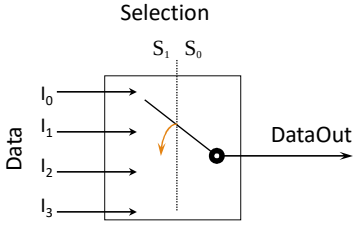
Digital systems: from bits to microcontrollers

- Introduction to digital systems fundamentals
- Combinatorial digital systems
- Sequential digital systems
- Introduction to microcontrollers
- Introduction to advanced implementation platforms

Multiplexers

A multiplexer has two types of inputs, according to their role:

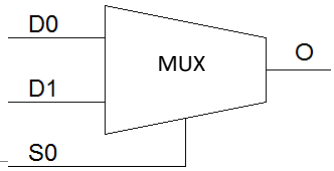
- control inputs, which are responsible for the selection of data inputs;
- data inputs, which provide data to be transferred to output according with value of selection.



Description of the behavior:

If (Selection = 0) then DataOut <= I0
else if (Selection =1) then DataOut <= I1
else if (Selection =2) then DataOut <= I2
else if (Selection =3) then DataOut <= I3

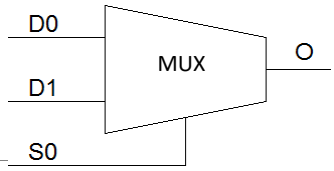
Multiplexer with 1 variable of selection



If (S0 = 0) then O <= D0
else O <= D1

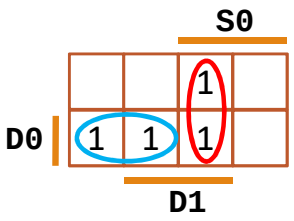
S0	D1	D0	O
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Multiplexer with 1 variable of selection

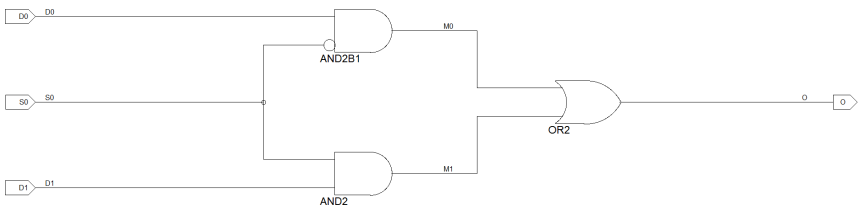


If (S0 = 0) then O <= D0
else O <= D1

S0	D1	D0	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



O = /S0 D0 + S0 D1



Multiplexer with 1 variable of selection

S0	D1	D0	O
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

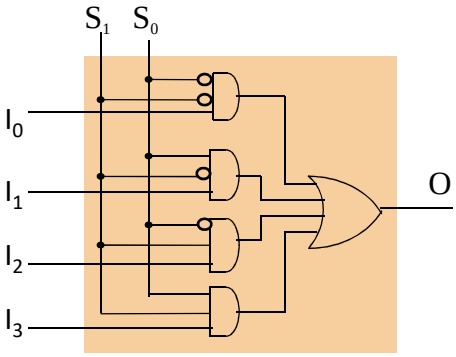
S0	O
0	D0
1	D1

Truth table with embedded variables

O = /S0 D0 + S0 D1

Multiplexer with 2 variables of selection

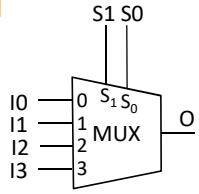
S1	S0	O
0	0	I0
0	1	I1
1	0	I2
1	1	I3



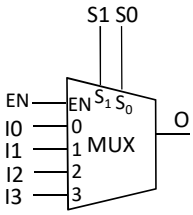
O = /S1. /S0. I0 +
/S1. S0. I1 +
S1. /S0. I2 +
S1. S0. I3

Introducing an enable input

S1	S0	O
0	0	I0
0	1	I1
1	0	I2
1	1	I3



EN	S1	S0	O
0	x	x	0
1	0	0	I0
1	0	1	I1
1	1	0	I2
1	1	1	I3



Modular composition of multiplexers

Would it be possible to implement a multiplexer with two selection variables with multiplexers with one selection variable?

