

ESE532: System-on-a-Chip Architecture

Day 7: September 24, 2018
Pipelining



Previously

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

Today

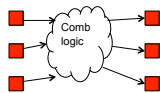
- Pipelining details (for gates, primitive ops)
- Systematic Approach
 - Justify Operator and Interconnect Pipelining
 - Loop Bodies
 - Cycles if the Dataflow Graph
 - C-slow [bonus if have time]

Message

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

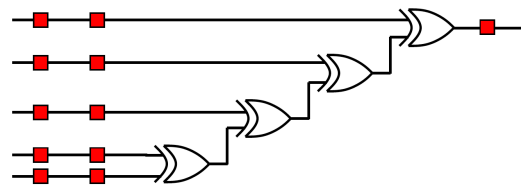
Synchronous Circuit Discipline

- Registers that sample inputs at clock edge and hold value throughout clock period
- Compute from registers-to-registers
- Cycle time large enough for longest logic path between registers
- Min cycle = Max path delay between registers



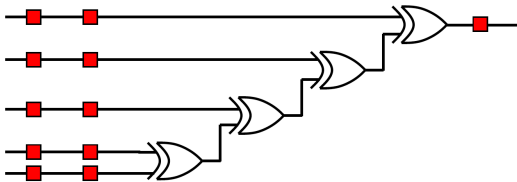
Preclass 1

- Delay between registers as shown?



Preclass 1

- Move registers so can clock at 2-xor-delays?

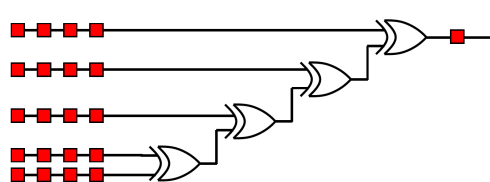


Penn ESE532 Fall 2018 -- DeHon

7

Preclass 2

- Move registers so can clock at xor-delays?



Penn ESE532 Fall 2018 -- DeHon

8

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



Penn ESE532 Fall 2018 -- DeHon

9

Levelize-and-Cut Pipelining

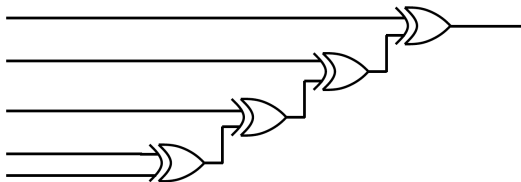
- Assuming willing to pipeline into as many cycles as necessary
- Draw circuit in levels by delay from input
 - Level= $\max(\text{level of inputs}) + \text{delay operator}$
- Given cycle time target
- Count forward to target
- Bisect circuit adding register on every cut link
- Repeat count-and-bisect until done

Penn ESE532 Fall 2018 -- DeHon

10

Apply to xor-chain

- Levelize
- Cut for 2 unit cycle target
 - ... for 1 unit



Penn ESE532 Fall 2018 -- DeHon

11

Note Registers on Links

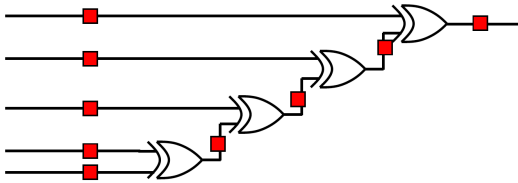
- Some links end up with multiple registers.
- Why?

Penn ESE532 Fall 2018 -- DeHon

12

What Happens?

- What would be wrong with this pipelining?



Penn ESE532 Fall 2018 -- DeHon

13

Consistent Pipelining

- Levelize-and-cut guarantees path from input to any gate input passes through the **same** number of registers
- Makes sure a consistent input set arrives at each gate/operator
 - Don't get mixing between input sets

Penn ESE532 Fall 2018 -- DeHon

14

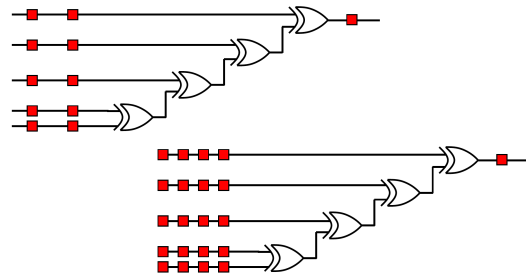
Add Registers and Move

- 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

Penn ESE532 Fall 2018 -- DeHon

15

Add Registers at Input

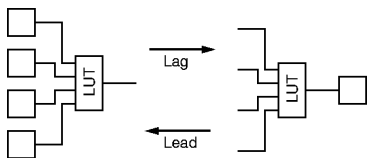


Penn ESE532 Fall 2018 -- DeHon

16

Legal Register Moves

- Retiming Lag/Lead



Penn ESE532 Fall 2018 -- DeHon

17

Add Register and Retime

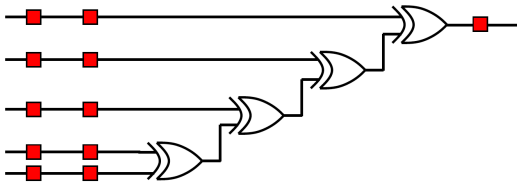
- Add chain of registers on every input
- Retime registers into circuit
 - Minimizing delay between registers

