Structured Hardware Design

Six lectures for CST Part Ia (50 percent).

Easter Term 2005.

(C) DJ Greaves.
Preface

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

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


W.Ditch. ‘Microelectronic Systems, A practical approach.’ Edward Arnold. The final chapters with details of the Z80 and 6502 are not relevant to this course.


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

Randy H Katz. ‘Contemporary logic design.’
Encoder and Decoder
(Revision)

Priority Encoder

Module: priencoder(d, Q);
output [1:0] Q;
input [3:0] d;
endmodule

Binary to Unary Decoder

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
Multiplexor (Revision)

Multiplexor

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

Distributed Multiplexor (Tri-State)

```
<table>
<thead>
<tr>
<th>d0</th>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>S1</th>
<th>S0</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>1</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>0</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A Tri-state Buffer

Truth Table

<table>
<thead>
<tr>
<th>EN</th>
<th>A</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Z</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Z</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Verilog: bufif(Y, A, en)

Tri-state wire must be driven at one point at a time only.

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

Makes a distributed multiplexor
Barrel Shifter

\[ \begin{array}{cccccccc}
\text{d}_0 & \rightarrow & \text{q}_0 \\
\text{d}_1 & \rightarrow & \text{q}_1 \\
\text{d}_2 & \rightarrow & \text{q}_2 \\
\text{d}_3 & \rightarrow & \text{q}_3 \\
\text{d}_4 & \rightarrow & \text{q}_4 \\
\text{d}_5 & \rightarrow & \text{q}_5 \\
\text{d}_6 & \rightarrow & \text{q}_6 \\
\text{d}_7 & \rightarrow & \text{q}_7 \\
\end{array} \]
Open Drain (open collector)

Distributed OR gate.
Leds and Switches
Interfacing

![Circuit Diagram](image-url)

- Light-emitting diodes (LEDs)
- Current limiting resistors
- Pullup resistors
- GND
- VCC
- Switches
**Bistable Revision**

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

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

Making a transparent latch from an RS latch:

Putting two together we get the D-type:

A more optimal circuit:

In this course, we go **upwards** from the D-type towards systems.
Adding a Clock Enable and Synch Reset

Adding a clock enable

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

alternatively

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

Adding a Synchronous Reset

always @(posedge clk) q <= (sr) ? 0: data_in;
A broadside register of $N$ bits is made out of $N$ D-types with a commoned clock input. It can hold $2^N$ different values.
parameter N = 8;
reg [N-1:0] br_q;
always @ (posedge clk) begin
    br_q <= data_in;
end
A broadside two-to-one multiplexor

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

An $n$-bit shifter

Adding a parallel load

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

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

