Maximal Sample Rate

The maximal sample rate of an algorithm is determined only by its recursive parts.

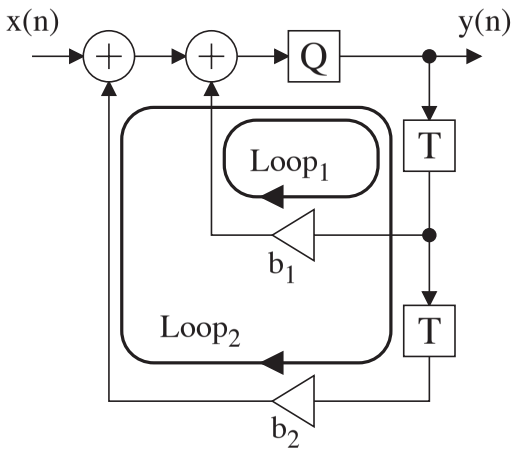
The minimal sample period for a recursive algorithm that is described by a fully specified signal-flow graph is

$$T_{min} = \max_i \left\{ \frac{T_{opi}}{N_i} \right\}$$

where T_{opi} is the total latency of the arithmetic operations, etc. and N_i is the number of delay elements in the directed loop i.

Nonrecursive parts of the signal-flow graph, e.g., input and output branches, generally do not limit the sample rate, but to achieve this limit additional delay elements may have to be introduced into the nonrecursive branches.

The minimal sample period is also referred to as the *iteration period bound*.



Loops that yield T_{min} are called *critical loops*.

The signal-flow graph has two loops. Loop 1 is the critical loop if

$$\frac{T_{b2} + 2T_{add} + T_Q}{2} < \frac{T_{b1} + T_{add} + T_q}{1}$$

otherwise loop 2 is the critical loop.

The iteration period bound is of course not possible to improve for a given algorithm. However, a new algorithm with a higher bound can often be derived form the original algorithm.

We recognize that there are two possibilities to improve the bound.

- Reduce the operation latency in the critical loop.
- Introduce additional delay elements into the loop

Example 6.5

The 3rd-order LWDF has been implemented by Siemens using so-called redundant, bit-parallel arithmetic in which long carry propagations are avoided. The adaptor coefficient is $\alpha = 0.375 = (0.011)_2$. The results obtained using a 2-μm, double metal CMOS process were

Input data word length, 8 bit

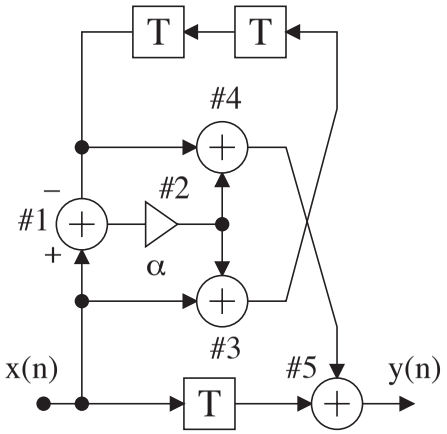
Output data word length, 11 bit

Maximal sample rate, ≈ 35 MHz

Number of devices, ≈ 9000

Power consumption, 150 mW

Chip area, 14.7 mm^2 (including pads)



The filter is a lowpass filter but it can also be used as a highpass filter, as well as for decimation or interpolation of the sample frequency by a factor two, with minor changes in the hardware.

OUR IMPLEMENTATION

Assume here that the data word length is 12 bit and that the filter is implemented with bit-serial arithmetic using so-called *true single-phase logic* (*TSPC*).

This type of logic is modeled by model 1, i.e., the latencies for addition, subtraction, and multiplication in this logic style are one clock cycle longer than the theoretical minimum.

Thus, we have $T_{add} = 1$ clock cycle and $T_{mult} = 4$ clock cycles.

