
Computer Aided Digital Systems Design - EE 4743/6743

Sherif Abdelwahed

Scheduling

**Department of Electrical and Computer Engineering
Mississippi State University**

Digital System Design

- At this point, we are trying to do
 complex datapaths + complex control
 - Digital system designers are typically faced with the problems of :
 - **Constraints:** minimum clock frequency, maximum number of clock cycles, target device, resource limits (don't have an infinite number of logic cells available)
 - **Execution unit architecture and number :** fast adder? Slow adder? Pipelined or non-pipelined multiplier? SRAM versus registers? How many do I need based on constraints?
 - **Scheduling:** what happens during what clock cycle?
-

Timing Constraints

- Two main constraints can be placed on a digital system design: clock period, and clock cycle constraints
 - A **clock period constraint** will define the minimum clock frequency.
 - Will affect the architecture of your execution units (fast adder versus slow adder, pipelined execution unit versus non-pipelined execution unit)
 - A **clock cycle constraint** limits the available number of clock cycles to perform operation
 - Total computation time = clock period * clock cycles
 - Other constraints: power, device type, input/output connections
-

Resource Estimation

- Given constraints, would like a lower bound estimate on the number of resources needed
- Resource types: Registers, Execution units (adders, multipliers, etc)
- Lets do resource estimation for the equation below:

$$Y = a_0 * X + a_1 * X@1 + a_2 * X@2 + a_3 * X@3$$



FIR Filter

$$Y = a_0 * X + a_1 * X@1 + a_2 * X@2 + a_3 * X@3$$

The equation above is an equation for a 4-Tap **Finite Impulse Response** (FIR) digital filter.

Each sample period a new value for X is input to the system. A sample period is measured in clock cycles, and the number of clock cycles per sample period will be an external constraint.

X is the value for current sample period.

$X@1$ is the value for one sample period back.

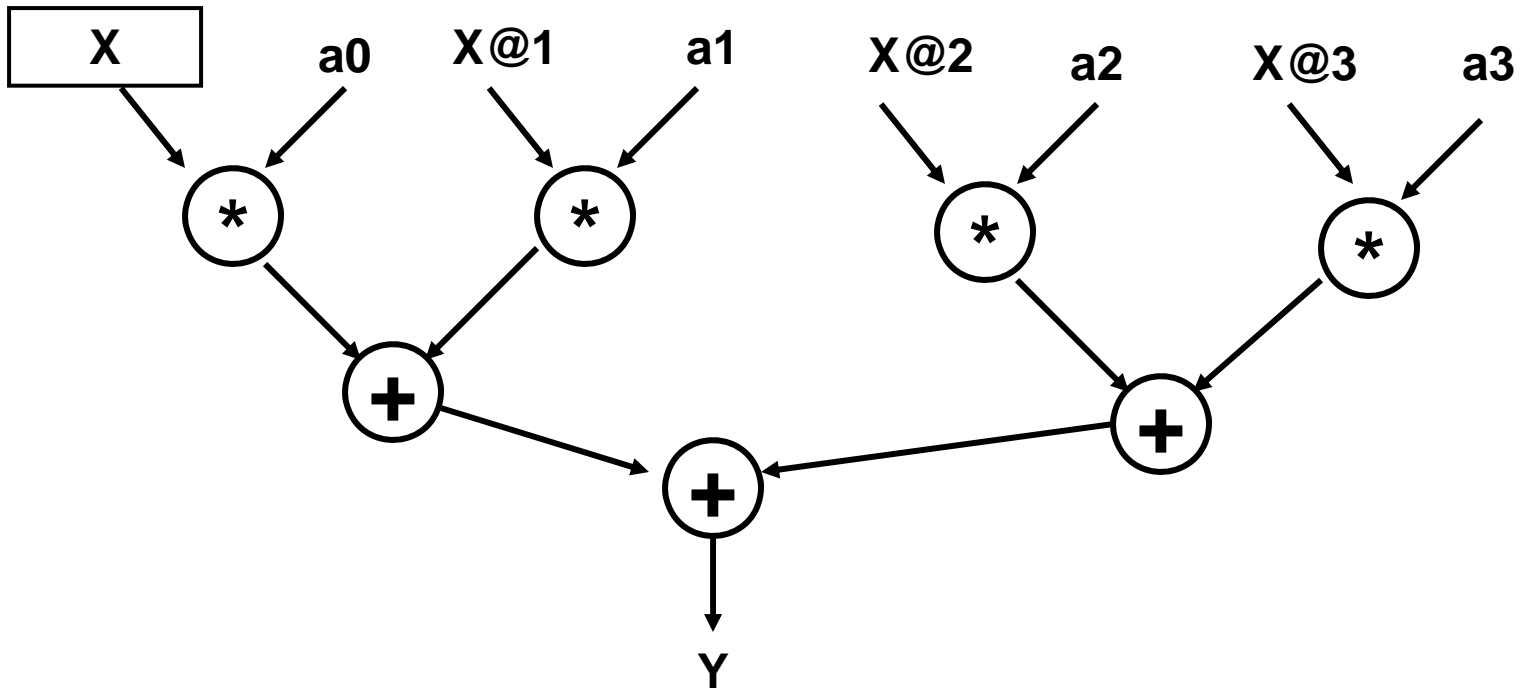
$X@2$ is the value for two sample periods back.

$X@3$ is the value for three sample periods back.

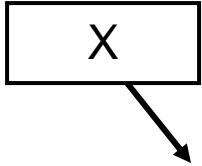
a_0 , a_1 , a_2 , a_3 are the filter coefficients. Changing these coefficients change the filter function; assumed to be preloaded.

Dataflow Graph

- We need a method of visualizing the data dependencies and operations to be performed. One method of doing this is the **dataflow** graph.



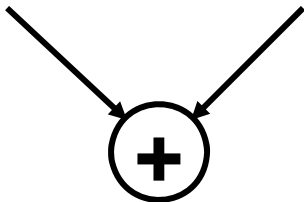
Operations in a Dataflow graph



An input operation. Inputs are assumed registered. An input operation will take 1 clock cycle.



