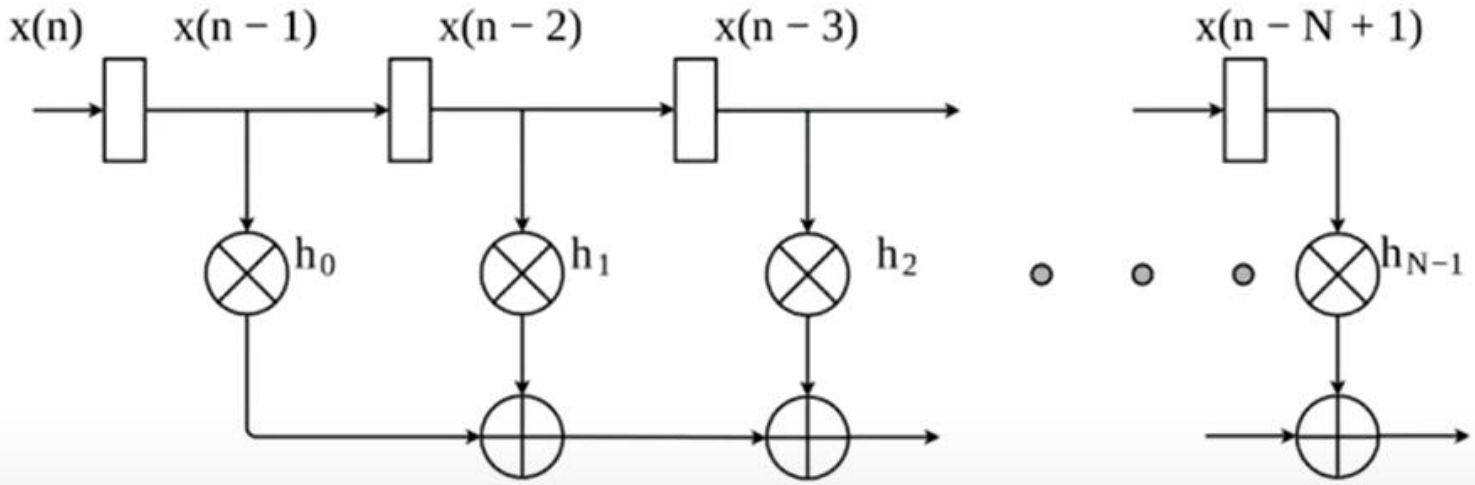
BASIC ARCHITECTURES

Lecture-13

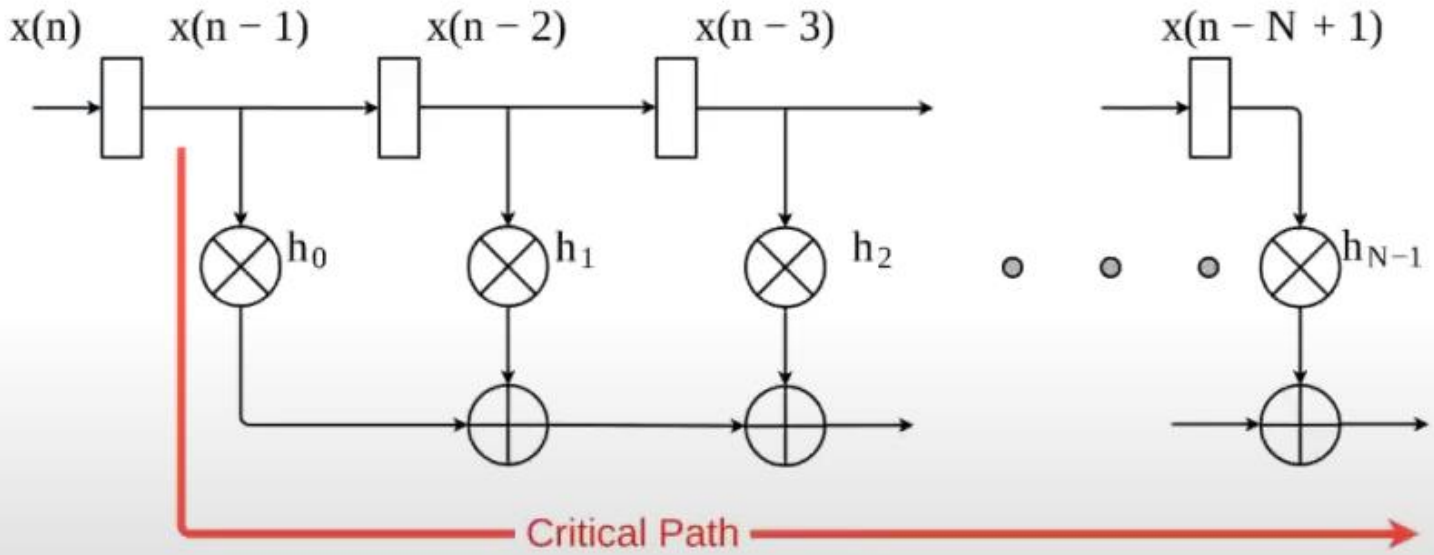
Simple FIR Filter

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$



- Direct form-I (DF-I)
- Assume hardware (technology dependent)
  - Adders
  - Multipliers
  - Registers

