The 6.004

Project Beta Handbook

or:

How I Learned to Stop Worrying and Love the β

by Eben Kunz

Table of Contents

v0.0.7

This is currently an active document. There may be errors.
Please send corrections or suggestions to cjt@mit.edu and ekunz@mit.edu with the subject "6.004 PBH"

Getting Started

First Steps:

- Make a backup copy of your Lab 6 Beta and ALU. It is worth points since it works.
- Read all of the Design Project handout and hook up your Lab 6 Beta.
- Try the checkoff. To calculate your Benmark, submit the checkoff. You can submit as many times as you'd like, just make sure your best is last.
- The starting clock period is 20ns, and if your Lab 6 beta worked, the project checkoff probably will too. If it fails there is hope, and the fix is probably quick, so read on!

Good Advice:

"I found a cure for the plague of the 20th century, and now I've lost it!"
- Medicine Man (movie, 1992)

- If you are at a point where your beta works and has a decent Benmark, make a copy.
- Data you care about is stored in more than one place. Even if we know you've been working hard, we can't give you credit for what you no longer have.
- When your Beta gets faster, reduce both clock period **and** overall simulation time. Otherwise your Benmark will remain the same.
- Don't make too many subcircuits. Your personal version of “mux32” makes it harder for course staff to read your code, can introduce errors, and doesn't save you any work.
- **Quizzes are as important as the final project. Study first.**

What Now?

- If your Lab 6 Beta didn't pass the checkoff, it is probably because your adder is too slow. Try a longer clock period or fix your adder.
- Read about Timing Analysis. This talks about how find out how fast your beta will run, and what is slowing it down.
- Read about Buffers. It is simple to add buffers, so get in the habit and add them as you go. Buffers are somewhat outside the scope of 6.004, so this section is very explicit about what you should do.
- Read other sections (or don't) and do what you understand best. You will find that you still need to figure things out on your own. Do what strikes your fancy, and have fun!

Timing Analysis

Making the Beta smaller is a fairly straightforward process. If your Beta still works, and the stuff you put in is smaller, life is good. Making your Beta faster can take more effort, and sometimes changes will not do what you expect. In order to figure out what is going on, some timing analysis may be required. This section is about the timing of individual components, not about improving the structure of critical paths.

Fixed delays:

Your Beta must have a 1k word data/instruction memory. Your Beta must also have a register memory.

- Reading from data/instruction memory: 4ns
- Reading from register memory: 2ns

Negligible Delays:

When starting off, there are a few things not to worry about until your Beta starts getting really fast.

- Unloaded or properly buffered gates: 0.03-0.25ns
- Setup time to write to a memory (data or register): 0.04ns

Potential Improvements:

In sections that follow, you can learn what to do about the following:

- Unbuffered gates with large loads: up to 2ns
- ALU Adder
  - Ripple adder with a slow carry chain: ~12ns

  If your Beta won't run with a 20ns clock period, this is probably why.
  - Ripple adder with a fast carry chain: ~5-7ns
  - Moderately fancy adder: ~2-4ns
- Generating control signals using a ROM: 2ns

Equivalent to a register file with no write port.

- Pipeline Structure: ???ns

The unpipelined Beta does everything in a single clock period. Is that the best way?

Examining Your Beta's Timing

- When you run a simulation, jsim tells you what the "min observed setup" was. This is the difference between your clock period and your critical path time, but **only if** your critical path time is less than your clock period. Otherwise it is basically a useless number.
- Open your Beta in jsim and make sure you are looking at the main project file. Click on the Gate-level Timing Analysis icon. You don't need to run a simulation first.

- Look at the file that pops up, which ends ".timing". There are up to four sections, two of which you shouldn't see for a Beta.
  - "Worst-case tPDs for top-level combinational paths"

This section is for test signals that don't end up in a register.
You shouldn't see this. (You will if testing your adder with lab3adder.jsim)

  - "Worst-case tPDs from clk to top-level outputs"

This section is for signals that start at a register and don't go anywhere.
You shouldn't see this.

  - "Hold time violations for clk"

Register hold time violations. You can usually ignore this.