Penn ESE532 Fall 2018 -- DeHon

18

Preclass 1

- Retime



Penn ESE532 Fall 2018 -- DeHon

19

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

Penn ESE532 Fall 2018 -- DeHon

20

Justify Pipelining

(or composing pipelined operators)

Penn ESE532 Fall 2018 -- DeHon

21

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

Penn ESE532 Fall 2018 -- DeHon

22

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

Penn ESE532 Fall 2018 -- DeHon

23

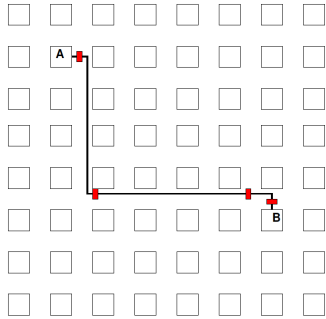
Interconnect Delay

- 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

Penn ESE532 Fall 2018 -- DeHon

24

Interconnect Example



Penn ESE532 Fall 2018 -- DeHon

25

Methodology: Pipelined Operator Graph

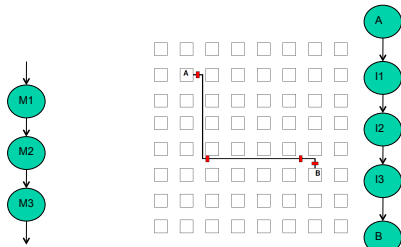
- Start with logical, unpipelined graph
- Treat each pipelined operator as a set of unit-delay operators of mandatory depth
- Treat each interconnect pipeline stage as a unit-delay buffer
- Add registers at input
- Retime into graph

Penn ESE532 Fall 2018 -- DeHon

26

Model

- 3-stage Multiplier
- Interconnect Delay



Penn ESE532 Fall 2018 -- DeHon

27

Pipeline Loop

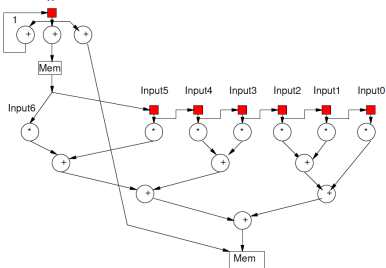
(and use for justify pipeline example)

Penn ESE532 Fall 2018 -- DeHon

28

Preclass 4

- Logical (unpipelined) dataflow graph for loop body



Penn ESE532 Fall 2018 -- DeHon

29

Example Operators

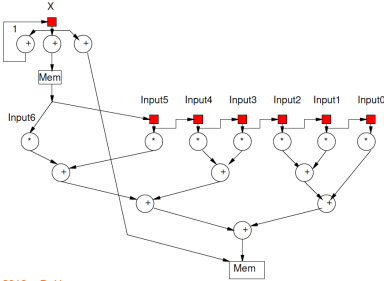
- Operator and Interconnect delays
 - Multiplier 3 cycles
 - Reading from Input array
 - Memory op is cycle after computing address
 - Takes one cycle delay bring data back to multiplier

Penn ESE532 Fall 2018 -- DeHon

30

Illustrate Need

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

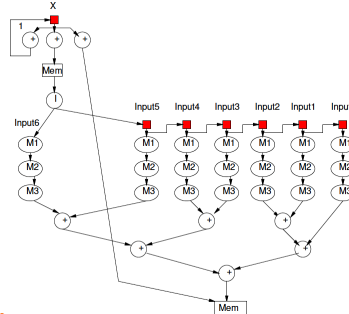


Penn ESE532 Fall 2018 -- DeHon

31

Model Graph

- Revised graph for modeling

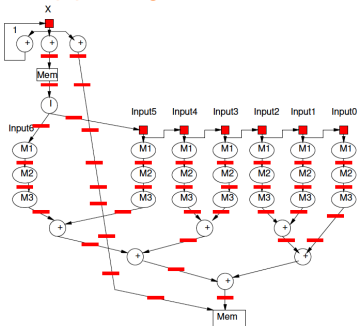


Penn ESE532 Fall 2018 -- DeHon

32

Pipeline Graph

- Result after pipelining?



Penn ESE532 Fall 2018 --

33

Pipelining Lesson

- Can always pipeline an **acyclic** graph 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 ESE532 Fall 2018 -- DeHon

