Structured Hardware Design

Six lectures for CST Part Ia (50 percent).

Easter Term 2004.

Preface

There are a few more slides here than will be used in lectures.

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:

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.

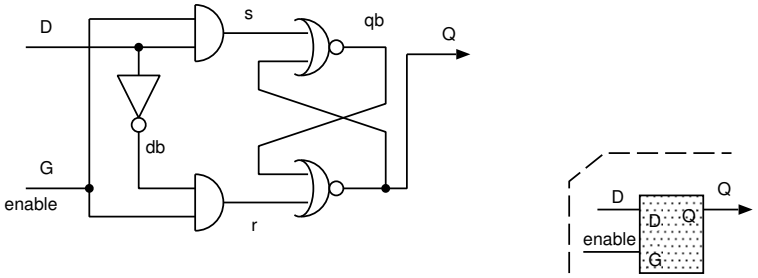
Floyd. *'Digital Fundamentals'* Prentice Hall International.

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

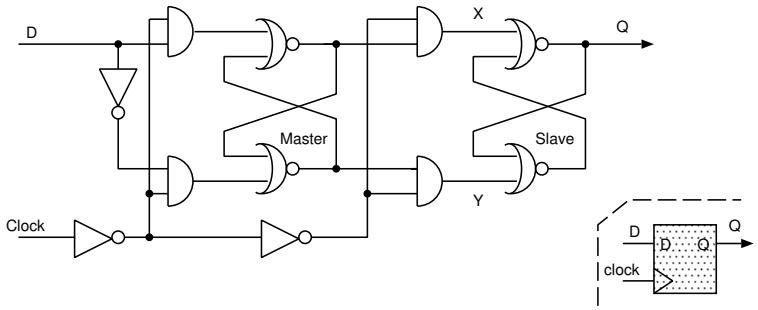
Randy H Katz. *'Contemporary logic design.'*

Flip-Flop Revision

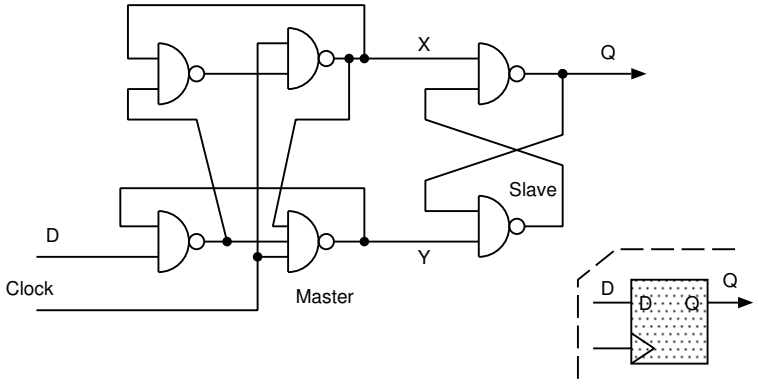
Making a transparent latch from an RS latch:



Putting two together we get the D-type:

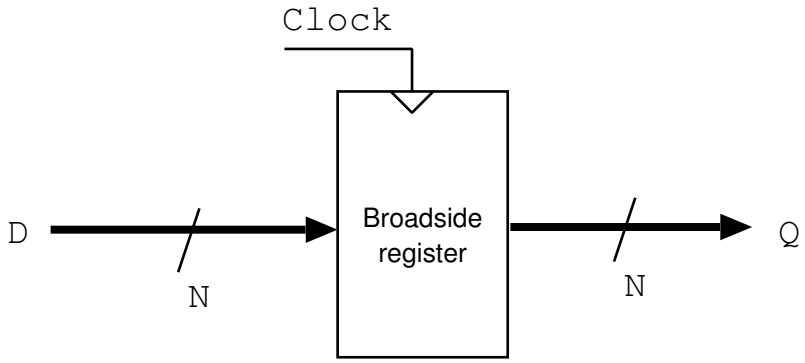


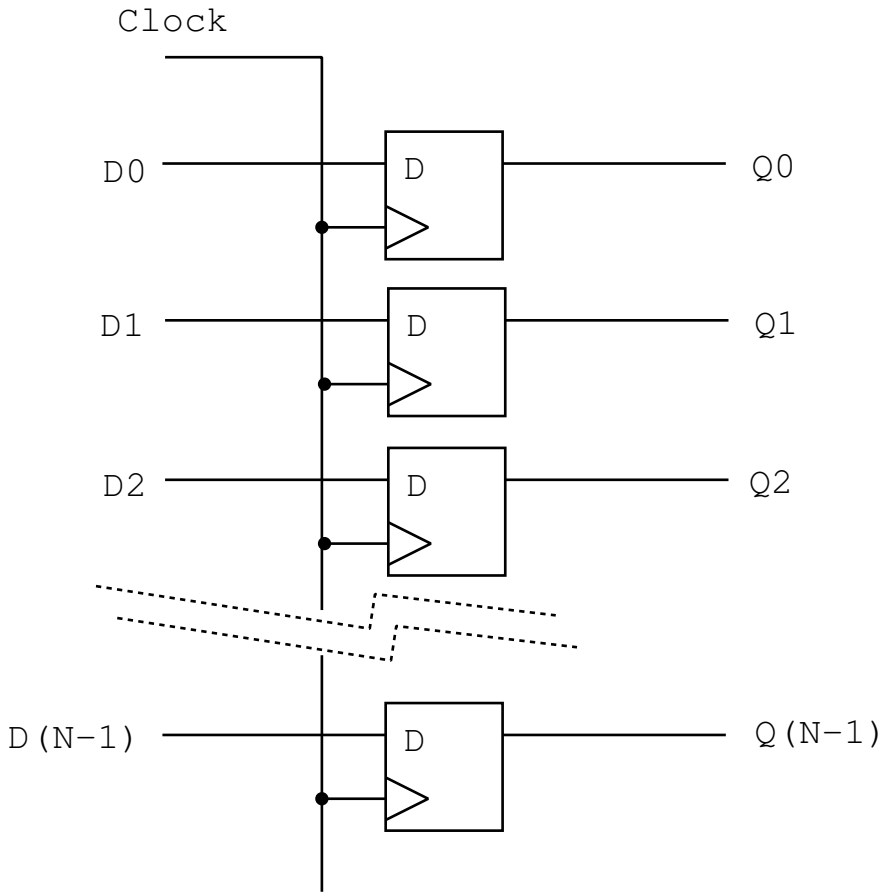
A more optimal circuit:



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

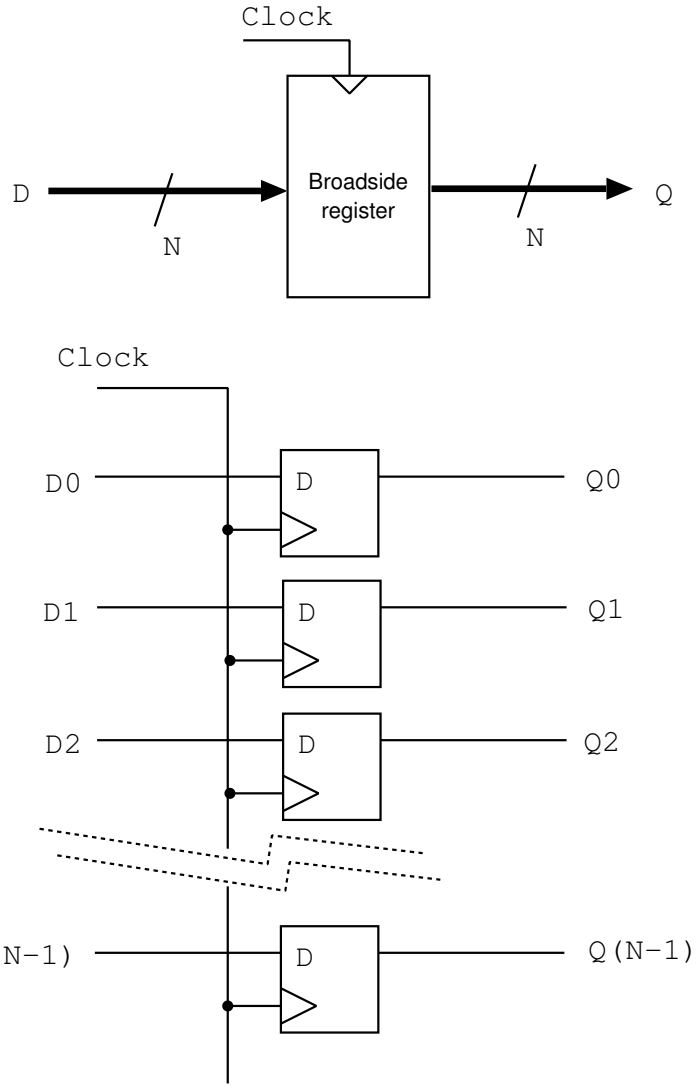
A Broadside Register





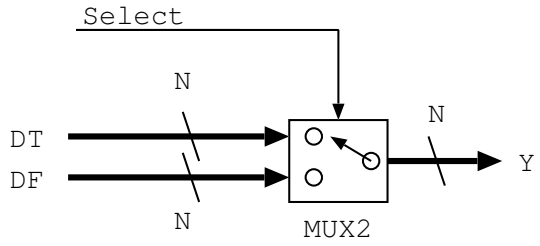
A broadside register of N bits is made out of N D-types with a commoned clock input. It can hold 2^N different values.

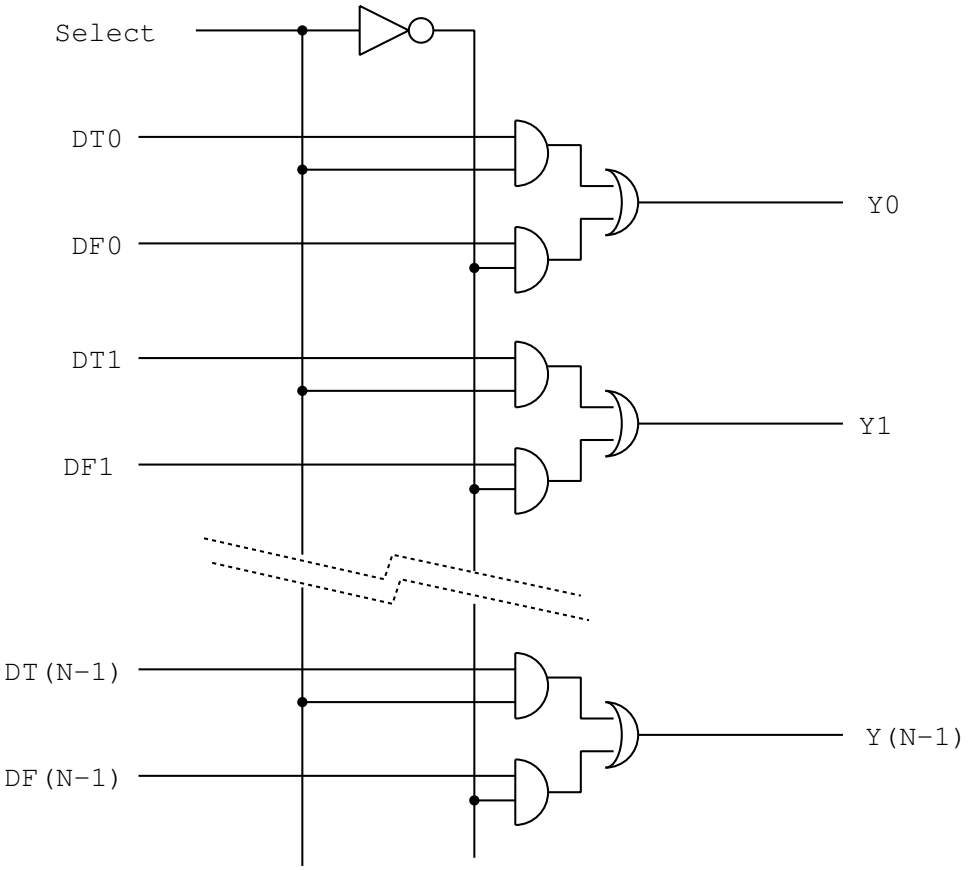
A Broadside Register - Verilog




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

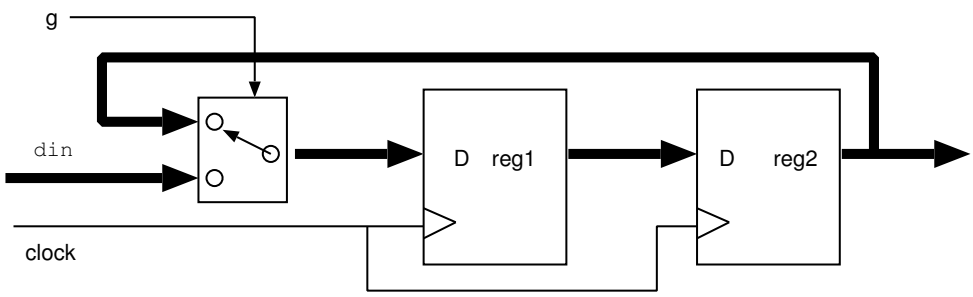
A broadside two-to-one multiplexor






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

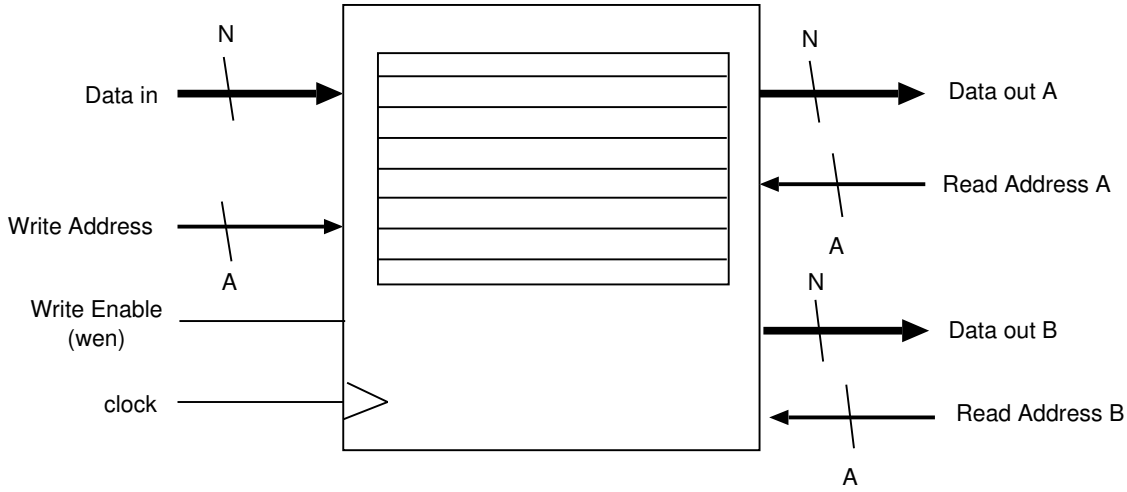
A Simple Structure



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
```

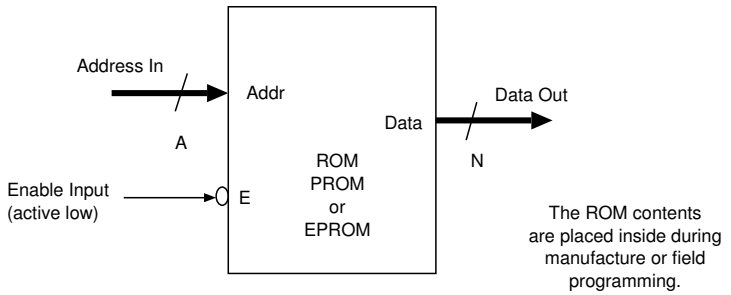
A Dual-Port Register File



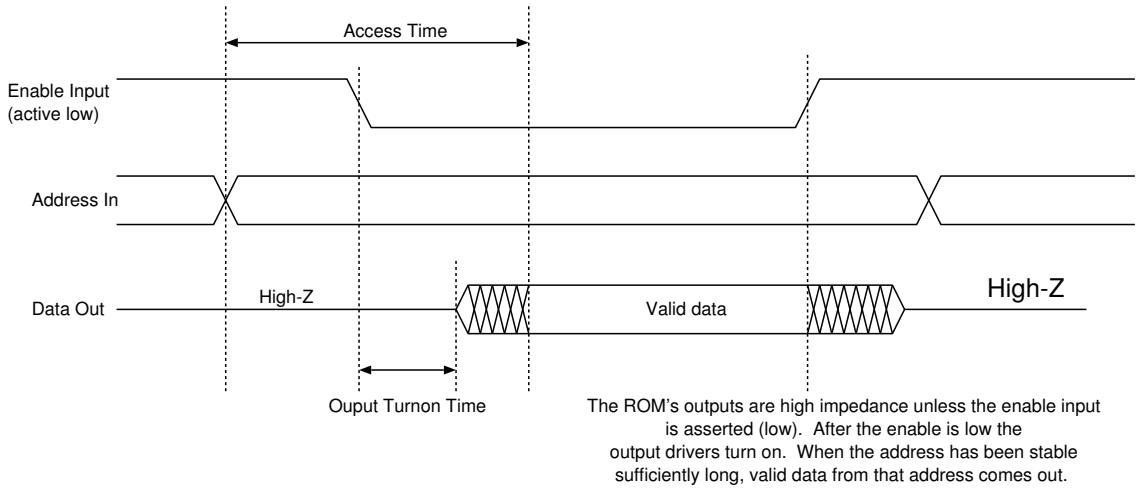

```
// Verilog for a dual-read ported register file.
 input [3:0] write_address, read_address_a,
             read_address_b;
 reg [7:0] regfile [15:0]
 always @(posedge clk) begin
       if (wen) regfile[write_address] <= din;
       end

 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 Only Memory (ROM)



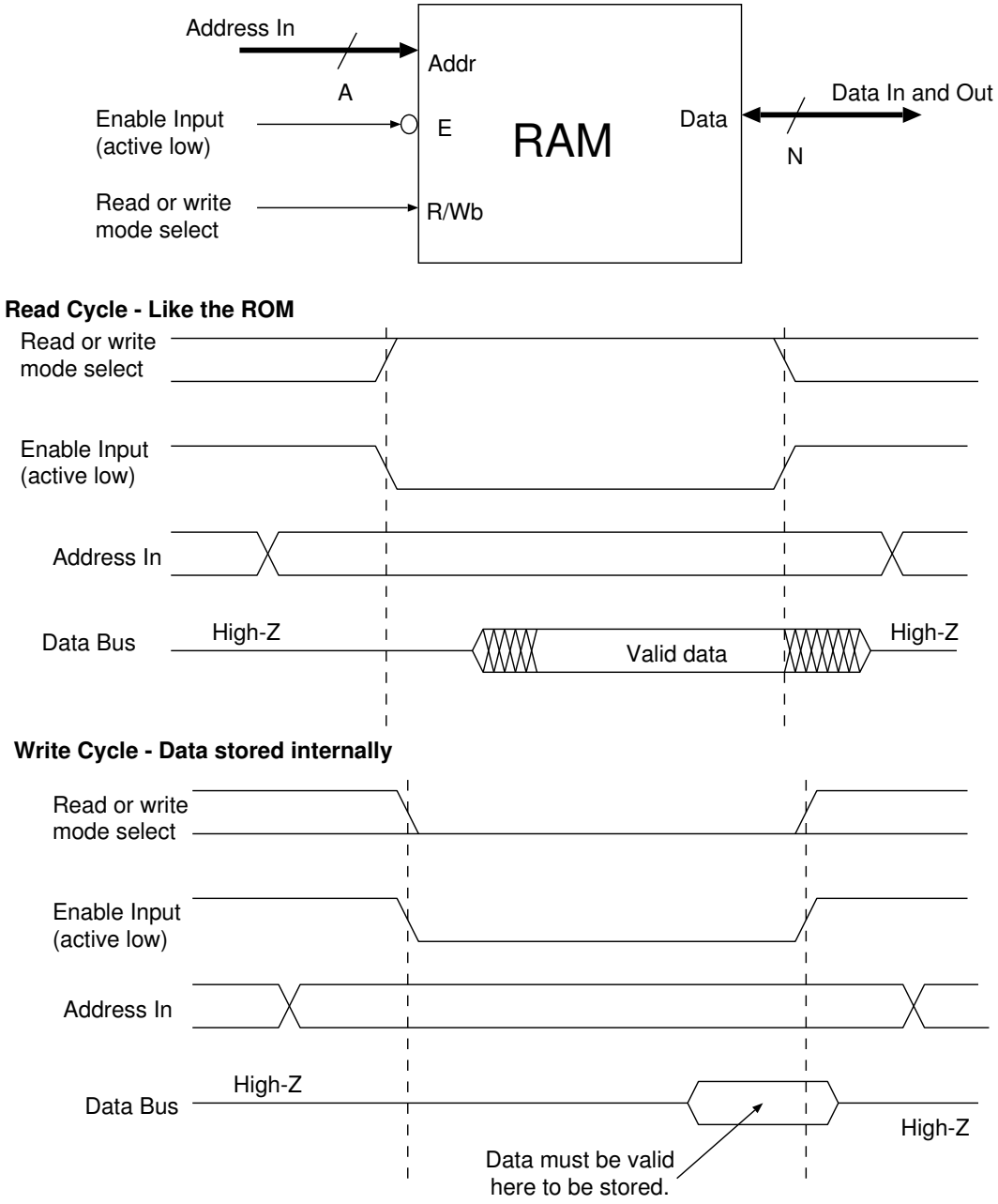
The ROM takes A address bits named A0 to A<A-1> and produces data words of N bits wide. For example, if A=5 and D=8 then the ROM contains 2**5 which is 32 locations of 8 bits each. The address lines are called A0, A1, A2, A3, A4 and the data lines D0, D1, ... D7



MASKED PROGRAMMED means contents inserted at time of manufacture.

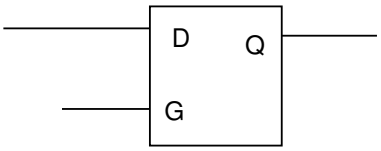
FLASH PROM uses static electricity on floating transistor gates.

Read/Write Memory (RAM)

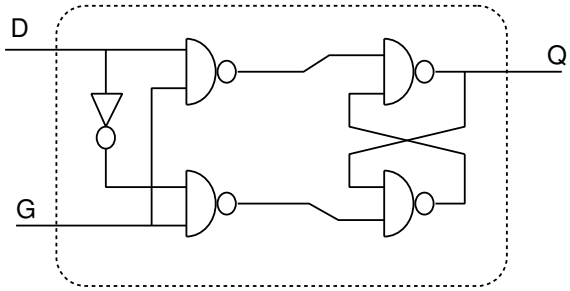


Each data bit internally stored in an RS latch.

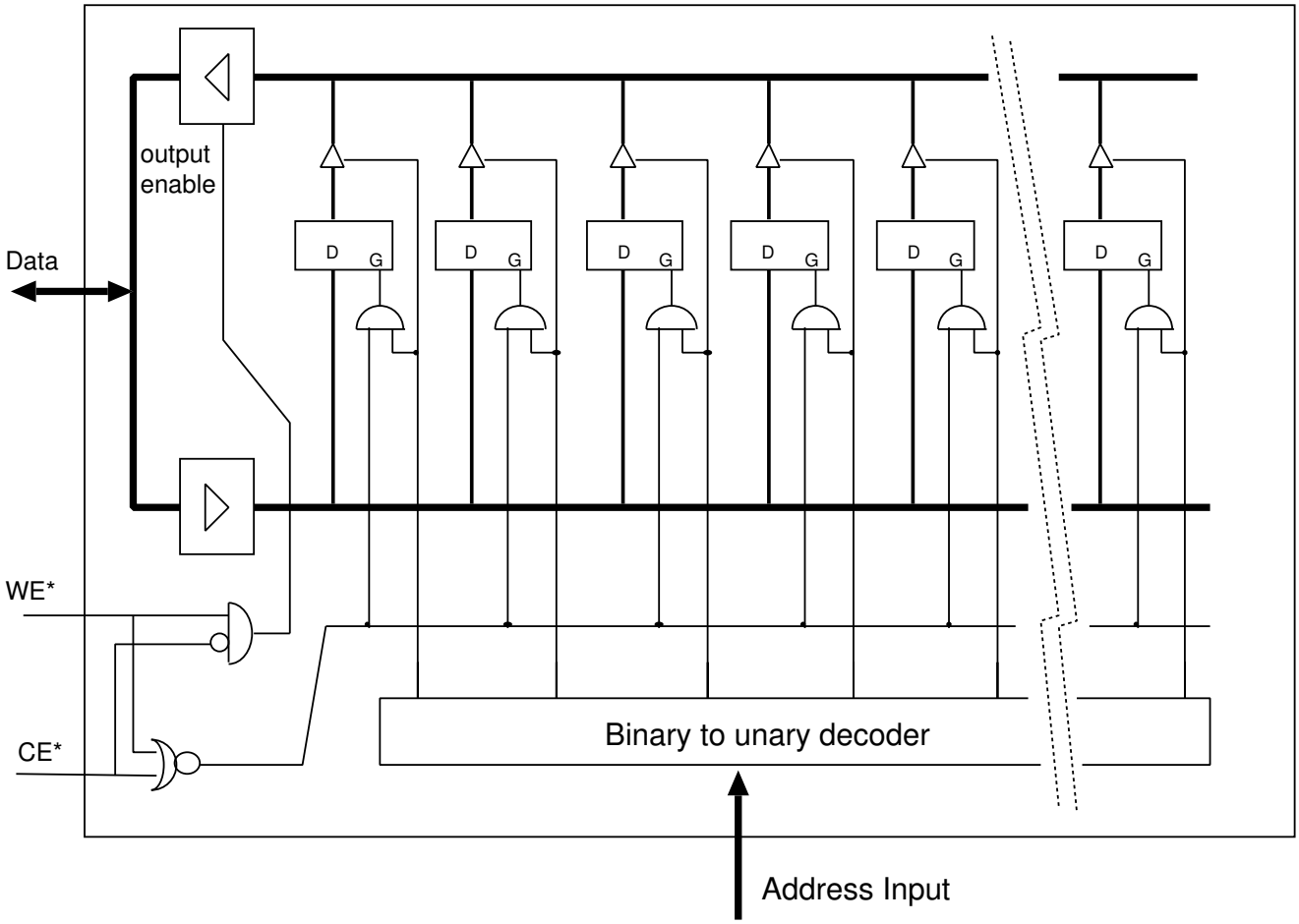
Unlike the edge-triggered flip-flop, the transparent latch passes data through in a transparent way when its enable input is high. When its enable input is low, the output stays at the current value.