  - "**Minimum cycle time for clk is xx.xxxns**"

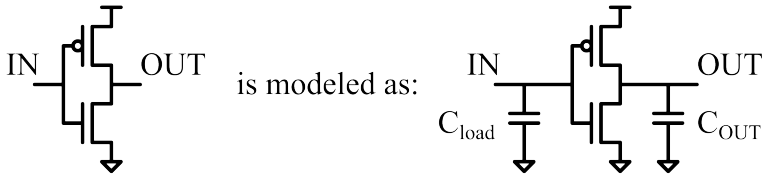
This is how fast you can successfully run your Beta, assuming it works otherwise.
This section is for paths between two registers.
The first path listed is the slowest path between two registers, and is therefore your critical path.

Examining Your Critical Path

- Look first for instruction memory access to get yourself oriented.
  - + 4.0xxns – the time the memory takes to read a word.
  - = 4.xxxns – the total time along the critical path at the end of the memory read.
  - idxx%0 – the memory output signal that is on the critical path.
  - [xmem] – the name of your memory.
- Check your adder delay. Some adder is inevitably in your critical path. Compare the total critical path time somewhere near the input of an adder with somewhere near the output of the same adder. How much of your clock period is this? Which adder is it? How many bits?
- What else is adding lots of time to your clock period? What do you want to improve first?
- Look at the individual gates. The times are higher than t_{PD} from the Lab 3 cell list. Read the Buffers section for an explanation.
- Check other paths listed, is there anything close to the critical path that also needs improvement?

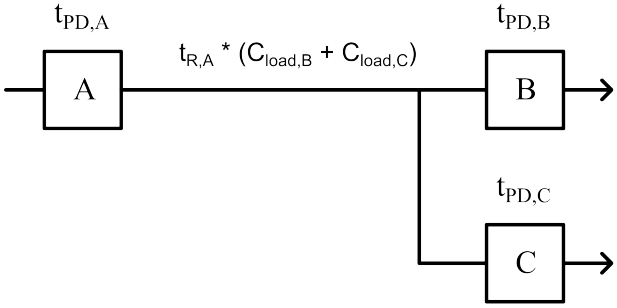
Buffers

Delays in digital circuits are, to a first order approximation, caused by capacitance within CMOS transistors and resistance of connections between them. Jsim models both, but accounting for capacitance is generally sufficient for the project Beta. Each gate has capacitance which must be charged or discharged in order for its output to transition. An inverter, for example, can be modeled as follows:



Each gate must always switch its own output capacitance, and propagation delay, t_{PD}, is the time it takes for a gate to drive its own output capacitance, C_{OUT}. Input capacitance must be driven by the output of another gate, and is referred to as the load (on other gates) of the gate. The time it takes a gate to charge or discharge a load connected to its output is calculated with t_R for a rising output and t_F for a falling output. Both are specified in ns/pf.

In the current set of jsim gates t_R is always higher than t_F. This is not always the case in jsim or reality, but for the project Beta each gate uses NMOS and PMOS transistors that are relatively close in size to each other. For a given gate area, PMOS has a higher resistance, so the current through the PMOS which charges load capacitance is smaller than the current through the NMOS which discharges load capacitance. NOR gates have a particularly high t_R since their pullup network consists of a chain of PMOS in series. We are concerned with the slowest part of our circuit, so we can ignore t_F and use t_R exclusively, since that is equivalent to using $\max(t_R, t_F)$. In the timing report, the total delay of a gate is $t_{PD}+t_R*C_{load}$, which is the total time it takes a gate to switch all capacitance connected to its output.



Buffers and inverters (referred to collectively as buffers) are designed to have a lower t_R than other gates, so adding a buffer reduces the delay from driving a particular load. New delay is introduced by the t_{PD} of the buffer and the time the original gate takes to drive the load introduced by the buffer. Choosing the "right" buffer means selecting the buffer which minimizes the total delay of the circuit. The reduction in total delay can sometimes be significant, and the size of the buffer itself is generally well worthwhile.

