

Introduction to Systolic Design

Wayne Luk
wl@doc.ic.ac.uk
Imperial College
March 2002

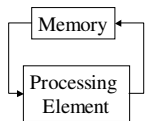
wl 3/2002 1

Overview

- history and motivation for systolic arrays
- systolic array features
- systolic design techniques
 - composing regular components
 - retiming
 - slowdown
 - clustering
 - bit-level design
 - systolic state machines
- topics not covered
- further reading

wl 3/2002 2

History and motivation



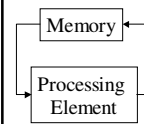
- introduced by Kung and Leiserson, 1978
- designs for matrix computations
- illustrated by snapshots of operation

motivations:

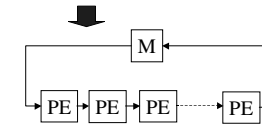
- improve performance of special-purpose systems
 - e.g. maximise processing per memory access
- reduce their design and implementation costs
 - e.g. exploit latest technology: FPGAs

wl 3/2002 3

Systolic arrays



Systolic: rhythmical contraction; describes the contraction of the heart forcing blood onward and keeping up the circulation.



Array: multiple PEs to maximise processing per memory access.

wl 3/2002 4

Systolic array features

- multiple use of each input data item
- extensive concurrency; usually by pipelining
- a few types of simple cells
- simple and regular data and control flow

these result in:

- simple and reduced costs
- high performance
- modular and expandable

wl 3/2002 5

Field-Programmable Gate Arrays (FPGAs)

- combine software flexibility and hardware performance
- off-the-shelf parts, factory-tested, many varieties
- matrix of cells, each has programmable function unit
- programmable connections
 - nearest neighbour / local / global routing
- technology: 10 million-gate FPGA, GHz clock speed
- good platform for implementing systolic designs
 - array structure
 - increased flexibility, adaptable at run time
 - reduced design/implementation time and cost

wl 3/2002 6

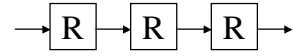
Applications

- signal, image, video, multimedia, numerical processing
 - add, multiply, divide, square root...in various number systems
 - recursive and non-recursive, linear and non-linear filtering
 - DFT, FFT, FHT, DCT, DWT, FNT
 - matrix and graph algorithms, algebraic path problem
 - neural nets, motion estimation, shading, texture mapping
- non-numerical processing
 - sorting, searching, matching, priority queue, LRU
 - dynamic programming
 - data compression and encryption
 - discrete event simulation
 - database operations

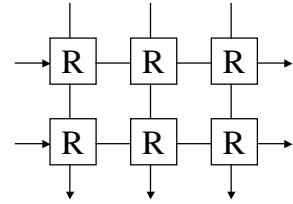
wl 3/2002 7

Array shapes: linear and rectangular

Linear array: chain

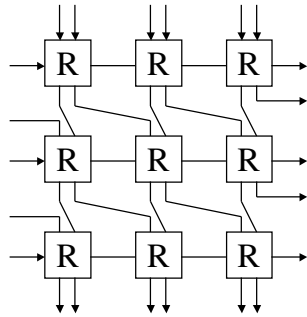


Rectangular array



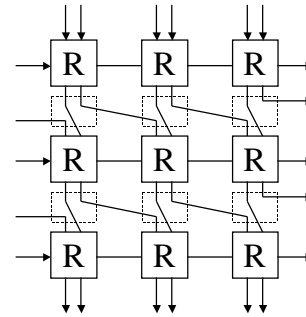
wl 3/2002 8

Hexagonal array



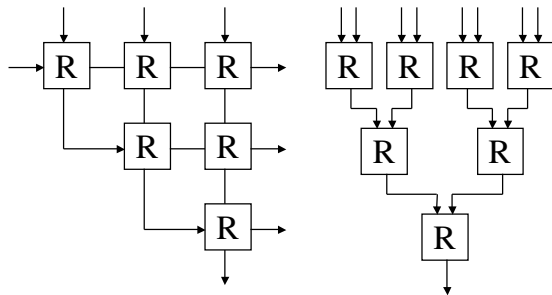
wl 3/2002 9

Hexagonal array



wl 3/2002 10

Triangular-shaped arrays



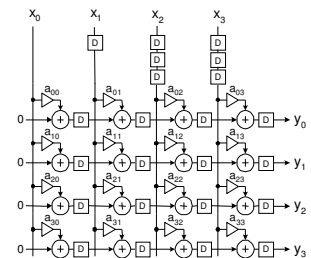
wl 3/2002 11

Example: matrix vector multiplier

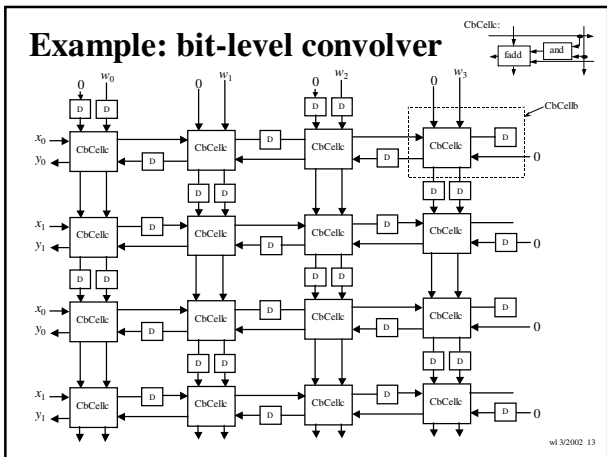
- $\mathbf{Ax}=\mathbf{y}$, $y_i = a_{i0} x_0 + a_{i1} x_1 + a_{i2} x_2 + a_{i3} x_3$
- D=delay=register
- constant multiplier:

