# Structured Hardware Design
# 6L 1A Easter 2005.

Dr David Greaves

May 3, 2005





## Preface

These six lectures are to the part 1A 50 percent candidates in the Easter Term. This course concentrates on system-level design issues, but nonetheless, a good knowledge of the digital electronics course is required as background material.

Together with the reading below, the best preparation for this course is to look around at electronic equipment, from cell-phones, to toys, disco lights, to lifts to teletext televisions: ask yourself about their structure: what components these items are made from, how were they designed, what is done in hardware and what in software ?

These printed notes follow roughly the same structure as the material lectured but are not a primary reference in terms of defining the syllabus for this course.

## Bibliography (Book list)

Books related to the course are:

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 directly to the Digital Electronics course.

Randy H Katz. *'Contemporary logic design.'* Benjamin Cummings ISBN 0 8053 2703 7

Texas Instruments. *'System 74 Series Logic Family.'* Texas Instruments.

*'IEEE Transactions on Consumer Electronics'.*

# Glossary of jargon and acronyms.

| | |
|---|---|
| ALU | Arithmetic and logic unit. |
| ASIC | Application specific integrated circuit. |
| BICMOS | A process with both CMOS and bipolar transistors on one die. |
| CAD | Computer aided design. |
| CAE | Computer aided engineering. |
| CAM | Computer aided manufacture. |
| CLB | Configurable logic block. |
| CRC | Cyclic redundancy check (added to end of a data frame). |
| CMOS | Complementary metal oxide of silicon. |
| DRAM | Dynamic random access memory. |
| DMA | Direct memory access. |
| DSP | Digital signal processor/ing. |
| EDO | Extended Data Out for DRAMS. |
| ECL | Emitter coupled logic. |
| EDA | Electronic Design Automation (CAD tools). |
| EMC | Electromagnetic compatibility. |
| EMI | Electromagnetic ingress or i(m)missions (sic). |
| FET | Field effect transistor. |
| FSM | Finite state machine. |
| FFT | Fast Fourier Transform. |
| FPGA | Field-programmable gate array. |
| GaAs | Gallium Arsenide. |
| Gbps | Giga-bits per second |
| HDL | Hardware description language. |
| IDE | The bus used for connecting hard drives and CD roms |
| IEEE | Institute of electrical and electronic engineers. |
| IPR | Intellectual Property Rights. |
| IRDA | Infra-red data association |
| JTAG | Joint technical advisory group |
| LSI | Large scale integration (say about 5000 gates, 10000 transistors). |
| LAN | Local Area Network. |
| LCA | Logic cell array. |
| LCD | Liquid-crystal display. |
| LED | Light-emitting diode. |
| Mbps | Mega-bits per second |
| MBPS | Mega-bytes per second |
| MAC | Media Access Control. |
| MCM | Multi-chip module. |
| MSI | Medium scale integration (i.e. the larger TTL devices). |
| NRE | Non recurring engineering costs. i.e paid once per design. |
| OTP | One-time programmable |
| PAL | Programmable array logic. |
| PIO | Programmed Input and Output. |
| PHY | Physical Layer (of a LAN). |
| PCB | Printed circuit board. |
| PLL | Phase-locked loop. |
| PLA | Programmable logic array. |
| PIN | Personal identification number. |
| RTL | Register transfer level. |
| SoC | System on a chip. |
| SRAM | Static RAM. |
| SIMM | Single in-line memory module. |
| SSI | Small scale integration (i.e. about 4 gates on a chip). |
| TTL | Transistor-transistor logic. |
| UART | Universal asynchronous recevier and transmitter. |
| USB | Universal serial bus. |
| UWB | Ultra wide band radio. |
| Verilog | An HDL language in wide use (not an acronym). |
| VHDL | VHSIC Hardware Description Language. |
| VHSIC | Very High Speed Integrated Circuit. |
| VNL | Verilog netlist. |
| VCO | Voltage controlled oscillator. |
| VGA | Video graphics adaptor. |
| VLISW | Very long instruction word. |
| VLSI | Very large scale integration (big chips). |
| XNF | Xilinx netlist format. |

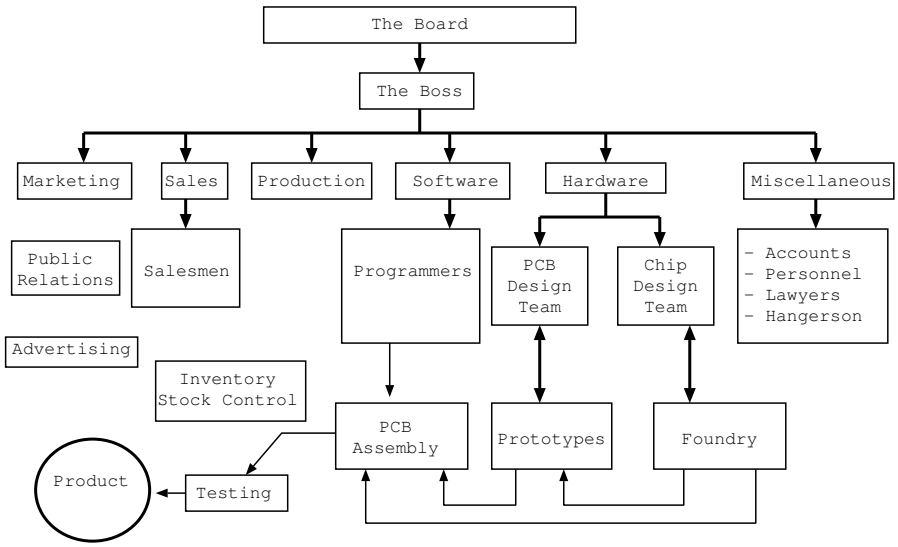



Figure 1: Structure of a small company with hardware products that use its own hardware, software and chips.

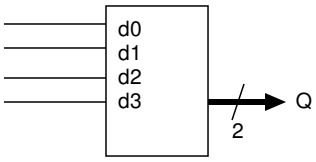
# Introduction

A good hardware design works the first time. A structured approach to its design is required if the system is complicated. Simulation and formal verification of the system behaviour before fabrication is also vital to help get it right. Multiple prototypes are normally tested in real situations before production runs commence. As Figure 1 shows, the final product relies on successful inter-working between custom chips and the other parts on the circuit boards, together with the software.

Most designs today are held in a database which contains details of custom circuits, printed circuit boards, other chips, software, part numbers and so on. The database can help automate every part of the design process, except for the initial design decisions. These decisions define the basic structure of the product and they are what this course is about.

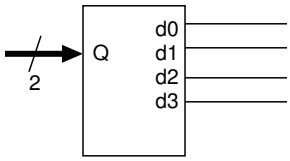
To create a new hardware design, as with software, it is important to use a top-down approach, starting with an accurate capture of the requirements specification and then decomposing the system into modules. As with software, the module must have well-defined functions and well-defined interfaces and the modules must be suitable for individual testing. However, two big differences are 1. that the modules in hardware are often made of different technology from one another, and 2. that the desirability of using pre-existing modules is much greater.

**Providing closer integration of software and hardware design processes is the main research area in EDA (Electronic Design Automation).**




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

Figure 2: **Priority Encoder (4 to 2)**




| 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 |

Figure 3: **Binary to Unary Decoder (2 to 4)**

# 1 Building Blocks in Hardware Systems.

*Learners' Guide:* What you should learn from this section is:

- Hardware designs are made of modules - used to be physical but now virtual.
- Modules are made of modules.
- Modules have easy to describe functions which could be tested in isolation from the rest of the system.
- Modules could be gates, collections of gates, chips, boards etc..
- Modules are designed to be re-usable.
- Examples of typical MSI modules and exposure to Verilog descriptions.

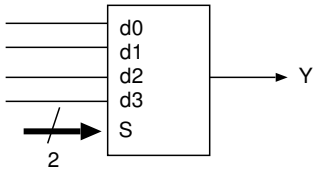
This section introduces a number of building blocks that are frequently used in hardware designs. A block may be an AND gate or it may be a PC. In times gone by, some blocks were refered to MSI (medium scale integration) and LSI (large scale integration) subsystems. Today, many of these blocks rarely exist as separate components: instead they are placed inside custom (or semi-custom) application-specific, integrated circuits (ASICs). Putting a PC inside a custom chip is not far off from being an everyday design step.

Verilog is a hardware description language (HDL). Certain fragments of Verilog are given throughout these notes. Verilog is not examinable for part Ia but is for Ib. There is more material in these notes than is given in the syllabus. Only the syllabus material is examinable.

## 1.1 Combinatorial (Combinational) Logic (Revision)

Figure 2 shows a four input priority encoder. A basic encoder with N inputs has $\log_2(N)$ outputs. If N is a power of 2, one of the inputs is a don't care or else an extra output is implemented that holds when any input is asserted.

```
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
```




| 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 |

Figure 4: **Multiplexor**

Figure 3 shows a two-to-four binary to unary decoder. When there are N inputs, there are (upto) $2^N$ outputs. In this figure, the inputs are called 'Q' and the outputs are called 'd': this unusual naming is given to aid visualisation that a chain of encoders and decoders could be formed, converting between binary and unary at every other stage.

```
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
```

Figure 4 shows a four-input multiplexor. When there are N inputs, there are $\log_2(N)$ selector inputs and a single output bit.

```
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
```

Note how the truth table for the multiplexor has been compressed using X to denote 'don't care' on the input side. Without using X's, 64 rows would be needed. The same notation was used for the priority encoder.

The multiplexor component can be implemented in a broadside fashion (as described in the next section). For instance, instead of taking N single bit inputs, it would take N words, each of some width $x$ bits, and produce a single $x$-bit output word. There would still be $\log_2(N)$ selector inputs however.

Figure 5 shows tri-state buffers. When the enable input is deasserted, the output pin is in the high-z state, which is as though the buffer was removed from the circuit. The tri-state buffer is used to produce a distributed multiplexor. Each tristate-buffer, and hence input to the multiplexor, can be placed on a separate printed circuit board and the interconnecting wire can be a trace on a motherboard. Examples are given later (e.g. in the small computer figure, figure 46 we see that each source driving the data bus is one input to a distributed multiplexor).

Figure 6 shows a general '*wired-or*' structure. A wire is pulled up to the logic one voltage by a resistor. A number of open-drain drivers are connected to it. Each driver may be on a different circuit card. When any driver turns on, it pulls the wire to logic zero. Overall, the output function $Y$ is the OR of the driver gate signals $a_i$. Wired-or gates are often used where a number of bus slots exist where each slot may or may not have a device plugged in. Any device that desires attention (such as signalling a processor interrupt) may turn on its driver to request attention. Although there can be multiple wires (e.g. IRQ1 though IRQ15 on a PC), frequently a number of

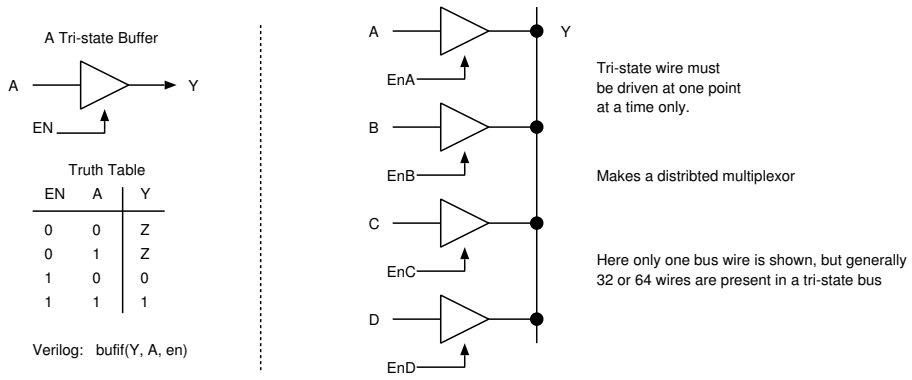



Figure 5: **Tri-state Distributed Multiplexor**

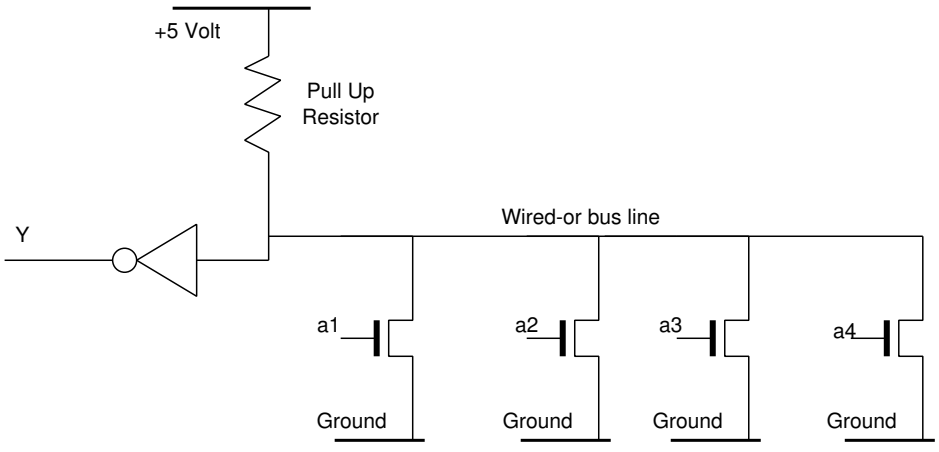



Figure 6: Wired-or distributed gate structure.

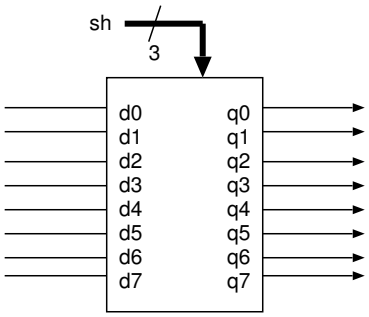



Figure 7: **An 8-Bit, Barrel Shifter**

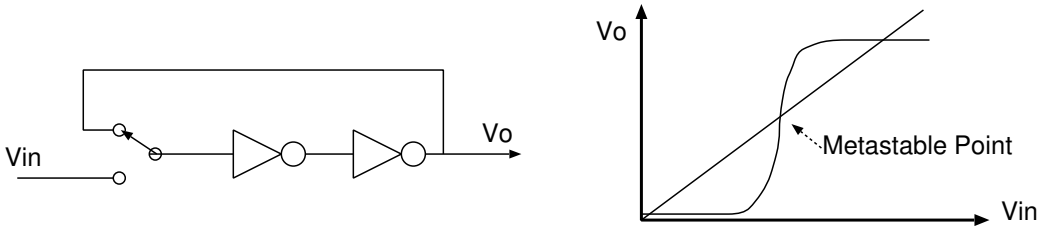



Figure 8: Basic principal of the bistable.

devices may be sharing one of them, and so the the system controller must use other means, such as polling, to find out which of the sharing devices sourced the request.

```
module BarrelShifter(d, sh, q);
   input [2:0] sh;
   input [7:0] d;
   output [7:0] y;
   assign y = (d << sh) |  (d >> (8-sh));
endmodule
```

Figure 7 shows an eight input barrel shifter or rotator. When there are N inputs, there are $\log_2(N)$ shift amount input bits. There are four forms of shift commonly used in computing: the rotate, the left shift, the logical shift right and the arithmetic shift right. Some barrel shifters might accept a further input to select which form of shift they implement.

*Exercise:* Draw out circuits, at the gate-level, for every component shown in this subsection. For the barrel shifter, a transistor-level circuit using pass transistors might be easier.

*Exercise:* The opposite to a multiplexor is a demultiplexor. Why is the demultiplexor not given a combinatorial circuit ?

## 1.2 Sequential Logic (Revision)

Figure 10 shows a pair of inverters in tandem. When the switch connects to a voltage source, the Vout/Vin plot can be taken. When the switch closes the loop, it is clear there are three points

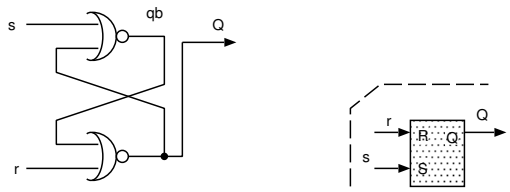



Figure 9: Adding inputs to the bistable: the RS latch.

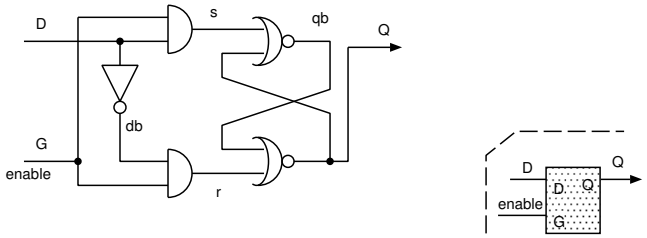



Figure 10: Transparent Latch and Schematic Symbol (inset)

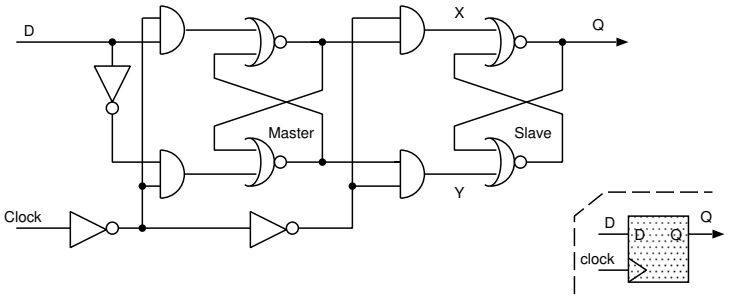



Figure 11: Gate-Level, Edge Triggered, Master-Slave, Flip-Flop, made from a pair of Transparent Latches in tandem.

where Vout=Vin. Two of these are stable, and hence this device is called a '*bistable*'. The third point is metastable.

Figure 10 shows a transparent latch. When its enable input is high, the Q output follows the D input after a small internal delay, but when the enable input is low, the internal RS latch maintains a stable output value. This is a '*level-sensitive*' device.

Figure 11 shows a pair of transparent latches in tandem to form an edge-triggered master slave, D-type flip-flop.

Figure 12 shows an improved D-type made of only six gates that also has asynchronous reset and preset inputs, and figure 13 shows the addition of active-low, asynchronous reset and preset inputs.

*Exercise:* Show how to convert an edge-triggered D-type into a transparent latch by enclosing it in a combinatorial circuit that uses a multiplexor. Sometimes we must resort to this approach when a transparent latch is needed in a technology that only makes available edge-triggered flops, such as in many FPGA families.

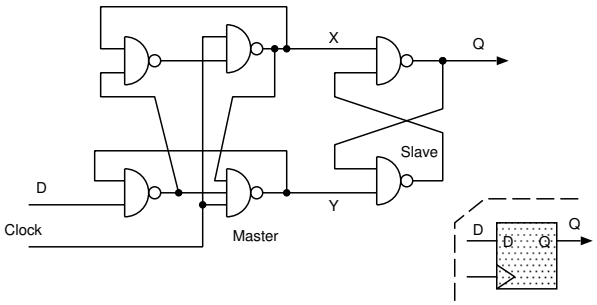



Figure 12: Six Gate, Master-Slave Flip Flop

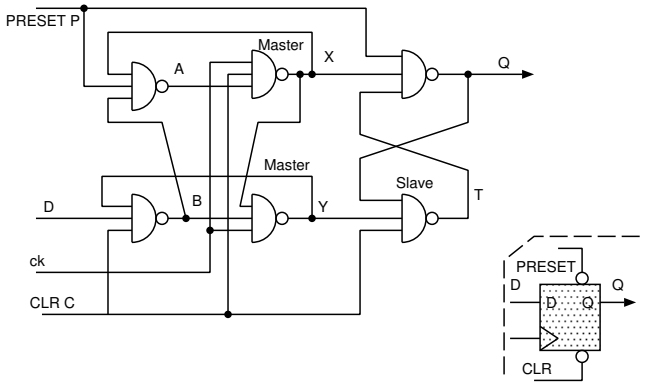



Figure 13: Flip Flop with Asynchronous Reset and Preset

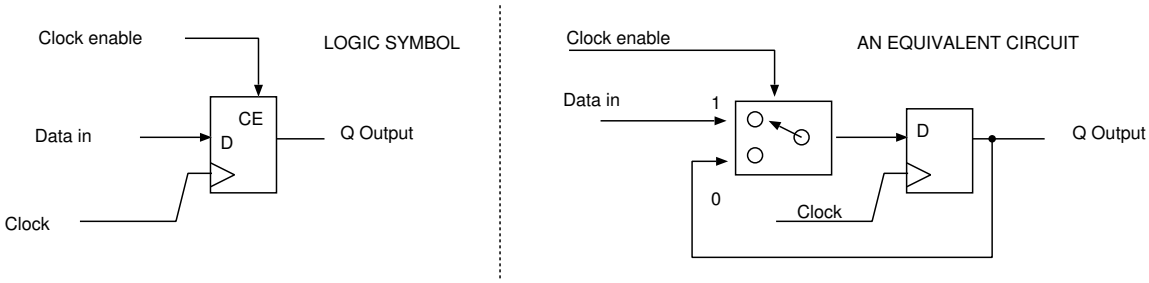



Figure 14: A D-type with clock enable and the equivalent circuit.

## 1.3 Clock Enables

Sometimes we do not wish a flip-flop to change its value every clock cycle. Thus we require something that performs a '*clock enable*'.

Figure 14 shows a D-type with a clock enable input and the equivalent circuit. The complexity of the additional multiplexor is sometimes not present in an implementation, since it can be adsorbed into the gate or transistor-level circuit that forms the D-type.

## 1.4 Broadside Ideas

A bus is a number of signals that operate in parallel to carry a binary number. In the schematics, a thicker line is used, and the number next to the slash is the number of signals in the bus.

Figure 15 demonstrates the concept of a broadside component, in this case, a broadside set of D-types. **To form a broadside component, the building block is instantiated a number of times with the data inputs and outputs connected to the member lines of input and output busses.** The control connections are put in parallel and driven from a single external source. A broadside set of flip-flops is a *register*.

A broadside register of $N$ bits is made out of $N$ D-types with a commoned clock input. It can

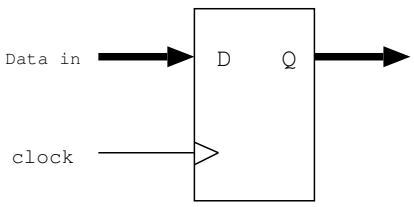



Figure 15: **Broadside Register**

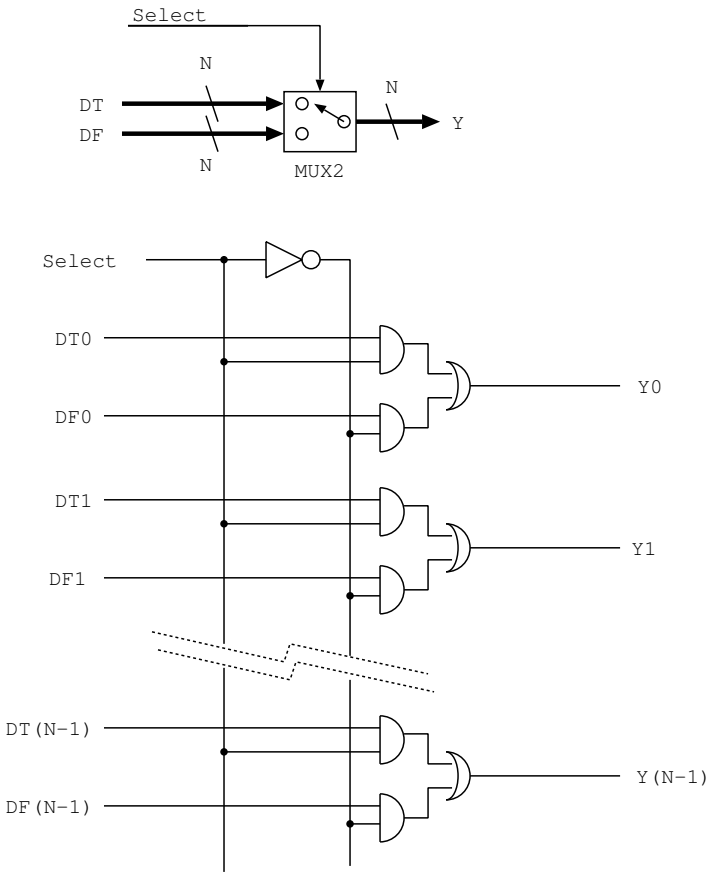



Figure 16: An N-bit broadside, two-to-one multiplexor.

hold $2^N$ different values.

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

In large designs, broadside registers with clock enables are very frequently used. When the clock enable does not hold, the contents of the register are not changed on the clock edge. Rather than gating the clock signal in anyway, this can be physically implemented by a signal routing that stores the previous data back in the register again. In RTL (register transfer level) HDLs, such as Verilog, a clock enable is inferred from an '`if`' statement. Any variables that are not assigned on a clock edge are required to hold their values, and are physically stored with their previous values.

```
reg [N-1:0] br_q;
always @(posedge clk) if (cen) br_q <= data_in;
```

This Verilog fragment has the same behaviour as

```
always @(posedge clk) br_q <= (cen) ? data_in: br_q;
```

Figure 16 shows the schematic and a suitable circuit implementation for a broadside, two-input multiplexor. Most logic systems provide two input multiplexors as a fundamental building block implemented at the transistor level, rather than forcing users to build them from gates.

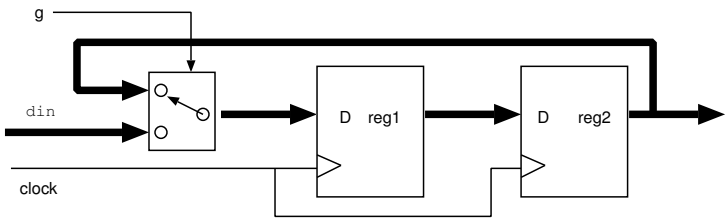



Figure 17: **Example: A pair of Registers Swap Values on ach Clock Cycle.**

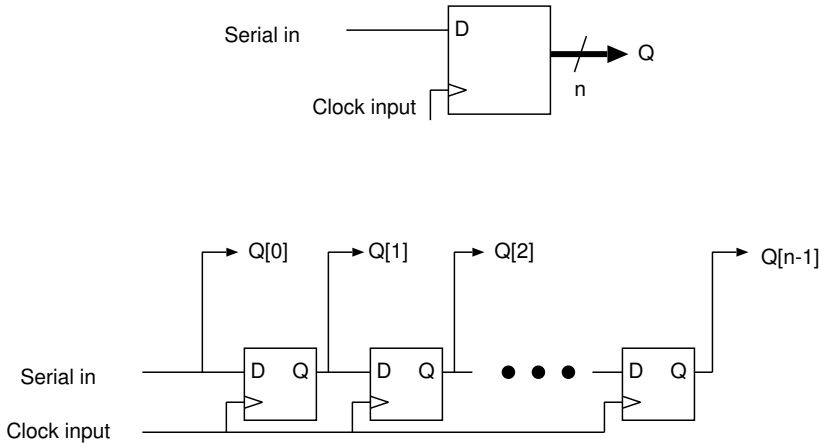



Figure 18: **An $n$-bit Shift Register:Schematic Symbol and Internal Circuit**

The following Verilog fragment swaps the values between a pair of registers if the guard is false, but a broadside multiplexer introduces a new value into the loop when the guard is enabled. The circuit for this is shown in figure 17.

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

It is the fundamental tenet of synchronous logic design that only one signal, the clock, needs to be distributed with tight timing tolerances, and that the sequential logic updates on the clock to a new state based on the current state.

## 1.5 Shift Register

In a shift register, on each active edge of the clock, the existing bits all move along one place and another bit enters from the serial input.

```
module ShiftReg8(Q, clk, din);
   input din, clk;
   output [7:0] Q;
   always @(posedge clk) Q <= (Q<<1) | din;
endmodule
```

A synchronous parallel load can be added to a shift register using an additional number of multiplexors. A synchronous load occurs on a clock edge when the load signal holds, otherwise the existing data shifts as normal. An asynchronous parallel load was commonly used in older designs (known as a jam load).

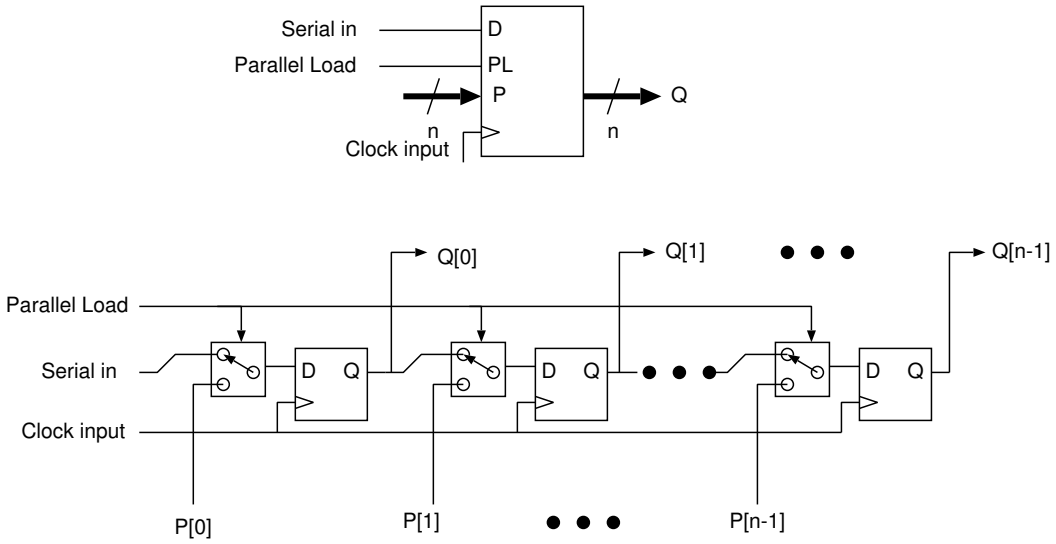



Figure 19: **Shift Register with Synchronous Parallel Load**

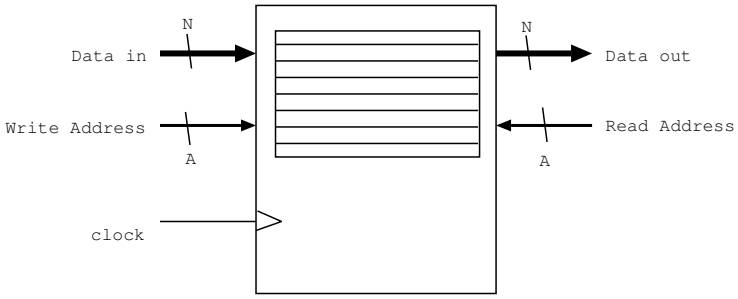



Figure 20: **A Register File: Schematic Symbol**

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

## 1.6 Register File

A register file of $M$ locations by $N$ bits is internally implemented as $M$ broadside registers of $N$ bits each. Figure 20 shows a register file with one read and one write port. $\text{Log}_2(M)$ bits of write address are required to select which port is written on the clock edge. A similar width address bus selects which register drives the output bus.

```
// 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];
```

Figure 21 shows the schematic symbol for a dual-port register file. This device internally contains a number of broadside registers and multiplexors, but they are grouped into the register file for

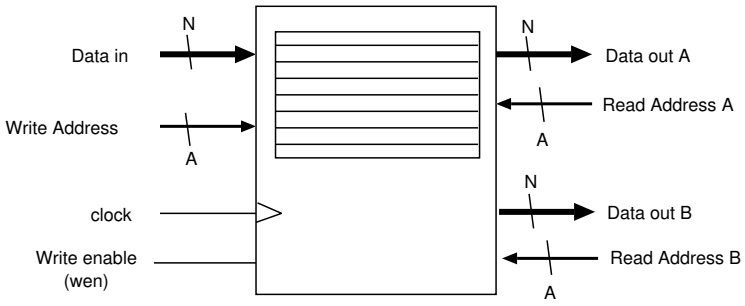



Figure 21: **Dual Read Ported Register File Symbol**

convenience of abstraction. Such a file is a key part of all current computers. The illustrated dual-port file has two reading ports and one writing port, so might be considered a triple-ported file. On the positive edge of the clock, the data on the write port is stored in the addressed register. The read process is identical to an SRAM: at any time the read address may be changed and the output data will change shortly afterwards.

