
EE108A

Lecture 3: Combinational Building Blocks

1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

1

Announcements

- Read Chapter 10 before Wednesday's Lecture
- Prof. Levis will not have office hours on Friday 1/18
- Homework 1 due today
- Handouts
 - Lecture slides
- Quiz 1 on January 30 (two weeks from today)

1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

2

Review

Lecture 1 – Introduction to digital design:

Representations, noise margins, Boolean algebra, Verilog

Lecture 2 – Combinational logic design

Representations: English, Truth table, Minterm list, Equation, Cubes, K-map, Verilog

K-map minimization: prime implicants, distinguished 1s, coverage

Don't cares

Product-of-sums

Verilog examples

Today's Lecture

- Combinational building blocks – the idioms of digital design
 - Decoder (binary to one-hot)
 - Encoder (one-hot to binary)
 - Multiplexer (select one of N)
 - Arbiter (pick first of N)
 - Comparators
 - Read-only memories

One-hot representation

- Represent a set of N elements with N bits
- Exactly one bit is set
- Example – encode numbers 0-7

Binary	One-hot
000	00000001
001	00000010
010	00000100
...	...
110	01000000
111	10000000

- What operations are simpler with one-hot representation? With binary?

1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

5

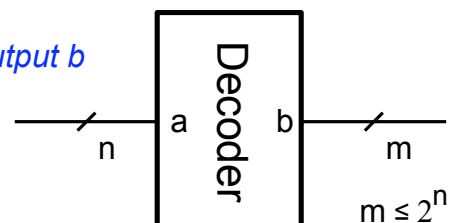
Decoder

- A decoder converts symbols from one code to another.
- A binary to one-hot decoder converts a symbol from binary code to a one-hot code.
 - One-hot code: at most one bit can be high at a time and each bit represents a symbol.

Binary input a to one-hot output b

$$b[i] = 1 \text{ if } a = i$$

$$b = 1 \ll a$$



1/16/2008

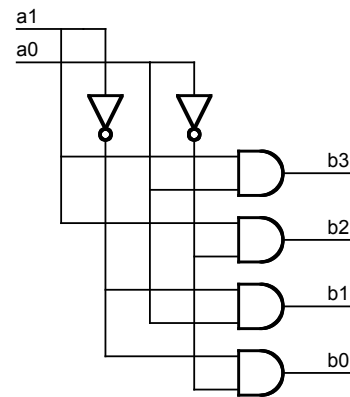
EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

6

Example of a Decoder

2 → 4 Decoder

a1	a0	b3	b2	b1	b0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

7

Can build a large decoder from several small decoders

- Example – build a 6 → 64 encoder from 3 2 → 4 encoders
- Each 2→4 decodes 2 bits
 - $a[1:0] \rightarrow x[3:0]$, $a[3:2] \rightarrow y[3:0]$, $a[5:4] \rightarrow z[3:0]$
- AND one bit each of x, y, and z to generate an output
 - $b[i] = x[i[1:0]] \& y[i[3:2]] \& z[i[5:4]]$
 - Think of each bit of b as a position in a 3-D cube

a5	a4	z3	z2	z1	z0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

a3	a2	y3	y2	y1	y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

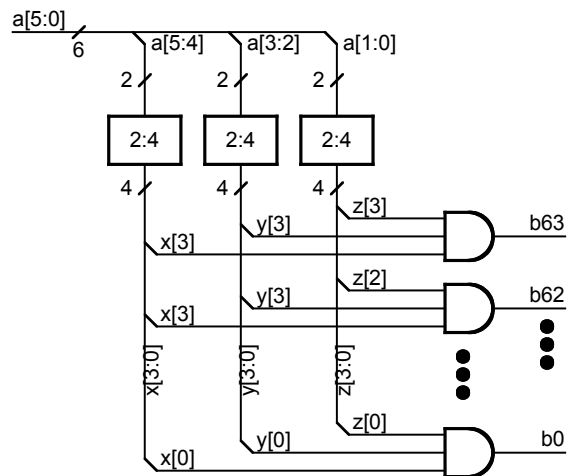
a1	a0	x3	x2	x1	x0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

