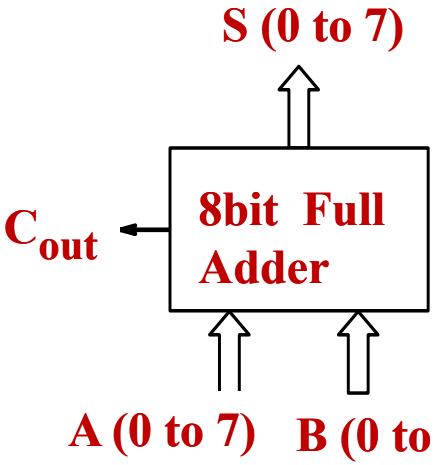
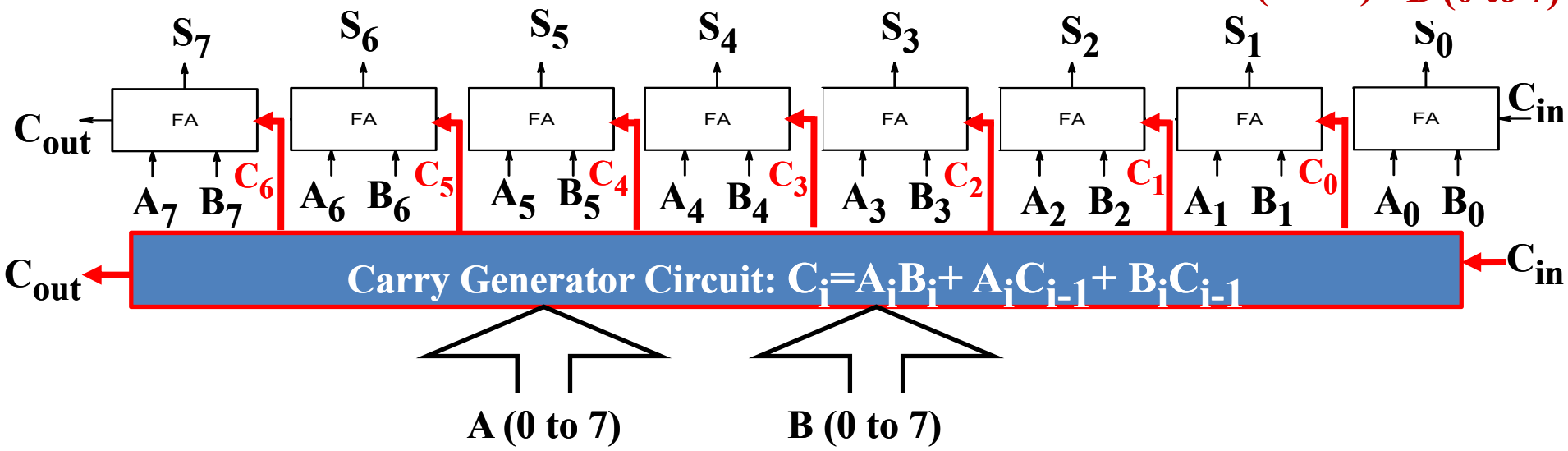
ESc201, Lecture 32: (Digital)

Carry Look Ahead (CLA)

Multi-bit binary adders built by 1-bit Full Adder becomes a botheration for large numbers. This is because the higher order bit addition has to wait for the trickling down of the carry bit from the lowe order bits. But there are no free lunches.



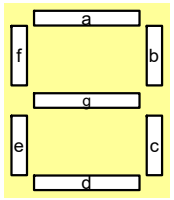


For the 1st stage of the Carry Generator Circuit, the Generate term, $G_i=A_iB_i$, is just an AND gate, and to get the Propagate term $P_i=A_i+B_i$, it is an OR gate. For the 2nd stage (Example: $C_0=G_0+P_0C_{in}$) to generate C_i. Similarly the last term of C_3 is $P_3P_2P_1P_0C_2$. Hence with the increase in the number of bits, the number of Fan-in's to the AND gate increases and it lands up in a law of diminishing returns.

Implementation : Not suitable for more than 8-bits.

