Penerapan Greedy dan DFS dalam Pemecahan Solusi K-Map

Sri Handika Utami / 13508006
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
if18006@students.if.itb.ac.id

Abstrak—Dalam menyederhanakan rangkaian logika, biasanya para engineer menggunakan Karnough Map (K-Map). K-Map dibentuk dari sebuah tebel kebenaran yang biasanya disebut sebagai *state table*. Tidak hanya langkah pencarian solusinya, penempatan objek yang berasal dari *state table* ke K-Map juga memiliki cara yang unik. Pengambilan solusi di K-Map adalah dengan mendapatkan area dengan ukuran 2^n terbesar pada K-Map. Area-area tersebut kemudian digabungkan menjadi solusi. Semakin luas cakupan area, maka akan lebih sedikit area yang harus digabungkan. Pencarian area solusi K-Map dapat dilakukan dengan memanfaatkan algoritma DFS (*Depth First Search*) yang memanfaatkan prinsip ketetanggan dalam penulisan K-Map. Untuk memilih area solusi terbesar, maka digunakan greedy. Area-area yang didapat dari algoritma tersebut kemudian d*egenerate* sehingga menghasilkan suatu persamaan logika.

Kata Kunci—K-Map, area solusi, DFS, Greedy.

I. PENDAHULUAN

Rangkaian digital merupakan rangkaian yang menggunakan dasar-dasar logika di dalam penerapannya. Rangkaian logika biasanya dibangun dengan menggunakan gerbang-gerbang logika.

Terdapat berbagai macam gerbang logika yang digunakan. Gerbang tersebut adalah gerbang AND, OR, NOT, NAND, NOR, XOR, dan XNOR. Gerbang-gerbang logika tersbut biasanya terdapat di dalam sebuah IC.

Hal yang dilakukan di dalam perancangan gerbang logika adalah mengkombinasikan gerbang logika yang ada dengan membuat persamaan logika untuk rangkaian gerbang logika tersebut. Langkah awal yang biasanya dilakukan para engineer adalah dengan membuat *Algorithm State Machine* (ASM) terlebih dahulu. Kemudian, ASM tersebut diterjemahkan ke dalam *state table*. Semua fungsi yang bernilai true (biasanya dilambangkan dengan 1) dari *state table* tersebut dihubungkan dengan fungsi OR untuk mendapatkan persamaan logika untuk rangkaian kombinasional dari rancangan tersebut.

Semakin sedikit gerbang logika yang digunakan maka akan semakin baik. Alasan dari pernyataan tersebut adalah dengan semakun sedikitnya gerbang logika yang digunakan di dalam rangkaian, maka komponen yang digunakan akan semakin sedikit sehingga biaya yang digunakan dalam pembangunan rangkaian logika akan semakin kecil.

Persamaan logika yang dihasilkan dari menerjemahkan *state table* biasanya bukanlah persamaan logika yang paling sederhana. Oleh karena itu, dibutuhkan metode yang digunakan untuk menyederhanakan persamaan logika tersebut.

Terdapat dua metode yang digunakan untuk menyederhanakan suatu rangkaian logika. Metode tersebut adalah penyederhaan dengan menggunakan aljabar boolean dan penyederhanaan dengan menggunakan Karnough Map (K-Map). Cara penyederhanaan dengan menggunakan K-Map adalah metode yang akan dibahas pada makalah ini. Pemilihan tersebut disebabkan oleh cara penyederhanaan dengan menggunakan K-Map cenderung lebih sistematis jika dibandingkan dengan menggunakan aljabar boolean.

II. PENYEDERHANAAN DENGAN K-MAP

A. Pemetaan dari State Table ke K-Map

Misalkan terdapat *state tabel* sebagai berikut,

A	B	C	X
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Gambar 1. Contoh *State Table*

A,B, dan C merupakan masukkan atau yang lebih populer dengan istilah input pada rangkaian logika. X adalah nilai fungsi atau keluaran yang dihasilkan untuk input tertentu. Maka *state table* tersebut dipetakan pada K-Map berikut,

C/AB	00	01	11	10
0	1	1	0	1
1	1	0	0	1

Dalam memetakan *state table* ke K-Map terdapat beberapa aturan yang harus dipenuhi.

