OBJECTIVE

You will give it data blocks (say any text data or a sequence of numbers), encode it, introduce errors, decode.

INTRODUCTION

Cyclic Redundancy Check (CRC) is a method to check whether the signal sent from transmitter side to receiver side does not contain any error. When data is transferred from transmitter data is appended with checksum which is calculated by dividing modulo 2 division of message data with suitable primitive polynomial. This appended version of data is called code. when this code is again divided with same primitive polynomial gives the remainder zero if the message which was sent does not contain any errors.

```
        110101
      ------------
10011)1100001010
      10011
      -----
       101101010
       10011
        -----
        01011010
         10011
         -----
         010110
          10011
          -----
          0101      → CHECKSUM
```

```
Message: 1100001010
Polynomial:10011
Message with checksum:1100010100101
```

METHOD

Transmitter Side

We know that the message bit is 12 bit and For 16- bit CRCs, the CCITT (an international telecommunications standards organization) has adopted the "CCITT polynomial" which is $x_{16} + x_{12} + x_5 + 1$ having bits "10001000000010000" .We have to Divide Modulo-2 division of 16 bits CRCs and Message bits to get 16 bits Checksum which will be appended to 12 bit message bits to get 28 bit code which will be sent to receiver.

Binary number ''10001000000010000'' → Decimal Number "69665 "

Let the message bit be 4095 → Binary ″111111111111"

Modulo-2 Division

Divisor is 10001000000010000 (17 bit)

Dividend is 111111111111(12 bit)

This is not possible as dividend is small so to make division possible add 16 zeros to message and 11 bits to polynomial so that in every iteration 1 top bit can be discarded.

Appending Zero Bits to message "1111111111110000000000000000" making 28 bits.

Appending Zero bits to polynomial "1000100000001000000000000000" making 28 bits.

After Doing XOR in every iteration 1 bit is discarded from MSB and 1 bit in LSB is appended. This is done 28 bits-16 bits =12 bits. If more than one zero appear in the MSB of message after the XOR operation than that number of Zeros are appended to message.

These operations are done for each and every sample to generate the crc code of 28 bits for each and every sample.

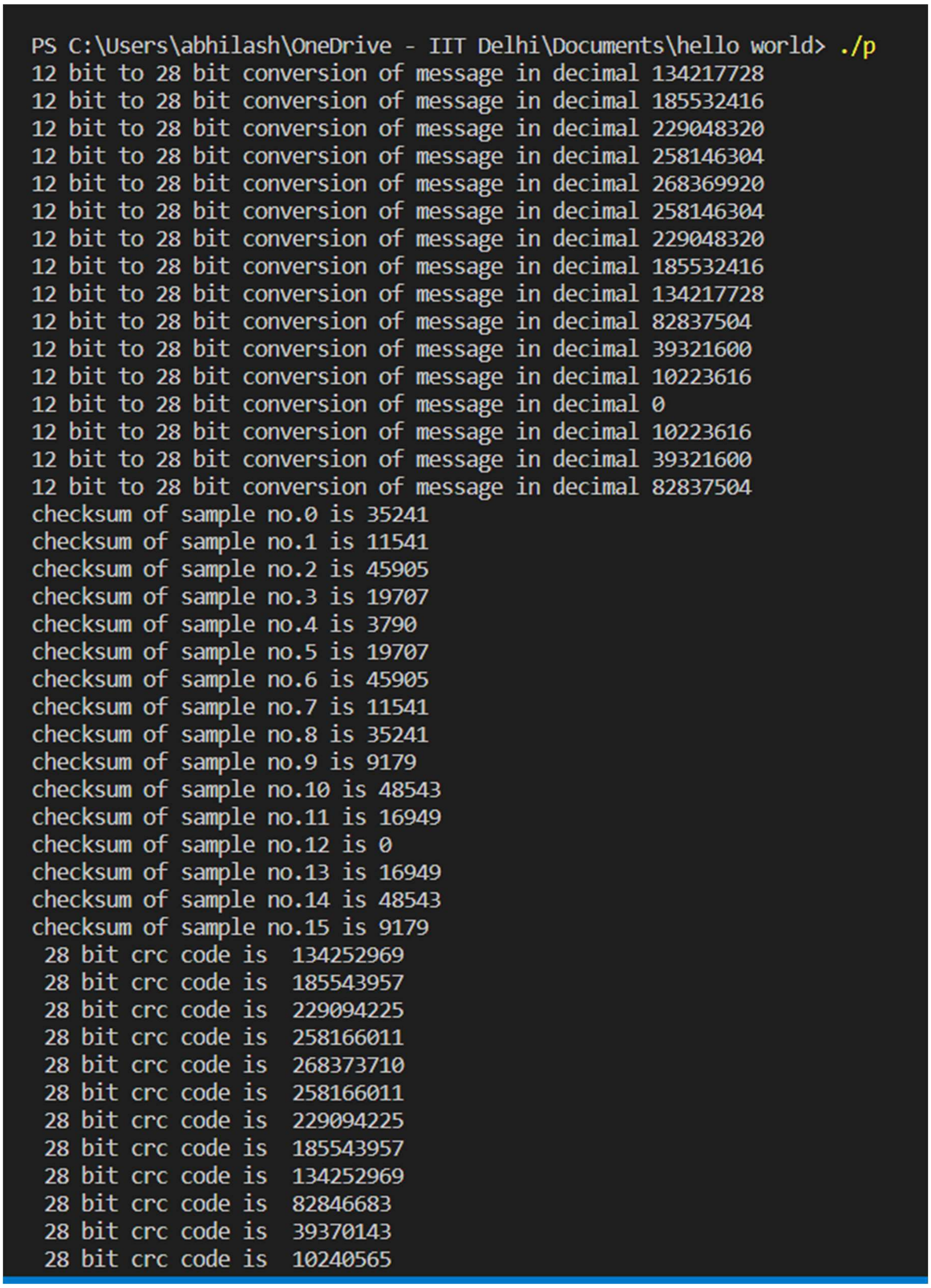
The Received 28 bits CRC code is again divided by 16-bit generating polynomial and if the remainder is zero than the received message bits are error free and otherwise the received message bits contains error.

