EECS150

Section 7

Finite State Machine Case Studies

Fall 2001

General FSM Design Procedure

(1) Determine inputs and outputs

(2) Determine possible states of machine
- State minimization

(3) Encode states and outputs into a binary code
- State assignment or state encoding
- Output encoding
- Possibly input encoding (if under our control)

(4) Realize logic to implement functions for states and outputs
- Combinational logic implementation and optimization
- Choices in steps 2 and 3 have large effect on resulting logic

Finite String Pattern Recognizer

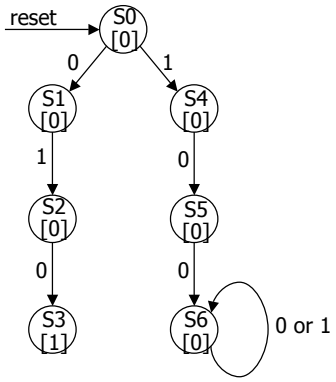
- Finite String Pattern Recognizer
  - One input (X) and one output (Z)
  - Output is asserted whenever the input sequence …010… has been observed, as long as the sequence 100 has never been seen
- Step 1: Understanding the Problem Statement
  - Sample input/output behavior:

X: 0 0 1 0 1 0 1 0 0 1 0 …
Z: 0 0 0 1 0 1 0 1 0 0 0 …

X: 1 1 0 1 1 0 1 0 0 1 0 …
Z: 0 0 0 0 0 0 0 1 0 0 0 …

Finite String Pattern Recognizer

- Step 2: Draw State Diagram
  - For the strings that must be recognized, i.e., 010 and 100
  - Moore implementation

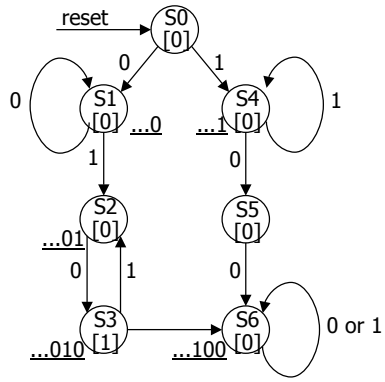


Finite String Pattern Recognizer

- Exit conditions from state S3: have recognized …010
  - If next input is 0 then have …0100 = ...100 (state S6)
  - If next input is 1 then have …0101 = …01 (state S2)

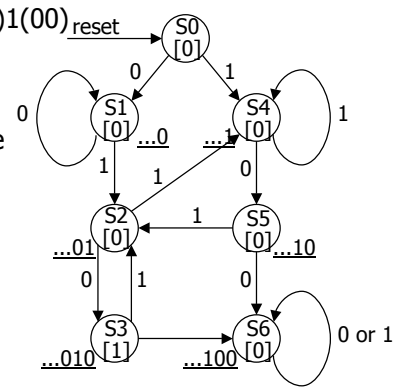
Exit conditions from S1:
recognizes strings of form …0
(no 1 seen);
loop back to S1 if input is 0

Exit conditions from S4:
recognizes strings of form …1
(no 0 seen);
loop back to S4 if input is 1



Finite String Pattern Recognizer

- S2 and S5 still have incomplete transitions
  - S2 = …01; If next input is 1, then string could be prefix of (01)1(00) S4 handles just this case
  - S5 = …10; If next input is 1, then string could be prefix of (10)1(0) S2 handles just this case
- Reuse states as much as possible
  - Look for same meaning
  - State minimization leads to smaller number of bits to represent states
- Once all states have complete set of transitions we have final state diagram



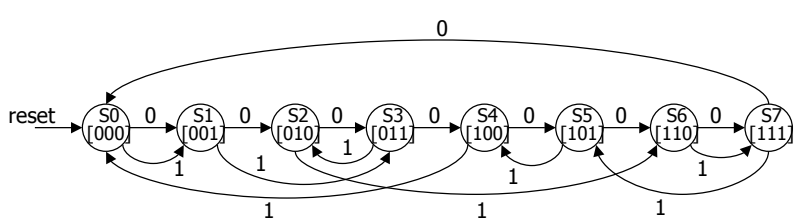
Complex Counter

- Synchronous 3-bit counter has a mode control M
  - When M = 0, the counter counts up in the binary sequence
  - When M = 1, the counter advances through the Gray code sequence
    binary: 000, 001, 010, 011, 100, 101, 110, 111
    Gray: 000, 001, 011, 010, 110, 111, 101, 100
- Valid I/O behavior (partial)

Mode Input M	Current State	Next State
0	000	001
0	001	010
1	010	110
1	110	111
1	111	101
0	101	110
0	110	111

