CHAPTER 3

SELF HEALING PHYSICAL LAYER ARCHITECTURE

3.1 INTRODUCTION

This chapter discusses an embedded Reconfigurable Architecture (RA) to perform error detection and correction in the physical layer of OSI network. Specifically, the three popular self-healing architectures namely, Cyclic Redundancy Check (CRC), Longitudinal Redundancy Check (LRC) and Character Stuffing; are implemented in the RA. Performance metrics are used to assess the feasibility of the designed reconfigurable architecture. The proposed system shall serve as a core for future embedded server with an ability to perform error detection and correction autonomously.

3.2 FUNCTIONALITY OF OSI MODEL

The seven layer OSI model includes the application, presentation, session, transport, network, data link and physical layer. In certain layers error detection and correction schemes, security techniques i.e. cipher schemes are embedded. For ex: in the physical layer, techniques such as the CRC, LRC and character stuffing are popularly used for detecting and correcting errors. Similarly in the presentation layer encryption algorithms such as DES encryption, Blowfish, XOR Cipher, Substitution cipher, Transportation cipher are commonly used. Time out algorithms are incorporated in the session layer. Server related login issues are handled in the transport layer. Routing algorithms such as distance vector, Aloha etc. are dealt in the Network layer. Stop and wait protocol and sliding window protocol are popular in the Data link layer. In this
chapter, a self-healing hardware oriented architecture suited for physical layer is reported. Both the (i) Sender and (ii) Receiver device are embedded in the RA unit and are shown in figure 3.1.

![Architecture of error detection using FPGA in physical layer](image)

**Figure 3.1 Architecture of error detection using FPGA in physical layer**

### 3.3 INTERFERENCE EFFECTS ON NETWORK ROUTE PERFORMANCE

Analysis to study the interference effects in a sensor network (modeled as Gaussian noise) on the route performance of sensor networks with regular topology is presented. This assumption holds, in the limit, when the number of interferer’s is very high. Moreover, the propagation loss model suggests that only the frames which are relatively close to the receiver will interfere significantly. Accordingly the effects are considered for the two cases; (i) Interference from frames within (Auto Packet Interference) and (ii) Interference from frames in packet 1 and packet 2 (Cross Packet Interference).
3.3.1 Auto packet interference model

Assumption 1:

Each of the frames in the packet, denoted as frames$_{11}$, generates an interference power equals to $P_r^{(11)} = P_r = \alpha P_t / d_{link}^2$. The received bit energy is $E_b^{(11)} = E_b = P_t / R_b$.

Assumption 2:

The number of frames in the packet, denoted as frames$_{12}$, generates an interference power equal to $P_r^{(12)} = P_r / 2 = \alpha P_t / 2 d_{link}^2$. The received bit energy is $E_b^{(12)} = E_b / 2$.

Suppose that there is only a single frame in a packet 11 interfering then this happens with probability $p$. In this case, the observable is written as:

$$r = c_{\text{sig}} + c_1 + \omega_{\text{thermal}}.$$ 

There are two possible cases: (i) either $c_1 = \sqrt{E_b}$ (i.e. a ‘+1’ is transmitted) or (ii) $c_1 = -\sqrt{E_b}$ (i.e. a ‘-1’ is transmitted). Therefore, assuming that the threshold for bit detection is placed at 0,

$$P\{\text{bit error | a single node from packet }11\text{ is transmitting}\} = P\{r < 0 | C_1 = \sqrt{E_b}\} P\{C_1 = \sqrt{E_b}\} + P\{r < 0 | C_1 = -\sqrt{E_b}\} P\{C_1 = -\sqrt{E_b}\}$$

$$= \frac{1}{2} \left( P\{\sqrt{E_b} + \omega_{\text{thermal}} < 0\} + P\{\sqrt{E_b} + \omega_{\text{thermal}} < 0\}\right)$$

$$= \frac{1}{2} \left[ Q\left(\frac{\sqrt{E_b}}{2\sigma}\right) + Q(0)\right]$$

Where $\sigma = \sqrt{K T_0 / 2}$. Since $Q(0) = \frac{1}{2}$, the link BER will always be $\geq \frac{1}{4}$, regardless of the transmit power $P_t$, and

$$\lim_{E_b \to \infty} P\{\text{bit error | a single node from packet }11\text{ is transmitting}\} = \frac{1}{4} \quad \ldots \quad (3.1)$$
3.3.2 Basis components of auto packet interference model

Equation (3.1) \( \Rightarrow \lim_{E_b \to \infty} P \{\text{bit error} | \text{a single node from packet 11 is transmitting 'destructively'}\} + \lim_{E_b \to \infty} P\{\text{bit error} | \text{a single node from packet 11 is transmitting 'Constructively'}\} \times P\{\text{the node transmits 'Constructively'} | \text{it is transmitting}\} \)

\( \ldots (3.2) \)

The different regions when the node from packet 11 is transmitting are shown in figure 3.2.

