OBJECTIVE

The objective is to use the generated PN sequence to scramble and descramble a binary image. Use a Linear-feedback shift register for pseudo-noise sequence generation. Store that in an array. Create an array containing a sample image say car.PNg. Show image on screen. Perform the scrambling operation and show the scrambled image. Calculate the autocorrelation of the PN sequence of the two images and the cross co-relation between the images. Descramble the image and display. Compute the autocorrelation of the descrambled image. After you have done this part in MATLAB and then using STMCube (only the scrambling and descrambling part), use the parts that you developed as part of the previous lab to demonstrate this on a sin/cos wave and show the scrambled and descrambled waveforms on the oscilloscope.

INTRODUCTION

Pseudo Noise (PN) or Pseudorandom sequence is a binary sequence whose autocorrelation resembles almost to white noise. It is periodic and have some characteristics of random white sequence within that period and is generated by linear feedback shift registers (LFSR). It is used widely in communication, scrambling and cryptography.

Scrambling of image

$x_s[n]=(x[n]+s[n])\text{mod } 2 = x[n] \text{ XOR } s[n]$

Descrambling of image

$x_d[n]=(x_s[n]+s[n])\text{mod } 2 = x_s[n] \text{ XOR } s[n]$

Autocorrelation

The auto-correlation Rxx[k] of a discrete-time signal x[n] is the cross-correlation with itself:

Rxx[k]=x[n]*x[n] where * is the convolution operator

Bitwise operations in C

There are six bitwise operations in c:

1. AND in c " & "
2. OR in c " | "
3. LEFT SHIFT in c " << "
4. RIGHT SHIFT in c " >> "
5. XOR in c " ^ "
6. NOT in c " ~ "

Using Bitwise Operation in Program gives advantage of memory saving and saving cost of using expensive units like adder. Generating PN sequence using bitwise operator saves memory and if efficient as in STM32 board has very limited memory.

METHOD

MATLAB CODE

In MATLAB 3d image is read through imread function and this 3d image is converted into 2d image using the rgb2gray and this grayscale image is then converted to binary image using the imbinarize function in MATLAB this binary image contains matrix of 1's and 0's which represent value of pixel. Using comm.PNSequence function in MATLAB PN sequence is generated of size which is equivalent to column length. For scrambling of binary image XOR is taken of binary image matrix and transpose of PN sequence to get scrambled image. To get back the original binary image again XOR is done between scrambled image and transpose of PN sequence. Autocorrelation of PN sequence is done from xcorr2() function in MATLAB and autocorrelation of binary image and cross correlation of by convolving using conv2() function.

C CODE

In c we are scrambling the sine wave which we generated from previous lab. We know the 12-bit samples of sine wave which we are sending to STM32 board. We are generating 12-bit PN sequence using bitwise operation and converting that 12-bit PN sequence to decimal and doing bitwise XOR with every sample which result in new scrambled set of samples to generate scrambled sine wave and again doing bitwise XOR of scrambled set of samples with decimal of 12bit PN sequence we get back our required sine wave samples.

Time Difference Calculation

We are also calculating the time difference between the generation of PN sequence using the bitwise operation and normal generation of PN sequence without bitwise operation.

STM32

In STM32 we are setting both the DAC channel that is Channel 1 whose output is at PIN PA4 and Channel 2 whose output is at PIN PA5 and setting the timer 2 and timer 4 in DMA settings and selecting circular mode. In Timer we are setting both Timer 2 and Timer 4 for DMA.