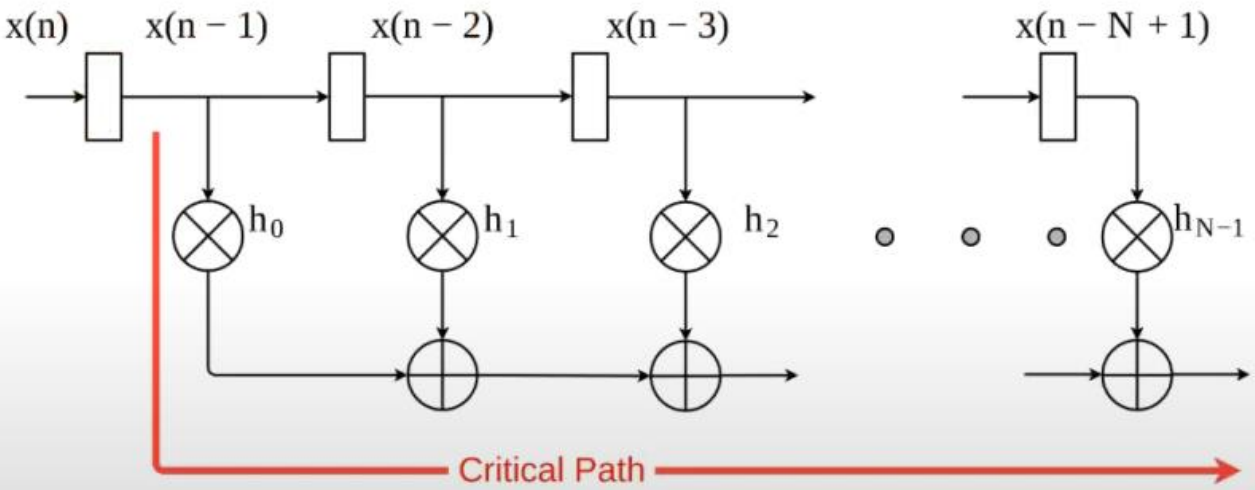
Critical path



- $$T_{cp} = T_M + (N-1) \times T_A$$

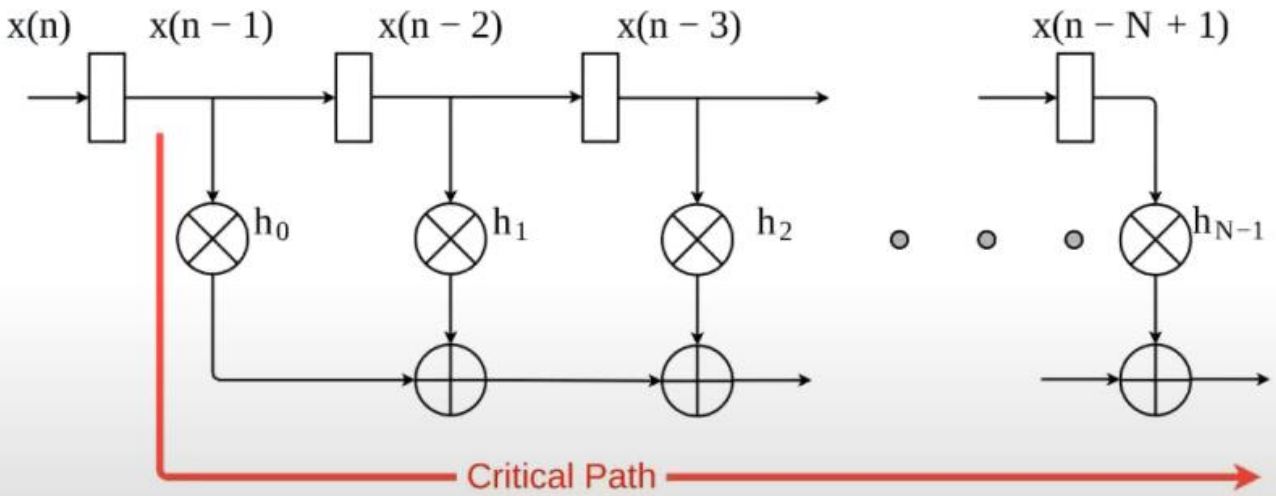
Critical path

Sample rate *vs* Clock rate



- $$T_{cp} = T_M + (N-1) \times T_A$$

Critical path

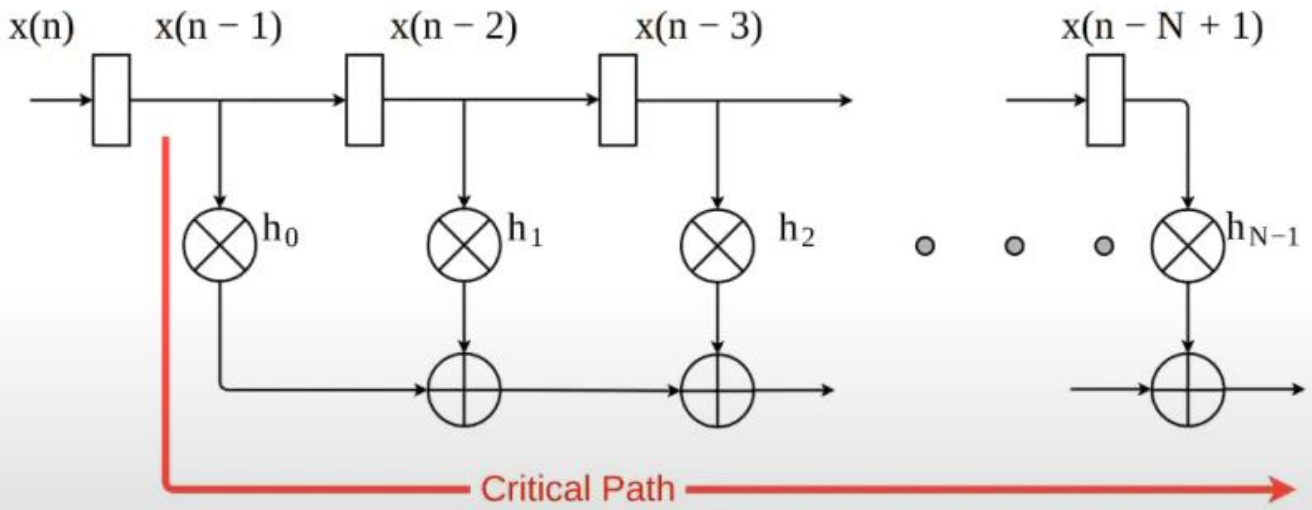


- $$T_{cp} = T_M + (N-1) \times T_A$$

Sample rate *vs* Clock rate

- Clock signal applied to all registers
- Every tick of clock pushes one sample through

