STANDARD COMBINATIONAL MODULES

- DECODERS
- ENCODERS
- MULTIPLEXERS (Selectors)
- DEMULTIPLEXERS (Distributors)
- SHIFTERS

BINARY DECODERS

HIGH-LEVEL DESCRIPTION:

Inputs: $\underline{x} = (x_{n-1}, \ldots, x_0), \quad x_j \in \{0,1\}$

Enable $E \in \{0,1\}$

Outputs: $\underline{y} = (y_{2^n-1}, \ldots, y_0), \quad y_i \in \{0,1\}$

Function: $y_i = \begin{cases} 1 \ \textbf{if} \ \ (x = i) \ \textbf{and} \ (E = 1) \\ 0 \ \textbf{otherwise} \end{cases}$

$$x = \sum_{j=0}^{n-1} x_j 2^j$$

and

$$i = 0, \ldots, 2^n - 1$$

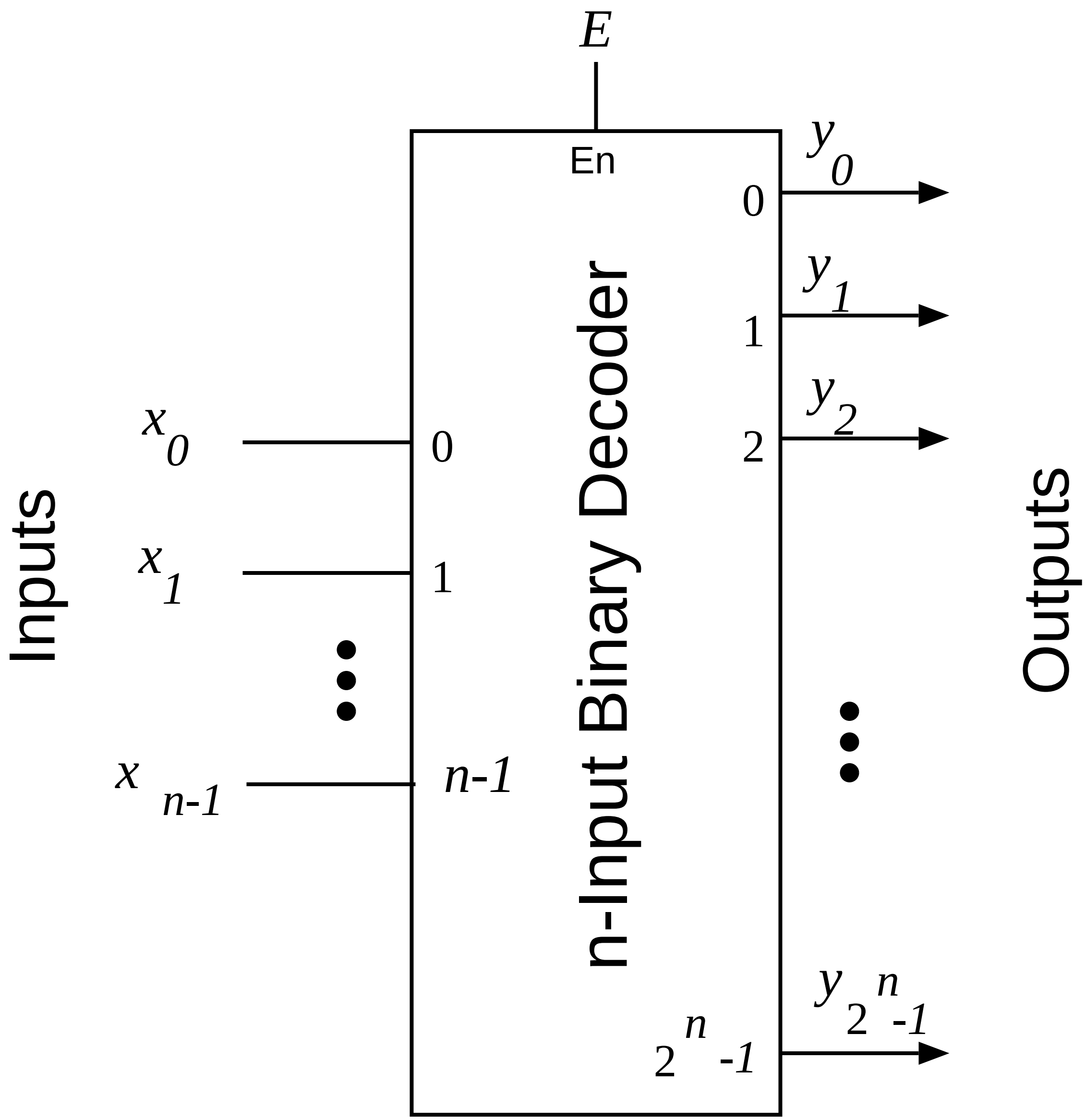


Figure 9.1: n-INPUT BINARY DECODER.

EXAMPLE 9.1: 3-INPUT BINARY DECODER

E	x_2	x_1	x_0	x	y_7	y_6	y_5	y_4	y_3	y_2	y_1	y_0
1	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	1	0	0	0	0	0	0	1	0
1	0	1	0	2	0	0	0	0	0	1	0	0
1	0	1	1	3	0	0	0	0	1	0	0	0
1	1	0	0	4	0	0	0	1	0	0	0	0
1	1	0	1	5	0	0	1	0	0	0	0	0
1	1	1	0	6	0	1	0	0	0	0	0	0
1	1	1	1	7	1	0	0	0	0	0	0	0
0	-	-	-	-	0	0	0	0	0	0	0	0

BINARY SPECIFICATION:

Inputs: $\underline{x} = (x_{n-1}, \ldots, x_0), \quad x_j \in \{0, 1\}$

$E \in \{0, 1\}$

Outputs: $\underline{y} = (y_{2^n-1}, \ldots, y_0), \quad y_i \in \{0, 1\}$

Function: $y_i = E \cdot m_i(\underline{x}) \quad , \quad i = 0, \ldots, 2^n - 1$

EXAMPLE 9.2: IMPLEMENTATION OF 2-INPUT DECODER

$$y_0 = x_1'x_0'E \quad y_1 = x_1'x_0E \quad y_2 = x_1x_0'E \quad y_3 = x_1x_0E$$

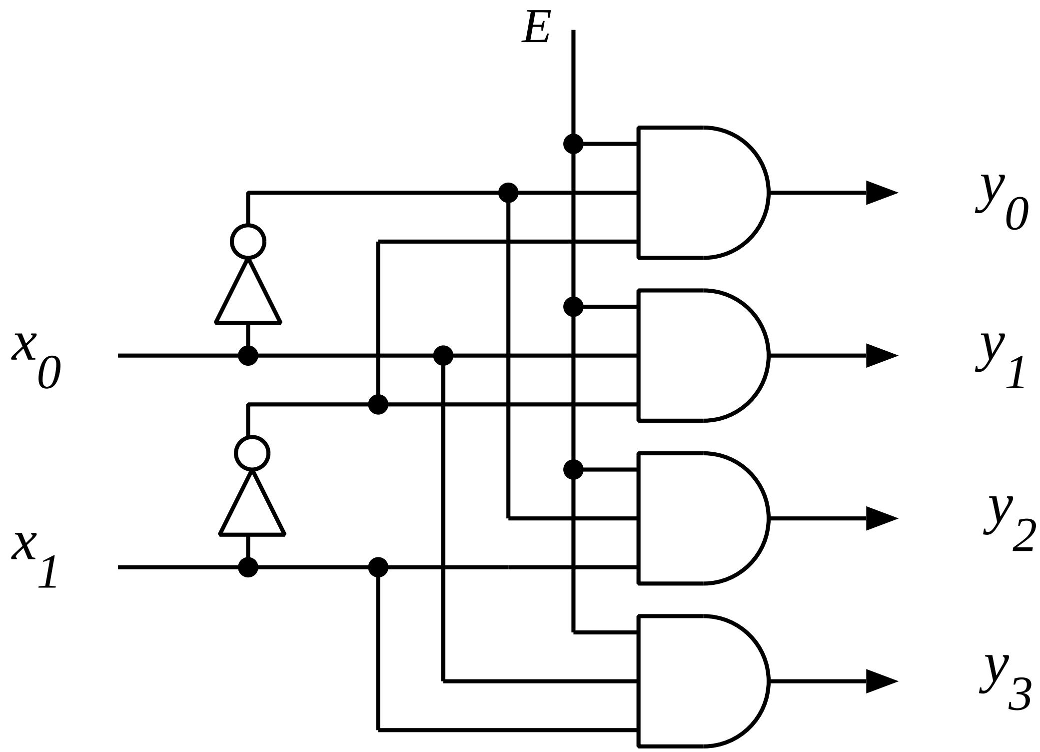


Figure 9.2: GATE NETWORK IMPLEMENTATION OF 2-INPUT BINARY DECODER.

DECODER USES

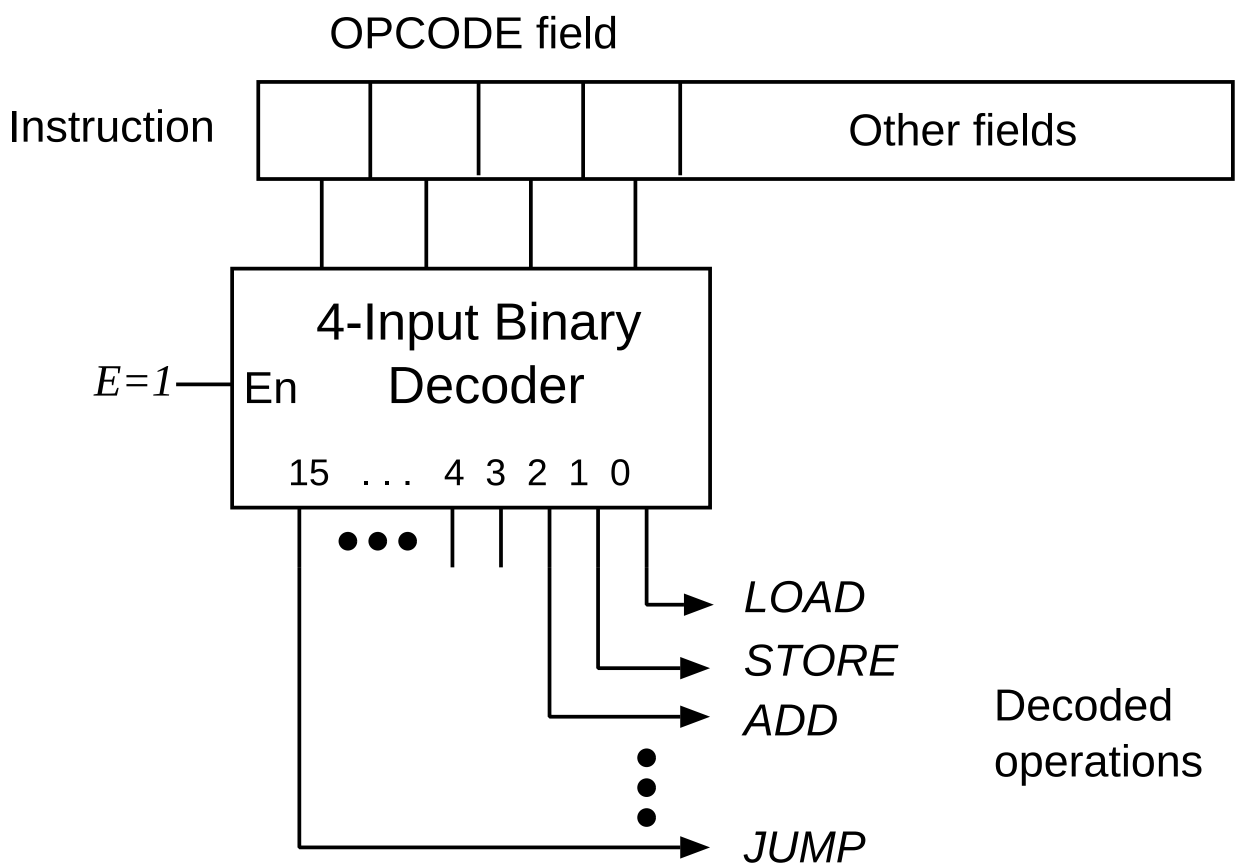


Figure 9.3: OPERATION DECODING.

DECODER USES

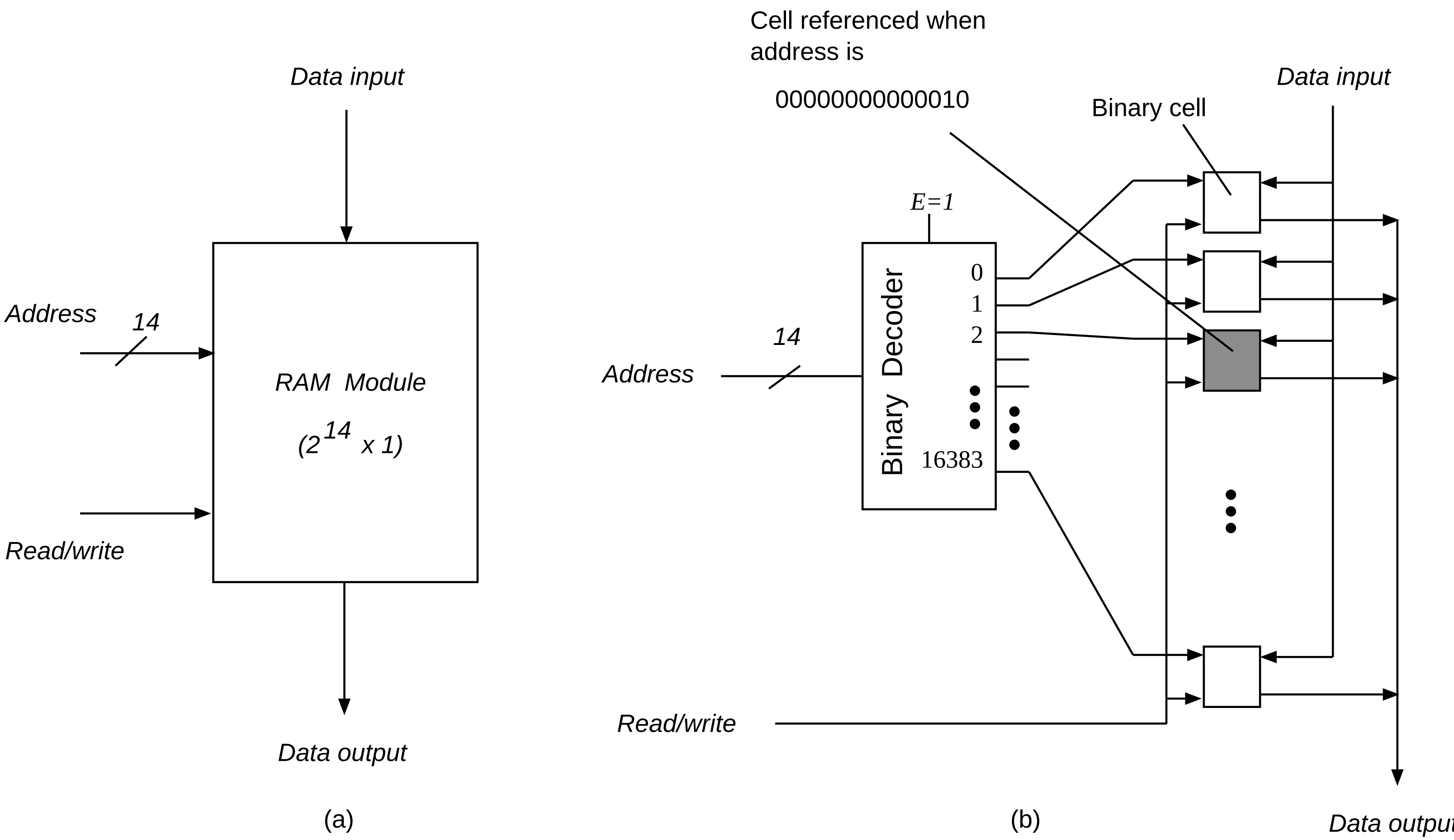


Figure 9.4: RANDOM ACCESS MEMORY (RAM): a) MODULE; b) ADDRESSING OF BINARY CELLS.

