Aplikasi Aljabar Boolean pada Cyclic Redundancy Check

Fedrianz Dharma - 13522090[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13522090@itb.ac.id

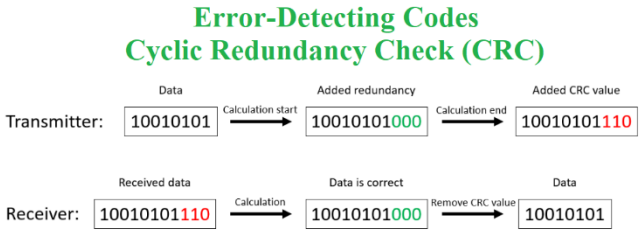
***Abstract*—*Cyclic Redundancy Check* adalah suatu teknik yang dapat digunakan untuk mendeteksi *errors* yang banyak digunakan dalam jaringan telekomunikasi digital. Dua cara utama untuk menghitung *Cyclic Redundancy Check*, yaitu *Polinomial Division* dalam GF(2) dan CRC *Shift Register*. Makalah ini membahas cara kerja dari *Cyclic Redundancy Check* secara umum dan aplikasi dari aljabar Boolean dalam *Cyclic Redundancy Check*, serta cara merancang rangkaian digitalnya dengan memanfaatkan gerbang logika dan *shift register*.**

***Keywords*—Aljabar Boolean, *Cyclic Redundancy Check*, *error-detecting code*, gerbang logika.**

I. PENDAHULUAN

Cyclic redundancy check (CRC) adalah suatu teknik yang dapat digunakan untuk mendeteksi *errors* (*error-detecting code*) yang ada pada data digital. William Wesley Peterson menciptakan teknik ini pada 1961 dan dikembangkan lebih lanjut oleh CCITT (Comité Consultatif International Telegraphique et Telephonique) [1]. CRC digunakan untuk memastikan integritas transmisi data dalam berbagai sistem telekomunikasi dan perangkat penyimpanan seperti HDD (*hard disk drives*).

CRC cukup populer karena mudah untuk diimplementasikan, tidak ada latensi, dan cocok untuk mendeteksi *burst errors*. CRC memanfaatkan kode polinomial untuk mendeteksi *error* atau perubahan yang terjadi pada data. Terdapat dua jenis pendekatan yang ada, yaitu *CRC Division* dan *CRC Shift Register*. *CRC Division* memanfaatkan pembagian polinomial pada GF(2) sedangkan *CRC Shift Register* memanfaatkan *linear-feedback shift register* dengan gerbang logika XOR.



Gambar 1 Ilustrasi cara kerja CRC
Sumber: https://www.knowelectronic.com/cyclic-redundancy-check/

Pada dasarnya, algoritma CRC bekerja dengan menambahkan N-1 bits 0 pada data yang ingin ditransmisikan. Kemudian data tersebut akan diproses untuk mendapatkan *CRC codes* atau biasa disebut *checksum*. *CRC codes* atau *checksum* ini akan ditambahkan pada bagian akhir dari data yang akan ditransmisikan. Penerima data akan memproses data tersebut dengan cara yang sama. Jika hasilnya adalah nol hal ini menunjukkan bahwa data tersebut aman dan tidak mengandung *errors*. Namun, jika hasilnya tidak nol hal ini menunjukkan bahwa data tersebut mengalami *errors*.

II. LANDASAN TEORI

A. *Aljabar Boolean*

Aljabar Boolean adalah salah satu bentuk aljabar yang memiliki nilai variable True atau False yang biasanya direpresentasikan oleh 1 untuk True dan 0 untuk False. Aljabar Boolean diperkenalkan oleh Geoge Boole pada 1847 melalui bukunya yang berjudul *The Mathematical Analysis of Logic* [2] dan dikemukakan secara lebih lengkap pada 1854 melalui *An Investigation of the Laws of Thought* [3].

Aljabar Boolean digunakan pada banyak bidang elektronik digital, seperti rangkaian pensaklaran, rangkaian digital, dan rangkaian *IC* (*integrated circuit)* komputer [4]. Aljabar boolean juga terdapat pada semua bahasa pemrogramman.

