Design and Implementation of Cyclic Redundancy Check for Downlink Transmission in NB-IoT

Manjunath Inamati[1], Goutami Naragund[2], Chetan Paranatti[3], Saroja Siddamal[4]
Suhas Shirol[5], Vijay H M[6], and Suneeta Budihal[7]

[1−7]School of Electronics and Communication Engineering
KLE Technological University
Hubli, India
[1]manjunathinamati8@gmail.com,
[2]goutami8296@gmail.com,
[3]paranatti.chetan@gmail.com,
[4]sarojavs@kletech.ac.in
[5]suhasshirol@kletech.ac.in,
[6]vijay.hm@kletech.ac.in,
[7]suneeta_vb@kletech.ac.in

Abstract. Message corruption is a common problem in communication systems. The received signal should always be verified to know whether the signal is corrupted, or the same signal is transmitted. In communication systems, the Cyclic Redundancy Check (CRC) is an essential block for ensuring data integrity and error detection. The authors propose a design and implementation of 16-bit CRC for downlink transmitter in NB-IoT. The series architecture involves sequential processing of data bits, while the parallel architecture utilizes parallel processing to enhance throughput and reduce latency. The design is simulated using Vivado HLS and synthesized RTL is tested on the Spartan 6 and Arty 7 target FPGA board. It is observed that parallel CRC consumes 48.33% more area and 72.42% higher power than series CRC, with the compromise in area and power, it is also observed that parallel CRC is 90% faster than series CRC. The result shows that CRC bits in parallel CRC are computed in 5 clock cycles whereas those in series CRC are computed in 50 clock cycles.

Keywords: NB-IoT, Series CRC, Parallel CRC, Downlink transmitter.

1 INTRODUCTION

The transmission chain in NB-IoT comprises several blocks that collectively enable successful data transmission. It starts with channels such as Narrowband Physical Broadcast Channel (NPBCH) for broadcasting system information, the Narrowband Physical Downlink Control Channel (NPDCC) for transmitting control information, and the Narrowband Physical Downlink Shared Channel (NPDSCH) for user data. The chain includes blocks such as CRC attachment, which adds a 16-bit CRC to the 34-bit input data, resulting in a 50-bit output. The channel coding block adds redundancy of 100 bits to the 50-bit input from the CRC attachment, resulting in a 150-bit output.

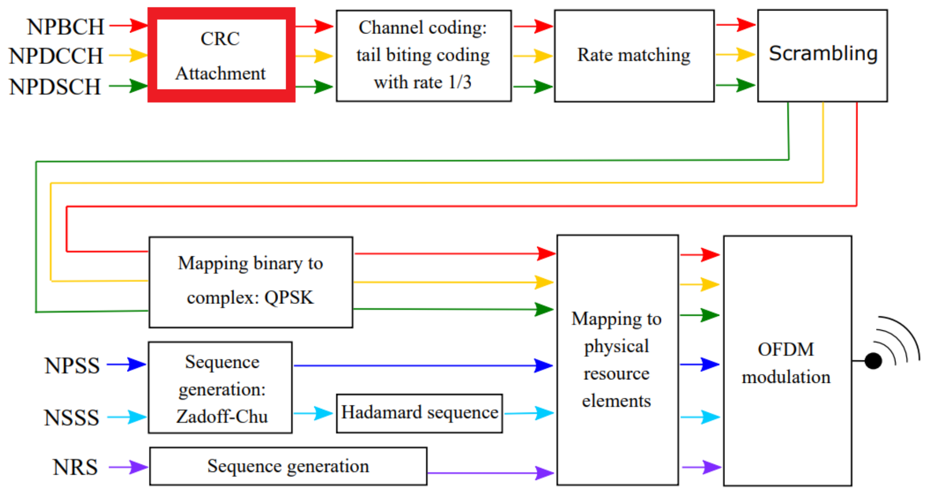

NPBCH
NPDCCH
NPDSCH
CRC Attachment
Channel coding: tail biting coding with rate 1/3
Rate matching
Scrambling
Mapping binary to complex: QPSK
NPSS
NSSS
NRS
Sequence generation: Zadoff-Chu
Hadamard sequence
Sequence generation
Mapping to physical resource elements
OFDM modulation

Fig. 1. Overall transmission chain of the NB-IoT eNB.

