CASE WESTERN RESERVE UNIVERSITY
ECES 318
HW 5

Fall 2014

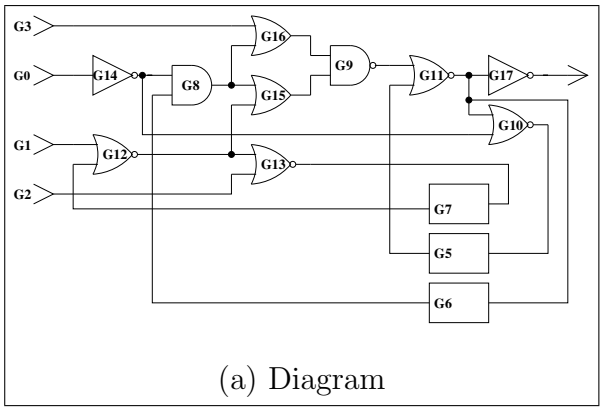
Due Dec 6, 2014

NAME:

Problem	Scale	Score
1	40	
2	60	
total	100	

Problem 1. [40 points]

Figure 1 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);

endmodule
```

(b) Circuit description

Figure 1: s27 verilog benchmark circuit

The verilog benchmark circuit description, shown in Figure 1(b), consists of set of lines. Every line describes how one logic gate is interconnected with other gates. For example, line 'dff1 XG1 (G5, 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 the line G1 is a primary input/(line G17 is a primary output).

In this part you are asked to write a (C or C++) computer program that reads in the verilog circuit description, adds buffers as needed and stores it in the data structure shown in Figure 2.
For every fanout branch add a buffer gate and modify your data structure to reflect the change. Figure 3 shows and AND gate with two fanout branches and the AND gate after adding the two buffers. (Add also buffers if needed for output nodes)
For every gate in the modified circuit associate a 'level'. The '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 outputs are set to zero and all other gates level is set to uninitialized.

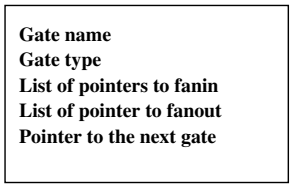


Figure 2: Data record

Figure 3: Fanout branches buffers

Then, the 'level' of a gate, with assigned level 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 number.
In this part, your 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.

Problem 2. [60 points]

(Part 2) In this part:

- use the data structure that you created in part1:
- Use 3-valued logic { 0, 1, X } to create a 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 two inputs table lookup.
- To schedule an event due a change on a line i do: (1) for each gete G on the fanout of line i, if G is not scheduled, then insert insert gate G in the schedule. Otherwise, no action is needed.
- Implement the algorithm shown in Figure 4. In this algorithm, Flip-Flop need not 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.
  E = list of scheduled gates;
  D = empty
  while( E is not empty) {
      for each gate g in E;
      E = E - {g} ;
      new_state = evaluate( gate);
      if( new_state != gate.state ) {
        gate.state = new_state ;
        D = D + schedule_fanout( gate );
    }
    E = D;
    D = empty;
  }
}
```

Figure 4: Simulation Flow

Implement the 'evaluate' routine using the two techniques discussed in class. Compare CPU time and memory requirement for the two techniques: (1) Input scanning and (2) table lookup.

(1) Input scanning: Controlling value 'c' and inversion 'i'

(2) Table lookup:

```
          | c  i
     -----+-----
      AND  | 0  0
      OR   | 1  0
      NAND | 0  1
      NOR  | 1  1

      evaluate(G,c,i)
         Uvalue = FALSE ;
         for every input value v of G
                if v == c then return (c xor i) ;
                if v == u then Uvalue = TRUE ;
         endfor
         if Uvalue return u; else return(cbar xor i)
      end

      evaluate(G)
           state_fanin0 = gate[G->fanin[0]]->state ;
           if gate_type == INV then return Inv_table[ state_fanin0 ] ;
           if gate_type == BUF then return state_fanin0 ;
           state_fanin1 = gate[G->fanin[1]]->state ;
           v = Table[gate_type][ state_fanin0 ][ state_fanin1 ]
           for( i = 2; i < number of fanin of G; i++) {
                  state_fanin0 = gate[G->fanin[i]]->state ;
                  v  = Table[gate_type][ state_fanin0 ][ v ] ;
           }
           return ( v ) ;
       end ;
```

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
```