BINARY DECODER AND OR GATE

- UNIVERSAL

Example 9.5:

$x_2x_1x_0$	z_2	z_1	z_0
000	0	1	0
001	1	0	0
010	0	0	1
011	0	1	0
100	0	0	1
101	1	0	1
110	0	0	0
111	1	0	0

$$
\begin{aligned}
(y_7, \ldots, y_0) &= \mathrm{DEC}(x_2, x_1, x_0, 1) \\
z_2(x_2, x_1, x_0) &= y_1 + y_5 + y_7 \\
z_1(x_2, x_1, x_0) &= y_0 + y_3 \\
z_0(x_2, x_1, x_0) &= y_2 + y_4 + y_5
\end{aligned}
$$

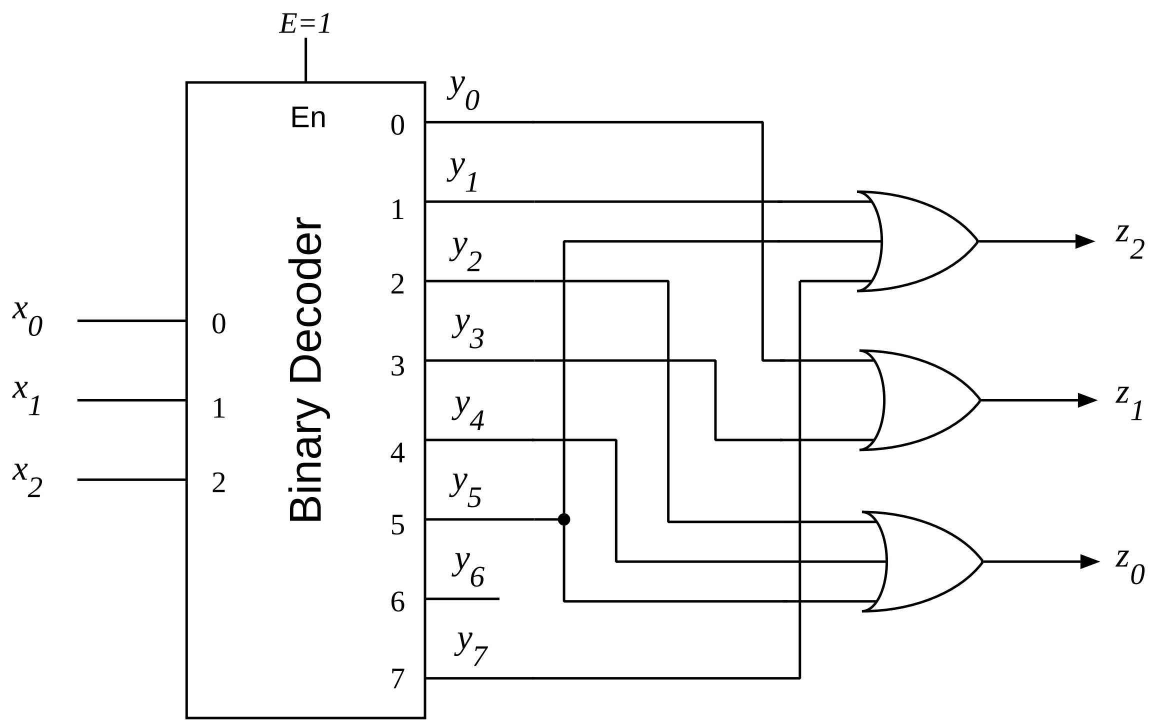


Figure 9.5: NETWORK IN EXAMPLE 9.5

DECODER NETWORKS: COINCIDENT DECODING

$$\underline{x} = (\underline{x}_{\text{left}}, \underline{x}_{\text{right}})$$

$$\begin{aligned} \underline{x}_{\text{left}} &= (x_7, x_6, x_5, x_4) \\ \underline{x}_{\text{right}} &= (x_3, x_2, x_1, x_0) \end{aligned}$$

$$x = 2^4 \times x_{\text{left}} + x_{\text{right}}$$

$$\begin{aligned} \underline{y} &= DEC(\underline{x}_{\text{left}}) \\ \underline{w} &= DEC(\underline{x}_{\text{right}}) \end{aligned}$$

$$z_i = AND(y_s, w_t)$$

$$i = 2^4 \times s + t$$

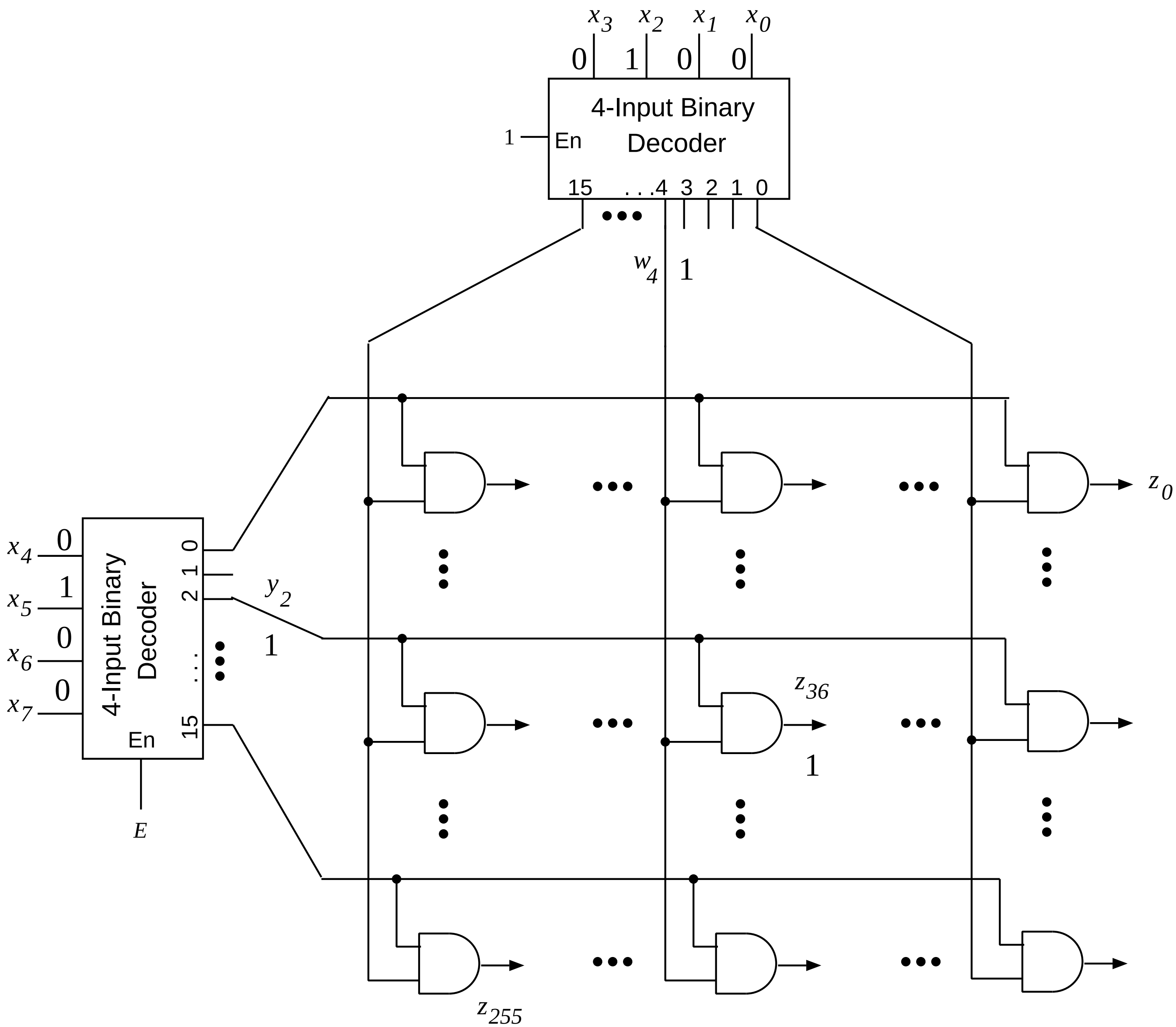


Figure 9.6: 8-INPUT COINCIDENT DECODER.

n-INPUT COINCIDENT DECODER

$$\underline{y} = \text{DEC}(\underline{x}_{\text{left}}, E)$$

$$\underline{w} = \text{DEC}(\underline{x}_{\text{right}}, 1)$$

$$\underline{z} = (\text{AND}(y_{2^{n/2}-1}, w_{2^{n/2}-1}), \ldots, \text{AND}(y_s, w_t), \ldots, \text{AND}(y_0, w_0))$$

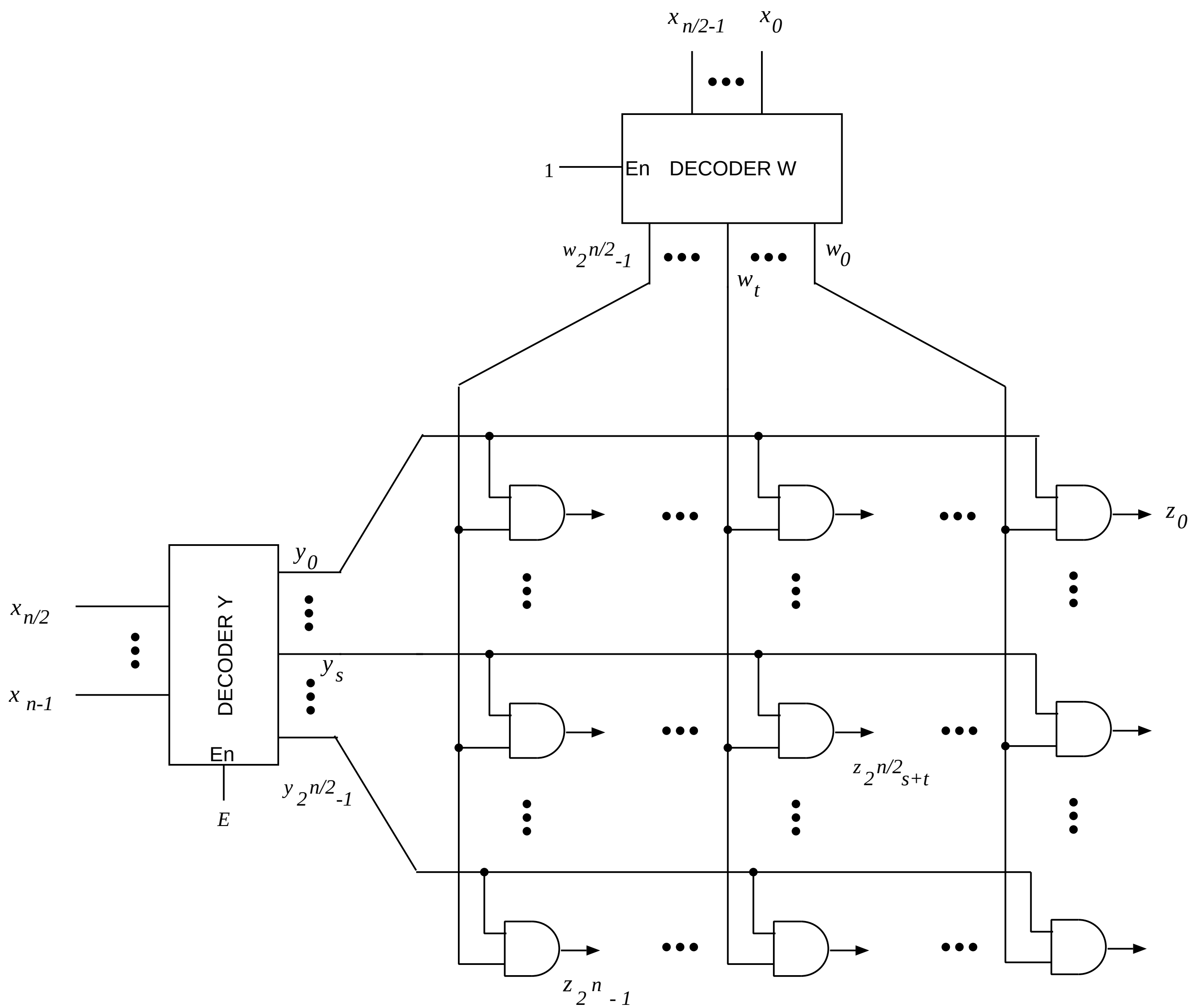


Figure 9.7: n-INPUT COINCIDENT DECODER.

TREE DECODING

$$\underline{x} = (\underline{x}_{\text{left}}, \underline{x}_{\text{right}})$$

$$\underline{x}_{\text{left}} = (x_3, x_2)$$

$$\underline{x}_{\text{right}} = (x_1, x_0)$$

4-INPUT TREE DECODER

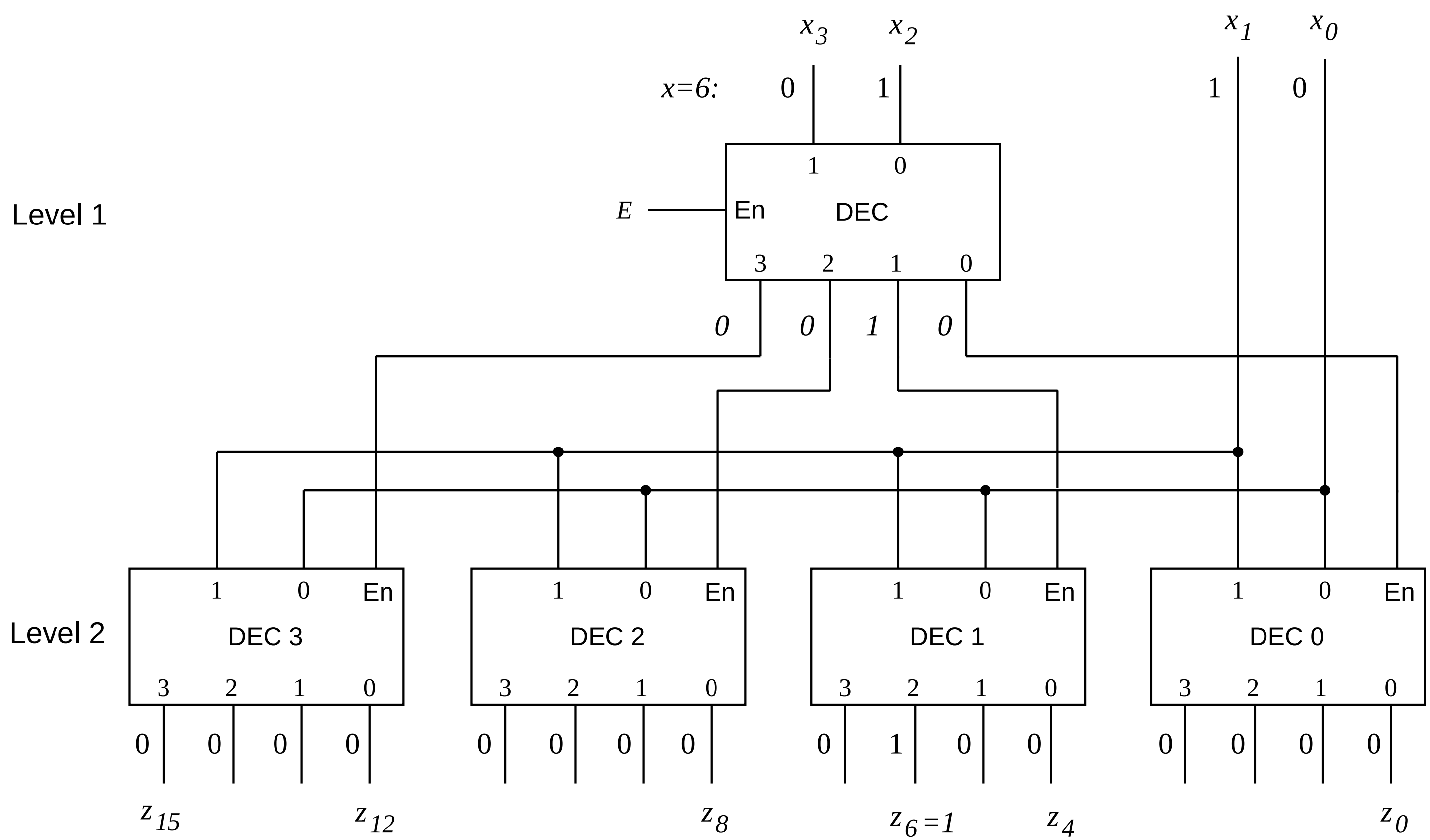


Figure 9.8: 4-INPUT TREE DECODER.

n-INPUT TREE DECODERr

$$\underline{w} = \mathrm{DEC}(\underline{x}_{\mathrm{left}}, E)$$

$$\underline{z} = (\mathrm{DEC}(\underline{x}_{\mathrm{right}}, w_{2^{n/2}-1}), \ldots, \mathrm{DEC}(\underline{x}_{\mathrm{right}}, w_t), \ldots, \mathrm{DEC}(\underline{x}_{\mathrm{right}}, w_0))$$

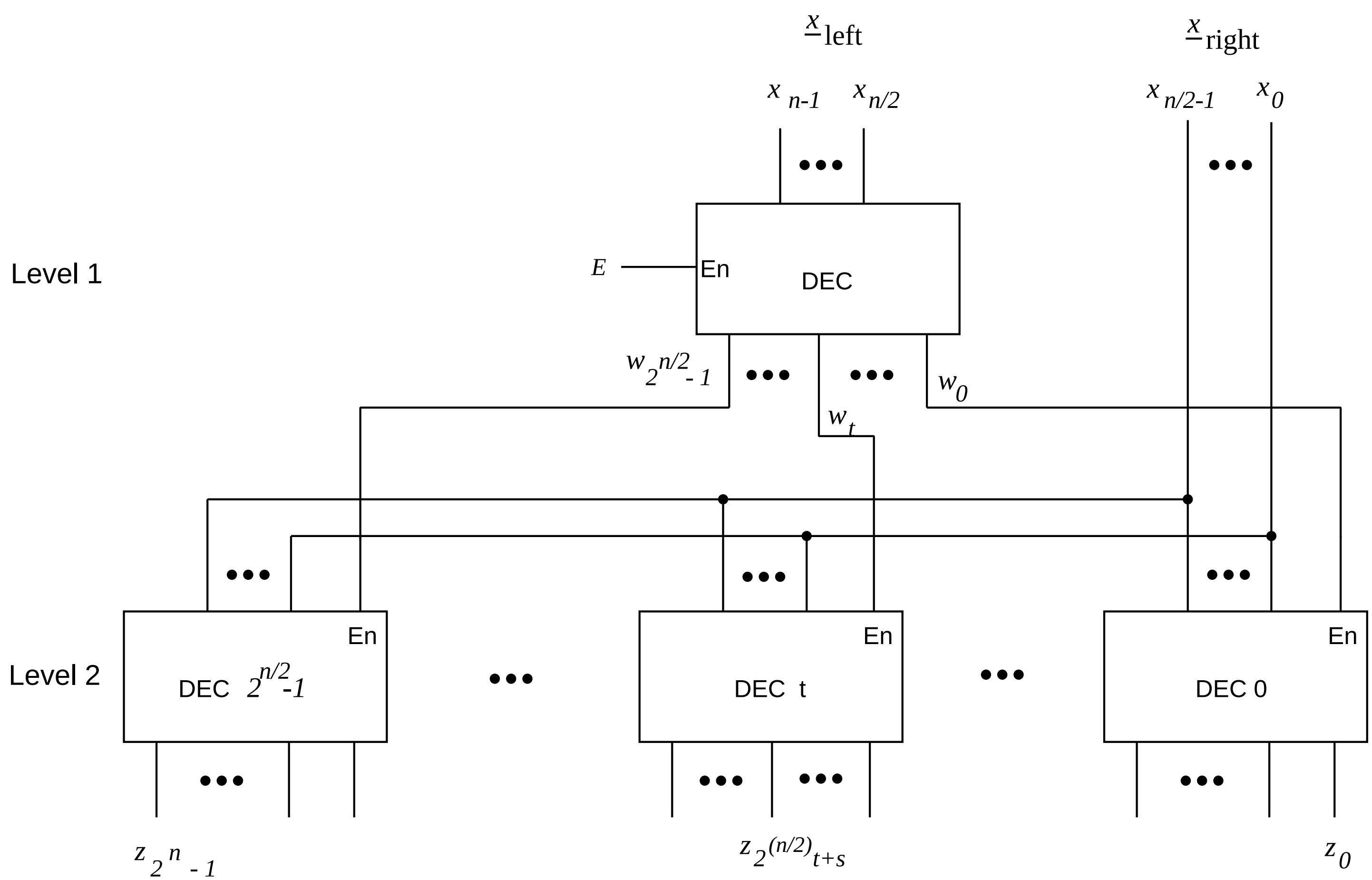


Figure 9.9: n-INPUT TWO-LEVEL TREE DECODER.

