Application Specific Integrated Circuits for Digital Signal Processing
Lecture 5

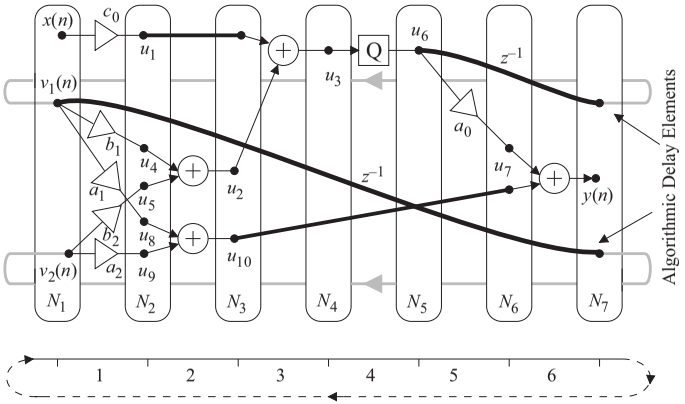
Oscar Gustafsson

Today's topic

- Computation graphs
- Latency and execution time
- Iteration period bound and critical path
- Pipelining and retiming
- Algorithm transformations
- Unfolding

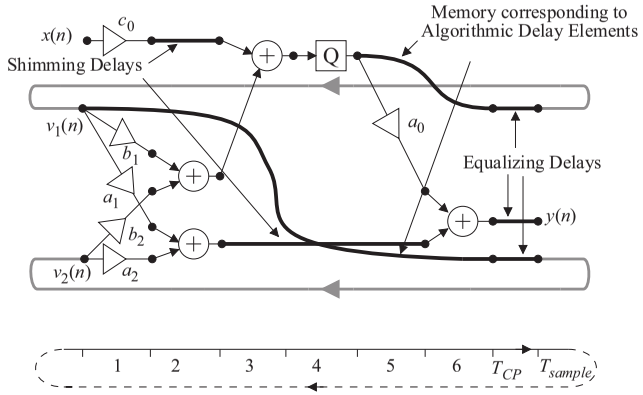
Computation graphs

- Precedence graphs only consider the order of operations, not the actual time it takes to finish an operation



Computation graphs

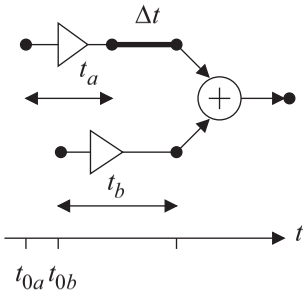
- Introduce time to get a *computation graph*



- The time may be different for different operations

Shimming delay

- When input data to an operation is available at different times *shimming delays* are needed



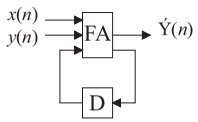
- Note that we want to reuse the operators, so keeping the value at the output is not an option

Latency and execution time

- Two time measures of interest
  - Time from input of sample to output of sample
  - Time between input sample and input of next sample
- Latency
  - Time to produce an output from the corresponding input
  - Denoted T_L
  - Of interest when determining the timing between operations
- Execution time
  - Time between successive input samples
  - Denoted T_{exe}
  - Of interest when we determine how many operators are required
- Often convenient to express the latency and execution time in number of clock cycles
- Combinatorial latency adds to the clock period

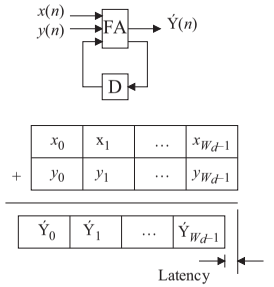
Relation latency and execution time?

- Is it possible to say anything about the relation between latency and execution time?