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

1. Identitas
   (i) $a + 0 = a$
   (ii) $a \cdot 1 = a$
2. Komutatif
   (i) $a + b = b + a$
   (ii) $a \cdot b = b \cdot a$
3. Distributif
   (i) $a \cdot (b + c) = (a \cdot b) + (a \cdot c)$
   (ii) $a + (b \cdot c) = (a + b) \cdot (a + c)$
4. Komplemen
   Untuk setiap $a \in B$ terdapat elemen unik $a' \in B$ sehingga
   (i) $a + a' = 1$
   (ii) $a \cdot a' = 0$

Aljabar Boolean juga memiliki beberapa hukum yang dapat

dilihat pada tabel di bawah.

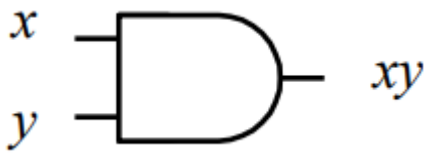
Tabel 1 Hukum-Hukum Aljabar Boolean [4]

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) a · a' = 0	Hukum dominansi: (i) a · 0 = 0 (ii) 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) a · b = b · a	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	

B. Gerbang Logika

Gerbang logika adalah suatu rangkaian elektronika yang berperan penting pada sebuah sirkuit digital. Sebagaian besar gerbang logika menerima dua sinyal input dan mengeluarkan satu sinyal output. Gerbang logika hanya dapat menghasilkan dua jenis output, yaitu False atau 0 dan True atau 1. Gerbang logika dapat diibaratkan seperti saklar lampu, pada satu posisi outputnya adalah OFF atau 0 dan pada posisi lain outputnya adalan ON atau 1 [4]. Terdapat 3 gerbang logika dasar, yaitu AND, OR, NOT dan 4 gerbang logika turunan, yaitu NAND, NOR, XOR, dan XNOR.

1. Gerbang AND



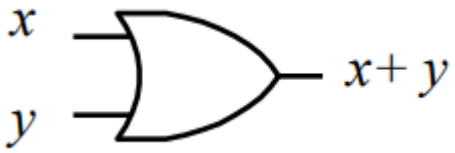
Gambar 2.1 Gerbang AND
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf

Gerbang logika AND akan menghasilkan output 1 jika semua inputnya adalah 1 dan menghasilkan output 0 jika salah satu inputnya adalah 0.

Tabel 2.1 Tabel hasil gerbang logika AND

Input		Output
X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

2. Gerbang OR



Gambar 2.2 Gebang OR
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf

Gerbang logika OR akan menghasilkan output 1 jika salah satu inputnya adalah 1 dan menghasilkan output 0 jika semua inputnya adalah 0.

Tabel 2.2 Tabel hasil gerbang logika OR

Input		Output
X	Y	XY
0	0	0
0	1	1
1	0	1
1	1	1

3. Gerbang NOT

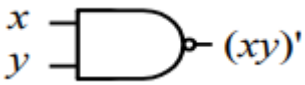
Gambar 2.3 Gerbang NOT
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf

Gerbang logika NOT akan menghasilkan output 1 jika inputnya adalah 0. Sebaliknya gerbang logika ini akan menghasilkan output 0 jika inputnya adalah 1.

Tabel 2.3 Tabel hasil gerbang logika NOT

Input	Output
X	X'
0	1
1	0

4. Gerbang NAND



Gambar 2.4 Gerbang NAND
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf

Gerbang logika NAND merupakan kombinasi dari gerbang logika dasar AND dan NOT. Gerbang logika NAND akan menghasilkan output 1 jika salah satu input adalah 0. Sebaliknya jika kedua input adalah 1 maka gerbang logika NAND akan menghasilkan output 0.

Tabel 2.4 Tabel hasil gerbang logika NAND

Input		Output
X	Y	XY
0	0	1
0	1	1
1	0	1
1	1	0

5. Gerbang NOR

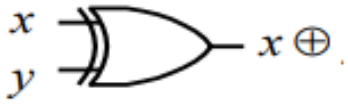
Gambar 2.5 Gerbang NOR
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf

Gerbang logika NOR merupakan kombinasi dari gerbang logika dasar OR dan NOT. Gerbang logika NOR akan menghasilkan output 1 jika kedua input adalah 0. Sebaliknya jika salah satu input adalah 1 maka gerbang logika NOR akan menghasilkan output 0.

