Logic Design V

CSci 2021: Machine Architecture and Organization
Lecture #40, May 4th, 2015

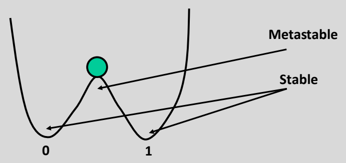
Your instructor: Stephen McCamant

Timing for Flip-Flops

- **Input must be steady around clock edge for reliable operation**
  - Setup time: amount of time before clock input must be right
  - Hold time: amount of time after clock that input must right
- **Delay before output changes**
  - Clock-to-output time: delay between clock edge and output edge
- **Important fact:**
  - (hold time) < (clock-to-output time)
  - If true of two flip-flops, it is safe to connect output of one to input of another, on the same clock

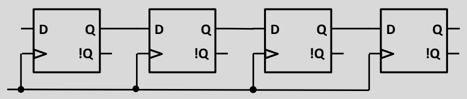
Metastability

- **Analogy:**



- **Flip-flop tries to return to a stable state**
  - Self-correcting feedback
- **But, if close to metastable point:**
  - Might take a long time to "decide"
  - Must avoid this situation for a reliable circuit

Shift Register



- **Flip-flops connected in series**
- **Behavior:**
  - Sequence of bits each move one stage per clock cycle
- **Variations:**
  - Serial or parallel input
  - Series or parallel output
  - Shift only one some cycles

Counters

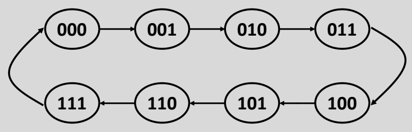
- **Simple kind of time-varying digital system**
  - Produces a single sequence of states, repeating
  - Changes every cycle or on a count pulse
- **Example: 3-bit binary up-counter**
  - Produces 000, 001, 010, 011, 100, 101, 110, 111, 000, 001, …
- **Variations:**
  - Down-counters
  - Decade counters (for decimal): 0 through 9
  - Gray code: sequence where only one bit changes at a time
  - Ring counter: circular shift register producing one-hot outputs

State Machines

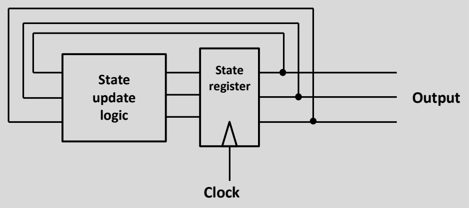
- **State machine perspective:**
  - If a device has limited memory, we can enumerate its possible *states*
  - *n* bits of memory -> at most 2^n states
  - Inputs cause state transitions
  - Outputs depend on state and possibly inputs
- **General approach for designing sequential circuits**
  - Enumerate states
  - Determine state transitions and I/O
    - Last two summarized in *state transition diagram*
  - Use flip-flops to remember state
  - Use combinational logic to implement update and output functions

Counters as State Machines

- **Simplified special case**
  - Output is generally identical to state
  - No input other than a clock
  - State transition diagram is a single cycle
- **Three-bit up counter:**



Counter Block Diagram



- **Register: parallel D flip-flops**
- **State update is a combinational circuit**
  - For 3-bit up counter, increment mod 8

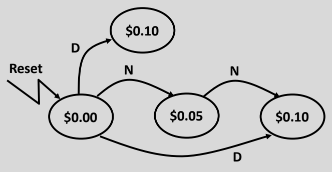
Administrative Notes

- **Final exam: Thursday, May 14th**
  - Location: 133 Tate Physics, same as regular lectures
  - Time: 8:00am-10:00am, same rules as quizzes
  - Topic coverage: comprehensive, but extra weight on post-quiz-2 material
- **Assignment V due Friday in lecture**
- **Wednesday and Friday lecture topics: review**
  - Wednesday: outline of course topics, course evaluations
    - Bring pen or pencil
  - Friday: practice problems from VM through logic design
- **Prof. Zhai will substitute for me on Friday and the final**
  - Still have office hours next week

Running Example: Vending Machine

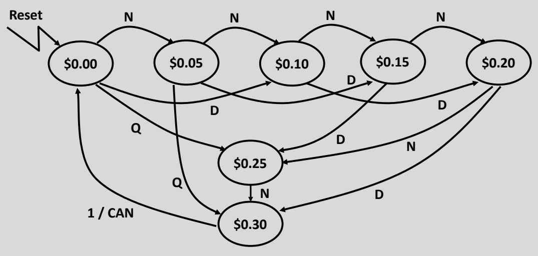
- **Sell some 30-cent product**
  - 3 oz cans of soda, etc.
- **Inputs: coin detectors**
  - Nickels (5 cents), dimes (10 cents), quarters (25 cents)
- **Output: can dispenser**
- **State: amount of money received so far**

State Diagram: Starting



- **Think about all possible sequences of inputs**
- **States are the same if future behavior is the same**
  - E.g., Two nickels equivalent to one dime

State Diagram: Up To $0.30



- **Check: are there N, D, and Q edges out of every state?**

Vending Machine Design

- What to do about overpayment?
  1. Just keep the money
  2. Apply to the next transaction
  3. Reject a single coin
  4. Abort the whole transaction
  5. Make change

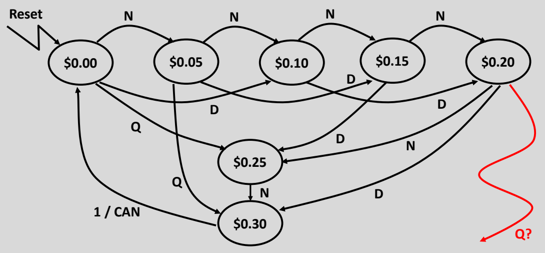
1–2: Can do with current I/O

3–4: Require new actions

5: Many complications

13

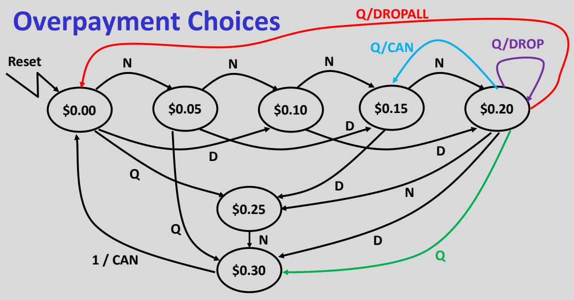
Overpayment State Diagram



- What can we do with this edge?

