Aplikasi Aljabar Boolean dalam Logika Perhitungan pada Kalkulator Analog

Valentino Chryslie Triadi - 13522164[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522164@std.stei.itb.ac.id

***Abstrak*—Kalkulator analog adalah salah satu alat hitung yang masih sering digunakan dalam perhitungan. Jenis perhitungan yang sering digunakan salah satunya adalah penjumlahan dua angka. Penjumlahan dua angka dapat dilakukan karena adanya komponen pada kalkulator analog yang melakukan perhitungan, salah satu komponen kalkulator analog tersebut adalah gerbang logika. Penggunaan gerbang logika seperti gerbang *and, or, xor, not,* dan yang lainnya harus disesuaikan dengan tujuan penggunaan. Penggunaan gerbang logika yang berlebih atau tidak sesuai dengan tujuan dapat membuat sistem yang terbentuk menjadi kurang baik. Sehingga, diperlukan sebuah metode untuk bisa menentukan kebutuhan gerbang logika yang tepat, yakni mengenai perhitungan aljabar boolean.**

***Kata Kunci*—Perhitungan, Kalkulator, Gerbang Logika, Aljabar Boolean, Peta Karnaugh.**

I. Pendahuluan

Aljabar Boolean adalah salah satu ilmu/keahlian yang sering digunakan dalam pembuatan atau penyusunan sebuah alat elektronik. Walaupun aljabar Boolean masih kurang dikenali oleh masyarakat, namun penerapannya sudah sangat sering terlihat. Salah satu penerapan dari aljabar Boolean yang sering terlihat adalah *countdown timer* pada lampu lalu lintas. Penerapan aljabar Boolean pada *countdown timer* dapat terlihat pada munculnya angka 0 hingga 9 yang dapat dibedakan dari bentuknya.

Penerapan aljabar Boolean yang paling sering kita gunakan salah satunya adalah *Aritmethic logic unit* (ALU) pada gadget yang kita gunakan. *Aritmethic logic unit* atau ALU sendiri menggunakan gerbang logika sebagai komponen utamanya, dimana gerbang logika ini dirangkai dengan arsitektur yang seminimal mungkin dan tepat. Perangkaian arsitektur *adder* pada *Aritmethic logic unit* (ALU) tentunya memerlukan keilmuan aljabar Boolean dalam menentukan arsitektur yang terbaik.

Apabila dalam penyusunan arsitektur komponen elektronik atau alat elektronik tidak digunakan aljabar Boolean, ada kemungkinan terjadinya pemakaian gerbang logika yang berlebihan. Dalam hal tersebut, meningkatnya penggunaan gerbang logika yang digunakan dapat meningkatkan harga produksi suatu komponen elektronik atau alat elektronik. Hal tersebut tentunya bukanlah sebuah hasil yang diharapkan.

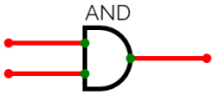
Kalkulator analog juga menggunakan keilmuan aljabar Boolean dalam komponen-komponennya, seperti arsitektur pada komponen untuk menampilkan angka di layar, *parallel adder, dan* operasi perkalian. Cara kerja komponen-komponen tersebut adalah dengan menyalurkan dan mengolah informasi dari masukan hingga keluaran dalam bentuk *high signal* dan *low signal* atau biasanya dikenal dengan sebutan 1 (*high)* dan 0 (*low)*. Cara kerja tersebut yang membuat peran aljabar Boolean dalam penyusunan arsitektur menjadi sangat penting.

II. Dasar Teori Gerbang Logika, Aljabar Boolean, dan Peta Karnaugh

A. Gerbang Logika

Gerbang logika adalah suatu komponen dalam elektronika dan matematika Boolean yang mengubah satu atau lebih logik masukan menjadi sebuah sinyal logik keluaran. Sinyal yang masuk atau keluar pada gerbang logika terdapat 2 jenis, yaitu sinyal rendah atau biasa direpresentasikan dengan angka "0" dan sinyal tinggi atau biasa direpresentasikan dengan angka "1". Gerbang logika dapat dibedakan dari kompleksitas penyusunannya, yaitu gerbang logika dasar dan gerbang logika kombinasi atau turunan. Gerbang logika dasar terdiri atas tiga gerbang logika, yaitu

1. Gerbang *AND*



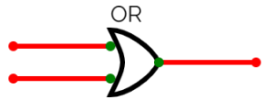
Gambar 1. Gerbang AND

Gerbang *AND* adalah gerbang yang menerima masukan sinyal dan mengembalikan sinyal tinggi jika dan hanya jika kedua sinyal masukan adalah sinyal tinggi juga. Ilustrasi masukan dan keluaran gerbang *AND* dapat dilihat pada tabel berikut.

MASUKAN 1	MASUKAN 2	AND (*)
0	0	0
0	1	0
1	0	0
1	1	1

Tabel 1. Logik operasi AND

2. Gerbang *OR*



Gambar 2. Gerbang OR

Gerbang *OR* adalah gerbang yang menerima masukan sinyal dan mengembalikan sinyal tinggi jika salah satu sinyal masukan adalah sinyal tinggi. Ilustrasi masukan dan keluaran gerbang *OR* dapat dilihat pada tabel berikut.