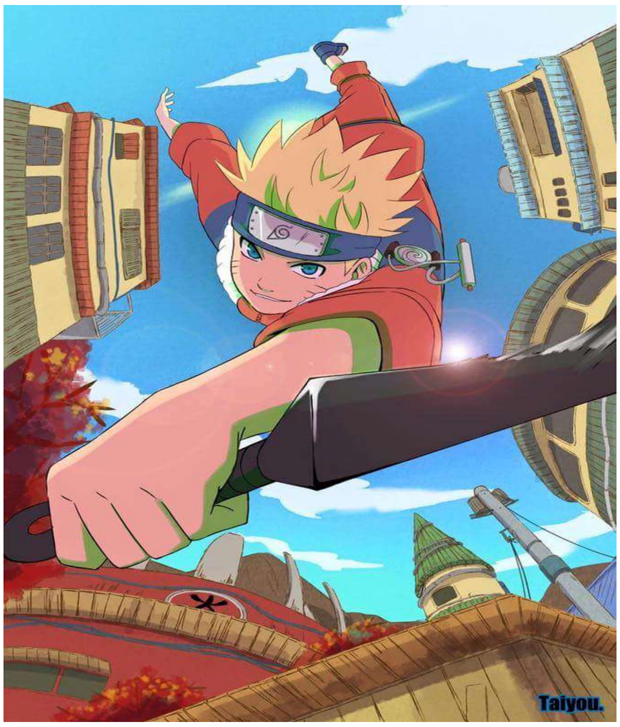
RESULT

MATLAB CODE

```
clc;
clear;
%reading image since image is 3D show its dimension is a * b * 3
threedimage = imread('1234.jpg');
figure
imshow(threedimage)
%converting 3D image to 2D image which contains value of pixel from 0 to
%255 and its dimension is a * b
grayimage=rgb2gray(threedimage);
%imbinarize converts 2D image to binary image in which each pixel contain
%values from 0 or 1
binaryimage =imbinarize(grayimage);
%STORE dimension of image to variable q(rows) and w(columns)
[q,w]=size(binaryimage);
tic %stopwatch on
%generated a pn sequence for polynomial of 3 degree x^3+x+1 with inital
%conditon as 1 0 1 and w is no. of pn sequence that is equal to column length
H = comm.PNSequence('Polynomial',[20 17 0],'InitialConditions',[1 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 1 1 0 1],'SamplesPerFrame',w*q);
k=H(); %storing pn sequence values in k
time1=toc;  %stopwatch off
tic %stopwatch on
a1 = uint8(9);
 i1 = 1;
 %calculating pn sequence using bitwise operation and calculating time for
 %above inbuilt function and bitwise operation
   while i1<w*q +1
       x = (bitand(bitxor(bitshift(a1,-1),a1),1));
       p(i1)=x;
       a1 = bitor((bitshift(a1,-1)),(bitshift((bitand(bitxor(bitshift(a1,-1),a1),1)),3)));
       i1=i1+1;
   end
time2=toc;   %stopwatch off
h=reshape(k,[960,676]);
%scrambling of binary image with pn sequence
scramble=xor(binaryimage,h);
```

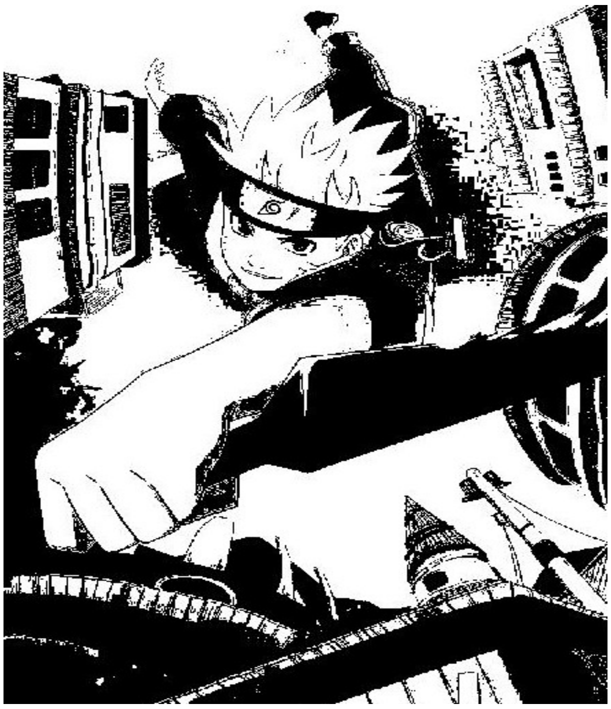
```
%recovering image from scrambled image
unscramble=xor(scramble,h);
%autocorrelation of pn sequence
acfofpnsequence=xcorr2(h);
%autocorrelation of binary image
auto_corr_of_binaryimage= conv2(binaryimage, rot90(binaryimage,2));
%cross-correlation of image and scramble image
cross_corr_of_image_and_scramble= conv2(binaryimage, rot90(scramble,2));
%cross-correlation of image and pn sequence
cross_corr_of_image_and_scramble1= conv2(binaryimage, rot90(h,2));
%cross-correlation of unscramble and scramble image
cross_corr_of_image_and_scramble2= conv2(scramble, rot90(unscramble,2));
figure
imshow(grayimage)
figure
imshow(binaryimage)
figure
imshow(scramble)
figure
imshow(unscramble)
figure
mesh(auto_corr_of_binaryimage)
figure
mesh(cross_corr_of_image_and_scramble)
figure
mesh(cross_corr_of_image_and_scramble1)
figure
mesh(cross_corr_of_image_and_scramble2)
```

