VLSI Testing and Design for Testability

Assignment4

B103526 中央電機黃聖倫

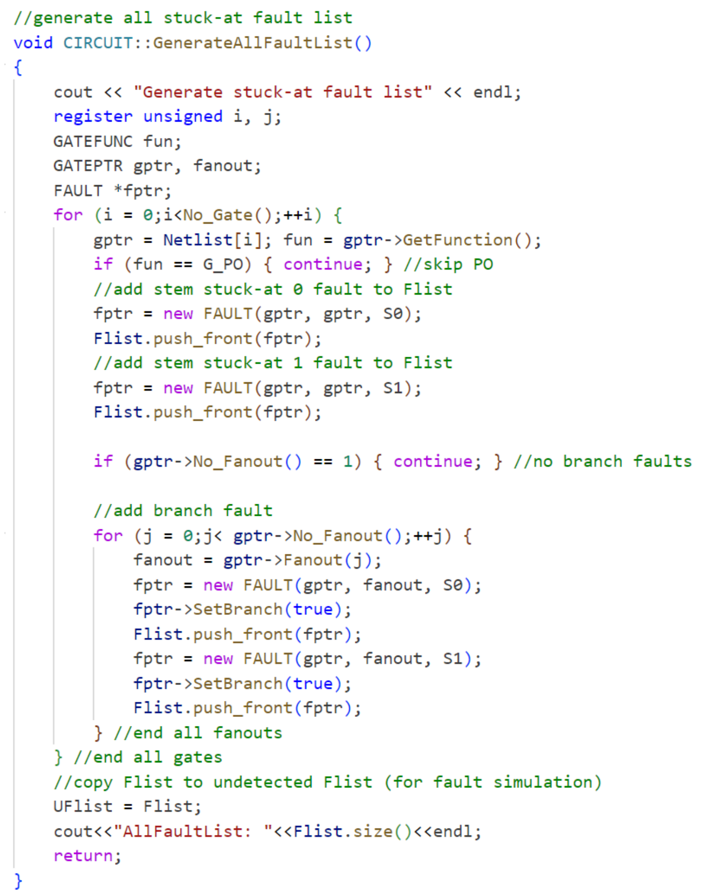
a)指令:

./atpg -check-point <絕對路徑+ [circuit_name]>

Method:

Checkpoint Theorm 是只考慮 Primary Input 和 fanout branch 兩個的 fault。

所以我通過使用 GenerateAllFaultList()的 Flist 得到 total 的 fault。



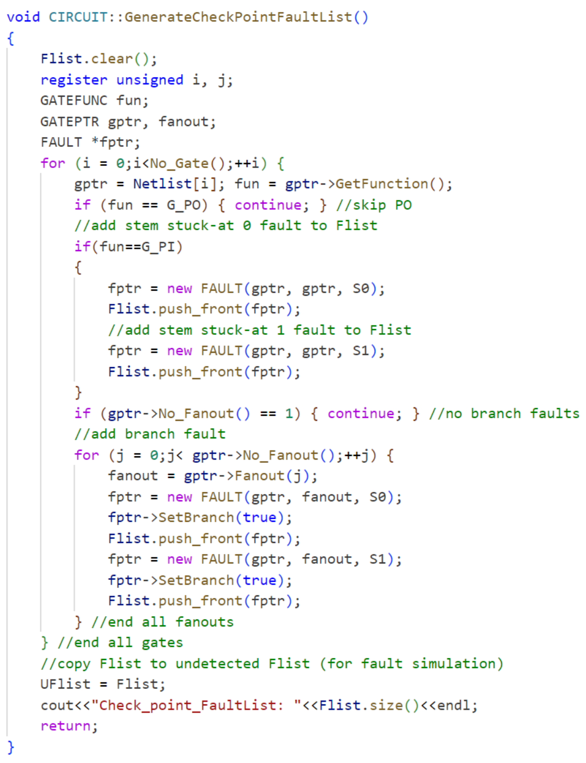
```
//generate all stuck-at fault list
void CIRCUIT::GenerateAllFaultList()
{
    cout << "Generate stuck-at fault list" << endl;
    register unsigned i, j;
    GATEFUNC fun;
    GATEPTR gptr, fanout;
    FAULT *fptr;
    for (i = 0;i<No_Gate();++i) {
        gptr = Netlist[i]; fun = gptr->GetFunction();
        if (fun == G_PO) { continue; } //skip PO
        //add stem stuck-at 0 fault to Flist
        fptr = new FAULT(gptr, gptr, S0);
        Flist.push_front(fptr);
        //add stem stuck-at 1 fault to Flist
        fptr = new FAULT(gptr, gptr, S1);
        Flist.push_front(fptr);

        if (gptr->No_Fanout() == 1) { continue; } //no branch faults

        //add branch fault
        for (j = 0;j< gptr->No_Fanout();++j) {
            fanout = gptr->Fanout(j);
            fptr = new FAULT(gptr, fanout, S0);
            fptr->SetBranch(true);
            Flist.push_front(fptr);
            fptr = new FAULT(gptr, fanout, S1);
            fptr->SetBranch(true);
            Flist.push_front(fptr);
        } //end all fanouts
    } //end all gates
    //copy Flist to undetected Flist (for fault simulation)
    UFlist = Flist;
    cout<<"AllFaultList: "<<Flist.size()<<endl;
    return;
}
```