MASUKAN 1	MASUKAN 2	OR (+)
0	0	0
0	1	1
1	0	1
1	1	1

Tabel 2. Logik operasi OR

3. Gerbang *NOT*

Gambar 3. Gerbang NOT

Gerbang *NOT* adalah gerbang yang menerima masukan sinyal dan mengembalikan sinyal keluaran yang berkebalikan dengan sinyal masukan. Ilustrasi masukan dan keluaran gerbang NOT dapat dilihat pada tabel berikut.

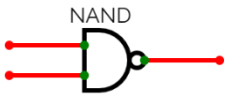
MASUKAN	NOT (~)
0	1
1	0

Tabel 3. Logik operasi NOT

Selain gerbang logika dasar, terdapat juga gerbang logika kombinasi atau turunan yang merupakan gabungan dari beberapa gerbang logika dasar. Beberapa contoh gerbang logika turunan adalah sebagai berikut

1. Gerbang *NAND*

Gerbang *NAND* merupakan gabungan dari gerbang *NOT* dan gerbang *AND*.



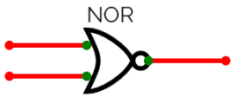
Gambar 4. Gerbang NAND

MASUKAN 1	MASUKAN 2	NAND (~ *)
0	0	1
0	1	1
1	0	1
1	1	0

Tabel 4. Logik operasi NAND

2. Gerbang *NOR*

Gerbang *NOR* merupakan gabungan dari gerbang *NOT* dan gerbang *OR*.



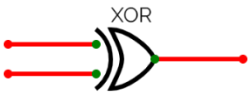
Gambar 5. Gerbang NOR

MASUKAN 1	MASUKAN 2	NOR (+)
0	0	1
0	1	0
1	0	0
1	1	0

Tabel 5. Logik operasi NOR

3. Gerbang *XOR*

Gerbang *XOR* merupakan gerbang *OR* yang eksklusif, dimana jika gerbang *OR* akan mengembalikan sinyal tinggi saat menerima 1 atau lebih sinyal masukan tinggi maka gerbang *XOR* hanya akan mengembalikan sinyal tinggi jika dan hanya jika kedua sinyal masukan bukanlah sinyal yang sejenis.



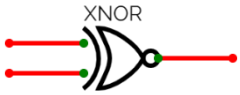
Gambar 6. Gerbang XOR

MASUKAN 1	MASUKAN 2	XOR (⊕)
0	0	0
0	1	1
1	0	1
1	1	0

Tabel 6. Logik operasi XOR

4. Gerbang *XNOR*

Gerbang *XNOR* merupakan gabungan antara gerbang *NOT* dan gerbang *OR* yang eksklusif. Gerbang ini juga dapat diartikan sebagai komplemen atau kebalikan dari gerbang *XOR*, dimana jika gerbang X*OR* akan mengembalikan sinyal tinggi saat menerima dua sinyal masukan yang tidak sejenis maka gerbang *XNOR* akan mengembalikan sinyal tinggi saat menerima dua sinyal masukan yang sejenis.



Gambar 7. Gerbang XNOR

MASUKAN 1	MASUKAN 2	XOR (⊕)
0	0	1
0	1	0
1	0	0
1	1	1

Tabel 7. Logik operasi XNOR

B. Aljabar Boolean

Aljabar Boolean pertama kali ditemukan oleh George Boole pada tahun 1854. Aljabar Boolean memiliki kemiripan dengan logika proposisi. Aljabar Boolean juga memiliki beberapa axioma, yakni symbol masukan harus berupa "0" atau "1", terdapat dua operasi biner (+ dan ·) dan satu operasi uner (’), dan memiliki kaidah untuk operasi biner dan uner sebagai berikut.

a	b	a · b
0	0	0
0	1	0
1	0	0
1	1	1

Tabel 8. Kaidah operasi biner ·

a	b	a + b
0	0	0
0	1	1
1	0	1
1	1	1

Tabel 9. Kaidah operasi biner +

a	a’
0	1
1	0

Tabel 10. Kaidah operasi uner ’

Selain itu, aljabar Boolean juga memiliki beberapa hukum yang berlaku

Hukum Identitas	• a + 0 = a • a · 1 = a
Hukum Idempoten	• a + a = a • a · a = a
Hukum Komplemen	• a + a’ = 1 • a · a’ = 0
Hukum Dominasi	• a + 1 = 1 • a · 0 = 0
Hukum Involusi	• (a’)’ = a
Hukum Penyerapan	• a + ab = a • a · (a + b) = a
Hukum Komutatif	• a + b = b + a • a · b = b · a
Hukum Asosiatif	• a + (b + c) = (a + b) + c • a (b c) = (a b) c
Hukum Distributif	• a + (b c) = (a + b)(a + c) • a · (b + c) = (a b) + (a c)
Hukum De Morgan	• (a + b)’ = a’ · b’ • (a · b)’ = a’ + b’
Hukum 0/1	• 0’ = 1 • 1’ = 0

Tabel 11. Hukum Aljabar Boolean

Dalam aljabar Boolean juga terdapat ekspresi Boolean yang dapat diekspresikan dalam dua bentuk kanonik, yakni