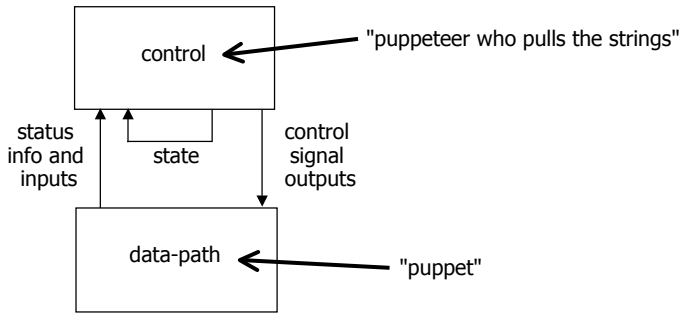
Complex Counter

- Deriving State Diagram
  - One state for each output combination
  - Add appropriate arcs for the mode control



Datapath and Control

- Digital hardware systems = data-path + control
  - Datapath: registers, counters, combinational functional units (e.g., ALU), communication (e.g., busses)
  - Control: FSM generating sequences of control signals that instructs datapath what to do next



Digital Combinational Lock

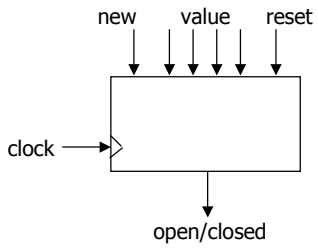
- Door Combination Lock:
  - Punch in 3 values in sequence and the door opens; if there is an error the lock must be reset; once the door opens the lock must be reset
  - Inputs: sequence of input values, reset
  - Outputs: door open/close
  - Memory: must remember combination or always have it available
  - Open questions: how do you set the internal combination?
    - Stored in registers (how loaded?)
    - Hardwired via switches set by user

Implementation in Software

```
integer combination_lock ( ) {
    integer v1, v2, v3;
    integer error = 0;
    static integer c[3] = 3, 4, 2;

    while (!new_value( ));
    v1 = read_value( );
    if (v1 != c[1]) then error = 1;

    while (!new_value( ));
    v2 = read_value( );
    if (v2 != c[2]) then error = 1;

    while (!new_value( ));
    v3 = read_value( );
    if (v2 != c[3]) then error = 1;

    if (error == 1) then return(0); else return (1);
}
```

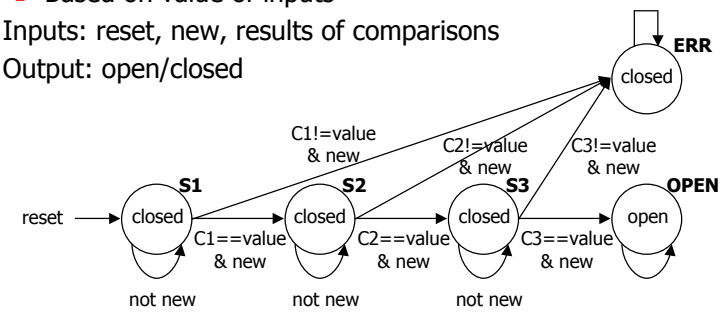

Specification

- How many bits per input value?
- How many values in sequence?
- How do we know a new input value is entered?
- What are the states and state transitions of the system?



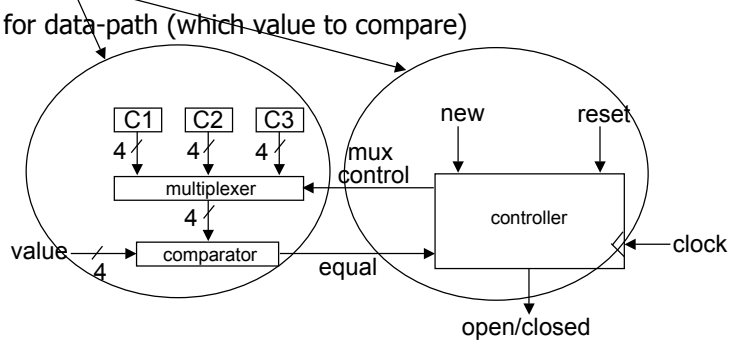
State Diagram

- States: 5 states
  - Represent point in execution of machine
  - Each state has outputs
- Transitions: 6 from state to state, 5 self transitions, 1 global
  - Changes of state occur when clock says its ok
  - Based on value of inputs
- Inputs: reset, new, results of comparisons
- Output: open/closed



Datapath and Control Structure

- Datapath
  - Storage registers for combination values
  - Multiplexer
  - Comparator
- Control
  - Finite-state machine controller
  - Control for data-path (which value to compare)



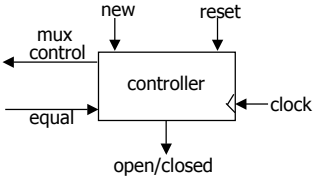
State Table for Combination Lock

- Finite-State Machine
  - Refine state diagram to take internal structure into account
  - State table ready for encoding

reset	new	equal	state	next state	mux	open/closed
1	–	–	–	S1	C1	closed
0	0	–	S1	S1	C1	closed
0	1	0	S1	ERR	–	closed
0	1	1	S1	S2	C2	closed
…						
0	1	1	S3	OPEN	–	open
…						

