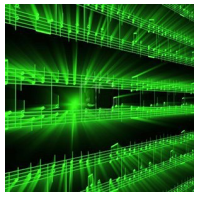
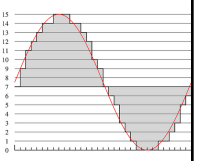
Penn Engineering

ESE





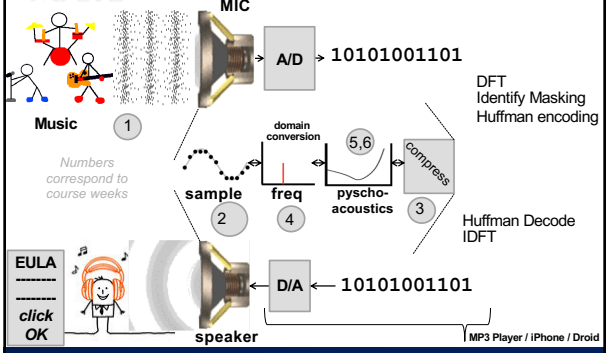
Lecture #13 – Combinational Logic

ESE 150 – Digital Audio Basics

ESE150 Spring 2021

Based on slides © 2009--2021 DeHon

So far



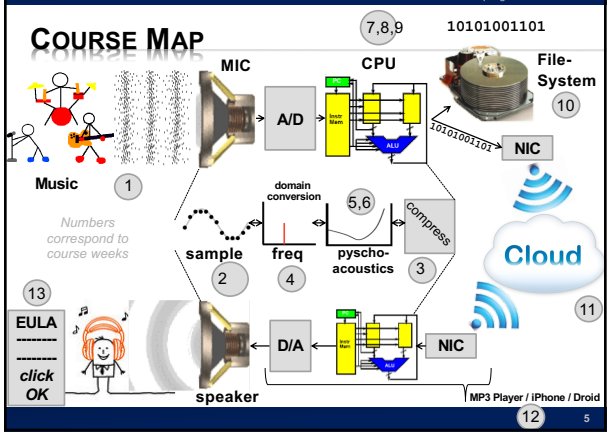
How Process

- **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 (maybe 2),
    - …and connect together large collections of them
  - can build a machine to perform *any* digital computation

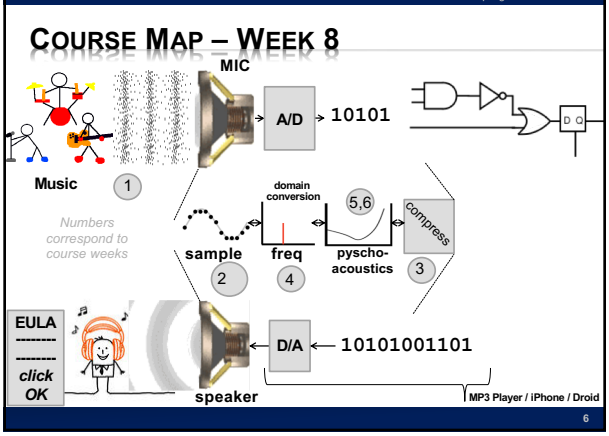
Lecture Topics

- Setup
- **Where are we?**
- **Combinational Logic**
- **Accumulator**
- **Next Lab**

Course Map



Course Map – Week 8



COMBINATIONAL LOGIC

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	

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

NOT GATE

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

input	NOT
0	1
1	0

OR GATE

- **OR**
  - Output is 1 (true) when any input is 1 (true)
  - (fillin truth table for OR)

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

CLAIM

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

Model: Combinational Logic

- **Compute some "function"**
  - $f(i_0,i_1,\ldots i_n) \rightarrow o_0,o_1,\ldots o_m$
- **Each unique input vector**
  - implies a particular, deterministic, output vector

Big AND

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

Big OR

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

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?**

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

Multiple Output Function

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

Combinational Logic as Gates

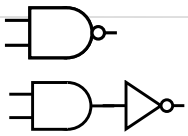
- **Start with truth table**
- **Single output {0, 1}**
  - Use inverters to produce complements of inputs
  - For each input case
    - If output is a 1
      - Develop an AND to detect that case
        - Decompose AND into gates
  - OR together the output of all such AND functions
    - Decompose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

