

CSCE 491

Design Specification for MAC Sub-Layer Receiver

Part 2: Cyclic Redundancy Check

1. Introduction

In networking systems a significant role of the MAC layer is to convert the potentially unreliable physical link between two machines into an apparently very reliable link. This is achieved by including redundant information in each transmitted frame. Depending on the nature of the link and the data, one can include just enough redundancy to make it possible to detect errors and then arrange for the retransmission of damaged frames. The cyclic redundancy check or CRC is a widely used parity bit based error detection scheme in serial data transmission applications.

All basic error detection techniques operate around the idea of trying to enable a receiver of a message transmitted through a noisy (error-introducing) channel to determine if the message was sent free of errors. In order to determine if an error is present, the transmitter appends a calculated value, called the checksum, to the end of the message. The receiver then has the ability to calculate the checksum in the same way as the transmitter, based on the data that was sent. If the checksum calculated by the receiver is the same, as that was, appended to the end of the message, then the data was sent free of error

1.1. Statement of Problem

The basic objective of the problem is to implement a CRC (Cyclic Redundancy Check) Error Detection algorithm. At the transmitter, a CRC will be calculated and appended to the end of the data stream. At the receiver, we will verify the CRC (a checksum) appended to the end of the data stream in order to check for errors. The bit stream received will use the CRC at the end of the bit stream to check if error occurred during transmission.

1.2. Design Objectives

Our design objective is to design a CRC for control frame subtypes used in the data transmission for 802.11. The control frame subtypes are

- Request to send (RTS): 20 Bytes
- Clear to send (CTS): 14 Bytes
- Data1: 2k MSDU
- Data 2: 128bytes MSDU
- Acknowledgement (ACK): 14Bytes

There are two types of implementation for the CRC algorithm.

- Serial Implementation
- Parallel Implementation

Serial Implementation uses bit-by-bit operation. This method is not optimal, as IEEE 802.11 transceivers communicate the data stream to the MAC layer over a 4-bit bus.

Parallel Implementation uses multiple bits of stream per clock cycle. This speeds up the operation. Therefore a parallel implementation is preferred.

1.3 Assumptions and Constraints

The assumptions for this specification are

- The specifications used for CRC 802.3 are same as CRC 802.11.
- Four bytes are transmitted in a parallel data transmission.
- Zero Propagation Delay for the XOR gates.

2. Functional Description

2.1.Serial Implementation

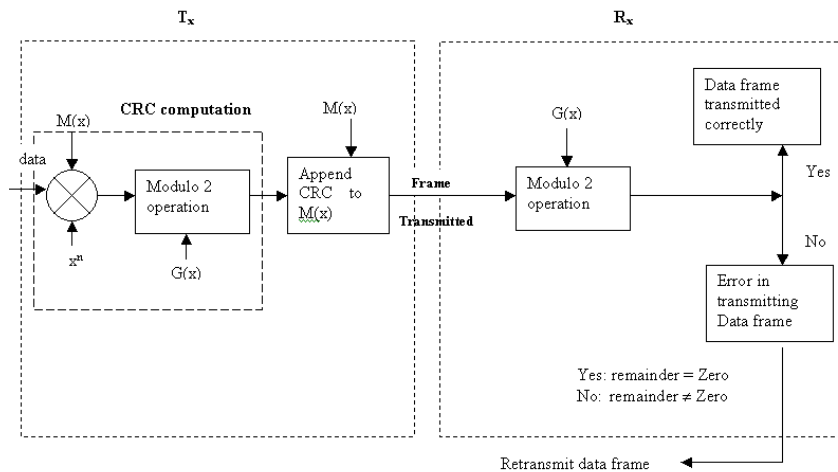


Fig. 1. BLOCK DIAGRAM OF CRC

The block diagram shown above explains the basic function of the CRC algorithm. The Block diagram has two parts. The first part deals with the description on the transmitter side and the second part deals with the description on the receiver side. The basic function of the CRC is to ensure error free transmission and reception of data. The operations generally carried out in CRC are modulo-2 operations i.e., all the operations (addition, multiplication, division) XOR operator.

