IT-EXPLORE
Jurnal Penerapan Teknologi Informasi dan Komunikasi
ISSN 2828-7940, e-ISSN 2829-1727
Volume 03 Nomor 01 Tahun 2024

PERANCANGAN DAN IMPLEMENTASI *ENCODER* DAN *DECODER* CRC-16 BERBASIS TABEL *LOOKUP* PADA ARDUINO

Prayogo Pangestu Pantow[1], Theophilus Wellem[2]

[1,2] Program Studi Teknik Informatika, Fakultas Teknologi Informasi
Universitas Kristen Satya Wacana
Jl. Dr. O. Notohamidjojo No. 1-10, Blotongan, Salatiga, Jawa Tengah, 50715
Email: 672018081@student.uksw.edu[1], theophilus.wellem@uksw.edu[2]

Riwayat artikel:
Submitted: 17-01-2024 *Revised: 14-02-2024* *Published: 18-02-2024*

Abstrak – Deteksi kesalahan pada data yang dikirimkan melalui saluran komunikasi sangat penting untuk mendapatkan data yang akurat dan sistem komunikasi yang handal. Salah satu kode yang populer dan banyak digunakan sebagai kode pendeteksi kesalahan adalah *Cyclic Redundancy Check* (CRC). Penelitian ini bertujuan untuk merancang dan mengimplementasikan *encoder* dan *decoder* CRC-16 dengan polinomial 0x8005 pada *platform* Arduino IoT. Penghitungan nilai CRC-16 dilakukan menggunakan algoritma yang memanfaatkan tabel pencarian. Algoritma berbasis tabel pencarian dipilih karena kecepatannya dalam menghitung nilai CRC dari data masukan. Hasil implementasi *encoder* dan *decoder* kemudian diuji menggunakan dua *board* Arduino yang mengirimkan data secara serial. Hasil penelitian menunjukkan bahwa *decoder* pada sisi penerima dapat mendeteksi kesalahan pada data yang dikirimkan, dan *encoder* pada sisi pengirim dapat menghitung nilai CRC dari data masukan dengan benar sesuai spesifikasi kode CRC-16. Waktu yang dibutuhkan *encoder* untuk menghitung nilai CRC-16 dari data input dengan panjang yang bervariasi menunjukkan bahwa data input dengan panjang 2 karakter (16 bit) dan 128 karakter (1024 bit) masing-masing membutuhkan waktu 0,016 milidetik dan 0,72 milidetik.

Kata Kunci – CRC-16, Error Detection Code, Error Detection System, Lookup Table, Arduino.

Abstract – *Error detection in data transmitted through communication channels is essential in achieving accurate data and reliable data communication systems. One code that is popular and widely used as an error detection code is the Cyclic Redundancy Check (CRC). This study aims to design and implement a CRC-16 encoder and decoder with a polynomial of 0x8005 on the Arduino IoT platform. Calculating the CRC-16 value is done using an algorithm that utilizes a lookup table. The lookup table-based algorithm was chosen because of its speed in calculating the CRC value from the input data. The results of the encoder and decoder implementation are then tested using two Arduino boards that transmit data serially. The experiment results show that the decoder on the receiving side can detect errors in the transmitted data, and the encoder on the sending side can calculate the CRC value of the input data correctly according to the CRC-16 code specification. Furthermore, the time required by the encoder to calculate the CRC-16 value from input data with varying lengths shows that input data with a length of 2 characters*

(16 bits) and 128 characters (1024 bits) takes 0.016 milliseconds and 0.72 milliseconds, respectively.

Keywords – *CRC-16, Error Detection Code, Error Detection System, Lookup Table, Arduino*.

I. PENDAHULUAN

Proses pengiriman data melalui jaringan komunikasi dapat mengalami kesalahan (*error*) yang diakibatkan oleh gangguan pada media transmisinya (*communication channel*). Gangguan ini umumnya berasal dari *noise* dan interferensi pada *channel* komunikasi tersebut. Oleh karena itu, perangkat yang terlibat dalam proses komunikasi data harus dapat melakukan deteksi atau koreksi terhadap *error* yang terjadi pada data [1][2]. Untuk memastikan keakuratan data dalam proses komunikasi, kode *Cyclic Redundancy Check* (CRC) umumnya digunakan dalam transmisi data, perangkat penyimpanan (*storage device*), dan memori untuk mendeteksi adanya perubahan bit yang tidak disengaja (*bit flipping*) pada data yang ditransmisikan atau disimpan.

Cyclic Redundancy Check (CRC) merupakan kode pendeteksi kesalahan di mana pada pesan atau data yang ditranmisikan dari pengirim (*transmitter*) ke penerima (*receiver*) ditambahkan sejumlah *redundant* bit yang dikenal sebagai CRC *checksum*. Jumlah *redundant* bit yang ditambahkan pada data ini secara umum memberikan nama untuk kode CRC tersebut, misalnya CRC-8, CRC-16, CRC-32, dan sebagainya. *Receiver* kemudian akan menggunakan *redundant* bit ini untuk mendeteksi apakah terdapat *error* pada data yang diterima dari *transmitter*. Kode CRC didefinisikan menggunakan suatu polinomial yang disebut dengan *generator polynomial* $g(x)$. Dengan *generator polynomial* ini dapat dibuat berbagai kode CRC dengan jumlah bit yang sama tetapi mempunyai *generator polynomial* yang berbeda. Sebagai contoh, kode CRC-16 dengan *generator polynomial* $x^{16} + x^{12} + x^5 + 1$ (dikenal sebagai CRC-16-CCITT) dan CRC-16 dengan dengan generator polynomial $x^{16} + x^{15} + x^2 + 1$ (dikenal sebagai CRC-16-ANSI). Kedua kode CRC ini mempunyai panjang atau jumlah bit yang sama yaitu 16 bit, tetapi memiliki nilai CRC yang berbeda untuk data yang sama. CRC-16-CCITT digunakan sebagai pendeteksi *error* pada komunikasi XMODEM, sedangkan CRC-16-ANSI digunakan pada komunikasi *Universal Serial Bus* (USB) [3].

