ESE5320: System-on-a-Chip Architecture

Day 7: September 26, 2022

Pipelining

1

Previously

- Pipelining in the large
  - Not just for gate-level circuits
- Throughput and Latency
- Pipelining as a form of parallelism

2

Today

Pipelining details (for gates, primitive ops)

- Systematic Approach (Part 1)
- Justify Operator and Interconnect Pipelining (Part 2)
- Loop Bodies
- Cycles in the Dataflow Graph (Part 3)
- C-slow [supplemental recording] (Part 4)

3

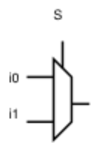
Message

- Pipelining is an efficient way to reuse hardware to perform the **same** set of operations at high throughput

4

Multiplexer Gate

- MUX
  - When S=0, output=i0
  - When S=1, output=i1



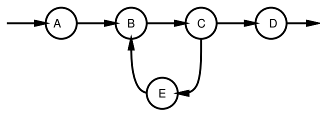
S	i0	i1	Mux2(S,i0,i1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

5

Cycle

Two uses of term in this lecture:

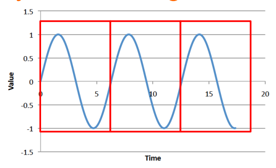
- Repetitive waveform
  - E.g. sine wave or square wave
- Graph cycle



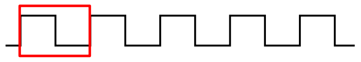
6

Waveform Cycles

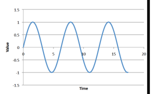
- How many cycles showing of sine wave?



- How many cycles of square wave?



Waveform Cycles

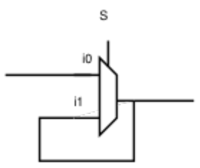


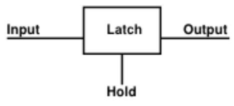
- How many cycles showing of sine wave?
- How many cycles of square wave?

- Note: clock on which we pipeline is a square wave
  - Talk about what happens in a clock cycle
  - Talk about number of clock cycles

Latch

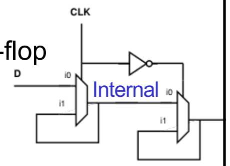
- Element that can hold a previous value of an input

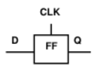


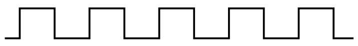


Register

- Use a pair to create a flip-flop
  - Also call register
- What happens when
  - CLK is low (0) ?
  - CLK is high (1) ?
  - CLK transitions from 0 to 1?
- What output Q until next 0 to 1 CLK transition?

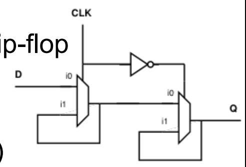


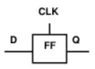




Register

- Use a pair to create a flip-flop
  - Also call register
- Sample D input on 0→1 transition of clock (CLK)
- Never an open path from D→Q
  - One of the mux latches always in hold state

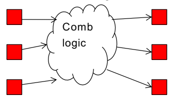




Synchronous Circuit Discipline

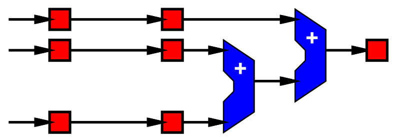
- Registers that sample inputs at clock edge and hold value throughout clock period

- Compute from registers-to-registers
- Clock Cycle time large enough for longest logic path between registers
- Min cycle = Max path delay between registers



Preclass 1

- Delay between registers as shown?



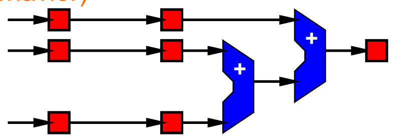
Penn ESE5320 Fall 2022 -- DeHon

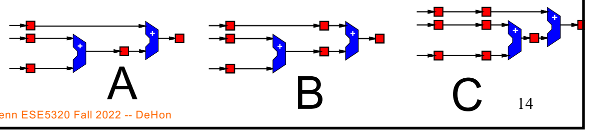
13

13

Preclass 1

- Move registers so can clock at adder delay?
  (which is of choices at bottom is correct = same behavior)





Penn ESE5320 Fall 2022 -- DeHon

14

14

Pipeline Reuse

- Lower delay between clocks
  - Higher clock rate
  - Higher potential throughput
  - Faster we reuse our logic
  - More capacity get out of design
    - Assuming registers cheap in area and time overhead
      - T_{setup}, $T_{clk\text{->}q}$ ~ 20ps, T_{add} ~ 500ps
      - Registers ~ 10 transistors/bit
      - Adder ~ 40—50 transistors/bit

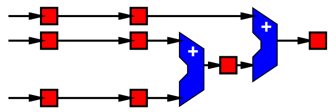
Penn ESE5320 Fall 2022 -- DeHon

15

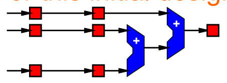
15

Preclass 2: What Happens?

- What would be wrong with this pipelining?