$T_L > T_{exe}$	$T_L = T_{exe}$	$T_L < T_{exe}$
Pipelined combinatorial	Combinatorial	Serial
		
Several clock cycles	No clock cycle	No or a few clock cycles

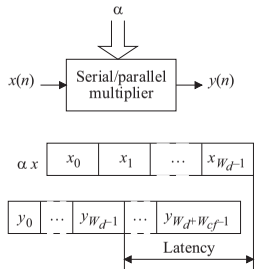
Serial arithmetic

- LSB first
- Latency of addition: 0 clock cycles ($T_{L,FA}$)



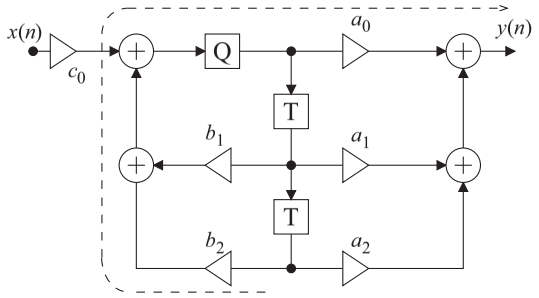
Serial arithmetic

- Latency of multiplication: W_f clock cycles
  - Data: $X = \sum_{i=0}^{W_d-1} x_i 2^{-i}$
  - Coefficient: $\alpha = \sum_{j=0}^{W_f} a_j 2^{-j}$
  - Result: $Y = X\alpha = \sum_{i=0}^{W_d-1} x_i 2^{-i} \sum_{j=0}^{W_f} a_j 2^{-j} = \sum_{k=0}^{W_f+W_d-1} y_k 2^{-k}$
  - First bit at the output of multiplier has weight $W_f + W_d - 1$
  - First bit of interest is of weight $W_d - 1$
  - Have to wait for W_f clock cycles (bits)



Critical path

- Longest time directed path is critical path
- T_{cp} = time of critical path



$$T_{cp} = 3T_{L,add} + T_{L,Q} + T_{L,mult_{b_2}} + T_{L,mult_{a_0}} \qquad (1)$$

Iteration period bound

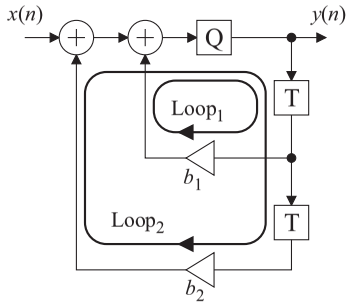
- Time for one iteration limited by recursive parts
- All computations in a loop must finish before the next iteration can start
- Iteration period bound, T_∞, (minimum sample period, T_{min})

$$T_\infty = \max_i \frac{\sum_{\text{Op. in loop } i} T_{L,k}}{N_i} = \frac{1}{f_{max}} \qquad (2)$$

  where N_i is the number of delay elements in loop i
- Processing K samples per iteration, the minimum sample period is $T_{min} = T_\infty / K$
- The loop that determines the sample period is called *critical loop*
- Theoretical bound that puts a limit on how fast the algorithm can be run
- Not always straightforward to find an implementation that runs as fast

Maximum sample rate – example

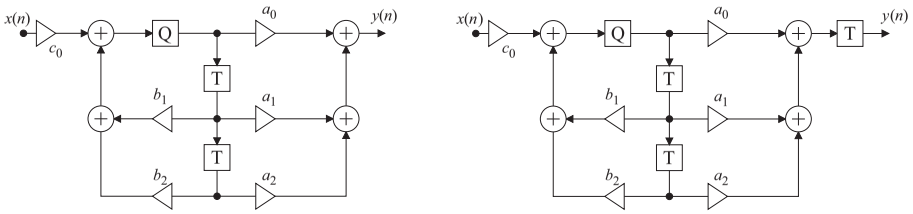
- SFG below has two loops



$$T_{min} = \max\left\{ \underbrace{\frac{T_{L,add} + T_{L,Q} + T_{L,mult_{b_1}}}{1}}_{Loop_1}, \underbrace{\frac{2T_{L,add} + T_{L,Q} + T_{L,mult_{b_2}}}{2}}_{Loop_2} \right\} \qquad (3)$$

