Lecture Summary – Module 3
Sequential Logic Circuits

Learning Outcome: an ability to analyze and design sequential logic circuits

Learning Objectives:
3-1. describe the difference between a combinational logic circuit and a sequential logic circuit
3-2. describe the difference between a feedback sequential circuit and a clocked synchronous state machine
3-3. define the state of a sequential circuit
3-4. define active high and active low as it pertains to clocking signals
3-5. define clock frequency and duty cycle
3-6. describe the operation of a bi-stable and analyze its behavior
3-7. define metastability and illustrate how the existence of a metastable equilibrium point can lead to a random next state
3-8. write present state – next state (PS-NS) equations that describes the behavior of a sequential circuit
3-9. draw a state transition diagram that depicts the behavior of a sequential circuit
3-10. construct a timing chart that depicts the behavior of a sequential circuit
3-11. draw a circuit for a set-reset (“S-R”) latch and analyze its behavior
3-12. discuss what is meant by “transparent” (or “data following”) in reference to the response of a latch
3-13. draw a circuit for an edge-triggered data (“D”) flip-flop and analyze its behavior
3-14. compare the response of a latch and a flip-flop to the same set of stimuli
3-15. define setup and hold time and determine their nominal values from a timing chart
3-16. determine the frequency and duty cycle of a clocking signal
3-17. identify latch and flip-flop propagation delay paths and determine their values from a timing chart
3-18. describe the operation of a toggle (“T”) flip-flop and analyze its behavior
3-19. derive a characteristic equation for any type of latch or flip-flop
3-20. identify the key elements of a clocked synchronous state machine: next state logic, state memory (flip-flops), and output logic
3-21. differentiate between Mealy and Moore model state machines, and draw a block diagram of each
3-22. analyze a clocked synchronous state machine realized as either a Mealy or Moore model
3-23. outline the steps required for state machine synthesis
3-24. derive an excitation table for any type of flip-flop
3-25. discuss reasons why formal state-minimization procedures are seldom used by experienced digital designers
3-26. describe three ways that state machines can be specified in ABEL: using a clocked truth table, using clocked assignment operators, or using a state diagram approach
3-27. list the ABEL attribute suffixes that pertain to sequential circuits
3-28. draw a circuit for an oscillator and calculate its frequency of operation
3-29. draw a circuit for a bounce-free switch based on an S-R latch and analyze its behavior
3-30. design a clocked synchronous state machine and verify its operation
3-31. define minimum risk and minimum cost state machine design strategies, and discuss the tradeoffs between the two approaches
3-32. compare state assignment strategy and state machine model choice (Mealy vs. Moore) with respect to PLD resources (P-terms and macrocells) required for realization
3-33. compare and contrast the operation of binary and shift register counters
3-34. derive the next state equations for binary “up” and “down” counters
3-35. describe the feedback necessary to make ring and Johnson counters self-correcting
3-36. compare and contrast state decoding for binary and shift register counters
3-37. describe why “glitches” occur in some state decoding strategies and discuss how to eliminate them
3-38. identify states utilized by a sequence recognizer: accepting sequence, final, and trap
3-39. determine the embedded binary sequence detected by a sequence recognizer
Lecture Summary – Module 3-A
Bistable Elements


- overview
  - combinational vs. sequential circuits
  - state of sequential circuit
  - finite state machine
  - clock signal
    - assertion level
    - period / frequency
    - duty cycle
  - types of sequential circuits
    - feedback
    - clocked synchronous

- bistable elements
  - “simplest” sequential circuit
  - no inputs (no way of controlling/changing state)
  - randomly powers up into one state or the other
  - digital analysis: two stable states
  - single state variable (Q)
  - analog analysis: additional quasi-stable state (metastable)

Transfer functions ("inverter"):

\[ V_{out1} = T(V_{in1}) \]
\[ V_{out2} = T(V_{in2}) \]

Equilibrium points:

\[ V_{in1} = V_{out2} \]
\[ V_{in2} = V_{out1} \]

Random noise drives circuit to stable operating point

- metastable behavior
  - comparable to dropping ball onto smooth hill
  - speed with which ball rolls to one side or the other depends on location it “hits”
  - important: if “simplest” sequential circuit is susceptible to metastable behavior, then clearly ALL sequential circuits are(!)
Lecture Summary – Module 3-B
The Set-Reset (S-R) Latch


