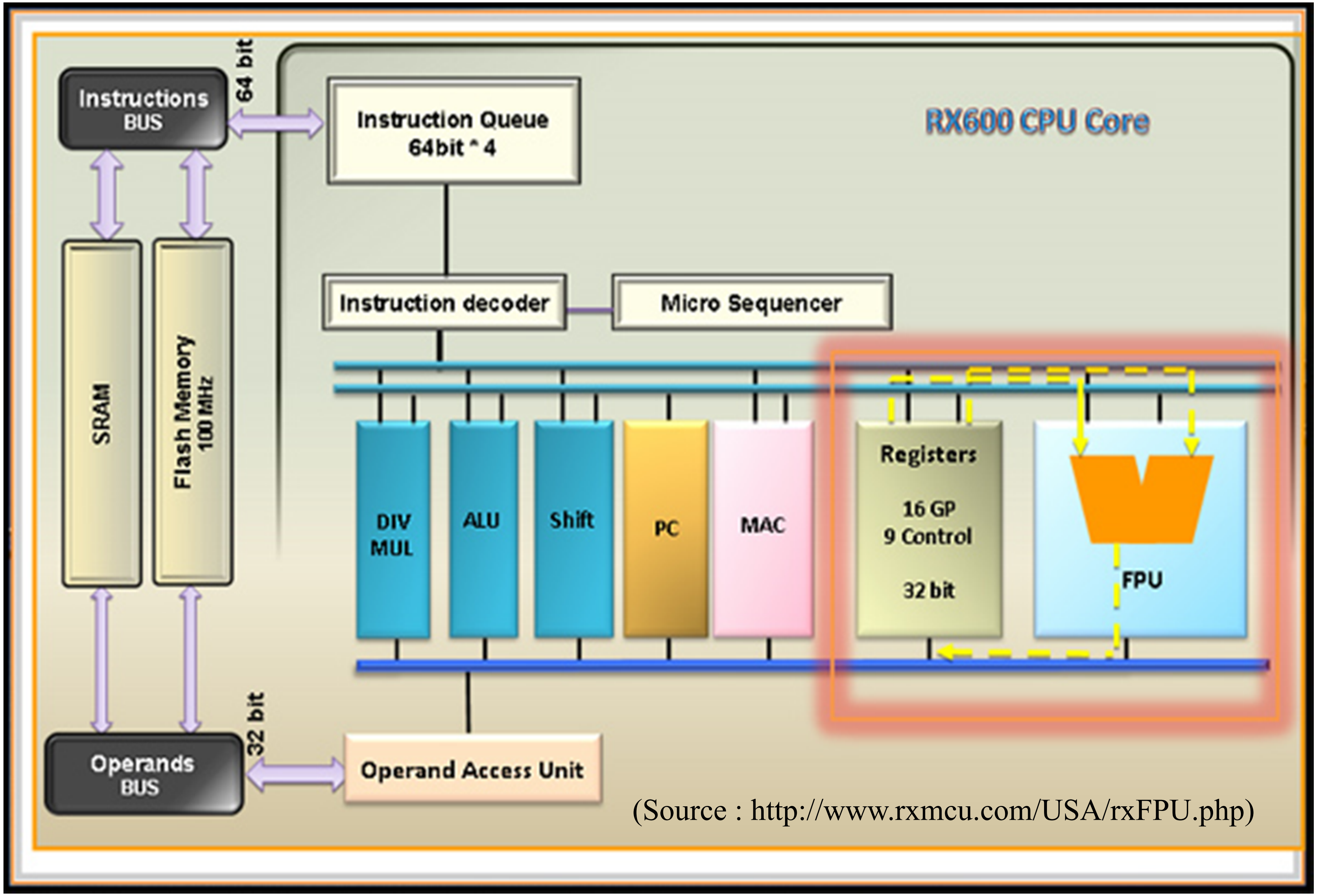
Computer Arithmetic (I)

- Number Systems
- 2s Complement
- Integer Multiplication
- Serial Multiplier



(Source : http://www.rxmcu.com/USA/rxFPU.php)

Positional Number Systems

The base, or radix of a number system defines the range of possible values that a digit may have: 0 – 9 for decimal; 0 – 1 for binary; 0 – F for hexadecimal.

257_{10}	radix-10 or decimal code
11011_{2}	radix-2 or binary code
$3C5B_{16}$	radix-16 or hexadecimal code

A symbol represents the quantity and its position represents the weighting.

$$541.25_{10} = (5 \times 10^{2)} + (4 \times 10^{1}) + (1 \times 10^{0}) + (2 \times 10^{-1}) + (5 \times 10^{-2})$$

$$= (541.25)_{10}$$

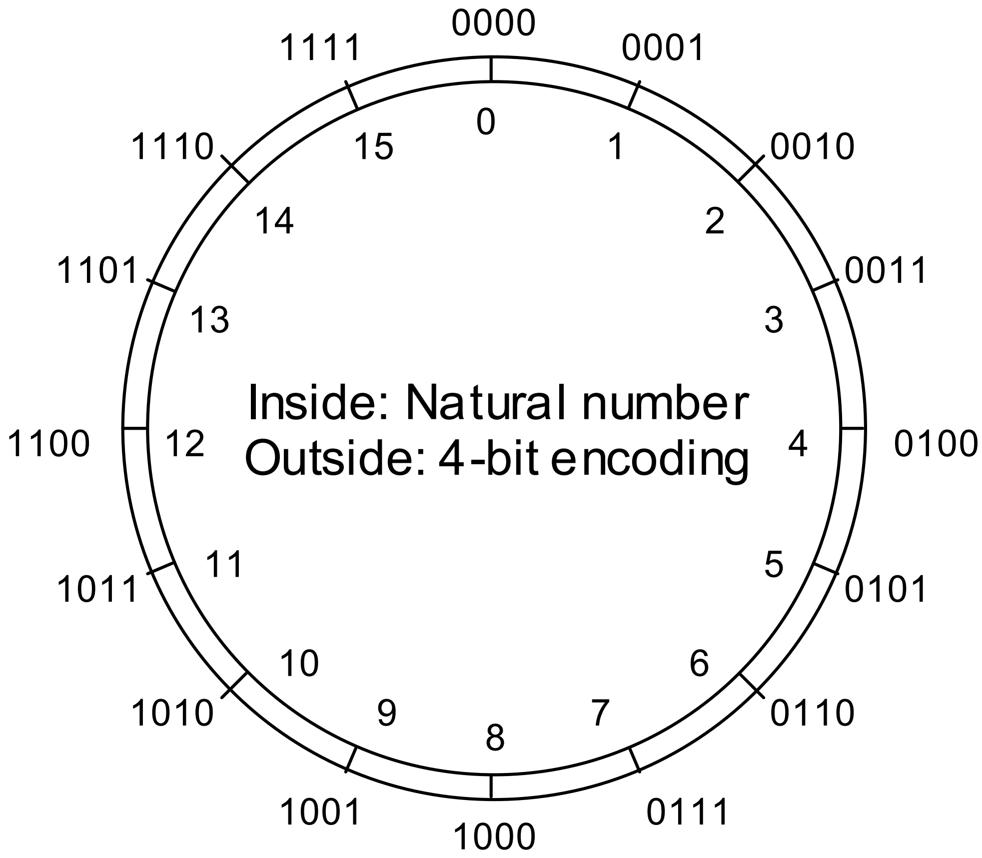
$$1110.11_{2} = (1\times2^{3}) + (1\times2^{2}) + (1\times2^{1}) + (0\times2^{0}) + (1\times2^{-1}) + (1\times2^{-2})$$

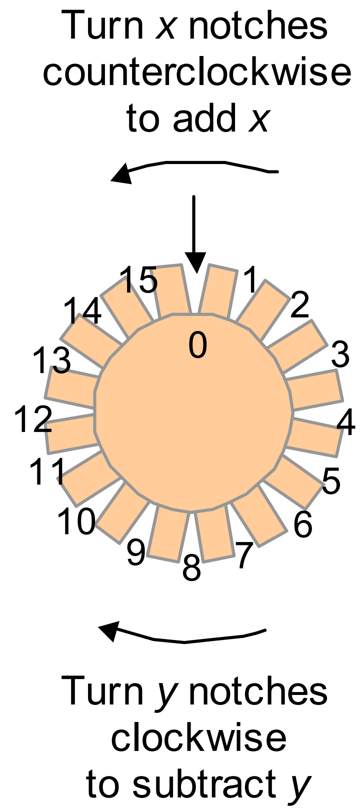
$$= 8 + 4 + 2 + 0 + 0.5 + 0.25 = 14.75$$

$$4B_{16} = (4\times16^{1}) + (11\times16^{0})$$

$$= 64 + 11 = 75$$

Schematic representation of 4-bit code for unsigned integers in 0 - 15.