- For this initial design:

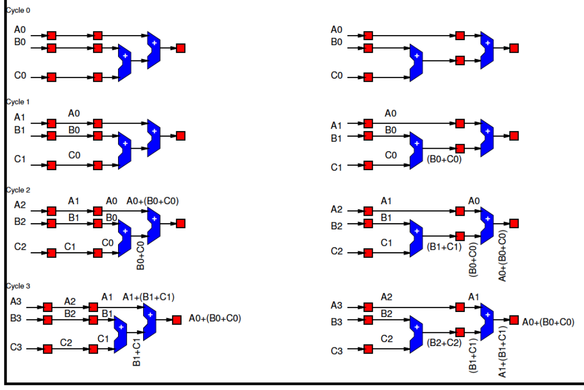


Penn ESE5320 Fall 2022 -- DeHon

16

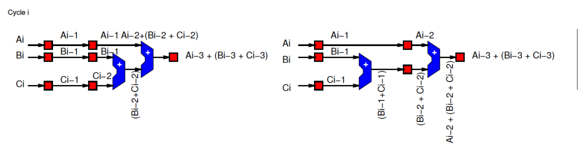
16

Behavior



17

Equations

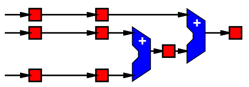


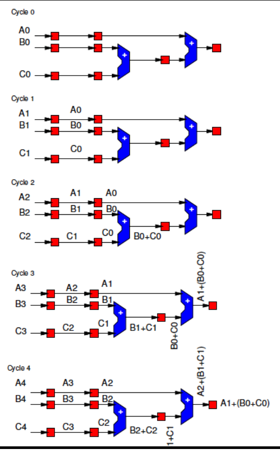
Penn ESE5320 Fall 2022 -- DeHon

18

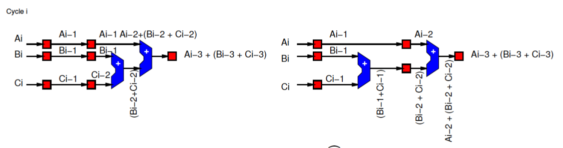
18

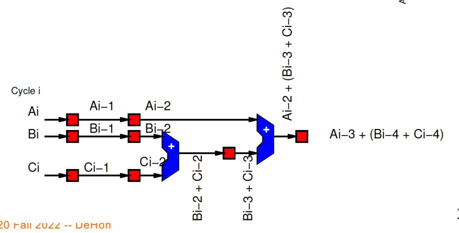
Behavior
Add Register





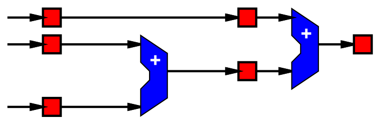
Equations





Note Registers on Links

- Some links end up with multiple registers.
- Why?

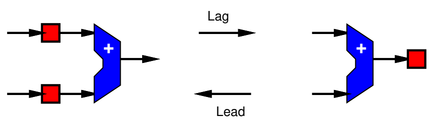


Consistent Pipelining

- Makes sure a consistent input set arrives at each gate/operator
  - Don't get mixing between input sets

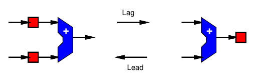
Legal Register Moves

- Retiming Lag/Lead

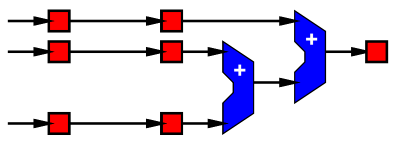


- Lag: remove register every input
  add register every output
- Lead: remove register every output
  add register every input

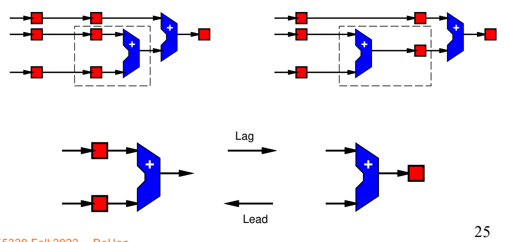
Preclass 1



- Retime using Lead/Lag



Preclass 1 (revisited)



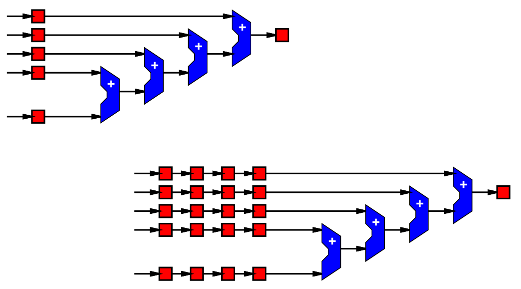
25

Add Registers and Move

- If we're willing to add pipeline delay
  - Add any number of pipeline registers at input
  - Move registers into circuit to reduce cycle time
    - Reduce max delay between registers

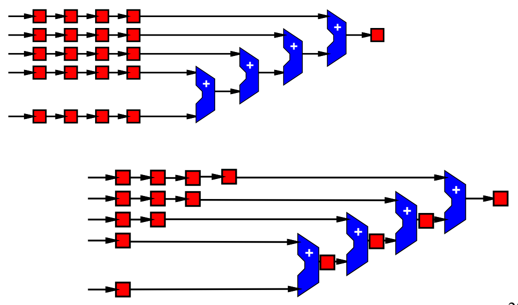
26

Add Registers at Input



27

Add Registers at Input and Retime



28

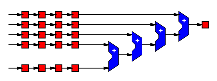
Add Register and Retime

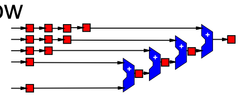
- Add chain of registers on every input
- Retime registers into circuit
  - Minimizing delay between registers

