Discussion Section 4

Sean Huang
February 12, 2021

Berkeley
UNIVERSITY OF CALIFORNIA

Karnaugh Map Example

Simplify this expression:

$$out = a'b'c'd' + a'b'c'd + a'b'cd' + a'bc'd' + \\ +a'bc'd + a'bcd + ab'c'd' + ab'cd' \\ +ab'cd + abc'd + abcd' + abcd$$

Karnaugh Map Example

Simplify this expression:

$$out = a'b'c'd' + a'b'c'd + a'b'cd' + a'bc'd' + \\ +a'bc'd + a'bcd + ab'c'd' + ab'cd' \\ +ab'cd + abc'd + abcd' + abcd$$

a	b	c	d	out
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Karnaugh Map Example

Simplify this expression:

$$out = a'b'c'd' + a'b'c'd + a'b'cd' + a'bc'd' + \\ + a'bc'd + a'bcd + ab'c'd' + ab'cd' \\ + ab'cd + abc'd + abcd' + abcd$$

#	a	b	c	d	out
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Karnaugh Map Example

- Truth Table maps to K-map
  - Can fill out K-map “in order”
- Squares in K-map not in same order as TT
  - Gray code sequencing inputs reorders terms
- Each box containing a 1 is a minterm

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Karnaugh Map Example

$$out = \overset{0}{a'b'c'd'} + \overset{1}{a'b'c'd} + \overset{2}{a'b'cd'} + \overset{4}{a'bc'd'}$$

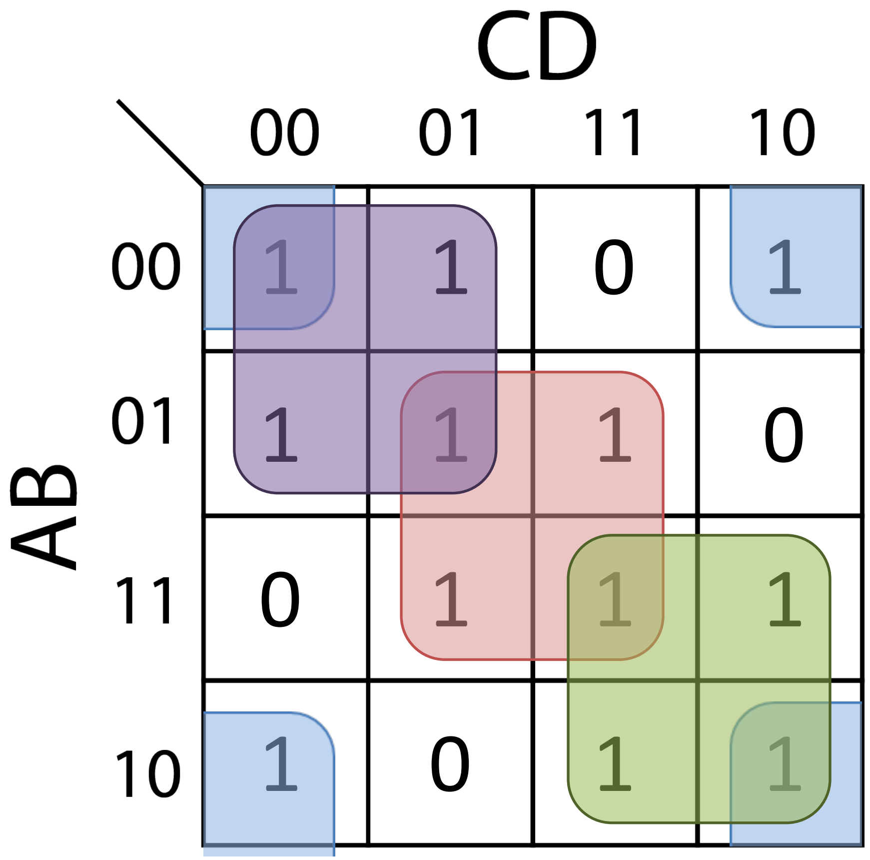
$$+\overset{5}{a'bc'd} + \overset{7}{a'bcd} + \overset{8}{ab'c'd'} + \overset{10}{ab'cd'}$$

$$+\overset{11}{ab'cd} + \overset{13}{abc'd} + \overset{14}{abcd'} + \overset{15}{abcd}$$