RESULT

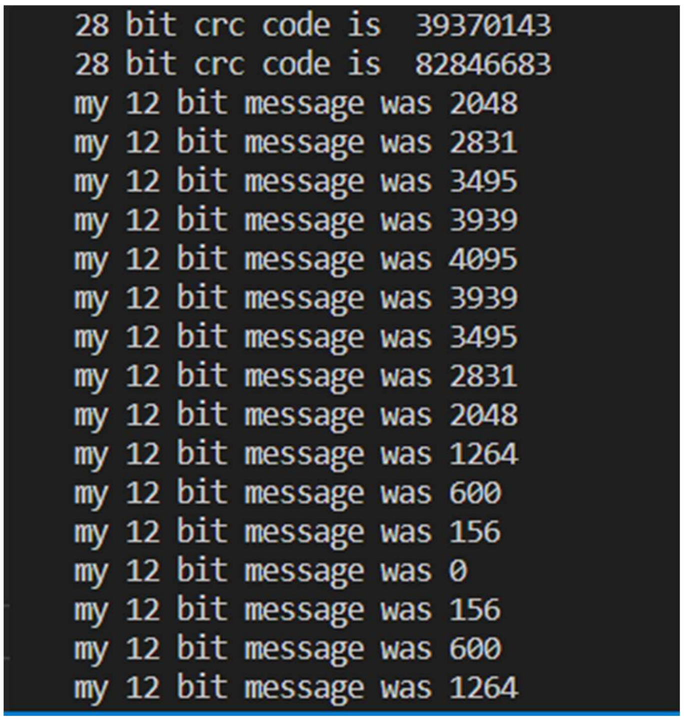
Errors are not Inserted

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

void main()
{
    uint32_t message[16] = {2048, 2831, 3495, 3939, 4095, 3939, 3495, 2831, 2048,
1264, 600, 156, 0, 156, 600, 1264};
    uint32_t r[16], array2[16],array3[16];
    uint32_t crccode[16];
    uint32_t rem[16];
    uint32_t errorcode[16];
    uint32_t correcteddata[16];
    uint16_t ERRORCODE[16];


    uint32_t genpoly = 69665;

    genpoly = genpoly << 11;
// Transmitter Side
    for (int x = 0; x < 16; x++)
    {
        array2[x] = message[x] << 16;
        r[x] = array2[x];
    }
    for (int k = 0; k < 16; k++)
    {
        printf("12 bit to 28 bit conversion of message in decimal %d\n",
array2[k]);
    }
    for (int j = 0; j < 16; j++)
    {
        for (int i = 0; i < 12; i++)
        {
            if (r[j] & 0x8000000)
            {
                r[j] = array2[j] ^ genpoly;
                r[j] = r[j] << 1;
                array2[j] = r[j];
            }
            else
            {
                r[j] = r[j] << 1;
                array2[j] = r[j];
            }
        }
        array2[j] = array2[j] >> 12;
    }
    for (int k = 0; k < 16; k++)
    {
        printf("checksum of sample no.%d is %d \n",k, array2[k]);
    }
    for (int x = 0; x < 16; x++)
    {
        crccode[x] = (message[x] << 16) | array2[x];
        rem[x] = crccode[x];
        array3[x]=crccode[x];

    }
    for (int k = 0; k < 16; k++)
    {
        printf(" 28 bit crc code is  %d\n", crccode[k]);
    }

    for (int j = 0; j < 16; j++)
    {
        for (int i = 0; i < 12; i++)
        {
            if (rem[j] & 0x8000000)
            {
                rem[j] = crccode[j] ^ genpoly;
                rem[j] = rem[j] << 1;
                crccode[j] = rem[j];
            }