Additionally, the chain incorporates a rate-matching block for adjusting transmission rates, and a scrambling block for ensuring data security and integrity. Furthermore, the chain involves converting binary signals into Quadrature Phase Shift Keying (QPSK) signals, generating synchronization sequences (NPSS, NSSS, NRS), allocating physical resources for transmission, and utilizing Orthogonal Frequency Division Multiplexing (OFDM) for spectral efficiency.

A Cyclic Redundancy Check is a block used in the communication system to add redundancy bits to the given data to detect the errors that occurred during the transmission. The redundant bits can be of any length based on the required specifications and are calculated based on simple binary division using a fixed divisor called the generator polynomial.

The rest of the paper is organized as follows, a detailed description and observation of the state-of-the-art paper is discussed in Section 2. In Section 3 design and implementation of series and parallel CRC is presented. The results and comparative analysis are discussed in Section 4 followed by a conclusion in Section 5.

2 LITERATURE SURVEY

This section discusses the state of art with various approaches, along with the advantages and limitations.

Implementation of CRC requires specifications such as the number of data bits, the number of CRC bits and the number of output bits. All the specifications and position of the CRC block in the NB-IoT transmission chain are referred from the work [1]. This work gives information on the physical layer of the NB-IoT system including the general overview and its features. The transmission chain of the NB-IoT eNB consists of various blocks such as CRC, channel encoding, scrambler, rate matching, etc. After getting the specifications another important factor is the selection of a generator polynomial. The authors of [2] propose a set of standards for choosing “good” CRC polynomials that take into account various factors. The paper aims to provide quantitative information and engineering guidelines for CRC selection for embedded network designers. The CRC can be implemented using both series and parallel architecture. However, the performance comparison of both architectures gives a suitable application for each architecture. Paper [3] compares two types of CRC designs for CRC-8 Header Error Control in Asynchronous Transfer Mode (ATM HEC) using multi-output LFSR (MLFSR). This work talks about the advantages of parallel versus serial LFSR implementation in terms of efficiency and power consumption. Since the performance comparison of series and parallel architecture has to be performed, the implementation of both architectures has to be carried out separately and analysed. The basic series architecture can be implemented using a Linear Feedback Shift Register (LFSR), consisting of a few flip-flops (FFs) and logic gates. The parallel CRC implementation can be done using different techniques, such as:

- F-Matrix method

  The authors of paper [4] presents a 32-bit parallel architecture for 64-bit data based on the F matrix. In comparison to conventional techniques, the design seeks to minimize computing time by 50%, increasing throughput in high-speed data networks. The suggested architecture provides a small and simple way for quick CRC generation.

- Table-based algorithm

  Paper [5] uses a ”Table-Based Algorithm for Pipelined CRC Calculation” for the implementation of CRC-16 and CRC-32. It calculates the 16 bit CRC for each eight-bit chunk and 32-bit CRC polynomial has also been calculated, one 16-bit chunk at a time. The tables are constructed from the architecture presented based on their generator polynomial.

- Fast CRC update

  The use of fast hardware for parallel CRC checksums is covered in the paper [6]. The theoretical result presents a recursive formula for parallel CRC realization. The number of bits that can be processed in parallel by the solution varies depending on the degree of the polynomial generator. Based on the provided polynomial, high-level parametric codes are created for autonomous circuit synthesis.

The look ahead method and Galois field theory are used in the paper [7] for quicker CRC computations, it methodically breaks down input messages into smaller sequences. The research reviewed in the study [8] endeavour is a generalized parallel CRC formulation that can be incorporated into any polynomial and data width for fast operation and to minimize the delay.