*Exercise:* Sketch the circuitry that would turn a collection of clock-enabled broadside registers into a dual-port file.

## 1.7 Read Only Memories

Figure 22 shows the logic symbol for a read only memory (ROM). The contents are non-volatile and are placed in the ROM during manufacture using a fabrication mask or with a special programming step in the field. In figure 22, the ROM holds $M$ words of size $N$ bits. The word addressed on the $A$-bit address input port selects one of the $M = 2^A$ internal stored words and delivers it to the output port provided chip select and output enable are asserted (i.e. low since they are active low signals). When not enabled, the output pins are in the high-z state. In terms of schematic symbol, the ROM is essentially the RAM, but it has no write input.

The techniques for storing field programmed information are:

- W-Sn fuse: PROM (programmable read only memory) A one-time programmable technology that operates by melting selected tiny fuses on the surface of the chip. This is an old technology which is not very reliable or dense.
- Floating gate technology with ultraviolet erase. A floating gate is a transistor gate that has no direct electrical connection to any other conductor. Hence, a static electric charge will remain on the gate indefinitely. Hence, like the fuse, the transistor is permanently on or off. To make a ROM cell, an electrical charge is placed the selected floating gate using a super-voltage that causes electron tunnelling or other breakdowns in the insulator. Discharge is achieved using intense ultraviolet illumination to make the charge leak away. When used for a ROM, gives an EPROM.

  A variant on EPROM is an OTP-EPROM (one-time-programmable) which the same device packaged in a windowless package at much lower cost.
- Electrically reprogrammable floating gate devices: EAPROM or EAROM. Again charge is stored on the gate, but electrical techniques used for programming can be applied for erasing in large blocks. Used widely in personal organisers and memory cards and in most modern PAL devices. Today, Intel's Flash memory is the most widely used EAROM and we tend to call it simply 'Flash'.

Mask programmed ROMs of several megabytes may cost sub 50 pence each in quantities of tens of thousands. Flash memory may be more than 100 times more expensive. The choice of which technology to use depends on the number of units that are going to be made and the expected number of reprogramming cycles required.

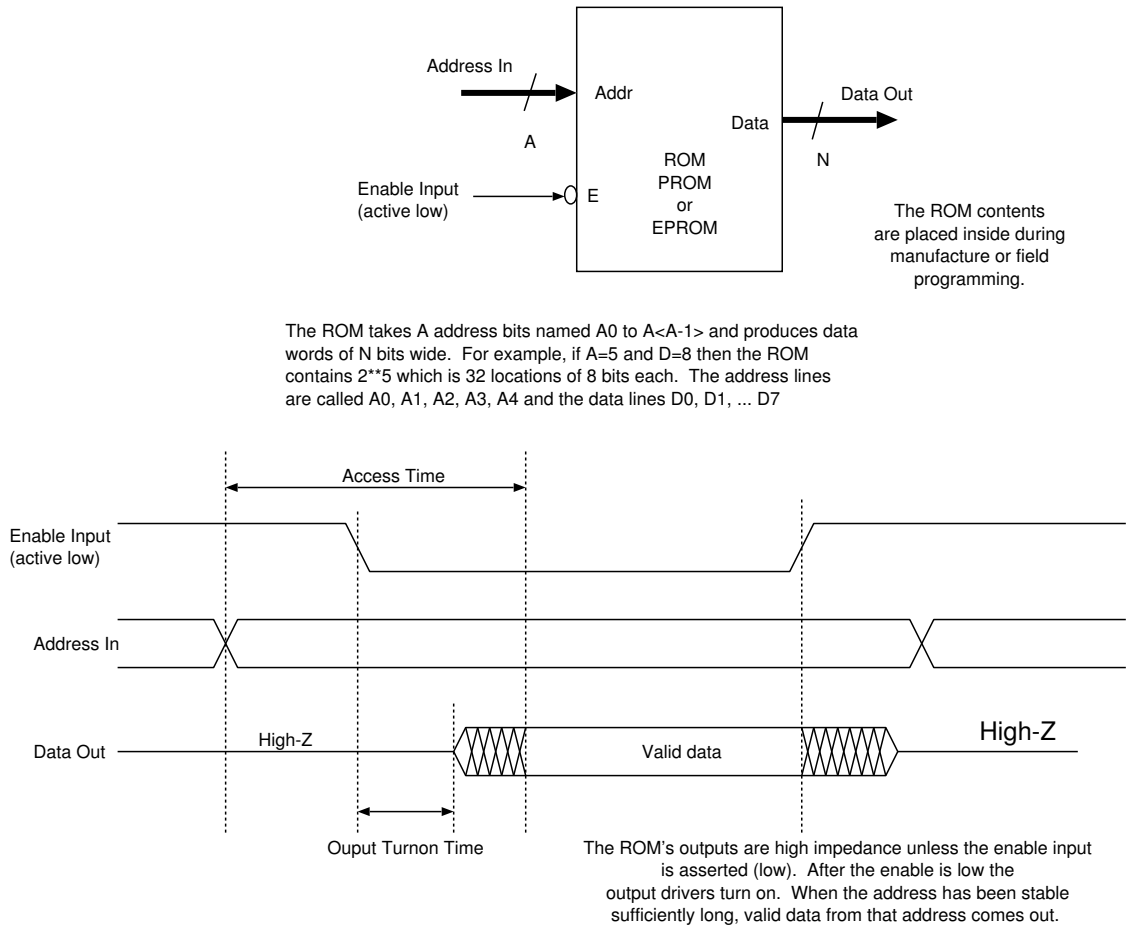



Figure 22: Read Only Memory (ROM).

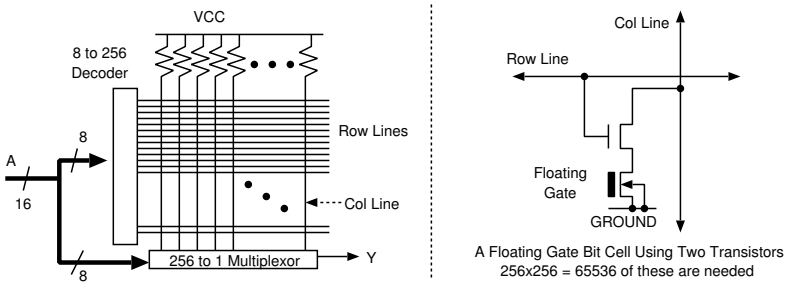



Figure 23: Flash Rom (64K locations by 1 bit) Read Structure and Bit Cell.

## 1.8 Non-Volatile Memories

A non-volatile may be written by the user and retains its contents without external power.

EAPROM/EAROMs (electrically erasable programmable read only memories) include 'Flash' memory. These can be programmed and erased electrically and are used in PCMCIA memory cards and USB memory sticks, digital cameras and so on. Modern flash memory has read access times similar to SRAM, but write times may be several orders slower and erase times are much slower. The erase and write times can be limited by the startup times of internal power supplies that produce the special programming voltages from the single supply rail used when only reading.

Figure 23 gives the general structure for a one bit wide ROM that is implemented as a square array of bit cells. Half of the address lines are used to enable one horizontal row line and all bits on that row that are zero pull down their column line. The remaining column lines are pulled to one by the pullup resistors. For a masked ROM, each bit cell would either contain a transistor or not, depending on whether a zero is wanted or not. For a FLASH memory, each cell contains a pair of transistors, one with a floating gate. A floating gate that is charged with respect to the substrate of the chip will be result in a the bit cell being zero, whereas if there is no charge on the gate, the transistor will be always off and a one will be read. The two transistor cell can be implemented using a single composite transistor. Recent technologies use three or four different charge quantities on the transistor and analog readout techniques to increase density. The circuitry to get charge on and off the gate is not shown in the figure.

Magnetic bubble memory is also used for non-volatile storage. It is strongly resilient against radiation and so is attractive to for military applications.

BB-RAM: Battery back up of the supply to a small SRAM is also widely used instead of EAROM for non-volatile storage. There is then no write cycle limitation (total cycles or speed). A small lithium battery is found in many phones, motherboards and PDAs because there is a requirement to maintain the real-time clock when main power is removed. This battery also feeds a small RAM to hold settings and data when system supply is removed.

| Name | Persistence | Read Speed | Write Rate | Erase Time | Comment |
|---|---|---|---|---|---|
| RAM | Volatile | Same as SRAM | Same as SRAM | not needed | |
| BB-RAM | Non-volatile | Same as SRAM | Same as SRAM | not needed | Battery Life issues |
| Mask PROM | Non-volatile | Same as SRAM | Not possible | Not Possible | |
| EPROM | Non-volatile | Same as SRAM | 10 us/byte | 20 Mins | Needs UV window |
| Sn-W PROM | Non-volatile | Same as SRAM | 10 us/byte | Not possible | |
| EAROM | Non-volatile | Same as SRAM | 10 us/byte | 100 ms/block | write cycle limit |

## 1.9 RAM memories

Figure 24 shows a random access memory or RAM. It is as equally random access as a ROM, so is a misnomer. RAM is normally volatile, meaning it loses its contents when power is disconnected. Static RAM (SRAM) normally uses one flip-flop per bit stored. A transparent latch suffices, so it is not as complicated as having a D-type per bit. A possible internal structure for an SRAM is shown in Figure 27. However this would lead to a long, skinny chip. In practice, a square array of storage elements is used, meaning that the on-chip tri-state bus is much wider than the external data bus. The on-chip bus will have width approximately equal to the square root of the number of bits stored in the device. Very large RAM chips will have multiple smaller square arrays. Modern SRAM devices have an access time of 20 nanoseconds and have capacities of about half a megabyte. Price is about six pounds per megabyte, but depends greatly on the access time required. Second-level cache chips for PCs are SRAMs and produced in high volume, so are currently a good way to purchase SRAM, even for other designs or uses.

A property of the SRAM, shared with the ROM, is that there is no clock and, during read, the output is essentially a combinatorial logic function of the address input value.

When writing, the address must not change and the data present on the data pins at the back edge of the (active low) write pulse is the data actually written.

A Static Random Access Memory (SRAM) of $M$ words of size $n$ bits each has a bidirectional $n$ bit

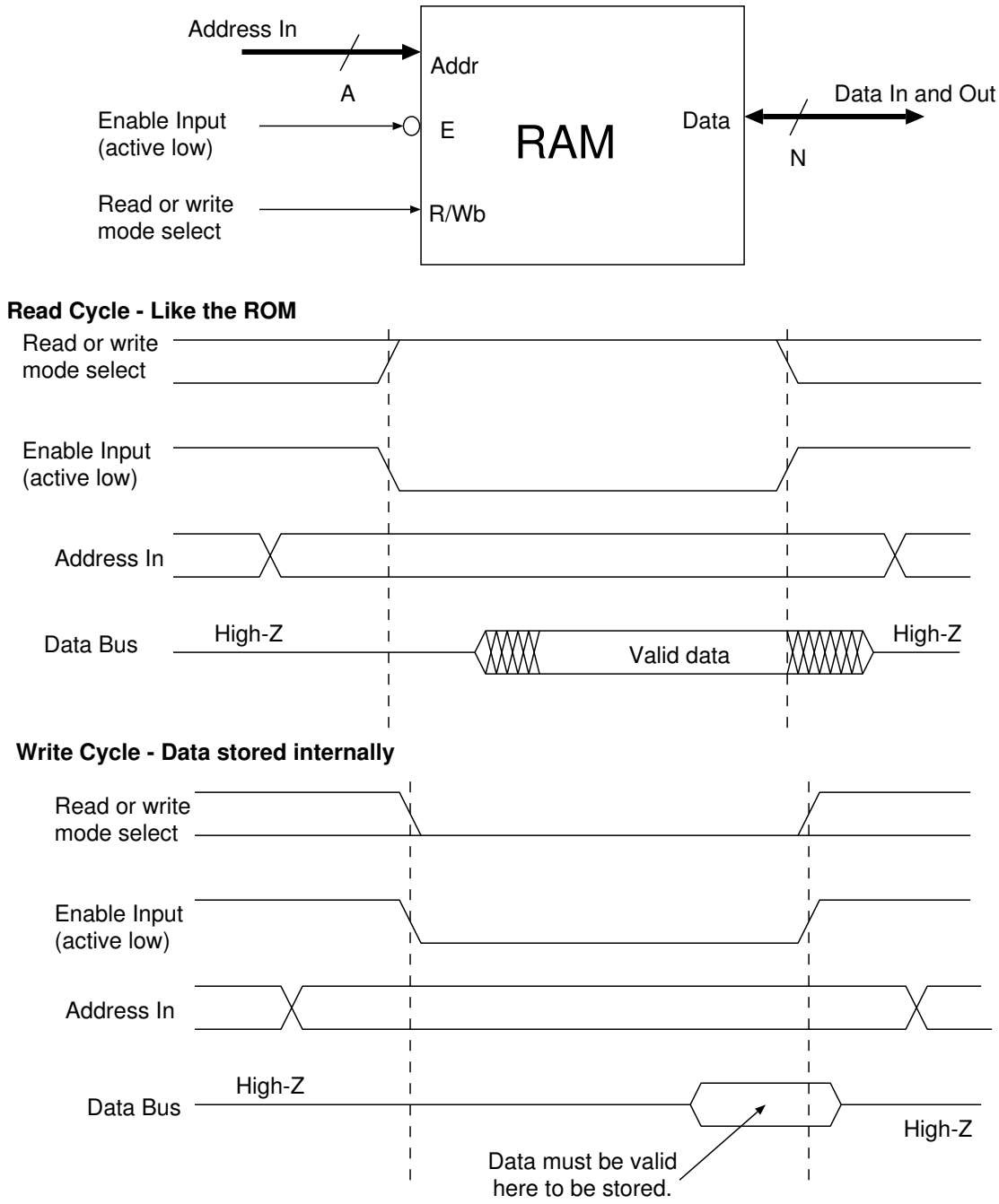



Figure 24: Read and Write Memory (RAM).

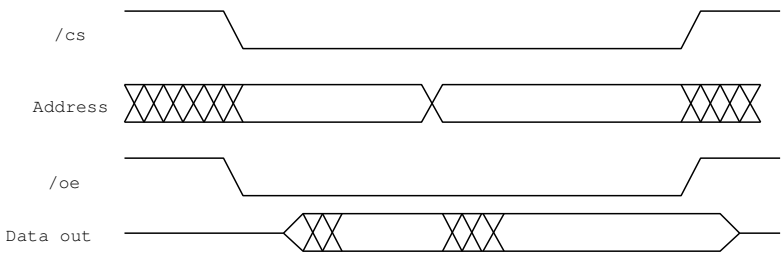



Figure 25: **RAM Memory Timing Diagram - Read Cycle**

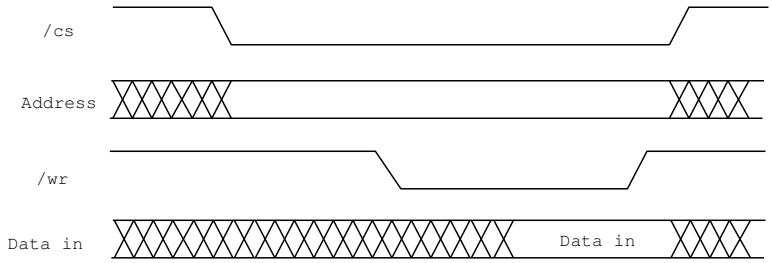



Figure 26: **RAM Memory Timing Diagram - Write Cycle**

data port and an A-bit ($A = log_2(M)$) address input port. The write enable and output enable control inputs are normally active low. The chip select input (also active low) must be asserted (ie logic zero) before the device will recognise a read or write cycle. The address input may be changed during a read, but should not be changed during a write cycle.

RAM is made from RS latches not D-types. This is possible because the RAM device does not have a clock input and is not edge-triggered on any of its inputs. This saves half the transistors compared with a master-slave (clocked) memory, such as the earlier register file.

Memory size is often given in kilobytes (kB) or megabytes (MB).

| | | |
|---|---|---|
| 1 kB | = | 1024 bytes |
| 1 MB | = | 1024 kB |

Within modern ASICs, registered RAM is frequently used. This is RAM that has broadside registers on its address and data inputs or data outputs or both. Functionally, it is identical, but there is a delay in implementing any operation. In essence, latency is traded for throughput using pipelining (ref section 3.8).

## 1.10 DRAM

Figure 28 shows the schematic symbol for a dynamic RAM. DRAMs store their data not in bistables, but in small capacitors. The value stored leaks away over time unless refreshed. It is also destroyed by readout: after readout, the device always performs an internal write-back cycle to restore the voltages on the capacitors. This means that the external logic must not start a new cycle immediately. Therefore the cycle time and access times of DRAM are different. Typically, today, DRAM cycle and access times are 60 and 120 nanoseconds and capacity of a single chip is 2 to 4 megabyte.

All DRAMs are accessed by applying the address in two halves. First a row address is presented and RAS taken low, then a column address is applied, and CAS is taken low. These row and column addresses originally corresponded to the actual physical topology of the rectangular array of bits on the device, but today the high order CAS bits select one of many internal arrays on the silicon chip. In external view, some DRAMs are one bit wide and others are four.

DRAM is much cheaper than SRAM owing to the smaller silicon area needed per bit. Smaller packages are also possible, owing to the multiplexed address bus.

Refresh cycles do not transfer data in or out of the chip, but must be performed sufficiently often anyway. A typical specification is that 512 refresh cycles are executed every 4 milliseconds. Since DRAM consumes most of its power when being refreshed, a designer who puts all of the refresh cycles in one batch at the end of every 4 millisecond period makes the design of power supplies much more difficult.

DRAMs are often packed on SIMMs (single in line memory modules). or DIMMS. Capacities of SIMMs are upto 512 Mbyte.

A few advances have occurred in recent years. EDO (extended data out) DRAMs guaranteed that the data will be valid on the output for a period after CAS is deasserted, thereby easing system timings and helping the designer. High performance systems today use synchronous DRAMs (SDRAM) which have a clock input and include broadside registering on the address or data paths. This gives a pipeline delay (section 3.8). The latency is increased in return for greater

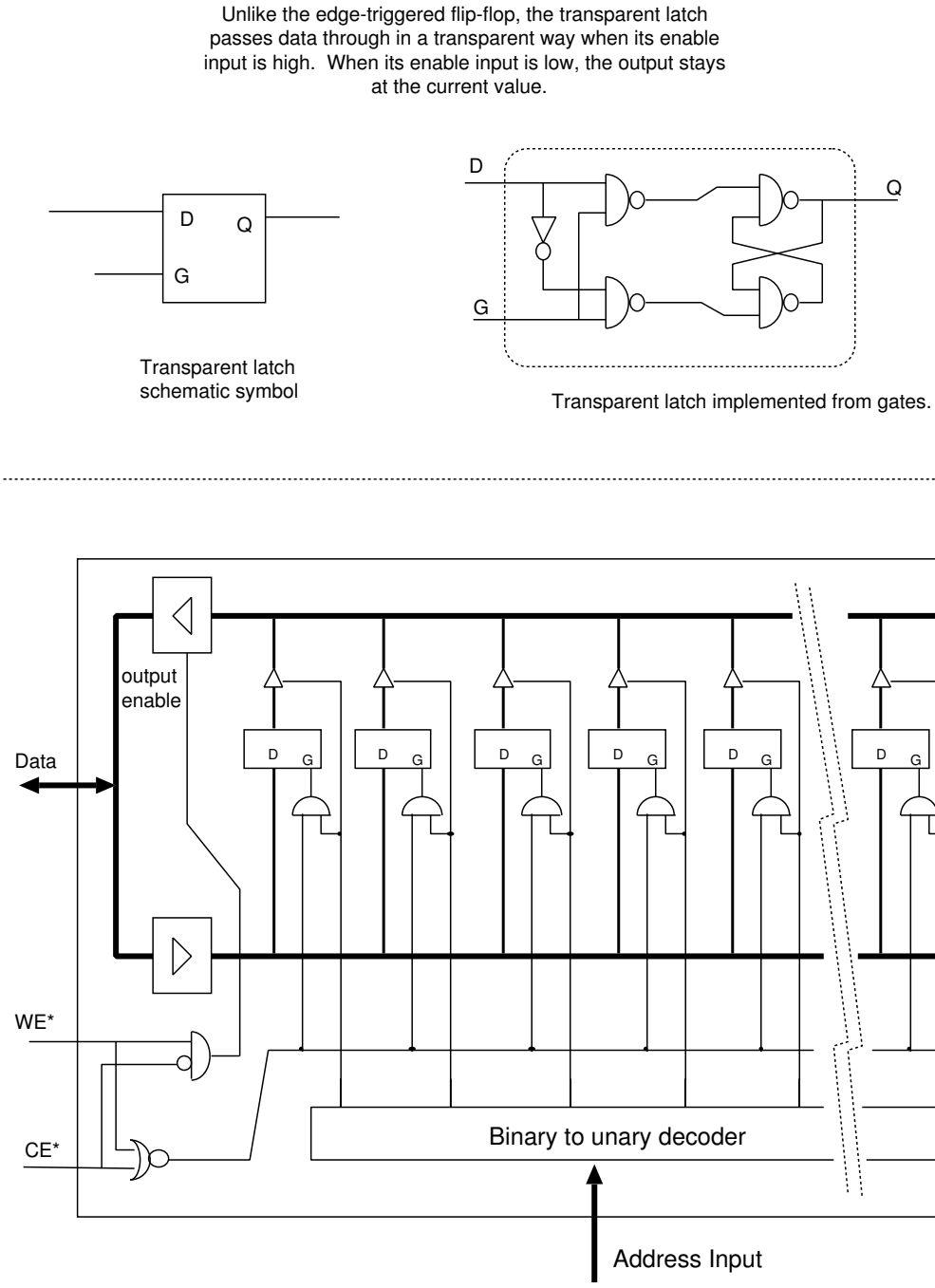
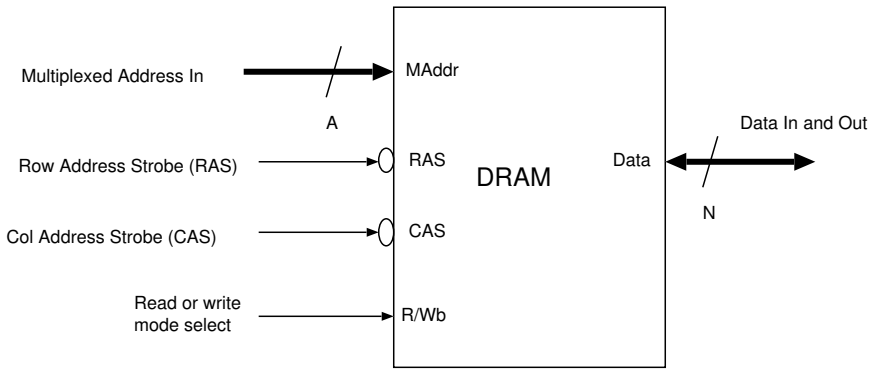
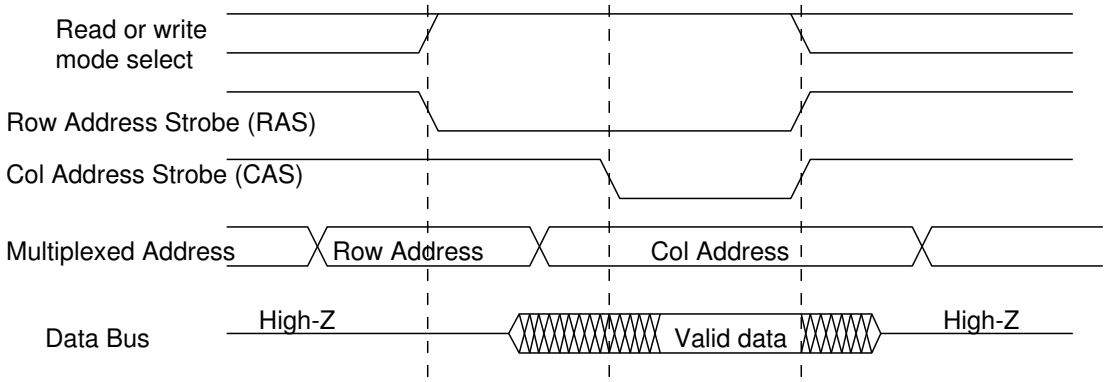



Figure 27: Schematic and possible implementation of a transparent latch and a naive implementation of an SRAM using broadside transparent latches and a distributed multiplexor made of tri-states.




**Read Cycle (write is similar)**

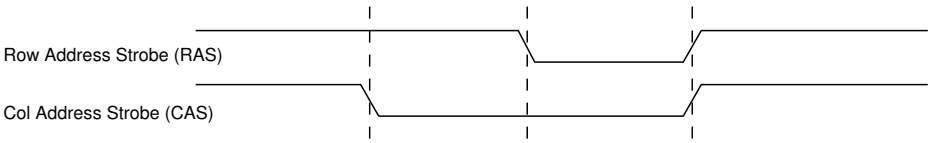



A DRAM has a multiplexed address bus and the address is presented in two halves, known as row and column addresses. So the capacity is 4**A x D. A 4 Mbit DRAM might have A=10 and D=4.

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.

Figure 28: Dynamic RAM (DRAM).

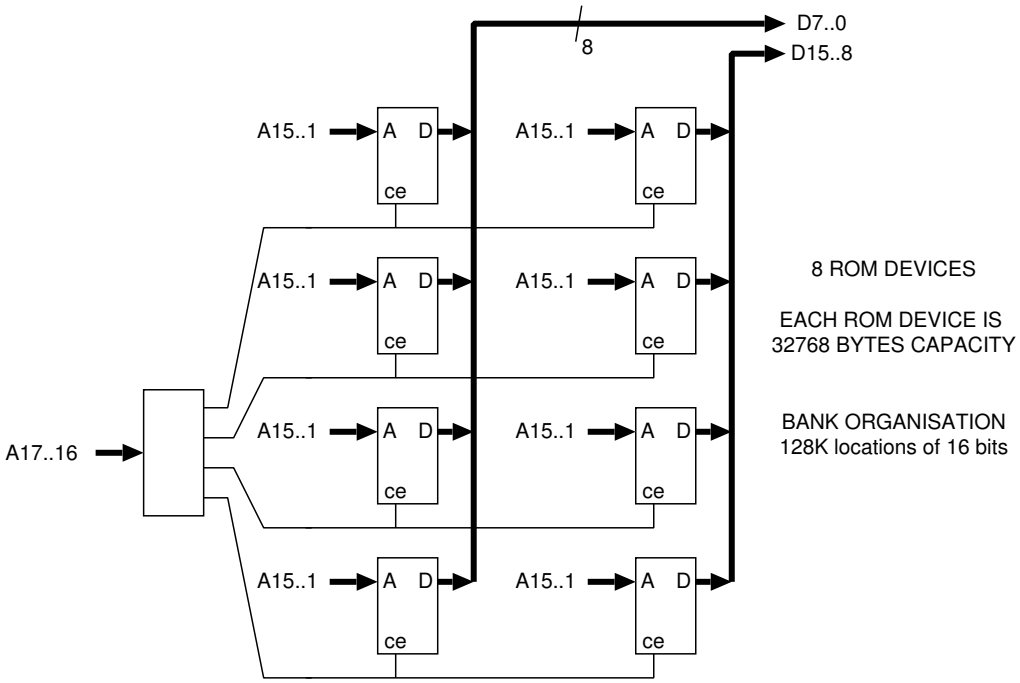



Figure 29: Forming a Larger Memory Bank from Multiple Smaller Devices.

throughput (e.g. 100 or 200 Mwords per second). Most recently, double rate SDRAMs have been introduced, that transfer data on both edges of the clock signal. However, DRAM performance has lacked growth compared with microprocessor performance.

## 1.11 Memory Banks.

Commonly, multiple, identical memory chips are wired in arrays to form a memory bank. This is done to increase performance and storage capacity with respect to the individual chip.

By wiring a number of chips with their address and control lines all commoned, we obtain a wider data bus that has greater bandwidth.

Alternatively, by wiring a number of chips with their data lines commoned we can use a binary to unary decoder to gain additional, high-order address lines that drive the individual chip-select inputs. This just increases capacity.

Figure 29 shows an example of constructing a larger ROM memory using a bank of multiple devices that uses banking in both dimensions.

In certain specialist applications, for instance high speed data capture from a radar or UWB receiver, memory arrays are arranged in temporal banks such that all their inputs are delayed by different fractions of a write cycle. This form of banking gives increased effective write rate and increased capacity.

*Exercise:* When would you use SRAM and when DRAM ?
*Exercise:* Sketch the circuitry that could be put around a bank of DRAM to make it appear like an SRAM with a single, non-multiplexed address bus and similar control signals. What aspects of the DRAM would you not be able to 'hide' in your circuitry?

