OBJECTIVE 1- **Simple DSP in Matlab**

Theory

Digital to Analog conversion is the process to convert digital code to voltage or current which is proportional to its digital value. Resolution of a Digital to Analog converter is the smallest change that can occur in the analog output as a result of a change in the digital input.

Y= Y + 1;

Y = Y*((2^RES-1)-2*OFFSET)/(2+OFFSET);

Y = round(Y);

We are shifting the cosine signal by adding one it transforms from the range [-1 1] to [0 2]. Multiply by 2^12 to get a value in the range [0 4096]. Divide by 2 to get [0 2048]. Which is suitable for transmission over a digital interface using unsigned bits. If 1 were not added to y then we would have to transmit values in the range [-2048 2048], and then there would be problem of the representations for negative values. Offset is zero so it is not affecting

Method

In matlab we are creating a vector T (time) of range 0 to 1 having interval of 0.001 so total number of time interval is 1000 in vector array. We are generating cosine signal and calculating its value for 1000 interval from 0 to 1 in interval. we are plotting value of cosine for both discrete and continuous by using plot and stem function after down sampling both time and cosine values.

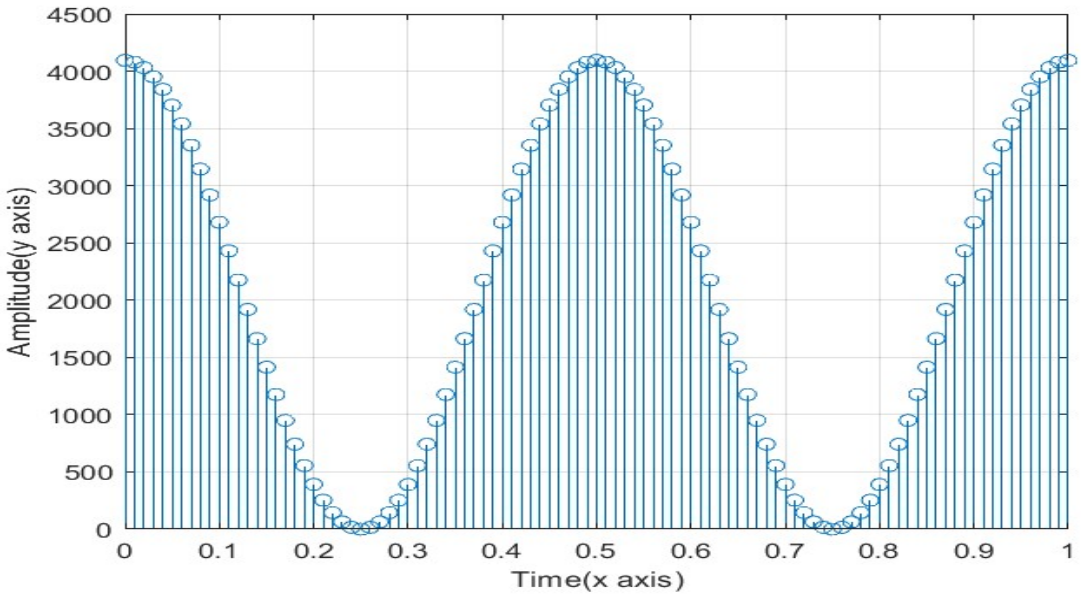
Result

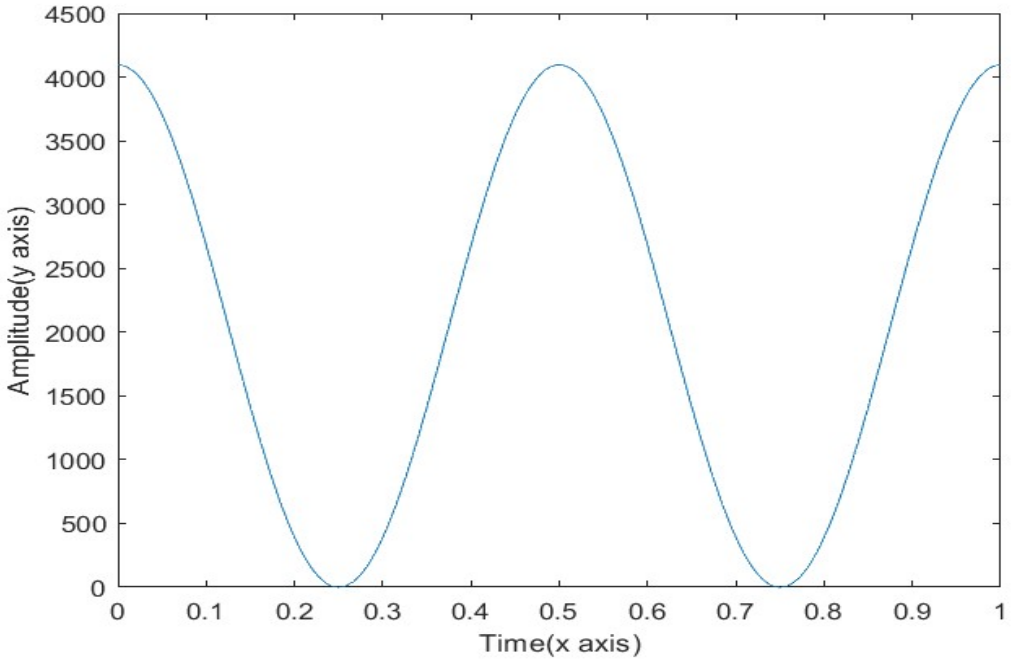
Code in matlab

```
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 = 0:0.001:1;
```

```
 Y4 = cos(4*pi*T);
 Y4= Y4 + 1;
 Y4 = Y4*((2^RES-1)-2*OFFSET)/(2+OFFSET);
 Y4 = round(Y4);
 y1=downsample(T,10);
 x1=downsample(Y4,10);
