ATALANTA 2.0 算法研究报告

10648052 任建国

1 算法研究

ATALANTA 2.0 主要由四种算法组成，包括 ATPG 生成算法 FAN，测试向量随机生成算法 RPT，组合电路故障模拟算法 FSIM 和时序电路故障模拟算法 HOPE。下面我们将简要阐述各种算法的基本原理。

1.1 总体算法结构

ATALANTA 2.0 的总体的算法结构如图 1.

1.2 随机向量生成（RPT）

RPT 利用初始种子生成初始向量，然后利用此向量进行故障模拟，如果检测到之前向量未检测到的故障，那么将此向量加入测试向量列表。其算法流程如图 2。

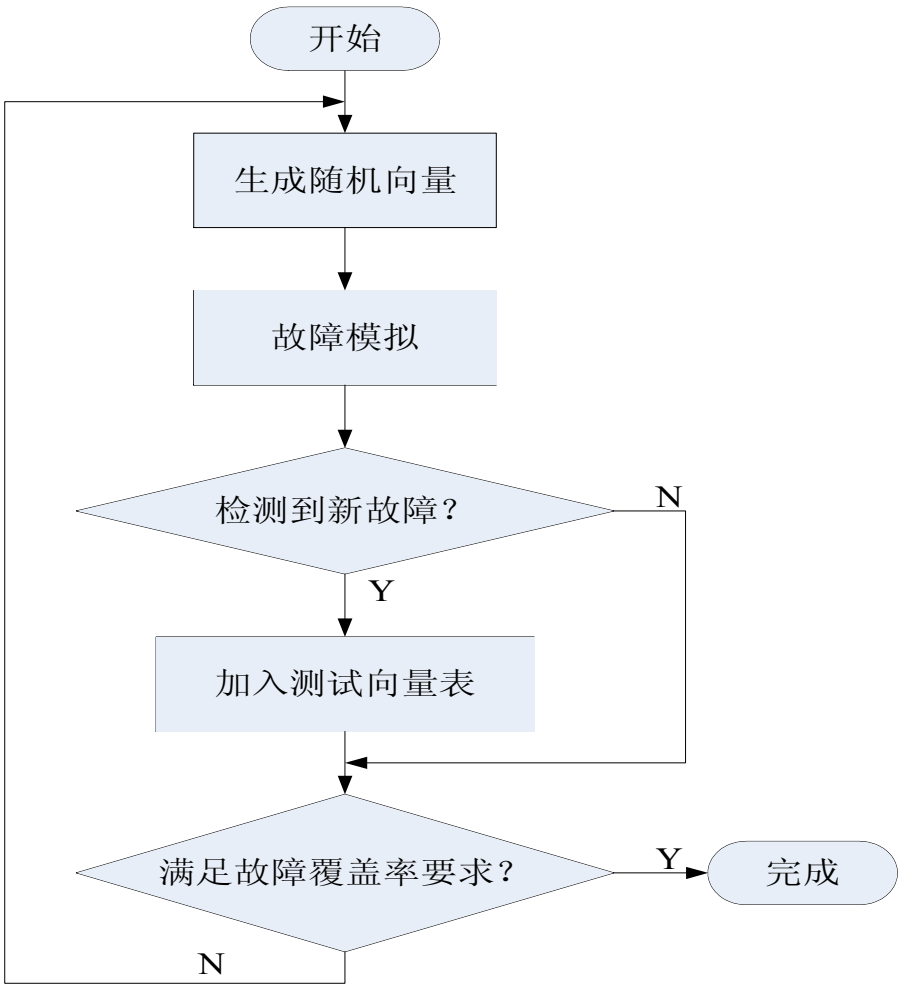


图 2 RPT 算法流程

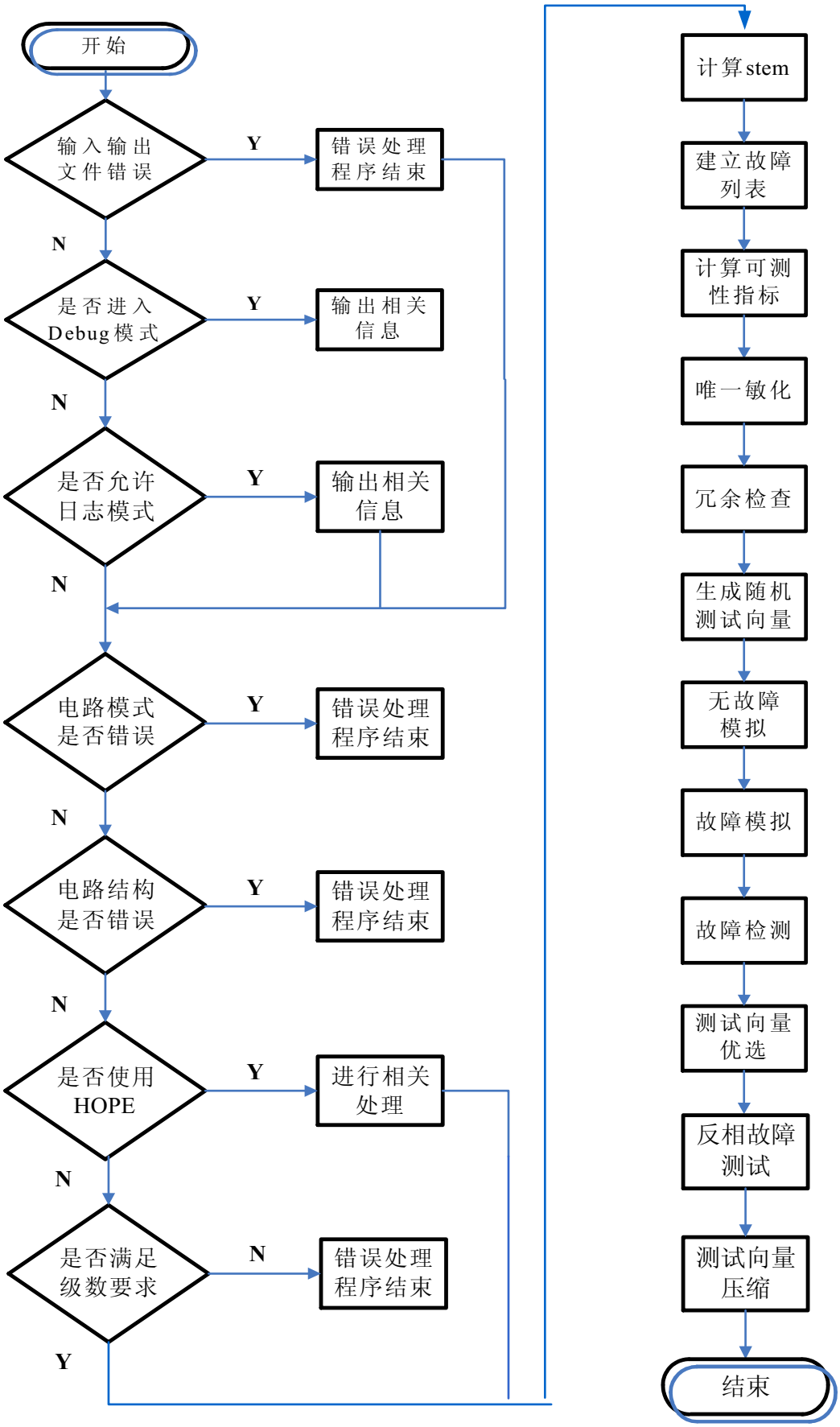


图 1 ATALANTA 算法总体流程

在 ATALANTA 2.0 中，如果连续 n 个随机测试向量无法检测到新的故障，那么 RPT 生成停止，其中 n 由使用者，其输入命令如下（默认情况，N=16）：

```
./atalanta -r n ****.bench
```

1.3 Fan 算法

FAN 算法是一种 ATPG 算法，采用分支限界的办法来进一步减小测试搜索空间，其流程如图 3。FAN 算法采用了几项新的策略来加速测试产生：

（1）充分蕴含。在算法执行的每一步尽可能多确定那些能被唯一决定的信号值，这样可以减少盲目选择，尽早发现矛盾。

（2）唯一敏化。当 D 前沿集合只包含一个门时，标记从该 D 前沿到原始输出的必经路段，然后赋予必要的值敏化这些路段。测试产生过程中的这种部分敏化称为唯一敏化。

（3）多重回溯。PODEM 算法所采用的多重回溯的策略有所不同，FAN 算法采用了广度优先的回溯方法，相比 PODEM 的深度优先回溯节省了大量时间。

（4）主导线。它们是指可将电路分块的点，使得通过切断成为主导线的单个连线可以将由 PI 驱动的逻辑锥与剩余的电路隔离开，从而消除 PI 判决树中的一个分支，而用其对于主导线的一个可能的选择值来代替。