Dengan semakin berkembangnya pemanfaatan teknologi *Internet-of-Things* (IoT), di mana transmisi data secara berkala dilakukan dari sensor ke IoT *board* (misalnya, Arduino) dan ke Internet, penelitian mengenai teknik perhitungan kode CRC yang cepat dan efisien serta implementasinya untuk perangkat-perangkat IoT terus dilakukan [4]. Penelitian mengenai metode perhitungan CRC penting untuk dilakukan karena kecepatan *link* (*link rate*) jaringan terus meningkat dari Gigabit per detik menuju Terabit per detik. Peningkatan *link rate* berarti jumlah *packet* yang harus diproses setiap detik akan semakin banyak. Oleh karena kode CRC digunakan untuk deteksi error pada setiap *packet* yang dikirimkan, maka kecepatan komputasi CRC juga harus ditingkatkan agar tidak menjadi bottleneck pada pemrosesan frame di *Layer* 2 (*data link layer*). Untuk perhitungan CRC, terdapat dua algoritma yang umum digunakan, yaitu menggunakan operasi SHIFT dan XOR (konvensional) dan algoritma menggunakan tabel *lookup*. Perhitungan CRC pada sisi *transmitter* dan *receiver* harus dilakukan secara cepat sesuai dengan kebutuhan (*link rate*) agar tidak menimbulkan *delay* yang besar dalam pemrosesan data. Penelitian ini bertujuan untuk merancang dan mengimplementasikan

encoder dan *decoder* untuk CRC-16 dengan polinomial 8005 (dalam notasi heksadesimal) atau $x^{16} + x^{15} + x^2 + 1$. Polinomial ini juga memiliki versi *reverse* atau *reflected*-nya yaitu A001 ($1 + x^2 + x^{15} + x^{16}$). Untuk implementasi perhitungan CRC-16 pada penelitian ini, teknik atau algoritma yang digunakan adalah perhitungan menggunakan tabel *lookup* (*lookup table*). Perhitungan nilai CRC menggunakan tabel *lookup* dikenal karena kecepatannya, walaupun membutuhkan konsumsi memori yang lebih besar untuk menyimpan tabel yang dibutuhkan untuk menghitung nilai CRC.

II. KAJIAN PUSTAKA

Beberapa penelitian terdahulu yang berkaitan dengan penelitian ini juga telah mengeksplorasi implementasi CRC sebagai kode pendeteksi *error* pada perangkat IoT [5] [4]. Penelitian oleh Novaldi, dkk. mengimplementasikan kode CRC-8 untuk mendeteksi *error* pada komunikasi serial Arduino [5]. Pada penelitian tersebut diukur waktu yang dibutuhkan untuk komputasi nilai CRC-8 dengan polinomial 0x31, 0xD5, dan 0x07. Pada penelitian oleh Chen, dkk, kode CRC diterapkan untuk pendeteksian error pada sistem IoT dengan *Bluetooth Low Energy* (BLE) sebagai standard komunikasinya [4]. Selain itu, penelitian oleh Saleh dan penelitian oleh Sridevi mengimplementasikan *encoder* dan *decoder* untuk CRC-8 pada Xilinx *Field Programmable Gate Array* (FPGA) [6][7]. Secara umum, implementasi CRC lebih banyak dilakukan menggunakan perangkat keras (*hardware*) dengan rangkaian *linear feedback shift register* (LFSR). Pilihan implementasi menggunakan perangkat keras ini dipilih karena umumnya komputasi CRC dilakukan pada *data link layer* (Layer 2) dalam suatu standard atau protokol komunikasi data. CRC-32 merupakan kode CRC yang paling umum dibahas pada berbagai penelitian karena kode ini digunakan sebagai *Frame Check Sequence* (FCS) pada suatu Ethernet *frame*. Penelitian oleh Augoestien mengimplementasikan CRC-32 untuk standard Ethernet pada Xilinx FPGA [8]. Implementasi pada penelitian tersebut hanya untuk bagian *encoder*-nya.

Penelitian ini merupakan pengembangan dari penelitian sebelumnya [9] yang telah melakukan implementasi *encoder* dan *decoder* untuk CRC-8 pada Arduino untuk mengeksplorasi implementasi kode CRC pada perangkat-perangkat IoT seperti Arduino, Raspberry Pi, dan sebagainya. Pada penelitian oleh Priyadi [9] diperoleh bahwa implementasi *encoder* dan *decoder* untuk CRC-8 dapat bekerja dengan baik pada *board* Arduino Uno R3. Waktu yang dibutuhkan oleh *encoder* untuk menghitung nilai CRC-8 (menggunakan teknik *lookup table*) pada input data sepanjang 128 karakter adalah 0.5 milidetik, sedangkan perhitungan nilai CRC-8 (menggunakan LFSR) membutuhkan waktu sebesar 2.37 milidetik. Pada penelitian ini, ruang lingkupnya dibatasi pada implementasi CRC-16 dengan *generator polynomial* $x^{16} + x^{15} + x^2 + 1$ pada *board* Arduino. CRC-16 dipilih karena komputasinya yang tidak sekompleks kode CRC dengan jumlah bit yang lebih banyak, misalnya CRC-32 dan CRC-64, sehingga memungkinkan untuk digunakan pada peralatan yang memiliki sumber daya komputasi dan kapasitas memori yang terbatas. Berbeda dengan penelitian sebelumnya, fokus pada penelitian ini adalah implementasi CRC-16 berbasis algoritma yang menggunakan *lookup table* pada Arduino Uno R3 karena penggunaan lookpup table menjanjikan perhitungan yang cepat dibandingkan algoritma perhitungan CRC yang lain, misalnya, dengan operasi SHIFT dan XOR.

Metode pendeteksian *error* pada transmisi data atau komunikasi data digital menggunakan kode CRC dapat dijelaskan secara singkat sebagai berikut: Pengirim data (*transmitter*) menghitung nilai CRC dari data yang akan dikirimkan berdasarkan algoritma yang telah ditetapkan. Perhitungan ini dilakukan oleh *encoder* dan menghasilkan suatu *checksum* (nilai CRC). *Checksum* ini kemudian disambungkan (*append*) dengan data dan dikirimkan ke penerima (*receiver*). *Decoder* pada penerima kemudian melakukan pemeriksaan apakah terjadi *error* pada data (termasuk *checksum*) yang diterima. Pemeriksaan ini dilakukan dengan cara mengeksekusi operasi pembagian (dalam aritmetika modulo-2) antara data yang diterima dengan *generator polynomial* yang digunakan untuk menghitung *checksum* pada *encoder*. Jika sisa hasil pembagian (*remainder*) ini tidak sama dengan nol, maka penerima akan menyimpulkan bahwa terdapat *error* pada data yang ditransmisikan dari pengirim. Sebaliknya, jika sisa hasil pembagian ini sama dengan nol, maka penerima dapat menyimpulkan bahwa tidak terdapat *error* pada data dan data tersebut dapat diproses lebih lanjut. Penjelasan lebih detail mengenai kode CRC dan penggunaannya dapat dilihat pada berbagai buku [1]–[3] dan penelitian sebelumnya [10]–[12].