The delay (latency) due to the arithmetic operations in the recursive loop is

$$T_{op} = 2T_{add} + T_{mult} = 6 \text{ clock cycles}$$

and the number of delay elements in the loop is $N_i = 2$. Hence, we get the iteration period bound

$$T_{min} = \frac{6}{2} = 3 \text{ clock cycles}$$

$$f_{samplemax} = \frac{1}{T_{min}} = \frac{f_{CL}}{3}$$

The bit-serial implementation must therefore have a clock frequency of at least 105 MHz to be as fast as the implementation using bit-parallel, redundant arithmetic. This represents a relatively modest clock frequency.

ALTERNATIVE IMPLEMENTATION

Using a static logic style, i.e., model 0, having minimum latencies T_{add} = 0 clock cycle and T_{mult} = 3 clock cycles, we get

$$T_{min} = \frac{2T_{add} + T_{mult}}{2} = \frac{2 \cdot 0 + 3}{2} = 1.5 \text{ clock cycles/sample}$$

Note that the length of the clock cycles are different between the two models.

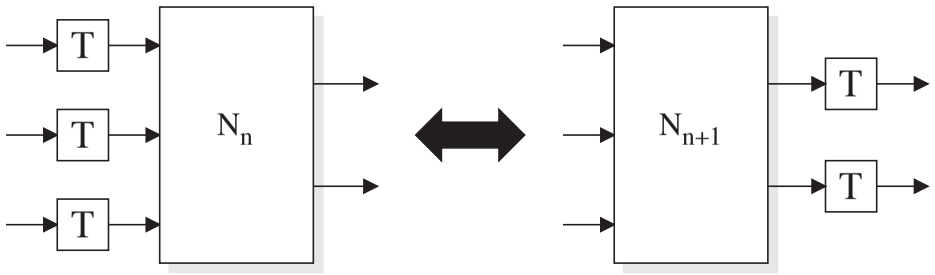
In practice, the implementation corresponding to model 1 may be the fastest, because of shorter propagations paths between the memory elements.

EQUIVALENCE TRANSFORMATIONS

Theorem 6.3

If an arbitrary, nonlinear, time-varying discrete time network, N_n, has delay elements in series with all inputs (outputs), then all the delays can be moved to the outputs (inputs) and the properties of N_n shifted, without changing the input–output behavior of the composite system.

N_n and N_{n+1} denote the properties of the network with reference to sample n and n+1, respectively.



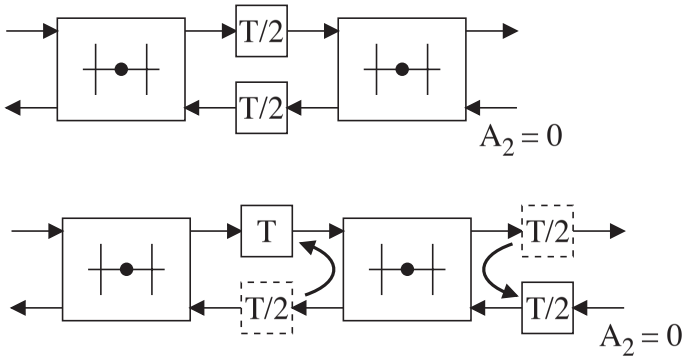
Essentially Equivalent Networks

Two networks that can be transformed into one another by the equivalence transformations just described, except for different (positive or negative) delays appearing in their input and output branches, are called *essentially equivalent* networks.

A delay element can not be propagated into a recursive loop.

However, the positions of the delay elements in a recursive loop can be changed as shown in example below.

The latency of the algorithm may be affected by such a change, but the maximal sample rate is unaffected.



INTERLEAVING AND PIPELINING

The throughput of a recursive algorithm is always bounded by the critical loop(*s*).

However, input, output, and other non-recursive branches and parts of loops in the signal-flow graph may have a critical path with a computation time longer than T_{min}.