8

2-Stage decoders – the picture



1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

9

Advantage Of Dividing Large Decoder

- 6->64 decoder requires:
 - 64 6-input AND gates (384 inputs)
- 6->64 decoder using 2->4 decoders requires:
 - 12 2-input AND gates (24 inputs)
 - 64 3-input AND gates (192 inputs)
- Faster, smaller, lower power

1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

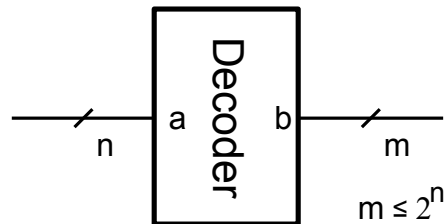
10

Verilog implementation of a decoder

```
// a - binary input    (n bits wide)
// b - one hot output  (m bits wide)
module Dec(a, b) ;
    parameter n=2 ;
    parameter m=4 ;

    input  [n-1:0] a ;
    output [m-1:0] b ;

    wire [m-1:0] b = 1<<a ;
endmodule
```



1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

11

Synthesizes to

```
module dec ( in, out );
    input  [1:0] in;
    output [3:0] out;
    wire n2, n3;
    NO210 U2 ( .A(n2), .B(n3), .Y(out[3]) );
    NO210 U3 ( .A(in[0]), .B(n2), .Y(out[2]) );
    NO210 U4 ( .A(in[1]), .B(n3), .Y(out[1]) );
    NO210 U5 ( .A(in[0]), .B(in[1]), .Y(out[0]) );
    IV110 U6 ( .A(in[1]), .Y(n2) );
    IV110 U7 ( .A(in[0]), .Y(n3) );
endmodule
```

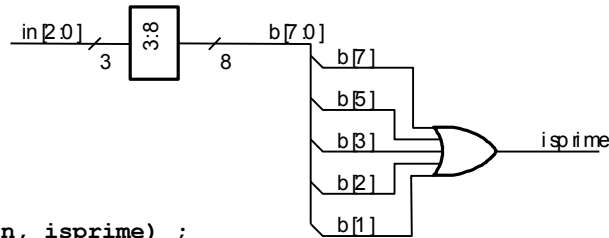
1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

12

Can implement an arbitrary logic function with a decoder

Example – prime number function



```

module Primed(in, isprime) ;
  input [2:0] in ;
  output      isprime ;
  wire [7:0] b ;

  // compute the output as the OR of the required minterms
  wire      isprime = b[1] | b[2] | b[3] | b[5] | b[7] ;

  // instantiate a 3->8 decoder
  Dec #(3,8) d(in,b) ;
endmodule

```

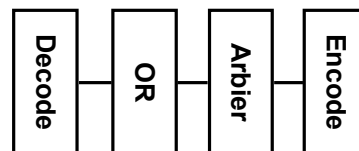
1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

13

Functions built from decoders and other blocks

- Example – find maximum of 256 small 4-bit numbers
 - Decode each number 4→16 bits (256 times)
 - OR these together w/16 256-bit ORs
 - (each a 4-level tree of 4-input ORs)
 - Use an Arbiter to get highest of 16
 - Encode the 16→4



- Need to see if this is more efficient than a tournament
- Can this determine which input has the 'winning' number?