III. METODE PENELITIAN

Metodologi yang digunakan dalam penelitian ini mengikuti *Systems Development Life Cycle* (SDLC) yang secara umum yang terdiri dari sejumlah tahapan yaitu, analisis kebutuhan (*requirements analysis*), perancangan dan implementasi (*design and implementation*), pengujian dan verifikasi sistem (*testing and verification*), dan perawatan sistem (*maintenance*) [13]. Tahapan-tahapan ini ditunjukkan pada Gambar 1 dan dapat dijelaskan sebagai berikut.

Gambar 1. Tahapan Penelitian

1. Analisis Kebutuhan: Pada tahap ini dilakukan analisis kebutuhan perangkat lunak (*software*) dan perangkat keras *(hardware*) untuk merancang dan mengimplementasikan *encoder* dan *decoder* CRC-16 pada Arduino. Pada tahap ini juga dilakukan studi literatur dari berbagai buku dan penelitian sebelumnya mengenai kode CRC dan implementasinya.
2. Perancangan dan Implementasi: Pada tahap ini algoritma untuk menghitung nilai CRC-16 diimplementasikan dalam bentuk perangkat lunak yang nantinya akan

diprogram atau diupload ke mikrokontroller yang ada pada *board* Arduino. Selain itu, koneksi antar perangkat keras juga dilakukan untuk mengimplementasikan sistem pendeteksi *error* pada transmisi data antar *board* Arduino. Dua perangkat lunak masing-masing *encoder* dan *decoder* diimplementasikan untuk diprogram ke mikrokontroler pada dua buah *board* Arduino. Penulisan kode program dilakukan menggunakan perangkat lunak Arduino IDE.

3. Pengujian dan Verifikasi: Pada tahap ini dilakukan pengujian dan verifikasi untuk memastikan apakah hasil implementasi dapat berjalan dengan baik dan hasil perhitungan nilai CRC-16 yang diperoleh adalah hasil yang tepat. Pengujian juga dilakukan untuk sistem pendeteksian *error* dengan cara mensimulasikan pengiriman data yang bebas dari *error* dan data yang mengalami *error*. Jika hasil implementasi sistem belum tepat, maka dilakukan modifikasi dan perbaikan hingga sistem dapat berjalan sesuai dengan fungsi yang diharapkan.
4. Perawatan Sistem: Pada tahap ini sistem telah dijalankan atau operasional. Jika ada yang fungsi-fungsi yang perlu ditambah, maka sistem akan diperbarui (*update*). Pada penelitian ini, tahap perawatan sistem ini tidak dilakukan karena *encoder* dan *decoder* yang dibuat belum atau tidak untuk disebarkan secara luas.

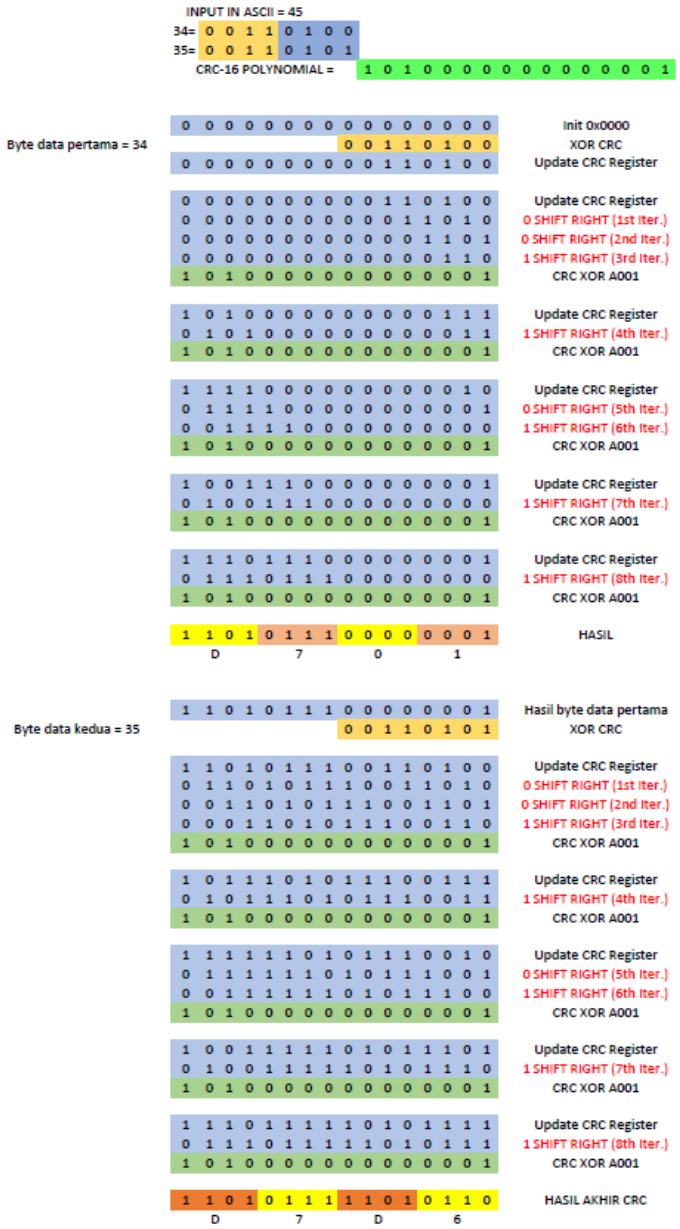
IV. HASIL DAN PEMBAHASAN

Perhitungan atau komputasi nilai CRC-16 dilakukan oleh *encoder* pada pengirim dan pendeteksian *error* pada data yang ditransmisikan dilakukan oleh *decoder* pada penerima. Pada penelitian ini, komputasi CRC dilakukan menggunakan algoritma yang memanfaatkan tabel *lookup*. Komputasi menggunakan tabel *lookup* dipilih karena kecepatannya dalam perhitungan CRC-16, meskipun membutuhkan memori yang lebih besar dibandingkan dengan teknik komputasi CRC-16 menggunakan LFSR. Pada bagian ini dijelaskan hasil perancangan dan implementasi *encoder* dan *decoder* untuk CRC-16 pada perangkat Arduino UNO R3 [14]. Sebelum menjelaskan komputasi CRC-16 dengan tabel *lookup*, akan diuraikan terlebih dahulu cara perhitungan konvensional CRC-16 menggunakan *shift register*.