1. Bentuk penjumlahan dari hasil kali atau *Sum-Of-Product* (SOP)

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

Bentuk tersebut mengandung beberapa minterm (suku yang mengandung literal lengkap dalam bentuk hasil kali)

2. Bentuk perkalian dari hasil jumlah atau *Product-Of-Sum* (POS).

$$f(x,y,z) = (x' + y' + z)(x + y + z)(x' + y + z')$$

Bentuk tersebut mengandung beberapa maxterm (suku yang mengandung literal lengkap dalam bentuk hasil tambah)

C. Peta Karnaugh

Peta Karnaugh adalah salah satu metode grafis untuk penyederhanaan fungsi Boolean. Metode ini ditemukan oleh Maurice Karnaugh pada tahung 1953. Peta Karnaugh adalah sebuah diagram yang berbentuk kotak-kotak yang bersisian. Tiap kotak dalam peta Karnaugh merepresentasikan sebuah minterm. Dengan menggunakan metode peta Karnaugh, menyederhanakan fungsi Boolean menjadi sebuah hal yang mudah.

Menyederhanakan sebuah fungsi Boolean dengan menggunakan peta Karnaugh dapat dilakukan dengan cara menggabungkan elemen-elemen bernilai 1 yang saling bersisian. Penggabungan elemen tersebut dapat dibentuk menjadi beberapa kelompok gabungan, yaitu oktet (delapan elemen bersisian), kuad (empat elemen berisisian), dan pasangan (dua elemen bersisian). Dalam penggabungan elemen pada peta Karnaugh diutamakan penggabungan elemen bersisian dari yang paling banyak terlebih dahulu, yakni engan urutan prioritas oktet - kuad - pasangan. Dalam pembuatan peta Karnaugh, perlu diperhatikan bahwa kunci yang bersebelahan dari setiap kolom atau baris hanya boleh berbeda maksimal 1 digit. Sehingga dalam pembuatannya tidak boleh ada kunci bersebelahan yang memiliki dua atau lebih digit yang berbeda. Pengisian peta Karnaugh harus sesuai dengan representasi minterm dimana variabel bukan komplemen bernilai 1 dan variabel komplemen bernilai 0. Berikut adalah contoh peta Karnaugh yang terbentuk dari fungsi Boolean dibawah ini.

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

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

Tabel 12. Peta Karnaugh

Peta Karnaugh diatas memiliki satu kuad (berwarna hijau) dan satu pasangan (berwarna biru). Sehingga hasil penyederhanaan fungsi Boolean tersebut adalah

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

III. Pemodelan Gerbang Logika

A. Half Adder

Half adder adalah pemodelan gerbang logika yang bertujuan untuk menambahkan dua buah bit (satu bit dengan satu bit lainnya) dan menghasilkan dua buah bit juga, yakni bit *SUM* dan bit *CARRY*. *SUM* adalah hasil dari penjumlahan dua buah bit yang dilakukan *cut* atau pemotongan saat terjadi *overflow* atau hasil dari penjumlahan dua buah bilangan biner adalah sebanyak dua bit. Sedangkan *CARRY* adalah sisa dari penjumlahan dua buah bit atau hasil *cut* yang tidak masuk ke *SUM*. Sebagai contoh apabila penjumlahan bilangan biner 1 dengan bilangan biner 1 maka akan menghasilkan bilangan biner 10 dimana hasil tersebut memiliki total 2 buah bit sehingga perlu dilakukan *cut* atau pemotongan sehingga *SUM* akan bernilai 0 dan *CARRY* akan bernilai 1. Berikut adalah ilustrasi masukan dan keluaran pada *half adder*.

Masukan		Keluaran	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Tabel 13. Tabel Kebenaran Half Adder

Dengan tabel kebenaran tersebut dapat dibuat dua buah peta Karnaugh, yakni peta Karnaugh *SUM* dan peta Karnaugh Carry.

A/B	0	1
0	0	1
1	1	0

Tabel 14. Peta Karnaugh SUM (Half Adder)

A/B	0	1
0	0	0
1	0	1

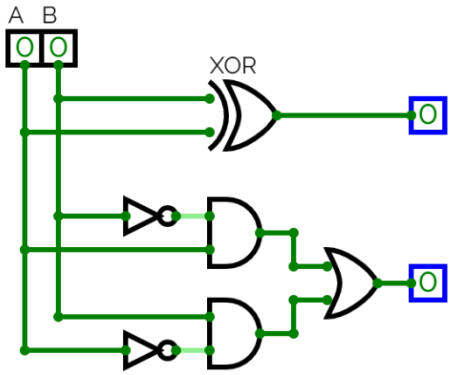
Tabel 15. Peta Karnaugh CARRY (Half Adder)

