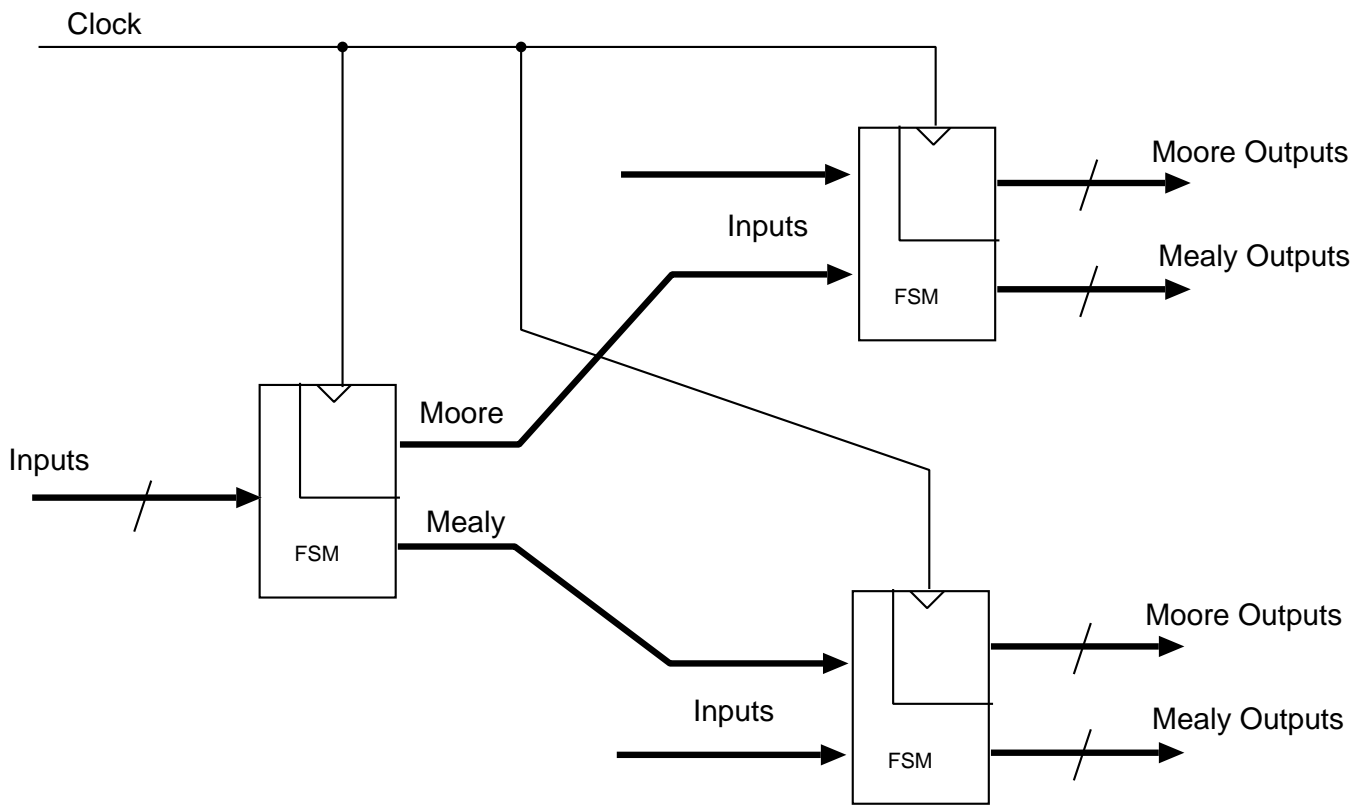
Clock input
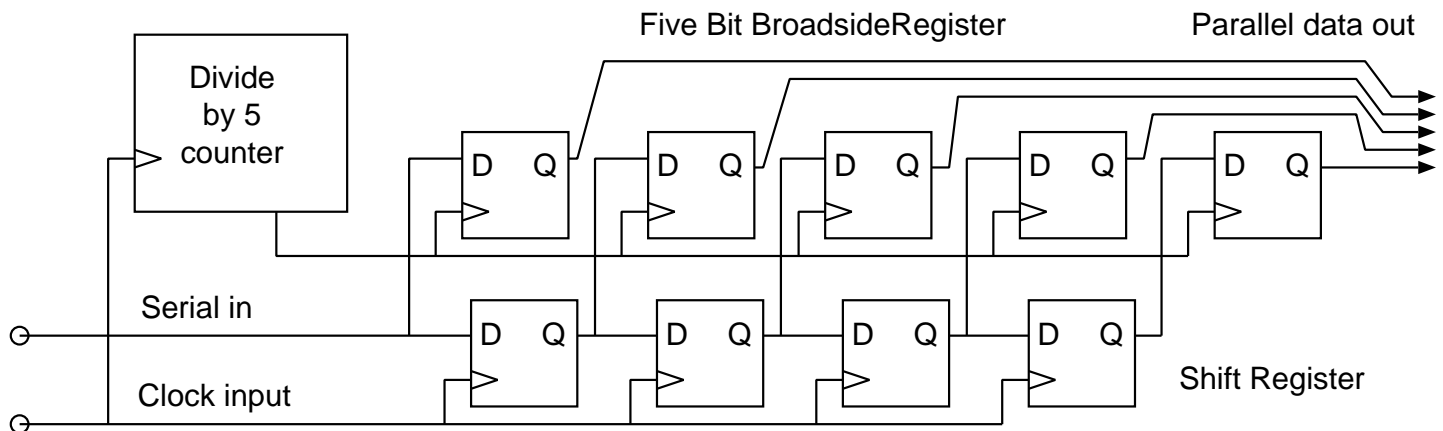
An example that uses (badly) a derived clock: a serial-to-parallel converter

```
reg [2:0] r2;
always @(posedge clock) r2 <= (r2==4)?0:r2+1;
wire bclock = r2[2];

reg [4:0] shift_reg;
always @(posedge clock)
    shift_reg <= serial_in | (shift_reg << 1);

reg [4:0] p_data;
always @(posedge bclock) p_data <= shift_reg;
```

Care is needed when gating clocks.

A Gated Clock
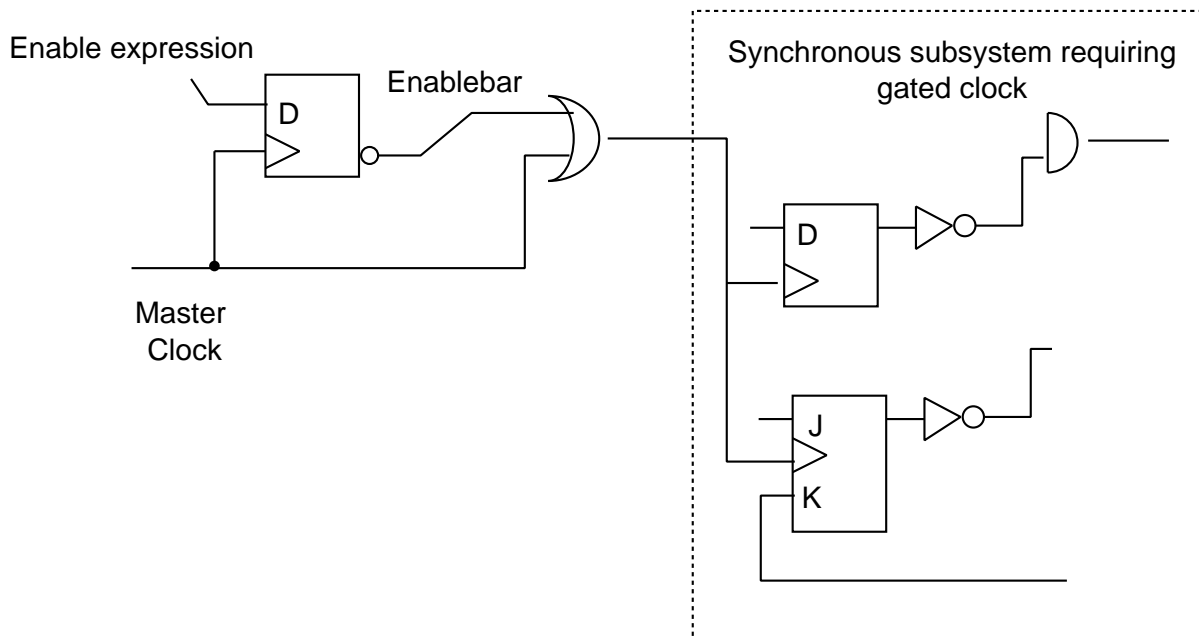
OR'ing with a negated enable works cleanly.

Use this to power down a sub-section of a chip or when synchronous clock enable becomes costly.

Clock Skew
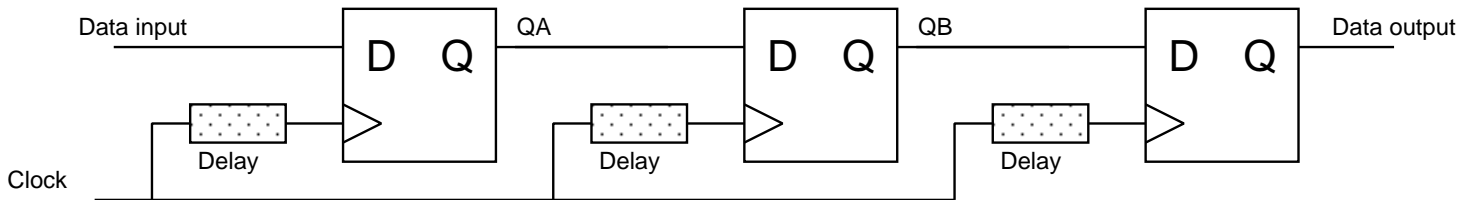
a) A three-stage shift register with some clock skew delays.
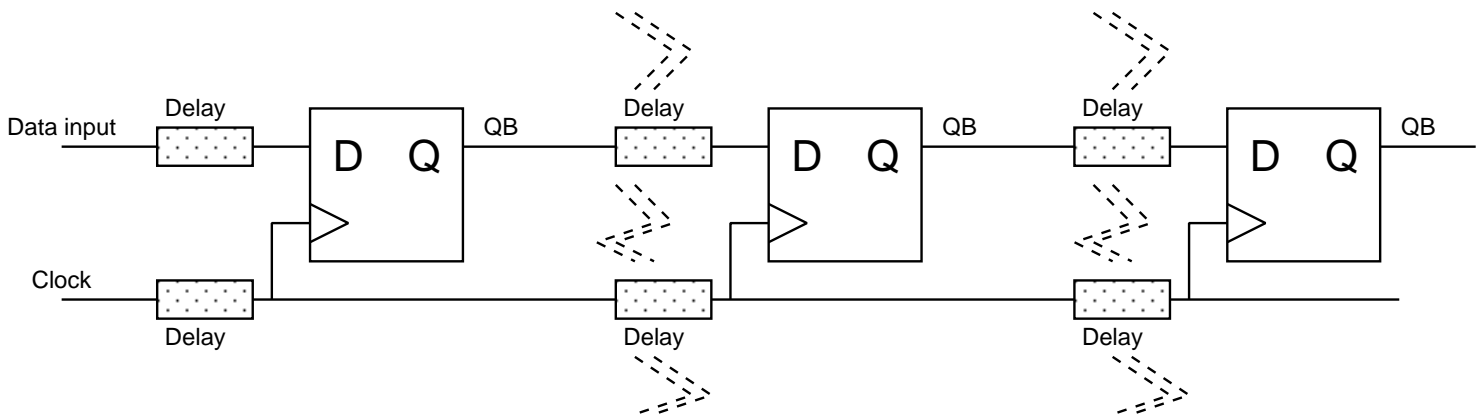
b) System interconnection with clock skews
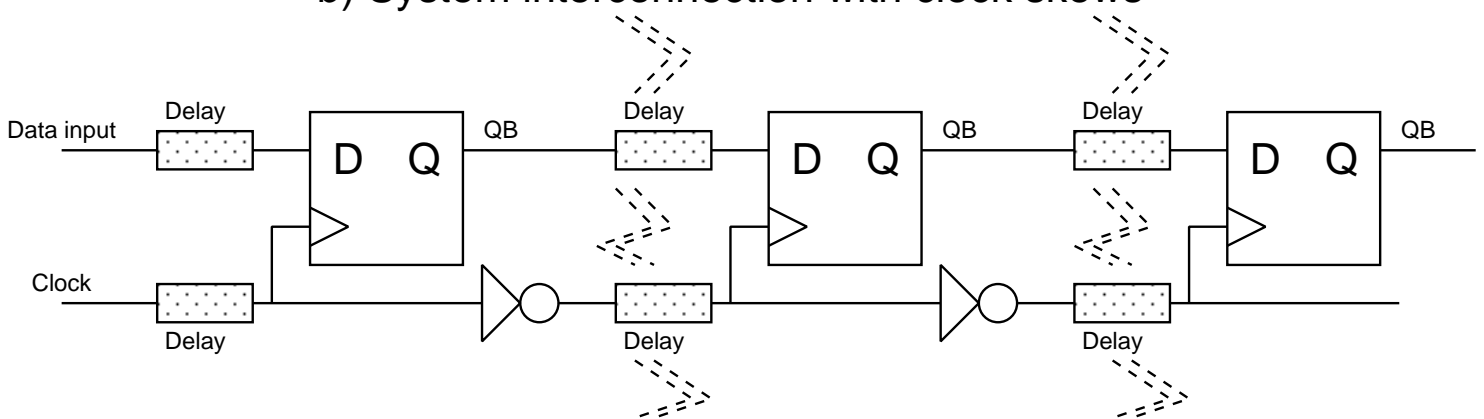
c) A solution for serious skew and delay problems ?