![Figure 3.2 (a) Destructive transmission region](image)

![Figure 3.2 (b) Constructive transmission region](image)

The probability density function corresponding to figure 3.2(a) & figure 3.2(b) on the \( \lim_{E_b \to \infty} \) is shown in figure 3.3.

![Figure 3.3 Representation of density function](image)
p{ a node from packet 11 transmits ‘destructively’| it is transmitting} = \frac{1}{2}

\lim_{E_b \to \infty} P\{\text{bit error} | \text{a single node from packet 11 is transmitting ‘constructively’}\} = 0

p{a node from packet 11 transmits ‘constructively’| it is transmitting} = \frac{1}{2}

\lim_{E_b \to \infty} P\{\text{bit error} | \text{a single node from packet 11 is transmitting}\}

= \frac{1}{2} \times \frac{1}{2} = \frac{1}{4} \quad \ldots(3.3)

Thus, when a node from packet 11 is transmitting, it happens with probability \( p \) and this probability depends on the traffic model considered.

### 3.4 BIT ERROR CALCULATIONS

For the case where a node from packet 11 and a node from packet 12 are transmitting. The observable can be written as

\[ r = C_{\text{sig}} + C_1 + C_2 + \omega_{\text{thermal}} \]

without loss of generality, where \( c = \sqrt{E_b} \), i.e. a ‘+1’ is transmitted. In this case,

\[
P\{\text{bit error} | \text{a node from packet 11 and a node 12 are transmitting}\}

= P\{\text{bit error} | C_1, C_2\}

= P\{\sqrt{E_b} + C_1 + C_2 + \omega_{\text{thermal}} < 0\}

= P\{\omega_{\text{thermal}} < -\sqrt{E_b} - C_1 - C_2\}

= P\{\omega_{\text{thermal}} > \sqrt{E_b} + C_1 + C_2\}

= Q\left(\frac{\sqrt{E_b} + C_1 + C_2}{\sigma}\right)\]
Possible scenarios with two interferers and the corresponding link BER is listed in table 3.1.

| $C_1$       | $C_2$       | $P\{\text{bit error}|C_1, C_2\}$ |
|-------------|-------------|----------------------------------|
| $+\sqrt{E_b}$ | $+\sqrt{E_b}/2$ | $Q\left(\frac{\sqrt{E_b} + \sqrt{E_b} + \sqrt{E_b}/2}{\sigma}\right) = Q\left(\frac{E_b}{\sigma}(2 + \frac{1}{\sqrt{2}})\right)$ |
| $+\sqrt{E_b}$ | $-\sqrt{E_b}/2$ | $Q\left(\frac{\sqrt{E_b} + \sqrt{E_b} - \sqrt{E_b}/2}{\sigma}\right) = Q\left(\frac{E_b}{\sigma}(2 - \frac{1}{\sqrt{2}})\right)$ |
| $-\sqrt{E_b}$ | $+\sqrt{E_b}/2$ | $Q\left(\frac{\sqrt{E_b} - \sqrt{E_b} + \sqrt{E_b}/2}{\sigma}\right) = Q\left(\frac{E_b}{\sqrt{2}\sigma}\right)$ |
| $-\sqrt{E_b}$ | $-\sqrt{E_b}/2$ | $Q\left(\frac{\sqrt{E_b} - \sqrt{E_b} - \sqrt{E_b}/2}{\sigma}\right) = Q\left(-\frac{\sqrt{E_b}}{\sqrt{2}\sigma}\right)$ |

Where the last passage is due to the fact that $\omega_{\text{thermal}}$ has an even distribution. For the four cases in table 3.1. Assuming uniform probability($\frac{1}{4}$) and independent interfering nodes,