Pipelining/retiming

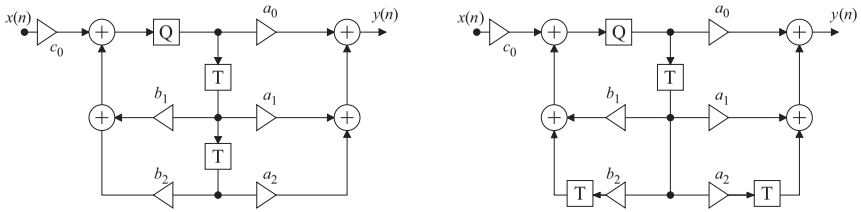
- Introduce delay elements at input or output – pipelining



- Changes the latency of the algorithm

Pipelining/retiming

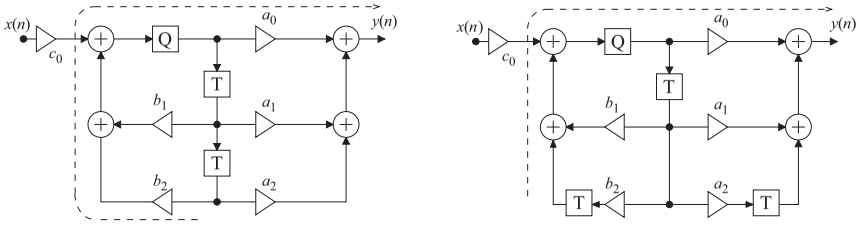
- Move delay elements – retiming



- Must have delay elements on all inputs (outputs) to move to outputs (inputs)

Pipelining/retiming

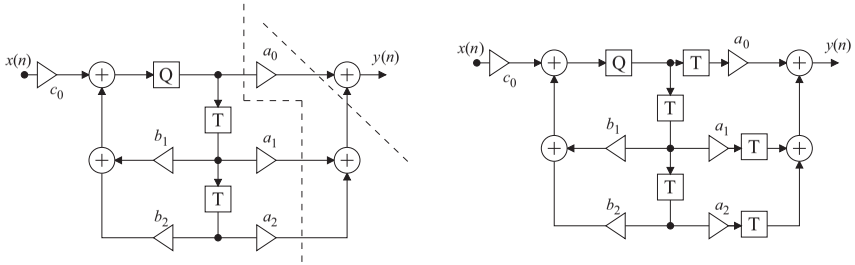
- May change critical path



- May NOT change the iteration period bound
- Never possible to move delay elements into/out of a loop

Cut-set pipelining

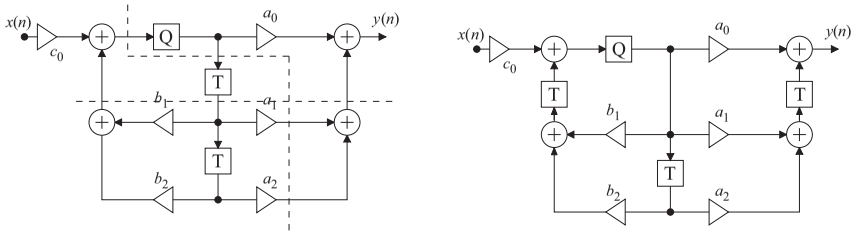
- Find a cut such that all signals go in the same direction



- Possible to add an arbitrary number of delay elements at the cut

Cut-set retiming

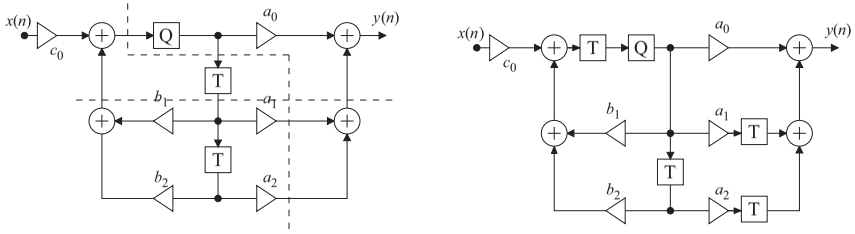
- A cut where signals go in opposite directions