Crossing an Asynchronous Domain Boundary

Transmit clock domain

Receiving clock domain

Guard signal

Good to have a second D-type

Command or info bus

N

TX clock

RX clock
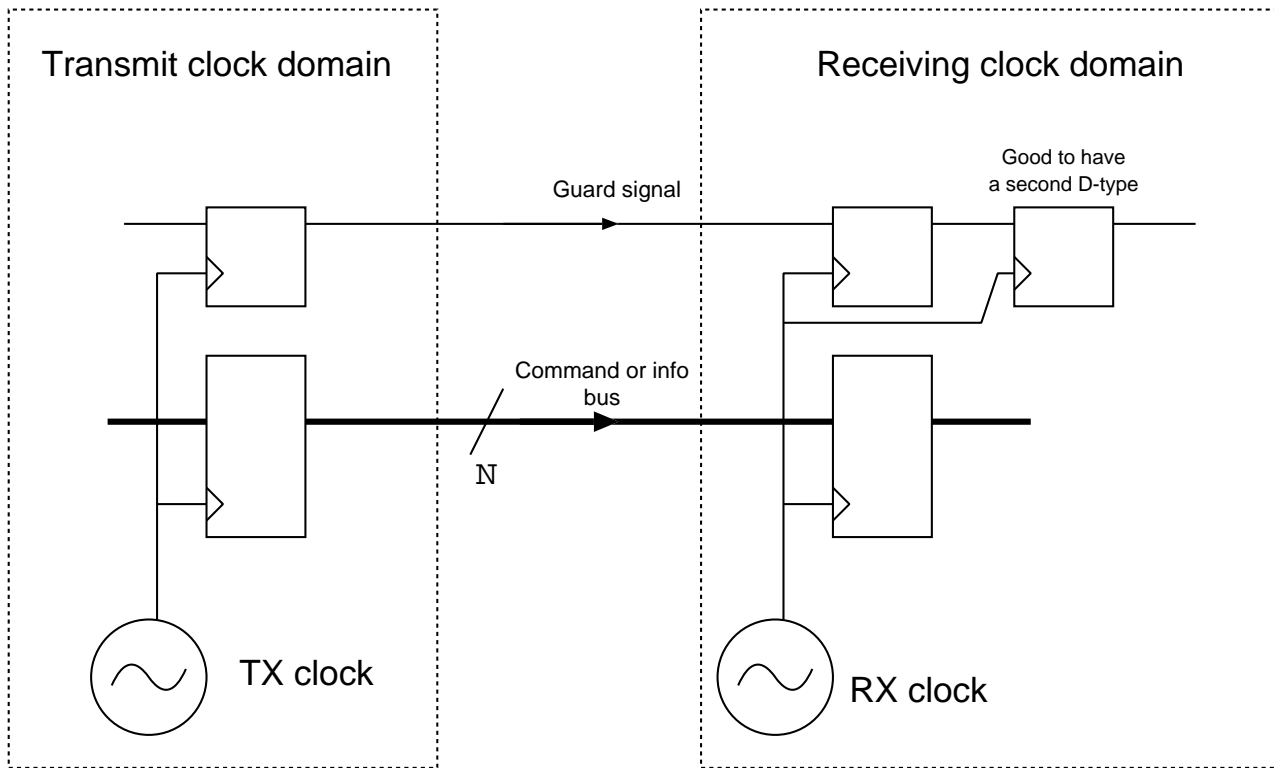
1. The wider the bus width, N, the fewer the number of transactions per second needed and the greater the timing flexibility in reading the data from the receiving latch.

2. Make sure that the transmitter does not change the guard and the data in the same transmit clock cycle.

3. Place a second flip-flop after the receiving decision flip-flop so that on the rare occurances when the first is metastable for a significant length of time (e.g. 1/2 a clock cycle) the second willpresent a good clean signal to the rest of the receiving system.

All real systems have many clock domains and frquently implement this style of solution.

Dicing a wafer
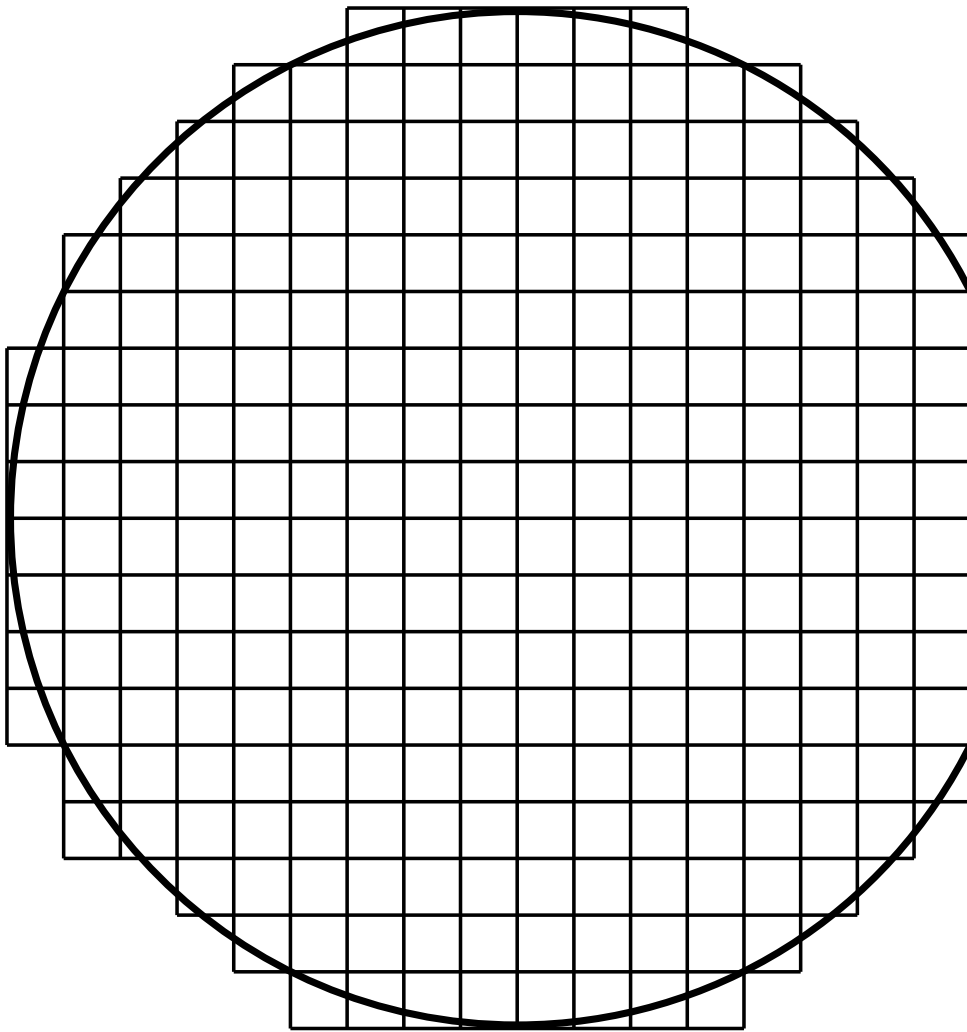
(Chips are not always square)

A wire-bonded chip in its package

PACKAGE

CHIP
DIE

PACKAGE
PIN

BOND
WIRE

BOND PAD
(unused)

CAVITY

IO and power pads

CORE AREA

Connections
to and from
core logic

Power Rail

Pad power
supply

Pad
Electronics

Ground
Pad

Edge of Die

Ground
Rail

Supply
Pad

Signal
Bond
Pad
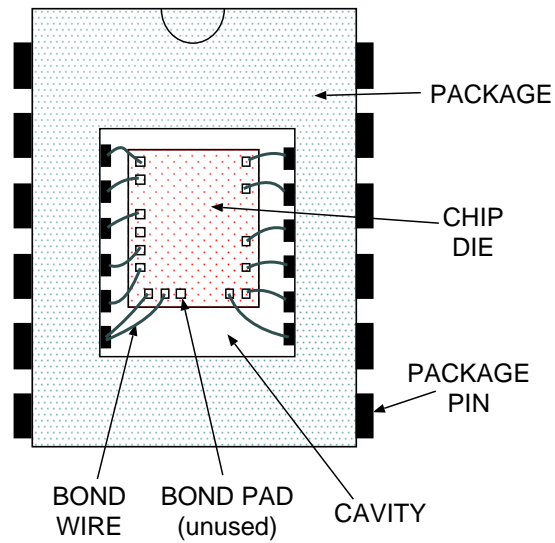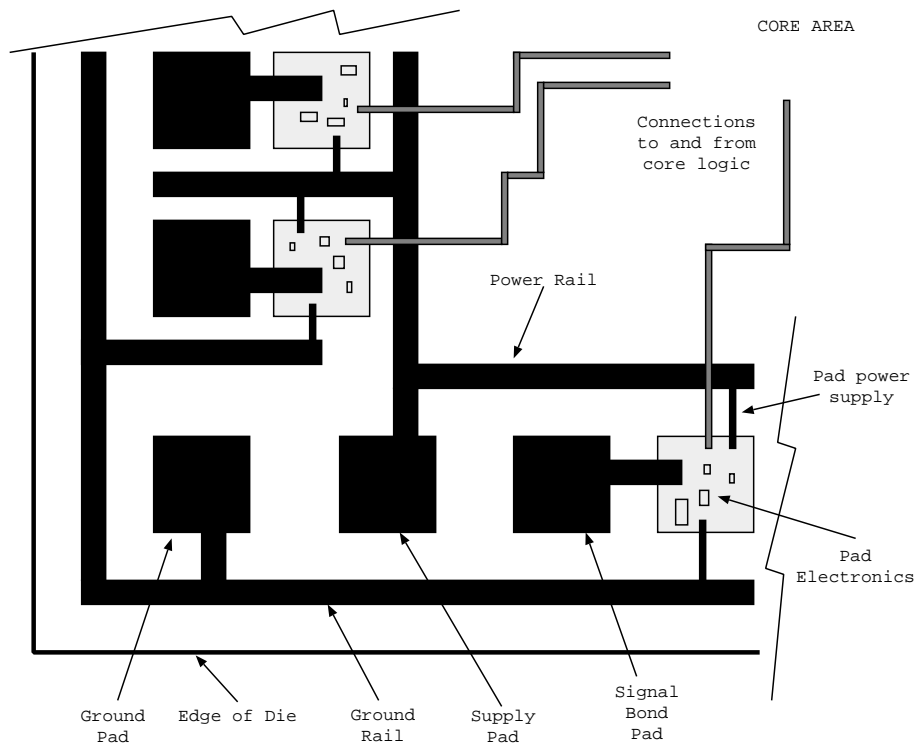
Die cost example

Area	Wafer dies	Working dies	Cost per working die
2	9000	8910	0.56
3	6000	5910	0.85
4	4500	4411	1.13
6	3000	2911	1.72
9	2000	1912	2.62
13	1385	1297	3.85
19	947	861	5.81
28	643	559	8.95
42	429	347	14.40
63	286	208	24.00
94	191	120	41.83
141	128	63	79.41
211	85	30	168.78
316	57	12	427.85
474	38	4	1416.89

A taxonomy of ICs

Digital Integrated Circuits

```
                         Digital Integrated Circuits
        ┌────────────────────────┼─────────────────────────┐
  ┌──────────┐             ┌──────────┐              ┌──────────────┐
  │ Standard │             │  ASICs   │              │    Field     │
  │  Parts   │             │          │              │ Programmable │
  └──────────┘             └──────────┘              │    Parts     │
                                                     └──────────────┘
```

| Full Custom | Semi Custom Standard Cell | | Full Custom | Semi Custom Standard Cell | | Array Logic (PALs) | FPGA e.g. | SOC FPGAs |

Full
Custom

Semi
Custom
Standard
Cell

Full
Custom

Semi
Custom
Standard
Cell

Array
Logic
(PALs)

FPGA
e.g.

