Lecture 6

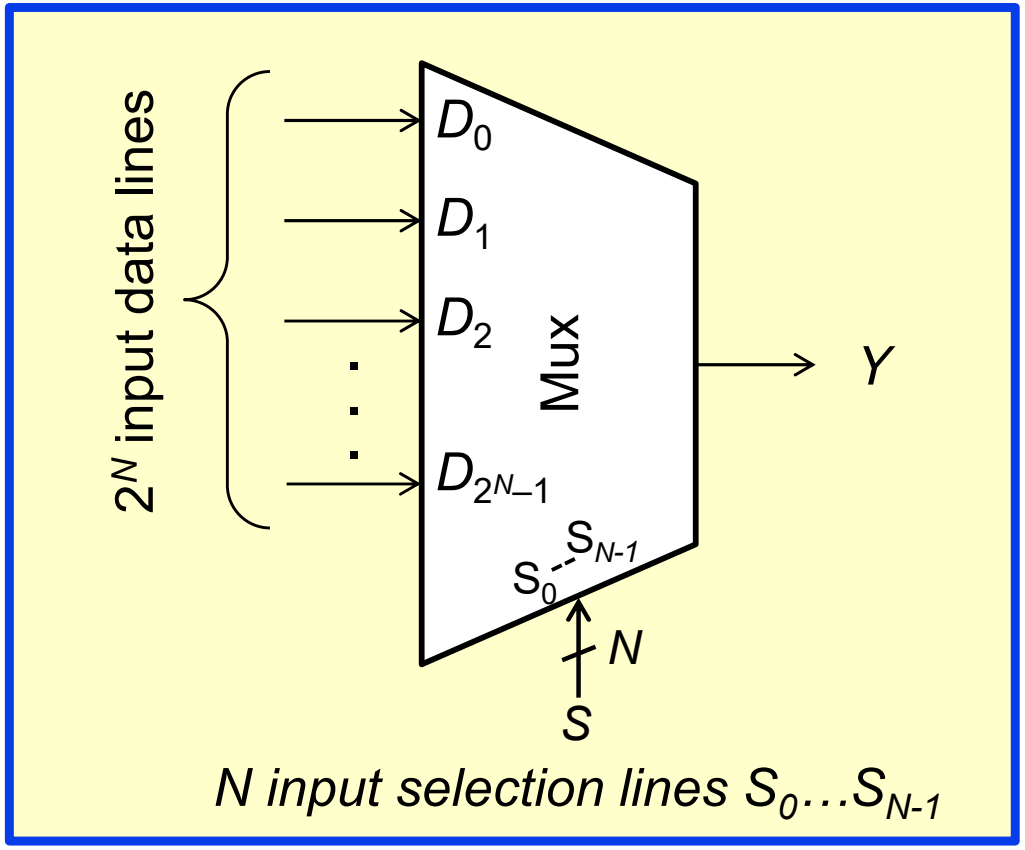
Multiplexers & Demultiplexers

Learning outcomes

- Explain the operation of a multiplexer and demultiplexer.
- Expand a multiplexer to handle more data inputs.
- Use a multiplexer to implement a Boolean function.
- Apply multiplexers to specific applications.

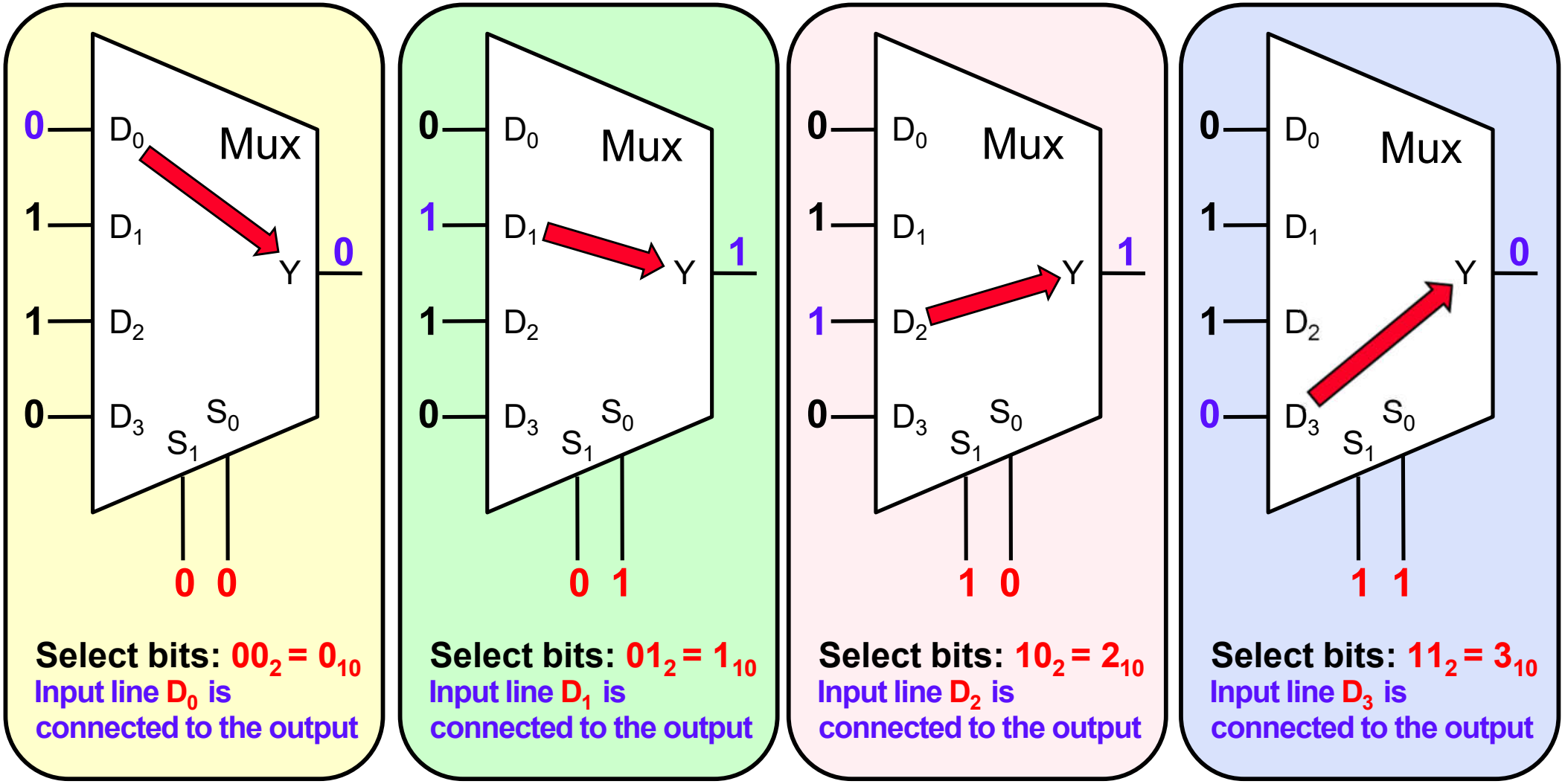
m-to-1 Multiplexer

- An m-to-1 multiplexer (also called “mux”)
  - m=2^N input data lines and *N* selection input lines
  - A single output
  - Connects a single input data line to the output
  - *N* input selection lines determine which input data line is fed to the output.



