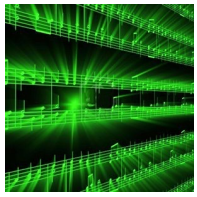
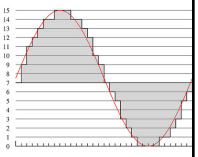
Penn Engineering

ESE





Lecture #14 – Sequential Logic, FPGAs

ESE 150 – Digital Audio Basics

ESE150 Spring 2022

Based on slides © 2009--2022 DeHon

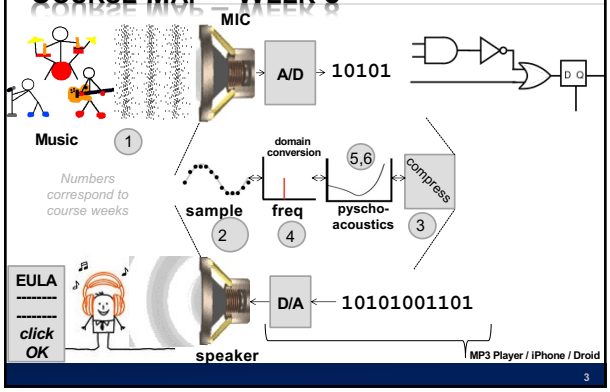
1

Lecture Topics

- Setup
- **Where are we?**
- **Part 1:**
  - Review: Combinational Logic
  - Registers
- **Part 2:**
  - FPGAs
- **Part 3:**
  - Finite-State Machines (FSM)

2

Course Map – Week 8



3

Combinational Logic

4

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	

5

Conclude

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

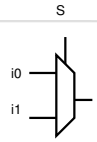
6

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….**

7

MULTIPLEXER GATE



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

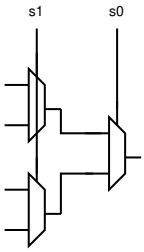
S	i0	i1	Mux2(S,i0,i1)
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Truth Table?

8

PRECLASS 3

- **How build 4-input mux from 2-input muxes?**

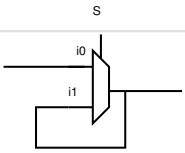


9

REGISTERS AND CLOCKING

10

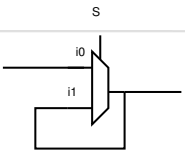
MUX WITH FEEDBACK



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

11

MUX WITH FEEDBACK

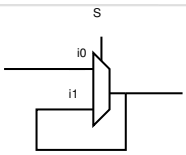


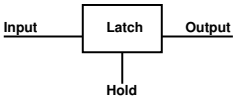
- **Assuming i0 doesn't change what happens when S goes from 0 to 1?**

12

LATCH

- **Element that can hold a previous value of an input**





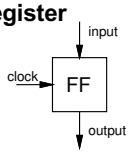
13

REGISTER

- **Clock**

  - Defines the rate of the computation
  - Typically a square wave
  - Rising clock edge defines beginning of new cycle

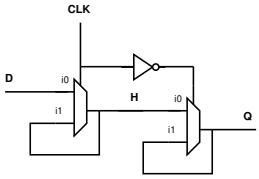
- **State Element – Flip-Flop (FF) or Register**
  - Returns the value it was given on previous cycle = before the last rising clock edge

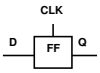


14

FLIP-FLOP (FF)

- **Use a pair of latches to create a flip-flop**
  - Also call register

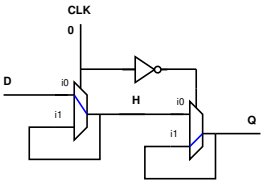


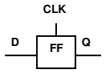


15

FLIP-FLOP (FF)

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

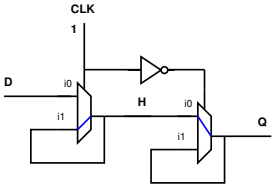


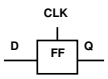


16

FLIP-FLOP (FF)

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

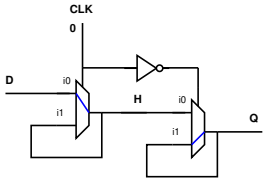


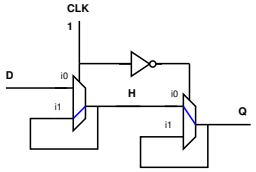


17

FLIP-FLOP (FF)

- **Use a pair to create a flip-flop**
  - Also call register
- **What happens when**
  - CLK transitions from 0 to 1?