SOC
FPGAs

Commodity
Parts

e.g.
LAN
Interface
Controller

Rarely
Used

e.g.
Toys
Cellphones
iPODs

e.g.
22V10

Xilinx
Spartan
Virtex
Altera

e.g
Altera
Excalibur

e.g.
Memories
Processors

Field Programmable Gate Arrays

Edge of die

Bond pad	IOB	Bond pad	IOB	Bond pad	IOB	Bond pad

SWITCH MATRIX — SWITCH MATRIX — SWITCH MATRIX — SWITCH MATRIX — IOB

CLB · CLB · CLB · CLB — Bond pad

SWITCH MATRIX — SWITCH MATRIX — SWITCH MATRIX — SWITCH MATRIX — IOB

CLB · CLB · CLB · CLB — Bond pad

SWITCH MATRIX — SWITCH MATRIX — SWITCH MATRIX — IOB

CLB · CLB — Bond pad
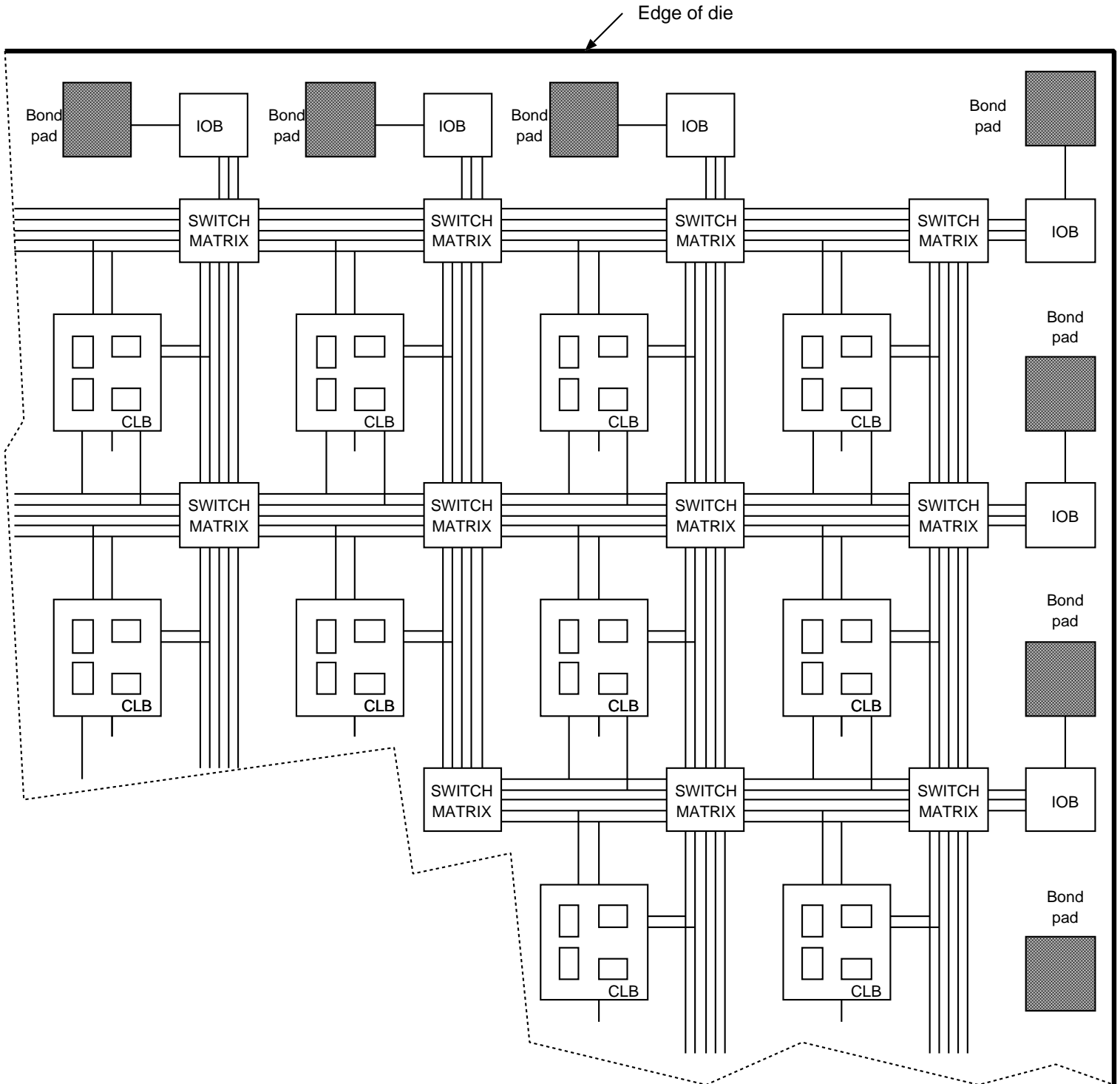
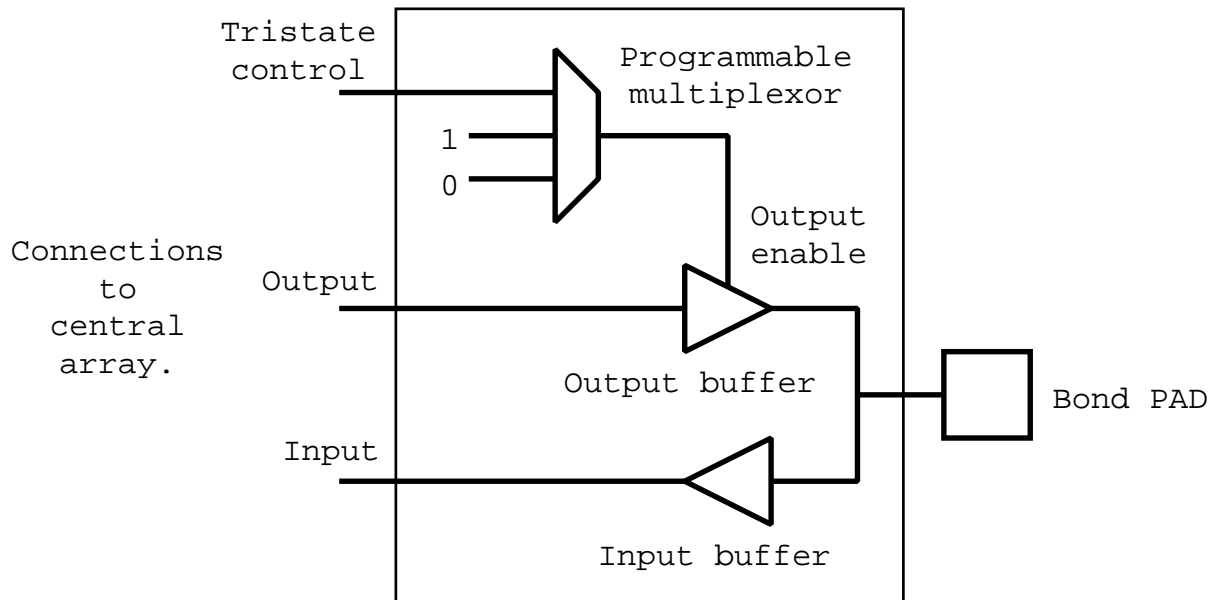# A configurable logic block for a look-up-table based FPGA
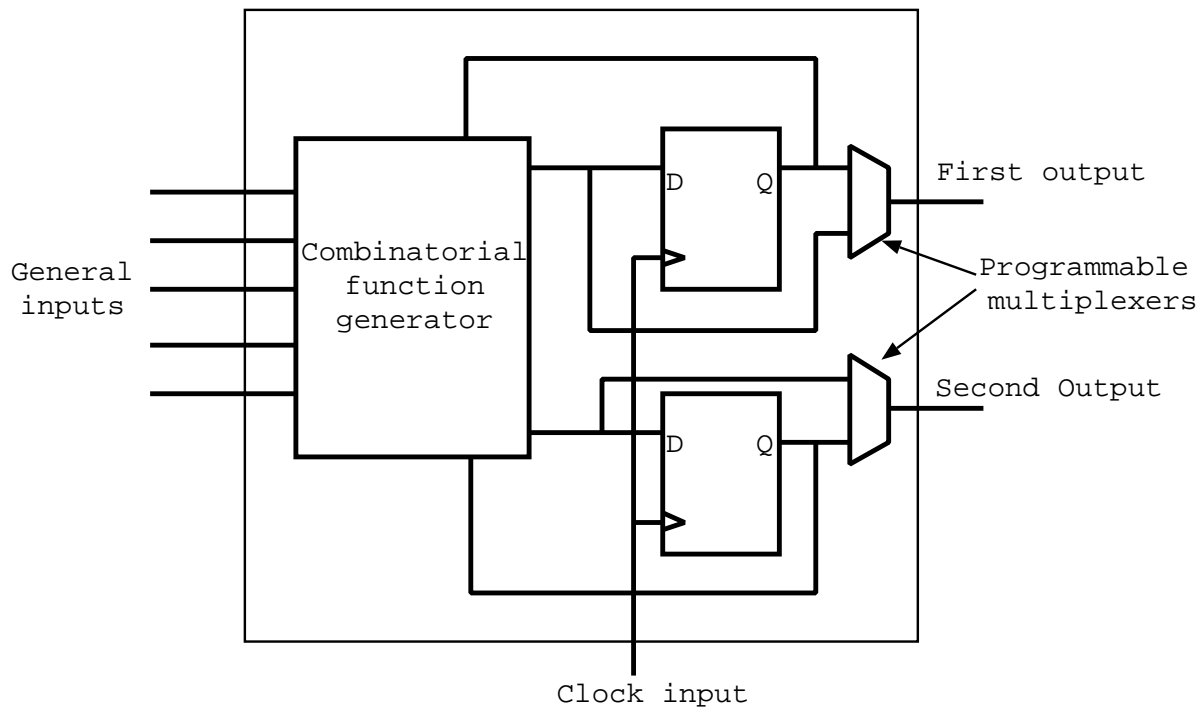
This CLB contains one LUT and two D-type's. The output can be sequential or combinational.

Seven LUT inputs: $2^7 = 128$

The LUT can be a RAM of 128 locations of two bits.

FPGA: Example I/O Block

```
          Tristate          Programmable
          control           multiplexor
                        1
                        0
                                     Output
   Connections                       enable
       to        Output
     central
     array.
                                Output buffer          Bond PAD

                Input

                                 Input buffer
```
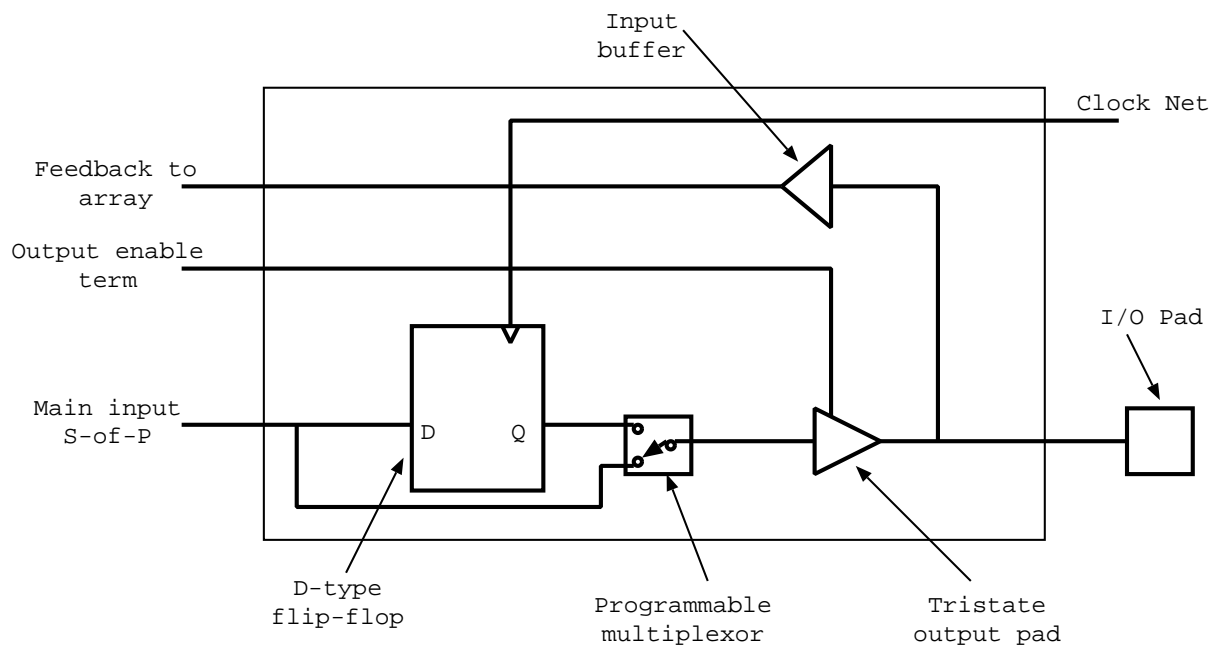
Pictured is a basic I/O block.

Modern FPGA's have have a variety of different I/O blocks: e.g. for PCI bus or 1 Gbps channel.

Clock input

General purpose inputs

Product line

Term line

Output enable product line

The cross points in these shaded regions are programmable points

Ground pin.

Power supply pin

Clock signal

Macro-cell

Output pad (can also be input).