3 DETAILED CRC DESIGN

CRC implementation is based on specifications such as a number of data bits and the number of CRC bits. If the resultant CRC should be n bits, then the polynomial should be of (n+1) bits.

Output Y = K + R

Where Y–Length of output data.

K–Length of input data.

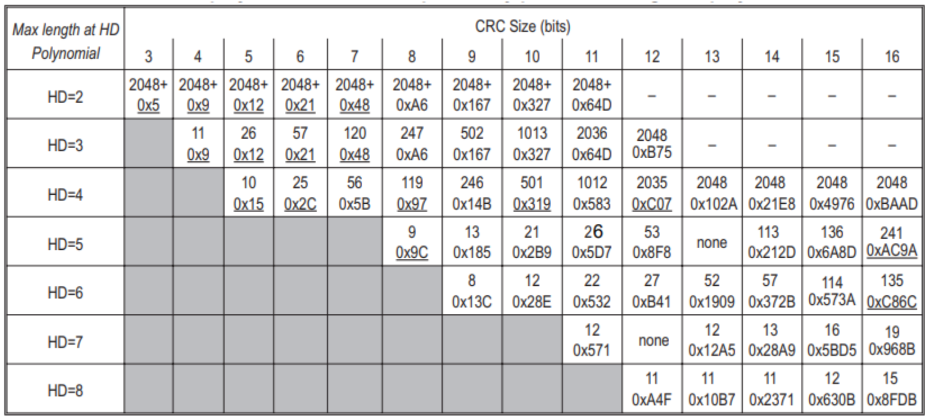
R–Length of redundant bits.

In NB-IoT, a 34-bit data is sent as an input to the CRC block and the output of the block is 50-bits. Considering the input and output bits the CRC bits are observed to be 16-bits.

3.1 Generator polynomial

In CRC design the key factor is the choice of generator polynomial. The generator polynomial is selected based on the hamming distance. The Hamming distance of a generator polynomial is the minimum number of bit changes required to transform it into another valid generator polynomial. The shorter the hamming distance better the performance of the CRC. In NB-IoT a 16-bit CRC should be appended to 34-bit input data. So, the generator polynomial 0xBAAD is selected which has the least hamming distance (HD = 4) among all the 16-bit polynomials mentioned in Fig. 2.

Polynomial - $x^{16} + x^{15} + x^{13} + x^{12} + x^{11} + x^9 + x^7 + x^5 + x^3 + x^2 + 1$



Max length at HD Polynomial	CRC Size (bits)													
	3	4	5	6	7	8	9	10	11	12	13	14	15	16
HD=2	2048+ 0x5	2048+ 0x9	2048+ 0x12	2048+ 0x21	2048+ 0x48	2048+ 0xA6	2048+ 0x167	2048+ 0x327	2048+ 0x64D	–	–	–	–	–
HD=3		11 0x9	26 0x12	57 0x21	120 0x48	247 0xA6	502 0x167	1013 0x327	2036 0x64D	2048 0xB75	–	–	–	–
HD=4			10 0x15	25 0x2C	56 0x5B	119 0x97	246 0x14B	501 0x319	1012 0x583	2035 0xC07	2048 0x102A	2048 0x21E8	2048 0x4976	2048 0xBAAD
HD=5						9 0x9C	13 0x185	21 0x2B9	26 0x5D7	53 0x8F8	none	113 0x212D	136 0x6A8D	241 0xAC9A
HD=6							8 0x13C	12 0x28E	22 0x532	27 0xB41	52 0x1909	57 0x372B	114 0x573A	135 0xC86C
HD=7									12 0x571	none	12 0x12A5	13 0x28A9	16 0x5BD5	19 0x968B
HD=8										11 0xA4F	11 0x10B7	11 0x2371	12 0x630B	15 0x8FDB

Fig. 2. Best polynomials for HD at given CRC size and data word length [2].

