Aplikasi Aljabar Boolean Dalam Sistem Pengaturan Lampu Lalu Lintas *Pelican Crossing*

Margaretha Olivia Haryono - 13521071
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13521071@std.stei.itb.ac.id

***Abstract*—Lampu lalu lintas adalah hal yang penting dalam mengatur keteraturan arus lalu lintas. *Pelican crossing* merupakan salah satu jenis lampu lalu lintas yang dapat menjadi solusi untuk menciptakan lingkungan penyeberangan jalan yang aman bagi pejalan kaki. Sistem *pelican* crossing ini memanfaatkan teori aljabar Boolean dalam pengaturannya. Makalah ini membahas implementasi teori aljabar Boolean dan gerbang logika beserta penyederhanaannya menggunakan Peta Karnaugh dalam merancang sistem lampu lalu lintas *pelican crossing*.**

***Keywords*—aljabar boolean, gerbang logika, lampu lalu lintas, pelican crossing.**

I. PENDAHULUAN

Lampu lalu lintas adalah alat pengatur lalu lintas yang bertujuan untuk mengendalikan arus lalu lintas kendaraan maupun pejalan kaki agar tercipta alur yang teratur. Dewasa ini, lampu lalu lintas semakin memegang peranan penting seiring dengan bertambahnya jumlah kendaraan bermotor. Salah satu jenis lampu lalu lintas yang sering ditemui yaitu lampu lalu lintas untuk penyeberang jalan atau yang dikenal dengan istilah *pelican crossing*.

Pelican crossing (Pedestrian Light Controlled Crossing) adalah jenis lampu lalu lintas penyeberangan yang dioperasikan oleh pejalan kaki dengan menekan tombol untuk menghentikan arus lalu lintas kendaraan. Sistem ini pertama kali diperkenalkan di jalan raya pada tahun 1969. Mekanisme penggunaan *pelican crossing* yaitu pejalan kaki menekan tombol untuk memberi sinyal ke lampu lalu lintas. Sinyal ini akan mengubah warna lampu menjadi merah yang mengisyaratkan pengemudi kendaraan untuk berhenti sehingga pejalan kaki dapat menyeberang jalan dengan aman. Setelah interval waktu tertentu, lampu lalu lintas akan secara otomatis kembali ke warna hijau sehingga kendaraan dapat berjalan kembali.

Gambar 1.1. Lampu lalu lintas
Sumber: https://www.wallpaperflare.com/

Sistem lampu lalu lintas *pelican crossing* ini adalah salah satu cara untuk meningkatkan keamanan bagi para pejalan kaki, khususnya di daerah yang ramai. Dalam pengaturannya, *pelican crossing* memanfaatkan teori aljabar Boolean untuk menginstruksikan lampu yang menyala pada waktu tertentu. Sistem tersebut dibangun dari rangkaian logika yang dibentuk dari gabungan berbagai gerbang logika. Perubahan warna lampu lalu lintas dipicu dari tombol yang ditekan oleh pejalan kaki yang dapat mengubah warna lampu hijau menjadi merah melalui rangkaian sistem tersebut. Dengan demikian, isyarat dari warna lampu yang menyala dapat diterima oleh pengguna jalan, baik pengemudi kendaraan maupun pejalan kaki, agar terjadi keteraturan dalam lalu lintas.

II. LANDASAN TEORI

A. Aljabar Boolean

Aljabar Boolean adalah aljabar logika yang nilai-nilai variabelnya adalah nilai kebenaran dan biasanya dilambangkan 1 untuk benar dan 0 untuk salah. Sistem aljabar ini ditemukan oleh George Boole pada tahun 1894 dalam buku yang diterbitkannya berjudul *The Laws of Thought*. Saat ini, aljabar Boolean menjadi dasar dari analisis rangkaian digital.

Misalkan B adalah himpunan yang didefinisikan pada dua operator biner, yaitu + dan ·, serta sebuah operator uner, yaitu ’. Misalkan juga 0 dan 1 adalah dua elemen yang berbeda dari B. Maka, tupel <B, +, ·, ’, 0, 1> disebut aljabar Boolean jika untuk setiap *a, b, c* ∈ *B* berlaku aksioma berikut:

1. Identitas
   (i) a + 0 = a
   (ii) a · 1 = a
2. Komutatif
   (i) a + b = b + a
   (ii) a · b = b . a
3. Distributif
   (i) a · (b + c) = (a · b) + (a · c)
   (ii) a + (b · c) = (a + b) · (a + c)
4. Komplemen
   Untuk setiap a ∈ B terdapat elemen unik a‘ ∈ B sehingga
   (i) a + a’ = 1
   (ii) a · a’ = 0

Aljabar Boolean memiliki beberapa hukum yang dapat dilihat pada tabel di bawah ini.