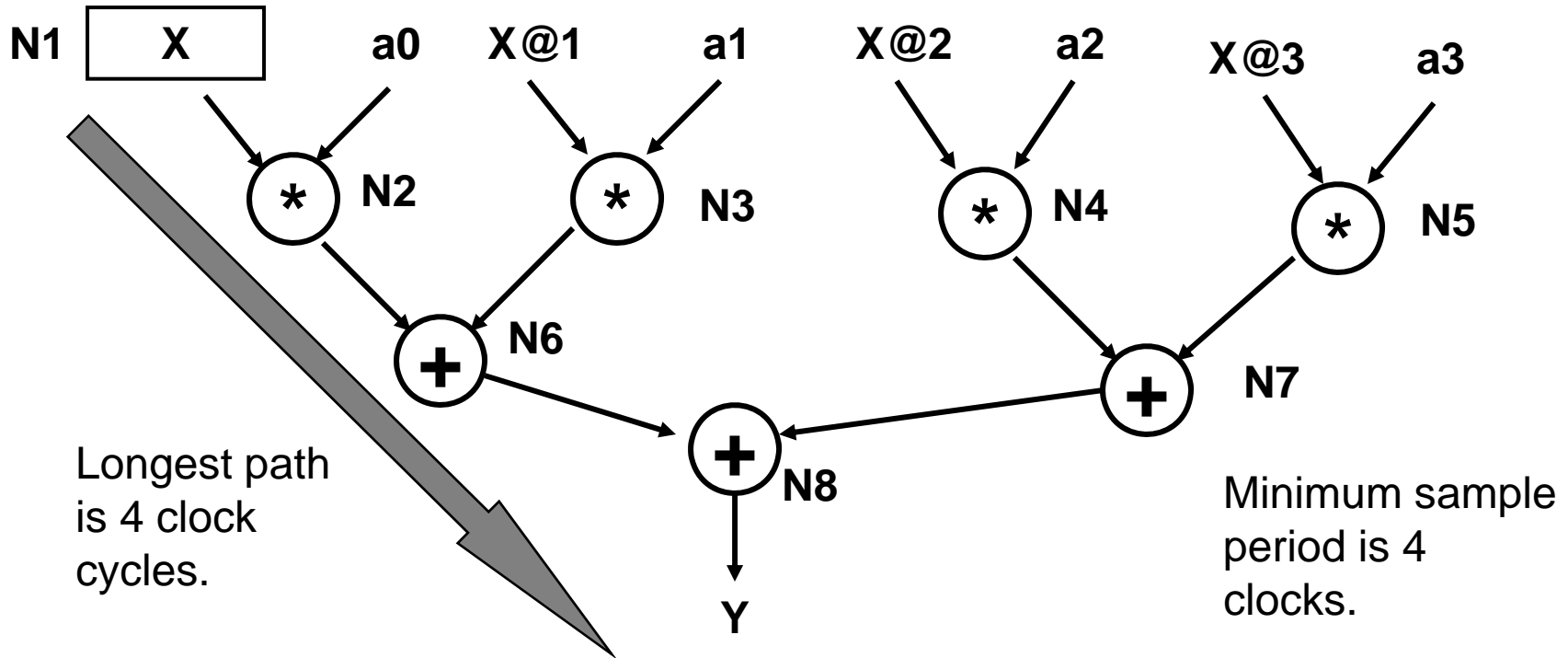
An output operation. Outputs are not assumed to be registered because they will be registered by the datapath they are being passed to. As such, they don't cost a clock cycle (its cycle cost is in the next datapath).



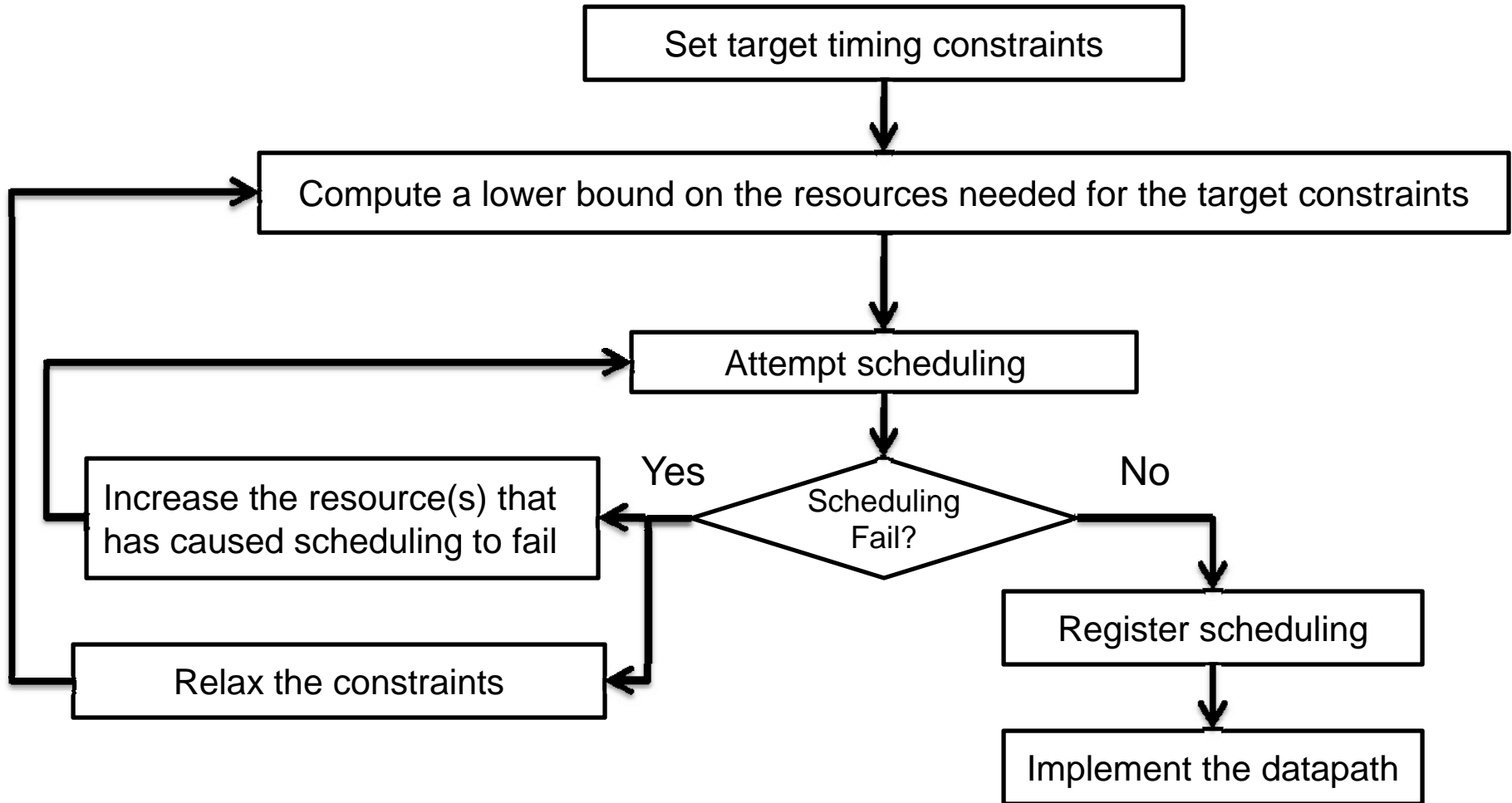
An execution unit operation. Based on clock period constraints, execution units can be **chained** (a multiplier output directly feeding an adder input without an intervening register) or **non-chained** (all inputs/outputs of execution units are registered).