AB \ CD	00	01	11	10
00	1	1	0	1
01	1	1	1	0
11	0	1	1	1
10	1	0	1	1

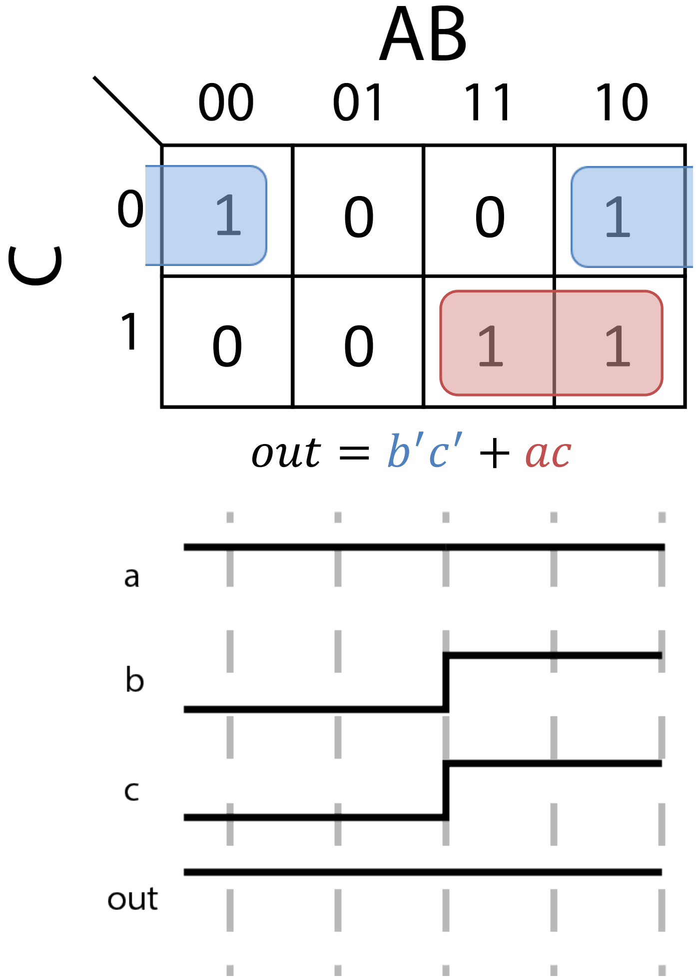
Karnaugh Map Example

- Group minterms together
  - Groups also called *implicants*
  - Each group is a simplified product term omitting one or more variable
  - Implicant groups can wrap around edges/corners
- $out = bd + b'd' + ac + a'c'$
- When all minterms are grouped and no bigger groups can be made, they are known as *prime implicants*