            else
            {
                rem[j] = rem[j] << 1;
                crccode[j] = rem[j];
            }
        }
        rem[j] = crccode[j] >> 12;
    }
    for (int k = 0; k < 16; k++)
    {
        if(rem[k]==0){
             correcteddata[k] = array3[k] >> 16;
             printf(" my 12 bit message was %d\n", correcteddata[k]);
        }
        else
        {
            correcteddata[k] =ERRORCODE[k]>>16;
            printf("there is error in sample no. %d and sample sent from
transmitter is %d and recieved on reciever side is
%d  \n",k,(array3[k]>>16),correcteddata[k]);
        }
    }
}
```

Output

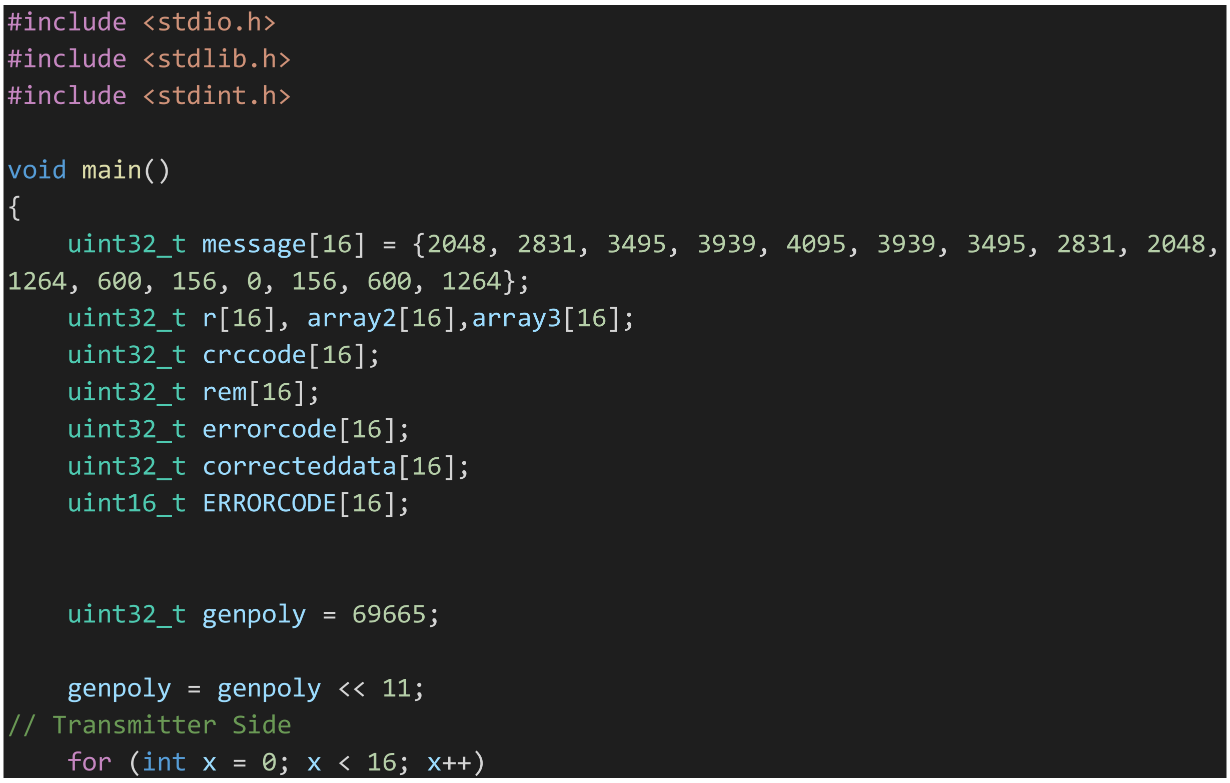