29

Add Registers and Retime

- Lets us think about behavior
  - What the pipelining is doing to cycles of delay
- Separate from details of how redistribute registers
- Behavioral equivalence between the registers-at-front and properly retimed version of circuit





30

Justify Pipelining

(or composing pipelined operators)
Part 2

Handling Pipelined Operators

- Given a pipelined operator
  - (or a pipelined interconnect)
- Discipline of picking a frequency target and designing everything for that
  - May be necessary to pipeline operator since its delay is too high
- Due to hierarchy
  - Pipelined this operator and now want to use it as a building block

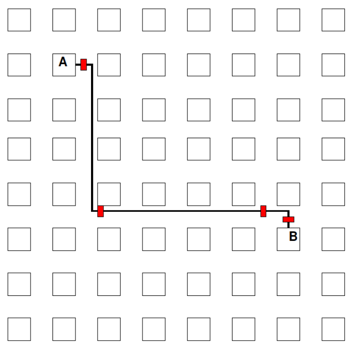
Examples

- Run at 500MHz
- Floating-point unit that takes 9ns
  - Can pipeline into 5, 2ns stages
- Multiplier that takes 6ns
- Memory can access in 2ns
  - Only if registers on address/inputs and output
  - i.e. exist in own clock stage

Interconnect Delay

- Chips >> Clock Cycles
- May have chip 100s of Operators wide
- May only be able to reach across 10 operators in a 2ns cycle
- Must pipeline long interconnect links

Interconnect Example

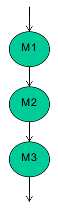


Methodology: Pipelined Operator Graph

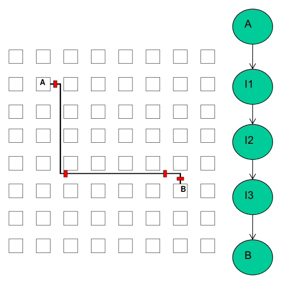
- Start with logical, unpipelined graph
- Treat each pipelined operator as a set of unit-delay operators of mandatory depth
- Treat each interconnect pipeline stage as a unit-delay buffer
- Add registers at input
- Retime into graph

Model

- 3-stage Multiplier



- Interconnect Delay



37

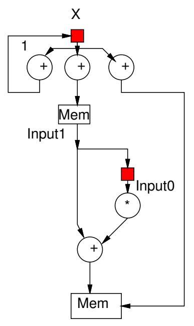
Pipeline Loop

(and use for justify pipeline example)

38

Preclass 4

- Logical (unpipelined) dataflow graph for loop body



39

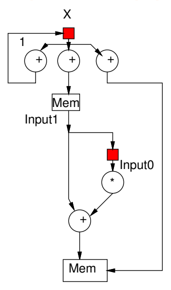
Example Operators

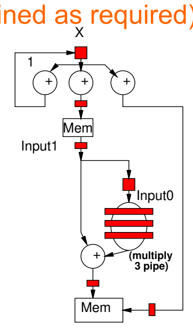
- Operator and Interconnect delays
  - Multiplier 3 cycles
  - Reading from Input array
    - Memory op is cycle after computing address
    - Takes one cycle delay bring data back to multiplier (or adder)

40

Illustrate Need

- What happens if just use graph as is (with operators pipelined as required)?

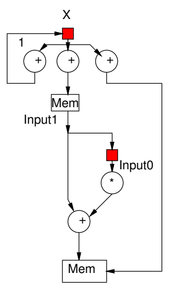


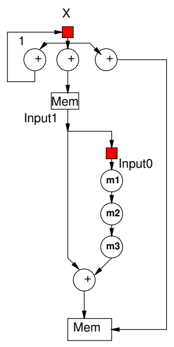


41

Model Graph

- Revised graph for modeling

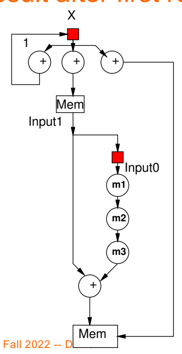




42

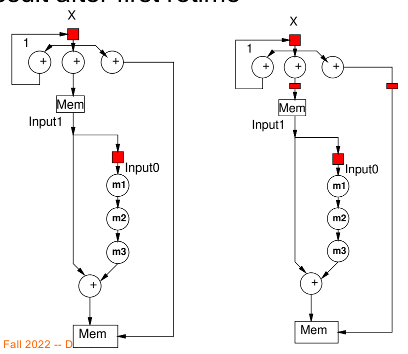
Pipeline Graph

- Result after first retime (top register)



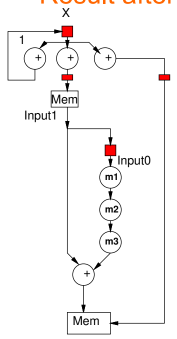
Pipeline Graph

- Result after first retime



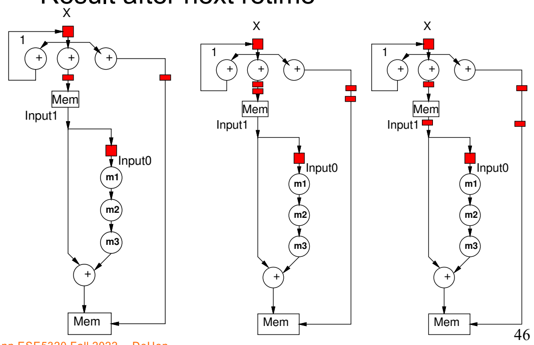
Pipeline Graph

- Result after next retime (top register)?