64
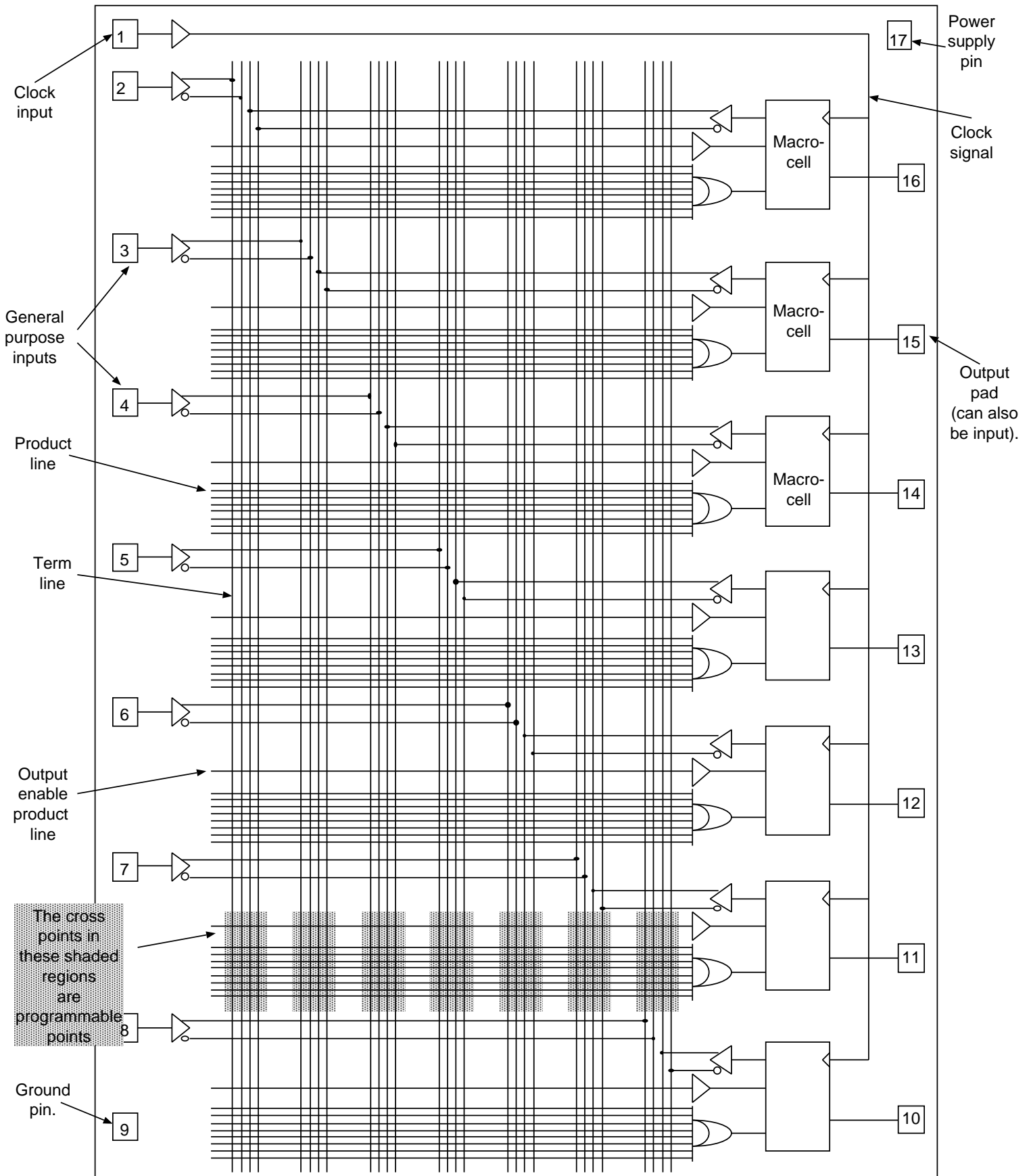
Contents of the PAL macrocell

Input buffer

Clock Net

Feedback to array

Output enable term

I/O Pad

Main input S-of-P

D-type flip-flop

Programmable multiplexor

Tristate output pad

# Example programming of a PAL showing only fuses for the top macrocell

```
pin 16 = o1;
pin 2 = a;
pin 3 = b;
pin 4 = c

o1.oe = ~a;
o1 = (b & o1) | c;
```

```
-x-- ---- ---- ---- ---- ---- ----   (oe term)
--x- x--- ---- ---- ---- ---- ----   (pin 3 and 16)
---- ---- x--- ---- ---- ---- ----   (pin 4)
xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx
xxxx xxxx xxxx xxxx xxxx xxxx xxxx
x                                    (macrocell fuse)
```
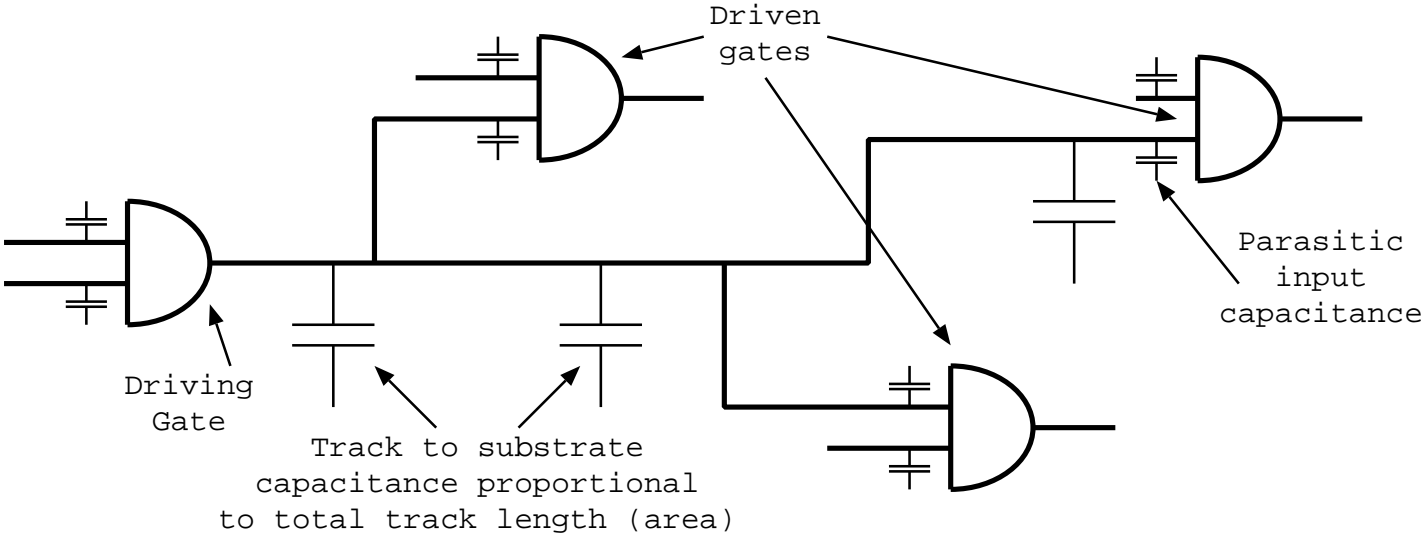
Delay-power style of technology comparison chart
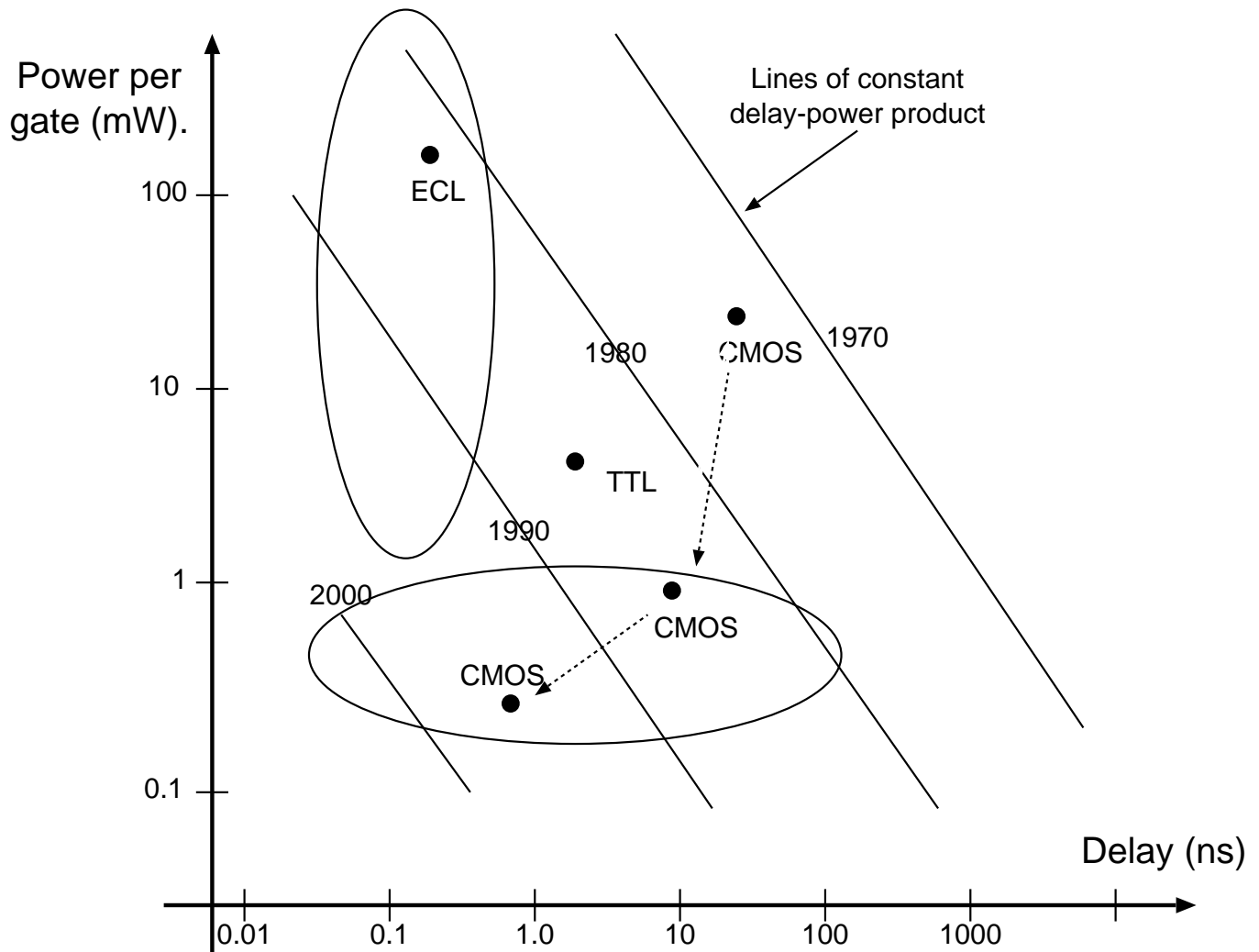
	Technology	device	propagation	power	product
	----------	------	-----------	------	-------
1977	CMOS	HEF4011	30 ns	32 mW	960 pJ
1982	ECL	sp92701	0.8 ns	200 mW	160 pJ
1983	CMOS	74hc00	7 ns	1 mW	7 pJ
1983	TTL	74f00	3.4 ns	5 mW	17 pJ
1996	CMOS	74LVT00	2.7 ns	0.4 mW	1.1 pJ

2-Input NAND gate. 74LVT00 is 3V3. On-chip logic is much faster.

Logic net with tracking and input load capacitances

Driven gates

Driving Gate

Track to substrate capacitance proportional to total track length (area)
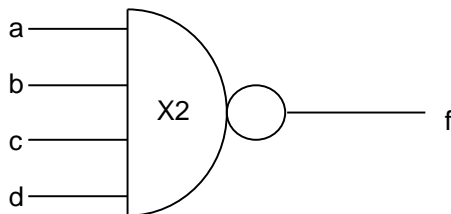
Parasitic input capacitance

An example cell from a manufacturer's cell library

NAND4 Standard Cell

Library: CBG0.5um

4 input NAND gate with x2 drive

Schematic Symbol

Simulator/HDL Call

NAND4X2(f, a, b, c, d);

Logical Function

F = NOT(a & b & c & d)

ELECTRICAL SPECIFICATION

Switching characteristics : Nominal delays (25 deg C, 5 Volt, signal rise and fall 0.5 ns)

Inputs	Outputs	O/P Falling		O/P Rising	
		(ps)	ps/LU	ps	ps/LU
A	F	142	37	198	33
B	F	161	37	249	33
C	F	165	37	293	33
D	F	170	37	326	34

Min and Max delays depend upon temperature range, supply voltage, input edge speed and process spreads. The timing information is for guidance only. Accurate delays are used by the UDC.

CELL PARAMETERS   : (One load unit = 49 fF)

Parameters	Pin	Value	Units
Input loading	a b c d	2.1 2.1 2.1 2.0	Load units
Drive capability	f	35	Load units

Current digital logic technologies

1994 - First 64 Mbit DRAM chip.