## 1.12 FIFOs

Figure 30 shows a schematic symbol for a FIFO. This has $M$-bit input and output ports. Internally, it can store some number of $M$-bit words. It has two clock inputs. On the positive edge of the write clock, if the write enable holds and the device is not full, a new word is stored. On the positive

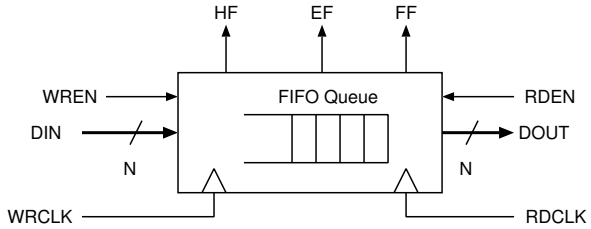



Figure 30: First-in, First Out (FIFO) Memory.

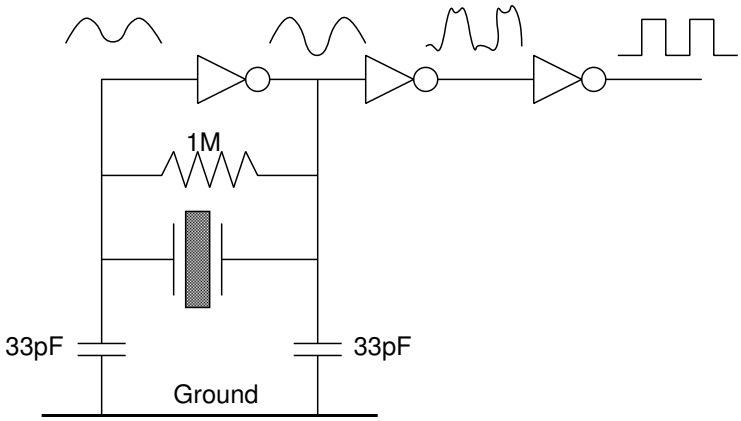



Figure 31: A crystal oscillator clock source.

edge of the read clock, if the device is not empty and read enable holds, then the oldest word from inside is presented at the output pins. Three flag outputs show full, half-full and empty status. Internally, the FIFO can be implemented as an SRAM with read and write pointer registers, or it can be implemented as a chain of broadside registers with a controller made of an asynchronous logic.

## 1.13 Miscellaneous Components

Figure 31 shows a circuit frequently used as a clock source. **A carefully cut slither of quartz crystal is used as the timing reference.** Quartz is piezoelectric, so can be made to change shape by applying an AC signal across metal coatings applied to its sides. Also, as it changes shape, it generates an equivalent electrical signal. Since the speed of sound in quartz is very high, a slice about 100 microns thick will resonate at several MHz. The details of the circuit are beyond the scope of this course, but suffice to note that the sine waves generated by the crystal become good quality square waves, suitable for use as a clock, after passing through a few inverters. A crystal might cost many cents to add to a design, but is accurate to about 50 parts per million in frequency, so is highly useful when compared with the RC oscillator, that can drift by percentage points with age, temperature and supply voltage.

Quartz crystals are comparatively expensive to manufacture. Figure 32 shows a cheaper circuit that serves as a clock when accuracies of only a few percent are needed. This is the RC oscillator. The first inverter must be a special Schmitt inverter with abrupt switching and hysteresis for the circuit to work. Hysteresis is a property, illustrated in the Vout/Vin characteristic whereby the forward trace is different from the reverse trace: there are two switching voltages. The capacitor is the only component that is hard to make on an integrated circuit, so a single pin on the device serves to connect the external capacitor. The crystal oscillator always needs two pins - another cost distinguisher.

Figure 33 shows the typical arrangement of clock distribution inside a large, modern IC. The clock is actually provided from a lower-frequency external reference and multiplied up internally with

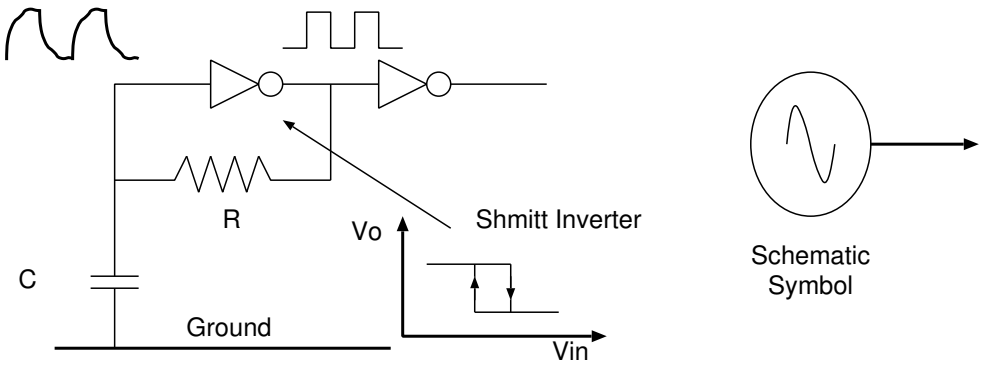



Figure 32: An RC oscillator clock source.

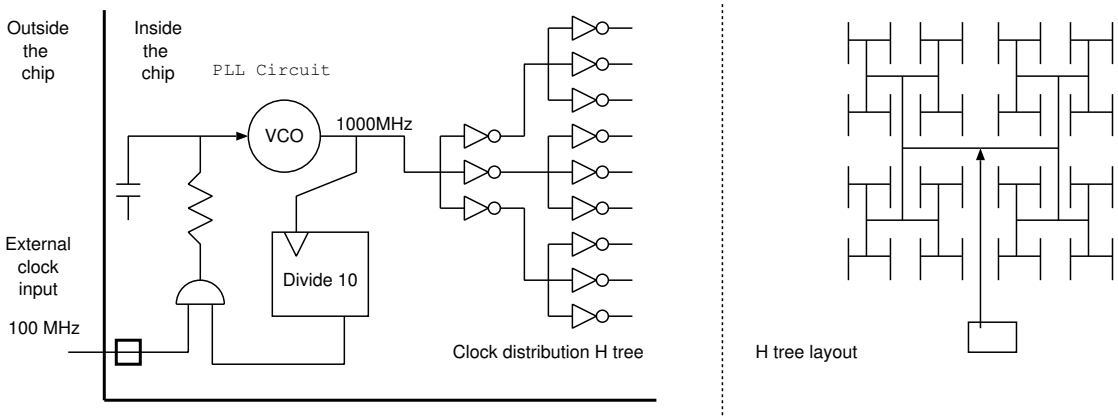



Figure 33: Typical clock multiplication and distribution system for a large IC.

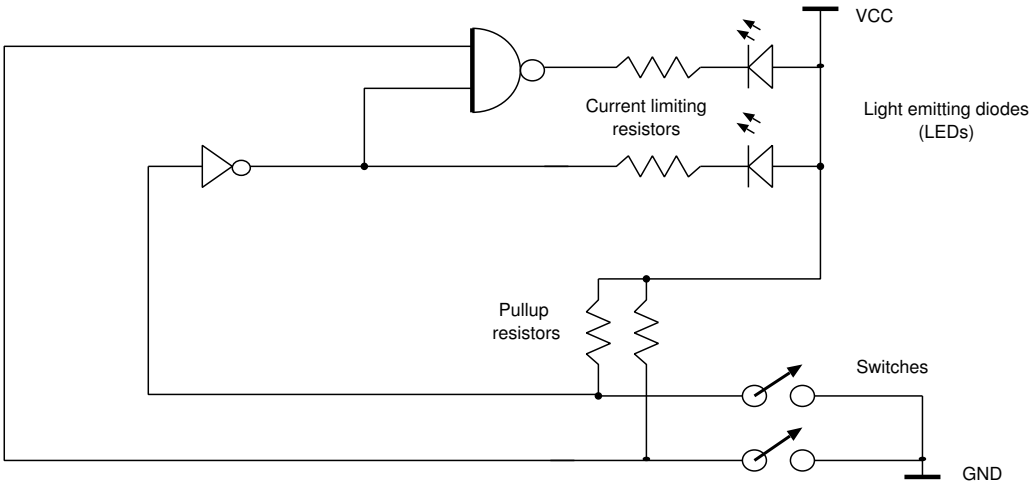



Figure 34: Connection of simple LEDs and Switches to Digital Logic.

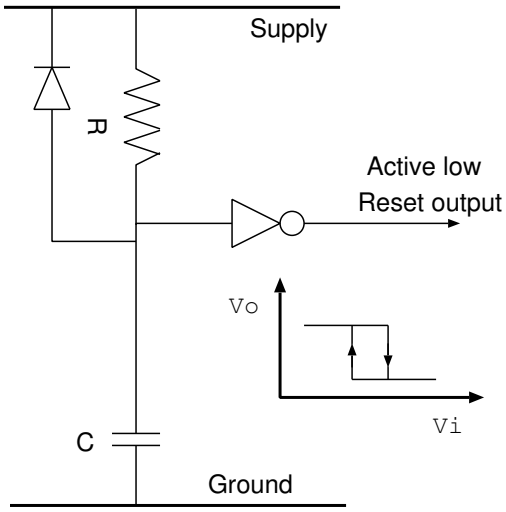



Figure 35: A circuit often used as a power up reset.

a phase-locked loop. **The clock is brought on to the chip at a lower frequency than required since it is easier to handle lower frequency signals on PCBs and the lower frequency may be one that can be directly generated by a crystal.** Skew in the delivery to the various parts of the device is minimised by using a balanced clock distribution tree. The physical layout can be as a fractal of H's, ensuring equal wire length to every destination. Inverters are used instead of non-inverting buffers to minimise pulse shrinkage (duty-cycle distortion).

Figure 34 illustrates digital input from switches and output to LEDs. When a switch is closed, it connects a net to ground, the logic zero voltage. Current flows through the pull up resistor to ground and is wasted. When the switch is open, the pull-up resistor pulls the net to the logic one voltage. For most types of logic, once the net is at that voltage, no further current flows in the resistor. A value of 10 to 100 Kohms is used for such a pullup resistor.

The LEDs are wired so that when a logic one is produced, they have no current flowing and so are off. When a logic zero is produced, current flows from the VCC supply into the output of the logic gate. The current limiting resistor limits the current to a suitable value, such as 20 mA. It is usual to wire LEDs in this negative logic fashion because many logic families are more able to sink current that source it at their outputs.

Figure 35 shows a circuit that is often used to generate a reset signal when a system is switched on. When power is applied, the capacitor is initially discharged, so has zero volts across it. Therefore

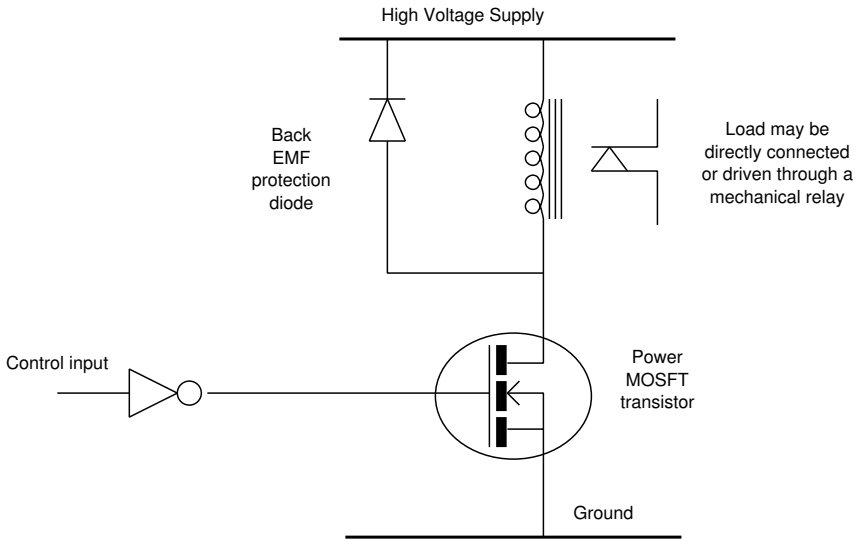



Figure 36: A typical structure used to drive a heavy current or high-voltage load.

the output is a logic one. The resistor charges the capacitor and after an interval of time the input to the inverter crosses the switching threshold of the Schmitt input and the output from the circuit is deasserted. Typical delay times used are about half a second, which is sufficiently long for the power supply to become stable and for oscillators to settle down. The diode discharges the capacitor into the supply rail when power is removed. On most devices, a reset switch is also fitted, which discharges the capacitor when pressed, therefore starting the device as from power up. **The reset signal is normally arranged to reset every flip-flop in every chip** (except for static RAM cells).

Figure 36 shows a typical structure used to drive a high power load from a logic signal. The semiconductor processing used to fabricate logic devices may be too expensive to waste on large transistors or may not be able to tolerate the higher voltages required to control the load. Therefore, external driver transistors in individual packages or in power driver ICs are used. Typical applications are motor, loudspeaker, solenoid, relay, and print head driving.

The EMF diode protects the transistor from the large voltage (so-called back EMF) produced when the current in an inductive load is suddenly removed.

A power MOSFET may have an on resistance of 0.01 ohms and an active channel area of several square centimetres. It is therefore millions of times larger than the sub micron gate's used in logic circuits.

Figure 37 shows the circuit sometimes used to get a clean signal out of a mechanical switch. In such a switch, when one contact hits another, it bounces off with a supersonic ping, which means that the circuit is made and broken a few thousand times for each switching operation executed. Using a double-throw switch and an RS latch formed from a pair of cross-coupled NAND gates, a clean output is generated. The systems works on the basis that the *first* time the commuting contact hits the stator contact, the latch changes state, and stays there until the switch is moved all the way back to the other side, where the reverse happens. A single-throw switch must be de-bounced using a timer.

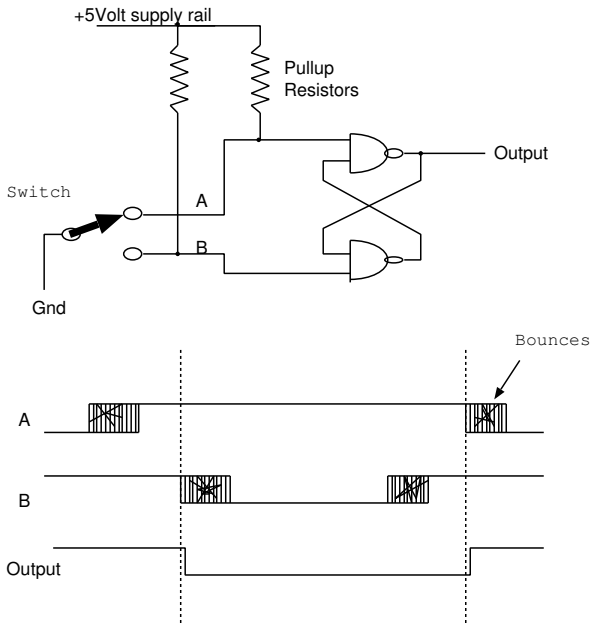



Figure 37: A de-bouncer circuit for a double-throw switch.

# 2 Further Blocks. Arithmetic and Micro-sequencers.

Next we consider the blocks used specifically for arithmetic, computation and computers.

## 2.1 Arithmetic Logic Unit and ALU circuits

The arithmetic and logic unit (ALU) is a fundamental block of all digital computers. Figure 38 shows the schematic symbol for an ALU. The ALU itself is combinatorial, but it is shown connected to a flag register, that possesses a clock input. For a detailed example, look at the 74181 in the Texas System 74 databook or online.

The illustrated ALU has a pair of $N$-bit inputs and a 4 bit function code input. The output is also N-bit. The function code determines what function of the two inputs is computed. Typical functions are add, add with carry, bitwise AND and OR, subtract and identity functions of the two inputs. The instruction set for the ARM microprocessor contains 16 data manipulation instructions that serve as a good example of the 16 most useful functions that an ALU can perform.

Like all combinatorial logic functions, the ALU is guaranteed to compute its output within a fixed time interval from the last input change. The value of this delay is typically dependent on the carry chain speed inside the ALU and it will set the maximal clock frequency of a simple microprocessor.

```
input [7:0] A, B;
input [3:0] 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
     ...
```

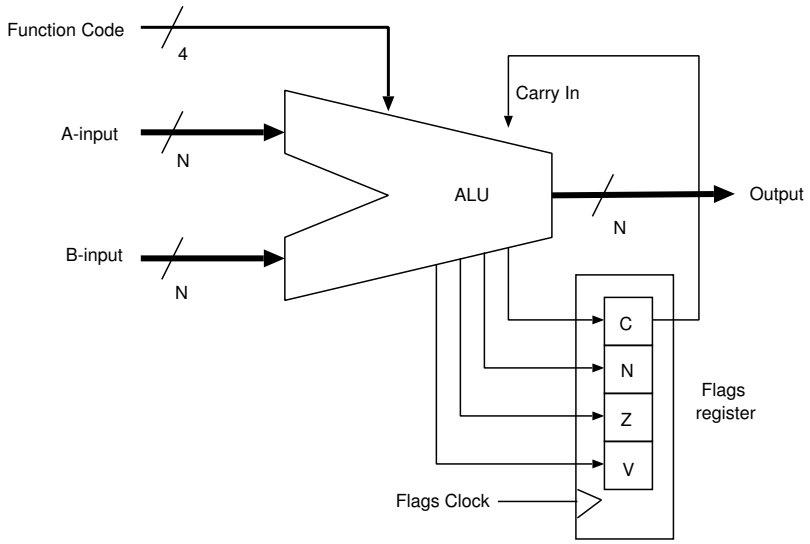



Figure 38: Schematic symbol for an ALU, connected to a flags register.

```
    endcase


assign Z = (Y == 0);
assign N = y[7];
```

*Question:* Suggest why ALU's typically only support addition and subtraction as arithmetic operations and not multiplication or division ?

A register file with two read ports is often used in conjunction with an Arithmetic Logic Unit (ALU). A counter can be used to index through 'function generator' ROMs that contain settings for the various control wires. Such an arrangement is called a micro-sequencer. As we shall see, a similar such arrangement also forms a central processor unit (CPU) or microprocessor, except that the function generator ROMs are then programmable rather than fixed.

Figure 39 is an illustrative circuit showing the interaction of an ALU and a register file. **These two units together form the heart of the execution unit of a computer.** In this circuit, since it is relatively straightforward, the addressed register is both read and written at the same time. Even though the address is incremented each time, it is important to understand that the old value read is updated in the ALU and written back to the same location. Accordingly, the only possible interference between one register and another is through the carry flip-flop, which will allow the output of one ALU operation to affect the behaviour of the next.

A pair of function generators are illustrated. These are just combinatorial blocks and could be considered as ROMs. To achieve some particular function these could be hardwired or programmed. In very early computers plugboards were used for such functions.

*Exercise:* If the ALU function code generator is programmed to specify 'A+B' when the input is one and 'B+Cin' otherwise, and the A input function generator is programmed to produce a constant value of 1, what have we produced overall? *Hint: Register zero is incremented each time and the others only when a carry is generated in the previous addition.* Will this unit complete its cycle this universe lifetime ?

## 2.2 Multiplier

Multiplication is a combinatorial function of quadratic complexity and for small numbers of bits it may be implemented as a combinatorial function that executes in less than a system clock cycle. Interested readers should consult Google about Wallace Trees and Carry-Save Adders. Such a multiplier is called a 'flash' multiplier (not to be confused with Intel's EAPROM technology of the

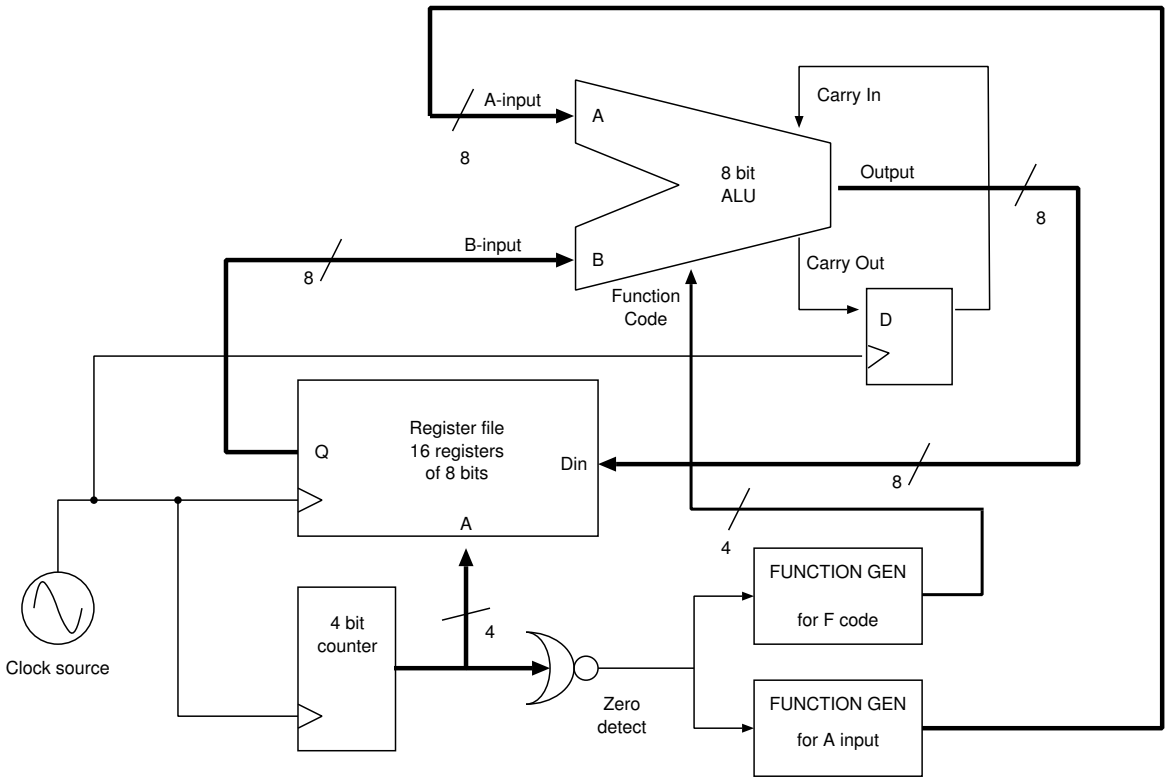



Figure 39: Illustrative Structure using an ALU and Register File.

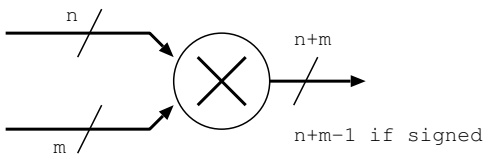



Figure 40: **Flash Multiplier Logic Symbol**

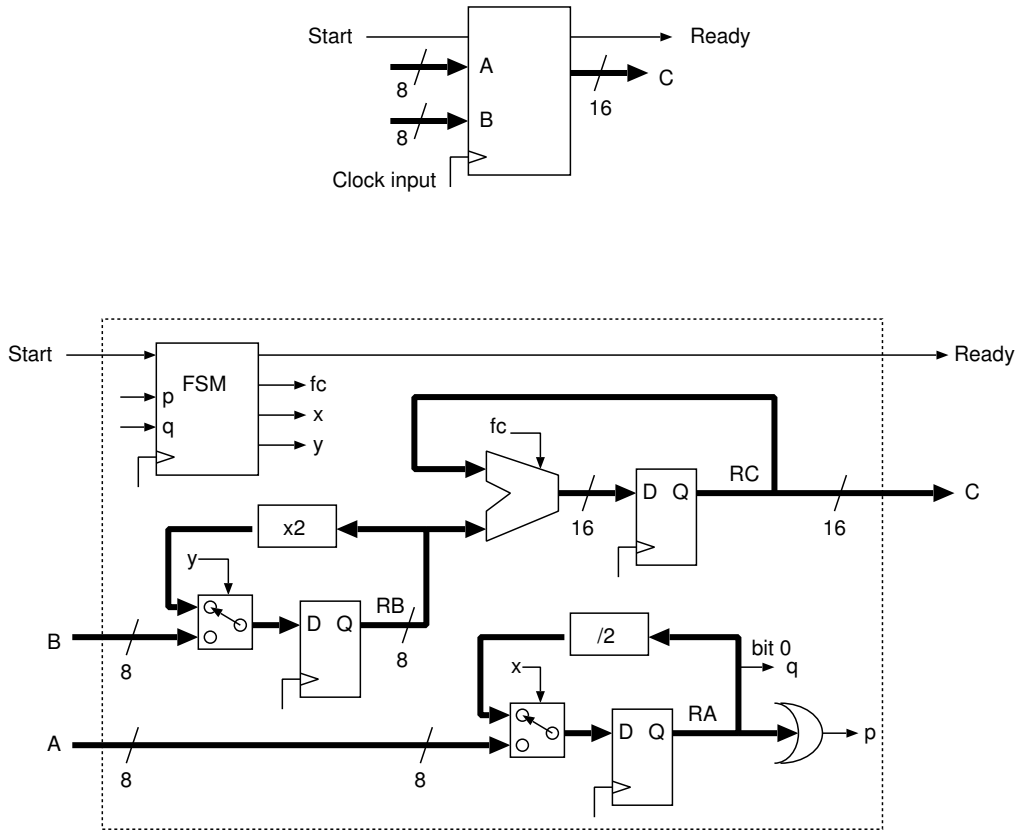



Figure 41: **Simple Sequential Multiplier: Schematic symbol and Internal Structure**

same name) and can consume considerable chip area. More commonly, a sequential multiplier is used.

A simple sequential multiplier is implemented with the following long-multiplication algorithm

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

Figure 41 shows a hardware architecture suitable for performing long multiplication. A finite state machine serves as the sequencer to manage the handshake signals and to control the internal multiplexors and ALU function code. The user must set up the values on the input busses before asserting the start signal, and then wait for the ready output before reading the result from the output bus. The internal datapath can be readily generated by considering for each register the sources it can be stored with, and providing multiplexors accordingly. An ALU that implements identity functions also serves as a multiplexor for this purpose. Note that left and right shifts by a constant amount are implemented using just wiring.

Custom datapaths along with their micro-sequencer, are today often synthesised automatically from imperative source code like that shown (lookup custom VLISW CPU Synthesis on the web).

*Exercise:* Design the sequencer for the simple multiplier shown.

Most sequential multipliers actually use Booths algorithm. Booth uses a single adder/subtractor to implement a base-four long multiplication.

```
(* 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
;

(* Booth's multiplier is twice as fast as binary long multiplication
  but still only uses a single adder.  The trick is that the adder is
  sometimes used as a subtractor, using the identity 3=4-1.  The
  multiplication and division and modulus by powers of 2 are all
  performed with wiring and so require no gates to perform. (The
  fixed-width addition of the carry to form n is considered part of the
  control logic, rather than counting as a datapath adder.)
 *)
```

*Exercise:* Design a datapath based around an ALU and register file, together with some control logic, that implements Booth's algorithm.

*Exercise:* The motivation for two's complement representation is that the same adders and subtractors can be used as for unsigned. Decide whether two's complement helps in this way for multiplication.

*Exercise:* A so-called 'quarter-squares' flash multiplier is implemented using ROMs that have been pre-programmed with the function $q = a^2$ and some adders or subtractors. The solution uses a rearrangement of the identity $(a+b)^2 = a^2 + 2ab + b^2$. Draw a block diagram of such a multiplier.

A flash divider is rarely implemented in hardware owing to its enormous complexity. On the other hand, sequential circuits for long division are not much more complex than those for long multiplication using this algorithm. However, in general, designers avoid division and instead prefer to multiply by reciprocals or use logarithmic data and subtractors.

## 2.3 Microprocessor Circuits

Figure 42 shows the schematic symbol and internal block diagram for a microprocessor. The internal block diagram is beyond the scope of this course, since it is coupled with a subject known as programming, but the external view is fair game.

The main features of the microprocessor are that it has a clock and reset input and it has an address bus output and a bi-directional data bus. The microprocessor generates read and write cycles on its busses and these are essentially identical in form to the waves shown in figure 24 for the SRAM.

The microprocessor places addresses on the address bus and sets the read/write signal to show the direction of the intended transfer on the data bus. It then raises its operation request output (opreq) and either places the data to be written on the data bus (for a write) or expects the addressed device to drive the data bus with its information (for a read).

**The behaviour of the microprocessor is to fetch an instruction from the reset location and execute it. If the instruction is a jump, then the next instruction may not be the one following it. Executing an instruction either modifies internal registers in the microprocessor or loads or stores data to the devices connected on the address and data bus.**

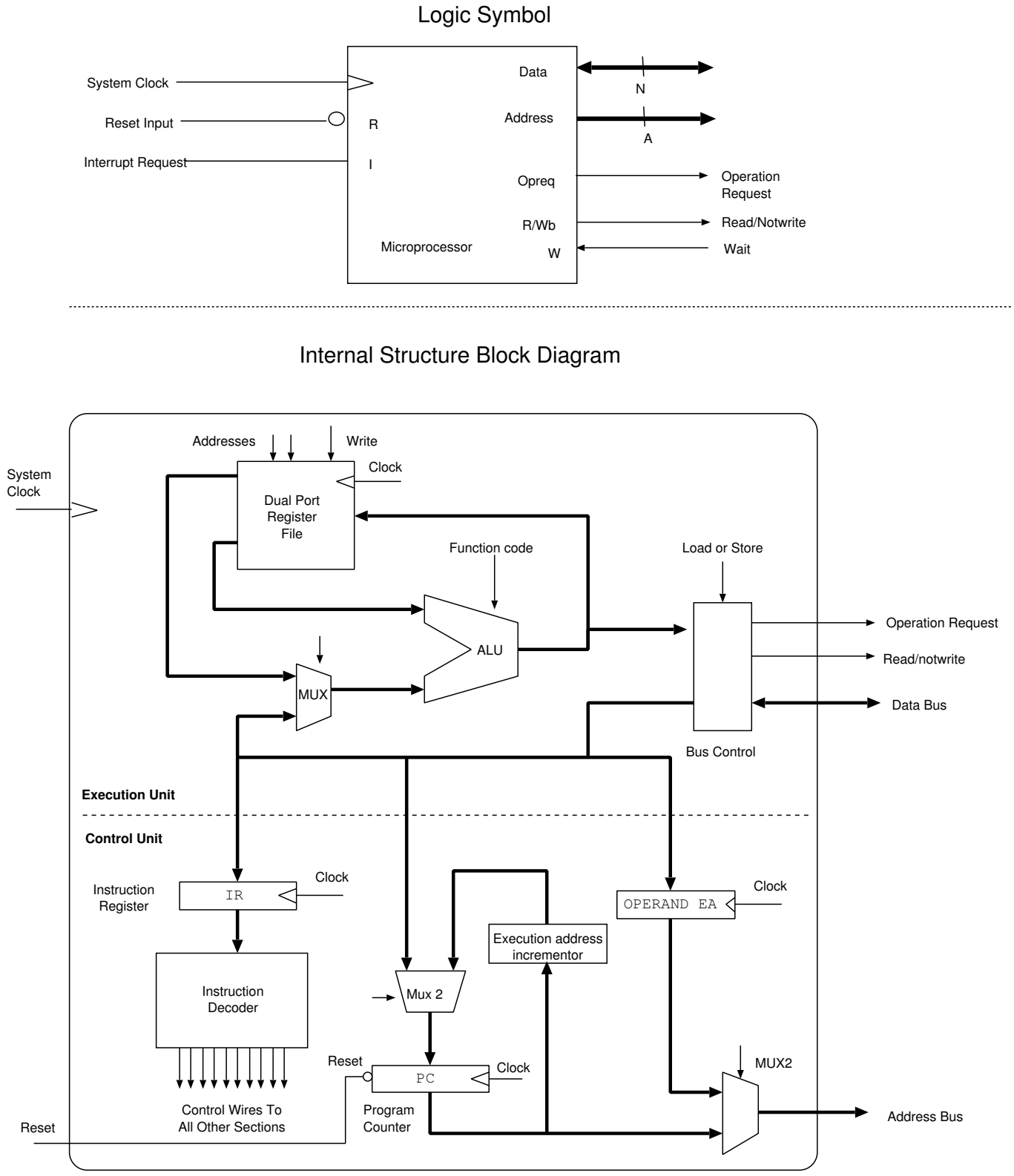



Figure 42: A microprocessor logic symbol and simplified internal structure.