MATLAB RESULTS

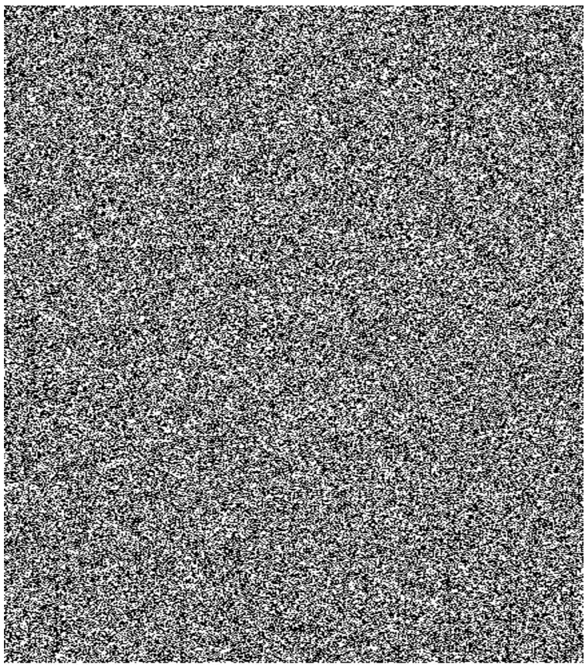


Original image(960x676x3)

Gray scale image(960x676)

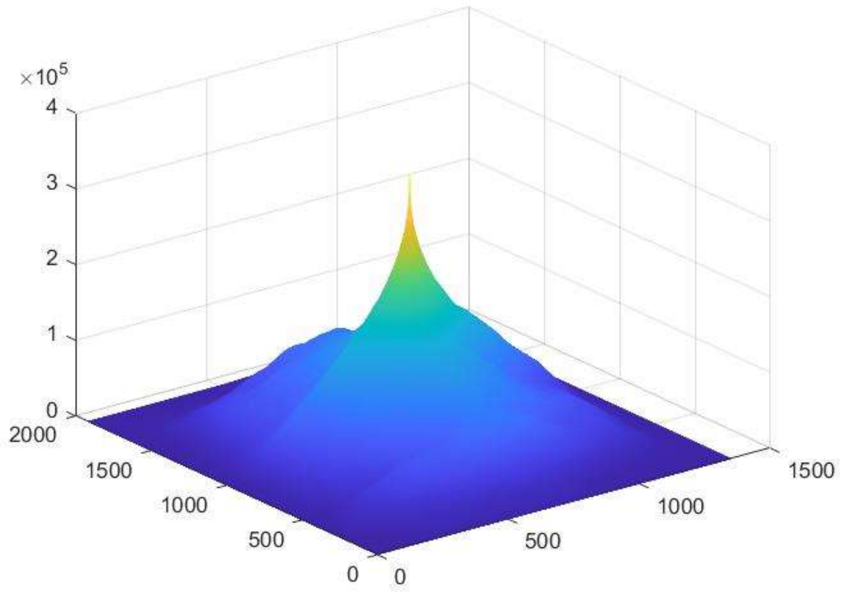


Binary Image(960x676)

