## Lecture 24

- Logistics
  - HW7 back today
  - Midterm2 back today (Average 80/100)
    - Solution on-line this PM
  - HW8 due Wednesday
  - Ant extra credit due Friday
  - Final exam, Wednesday March 18, 2:30-4:20 pm here
    - Review session Monday, March 16, 4:30 pm, Place EEB 037
- Last lecture
  - State encoding
    - One-hot encoding
    - Output encoding
  - State partitioning
- Today
  - Encoding & Partitioning examples



## State-encoding strategies

- No guarantee of optimality
  - An intractable problem
- Most common strategies
  - Binary (sequential) – number states as in the state table
  - Random – computer tries random encodings
  - Heuristic – rules of thumb that seem to work well
    - e.g. Gray-code – try to give adjacent states (states with an arc between them) codes that differ in only one bit position
  - One-hot – use as many state bits as there are states
  - Output – use outputs to help encode states
  - Hybrid – mix of a few different ones (e.g. One-hot + heuristic)



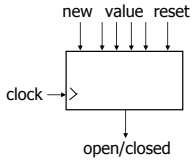
## One-hot encoding

- One-hot: Encode n states using n flip-flops
  - Assign a single "1" for each state
    - Example: 0001, 0010, 0100, 1000
  - Propagate a single "1" from one flip-flop to the next
    - All other flip-flop outputs are "0"
- The inverse: One-cold encoding
  - Assign a single "0" for each state
    - Example: 1110, 1101, 1011, 0111
  - Propagate a single "0" from one flip-flop to the next
    - All other flip-flop outputs are "1"
- "almost one-hot" encoding (modified one-hot encoding)
  - Use no-hot (000…0) for the initial (reset state)
  - Assumes you never revisit the reset state till reset again.



## Another Encoding Example: Digital combination lock

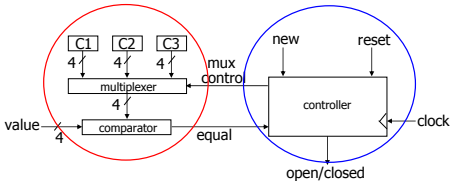
- An output-encoded FSM
  - Punch in 3 values in sequence and the door opens
  - If there is an error the lock must be reset
  - After the door opens the lock must be reset
  - Inputs: sequence of number values, reset
  - Outputs: door open/close





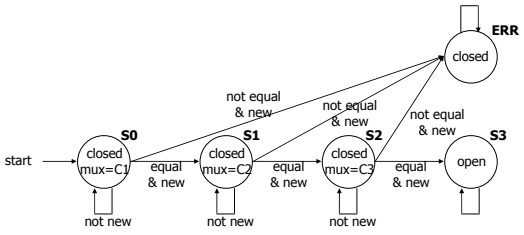

## Separate data path and control

- Design datapath first
  - After the state diagram
  - Before the state encoding
- Control has 2 outputs
  - Mux control to datapath
  - Lock open/closed

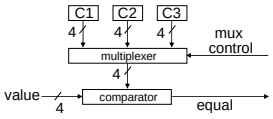


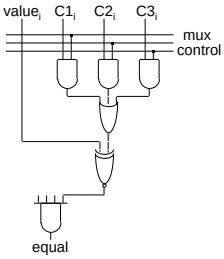



## Draw the state diagram



## Design the datapath




- Choose simple control
  - 3-wire mux for datapath
    - Control is 001, 010, 100
  - Open/closed bit for lock state
    - Control is 0/1



## Output encode the FSM

- FSM outputs
  - Mux control is 100, 010, 001
  - Lock control is 0/1
- State are: S0, S1, S2, S3, or ERR
  - Can use 3, 4, or 5 bits to encode
  - Have 4 outputs, so choose 4 bits
    - Encode mux control and lock control in state bits
    - Lock control is first bit, mux control is last 3 bits