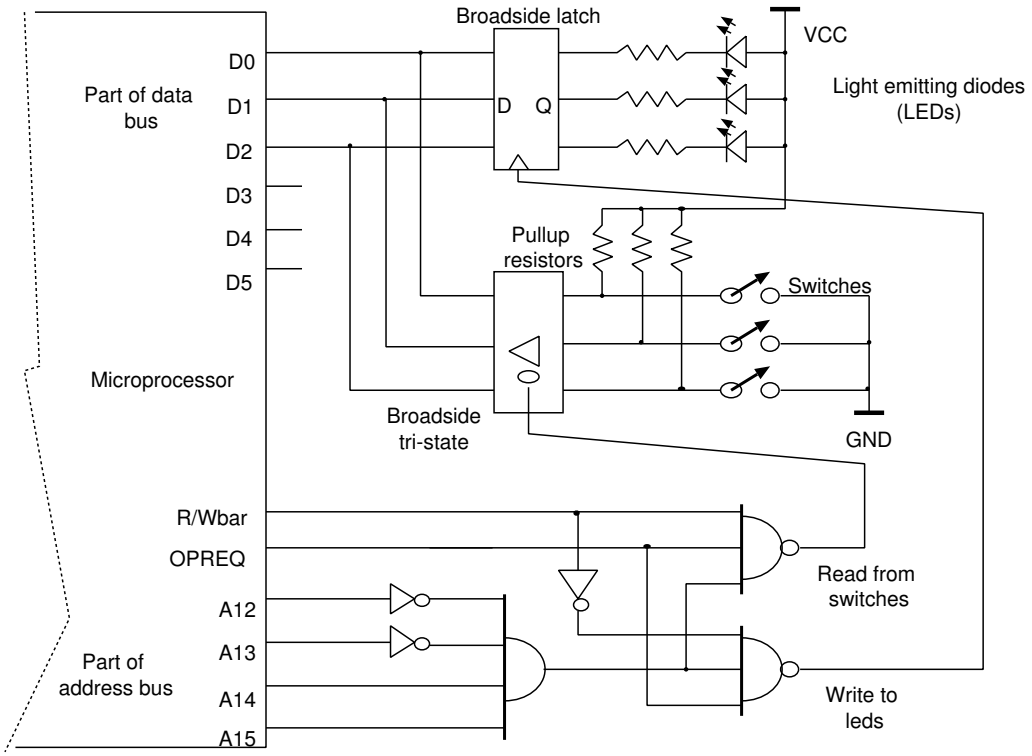



Figure 43: Example of memory address decode and simple LED and switch interfacing for programmed IO (PIO) to a microprocessor.

Figure 43 shows how to connect some LEDs and switches to a microprocessor for simple programmed input and output. Only the low order data bits are used, so when reading from the switches, random values may appear on the higher order bits. Conversely, when writing to the LED registers, the higher order bits are ignored.

## 2.4 Central Processor and Microprocessor

In the early days of computing, the CPU (central processor unit) was a large box containing the following

- Control Unit
- Execution Unit (Arithmetic Logic Unit plus register file)
- Main Memory

The breakthrough that was the invention of the *microprocessor* was to put the first two of these three on a single chip. (Today it is also possible to also put nearly a megabyte of main memory on the same chip, but this is frequently used as a cache.)

The microprocessor makes bus cycles. A bus cycle happens as follows:

- It places the address on the address bus and sets read/writebar low if this is to be a write cycle and high if it is a read cycle.
- If this is a write cycle, it drives the data to be written out to the data bus. If it is a read, it makes the data bus high-impedance ready to accept data.
- It asserts operation request (i.e. it puts it to logic 1).
- It waits half a clock cycle.

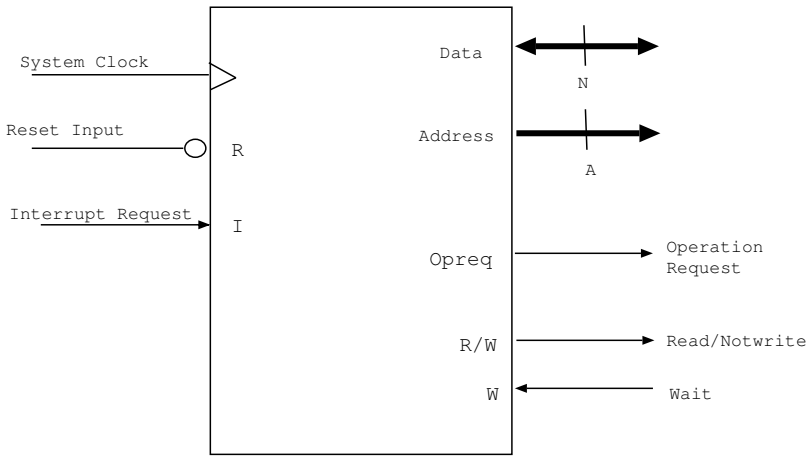



Figure 44: **Microprocessor Logic Symbol.**

- If the wait input is asserted, it waits whole clock cycles until wait is not de-asserted.
- If this is a read cycle, the processor copies whatever value has been driven onto the data bus to its internal destination
- It de-asserts operation request.

## 2.5 Memory Map

In a Von Neumann computer, the program and data share a single address space. The memory is a collection of binary digits (ones and zeros) or bits.

If there are $a$ address lines, then the address space contains $2^a$ locations. In a byte-addressed machine, the bits in the memory are grouped into eight bit bytes for addressing. Each byte has its own address and each bit may be addressed via a byte address and an offset (from 0 to 7) within a byte. Figure 46 is the circuit of small, microprocessor based computer. The microprocessor can address $2^{16}$ locations of eight bits (65536 bytes) and some of these are filled in with devices.

For example, the address space (memory map) might be organised as follows

```
-------  -----     ----------------------
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
-------  -----     ----------------------
```

The processor makes access to the various resources in the machine by generating addresses on the address bus. Address decoding logic matches patterns on the high-order address lines to generate enable signals for each particular device. This logic sets the memory map (or address map) of the computer. The low-order address line are fed to each device to distinguish access to individual locations within the device.

```
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
```

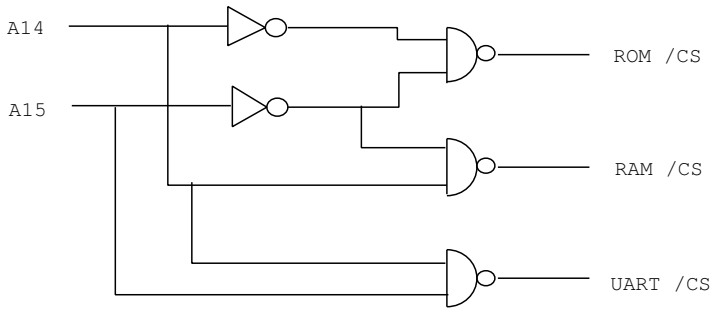



Figure 45: **Decoder logic to create the given memory map**

```
   assign uart_cs = !(abus == 2'b11);// 0xC000
endmodule
```

*Exercise:* If the address bus is 16 bits, how many addresses are consumed by the switches and LEDs in figure 43. If the data bus is 8 bits, what percentage of addressable bits have been wasted and is this a sensible design ?

*Exercise:* What would happen if the microprocessor tried to write to the read only memory ?

## 2.6 The PC as a component.

The PC motherboard has evolved in a backward-compatible way for 25 years. The mounting holes and connectors and architecture are largely unchanged, and the changes which are not back-compatible have been controlled and rare. Today, the PC motherboard offers exceptional value for money: 300 MHz processor, 32 Mbyte RAM and the logic for many useful interfaces (keyboard, USB, VGA, Floppy, IDE, IRDA, etc.) all for under 50 pounds. A typical PC motherboard is shown in figure 47. In addition, a huge variety of hardware devices is available to plug in.

Smaller, off-the-shelf, *euro-boards* containing a PC have dimensions 100 by 160 millimetres and can easily slot into a rack-oriented hardware design. Putting a PC on a custom chip is a possibility for the near future, especially if memory and CPU levels are relaxed by 3 to 4 years' worth of performance gain.

It is easy to think of a PC as something that always has a screen and a keyboard and runs Microsoft code. However, the PC motherboard is a general purpose computing platform that, given the correct software, is just as happy providing, for instance, the processor inside an airport arrivals VDU cluster. In this example, a video display board and a network board would have to be fitted.

## 2.7 Printed Circuit Boards

Nearly all designs require the fabrication of new printed circuit boards. A cheap PCB has just one layer of wiring, but is not suitable for most of today's highly-compact electronic devices. Most digital boards have 4 layers, with the internal pair just used for power. Some high-tech boards need more layers of wiring. See examples in lectures.

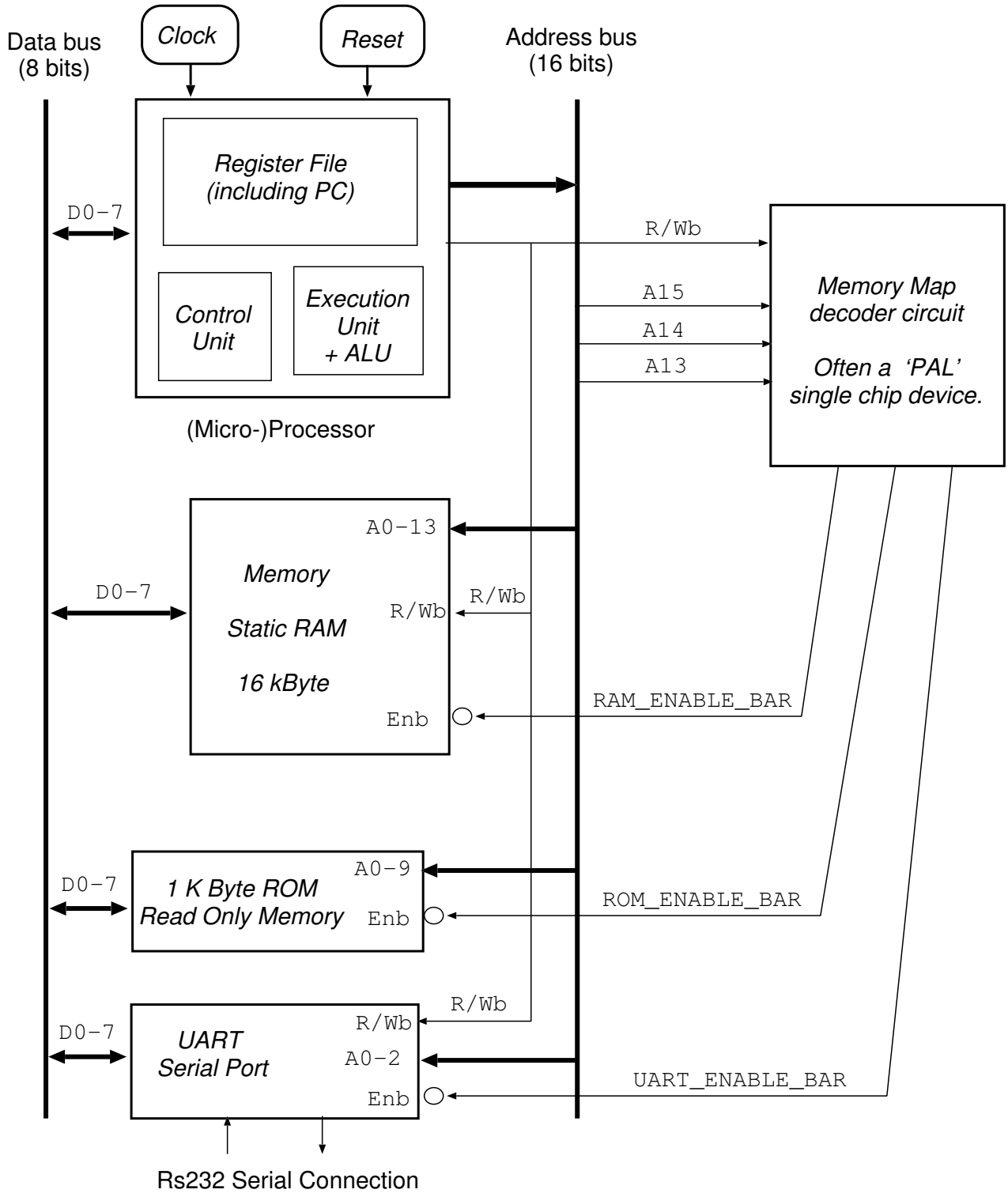



Figure 46: A small computer (A=16, D=8).

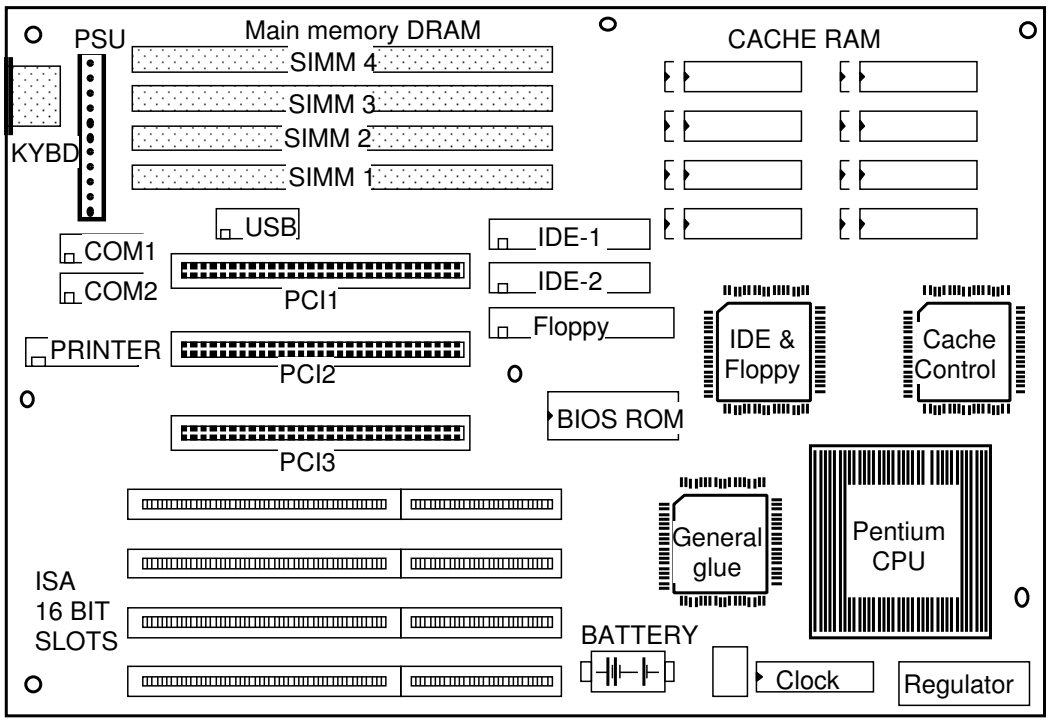



Figure 47: PC motherboard, 1997 vintage.

# 3 Finite State Machines, IO and Clock Domains.

*Learners' Guide:* What you should learn from this section is:

- Systems are made of interconnected modules and have more than one clock.
- Interfaces are standardised and have a long lifetime.
- When signals have to cross from one clock domain to another the design should accommodate that there will be skew and metastability.
- The maximum clock rate is normally set by the longest path of logic.

## 3.1 Connecting Boards Together.

When boards are interconnected, some of the main issues are:

- How much data in bits per second needs to flow?
- Will the connection be synchronous or asynchronous?
- Is flow-control needed to limit the rate of data flow?
- How long do the wires or cables have to reach?
- Is it desirable to carry power with the cables ?
- Is hot-pluggable mode needed (plug in/out with power on) ?
- Is the interconnection topology fixed at design time or does the system need to adapt to various configurations?
- Should we use an existing standard or design a new system ?

Major electronics issues when connecting boards together is whether the cables will give off interference and whether the electronics at the ends will be damaged when people with a large static charge touch the exposed pins.

**The bits stored in a data cable per wire is given by the signalling rate multiplied by the length divided by 200 m/$\mu$s = $0.66c$.**

### 3.1.1 Parallel Port

With the parallel port, data is transferred at a rate dependent on the slower of the sender and the receiver, since the transmitter is not allowed to strobe the next byte of data when the busy signal is asserted.

Modern parallel ports support bidirectional transfer of data, but originally they were just designed for printer connection.

The reach is potentially dominated by skew of the parallel signals, since they must all be valid at the receiver before the strobe signal. Flow control is provided by the busy signal.

*Exercise:* Figure 48 shows the signals involved in a parallel port, as frequently used to connect printers to computers. The basic operation is that the master places a byte of data on the data bus and then strobes the strobe signal low until acknowledge is asserted by the device. The master must not present a new byte to the printer while the printer has put busy high. Design the parallel port interface logic by drawing on the ideas shown in figure 43. To do this, it is sufficient if the busy signal appears to the processor like one of the switch inputs and the data output is rather like eight LEDs. Considering that the processor really does not care about the acknowledge signal or when the strobe signal is deasserted, you might care to use an RS latch to help the processor generate the strobe signal.

### 3.1.2 Serial Port

The serial port uses one pin for each direction, whereas the parallel port sends only (in its basic form) and uses 8 data pins to transfer a byte.

With the serial port, the receiving device must be operated at the same baud rate as the transmitter, to make sense of the data. With only one signal pin, there is no skew problem, but the maximum transition rate on the data wires is much faster. **The baud rate is the number of signalling events per second on the line.** In a binary two-level coding system, the baud rate is the raw bit rate. A serial port with a start and stop bit per byte achieves 80 percent of its raw bit rate as useful throughput.

Flow control is the matter of arranging that no more data is sent than can currently be accommodated at the receiver. Additional signals on the connector are sometimes used for flow control: a wire in each direction enables transmission in the other direction. Alternatively, flow control may be implemented using a software protocol where two of the possible byte codes are not used for data, but instead reserved for flow control. An end sends the Xoff code when it is running low on space to receive data, and sends an Xon when it has space again.

### 3.1.3 PS2 and Keyboard port

The original 1977 IBM PC introduced a connector that has become widely used for PC keyboards and mice. It is a point-to-point connection that carries power, although the total current available is limited by a fuse or similar protection device. The connection is bidirectional and byte-oriented. A pair of open drain lines (see figure 6) are used for clock and data. When one end wants to send data to the other, it pulls down the data line, forming a start bit, the other end then clocks the data line until a byte has been received.

**The PS/2 port is unsuitable for high bit rates or long cables since the cable length must not be longer than a clock wavelength.**

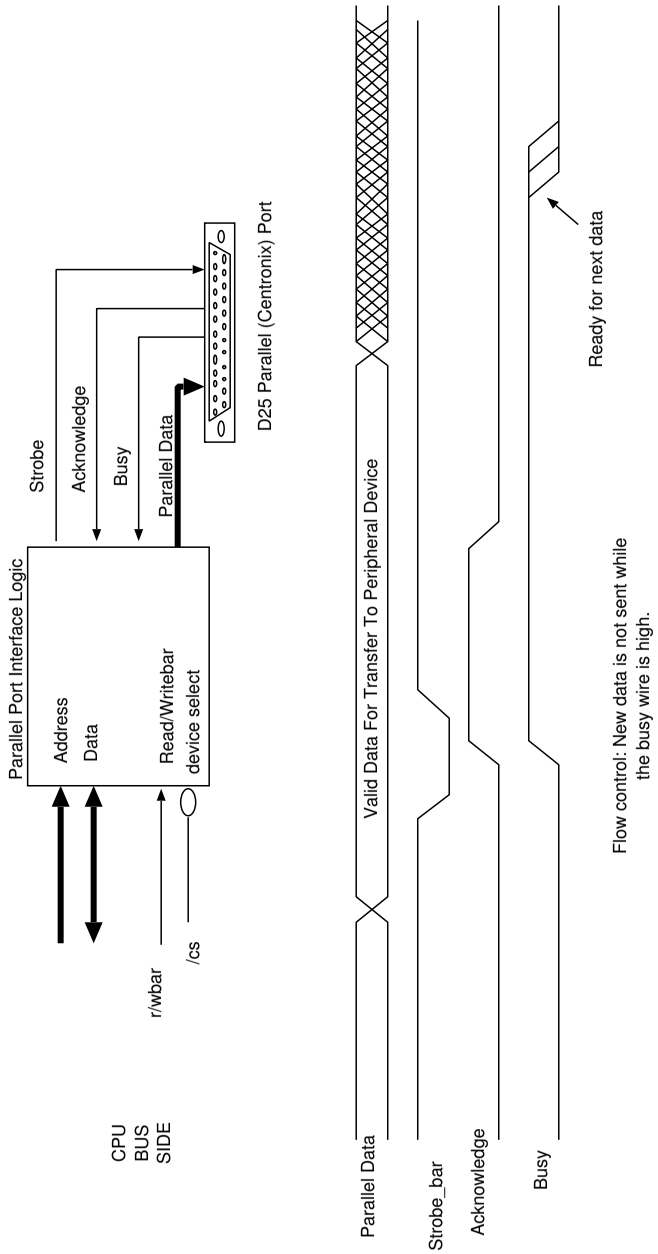



Figure 48: PC or Centronics Parallel Printer Port.

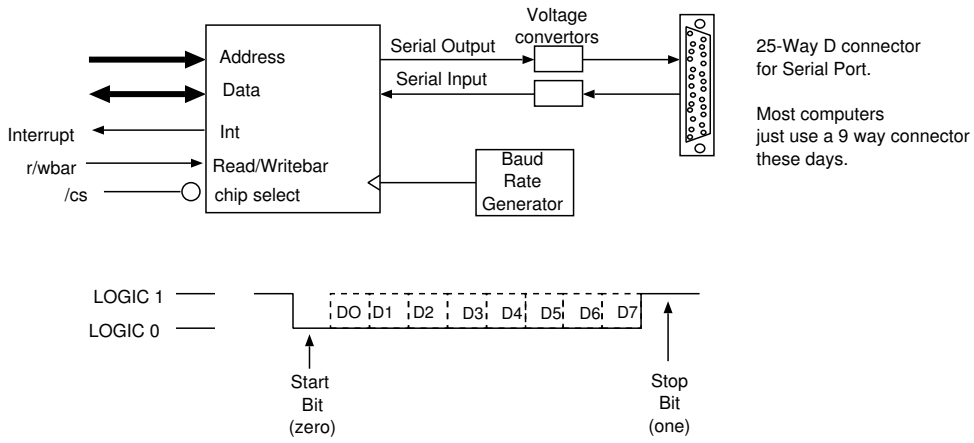



Figure 49: Serial port or UART arrangement

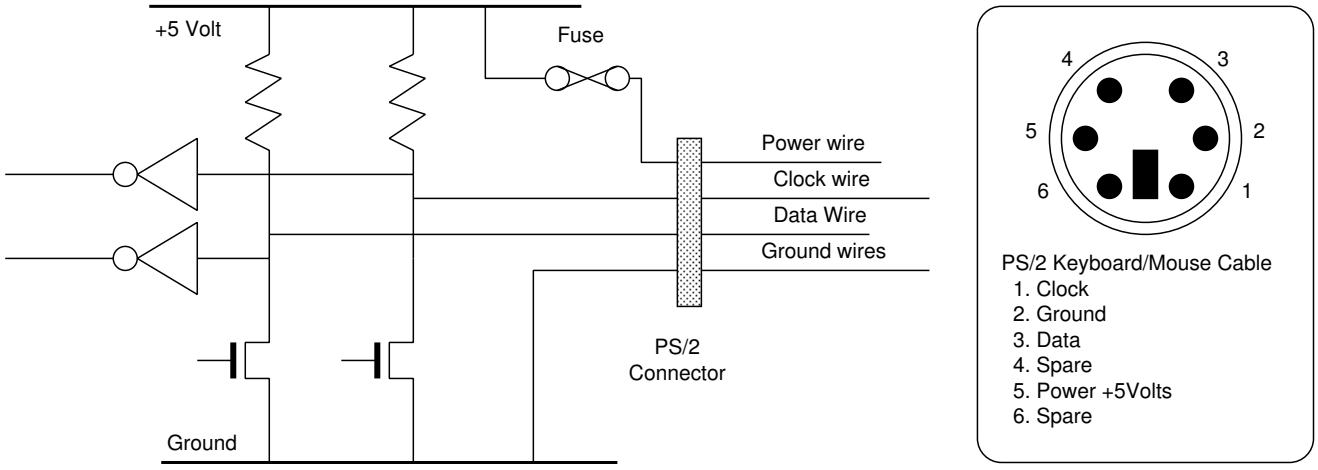



Figure 50: Keyboard and/or PS/2 port.

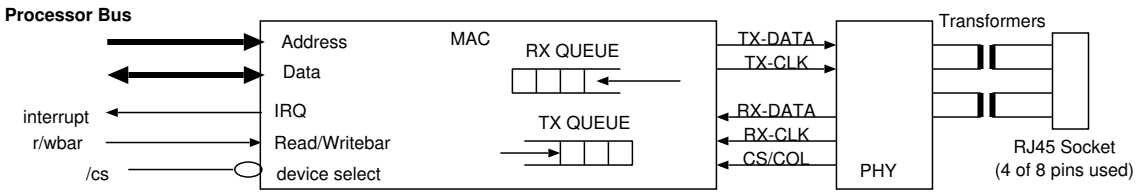



Figure 51: Ethernet or other LAN structure.

### 3.1.4 USB port

The Universal Serial Bus uses similar electrical technology to the PS/2 port, but the protocol for using the two wires is different. Small hubs, perhaps integrated into the back of a keyboard, allow a desk-top network to be made. Addressing and topology determination are supported and the data rate is 12 Mbps, or 1.5 for slow devices.

The Firewire port (P1394) found on many computers operates in roughly the same way as USB at the physical layer, using a pair of wires. However, it is a true LAN (local area network) in that any device may transmit a packet at any time, whereas in USB, one device acts as a master controller for the network.

### 3.1.5 Ethernet LAN Port

Figure 51 shows the basic structure of a LAN controller, such as Ethernet. The controller has small FIFO queues for packets just received or about to be sent on the network. The controller implements a protocol for when to send a packet - the media access control (MAC). The physical layer frequently used today is known as 100BaseT and uses two twisted-pairs, one for sending and one for receiving. The PHY component in the figure maps the digital data to and from the analog waveforms needed on the wire. It also detects collisions when two hosts start to send a packet at once.

For high performance, most LAN controllers implement DMA (direct memory access) whereby the address wires are temporarily reversed in direction and the controller generates addresses in the main system memory for loading and storing data. This is not shown in the figure.

## 3.2 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. These rules are not sufficient to act as a general procedure for logic minimisation. Indeed, the following example cannot be simplified in this way.

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

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

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

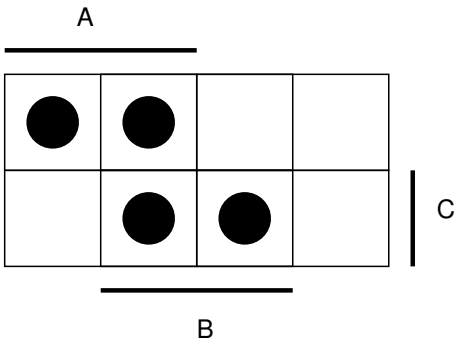



Figure 52: An example Karnaugh Map.

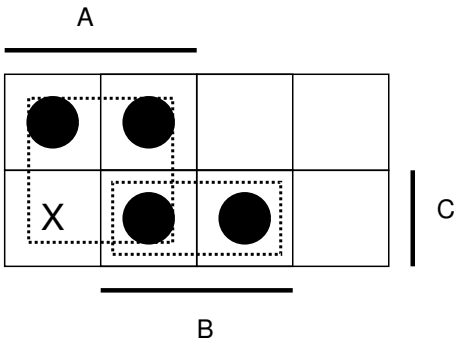



Figure 53: An example Karnaugh Map.

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

The 'ESPRESSO' algorithm is one of the best logic minimisers around, but it is still only a heuristic approach. There is no algorithm that can achieve as good results as an approach of looking at all possible expressions of a function and choosing the one that has the best metric.

The above discussion has related to one output only in two-level logic. By two-level we mean one AND array and then one OR array. Although any function can be expressed in sum-of-products form, a multi-level gate structure can achieve fewer gates. Where multiple outputs are required, implicants can be selected so that they are useful for multiple outputs.

The programmable logic array (PLA) uses the idea that any combinational logic function can be expressed in some-of-products form. During logic minimisation and fitting, one selects implicants that can be used in more than one sum, where possible.

## 3.3 Clocking and Synchronous FSM Combination.

This section looks at aspects of clock signals and considers the how, when and why of multiple clocks in a system. The decisions relating to clocks are fundamental to any hardware design and are normally taken very early in the process.

