PERANCANGAN ATPG PADA RANGKAIAN KOMBINASIONAL DENGAN MENGGUNAKAN ALGORITMA EXHAUSTIVE

LAPORAN TUGAS BESAR

Laporan sebagai salah satu tugas untuk mata kuliah

Pengujian dan Keterujian Sistem Digital

Oleh

Asyraf Atthariq Putra Gary - 23221103

Marta Diana - 23221307

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

ABSTRAK

ATPG merupakan sebuah metode atau teknologi otomasi desain elektronik yang digunakan untuk menemukan urutan input yang ketika diterapkan pada sirkuit digital memungkinkan peralatan uji otomatis untuk membedakan antara perilaku sirkuit yang benar dan perilaku sirkuit yang salah yang disebabkan oleh cacat. Dalam penerapannya, algoritma ATPG ini akan memodelkan sebuah kesalahan dalam suatu rangkaian dengan melakukan injeksi logika pada salah satu *node*-nya. Pada pengujian kali ini, akan dilakukan perancangan ATPG pada rangkaian combinational guna menentukan test vector minimum dan test fault coverage maksimum, algoritma ATPG yang digunakan dalam pengujian kali ini adalah algoritma exhaustive. Algoritma exhaustive ini merupakan salah satu algoritma ATPG yang sederhana dan mampu membangkitkan seluruh pola pengujian dari seluruh test vector yang diberikan. Hasil dari pengujian dilakukan dengan menggunakan rangkaian combinational 8 input yang menghasilkan 256 pola pengujian atau test vector, dengan melakukan reduksi nilai *stuck-at-fault* yang terdeteksi pada tiap-tiap test vector maka didapatkan test vector minimum sebanyak 7 buah test vector dengan *fault coverage* sebesar 100%. Sebagai proses verifikasi program ATPG dilakukan juga pengujian pada rangkaian ISCAS'85 C17 dan rangkaian Tree Structure. Pada rangkaian ISCAS'85 C17 didapatkan *fault coverage* sebesar 100% dengan test vector minimum sebanyak 4 buah, sedangkan pada rangkaian Tree Structure didapatkan juga *fault coverage* sebesar 100% dengan test vector minimum sebanyak 4 buah.

Kata Kunci: *ATPG, exhaustive, combinational, test vector, stuck-at-fault, fault coverage, ISCAS'85 C17, Tree Structure.*

ABSTRACT

ATPG is an electronic design automation method or technology used to find input sequences which when applied to digital circuits enable automated test equipment to distinguish between correct circuit behavior and incorrect circuit behavior caused by defects. In its modeling, this ATPG algorithm will model a fault in a circuit by performing logic injection on one of its nodes. In this test, the design of ATPG in a combinational circuit will be carried out to determine the minimum test vector and maximum fault coverage test, the ATPG algorithm used in this test is the exhaustive algorithm. This exhaustive algorithm is one of the simple ATPG algorithms and is able to generate the entire test pattern of all given test vectors. The results of the test were carried out using a combinational series of 8 inputs that produced 256 test vector patterns or test vectors, by reducing the stuck-at-fault values detected in each test vector, a minimum vector test of 7 test vectors with fault coverage of 100% was obtained. As a verification process for the ATPG program, testing was also carried out on the ISCAS'85 C17 circuit and the Tree Structure circuit. In the ISCAS'85 C17 circuit, 100% fault coverage is obtained with 4 minimum test vectors, while in the Tree Structure circuit, 100% fault coverage is also obtained with 4 minimum test vectors.

Keywords: *ATPG, exhaustive, combinational, test vector, stuck-at-fault, fault coverage, ISCAS'85 C17, Tree Structure.*

DAFTAR ISI

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

ATPG (*Automatic Test Pattern Generation)* merupakan sebuah metode otomatisasi desain elektronik yang digunakan untuk menemukan urutan input pada suatu rangkaian digital guna menentukan cacat atau tidaknya sebuah rangkaian digital tersebut. Pada proses uji ATPG, bila diterapkan pada sebuah rangkaian digital akan menghasilkan sebuah peralatan uji otomatis untuk membedakan antara perilaku rangkaian yang benar dan perilaku rangkaian yang salah. Pada umumnya pola pengujian ini digunakan untuk menguji perangkat semikonduktor setelah melalui proses *manufacture* dan dalam beberapa kasus digunakan untuk menentukan penyebab kegagalan suatu rangkaian. Efektivitas dari ATPG ini diukur dari jumlah model kesalahan (*fault model)* yang terdeteksi dan jumlah pola uji yang dihasilkan, hal ini umumnya merujuk pada kualitas pengujian dan waktu pengujian. Sedangkan efisiensi ATPG dipengaruhi oleh model kesalahan, jenis rangkaian yang diujikan, kerumitan dari suatu rangkaian, dan kualitas test yang dibutuhkan.

Untuk menentukan cacat atau tidaknya suatu rangkaian pada chip, maka ditentukan DUT (*Device Under Test)* dengan beberapa parameter pengujian seperti, identifikasi *faults*, menghitung kecacatan dalam suatu rangkaian atau chip (*defect)*, dan level kualitasnya. Pada tahap pengujian kali ini, penulis akan membangkitkan pola pengujian suatu rangkaian *combinational* agar menghasilkan *test vector* minimum dan *test coverage* maksimum. *Test vector* merupakan sebuah kumpulan *input* yang disediakan untuk suatu sistem untuk menguji keberhasilan sebuah sistem tersebut, sedangkan *test coverage* merupakan sebuah persentase keberhasilan dari serangkaian proses pengujian yang telah dilakukan.

Salah satu metode yang digunakan dalam membangkitkan pola uji pada rangkaian kali ini yaitu dengan menggunakan metode algoritma exhaustive, yang mana metode ini merupakan sebuah teknik pencarian solusi secara *brute force* untuk masalah yang melibatkan pencarian elemen dengan sifat khusus. Metode ini bekerja dengan cara mendeklarasikan semua input yang ada pada rangkaian, setiap *n-input* akan dihasilkan sebanyak 2^n pola uji. Pada penerapannya metode ini digunakan untuk input rangkaian yang tidak terlalu besar dan cenderung digunakan pada rangkaian yang sederhana maka dari itu penulis menggunakan metode pengujian ini dikarenakan rangkaian yang akan diujikan cukup sederhana, yaitu kurang dari 15 input dan metode ini juga dapat membangkitkan seluruh pola pengujian yang diharapkan dapat mempropagasikan seluruh nilai *stuck-at-fault* pada setiap node rangkaian

dengan optimal. Pada pengujian kali ini juga dilakukan pengujian terhadap rangkaian ISCAS'85 C17 dan rangkaian Tree Structure yang bertujuan untuk memvalidasi program ATPG yang telah dirancang.

1.2 Tujuan

Tujuan dari perancangan ATPG (*Automatic Test Pattern Generation)* pada rangkaian *combinational* yaitu:

1. Untuk dapat menentukan *test vector* minimum pada rangkaian.
2. Untuk dapat menghasilkan *fault coverage* maksimum dari rangkaian *combinational* menggunakan metode *exhaustive*.