Fungsi boolean yang terbentuk dari dua peta Karnaugh diatas adalah

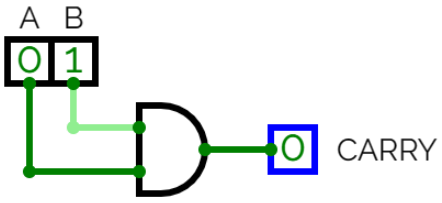
$$SUM(A,B) = A'B + AB' = A \oplus \mathrm{B}$$

$$CARRY(A,B) = AB$$

Susunan gerbang logika dari kedua fungsi diatas adalah sebagai berikut

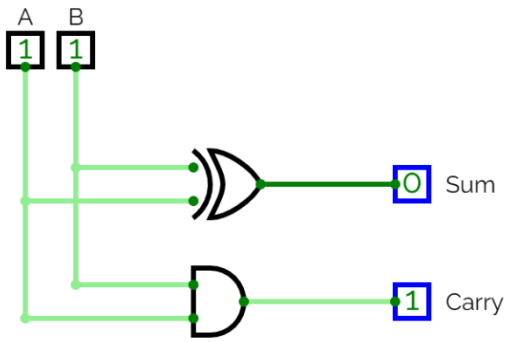


Gambar 8. Rangkaian Gerbang Logika SUM (Half Adder)



Gambar 9. Rangkaian Gerbang Logika CARRY (Half Adder)

Rangkaian gerbang logika *SUM* memiliki kesamaan dengan gerbang logika *XOR*. Sehingga, rangkaian gerbang logika *Half Adder* adalah sebagai berikut



Gambar 10. Rangkaian Gerbang Logika Half Adder

B. Full Adder

Dalam penggunaannya, ternyata *half adder* memiliki kekurangan, yaitu hanya bisa menjumlahkan bilangan biner dengan 1 buah bit saja dan tidak bisa menjumlahkan informasi *carry* yang masuk. Sehingga untuk menutupi kekurangan tersebut, *Full adder* dibuat dengan masukan sebanyak tiga buah bit, yaitu dua buah bilangan biner 1 bit dan 1 buah bit *carry in*. Berikut adalah ilustrasi masukan dan keluaran untuk *Full Adder*.

Masukan			Keluaran	
A	B	Cin	SUM	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Tabel 16. Tabel Kebenaran Full Adder

Dengan tabel kebenaran tersebut dapat dibuat dua buah peta Karnaugh, yakni peta Karnaugh *SUM* dan peta Karnaugh Carry.

A \ B Cin	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Tabel 17. Peta Karnaugh SUM (Full Adder)

A \ B Cin	00	01	11	10
0	0	0	1	0
1	0	1	1	1

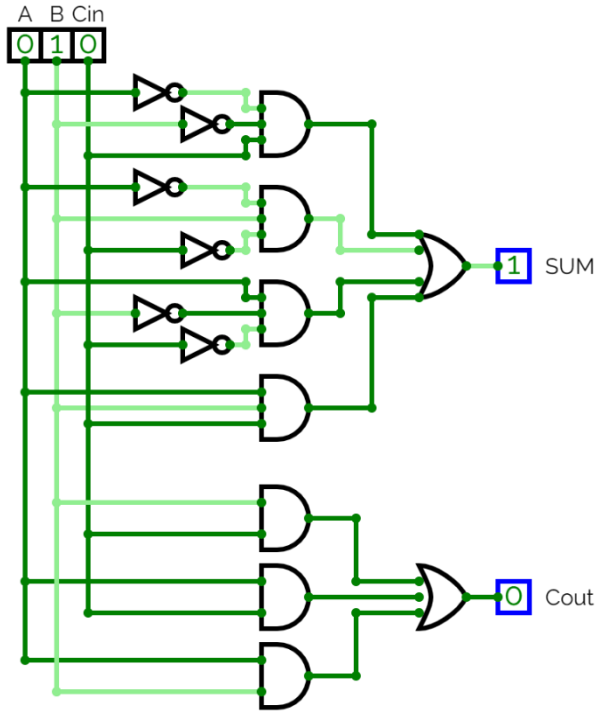
Tabel 18. Peta Karnaugh Cout (Full Adder)

Fungsi boolean yang terbentuk dari dua peta Karnaugh diatas adalah

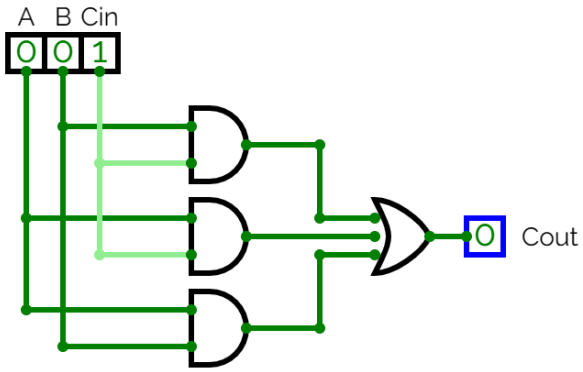
$$SUM(A,B) = A'B'Cin + A'BCin' + AB'Cin' + ABCin$$

