Lecture 6: Documentation & Lab 2

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

Overview

- Documentation
  - See the web page for details
  - Short and sweet. Make it clear how your design works and why you chose that approach.
- Lab 2: Fractals
  - Understand the matlab/perl code algorithm
  - Think about the **data path** and the **control path**
  - What can you re-use from Lab 1?
  - Determine your number format from the matlab code
  - Map the -2 to 2 plane to a 0 to 63 screen by extracting bits and choosing a binary point

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

- Any questions?
  - Lab 1 Demo due Friday at 5pm.
    We'll be in the lab from 3-5 on Friday.
    You can demo any time we're around.
  - Writeup due Monday at midnight
    Email URL of PDF to Joel
- Late labs will really hurt your grade.
  - It is very important to stay on track with this course. There will be **no** free late days or extensions given. The late penalty, both for demos and writeups, is 2 points per calendar day late (10%). Labs must be completed, even for 0 points, to finish the course.

Documentation

- What do we expect?
  Fundamentally:
  - **What** did you do? (project design)
  - **How** did you do it? (system architecture)
  - **How** did it work out? (post-lab analysis)
- Keep it short and to the point. Make it clear what you've learned in terms of system design and implementation.
- It is okay to admit that in retrospect you should have done the design differently.

Documentation

- What things would a competent engineer in the field need to understand to modify or use this design?
  - More often than not, that engineer is ***<u>YOU</u>*** in 12 months.
  - Don't loose the forest in the trees.
    - They can always "Use the Source" for the trees.
- Incrementally work on the documentation — don't leave it until after the design is complete!!

From "The Tao of EE183"

- Format = PDF
- Contents
  - Title Page
  - Introduction
  - Design (similar to the pre-lab)
  - Results
  - Conclusions
  - Appendices
    - Simulations
    - Implementation (verilog files/test bench files)
    - Performance Metrics (layout images)
- **Not** a "formal" lab report.

Introduction

- "I implemented Conway's Game of Life in hardware with input from a Sega gamepad and output to a VGA display."
- Any special features? Mention them here.
- We know what the Game of Life is
- You will tell us more details later

Design

- Similar to your pre-lab report
- **How** does your implementation work?
- **Why** is/was this a good design?

- State machines
- Block diagrams
- Description of how they work together.
  - Make it clear how your design works and motivate it by why this is (or why you thought it was) a good design.

EE183 Lecture 6 - Slide 9

Results

- How did it work out?
- Max speed/area usage.

- For lab 2: What was your critical path? How could you fix it?

EE183 Lecture 6 - Slide 10

Conclusions

- What do you think of your design? *This is important!*
- What worked? What didn't?
- What would you do differently next time? I.e., what did you learn?

If you don't have anything to say here then you shouldn't be taking this class.

EE183 Lecture 6 - Slide 11

Appendices

- Simulations
  - You must annotate **on the waveforms** what is going on and why it is important
  - Simulate all important/complicated FSMs
- Implementation
  - All your .v files. These should already be commented to the point where we can "just" read them.
  - Include your .UCF and simulation files
- Performance Metrics
  - Usage and Routing images
  - Speed and area statistics
  - Lab 2: Highlight the critical path.

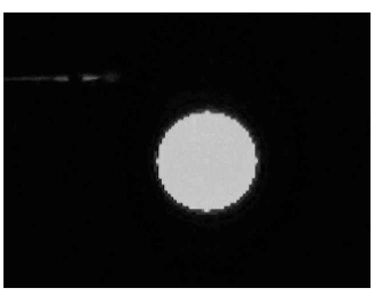
EE183 Lecture 6 - Slide 12

Questions on Documentation?

Timing

- What limits our speed?
- Last time…
  - RTL design - Register Transfer Logic
    - Speed limited by the time it takes to get from one register (flip flop) to another
- Today…(Lab 2)
  - How can we change the architecture to do more at once while keeping the clock rate up?
  - Parallelism and Pipelining

Lab 2: Fractals!