Timing Example:

Timing of a nand2 driving 32 mux2 inputs:

$t_{PD,NAND2} = 0.03\,\text{ns}$

$t_{R,NAND2} = 4.5\,\text{ns/pf}$

$C_{load} = 32 * 0.005\,\text{pf} = 0.160\,\text{pf}$

$t_{NAND2} = t_{PD,NAND2} + t_{R,NAND2} * C_{load} = 0.03\,\text{ns} + 4.5\,\text{ns/pf} * 0.160\,\text{pf}$

$t_{NAND2} = 0.750\,\text{ns}$

Timing of a nand2 driving a buffer_4, which then drives 32 mux2 inputs:

$t_{PD,BUF4} = 0.07\,\text{ns}$

$t_{R,BUF4} = 0.56\,\text{ns/pf}$

$C_{load,BUF4} = 0.010\,\text{pf}$

$t_{NAND2} = t_{PD,NAND2} + t_{R,NAND2} * C_{load,BUF4} = 0.03\,\text{ns} + 4.5\,\text{ns/pf} * 0.010\text{pf}$

$t_{NAND2} = 0.075\,\text{ns}$

$t_{BUF4} = t_{PD,BUF4} + t_{R,BUF4} * C_{load} = 0.07\,\text{ns} + 0.56\,\text{ns/pf} * 0.160\text{pf}$

$t_{BUF4} = 0.079\,\text{ns}$

$t_{TOTAL} = 0.154\,\text{ns}$ That could reduce your clock period by 5% in the right place!

Rules for Buffering:

- Don't buffer the clock. The clock is an ideal source, and can drive any load with no delay.
- Buffers may not be necessary outside of the critical path. That said, the easiest way to prevent unnecessary delays is to buffer everything that drives more than about 7 other gates.
- The load of a buffer can be rather high, so if you go crazy and add too many buffers to the output of a gate, it may actually increase overall delay.
- Don't worry too much about buffering versus inverting. The difference is minimal compared to the benefit of buffering in general. Do what is convenient and makes sense to you.
- On the timing report, no gate delay should get much above 0.25ns. It probably means you've forgotten to put a buffer somewhere.
- Avoid buffer trees. The extra propagation delays usually make them a poor choice.

Good Starting Choices for Buffers:

t_R	C_{load}	Buffer / Inverter
4.5 ns/pf (most gates)	40-87 pf (8-17 gates)	buffer_4 / inverter_2
	88-160+ pf (18-32+ gates)	buffer_4 / inverter_8
6.7-9.5 ns/pf (nor)	20-82 pf (4-16 gates)	buffer_2 / inverter_2
	83-160+ pf (17-32+ gates)	buffer_4 / inverter_8

Syntax Example:

```
Xmux select#32 a[31:0] b[31:0] foo[31:0] mux2
    becomes
Xselect_buf select select_buf buffer_4
Xmux select_buf#32 a[31:0] b[31:0] foo[31:0] mux2
```

Trimming the Fat

Put your Beta on a diet!

Checkoff Limitations

The checkoff file does not ask your Beta to perform every possible operation, which makes it easy for you to shrink your Beta by removing the unused parts. It is better to consider these early on since it may make later changes easier.

- MUL and MULC are never used. Good job on that multiplier but it is huge...
- Although it's tempting to take advantage of other limitations in the benchmarks (e.g., the memory only needs to be 1024 locations, so the high bits of the PC aren't "needed"), please don't! *The spirit of the design project is to create a Beta that meets all the specifications*, but is fast and small – we're not trying to produce hardware optimized for just the benchmark code.

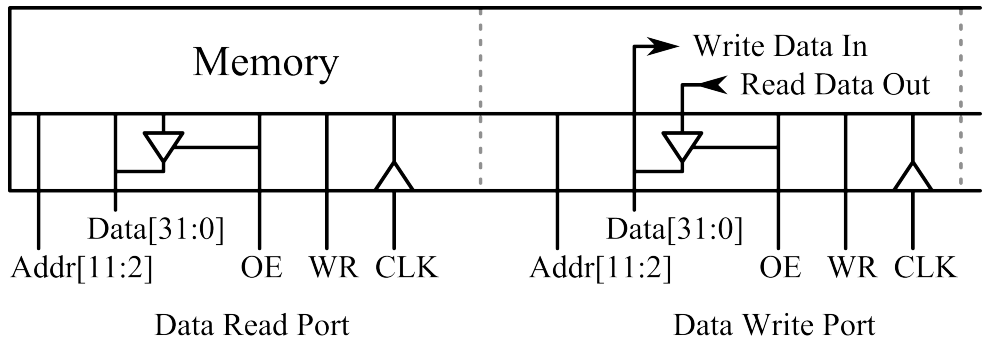
Other Size Improvements

Here are a few hints about other improvements that may reduce the size of your Beta. These are safe too.