A. Komputasi CRC-16 dengan SHIFT dan XOR

Nilai CRC-16 dari suatu input data dapat dihitung menggunakan algoritma yang melakukan manipulasi bit pada data dengan operasi SHIFT dan XOR (untuk implementasi dengan perangkat lunak). Algoritma tersebut dapat dijelaskan sebagai berikut. Pada awal proses perhitungan, variabel yang digunakan untuk menyimpan nilai CRC diinisialisasi (crc = 0x0000). Setelah itu, satu *byte* dibaca dari input data dan dilakukan operasi XOR antara input data tersebut dengan variabel untuk menyimpan nilai CRC (crc). Langkah berikutnya adalah memeriksa *least signicifant bit* (LSB) dari variabel crc. Jika LSB = ’1’, geser variabel crc sebanyak satu bit ke kanan (*right shift*) dan XOR-kan hasilnya dengan generator polynomial (A001). Sebaliknya, jika LSB = ’0’, geser variabel crc sebanyak satu bit ke kanan (*right shift*). Langkah ini dilakukan sebanyak delapan kali untuk satu *byte* data yang dibaca pada langkah sebelumnya. Selanjutnya, jika tidak terdapat *byte* selanjutnya pada input data, maka variabel crc menyimpan hasil akhir perhitungan CRC untuk input data tersebut. Sebaliknya, jika masih ada *byte* selanjutnya pada input data, maka *byte* tersebut dibaca dan proses diulang, hingga semua *byte* pada input data diproses.

Contoh perhitungan nilai CRC-16 untuk input data (*string*) ”45” atau 3435 dalam heksadesimal dengan hasil D7D6 ditunjukkan pada Gambar 2. Komputasi nilai CRC-16 dengan cara konvensional menggunakan *shift register* dan operasi XOR membutuhkan waktu yang lama jika input data yang diberikan cukup panjang (misalnya, 128 karakter atau 1024 bit). Oleh karena itu, dikembangkan berbagai algoritma atau metode lain untuk mempersingkat waktu perhitungan CRC. Salah satunya adalah algoritma yang menggunakan tabel *lookup*.



Gambar 2. Contoh perhitungan CRC-16 untuk input *string* “45”

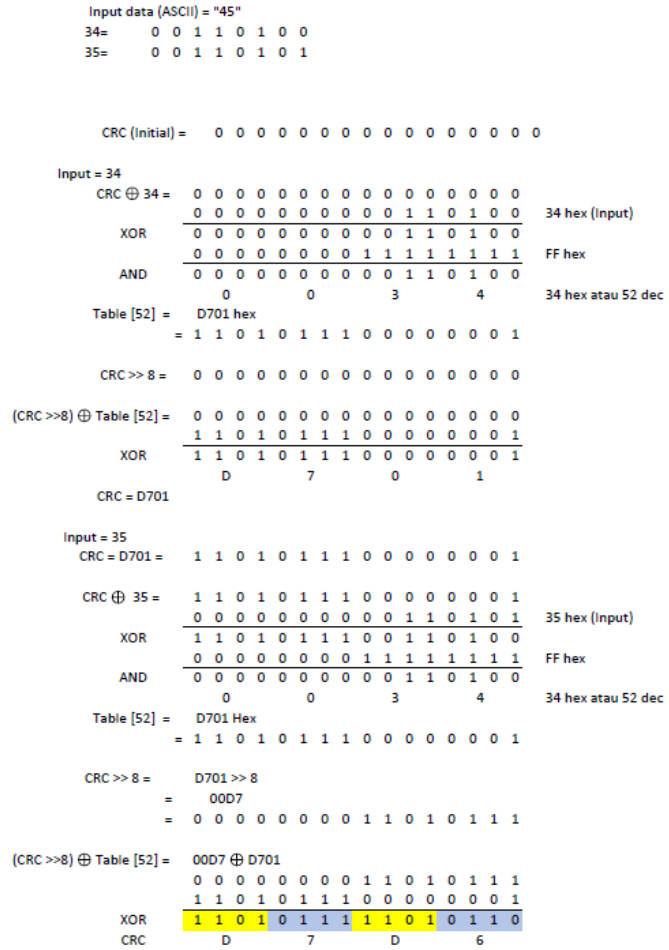
B. Komputasi CRC-16 dengan Tabel Lookup

Pada perhitungan nilai CRC-16 menggunakan tabel *lookup*, nilai-nilai pada tabel dihitung terlebih dahulu dan kemudian dipakai untuk mengisi tabel yang akan digunakan untuk memperoleh nilai CRC-16 dari input data. Komputasi nilai CRC-16 menggunakan tabel *lookup* memproses input data *byte* per *byte*. Saat tiap *byte* dibaca dari input data, nilainya diambil untuk mengindeks tabel *lookup*. Proses perhitungan nilai CRC-16 menggunakan tabel *lookup* adalah sebagai berikut. Pada awal proses perhitungan dilakukan inisialisasi variabel yang digunakan untuk menyimpan nilai CRC (crc = 0x0000). Langkah berikutnya adalah membaca satu *byte* dari input data dan melakukan operasi XOR antara variabel crc dengan *byte* yang dibaca tersebut. Selanjutnya, operasi AND dilakukan antara hasil dari operasi XOR pada langkah sebelumnya dengan nilai hex FF. Hasil operasi AND ini akan menghasilkan sebuah nilai yang akan digunakan sebagai indeks untuk mengindeks tabel *lookup*. Setelah mengambil nilai dari tabel, dilakukan operasi SHIFT ke kanan pada variabel crc sebanyak 8 bit dan hasilnya dioperasikan secara XOR dengan nilai yang diperoleh dari tabel. Hasil operasi XOR ini kemudian disimpan hasilnya ke variabel crc (hal ini merupakan proses *update* nilai CRC). Pada tahap ini, proses perhitungan nilai CRC telah selesai untuk 1 *byte* input data. Jika masih terdapat *byte* selanjutnya pada input data, maka *byte* tersebut dibaca dan proses perhitungan dieksekusi lagi. Hal ini dilakukan hingga seluruh *byte* pada input data diproses.