$$x_i \rightarrow \triangle_{a_{ij}} a_{ij} x_i$$

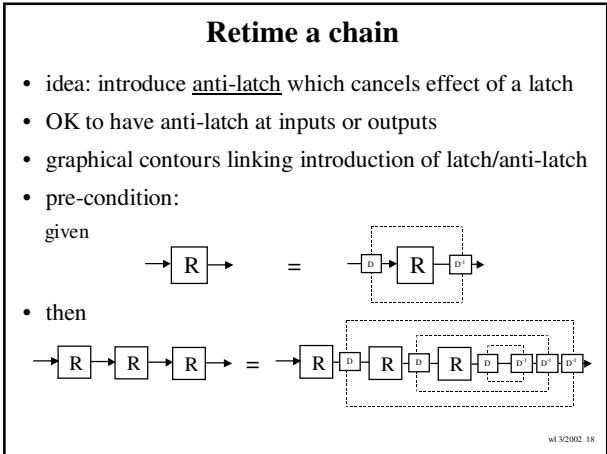
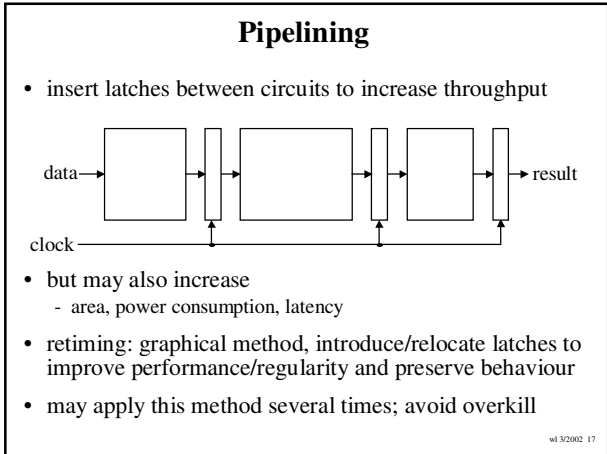
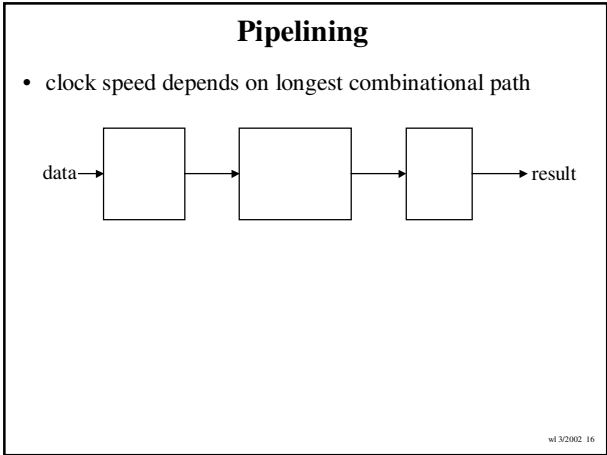
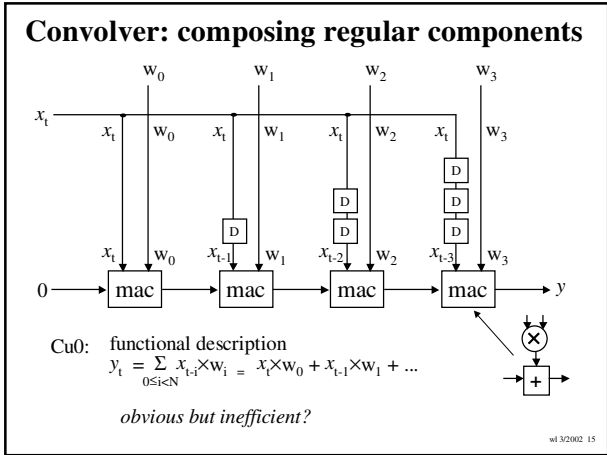
- does it work?
- are there alternatives?

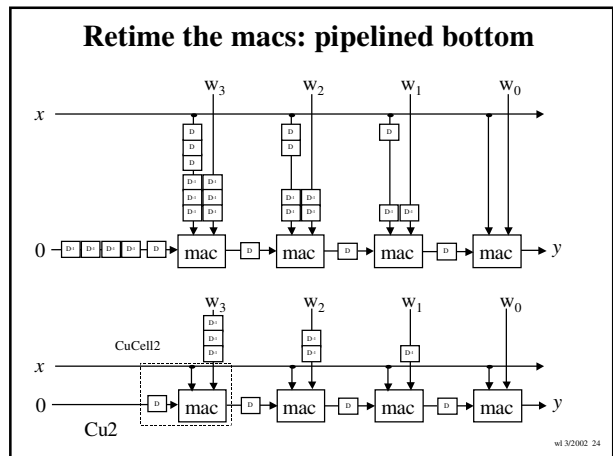
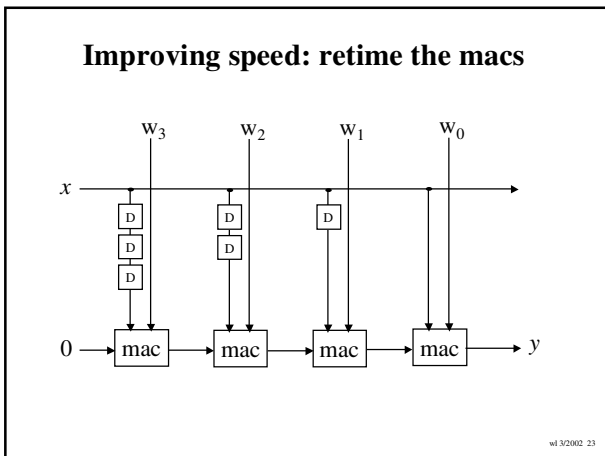
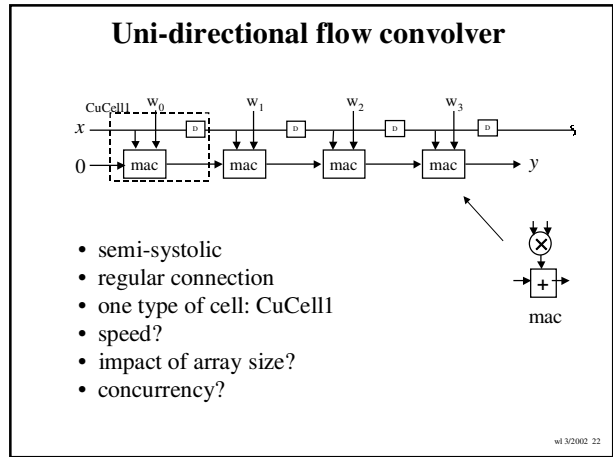
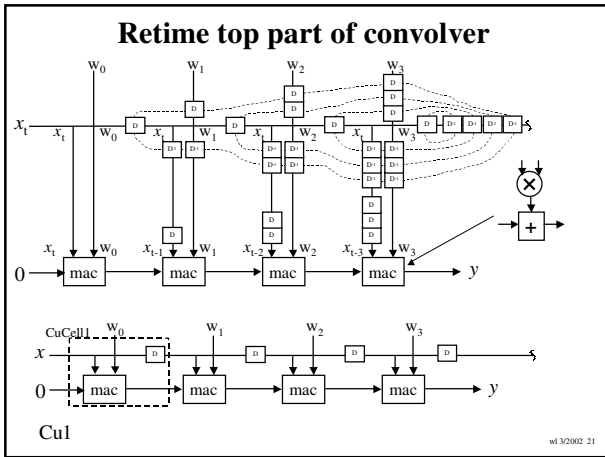
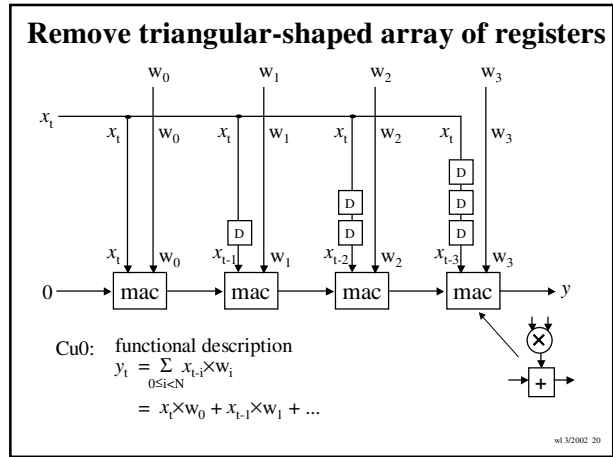
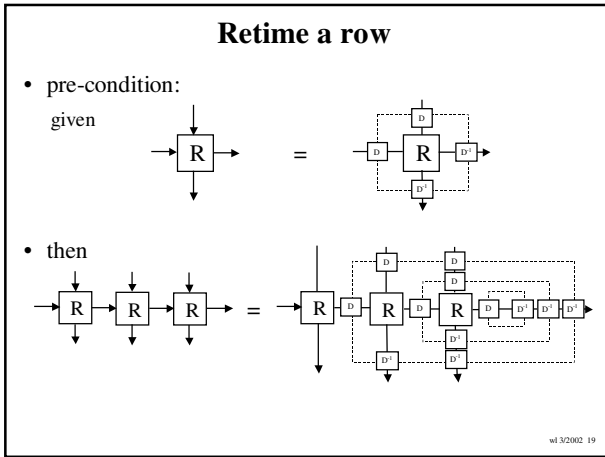


