(T)EE2026

Digital Fundamentals

Combinational Building Blocks and Structural/Dataflow Verilog Description

Prof. Massimo Alioto

Dept of Electrical and Computer Engineering

Email: ***massimo.alioto@nus.edu.sg***

Outline

- Introduction
- Binary adders
  - Half adders, full adders, ripple adders.
- Magnitude comparators
- Decoders, BCD to 7-segment decoders
- Encoders, Multiplexers
- Demultiplexers
- Tri-state logic elements

Introduction

- There are two types of logic circuits
  - Combinational and sequential logic circuits

- Combinational logic
  - The output depends only on the current inputs
- Sequential logic
  - The output depends on both past and present inputs, which implies that there is a memory element in the sequential circuit

- Combinational building blocks that are commonly used in digital systems

Half Adders

- It is a one bit binary adder with two inputs of A_i and B_i

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10

Carry → C_{i+1}

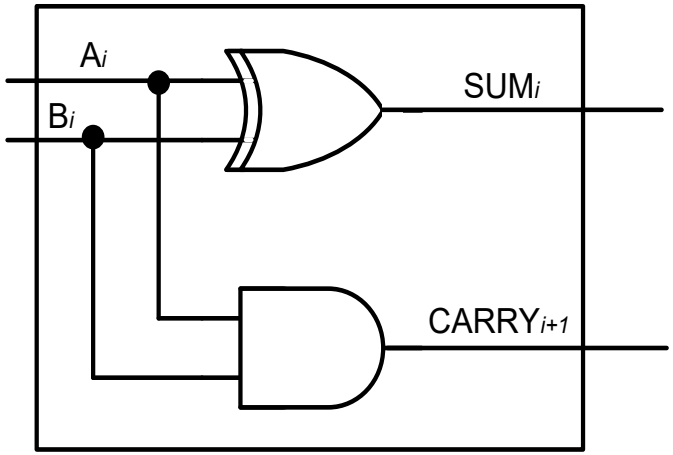
$A : A_n \; \text{K} \; A_{i+1} \; A_i \; \text{K} \; A_0$

$B : B_n \; \text{K} \; B_{i+1} \; B_i \; \text{K} \; B_0$

Sum → S_i

A_i	B_i	Sum_i	$Carry_{i+1}$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Carry in from *i-1* bit cannot be added



$$S_i = \bar{A}_i \cdot B_i + A_i \cdot \bar{B}_i \cdot$$

$$C_{i+1} = A_i \cdot B_i$$

Half Adders (cont.)

- Dataflow Verilog description

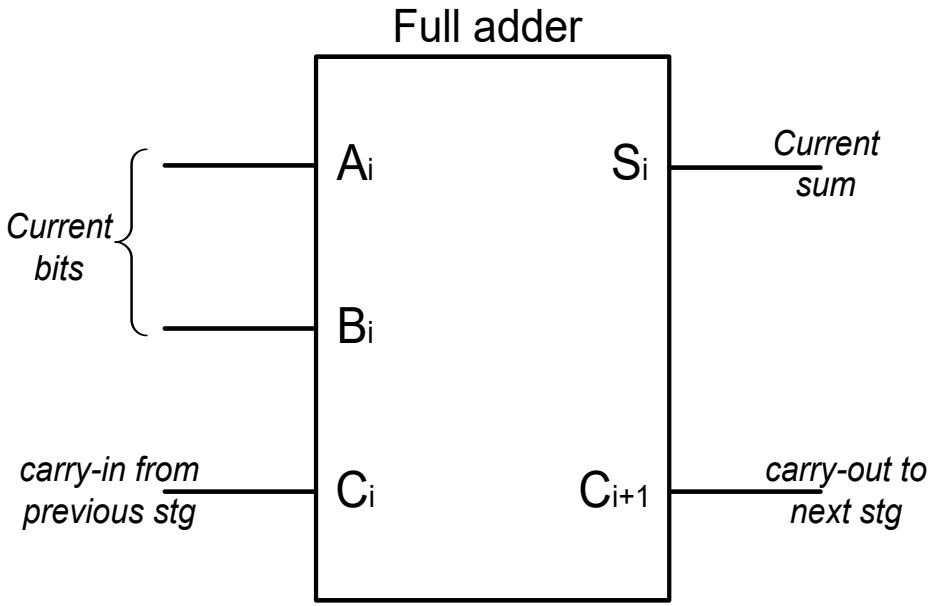
$$S_i = A_i \oplus B_i$$
$$C_{i+1} = A_i \cdot B_i$$

A_i	B_i	Sum_i	$Carry_{i+1}$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

```
module ha(S,Cout,A,B);
    input A, B;
    output S, Cout; // Cout is the carry output
    assign S = A ^ B;
    assign Cout = A & B;
endmodule
```

Full Adders

- Full adders can use the carry bit from the previous stage of addition



A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Full Adders (cont.)

K-map for SUM

A_i / B_iC_i	0	1
00	0	1
01	1	0
11	0	1
10	1	0

Note: C_{i+1} is not a MSOP, but less overall hardware is reqd. if we use this expression. It allows sharing of A_i XOR B_i between SUM_i and C_{i+1}.