在[1]中，关于 Fan 算法有详尽的阐述，在此不再赘述。

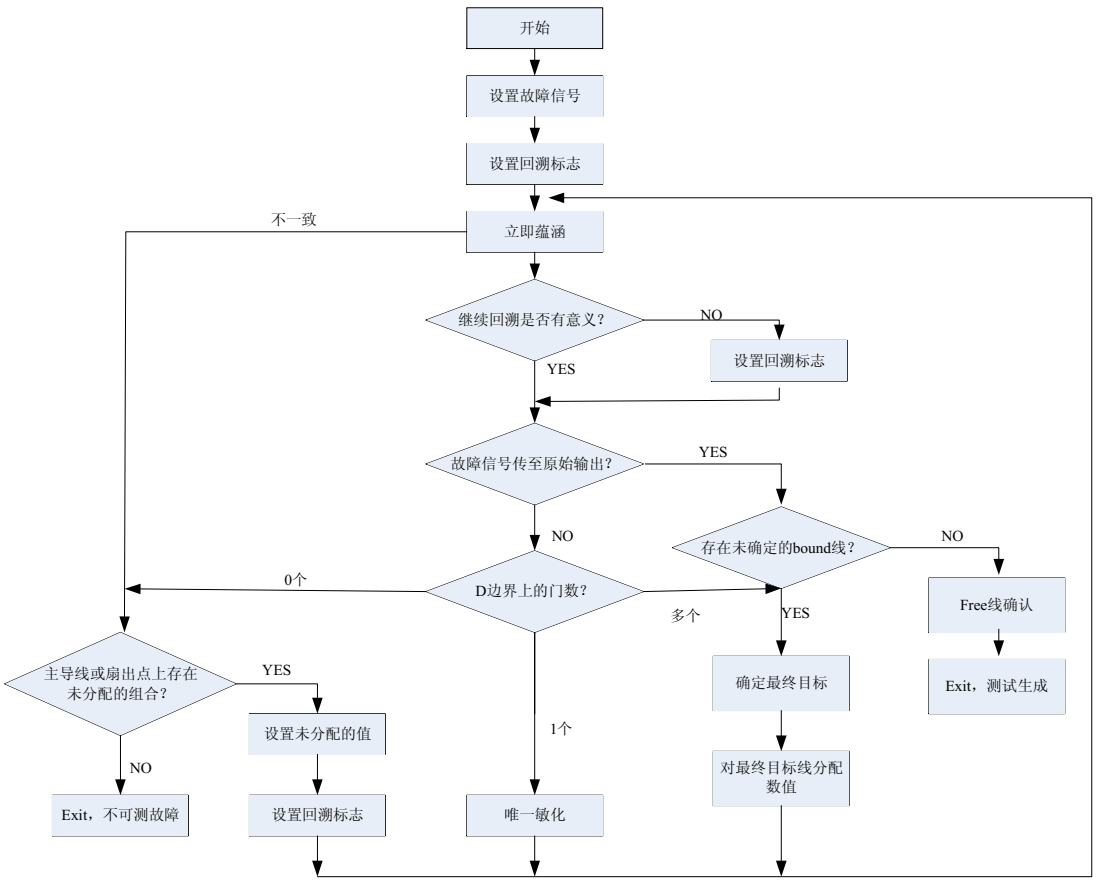


图 3 Fan 算法流程

1.4 Fsim 算法

相对于并行矢量单故障传播 PPSFP 算法，Fsim 算法仅仅模拟故障传播路径上的门，加快模拟速度，同时通过采用多个 LIFO 堆栈加快模拟速度。

Fsim 算法的核心内容包括以下几个方面：

1、 按前向顺序仿真电路，在较早的阶段去除不可测得区域。

如图4所示电路结构，通过预处理将电路划分为独立的fanout free regions（FFRs）。同时根据故障分布，以及支配节点分布，可以将FFRs分为三类：（1）包含故障的FFRs，如FFR（S1）；（2）包含至少一个支配节点但不包含故障的FFRs，如FFR（S5）；（3）既不包含故障也不包含支配节点的FFRs，如FFR（S2）。

对于第（1）类FFRs和故障传播路径上的第（2）类FFRs，需要进行故障模拟，而其他FFRs则不必进行故障模拟。我们通过S1，S3，S5，S6 FFRs解释如何判断需要模拟的第（2）类FFRs。

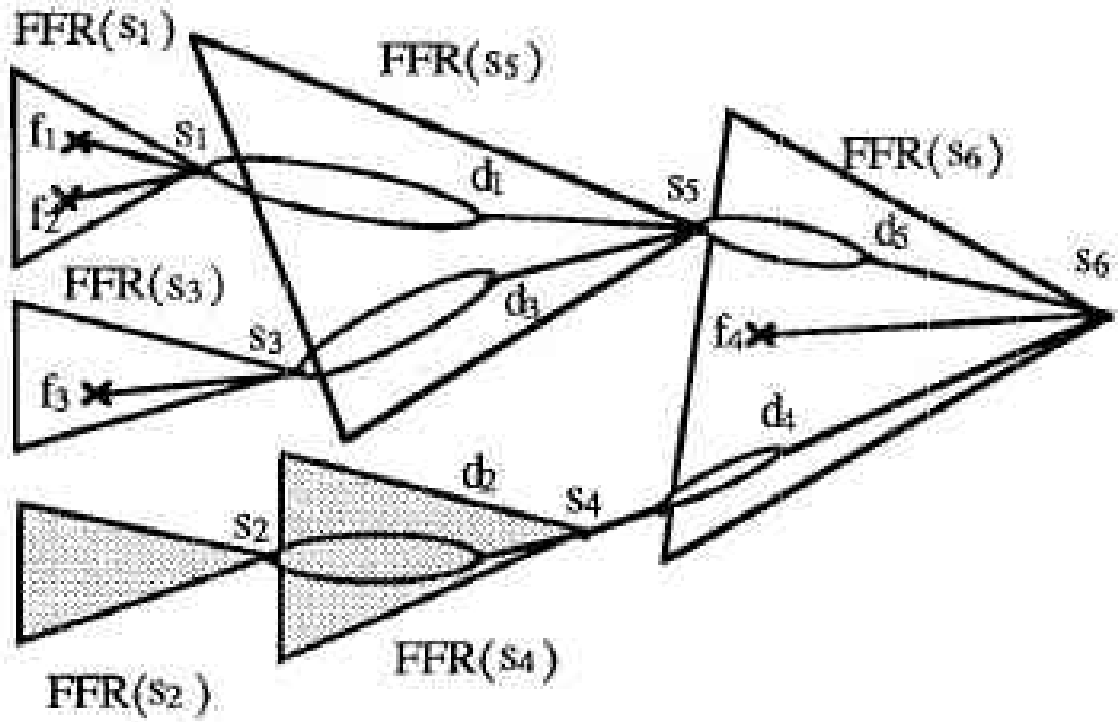


Figure 2. An Example Forward Order Simulation

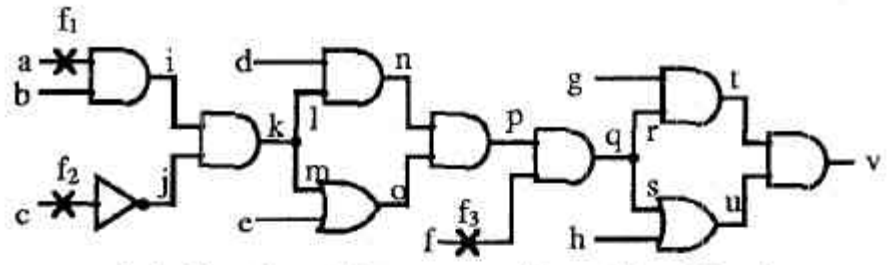
图 4 正向模拟电路举例

假设S1处可以检测到f1或f2中的某个故障，如果在支配节点d1可以检测到S1值，即D（S1，d1）=1，那么，d1处于故障的传播路径上，那么需要模拟FFR（S5），否则不必模拟。以此类推，进一步考察D（S5，d5）。

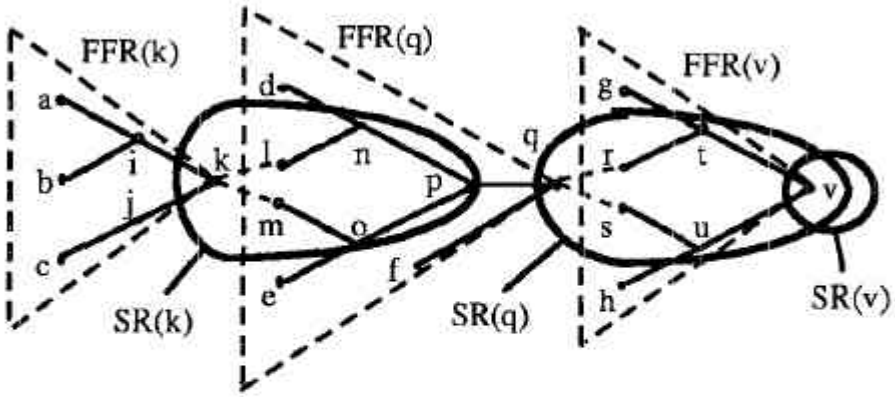
2、 钝化不必要的测试矢量。

并行模拟中的某些矢量无法检测到考虑的故障，如果这些故障能够尽早地被识别出来，那么仅仅这些矢量所影响FFRs就不必进行模拟，可以减少测试时间。

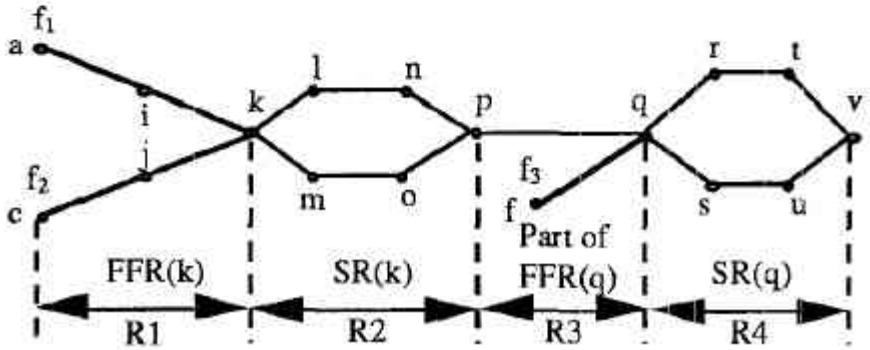
假设封装的矢量个数为 4 个，考虑的故障为图 5 中的 f1 和 f2，茎 k 和支配点 p，假设模拟结果如图 5，茎 k 处，f1 由第一个矢量检测到，而 f2 由第二个矢量检测到，因此，对于封装矢量 D（f1，k）=（1000），D（f2，k）=（0100）。但是由于故障未能传播至支配点



