ESE532: System-on-a-Chip Architecture

Day 7: September 25, 2017
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)

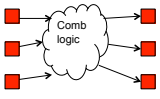
- Systematic Approach
- Justify Operator and Interconnect Pipelining
- Loop Bodies
- Cycles
- C-slow

Message

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

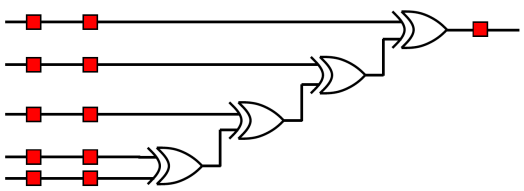
Synchronous Circuit Discipline

- 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



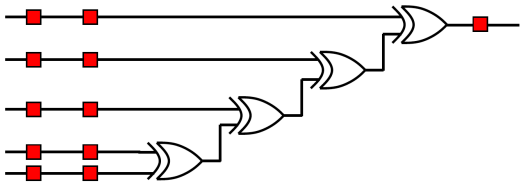
Preclass 1

- Delay between registers as shown?



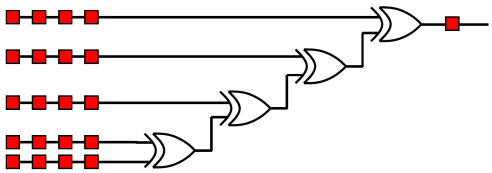
Preclass 1

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



Preclass 2

- Move registers so can clock at xor-delays?



Pipeline Reuse

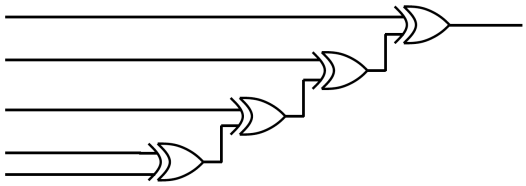
- Lower delay between clocks
  - Higher clock rate
  - Higher potential throughput
  - Faster we reuse our logic
  - More capacity get out of design
    - Assuming registers cheap in are and time overhead

Levelize-and-Cut Pipelining

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

Apply to xor-chain

- Levelize and Cut

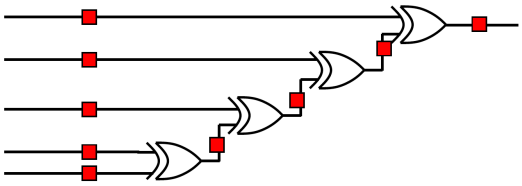


Note Registers on Links

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

What Happens?

- What would be wrong with this pipelining?

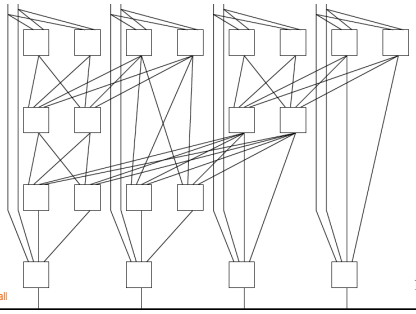


Consistent Pipelining

- 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

Levelize and Cut

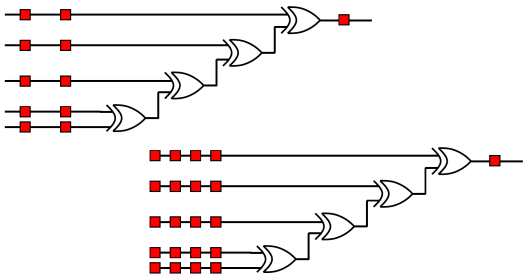
- Pipeline to operator level



Add Registers and Move

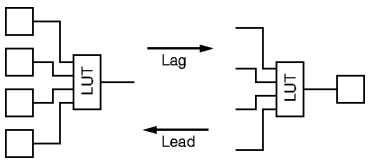
- If we're willing to add pipeline delay
  - Add any number of pipeline registers at input
  - Move registers into circuit to reduce cycle time
    - Reduce max delay between registers

Add Registers at Input



Legal Register Moves

- Retiming Lag/Lead

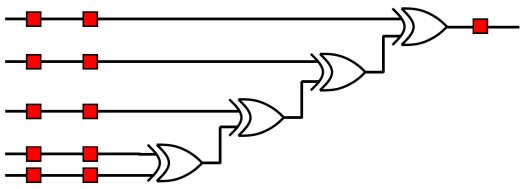


Add Register and Retime

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

Preclass 1

- Retime



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

Automation

- RTL Synthesis tools will take care of retime
- RTL Synthesis will **not** add registers
  - Changes behavior
  - Changes number of clocks
- Add registers and retime --- leave retiming to automated tools