3.2 Design of series CRC

The series CRC design requires flip-flops for the data storing and XOR gates for binary subtraction. The architecture is based on the binary division carried out by shifting the data bits with respect to polynomial bits. The data bits are fed to the Linear Feedback Shift Registers (LFSR) from D33 (MSB) to D0 (LSB) along with a 16-bit zero-initialized CRC. The XOR gates are placed according to the coefficients of the polynomial selected for the calculation. The final data in the registers give the 16-bit CRC which must be appended to the input data to form an output of 50-bits. The series architecture shown in Fig. 3, features the output of the shift register (SR15) being tapped as one of the inputs for all the XOR gates.

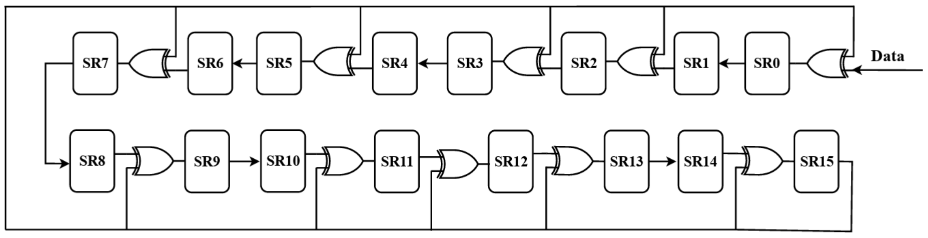


Fig. 3. Architecture of series CRC.

3.3 Design of parallel CRC

Series CRC calculation is slower as it processes each bit sequentially, leading to a higher number of clock cycles for computation.

Looking at this disadvantage in series CRC, the authors propose a novel architecture for CRC calculation. The literature says there are various techniques available in computing parallel CRC such as:

1. A Table-Based Algorithm for Pipelined CRC Calculation.
2. Fast CRC Update.
3. F-matrix-based parallel CRC generation.
4. Unfolding, Retiming and Pipelining Algorithm.

In this paper, the authors have proposed a “Table-Based Algorithm for pipelined CRC calculation” method. In table based approach a table is constructed for each shift depending on the CRC bits to be generated.

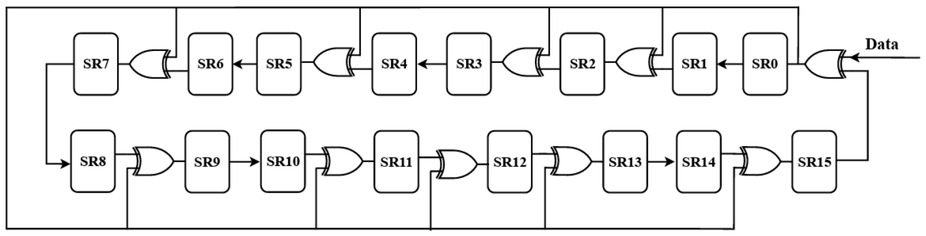


Fig. 4. Derived series architecture for parallelism.

As the processing of the data bits is sequential in series, the parallelism in the architecture is brought by calculating the CRC by XORing the contents of the table after 8 shifts. So that 8-bit data can be executed in one clock cycle instead of sequential processing. The tables are constructed based on derived architecture including the below-mentioned properties and notes.

NOTE

1. (SRn) is the nth bit of the CRC shift register.
2. Cn is the content of the n-th bit of the initial CRC before any shifts have taken place.
3. SR0 is the LSB.
4. The entries under each CRC shift register bit indicate the value to be XORed together to generate the content of that bit in the CRC shift register.
5. Dn is the data input, with MSB input first.
6. D7 is the MSB of the input byte, and D0 is the LSB.

PROPERTIES

1. Commutative Property (X XOR Y = Y XOR X).
2. Associative Property (X XOR Y XOR Z = X XOR Z XOR Y).
3. Involution Property (X XOR X = 0).

