

Penn Engineering **ESE**

Lecture #7 – Digital Logic

ESE 150 – DIGITAL AUDIO BASICS

ESE150 Spring 2019

Based on slides © 2009–2019 DeHon

ESE150 Spring 2019

SO FAR

MUSIC (1) → MIC → A/D → 10101001101

10101001101 → DFT Identify Masking Huffman encoding

10101001101 → domain conversion (5,6) → compress (3) → Huffman Decode IDFT

10101001101 → D/A → SPEAKER

Labels: sample (2), freq (4), pyscho-acoustics (3), EULA, click OK, MP3 Player / iPhone / Droid

Numbers correspond to course weeks

ESE150 Spring 2019

HOW PROCESS

- × How do we build a machine to perform these operations?
 - + From Digital Samples → compressed digital data → Digital Samples
- × Down to bottom
 - + If we can build **one** kind of primitive element,
 - × ...and connect together large collections of them
 - + can build a machine to perform *any* digital computation

3

ESE150 Spring 2019

LECTURE TOPICS

- × Setup
- × Where are we?
- × Combinational Logic
- × Sequential Logic
- × FPGAs
- × Next Lab

4

ESE150 Spring 2019

COURSE MAP

MUSIC (1) → MIC → A/D → CPU (7,8,9) → File-System (10) → 10101001101 → NIC (11) → Cloud (11)

10101001101 → D/A → SPEAKER

Labels: sample (2), freq (4), pyscho-acoustics (3), EULA, click OK, MP3 Player / iPhone / Droid

Numbers correspond to course weeks

12

ESE150 Spring 2019

COURSE MAP – WEEK 8

MUSIC (1) → MIC → A/D → 10101 → Logic Gate → Flip-Flop

10101001101 → D/A → SPEAKER

Labels: sample (2), freq (4), pyscho-acoustics (3), EULA, click OK, MP3 Player / iPhone / Droid

Numbers correspond to course weeks

6

ESE150 Spring 2019

COMBINATIONAL LOGIC

7

ESE150 Spring 2019

GATE

- × **Primitive binary function**
 - + Computes a binary output from a small number of binary inputs
- × **Can specify function with a Truth Table**
 - + Defines the output for each input combination


Input 0	Input 1	Output
0	0	
0	1	
1	0	
1	1	

8

ESE150 Spring 2019

AND GATE

- × **AND**
 - + Output is 1 (true) when all inputs are 1 (true)




Input 0	Input 1	AND
0	0	0
0	1	0
1	0	0
1	1	1

9

ESE150 Spring 2019

NOT GATE

- × **Not**
 - + Output is opposite of input




Input	NOT
0	1
1	0

10

ESE150 Spring 2019

OR GATE

- × **OR**
 - + Output is 1 (true) when any input is 1 (true)
 - + (fill in truth table for OR)



Input 0	Input 1	OR
0	0	
0	1	
1	0	
1	1	

11

ESE150 Spring 2019

CLAIM

- × **Can compute any Boolean Function from AND, OR, NOT**
 - + (actually from NAND)

12

MODEL: COMBINATIONAL LOGIC

- × **Compute some “function”**
 - + $f(i_0, i_1, \dots, i_n) \rightarrow o_0, o_1, \dots, o_m$
- × **Each unique input vector**
 - + implies a particular, deterministic, output vector

13

BIG AND

- × **AND**
 - + Output is 1 (true) when all inputs are 1 (true)
- × **How build n-input AND from AND2 gates?**



14

BIG OR

- × **OR**
 - + Output is 1 (true) when any input is 1 (true)
- × **How build n-input OR from OR2?**



15

INPUT CASE

- × **How can we create an expression that is true for a specific input case?**
 - + E.g. have a function of 4 inputs: a, b, c, d
- × **How many potential values for a, b, c, d?**
 - + Rows in our truth table
- × **Give one example of values for a, b, c, d?**
- × **How create an expression that is true for that case?**

16

SINGLE OUTPUT DIGITAL FUNCTION

- × **Given have logic to implement each input case**
- × **How implement entire function?**

a	b	c	F(a,b,c)
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

17

MULTIPLE OUTPUT FUNCTION

- × **What do you do if your Digital Function needs multiple output bits?**

18

ESE150 Spring 2019

COMBINATIONAL LOGIC AS GATES

- × **Start with truth table**
- × **Single output {0, 1}**
 - + Use inverters to produce complements of inputs
 - + For each input case
 - × If output is a 1
 - × Develop an AND to detect that case
 - × Decompose AND into gates
 - + OR together the output of all such AND functions
 - × Decompose OR into gates
- × **Multiple outputs**
 - + Repeat for each output

This solution won't typically be the smallest or fastest...

19

ESE150 Spring 2019

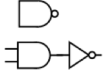
CONCLUDE

- × **Can implement any combinational logic function out of a collection of**
 - + OR2, AND2, NOT gates

20

ESE150 Spring 2019

NAND2 GATE



- × **NAND = NOT AND**
 - + Output is 0 (true) when all inputs are 1 (true); 0 otherwise

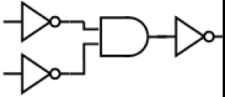
Input 0	Input 1	NAND
0	0	1
0	1	1
1	0	1
1	1	0

21

ESE150 Spring 2019

NAND UNIVERSALITY

- × **How implement**
 - + NOT from NAND2?
 - + AND2 from NAND2?
 - + What function does this circuit implement?

