ECE 550: Fundamentals of Computer Systems and Engineering

Storage and Clocking

VHDL: Behavioral vs Structural

- A few words about this
  - Structural:
    - Spell out at (roughly) gate level
    - Abstract piece into entities for abstraction/re-use
    - Very easy to understand what synthesis does to it
  - Behavioral:
    - Spell out at higher level
    - Sequential statements, for loops, process blocks
    - Can be difficult to understand how it synthesizes
    - Difficult to resolve performance issues

Last time...

- Who can remind us what we did last time?

So far...

- We can make logic to compute “math”
  - Add, subtract,... (we'll see multiply/divide later)
  - Bitwise: AND, OR, NOT,...
  - Shifts
  - Selection (MUX)
  - ...pretty much anything
- But processors need state (hold value)
  - Registers
  - ...

Memory Elements

- All the circuits we looked at so far are combinational circuits: the output is a Boolean function of the inputs.
- We need circuits that can remember values. (registers)
- The output of the circuit is a function of the input AND a function of a stored value (state).
- Circuits with memory are called sequential circuits.
- Key to storage: loops from outputs to inputs
NOR-based Set-Reset (SR) Latch

Don’t set both S & R to 1

Set-Reset Latch (Continued)

Set Signal Goes High
Output Signal Goes High

Set Signal Goes Low
Output Signal Stays High

Set-Reset Latch (Continued)

SR Latch

- Downside: S and R at once = chaos
- Downside: Bad interface

- So let’s build on it to do better
Data Latch (D Latch)

We start with SR Latch

...and add some gates at the front which change the interface

Data + Enable

E goes high
D “latched”
Becomes new output

Slight Delay
(Logic gates take time)

E goes low
Output unchanged
By changes to D

$Q$
Logic Takes Time

- Logic takes time:
  - Gate delays: delay to switch each gate
  - Wire delays: delay for signal to travel down wire
  - Fan out: related to capacitance
- Need to make sure that signals timing is right
  - Don’t want to have races or whacky conditions..

Clocks

- Processors have a clock:
  - Alternates 0 1 0 1
  - Latch -> logic -> latch in one clock cycle
  - 3.4 GHz processor = 3.4 Billion clock cycles/sec

Strawman: Level Triggered

- First thoughts: Level Triggered
  - Latch enabled when clock is high
  - Hold value when clock is low

Strawman: Level Triggered

- How we’d like this to work
  - Clock is low, all values stable

Strawman: Level Triggered

- How we’d like this to work
  - Clock goes high, latches capture and xmit new val

Strawman: Level Triggered

- How we’d like this to work
  - Signals work their way through logic w/ high clk
Strawman: Level Triggered

• How we'd like this to work
  • Clock goes low before signals reach next latch

Clk

111 010
000 100

D latch

Logic

D latch

D latch

E latch

Clk

0

ECE 550 (Hilton): Storage and Clocking

Strawman: Level Triggered

• How we'd like this to work
  • Clock goes low before signals reach next latch

Clk

111 010
000 100

D latch

Logic

D latch

Clk

0

ECE 550 (Hilton): Storage and Clocking

Strawman: Level Triggered

• How we'd like this to work
  • Everything stable before clk goes high

Clk

111 010
000 100

D latch

Logic

D latch

Clk

0

ECE 550 (Hilton): Storage and Clocking

Strawman: Level Triggered

• How we'd like this to work
  • Clk goes high again, repeat

Clk

111 010
000 000

D latch

Logic

D latch

Clk

1

ECE 550 (Hilton): Storage and Clocking

Strawman: Level Triggered

• Problem: What if signal reaches latch too early?
  • I.e., while clk is still high

Clk

111 111
101 000

D latch

Logic

D latch

Clk

1

ECE 550 (Hilton): Storage and Clocking

Strawman: Level Triggered

• Problem: What if signal reaches latch too early?
  • Signal goes right through latch, into next stage..

Clk

111 111
101 000

D latch

Logic

D latch

Clk

1

ECE 550 (Hilton): Storage and Clocking
That would be bad...

- Getting into a stage too early is bad
  - Something else is going on there: corrupted
  - Also may be a loop with one latch

- Consider incrementing counter
  - Too fast: increment twice? Eeep...

![D Flip-Flop Diagram]

- Rising edge triggered D Flip-flop
  - Two D Latches w/ Opposite clking of enables

![D Flip-Flop Diagram]

- Rising edge triggered D Flip-flop
  - Two D Latches w/ Opposite clking of enables
  - On Low Clk, first latch enabled (propagates value)
  - Second not enabled, maintains value

- On High Clk, second latch enabled
  - First latch not enabled, maintains value

Edge Triggered

- Instead of level triggered
  - Latch a new value at a level (high or low)

- We use edge triggered
  - Latch a value at an edge (rising or falling)

![Edge Triggered Diagram]

- No possibility of “races” anymore
  - Even if I put 2 DFFs back-to-back...
  - By the time signal gets through 2nd latch of 1st DFF
    1st latch of 2nd DFF is disabled