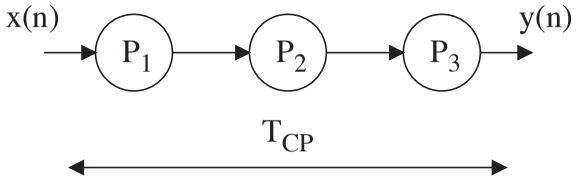
In that case, the critical path(*s*) will limit the sample rate.

However, the minimal iteration period can be achieved by interleaving and/or pipelining.

Both techniques can be simultaneously applied at different levels within a system—for example, at the algorithm level and at the hardware level.

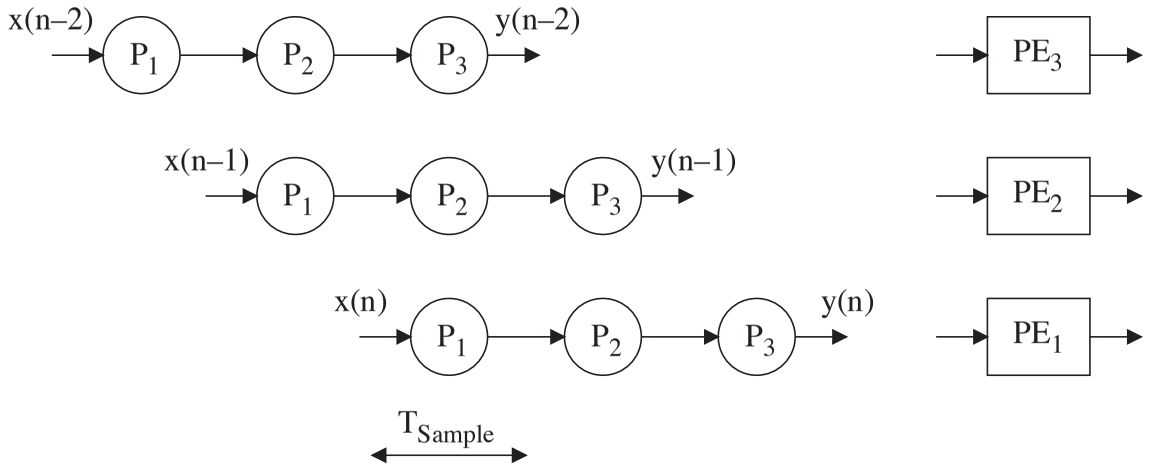
INTERLEAVING

Consider a sequential algorithm with three processes.



The throughput is the reciprocal of the length of the critical path, T_{CP}.

The throughput can be increased if the three processes P_1, P_2, and P_3 are mapped onto three processors – interleaving at the process level



The throughput is increased by a factor three, but the latency remains the same.

Interleaving of the computations can be used to increase the throughput of a sequential algorithm by an arbitrarily large amount.

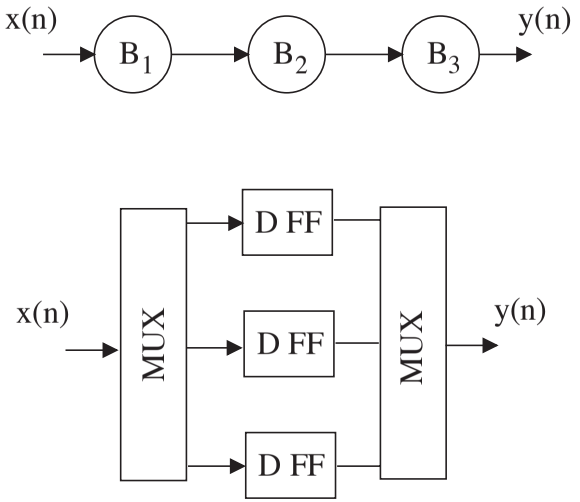
However, it requires a corresponding increase in resources since operations corresponding to several different time indices must be computed concurrently.