Justify Pipelining

(or composing pipelined operators)

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 it's delay is too high
- Due to hierarchy
  - Pipelined this operator and now want to use it as a building block

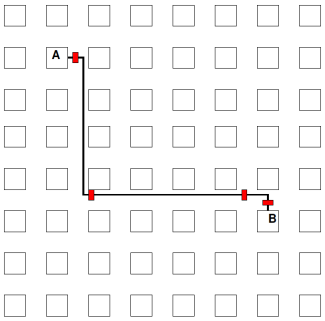
Examples

- 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

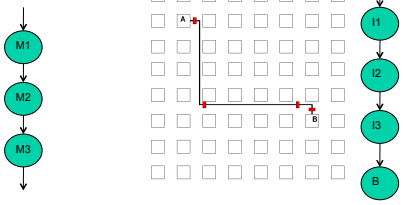


Pipelined Operator Graph

- 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 as input
- Retime into graph

Model

- 3-stage Multiplier
- Interconnect Delay

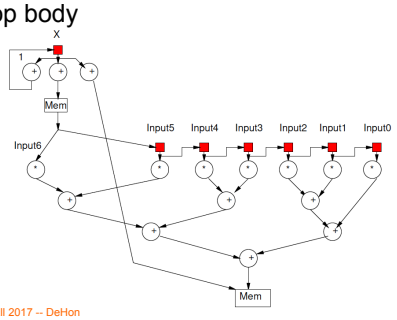


Pipeline Loop

(and use for justify pipeline example)

Preclass 4

- Logical (unpipelined) dataflow graph for loop body

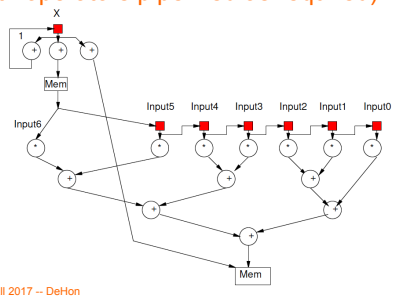


Example Operators

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

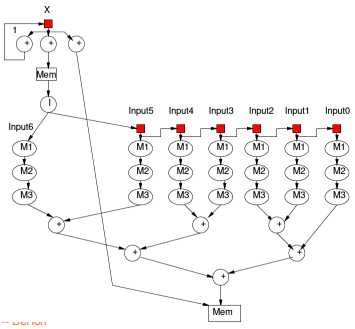
Example Need

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



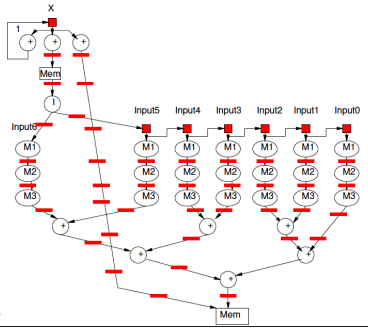
Model Graph

- Revised graph for modeling



Pipeline Graph

- Result after pipelining?



Pipelining Result

- 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

Preclass 5

- How preclass 5 relate to preclass 4?

```
for (int X = 0; X < OUTPUT_WIDTH; X++)
{
  unsigned int Sum = 0;
  for (int i = 0; i < FILTER_LENGTH; i++)
    Sum += Coefficients[i] * Input[Y * INPUT_WIDTH + X + i];
  Output[Y * OUTPUT_WIDTH + X] = Sum >> 8;
}
```

```
Sum = Coefficients_0 * Input0 +
      Coefficients_1 * Input1 +
      Coefficients_2 * Input2 +
      Coefficients_3 * Input3 +
      Coefficients_4 * Input4 +
      Coefficients_5 * Input5 +
      Coefficients_6 * Input6;
Output[Y * OUTPUT_WIDTH + X] = Sum >> 8;
```


Loop Unrolling

- Instantiate the loop body multiple times (with suitable change of loop variables)
- Full unrolling
  - Replace whole loop with straight-line code sequence that performs the same thing
  - Roughly with N copies of the loop body
- Partial
  - Some number of instances

Simple Unrolling

- For (i=0;i<4;i++)
  C[i]=A[i]*B[i];
- Unroll 2
- For (i=0;i<4;i+=2)
  C[i]=A[i]*B[i];
  C[i+1]=A[i+1]*B[i+1];

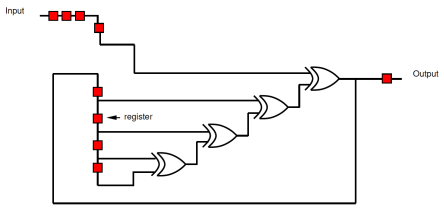
- Full Unroll

C[0]=A[0]*B[0]
C[1]=A[1]*B[1]
C[2]=A[2]*B[2]
C[3]=A[3]*B[3]