a) A Gate Level Representation of the Circuit



b) A Graph Model of the Circuit



c) A Simplified Graph Model of the Circuit

图 5 故障电路举例

无故障电路：V（k）=（0100），V（p）=（0000）
考虑故障 f1：V*（k）=（1100），V*（p）=（0000）
考虑故障 f2：V*（k）=（0000），V*（p）=（0000）

图 6 封装矢量故障模拟

p，因此这四个矢量无法检测到故障。

考虑固定故障，那么如果在 k 处能够检测到故障的要求是：V*（k）为 V（k）的反，即 V*（k）=（1011），假设 k 到原始输出的可观测性 D（k，PO）=（0001），即只有第四个矢量在 k 处检测到的故障能够传播至输出，但是实际上第四个向量没有将故障传播至 k，因此这四个封装矢量均无法检测到 f1 或 f2 故障，但却需要进行 R2，R3，R4 区域的故障模拟，

增加测试时间。

如果考虑到前两个矢量能够在 k 处敏化故障，而后两个不能，那么考虑 k 处故障时仅仅取前两个矢量的反，而后两个矢量保持不变，那么由于 D（k，PO）=（0001），即传播至原始输出的值与无故障电路相同，不必再进行模拟。

假设 CD（k）为封装矢量在 k 处的累积可观测性，那么 V*（k）=V（k）⊕CD（k），对于某个矢量，如果 CD（k）=0，那么其无法传播故障至原始输出处，此故障被钝化，仅仅与之相关的 FFRs 不再进行模拟。CD（k）的计算方法参考[2]。

3、 使用多个 LIFO 堆栈

原始的 LIFO 堆栈在寻找输入级最低的门时时间代价为 O（n*log（n）），Fsim 建议将同一级的门放在此级对应得 LIFO 堆栈中，将寻找时间代价减少到 O（n），其中 n 为故障传播域中门的个数。

Fsim 的流程如图 7。

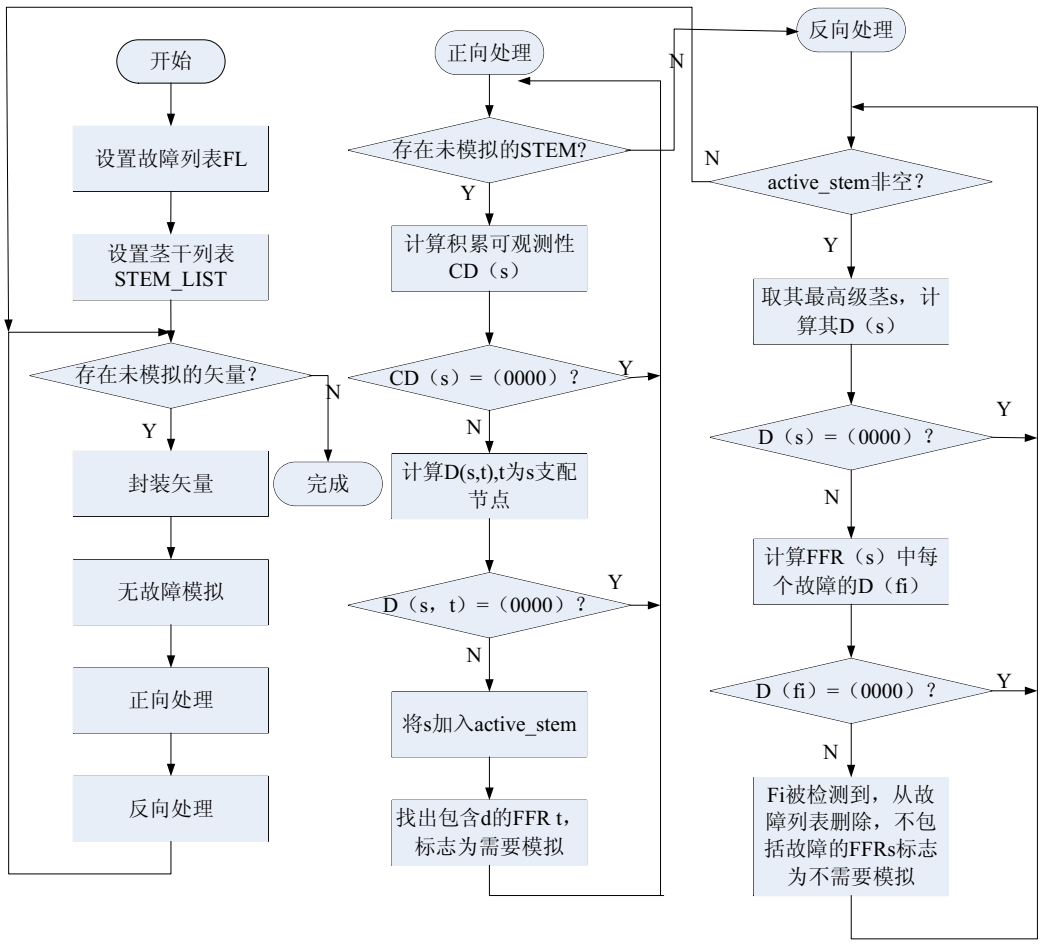


图 7，Fsim 算法流程

1.5 HOPE 算法

HOPE 算法是一个时序并行故障模拟器，相对于 PROOFS 故障模拟器，HOPE 在以下几个方面加快了故障模拟的速度[3]。

1、 减少需要并行模拟的故障的数目。

作者将故障分为两类：单事件故障和多事件故障。单事件故障其触发器输出（PPI）与无故障电路一致，而多事件故障，其触发器输出（PPI）与无故障电路不一致。考虑到大部分故障是单事件故障，作者通过以下两个策略来减少需要模拟的单事件故障，进而减少需要模拟的总的故障。在第一个阶段，在某个 FFR 中，所有的故障映射到 FFR 对应得茎故障。如图 8 所示。T 为 FFR（t）的支配节点，利用局部模拟可以将 FFR（t）中的故障可以映射到 t，有 3 种可能性，分别为 0，1 或 x，其中必定有值与无故障电路一致，因此 FFR（t）中的故障可以减少为茎 t 处的两种故障。在第二个阶段，将故障传至下一级支配节点或下一级门（如果没有支配节点）。如果故障可以传播，那么需要模拟，否则将故障从故障列表中摘除。

2、 减少故障注入时间。

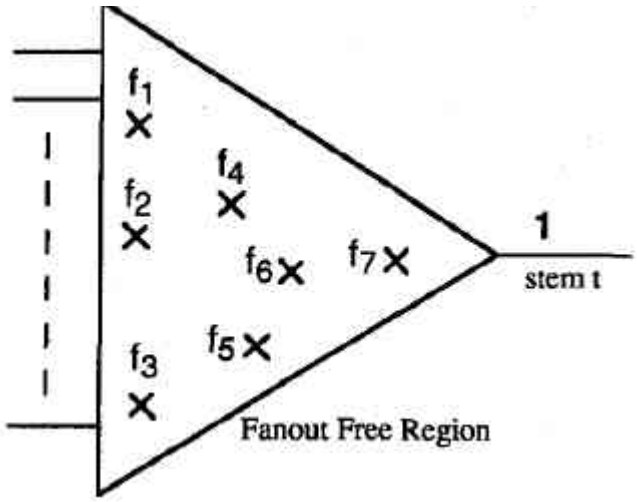


图 8 非茎故障映射

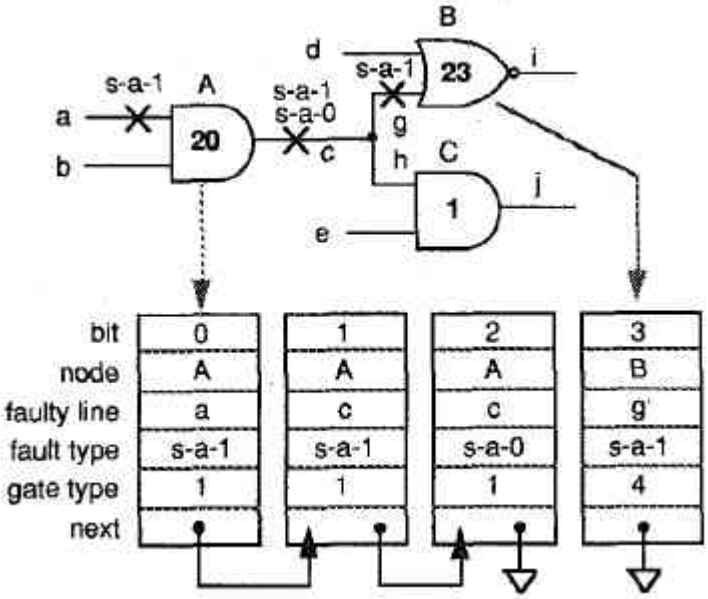


图 9 功能故障注入

在 PROOFS 中，对每个固定故障增加两个门，一个为注入故障门，另一个为单输入原始门，对于 SA1 故障，注入故障门为或门，单输入原始门为 1；对于 SA0 故障注入故障门为与门，单输入原始门为 0。需要修改电路，增加了需要模拟的门以及事件。