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

Conclude

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

NAND2 Gate

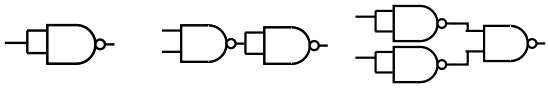


- **NAND = NOT AND**
  - Output is 0 (false) when all inputs are 1 (true); 0 otherwise

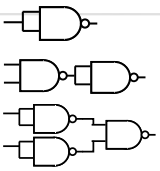
Input 0	Input 1	NAND
0	0	1
0	1	1
1	0	1
1	1	0

NAND Universality

- **How implement**
  - What function does each circuit implement?

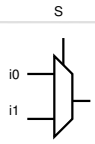


NAND Universality



- **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**

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?

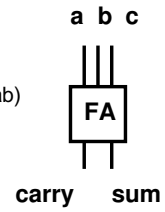
AND, OR, NOT Implementation?

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….**

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)



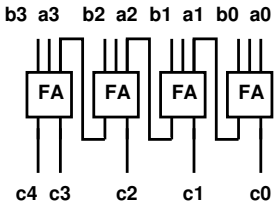
EXAMPLE: BIT-LEVEL ADDITION

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

C: 1**1011010000**
A: 01101101010
B: 01100101100
S: 1**1010010110**

N-BIT ADDER

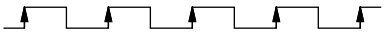
- **Given Full Adders**
  - Can build N-bit adder by connecting N full adders



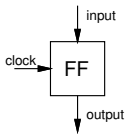
ACCUMULATOR

REGISTER

- **Clock**



  - Defines the rate of the computation
  - Typically a square wave
  - Rising clock edge defines beginning of new cycle
- **State Element – Flip-Flop (FF) or Register**
  - Returns the value it was given on previous cycle = before the last rising clock edge
- **More Details next time**



ACCUMULATOR

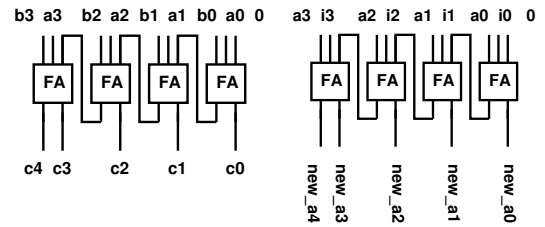
- **Sum a sequence of values**
  - **a=0**
  - **while (true)**
    - a=a+getInput();

$$a = \sum_{i=0} input[i]$$

Input[i]	a
	0
1	
	1
3	
	4
1	
	5

ACCUMULATOR

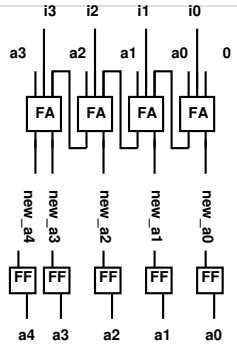
- **Start with an Adder**



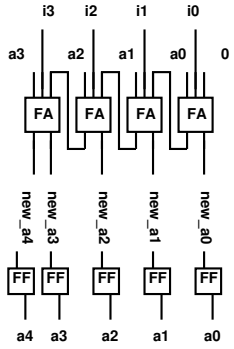
ACCUMULATOR

- **Store running sum as state in registers**
- **Sum up new values provided on successive cycles**