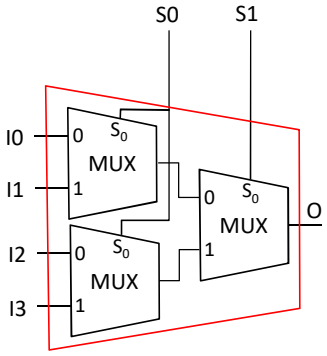
S1	S0	O
0	0	I0
0	1	I1
1	0	I2
1	1	I3

Modular composition of multiplexers

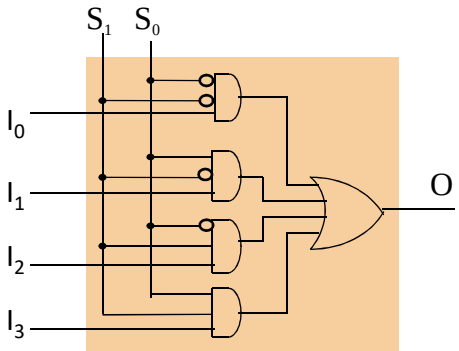
Would it be possible to implement a multiplexer with two selection variables with multiplexers with one selection variable?

S1	S0	O	
0	0	I0	MUX0
0	1	I1	
1	0	I2	MUX1
1	1	I3	

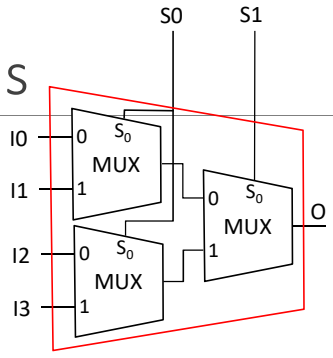
S1	O
0	MUX0
1	MUX1

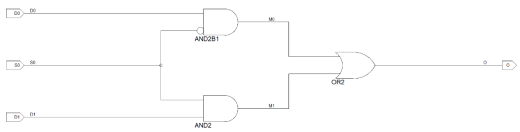


Alternative implementations



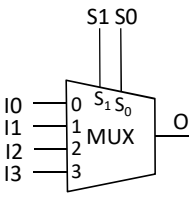
versus
!?

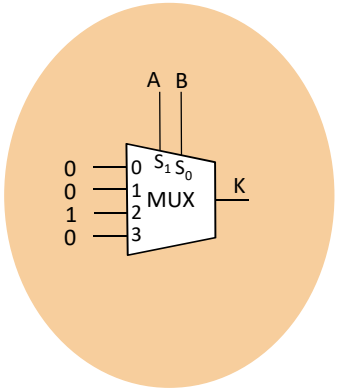




Multiplexer for function implementation

S1	S0	O
0	0	I0
0	1	I1
1	0	I2
1	1	I3





A	B	K
0	0	0
0	1	0
1	0	1
1	1	0

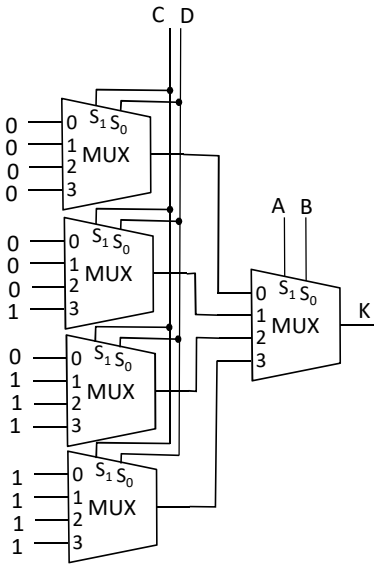
K = A . /B

A B C D		K
0 0 0 0	m_0	0
0 0 0 1	m_1	0
0 0 1 0	m_2	0
0 0 1 1	m_3	0
0 1 0 0	m_4	0
0 1 0 1	m_5	0
0 1 1 0	m_6	0
0 1 1 1	m_7	1
1 0 0 0	m_8	0
1 0 0 1	m_9	1
1 0 1 0	m_{10}	1
1 0 1 1	m_{11}	1
1 1 0 0	m_{12}	1
1 1 0 1	m_{13}	1
1 1 1 0	m_{14}	1
1 1 1 1	m_{15}	1