```verilog
reg [7:0] reg1, reg2;
always @(posedge clock) begin
  reg1 <= (g) ? din: reg2;
  reg2 <= reg1;
end
```
A Dual-Port Register File

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

wire [7:0] data_out_a = regfile[read_address_a];
wire [7:0] data_out_b = regfile[read_address_b];

Ex: Draw out the full circuit at the gate level!
Read/Write Memory (RAM)

Read Cycle - Like the ROM
- Read or write mode select
- Enable Input (active low)
- Address In
- Data Bus
- Data In and Out

Valid data

Write Cycle - Data stored internally
- Read or write mode select
- Enable Input (active low)
- Address In
- Data Bus
- Data must be valid here to be stored.

Each data bit internally stored in an RS latch.
The ROM takes A address bits named A0 to A-1 and produces data words of N bits wide. For example, if A=5 and D=8 then the ROM contains 2^5 which is 32 locations of 8 bits each. The address lines are called A0, A1, A2, A3, A4 and the data lines D0, D1, ... D7.

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

MASKED PROGRAMMED means contents inserted at time of manufacture.

FLASH PROM uses static electricity on floating transistor gates.
## Non-volatile Technologies

<table>
<thead>
<tr>
<th>Name</th>
<th>Persistence</th>
<th>Read Speed</th>
<th>Write Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>Volatile</td>
<td>Same as SRAM</td>
<td>Same as SRAM</td>
</tr>
<tr>
<td>BB-RAM</td>
<td>Non-volatile</td>
<td>Same as SRAM</td>
<td>Same as SRAM</td>
</tr>
<tr>
<td>Mask PROM</td>
<td>Non-volatile</td>
<td>Same as SRAM</td>
<td>Not possible</td>
</tr>
<tr>
<td>EPROM</td>
<td>Non-volatile</td>
<td>Same as SRAM</td>
<td>10 us/byte</td>
</tr>
<tr>
<td>Sn-W PROM</td>
<td>Non-volatile</td>
<td>Same as SRAM</td>
<td>10 us/byte</td>
</tr>
<tr>
<td>EAROM</td>
<td>Non-volatile</td>
<td>Same as SRAM</td>
<td>10 us/byte</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Erase Time</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAM</td>
<td>not needed</td>
<td>Battery Life</td>
</tr>
<tr>
<td>BB-RAM</td>
<td>not needed</td>
<td>Needs UV window</td>
</tr>
<tr>
<td>Mask PROM</td>
<td>Not Possible</td>
<td>write cycle limit</td>
</tr>
<tr>
<td>EPROM</td>
<td>20 Mins</td>
<td></td>
</tr>
<tr>
<td>Sn-W PROM</td>
<td>Not possible</td>
<td></td>
</tr>
<tr>
<td>EAROM</td>
<td>100 ms/block</td>
<td></td>
</tr>
</tbody>
</table>
Memory Banks

8 ROM DEVICES
EACH ROM DEVICE IS 32768 BYTES CAPACITY
BANK ORGANISATION
128K locations of 16 bits
Unlike the edge-triggered flip-flop, the transparent latch passes data through in a transparent way when its enable input is high. When its enable input is low, the output stays at the current value.
Synchronous FIFO Memory

![Diagram of a FIFO Queue with inputs WREN, DIN, WRCLK, and outputs HF, EF, FF, RDEN, DOUT, RDCLK]
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 $4^A \times D$. 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 extended period after CAS, thus helping system timing design at high CAS rates.

Refresh Cycle - must happen sufficiently often!

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

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

RC oscillator clock source
Clock multiplication and distribution

Inside the chip

PLL Circuit

1000MHz

Divide 10

External clock input
100 MHz

Outside the chip

Clock distribution H tree

H tree layout

Power-on reset

Supply

Active low
Reset output

Vo

Vi

Ground

C

XP

V

26
Driving a heavy current or high-voltage load

Transistor active area could be 1 square centimeter.
Debouncer circuit for a double-throw switch

Switch

Gnd

+5 Volt supply rail

Pullup Resistors

Output

Bounces

A

B

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

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

assign Z = (Y == 0); assign N = y[7];
An example structure using an ALU and register file.

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

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

Sequential Long Multiplication

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

31
Micro Architecture for a Long Multiplier
Booth’s multiplier

Booth does two bits per clock cycle:

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

Ex: Design a micro-architecture consisting of an ALU and register file to implement Booth. Design the sequencer too.
Example of memory address decode and simple LED and switch interfacing for programmed IO (PIO) to a microprocessor.
A D8/A16 Computer

Data bus (8 bits)

Clock

Reset

Address bus (16 bits)

Register File (including PC)

Control Unit

Execution Unit + ALU

Memory

Static RAM

16 kByte

(Micro-)Processor

Memory Map decoder circuit

Often a ‘PAL’ single chip device.

1 K Byte ROM
Read Only Memory

UART Serial Port

Rs232 Serial Connection

Address bus (16 bits)

A15

A14

A13

A0-13

R/Wb

Enb

RAM_ENABLE_BAR

A0-9

Enb

ROM_ENABLE_BAR

A0-2

R/Wb

Enb

UART_ENABLE_BAR

D0-7

D0-7

D0-7

36
Memory Address Mapping

<table>
<thead>
<tr>
<th>Start</th>
<th>End</th>
<th>Resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>03FF</td>
<td>EPROM</td>
</tr>
<tr>
<td>0400</td>
<td>3FFF</td>
<td>Unused images of EPROM</td>
</tr>
<tr>
<td>4000</td>
<td>7FFF</td>
<td>RAM</td>
</tr>
<tr>
<td>8000</td>
<td>BFFF</td>
<td>Unused</td>
</tr>
<tr>
<td>C000</td>
<td>C001</td>
<td>Registers in the UART</td>
</tr>
<tr>
<td>C002</td>
<td>FFFF</td>
<td>Unused images of the UART</td>
</tr>
</tbody>
</table>

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

    assign rom_cs = (abus == 2’b00); // 0x0000
    assign ram_cs = (abus == 2’b01); // 0x4000
    assign uart_cs = !(abus == 2’b11); // 0xC000
endmodule
```
PC Motherboard, 1997 vintage

PSU

KYBD

COM1

COM2

PRINTER

USB

IDE-1

IDE-2

Floppy

PCI1

PCI2

PCI3

ISA

16 BIT SLOTS

Main memory DRAM

SIMM 4

SIMM 3

SIMM 2

SIMM 1

CACHE RAM

BIOS ROM

IDE & Floppy

Cache

Control

General glue

Pentium CPU

BATTERY

Clock

Regulator

Cache

CTRL
Parallel Port Interface Logic

- Address Data
- Read/Writebar device select
- Strobe bar
- Acknowledge
- Parallel Data

D-25 Parallel (Centronics) Port

Flow control: New data is not sent while the busy wire is high.
Serial Port (UART)

Most computers just use a 9 way connector these days.
Keyboard and/or PS/2 port

Open drain/collector wiring using two signalling wires.