$$Cout(A,B) = BCin + ACin + AB$$

Susunan gerbang logika dari kedua fungsi diatas adalah sebagai berikut

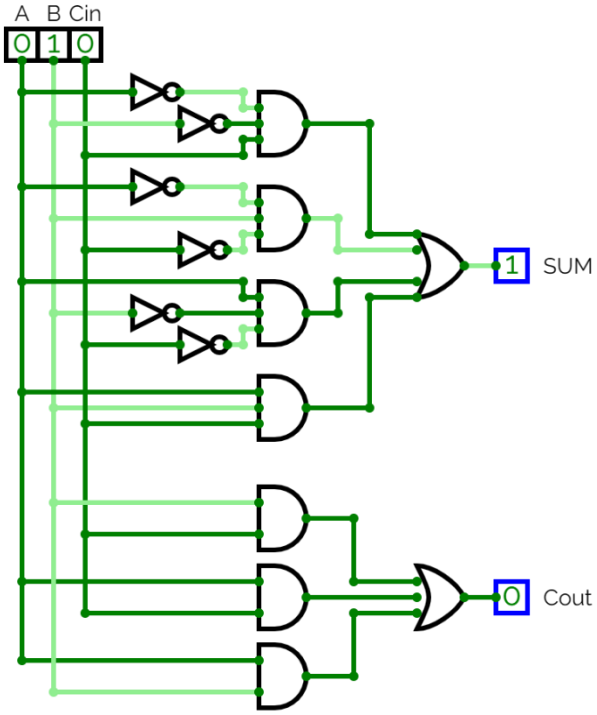


Gambar 11. Rangkaian Gerbang Logika SUM (Full Adder)



Gambar 12. Rangkaian Gerbang Logika Cout (Full Adder)

Sehingga, rangkaian gerbang logika *Half Adder* adalah sebagai berikut



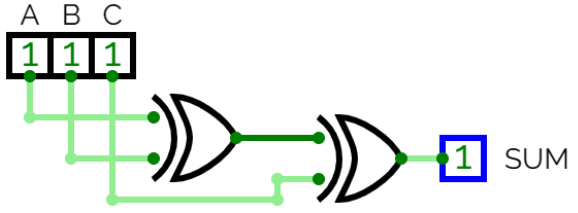
Gambar 13. Rangkaian Gerbang Logika Full Adder

.

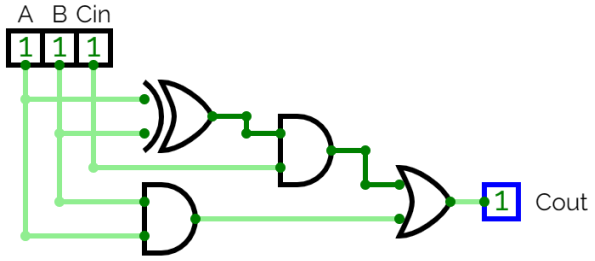
Rangkaian gerbang logika diatas disederhanakan dengan memanfaatkan peta Karnaugh. Namun ternyata penyederhanaan fungsi boolean tersebut masih kurang efektif karena masih banyaknya gerbang logika yang digunakan. Sehingga, jika menyederhanakan fungsi boolean tersebut dengan menggunakan hukum-hukum boolean dan ke-komplemen an *product of sum* dan *sum of product* akan didapat hasil sebagai berikut.

$$
\begin{aligned}
SUM(A,B) &= A'B'Cin + A'BCin' + AB'Cin' + ABCin \\
&= (A'B' + AB)Cin + (A'B + AB')Cin' \\
&= ((A+B)(A'+B'))'Cin + (A \oplus \mathrm{B})Cin' \\
&= (A'B + AB')'Cin + (A \oplus \mathrm{B})Cin' \\
&= (A \oplus \mathrm{B})'Cin + (A \oplus \mathrm{B})Cin' \\
&= (A \oplus \mathrm{B}) \oplus Cin \\
Cout(A,B) &= A'BCin + AB'Cin + ABCin' + ABCin \\
&= (A'B + AB')Cin + AB(Cin' + Cin) \\
&= (A \oplus \mathrm{B})Cin + AB
\end{aligned}
$$

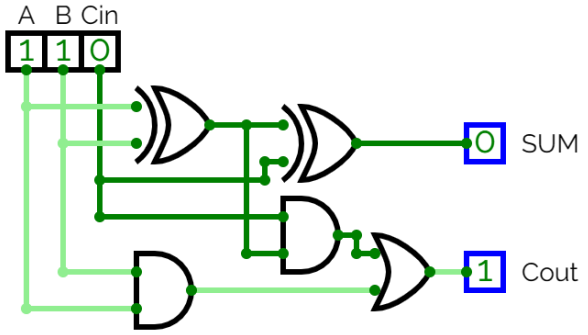
Sehingga, rangkaian gerbang logika *Half Adder* sederhana dengan menggunakan hukum aljabar boolean dan ke-komplemen an *product of sum* dan *sum of product* adalah sebagai berikut



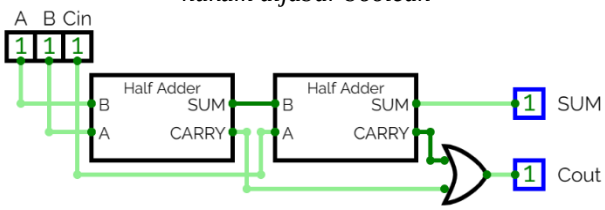
Gambar 14. Rangkaian Gerbang Logika SUM (Full Adder) dengan hukum aljabar boolean



Gambar 15. Rangkaian Gerbang Logika Cout (Full Adder) dengan hukum aljabar boolean



Gambar 16. Rangkaian Gerbang Logika Full Adder dengan hukum aljabar boolean



Gambar 17. Rangkaian Gerbang Logika Full Adder dengan memanfaatkan kemiripan susunan dengan Half Adder

