

Space-division switch fabrics

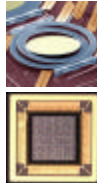
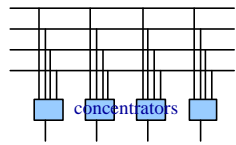
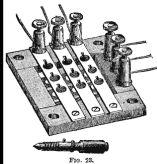
Copyright © 2003, Tim Moors

Outline: Space-division switches

- Single-stage
 - Crossbar, Knockout
- Staged switches: Multiple switching elements between input and output
 - Networks of basic elements
 - Clos
 - Banyan
 - Batcher sorting networks

Copyright © 2003, Tim Moors

Crossbar switch



At the intersection of each input and each output, there is a crosspoint, which can be selectively enabled, allowing communication from input to output.
e.g. Intel 470 switches

Feasible to implement in VLSI (e.g. PMC-Sierra PM9312),
problem is high-speed I/O to chip - pincount

Copyright © 2003, Tim Moors

Sketch from www.cse.cmu.edu/~j4568/impact/contacts/imp73-3.pdf. Photos from Lucent

Assessment of crossbar switches

Advantages:

- Simple structure
- Low latency – minimal number of connecting points between arbitrary input and output.
- No internal blocking (may have output port blocking)
- Readily supports multicast

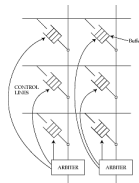
Disadvantages:

- Complexity grows with n^2 , where n is the number of ports
- Fanout: Number of crosspoints on each line increases linearly with number of ports, increasing capacitive loading, slowing transmission
- No fault tolerance: each crosspoint is needed for one connection or another.
- Difficult to expand
- Output buffer speed increases in proportion to fabric, not line.

Copyright © 2003, Tim Moors

Crossbar variations

Internal buffering, e.g. for variable-length packets & output contention



Design of arbiter/concentrator, e.g. Knockout switch...

Picture from Keshav.

Copyright © 2003, Tim Moors

Knockout switch

Crossbar with concentrators modelled on a knockout tournament:

- Competitors (inputs) play and those who win progress to the next round. At the end, one competitor is selected (transferred to output), and all others have lost one match.
- Losers then compete again (clean slate) to determine next competitor to be selected.
- Repeat to capacity of output port.

Copyright © 2003, Tim Moors

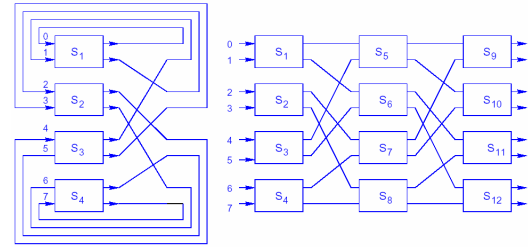
Staged switches

Multistage switches:

- Composed of networks of smaller switches (e.g. crossbars and shared-media), often 2x2
- Selection of switch in first and last stage is determined by which input & output are being connected (⇒ use 3 or more stages to get benefits)
- There may be variety of choice of switch in intermediate stages ⇒
 - Lower blocking probability
 - Increased reliability: can still connect input and output even if a component switch has fail

Copyright © 2003, Tim Moors

Multistage networks: Recirculation vs spatially separate switches

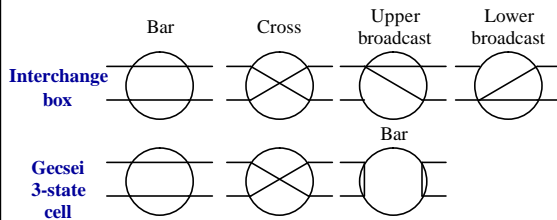


Multistage switches work well as pipelines if each stage takes same time ⇒ Prefer that all units of information have the same length (cells) ⇒ AT

Figure from H. Peyravi

Copyright © 2003, Tim Moors

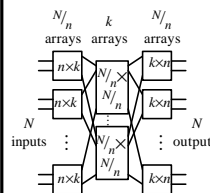
Basic 2x2 switching elements



Copyright © 2003, Tim Moors

Clos switches

Multiple stages of small switches (e.g. crossbars)
Each switch of a stage has one output feeding into each switch of the next stage (rectangular, not square)
Fewer crosspoints, path diversity, chance of blocking