1.3 Metodologi Pengujian

Pengujian dilakukan dengan merancang sebuah program yang dapat membaca sebuah rangkaian *combinational* sederhana yang masing-masing terdiri dari gerbang AND, OR, NAND, NOR dan NOT. Pengujian juga dilakukan dengan menggunakan model ATPG untuk rangkaian *combinational* dengan metode algoritma *exhaustive*, metode ini digunakan sebab rangkaian yang akan diuji merupakan sebuah rangkaian *combinational* yang bersifat sederhana dan metode ini dapat membangkitkan seluruh pola uji secara optimal.

BAB 2
TINJAUAN PUSTAKA

2.1 *Automatic Test-Pattern Generator* (ATPG)

ATPG merupakan sebuah metode atau teknologi otomasi desain elektronik yang digunakan untuk menemukan urutan input yang ketika diterapkan pada sirkuit digital, memungkinkan peralatan uji otomatis untuk membedakan antara perilaku sirkuit yang benar dan perilaku sirkuit yang salah yang disebabkan oleh cacat, serta ATPG juga dapat diartikan sebagai sebuah proses untuk menghasilkan sebuah pola pengujian pada suatu rangkaian, yang di jelaskan pada *logic- level netlist*. ATPG umumnya digunakan untuk menghasilkan pola uji pada rangkaian, menemukan pola uji atau logika pada rangkaian yang berlebihan atau tidak perlu, dapat membuktikan implementasi pada rangkaian.

Dalam pemodelannya, algoritma ATPG ini akan memodelkan sebuah kesalahan dalam suatu rangkaian, dengan melakukan injeksi logika pada salah satu *node*-nya. Kemudian dengan menggunakan berbagai mekanisme, ATPG akan membangkitkan pola kesalahan tersebut agar menyebar dan sampai pada *output* rangkaian, jika sinyal pada *output* berubah dari nilai yang diharapkan (untuk rangkaian tanpa kesalahan), maka hal ini menyebabkan kesalahan pada rangkaian dapat dideteksi, dan jika sinyal pada *output* sama dengan nilai yang diharapkan (untuk rangkaian tanpa kesalahan), maka kesalahan pada rangkaian tidak dapat dideteksi.

Efek kesalahan ini dapat dilakukan atau dimodelkan dengan mempropagasikan nilai input dari gerbang AND/NAND ke nilai *output*-nya dengan menyetel salah satu input gerbang dengan nilai 1, sedangkan pada gerbang OR/NOR dengan nilai 0. Hal ini dilakukan karena pada gerbang AND/NAND akan sensitif jika diberi nilai 0, maka dari itu untuk mempropagasikan nilai kesalahan salah satu dari input gerbang harus diberikan nilai kebalikan dari 0, yaitu nilai 1, sedangkan pada gerbang OR/NOR akan sensitif jika diberi nilai 1, maka dari itu untuk mempropagasikan nilai kesalahan salah satu input dari gerbang harus diberikan nilai kebalikan dari 1, yaitu 0.

2.2 Pemodelan Stuck-at Fault

Pemodelan stuck-at fault dilakukan dengan mengasumsikan salah satu jalur sinyal atau *node* pada rangkaian terjebak pada logika yang tetap (baik itu logika 0 ataupun 1) terlepas dari input yang telah diberikan pada rangkaian saat pengujian. Maka dari itu jika sebuah rangkaian memiliki n jalur sinyal yang akan diujikan, maka terdapat 2^n *stuck-at fault* yang dapat didefinisikan pada rangkaian. Pemodelan stuck-at fault ini merupakan model kesalahan yang logis karena tidak ada informasi *delay* pada rangkaian yang dikaitkan dengan definisi kesalahan

yang terjadi, serta model ini juga disebut sebagai model gangguan permanen karena efek gangguan diasumsikan permanen, berbeda dengan gangguan pada umumnya terjadi secara acak, yang biasanya disebabkan oleh suhu, daya, tegangan, dan jalur sinyal. Pemodelan single-stuck at fault ini bersifat struktural, hal ini dikarenakan pemodelan ditentukan berdasarkan model rangkaian tingkat gerbang struktural yang akan diujikan. Sebuah pembangkitan pola uji dapat mencakup 100% deteksi kesalahan untuk mendeteksi setiap kemungkinan stuck-at-fault pada rangkaian. Namun 100% stuck-at fault yang terdeteksi tidak selalu menjamin kualitas tinggi dari suatu rangkaian, hal ini dikarenakan banyak jenis kesalahan lain seperti kesalahan dalam penghubung, kesalahan terbuka, dan kesalahan transisi (*delay).*

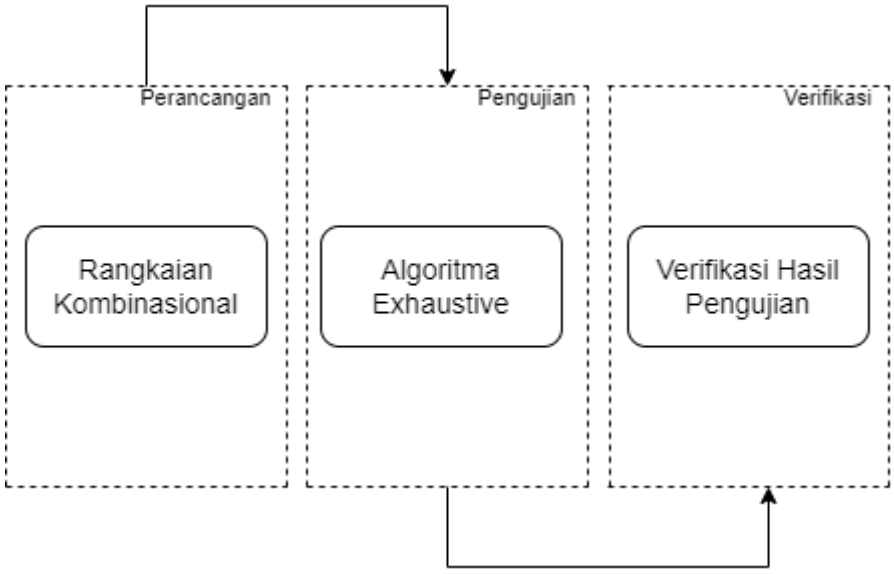
2.3 Algoritma ATPG

Adapun beberapa tipe dari algoritma ATPG yang akan digunakan untuk pengujian rangkaian pada tugas besar ini, yaitu metode Exhaustive. Algoritma exhaustive ini biasanya digunakan untuk rangkaian dengan input tidak terlalu besar (≤ 15 *inputs*). Dalam pendekatannya untuk rangkaian dengan n-input, maka harus membangkitkan semua pola pengujiannya dengan total 2^n pola uji. Pembangkitan pola uji dengan algoritma exhaustive ini dilakukan dengan membuat pola uji lengkap pada setiap *node*-nya dan membangkitkan nilai *fault* (*stuck-at-0* dan *stuck-at-1*) pada setiap *node*-nya.

BAB III
PERANCANGAN DAN ALGORITMA SISTEM

1.1 Rancangan Sistem

Sistem yang akan dibuat dalam tugas besar ini yaitu sistem pengujian yang dapat menemukan test vector minimum dan test fault coverage maksimum pada rangkaian combinational. Dalam perancangan sistem ini akan dilakukan dalam 3 tahapan seperti yang digambarkan pada Gambar 1, yaitu tahap perancangan rangkaian, tahap pengujian rangkaian, dan tahap verifikasi pengujian dari rangkaian.