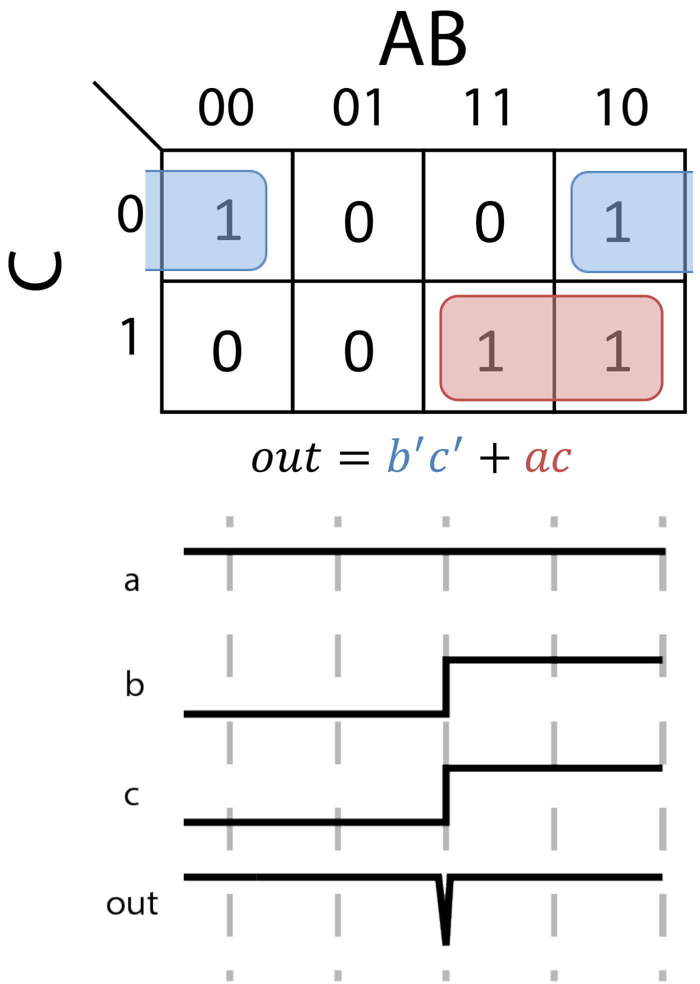
Karnaugh Maps and Glitches

- Prime implicants that do not overlap are called *essential prime implicants*
  - Both terms cannot be true at the same time
- Output works fine if both AND gates are identical and arrive at OR at the same time
  - This is unrealistic. Real gates have varying delays



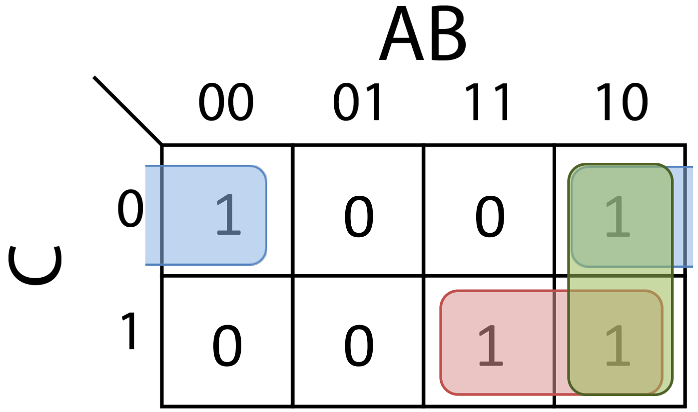
Karnaugh Maps and Glitches

- Prime implicants that do not overlap are called *essential prime implicants*
  - Both terms cannot be true at the same time
- Creates possibility of a glitch
  - Also known as a *static hazard*
- If one term evaluates to false before the other term evaluates to true, output can momentarily glitch

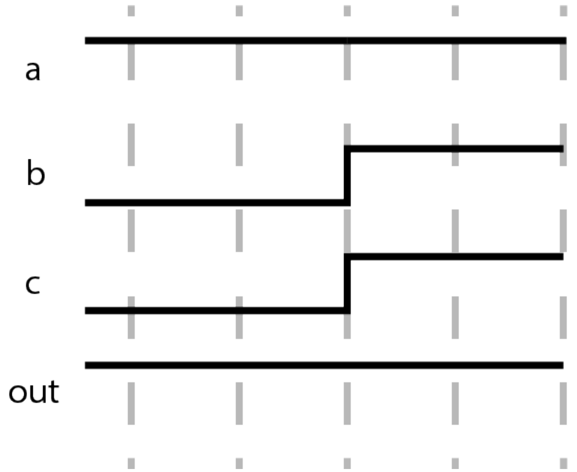


Karnaugh Maps and Glitches

- Prime implicants that do not overlap are called *essential prime implicants*
  - Both terms cannot be true at the same time
- Creates possibility of a glitch
  - Also known as a *static hazard*
- If one term evaluates to false before the other term evaluates to true, output can momentarily glitch
- Can avoid with redundant terms
  - Output “covered” by additional term