```
PS C:\Users\abhilash\OneDrive - IIT Delhi\Documents\hello world> ./p
12 bit to 28 bit conversion of message in decimal 134217728
12 bit to 28 bit conversion of message in decimal 185532416
12 bit to 28 bit conversion of message in decimal 229048320
12 bit to 28 bit conversion of message in decimal 258146304
12 bit to 28 bit conversion of message in decimal 268369920
12 bit to 28 bit conversion of message in decimal 258146304
12 bit to 28 bit conversion of message in decimal 229048320
12 bit to 28 bit conversion of message in decimal 185532416
12 bit to 28 bit conversion of message in decimal 134217728
12 bit to 28 bit conversion of message in decimal 82837504
12 bit to 28 bit conversion of message in decimal 39321600
12 bit to 28 bit conversion of message in decimal 10223616
12 bit to 28 bit conversion of message in decimal 0
12 bit to 28 bit conversion of message in decimal 10223616
12 bit to 28 bit conversion of message in decimal 39321600
12 bit to 28 bit conversion of message in decimal 82837504
checksum of sample no.0 is 35241
checksum of sample no.1 is 11541
checksum of sample no.2 is 45905
checksum of sample no.3 is 19707
checksum of sample no.4 is 3790
checksum of sample no.5 is 19707
checksum of sample no.6 is 45905
checksum of sample no.7 is 11541
checksum of sample no.8 is 35241
checksum of sample no.9 is 9179
checksum of sample no.10 is 48543
checksum of sample no.11 is 16949
checksum of sample no.12 is 0
checksum of sample no.13 is 16949
checksum of sample no.14 is 48543
checksum of sample no.15 is 9179
 28 bit crc code is  134252969
 28 bit crc code is  185543957
 28 bit crc code is  229094225
 28 bit crc code is  258166011
 28 bit crc code is  268373710
 28 bit crc code is  258166011
 28 bit crc code is  229094225
 28 bit crc code is  185543957
 28 bit crc code is  134252969
 28 bit crc code is  82846683
 28 bit crc code is  39370143
 28 bit crc code is  10240565
```



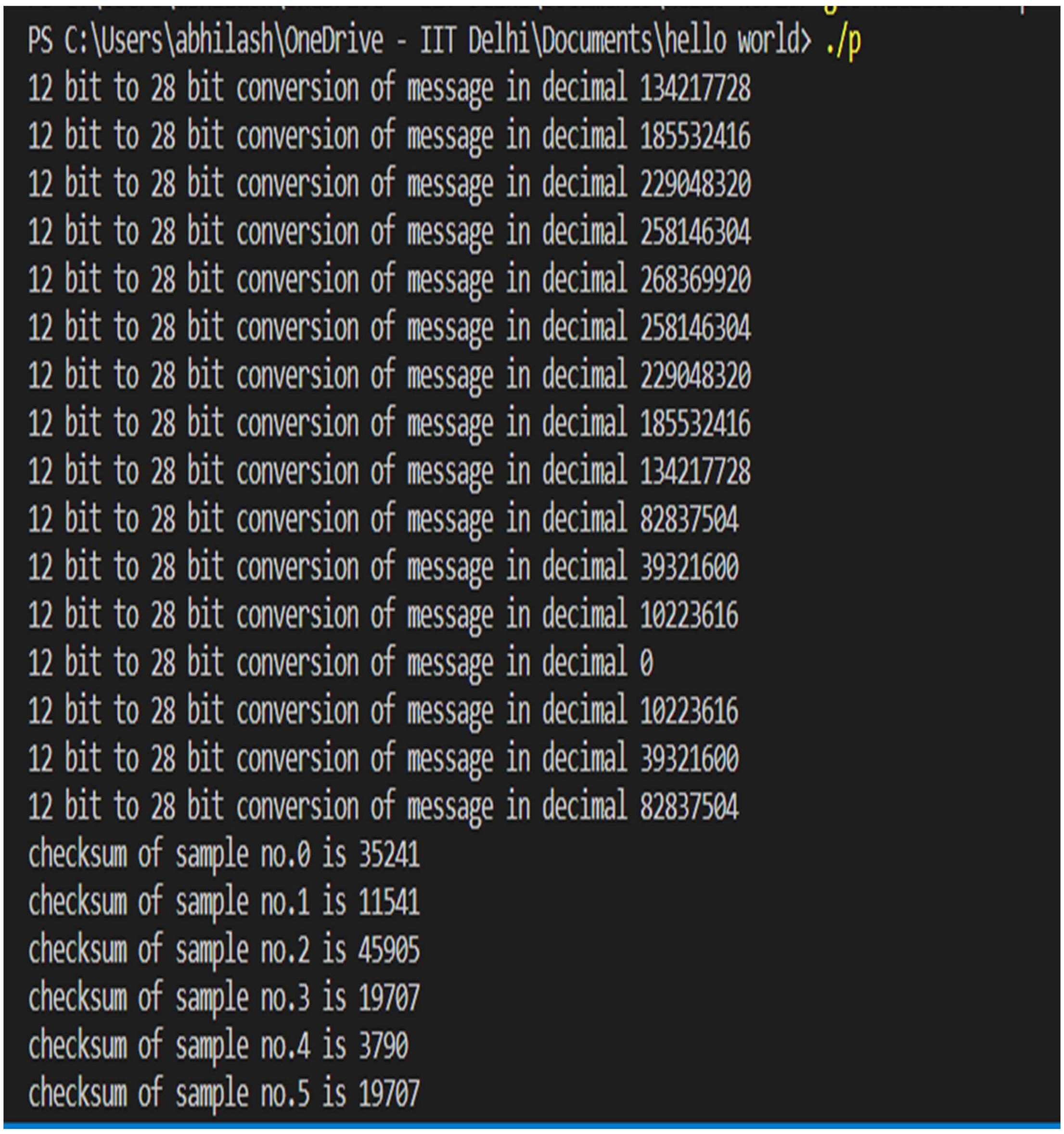
```
28 bit crc code is  39370143
28 bit crc code is  82846683
my 12 bit message was 2048
my 12 bit message was 2831
my 12 bit message was 3495
my 12 bit message was 3939
my 12 bit message was 4095
my 12 bit message was 3939
my 12 bit message was 3495
my 12 bit message was 2831
my 12 bit message was 2048
my 12 bit message was 1264
my 12 bit message was 600
my 12 bit message was 156
my 12 bit message was 0
my 12 bit message was 156
my 12 bit message was 600
my 12 bit message was 1264
```

When some errors are introduced

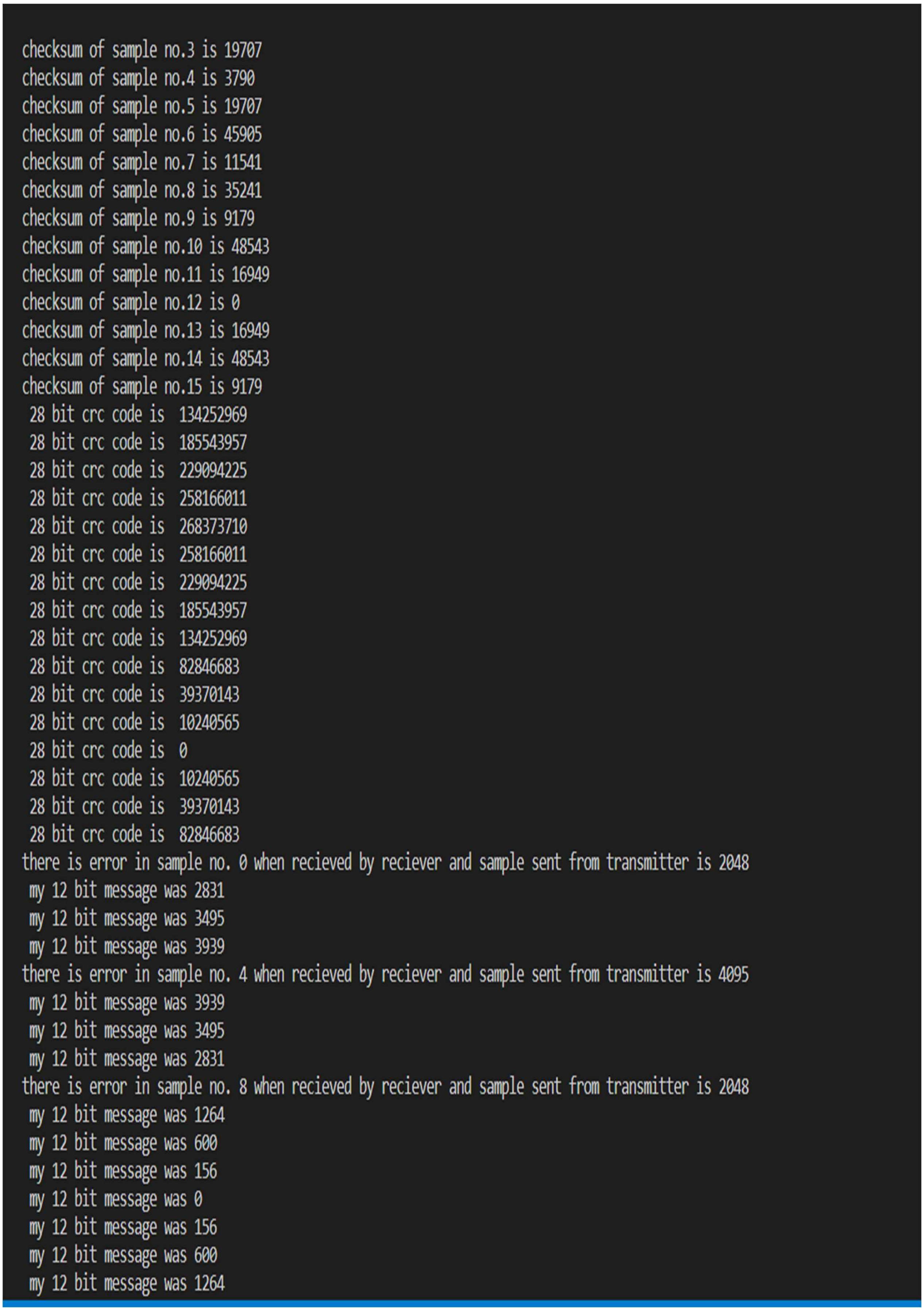


