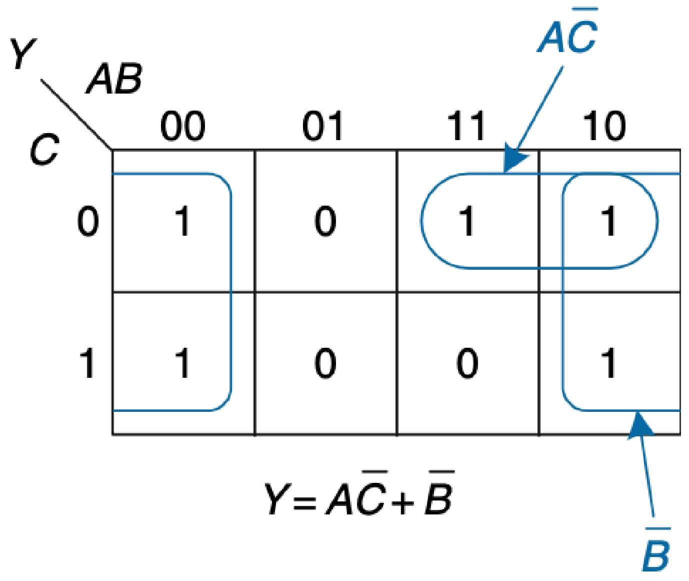
Chapter 1

Combinational Building Blocks (2.8)

1.1 K-maps

> Graphical way to find simplified equations.

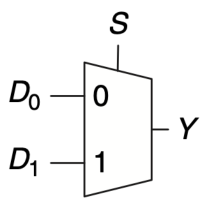
The goal is to circle the largest groups of truth outputs. For each circle, create a sum of products equation where each term includes the products that change. For example:

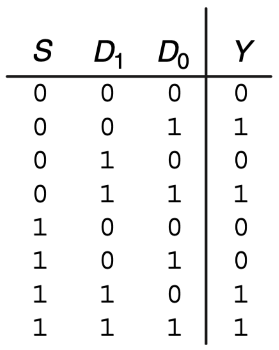


1.2 Multiplexers (Mux)

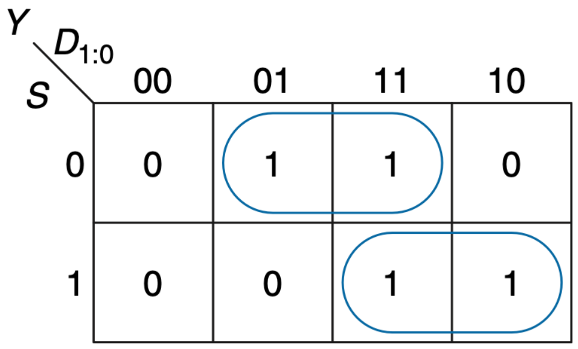
> A multiplexer is a combinational circuit that chooses among N inputs with an N-bit *select* signal

Below is a k-map and implementation of a Mux



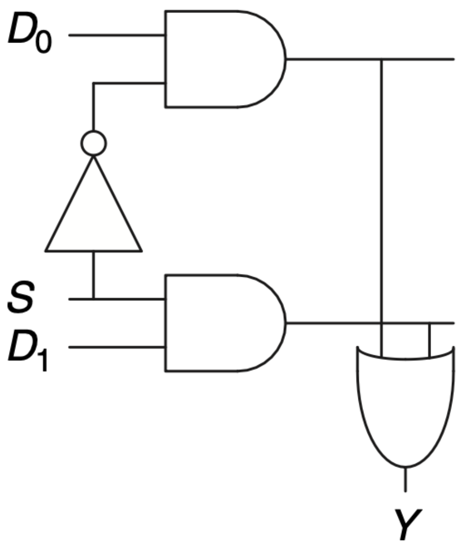


S	D_1	D_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



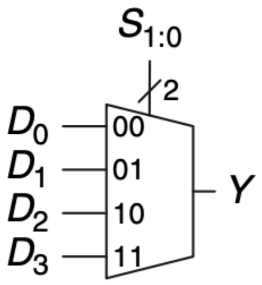
S \ $D_{1:0}$	00	01	11	10
0	0	1	1	0
1	0	0	1	1