圖一、Function GenerateAllFaultList()

再通過仿照 GenerateAllFaultList()寫出 GenerateCheckPointFaultList()去判斷只有 Primary Input 和 fanout branch 兩個的 fault。

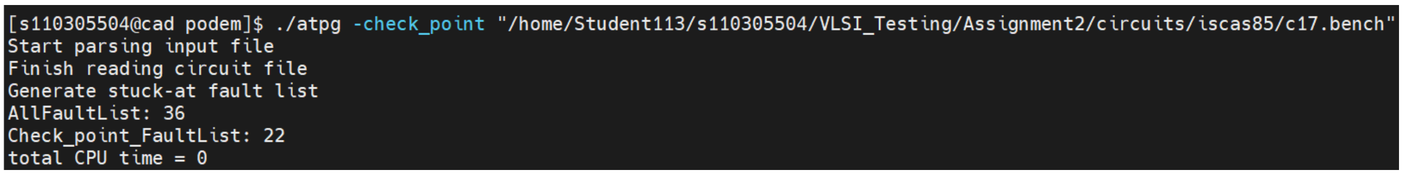
表一、ALL Faults 和 Faults by using Checkpoint Theorm 對比

Testbench	All Faults	Faults(CheckPoint)	% of faults have been collapsed
c17.bench	36	22	38.89%
c499.bench	2390	1282	46.36%
c7552.bench	19456	8098	58.38%



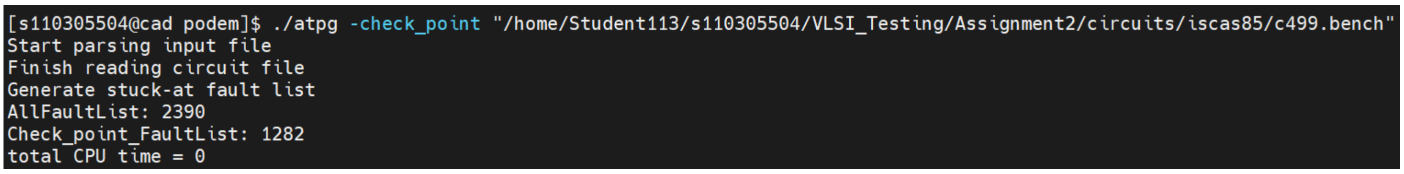
```
void CIRCUIT::GenerateCheckPointFaultList()
{
    Flist.clear();
    register unsigned i, j;
    GATEFUNC fun;
    GATEPTR gptr, fanout;
    FAULT *fptr;
    for (i = 0;i<No_Gate();++i) {
        gptr = Netlist[i]; fun = gptr->GetFunction();
        if (fun == G_PO) { continue; } //skip PO
        //add stem stuck-at 0 fault to Flist
        if(fun==G_PI)
        {
            fptr = new FAULT(gptr, gptr, S0);
            Flist.push_front(fptr);
            //add stem stuck-at 1 fault to Flist
            fptr = new FAULT(gptr, gptr, S1);
            Flist.push_front(fptr);
        }
        if (gptr->No_Fanout() == 1) { continue; } //no branch faults
        //add branch fault
        for (j = 0;j< gptr->No_Fanout();++j) {
            fanout = gptr->Fanout(j);
            fptr = new FAULT(gptr, fanout, S0);
            fptr->SetBranch(true);
            Flist.push_front(fptr);
            fptr = new FAULT(gptr, fanout, S1);
            fptr->SetBranch(true);
            Flist.push_front(fptr);
        } //end all fanouts
    } //end all gates
    //copy Flist to undetected Flist (for fault simulation)
    UFlist = Flist;
    cout<<"Check_point_FaultList: "<<Flist.size()<<endl;
    return;
}
```