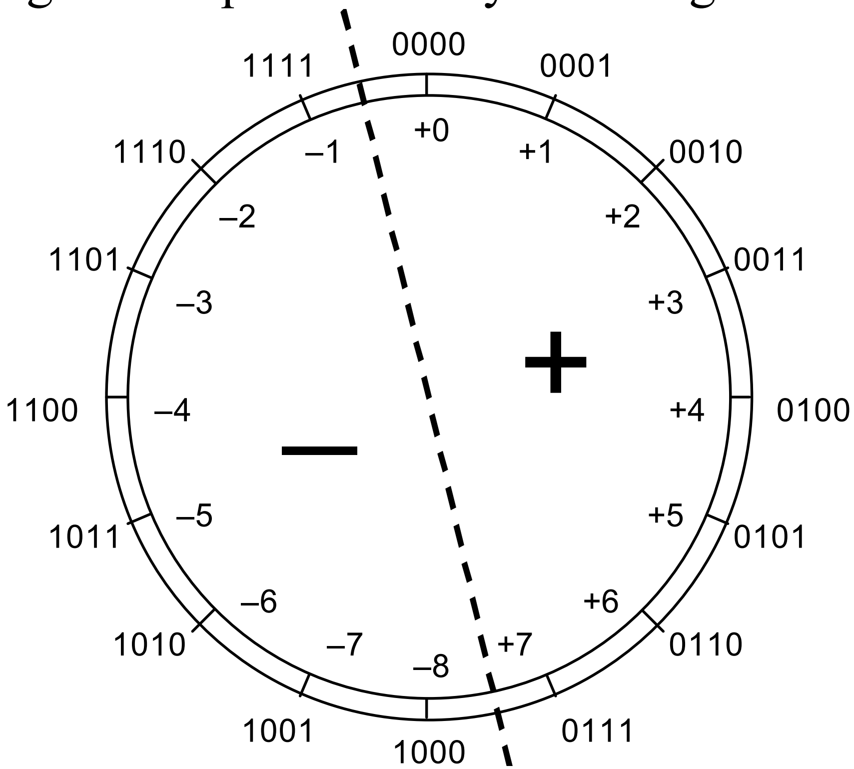


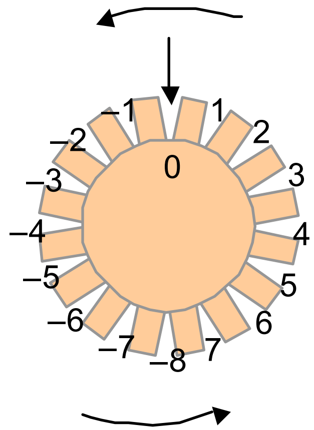


Two's-Complement Representation

With k bits, numbers in the range $[-2^{k-1}, 2^{k-1} - 1]$ represented.

Negation is performed by inverting all bits and adding 1.





Schematic representation of 4-bit 2's-complement code for integers in [–8, +7].

Two's-Complement Representation

With k bits, numbers in the range $[-2^{k-1}, 2^{k-1} - 1]$ represented.

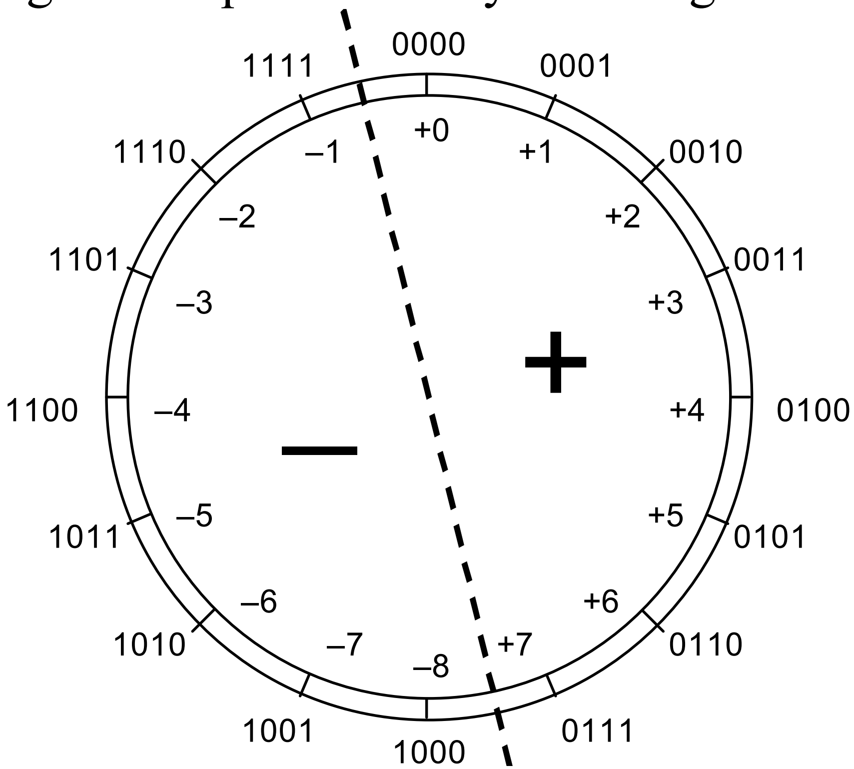
Negation is performed by inverting all bits and adding 1.