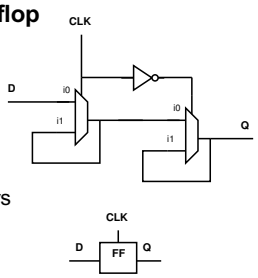




18

Flip-Flop (FF)

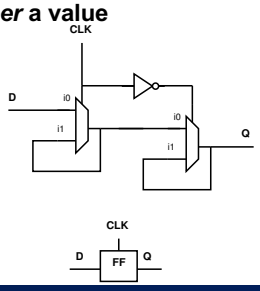
- **Use a pair to create a flip-flop**
  - Also call register
- **Sample D input on 0→1 transition of clock (CLK)**
- **Never an open path from D→Q**
  - One of the mux latches always in hold state



19

State Element

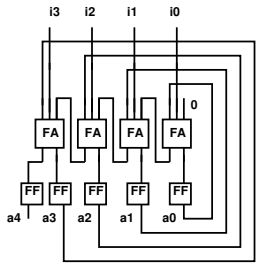
- **Latch or Register is a state element**
- **Allows circuit to *remember* a value**
- **Build computations that**
  - Depend on past inputs
  - Reuse hardware in time

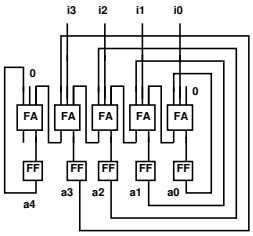


20

Accumulator Revisited

- Maybe extend accumulator bits to hold larger sum





- Maybe more…

21

Penn Engineering

ESE

Part 2

Programmable Logic

22

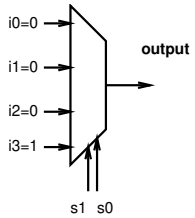
Preclass 2

- **How build 4-input mux from 2-input muxes?**

23

Preclass 4

- **What function of s0, s1 is this circuit configuration computing?**

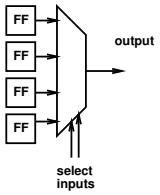


24

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

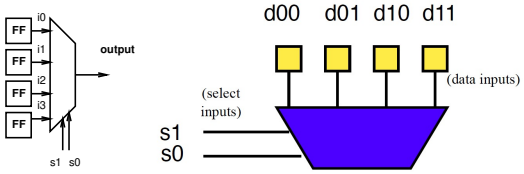
Input 0	Input 1	Output
0	0	
0	1	
1	0	
1	1	



25

EXAMPLE: OR (PRECLASS 5)

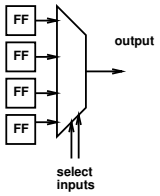
- **How do we program to behave as OR2?**



26

LOOK-UP TABLE (LUT)

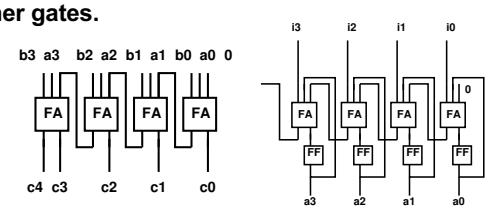
- **Can generalize to any number of inputs**



27

CONNECTING GATES

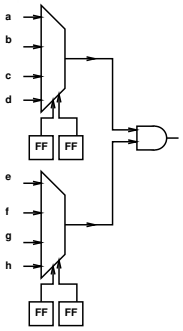
- **Once we can build gates**
- **…still need to connect the gates together.**
- **Select which gate outputs become inputs to other gates.**



28

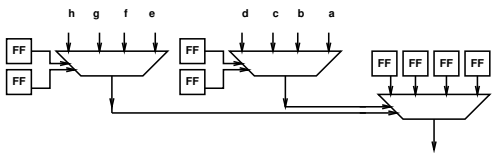
MUX CAN BE PROGRAMMABLE INTERCONNECT

Trick: Use multiplexer to programmably select gate input.



29

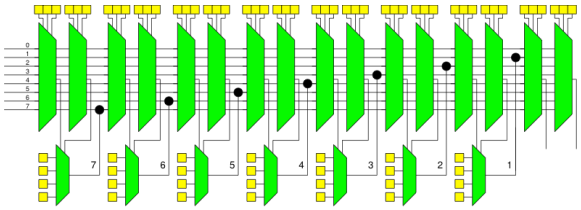
PROGRAMMABLE BLOCKS



30

Programmable Gates and Interconnect

Preclass 6:
How program (fill in yellow programmable cells) to implement a full adder?