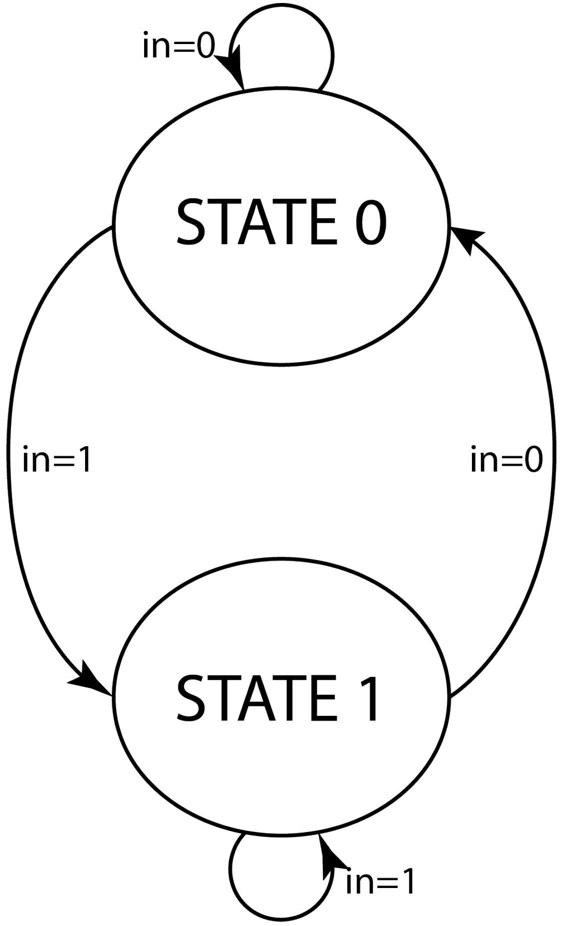
$$out = b'c' + ac + ab'$$



Finite State Machines

Finite State Machines

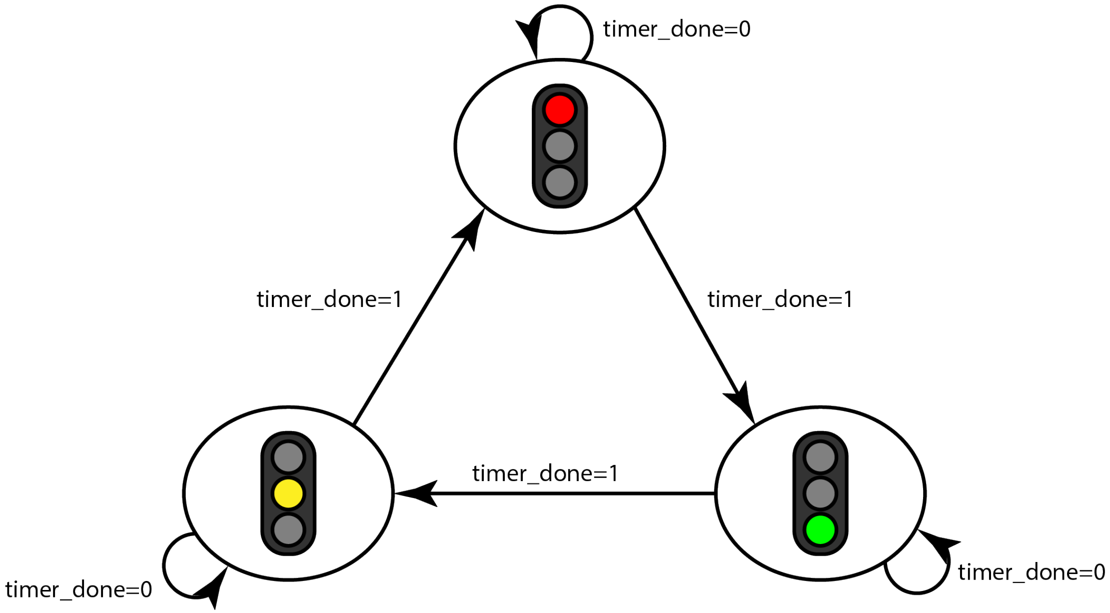
- Simple automata
- All sequential systems can be modeled as an FSM
- Useful abstraction of stateful behavior
- Represented with State Transition Diagrams
  - Describes trajectory of state machine depending on inputs
  - Usually traverse an edge every clock cycle