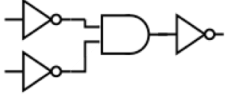


Input 0	Input 1	Function?
0	0	
0	1	
1	0	
1	1	

22

ESE150 Spring 2019

NAND UNIVERSALITY

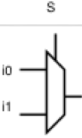


- × **Can implement**
 - + NOT from NAND2
 - + AND2 from NAND2
 - + OR2 from NAND2
- × **Can implement any combinational logic function out of a collection of**
 - + OR2, AND2, NOT gates
- × **Therefore: Can implement any combinational logic function out of a collection of NAND2 gates**

23

ESE150 Spring 2019

MULTIPLEXER GATE



- × **MUX**
 - + When S=0, output=i0
 - + When S=1, output=i1

S	i0	i1	Mux2(S,i0,i1)
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Truth Table?
AND, OR, NOT
Implementation?

24

ESE150 Spring 2019

ARITHMETIC

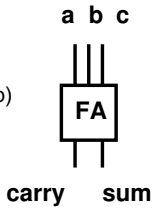
- × **Addition is also a digital logic function**
 - + Maps set of inputs (a3 a2 a1 a0 b3 b2 b1 b0)
 - + To an output bit vector (c4 c3 c2 c1 c0)
- × **...as is subtraction, multiplication, division, square root....**

25

ESE150 Spring 2019

FULL ADDER

- × **Adds 3 inputs to produce 2b output**
 - + Binary inputs: a, b, c
 - + Binary outputs: carry, sum
 - + Two bit result:
 - + $carry * 2 + sum = a + b + c$
 - + Can produce truth table and logic (Lab)



26

Penn ESE534 Spring2016 --

EXAMPLE: BIT LEVEL ADDITION

- × **Addition**
 - + Base 2 example
 - + *Work together*

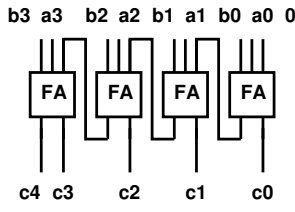
C: 1101101000
A: 01101101010
B: 01100101100
S: 11010010110

27

ESE150 Spring 2019

N-BIT ADDER

- × **Given Full Adders**
 - + Can build N-bit adder by connecting N full adders



28

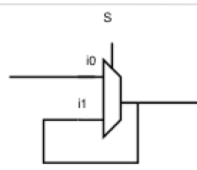
ESE150 Spring 2019

SEQUENTIAL LOGIC

29

ESE150 Spring 2019

MUX WITH FEEDBACK



- × **What happens when S=0?**
- × **What happens when S=1?**

30

ESE150 Spring 2019

MUX WITH FEEDBACK

- Assuming i_0 doesn't change what happens when S goes from 0 to 1?

31

ESE150 Spring 2019

LATCH

- Element that can hold a previous value of an input

32

ESE150 Spring 2019

FLIP-FLOP (FF)

- Use a pair to create a flip-flop
 - Also call register
- What happens when
 - CLK is low (0)?
 - CLK is high (1)?
 - CLK transitions from 0 to 1?

33

ESE150 Spring 2019

FLIP-FLOP (FF)

- Use a pair to create a flip-flop
 - Also call register
- Sample D input on 0→1 transition of clock (CLK)
- Never an open path from D→Q
 - One of the mux latches always in hold state

34

ESE150 Spring 2019

STATE ELEMENT

- Latch or Register is a state element
- Allows circuit to remember a value
- Build computations that
 - Depend on past inputs
 - Reuse hardware in time

35

ESE150 Spring 2019

ACCUMULATOR

- What happens:
 - Start with $a_3:a_0$ at 0
 - CLK low
 - $I_3:i_0=2$ (0010)
 - FF inputs?
 - CLK goes high: $a_3:a_0?$
 - $I_3:0=3$ (0011)
 - FF inputs?
 - CLK goes high: $a_3:a_0?$

36

ESE150 Spring 2019

ACCUMULATOR

- × **a=0**
- × **while (true)**
 - + a=a+getInput();
- × **Accumulates input values i**
 - + Integration or summation

37

ESE150 Spring 2019

STATE FOR SEQUENCING AND CONTROL

- × **Useful when trying to control things**
 - + E.g. Perform a sequence of operations
- × **Robot**
 - + Open-gripper
 - + Move-forward
 - + Close-gripper
 - + Lift

38

ESE150 Spring 2019

STATE FOR CONDITIONAL CONTROL

- × **Useful when need to behave differently based on something in the past**
 - + Remember if elevator going up or down
 - + Remember/count coins from consumer
 - + Remember some mode set by user
 - × Displaying in Centigrade or Fahrenheit
- × **Idea**
 - + Store state
 - + Use as input to logic