Lab 2 Requirements

- Pipelined calculation of a 64x64x4-bit fractal from -2 to 2 in the real and imaginary planes
- Switch display between Mandelbrot and Julia set
- Julia set constants chosen by the position of a blinking cursor as in lab 1
- You must have at least one of:
  - Animation around an "interesting" path for the Julia set
  - Zoom in/out capability (much cooler)
- Encouraged:
  - Color animation
  - Parallel computation

Key Concepts for Lab 2

- Data path and control path separation
- Parallelism
- Pipelining
- Fixed-point math

Mandelbrot Fractal

- The Mandelbrot set is the set of points in the complex c-plane that do not go to infinity when iterating $z_{n+1} = z_n^2 + c$ starting with $z = 0$. One can avoid the use of complex numbers by using $z = x + iy$ and $c = a + ib$, and computing the orbits in the ab-plane for the 2-D mapping

$$x_{n+1} = x_n^2 - y_n^2 + a$$
$$y_{n+1} = 2x_n y_n + b$$

with initial conditions $x = y = 0$ (or equivalently $x = a$ and $y = b$). It can be proved that the orbits are unbounded if $|z| > 2$ (i.e., $x^2 + y^2 > 4$).
  - http://www.olympus.net/personal/dewey/mandelbrot.html
  - http://www.jade-leaves.com/mandelbrot_set/index.shtml

Julia Set

- Very similar except for the next state generation except the (a,b)
  - these are constants throughout the calculation
- Sample code for matlab and perl is in
  - http://www.stanford.edu/class/ee183/fractals/
  - Note: the perl ouput looks funny since ascii character dimensions are not proportional

Look Complicated?

Really just iterate over the -2 to 2 real (x) and imaginary (y) planes (i.e., the screen) calculating:

$$x_{n+1} = x_n^2 - y_n^2 + a$$
$$y_{n+1} = 2x_n y_n + b$$

Until $x^2 + y^2 > 4$ or the number of iterations is > 64. Then **the number of iterations it took is what you display** at that location on a 64x64x4-bit display.

Complicated bits

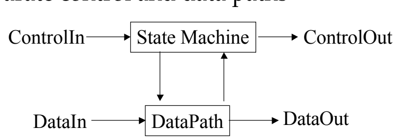
- How do we do the multiplication?
- How do we get the numbers -2 to 2 to map to a screen 64 pixels wide? Fractions!?
- How do we zoom in?
- How do we make it run fast?

Easy bits

- How do we interface to the game pad?
- How do we get a flashing cursor?
- How do we switch modes between Mandelbrot and Julia sets?
- How do we output an image to the VGA?
- How do we store an image to a BRAM?

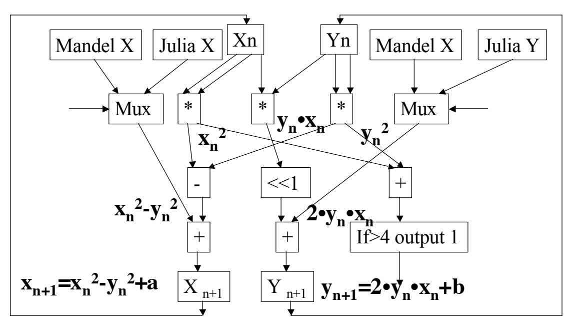
Implementation

- Separate control and data paths



- Data path is regularly structured so you can try to optimize it for speed
- If it doesn't fit in 1 clock cycle then pipeline it, but make sure the control logic knows this!

Data Path



Feedback for next iteration

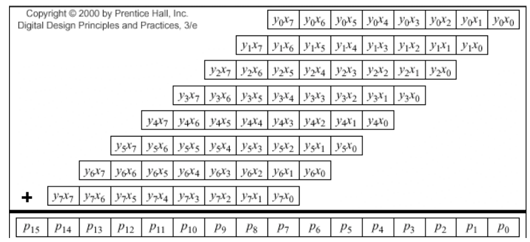
We need multipliers

- We know how to build everything else in the data path (+, -, <<, >, FFs)
- CoreGen to the rescue, but first, what is a multiplier?