## 3.4 Finite State Machines.

Figure 55 shows a schematic symbol and canonical circuit for a finite-state machine. It is a mantra among mature hardware designers to consider everything as a collection of finite-state machines. Beginner hardware designers tend not to have this approach, and instead think nothing of treating the clock input to a flip-flop as suitable for wiring to anything convenient. They do not get far like

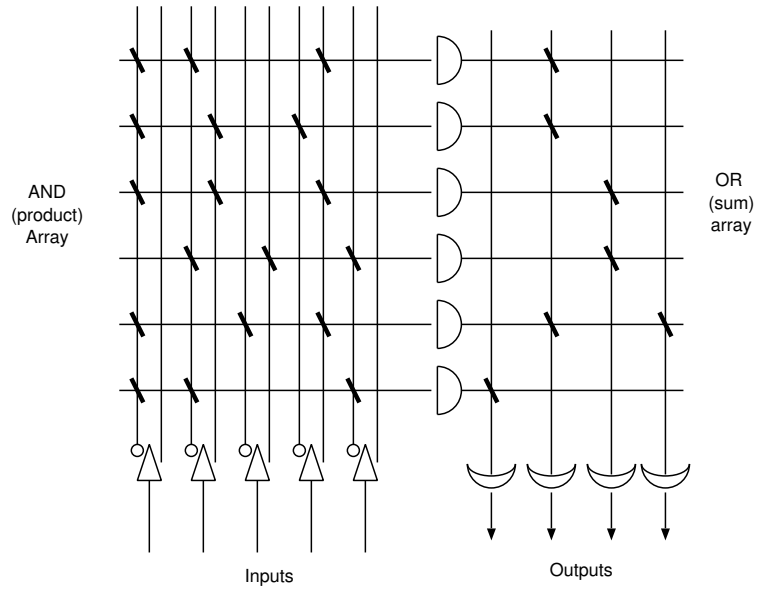



Figure 54: An example PLA.

this, since contemporary CAD tools, design techniques and formal methods will stop providing any help and circuit bugs will creep up everywhere. (An exception is that there are certain *licensed* asynchronous hardware designers, like Dr Simon Moore, who often don't even have clocks in their circuits.)

The FSM has inputs and a clock. It has two types of outputs. Moore outputs depend only on the current state. Mealy outputs may also depend on the current inputs.

Figure 56 shows a D-type flip-flop and its three significant timing specifications. The flip-flop input must not change at the same time as the positive edge of the clock: indeed it must be stable for a period before (the set-up time) and remain stable a period after (the hold time). The output will change after a propagation delay time from the positive edge of the clock. The negative edge of the clock has no effect. The J and K inputs of JKs and the T inputs of Ts have the same set and hold requirements.

In a synchronous FSM, the maximum clock frequency is normally determined by the longest path of logic which ends at the input of a flip-flop. This path is know as the *critical path* and is illustrated in figure 57.

## 3.5 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.
```

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.

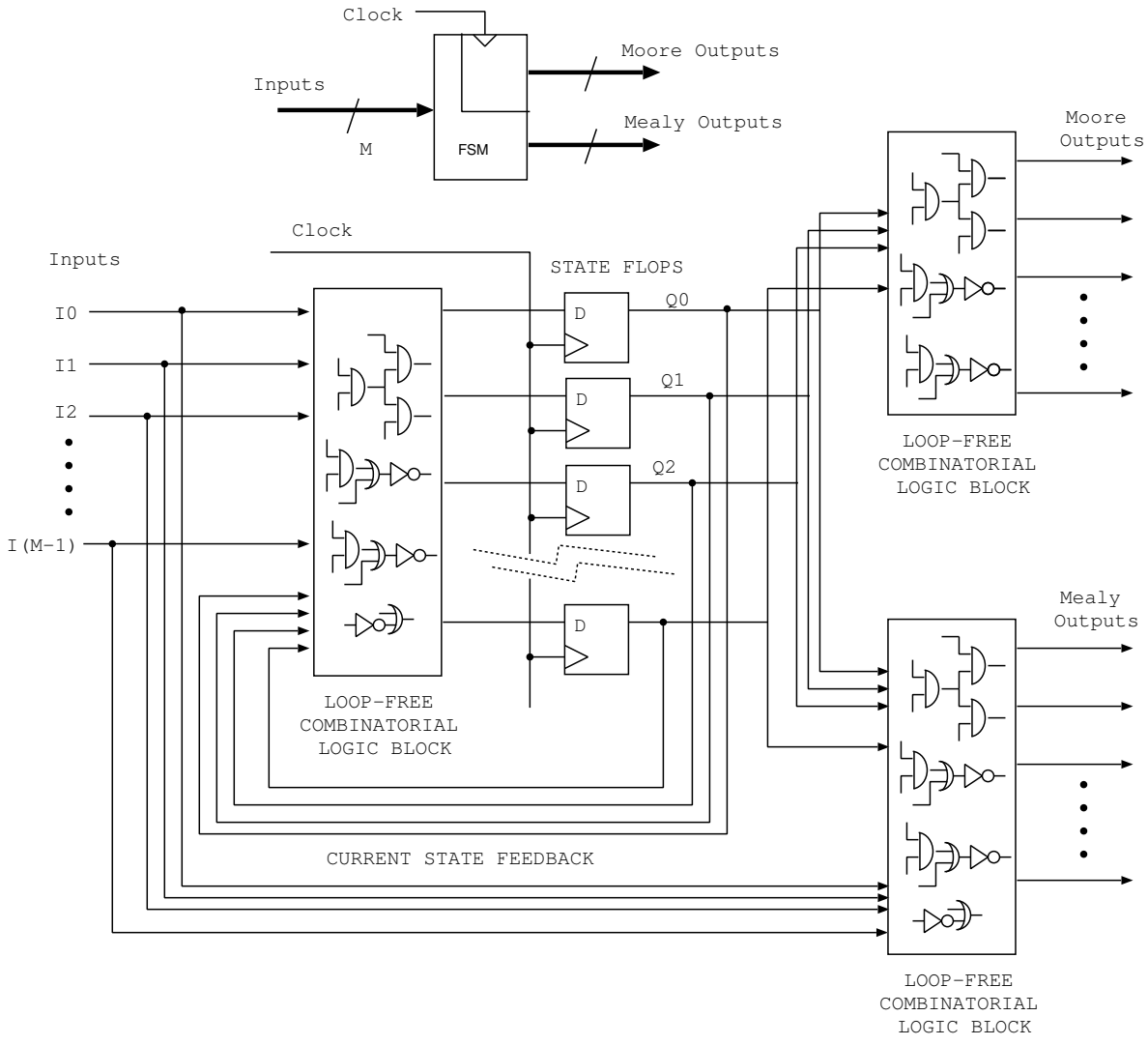



Figure 55: Canonical synchronous, finite-state machine.

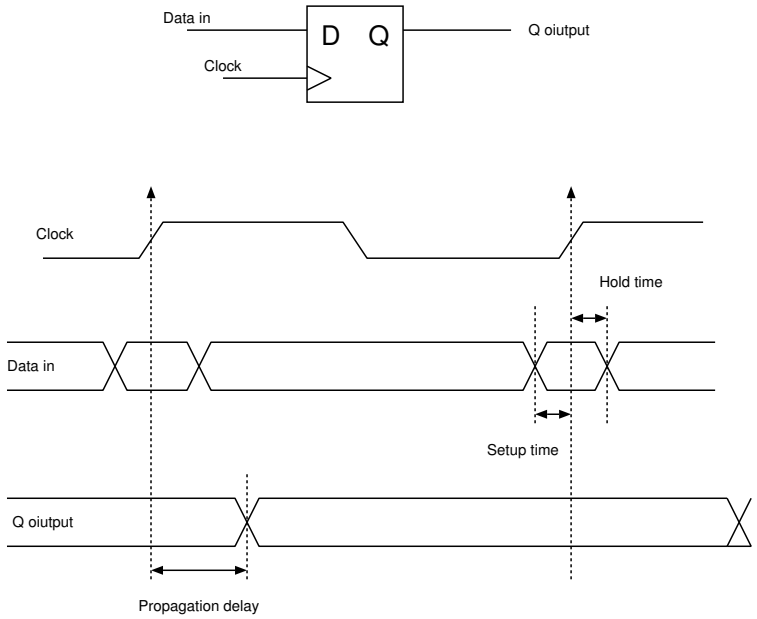



Figure 56: Timing specifications for an edge-triggered D-type flip-flop.

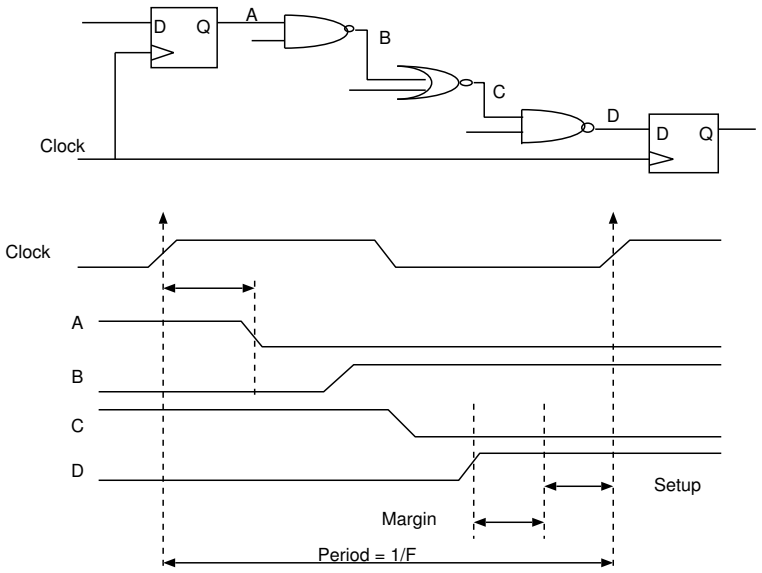



Figure 57: Typical nature of a critical path.

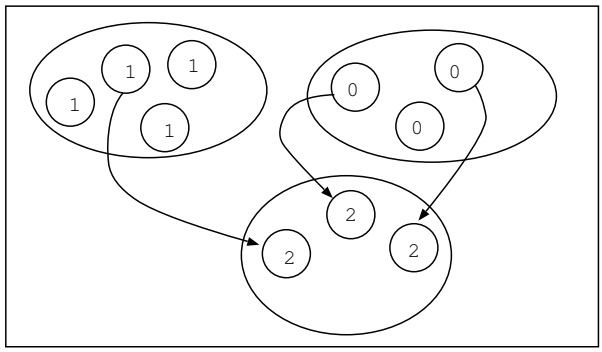



Figure 58: Bi-simulation Construction.

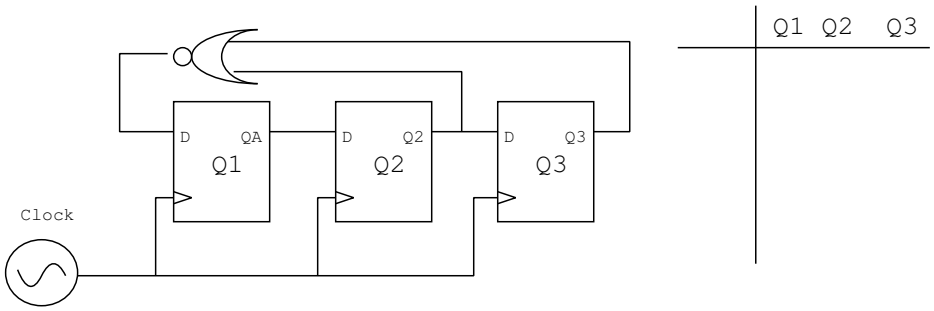



Figure 59: A counter that does not use binary.

A small modification of this procedure allows us to tell if whether a pair of finite state machines are equivalent in terms of observable external behaviour.

## 3.6 Speed Increase with Johnson Counter

A *Johnson* counter is based around a shift register. Figure 59 shows a Johnson counter that divides by five. The significant features of Johnson counters actually become clearer for larger counters, but they are twofold: 1. the division ratio is restricted to twice the number of flip-flops (e.g. 5 flip-flops can give a divide by 10 at maximum) and 2. the number of logic gates is always low, and so the counter is fast.

*Exercise:* Design a Johnson Counter that also has a control input that makes it divide by six or seven. Optimise your design for two features: 1. maximum clock speed, 2. that the ratio input can be changed at any time without causing a hazard. Note: clearly, metastability cannot be avoided, but by being hazard free, we mean that the state path taken will always be one or the other of the two desired paths, whatever happens.

## 3.7 Speed Increase with One Hot Coding

In *One Hot* coding, at all times, the flip-flops in a counter or FSM are arranged to be all zero except for one that is one. This clearly minimises the number of gates required to decode the current state and can greatly reduce the number of gates that determine the next state.

## 3.8 Speed Increase using Pipelineing

Figure 60 shows a circuit before and after the addition of a pipelining stage. The pipeline stage may have a fewer or greater number of flops as the input or output. The benefit of the pipeline stage is that a complex combinatorial logic function is split into two halves, thereby reducing the maximum length of unclocked logic leading to the inputs of the output flip-flops. The disadvantage of the pipeline stage is that data takes longer to be processed. This disadvantage is not often a problem: for instance, if the data has just come down a link in Gbps fibre-optic link, then the additional delay will be just as if the link were a few meters longer. More than one stage of pipelining can be inserted if needed.

Some modern logic synthesiser tools will insert pipeline delays where possible, if helpful. Others will back-propagate them to balance delays when they are manually instantiated at the outputs.

*Exercise* Sketch the circuit of a large adder with ripple carry and then modify it to include a pipeline stage. Note, the output signals should always reflect the result of a single addition in any one clock cycle. Explain or estimate the effect of the pipeline delay on the maximum clock frequency.

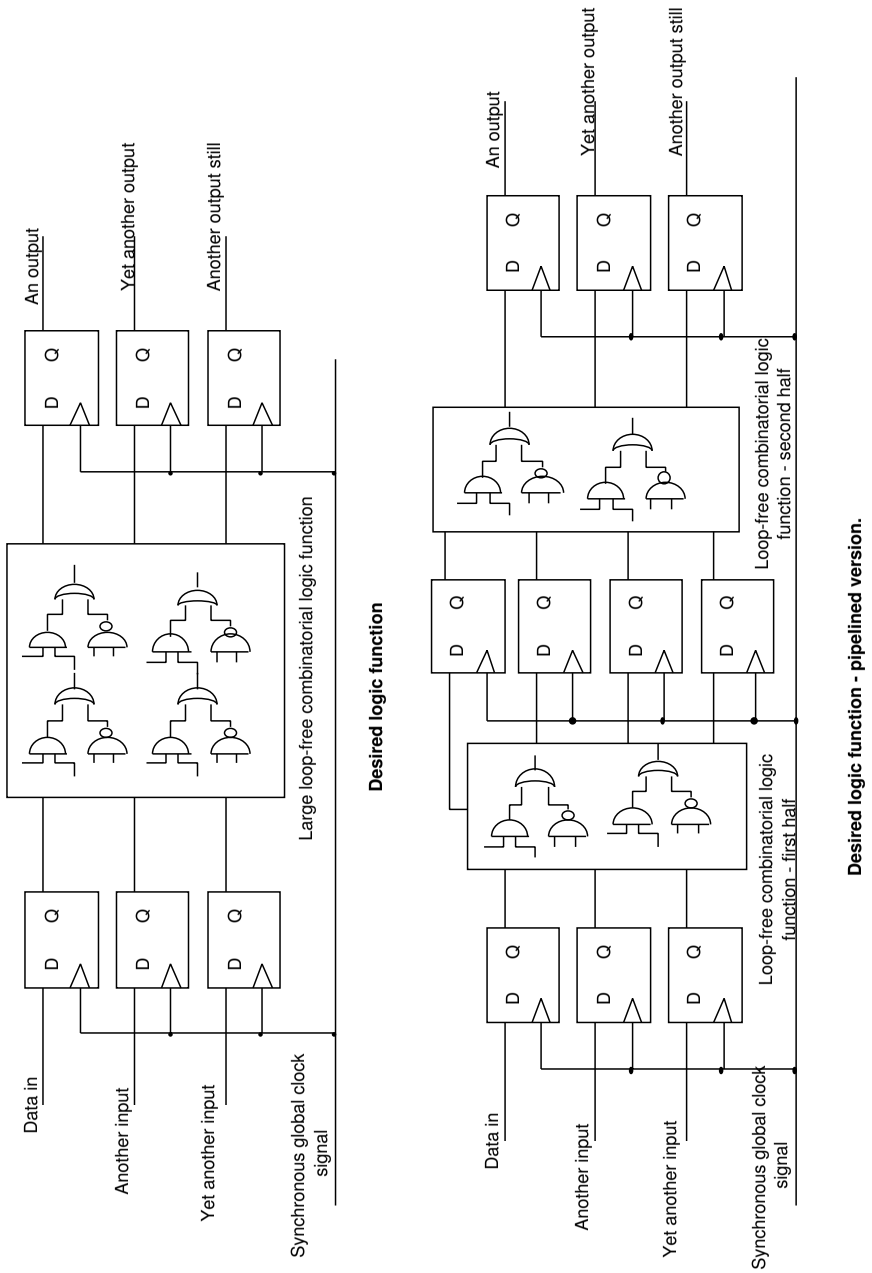



Figure 60: Example of introducing pipelining to increase throughput, but also delay.

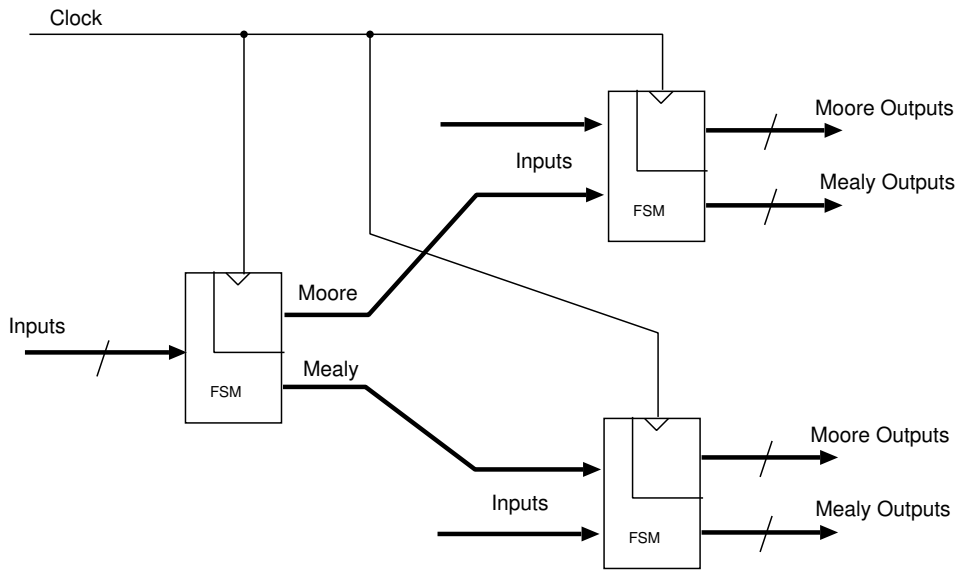



Figure 61: Cascading FSMs to produce larger circuits.

## 3.9 Derived Clocks.

Figure 61 shows a concatenation of finite-state machines running from the same clock. Of course, viewed another way, these together simply form one larger FSM. However, the decomposed view is vital in practice, since the density of wiring between FSMs will be lower than within the FSM and the functionality, authorship, history and even ownership of the IPR (intellectual property rights) of the FSMs will be different.

If we feed a Mealy output into another FSM, delays accumulate, reducing maximum clock frequency. This does not happen with Moore outputs. Moore outputs should be used if possible, but for some designs, a Mealy output is needed in order to reduce processing delay of the unit as a whole.

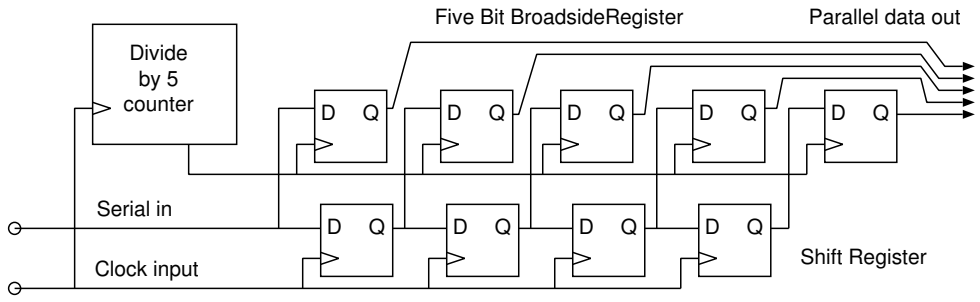



Figure 62: An example that uses (badly) a derived clock: a serial-to-parallel convertor.

