# CS61C – Machine Structures

## Lecture 22 - Brief Intro to Verilog, State Elements

10/17/2007

John Wawrzynek

(www.cs.berkeley.edu/~johnw)

www-inst.eecs.berkeley.edu/~cs61c/

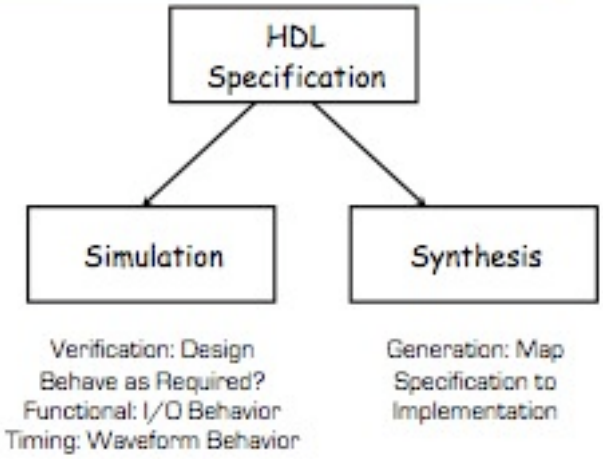


## Review

- All synchronous digital systems made up of combinational logic blocks and state elements.
- CL blocks can be described by their truth-table.
- State elements use to store values and control flow of data.
- Hierarchy important to manage design complexity.
- Data multiplexor important CL block for for data selection

## Intro to Hardware Description Languages

- **Basic Idea: Language constructs and code describe circuits**
- **Originally introduced for circuit simulation.**
  - **Now "logic synthesis" tools exist to automatically convert from language descriptions to design data for implementation.**




- **Warning! Even though we use language constructs to describe hardware that doesn't mean that hardware design is equivalent to writing software.**



## Verilog Hardware Description Language

- **One of two popular standard languages (VHDL is the other)**
  - **C-like syntax**
  - **widely used in industry**
- **We will describe our circuits in Verilog (class, labs, project) and use a simulator (ModelSim) to examine their behavior (*read the online tutorial*).**
  - **We will use a limited subset of the language.**
  - **Also, no time to implement (take CS150/152, EE141)**

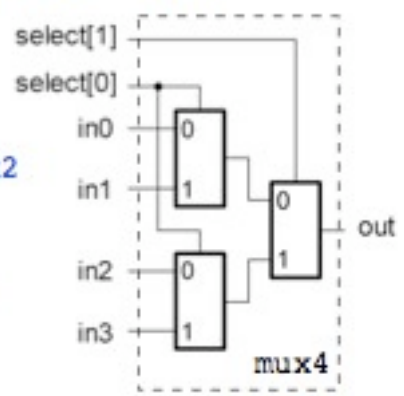
## *Very* Brief Verilog Introduction

- the `module` describes a component in the circuit
- Two ways to describe:
  - Structural Verilog
    - List of components and how they are connected
    - One-to-one correspondence to the actual circuit
    - *The* way we will describe our circuits
  - Behavioral Verilog
    - Describe *what* a component does, not *how* it does it
    - Useful before details of the actual circuit is worked out
    - Useful for describing the test algorithm for another circuit
- Build up a hierarchy of modules. Top-level module is your entire design (or the tester).



## Verilog Example: 4-input multiplexor

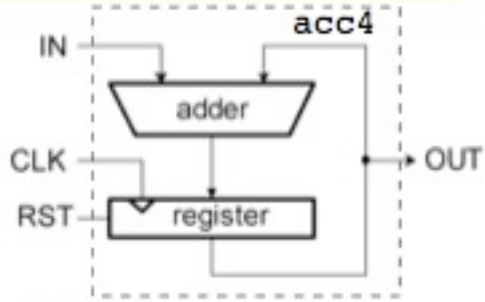
```
//4-input multiplexor built from 3 2-input multiplexors
module mux4 (in0, in1, in2, in3, select, out);
  input in0,in1,in2,in3;
  input [1:0] select;
  output out;
  wire w0,w1;
  mux2 // Declare instances of mux2
   m0 (in0, in1, select[0], w0),
   m1 (in2, in3, select[0], w1),
   m3 (w0,  w1,  select[1], out);
endmodule // end of mux4
```