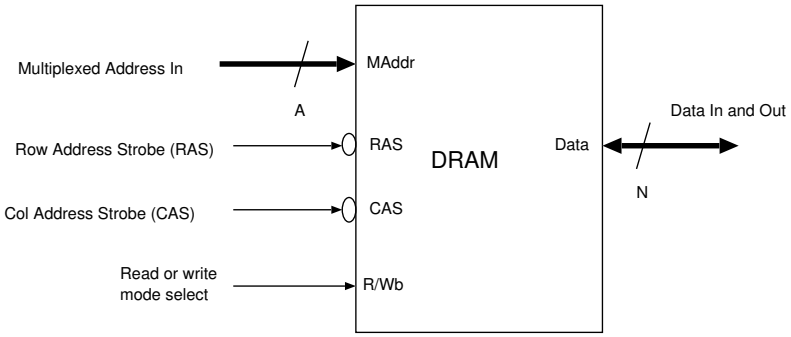
Transparent latch schematic symbol



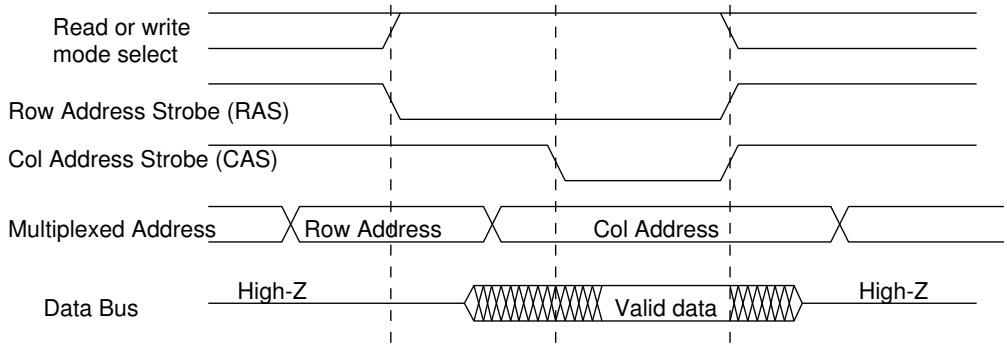
Transparent latch implemented from gates.



DRAM



Read Cycle (write is similar)

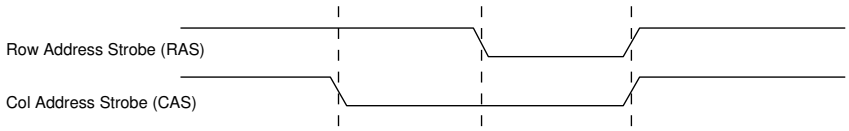


A DRAM has a multiplexed address bus and the address is presented in two halves, known as row and column addresses. So the capacity is 4**A x D. A 4 Mbit DRAM might have A=10 and D=4.

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.

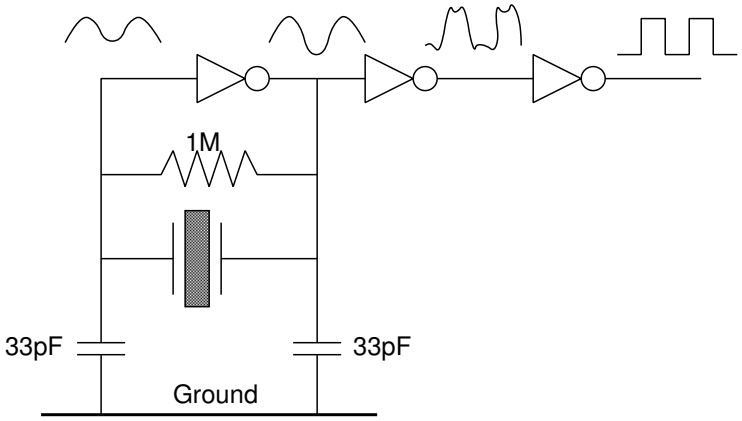
Refresh Cycle - must happen sufficiently often!



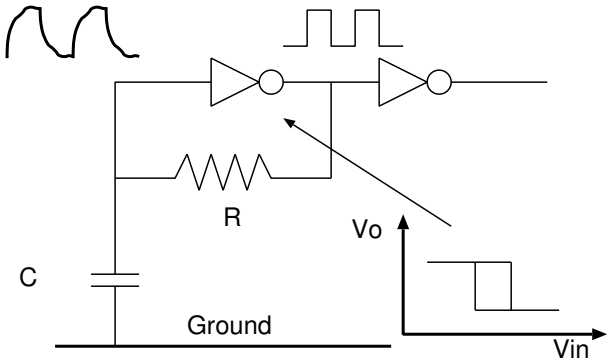
No data enters or leaves the DRAM during refresh, so it 'eats memory bandwidth'.
Typically 512 cycles of refresh must be done every 8 milliseconds.

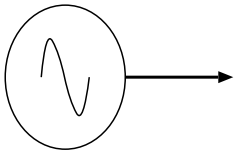
Modern DRAM has a clock input at 200 MHz and transfers data on both edges.

Crystal oscillator clock source



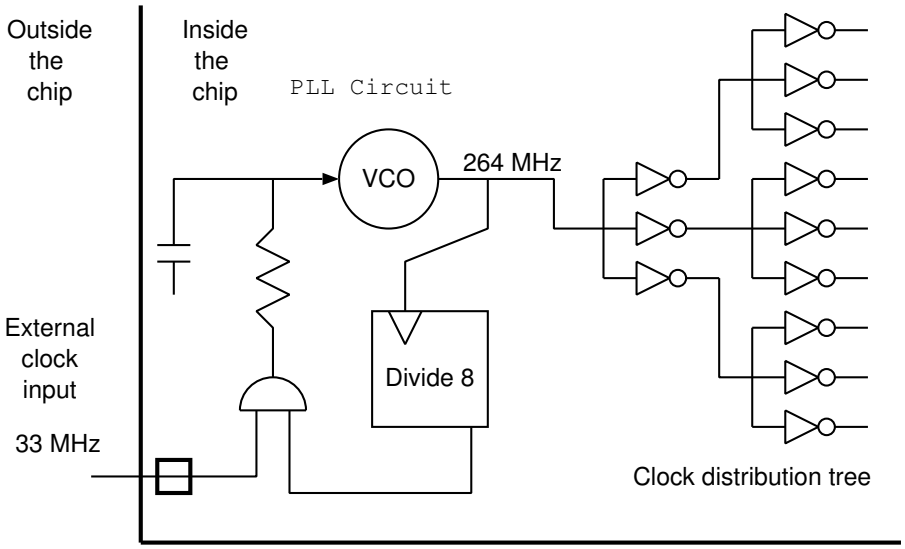
RC oscillator clock source



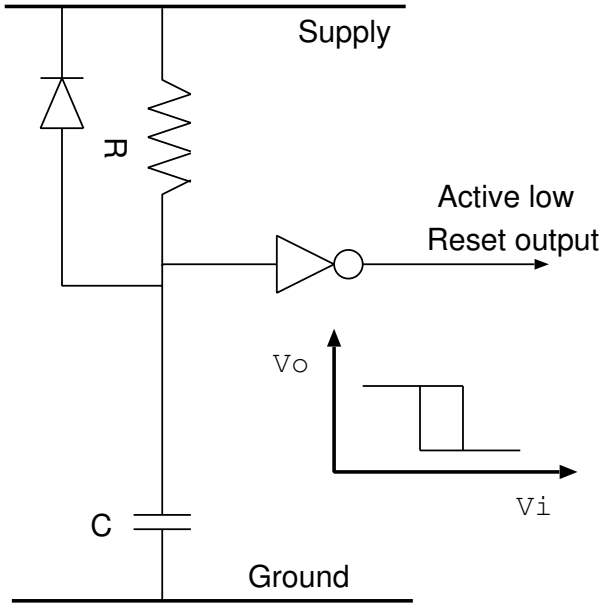


Schematic Symbol

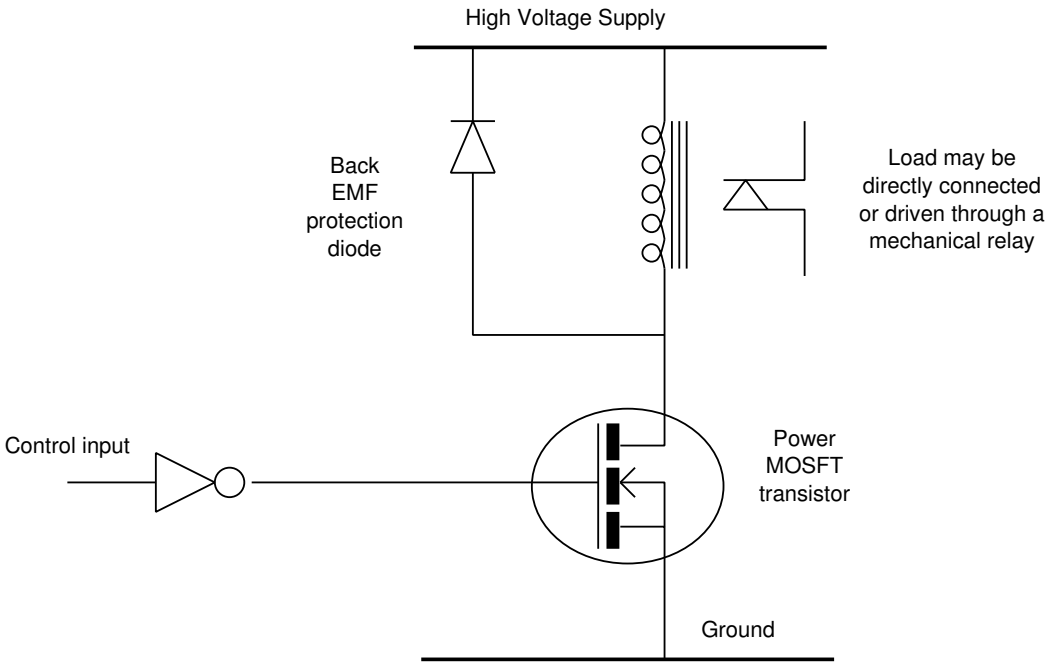
Clock multiplication and distribution



Power-on reset

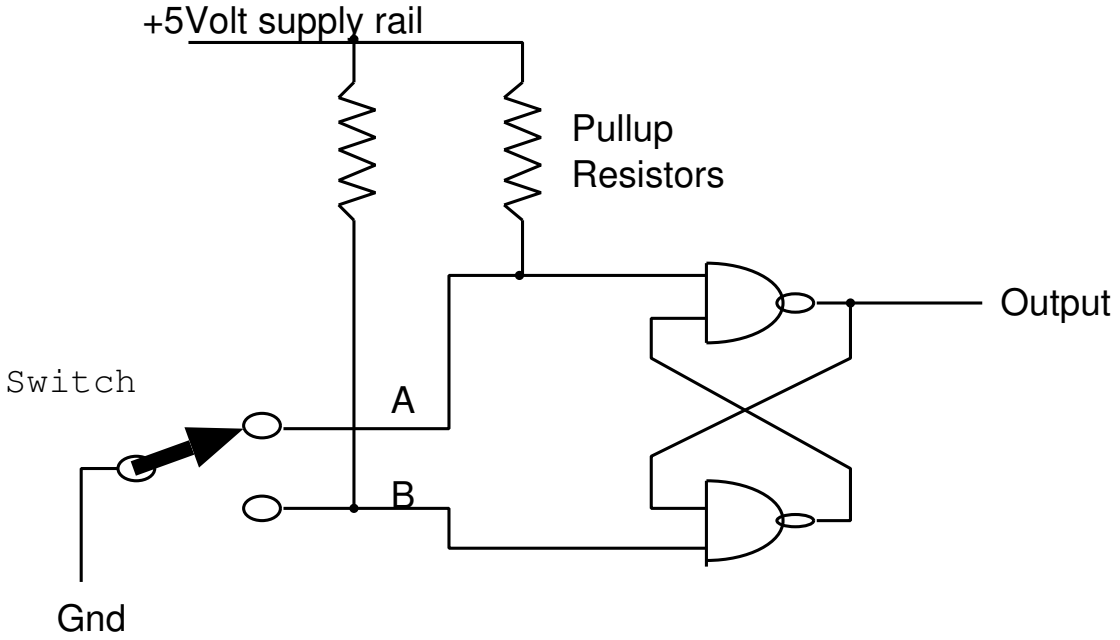


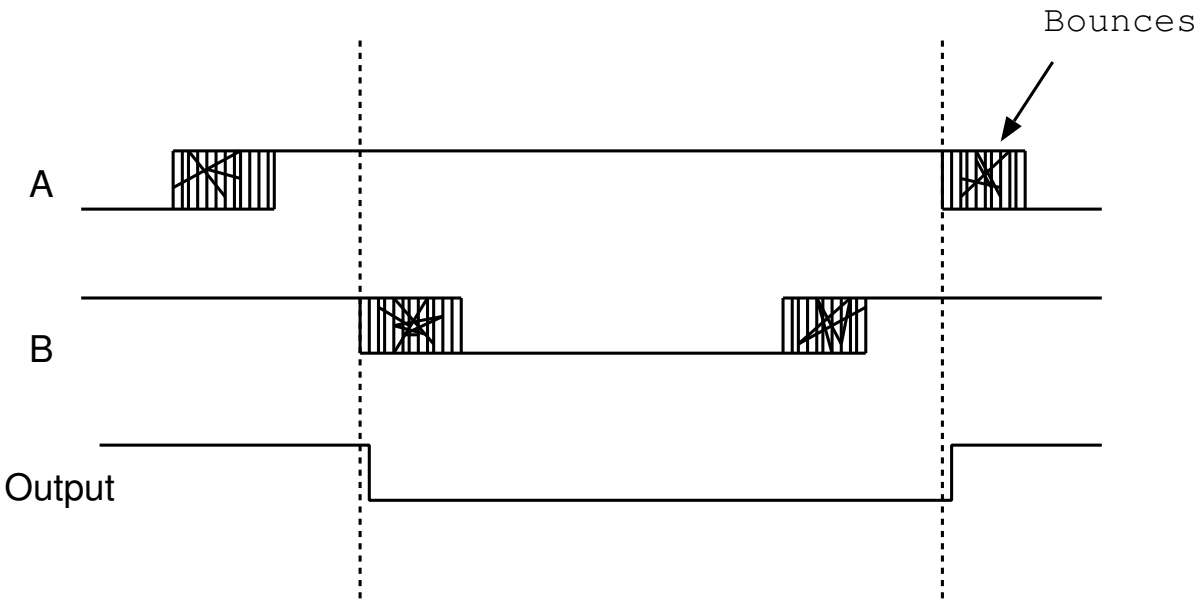
Driving a heavy current or high-voltage load



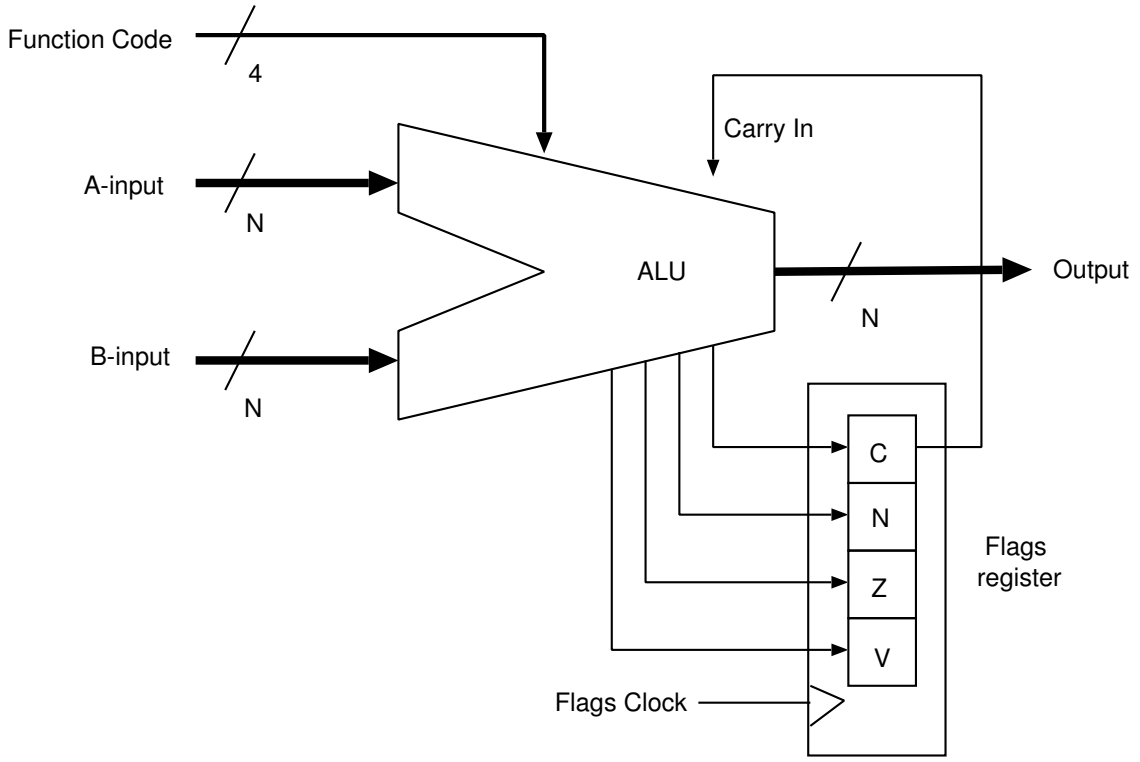
Transistor active area could be 1 square centimeter.

Debouncer circuit for a two-pole switch



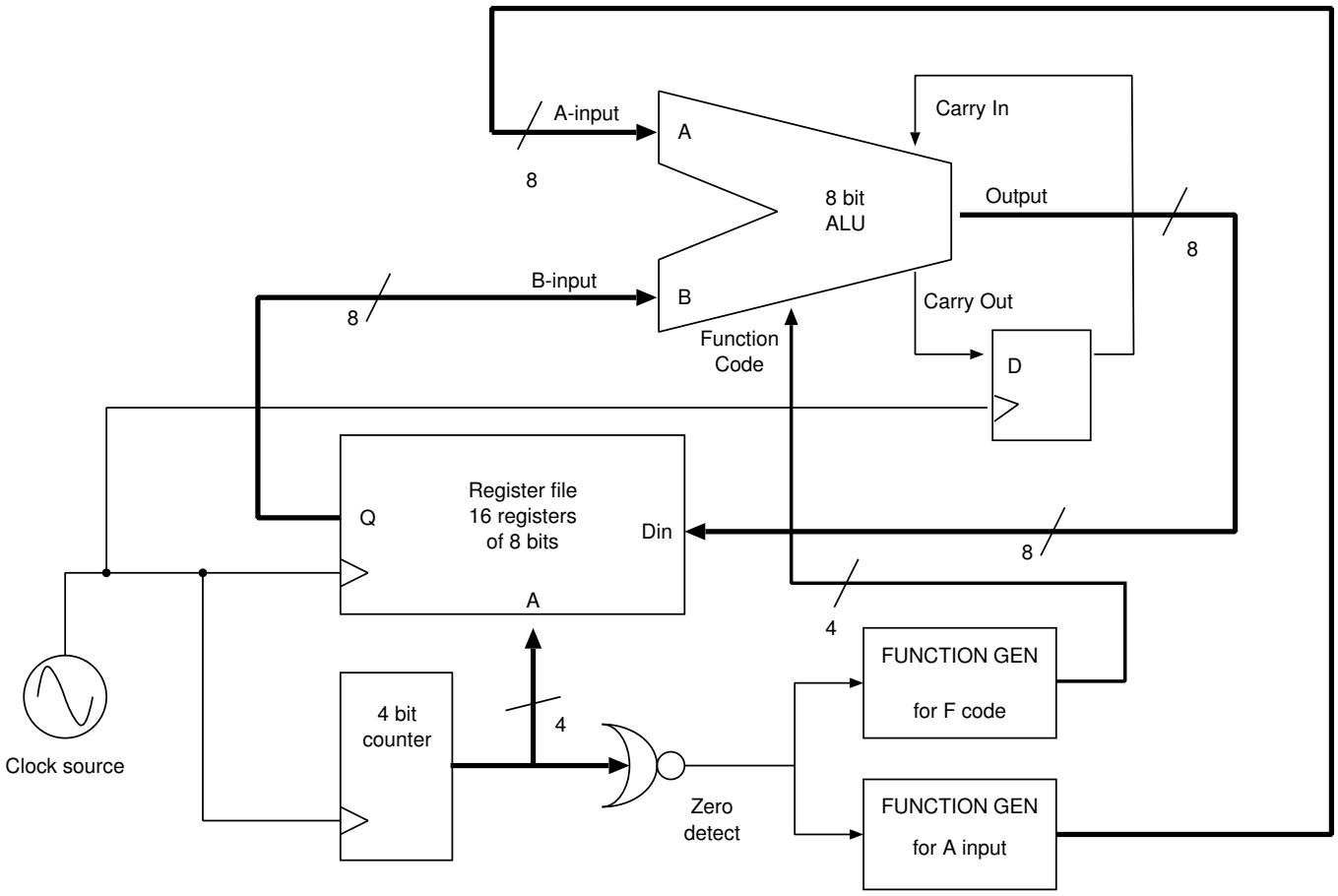


ALU and Flags Register




```
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[7];
```

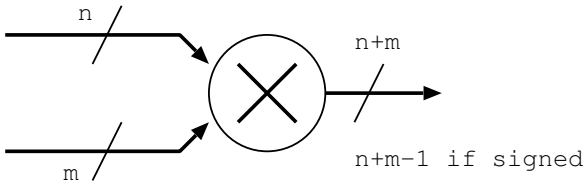
ALU and Register File



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



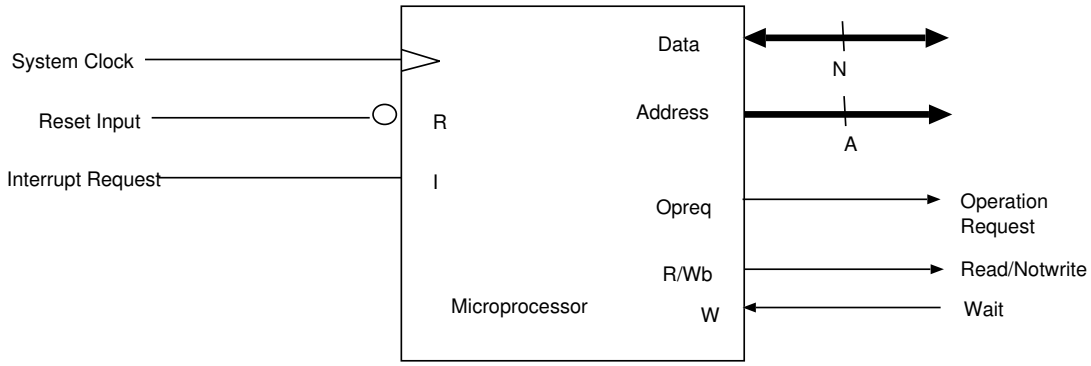
FLASH MULTIPLIER - combinatorial implementation (e.g. a Wallace Tree).

