

Sequential Logic Examples

- Finite State Machine Concept
 - FSMs are the decision making logic of digital designs
 - Partitioning designs into datapath and control elements
 - When inputs are sampled and outputs asserted
- Basic Design Approach: 4-step Design Process
- Implementation Examples and Case Studies
 - Finite-string pattern recognizer
 - Complex counter
 - Traffic light controller
 - Door combination lock

CS 150 - Spring 2001 - Sequential Logic Examples - 1

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

CS 150 - Spring 2001 - Sequential Logic Examples - 2

Finite String Pattern Recognizer (Step 1)

- 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:

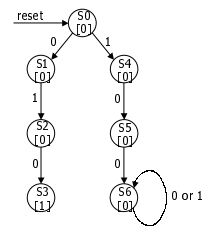
X: 00101010010 ...
 Z: 00010101000 ...

X: 11011010010 ...
 Z: 00000001000 ...

CS 150 - Spring 2001 - Sequential Logic Examples - 3

Finite String Pattern Recognizer (Step 2)

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



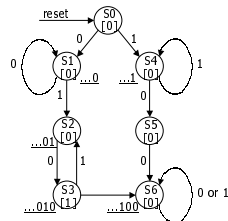
CS 150 - Spring 2001 - Sequential Logic Examples - 4

Finite String Pattern Recognizer (Step 2, cont'd)

- 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



CS 150 - Spring 2001 - Sequential Logic Examples - 5

Finite String Pattern Recognizer (Step 2, cont'd)

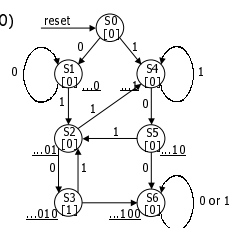
- S2 and S5 still have incomplete transitions

- 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)

- 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



CS 150 - Spring 2001 - Sequential Logic Examples - 6

Finite String Pattern Recognizer (Step 3)

- Verilog description including state assignment (or state encoding)

```

module string (clk, X, rst, Q0, Q1, Q2, Z);
input clk, X, rst;
output Q0, Q1, Q2, Z;

reg state[0:2];

`define S0 = [0,0,0]; //reset state
`define S1 = [0,0,1]; //strings ending in ...0
`define S2 = [0,1,0]; //strings ending in ...01
`define S3 = [0,1,1]; //strings ending in ...010
`define S4 = [1,0,0]; //strings ending in ...1
`define S5 = [1,0,1]; //strings ending in ...10
`define S6 = [1,1,0]; //strings ending in ...100

assign Q0 = state[0];
assign Q1 = state[1];
assign Q2 = state[2];
assign Z = (state == `S3);

always @(posedge clk) begin
    if rst state = `S0;
    else
        case (state)
            `S0: if (X) state = `S4 else state = `S1;
            `S1: if (X) state = `S2 else state = `S1;
            `S2: if (X) state = `S4 else state = `S3;
            `S3: if (X) state = `S2 else state = `S6;
            `S4: if (X) state = `S4 else state = `S5;
            `S5: if (X) state = `S2 else state = `S6;
            `S6: state = `S6;
            default: begin
                $display ("invalid state reached");
                state = 3'bxxx;
            endcase
        endcase
    end
endmodule
    
```

CS 150 - Spring 2001 - Sequential Logic Examples - 7

Finite String Pattern Recognizer

- Review of Process
 - Understanding problem
 - Write down sample inputs and outputs to understand specification
 - Derive a state diagram
 - Write down sequences of states and transitions for sequences to be recognized
 - Minimize number of states
 - Add missing transitions; reuse states as much as possible
 - State assignment or encoding
 - Encode states with unique patterns
 - Simulate realization
 - Verify I/O behavior of your state diagram to ensure it matches specification

CS 150 - Spring 2001 - Sequential Logic Examples - 8

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

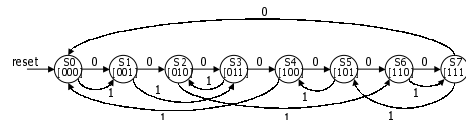
- Valid I/O behavior (partial)

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

CS 150 - Spring 2001 - Sequential Logic Examples - 9

Complex Counter (State Diagram)