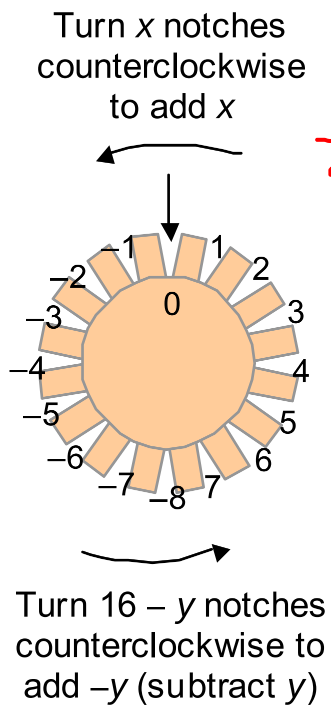
$2 - 4 = -2$

$2 + (-4) = -2$

$2 + (16 - 4)$

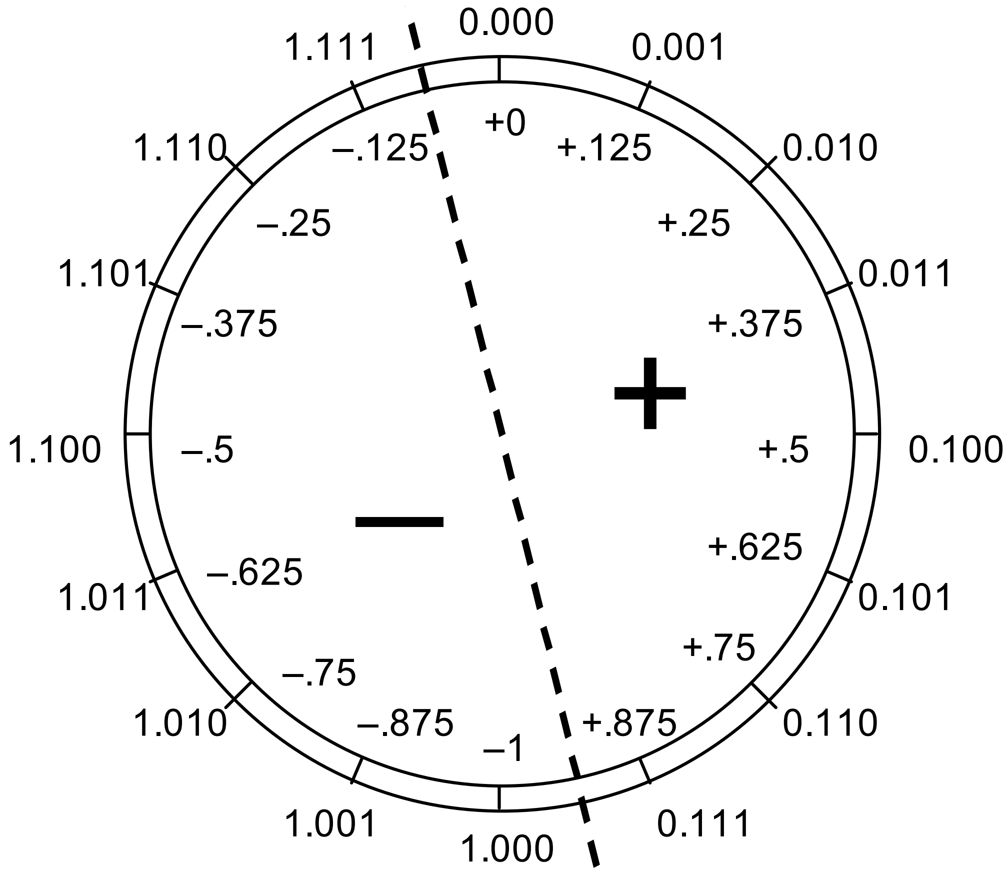
$2 + 12 = -2$





Schematic representation of 4-bit 2's-complement code for integers in [–8, +7].

Fixed-Point 2's-Complement Numbers



Schematic representation of 4-bit 2's-complement encoding for (1 + 3)-bit fixed-point numbers in the range [–1, +7/8].

What is multiplication ?

$$\begin{array}{r} 13 \\ \times\ 11 \\ \hline 143 \end{array}$$

Solution 1:
repeated addition

$$\begin{array}{r} 13 \\ 13 \\ 13 \\ 13 \\ 13 \\ 13 \\ 13 \\ 13 \\ 13 \\ 13 \\ +\ 13 \\ \hline 143 \end{array}$$

Solution 2:
form partial products

$$11 = 10 + 1$$

$$\begin{array}{rr} 1 \times 13 = & 13 \\ 10 \times 13 = & 130\ + \\ \hline & 143 \end{array}$$

Decimal example:

$$
\begin{array}{lclcr}
 & & & & 456 \\
 & & & & \times\ 123 \\
\hline
3 \times 456 \times 10^0 & = & 1368 \times 10^0 & = & 1368 \\
2 \times 456 \times 10^1 & = & 912 \times 10^{\color{red}1} & = & +\ 912{\color{red}0} \\
1 \times 456 \times 10^2 & = & 456 \times 10^{\color{red}2} & = & +\ 456{\color{red}00} \\
\hline
 & & & & 56088
\end{array}
$$

Binary example:

$$
\begin{array}{rrcrcr}
 & & & 1101 & \longleftarrow & 13 \\
 & & & \times \quad \underline{1011} & \longleftarrow & 11 \\
1 \times 1 \times 13 = & 13 & \longrightarrow & 1101 & & \\
1 \times 2 \times 13 = & 26 & \longrightarrow & 11010 & & \\
0 \times 4 \times 13 = & 0 & \longrightarrow & 000000 & & \\
1 \times 8 \times 13 = & \underline{104} & \longrightarrow & \underline{1101000} & & \\
 & 143 & & 10001111 & \longleftarrow & 143
\end{array}
$$

Base 2 is similar to base 10 except you only multiply by 1 or 0 - hence easier

Multiplicand *Multiplier*

7 x 11 (in 5 bits) = 00111 x 01011

```
                   00111
        x          01011
        -----------------
 7  -->            00111  1
14  -->           001110  1
                 0000000  0
56  -->         00111000  1
               000000000  0
        -----------------
              0001001101
```

Partial Products

Product = 56+14+7 = 77

Shifts due to significance of multiplier digit

We can define a function $bit_i()$ which returns the value of the i[th] bit (LSB is i=0) of an N bit number.

In this case we can write the product as:

$$pr = \sum_{i=0}^{N-1} bit_i(mr) \cdot md \cdot 2^i$$

mr = multiplier, md = multiplicand, pr = product

Note: multiplying an M and N bit number yields an M+N bit product

Fractional Multiplication

The size of a number depends on the positioning of the binary point (base 2 equivalent of the decimal point) in the same way as with base 10 numbers.

e.g. abcd.efgh is equivalent to:

$$a \times 2^{3} + b \times 2^{2} + c \times 2^{1} + d \times 2^{0} +$$
$$e \times 2^{-1} + f \times 2^{-2} + g \times 2^{-3} + h \times 2^{-4}$$

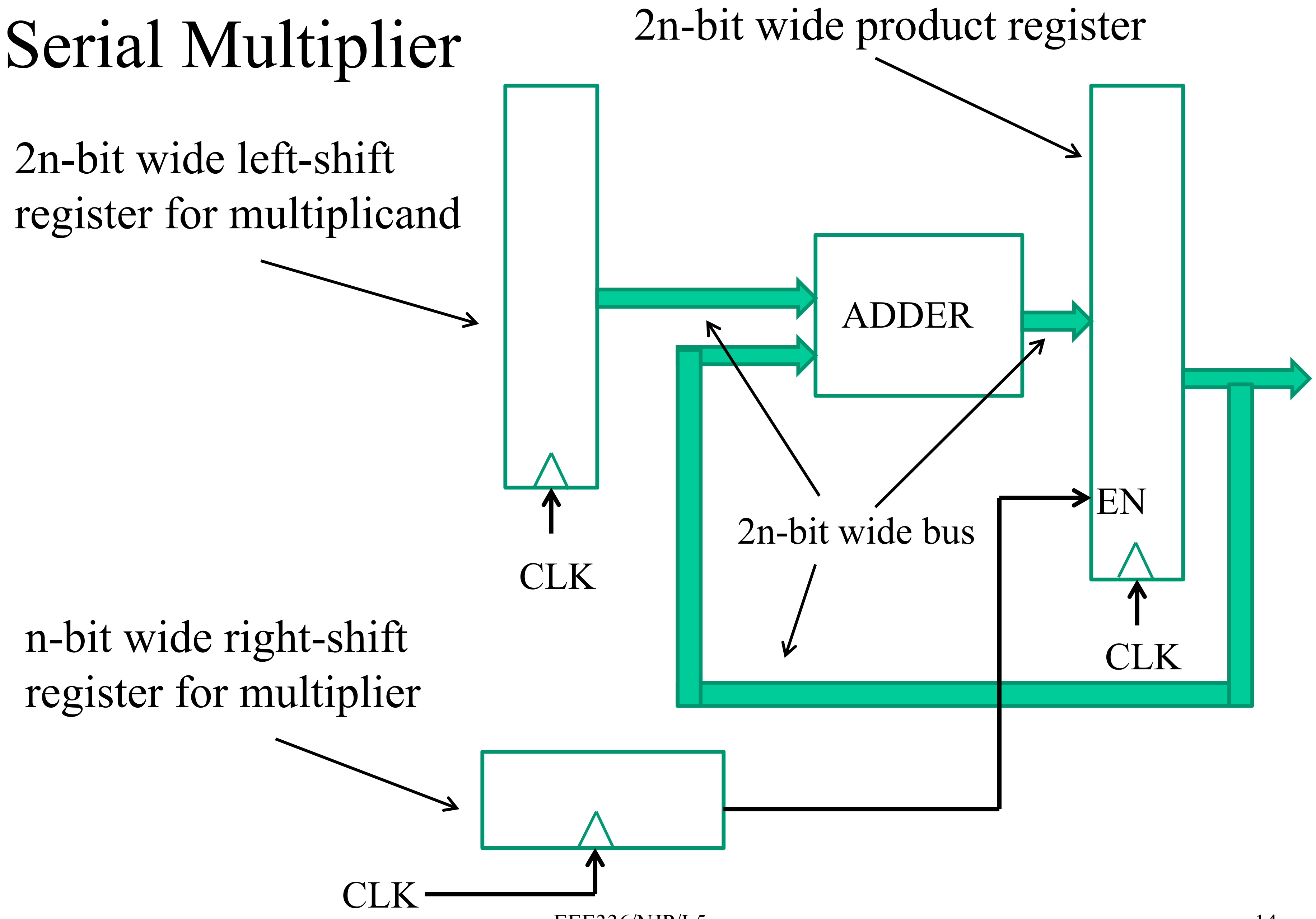
To multiply fractions:

- ignore the binary point
- multiply the two numbers
- add up the number of fractional digits and position the binary point this number of digits from the least significant end of the product

e.g. $3.5 \times 2.75 = 9.625$

$0011.1 \times 010.11 = 01001.101 = 9.625\ (77 / 8)$

Serial Multiplier



Operation

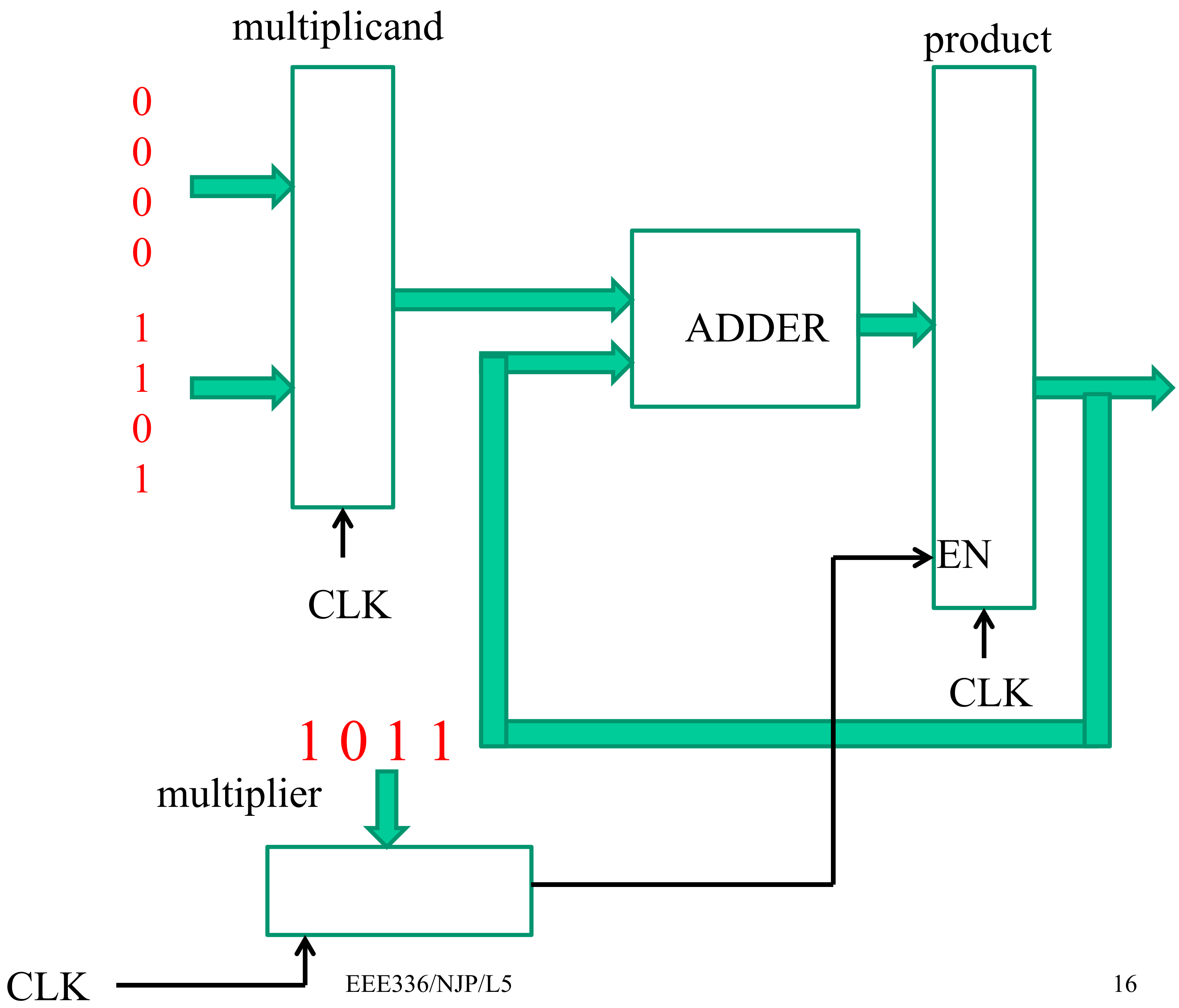
At the start of the process, we load the n-bit multiplicand into the bottom end of a 2n-bit left-shift shift register and load 0's into the top n bits. When a shift takes place, a 0 will be input to the LSB position.

Load the multiplier into a n-bit right shift register.