 xlabel("Time(x axis)");
 ylabel("Amplitude(y axis)");
  figure(1)
  plot(T, Y4);
  xlabel("Time(x axis)");
  ylabel("Amplitude(y axis)");
  figure(2)
  stem(y1,x1);
  xlabel("Time(x axis)");
  ylabel("Amplitude(y axis)");
  grid
  %--------------[ Print The Sample Points ]---------------
  fprintf('%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d,
%d, %d, %d, \n', Y4);
 fprintf("\n");
```

Output





Discussion

we are generating sine signal both in discrete and continuous time and getting familiar with with matlab functions like plot ,downsample,stem ,xlabel,ylabel ,title,fprint etc

OBJECTIVE 2- **Simple Waveform with STM32 using Lookup Table**

Theory

Stm32 is 32-bit microcontroller designed by ST Microelectronics. We are using STM32F429ZIT6 MPU. It is very good in system performance for code execution, data transfer and data processing. STM32F4 series offers from 16Mhz to 180Mhz.

The lookup table is an array of unsigned integer values that represents the sample points of a specific waveform for a complete cycle (from 0 to 2π). The length of the lookup table is denoted as Ns or the number of sample points per complete cycle. The more sample points per cycle, the better the output waveform.

Method

We are converting continuous signal x(t)=u(t)sin(2πfot) to discrete signal x[n]=u[n]sin(2π(fo/fs) n) where fo and fs are given as 1Khz and 16Khz and sending this discrete signal to DAC in STM32F429ZIT6 whose output is associated with the pin PA4 in board where we are taking Analog output to Digital Oscilloscope to measure the frequency.

