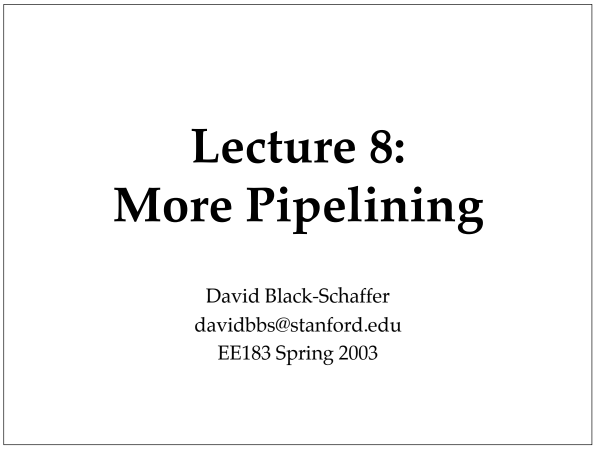
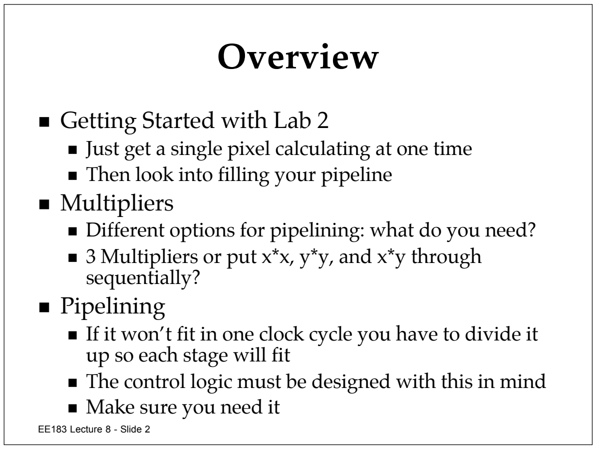


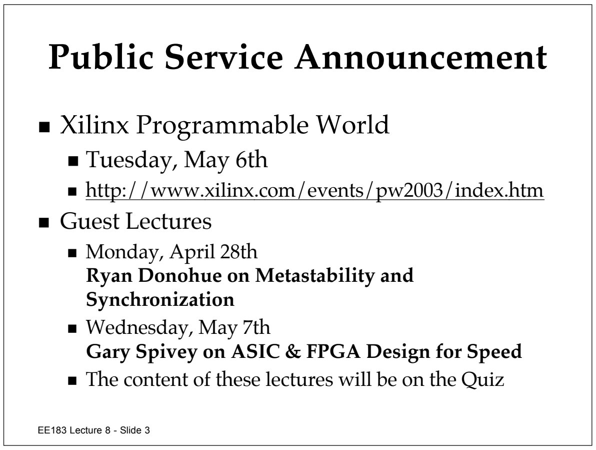
Lecture 8: More Pipelining

David Black-Schaffer
davidbbs@stanford.edu
EE183 Spring 2003



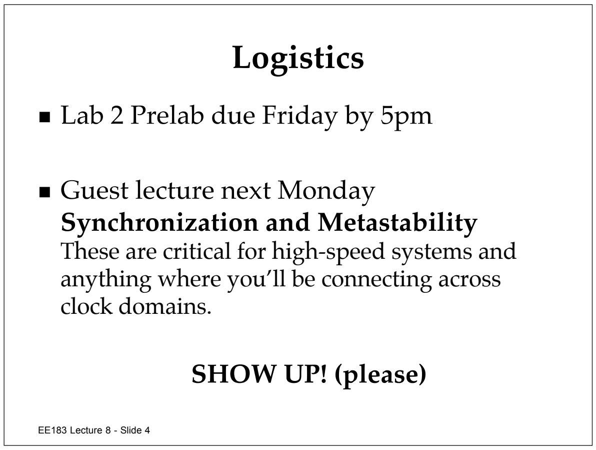
Overview

- Getting Started with Lab 2
  - Just get a single pixel calculating at one time
  - Then look into filling your pipeline
- Multipliers
  - Different options for pipelining: what do you need?
  - 3 Multipliers or put x*x, y*y, and x*y through sequentially?
- Pipelining
  - If it won't fit in one clock cycle you have to divide it up so each stage will fit
  - The control logic must be designed with this in mind
  - Make sure you need it



Public Service Announcement

- Xilinx Programmable World
  - Tuesday, May 6th
  - http://www.xilinx.com/events/pw2003/index.htm
- Guest Lectures
  - Monday, April 28th
    Ryan Donohue on Metastability and Synchronization
  - Wednesday, May 7th
    Gary Spivey on ASIC & FPGA Design for Speed
  - The content of these lectures will be on the Quiz



Logistics

- Lab 2 Prelab due Friday by 5pm

- Guest lecture next Monday
  Synchronization and Metastability
  These are critical for high-speed systems and anything where you'll be connecting across clock domains.