```
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;
```

Figure 62 shows a typical example of the use of a derived clock. Power consumption is increased in parts of a circuit where fast clocks are used, and so using lower frequency clocks where needed is an important design consideration.

The serial-to-parallel convertor can be redrawn as a pair of FSMs as shown in general in figure 66. It is then clear that the design rule about two Moore outputs from the master not changing at once cannot be applied (see section 3.13).

*Exercise:* Give the schematic for a reciprocal parallel-to-serial convertor. Does combining the two converters result in the identity function ? If not, why not. Note: they can be combined in both orders.

## 3.10 Gated and guarded clocks

Sometimes we do not wish a flip-flop to change its value every clock cycle. Clearly, J-K device support this directly: we can put both the J and K inputs low. For subsystem FSMs or broadside D-type registers, we need the concept of the clock enable.

Figure 14 showed a D-type with a clock enable input and the equivalent circuit. Gating clocks is avoided as a general design practice but clocks do have to be gated at times though, most commonly in powering down complete sections of a system. They are also needed when generating a write strobe to an SRAM (figure 24) or other level-sensitive register.

Figure 63 shows the safe way to gate a clock. The clock is OR-ed with the negated enable signal. In this clock gating method, the derived clock is kept normally high, except when a clock pulse is to be let through. Other arrangements for gating clocks tend to produce glitches in the derived clock.

*Exercise:* Design the logic to place in front of a J-K flip flop that turns it into a D-type with clock enable input.

Given that from Digital Electronics you know that a J-K can be equally as simple as a D-type to implement from transistors or gates, it is reasonable to assume that a clock enable D-type is also as simple. Therefore why should we ever use gated clocks instead of clock enable inputs ? The answer is that if we have a large number of flip-flops that are all to share the same clock-enable expression, we would have two penalties of using clock enable: 1. we have to route the clock enable signal to all of them, and 2. we have to make the flip-flops from logic which is capable of being clocked at every opportunity. We use clock-enable inputs where single or smaller numbers of flip-flops are to

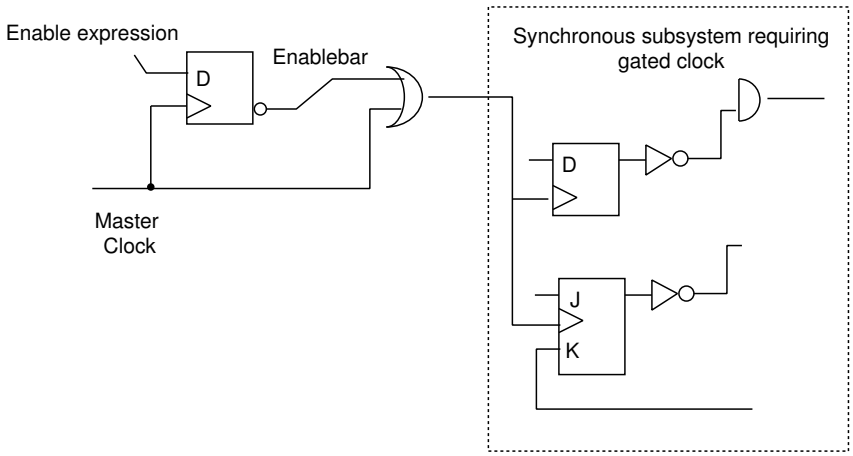



Figure 63: A 'safe' alternative to using clock enables ?

be enabled in unison, such as in a broadside register with clock enable. For single flip-flops, the clock enable is best considered as just one aspect of the FSM that the flip-flop is part of.

## 3.11 Clock and Data Skews

Skew is when a signal that is intended to arrive at several places at the same time actually arrives at various times.

Figure 64 shows a shift register circuit with clock skew. Skew is the enemy of synchronous, finite-state machines, since in the worst scenario, the output of one flip-flop may have changed, causing a change in the input to another flip-flop, before that second flip-flop has been clocked itself: this destroys the FSM behaviour.

If there is considerable delay in both the data and the clock, and these delays are random, then the shift register is not likely to work well. An example of such a system is where the shift register is actually a communications medium snaking around a number of circuit boards in a system. The delays are in the interconnecting cables. Typically, the logic at each station is not simply a D-type, but is logic which can read and change the data as it passes and the data may be a bus instead of a single wire. In this case, **for low-speeds, the solution is to invert the clock at each hop.**

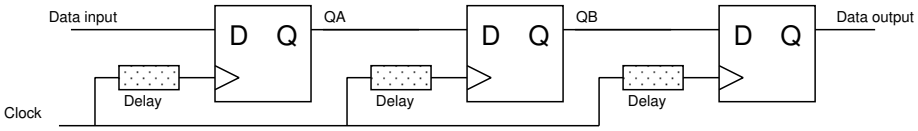
*Exercise:* Draw a timing diagram for the inverting-the-clock solution and work out the maximum frequency of operation. Use the following figures: conductor delay random between 4 and 9 nanoseconds, flip-flop set-up time 2 nanoseconds, hold time 1 nanoseconds and propagation time 5 nanoseconds. Note: in practice you would not run such a system close to the maximum you have just calculated.

## 3.12 Crossing Asynchronous Boundaries

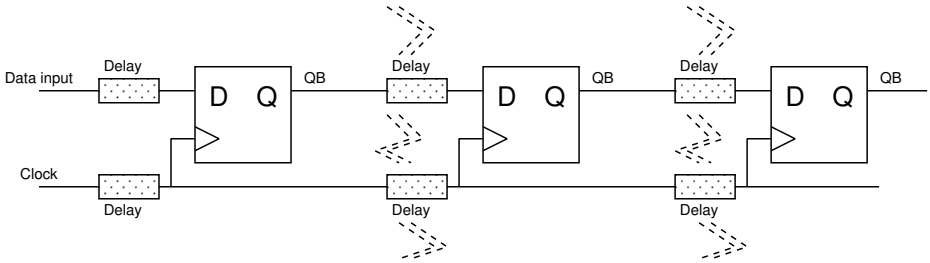
Figure 65 shows two clock domains with signals crossing between them. There may be signals in the reverse direction between the clock domains, but these would not demonstrate anything further that is significant.

The first thing to note is that for a single signal between the domains, there is always the possibility of metastability and failure of the whole system. However, with careful flip-flop design, a designer can reduce this to less than the probability of winning the lottery every week for a year: so this is not a real concern. Since there is more than one clock in the world, we do have to build these circuits in reality.

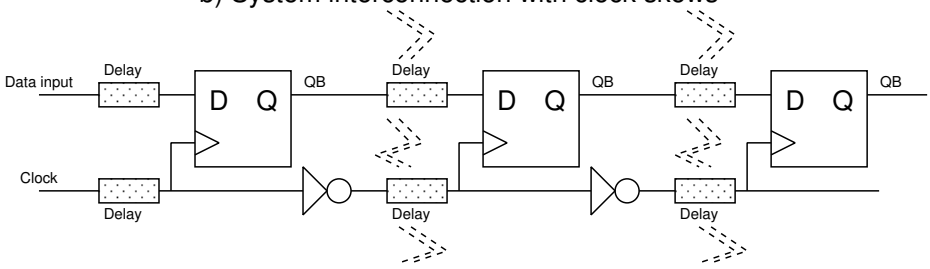
The main thing to note is that there will be skew between the lines and in the characteristics of the individual flip-flops. Therefore it is impossible to be sure of capturing a consistent view of them at the receiver if they are all changed at once or changed freely by the source. Therefore a protocol is required. A suitable protocol is to denote one of the signals as a *guard* or *qualifier*. Only if the




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 ?

Figure 64: Synchronous systems, but with clock skew.

guard has its active value is it implied that the data on the remaining signals is valid. The source must execute or implement a protocol that simply delays the assertion of the guard with respect to the other wires such that the other wires are sure to be seen as settled if the guard is seen to hold. The parallel printer port uses this approach (figure 48).

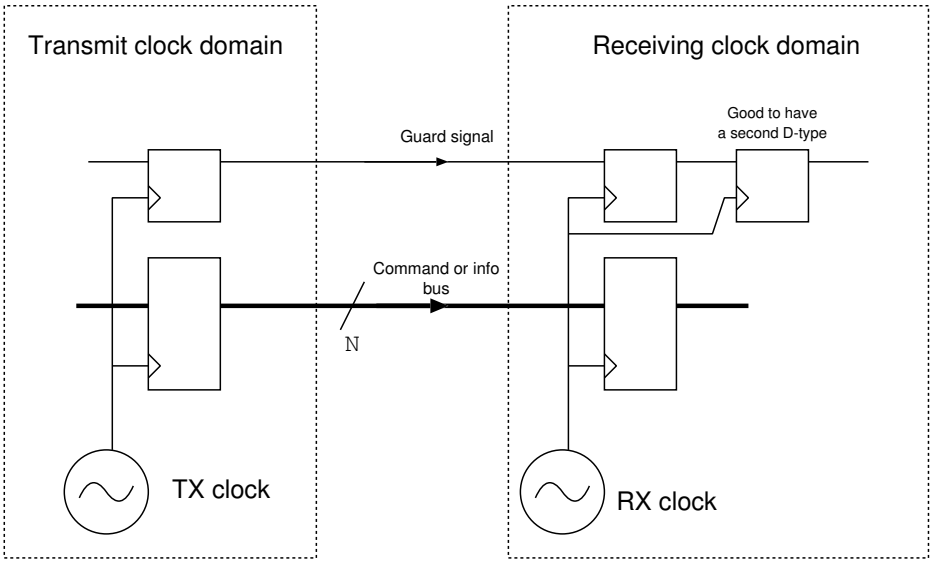
A good way to help avoid metastability is to use two flip-flops in tandem (as shown for the guard). Even if the first oscillates a while after being violated, it is most unlikely that it is still oscillating a clock cycle later when transferred to the second flip-flop. The additional delay introduced by the second flip-flop will result in the guard being presented to the receiver logic later than the signal lines, thus ensuring that they are stable.

*Exercise:* Is it possible to use changes of state of a guard signal, rather than its level, to indicate that the data on the remaining wires is valid? If so, sketch an example circuit and explain the benefits. What assumptions regarding the relative clock frequencies in use have you made, if any ?

## 3.13 FSM clocks derived from another FSM

Figure 66 shows a FSM whose outputs are used to clock another FSM. Examples of this are common, as in the serial to parallel convertor shown in figure 62 and the CFR two-chip decomposition shown in section 5.4. Such structures must be used carefully. The Mealy output clocking a slave FSM is probably a bad idea. The Moore output used to clock a slave is normally fine, but the logic function generating it must be free from hazards, otherwise the clock will glitch. A function of one state variable is best, and the state assignment should have been considered with that in mind.

The non-clock signals between the master and slave FSM also need attention. The Moore output from the slave to the master will not cause hazards, but will tend to restrict the maximum frequency of safe clocking for the system as a whole. The Moore output from the master to the slave needs careful attention to hazards, since it must never change at the same time as the slave clock signal.




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.

Figure 65: Use of a guard to cross an asynchronous boundary

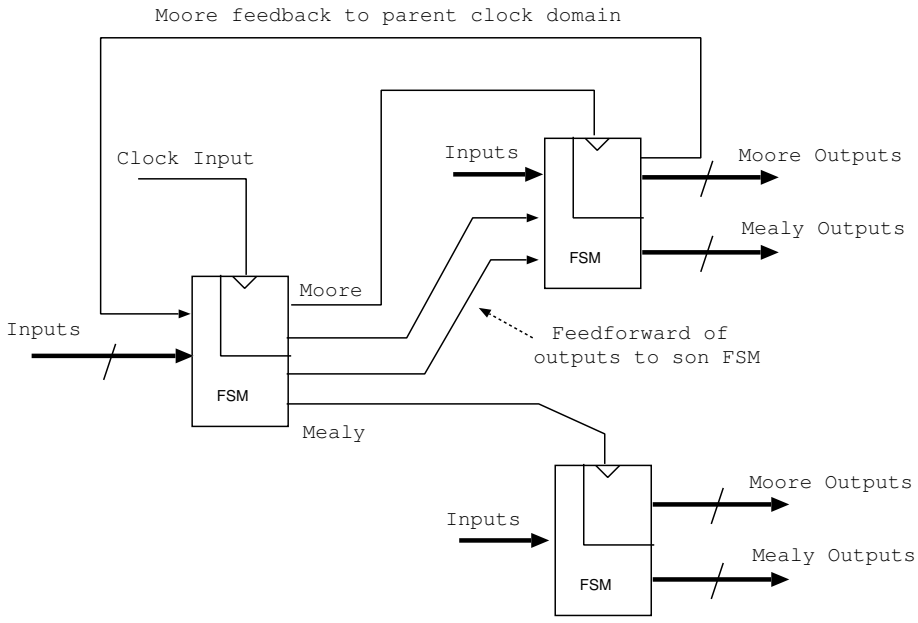



Figure 66: Example paths between FSMs with derived clocks

*Exercise (long):* Consider an FSM or system which generates a set of output signals that have to be carried via an intermediate subsystem which uses a separate (asynchronous), faster clock, and then delivered to a receiving system that must be clocked with a transported version of the original clock. An example of this would be where a company has fire alarm systems on two sites which are interconnected on a high speed data link owned by a telephone company, where the link is also carrying many other services. Sketch the basics of the circuits involved.

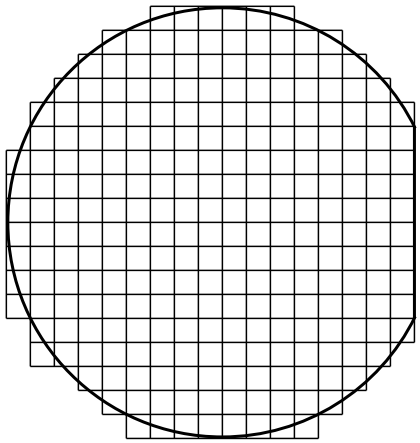


Figure 67: A wafer outline showing where it will be diced (Chips are not always square).

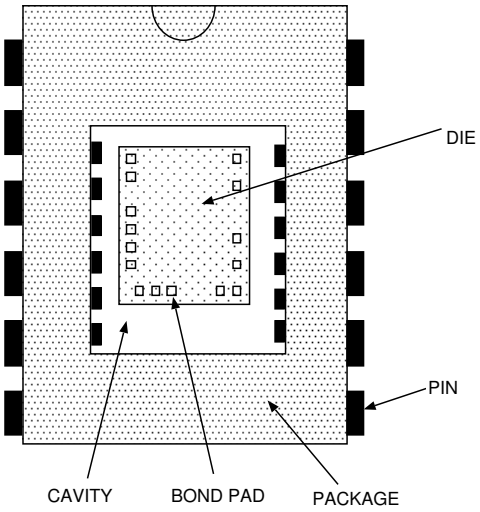



Figure 68: A chip in its package, ready for bond wires.

# 4 Technology, Speed, Power and Delay

*Learners' Guide:* What you should learn from this section is:

- Most designs for volume manufacture require new chips to be made.
- Field-programmable devices are used when volume does not dictate new chips or for prototypes.
- Systems are getting faster for a given power consumption, mainly due to shrinking physical feature size.
- It is sensible to implement systems with chips up to 1 centimetre on a side.
- Because of shrinking, modules which were previously complete chips now occupy just part of a new chip.

The cost of a system depends on the cost of its parts. There are many ways to design a system and typically we want to minimise cost. Therefore, before you can design a system, you need to be familiar with a vocabulary of technologies and pre-existing parts. You also need to know how to produce new integrated circuits for your application design. Application-specific integrated circuits

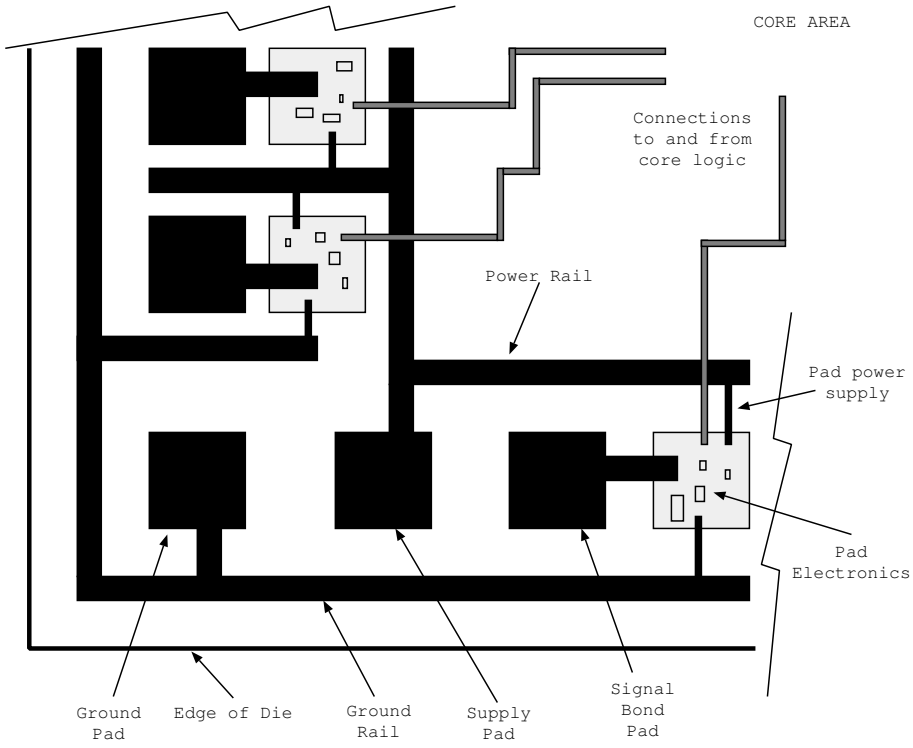



Figure 69: The corner of a typical chip showing IO and power pads.

(ASICs) and make up a large fraction of the IC market by revenue.

## 4.1 Basics of ICs.

*Note: this section is a briefest of introductions to VLSI.*

Chips are made of Si or GaAs with layers of Al on top for signal wiring. Chips are laid out as a rectangular grid on a wafer of the substrate material. Wafers are about 0.3 mm thick and from 3 to 12 inches in diameter. Most of the wafer depth is only present for handling reasons and the active portion of the chip lies in the top few microns of the wafer or is built up above the top surface by deposition.

Wafers are processed using ion implantation and photo-resistive etching at elevated temperatures in the presence of dopant and etchant gasses. Each photo-resist is first exposed to an ultraviolet or X-ray source under a mask generated by the CAD tools. Between 7 and 25 masks might be used, depending on the technology. The masks are stepped over the wafer to form the rectangular array of chips. Masked areas are left with an intact layer of resist and then remain unaffected by the next processing step, hence successive masks build up complex three dimensional structures forming the active components and conductors. Typical chips are today made using 0.18 to 0.3 micron feature sizes. The smaller a transistor the faster it operates.

A wafer contains many chips which are diced after wafer testing. The chip is then packaged to ease handling and to dissipate heat.

A typical chip is 4 to 100 square millimetres.

Wires are connected to chips at bond pads. Bond pads are about 100 microns square and 150 microns apart or half this size (linear dimension) in modern devices. The corner of a typical device is shown in Figure 69. Most chips have the shown power and ground 'rings' around the outside with the bond pads in between. These rings provide low resistance power supplies to the pad logic. The pad logic is buffer circuitry to provide high-power drive to off-chip loads and to protect the internal circuitry from external static discharges.

Figure 68 shows the chip carrier cavity containing a die. Connections are made from the conductors in the package to the bond pads around the chip's perimeter using cold-welded gold wires.

## 4.2 Will an ASIC be core or pad bound ?

A chip may be either:

- **Pad bound** if it has many more pins than natural perimeter, or
- **Core bound** if it has sufficient active core area to dominate.

A pad bound chip is wasteful of area since we have to use a bigger die to get the pads on, but the body of the die is not fully occupied with gates.

Generally **we end up with separate fabrication processes for each of the following subsystems:**

- memory,
- random logic,
- analogue circuits and analogue to/from digital conversion,
- high speed (e.g. clock rate greater than about 1000 MHz).

Also, we must group together functions which can be implemented in the same technology, so they can most easily be put on one chip. This is hard if the functions do not otherwise have much to do with each other: we might prefer to put the two sections of logic further apart.

## 4.3 Chip cost versus area

**The cost of a chip divides into two parts: NRE and per-device cost.**

NRE — non-recurring expenditure. NRE is paid once, regardless of how many devices are made. It consists of:

- design cost (labour, computer time, management overheads),
- mask making cost.

Per device — recurring expenditure consists of:

- cost of basic silicon wafer,
- cost of processing a wafer,
- cost of testing devices,
- cost of device packages and packaging,
- cost of further testing, possibly under stressful conditions.

The per device cost is influenced by the yield — the fraction of working dice.

The fraction of wafers where at least some of the die work is the 'wafer yield' and is typically close to 100 percent for a mature fabrication process.

The fraction of die which work on a wafer (often simply the 'yield') depends on wafer impurity density and device size. Die yield goes down with chip area.

The fraction of devices which pass wafer probe (i.e. before the wafer is diced) and fail post packaging tests is very low. However, full testing of analog sections or other lengthy operations are typically skipped at the wafer probe stage.

| 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 |

Table 1: Output from the die cost program.

## 4.4 Example of CMOS die cost

A six inch diameter wafer has area $(3.14r^2) = 18000$ mm$^2$.

A chip has area $A$, which can be anything between 2 to 200 mm$^2$ (including scoring lines).

Dies per wafer is $18000/A$

Processed wafer cost might be 5000 pounds. (Actually it's probably about one third this cost to the silicon foundry, but there are overheads and profit to add on.)

Probability of working = wafer yield $\times$ die yield (assume wafer yield is 1.0 or else included in the wafer cost).

Assume 99.5 percent of square millimetres are defect free. Die yield is then

$$P(\text{All A squares work}) = 0.995^A$$

cost of working dice is

$$\frac{5000}{\frac{18000}{A}0.995^A} \text{ pounds each.}$$

Here is a program to estimate the cost of a working die given a six inch wafer with a processing cost of 5000 pounds and a probability of a square millimetre being defect free of 99.55 percent. Its output is shown in table 1.

```
#include <math.h>
display(double area)
{
   double dies = 18000.0 / area;
   double yield = dies * pow(0.995,  area);
   double cost = 5000.0 / yield;
   printf("%.0f  %.0f %.0f %.2f\n", area, dies, yield, cost);
}
int main()
 {
    double area = 2.0; /* area in square mm */
    while (area < 500.0)
     {
       display(area);
       area = trunc(area * 1.5);
     }
  }
```

Large dies sometimes use redundant design such that a small number of faults can be tolerated.

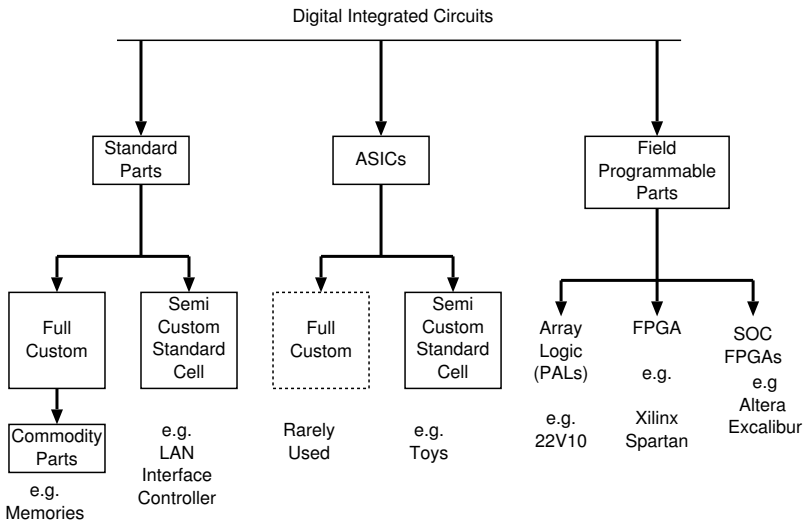



Figure 70: A taxonomy of integrated circuits.

Automatic circuitry or else laser or ion beam physical trimming is used to patch out faulty sections (of say a RAM array) and substitute previously redundant sections.

## 4.5 Classes of Integrated Circuits

Figure 70 presents a taxonomy of chip design approaches. The top-level division is between standard parts, ASICs and field-programmable parts. **Where a standard part is not suitable the choice between full-custom and semi-custom and field-programmable approaches has to be made, depending on performance and cost requirements.**

### 4.5.1 Standard Parts

A *standard part* is essentially any chip that a chip manufacturer is prepared to sell to someone else along with a datasheet and EDA models. The design may actually previously have been an ASIC for a specific customer that is now on general release. However, most standard parts are general purpose logic, memory and microprocessor devices. These are normally full custom designs designed in-house by the chip manufacturer to make the most of in-house fabrication line, perhaps using optimisations not made available to others who use the line as a foundry. Other standard parts include graphics controllers, LAN controllers, bus interface devices, and miscellaneous useful chips.

### 4.5.2 Masked ASICs.

A masked ASIC (application specific integrated circuit) is a device manufactured for a customer involving a set of masks where at least some of the masks are used only for that device. These devices include full-custom and semi-custom ASICs.

A full-custom chip (or part of a chip) has had detailed manual design effort expended on its circuits and the position of each transistor and section of interconnect. This allows an optimum of speed and density and power consumption.

**Full-custom design is used for devices which will be produced in very large quantities:** e.g. millions of parts where the design cost is justified. Full-custom design is also used when required for performance reasons. Microprocessors, memories and digital signal processing devices are primary users of full-custom design.

In semi-custom design, each cell has a fixed design and is repeated each time it is used, both within

a chip and across many devices which have used the library. This simplifies design, but drive power of the cell is not optimised for each instance.

**Semi-custom is achieved using a library of logic cells and is used for general-purpose VLSI design.** Figure 80 shows a cell from the data book for a standard cell library. This device has twice the 'normal' drive power, which indicates one of the compromises implicit in standard cell over full-custom, which is that the size (driving power) of transistors used in a cell is not tuned on a per-instance basis.

There are two types of semi-custom devices:

- standard cell (for high volume)
- gate array (for volume $<$ 5K devices).

In standard cell designs, cells from the library can freely be placed anywhere on the device and the number of IO pads and the size of the die can be freely chosen. Clearly this requires that all of the masks used for a device are unique to that device and cannot be used again. (Mask making is one of the largest costs in chip design).

In gate array designs, the silicon vendor offers a range of chip sizes. Each size of chip has a fixed layout and the location of each transistor, resistor and IO pad is common to every design that uses that size. A gate array is configured for a particular design by wiring up the transistors (and other components) in the desired way. Many transistors will be unused. The wiring up is, as usual, done with the top two or three layers of metal wiring. Therefore only two or three masks need be made to make a new design.

Silicon vendors will typically have stocks of wafers which have had the bottom 15 to 20 process steps made and are only awaiting final metallisation to be turned into usable devices. Hence design time is low. Risk is also low since the finished chip has only a small NRE cost and gate array technology is intrinsically conservative and hence reliable.

The disadvantage of gate arrays is their intrinsic low density of active silicon.

Standard cell designs use a set of well-proven logic cells on the chip, much in the way that previous generations of standard logic have been used as board-level products, such as Texas Instruments' System 74.

A variation on the semi-custom approach is to include full-custom macrocells such as processor cores in fixed positions on the wafer.

*Exercise:* Ignoring the title, determine from the data sheet whether the cell in figure 80 is for standard cell or gate array use ? What other information about the cell is needed to prepare an audit of resources used in a design which has used this cell ? The audit refered to is typically a report generated by the CAD tools which gives summary information.

### 4.5.3 Field-programmable logic

About 25 to 40 percent of chip sale revenue now comes from field programmable logic devices. These are chips which can be programmed electronically on the user's site to provide the desired function. The Xilinx FPGA parts used in the Part 1B E+A classes are one of the most important examples of field-programmable logic.

Field-programmable devices may be volatile (need programming every time after power up), re-programmable or one-time programmable. This relates to how the programming information is stored inside the devices, which can be in RAM cells or in any of the ways mentioned for ROM devices in section 1.7.

### 4.5.4 FPGAs

An FPGA (field-programmable gate array) consists of an array of configurable logic blocks (CLBs), as shown in Figure 71. Not shown is that the device also contains a good deal of hidden logic used

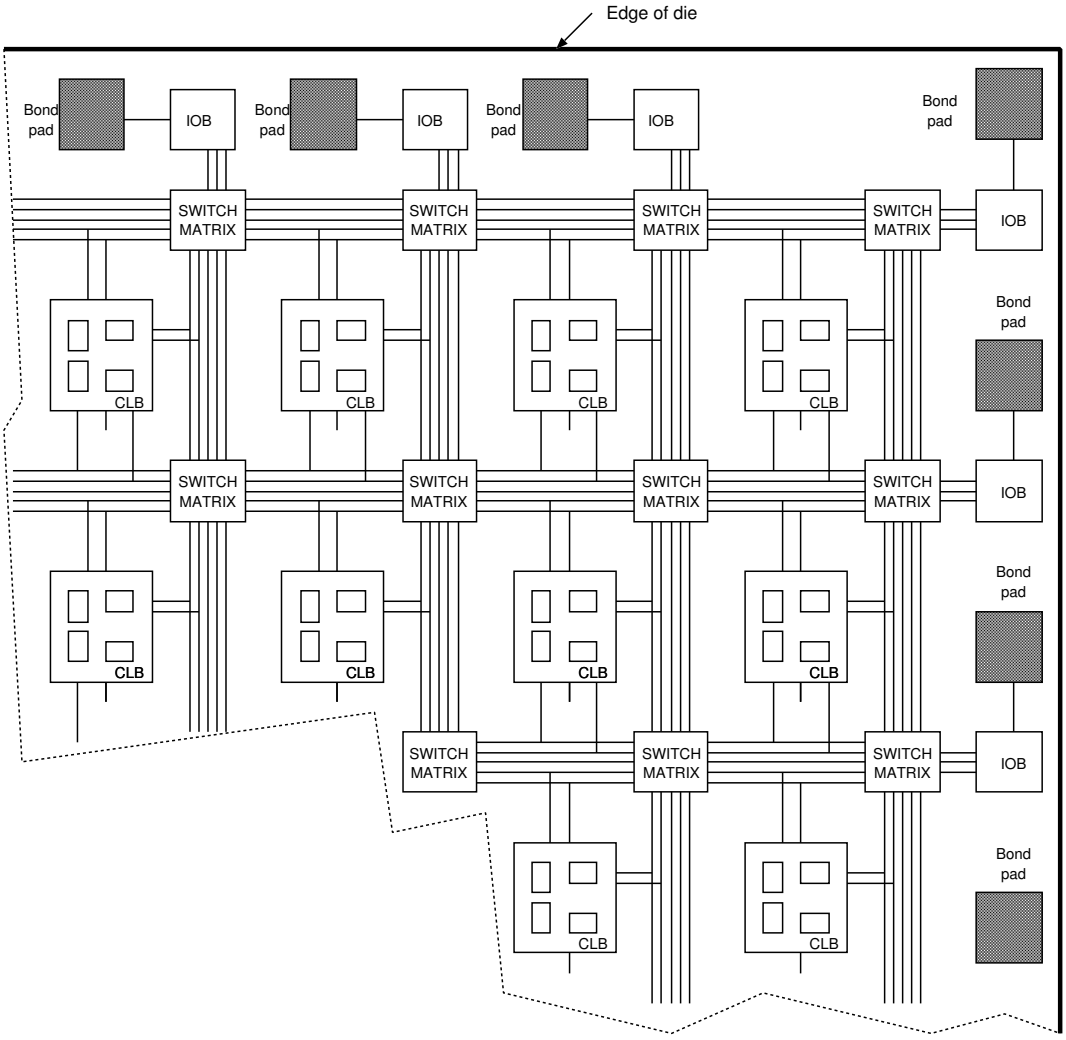



Figure 71: Field-programmable gate array structure, showing IO blocks around the edge, interconnection matrix blocks and configurable logic blocks.

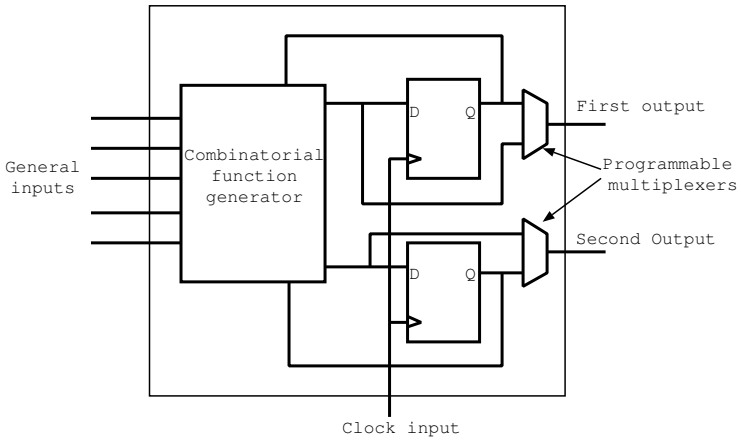



Figure 72: A configurable logic block for a look-up-table based FPGA.

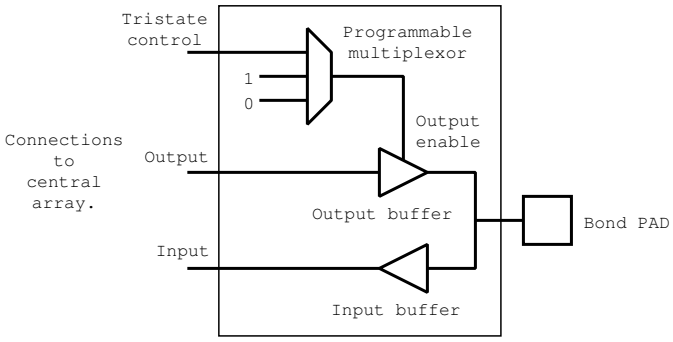



Figure 73: A simple IO block for an FPGA.

just for programming it. Some pins are also dedicated to programming. Such FPGA devices have been popular since about 1990.

Each CLB (Figure 72) typically contains one or two flip-flops, and has a few (five shown) general purpose inputs, some special purpose inputs (only a clock is shown) and two outputs. The illustrated CLB is of the look-up table type, where the logic inputs index a small section of preconfigured RAM memory that implements the desired logic function. For five inputs and one output, a 32 by 1 SRAM is needed. Some FPGA families now give the designer write access to this SRAM, thereby greatly increasing the amount of storage available to the designer. However, it is still an expensive way to buy memory.

All CLBs within a FPGA generally have the same structure, but FPGAs are available with lower and higher functionality CLBs. The best size of CLB is not yet clear. Modern designs of FPGA have a hierarchy of CLB interconnection patterns, giving CLB clusters within clusters.

An FPGA is very like a mask-programmed gate array to use. The design flow and CAD tools are virtually identical. The expenditure before the designer has the first device in her hands might be 1000 times lower. The cost of further devices is at least 10 times higher than mask-programmed devices, owing to the programming cost and wasted die area devoted to the programming activities. Clearly there is a crossover production volume point!

FPGAs also tend to be quite slow, owing to the signals passing through hidden logic used only for configuration.

Often a designer or company will build prototypes and early production units using FPGAs and then use a drop-in mask-programmed equivalent once the design is mature and sales volumes pick up.

*Exercise:* Attempt to sketch the logic needed inside one of the programmable switch matrix boxes. You should find this is a vast amount of logic and therefore understand why FPGAs are so expensive for their functionality. (If you know how to use pass transistors, this will help your design).

### 4.5.5 PALs

A PAL is programmable array logic device. Figure 74 shows a typical device. Such devices have been popular since about 1985. The illustrated device has 8 product terms per logic function, and so can support functions of medium complexity. Such devices are very widely used and can feature high speed operation with clock rates of above 100 MHz. They are really just highly structured gate arrays. Every logic function must be multiplied out into sum-of-products form and hence is achieved in just two gate delays.

Programmable *macrocells* (Figure 75) enable the output functions to be either registered or combinatorial. Small devices (e.g. with up to 10 macrocells) offer one clock input; larger devices with up to about 100 macrocells are also available, and generally offer several clock options. Often some macrocells are not actually associated with a pin, providing a so called *buried state* flip-flop.

Figure 76 shows part of a PAL description for the PAL shown in figure 74, as entered by a designer

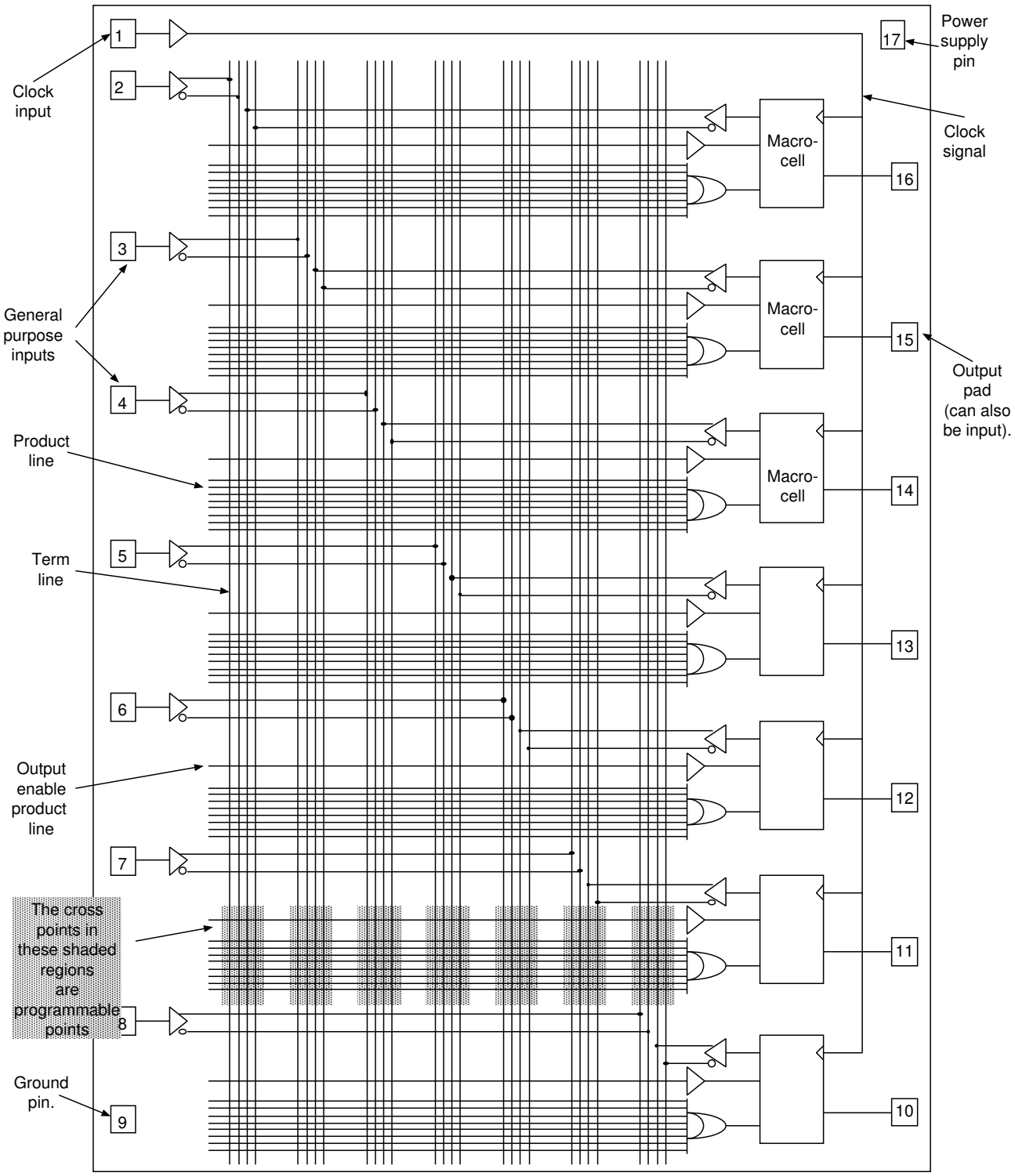



Figure 74: A typical PAL with 7 inputs and 7 I/Os.

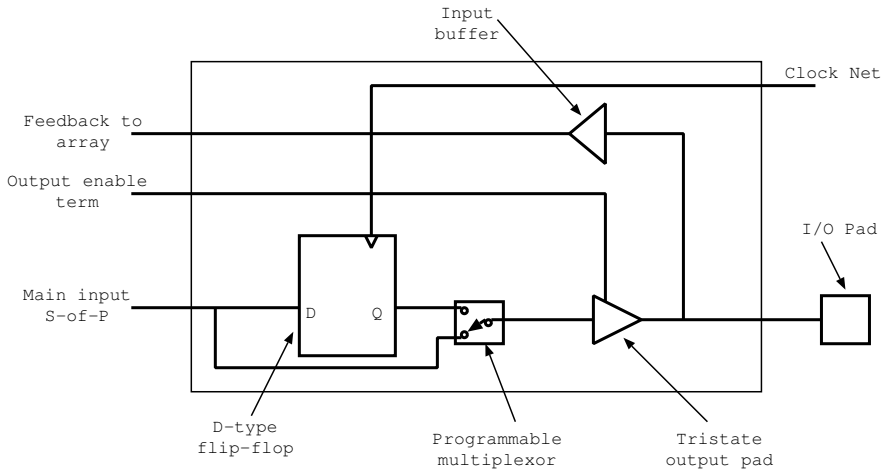



Figure 75: Contents of the PAL macrocell.