```c
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

void main()
{
    uint32_t message[16] = {2048, 2831, 3495, 3939, 4095, 3939, 3495, 2831, 2048,
1264, 600, 156, 0, 156, 600, 1264};
    uint32_t r[16], array2[16],array3[16];
    uint32_t crccode[16];
    uint32_t rem[16];
    uint32_t errorcode[16];
    uint32_t correcteddata[16];
    uint16_t ERRORCODE[16];


    uint32_t genpoly = 69665;

    genpoly = genpoly << 11;
// Transmitter Side
    for (int x = 0; x < 16; x++)
    {
        array2[x] = message[x] << 16;
        r[x] = array2[x];
    }
    for (int k = 0; k < 16; k++)
    {
        printf("12 bit to 28 bit conversion of message in decimal %d\n",
array2[k]);
    }
    for (int j = 0; j < 16; j++)
    {
        for (int i = 0; i < 12; i++)
        {
            if (r[j] & 0x8000000)
            {
                r[j] = array2[j] ^ genpoly;
                r[j] = r[j] << 1;
                array2[j] = r[j];
            }
            else
            {
                r[j] = r[j] << 1;
                array2[j] = r[j];
            }
        }
        array2[j] = array2[j] >> 12;
    }
    for (int k = 0; k < 16; k++)
    {
        printf("checksum of sample no.%d is %d\n",k, array2[k]);
    }
    for (int x = 0; x < 16; x++)
    {
        crccode[x] = (message[x] << 16) | array2[x];
        rem[x] = crccode[x];
        array3[x]=crccode[x];

    }
    for (int k = 0; k < 16; k++)
    {
        printf(" 28 bit crc code is  %d\n", crccode[k]);
    }
   // Reciever Side
         for (int k = 0; k < 16; k=k+4)
     {
         crccode[k] = array3[k] ^ (0xFEFFFFF);
          ERRORCODE[k]=crccode[k];
     }
    for (int j = 0; j < 16; j++)
    {
        for (int i = 0; i < 12; i++)
        {
            if (rem[j] & 0x8000000)
            {
                rem[j] = crccode[j] ^ genpoly;
                rem[j] = rem[j] << 1;
                crccode[j] = rem[j];
            }

            else
            {
                rem[j] = rem[j] << 1;
                crccode[j] = rem[j];
            }
        }
        rem[j] = crccode[j] >> 12;
    }
    for (int k = 0; k < 16; k++)
    {
        if(rem[k]==0){
             correcteddata[k] = array3[k] >> 16;
             printf(" my 12 bit message was %d\n", correcteddata[k]);
            // printf("dividing 28 bit code with generator polynomial gives
remainder %d",rem[k]);
        }
        else
        {
            correcteddata[k] =ERRORCODE[k]>>16;
            printf("there is error in sample no. %d when recieved by reciever and
sample sent from transmitter is %d  \n",k,(array3[k]>>16));
        }
    }
}
```

Output



```
PS C:\Users\abhilash\OneDrive - IIT Delhi\Documents\hello world> ./p
12 bit to 28 bit conversion of message in decimal 134217728
12 bit to 28 bit conversion of message in decimal 185532416
12 bit to 28 bit conversion of message in decimal 229048320
12 bit to 28 bit conversion of message in decimal 258146304
12 bit to 28 bit conversion of message in decimal 268369920
12 bit to 28 bit conversion of message in decimal 258146304
12 bit to 28 bit conversion of message in decimal 229048320
12 bit to 28 bit conversion of message in decimal 185532416
12 bit to 28 bit conversion of message in decimal 134217728
12 bit to 28 bit conversion of message in decimal 82837504
12 bit to 28 bit conversion of message in decimal 39321600
12 bit to 28 bit conversion of message in decimal 10223616
12 bit to 28 bit conversion of message in decimal 0
12 bit to 28 bit conversion of message in decimal 10223616
12 bit to 28 bit conversion of message in decimal 39321600
12 bit to 28 bit conversion of message in decimal 82837504
checksum of sample no.0 is 35241
checksum of sample no.1 is 11541
checksum of sample no.2 is 45905
checksum of sample no.3 is 19707
checksum of sample no.4 is 3790
checksum of sample no.5 is 19707
```



