

# EE382m: Homework 4

## Design Verification

Professor Adnan Aziz

`adnan@ece.utexas.edu`

Department of Electrical and Computer Engineering

The University of Texas

Due 4.5.04

1. Consider the sequential netlist  $\eta$  containing two latches whose next state logic is given by the following functions:

$$f_1(x_1, x_2, u_1) = x_1 \oplus x_2 \oplus u_1$$

$$f_2(x_1, x_2, u_1) = \bar{x}_1 \cdot \bar{x}_2 \cdot u_1$$

(here  $u_1$  is a primary input).

You are to compute the pre-image of the set  $A$  whose characteristic function is given by the function

$$f_A(x_1, x_2) = \bar{x}_1 \cdot \bar{x}_2$$

using

- (a) Algebraic expansion: write the Boolean functions for  $F_T$  the transition relation,  $F_T \cdot F_A$ , and grind through the existential quantification using  $(\exists\alpha)f = f_{\bar{\alpha}} + f_{\alpha}$ .

- (b) BDDs: represent functions using BDDs, and manipulate them using BDD operations. Use the variable ordering  $u_1 \prec x_2 \prec y_2 \prec x_1 \prec y_1$ .

Hint—the BDD calculations are quite tedious. You may find it easier to use the BDD package you developed in earlier homework to build the BDD's for  $F_T$  and  $F_A$  and to perform the pre-image computation operation using ITE and cofactor. You will need to add a routine to print a BDD, but that should be relatively straight-forward.

**15 marks**

2. Let  $\eta$  be a netlist. A *Strong fairness constraint* for  $\eta$  is a set of pair of subsets of  $\eta$ 's state space, i.e., of the form  $\{(A_0, B_0), (A_1, B_1), \dots, (A_{n-1}, B_{n-1})\}$ . An infinite sequence of states  $\sigma$  is defined to satisfy a strong fairness constraint if for each pair  $(A_i, B_i)$ , whenever the set of states occurring infinitely often in  $\sigma$  has a nonempty intersection with  $A_i$  then it has a nonempty intersection with  $B_i$ . (Mathematically,  $\sigma$  satisfies the constraint if  $(\forall i) (inf(\sigma) \cap A_i \neq \emptyset \rightarrow (inf(\sigma) \cap B_i \neq \emptyset))$  where  $inf(\sigma)$  is the set of states occurring infinitely often in  $\sigma$ .)

Give pseudo-code (similar to that given in class for model checking) for an explicit algorithm which takes a netlist  $\eta$  and a strong fairness constraint  $\{(A_0, B_0), (A_1, B_1), \dots, (A_{n-1}, B_{n-1})\}$  and returns the set of “fair” states, i.e., states for which there is an infinite input sequence for which the resulting path satisfies the strong fairness constraint.

**15 marks**

3. You are to design a mutual exclusion protocol. This protocol is structured in the form of a binary tree; the leaves are the processors, the internal nodes are the arbiter cells. The file `treeDME.v` contains a template in Verilog; specifically, the model for the processors, and the interconnects. Your job is to design the logic for the arbiter cell in such a way that the following two requirements are met:

- It is never the case that two processors are granted the resource at the same time.
- If a processor ever makes a request for the resource, it is eventually granted the resource.

The existing code for the processor and the interconnections should remain unchanged; you confine your work to the module *arbitCell*. The arbiter should be deterministic, i.e., the next state should be purely a function of the present state and the inputs from the children and parent.<sup>1</sup>

The design should be compiled to the *blif-mv* format using the *v12mv* compiler; VIS will read the generated *.mv* file using the *read\_blif\_mv* command; following this the *init* command initializes verification. Please **document** your design.

I have specified the properties as CTL formulas for you; these are given in the file *treeDME.ct1*; you can check them using the *model\_check* command.

The file *treeDME.v* contains a Verilog description of the processors, and the signalling mechanisms. In order to ensure that the processors do not remain in the state *lock* forever, you will need to read in the fairness constraint specified in *treeDME.fair* before performing model checking; this can be done using the *read\_fairness* command.

#### **Hints:**

- (a) It is convenient to view such protocols as passing a token, which defines which processor/cell has exclusive access to the resource. In this example, a processor holds the token if it is in state *lock*; you may find it convenient to have a one bit register in the arbiter cell which is 1 iff the cell holds the token.
- (b) Use the simulator in VIS in the initial phases of your design; this will catch the easy bugs much faster than formal verification. The simulator is invoked using the *sim* command.
- (c) You should start by reading the VIS user's manual, which describes Verilog, and also the commands you will need. The documentation hangs off the VIS home page: [www-cad.EECS.Berkeley.EDU/~vis/](http://www-cad.EECS.Berkeley.EDU/~vis/). Each command has online help which you can access with the *-h* option. The vis executable is  

```
/home/projects/ece/verif/vis-1.2/vis
```

---

<sup>1</sup>In essence, this is the tree-based arbitration scheme we've alluded to in class; you are to design the logic for the arbiter cell.

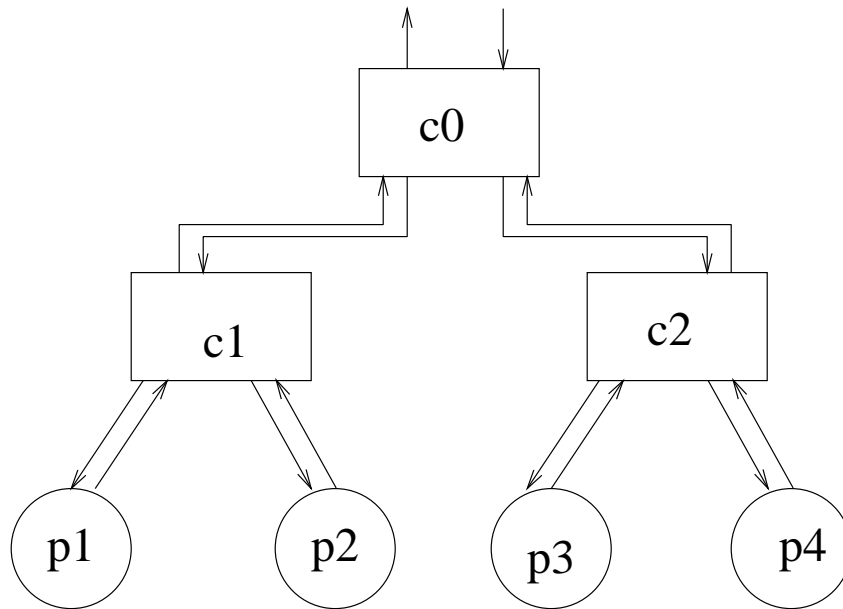


Figure 1: Tree based DME scheme.

The templates are located in the directory

`/home/projects/ece/verif/vis/arbit-template`

The `vl2mv` executable is in

`/home/projects/ece/verif/vis-1.2/vl2mv`.

**50 marks**