Tabel 2.5 Tabel hasil gerbang logika NOR

Input		Output
X	Y	XY
0	0	1
0	1	0
1	0	0
1	1	0

6. Gerbang XOR



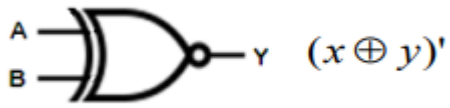
Gambar 2.6 Gerbang XOR
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf

Gerbang logika XOR akan menghasilkan output 1 jika kedua inputnya berbeda, misalnya input 1 dan 0 (atau 0 dan 1). Sebaliknya jika kedua inputnya adalah 0 atau kedua inputnya adalah 1 maka gerbang logika XOR akan menghasilkan output 0.

Tabel 2.6 Tabel hasil gerbang logika XOR

Input		Output
X	Y	XY
0	0	0
0	1	1
1	0	1
1	1	0

7. Gerbang XNOR



Gambar 2.7 Gerbang XNOR
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf

Gerbang logika XNOR merupakan kombinasi dari gerbang logika XOR dan NOT. Gerbang logika XNOR akan menghasilkan output 1 jika kedua inputnya sama, misalnya input 1 dan 1 (atau 0 dan 0). Sebaliknya jika kedua inputnya berbeda, misalnya 1 dan 0 (atau 0 dan 1) maka gerbang logika XNOR akan menghasilkan output 0.

Tabel 2.7 Tabel hasil gerbang logika XNOR

Input		Output
X	Y	XY
0	0	1
0	1	0
1	0	0
1	1	1

C. Cylic Redundancy Check

Sebelum mengimplementasikan CRC, hal pertama yang harus dilakukan adalah memilih *generator polinomial* yang akan digunakan. *Generator polinomial* yang dipilih akan memengaruhi efektivitas dan efisiensi algoritma CRC.

Algoritma CRC akan membagi *bits* pada data yang akan ditransmisikan *B*(x) dengan polinomial yang dipilih *G*(*x*) dan menghitung sisa hasil pembagiannya *R*(*x*). Panjang dari *R*(*x*) dalam *bits* selalu sama dengan panjang dari *G*(*x*) dalam *bits* dikurang 1. *R*(*x*) dalam *bits* akan disambungkan pada data *B*(*x*) sebelum ditransmisikan. Penerima akan memverivikasi data tersebut dengan melakukan pembagian modulo-2. Pertambahan dan pengurangan pada pembagian modulo-2 sama saja dengan operasi logika eksklusif OR (XOR). Jika sisa hasil pembagiannya adalah 0 hal ini menunjukkan tidak ada perubahan pada data. Namun, jika hasilnya adalah bukan 0 hal ini menunjukkan data telah mengalami perubahan atau *error*. Setelah terdeteksi adanya *error* pada data, penerima dapat meminta agar data dikirimkan ulang.

Misalkan data yang ingin ditransmisikan adalah 0xAB. CRC akan mengubah data tersebut menjadi biner 10101011. Misalkan *generator polinomial* yang dipilih adalah $1x^5+0x^4+0x^3+1x^2+0x^1+1x^0$, maka jika diubah ke dalam biner akan menjadi 100101. Data yang akan ditransmisikan akan menjadi *dividend* dan polinomial yang dipilih akan menjadi *divisor*. Sebelum dilakukan pembagian, data akan disambung dengan *n-1 bits* 0 (n adalah panjang *bits divisor*). Karena panjang *bits divisor* adalah 6 *bits* maka data akan disambung dengan 5 *bits* 0.

CRC polinomial divisor → 100101)$\overline{101010\,1\,100000}$ ← *dividend*
100101⊕ ← *XOR*
001111

Gambar 2.8 CRC Polinomial Division

100101)$\overline{1010101100000}$
100101
0011111100000
100101
01101000000
100101
0100010000
100101
000111000
100101
011101 ← *remainder*

Gambar 2.9 Gambar sisa hasil bagi dari data

Sisa hasil pembagian tersebut adalah *CRC bits* atau *checksum*. Sebelum data ditransmisikan, data akan disambung dengan *CRC bits* yang telah didapat. Inilah mengapa algoritma CRC dikatakan *"redundant"* karena data yang ditransmisikan akan bertambah menjadi lebih besar tanpa adanya informasi yang baru [1].

```
100101 ) 1010101111101  ← setelah ditambahkan CRC bits
         100101
         ------
         001111111101
           100101
           ------
           01101011101
            100101
            ------
            0100001101
             100101
             ------
             000100101
               100101
               ------
               000000   ← remainder
```

Gambar 2.10 Gambar sisa hasil bagi dari data setelah ditambahkan *CRC bits*

Gambar 2.10 menunjukkan bahwa data tidak mengalami *error* karena sisa hasil pembagiannya adalah 0. Jika data mengalami perubahan maka sisa hasil baginya tidak akan 0 seperti yang ditunjukkan pada gambar di bawah ini.