```
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)
```

Figure 76: Example programming of a PAL showing only fuses for the top macrocell.

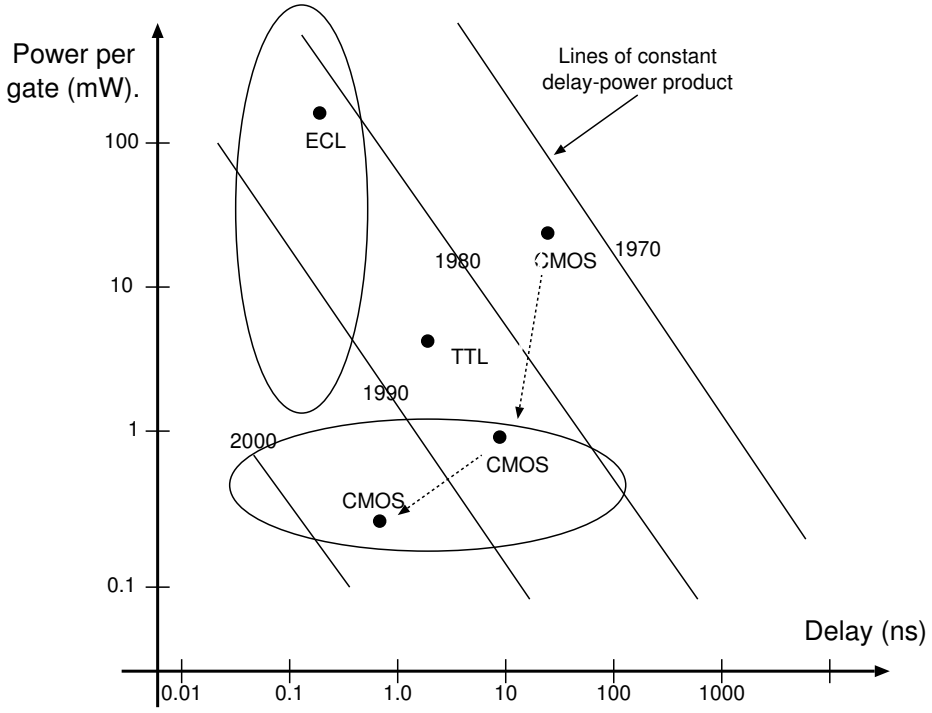



Figure 77: Delay-power style of technology comparison chart.

in a typical PAL language, and part of the fuse map that would be generated by the PAL compiler. Each product line has seven groups of four fuses and produces the logical AND of all of the signals with intact fuses. An 'x' denotes an intact fuse and all of the fuses are left intact on an unused product lines in order to prevent the line ever generating a logical one (a gets ANDed with abar etc.). The fuse map is loaded into a programming machine (in a file format known as JEDEC), an unused PAL is placed in the machine's socket and the machine programs the fuses in the PAL accordingly.

| *Exercise:* How large a binary counter can the illustrated device implement ? (Hint: are there sufficient product terms for the most-significant bit ?) |
|---|

PALs achieve their speed by being highly structured. Their applicability is restricted to small finite state machines and other *glue logic* applications.

## 4.6 Delay-power product

Aside from the selection of design methodologies just discussed, the designer must choose the basic circuit technology to use:

- silicon or GaAs
- bipolar or unipolar
- saturating or non-saturating.

As the size of transistors on chips is reduced with ever improving fabrication equipment, the number of devices per unit area increases and the delay-power product (power consumption multiplied by propagation delay) decreases.

We here consider the main three circuit technologies of the 70's and 80's using a quad AND gate chip SSI chip for example (shown generically in Figure 78). In the 90's, CMOS is dominating most application areas.

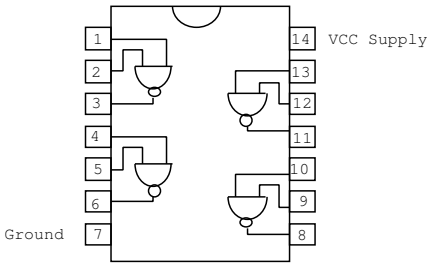



Figure 78: A SSI quad NAND gate in the 7400 pinout.

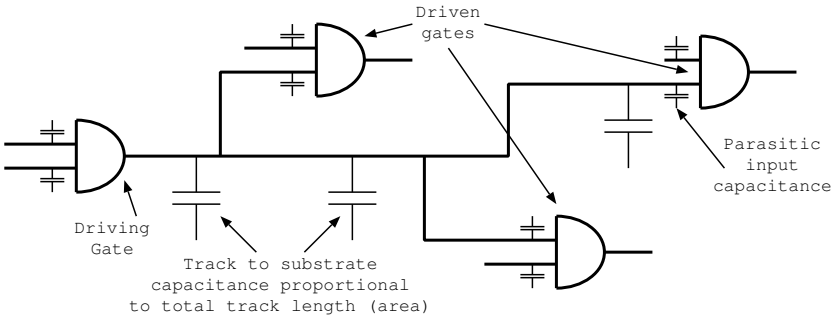



Figure 79: Logic net with tracking and input load capacitances.

| technology | device | propagation delay (ns) | power (mW) | product (pJ) |
|---|---|---|---|---|
| CMOS | 74hc00 | 7 ns | 1 mW | 7 |
| TTL | 74f00 | 3.4 ns | 5 mW | 17 |
| ECL | sp92701 | 0.8 ns | 200 mW | 160 |

The CMOS gate has the property that it consumes virtually no power when outputs are not changing.

The ECL gate is an old technology, but it is still the fastest.

Gates in the core of an IC tend to be one order better in speed or power when compared with the SSI gate used in the example (they have reduced drive requirement).

## 4.7 Fanout and delay estimations.

Figure 79 shows a typical net, driven by a single source. To change the voltage on the net, the source must overcome the stray capacitance and input loads. The fanout of a gate is the number of devices that its output feeds. The term *fanout* is also sometimes used for the maximum number of inputs to other gates a given gate is allowed to feed, and forms part of the design rules for the technology.

The speed of the output stage of a gate, in terms of its propagation delay, decreases with output load. Normally, the dominant aspect of output load is capacitance, and this is the sum of:

1. the capacitance proportional to the area of the output conductor
2. the sum of the input capacitances of the devices fed.

To estimate the delay from the input to a gate, through the internal electronics of a gate, through its output structure and down the conductor to the input of the next gate, we must add three things:

## 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<br>b<br>c<br>d | 2.1<br>2.1<br>2.1<br>2.0 | Load units |
| Drive capability | f | 35 | Load units |

Figure 80: An example cell from a manufacturer's cell library.

| Technology | Maximum clock speed | Maximum gate count | Maximum pin count |
|---|---|---|---|
| GaAs bipolar | 50 GHz | 50 | 5 |
| GaAs fet | 3 GHz | 300 | 100 |
| Si ECL | 3 GHz | 5000 | 300 |
| Si CMOS | 400 MHz | 1 E6 | 500 |
| Within Si CMOS | | | |
| Full-custom | 400 MHz | 1 E6 | 500 |
| Standard cell | 200 MHz | 40000 | 500 |
| Gate array | 100 MHz | 20000 | 500 |
| PAL | 75 MHz | 500 | 68 |
| FPGA | 35 MHz | 4000 | 200 |

Table 2: Approximate limits on contemporary technology (1987).

1. the internal delay of the gate, termed the intrinsic delay
2. the reduction in speed of the output stage, owing to the fanout/loading, termed the derating delay,
3. the propagation delay down the conductor.

The propagation delay down a conductor obeys standard transmission line formula and depends on the distributed capacitance, inductance and resistance of the conductor material and adjacent insulators. For circuit board traces, resistance can be neglected and the delay is just the *speed of light* in the circuit board material: about 7 inches per nanosecond, or 200 metres per microsecond.

On a chip, the speed of light can be neglected because chips are physically small, but the resistance of the aluminium conductors is sufficiently large to have an effect for critical applications, such as master clock signals.[1]

The upshot of this is that on a chip, we can mostly neglect the third aspect of delay, whereas on a circuit board, we need to model certain critical conductors as components in themselves. These have a simple delay model, whose value can be set by post routing back annotation.

The first two aspects of delay may easily be absorbed into the model for a component. We can use the following formula

$$\text{device delay} = (\text{intrinsic delay}) + (\text{output load} \times \text{derating factor}).$$

The intrinsic speed of a device is given in its data sheet, as is the derating factor and the loading factor of its inputs. Typical values for a standard cell array are shown in Figure 80. The output load is the sum of a track dependent part and the fanout dependent part.

The track dependent part is a library constant times the track area.

The load dependent part is the sum of the input loads of all of the devices being fed.

*Exercise:* How true is the above model of signal delay when we use field-programmable gate arrays with high effective track resistance ? The conductors on FPGAs consist of many sections of real metallic conductor interconnected by the user-programmed connection points. We then have considerable resistances at points along each conductor's path. How would you estimate the various delays for the staggered arrival of a signal on each part of the net ?

## 4.8 Comparative view of digital logic technologies

Table 2 gives some approximate numbers for what can be achieved with today's (1987) chips. These numbers are only accurate to about half an order of magnitude. Technology is always advancing, so these figures soon become way out of date!

[1] Older technologies used polysilicon interconnections which had significant resistance and so the different gates connected to a polysilicon net would experience different arrival times of a signal at their inputs.

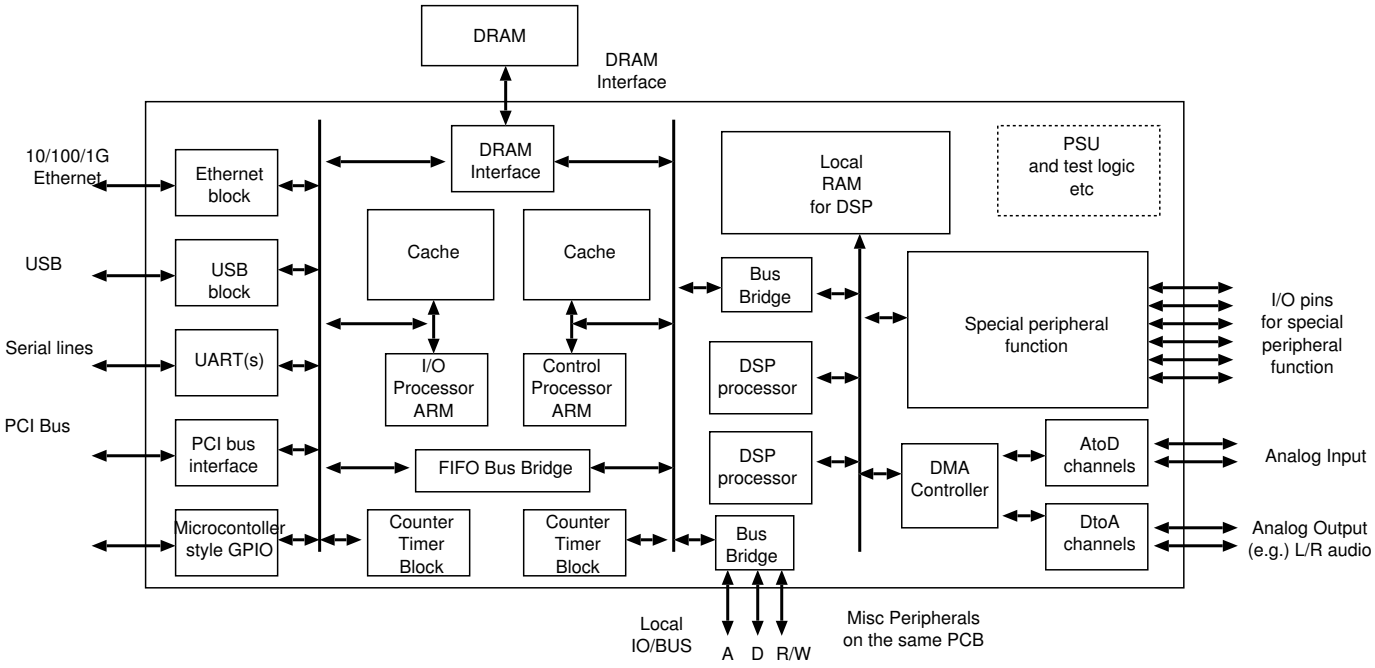



Figure 81: Typical Platform Chip

For CMOS technology, the number of transistors on a chip and the clock rate both seem to double every three years.

## 4.9 SoC Platform Chips

A platform chip is the modern (2004) equivalent of a microcontroller. The set of components remains the same, but each has far more complexity: e.g. 32 bit processor instead of 8. In addition, rather than putting a microcontroller on a PCB as the heart of a system, the whole system is placed on the same piece of silicon as the platform components. This gives us a system on a chip (SoC).

The example illustrated in figure 81 has two ARM processors and two DSP processors. Each ARM has a local cache and both store their programs and data in the same off-chip 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, albeit 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 peripheral.' 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.

# 5 Hardware, Software and Design Partition.

*Learners' Guide:* What you should learn from this section is:

- The functions of a system can be expressed in a programming language or similar form and this can be compiled fully to hardware or left partly as software
- Choosing what to do in hardware and what to do in software is a key decision. Hardware gives speed (throughput) but software supports complexity.
- Partitioning of logic over chips or processors is motivated by speed, technology and module reuse.

A hardware system can be designed using only hardware or hardware with an embedded processor and software in ROM. Perhaps several processors are to be used. A design also must be partitioned into standard components and application specific integrated circuits (ASICs). The choice of which parts of the design are realised in what technology is the design partition problem.

## 5.1 Embedded Systems

A processor that is permanently connected to a single ROM which contains all of the software that the processor will ever execute is called an *embedded processor*. Here we consider when to use an embedded processor, and then, when using one, the designer has to chose whether to design a new one or use an existing one.

**The main difference between a hardware solution and a software solution is the degree of parallelism.** Processors typically execute one instruction at a time, re-using the same hardware components again and again. Hardware solutions tend to have dedicated circuits for each function. If most of the hardware is likely to be idle for most of the time, a processor is preferred, but if a processor cannot achieve the throughput required, increased parallelism using hardware is preferred.

Complex functions normally require a processor, but CAD tools are evolving, allowing complex functions to be expressed algorithmically and then automatically converted to a logic gate implementation.

High-speed processing normally requires dedicated hardware. For instance, consider the error correction performed by a CD player to overcome dirt and scratches. When CD players first came out, this error correction was done with dedicated hardware, but today, microprocessors have increased in speed, and so the function can be done using the processor that is already there to provide other complex functions (e.g. track skip). However, on the latest, x52 speed CD/DVD drives the error correction must be performed that much faster, and so dedicated hardware is re-introduced.

**Processors can be placed on an ASIC.**

Processors give flexibility: if the design is likely to be changed in minor ways to form new models, or to implement field upgrades, then using a processor that may be given a new software release is a good technique.

Standard processor chips can be very cheap for a given performance. This is because a very great deal of effort is put into their design and they are sold in large quantities, thereby reducing price. Part of the cost of any product is in testing it. A standard processor not only comes with its own test qualification programme, it is able to execute software to perform tests on the rest of the system.

The decision to design and use a custom processor for an application should not be taken lightly. It can be useful, however, when the application is fairly simple or highly specialised. Examples are found in signal processing, where a particular algorithm needs to be executed at high speed; for instance, to track a missile or in the stready-cam function of a camcorder.

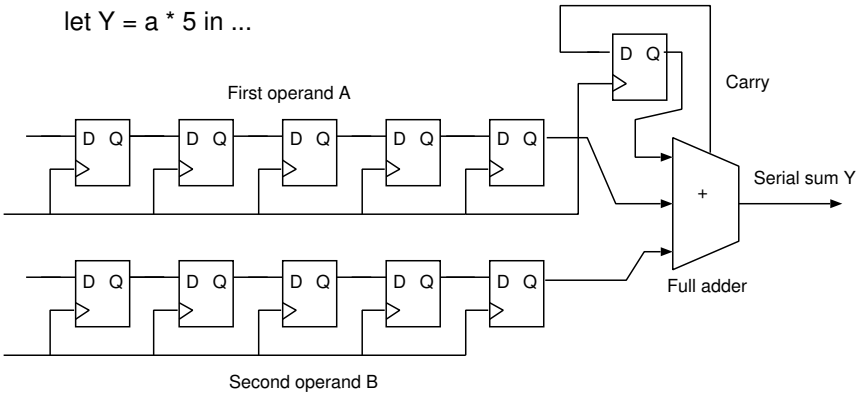



Figure 82: Addition of two integers serially, l.s.b. first

## 5.2 Logic Synthesis and Time/Space Exchange

If one considers an embedded processor connected to a ROM, it may be viewed as one large FSM. Since for any given piece of software, the ROM is unlikely to be full and there are likely to be resources in the processor that are not used by that software, the application of a good quality logic minimiser to the system, while it is in the design database, could trim it greatly. In most real designs, this will not be helpful: for instance, the advantages of full-custom applied to the processor core will be lost. In fact, the minimisation function may be too complex for most algorithms to tackle on today's computers.

On the other hand, algorithms to create a good static scheduling of a fixed number of hardware resources work quite well. A processing algorithm typically consists of multiple processing stages (e.g. called pre-emphasis, equalisation, coefficient adaptation, FFT, deconvolution, reconstruction and so on). Each of these steps normally has to be done within tight real-time bounds and so parallelism through multiple instances of ALU and register hardware is needed. The Cathedral DSP compiler was an early and famous tool for helping design such circuits. Such tools perform time/space folding/unfolding of the algorithm to generate the static shedule that maps operations and variables in a high-level description to actual resources in the hardware. Data dependencies can cause variations in the time for certain steps, so a potentially a dynamic schedule could make better use of resources but the overhead of dynamic scheduling can outweigh the cost of the resources saved if the data dependencies are rare.

### 5.2.1 Hardwired, bit-serial algorithms.

Bit-serial arithmetic is also a key approach in custom signal processing. The adder shown in figure 82 requires only about 10 gates to add any number of bits. It operates l.s.b. first so that the carry from one digit is available for the next digit. The clock speed for the calculation does not need to be reduced if longer words are processed, but the pipeline delay does go up. Multiplication is also easy in bit serial form, as shown for multiplication by a constant in figure 83. Consider the same functions implemented in naive, broadside implementations.

A battery operated walkman CD ROM player may implement a great deal of DSP. The printing on the unit may advertise '8 times oversampling D-to-A bitstream convertor and MASH bit mapping'. The processing power required for this is comparabel to the performance of a desktop computer, **but through the use of bit serial arithmetic and hard-wired algorithms, the processing cab be implemented with little power or silicon area.**

The problem with bit-serial processing for the human designer is that it is very tricky to work out where individual bits are at any one time, and the optimisations that are possible cannot readily be seen by the human. Therefore, good quality CAD tools are vital in the design of such systems.

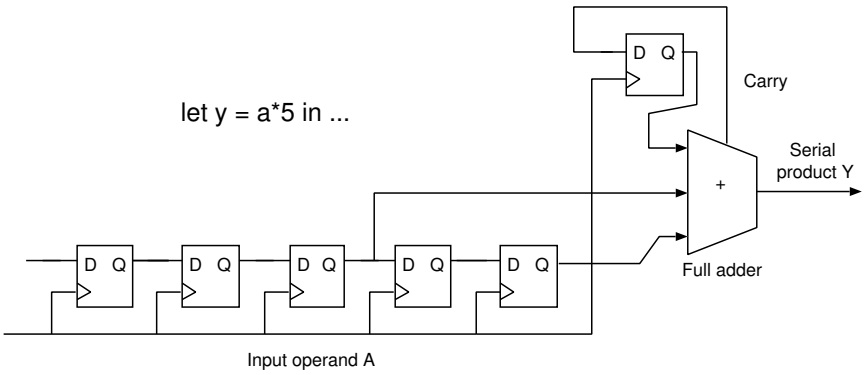



Figure 83: Bit-serial multiplication of an integer by a hardwired constant

## 5.3 ASICs

The cost of developing an ASIC has to be compared with the cost of using an existing part. The existing part may not perform the required function exactly, requiring either a design specification change, or some additional *glue logic* to adapt the part to the application.

More than one ASIC may be needed under any of the following conditions:

- application specific functions are physically distant
- application specific functions require different technologies
- application specific functions are just too big for one ASIC
- it is desired to split the cost and risk or reuse part of the system later on.

Factors to consider on a per chip basis:

- pad count limitation (pad density limit of 15 per mm)
- power consumption limitation (powers above 5 Watts need special attention)
- gate count limitation (above 500 kgates is getting big for CMOS)
- speed of operation — influences choice of technology
- will it be core or pad bound ?
- special considerations :
  - special static or dynamic RAM needs
  - analogue parts - can these also be integrated onto the ASIC ?
  - high power handling outputs for load control: e.g. motors.
- availability of a developed module for future reuse

Power dissipation is generally proportional to:

$$\text{Power} \propto (\text{clock speed}) \times (\text{number of gates}) \times (\text{supply voltage squared}).$$

Various technologies can be positioned in the speed-power plane, as shown in Figure 77 and as time goes by, things get better.

## 5.4 Partitioning example: The Cambridge Fast Ring two chip set.

Two devices were developed for the CFR local-area network, illustrating the almost classical design partition required in high-speed networking. They were never given grander names than the *ECL* chip and the *CMOS* chip. The block diagram for an adaptor card is shown in Figure 84.

The ECL chip clocks at 100 MHz and contains the minimal amount of logic that needs to clock at the full network clock rate. Its functions are:

- implement serial transmission modulator and demodulator
- convert from 1 bit wide to 8 bits wide and the other way around
- perform reception byte alignment (when instructed by logic in the CMOS chip).

Other features:

- ECL logic can support analogue line receivers at low additional cost so can receive the incoming signal directly on to the chip.
- ECL logic has high output power if required (1 volt into 25 ohms) and so can drive outgoing twisted pair lines directly.

The CMOS chip clocks at one eighth the rate and handles the complex logic functions:

- CRC generation
- full/empty bit protocol
- minipacket storage in on-chip RAM
- host processor interface
- ring monitoring and maintenance functions.

The ECL chip has at least 50 times the power consumption of the CMOS chip. The CMOS chip has more than 50 times the gates of the ECL chip.

Two standard parts are used to augment the CFR set: the DRAM chip incorporates a dense memory array which could not have been achieved for anywhere near the same cost onboard the CMOS chip and the VCO (Voltage Controlled Oscillator) device used for clock recovery was left off the ECL chip since it was a difficult-to-design analogue component where the risk of having it on the chip was not desired.

PALs are used to 'glue' the network interface itself to a particular host system bus. Only the glue logic needs to be redesigned when a new machine is to be fitted with the chipset. PALs have a short design turn-around time since they are field-programmable.

For a larger production run, the PALs would be integrated onto a custom variant of the CMOS chip.

## 5.5 Partitioning example: An external PC Modem.

Figure 86 shows the block diagram of a typical modem. The illustrated device is an external modem, meaning that it sits in a box beside the computer and has an RS-232 serial connection to the computer. It also requires its own power supply.

The device contains a few analog components which behave broadly like a standard telephone, but most of it is digital. A relay is used to connect the device to the line and its contacts mirror the 'off-hook' switch which is part of every telephone. It connects a transformer across the line. The relay and transformer provide isolation of the computer ground signal from the line voltages. Similarly the ringing detector often uses a opto-coupler to provide isolation. *Clearly, these analog aspects of the design are particular to a modem and are designed by a telephone expert.*

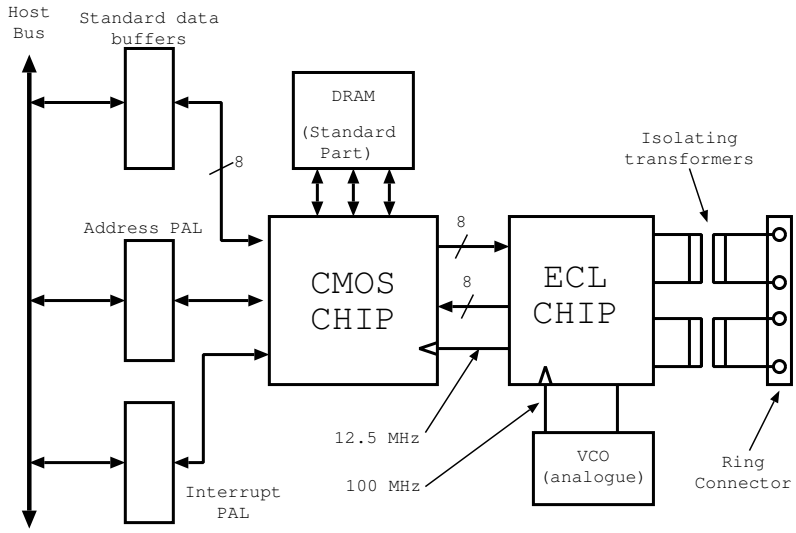



Figure 84: Example of a design partition — the adaptor card for the Cambridge Fast Ring.

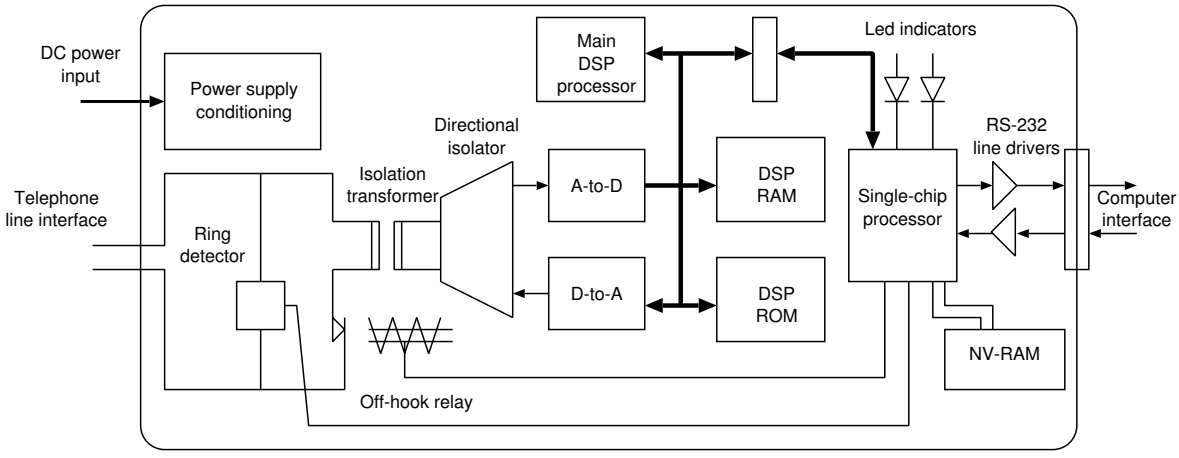



Figure 85: Example of a design partition — an external modem.

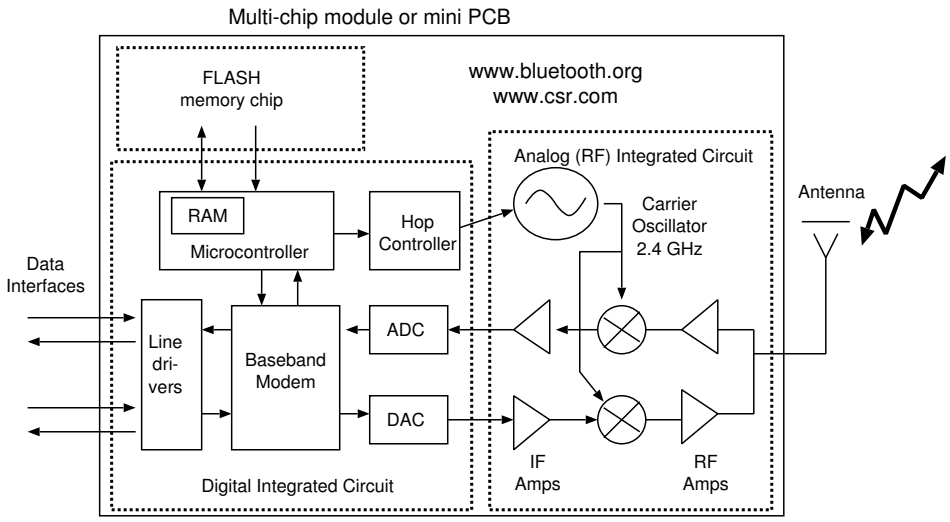



Figure 86: Example of a design partition — an early Bluetooth radio module.

Modems from the 1960's implemented everything in analog circuitry since microprocessors and DSP were not available. Today, two microprocessors are often used, but as processing power increases, this can be reduced to one (or sometimes even none, if the main processor of a PC provides the functions needed).

The reason for two processors are interesting and will be discussed in lectures.

Note that the non-volatile RAM requires a special manufacturing processing step and so is not included as a resource on board the singe chip processor. Similarly, the RS-232 drivers need to handle voltages of +/- 12 volts and so these cannot be included on chip without increasing the cost of the rest of the chip by using a fabrication process which can handle these voltages. The NV-RAM is used to store the owner's settings, such as whether to answer an incoming call and what baud rate to attempt a first connection, etc..

## 5.6 Partitioning example: A Bluetooth Module.

An initial implementation of the Bluetooth radio was made of three pieces of silicon bonded onto a small fibreglass substrate with overall area of 3 square centimetres. One application for these miniature radios is the replacement of wired cables, and so the module has line drivers that are compatible with the signals on the cables being replaced.

The module was partitioned into three pieces of silicon partly because the overall area required would give a low yield, but mainly because the three sections used widely different types of circuit structure.

The analog integrated circuit contained amplifiers, oscillators, filters and mixers that operate in the 2.4 GHz band. This was too fast for CMOS transistors and so bipolar transistors with thin bases were used. The module amplifies the radio signals and converts them using the mixers down to an intermediate frequency of a few MHz that can be processed by the ADC and DAC components on the digital circuit.

The digitial circuit had a small amount of low-frequency analog circuitry in its ADC and DACs and perhaps in its line drivers if these are analog (e.g. hifi). However, it was mostly digital, with random logic implementations of the modem functions and a microcontroller with local RAM. The local RAM holds a system stack, local variables and temporary buffers for data being sent or received.

The FLASH chip is a large, regular, non-volatile memory array that can hold firmware for the microcontroller, parameters for the modem and encryption keys and other smart card functions.

Today, the complete Bluetooth module can be implemented on one piece of silicon, but this

still presents a major technical challenge owing to the diverse requirements of each of the sub-components.

## 5.7 Hardware-Software Codesign

**Codesign is the name for an approach to system design where both the hardware and software components are entered in a common language and the selection of target architecture is specified in other ways, including automatically.** New languages for codesign are a research area. Existing languages like C and Java can also be used provided program bounds in terms of recursion depth and dynamic storage allocation can be determined fully at compile time.

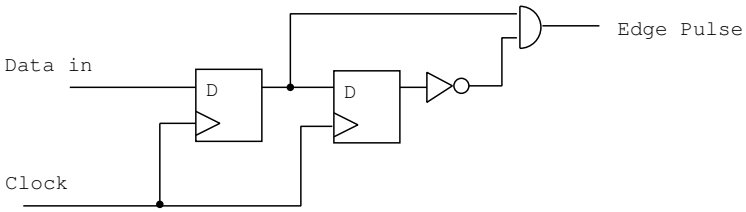



Figure 87: A simple edge detector

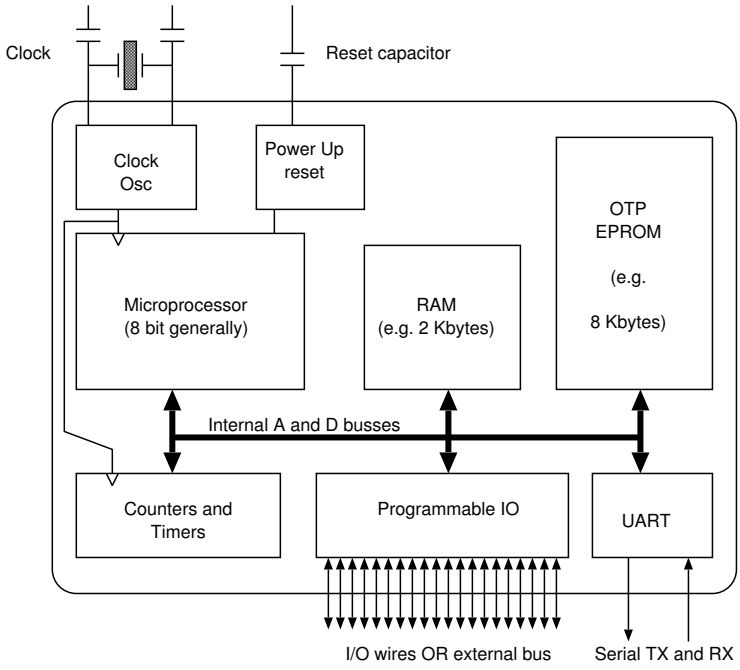



Figure 88: A typical single-chip microcontoller.

# 6 Further Example Circuit Structures.

*Learners' Guide:* What you should learn from this section is:

- Several further illustrations of design approaches.
- Consider the complexity of providing a user interface to items of hardware.
- A microprocessor often helps with control and the design of the user interface even if it is not really needed for the actual operation of a device.
- Whether to use custom integrated circuits depends on selling volume and availability of standard parts.

This section introduces some further, fundamental building blocks and includes examples of real products. The important points to note are the reasons for the design partition.

## 6.1 A Microcontroller: A Single Chip Processor

Figure 88 shows the block diagram of a typical single-chip computer or microcontroller. Such devices are available in 24, 40 or 80 pin packages and can cost around a pound. The device contains the whole of a computer, requiring externally only a power supply and some clock and reset components. A number of programmable I/O pins are provided, but generally another basic

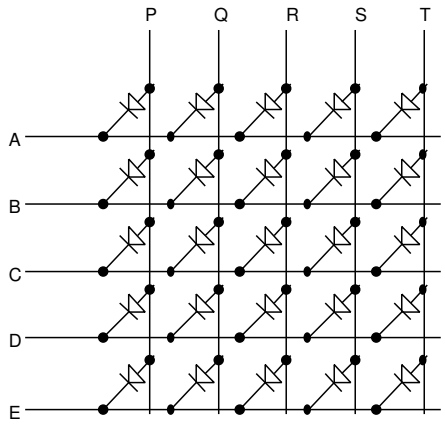



Figure 89: LEDs wired in a matrix to reduce external pin count

mode of operation is provided where 24 or so of them turn into the processor address and data busses for connection of external devices, such as further RAM or EPROM.

Programming is performed in all of the ways available for ROMs: today this is mainly mask programming and Flash.

A Universal asynchronous recevier and transmitter (UART) is normally always provided to help connect to RS-232 serial interfaces. Other standard interfaces, including Ethernet and USB are becoming common on microcontrollers.

These devices are found everywhere today, from keyboards, clock radios, mice, telephones, car window winders, 'computer controlled' cassette recorders, cell phones and modems.

## 6.2 Scan Multiplexing.

When a large number of leds or switches need to connected, as in a display or keyboard, the number of connections can normally reduced by connecting them in a matrix. Figure 89 shows that 25 LEDs can be connected to just ten signals.

The matrix gives a *scan-multiplexed* display or keyboard. In the display, one vertical column line can be driven to logic one at a time and a zero placed on the horizontal lines that should be illuminated in that column. A circuit to repeatedly read out the contents of a RAM and display it is shown in figure 91. Clearly, the desired LEDs are not all on at once, but by scanning the display faster than the human eye can detect flashes (about 50 Hz) and by using sufficiently large currents, so that the elements are brighter than would otherwise be required, this is overcome. The current is set by the value of a series current limiting resistor that is not shown.

For a scan-multiplexed keyboard, the switches take the place of the LEDs. Push-to-make, normally open switches must be used and the user should not press two at once. The scanning circuit must take one row line low in turn. Pull-up resistors keep the column lines at logic one unless a switch to a low row line is pressed.

Figure 90 shows the full PCB circuit of a typical infra-red remote controller. The PCB circuit of a toy electric organ would be identical, but with the IR diode replaced with a loudspeaker: clearly the chip would be different. A pocket calculator also has roughly the same circuit, except there is a keyboard and a display to multiplex.

*Exercise (long):* Sketch the full circuit of the chip in an infra-red remote control handset, as used for TVs and VCRs etc.. The device should have about 50 push keys. Output is through a pair of infra-red transmitting diodes which go on or off together. The design must include a ROM which can be programmed at chip manufacture with the desired sequence of pulses needed. The ROM must contain 16 bits for each key and these bits must be transmitted in turn through the IR diode using a 1 millisecond pulse of light for a zero and a 3 millisecond pulse for a one, with a 1 millisecond gap between each pulse.

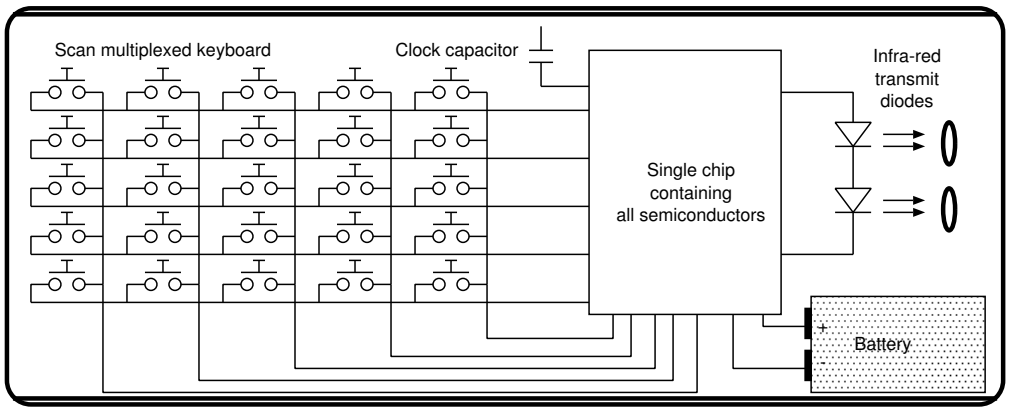



Figure 90: IR Handset Internal Circuit

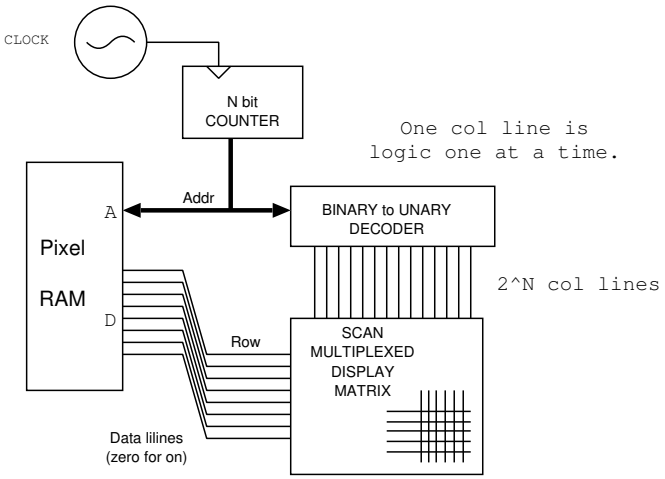



Figure 91: Scan multiplex logic for an LED pixel-mapped display

*Exercise:* For the IR handset, is it necessary for the clock to be running at all times or only when a key is pressed ? This design aspect will affect battery life greatly.

Figure 92 shows how a multiplexor can be added to allow update of the display RAM from a processor or similar. The display will briefly show the wrong data while being updated, but this may be imperceptible to the human. The compromise implied by such *psudo*-dualporting is accepted instead of having the cost penalty of truely dual-ported storage. (A second independent port to a RAM or register file greatly adds to silicon area).

Figure 93 shows the use of a ROM as a function look up table. All electric guitarists use a distorting amplifier and are pernickety about the exact nature of the distortion. By using a ROM which contains a function appropriately distorted from the identity function, any desired distortion can be achieved.

The input A-to-D circuit samples the input signal at 44100 samples per second, which is the CD sampling frequency. The D-to-A converter on the output reconstructs an analogue signal.

The analog signals will vary both positively and negatively about zero. Two's complement representations are therefore normally used. Note that to convert from offset binary to two's complement, one just puts an inverter in the most significant bit.

*Exercise:* Sketch an alternative circuit using a microprocessor instead of the hardwired solution. What pros and cons do the two approaches have ? How does a processor help with the user interface?

Figure 94 shows the use of a static RAM to achieve a delay. The counter addresses each location in turn, and at each location, the old contents are read out before the new contents are put back.

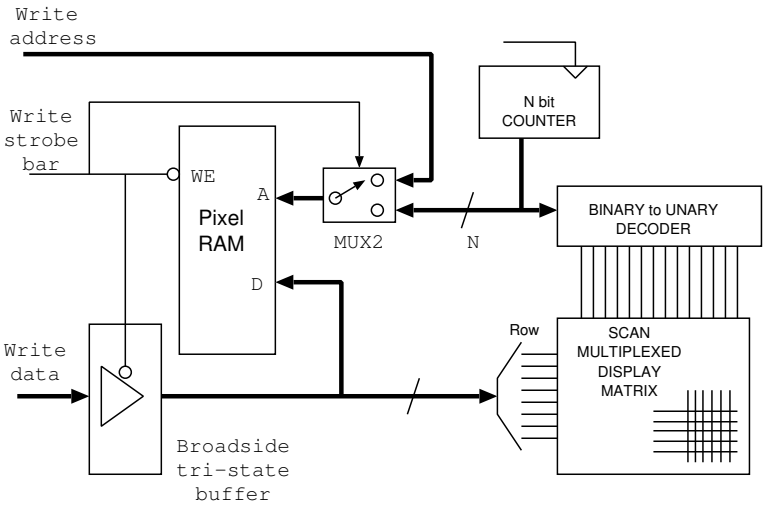



Figure 92: Addition of psudo dual-porting logic.

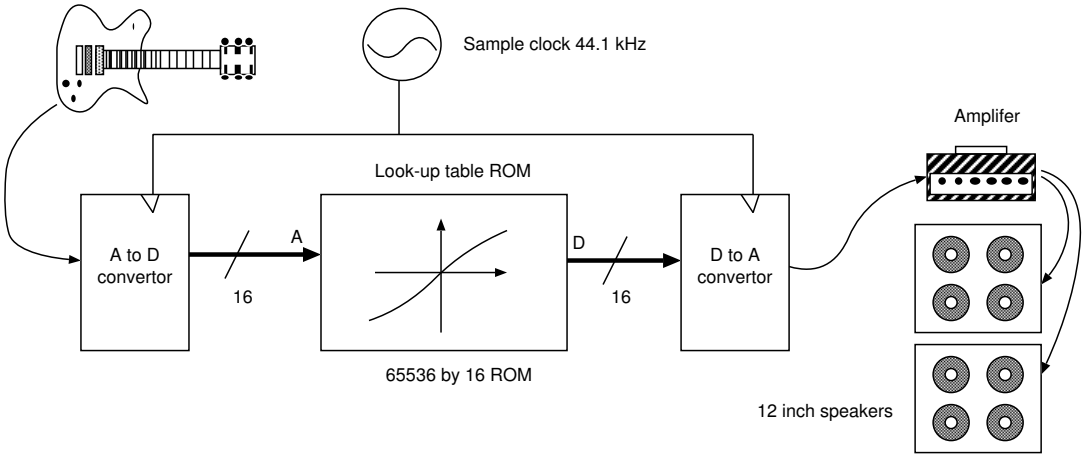



Figure 93: Use of a ROM as a function look-up table.

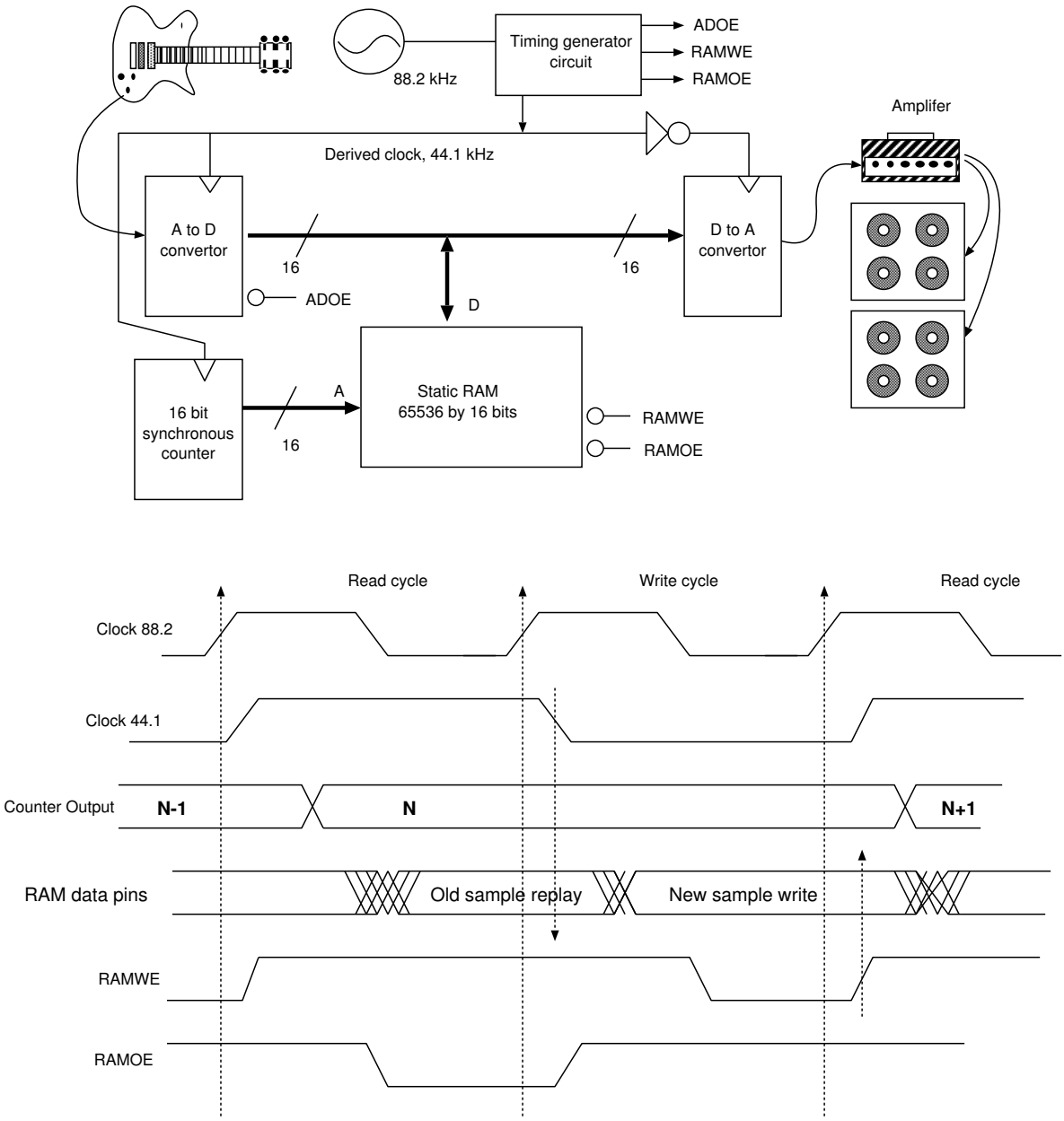



Figure 94: Use of an SRAM to make the delay required for an echo unit.

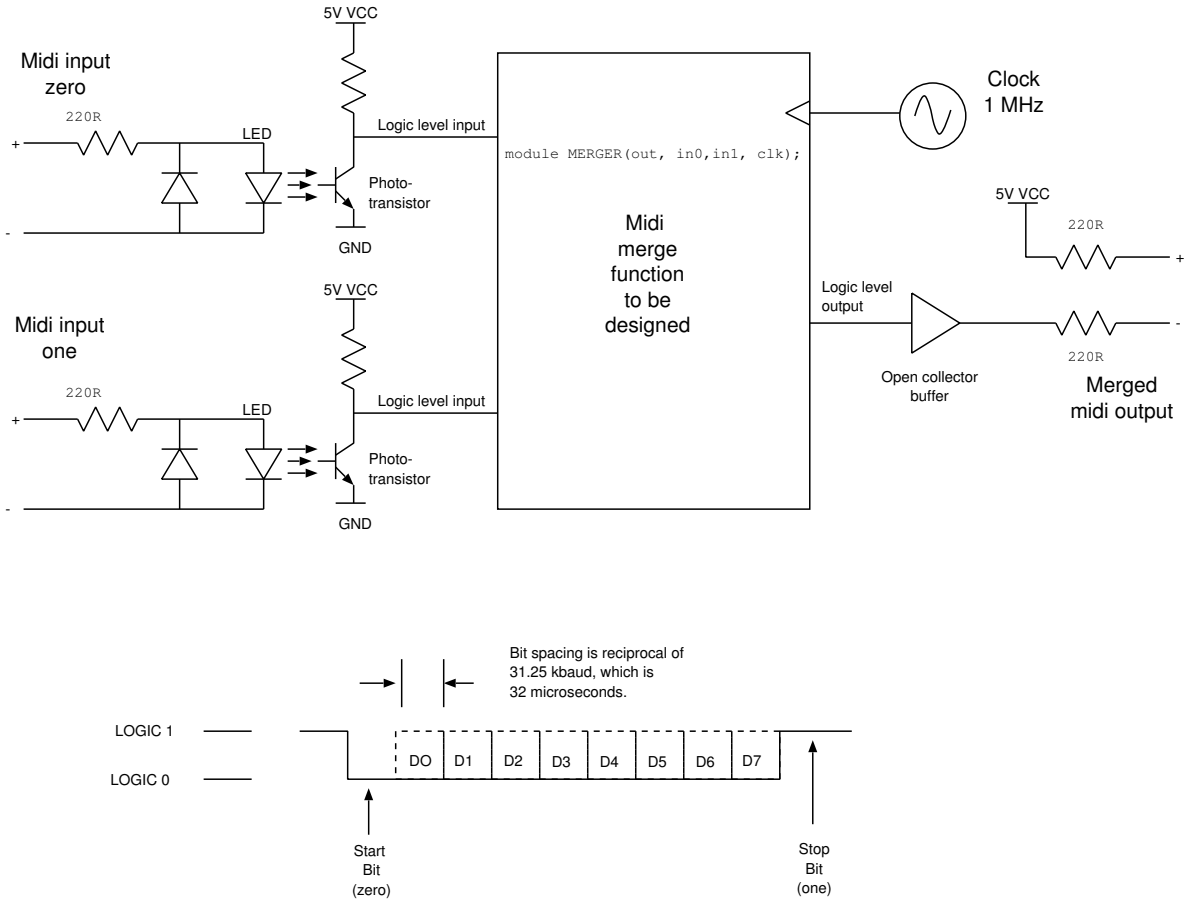



Figure 95: MIDI interface details and merge unit block diagram. MIDI serial data format.

Audio frequencies up to 20 kHz can be represented digitally by sampling at 40 kHz. A 96 dB signal-to-noise ratio is achieved with sixteen bits of sampling resolution, therefore 80 bytes of RAM is needed per millisecond of delay.

*Exercise:* The circuit shown produces only a delay. For a true echo effect, the original signal must also appear at the output. To achieve multiple echos, the output must be fed back to the input at reduced amplitude. Introduce an adder and any additional logic to achieve these results. Use the fact that division by a constant power of two can be achieved by shifting right while leaving the sign bit unchanged.

## 6.3 Midi Design Example.

This section contains a design example. The design example is a MIDI merge unit. *No part of this section is directly examinable.*

## 6.4 MIDI Merge Unit Specification

MIDI is the musical instrument digital interface and it is found on most electronic musical instruments. A MIDI connection is unidirectional and contains both control and real-time traffic. In this example, we will only concern ourselves with the real-time *note* data which is sent from a keyboard to a sound generator as the keyboard is played. A three-byte note command is sent for each key pressed or released on the keyboard. The three MIDI note commands we are interested in have the following formats (in hexadecimal)