```
checksum of sample no.3 is 19707
checksum of sample no.4 is 3790
checksum of sample no.5 is 19707
checksum of sample no.6 is 45905
checksum of sample no.7 is 11541
checksum of sample no.8 is 35241
checksum of sample no.9 is 9179
checksum of sample no.10 is 48543
checksum of sample no.11 is 16949
checksum of sample no.12 is 0
checksum of sample no.13 is 16949
checksum of sample no.14 is 48543
checksum of sample no.15 is 9179
 28 bit crc code is  134252969
 28 bit crc code is  185543957
 28 bit crc code is  229094225
 28 bit crc code is  258166011
 28 bit crc code is  268373710
 28 bit crc code is  258166011
 28 bit crc code is  229094225
 28 bit crc code is  185543957
 28 bit crc code is  134252969
 28 bit crc code is  82846683
 28 bit crc code is  39370143
 28 bit crc code is  10240565
 28 bit crc code is  0
 28 bit crc code is  10240565
 28 bit crc code is  39370143
 28 bit crc code is  82846683
there is error in sample no. 0 when recieved by reciever and sample sent from transmitter is 2048
 my 12 bit message was 2831
 my 12 bit message was 3495
 my 12 bit message was 3939
there is error in sample no. 4 when recieved by reciever and sample sent from transmitter is 4095
 my 12 bit message was 3939
 my 12 bit message was 3495
 my 12 bit message was 2831
there is error in sample no. 8 when recieved by reciever and sample sent from transmitter is 2048
 my 12 bit message was 1264
 my 12 bit message was 600
 my 12 bit message was 156
 my 12 bit message was 0
 my 12 bit message was 156
 my 12 bit message was 600
 my 12 bit message was 1264
```

Cube IDE Code

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2022 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32F4xx_hal.h"
void ADC_Select_CH0 (void);
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc1;

DAC_HandleTypeDef hdac;
DMA_HandleTypeDef hdma_dac1;

TIM_HandleTypeDef htim2;

/* USER CODE BEGIN PV */

/* USER CODE END PV */
```

```c
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_DAC_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
uint32_t Vadc;
float V;
#define NS (1000)
uint32_t Wave_LUT[NS];
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_DAC_Init();
  MX_ADC1_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */
  for(int i=0; i<NS; i++)
  {
  // ADC_Select_CH0();
  // uint32_t start = HAL_GetTick();
  HAL_ADC_Start(&hadc1);
  HAL_ADC_PollForConversion(&hadc1, 100);
  Vadc = HAL_ADC_GetValue(&hadc1);
  HAL_ADC_Stop(&hadc1);
  // V = ((float)Vadc * 3300 / 4096);
  Wave_LUT[i]=Vadc;
  // uint32_t end = HAL_GetTick();
  HAL_Delay(10000/NS); // - (end - start));
  }

      uint32_t r[NS], array2[NS],array3[NS];
      uint32_t crccode[NS];
      uint32_t rem[NS];
      uint32_t transmitteddata[NS];
      uint32_t genpoly = 69665;
         genpoly = genpoly << 11;
      for (uint32_t x = 0; x < NS; x++)
      {
          array2[x] =  Wave_LUT[x] << 16;
          r[x] = array2[x];
      }

      for (uint32_t j = 0; j < NS; j++)
      {
          for (uint32_t i = 0; i < 12; i++)
          {
              if (r[j] & 0x8000000)
              {
                  r[j] = array2[j] ^ genpoly;
                  r[j] = r[j] << 1;
                  array2[j] = r[j];
              }
              else
              {
                  r[j] = r[j] << 1;
                  array2[j] = r[j];
              }
          }
          array2[j] = array2[j] >> 12;
      }
      for (uint32_t x = 0; x < NS; x++)
      {
          crccode[x] = ( Wave_LUT[x] << 16) | array2[x];
          rem[x] = crccode[x];
          array3[x]=crccode[x];
      }

//        for (uint32_t k = 0; k < NS; k=k+20)
//        {
//            array3[k] = array3[k] ^ (0xFFFFFFF);
//        }
//        for(uint32_t k=0;k<NS;k++){
//         transmitteddata[k]=array3[k]>>16;
//      }



      for (uint32_t j = 0; j < NS; j++)
      {
          for (uint32_t i = 0; i < 12; i++)
          {
              if (rem[j] & 0x8000000)
              {
                  rem[j] = crccode[j] ^ genpoly;
                  rem[j] = rem[j] << 1;
                  crccode[j] = rem[j];
              }

              else
              {
                  rem[j] = rem[j] << 1;
                  crccode[j] = rem[j];
              }
          }
          rem[j] = crccode[j] >> 12;
      }
      for (uint32_t k = 0; k < NS; k++)
      {
         if(rem[k]==0)
         {
                transmitteddata[k]=array3[k]>>16;
         }
         else
         {
              transmitteddata[k]=4095;
         }

      }
  HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, (uint32_t*)transmitteddata,
NS,DAC_ALIGN_12B_R);
  HAL_TIM_Base_Start(&htim2);
  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
```

```c
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 84;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief ADC1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_ADC1_Init(void)
{

  /* USER CODE BEGIN ADC1_Init 0 */

  /* USER CODE END ADC1_Init 0 */

  ADC_ChannelConfTypeDef sConfig = {0};
  /* USER CODE BEGIN ADC1_Init 1 */

  /* USER CODE END ADC1_Init 1 */

  /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and 
number of conversion)
  */
  hadc1.Instance = ADC1;
  hadc1.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV4;
  hadc1.Init.Resolution = ADC_RESOLUTION_12B;
  hadc1.Init.ScanConvMode = DISABLE;
  hadc1.Init.ContinuousConvMode = DISABLE;
  hadc1.Init.DiscontinuousConvMode = DISABLE;
  hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
  hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
  hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
  hadc1.Init.NbrOfConversion = 1;
  hadc1.Init.DMAContinuousRequests = DISABLE;
  hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
  if (HAL_ADC_Init(&hadc1) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure for the selected ADC regular channel its corresponding rank in the 
sequencer and its sample time.
  */
  sConfig.Channel = ADC_CHANNEL_0;
  sConfig.Rank = 1;
  sConfig.SamplingTime = ADC_SAMPLETIME_3CYCLES;
  if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN ADC1_Init 2 */

  /* USER CODE END ADC1_Init 2 */

}

/**
  * @brief DAC Initialization Function
  * @param None
  * @retval None
  */
static void MX_DAC_Init(void)
{

  /* USER CODE BEGIN DAC_Init 0 */

  /* USER CODE END DAC_Init 0 */

  DAC_ChannelConfTypeDef sConfig = {0};
  /* USER CODE BEGIN DAC_Init 1 */

  /* USER CODE END DAC_Init 1 */

  /** DAC Initialization
  */
  hdac.Instance = DAC;
  if (HAL_DAC_Init(&hdac) != HAL_OK)
  {
    Error_Handler();
  }

  /** DAC channel OUT1 config
  */
  sConfig.DAC_Trigger = DAC_TRIGGER_T2_TRGO;
  sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
  if (HAL_DAC_ConfigChannel(&hdac, &sConfig, DAC_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN DAC_Init 2 */

  /* USER CODE END DAC_Init 2 */

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 45000;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
```

```c
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Stream5_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOA_CLK_ENABLE();

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

MATLAB CODE