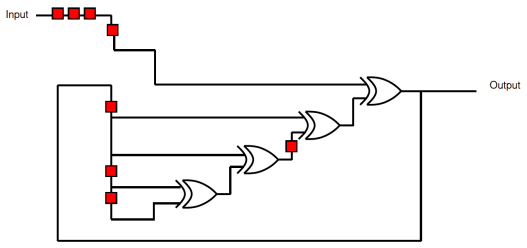
Graph Cycles

Preclass 3

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

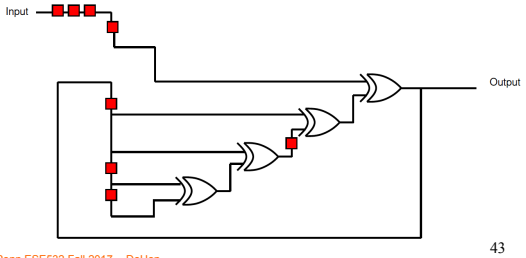


Retimed Preclass 3



Retiming Limits?

- What prevents us from further retiming?

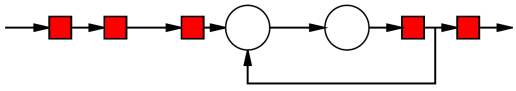


Cycle Observation

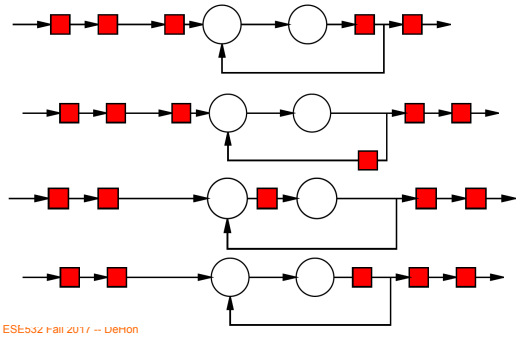
- Retiming does not allow us to change the number of registers inside a cycle.
- Limit to cycle time
  - Max delay in cycle / Registers in cycle
- Pipelining doesn't help inside cycle
  - Cannot push registers into cycle

Simple Cycle

- What happens to cycle if try to apply lead/lag?



Retiming



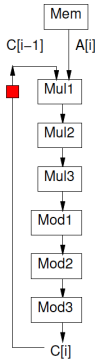
Loop

- 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;

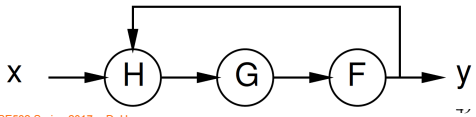
Loop

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



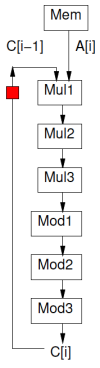
Initiation Interval (II)

- 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



Loop

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



Class Ended Here

C-Slow

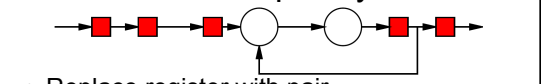
Problem

- 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?

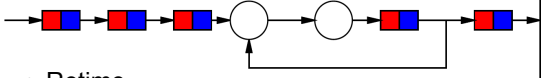
C-Slow

- **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

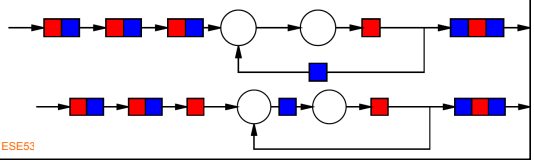
2-Slow Simple Cycle



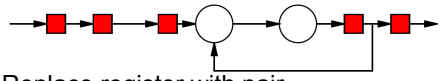
- Replace register with pair



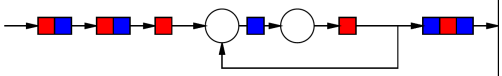
- Retime



2-Slow Simple Cycle



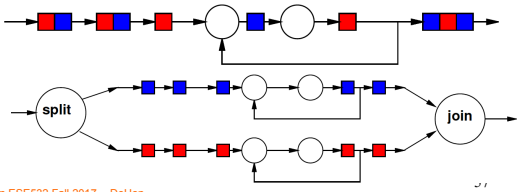
- Replace register with pair
- Retime



- Observe independence of red/blue computations

Equivalence

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



Automation

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

Lesson

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

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

Admin

- Reading for Day 8 on web
- HW4 due Friday