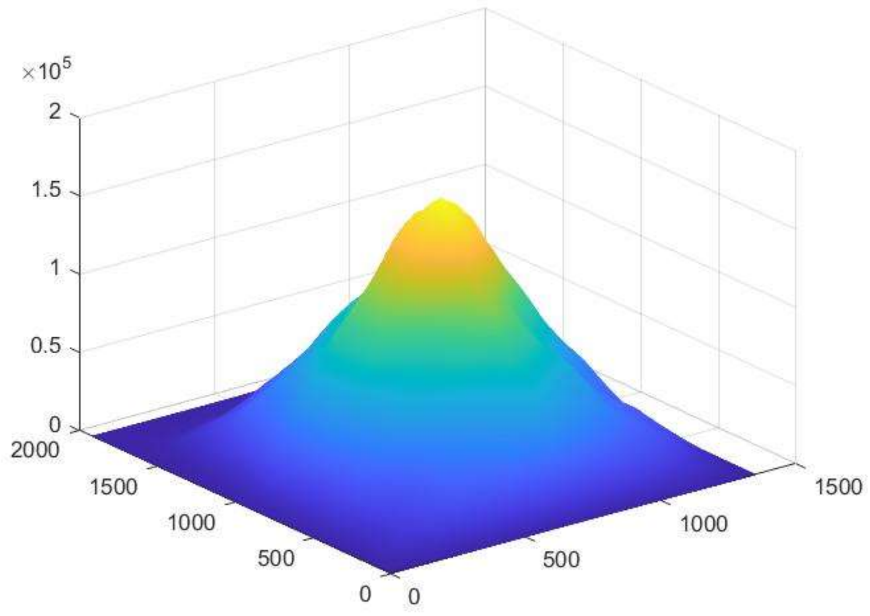


Scramble Image(960x676)

3D GRAPH of Autocorrelation Of Binary Image and Cross correlation Of Binary Image and Scramble Image