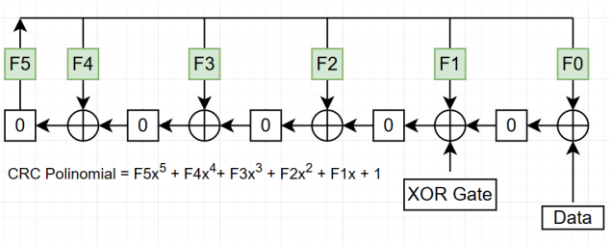
```
100101 ) 1011101111101
         100101
         ------
         0010111111101
           100101
           ------
           00101011101
            100101
            ------
            001110101
              100101
              ------
              0111111
               100101
               ------
               011010
```

Gambar 2.11 Gambar sisa hasil bagi dari data jika terjadi *error*

Pada gambar 2.11, data yang ditransmisikan mengalami perubahan pada *bit* ke-4 dari 0 menjadi 1. Dengan begitu ketika penerima melakukan pembagian polinomial pada data tersebut, sisa hasil baginya tidaklah nol.

III. Aplikasi Aljabar Boolean pada CRC

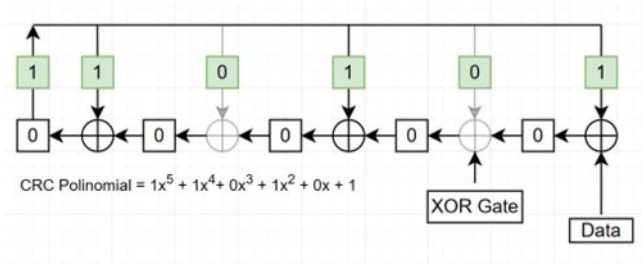
A. *Shift Register*



Gambar 3.1 Gambar pengaturan *shift register* untuk 6 *bits feedback polinomial*

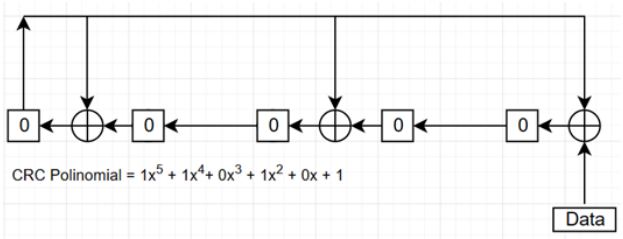
Selain dengan menggunakan teknik *polinomial division*, CRC juga dapat diimplementasikan dengan menggunakan *shift register* bertipe *LFSR*. *LFSR* merupakan *shift register* yang inputnya linier dengan hasil sebelumnya. Penyusunan komponen-kompenan *LFSR* untuk mengimplementasi CRC dapat dilakukan dengan meletakkan *register* dan gerbang XOR berselang-seling. Jumlah *register* yang dibutuhkan adalah panjang *polinomial generator* dalam *bits* dikurangi satu.

Gambar 3.1 menunjukkan tata penyusunan *register* dan gerbang XOR untuk 6 *bits* polinomial. Pada polinomial dengan panjang 6 *bits* terdapat 5 buah *register*. *Register* digunakan untuk menampung nilai 0/1 dari hasil gerbang XOR. *Register* digunakan menggeser nilai 0/1 ke sebelah kiri melalui gerbang XOR ke dalam *register* lain.