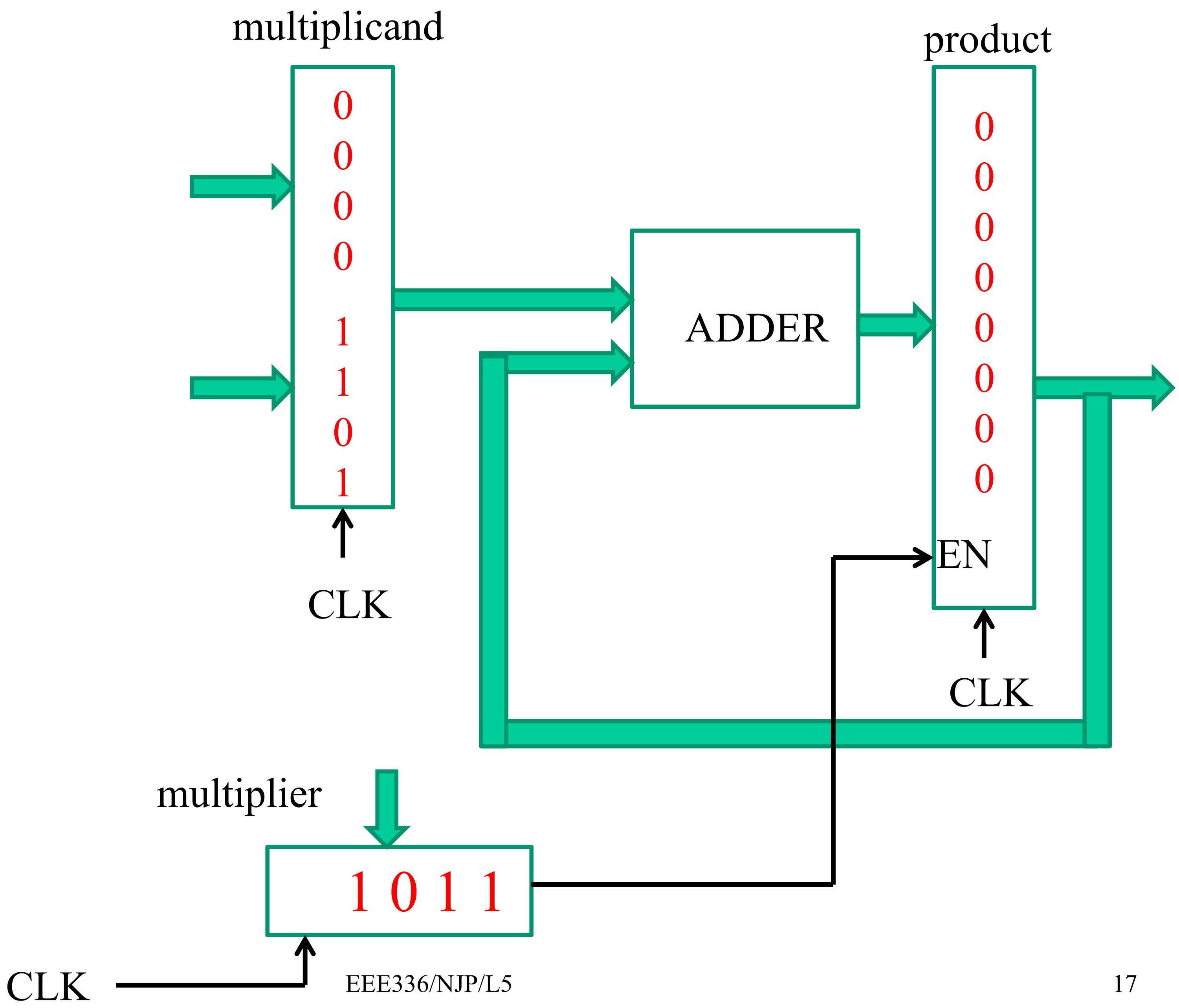
Load 0's into the 2n-bit wide product register.

Takes n clock cycles for an n-bit multiplier.

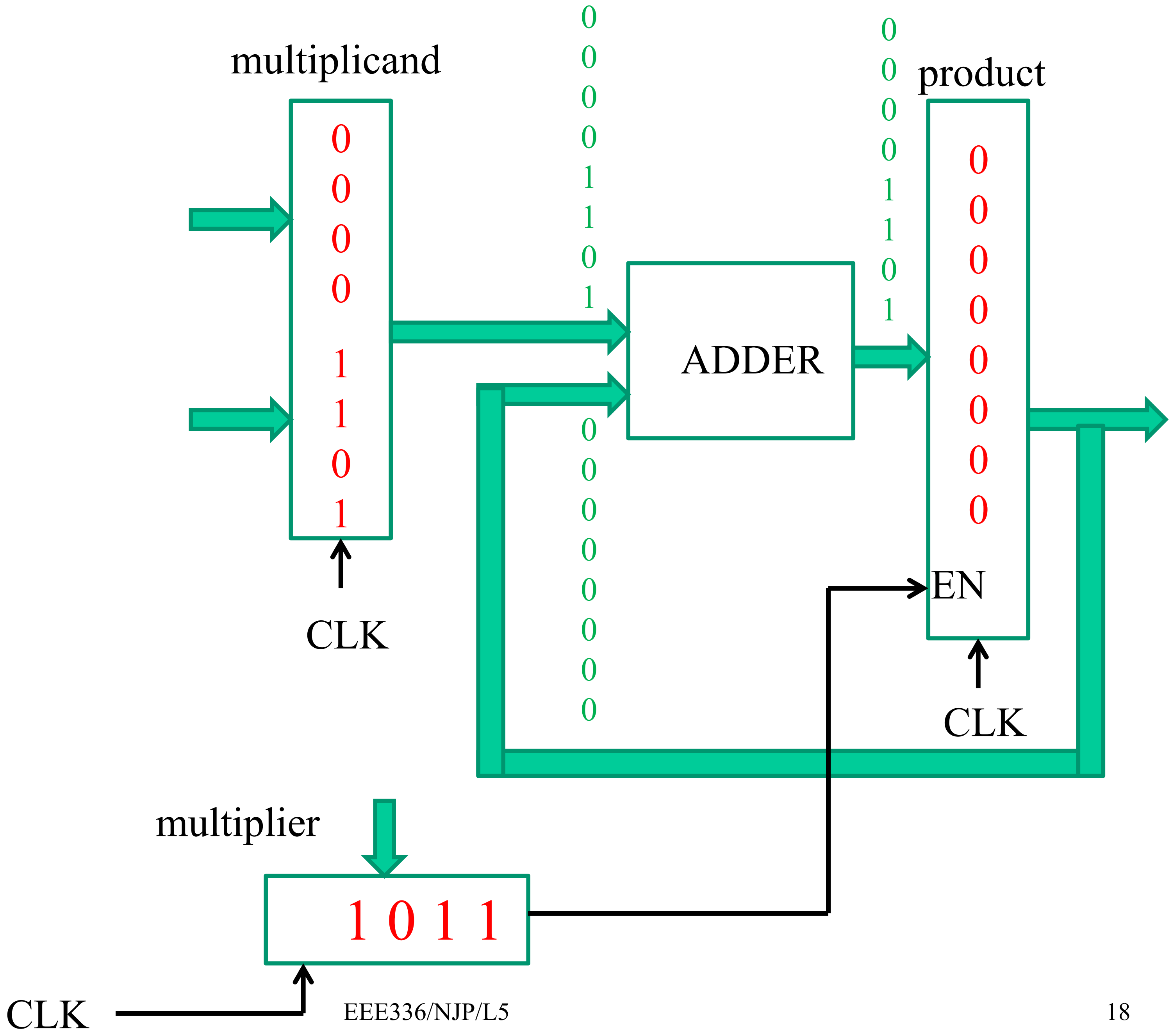
```
     1101
×    1011
---------
     1101
    11010
   000000
  1101000
---------
 10001111
```