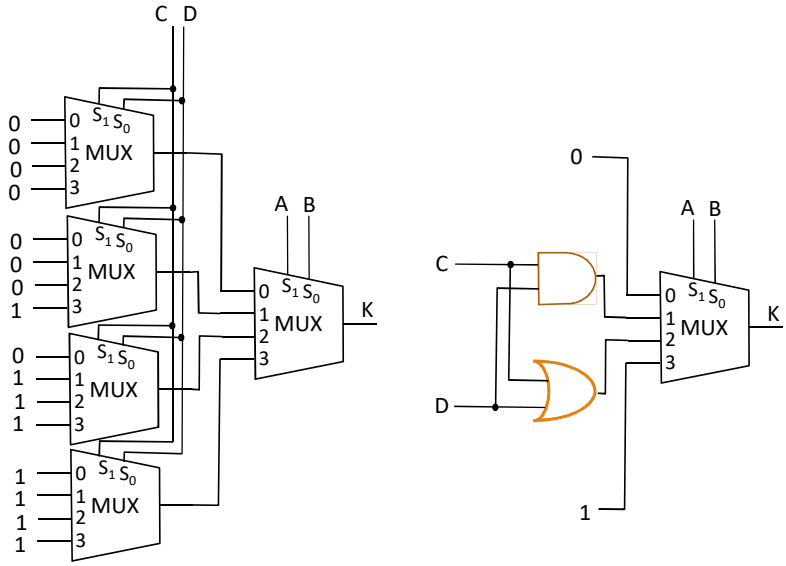
Implementation of function K using multiplexers with 2 selection variables.

A B C D		K	
0 0 0 0	m_0	0	MUX0
0 0 0 1	m_1	0	
0 0 1 0	m_2	0	
0 0 1 1	m_3	0	
0 1 0 0	m_4	0	MUX1
0 1 0 1	m_5	0	
0 1 1 0	m_6	0	
0 1 1 1	m_7	1	
1 0 0 0	m_8	0	MUX2
1 0 0 1	m_9	1	
1 0 1 0	m_{10}	1	
1 0 1 1	m_{11}	1	
1 1 0 0	m_{12}	1	MUX3
1 1 0 1	m_{13}	1	
1 1 1 0	m_{14}	1	
1 1 1 1	m_{15}	1	

A B C D		K
0 0 0 0	m_0	0
0 0 0 1	m_1	0
0 0 1 0	m_2	0
0 0 1 1	m_3	0
0 1 0 0	m_4	0
0 1 0 1	m_5	0
0 1 1 0	m_6	0
0 1 1 1	m_7	1
1 0 0 0	m_8	0
1 0 0 1	m_9	1
1 0 1 0	m_{10}	1
1 0 1 1	m_{11}	1
1 1 0 0	m_{12}	1
1 1 0 1	m_{13}	1
1 1 1 0	m_{14}	1
1 1 1 1	m_{15}	1



A B C D		K
0 0 0 0	m_0	0
0 0 0 1	m_1	0
0 0 1 0	m_2	0
0 0 1 1	m_3	0
0 1 0 0	m_4	0
0 1 0 1	m_5	0
0 1 1 0	m_6	0
0 1 1 1	m_7	1
1 0 0 0	m_8	0
1 0 0 1	m_9	1
1 0 1 0	m_{10}	1
1 0 1 1	m_{11}	1
1 1 0 0	m_{12}	1
1 1 0 1	m_{13}	1
1 1 1 0	m_{14}	1
1 1 1 1	m_{15}	1



A B C D		K	
0 0 0 0	m_0	0	MUX0
0 0 0 1	m_1	0	
0 0 1 0	m_2	0	
0 0 1 1	m_3	0	
0 1 0 0	m_4	0	
0 1 0 1	m_5	0	
0 1 1 0	m_6	0	
0 1 1 1	m_7	1	
1 0 0 0	m_8	0	MUX1
1 0 0 1	m_9	1	
1 0 1 0	m_{10}	1	
1 0 1 1	m_{11}	1	
1 1 0 0	m_{12}	1	
1 1 0 1	m_{13}	1	
1 1 1 0	m_{14}	1	
1 1 1 1	m_{15}	1	

