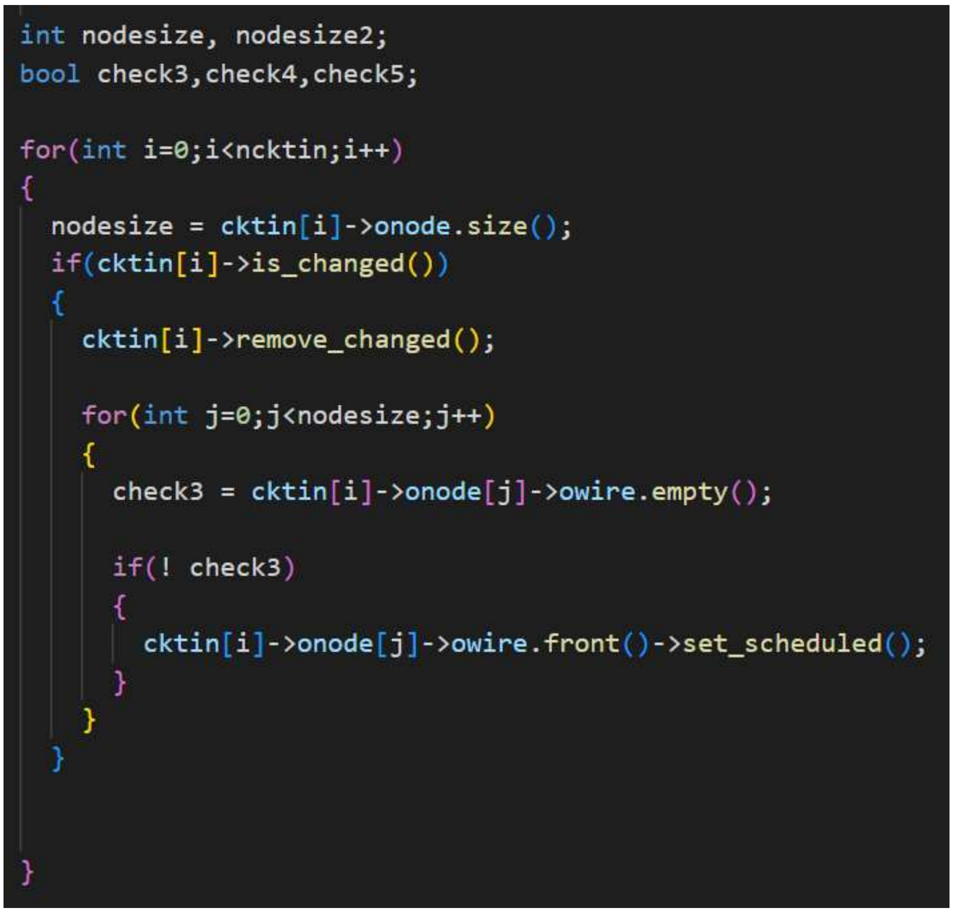
學號:R11943113  姓名:葉冠宏

## 一、程式修改

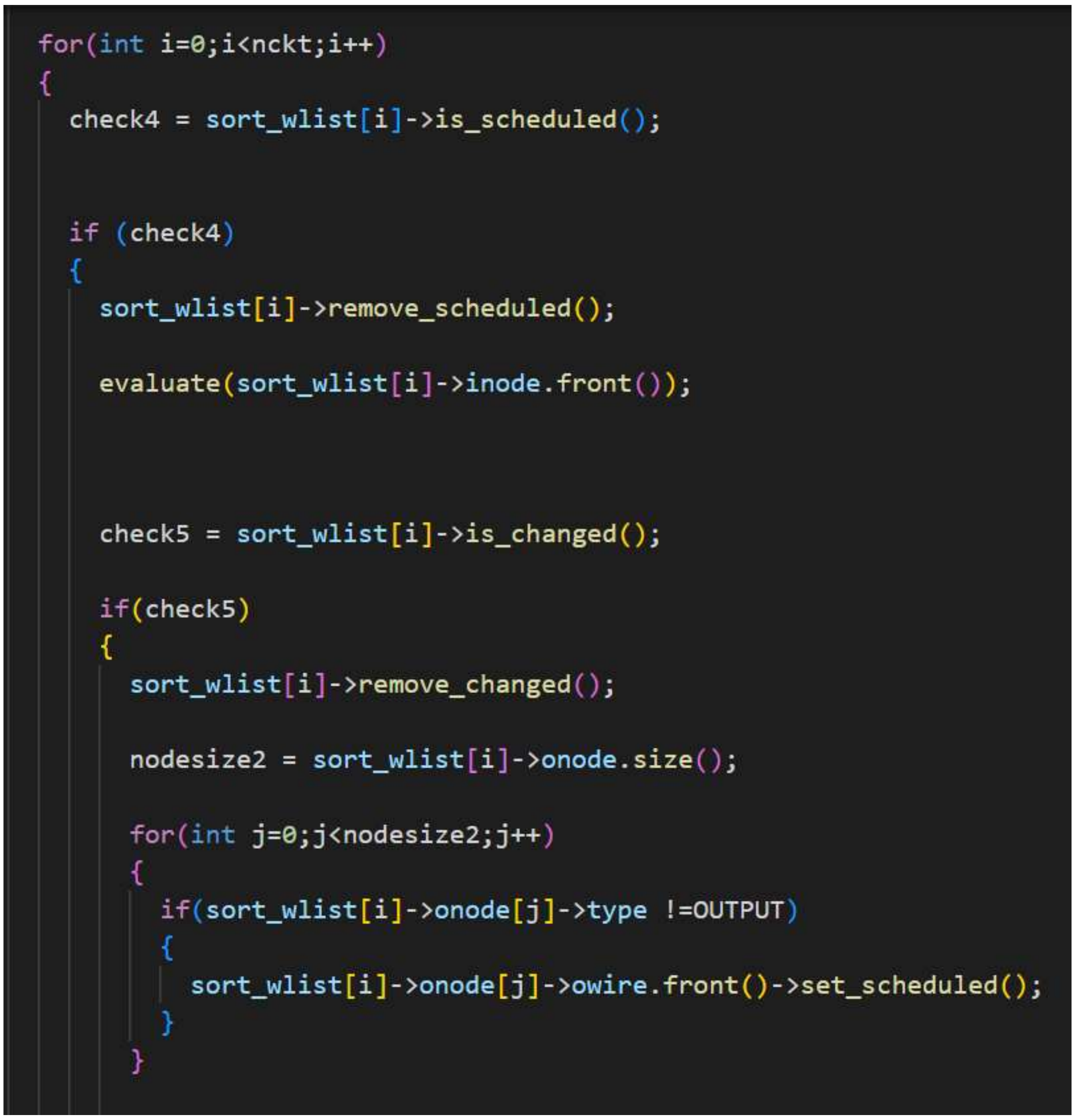
### 1.Sim.cpp

在 Sim.cpp 的檔案中，針對每個有改變的 input，我去找到對應連接到受其影響的 node 或是 gate 的 fanout wire，並對其進行 schedule 排程的動作，有待之後做進一步的處理。在過程中，我們必須去確保所影響的 fanout wire 是不是真的存在才去進行動作。除此之外，為了避免程式出現錯誤重複去檢查到相同的改變，因此要去把有討論過的改變的 input wire 去做 remove_changed 的動作。



```cpp
int nodesize, nodesize2;
bool check3,check4,check5;

for(int i=0;i<ncktin;i++)
{
  nodesize = cktin[i]->onode.size();
  if(cktin[i]->is_changed())
  {
    cktin[i]->remove_changed();

    for(int j=0;j<nodesize;j++)
    {
      check3 = cktin[i]->onode[j]->owire.empty();

      if(! check3)
      {
        cktin[i]->onode[j]->owire.front()->set_scheduled();
      }
    }
  }


}
```