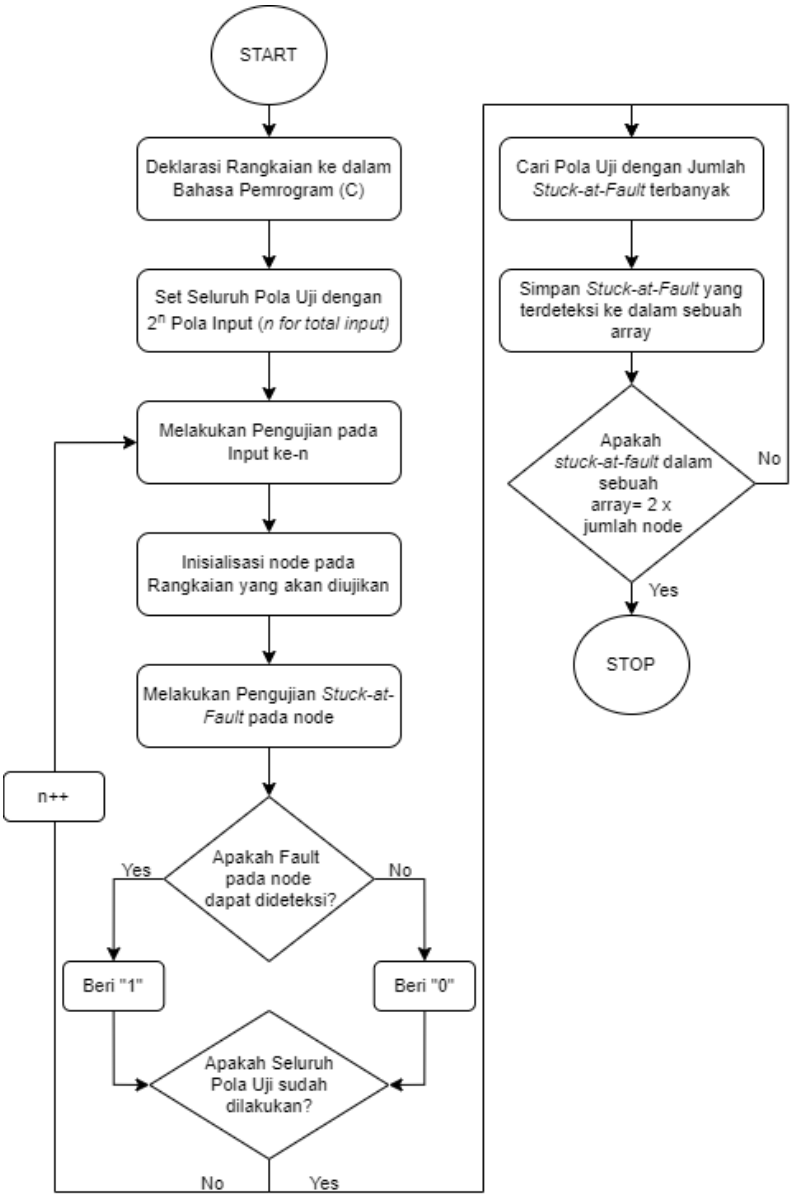
Gambar 1. Rancangan Sistem

Pada tahap perancangan, akan dirancang sebuah rangkaian combinational 8 input dengan 10 gerbang logika yang akan digunakan sebagai rangkaian representatif dari sistem pengujian. Setelah rancangan dari rangkaian selesai dilakukan maka masuk ke tahapan kedua, yaitu pengujian sistem, pengujian sistem dilakukan dengan menguji rangkaian yang telah dirancang dengan menggunakan salah satu algoritma ATPG yaitu algoritma exhaustive, pada tahapan ini akan dilakukan sebuah pola pengujian test vector pada setiap input yang telah dideklarasikan untuk mendapatkan *stuck-at-fault* pada setiap node dari rangkaian yang diujikan. Hasil dari *stuck-at-fault* ini akan di akumulasi lalu direduksi untuk mendapatkan test vector minimum dan *fault coverage* dari rangkaian. Tahapan terakhir dari perancangan sistem ini, yaitu melakukan verifikasi hasil pengujian, yang mana verifikasi ini merupakan sebuah proses pembuktian membangun kebenaran, akurasi atau validitas dari hasil pengujian yang telah dilakukan. Verifikasi yang akan dilakukan dari pengujian sistem ini, yaitu dengan

membandingkan hasil dari pengujian yang telah dilakukan dengan hasil teori yang telah didapatkan pada matakuliah Digital System Testing and Testable Design.

1.2 Algoritma Sistem

Algoritma dari sistem pengujian *stuck-at-fault* dalam tugas besar ini direpresentasikan melalui *flowchart* pada Gambar 2.



Gambar 2. Flowchart Sistem

Pada *flowchart* sistem dijelaskan bahwa sistem pembangkitan pola uji kesalahan pada rangkaian combinational ini diawali dengan melakukan perancangan rangkaian seperti pada Gambar 1 dan dari rancangan rangkaian akan terjemahkan ke dalam sebuah bahasa pemrograman. Pola pengujian yang akan dilakukan yaitu menggunakan tipe algoritma

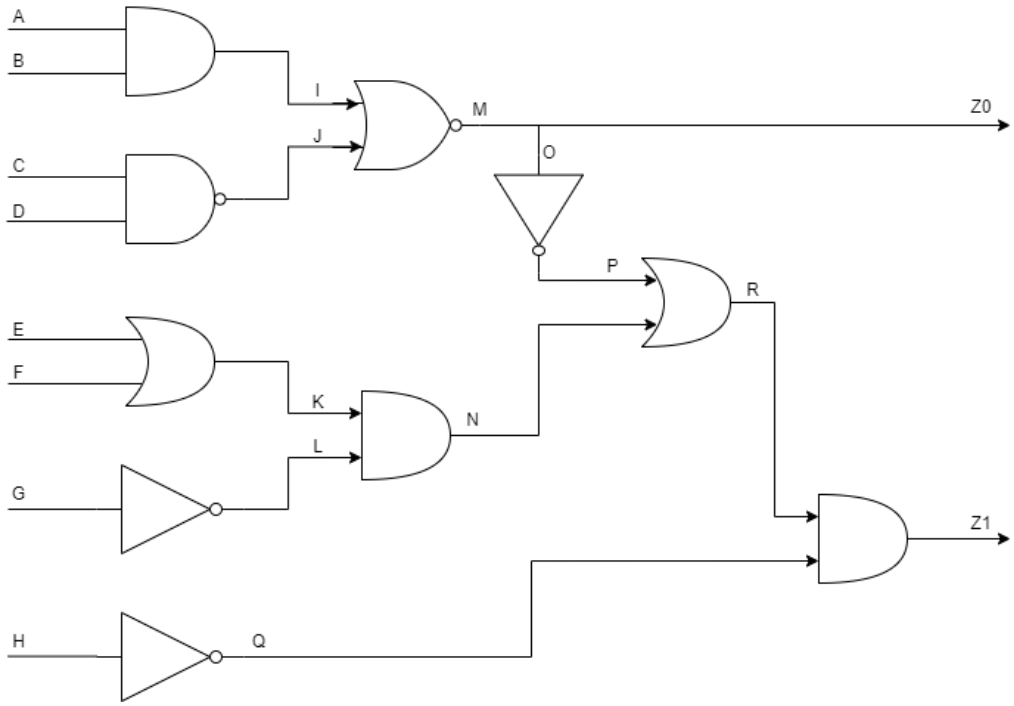
exhaustive, maka dari itu penting untuk menginisialisasi pola primary input sebanyak 2^n pola (yang mana n merupakan total keseluruhan input), pola ini berdasarkan aturan yang telah ditetapkan pada metode pengujian exhaustive. Setelah mendapatkan pola pengujian, maka pola-pola tersebut akan dibangkitkan atau di-*generate* melewati setiap *node* pada gerbang hingga *primary output*, pada pola pengujian ini didapatkan *output* untuk rangkaian sebelum diinjeksi oleh *fault* (*good circuit)*. Setelah mendapatkan seluruh hasil logika untuk rangkaian tanpa *fault* (*good circuit)*, maka dilanjutkan dengan membangkitkan pola uji kesalahan pada tiap-tiap *node* untuk dapat memodelkan *stuck-at-fault* pada rangkaian. Pemodelan kesalahan ini dilakukan dengan mengabaikan nilai *primary input* dan memaksa nilai suatu *node* menjadi 0 (*stuck-at-0*) atau 1 (*stuck-at-1*) tanpa pengaruh dari *primary input* yang telah diberikan. Pemodelan kesalahan ini akan diteruskan sampai *primary output*, dan jika hasil *primary output* dari nilai pemodelan kesalahan ini berbeda dengan nilai *primary output* pada rangkaian tanpa injeksi *fault* (*good circuit)* maka *fault* pada titik node (yang diuji) dapat dideteksi, namun jika tidak maka *fault* pada titik node (yang diuji) tidak dapat dideteksi.