B	C	D	MUX0
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

B	C	D	MUX1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

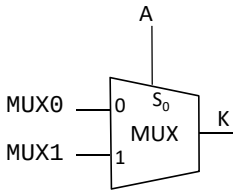
A	K
0	MUX0
1	MUX1

A B C D		K	
0 0 0 0	m_0	0	MUX0
0 0 0 1	m_1	0	
0 0 1 0	m_2	0	
0 0 1 1	m_3	0	
0 1 0 0	m_4	0	
0 1 0 1	m_5	0	
0 1 1 0	m_6	0	
0 1 1 1	m_7	1	
1 0 0 0	m_8	0	MUX1
1 0 0 1	m_9	1	
1 0 1 0	m_{10}	1	
1 0 1 1	m_{11}	1	
1 1 0 0	m_{12}	1	
1 1 0 1	m_{13}	1	
1 1 1 0	m_{14}	1	
1 1 1 1	m_{15}	1	

B	C	D	MUX0
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

B	C	D	MUX1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

A	K
0	MUX0
1	MUX1

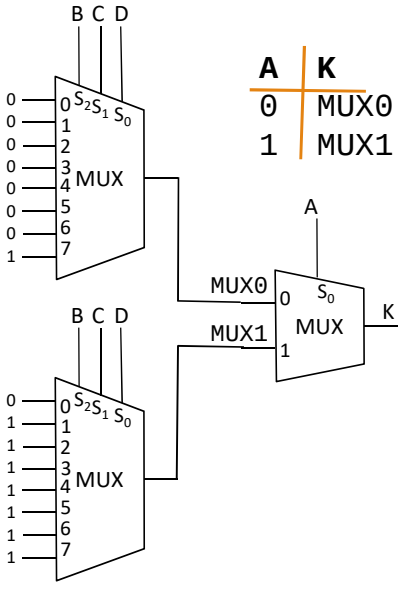


A B C D		K	
0 0 0 0	m_0	0	MUX0
0 0 0 1	m_1	0	
0 0 1 0	m_2	0	
0 0 1 1	m_3	0	
0 1 0 0	m_4	0	
0 1 0 1	m_5	0	
0 1 1 0	m_6	0	
0 1 1 1	m_7	1	
1 0 0 0	m_8	0	MUX1
1 0 0 1	m_9	1	
1 0 1 0	m_{10}	1	
1 0 1 1	m_{11}	1	
1 1 0 0	m_{12}	1	
1 1 0 1	m_{13}	1	
1 1 1 0	m_{14}	1	
1 1 1 1	m_{15}	1	

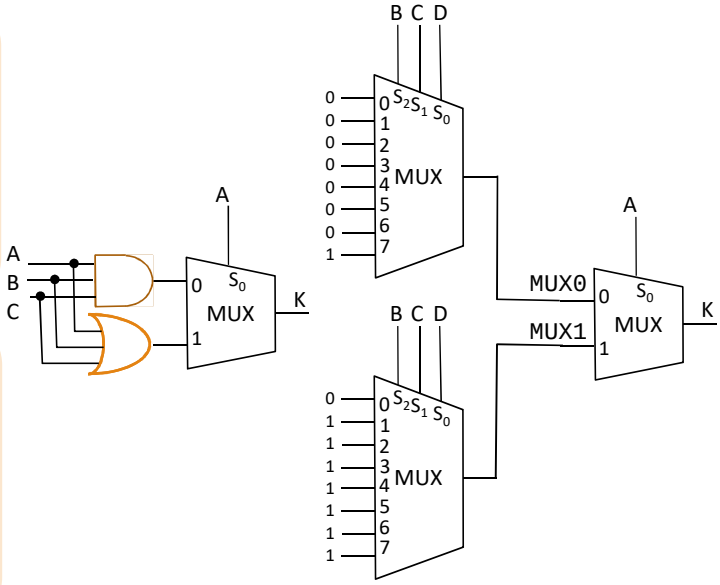
B	C	D	MUX0
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

B	C	D	MUX1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

A	K
0	MUX0
1	MUX1



A B C D		K	
0 0 0 0	m_0	0	MUX0
0 0 0 1	m_1	0	
0 0 1 0	m_2	0	
0 0 1 1	m_3	0	
0 1 0 0	m_4	0	
0 1 0 1	m_5	0	
0 1 1 0	m_6	0	
0 1 1 1	m_7	1	
1 0 0 0	m_8	0	MUX1
1 0 0 1	m_9	1	
1 0 1 0	m_{10}	1	
1 0 1 1	m_{11}	1	
1 1 0 0	m_{12}	1	
1 1 0 1	m_{13}	1	
1 1 1 0	m_{14}	1	
1 1 1 1	m_{15}	1	

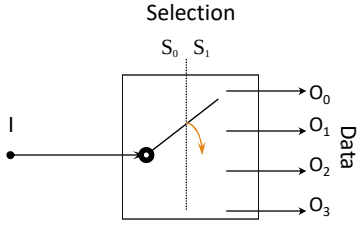


Demultiplexers

A demultiplexer has two types of inputs, according to their role:

- control inputs, which are responsible for the selection of data outputs;

- data input, which provide data to be transferred to one output according with value of selection.



