TRƯỜNG ĐẠI HỌC BÁCH KHOA – ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH

KHOA ĐIỆN – ĐIỆN TỬ

CHUYÊN NGÀNH ĐIỀU KHIỂN VÀ TỰ ĐỘNG HÓA

.....o0o.....

BÁO CÁO BÀI TẬP LỚN

MÔN: HỆ THỐNG ĐIỀU KHIỂN NHÚNG

ĐỀ TÀI

MÁY PHÁT XUNG 2 KÊNH – XUNG VUÔNG, TAM GIÁC, SINE – ĐIỀU CHỈNH BIÊN ĐỘ, TẦN SỐ

GVHD: TS. NGUYỄN VĨNH HẢO

NHÓM THỰC HIỆN: NHÓM 2 – L01

STT	Họ tên	MSSV
1	Đinh Văn Khánh	1810223
2	Ngô Kiến Hoàng	1810713
3	Trần Quang Huy	1810184
4	Phạm Lê Nam	1810339

TP. Hồ Chí Minh, ngày 10 tháng 12 năm 2021

BẢNG PHÂN CÔNG CÔNG VIỆC CỦA NHÓM

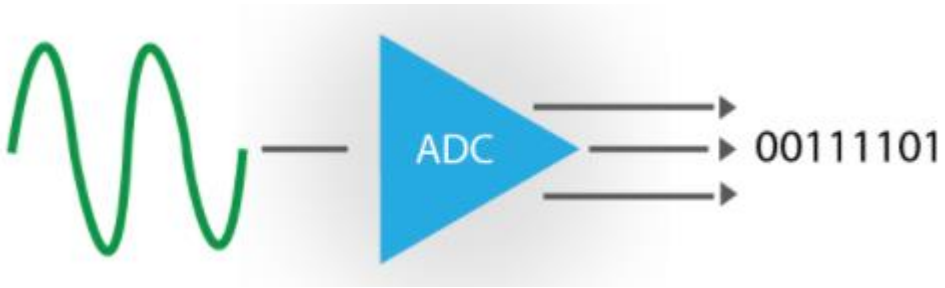
Sinh viên thực hiện	Nội dung thực hiện	Mức độ hoàn thành công việc	Phần trăm đóng góp vào bài tập lớn
Đinh Văn Khánh (1810223)	Lập trình vi điều khiển STM32F4 – DAC và truyền dữ liệu UART	100%	25%
Ngô Kiến Hoàng (1810713)	Lập trình vi điều khiển STM32F4 – ADC và nhận dữ liệu UART	100%	25%
Trần Quang Huy (1810184)	Thiết kế giao diện điều khiển và tổng hợp báo cáo	100%	25%
Phạm Lê Nam (1810339)	Thiết kế giao diện điều khiển và tổng hợp báo cáo	100%	25%

MỤC LỤC

A. CƠ SỞ LÝ THUYẾT

I. ADC

ADC (analog-to-digital converter) bộ chuyển đổi tín hiệu tương tự – số là thuật ngữ nói đến sự chuyển đổi một tín hiệu tương tự thành tín hiệu số để dùng trong các hệ thống số (digital systems) hay vi điều khiển.



Trong STM32F407VG có 3 bộ ADC chuyển đổi tín hiệu điện áp thành tín hiệu số với độ phân giải 12-bit. Mỗi ADC có khả năng tiếp nhận tín hiệu từ 16 kênh ngoài.

Để hiểu quá trình số hóa trong STM32 diễn ra như thế nào ta xem ví dụ sau. Giả sử ta cần đo điện áp tối thiểu là 0V và tối đa là 3.3V, trong STM32 sẽ chia 0->3.3V thành 4096 khoảng giá trị (từ 0 -> 4095, do 2^12=4096), giá trị đo được từ chân IO tương ứng với 0V sẽ là 0, tương ứng với 1.65V là 2047 và tương ứng 3.3V sẽ là 4095.

II. DAC

DAC (digital-to-analog converter) bộ chuyển đổi tín hiệu số – tương tự là thuật ngữ nói đến sự chuyển đổi một tín hiệu số thành tín hiệu tương tự để dùng trong các hệ thống số (digital systems) hay vi điều khiển.

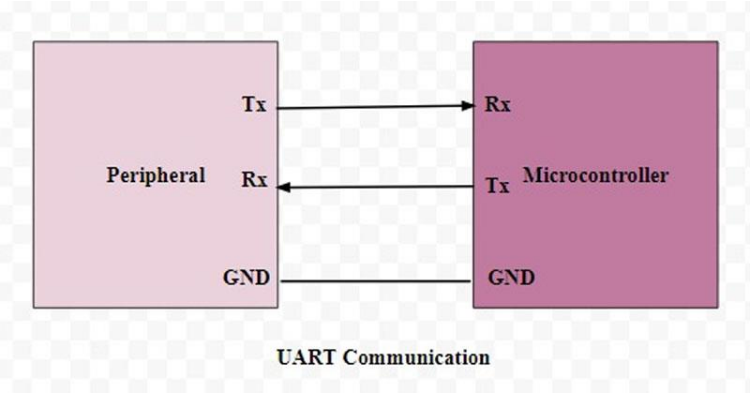
STM32F4 discovery có tích hợp sẵn 2 kênh DAC với độ phân giải 8-12 bits.

2 kênh có thể hoạt động độc lập hoặc đồng thời với nhau và có thể dùng trigger từ bên ngoài để điều khiển hoạt động. Mỗi kênh có thể sử dụng DMA riêng.

III. UART

UART có chức năng chính là truyền dữ liệu nối tiếp.

Trong UART, giao tiếp giữa hai thiết bị có thể được thực hiện theo hai phương thức là giao tiếp dữ liệu nối tiếp và giao tiếp dữ liệu song song.



UART Communication

Các thông số chính trong giao tiếp UART:

+ Baud rate: khoảng thời gian để 1 bit được truyền đi. Cài đặt giống nhau ở cả phần gửi và phần nhận.

+ Frame: khung truyền quy định về mỗi lần truyền bao nhiêu bit.

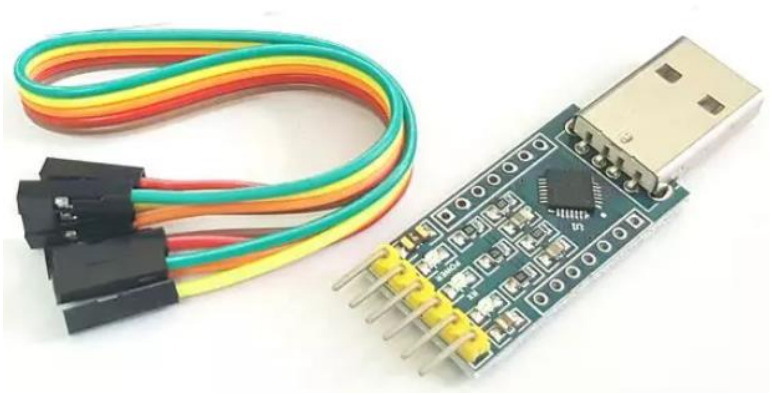
+ Start bit: là bit đầu tiền trong 1 Frame. Báo hiệu cho thiết bị nhận có một gói dữ liệu sắp được truyền đến. Đây là bit bắt buộc.

+ Data: dữ liệu cần truyền. Bit có trọng số nhỏ nhất LSB truyền trước sau đó đến bit MSB.

+ Parity bit: kiểm tra dữ liệu truyền có đúng hay không.

+ Stop bit: là 1 hoặc các bit cảnh báo cho thiết bị rằng các bit đã được gửi xong, Thiết bị nhận sẽ tiến hành kiểm tra khung truyền nhằm đảm bảo tính đúng đắn của dữ liệu. Đây là bit bắc buộc.

Để giao tiếp truyền – nhận UART giữa vi xử lý và máy tính, nhóm dùng mạch chuyển UART CP2102



Mô tả chân như sau:

+ TXD: chân truyền dữ liệu UART TTL (3.3VDC), dùng kết nối đến chân nhận RX của các module sử dụng mức tín hiệu TTL 3.3~5VDC.

+ RXD: chân nhận dữ liệu UART TTL (3.3VDC), dùng kết nối đến chân nhận TX của các module sử dụng mức tín hiệu TTL 3.3~5VDC.

+ GND: chân mass hoặc nối đất.

+ 5V: Chân cấp nguồn 5VDC từ cổng USB, tối đa 500mA.

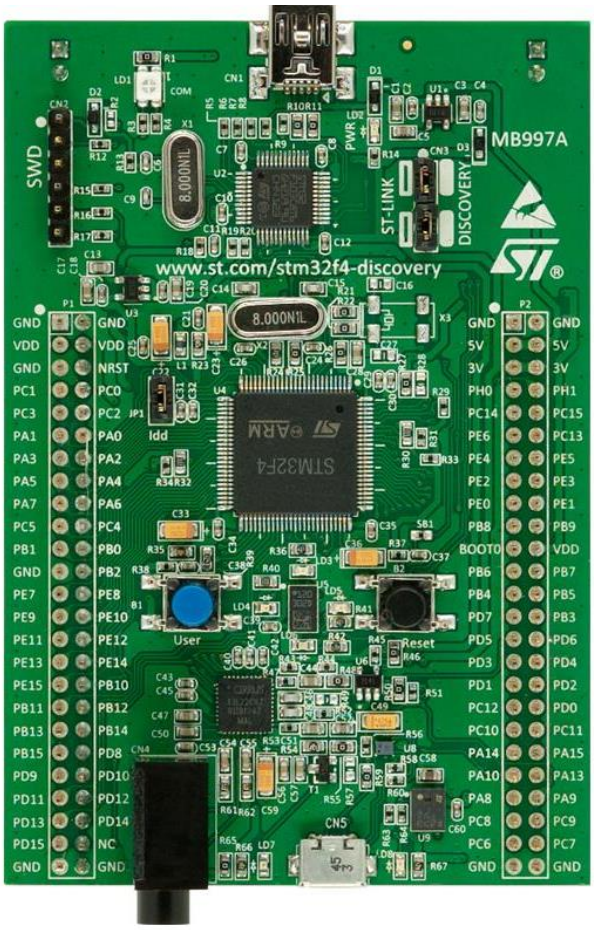
+ DTR: Chân tín hiệu DTR, thường được dùng để cấp tín hiệu Reset nạp chương trình cho mạch Arduino.

+ 3.3V: Chân nguồn 3.3VDC (dòng cấp rất nhỏ tối đa 100mA), không sử dụng để cấp nguồn, thường chỉ sử dụng để thiết đặt mức tín hiệu Logic.