$$Y = D_0\overline{S} + D_1 S$$

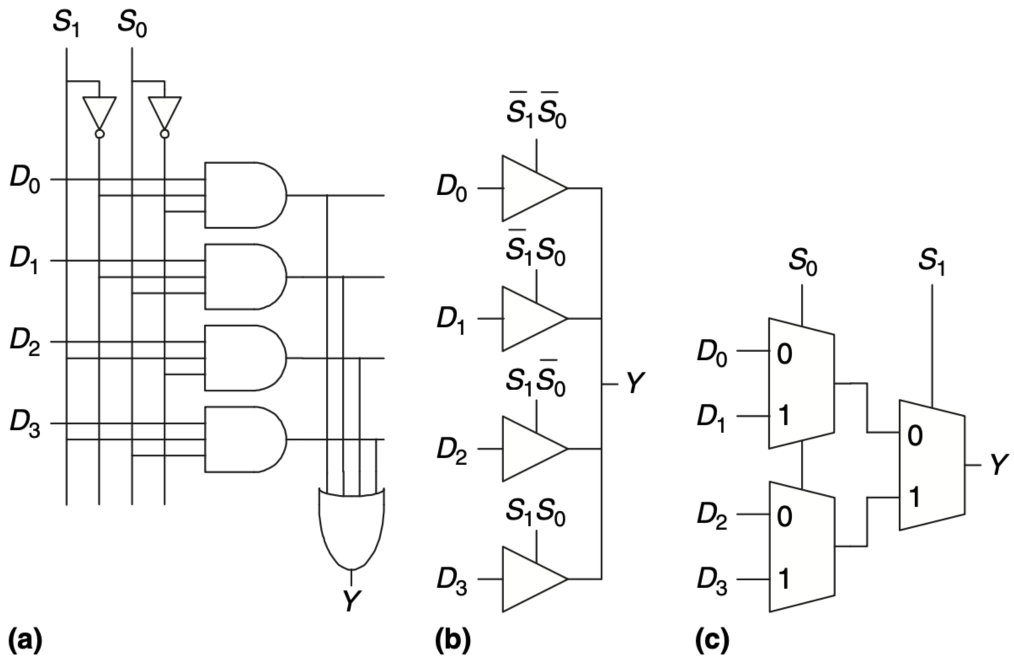


1.3 Larger Multiplexers

Larger multiplexers can be used as well. We will need a *select* signal of $\lg(n)$ for an n:1 input multiplexer



There are three ways to implement a mux: Two level logic, Tri-states, and hierarchical

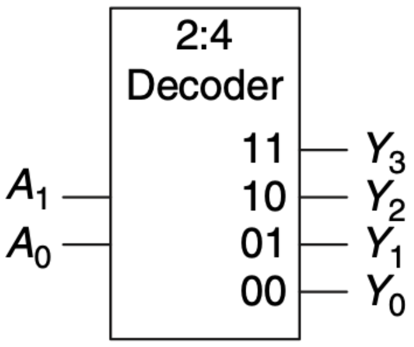


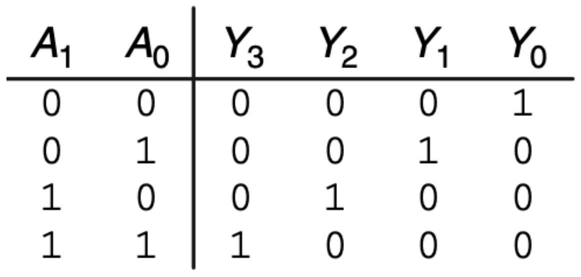
Larger Multiplexers can also be used as *lookup* tables to perform any function and can be reprogrammed with different inputs. In general, a 2^n input Multiplexer can be "reprogrammed" to perform any n-input logic function. In addition, one strategy for cutting the size in half. You do this by providing one of the literals of the logic function to the input of the Mux. This allows for 2^{n-1} input Mux.