Index (Dec)	Value (Hex)	Index (Dec)	Value (Hex)	Index (Dec)	Value (Hex)	Index (Dec)	Value (Hex)	Index (Dec)	Value (Hex)	Index (Dec)	Value (Hex)	Index (Dec)	Value (Hex)	Index (Dec)	Value (Hex)
0	0000	32	D801	64	F001	96	2800	128	A001	160	7800	192	5000	224	8801
1	C0C1	33	18C0	65	30C0	97	E8C1	129	60C0	161	B8C1	193	90C1	225	48C0
2	C181	34	1980	66	3180	98	E981	130	6180	162	B981	194	9181	226	4980
3	0140	35	D941	67	F141	99	2940	131	A141	163	7940	195	5140	227	8941
4	C301	36	1B00	68	3300	100	EB01	132	6300	164	BB01	196	9301	228	4B00
5	03C0	37	DBC1	69	F3C1	101	2BC0	133	A3C1	165	7BC0	197	53C0	229	8BC1
6	0280	38	DA81	70	F281	102	2A80	134	A281	166	7A80	198	5280	230	8A81
7	C241	39	1A40	71	3240	103	EA41	135	6240	167	BA41	199	9241	231	4A40
8	C601	40	1E00	72	3600	104	EE01	136	6600	168	BE01	200	9601	232	4E00
9	06C0	41	DEC1	73	F6C1	105	2EC0	137	A6C1	169	7EC0	201	56C0	233	8EC1
10	0780	42	DF81	74	F781	106	2F80	138	A781	170	7F80	202	5780	234	8F81
11	C741	43	1F40	75	3740	107	EF41	139	6740	171	BF41	203	9741	235	4F40
12	0500	44	DD01	76	F501	108	2D00	140	A501	172	7D00	204	5500	236	8D01
13	C5C1	45	1DC0	77	35C0	109	EDC1	141	65C0	173	BDC1	205	95C1	237	4DC0
14	C481	46	1C80	78	3480	110	EC81	142	6480	174	BC81	206	9481	238	4C80
15	0440	47	DC41	79	F441	111	2C40	143	A441	175	7C40	207	5440	239	8C41
16	CC01	48	1400	80	3C00	112	E401	144	6C00	176	B401	208	9C01	240	4400
17	0CC0	49	D4C1	81	FCC1	113	24C0	145	ACC1	177	74C0	209	5CC0	241	84C1
18	0D80	50	D581	82	FD81	114	2580	146	AD81	178	7580	210	5D80	242	8581
19	CD41	51	1540	83	3D40	115	E541	147	6D40	179	B541	211	9D41	243	4540
20	0F00	52	D701	84	FF01	116	2700	148	AF01	180	7700	212	5F00	244	8701
21	CFC1	53	17C0	85	3FC0	117	E7C1	149	6FC0	181	B7C1	213	9FC1	245	47C0
22	CE81	54	1680	86	3E80	118	E681	150	6E80	182	B681	214	9E81	246	4680
23	0E40	55	D641	87	FE41	119	2640	151	AE41	183	7640	215	5E40	247	8641
24	0A00	56	D201	88	FA01	120	2200	152	AA01	184	7200	216	5A00	248	8201
25	CAC1	57	12C0	89	3AC0	121	E2C1	153	6AC0	185	B2C1	217	9AC1	249	42C0
26	CB81	58	1380	90	3B80	122	E381	154	6B80	186	B381	218	9B81	250	4380
27	0B40	59	D341	91	FB41	123	2340	155	AB41	187	7340	219	5B40	251	8341
28	C901	60	1100	92	3900	124	E101	156	6900	188	B101	220	9901	252	4100
29	09C0	61	D1C1	93	F9C1	125	21C0	157	A9C1	189	71C0	221	59C0	253	81C1
30	0880	62	D081	94	F881	126	2080	158	A881	190	7080	222	5880	254	8081
31	C841	63	1040	95	3840	127	E041	159	6840	191	B041	223	9841	255	4040

Gambar 3. Tabel *lookup* untuk perhitungan CRC-16

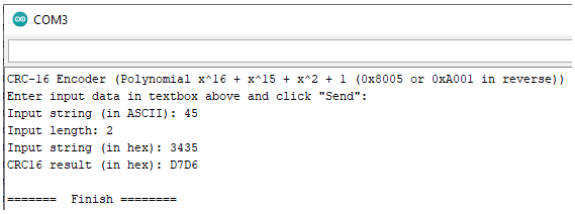
Gambar 3 menunjukkan tabel untuk perhitungan nilai CRC-16 dengan *generator polynomial* $x^{16} + x^{15} + x^2 + 1$ yang direfleksi $(1 + x^2 + x^{15} + x^{16}$, A001). Tabel *lookup* ini diimplementasikan pada kode sumber (*source code*) menggunakan struktur data *array*. Contoh perhitungan nilai CRC-16 untuk input data (*string*) ”45” menggunakan tabel *lookup* ditunjukkan pada Gambar 4. Tabel ini berisi 256 angka 16-bit (2 *byte*) sehingga ruang memori yang digunakan untuk menyimpan tabel ini adalah sebesar 512 *byte*.



Gambar 4. Contoh perhitungan menggunakan tabel *lookup*

C. Implementasi Perangkat Lunak

Penulisan kode sumber untuk *encoder* dan *decoder* dilakukan menggunakan bahasa C/C++ pada Arduino IDE. Hasil implementasi *encoder* CRC-16 ditunjukkan pada Gambar 5.



Gambar 5. Hasil Implementasi *Encoder* CRC-16 pada Arduino

Seperti yang dapat dilihat pada Gambar 5, pengguna memasukkan data (dalam karakter ASCII) yang akan dihitung nilai CRC-nya melalui *Serial Monitor* pada Arduino IDE (di sini digunakan *string* ”45” sebagai contoh input). Hasil perhitungan CRC-nya adalah D7D6 dan diverifikasi kebenarannya menggunakan kalkulator CRC [15]. Karena hasil dari kalkulator CRC juga menghasilkan D7D6, maka dapat disimpulkan bahwa implementasi *encoder* telah sesuai. Verifikasi ini juga dilakukan untuk berbagai macam

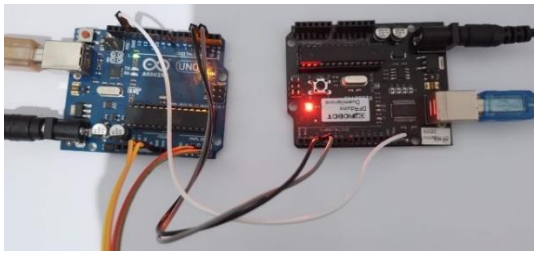
input data dan diperiksa hasilnya menggunakan kalkulator CRC. Hasil yang diperoleh dari *encoder* dan kalkulator menunjukkan angka yang sama.