Based on the generator polynomial, the tables are constructed for 8-bit data and 16-bit CRC. The number of data bits to be executed in one cycle corresponds to the number of tables to be constructed. As the complexity of the table increases after every shift, the tables are constructed only for 8 shifts, which completes the evaluation of CRC in 5 cycles. Table 1 contains the initial contents of LFSR, Table 2 contains the LFSR contents after one shift, and Table 3 contains the LFSR contents after 8 shifts. The final LFSR contents of Table 3 must be XORed to get the corresponding CRC bits for the given 8-bit data.

Table 1. Initial LFSR contents.

SR15	**SR14**	**SR13**	**SR12**	**SR11**	**SR10**	**SR9**	**SR8**	**SR7**	**SR6**	**SR5**	**SR4**	**SR3**	**SR2**	**SR1**	**SR0**
C15	C14	C13	C12	C11	C10	C9	C8	C7	C6	C5	C4	C3	C2	C1	C0

Table 2. LFSR contents after one shift.

SR15	**SR14**	**SR13**	**SR12**	**SR11**	**SR10**	**SR9**	**SR8**	**SR7**	**SR6**	**SR5**	**SR4**	**SR3**	**SR2**	**SR1**	**SR0**
D7	C13	D7	D7	D7	C9	D7	C7	D7	C5	D7	C3	D7	D7	C0	D7
C15		C15	C15	C15		C15		C15		C15		C15	C15		C15
C14		C12	C11	C10		C8		C6		C4		C2	C1		

The calculation of CRC for the whole 34-bit has been carried out by dividing the data into 8-bit chunks. The 34-bit data is rounded off to 40-bit data by appending zeros

Table 3. LFSR contents after eight shifts.

SR15	SR14	SR13	SR12	SR11	SR10	SR9	SR8	SR7	SR6	SR5	SR4	SR3	SR2	SR1	SR0
D0	D1	D0	D0	D0	D1	D0	D1	D0	D1	D0	D1	D0	D0	D1	D0
D1	D3	D2	D4	D1	D2	D1	D2	D1	D2	D1	D5	D4	D1	D2	D1
D4	C9	C8	D5	D3	D4	D3	D4	D3	D6	D5	C9	C8	D3	D3	D2
D5	C11	C10	D6	D6	D7	D6	D6	D5	C9	D7	C13	C12	D4	D6	D5
D6	C6	C5	C8	D7	C9	C8	D7	D6	C10	C8			D5	D7	D6
D7			C12	C8	C10	C9	C9	C8	C14	C9			D6	C9	D7
C8			C13	C9	C12	C11	C12	C9		C13			C8	C10	C8
C9			C14	C11	C15	C14	C14	C11		C15			C9	C11	C9
C12			C4	C14	C2	C1	C15	C13					C11	C14	C10
C13				C15			C10	C14					C12	C15	C13
C14				C3			C0						C13		C14
C15													C14		C15
C7															

at MSB to divide the data equally into 8-bit chunks. The visual representation of the processing of 8-bit data input chunks in each clock cycle is shown in Fig. 5.

Initially, the contents of the CRC for the first chunk of the data are made zero. Later the initial CRC contents of (n+1)th chunk are initialised with the evaluated CRC of n-th chunk.

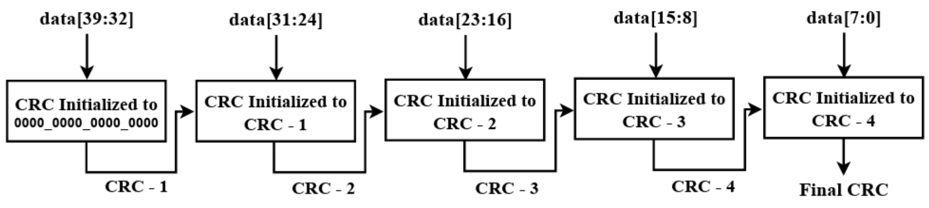

data[39:32]
data[31:24]
data[23:16]
data[15:8]
data[7:0]
CRC Initialized to 0000_0000_0000_0000
CRC Initialized to CRC - 1
CRC Initialized to CRC - 2
CRC Initialized to CRC - 3
CRC Initialized to CRC - 4
CRC - 1
CRC - 2
CRC - 3
CRC - 4
Final CRC