- Jumlah *state* total yang terdapat pada K-Map adalah nilai perpangkatan dua dari variabel yang dimilikinya. Pada K-Map di atas terdapat 3 variabel sehingga terdapat 8 *state* yang harus digambarkan.
- Dalam penulisan baris atau kolom yang hanya terdiri dari dua keadaan, maka kondisi ' 0' (*false*) harus lebih dulu dari pada kondisi '1' (*true*). Sebagai contoh lihat kondisi untuk *state* C pada K-Map di atas (pada bidang vertikal).
- Dalam penulisan baris atau kolom yang terdiri dari empat keadaan maka bagian pertama hingga terkhir harus diisikan *state* berikut secara berurutan '00', '01', '11', '10'. Hal ini disebabkan karena keseluruhan state yang terdapat pada K-Map merupakan keadaan yang siklik. Aturan ini akan digunakan pada pencarian solusi yang akan dilakukan. Untuk lebih jelasnya hal ini akan dijelaskan pada penjelasan tentang pencarian solusi K-Map.

B. Pencarian Solusi pada K-Map

Solusi yang diharapkan didapatkan dari K-Map adalah solusi dengan persamaan yang paling sederhana. Namun, pada level aplikasi mungkin saja solusi yang dianggap paling sederhana itu bukan merupakan solusi yang paling murah. Akan tetapi, solusi yang paling sederhana pada pembahasan kali ini adalah solusi dengan penggunaan variabel dan operator logika yang paling sedikit.

Oleh karena itu pada K-Map, daerah yang diambil terlebih dahulu sebagai daerah solusi adalah daerah dengan cakupan yang paling luas. Untuk menjadikan suatu area sebagai area solusi terdapat aturan dalam pengambilan area tersebut.

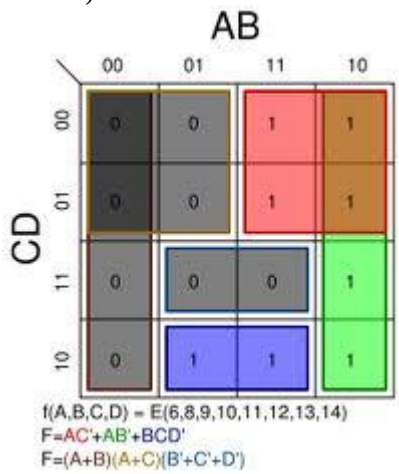
Area yang diambil haruslah merupakan daerah yang berupa persegi panjang atau persegi yang luas areanya 2^n. Masalah pencarian solusi K-Map kali ini hanya dibatasi untuk maksimal empat variabel saja. Dengan demikian, berarti luas maksimum K-Map yang akan kita jadikan permasalahan hanya dibatasi menjadi K-Map 4x4. Hal ini disebabkan oleh K-Map dengan jumlah variabel lebih dari empat akan sangat sulit digambarkan karena permasalahan definisi ketetanggaan yang unik pada K-Map.

Terdapat dua cara penemuan solusi pada K-Map. Cara pertama disebut dengan *Sum of Product* (SOP) dan cara lainnya disebut dengan *Product of Sum* (POS). SOP adalah cara pencarian solusi dengan menemukan bit 1 yang berdekatan dan seluruh hasil penemuan kumpulan bit 1 tersebut disebut dengan SOP. Pada penemuan dengan SOP maka seluruh pengelompokkan bit '1' digabungkan dengan menggunakan operator logika OR.

Pada gambar 2, solusi yang akan ditemukan dengan menggunakan SOP adalah AC`, AB`, dan BCD`. Solusi-solusi lokal tersebut digabungkan sehingga didapatkan solusi akhir minimal untuk K-Map tersebut adalah f = AC` + AB` + BCD`.

Pencarian solusi dengan menggunakan POS adalah dengan mencari solusi yang paling sederhana untuk bit '0' (seperti pencarian bit '1' pada SOP). Hasil pencarian tersebut merupakan kebalikan dari solusi akhir dari K-Map tersebut. Oleh karena itu untuk menemukan solusi akhir maka solusi yang ditemukan harus diberi *not* sebelumnya terlebih dahulu. Namun, terdapat cara untuk mempermudah langkah tersebut. Cara yang ditempuh untuk mendapatkan solusi dari kumpulan bit '0' secara langsung adalah dengan jalan melakukan operasi OR (+) pada setiap elemen yang didapatkan pada solusi local dan setiap solusi local digabungkan dengan jalan melakuakan operasi AND (.) pada seluruh solusi local tersebut.