EE183 Lecture 6 - Slide 25

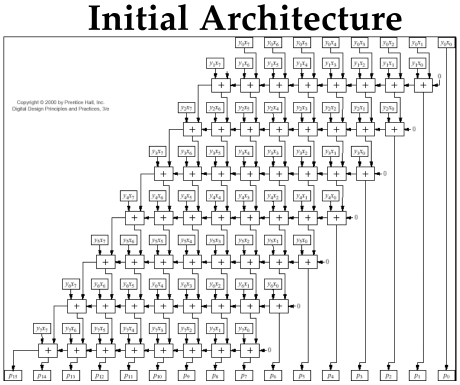
Multipliers

- Fundamentally Repeated Addition



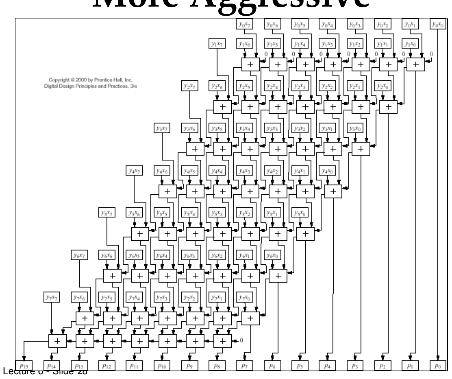
EE183 Lecture 6 - Slide 26

Initial Architecture



EE183 Lecture 6 - Slide 27

More Aggressive



EE183 Lecture 6 - Slide 28

Multipliers Summary

- Lots of clever architectures out there. They all do the same thing—multiply!
- Consider routing delay in addition to logic delay.
- These are going to be big and probably slow. (how slow?)

Fractions?

- Those were all Integer Multipliers
  - Signed operands in twos-complement work fine
- Our algorithm calls for fractional arithmetic
  - Normally implemented as Floating Point Math
    - Very painful (lab 2 used to be doing a floating point adder)
  - So use Fixed Point Math
    - Assume numbers are always in the form: X.Y where X and Y have constant width
    - I.e., in 4.3 twos-complement format we have:
      0010.000 = 2.0 0010.100 = 2.5
      0010.110 = 2.75 1010.000 = -6.0

Fixed Point Math

- Addition/Subtraction as normal
- Multiplication works as normal if you select the right thing in CoreGen
- However, if you multiply two 4.3 numbers what do you get?
- You get an 8.6 number. What part do you want?

Fixed Point Partition?

- How big should the integer part or fractional part be?
  - Want to minimize these since multipliers grow quickly (roughly power of 2) in size and latency with operand size
  - Don't want them so small that overflow of the integer part occurs (results in aliasing) or that the fractional part has large quantization error
- We stop the loop when magnitude is greater than 4
  - Use that knowledge to approximate size of intermediate operands
  - Run a matlab simulation and figure out the largest value
  - What about zooming? Need more precision? How much?

Tricky Bit

- We have a 64x64 pixel screen. We want to map this to -2 to 2. How do we do that?
- Hint:
  - Counting from 0 to 64 goes **000000** to **111111** in 6.0 notation or **0000000** to **0111111** in 7.0 notation
  - Counting from 0.00 to 4.00 goes **000.0000** to **011.1111** in 3.4 notation
  - Notice that counting from 0.00 to 4.00 looks an awful lot like what you are getting out of the VGA 0 to 64 coordinates, but it goes from 0 to 4 instead of -2 to 2
  - What can you do to **easily** fix that?

Monday: Pipelining

- What do we do if the whole data path doesn't fit in 20ns?
- Split it up into smaller chunks: pipelining
- But, we have to put multiple chunks through at the same time to increase the throughput.

Lecture 6 Key Points

- Documentation should focus on **what** you did and **why** you did it with an analysis of **how well** it worked. See web page.
- Think about the data path and the control path
- Fixed point notation is easy if you realize that the **counting is the same**, **but the interpretation is different**.
- Logistics
  - **Lab 1 DEMO due Friday at 5pm (56 hours left).**
  - Office Hours/Wed/Fri 10-11am, Thur 7-9pm, and demos Fri 3-5pm
  - Writeup due Monday by midnight.