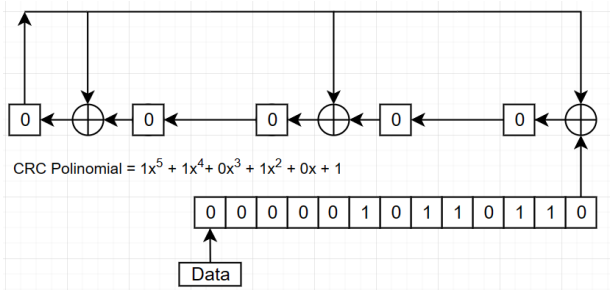
Gambar 3.2 Gambar pengaturan *gerbang XOR*

Jumlah gerbang XOR akan bergantung pada koefisien dari *polinomial generator* yang dipilih. Masing-masing gerbang XOR akan berkoresponden satu-satu dengan suku dari polinomial mulai dari pangkat kedua terbesar. Masing-masing *feedback input* ke dalam gerbang XOR akan dikalikan dengan koefisien dari polinomial pasangannya. Oleh karena itu gerbang XOR yang berkoresponden dengan 0 pada polinomial akan dihilangkan karena nilai *feedback inputnya* akan selalu 0 dan tidak berefek. Konfigurasi dari rangkaian akan terlihat seperti gambar di bawah ini.



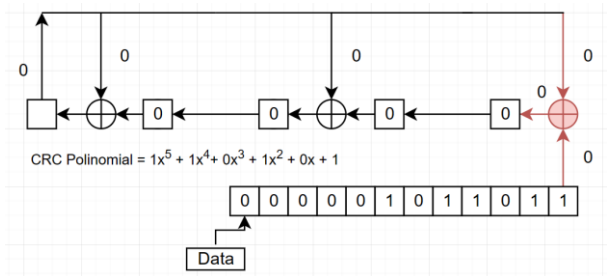
Gambar 3.3 Gambar konfigurasi rangkaian setelah meminimalisir *gerbang XOR*

B. *Ilustrasi*



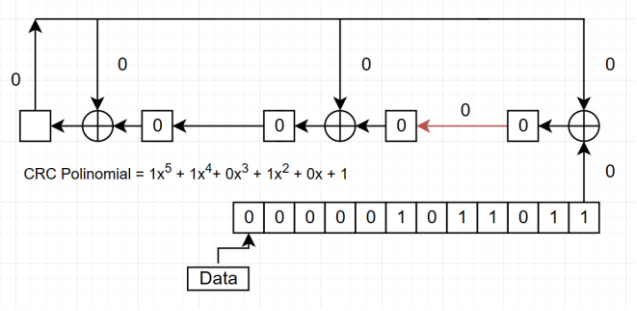
Gambar 3.4 Gambar contoh CRC dengan Shift Register

Misalkan data yang inigin ditransmisikan adalah 01101101. Pertama-tama, *register* akan dinisialisasi dengan nilai 0. Nilai dar *register* yang berada di paling kiri akan menjadi *feedback input* untuk gerbang XOR. Kemudian data akan disambung dengan 0 sejumlah panjang polinomial dalam *bits* dikurang 1 menjadi 0110110100000.



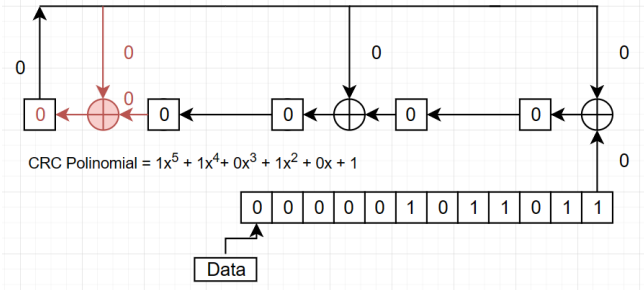
Gambar 3.5 Ilustrasi cara kerja CRC dengan Shift Register pertama kali mulai

Gambar 3.5 menunjukkan *bit* pertama dari data akan dikirimkan ke gerbang XOR palingn kanan dan melakukan XOR dengan *feedback input.* Hasil dari operasi tersebut akan menggeser nilai dari *register* pertama dan menempatinya.