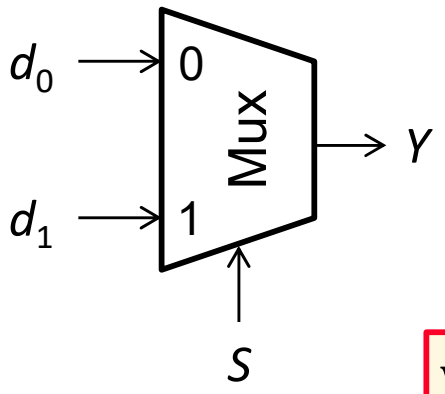
4-to-1 multiplexer – Example

- A 4-to-1 multiplexer or 4-to-1 Mux:
  - 2^2=4 input lines and 2 selection input lines
  - A single output



Multiplexers – Truth tables & Boolean functions

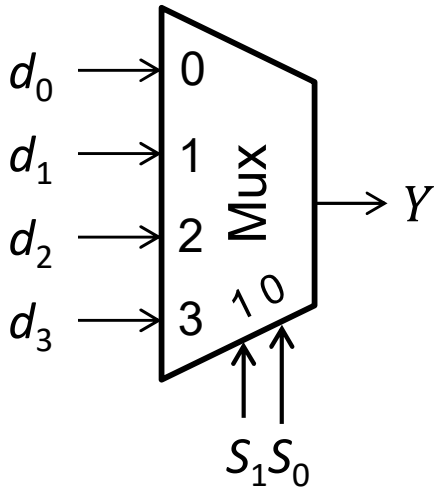
2-to-1 Multiplexer



$$Y = d_0\,S' + d_1\,S$$

Inputs			Output
S	d_0	d_1	Y
0	0	X	$0 = d_0$
0	1	X	$1 = d_0$
1	X	0	$0 = d_1$
1	X	1	$1 = d_1$

4-to-1 Multiplexer

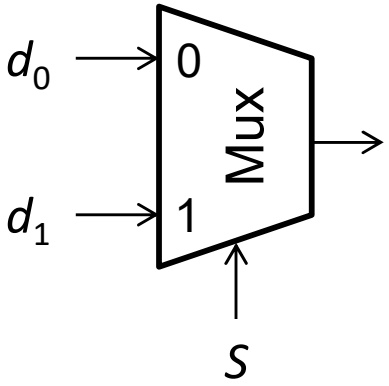


$$Y = d_0\,S_1'S_0' + d_1\,S_1'S_0 + d_2\,S_1S_0' + d_3\,S_1S_2$$

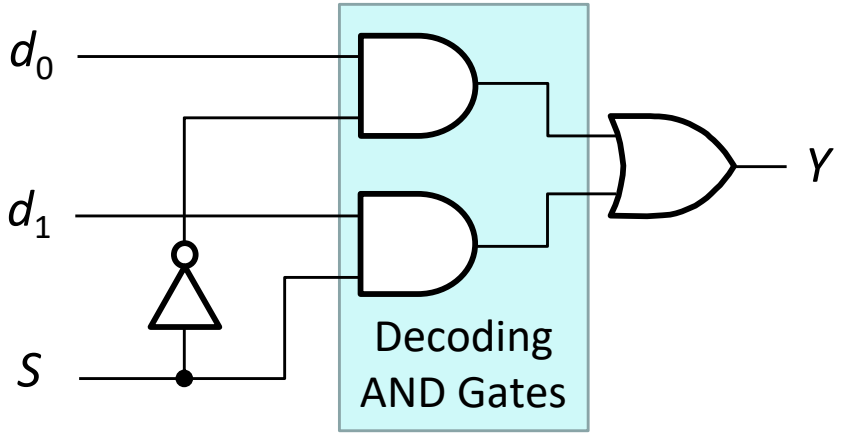
Inputs						Output
S_1	S_0	d_0	d_1	d_2	d_3	Y
0	0	0	X	X	X	$0 = d_0$
0	0	1	X	X	X	$1 = d_0$
0	1	X	0	X	X	$0 = d_1$
0	1	X	1	X	X	$1 = d_1$
1	0	X	X	0	X	$0 = d_2$
1	0	X	X	1	X	$1 = d_2$
1	1	X	X	X	0	$0 = d_3$
1	1	X	X	X	1	$1 = d_3$

Multiplexers – Circuit implementation

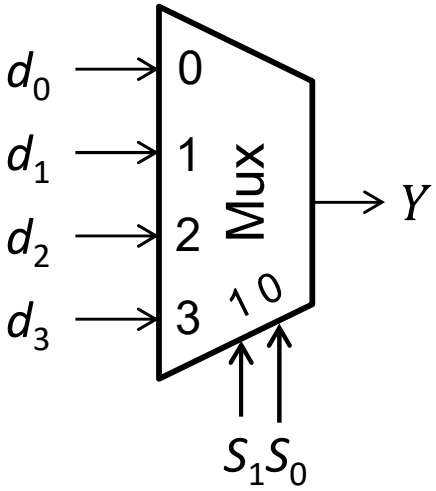
2-to-1 Multiplexer

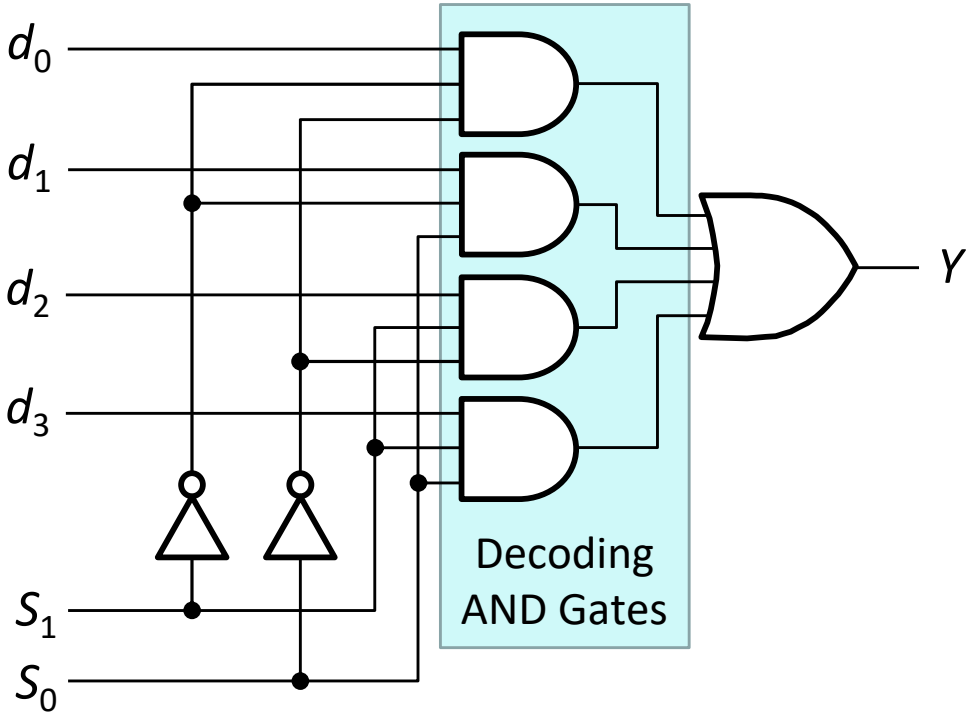


$$Y = d_0\,S' + d_1\,S$$



4-to-1 Multiplexer

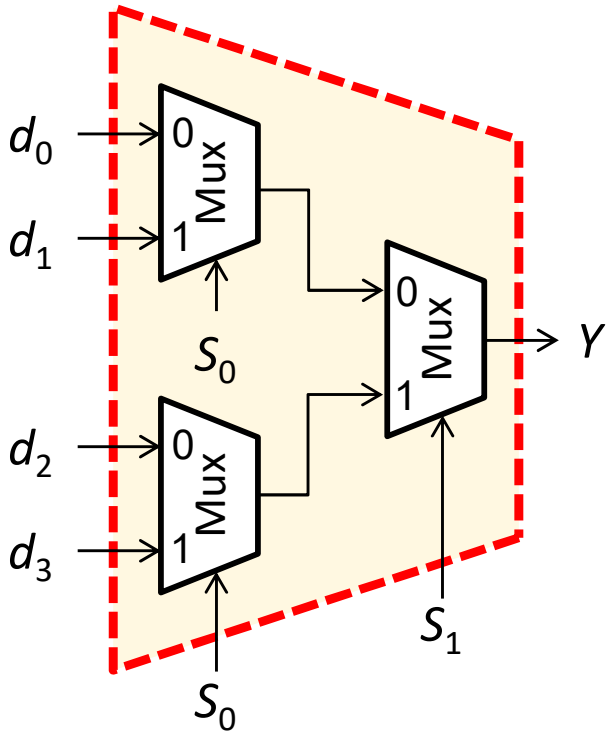




$$Y = d_0\,S_1'S_0' + d_1\,S_1'S_0 \;+ d_2\,S_1S_0' + d_3\,S_1S_2$$

Implementing larger multiplexers – Example 1

- Smaller multiplexers can be cascaded to build larger multiplexers:
  - Muxes are organized in stages, with the first stage having the required number of inputs and the last stage having a single multiplexer whose output is the output of the larger multiplexer.
  - The outputs of each stage are fed to the subsequent stage.
  - Muxes in a stage use the same select bits. LSB is typically used for the first stage and the MSB for the final stage.