Jika diterapkan pada contoh K-Map yang terdapat pada gambar 2, maka didapatkan solusi local (A + C), (A+B), dan (B` + C` + D` ). Sehingga didapatkan solusi akhir dari seluruh solusi local tersebut berupa f = (A+C) (A+B) (B`+C`+D`).



Gambar 2. Pencarian Solusi K-Map

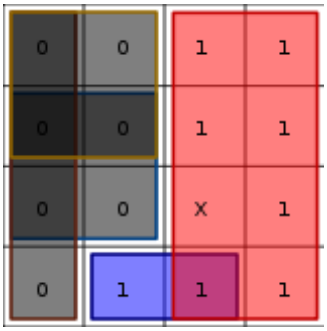
C. Don't Care Pada K-Map

Dalam pembuatan sebuah rangkaian digital terdapat beberapa *state* yang diabaikan. Misalnya dalam rangkaian BCD Counter, pada rangkaian ini terdapat empat masukkan dan empat keluaran untuk *next state* serta satu keluaran sebagai carry. Rangkaian ini biasanya digunakan untuk melakukan *counter*. Karena rangkaian ini memiliki empat variabel, maka akan terbentuk 16 *state* dari empat variabel tersebut. Namun, di dalam

melakukan counter, biasanya hanya akan digunakan 10 *state* yaitu *state* 0 (‘0000’) – 9 (‘1001’). Saat berada di *state* 9(‘1001’), maka rangkaian akan kembali lagi ke *state* 0 (‘0000’) dan memberikan keluaran ‘1’ pada *carry*. Oleh karena itu untuk masukkan ‘1010’, ‘1011’, dan seterusnya hingga ‘1111’ keluarannya akan diabaikan.

State yang keluarannya diabaikan tersebut tentu saja harus diperhitungkan dan tidak boleh dikosongkan begitu saja. Oleh karena itu, pada K-Map terdapat satu keadaan lagi yang dapat ditambahkan selain ‘0’ dan ‘1’ yaitu *Don’t Care* (biasanya dilambangkan dengan ‘X’ atau ‘Ø’).

Di dalam pencarian solusi, *Don’t care* dapat dimasukkan atau tidak di dalam perhitungan. Sifat *don’t care* dalam pencarian solusi adalah netral. Jadi, *state* yang berisi *Don’t care* dapat dianggap sebagai ‘0’ atau ‘1’ di dalam K-Map yang sama. Apabila *state* yang berisi *don’t care*, apabila dimasukkan ke dalam solusi dapat menyederhanakan solusi maka *state* tersebut dapat dimasukkan ke dalam solusi.



Gambar 3. K-Map dengan *Don’t Care*

Pada gambar 3 di atas, akan lebih baik jika ‘X’ pada *state* ABCD dianggap sebagai ‘1’ jika solusi yang digunakan adalah SOP sehinga solusi yang ditemukan akan jauh lebih sederhana jika dibandingkan dengan jika X dianggap sebagai ‘0’. Solusi yang ditemukan jika menggunakan SOP adalah A+BC.notD. Namun, jika pencarian solusi dilakukan dengan POS maka akan lebih baik jika *state* ABCD yang diisi oleh ‘X’ tidak dianggap. Solusi tersebut akan menghasilkan solusi paling sederhana yaitu (A+C) (A+notD) (A+B). Namun, jika dibandingkan maka solusi yang didapatkan dengan metode POS dan SOP tidak akan menghasilkan keluaran yang ekivalen.

III. PENCARIAN SOLUSI K-MAP DENGAN GREEDY

Pada makalah kali ini pencarian solusi K-Map yang dibahas adalah pencarian solusi K-Map dengan menggunakan algoritma Greedy. Pencarian solusi tersebut akan dibahas pada uraian di bawah ini.

A. Sekilas tentang Greedy dan DFS

Greedy merupakan algoritma yang biasanya digunakan untuk mengatasi masalah yang berkaitan dengan pencarian solusi optimal. Solusi yang diinginkan tersebut dapat berupa solusi yang mengharapkan nilai maksimal atau solusi yang mengharapkan nilai minimal.

Algoritma greedy biasanya mencari solusi langkah perlangkah. Dari setiap langkah tersebut ditemukan sebuah solusi optimal yang biasanya disebut optimum local. Nilai optimum lokal tersebut akan dibandingkan untuk mendapatkan nilai optimum yang berlaku umum untuk permasalahan yang ada.