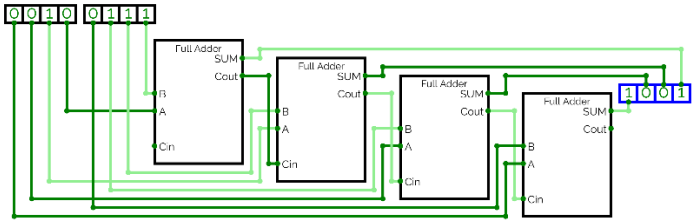
C. Parallel Adder

Parallel Adder adalah salah satu bentuk implementasi dari *full adder* yang digunakan dalam penjumlahan beberapa bit. *Parallel adder* ini memanfaatkan beberapa rangkaian *full adder* untuk melakukan penjumlahan. Jumlah rangkaian *full adder* yang digunakan adalah sebanyak jumlah bit masukan. Sehingga, untuk melakukan penjumlahan n-bit bilangan biner dengan n-bit bilangan biner akan diperlukan rangkaian *full adder* sebanyak n buah. Tabel kebenaran untuk rangkaian *parallel adder* adalah sebagai berikut.

A0	A1	A2	A3	B0	B1	B2	B3	R0	R1	R2	R3
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	0	0	1	1	0	0	1	1
0	0	0	0	0	1	0	0	0	1	0	0
...											
1	1	1	1	1	1	1	1	1	1	1	0

Tabel 19. Tabel Kebenaran 4-bit Parallel Adder

Dengan tabel kebenaran (selengkapnya dapat dilihat pada https://bit.ly/TabelKebenaranLogikaOperasi) tersebut dapat dibuat rangkaian *4-bit parallel adder* sebagai berikut.



Gambar 18. Rangkaian 4-bit Parallel Adder

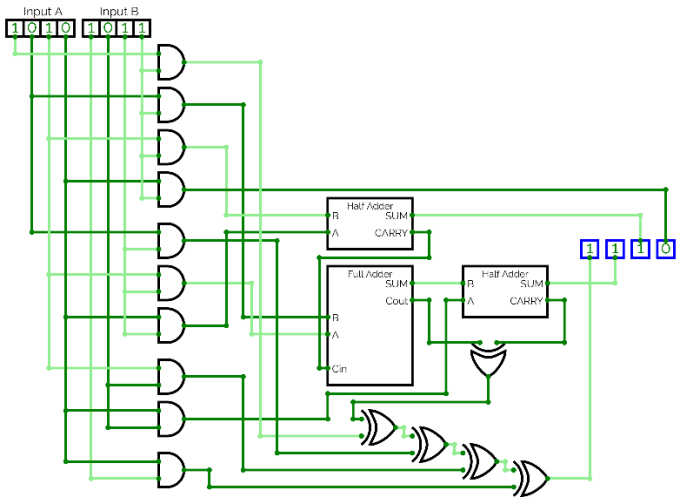
D. 4-bit Product

Selain operasi penjumlahan dan pengurangan, logika perhitungan yang dapat dibentuk dengan gerbang logika adalah operasi perkalian. Namun karena besarnya hasil perkalian yang *overflow* atau melebihi jumlah bit yang ditentukan, maka untuk logika operasi perkalian hanya akan dibatasi hingga 4-bit. Tabel kebenaran untuk operasi perkalian adalah sebagai berikut.

A0	A1	A2	A3	B0	B1	B2	B3	R0	R1	R2	R3
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0
0	0	0	0	0	0	1	1	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0
...											
1	1	1	1	1	1	1	1	0	0	0	1

Tabel 20. Tabel Kebenaran 4-bit Product

Dengan tabel kebenaran (selengkapnya dapat dilihat pada https://bit.ly/TabelKebenaranLogikaOperasi) tersebut dapat dibuat rangkaian *4-bit product* sebagai berikut.



Gambar 19. Rangkaian 4-bit Product

IV. PENGUJIAN RANGKAIAN LOGIKA

Rangkaian gerbang logika operasi perhitungan pada kalkulator analog bisa dicek kebenarannya dengan bantuan program *python* seperti berikut.

```
class HalfAdder:
    # Count 1 bit and 1 bit add operation
    def __init__(self) -> None:
        self.Sum = 0 # Sum
        self.Cout = 0 # Carry Out

    def Add (self, First:int, Second:int):
        # Sum is First XOR Second
        self.Sum = First ^ Second

        # Carry Out is First AND Second
        self.Cout =  First & Second

        return self.Sum, self.Cout

# Contoh pemanggilan
Adder = HalfAdder()
sum = Adder.Add(0, 0) # Count: 0 add 0
```