What is minimum no. of clock cycles needed?

Assume that clock period constraint does not allow execution unit chaining (registers are between execution units). Minimum number of clock cycles will be longest path through the datapath.



Datapath Design Methodology



Resource Estimation

Given a clock cycle constraint (sample period), can estimate minimum number of needed resources.

Assume the minimum sample period of 4 clocks.

Minimum resource estimation is:

$$\# \text{ operations} / \# \text{ of clocks}$$

Minimum Resource estimation:

$$\# \text{ multipliers} = \# \text{ multiplies} / \# \text{ clocks} = 4/4 = 1$$

$$\# \text{ adders} = \# \text{ additions} / \# \text{ clocks} = 3/4 = 1$$

Minimum resource estimation is 1 multiplier, 1 adder.

Register estimation is tougher.

Need to store $X@1$, $X@2$, $X@3$ + four coefficients. Need at least 7 registers.

Scheduling

- **Scheduling** is mapping operations onto execution units. Use a scheduling table which lists clock cycles versus resources. Lets first just worry about execution units, and not about registers for now.

Cycle Start	Adder	Multiplier	IO
#1	Idle	$x@3*a3$ (N5)	Input x
#2	Idle	$x@2*a2$ (N4)	
#3	N7 op (N5+N4)	$x@1*a1$ (N3)	
#4	Idle	$x*a0$ (N2)	
Utilization	25%	100%	25%

Scheduling Failed!

- The scheduling failed! We were not able to schedule the adder operations represented by nodes N6 and N8.
 - The minimum resource estimation is a **lower bound**; may not find a schedule to fit it.
 - If scheduling fails, the two options:
 - a. Increase resources, keep same # of clock
 - b. Increase # of clocks, keep same number of resources
 - We want a minimum sample period, so do option (a).
 - The bottleneck is the multiplier. Lets add another multiplier.
-

Scheduling (2nd try)

Cycle Start	Adder	Multiplier 1	Multiplier 2	IO
#1	Idle	$x@3*a3$ (N5)	$x@2*a2$ (N4)	Input X
#2	N7 op (N5+N4)	$x@1*a1$ (N3)	$x*a0$ (N2)	
#3	N6 op (N3+N2)	Idle	Idle	
#4	N8 op (N7+N6)	Idle	Idle	
Utilization	75%	50%	50%	25%

Scheduling succeeds

Register Allocation

- At this point, need to allocate registers to save temporary results. At beginning of operation, we know that we need to have the values $a_0, a_1, a_2, a_3, x@3, x@2, x@1$ stored. So we need at least 7 registers.
- The registers holding a_0 - a_3 will not change value during the computation, so we will not consider them in our scheduling.
- Assume at Start:

$$RA = x@3, \quad RB = x@2, \quad RC = x@1$$

Register Scheduling (Clock 1)

Registers: $RA = x@3$, $RB = x@2$, $RC = x@1$

Clock 1:

Input X ??? Where to put this?

Add another register RD

Input X : $RD \leftarrow X$

$x@3 * a3$ (N5): $RA \leftarrow RA * a3$ (don't need $x@3$ after this, destroy RA)

$x@2 * a2$ (N4) $??? \leftarrow RB * a2$ (will need $x@2$ next time, can't destroy RB !)

Add another register RE

$x@2 * a2$ (N4) $RE \leftarrow RB * a2$ (will need $x@2$ next time, can't destroy RB !)

Scheduling these operations forced us to add two additional registers (RD , RE).

Register Scheduling (Clock #2)

Registers: RA = N5, RB=x@2, RC=x@1, RD=x, RE=N4

Clock 2:

N4 + N5 (N7): RA ← RE + RA (destroy RA, don't need N5 anymore)
x@1*a1 (N3): ??? ← RC * a1 (will need x@1 next time, can't destroy RC!!)

Look for a free register. Don't need RE (N4) after this clock cycle, use it.

x@1*a1 (N3): RE ← RC * a1 (store result in RE).
x*a0 (N2): ??? ← RD * a0 (will need "x" next time, can't destroy RD!)

Any free registers? None available. **Add another register.**

x*a0 (N2): RF ← RD * a0

Scheduling these operations forced us to add one more register (RF).

Register Scheduling (Clock 3 and Clock 4)

Registers: RA = N7, RB=x@2, RC=x@1, RD=x, RE=N3, RF=N2

Clock 3:

N6 op (N3+N2) RE ← RE + RF (destroy RE, don't need N3 anymore)

Registers: RA = N7, RB=x@2, RC=x@1, RD=x, RE=N6, RF=N2

Clock 4:

N8 op (N7+N6) Yout ← RA + RE (output is unregistered)

What about initial conditions for next sample period?

RA = x@3, RB=x@2, RC=x@1 ??

x@1 ← x RC ← RD
x@2 ← x@1 RB ← RC
x@3 ← x@2 RA ← RB

Note that x in this sample period becomes x@1 for the next sample period, x@1 becomes x@2, and x@2 becomes x@3. .

Final Requirements

- For sample period = 4 clocks:
 - 2 Multipliers, 1 adder
 - 10 registers (RA-RF, plus 4 registers for a0,a1,a2,a3)
 - Is this the best hardware allocation?
 - Maybe not, if we try harder may be able to remove a register or two.
 - Lets go with this and try to build the datapath
-