$P\{\text{bit error}|a\ node\ from\ packet\ 11\ and\ a\ node\ from\ packet\ 12\ are$\ Transmitting $\} = \frac{1}{2} \left[ Q\left(\frac{\sqrt{E_b}}{\sigma}(2 + \frac{1}{\sqrt{2}})\right) + Q\left(\frac{\sqrt{E_b}}{\sigma}(2 - \frac{1}{\sqrt{2}})\right) + Q\left(-\frac{\sqrt{E_b}}{\sqrt{2}\sigma}\right)\right]$
Since \( \lim_{E_b \to \infty} P\{ \text{bit error} | \text{a node from packet 11 and anode from packet 12 are transmitting} \} = \frac{1}{4} \)

In this case as well, one can write

\[
\lim_{E_b \to \infty} P\{ \text{bit error} | \text{a node from packet 11 and a node from packet 12 are transmitting} \}
= p\{\text{both nodes from packet 11 and packet 12 interfere 'destructively'}\}
= p\{\text{sign}(C_1) = -\text{sign}(C_{sig})\}p\{\text{sign}(C_2) = -\text{sign}(C_{sig})\}
= \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}
\]

Denoting as \( \overline{BER}_{\text{link}} \) the average link BER and generalizing the previous analysis,

\[
\overline{BER}_{\text{link}} = \sum_{n_{11}=0}^{\text{nodes}_{11}} \sum_{n_{12}=0}^{\text{nodes}_{12}} P\{ \text{bit error} | n_{11} \text{ nodes are transmitting from packet 1 and } n_{12} \text{ nodes are transmitting from packet 2} \} \times p\{n_{11} \text{ nodes are transmitting from packet 1 and } n_{12} \text{ nodes are transmitting from packet 2} \}
\]

Taking into account the independence between the nodes and assuming that each node has probability \( p \) of transmitting, it follows that

\[
P\{n_{11} \text{ nodes are transmitting from packet 11 and } n_{12} \text{ nodes are transmitting from packet 12} \}
= \binom{\text{nodes}_{11}}{n_{11}} \binom{\text{nodes}_{12}}{n_{12}} p^{n_{11}+n_{12}} (1-p)^{1-n_{11}-n_{12}}
\]

In equation (3.5)

(i) Conditional probability and
(ii) All possible data transmit combinations need to be evaluated. For binary transmission $2^{n_{11}+n_{12}}$ equally likely combinations exist and can be evaluated as shown in figure 3.4.

![Figure 3.4 Distribution differences between interference from node 11 and node 12](image)

**Figure 3.4 Distribution differences between interference from node 11 and node 12**

### 3.5 Determining Interfering Nodes Using Sign Notation

For notational consistencies, the notation $term_{ij}(h,k)$, is used where

- $i \mapsto$ refers to the node
- $j \mapsto$ refers to the packet $i \, j$ within the node, within which there are $i \, j$ nodes;
- $h=0,\ldots,$ $nodes_{ij} \mapsto$ indicates how many nodes from packet $ij$ are transmitting;
- $k=1,\ldots, 2^h$ is an index of the possible ways in which $h$ nodes can transmit since we are considering binary transmissions, the total number of ways in which the nodes can transmit is $2^h$.

The quantity $\text{term}_{ij}(h,k)$ will be used to characterize the overall interfering signal by combining the signs of the component interfering signals. In particular for packet 11 there can be at most $n_{11}=3$ interfering nodes and for packet 12 there can be at most $n_{12}=4$ interfering nodes, as shown in figure 3.5.

![Figure 3.5 Interference from two sign components](image)

Considering the $\text{term}_{11}(2,1),\text{term}_{11}(2,2),\text{term}_{11}(2,3)$ and $\text{term}_{11}(2,4)$, when there are two nodes transmitting from packet 11, there are four different ways the interference can manifest itself. The two nodes interfere constructively (the cases with $\text{term}_{11}(2,1)$ and $\text{term}_{11}(2,4)$ with different polarities), and destructively (the cases from $\text{term}_{11}(2,2)$ and $\text{term}_{11}(2,3)$) where the total interference is zero. Similarly,

$$\text{term}_{12}(h,k) = \text{term}_{11}(h,k) h=0,\ldots,3 \quad k=1,\ldots,2^h$$

$$\text{term}_{12}(4,1) = +1+1+1+1=+4$$
\[ \text{term}_{12}(4, 2) = +1 + 1 + 1 - 1 = +2 \]

\[ \ldots \]

\[ \ldots \]

\[ \ldots \]

\[ \text{term}_{12}(4, 2^4) = -1 - 1 - 1 - 1 = -4 \]