34

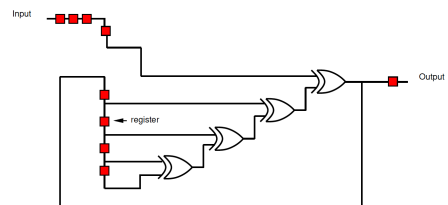
Graph Cycles

Penn ESE532 Fall 2018 -- DeHon

35

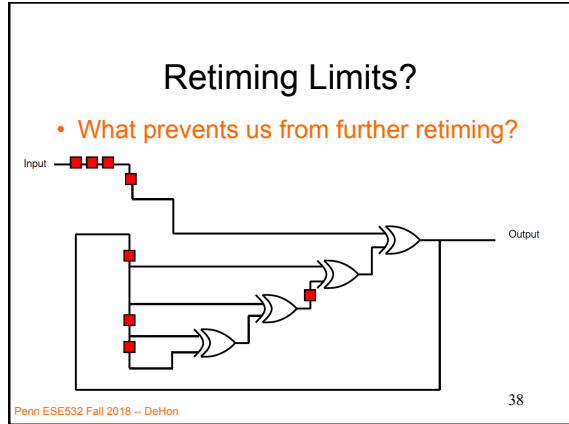
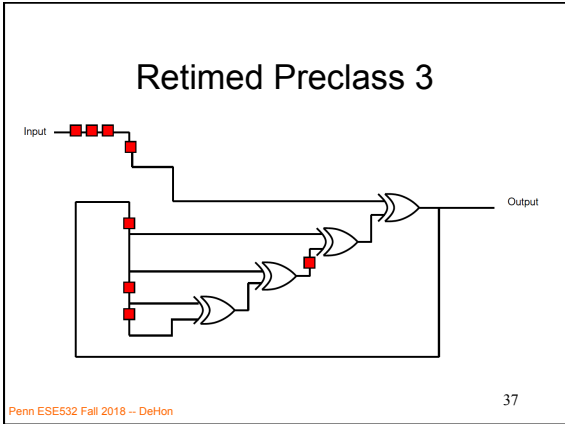
Preclass 3

- What cycle time can we achieve?
- How retime?

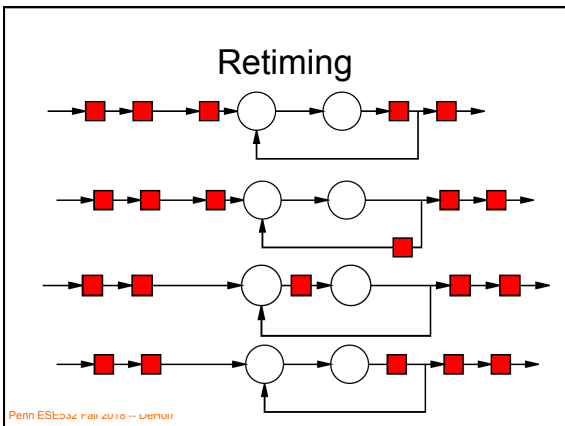
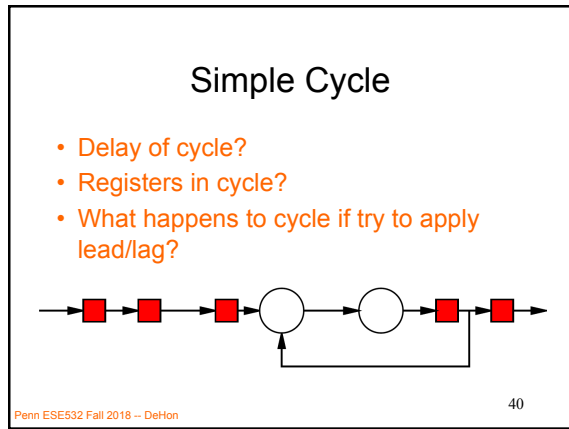


Penn ESE532 Fall 2018 -- DeHon

36



- ### Cycle Observation
- Retiming does not allow us to change the number of registers inside a cycle.
 - Limit to cycle time
 - Max delay in cycle / Registers in cycle
 - Pipelining doesn't help inside cycle
 - Cannot push registers into cycle
- Penn ESE532 Fall 2018 -- DeHon 39



- ### Loop
- What does graph look like for this loop body?
[multiply and mod each take 3 cycles]
 - For (i=0; i<N; i++)
C[i]=(C[i-1]*A[i])%N;
- 42
- Penn ESE532 Fall 2018 -- DeHon

Loop

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

Penn ESE532 Fall 2018 -- DeHon 43

Initiation Interval (II)

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

Penn ESE532 Fall 2018 -- DeHon 44

Loop

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

Penn ESE532 Fall 2018 -- DeHon 45

C-Slow

(if time permits)

Penn ESE532 Fall 2018 -- DeHon 46

Problem

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

Penn ESE532 Fall 2018 -- DeHon 47

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

Penn ESE532 Fall 2018 -- DeHon 48

2-Slow Simple Cycle

- Replace register with pair
- Retime

Penn ESE53

2-Slow Simple Cycle

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

Penn ESE532 Fall 2018 -- DeHon

50

Equivalence

- The 2-slow operator is equivalent to two data parallel operators running at half the speed

Penn ESE532 Fall 2018 -- DeHon

Automation

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

Penn ESE532 Fall 2018 -- DeHon

52

Lesson

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

Penn ESE532 Fall 2018 -- DeHon

53

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
- Cycles limit pipelining on single stream
- C-slow to share hardware among multiple, data-parallel streams

Penn ESE532 Fall 2018 -- DeHon

54

Admin

- Reading for Day 8 on web
- HW4 due Friday