2.1.1. CRC/ Transmitter

The CRC is generated using a generator polynomial. The standard CRC-32 polynomial used is as follows:

$$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^8 + X^5 + X^4 + X^2 + X + 1$$

In general, n-bit CRC is calculated by representing the data stream as a polynomial, $M(x)$. Then we multiply $M(x)$ by x^n (where n is the degree of the polynomial $G(x)$), and divide the result by a generator polynomial $G(x)$ (all these procedure is carried out by using Modulo-2 operation). The resulting remainder of the above operation is CRC. The CRC thus generated is appended to the data stream. The whole of the above operation can be mathematically represented as

$$CRC = \text{Remainder of } (M(x) * x^n / G(x))$$

The RTL description of the above logic is explained in Section 3.

2.1.2. CRC/ Receiver

The complete transmitted polynomial $T(x)$ is then divided by the same generator polynomial $G(x)$ at the receiver end. If the resulting solution has no remainder then there are no errors in transmission else, the receiver sends no acknowledgment and the data frame has to be retransmitted. This operation is carried out by DCF (Distributed Coordinate Function) and PCF (Point Coordinate Function) algorithms in the IEEE 802.11 standard.

2.2.Parallel Implementation

To speed up the whole process of CRC generation, parallel implementation is preferred. This implementation operates on multiple bits of data stream per clock cycle. (In our implementation we use 4 bits at a time). The whole implementation could be made simpler by storing the results of the most of the calculations in a table or a storage array. This is referred to as table-driven implementation.

A table can be created to store all possible values of the Poly XORed with the Shifted Bits of the message. For example, in our case since we are processing 4-bits per clock cycle, there are 16 possible values of the Poly XORed with the 4-bits shifted out of the register. Thus we created a table that can store these 16 calculated values in a byte addressable storage array. These precompiled results are retrieved during CRC computation.

We can basically state the above procedure in the form of an algorithm given below:

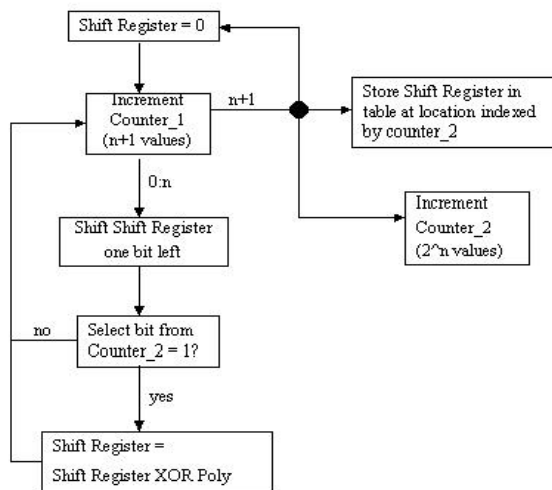
While (more message bits exist)

- TOP = top (Register);
- Register = (Register << 28) | next_augmessage;
- Register = Register XOR precomputed_table [TOP]

The basic operations we use in the implementation of the algorithm include an OR, a shift, an XOR and a table lookup. Therefore, the implementation consists of two modes: Table Generation and CRC Processing.

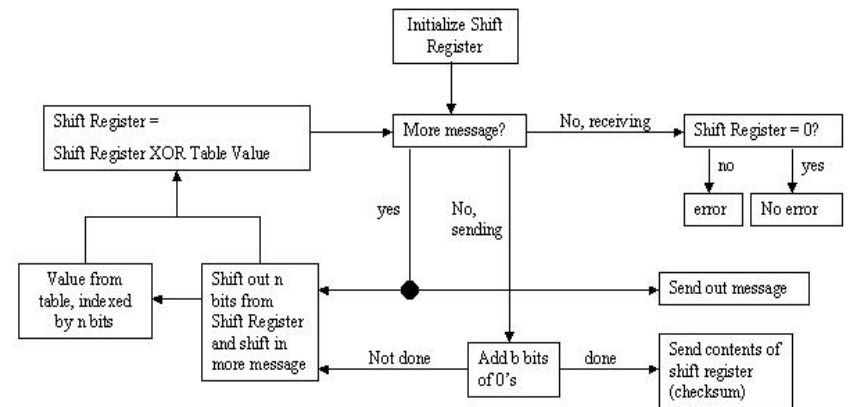
2.2.1. Table Generation

During table generation, we calculate all possible values of the 8-bit polynomial 'divided' by any 4-bits that will be shifted out of the main register during the calculation of the checksum. These sixteen 8-bit values are then stored in a byte-addressable table to be used throughout the second mode of the implementation.