According to the considered definitions, it always holds that \( \text{term}_{ij}(h, l) = +h \) (all the \( h \) interfering nodes from packet \( j \) are interfering constructively) and \( \text{term}_{ij}(h, 2^h) = -h \) (all the interfering nodes from packet \( j \) are interfering constructively again with different polarity). Therefore, the first probability on the right-hand side of eqn. (3.4) can be expressed as follows:

\[
P \{ \text{bit error} \mid n_{11} \text{ nodes are transmitting from packet 1 and } n_{12} \text{ nodes are transmitting from packet 2} \}
= \sum_{i_{11}=1}^{2^{n_{11}}} \sum_{i_{12}=1}^{2^{n_{12}}} P \{ \text{bit error } \mid n_{11} \text{ nodes in the } i_{11} \text{ Configuration and } n_{12} \text{ nodes in the } i_{12} \text{ Configuration} \} 
\times P \{ i_{11} \text{ th and } i_{12} \text{ th Configurations} \}
\]

\[
\frac{1}{2^{n_{11}} + n_{12}}
\]

Thus, for instance, that the transmitted signal is a ‘+1’, it follows that \( P \{ \text{bit error} \mid n_{11} \text{ nodes in the } i_{11} \text{ Configuration and } n_{12} \text{ nodes in the } i_{12} \text{ Configuration} \} \)

\[
= P \{ \sqrt{E_b} + \text{term}_{11}(n_{11}, i_{11}) \sqrt{E_b^{(11)}} + \text{term}_{12}(n_{12}, i_{12}) \sqrt{E_b^{(12)}} + \omega_{\text{thermal}} < 0 \}
\]

\[
= P \left\{ \omega_{\text{thermal}} > \sqrt{E_b} + \text{term}_{11}(n_{11}, i_{11}) \sqrt{E_b^{(11)}} + \text{term}_{12}(n_{12}, i_{12}) \sqrt{E_b^{(12)}} \right\}
\]
\[ Q \left( \frac{\sqrt{E_b + \text{term}_{11}(n_{11}, i_{11})} + \text{term}_{12}(n_{12}, i_{12})}{\sigma} \right) \]

Note that \( E_{b}^{(11)} = E_b \) and \( E_{b}^{(12)} = \frac{E_b}{2} \), one finally obtains

\[ \overline{BER}_{\text{link}} = \left( \frac{\text{nodes}_{11}}{n_{11}} \right) \left( \frac{\text{nodes}_{12}}{n_{12}} \right) p^{n_{11}+n_{12}} \cdot (1 - p)^{\text{nodes}_{11} + \text{nodes}_{12} + n_{11} - n_{12}} \frac{1}{2^{n_{11} + n_{12}}} \cdot \sum_{i_{11}=1}^{n_{11}} \sum_{i_{12}=1}^{n_{12}} Q \left( \frac{\sqrt{E_b}}{\sigma} \left( 1 + \text{term}_{11}(n_{12}, i_{11}) + \frac{\text{term}_{12}(n_{12}, i_{12})}{\sqrt{2}} \right) \right) \]  

\[ \text{(3.6)} \]

### 3.6 CONNECTIVITY: AVERAGE SUSTAINABLE NUMBER OF HOPS

The maximum sustainable number of hops \( n_{\text{sh}}^{\text{max}} \) corresponding to a maximum tolerable route BER, denoted as \( BER_{\text{route}}^{\text{max}} \), is used in this work to make the network fault tolerant. The route BER floor after a generic number \( n_h \) of hops

\[ BER_{\text{route}}^{(n_h)} = \max \{ 1 - (1 - BER_{\text{link}}^{\text{Gauss}})^{n_h}, n_h BER_{\text{link, floor}} \} \]  

\[ \text{(3.7)} \]