K-map for CARRY

A_i / B_iC_i	0	1
00	0	0
01	0	1
11	1	1
10	0	1

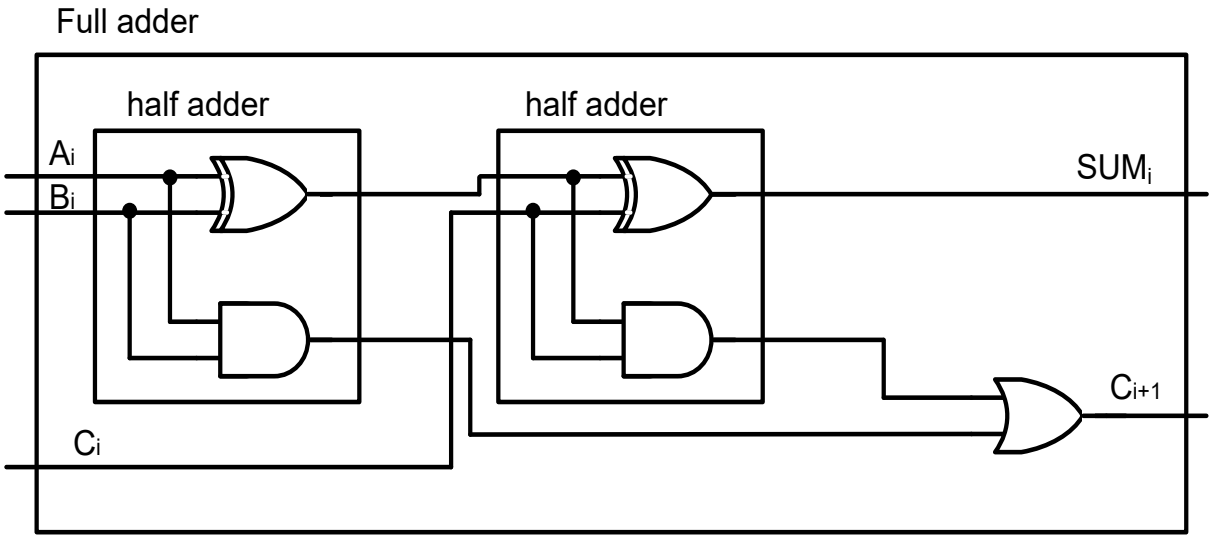
$$\begin{aligned} SUM &= \bar{A}_i\bar{B}_iC_i + \bar{A}_iB_i\bar{C}_i + A_i\bar{B}_i\bar{C}_i + A_iB_iC_i \\ &= \bar{A}_i(\bar{B}_iC_i + B_i\bar{C}_i) + A_i(\bar{B}_i\bar{C}_i + B_iC_i) \\ &= \bar{A}_i(B_i \oplus C_i) + A_i(\overline{B_i \oplus C_i}) \\ &= A_i \oplus B_i \oplus C_i \end{aligned}$$

$$\begin{aligned} C_{i+1} &= A_iB_i + A_i\bar{B}_iC_i + \bar{A}_iB_iC_i \\ &= A_iB_i + C_i(A_i\bar{B}_i + \bar{A}_iB_i) \\ &= A_iB_i + C_i(A_i \oplus B_i) \end{aligned}$$

Full Adder Circuit

$$SUM = (A_i \oplus B_i) \oplus C_i$$

$$C_{i+1} = A_i B_i + C_i (A_i \oplus B_i)$$



Note: A full adder adds 3 bits. Can also consider as first adding first two and then the result with the carry

Full Adder Circuit

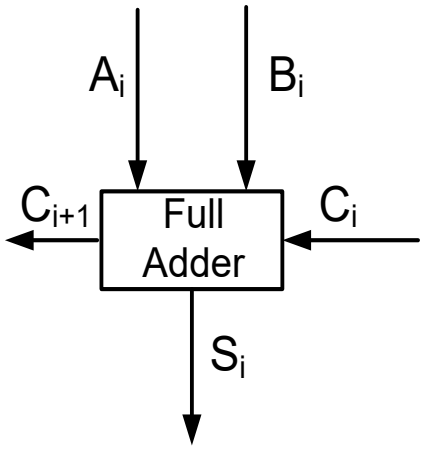
- Verilog dataflow description

$$SUM = (A_i \oplus B_i) \oplus C_i$$

$$C_{i+1} = A_i B_i + C_i (A_i \oplus B_i)$$

```
module fa(S,Cout,A,B,Cin);
      input A, B, Cin; // Cin is the carry input
      output S, Cout; // Cout is the carry output
      assign S = A ^ B ^ Cin;
      assign Cout = A & B | Cin & (A ^ B);
endmodule
```

