Decoders, muxes and intro to sequential circuits

Lecture 4 — § 2.2 - 2.3

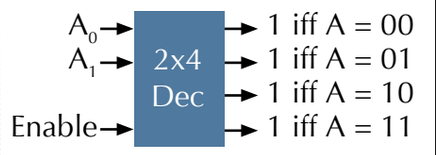
Computer Science 218

Mike Feeley

Decoders and encoders

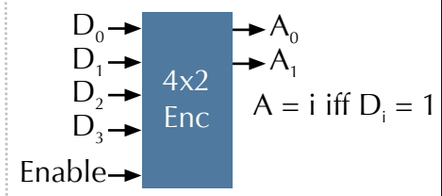
decoder selects one labelled output

- n-bit input pattern selects the output
  - you can think of this pattern as a number (address)
- m outputs labelled with unique patterns
- output is 1 iff its label matches input
- $m \leq 2^n$ outputs
  - its less if some input patterns are ignored



encoder does the opposite

- inputs are labelled with unique pattern
- output encodes pattern of input that = 1
  - there can be only one such input



uses

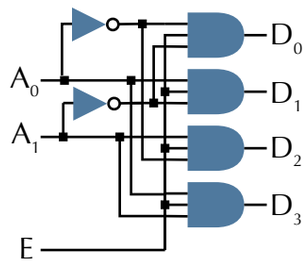
- addressing memory cells
- getting interrupt device address

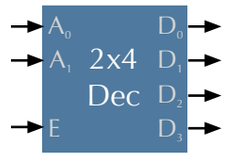
Implement a decoder ...

a 2x4 decoder

- two bit input; four bit output
- package and truth table look like this

draw the circuit ...



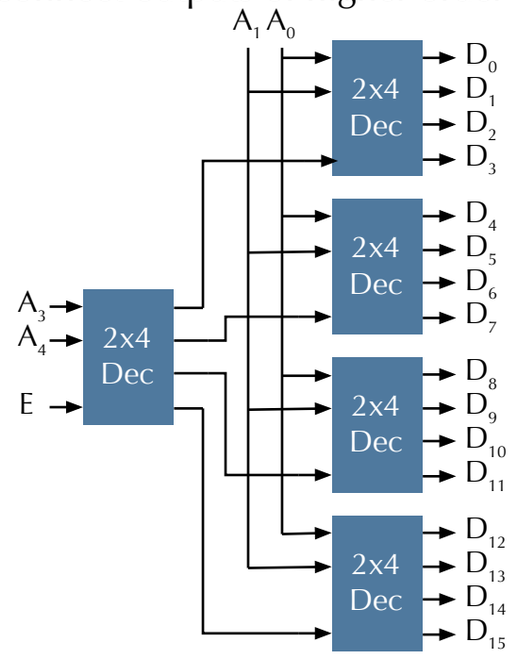


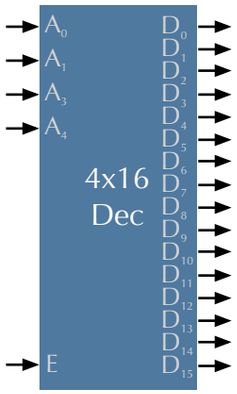
E	A_1	A_0	D_0	D_1	D_2	D_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1
0	x	x	0	0	0	0

Composing decoders using *enable*

build *2n*-bit decoder from *n+1 n*-bit decoders

- high-order inputs to one decoder, low-order inputs to all others
- connect outputs of higher-order decoder to *enables* of each other



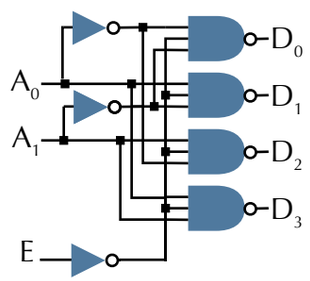


NAND decoders

NAND gates are cheaper than ANDs

- require fewer transistors to implement

a NAND decoder looks like this

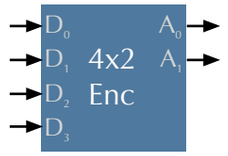


and this inverted truth table

E	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	x	x	1	1	1	1

Implementing an Encoder

encoder does the opposite of decoder



E	D_0	D_1	D_2	D_3	A_1	A_0
1	1	0	0	0	0	0
1	0	1	0	0	0	1
1	0	0	1	0	1	0
1	0	0	0	1	1	1
0	0	0	0	0	x	x

its implemented like this ...