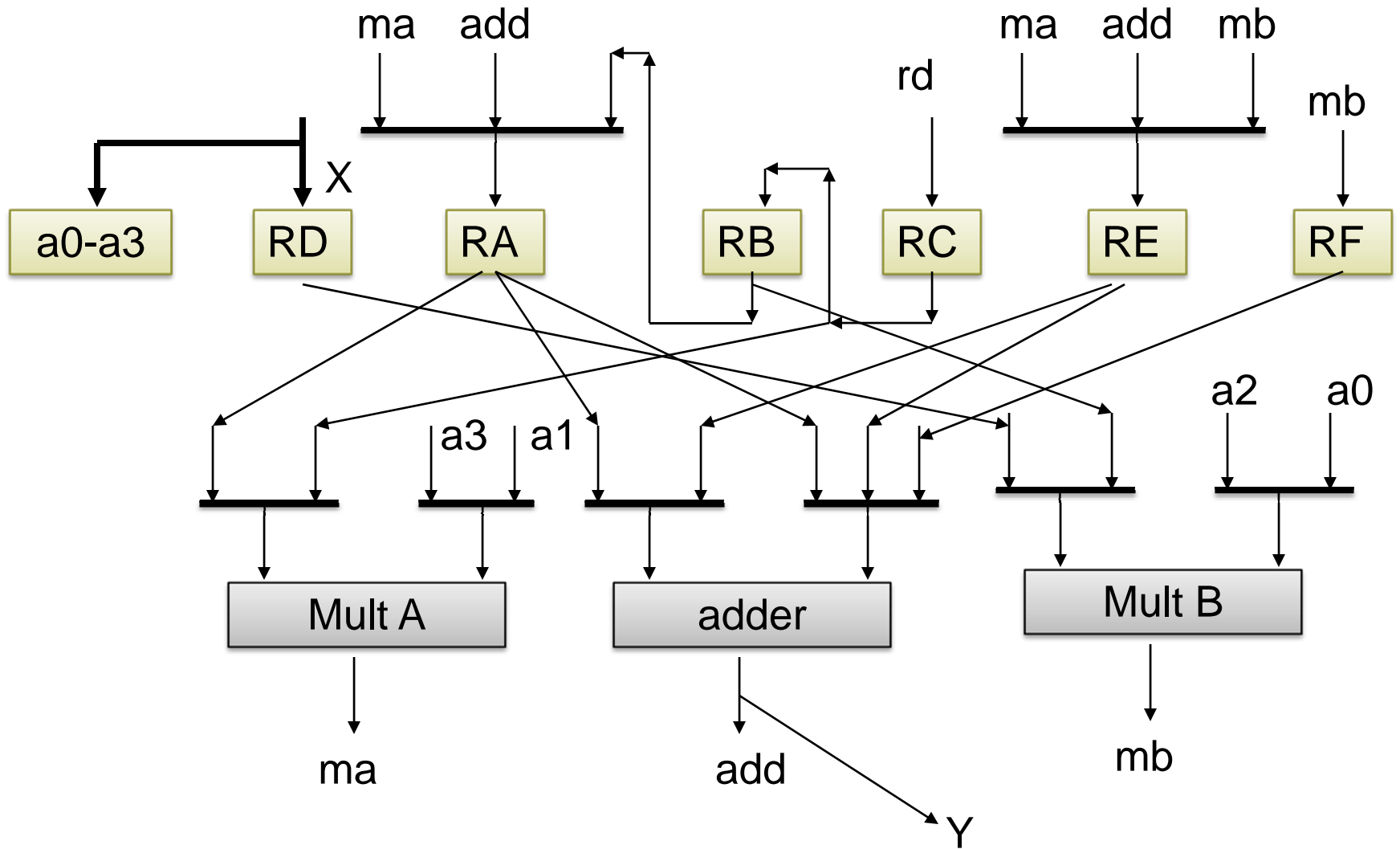
Datapath Execution Units Inputs

Mult A: Left sources: RA, RC Right sources: a3, a1
Mult B: Left sources: RB, RD Right sources: a2, a0
Adder: Left sources: RE, RA Right sources: RA, RF, RE

RA src: MultA, Adder, RB
RB src: RC
RC src: RD
RD src: X
RE src: Adder, Mult A, Mult B
RF src: Multiplier B

a0-a3 registers loaded from external databus X .

Datapath



Comments

- Saving on Execution units leads to lots of wiring and muxes because of the amount of execution unit sharing that is required
 - Could probably have reduced some of the mux requirements by more careful assignment of temporary values to registers
 - This datapath would require a FSM with four states; each state corresponding to a clock cycle.
 - Output of FSM would be mux select lines, register load lines
 - May need extra states if handshaking control (input_rdy, output_rdy) is required.
-