- latches and flip-flops
  - flip-flop changes state based on *clocking signal*
  - latch changes its output any time it is *enabled*
- set-reset (S-R) latch
  - change bistable into latch by “adding an input” to each inverter (NOR gate)
  - two inputs
    - asserting S “sets” the latch state (Q output) to 1
    - asserting R “resets” the latch state to 0
    - if both S and R are negated, circuit behaves like bistable (retains its state)
    - if both S and R are asserted and then negated simultaneously, random next state
- exercise: construct a timing chart for the NOR-implemented S-R latch
  - assume each gate has delay τ
  - write the next state equations for Q and QN

\[
Q(t+\tau) = \\
QN(t+\tau) =
\]

- create a present state – next state (PS-NS) table and state transition diagram (STD)

<table>
<thead>
<tr>
<th>Present State</th>
<th>Present Input</th>
<th>Next State</th>
<th>Q(t+\tau)</th>
<th>QN(T+\tau)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q(t)</td>
<td>QN(t)</td>
<td>S(t)</td>
<td>R(t)</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
• exercise, continued…
  o construct a timing chart based on the initial conditions and given inputs

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>QN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

• exercise: investigate response to the “1-1” input combination

• exercise: investigate response to a “glitch”

• propagation delay – time for an output to respond to an input transition
  o need to specify “path”
  o example: $t_{\text{pLH}(S\rightarrow Q)}$ is the rise propagation delay of the Q output in response to assertion of the S input
  o note that rise and fall propagation delays are typically different

• minimum pulse width requirement (see “glitch” timing chart)
- Variations
  - NAND-implemented $S'$-$R'$ latch
  - NAND-implemented S-R latch with ENABLE (“C”)

- Transparent D ("data") latch
  - Just an S-R latch with an inverter between the S and R inputs
  - Basic "memory bit"
  - Called "transparent" (or "data following") because that is what it is (does) when "open"
  - Retains value when enable is negated (latch "closed"
  - Propagation delay parameters
  - Setup and hold times (what happens if either is violated)
Lecture Summary – Module 3-C
Data (D) and Toggle (T) Flip-Flops


- edge-triggered D flip-flop
  - changes state ("triggers") on clock edge
  - can be positive (rising) edge triggered or negative (falling) edge triggered
  - created using two latches cascaded together, that open on opposite clock phases
    - input latch "master"
    - output latch ("slave")
  - triangle = dynamic input indicator (clock)
  - characteristic equation: \( Q^* = D \)
  - propagation delay parameters
    - setup and hold times

- negative edge-triggered D flip-flop

- edge-triggered D flip-flop with enable
- edge-triggered T ("toggle") flip-flop
  - toggles state \((Q^* = Q')\) if T input is 1
  - stays in same state \((Q^* = Q)\) if T input is 0
  - characteristic equation: \(Q^* = T \oplus Q\) (can synthesize using D flip-flop as "building block")

- flip-flop timing parameters
  - clock pulse width
  - clock period
  - clock duty cycle
  - nominal setup time
  - nominal hold time
  - \(t_{PLH(C \rightarrow Q)} = t_{PHL(C \rightarrow Q_L)}\)
  - \(t_{PHL(C \rightarrow Q)} = t_{PHL(C \rightarrow Q_L)}\)

- response of latch vs. flip-flop
Lecture Summary – Module 3-D
Clocked Synchronous State Machine Structure and Analysis


- **introduction**
  - state machine (sequential circuit)
  - clocked
  - synchronous (all flip flops share common clocking signal)

- **state machine basic blocks**
  - next state ("excitation") logic
  - state memory (flip flops)
  - output logic

- **state machine models**
  - Moore
  - Mealy

- can map a given state machine into either model
- important: *how model chosen satisfies the design requirements*

- **state machine analysis**
  - determine next state and output functions
  - construct a present state – next state / output table
  - draw state transition diagram
- draw a timing diagram
- example: Mealy machine analysis

<table>
<thead>
<tr>
<th>PS</th>
<th>PI</th>
<th>NS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Q0</td>
<td>EN</td>
<td>Q1* Q0*</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- example: Moore machine analysis