3D Graph of Autocorrelation of Binary Image

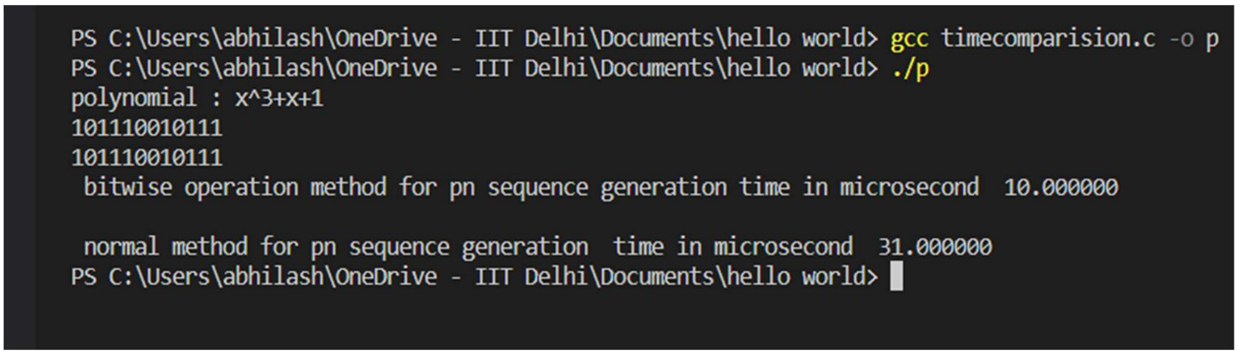


3D Graph of Cross correlation Of Binary Image and Scramble Image

C CODE FOR PN SEQUENCE GENERATION AND CALCULATING TIME DIFFERENCE

OUTPUT

```
#include <stdio.h>
#include <time.h>
#include <sys/time.h>
#include <unistd.h>
int fun_xor(int x, int y)
{
    if (x == 0 & y == 0)
    {
        return 0;
    }
    if (x == 0 & y == 1)
    {
        return 1;
    }
    if (x == 1 & y == 0)
    {
        return 1;
    }
    if (x == 1 & y == 1)
    {
        return 0;
    }
}
int main()
{
    struct timeval start1, end1, start2, end2;
    int initial = 0;
    int initial1 = 0;
    int initial2 = 1;
    int x, b[1000];
    int i = 0;
    char a = 1;
    int i1 = 0;
    int x1[1000];
    gettimeofday(&start1, NULL);
    while (i1 < 1000)
    {
        x1[i1] = ((a >> 1) ^ a) & 1;
        a = (a >> 1) | ((((a >> 1) ^ a) & 1) << 2);
        i1++;
    }
    gettimeofday(&end1, NULL);

    for (int j = 0; j < 12; j++)
        printf("%d", x1[j]);

    printf("\n");
    gettimeofday(&start2, NULL);
    while (i < 1000)
    {
        x = fun_xor(initial1, initial2);
        initial2 = initial1;
        initial1 = initial;
        initial = x;
        b[i] = x;
        i++;
    }
    gettimeofday(&end2, NULL);
    for (int j = 0; j < 12; j++)
        printf("%d", b[j]);
    float time1;
    time1 = (float)(((end1.tv_sec * 1000000 + end1.tv_usec) - (start1.tv_sec * 1000000 + start1.tv_usec)));
    printf("\n bitwise operation method for pn sequence generation time in microsecond  %f\n", time1);
    float time2;
    time2 = (float)(((end2.tv_sec * 1000000 + end2.tv_usec) - (start2.tv_sec * 1000000 + start2.tv_usec)));
    printf("\n normal method for pn sequence generation  time in microsecond  %f\n", time2);

    return 0;
}
```

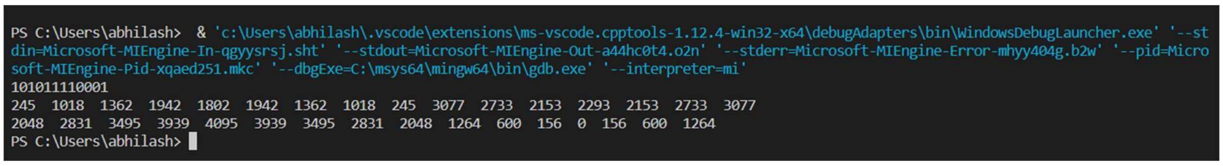


```
PS C:\Users\abhilash\OneDrive - IIT Delhi\Documents\hello world> gcc timecomparision.c -o p
PS C:\Users\abhilash\OneDrive - IIT Delhi\Documents\hello world> ./p
polynomial : x^3+x+1
101110010111
101110010111
 bitwise operation method for pn sequence generation time in microsecond  10.000000

 normal method for pn sequence generation  time in microsecond  31.000000
PS C:\Users\abhilash\OneDrive - IIT Delhi\Documents\hello world> 
```

C CODE FOR SCRAMBLING

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

int main()
{

    char initalcondition = 9;
    int i = 0;
    int arraytostorepn[12];
    int array[16] = {2048, 2831, 3495, 3939, 4095, 3939, 3495, 2831, 2048, 1264,
600, 156, 0, 156, 600, 1264};
    int array1[16], array2[16];

    while (i < 12)
    {
        arraytostorepn[i] = ((initalcondition >> 1) ^ initalcondition) & 1;
        initalcondition = (initalcondition >> 1) | ((((initalcondition >> 1) ^
initalcondition) & 1) << 3);
        i++;
    }

    for (int j = 0; j < 12; j++)
        printf("%d", arraytostorepn[j]);

    printf("\n");
    int n = 0;
    for (int j = 0; j < 12; j++)
    {
        n = n + (arraytostorepn[j] * pow(2, j));
    }
    for (int j = 0; j < 16; j++)
    {
        array1[j] = array[j] ^ n;
    }
    for (int j = 0; j < 16; j++)
    {
        printf("%d  ", array1[j]);
    }
    printf("\n");

    for (int j = 0; j < 16; j++)
    {
        array2[j] = array1[j] ^ n;
    }
    for (int j = 0; j < 16; j++)
    {
        printf("%d  ", array2[j]);
    }
    return 0;