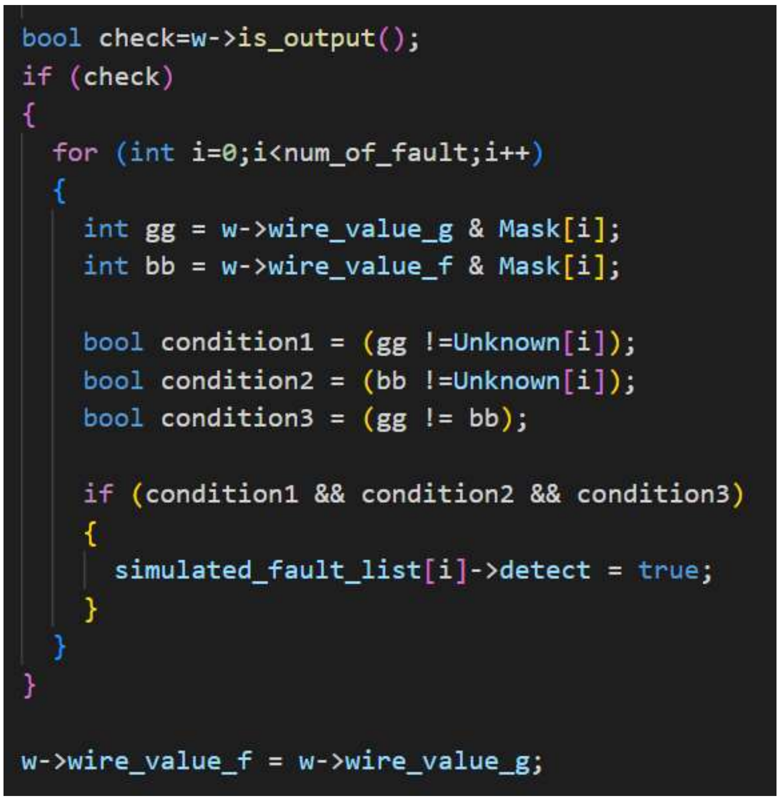
接著，我們就可以去逐步的 level by level 去評估 evaluate 每個已經被排程 scheduled 要去處理的每個 level 的 wire list 的 fan-in node，並考慮其衍生的影響。由於已經被探討過了，所以我們要去從排程中刪除。如果發現某個 wire 確實有因此而改變，我們要去探討其 fanout node 的 fanout wire 去做排程 schedule 的動作，有待之後去做進一步的處理。當然，我們也必須確保其不是 primary output 輸出的型態才會去探討其擴散的影響力。在探討完有改變的 wire 之後，記得要去對其 change 的狀態做回復。



```
for(int i=0;i<nckt;i++)
{
  check4 = sort_wlist[i]->is_scheduled();


  if (check4)
  {
    sort_wlist[i]->remove_scheduled();

    evaluate(sort_wlist[i]->inode.front());



    check5 = sort_wlist[i]->is_changed();

    if(check5)
    {
      sort_wlist[i]->remove_changed();

      nodesize2 = sort_wlist[i]->onode.size();

      for(int j=0;j<nodesize2;j++)
      {
        if(sort_wlist[i]->onode[j]->type !=OUTPUT)
        {
          sort_wlist[i]->onode[j]->owire.front()->set_scheduled();
        }
      }
```

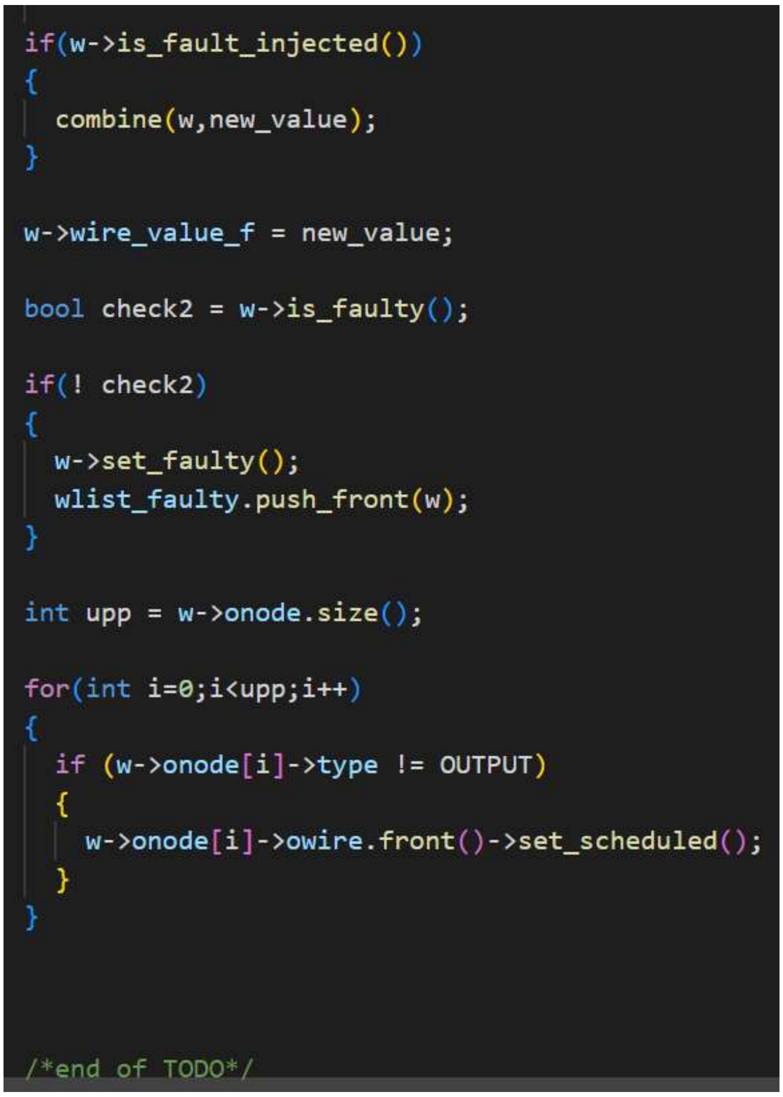
2.faultsim.cpp

在這個檔案中，我做了一些修改。在以下的程式碼中，我先確定 wire 是不是 output 輸出。如果是的話，我們就去看 fault-free wire 和有 fault 的 wire 的值是否確實有不一樣，而且兩者的值都不能是未知，則我們就可以說我們確實有偵測到那個錯了。在過程中，我們用 Mask 去做 bitwise 位數的比較，因為我們是用[00|11|01]來去代表 0, 1, 和未知。在標記完 true 之後，為了避免重複探討，我們會去把 faulty wire 改成 good wire。