IV. Phần cứng và phần mềm lập trình giao diện

a) STM32F407 DISCOVERY

Kit STM32F407 Discovery với vi điều khiển hiệu suất cao STM32F407VGT6, cho phép người dùng dễ dàng phát triển các ứng dụng xử lý tín hiệu số (hình ảnh, video…). Nó bao gồm một công cụ ST-LINK tích hợp sẵn trên bảng mạch giúp nạp chương trình, gỡ lỗi nhanh chóng. Hiện nay, Kit STM32F407 Discovery là loại kit được sử dụng rất nhiều ở các trường đại học, cao đẳng trong giảng dạy vi điều khiển ARM.



Thông số kỹ thuật:

• Vi điều khiển chính: STM32F407VGT6 microcontroller featuring 32-bit ARM Cortex-M4F core, 1 MB Flash, 192 KB RAM in an LQFP100 package.

• Tích hợp sẵn mạch nạp và Debug ST-LINK/V2 with selection mode switch to use the kit as a standalone ST-LINK/V2 (with SWD connector for programming and debugging).

• Nguồn cấp từ cổng Mini USB qua các IC nguồn chuyển thành 3v3 để cấp cho MCU.

• Có sẵn các chân nguồn: 3 V and 5 V.

• Có sẵn cảm biến gia tốc: LIS302DL, ST MEMS motion sensor, 3-axis

• Có sẵn bộ xử lý âm thanh: MP45DT02, ST MEMS audio sensor, omni-directional digital microphone.

• Có sẵn bộ: CS43L22, audio DAC with integrated class D speaker driver.

• Có sẵn 8 Led và Led thông báo trạng thái nguồn.

• Có nút nhấn và nút Reset tích hợp.

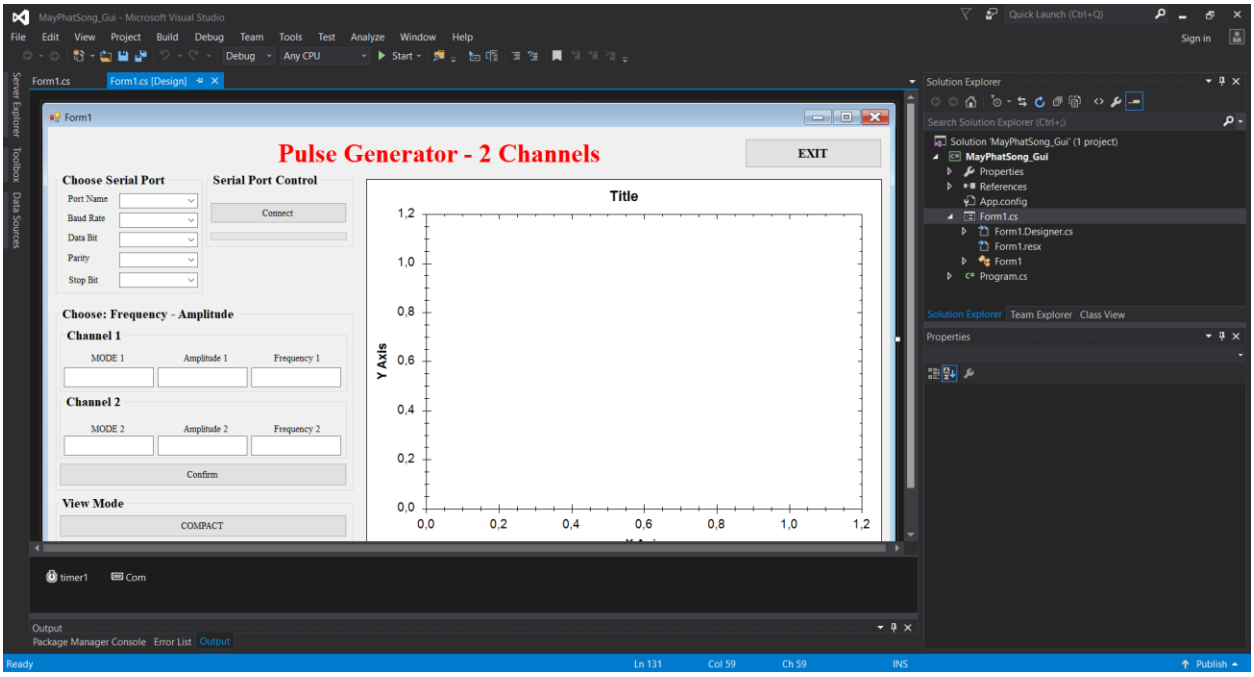
• Có cổng Micro USB OTG

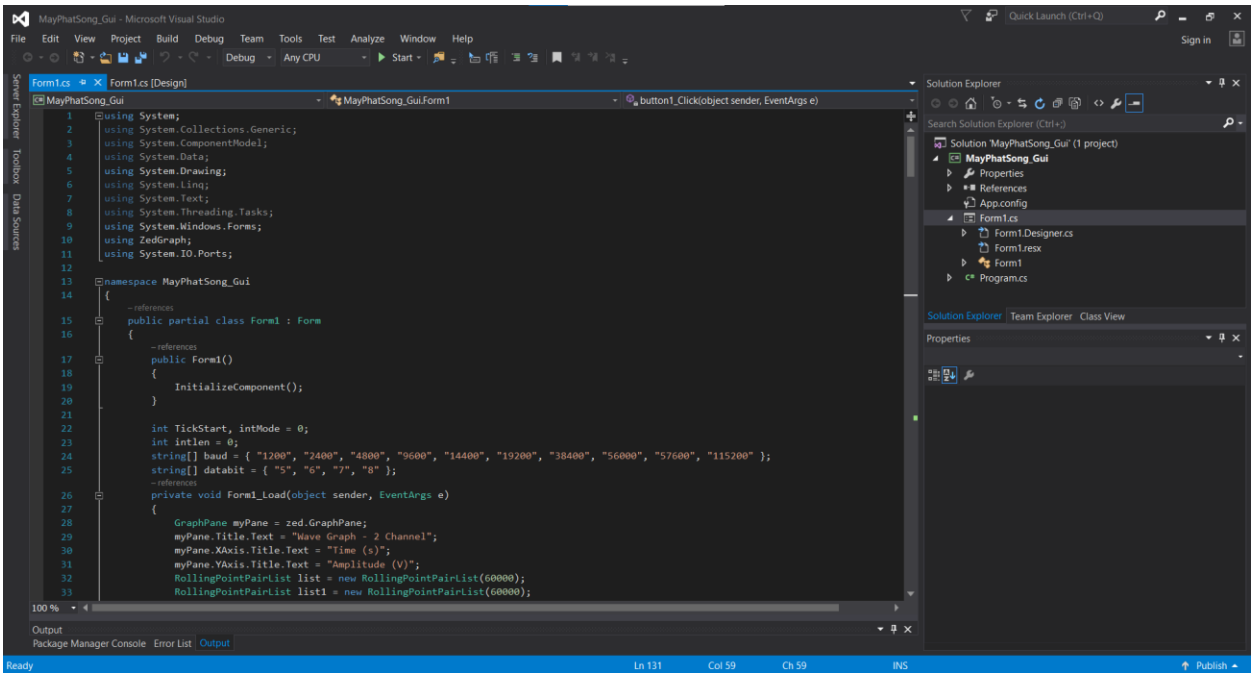
• DSP Application.

b) VISUAL STUDIO

Visual studio là một phần mềm hỗ trợ lập trình website do Microsoft phát hành. Hai ngôn ngữ chính sử dụng trong visual studio là VB+ và C#. Những ngôn ngữ này giúp người dùng có khả năng lập trình hệ thống một cách dễ dàng và nhanh chóng.

Nhóm sử dụng tính năng Windows Forms Designer: dùng visual studio với mục đích xây dựng GUI sử dụng Windows Forms, để xây dựng các nút điều kiện bên trong, đôi khi có thể khoá chúng vào bên cạnh mẫu.




MayPhatSong_Gui - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Sign in
Debug Any CPU Start
Form1.cs Form1.cs [Design]
MayPhatSong_Gui MayPhatSong_Gui.Form1 button1_Click(object sender, EventArgs e)
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using ZedGraph;
using System.IO.Ports;
namespace MayPhatSong_Gui
{
public partial class Form1 : Form
{
public Form1()
{
InitializeComponent();
}
int TickStart, intMode = 0;
int intlen = 0;
string[] baud = { "1200", "2400", "4800", "9600", "14400", "19200", "38400", "56000", "57600", "115200" };
string[] databit = { "5", "6", "7", "8" };
private void Form1_Load(object sender, EventArgs e)
{
GraphPane myPane = zed.GraphPane;
myPane.Title.Text = "Wave Graph - 2 Channel";
myPane.XAxis.Title.Text = "Time (s)";
myPane.YAxis.Title.Text = "Amplitude (V)";
RollingPointPairList list = new RollingPointPairList(60000);
RollingPointPairList list1 = new RollingPointPairList(60000);
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'MayPhatSong_Gui' (1 project)
MayPhatSong_Gui
Properties
References
App.config
Form1.cs
Form1.Designer.cs
Form1.resx
Form1
Program.cs
Solution Explorer Team Explorer Class View
Properties
Output
Package Manager Console Error List Output
Ready
Ln 131 Col 59 Ch 59 INS
Publish

B. THỰC HIỆN GIẢI THUẬT

I. DAC và các chương trình tạo sóng

Nguyên lý hoạt động: Giá trị điện áp được đẩy vào thanh ghi DAC_DHRyyyD và sẽ tự động được chuyển vào thanh ghi DAC_DORx sau 1 chu kỳ xung clock (với mode tự đông), khi thanh ghi DAC_DORx có giá trị thì mức điện áp sẽ xuất ra tương ứng. Mức điện áp:

DACoutput = Vref x DOR/4095 (0 – 3V)

1. Sơ lược về cấu hình DAC

- Nhóm sử dụng hai kênh DAC, kết hợp với DMA và Timer để tạo sóng:

+ DAC1 (Pin: PA4): DMA1 – Stream 5 – Channel 7 – Timer 7

+ DAC2 (Pin: PA5): DMA1 – Stream 6 – Channel 7 – Timer 6

Table 20. DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX		SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX		SPI3_TX
Channel 1	I2C1_RX		TIM7_UP		TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1		I2S2_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5			TIM3_CH4 TIM3_UP		TIM3_CH1 TIM3_TRIG	TIM3_CH2		TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2		TIM5_UP	
Channel 7		TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