BOOTH MULTIPLIER - Does two bits per clock cycle:

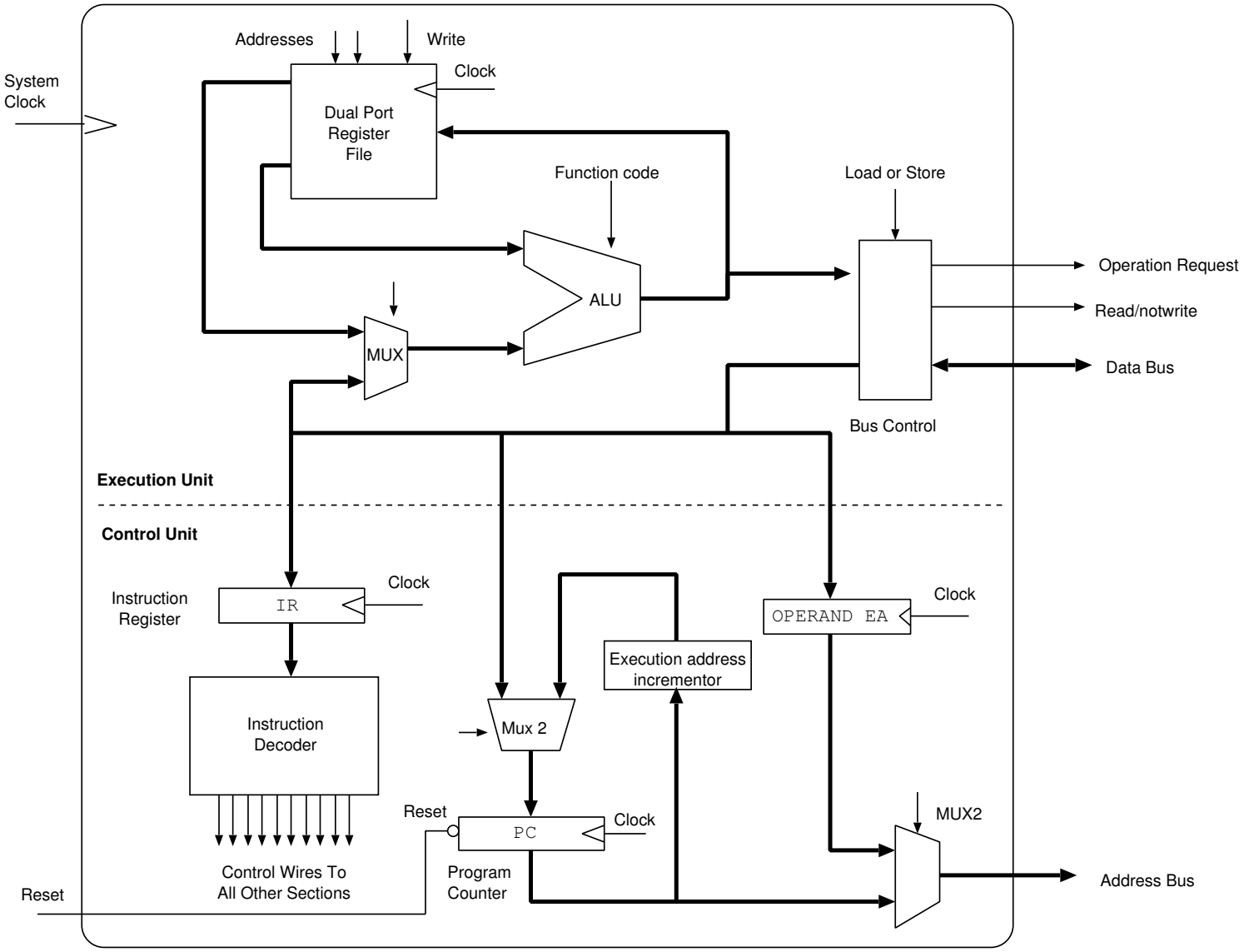
```
(* Call this function with c=0 and carry=0 to multiply
    x by y. *)
fun booth(x, y, c, carry) =
    if(x=0 andalso carry=0) then c else
let val x' = x div 4
    val y' = y * 4
    val n  = (x mod 4) + carry
    val (carry', c') = case (n) of
      (0) => (0, c)
     |(1) => (0, c+y)
     |(2) =>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

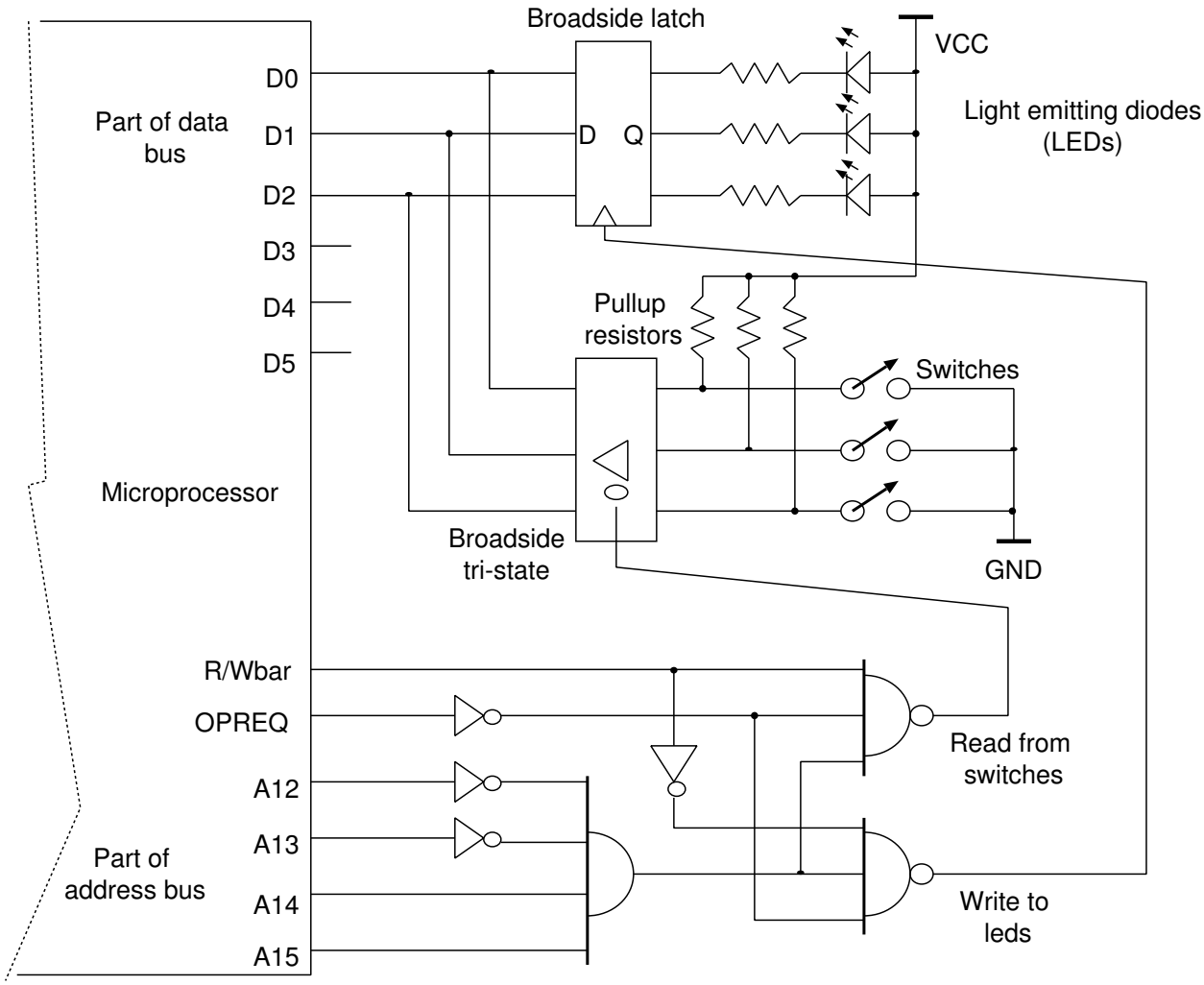
Ex: Design a controller for an ALU and register file to implement Booth.

Logic Symbol



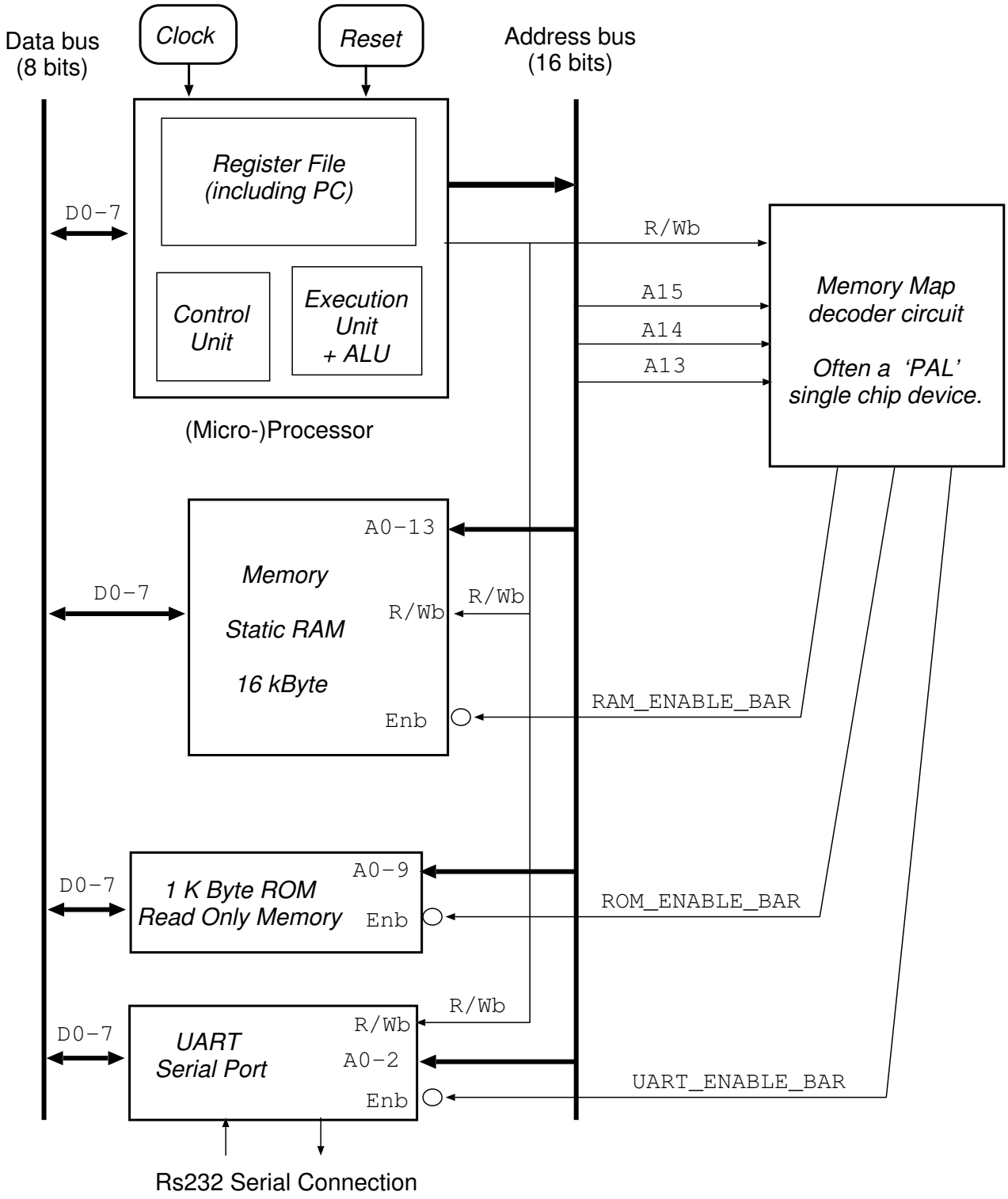
Internal Structure Block Diagram



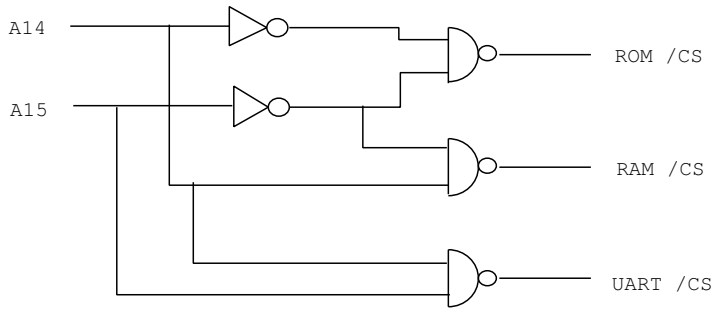


Example of memory address decode and simple LED and switch interfacing for programmed IO (PIO) to a microprocessor.

A D8/A16 Computer



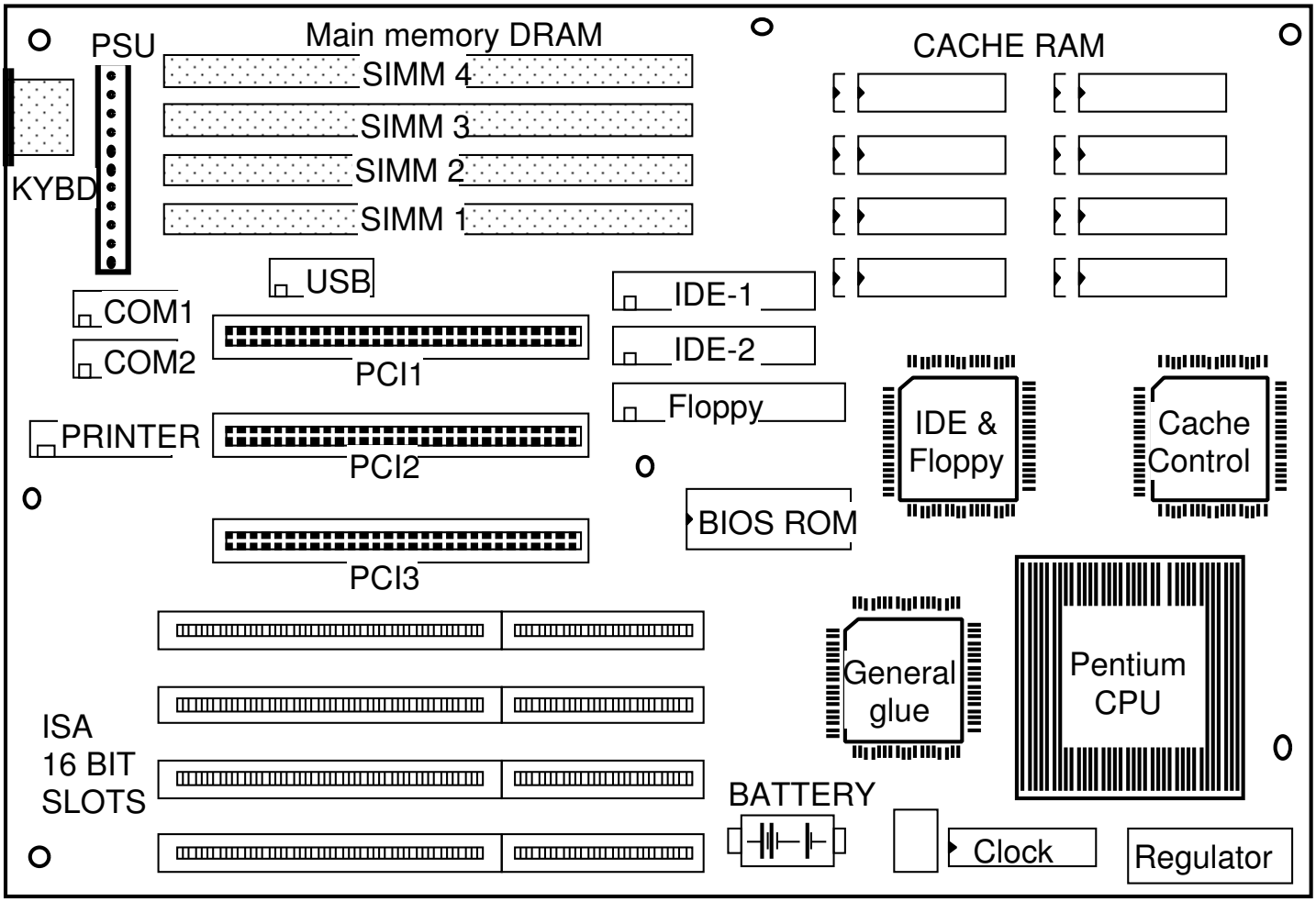
Memory Address Mapping



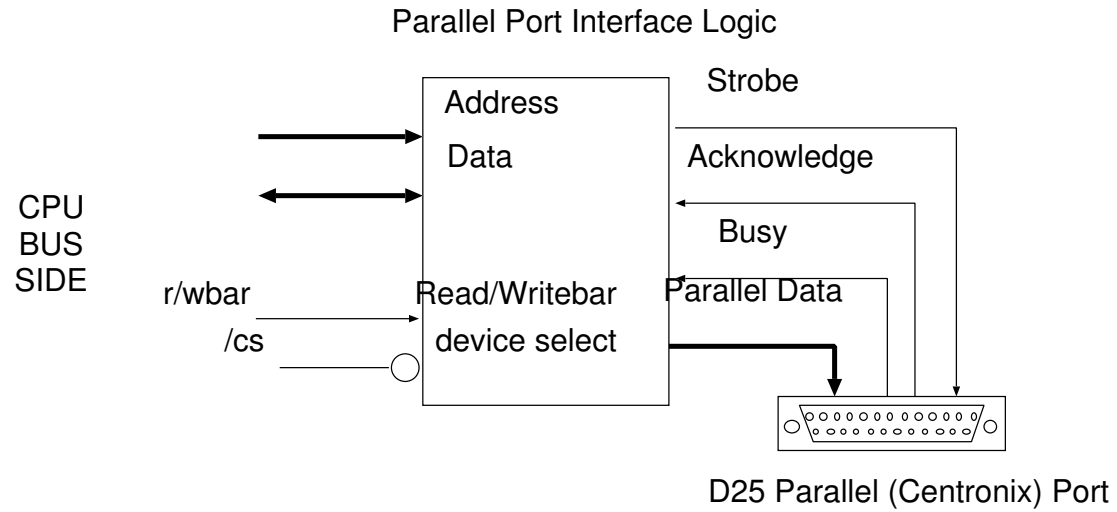
Start	End	Resource
0000	03FF	EPROM
0400	3FFF	Unused images of EPROM
4000	7FFF	RAM
8000	BFFF	Unused
C000	C001	Registers in the UART
C002	FFFF	Unused images of the UART

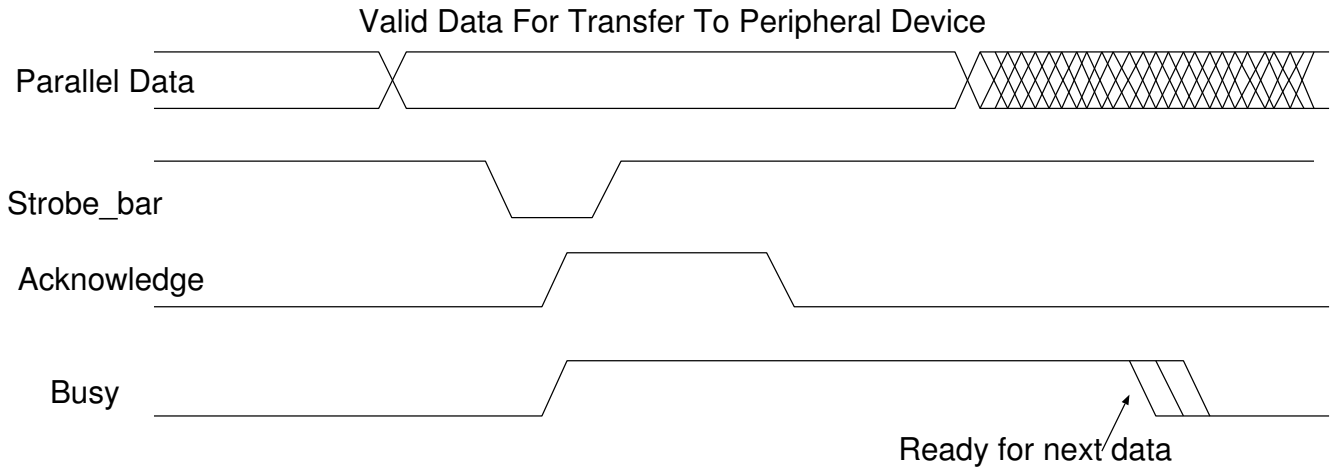
```
module address_decode(abus, rom_cs, ram_cs, uart_cs);
   input [15:14] abus;
   output rom_cs, ram_cs, uart_cs);

   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



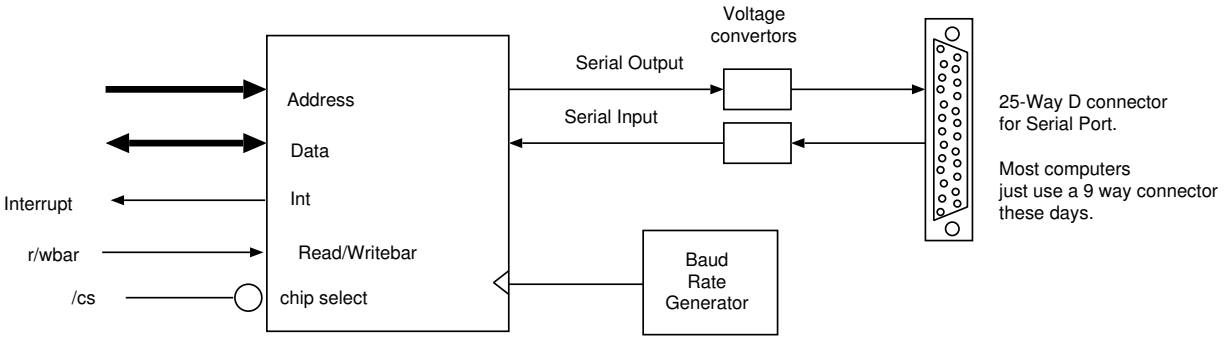
Parallel Port

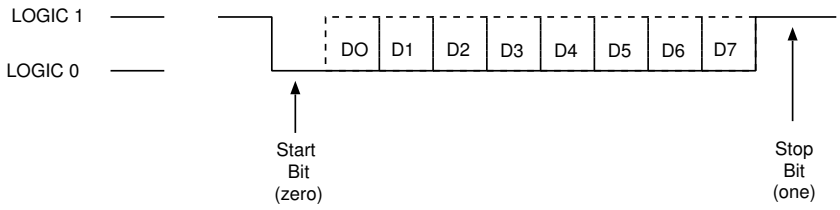




Flow control: New data is not sent while the busy wire is high.

Serial Port (UART)





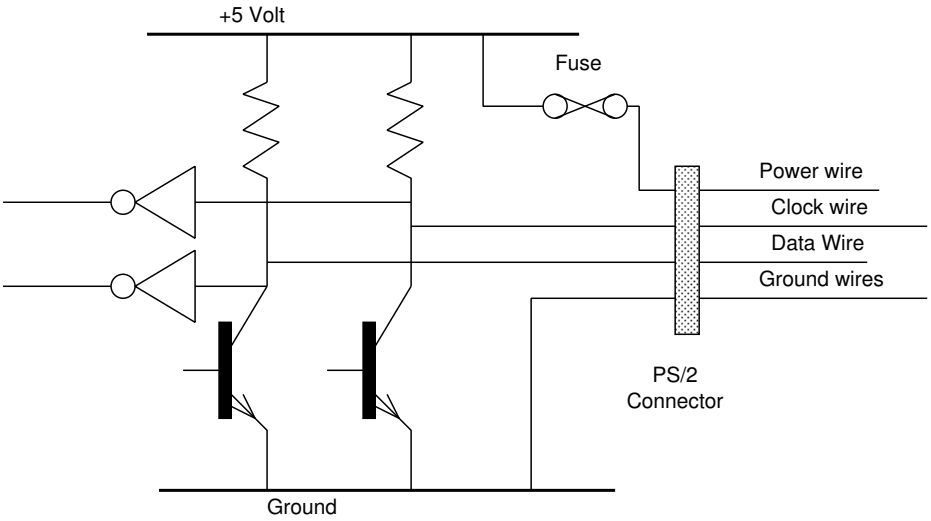
Flow control: New data can be sent at any time unless either:

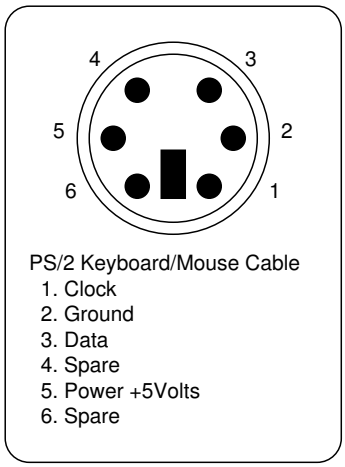
additional signals are used to indicate clear to send

or

a software protocol is defined to run on top (Xon/Xoff) by reserving certain of the bytes.

