Machine Project: Phase I

University of Toronto
ECE 1767: Design for Test and Testability
Spring Semester 2002

In this phase you will build a parser that reads ISCAS'85 combinational circuits and ISCAS'89 sequential circuits. This family of benchmark circuits can be found on the WWW page for this class. In more detail, the flow of things to be done during this phase is as follows:

Step 1: Read the circuit and create a Linked List (LL) data structure that holds the gates of the circuit. You will also create a hash table that holds the gates and write code for a routine that given the name of a gate it searches in the hash table for this gate and returns a pointer to it.

Step 2: Initialize the fan-in and fan-out lists for every entry (*i.e.* gate) of the LL.

Step 3: Insert buffers in branches.

1 Step 1

1.1 Reading ISCAS circuits

In this project we parse ISCAS benchmark (combinational and sequential) circuits with `AND, OR, NAND, NOR, NOT, BUFFER, INPUT, OUTPUT` and D flip-flop (memory elements) gates. Every distinct gate type should be associated with a *unique* gate number, *i.e.* 1 for `AND`, 2 for `OR` etc. Your project will not handle `XOR` gates. As you can see from the combinational suite of the ISCAS benchmark circuits, `XOR` gates have been commented out with the special character #. When you read a circuit, your program should ignore lines that begin with #.

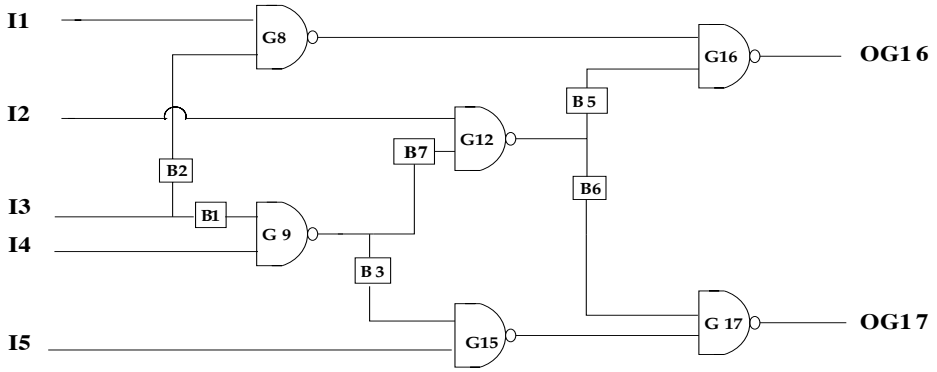

Figure 1: ISCAS'85 c17.bench combinational circuit

ISCAS circuits have a simple Verilog-like format where the statement `G17 = NAND(G12,G15)` indicates a `NAND` gate with name `G17` and fan-ins the output of the gates `G12` and `G15`. In our project, every gate will have a *unique* name. In addition, the primary `INPUT` and primary `OUTPUT` lines of the circuit are also considered as gates with distinct gate types and names. Since primary outputs, according to ISCAS

terminology, are not given a name, in your program you should give them a unique name such as the character "O" followed by the name of the gate that drives them. For example, ISCAS'85 benchmark circuit *c17.bench* is shown in Fig. 1 (for a moment, ignore the buffers, shown as small boxes, as they will be explained later).

Your first task is to write a code fragment that reads the ISCAS benchmark circuits and creates a linked list where every entry of this list is a unique gate structure. In detail, every gate should be represented by a structure similar to the one that follows:

```
typedef struct gate {
 char *gname;                 /* unique gate name */
 struct gate *nxt;            /* pointer to next gate in LL */
 struct list *fin, *fout;     /* explained in Step 3 */
 int gtype;                   /* unique gate type number */
 int level;                   /* circuit level, Phase II */
 int state;                   /* logic simulation, Phase II */
 int sdff;                    /* D flip-flop state, Phase III */
 int sa0, sa1;                /* stuck-at faults, Phase II-IV */
 struct gate *sched;          /* gate scheduling, Phase III-IV */
 struct gate *dfnxt;          /* PODEM's D-frontier, Phase IV */
 struct gate *lev;            /* circuit level, Phase II */
 struct gate *htable;         /* pointer to next hash table entry, see below */
 unsigned long int
    L0, L1, mask;             /* parallel fault simulation, Phase III */
 int D[2];                    /* PODEM's D-frontier, Phase IV */
} GATE;
```

1.2 Hash Table

Throughout the different phases of this project you will need efficient access to gate elements, *i.e.* given the name of a gate return a pointer to that gate in the LL. To do this you need to build a **hash table** and enter (hash) all gates, as you parse the benchmark circuit (subsection 1.1). A hash table is a type of data structure that allows efficient implementation of the "dictionary operations" INSERT, SEARCH, DELETE. It is out of the context of this class to explain how a hash table works. You can study about hash tables in any introductory combinatorics or data-structures textbook such as [1] [2].

Every time you insert a gate into the LL, you should also INSERT (*i.e.* hash the entry using a hash function) this gate into the appropriate bucket of the hash table (use the `htable` pointer). There are many ways to implement a *hash function*. One simple way to do it efficiently would be to *xor* the various characters that comprise the name of the gate (hash function input) and use this number as your hash entry bucket number (hash function output).

Finally, you should write the code for a function that given the name of a gate, it performs a SEARCH in the hash table and returns a pointer to the gate.

2 Step 2

In this step you should compile the list of fan-in and fan-outs link lists for every gate in the circuit and appropriately update the LL structure. This step can be performed efficiently with the use of the hash table.

Every fan-in and fan-out entry is represented by the structure similar to the one that follows:

```
typedef struct list {
 char *gname;                /* name of fan-in/fan-out gate */
 struct gate *ptr;           /* pointer to that gate in LL */
 struct list *nxt;           /* next element in fan-in/fan-out list */
} LIST;
```

For example, the `fin` pointer of the `gate` structure for gate `G17` will point to a linked-list with two `list` elements containing `G12` followed by `G15`. Equivalently, the `fout` pointer will point to a linked-list with a single `list` element, *i.e.* `OG17`.

3 Step 3

For this step you should insert a `BUFFER` at each line of a branch and give it a unique name of your own choice (possibly with prefix *BUF*). These buffers are shown as boxes in Fig. 1. In addition, you need to perform the work in sections 1 and 2 for each such buffer inserted.

4 Suggestions and Deadline

Read the description carefully. Draw a flowchart of the things you need to do and a high level description (pseudocode) of the functions you need to implement. See how you can write efficient code that can be re-used for the different requirements of this assignment. Post in the newsgroup if you have questions. Start immediately!

There is no particular deadline for Phase I but you will *need* to have Phase I completed before you start Phase II.

References

[1] A. V. Aho, J. .E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms, pp. 111–113*, Addison–Wesley, 1974.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms, pp. 219–232*, McGraw–Hill, 1990.