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

- Ethernet
- MAC
- PHY
- TX-DATA
- TX-CLK
- RX-DATA
- RX-CLK
- CS/COL
- RX QUEUE
- TX QUEUE
- RJ45 Socket
- (4 of 8 pins used)
- Processor Bus
  - Address
  - Data
  - IRQ
  - Read/Write
  - device select
  - device select
  - interrupt
  - r/wbar
  - cs

Transformers

42
Canonical Synchronous FSM

FSM = (Set of Inputs, Set of states $Q$, Transition 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.

43
Canonical Logic Array
Combinational Logic Minimisation

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

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

\[(A \& \neg B \& C) + (A \& \neg B) \rightarrow (A \& \neg B)\]

\[(A \& \neg B \& C) + (A \& \neg B \& \neg C) \rightarrow (A \& \neg B)\]

Lookup ‘Kline-McClusky’ for more information.
Karnaugh Maps are convenient to allow the human brain to perform minimisation by pattern recognition.

\[(A \land \neg C) + (A \land B) + (B \land C) \rightarrow (A \land \neg C) + (B \land C)\]

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

\[(A \land \neg C) + (A \land B) + (B \land C) \rightarrow A + (B \land C)\]

Lookup ‘ESPRESSO’ for more information.
Sequential Logic Minimisation

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

A Moore machine can be simplified by the following procedure

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

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

3. The final blocks are the new states.
Timing Specifications

Data in → Clock

Clock

D → Q

Q output

Clock

Data in

Q output

Hold time

Setup time

Propagation delay
Typical Nature of a Critical Path

Clock speed can be increased while margin is positive.
Johnson counters
Pipelining

Data in

Another input

Yet another input

Synchronous global clock signal

Large loop-free combinatorial logic function

An output

Yet another output

Another output still

Desired logic function

Data in

Another input

Yet another input

Synchronous global clock signal

Loop-free combinatorial logic function - first half

An output

Yet another output

Another output still

Loop-free combinatorial logic function - second half

Desired logic function - pipelined version.
Cascading FSMs
An example that uses (badly) a derived clock: a serial-to-parallel converter

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

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

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

Care is needed when gating clocks.
A Gated Clock

OR’ing with a negated enable works cleanly.

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

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

b) System interconnection with clock skews

c) A solution for serious skew and delay problems?
Crossing an Asynchronous Domain Boundary

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 occurrences when the first is metastable for a significant length of time (e.g. 1/2 a clock cycle) the second will present a good clean signal to the rest of the receiving system.

All real systems have many clock domains and frequently implement this style of solution.
Dicing a wafer

(Chips are not always square)
A chip in its package, ready for bond wires

IO and power pads
Die cost example

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

Digital Integrated Circuits

Standard Parts

ASICs

Field Programmable Parts

Full Custom

Semi Custom Standard Cell

e.g. LAN Interface Controller

Rarely Used

Semi Custom Standard Cell

e.g. Toys

Array Logic (PALs)

e.g. 22V10

FPGA

e.g.

Xilinx Spartan

SOC

FPGAs

e.g.

Altera Excalibur

Commodity Parts

e.g. Memories
Field Programmable Gate Arrays
A configurable logic block for a look-up-table based FPGA

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

Seven LUT inputs: \(2^7 = 128\)

The LUT can be a RAM of 128 locations of two bits.
Pictured is a basic I/O block.

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

Clock signal

Output pad (can also be input).

Clock input

General purpose inputs

Product line

Term line

Output enable product line

Ground pin.

The cross points in these shaded regions are programmable points.
Contents of the PAL macrocell

```
Contents of the PAL macrocell

- Input buffer
- Clock Net
- I/O Pad
- Tristate output pad
- Programmable multiplexor
- D-type flip-flop
- Main input S-of-P
- Output enable term
- Feedback to array
```