14

Overpayment Choices



1. Just keep the money
2. Apply to the next transaction
3. Reject a single coin
4. Abort the whole transaction

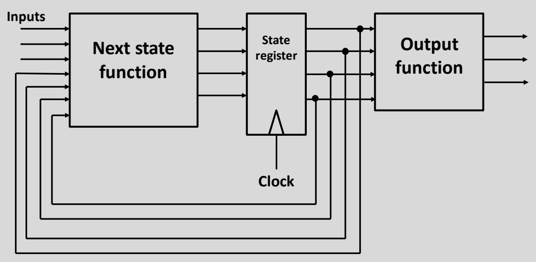
15

Two Ways to Handle Output

- Moore machine
  - Output depends only on the current state
  - Output changes only with the clock
  - Output written in the state circle
  - Requires more states
- Mealy machine
  - Output based on state and inputs
  - Output can change asynchronously
  - Output written with transition, after a slash
  - Requires fewer states

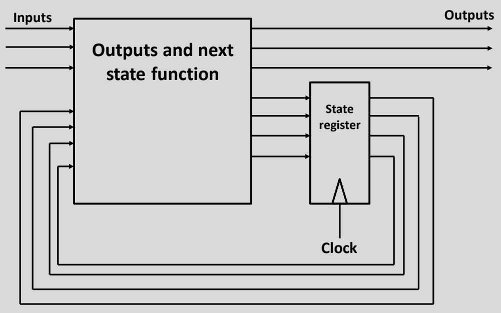
16

Moore Machine Block Diagram



17

Mealy Machine Block Diagram



18

State Diagram Conventions

- **Indicate the starting/reset state**
  - With an arrow coming from nowhere
- **Input signals on edges can be separated with commas**
  - D, Q: dime or quarter
- **Or, input signals can be combinational functions**
  - D | Q: same
- **Edges leaving a state must be mutually exclusive**
  - "Deterministic" state machine
- **Often omitted:**
  - Transitions that stay in the same state
    - Especially if all inputs inactive
  - Inactive outputs

Choosing a State Encoding

- **Unique pattern of state bits for each state**
  1. One-hot
     - Simplifies logic if not too many states
     - Scales poorly with many states
  2. Binary encoding
     - Fewer wires and flip-flops
     - Logic may be more complicated
- **Careful choice of encoding makes logic simpler**
  - Use similar bit patterns for similar states
  - There are automated CAD tools for this too

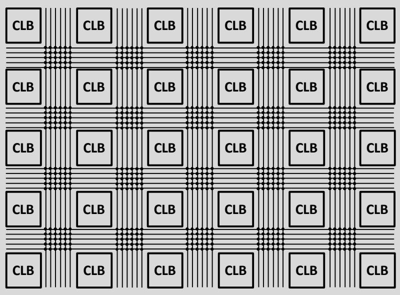
State Machine Logic

- **Given encoding, create truth tables**
- **Then, reduces to combinational design**
  - Karnaugh maps, etc.
- **Optionally, could use other kinds of flip-flops**
  - Sometimes leads to modest simplification

Some Implementation Technologies

- **Breadboard and SSI ICs**
  - E.g., "hex inverter" chip
  - For production, design a printed circuit board
  - 70s technology, good for hobby projects and intro courses
- **Application-Specific Integrated Circuit (ASIC)**
  - I.e., rent a microchip factory, like Intel's
  - Cost effective only in large quantities
- **Reconfigurable logic**
  - "Field-Programmable Gate Array" (FPGA)
  - General-purpose hardware can be electronically configured into a chosen circuit
  - Blurs the line between software and hardware

FPGA Overall Structure



Configurable Logic Block (CLB)

- **A small unit of reconfigurable hardware**
- **Might contain, for instance:**
  - An arbitrary 5-input Boolean function implemented as a lookup table
  - A full adder
  - A D flip-flop
- **The lookup table is just an SRAM**
  - So it's programmable
  - But in operation, it typically stays fixed after bootup

Interconnect

- **Have to wire these blocks into a circuit**
- **CLBs are connected with configurable channels**
  - I.e., wires with transistor-based switches at intersections
- **Statically configured**
  - A certain wire is always used for the same connection
  - Switches increase delay compared to a plain wire
- **Length tradeoffs**
  - Adjacent blocks have fastest connections
  - Many switches → optimized for short paths
  - Smaller number of long wires
  - Specialized lines for clock distribution

Programming and Using FPGAs

- **Automatic computerized layout standard**
  - Proprietary tools optimized by FPGA manufacturers
- **Program might be:**
  - Loaded at power-on
  - Stored in on-FPGA flash memory
- **What are FPGAs good for?**
  - Prototyping complex hardware descriptions
  - Low- to medium-volume specialized devices
  - Accelerating certain very-parallel computations