SHOW UP! (please)

Easier FSMs

```
always @(button or current_state)
begin
  write_en <= 0;
  output <= 1;
  next_state <= `START_STATE;

  case(current_state)
    `START_STATE:
     begin
       write_en <= 1;
       if (button)
         next_state <= `WAIT_STATE;
     end
     `WAIT_STATE:
     begin
       output <= 0;
     end
end
```

Do this if nothing else is specified.

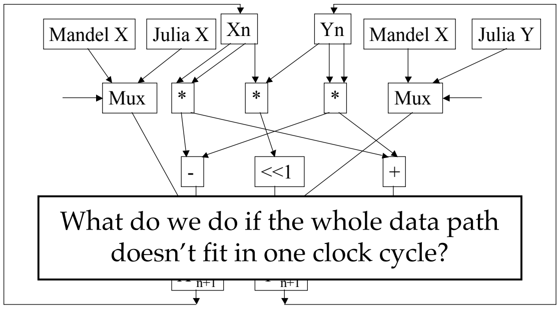
Note that the **`else`** is not specified.

What is the next state?

Be careful!

Easy way to infer latches!

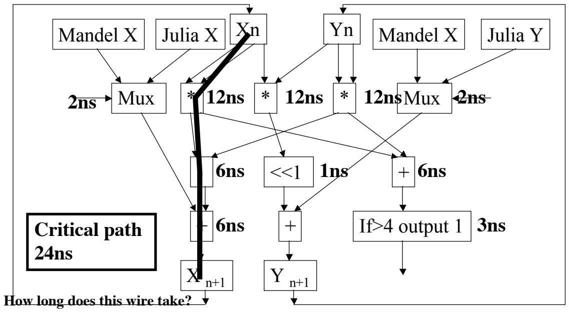
Data Path



What do we do if the whole data path doesn't fit in one clock cycle?

Feedback for next iteration

Pipelining Example 1

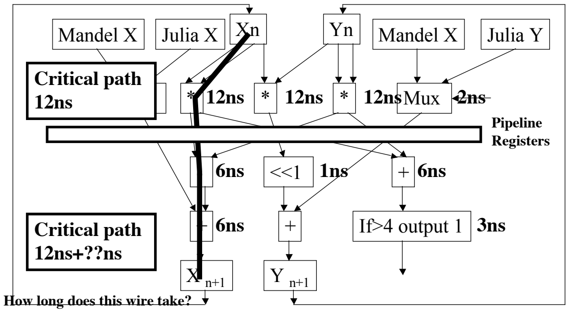


Critical path 24ns

How long does this wire take?

Feedback for next iteration

Pipelining Example 2



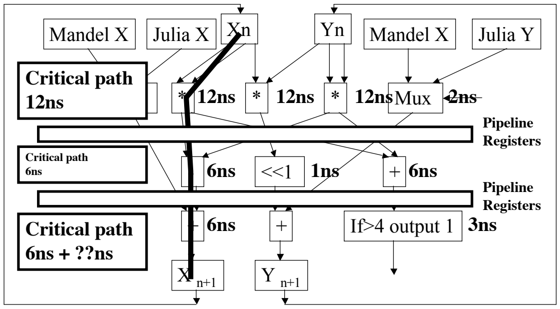
Critical path 12ns

Critical path 12ns+??ns

How long does this wire take?