Berkeley
UNIVERSITY OF CALIFORNIA

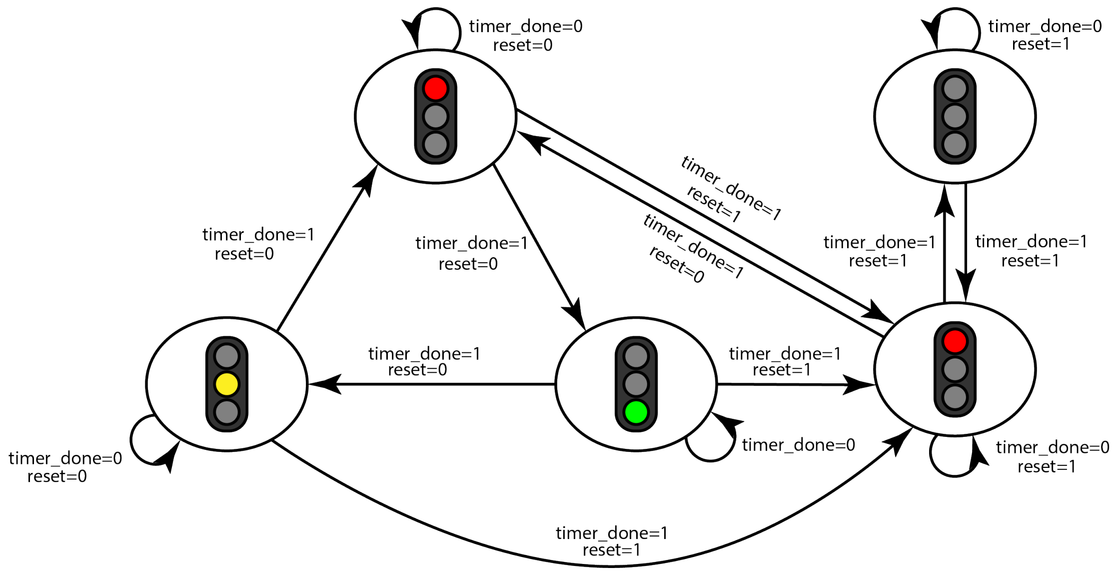
Simple Example (Traffic Lights)

- Simple timer-based traffic light system
- Each light is a particular state
  - Light changes once timer runs out
- Each edge depends on same input signal
  - Which state is next depends on current state



Less Simple Example (Traffic Lights)

- Reset to a flashing red light until reset goes low again
  - Add “off” state, and new “red_rst” state
  - New states flash back and forth until reset goes low
  - All other states reset to “red_rst” state until ready to continue normal operation