圖二、Function GenerateAllFaultList()



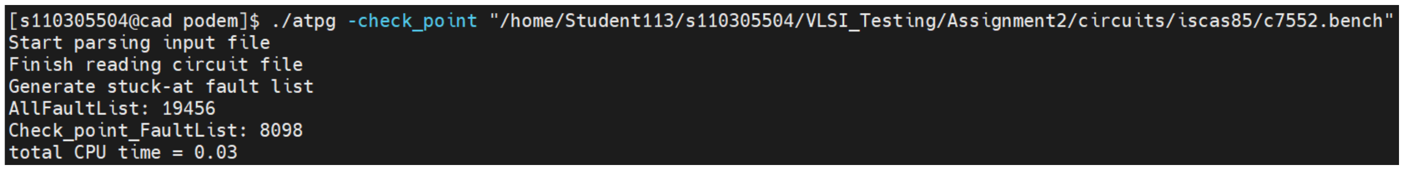
```
[s110305504@cad podem]$ ./atpg -check_point "/home/Student113/s110305504/VLSI_Testing/Assignment2/circuits/iscas85/c17.bench"
Start parsing input file
Finish reading circuit file
Generate stuck-at fault list
AllFaultList: 36
Check_point_FaultList: 22
total CPU time = 0
```

圖三、c17.bench 模擬結果



```
[s110305504@cad podem]$ ./atpg -check_point "/home/Student113/s110305504/VLSI_Testing/Assignment2/circuits/iscas85/c499.bench"
Start parsing input file
Finish reading circuit file
Generate stuck-at fault list
AllFaultList: 2390
Check_point_FaultList: 1282
total CPU time = 0
```

圖四、c499.bench 模擬結果

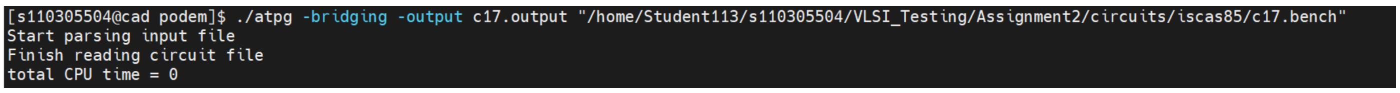


```
[s110305504@cad podem]$ ./atpg -check_point "/home/Student113/s110305504/VLSI_Testing/Assignment2/circuits/iscas85/c7552.bench"
Start parsing input file
Finish reading circuit file
Generate stuck-at fault list
AllFaultList: 19456
Check_point_FaultList: 8098
total CPU time = 0.03
```

圖五、c7552.bench 模擬結果

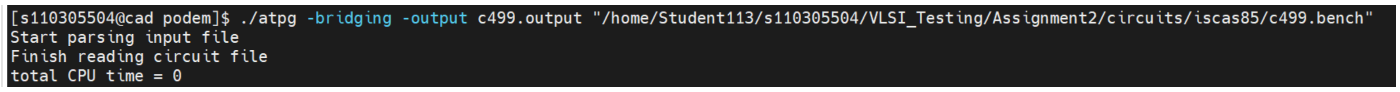
b)指令

./atpg -bridging -output [output_file_name] **<絕對路徑+ [circuit_name]>**



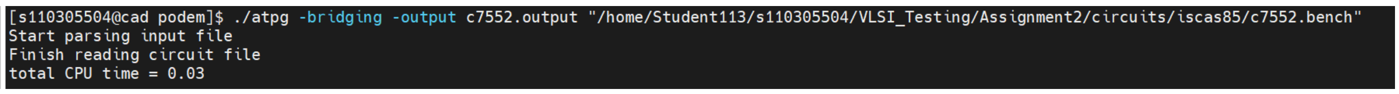
```
[s110305504@cad podem]$ ./atpg -bridging -output c17.output "/home/Student113/s110305504/VLSI_Testing/Assignment2/circuits/iscas85/c17.bench"
Start parsing input file
Finish reading circuit file
total CPU time = 0
```