1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

14

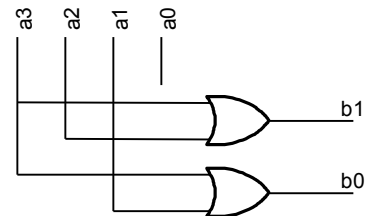
Encoder

- An encoder is an inverse of a decoder.
- Encoder is a logic module that converts a one-hot input signal to a binary-encoded output signal.
- Example: a 4→2 encoder.

a3	a2	a1	a0	b1	b0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

$$b0 = a3 \vee a1$$

$$b1 = a3 \vee a2$$



1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

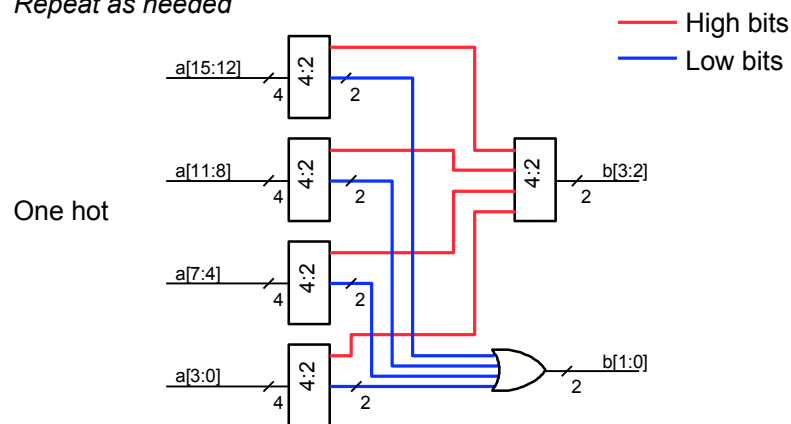
15

Designing A Large Encoder – 16→4 from 5 4→2s

Need an 'any input true' output on first rank of encoders

First rank encodes low bits, second rank encodes high bits

Repeat as needed



1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

16

```

// encoder - fixed width - with summary output
module Enc42a(a, b, c) ;
  input  [3:0] a ;
  output [1:0] b ;
  output c ;
  wire   [1:0] b ;
  wire   c ;          // c is true if any input is true

  assign b[1] = a[3] | a[2] ;
  assign b[0] = a[3] | a[1] ;
  assign c = |a ;
endmodule

// factored encoder
module Enc164(a, b) ;
  input [15:0] a ;
  output[3:0] b ;
  wire [3:0] b ;
  wire [7:0] c ; // intermediate result of first stage
  wire [3:0] d ; // if any set in group of four

  // four LSB encoders each include 4-bits of the input
  Enc42a e0(a[3:0], c[1:0],d[0]) ;
  Enc42a e1(a[7:4], c[3:2],d[1]) ;
  Enc42a e2(a[11:8], c[5:4],d[2]) ;
  Enc42a e3(a[15:12],c[7:6],d[3]) ;

  // MSB encoder takes summaries and gives msb of output
  Enc42 e4(d[3:0], b[3:2]) ;

  // two OR gates combine output of LSB encoders
  assign b[1] = c[1]|c[3]|c[5]|c[7] ;
  assign b[0] = c[0]|c[2]|c[4]|c[6] ;
endmodule

```

```

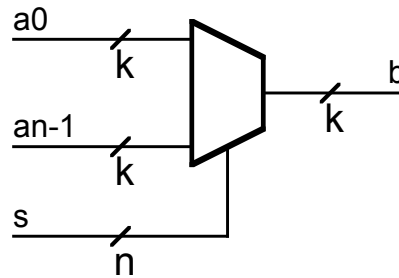
# 0000000000000001 0000
# 0000000000000010 0001
# 0000000000000100 0010
# 0000000000001000 0011
# 0000000000010000 0100
# 0000000000100000 0101
# 0000000001000000 0110
# 0000000010000000 0111
# 0000000100000000 1000
# 0000001000000000 1001
# 0000010000000000 1010
# 0000100000000000 1011
# 0001000000000000 1100
# 0010000000000000 1101
# 0100000000000000 1110
# 1000000000000000 1111
# 0000000000000000 0000