- Cấu hình cho một kênh DAC DMA để tạo sóng bao gồm các phần:

+ Cấu hình xung clock cho phép hoạt động DAC, DMA1, GPIOA

+ Cấu hình chân ngõ ra DMA (PA4, PA5)

+ Cấu timer trigger cho DAC (Timer 7, Timer 6)

+ Cấu hình DMA cho DAC: DMA_Channel, DMA_PeripheralBaseAddr, DMA_Memory0BaseAddr, DMA_BufferSize, DMA_DIR = DMA_DIR_MemoryToPeripheral, …

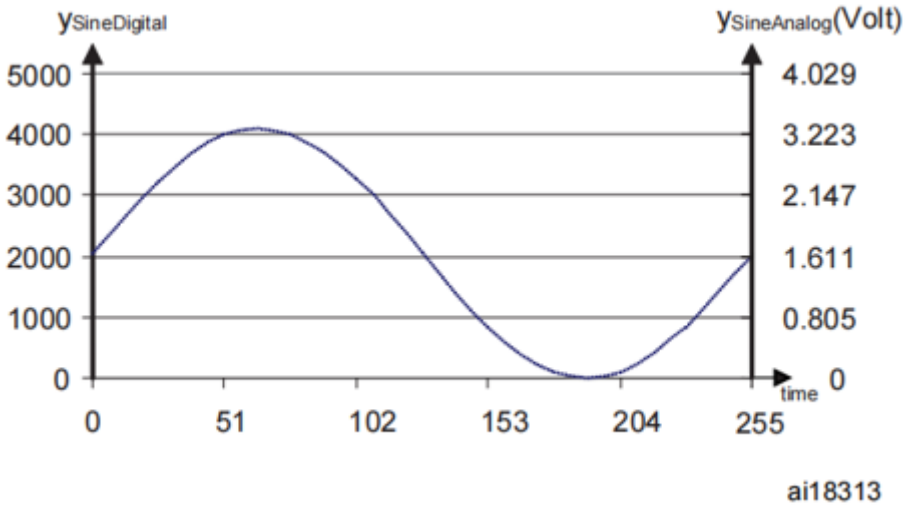
- Minh họa cho cấu hình DAC1 (Pin: PA4): DMA1 – Stream 5 – Channel 7 – Timer 7

```
GPIO_InitTypeDef GPIO_InitStructure;
		DAC_InitTypeDef  DAC_InitStructure;
		DMA_InitTypeDef  DMA_InitStructure;

		/* DMA1 clock enable */
		RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);
		/* GPIOA clock enable (to be used with DAC) */
		RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
		/* DAC Periph clock enable */
		RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);

		// Cấu hình chân ngõ ra PA4 - DAC Channel 1
		GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
		GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
		GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
		GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
		GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
		GPIO_Init(GPIOA, &GPIO_InitStructure);

		// Cấu hình trigger từ Timer:
		DAC_InitStructure.DAC_Trigger = DAC_Trigger_T7_TRGO;
		DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
		DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
		DAC_Init(DAC_Channel_1, &DAC_InitStructure);

		// Cấu hình DMA cho DAC: DAC1 sử dụng DMA1 Stream 5 Channel 7
		DMA_DeInit(DMA1_Stream5);
		DMA_InitStructure.DMA_Channel = DMA_Channel_7;
		DMA_InitStructure.DMA_PeripheralBaseAddr = DAC_DHR12R1_ADDRESS; // địa chỉ
thanh ghi chứa giá trị điện áp
		DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&value1;
		DMA_InitStructure.DMA_BufferSize = 1000; // 1000 mẫu dữ liệu
		DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
		DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
```

```
                DMA_InitStructure.DMA_DIR = DMA_DIR_MemoryToPeripheral;
                DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
                DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
                DMA_InitStructure.DMA_Mode = DMA_Mode_Circular; // Mode Circular
                DMA_InitStructure.DMA_Priority = DMA_Priority_High;
                DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
                DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
                DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
                DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
                DMA_Init(DMA1_Stream5, &DMA_InitStructure);

// Cấu hình cho phép hoạt động chức năng DMA, DAC, DAC DMA
/* Enable DMA1_Stream5 */
 DMA_Cmd(DMA1_Stream5, ENABLE);
 /* Enable DAC Channel1 */
 DAC_Cmd(DAC_Channel_1, ENABLE);
 /* Enable DMA for DAC Channel1 */
 DAC_DMACmd(DAC_Channel_1, ENABLE);
```

2. Các chương trình tạo sóng (sine, vuông, tam giác, sóng răng cưa)

a) Sóng sine



Công thức tạo giá trị sóng sin cho bộ DAC:

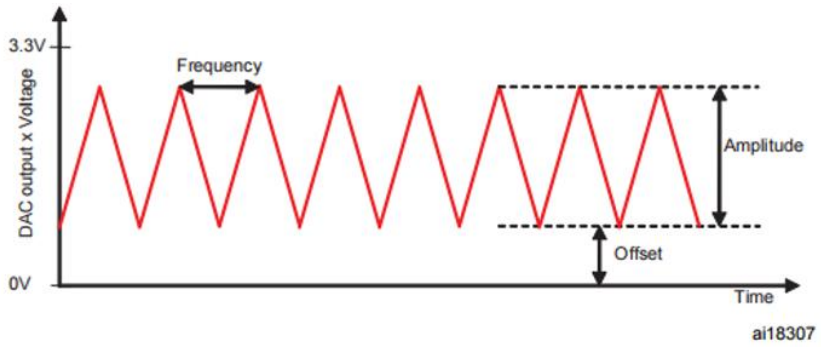
$$Y_{SineDigital}(x) = \left(\sin\left(2\Pi \cdot \frac{x}{n_S}\right) + 1\right) \cdot \frac{0xFFF + 1}{2}$$

Với công thức trên, ta tiến hành dùng hàm sin trong thư viện math.h để khởi tạo một mảng giá trị tạo sóng sin gồm 1000 mẫu dữ liệu như sau:

```
void Sine_Wave(uint16_t *value, float A)
{
    for(int i=0;i<1000;i++)
        value[i] = (A*sin(2*pi*i/1000)+A)*4095/3/2;
}
```

Với A là hệ số khuếch đại biên độ sóng, và value là mảng dùng để lưu các giá trị khởi tạo, mảng này có 1000 phần tử.

b) Sóng tam giác

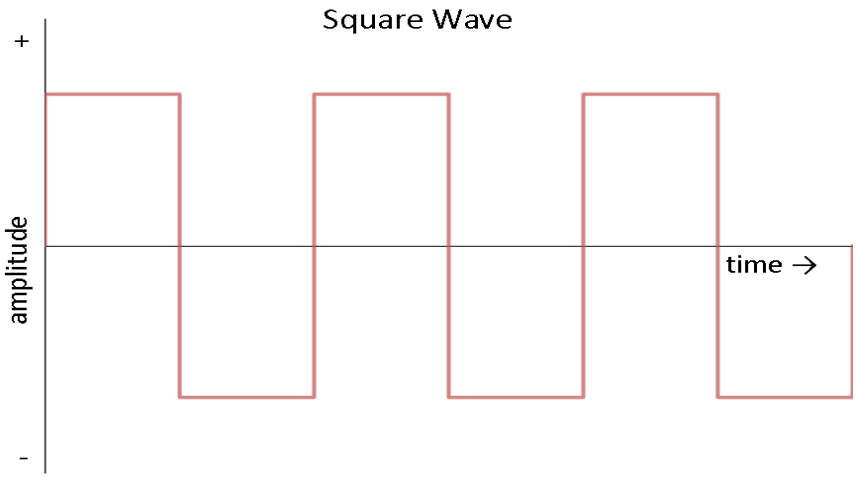


Hàm tạo sóng tam giác:

```
void Triangular_Wave(uint16_t *value, float A)
{
    for(int i=0;i<1000;i++)
        if(i<500) value[i] = (A*i/500)*4095/3;
        else value[i] = (2*A-A*i/500)*4095/3;
}
```

Với A là hệ số khuếch đại biên độ sóng, và value là mảng dùng để lưu các giá trị khởi tạo, mảng này có 1000 phần tử.

c) Sóng vuông

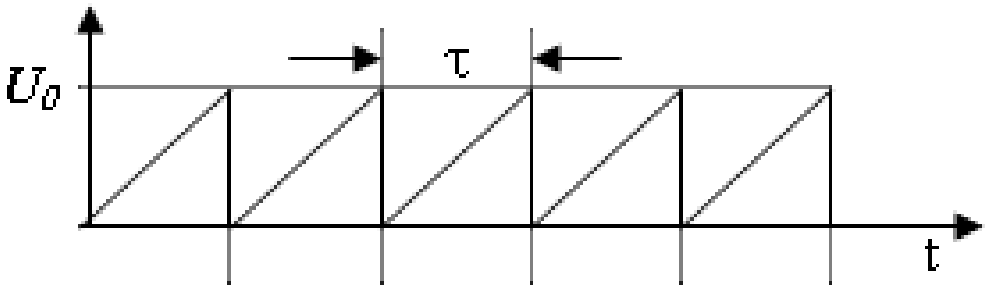


Hàm tạo sóng vuông:

```
void Square_Wave(uint16_t *value, float A)
{
    for(int i=0;i<1000;i++)
        if(i<500) value[i] = 0;
        else value[i] = A*4095/3;
}
```

Với A là hệ số khuếch đại biên độ sóng, và value là mảng dùng để lưu các giá trị khởi tạo, mảng này có 1000 phần tử.

d) **Sóng răng cưa**



Hàm tạo sóng răng cưa:

```
void Linear_Wave(uint16_t *value, float A)
{
    for(int i=0;i<1000;i++)
     value[i] = (A*i/1000)*4095/3;
}
```

Với A là hệ số khuếch đại biên độ sóng, và value là mảng dùng để lưu các giá trị khởi tạo, mảng này có 1000 phần tử.

❑ **Về tần số của sóng tạo ra:**

- Để đặt tần số của tín hiệu sóng, người dùng phải đặt tần số ($f_{timerTRGO}$) của đầu ra kích hoạt bộ định thời. Tần số của sóng tạo ra là:

$$f_{wave} = \frac{f_{timerTRGO}}{n_s}$$

- Với $f_{timerTRGO}$ là tần số kích của Timer, và n_s là số mẫu giá trị của sóng trong 1 chu kỳ của nó và bằng số giá trị khởi tạo sóng được lưu là 1000. Như vậy để điều chỉnh tần số cho sóng ta sẽ thay đổi giá trị tần số kích của timer ($f_{timerTRGO}$) bằng cách thay đổi giá trị của 2 thanh ghi PSC và ARR cho phù hợp.