Jika seluruh pola pengujian selesai dilakukan maka akan dicari jumlah *stuck-at-fault* dengan nilai unik terbanyak yang ada pada test vector yang telah terinisialisasi, jumlah terbanyak tersebut akan disimpan ke dalam sebuah array sejumlah dua kali node (karena pengujian tiap node terdiri dari *stuck-at-1* dan *stuck-at-0*), jika masih ada array yang belum terisi maka akan dilakukan perulangan kembali yaitu dengan mencari test vector dengan jumlah *stuck-at-fault* terbanyak kedua, hal ini akan dilakukan secara berulang hingga nilai dalam array telah terisi semua. Setelah nilai dalam array terisi semua, maka otomatis didapatkan juga jumlah test vector minimum dan fault coverage maksimum pada sebuah rangkaian yang akan diujikan.

BAB IV
HASIL DAN ANALISA SISTEM

4.1 Rancangan Rangkaian

Pada tugas besar ini akan dibuat sebuah rangkaian yang terdiri dari gerbang AND, OR, NAND, NOR dan NAND dengan 8 *primary input* dan 2 *primary output* seperti yang digambarkan pada Gambar 3. Rancangan Rangkaian.



Gambar 3. Rancangan Rangkaian

Rangkaian pada Gambar 3 ini akan dilakukan pengujian simulasi *stuck at fault* menggunakan ATPG dengan algoritma tipe exhaustive, yang mana tipe algoritma ini bekerja dengan cara meneruskan *fault* dari semua titik *node* (A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, Z0 dan Z1) ke *primary output*, jika hasil *primary output* dari semua propagasi *fault* ini mengalami perbedaan dengan *good circuit* atau rangkaian tanpa injeksi *fault*, maka pembangkitan pola uji kesalahan dapat dilakukan dalam rangkaian tersebut namun jika tidak maka pembangkitan pola uji kesalahan dalam rangkaian tidak dapat dibangkitkan atau tidak dapat mencapai target maksimum. Dari pola uji pembangkitan nilai *stuck-at-fault* pada rangkaian, akan didapatkan nilai *fault coverage* yang mengindikasikan persentase total kesalahan yang dapat diujikan dalam suatu rangkaian.

4.2 Pengujian Sistem

Pengujian rangkaian yang digambarkan pada Gambar 3 dilakukan dengan menggunakan software Visual Studio Code. Berikut tahapan yang dilakukan pada proses pengujian sistem.

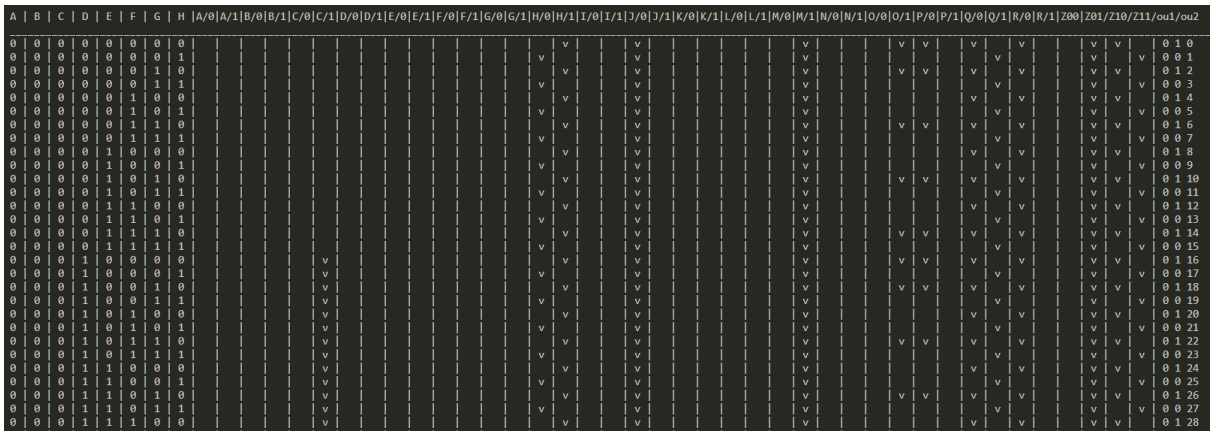
1. Melakukan deklarasi tiap-tiap gerbang logika yang digunakan pada rangkaian ke dalam sebuah fungsi, hal ini dilakukan agar memudahkan pemanggilan tiap gerbang logika jika ada perubahan rangkaian. Pada kasus ini pembuatan fungsi pada gerbang digambarkan pada Gambar. 4, deklarasi dilakukan pada gerbang AND, OR, NAND, NOR dan NOT.

```
/* Fungsi untuk Gate */

int andGate (int a, int b) { // fungsi gerbang and
    return a && b;
}

int orGate (int a, int b) { // fungsi gerbang or
    return a || b;
}

int nandGate (int a, int b) { // fungsi gerbang nand
    return !(a&&b);
}

int norGate(int a, int b){ // fungsi gerbang nor
    return !(a||b);
}

int notGate(int a){ // fungsi gerbang not
    return !a;
}

/* End of Fungsi untuk Gate */
```