<table>
<thead>
<tr>
<th>PS</th>
<th>PI</th>
<th>NS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>Q0</td>
<td>EN</td>
<td>Q1* Q0*</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Lecture Summary – Module 3-E
Clocked Synchronous State Machine Synthesis


- introduction – the creative process
  - potentially imprecise description
  - choose among different ways of doing things
  - handle special cases
  - keep track of several ideas in your head
  - *not an algorithm*
  - circuit will perform exactly as designed
  - no guarantee it will work the first time

- state machine design steps
  - construct PS-NS/O table and/or STD
  - minimize “obvious” redundant states
  - assign state variable combinations
  - update PS-NS/O table and/or STD accordingly
  - (choose flip-flop type) – we will use D-type for most designs
  - (excitation table/equations – not needed for D-type flip flops = why?)
  - derive output equations
  - draw logic diagram or realize equations directly in a PLD (using edge-triggered D-type)

- derivation of excitation table for an S-R latch

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>Q*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>d</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q</th>
<th>Q*</th>
<th>S</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>d</td>
<td>0</td>
</tr>
</tbody>
</table>

- derivation of excitation table for a T flip flop

<table>
<thead>
<tr>
<th>T</th>
<th>Q</th>
<th>Q*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Q</th>
<th>Q*</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
- three basic ways to specify state machines in ABEL
  - “clocked” truth table, using $>$ operator
  - as next state equations, using := operator
  - as a state diagram, using GOTO and/or IF-THEN-ELSE clauses to specify the state transitions
- attribute suffixes allowed on right-hand side of an equation
  - internal flip flop output .Q
  - internal feedback path .FB
  - external signal at pin .PIN
- equations that can be written for macrocell functions
  - flip flop input .D
  - flip flop clock input .CLK
  - output pin tri-state buffer enable .OE
  - flip flop asynchronous (pre)set .AP
  - flip flop asynchronous reset .AR
- differences in macrocell architecture

GAL22V10 Output Logic Macrocell ("OLMC")

GAL22V10 Output Logic Macrocell ("OLMC")

All OLMC edge-triggered D flip-flops utilize common clock (CLK), asynchronous reset (AR), and asynchronous preset (SP) signals

ispMACH 4000ZE Macrocell

ispMACH 4000ZE I/O Cell
- periodic clock generation circuits
  - typically based on crystal or R-C time constant
  - issues of interest
    - frequency
    - duty cycle
    - transition time (slew rate)
    - ringing (undershoot / overshoot)
    - stability (drift / jitter)
    - driving capability
    - skew (based on different physical path lengths)

- CMOS “ring” oscillator and crystal oscillator circuits

- ispMach 4000ZE internal oscillator setup/use
  \[
  f = \left(2C(0.4R_{eq} + 0.7R_2)\right)^{1} \text{ where } R_{eq} = (R_1R_2)/(R_1+R_2)
  \]

```verbatim
MODULE OscTest
TITLE 'ispMACH 4256ZE Oscillator Setup'
LIBRARY 'lattice';

DECLARATIONS
  " Use maximum possible internal divisor -> yields approx 4 Hz output frequency
  XLAT_OSCTIMER(DYNOSCDIS, TIMERRES, OSCOUT, TIMEROUT, 1048576);
  timdiv node istype 'reg_d,buffer';
  osc_dis, osc_rst, osc_out, tmr_out node istype 'com';

EQUATIONS
  osc_dis=0;
  osc_rst=0;
  I1 OSCTIMER(osc_dis, osc_rst, osc_out, tmr_out);
  " Divide tmr_out frequency by 2 to get approx 2 Hz clocking freq at node timdiv
  timdiv.clk = tmr_out;
  timdiv := !timdiv;
END
```

- timing diagrams and specifications

\[
\text{clock frequency } (f) = 1/\tau_{\text{clk}} \quad \text{duty cycle } = t_H/(t_H+t_L)
\]
- event clock generation circuits
  - examples of events
    - pushing button
    - sensor firing
  - problem: contact bounce

solution: “bounce-free” (or “bounce-less”) switch implemented using a S.P.D.T. (single pole, double throw pushbutton and an S’R’ latch