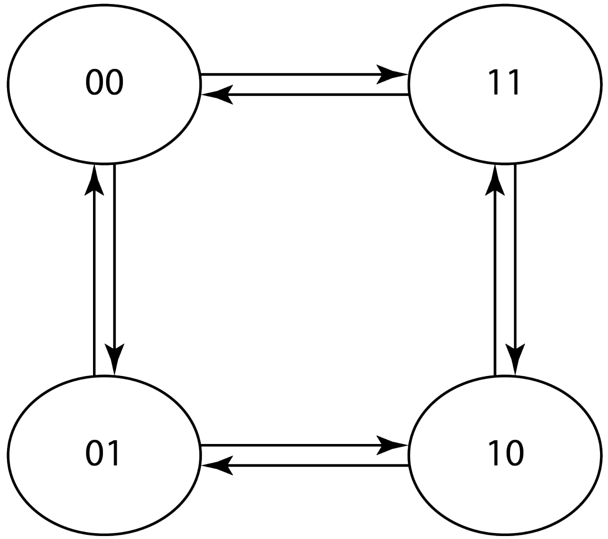
Implementing FSMs

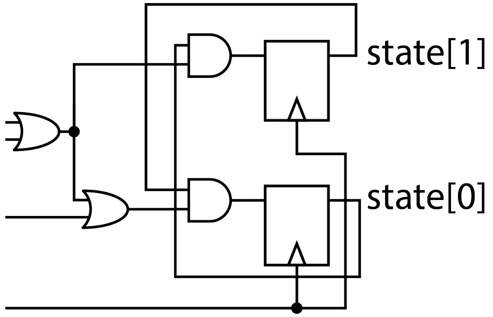
- FSMs implemented in hardware with registers and logic
- How to keep track of current state?
- Could store a number in the registers
  - What format to use?
    - Binary
    - One-hot
    - Gray coding
    - Something else?

Berkeley
UNIVERSITY OF CALIFORNIA

Binary Encoding

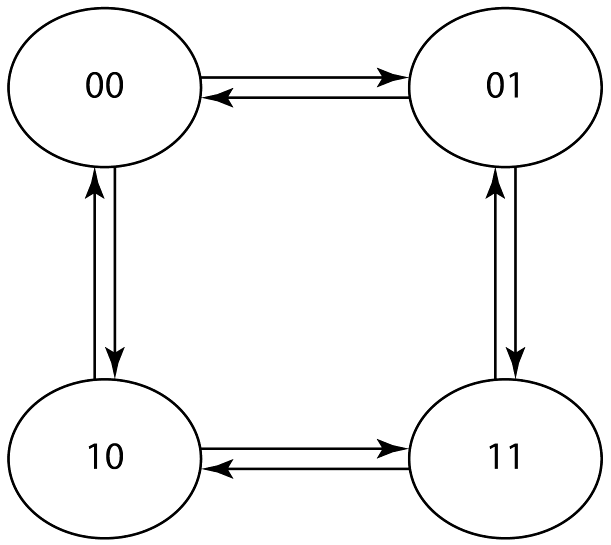
- Probably first idea you come to
- Encode each state as binary number
- Next state depends on current state code and inputs
- Code efficiently uses registers 🙂
  - Only need $\log_2(\text{N\_States})$ bits
- Per-bit logic can get complicated 🙁
  - State transitions may involve changing several bits
  - Outputs may need to be decoded

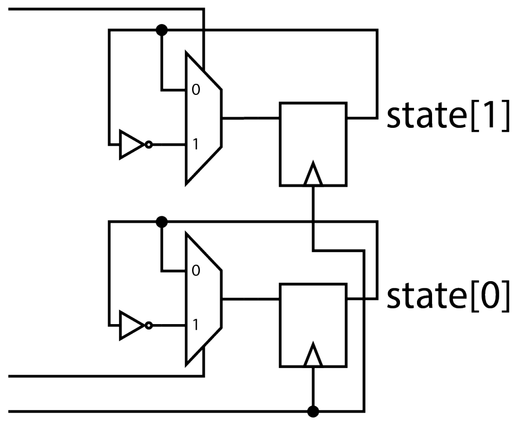




Gray Code Encoding

- Modification on Binary Encoding
- Only allow one bit to change at a time
- Simpler per-bit logic 🙂
  - Basically just decide which bit to change at each edge