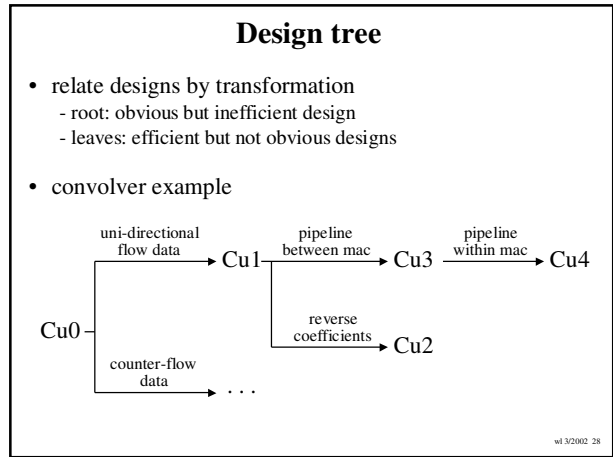
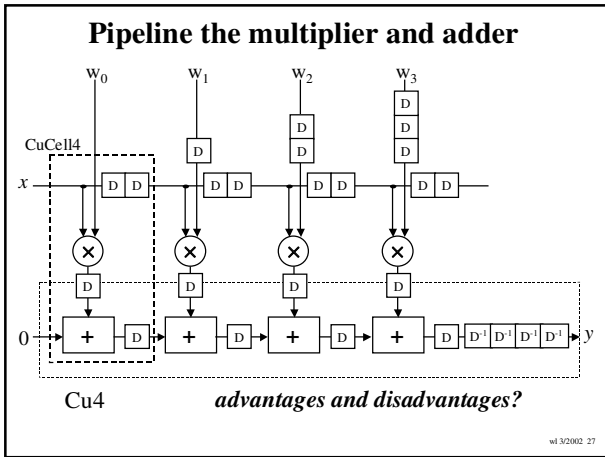
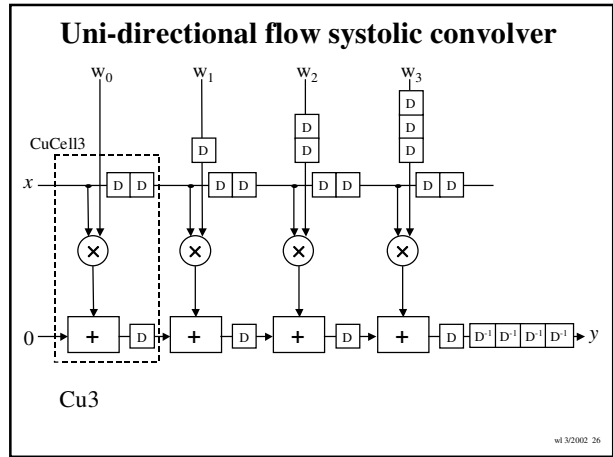
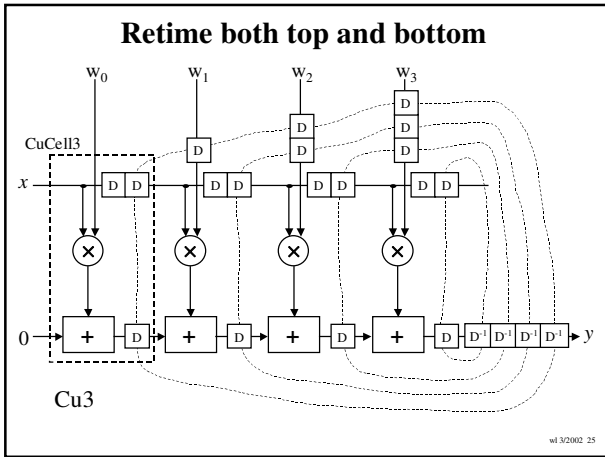
wl 3/2002 12



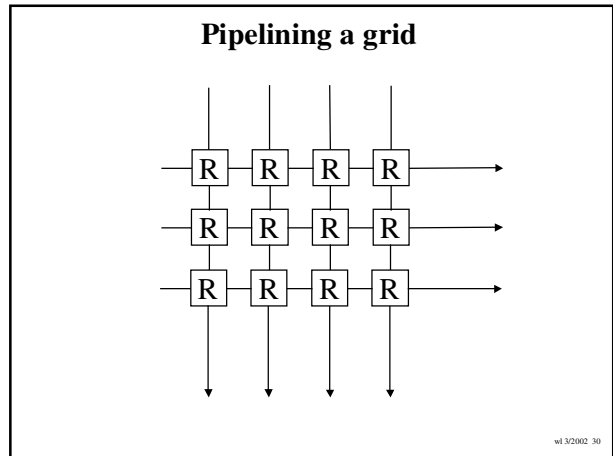
- ### Systolic design techniques
- systematic design of systolic arrays
 - transform obvious design to efficient but less obvious designs
 - circuit-oriented block diagram approach
 - simple ideas behind design automation algorithms
 - techniques
 - composing regular components: focus on desired behaviour
 - retiming: relocate latches in a circuit
 - slowdown: replicate latches
 - clustering: arrange hierarchy for pipelining
 - bit-level design: useful for hardware libraries
 - systolic state machines: pipeline state transition functions
 - illustrated by simple examples
 - convolution, matrix vector multiplication, sorting
- wt 3/2002 14



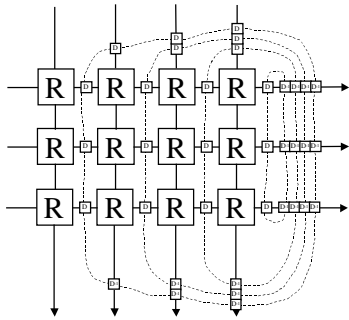




- ### Characterise designs
- express features of a composite design in terms of the number and features of its components
 - number of cells and registers: impact on size and power consumption and latency
 - latency: determined by the path from input to output which has the maximum number of registers
 - critical path: determined by the path from input to output which has the largest combinational delay
 - e.g. Cu1: N latches, $N-1$ cycles of latency, $T_{mult} + NT_{add}$ critical path
 - assumptions: negligible effect of wires and word-length growth
- wl 3/2002 29