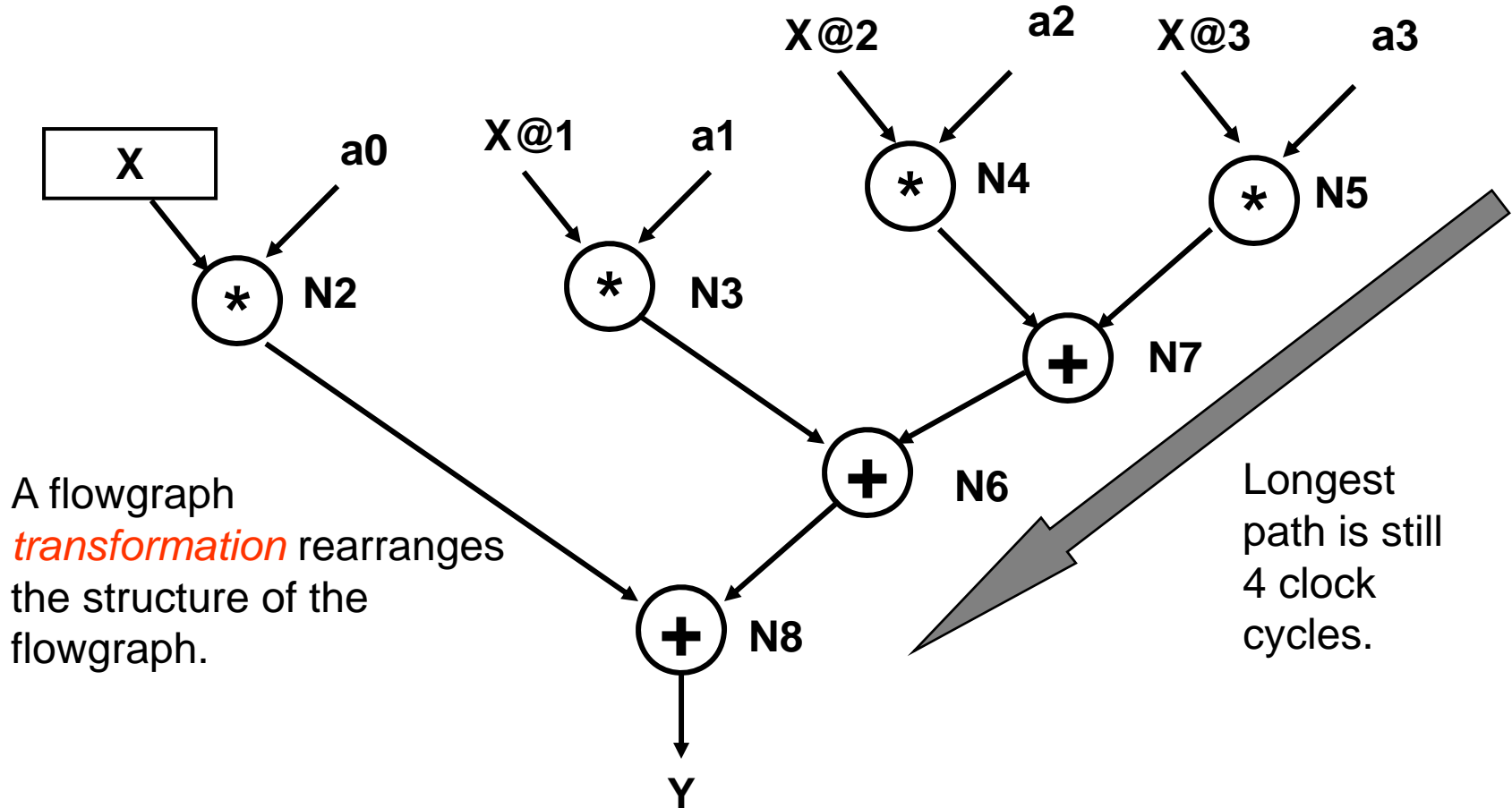
Scheduling

- Lets increase sample period from 4 to 5, and see if we can get rid of multiplier.

Cycle Start	Adder	Multiplier	IO
#1	Idle	$x@3*a3$ (N5)	Input X
#2	Idle	$x@2*a2$ (N4)	
#3	N7 op (N5+N4)	$x@1*a1$ (N3)	
#4	Idle	$x*a0$ (N2)	
#5	N6 op (N2 + N3)	Idle	

Scheduling Still Failed

Did not schedule Node 8 (N8). There should be a way in which we can make better use of the adder. Try restructuring the flowgraph.



Try again with Sample Period = 5

Cycle Start	Adder	Multiplier	IO
#1	Idle	$x@3 * a3$ (N5)	Input X
#2	Idle	$x@2 * a2$ (N4)	
#3	N7 op (N5 + N4)	$x@1 * a1$ (N3)	
#4	N6 op (N2 + N3)	$x * a0$ (N2)	
#5	N8 op (N2 + N6)	Idle	

Scheduling succeeds with new flowgraph

Flowgraph for Matrix Multiply

$$\begin{pmatrix} X' \\ Y' \\ Z' \\ W' \end{pmatrix} = \begin{pmatrix} T00 & T01 & T02 & T03 \\ T10 & T11 & T12 & T13 \\ T20 & T21 & T22 & T23 \\ T30 & T31 & T32 & T33 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ W \end{pmatrix}$$

$$X' = X^*T00 + Y^*T01 + Z^*T02 + W^*T03$$

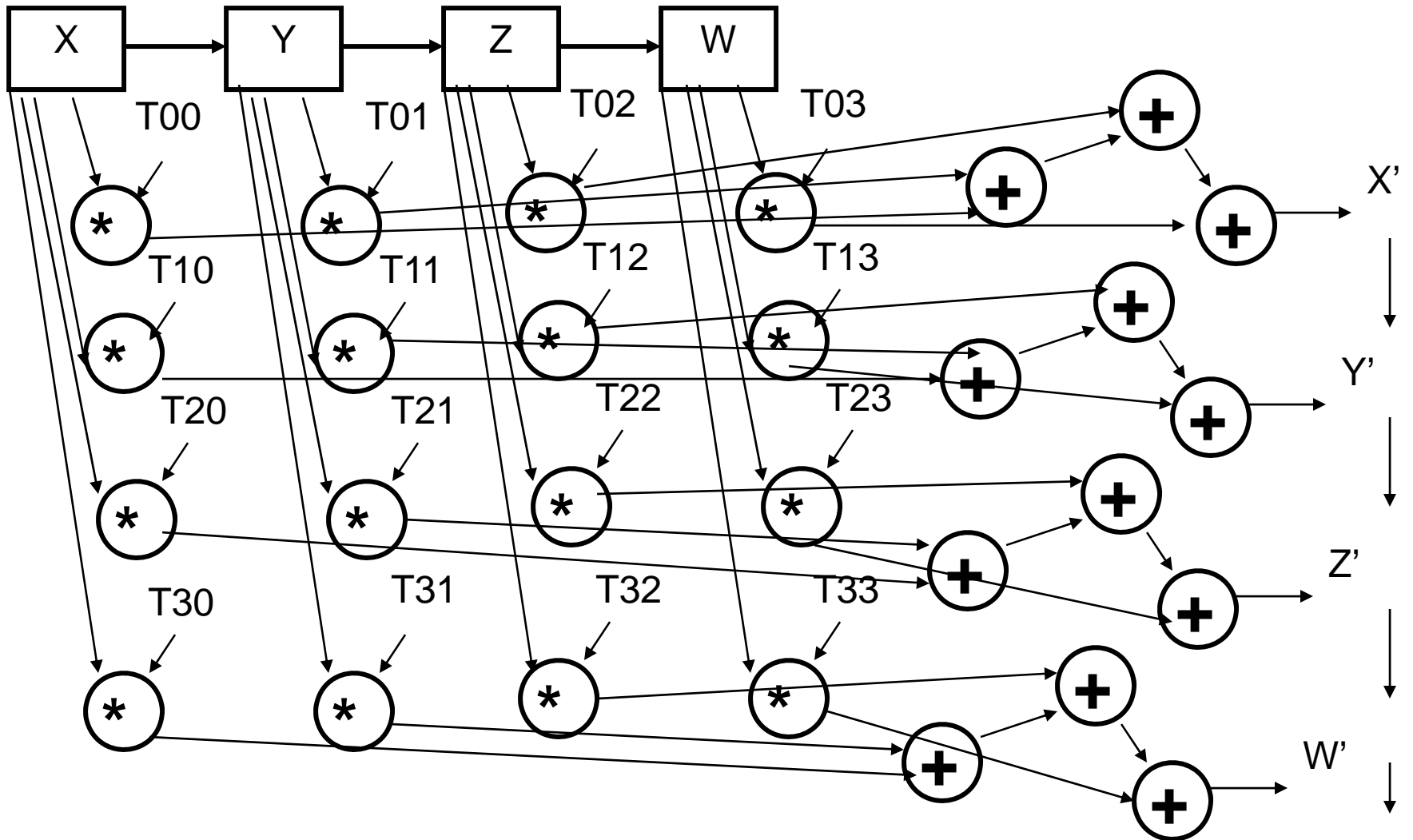
$$Y' = X^*T10 + Y^*T11 + Z^*T12 + W^*T13$$

