More Digital Circuits
Type of Circuits

- **Synchronous Digital Systems** consist of two basic types of circuits:
  - **Combinational Logic (CL) circuits**
    - Output is a function of the inputs only, not the history of its execution
    - E.g., circuits to add A, B (ALUs)
  - **Sequential Logic (SL)**
    - Circuits that “remember” or store information
    - aka “State Elements”
    - E.g., memories and registers (Registers)
Uses for State Elements

• Place to store values for later re-use:
  • Register files (like x1-x31 in RISC-V)
  • Memory (caches and main memory)

• Help control flow of information between combinational logic blocks
  • State elements hold up the movement of information at input to combinational logic blocks to allow for orderly passage
Accumulator Example

Why do we need to control the flow of information?

Want: \( S = 0; \)

\[
\text{for } (i=0; i<n; i++) \\
S = S + X_i
\]

Assume:

- Each \( X \) value is applied in succession, one per cycle
- After \( n \) cycles the sum is present on \( S \)
First Try: Does this work?

No!
Reason #1: How to control the next iteration of the ‘for’ loop?
Reason #2: How do we say: ‘S=0’?
Register Internals

- n instances of a “Flip-Flop”
- Flip-flop name because the output flips and flops between 0 and 1
- D is “data input”, Q is “data output”
- Also called “D-type Flip-Flop”
Flip-Flop Operation

- Edge-triggered d-type flip-flop
  - This one is “positive edge-triggered”

- “On the rising edge of the clock, the input d is sampled and transferred to the output. At all other times, the input d is ignored.”

- Example waveforms:
Flip-Flop Timing

- Edge-triggered d-type flip-flop
- This one is “positive edge-triggered”

- “On the rising edge of the clock, the input d is sampled and transferred to the output. At all other times, the input d is ignored.”

- Example waveforms (more detail):

```
clk
\[\rightarrow\]

\(d\)
\[\rightarrow\]

\(q\)
\[\rightarrow\]

Input data must be stable in this period. "setup" time

"hold" time

"clk-to-q" delay
```
Camera Analogy Timing Terms

- Want to take a portrait – timing right before and after taking picture
- **Set up time** – don’t move since about to take picture (open camera shutter)
- **Hold time** – need to hold still after shutter opens until camera shutter closes
- **Time click to data** – time from open shutter until can see image on output (viewscreen)
Hardware Timing Terms

- **Setup Time**: when the input must be stable *before* the edge of the CLK
- **Hold Time**: when the input must be stable *after* the edge of the CLK
- **“CLK-to-Q” Delay**: how long it takes the output to change, measured from the edge of the CLK
So How To Build A Flip Flop? Two "Latches". An example...

- When clk is high...
  - $D \rightarrow Q$
- When clk is low...
  - $Q$ stays with whatever it was
- Chain 2 latches together to create a flip-flop
- Setup time:
  - Need to propagate $D$ to $Q$ on the first latch
- Hold time:
  - Need to make sure the first latch doesn't change before the clock fully switches
- Clk->Q time:
  - Time needed to go through the second latch
Accumulator Timing 1/2

- Reset input to register is used to force it to all zeros (takes priority over D input).
- $S_{i-1}$ holds the result of the $i^{th}$-1 iteration.
- Analyze circuit timing starting at the output of the register.
Accumulator Timing 2/2

- reset signal shown.
- Also, in practice $X$ might not arrive to the adder at the same time as $S_{i-1}$
- $S_i$ temporarily is wrong, but register always captures correct value.
- In good circuits, instability never happens around rising edge of $clk$. 
Model for Synchronous Systems

- Collection of Combinational Logic blocks separated by registers
- Feedback is optional
- Clock signal(s) connects only to clock input of registers
- Clock (CLK): steady square wave that synchronizes the system
- Register: several bits of state that samples on rising edge of CLK (positive edge-triggered) or falling edge (negative edge-triggered)
Maximum Clock Frequency

• What is the maximum frequency of this circuit?

Hint:
Frequency = 1/Period

Period = Max Delay = CLK-to-Q Delay + CL Delay + Setup Time
Critical Paths

Timing...