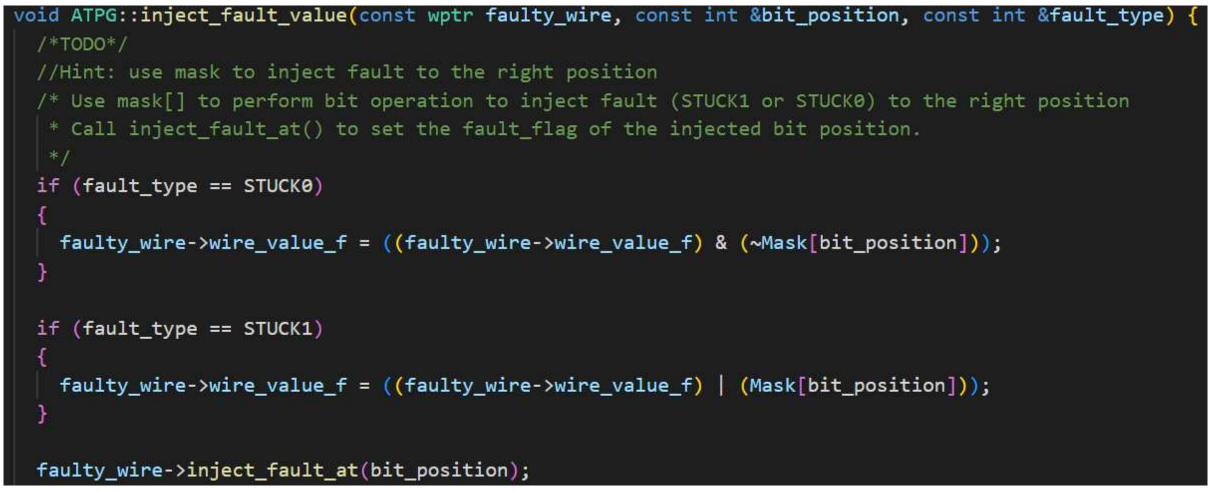
```
bool check=w->is_output();
if (check)
{
  for (int i=0;i<num_of_fault;i++)
  {
    int gg = w->wire_value_g & Mask[i];
    int bb = w->wire_value_f & Mask[i];

    bool condition1 = (gg !=Unknown[i]);
    bool condition2 = (bb !=Unknown[i]);
    bool condition3 = (gg != bb);

    if (condition1 && condition2 && condition3)
    {
      simulated_fault_list[i]->detect = true;
    }
  }
}

w->wire_value_f = w->wire_value_g;
```

接下來，在下一個程式修改中，我們是為了要去更新 faulty wire 的狀態。我們先使用 combine()的函數來去確保 injected 的 fault 在 new value 進來後還是被 injected 的。然後我們去更新 faulty wire 的值為 new_value。如果這個 wire 之前沒有被設定為 faulty，則我們去設定為 faulty，並把 wlist_faulty 中加入這個 wire。最後如果這個 wire 的 fan out node 不是 primary output 的話，我們就去對其 fanout node 的 fanout wire 去做排程 schedule，有待日後做進一步的處理。



```
if(w->is_fault_injected())
{
  combine(w,new_value);
}

w->wire_value_f = new_value;

bool check2 = w->is_faulty();

if(! check2)
{
  w->set_faulty();
  wlist_faulty.push_front(w);
}

int upp = w->onode.size();

for(int i=0;i<upp;i++)
{
  if (w->onode[i]->type != OUTPUT)
  {
    w->onode[i]->owire.front()->set_scheduled();
  }
}



/*end of TODO*/
```

在最後，我們對 inject_fault_value function 做了修改。我們分別對 stuck at 0 fault 和 stuck at 1 fault 做了處理。我們使用了 mask 去做 bitwise 的處理。在 stuck at 0 中，因為 and operation 只要有輸入為 0，其就恆等於 0，所以我們使用了&。而在 stuck at 1 中，因為 or operation 只要有輸入為 1，其就恆等於 1，所以我們用 |。最後我們用 inject_fault_at 去對 injected 錯誤的 bit 的位置去做紀錄。

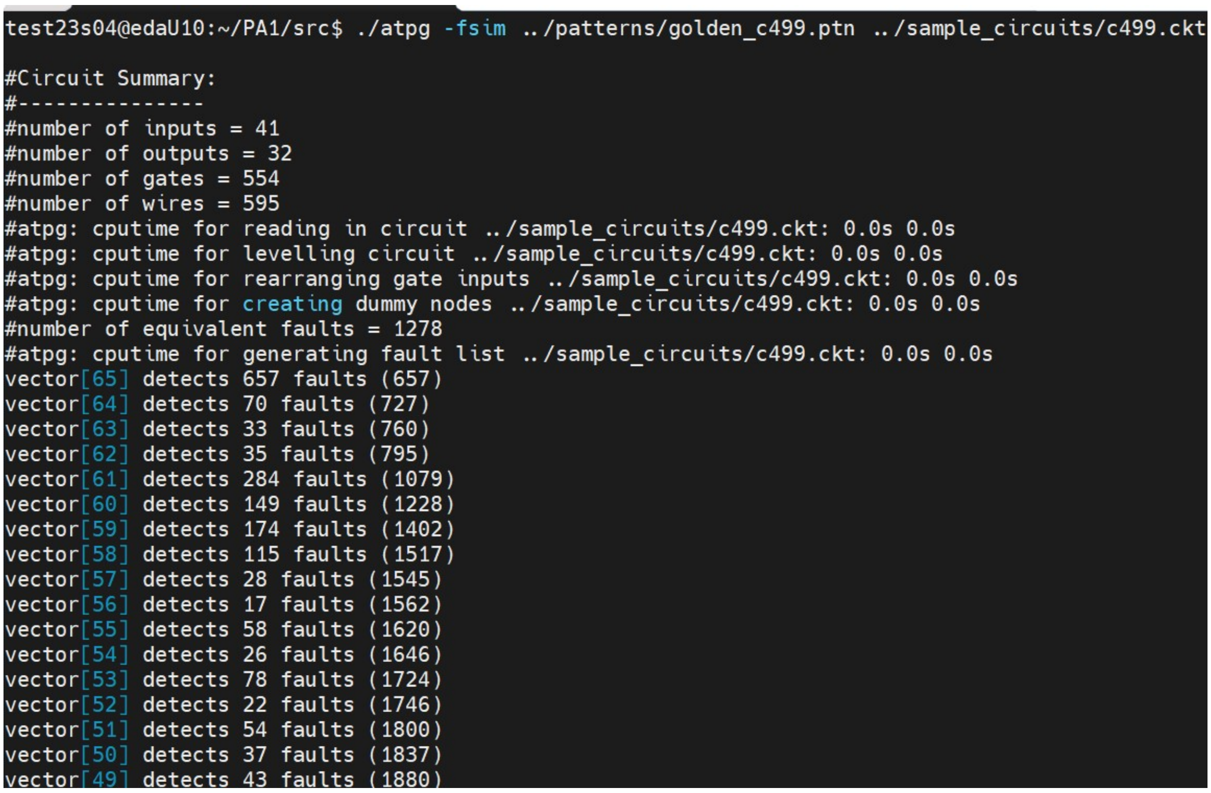


```
void ATPG::inject_fault_value(const wptr faulty_wire, const int &bit_position, const int &fault_type) {
  /*TODO*/
  //Hint: use mask to inject fault to the right position
  /* Use mask[] to perform bit operation to inject fault (STUCK1 or STUCK0) to the right position
   * Call inject_fault_at() to set the fault_flag of the injected bit position.
   */
  if (fault_type == STUCK0)
  {
    faulty_wire->wire_value_f = ((faulty_wire->wire_value_f) & (~Mask[bit_position]));
  }

  if (fault_type == STUCK1)
  {
    faulty_wire->wire_value_f = ((faulty_wire->wire_value_f) | (Mask[bit_position]));
  }

  faulty_wire->inject_fault_at(bit_position);
```