Keyboard and/or PS/2 port





Open collector wiring using two signalling wires.

The 1394 Firewire and USB ports are essentially the same as this.

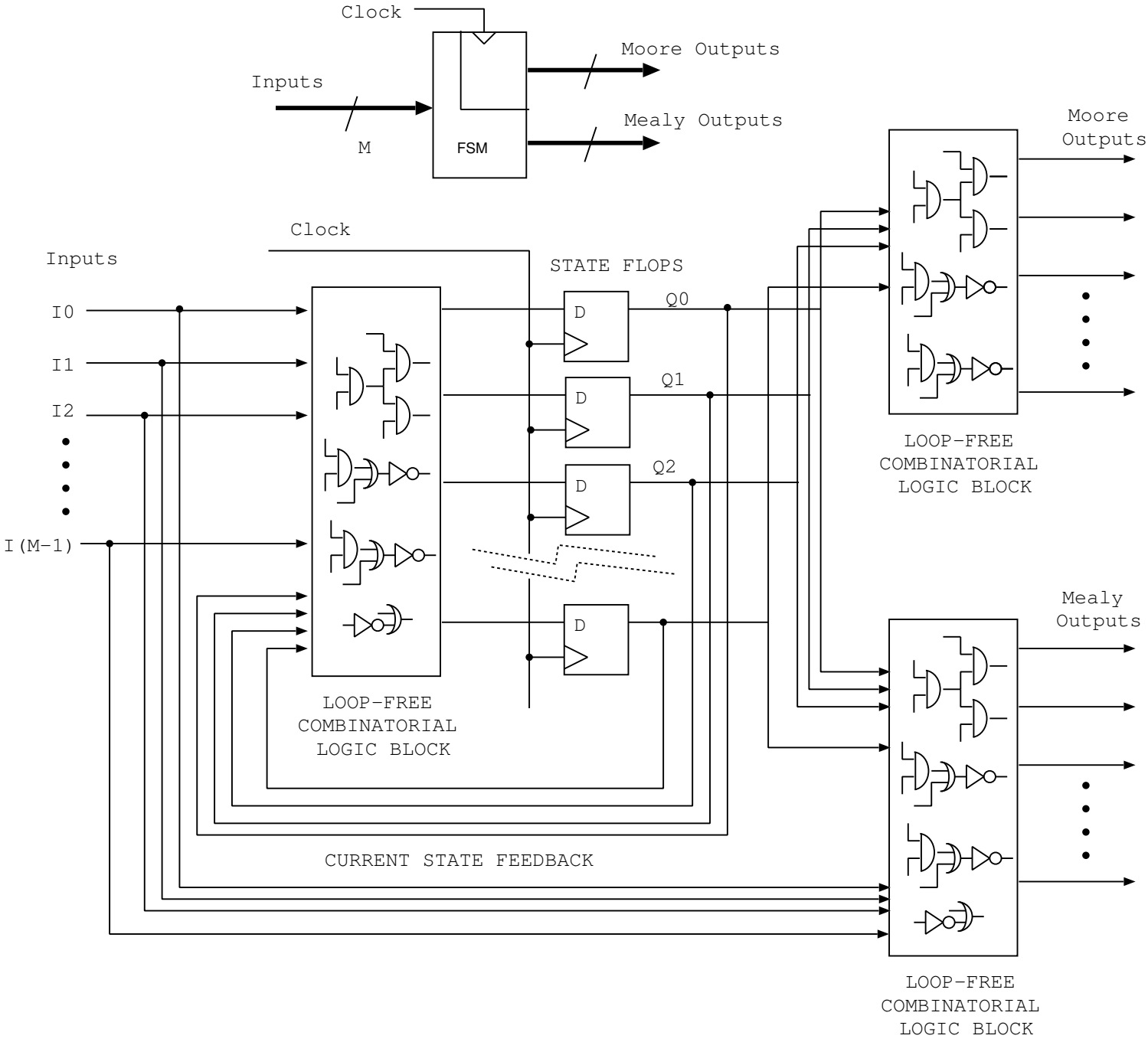
Canonical Synchronous FSM

FSM = { Set of Inputs, Set of states Q, Transiton function D)

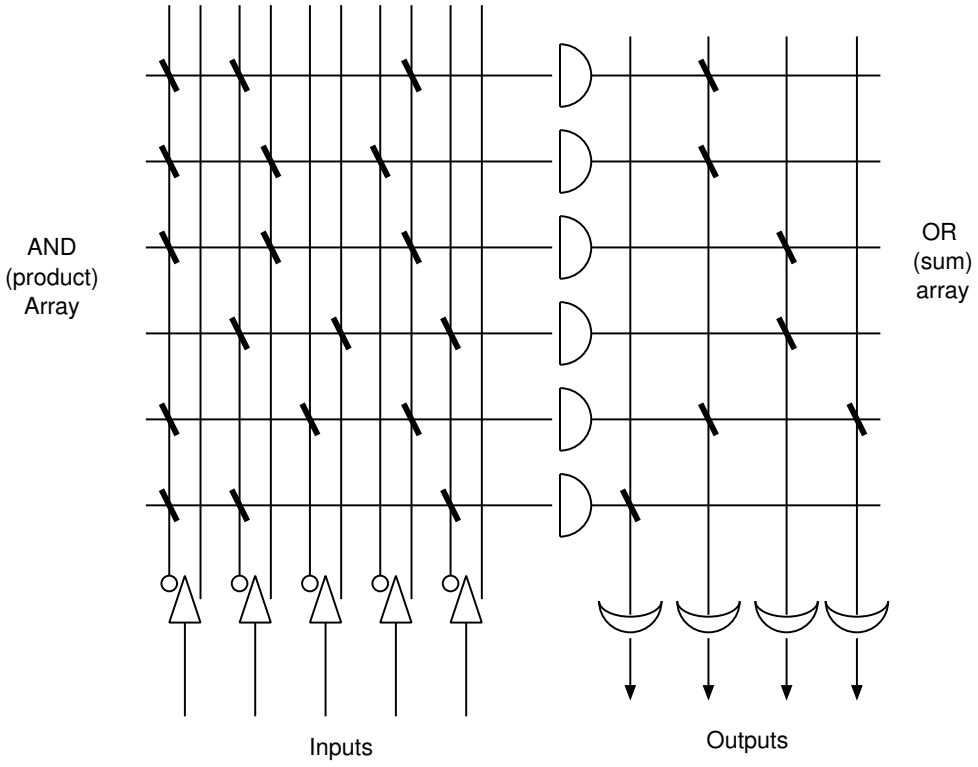
An initial state can be jumped to by terming one of the inputs a reset.

An accepting state would be indicated by a single Moore output.

In hardware designs, we have multiple outputs of both Mealy and Moore style.



Canonical Logic Array



Combinational Logic Minimisation

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

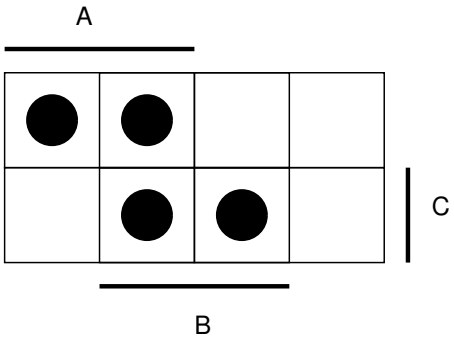
Where two min-terms differ in one literal, they can alway be combined:

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

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

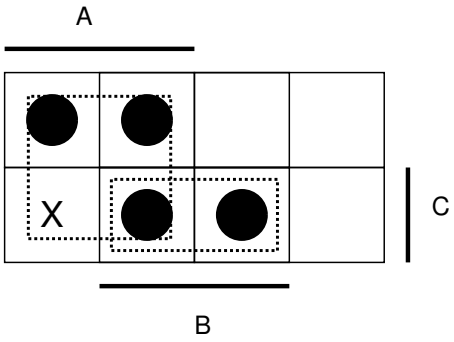
Lookup ‘Kline-McClusky’ for more information.

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:

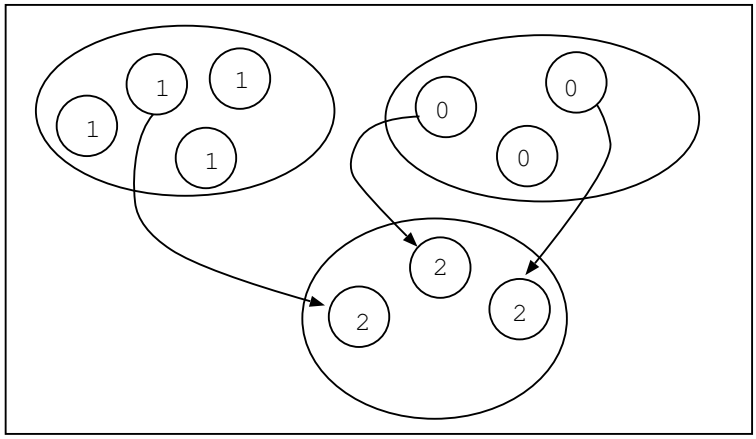



```
(A & ~C) + (A & B) + (B & C)  -->
                            A  + (B & C)
```

Lookup 'ESPRESSO' for more information.

Sequential Logic Minimisation

A finite state machine may have more states than it needs to perform its observable function.



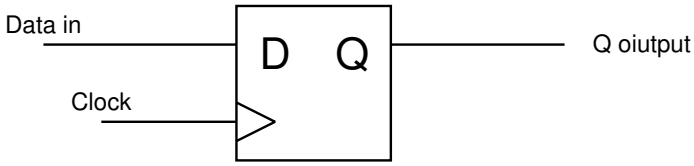
A Moore machine can be simplified by the following procedure

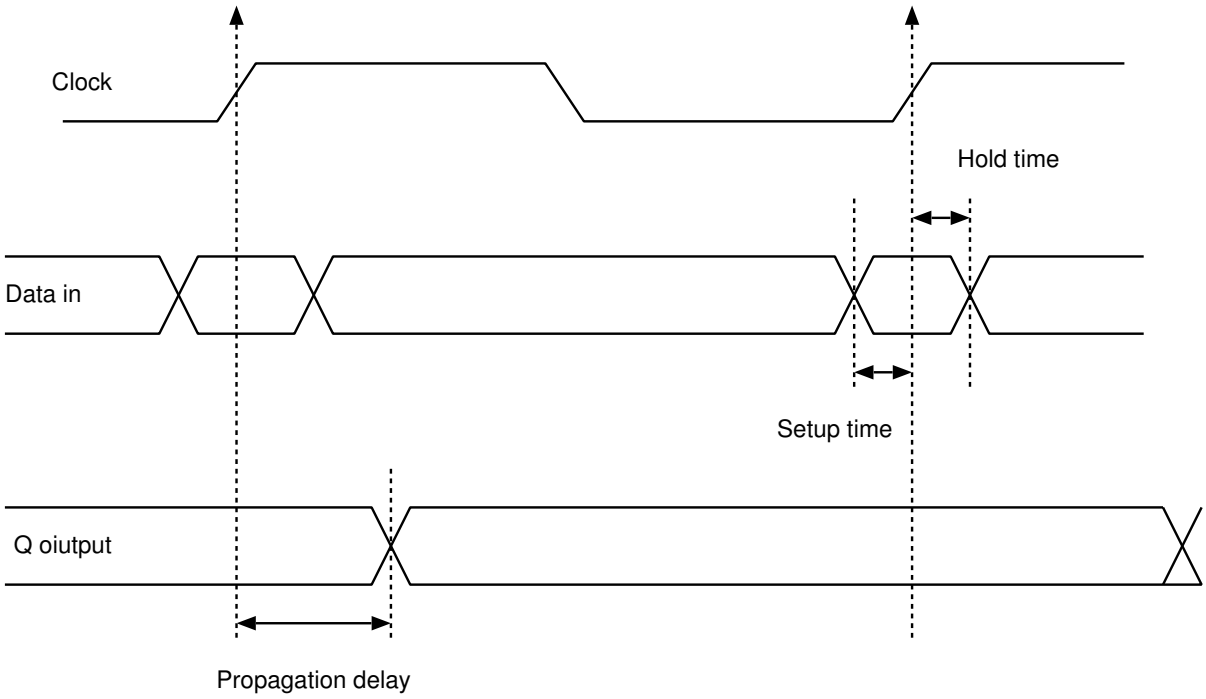
1. Partition all of the state space into blocks of states where the observable outputs are the same for all members of a block.

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.

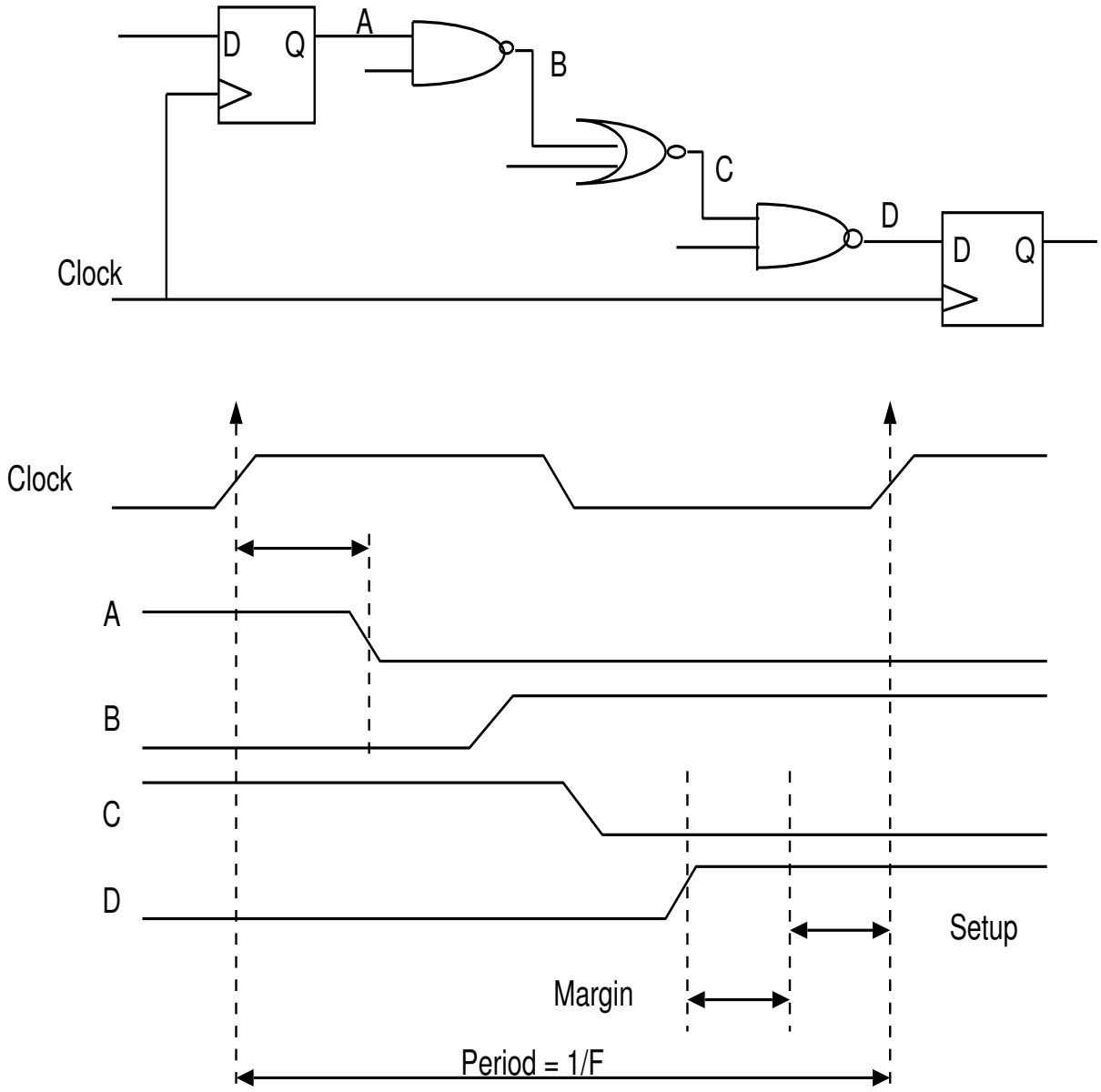
3. The final blocks are the new states.

Timing Specifications



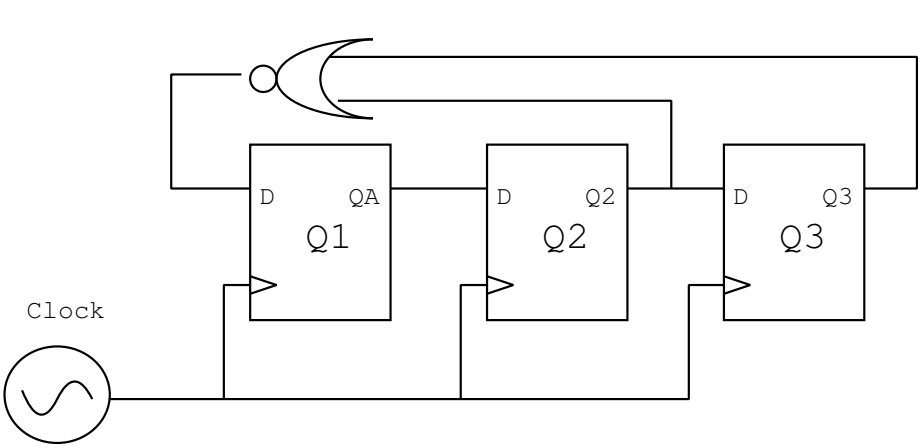


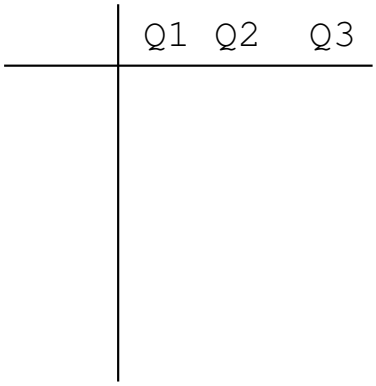
Typical Nature of a Critical Path



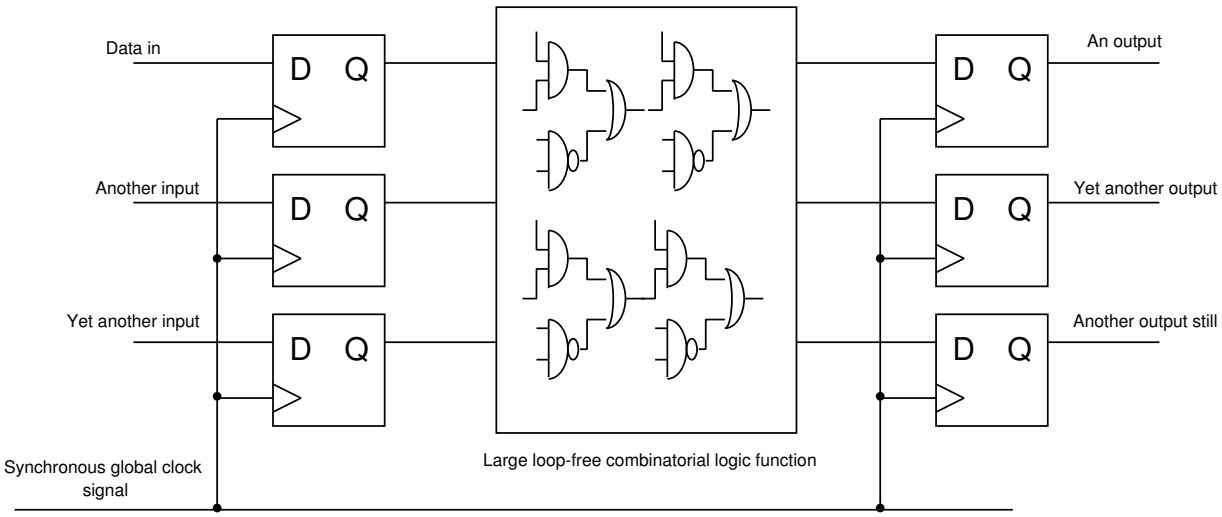
Clock speed can be increased while margin is positive.