Pada *board* Arduino yang berperan sebagai pengirim, selain *encoder*, diimplementasikan juga fungsi untuk mengirimkan data berserta nilai CRC yang telah dihitung menggunakan *library SoftwareSerial*. Implementasi *decoder* pada *board* Arduino yang berperan sebagai penerima mirip dengan implementasi *encoder* pada sisi pengirim. Hal ini karena baik *encoder* maupun *decoder*, keduanya memanfaatkan tabel *lookup* yang sama untuk melakukan operasi pembagian antara input yang diberikan dengan *generator polynomial* yang digunakan. Perbedaannya terletak pada input yang diterima oleh *encoder* dan *decoder*. Pada *encoder*, input-nya adalah data yang akan dihitung nilai CRC-nya, sedangkan pada *decoder*, input-nya adalah data beserta nilai CRC dari data tersebut yang diterima dari pengirim. Sebagai contoh, *encoder* menerima input string ”45” (atau 3435 dalam heksadesimal) dan *decoder* menerima input string ”45×Ö” atau 3435D7D6 dalam heksadesimal (jika tidak terdapat *error* pada data saat transmisi).

Berdasarkan hasil kompilasi program, implementasi *encoder* CRC-16 yang menggunakan tabel *lookup* pada *board* Arduino UNO R3 (tidak termasuk bagian untuk pengiriman data secara serial) membutuhkan ruang penyimpanan program sebesar 4560 *byte* dari 32256 *byte* (14%) memori yang tersedia. Untuk keseluruhan program pada sisi pengirim, memori yang dibutuhkan adalah 7940 *byte* atau 24% dari kapasitas memori yang tersedia. Program pada sisi penerima (bagian penerimaan data serial dan *decoder*) menggunakan 7022 *byte* dari 30720 *byte* (22%) memori yang tersedia.

D. Implementasi Perangkat Keras

Implementasi pada perangkat keras dilakukan menggunakan dua *board* Arduino (Arduino UNO R3 dan DFRduino Duemilanove 328) yang dihubungkan untuk berkomunikasi secara serial. Pin yang digunakan sebagai Rx dan Tx masing-masing adalah pin 2 dan pin 3. Hasil implementasi ditunjukkan pada Gambar 6.

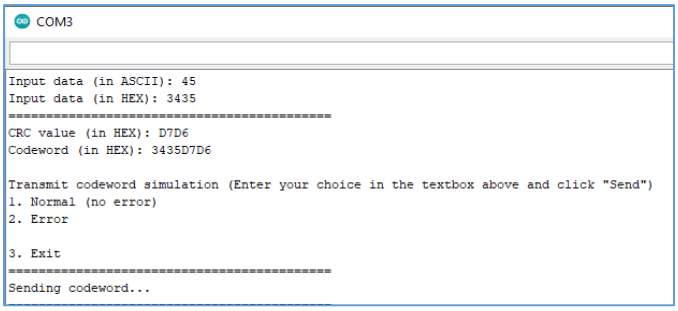


Gambar 6. Hasil Implementasi Perangkat Keras

E. Pengujian Sistem Pendeteksi Error

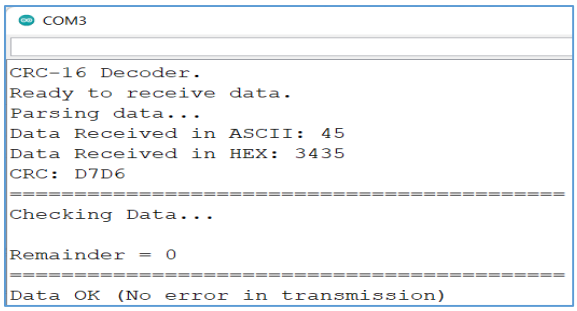
Encoder dan *decoder* CRC-16 yang telah diimplementasikan digunakan untuk sistem pendeteksi *error* pada transmisi data antar dua Arduino *board*. *Board* Arduino Uno R3 berperan sebagai pengirim data yang melakukan *encoding* terhadap data, sedangkan *board* DFRduino Duemilanove 328 berperan sebagai penerima yang melakukan *decoding* terhadap data yang diterima dan memeriksa apakah terdapat *error* pada data tersebut. Pengujian sistem pendeteksi *error* dilakukan menggunakan dua skenario yaitu, transmisi data tanpa error dan transmisi data dengan *error*. *Error* disimulasikan dengan cara

mengubah data yang ditransmisikan sehingga tidak sesuai dengan nilai CRC-16 dari data tersebut. Pengujian transmisi data dan pendeteksian *error* ditunjukkan pada Gambar 7 dan Gambar 8 di bawah ini.



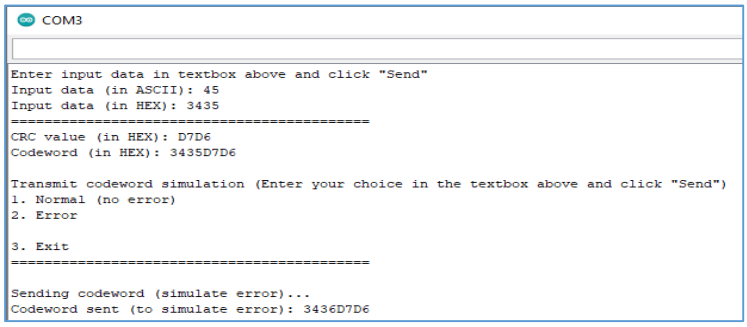
Gambar 7. Pengujian sistem (transmisi tanpa *error*) – Pengirim

Pada Gambar 7, pengguna memasukkan data yang akan ditransmisikan pada *Serial Monitor* dan kemudian menekan tombol ”Send”. Nilai CRC dari data kemudian dihitung dan pengguna akan diberikan pilihan lagi untuk mengirimkan data tanpa *error* atau dengan *error*. Hal ini untuk mensimulasikan dua kondisi dalam transmisi data. Untuk kasus yang ditunjukkan pada Gambar 7, pengguna memasukkan pilihan 1 (transmisi data tanpa *error*).