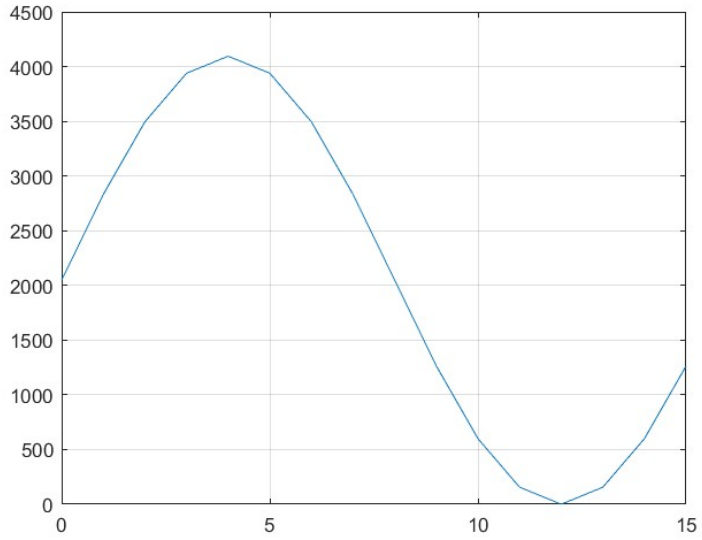
Step are

1. **Generating lookup table from matlab for discrete signal**
   To generate lookup table the code in matlab is

```
clear; clc;    % Clear The Previous Points
Ns     = 16;   % Set The Number of Sample Points
RES    = 12;   % Set The DAC Resolution
OFFSET = 0;    % Set An Offset Value For The DAC Output

fs=16000;
fo=1000;
n = 0:1:15;
```

```
Y = sin(2*pi*fo*n/fs);                  % Sin
Y = Y + 1;
Y = Y*((2^RES-1)-2*OFFSET)/(2+OFFSET);
Y = round(Y);
plot(n, Y)
stem(n,Y);
grid

fprintf('%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d,
%d, %d, %d, %d, \n', Y);
```

Signal displayed in matlab



Lookup Table data points are

2048, 2831, 3495, 3939, 4095, 3939, 3495, 2831, 2048, 1264, 600, 156, 0, 156, 600, 1264

2. Calculating the value of ARR

Ns=16 (number of samples in lookup table)

Since number of sample are 16

Trigger Frequency=Fclk\(PSC+1)(ARR+1)

Fclk is Frequency of Clock used by timer module

PSC is Prescaler

ARR is value of Auto Reload Register

Output Wave Frequency=Trigger Frequency/Ns

We want output frequency 1khz and have Ns =16

We get Trigger Frequency=16Khz

We set Fclk=89Mhz & PSC=0

And calculated the value of ARR which is equal to 5561

3. **Open Cube MX create a new project**
4. **Select stm32f429zit6 MPU and double click it**
5. **Go to clock configuration and set Fclk to 89Mhz and clear the pins**
6. **Enable DAC in analog set it OUT1(which enable PA4 pin as DAC output)**
7. **Set DMA settings by adding it and setting priority high with circular mode and data width word**
8. **Go to Timer select TIM2**
9. **Select clock source as internal clock**
10. **In counter clock setting set psc =0**
11. **Set ARR=5561**
12. **Generate Code**
13. **Open the Project in Cube IDE**
14. **Open main .c and write the lookup table and run the debugger**
15. **Calculate the frequency of generated signal in DSO**

Result

Code for main.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"

/* 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;

TIM_HandleTypeDef htim2;
```

```c
/* 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);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
#define NS 16

uint32_t Wave_LUT[NS] = {
              2048, 2831, 3495, 3939, 4095, 3939, 3495, 2831, 2048, 1264, 600, 156, 0,
156, 600,
              1264

};
/* 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();
```

```c
  MX_DMA_Init();
  MX_DAC_Init();
  MX_TIM2_Init();
  /* USER CODE BEGIN 2 */

  HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, (uint32_t*)Wave_LUT, 16, 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
  */
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 = 89;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 4;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Activate the Over-Drive mode
  */
  if (HAL_PWREx_EnableOverDrive() != HAL_OK)
```

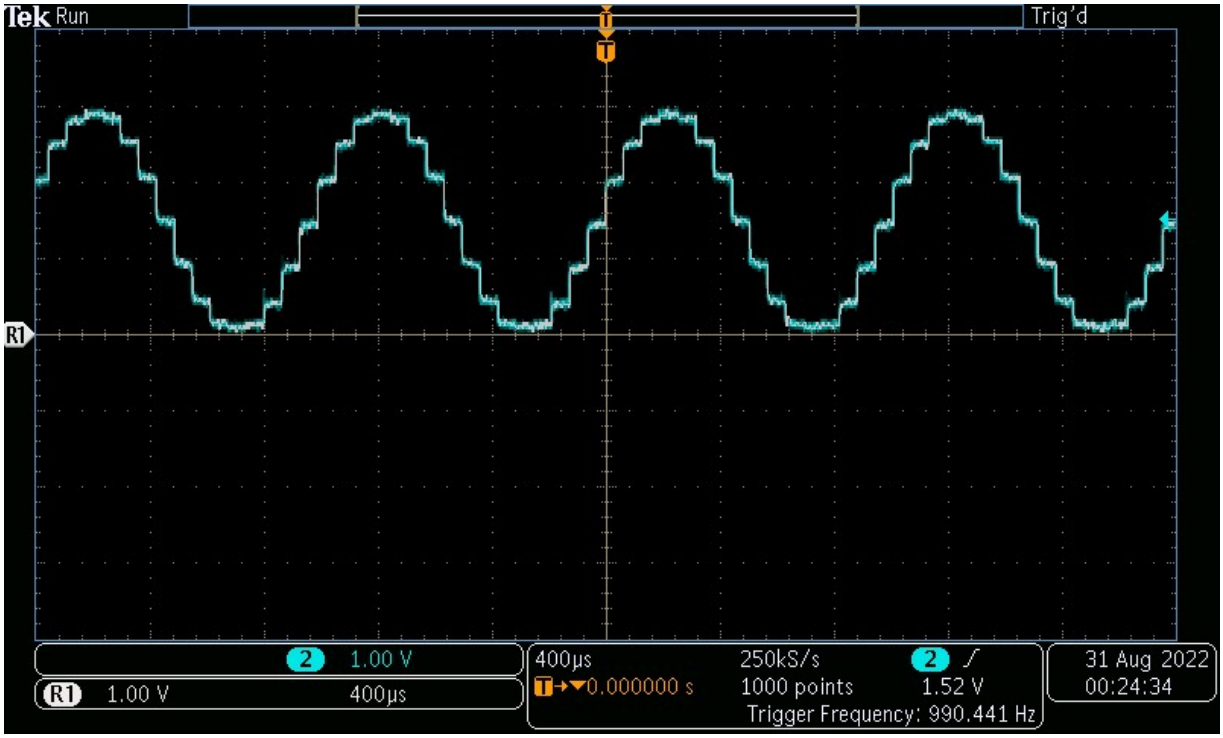
```
  {
    Error_Handler();
  }