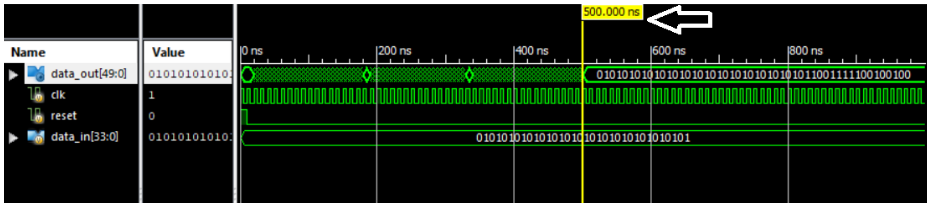
Fig. 5. Visual representation of parallel processing.

4 RESULTS

The designed series and parallel CRC were simulated and synthesized using Vivado HLS and then the synthesized RTL netlists were ported to the Arty 7 and Spartan 6 FPGA board for testing.

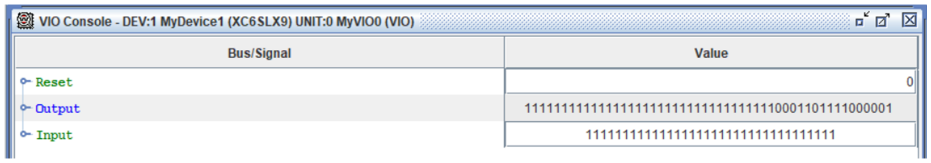
4.1 Simulation of series CRC

Fig. 6 shows the simulation result of series CRC with an input data size of 34 bits, the 16-bit CRC is observed after 50 clock cycles. Fig. 7 shows the post-synthesized result and generation of 16-bit CRC after porting to FPGA.



```
Name            Value
data_out[49:0]  01010101010   0101010101010101010101010101010101011001111100100100
clk             1
reset           0
data_in[33:0]   01010101010   0101010101010101010101010101010101
```

Fig. 6. Simulation result of series CRC.

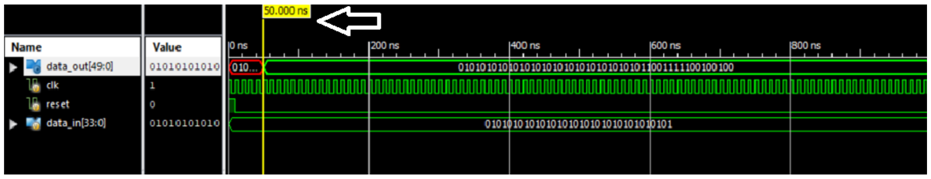

VIO Console - DEV:1 MyDevice1 (XC6SLX9) UNIT:0 MyVIO0 (VIO)

Bus/Signal	Value
Reset	0
Output	11111111111111111111111111111111111110001101111000001
Input	1111111111111111111111111111111111

Fig. 7. Post Synthesis result of series CRC.

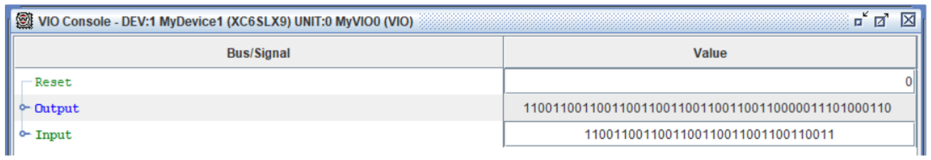
4.2 Simulation of parallel CRC

Fig. 8 shows the simulation result of parallel CRC with an input data size of 34 bits, the 16-bit CRC is observed after 5 clock cycles. Fig. 9 shows the post-synthesized result and generation of 16-bit CRC after porting to FPGA.



```
Name            Value
data_out[49:0]  01010101010   010...   0101010101010101010101010101010101100111100100100
clk             1
reset           0
data_in[33:0]   01010101010   0101010101010101010101010101010101
```

Fig. 8. Simulation result of parallel CRC.