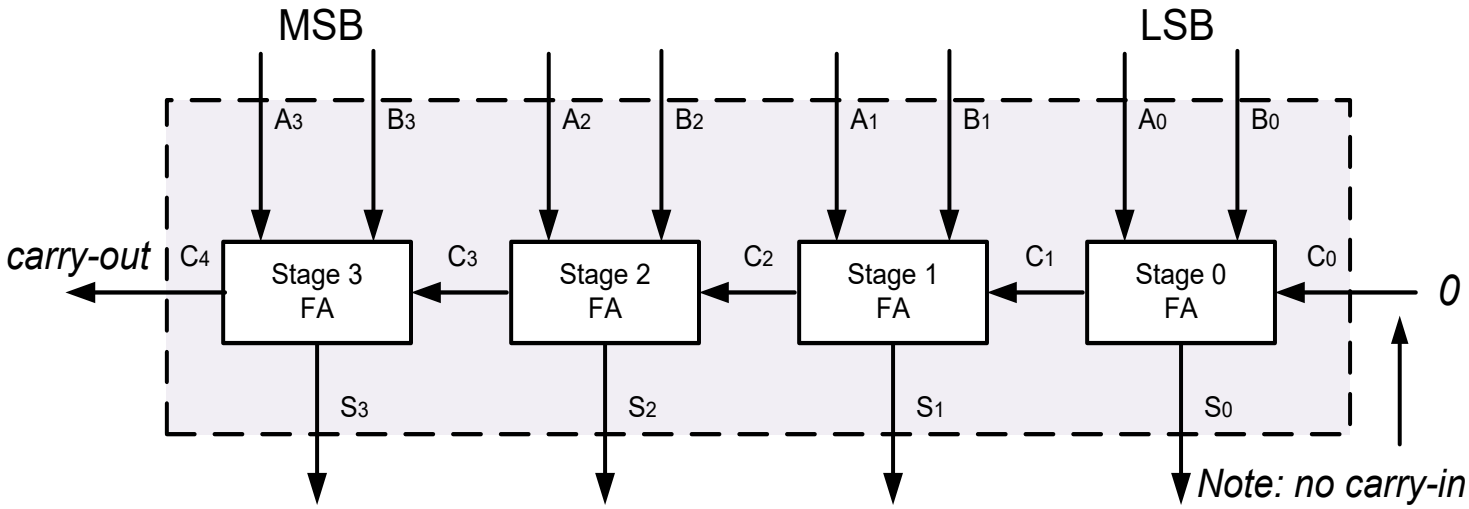
Parallel Adders



C_4 C_3 C_2 C_1

A_3 A_2 A_1 A_0

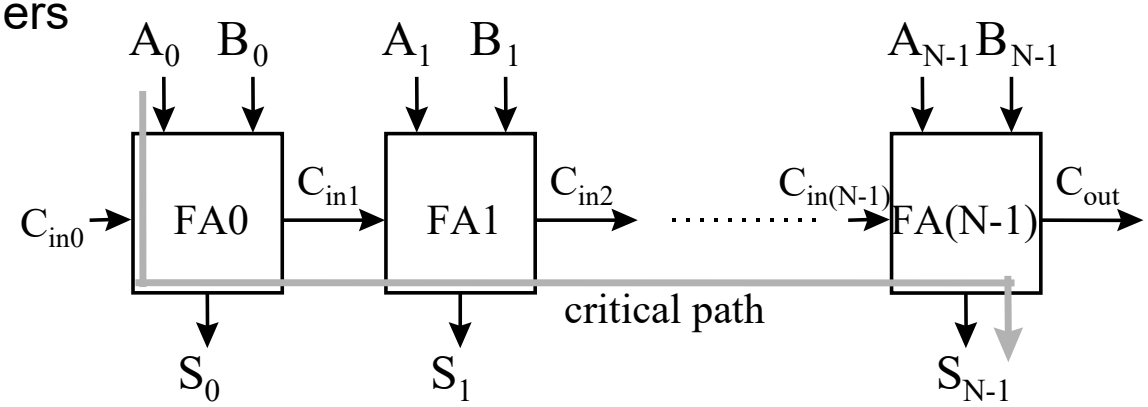
B_3 B_2 B_1 B_0



4 cascaded full adders

Parallel Adders (cont.)

- In general, *n* full adders need to be used to form an *n*-bit adder
- Carry ripple effect
  - output of each full adder is not available until the carry-in from the previous stage is delivered
  - carry bits have to propagate from one stage to the next
  - as the carries *ripple* through the carry chain → also known as *ripple carry* adders



- This slow rippling effect is substantially reduced by using *carry look ahead adders*

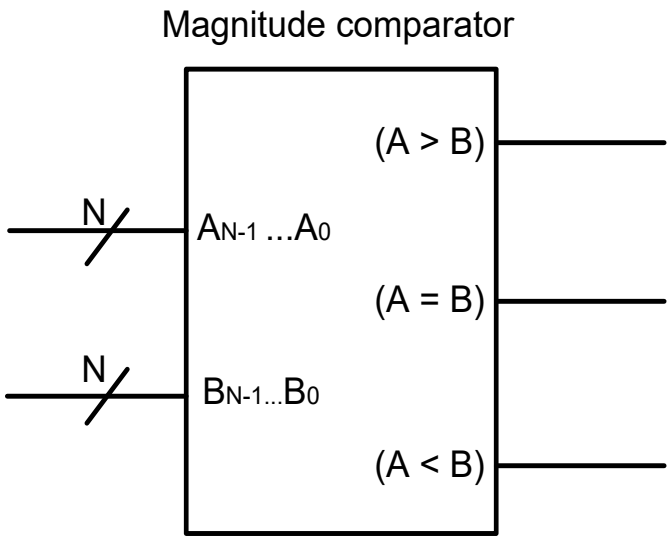
Parallel Adders (cont.)