COMPARISON OF DECODER NETWORKS

	Coincident	Tree
Decoder modules	2	$2^k + 1$
AND gates	2^{2k}	–
Load per network input	1 decoder input	2^k decoder inputs (max)
Fanout per decoder output	2^k AND inputs	1 enable input
Number of module inputs (related to number of connections)	$2k + 2 + 2^{2k+1}$	$1 + k + 2^k + k2^k$
Delay	$t_{\mathrm{decoder}} + t_{\mathrm{AND}}$	$2t_{\mathrm{decoder}}$

EXAMPLE 9.6: 6-INPUT DECODER

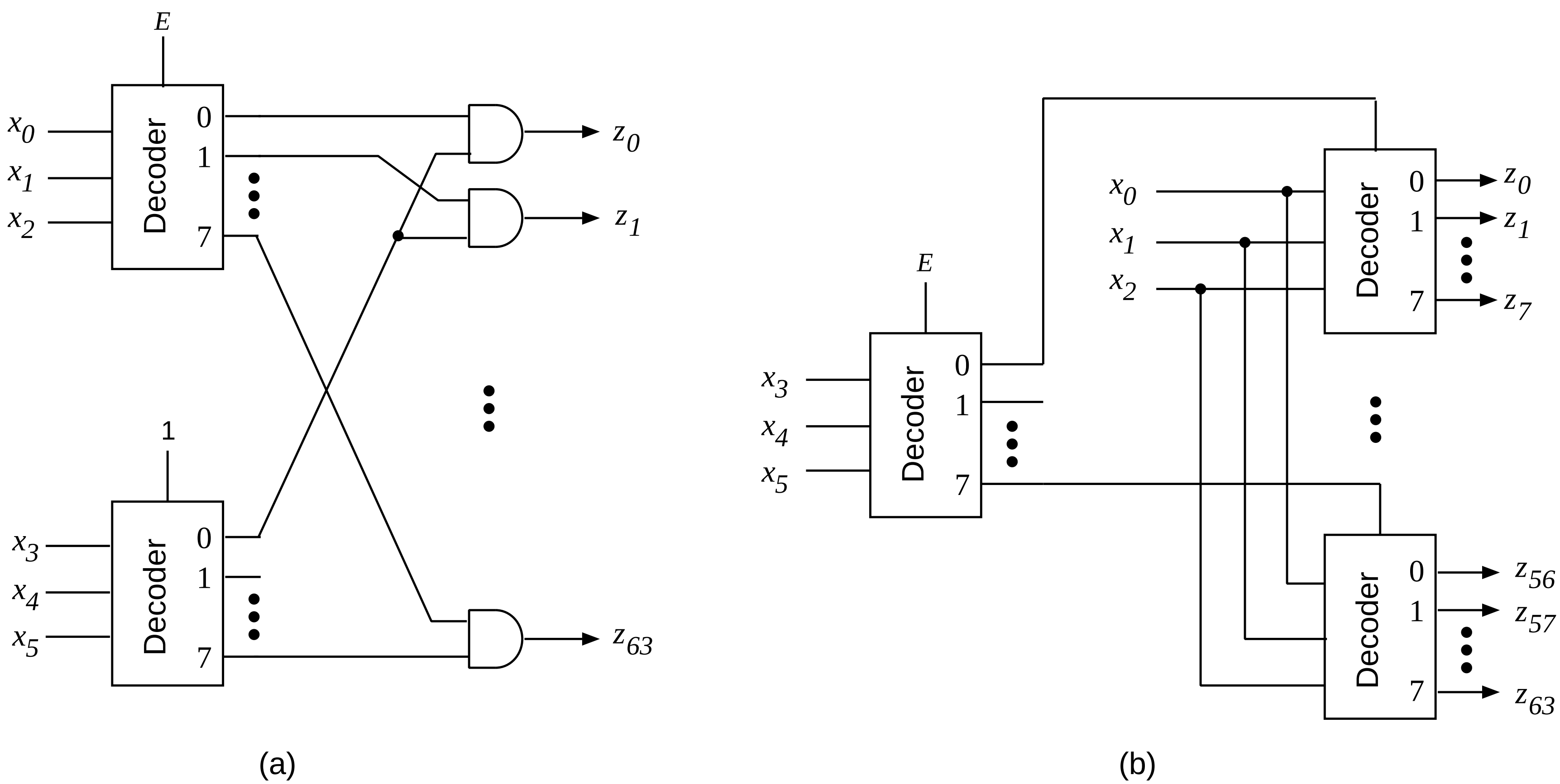


Figure 9.10: IMPLEMENTATION OF 6-INPUT DECODER. a) COINCIDENT DECODER. b) TREE DECODER.

EXAMPLE 9.6 (cont.)

	Coincident	Tree
Decoder modules	2	9
AND gates	64	–
Load per network input	1 decoder input	8 decoder inputs (max)
Fanout per decoder output	8 AND inputs	1 enable input
Number of module inputs	136	36
Delay	$3d$	$4d$

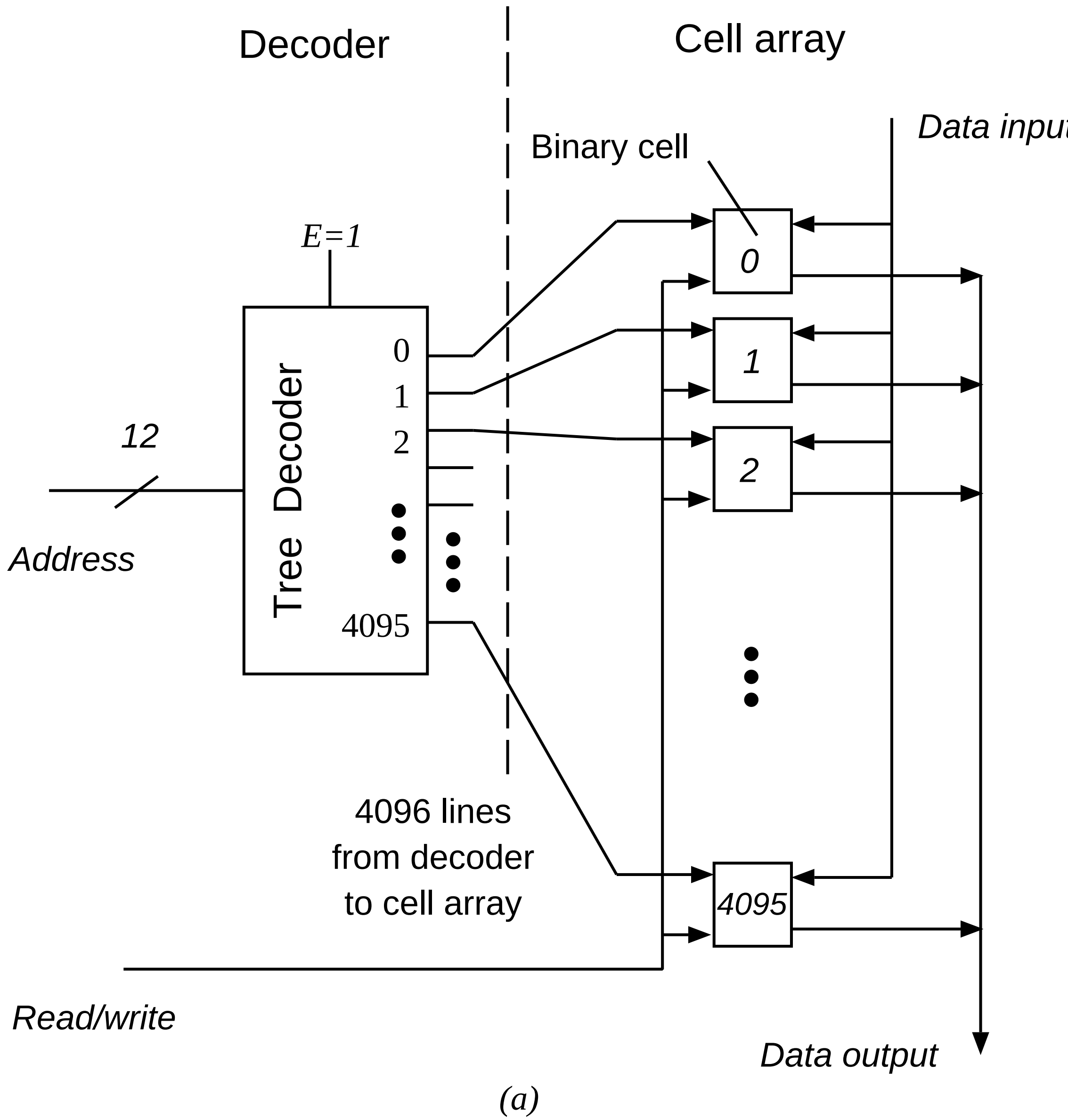


Figure 9.11: a) SYSTEM WITH TREE DECODER.

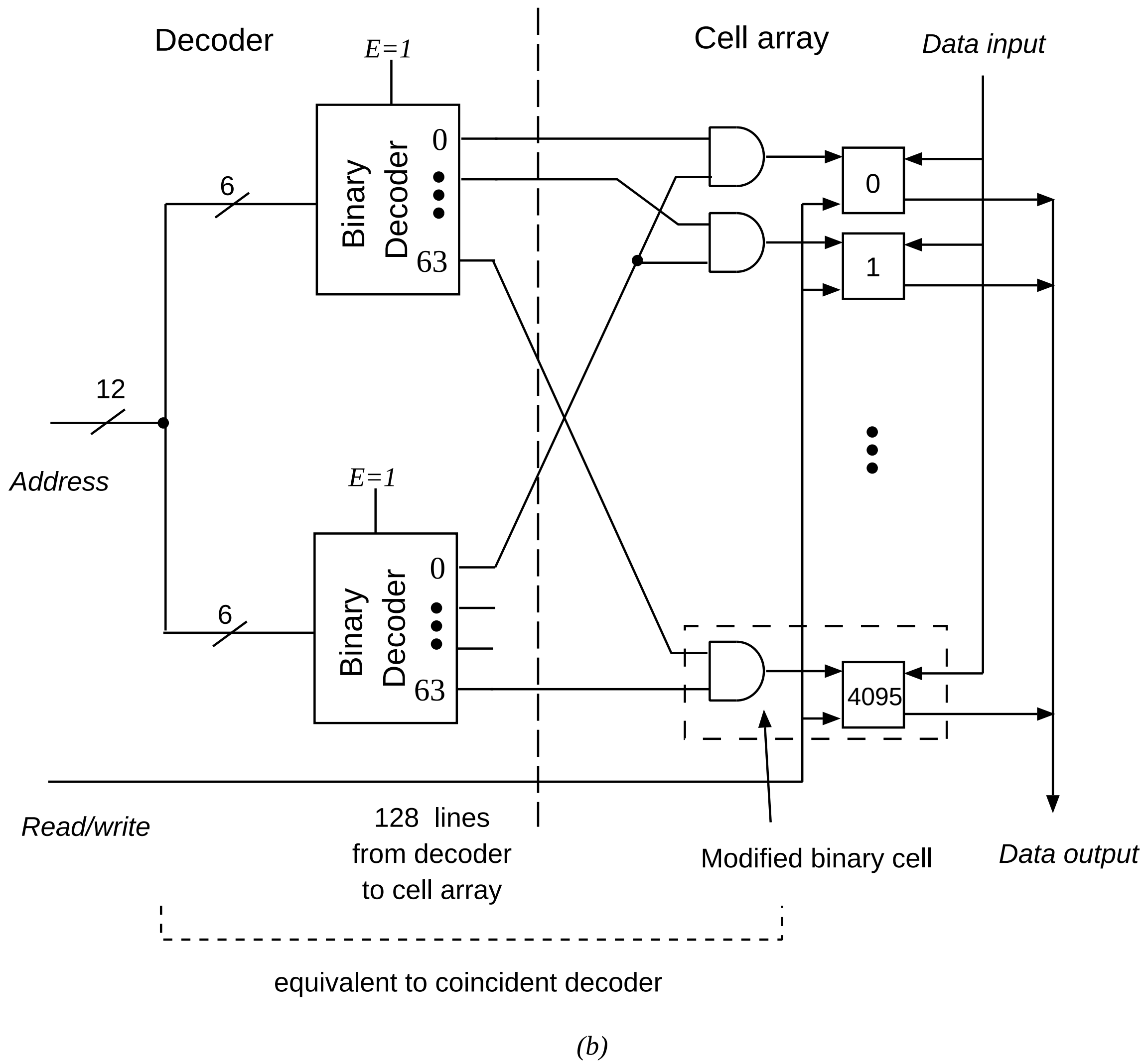


Figure 9.11: b) SYSTEM WITH COINCIDENT DECODER.

BINARY ENCODERS

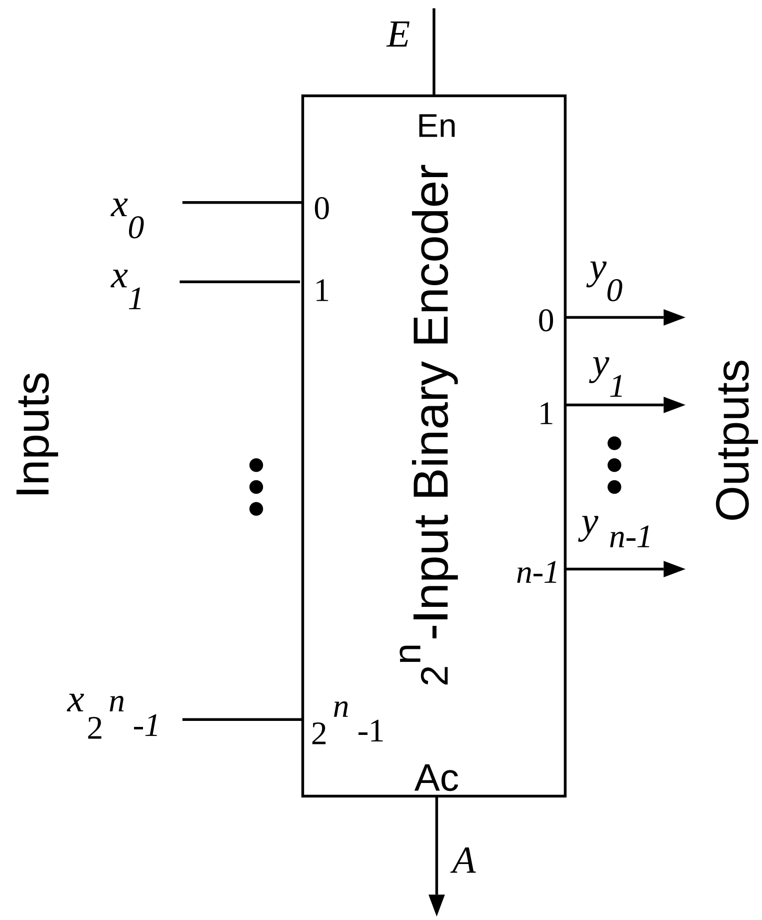


Figure 9.12: 2^n-INPUT BINARY ENCODER.