39

ESE150 Spring 2019

FINITE-STATE MACHINE (FSM)

- × **Sequential model of computation**
- × **State (in registers) + combinational logic**
- × **Compute outputs and next state from inputs and state**

40

ESE150 Spring 2019

FSM EXAMPLE

- × **Simplified Vending Machine**
 - + Only input quarters
 - + Only vend one item (output signal to indicate vending)
 - + Item costs 2 quarters
 - + Coin Return request and control
- × **Two states: waiting, one-quarter (one)**
- × **Two inputs: quarter, coin-return (creturn)**
- × **Two outputs: vend, return-quarter (qreturn)**

41

ESE150 Spring 2019

TRUTH TABLE MODEL

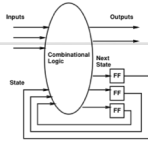
Complete together

state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1			
waiting	1	0	0	0	one
waiting	1	1			
one	0	0			
one	0	1			
one	1	0			
one	1	1			

42

ESE150 Spring 2019

TRUTH TABLE MODEL



state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1	0	0	waiting
waiting	1	0	0	0	one
waiting	1	1	0	1	waiting
one	0	0	0	0	one
one	0	1	0	1	waiting
one	1	0	1	0	waiting
one	1	1	0	1	one

43

ESE150 Spring 2019

SWITCH-STATEMENT MODEL

```

* While (true)
*   switch (state) {
*     case waiting:
*       if (quarter && !creturn)
*         state=one;
*       else
*         state=waiting;
*         qreturn=quarter && creturn;
*         vend=0;
*       break;

```

44

ESE150 Spring 2019

SWITCH-STATEMENT MODEL (CONT.)

```

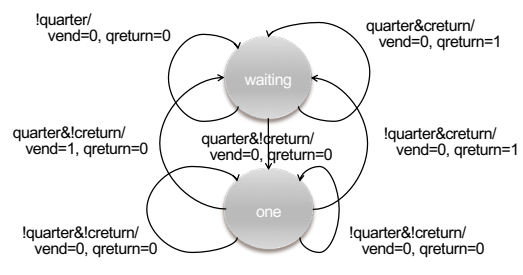
*   case one:
*     if ((quarter && !creturn)||
*         (!quarter&&creturn))
*       state=waiting;
*     else
*       state=one;
*       qreturn=creturn;
*       vend=quarter&& !creturn;
*       break;
* } // switch
* } // while

```

45

ESE150 Spring 2019

FSM GRAPH MODEL



46

ESE150 Spring 2019

PROGRAMMABLE LOGIC

47

ESE150 Spring 2019

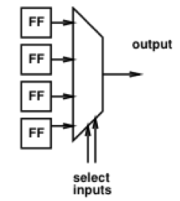
MUX CAN BE A PROGRAMMABLE GATE

```

* Programmable Gate
+ Can be programmed to act as any gate
+ Use state (e.g. FF) to "program" truth table of a gate

```

Input 0	Input 1	Output
0	0	
0	1	
1	0	
1	1	



48

ESE150 Spring 2019

EXAMPLE: AND

- How do we program to behave as AND?

49

ESE150 Spring 2019

EXAMPLE: OR

- How do we program to behave as OR?

50

ESE150 Spring 2019

LOOK-UP TABLE (LUT)

- Can generalize to any number of inputs

51

ESE150 Spring 2019

CONNECTING GATES

- Once we can build gates
- ...still need to connect the gates together.
- Select which gate outputs become inputs to other gates.

52

ESE150 Spring 2019

MUX CAN BE PROGRAMMABLE INTERCONNECT

Trick: Use multiplexer to programmably select gate input.

53

ESE150 Spring 2019

PROGRAMMABLE BLOCKS

54

ESE150 Spring 2019

PROGRAMMABLE GATES AND INTERCONNECT

If time permits:
How program (fill in yellow programmable cells) to implement a full adder?

55

ESE150 Spring 2019

FIELD-PROGRAMMABLE GATE ARRAY (FPGA)

- × **Collection of Programmable Gates**
 - + Can “program” by setting state bits
 - + LUTs that can be programmed to be any gate
 - × With optional Flip-Flops to use for state
 - + Programmable interconnect to “wire” the gates together

56

ESE150 Spring 2019

FIELD-PROGRAMMABLE GATE ARRAY (FPGA)

57

ESE150 Spring 2019

NEXT LAB

- × **Program an FPGA in Verilog**
 - + Build an adder
 - + Build an accumulator

58

ESE150 Spring 2019

BIG IDEAS

- × Can implement any combinational digital logic function from nand2 gates
- × Can implement any FSM from nand2 gates and registers
- × Can build a single chip that can be programmed to behave as any collection of gates
 - + As long as don't need more gates than it provides

59

ESE150 Spring 2019

LEARN MORE

- × CIS240 – do a bit more logic
- × ESE370 – how to implement gates, latches, and memories from transistors
- × ESE532 – how to build large-scale computations from logic

60

REMINDER

- × **Formal Lab Report Due Sunday (11:59pm)**
- × **Extra Lab Session to help**
 - + Don't get stuck for hours on any piece