- Structural Verilog description (parameterized, arbitrary bit width)

```
module rca(S,Cout,A,B,Cin); // 4-bit ripple carry adder
    parameter N = 4; // parameterized bit width

    input [N-1:0] A, B;
    input Cin; // Cin is the adder carry input (at LSB)
    output [N-1:0] S;
    output Cout; // Cout is the adder carry output (at MSB)
    wire [N:0] C; // carry inputs of all full adders + carry output of last one

    assign C[0] = Cin;
    assign Cout = C[N];

    genvar i; // temp variable used only in generate loop
        generate for(i=0;i<N;i=i+1) begin
            fa FAinstance (.S(S[i]),.Cout(C[i+1]),.A(A[i]),.B(B[i]),.Cin(C[i]));
            end
        endgenerate
endmodule
```

Magnitude Comparator

- Outputs are functions of relative magnitudes of input binary numbers A and B



Functional block diagram

Magnitude Comparator: Truth Table

2-bit magnitude comparator

A_1	A_0	B_1	B_0	(A > B)	(A = B)	(A < B)
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	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	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

K-maps for A>B and A<B

A>B

A_1A_0 / B_1B_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$$(A > B) = A_1\overline{B}_1 + A_0\overline{B}_1\overline{B}_0 + A_1A_0\overline{B}_0$$

A<B

A_1A_0 / B_1B_0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$$(A < B) = \overline{A}_1B_1 + \overline{A}_1\overline{A}_0B_0 + \overline{A}_0B_1B_0$$

K-map for A=B

A=B

B_1B_0 \ A_1A_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

$$(A=B) = \overline{A}_1\overline{A}_0\overline{B}_1\overline{B}_0 + \overline{A}_1A_0\overline{B}_1B_0 + A_1A_0B_1B_0 + A_1\overline{A}_0B_1\overline{B}_0$$

This can be generated indirectly using (A<B) and (A>B)

$$(A=B) = \overline{(A<B)} \cdot \overline{(A>B)}$$

Magnitude Comparator: Verilog

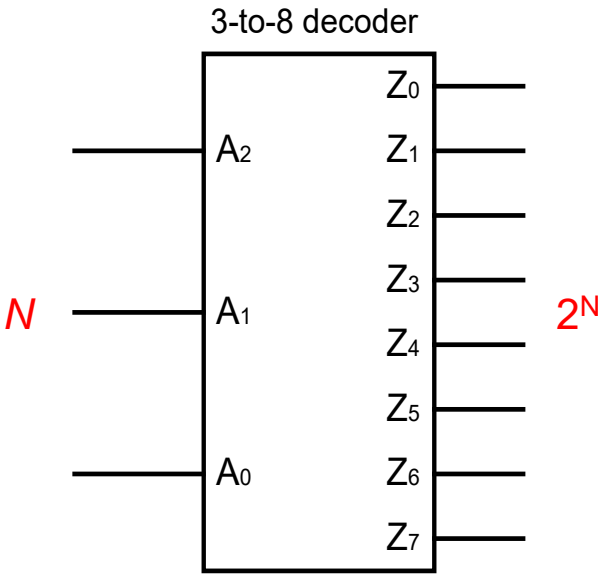
- Dataflow Verilog description (parameterized, arbitrary bit width)

```
module magcomp(AgreaterB,AequalB,AlowerB,A,B);
      parameter N = 4;

      input [N-1:0] A, B;
      output AgreaterB, AequalB, AlowerB;

      assign AgreaterB = (A > B);
      assign AequalB = (A == B);
      assign AlowerB = (A < B);
      /*  to reduce complexity at the cost of slightly worse performance: assign
          AlowerB = ~AgreaterB & ~AequalB */
endmodule
```

Decoder

- Input: N-bit input code
- A decoder activates a (single) appropriate output line among M (more than N, usually 2^N) as a function of the input

Functional block diagram

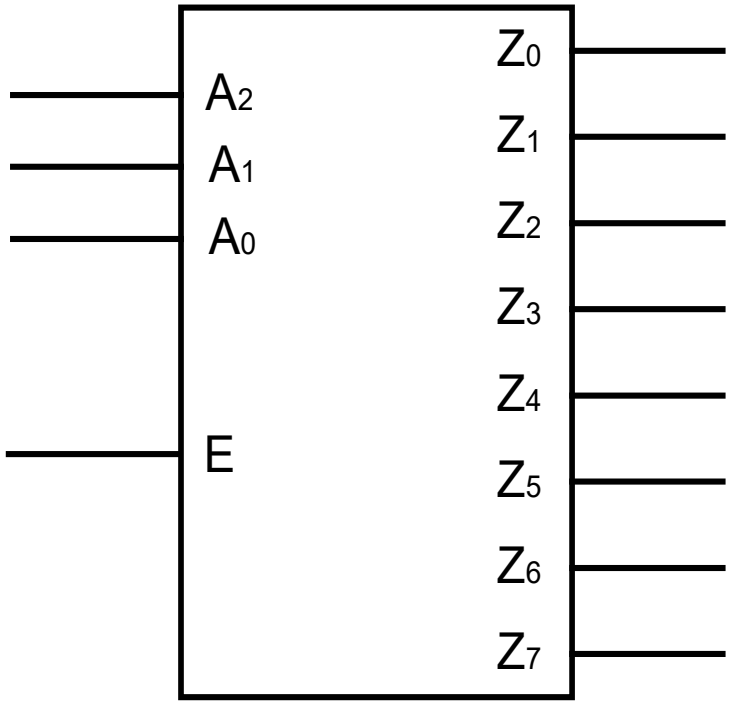


Truth Table

A_2	A_1	A_0	Z_0	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Example: Decoder 3-8

- A decoder has 2^N output lines for N inputs
  - named N-2^N decoder
  - output can be single- or multi-bit
- Enable signal
  - if E = 1, normal operation
  - if E= 0, disable outputs (all 0's)
  - N enables permit to combine multiple decoders (see videolecture on MSIs)



