Logic Design

Ref: Appendix B

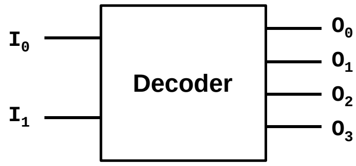
Common Components

- There are many commonly used components in processor design.
- We will use these components when we design control systems (later).
- We will look at the functionality and design of some of these components now.

Some commonly used components

- Decoders: n inputs, 2^n outputs.
  - *the inputs are used to select which output is turned on. At any time exactly one output is on.*
- Multiplexors: 2^n inputs, n selection bits, 1 output.
  - *the selection bits determine which input will become the output.*

2 input Decoder



Treat I_0I_1 as a 2 bit integer i. The i^{th} output will be turned on (O_i=1), all the others off.

4

Decoder Truth Table

I_0	I_1	O_0	O_1	O_2	O_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

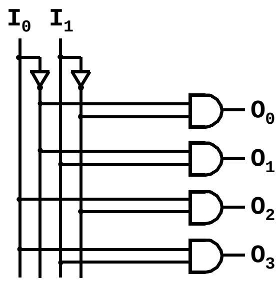
5

Decoder Boolean Expressions

$$O_0 = \overline{I_0} \bullet \overline{I_1}$$
$$O_1 = \overline{I_0} \bullet I_1$$
$$O_2 = I_0 \bullet \overline{I_1}$$
$$O_3 = I_0 \bullet I_1$$

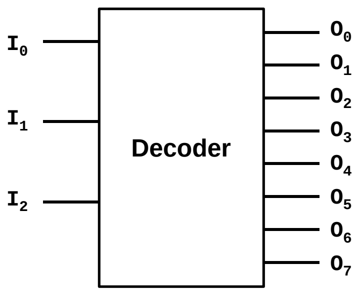
6

Decoder Implementation



7

3 Input Decoder



8

3 Input Decoder Truth Table

I_0	I_1	I_2	O_0	O_1	O_2	O_3	O_4	O_5	O_6	O_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

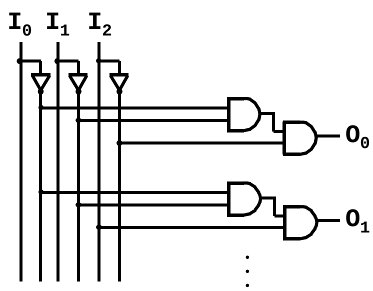
9

3-Decoder Boolean Expressions

O_0	$= \overline{I_0} \bullet \overline{I_1} \bullet \overline{I_2}$	*000*
O_1	$= \overline{I_0} \bullet \overline{I_1} \bullet I_2$	*001*
O_2	$= \overline{I_0} \bullet I_1 \bullet \overline{I_2}$	*010*
O_3	$= \overline{I_0} \bullet I_1 \bullet I_2$	*011*
O_4	$= I_0 \bullet \overline{I_1} \bullet \overline{I_2}$	*100*
O_5	$= I_0 \bullet \overline{I_1} \bullet I_2$	*101*
O_6	$= I_0 \bullet I_1 \bullet \overline{I_2}$	*110*
O_7	$= I_0 \bullet I_1 \bullet I_2$	*111*

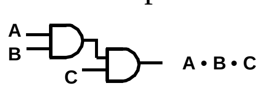
10

3-Decoder Partial Implementation

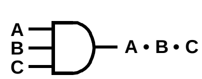


11

A Useful Simplification



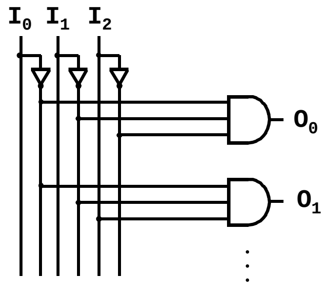
The above logic diagram is often abbreviated as shown below:



We can do this (without possible confusion) because of the associative property.