$$Z' = X^*T20 + Y^*T21 + Z^*T22 + W^*T23$$

$$W' = X^*T30 + Y^*T31 + Z^*T32 + W^*T33$$

IO Constraint: Single input bus, single output bus

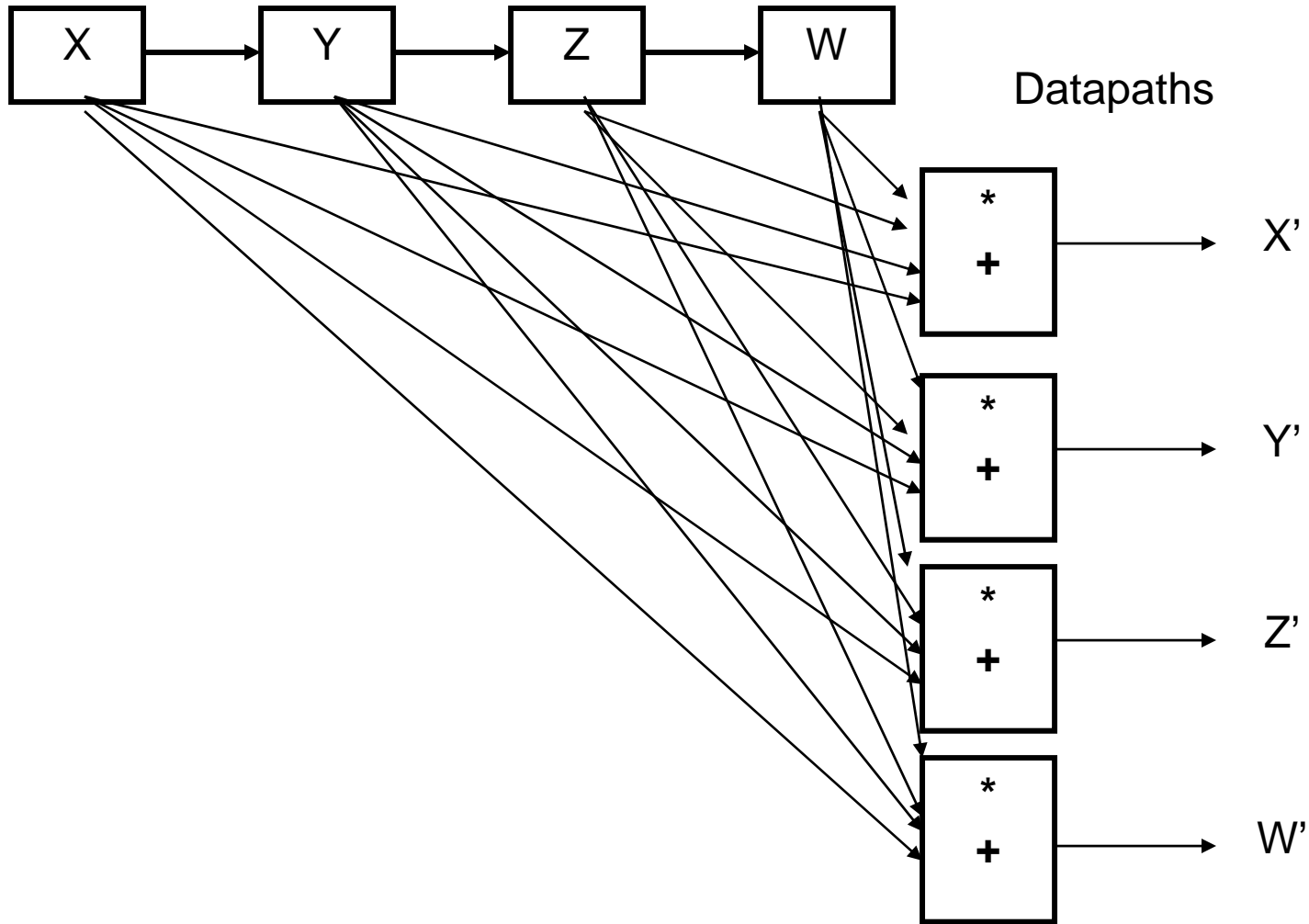
Flowgraph for Matrix Multiply



Comments on MM Flowgraph

- The main thing to notice about the graph is that you don't have to wait until you have X, Y, Z, W before you begin operations
 - Once you have X , you can do four multiply operations
 - Another thing to note is the symmetry and parallelism available
 - You could have four parallel datapaths, each one containing a multiplier and an adder, and produce X', Y', Z', W' from these four datapaths
-

Parallel Datapaths for MM



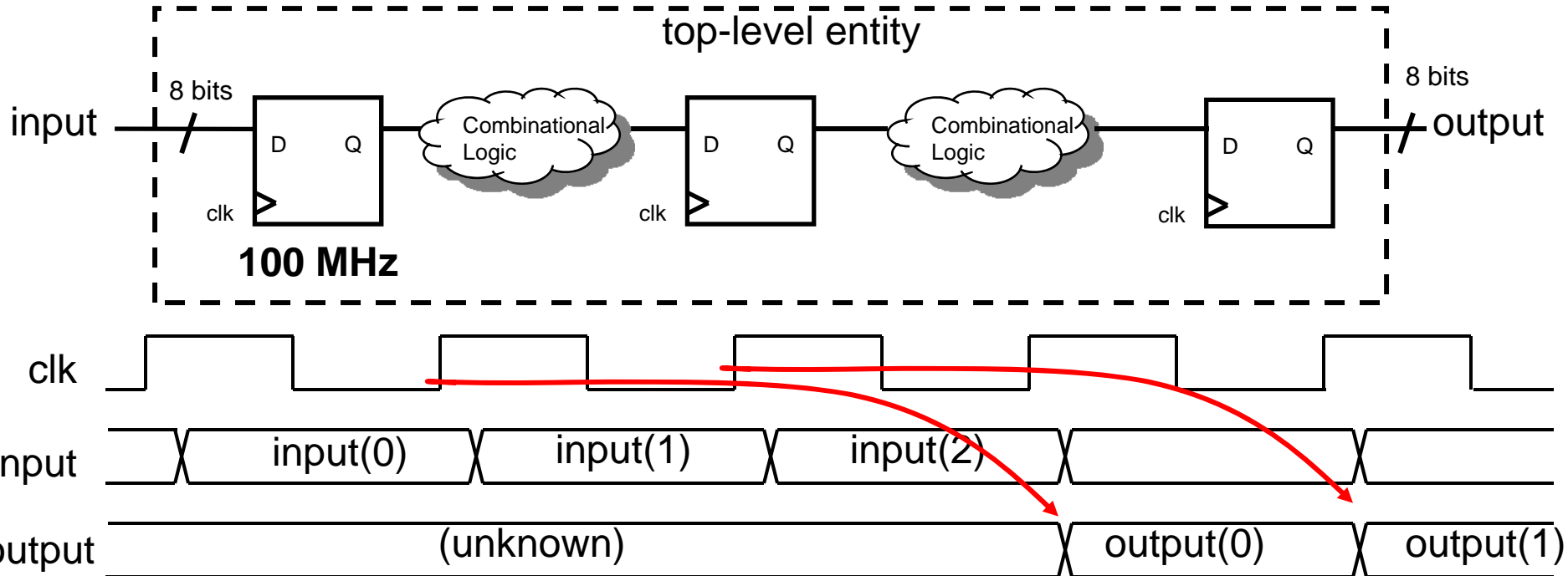
Timing Parameters

	definition	units
delay	time from point→point	ns
clock period T	rising edge →rising edge of clock	ns
clock frequency	$\frac{1}{\text{clock period}}$	MHz