Functional block diagram

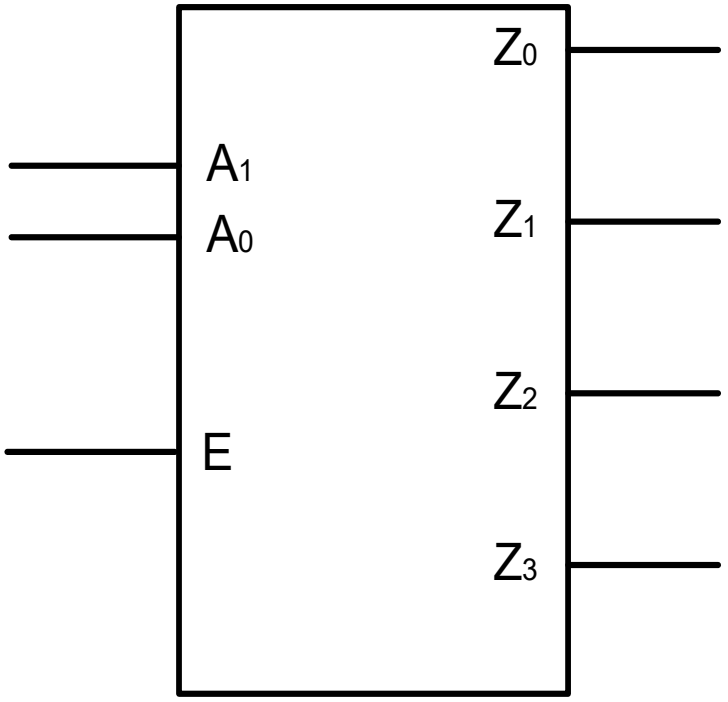
Decoder with Enable Signal

Truth Table including Enable signal (single-bit output)

Inputs				Outputs							
E	A_2	A_1	A_0	Z_0	Z_1	Z_2	Z_3	Z_4	Z_5	Z_6	Z_7
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

Example: Decoder 2-4

- A 2:4 decoder has 2^2 output lines for *N* inputs
  - output can be single- or multi-bit
- Enable signal
  - if E = 1, normal operation
  - if E= 0, disable outputs (all 0's)



Functional block diagram

Decoder with Enable Signal

Truth Table including Enable signal (single-bit output)

Inputs			Outputs			
E	A_1	A_0	Z_0	Z_1	Z_2	Z_3
0	X	X	0	0	0	0
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

Decoder: Verilog

- Dataflow Verilog description of a 2:4 decoder

```
module decoder24(Z,A,E);
      input [1:0] A;
      input E;
      output [0:3] Z;

      assign Z = ((A == 2'b00) & E) ? 4'b1000 :
                 ((A == 2'b01) & E) ? 4'b0100 :
                 ((A == 2'b10) & E) ? 4'b0010 :
                 ((A == 2'b11) & E) ? 4'b0001 :
                 4'b0000;
                 // 0000 is assigned if E=0
endmodule
```

Inputs			Outputs			
E	A_1	A_0	Z_0	Z_1	Z_2	Z_3
0	X	X	0	0	0	0
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

Decoder: Verilog

- Dataflow Verilog description of a 2:4 decoder

```
module decoder24(Z,A,E);
      input [1:0] A;
      input E;
      output [0:3] Z;

      assign Z = ((A == 2'b00) & E) ? 4'b1000 :
                 ((A == 2'b01) & E) ? 4'b0100 :
                 ((A == 2'b10) & E) ? 4'b0010 :
                 ((A == 2'b11) & E) ? 4'b0001 :
                 4'b0000; // 0000 if E=0
endmodule
```

Inputs			Outputs			
E	A_1	A_0	Z_0	Z_1	Z_2	Z_3
0	X	X	0	0	0	0
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

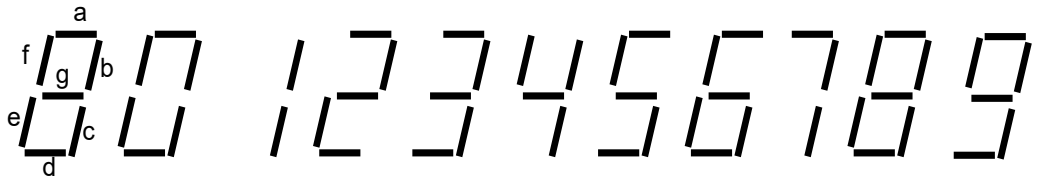
Decoder: Verilog

- Parameterized dataflow Verilog description (arbitrary bit width)

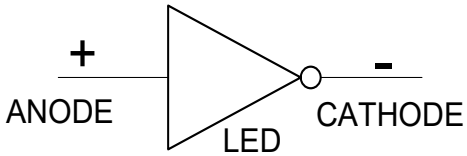
```
module decoder(Z,A,E);
    parameter M = 4; // parameterized design (sets # inputs)
    parameter N = 16; // parameterized design (sets # outputs=2^M)

    input [M-1:0] A;
    input E;
    output [N-1:0] Z;

    wire [N-1:0] zerovec = {N{1'b0}}; // replication operator N(.)
    /* other option: define constant zerovec=1'b0, it will be extended to the left
        with zeros to the correct bit width */

    assign Z = (enable) ? (1 << A) : zerovec;
    // if enable=0, output is set to to zerovec = 00...0
    // if enable=1, shift “1” A times and fill all other positions with zeros
endmodule
```

Example: BCD-to-7 Segment Decoder

- Converts a BCD number into signals required to display that number on a 7-segment display