Critical path



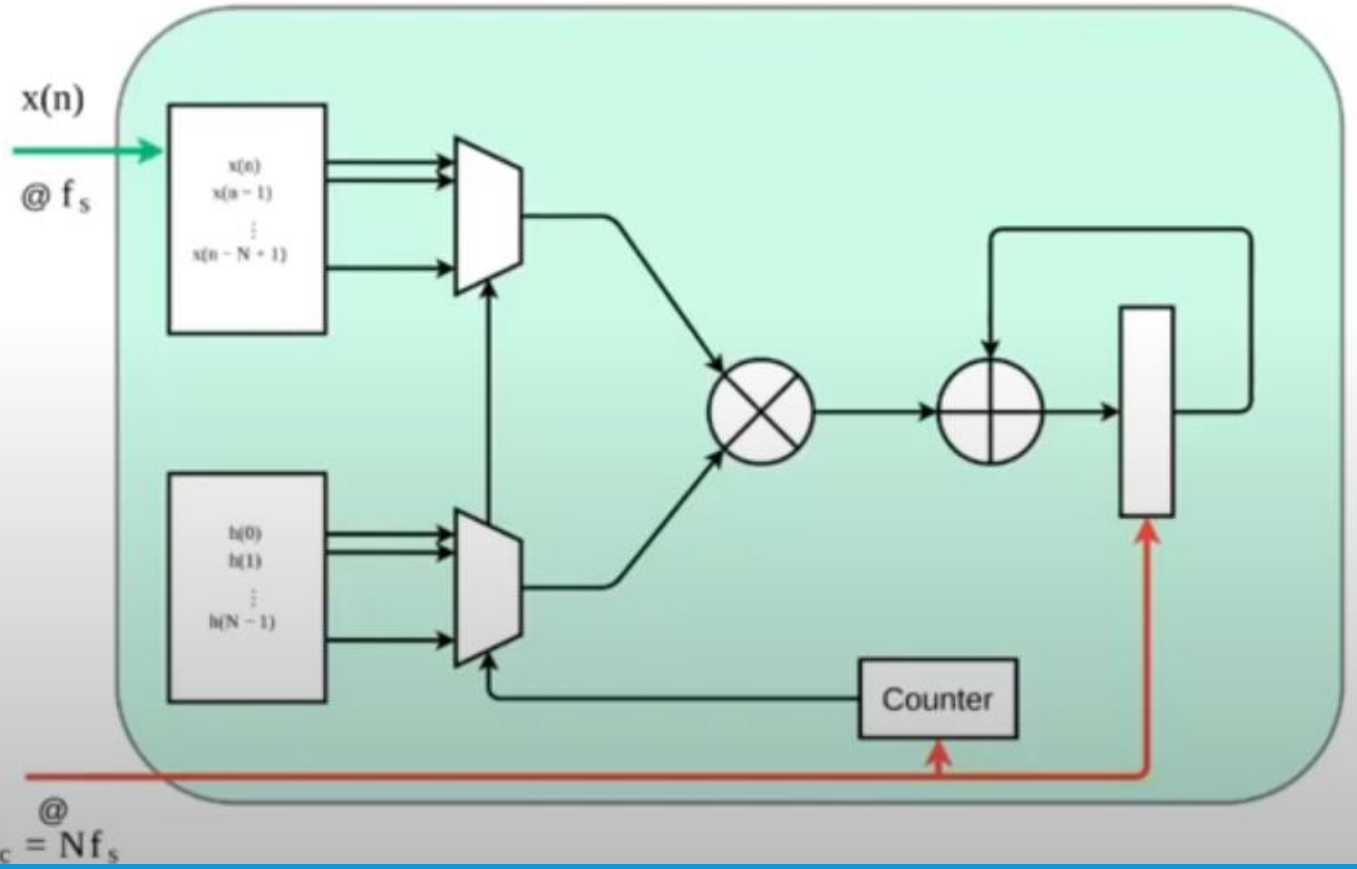
- $$T_{cp} = T_M + (N-1) \times T_A$$

Sample rate *vs* Clock rate

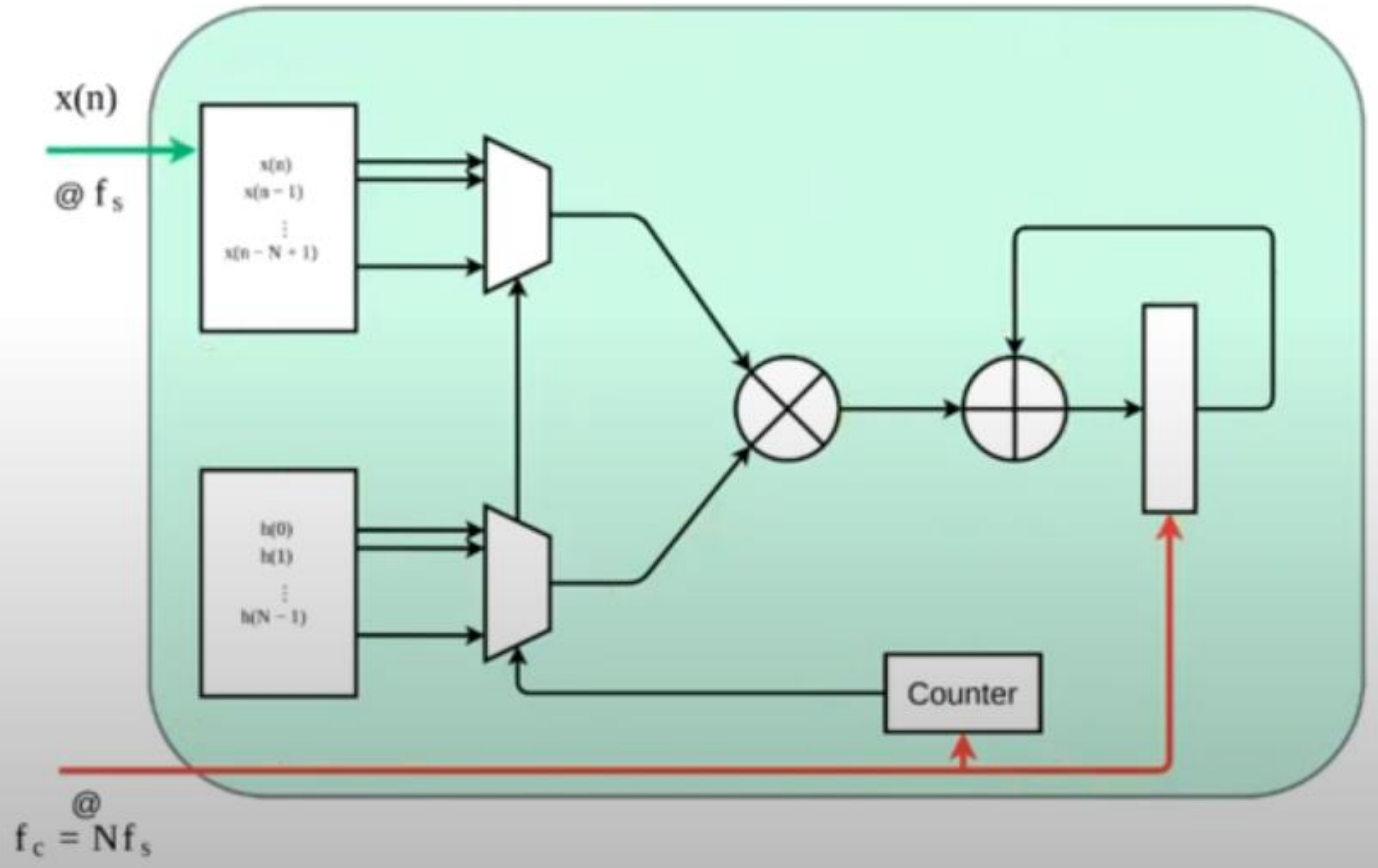
- Clock signal applied to all registers
- Every tick of clock pushes one sample through

Sample rate = Clock rate

Shared hardware

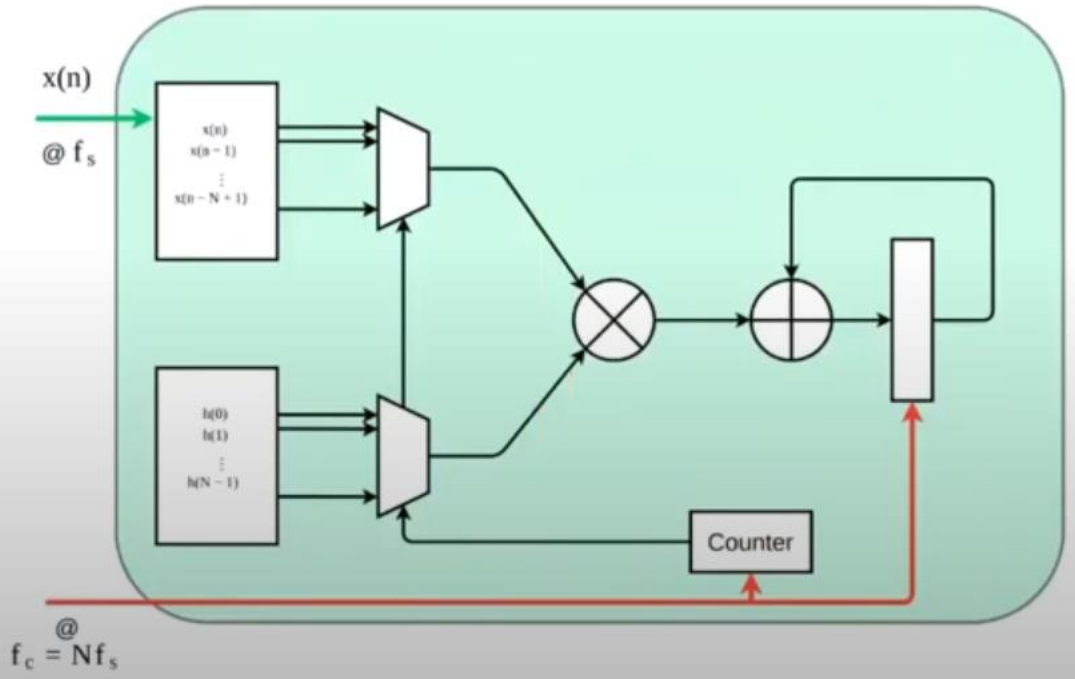


Shared hardware



- Single MAC unit
- Memory blocks to hold *x*, *h*
- Counter to sequence operations

Shared hardware



- Single MAC unit
- Memory blocks to hold *x*, *h*
- Counter to sequence operations