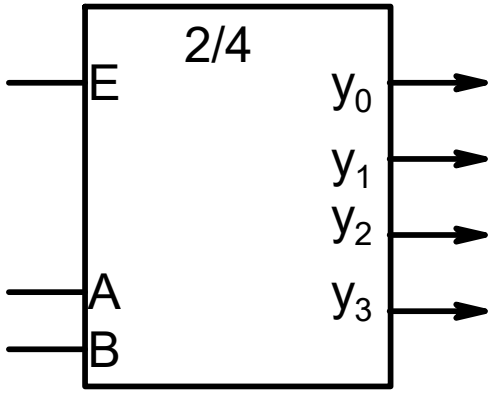
ESc201, Lecture 32: (Digital) Combinational Logic Topics left

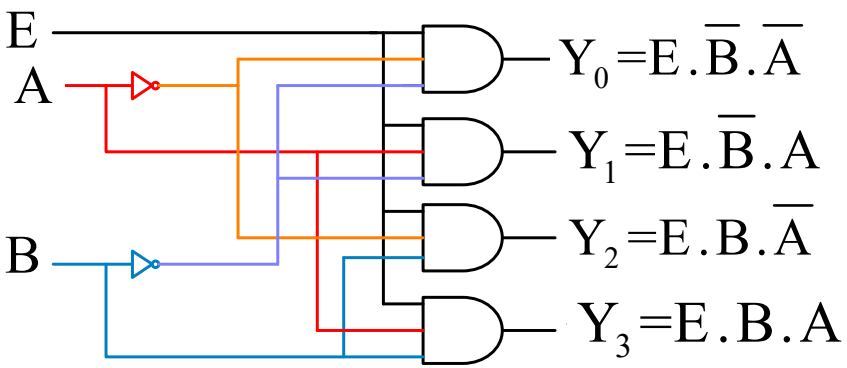
1. Decoders, Encoders
2. Multiplexers
3. Adder/Subtractors, Multipliers
4. Comparators
5. Parity Generators
6. ……..



DC \ BA	00	01	11	10
00	1	0	1	1
01	0	1	1	0
11	0	1	0	0
10	1	1	0	0

E	B	A	Y_0	Y_1	Y_2	Y_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





Decoders with 'Enable' input

Dec or Function	Input					Output						
	D	C	B	A	BI	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	1	0
1	0	0	0	1	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	1	0	0	1
4	0	1	0	0	1	0	1	1	0	0	1	1
5	0	1	0	1	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	0	1	1	1	1	1
7	0	1	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	0	1	1
10	1	0	1	0	1	0	0	0	1	1	0	1
11	1	0	1	1	1	0	0	1	1	0	0	1
12	1	1	0	0	1	0	1	0	0	0	1	1
13	1	1	0	1	1	1	0	0	1	0	1	1
14	1	1	1	0	1	0	0	0	1	1	1	1
15	1	1	1	1	1	0	0	0	0	0	0	0
BI	x	x	x	x	0	0	0	0	0	0	0	0

$a = (\overline{D}+\overline{B}).(\overline{C}+A).(D+C+B+\overline{A})$

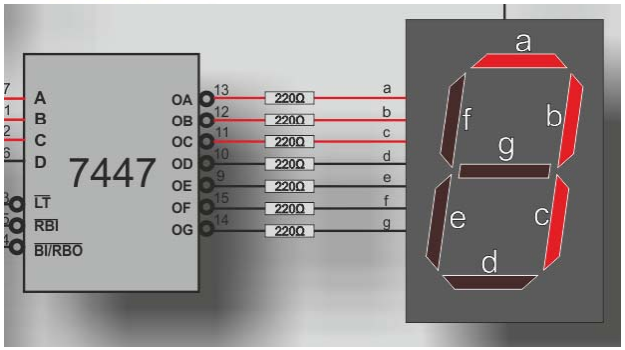
'Enable' used to blank a digit when not used or to hold a value steady.

ESc201, Lecture 32: Digital : Binary Coded Decimal (BCD)

Code each number 0-9 into 4 bits and concatenate, such as:

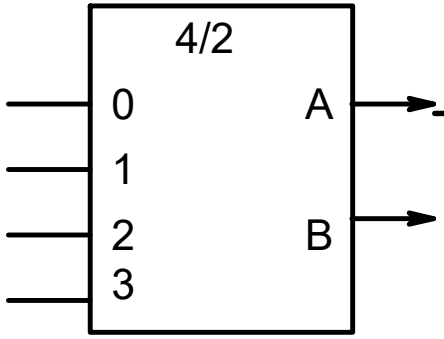
136 = 0001 0011 0110 = 0001 0011 0110 ≠ 310

1 3 6



Sometimes it is simpler to implement functions using BCD than raw binary. But, inefficient due to a large waste of states (10-15) of the 4-bits. An example is a 7-segment display decoder.