1.4 Decoders

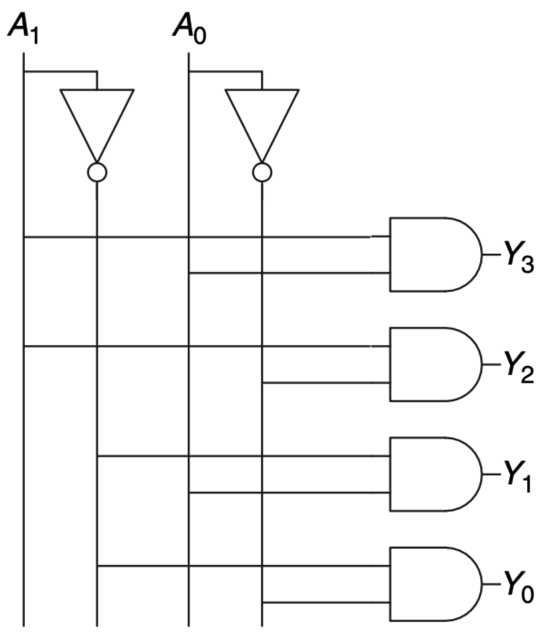
A combinational circuit that transforms N-bit inputs to a 2^n output

A decorder is, in a sense, the opposite of a Mux. It has N inputs, and 2^n outputs. The outputs are *one hot*. That is, only one is True / 1.

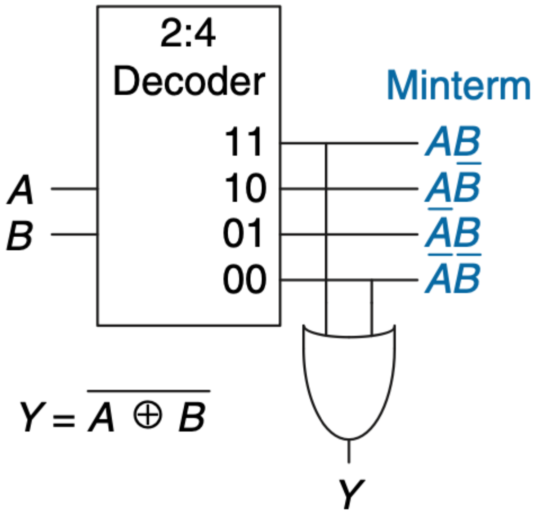




A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0



Decoders combined with OR gates can be used to create other logic functions. Since they are *one-hot* they represent a single min-term. It makes it easy to implement something like an XNOR.



Chapter 2

Arithmetic Circuits (5.2)

Arithmetic circuits are combinational logic functions that perform the typical arithmetic operations (addition, subtraction, multiplication, division). They are fundemental in modern computing.

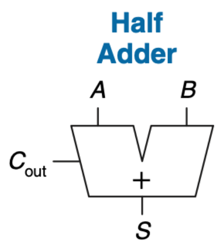
2.1 addition

2.1.1 Half Adder

> A circuit that adds two bits and has one *carry-out* bit to perform binary addition.

There are 2 inputs for each singular bit and then two outputs. The output signal S represents the single bit output and the output signal C_{out} is for the Carry-bit for the next position / adder.

They can be made from an XOR gate and an AND gate. The C_{out} bit will be used when creating multi-bit adders from these single bit ones.