S0 = 0001 (lock closed, mux first code)
S1 = 0010 (lock closed, mux second code)
S2 = 0100 (lock closed, mux third code)
S3 = 1000 (lock open)
ERR = 0000 (error, lock closed)



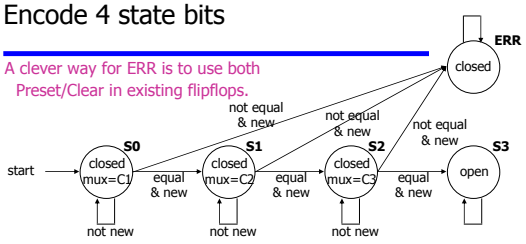
## FSM has 4 state bits and 2 inputs...

- Output encoded!
  - Outputs and state bits are the same
- How do we minimize the logic?
  - FSM has 4 state bits and 2 inputs (equal, new)
  - 6-variable K-map for all five states?
    - Way too complicated
- Notice the state assignment is close to one-hot
  - ERR state (0000) is only deviation
  - Is there a clever design we can use?



## Encode 4 state bits

A clever way for ERR is to use both Preset/Clear in existing flipflops.




$S_0^+ = S_0N'$
$S_1^+ = S_0EN + S_1N'$
$S_2^+ = S_1EN + S_2N'$
$S_3^+ = S_2EN + S_3$

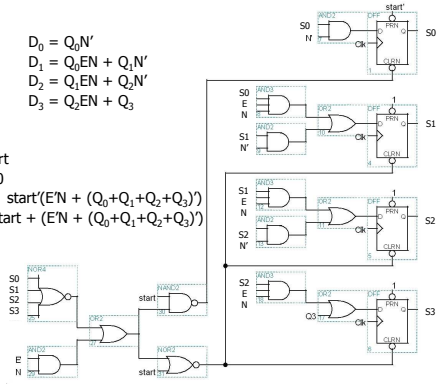
Not equal & new / Already in ERR

$Preset_0$ = start
$Preset_{1,2,3} = 0$
$Reset_0$ = start'(E'N + ($Q_0$+$Q_1$+$Q_2$+$Q_3$)')
$Reset_{1,2,3}$ = start + (E'N + ($Q_0$+$Q_1$+$Q_2$+$Q_3$)')



$D_0 = Q_0N'$
$D_1 = Q_0EN + Q_1N'$
$D_2 = Q_1EN + Q_2N'$
$D_3 = Q_2EN + Q_3$

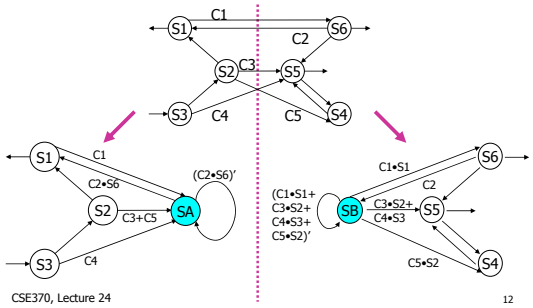
$Preset_0$ = start
$Preset_{1,2,3} = 0$
$Reset_0$ = start'(E'N + ($Q_0$+$Q_1$+$Q_2$+$Q_3$)')
$Reset_{1,2,3}$ = start + (E'N + ($Q_0$+$Q_1$+$Q_2$+$Q_3$)')