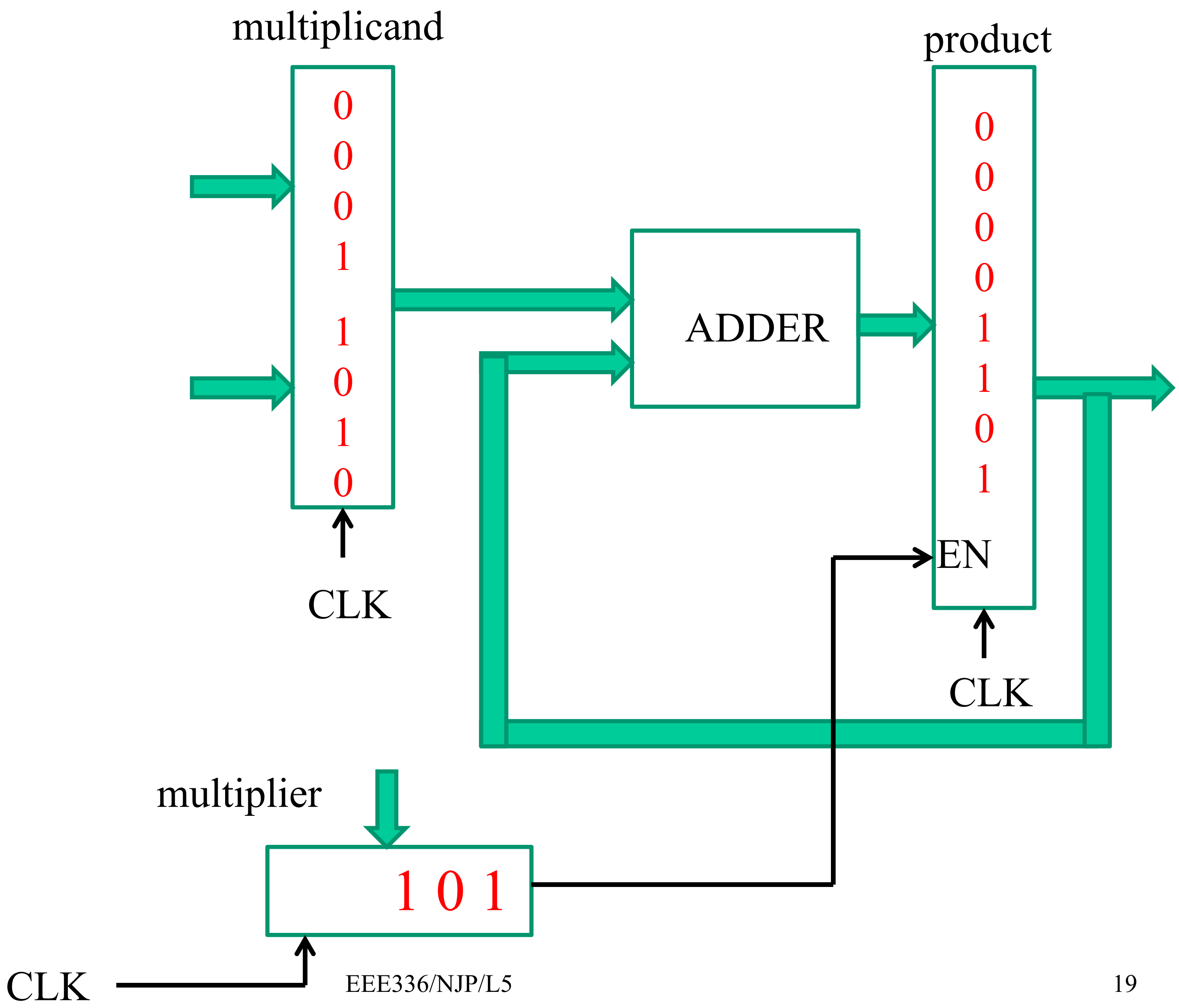
$$
\begin{array}{r}
1101 \\
\times \quad 1011 \\
\hline
1101 \\
11010 \\
000000 \\
1101000 \\
\hline
10001111
\end{array}
$$



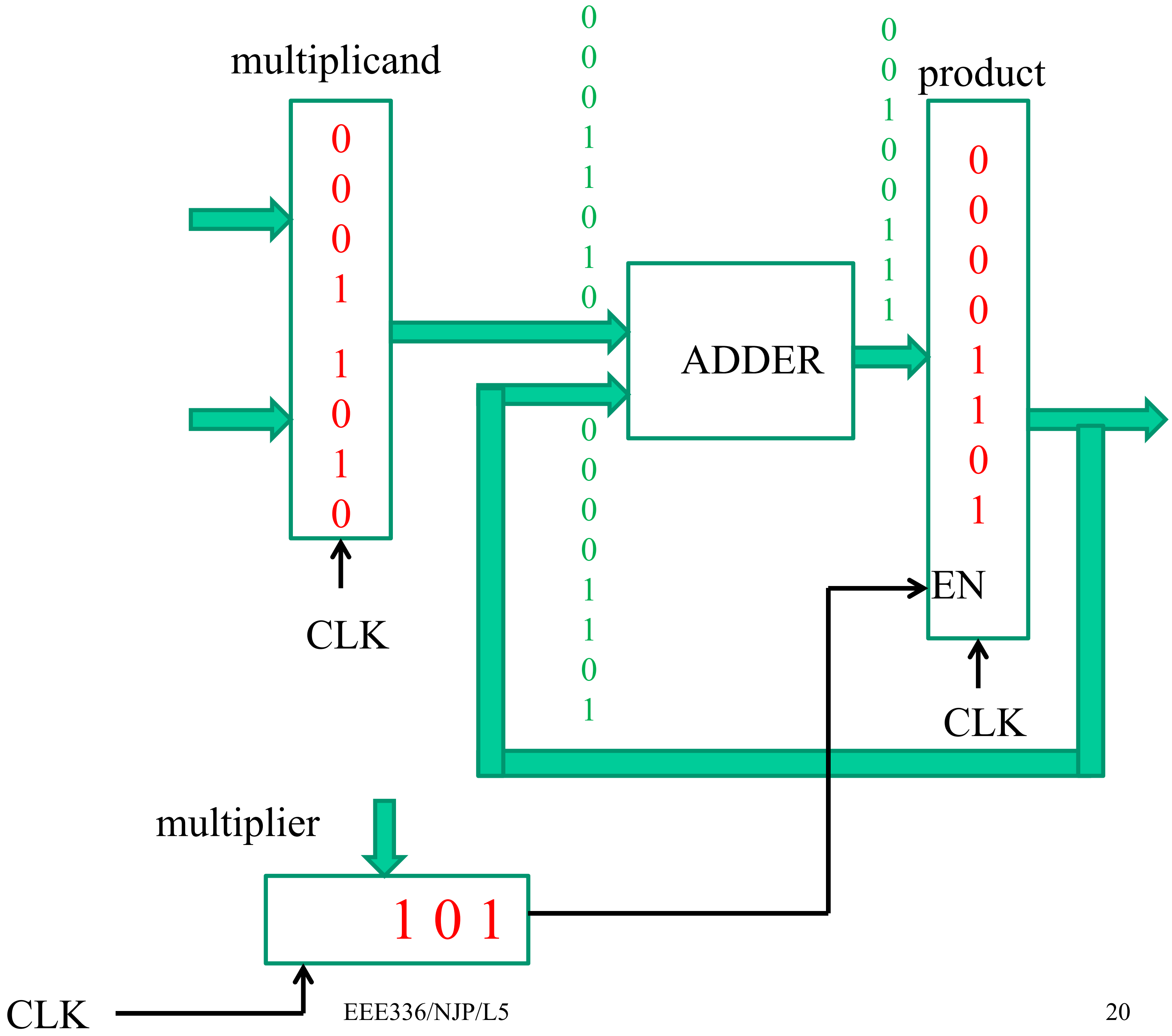
```
      1101
×     1011
----------
      1101
     11010
    000000
   1101000
----------
  10001111
```