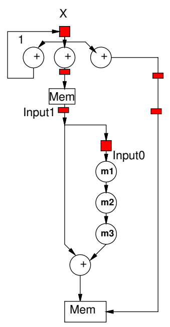
Pipeline Graph

- Result after next retime



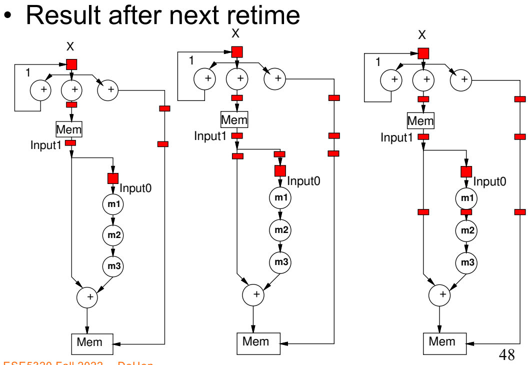
Pipeline Graph

- Result after next retime (top register)?



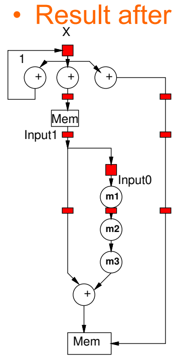
Pipeline Graph

- Result after next retime



Pipeline Graph

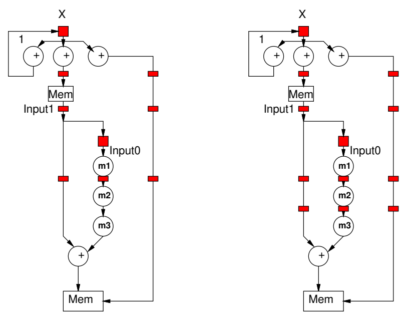
- Result after next retime (top register)?



49

Pipeline Graph

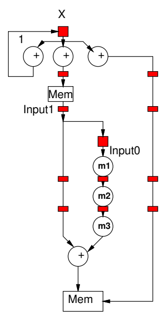
- Result after next retime



50

Pipeline Graph

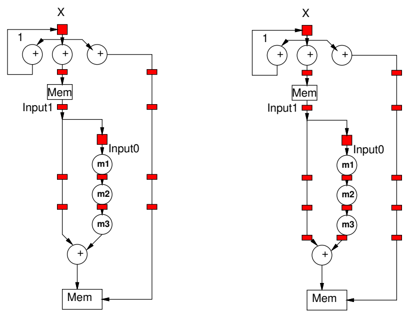
- Result after next retime (top register)?



51

Pipeline Graph

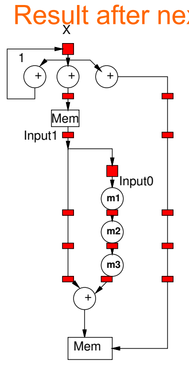
- Result after next retime



52

Pipeline Graph

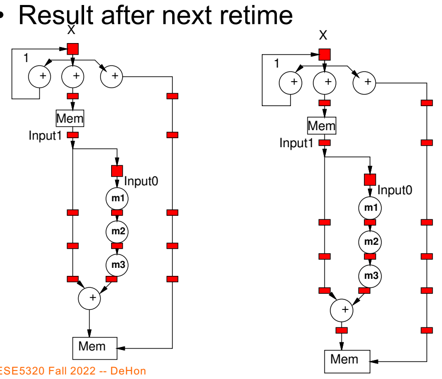
- Result after next retime (top register)?