Where \( BER_{\text{link}}^{\text{Gauss}} \) is the BER under the Gaussian assumption and \( BER_{\text{link, floor}} \) is the link BER floor. Eqn. (3.7) is maximized by using self-healing schemes in the MAC protocol. Since the link BER given by \( BER_{\text{link}}^{\text{Gauss}} \) is \( \ll 1 \), a first-order Taylor series expansion for the first term within the maximum operation in eqn. (3.7) is used. Hence,

\[ BER_{\text{route}}^{(n_h)} \sim \max \{ n_h BER_{\text{link}}^{\text{Gauss}}, n_h BER_{\text{link, floor}} \} = n_h \max \{ BER_{\text{link}}^{\text{Gauss}}, BER_{\text{link, floor}} \} \]  

\[ \text{(3.8)} \]
Under the constraint that the maximum tolerable BER at the end of a multi-hop route is $BER_{route}^{max}$, from eqn.(3.8) the maximum number of sustainable hops is obtained by imposing the equality:

$$BER_{route}^{(n_{sh}^{max})} = BER_{route}^{max}$$

Therefore,

$$n_{sh}^{max} = \left\lfloor \frac{BER_{route}^{max}}{\max\{BER_{link}^{Gauss}, BER_{link, floor}\}} \right\rfloor$$

Thus, the main difference with respect to the ideal (no INI) case is the presence of a link BER floor, which is due to the interference. Moreover, in the link BER under the Gaussian assumption the average interference power has to be considered as well. The average sustainable number of hops is also defined as:

$$\bar{n}_{sh} = \min\{\bar{n}_{h}, n_{sh}^{max}\}.$$ 

- Full connectivity is possible, and this corresponds to a scenario with $\bar{n}_{h} < n_{sh}^{max}$, i.e. $\bar{n}_{sh} = \bar{n}_{h}$
- Full connectivity is never possible, and this corresponds to a scenario with $\bar{n}_{h} > n_{sh}^{max}$, $\bar{n}_{sh} = n_{sh}^{max}$

3.7 DESIGN OF TRANSMITTER AND RECEIVER

The transmitter acts under the physical layer of OSI model. Basically it enhances the error detection and control mechanisms of CRC and LRC. It reads the input frames given by the user and converts into a framing sequence using character stuffing. The hardware functions as a self-healing physical layer implementation suited for future embedded web server. The receiver receives the information, detects and corrects errors and retransmits it to the side-by-side configured server system.
The receiver takes the input and gives an acknowledgement to the received data before the next sequence is transmitted. During any error an out of sequence error message is displayed.

### 3.7.1 Implementation of CRC

A cyclic redundancy check (CRC) is an error detecting code designed to detect accidental changes to raw computer data. A CRC enabled device calculates a short, fixed-length binary sequence, known as the check value and appends it to the data, forming a code word. The CRC mechanism block is shown in figure 3.6.

![Figure 3.6 CRC calculation mechanisms](image)

The following code demonstrates the method of generating CRC:

```c
CRC = Calc_Crc(data, (int)strlen(data));
unsigned int Calc_Crc(unsigned char * data, int length)
reg_crc ^= data[cnt]
for (j = 0; j < 8; j++)
{
    if ((reg_crc & 0x01) != 0)
    {
        reg_crc = (unsigned short)((reg_crc >> 1) ^ 0xA001);
    }
    else
    {
        reg_crc = (unsigned short)(reg_crc >> 1);
    }
    cnt++;
    length--;
return reg_crc;
}
```

The flowchart illustrating the above is shown in figure 3.7.
Figure 3.7 Generation of CRC
3.7.2 Implementation of CRC method

Cyclic redundancy check (CRC) is a type of function that takes as input a data stream of any length, and produces an output. Bit strings are treated as representation of polynomials with coefficients of ‘0’ and ‘1’. The sender and receiver must agree upon the generator polynomial in advance. The size of data must be greater than the size of the generator polynomial to compute the checksum. The computed checksum is appended to the transmitting frame. If the receiver gets the frame it tries dividing it by generator polynomial, if there is a reminder there has been a transmission error, else no error.

The CRC method implementation in two ways: (i) No error case
(ii) Error case

Case (i) No error case

Sender side:

Step 1: Read the input from “input.txt”. Parse the file content.
Step 2: Call the In-built function read file.
   (i) Convert the data string to be transmitted into binary representation
   (ii) Get ASCII value of the first character.
   (iii) Divide the ASCII value by 2; append the remainder to a string.
       The quotient is taken as the next number for the division.
Step 3: Repeat above division processes until the quotient of the number is less than 2.
Step 4: Finally we get the remainder string having 1’s and 0’s, which is the binary representation of that character.
Step 5: Continue above process for all the characters in the data string.
Step 6: Append 12 ‘0’s to the final binary string formed by the above step.
Step 7: Divide the binary converted data string using the generator polynomial (given above CRC12).