Dividing each n-bit value by the polynomial creates the table. The remainder is stored in the table. The division part of the process is taken care by the boxes on the left. The Counter_2 is used to increment through each n-bit number. The shift register contains the remainder throughout the process of division. A multiplexer is used to select between Counter_1 and Counter_2.

2.2.2 CRC/CRC Processing



The CRC processing mode consists of

- Appending the checksum to the message.
- Receiving the message and checking whether the message was received properly or not.

In the above diagram, instead of performing the division, the values from the table are accessed.

2.3. Example of CRC Generation

Transmitted Frame: 1101011011
 Generator: 10011
 Message after appending 4 zero bits: 11010110000

```

          1 1 0 0 0 0 1 0 1 0
10011 | 1 1 0 1 0 1 1 0 1 1 0 0 0 0
      1 0 0 1 1
      1 0 0 1 1
      1 0 0 1 1
      0 0 0 0 1
      0 0 0 0 0
      0 0 0 1 0
      0 0 0 0 0
      0 0 1 0 1
      0 0 0 0 0
      0 1 0 1 1
      0 0 0 0 0
      1 0 1 1 0
      1 0 0 1 1
      0 1 0 1 0
      0 0 0 0 0
      1 0 1 0 0
      1 0 0 1 1
      0 1 1 1 0
      0 0 0 0 0
      1 1 1 0
    
```

CALCULATION OF POLYNOMIAL CODE CHECKSUM (TRANSMITTED END)

Remainder

Transmitted frame: 11010110111110

CRC Detection

Received Frame: 1101011001
 Generator: 10011
 Message after appending 4 zero bits: 11010010000

```

          1 1 0 0 0 0 1 0 0 0
10011 | 1 1 0 1 0 1 1 0 0 1 0 0 0 0
      1 0 0 1 1
      1 0 0 1 1
      1 0 0 1 1
      0 0 0 0 1
      0 0 0 0 0
      0 0 0 1 0
      0 0 0 0 0
      0 0 1 0 0
      0 0 0 0 0
      1
      0 1 0 0 1
      0 0 0 0 0
      1 0 0 1 0
      1 0 0 1 1
      0 0 0 1 0
      0 0 0 0 0
      0 0 1 0 0
      0 0 0 0 0
      0 1 0 0 0
      0 0 0 0 0
      1 0 0 0
    
```

CALCULATION OF POLYNOMIAL CODE CHECKSUM (RECIEVER END)

ERRONEOUS DATA RECEIVED!!!

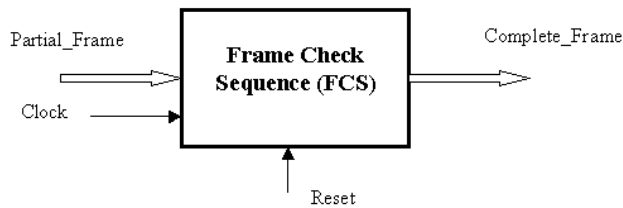
Remainder

Received frame: 11010110111000

3. Architectural Description

3.1. Transmitter

The block diagram showing the Input and Output signals to the CRC Encoder at the transmitter end is given below.



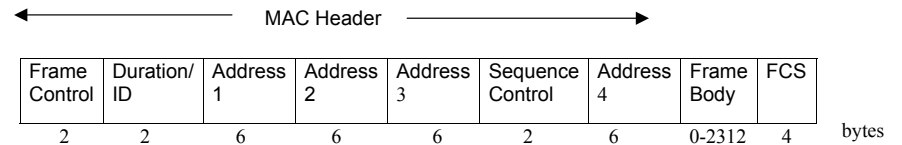
The Description of the signals is contained in the table below.