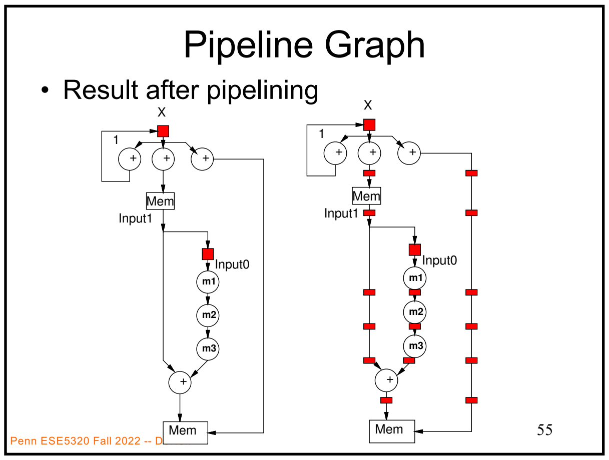
53

Pipeline Graph

- Result after next retime



54



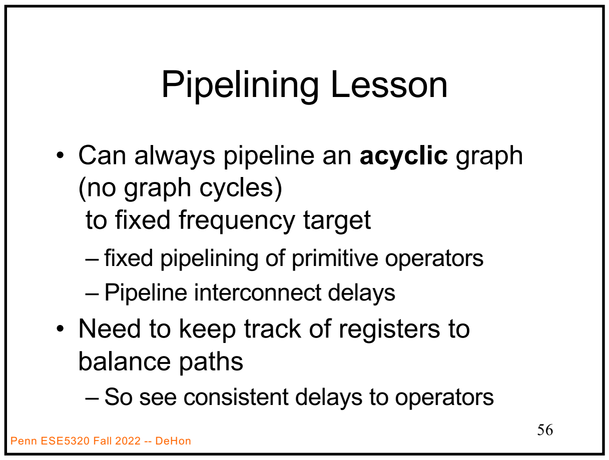
Pipeline Graph

- Result after pipelining

Penn ESE5320 Fall 2022 -- DeHon

55

55



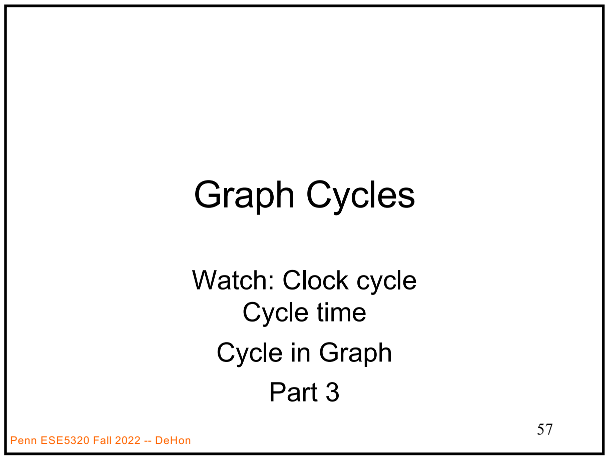
Pipelining Lesson

- Can always pipeline an **acyclic** graph (no graph cycles)
  to fixed frequency target
  - fixed pipelining of primitive operators
  - Pipeline interconnect delays
- Need to keep track of registers to balance paths
  - So see consistent delays to operators

Penn ESE5320 Fall 2022 -- DeHon

56

56



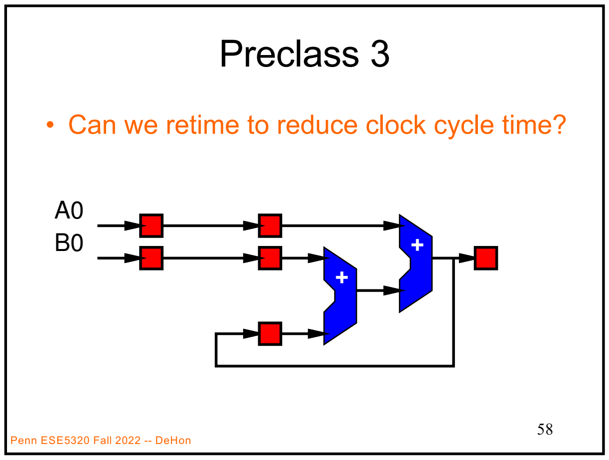
Graph Cycles

Watch: Clock cycle
Cycle time
Cycle in Graph
Part 3

Penn ESE5320 Fall 2022 -- DeHon

57

57



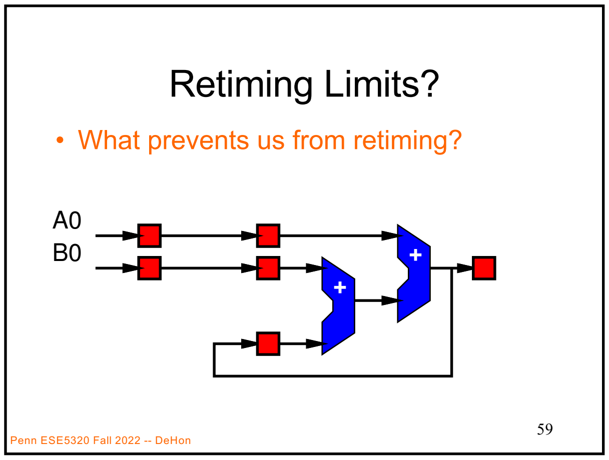
Preclass 3

- Can we retime to reduce clock cycle time?

Penn ESE5320 Fall 2022 -- DeHon

58

58



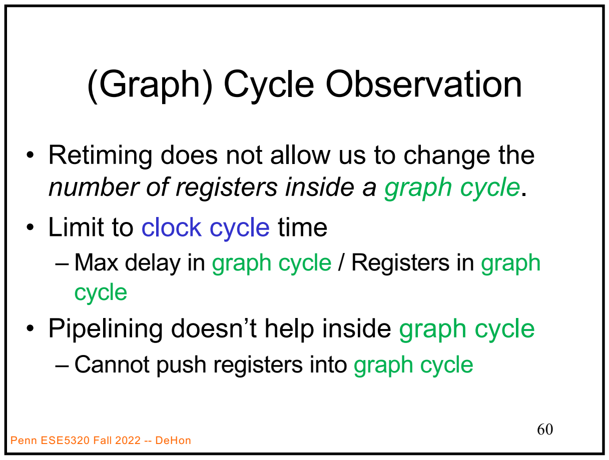
Retiming Limits?

- What prevents us from retiming?