- 0.35 micron CMOS

- 1.5 micron^2 cell size (64E6 $\times$ 1.5 um^2 = 96E6)

- 170 mm^2 die size
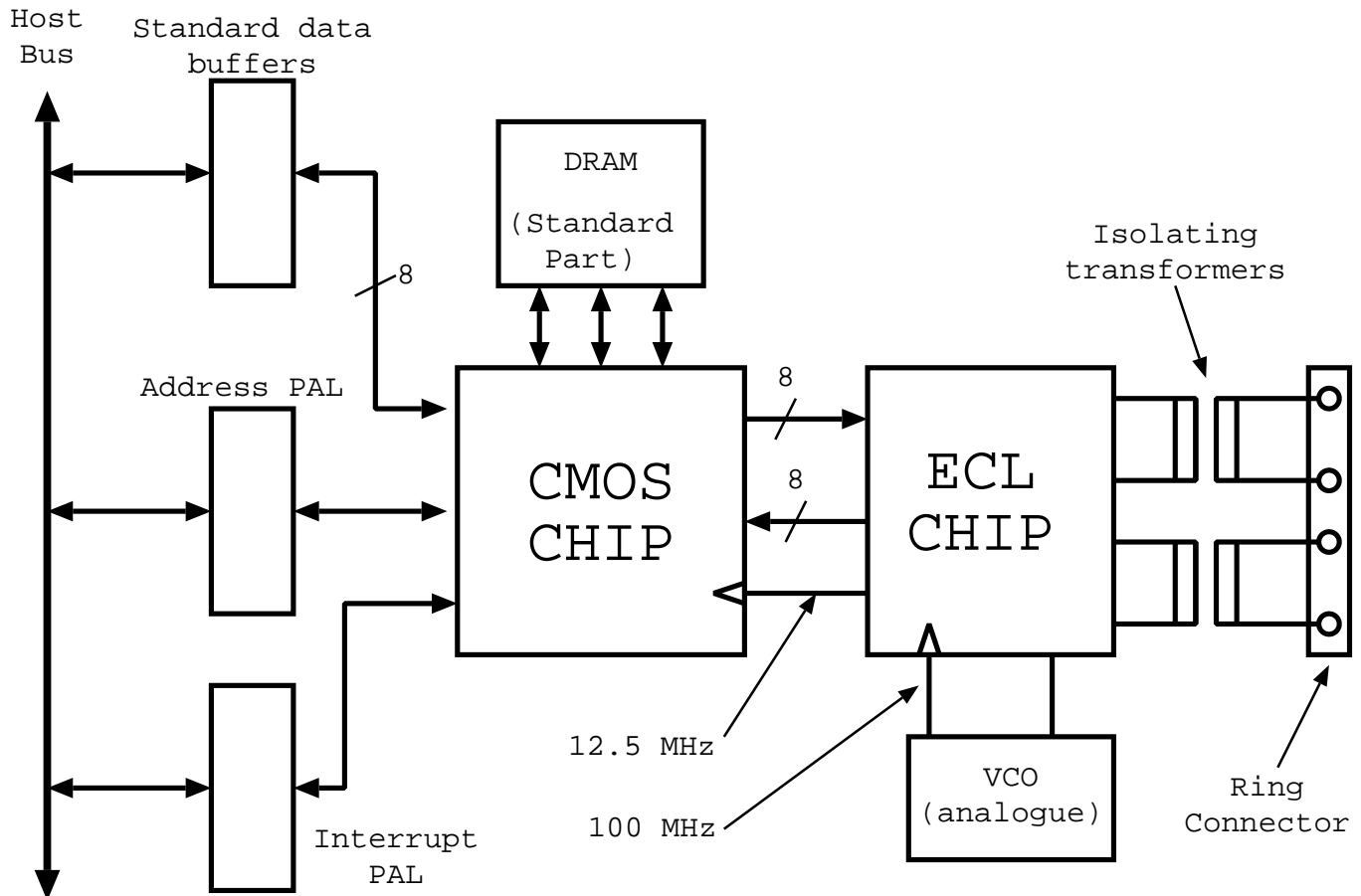
1999 - Intel Pentium Three

- 0.18 micron line size

- 28 million transistors

- 500-700 MHz clock speed

- 11x12 mm (140 mm^2) die size

2003 - Lattice FPGA

- 1.25 million use gate equivs

- 414 Kbits of SRAM

- 200 MHz Clock Speed

- same die size.

See www.icknowledge.com

Design partitioning: The Cambridge Fast Ring

Host
Bus

Standard data
buffers

DRAM

(Standard
Part)

8

Address PAL

CMOS
CHIP

8

8

ECL
CHIP

Isolating
transformers

12.5 MHz

100 MHz

VCO
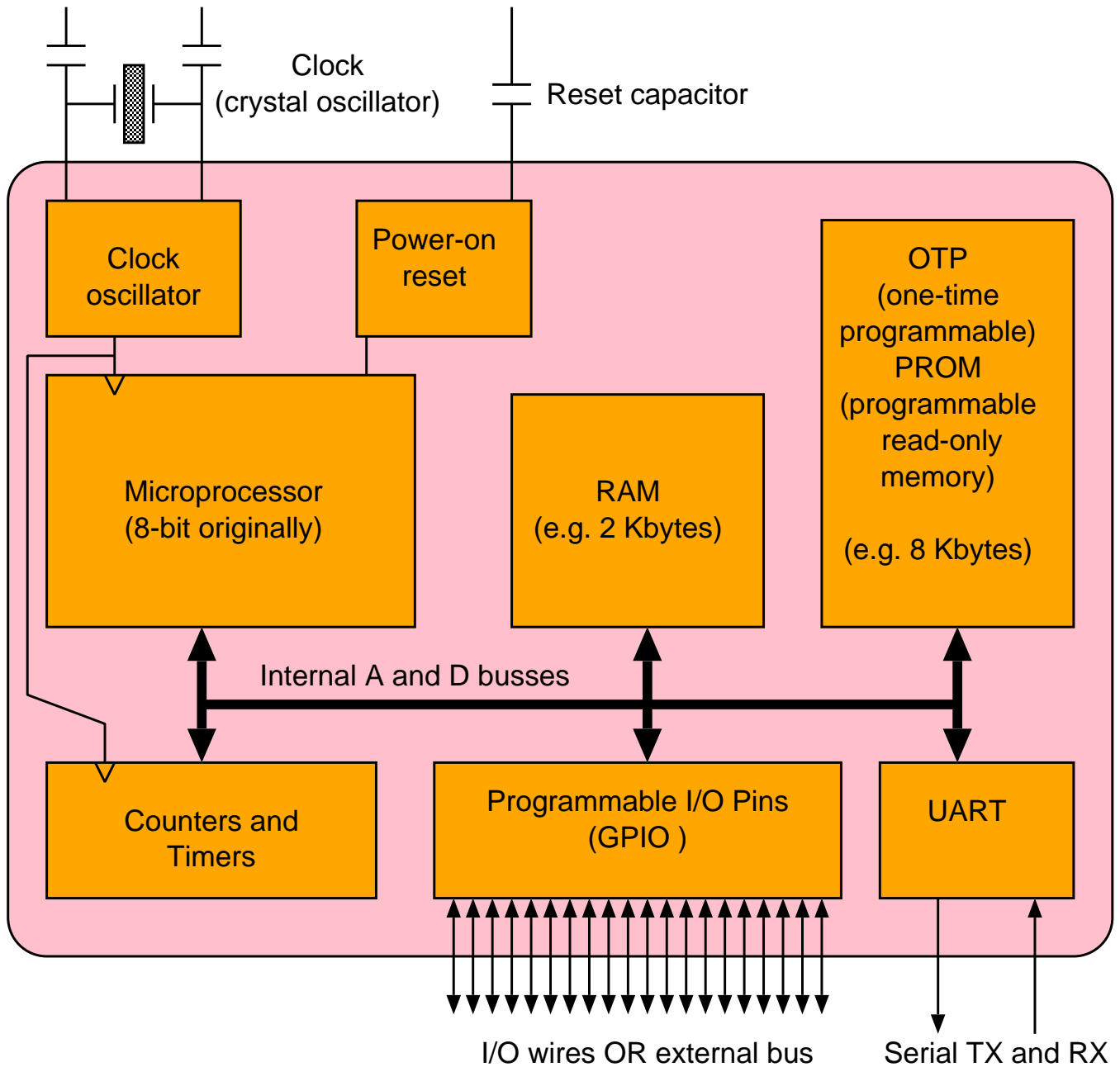(analogue)

Ring
Connector

Interrupt
PAL

Designed in 1980.

ECL Chip 100 MHz, bit serial.

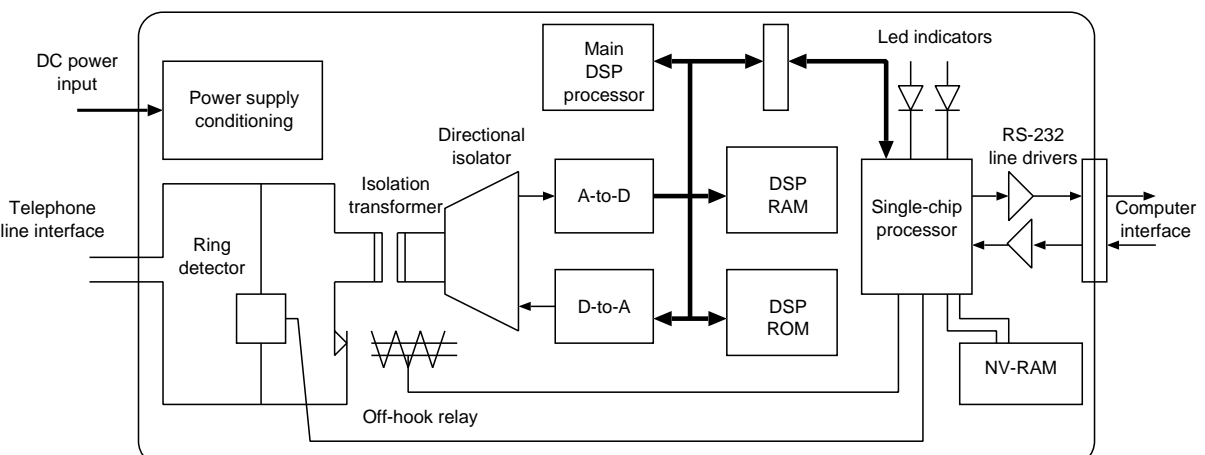CMOS Chip 12.5 MHz, byte-wide data.
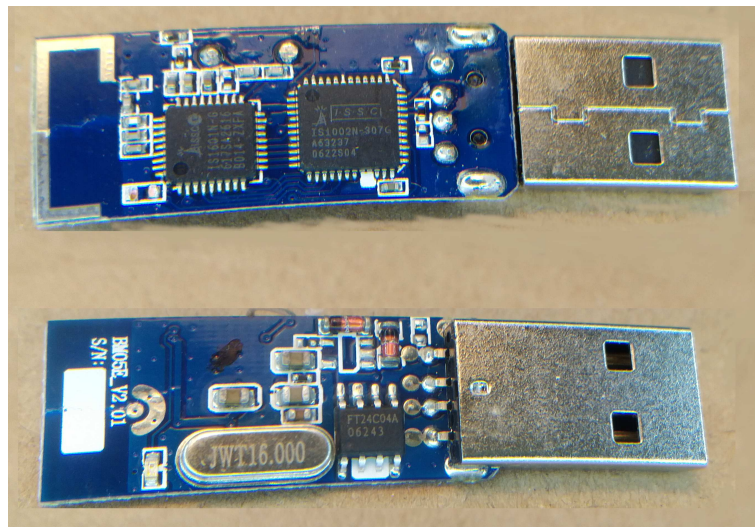
A Basic Microcontroller
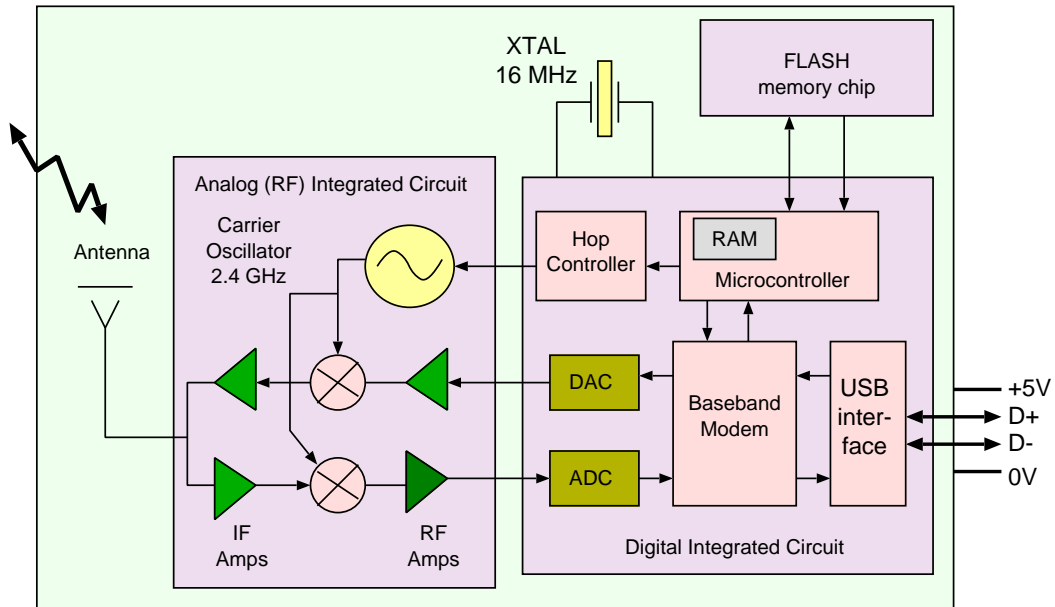

Introduced 1989-85.