BINARY ENCODER: HIGH-LEVEL SPECIFICATION

Inputs: $\underline{x} = (x_{2^n-1}, \ldots, x_0), \quad x_i \in \{0,1\}$, with at most one $x_i = 1$
$E \in \{0,1\}$

Outputs: $\underline{y} = (y_{n-1}, \ldots, y_0), \quad y_j \in \{0,1\}$
$A \in \{0,1\}$

Function: $y = \begin{cases} i & \textbf{if} \;\; (x_i = 1) \textbf{ and } (E = 1) \\ 0 & \textbf{otherwise} \end{cases}$

$A = \begin{cases} 1 & \textbf{if} \;\; (\text{some } x_i = 1) \textbf{ and } (E = 1) \\ 0 & \textbf{otherwise} \end{cases}$

$$y = \sum_{j=0}^{n-1} y_j 2^j$$

and

$$i = 0, \ldots, 2^n - 1$$

EXAMPLE 9.7: FUNCTION OF AN 8-INPUT BINARY ENCODER

E	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	y	y_2	y_1	y_0	A
1	0	0	0	0	0	0	0	1	0	0	0	0	1
1	0	0	0	0	0	0	1	0	1	0	0	1	1
1	0	0	0	0	0	1	0	0	2	0	1	0	1
1	0	0	0	0	1	0	0	0	3	0	1	1	1
1	0	0	0	1	0	0	0	0	4	1	0	0	1
1	0	0	1	0	0	0	0	0	5	1	0	1	1
1	0	1	0	0	0	0	0	0	6	1	1	0	1
1	1	0	0	0	0	0	0	0	7	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	-	-	-	-	-	-	-	-	0	0	0	0	0

BINARY SPECIFICATION OF ENCODER

Inputs: $\underline{x} = (x_{2^n-1}, \ldots, x_0), \quad x_i \in \{0,1\}$, with at most one $x_i = 1$

$E \in \{0,1\}$

Outputs: $\underline{y} = (y_{n-1}, \ldots, y_0), \quad y_j \in \{0,1\}$

$A \in \{0,1\}$

Function: $y_j = E \cdot \Sigma(x_k), \quad j = 0, \ldots, n-1$

$A = E \cdot \Sigma(x_i), \quad i = 0, \ldots, 2^n - 1$

EXAMPLE 9.8

$$y_0 = E \cdot (x_1 + x_3 + x_5 + x_7)$$

$$y_1 = E \cdot (x_2 + x_3 + x_6 + x_7)$$

$$y_2 = E \cdot (x_4 + x_5 + x_6 + x_7)$$

$$A = E \cdot (x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7)$$

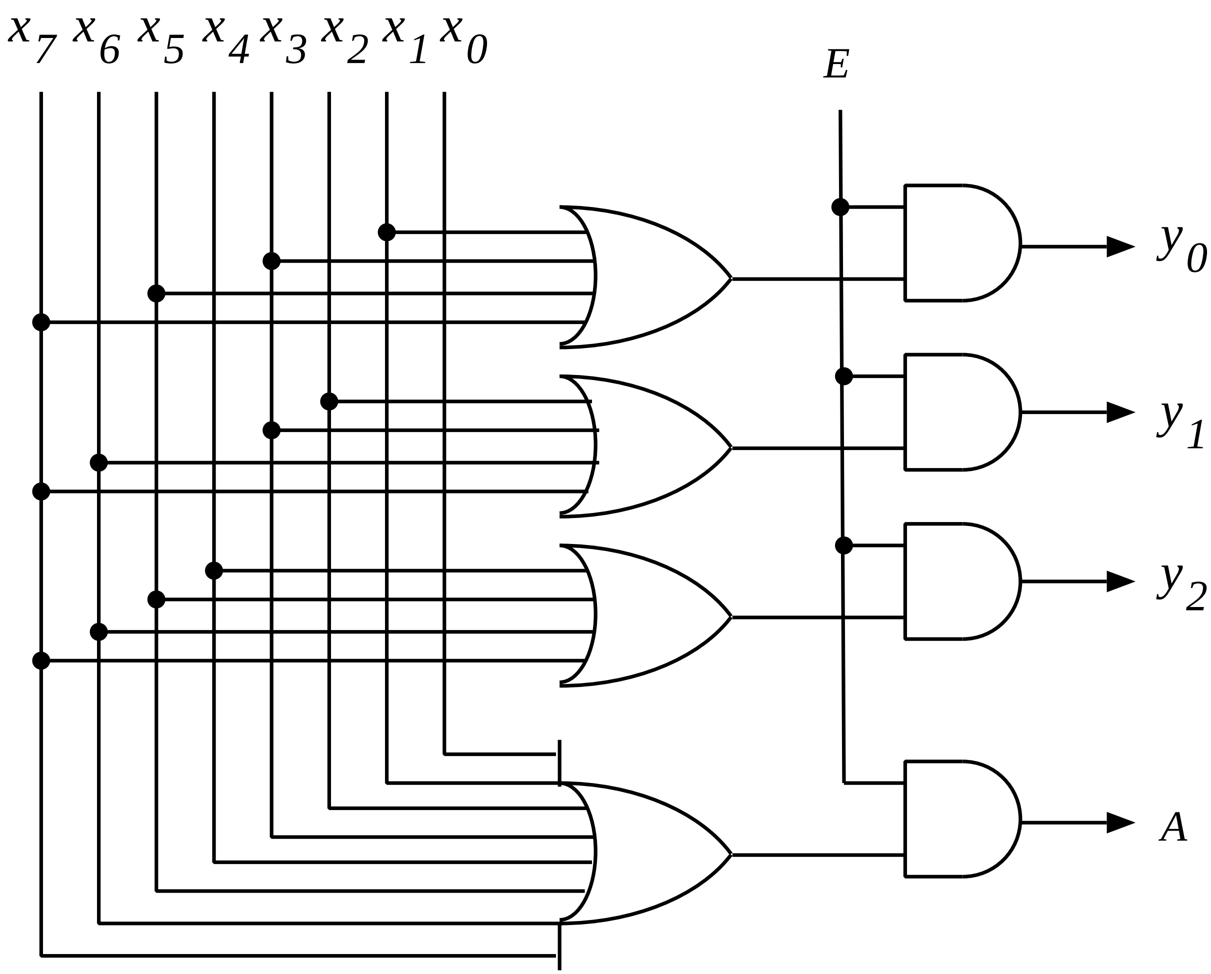


Figure 9.13: IMPLEMENTATION OF AN 8-INPUT BINARY ENCODER.

USES OF BINARY ENCODERS

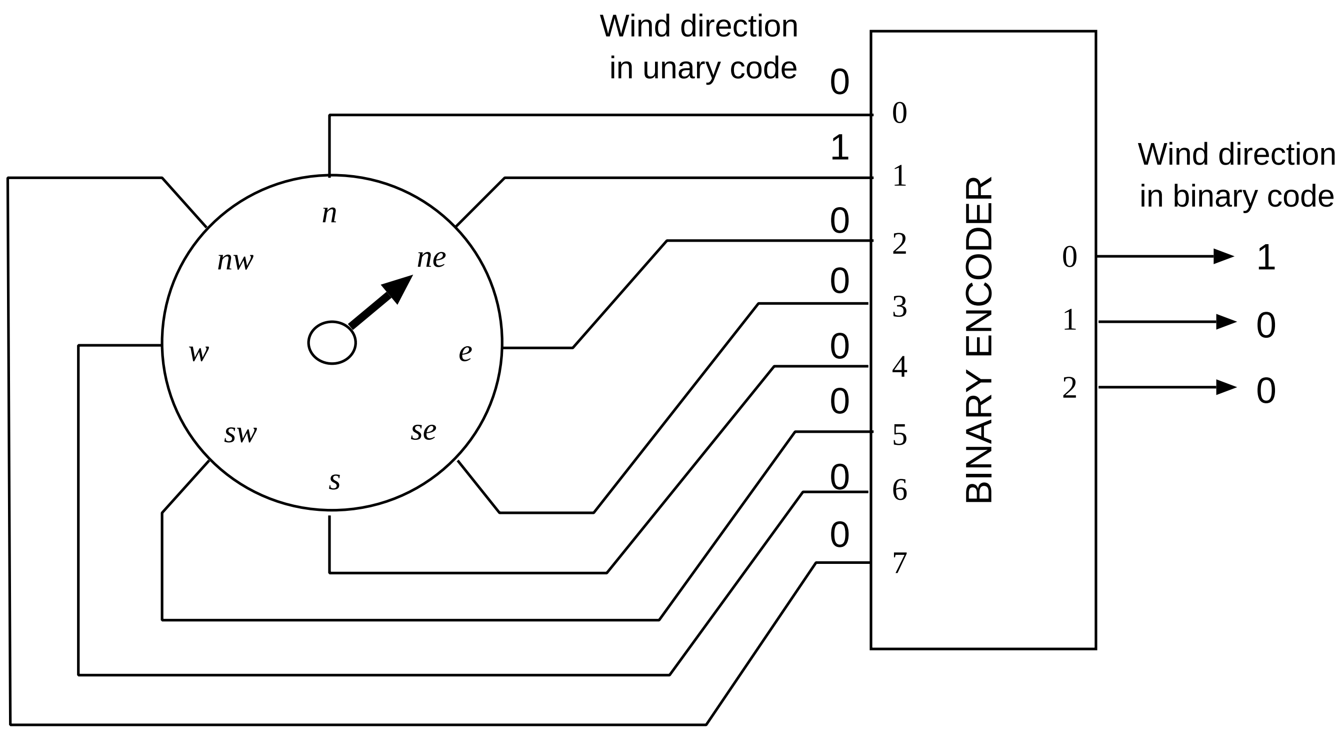


Figure 9.14: WIND DIRECTION ENCODER.

PRIORITY ENCODERS: HIGH-LEVEL DESCRIPTION

Inputs: $\underline{x} = (x_{2^n-1}, \ldots, x_0), \quad x_i \in \{0,1\}$
Outputs: $\underline{y} = (y_{n-1}, \ldots, y_0), \quad y_j \in \{0,1\}$

Function:

$$y = \begin{cases} i \ \textbf{if} \ \ (x_i = 1) \ \textbf{and} \ (x_k = 0, \ k > i) \ \textbf{and} \ (E = 1) \\ 0 \ \textbf{otherwise} \end{cases}$$

$$A = \begin{cases} 1 \ \textbf{if} \ \ (\text{some } x_i = 1) \ \textbf{and} \ (E = 1) \\ 0 \ \textbf{otherwise} \end{cases}$$

$$y = \sum_{j=0}^{n-1} y_j 2^j$$

and

$$i, k \in \{0, 1, \ldots, 2^n - 1\}$$

8-INPUT PRIORITY ENCODER

E	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	y_2	y_1	y_0	A
1	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	0	0	1	-	0	0	1	1
1	0	0	0	0	0	1	-	-	0	1	0	1
1	0	0	0	0	1	-	-	-	0	1	1	1
1	0	0	0	1	-	-	-	-	1	0	0	1
1	0	0	1	-	-	-	-	-	1	0	1	1
1	0	1	-	-	-	-	-	-	1	1	0	1
1	1	-	-	-	-	-	-	-	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0	0	0
0	-	-	-	-	-	-	-	-	0	0	0	0

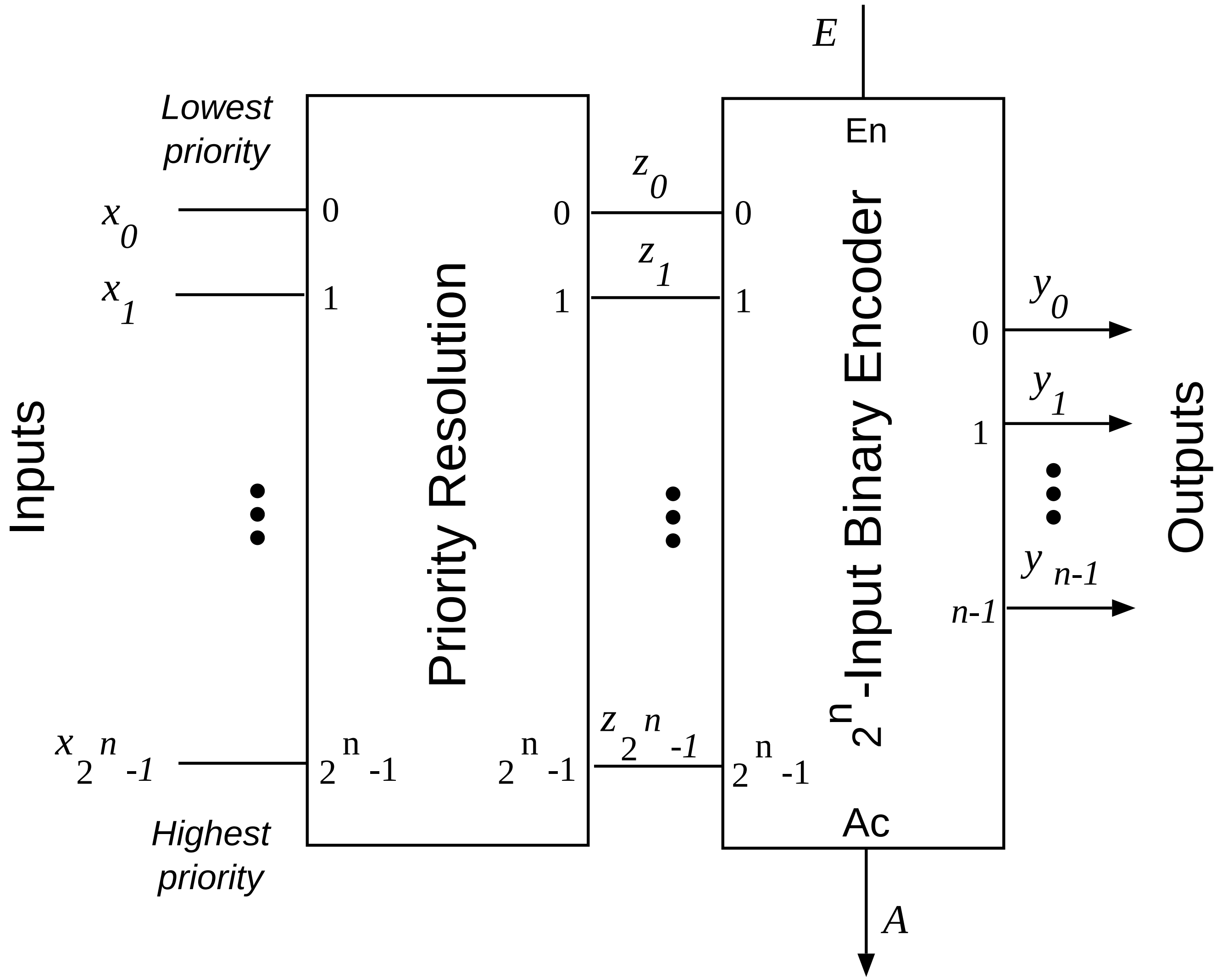


Figure 9.15: PRIORITY ENCODER.