Interleaving is an inefficient technique in terms of resources since the three processors must be capable of executing all processes and no specialization is possible.

Example

A shift-register represent an algorithm with sequential storings and shifting of bits.

Advantages?

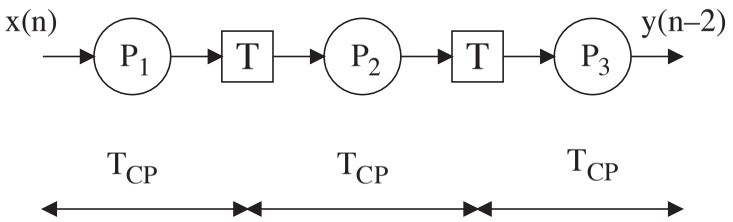


Pipelining

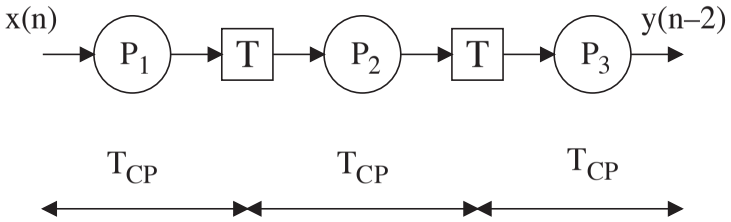
Pipelining, which is another method of increasing the throughput of a sequential algorithm, can be used if the application permits the algorithmic delay to be increased.

This is usually the case when the system (algorithm) is not inside a recursive loop, but there are many cases when the algorithmic delay must be kept within certain limits.

Pipelining of the three processes is accomplished by propagating algorithmic delay elements into the original critical path so that a set of new and shorter critical path(s) is obtained.



Ideally the critical path should be broken into paths of equal length.



For each of the three processors a new computation can begin as soon as the results of the current computations are stored in the corresponding (output) delay elements.

At steady state, three successive output value computations are performed concurrently.

Throughput is the reciprocal of the longest critical path, i.e., the same throughput as with interleaving.

The differences between interleaving and pipelining are that in the latter case the output is delayed by two sample periods and that the amount of required resources is reduced.

In pipelining, each PE operates on different processes while in interleaving a PE operates on every *n*th sample.

The number of PEs is still three, but each PE is now required to execute only one process per sample period. Hence a specialized, and thereby less expensive, PE may be used for each process.

Pipelining can be applied to sequential algorithms at any level of abstraction.

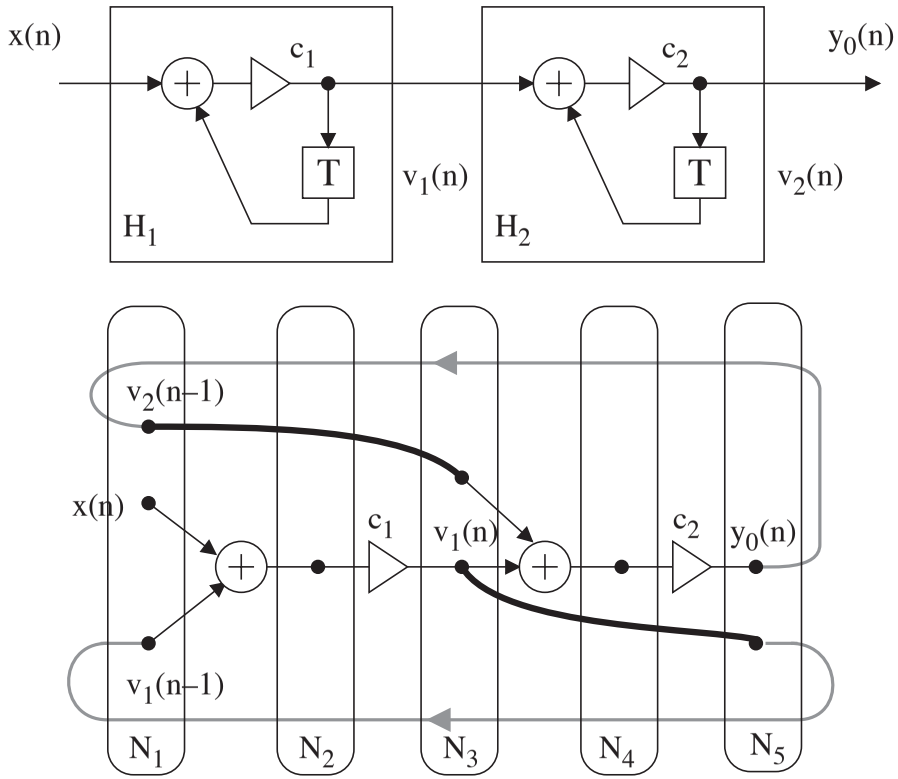
Typically, both basic DSP algorithms and arithmetic operations are pipelined.