Such microcontrollers had an D8/A16 architecture and would be used in a mouse or smartcard.
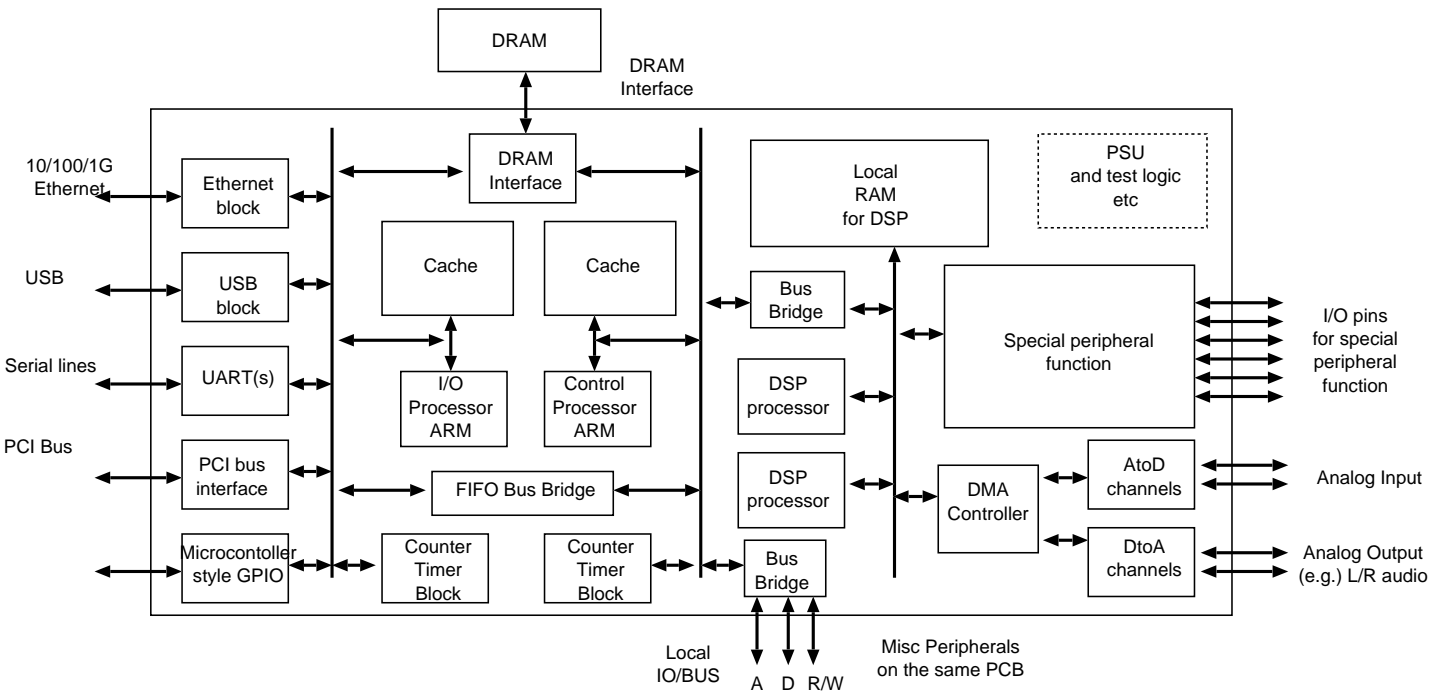
Design partitioning: A Modem.

In 1980 we used a microcontroller with external DSP components.

DC power input	Power supply conditioning		Main DSP processor		Led indicators		
Telephone line interface	Ring detector	Isolation transformer	Directional isolator	A-to-D	DSP RAM	Single-chip processor	RS-232 line drivers
				D-to-A	DSP ROM		Computer interface
		Off-hook relay				NV-RAM	

Design partitioning: A Miniature Radio Module

System on a Chip = SoC design.

Our platform chip has two ARM processors and two DSP processors. Each ARM has a local cache and both store their programs and data in the same offchip DRAM.

The left-hand-side ARM is used as an I/O processor and so is connected to a variety of standard peripherals. In any typical application, many of the peripherals will be unused and so held in a power down mode.

The right-hand-side ARM is used as the system controller. It can access all of the chip's resources over various bus bridges. It can access off-chip devices, such as an LCD display or keyboard via a general purpose A/D local bus.
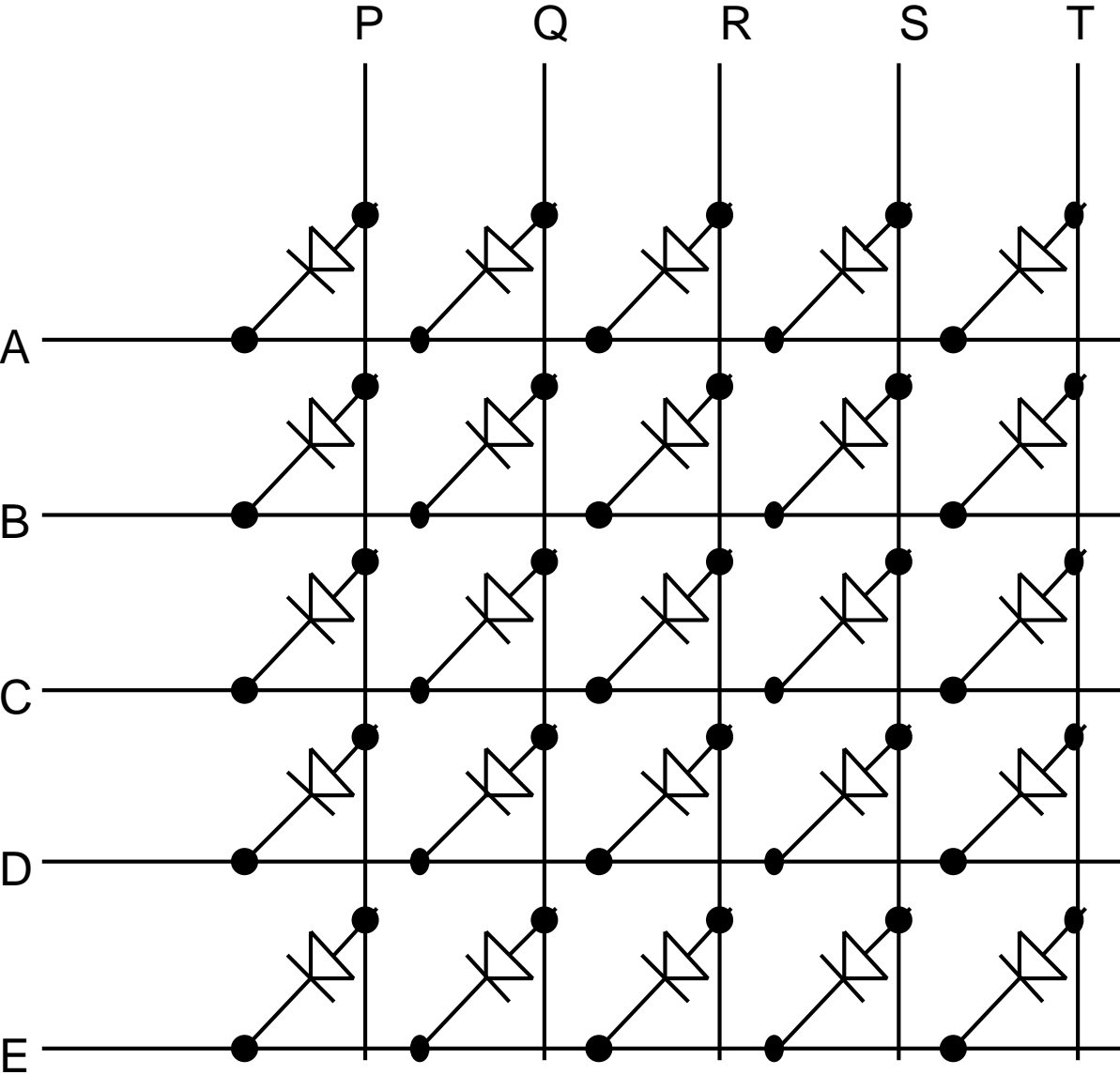
The bus bridges map part of one processor's memory map into that of another so that cycles can be executed in the other's space, allbeit with some delay and loss of performance. A FIFO bus bridge contains its own transaction queue of read or write operations awaiting completion.