```

Multiplexer

- Multiplexer:
 - n k-bit inputs
 - n-bit one-hot select signal s
 - Multiplexers are commonly used as *data selectors*

Selects one of n k-bit inputs
 s must be one-hot
 $b = a[i]$ if $s[i] = 1$

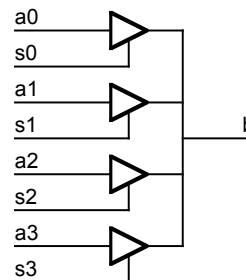
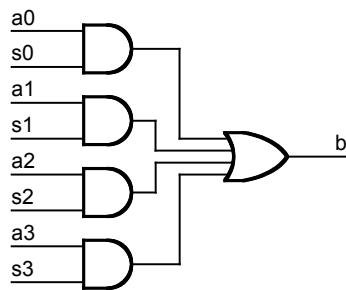


1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

19

Multiplexer Implementation



1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

20

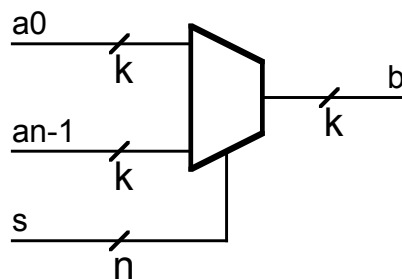
```
// four input k-wide mux with one-hot select
module Mux4(a3, a2, a1, a0, s, b) ;
  parameter k = 1 ;
  input [k-1:0] a0, a1, a2, a3 ; // inputs
  input [3:0] s ; // one-hot select
  output[k-1:0] b ;
  wire [k-1:0] b = ({k{s[0]}} & a0) |
                   ({k{s[1]}} & a1) |
                   ({k{s[2]}} & a2) |
                   ({k{s[3]}} & a3) ;

endmodule

Mux4 #(2) mx(2'd3, 2'd2, 2'd1, 2'd0, f, h) ;

      f      h
# 0001 00
# 0010 01
# 0100 10
# 1000 11
```

k-bit Binary-Select Multiplexer

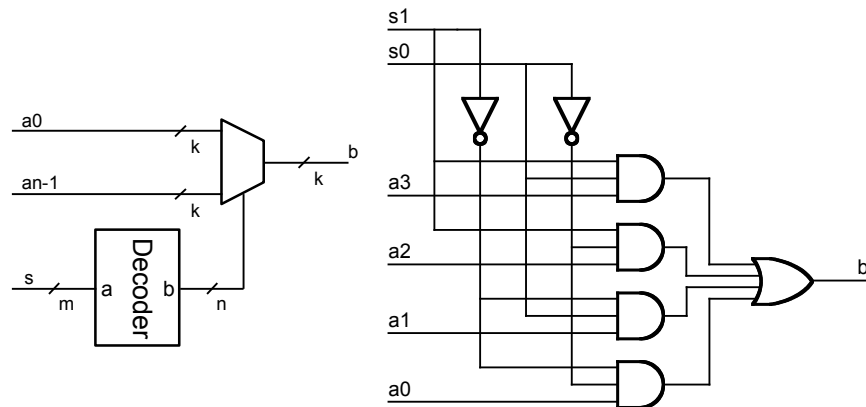


Selects one of n k-bit inputs

s must be one-hot

$b = a[i]$ if $s[i] = 1$

k-bit Binary-Select Multiplexer (Cont)



1/16/2008

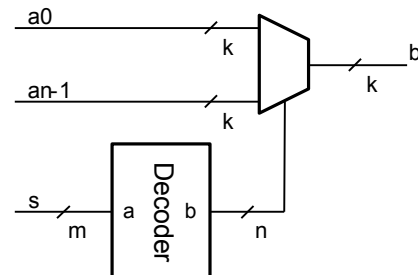
EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