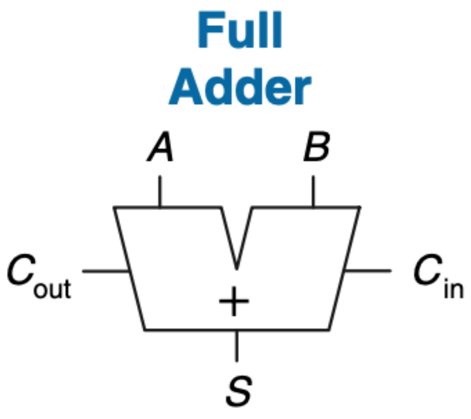
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$S = A \oplus B$$
$$C_{out} = AB$$

2.1.2 Full Adder

> Circuit that, like a half-adder performs binary addition, but also has a carry-in, C_{in}, bit

It basically adds 3 1-bit inputs in order to output two bits



C_{in}	A	B	C_{out}	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

$$S = A \oplus B \oplus C_{in}$$
$$C_{out} = AB + AC_{in} + BC_{in}$$

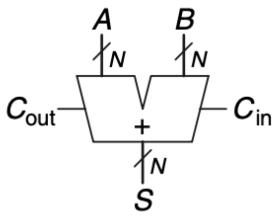
2.1.3 Carry Propagate Adder (CPA)

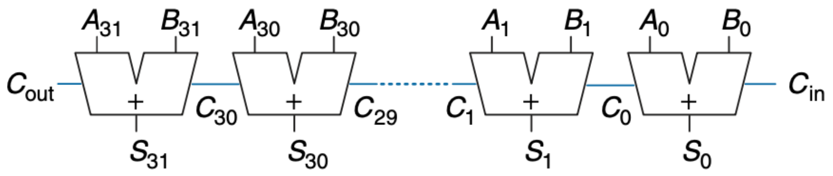
These circuits can be put together to create a Carry Propagate Adder. This takes two N-bit inputs and a 1-bit C_{in} to produce a N-bit S, and a 1-bit C_{out}.

2.1.4 Ripple Carry Adder

A linear chain of 1-bit adders to perform N-bit addition

This is the simplest way to create a CPA. Here, you chain together full adders along with 1 Half-adder at the end.





The disadvantage with Ripple Carry Adders is that it becomes slow when N is large. The bits must wait for each bit to add before it.

This is the formula for the speed of a ripple carry adder, where t_{FA} is the delay of the full adder:

$$t_{\mathrm{ripple}} = N t_{\mathrm{FA}}$$

2.1.5 Carry-Lookahead Adder

Another way to perform N-bit addition. Divides Adders into blocks to perform addition quicker.

This is another type of CPA, which tries to fix the speed problem of Ripple-carry adders. It attempts to divide the adder into blocks and determines the C_{out} as soon as the C_{in} is known.

CLA's use G Generate and P Propagate signals to find the C_{out}. These signals do not depend on previous inputs

1. A G_i signal is made if both A_i and B_i are 1
2. A P_i if a carry if either A_i or B_i are 1

$$C_i = A_iB_i + (A_i + B_i)C_{i-1} = G_i + P_iC_{i-1}$$

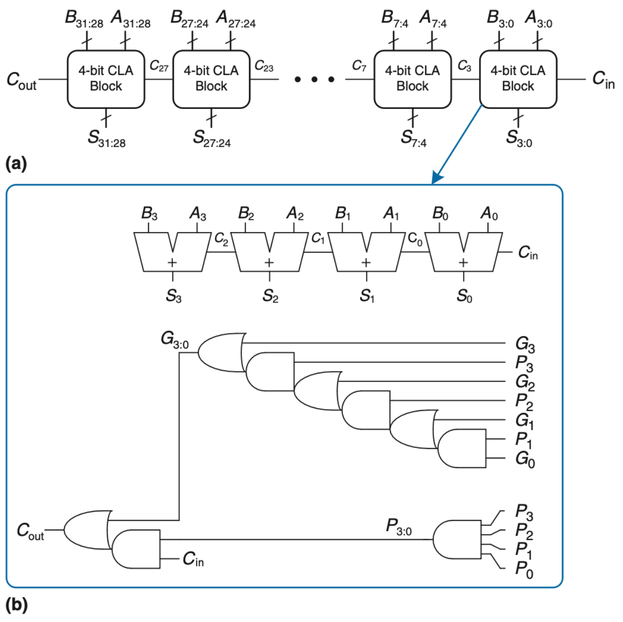
In other words, there will be a carry if the i^{th} column generates or is able to propagates a carry. We can rewrite this into blocks.