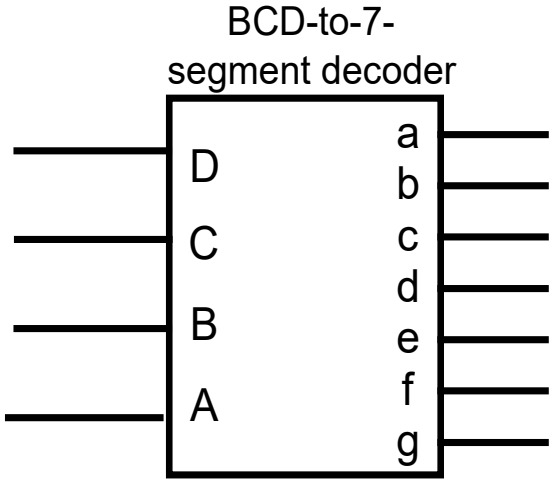


A 7-segment display. Each segment is an LED which will light when a logic T signal is applied to it

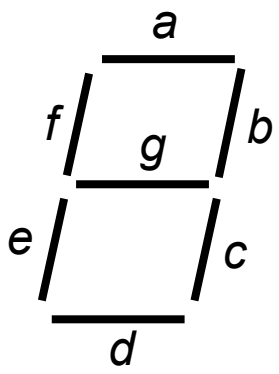


- 7-segment displays are of 2 types: *common anode* and *common cathode*
- Common anode display has all LED anodes connected and is active low, whereas the common cathode display is active high

BCD-to-7 Segment Decoder – cont.



Functional block diagram



Truth Table

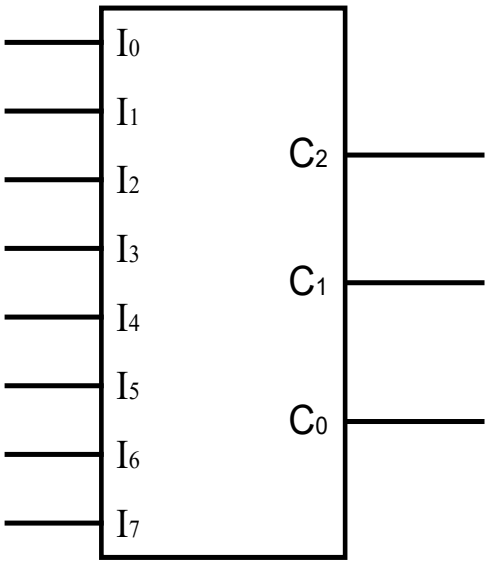
D	C	B	A	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
1	0	1	0	X	X	X	X	X	X	X
1	0	1	1	X	X	X	X	X	X	X
1	1	0	0	X	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X	X

- Verilog: essentially, description of a truth table
  - **no clever dataflow Verilog description** (behavioral, see later)

Encoder

- For different input bits (usually 2^N), encoder generates a code with fewer bits (usually N bits) uniquely identifying the input
  - performs the inverse of the decoding function

Functional block diagram



Truth Table (an 8-3 encoder)

I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	C_2	C_1	C_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

- **No clever dataflow Verilog description** (behavioral, see later)

Example: Priority Encoder

- Generic encoders: error flagged if multiple input bits are 1
- Priority encoder allows multiple input bits to be 1
  - output set by the input bit with highest priority (i.e., most significant position), ignoring those with lower priority

I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	C_2	C_1	C_0
1	0	0	0	0	0	0	0	0	0	0
X	1	0	0	0	0	0	0	0	0	1
X	X	1	0	0	0	0	0	0	1	0
X	X	X	1	0	0	0	0	0	1	1
X	X	X	X	1	0	0	0	1	0	0
X	X	X	X	X	1	0	0	1	0	1
X	X	X	X	X	X	1	0	1	1	0
X	X	X	X	X	X	X	1	1	1	1

Example: Priority Encoder

- Dataflow Verilog description of 4-2 priority decoder

I_0	I_1	I_2	I_3	C_1	C_0
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

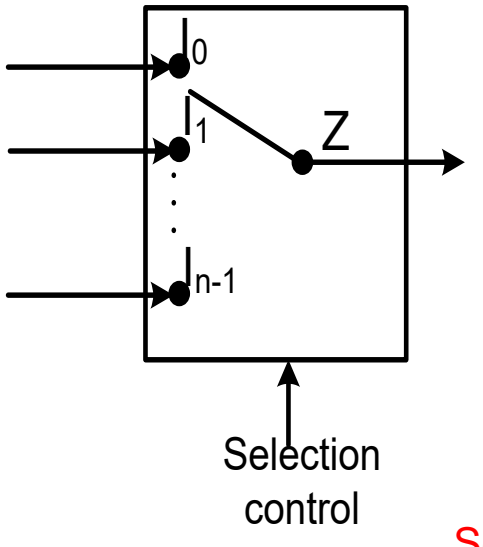
- Use nested conditional operators, starting from MSB and progressively moving to the LSB

```
module priorityencoder(C,I);
      input [3:0] I;
      output [1:0] C;
      assign C = (I[3]) ? (2'b11) : (I[2] ? (2'b10) : (I[1] ? (2'b01) : (2'b00)) );
                     // if I[3]=1, C=11
                    // else, if I[2]=1, C=10, etc.
endmodule
```