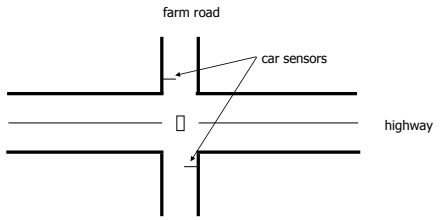

## State Partitioning

- Add idles states to handoff control between machines



## Example: Traffic light controller

- Highway/farm road intersection





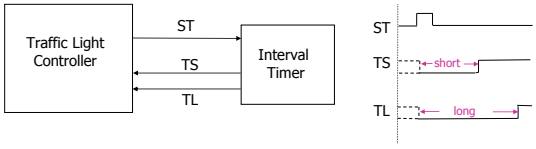

## Example: traffic light controller

- A busy highway is intersected by a little used farm road
- Detectors C sense the presence of cars waiting on the farm road
  - with no car on farm road, lights remain Green in highway direction
  - if vehicle on farm road, highway lights go from Green to Yellow to Red, allowing the farm road lights to become Green
  - these stay Green only as long as a farm road car is detected but never longer than a set interval
  - when these are met, farm lights transition from Green to Yellow to Red, allowing highway to return to Green
  - even if farm road vehicles are waiting, highway gets at least a set interval as Green



## Example: traffic light controller

- Assume you have an interval timer that in response to a set (ST) signal generates both:
  - a short time pulse (TS) and
  - a long time pulse (TL)
- TS is to be used for timing yellow lights and TL for green lights





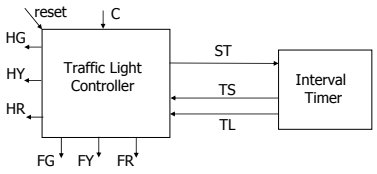

## Example: traffic light controller

- **Inputs** — **Description**
  - reset — place FSM in initial state
  - C — detect vehicle on the farm road
  - TS — short time interval expired
  - TL — long time interval expired
- **Outputs** — **Description**
  - HG, HY, HR — assert green/yellow/red highway lights
  - FG, FY, FR — assert green/yellow/red farm road lights
  - ST — start timing a short or long interval
- States – some light configurations imply others

| State | Description |
|---|---|
| HG | highway green (farm road red) |
| HY | highway yellow (farm road red) |
| FG | farm road green (highway red) |
| FY | farm road yellow (highway red) |



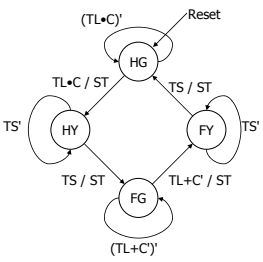
## Example: traffic light controller





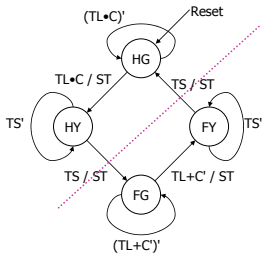

## Example: traffic light controller

- State diagram



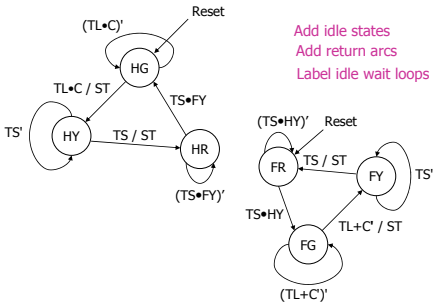

Outputs not shown:
FR=HG+HY
HR=FG+FY

## Example: State Partitioning






## State partitioning for traffic light controller






## Minimize communication between partitions

- Ideal world: Two machines handoff control
  - Separate I/O, states, etc.
- Real world: Minimize handoffs and common I/O
  - Minimize number of state bits that cross boundary
  - Merge common outputs
- Look for:
  - Disjoint inputs used in different regions of state diagram
  - Outputs active in only one region of state diagram
  - Isomorphic portions of state diagram
    - ⇨ Add states, if necessary, to make them so
  - Regions of diagram with a single entry and single exit point



## FSM design: A multi-step process

1. Understand the problem
   - State diagram and state-transition table
2. Determine the machine's states
   - Consider missing transitions: Will the machine start?
   - Minimize the state diagram: Reuse states where possible
3. Encode the states
   - Encode states, outputs with a reasonable encoding choice
   - Consider the implementation target
4. Design the next-state logic
   - Minimize the combinational logic
   - Choices made in steps 2 & 3 affect the logic complexity
5. Implement the FSM