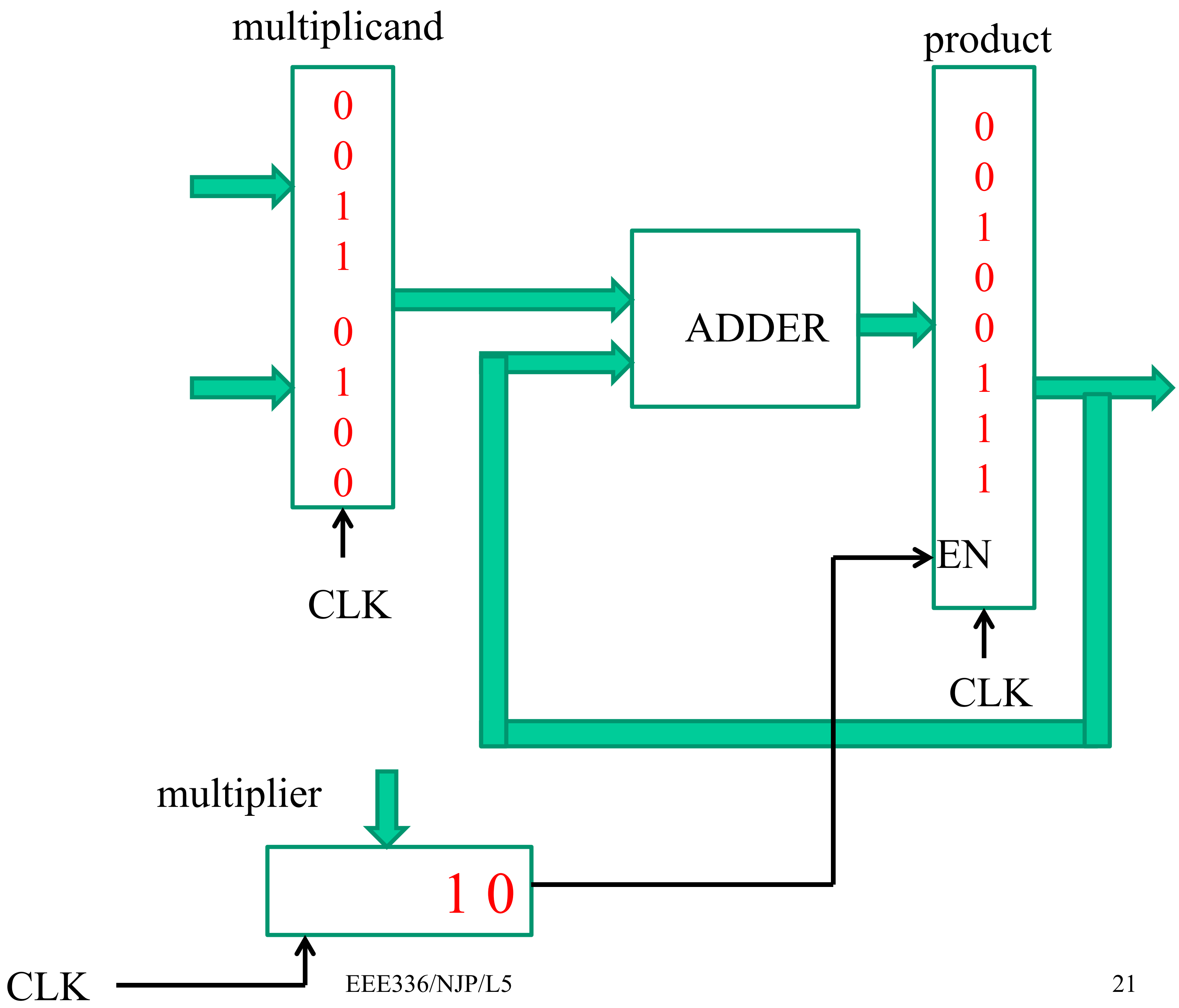
```
    1101
×   1011
--------
    1101
   11010
  000000
 1101000
--------
10001111
```



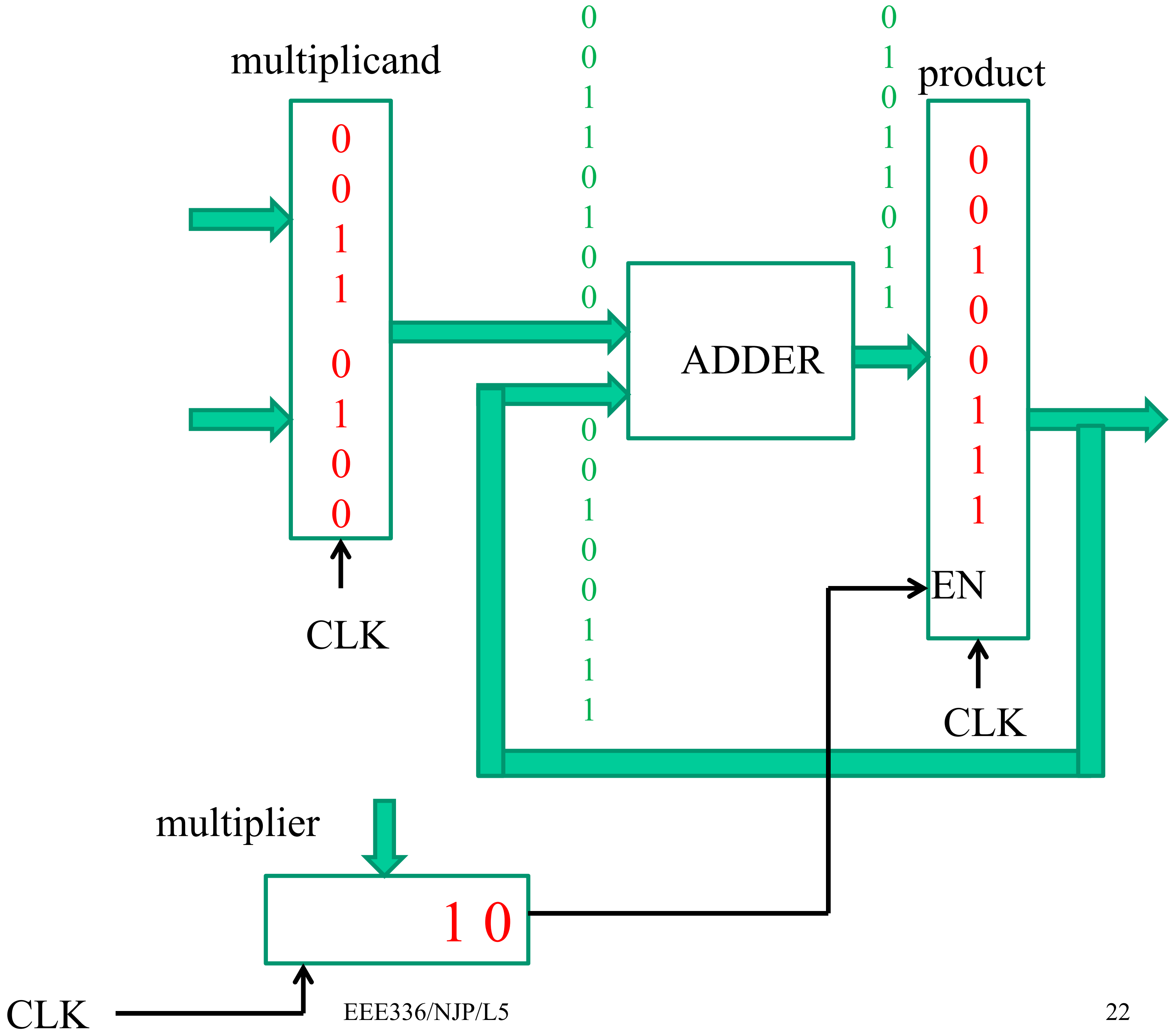
```
    1101
×   1011
--------
    1101
   11010
  000000
 1101000
--------
10001111
```