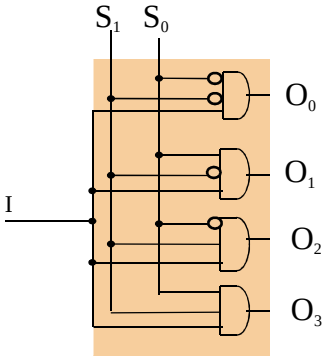
Description of the behavior:

If (Selection = 0) then O0 <= I
else if (Selection =1) then O1 <= I
else if (Selection =2) then O2 <= I
else if (Selection =3) then O3 <= I

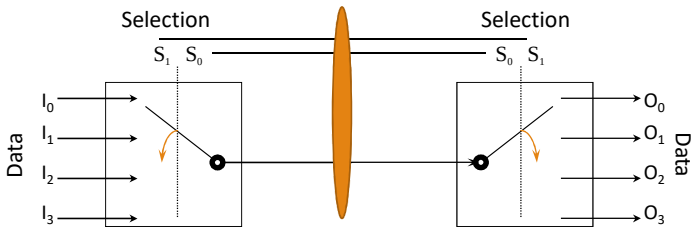
Demultiplexer with 2 variables of selection

S1	S0	I	O0	O1	O2	O3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

S1	S0	O0	O1	O2	O3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I



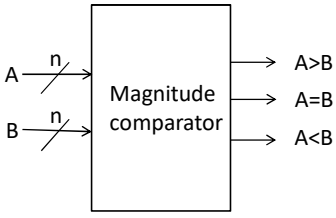
Composition of mux and demux



Introducing digital arithmetic

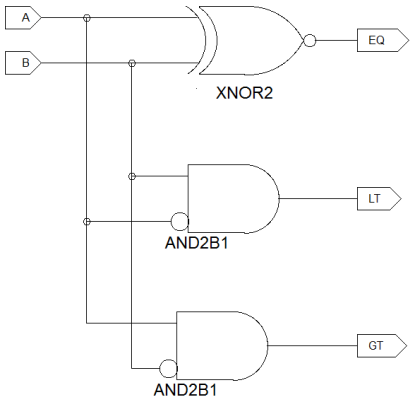
- Comparators
- Adders
- Basic arithmetic

Magnitude comparator



1-bit magnitude comparator

A	B	A = B	A < B	A > B
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0



$A=B \rightarrow \overline{A \oplus B} = A \odot B$

A<B → /A.B

A>B → A./B

1-bit magnitude comparator

A	B	A = B	A < B	A > B
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

1-bit magnitude comparator

A	B	A = B	A < B	A > B
0	0	1	0	0
0	1	0	1	0
1	0	0	0	1
1	1	1	0	0

A=B	A<B	A>B
0	0	1
0	1	0
1	0	0
1	1	x

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

$= \overline{A=B + A<B}$

3-bit magnitude comparator

Input numbers: $A_2A_1A_0$ and $B_2B_1B_0$

Output (A=B)