Encoders (Inverse of decoders)



d_3	d_2	d_1	d_0	B	A
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

d_3d_2 \ d_1d_0	00	01	11	10
00	x	0	x	1
01	0	x	x	x
11	x	x	x	x
10	1	x	x	x

$A = \overline{d_2}\ \overline{d_0}$

d_3d_2 \ d_1d_0	00	01	11	10
00	x	0	x	0
01	1	x	x	x
11	x	x	x	x
10	1	x	x	x

$B = \overline{d_1}\ \overline{d_0}$

Errors:

- Suppose high quality filters, amplifiers, antennas etc. are available.
- In typical communication systems, error rates are low, e.g. 1 bit error every 100,000 bits
- But not zero! What happens when a bit flips?
- Bit flip error: 0 instead of 1 or 1 instead of 0
- Suppose 100 is transmitted, but a bit error may result in 000 being received.

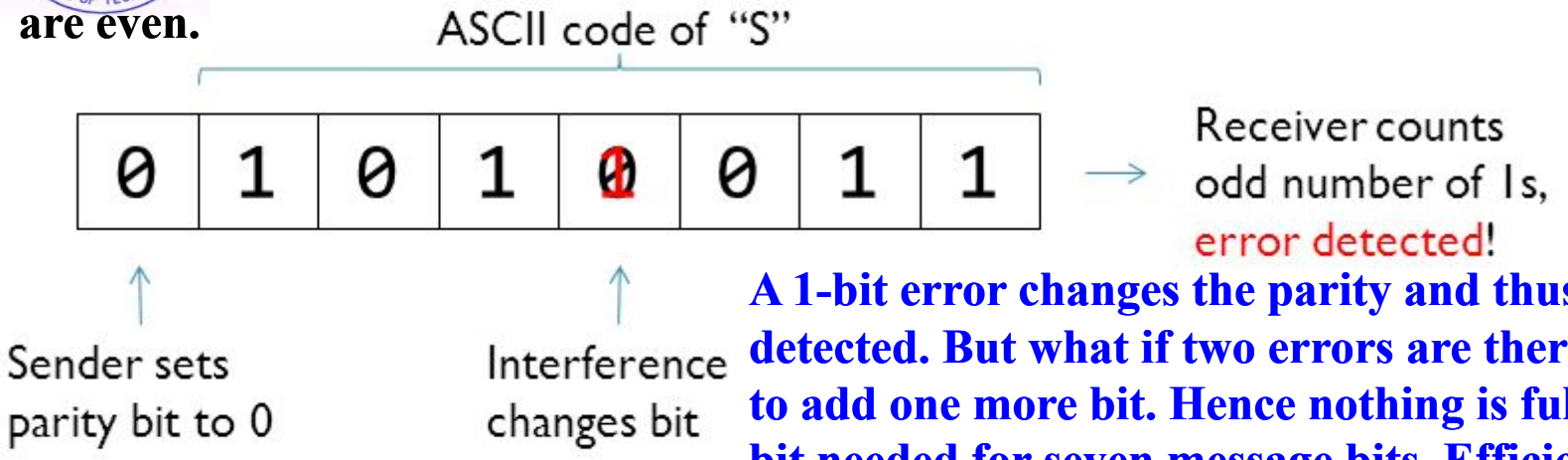
100 (4) ⟹ 000 (0)

Single bit flip results in a catastrophic error !!!

ESc201, Lecture 32: (Digital)

Error detection and correction -- Parity

Even parity: Set MSB so as to ensure that the total number of ones in the string are even.



A 1-bit error changes the parity and thus can be detected. But what if two errors are there ? One needs to add one more bit. Hence nothing is full proof and 1 bit needed for seven message bits. Efficiency is 14% !!