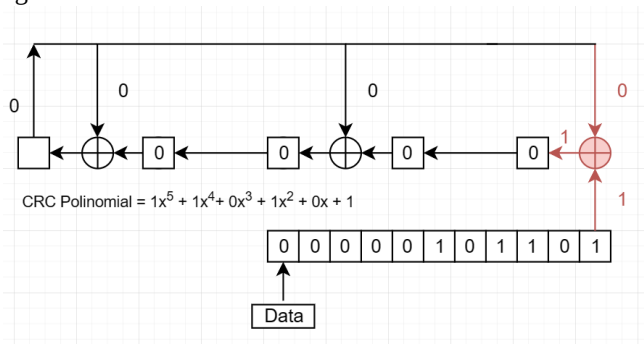
Gambar 3.6 Gambar pergeseran nilai dari *Register* ke *Register*

Nilai dari *register* pertama akan berpindah ke *register* yang kedua. Jika ada gerbang XOR di antara kedua *register*, maka akan dilakukan operasi XOR terlebih dahulu dengan *feedback input* sebelum berpindah.



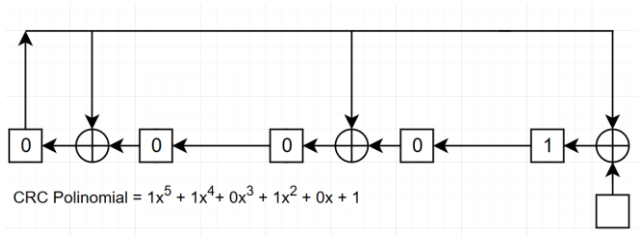
Gambar 3.7 Gambar dari *feedback input* yang baru

Hal ini akan terus dilakukan hingga *register* yang paling kiri mendapatkan nilai yang baru dari register sebelumnya. Nilai yang baru pada *register* paling kiri akan menjadi *feedback input* yang baru.



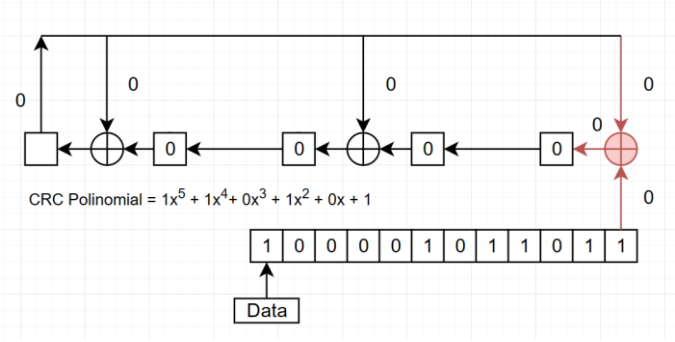
Gambar 3.8 Gambar CRC pada siklus yang kedua

Kemudian *bit* kedua dari data akan dikirimkan ke gerbang XOR yang paling kanan. Hal ini akan terus dilakukan hingga *bits* pada data yang akan ditransmisikan habis.



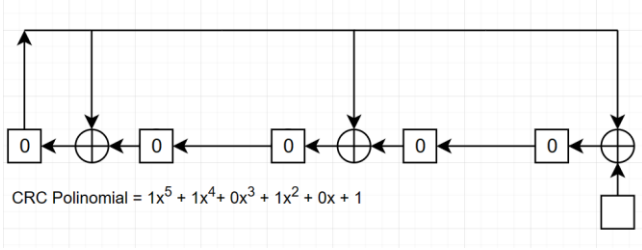
Gambar 3.9 Gambar keadaan CRC saat data habis

Nilai yang berada pada *register* saat *bits* pada data sudah habis akan menjadi *CRC bits* atau *checksum*. Nilai tersebut akan disambungkan pada *bits* data awal. Data yang akan ditransmisikan menjadi 0110110100001.



Gambar 3.10 Gambar CRC pada penerima data

Bit pertama dari data yang diterima akan dikirimkan ke gerbang XOR paling kanan dan melakukan XOR dengan *feedback input*. Hasil dari operasi tersebut akan menggeser nilai dari *register* pertama dan menempatinya. Algoritma CRC pada penerima data akan melakukan hal yang sama seperti pengirim hingga data yang diterima telah habis.



Gambar 3.10 Gambar CRC pada penerima data saat data habis dan tidak terjadi *error*

Hal yang berbeda adalah nilai yang berada pada *register* pada saat *bits* pada data sudah habis akan dicek apakah bernilai nol atau tidak. Jika bernilai nol hal itu menunjukkan bahwa data aman dan tidak ada *error*. Sebaliknya, jika bernilai tidak nol hal itu menunjukkan bahwa pada data telah terjadi *error* dan penerima bisa meminta agar data dikirim ulang.