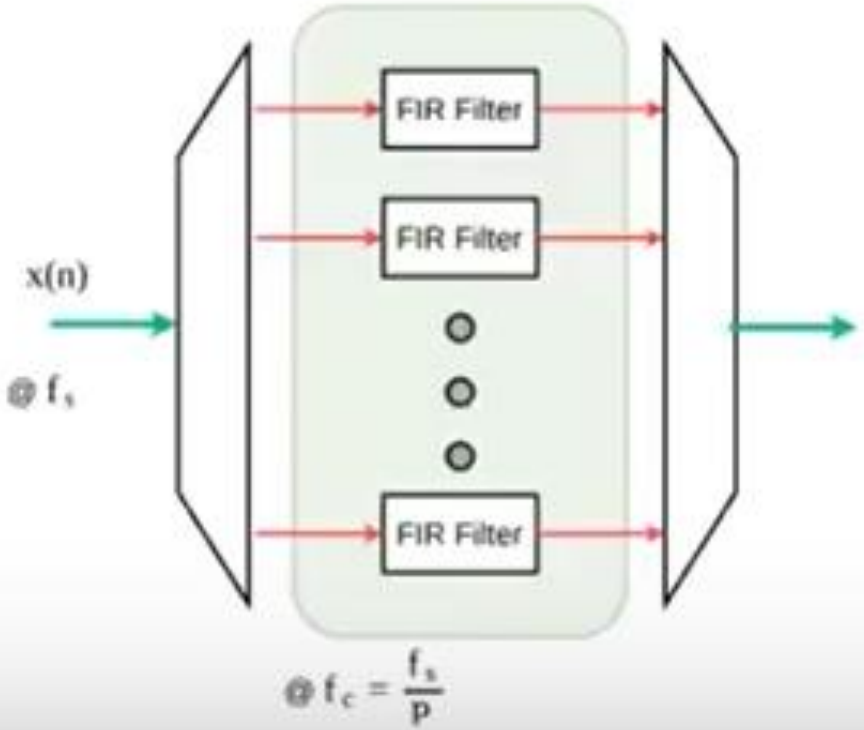
Sample rate < Clock rate!

Speeding up?

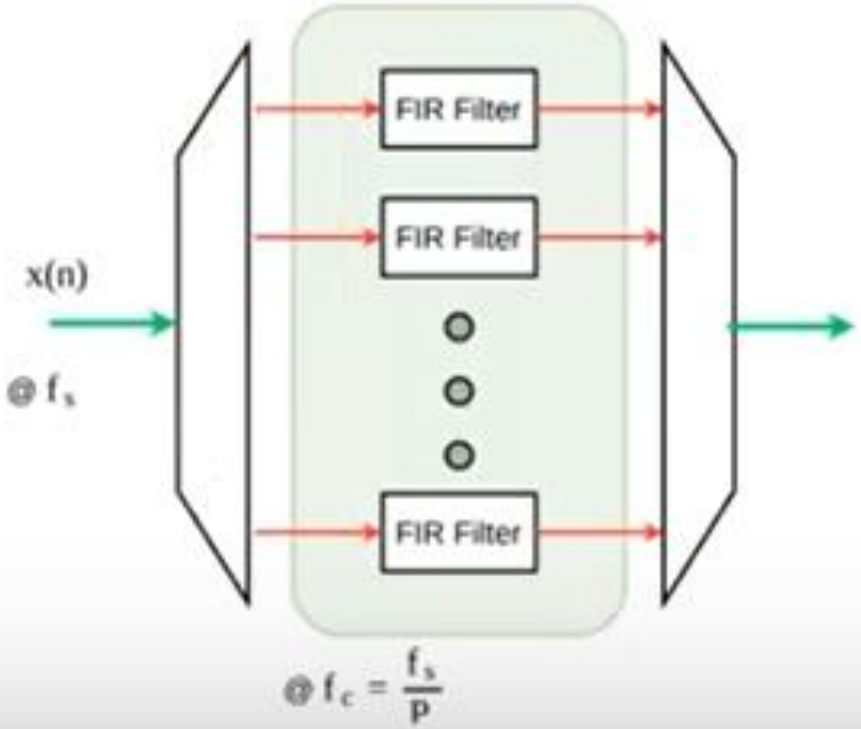
- Run at a faster clock speed
  - better technology needed!
- Architecture changes
  - Direct Form-II
  - Bit-width reduction
  - Pipelining and other transformations
- Algorithm
  - FFT based convolution