在 HOPE 中，作者将故障门等效为具有故障功能的门，其功能指针由 20 与其在故障矢量中的位置之和决定。如图 9 所示，假设 a/1，c/1，c/0，g/1 分别为 4 位故障矢量的 0，1，2，3 位，其描述分别如图，功能指针分别为 20，21，22，23，完成故障注入。

3、 故障分组方法

并行故障模拟每次进行多个故障的模拟时，故障矢量组的选择会影响到故障模拟的速度。排序的原则是尽量将产生相同事件的故障组合在一起模拟。在 HOPE 中作者采用静态和动态排序的方法完成故障组合。

A、静态故障分组。考虑图 10 的电路，在 PROOFS 中，作者采用深度优先搜索方法，其故

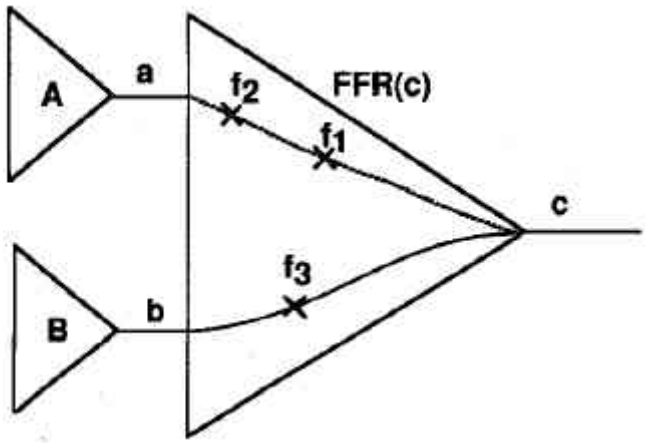


图 10 静态故障分组

障排序为：f1，f2，fA，f3，fB，如果 fA 足够多，那么 f1，f2，f3 有可能不在同一个故障矢量里面，但是由于它们都经茎 c 那么它们产生相同事件的概率很大。考虑到这个特点，HOPE 采用广度优先的搜索方法，其故障排序方式为：f1，f2，f3，fA，fB，使 f1，f2，f3 这三个在同一个 FFR 中的故障分布在同一个故障矢量中的概率大大增加。

B、动态故障分组。由于故障的存在，可能阻止触发器触发至特定的值，产生 X 态，对于潜在可测试性故障，必然存在从触发器至输出的传输途径。即潜在可测试性故障具有更高的活性。将高活性的故障分为一组，各个事件传播路径重复的概率增大，可以减小模拟时间。作者将故障分为两组，A 组不包括潜在可检测故障，B 组包括潜在可检测故障。

2、部分代码注释

2.1、结构体定义

```
typedef struct HASH {
    int key;                        /* variable of the given symbol */
    GATEPTR pnode;
    struct HASH *next;              /* next string */
    char *symbol;                    /* symbolic name */
} HASHTYPE;

/* gate data structure */
typedef struct GATE {
      int index;                    /*  门的序号*/
      logic fn;         /* type of gate */
      short ninput;         /* number of fanins */
      struct GATE **inlis;      /* fan-in list */
      short noutput;        /* number of fan-outs */
      struct GATE **outlis;     /* fan-out list */
      short dpi, dpo;       /* depth from PIs & POs */
      status changed;           /* flag for event queue operations*/
      HASHPTR symbol;           /* pointer to the symbol table */
```

```
    boolean freach1,            /* is reached from faulty gate */
        freach;         /* is reached from faulty gate */
    status xpath;       /* x path exists or not */
    line_type ltype;    /* head line, free line, bound line */
    int numzero,numone;     /* expected number of zeros and ones */
    LINKPTR u_path;     /* unique path */
    LINKPTR u_path1;/* unique path */
    int cont0,cont1;    /* 0 and 1 controllability */
    int fos;        /* fanout stem indication */
    int nfault;
    level cobserve,observe;  /* cumulated and local detectabilities */
    FAULTPTR pfault,*dfault;/* fault list */
#ifdef LEARNFLG
    struct LEARN *plearn;
#endif
    GATEPTR next;               /*作用是将门按照输入顺序反序连接起来*/
#ifdef ISCAS85_NETLIST_MODE
    int gid;        /* gate identification */
#endif
#ifdef INCLUDE_HOPE
        level GV[2];            /* Good value; (V0,V1) */
        level FV[2],Gid;        /* Faulty Value; (V0,V1,Gid) */
        int stem;               /* indication of the stems */
        status sense;           /* simulated or not --- version 3 */
        level SGV;              /* Good Value for SPF */
#else
    level output;       /* output value for fan */
    level output1;      /* output value for fsim */
#endif
} GATETYPE;

/* fault list data structure */
typedef struct FAULT {
    struct GATE *gate; /* faulty gate */
    int line;       /* faulty line, -1 if output fault */
    fault_type type;    /* fault type */
    boolean detected;   /* detected or not */
    level observe;      /* detectability */
    struct FAULT *next;     /* pointer to the next fault */
    struct FAULT *previous;/* pointer to the previous fault */
#ifdef INCLUDE_HOPE
    int npot;       /* # potentially detected --- HOPE */
    EVENTPTR event;     /* event list --- HOPE */
#endif
```

```
} FAULTTYPE;

typedef struct link {
        GATEPTR ngate;
        struct link *next;
} LINKTYPE;

typedef struct LEARN {                          /* for learning */
                struct LEARN *next;
                int node;
                char sval, tval;
} LEARNTYPE;

typedef struct EDEN {                           /* constant values */
                struct EDEN *next;
                int node;
                char val;
} EDENTYPE;

typedef struct STACK {                  /* LIFO stack */
        int last;
        struct GATE **list;
} STACKTYPE,*STACKPTR;

/* Decision tree data structure */
typedef struct TREENODE {
        GATEPTR gate;               /* pointer to the gate */
        boolean flag;           /* flag for backtracking */
        int pstack;             /* pointer for implication */
} TREETYPE,*TREEPTR;

struct ROOTTREE {                                       /* tree for backtracking */
                int last;
                TREEPTR list;
};

struct FLIST {                              /* fault list */
                int last;
                FAULTPTR *list;
};

#ifdef INCLUDE_HOPE
typedef struct EVENT {          /* for HOPE */
                int node;                       /* event node */
```

```
        level value;                    /* v0 & v1 */
        struct EVENT *next;                     /* pointer to next event */
} EVENTTYPE;

typedef struct STEM *STEMPTR;
typedef struct STEM {
        int gate;
        int dominator;
        int checkup;
        FAULTPTR fault[3];
        short flag[3];
} STEMTYPE
```