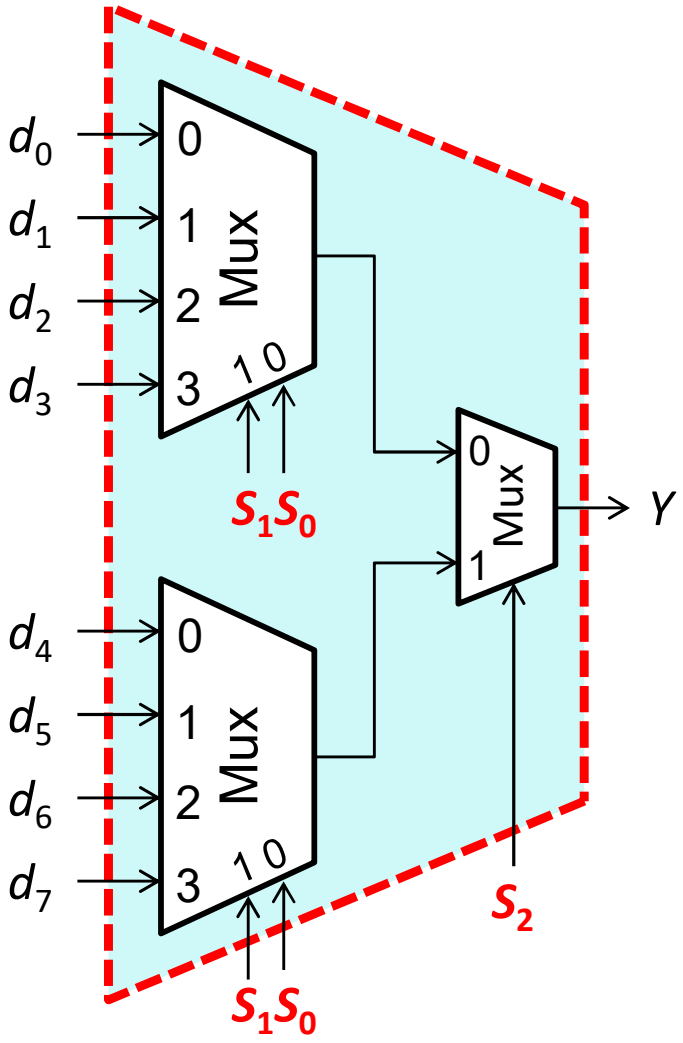


4-to-1 multiplexer using three 2-to-1 multiplexers.

Inputs						Output
S_1	S_0	d_0	d_1	d_2	d_3	Y
0	0	0	X	X	X	$0 = d_0$
0	0	1	X	X	X	$1 = d_0$
0	1	X	0	X	X	$0 = d_1$
0	1	X	1	X	X	$1 = d_1$
1	0	X	X	0	X	$0 = d_2$
1	0	X	X	1	X	$1 = d_2$
1	1	X	X	X	0	$0 = d_3$
1	1	X	X	X	1	$1 = d_3$

Implementing larger multiplexers – Example 2

- 8-to-1 Mux using two 4-to-1 Muxes and one 2-to-1 Mux.



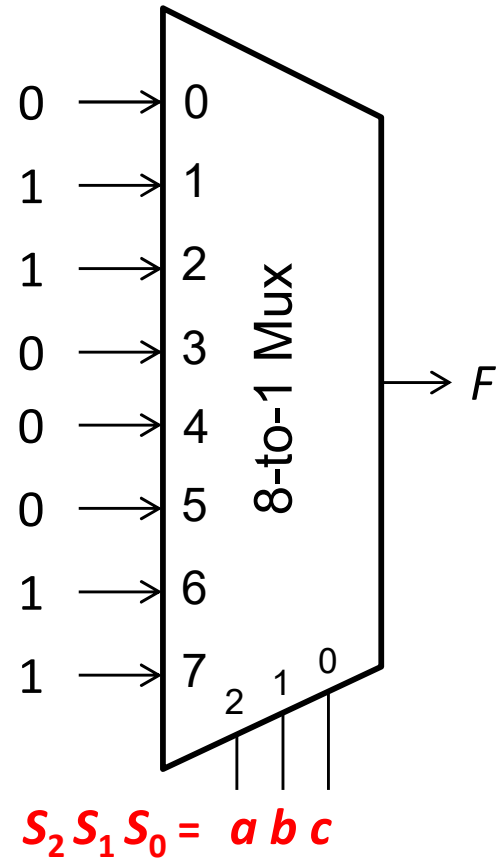
Inputs											Output
S_2	S_1	S_0	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	Y
0	0	0	0	X	X	X	X	X	X	X	$0 = d_0$
0	0	0	1	X	X	X	X	X	X	X	$1 = d_0$
0	0	1	X	0	X	X	X	X	X	X	$0 = d_1$
0	0	1	X	1	X	X	X	X	X	X	$1 = d_1$
0	1	0	X	X	0	X	X	X	X	X	$0 = d_2$
0	1	0	X	X	1	X	X	X	X	X	$1 = d_2$
0	1	1	X	X	X	0	X	X	X	X	$0 = d_3$
0	1	1	X	X	X	1	X	X	X	X	$1 = d_3$
1	0	0	X	X	X	X	0	X	X	X	$0 = d_4$
1	0	0	X	X	X	X	1	X	X	X	$1 = d_4$
1	0	1	X	X	X	X	X	0	X	X	$0 = d_5$
1	0	1	X	X	X	X	X	1	X	X	$1 = d_5$
1	1	0	X	X	X	X	X	X	0	X	$0 = d_6$
1	1	0	X	X	X	X	X	X	1	X	$1 = d_6$
1	1	1	X	X	X	X	X	X	X	0	$0 = d_7$
1	1	1	X	X	X	X	X	X	X	1	$1 = d_7$

Implementing a Boolean function using a Mux

- One can implement any Boolean function using a multiplexer. If Boolean function has N input variables, then:
  - a single 2^N-to-1 multiplexer is sufficient without the need of any other logic circuitry
  - Or a smaller multiplexer with logic gates in front of the input data lines.
- Example: Implement F(a, b, c) = ∑(1, 2, 6, 7) using a single Mux

The inputs are used as select lines to a Mux. An 8-to-1 Mux is used because there are 3 variables