$$(A_2 = B_2).(A_1 = B_1).(A_0 = B_0)$$

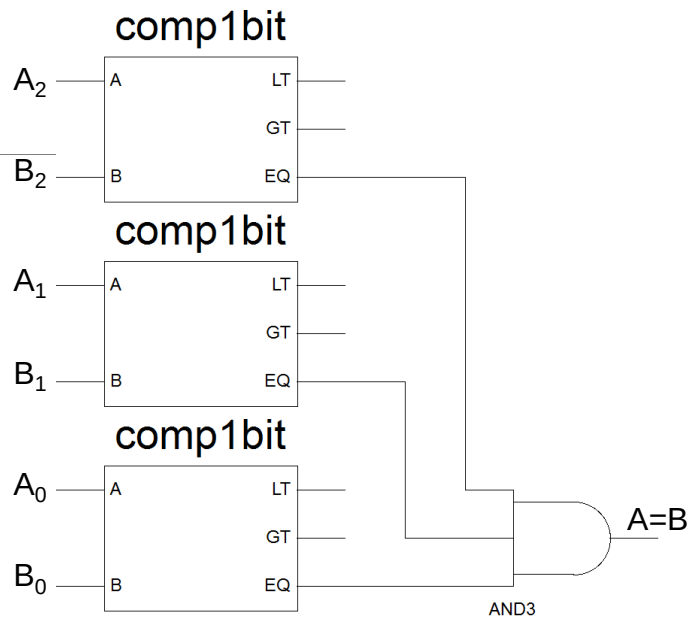
Output (A>B)

$$(A_2 > B_2) + (A_2 = B_2).(A_1 > B_1) + (A_2 = B_2).(A_1 = B_1).(A_0 > B_0)$$

Output (A<B)

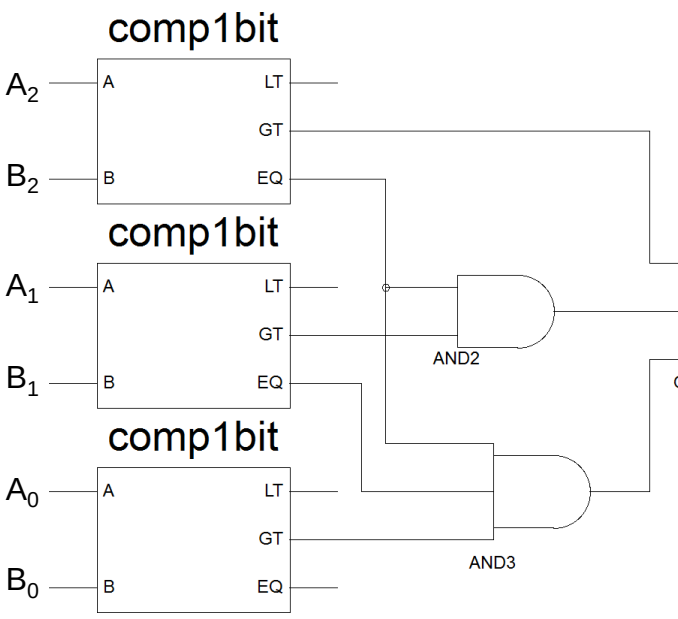
$$(A_2 < B_2) + (A_2 = B_2).(A_1 < B_1) + (A_2 = B_2).(A_1 = B_1).(A_0 < B_0)$$

3-bit magnitude comparator



Output (A=B)

$(A_2 = B_2).(A_1 = B_1).(A_0 = B_0)$



$(A_2 > B_2)$ +

$(A_2 = B_2).(A_1 > B_1)$ +

$(A_2 = B_2).(A_1 = B_1).(A_0 > B_0)$

Decimal addition

```
   2 0 1 5
+  1 9 5 8
-------------
   0 0 1 ↵↓
-------------
   3 9 7 3
```

Decimal addition table

	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

Binary addition

```
     1 0 1 1
+      1 1 0
-------------
   1 1 1 0 ↵↓
-------------
   1 0 0 0 1
```