Encodings for Combination Lock

- Encode state table
  - State can be: S1, S2, S3, OPEN, or ERR
    - Needs at least 3 bits to encode: 000, 001, 010, 011, 100
    - And as many as 5: 00001, 00010, 00100, 01000, 10000
    - Choose 4 bits: 0001, 0010, 0100, 1000, 0000
  - Output mux can be: C1, C2, or C3
    - Needs 2 to 3 bits to encode
    - Choose 3 bits: 001, 010, 100
  - Output open/closed can be: open or closed
    - Needs 1 or 2 bits to encode
    - Choose 1 bit: 1, 0

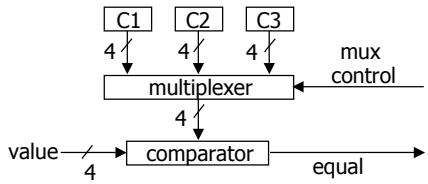


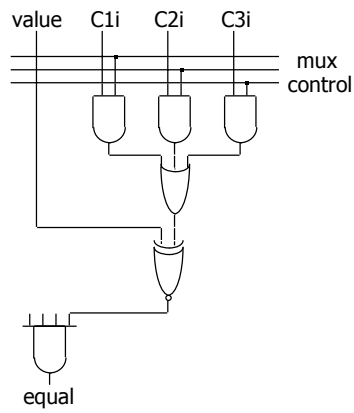
reset	new	equal	state	next state	mux	open/closed
1	–	–	–	0001	001	0
0	0	–	0001	0001	001	0
0	1	0	0001	0000	–	0
0	1	1	0001	0010	010	0
…						
0	1	1	0100	1000	–	1
…						

mux is identical to last 3 bits of state
open/closed is identical to first bit of state therefore, we do not even need to implement FFs to hold state, just use outputs

Datapath Implementation

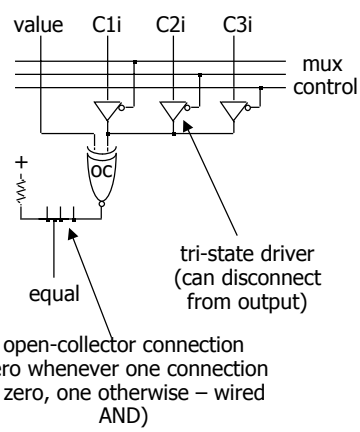
- Multiplexer
  - Easy to implement as combinational logic when few inputs
  - Logic can easily get too big for most PLDs

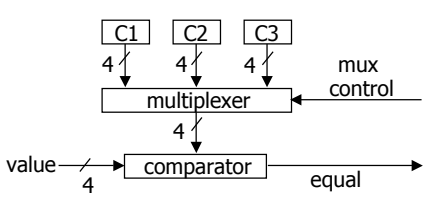




Datapath Implementation

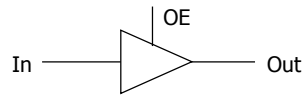
- Tri-State Logic
  - Utilize a third output state: "no connection" or "float"
  - Connect outputs together as long as only one is "enabled"
  - Open-collector gates can only output 0, not 1
    - Can be used to implement logical AND with only wires





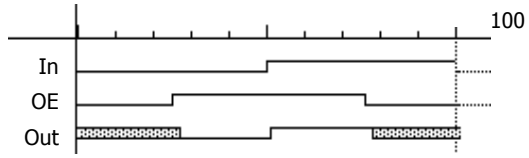
Tri-State Gates

- Third value
  - Logic values: "0", "1"
  - Don't care: "X" (must be 0 or 1 in real circuit!)
  - Third value or state: "Z" — high impedance, infinite R, no connection
- Tri-state gates
  - Additional input – output enable (OE)
  - Output values are 0, 1, and Z
  - When OE is high, the gate functions normally
  - When OE is low, the gate is disconnected from wire at output
  - Allows more than one gate to be connected to the same output wire
    - As long as only one has its output enabled at any one time (otherwise, sparks could fly)



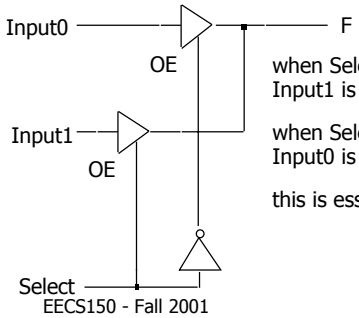
non-inverting tri-state buffer

In	OE	Out
X	0	Z
0	1	0
1	1	1



Tri-State and Multiplexing

- When Using Tri-State Logic
  - (1) Never more than one "driver" for a wire at any one time (pulling high and low at same time can severely damage circuits)
  - (2) Only use value on wire when its being driven (using a floating value may cause failures)
- Using Tri-State Gates to Implement an Economical Multiplexer



when Select is high
Input1 is connected to F

when Select is low
Input0 is connected to F

this is essentially a 2:1 mux

Digital Lock (New Datapath)

- Decrease number of inputs
- Remove 3 code digits as inputs
  - Use code registers
  - Make them loadable from value
  - Need 3 load signal inputs (net gain in input (4*3)–3=9)
    - Could be done with 2 signals and decoder (ld1, ld2, ld3, load none)