At the digital circuit level pipelining corresponds to inserting latches between different circuit levels so that each level has the same transistor depth.

Note that parallelism in a structure is a fundamental property and can not be changed.

By inserting delay elements into the critical path (for example, by pipelining) a new structure with a higher degree of parallelism is obtained.

Example 6.9



The structure is completely sequential with respect to the operations.

The length of the critical path, which is equal to the latency, is

$$T_{CP} = 2(T_{add} + T_{mult})$$

Note that all operations lie on the critical path.

The transfer function is

$$H_0(z) = H_2(z)H_1(z) \quad , \quad |z| > R_+$$

where

$$H_1(z) = \frac{c_1}{1 - c_1 z^{-1}} \text{ and } H_2(z) = \frac{c_2}{1 - c_2 z^{-1}}$$

The original filter, $H_0(z)$, is now cascaded with a single delay element. A cascaded delay element will increase the group delay by a constant factor, T.

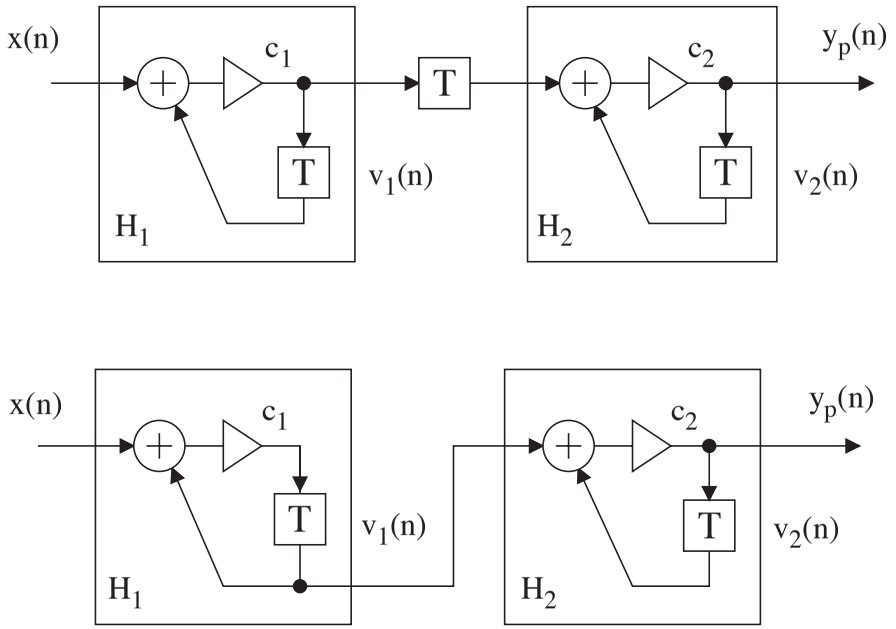
We get

$$H_p(z) = H_0(z)z^{-1} = z^{-1}H_2(z)H_1(z) \quad , \quad |z| > R_+$$

The order of the linear, shift-invariant filters and the delay element can be interchanged. Hence, we get

$$H_p(z) = H_2(z)z^{-1}H_1(z) = \frac{c_2}{1-c_2z^{-1}}z^{-1}\frac{c_1}{1-c_1z^{-1}}$$

The signal-flow graph for the new, pipelined structure is.