Johnson counters

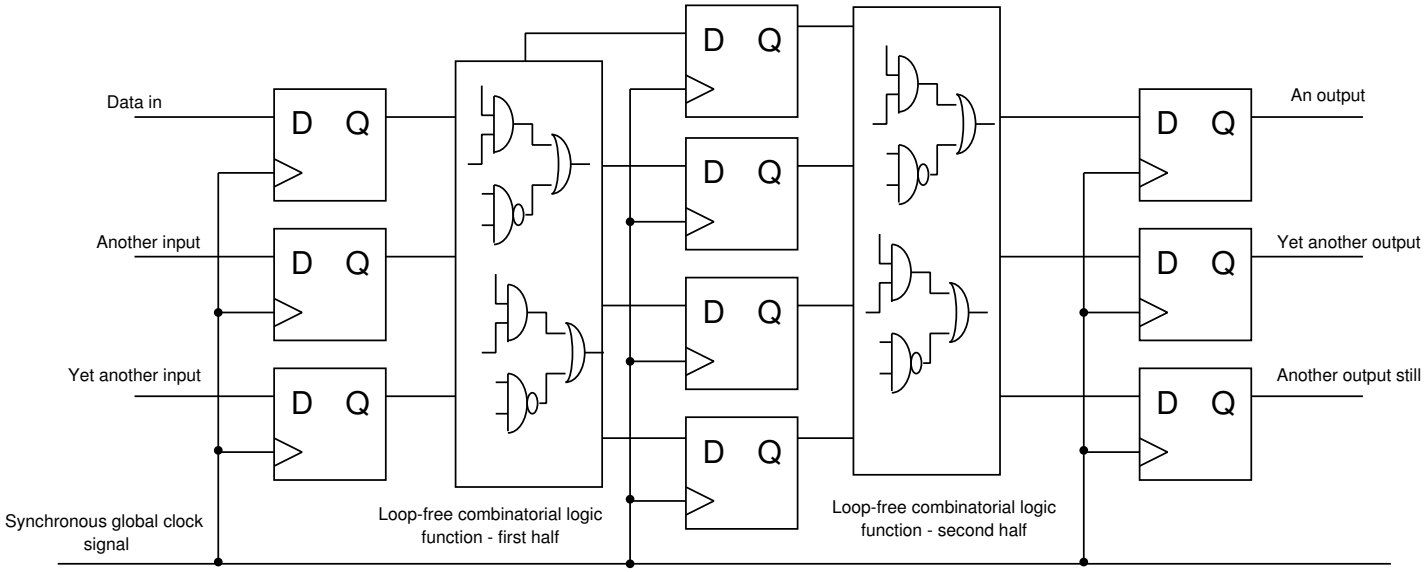




Pipelining

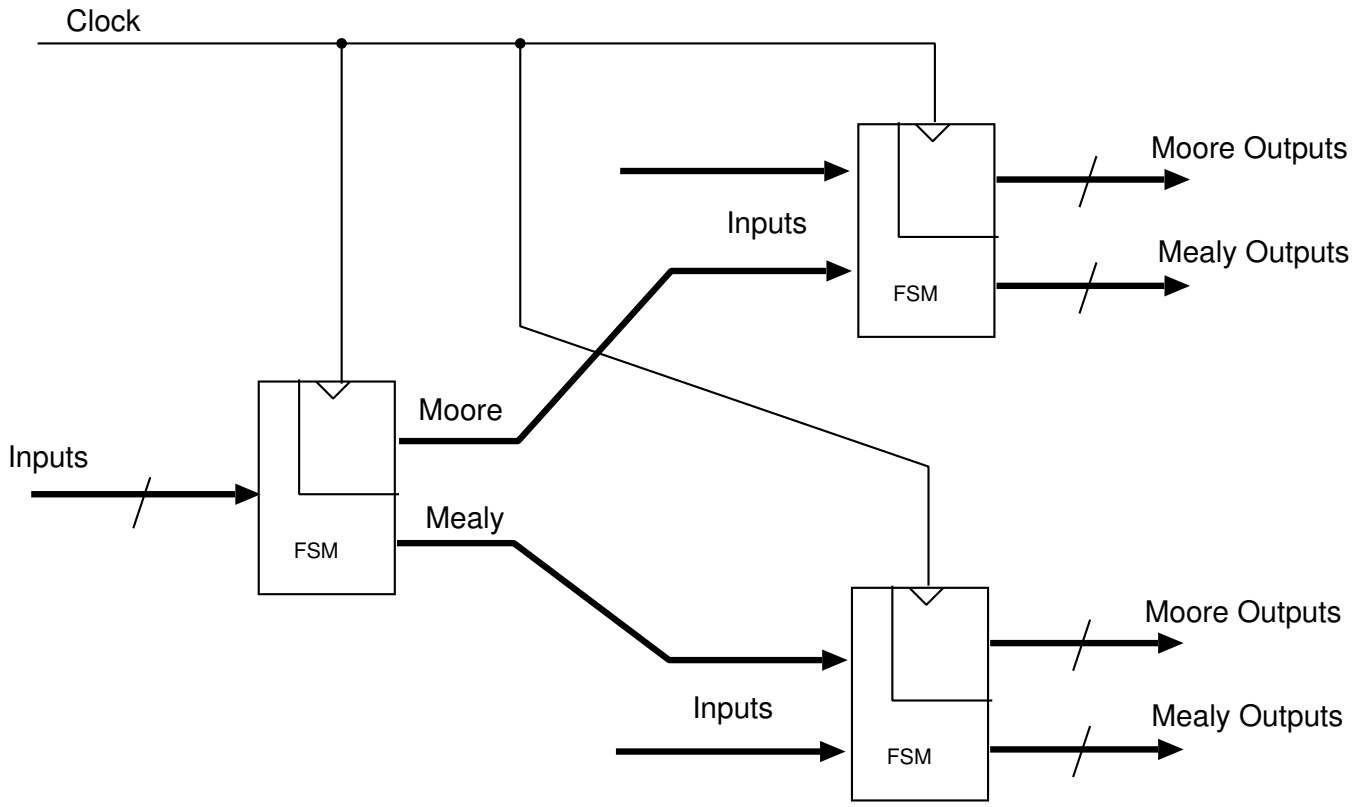


Desired logic function

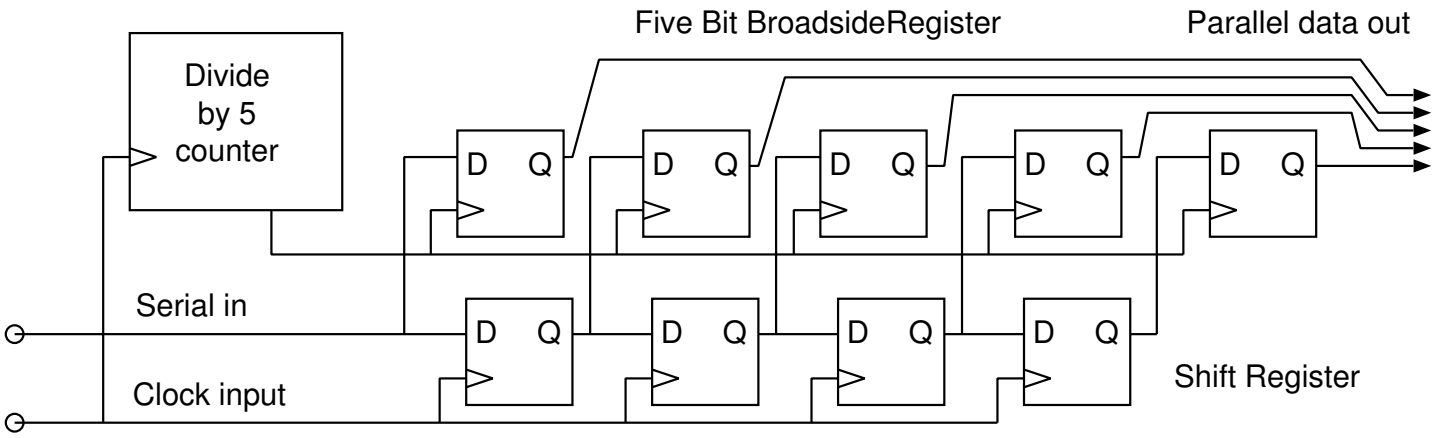


Desired logic function - pipelined version.