2.2、读取电路网表

```
/*------getsymbol-----------------------------------------------------
    reads next symbol from the given circuit file
    inputs : file     the name of input file
    outputs: s symbol string output
            returns delimiter following symbol
            =     gate name
            (     gate type
            ,     input signal --- will be continued
            )     input --- end of input
-------------------------------------------------------------------*/
char getsymbol(file,s)//file 输入文件，s 记录隔离符前内容，函数返回隔离符号
FILE *file; register char *s;
{
    register char c;
    int comm=0;

    while((c=getc(file))!=EOF) {
        if(is_comment(c)) { comm=1; continue; }
        if(comm==1) {
            if(c=='\n') comm=0;
            continue;
        } //略过注释行
        if(is_white_space(c)) continue;
        if(is_delimiter(c)) break;
        *s++=c;
    }
    *s=EOS;
    return(c);
}
```

```
/*------gatetype: returns type of the gate symbol------*/
int gatetype(symbol)//函数返回门类型
char *symbol;
{
    int fn;

    if(strcmp(symbol,"NOT") == 0) fn=NOT;
    else if(strcmp(symbol,"AND") == 0) fn=AND;
    else if(strcmp(symbol,"NAND") == 0) fn=NAND;
    else if(strcmp(symbol,"OR")== 0) fn=OR;
    else if(strcmp(symbol,"NOR") == 0) fn=NOR;
    else if(strcmp(symbol,"DFF") == 0) fn=DFF;
    else if(strcmp(symbol,"XOR") == 0) fn=XOR;
    else if(strcmp(symbol,"XNOR") == 0) fn=XNOR;
    else if(strcmp(symbol,"BUFF") == 0) fn=BUFF;
    else if(strcmp(symbol,"BUF") == 0) fn=BUFF;
    else if(strcmp(symbol,"INPUT") == 0) fn=PI;
    else if(strcmp(symbol,"OUTPUT") == 0) fn=PO;
    else if(strcmp(symbol,"not") == 0) fn=NOT;
    else if(strcmp(symbol,"and") == 0) fn=AND;
    else if(strcmp(symbol,"nand") == 0) fn=NAND;
    else if(strcmp(symbol,"or")== 0) fn=OR;
    else if(strcmp(symbol,"nor") == 0) fn=NOR;
    else if(strcmp(symbol,"dff") == 0) fn=DFF;
    else if(strcmp(symbol,"xor") == 0) fn=XOR;
    else if(strcmp(symbol,"xnor") == 0) fn=XNOR;
    else if(strcmp(symbol,"buff") == 0) fn=BUFF;
    else if(strcmp(symbol,"buf") == 0) fn=BUFF;
    else if(strcmp(symbol,"input") == 0) fn=PI;
    else if(strcmp(symbol,"output") == 0) fn=PO;
    else fn=(-1);

    return(fn);
}

char *spc_to_und(buf)//将空格转化为下划线？
char *buf;
{
    int i=0;
    while (buf[i] == ' ') buf++;

    while (buf[i] != '\0') {
       if (buf[i] == ' ') buf[i] = '_';
       i++;
```

```
    }
    return (buf);
}

#define get_string(file,string) \
if(fscanf(file,"%s",string)<GOOD) fatalerror(CIRCUITERROR)

/*------read_circuit-----------------------------------------------
            Reads a circuit file.
      The netlist format is the same as that of ISCAS89 benchmark
      netlists except that:

                1) The first line should start with # followed by
               the name of the circuit, and
                2) Comment lines which starts with # can be inserted
               in any part of the circuit netlist.

      Constructs basic data stuructures
            and allocates necessary memory spaces.

      Users should provide the following information;
      1. index field : identification of gates (same as array index)
      2. fn          : gate type
      3. ninput      : number of fan-in lines
      4. inlis       : list of fan-in lines
      5. noutput     : number of fan-out lines
      6. outlis      : list of fan-out lines
      7. name        : pointer to the hash table

      inputs: circuit input file (circuit)
      outputs: data structures

      Note: The circuit description is topologically sorted  in the
            levelized order after parsing.
      Laveling order: primary input starts 0 to n-1   (n inputs)
                flip/flops starts n to n+m-1    (m f/f)
                gate starts n+m to n+m+l-1      (l gates)
-----------------------------------------------------------------------*/
int read_circuit(circuit,name)
FILE *circuit; char name[];
{
    register int i,j;
    register char c;
    register GATEPTR cg;
```

```
    register HASHPTR hp;
    int int_nog,int_nopi,int_nopo,int_noff;
    char symbol[MAXSTRING];
    char buf[MAXSTRING], *bufptr;    /* store a line of text */
    int fn, nofanin;//number of fanin
    int net_size;
    GATETYPE *pg;
    GATEPTR pfanin[MAXFIN+100];
    GATEPTR po_gates[MAXPO+100];
    int     nerrs = 0;

    /* The first line should start with # followed by
        the name of the circuit */
    while((c=getc(circuit)) != EOF)
        if(c=='#' || c=='\n') break;
    if(c==EOF) return((-1));
    if(c=='#') {
       if (!fgets(buf, MAXSTRING, circuit))
            fatalerror(CIRCUITERROR);
       buf[strlen(buf)-1] = '\0';
       bufptr = spc_to_und(buf);
       strcpy(name, bufptr);
                    }//提取电路名称

    InitHash(symbol_tbl,HASHSIZE);//初始化 hash 结构

    /* Pass 1:
      Adds the gate symbols to symbol_tbl[] and
      counts # of gates, pi's, po's, ff's
    */
    int_nopi=int_nopo=int_noff=int_nog=0;
    nofanin=0;
    begnet=(GATEPTR)NULL;

/*
 *  下面一段 while 代码，确定 define.c 中 GATE struct 中的
 * index fn ninput inlis noutput outlis next
*/

    while((c=getsymbol(circuit,symbol)) != EOF) {
        switch(c) {
        case '=' :      /* a new gate */
            hp=Find_and_Insert_Hash(symbol_tbl,HASHSIZE,symbol,0);
            if((cg=hp->pnode) == NULL) {
```

```
            ALLOCATE(cg, GATETYPE, 1);
            hp->pnode = cg;
            cg->symbol = hp;
            cg->next = begnet; //next 的作用是将门按照输入顺序反序连接起来
            begnet = cg;
        }
break;
  case '(' :                    /* gate type */
        if((fn=gatetype(symbol)) < 0) {
            fprintf(stderr,"Error: Gate type %s is not valid\n",symbol);
            return(-1);
        }
        break;
  case ',':                                 /* fanin list */
        hp=Find_and_Insert_Hash(symbol_tbl,HASHSIZE,symbol,0);
        if((pg=hp->pnode) == NULL) {
            ALLOCATE(pg, GATETYPE, 1);
            hp->pnode = pg;
            pg->symbol = hp;
            pg->index = (-1);//对应的门排序数尚未确定
            pg->next = begnet;
            begnet = pg;
        }
        pfanin[nofanin++]=pg;//扇入数增加,门扇入数组，即扇入连接的门
        break;
  case ')':                     /* terminator, fanin list */
        hp=Find_and_Insert_Hash(symbol_tbl,HASHSIZE,symbol,0);
        if((pg=hp->pnode) == NULL) {
            ALLOCATE(pg, GATETYPE, 1);
            hp->pnode = pg;
            pg->symbol = hp;
            pg->index = (-1);//对应的门排序数尚未确定
            pg->next = begnet;
            begnet = pg;
        }
switch(fn) {
case PI:
            int_nopi++;//初始输入加 1
            pg->index = int_nog++;
            pg->ninput=0; //初始输入的扇入端数目为 0
            pg->inlis=(GATEPTR *)NULL;
            pg->fn=PI;      //门类型为初始输入
            pg->noutput=0;
            pg->outlis=(GATEPTR *)NULL;
```

```
                break;
        case PO:
            po_gates[int_nopo++] = pg;//输出门数组
            break;
        default:
            pfanin[nofanin++] = pg;//扇入数增加,门扇入数组，即扇入连接的门
                switch(fn) {
                    case DFF: int_noff++; break;//flip-flop 数目加 1
                    case XOR: case XNOR:
                        if(nofanin!=2) {//如果 xor 或 xnor 的输入数不为 2，系统不能处理
                            fprintf(stderr,"Error: %d-input %s gate is not supported\n",
                                    nofanin, fn_to_string[fn]);
                            return (-1);
                        }
                }
                if(cg==NULL) {
                    fprintf(stderr,"Error: Syntax error in the circuit file\n");
                    return(-1);
                }
                cg->index = int_nog++;//定义此门为得到的第几个门，门数增加，其中初始输
入也包括在 int_nog 中
                cg->fn=fn;//门类型
                if((cg->ninput=nofanin) == 0) cg->inlis=NULL;//门没有扇入
                else { ALLOCATE(cg->inlis,GATEPTR,cg->ninput); }//分配扇入门存储空间
                for(i=0; i<nofanin; i++) cg->inlis[i]=pfanin[i];//确定扇入门列表
                cg->noutput=0;//门扇出目前尚未确定
                cg->outlis=(GATEPTR *)NULL;

                nofanin=0;
                cg = (GATEPTR) NULL;
            break;
             }
         }
    }

    /* Pass 2: Construct the circuit data structure */
    net_size=int_nog+int_nopo+int_noff+SPAREGATES;//网表大小
    ALLOCATE(net,GATEPTR,net_size);
    ALLOCATE(primaryin,int,int_nopi);//初始输入列表
    ALLOCATE(primaryout,int,int_nopo);//初始输出列表
    ALLOCATE(flip_flops,int,int_noff);//触发器列表
#ifdef ATALANTA
    ALLOCATE(headlines,int,int_nopi);
#endif
```

```
    nog=nopi=nopo=noff=0;
    for(cg=begnet; cg!=NULL; cg=cg->next) {
        if(cg->index < 0) {
            fprintf(stderr,"Error: floating net %s\n",cg->symbol->symbol);
            fprintf(stderr, "Workaround. You have to take one of the two actions:\n");
            fprintf(stderr, "      1. Remove all the floating input and associated gates, or\n");
            fprintf(stderr, "      2. Make each floating input a primary output.\n");
            return(-1);
        }
        net[cg->index] = cg;//将 net[i]与门的 index 一一对应起来
        nog++;
    }
    for(i=nog; i<net_size; i++) net[i]=(GATEPTR)NULL;

    if(nog != int_nog) {//每个输入门（包括原始输入）唯一对应一个 index,如果不等，在读的过程中发生错误
        fprintf(stderr,"Error in read_circuit\n");
        return(-1);
    }

    /* Pass 3: Compute fanout list */

/*
 *   以下一段 for 代码确定 GATE struct 中的 noutput  和 plearn
 * 以及确定本程序需要计算的 primaryin，flip_flop 数组
*/
    for(i=0; i<nog; i++) {
        cg=net[i];
#ifdef LEARNFLG
        cg->plearn=NULL;
#endif
        for(j=0;j<cg->ninput;j++) cg->inlis[j]->noutput++;
        switch(cg->fn) {
        case PI: primaryin[nopi++] = i; break;
        case DFF: flip_flops[noff++] = i; break;
        }
    }
    for(i=0; i<int_nopo; i++) primaryout[nopo++]=po_gates[i]->index;//原始输出数组
    for(i=0; i<nog; i++) {
        cg=net[i];
        if(cg->noutput>0) {
       ALLOCATE(cg->outlis, GATEPTR, cg->noutput);
       maxfout=MAX(maxfout,cg->noutput);
```

```
        cg->noutput=0;
         }
    }
/*
 * 以下 for 代码确定定 GATE struct 中的 outlis
 *
*/
    for(i=0; i<nog; i++) {
        cg=net[i];
        for(j=0;j<cg->ninput;j++)
            cg->inlis[j]->outlis[(cg->inlis[j]->noutput)++]=cg;
    }


    for(i=0; i<nog; i++) {
        cg=net[i];
        if (cg->noutput > 0) continue;
        for(j=0;j<int_nopo;j++)
           if (cg == po_gates[j]) break;
        if (j == int_nopo) {//如果没有输出门列表，但也不属于原始输出，那么有 error
           fprintf(stderr, "Error: floating output '%s' detected!\n", cg->symbol->symbol);
           nerrs++;
        }
    }
    if (nerrs > 0) {
       fprintf(stderr, "Workaround. You have to take one of the two actions:\n");
       fprintf(stderr, "      1. Remove all the floating output and associated gates, or\n");
       fprintf(stderr, "      2. Make each floating output a primary output.\n");
       return(-1);
    }

    if(int_nog==nog) return(nog);
    else return(-1);
}
```