PRIORITY RESOLUTION: HIGH-LEVEL AND BINARY-LEVEL DESCRIPTION

Inputs: $\underline{x} = (x_{2^n-1}, \ldots, x_0), \quad x_i \in \{0,1\}$
Outputs: $\underline{z} = (z_{2^n-1}, \ldots, z_0), \quad z_i \in \{0,1\}$

Function: $z_i = \begin{cases} 1 \ \textbf{if} \ \ (x_i = 1) \ \textbf{and} \ (x_k = 0, \quad k > i) \\ 0 \ \textbf{otherwise} \end{cases}$
with $i, k = 0, 1, \ldots, 2^n - 1$

- BINARY DESCRIPTION:

$$z_i = x'_{2^n-1} x'_{2^n-2} \ldots x'_{i+1} x_i \quad , \quad i = 0, 1, \ldots, 2^n - 1$$

OR ITERATIVELY

$$\begin{aligned} c_{i-1} &= c_i + x_i \\ z_i &= c'_i x_i \end{aligned}$$

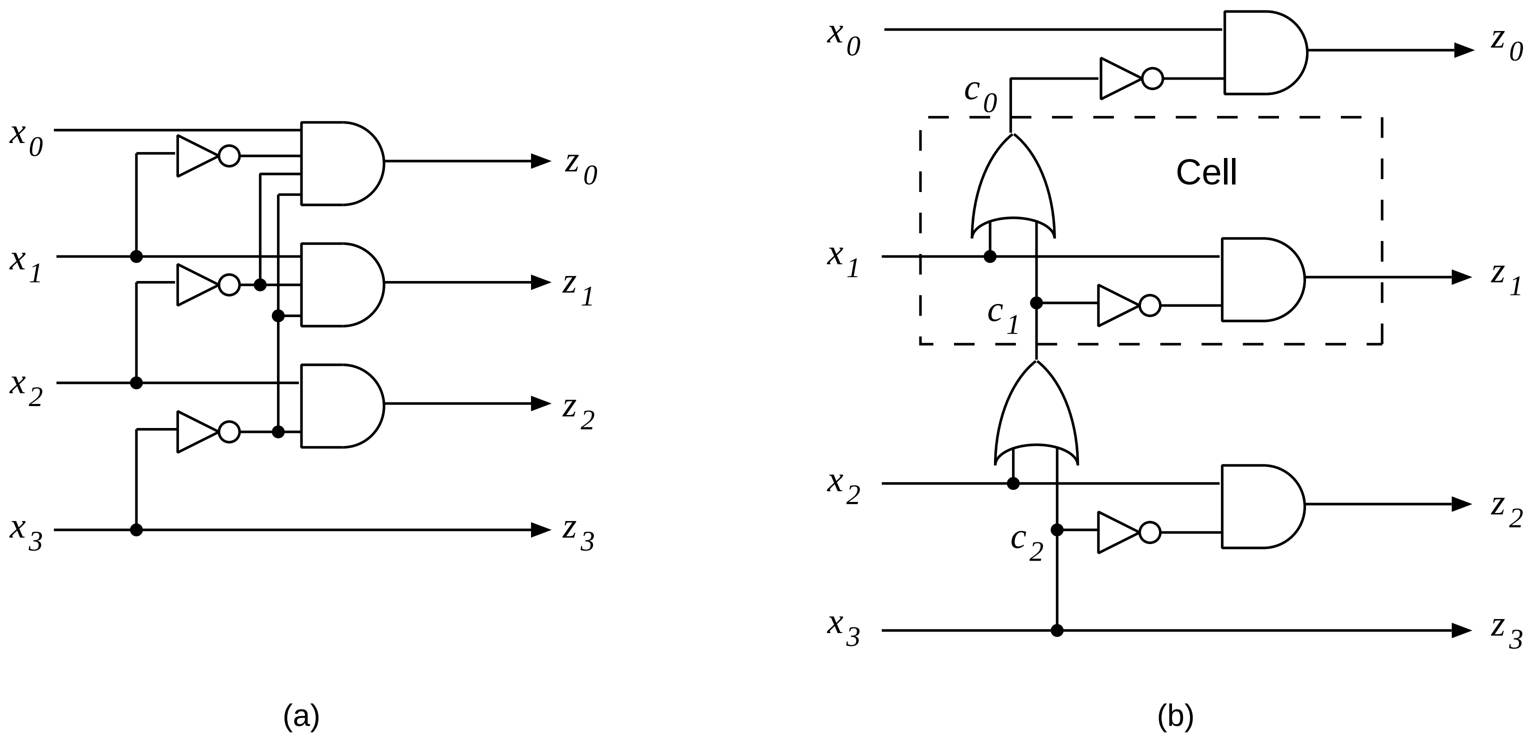


Figure 9.16: 4-BIT PRIORITY RESOLUTION NETWORKS: a) PARALLEL; b) ITERATIVE.

USES OF PRIORITY ENCODERS

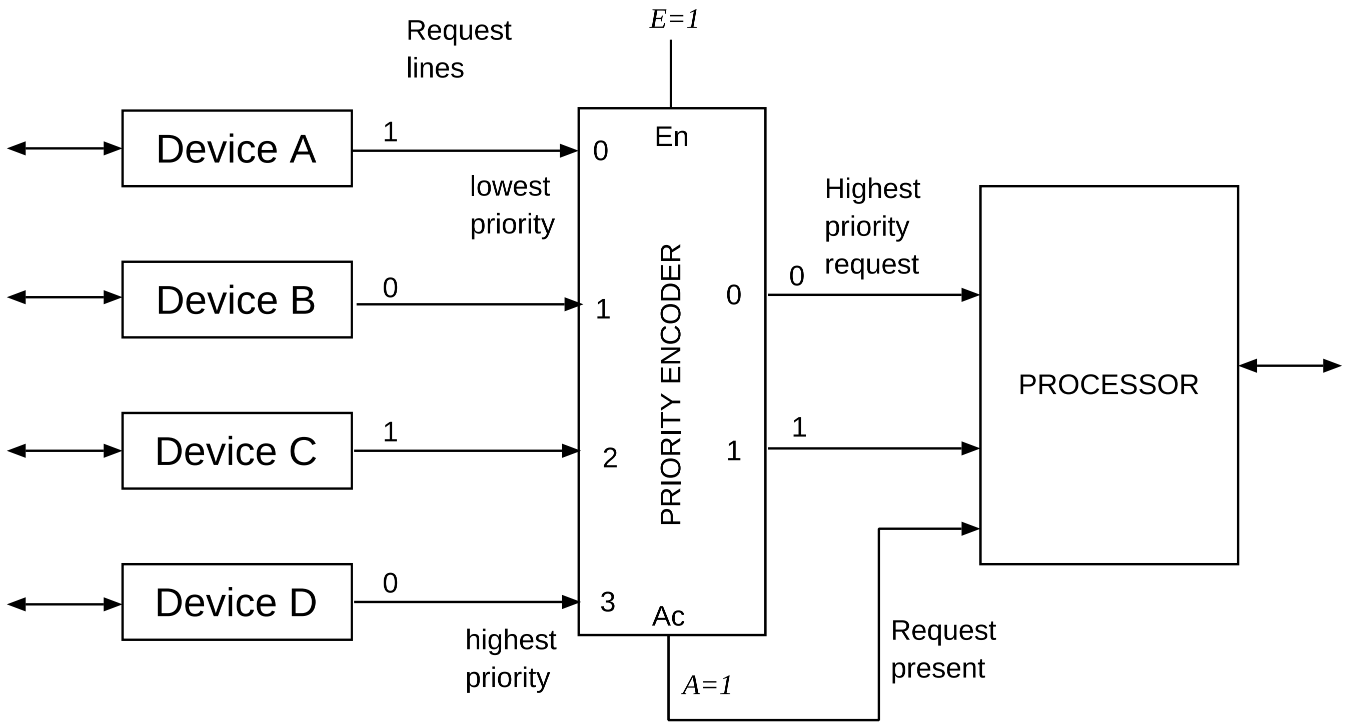


Figure 9.17: RESOLVING INTERRUPT REQUESTS USING A PRIORITY ENCODER.

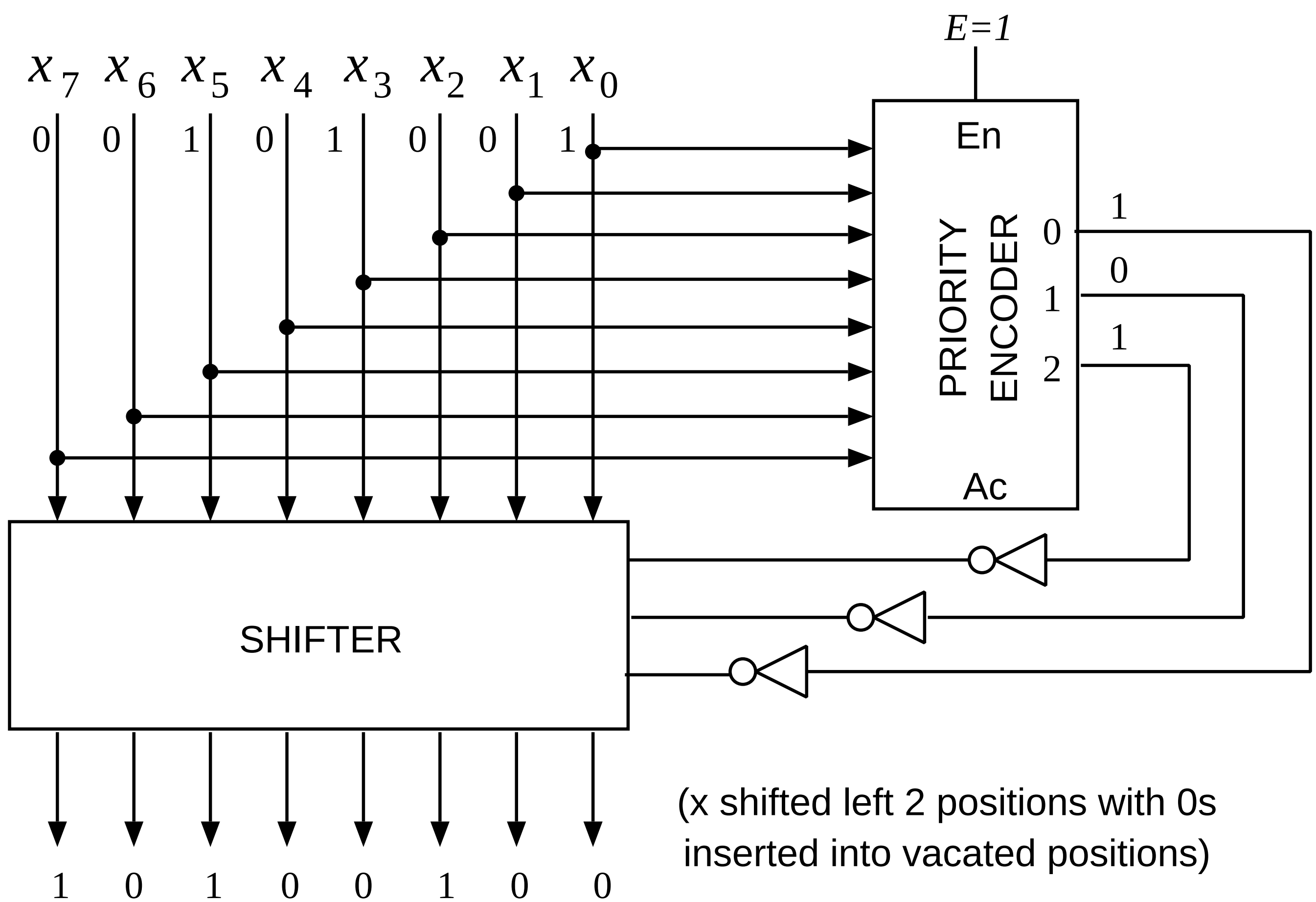


Figure 9.18: DETECTING THE LEFTMOST 1 IN A BIT-VECTOR AND REMOVING LEADING ZEROES.

MULTIPLEXERS (selectors)

- HIGH-LEVEL AND BINARY-LEVEL DESCRIPTION

Inputs: $\underline{x} = (x_{2^n-1}, \ldots, x_0), \quad x_i \in \{0,1\}$
$\underline{s} = (s_{n-1}, \ldots, s_0), \quad s_j \in \{0,1\}$
$E \in \{0,1\}$

Outputs: $z \in \{0,1\}$

Function: $z = \begin{cases} x_s & \textbf{if} \quad E = 1 \\ 0 & \textbf{if} \quad E = 0 \end{cases}$

$$s = \sum_{j=0}^{n-1} s_j 2^j$$

$$z = E \cdot \left[\sum_{i=0}^{2^n-1} x_i \cdot m_i(\underline{s}) \right]$$

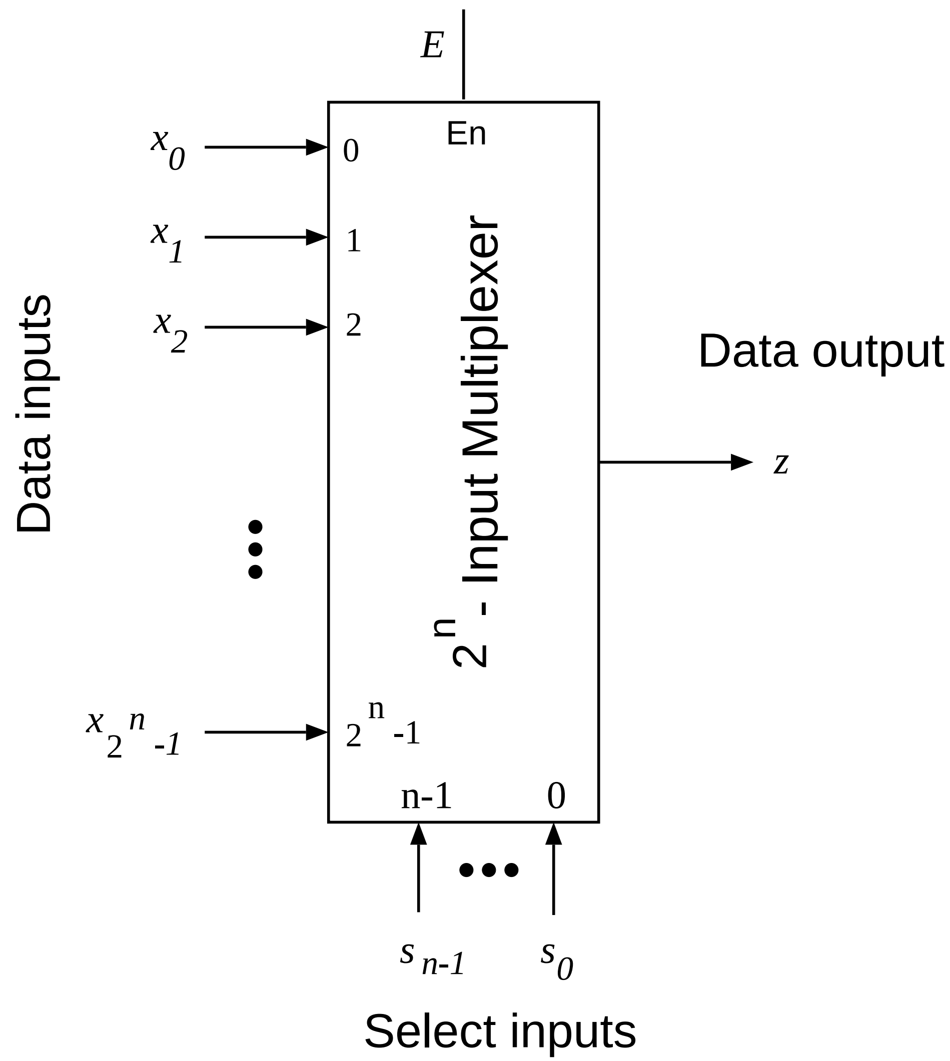


Figure 9.19: 2^n-INPUT MULTIPLEXER.

EXAMPLE 9.11: 4-INPUT MULTIPLEXER

E	s_1	s_0	z
1	0	0	x_0
1	0	1	x_1
1	1	0	x_2
1	1	1	x_3
0	-	-	0

$$\begin{aligned} z &= E \cdot (x_0 m_0(s_1, s_0) + x_1 m_1(s_1, s_0) + x_2 m_2(s_1, s_0) + x_3 m_3(s_1, s_0)) \\ &= E \cdot (x_0 s_1' s_0' + x_1 s_1' s_0 + x_2 s_1 s_0' + x_3 s_1 s_0) \end{aligned}$$

