VLSI Testing PA2 Report

R05921058

楊承翰

1. Testcase result

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| circuit number | number of gates | number of total faults | number of detected faults | number of undetected faults | fault coverage | number of test vector | run time |
| C432 | 245 | 1110 | 149 | 961 | 13.42% | 20 | 0.041 s |
| C499 | 554 | 2390 | 2263 | 127 | 94.69% | 66 | 0.146 s |
| C880 | 545 | 2104 | 1254 | 850 | 59.60% | 65 | 0.147 s |
| C1355 | 554 | 2726 | 1702 | 1024 | 62.44% | 63 | 0.450 s |
| C2670 | 1785 | 6520 | 6278 | 242 | 96.29% | 135 | 0.422 s |
| C3540 | 2082 | 7910 | 2424 | 5486 | 30.64% | 98 | 5.374 s |
| C6288 | 4800 | 17376 | 17109 | 267 | 98.46% | 42 | 0.434 s |
| C7552 | 5679 | 19456 | 19144 | 312 | 98.40% | 289 | 1.812 s |

1. Code explanation
2. In the TODO of X-path detection, check if a wire has unknown value and is a primary output. If not, recursively check its fanout.

|  |
| --- |
| if (w->value == U) {  for (i = 0; i < ncktout; ++i)  if (cktout[i] == w)  return TRUE;  for (i = 0; i < w->nout; ++i)  for (j = 0; j < w->onode[i]->nout; ++j)  if (trace\_unknown\_path(w->onode[i]->owire[j]) == TRUE)  return TRUE;  }  return FALSE; |

1. In the TODO of check\_test(), check if any output’s value is D or B(D’).

|  |
| --- |
| for (i = 0; i < ncktout; ++i) {  wptr out = cktout[i];  if (out->value == D || out->value == B)  is\_test = TRUE;  }  return is\_test; |

1. In the TODO of fault excitation, backward imply the value corresponding to the fault and check if a PI is reached.

|  |
| --- |
| int v = (fault->fault\_type ? 0 : 1);  switch (backward\_imply(w, v)) {  case TRUE: pi\_is\_reach = TRUE; break;  case CONFLICT: return CONFLICT; break;  case FALSE: break;  } |

1. Speed up technique

The original X-path detection will check a wire multiple times if some output converges. We can add a flag to mark if a wire is already visited to prevent it. This method adds some overhead to runtime, but will be useful when there are a lot of output convergence.

|  |
| --- |
| struct WIRE {  ...  int dfs\_flag;  ...  }; |

|  |
| --- |
| trace\_unknown\_path(w) wptr w; {  int trace\_unknown\_path\_rec();  dfs\_counter += 1;  return trace\_unknown\_path\_rec(w);  }  trace\_unknown\_path\_rec(w) wptr w; {  if (w->dfs\_flag == dfs\_counter) return FALSE;  w->dfs\_flag – dfs\_counter;  ... // same as original  } |

The following table shows the runtime (in seconds) of the golden program (**golden\_atpg**), my program before improvement (**atpg**) and my program after improvement (**atpg\_improve**).

|  |  |  |  |
| --- | --- | --- | --- |
| circuit number | golden\_atpg | atpg | atpg\_improve |
| C17 | 0.002 s | 0.002 s | 0.002 s |
| C432 | 0.052 s | 0.041 s | 0.046 s |
| C499 | 0.180 s | 0.146 s | 0.107 s |
| C880 | 0.105 s | 0.147 s | 0.101 s |
| C1355 | 0.296 s | 0.450 s | 0.330 s |
| C2670 | 0.318 s | 0.422 s | 0.416 s |
| C3540 | 5.647 s | 5.374 s | 5.377 s |
| C6288 | 0.466 s | 0.434 s | 0.330 s |
| C7552 | 1.997 s | 1.812 s | 1.655 s |