- Ta giả sử giá trị nạp vào thanh ghi PSC là 84*x - 1 và giá trị nạp vào thanh ghi ARR là y - 1.

$$f_{wave} = \frac{84000000Hz}{84 * x * y * 1000} = \frac{1000}{x * y}$$

Ta tiến hành xác định giá trị x và y như sau:

```
// Tinh toan x, y --> Prescale, Counter Period nap vao cho Timer Trigger
uint32_t x=1, y=0;
for (x=10; x<=20; x++)
{
    y =(uint32_t)1000/(x*F1);
    if((10<= y) && (y<=5000)) break;
}

TIM7->CR1 = 0;      // stop Timer7
TIM7->PSC = x*84-1;       // clk = SystemCoreClock /2/(PSC+1) = 1MHz
TIM7->ARR = y-1;
TIM7->CNT = 0;
TIM7->EGR = 1;      // update registers;
TIM7->SR  = 0;      // clear overflow flag
TIM7->CR1 = 1;      // enable Timer7
```

Ta dùng bộ Timer 7 làm tín hiệu kích cho DAC1 và Timer 6 làm tín hiệu kích cho DAC 2 và cách tính toán giá trị nạp vào hai thanh ghi PSC và ARR là như nhau cho 2 timer.

II. ADC đọc tín hiệu Analog từ hai kênh bộ DAC

- Vì không có dao động ký nên nhóm đã sử dụng thêm ngoại vi ADC tích hợp sẵn trong vi điều khiển, tiến hành đọc tín hiệu Analog được tạo ra từ hai kênh của bộ DAC rồi giử dữ liệu để hiển thị lên giao diện trên máy tính.
- Nhóm sử dụng ADC1 trên chân PA6 và ADC2 trên chân PA7 của STM32F4. Do đó, để đọc được tín hiệu DAC xuất ra từ hai kênh DAC1 (PA4) và DAC2 (PA5) thì ta phải nối các chân như sau:

Chân PA6 (ADC1) nối với chân PA4 (DAC1)

Chân PA7 (ADC2) nối với chân PA5 (DAC1)

- Bên cạnh đó, sử dụng:

+ DMA cho ADC1: DMA2, Channel 0, Stream 0.

+ DMA cho ADC2: DMA2, Channel 1, Strem 2.

Table 21. DMA2 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1		TIM8_CH1 TIM8_CH2 TIM8_CH3		ADC1		TIM1_CH1 TIM1_CH2 TIM1_CH3	
Channel 1		DCMI	ADC2	ADC2				DCMI
Channel 2	ADC3	ADC3				CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX		SPI1_RX	SPI1_TX		SPI1_TX		
Channel 4			USART1_RX	SDIO		USART1_RX	SDIO	USART1_TX
Channel 5		USART6_RX	USART6_RX				USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	
Channel 7		TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3			TIM8_CH4 TIM8_TRIG TIM8_COM

- Cấu hình cho hai kênh ADC1 và ADC2 bằng lệnh: CONFIG_ADC_Channel1 () và CONFIG_ADC_Channel2 ().

- Một số cấu hình cơ bản của ADC như sau:

+ Cấu hình clock cho ADC, GPIOA và DMA

+ Cấu hình GPIO chế độ Input (PA6, PA7)

+ Cấu hình chung cho ADC (ADC_Mode_Independent, ADC_Prescaler_Div4, ADC_Resolution_12b, ADC_DataAlign_Right …).

+ Cấu hình DMA cho ADC (DMA_Channel, DMA_PeripheralBaseAddr, DMA_Memory0BaseAddr, DMA_DIR = DMA_DIR_PeripheralToMemory, …)

- Minh họa cho cấu hình ADC1 (pin PA6), DMA2, Channel 0, Stream 0:

```
ADC_InitTypeDef       ADC_InitStructure;
ADC_CommonInitTypeDef ADC_CommonInitStructure;
DMA_InitTypeDef       DMA_InitStructure;
GPIO_InitTypeDef      GPIO_InitStructure;

// Cấu hình xung clock cho ADC, GPIO, DMA
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);

// Cấu hình chân PA6 ADC1
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
```

```
		GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
		GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
		GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
		GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
		GPIO_Init(GPIOA, &GPIO_InitStructure);

		 /* ADC Common Init **********************************************************/
		ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
		ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div4;
		ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
		ADC_CommonInitStructure.ADC_TwoSamplingDelay =
ADC_TwoSamplingDelay_5Cycles;
		ADC_CommonInit(&ADC_CommonInitStructure);

/* ADC1 Init ****************************************************************/
		ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
		ADC_InitStructure.ADC_ScanConvMode = DISABLE;
		ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
		ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
		ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T1_CC1;
		ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
		ADC_InitStructure.ADC_NbrOfConversion = 1;
		ADC_Init(ADC1, &ADC_InitStructure);

// Cau hinh DMA2, CHANNEL 0, STREAM 0
		DMA_InitStructure.DMA_Channel = DMA_Channel_0;
		DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)&(ADC1->DR);
		DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&adc_value[0];
		DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
		DMA_InitStructure.DMA_BufferSize = 1;
		DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
		DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Disable;
		DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
		DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
		DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
		DMA_InitStructure.DMA_Priority = DMA_Priority_High;
		DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
```

```
		DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
		DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
		DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
		DMA_Init(DMA2_Stream0, &DMA_InitStructure);
		DMA_Cmd(DMA2_Stream0, ENABLE);

		ADC_RegularChannelConfig(ADC1,ADC_Channel_6,1,ADC_SampleTime_15Cycles);
		/* Start ADC1 */
		ADC_Cmd(ADC1,ENABLE);

		/* Enable DMA request after last transfer (Single-ADC mode) */
ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);
		/* Enable ADC1 DMA */
ADC_DMACmd(ADC1, ENABLE);
```

- Nguyên lý hoạt động của ADC:

Sau khi cấu hình ADC cho hai kênh, ta kích hoạt bộ ADC của hai kênh hoạt động bằng lệnh ADC_SoftwareStartConv(ADC1) và ADC_SoftwareStartConv(ADC2).

Khi hai lệnh trên được kích hoạt, hai kênh ADC sẽ lấy dữ liệu lần lượt ở chân PA6 (ADC1) và PA7 (ADC2). Giá trị ADC sẽ được lưu vào thanh ghi ADC1->DR và ADC2->DR. Ta cài đặt DMA ở chế độ Circular nên khi kết thúc một lần chuyển đổi, bộ ADC sẽ tiếp tục thực hiện một lần chuyển đổi khác ngay lập tức.

Hai thanh ghi ADC1->DR và ADC2->DR sau khi nhận được dữ liệu sẽ gửi yêu cầu cho DMA chuyển dữ liệu này vào RAM ở địa chỉ của adc_value [0] và adc_value [1]. Dữ liệu từ adc_value [0] và adc_value [1] sẽ được gửi lên máy tính để vẽ dạng sóng.

III. Truyền - nhận và xử lý dữ liệu qua UART

a) Sơ lược về cấu hình UART

Sử dụng UART4 kết hợp DMA cho quá trình truyền và nhận dữ liệu. Một số cấu hình cơ bản của UART4:

+ Cho phép xung clock hoạt động UART4, GPIOA, DMA1

+ Connect UART4 pins to AF2

+ Cấu hình chân UART TX: PA0, chân UART RX: PA1

+ Tốc độ baud rate là 115200; 8bit dữ liệu; 1 stop bit và no parity; …

+ Cho phép UART4 hoạt động và cho phép UART4 DMA Rx&Tx hoạt động

+ Cấu hình DMA UART Rx: DMA1, Channel 4, Stream 2 (DMA_Channel_4; DMA_PeripheralBaseAddr = (uint32_t)&UART4->DR; DMA_Memory0BaseAddr = (uint32_t)rxbuff; DMA_DIR = DMA_DIR_PeripheralToMemory; DMA_BufferSize = BUFF_SIZE_RX; …)

+ Cấu hình DMA UART Tx: DMA1 Channel 4, Stream 4 (DMA_Channel_4; DMA_PeripheralBaseAddr = (uint32_t)&(UART4->DR); DMA_Memory0BaseAddr = (uint32_t)txbuff; DMA_DIR = DMA_DIR_MemoryToPeripheral; DMA_BufferSize = BUFF_SIZE_TX; …)

b) Phương thức truyền dữ liệu từ vi xử lý lên máy tính

- Sử dụng Timer 3 để tạo ngắt tiến hành truyền chuỗi. Quá trình truyền dữ liệu từ vi xử lý lên máy tính khi Timer 3 tràn.

- Đầu tiên chúng ta đọc giá trị sóng tạo được về bằng cách nối bộ ADC vào chân tạo sóng DAC sau đó lưu giá trị ADC đọc về được vào giá trị cụ thể, ta sẽ lưu giá trị ADC đọc về vào 2 giá trị adc_value[0] và adc_value[1]. Sau đó ta tiến hành nạp cả 2 giá trị adc_value này vào một chuỗi để truyền một lần cả hai giá trị này lên máy tính với cú pháp như sau:

start|<giá trị 1>|<giá trị 2>|end|

với <giá trị 1> và <giá trị 2> là 2 giá trị adc_value [0] và adc_value [1] sau khi đã được lưu thành dạng chuỗi ký tự.