![Pushbutton Pressed (S-R latch set)](image)

![Pushbutton Bouncing (S-R latch stays in same state)](image)

---

**MODULE**  
**bf_switch**

**TITLE** 'Bounce-free Switch in ABEL'

**DECLARATIONS**

" Inputs are active low
!NO pin; " normally open switch contact
!NC pin; " normally closed switch contact
" Bounce-free clock output
BFC pin istype 'reg'; " can be a node instead of a pin

**EQUATIONS**

BFC.D = 0;
BFC.CLK = 0;
BFC.AP = NO;
BFC.AR = NC;

END

Here, we are essentially using the D flip-flop as an S-R latch via its asynchronous preset (.AP) and asynchronous reset (.AR) inputs
Lecture Summary – Module 3-F
State Machine Design Examples: Sequence Generators


- a sequence generator state machine produces a (periodic) series of output signal assertions that constitute a pre-defined pattern
- two different design strategies
  - minimum cost (don’t cares in next states are allowed)
  - minimum risk (unused states explicitly assigned a next state)
- character sequence display – displays AbC or CbS on a 7-segment display (Moore model)

```
MODULE tv_disp
TITLE 'Character Sequence Display'
DECLARATIONS
CLOCK pin;
M pin;    " mode control
Q..Q0 pin istype 'reg';
" 7-segment display outputs (common anode, active low)
!LA,!LB,!LC,!LD,!LE,!LF,!LG pin istype 'com';

TRUTH_TABLE ([Q1, Q0] -> [LA, LB, LC, LD, LE, LF, LG])
[ 0, 0 ] -> [ 1, 1, 1, 0, 1, 1, 1];  " A
[ 0, 1 ] -> [ 0, 0, 1, 1, 1, 1, 1];  " b
[ 1, 0 ] -> [ 1, 0, 0, 1, 1, 1, 0];  " C
[ 1, 1 ] -> [ 1, 0, 1, 1, 0, 1, 1];  " S

TRUTH_TABLE ([Q1, Q0, M] :> [Q1, Q0])
[ 0, 0, 0 ] :> [ 0, 1];
[ 0, 0, 1 ] :> [ 1, 0];
[ 0, 1, 0 ] :> [ 1, 0];
[ 0, 1, 1 ] :> [ 1, 1];
[ 1, 0, 0 ] :> [ 0, 0];
[ 1, 0, 1 ] :> [ 0, 1];
[ 1, 1, 0 ] :> [ 0, 0];
[ 1, 1, 1 ] :> [ 1, 0];

EQUATIONS
[Q1..Q0].CLK = CLOCK;
END
```
- 4-mode light sequencer – Moore model

<table>
<thead>
<tr>
<th>PS</th>
<th>Q2</th>
<th>Q1</th>
<th>Q0</th>
<th>PI</th>
<th>NS</th>
<th>Q2*</th>
<th>Q1*</th>
<th>Q0*</th>
<th>PO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>1 0</td>
<td>0 0</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0</td>
<td>0 1</td>
<td>0 1</td>
<td>1 0</td>
<td>0 1</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>0 0</td>
<td>1 0</td>
<td>0 0</td>
<td>1 1</td>
<td>0 1</td>
<td>0 1</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 0</td>
<td>1 1</td>
<td>0 0</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>0 1</td>
<td>0 1</td>
<td>0 1</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>0 1</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 1</td>
<td>0 1</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>0 1</td>
<td>1 1</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 1</td>
<td>1 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 1</td>
<td>1 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td>0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This realization uses 6 macrocells
- check alternate state/output assignments (where output functions are the state variables)