Number of crosspoints:

$$N_x = 2N(2n-1) + (2n-1) \left(\frac{N}{n}\right)^2$$

$$N_x(\min) = 4N(\sqrt{2N-1})$$

Ports $N_x(3\text{-stage Clos})$ $N_x(\text{Cross})$

128	7,680	16,356
32768	33M	1B

Copyright © 2003, Tim Moors

Clos switches

Used in some commercial products:

- NEC ATOM ATM switch
- Myrinet switch

Clos switches provide multiple paths; c.f. Banyan



C. Clos: "A study of non-blocking switching networks", *Bell Sys. Tech. J.*, 32(3):406-24, Mar. 1953

Copyright © 2003, Tim Moors

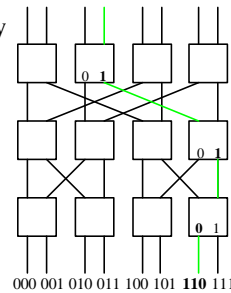
Banyan switches

Self/source-routing using binary representation of output port ⇒ can't multicast

n -port switch has $\log_2 n$ stages each with $n/2$ 2x2 switches

Direction for each stage

specified by bit corresponding to that stage (MSb 1st)



1 ⇒ switch to right output port of switch in this stage
0 ⇒ switch to left output port

Copyright © 2003, Tim Moors

Blocking in Banyan switches

In the worst case, half of the traffic is blocked (when increasing input port number destined to decreasing output port number)

Destinations:
100 110 111 101 010 011 001 000

Output port numbers
000 001 010 011 100 101 110 111

Copyright © 2003, Tim Moors

Dealing with Banyan internal blocking

Buffering within the switching network

Dilation:

- Multiple planes of latter stages
- Move traffic blocked in one plane to higher plane (R→G→B)

Sorting networks

Copyright © 2003, Tim Moors

Sorting networks

Sorting is essentially the same as switching, except need to spread output after sorting to account for idle ports

Input port: a b c d e f g h
 Destination: 101_a --_b 111_c 001_d 010_e 110_f --_g 011_h
 Sorted: 001_d 010_e 011_h 101_a 110_f 111_c --_g --_x
 Switched: --_x 001_d 010_e 011_h --_x 101_a 110_f 111_c

Contention for port 101 (between inputs a & d)

Destination: 101_a --_b 111_c 101_d 010_e 110_f --_g 011_h
 Sorted: 010_e 011_h 101_a 101_d 110_f 111_c --_g --_x
 Switched: --_x --_x 010_e 011_h --_x 101_a 110_f 111_c

Copyright © 2003, Tim Moors

Batcher sorting networks

Simpler to sort half sets & merge, than to sort whole.

Merging: Take multiple sorted lists & merge them to form a single, larger, sorted list.
 e.g. {1, 2, 5, 7} + {0, 3, 4, 6} → {0, 1, 2, 3, 4, 5, 6, 7}

Merge sort: Split items into groups. Sort each group (e.g. recursively, by using merge sort). Merge groups.

K. E. Batcher: "Sorting networks and their applications", Proc. AFIPS Spring Joint Computer Conference, pp. 307-14, 1968
Copyright © 2003, Tim Moors

Sorting and Merging bitonic list

bitonic list

1	6	7	7	7	7	8
6	1	6	3	3	5	7
2	2	2	6	6	6	6
7	7	1	4	4	8	5
8	8	3	2	5	3	4
4	4	4	5	2	2	3
5	3	5	1	8	4	2
3	5	8	8	1	1	1

Numerical example from C. Partridge: Gigabit Networks
Copyright © 2003, Tim Moors

Trap modules

Sorting alone doesn't resolve output port contention ⇒ trap duplicates

Starlite switch: **Recirculate** duplicates

Resequencing is needed. Often give priority to recirculated cells to avoid prolonged resequencing.

Copyright © 2003, Tim Moors