Hukum identitas: (i) a + 0 = a (ii) a · 1 = a	Hukum idempoten: (i) a + a = a (ii) a · a = a
Hukum komplemen: (i) a + a' = 1 (ii) aa' = 0	Hukum dominansi: (i) a · 0 = 0 (i) a + 1 = 1
Hukum involusi: (i) (a')' = a	Hukum penyerapan: (i) a + ab = a (ii) a(a + b) = a
Hukum komutatif: (i) a + b = b + a (ii) ab = ba	Hukum asosiatif: (i) a + (b + c) = (a + b) + c (ii) a (b c) = (a b) c
Hukum distributif: (i) a + (b c) = (a + b) (a + c) (ii) a (b + c) = a b + a c	Hukum De Morgan: (i) (a + b)' = a'b' (ii) (ab)' = a' + b'
Hukum 0/1 (i) 0' = 1 (ii) 1' = 0	

Tabel 2.1. Hukum-hukum aljabar Boolean

Ekspresi Boolean dapat ditulis dalam dua bentuk, yaitu:

1. *Sum of Product* (SOP)

   Bentuk SOP adalah penjumlahan dari hasil kali. Jika suatu suku dari fungsi SOP memiliki literal lengkap, maka suku tersebut disebut *minterm*. *Minterm* dibentuk dengan menyatakan setiap peubah yang bernilai 0 dalam bentuk komplemen dan yang bernilai 1 tanpa komplemen. *Minterm* memiliki lambang m atau notasi $\sum$.
2. *Product of Sum* (POS)

   Bentuk POS adalah perkalian dari hasil jumlah. Jika suatu suku dari fungsi POS memiliki literal lengkap, maka suku tersebut disebut *maxterm*. *maxterm* dibentuk dengan menyatakan setiap peubah yang bernilai 0 tanpa komplemen dan yang bernilai 1 dalam komplemen. *Maxterm* memiliki lambang M atau notasi $\prod$.

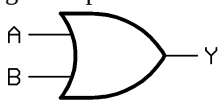
Fungsi Boolean dapat ditulis dalam bentuk kanonik dari tabel kebenaran dengan cara mengambil *minterm* dari setiap baris yang bernilai 1 (untuk SOP) atau mengambil *maxterm* dari setiap baris yang bernilai 0 (untuk POS). Sehingga, fungsi SOP adalah komplemen dari fungsi POS dan sebaliknya.

B. Gerbang Logika

Gerbang logika adalah rangkaian elektronika dengan dua atau lebih sinyal input serta satu sinyal output. Terdapat tiga gerbang logika dasar, yaitu OR, AND, dan NOT.

1. Gerbang OR

   Output dari gerbang OR bernilai 1 jika salah satu input bernilai 1. Gerbang OR melakukan logika pertambahan dengan ekspresi Boolean Y = A + B.



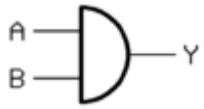
Gambar 2.1. Simbol gerbang OR

Input		Output
A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	1

Tabel 2.2. Tabel kebenaran gerbang OR dua input

2. Gerbang AND

   Output dari gerbang AND bernilai 1 jika dan hanya jika semua input bernilai 1. Gerbang AND melakukan logika perkalian dengan ekspresi Boolean Y = A . B.



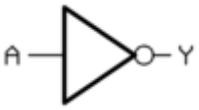
Gambar 2.2. Simbol gerbang AND

Input		Output
A	**B**	**Y**
0	0	0
0	1	0
1	0	0
1	1	1

Tabel 2.3. Tabel kebenaran gerbang AND dua input

3. Gerbang NOT

   Gerbang NOT atau *inverter* adalah gerbang yang hanya menerima satu input dan memberi output kebalikan atau komplemen dari nilai inputnya. Ekspresi Boolean dari gerbang NOT adalah Y = A'.