Note: delay of 1 clock cycle from input to output.
Clock period limited by propagation delay of adder/shifter.
Pipelining to improve performance

- Insertion of register allows higher clock frequency
- More outputs per second (higher bandwidth)
- But each individual result takes longer (greater latency)
Recap of Timing Terms

- **Clock (CLK)** - steady square wave that synchronizes system
- **Setup Time** - when the input must be stable before the rising edge of the CLK
- **Hold Time** - when the input must be stable after the rising edge of the CLK
- **“CLK-to-Q” Delay** - how long it takes the output to change, measured from the rising edge of the CLK
- **Flip-flop** - one bit of state that samples every rising edge of the CLK (positive edge-triggered)
- **Register** - several bits of state that samples on rising edge of CLK or on LOAD (positive edge-triggered)
Administrivia

- Project PAARRTTTAAYYY!!!
  - Wednesday, do go, its useful
- Yet Another Homework Released...
- Clicker attendance will NO LONGER COUNT starting this week for EPA...
  - Damn you COVAD-19!
Aside:
The 0, 1, ∞ Rule...

- When you are designing a computing system, you give a set of resources to the programmer
  - Give them none: RISC-V multiply in the base instruction set
  - Give them one: Instructions are executed one at a time
  - Give them ∞: Processes are given the illusion of effectively infinite memory (Virtual Memory) and ability to spawn infinite threads
- But ∞ is always a lie: our computers are always finite
  - So what happens as things approach or exceed the stated limit?
    - Lucky: Performance just degrades slowly
    - Common reality: Performance drops off a cliff!
      - We will see this when we talk about caches, virtual memory, and performance programming
The 0, 1, ∞ Rule and COVID-19

- For this audience, COVID-19 is really an annoying cold **even if you notice you are infected**
  - It is about as severe as asking for a beer and getting a Corona (*GAK*):
    Just wash your hands, stop touching your face, and don't bother buying a mask...

- But it is severe for older populations
  - It does appear to be significantly worse than run of the mill Influenza
  - And survivability **highly** depends on supportive health care

- The health care system is also promising ∞ that it can't deliver
  - So if there are too many patients, things will go to heck...

- Containment per-se is no longer an option:
  Public health is instead going to focus on limiting transmission rates
  - May very well include limiting large gatherings...
    Like, oh, say college campuses...
CS61C Plans: We *Gots* Them

- **IF** the University suspends classes for a few days or weeks:
  - Lecture will be webcast
  - Discussion sections & office hours on Google Hangouts and/or Zoom
  - Remote checkoff for Lab based on the autograder & a hangout

- **IF** the University suspends classes during Midterm 2:
  - Midterm 2 will just be dropped from the grading:
    All other components adjusted to compensate

- **IF** the University suspends classes during the Final:
  - The final will be shifted to a take-home exam:
    Fair bit of programming type problems
  - Uncertain yet whether it will be 1-day timed during dead RRR week or 3 hours timed at the final schedule
What is maximum clock frequency? (assume all unconnected inputs come from some register)

- A: 5 GHz
- B: 200 MHz
- C: 500 MHz
- D: 1/7 GHz
- E: 1/6 GHz

Clock->Q  1ns
Setup 1ns
Hold 1ns
AND delay 1ns
Problems With Clocking...

- The clock period **must be** longer than the critical path
  - Otherwise, you will get the wrong answers
  - But it can be even longer than that
- Critical path:
  - clk->q time
    - Necessary to get the output of the registers
  - **worst case** combinational logic delay
  - **Setup time** for the next register
  - Must meet all of these to be correct
Hold-Time Violations...

- An alternate problem can occur...
  - Clk->Q + **best case** combinational delay < Hold time...

- What happens?
  - Clk->Q + data propagates...
  - And now you don't hold the input to the flip flop long enough

- Solution:
  - **Add** delay on the best-case path (e.g. two inverters)
Finite State Machines (FSM) Intro

• A convenient way to conceptualize computation over time

• We start at a state and given an input, we follow some edge to another (or the same) state

• The function can be represented with a “state transition diagram”.

• With combinational logic and registers, any FSM can be implemented in hardware.
FSM Example: 3 ones…

FSM to detect the occurrence of 3 consecutive 1’s in the input.

Draw the FSM:

Assume state transitions are controlled by the clock:
On each clock cycle the machine checks the inputs and moves to a new state and produces a new output…
State Machine: Mealy v Moore Machines

- **Mealy machine:**
  - Output is a function of both input and current state
  - Draw the outputs on the transitions

- **Moore machine:**
  - Output is *only* a function of the current state
  - Draw the outputs with the state bubbles
FSM Example: 3 ones…
Moore Machine

FSM to detect the occurrence of 3 consecutive 1’s in the input.
Advantages and Disadvantages

- **Mealy Machine**
  - Smaller/fewer states
  - Potentially slower in terms of clock rate:
    - Critical path of the *system* has to include both the input and output logic and whoever is using it

- **Moore Machine**
  - Bigger/more states
  - Slower in terms of clock cycles to respond:
    - Going to take an extra cycle to give the output from the input
  - Potentially higher clock rate:
    - Critical path of the *system*
Hardware Implementation of FSM

...therefore a register is needed to hold the representation of which state the machine is in. Use a unique bit pattern for each state.