- Assumes we previously defined a verilog module for a 2-input mux with the following interface:

```
module mux2 (in0, in1, select, out);
```

## Verilog Example: Accumulator




```
//Accumulator
module acc4 (CLK,RST,IN,OUT);
  input CLK, RST;
  input [3:0] IN;
  output [3:0] OUT;
  wire [3:0] W0;
  add4 myAdd (.A(IN), .B(OUT), S(W0));
  reg4 myReg (.RST(RST), .D(W0), .Q(OUT), .CLK(CLK));
endmodule // acc4
```

- **Makes use of two previously defined modules:**

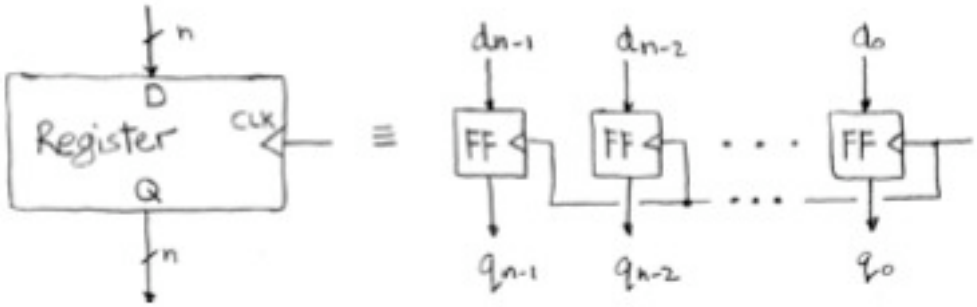
```
module add4 (S,A,B);

module reg4 (CLK,Q,D,RST);
```



## Last word (*for now*) on Verilog

- **Verilog is a very rich language with many features.**
  - **Many not necessary for CS61c**
    - **We focus on "structural only" and a few procedural constructs for testing**
  - **This week: very simple use in lab (similar to class examples)**
  - **More next week and on project**
- **Tutorials online:**
  - **Verilog, Modelsim**
  - **much more information on the web**

## Register Details…What's inside?



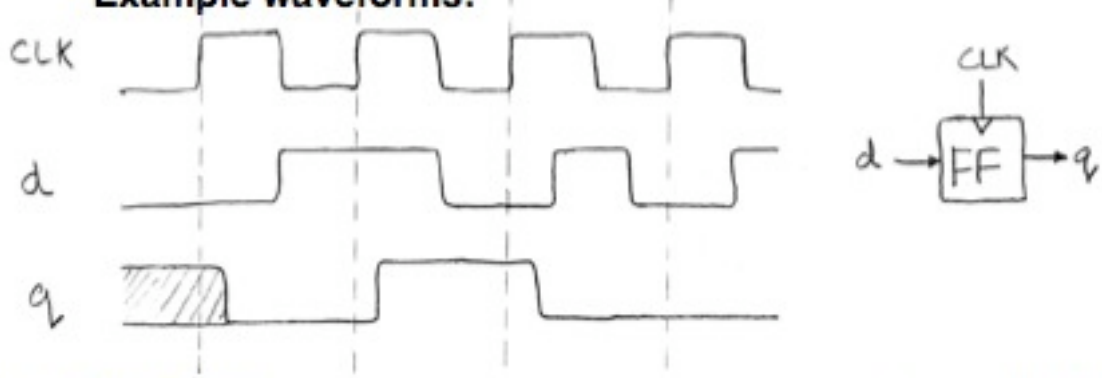

- n instances of a "Flip-Flop"
- Flip-flop name because the output flips and flops between and 0,1
- D is "data", Q is "output"
- Also called "d-type Flip-Flop"



## Flip-flop Timing? (1/2)

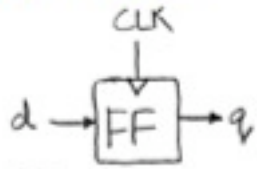
- Edge-triggered d-type flip-flop
  - This one is "positive edge-triggered"