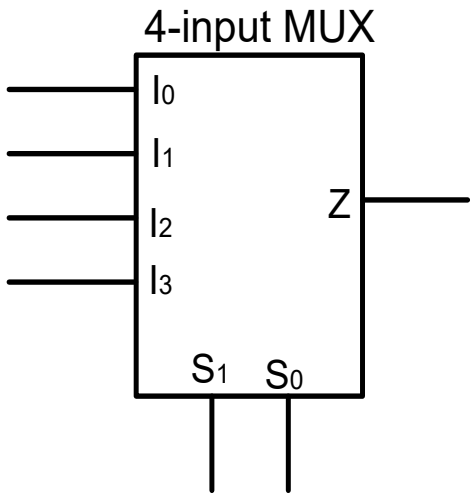
- **No clever parameterized dataflow Verilog description**

Multiplexer

- A multiplexer (MUX) is a combinational circuit element that selects data from one of 2^N inputs and directs it to a single output, according to an N-bit selection signal
  - inputs/outputs can be 1 or M>1 bit wide
  - examples with 1-bit inputs/output



Functional block diagram



Selection inputs allow one of the inputs to pass through to the output

Condensed truth table

S_1	S_0	Z
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Actual truth table would have 2^6 rows corresponding to I_0, I_1, I_2, I_3, S_0 and S_1

Example: 4:1 MUX

- Sometimes include enable input signal

$$Z = E \cdot (\overline{S_0}\,\overline{S_1} I_0 + S_0 \overline{S_1} I_1 + \overline{S_0} S_1 I_2 + S_0 S_1 I_3)$$

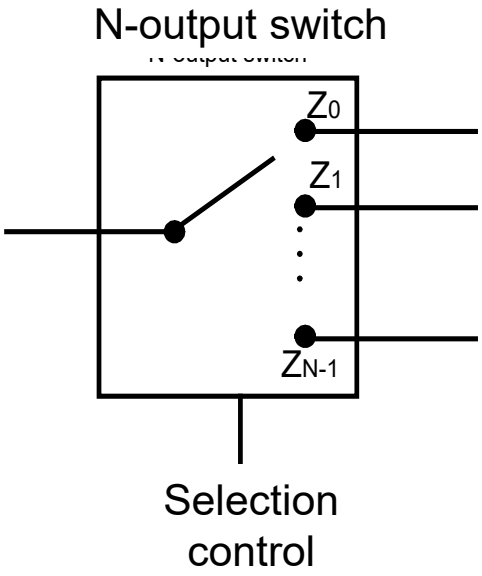
- M-bit inputs/output: use M 1-bit MUXes

E	S_1	S_0	I_0	I_1	I_2	I_3	Z
0	X	X	X	X	X	X	0
1	0	0	0	X	X	X	0
1	0	0	1	X	X	X	1
1	0	1	X	0	X	X	0
1	0	1	X	1	X	X	1
1	1	0	X	X	0	X	0
1	1	0	X	X	1	X	1
1	1	1	X	X	X	0	0
1	1	1	X	X	X	1	1

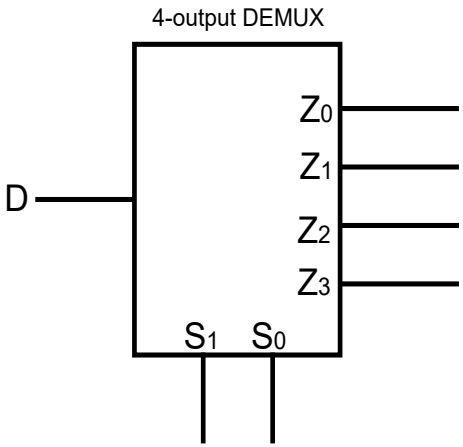
```
module mux41(Z,S,I0,I1,I2,I3,E);
    parameter M = 16; // 16-bit inputs and outputs
    input [M-1:0] I0, I1, I2, I3; // inputs
    input [1:0] S; // 2-bit selection signal
    input E; // enable
    output [M-1:0] Z;
    assign Z = E ? (S[1] ? (S[0] ? I3 : I2) : (S[0] ? I1 : I0)) : 0;
endmodule
```

Demultiplexer

- A Demultiplexer (DEMUX) connects an input signal to any of 2^N output lines, based on an N-bit selection control
  - inputs/outputs can be 1 or M>1 bit wide
  - examples with 1-bit inputs/output



Functional block diagram of 4-output demux



Truth table

D	S_1	S_0	Z_0	Z_1	Z_2	Z_3
0	X	X	0	0	0	0
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

Example: 1:4 DEMUX

- Boolean expression of output

$$Z_0 = D \cdot \overline{S_0} \cdot \overline{S_1}$$
$$Z_1 = D \cdot S_0 \cdot \overline{S_1}$$
$$Z_2 = D \cdot \overline{S_0} \cdot S_1$$
$$Z_3 = D \cdot S_0 \cdot S_1$$

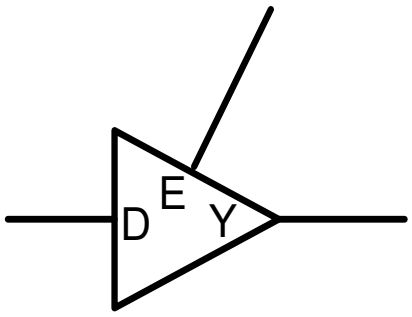
D	S_1	S_0	Z_0	Z_1	Z_2	Z_3
0	X	X	0	0	0	0
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

```
module demux41(Z0,Z1,Z2,Z3,S,D);
    parameter M = 16; // 16-bit inputs and outputs
    input [M-1:0] D; // input
    input [1:0] S; // 2-bit selection signal
    output [M-1:0] Z0, Z1, Z2, Z3;
    assign Z0 = (S == 2'b00) ? D : 1'b0; // zeros extended to other bits of Z0
    assign Z1 = (S == 2'b01) ? D : 1'b0; // zeros extended to other bits of Z0
    assign Z2 = (S == 2'b10) ? D : 1'b0; // zeros extended to other bits of Z0
    assign Z3 = (S == 2'b11) ? D : 1'b0; // zeros extended to other bits of Z0
endmodule
```