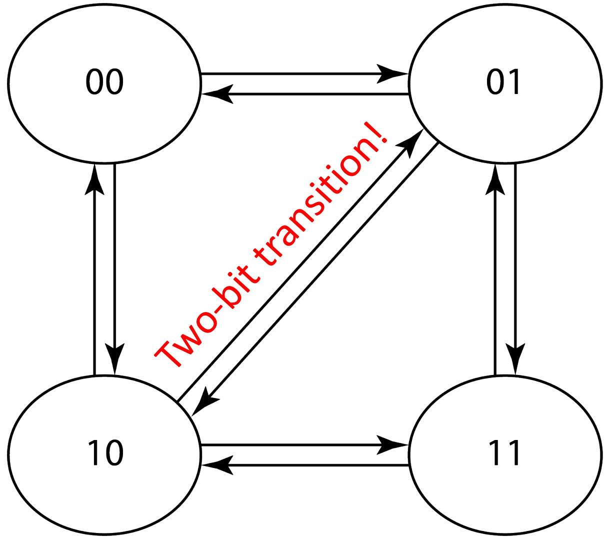


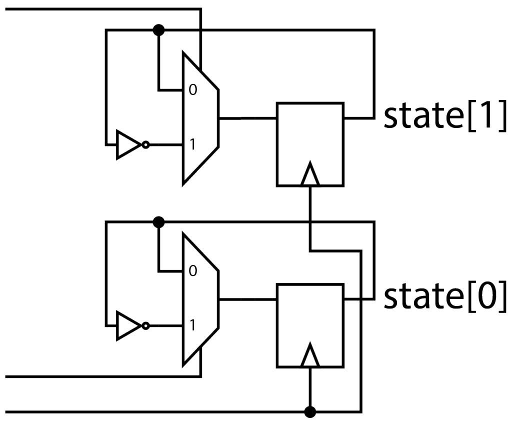


Berkeley
UNIVERSITY OF CALIFORNIA

Gray Code Encoding

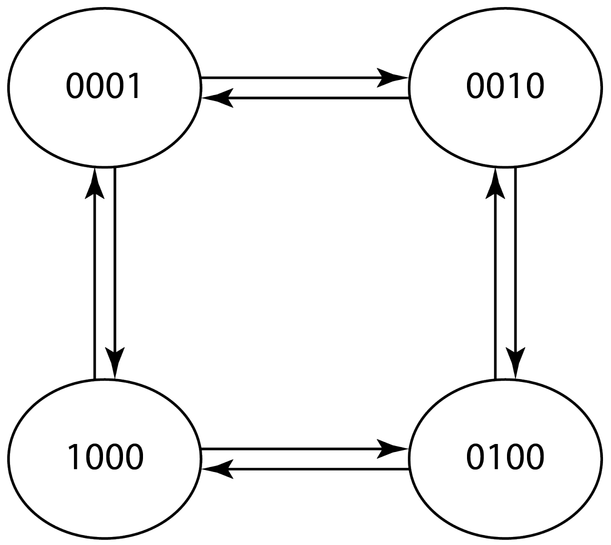
- Modification on Binary Encoding
- Only allow one bit to change at a time
- Simpler per-bit logic 🙂
  - Basically just decide which bit to change at each edge
- Not all state diagrams work 🙁
  - Some state changes cannot be Gray coded