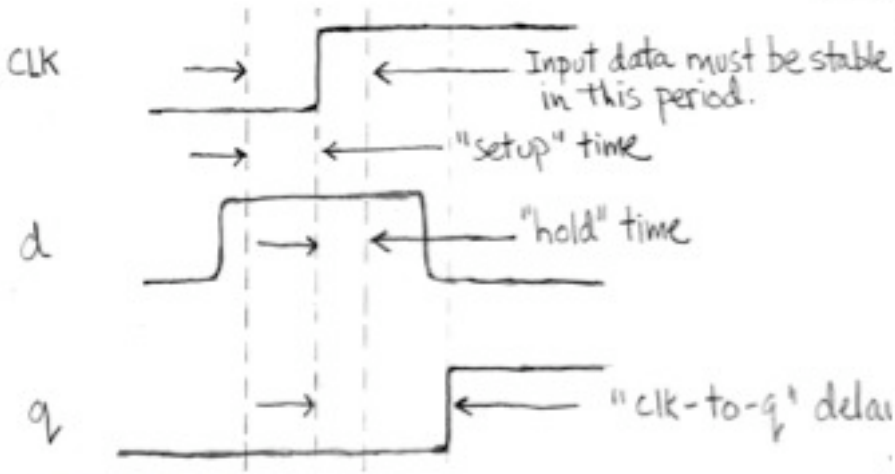
- "On the rising edge of the clock, the input d is sampled and transferred to the output. At all other times, the input d is ignored."
- Example waveforms:



# Flip-flop Timing? (2/2)

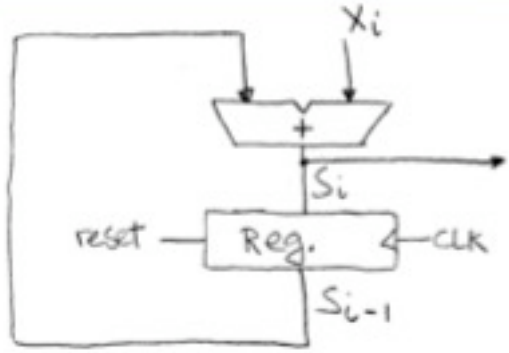
- "On the rising edge of the clock, the input d is sampled and transferred to the output. At all other times, the input d is ignored."
- **Constraints:**