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



CS 150 - Spring 2001 - Sequential Logic Examples - 10

Complex Counter (State Encoding)

- Verilog description including state encoding

```

module string (clk, M, rst, Z0, Z1, Z2);
input clk, X, rst;
output Z0, Z1, Z2;

reg state[0:2];

`define S0 = [0,0,0];
`define S1 = [0,0,1];
`define S2 = [0,1,0];
`define S3 = [0,1,1];
`define S4 = [1,0,0];
`define S5 = [1,0,1];
`define S6 = [1,1,0];
`define S7 = [1,1,1];

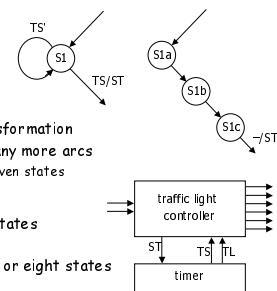
assign Z0 = state[0];
assign Z1 = state[1];
assign Z2 = state[2];

always @(posedge clk) begin
    if rst state = `S0;
    else
        case (state)
            `S0: state = `S1;
            `S1: if (M) state = `S3 else state = `S2;
            `S2: if (M) state = `S6 else state = `S3;
            `S3: if (M) state = `S2 else state = `S4;
            `S4: if (M) state = `S0 else state = `S5;
            `S5: if (M) state = `S4 else state = `S6;
            `S6: if (M) state = `S7 else state = `S7;
            `S7: if (M) state = `S5 else state = `S0;
            endcase
        endcase
    end
endmodule
    
```

CS 150 - Spring 2001 - Sequential Logic Examples - 11

Traffic Light Controller as Two Communicating FSMs

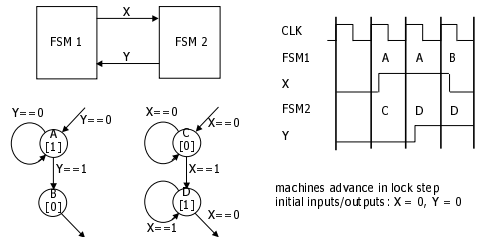
- Without Separate Timer
 - S0 would require 7 states
 - S1 would require 3 states
 - S2 would require 7 states
 - S3 would require 3 states
 - S1 and S3 have simple transformation
 - S0 and S2 would require many more arcs
 - C could change in any of seven states
- By Factoring Out Timer
 - Greatly reduce number of states
 - 4 instead of 20
 - Counter only requires seven or eight states
 - 12 total instead of 20



CS 150 - Spring 2001 - Sequential Logic Examples - 12

Communicating Finite State Machines

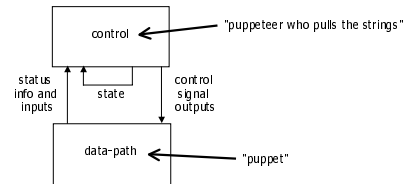
- One machine's output is another machine's input



CS 150 - Spring 2001 - Sequential Logic Examples - 13

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



CS 150 - Spring 2001 - Sequential Logic Examples - 14

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
 - Output: 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

CS 150 - Spring 2001 - Sequential Logic Examples - 15

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;

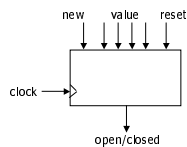
    while (!new_value( ));
    v3 = read_value( );
    if (v3 != c[3]) then error = 1;

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

CS 150 - Spring 2001 - Sequential Logic Examples - 16

Determining Details of the Specification

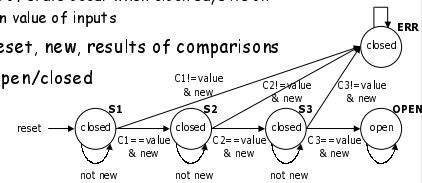
- How many bits per input value?
- How many values in sequence?
- How do we know a new input value is entered?
- What are the states and state transitions of the system?