- Chi tiết về hàm truyền: **void Transmit_Data_FromADC(void)**, các bước thực hiện:

+ Đầu tiên, ta tiến hành lưu giá trị ADC vào 2 biến tạm để tiến hành chuyển thành dạng chuỗi:

```
void Transmit_Data_FromADC(void)
{
    // Gán giá trị ADC về
    int16_t ReceiveADC1 = adc_value[0];
    int16_t ReceiveADC2 = adc_value[1];
    int8_t indexADC1, indexADC2;
```

+ Tiếp theo, tiến hành lưu 2 giá trị ADC này thành dạng chuỗi chuẩn bị cho quá trình truyền dữ liệu lên máy tính, nhằm tối ưu bộ nhớ và tốc độ truyền, ta sẽ tiến hành kiểm tra giá trị ADC trước để cấp bộ nhớ cho phù hợp:

```
// Kiểm tra độ lớn của giá trị ADC để cấp bộ nhớ cho phù hợp
if (ReceiveADC1<10) // Gia tri ADC1 doc ve nho hon 10
    indexADC1 =1; // de bit 0 chua ADC1=a, bit 1 chua "|"
else if (ReceiveADC1<100)
    indexADC1 = 2; // de bit index=0 1 chua gia tri ADC1=a b, index=2 chua "|"
else if (ReceiveADC1<1000)
    indexADC1 = 3; // de bit index = 0 1 2 chua gia tri ADC1 = a b c, index = 3 chua "|"
else
    indexADC1 = 4; // de bit index = 0 1 2 3 chua gia tri ADC1 = a b c d, index = 4 chua "|"

if (ReceiveADC2<10)
    indexADC2 =1;
else if (ReceiveADC2<100)
    indexADC2 = 2;
else if (ReceiveADC2<1000)
    indexADC2 = 3;
else
    indexADC2 = 4;

// Gan gia tri ADC doc ve thanh 1 chuoi su dung ham snprintf()
snprintf(str_ADC1, 10, "%i", ReceiveADC1);
str_ADC1[indexADC1]='|';
snprintf(str_ADC2, 10, "%i", ReceiveADC2);
str_ADC2[indexADC2]='|';
```

+ Sau khi đã lưu 2 giá tri ADC về dạng chuỗi, ta tiến hành lưu chúng vào mảng txbuff theo cú pháp start|<giá trị 1>|<giá trị 2>|end| và cuối cùng là tiến hành truyền mảng dữ liệu này lên máy tính bằng chế độ DMA (đầu tiên xóa cờ DMA_Stream4 là DMA_FLAG_TCIF4 để cho phép truyền hay truyền tiếp lần thứ hai; nạp giá trị NDTR (thanh ghi chỉ số DMA) bằng số byte cần truyền; sau đó cho phép DMA, sau khi cho phép DMA thì DMA sẽ hoạt động ngay và yêu cầu truyền dữ liệu)

```
// Dong goi du lieu vao txbuff
txbuff[0] = 's';
txbuff[1] = 't';
txbuff[2] = 'a';
txbuff[3] = 'r';
txbuff[4] = 't';
txbuff[5] = '|';
for(int j=6; j<= indexADC1 + 6; j++) txbuff[j] = str_ADC1[j-6];
for(int j=indexADC1 + 7; j<= indexADC1 + 7 + indexADC2 ; j++) txbuff[j] = str_ADC2[j-indexADC1-7];
txbuff[indexADC1 + 7 + indexADC2 + 1] = 'e';
txbuff[indexADC1 + 7 + indexADC2 + 2] = 'n';
txbuff[indexADC1 + 7 + indexADC2 + 3] = 'd';
txbuff[indexADC1 + 7 + indexADC2 + 4] = '|';

DMA_ClearFlag(DMA1_Stream4, DMA_FLAG_TCIF4);
DMA1_Stream4->NDTR = BUFF_SIZE_TX;
DMA_Cmd(DMA1_Stream4, ENABLE);
```

❑ **Xét về phương diện máy tính:**

- Khi nhận được chuỗi dữ liệu từ vi xử lý gửi lên sẽ qua bước tiến hành xử lý chuỗi dữ liệu đó để có thể vẽ đồ thị dạng sóng:

```
string receive, sum;
string[] List;
1 reference
private void Com_DataReceived(object sender, SerialDataReceivedEventArgs e)
{
    receive = Com.ReadExisting();

    List = receive.Split('|');
    if ((List[0] == "start") && (List[3] == "end"))
    {
        Draw(List[1], List[2]);
    }
}
```

- Như đã trình bày, chuỗi dữ liệu gửi lên máy tính sẽ có dạng: start|<giá trị 1>|<giá trị 2>|end|. Do đó tiến hành cắt chuỗi dữ liệu này và lưu vào một List, với ký tự cắt chuỗi sẽ là "|". Khi cắt xong, ta se xét nếu List[0] == "start"; List[3] = "end" thì List[1] và List[2] lần lượt là hai giá trị adc_value[0] (ADC1) và adc_value[1] (ADC2) của hai kênh ADC, sau đó ta gọi hàm Draw để vẽ đồ thị hai giá trị này.

- Chi tiết về hàm vẽ đồ thị được trình bày ở phần giao diện.

c) Phương thức nhận dữ liệu từ máy tính gửi về vi xử lý

❑ **Xét về phương diện máy tính:** Đầu tiên để thực hiện việc nhận dữ liệu từ máy tính về vi xử lý, ta sẽ xét cách xử lý dữ liệu và truyền từ máy tính xuống vi xử lý.

- Ở phần giao diện điều khiển, khi người dùng điền các giá trị MODE 1, Amplitude 1, Frequency 1, MODE 2, Amplitude 2, Frequency 2, sau đó nhấn nút Confirm, ta tiến hành thực hiện việc truyền các giá trị này về cho vi xử lý thực hiện. Giải thuật truyền dữ liệu từ trên máy tính như sau (lấy ví dụ truyền cho 1 kênh, kênh còn lại thực hiện tương tự):

```
string s;
            if (pbConnect.Text == "Disconnect" && Mode1.Text != "" && Amplitude1.Text != 
"" && Frequency1.Text != "")
            {
                s = "|";
                switch (Amplitude1.TextLength)
                {
                    case 1:
                        Amplitude1.Text = "0" + "0" + Amplitude1.Text;
                        break;
                    case 2:
                        Amplitude1.Text = "0" + Amplitude1.Text;
                        break;
                    case 3:
                        Amplitude1.Text = Amplitude1.Text;
                        break;
                }
                switch (Frequency1.TextLength)
                {
                    case 1:
                        Frequency1.Text = "0" + "0" + "0" + "0" + "0" + Frequency1.Text;
                        break;
                    case 2:
                        Frequency1.Text = "0" + "0" + "0" + "0" + Frequency1.Text;
                        break;
```

```
                    case 3:
                        Frequency1.Text = "0" + "0" + "0" + Frequency1.Text;
                        break;
                    case 4:
                        Frequency1.Text = "0" + "0" + Frequency1.Text;
                        break;
                    case 5:
                        Frequency1.Text = "0" + Frequency1.Text;
                        break;
                    case 6:
                        Frequency1.Text = Frequency1.Text;
                        break;
                }
                if (Mode1.TextLength >= 2)
                {
                    Mode1.Text = "3";
                }
            }
            else
            {
                MessageBox.Show("Vui long nhap thong so dieu khien", "Thong Bao",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }
```

- Giải thích:

+ Quy định chuỗi (MODE, biên độ, tần số của 2 kênh) truyền về cho vi xử lý là một chuỗi gồm 30 ký tự.

+ Vậy, quy ước cho phần chuỗi biên độ sẽ gồm 3 ký tự, tần số có 6 ký tự và MODE sẽ là 3 ký tự. Giữa những giá trị MODE, biên độ, tần số này với nhau, ta chèn các ký tự "|" để ngăn cách, do đó ở mỗi kênh cần thêm 3 ký tự "|".

+ Vậy, số lượng ký tự của 1 chuỗi ((MODE, biên độ, tần số của 2 kênh) sẽ là:

(3+6+3+3)*2 = 30 ký tự

+ Nếu biên độ và tần số nhập vào không đủ cấu trung khung quy định như trên, ta sẽ tiến hành chèn thêm các số "0" để đáp ứng đủ số lượng ký tự.

+ Sau đó thực hiện việc truyền dữ liệu từ máy tính về vi xử lý như sau:

```
       Com.Write(Amplitude1.Text);
                 Com.Write(s);
                 Com.Write(Frequency1.Text);
                 Com.Write(s);
                 Com.Write(Mode1.Text);
                 Com.Write(s);
                 Com.Write(Amplitude2.Text);
                 Com.Write(s);
                 Com.Write(Frequency2.Text);
                 Com.Write(s);
                 Com.Write(Mode2.Text);
                 Com.Write(s);
                 int lensend = Mode1.TextLength + Amplitude1.TextLength +
Frequency1.TextLength + Mode2.TextLength + Amplitude2.TextLength + Frequency2.TextLength
+ 6;
                 for (int a = 0; a < 30 - lensend; a++)
                 {
                     Com.Write(s);
                 }
```

❑ Vi xử lý nhận chuỗi và xử lý dữ liệu:

- Trong phần UART RX DMA, ta sẽ sử dụng ngắt, để khi nhận dữ liệu xong báo lên cho CPU biết dữ liệu đã xong:

```
/* Enable DMA Interrupt to the highest priority */
NVIC_InitStructure.NVIC_IRQChannel = DMA1_Stream2_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

// Bật cờ DMA_IT_TC để cho phép ngắt DMA UART RX ở kênh IRQn
DMA_ITConfig(DMA1_Stream2, DMA_IT_TC, ENABLE);
```

- Nếu ngắt DMA xảy ra thì nhảy vào kênh IRQn, để cho phép thực hiện phải bậc cờ DMA_IT_TC.

- Chương trình ngắt DMA1_Stream2_IRQHandler(void) khi cờ DMA_IT_TC bậc lên, ta sẽ vào ngắt này để nhận dữ liệu về và xử lý dữ liệu.

- Trong chương trình ngắt này:

+ Đầu tiên, ta xóa cờ ngắt DMA_IT_TCIF2 đi và chép toàn bộ dữ liệu từ rxbuff về một mảng receive_data:

```
void DMA1_Stream2_IRQHandler(void)
{
  /* Clear the DMA1_Stream2 TCIF2 pending bit */
  DMA_ClearITPendingBit(DMA1_Stream2, DMA_IT_TCIF2);
  for(uint8_t i=0; i<BUFF_SIZE_RX; i++) receive_data[i]=rxbuff[i];
```

+ Có thể thấy với cách truyền dữ liệu từ máy tính như trên, chuỗi rxbuff của ta sẽ có cấu trúc cơ bản như sau:

Amplitude1|Frequency1|Mode1|Amplitude2|Frequency2|Mode2|

+ Do đó, tổng cộng sẽ có 6 thông số cần đọc về và xử lý. Giải thuật tách từng thông số trong 6 thông số đó như sau:

```
// Sau khi đọc về, tiến hành tách chuỗi dữ liệu
uint8_t m = 0;
uint8_t n=0, p=0, q=0;
for (m = 0; m<6; m++) // Tổng cộng có 6 phần tử cần đọc về là A1, F1, Mode1, A2, F2, Mode2
{
    for(n=0; n<BUFF_SIZE_RX; n++)
    {
        if (receive_data[n] != 124) // khac dau | (ASCII cua | = 124)
            tmp_string[n] = receive_data[n];
        else
        {
            tmp_string[n]='₩0'; // ky tu ₩0 ket thuc chuoi
            pointer = &parameter[m];
            *pointer = atof(tmp_string); // Chuyen doi chuoi thanh gia tri so thuc double
            break;
        }
    }
    // Tien hanh cat bo thong so vua duoc lay o tren trong mang receive_data
    p = 0;
        for (q = n+1; q<BUFF_SIZE_RX; q ++)
        {
            receive_data[p] = receive_data[q];
            if (p<BUFF_SIZE_RX) p++;
            else
            {
                p = 0;
                break;
            }
        }
}
```

- Giải thích:

+ Đặt một biến m là số thông số cần đọc về, dùng một vòng lập để tách từng giá trị Amplitude, Frequency, Mode của mỗi kênh ra từ mảng receive_data.

