EE2001 - Digital systems lab

Vinita Vasudevan

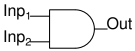
Experiment 1: Logic Simulation using C

Objective: Describe a simple circuit using logic gates and simulate for all possible inputs to test for correctness. Write the simulator using C programming language.

- Write functions to describe basic gates.
- Write a function to describe the circuit using these gates.
- Write a testbench

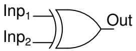
Gates - Symbol and Truth Table

Output	Inp1	Inp2
0	0	0
0	0	1
0	1	0
1	1	1



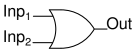
(a) AND Gate

Output	Inp1	Inp2
0	0	0
1	0	1
1	1	0
0	1	1



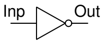
(c) XOR Gate

Output	Inp1	Inp2
0	0	0
1	0	1
1	1	0
1	1	1



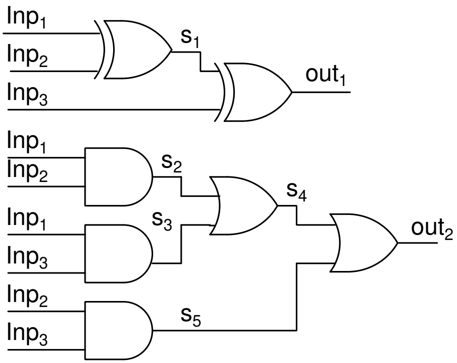
(b) OR Gate

Output	Inp1
1	0
0	1



(d) NOT Gate

Sample circuit



Inp_1	Inp_2	Inp_3	out_1	out_2
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

- Test vectors: The set of all combinations of the inputs Inp_1, Inp_2, Inp_3.
- Build the circuit and test output for all combinations - Not practical

Logic Simulator

- Design a circuit for a given Boolean function
- Read in test vectors from a file

Time	Inp_1	Inp_2	Inp_3
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

- Write functions to emulate gates
- Write a function to describe the circuit (interconnection between gates)
- Main program to read in inputs and get outputs.
- Two ways to write it - behavioural, structural model of the circuit.

Logic Simulation using C

Data Types in C

- char (1 byte)
- short int (2 bytes)
- int (4 bytes)
- long int (8 bytes)
- float (4 bytes)
- double (8 bytes)

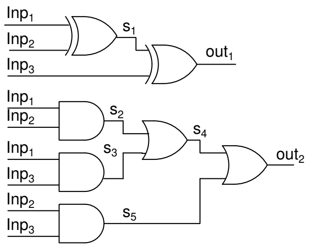
Bit Operations

- AND: &
- OR: |
- XOR: ^
- NOT: ~

Example:

```
unsigned short int a, b;
a = 8 (0000000000001000); b = 10 (0000000000001010);
printf("%hu %hu %hu", a^b, a&b, ~a);
2   8   65527(1111111111110111)
```

Behavioural Model



$$
\begin{aligned}
s1 &= Inp_1 \hat{}\ Inp_2; \\
out_1 &= s1\ \hat{}\ Inp_3; \\
s2 &= Inp_1 \&\ Inp_2; \\
s3 &= Inp_1\ \&\ Inp_3; \\
s5 &= Inp_2\ \&\ Inp_3; \\
s4 &= s2 \mid s3; \\
out_2 &= s4 \mid s5;
\end{aligned}
$$

Simulator - Structural model

gates.C: Should have all the functions need to simulate the gate.
Example:

```
void AND(unsigned short int *out, unsigned short int Inp1, unsigned short int Inp2
{
  *out = Inp1 & Inp2;
}
```

Circuit.C: Should describe the circuit in terms of gates
Example:

```
void circuit( unsigned short int *out1, unsigned short int *out2,
               unsigned short int Inp1, unsigned short int Inp2)
{
  void NOT( unsigned short int*, unsigned short int);
  void NAND( unsigned short int*, unsigned short int, unsigned short int);

  unsigned short int s1;

  NAND(&s1, Inp1, Inp2);
  NAND(out1, Inp2, Inp3);
  NOT(out2, Inp2);
}
```

Read from a file using fscanf

main.C: Testbench. It reads values of input signals from a file, passes it to the function “circuit” and prints output.

- The format is fscanf(formatting string, arguments). The arguments must be pointers.
  Example: fscanf(fin, ”%d %hu %hu ”, &time, &Inp1, &Inp2)
- fscanf stops reading when its formatting string is exhausted. On end of file EOF is returned.
- The next call resumes searching immediately after the last character read.

The formatting string can contain

- Blank or tab which are ignored ("%d%d%d" same as "%d %d %d")
- Ordinary characters other than %, which are expected to match the next non-whitespace character. ("%d, %d": comma separated inputs. "%d %d": tab or space separated inputs)
- fscanf also reads across line boundaries since newlines are treated as whitespace. (Whitespace characters are blank, tab, newline, carriage return).

So to read a file, you can put it in a "while" loop as follows.

```
while (fscanf(fin, "%d %hu %hu ", &time, &Inp1, &Inp2)!=EOF)
{
  printf("Inp1=%hu, Inp2=%hu\n", Inp1, Inp2);
}
```

- Pass outputs as pointers to the function. That way a function can return more than one output variables.

```
int main(void)
{
  void Circuit(int *, int*, int, int);
  int out1, out2;
  int a,b;

  C1(&out1, &out2, a, b);

  return 0;
}
void Circuit(int*out1, int*out2, int a, int b)
{
  *out1 = .....
  *out2 = ......
}
```

Pointers - When and why do you get segmentation fault?

Example:

```
int main(void)
{
  void AND(int *, int, int);
  int *out;
  int a,b;

  AND(out, a, b);

  return 0;
}
void AND(int*out, int a, int b)
{
  *out = a & b;  This is the problem
}
```

This will most likely result in a segmentation fault. You have declared a pointer, but it does not necessarily point to a valid location. Most likely it is pointing to location 0, reserved for the operating system. So when you try and assign a value to it, it gives a segmentation fault.

Two ways to fix it.

```
int main(void)
{ void AND(int *, int , int );
   int *out;
   int a,b;

   out = (*int)malloc(sizeof(int));
   AND(out , a , b);

   free(out);
   return 0;
}
```

Explicity request space and have the pointer out point to that space.

Second method - I prefer this:

```
int main(void)
{
  void AND(int *, int , int );
  int out;
  int a,b;

  AND(&out, a, b);
  return 0;
}
void AND(int*out, int a, int b)
{
  *out = a & b;
}
```

When you declare the variable “out” as int, the compiler will allocate memory for it. You can then pass the pointer to “out” safely.
Please pass pointers to the output variable rather than declare the function as “unsigned short int AND(......)”. That way even if your circuit has more than one output, the structure of the program will remain the same, making it easy to debug.

Experiment 1

- Write C functions for all basic input gates: AND,OR,XOR,NOT,NAND,NOR,XNOR. These functions should be in the file gates.c
- Write functions to describe the following circuits using behavioural models. These should be in a file named circuit.c
  - $z = (a + c)(a + \bar{b})(\bar{a} + b + \bar{c})$
  - $z = (cd + \bar{b}c + b\bar{d})(b + d)$
  - $F(a, b, c, d) = \sum(2, 4, 7, 10, 12, 14)$
- Implement the same circuits using structural models.
- Write a testbench - should be in main.c. Your program should print the expected output and output obtained using your program.

To compile use

```
gcc main.c gates.c circuit.c -o lsim
```

- What changes would you have to make if you use “char” instead of “short int”. Use of “char” will save on memory.
- List out any computational inefficiencies (both in terms of operations and memory) that you can think of.