Penn ESE5320 Fall 2022 -- DeHon

59

59



(Graph) Cycle Observation

- Retiming does not allow us to change the *number of registers inside a graph cycle*.
- Limit to clock cycle time
  - Max delay in graph cycle / Registers in graph cycle
- Pipelining doesn't help inside graph cycle
  - Cannot push registers into graph cycle

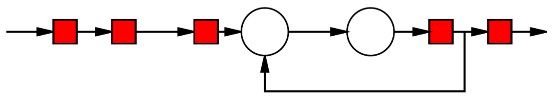
Penn ESE5320 Fall 2022 -- DeHon

60

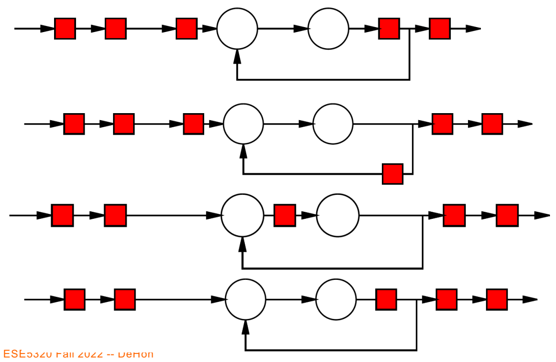
60

Simple Graph Cycle

- Delay of graph cycle?
- Registers in graph cycle?
- What happens to graph cycle if try to apply lead/lag?

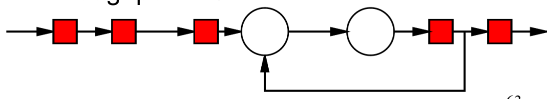


Retiming



Initiation Interval (II)

- Cyclic dependencies in a dataflow graph can limit throughput
- Due to data-dependent cycles in graph,
  - May not be able to initiate a new computation on every clock cycle
- II – clock cycles (delay) before can initiate
- Throughput = 1/II

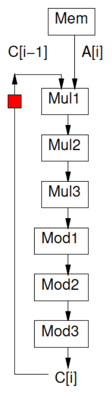


Loop

- Consider
  - [multiply and mod each take 3 cycles]
- For (i=0;i<N;i++)
  C[i]=(C[i-1]*A[i])%N;

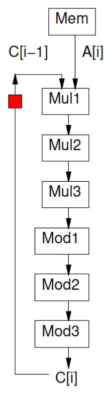
Loop

- For (i=0;i<N;i++)
  C[i]=(C[i-1]*A[i])%N;



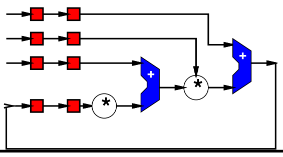
Loop

- For (i=0;i<N;i++)
  C[i]=(C[i-1]*A[i])%N;
- Initiation Interval?

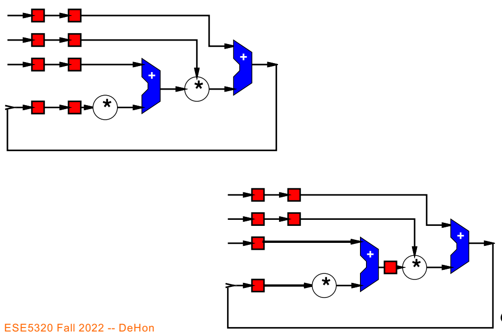


Initiation Interval

- Delay Around graph cycle?
  - Assume multiply 3, add 1
- Registers in graph cycle?
- Retiming clock cycle bound = II ?
- Achievable?

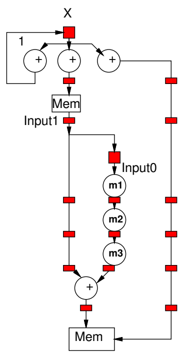


Retimed



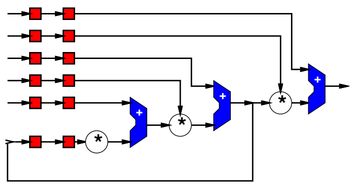
II and Latency

- Actually is a cycle
  - II?
  - Latency?

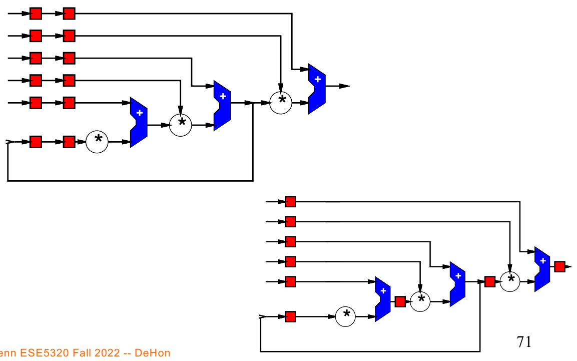


II and Latency

- II? (assume willing to pipeline inputs)
- Latency?