Solusi optimal yang ditemukan dengan menggunakan algoritma greedy, biasanya disebut dengan optimum lokal, mungkin bukan merupakan solusi yang benar-benar optimal. Namun, solusi ini bisa dijadikan hampiran dalam menemukan solusi yang optimal.

DFS (*Depth First Serach*) merupakan algoritma encarian yang melakukan pencarian secara mendalam. Langkah pencarian yang dilakukan oleh algoritma ini adalah sebagai berikut.

- Kunjungi simpul v
- Kunjungi simul w yang bertetangga dengan simpul v
- Ulangi DFS mulai dari simpul w.
- Ketika mencapai simpul u sedemikian sehingga semua simpul yang bertetangga dengannya telah dikunjungi, pencarian dirunut-balik (backtrack) ke simpul terakhir yang dikunjungi sebelumnya dan mempunyai simpul w yang belum dikunjungi.
- Pencarian berakhir bila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

B. Solusi K-Map dengan Menggunakan Greedy dan DFS

Pencarian solusi untuk K-Map biasanya dilakukan dengan mencari solusi untuk area solusi yang paling luas terlebih dahulu. Apabila masih ada area yang memuat angka satu yang belum dimasukkan ke dalam area solusi, maka pencarian solusi dilakukan untuk area yang belum dijelajahi tersebut.

Jika diamati dengan lebih seksama lagi, maka algoritma untuk mencari solusi K-Map tidak murni algoritma greedy, melainkan juga menggunakan algoritma *Depth First Search* (DFS) dalam pencarian areanya. Area yang dicari tersebut diusahakan seluas mungkin sehingga dalam pengambilan area tersebut digunakan algoritma greedy.

Pada pencarian solusi K-Map tentunya dibutuhkan sebuah tipe yang menggambarkan K-Map tersebut. K-Map dipandang sebagai sekumpulan *cell* dengan ukuran tertentu yang setiap *cell* tersebut didefinisikan sebagai sebuah tipe bentukkan yang disebut K-Map. Setiap *cell* K-Map memiliki atribut NOT yang merupakan *array of character* yang menyatakan apakah variabel apa saja yang bernilai FALSE untuk *state* atau *cell* tersebut. Sedangkan variabel yang bernilai true dianggap sebagai variabel yang tidak termasuk ke dalam variabel yang ada di *array* NOT. Variabel merupakan *array of character* yang menyatakan seluruh variabel yang terlibat di dalam penarian yang dijadikan sebagai variabel global. Selain

NOT, *cell* juga memiliki atribut NILAI yang bertipe karakter. NILAI digunakan untuk menyatakan isi dari state tersebut apakah 1, 0, atau X (*Don't Care*). Pada *cell* juga terdapat FLAG yang menyatakan apakah *cell* tersebut telah dimasukkan ke dalam solusi atau belum. Berikut deklarasi untuk struktur *cell* tersebut.

```
typedef struct
{
   char Nilai;
   int Flag;
   int Not[4]; //Jumlah maksimum variabel
yang ada adalah 4 (batasan)
}Cell;
```

Variabel global yang digunakan selain varibel dan JumlahVariabel adalah baris dan kolom yang menyatakan jumlah baris dan kolom yang digunakan di dalam K-Map. Jumlah baris dan kolom tersebut nilainya bergantung pada nilai yang dimiliki oleh variabel yang menyatakan jumlah variabel. Jika jumlah variabel dua maka hanya digunakan satu baris dan dua kolom, jika jumlah variabel dua maka akan digunakan dua baris dan dua kolom, jika jumlah variabel adalah tiga, maka akan digunakan dua baris dan empat kolom, dan apabila jumlah variabel empat, maka akan digunakan empat baris dan empat kolom.