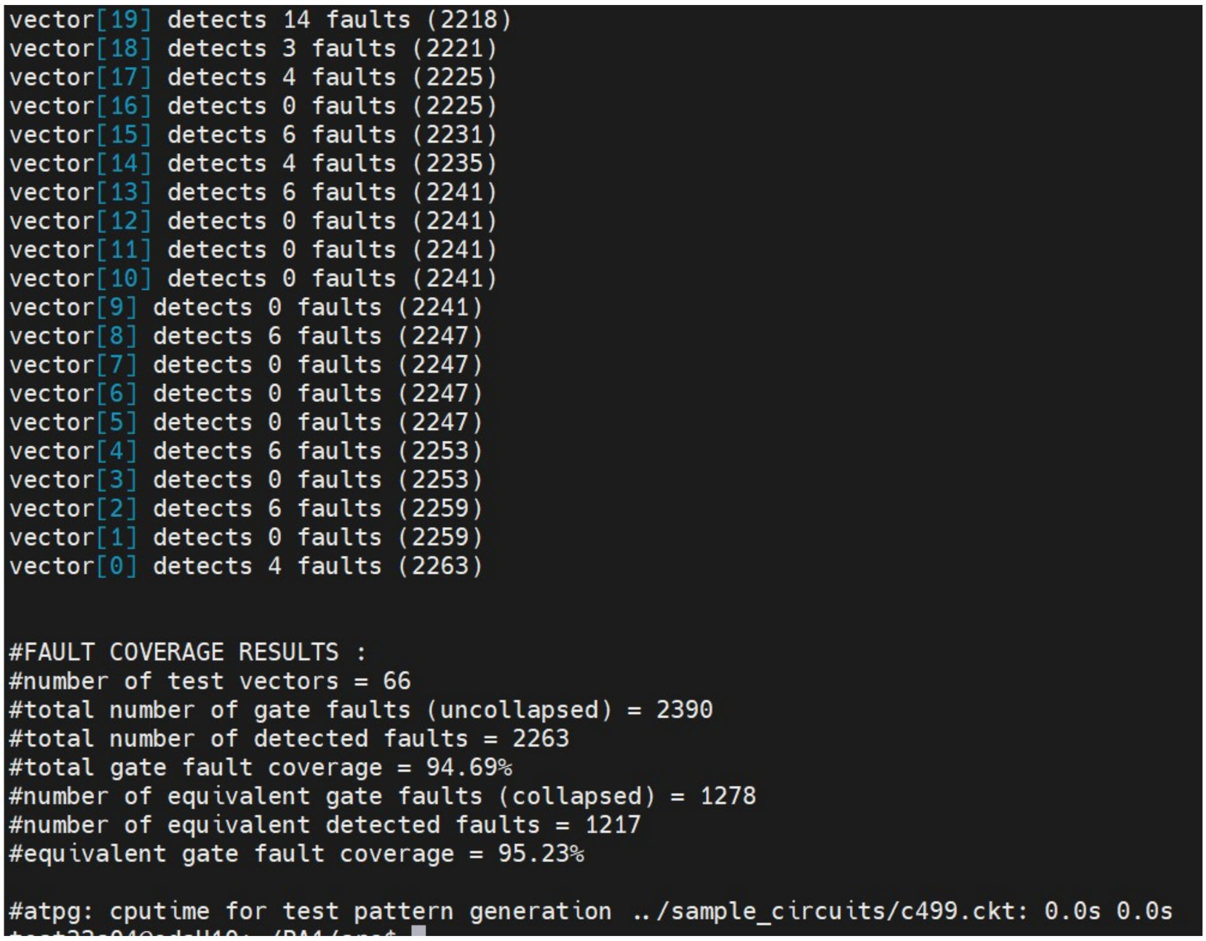
## 二、實驗結果

| circuit number | number of test vector | number of gates | number of total faults | number of detected faults | number of undetected faults | fault coverage |
|---|---|---|---|---|---|---|
| C499 | 66 | 554 | 2390 | 2263 | 127 | 94.69% |
| C1355 | 63 | 554 | 2726 | 1702 | 1024 | 62.44% |
| C6288 | 42 | 4800 | 17376 | 17109 | 267 | 98.46% |
| C7552 | 289 | 5679 | 19456 | 19144 | 312 | 98.40% |

C499:

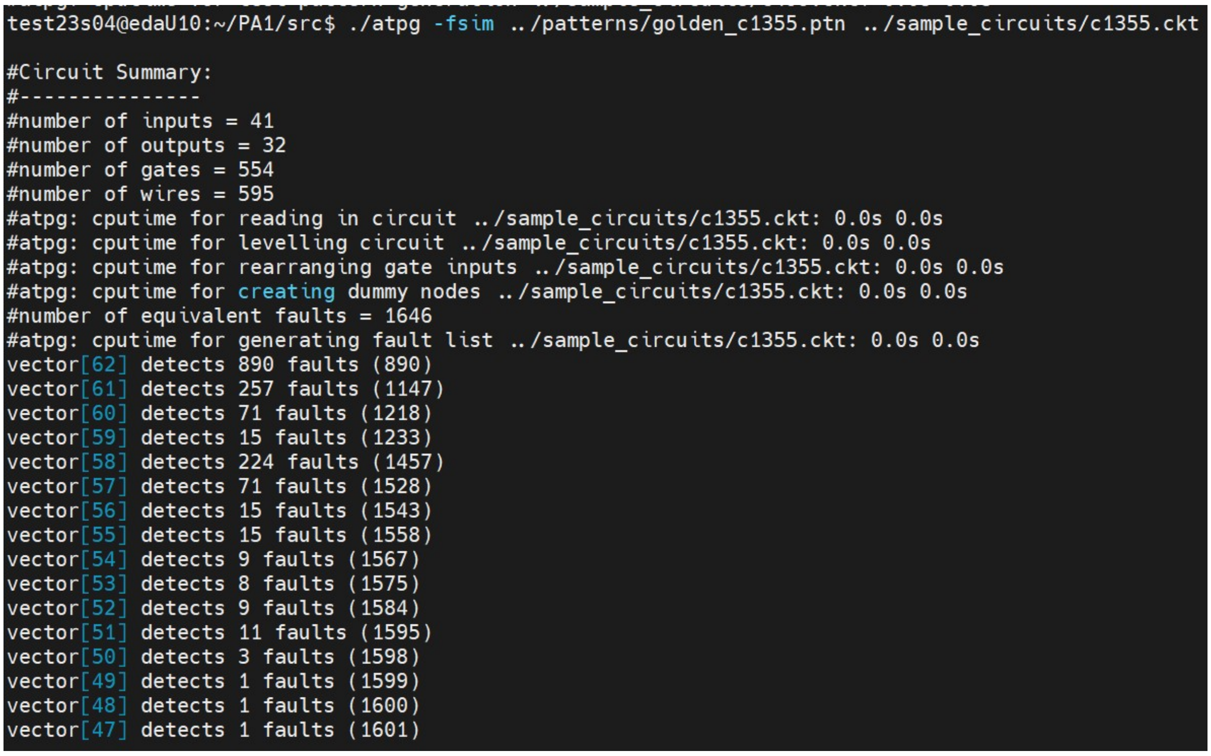


```
test23s04@edaU10:~/PA1/src$ ./atpg -fsim ../patterns/golden_c499.ptn ../sample_circuits/c499.ckt

#Circuit Summary:
#---------------
#number of inputs = 41
#number of outputs = 32
#number of gates = 554
#number of wires = 595
#atpg: cputime for reading in circuit ../sample_circuits/c499.ckt: 0.0s 0.0s
#atpg: cputime for levelling circuit ../sample_circuits/c499.ckt: 0.0s 0.0s
#atpg: cputime for rearranging gate inputs ../sample_circuits/c499.ckt: 0.0s 0.0s
#atpg: cputime for creating dummy nodes ../sample_circuits/c499.ckt: 0.0s 0.0s
#number of equivalent faults = 1278
#atpg: cputime for generating fault list ../sample_circuits/c499.ckt: 0.0s 0.0s
vector[65] detects 657 faults (657)
vector[64] detects 70 faults (727)
vector[63] detects 33 faults (760)
vector[62] detects 35 faults (795)
vector[61] detects 284 faults (1079)
vector[60] detects 149 faults (1228)
vector[59] detects 174 faults (1402)
vector[58] detects 115 faults (1517)
vector[57] detects 28 faults (1545)
vector[56] detects 17 faults (1562)
vector[55] detects 58 faults (1620)
vector[54] detects 26 faults (1646)
vector[53] detects 78 faults (1724)
vector[52] detects 22 faults (1746)
vector[51] detects 54 faults (1800)
vector[50] detects 37 faults (1837)
vector[49] detects 43 faults (1880)
```

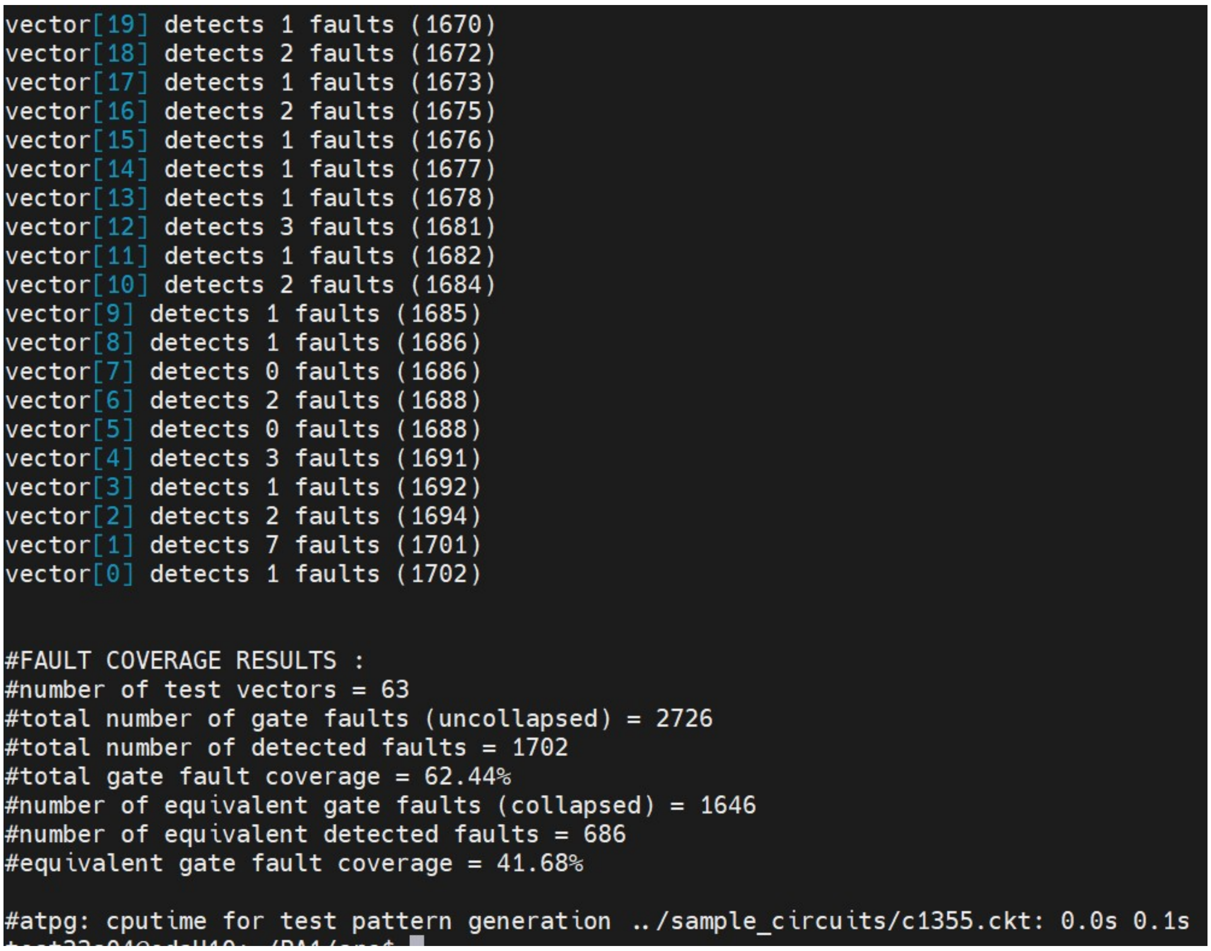


```
vector[19] detects 14 faults (2218)
vector[18] detects 3 faults (2221)
vector[17] detects 4 faults (2225)
vector[16] detects 0 faults (2225)
vector[15] detects 6 faults (2231)
vector[14] detects 4 faults (2235)
vector[13] detects 6 faults (2241)
vector[12] detects 0 faults (2241)
vector[11] detects 0 faults (2241)
vector[10] detects 0 faults (2241)
vector[9] detects 0 faults (2241)
vector[8] detects 6 faults (2247)
vector[7] detects 0 faults (2247)
vector[6] detects 0 faults (2247)
vector[5] detects 0 faults (2247)
vector[4] detects 6 faults (2253)
vector[3] detects 0 faults (2253)
vector[2] detects 6 faults (2259)
vector[1] detects 0 faults (2259)
vector[0] detects 4 faults (2263)


#FAULT COVERAGE RESULTS :
#number of test vectors = 66
#total number of gate faults (uncollapsed) = 2390
#total number of detected faults = 2263
#total gate fault coverage = 94.69%
#number of equivalent gate faults (collapsed) = 1278
#number of equivalent detected faults = 1217
#equivalent gate fault coverage = 95.23%

#atpg: cputime for test pattern generation ../sample_circuits/c499.ckt: 0.0s 0.0s
```