Step 8: Take the MSB 13 bits from the binary data string.

Step 9: If the leading bit of the binary data string is ‘0’, do an XOR operation with the 13 bits of ‘0’s and get the remainder which is also 13 bits.

Step 10: Else if the leading bit of the binary data string is ‘1’, do an XOR operation with the 13 bits with binary converted CRC 12 polynomial and get the remainder which is also 13 bits.

Step 11: The first bit of the remainder is left out and the remainder is made to 12 bits.

Step 12: If there are successive bits in the data, make the previous remainder bits to 13 bits by bringing down the next bit in the data.

Step 13: Convert the final remainder into HEXA equivalent character.

Step 14: Divide the final remainder into blocks of 4 bits.

Step 15: Convert each block into equivalent HEXA character.

Step 16: Thus the output of this whole process will be a 3- nibble checksum that is attached to the transmitting frame.

**Receiver side:**

Step 1: Repeat the steps 1 to 4 of the sender side CRC coding and get the CRC value of the data.

Step 2: Convert the HEXA equivalent string to its binary format. This is the Hexadecimal to binary conversion process.

Step 3: Append the binary format of the above step to the CRC variable. Then append 12 ‘0s’ to the CRC variable.
Step 4: Repeat the steps 6 and 7 of the sender side CRC coding to calculate the CRC value for the receiver side.

Case (ii): Error case

Sender side:

Step 1: Read the input from “input.txt”. Parse the file content.

Step 2: Call the In-built function read file.

(i) Convert the data string to be transmitted into binary representation.
(ii) Get ASCII value of the first character.
(iii) Divide the ASCII value by 2; append the remainder to a string.
The quotient is taken as the next number for the division.

Step 3: Repeat above division processes until the quotient of the number is less than 2.

Step 4: Finally we get the remainder string having 1’s and 0’s, which is the binary representation of that character.

Step 5: Continue above process for all the characters in the data string.

Step 6: Append 12 ‘0’s to the final binary string formed by the above step.

Step 7: Divide the binary converted data string using the generator polynomial (given above CRC12).

Step 8: Take the MSB 13 bits from the binary data string.

Step 9: If the leading bit of the binary data string is ‘0’, do an XOR operation with the 13 bits of ‘0’s and get the remainder which is also 13 bits.
Step 10: Else if the leading bit of the binary data string is ‘1’, do an XOR operation with the 13 bits with binary converted CRC 12 polynomial and get the remainder which is also 13 bits.

Step 11: The first bit of the remainder is left out and the remainder is made to 12 bits.

Step 12: If there are successive bits in the data, make the previous remainder bits to 13 bits by bringing down the next bit in the data.

Step 13: Convert the final remainder into HEXA equivalent character.

Step 14: Divide the final remainder into blocks of 4 bits.

Step 15: Convert each block into equivalent HEXA character.

Step 16: Thus the output of this whole process will be a 3-nibble checksum that is attached to the transmitting frame.

Receiver side:

Step 1: Repeat the steps 1 to 4 of the sender side CRC coding and get the CRC value of the data.

Step 2: Make some errors in the string CRC by changing certain characters. The error string is stored in the variable CRC.

Step 3: Write the error string CRC into the file ‘Input.txt’. The file path for the file ‘Input.txt’ is the path where you have installed the Emulator application (Mandatory).

Step 4: Convert the HEXA equivalent string CRC to its binary format. This is the Hexadecimal to binary conversion process.

Step 5: Append the binary format of the above step to the CRC variable. Then append 12 ‘0s’ to the CRC variable.
Step 6: Repeat the steps 6 and 7 of the sender side CRC coding to calculate the CRC value for the receiver side.

3.7.3 CRC implementation results

The generated frame is displayed in figure 3.8.

i. Enter the “IP Address” of the node to which data is to be sent.

ii. Enter the “Text” to be sent.

iii. Click on “FORM FRAME” button.