Feedback for next iteration

Pipelining Example 3



Mandel X | Julia X | Xn | Yn | Mandel X | Julia Y

Critical path 12ns

* 12ns | * 12ns | * 12ns | Mux 2ns

Pipeline Registers

Critical path 6ns

6ns | <<1 1ns | + 6ns

Pipeline Registers

Critical path 6ns + ??ns

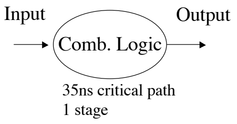
- 6ns | + | If>4 output 1 3ns

X_{n+1} | Y_{n+1}

Feedback for next iteration

Latency and Throughput

Not Pipelined



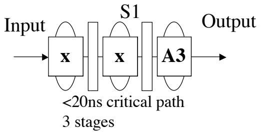
Input → Comb. Logic → Output

35ns critical path
1 stage

Time	Input	Output
t=0	A1	
t=35ns	A2	A1
t=70ns	A3	A2
t=**105ns**		A3

First output takes longer because it has to go through all the stages but subsequent results ***can*** come out every clock cycle.

Pipelined



Input → x | x | A3 → Output

S1

<20ns critical path
3 stages

Time	Input	S1	Output
t=0	A1		
t=20ns	A2	A1	
t=40ns	A3	A2	A1
t=60ns		A3	A2
t=**80ns**			A3

Key points on Pipelining

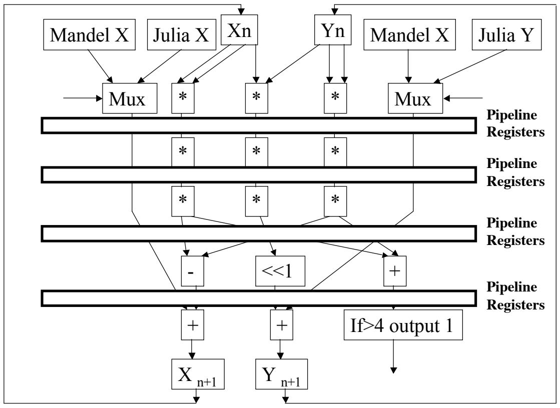
- Increases utilization for operators
  - You can do multiple calculations at once so you can use everything maximally (ideally)
  - This is the point! Store the results from smaller calculations to make the overall calculation faster.
- Insert the next data item into the datapath before the previous one has finished
- The pipe registers keep the computation separate
  - You will have a lot of pipe registers if you're doing a lot of calculations (I.e., Lab 3!)
- What is the effect of the algorithm feeding back on itself?
  - Do all points have the same number of iterations? *control*
  - Is the data dependent between pipeline stages? *hazards*

Multipliers

- CoreGen gives you several pipelining options
- Which is best?
  - Depends on your design
  - Data size will determine speed and pipelining
- Design is an iterative process so you won't be able to choose the best approach at first (i.e., get started early!)

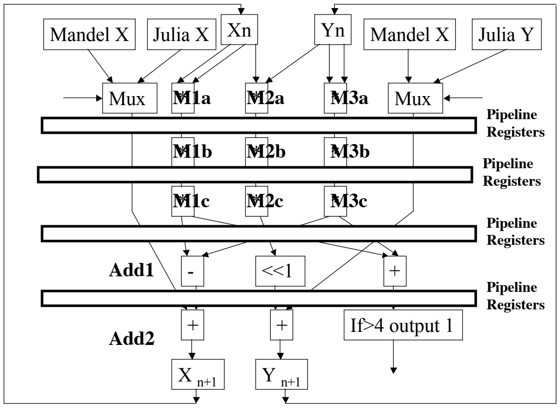
Multiplier Issues

- Multipliers are BIG
  - How can we get away with fewer multipliers?
- Multipliers may be SLOW
  - How can we utilize them maximally?



Now we have…

- With a 3-stage multiplier you've now got 5 pipeline stages
- How can you keep the pipeline full?
- How many things do you need to calculate at once?
- What is full? Will you ever get 100% utilization? What is good enough?



Pipeline Performance Analysis

- With the bad data path (3, 3 stage multipliers and 2 stages after that; one pixel at a time)

Clk	M1a	M2a	M3a	M1b	M2b	M3b	M1c	M2c	M3c	Add1	Add2
1	●	●	●								
2				●	●	●					
3							●	●	●		
4										●	
5											●

- In 5 cycles we used 11 units out of 55 available: 20% average utilization

Pipeline Performance Analysis

- With the bad data path (3, 3 stage multipliers and 2 stages after that; **multiple** pixels at a time)

Clk	M1a	M2a	M3a	M1b	M2b	M3b	M1c	M2c	M3c	Add1	Add2
1	●	●	●								
2	▲	▲	▲	●	●	●					
3	●	●	●	▲	▲	▲	●	●	●		
4	❖	❖	❖	●	●	●	▲	▲	▲	●	
5	●	●	●	❖	❖	❖	●	●	●	▲	●

- We approach 100% utilization if there are no stalls or dependencies and we can keep getting new data

What performance is required?