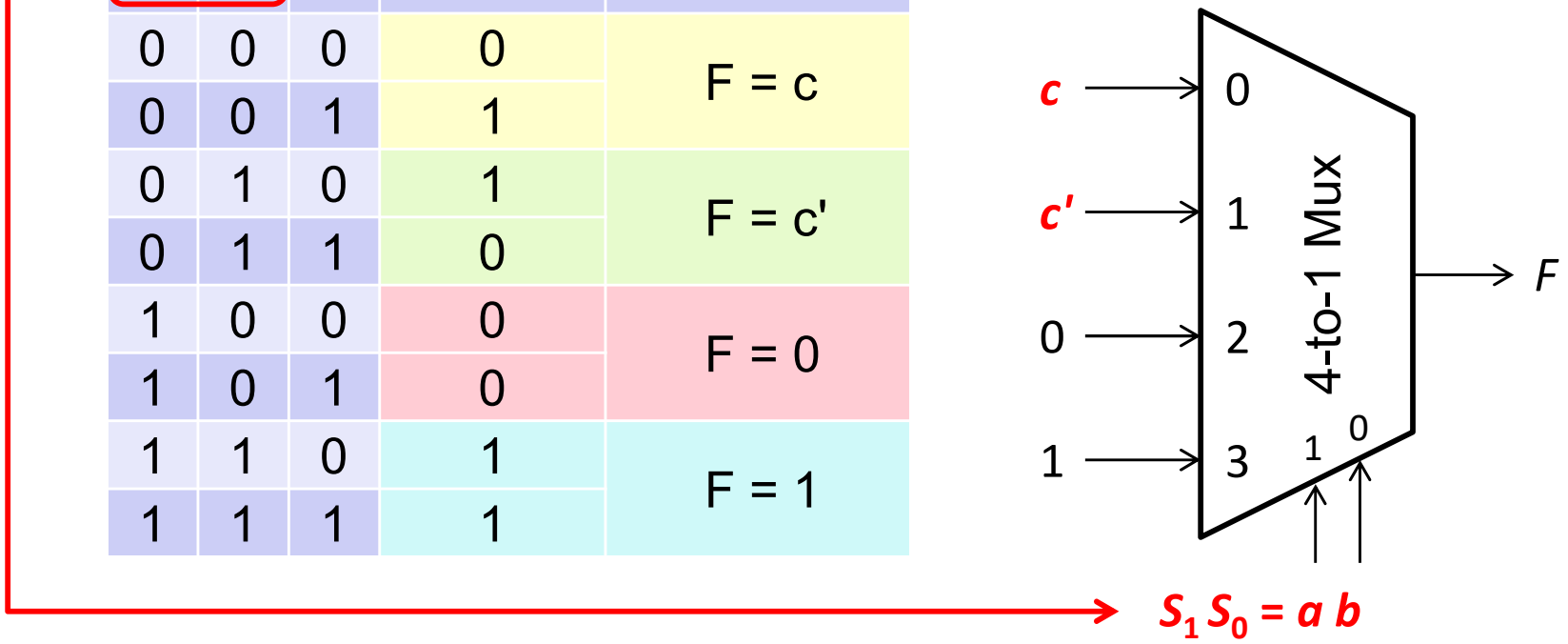
Inputs			Output
a	b	c	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



Implementing a Boolean function using a Mux

- Re-implement $F(a, b, c) = \sum(1, 2, 6, 7)$ with a smaller 4-to-1 Mux
- We will use the two select lines for variables *a* and *b*
- Variable *c* and its complement are used as inputs to the Mux

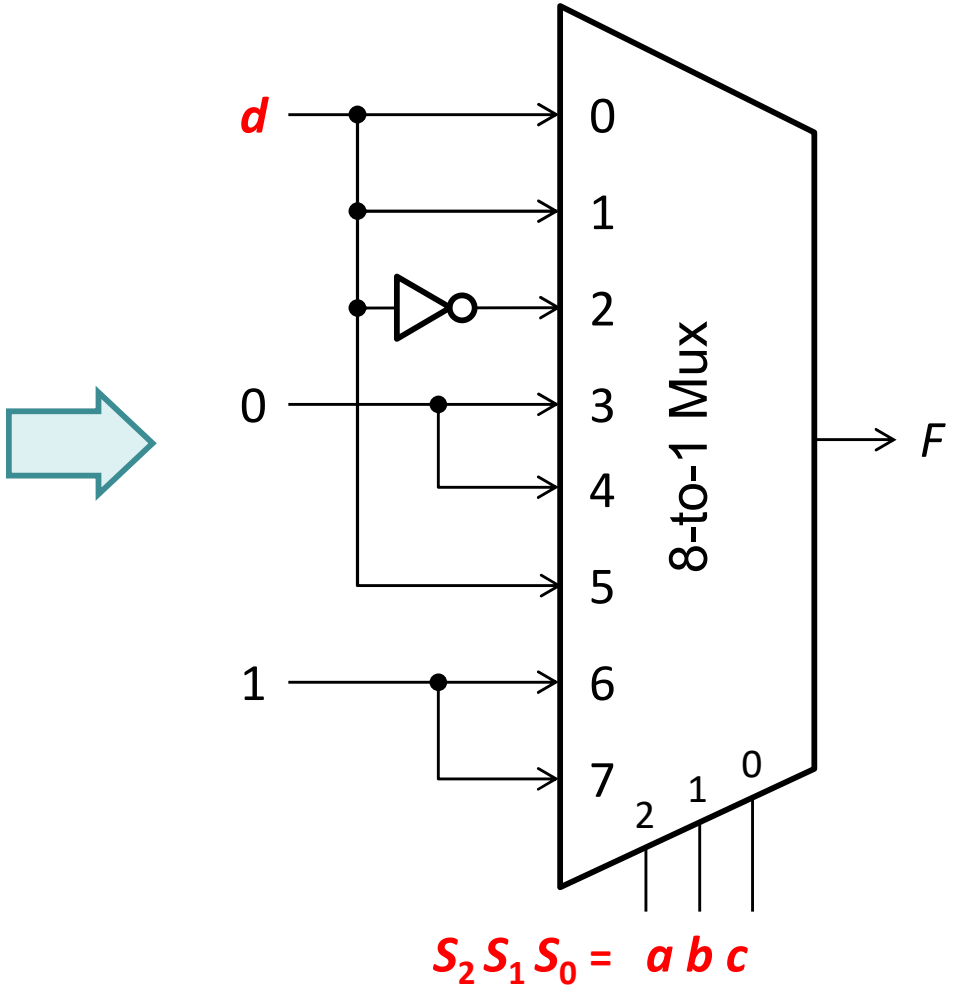
Inputs			Output	Comment
a	b	c	F	F
0	0	0	0	F = c
0	0	1	1	
0	1	0	1	F = c'
0	1	1	0	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	



Implementing a Boolean function using a Mux

- Implement F(a, b, c, d) = ∑(1,3,4,11,12,13,14,15) using 8-to-1 Mux

Inputs				Output	Comment
a	b	c	d	F	F
0	0	0	0	0	F = d
0	0	0	1	1	
0	0	1	0	0	F = d
0	0	1	1	1	
0	1	0	0	1	F = d'
0	1	0	1	0	
0	1	1	0	0	F = 0
0	1	1	1	0	
1	0	0	0	0	F = 0
1	0	0	1	0	
1	0	1	0	0	F = d
1	0	1	1	1	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	



Shannon's Expansion Theorem

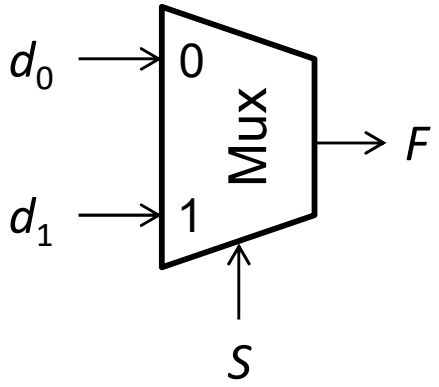
- Any Boolean function f $(x_1, \ldots, x_i, \ldots, x_n)$ can be written in the form:

$$f(x_1, \ldots, x_i, \ldots, x_n) = x_i' \cdot f_{x_i'} + x_i \cdot f_{x_i}$$

with $f_{x_i'} = f(x_1, \ldots, x_i = 0, \ldots, x_n)$ and $f_{x_i} = f(x_1, \ldots, x_i = 1, \ldots, x_n)$

- This expansion can be done in terms of any of the n input variables.
- We saw previously that the output of a 2-to-1 mux is given by:

$$F = d_0\, S' + d_1\, S$$



- Comparing the above 2 expressions, Shannon's expansion theorem indicates that one can use x_i as the select bit *S* of a 2-to-1 mux with:
  - $f_{x_i'} = f(x_1, \ldots, x_i = 0, \ldots, x_n)$ as input 0 and
  - $f_{x_i} = f(x_1, \ldots, x_i = 1, \ldots, x_n)$ as input 1.