  /** 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();
  }
  /* USER CODE BEGIN DAC_Init 2 */
```

```c
  /* 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 = 5561;
  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 */

}

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

```c
  /* 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 */
```

DISPLAY IN OSCILLOSCOPE



Here, We Can see that the frequency we are getting in Digital Oscilloscope is 990.441 Hz

Discussion

Here we are learning that there is some error in the frequency approximately about 10Hz as the frequency of signal is 1khz and we are getting 990.441 Hz which we get from measuring from oscilloscope. This can happen maybe because of Number of samples are less.

OBJECTIVE 3- **Plotting different functions in oscilloscope using ARM microcontroller**

Theory

In MATLAB we are generating signal using

- Sine signal by using sin() in matlab
- Periodic Sinc Function using diric() in matlab
- Sawtooth signal Using the sawtooth() in matlab
- Triangular Signal Using the Sawtooth () in matlab

Here, we are getting all the sample points of the all the signal.

We are taking 128 sample points from matlab and using it in lookup table to generate signal in DSO via stm32 microcontroller.

Y= Y + 1;

Y = Y*((2^RES-1)-2*OFFSET)/(2+OFFSET);

Y = round(Y);

We are shifting the cosine signal by adding one it transforms from the range [-1 1] to [0 2]. Multiply by 2^12 to get a value in the range [0 4096]. Divide by 2 to get [0 2048]. Which is suitable for transmission over a digital interface using unsigned bits. If 1 were not added to y then we would have to transmit values in the range [-2048 2048], and then there would be problem of the representations for negative values. Offset is zero so it is not affecting

Method

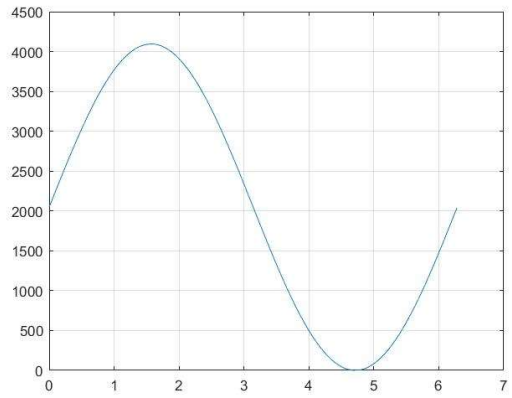
The code in MATLAB which we using to generate sample points for lookup table