The twin DSP devices run completely out of on-chip SRAM. Such SRAM may dominate the die area of the chip. If both are fetching instructions from the same port of the same RAM, then they had better be executing the same program in lock-step or else have some own local cache to avoid huge loss of performance in bus contention.
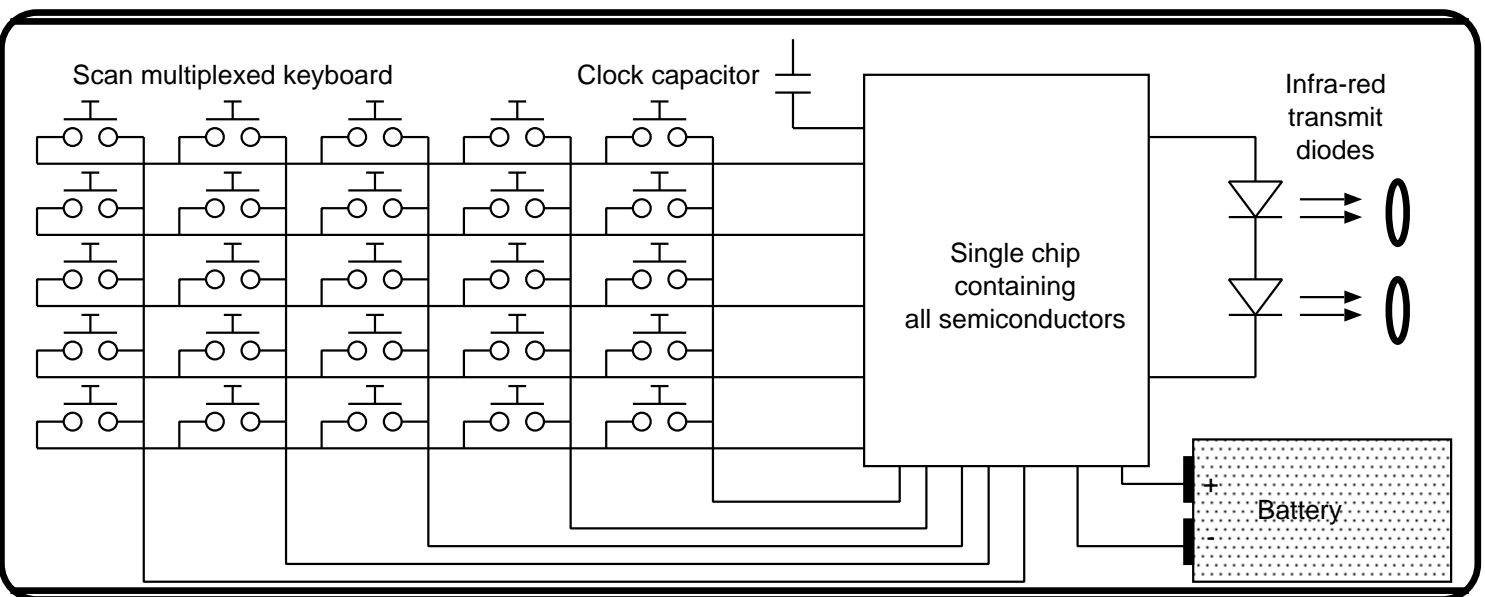
The rest of the system is normally swept up onto the same piece of silicon and this is denoted with the 'special function periperhal.' This would be the one part of the design that varies from product to product. The same core set of components would be used for all sorts of different products, from iPODs, digital cameras or ADSL modems.
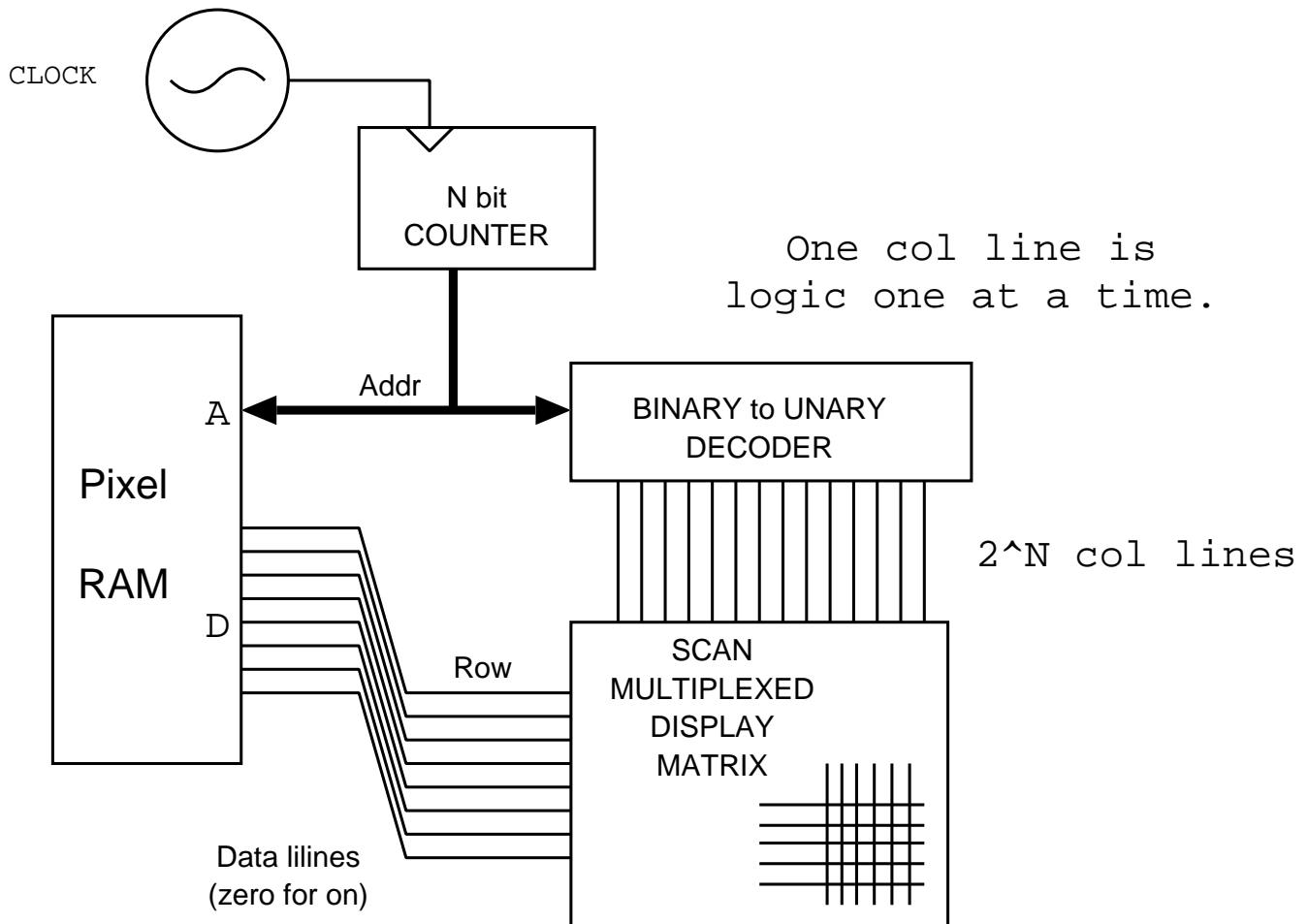
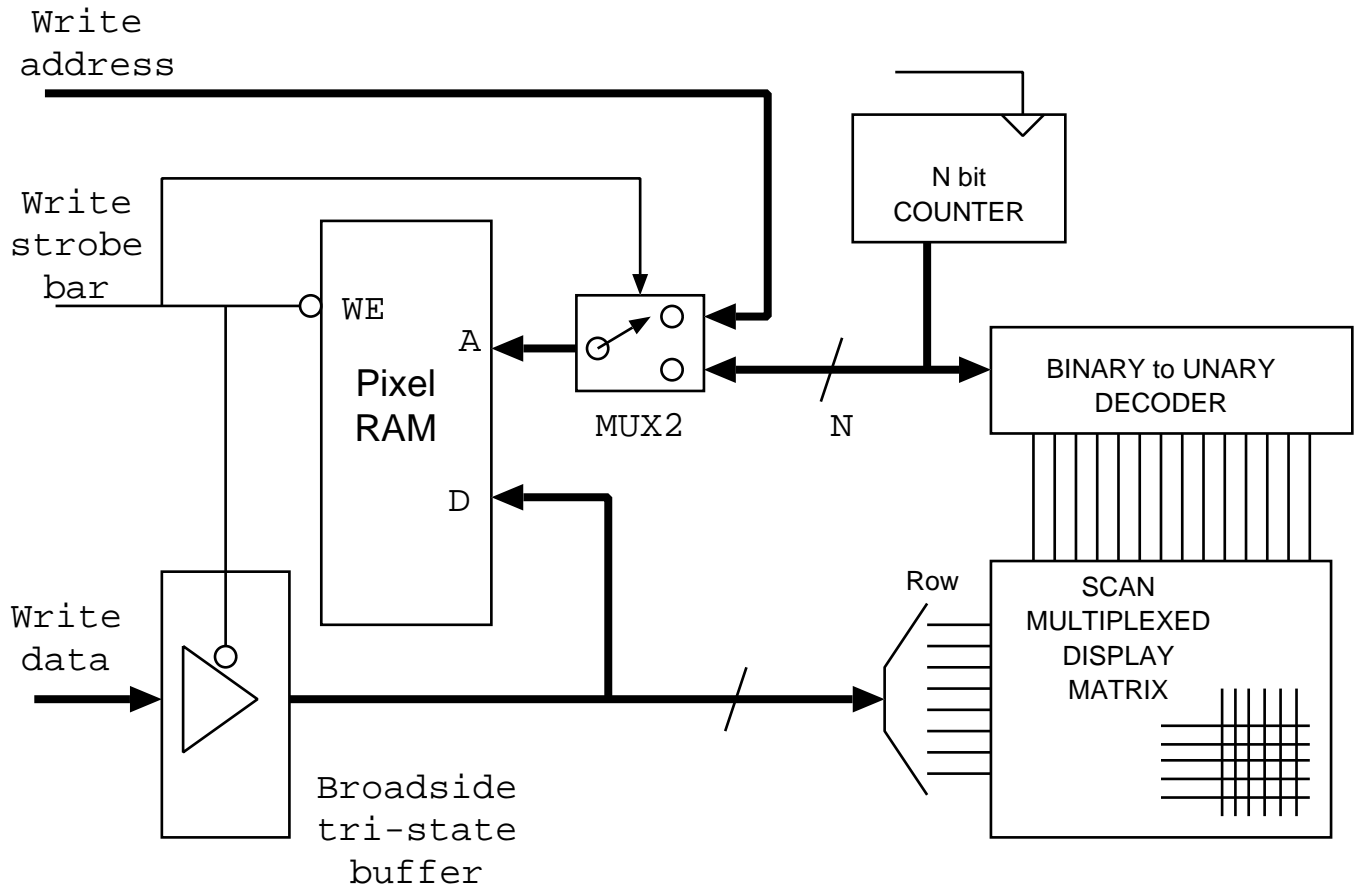# LEDs wired in a matrix to reduce external pin count
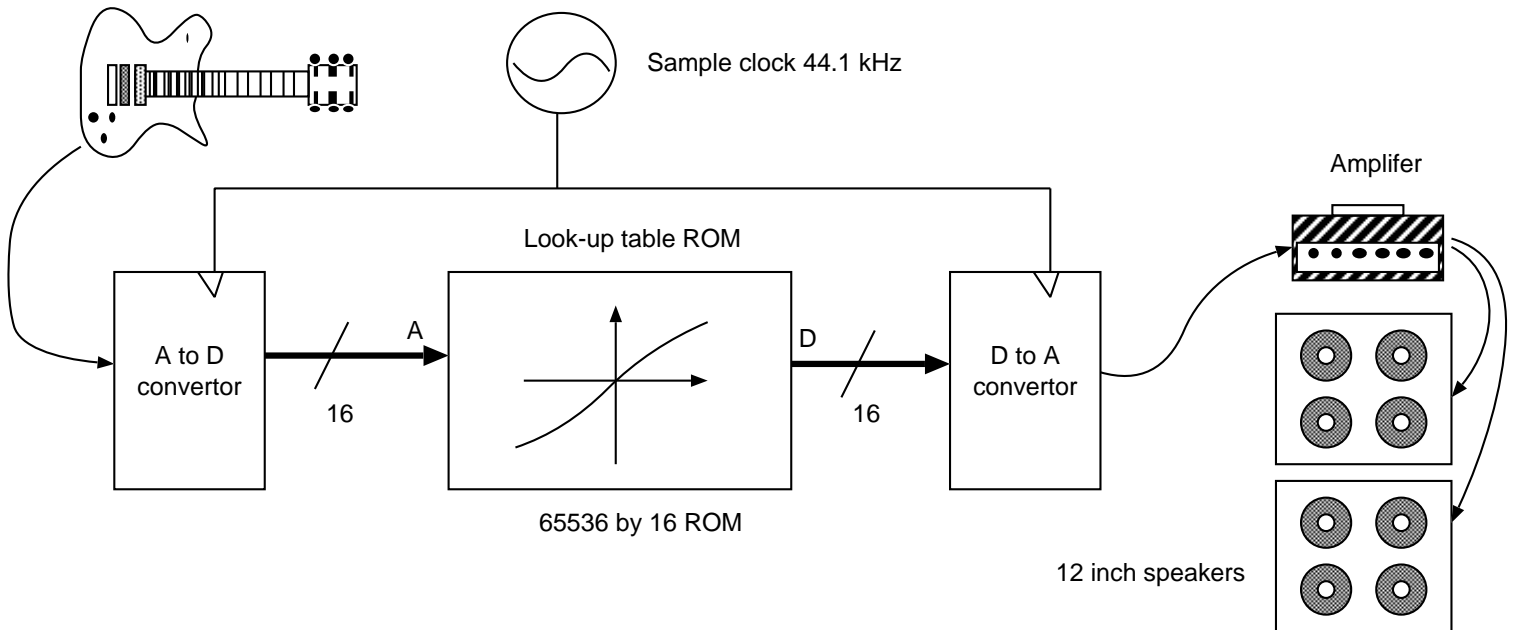
IR Handset Internal Circuit

Scan multiplexed keyboard

Clock capacitor

Single chip containing all semiconductors

Infra-red transmit diodes

Battery

Scan multiplex logic for an LED pixel-mapped display

CLOCK

N bit
COUNTER

One col line is
logic one at a time.

Addr

A

Pixel

RAM

D

BINARY to UNARY
DECODER

2^N col lines

Row

SCAN
MULTIPLEXED
DISPLAY
MATRIX

Data lilines
(zero for on)

You made one of these in the Ia H/W classes.

Addition of psudo dual-porting logic

Write address

Write strobe bar

N bit COUNTER

WE

A

Pixel RAM

MUX2

N

BINARY to UNARY DECODER

D

Row

SCAN MULTIPLEXED DISPLAY MATRIX

Write data

Broadside tri-state buffer

You did this too!

Use of a ROM as a function look-up table

Sample clock 44.1 kHz

Amplifer

Look-up table ROM

A to D convertor

A

16

D

16

D to A convertor

65536 by 16 ROM
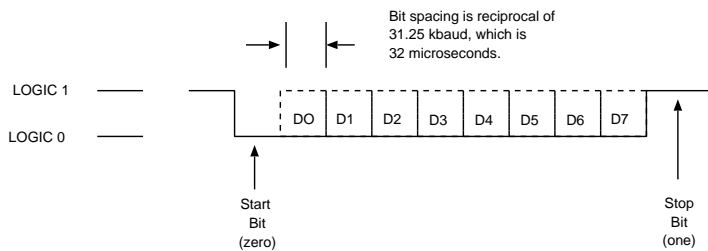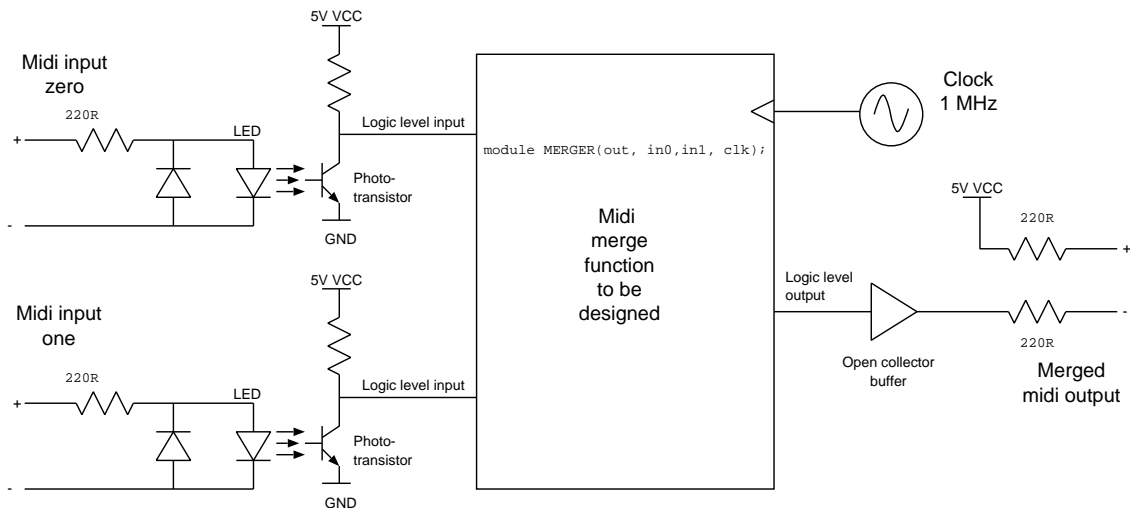
12 inch speakers

The ROM contains the exact imperfections of a 1950's valve amplifier.