Program di atas adalah program untuk menghitung penjumlahan *half adder* dengan menggunakan operasi logika atau *bitwise operator* saja dan telah dirancang sesuai dengan pemodelan rangkaian gerbang logika *half adder* pada bab III. Program tersebut menerima dua buah parameter berupa 1-bit bilangan biner pertama dan 1-bit bilangan biner kedua. Program tersebut akan mengolah masukan dan mengembalikan 2-*tuple*, yaitu (*SUM*, *Carry*).

```
from halfAdder import HalfAdder
class FullAdder:
  # Count 1 bit and 1 bit full add operation
  # Carry in is the carry out from the previous
bit
  def __init__(self) -> None:
    # Half Add of First bit and Second bit
    self.tempSum = 0
    self.firstCout = 0

    # Half Add of tempSum and Carry In
    self.Sum = 0
    self.secondCout = 0

    # Carry Out
    self.Cout = 0
```

```
  def Add(self, First:int, Second:int, CarryIn:int):
    # Half Add of First bit and Second bit
    addFirstSecond = HalfAdder().Add(First, Second)
    # Sum
    self.tempSum = addFirstSecond[0]
    self.Sum = HalfAdder().Add(self.tempSum,CarryIn)[0]

    # Carry Out
    self.firstCout = addFirstSecond[1]
    self.secondCout = HalfAdder().Add(self.tempSum, CarryIn)[1]
    self.Cout = self.firstCout | self.secondCout

    return self.Sum, self.Cout

# Contoh pemanggilan
Adder = FullAdder()
# Count: 0 add 0, Carry in = 0
sum = Adder.Add(0, 0, 0)
```

Program di atas adalah program untuk menghitung penjumlahan *full adder* dengan menggunakan operasi logika atau *bitwise operator* dan rangkaian *half adder* saja serta telah dirancang sesuai dengan pemodelan rangkaian gerbang logika *full adder* pada bab III. Program tersebut menerima tiga buah parameter berupa 1-bit bilangan biner pertama, 1-bit bilangan biner kedua, dan 1-bit bilangan biner *Carry In*. Program tersebut akan mengolah masukan dan mengembalikan 2-*tuple*, yaitu (*SUM, Carry Out*).

```
from fullAdder import FullAdder

class ParallelAdder:
  def __init__(self) -> None:
    self.totalDigit = 0
    self.tempDigit = 1
    self.Sum = ""
    self.Cin = 0

  def Add(self, First:str, Second:str, totalBit:int):
    self.totalDigit = totalBit

    # merubah input binary menjadi bentuk n-bit dengan n adalah totalDigit yang dimasukan
    First = First.rjust(self.totalDigit,'0')
    Second = Second.rjust(self.totalDigit,'0')

    # menghitung sum per bit dimulai dari bit paling kanan menggunakan FullAdder
    self.Sum += str(FullAdder().Add(int(First[self.totalDigit-self.tempDigit]),int(Second[self.totalDigit-self.tempDigit]),self.Cin)[0])

    # menghitung carry out dari perhitungan bit diatas
    self.Cin = FullAdder().Add(int(First[self.totalDigit-self.tempDigit]),int(Second[self.totalDigit-self.tempDigit]),self.Cin)[1]

    # menggeser pencacahan bit ke kiri sebanyak 1
    self.tempDigit += 1

    # jika pencacahan telah melebihi total digit maka akan mengembalikan nilai sum
    if self.tempDigit > self.totalDigit:
      self.Sum = self.Sum[::-1]
      return self.Sum

    # jika pencacahan masih belum melebihi total digit maka perhitungan masih dilajutkan
    else:
      self.Add(First, Second, totalBit)
      return self.Sum

# Contoh pemanggilan
n = int(input("Masukkan jumlah bit: "))
First = int(input("Masukan angka ke-1: "))
Second = int(input("Masukan angka ke-2: "))
# Ubah integer menjadi binary
First = str(bin(First)[2::]).rjust(n,"0")
Second = str(bin(Second)[2::]).rjust(n,"0")
Sum = ParallelAdder().Add(First, Second, n)
```

Program di atas adalah program untuk menghitung penjumlahan beberapa bit bilangan atau disebut dengan *parallel adder*. Program tersebut hanya menggunakan operasi logika atau *bitwise operator* dan rangkaian *full adder* saja serta telah dirancang sesuai dengan pemodelan rangkaian gerbang logika *full adder* pada bab III. Program tersebut menerima tiga buah parameter berupa **n**-bit bilangan biner pertama, **n**-bit bilangan biner kedua, dan jumlah bit yang digunakan (**n**). Program tersebut akan mengolah masukan dan mengembalikan nilai penjumlahan kedua bilangan bilangan biner dalam bentuk **n**-bit bilangan biner.

```
from halfAdder import HalfAdder
from fullAdder import FullAdder

class fourBitProduct:
  def __init__(self) -> None:
    pass

  def mulEach(self, base:str, bin:int):
    s = ""
    for i in range(4):
        s += str(int(base[i]) & bin)
    return s

  def multiply(self, First:str, Second:str):
    # multiply each
    m1 = self.mulEach(First, int(Second[3]))
    m2 = self.mulEach(First, int(Second[2]))
    m3 = self.mulEach(First, int(Second[1]))
    m4 = self.mulEach(First, int(Second[0]))

    # find bit value
    bit1 = m1[3]

    temp =
HalfAdder.Add(self,int(m1[2]),int(m2[3]))
    bit2 = temp[0]

    temp1 =
FullAdder.Add(self,int(m1[1]),int(m2[2]),int(temp
[1]))
    temp2 =
HalfAdder.Add(self,int(temp1[0]),int(m3[3]))
    bit3 =  temp2[0]

    bit4 = temp1[1] ^ temp2[1]
    bit4 = bit4 ^ int(m1[0])
    bit4 = bit4 ^ int(m2[1])
    bit4 = bit4 ^ int(m3[2])
    bit4 = bit4 ^ int(m4[3])

    return (str(bit4) + str(bit3) + str(bit2) +
str(bit1))

# Contoh pemanggilan
n = int(input("Masukkan jumlah bit: "))
First = int(input("Masukkan angka ke-1: "))
Second = int(input("Masukkan angka ke-2: "))
# Ubah integer menjadi binary
First = str(bin(First)[2::]).rjust(n,"0")
Second = str(bin(Second)[2::]).rjust(n,"0")
Result = fourBitProduct().multiply(First,Second)
```