Counting sequence

0
1
10
11

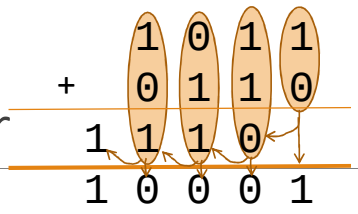
2 bit addition

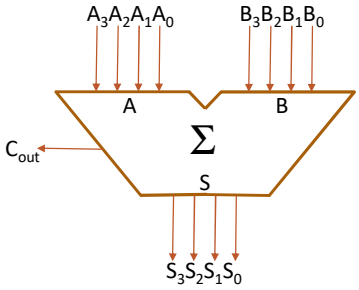
	0	1
0	0	1
1	1	10

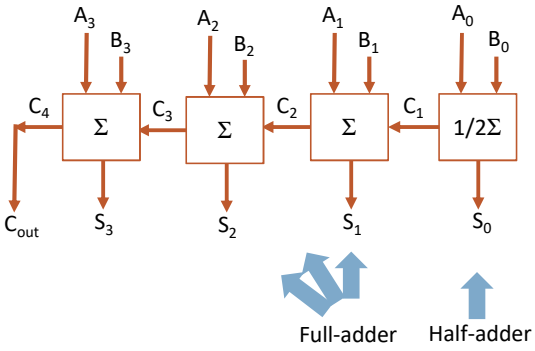
3 bit addition

Inputs			Carry	Add
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

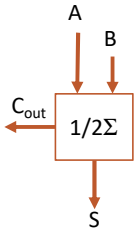
Pseudo-parallel adder



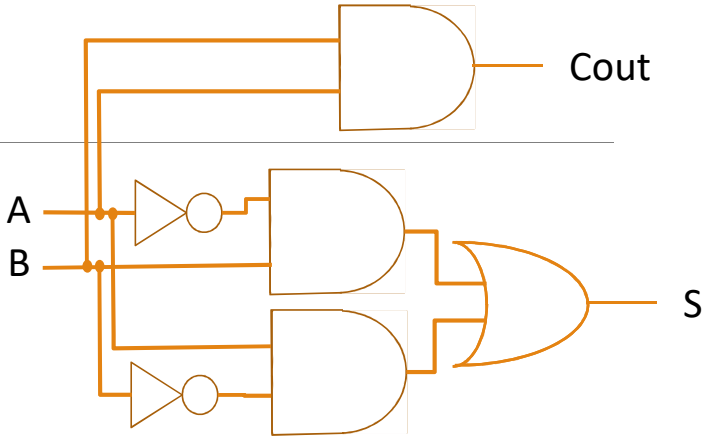




Half-adder



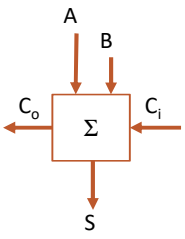
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



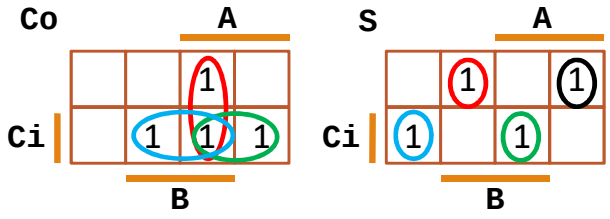
$C_{out} = A.B$

$S = A \oplus B = \overline{A}.B + A.\overline{B}$

Full-adder



A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



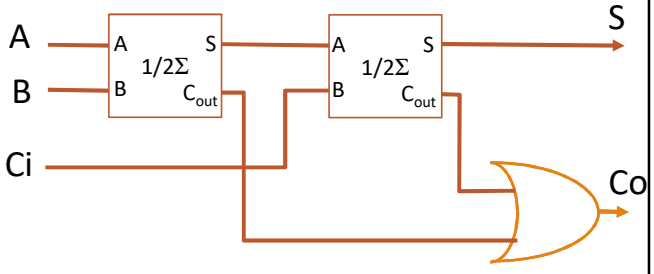
$$Co = A.B + A.Ci + B.Ci$$

$$S = \overline{A}.\overline{B}.Ci + \overline{A}.B.\overline{Ci} + A.B.Ci + A.\overline{B}.\overline{Ci}$$