Name Of The Signal	Type Of The Signal	Description
Partial_Frame	Input for the frame check sequence	It Contains the information regarding the header and body. The length depends on the type of the frame body
Complete_Frame	Output of the frame check sequence	It contains the information of CRC appended to the frame (FCS)
Clock	Input	1 bit signal
Reset	Input	1 bit signal

The Partial_Frame consists of these control frame subtypes

- Request to send (RTS): 20 Bytes
- Clear to send (CTS): 14 Bytes
- Data1: 2k MSDU
- Data 2: 128bytes MSDU
- Acknowledgement (ACK): 14Bytes

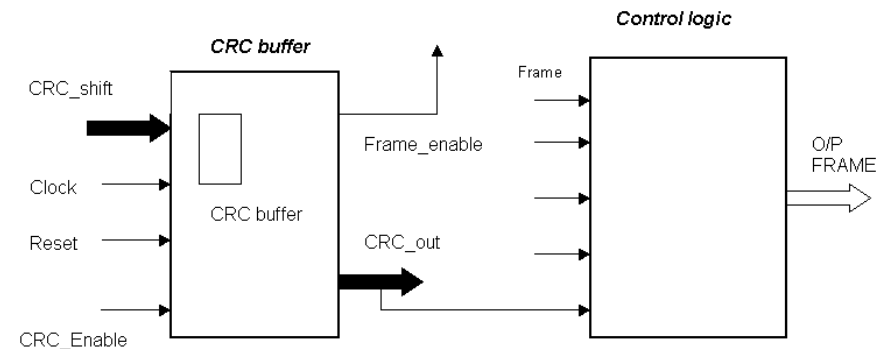
The General IEEE 802.11 Frame Format is shown below



The frame begins with a MAC header. The start of the header is the frame control field then a field that contains the duration information for the network allocation vector followed by the three addressing fields. The next field contains frame sequence information. The final field of the MAC header is the fourth address field. Following the MAC header is the frame body. The final field in the MAC frame is the frame check sequence

3.2. Receiver

The block diagram showing the Input and Output signals to the CRC Decoder at the receiver end is given below



The Description of the signals is below

Name of the signal	Type of the signal	Size	Description
CRC_shift	I/P	16 – bits	Data received from the 16 bit shift register
CRC_Enable	I/P	1 -bit	Used to enable CRC decoder
Clock	I/P	1-bit	Used for synchronization
Reset	I/P	1-bit	Reset all the signals
CRC_out	O/P	32 - bits	The data is combined into 32 bits after 2 clock cycles
Frame_enable	O/P	1-bit	This signal is sent to frame encoder to confirm that the data received is error free.

The frame received consists of Frame Control, Duration/ID, Addresses, Sequence Control, Frame Body and Frame Check Sequence (CRC). The headers are separated at the receiver end by using a serial to parallel shift register whose output is of 16-bit length. This is done in order to separate the data from the remaining headers and also to check whether the data has been received correctly or not. This is done by the CRC decoder.

The CRC Decoder gets the 16-bit input from the shift register (indicated by *CRC_shift*). The CRC Decoder is selected by a signal *CRC_Enable*. In order to use a CRC of 32 bits, we use a buffer in the CRC Decoder, which is used to store the first 16 bits given as the input to the decoder during one clock cycle. During the next clock cycle, the CRC Decoder gets the second set of 16 bits, which is then combined, with the 16 bits stored in the buffer to obtain the 32-bit CRC. This 32-bit CRC is then compared with the CRC obtained in the transmitter side to decide whether the data has been transmitted

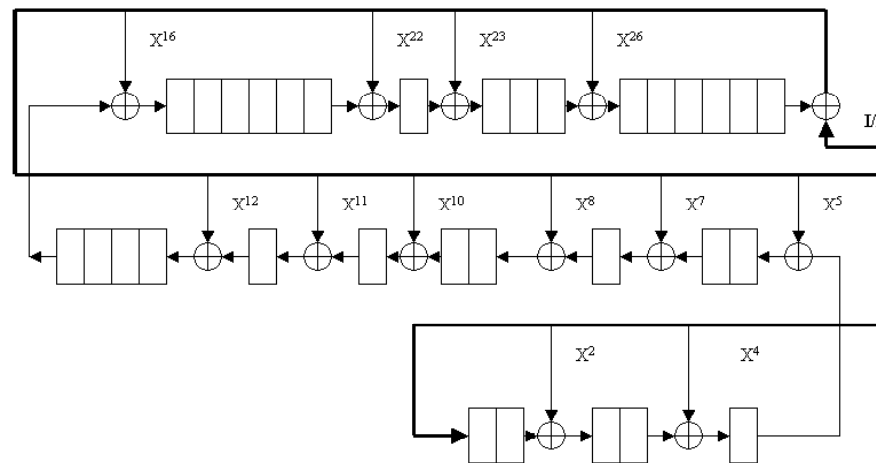
correctly or not. The *Clock* signal is used for synchronization. The *Reset* signal is used to reset the functionality.