+ Ý tưởng cho việc tách này là kiểm tra xem phần tử thứ n ở mảng receive_data có khác ký tự "|" có giá trị ASCII là 124 hay không. Nếu không phải là ký tự "|" thì đưa giá trị này vào 1 mảng tạm. Nếu phải thì thêm ký tự "\0" để kết thúc mảng, gán giá trị từ mảng tạm vừa tách đó vào mảng parameter[m] tương ứng với thông số thứ m.

+ Sau đó tiến hành lấy ra các ký tự của thông số vừa tách ở trên trong mảng receive_data ra và cứ lặp lại quá trình như thế cho đến khi m = 6 là ta đã có đủ 6 thông số về mode, biên độ, tần số của hai kênh từ máy tính gửi về.

- Với cách thức thực hiện như trên, sau khi tách chuỗi dữ liệu nhận về từ máy tính, ta sẽ có các giá trị biên độ tần số như sau:

A1 = parameter[0];

F1 = parameter[1];

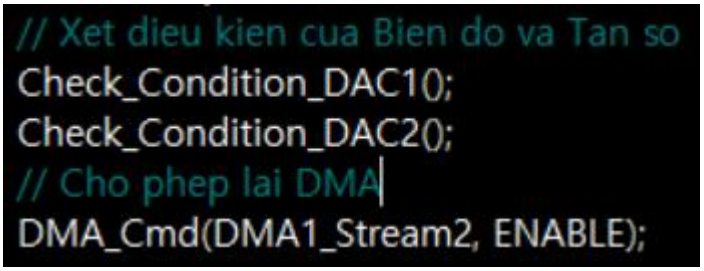
MODE1 = parameter[2];

A2 = parameter[3];

F2 = parameter[4];

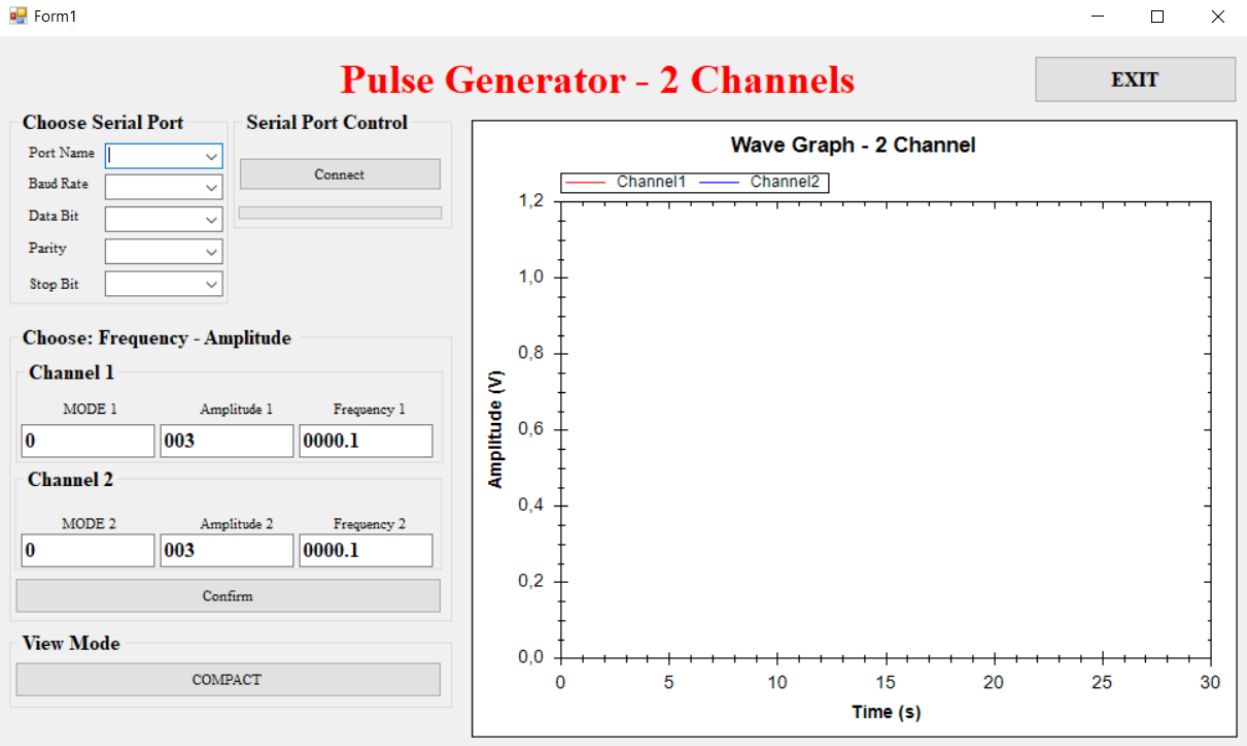
MODE2 = parameter[5];

- Tiếp theo ta sẽ dùng các giá trị A1, F1, MODE1 và A2, F2, MODE2 này để kiểm tra điều kiện và tiến hành vẽ sóng. Sau đó cho phép lại DMA bằng lệnh DMA_Cmd(DMA1_Stream2, ENABLE).



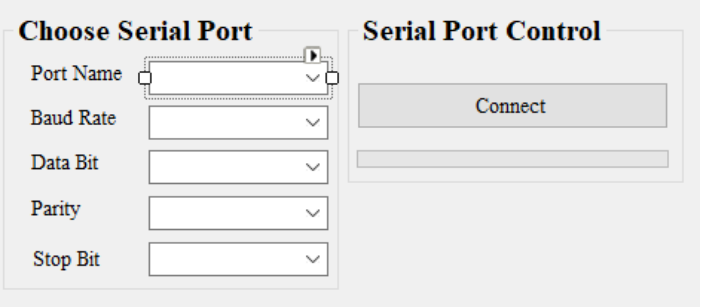
IV. Thiết kế giao diện trên Visual Studio với C#

1. Tổng quan về giao diện

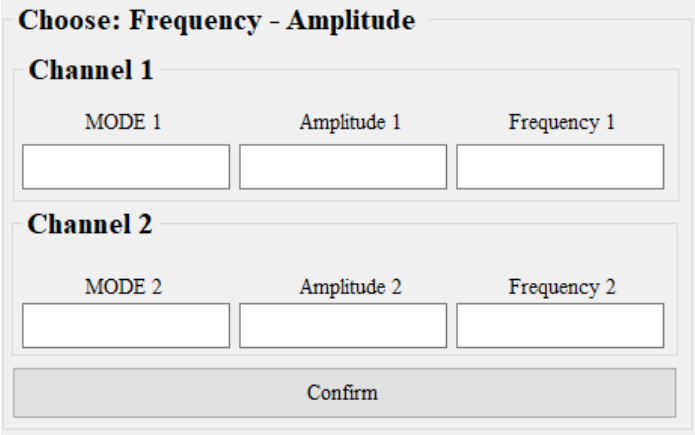


Trong đó:

- Chọn các giá trị Port Name, Baud Rate, Data Bit, Parity, Stop Bit, sau đó nhấn Connect để kết nối UART.

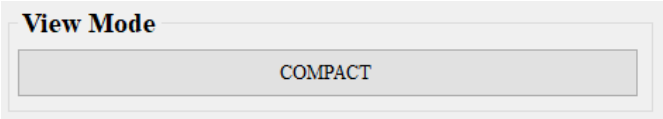


- Thực hiện việc cấu hình cho 2 chanel. Người dùng nhập giá trị biên độ, tần số cũng như Mode cho mỗi Channel, sau đó nhấn Confirm để thấy dạng sóng được vẽ.



MODE 0, 1, 2, 3 tương ứng với sóng sine, sóng vuông, sóng tam giác, sóng răng cưa.

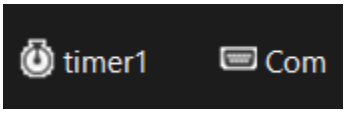
- Chọn chế độ xem.



- Nếu ở chế độ COMPACT thì quan sát dạng sóng từ thời điểm t=0, chế độ SCROLL thì quan sát dạng sóng trong một khoảng thời gian nhất định.

2. Sơ lược về giải thuật lập trình cho giao diện

- Chúng ta cần 1 timer để liệt kê các cổng COM và cổng Serial Port để truyền nhận dữ liệu.