> Parallelism

Parallel FIR

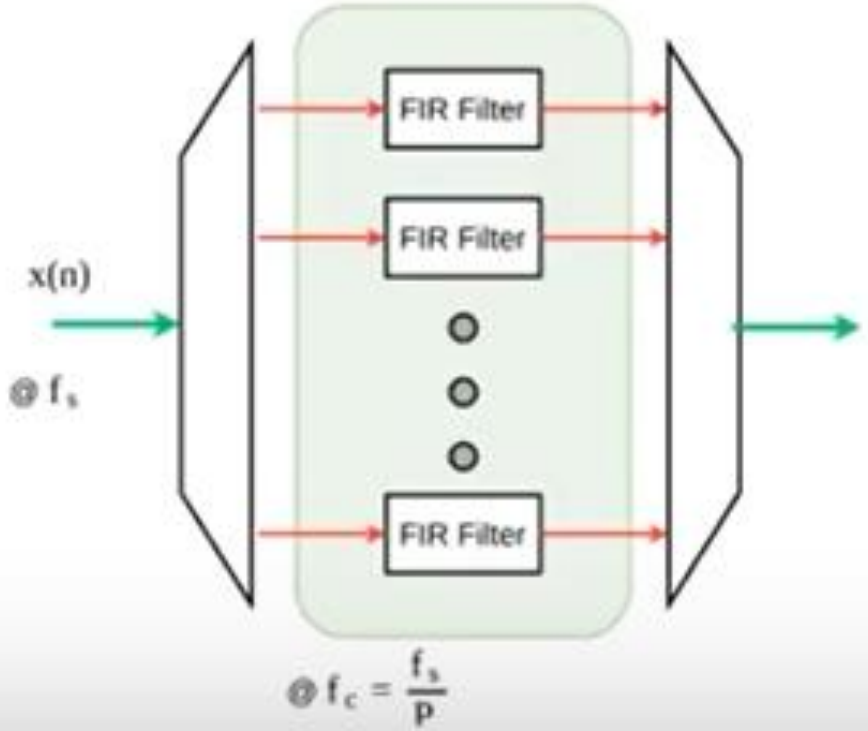


Parallel FIR



- Multiple full FIR filters
- Serial-to-Parallel / Parallel-to-Serial

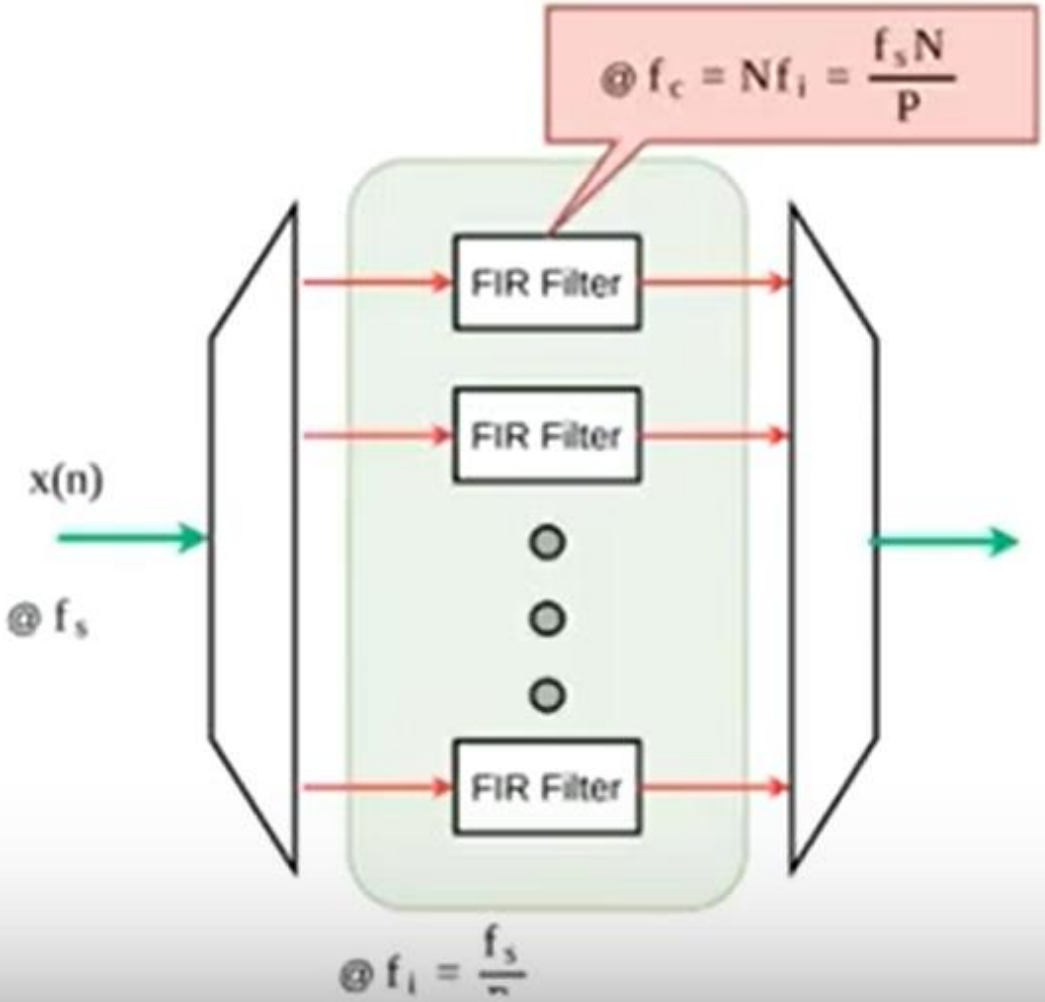
Parallel FIR



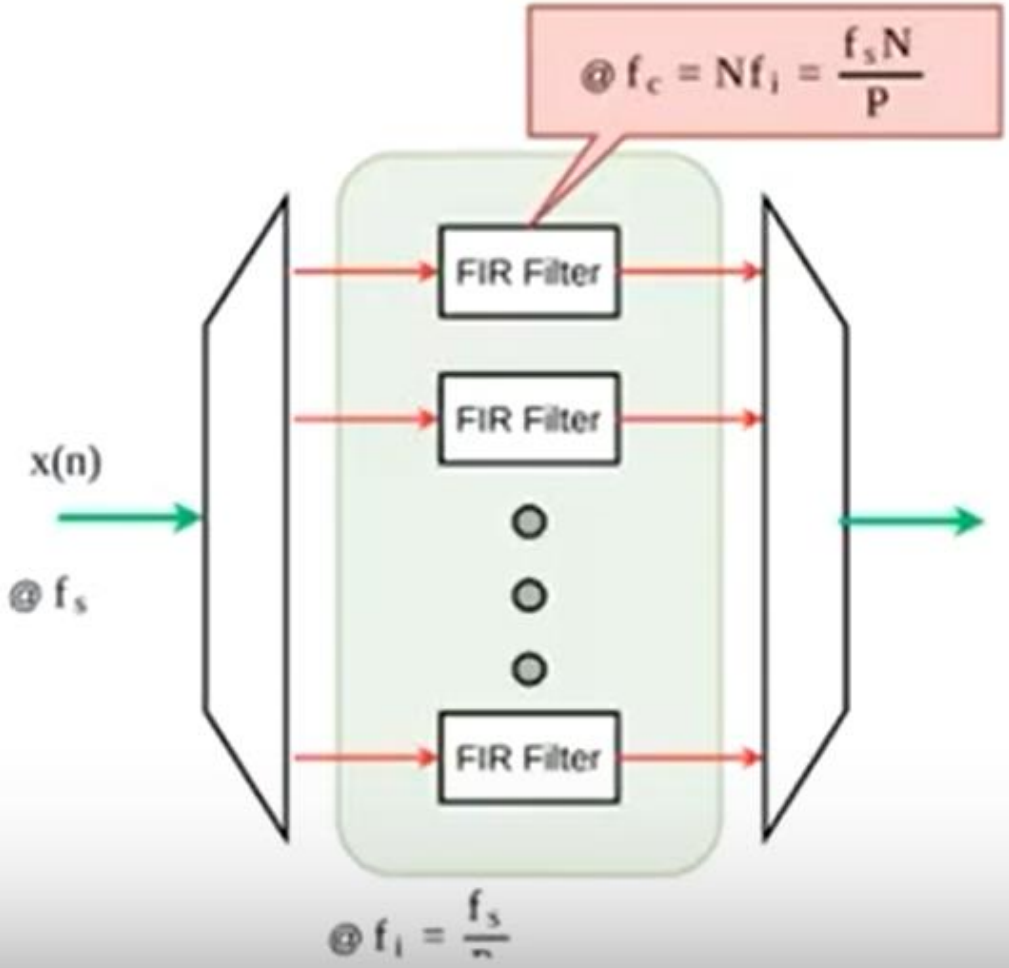
- Multiple full FIR filters
- Serial-to-Parallel / Parallel-to-Serial

Sample rate > Clock rate!

Mix-and-Match



Mix-and-Match



- S2P / P2S used for rate changing
- Serial / resource shared FIR filters

Why? Eh, because we can...

Minimum possible sample period

- Sample period *inverse* of sample rate
- Parallel implementation:
  - Theoretically no minimum bound!

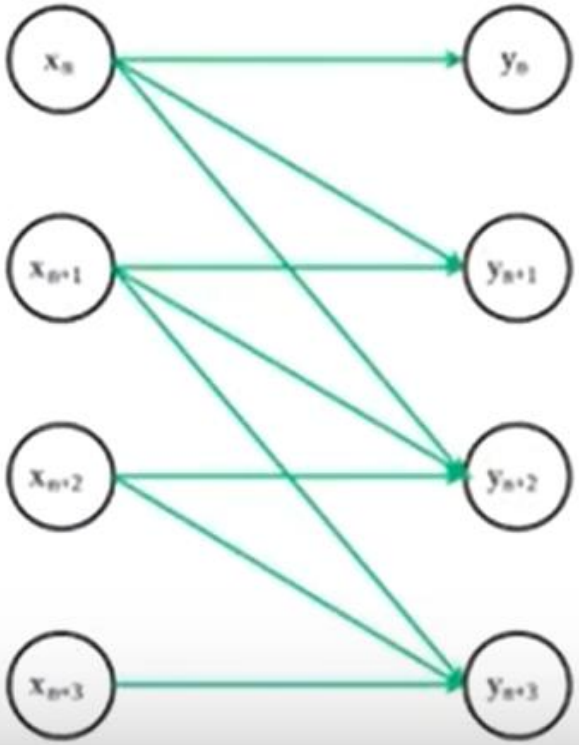
$$
\begin{aligned}
y(n) &= ax(n) + bx(n-1) + cx(n-2) \\
y(n+1) &= ax(n+1) + bx(n) + cx(n-1) \\
y(n+2) &= ax(n+2) + bx(n+1) + cx(n) \\
\vdots &= \vdots \quad \vdots \quad \vdots \quad \vdots
\end{aligned}
$$