Cascading FSMs



How Not To Do It

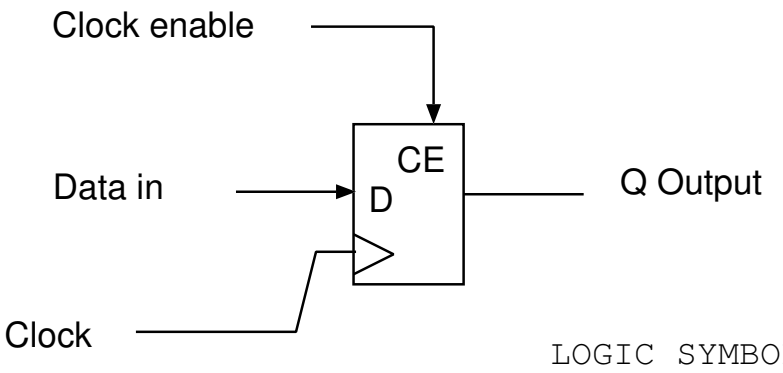


An example that uses (badly) a derived clock: a serial-to-parallel converter

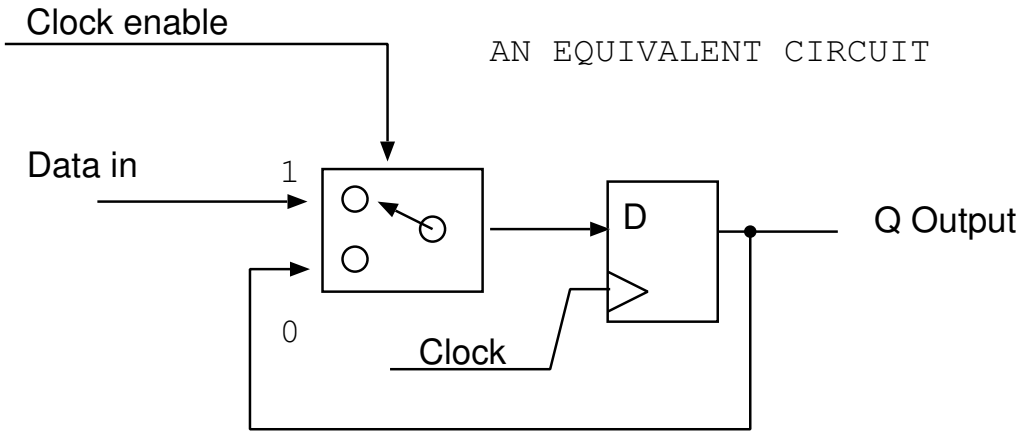
```
reg [2:0] r2;
always @(posedge clock) r2 <= (r2==4)?0:r2+1;
wire bclock = r2[2];

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 D-type with clock-enable



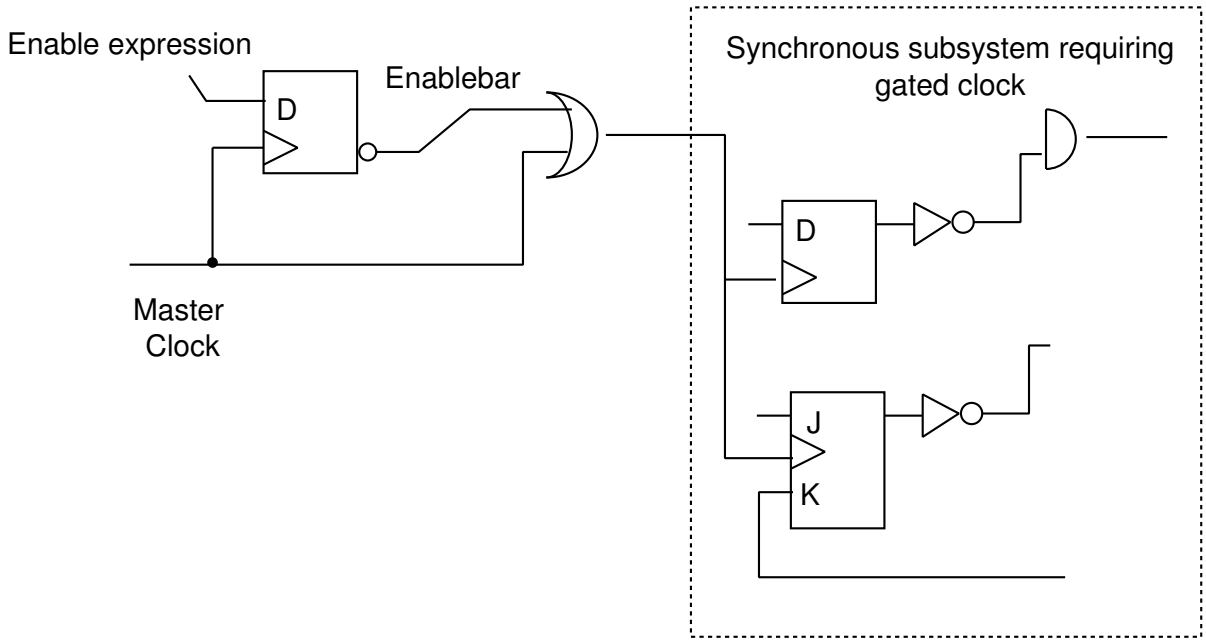
LOGIC SYMBOL



AN EQUIVALENT CIRCUIT

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

alternatively

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

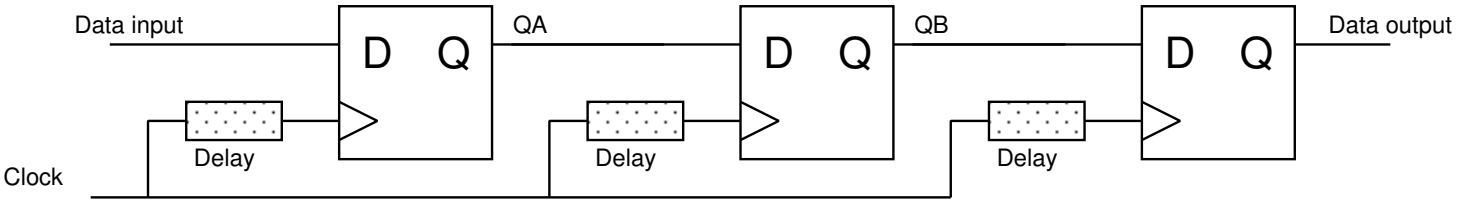
A Gated Clock



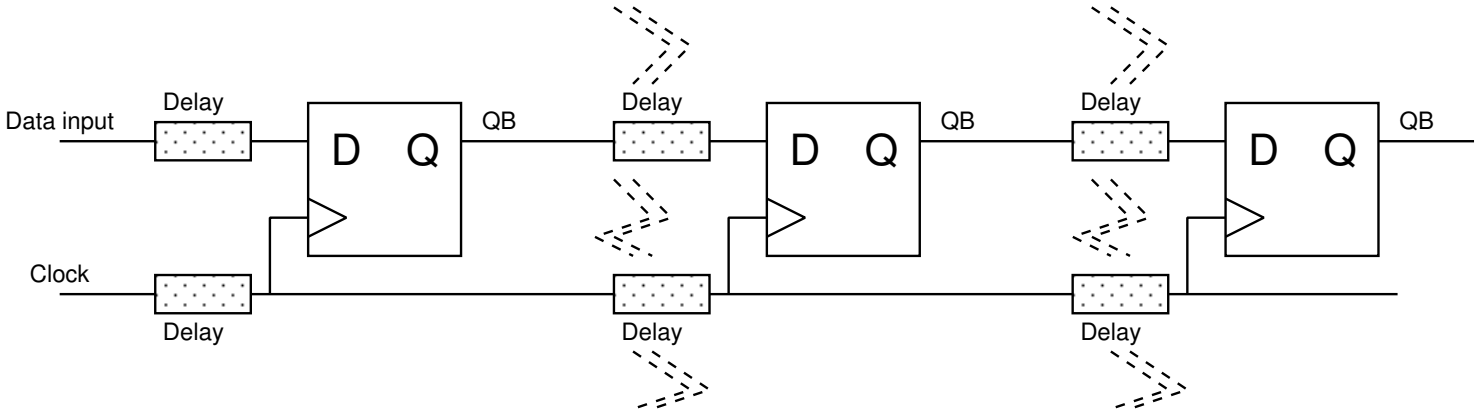
OR'ing with a negated enable works cleanly.

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

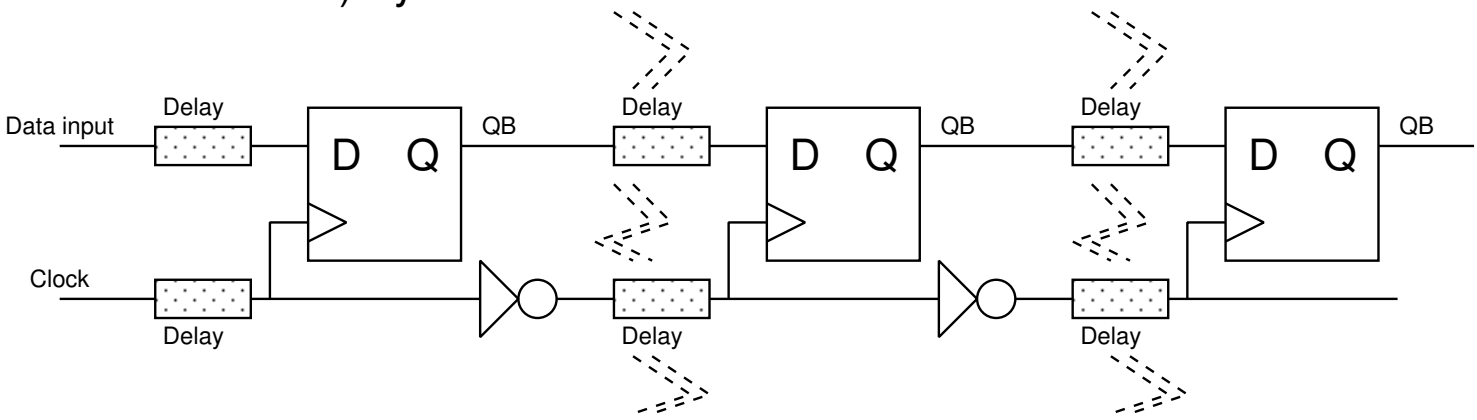
Clock Skew



a) A three-stage shift register with some clock skew delays.

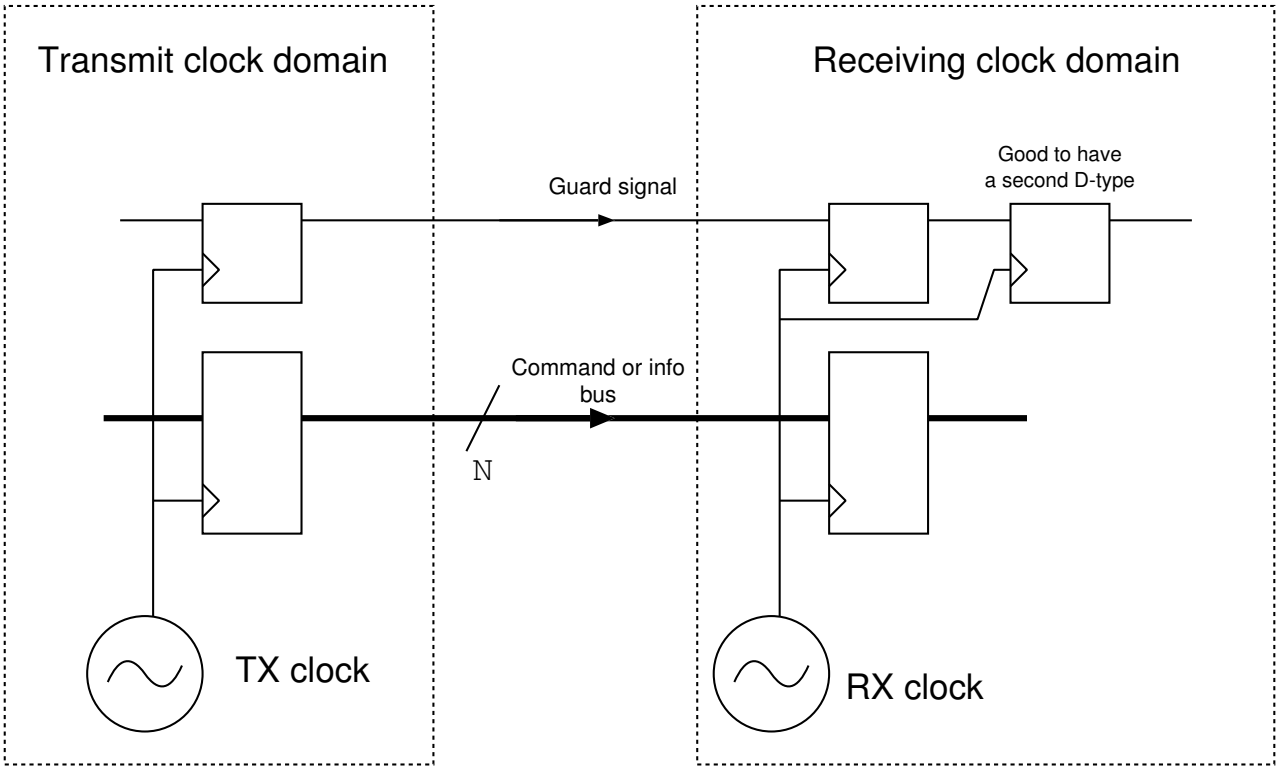


b) System interconnection with clock skews



c) A solution for serious skew and delay problems ?

Crossing an Asynchronous Domain Boundary



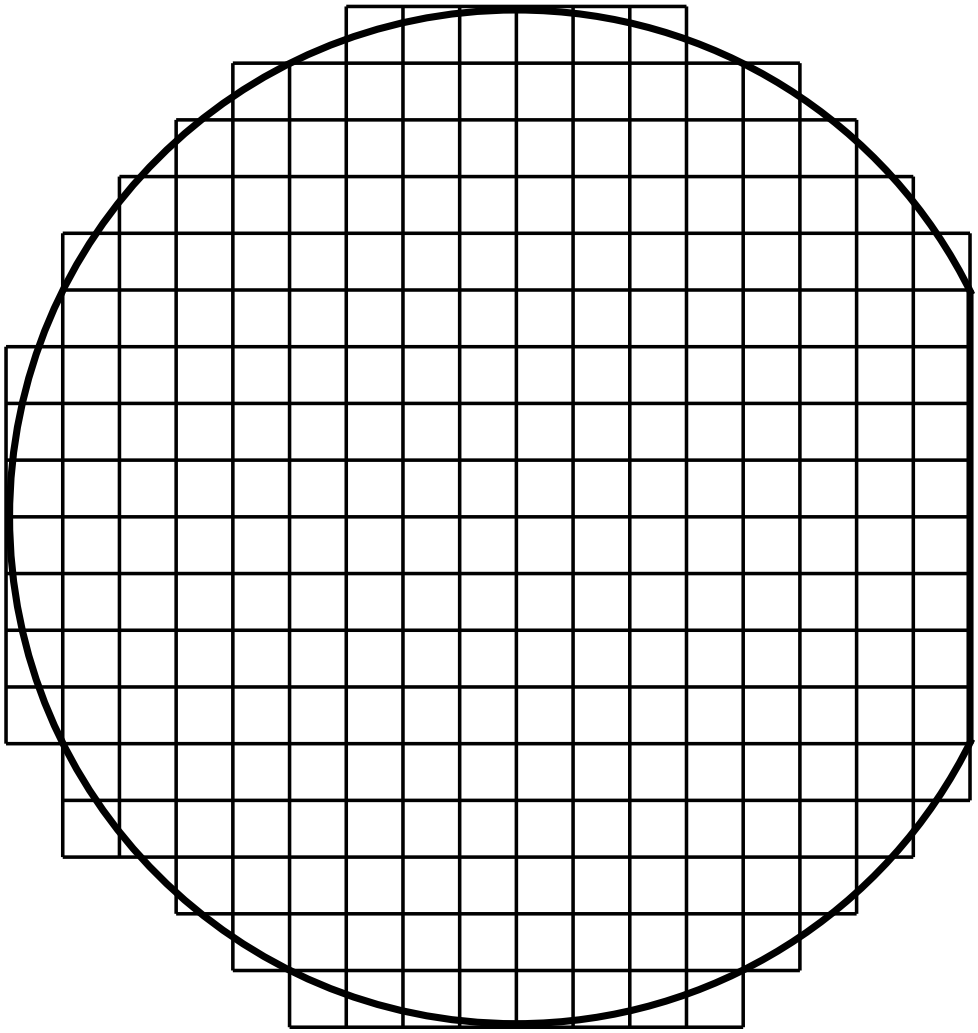
1. The wider the bus width, N, the fewer the number of transactions per second needed and the greater the timing flexibility in reading the data from the receiving latch.

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.

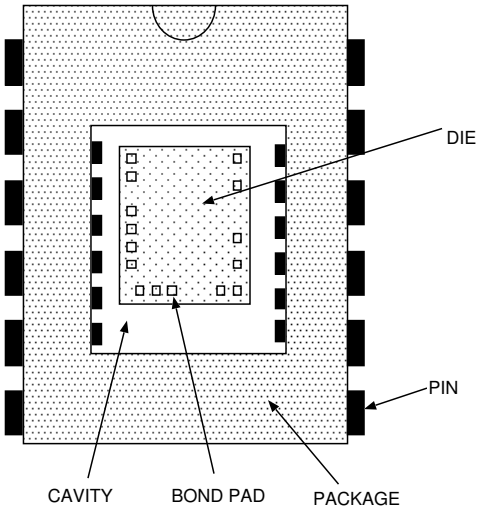
All real systems have many clock domains and frquently implement this style of solution.

Dicing a wafer

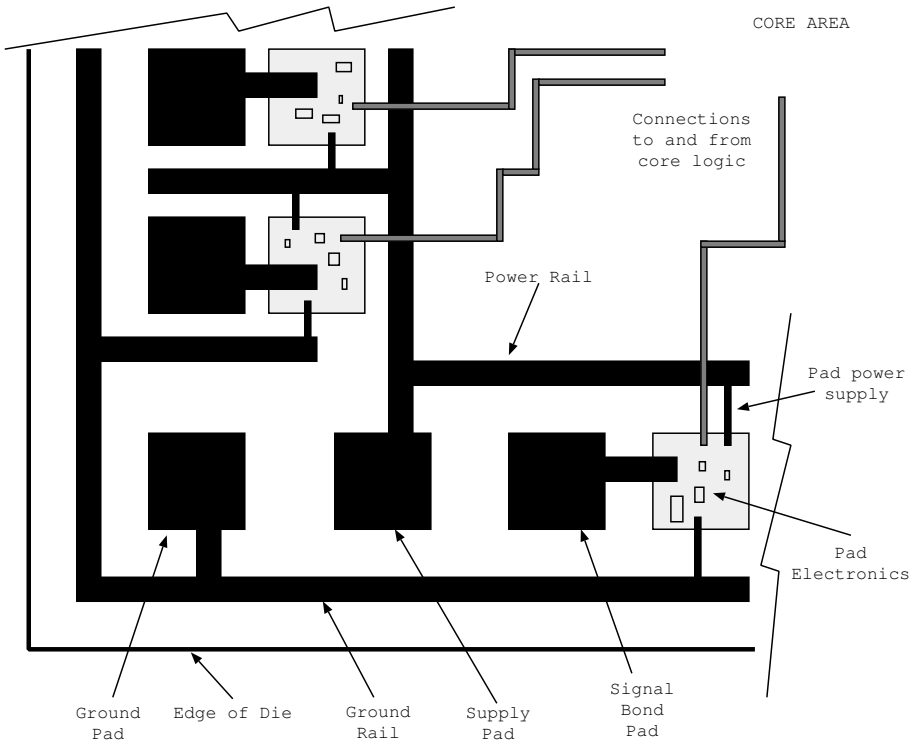


(Chips are not always square)

A chip in its package, ready for bond wires



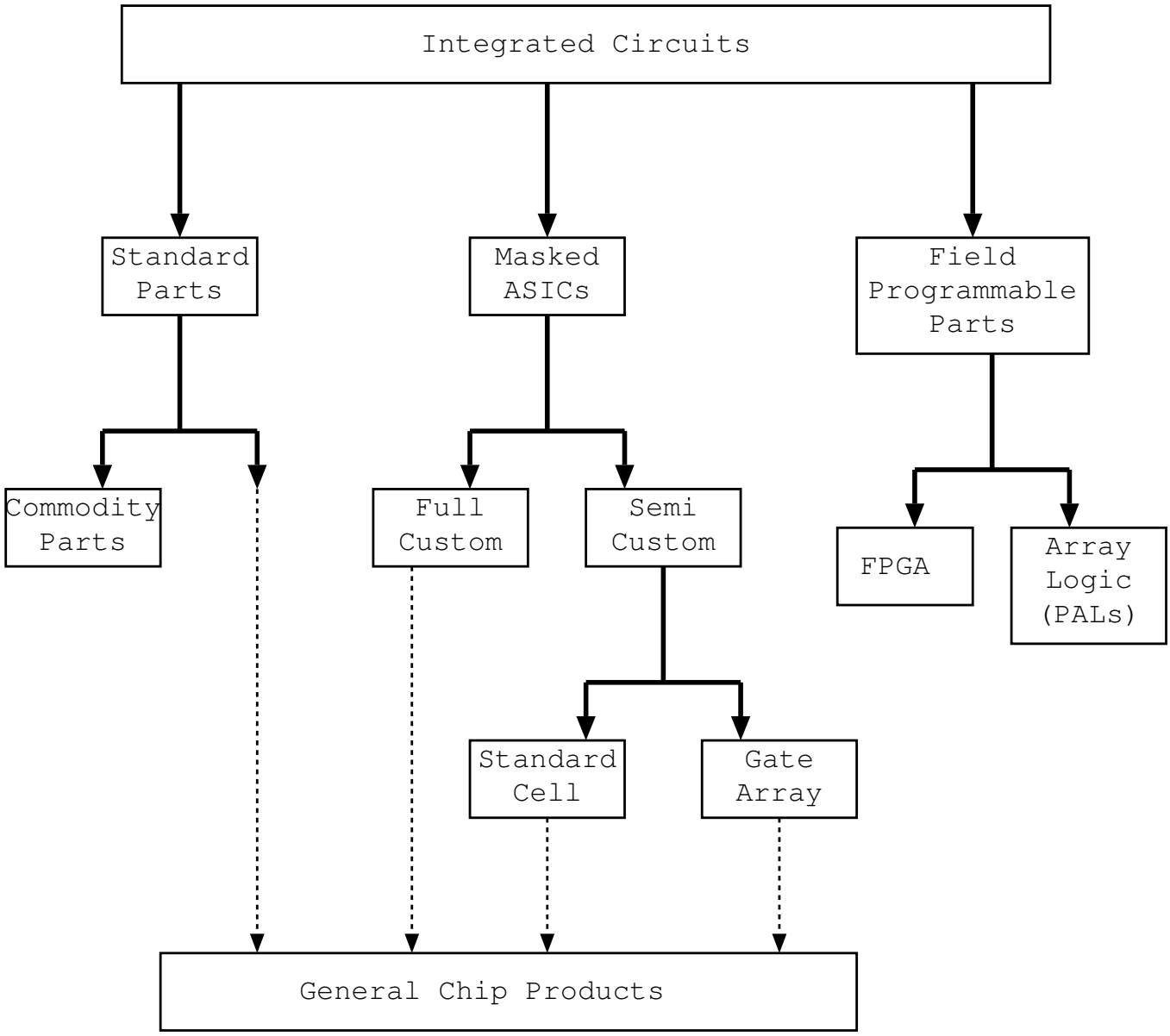
IO and power pads



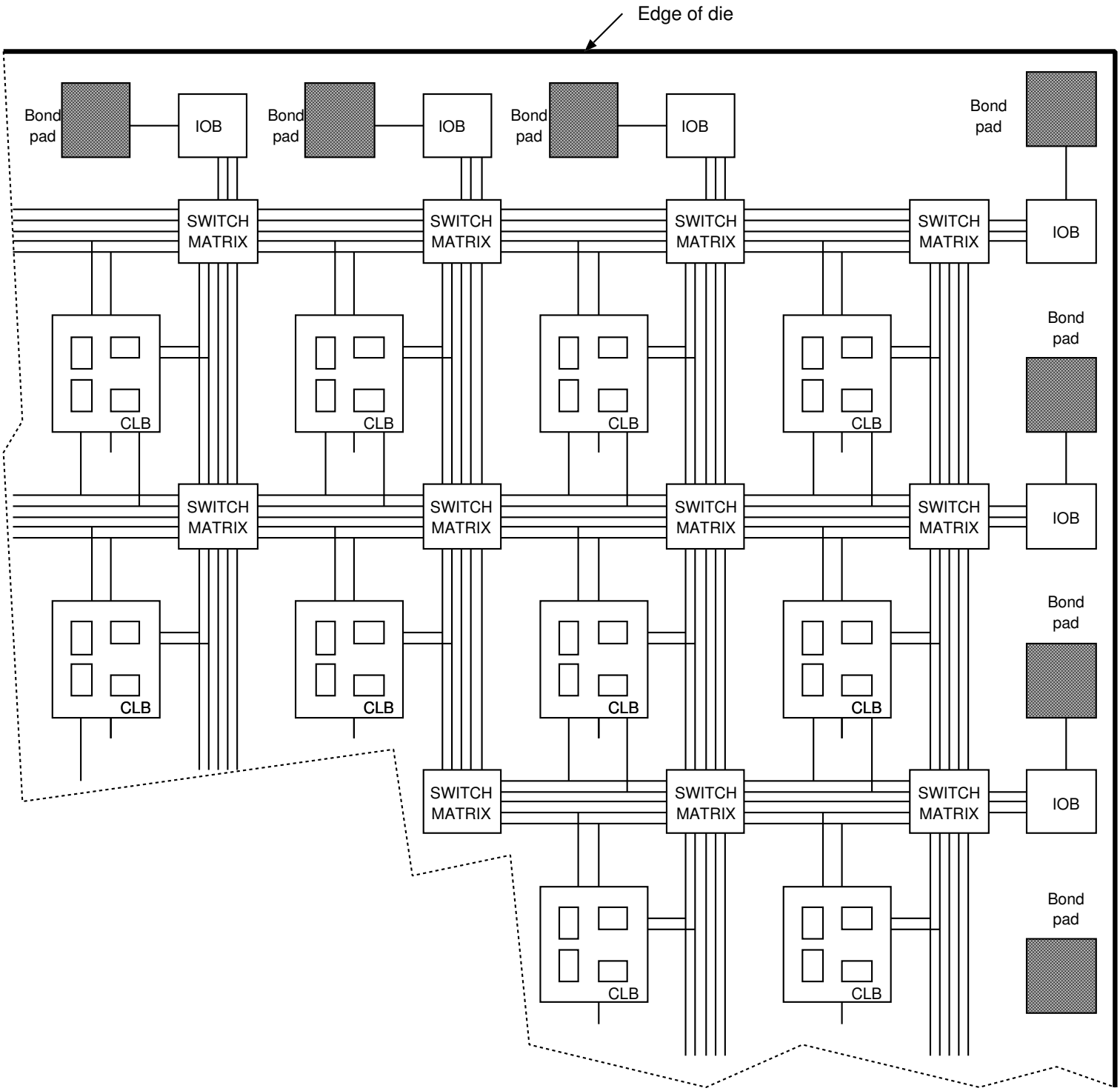
Die cost example

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

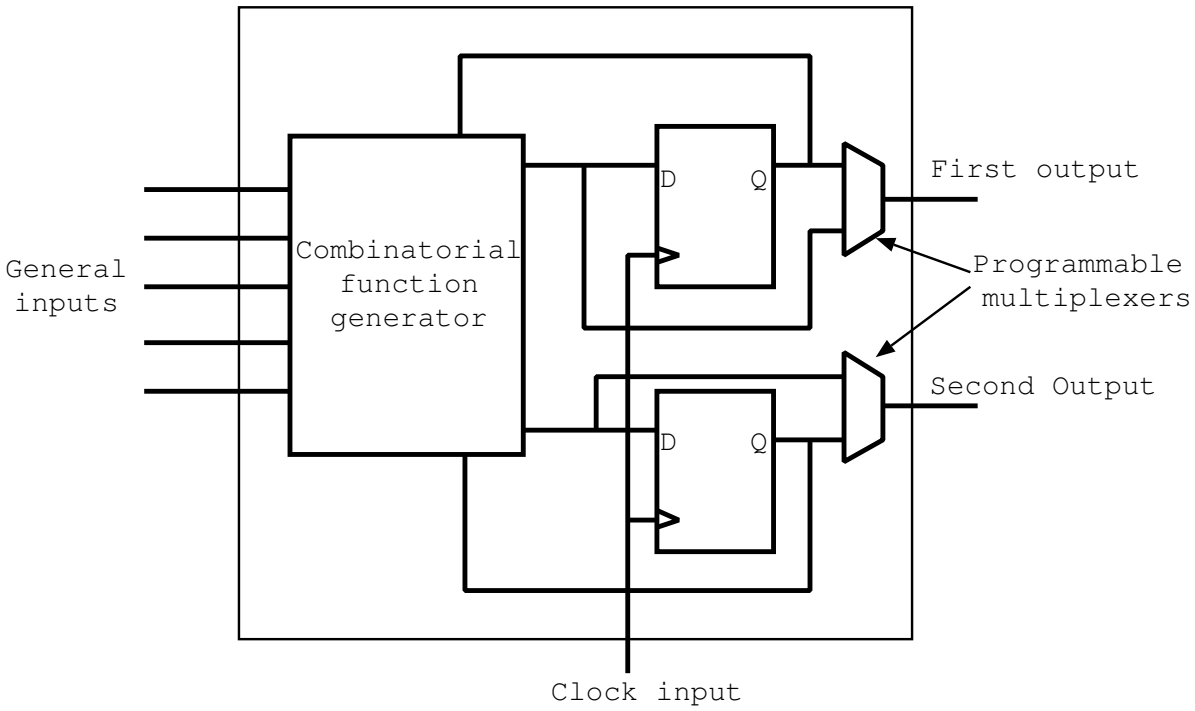
A taxonomy of ICs



Field Programmable Gate Arrays



A configurable logic block for a look-up-table based FPGA

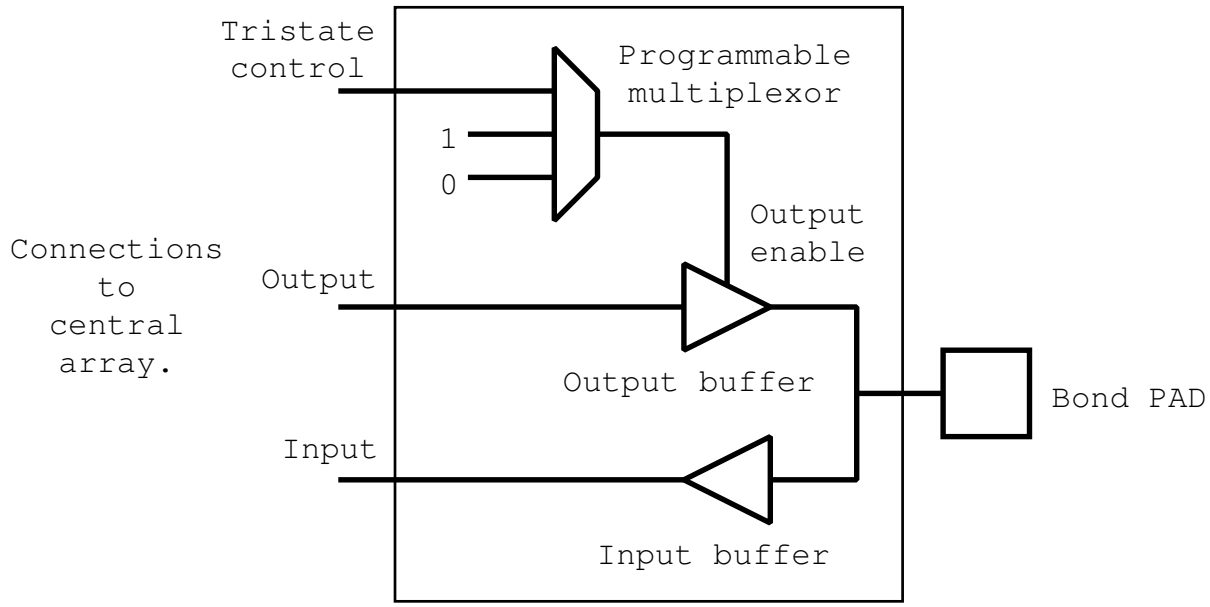


This CLB contains one LUT and two D-type's. The output can be sequential or combinational.

Seven LUT inputs: $2^7 = 128$

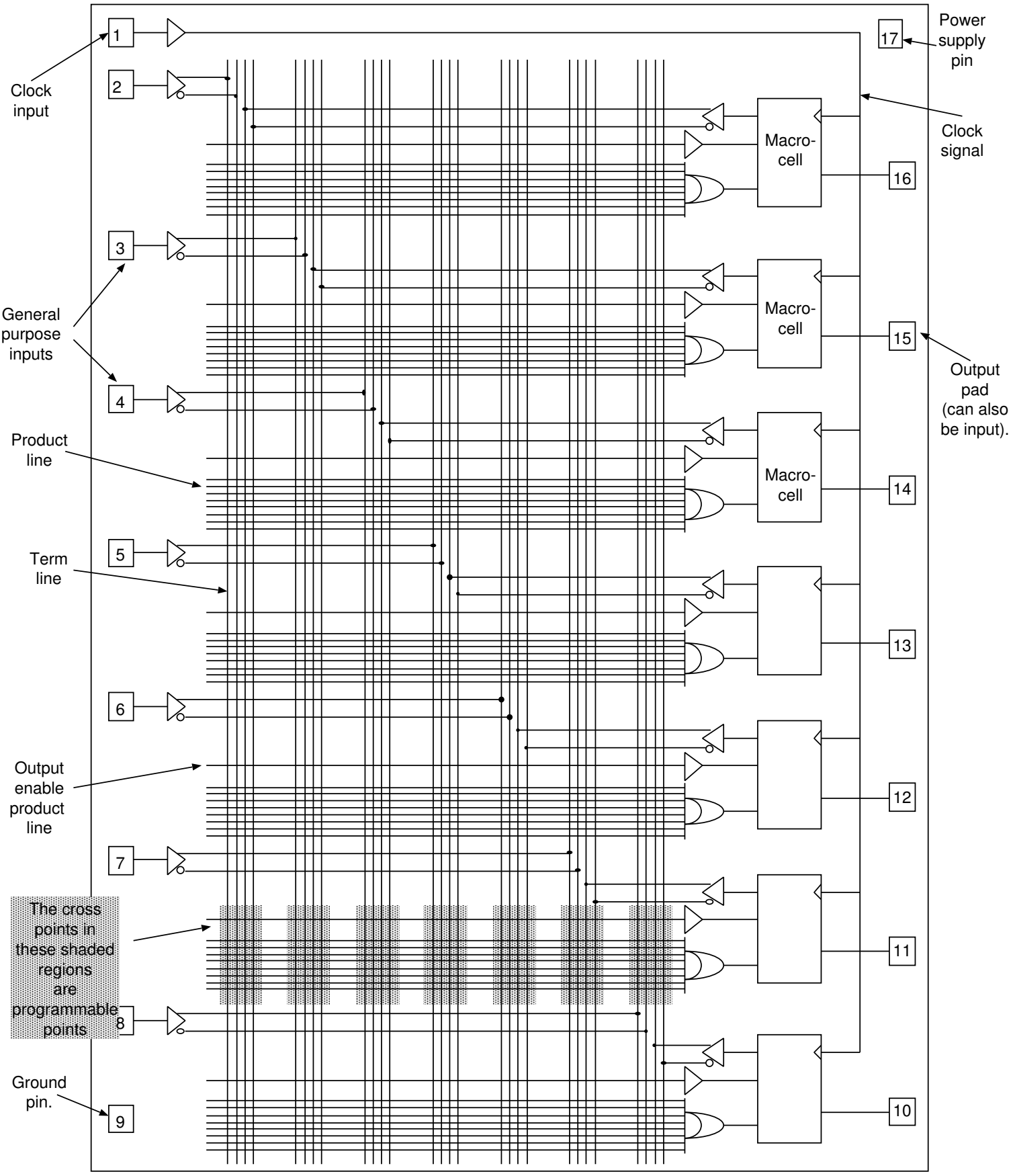
The LUT can be a RAM of 128 locations of two bits.

FPGA: Example I/O Block

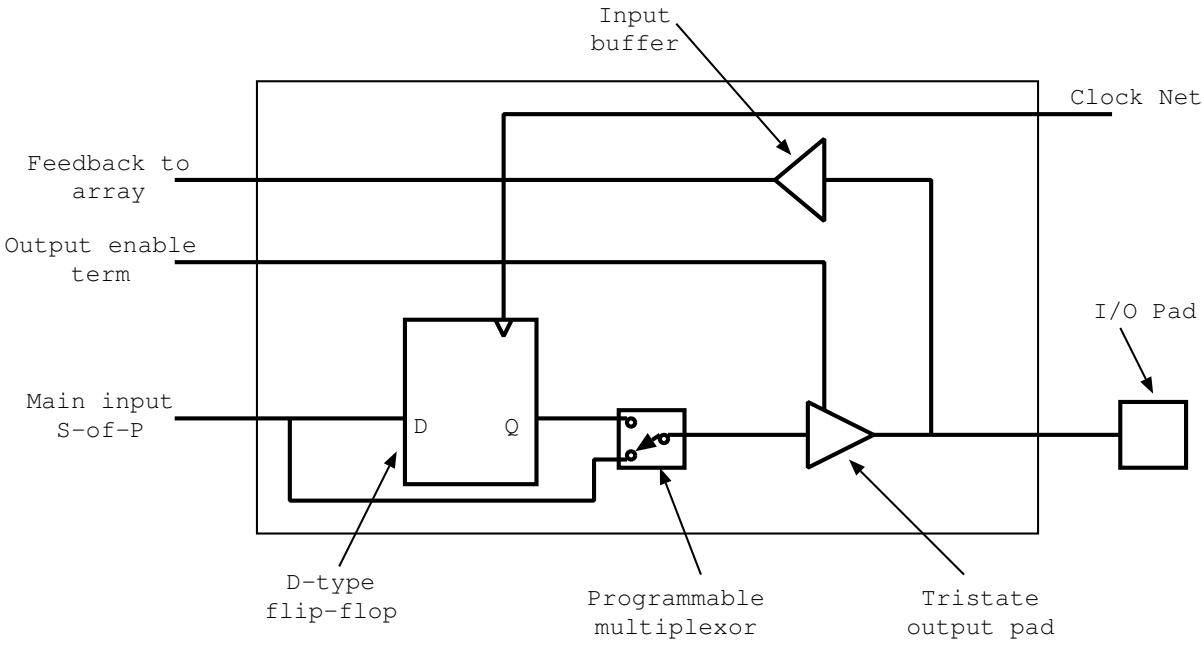


Pictured is a basic I/O block.

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
Output pad (can also be input).
Macro-cell
Macro-cell
Macro-cell
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

Contents of the PAL macrocell

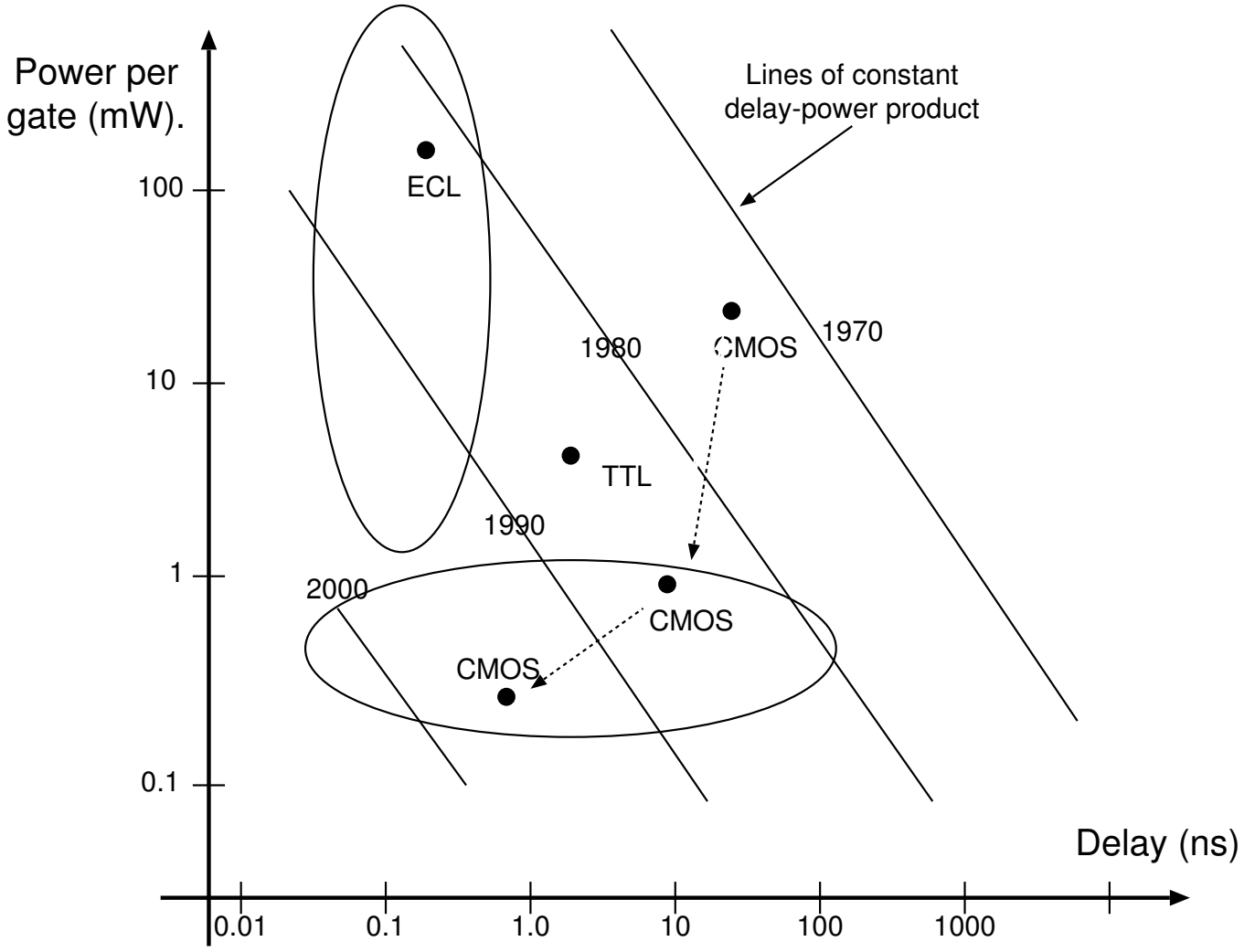


Example programming of a PAL showing only fuses for the top macrocell