IV. KESIMPULAN

Cyclic Redundancy Check adalah salah satu dari banyak macam *error-detecting code* yang banyak digunakan pada jaringan telekomunikasi digital untuk mendeteksi adanya *error* pada data secara cepat dan efisien. Dengan menerapkan prinsip aljabar Boolean, CRC dapat dihitung dengan menggunakan operasi logika XOR dan dapat diimplementasikan ke dalam bentuk rangakaian digital dengan menggunakan gerbang XOR dan *shift register*.

V. UCAPAN TERIMA KASIH

Terima kasih kepada Tuhan Yang Maha Esa karena dengan berkat rahmat dan anugerah-Nya penulis dapat menyelesaikan makalah ini dengan baik, tanpa kendala, dan tepat waktu. Penulis juga ingin mengucapkan terima kasih kepada para dosen pengampu Mata Kuliah Matematika Diskrit Semester I tahun 2023/2024, terutama Ibu Dr. Fariska Zakhralativa Ruskanda, S.T., M.T. selaku dosen pengampu Mata Kuliah Matematika Diskrit Semester I tahun 2023/2024 Kelas 02 atas bimbingan, pengajaran, dan dukungannya selama satu semester ini. Tak lupa penulis juga ingin mengucapkan terima kasih kepada orang tua yang senantiasa memberikan dukungan, serta seluruh pihak yang telah membantu dan mendukung penulis dalam menyelesaikan makalah ini.

REFERENSI

[1] M. Rouse, "*Cyclic Redundancy Check*". What is Cyclic Redundancy Check (CRC)? – Definition from Techopedia. Available: https://www.techopedia.com/definition/1793/cyclic-redundancy-check-crc (accessed Dec. 8, 2023).

[2] G. Boole. *The Mathematical Analysis of Logic Being an Essay Towards a Calculus of Deductive Reasoning*. (1847; Project Gutenberg, 2011). Available: https://www.gutenberg.org/ebooks/36884 (accessed Dec. 8, 2023).

[3] G. Boole. *An Investigation of the Laws of Thought*. (1854; Project Gutenberg 2005). Available: https://www.gutenberg.org/ebooks/15114 (accessed Dec. 8, 2023).

[4] Munir, Rinaldi. *Aljabar Boolean (Bagian 1)*. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2023-2024/11-Aljabar-Boolean-(2023)-bagian1.pdf (accessed Dec. 8, 2023).

[5] TechTarget Contibutor, "*What is Logic Gate (and, or, XOR, not, Nand, nor and xnor)?: Definition from TechTarget."* WhatIs. Available at: https://www.techtarget.com/whatis/definition/logic-gate-AND-OR-XOR-NOT-NAND-NOR-and-XNOR (Accessed: 08 December 2023).

[6] Weimich, *"CRC lookup table, fast CRC without table, reverse CRC. C code."* Digital Signal Processing and Software. Available: https://www.dsp-weimich.com/programming/cyclic-redundancy-check-crc-bitwise-lookup-table-fast-crc-without-lookup-table-reversing-crc-c-implementation-using-the-octave-gnu-tool/ (accessed Dec. 4, 2023).

[7] Phil Koopman. *L600 CRC Operation Examples: Division, Shift Register and Both Compared*. (Jan 1, 2023). [Online video]. Available: https://www.youtube.com/watch?v=qRqvdOAfxcA&ab_channel=PhilKoopman (accessed Dec. 4, 2023).

[8] Wisc-Online. *CRC - Cyclic Redundancy Check*. (Jun 22, 2016). [Online video]. Available: https://www.youtube.com/watch?v=iwj8ZgyzqZk&t=289s&ab_channel=Wisc-Online (accessed Dec. 4, 2023).

[9] T.Schmidt, Microchip Technology Inc. *CRC Generating and Checking*. Microchip. https://ww1.microchip.com/downloads/en/AppNotes/00730a.pdf (accessed Dec. 4, 2023).

[10] Basar. "Error-Detecting Codes – cyclic redundancy check (CRC)." Explain Cyclic Redundancy Check – Error Detecting Codes. Available: https://www.knowelectronic.com/cyclic-redundancy-check/ (accessed Dec. 9, 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.

Bandung, 11 Desember 2023

Fedrianz Dharma 13522090