Gambar 2.3. Simbol gerbang NOT

Input	Output
A	**Y**
0	1
1	0

Tabel 2.4. Tabel kebenaran gerbang NOT

C. Peta Karnaugh

Peta Karnaugh atau *K-map* adalah bentuk grafis dari tabel kebenaran yang digunakan untuk menyederhanakan fungsi Boolean. Menyederhanakan fungsi Boolean berarti mencari bentuk fungsi lain yang ekivalen namun menggunakan jumlah literal atau operasi yang lebih sedikit. Penyederhanaan fungsi Boolean penting untuk dilakukan karena fungsi yang sama akan menggunakan jumlah gerbang logika yang lebih sedikit dan rangkaian logikanya akan menjadi lebih sederhana.

Metode peta Karnaugh dalam penyederhanaan fungsi Boolean pertama kali ditemukan oleh Maurice Karnaugh pada tahun 1953. Peta Karnaugh terdiri dari kotak-kotak yang bersisian membentuk persegi. Jumlah dari kotak bergantung pada jumlah peubah pada ekspresi Boolean yang ingin disederhanakan. Setiap kotak merepresentasikan sebuah *minterm* dan dikatakan bertetangga jika *minterm-minterm* tersebut hanya berbeda satu literal. Pengisian peta Karnaugh dilakukan dengan cara mengisi 1 untuk kotak yang menyatakan *minterm* dan mengisi 0 untuk kotak sisanya. Untuk menyederhanakan fungsi Boolean, dilakukan pengelompokan kotak-kotak pada peta Karnaugh yang bernilai 1 dan saling bersisian dengan kelompok pasangan (dua), kuad (empat), dan delapan (oktet). Pengelompokan dilakukan dengan jumlah yang lebih banyak terlebih dahulu, yaitu delapan, empat, lalu dua.

Sebagai contoh, misal ingin dilakukan minimisasi fungsi Boolean berikut.

f(w, x, y, z) = w'x'y' + x'yz' + w'xyz' + wx'y'

Fungsi Boolean tersebut dapat diminimisasi menggunakan

peta Karnaugh dengan cara mengelompokan kotak-kotak yang bernilai 1 dan saling bersisian. Pengelompokan tersebut dapat dilihat pada tabel di bawah ini.

wx \ yz	00	01	11	10
00	1	1	0	1
01	0	0	0	1
11	0	0	0	0
10	1	1	0	1

Tabel 2.5. Penyederhanaan fungsi Boolean f(w, x, y, z) menggunakan peta Karnaugh

Melalui pengelompokan tersebut, didapat bentuk sederhana dari fungsi tersebut hasil minimisasi menggunakan teknik peta Karnaugh yaitu:

f(w, x, y, z) = x'y' + x'z' + w'yz'

Perlu diingat bahwa hasil dari penyederhanaan fungsi Boolean menggunakan peta Karnaugh tidak selalu unik. Hal ini berarti terdapat beberapa bentuk fungsi minimisasi yang berbeda dengan jumlah literal dan *term* yang sama.

D. Keadaan Don't Care

Keadaan *don't care* adalah kondisi nilai peubah yang tidak diperhitungkan oleh fungsinya. Hal ini berarti pada peubah *don't care*, apapun nilainya, baik 1 atau 0, tidak akan memengaruhi hasil dari fungsi tersebut. Keadaan *don't care* biasanya disimbolkan dengan X yang nilainya dapat disesuaikan dengan kebutuhan. Keadaan *don't care* bermanfaat dalam menyederhanakan peta Karnaugh karena kita dapat menganggap semua nilai X sama dengan 1 dan membentuk kelompok sebesar mungkin yang mengikutsertakan angka 1 termasuk tanda X tersebut. Kemudian, nilai X yang tidak termasuk dalam kelompok tersebut akan dianggap bernilai 0. Sehingga, keadaan-keadaan *don't care* tersebut dapat dimanfaatkan dengan baik dalam melakukan minimisasi fungsi Boolean yang lebih sederhana.