- Cài đặt các thông số bắt đầu như cổng COM, số bit truyền dữ liệu, MODE điều khiển và bảng hiển thị sóng.

```
string[] baud = { "1200", "2400", "4800", "9600", "14400", "19200", "38400", "56000", "57600", "115200" };
string[] databit = { "5", "6", "7", "8" };
private void Form1_Load(object sender, EventArgs e)
{
    GraphPane myPane = zed.GraphPane;
    myPane.Title.Text = "Wave Graph - 2 Channel";
    myPane.XAxis.Title.Text = "Time (s)";
    myPane.YAxis.Title.Text = "Amplitude (V)";
    RollingPointPairList list = new RollingPointPairList(60000);
    RollingPointPairList list1 = new RollingPointPairList(60000);

    LineItem curve = myPane.AddCurve("Channel1", list, Color.Red, SymbolType.None);
    LineItem curve1 = myPane.AddCurve("Channel2", list1, Color.Blue, SymbolType.None);

    myPane.XAxis.Scale.Min = 0;
    myPane.XAxis.Scale.Max = 30;
    myPane.XAxis.Scale.MinorStep = 1;
    myPane.XAxis.Scale.MajorStep = 5;
    zed.AxisChange();
    TickStart = Environment.TickCount;

    cbBaudrate.Items.AddRange(baud);
    cBoxDataBit.Items.AddRange(databit);
    cBoxParity.Items.AddRange(Enum.GetNames(typeof(Parity)));

    Mode1.Text = "0";
    Amplitude1.Text = "003";
    Frequency1.Text = "0000.1";
    Mode2.Text = "0";
    Amplitude2.Text = "003";
    Frequency2.Text = "0000.1";
```

- Thông báo lỗi khi nhập thiếu hay sai dữ liệu bằng các MessageBox.

```
private void btConnect_Click(object sender, EventArgs e)
{
    if (CbSecCom.Text == "")
    {
        MessageBox.Show("Vui long chon cong COM", "Thong Bao", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
    if (cbBaudrate.Text == "")
    {
        MessageBox.Show("Vui long chon Baudrate", "Thong Bao", MessageBoxButtons.OK, MessageBoxIcon.Error);
        return;
    }
```

- Hàm Draw để vẽ đồ thị sử dụng ZedGraph:

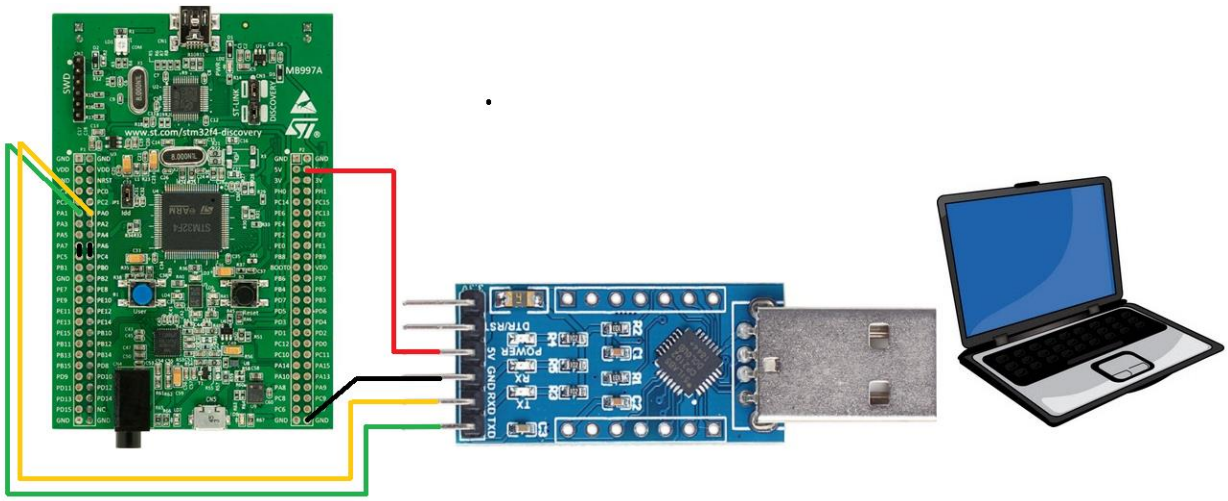
```
1 reference
private void Draw(string channel1, string channel2)
{

    double.TryParse(channel1, out intchannel1);
    double.TryParse(channel2, out intchannel2);

    double.TryParse(Amplitude1.Text, out intAmp1);
    double.TryParse(Amplitude2.Text, out intAmp2);

    intchannel1 = intchannel1 * 3 / 4096;
    intchannel2 = intchannel2 * 3 / 4096;

    if (zed.GraphPane.CurveList.Count <= 0)
        return;

    LineItem curve = zed.GraphPane.CurveList[0] as LineItem;
    LineItem curve1 = zed.GraphPane.CurveList[1] as LineItem;
    if (curve == null)
        return;
    if (curve1 == null)
        return;

    IPointListEdit list = curve.Points as IPointListEdit;
    IPointListEdit list1 = curve1.Points as IPointListEdit;
    if (list == null)
        return;
    if (list1 == null)
        return;
    double time = (Environment.TickCount - TickStart) / 1000.0;

    list.Add(time, intchannel1);
    list1.Add(time, intchannel2);

    Scale xScale = zed.GraphPane.XAxis.Scale;
    if (time > xScale.Max - xScale.MajorStep)
    {
        if (intMode == 1)
        {
            xScale.Max = time + xScale.MajorStep;
            xScale.Min = xScale.Max - 20.0;
        }
        else
        {
            xScale.Max = time + xScale.MajorStep;
            xScale.Min = 0;
        }
    }

    zed.AxisChange();
    zed.Invalidate();
}
```

C. KẾT QUẢ VÀ NHẬN XÉT

I. Kết nối phần cứng

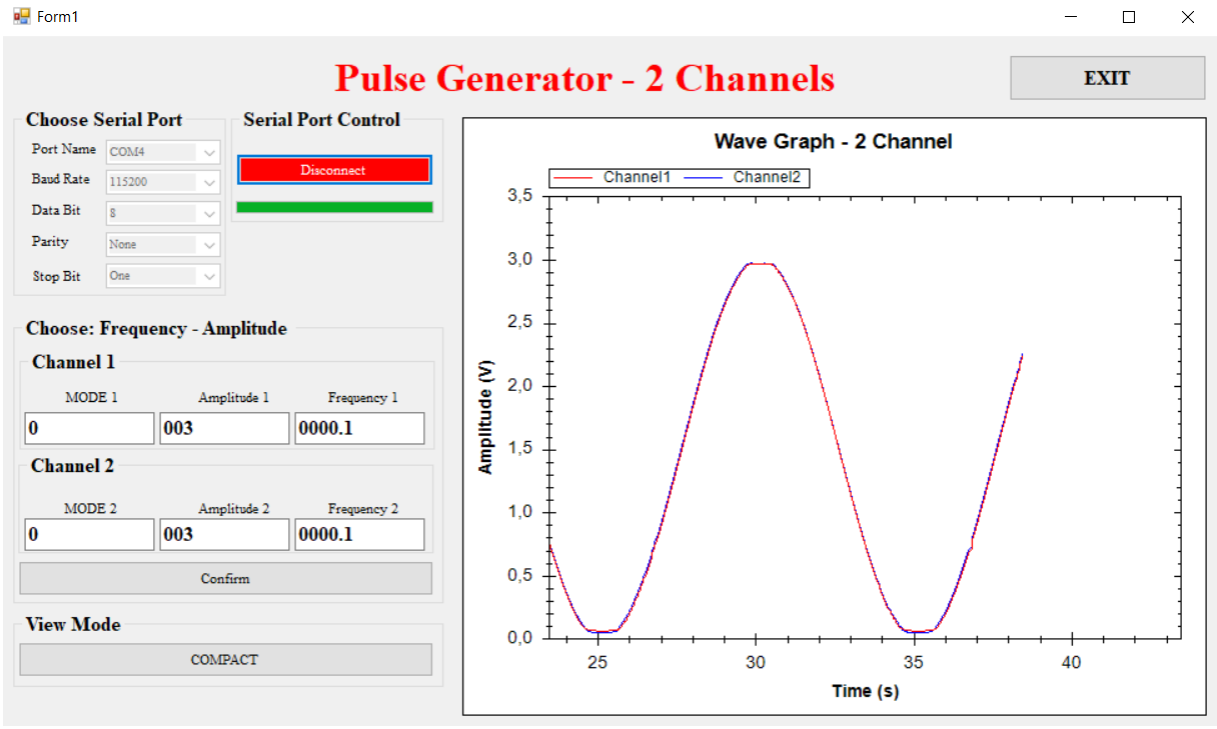
Chân PA4 (DAC1)	⇔	chân PA6 (ADC1)
Chân PA5 (DAC2)	⇔	chân PA7 (ADC2)
Chân TXD của module UART	⇔	chân PA1
Chân RXD của module UART	⇔	chân PA0
Chân 5V của module UART	⇔	chân 5V của STM32F407
Chân GND của module UART	⇔	chân GND của STM32F047

