Department of Computer Engineering
Class: T.Y. B.Tech Semester: V
Course Code: DJ19CEL405 Course Name: Computer Networks Lab

Name: Arnav Deo	SAP ID: 60004230285
Date of Performance: 21/08/2024	Date of Submission: 28/08/2024

Experiment No: 3

Aim: Write a program to implement error detection and correction

Techniques: CRC, Hamming(4, 7) implementation

Program:

```
#include <stdio.h>


int get_bit_count(int num)
{
    int bit_shift = 1;
    while (num ⩾ (1 << bit_shift))
    {
         bit_shift++;
    }
    return bit_shift;
}
int calculate_crc(int payload, int divisor)
{
    int div_bit_count = get_bit_count(divisor);
    int payload_bit_count = get_bit_count(payload);
    int shifted_payload = payload << (div_bit_count - 1);
    int shifted_payload_bit_count =
get_bit_count(shifted_payload);
    int div_shift_bit_count = shifted_payload_bit_count -
```

```
div_bit_count;

    int shifted_div = divisor << div_shift_bit_count;

    printf("Payload: %08b [%d]\nDivisor: %08b [%d]\nShiftPl: %08b
[%d]\n", payload, payload_bit_count, divisor, div_bit_count,
shifted_payload, shifted_payload_bit_count);

    printf("ShiftDv: %08b [%d]\n", shifted_div,
div_shift_bit_count);

    while (div_shift_bit_count)

    {

        printf("[%d] %08b ^ %08b = %08b\n", div_shift_bit_count,
shifted_payload, shifted_div, shifted_payload ^ shifted_div);

        shifted_payload ^= shifted_div;

        shifted_div >>= 1;

        div_shift_bit_count -= 1;

    }

    return shifted_payload;

}


int main()

{

    int payload = 0;

    int divisor = 1;

    printf("Enter payload: ");

    scanf(" %d", &payload);

    printf("Enter divisor: ");

    scanf(" %d", &divisor);

    int crc = calculate_crc(payload, divisor);
```

```
    printf("Calculated CRC is %b\n", crc);
}


#include <stdio.h>
#include <stdlib.h>


int get_bit_count(int num)
{
    int bit_shift = 1;
    while (num ⩾ (1 << bit_shift))
    {
         bit_shift++;
    }
    return bit_shift;
}


int get_parity_bit_count(int message)
{
    int m = get_bit_count(message);
    int p = 1;
    while ((1 << p) < (p + m + 1))
    {
         p++;
    }
    return p;
}
```

```
int get_bit_from_lsb(int val, int pos)
{
    return (!!(val & 1 << pos)) ? 1 : 0;
}


int get_bit_from_msb(int val, int pos)
{
    int bc = get_bit_count(val);
    return get_bit_from_lsb(val, bc - (pos + 1));
}


int calculate_hamming_code(int message)
{
    int m = get_bit_count(message);
    int p = get_parity_bit_count(message);
    int total_bit_count = m + p;
    printf("Using %d bits for the message with %d m bits and %d p
bits\n", total_bit_count, m, p);
    int *parity_offset_map = malloc(sizeof(int) * p);
    if (parity_offset_map == NULL)
    {
         return 0;
    }
    int ham_mask = 0;
    for (int p_bit_msb_offset_mul = 0; p_bit_msb_offset_mul < p;
```

```
p_bit_msb_offset_mul++)
    {
        int parity_bit_offset = ((total_bit_count - 1) - ((1 <<
p_bit_msb_offset_mul) - 1));
        *(parity_offset_map + p_bit_msb_offset_mul) =
parity_bit_offset;
        ham_mask |= 1 << parity_bit_offset;
    }

    int ham_code = 0;
    int src_bit_index = 0;
    for (int i = 0; i < total_bit_count; i++)
    {
        int is_mask = get_bit_from_lsb(ham_mask, i);
        if (is_mask)
        { /* Do nothing */
        }
        else
        {
            // Get the bit from the actual value
            int src_bit = get_bit_from_lsb(message,
src_bit_index);
            ham_code |= src_bit << i;
            src_bit_index++;
        }
    }
```

```
    printf("Your Message in Binar: %032b\n", message);

    printf("Hamming Bit Positions: %032b\n", ham_mask);

    printf("Code with Mesg Filled: %032b\n", ham_code);


    for (int i = 0; i < p; i++)

    {

         int pos = 1 << i;

         // Start with 1 for odd parity, 0 for even parity

         int parity = 0;

         for (int j = 1; j < total_bit_count + 1; j++)

         {

              // Easier to count 1, 3, 5, 7 rather than 0, 2, 4,
6

              int real_pos = j - 1;

              if (j & pos)

              {

                   // The position is covered by the current
hamming bit

                   // This also includes the hamming bit as well,

                   // but it will be set to 0 always, giving us
the parity value in the end

                   int pos_bit = get_bit_from_msb(ham_code,
real_pos);

                   parity ^= pos_bit;

              }

         }
```

```
        // Good thing we are keeping track of where the parity
bit will end
        int par_offset_from_lsb = parity_offset_map[i];
        ham_code |= parity << par_offset_from_lsb;
    }

    printf("Code with Hamming ECC: %032b\n", ham_code);
    printf("Code only Hamming Bit: %032b\n", ham_code &
ham_mask);

    return ham_code;
}

int main()
{
    int message = 0;
    printf("Enter message: ");
    scanf(" %d", &message);
    int hamming_code = calculate_hamming_code(message);
    return 0;
}
```

Screenshots:

```
arnitdo  Experiment 3 CRC and ECC  ✓  !!
gcc -w -O0 crc.c -o crc && ./crc
Enter payload: 100
Enter divisor: 3
Payload: 0000000001100100 [7]
Divisor: 0000000000000011 [2]
ShiftPl: 0000000011001000 [8]
ShiftDv: 0000000011000000 [6]
[6] 0000000011001000 ^ 0000000011000000 = 0000000000001000
[5] 0000000000001000 ^ 0000000001100000 = 0000000001101000
[4] 0000000001101000 ^ 0000000000110000 = 0000000001011000
[3] 0000000001011000 ^ 0000000000011000 = 0000000001000000
[2] 0000000001000000 ^ 0000000000001100 = 0000000001001100
[1] 0000000001001100 ^ 0000000000000110 = 0000000001001010
Calculated CRC is 0
```

```
arnitdo  Experiment 3 CRC and ECC  ✓  gcc -w hamming.c -o hamming -lm && ./hamming
Enter message: 11
Using 7 bits for the message with 4 m bits and 3 p bits
Your Message in Binar: 00000000000000000000000000001011
Hamming Bit Positions: 00000000000000000000000001101000
Code with Mesg Filled: 00000000000000000000000000010011
Code with Hamming ECC: 00000000000000000000000000010011
Code only Hamming Bit: 00000000000000000000000000000000
```

Conclusion:

Thus, we have studied and implemented the various error detection and correction.