IIR filter

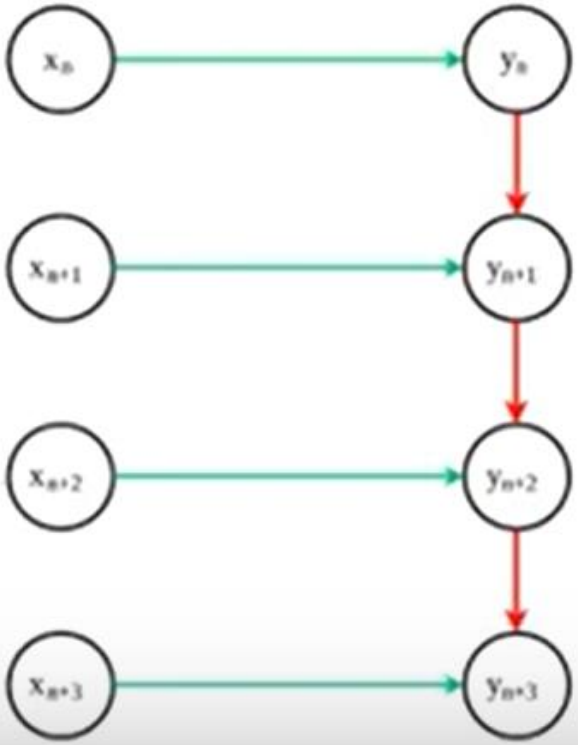
$$
\begin{aligned}
y(n) &= y(n-1) + ax(n) \\
y(n+1) &= y(n) + ax(n+1) \\
y(n+2) &= y(n+1) + ax(n+2) \\
\vdots &= \vdots \quad \vdots \quad \vdots
\end{aligned}
$$

Dependencies

FIR Filter



IIR Filter



KPN AND DATAFLOW

Generalize from filter

Kahn Process Networks

- Distributed model of computation
- Sequential Processes
- Communicate through FIFO channels

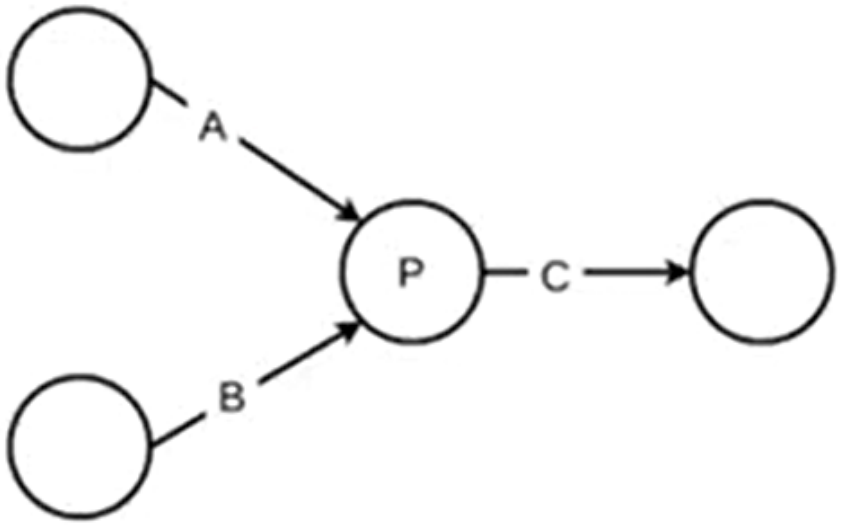
> G. Kahn and J. L. Rosenfeld, "The semantics of a simple language for parallel programming", Proc. IFIP Congress on Information Processing, 1974

Generalize from filter

Kahn Process Networks

- Distributed model of computation
- Sequential Processes
- Communicate through FIFO channels

> G. Kahn and J. L. Rosenfeld, "The semantics of a simple language for parallel programming", Proc. IFIP Congress on Information Processing, 1974



- A, B, C: (potentially unbounded) FIFO channels
- P: processing node

KPN semantics

- Tokens - atomic read / write elements
- Communication via unbounded FIFO channels
  - non-blocking writes
  - blocking reads
  - Cannot *test* for presence of a token without consuming it
- One FIFO cannot be consumed by multiple processes
- Multiple processes cannot write to one FIFO

> Given a specific input sequence of tokens, the execution must be deterministic

Processes

- Without any inputs: source
- Without any outputs: sink
- Fixed numbers to consume from inputs, produce on outputs at each firing

Processes

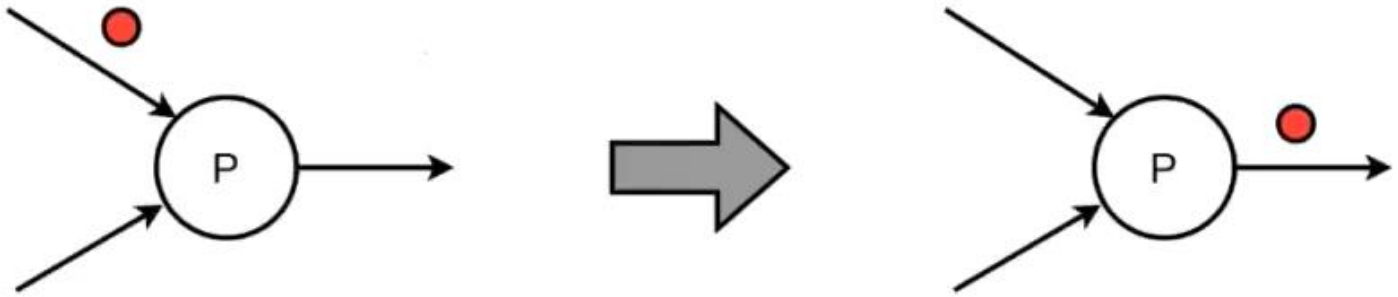
- Without any inputs: source
- Without any outputs: sink
- Fixed numbers to consume from inputs, produce on outputs at each firing

> State machine:
>
> - Wait: for enough inputs
> - Active: execute functionality, generate outputs, return to *wait*