Gambar 4. Deklarasi Gerbang Logika

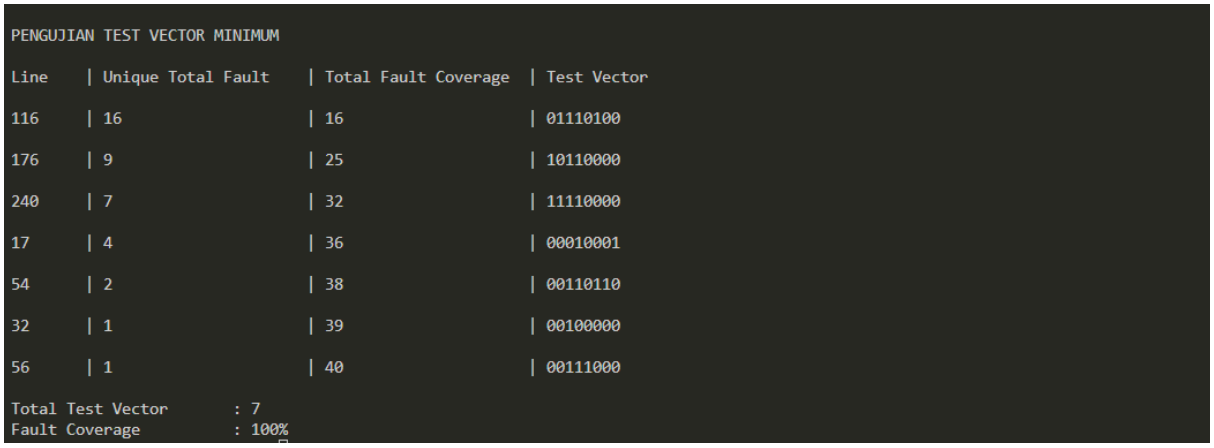
2. Melakukan iterasi nilai 0 dan 1 pada input yang nantinya akan digunakan sebagai test vector, iterasi yang dilakukan yaitu sebanyak 2^n (n untuk banyak input) pada kasus ini, yaitu sebanyak 2^8 atau sebanyak 256, yang mana 8 ini merupakan jumlah input pada rangkaian yang akan diujikan.
3. Melakukan iterasi per node pada rangkaian yang akan diujikan, iterasi ini bertujuan untuk menentukan *stuck-at-fault* pada tiap-tiap node. Iterasi dilakukan sebanyak dua kali jumlah node, pada kasus ini yaitu 40 kali sesuai dengan jumlah *stuck-at-fault* pada seluruh node yang diujikan, setiap dua kali iterasi yang dilakukan akan menginjeksikan nilai 0 dan 1 pada sebuah node, hal ini direpresentasikan pada formula arrayInput [i/2] = i%2.
4. Melakukan injeksi nilai fault pada rangkaian, injeksi fault dalam rangkaian ini bertujuan untuk menemukan nilai *stuck-at-fault* yang dapat teruji pada tiap-tiap node. Proses injeksi ini dilakukan dengan cara memberikan nilai fault yang telah tersimpan dalam array ke tiap-tiap node agar dapat dioperasikan hingga output rangkaian.
5. Setelah melakukan injeksi Fault pada Rangkaian maka akan dilakukan pengecekan nilai output pada *good circuit* (output pada rangkaian sebelum diinjeksi fault) dengan output pada

rangkaian yang telah diinjeksi oleh fault. Jika didapatkan kedua output yaitu Z0 dan Z1 pada *good circuit* memiliki hasil yang sama dengan Z0 dan Z1 pada *bad circuit* (rangkaian yang telah diinjeksi oleh fault) maka fault tidak dapat dideteksi, namun jika salah satu atau kedua output pada *good circuit* dan *bad circuit* berbeda maka fault dapat dideteksi. Setelah melakukan hasil pengecekan dari seluruh node, maka akan didapatkan hasil *stuck-at-fault* dari tiap-tiap node yang dapat dideteksi seperti yang digambarkan pada Gambar 5.



Gambar 5. Hasil Stuck-at-Fault

6. Setelah mendapatkan nilai *stuck-at-fault* pada seluruh node, maka akan dilakukan pencarian nilai test vector minimum dan fault coverage. Proses mencari test vector minimum ini dilakukan dengan cara mencari sebuah test vector pada suatu baris yang memiliki hasil pengujian *stuck-at-fault* unik terbanyak lalu disimpan dalam sebuah elemen array sepanjang n elemen data (n merupakan nilai untuk jumlah total *stuck-at-fault* pada rangkaian) pada kasus ini yaitu 40. Jika terdapat elemen array yang belum terisi maka akan dilakukan proses pencarian test vector kembali pada suatu baris yang memiliki hasil pengujian *stuck-at-fault* unik terbanyak berikutnya lalu disimpan ke dalam array, jika *stuck-at-fault* memiliki nilai yang sama dengan nilai *stuck-at-fault* pada test vector berikutnya, maka nilai tersebut akan diabaikan, dan jika berbeda maka nilai tersebut akan disimpan dalam sebuah elemen array yang masih kosong, hal tersebut akan dilakukan secara berulang hingga elemen pada array semua terisi.



```
PENGUJIAN TEST VECTOR MINIMUM

Line    | Unique Total Fault    | Total Fault Coverage  | Test Vector

116     | 16                    | 16                    | 01110100

176     | 9                     | 25                    | 10110000

240     | 7                     | 32                    | 11110000

17      | 4                     | 36                    | 00010001

54      | 2                     | 38                    | 00110110

32      | 1                     | 39                    | 00100000

56      | 1                     | 40                    | 00111000

Total Test Vector       : 7
Fault Coverage          : 100%
```

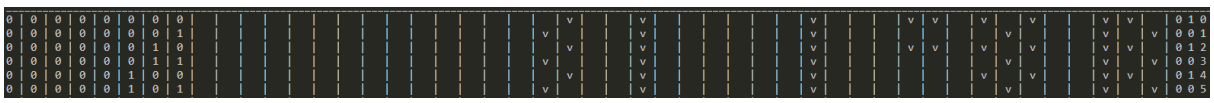
Gambar 6. Hasil Pengujian Test Vector Minimum

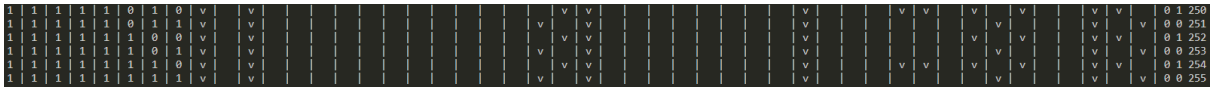
Pada Gambar 6, Line merupakan baris dimana terdapat test vector yang memiliki nilai *stuck-at-fault* unik terbanyak, Unique of Fault merupakan total dari *stuck-at-fault* unik dalam suatu test vector yang dapat dideteksi, Total Fault Coverage merupakan akumulasi dari banyaknya *stuck-at-fault* yang dapat terdeteksi, atau dengan kata lain Total Fault Coverage ini merupakan total dari elemen array yang telah terisi. Dari Gambar 6, dapat dijelaskan bahwa Total Fault Coverage pada baris terakhir telah mencapai 40, yang mana 40 ini merupakan total dari nilai *stuck-at-fault* pada node keseluruhan untuk rangkaian pada Gambar 3, hal ini dapat diartikan bahwa seluruh *stuck-at-fault* dapat terdeteksi dengan menggunakan 7 test vector dan karena elemen array telah terisi semua maka fault coverage yang didapatkan pun maksimum yaitu 100%.