23

Implementing k-bit Binary-Select Multiplexer Using Verilog

```
// 3:1 multiplexer with binary select (arbitrary width)
module Muxb3(a2, a1, a0, sb, b) ;
  parameter k = 1 ;
  input [k-1:0] a0, a1, a2 ; // inputs
  input [1:0] sb ; // binary select
  output[k-1:0] b ;
  wire [2:0] s ;

  Dec #(2,3) d(sb,s) ; // Decoder converts binary to one-hot
  Mux3 #(k) m(a2, a1, a0, s, b) ; // multiplexer selects input
endmodule
```

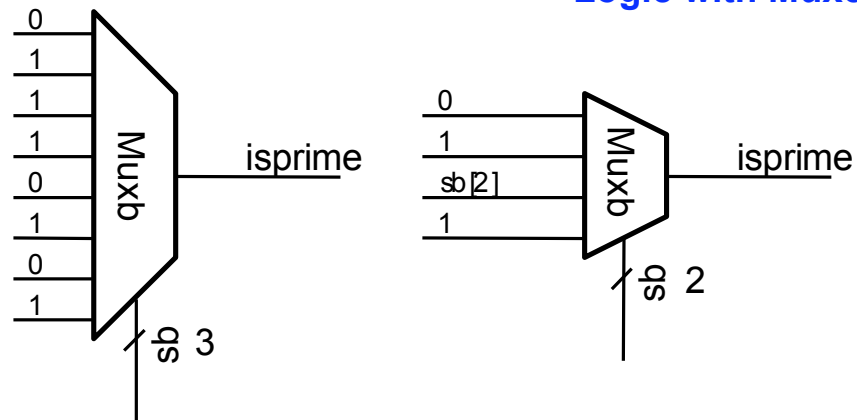


1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

24

Logic with Muxes



```

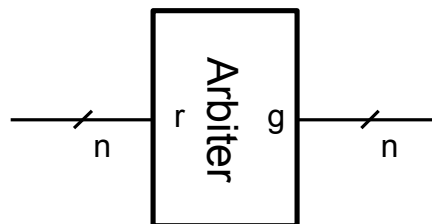
module Primem(in, isprime) ;
    input [2:0] in ;
    output      isprime ;

    Muxb8 #(1) m(1, 0, 1, 0, 1, 1, 1, 0, in, isprime) ;
endmodule

```

Arbiter

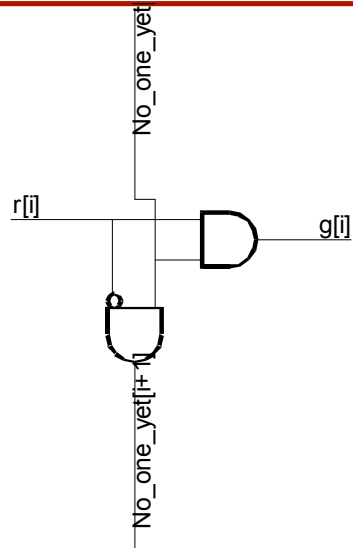
Arbiter handles requests from multiple devices to use a single resource



Finds first "1" bit in r

$g[i]=1$ if $r[i]=1$ and $r[j]=0$ for $j \geq i$

Logic Diagram Of One Bit Of An Arbiter

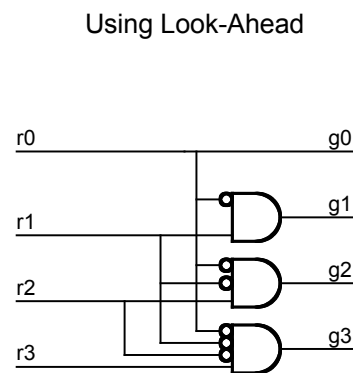
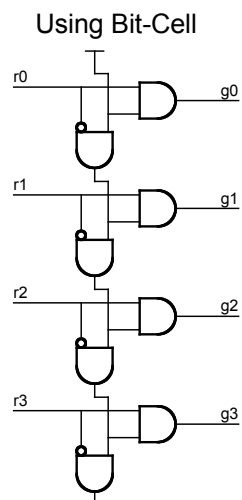