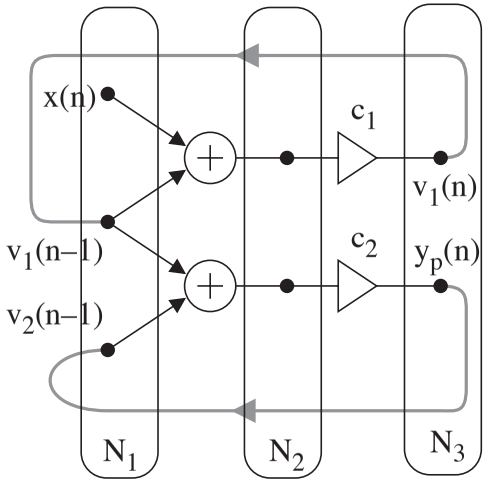
The maximal sample rate is determined by the recursive loops. The minimal iteration period bound for both loops is

$$T_{min} = T_{add} + T_{mult}$$

The precedence graph has two equally long critical paths of length T_{min}.

The new algorithm has a higher degree of parallelism compared to the original structure which is completely sequential.

The throughput is twice that of the original algorithm.



The algorithmic delay has increased, but the latency has remained the same.

$$y_p(n) = y_0(n-1)$$

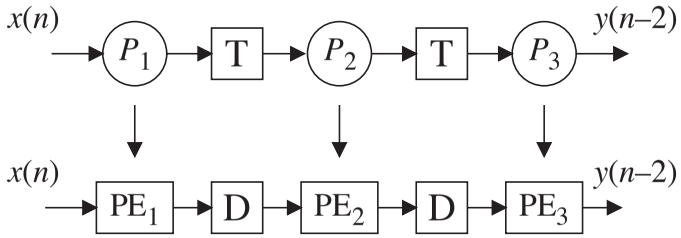
Structural Pipelines

In classical pipelines, only one operation is executed by each pipeline stage (PE) separated by registers (delay elements).

A PE is allocated to each stage of the pipeline.

The pipeline stages do not share resources. This leads to a fixed and simple communication pattern between the processors.

This case is often referred to as a *structural pipeline*.