Both realizations (clocked operator table and state diagram) use 3 macrocells
• Mealy model

```
MODULE mealylsa
TITLE 'Light Sequencer - Mealy Model A'
DECLARATIONS
  CLOCK pin;
  M0, M1 pin;
  Q1, Q0 pin istype 'reg';
  L2, L1, L0 pin istype 'com';
truth_table ([Q1,Q0,M1,M0]->[Q1,Q0])
  [ 0, 0, 0, 0] -> [ 0, 1];
  [ 0, 0, 0, 1] -> [ 1, 1];
  [ 0, 0, 1, 0] -> [ 0, 1];
  [ 0, 0, 1, 1] -> [ 0, 1];
  [ 0, 1, 0, 0] -> [ 1, 0];
  [ 0, 1, 0, 1] -> [ 0, 0];
  [ 0, 1, 1, 0] -> [ 0, 0];
  [ 0, 1, 1, 1] -> [ 1, 0];
  [ 1, 0, 0, 0] -> [ 1, 0];
  [ 1, 0, 0, 1] -> [ 1, 1];
  [ 1, 0, 1, 0] -> [ 0, 0];
  [ 1, 0, 1, 1] -> [ 0, 0];
  [ 1, 1, 0, 0] -> [ 0, 0];
  [ 1, 1, 0, 1] -> [ 0, 1];
  [ 1, 1, 1, 0] -> [ 0, 0];
  [ 1, 1, 1, 1] -> [ 1, 0];
EQUATIONS
  [Q1..Q0].CLK = CLOCK;
END
```

```
truth_table ([Q1,Q0,M1,M0]->[L2,L1,L0])
  [ 0, 0, 0, 0] -> [ 0, 0, 0];
  [ 0, 0, 0, 1] -> [ 0, 0, 0];
  [ 0, 0, 1, 0] -> [ 0, 0, 0];
  [ 0, 0, 1, 1] -> [ 0, 0, 0];
  [ 0, 1, 0, 0] -> [ 1, 0, 0];
  [ 0, 1, 0, 1] -> [ 0, 0, 0];
  [ 0, 1, 1, 0] -> [ 0, 0, 0];
  [ 0, 1, 1, 1] -> [ 1, 0, 0];
  [ 1, 0, 0, 0] -> [ 0, 1, 0];
  [ 1, 0, 0, 1] -> [ 0, 1, 0];
  [ 1, 0, 1, 0] -> [ 1, 1, 0];
  [ 1, 0, 1, 1] -> [ 0, 1, 1];
  [ 1, 1, 0, 0] -> [ 0, 0, 1];
  [ 1, 1, 0, 1] -> [ 0, 0, 1];
  [ 1, 1, 1, 0] -> [ 0, 0, 1];
  [ 1, 1, 1, 1] -> [ 1, 1, 1];
EQUATIONS
  [Q1..Q0].CLK = CLOCK;
END
```

This realization uses 5 macrocells
• check alternate Mealy state/output assignments

<table>
<thead>
<tr>
<th>truth_table</th>
<th>([Q1,Q0,M1,M0]-&gt;[Q1,Q0])</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ 0, 0, 0, 0]--&gt;[ 0, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 0, 0, 1]--&gt;[ 0, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 0, 1, 0]--&gt;[ 0, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 0, 1, 1]--&gt;[ 0, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 0, 0]--&gt;[ 1, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 0, 1]--&gt;[ 1, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 1, 0]--&gt;[ 1, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 1, 1]--&gt;[ 1, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 0, 0, 0]--&gt;[ 0, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 0, 0, 1]--&gt;[ 0, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 0, 1, 0]--&gt;[ 0, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 0, 1, 1]--&gt;[ 0, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 0, 0]--&gt;[ 1, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 0, 1]--&gt;[ 1, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 1, 0]--&gt;[ 1, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 1, 1]--&gt;[ 1, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 0, 0, 0]--&gt;[ 0, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 0, 0, 1]--&gt;[ 0, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 0, 1, 0]--&gt;[ 0, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 0, 1, 1]--&gt;[ 0, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 0, 0]--&gt;[ 0, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 0, 1]--&gt;[ 0, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 1, 0]--&gt;[ 0, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 0, 1, 1, 1]--&gt;[ 0, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 1, 0, 0, 0]--&gt;[ 1, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 1, 0, 0, 1]--&gt;[ 1, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 1, 0, 1, 0]--&gt;[ 1, 0];</td>
<td></td>
</tr>
<tr>
<td>[ 1, 0, 1, 1]--&gt;[ 1, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 1, 1, 0, 0]--&gt;[ 0, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 1, 1, 0, 1]--&gt;[ 0, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 1, 1, 1, 0]--&gt;[ 0, 1];</td>
<td></td>
</tr>
<tr>
<td>[ 1, 1, 1, 1]--&gt;[ 1, 1];</td>
<td></td>
</tr>
</tbody>
</table>