```matlab
clear; clc; % Clear The Previous Points
RES = 12; % Set The DAC Resolution
OFFSET = 0; % Set An Offset Value For The DAC Output
%------------[ Calculate The Sample Points ]-------------
T = 0:0.01:1;
Y4 = cos(4*pi*T);
Y4= Y4 + 1;
Y4 = Y4*((2^RES-1)-2*OFFSET)/(2+OFFSET);
Y4 = round(Y4);
y=de2bi(Y4, 12, 'left-msb');
gen = crc.generator([1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1]);
msg = reshape(y,12*101,1);
encoded = generate(gen,msg);
det = crc.detector([1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 1]);
encoded(1) = ~encoded(1);      % Introduce an error
[outdata error] = detect(det, encoded); % Detect the error
noErrors = isequal(msg, outdata)        % Should be 0
 error                                  % Should be 1
```

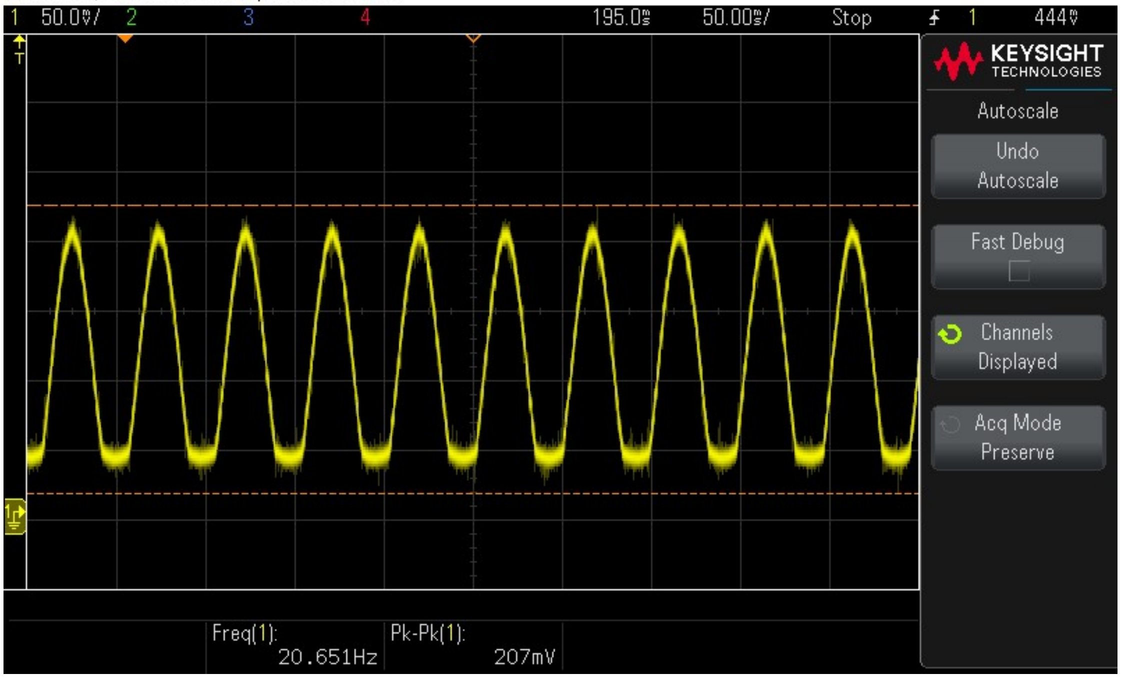
OUTPUT