II and Latency



Lesson

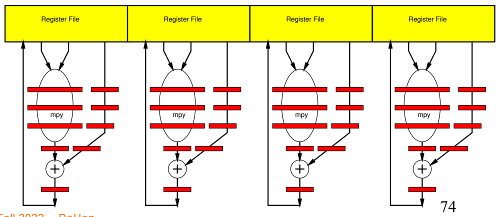
- Cyclic dependencies limit throughput on single task or data stream
  - Cycle-length / registers-in-cycle

Vector Pipelines

- Data Parallel Vector Operations are interesting even when
  Vector Lanes<Vector Length
- Within Vector operation, data parallel so no cyclic dependencies
  - So get an II=1 issuing Vector Lane operations
  - May have data dependences between Vector operations

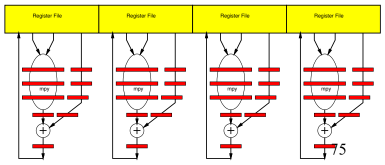
Vector Pipeline Example

for (int i=0;i<32; i++)
    d[i]=a[i]*b[i]+c[i]



Dependence between Vector Operations

for (int i=0;i<32; i++)
    d[i]=a[i]*b[i]+c[i]
for (int i=0;i<32; i++)
    g[i]=e[i]*f[i]+d[i]



Big Ideas

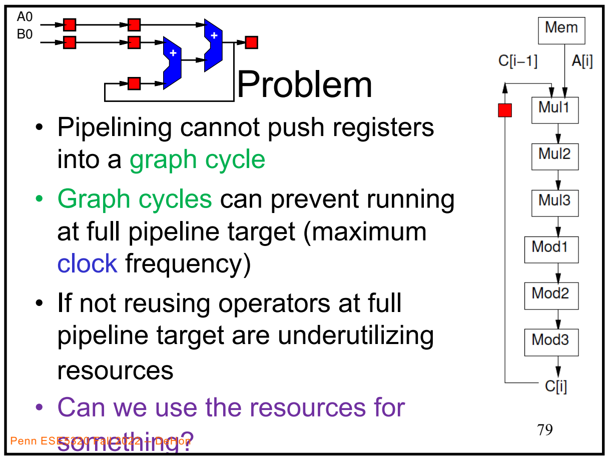
- Pipeline computations to reuse hardware and maximize computational capacity
- Can compose pipelined operators and accommodate fixed-frequency target
  - Be careful with data retiming
- Graph cycles limit pipelining on single stream – II (Initiation Interval)
- C-slow to share hardware among multiple, data-parallel streams (part 4)

Admin

- Remember Feedback form
  - Including HW3
- Reading for Day 8 on web
- HW4 due Friday

C-Slow

(See uploaded recording)
Part 4

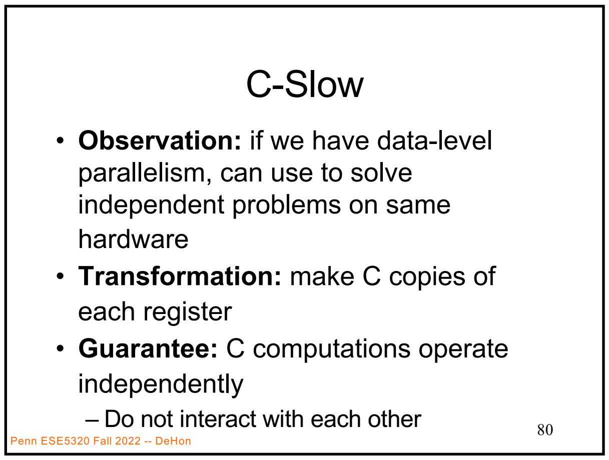


A0
B0
Mem
C[i–1]
A[i]
Mul1
Mul2
Mul3
Mod1
Mod2
Mod3
C[i]

Problem

- Pipelining cannot push registers into a graph cycle
- Graph cycles can prevent running at full pipeline target (maximum clock frequency)
- If not reusing operators at full pipeline target are underutilizing resources
- Can we use the resources for something?

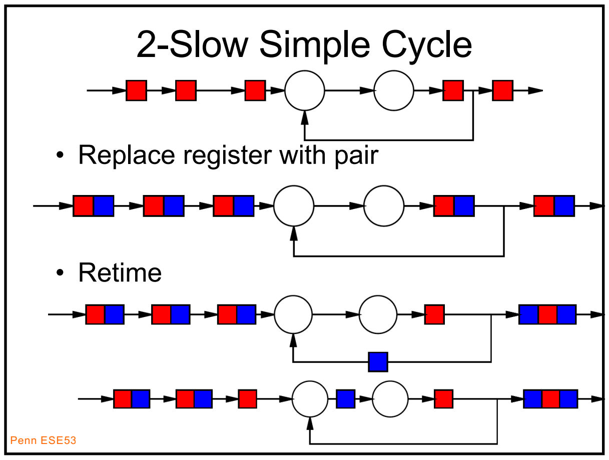
79



C-Slow

- **Observation:** if we have data-level parallelism, can use to solve independent problems on same hardware
- **Transformation:** make C copies of each register
- **Guarantee:** C computations operate independently
  - Do not interact with each other