C1355:

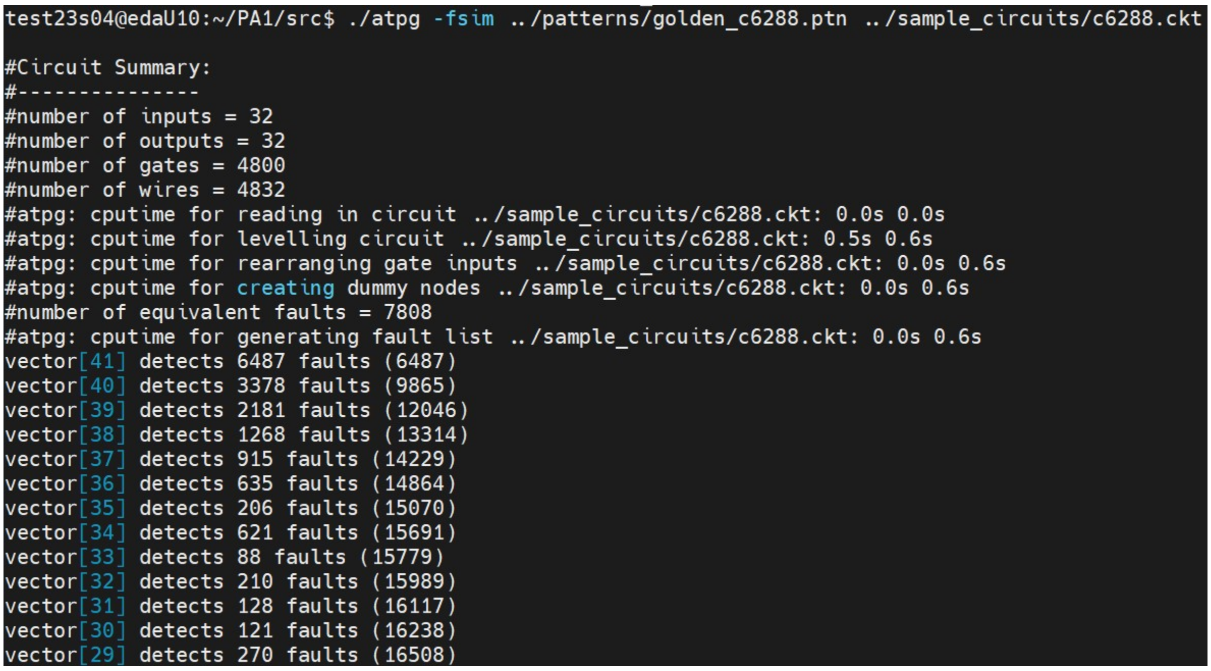


```
test23s04@edaU10:~/PA1/src$ ./atpg -fsim ../patterns/golden_c1355.ptn ../sample_circuits/c1355.ckt

#Circuit Summary:
#---------------
#number of inputs = 41
#number of outputs = 32
#number of gates = 554
#number of wires = 595
#atpg: cputime for reading in circuit ../sample_circuits/c1355.ckt: 0.0s 0.0s
#atpg: cputime for levelling circuit ../sample_circuits/c1355.ckt: 0.0s 0.0s
#atpg: cputime for rearranging gate inputs ../sample_circuits/c1355.ckt: 0.0s 0.0s
#atpg: cputime for creating dummy nodes ../sample_circuits/c1355.ckt: 0.0s 0.0s
#number of equivalent faults = 1646
#atpg: cputime for generating fault list ../sample_circuits/c1355.ckt: 0.0s 0.0s
vector[62] detects 890 faults (890)
vector[61] detects 257 faults (1147)
vector[60] detects 71 faults (1218)
vector[59] detects 15 faults (1233)
vector[58] detects 224 faults (1457)
vector[57] detects 71 faults (1528)
vector[56] detects 15 faults (1543)
vector[55] detects 15 faults (1558)
vector[54] detects 9 faults (1567)
vector[53] detects 8 faults (1575)
vector[52] detects 9 faults (1584)
vector[51] detects 11 faults (1595)
vector[50] detects 3 faults (1598)
vector[49] detects 1 faults (1599)
vector[48] detects 1 faults (1600)
vector[47] detects 1 faults (1601)
```

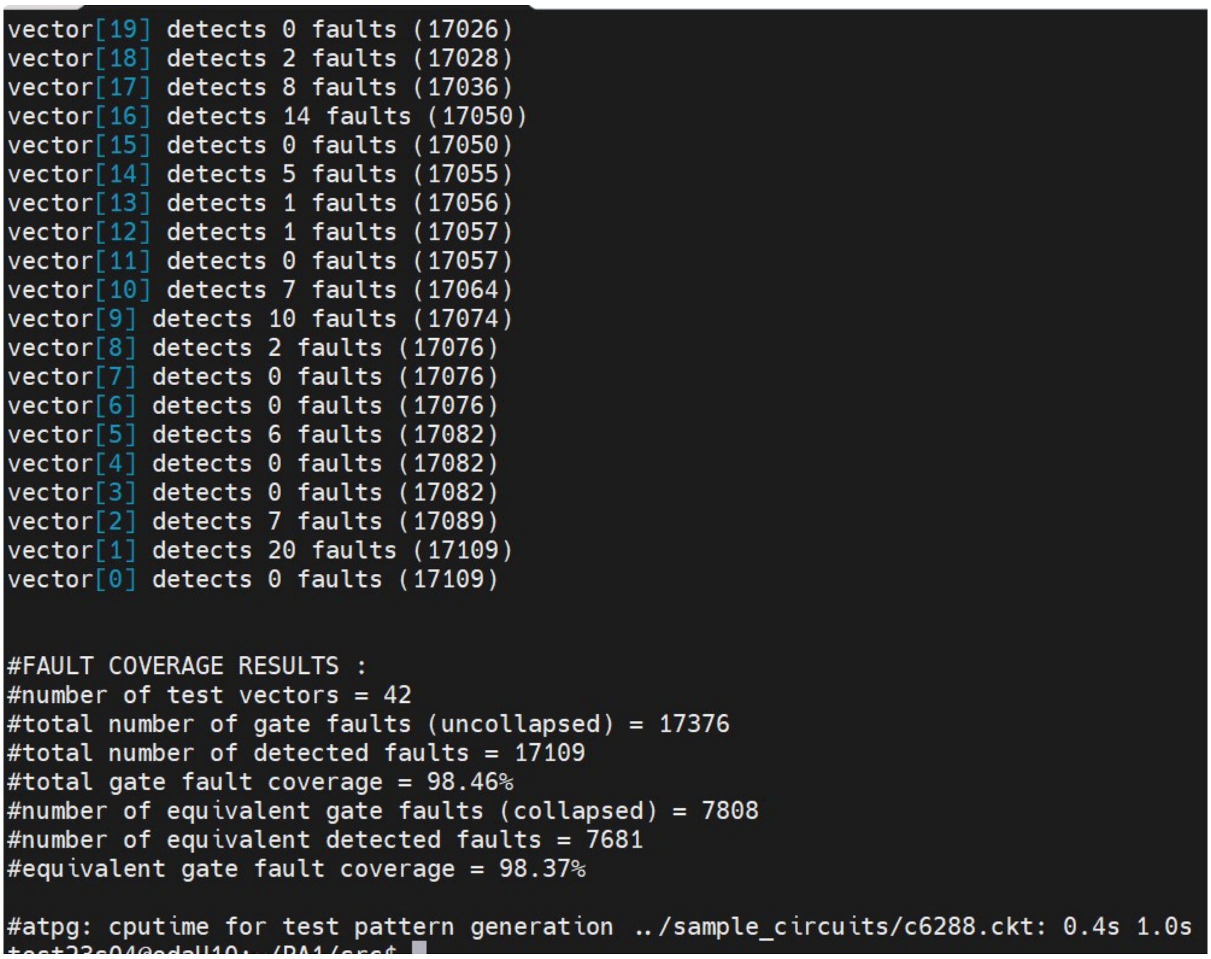


```
vector[19] detects 1 faults (1670)
vector[18] detects 2 faults (1672)
vector[17] detects 1 faults (1673)
vector[16] detects 2 faults (1675)
vector[15] detects 1 faults (1676)
vector[14] detects 1 faults (1677)
vector[13] detects 1 faults (1678)
vector[12] detects 3 faults (1681)
vector[11] detects 1 faults (1682)
vector[10] detects 2 faults (1684)
vector[9] detects 1 faults (1685)
vector[8] detects 1 faults (1686)
vector[7] detects 0 faults (1686)
vector[6] detects 2 faults (1688)
vector[5] detects 0 faults (1688)
vector[4] detects 3 faults (1691)
vector[3] detects 1 faults (1692)
vector[2] detects 2 faults (1694)
vector[1] detects 7 faults (1701)
vector[0] detects 1 faults (1702)


#FAULT COVERAGE RESULTS :
#number of test vectors = 63
#total number of gate faults (uncollapsed) = 2726
#total number of detected faults = 1702
#total gate fault coverage = 62.44%
#number of equivalent gate faults (collapsed) = 1646
#number of equivalent detected faults = 686
#equivalent gate fault coverage = 41.68%

#atpg: cputime for test pattern generation ../sample_circuits/c1355.ckt: 0.0s 0.1s
```