Basic semantics

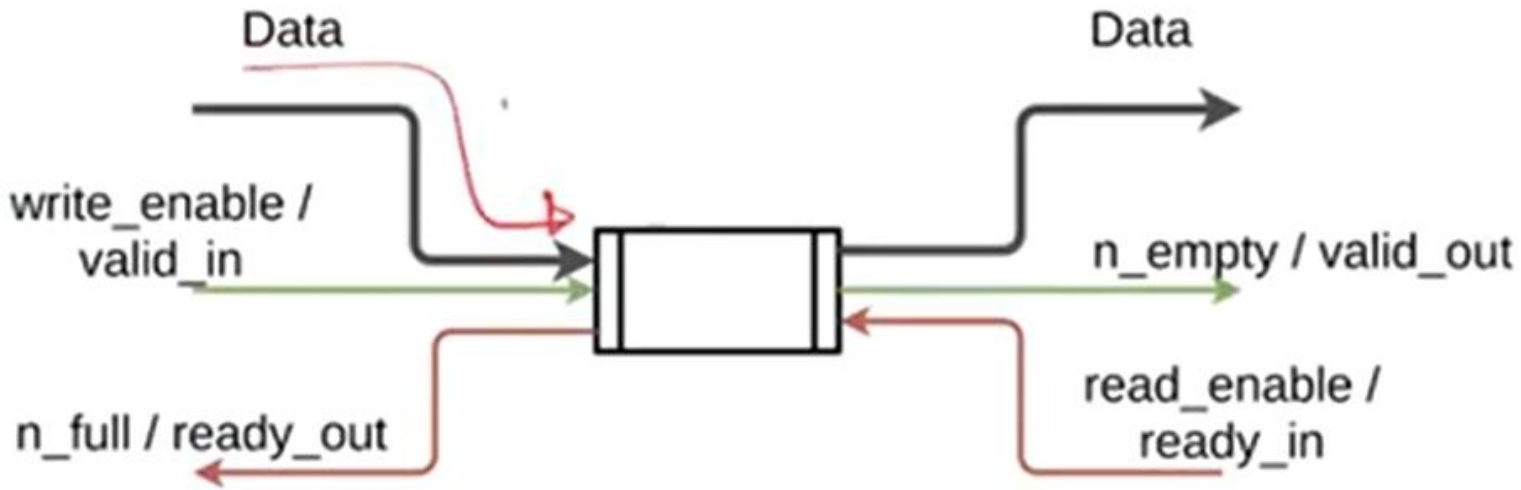

```
process A(in U, out V) {
    x = U.read();
    y = func(x);
    V.write(y);
}
```

Complex processes




```
process A(in U, in V, out W) {
    static bool B = true;
    if (B) x = U.read();
    else   x = V.read();
    y = func(x);
    W.write(y);
    B = !B;
}
```

FIFO buffer




```
if (valid_in && ready_out) { // (n_full && write_enable)
    write_into_fifo(x);
}
if (valid_out && ready_in) { // (n_empty && read_enable)
    y = read_from_fifo();
}
```

Static Dataflow

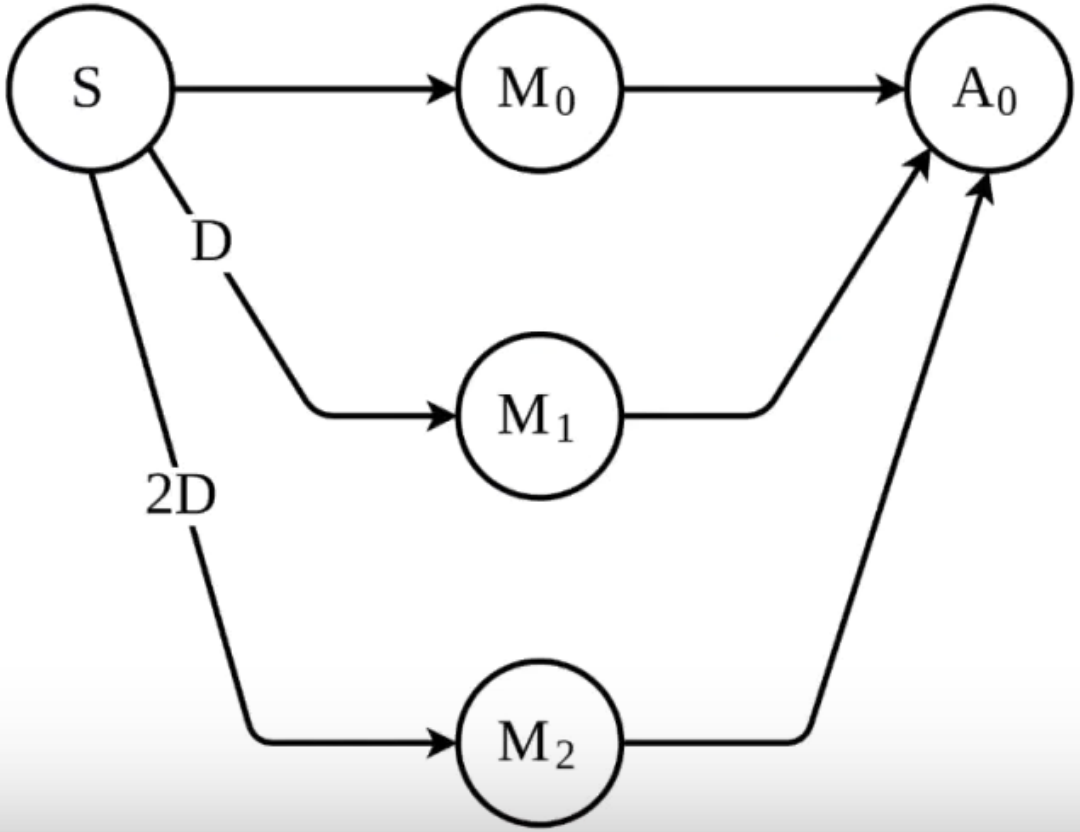
- Simpler version of KPN
  - All behaviour completely static: no conditionals
- Producer / Consumer relationships
- Good fit for *pure* DSP
  - without conditionals
- Allows system-level analysis
  - buffers, deadlocks...

Iteration

> The repetition of a process...

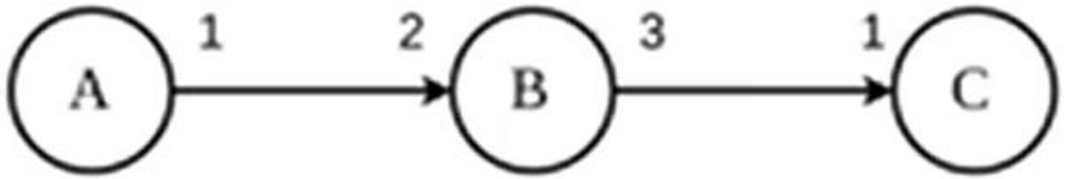
- Single sequence of operations
  - Repeat for each incoming sample
  - One complete sequence generates one output

SDF for FIR



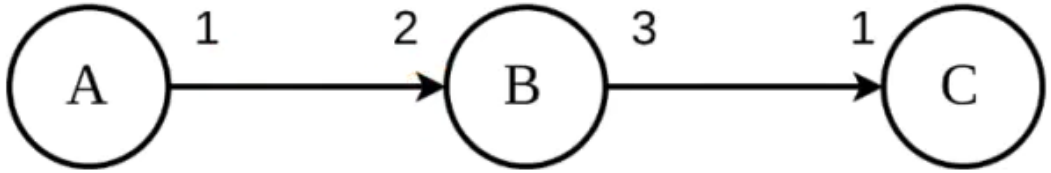
- Registers replaced by D
- *D* indicates a delay or token
  - initial value?