<table>
<thead>
<tr>
<th>Figure no.</th>
<th>Details</th>
<th>Medium</th>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.8</td>
<td>CRC Frame formation and data</td>
<td>Transmission</td>
<td>Physical</td>
</tr>
<tr>
<td></td>
<td>transmission</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.9</td>
<td>Acknowledgement for receiver message</td>
<td>Receiver</td>
<td>Physical</td>
</tr>
</tbody>
</table>

The CRC value in decimal is generated and displayed in the textbox.

i. Click on the “Send Frame” button to send the frame to the destination IP.
In the receiver side (figure 3.9), the "CHECK CRC" button calculate the CRC on receiver side and verify whether the calculated CRC and received CRC match. The calculated and received CRC are displayed in the receiver and either a match or disparity is detected.
A LRC is a form of redundancy check that is applied independently to each of a parallel group of bit streams. The data is divided into transmission blocks, to which the additional check data is added. While simple longitudinal parity can only detect errors, it can be combined with additional error control coding schemes, such as a transverse redundancy check, to correct the errors. The architecture in figure 3.10 represents the LRC generation method that uses 1’s and 2’s complement.
3.8.1 Implementation of LRC

```c
LRC = Calc_LRC(data, (int)strlen(data));
{
    printf("LRC is %u",LRC);
    getchar();
    return;
}
unsigned char Calc_Crc(unsigned char *Arr, int count)
{
    unsigned char LRC = 0x00;
    int index;
    for (index = 0; index < count; index++)
    {
        LRC = (byte)(Arr[index] + LRC);
    }
    LRC = (byte)(0xFF - LRC); // 1's complement
    LRC = (byte)(LRC+1); // 2's complements
    return (LRC);
}
```

The flow chart illustrating the above is shown in figure 3.11
3.8.2 LRC implementation results

Initially, it reads the IP Address and takes the text to form the frame and generate the table with byte and ASCII values. Figure 3.12 represents the LRC value, in figure 3.13 at the receiver the LRC is again calculated from the received data and checked for matching.
Figure 3.12 LRC transmitter message

Figure 3.13 LRC matching check at receiver
3.9 CHARACTER STUFFING

In certain communication protocols, each frame starts with the ASCII character sequence DLESTX and ends with the sequence DLEETX. If the destination ever loses synchronization, it only has to look for DLESTX and DLEETX characters. If however, binary data is being transmitted then there exists a possibility of the characters DLESTX and DLEETX occurring in the data. Since this can interfere with the framing, a technique called character stuffing is used.

3.9.1 Implementation of character stuffing

The pseudo code for the character stuffing is given as follows:

```c
FRAMESTART=getchar();
{
    printf("Enter End of Frame:");
    fflush(stdin);
    FRAME END=getchar();
    {
        char_stuff(datain,dataout,FRAMESTART,FRAMEEND);
        {
            printf("STUFFED FRAME: is %s\n",dataout);
            printf("Press Any Key to UnStuff\n");
            fflush(stdin);
            getchar();
        }
        char_unstuff(dataout,datain,FRAMESTART,FRAMEEND);
        {
            printf("UNSTUFFED DATA: is %s\n",datain);
            getchar();
            return;
        }
    }
}
```

The flow chart illustrating the above is shown in figure 3.14.
Figure 3.14 Generation of character stuffing and unstuffing
The sender’s data link layer inserts an ASCII DLE character just before the DLE character in the data (figure 3.15).

**Figure 3.15 Transmitter character stuffing**

The receiver’s data link layer removes this DLE (figure 3.16) before giving this data to the upper layer.

**Figure 3.16 Character stuffing implementation**
Figure 3.17 shows the receiver window, the frame received is displayed in the table. The STX and ETX are entered and the original data is unstuffed. If the STX and ETX is other than the one entered at the transmitter, then the text box shows frame error. However, character stuffing is closely associated with 8-bit characters and this is a major hurdle in transmitting arbitrary sized characters.

![Figure 3.17 Receiver with character unstuffing](image)

### 3.10 CHAPTER SUMMARY

In this work, interference effects on node placement, maximum no. of hops etc. were studied. To mitigate the errors introduced due to the interference automatic error detection and correction schemes for physical layer models is presented. This work supplements the Xilinx verification framework for sensor network security protocol implementations. The key advantage of proposed OSI model is that it automatically extracts verifiable models from physical layer implementations. The approach described in this work auto-generates self-validating models from partial specification of the
system there by providing annotations in the implementation itself. The proposed approach thus makes it easier to maintain correspondence between the error detection schemes, which is difficult to achieve in manual verification methods. It also brings the advantages of explicit-state model checking to the verification and error correction in OSI network applications.