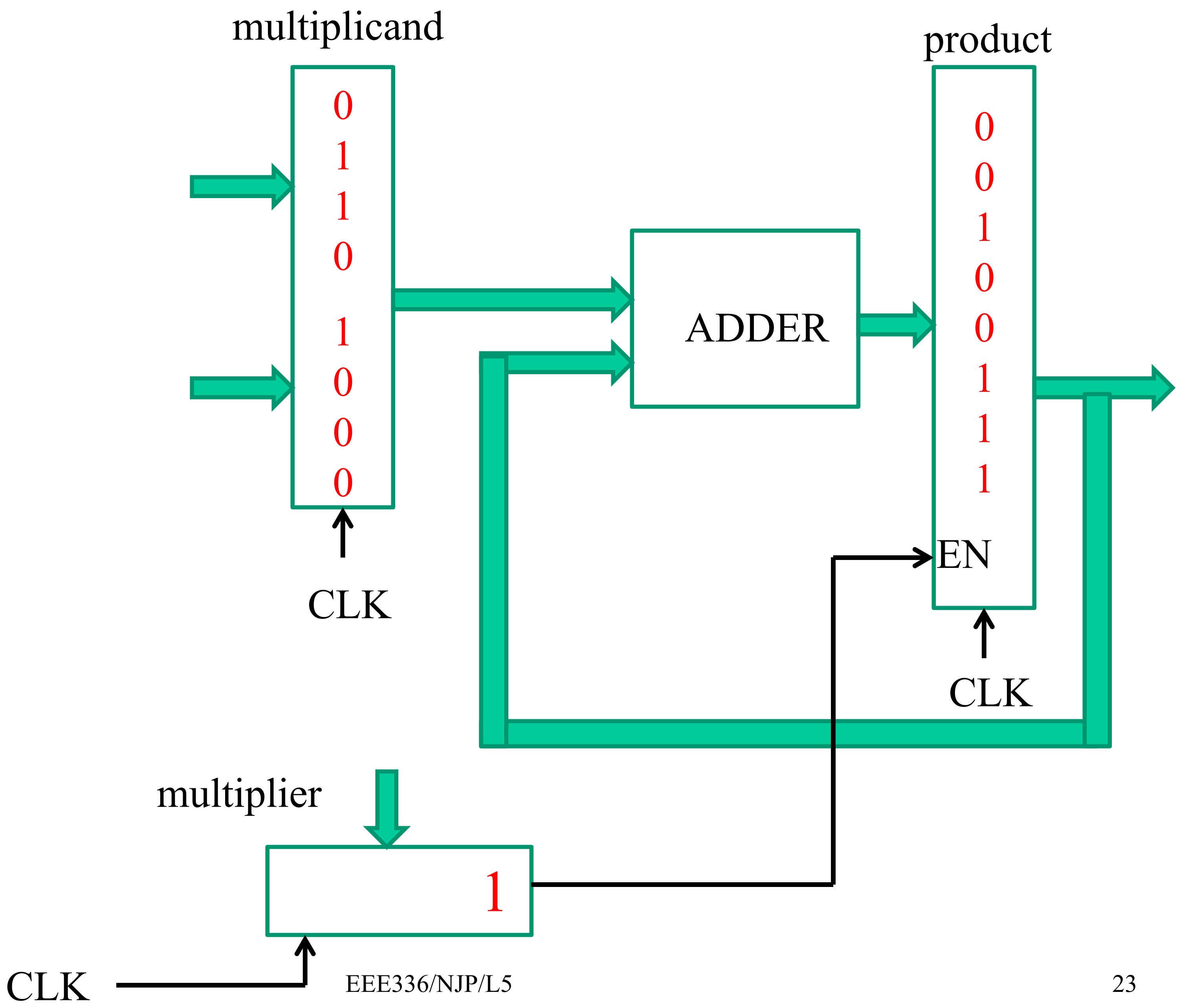
```
    1101
×   1011
--------
    1101
   11010
  000000
 1101000
--------
10001111
```



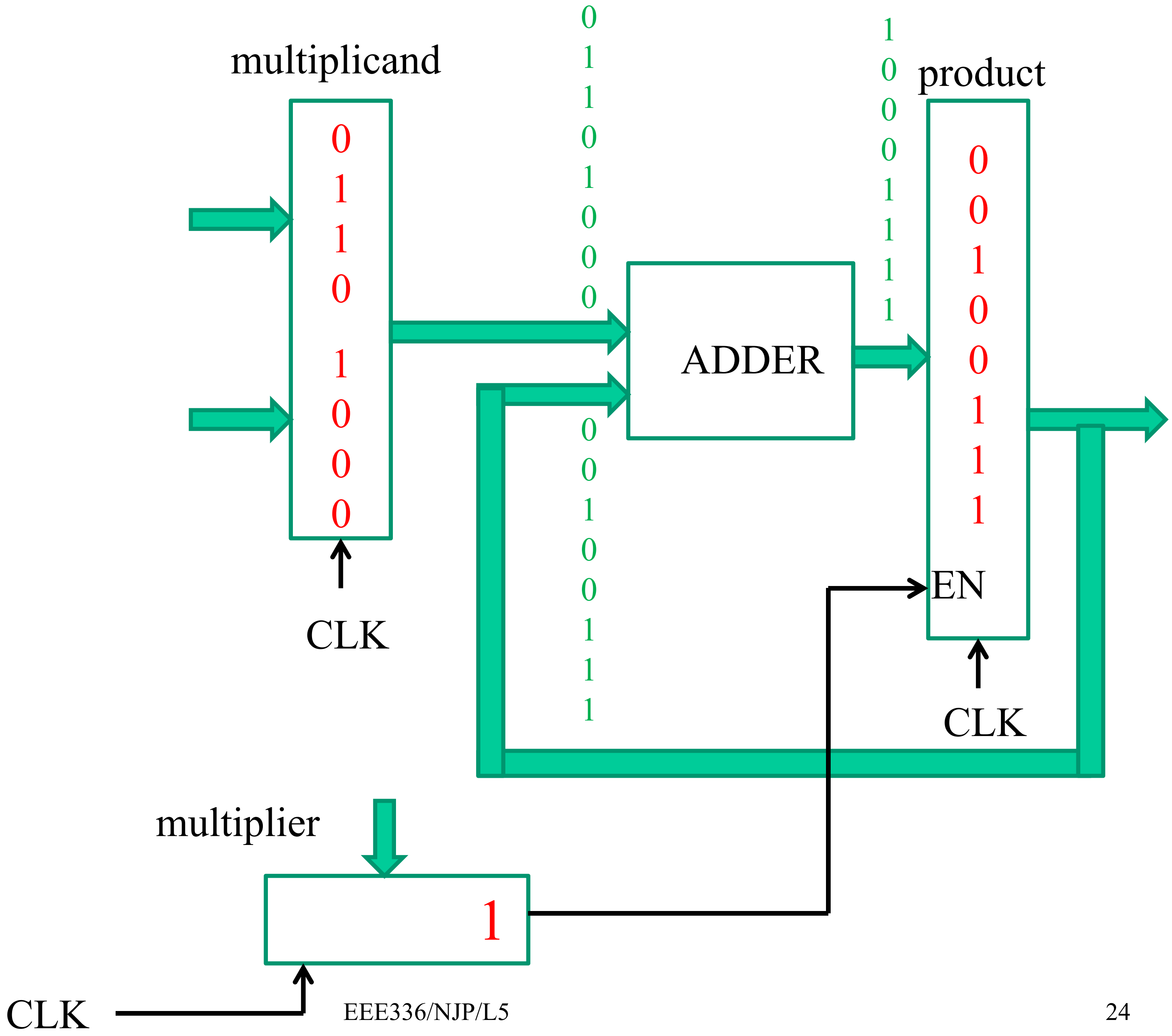
$$
\begin{array}{r}
1101 \\
\times \quad 1011 \\
\hline
1101 \\
11010 \\
000000 \\
1101000 \\
\hline
10001111
\end{array}
$$



$$
\begin{array}{r}
1101 \\
\times \quad 1011 \\
\hline
1101 \\
11010 \\
000000 \\
1101000 \\
\hline
10001111
\end{array}
$$



```
       1101
  ×    1011
  ---------
       1101
      11010
     000000
    1101000
  ---------
   10001111
```



$$
\begin{array}{r}
1101 \\
\times \quad 1011 \\
\hline
1101 \\
11010 \\
000000 \\
1101000 \\
\hline
10001111
\end{array}
$$