VIO Console - DEV:1 MyDevice1 (XC6SLX9) UNIT:0 MyVIO0 (VIO)

Bus/Signal	Value
Reset	0
Output	11001100110011001100110011001100110000011101000110
Input	1100110011001100110011001100110011

Fig. 9. Post synthesis result of parallel CRC.

4.3 Result analysis of hardware implementation

To obtain area and power, the RTL was synthesized in 180nm CMOS technology node using Cadence tool. Table 4 shows the area, power and clock cycles of series and parallel CRC. When compared it is found that parallel CRC occupies 48.33% more area and consumes 72.42% more power than series CRC, but it is 90% faster than series CRC in terms of clock cycles.

Table 4. Comparison of series and parallel CRC.

Parameters	**Series**	**Parallel**	**%**
Area (u m^2)	3195.856	6185.841	48.33
Power (m Watt)	1.032	3.742	72.42
No. of clock cycles	50	5	90

5 CONCLUSION

The CRC block has a major role in every communication system. The architectures are analysed for area, power and speed. It is observed that parallel CRC consumes 48.33% more area and 72.42% higher power than series CRC, with the compromise in area and power, it is also observed that parallel CRC is 90% faster than series CRC. The series architecture involves sequential processing of data bits consuming less area and power but computes the 16-bit CRC in 50 clock cycles. The parallel architecture utilizes parallel processing to enhance throughput and reduce latency by computing 16-bit CRC in 5 clock cycles making it 90% faster than series architecture. Parallel architecture can be made more efficient by constructing a table after 34 shifts which gives the computed CRC after 1 clock cycle. The efficient architecture can be selected among series and parallel keeping the area and power constraints into consideration.

6 ACKNOWLEDGMENT

The authors would like to acknowledge the support provided by the Chips to Start (C2S) Program, an initiative of the Ministry of Electronics and Information Technology (MietY).

References

1. Kanj, Matthieu, Vincent Savaux, and Mathieu Le Guen. ”A tutorial on NB-IoT physical layer design.” IEEE Communications Surveys & Tutorials 22.4 (2020): 2408-2446.
2. Koopman, Philip, and Tridib Chakravarty. ”Cyclic redundancy code (CRC) polynomial selection for embedded networks.” International Conference on Dependable Systems and Networks, 2004. IEEE, 2004.
3. Panda, Avipsa S., and G. L. Kumar. ”Comparison of serial data-input CRC and parallel data-input CRC design for CRC-8 ATM HEC employing MLFSR.” 2014 International Conference on Electronics and Communication Systems (ICECS). IEEE, 2014.

4. Mathukiya, Hitesh H., and Naresh M. Patel. ”A Novel Approach for Parallel CRC generation for high speed application.” 2012 International conference on communication systems and network technologies. IEEE, 2012.
5. Parallel Cyclic Redundancy Check for HOTLink, www.cypress.com
6. Campobello, Giuseppe, Giuseppe Patane, and Marco Russo. ”Parallel CRC realization.” IEEE Transactions on Computers 52.10 (2003): 1312-1319.
7. MING-DER SHIEH, MING-HWA SHEU, Chung-Ho Chen, and Hsin-Fu Lo. ”A systematic approach for parallel CRC computations.” Journal of information science and engineering 17 (2001): 445-461.
8. Arifin, Md Mashrur, et al. ”Design and implementation of high performance parallel crc architecture for advanced data communication.” 2019 4th International Conference on Electrical Information and Communication Technology (EICT). IEEE, 2019.