$$G_{3:0} = G_3 + P_3(G_2 + P_2(G_1 + P_1G_0))$$

$$P_{3:0} = P_3P_2P_1P_0$$

$$\boxed{C_i = G_{i:j} + P_{i:j}C_j}$$

The key is that the Generate and Propagate signals can be found immediately without waiting for the previous column



Where t_{pg} is the delay of the (P, G) gates. $t_{\text{pg\_block}}$ is the delay to find the gen/prop signals. k is the bits per block and N is the number of blocks.

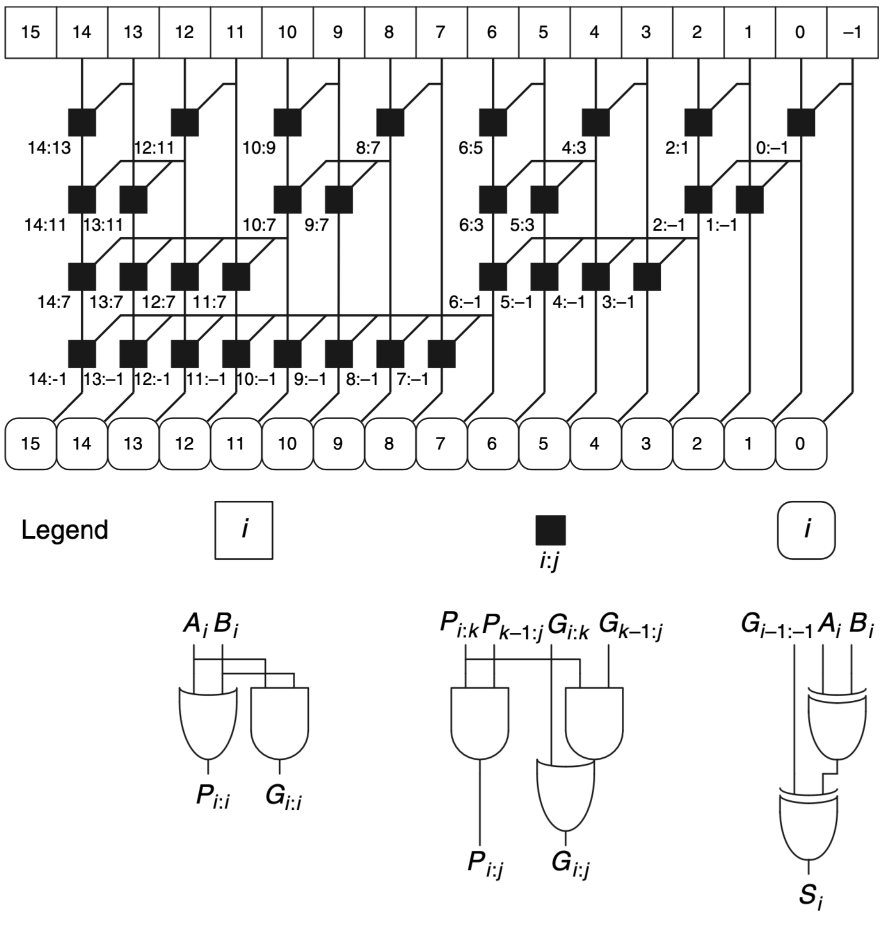
$$T_{\text{CLA}} = t_{\text{pg}} + t_{\text{pg\_block}} + \left(\frac{N}{k} - 1\right) t_{\text{AND\_OR}} + k t_{\text{FA}}$$

2.1.6 Prefix Adders

A way to perform N-bit addition like CLA, but recursively breaks down the N-bits into blocks.