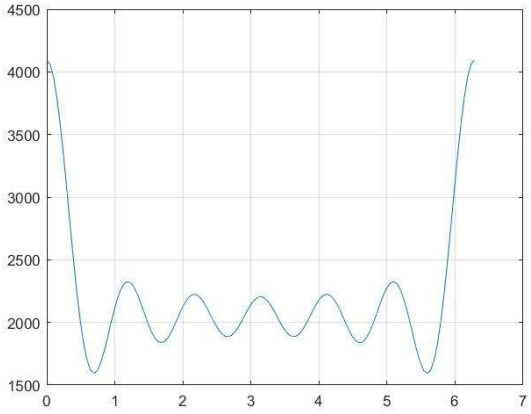
```
clear; clc;
```

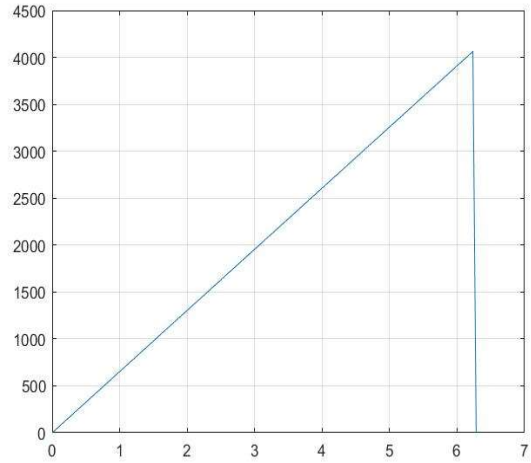
```
Ns     = 128;
RES    = 12;
OFFSET = 0;
T = 0:((2*pi/(Ns-1))):(2*pi);
Y1 = sin(T);              % Sin
Y2 = diric(T, 13);     % Periodic Sinc
Y3 = sawtooth(T)         % Sawtooth
Y4 = sawtooth(T, 0.5);  % Triangular

Y1 = Y1 + 1;
Y1 = Y1*((2^RES-1)-2*OFFSET)/(2+OFFSET);
Y1 = round(Y1);

Y2 = Y2 + 1;
Y2 = Y2*((2^RES-1)-2*OFFSET)/(2+OFFSET);
Y2 = round(Y2);

Y3 = Y3 + 1;
Y3 = Y3*((2^RES-1)-2*OFFSET)/(2+OFFSET);
Y3 = round(Y3);

Y4 = Y4 + 1;
Y4 = Y4*((2^RES-1)-2*OFFSET)/(2+OFFSET);
Y4= round(Y4);
Figure(1)
plot(T, Y1);
grid
Figure(2)
plot(T, Y2);
grid
Figure(3)
plot(T, Y3);
grid
Figure(4)
plot(T, Y4);
grid


fprintf('%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d,
\n', Y1);


fprintf('%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d,
\n', Y2);


fprintf('%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d,
\n', Y3);
```

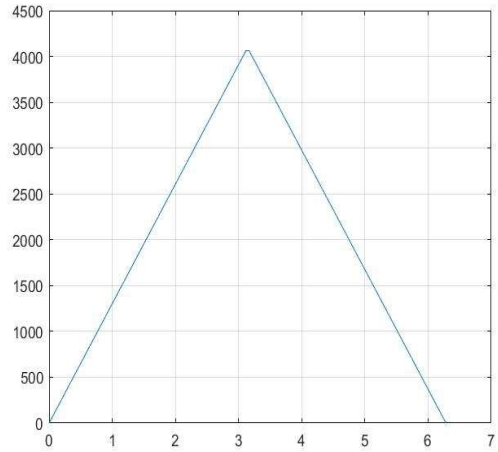
```
fprintf('%d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d, %d,
\n', Y4);
```

Result









lookup table for all signal is generated in command window which can be used for stm32 for generating signal in DSO

- sample points for sine signal

2048, 2149, 2250, 2350, 2450, 2549, 2646, 2742, 2837, 2929, 3020, 3108, 3193, 3275, 3355,
3431, 3504, 3574, 3639, 3701, 3759, 3812, 3861, 3906, 3946, 3982, 4013, 4039, 4060, 4076,
4087, 4094, 4095, 4091, 4082, 4069, 4050, 4026, 3998, 3965, 3927, 3884, 3837, 3786, 3730,
3671, 3607, 3539, 3468, 3394, 3316, 3235, 3151, 3064, 2975, 2883, 2790, 2695, 2598, 2500,
2400, 2300, 2199, 2098, 1997, 1896, 1795, 1695, 1595, 1497, 1400, 1305, 1212, 1120, 1031,
944, 860, 779, 701, 627, 556, 488, 424, 365, 309, 258, 211, 168, 130, 97,
69, 45, 26, 13, 4, 0, 1, 8, 19, 35, 56, 82, 113, 149, 189,
234, 283, 336, 394, 456, 521, 591, 664, 740, 820, 902, 987, 1075, 1166, 1258,
1353, 1449, 1546, 1645, 1745, 1845, 1946, 2047,