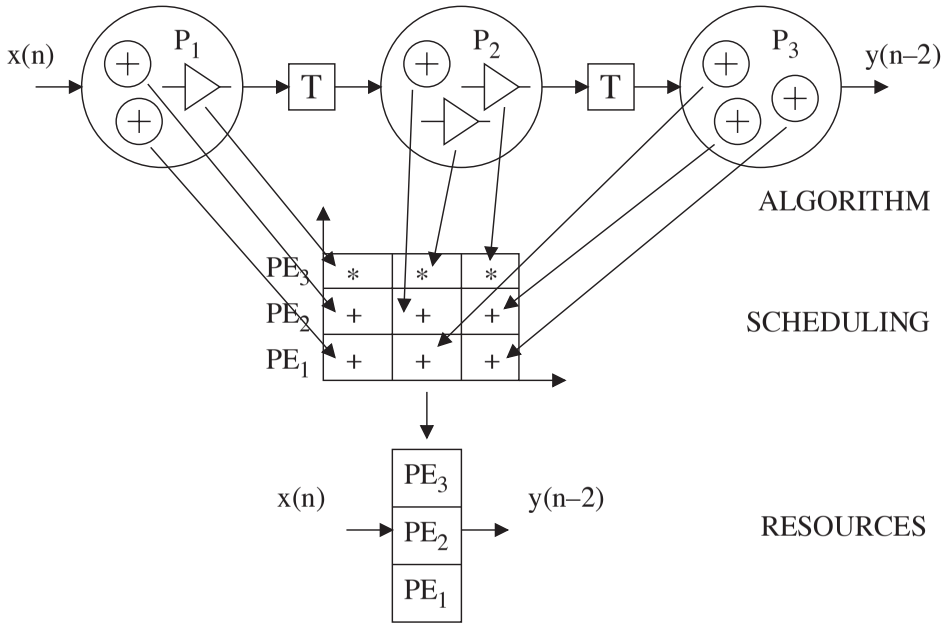
Functional Pipelines

A more general case is to partition the pipeline stages into several smaller subtasks (operations) that are subsequently mapped onto a hardware structure.

The operations can of course be executed by a structural pipeline, but a more efficient way to organize the hardware structure in terms of utilization of the resources is to allow the pipeline stages to share resources.

That is, operations within a pipeline stage are not bounded to a particular PE, and operations that are not concurrent may share resources.

Functional pipelining is used to increase the parallelism in the algorithm so that the operations may be scheduled more freely.

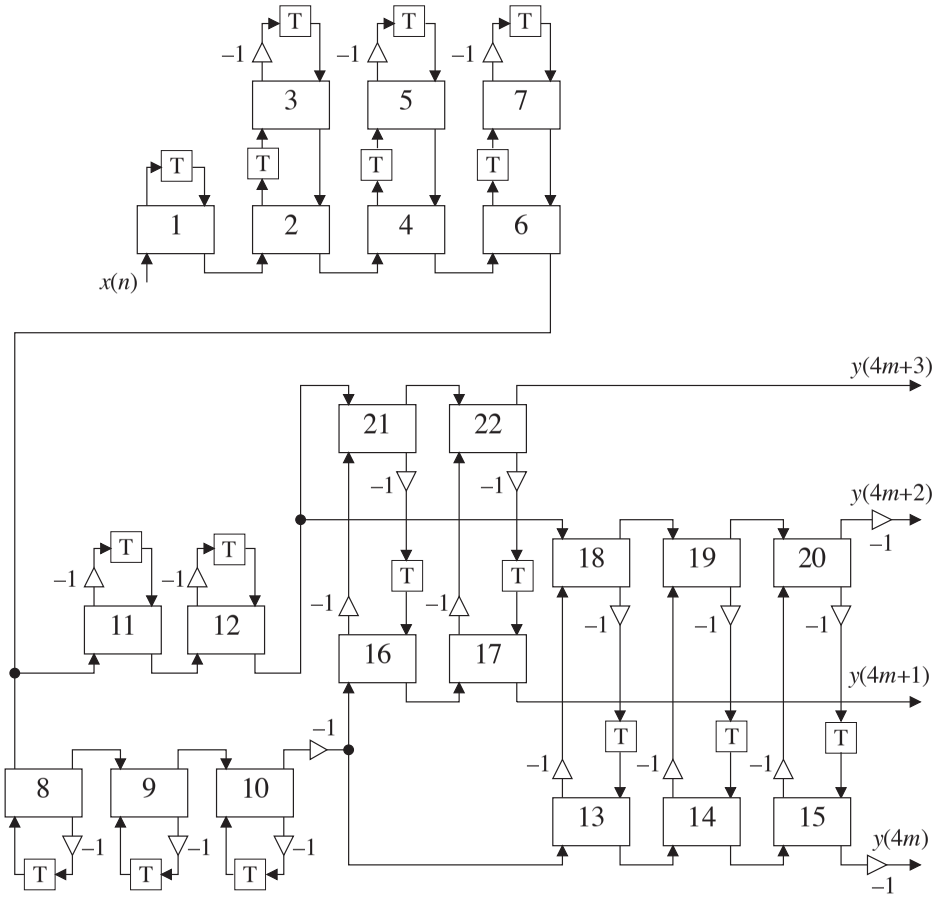


INTERPOLATOR, CONT.