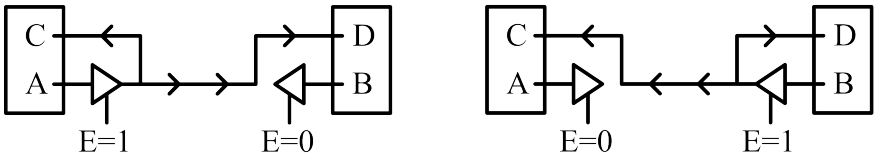
- Only use one ALU. If you hard wire ALUFN[5:0], you aren't using the capabilities of a complete ALU anyhow.
- Can there be a simultaneous read and write to main memory? Eliminating a memory port is a **significant** size improvement.
- How big is the Opcode ROM? Could you make it smaller using some other method to generate the control signals? Is it worth the effort? In a specific case?
- Do you ever need the branch-offset adder at the same time as the ALU?

Memory Port Elimination

The main memory of the Beta is accessed via 3 ports by default. One port reads an instruction based on the PC, and two are used for data memory: one to read and one to write. Each memory port is inherently capable of being used as both a read port and a write port.



The circuitry to make each port is significant in size, and since the Beta instruction set never reads and writes data at the same time, it is beneficial to reuse a single port for both. In order to use a port bidirectionally, the same wires need to send data in two directions. This is done with tristates. A tristate only sets the value of its output (Z) when its control input (E) is 1.



The OE input of a memory port has the effect of an internal tristate, so when OE is high that port drives its data lines. Your mission, should you choose to accept it, is to use this tristate scheme in your own Beta so that your main memory only needs two ports instead of three. Your changes should be minimal and local, so be sure to leave your original memory in a comment just in case. Remember it may be easier to connect two buses rather than to rename one of them.

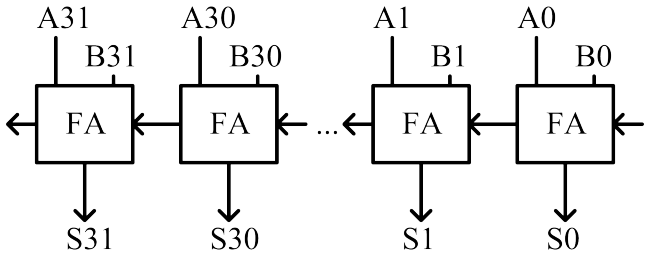
Adders

In order to discuss adders efficiently, delay is measured in whole numbers. One bit adders, mux2s, and buffers will each have a delay of 1. It is up to you to decide how close these proportions are to actual jsim results and adjust accordingly.

lab3adder.jsim is the best tool for getting a new adder to work. Once it works, run the project checkoff to look at your timing.

Ripple Adders

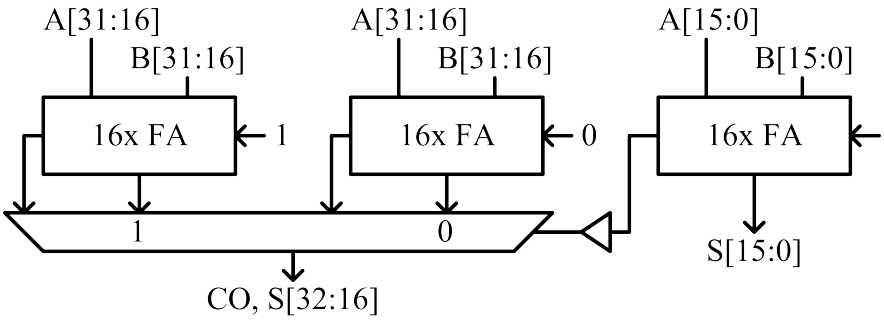
Adders can either be fast or small, but not both. Ripple adders are small and slow, but easy to make and understand. The Beta's speed can be significantly affected by adder speed, but its size won't be, so a faster adder can be useful. Your original adder has a critical path through every full adder for a total delay of 32. Any improvement should be better than that.



Did you make that ripple zero calculator? What assumptions is it based on?