4.3 Verifikasi Sistem

Verifikasi sistem merupakan sebuah proses yang dilakukan untuk membangun sebuah kebenaran, akurasi dan validitas sistem. Akan dilakukan 2 verifikasi pada sistem pengujian kali ini, yaitu verifikasi untuk total pola pengujian dan verifikasi pada test vector dan fault coverage maksimum.

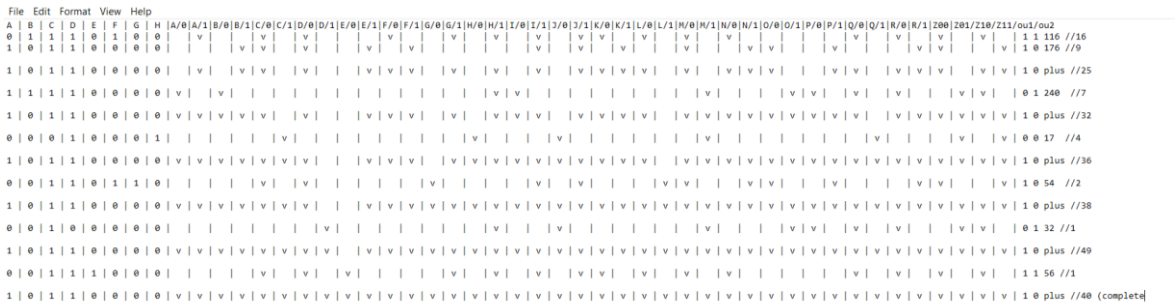
1. Pada metode exhaustive, pola pengujian dilakukan dengan membangun test vector sebanyak 2^n, yang mana nilai n merupakan total dari input suatu rangkaian. Untuk rangkaian pada Gambar 3 memiliki input sebanyak 8, yang mana total dari pola pengujian yang didapatkan yaitu sebanyak 2^8 pengujian atau sama dengan 256 pola uji, hal ini dibuktikan pada Gambar 7, bahwasanya pengujian yang telah dilakukan yaitu sebanyak 256 pengujian yang diawali oleh test vector 00000000 dan diakhir i oleh test vector 11111111.





Gambar 7. Jumlah Pola Uji

2. Verifikasi pada test vector minimum dan fault coverage dilakukan dengan memberikan pola pengujian yang sama untuk rangkaian Gambar 5 dengan test vector yang didapatkan pada Gambar 6 yaitu, 01110100, 10110000, 11110000, 00010001, 00110110, 00100000 dan 00111000.



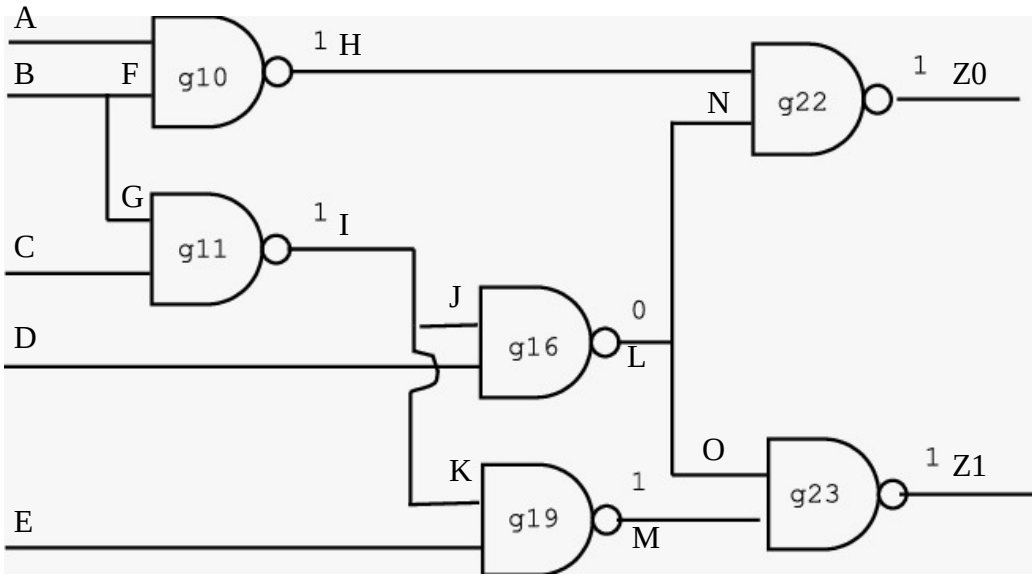
Gambar 8. Verifikasi Test Vector dan Fault Coverage

Pada Gambar 8, dijabarkan bahwa inisialisasi pertama test vector, yaitu 01110100 pada baris 116 mendapatkan total *stuck-at-fault* unik sebanyak 16, dan test vector kedua, yaitu 10110000 dengan total *stuck-at-fault* yaitu 15 dan memiliki nilai *stuck-at-fault* unik (nilai *stuck-at-fault* yang berbeda dengan nilai *stuck-at-fault* pada test vector sebelumnya) sebanyak 9, index plus pada Gambar 8 merupakan total test vector gabungan dari test vector ke n sampai n+1, yang mana dari kedua test vector ini telah mampu mendeteksi sebanyak 25 *stuck-at-fault* dari total keseluruhan (40 *stuck-at-fault).* Test vector ketiga, yaitu 11110000 yang mana test vector ini menghasilkan sebuah deteksi *stuck-at-fault* sebanyak 11 dan memiliki nilai *stuck-at-fault* unik sebanyak 7, maka total *stuck-at-fault* yang dapat terdeteksi dari ketiga test vector yaitu 32 (25 + 7) *stuck-at-fault*. Test vector keempat, yaitu 00010001 dengan total *stuck-at-fault* sebanyak 7 dan *stuck-at-fault* unik sebanyak 4 maka total *stuck-at-fault* dari test vector sebelumnya ditambahkan total *stuck-at-fault* unik dari test vector keempat adalah 36 *stuck-at-fault*. Test vector kelima, yaitu 00110110 dengan total *stuck-at-fault* sebanyak 13 dan *stuck-at-fault* unik sebanyak 2 maka total dari *stuck-at-fault* dari test vector sebelumnya ditambahkan total *stuck-at-fault* unik dari test vector kelima adalah 38 *stuck-at-fault*. Test vector keenam, yaitu 00100000 dengan total *stuck-at-fault* sebanyak 10 dan *stuck-at-fault* unik sebanyak 1 maka total dari *stuck-at-fault* dari test vector sebelumnya ditambahkan dengan total *stuck-at-fault* unik dari test vector keenam

adalah 39 *stuck-at-fault*. Test vector ketujuh, yaitu 00111000 dengan total *stuck-at-fault* sebanyak 15 dan *stuck-at-fault* unik sebanyak 1 maka total dari *stuck-at-fault* dari test vector sebelumnya ditambahkan dengan total *stuck-at-fault* unik dari test vector ketujuh adalah 40 *stuck-at-fault.* Dari hal ini dapat dijelaskan bahwa dengan menggunakan 7 test vector yang sesuai dengan Gambar 6 akan didapatkan nilai *stuck-at-fault* pada setiap node, yang berarti jika setiap node terdeteksi maka akan menghasilkan nilai persentase fault coverage sebesar 100% dan terbukti.