Langkah pertama yang dilakukan pada program yang dibuat adalah menerima masukkan untuk jumlah variabel yang digunakan. Dari jumlah variabel, maka dilakukan inputan untuk variabel yang digunakan apa saja (untuk dimasukkan ke variabel). Selanjutnya dilakukan penerimaan masukkan untuk input.

```
int Variabel[4];
int JumlahVariabel;
int Baris;
int Kolom;
int Metode; //POS atau SOP
char* Solusi;
Cell KMap[4][4];

int main()
{
  scanf("%d", JumlahVariabel);
  InisialisasiBarisKolom();
  InputCell();
  Scanf("%d", Metode); //0 untuk SOP
                       //1 untuk POS
```

Selanjutnya dilakukan pencarian solusi. Dilakukan iterasi dari cell[0,0] hingga cell[Baris,Kolom] apabila terdapat cell yang nilainya sesuai dengan yang dicari dan cell tersebut sebelumnya bukan merupakan bagian dari solusi, maka dilakukan pencarian solusi lokal. Pencarian solusi merupakan sebuah fungsi rekursif yang dilakukan berdasarkan metode pencarian. Jadi apabila metode pencarian yang digunakan adalah POS maka fungsi yang digunakan adalah POS, namun jika fungsi yang digunakan adalah SOP, maka fungsi yang akan digunakan untuk pencarian solusi adalah fungsi SOP.

Prinsip pencarian solusi pada fungsi ini adalah melakukan pencarian dengan area dengan menggunakan DFS (*Depth First Search*). DFS merupakan algoritma yang melakukan pencarian solusi yang mendalam terlebih dahulu.

Pada awal fungsi, dilakukan pengecekkan apakah *cell* yang dicek merupakan *cell* yang dapat dimasukkan ke dalam solusi atau tidak. Jika *cell* bukan merupakan *cell* yang dapat dimasukkan ke dalam solusi, maka langsung dikembalikan 0. Namun, apabila *cell* masih bisa dipertimbangkan untuk dijadikan solusi, contohnya 'X' (*Don't Care*), pencarian solusi dilanjutkan ke langkah selanjutnya.

Pada tahap selanjutnya dilakukan pencarian untuk *cell* di sebelah kanan *cell* yang dicari. Sebelah kanan di sini adalah apabila *cell* berada pada kolom terakhir, maka sebelah kanan *cell* adalah kolom awal dengan baris yang sama dengan *cell*, namun apabila *cell* tidak berada di kolom terakhir, maka sebelah kanan yang dimaksudkan adalah *cell* yang berada pada kolom sesudah *cell* sekarang dan berada pada baris yang sama. Pada *cell* kanan tersebut dilakukan pencarian dengan metode yang sama (proses rekursif).

Setelah dilakukan pengecekkan untuk *cell*, di sebelah kanan dan di dapatkan luas area solusi yang di dapatkan pada *cell* di sebelah kanan tersebut. Selanjutnya dengan proses yang hampir mirip dilakukan pencarian untuk *cell* yang berada di sebelah kiri, atas, dan bawah. Pada tahap selanjutnya, dengan memanfaatkan fungsi maksimum, maka ditinjau solusi lokal (kanan, kiri, atas, atau bawah) yang menghasilkan luas area yang terbesar. Selanjutnya dicek apakah bagian tersebut bisa dimasukkan ke dalam solusi atau tidak. Jika bagian tersebut tidak bisa dimasukkan ke dalam area solusi, maka dilakukan pengecekkan apakah *cell* yang sedang ditinjau merupakan *cell* yang bernilai 'X' (*Don't Care*) atau tidak. Jika ya, maka fungsi akan menghasilkan keluaran 0, artinya *cell* tersebut gagal dianggap sebagai *cell* solusi. Jika bukan 'X', maka *cell* tersebut merupakan solusi yang berdiri sendiri, maksudnya, solusi yang tidak memiliki tetangga yang juga merupakan solusi.

Berikut diperlihatkan *pseudocode* dari penjelasan di atas.

```
//prosedur yang digunakan untuk
pencarian SOP
//mengembalikan jumlah cell yang
menjadi cell solusi
//mengembalikan 0 jika bukan cell
solusi
int SOP(int baris, int kolom, Cell*
Area)
{
   //Kamus Lokal
   int Kanan, Kiri, Atas, Bawah;
   int Luas;
   int BagianSolusi;   //flag yang
menandakan kanan, kiri atas atau bawah
```

```
      Cell* AreaKanan, AreaKiri,
AreaAtas, Area Bawah;

   if (KMap[baris][kolom] == '1' ||
KMap[baris][kolom] == 'X')
   {
      //pengecekkan kotak di sebelah
kanan
      if (kolom < (Kolom-1))
        Kanan = SOP(baris, kolom + 1,
AreaKanan)
      else
        SOP(baris, 0, AreaKanan);

      //pengecekkan kotak di kiri
      if (kolom > 0)
        Kiri = SOP(baris, kolom + 1,
AreaKiri)
      else
        Kiri = SOP(baris, Kolom – 1,
AreaKiri

                  :
                Dst
                  :
                  :
      Luas = Maksimum(Kanan, Kiri,
Atas, Bawah, BagianSolusi);

     if(Luas == 0 &&
KMap[baris][kolom] == 'X')
     {
         return 0;
      }
      else if(Luas == 0 &&
KMap[baris][kolom] == '1')
      {
        pushCell(&KMap[baris][kolom],
Area);
         return 1;
      }
      else
      {
        pushCellbyBagian(
&BagianSolusi, Area);
        pushCell(&KMap[baris][kolom],
Area);
        return (Luas + 1);
      }
   }
   else return 0;}
```