1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

27

Two Implementations Of A 4-Bit Arbiter



1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

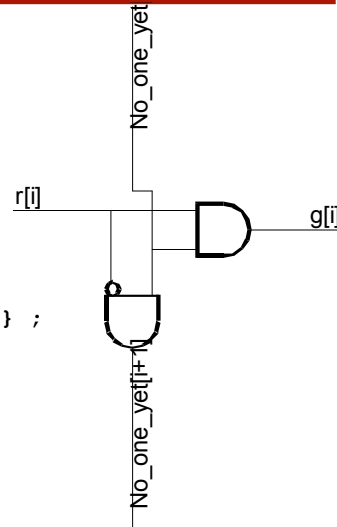
28

Implementing Arbitrary Width Arbiter Using Verilog

```
// arbiter (arbitrary width)
module Arb(r, g) ;
  parameter n=8 ;
  input  [n-1:0] r ;
  output [n-1:0] g ;
  wire  [n-1:0] c ;
  wire  [n-1:0] g ;

  assign c = {(~r[n-2:0] & c[n-2:0]), 1'b1} ;
  assign g = r & c ;
endmodule
```

Bit-slice coding style using
concatenation {a, b}
index ranges c[n-2:0]
c is 1s up to first 1 in r, then 0s



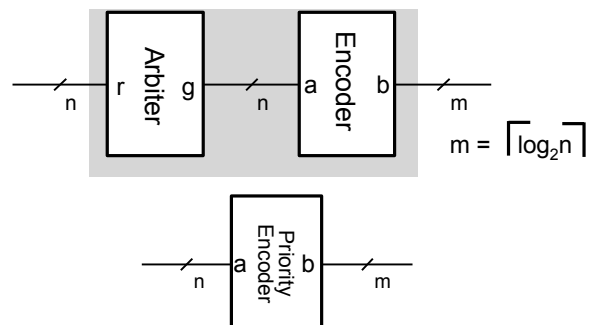
1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

29

Priority Encoder

- Priority Encoder:
 - n-bit input signal **a**
 - m-bit output signal **b**
 - b indicates the position of the first 1 bit in **a**



1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

30

Verilog for Priority Encoder

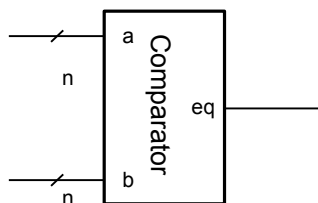
```
// priority encoder (arbitrary width)
module PriorityEncoder83(r, b) ;
  input  [7:0] r ;
  output [2:0] b ;
  wire  [7:0] g ;
  Arb #(8) a(r, g) ;
  Enc83    e(g, b) ;
endmodule
```

1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

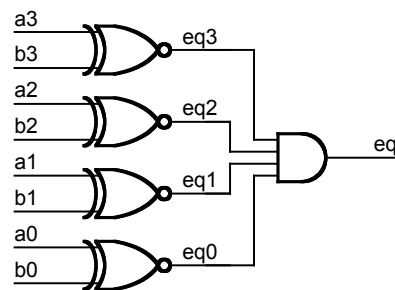
31

Equality Comparator



```
// equality comparator
module EqComp(a, b, eq) ;
  parameter k=8;
  input  [k-1:0] a,b;
  output eq ;
  wire  eq;

  assign eq = (a==b) ;
endmodule
```

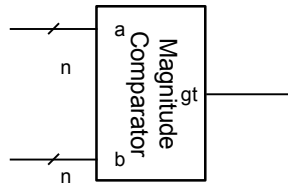


1/16/2008

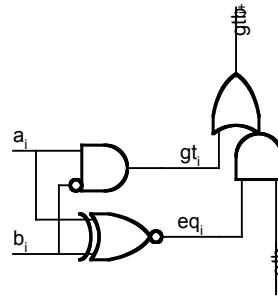
EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