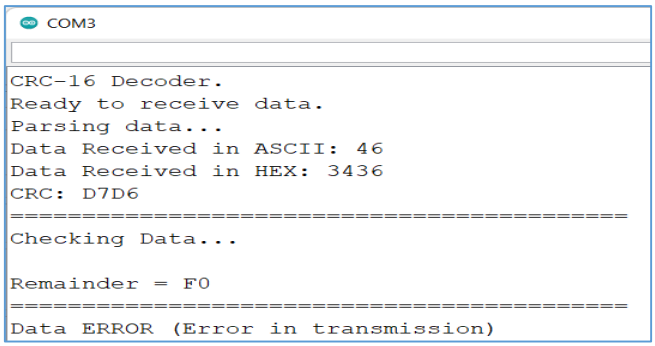
Gambar 8. Pengujian sistem (transmisi tanpa *error*) – Penerima

Gambar 8 menunjukkan tampilan pada sisi penerima dan dapat dilihat bahwa hasil pemeriksaan terhadap data yang diterima adalah tidak ada *error*. Pemeriksaan ini dilakukan dengan cara yang telah disebutkan sebelumnya yaitu melakukan pembagian (dalam aritmetika modulo-2) antara data yang diterima dengan polynomial A001. *Remainder* dari pembagian ini adalah nol sehingga dapat disimpulkan bahwa tidak terdapat *error* pada data diterima.



Gambar 9. Pengujian sistem (transmisi dengan *error*) – Pengirim

Gambar 9 dan Gambar 10 menunjukkan pengujian transmisi data dengan *error*. Untuk pengujian transmisi data dengan *error*, data yang dimasukkan adalah *string* "45" atau 3435 dalam heksadesimal. Nilai CRC dari input tersebut adalah D7D6 (1101011111010110), sehingga data yang seharusnya ditransmisikan adalah 3435D7D6 (atau 0011 0100 0011 0101 1101 0111 1101 0110). *Error* pada data disimulasikan dengan mengubah dua bit pada data 3435D7D6 menjadi 3436D7D6 (atau 0011 0100 0011 01**10** 1101 0111 1101 0110) seperti yang ditunjukkan pada Gambar 9. Pada Gambar 10 dapat dilihat bahwa sisa hasil pembagian (*remainder*) saat pemeriksaan data tidak menghasilkan nol, sehingga *receiver* menyimpulkan bahwa terdapat *error* pada data yang diterima.



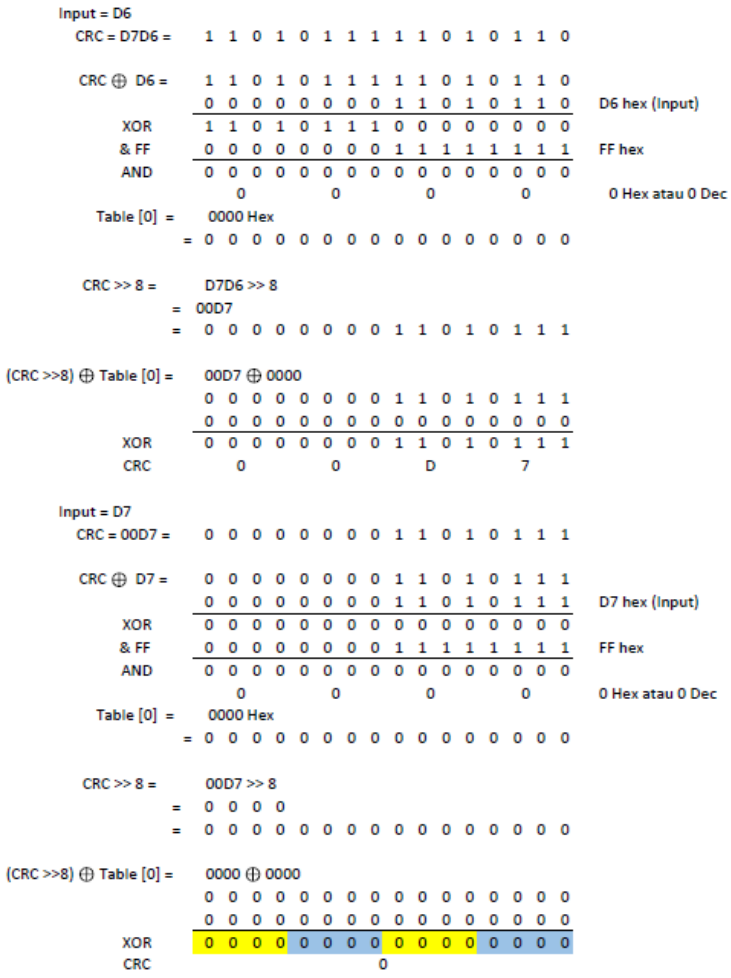
Gambar 10: Pengujian sistem (transmisi dengan *error*) – Penerima

Komputasi yang dilakukan oleh *decoder* pada sisi penerima untuk mendeteksi ada atau tidaknya *error* yang terjadi pada data yang ditransmisikan ditunjukkan pada Gambar 11. Input yang diterima oleh proses oleh *decoder* adalah 3435D6D7. Perlu dicermati bahwa nilai CRC yang digunakan pada proses ini adalah D6D7 yang merupakan nilai CRC dari input data "45" (D7D6) yang di-*reverse*.

Tabel 1 Waktu *Encoding* CRC-16 dengan Tabel *Lookup* dan SHIFT-XOR

Panjang Input Data	Waktu (milidetik)	
	Table Lookup	SHIFT-XOR
2 karakter	0.016	0.05
16 karakter	0.112	0.392
32 karakter	0.192	0.768
64 karakter	0.368	1.61
128 karakter	0.72	3.42

Gambar 11 menunjukkan proses perhitungan hanya untuk input D6 dan D7 (proses untuk input sebelumnya yaitu 34 dan 35 sama seperti yang telah ditunjukkan pada Gambar 4). Pada awal proses perhitungan, nilai variabel CRC adalah D7D6 yang merupakan hasil perhitungan dari dua input *byte* sebelumnya yaitu 34 dan 35.