General SDF



- *A* is a source: can fire at any time
  - produces one output token on each firing
- *B*: each time it fires -
  - consumes 2 tokens
  - produces 3 tokens
- *C*: sink
  - consumes 1 token on each firing

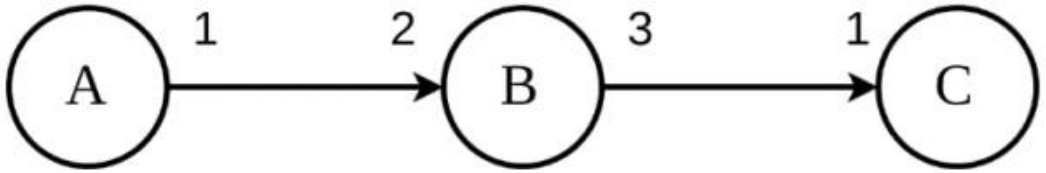
General SDF



- *A* is a source: can fire at any time
  - produces one output token on each firing
- *B*: each time it fires -
  - consumes 2 tokens
  - produces 3 tokens
- *C*: sink
  - consumes 1 token on each firing

Firing sequences

General SDF

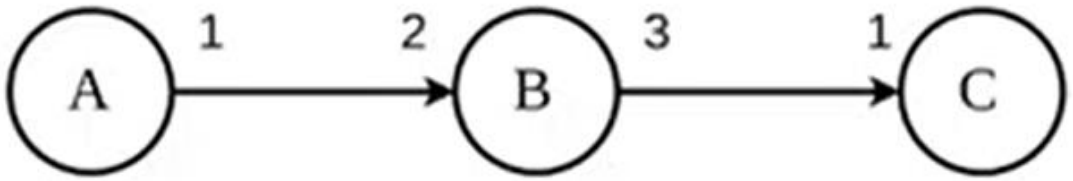


- *A* is a source: can fire at any time
  - produces one output token on each firing
- *B*: each time it fires -
  - consumes 2 tokens
  - produces 3 tokens
- *C*: sink
  - consumes 1 token on each firing

Firing sequences

AABCCC

General SDF



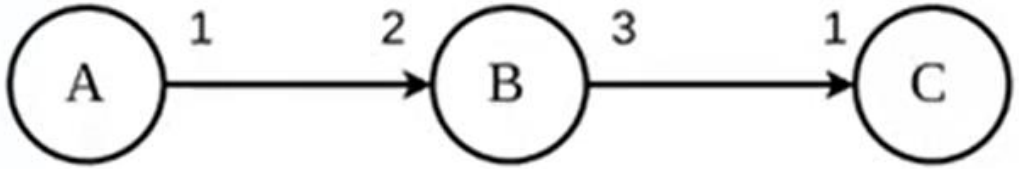
- *A* is a source: can fire at any time
  - produces one output token on each firing
- *B*: each time it fires -
  - consumes 2 tokens
  - produces 3 tokens
- *C*: sink
  - consumes 1 token on each firing

Firing sequences

AABCCC

ABC

General SDF



- *A* is a source: can fire at any time
  - produces one output token on each firing
- *B*: each time it fires -
  - consumes 2 tokens
  - produces 3 tokens
- *C*: sink
  - consumes 1 token on each firing

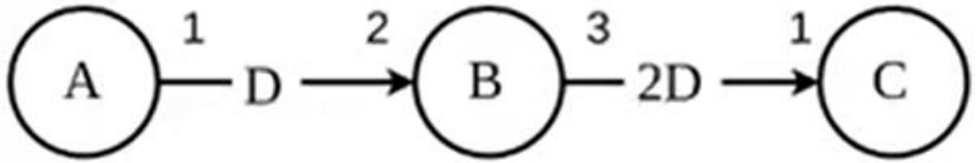
Firing sequences

AABCCC

ABC

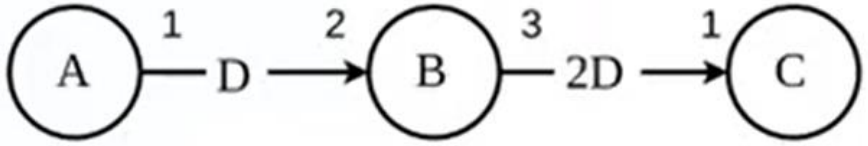
AAAABCCBCCCC

Initial tokens



- *D* on edge *AB* is an initial token
  - now *A* only needs to fire once more to make *B* ready

Initial tokens



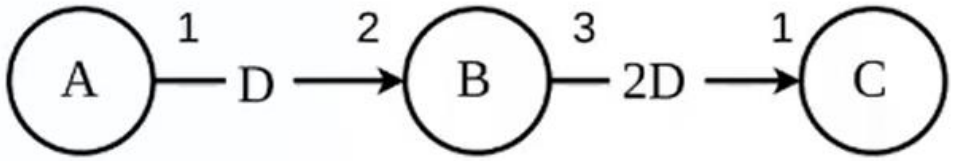
- *D* on edge *AB* is an initial token
  - now *A* only needs to fire once more to make *B* ready

Firing sequences

ABCCCCC

All edges empty of tokens

Initial tokens



- *D* on edge *AB* is an initial token
  - now *A* only needs to fire once more to make *B* ready

Firing sequences

ABCCCCC

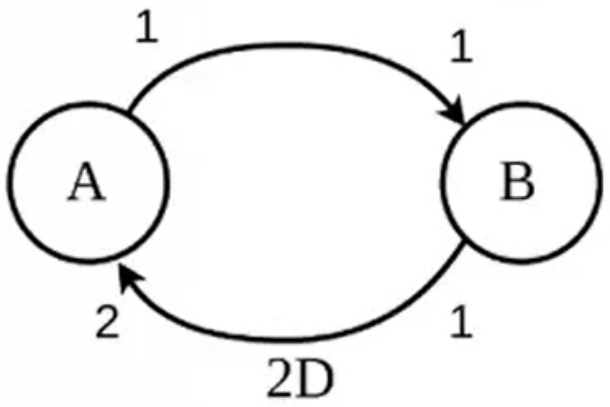
All edges empty of tokens

AABCCC

Same sequence as before?

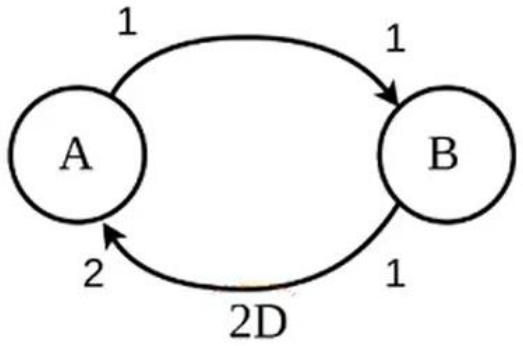
Problems?

Deadlock

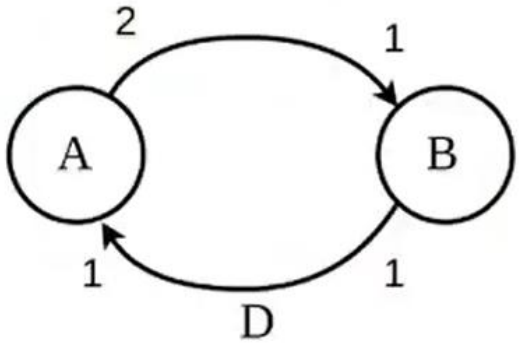


Problems?

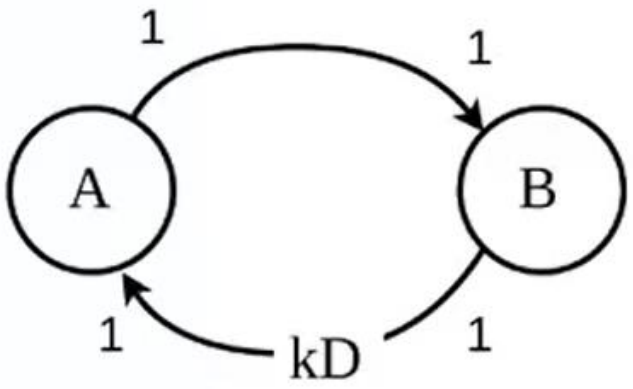
Deadlock



Unbounded FIFO



Bounded FIFOs?



Any Question…

Thank you