Shannon's Expansion Theorem

- Shannon's expansion can be carried out in terms of more than one input variable by recursively applying the theorem.
- For example, expanding the Boolean function in terms of input variables x_1 and x_2 will lead to:

$$
\begin{aligned}
f(x_1, \ldots, x_i, \ldots, x_n) = {} & \boldsymbol{x_1'\, x_2'} \cdot f(x_1 = \mathbf{0}, x_2 = \mathbf{0}, \ldots, x_i, \ldots, x_n) \; + \\
& \boldsymbol{x_1'\, x_2} \cdot f(x_1 = \mathbf{0}, x_2 = \mathbf{1}, \ldots, x_i, \ldots, x_n) \; + \\
& \boldsymbol{x_1\, x_2'} \cdot f(x_1 = \mathbf{1}, x_2 = \mathbf{0}, \ldots, x_i, \ldots, x_n) \; + \\
& \boldsymbol{x_1\, x_2} \cdot f(x_1 = \mathbf{1}, x_2 = \mathbf{1}, \ldots, x_i, \ldots, x_n)
\end{aligned}
$$

- This expansion gives a form that can be implemented using a 4-to-1 multiplexer.
- When Shannon's expansion is carried out with respect to all *n* variables, then the result is the canonical sum-of- products form.

Shannon's Expansion Theorem – Example

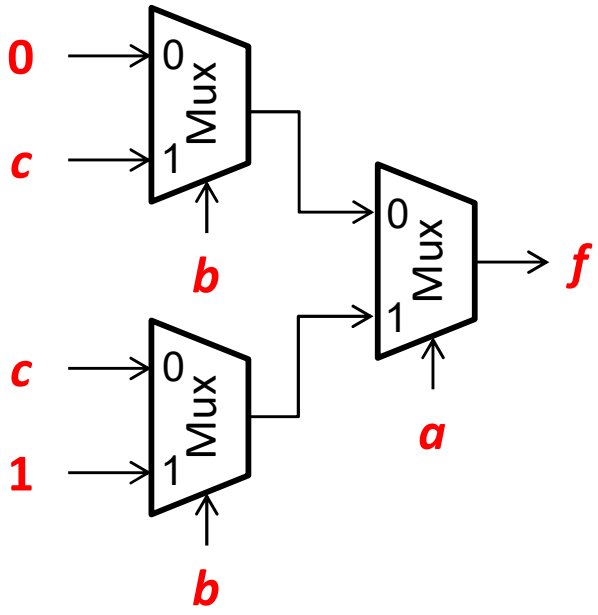
- Consider the Boolean function:

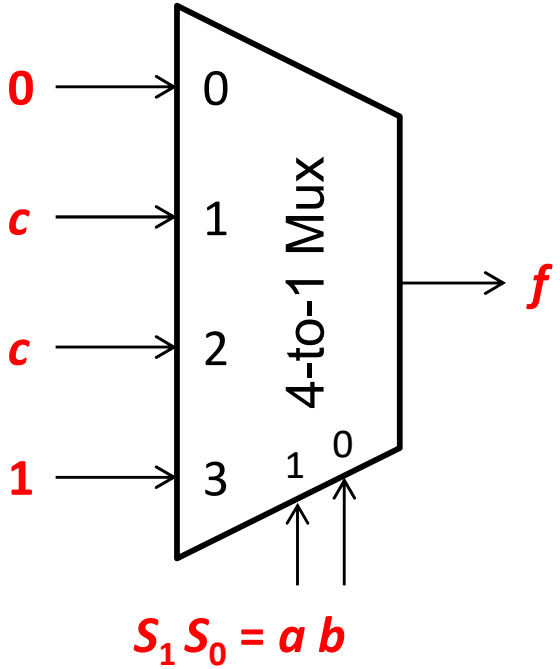
$$f(a,b,c) = ab + ac + bc = a'(bc) + a(b + c)$$

- By recursively applying the theorem:

$$f(a,b,c) = a'(b'(0) + b(c)) + a(b'(c) + b(1))$$

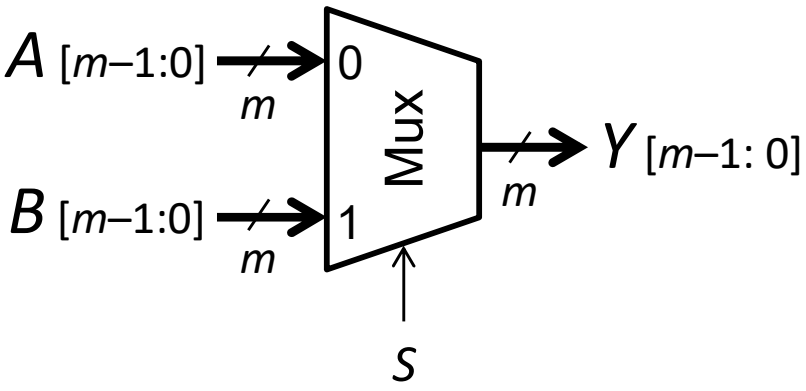
$$f(a,b,c) = a'b'(0) + a'b(c) + ab'(c) + ab(1)$$