CS 150 - Spring 2001 - Sequential Logic Examples - 17

Digital Combination Lock 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



CS 150 - Spring 2001 - Sequential Logic Examples - 18

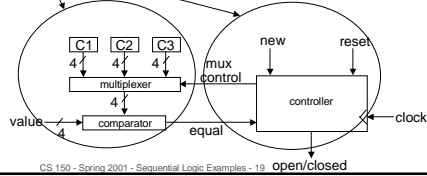
Datapath and Control Structure

Datapath

- Storage registers for combination values
- Multiplexer
- Comparator

Control

- Finite-state machine controller
- Control for data-path (which value to compare)



CS 150 - Spring 2001 - Sequential Logic Examples - 19

State Table for Combination Lock

Finite-State Machine

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

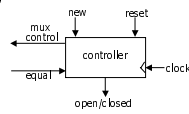
reset	new	equal	state	next state	mux	open/closed
1	-	-	-	S1	C1	closed
0	0	-	S1	S1	C1	closed
0	1	0	S1	ERR	-	closed
0	1	1	S1	S2	C2	closed
...
0	1	1	S3	OPEN	-	open
...

CS 150 - Spring 2001 - Sequential Logic Examples - 20

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
...

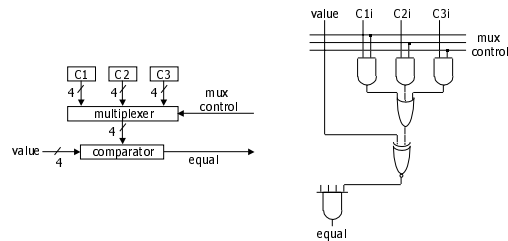
CS 150 - Spring 2001 - Sequential Logic Examples - 21

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 for Combination Lock

Multiplexer

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

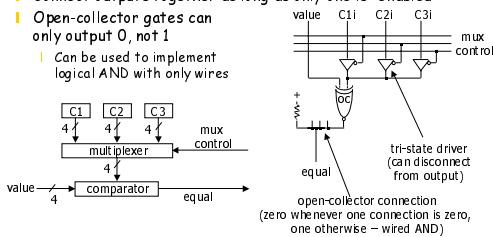


CS 150 - Spring 2001 - Sequential Logic Examples - 22

Datapath Implementation (cont'd)

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



CS 150 - Spring 2001 - Sequential Logic Examples - 23

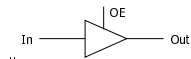
Tri-State Gates

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)



	In	OE	Out
non-inverting	X	0	Z
tri-state	0	1	0
buffer	1	1	1

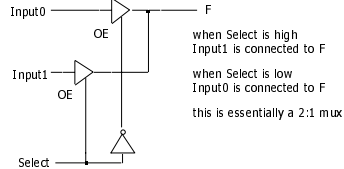
CS 150 - Spring 2001 - Sequential Logic Examples - 24

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)

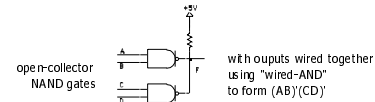
Using Tri-State Gates to Implement an Economical Multiplexer



CS 150 - Spring 2001 - Sequential Logic Examples - 25

Open-Collector Gates and Wired-AND

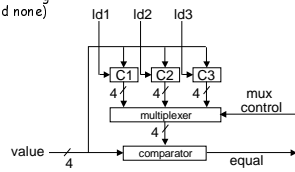
- Open collector:** another way to connect gate outputs to same wire
 - Gate only has the ability to pull its output low
 - Cannot actively drive wire high (default - pulled high through resistor)
- Wired-AND can be implemented with open collector logic**
 - If A and B are "1", output is actively pulled low
 - If C and D are "1", output is actively pulled low
 - If one gate output is low and the other high, then low wins
 - If both outputs are "1", the wire value "floats", pulled high by resistor
 - Low to high transition usually slower than if gate pulling high
 - Hence, the two NAND functions are ANDed together



CS 150 - Spring 2001 - Sequential Logic Examples - 26

Digital Combination 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)



CS 150 - Spring 2001 - Sequential Logic Examples - 27

Section Summary

- FSM Design**
 - Understanding the problem
 - Generating state diagram
 - Implementation using synthesis tools
 - Iteration on design/specification to improve qualities of mapping
 - Communicating state machines
- Four case studies**
 - Understand I/O behavior
 - Draw diagrams
 - Enumerate states for the "goal"
 - Expand with error conditions
 - Reuse states whenever possible

CS 150 - Spring 2001 - Sequential Logic Examples - 28