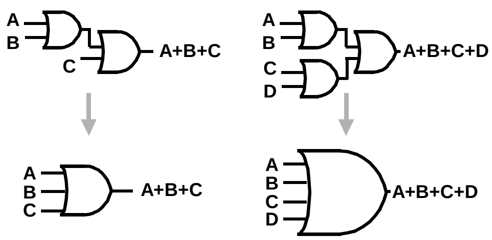
12

Revised Partial 3-Decoder



13

Multiple Input Or Gates

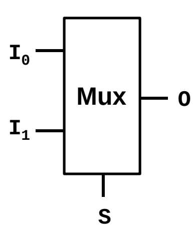


14

2 Input Multiplexor

Inputs: I_0 and I_1
Selector: **S**
Output: **O**

If **S** is a 0: $O=I_0$
If **S** is a 1: $O=I_1$



15

2-Mux Boolean Function

- The output depends on I_0 and I_1
- The output also depends on **S** !!!
- We must treat **S** as an input.

$$O = f(I_0, I_1, S)$$

16

2-Mux Truth Table

Abbreviated Truth Table

S	O
0	I_0
1	I_1

S	I_0	I_1	O_0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

17

2-Mux Boolean Expression

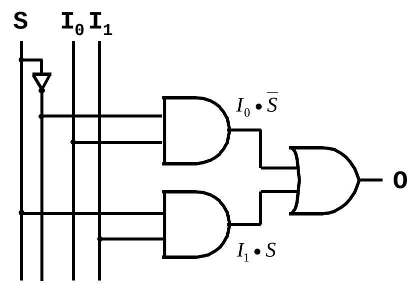
$$O = (I_0 \bullet \overline{S}) + (I_1 \bullet S)$$

terms

Since **S** can't be both a 1 and a 0, only one of the *terms* can be a 1.

18

2-Mux Logic Design



19

4 Input Multiplexor

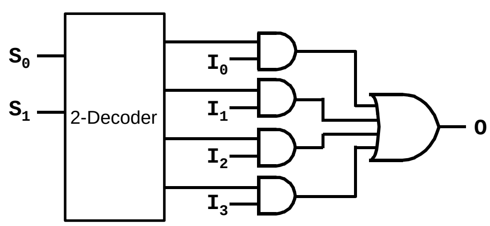
- If we have 4 inputs, we need to have 2 selection bits: S_0 S_1

Abbreviated Truth Table

S_0	S_1	O
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

20

One Possible 4-Mux



21

Common Implementations

- There are two general forms that are used in many circuit implementations:
  - Product of Sums
    - A bunch of ORs leading to a big AND gate
  - Sum of Products
    - A bunch of ANDs leading to a big OR gate

22

Sum of Products

- Express the function by listing all the combinations of inputs for which the output should be a 1.
- These combinations are rows in the truth table where the function has the value 1.
- Represent each combination with an AND gate.
  - Inputs are negated if they correspond to 0s, not if they need to be 1.
- OR the outputs of all the AND gates to generate the final output.

23

SOP Example: 2-Mux

Find rows in truth table where the output is 1.

If S is 1 in that row, connect **S** to a 3-input AND gate, otherwise connect $\overline{S}$.

Connect I_0 and I_1 in the same way.

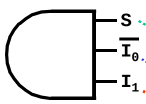
Each AND gate corresponds to the row in the truth table.

S	I_0	I_1	O
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

24

SOP Example: 2-Mux (cont).

If the output of this AND gate is a 1,the value of the function is a 1!



S	I_0	I_1	O
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

25

SOP Construction

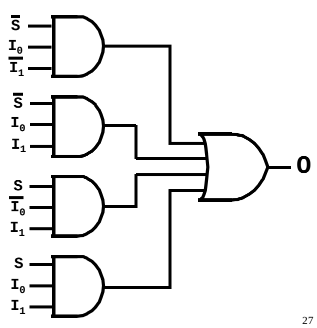
- For each row on the truth table that has the value 1 (the function has the value 1) build the corresponding AND gate.
- Ignore all rows where the function has the value 0!
- Connect the output of all the AND gates to one big OR gate.

26

4-Mux Sum Of Products

Truth Table

S	I_0	I_1	O_0
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



27

Product of Sums

- Express the function by listing all the combinations of inputs for which the output should be a 0.
- These combinations are rows in the truth table where the function has the value 0.
- Represent each combination with an OR gate.
  - Each input that has to be a 1 must be negated!
- AND all the OR gates to generate the output.

28

POS Example: 2-Mux

Find rows in truth table where the output is 0.

If S is 0 in that row, connect **S** to a 3-input OR gate, otherwise connect $\overline{S}$.

Connect I_0 and I_1 in the same way.

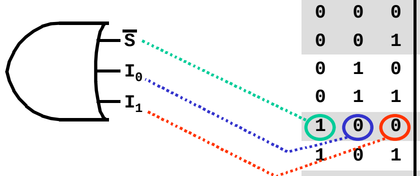
The OR gate corresponds to the row in the truth table.