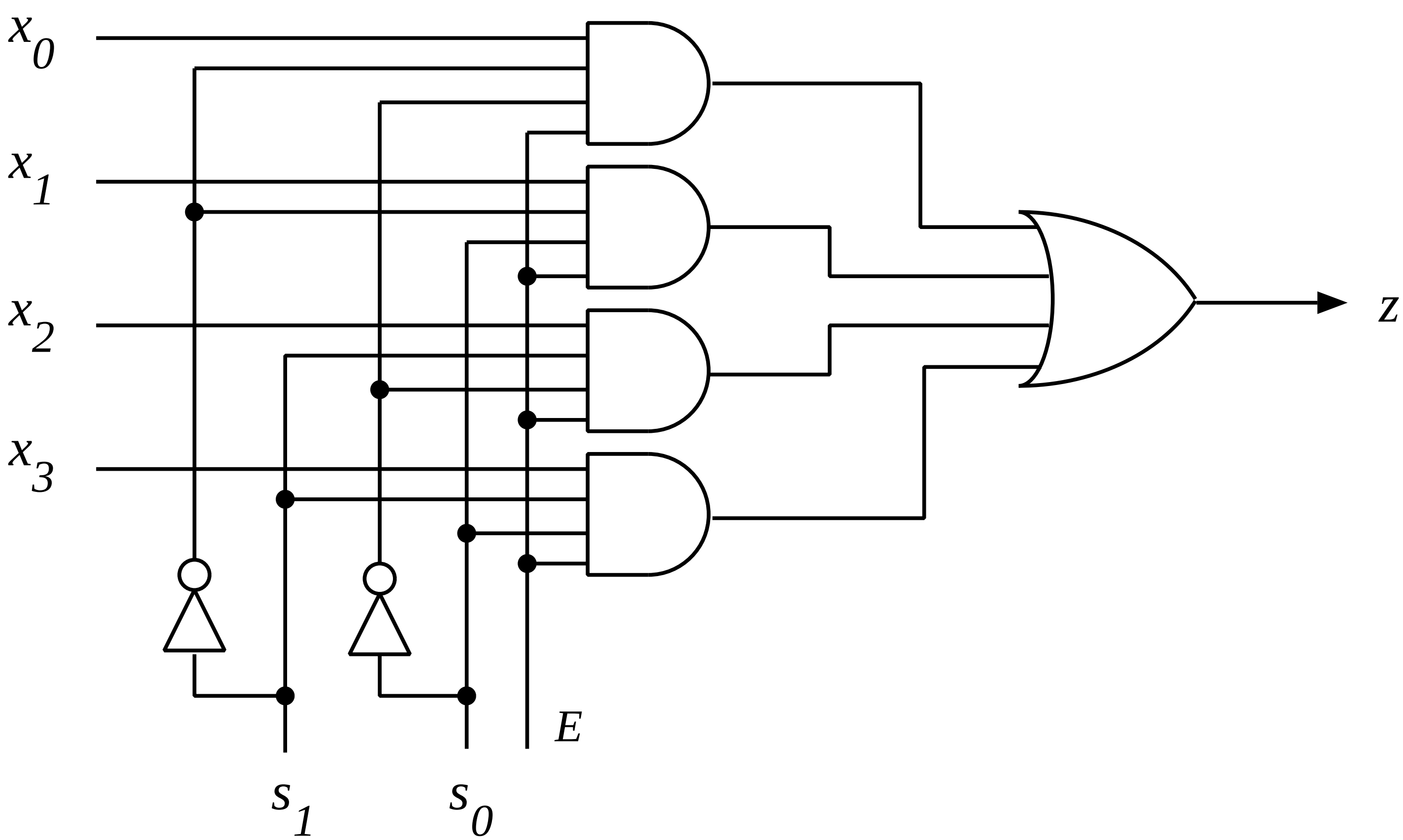


Figure 9.20: GATE IMPLEMENTATION OF 4-INPUT MULTIPLEXER

TYPICAL USES

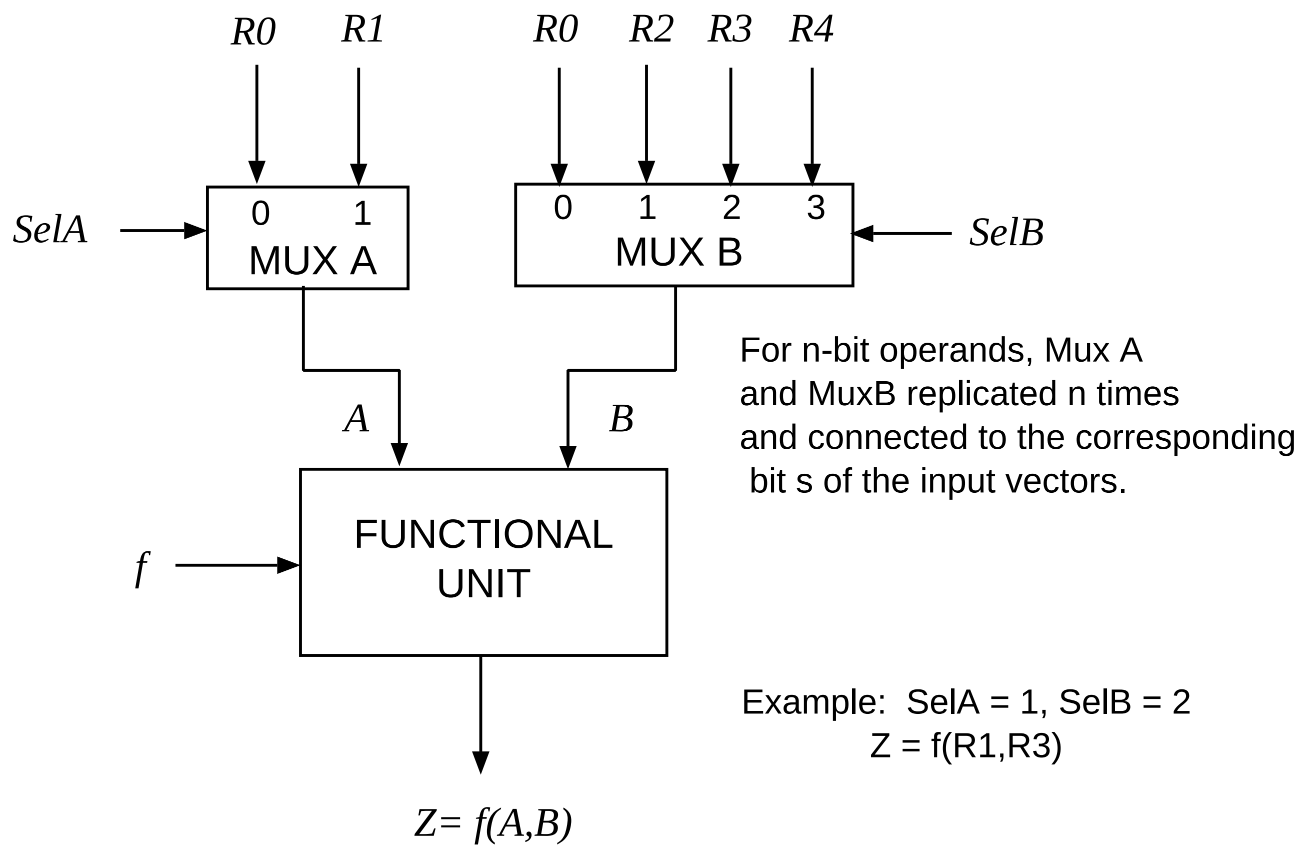


Figure 9.21: MULTIPLEXER: EXAMPLE OF USE.

MULTIPLEXER AS UNIVERSAL COMBINATIONAL MODULE

- connect input variables $\underline{x}$ to select inputs of multiplexer $\underline{s}$
- set data inputs to multiplexer equal to values of function for corresponding assignment of select variables
- using a variable at data inputs reduces size of the multiplexer

EXAMPLE

$$
\begin{aligned}
E(x_2, x_1, x_0) &= \sum m(1,2,4,6,7) \\
&= x_2'(x_1'x_0) + x_2'(x_1x_0') + x_2(x_1'x_0') + x_2(x_1x_0') + x_2(x_1x_0) \\
&= x_2'm_1(x_1,x_0) + x_2'm_2(x_1,x_0) \\
&\quad + x_2m_0(x_1,x_0) + x_2m_2(x_1,x_0) + x_2m_3(x_1,x_0) \\
&= x_2m_0(x_1,x_0) + x_2'm_1(x_1,x_0) + 1 \cdot m_2(x_1,x_0) + x_2m_3(x_1,x_0)
\end{aligned}
$$

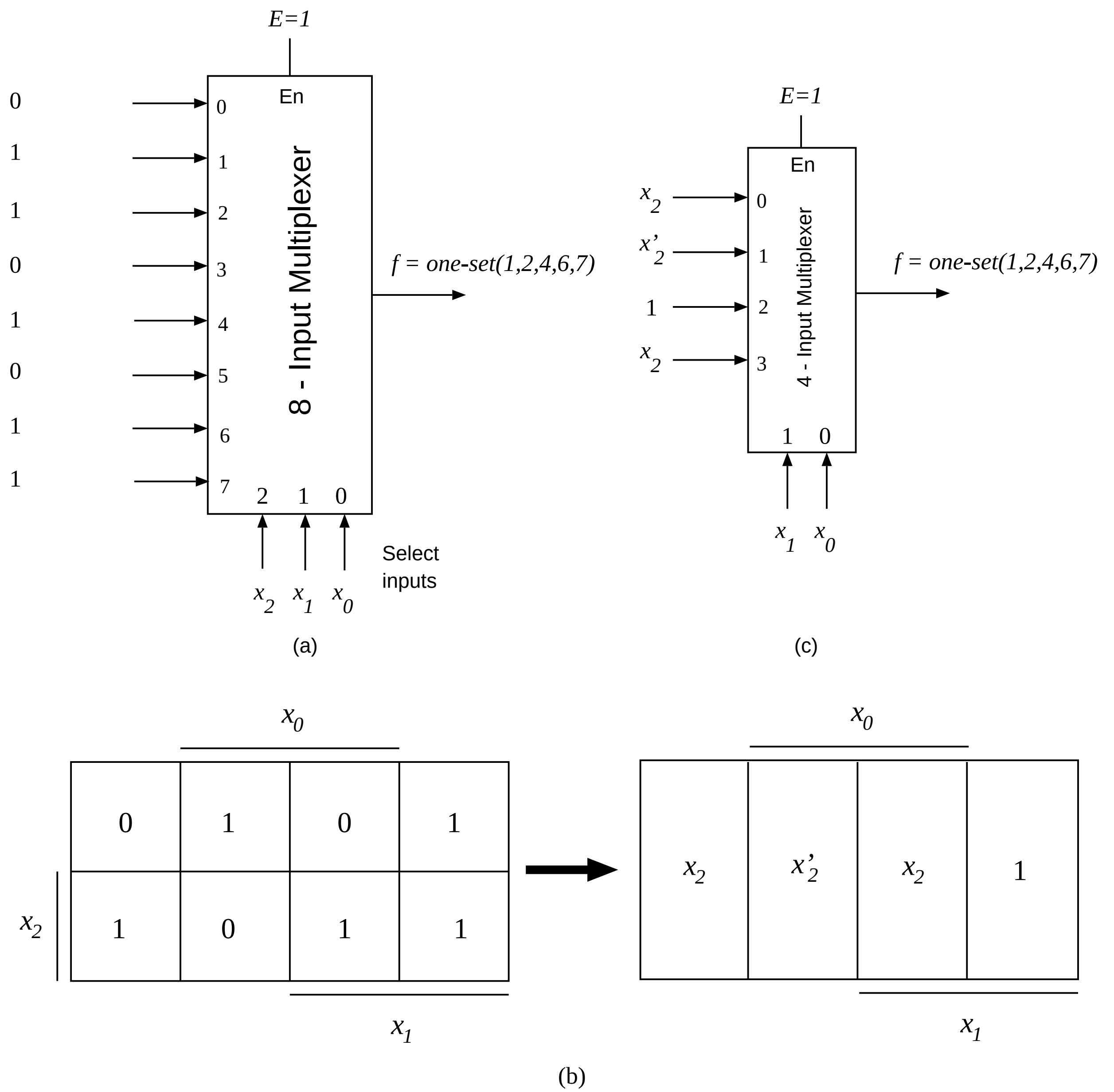


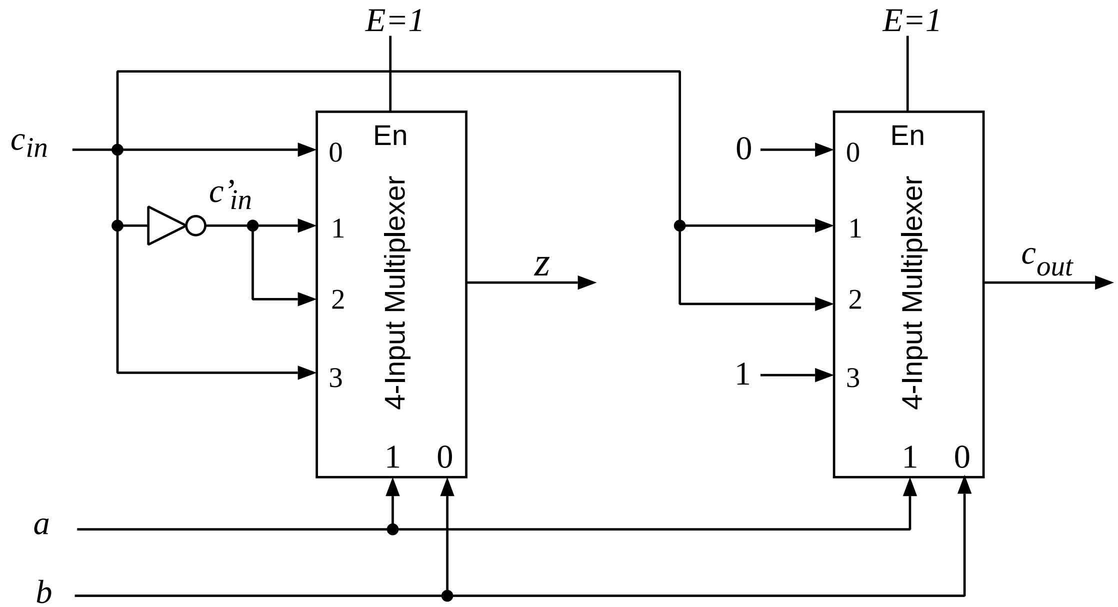
Figure 9.22: IMPLEMENTATION OF $f(x_2, x_1, x_0) = \textit{one-set}$(1,2,4,6,7): a) 8-INPUT MULTIPLEXER; b) K-map; c) 4-INPUT MULTIPLEXER.

EXAMPLE 9.12: ONE-BIT ADDER

Inputs: $a, b, c_{in} \in \{0,1\}$
Outputs: $z, c_{out} \in \{0,1\}$

a	b	c_{in}	z	c_{out}
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

$$\begin{aligned} z &= (a'b') \cdot c_{in} + (a'b) \cdot c'_{in} + (ab') \cdot c'_{in} + (ab) \cdot c_{in} \\ &= c_{in} m_0(a,b) + c'_{in} m_1(a,b) + c'_{in} m_2(a,b) + c_{in} m_3(a,b) \\ c_{out} &= 0 \cdot m_0(a,b) + c_{in} m_1(a,b) + c_{in} m_2(a,b) + 1 \cdot m_3(a,b) \end{aligned}$$



z:

		c_{in}	c_{in}	
	0	1	0	1
a	1	0	1	0
			b	b

c_{out}:

		c_{in}	c_{in}	
	0	0	1	0
a	0	1	1	1
			b	b

		b
	c_{in}	c'_{in}
a	c'_{in}	c_{in}

		b
	0	c_{in}
a	c_{in}	1

Figure 9.23: IMPLEMENTATION OF ONE-BIT ADDER WITH 4-INPUT MULTIPLEXERS.

MULTIPLEXER TREES

$$\underline{s}_{\text{left}} = (s_3, s_2)$$

$$\underline{s}_{\text{right}} = (s_1, s_0)$$

$$w_j = x_{(4j+s_{\text{right}})} \quad , \quad 0 \leq j \leq 3$$

$$z = w_{s_{\text{left}}}$$

$$s = 4s_{\text{left}} + s_{\text{right}}$$

$$z = x_{4s_{\text{left}}+s_{\text{right}}} = x_s$$

16-INPUT TREE MULTIPLEXER

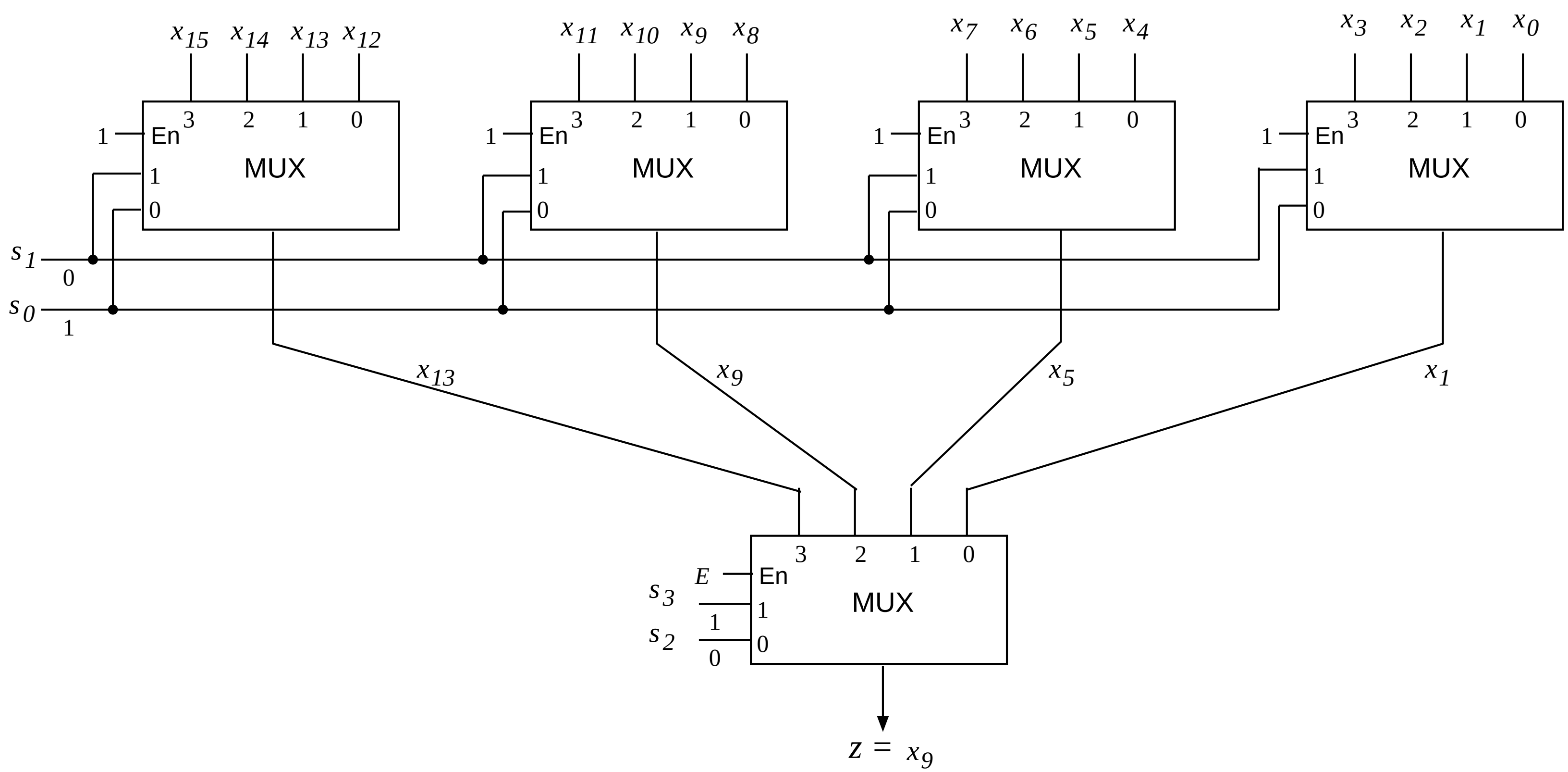


Figure 9.24: TREE IMPLEMENTATION OF A 16-INPUT MULTIPLEXER.