Program di atas adalah program untuk menghitung perkalian 4-bit bilangan biner atau disebut dengan *4-bit product.* Program tersebut telah dirancang hanya menggunakan operasi logika atau *bitwise operator* dan rangkaian *half adder* serta rangkaian *full adder* saja. Program tersebut juga telah dirancang sesuai dengan pemodelan rangkaian gerbang logika *4-bit product* pada bab III. Program tersebut menerima dua buah parameter berupa 4-bit bilangan biner pertama dan 4-bit bilangan biner kedua. Program tersebut akan mengolah masukan dan mengembalikan nilai perkalian kedua bilangan bilangan biner dalam bentuk 4-bit bilangan biner.

Setelah membuat program implementasi rangkaian gerbang logika, maka bisa dilakukan pengetesan dengan menggunakan beberapa unit tes yang bisa dilihat secara lengkap pada https://bit.ly/SourceCodeRGL. Hasil dari tes adalah sebagai berikut.

```
....
----------------------------------------
Ran 4 tests in 0.006s

OK
```

Hal tersebut membuktikan bahwa program implementasi rangkaian gerbang logika tersebut sudah benar dan begitu juga dengan rangkaian gerbang logika nya.

V. KESIMPULAN

Kalkulator analog masih tersusun atas beberapa komponen yang menggunakan rangkaian gerbang logika. Beberapa rangkaian gerbang logika pada kalkulator analog, seperti rangkaian logika penjumlahan, rangkaian logika pengurangan, dan rangkaian logika perkalian dapat disusun secara baik dan efisien dengan bantuan keilmuan aljabar Boolean. Selain berperan dalam penyusunan rangkaian gerbang logika, keilmuan aljabar Boolean juga berperan dalam penyederhanaan rangkaian gerbang logika, sehingga penggunaan gerbang logika dapat seminimal mungkin. Dengan terciptanya rangkaian gerbang logika yang tepat dan sederhana, maka banyak pengaruh positif yang timbul dari perkembangan alat elektronik atau komponen elektronik, yaitu harga beli yang semakin murah, ukuran yang semakin kecil, berat yang semakin ringan, dan masih banyak lagi.

VI. UCAPAN TERIMA KASIH

Puji Syukur penulis panjatkan kepada Tuhan Yang Maha Esa yang telah memberi berkat rahmat dan kasih karunia-Nya. Sehingga, penulis dapat menyelesaikan makalah yang berjudul “Aplikasi Aljabar Boolean dalam Operasi Perhitungan dengan Gerbang Logika” dengan baik dan selesai tepat waktu. Tak lupa juga penulis mengucapkan terima kasih kepada keluarga yang

telah memberikan dukungan sehingga penulis dapat menyelesaikan makalah ini. Terima kasih juga penulis sampaikan kepada dosen-dosen pengampuh mata kuliah IF2120 terutama kepada Dr. Ir. Rinaldi Munir, M.T. selaku dosen pengampuh mata kuliah IF2120 untuk kelas 03 karena telah membimbing penulis dan memberikan sumber belajar untuk bisa memahami keilmuan dalam makalah ini. Penulis berharap bahwa makalah ini nantinya dapat digunakan sebagai referensi baik bagi para pelajar yang penasaran dengan keilmuan terkait atau bahkan sebagai referensi penulis kedepannya.

DAFTAR PUSTAKA

[1] Wahyudi Raka Nur, "Organisasi Komputer: Makalah Gerbang Logika". http://informaraka.blogspot.com/2016/01/makalah-gerbang-logika.html, diakses pada tanggal 2 Desember 2023.

[2] Tim Dosen Universitas Mercu Buana, "Sistem Digital: Operasi Bilangan Biner", Chapter 2. https://ocw.upj.ac.id/files/Textbook-INF203-Buku-Sistem-Digital-for-TI.pdf, diakses pada tanggal 2 Desember 2023.

[3] Tim Dosen Universitas Mercu Buana, "Sistem Digital: Gerbang Logika Dasar", Chapter 3. https://ocw.upj.ac.id/files/Textbook-INF203-Buku-Sistem-Digital-for-TI.pdf, diakses pada tanggal 2 Desember 2023.

[4] Tim Dosen Universitas Mercu Buana, "Sistem Digital: Aljabar Boolean", Chapter 4. https://ocw.upj.ac.id/files/Textbook-INF203-Buku-Sistem-Digital-for-TI.pdf, diakses pada tanggal 2 Desember 2023.

[5] Rinaldi Munir, "Matematika Diskrit: Aljabar Boolean". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf diakses pada tanggal 2 Desember 2023

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.

Jatinangor, 9 Desember 2023

Valentino Chryslie Triadi 13522164