Error detection and correction -- Checksum

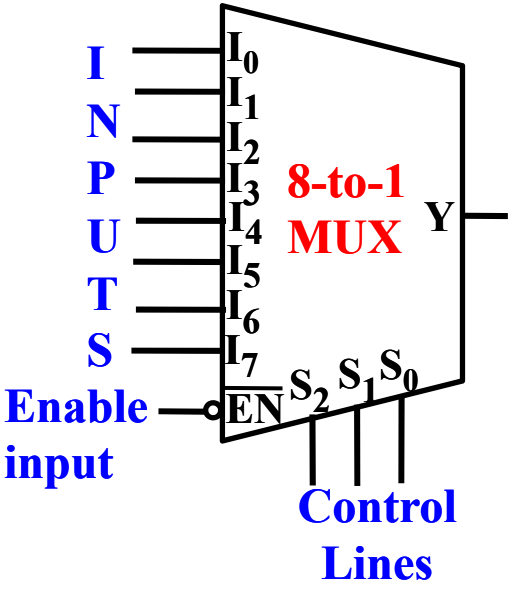
Checksum: When data is to be transmitted over a large distance, as ingle Parity bit is not really effective, as the data is subjected to burst noise which may persisit for several ms. This will change the data in the process. The Checksum character is obtained by summing all the messages and is transmitted to the receiver after the message is sent. The receiver receives the checksum and figures out if a correction is needed.

Example: The checksum code for a block of messages $(48)_{16}$, $(65)_{16}$, $(6C)_{16}$, $(70)_{16}$, and $(21)_{16}$. Sum gives in the LSB 8+5+C+0+1=26. Divide by 16 gives 1 as dividend (to be carried over to next higher bit) and 10 as remainder $=(A)_{16}$. In the next higher bit (here MSB) the sum is 1(carry)+4+6+6+7+2=26. Again divide by 16 to get 10 as remainder $=(A)_{16}$ with a carry over of 1 out of the MSB. Ignoring this carry, the sum is $(AA)_{16}$. Hence the checksum is $(AA)_{16}$. This, however, is not the only algorithm and others more complex ones are commonly what is commercially used.

ESc201, Lecture 32: (Digital) Multiplexers (MUX)

Multiplexing means to transmit a large number of information units over a small number of lines (or channels in communication). A Multiplexer, in general, may have M-data inputs and 1 or more output lines. The path of the M-inputs to the output line is controlled by combinational circuits by a set of N-select lines. The relation between M-input lines to 1-output line is given by $M=2^N$, an N-bit binary code can generate 2^N address codes, where each address code controls one input out of M. Besides the inputs and output there is usually one more input called the 'Enable' line. Usually the 'Enable' line is low for the MUX to be active and it can be made inactive by applying a high-input to the 'Enable' line. Then the truth table for a 8-to-1 MUX is:

Enable (EN)	Select Inputs			Selected Input	Output Y will be the same as one Input
	S_2	S_1	S_0		
0	0	0	0	I_0	I_0
0	0	0	1	I_1	I_1
0	0	1	0	I_2	I_2
0	0	1	1	I_3	I_3
0	1	0	0	I_4	I_4
0	1	0	1	I_5	I_5
0	1	1	0	I_6	I_6
0	1	1	1	I_7	I_7
1	X	X	X	None	0

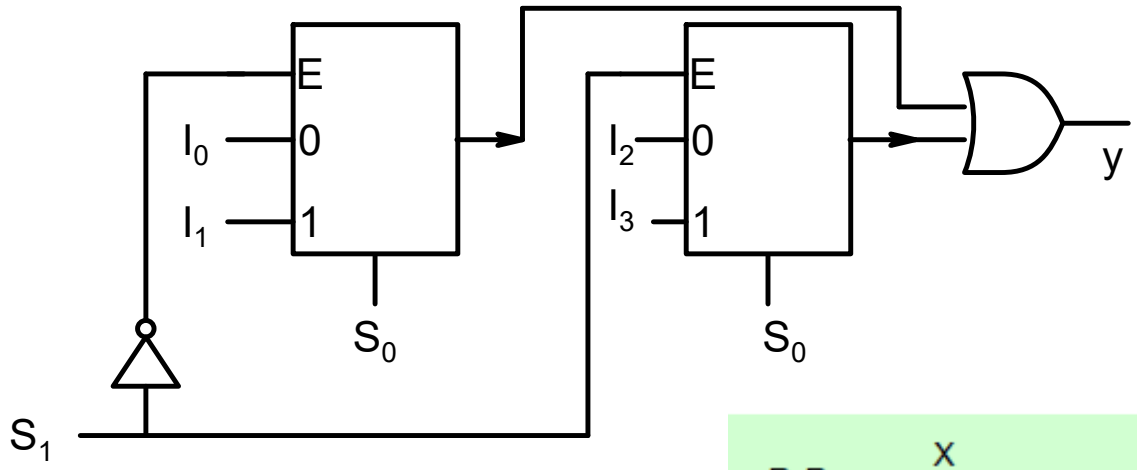


The combinational circuit can now be built up with the algebra on the right. (m's are the minterms)