DEMULTIPLEXERS (distributors)

- A HIGH-LEVEL DESCRIPTION:

Inputs: $x, E \in \{0,1\}$

$\underline{s} = (s_{n-1}, \ldots, s_0) \quad , \quad s_j \in \{0,1\}$

Outputs: $\underline{y} = (y_{2^n-1}, \ldots, y_0) \quad , \quad y_i \in \{0,1\}$

Function: $y_i = \begin{cases} x \ \textbf{if} \ (i = s) \ \textbf{and} \ (E = 1) \\ 0 \ \textbf{if} \ (i \neq s) \ \textbf{or} \ (E = 0) \end{cases}$

$$s = \sum_{j=0}^{n-1} s_j 2^j, \ 0 \leq i \leq 2^n - 1$$

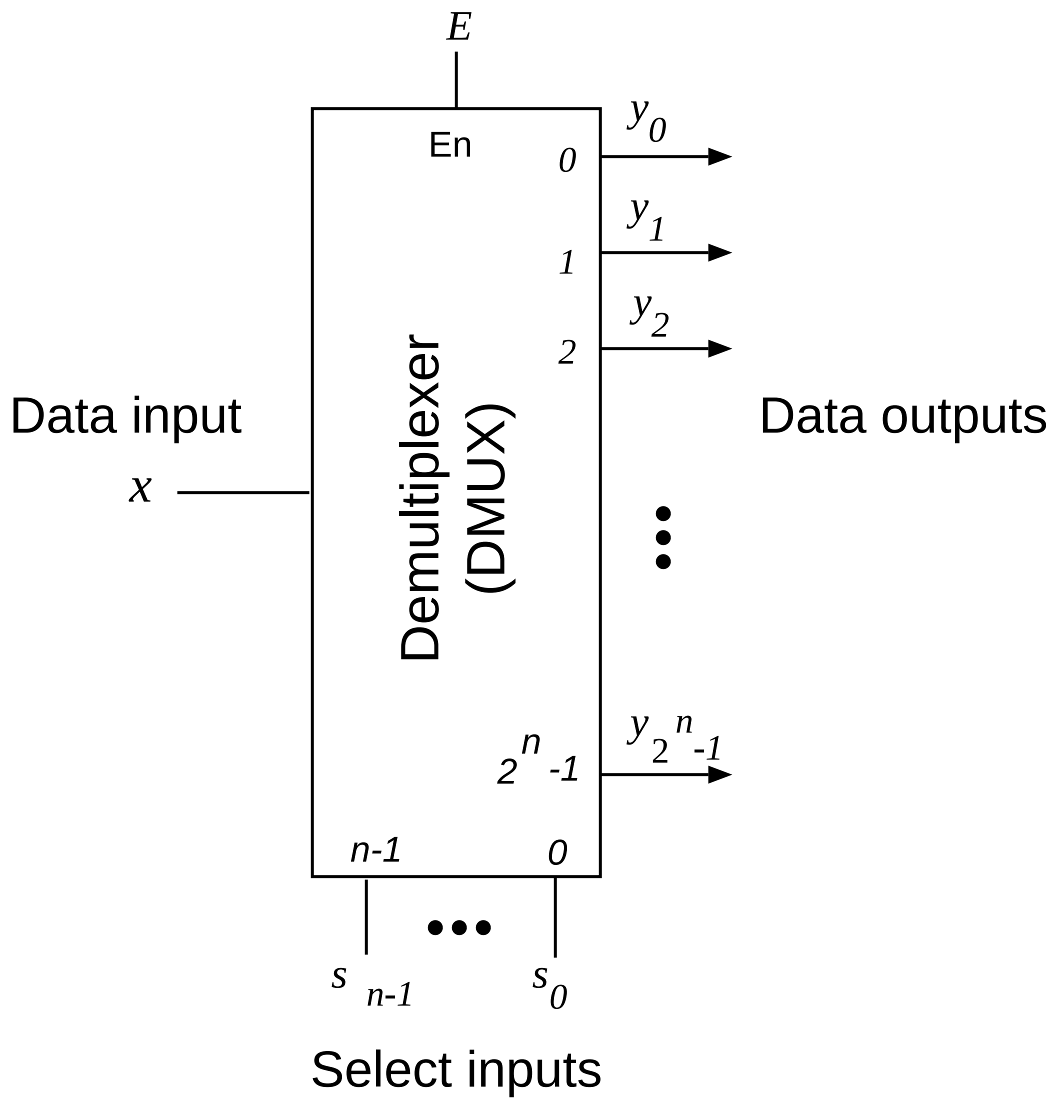


Figure 9.25: 2^n-OUTPUT DEMULTIPLEXER.

EXAMPLE 9.13: 4-OUTPUT DEMULTIPLEXER

E	s_1	s_0	s	y_3	y_2	y_1	y_0
1	0	0	0	0	0	0	x
1	0	1	1	0	0	x	0
1	1	0	2	0	x	0	0
1	1	1	3	x	0	0	0
0	-	-	-	0	0	0	0

$$y_i = E \cdot x \cdot m_i(\underline{s}), \qquad 0 \leq i \leq 2^n - 1$$

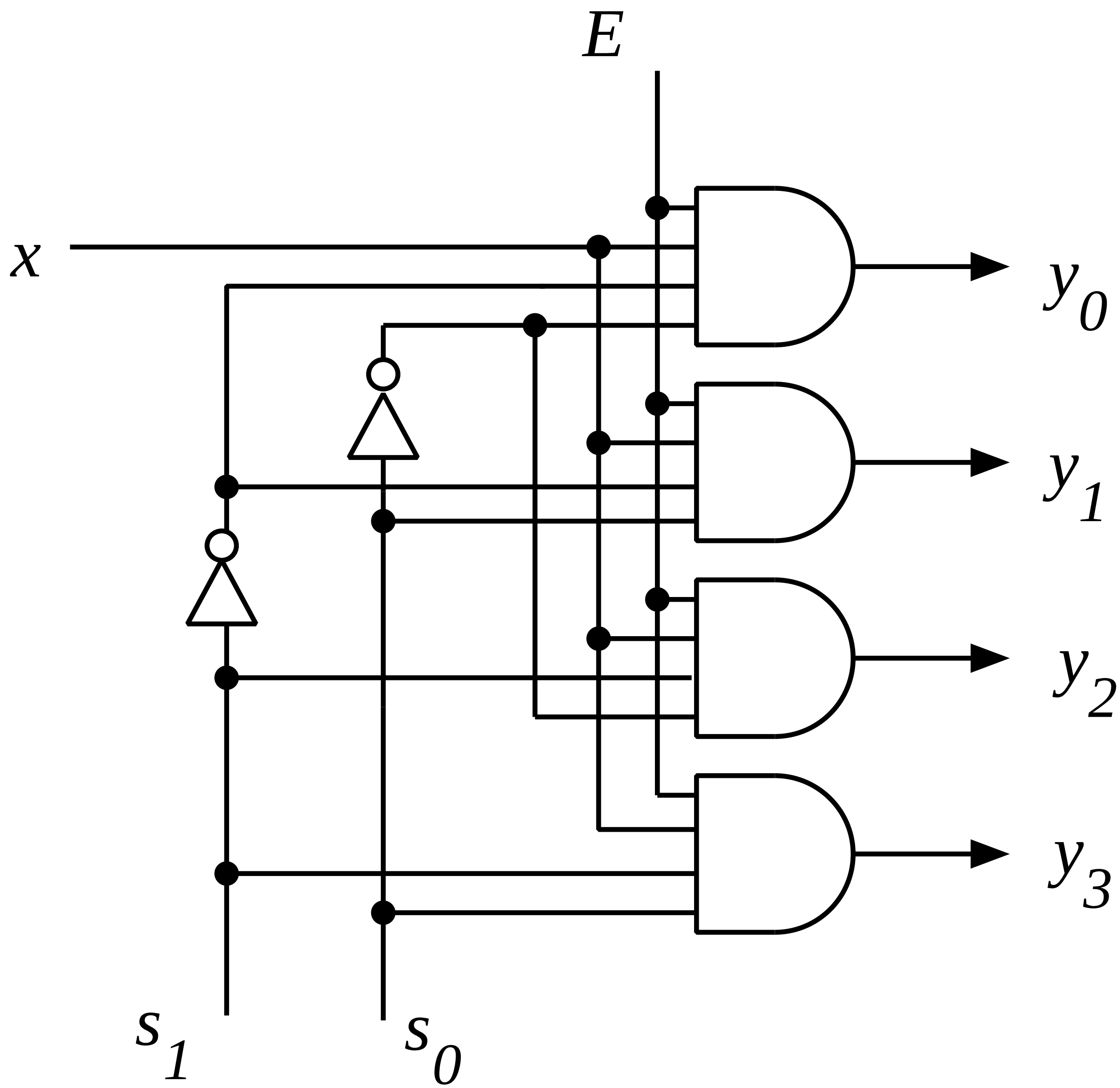


Figure 9.26: GATE NETWORK IMPLEMENTATION OF A 4-OUTPUT DEMULTIPLEXER.

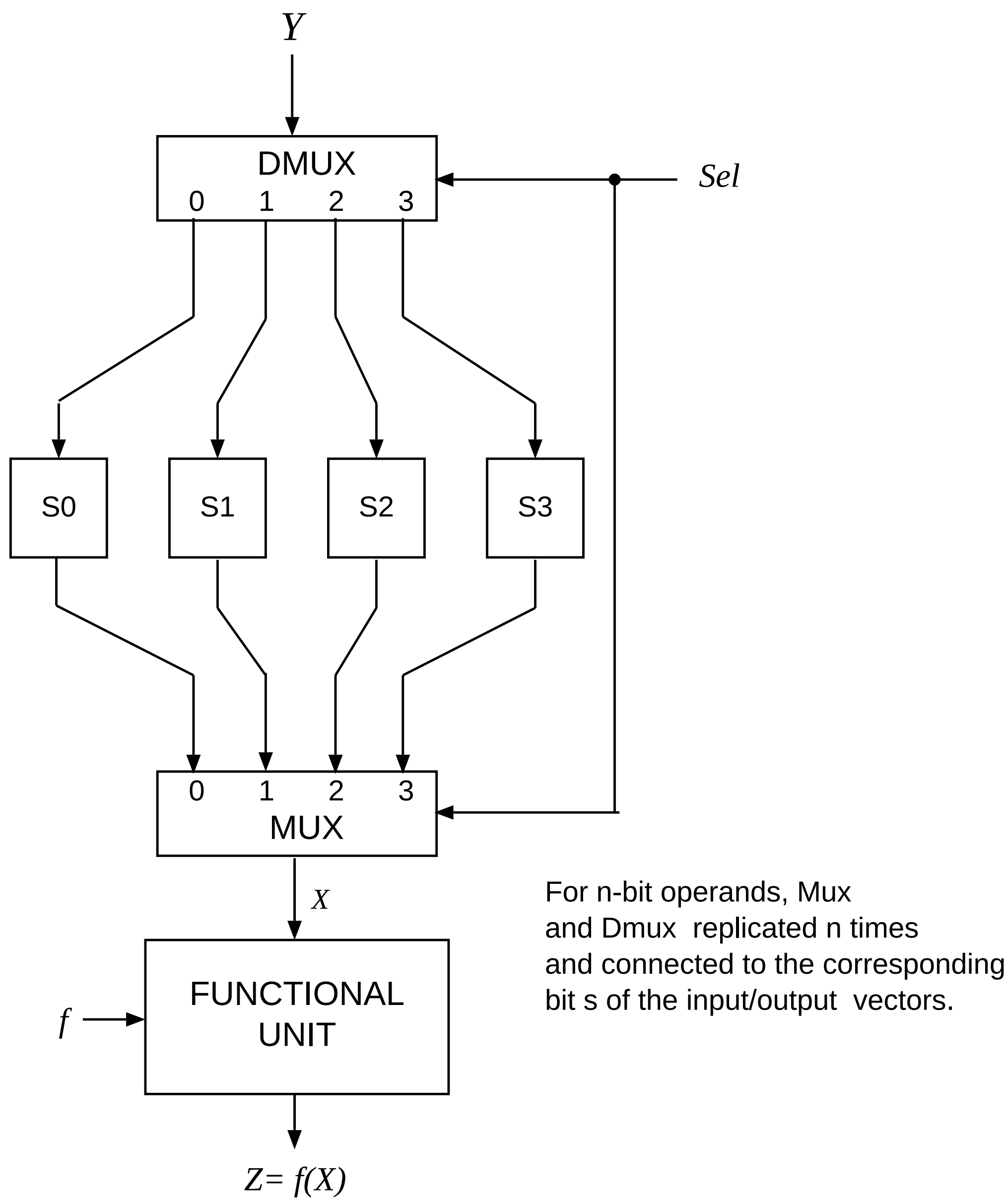


Figure 9.27: DEMULTIPLEXER: example of use.

SIMPLE SHIFTER

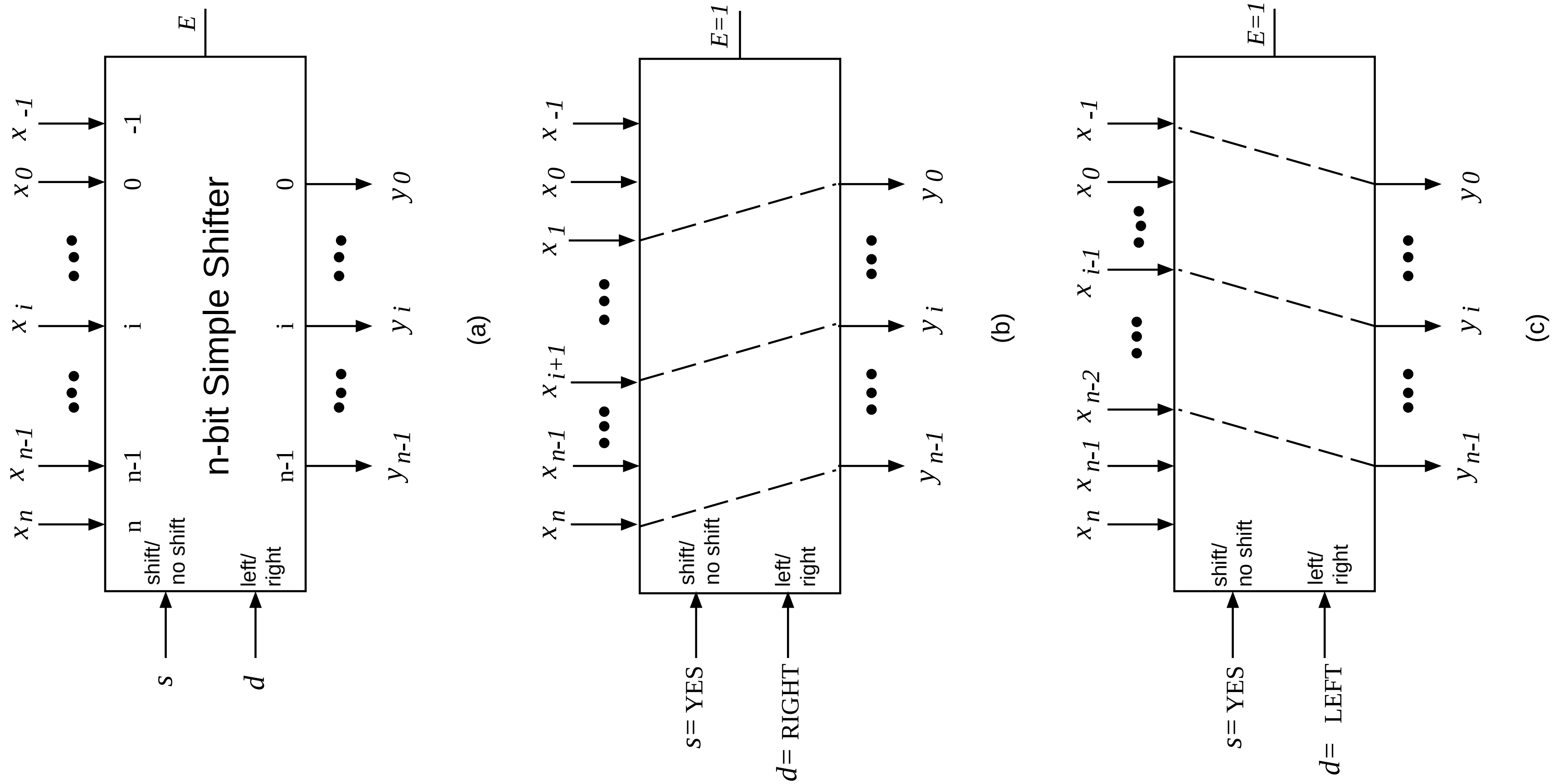


Figure 9.28: n-BIT SIMPLE SHIFTER: a) BLOCK DIAGRAM; b) RIGHT SHIFT; c) LEFT SHIFT.