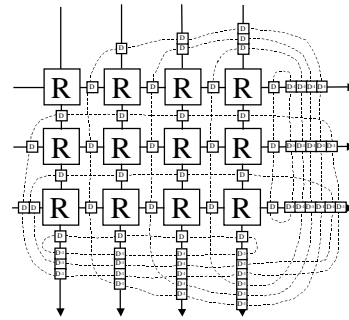


Pipelining a grid



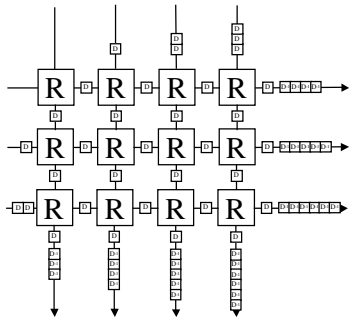
wl 3/2002 31

Pipelining a grid



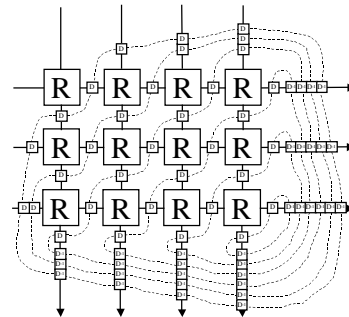
wl 3/2002 32

Pipelining a grid



wl 3/2002 33

Pipelining a grid

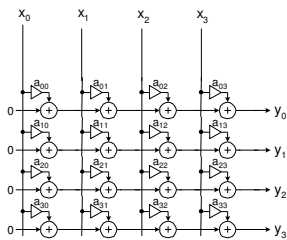


wl 3/2002 34

Combinational matrix vector multiplier

- $\mathbf{Ax}=\mathbf{y}$, $y_i = a_{i0} x_0 + a_{i1} x_1 + a_{i2} x_2 + a_{i3} x_3$
- unpipelined
- constant multiplier:

$$x_i \rightarrow \triangle_{a_{ij}} a_{ij} x_i$$

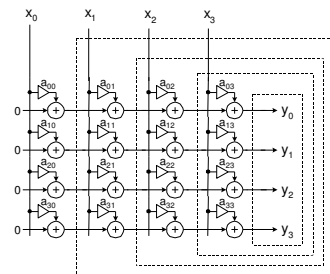


wl 3/2002 35

Matrix vector multiplier: contours

- $\mathbf{Ax}=\mathbf{y}$, $y_i = a_{i0} x_0 + a_{i1} x_1 + a_{i2} x_2 + a_{i3} x_3$
- constant multiplier:

$$x_i \rightarrow \triangle_{a_{ij}} a_{ij} x_i$$

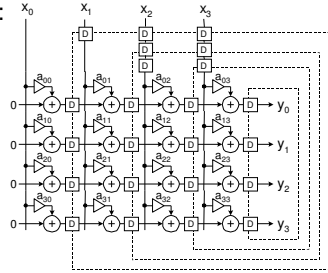


wl 3/2002 36

Matrix vector multiplier: adding registers

- $Ax=y$
- D =delay=register
- constant multiplier:

$$x_i \rightarrow a_{ij} \quad a_{ij} x_i$$

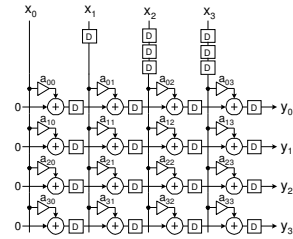


wl 3/2002 37

Pipelined matrix vector multiplier

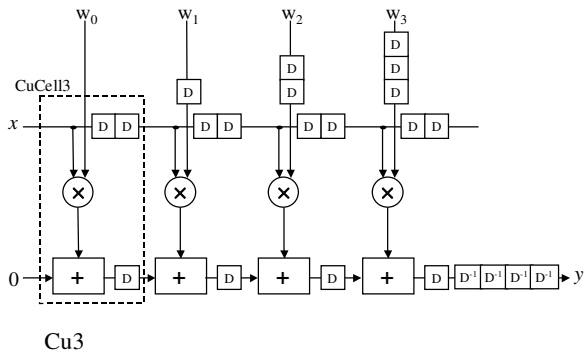
- $Ax=y$
- D =delay=register
- constant multiplier:

$$x_i \rightarrow a_{ij} \quad a_{ij} x_i$$



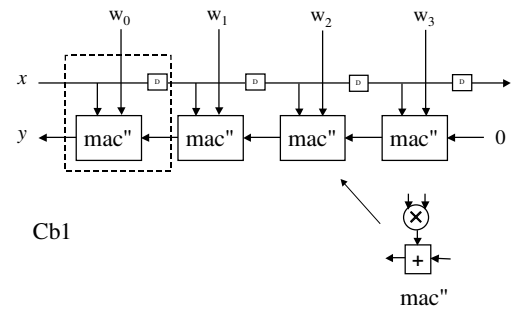
wl 3/2002 38

Uni-directional flow systolic convolver



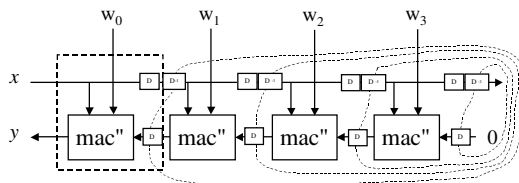
wl 3/2002 39

Convolver with counter-flowing data



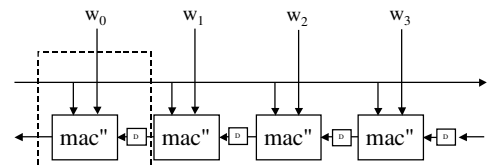
wl 3/2002 40

Convolver with counter-flowing data



wl 3/2002 41

Convolver with counter-flowing data



Cb2
still not fully pipelined!

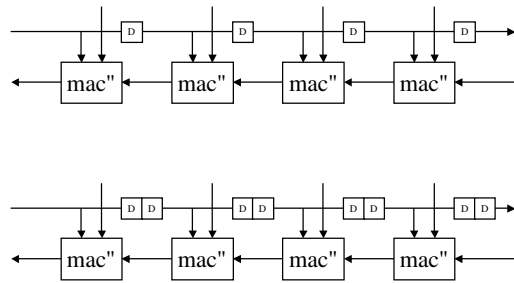
wl 3/2002 42

Slowdown

- n-slow: can replace each latch by n latches in series, provided that (n-1) extra values are inserted between successive inputs; similarly for outputs
- graphically: introduce additional D or D⁻¹ by replacing each D (or D⁻¹) by n copies in series
- interpretation:
 - interleaved n data streams/computations concurrently
 - sample output every n cycles to get result of each computation

