# 利用DAC在示波器上画图

## 设计思路

X坐标数组通过DMA通道一传输到DAC1通道一转换为模拟电压在PA4引脚输出模拟电压。同理，Y坐标数组通过DMA通道二传输到DAC1通道二转换为模拟电压在PA5引脚输出模拟电压。利用X与Y坐标在示波器的X-Y时基显示模式下绘制二维图像。

示波器检测到此模拟电压后，将绘制图像。

DAC1通道1触发源为定时器6，DAC1通道2触发源为定时器7。定时器6与定时器7频率相同时，x与y坐标变化速度相同达到动态平衡显示静态图形，当定时器6与定时器7频率不相同时，x与y坐标变化速度不相同，产生动态图像。

利用外部中断按键PC13，切换动态显示与静态显示。

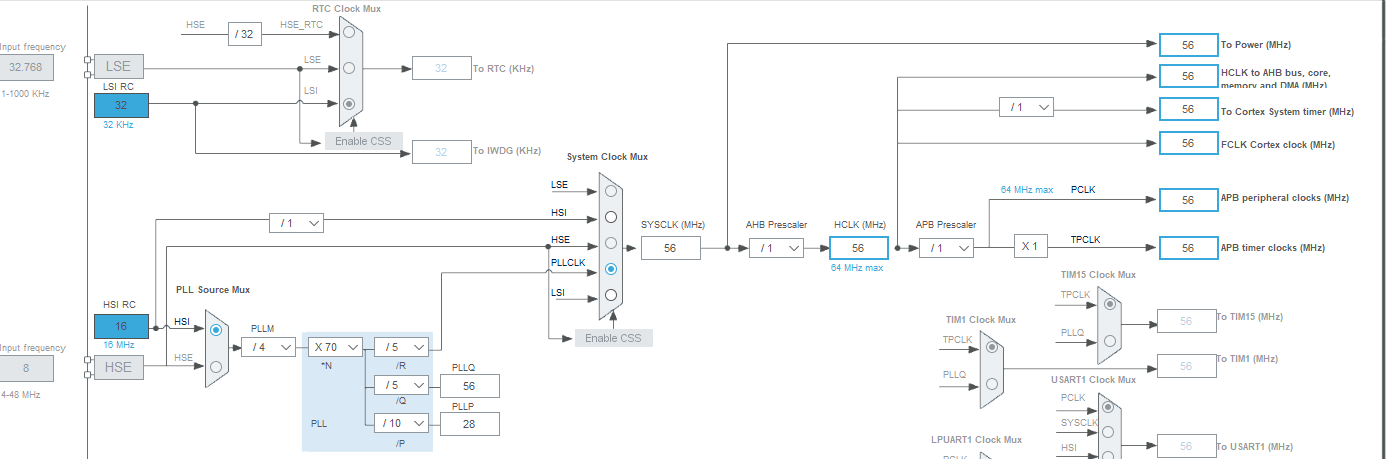
利用宏定义选择X与Y坐标数组的定义，区分绘制圆形与正方形。

## 创建工程

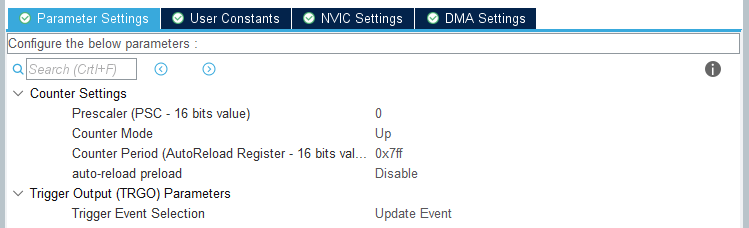
将RCC如下设置更改为“FALSE”