2.3、ATALANTA 主程序定义

```
/*------main: Main program of atalanta----------------------------*/

void main(argc,argv)
int argc; //命令参数个数
char *argv[];//命令各个参数字符内容
{
     register int i,j;
     int shuf;
     boolean done=FALSE;
```

```
    int number;
    int LEVEL=0;
    int nrestoredfault,maxdetect;
    FAULTTYPE *pcurrentfault,*f;
    GATEPTR gut,*stem;
    int nstem;
    int k,n,iteration;
    status state,fault_selection_mode;
    int nbacktrack;
    int ndetect=0, nredundant=0, noverbacktrack=0, ntest=0;
    int ntest1=0, ntest2=0, ntest3=0;
    int ndetect3=0;
    int narray[MAXTEST],store=0;
    int tbacktrack=0;
    int lastfault;
    int ncomp=INFINITY,stop=ONE;
    double minutes,seconds,starttime,inittime,runtime1,runtime2;
    double fantime,fan1time,simtime1,simtime2,simtime3;

    int maxbits=BITSIZE;
    int fault_profile[BITSIZE];
    int nbit=0,npacket=0;
    char c;
    level LFSR[MAXPI],ran;
    int bit=0,packet=0;
    int phase2,nd;

    /*****************************************************************
     *                                                               *
     *              step 0: preprocess ---                           *
     *                      input and output file interface          *
     *                                                               *

*****************************************************************/
    option_set(argc,argv);//读取输入命令，进行功能设置

    if((circuit = fopen(name1,"r")) == NULL) {//网表文件不存在
        fprintf(stderr,"Fatal error: no such file exists %s\n",name1);
        exit(0);
    }

    strcpy(cctname,name1);

    i=0; j=0;
```

```
    if(name2[0]=='\0') {//可以在 option_set 中给 name2 赋值，如果没有命名，那么
name2 空，此处进行默认命名
        while((c=name1[i++])!='\0') {
            if(c=='/') j=0;
            else if(c=='.') break;
            else name2[j++]=c;
        }
        name2[j]='\0';
        strcat(name2,".test");//测试输出结果
    }

    i=0; j=0;
    if(name3[0]=='\0') {
        while((c=name1[i++])!='\0') {
            if(c=='/') j=0;
            else if(c=='.') break;
            else name3[j++]=c;
        }
        name3[j]='\0';
        strcat(name3,".log");//日志文件
    }

    if((test = fopen(name2,"w")) == NULL) {
        fprintf(stderr,"Fatal error: %s file open error\n",name2);
        exit(0);
    }

    if(logmode=='y')
        if((logfile = fopen(name3,"w")) == NULL) {
      fprintf(stderr,"Fatal error: %s file open error\n",name3);
      exit(0);
        }

#ifdef DEBUG

fprintf(stderr,"##################################################################
#\n");
    fprintf(stderr,"Warning: Atalanta is compiled in debug mode.\n");
    fprintf(stderr,"Warning: This mode may create an incorrect result or a wrong
output file.\n");
    fprintf(stderr,"Warning: To avoid it, please comment out DEBUG flag in
\"define.h\",\n");
    fprintf(stderr,"Warning: and recompile the program.\n");
```

```
fprintf(stderr,"##################################################################
#\n");
#else
    if(logmode=='y') print_log_topic(logfile,cctname);
#endif

#ifdef ISCAS85_NETLIST_MODE
    if(cctmode==ISCAS85 && faultmode=='f') {
        fprintf(stderr,"Fatal error in options:\n");
        fprintf(stderr,"The option -f can not combined with the option -I.\n");
        exit(0);
    }
#endif

    if(faultmode=='f')
        if((faultfile = fopen(faultname,"r")) == NULL) {
      fprintf(stderr,"Fatal error: %s file open error\n",faultname);
      exit(0);
        }

    iseed=Seed(iseed);

    gettime(&minutes,&seconds,&runtime1);
    starttime=runtime1;

    /*************************************************************
     *                                                           *
     *              step 1: preprocess ---                       *
     *                      construction of data structures      *
     *                                                           *
     *************************************************************/

    if(read_circuit(circuit,cctname) < 0) fatalerror(CIRCUITERROR);//read_cct.c,读
入网表，配置 GATE struct 中的部分参数
    fclose(circuit);

    if(nog<=0 || nopi<=0 || nopo<=0) {//如果门数或原始输入输出数为 0，那么网表
有故障
        fprintf(stderr,"Error: #pi=%d, #po=%d, #gate=%d\n",nopi,nopo,nog);
        fatalerror(CIRCUITERROR);
    }
    if(noff > 0) {//如果有 flip-flop 存在，软件无法进行测试生成
        fprintf(stderr,"Error: %d flip-flop exists in the circuit.\n",noff);
        fatalerror(CIRCUITERROR);
```

```
    }
    nodummy=add_PO();//structure.c，adds a PO gate at each primary output

    if(cctmode==ISCAS89) {
        ALLOCATE(stack.list,GATEPTR,nog+10);
        clear(stack);

#ifdef INCLUDE_HOPE
        allocate_stacks();//structure.c,分配堆栈
/*
 * structure.c,
 * psuedo-levels of each gate in net structure,
 * returns real depth of the circuit, maxlevel+1
*/
        maxlevel=compute_level();
        allocate_event_list();//structure.c, allocates levelized queue for event_list
        levelize();//gates are re-numbered and sorted in the levelized order affected
data structure
        add_sparegates();//structure.c, allocates memory spaces for the fault injection
        lastgate=nog-1;

        i=SetFFR();//stem.c,Identifies Fanout Free Region (FFR) and stems,确定扇出
分支
        j=SetDominator();//stem.c,Identifies dominators of each fanout stem

#else
        if(levelize(net,nog,nopi,nopo,noff,stack.list) <0) {//gates are re-numbered and
sorted in the levelized order affected data structure
            fprintf(stderr,"Fatal error: Invalid circuit file.\n");
            exit(0);
        }
#endif

        if(noff > 0) {
            fprintf(stderr,"Error: Invalid type DFF is defined.\n");
            exit(0);
        }
    } else stack.list=NULL;

/*
 * io.c，
 * Set several circuit parameters (ltype) of the net data structure.
 * Computes the level (dpi) of each gate
 *
```

```
*/
    maxlevel=set_cct_parameters(nog,nopi);
#ifdef INCLUDE_HOPE
    LEVEL=maxlevel+2;
#else
    LEVEL=maxlevel;
#endif

    if(_MODE_SIM=='f') { /* FSIM */
        if(!allocate_dynamic_buffers(nog)) {
            fprintf(stderr,"Fatal error: memory allocation error\n");
            exit(0);
        }

        nstem=0;
        for(i=0;i<nog;i++)
            if(is_fanout(net[i]) || net[i]->fn==PO) nstem++;
        stem=(GATEPTR *)malloc((unsigned)(sizeof(GATEPTR)*nstem));
/*
 *  io.c
 *  setfanoutstem
 *  For fault simulation.
 *  Identifies fanout stems and builds fanout free region.
*/
        setfanoutstem(nog,stem,nstem);

        nof = (faultmode=='f') ?
                readfaults(faultfile,nog,nstem,stem)    //define_fault_list.c,    如    果
faultmode 为 f，读故障列表
          set_all_fault_list(nog,nstem,stem); //define_fault_list.c，如果 faultmode 不
为 f，创建故障列表,进行故障压缩
    }

#ifdef INCLUDE_HOPE
/*
 * define_fault_list.c
 * defines fault list in the order of DFS from primary outputs
 * use recursive version DFS routines.
 * main of DFS_po.
*/
    else { /* HOPE */
        if(faultmode=='d') FWD_faults();
        else readfaults_hope(faultfile);//reads fault list and constructs fault list data
structure
```

```
    }
#endif

    if(nof<0) {
        fprintf(stderr,"Fatal error: error in setting fault list\n");
        exit(0);
    }

    set_testability(nog);//testability.c，Assign controllibility and observability

    for(i=0;i<nog;i++) {
        reset(net[i]->changed);
        net[i]->freach=nog;
        if(net[i]->dpi>=PPOlevel)
            printf("Error: gut=%s dpi=%d\n",net[i]->symbol->symbol, net[i]->dpi);

    }
/*
 *  io.c
 *  set_dominator
 *  Finds the immediate dominators of all fanout stems.
 *  Inputs   : circuit structure +
 *           nog (number of gates), maxdpi(maximum depth)
 *           Outputs    : dominators (u_path)
*/
    set_dominator(nog,LEVEL);
/*
 *  io.c
 *  set_unique_path
 *  For every fanout stem which has a dominator,
 *  this routine finds the unique path and property of the dominator.
 *  Used in FAN.
 *  Should be called after setting dominators.
*/
    set_unique_path(nog,LEVEL);

print_test_topic(test,nopi,nopo,name1);//pio.c

/*
 * learn.c
 * learn
 * preprocessing for static learning
 *
*/
```

```
    if(learnmode=='y') learn(nog,LEVEL);

    for(i=0;i<nof;i++) {
        faultlist[i]->detected=UNDETECTED;
        faultlist[i]->observe=ALL0;
    }
/*
 *  define_fault_list.c
 *  check_redundant_faults
 *  Identifies redundant faults in which more than two inputs of
 *  a gate are connected to one gate simultaneously and deletes
 *  the faults from the fault list.
 *
 *  Input: nog, the number of gates of the circuit
 *         Output: number of identified redundant faults
 *         Note: changed flags of all gates should be reset to 0
 *                 prior to call this function.