The outputs of the CRC Decoder are *CRC_out* and *Frame_enable*. *CRC_out* represents the 32-bit CRC. This 32-bit CRC is used for sending the frames from the receiver to the transmitter such as *CTS* (Clear to Send) and *ACK* (Acknowledgement). *Frame_enable* is used to indicate whether the data has been transmitted correctly or not. If the CRC computed matches with that of the CRC of the transmitter side, then this signal is asserted high, giving an indication to Frame Decoder that the frame input is error free. If the CRC computed does not match with that of the CRC of the transmitter side, then this signal is asserted low, giving an indication to Frame Decoder that the frame input consists of errors.

4. Behavioral and RTL Description

4.1 Serial Implementation

CRC GENERATION

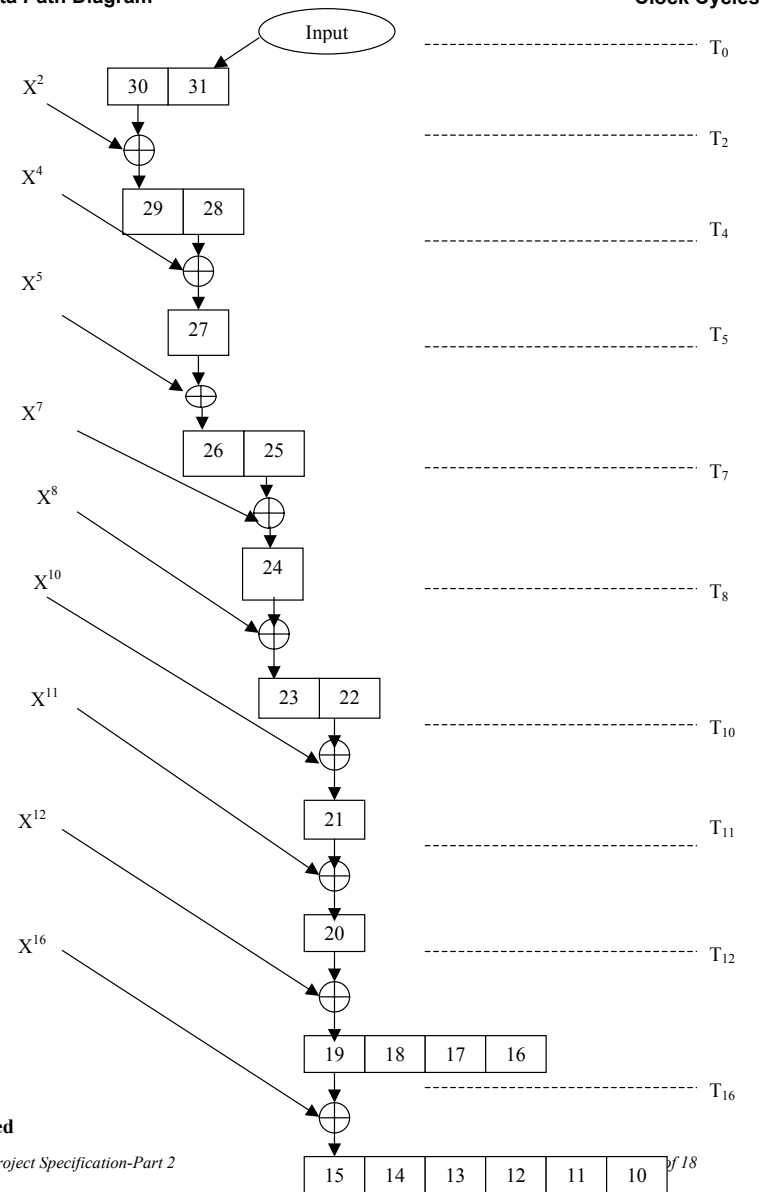


$$G(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^8 + X^5 + X^4 + X^2 + 1$$