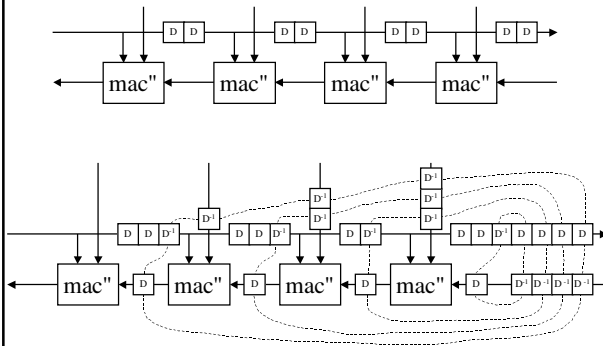
wl 3/2002 43

Derive fully-pipelined convolver: slowdown



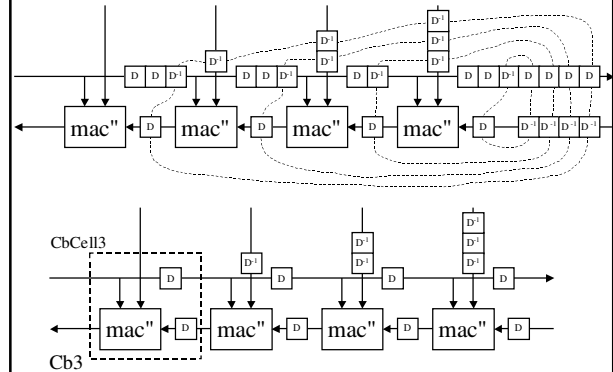
wl 3/2002 44

Retime after slowdown



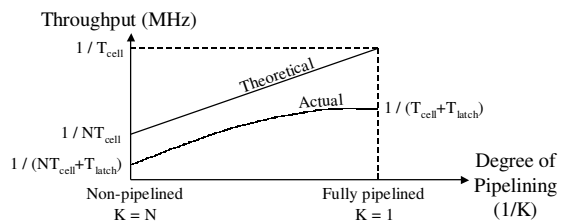
wl 3/2002 45

Fully-pipelined convolver



wl 3/2002 46

Pipelining may become less effective

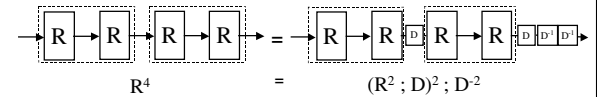


- input-output speed limit
- clock skew
- clock rise and fall times significant
- * control degree of pipelining

wl 3/2002 47

Controlled pipelining: clustering

- cluster elements into groups, and retime the groups



- vary size of groups to control degree of pipelining
 - ↑ size of each group
 - ↓ degree of pipelining

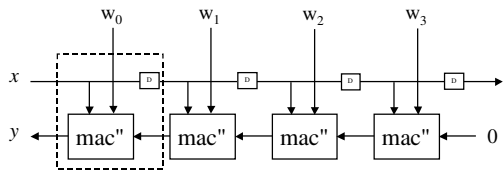
- reason about regular patterns of pipelining