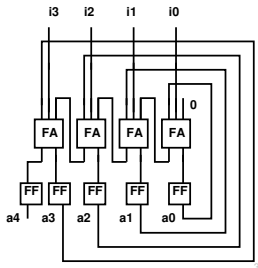
Input[i]	a
	0
1	
	1
3	
	4
1	
	5



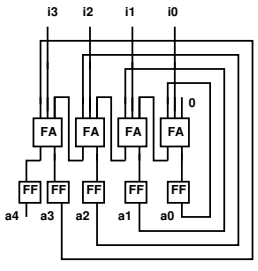
ACCUMULATOR



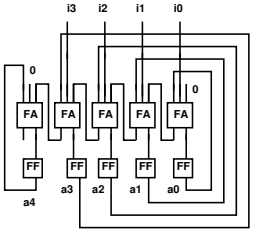
- Wrap register outputs back to inputs



ACCUMULATOR



- Maybe extend accumulator bits to hold larger sum

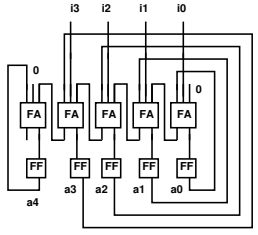


- Maybe more…

ACCUMULATOR

Input[i]	a
	0
15	
	15
15	
	30
15	
	45
15	
	60
15	
	75

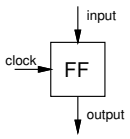
- Maybe extend accumulator bits to hold larger sum

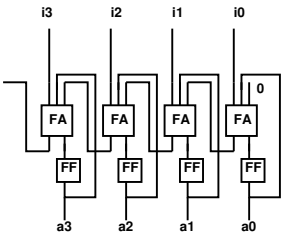


- Maybe more…

ACCUMULATOR

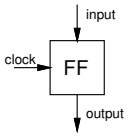
- **What happens:**
  - Start with a3:a0 at 0
  - CLK low
  - I3:i0=2 (0010)
  - FF inputs?

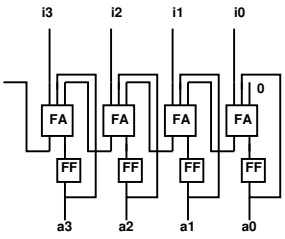




ACCUMULATOR

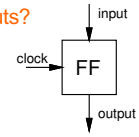
- **What happens:**
  - Start with a3:a0 at 0
  - CLK low
  - I3:i0=2 (0010)
  - FF inputs?
  - CLK goes high: a3:a0?

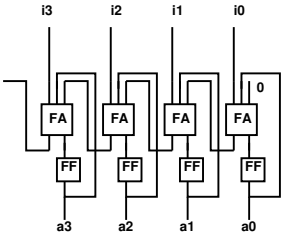




ACCUMULATOR

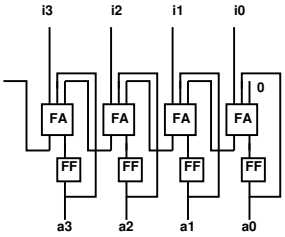
- **What happens:**
  - Start with a3:a0 at 0
  - CLK low
  - I3:i0=2 (0010)
  - CLK goes high: a3:a0?
  - I3:0=3 (0011)
  - FF inputs?





ACCUMULATOR

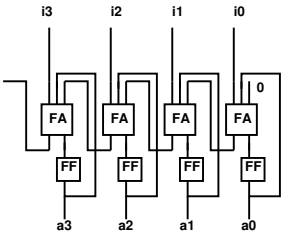
- **What happens:**
  - Start with a3:a0 at 0
  - CLK low
  - I3:i0=2 (0010)
  - CLK goes high: a3:a0?
  - I3:0=3 (0011)
  - FF inputs?
  - CLK goes high: a3:a0?



ACCUMULATOR

- **a=0**
- **while (true)**
  - a=a+getInput();

$$a = \sum_{i=0} input[i]$$



NEXT LAB

- **Lab 7 posted on web**
- **Program an FPGA in Verilog**
  - Build an adder
  - Build an accumulator
- **FPGA is in your kit**
- **Will need to install software on your computer**

BIG IDEAS

- **Can implement any combinational digital logic function from nand2 gates**
- **Can store previous values**
  - Flip-flops or registers

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**

REMINDER

- **Feedback**
- **Extra TA Session on Saturday**
- **Spring Forward Sunday morning**
  - Daylight savings time begins
  - Lose an hour
- **Formal Lab Report Due Sunday (11:59pm PDT)**