Homework: Implementing a BDD package

1 Introduction

In this assignment you will write a simple BDD package checker. The primary purpose of this assignment is to make you familiar with the concepts related to manipulating BDDs and implementing a BDD package that we covered in class, and also the data structures and exported functions in the VIS system.

1.1 Task Identification

You will be given a combinational netlist with several outputs; your task is to build the BDDs for the outputs in terms of the inputs, and print the BDDs graphically (be careful not to print subnodes repeatedly).

Generate the graphical view using the graphviz system from ATT:

`www.research.att.com/sw/tools/graphviz/refs.html`

Basically, you write the BDD as in `.dot` format, and run the `dot` program to generate pdf of a pictorial representation. The `dot` program does the placing of nodes and routing of edges for you. Here is a graph in `.dot` format:

```
digraph G {
size ="6,6";
        A -> B;
        B -> C;
        C -> D;
        D -> E;
        E -> A;
}
```

You create pdf from the graph by running

```
dot -Tps mygraph.dot -o mygraph.ps;
ps2pdf mygraph.ps
```

The construction of the output BDD should be in terms of the ITE operator, which you are required to implement. For efficiency, you are to keep a unique table and an ITE cache.

2 Implementation

Copy the directory `/home/projects/ece/verif/v-1.0/template` over to your work area. The makefile and templates are in this directory. In addition to the README, there is extensive documentation in the C template; please read through it carefully before you proceed.

To keep things simple, the only gates in the design will be inverters and 2-i/p AND gates. To determine the function of the gate, simply look at the number of inputs.

A BDD node is defined by the following:

```
typedef struct Pdd_Node_t pdd_t;

struct Pdd_Node_t {
  int id; /* lower id => closer to the top */
  pdd_t *lChild; /* leftChild */
  pdd_t *rChild; /* rightChild */
};
```

I have called these pdd_t's to avoid name conflicts with the existing BDD package.

We will not use complement pointers in this assignment, so to build the BDD B_f for an inverter, simply compute $ITE(B_g, 0, 1)$, where B_g is the BDD for the function of the fanin to the inverter.

Similarly, to build the BDD B_f for a 2-i/p AND gate f, simply compute $ITE(B_g, B_h, 0)$, where B_g and B_h are the BDDs for the two fanins of f. (Note that $ITE(B_g, B_h, 0) = ITE(B_h, B_g, 0)$, so you don't have to worry about the order of the fanins.)

There are two examples in the directory: *ex1* implements $p \cdot q \cdot r \cdot s$. *ex2* implements the function $a \cdot b + c \cdot d$;

Be sure you are using gnu make, and the gnu C compiler `gcc`. Both are available on LRC machines under the path `/usr/local/gnu/bin/`. It is easy to install VIS on your home machines (instructions for doing this are given under the VIS homepage).

3 References

Before starting, please read the VIS engineering manual, and the VIS programmer manual, available under the programmer documentation URL

`http://www-cad.eecs.berkeley.edu/~vis/prgDoc.html`

You will find the routines for hash table manipulation described in the `st` part of the generic routines documentation under at the programmer documentation URL useful. Brushing up on the basics of hashing will be helpful.

4 Deliverables

Write the output of your program on the files `ex*.bdd`. (Do not turn in code.)