latency	time from input→output	ns
----------------	------------------------	----

throughput	#output bits/time unit	Mbits/s
-------------------	------------------------	---------

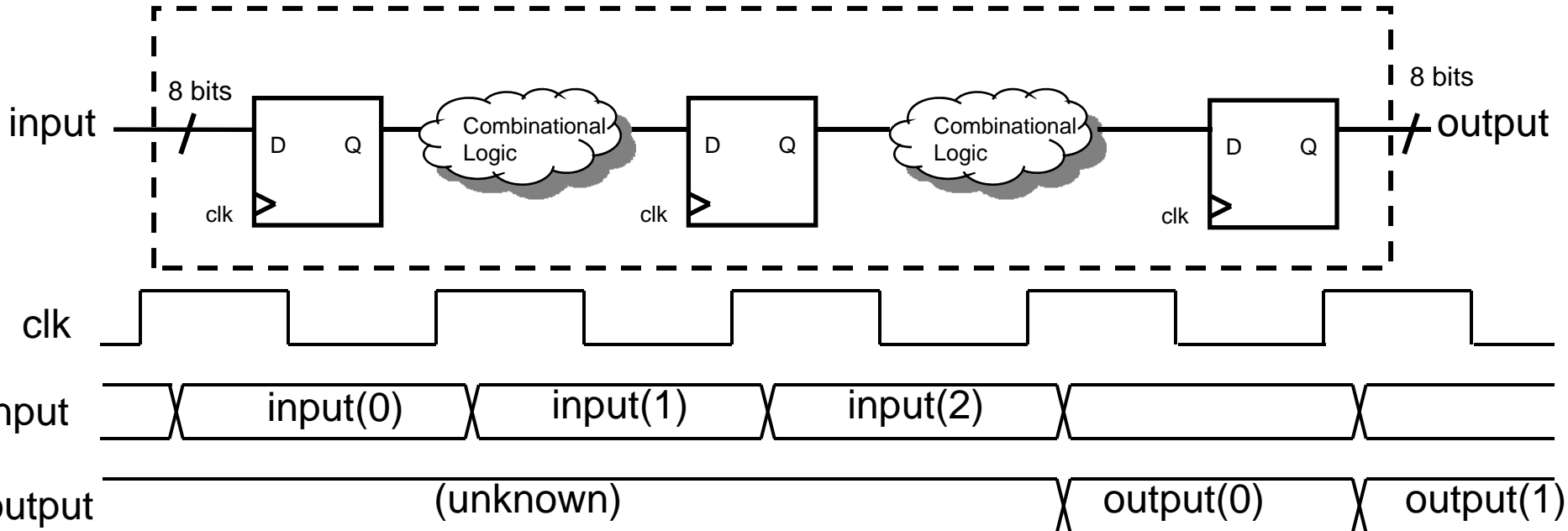
Latency



- Latency is the time between input(n) and output(n)
 - i.e. time it takes from first input to first output, second input to second output, etc.
 - Latency is usually constant for a system (but not always)
 - Also called input-to-output latency
- **Count the number of rising edges of the clock!**
 - In this example, 3 rising edges from input to output → latency is 3 cycles
- Latency is measured in clock cycles (then translated to seconds)
 - In this example, say clock period is 10 ns, then latency is 30 ns

Throughput

top-level entity



- **Throughput = (bits per output sample) / (time between consecutive output samples)**

- Bits per output sample:

- In this example, 8 bits per output sample

- Time between consecutive output samples: clock cycles between output(n) to output(n+1)

- Can be measured in clock cycles, then translated to time

- In this example, time between consecutive output samples = 1 clock cycle = 10 ns

- Throughput = (8 bits per output sample) / (10 ns) = 0.8 bits / ns = 800 Mbits/s

Initiation Rate

■ Initiation Rate

- Rate at which new values are input
 - Defined in either seconds or # of clocks
- In the Matrix Multiply example: Initiation rate will be number of clocks from inputting X for one set of (X, Y, W, Z) to inputting the next X for a new set of (X, Y, W, Z)
-

Defining Initiation Rate, Latency

Input X0 ←

Input Y0 (compute)

Input Z0 (compute)

Input W0 (compute)

compute

compute

compute

compute

Output X0'

Output Y0'

Output W0'

Output Z0'