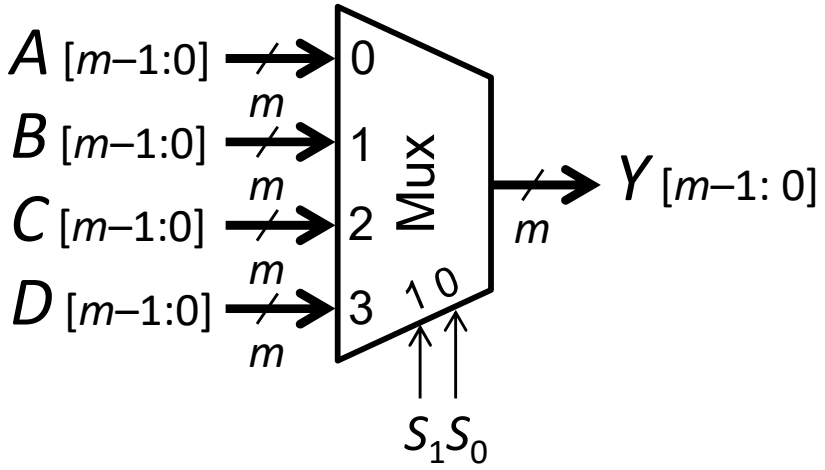




Multiplexers with Vector Input and Output

The inputs and output of a multiplexer can be *m*-bit vectors





2-to-1 Multiplexer with *m* bits

Inputs and output are *m*-bit vectors

Using *m* copies of a 2-to-1 Mux

4-to-1 Multiplexer with *m* bits

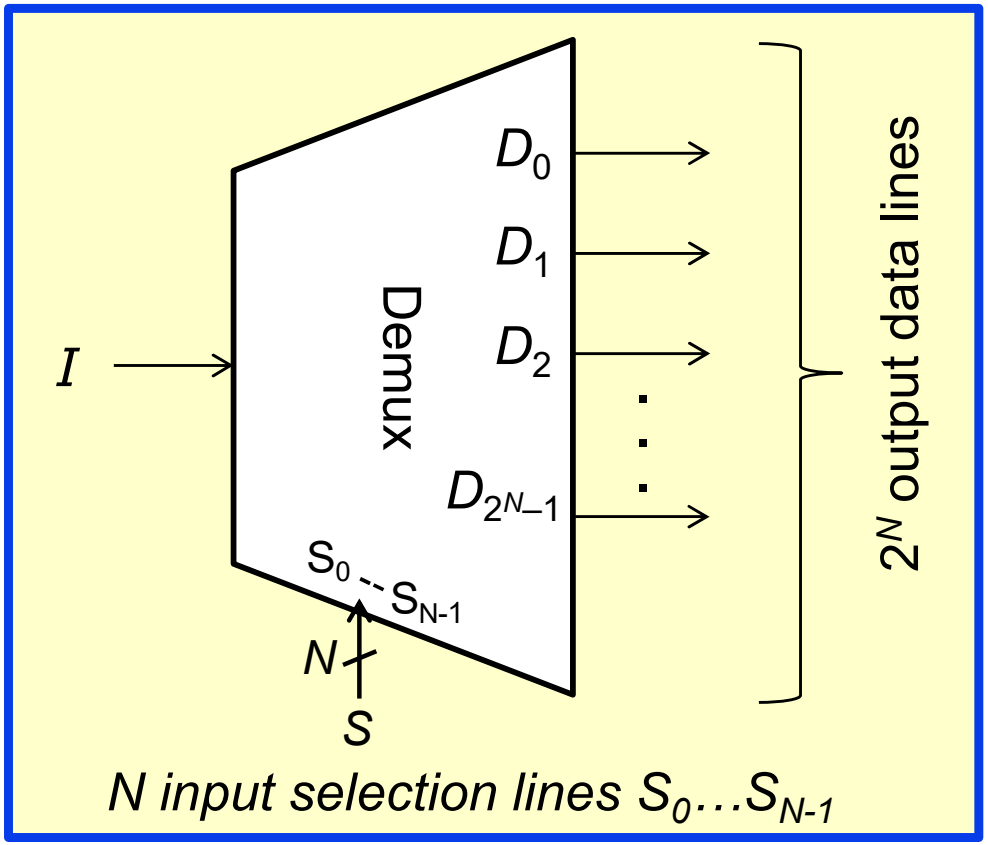
Inputs and output are *m*-bit vectors

Using *m* copies of a 4-to-1 Mux

Demultiplexers

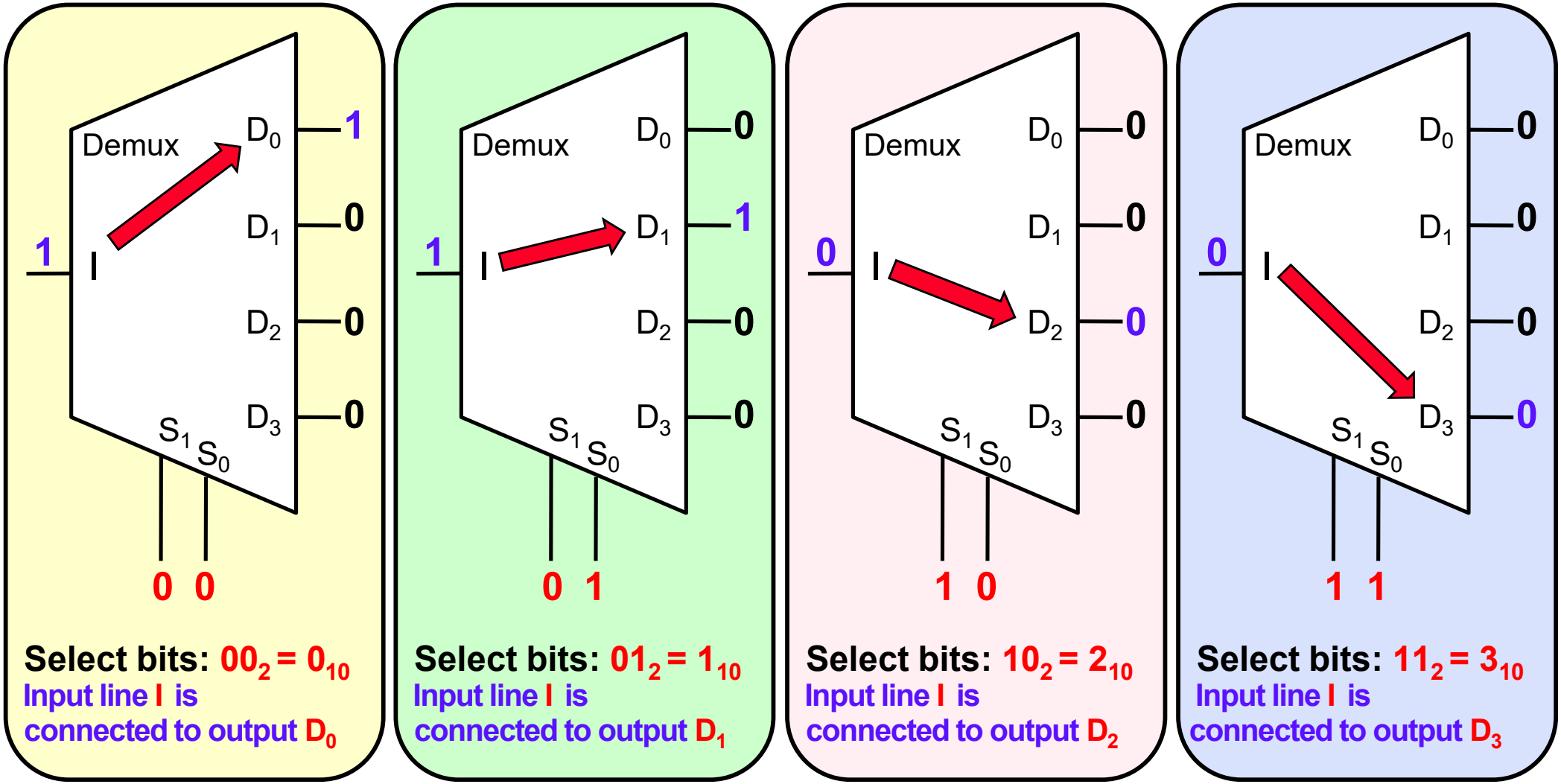
1-to-m Demultiplexer

- A 1-to-m demultiplexer (also called “Demux”) performs the inverse function of multiplexers:
  - A single input data line *I* and m=2^N output data lines D_0-D_{2N-1}
  - The input data line is connected to a single output.
  - *N* selection input lines determine which output data line is connected to the input. All other output data lines are inactive (i.e. held low/high).



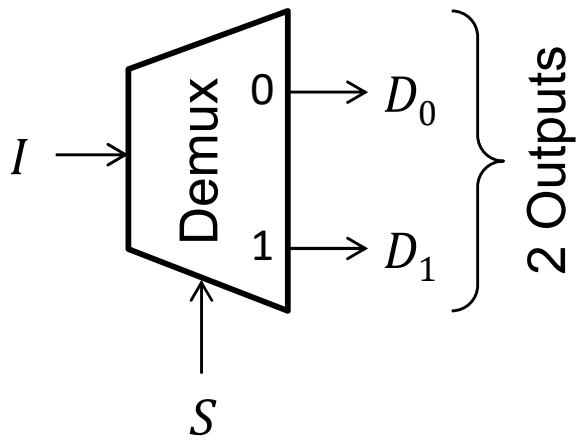
1-to-4 Demultiplexer – Example

- A 1-to-4 Demultiplexer or 1-to-4 Demux:
  - A single input
  - 2^2=4 output data lines and 2 selection input lines
  - Inactive output data lines are held low in this example