```
   logical

    0


error =

   logical

    1
```

RESULT

Original message signal direct from the Function Generator

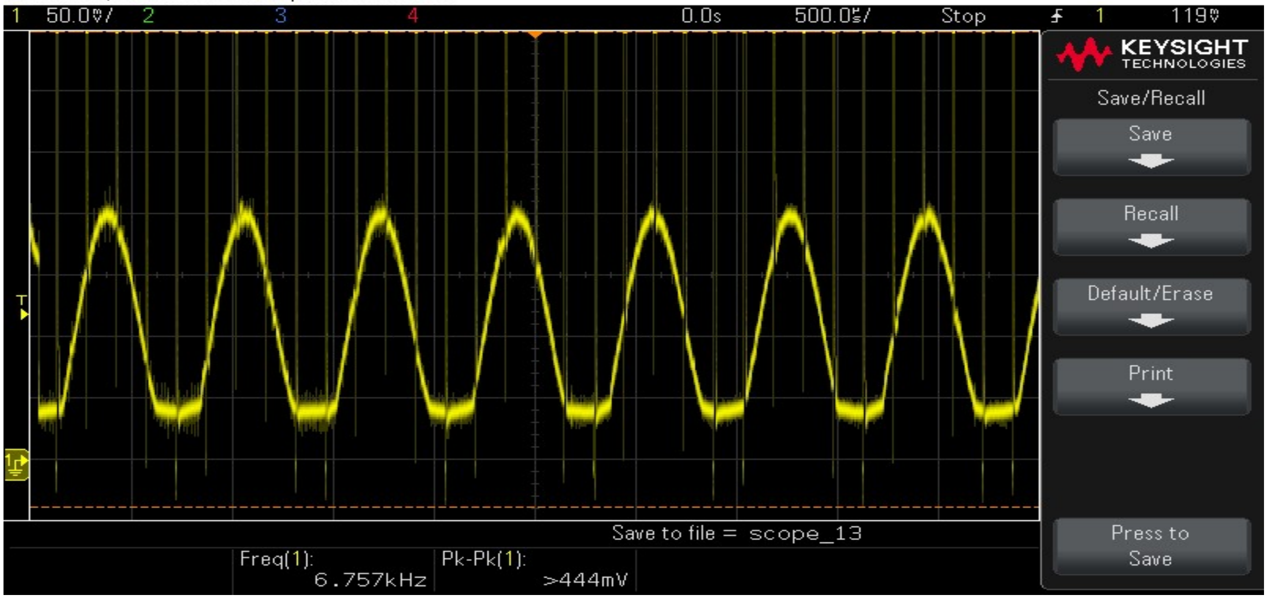
DSOX1204A, CN61166219: Sat Sep 17 10:34:42 2022



Freq(1): 20.651Hz
Pk-Pk(1): 207mV

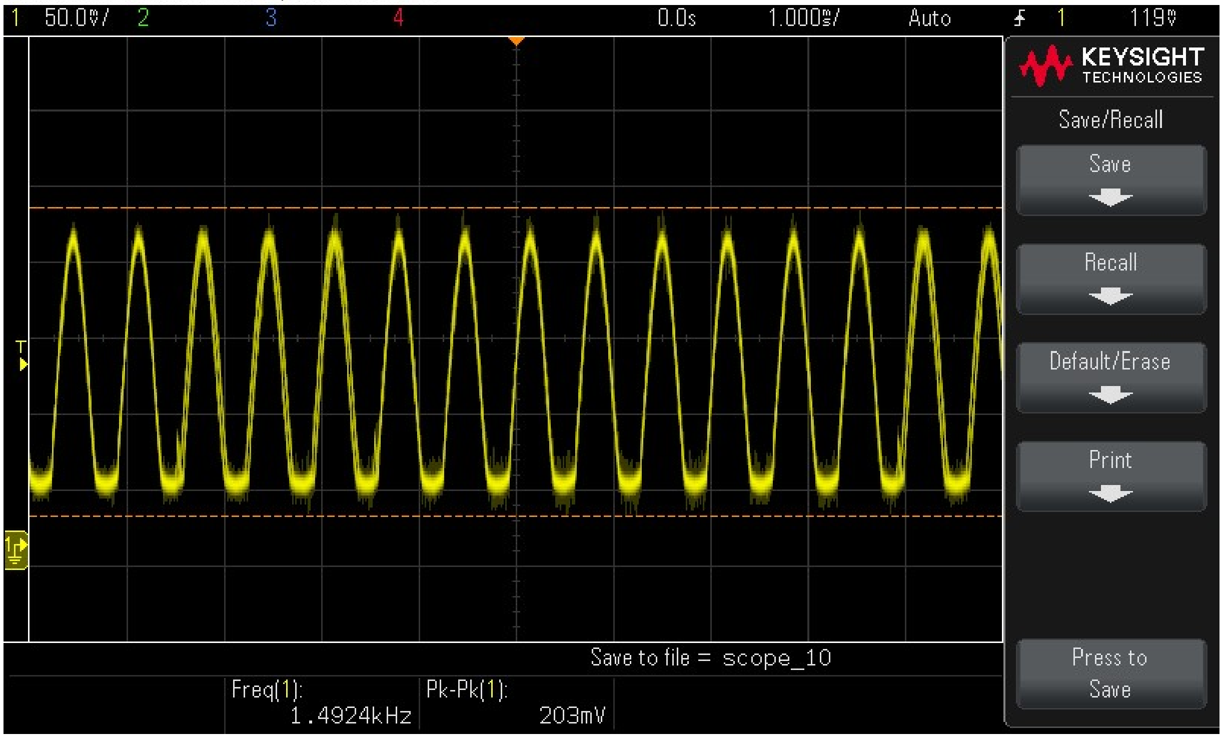
Some bits are flipped

DSOX1204A, CN61166219: Sat Sep 17 10:27:36 2022

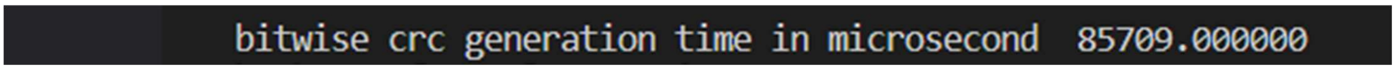


Recovered Samples from the CRC Code

DSOX1204A, CN61166219: Sat Sep 17 10:23:29 2022



Time Calculation for bitwise Operation



```
bitwise crc generation time in microsecond  85709.000000
```

CONCLUSION

- Time Taken for generation of CRC Code is 85709 microsecond using bitwise implementation.
- Checksum is generated for each sample by implementing Modulo-2 Division
- In MATLAB we are appending checksum at last of the whole data bits
- If there is no error in code then the remainder is zero and code is transmitted from transmitted to receiver.