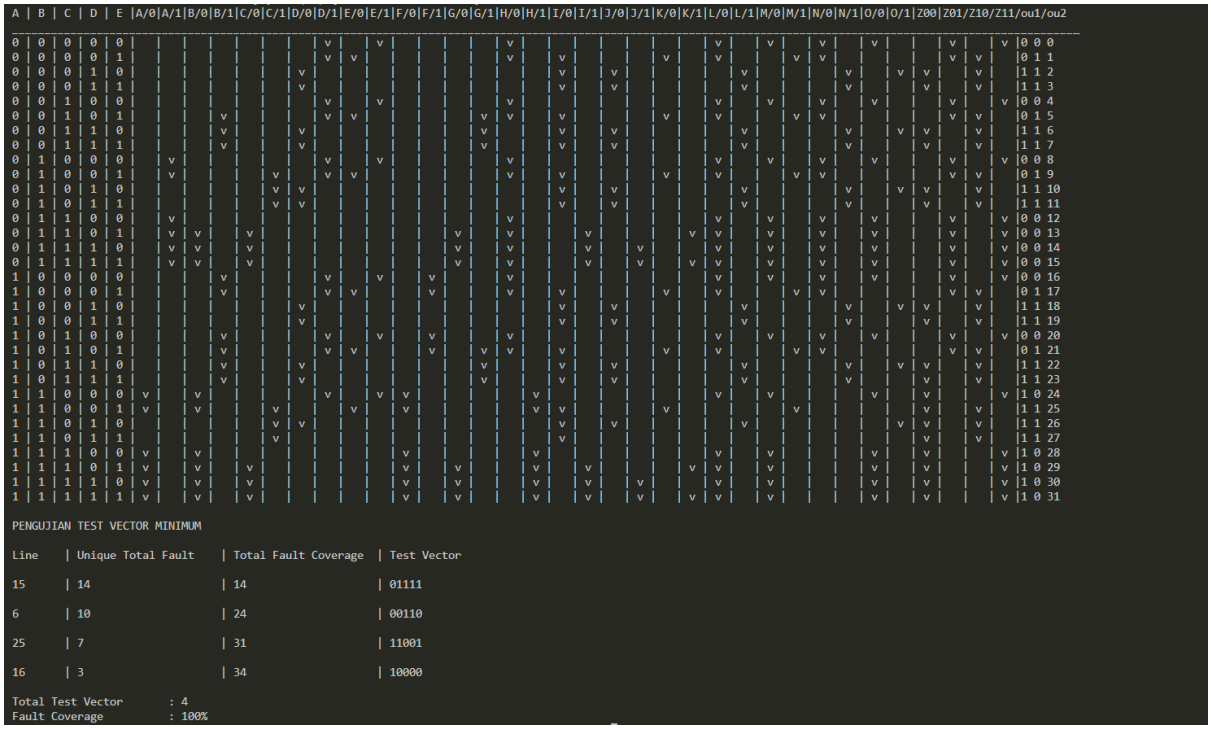
4.4 Verifikasi dengan Rangkaian ISCAS'85 C17

Pada uji verifikasi kali ini akan dilakukan juga pengecekan pada rangkaian ISCAS'85 C17 seperti yang digambarkan pada Gambar 9.



Gambar 9. Rangkaian ISCAS'85 C17

Pada Gambar 9 untuk rangkaian ISCAS'85 C17 terdapat 5 input, yaitu A, B, C, D dan E dan 2 output, yaitu Z0 dan Z1. Dengan menggunakan algoritma program yang digambarkan pada Gambar 2 serta membuat program pengecekan test vector minimum secara modular maka didapatkan pola pengujian dengan metode exhaustive sebanyak 2^5 atau 32 (nilai 5 direpresentasikan sebagai jumlah input) dengan jumlah node sebanyak 17 buah yang mana setiap node akan memiliki 2 buah *stuck at fault* yaitu *stuck-at-0* dan *stuck-at*-1, yang mana hal ini akan didapatkan pengujian *stuck-at-fault* sebanyak 34. Dari hasil pengujian yang telah dilakukan pada Rangkaian Gambar 9, didapatkan persentase *fault coverage* sebesar 100% dengan test vector minimum sebanyak 4 yaitu pada pola pengujian 0111, 00110, 11001 dan 1000, hal ini digambarkan pada Gambar 10.



```
A | B | C | D | E |A/0|A/1|B/0|B/1|C/0|C/1|D/0|D/1|E/0|E/1|F/0|F/1|G/0|G/1|H/0|H/1|I/0|I/1|J/0|J/1|K/0|K/1|L/0|L/1|M/0|M/1|N/0|N/1|O/0|O/1|Z00|Z01|Z10|Z11|ou1|ou2
```

PENGUJIAN TEST VECTOR MINIMUM

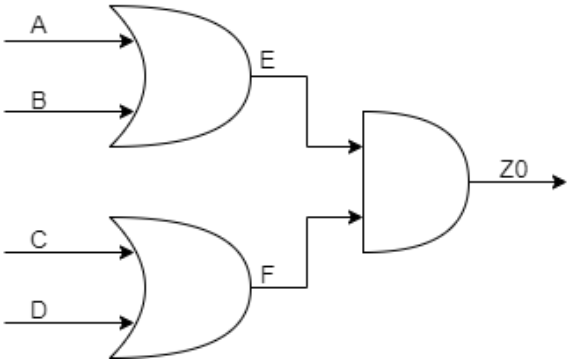
Line	Unique Total Fault	Total Fault Coverage	Test Vector
15	14	14	01111
6	10	24	00110
25	7	31	11001
16	3	34	10000