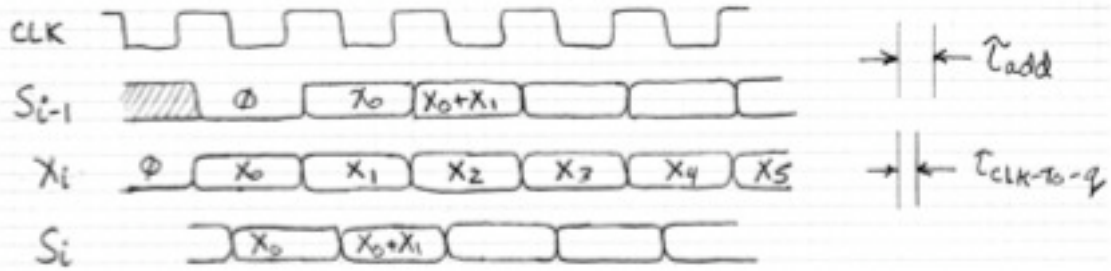





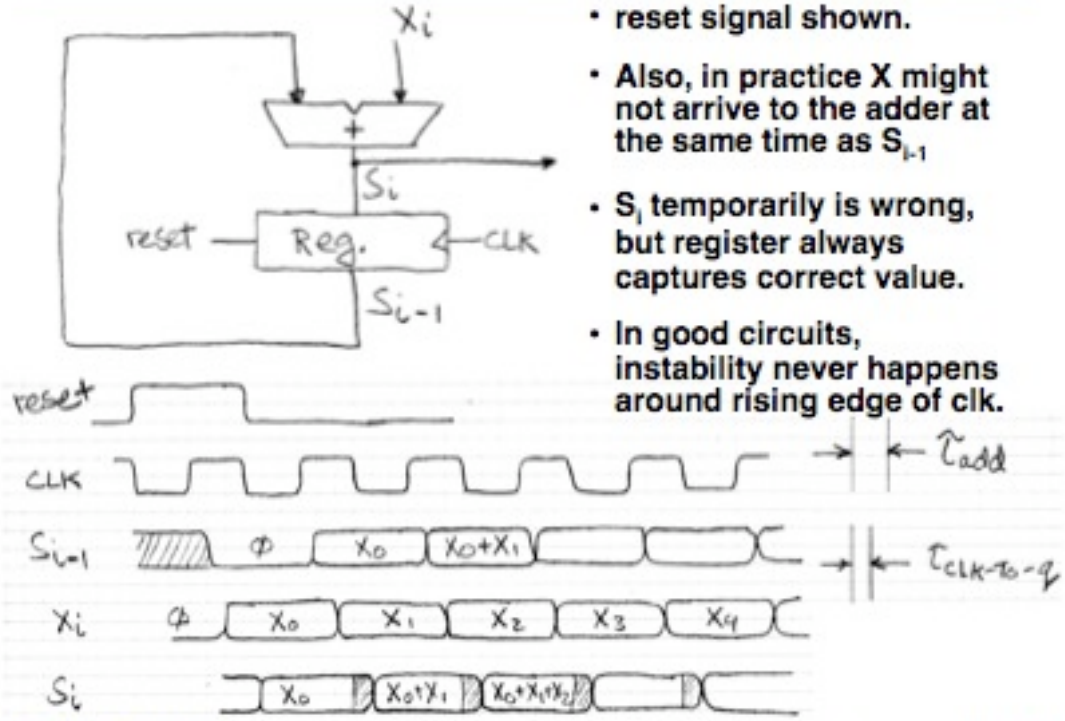

# Accumulator Revisited (proper timing 1/2)




- 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 Revisited (proper timing 2/2)

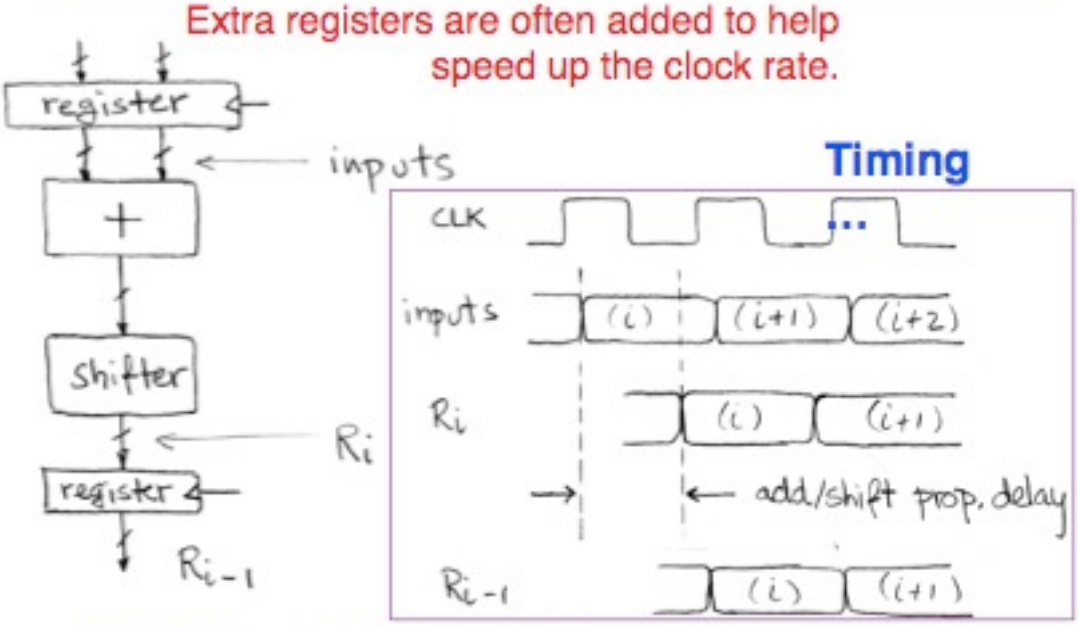



- reset signal shown.
- Also, in practice X might not arrive to the adder at the same time as $S_{i-1}$
- $S_i$ temporarily is wrong, but register always captures correct value.
- In good circuits, instability never happens around rising edge of clk.



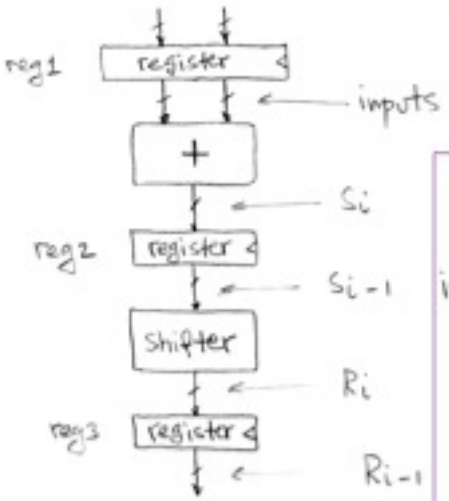
# Pipelining to improve performance (1/2)

Extra registers are often added to help speed up the clock rate.