C6288:

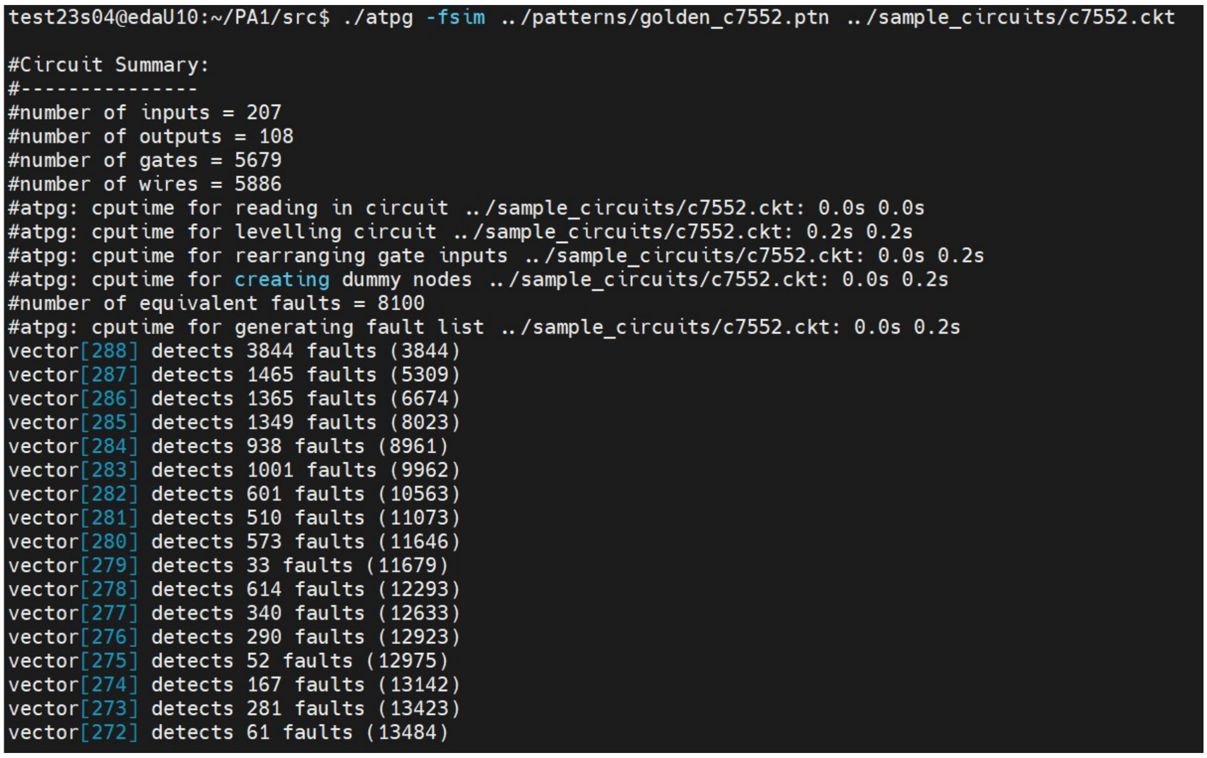


```
test23s04@edaU10:~/PA1/src$ ./atpg -fsim ../patterns/golden_c6288.ptn ../sample_circuits/c6288.ckt

#Circuit Summary:
#---------------
#number of inputs = 32
#number of outputs = 32
#number of gates = 4800
#number of wires = 4832
#atpg: cputime for reading in circuit ../sample_circuits/c6288.ckt: 0.0s 0.0s
#atpg: cputime for levelling circuit ../sample_circuits/c6288.ckt: 0.5s 0.6s
#atpg: cputime for rearranging gate inputs ../sample_circuits/c6288.ckt: 0.0s 0.6s
#atpg: cputime for creating dummy nodes ../sample_circuits/c6288.ckt: 0.0s 0.6s
#number of equivalent faults = 7808
#atpg: cputime for generating fault list ../sample_circuits/c6288.ckt: 0.0s 0.6s
vector[41] detects 6487 faults (6487)
vector[40] detects 3378 faults (9865)
vector[39] detects 2181 faults (12046)
vector[38] detects 1268 faults (13314)
vector[37] detects 915 faults (14229)
vector[36] detects 635 faults (14864)
vector[35] detects 206 faults (15070)
vector[34] detects 621 faults (15691)
vector[33] detects 88 faults (15779)
vector[32] detects 210 faults (15989)
vector[31] detects 128 faults (16117)
vector[30] detects 121 faults (16238)
vector[29] detects 270 faults (16508)
```