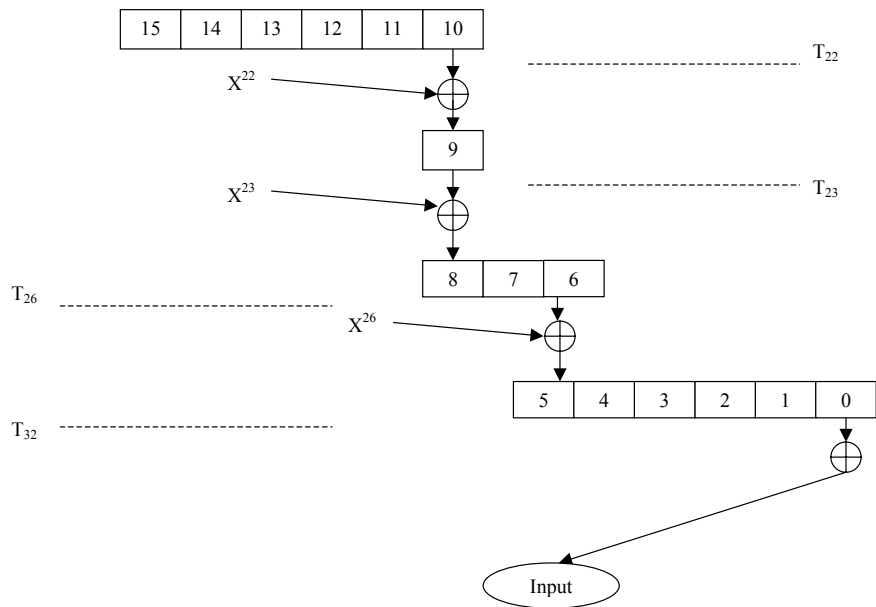
The RTL shown above can be implemented by following the methodology given.

- Set all shift registers to zero initially.
- The input should be directed towards the channel and shift register. All the bits of message polynomial having a zero coefficient are shifted using a shift register and all the bits of the polynomial having a coefficient of 1, the operation of X-OR is performed on them with the generator polynomial.
- The shift register contains the CRC when the last bit of input has passed.
- 12 X-OR gates need to be used.

4.1.1 Data Path Diagram



Continued



The figure above shows the data path diagram of the serial implementation of CRC generation. The whole process takes about 32 clock cycles if we assume

- The shifting through a single register takes only one clock cycle.
- There is no propagation delay through the XOR gate.

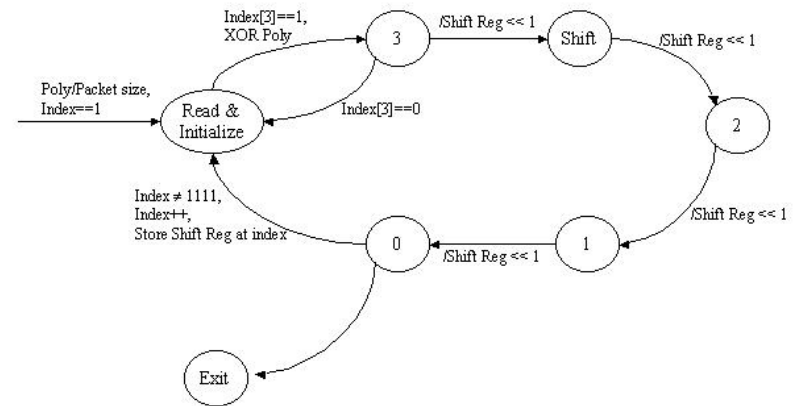
4.2.Parallel Implementation

The behavioral and RTL description mainly consists of state diagrams representing the table creation and CRC error detection.