$$= A \oplus B \oplus Ci$$

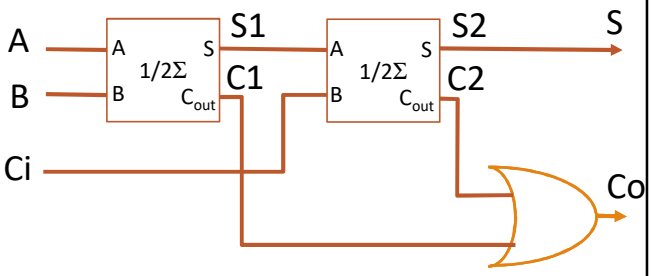
Decompose full-adder into half-adders

A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

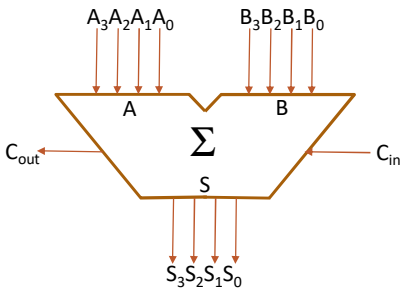


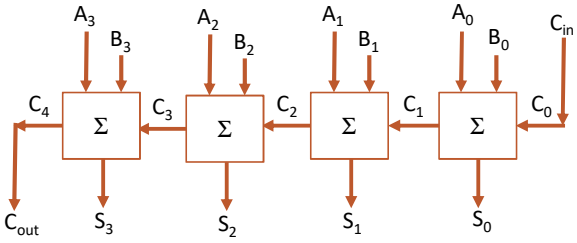
Decompose full-adder into half-adders

A	B	C_i	C_o	S	C1	S1	C2	S2	Co	S
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	1	0	1
0	1	0	0	1	0	1	0	1	0	1
0	1	1	1	0	0	1	1	0	1	0
1	0	0	0	1	0	1	0	1	0	1
1	0	1	1	0	0	1	1	0	1	0
1	1	0	1	0	1	0	0	0	1	0
1	1	1	1	1	1	0	0	1	1	1

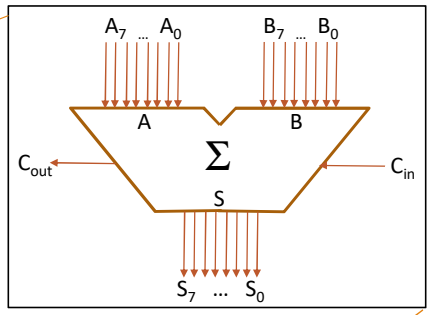


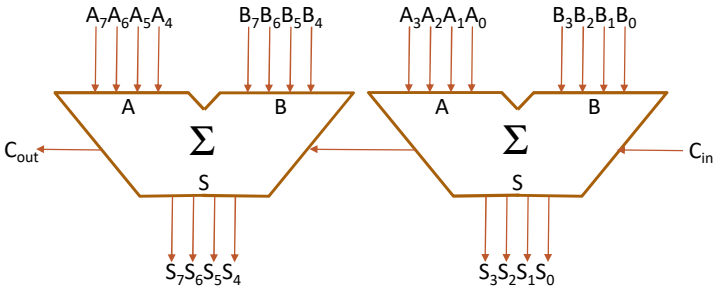
Pseudo-parallel adder



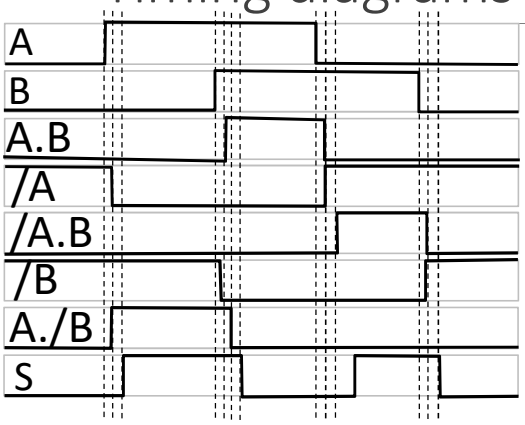


Modular composition of parallel adders

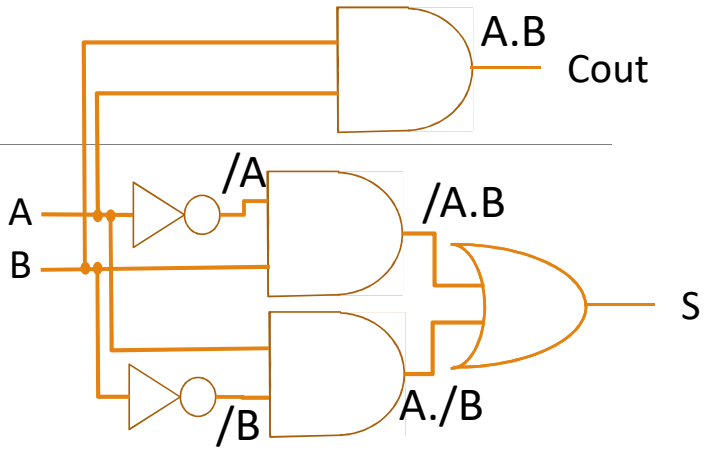




Timing diagrams



delays of ns (depending on selected technology)



$S = /A.B + A./B$