- Possible to move an arbitrary number of delay elements from one direction to the other
- Must move the same number of delay elements from all signals going in one direction

Cut-set retiming

- May accidentally introduce pipelining (delay elements between the input and the output)

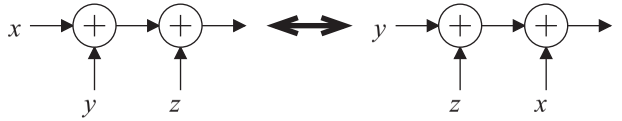


Maximum sample rate – some important notes

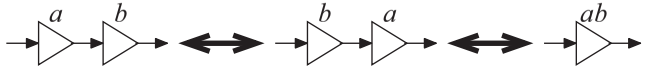
- Recursive algorithms have a bound on achievable sample rate caused by the loops
- Loops can never be pipelined
- Non-recursive (parts of) algorithms can always be pipelined
- No upper bound on achievable sample rate for non-recursive algorithms

Algorithm transformations

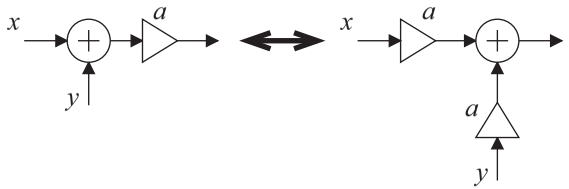
- Still some opportunities to modify the algorithm to, e.g., possibly reduce the sample period
- Associativity: $(x + y) + z = x + (y + z)$



- Commutativity: $ab = ba$



- Distributivity: $a(x + y) = ax + ay$

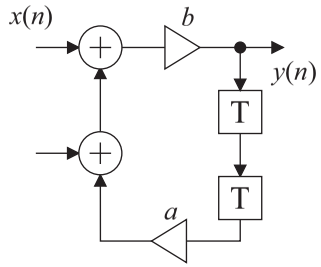


Algorithm transformations

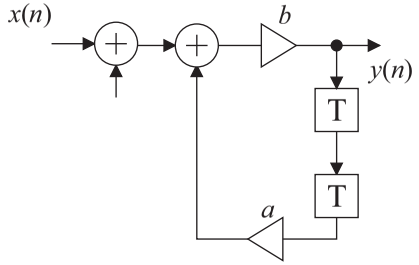
- Possible to obtain one loop with one multiplication and one addition
- May increase operations in other loops
- Ignores finite-wordlength effects
  - Merging multiplications leads to longer coefficient wordlength
  - Re-quantizing the coefficient may lead to an incorrect algorithm