Kết nối cổng USB của modue UART với máy tính

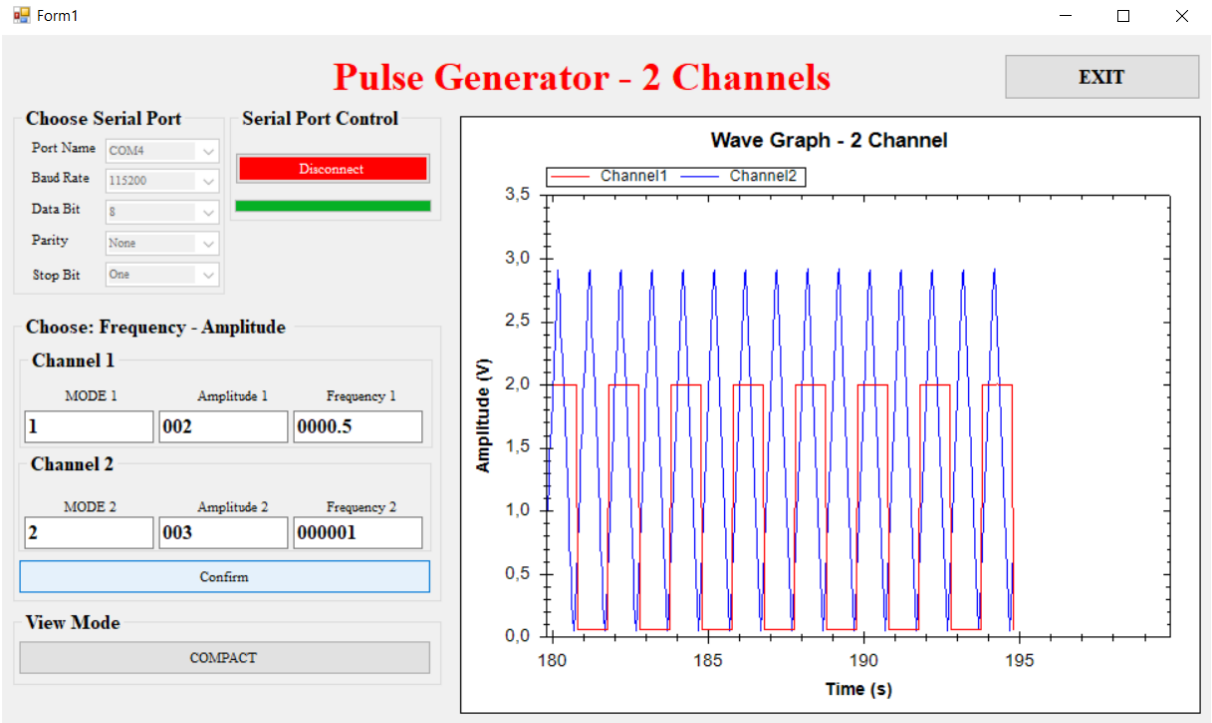


II. Kết quả hiển thị các dạng sóng trên giao diện

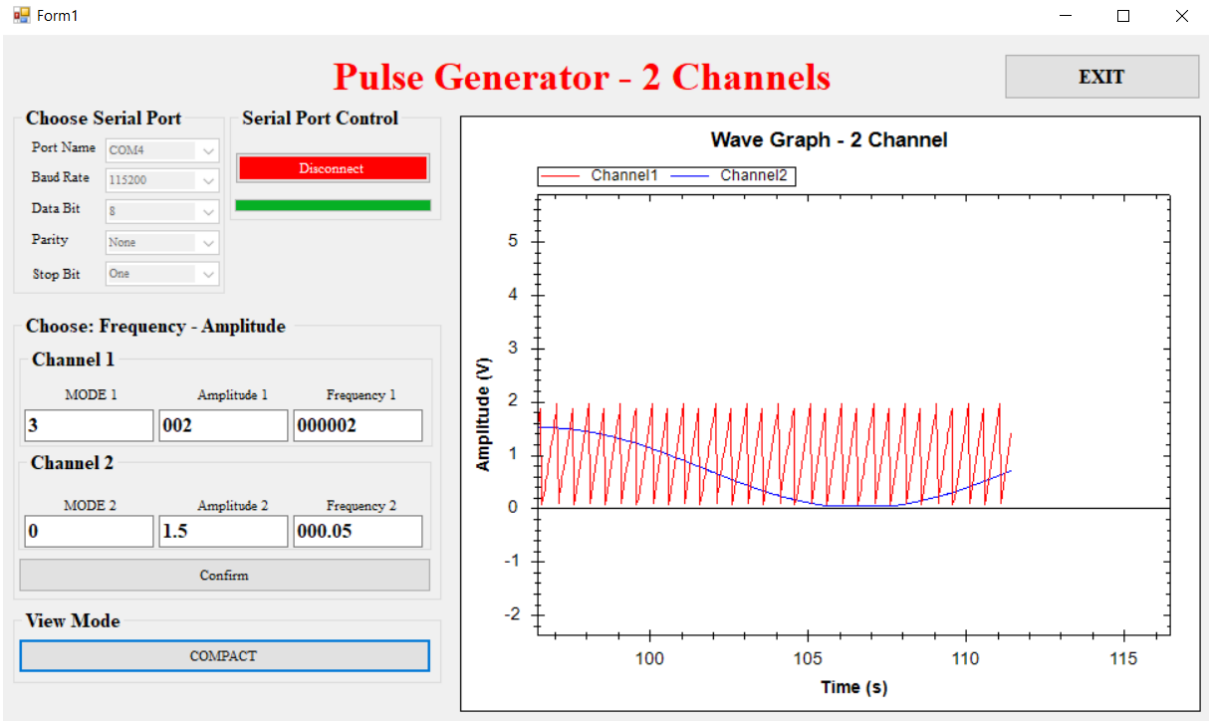
Mode1 = 0 (Sóng sine) A1=3V F1= 0.1Hz hay T1=10s	Mode2 = 0 (Sóng sine) A2=3V F2= 0.1Hz hay T2=10s



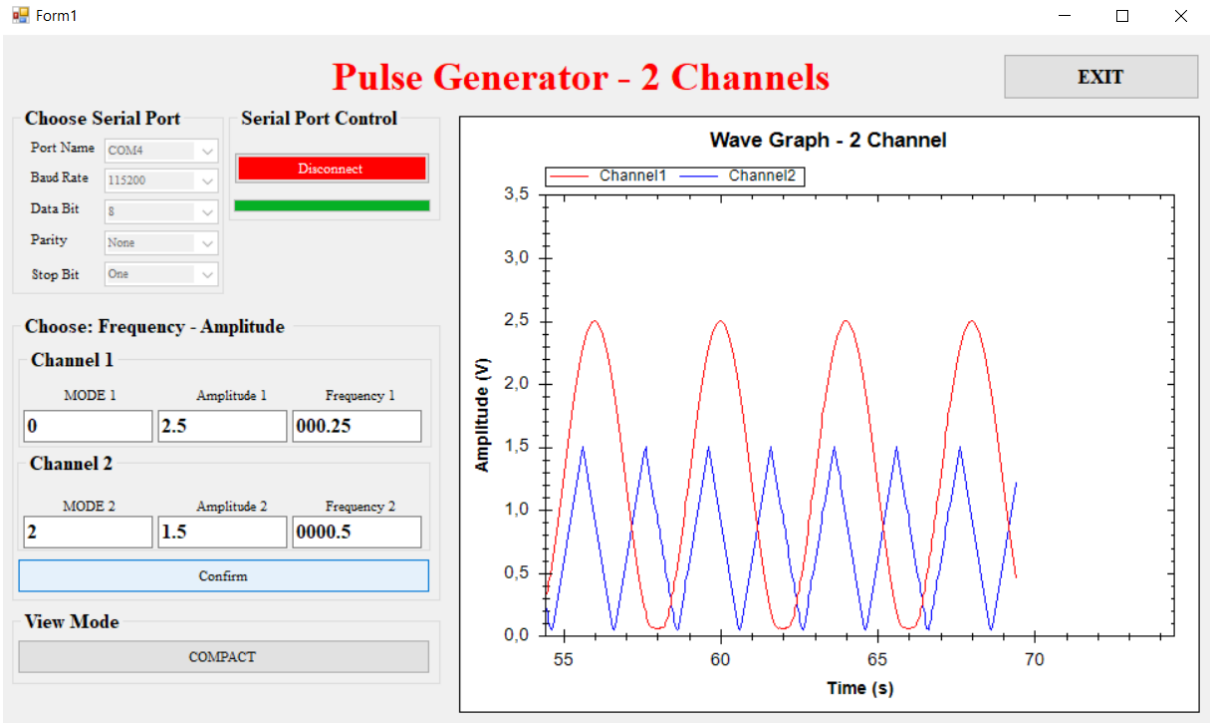
Mode1 = 1 (Sóng vuông) A1=2V F1= 0.5Hz hay T1=2s	Mode2 = 2 (Sóng tam giác) A2=3V F2= 1Hz hay T2=1s



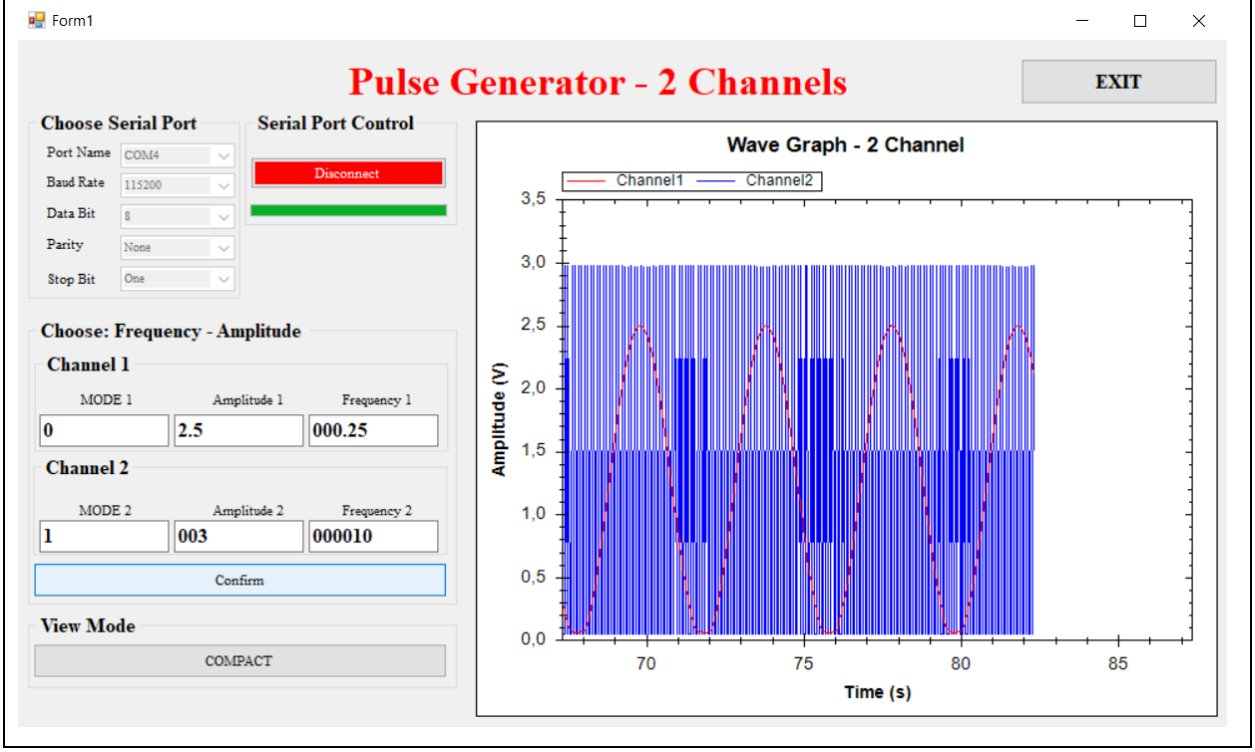
Mode1 = 3 (Sóng răng cưa) A1=2V F1= 2Hz hay T1=0,5s	Mode2 = 0 (Sóng sine) A2=1.5V F2= 0,05Hz hay T2=20s



Mode1 = 0 (Sóng sine) A1=2.5V F1= 0.25Hz hay T1=4s	Mode2 = 2 (Sóng tam giác) A2=1.5V F2= 0.5 hay T2=2s



Mode1 = 0 (Sóng sine) A1=2.5V F1= 0.25Hz hay T1=4s	Mode2 = 1 (Sóng vuông) A2=3V F2= 10 hay T2=0.1s



2. Nhận xét

- Giao diện hiển thị khá tốt về các sóng, đồng thời cho phép người dùng có thể thay đổi giá trị biên độ, tần số theo yêu cầu mong muốn ngay trên giao diện và gửi dữ liệu xuống vi xử lý sau đó đồ thị trên giao diện sẽ được cập nhật rất nhanh.

- Hiển thị chính xác về các thông số biên độ, tần số của các dạng sóng.

- Với các trường hợp tần số lớn (từ 15Hz trở lên) thì dường như sóng không thể quan sát được trên giao diện mà dùng các thiết bị chuyên dụng như Oscilloscope, … Nguyên nhân của việc này có thể là do tốc độ truyền nhận chậm hơn tốc độ tạo tín hiệu.