AI, captain! First autonomous ship prepares for maiden voyage

“The "Mayflower 400"—the world's first intelligent ship—bobs gently in a light swell as it stops its engines in Plymouth Sound, off England's southwest coast, before self-activating a hydrophone designed to listen to whales.”

“The 50-foot (15-metre) trimaran, which weighs nine tonnes and navigates with complete autonomy, is preparing for a transatlantic voyage…”

“The autonomous ship is scheduled to embark on May 15 ... The journey to Plymouth, Massachusetts … will take three weeks.”

https://techxplore.com/news/2021-04-ai-captain-autonomous-ship-maiden.html

Clock Divider (not for simulation)

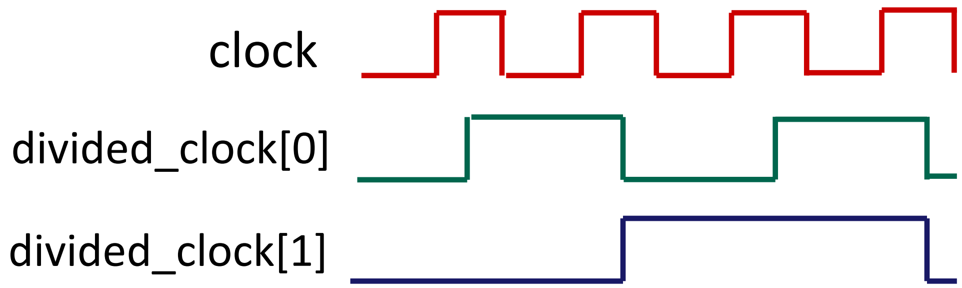
- Why/how does this work?

```
// divided_clocks[0]=25MHz, [1]=12.5Mhz, ...
module clock_divider (clock, divided_clocks);
   input  logic        clock;
   output logic [31:0] divided_clocks;

   initial
     divided_clocks = 0;

   always_ff @(posedge clock)
     divided_clocks <= divided_clocks + 1;

endmodule
```



Outline

- FSM Design
- Multiplexors
- Adders

FSM Design Process

1) Understand the problem

2) Draw the state diagram

3) Use state diagram to produce truth table

4) Implement the combinational control logic

Practice: String Recognizer FSM

- Recognize the string `101` with the following behavior
  - Input: `1 0 0 1 0 1 0 1 1 0 0 1 0`
  - Output: `0 0 0 0 0 1 0 1 0 0 0 0 0`
- State diagram to implementation:

Subdividing FSMs

- Some problems best solved with multiple pieces

- "Psychic Tester"
  - Machine generates a 4-bit pattern
  - User tries to guess 8 patterns in a row to be deemed psychic

- States?

Subdividing FSMs

- Pieces?
  - Generate/pick pattern
  - User input (guess)
  - Check guess
  - Count correct guesses

Subdividing FSMs

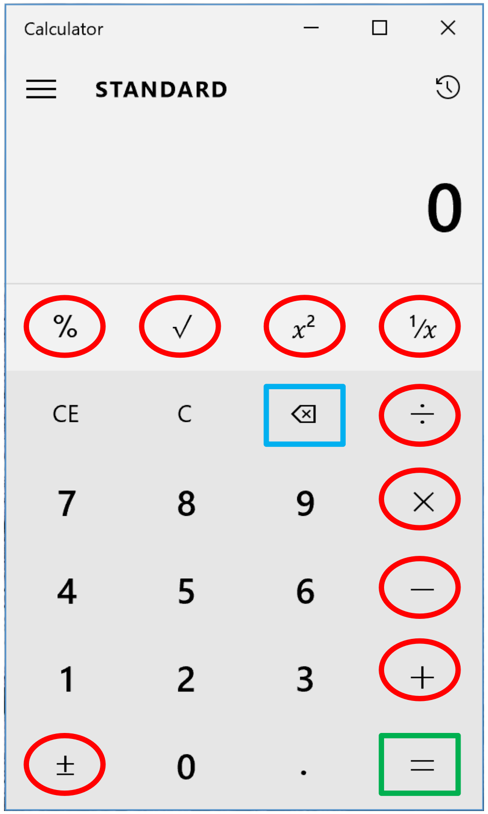
- Pieces?
  - Generate/pick pattern
    - `module genPatt(pattern, next, clock);`
  - User input (guess)
    - `module userIn(guess, enable, SW);`
  - Check guess
    - `module checkGuess(correct, guess, pattern);`
  - Count correct guesses
    - `module countRight(psychic, next, correct, enable, clock);`