- Replication and Pipelining are not trivial to implement—make sure you need them
- Is either needed for Lab #2?
- How would you tell?
  - Hint: each Julia image takes at most (64*64*64*7*1/50e6) = 0.036s to create.
  - Is this "real-time" enough for an animation?
  - Other issues? Need to meet timing for the VGA.

What do we expect

- The previous data path is terribly inefficient if you only put one pixel through at a time, but doing multiple pixels at once is very complicated

- As an alternative you can use one multiplier and put your x*x, y*y, and x*y through it in a pipelined manner.
- What's the efficiency? Is it a good tradeoff for area/speed? **This analysis is critical!**

Pipeline Performance Analysis

- Single multiplier, put $\mathbf{x}^2$ (●) through first, then $\mathbf{y}^2$ (▲), then $\mathbf{x}\bullet\mathbf{y}$(●).

Clk	Ma	Mb	Mc	Add1	Add2
1	●				
2	▲	●			
3	●	▲	●		
4		●	▲		
5			●		
6				●	
7					●

- In 7 cycles we use 11/35 functional units = 31%
- But we only have 1 multiplier
- How much space do we save? What is most important?

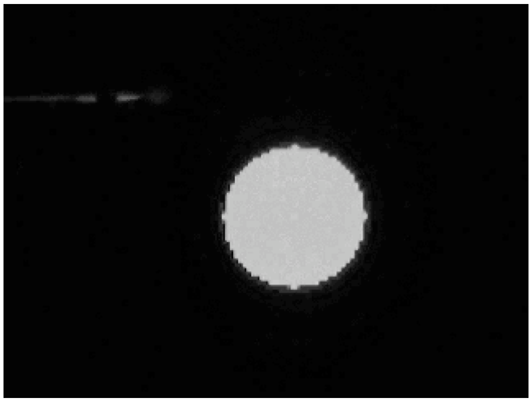
Pipeline Performance Conclusions

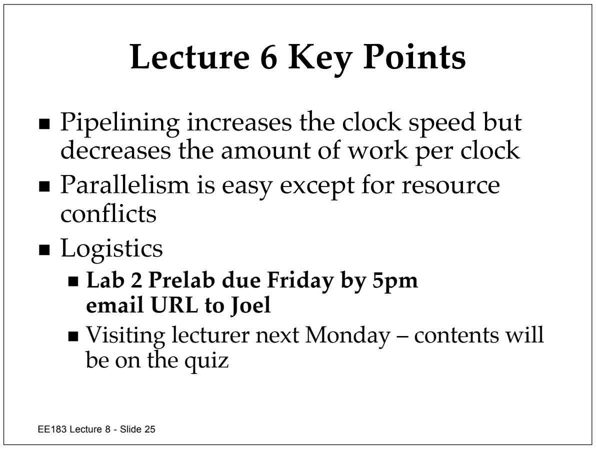
- You need to know your algorithm and what tradeoffs you are making

- What do you care about?
  - Speed?
  - Area?
  - Both.
    (Power is a function of speed and number of transistors, i.e., area.)

Zooming

- An arbitrary portion of the screen can be described in many ways. Here are two:
  - Xmin, Xmax, Ymin, Ymax
    - Requires dividing by the number of pixels
  - Xorigin, Yorigin, scale
    - Requires a fixed number of pixels
- Hints for Zooming:
  - Have registers with the X, Y origins and increment/decrement them with the up/down, left/right buttons
  - Have a scale register which goes up/down with the zooming in/out
  - When converting form 0..63 to -2.00 to 2.00 use the scale and origin to calculate the new value





Lecture 6 Key Points

- Pipelining increases the clock speed but decreases the amount of work per clock
- Parallelism is easy except for resource conflicts
- Logistics
  - **Lab 2 Prelab due Friday by 5pm**
    email URL to Joel
  - Visiting lecturer next Monday – contents will be on the quiz