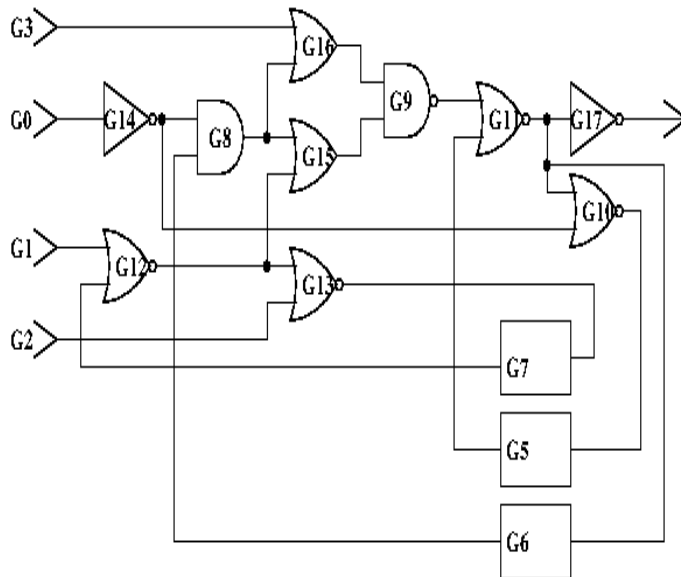


Part 1: Figure P1 shows the logic diagram and the corresponding description for a synchronous sequential circuit.



(a) Diagram

```

module main(G0,G1,G2,G3,G17);
input G0;
input G1;
input G2;
input G3;

output G17;
wire G5,G6,G7,G14,G8,G12,
      G15,G16,G13,G9,G11,G10;

dff1 XG1 (G5,G10);
dff1 XG2 (G6,G11);
dff1 XG3 (G7,G13);
not XG4 (G14,G0);
and XG5 (G8,G6,G14);
nor XG6 (G12,G7,G1);
or XG7 (G15,G8,G12);
or XG8 (G16,G8,G3);
nor XG9 (G13,G12,G2);
nand XG10 (G9,G15,G16);
nor XG11 (G11,G9,G5);
nor XG12 (G10,G11,G14);
not XG13 (G17,G11);

end
endmodule

```

(b) Circuit description

Figure P1: ISCAS s27 benchmark circuit

The ISCAS benchmark circuit description, shown in **Figure P1(b)**, consists of set of lines. Every line describes how one logic gate is interconnected with other gates. For example, line 'G5 = DFF(G10)' indicates the existence of a D-type Flip-flop with output G5 (connected to the state line q) and input G10 (connected to d). In this description INPUT (G1)/OUTPUT (G17); indicates that line G1 is a primary input/line G17 is a primary output.

a) Write a (C, C++ or JAVA) computer program that reads in the ISCAS circuit description as shown in **Figure P2**.

b) For every gate circuit, we associate a 'level'. The gate 'level' indicates the distance of that gate from primary inputs or pseudo inputs (D flip-flop Q's). Initially, the level of primary inputs and DFF flip-flops are set to zero. Gates 'level' for other gates are set to a negative value indicating uninitialized 'level'. Then, the a gate 'level', with assigned positive value on all of its inputs, is equal to the maximum 'level' of its inputs plus one. This step is repeated until every gate 'level' is assigned a positive number.

c) Your Part1 program should prints the following:

1. The total number of gates.
2. For every 'level' n print the number of gates assigned level n.
3. Print a listing of the final stored circuit using intermediate file format. The file consists of lines where each line represents a gate in the circuit. The file is shown below where each line is as follow:
 - i. Gatetype: is an integer representing the gatetype

Gate name
Gate type
List of pointers to fanin
List of pointer to fanout
Pointer to the next gate

Figure P2: Data record

GateType	Output	GateLevel	#faninN	fin1	fin2	...	finN	#fanoutM	fout1	fout2	...	foutM	GateName
----------	--------	-----------	---------	------	------	-----	------	----------	-------	-------	-----	-------	----------

- ii. A flag 1 if gate is output; 0 otherwise
- iii. #of fanin followed by a list of fanin ; each fanin is an integer representing the fan gate
- iv. #of fanout followed by a list of fanout ; each fanout is an integer representing the fanout gate
- v. Ascii gate name as it is define in ISCAS circuit file.