Pseudocode di atas belum sempurna karena di dalam prosedur yang diperlihatkan pada *pseudocode*, belum dicantumkan langkah yang melakukan pengecekkan apakah *cell* yang dicek sudah dibangkitkan sebelumnya dan merupakan bapak dari *cell* yang sedang dicek sekarang. Jika ya, maka pencarian solusi dibatalkan, karena jika pencarian tidak dibatalkan maka akan terjadi perulangan secara terus menerus karena K-Map merupakan suatu tabel yang siklik, artinya cell terakhir dan cell awal merupakan cell yang bertetangga pada kolom atau baris yang sama.

Setelah keluar dari fungsi rekursif tersebut, maka dilakukan pembuatan solusi untuk *array of Cell* yang ditemukan dari solusi tersebut dengan menggunakan prosedur GenerateSolusi(char* solusi, Cell* Area). Prosedur GenerateSolusi akan mencari solusi untuk ArrayofCell yang ditemukan dan menggabungkan solusi tersebut denngan solusi sebelumnya.

C. Optimalitas Solusi yang Dihasilkan

Solusi di atas didapatkan dengan menggabungkan antara algoritma greedy dan DFS untuk fungsionalitas yang berbeda. Algoritma greedy digunakan untuk menentukan area yang akan dijadikan sebagai solusi lokal dengan prinsip area yang dijadikan solusi lokal adalah area yang memiliki luas area yang paling besar. Sedangkan algoritma DFS digunakan untuk mencari area solusi.

Algoritma DFS yang digabungkan dengan algoritma greedy kemungkinan besar dapat menghasilkan solusi yang optimal untuk permasalahan K-Map. Namun solusi yang dikembangkan pada makalah ini belum sempurna karena sebenarnya dalam penemuan daerah, luas area harus dibatasi sebesar perpangkatan dari 2. Hal ini harus dilakukan agar mendapatkan solusi yang valid. Pembatasan jumlah tersebut dapat diberikan pada prosedur POS atau SOP yang terdapat pada solusi yang diberikan sebelumnya.

IV. KESIMPULAN

Hasil penggabungan algoritma greedy dan DFS dapat memecahkan masalah pencarian solusi untuk K-Map. Beberapa langkah yang dijabarkan pada makalah ini harus dilengkapi lagi dengan langkah yang sudah dijelaskan pada penjabaran tentang pencarian solusi K-Map. Kompleksitas waktu untuk solusi yang ditawarkan adalah eksponensial. Namun, beberapa optimasi bisa dilakukan untuk mengurangi kompleksitas waktu tersebut.

REFERENSI

[1] Brown Stephen, Vranesic Avonko, *Fundamental of Digital Logic with VHDL Design*, Toronto : Dept. of Electrical and Computer Engineering, 2005, 2nd Edition

[2] http://www.informatika.org/~rinaldi/Stmik/2010-2011/ AlgoritmaGreedy(Bagian1).pdf, waktu akses : 1 Desember 2010, 10:05.

[3] http://www.informatika.org/~rinaldi/Stmik/2010-2011/ BFSdanDFS.pdf, waktu akses : 1 Desember 2010, 10:18.

[4] http://www.allaboutcircuits.com/vol_4/chpt_8/5.html, waktu akses : 1 Desember 2010, 10:20

[5] http://www.allaboutcircuits.com/vol_4/chpt_8/8.html, waktu akses : 1 Desember 2010, 10:25.

[6] http://en.wikipedia.org/wiki/Karnaugh_map, Waktu akses : 1 Desember 2010, 13:05.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 29 April 2010

Sri Handika Utami
13508006