$$
\begin{aligned}
Y = (&\overline{S_2}.\overline{S_1}.\overline{S_0}.I_0 \\
&+\overline{S_2}.\overline{S_1}.S_0.I_1 \\
&+\overline{S_2}.S_1.\overline{S_0}.I_2 \\
&+\overline{S_2}.S_1.S_0.I_3 \\
&+S_2.\overline{S_1}.\overline{S_0}.I_4 \\
&+S_2.\overline{S_1}.S_0.I_5 \\
&+S_2.S_1.\overline{S_0}.I_6 \\
&+S_2.S_1.S_0.I_7)\overline{EN} \\
&= \Big(\sum_{i=0}^{2^n-1} m_i I_i\Big)\overline{EN}
\end{aligned}
$$

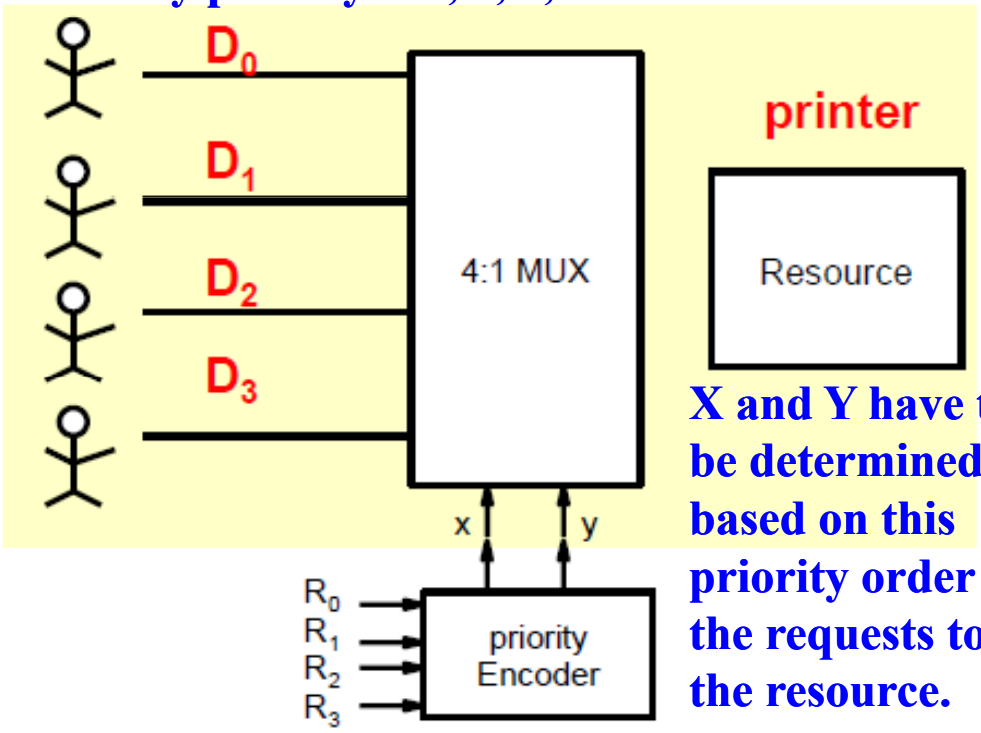
ESc201, Lecture 32: (Digital) MUX Application

MUX expansion:
TWO 2x1 MUX
to 4x1 MUX.



MUX + priority encoders--- who gets to use a Printer first?

Let's say priority is 3, 2, 1, and 0 with user 3 having the highest Priority



R_0	R_1	R_2	R_3	x	y
0	0	0	0	x	x
1	0	0	0	0	0
x	1	0	0	0	1
x	x	1	0	1	0
x	x	x	1	1	1

X and Y have to be determined based on this priority order and the requests to use the resource.

$$X = R_2 + R_3$$

$$Y = R_1\overline{R_2} + R_3$$