```
9n kk vv        (note on)
```

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

where $n$ is a channel number, $kk$ is a note number and $vv$ is a velocity. The values of $n$ range from 0 to 15 and allow a single MIDI link to carry 16 virtual MIDI channels of data. The values of $kk$ are in semitones in the range 0 to 127 with 60 being middle 'C'. The values of $vv$ are from 1 to 127 where 127 is the loudest. Note that these parameters are all less than 80 hex. The first command (note on) is generated when a key is pressed and the second (note off) when it is released. The third is an alternative to the note off command when the release velocity is not used (which is the case for most keyboards). The benefit of the alternative accrues when *running status* is used. Running status is an optimisation of the MIDI protocol which brings the average number of bytes actually sent per note command down from three to closer to two. Since this is a real-time music playing protocol, and many note commands need to be sent when a large chord is played, reducing the number of bytes in this way is helpful, but it will impact our merge function. A status byte in MIDI is a MIDI byte in the range 80 to EF, and so this includes the note on and off commands. The rule for running status is that if a status byte needs to be sent and it is the same as the last one actually sent, then it does not need to be sent.

Figure 95 shows the desired configuration of the MIDI merge unit and the format of MIDI bytes on the wire. The actual MIDI cables operate with two wires carrying the send and return currents: this is known as a current loop circuit. The current is on for a zero and off for a one. The current is sufficient to drive a small led which is mounted in a dark space next to a photo-transistor. This optical link between devices avoids electrical connections which can cause earthing problems in a studio environment. When light falls on the photo-transistor, it conducts and pulls down the input wire to close to ground, creating a valid logic zero level. Otherwise the inputs are one. One is the idle line state. A MIDI bytes starts with a start bit of zero for 32 microseconds, followed by the data bits, l.s.b. first.

The function of the MIDI merge unit is to merge two streams of MIDI data: e.g. two different channels of MIDI bytes (different values of $n$) may need to be merged. The unit must be prepared to accept arbitrary timing and status relationships between its two inputs. If they are simultaneously active, then the data from one of the inputs will have to be slightly delayed and sent after the data from the other. If both are busy to more than 50 percent utilisation, the output cannot cope and some data must be discarded by the unit. Since this is a real-time system, the output needs to have as low a delay as possible from the input.

## 6.5 Top down design.

The next stage of design is to sketch an internal block diagram, as shown in figure 96. The system will assemble bits into bytes, then assemble bytes into three-byte command words. The command words will be stored in FIFO (first-in first-out) queues and then read out and merged. The reverse process is done on the transmit side.

Verilog is a textual language for designing hardware. The blocks in the block diagram are each suitable for encoding as a Verilog module. Except for the final parallel to serial block, each module will need an output guard signal which indicates when the data on the output wires of the module is valid. For most of the blocks we will define this to go high once, each time a new word is valid. For the FIFO blocks, we will implement a *handshake* on the output, where the valid signal indicates that the FIFO has some data in it (and that that data is currently sitting on the output wires) and a read signal which indicates that we have accepted the data, thereby causing the next data to move to the output.

We can define the signatures of the verilog modules as follows. First, the serial to parallel convertor

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

The running status remover should be

```
input clk;
```

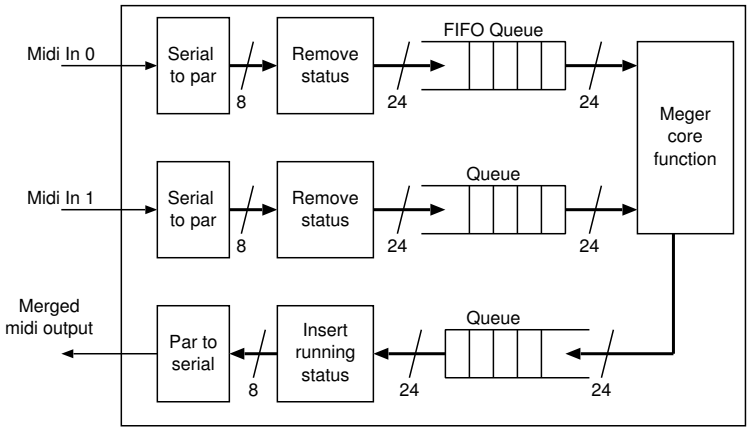



Figure 96: MIDI merge unit internal functional units.

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

For the FIFOs we should have

```
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 we should have

```
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;
```

And for the running status inserter and parallel to serial unit the signatures will be the reverse of the reciprocal units.

A full set of Verilog for the design is on the ECAD WWW page http://www.cl.cam.ac.uk/Teaching/1998/InroECA

## 6.6 Software Alternative

The function could be implemented with a microcontroller (single-chip processor) with dual serial ports. A suitable device is a PIC from Microchip and would cost just a pound or so. See http://www.microchip.com.

*Exercise:* Consider the pros and cons of the software implementation compared with the hardware implementation. Consider that the MIDI standard is mature and has not significantly changed in ten years. Consider the real-time response possible. *This exercise might be better undertaken after the Computer Design course.*

## 6.7 Design Compiler Challenge

Because the MIDI protocol works in real time, for the best performance, the MIDI merge unit should have a very low delay through it. The minimal possible delay might be just one clock tick, but the design presented in lectures and above always completely receives each command before sending it onward. A good part II project would be to write a program which read in the netlist of the MIDI merge unit and wrote out a netlist of an improved design with lower delay. The program

would work by applying general purpose transformations on the netlist that can manipulate it to achieve some goal, such as low delay. The interested reader should study symbolic bi-simulation algorithms to see how this could be done.

## Example Exam Questions

Q.An LED display has a diode at each cross point and is easy to scan multiplex because the diode only conducts in one direction. An LCD display has elements that darken when either polarity of voltage is applied. Explain why this makes scan multiplexing harder. One way to scan multiplex an LCD is to use tri-state buffers and to only drive, at any one instant, exactly one row and column line, with all other lines being tri-state. Design a scan multiplexing circuit for an LCD that works in this way. Comment on how you would improve the refresh rate of your design.

Q. An FPGA is to be used to generate a sequencer for disco lights. The FPGA will receive some pulses that are roughly in time with the music (generated by a crude analog circuit or else by a maniac with some switches) and generate 10 or so output signals to control the lights. There are also some control inputs to select the basic mode (blackout, strobes, dark, light, pattern select etc.). Sketch a block diagram of the whole system and sketch the circuitry of the FPGA. Is using an FPGA a good idea?

Q. A design is required to provide timing and information displays for the international final of the table-top Grand Prix (Scalelectrix) competition, which is to be televised by Eurovision. The budget for the system is fifty thousand pounds, including the sums paid to consultants (like yourself) and the system is expected to last for several years. Cars will be fitted with unique active tags, as used for counting livestock though gates on farms, and a sensor for the tags, together with light beams for accurate car detection, will be placed at the start-finish line and two intermediate points on the track. A number of display panels are needed based on LCD VGAs and a master operator's console is needed to control the system. Design the system. Pay attention to the modules needed and the wires that interconnect them. Give a 100 word instruction manual for the operator. If microprocessors are used, explain why each one is needed.

Q. What is semi-custom design ? When would you use semi-custom? Why might you use an FPGA to prototype a design that is later to be built in semi-custom? What cost difference would you expect between an FPGA and a semi-custom versions and why? How might your partition be affected by the amount of RAM needed?

Q. A design is required for a wand computer toy. The wand will have a barcode reader which will scan barcodes found on groceries and other goods and accumulate credit inside the wand in an unforgeable, non-volatile way. A group of young friends, each with their own wand, will be able to compete against each other, gathering points from each item scanned, with more points for rare/valuable goods. Design the wand, including its display and controls and a method for wands to connect to each other to exchange/swap barcode sightings.

Q. A circuit is needed to divide a very high frequency clock by an average of 21.6. This is to be done with a counter which accepts a division ratio input to make it divide either by 21 or 22. A further counter, made from slower and cheaper logic, is clocked from the output which generates the control signal to the divider, such that the ratio of 21 and 22 counts overall gives the desired target division ratio. Design it. Prove there are no hazards in your design. Estimate the maximum clock frequency given that the setup time and gate delays of the higher speed logic are 1 nanosecond. (This question is no longer core to this course).