- sample points for sinc signal

4095, 4060, 3958, 3793, 3578, 3323, 3045, 2758, 2479, 2222, 2000, 1822, 1694, 1618, 1594,
1616, 1677, 1766, 1874, 1988, 2097, 2191, 2264, 2310, 2326, 2315, 2277, 2219, 2148, 2071,
1996, 1930, 1879, 1847, 1837, 1848, 1878, 1924, 1981, 2043, 2103, 2155, 2195, 2219, 2225,
2213, 2185, 2143, 2093, 2039, 1987, 1942, 1908, 1889, 1886, 1900, 1928, 1968, 2016, 2067,
2116, 2157, 2187, 2203, 2203, 2187, 2157, 2116, 2067, 2016, 1968, 1928, 1900, 1886, 1889,
1908, 1942, 1987, 2039, 2093, 2143, 2185, 2213, 2225, 2219, 2195, 2155, 2103, 2043, 1981,
1924, 1878, 1848, 1837, 1847, 1879, 1930, 1996, 2071, 2148, 2219, 2277, 2315, 2326, 2310,
2264, 2191, 2097, 1988, 1874, 1766, 1677, 1616, 1594, 1618, 1694, 1822, 2000, 2222, 2479,
2758, 3045, 3323, 3578, 3793, 3958, 4060, 4095

- sample points for sawtooth signal

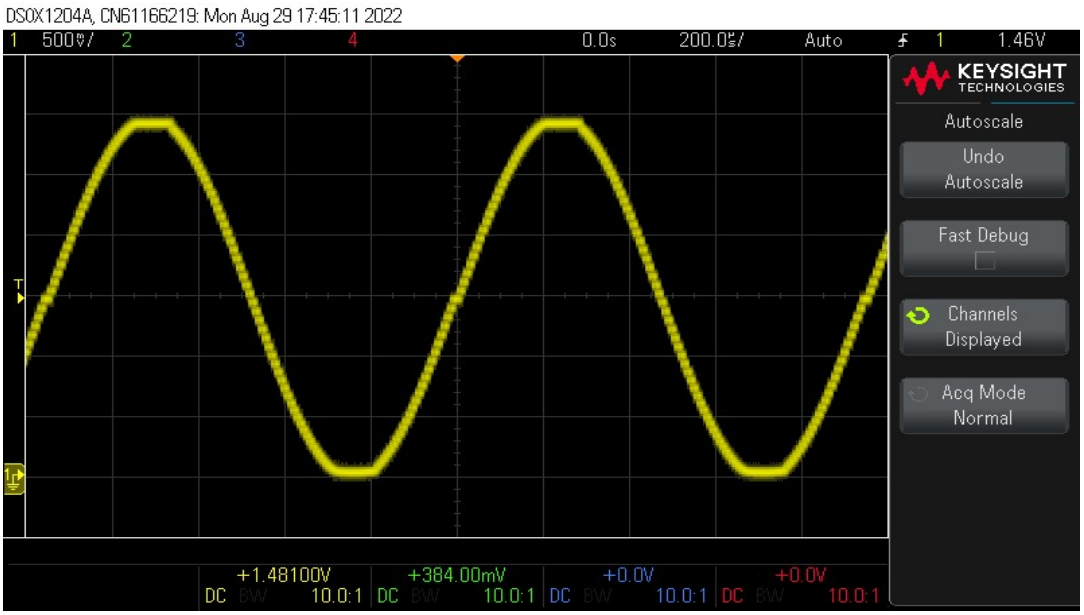
0, 32, 64, 97, 129, 161, 193, 226, 258, 290, 322, 355, 387, 419, 451,
484, 516, 548, 580, 613, 645, 677, 709, 742, 774, 806, 838, 871, 903, 935,
967, 1000, 1032, 1064, 1096, 1129, 1161, 1193, 1225, 1258, 1290, 1322, 1354, 1386, 1419,
1451, 1483, 1515, 1548, 1580, 1612, 1644, 1677, 1709, 1741, 1773, 1806, 1838, 1870, 1902,