圖六、c17.bench 模擬結果



```
[s110305504@cad podem]$ ./atpg -bridging -output c499.output "/home/Student113/s110305504/VLSI_Testing/Assignment2/circuits/iscas85/c499.bench"
Start parsing input file
Finish reading circuit file
total CPU time = 0
```

圖七、c499.bench 模擬結果



```
[s110305504@cad podem]$ ./atpg -bridging -output c7552.output "/home/Student113/s110305504/VLSI_Testing/Assignment2/circuits/iscas85/c7552.bench"
Start parsing input file
Finish reading circuit file
total CPU time = 0.03
```

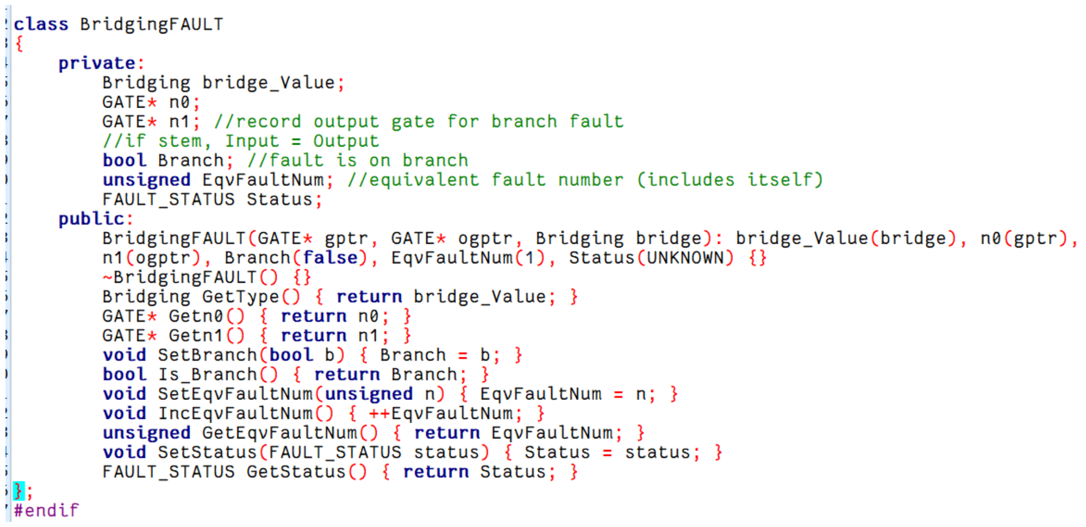
圖八、c7552.bench 模擬結果

表二、各個 case bridging faults 數量

Testcase	# of bridging faults
c17.bench	16
c499.bench	1140
c7552.bench	11642

表二的 c17.bench 有通過自己畫出所有 gate、PI、PO 來確認出 bridging faults 數量和結果一樣。

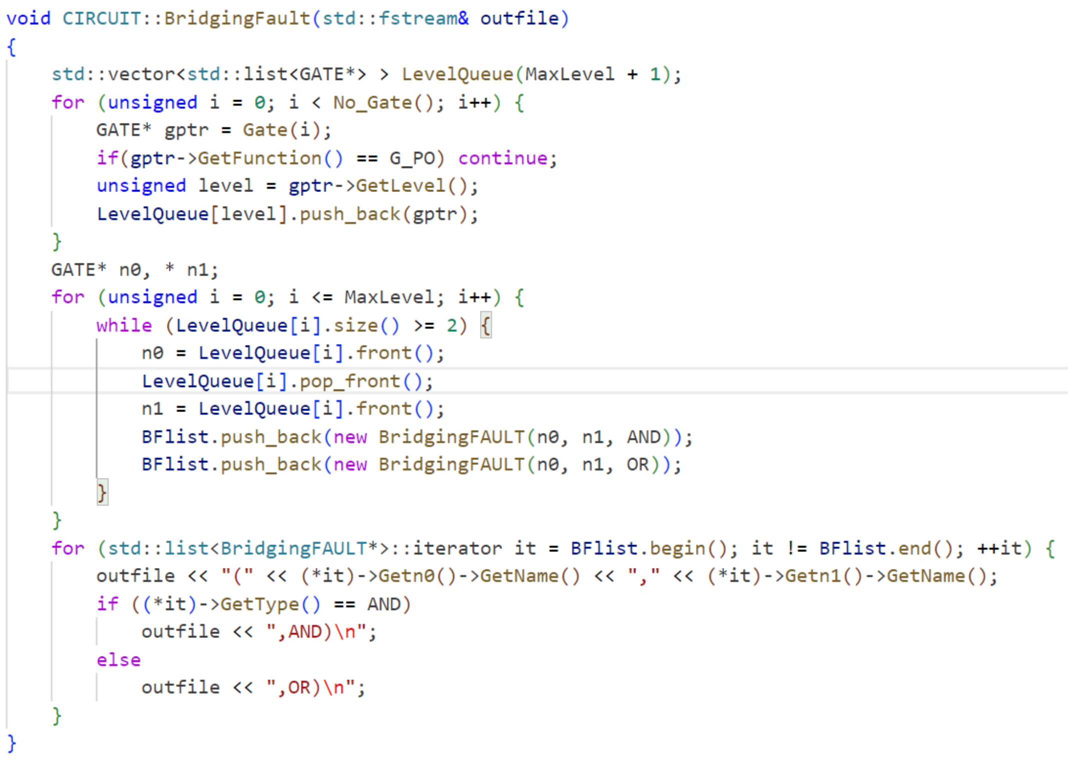
Method:



```
class BridgingFAULT
{
    private:
        Bridging bridge_Value;
        GATE* n0;
        GATE* n1; //record output gate for branch fault
        //if stem, Input = Output
        bool Branch; //fault is on branch
        unsigned EqvFaultNum; //equivalent fault number (includes itself)
        FAULT_STATUS Status;
    public:
        BridgingFAULT(GATE* gptr, GATE* ogptr, Bridging bridge): bridge_Value(bridge), n0(gptr),
        n1(ogptr), Branch(false), EqvFaultNum(1), Status(UNKNOWN) {}
        ~BridgingFAULT() {}
        Bridging GetType() { return bridge_Value; }
        GATE* Getn0() { return n0; }
        GATE* Getn1() { return n1; }
        void SetBranch(bool b) { Branch = b; }
        bool Is_Branch() { return Branch; }