}
```

OUTPUT



```
PS C:\Users\abhilash>  & 'c:\Users\abhilash\.vscode\extensions\ms-vscode.cpptools-1.12.4-win32-x64\debugAdapters\bin\WindowsDebugLauncher.exe' '--st
din=Microsoft-MIEngine-In-qgyysrsj.sht' '--stdout=Microsoft-MIEngine-Out-a44hc0t4.o2n' '--stderr=Microsoft-MIEngine-Error-mhyy404g.b2w' '--pid=Micro
soft-MIEngine-Pid-xqaed251.mkc' '--dbgExe=C:\msys64\mingw64\bin\gdb.exe' '--interpreter=mi'
101011110001
245  1018  1362  1942  1802  1942  1362  1018  245  3077  2733  2153  2293  2153  2733  3077
2048  2831  3495  3939  4095  3939  3495  2831  2048  1264  600  156  0  156  600  1264
PS C:\Users\abhilash> 
```

STM32 CODE

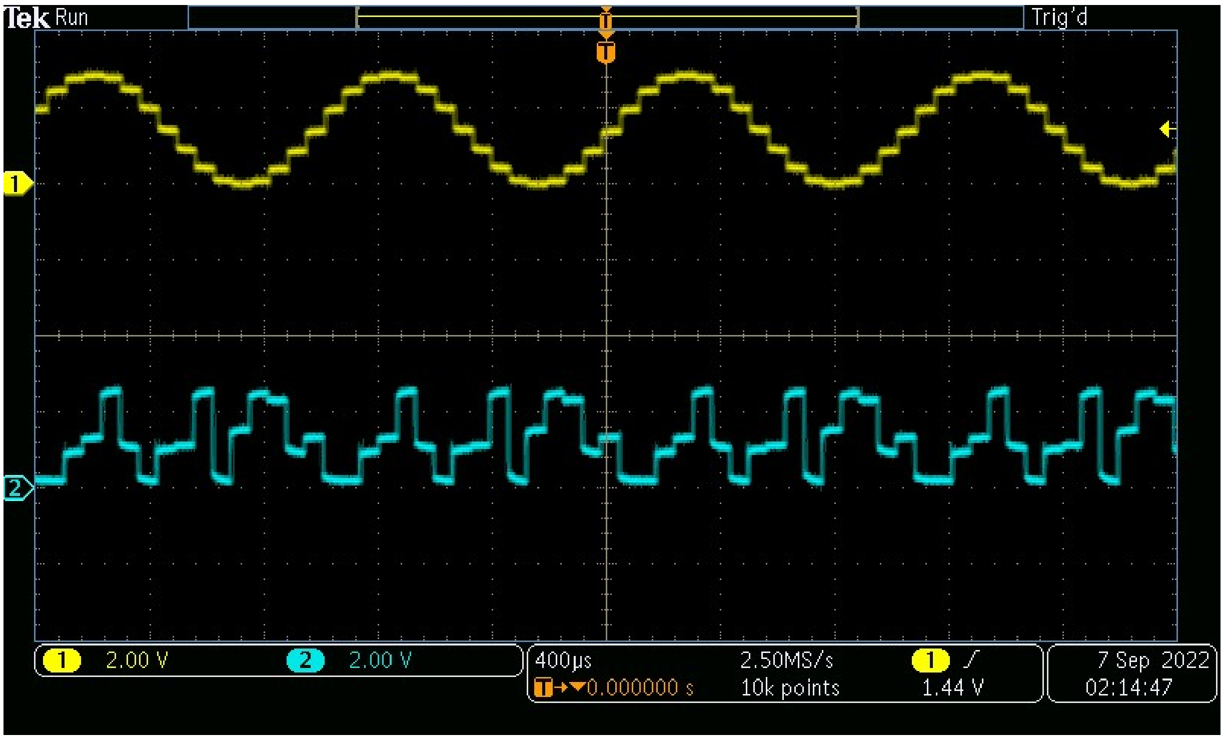
```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (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 "math.h"
/* 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 ---------------------------------------------------------*/
DAC_HandleTypeDef hdac;
DMA_HandleTypeDef hdma_dac1;
DMA_HandleTypeDef hdma_dac2;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim4;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* 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_TIM2_Init(void);
static void MX_TIM4_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