Use of an SRAM to make the delay required for an echo unit
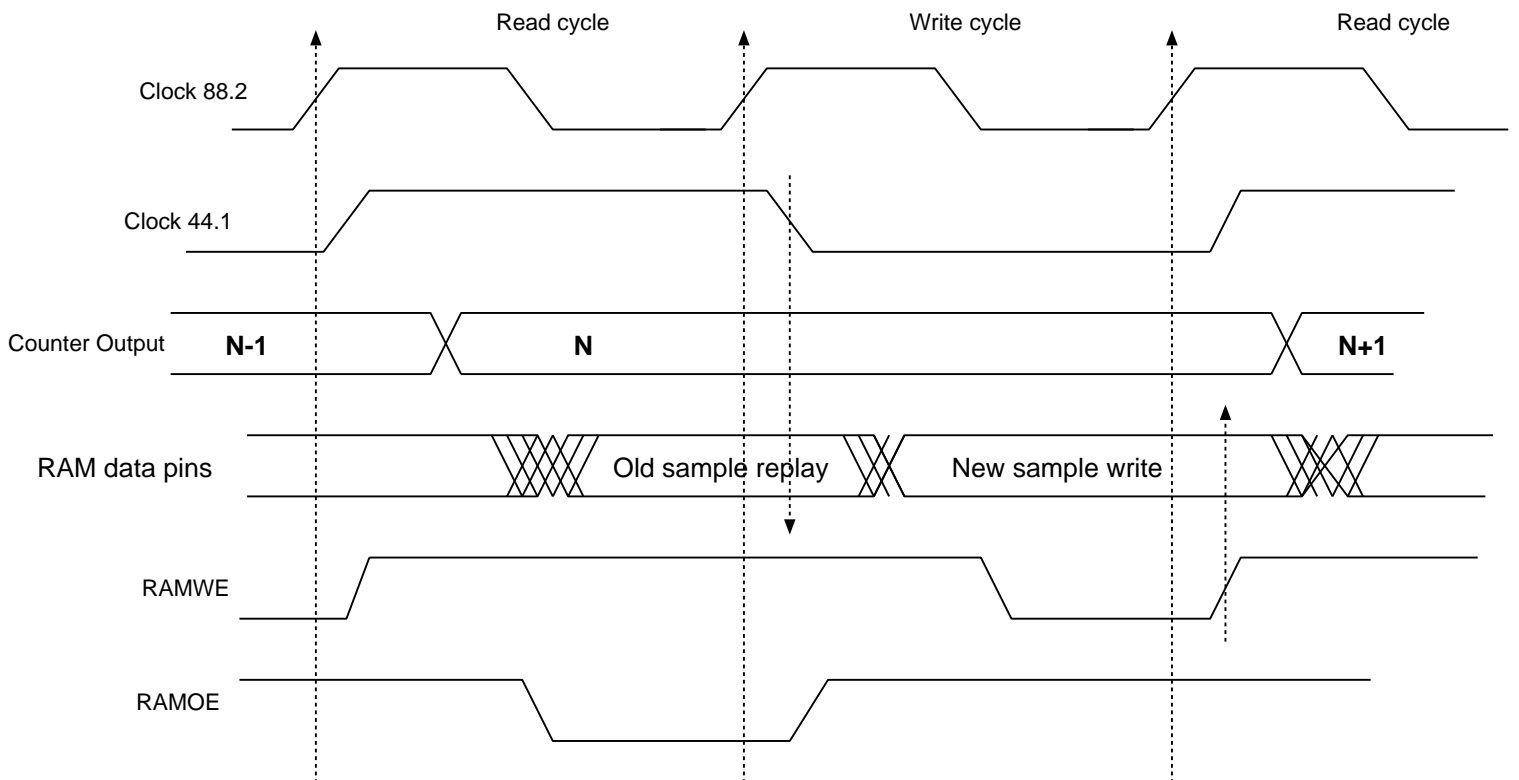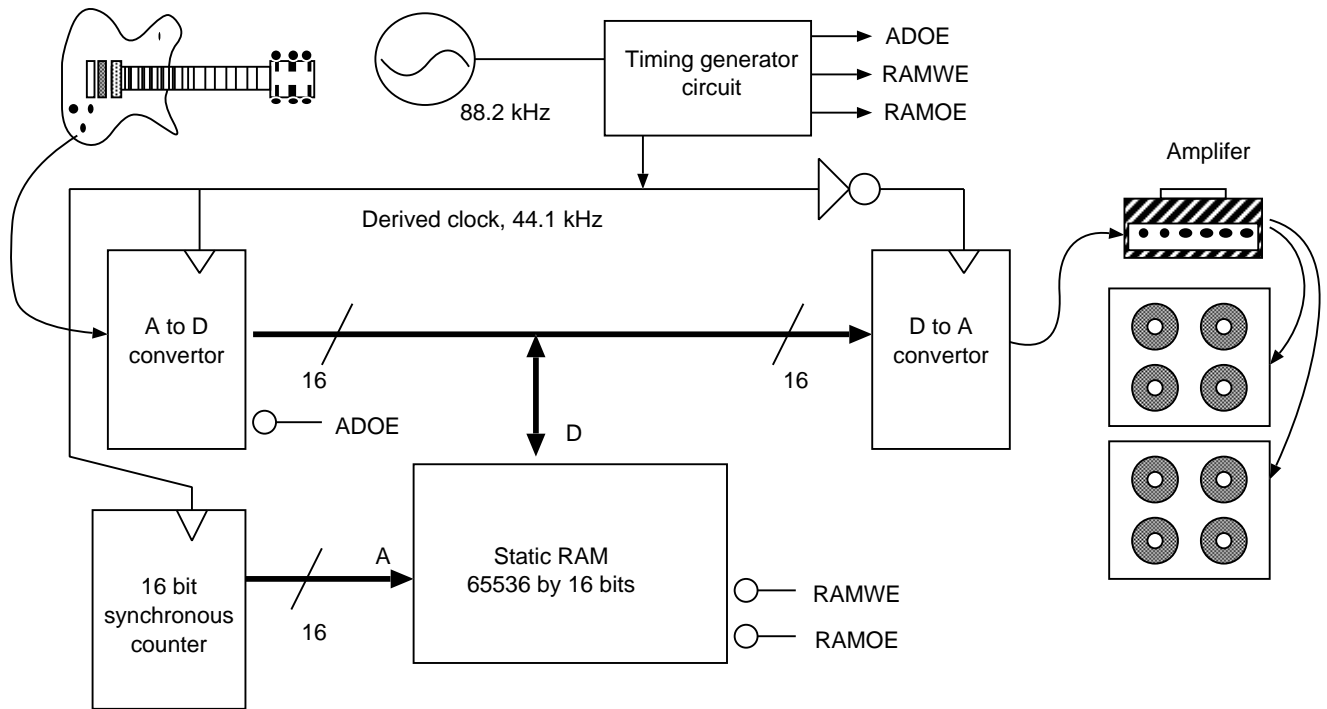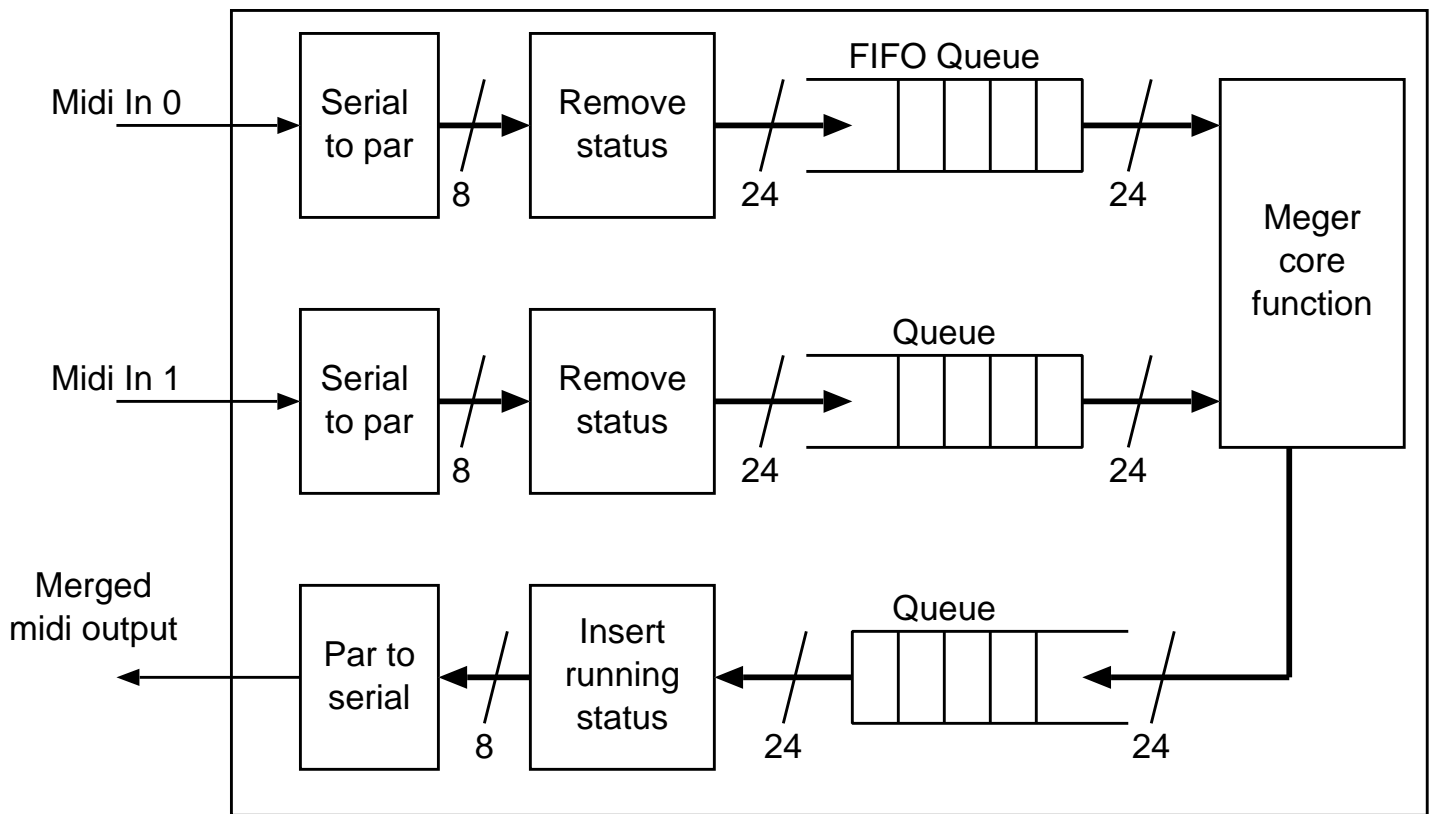
Merge unit block diagram

MIDI serial data format

```
9n kk vv        (note on)
8n kk vv        (note off)
9n kk 00        (note off with zero velocity)
```

MIDI merge unit internal functional units

The serial to parallel converter:

```
input clk;
output [7:0] pardata;    output guard;
```

The running status remover:

```
input clk;
input guard_in;    input [7:0] pardata_in;
output guard_out; output [23:0] pardata_out
```

For the FIFOs:

```
input clk;
input guard_in; input [7:0] pardata_in;
input read; output guard_out;   output [23:0] pardata_out;
input read; output guard_out;   output [23:0] pardata_out;
```

For the merge core unit:

```
input clk;
input guard_in0; input [23:0] pardata_in0; output read0;
input guard_in1; input [23:0] pardata_in1; output read1;
output guard_out; output [23:0] pardata_out;
input read; output guard_out;   output [23:0] pardata_out;
```

Status inserter / parallel to serial converter are reverse of reciprocal units