- Still must ensure signals reach DFF before clk rises
  - Important concern in logic design "making timing"
Making Timing

- Making timing is important in a design
  - If you don’t make timing, your logic won’t compute right
- Synthesis tool (Quartus) tells you what max freq
  - Running above this your logic doesn’t “finish” in time

D Flip-flops (continued…)

- Could also do falling edge triggered
  - Switch which latch has NOT on clk
- D Flip-flop is ubiquitous
  - Typically people just say “latch” and mean DFF
  - Which edge: doesn’t matter
    - As long as consistent in entire design
    - We’ll use rising edge

D flip flops

- Generally don’t draw clk input
  - Have one global clk, assume it goes there
  - Often see > as symbol meaning clk
- Maybe have explicit enable
  - Might not want to write every cycle
  - If no enable signal shown, implies always enabled
- Get output and NOT(output) for “free”

DFFs in VHDL

```vhdl
x: dffe port map (
  clk => clk,       --the clock
  d => someInput,   --the input is d
  q => theOutput,   --the output is q
  ena => en,        --the enable
  clrn => not(rst), --clear
  prn => '1');      --set
```

Also, comes in “dff” with no enable

A word of advice

```vhdl
signal x_q : std_logic;
signal x_d: std_logic;
x : dffe port map (.., d=> x_d, q=>x_q,..);
```

- Use naming convention: x_d, x_q
- Write x_d, read x_q
- Remember new value shows up next cycle

A few words about timing

- Homework 2: VGA Controller
  - Requires certain clock frequency
  - Else won’t control monitor properly
- Quartus will tell you what timing you make
  - Fmax: how fast can this be clocked
  - Tells you your worst timing paths
  - From which dff to which dff
  - Can see in schematic viewer (usually)
- Homework 2
  - Should be plenty of slack
  - But if not...
Fixing timing misses

- Typical approach: reduce logic (gate delays)
  - Better adder?
  - Rethink approach?
  - Change "don’t care" behavior?
- Fix high fanout
  - Duplicate high FO/simple logic
- Also, feel free to ask for help from me/Tas
  - Quartus’s tools to help you fix them aren’t the best

Register File

- Can store one value... How about many?
- Register File
  - In processor, holds values it computes on
  - MIPS, 32 32-bit registers
  - How do we build a Register File using D Flip-Flops?
  - What other components do we need?

Register File: Interface

- 4 inputs
  - 3 register numbers (5 bit): 2 read, 1 write
  - 1 register write value (32 bits)
- 2 outputs
  - 2 register values (32 bits)

Register file strawman

- Use a mux to pick read?
  - 32 input mux = slow
  - (other regs not pictured)
- Writing the registers
  - Need to pick which reg
  - Have reg num (e.g., 19)
  - Need to make En19=1
    - En0, En1,... = 0

First: A Decoder

- First task: convert binary number to "one hot"
  - Saw this before
  - Take register number
Register File

- Now we know how to write:
  - Use decoder to convert reg # to one hot
  - Send write data to all regs
  - Use one hot encoding of reg # to enable right reg
- Still need to fix read side
  - 32 input mux (the way we've made it) not realistic
  - To do this: expand our world from 1, 0 to 1, 0, Z

CMOS: Complementary MOS

- 2 inputs: E and D. What does this do?
  - Write truth table for output

High Impedance: Z

- Z = High Impedance
  - No path to power or ground
  - "Gate" does not produce a 1 or a 0
- Previous slide: tri-state inverter
  - More commonly drawn: tri-state buffer
  - E = enable, D = data
  - Previous slide: tri-state inverter
    - More commonly drawn: tri-state buffer
    - E = enable, D = data

Remember this rule?

- Remember I told you not to connect two outputs?
  - If one gate tries to drive a 1 and the other drives a 0
    - One pumps water in.. The other sucks it out
    - Except its electric charge, not water
    - "Short circuit"—lots of current -> lots of heat
We’ve had this rule one day...

It's ok to connect multiple outputs together
Under one circumstance (*):
All but one must be outputting Z at any time

(*) Disclaimer: there are other circumstances... but not doing them now

Mux, implemented with tri-states

- We can build effectively a mux from tri-states
- Much more efficient for large #s of inputs (e.g., 32)

Ports

- What we just saw: read port
  - Ability to do one read / clock cycle
  - Originally said want 2 reads: read 2 src registers /insn
    - Maybe even more if we do many insns at once
- This design: can just put replicate
  - Another decoder
  - Another set of tri-states
  - Another output bus (wire connecting the tri-states)
- Earlier: write port
  - Ability to do one write/cycle
  - Could add more: need muxes to pick wr values

Minor Detail

- FYI: This is not how a register file is implemented
  - (Though it is how other things are implemented)
  - Actually done with SRAM
  - We'll see how those work soon

Summary

Can layout logic to compute things
  - Add, subtract,...
Now can store things
  - D flip-flops
  - Registers
Also understand clocks