$A_0 = ED_1 + ED_3 = E(D_1 + D_3)$

$A_1 = ED_2 + ED_3 = E(D_2 + D_3)$

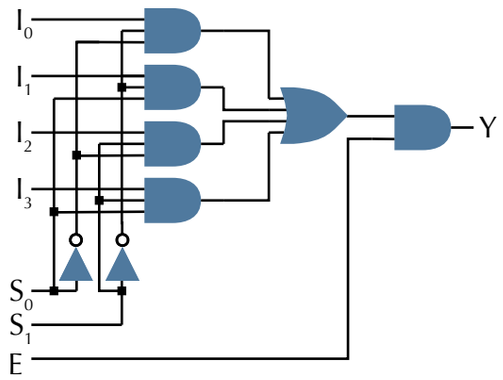
or gates and an **and** for enable

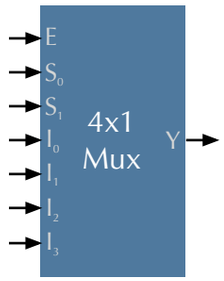
Multiplexers

select one input to send to output

- 2^n *data* inputs plus n *select* inputs
  - data inputs are labelled with unique n-bit number
- one output
  - has value of data input with label matching select

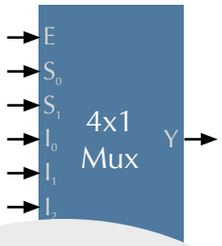
implementation

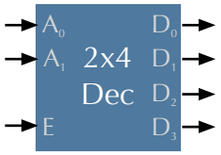


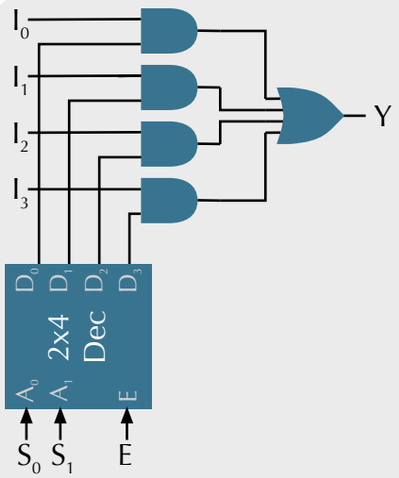


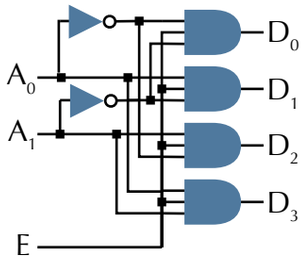
E	S_1	S_0	Y
1	0	0	I_0
1	0	1	I_1
1	1	0	I_2
1	1	1	I_3
0	x	x	0