 *
 *         ppsfp.c
 *            pinit_simulation
 *            Initializes necessary flags and counters.
 *            changed=0, observe=ALL0 clear event_list
*/
    if(_MODE_SIM=='f') {
        nredundant=check_redundant_faults(nog);
        pinit_simulation(nog,LEVEL,nopi);
    } else {
       InitFaultSim();//fsim.c, Initializes flags for the fault simulation
    }

    maxdetect=nof;

    all_one=ALL1;

    gettime(&minutes,&seconds,&runtime2);
    inittime=runtime2-runtime1;
    runtime1=runtime2;
    simtime1=0.0;

    /*****************************************************************
     *                                                               *
     *              step 2: Random pattern testing session           *
     *                      1. generate 32 random patterns           *
     *                      2. fault free simulation                 *
```

```
     *                    3. fault simulation                                  *
     *                    4. fault dropping                                     *
     *                                                                              *
**************************************************************/

    if(is_random_mode(rptmode)) {

        ndetect=random_sim(nog,nopi,nopo,LEVEL,nstem,stem,LFSR,randomlimit,
                maxbits,maxdetect,&ntest,&npacket,&nbit,test);
/*sim.c
 * Performs random similation until n consecutive packets of
 * random patterns do not detect any new fault
*/
        ntest1=ntest;
        gettime(&minutes,&seconds,&runtime2);
        simtime1=runtime2-runtime1;
        runtime1=runtime2;
    }


/*****************************************************************
     *
*
     *       step 3: Deterministic Test Pattern Generation Session          *
     *                    (fan with unique path sensitization                    *
     *
*

**************************************************************/
    reset(phase2);/* 0: static, 1:dynamic unique path sensitization */
    fantime=0;

ndetect+=testgen(nog,nopi,nopo,LEVEL,maxbits,nstem,stem,maxbacktrack,//sim.c
                    phase2,&nredundant,&noverbacktrack,&tbacktrack,
                    &ntest,&npacket,&nbit,&fantime,test);
    ntest2=ntest;


/*****************************************************************
     *
*
     *       step 4: Deterministic Test Pattern Generation Session          *
     *                    Phase 2: Employs dynamic unique path sensitization    *
```

```
     *
*

*****************************************************************/

    state=NO_TEST;
    if(maxbacktrack1>0 && nof-ndetect-nredundant>0) {
        set(phase2);
        for(i=0;i<nof;i++) {
            f=faultlist[i];
            if(f->detected==PROCESSED) { f->detected=UNDETECTED; }
        }
        fan1time=0;
        ndetect+=testgen(nog,nopi,nopo,LEVEL,maxbits,nstem,stem,maxbacktrack1,
                                phase2,&nredundant,&noverbacktrack,&tbacktrack,
                        &ntest,&npacket,&nbit,&fan1time,test);
        fantime+=fan1time;
    }

    ntest2=ntest;
    gettime(&minutes,&seconds,&runtime2);
    simtime2=runtime2-fantime-runtime1;
    runtime1=runtime2;



/*****************************************************************
     *
*
     *          step 5: Test compaction session                    *
     *                  32-bit reverse fault simulation           *
     *                  + shuffling compaction                     *
     *
*

*****************************************************************/

    if(ntest==0) {
        ntest3=0;
        ndetect3=0;
    } else if(compact=='n') {
        ntest3=ntest;
        ndetect3=ndetect;
    } else {
```

```
        if(maxcompact==0) compact='r';
        ntest3=    //sim.c

compact_test(nog,nopi,nopo,LEVEL,nstem,stem,nof,&shuf,&ndetect3,npacket,nbit,m
axbits,test);
        if(ndetect3 != ndetect) {
            printf("Error in test compaction: ndetect=%d,
ndetect3=%d\n",ndetect,ndetect3);
            exit(0);
        }
    }

    gettime(&minutes,&seconds,&runtime2);
    simtime3=runtime2-runtime1;
    runtime1=runtime2;
    msize = (int)sbrk(0)/1000;

    /* print out the results */
    print_atpg_head(stdout);//print.c，输出测试结果开头
    print_atpg_result(stdout,cctname,nog,nopi,nopo,LEVEL,//print.c，输出测试结果
                maxbacktrack,maxbacktrack1,phase2,ntest2,ntest3,nof,ndetect,
                nredundant,tbacktrack,shuf,inittime,
                simtime1+simtime2+simtime3,fantime,runtime2-starttime,'n',msize);

#ifndef DEBUG
    if(logmode=='y') {
        fprintf(logfile,"\nEnd of test pattern generation.\n\n");
        print_atpg_head(logfile);
        print_atpg_result(logfile,cctname,nog,nopi,nopo,LEVEL,
                maxbacktrack,maxbacktrack1,phase2,ntest2,ntest3,nof,ndetect,
                nredundant,tbacktrack,shuf,inittime,
                simtime1+simtime2+simtime3,fantime,runtime2-starttime,'y',msize);
    }
#endif

    if(ufaultmode=='y') {
        if((rfaultmode=='y' && ndetect<nof) ||
            (rfaultmode=='n' && ndetect+nredundant<nof)) {
            i=j=0;
            if(nameufaults[0]=='\0') {
                while((c=name1[i++])!='\0') {
                    if(c=='/') j=0;
                    else if(c=='.') break;
                    else nameufaults[j++]=c;
```

```
                }
                nameufaults[j]='\0';
                strcat(nameufaults,".ufaults");
            }
            if((ufaultfile = fopen(nameufaults,"r")) != NULL) {
                fprintf(stderr,"Warning: %s file already exists in the run directory\n" ,nameufaults);
                fprintf(stderr,"Warning: The undetected fault list file is not created.\n");
            } else {
                if((ufaultfile = fopen(nameufaults,"w")) == NULL) {
                    fprintf(stderr,"Fatal error: %s file open error\n",nameufaults);
                    exit(0);
                }
                print_undetected_faults(ufaultfile,'s',rfaultmode,0);//print.c，输出未检测到的故障
            }
            fclose(ufaultfile);
        }
    }

    fclose(test);
    if(logmode=='y') fclose(logfile);
}


int read_option(option,array,i,n)
char option,*array[];
int i,n;
{
    if(i+1 >= n) return((-1));

    switch(option) {
        case 'd': inputmode='d'; break;
#ifdef ISCAS85_NETLIST_MODE
        case 'I': cctmode=ISCAS85; break;
#endif
        case 'r': sscanf(array[++i],"%d",&randomlimit);
                  if(randomlimit==0) rptmode='n'; break;
        case 's': sscanf(array[++i],"%d",&iseed); break;
        case 'N': compact='n'; maxcompact=0; break;
        case 'c': sscanf(array[++i],"%d",&maxcompact); break;
        case 'b': sscanf(array[++i],"%d",&maxbacktrack); break;
        case 'B': sscanf(array[++i],"%d",&maxbacktrack1); break;
```

```
        case 't': strcpy(name2,array[++i]); break;
        case 'l': logmode='y'; strcpy(name3,array[++i]); break;
        case 'n': strcpy(name1,array[++i]); break;
        case 'h': helpmode=*(array[++i]); break;
        case 'L': learnmode='y'; break;
        case 'f': faultmode='f'; strcpy(faultname,array[++i]); break;
        case 'H': _MODE_SIM='h'; break;
        case '0': fillmode='0'; break;
        case '1': fillmode='1'; break;
        case 'X': fillmode='x'; break;
        case 'R': fillmode='r'; break;
        case 'A': gen_all_pat='y'; break;
        case 'D': sscanf(array[++i],"%d",&ntest_each_limit);
                  gen_all_pat='y'; break;
        case 'Z': no_faultsim='y'; break;
        case 'u': ufaultmode='y'; break;
        case 'U': ufaultmode='y'; strcpy(nameufaults,array[++i]); break;
        case 'v': ufaultmode='y'; rfaultmode='y'; break;
        default:    i=(-1);
    }
    return(i);
}

int option_set(argc,argv)
int argc; char *argv[];
{
    int i;

    if(argc==1) {//如果只有一个参数，那么采用默认配置
        helpmode='d';
    } else
        for(i=1;i<argc;i++) {
            if(argv[i][0]=='-') {//‘-’说明是参数
                if((i=read_option(argv[i][1],argv,i,argc))<0)  //<0 说明参数赋值结束,
提取每个参数内容，赋值给相应得内部参数
                { helpmode='d'; break; }
      }
      else strcpy(name1,argv[i]);//没有‘-’说明输入的是网表文件
        }

    if(helpmode!='q') { help(helpmode); exit(0); }
    if(fillmode=='x') _MODE_SIM='h';
    if(gen_all_pat=='y') {    //如果在 gen_all_pat 和 no_faultsim 模式中，在
read_option 中定义的某些参数需要重新赋值
```

```
        randomlimit=0;
        rptmode='n';
        maxbacktrack1=0;
        fillmode='x';
        compact='n';
        maxcompact=0;
        _MODE_SIM='h';
        no_faultsim='y';
    }

    if(no_faultsim=='y') {
        randomlimit=0;
        rptmode='n';
        fillmode='x';
        compact='n';
        maxcompact=0;
        _MODE_SIM='h';
    }
}
```