65
Example programming of a PAL showing only fuses for the top macrocell

```c
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`  
- `xxxx xxxx xxxx xxxx xxxx xxxx xxxx`  

x  

(macrocell fuse)
Delay-power style of technology comparison chart

<table>
<thead>
<tr>
<th>Technology</th>
<th>device</th>
<th>propagation</th>
<th>power</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>1977 CMOS</td>
<td>HEF4011</td>
<td>30 ns</td>
<td>32 mW</td>
<td>960 pJ</td>
</tr>
<tr>
<td>1982 ECL</td>
<td>sp92701</td>
<td>0.8 ns</td>
<td>200 mW</td>
<td>160 pJ</td>
</tr>
<tr>
<td>1983 CMOS</td>
<td>74hc00</td>
<td>7 ns</td>
<td>1 mW</td>
<td>7 pJ</td>
</tr>
<tr>
<td>1983 TTL</td>
<td>74f00</td>
<td>3.4 ns</td>
<td>5 mW</td>
<td>17 pJ</td>
</tr>
<tr>
<td>1996 CMOS</td>
<td>74LVT00</td>
<td>2.7 ns</td>
<td>0.4 mW</td>
<td>1.1 pJ</td>
</tr>
</tbody>
</table>

2-Input NAND gate. 74LVT00 is 3V3. On-chip logic is much faster.
Logic net with tracking and input load capacitances
An example cell from a manufacturer’s cell library

NAND4 Standard Cell

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)

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
<th>O/P Falling</th>
<th>O/P Rising</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>(ps)</td>
<td>ps/LU</td>
</tr>
<tr>
<td>A</td>
<td>F</td>
<td>142</td>
<td>37</td>
</tr>
<tr>
<td>B</td>
<td>F</td>
<td>161</td>
<td>37</td>
</tr>
<tr>
<td>C</td>
<td>F</td>
<td>165</td>
<td>37</td>
</tr>
<tr>
<td>D</td>
<td>F</td>
<td>170</td>
<td>37</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Pin</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input loading</td>
<td>a</td>
<td>2.1</td>
<td>Load units</td>
</tr>
<tr>
<td></td>
<td>b</td>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>c</td>
<td>2.1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d</td>
<td>2.0</td>
<td></td>
</tr>
<tr>
<td>Drive capability</td>
<td>f</td>
<td>35</td>
<td>Load units</td>
</tr>
</tbody>
</table>
Current digital logic technologies

1994 - First 64 Mbit DRAM chip.

- 0.35 micron CMOS
- 1.5 micron² cell size \((64\text{E6} \times 1.5 \text{um}^2 = 96\text{E6})\)
- 170 mm² die size

1999 - Intel Pentium Three

- 0.18 micron line size
- 28 million transistors
- 500-700 MHz clock speed
- 11x12 mm (140 mm²) die size

2003 - Lattice FPGA

- 1.25 million use gate equivs
- 414 Kbits of SRAM
- 200 MHz Clock Speed
- same die size.

See www.icknowledge.com
Design partitioning: The Cambridge Fast Ring

Designed in 1980.

ECL Chip 100 MHz, bit serial.

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

 Introduced 1989-85.

Such a micro-controller has an D8/A16 architecture and would be used in a mouse or smartcard.
In 1980 we used a microcontroller with external DSP components. A Modem.
Design partitioning: A Miniature Radio Module

Introduced 1998.
1998: A Platform Chip: D32/A32
System on a Chip = SoC design.

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

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

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

The bus bridges map part of one processor’s memory map into that of another so that cycles can be executed in the other’s space, 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 peripherhal.’ This would be the one part of the design that varies from product to product. The same core set of components would be used for all sorts of different products, from iPODs, digital cameras or ADSL modems.
LEDs wired in a matrix to reduce external pin count
IR Handset Internal Circuit

- Battery
- Scan multiplexed keyboard
- Clock capacitor
- Single chip containing all semiconductors
- Infrared transmit diodes
Scan multiplex logic for an LED pixel-mapped display

You made one of these in the Ia H/W classes.
Addition of pseudo dual-porting logic

You did this too!
Use of a ROM as a function look-up table

The ROM contains the exact imperfections of a 1950’s valve amplifier.
Use of an SRAM to make the delay required for an echo unit

A to D convertor

D to A convertor

16 bit synchronous counter

Static RAM 65536 by 16 bits

Timing generator circuit

Derived clock, 44.1 kHz

Amplifier

Clock 44.1

Clock 88.2

Read cycle

Write cycle

Read cycle

Counter Output

N-1

N

N+1

RAM data pins

Old sample replay

New sample write

RAMWE

RAMOE

81
Merge unit block diagram

MIDI serial data format

9n kk vv  (note on)
8n kk vv  (note off)
9n kk 00  (note off with zero velocity)
MIDI merge unit internal functional units

Midi In 0
Serial to par

Remove status

FIFO Queue

Meger core function

Midi In 1
Serial to par

Remove status

Queue

Par to serial

Insert running status

Queue

Merged midi output
The serial to parallel converter:

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

The running status remover:

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

For the FIFOs:

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

For the merge core unit:

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

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