32

Magnitude Comparator



```
// magnitude comparator
module MagComp(a, b, gt) ;
  parameter k=8 ;
  input  [k-1:0] a, b ;
  output gt ;
  wire   [k-1:0] eqi = a ^ b ;
  wire   [k-1:0] gti = a & ~b ;
  wire   [k:0]   gtb {((eqi[k-1:0] & gtb[k-1:0]) | gti[k-1:0]), 1'b0} ;
  wire   gt = gtb[k] ;
endmodule
```

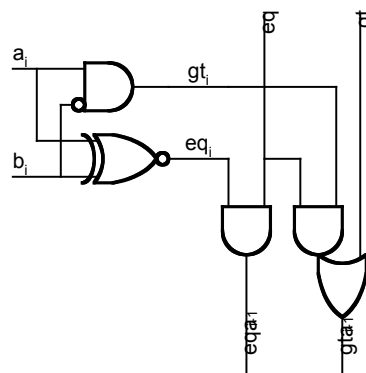


1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

33

Another Implementation Of Magnitude Comparator

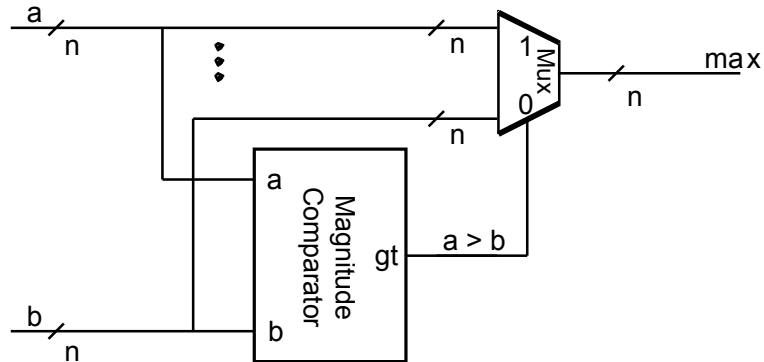


1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

34

Putting things together – Maximum unit

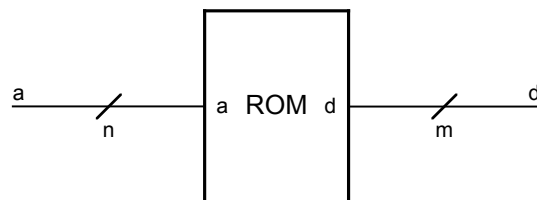