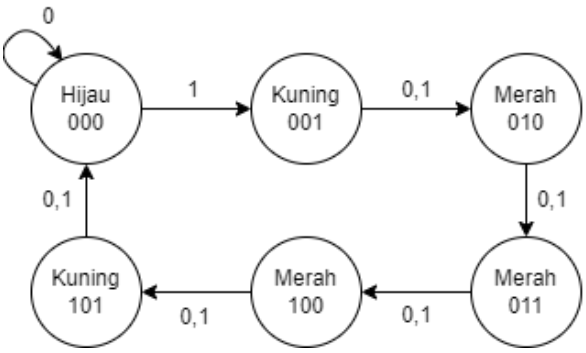
III. PENERAPAN ALJABAR BOOLEAN PADA SISTEM LAMPU LALU LINTAS *PELICAN CROSSING*

A. State Diagram

Sebelum merancang rangkaian lampu lalu lintas *pelican crossing* secara utuh, perlu dibuat kerangka dari sistem yang ingin dibangun beserta *state* atau keadaan-keadaan yang ingin dicapai. Hal ini dapat digambarkan dalam bentuk *state diagram.* Dalam lampu lalu lintas *pelican crossing*, pejalan kaki dapat menekan tombol untuk mengirimkan instruksi lampu lalu lintas untuk mengatur alur kendaraan sehingga pejalan kaki dapat aman menyeberang jalan. Pada sistem ini, dibutuhkan tiga sinyal lampu lalu lintas bagi kendaraan, yaitu merah, kuning, dan hijau. Ketiga sinyal ini kita representasikan dalam kode biner dengan tiga bit yang melambangkan *state* atau keadaan dari sistem lampu lalu lintas yang akan dirancang. Representasi biner tersebut adalah sebagai berikut.

000 : Sinyal H (Hijau)
001, 101 : Sinyal K (Kuning)
010, 011, 100 : Sinyal M (Merah)

Selain ketiga sinyal lampu, dibutuhkan juga input berupa tombol yang dapat dioperasikan oleh pejalan kaki. Tombol tersebut akan ditekan oleh pejalan kaki ketika ingin menyeberang jalan. Ketika tombol ditekan, maka input bernilai 1. Sebaliknya, ketika tombol tidak ditekan, maka input bernilai 0. Kemudian, kita dapat membuat *state diagram* untuk mengatur perubahan warna dari lampu lalu lintas yang menyala sebagai berikut.



Gambar 3.1. State diagram lampu lalu lintas

Dari *state diagram* di atas, dapat dilihat bahwa pada kondisi awal, lampu yang menyala adalah lampu hijau. Ketika lampu hijau menyala dan pejalan kaki tidak menekan tombol (input bernilai 0), maka lampu hijau akan terus menyala. Namun, ketika tombol ditekan (input bernilai 1), *state* dari lampu akan berubah sehingga lampu kuning yang menyala. Kemudian, apapun input setelahnya akan membuat sistem berpindah *state* ke lampu merah. Hal ini menginstruksikan pengendara untuk berhenti sehingga pejalan kaki dapat menyeberang jalan dengan aman. Lampu merah ini akan terus menyala dalam tiga *state*. Hal ini mensimulasikan waktu yang diberikan kepada pejalan kaki untuk menyeberang jalan. Setelah melewati tiga *state* lampu merah, maka sistem akan menuju ke *state* lampu kuning dan berakhir ke *state* lampu hijau kembali hingga tombol ditekan lagi untuk mengulangi keadaan seperti penjelasan di atas.

B. Tabel Kebenaran

Dari *state diagram* yang telah dibuat pada bagian sebelumnya, kita dapat merancang tabel kebenarannya. Kolom-kolom pada tabel kebenaran tersebut berisi setiap bit yang merepresentasikan *current state* (keadaan saat ini), input (tombol yang dapat ditekan oleh pejalan kaki), *next state* (keadaan berikutnya berdasarkan *current state* dan input tombol), dan output (warna lampu lalu lintas yang akan menyala) berdasarkan kombinasi ketiganya. Sedangkan baris pada tabel tersebut adalah kombinasi atau kemungkinan yang dapat dicapai. dari *current state* yang direpresentasikan dengan tiga bit dan input yang direpresentasikan dengan satu bit, total semua kombinasi kemungkinan yang dapat dicapai adalah 2^{3+1} = 16 kemungkinan. Semua kombinasi kemungkinan tersebut dapat lebih jelas dilihat pada tabel di bawah ini.