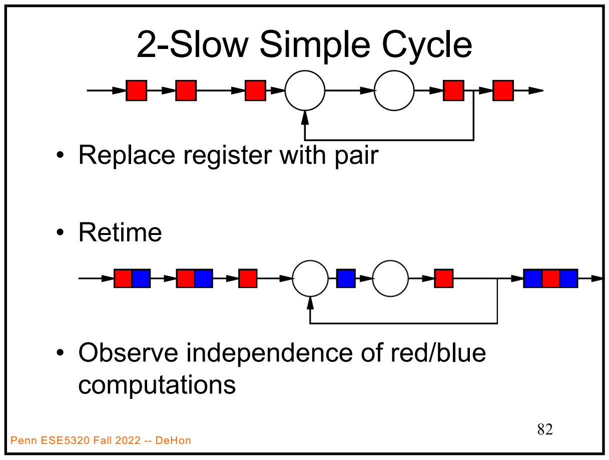
80



2-Slow Simple Cycle

- Replace register with pair
- Retime

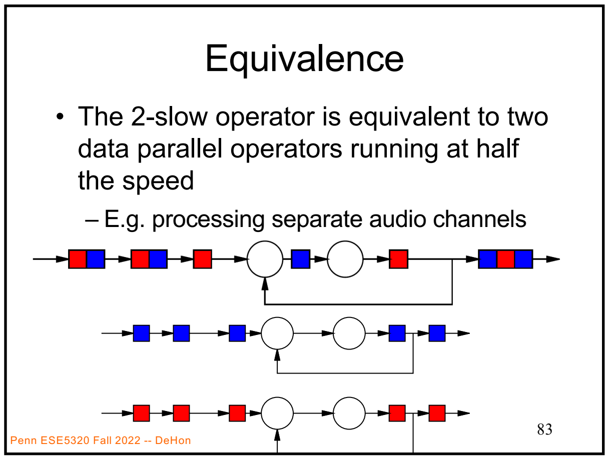
81



2-Slow Simple Cycle

- Replace register with pair
- Retime
- Observe independence of red/blue computations

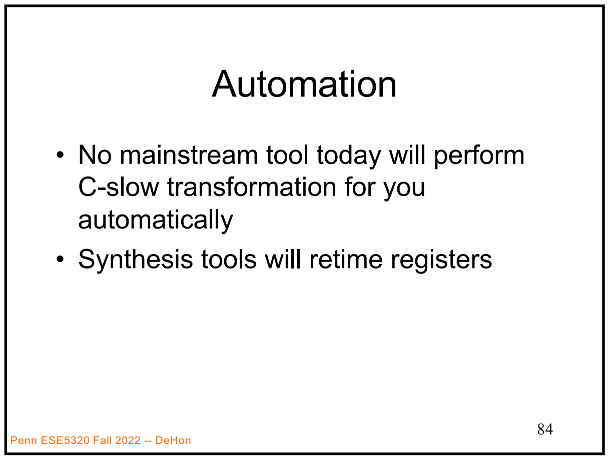
82



Equivalence

- The 2-slow operator is equivalent to two data parallel operators running at half the speed
  - E.g. processing separate audio channels

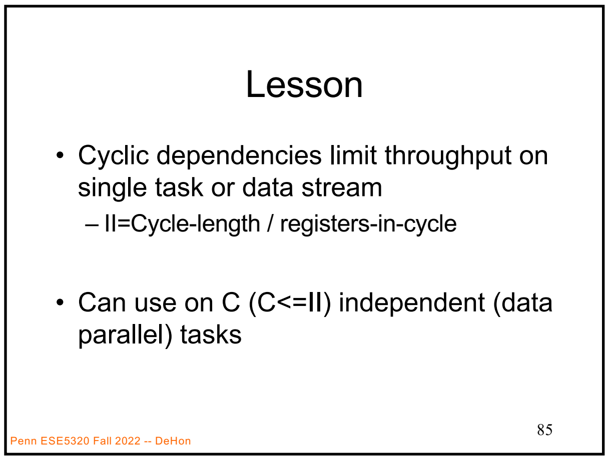
83



Automation

- No mainstream tool today will perform C-slow transformation for you automatically
- Synthesis tools will retime registers

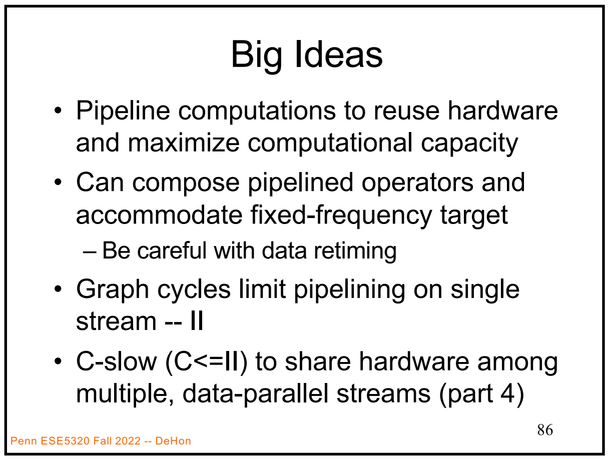
84



Lesson

- Cyclic dependencies limit throughput on single task or data stream
  - II=Cycle-length / registers-in-cycle
- Can use on C (C<=II) independent (data parallel) tasks

85



Big Ideas

- Pipeline computations to reuse hardware and maximize computational capacity
- Can compose pipelined operators and accommodate fixed-frequency target
  - Be careful with data retiming
- Graph cycles limit pipelining on single stream -- II
- C-slow (C<=II) to share hardware among multiple, data-parallel streams (part 4)

86