$$R^{KN} = (R^K; D)^N; D^{-N} \quad (\text{given } R = D^{-1}; R; D)$$

KN = M, fully-pipelined : K = 1, N = M
non-pipelined : K = M, N = 1

wl 3/2002 48

Convolver with counter-flowing data

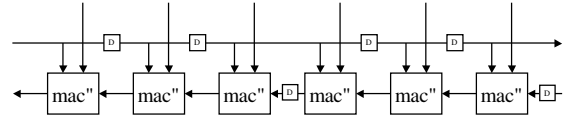


Cb1

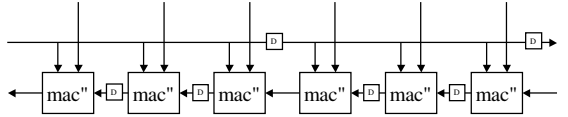
wl 3/2002 49

Partially-pipelined designs

- Cb4



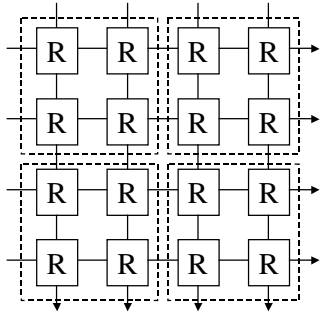
- Cb5



- boundary conditions not shown

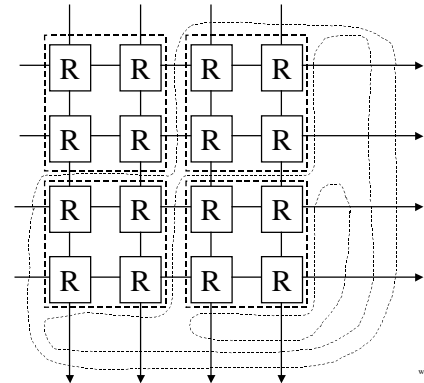
wl 3/2002 50

Clustering rectangular array



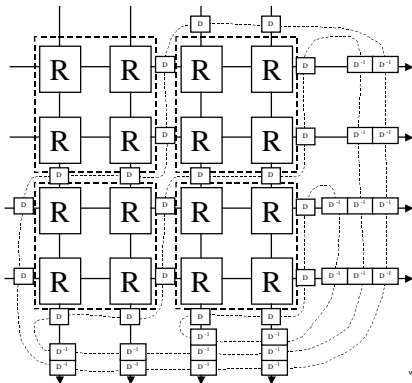
wl 3/2002 51

Retime around the contours



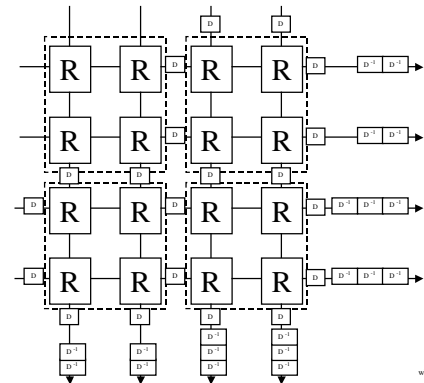
wl 3/2002 52

Result

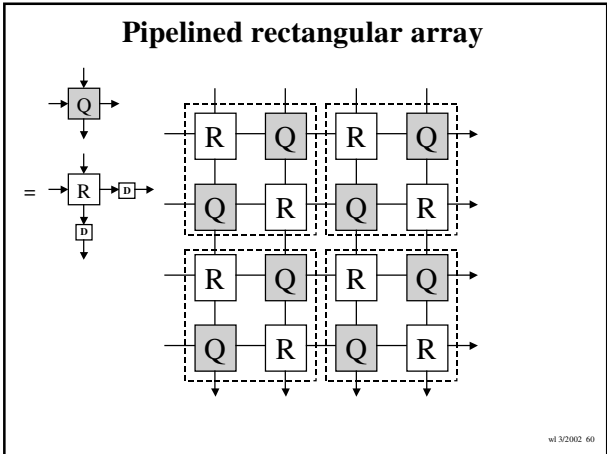
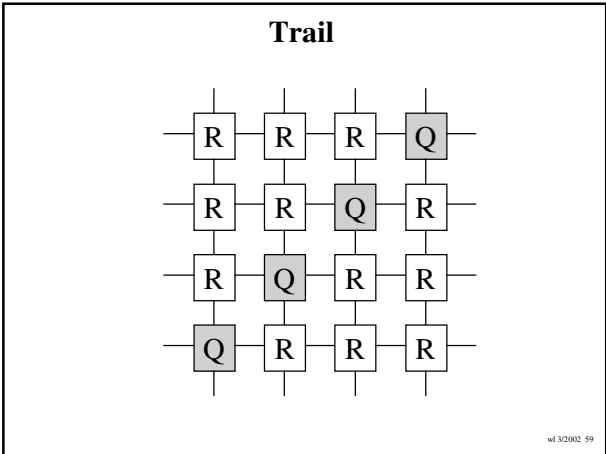
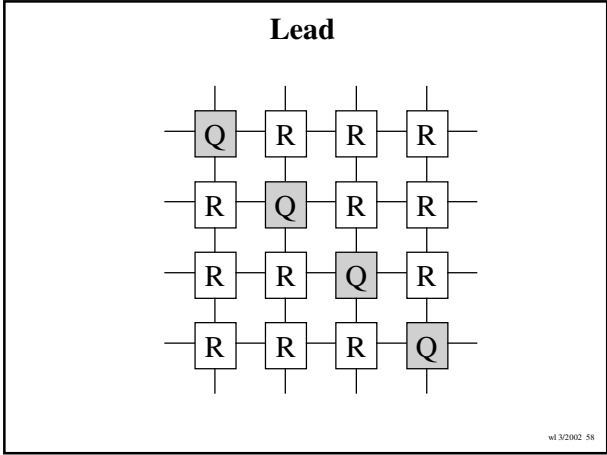
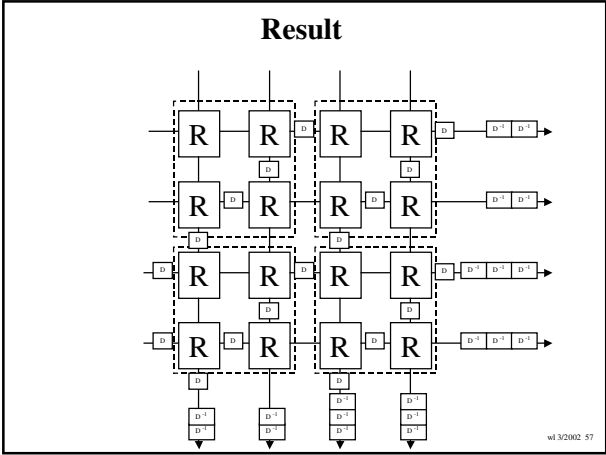
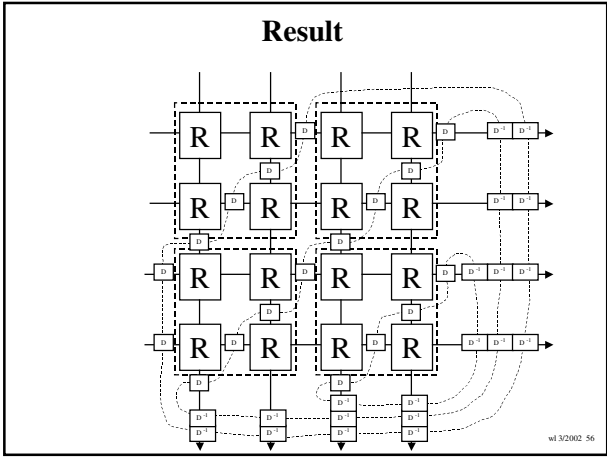
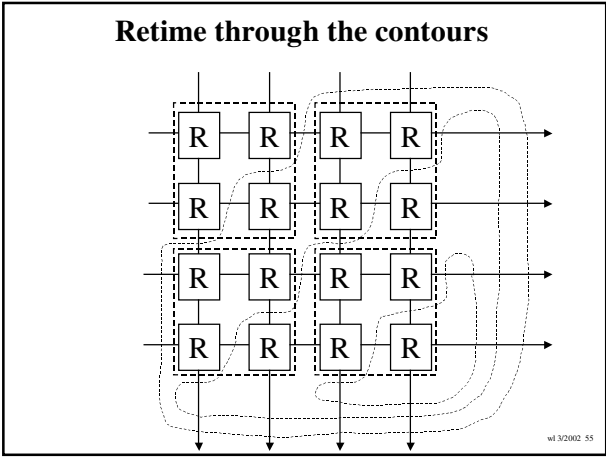


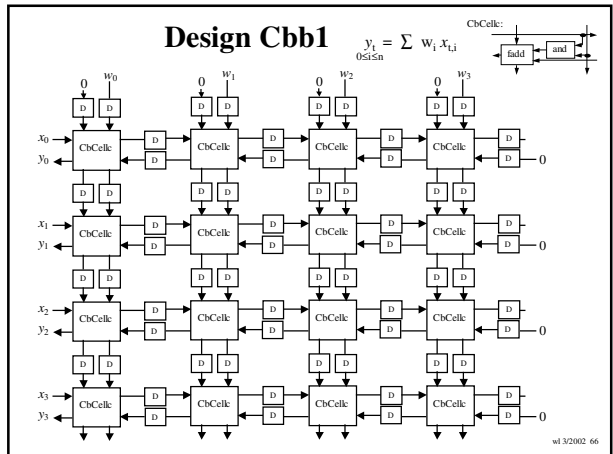
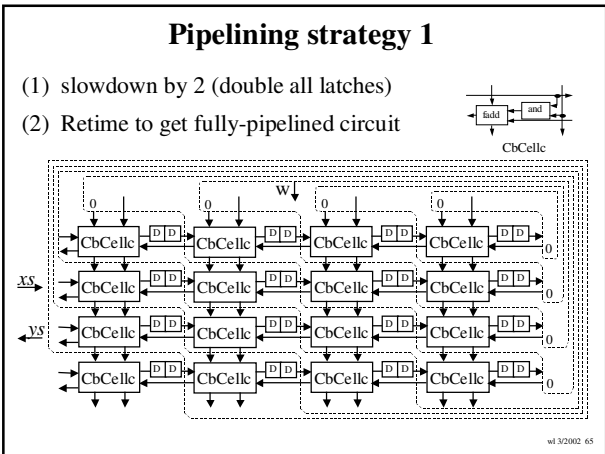
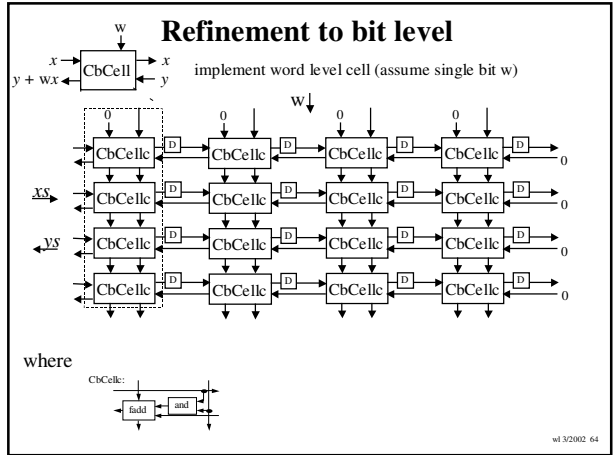
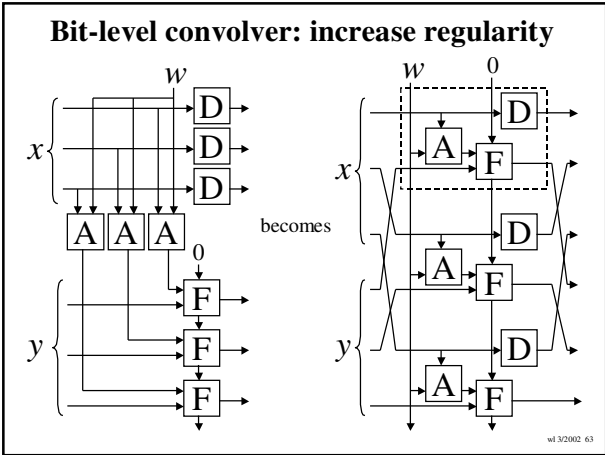
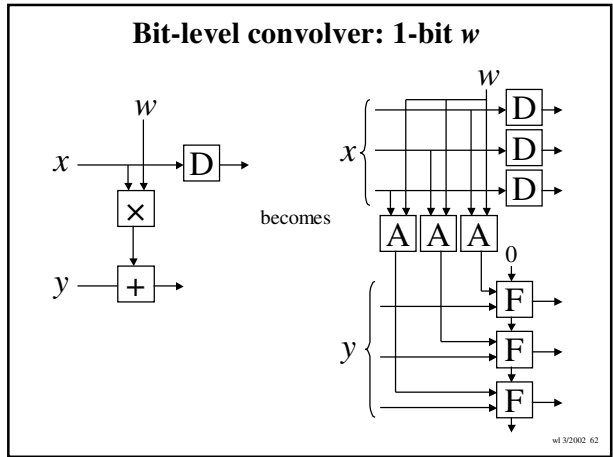
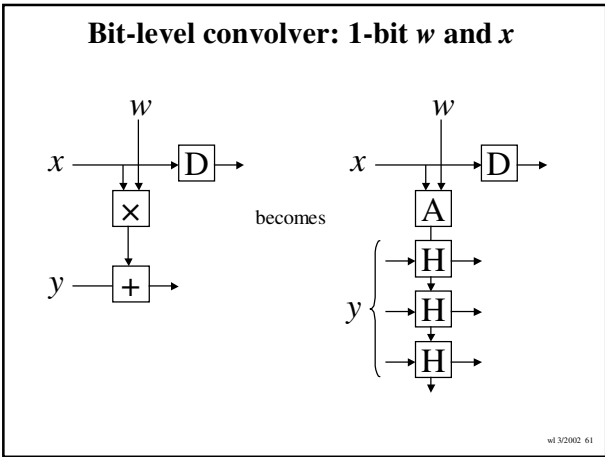
wl 3/2002 53