Current State			**Input**	**Next State**			**Output**		
w	**y**	**x**	**z**	**N_1**	**N_2**	**N_3**	**M**	**K**	**H**
0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0	0	1
0	0	1	0	0	1	0	0	1	0
0	0	1	1	0	1	0	0	1	0
0	1	0	0	0	1	1	1	0	0

0	1	0	1	0	1	1	1	0	0
0	1	1	0	1	0	0	1	0	0
0	1	1	1	1	0	0	1	0	0
1	0	0	0	1	0	1	1	0	0
1	0	0	1	1	0	1	1	0	0
1	0	1	0	0	0	0	0	1	0
1	0	1	1	0	0	0	0	1	0
1	1	0	0	X	X	X	X	X	X
1	1	0	1	X	X	X	X	X	X
1	1	1	0	X	X	X	X	X	X
1	1	1	1	X	X	X	X	X	X

Tabel 3.1. Tabel kebenaran sistem lampu lalu lintas

Dari tabel kebenaran tersebut, setiap bit pada *next state* serta output akan dibuat dalam fungsi Boolean. Dari tabel kebenaran tersebut, maka fungsi Boolean untuk masing-masing bit *next state* dan output dalam bentuk *Sum of Product* (SOP) adalah sebagai berikut.

$N_1 = m_6 + m_7 + m_8 + m_9 = \sum (6, 7, 8, 9)$

$N_2 = m_2 + m_3 + m_4 + m_5 = \sum (2, 3, 4, 5)$

$N_3 = m_1 + m_4 + m_5 + m_8 + m_9 = \sum (1, 4, 5, 8, 9)$

$M = m_4 + m_5 + m_6 + m_7 + m_8 + m_9 = \sum (4, 5, 6, 7, 8, 9)$

$K = m_2 + m_3 + m_{10} + m_{11} = \sum (2, 3, 10, 11)$

$H = m_0 + m_1 = \sum (0, 1)$

Terdapat beberapa kombinasi *current state* dan input yang tidak pernah tercapai dan tidak perlu diperhitungkan dalam perancangan sistem karena kondisi tersebut tidak berpengaruh pada hasil akhirnya. Oleh karena itu, kondisi-kondisi tersebut diisi dengan keadaan *don't care*. Kondisi *don't care* ini dapat dilihat pada empat baris terakhir pada tabel kebenaran di atas. Kondisi-kondisi tersebut menjadi keadaan *don't care* karena tidak ada representasi 110 maupun 111 pada bit biner warna lampu lalu lintas yang kita rancang. Oleh karena itu, apapun nilainya nanti tidak akan berpengaruh pada hasil akhir.

Jika diperhatikan dengan teliti, fungsi Boolean dari *next state* maupun output warna lampu lalu lintas yang dibuat dari tabel kebenaran tersebut belum merupakan fungsi yang paling sederhana. Oleh karena itu, diperlukan penyederhanaan untuk fungsi-fungsi Boolean tersebut. Teknik penyederhanaan yang dipakai yaitu menggunakan peta Karnaugh.

C. Peta Karnaugh dan Rangkaian Logika Next State

Selanjutnya, dibuat peta Karnaugh untuk masing-masing bit dari *next state* dengan empat peubah, yaitu bit dari *current state* (dilambangkan dengan w, x, dan y) serta bit dari input yang dilambangkan dengan z. Peta Karnaugh ini bertujuan untuk menyederhanakan setiap fungsi Boolean yang telah kita buat dari tabel kebenaran sehingga dapat meminimalkan penggunaan gerbang logika pada rangkaiannya nanti.

Setiap kotak pada peta Karnaugh diisi dengan 0, 1, atau X berdasarkan tabel kebenaran yang telah dibuat sebelumnya. Setelah mengisi setiap kotak dengan nilai yang sesuai, dilakukan penyederhanaan untuk fungsi Boolean tersebut. Penyederhanaan yang dilakukan yaitu dengan menggabungkan kotak-kotak yang bernilai 1 dan bersisian sesuai dengan aturan minimisasi peta Karnaugh. Oleh karena itu, hasil dari penyederhanaan fungsi Boolean tersebut adalah dalam bentuk *Sum of Product* (SOP).

Yang pertama, kita akan melakukan penyederhanaan untuk fungsi Boolean N_1, yaitu bit paling kiri dari *next state*. Berikut peta Karnaugh dari fungsi N_1 beserta pengelompokannya.

wx \ yz	**00**	**01**	**11**	**10**
00	0	0	0	0
01	0	0	1	1
11	X	X	X	X
10	1	1	0	0

Tabel 3.2. Peta Karnaugh N_1

Berdasarkan pengelompokan peta Karnaugh di atas, hasil penyederhanaan dari fungsi Boolean tersebut adalah:

$N_1 = wy' + xy$

Selanjutnya, kita juga melakukan penyederhanaan untuk fungsi Boolean N_2, yaitu bit tengah dari *next state*. Berikut peta Karnaugh dari fungsi N_2 beserta pengelompokannya.