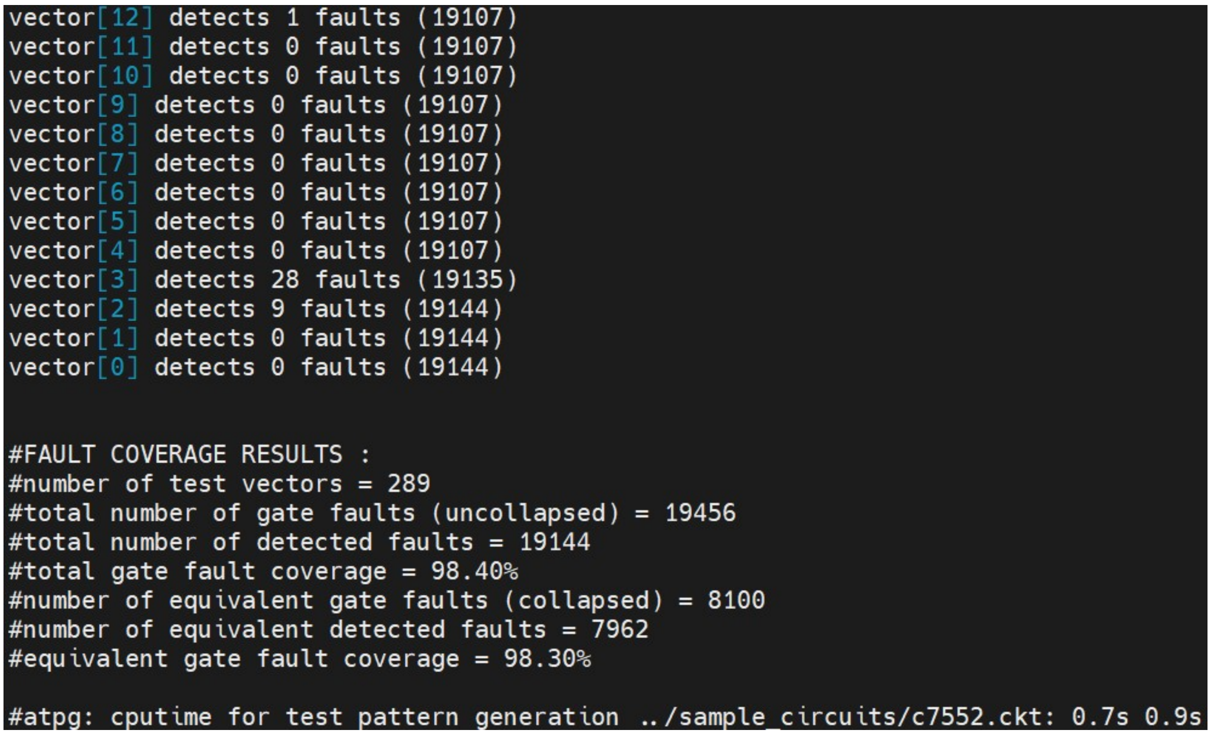


```
vector[19] detects 0 faults (17026)
vector[18] detects 2 faults (17028)
vector[17] detects 8 faults (17036)
vector[16] detects 14 faults (17050)
vector[15] detects 0 faults (17050)
vector[14] detects 5 faults (17055)
vector[13] detects 1 faults (17056)
vector[12] detects 1 faults (17057)
vector[11] detects 0 faults (17057)
vector[10] detects 7 faults (17064)
vector[9] detects 10 faults (17074)
vector[8] detects 2 faults (17076)
vector[7] detects 0 faults (17076)
vector[6] detects 0 faults (17076)
vector[5] detects 6 faults (17082)
vector[4] detects 0 faults (17082)
vector[3] detects 0 faults (17082)
vector[2] detects 7 faults (17089)
vector[1] detects 20 faults (17109)
vector[0] detects 0 faults (17109)


#FAULT COVERAGE RESULTS :
#number of test vectors = 42
#total number of gate faults (uncollapsed) = 17376
#total number of detected faults = 17109
#total gate fault coverage = 98.46%
#number of equivalent gate faults (collapsed) = 7808
#number of equivalent detected faults = 7681
#equivalent gate fault coverage = 98.37%

#atpg: cputime for test pattern generation ../sample_circuits/c6288.ckt: 0.4s 1.0s
```

C7552:



```
test23s04@edaU10:~/PA1/src$ ./atpg -fsim ../patterns/golden_c7552.ptn ../sample_circuits/c7552.ckt

#Circuit Summary:
#---------------
#number of inputs = 207
#number of outputs = 108
#number of gates = 5679
#number of wires = 5886
#atpg: cputime for reading in circuit ../sample_circuits/c7552.ckt: 0.0s 0.0s
#atpg: cputime for levelling circuit ../sample_circuits/c7552.ckt: 0.2s 0.2s
#atpg: cputime for rearranging gate inputs ../sample_circuits/c7552.ckt: 0.0s 0.2s
#atpg: cputime for creating dummy nodes ../sample_circuits/c7552.ckt: 0.0s 0.2s
#number of equivalent faults = 8100
#atpg: cputime for generating fault list ../sample_circuits/c7552.ckt: 0.0s 0.2s
vector[288] detects 3844 faults (3844)
vector[287] detects 1465 faults (5309)
vector[286] detects 1365 faults (6674)
vector[285] detects 1349 faults (8023)
vector[284] detects 938 faults (8961)
vector[283] detects 1001 faults (9962)
vector[282] detects 601 faults (10563)
vector[281] detects 510 faults (11073)
vector[280] detects 573 faults (11646)
vector[279] detects 33 faults (11679)
vector[278] detects 614 faults (12293)
vector[277] detects 340 faults (12633)
vector[276] detects 290 faults (12923)
vector[275] detects 52 faults (12975)
vector[274] detects 167 faults (13142)
vector[273] detects 281 faults (13423)
vector[272] detects 61 faults (13484)
```



```
vector[12] detects 1 faults (19107)
vector[11] detects 0 faults (19107)
vector[10] detects 0 faults (19107)
vector[9] detects 0 faults (19107)
vector[8] detects 0 faults (19107)
vector[7] detects 0 faults (19107)
vector[6] detects 0 faults (19107)
vector[5] detects 0 faults (19107)
vector[4] detects 0 faults (19107)
vector[3] detects 28 faults (19135)
vector[2] detects 9 faults (19144)
vector[1] detects 0 faults (19144)
vector[0] detects 0 faults (19144)


#FAULT COVERAGE RESULTS :
#number of test vectors = 289
#total number of gate faults (uncollapsed) = 19456
#total number of detected faults = 19144
#total gate fault coverage = 98.40%
#number of equivalent gate faults (collapsed) = 8100
#number of equivalent detected faults = 7962
#equivalent gate fault coverage = 98.30%

#atpg: cputime for test pattern generation ../sample_circuits/c7552.ckt: 0.7s 0.9s
```

三、討論

我們可以從上述的實驗結果發現，不見得選取的 test vector 數量越多一定會產生較好的 fault coverage。例如:在 C1355 中，即使其有比較少的 554 個 gate，而且使用較多的 63 個 test vector，和 C6288 比其來，雖然 C6288 有 4800 個比較多的 gate，和使用 42 個比較少的 test vector，但 C6288 卻可以達到 98.46%的 fault coverage，明顯比 C1355 的 62.44%的 fault coverage 還要好。而 C499 和 C1355 使用相近的 test vector 數量以及 gate number，最後達成的 fault coverage 卻差異很大。而 C7552 比 C6288 使用很多倍的 test vector，然而所得出的 fault coverage 卻沒提升多少。因此從實驗中可以發現所選取的 test pattern 的重要性對於電路 fault 的偵測有著相當程度的影響。