Result



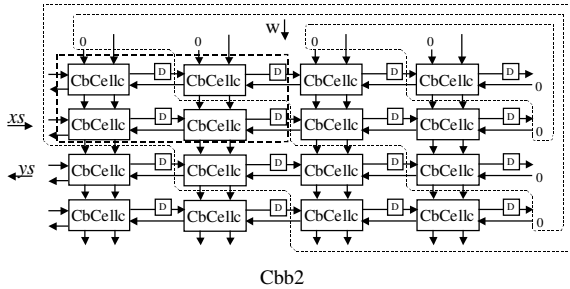
wl 3/2002 54





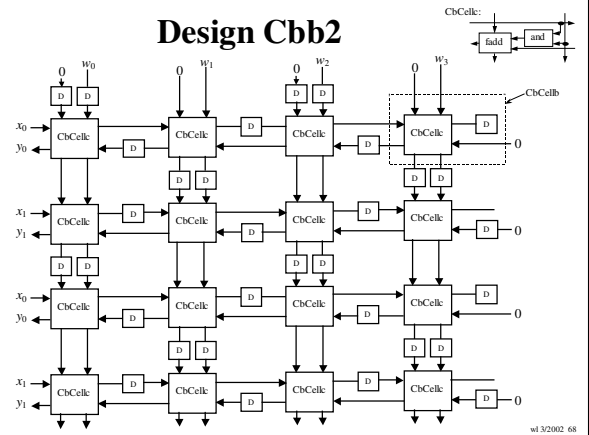
Pipelining strategy 2

- pipeline clusters of K by K cells ($K > 1$) e.g. $K = 2$



wl 3/2002 67

Design Cbb2



wl 3/2002 68

Summary: optimising digital designs

useful rules:

- retiming: can add a latch at all inputs, provided adding an anti-latch at all outputs



- use clustering to control degree of pipelining
- n-slow: can replace each latch by n latches in series, provided that (n-1) additional values are inserted between successive inputs; similarly for outputs

wl 3/2002 69

Performance and resource usage

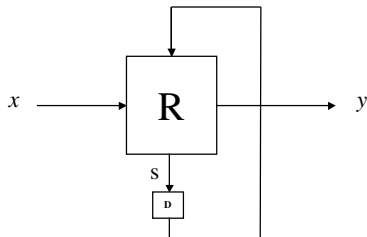
Design	min clock period	latency	number of latches
Cu2	$T_m + T_a$	$N-1$	$N(N+1) / 2$
Cu3	$T_m + T_a$	$2N-1$	$N(N+5) / 2$
Cu4	$\max(T_m, T_a)$	$2N$	$N(N+7) / 2$

Note that the minimum clock period for Cu2 should be $(N-1) T_p + T_m + T_a$, where T_p is the delay across the wiring cell

wl 3/2002 70

State machines

- state-transition function R usually includes an output part y and a next state part s