Algorithm transformations – example

- Initial algorithm

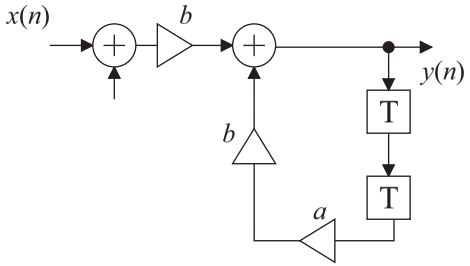


- Move addition out of loop

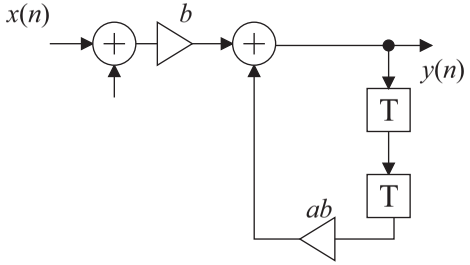


Algorithm transformations – example

- Propagate multiplication with b

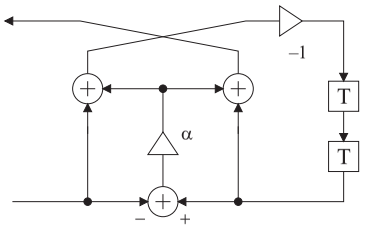


- Merge multiplications



Timing in non-time-multiplexed architectures

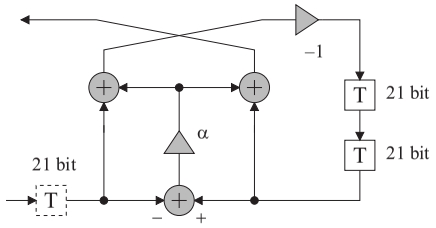
- For architectures where no time-multiplexing of operators is used, it is still important to introduce shimming delays when the operators have a non-zero clock cycle latency
- Introduce registers/flip-flops to synchronize the signals taking the latency of the operations into consideration
- Example – bit-serial second-order bireciprocal LWDF allpass section



  - Use pipelining register after each operation to increase clock frequency
    - $T_{L,add} = 1$ clock cycle
    - $T_{L,mult} = 9$ clock cycles
  - Sample time of 21 clock cycles

Timing in non-time-multiplexed architectures

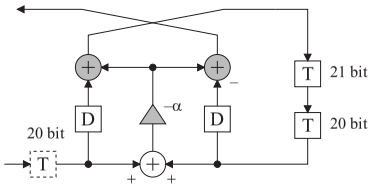
- Replace delay elements with 21 registers/flip-flops



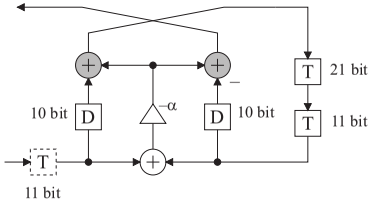
- Each operation needs a number of registers corresponding to the latency
- Move negation to avoid an additional operator

Timing in non-time-multiplexed architectures

- Move one register into the first adder (marked with white)

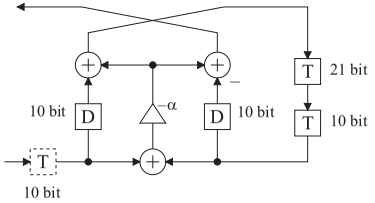


- Move nine register into the multiplier

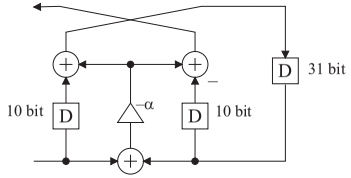


Timing in non-time-multiplexed architectures

- Move one register into the final adders



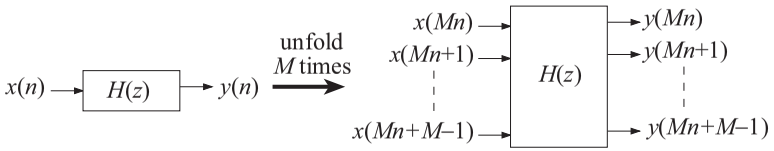
- Final realization



- All loops should have a latency that is an integer multiple of the sample period
- All paths from input to output should have a latency that is an integer multiple of the sample period (plus some constant)

Unfolding

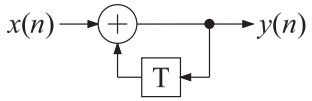
- Express algorithm over several sample periods



- Increase probability of applying algorithm transformations
- Express multi-rate algorithms as single-rate