        void SetEqvFaultNum(unsigned n) { EqvFaultNum = n; }
        void IncEqvFaultNum() { ++EqvFaultNum; }
        unsigned GetEqvFaultNum() { return EqvFaultNum; }
        void SetStatus(FAULT_STATUS status) { Status = status; }
        FAULT_STATUS GetStatus() { return Status; }
};
#endif
```

圖九、class Bridging Fault 定義

仿照 class fault 創建 Bridging Fault。



```
void CIRCUIT::BridgingFault(std::fstream& outfile)
{
    std::vector<std::list<GATE*> > LevelQueue(MaxLevel + 1);
    for (unsigned i = 0; i < No_Gate(); i++) {
        GATE* gptr = Gate(i);
        if(gptr->GetFunction() == G_PO) continue;
        unsigned level = gptr->GetLevel();
        LevelQueue[level].push_back(gptr);
    }
    GATE* n0, * n1;
    for (unsigned i = 0; i <= MaxLevel; i++) {
        while (LevelQueue[i].size() >= 2) {
            n0 = LevelQueue[i].front();
            LevelQueue[i].pop_front();
            n1 = LevelQueue[i].front();
            BFlist.push_back(new BridgingFAULT(n0, n1, AND));
            BFlist.push_back(new BridgingFAULT(n0, n1, OR));
        }
    }
    for (std::list<BridgingFAULT*>::iterator it = BFlist.begin(); it != BFlist.end(); ++it) {
        outfile << "(" << (*it)->Getn0()->GetName() << "," << (*it)->Getn1()->GetName();
        if ((*it)->GetType() == AND)
            outfile << ",AND)\n";
        else
            outfile << ",OR)\n";
    }
}
```

圖十、function BridgingFault

通過對所以 signal 進行判斷 level 為多少後加入 LevelQueue 中。

之後通過訪問相同 level 的 signal，並從 LevelQueue 中取出可得到相鄰並且同 level 的 signal。

Build:

make