Total Test Vector : 4
Fault Coverage : 100%

Gambar 10. Pengujian pada Rangkaian ISCAS'85 C17

4.5 Verifikasi dengan Rangkaian Tree Structure

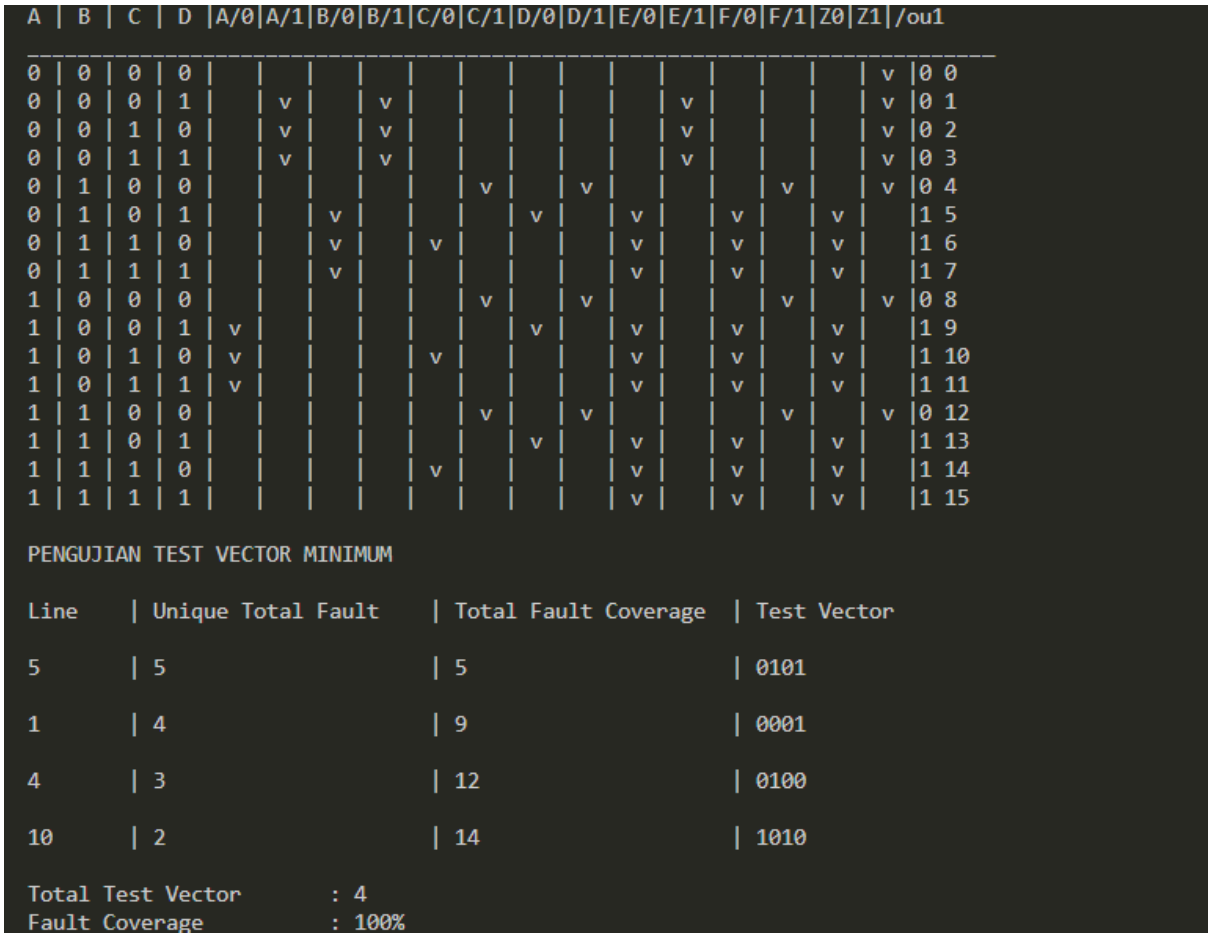
Pada uji verifikasi program ATPG kedua dilakukan dengan menggunakan rangkaian tree structure yang mana pada rangkaian ini didesain sederhana tanpa menggunakan percabangan dan hanya memiliki 4 input dan 1 output seperti yang digambarkan pada Gambar 11.



Gambar 11. Rangkaian Tree Structure

Dengan menggunakan algoritma program yang digambarkan pada Gambar 2 dan dengan menerapkan metode exhaustive maka akan didapatkan pola pengujian sebanyak 2^4 atau sebanyak 16 yang mana nilai 4 merupakan jumlah input pada rangkaian Gambar 16 dengan jumlah node sebanyak 7 (A, B, C, D, E, F dan Z0). Setiap node akan memiliki 2 buah *stuck-at-fault,* yaitu *stuck-at-0* dan *stuck-at-1,* pada kasus pengujian kali ini jumlah *stuck-at-fault* yang didapatkan adalah 14 (2 kali jumlah node). Pada Gambar 12 merupakan hasil running

program dari rangkaian Gambar 11 yang mana didapatkan *fault coverage* sebesar 100% dengan test vector minimum sebanyak 4 yaitu, 0101, 0001, 0100 dan 1010.



A	B	C	D	A/0	A/1	B/0	B/1	C/0	C/1	D/0	D/1	E/0	E/1	F/0	F/1	Z0	Z1	/ou1
0	0	0	0													v		0 0
0	0	0	1		v		v						v				v	0 1
0	0	1	0		v		v						v				v	0 2
0	0	1	1		v		v						v				v	0 3
0	1	0	0						v		v				v		v	0 4
0	1	0	1			v				v		v		v		v		1 5
0	1	1	0			v		v				v		v		v		1 6
0	1	1	1			v						v		v		v		1 7
1	0	0	0						v		v				v		v	0 8
1	0	0	1	v						v		v		v		v		1 9
1	0	1	0	v				v				v		v		v		1 10
1	0	1	1	v								v		v		v		1 11
1	1	0	0						v		v				v		v	0 12
1	1	0	1							v		v		v		v		1 13
1	1	1	0					v				v		v		v		1 14
1	1	1	1									v		v		v		1 15

PENGUJIAN TEST VECTOR MINIMUM

Line	Unique Total Fault	Total Fault Coverage	Test Vector
5	5	5	0101
1	4	9	0001
4	3	12	0100
10	2	14	1010

Total Test Vector : 4
Fault Coverage : 100%

Gambar 12. Pengujian Rangkaian Tree Structure

BAB V
KESIMPULAN

5.1. Kesimpulan

Berdasarkan hasil perancangan, pengujian, dan verifikasi sistem maka didapatkan beberapa kesimpulan yang merupakan hasil dari keseluruhan proses pengerjaan tugas besar ini, sebagai berikut:

1. Metode exhaustive bekerja dengan baik untuk rangkaian dengan input <15, dan berhasil mempropagasikan nilai fault sampai ke output.
2. Test vector minimum dan fault coverage maksimum dapat dihasilkan dengan cara membuat elemen array penampung sebanyak n elemen (nilai n merupakan nilai total *stuck-at-fault* yang terdeteksi oleh program), yang mana elemen array ini akan diisi oleh total *stuck-at-fault* terbanyak pada suatu test vector dan hal ini akan dilakukan berulang sampai elemen array terisi penuh, jika hasil *stuck-at-fault* antara elemen n+1 dan n sama maka akan diabaikan guna menghindari nilai ganda. Ketika elemen array terisi penuh maka dapat disimpulkan bahwa *stuck-at-fault* pada semua node dapat terdeteksi dan menghasilkan *fault coverage* 100%.
3. Dengan menggunakan verifikasi rangkaian ISCAS'85 C17 didapatkan *fault coverage* 100% dengan test vector minimum sebanyak 4, hal ini membuktikan bahwa program berhasil mempropagasikan nilai fault dan menghasilkan test vector minimum dari rangkaian ISCAS'85 C17.
4. Dengan menggunakan verifikasi rangkaian Tree Structure juga didapatkan *fault coverage* 100% (setiap node berhasil mempropagasikan nilai *stuck-at*-fault) dengan test vector minimum sebanyak 4, hal ini mengindikasikan bahwa program dapat menguji rangkaian tanpa percabangan dan meneruskan nilai fault hingga output.

5.2. Saran

Dari hasil perancangan, pengujian, dan verifikasi sistem maka didapatkan beberapa saran yang dapat bermanfaat bagi peneliti selanjutnya, yaitu sebagai berikut:

1. Peneliti selanjutnya diharapkan mampu untuk menerapkan algoritma ATPG lain untuk rangkaian dengan kompleksitas yang cukup tinggi.

DAFTAR PUSTAKA

Rahmatullah, Griffani, dkk. 2019. *Pengujian Rangkaian Sekuensial Iscas'89 S641 Menggunakan Synopsys Tetramax*. ISU TEKNOLOGI STT MANDALA 14[2] pp.100-105

Vishwani D. Agrawal.(2001). Essentials Of Electronic Testing For Digital, Memory And Mixed-Signal VLSI Circuit

LAMPIRAN PROGRAM