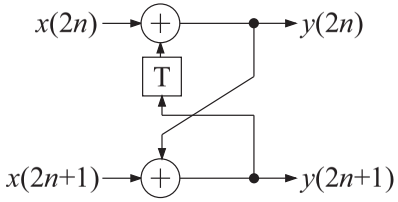
Unfolding – motivational example 1

- Consider a simple accumulator



$$T_{min} = \frac{T_{L,add}}{1} \tag{4}$$

- Express accumulator over two sample periods

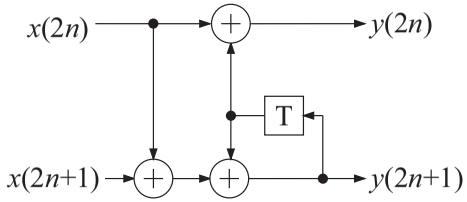


$$T_{min} = \frac{2T_{L,add}}{1} \tag{5}$$

but computing two samples per iteration

Unfolding – motivational example 1

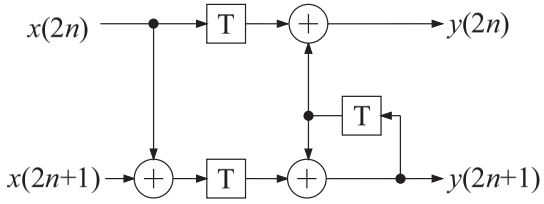
- Re-order additions



$$T_{min} = \frac{T_{L,add}}{1} \tag{6}$$

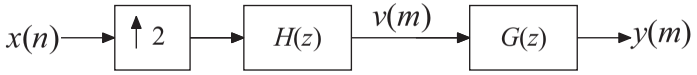
still computing two samples per iteration

- Non-recursive parts can be pipelined

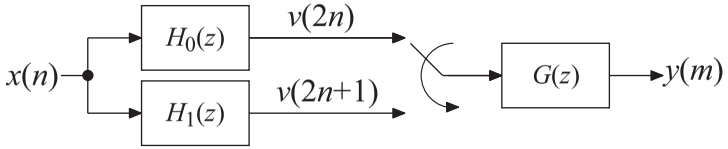


Unfolding – motivational example 2

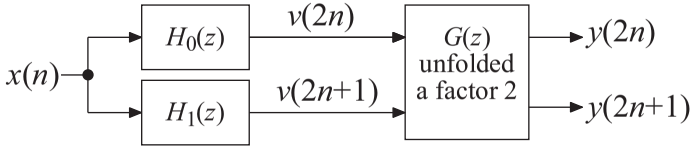
- Consider a multi-rate algorithm according to below



- $H(z)$ can be poly-phase decomposed but $G(z)$ can not

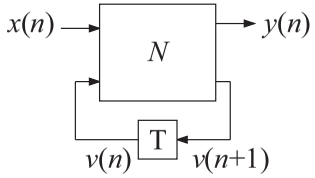


- Unfold $G(z)$ and obtain a single-rate formulation

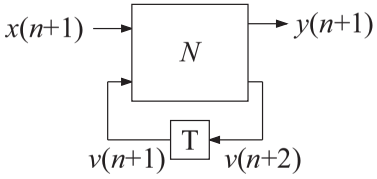


Unfolding – derivation

- Consider an algorithm separated in operations, N, and delay element

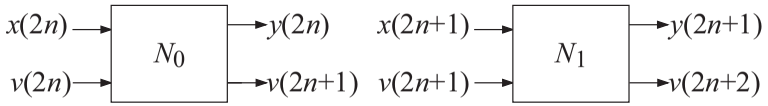


- In the next iteration the sample indices are as follows

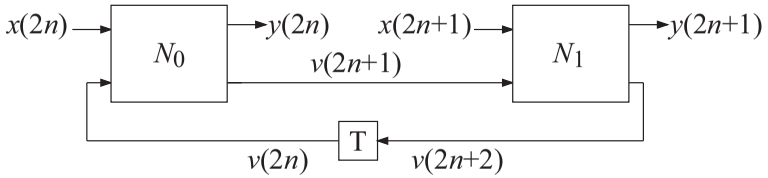


Unfolding – derivation

- Looking at two iterations, the following samples are used



- One iteration updates the induces a factor of two
- Connect according to indices



Unfolding – systematic equations

- Unfolding with aq factor M can be performed systematically as follows
  - There are M computational blocks, denoted $N_i, i = 0, 1, 2, \ldots, M-1$
  - The input and output to block N_i are $x(Mn+i)$ and $y(Mn+i)$, respectively
  - An internal state from block N_i with L delay elements is connected to block N_j with K delay elements where