wx \ yz	**00**	**01**	**11**	**10**
00	0	0	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	0	0

Tabel 3.3. Peta Karnaugh N_2

Berdasarkan pengelompokan peta Karnaugh di atas, hasil penyederhanaan dari fungsi Boolean tersebut adalah:

$N_2 = xy' + w'x'y$

Dengan cara yang sama, dilakukan penyederhanaan untuk fungsi Boolean N_3, yaitu bit paling kanan dari *next state*. Berikut peta Karnaugh dari fungsi N_3 beserta pengelompokannya.

wx \ yz	**00**	**01**	**11**	**10**
00	0	1	0	0
01	1	1	0	0
11	X	X	X	X
10	1	1	0	0

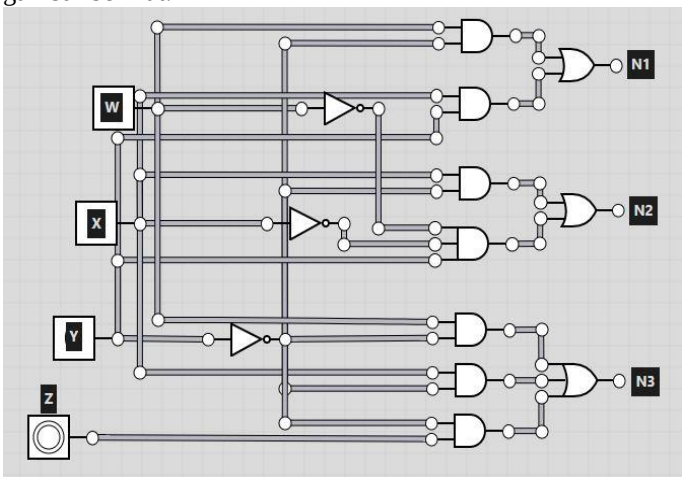
Tabel 3.4. Peta Karnaugh N_3

Berdasarkan pengelompokan peta Karnaugh di atas, hasil penyederhanaan dari fungsi Boolean tersebut adalah:

$N_3 = wy' + xy' + y'z$

Setelah melakukan penyederhanaan fungsi Boolean untuk *next state*, kita dapat menggambarkannya melalui rangkaian logika menggunakan gerbang AND, OR, dan NOT. Ekspresi Boolean perkalian digambarkan dalam gerbang AND, ekspresi penjumlahan digambarkan dalam gerbang OR, dan ekspresi komplemen digambarkan dalam gerbang NOT. Rangkaian logika dari N_1, N_2, dan N_3 ini berfungsi untuk mengatur keadaan berikutnya (*next state*) berdasarkan kombinasi nilai peubah, yaitu keadaan saat ini atau lampu yang sedang menyala (*current state*) serta input tombol yang dioperasikan oleh pejalan kaki. Urutan *state* yang akan dicapai sesuai dengan *state diagram* yang telah dirancang sebelumnya. Rangkaian logika untuk

ketiga fungsi Boolean bit *next state* tersebut dapat dilihat pada gambar berikut.



Gambar 3.2. Rangkaian logika N_1, N_2, dan N_3

Pada gambar di atas, variabel w, x, dan y adalah representasi bit *current state*, sedangkan variabel z menggambarkan input berupa tombol yang dapat ditekan oleh pejalan kaki. Input tersebut akan bernilai 1 jika tombol ditekan, atau bernilai 0 jika tombol tidak ditekan. Dengan adanya penyederhanaan melalui peta Karnaugh, rangkaian yang dibuat dapat lebih sederhana dengan menggunakan gerbang-gerbang logika seminimal mungkin.

D. Peta Karnaugh dan Rangkaian Logika Output

Seperti pada bagian sebelumnya, kita juga perlu melakukan penyederhanaan untuk fungsi Boolean pada output lampu lalu lintas karena nantinya akan dibuat juga rangkaian logika untuk ketiga lampu ini. Untuk itu, kita perlu membuat peta Karnaugh untuk masing-masing output warna lampu. Disini, setiap fungsi Boolean lampu dilambangkan dengan inisial dari warna lampu tersebut, yaitu M untuk lampu berwarna merah, K untuk lampu berwarna kuning, dan H untuk lampu berwarna hijau. Sama seperti sebelumnya, penyederhanaan yang dilakukan yaitu dengan menggabungkan kotak-kotak yang bernilai 1 dan bersisian, sehingga hasilnya adalah dalam bentuk *Sum of Product* (SOP).