Comparing mux and decoder



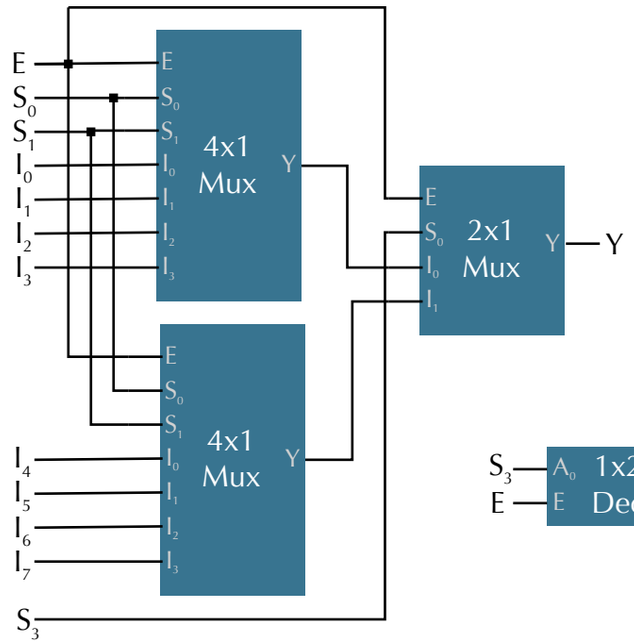






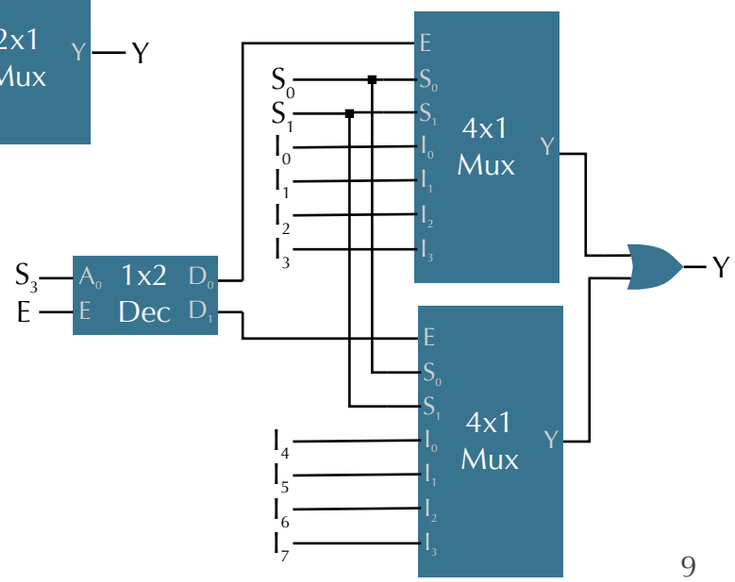
Can you implement a mux using a dec? ...

Composing to form *2nx1* mux



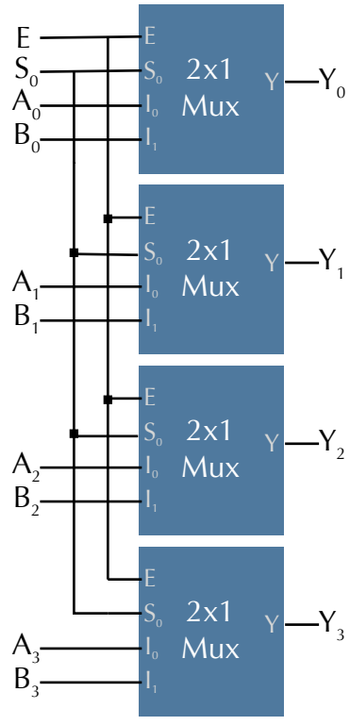
two 4x1 and one 2x1 mux

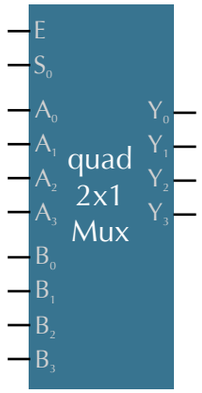
two 4x1 and one 2x1 dec



Composing to form *multipole* mux

quadruple mux switches 4 bits at once





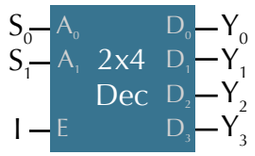
E	S	Y
0	x	all 0's
1	0	A
1	1	B

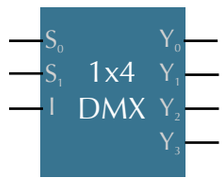
Demultiplexer

direct input to one of 2^n outputs

- one data input, n select inputs
- 2^n outputs each labelled with number
  - output label named by select = input; others = 0

what does this look like? ...