Both realizations (clocked operator table and state diagram) use 5 macrocells

• conclusions
  o choosing the “right” state variable assignment and machine model can make a significant difference in the PLD resources consumed and the amount of work required
  o the only formal way to find the “best” assignment is to try all of the assignments
  o experience is needed to do this well (see text for guidelines)
  o there is no substitute for practice (developing “applied intuition”)
Lecture Summary – Module 3-G
State Machine Design Examples: Counters and Shift Registers


- the term counter is used for any clocked sequential circuit whose state diagram contains a single cycle
  - the modulus of a counter is the number of states in the cycle – a counter with \( M \) states is called a modulo-\( M \) counter (or sometimes a divide-by-\( M \) counter)
  - a synchronous counter connects all of its flip-flop clock inputs to the same common CLOCK signal, so that all the flip-flop outputs change state simultaneously
  - UP counter \( K \)th bit next state: \( Q_k^* = Q_k \oplus (Q_{k-1} \cdot Q_{k-2} \cdot \ldots \cdot Q_1 \cdot Q_0) \)
  - DOWN counter \( K \)th bit next state: \( Q_k^* = Q_k \oplus (Q'_{k-1} \cdot Q'_{k-2} \cdot \ldots \cdot Q'1 \cdot Q'0) \)
  - ABEL program for 8-bit UP/DOWN counter

```
MODULE cnt8ud
TITLE '8-bit Binary UP/DOWN Counter'
DECLARATIONS
Q0..Q7 pin istype 'reg';
M, CLOCK pin;
" M=0 count DOWN, M=1 count UP
EQUATIONS
Q0 := !Q0;
Q1 := Q1 $ (M&Q0 # !M&!Q0);
Q2 := Q2 $ (M&Q1&Q0 # !M&!Q1&!Q0);
Q3 := Q3 $ (M&Q2&Q1&Q0 # !M&!Q2&!Q1&!Q0);
Q4 := Q4 $ (M&Q3&Q2&Q1&Q0 # !M&!Q3&!Q2&!Q1&!Q0);
Q5 := Q5 $ (M&Q4&Q3&Q2&Q1&Q0 # !M&!Q4&!Q3&!Q2&!Q1&!Q0);
Q6 := Q6 $ (M&Q5&Q4&Q3&Q2&Q1&Q0 # !M&!Q5&!Q4&!Q3&!Q2&!Q1&!Q0);
Q7 := Q7 $ (M&Q6&Q5&Q4&Q3&Q2&Q1&Q0 # !M&!Q6&!Q5&!Q4&!Q3&!Q2&!Q1&!Q0);
[Q0..Q7].CLK = CLOCK;
END
```

- ABEL program for 8-bit resettable UP counter

```
MODULE rcnt8u
TITLE 'Resettable 8-bit Binary UP Counter'
DECLARATIONS
Q0..Q7 pin istype 'reg';
R, CLOCK pin;
" if R=1, next state will be 00...0
EQUATIONS
Q0 := !R & !Q0;
Q1 := !R & (Q1 $ Q0);
Q2 := !R & (Q2 $ (Q1&Q0));
Q3 := !R & (Q3 $ (Q2&Q1&Q0));
Q4 := !R & (Q4 $ (Q3&Q2&Q1&Q0));
Q5 := !R & (Q5 $ (Q4&Q3&Q2&Q1&Q0));
Q6 := !R & (Q6 $ (Q5&Q4&Q3&Q2&Q1&Q0));
Q7 := !R & (Q7 $ (Q6&Q5&Q4&Q3&Q2&Q1&Q0));
[Q0..Q7].CLK = CLOCK;
END
```
a shift register whose state diagram is cyclic is called a shift-register counter (i.e., does not count “up” or “down”)
  - self-correcting ring counter

```plaintext
MODULE ring4sc

TITLE 'Self-Correcting 4-bit Ring Counter'

" Uses NOR function to make sure that the " next state after d000 is 0001

DECLARATIONS
CLOCK pin;
Q0..Q3 pin istype 'reg';

EQUATIONS
Q3 := Q2;
Q2 := Q1;
Q1 := Q0;
Q0 := !Q2#Q1#Q0; 

[Q0..Q3].CLK = CLOCK;

END
```