int main(void)
{
  /* USER CODE BEGIN 1 */

		char a = 15;
		uint32_t i = 0;
		uint32_t a1[12];
		uint32_t array1[16] = {2048, 2831, 3495, 3939, 4095, 3939, 3495, 2831,2048, 1264, 600, 156, 0, 156, 600, 
1264};
		uint32_t array2[16];
		uint32_t array3[16];

		  while(i<12)
		  {
		    a1[i] = ((a>>1)^a)&1;
		    a = (a>>1) | ((((a>>1)^a)&1)<<3);
		    i++;
		  }

		  uint32_t n = 0;
		  for(uint32_t j=0;j<12;j++)
		    n = n+(a1[j]*pow(2,j));
```

```
        for(uint32_t j=0;j<16;j++)
            array2[j] = array1[j]^n;




        for(int j=0;j<16;j++)
            array3[j] = array2[j]^n;



  /* 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_TIM2_Init();
  MX_TIM4_Init();
  /* USER CODE BEGIN 2 */

  HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, (uint32_t*)array2, 16, DAC_ALIGN_12B_R);
  HAL_TIM_Base_Start(&htim2);
  HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_2, (uint32_t*)array3, 16, DAC_ALIGN_12B_R);
   HAL_TIM_Base_Start(&htim4);
  /* 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
  */
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_SCALE3);

  /** 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 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();
  }

  /** DAC channel OUT2 config
  */
  sConfig.DAC_Trigger = DAC_TRIGGER_T4_TRGO;
  if (HAL_DAC_ConfigChannel(&hdac, &sConfig, DAC_CHANNEL_2) != 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 = 5249;
  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;
  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 */

}

/**
  * @brief TIM4 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM4_Init(void)
{

  /* USER CODE BEGIN TIM4_Init 0 */

  /* USER CODE END TIM4_Init 0 */

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

  /* USER CODE BEGIN TIM4_Init 1 */

  /* USER CODE END TIM4_Init 1 */
  htim4.Instance = TIM4;
  htim4.Init.Prescaler = 0;
  htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim4.Init.Period = 5249;
  htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
  {
    Error_Handler();
  }
```

```c
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_UPDATE;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM4_Init 2 */

  /* USER CODE END TIM4_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);
  /* DMA1_Stream6_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Stream6_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Stream6_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 */
```

RESULTS



DISCUSSION

- PN sequence is pseudo random sequence which is periodic sequence whose autocorrelation is approximately equal to that of white noise
- Converting RGB color image to grayscale image and from grayscale image to binary image is irreversible process that we cannot do vice versa.
- Scrambling image can be recovered which means this method can be use for encryption of data

- Time calculated for generation of PN sequence in MATLAB by both inbuilt function and using the bitwise operator of MATLAB differs and bitwise operation is faster and efficient
- Auto correlation of binary image results in graph which show bel shaped 3d curve and it is sharper at peak means the same image
- Cross correlation of binary image and scramble image is normal bell shaped showing that some correlation exists between the image and image is not same
- Time calculated using the bitwise operators is taking low time and also efficient in space complexity than the one which is calculating pn sequence by calling function and using the if else conditions.
- Stm32 board generation of both scrambled and descrambled sine wave is generated in same oscilloscope at 2 different channels at same time using both output pin of dac in stm32f429zit6 that is pa4 and pa5 using two timers that is timer2 and timer3.