```
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)
```

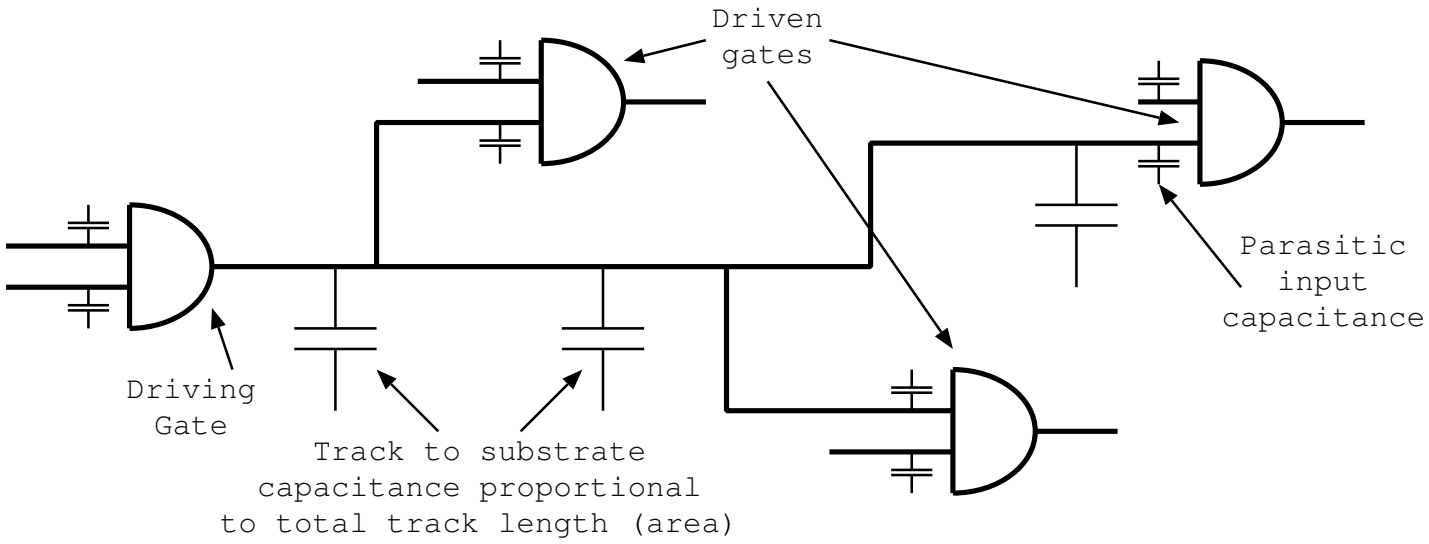
Delay-power style of technology comparison chart



	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



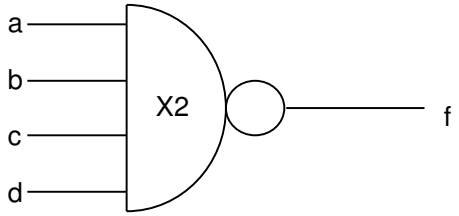
An example cell from a manufacturer's cell library

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

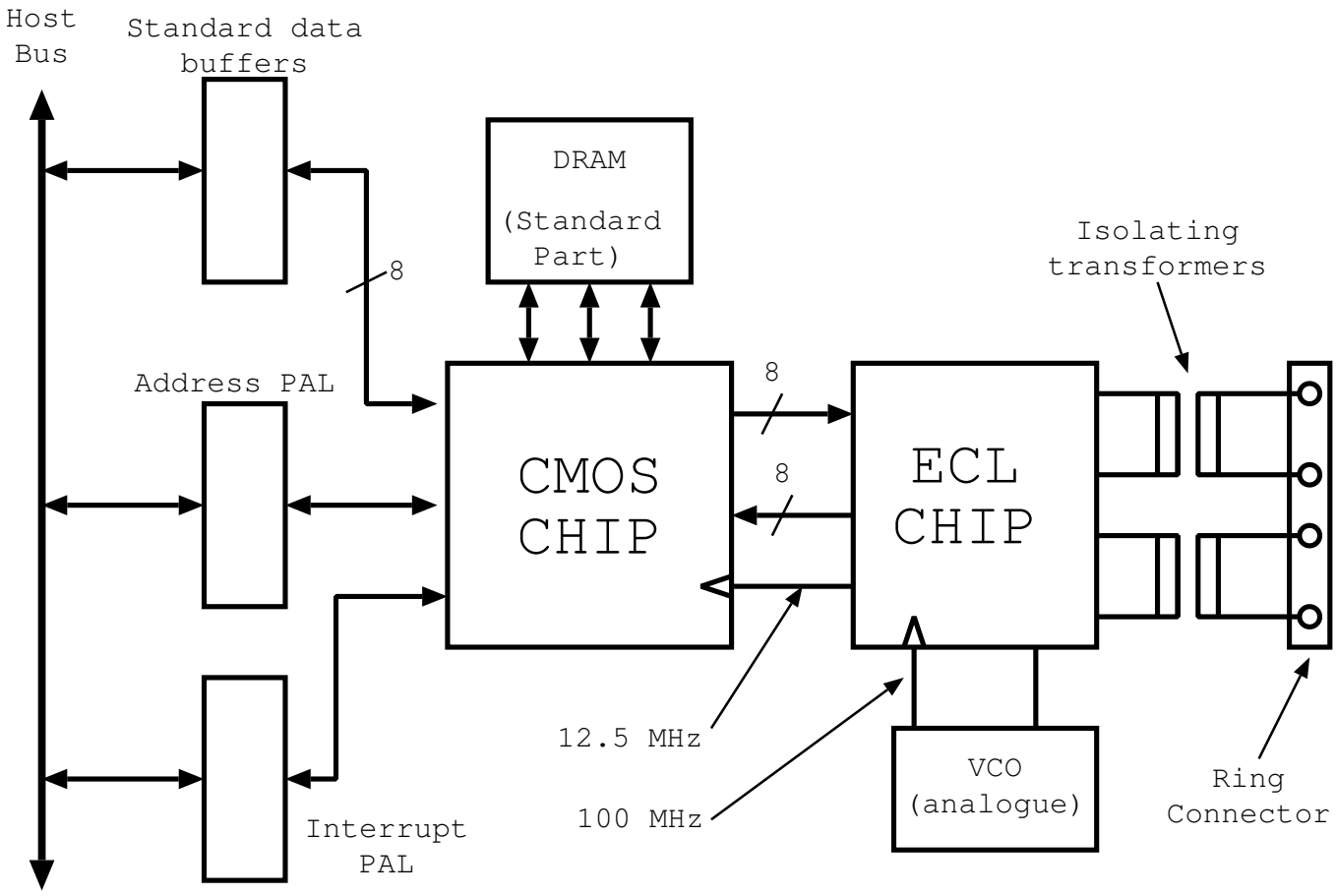
- 0.18 micron line size
- 28 million transistors
- 500-700 MHz clock speed
- 11x12 mm (140 mm^2) die size

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

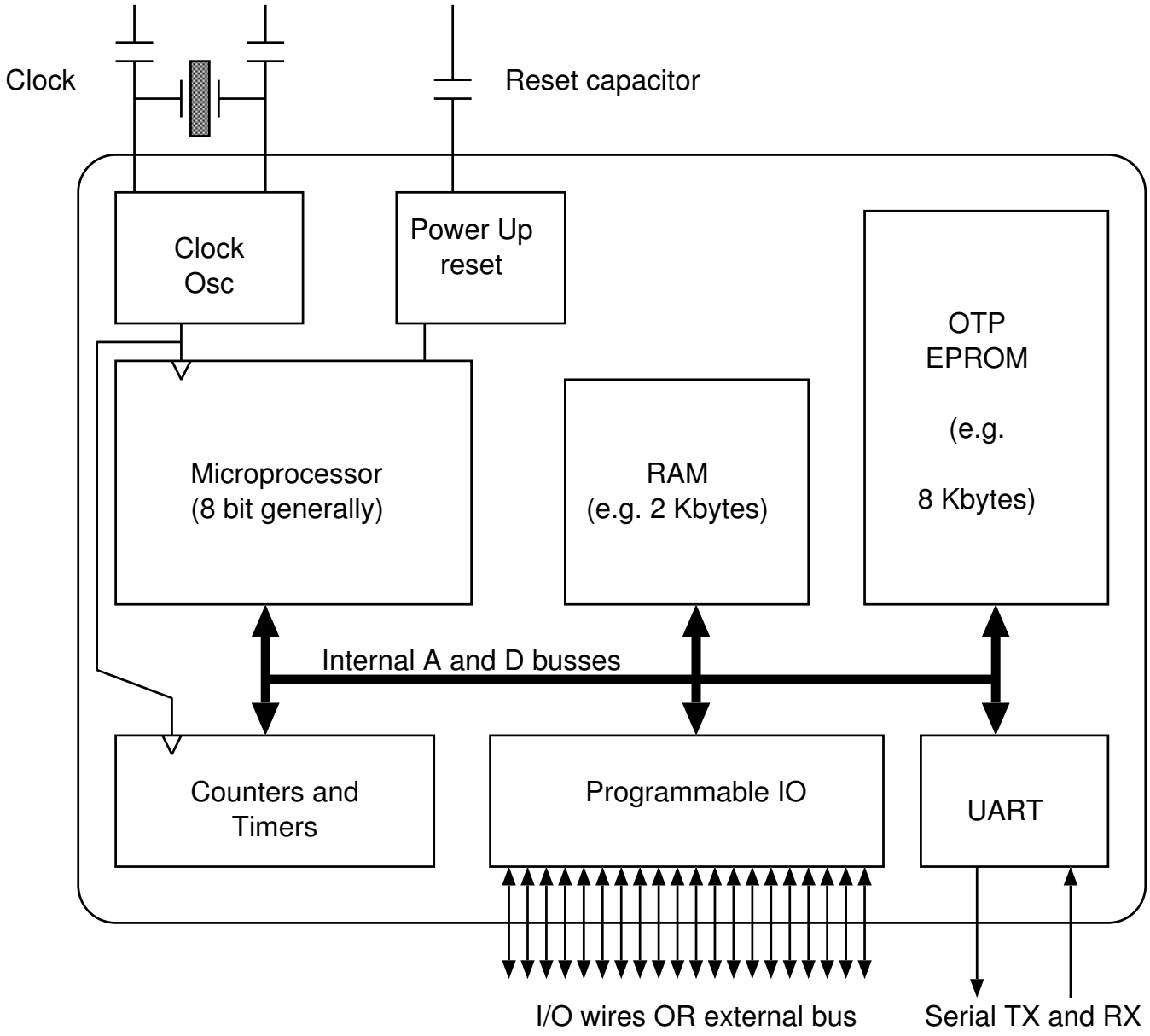


Designed in 1980.

ECL Chip 100 MHz, bit serial.

CMOS Chip 12.5 MHz, byte-wide data.

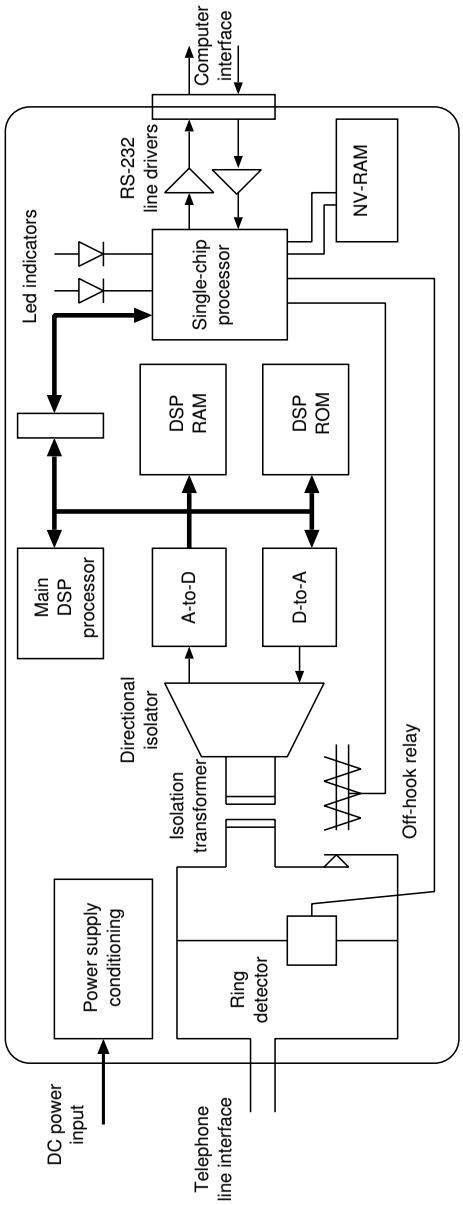
A Basic Micro-Controller



Introduced 1989-85.

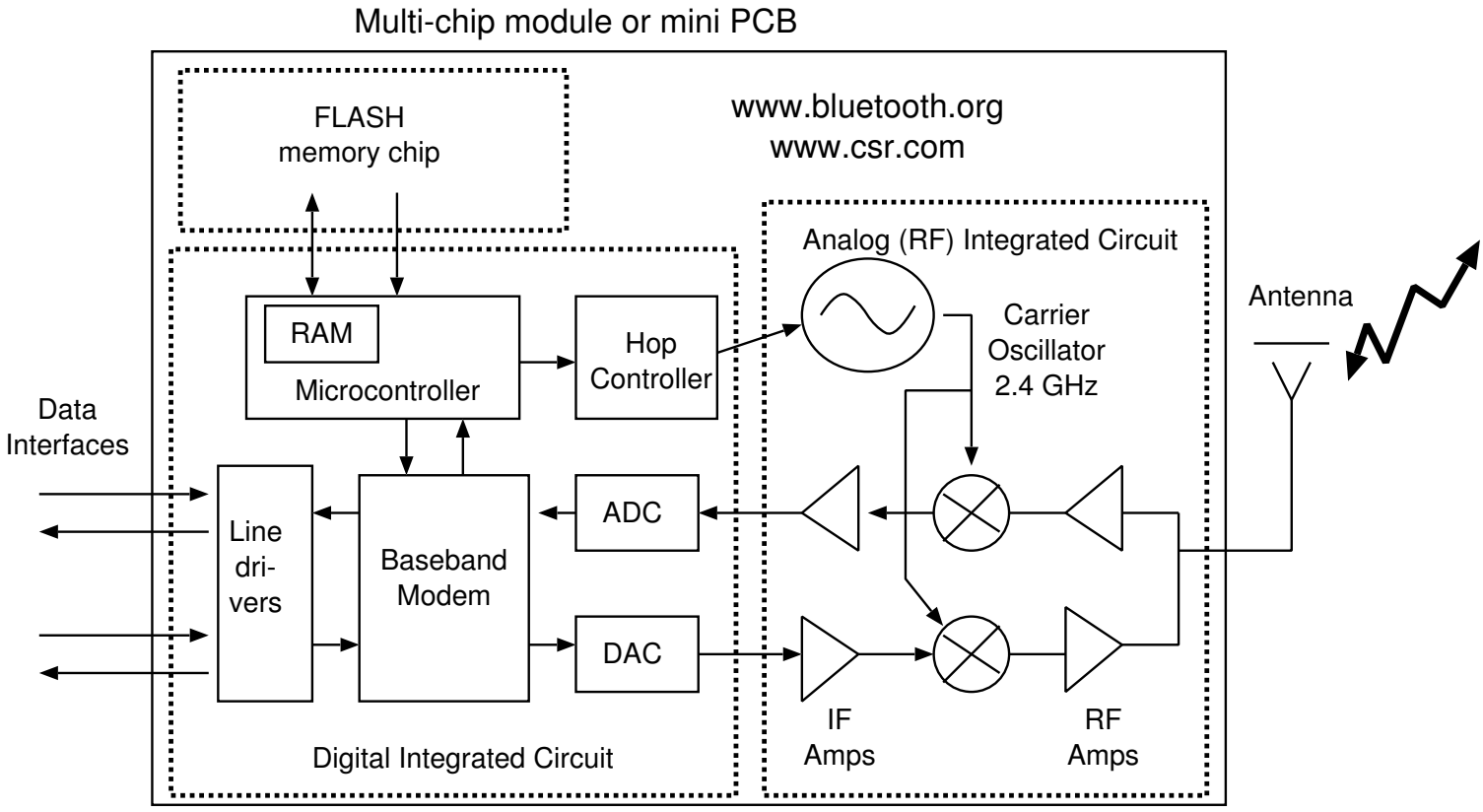
Such a micro-controller has an D8/A16 architecture and would be used in a mouse or smartcard.

Design partitioning: A Modem.



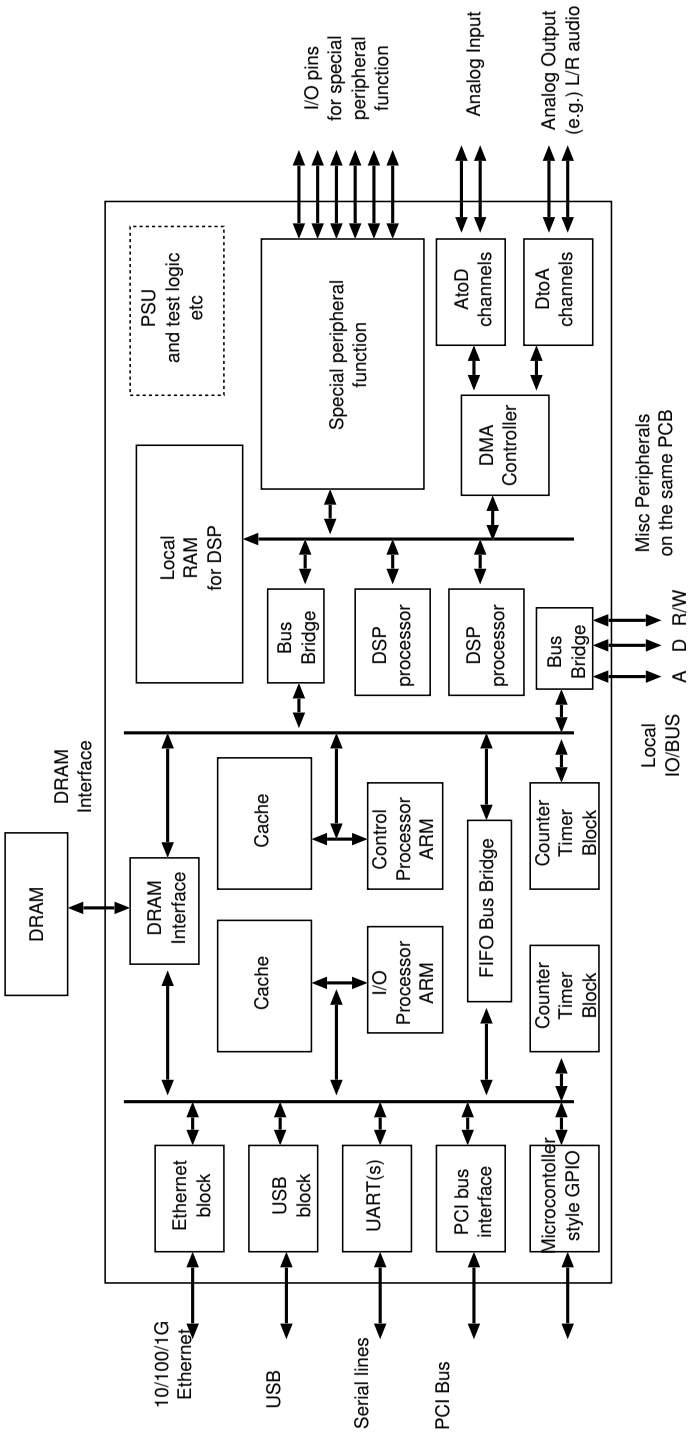
In 1980 we used a microcontroller with external DSP components.

Design partitioning: A Miniature Radio Module



Introduced 1998.

1998: A Platform Chip: D32/A32 twice!



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.

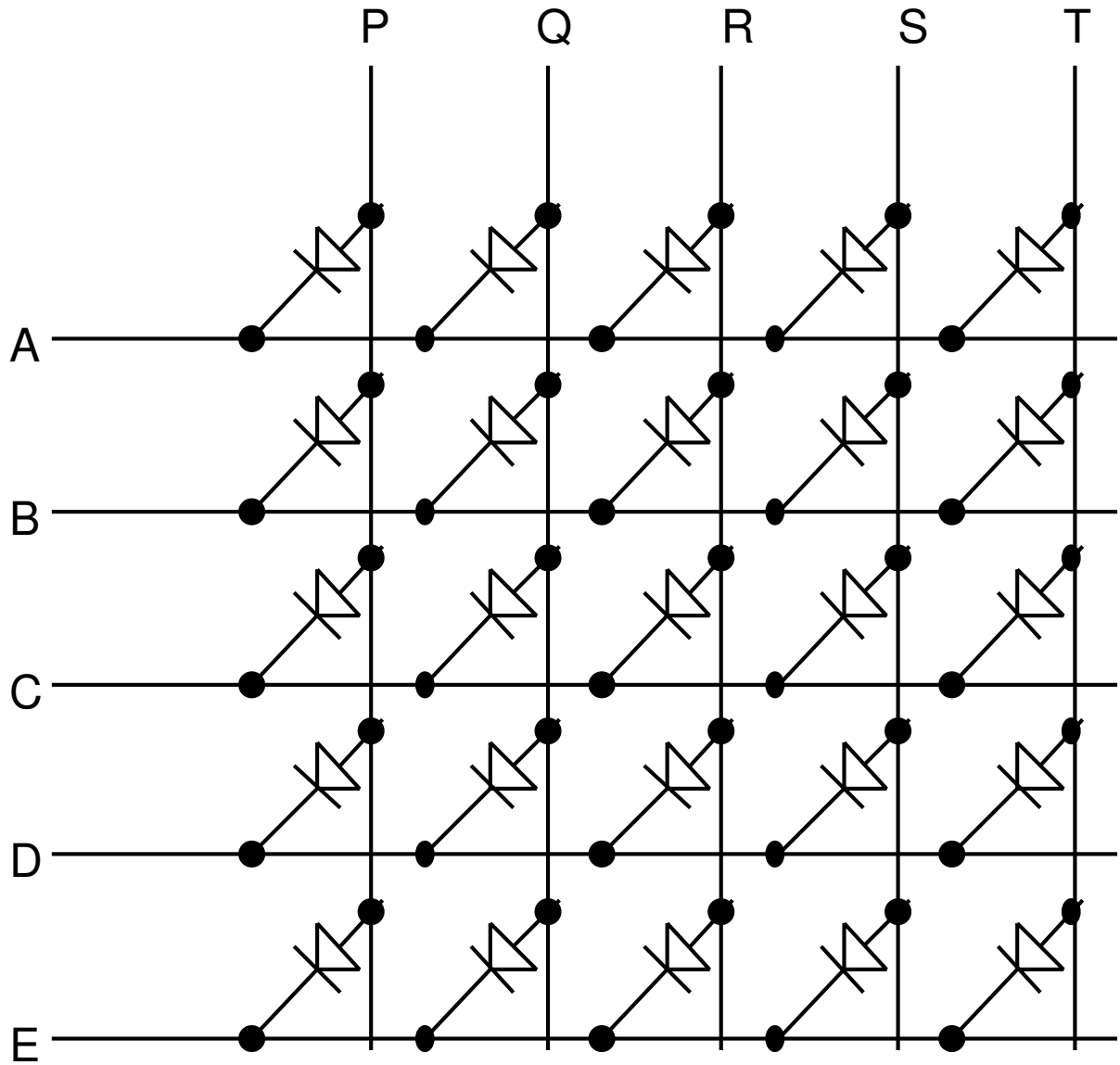
The right-hand-side ARM is used as the system controller. It can access all of the chip's resources over various bus bridges. It can access off-chip devices, such as an LCD display or keyboard via a general purpose A/D local bus.