- self-correcting Johnson counter

```plaintext
MODULE john4sc

TITLE 'Self-Correcting 4-bit Johnson Counter'

DECLARATIONS
CLOCK pin;
Q0..Q3 pin istype 'reg';

R = !Q3&!Q0; " match 0odd0

EQUATIONS
Q3 := !R&Q2;
Q2 := !R&Q1;
Q1 := !R&Q0;

" Loads 0001 as next state when " current state is 0odd0 
Q0 := !R&!Q3 # R;

[Q0..Q3].CLK = CLOCK;

END
```
- state decoding
  - ring – none (“one hot”), glitch-free
    - 2n two-input AND or NAND gates, glitch-free
  - Johnson – 2n two-input AND or NAND gates, glitch-free
    - comparison with binary counter state decoding – not glitch-free
      - n-bit counter with 2^n states that can be decoded glitch-free: Gray-code
Lecture Summary – Module 3-H
State Machine Design Examples: Sequence Recognizers


- A sequence recognizer state machine responds to a pre-defined input pattern of signal assertions and produces corresponding output signal assertions
- Use of Moore model generally preferred
- Special states
  - Final state of accepting sequence (pattern being recognized)
  - Trap state

- Simple embedded sequence recognizer

![State Machine Diagram]

Assuming the state machine is initialized to state 00, determine the output sequence generated in response to the following input sequence: 11 0 1 0 0 0 1 0 0 0 0 1 0 0 0

Determine the embedded binary sequence recognized by this state machine: 0 1 0

- Digital combination lock
  - Fixed (“hard wired”) combination
  - Three input signals
    - X – combination data
    - R – (synchronous) relock
    - RESET – asynchronous reset (only way out of trap state)
  - Three output signals
    - LOCKED
    - UNLOCKED
    - ALARM
  - Moore model
    - (Initial) “locked” state
    - Six states to accept combo
    - “alarm” state
    - Total states needed: 8
  - Types of states
    - Accepting sequence (entering combination)
    - Final state (sequence correctly entered)
    - Trap state (error made while entering combination)
MODULE dcl
TITLE 'Digital Combination Lock'

X pin;      "combination data input"
R pin;      "relock input"
RESET pin;  "asynchronous reset"
CLOCK pin;
Q2, Q1, Q0 pin istype 'reg';
LOCKED pin istype 'com';  "LOCKED indicator"
UNLOCKED pin istype 'com';  "UNLOCKED indicator"
ALARM     pin istype 'com';  "ALARM indicator"

QALL = [Q2,Q1,Q0];
A0 = [ 0, 0, 0];
A1 = [ 0, 0, 1];
A2 = [ 0, 1, 0];
A3 = [ 0, 1, 1];
A4 = [ 1, 0, 0];
A5 = [ 1, 0, 1];
A6 = [ 1, 1, 0];
A7 = [ 1, 1, 1];

EQUATIONS
QALL.CLK = CLOCK;
QALL.AR = RESET;

LOCKED = !Q2&!Q1&!Q0;
UNLOCKED = Q2&Q1&!Q0;
ALARM = Q2&Q1&Q0;

STATE_DIAGRAM QALL

state A0: if (R==1) then A0 
            else if (R==0)&(X==0) then A7 
            else if (R==0)&(X==1) then A1;

state A1: if (R==1) then A0 
            else if (R==0)&(X==0) then A2 
            else if (R==0)&(X==1) then A7;

state A2: if (R==1) then A0 
            else if (R==0)&(X==0) then A7 
            else if (R==0)&(X==1) then A3;

state A3: if (R==1) then A0 
            else if (R==0)&(X==0) then A7 
            else if (R==0)&(X==1) then A4;

state A4: if (R==1) then A0 
            else if (R==0)&(X==0) then A7 
            else if (R==0)&(X==1) then A5;

state A5: if (R==1) then A0 
            else if (R==0)&(X==0) then A6 
            else if (R==0)&(X==1) then A7;

state A6: if (R==1) then A0;

state A7: goto A7;

END