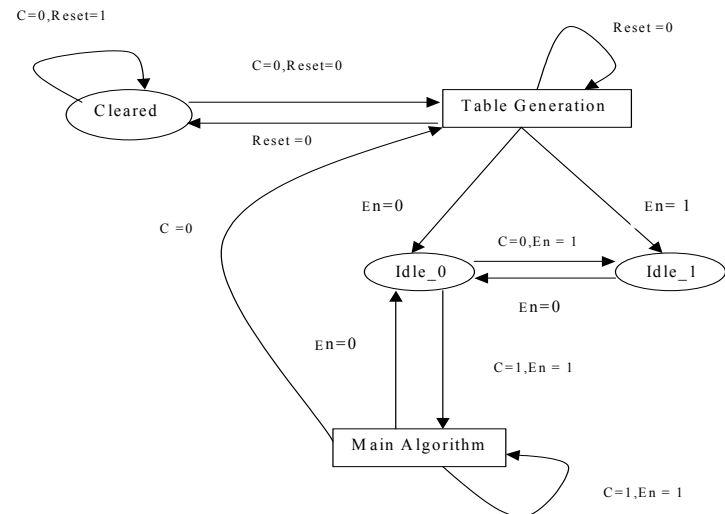
4.2.1. Table Creation

The state diagram below represents the table creation mode. Initially, the message bits are read and XOR'ed during the first state depending on the index. When the message bit is zero, only shift operation is done as shown in the states following the

state shift. After all the four bits are shifted, the value of the register XORed with the Poly will be stored in the register.



4.2.2. CRC Error Detection



The state diagram for CRC Error Detection is shown above.

- In the Cleared state, $C = 0$ indicates that the table is being constructed and $\text{Reset} = 1$ indicates that the control returns back to this state.
- When $\text{Reset} = 0$ and $C = 0$, the control gets transferred to the state where the table is constructed.
- Depending on the Enable signal, the data packets will be sent for error detection through the input data. When $\text{En} = 1$, the data packets will be sent for error detection.
- $C = 1$ means that the state is ready to check the errors on incoming data packets
- During the 'Main Algorithm' state, the error detection process takes place where in the CRC is computed and checked with that obtained in the transmitter side.
- When $C = 0$, the control is transferred to the 'Table Generation' state
- During any of these operations if Reset becomes zero the control will be transferred to 'Cleared' state.

5. Performance Estimation

The Performance of IEEE 802.11 standard chip depends on the efficiency with which the data can be transferred. Here we use a 16-bit data bus to achieve our purpose.

The performance of data path explained for the '**CRC generation**' can be analyzed by the clock cycles required for the whole process. In the data path, 32 clock cycles are required for the execution of the whole process as we assume that there is no propagation delay for the XOR gates.

At the receiver side a 16-bit serial-parallel shifter is used. So for a 32-bit long data, two clock cycles are required for the completion of the whole process. Hence a buffer is required for the successful implementation of the logic. In order to improve performance we can use a 32-bit serial parallel shifter, so the whole CRC frame can be transmitted in a single clock cycle. The performance can be enhanced further, by using a parallel implementation for a CRC-32 instead of a serial implementation. The performance estimation for CRC-32 in this spec is done for serial implementation in accordance with the requirements of different blocks involved in the design.

6 Summary and Conclusions

- The Cyclic Redundancy Check is one of the most important aspects of the IEEE 802.11 standard as it ensures error free data transmission.
- The technique of calculating the CRC is explained effectively with an example for easy implementation. Various input and output signals to the CRC encoder and decoder block have been represented clearly.
- We have discussed in this spec. different implementations (Serial and Parallel) of the CRC algorithm. We have also discussed how the CRC is appended to the Frame (body and header) at the transmitter and how it is detected at the receiver.
- The data flow diagram of the serial implementation of the CRC generation has been dealt with explicit detail. The performance estimation of the data flow diagram has been determined.
- The procedure in arriving at CRC using parallel and serial implementation has been individually dealt with emphasis on its utility and implementation to the IEEE 802.11 standard.
- CRC plays a significant role in error detection of the data frame. CRC performance can be improved by parallel implementation. The design objectives have been fulfilled.

7. References

- The IEEE 802.11 Handbook: A Designer's Companion
- Chris Borelli, "IEEE 802.3 Cyclic Redundancy Check", Xilinx Application Note, XAPP209 (v1.0) March23, 2001
- Greg Morse, "Calculating CRC's by Bits and Bytes", Programming Project, September 1986
- <http://www.owl.net/~mjhaag/vlsi/overview.htm>
- <http://www.ece.nwu.edu/~rberry/ECE333/Lectures/lec07.PDF>