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.

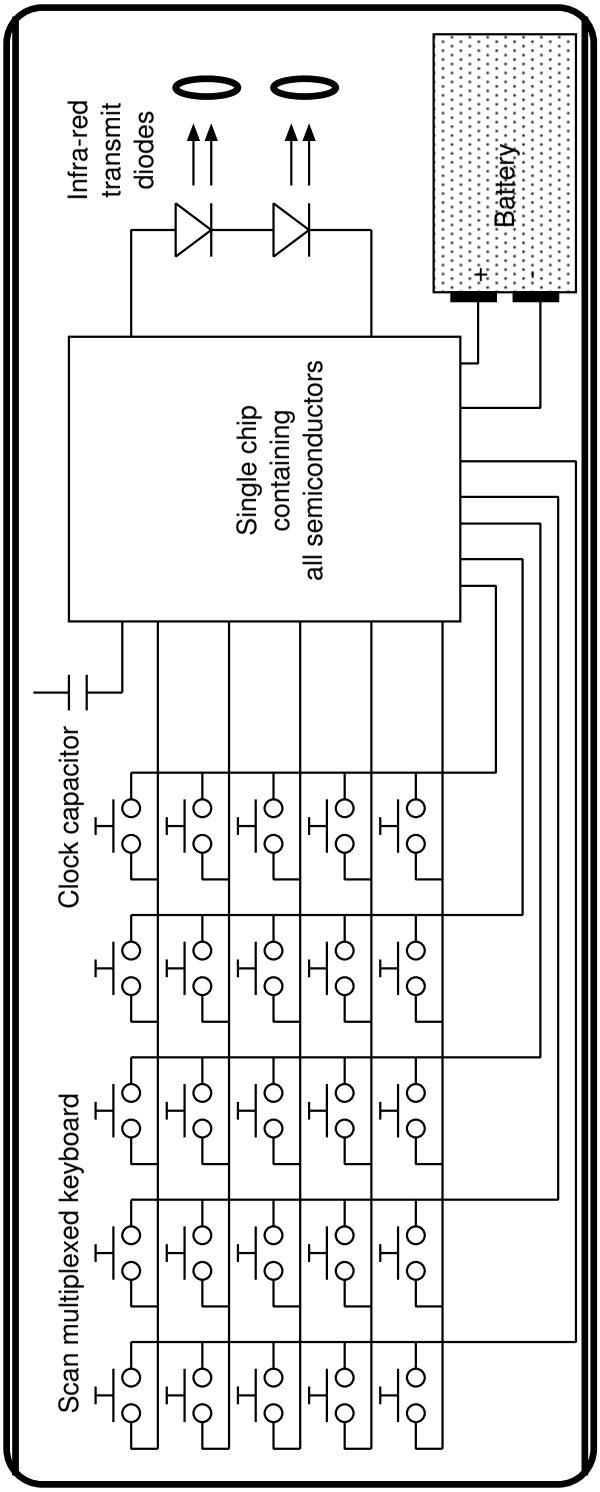
The twin DSP devices run completely out of on-chip SRAM. Such SRAM may dominate the die area of the chip. If both are fetching instructions from the same port of the same RAM, then they had better be executing the same program in lock-step or else have some own local cache to avoid huge loss of performance 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.

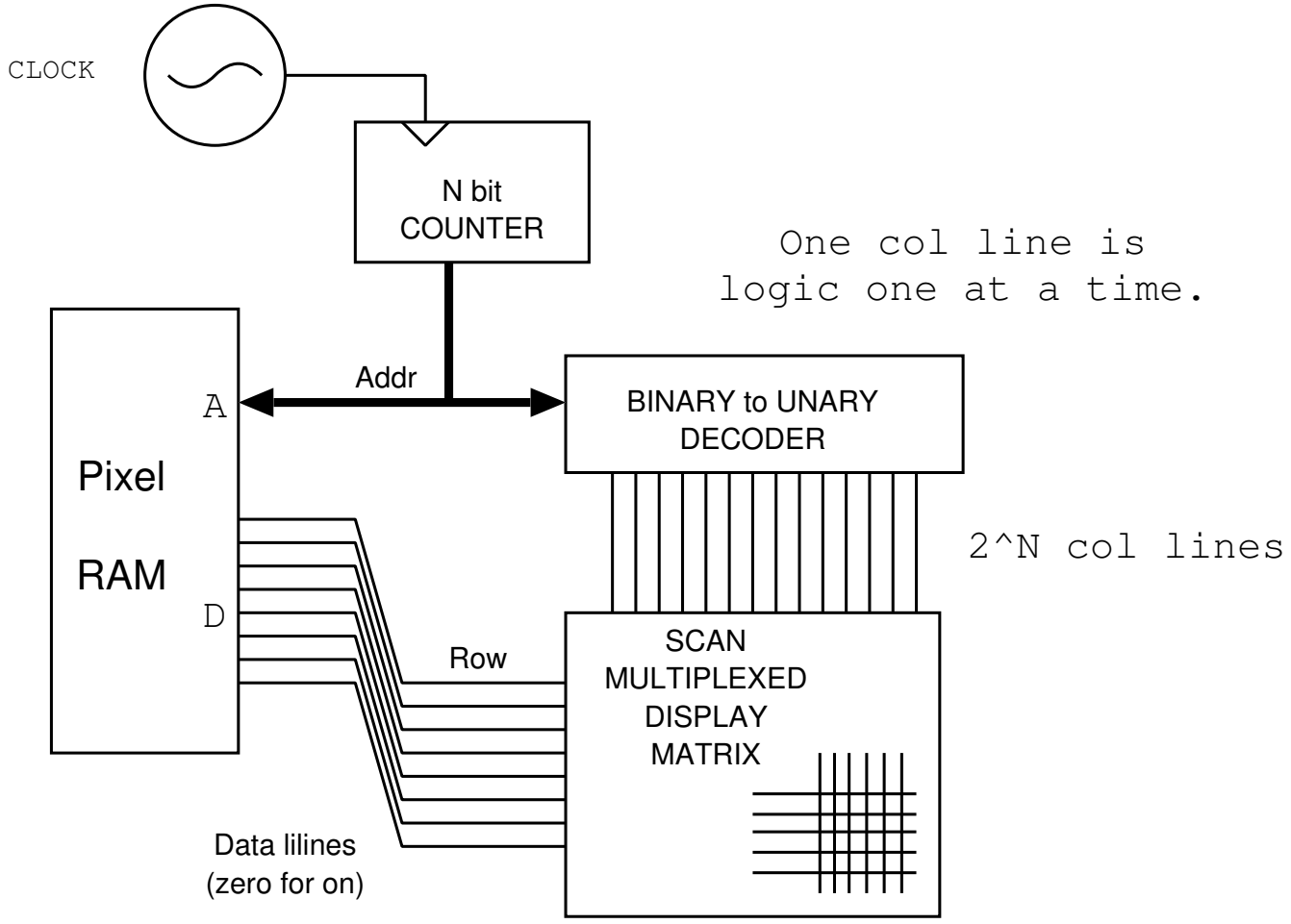
LEDs wired in a matrix to reduce external pin count



IR Handset Internal Circuit

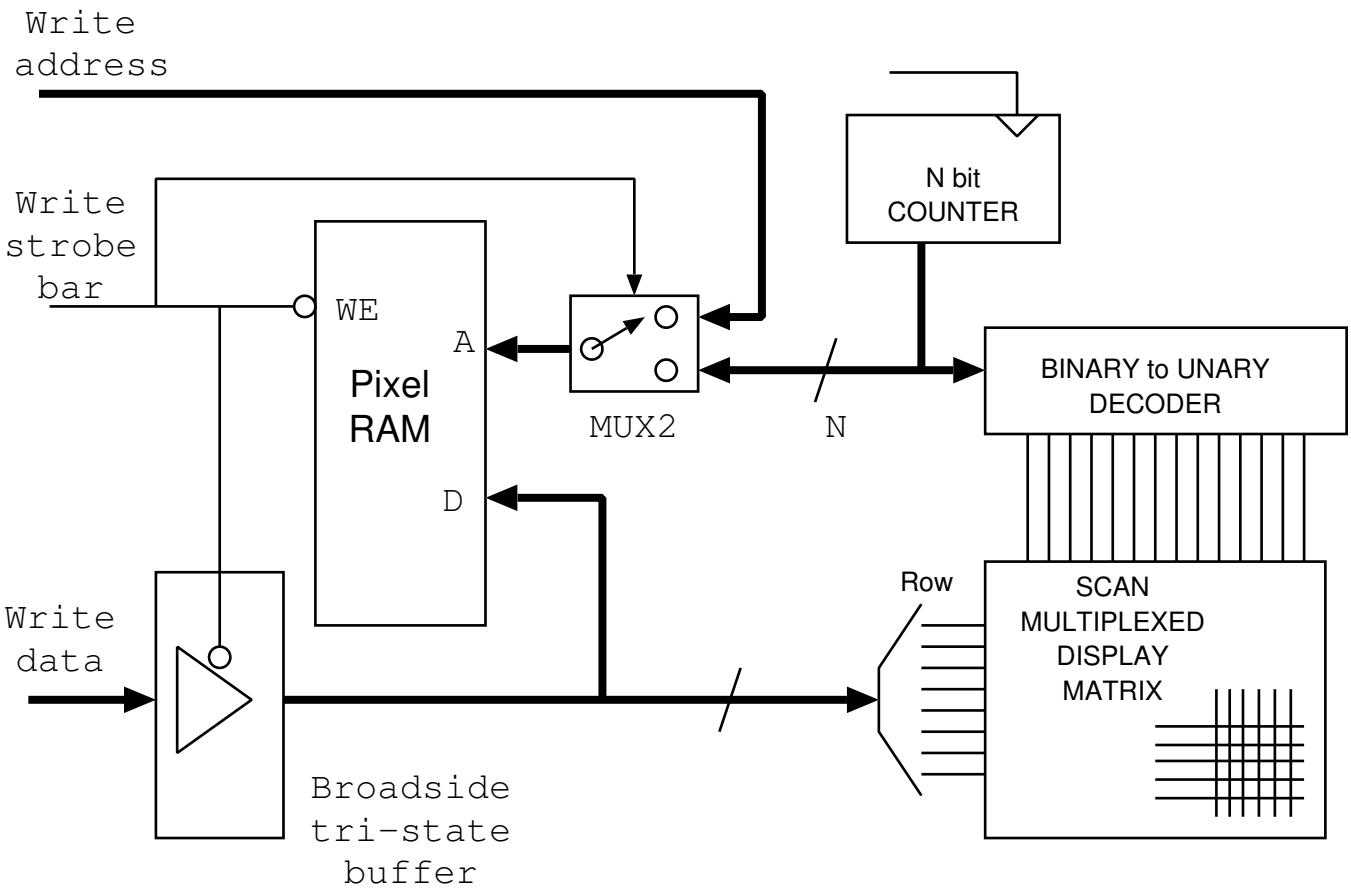


Scan multiplex logic for an LED pixel-mapped display



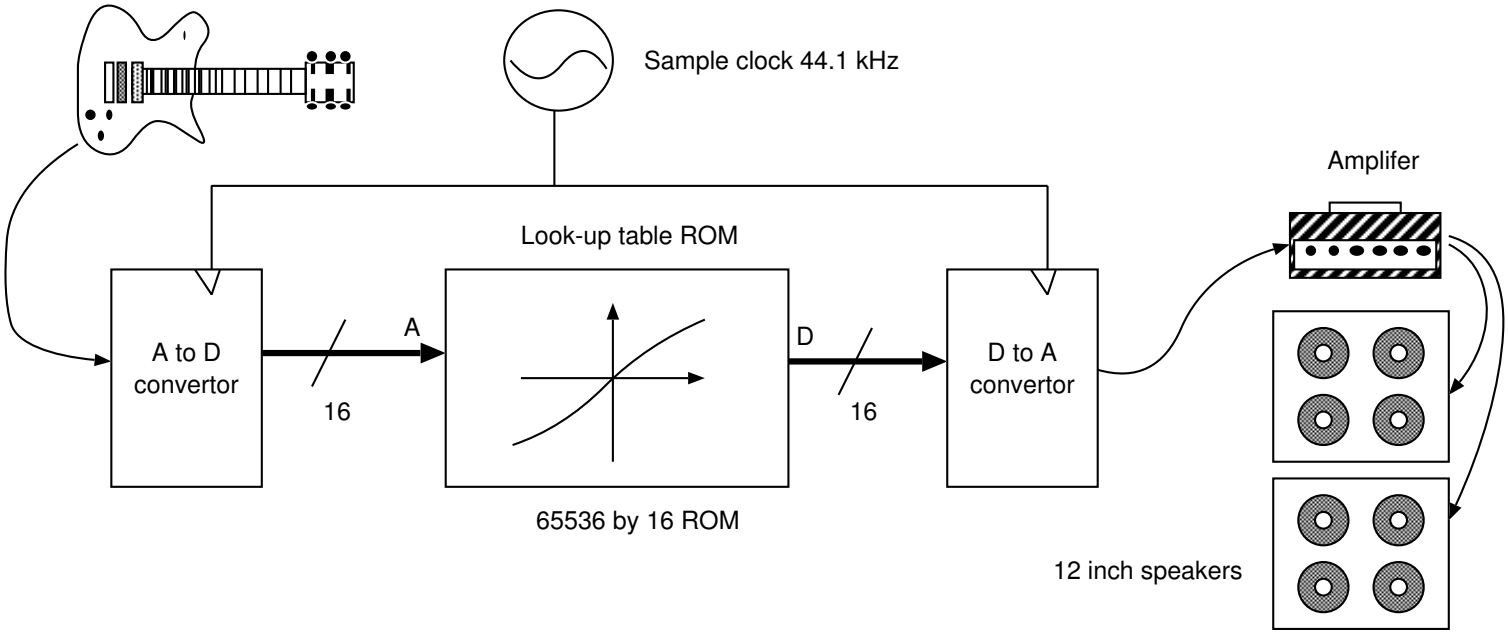
You made one of these in the Ia H/W classes.

Addition of psudo dual-porting logic



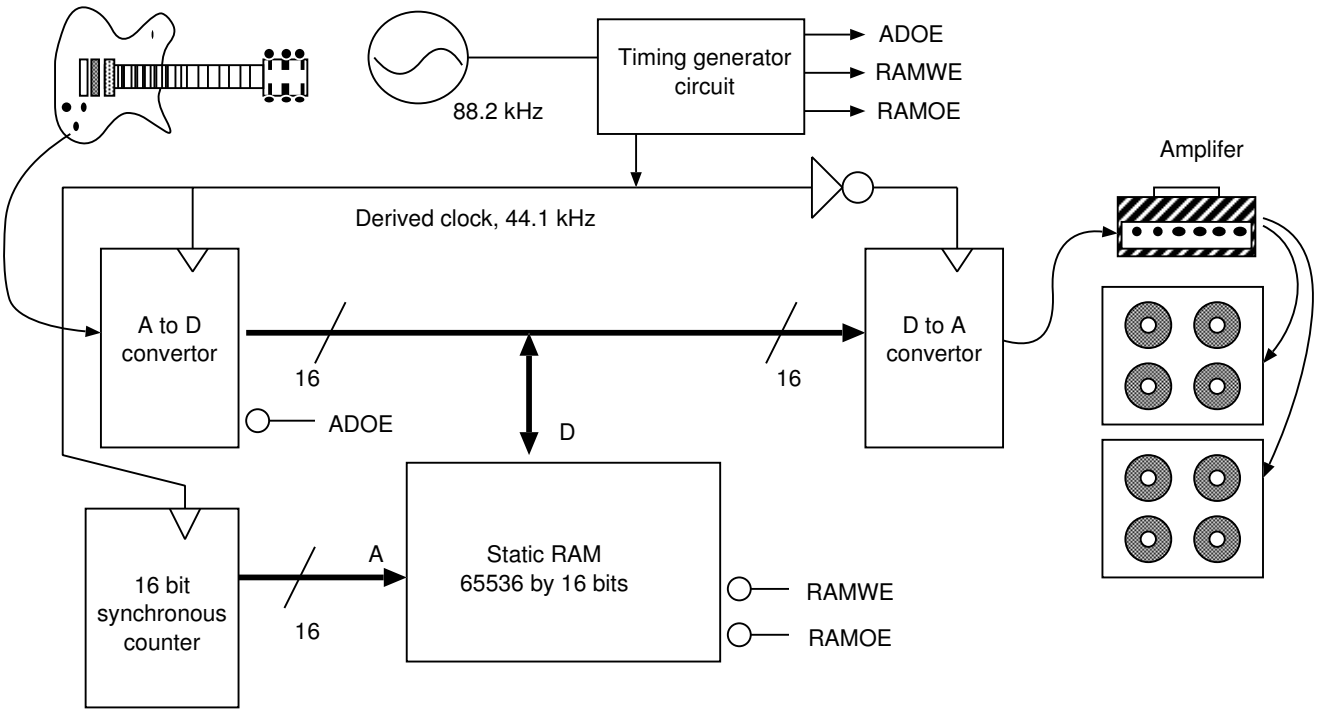
You did this too!

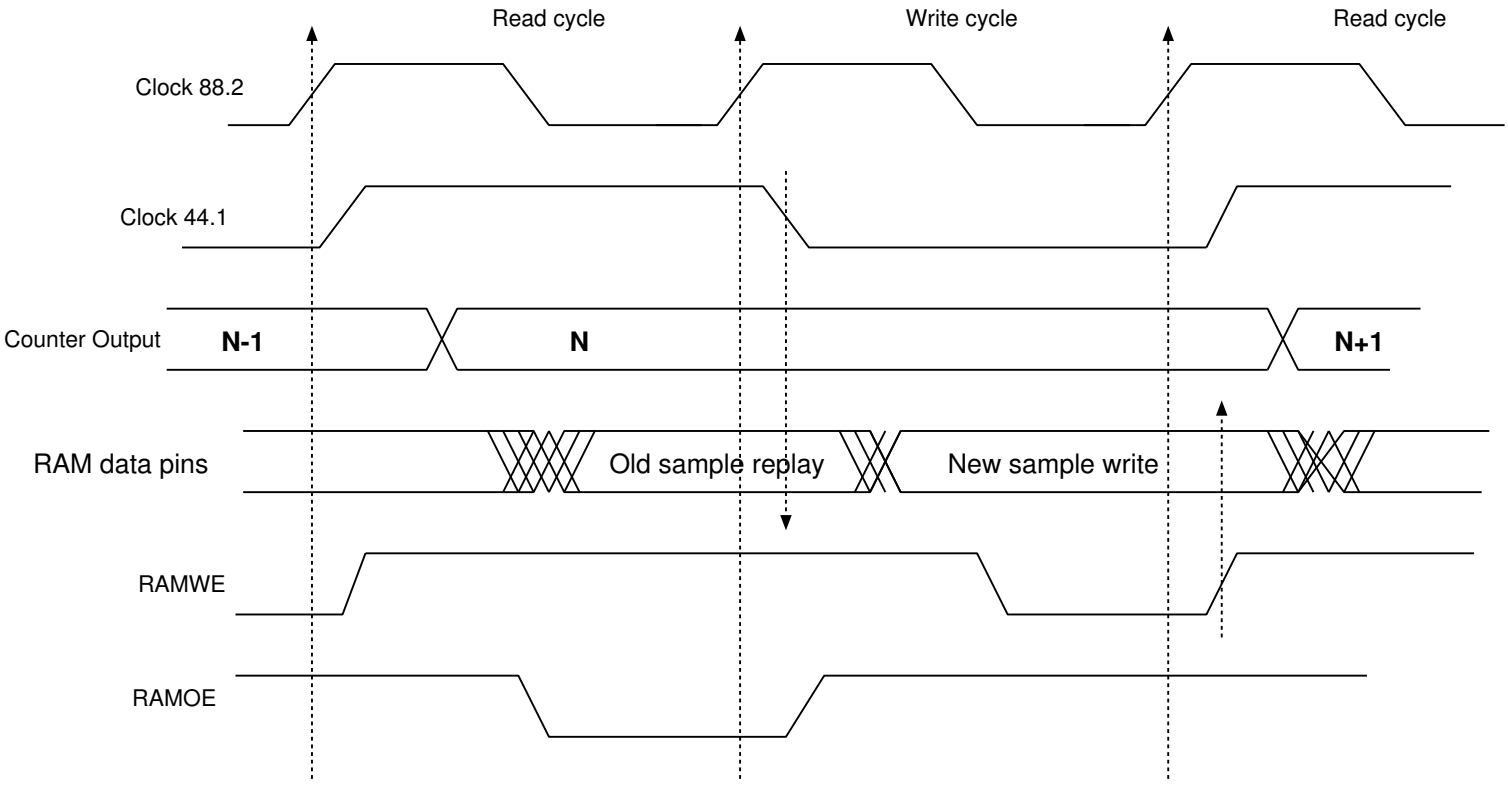
Use of a ROM as a function look-up table



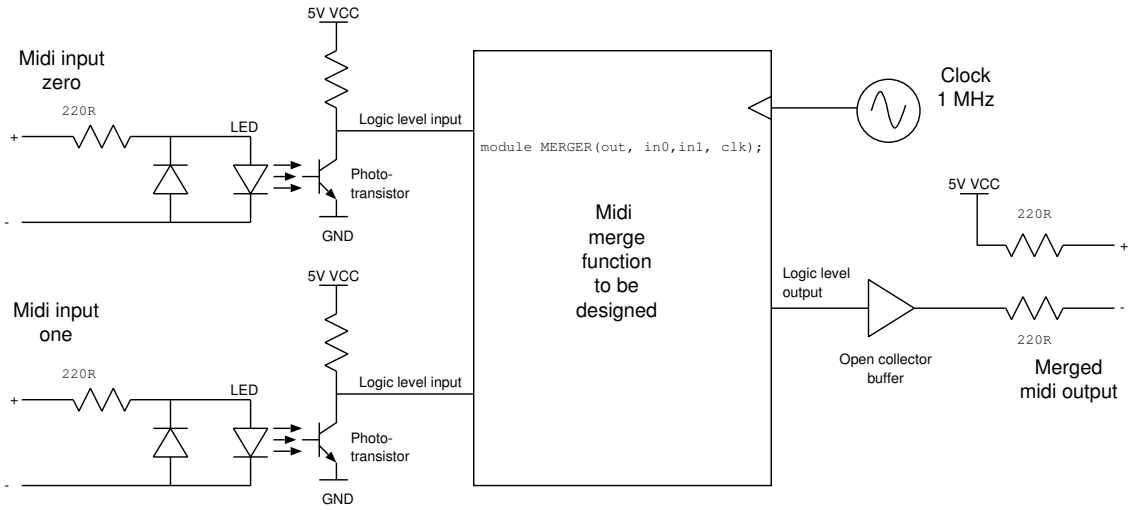
The ROM contains the exact imperfections of a 1950's valve amplifier.

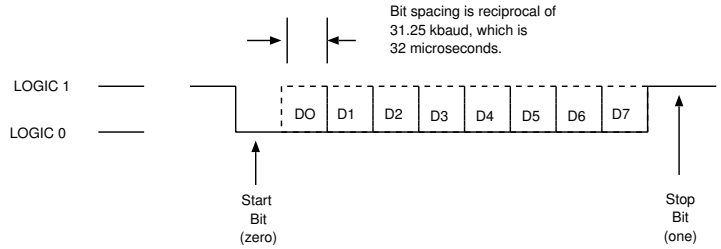
Use of an SRAM to make the delay required for an echo unit





Merge unit block diagram

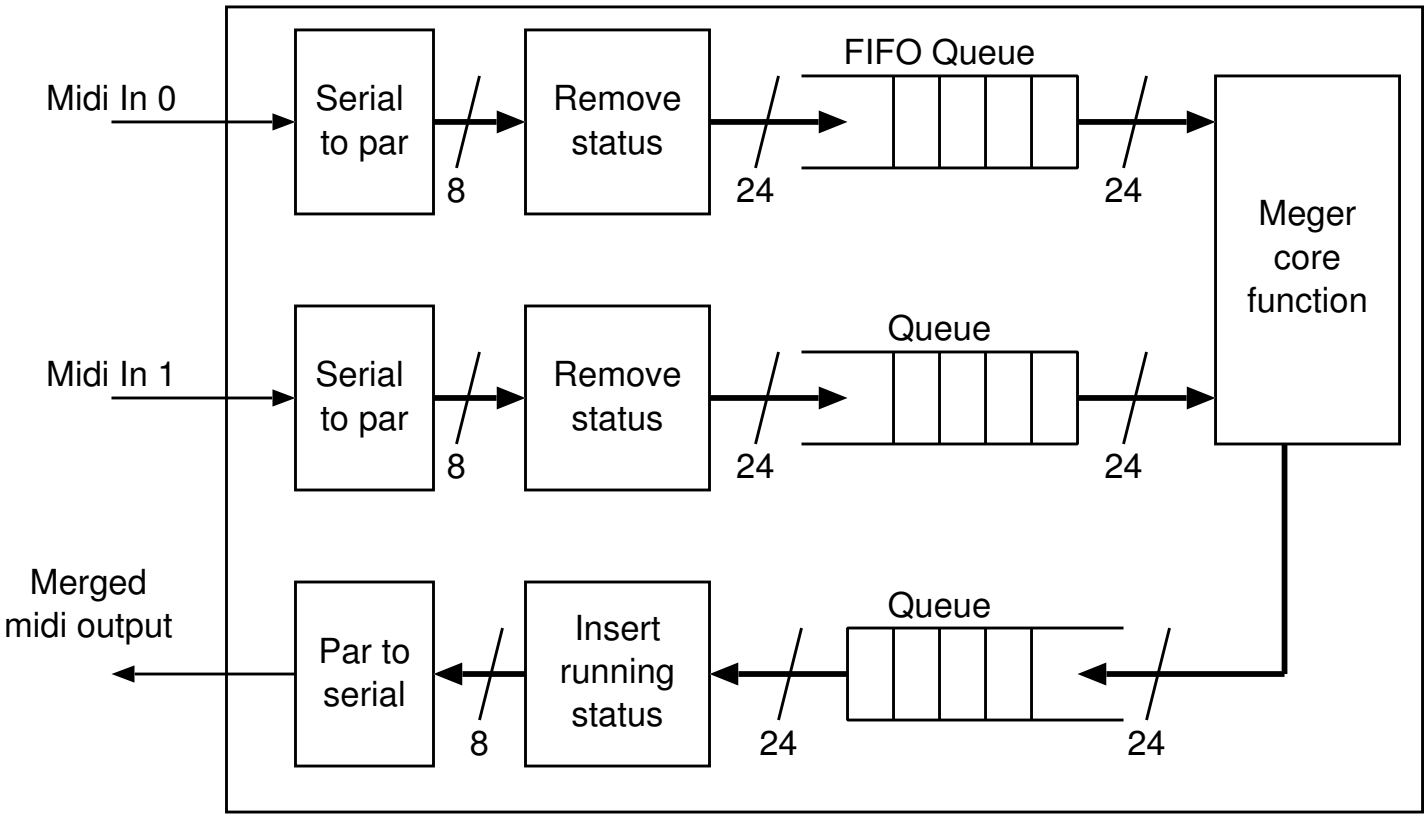




MIDI serial data format

```
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