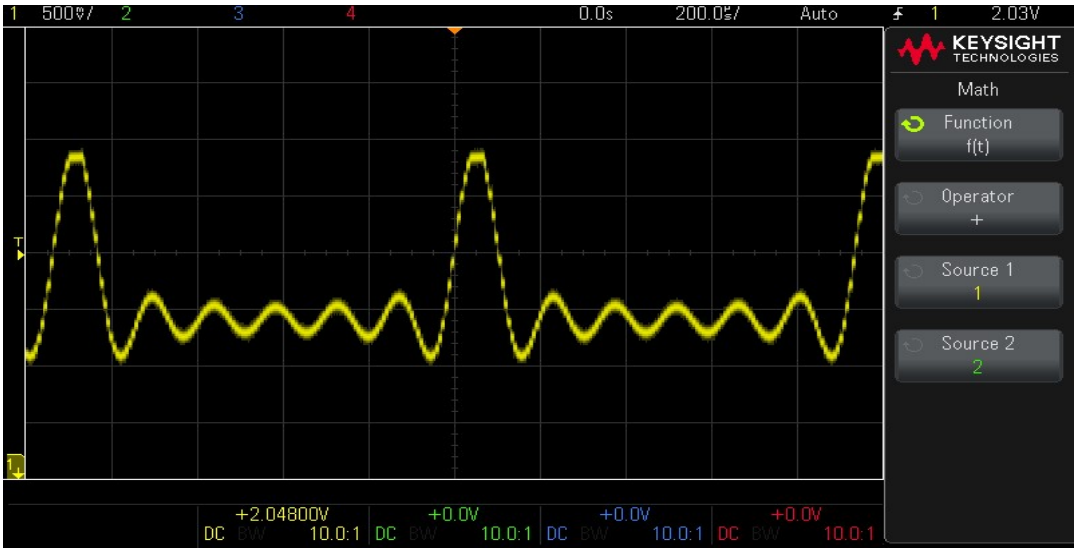
1935, 1967, 1999, 2031, 2064, 2096, 2128, 2160, 2193, 2225, 2257, 2289, 2322, 2354, 2386, 2418, 2451, 2483, 2515, 2547, 2580, 2612, 2644, 2676, 2709, 2741, 2773, 2805, 2837, 2870, 2902, 2934, 2966, 2999, 3031, 3063, 3095, 3128, 3160, 3192, 3224, 3257, 3289, 3321, 3353, 3386, 3418, 3450, 3482, 3515, 3547, 3579, 3611, 3644, 3676, 3708, 3740, 3773, 3805, 3837, 3869, 3902, 3934, 3966, 3998, 4031, 4063, 0