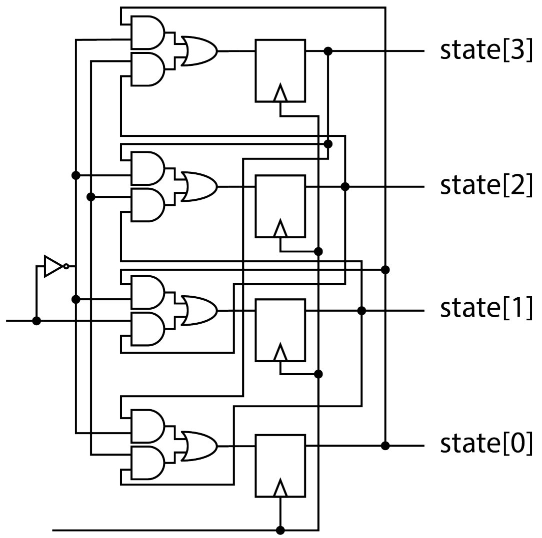




One-Hot Encoding

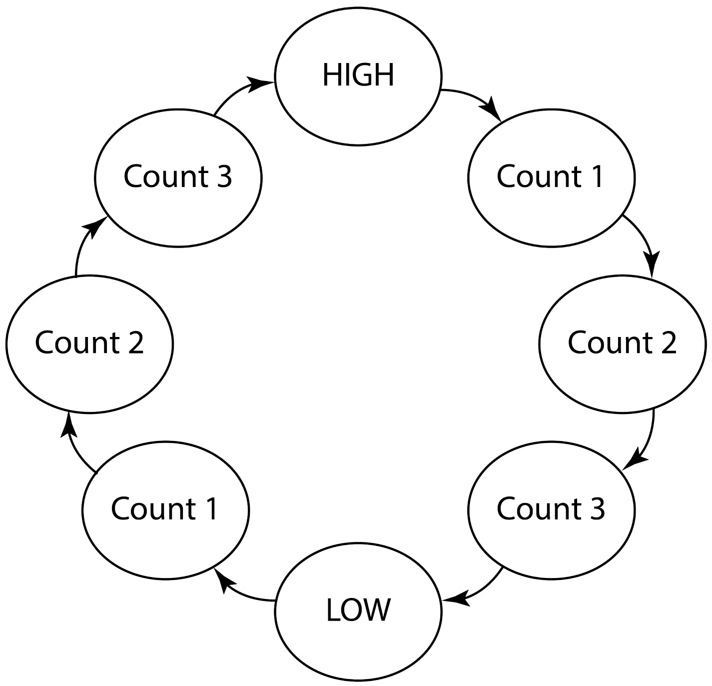
- Only 1 wire high at a time
- Each wire represents a state
- Easier to design and debug 🙂
  - Can design per-bit logic in isolation
  - Each register responsible for a state, easy to find current state from waveform/log
  - Maps well to FPGAs
- Not register efficient 🙁
  - Per-bit logic can get even messier than binary
  - Needs N_States registers/wires

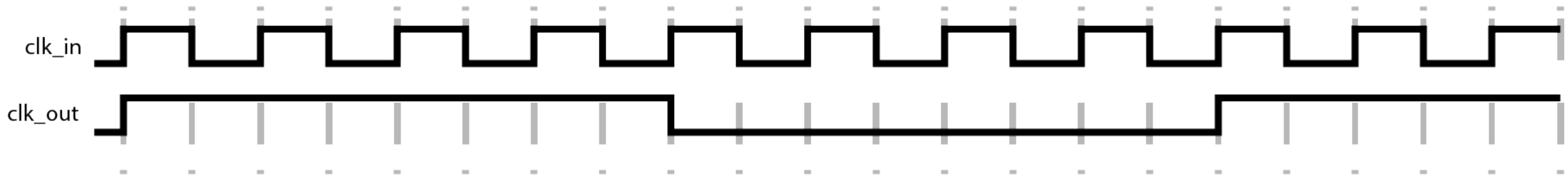




Counters and State Machines

- Divide-by-4 Clock Divider
  - Output toggles every 4 clock cycles
  - Many linear state transitions
- Do we really need a state for all the counting steps?





Berkeley
UNIVERSITY OF CALIFORNIA

Counters and State Machines

- Replace counting states with a counter
  - Counter sets flag when finished counting
- Counters and Accumulators commonly used with state machines
  - Repetitive and linear steps can be delegated to a counter/accumulator