Part 2: Write a simulation program that reads the intermediate file generated in part 1 and stores the gates into an array of record. In each record, dynamically allocate integer arrays to store the fanin and fanout for each gate. Create an extra gate structure and store the index in variable 'dummy_gate'.

- A. Add to every gate structure an int 'sched' and set that pointer initially to -1. This pointer should be used to schedule the corresponding gate.
- B. Create an array 'levels' of pointers to a gate structure of size 'max level'. 'max_level' is an integer holding the value of max level in your design. Initially, levels[i] is set to dummy_gate for all i's.
- C. Use 3-valued logic {0, 1, X} to create two inputs lookup tables for the following gates AND, OR, XOR, and NOT. Gates with more than two inputs can be evaluated by more than one table lookup.
- D. To schedule events due to a change of the state of gate i:
 - a. For each gate f in the fanout list of gate i, if the field 'sched' of gate f is -1, then insert f at the head of the list at the corresponding level. Otherwise, no action is needed (schedule_fanout(gaten) in **Figure P1**).
- E. Implement the algorithm shown in Figure P1. In this algorithm, Flip-Flop do not need to be scheduled:

```

while() {
    print logic values at PI, PO, and States
    read inputs and schedule fanouts of changed inputs.
    load next state and schedule fanout.
    i = 0;
    while( i < max level) {
        gaten = levels[i];
        while( gaten != dummy_gate ) {
            newstate = evaluate( gate);
            if( new_state != gate.state ) {
                gate[gaten].state = new_state ;
                schedule_fanout( gaten );
            }
            tempn = gaten ;
            gaten = gate[gaten].sched ;
            gate[gaten].sched = 0 ;
            gaten = tempn;
        }
        levels[i] = gaten ;
        i = i + 1 ;
    }
}

```

Figure P1: Simulation Flow

Implement the 'evaluate' routine using the two techniques discussed in class:

- a. Input scanning:

Controlling value 'c' and inversion 'i'		
	c	i
AND	0	0
OR	1	0
NAND	0	1
NOR	1	1

```
evaluate(Gn){
    Uvalue = FALSE
    for(i=0; i<gate[Gn].nfanin;i=i+1){
        V = gate [ gate[Gn].fanin[i] ].state;
        if( V = c ) return( c  $\otimes$  i)
        if( V = X) Uvalue = TRUE;
    }
    if( Uvalue ) return X; else return c  $\otimes$  i;
}
```

- b. Table lookup:

```
evaluate(Gn){
    state_fanin0 = gate[ gate[Gn].fanin[0] ].state ;
    gate_type = gate[Gn].gtype;
    if gate_type == INV then return Inv_table[ state_fanin0 ] ;
    if gate_type == BUF then return state_fanin0 ;
    state_fanin1 = gate[ gate[Gn].fanin[1] ].state ;
    v = Table[gate_type][ state_fanin0 ][ state_fanin1 ]
    nfanin = gate[Gn].nfanin;
    for( i = 2; i < nfanin ; i++) {
        state_fanin0 = gate[ gate[Gn].fanin[i] ].state ;
        v = Table[gate[Gn].gate_type][ state_fanin0 ][ v ] ;
    }
    If( gate[Gn].i ) return (Inv_table[v]; else return ( v ) ;
}
```

- c. Record the CPU time your program requires to run both examples circuit posted on the website using the input scanning and table-lookup. Compare the two techniques.

Here is an input/output for s27

input file for s27: G0, G1, G2, G3

0000
0010
0100
1000
1111

output file: 4 represents undefined

INPUT :0000 // G0, G1, G2, G3
STATE :444 // G5, G6, G7
OUTPUT :4 // G17

INPUT :0010
STATE :044
OUTPUT :4

INPUT :0100
STATE :040
OUTPUT :4

INPUT :1000
STATE :041
OUTPUT :1

INPUT :1111
STATE :101
OUTPUT :1