3、测试结果

3.1 待测电路

```
# c17
# 5 inputs
# 2 outputs
# 0 inverter
# 6 gates ( 6 NANDs )

INPUT(1)
INPUT(2)
INPUT(3)
INPUT(6)
INPUT(7)

OUTPUT(22)
OUTPUT(23)

10 = NAND(1, 3)
11 = NAND(3, 6)
16 = NAND(2, 11)
19 = NAND(11, 7)
22 = NAND(10, 16)
23 = NAND(16, 19)
```

3.2 测试结果

3.2.1 普通模式

命令：**./atalanta –x –l log c17.bench >temp**

输出结果：

```
* Log file for the circuit c17.bench.
* Number of faults detected by each test pattern:

test      1: 10101 11      7 faults detected
test      2: 01010 11      5 faults detected
test      3: 10010 00      6 faults detected
test      4: 01110 00      3 faults detected
test      5: 10111 10      1 faults detected

End of test pattern generation.

      ********************************************************
      *                                                      *
      *              Welcome to atalanta (version 2.0)       *
      *                                                      *
      *                  Dong S. Ha (ha@vt.edu)              *
      *               Web: http://www.ee.vt.edu/ha           *
      *   Virginia Polytechnic Institute & State University  *
      *                                                      *
      ********************************************************

******     SUMMARY OF TEST PATTERN GENERATION RESULTS     ******
1. Circuit structure
     Name of the circuit                    : c17
     Number of primary inputs               : 5
     Number of primary outputs              : 2
     Number of gates                        : 6
     Level of the circuit                   : 3

2. ATPG parameters
     Test pattern generation mode           : RPT + DTPG + TC
     Limit of random patterns (packets)     : 16
     Backtrack limit                        : 10
     Initial random number generator seed   : 1181224131
     Test pattern compaction mode           : REVERSE + SHUFFLE
     Limit of suffling compaction           : 2
     Number of shuffles                     : 4
```

3. Test pattern generation results

Number of test patterns before compaction : 8
Number of test patterns after compaction : 5
Fault coverage : 100.000 %
Number of collapsed faults : 22
Number of identified redundant faults : 0
Number of aborted faults : 0
Total number of backtrackings : 0

4. Memory used : 138113 Kbytes

5. CPU time

Initialization : 0.000 secs
Fault simulation : 0.000 secs
FAN : 0.000 secs
Total : 0.000 secs

3.2.2 诊断模式

命令：**./atalanta –A –l log c17.bench >temp**

输出结果：

*** Log file for the circuit c17.bench.**

*** Number of faults detected by each test pattern:**

7 /1

1: x00x0 00

11->19 /1

1: xx111 x0

16->23 /1

1: x10x0 11

2: x1100 11

23 /0

1: x10xx 11

2: x110x 11

3: x00x1 01

4: x0101 x1

19 /1

1: x00x1 01

2: x0101 x1

23 /1

1: x0xx0 x0

2: x0111 x0

3: x111x x0

6 /1

1: 0110x 1x

11 /1

1: 0111x 0x
2: 11110 10
3: 11111 10
4: x0111 x0
3->11 /1
1: x101x 1x
2: x0011 01
11 /0
1: x10xx 1x
2: 0110x 1x
3: 11100 11
4: 11101 11
5: x00x1 01
6: x0101 x1
2 /1
1: 000xx 0x
11->16 /1
1: x111x x0
16 /0
1: 00xxx 0x
2: 0111x 00
3: 100xx 0x
4: 101x0 10
5: 10111 10
6: 1111x 10
16 /1
1: 010xx 1x
2: 0110x 1x
3: 110xx 1x
4: 11100 11
3 /1
1: 100xx 0x
3 /0
1: 101xx 1x
1 /1
1: 001xx 0x
3->10 /1
1: 100xx 0x
16->22 /1
1: 010xx 11
2: 0110x 11
3: 110xx 11
22 /0
1: 1x1xx 1x

```
        2: 110xx 11
        3: 010xx 11
        4: 0110x 11
10 /1
        1: 101xx 1x
        2: 1111x 10
22 /1
        1: 00xxx 0x
        2: 0111x 00
        3: 100xx 0x

End of test pattern generation.

    ********************************************************
    *                                                      *
    *              Welcome to atalanta (version 2.0)       *
    *                                                      *
    *                  Dong S. Ha (ha@vt.edu)              *
    *               Web: http://www.ee.vt.edu/ha           *
    *  Virginia Polytechnic Institute & State University   *
    *                                                      *
    ********************************************************

******     SUMMARY OF TEST PATTERN GENERATION RESULTS     ******
1. Circuit structure
    Name of the circuit                     : c17
    Number of primary inputs                : 5
    Number of primary outputs               : 2
    Number of gates                         : 6
    Level of the circuit                    : 3

2. ATPG parameters
    Test pattern generation mode            : DTPG + TC
    Backtrack limit                         : 10
    Initial random number generator seed    : 1181224853
    Test pattern compaction mode            : NONE

3. Test pattern generation results
    Number of test patterns                 : 54
    Fault coverage                          : 100.000 %
    Number of collapsed faults              : 22
    Number of identified redundant faults   : 0
    Number of aborted faults                : 0
    Total number of backtrackings           : 23
```

4. Memory used : 138113 Kbytes

5. CPU time

Initialization : 0.000 secs

Fault simulation : 0.000 secs

FAN : 0.000 secs

Total : 0.000 secs

4、 算法修改

在 Fsim 的算法中（图 11），如果将第二个用铅笔圈处改为：if CD（s）*D（s，t）≠（00000）。其原因是：

图 11 Fsim 算法修改

参考文献：

[1] Michael L. Bushnell 和 Vishwani D. Agrawal 著，蒋安平，冯建华等译，“超大规模集成电路测试”，p142，电子工业出版社，2005

[2] Hyung Ki Lee; Dong Sam Ha, AN EFFICIENT, FORWARD FAULT SIMULATION ALGORITHM BASED ON THE PARALLEL PATTERN SINGLE FAULT PROPAGAT, Test Conference, 1991, Proceedings, International, 1991

[3] Hyung Ki Lee; Dong Sam Ha; HOPE: an efficient parallel fault simulator for synchronous sequential circuits， Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on Volume 15, Issue 9, Sept. 1996 Page(s):1048 – 1058

[4] H. K. Lee and D. S. Ha, "On the Generation of Test Patterns for Combinational Circuits,