SIMPLE SHIFTER: HIGH-LEVEL DESCRIPTION

Inputs: $\underline{x} = (x_n, x_{n-1}, \ldots, x_0, x_{-1})$, $x_j \in \{0, 1\}$

$d \in \{RIGHT,\ LEFT\}$

$s \in \{YES,\ NO\}$

$E \in \{0, 1\}$

Outputs: $\underline{y} = (y_{n-1}, \ldots, y_0)$, $y_j \in \{0, 1\}$

Function:

$$y_i = \begin{cases} x_{i-1} & \textbf{if } (d = LEFT) \textbf{ and } (s = YES) \textbf{ and } (E = 1) \\ x_{i+1} & \textbf{if } (d = RIGHT) \textbf{ and } (s = YES) \textbf{ and } (E = 1) \\ x_i & \textbf{if } (s = NO) \textbf{ and } (E = 1) \\ 0 & \textbf{if } (E = 0) \end{cases}$$

for $0 \leq i \leq n - 1$.

$$x_{-1} = \begin{cases} 0 & \text{left shift with 0 insert} \\ 1 & \text{left shift with 1 insert} \\ x_{n-1} & \text{left rotate} \end{cases}$$

$$x_n = \begin{cases} 0 & \text{right shift with 0 insert} \\ 1 & \text{right shift with 1 insert} \\ x_0 & \text{right rotate} \end{cases}$$

EXAMPLE 9.14: 4-INPUT SHIFTER

	Control		Data					
	s	d	x_4	x_3	x_2	x_1	x_0	x_{-1}
			1	0	0	1	1	0
No shift	NO	–		0	0	1	1	
Right shift	YES	RIGHT		1	0	0	1	
Left shift	YES	LEFT		0	1	1	0	
				y_3	y_2	y_1	y_0	

Coding:

s	
0	NO
1	YES

d	
0	RIGHT
1	LEFT

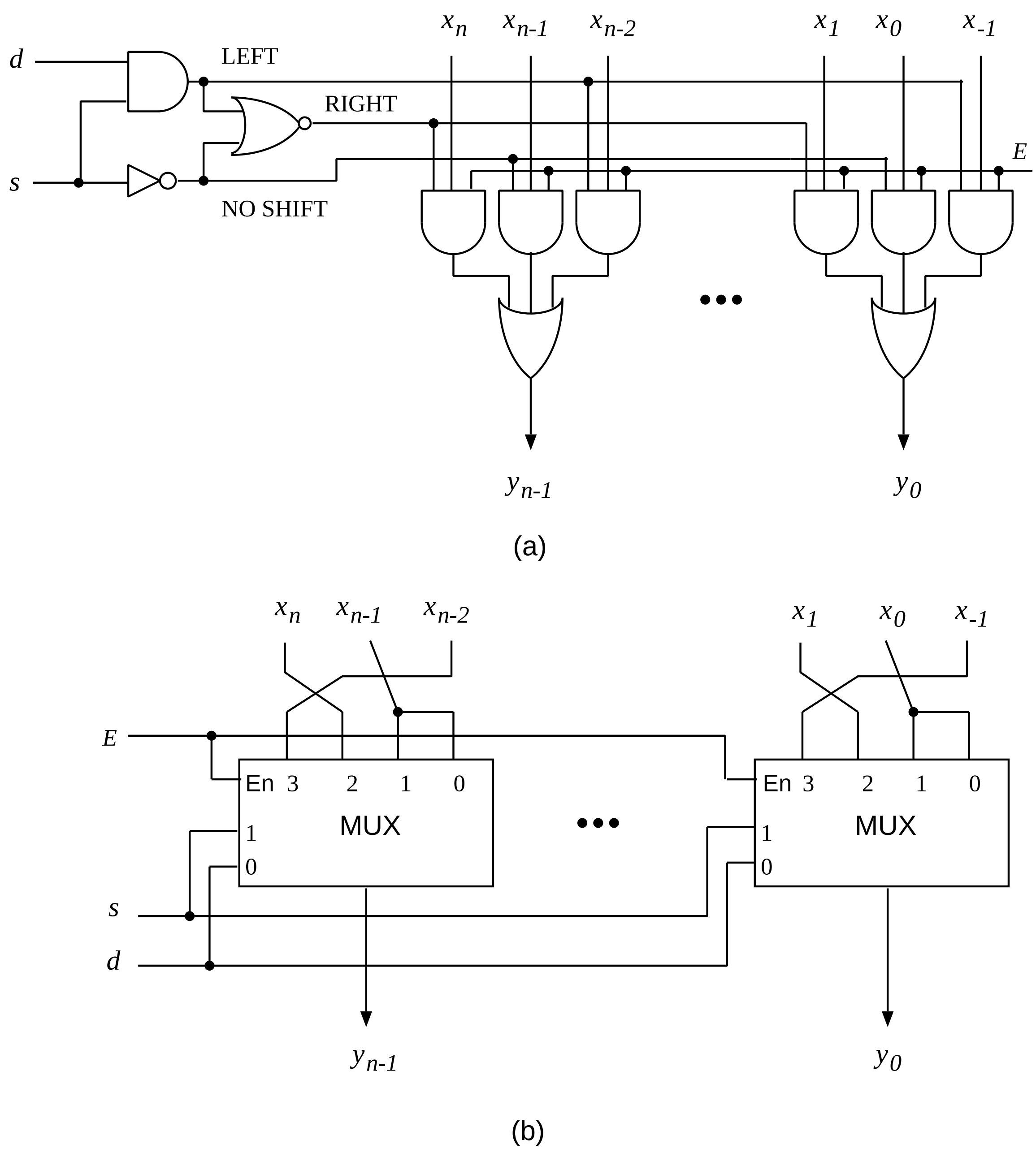


Figure 9.29: IMPLEMENTATION OF A SIMPLE SHIFTER: a) WITH GATES; b) WITH MULTIPLEXERS.

p-SHIFTER: HIGH-LEVEL DESCRIPTION

Inputs: $\underline{x} = (x_{n+p-1}, \ldots, x_n, x_{n-1}, \ldots, x_0, x_{-1}, \ldots, x_{-p})\ , \quad x_j \in \{0, 1\}$
$s \in \{0, 1, ..., p\}$
$d \in \{LEFT,\ RIGHT\}$
$E \in \{0, 1\}$

Outputs: $\underline{y} = (y_{n-1}, \ldots, y_0)\ \ , \quad y_j \in \{0, 1\}$

Function:

$$y_i = \begin{cases} x_{i-s} \ \textbf{if} \ \ (d = LEFT) \ \textbf{and} \ (E = 1) \\ x_{i+s} \ \textbf{if} \ \ (d = RIGHT) \ \textbf{and} \ (E = 1) \\ 0 \quad\ \ \textbf{if} \ \ (E = 0) \end{cases}$$

$$0 \leq i \leq n - 1$$

p-SHIFTER

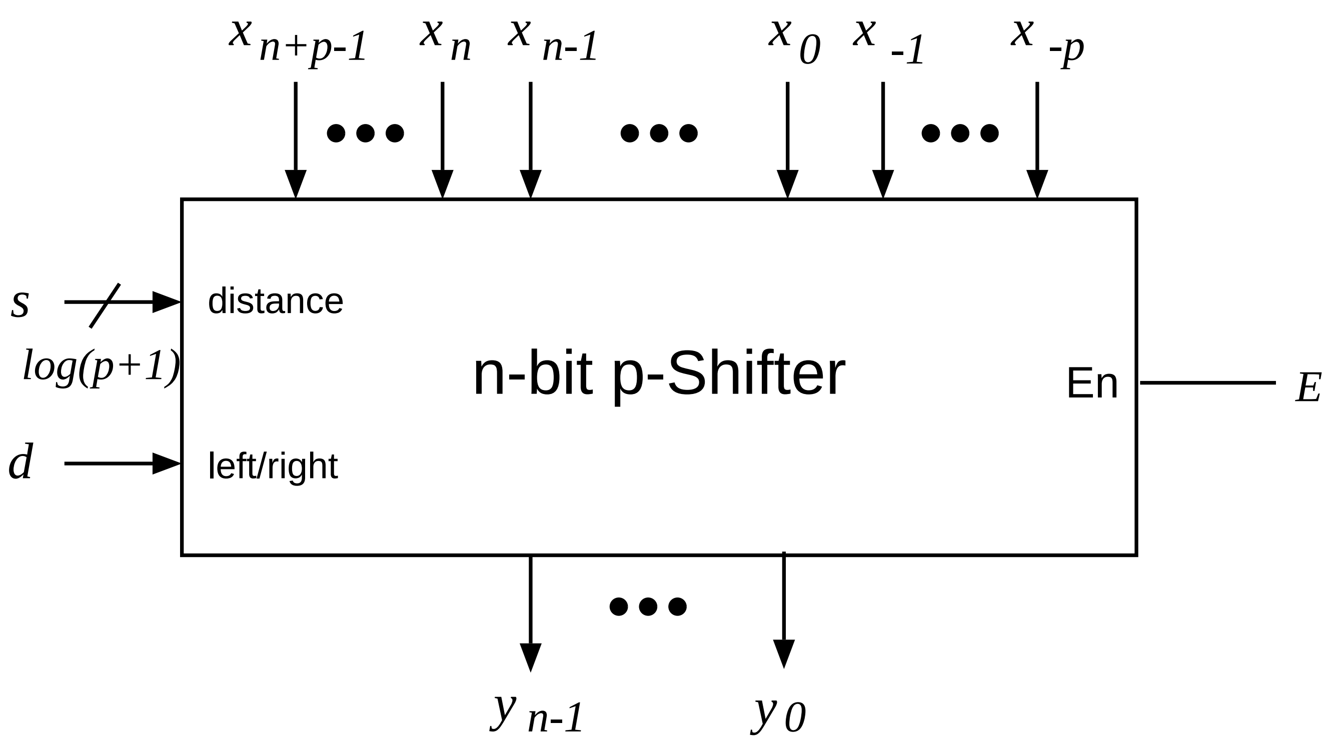


Figure 9.30: n-BIT p-SHIFTER.

BARREL SHIFTER

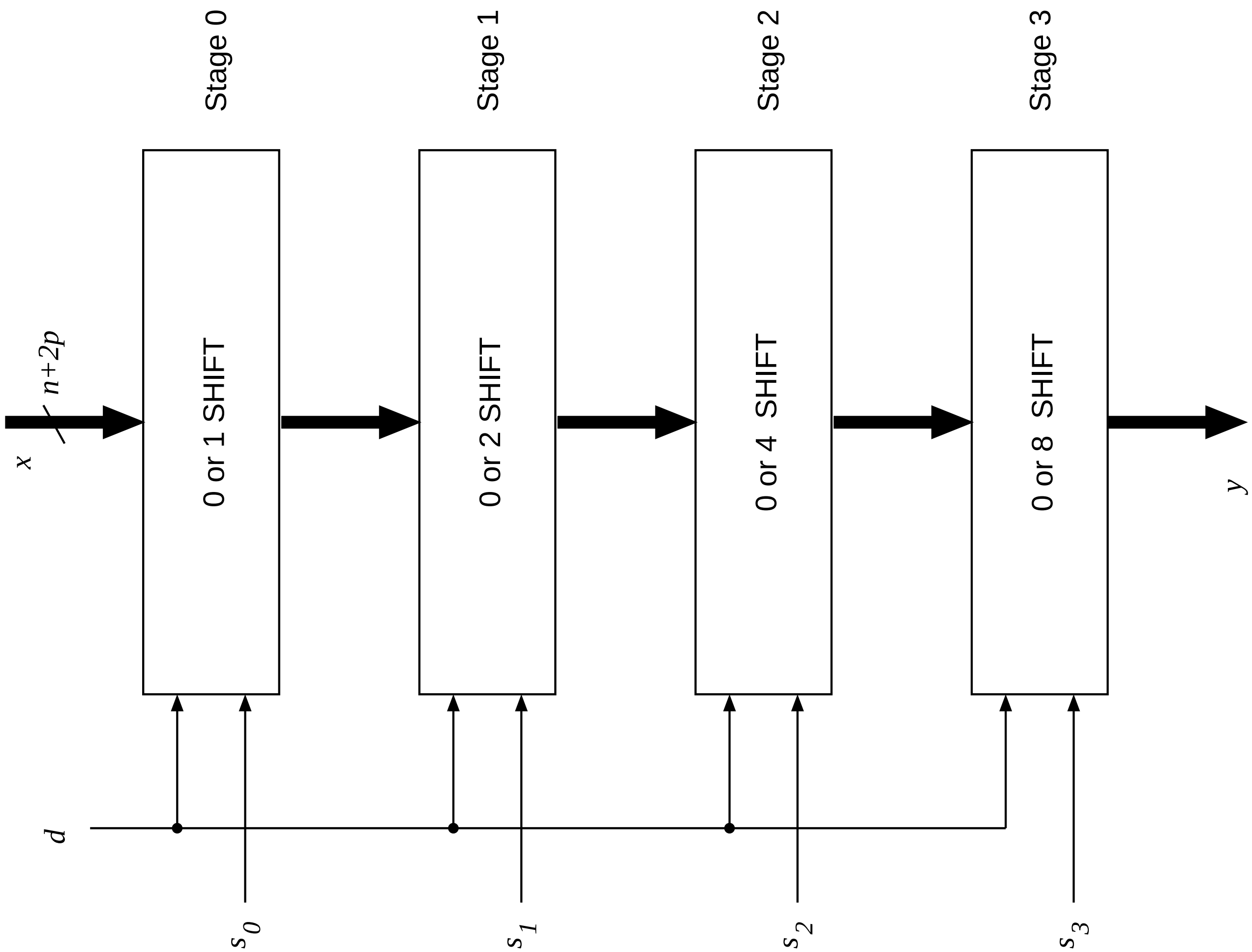


Figure 9.31: BARREL SHIFTER FOR $p = 15$.

UNIDIRECTIONAL SHIFTERS

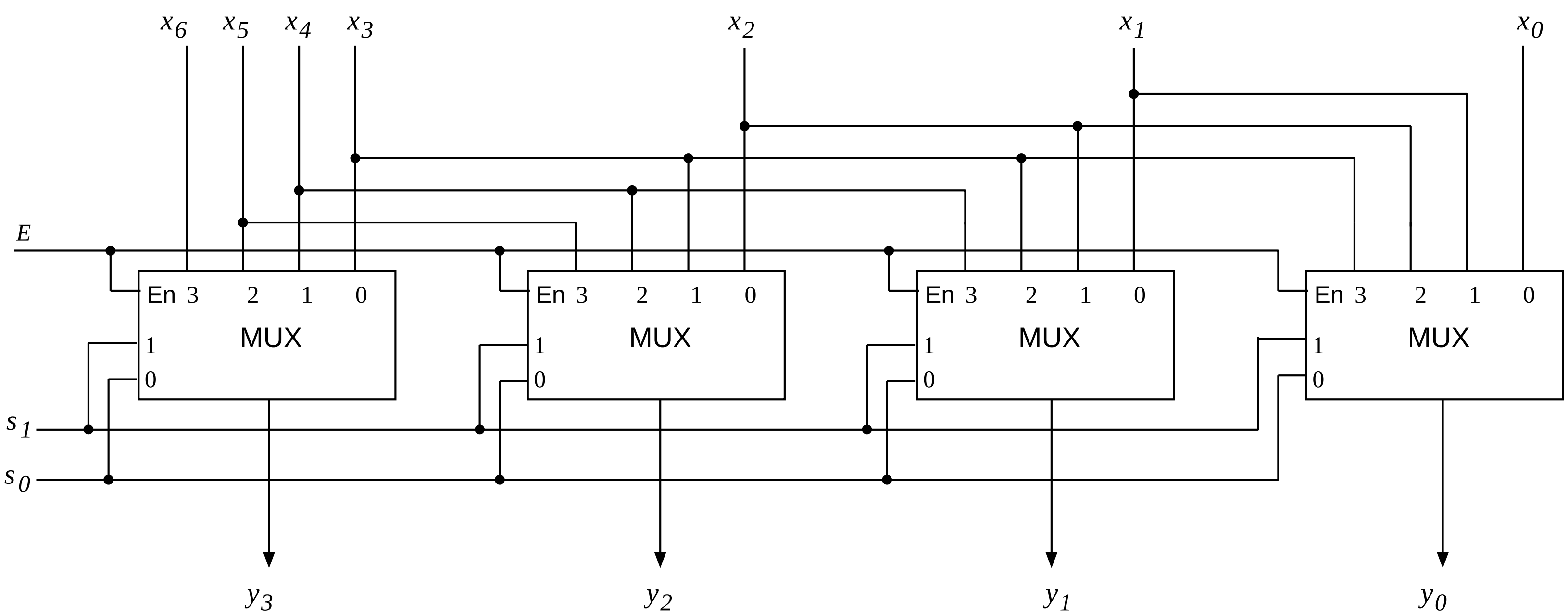


Figure 9.32: MULTIPLEXER IMPLEMENTATION OF A 4-BIT RIGHT 3-SHIFTER.

TYPICAL USES OF SHIFTERS

- ALIGNMENT OF A BIT-VECTOR
- REMOVAL OF THE LEADING (or trailing) BITS OF A VECTOR
- PERFORMING MULTIPLICATION OR DIVISION BY A POWER OF TWO
- EXTRACTING A SUBVECTOR from a bit-vector, using a shifter instead of a selector