S	I_0	I_1	O
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

29

POS Example: 2-Mux (cont).

If the output of this OR gate is a 0, the value of the function is a 0!



S	I_0	I_1	O
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

30

POS Construction

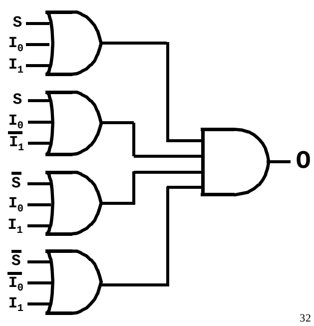
- For each row on the truth table that has the value 0 (the function has the value 0) build the corresponding OR gate.
- Ignore all rows where the function has the value 1!
- Connect the output of all the OR gates to one big AND gate.

31

4-Mux Product of Sums

Truth Table

S	I_0	I_1	O
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



32

Minimization

- SOP and POS forms provide a simple translation from truth table to circuit.
- The resulting designs may involve more gates than are necessary.
- There are a number of techniques used to minimize such circuits.

33

Minimization Techniques

- Boolean Algebra
  - use postulates and identities to reduce expressions.
- Karnaugh Maps
  - graphical technique useful for small circuits (no more than 4 or 5 inputs)
- Tabular Methods
  - suitable for large functions – usually done by a computer program.

34

Karnaugh Map (K-map)

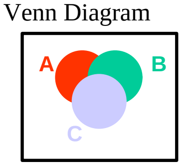
- Based on SOP form.
- It may be possible to *merge* terms.
- Example: $\mathbf{f} = (\mathbf{A} \bullet \mathbf{B} \bullet \mathbf{C}) + (\overline{\mathbf{A}} \bullet \mathbf{B} \bullet \mathbf{C})$
  - Close inspection reveals that it doesn't matter what the value of A is!
  - Here is a simpler version of the same function:

$$\mathbf{f} = (\mathbf{B} \bullet \mathbf{C})$$

35

Graphical Representation

- The idea is to draw a picture in which it will be easy to see when *terms* can be merged.
- We draw the truth table in 2-D, the result is similar to a Venn Diagram



36

K-Map Example $f = A \bullet B + \overline{A} \bullet B$

Truth Table

A	B	f
0	0	0
0	1	1
1	0	0
1	1	1

K-Map

	B=0	B=1
A=0	0	1
A=1	0	1

In the K-Map it's easy to see that the value of *A* doesn't matter

37

Another Example: The Majority Function

- The majority function is 1 whenever the majority of the inputs are 1.
- Here is an SOP Boolean equation for the 3-input majority function:

$$f = A \bullet B \bullet C + \overline{A} \bullet B \bullet C + A \bullet \overline{B} \bullet C + A \bullet B \bullet \overline{C}$$

38

K-Map for Majority Function

Truth Table

A	B	C	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

K-Map

C \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

39

K-Map Construction

C \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

This order of labeling is critical!

- Notice that any 2 adjacent cells differ by exactly one bit in the input.
  - either A is different, or B is different or C is different.
  - Never more then 1 variable is different!

40

How to use K-Map

K-Map

C \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

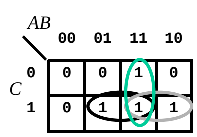
Rectangular collections of cells that all have the value 1 indicate it is possible to *merge* the corresponding terms in SOP expression.

The number of cells in the rectangle must be a power of 2!

41

Possible Mergings

K-Map



- There are 3 possible *mergings* of terms in this K-Map.

42

One of the merges

K-Map

C \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

- The merge shown means "if C is 1 and B is 1, it doesn't matter what the value of A is"

$$\overline{A}\bullet B\bullet C + A\bullet B\bullet C = B\bullet C$$

43

All 3 reductions

K-Map

C \ AB	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Original: $f = A\bullet B\bullet C + \overline{A}\bullet B\bullet C + A\bullet \overline{B}\bullet C + A\bullet B\bullet \overline{C}$

Reduced: $f = B\bullet C + A\bullet C + B\bullet C$

44

K-Map Concept

- A professional *Logic Designer* would need to use minimization techniques every day.
- We are just amateurs, so all we need to know is the general idea.
  - that there are systematic procedures for minimizing SOP and POS form Boolean equations.

45

Possible Test Questions

- Given a description of a function (or number of related boolean functions), derive the SOP and POS expressions.
- Draw a logic diagram (bunch of connected and labeled gates) that implements a given function.
- Know what a decoder and multiplexor are. Be able to design them.

46