These are the same as the Carry lookahead adders, however they calculate the G_i and P_i signals for each 2 columns, then every 4 columns etc.

$$S_i = (A_i \oplus B_i) \oplus C_{i-1}$$



The critical path of an N-bit prefix adder involves the combination of P_i and G_i followed by $\lg N$ stages of black prefix cells

$$t_{PA} = t_{pg} + \lg N \left(t_{pg\_\text{prefix}}\right) + t_{\text{XOR}}$$

2.1.7 Subtraction

> Using two's complement, it is exactly the same as addition. That is flipping the bits and adding 1.

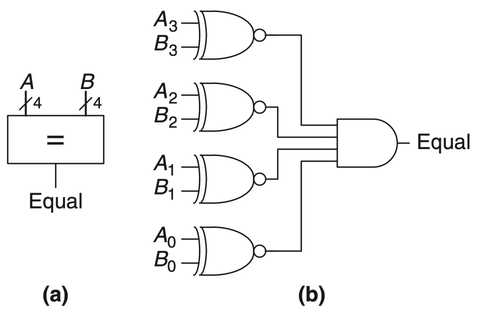
2.2 Comparators

> Determines if two binary numbers are equal or if one is less than the other

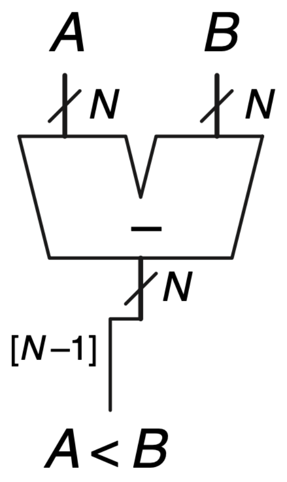
There are two types:

1. Equality Comparators
2. Magnitude Comparators

Equality Comparators only determine if N-bit inputs are equal.

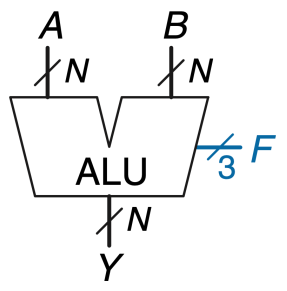


Magnitude determines if one is less than the other by subtraction and looking at the sign.



2.3 ALU

A circuit that combines different mathematical and logical operations in a single unit. Decides function from a *function* signal.

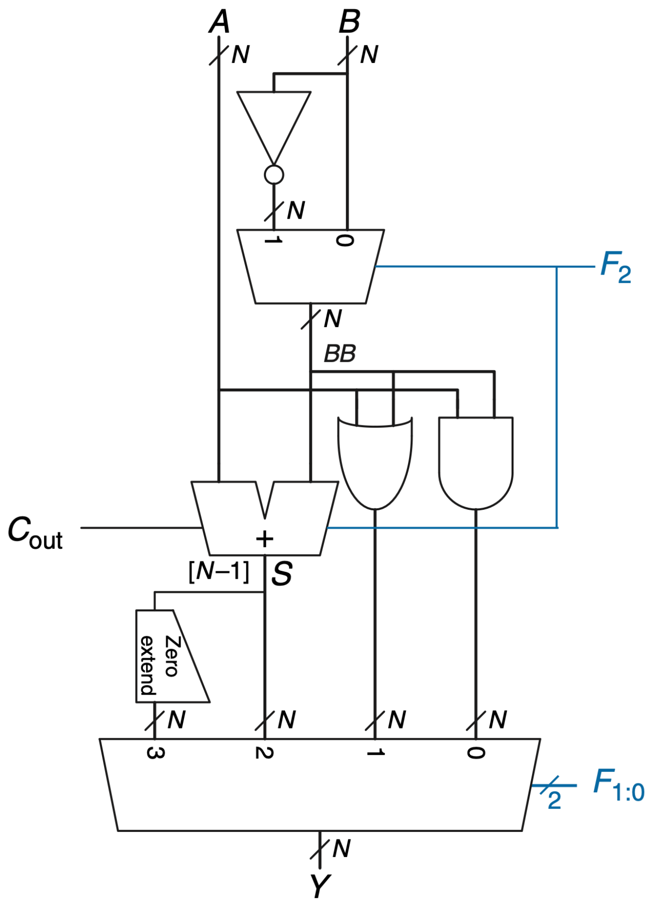


May be able to perform, addition, subtraction, Magnitude comparison, etc.