Berikut adalah peta Karnaugh dari fungsi Boolean M untuk lampu merah beserta pengelompokannya.

wx \ yz	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	X	X	X	X
10	1	1	0	0

Tabel 3.5. Peta Karnaugh lampu merah

Berdasarkan pengelompokan peta Karnaugh di atas, hasil penyederhanaan dari fungsi Boolean tersebut adalah:

M = x + wy'

Selanjutnya adalah penyederhanaan fungsi Boolean lampu kuning. Tabel berikut menunjukkan peta Karnaugh beserta pengelompokannya untuk fungsi Boolean K (lampu kuning).

wx \ yz	00	01	11	10
00	0	0	1	1
01	0	0	0	0
11	X	X	X	X
10	0	0	1	1

Tabel 3.6. Peta Karnaugh lampu kuning

Berdasarkan pengelompokan peta Karnaugh di atas, hasil penyederhanaan dari fungsi Boolean tersebut adalah:

K = x'y

Dilakukan juga penyederhanaan untuk fungsi Boolean lampu hijau. Tabel berikut menunjukkan peta Karnaugh beserta pengelompokannya untuk fungsi Boolean H (lampu hijau).

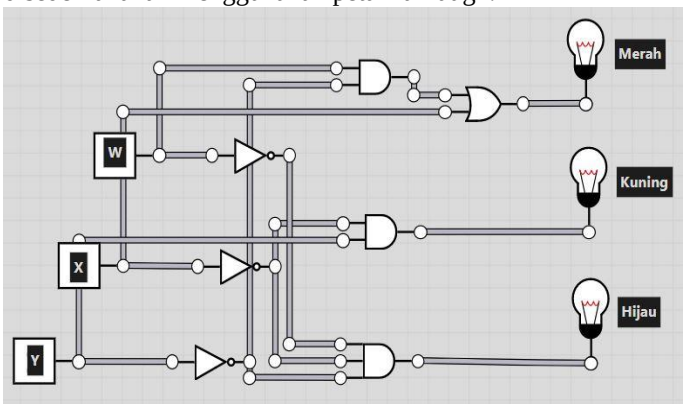
wx \ yz	00	01	11	10
00	1	1	0	0
01	0	0	0	0
11	X	X	X	X
10	0	0	0	0

Tabel 3.7. Peta Karnaugh lampu hijau

Berdasarkan pengelompokan peta Karnaugh di atas, hasil penyederhanaan dari fungsi Boolean tersebut adalah:

H = w'x'y'

Sama seperti *next state*, dibuat juga rangkaian logika untuk fungsi Boolean output lampu lalu lintas yang telah disederhanakan menggunakan peta Karnaugh.



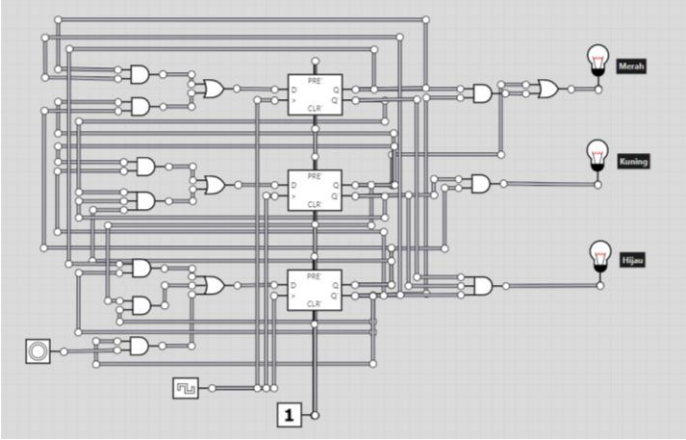
Gambar 3.3. Rangkaian logika lampu merah, kuning, dan hijau

Dari rangkaian di atas, dapat dilihat bahwa variabel yang digunakan hanya bit untuk *current state* saja, sedangkan nilai variabel input atau tombol tidak dimasukkan dalam rangkaian tersebut. Hal ini karena kondisi warna lampu yang menyala ditentukan oleh *current state*-nya saja. Oleh karena itu, variabel input dapat dihilangkan karena tidak memengaruhi hasil akhir.