Gambar 11: Perhitungan untuk transmisi data tanpa *error*

Selain melakukan pengujian sistem pendeteksi *error*, pengujian dilakukan juga dilakukan untuk mengukur kecepatan proses yang dilakukan oleh *encoder* untuk menghitung nilai CRC dari input data yang panjangnya bervariasi. Hasil pengujian waktu pengkodean (*encoding*) ditunjukkan pada Tabel 1. Berdasarkan hasil pada Tabel 1 dapat dilihat bahwa waktu yang dibutuhkan oleh algoritma yang menggunakan *tabel lookup* lebih cepat dibandingkan algortima yang menggunakan operasi SHIFT-XOR (konvensional). Selain itu, waktu perhitungan meningkat jika input karakter semakin panjang.

V. SIMPULAN DAN SARAN

Penelitian ini menyajikan desain dan implementasi *encoder* dan *decoder* untuk kode pendeteksi *error* CRC-16 pada Arduino. Algoritma yang digunakan untuk komputasi CRC-16 adalah metode yang memanfaatkan sebuah tabel *lookup*. Berdasarkan hasil pengujian yang dilakukan, *encoder* dapat menghasilkan nilai CRC-16 yang benar (*correct*) dari data dengan panjang yang bervariasi. Panjang data yang diuji mulai dari 2 karakter hingga 128 karakter. Verifikasi hasil perhitungan CRC-16 dilakukan dengan membandingkan nilai yang dihasilkan dengan perhitungan menggunakan kalkulator CRC. Pada pengujian sistem pendeteksi *error*, didapatkan bahwa sistem berjalan dengan baik dan dapat mendeteksi error (*bit flipping*) yang terjadi pada data yang

ditransmisikan dari pengirim ke penerima. Untuk pengujian waktu *encoding*, didapatkan hasil bahwa dibutuhkan waktu selama 0.72 milidetik untuk menghitung nilai CRC-16 dari input data sepanjang 128 karakter. Selanjutnya, untuk penggunaan memori diperoleh hasil masing-masing 24% dan 22% untuk program pada sisi pengirim dan sisi penerima, sehingga masih terdapat ruang memori yang cukup untuk mengimplementasikan fungsi-fungsi lain yang dibutuhkan untuk sebuah aplikasi yang lengkap pada *board* Arduino yang digunakan (misalnya, sistem pengontrolan lampu, *relay*, dan lain sebagainya).

Hasil ini menunjukkan bahwa kode pendeteksi *error* yang handal seperti CRC dapat diterapkan pada perangkat-perangkat yang memiliki *resource* atau sumber daya komputasi yang terbatas. Untuk pengembangan lebih lanjut, akan dieksplorasi implementasi kode-kode pendeteksi *error* yang lain maupun kode pengoreksi *error* seperti kode BCH (*Bose-Chaudhuri-Hocquenghem*).

DAFTAR PUSTAKA

[1] W. Stallings, *Data and Computer Communications*, 10th ed. Pearson, 2014.
[2] S. Lin and J. Li, *Fundamentals of Classical and Modern Error-Correcting Codes*, 1st ed. Cambridge University Press, 2021.
[3] B. Forouzan, *Data Communications and Networking*, 5th ed. McGraw-Hill, 2013.
[4] Z. Chen, D. Luo, Y. Luo, and J. Wu, "A Research on Internet of Things with CRC Method," *Proc. - 2021 Int. Conf. Wirel. Commun. Smart Grid, ICWCSG 2021*, pp. 138–142, 2021, doi: 10.1109/ICWCSG53609.2021.00035.
[5] R. Novaldi, S. Akbar, and P. Rakhmadhany, "Implementasi Error Detection System Pada Komunikasi Serial Arduino Menggunakan Metode Cyclic Redundancy Check (CRC)," *J. Pengemb. Teknol. Inf. dan Ilmu Komput.*, vol. 3, no. 2, pp. 1480–1485, 2018, [Online]. Available: https://j-ptiik.ub.ac.id/index.php/j-ptiik/article/view/4412
[6] A. R. Saleh and S. A. Sudiro, "CRC 8-bit Encoder-Decoder Component in FPGA using VHDL," *ELKOMIKA J. Tek. Energi Elektr. Tek. Telekomun. Tek. Elektron.*, vol. 8, no. 1, p. 58, Jan. 2020, doi: 10.26760/elkomika.v8i1.58.
[7] N. Sridevi, K. Jamal, and K. Mannem, "Implementation of Cyclic Redundancy Check in Data Recovery," *Proc. 2nd Int. Conf. Electron. Sustain. Commun. Syst. ICESC 2021*, pp. 17–24, Aug. 2021, doi: 10.1109/ICESC51422.2021.9532802.
[8] N. G. Augoestien and R. Aditya, "Implementasi Rangkaian CRC (Cyclic Redundancy Check) Generator pada FPGA (Field Programmable Gate Array)," *IJEIS (Indonesian J. Electron. Instrum. Syst.*, vol. 9, no. 1, pp. 65–74, Apr. 2019, doi: 10.22146/IJEIS.43906.
[9] D. Priyadi and T. Wellem, "Perancangan dan Implementasi Encoder dan Decoder CRC-8 untuk Pendeteksian Error pada Transmisi Data antar Perangkat IoT," *J. MEDIA Inform. BUDIDARMA*, vol. 6, no. 3, pp. 1677–1685, Jul. 2022, doi: 10.30865/MIB.V6I3.4366.
[10] J. Cabal, L. Kekely, and J. Korenek, "High-Speed Computation of CRC Codes for FPGAs," *Proc. - 2018 Int. Conf. Field-Programmable Technol. FPT 2018*, pp. 237–240, Dec. 2018, doi: 10.1109/FPT.2018.00042.
[11] N. N. Qaqos, "Optimized FPGA Implementation of the CRC Using Parallel Pipelining Architecture," *2019 Int. Conf. Adv. Sci. Eng. ICOASE 2019*, pp. 46–51, Apr. 2019, doi: 10.1109/ICOASE.2019.8723800.
[12] A. Bruen, M. Forcinito, and J. McQuillan, "Cyclic Linear Codes, Shift Registers,

and CRC," *Cryptogr. Inf. Theory, Error-Correction*, pp. 453–472, Jul. 2021, doi: 10.1002/9781119582397.CH21.
[13] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2016.
[14] "Arduino UNO R3." https://docs.arduino.cc/hardware/uno-rev3 (accessed Aug. 25, 2022).
[15] "Online CRC-8 CRC-16 CRC-32 Calculator." https://crccalc.com/ (accessed Aug. 30, 2022).