S_1	S_0	Y_0	Y_1	Y_2	Y_3
0	0	I			
0	1		I		
1	0			I	
1	1				I

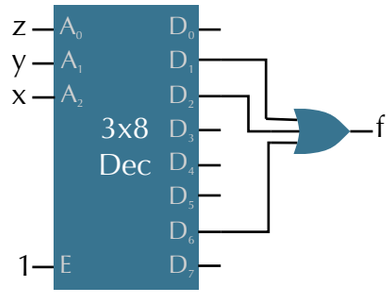
Fun with decoders and muxes

convert truth table to logic diagram

- not necessarily efficient, but its easy

decoder

- connect input variables to dec's select
- one dec output for each row of table
- **or** dec outputs for row's where f=1

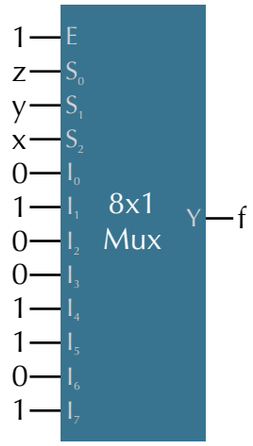


x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

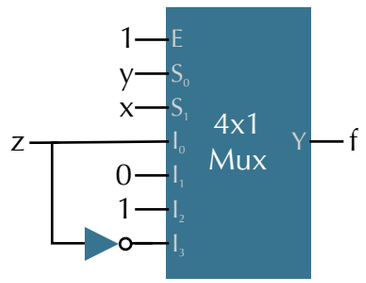
More fun

mux

- connect table row values to mux inputs
- connect input variables to mux select
- mux output is function output



x	y	z	f	
0	0	0	0	I_0
0	0	1	1	
0	1	0	0	I_1
0	1	1	0	
1	0	0	1	I_2
1	0	1	1	
1	1	0	1	I_3
1	1	1	0	



can we use a smaller mux?

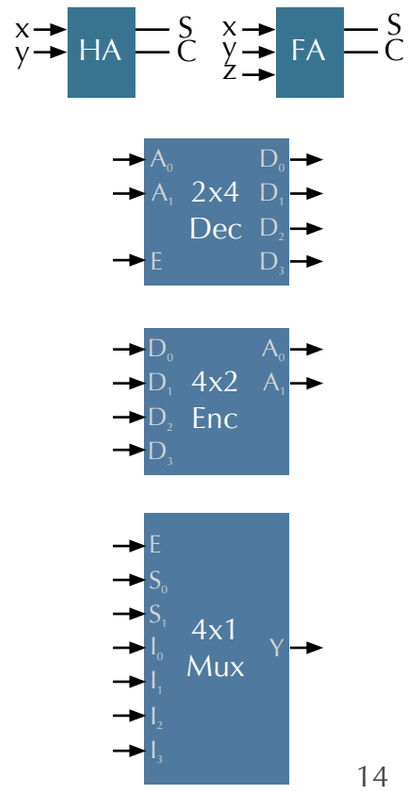
Summary

combinational circuits

- no loops — no state
- basic building block
- describe implementation with
  - boolean algebra
  - truth tables
  - logic diagrams
- abstract as
  - package

combinational circuits you know

- adders: half, full, multi-bit
- decoders, encoders and multiplexers



Flip flops and sequential circuits

Overview

sequential circuits

- adding state to combinational circuits
- clocks

flip-flops

- a state-holding component
- there are several of them: SR,D,JK and T
- excitation tables

designing a sequential circuit

- state tables and diagrams
- some examples

Sequential circuits

output depends on **past** as well as current inputs

- assembled by combining combinational circuit with memory
- memory elements are called *flip flops*

uses

- reduce combinational-circuit propagation delay by pipelining
  - divide circuit into multiple "stages" with flip flops between inputs and outputs
- use internal state as addition inputs
  - just about any interesting problem requires this
    - *e.g., clock, counter, microwave oven*

approaches

- synchronous
  - central clock controls when values are locked into flip-flops 
- asynchronous
  - self timed circuits that do not use a clock

Examples

combinational circuit
time to get one answer = 7 * inverter-gate-delay
answers per second = 1 / (7 * inverter-gate-delay)

pipelined sequential circuit
time to get one answer = 7 * inverter-gate-delay + 8 * flip-flop-gate-delay
answers per second = 1 / (1 * inverter-date-delay + 1 * flip-flop-gate-delay

sequential circuit
one of inputs is output from previous step