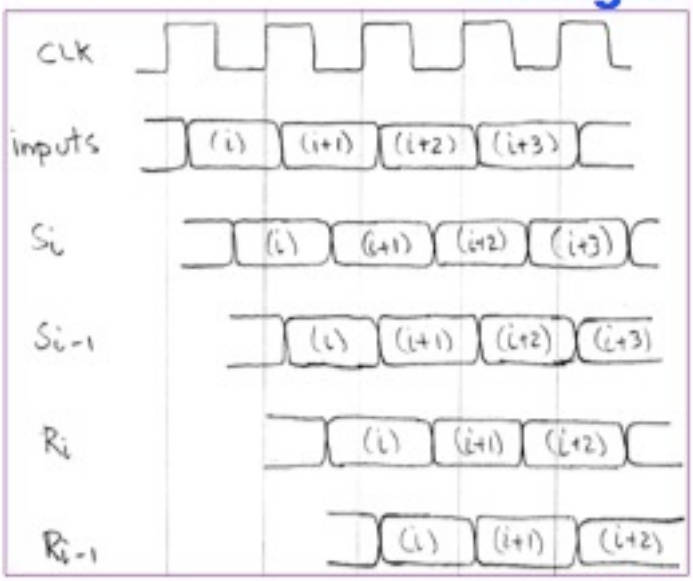

Note: delay of 1 clock cycle from input to output.
Clock period limited by propagation delay of adder/shifter.

# Pipelining to improve performance (2/2)




- Insertion of register allows higher clock frequency.
- More outputs per second.

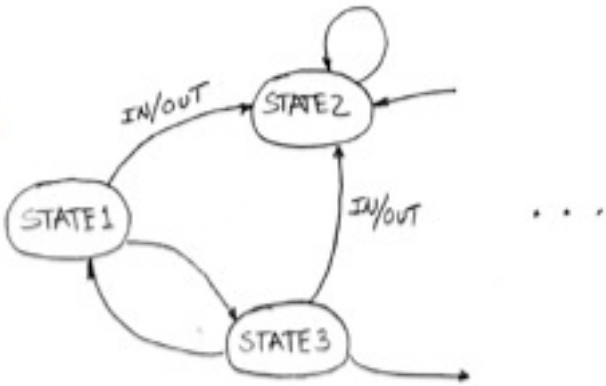
**Timing…**





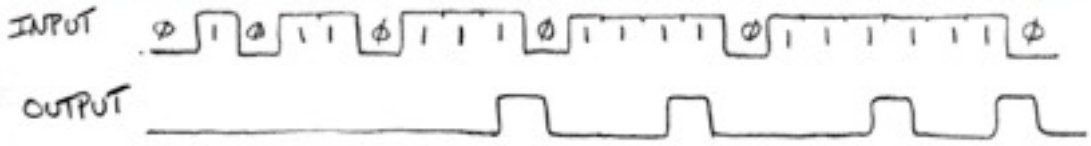

# Finite State Machines (FSM) Introduction

- You have seen FSMs in other classes.
- Same basic idea.
- The function can be represented with a "state transition diagram".
- With combinational logic and registers, any FSM can be implemented in hardware.

