Nand To Tetris

Kunwar Shaanjeet Singh Grover

Contents

Chapter 1

Introduction

1.1 The Big Picture

In Part-1 we will build hardware of the computer. In Part-2 we will complete the picture and build the software heirarchy of the computer.

1.1.1 The Road Ahead

How do you actually print "Hello World"? Not writing code for it, but how does it actually work? Why don't we have to worry about it? We only care about "what" is to be done.

"How" ← Implementation "What" ← Abstraction

But who will worry about the "how"? Someone has to do it. A nice thing about computers is once we have done the "how" we only need to worry about the "what".

1.1.2 Multiple Layers of Abstraction

Once we have built the lower level, we dont need to worry about it and can abstract it.

Every week, we will worry about a single level, take the lower level as given, implement the higher level and test that it works.

By the end of the course, we will have built a complete functioning computer and can run anything including games like Tetris.

1.1.3 Two Parts

1. Part-I : Hardware
   (a) Start with *Nand*
   (b) Create the *HACK* computer
2. Part-II : Software
   (a) Start with the *HACK* computer
   (b) Create a fill sotware hierarchy that ...
   (c) ... runs applications like *Tetris*

1.1.4 From Nand To Hack

Nand $\xrightarrow{\text{Combinational Logic}}$ Elementary Logic Gates $\xrightarrow{\text{Comb. and Seq. Logic}}$ CPU, RAM, chipset $\xrightarrow{\text{Digital Design}}$ Computer Architecture $\xrightarrow{\text{Assembler}}$ Low Level Code

1.1.5 How to build a chip

We will use build our chip on a hardware simulator. We will do this in a HDL (Harware Discription Language).

Chapter 2

Boolean Functions and Gate Logic

2.1 Boolean Logic

$x\ AND\ y \rightarrow x \wedge y$
$x\ OR\ y \rightarrow x \vee y$
$NOT(x) \rightarrow \neg x$

2.1.1 Boolean Functions

$f(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$

Table 2.1: $f(x, y, z)$

x	y	z	f
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

Both are identical representations

2.1.2 Boolean Identies

Commutative Laws:

$$x \wedge y = y \wedge x$$
$$x \vee y = y \vee x$$

Associative Laws:

$$x \wedge (y \wedge z) = (x \wedge y) \wedge z$$
$$x \vee (y \vee z) = (x \vee y) \vee z$$

Distributive Laws:

$$x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$$
$$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$$

De Morgan Laws:

$$\neg(x \wedge y) = \neg(x) \vee \neg(y)$$
$$\neg(x \vee y) = \neg(x) \wedge \neg(y)$$

2.1.3 Boolean Functions Synthesis

Given a Truth Table, how do we construct a boolean function for it?

Lets take an example,

Table 2.2: Truth Table for a Boolean Function

x	y	z	f
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

This table can be represented by taking OR of the "true"" statements. We can represent the "true" statements by a boolean expression. For example, $x = 0, y = 0, z = 1, f = 1$ can be represented by $\neg x \wedge \neg y \wedge z$. This expression is only true when $x = 0, y = 0, z = 1$ and false on all other cases. So, we can

represent every boolean function by AND of such terms.

The truth table 2.2 can be represented by the expression:

$$(\neg x \wedge \neg y \wedge z) \vee (x \wedge y \wedge \neg z)$$

But what is the minimum size expression we can build from a truth table? This is a NP-Complete problem and cannot be solved in polynomial time if $P \neq NP$.

2.1.4 Why NAND?

We can represent every boolean expression only using NAND. This can be trivially proved.

2.2 Gate Logic

A technique for implementing Boolean functions using logic gates.

Logic Gates:

1. Elementary (Nand, And, Or, Not)
2. Composite (Mux, Adder)

2.2.1 Elementary logic gates: Nand

Gate Diagram:

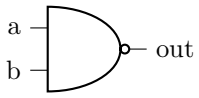

Figure 2.1: A Nand Gate

Function Specification:
if (a == 1 and b == 1): then out=0 else out=1

2.2.2 Elementary logic Gates: And, Or, Not

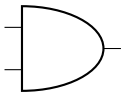

Figure 2.2: A And Gate

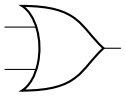

Figure 2.3: A Or Gate

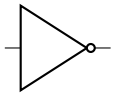

Figure 2.4: A Not Gate

2.2.3 Composite Gates

A 3-Input And Gate is an example of a composite gate. It can be build using two 2-Input And Gates.

Functional Specification:
if (a==1 and b==1 and c==1): then out=1 else out=0

2.3 Hardware Description Language