$F_{2:0}$	Function
000	A AND B
001	A OR B
010	A + B
011	not used
100	A AND $\overline{B}$
101	A OR $\overline{B}$
110	A – B
111	SLT

2.3.1 ALU Implementation

Here we can see the implementation of an ALU.



It takes an input of A and B. And the *Function* signal first decides whether to invert B. If the function is to add them, it does that. If it is to subtract, F_2 will be 1 and there will be a carry-in when adding the inputs. Finally, it choses which output from a 4:1 mux.

2.4 Shifters and Rotators

Shifters and rotators simply shift bits or divide by powers of 2.

Logical Shifter: Shifts bits to the left (LSR) or right (LSR) and fills empty spots with 0. This loses the data.

Arithmetic Shifter: Same as Logical shifter, but shifts right and copies the old MSB to empty spots. This also loses data. But is useful for keeping signs.

Rotator Shifts bits left or right, but keeps the data. Empty spots are filled with bits from the opposite side.

An N-bit shifter can be built with N, N:1, Muxes.

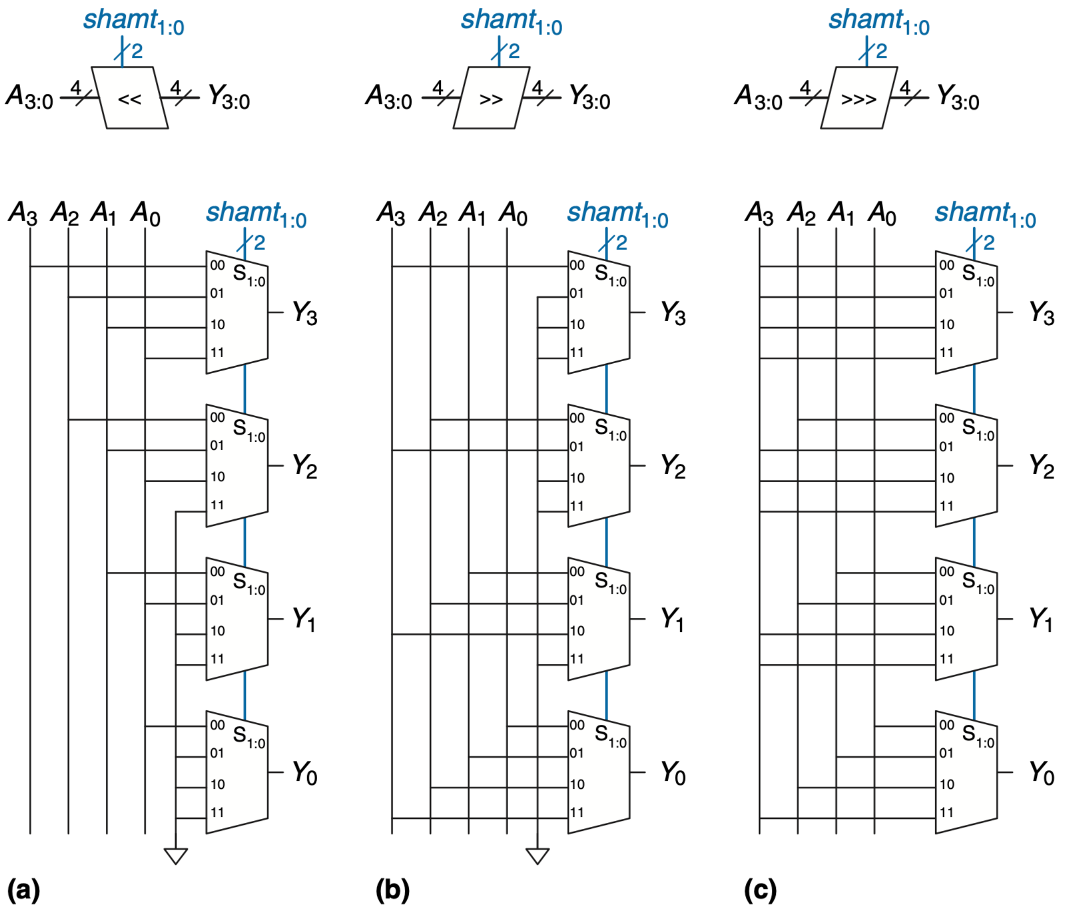


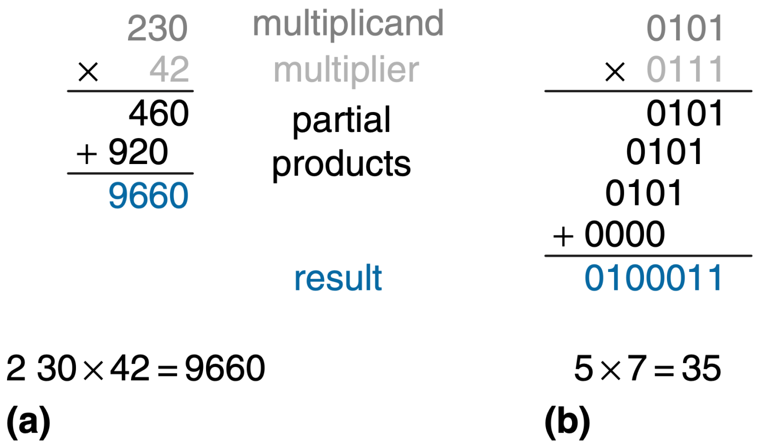
Figure 2.1: (A): Left Shifter, (B): Right Shifter, (C): Logial Right Shifter

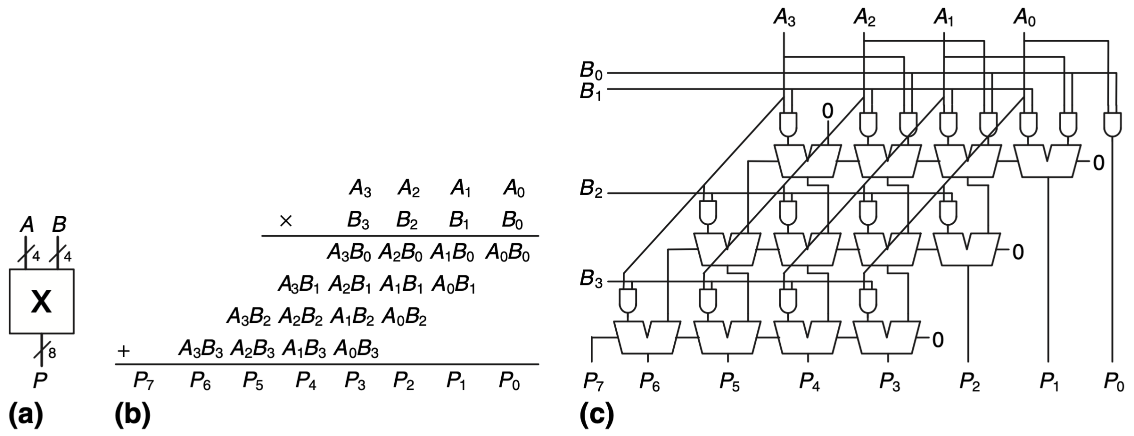
Note:-
Shifting is the same as multiplying and dividing by powers of 2^n

2.4.1 Multiplication

> Multiplication of two N-bit numbers, A and B, involve shifting A from 0 to N bits determined by B and summing the result

In general, a N x N multiplication will result in a 2N product.





2.4.2 Division