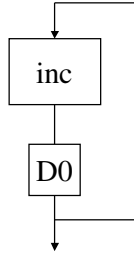
wl 3/2002 71

Simple state machines

- counter
- sorter
- priority queue
- LRU processor

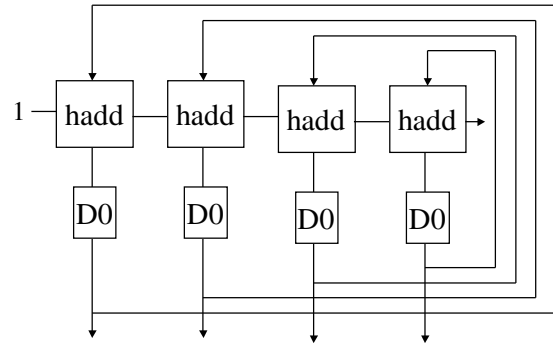
wl 3/2002 72

Simple state machines



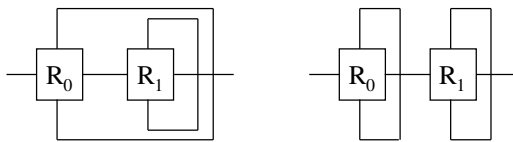
wl 3/2002 73

Simple state machines



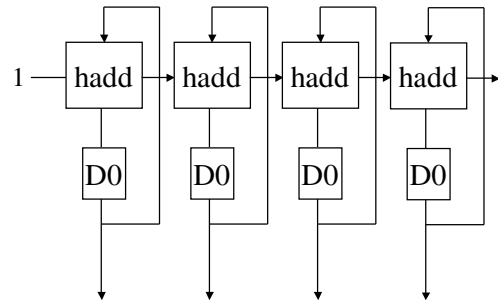
wl 3/2002 74

Decomposing state machines



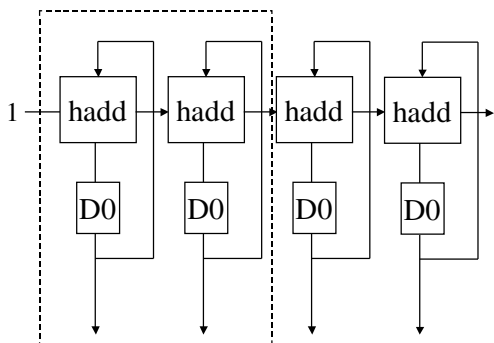
wl 3/2002 75

Simple state machines



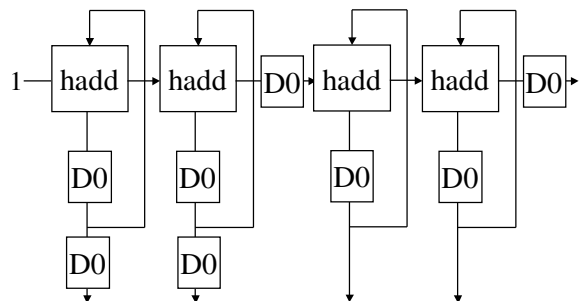
wl 3/2002 76

Simple state machines



wl 3/2002 77

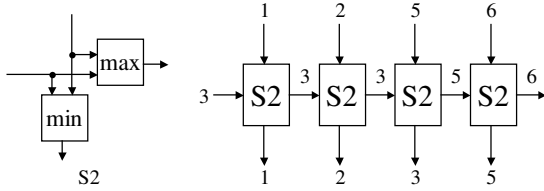
Simple state machines



wl 3/2002 78

Example: inserter

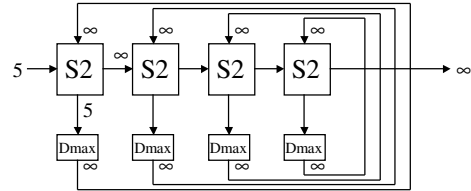
- insert an element into an ordered list to form an ordered list: insert $\langle 3, \langle 1, 2, 5, 6 \rangle \rangle = \langle \langle 1, 2, 3, 5 \rangle, 6 \rangle$



wl 3/2002 79

Insertion sort

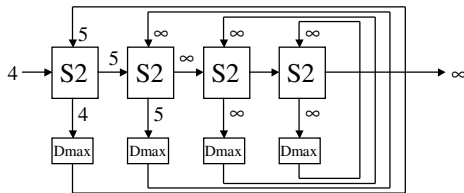
- state registers initialised with $+\infty$
- load n values to be sorted cycle by cycle
- load $n -\infty$ values to extract the sorted result



wl 3/2002 80

Insertion sort

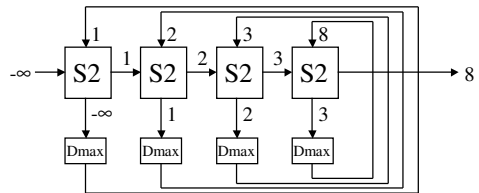
- state registers initialised with $+\infty$
- load n values to be sorted cycle by cycle
- load $n -\infty$ values to extract the sorted result



wl 3/2002 81

Insertion sort (cont)

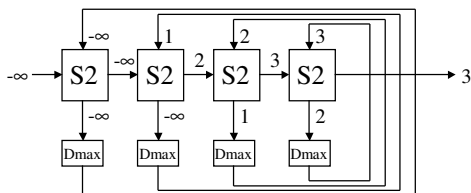
- to extract the sorted sequence cycle by cycle, input $-\infty$



wl 3/2002 82

Insertion sort (cont)

- to extract the sorted sequence cycle by cycle, input $-\infty$

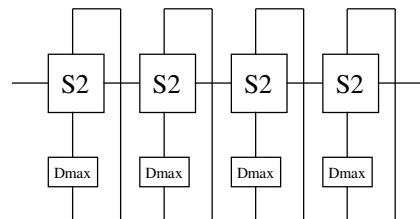


Q1: can we avoid reloading the $-\infty$'s (and $+\infty$'s)?

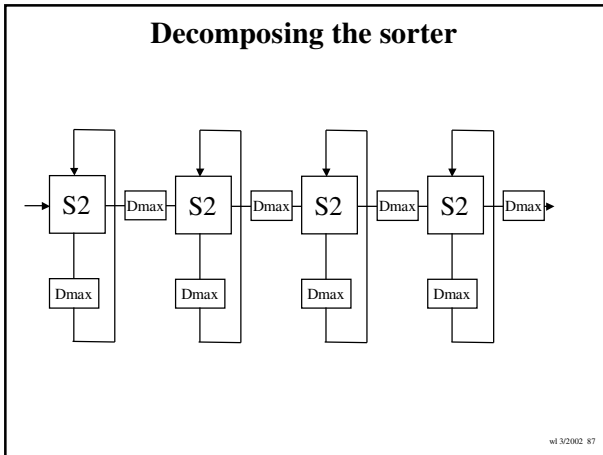
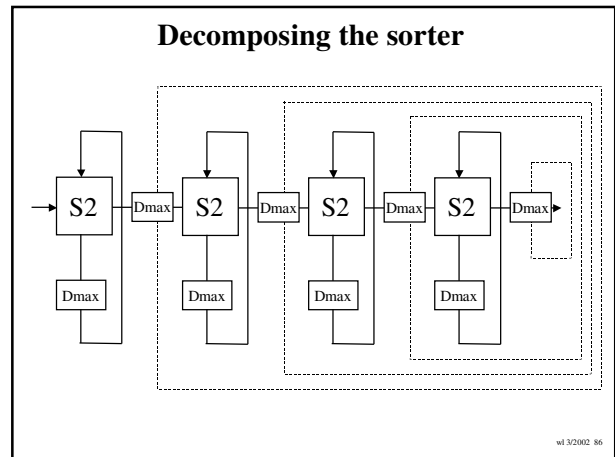
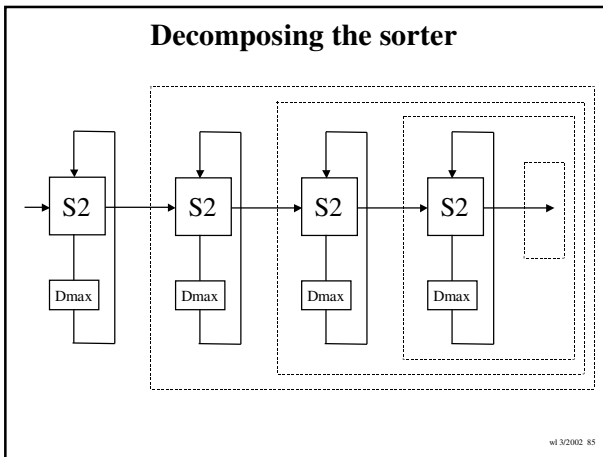
Q2: can we reduce the combinational delay through S2s?

wl 3/2002 83

Decomposing the sorter



wl 3/2002 84



- ### Summary: systolic state machines
- start with state-transition function
 - include loop and state registers to ensure computing the desired function
 - make sure that registers are initialised appropriately
 - decompose a large state machine into a collection of small state machines
 - pipeline the collection of state machines as required
- wl 3/2002 88

- ### Topics not covered
- composite and hybrid systolic systems: ensure boundary conditions match
 - multi-dimensional arrays: nearest-neighbour connections in 3D
 - reconfigurable designs: pipeline morphing and virtual pipelines
 - space optimisation techniques: digit serial
 - systolic array implementations and platforms: iWarp, Splash, Pilchard, RC1000, and Sonic
 - languages and tools for systolic design and verification: Ruby, Pebble, Lava, CSP, CirCal, Alpha
- wl 3/2002 89

- ### Further reading
- H. T. Kung. "Why Systolic Architectures?," IEEE Computer, 15(1):37-46, January 1982.
- *excellent introductions to systolic architectures*
 - S.Y. Kung, VLSI Array Processors, Prentice Hall, 1988.
- *comprehensive reference textbook*
 - Proceedings of IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP). This conference began as the International Workshop on Systolic Arrays, 1986.
- *research on theory and practice of systolic design*
 - Proceedings of IEEE International Conference on Field-Programmable Custom Computing Machines.
- *research on systolic systems implemented by FPGAs*
- wl 3/2002 90