Linear Carry Select Adders

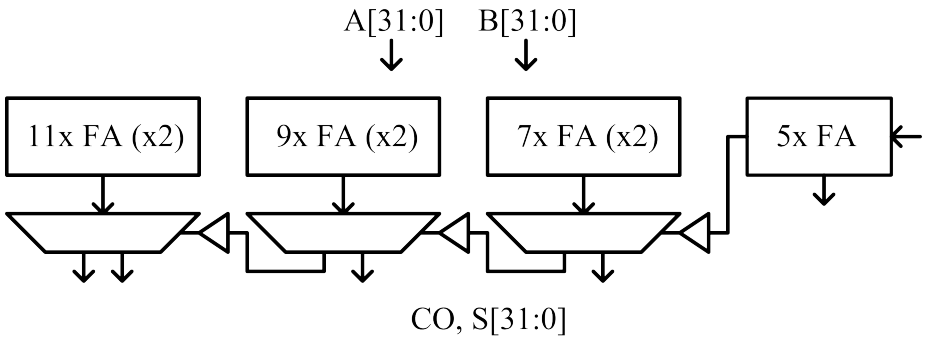
For the basics of Carry Select Adders, look at Fall 2010 Quiz 2 problem 3 as well as the Design Project handout. A linear carry select adder has ripple adder chunks of uniform length, so the critical path for a chunk size of 16 bits will have a delay of 18. The critical path goes through the low bits, then the mux to select the high bits. By the time the mux select arrives, the high order bits are also ready. You remembered to put a buffer in there, right? Relative to the ripple adder, the 16 bit chunk linear carry select adder is approximately 1.5 times the size for half of the critical path time. Overall this is a good tradeoff.



When the chunk size is reduced, there is a mismatch between the mux timing and the chunk timing. If the chunk size is 4 bits, the last mux select arrives with a delay of 10 (no buffers needed), which means the last sum bits have been waiting at the mux for 6. Part of optimizing adders is reducing this wasted time, so what can be done?

Logarithmic Carry Select Adders

In order to reduce wasted time among ripple adder chunks, the size of each chunk is modified so that for each mux, the select bit arrives at the same time as the sum bits. Adding 2 bits to the size of each successive chunk accounts for the buffer and mux between chunks. As an example, if the chunk sizes are 5,7,9, and 11, the last mux has a select input delay of 10 and a sum input delay of 11, for a total delay of 12. The hardware is about the same as a 16 bit chunk size linear carry select adder, but the overall delay is reduced by another 50%. You will need to tune the size and number of chunks to achieve optimal jsim timing, and keep in mind that it may benefit you to make sure that the lower 12 bits of your sum arrive quickly. Why?



Other Adder Structures

There are plenty of other adders you can use, but I believe the carry select adder is the sweet spot unless you are really interested in adders. If you are, do some research!

Pipelines and Critical Path Improvement

For a very thorough discussion of pipelining the Beta, see:

- Tutorial Problems – Quiz #3 – "Building the Beta" #1C
- Handouts – Lectures – L22: Pipelining the Beta
- Handouts – Lectures – L23: Pipeline Issues

…

Now that you have read about pipelining, some words of advice: The biggest benefit is from simple changes. You may want to start with a two stage Beta, but maybe there is a simpler change that is faster? Figure out for yourself what needs to be done. Get lots of scrap paper and make diagrams of the critical data paths in your Beta. Don't worry too much about calling things "stages" and making everything match that model. The model is only a starting point. Where do you need bypasses and where can you avoid them? Where do you need an extra clock cycle and where can you avoid it? Is there a better way to break up the critical path?

Advanced Steps

If you are a 6.004 student and this is the last week of classes, stop now. Your Benmark probably already got you the maximum point score and then some. When is that paper due?

…

Now then, if you just haven't had enough, here are a few hints to get you on your way to maximum Beta:

- Kogge-Stone adders are rather fast.
- Look at the bottom of stdcell.jsim. A latch? What is good about a latch?

*** The timing report doesn't work well with latches. ***

- How can buffers be useful if the same signal needs to be one place faster than others?
- Can you improve on the mux?
- Pipelines
  - Don't get too complex. Complex can turn into slow.
  - What ALU functions really need to make that clock edge?
  - Do all instructions need to take the same amount of time?
  - What is the limiting resource in your Beta and how effectively are you using it?
  - Could you go back to 2 port main memory and make good use of that?
- What is the minimum clock period you could possibly have?
- Extra clocks and reset signals are free!
- Register clock to Q is soo long. What can you do that is faster? What is the tradeoff?

Take 6.111, 6.374, or 6.012 for more low level goodness!