Demultiplexers – Truth tables & Boolean functions

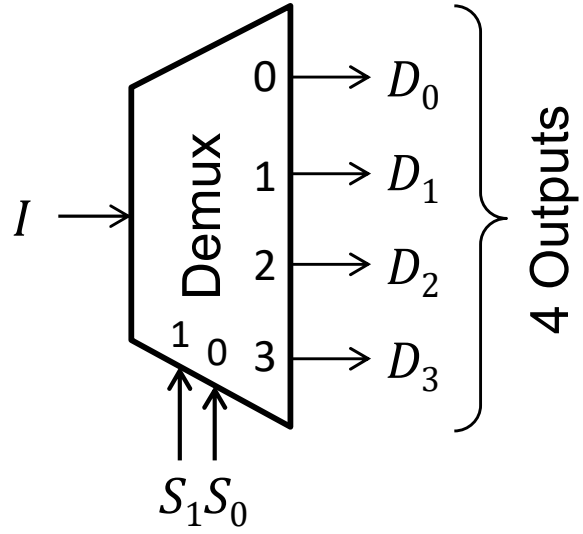
1-to-2 Demultiplexer



$$D_0 = I\,S'$$
$$D_1 = I\,S$$

Inputs		Outputs	
S	I	D_0	D_1
0	0	0	0
0	1	1	0
1	0	0	0
1	1	0	1

1-to-4 Demultiplexer

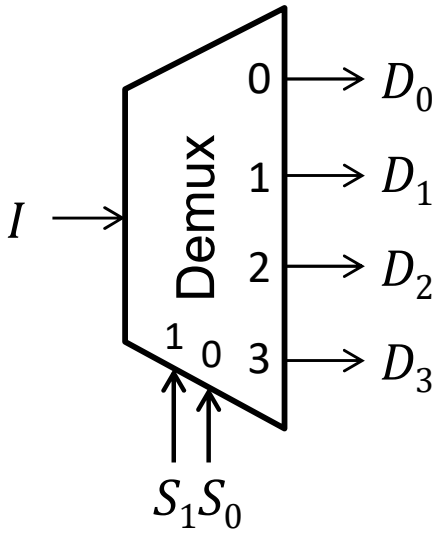


$$D_0 = I\,S_1'S_0'$$
$$D_1 = I\,S_1'S_0$$
$$D_2 = I\,S_1S_0'$$
$$D_3 = IS_1S_0$$

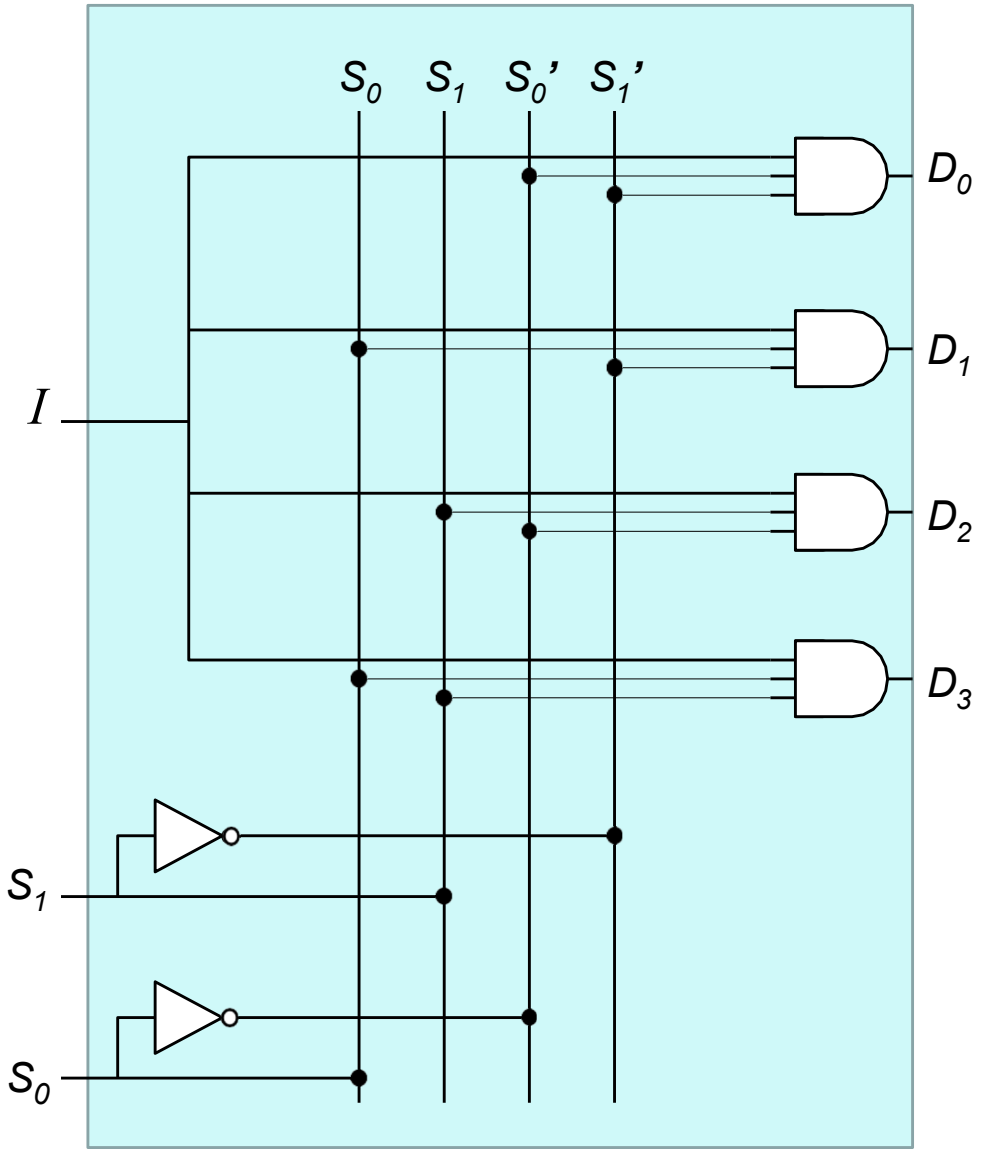
Inputs			Outputs			
S_1	S_0	I	D_0	D_1	D_2	D_3
0	0	0	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	0	0
0	1	1	0	1	0	0
1	0	0	0	0	0	0
1	0	1	0	0	1	0
1	1	0	0	0	0	0
1	1	1	0	0	0	1