In order to schedule the operations and determine the necessary amount of computational resources for the interpolator, we first need to determine the precedence relations for the operations.

We choose to use the adaptors as atomic operations, i.e., an operation with three inputs (two data and one coefficient) and two output values.

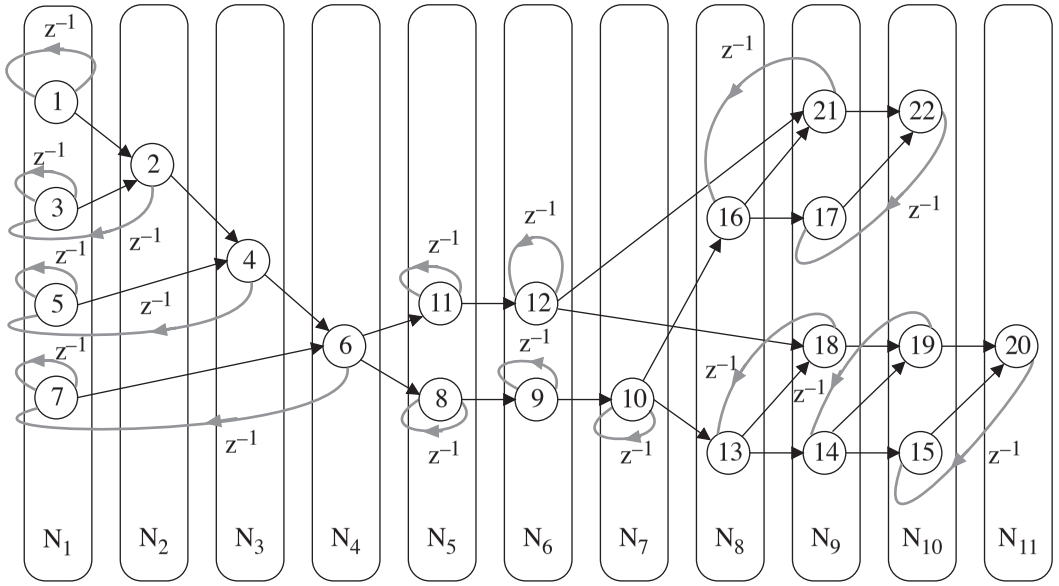
First, we draw the wave-flow graph without switches to demonstrate the algorithm more clearly.



Adaptors that have valid inputs can be executed immediately. In the first time slot, only adaptors 1, 3, 5, and 7 have valid inputs.

Only adaptor 2 can be executed in the second time slot, and so on.

Altogether, 11 time slots are required to complete the whole filter. The number of concurrent operations ranges between 1 and 4. The average number is only 2 adaptors/time slot.



In order to determine the maximal sample rate for the interpolator, we assume that a multiplication takes 21 clock cycles and an addition takes 1 clock cycle.

The computation paths through an adaptor involve two additions and one multiplication, i.e., (21+2) clock cycles.

The usable clock frequency for the target technology is estimated to be at least 220 MHz. The critical path for the interpolator is through 11 adaptors, i.e.,

$$T_{CP} = \frac{11(21+2)}{220\ 10^6} = 1.15 \quad \mu\text{s or } 869.6 \text{ kHz}$$

The throughput will be too low, since the required input sampling frequency is 1.6 MHz.

Four output values are generated for each input sample. Hence, the output sampling frequency is only 3.478 MHz.

We will therefore use pipelining of the interpolator to reach the required output sample frequency of 6.4 MHz.

From Figure 11.60 it is evident that the minimal input sampling period is determined by a loop through two adaptors. We get

$$T_{min} = \frac{2(21+2)}{220\ 10^6} = 0.209 \quad \mu\text{s or } 4.785 \text{ MHz}$$

It is therefore possible to achieve the required sample frequency by breaking the critical path into smaller pieces using pipelining.