Tri-State Logic Elements

- Ordinarily, a digital device has 2 states
  - tri-state devices also have *high impedance state* (Z)
    - floating output: the device does not force any voltage
    - voltage set by the output of some other device
    - if only one device is enabled at a time (all others in Z), multiple devices can drive the same node without conflicting
    - several tri-state logic gates
    - example: tri-state buffer with active-high enable

Functional block diagram



Voltage table

E	D	Y
1	0	0
1	1	1
0	X	Z

← Z = high impedance

Tri-State Logic Gates: Verilog

- Dataflow Verilog description of various logic gates

tristate buffer with active-high enable

```
module tristatebuffer(Y,D,E);
      input D, E;
      output Y;
      assign Y = E ? D : 1'bz;
endmodule
```

tristate buffer with active-low enable

```
module tristatebuffer(Y,D,E);
      input D, E;
      output Y;
      assign Y = E ? 1'bz : D;
endmodule
```

tristate inverter with active-high enable

```
module tristateinv(Y,D,E);
      input D, E;
      output Y;
      assign Y = E ? ~D : 1'bz;
endmodule
```

tristate inverter with active-low enable

```
module tristateoinv(Y,D,E);
      input D, E;
      output Y;
      assign Y = E ? 1'bz : ~D;
endmodule
```

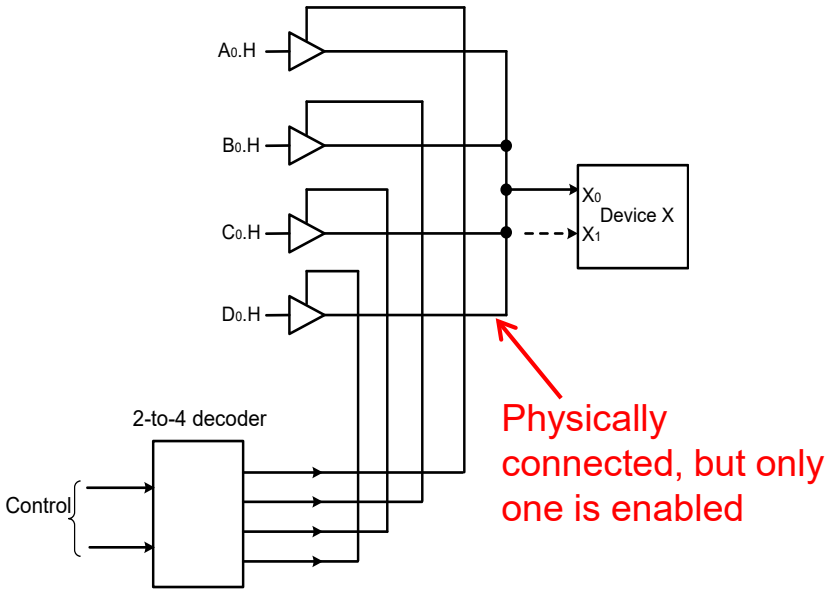
Can you write the dataflow Verilog description of tristate NAND2 and MUX4:1?

MUXes Based on Tri-State Elements

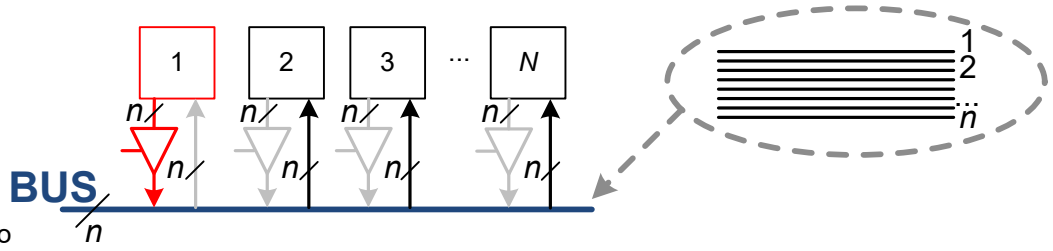
- Tri-state gates with common output implement MUXes

When Control = 00, tri-state device for A_0 is enabled, others are disabled. Hence A_0 is connected to X_0, etc.

Control signals select which input goes to X
⇒ effectively it behaves like a MUX



- Useful to connect several resources to same bus
  - avoids expensive point-to-point interconnection
  - the enabled resource drives the bus (others in Z may receive)



Summary

- Introduction to combinational building blocks and their structural/dataflow Verilog description
- Binary adders
  - half adders, full adders, ripple carry adders
- Magnitude comparators
- Decoders, BCD-to-7-segment decoders
- Encoders, Priority encoders
- Multiplexers
- Demultiplexers
- Tri-state logic elements
- Behavioral Verilog description style is also possible: see next lectures...

Suggestions for Self-Improvement

- In addition to the lecture/tutorials/lab sessions on Verilog, you may want to read chapter 4 of the textbook (see IVLE Workbin)
  - description of logic functions