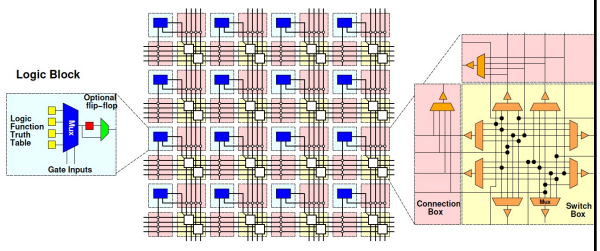
31

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

32

Field-Programmable Gate Array (FPGA)



33

Penn Engineering

ESE

Part 3

Finite State Machines (FSMs)

34

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

35

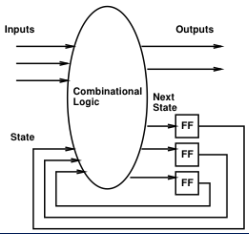
State for Conditional Control

- **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

36

Finite-State Machine (FSM)

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



37

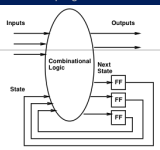
FSM Example

- **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)**

38

Truth Table Model

Complete together

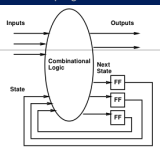


state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1			
waiting	1	0	0	0	one
waiting	1	1			
one	0	0			
one	0	1			
one	1	0			
one	1	1			

39

Truth Table Model

Complete together

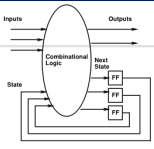


state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1	0	0	waiting
waiting	1	0	0	0	one
waiting	1	1			
one	0	0			
one	0	1			
one	1	0			
one	1	1			

40

Truth Table Model

Complete together

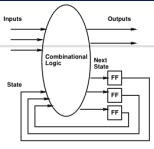


state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1	0	0	waiting
waiting	1	0	0	0	one
waiting	1	1	0	1	waiting
one	0	0			
one	0	1			
one	1	0			
one	1	1			

41

Truth Table Model

Complete together

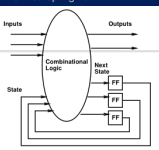


state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1	0	0	waiting
waiting	1	0	0	0	one
waiting	1	1	0	1	waiting
one	0	0	0	0	one
one	0	1			
one	1	0			
one	1	1			

42

Truth Table Model

Complete together

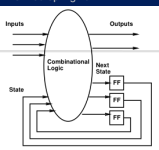


state	quarter	creturn	vend	qreturn	next
waiting	0	0	0	0	waiting
waiting	0	1	0	0	waiting
waiting	1	0	0	0	one
waiting	1	1	0	1	waiting
one	0	0	0	0	one
one	0	1	0	1	waiting
one	1	0			
one	1	1			

43

Truth Table Model

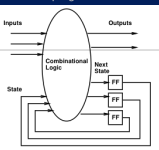
Complete together



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			

44

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

45

Switch-Statement Model

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

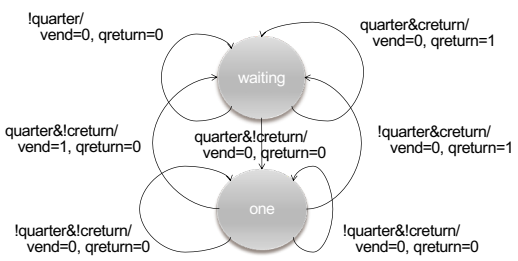

46

Switch-Statement Model (cont.)

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


47

FSM Graph Model



48

BIG IDEAS

- **Can implement any combinational digital logic function from:**
  - and, or, not gates
- **Can implement any FSM from:**
  - and, or, not 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

49

LEARN MORE

- **CIS240 – do a bit more logic**
- **ESE370 – how to implement gates, latches, and memories from transistors**
- **CIS471 – implement processor (next time) using Verilog mapped to FPGAs**
- **ESE532 – how to build large-scale computations from logic**

50

REMINDER

- **Feedback including Lab**
- **Lab 6 Report/formal writeup due tonight**
  - Before midnight
- **Lab 7 on Wednesday**
  - Prelab (with canvas/quiz turnin)

51