1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

35

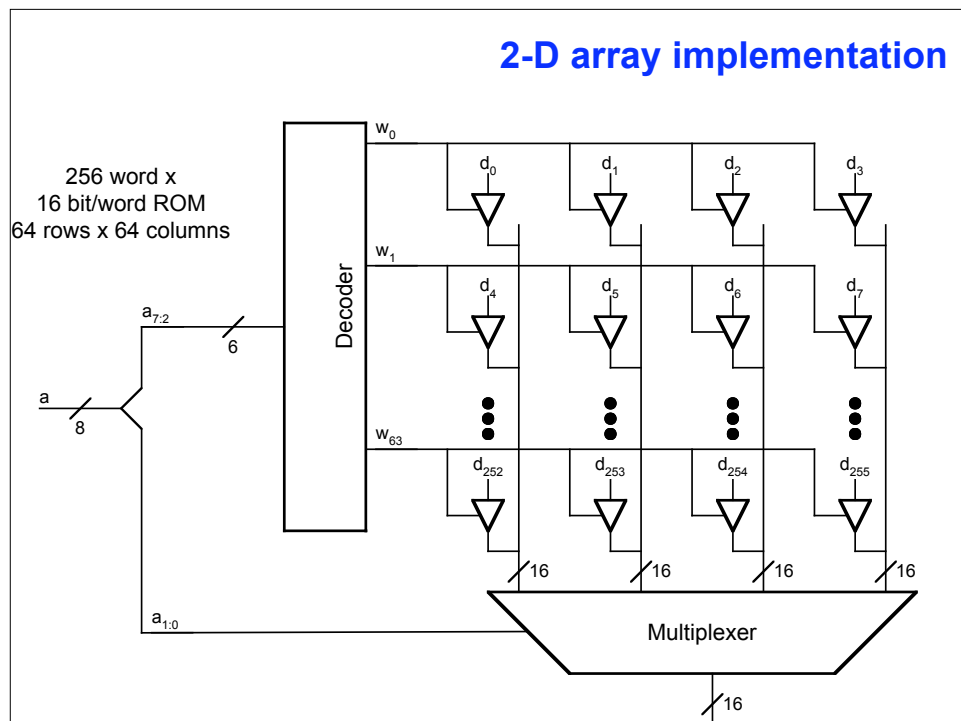
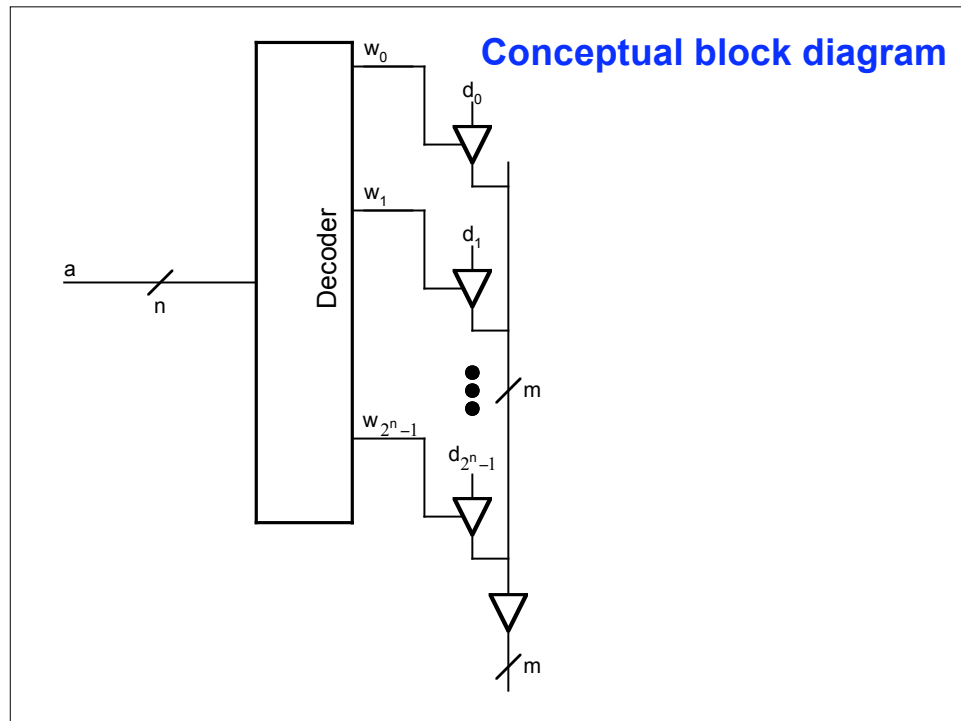
Read-only memory (ROM)



1/16/2008

EE 108A Lecture 3 (c) 2005-8 W. J. Dally and P. Levis

36



Summary

- Assemble combinational circuits from pre-defined building blocks
 - Decoder – converts codes (e.g., binary to one-hot)
 - Encoder – encodes one-hot to binary
 - Multiplexer – select an input (one-hot select)
 - Arbiter – pick first true bit
 - Comparators – equality and magnitude
 - ROMs
- Divide and conquer to build large units from small units
 - Decoder, encoder, multiplexer
- Logic with multiplexers or decoders
- Bit-slice coding style