Demultiplexers – Truth tables & Boolean functions

- **1-to-4 Demultiplexer**



$$D_0 = I\,S_1'S_0'$$
$$D_1 = I\,S_1'S_0$$
$$D_2 = I\,S_1S_0'$$
$$D_3 = IS_1S_0$$

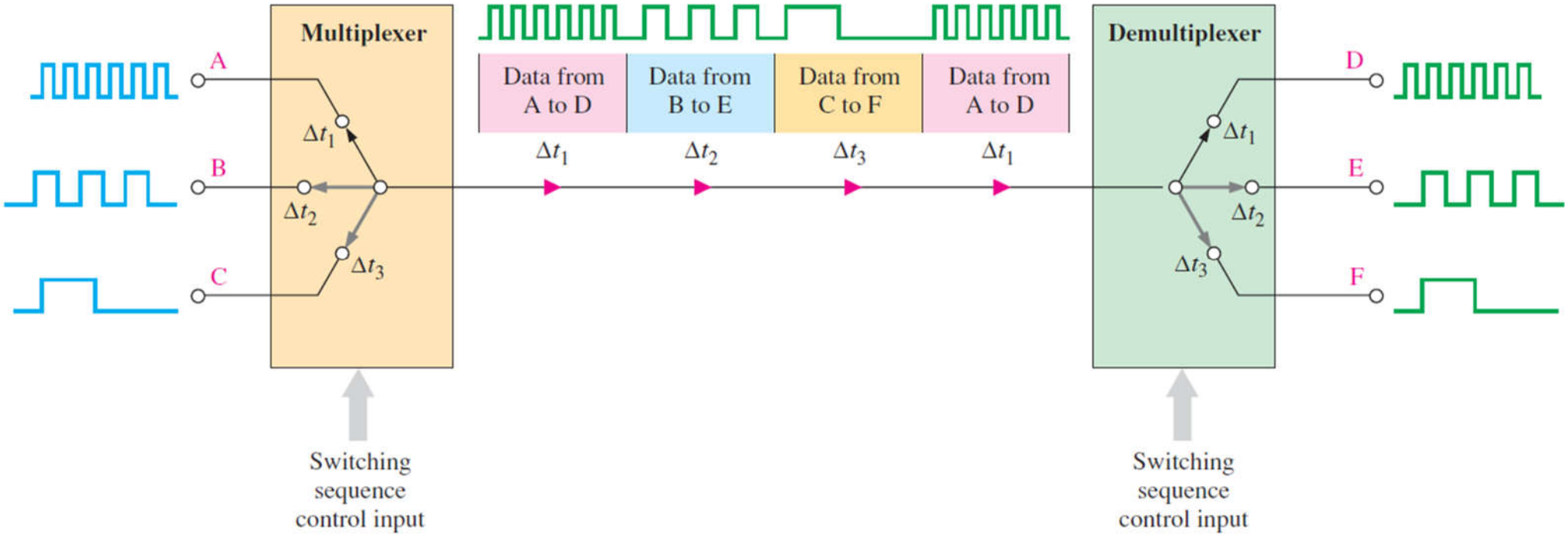


Examples of multiplexer applications

- **Boolean function implementation:** multiplexers can be used to implement logic functions directly from the truth table without simplification.
- **Programmable logic devices**: logic functions can easily be changed by simply changing the 1's and 0's on the Mux inputs.
- **Data routing:** multiplexers can route data from one of several sources to one destination.
- **Parallel-to-Serial conversion:** Many digital systems process binary data in parallel form (all bits simultaneously) because it is faster. When data are to be transmitted over relatively long distances, however, the parallel arrangement is undesirable because it requires a large number of transmission lines. For this reason, binary data or information in parallel form is often converted to a serial form before being transmitted to a remote destination.

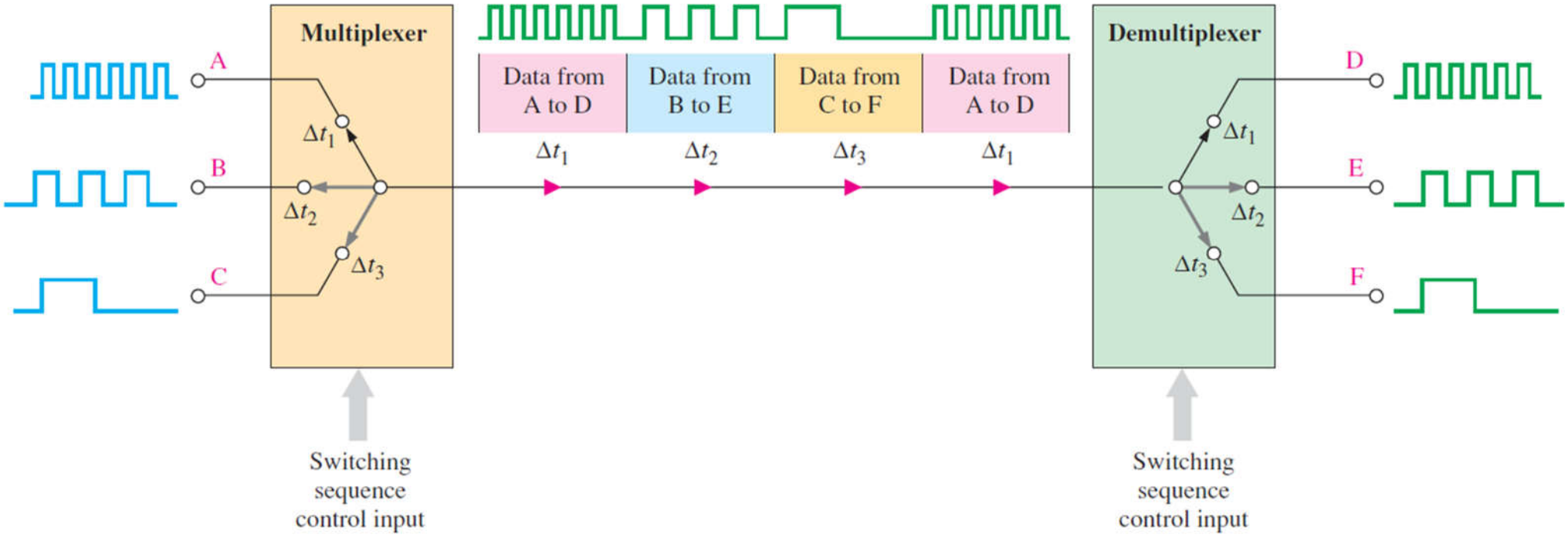
Multiplexing/demultiplexing application

- Multiplexing and demultiplexing are used when data from several sources are to be transmitted over one line to a distant location and redistributed to several destinations.
- The figure below illustrates this type of application where digital data from three sources are sent out along a single line to three terminals at another location.



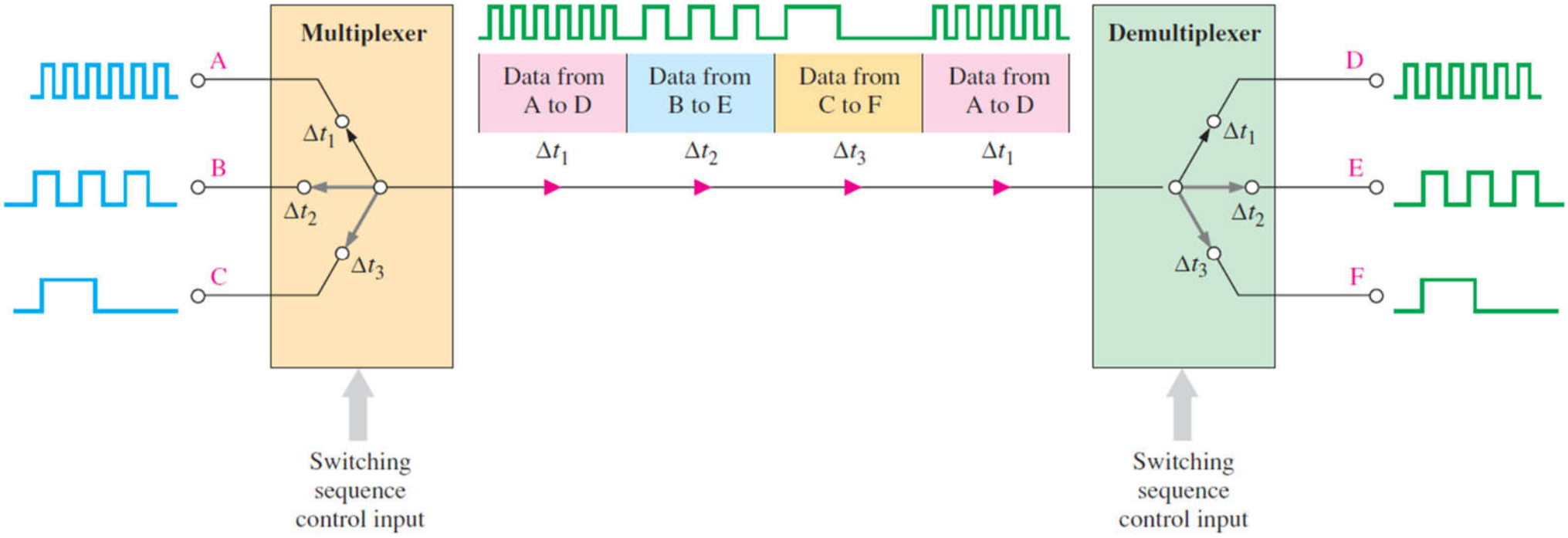
Multiplexing/demultiplexing application

- In this example, data from input A are connected to the output line during time interval Δt_1 and transmitted to the demultiplexer that connects them to output D.
- Then, during interval Δt_2, the multiplexer switches to input B and the demultiplexer switches to output E.
- During interval Δt_3, the multiplexer switches to input C and the demultiplexer switches to output F.



Multiplexing/demultiplexing application

- To summarize, during the first time interval, input A data go to output D. During the second time interval, input B data go to output E. During the third time interval, input C data go to output F. After this, the sequence repeats.
- Because the time is divided up among several sources and destinations where each has its turn to send and receive data, this process is called *time division multiplexing* (TDM).



Acknowledgments

- Credit is acknowledged where credit is due. Please refer to the full list of references.