Outline

- FSM Design
- Multiplexors
- Adders

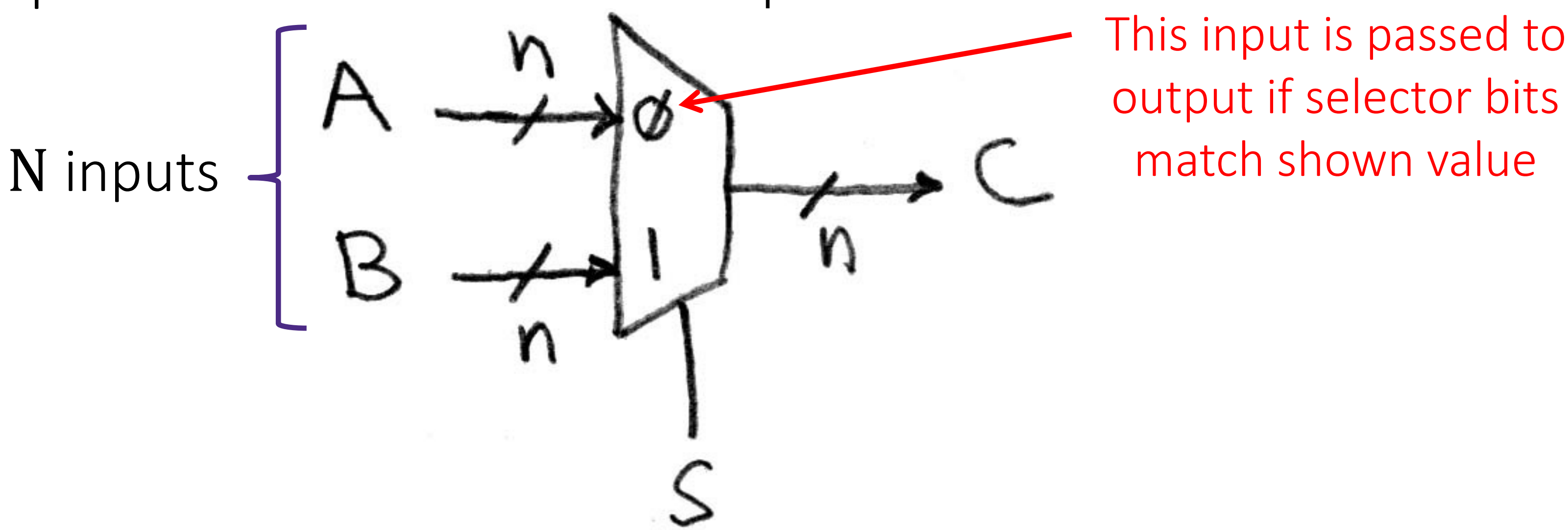
Motivating Example

- **Problem:** Implement a simple pocket calculator
- **Need:**
  - Display: Seven segment displays
  - Inputs: Buttons
  - Math: Arithmetic & Logic Unit (ALU)



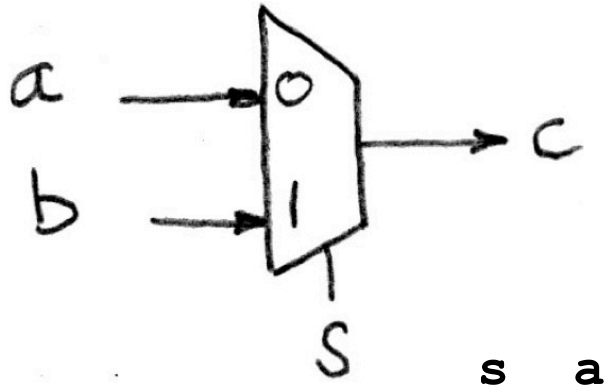
Data Multiplexor

- Multiplexor ("MUX") is a *selector*
  - Called a n-bit, N-to-1 MUX
  - Direct one of many n-bit wide inputs onto output
  - s selector bits with N inputs ($\mathrm{N} = 2^s$)
- <u>Example</u>: n-bit 2-to-1 MUX
  - Input S selects between two inputs of n bits each



Review: Implementing a 1-bit 2-to-1 MUX

- Schematic:



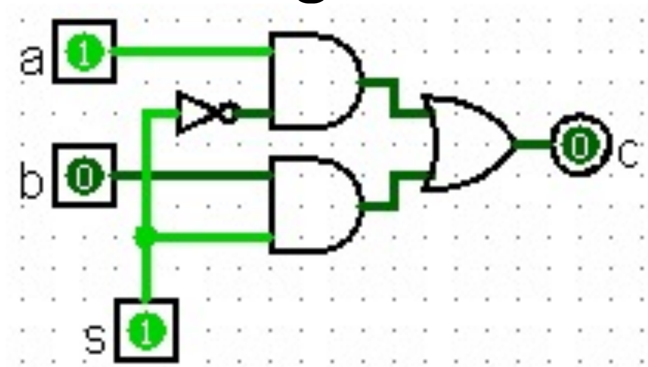
- Truth Table:

s	a	b	c
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- Boolean Algebra:

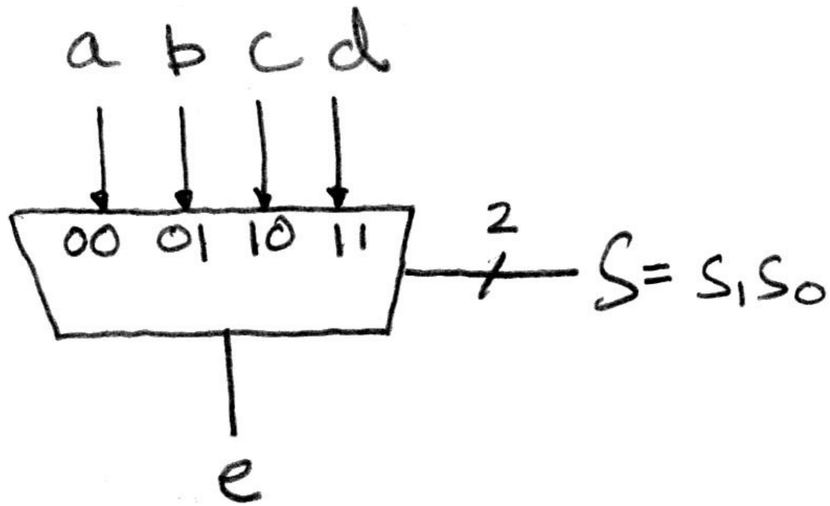
$$\begin{aligned} c &= \overline{s}a\overline{b} + \overline{s}ab + s\overline{a}b + sab \\ &= \overline{s}(a\overline{b} + ab) + s(\overline{a}b + ab) \\ &= \overline{s}(a(\overline{b} + b)) + s((\overline{a} + a)b) \\ &= \overline{s}(a(1) + s((1)b) \\ &\boxed{= \overline{s}a + sb} \end{aligned}$$

- Circuit Diagram:



1-bit 4-to-1 MUX

- Schematic:



- Truth Table: How many rows?
- Boolean Expression:

$$e = \overline{s_1}\,\overline{s_0}a + \overline{s_1}s_0b + s_1\overline{s_0}c + s_1s_0d$$

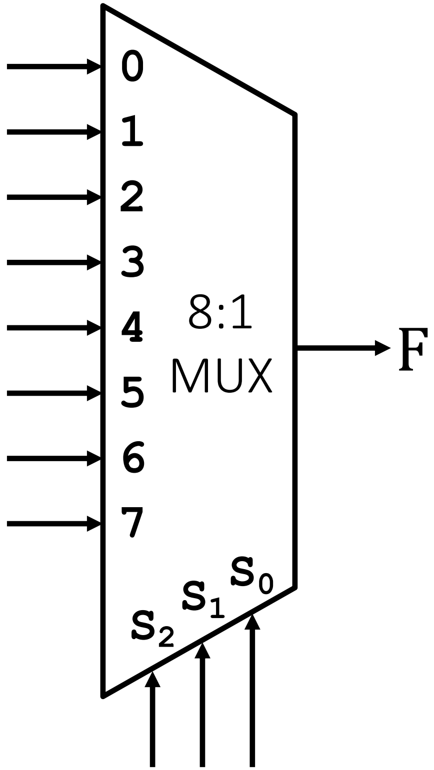
1-bit 4-to-1 MUX

- Can we leverage what we’ve previously built?

Multiplexers in General Logic

- Implement $F = X\overline{Y}Z + Y\overline{Z}$ with a 8:1 MUX

X	Y	Z	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



Technology Break

https://xkcd.com/257/

Outline

- FSM Design
- Multiplexors
- Adders

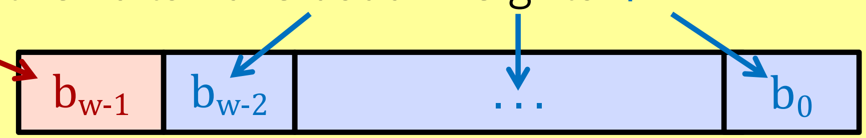
Review: Unsigned Integers

- Unsigned values follow the standard base 2 system
  - $b_7b_6b_5b_4b_3b_2b_1b_0 = b_7 2^7 + b_6 2^6 + \cdots + b_1 2^1 + b_0 2^0$
- In n bits, represent integers 0 to 2^n-1
- Add and subtract using the normal "carry" and "borrow" rules, just in binary

```
  63      00111111          64      01000000
+  8     +00001000        -  8     -00001000
  71      01000111          56      00111000
```

Review: Two's Complement (Signed)

b_{w-1} has weight -2^{w-1}, other bits have usual weights $+2^{i}$



- **Properties:**
  - In n bits, represent integers -2^{n-1} to $2^{n-1}-1$
  - Positive number encodings match unsigned numbers
  - Single zero (encoding = all zeros)
- **Negation procedure:**
  - Take the bitwise complement and then add one
    `( ~x + 1 == -x )`