IV. IMPLEMENTASI

Setelah melakukan penyederhanaan fungsi Boolean dan membuat rangkaian logika untuk *next state* dan output lampu lalu lintas, langkah terakhir dalam pembuatan lampu lalu lintas ini yaitu menggabungkan seluruhnya menjadi rangkaian sistem lampu lalu lintas *pelican crossing* yang utuh. Pada implementasi ini, penulis memanfaatkan rangkaian D Flip-Flop. Flip-Flop

adalah rangkaian elektronika yang memiliki dua *state* stabil yang digunakan untuk menyimpan data biner pada suatu waktu. Salah satu jenis dari flip-flop adalah D Flip-Flop yang memiliki satu input data dan satu input *clock*. Pada implementasi ini, flip-flop digunakan untuk menyimpan data dari *next state*. Berikut adalah gambar dari rangkaian lampu lalu lintas yang memanfaatkan komponen D Flip Flop.



Gambar 4.1. Rangkaian logika sistem lampu lalu lintas *pelican crossing*

Pada rangkaian tersebut, digunakan tiga buah flip-flop yang masing-masing merepresentasikan satu. Output dari flip-flop adalah bit dari *current state*, sedangkan masukannya adalah bit dari *next state*. Kondisi lampu yang menyala ditentukan oleh nilai dari kondisi saat ini atau *current state*. Oleh karena itu, output lampu dibuat berdasarkan keluaran dari flip-flop. Selain mengatur output lampu yang menyala, nilai dari *current state* juga memengaruhi nilai dari *next state*. Karena masukan flip-flop merepresentasikan nilai dari *next state*, maka output dari flip-flop akan menjadi variabel bagi masukannya. Nilai dari *next state* tersebut disimpan dan nantinya akan menjadi nilai *current state* untuk *clock* berikutnya. Sehingga, kondisi lampu akan berubah sesuai kondisi pada *next state* untuk *current state* pada *clock* sebelumnya. Hal ini akan terus berlangsung sesuai dengan rancangan *state diagram* yang telah dibuat pada gambar 3.1. Ketika tombol ditekan, sistem akan menerima input bernilai 1 dan memulai putaran *state*-nya sesuai mekanisme yang telah dijelaskan sebelumnya. Lampu kuning akan menyala, kemudian lampu merah menyala dalam tiga *clock*, dilanjutkan dengan lampu kuning, dan diakhiri dengan lampu hijau yang menyala hingga sistem menerima input bernilai 1 kembali.

V. KESIMPULAN

Prinsip aljabar Boolean sering diaplikasikan dalam rangkaian digital, salah satunya dalam sistem pengaturan lampu lalu lintas *pelican crossing*. Dengan menerapkan prinsip aljabar Boolean, setiap *state* dari rangkaian lampu lalu lintas *pelican crossing* dapat direpresentasikan dengan bit biner serta disederhanakan dengan mudah menggunakan teknik minimisasi melalui peta Karnaugh. Dengan demikian, sistem yang dibangun juga lebih optimal menggunakan gerbang logika seminimal mungkin.

VI. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas berkat dan rahmat-Nya sehingga makalah ini dapat diselesaikan tepat waktu. Penulis juga mengucapkan terima kasih kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc selaku dosen pengampu mata kuliah Matematika Diskrit Semester I 2022/2023 Kelas 01 yang telah memberikan bimbingan dan pengajarannya selama satu semester ini. Tidak lupa penulis juga mengucapkan terima kasih kepada orang tua yang senantiasa memberikan dukungan selama proses pendidikan penulis, serta seluruh pihak yang telah membantu dan mendukung penulis dalam menyelesaikan makalah ini.

REFERENSI

[1] Shale-Hester, Tristan. 2020. "What is a Pelican crossing?" https://www.autoexpress.co.uk/car-news/105226/what-is-a-pelican-crossing , diakses 2 Desember 2022.

[2] Munir, Rinaldi. 2020. "Aljabar Boolean (Bagian 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Aljabar-Boolean-(2020)-bagian1.pdf , diakses 3 Desember 2022.

[3] Munir, Rinaldi. 2020. "Aljabar Boolean (Bagian 2)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Aljabar-Boolean-(2020)-bagian2.pdf , diakses 3 Desember 2022.

[4] Panda, Sneha. 2021. "D Flip Flop: Circuit, Truth Table, Working, Critical Differences" https://lambdageeks.com/d-flip-flop-circuit-working-truthtable-differences/ , diakses 8 Desember 2022.

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, 9 Desember 2022

Margaretha Olivia Haryono 13521071