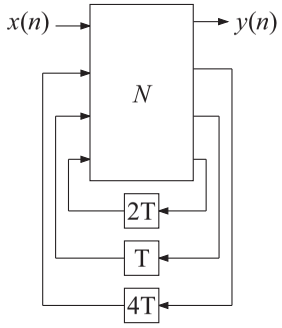
$$j = (i + L) \bmod M \tag{7}$$

and

$$K = \left\lfloor \frac{i+L}{M} \right\rfloor \tag{8}$$

Unfolding – example 1

- Unfold the following SFG $M = 3$ times



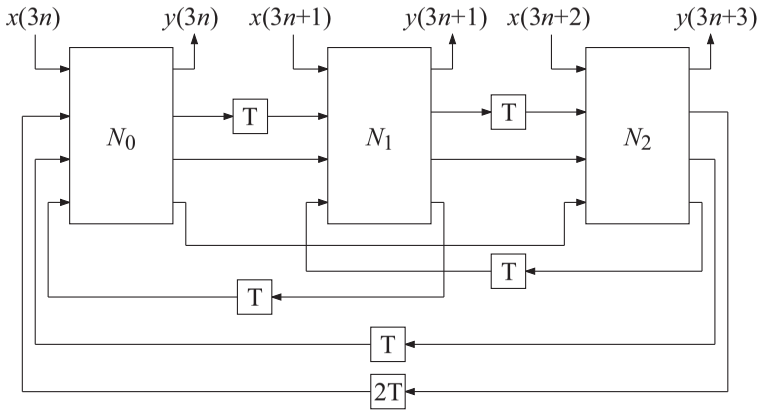
Unfolding – example 1

- Compute connections

i	L	j	K
0	1	$(0+1) \bmod 3 = 1$	$\left\lfloor \frac{0+1}{3} \right\rfloor = 0$
0	2	$(0+2) \bmod 3 = 2$	$\left\lfloor \frac{0+2}{3} \right\rfloor = 0$
0	4	$(0+4) \bmod 3 = 1$	$\left\lfloor \frac{0+4}{3} \right\rfloor = 1$
1	1	2	0
1	2	0	1
1	4	2	1
2	1	0	1
2	2	1	1
2	4	0	2

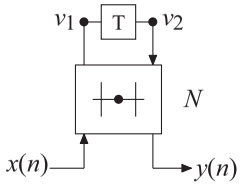
Unfolding – example 1

- Resulting structure

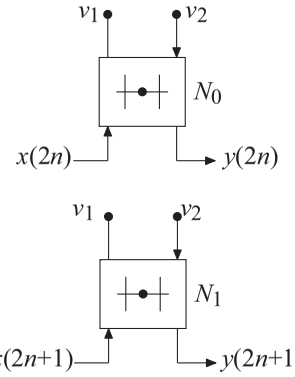


Unfolding – example 2

- Unfold first-order WDF allpass structure a factor $M = 2$



- Two copies

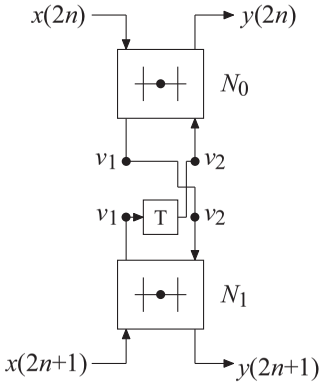


Unfolding – example 2

- Connections

i	L	j	K
0	1	1	0
1	1	0	1

- Resulting structure



Unfolding and iteration period bound

- What happens to the iteration period bound?
- Every loop have M times more operations or M times fewer delay elements $\rightarrow$ iteration period bound increases a factor of M
- Process M samples per iteration
- Resulting minimal sample rate is the same
- Unfolding does not affect the parallelism of a recursive algorithms
  - More operations may lead to better algorithm transformations
  - May increase the resource utilization (next lectures)