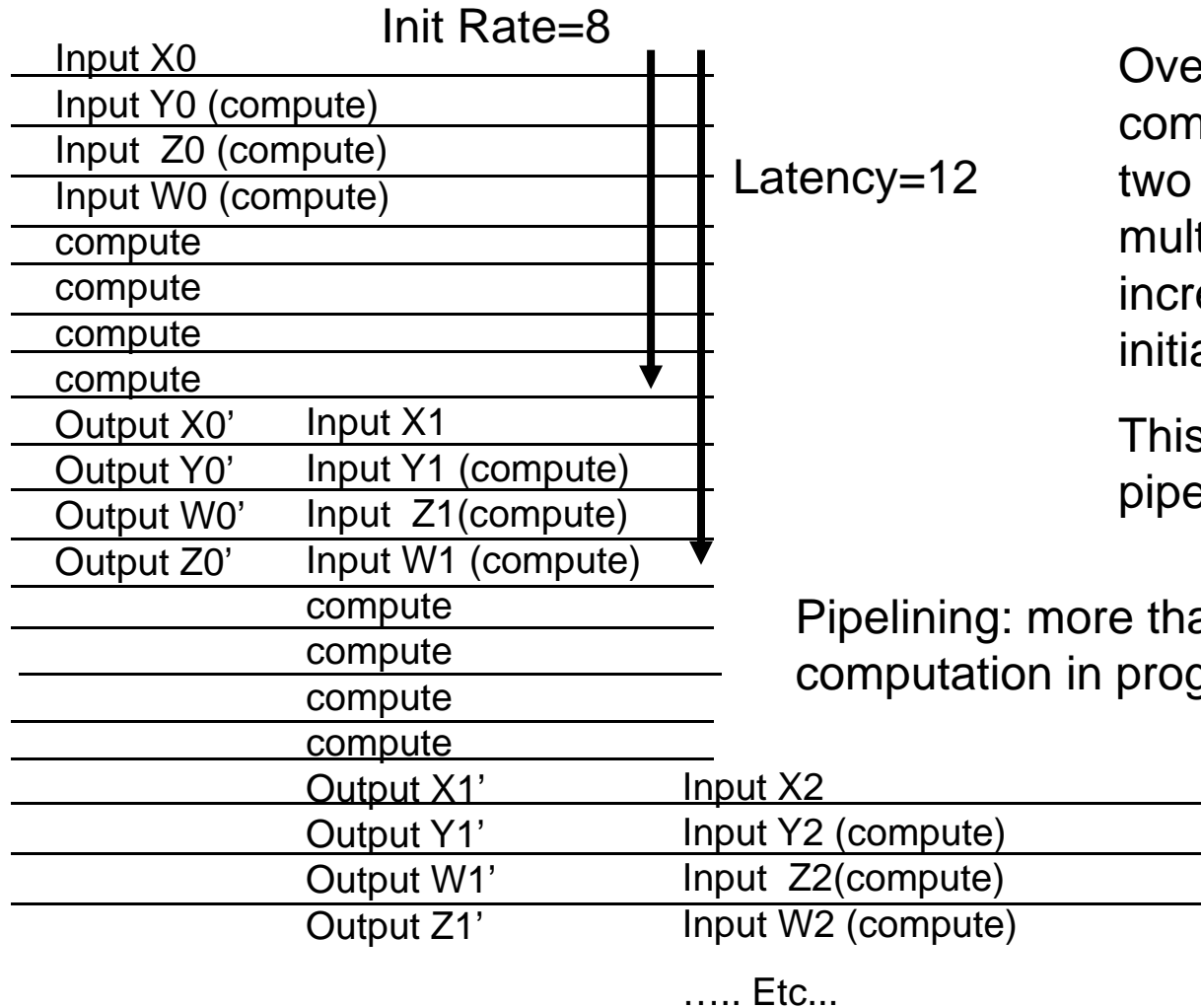
Input X1 ←

Input Y1..

Initiation Rate = 12

Latency = 12

Reducing Init Rate



Latency=12

Overlapping computation of two matrix multiplies to increase initiation rate.

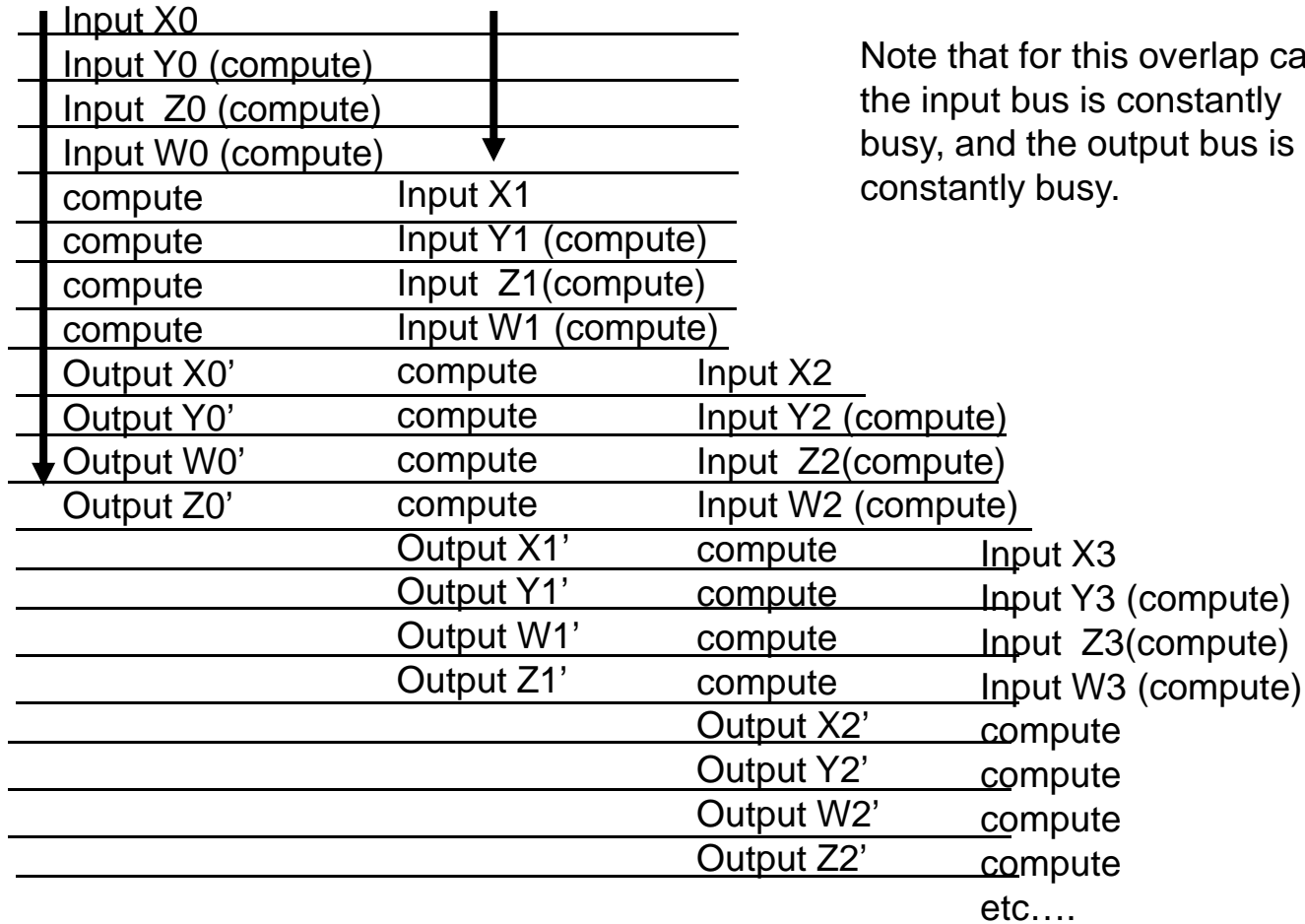
This is a form of pipelining!!!

Pipelining: more than one computation in progress.

Reducing Init Rate

Latency=12

Init Rate=4



Note that for this overlap case the input bus is constantly busy, and the output bus is constantly busy.