Combinational logic circuit is used to implement a function that maps from present state and input to next state and output.
FSM Combinational Logic

Specify CL using a truth table

**Truth table...**

<table>
<thead>
<tr>
<th>PS</th>
<th>Input</th>
<th>NS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>00</td>
<td>1</td>
</tr>
</tbody>
</table>
Alternate State Machines: One-Hot FSM

• Every State is a Register...
Building Standard Functional Units

- Data multiplexers
- Arithmetic and Logic Unit
- Adder/Subtractor
Data Multiplexer ("Mux") (here 2-to-1, n-bit-wide)
N instances of 1-bit-wide mux

How many rows in TT?

\[
c = \overline{s}ab + \overline{s}ab + s\overline{a}b + sab \\
= \overline{s}(a\overline{b} + ab) + s(ab + \overline{a}b) \\
= \overline{s}(a(\overline{b} + b)) + s((\overline{a} + a)b) \\
= \overline{s}(a(1) + s((1)b) \\
= \overline{s}a + sb
\]
How do we build a 1-bit-wide mux?

\[ \overline{s}a + sb \]
4-to-1 multiplexer?

How many rows in TT?

\[ e = \overline{s_1 s_0}a + \overline{s_1 s_0}b + s_1 \overline{s_0}c + s_1 s_0 d \]
Another way to build 4-1 mux?

Ans: Hierarchically! Hint: NCAA tourney!
Arithmetic and Logic Unit

• Most processors contain a special logic block called the “Arithmetic and Logic Unit” (ALU)
• We’ll show you an easy one that does ADD, SUB, bitwise AND, bitwise OR

\[
\text{when } S=00, \ R=A+B \\
\text{when } S=01, \ R=A-B \\
\text{when } S=10, \ R=A \ \text{AND} \ B \\
\text{when } S=11, \ R=A \ \text{OR} \ B
\]
Our simple ALU
How to design Adder/Subtractor?

• Truth-table, then determine canonical form, then minimize and implement as we’ve seen before

• Look at breaking the problem down into smaller pieces that we can cascade or hierarchically layer
Adder/Subtractor – One-bit adder LSB…

\[
\begin{array}{cccc}
\text{a}_3 & \text{a}_2 & \text{a}_1 & \text{a}_0 \\
\hline
\text{b}_3 & \text{b}_2 & \text{b}_1 & \text{b}_0 \\
\hline
\text{s}_0 & \text{s}_1 & \text{s}_2 & \text{s}_3
\end{array}
\]

<table>
<thead>
<tr>
<th>a_0</th>
<th>b_0</th>
<th>s_0</th>
<th>c_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[s_0 = \]  
\[c_1 = \]
Adder/Subtractor – One-bit adder (1/2)…”

\[
\begin{array}{c|c|c|c|c|c}
\hline
a_i & b_i & c_i & s_i & c_{i+1} \\
\hline
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 0 \\
0 & 1 & 1 & 0 & 1 \\
1 & 0 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 \\
\hline
\end{array}
\]
Adder/Subtractor – One-bit adder (2/2)

\[ s_i = \text{XOR}(a_i, b_i, c_i) \]
\[ c_{i+1} = \text{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i \]
N 1-bit adders $\Rightarrow$ 1 N-bit adder

What about overflow?
Overflow = $c_n$?
Extremely Clever Adder/Subtractor: "Invert and add one"

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>XOR(x,y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

XOR serves as conditional inverter!
iClicker Question

Convert the truth table to a boolean expression (no need to simplify):

A: \( F = xy + x(\neg y) \)

B: \( F = xy + (\neg x)y + (\neg x)(\neg y) \)

C: \( F = (\neg x)y + x(\neg y) \)

D: \( F = xy + (\neg x)y \)

E: \( F = (x+y)(\neg x+\neg y) \)

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>F(x,y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
In Conclusion

- Finite State Machines have clocked state elements plus combinational logic to describe transition between states
- Clocks synchronize D-FF change (Setup and Hold times important!)
- Standard combinational functional unit blocks built hierarchically from subcomponents