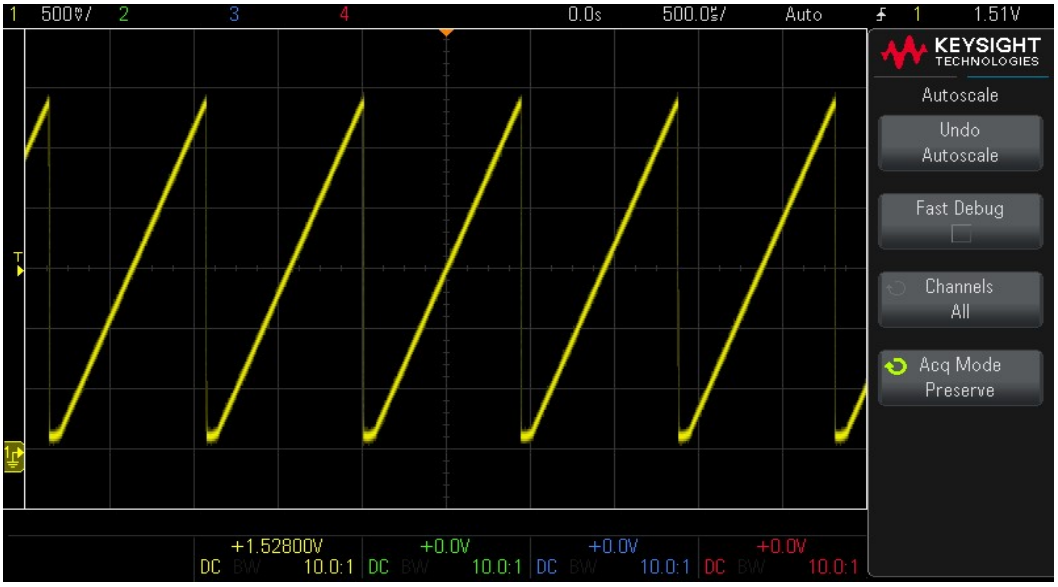
- sample points for triangular signal

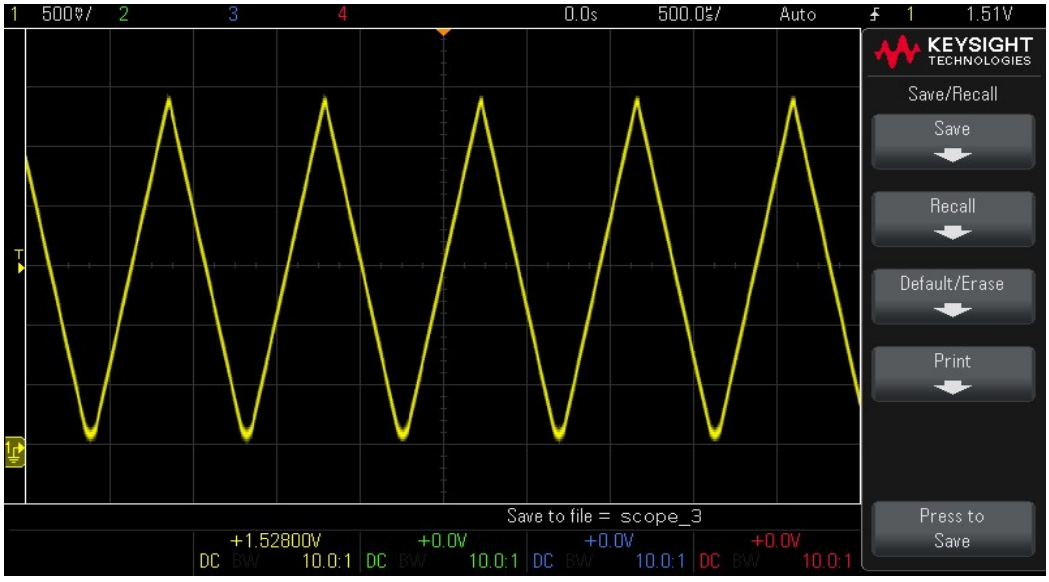
0, 64, 129, 193, 258, 322, 387, 451, 516, 580, 645, 709, 774, 838, 903,
967, 1032, 1096, 1161, 1225, 1290, 1354, 1419, 1483, 1548, 1612, 1677, 1741, 1806, 1870, 1935, 1999, 2064, 2128, 2193, 2257, 2322, 2386, 2451, 2515, 2580, 2644, 2709, 2773, 2837, 2902, 2966, 3031, 3095, 3160, 3224, 3289, 3353, 3418, 3482, 3547, 3611, 3676, 3740, 3805, 3869, 3934, 3998, 4063, 4063, 3998, 3934, 3869, 3805, 3740, 3676, 3611, 3547, 3482, 3418, 3353, 3289, 3224, 3160, 3095, 3031, 2966, 2902, 2837, 2773, 2709, 2644, 2580, 2515, 2451, 2386, 2322, 2257, 2193, 2128, 2064, 1999, 1935, 1870, 1806, 1741, 1677, 1612, 1548, 1483, 1419, 1354, 1290, 1225, 1161, 1096, 1032, 967, 903, 838, 774, 709, 645, 580, 516,
451, 387, 322, 258, 193, 129, 64, 0

In digital oscilloscope the signal we get are


DSOX1204A, CN61166219: Mon Aug 29 17:45:11 2022
1 500mV/ 2 3 4 0.0s 200.0ks/ Auto 1 1.46V
KEYSIGHT TECHNOLOGIES
Autoscale
Undo Autoscale
Fast Debug
Channels Displayed
Acq Mode Normal
+1.48100V +384.00mV +0.0V +0.0V
DC BW 10.0:1 DC BW 10.0:1 DC BW 10.0:1 DC BW 10.0:1


1 500mV/ 2 3 4 0.0s 200.0ks/ Auto 1 2.03V
KEYSIGHT TECHNOLOGIES
Math
Function f(t)
Operator +
Source 1 1
Source 2 2
+2.04800V +0.0V +0.0V +0.0V
DC BW 10.0:1 DC BW 10.0:1 DC BW 10.0:1 DC BW 10.0:1





Discussion

Here, we are creating 4 lookup table for 4 different signal having 128 sample points each. So we can see that the signal which are generating in DSO is very clear. So if number of sample point is more we will get more clear signal in oscilloscope.