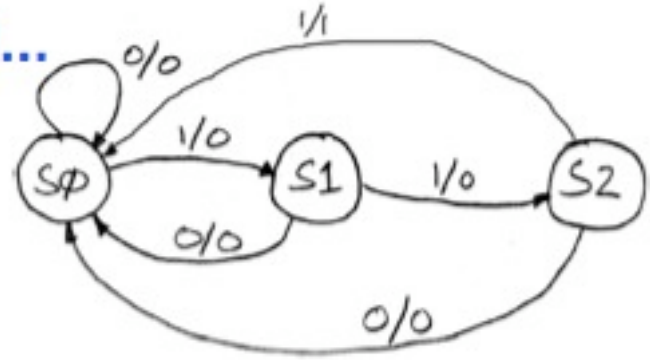


# Finite State Machine Example: 3 ones…

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




## Draw the FSM…

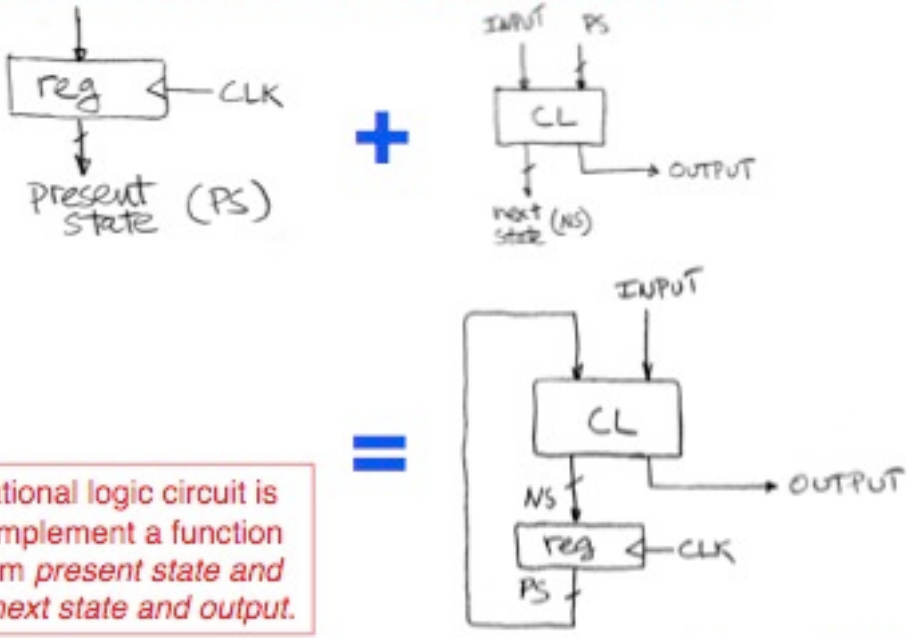



Assume state transitions are controlled by the clock:
on each clock cycle the machine checks the inputs and moves to a new state and produces a new output…



# Hardware Implementation of FSM

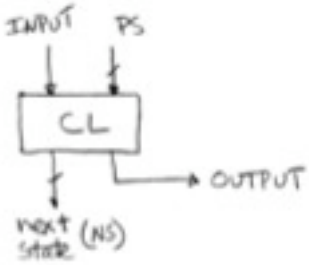
… Therefore a register is needed to hold a representation of which state the machine is in. Use a unique bit pattern for each state.

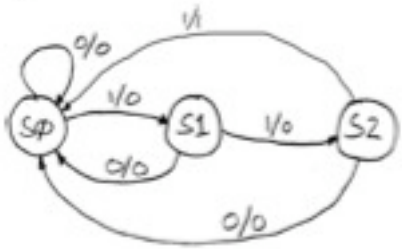



Combinational logic circuit is used to implement a function maps from *present state and input* to *next state and output.*

## Hardware for FSM: Combinational Logic

Next lecture we will discuss the detailed implementation, but for now can look at its functional specification, truth table form.




Let: S0 = 00
S1 = 01
S2 = 10

### Truth table…

| PS | Input | NS | Output |
|---|---|---|---|
| 00 | 0 | 00 | 0 |
| 00 | 1 | 01 | 0 |
| 01 | 0 | 00 | 0 |
| 01 | 1 | 10 | 0 |
| 10 | 0 | 00 | 0 |
| 10 | 1 | 00 | 1 |




## Peer Instruction

**A. The function of the accumulator circuit could be replaced by one that operates in a single clock cycle. (Assuming all X inputs are available at the same time)**

**B. A "hardware" stack could be built with nothing but registers.**

**C. You can build a FSM to signal when an equal number of 0s and 1s has appeared at the input.**

```
   ABC
1: FFF
2: FFT
3: FTF
4: FTT
5: TFF
6: TFT
7: TTF
8: TTT
```

## "And In conclusion…"

- Verilog is used to describe and simulate hardware.
  - Simple sub-set used in 61c
- D-flip-flops used to build registers
- Clocks tell us when D-flip-flops change
  - Setup and Hold times put constraints on use
- We pipeline long-delay CL for faster clock
- Finite State Machines extremely useful
  - You'll see them again (150,152) & 164