C:\Users\李天凌\AppData\Local\Temp\WeChat Files\dbdb5c068d8385810d84fc24d2edc18.png

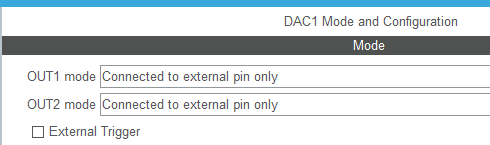
按照下图所示配置时钟



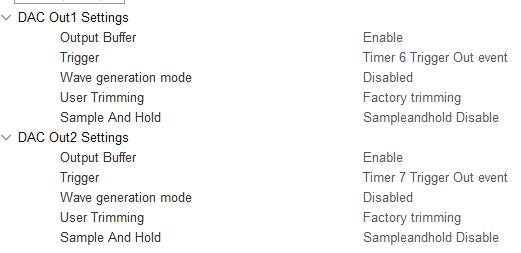
使能定时器6与定时器7配置如下图



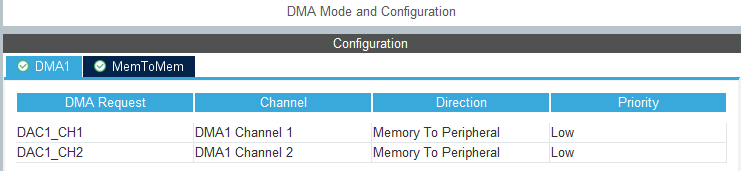
设定DAC1两通道工作模式



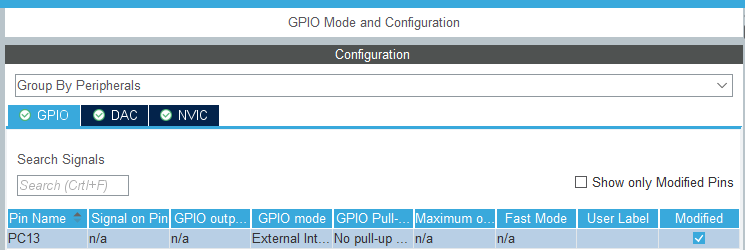
配置DAC两个通道的触发源



启用DMA通道一与通道二进行直接数据传输



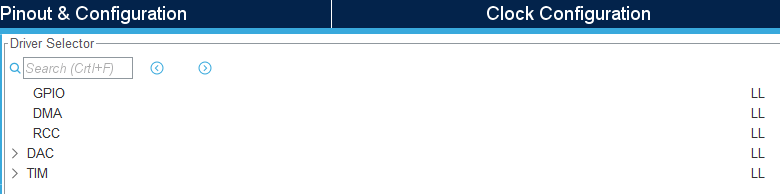
将PC13引脚配置为下降沿触发的外部中断



使能4-15号外部中断

C:\Users\李天凌\AppData\Local\Temp\WeChat Files\7f1b67f781955966835e77e84f8f29c.png

将所有模块改为底层驱动



创建工程，生成代码。

## 代码设计

main.c文件修改为以下内容

|  |
| --- |
| #include "main.h"  //#define CIRCLE //图形选择，注释掉此行，输出即为正方形  int MOVE=0; //定义为0静态图像，定义为1动态图像  #define VDDA\_APPLI ((uint32\_t)3300) /\* 定义环境模拟电压，模拟电压值单位mV \*/  #define DIGITAL\_SCALE\_12BITS (\_\_LL\_DAC\_DIGITAL\_SCALE(LL\_DAC\_RESOLUTION\_12B)) /\* 12位分辨率的满量程数字值，电压范围由模拟电压参考Vref+和Vref-确定。 \*/  /\* 波形生成:波形参数 \*/  #define WAVEFORM\_AMPLITUDE (VDDA\_APPLI) /\* 波形振幅(单位mV) \*/  #define WAVEFORM\_FREQUENCY ((uint32\_t)1000) /\* 波形频率(单位:Hz) \*/  #define WAVEFORM\_SAMPLES\_SIZE (sizeof(WaveformSine\_12bits\_32samples)/ sizeof(uint16\_t)) /\* 包含DAC波形样本的阵列大小 \*/    /\* 波形产生:定时器参数(用作DAC触发器) \*/  /\*定时器频率(单位:Hz)。与计时器16位和时基频率最小1Hz，范围是最小=1Hz，最大=32kHz。\*/  #define WAVEFORM\_TIMER\_FREQUENCY (WAVEFORM\_FREQUENCY \* WAVEFORM\_SAMPLES\_SIZE)  #define WAVEFORM\_TIMER\_FREQUENCY\_RANGE\_MIN ((uint32\_t) 1) /\* 定时器最小频率(单位:Hz)，用于计算频率范围。 使用16位的定时器，最大频率将是这个值的32000倍。 \*/  #define WAVEFORM\_TIMER\_PRESCALER\_MAX\_VALUE ((uint32\_t)0xFFFF-1) /\* 定时器预分频器最大值(0xFFFF为一个定时器16位) \*/  //从数字刻度上的最大值12位(对应于电压Vdda)到新刻度上的值(对应于由WAVEFORM\_AMPLITUDE定义的电压)的计算。  #define \_\_WAVEFORM\_AMPLITUDE\_SCALING(\_\_DATA\_12BITS\_\_) \  (\_\_DATA\_12BITS\_\_ \  \* \_\_LL\_DAC\_CALC\_VOLTAGE\_TO\_DATA(VDDA\_APPLI, WAVEFORM\_AMPLITUDE, LL\_DAC\_RESOLUTION\_12B) \  / \_\_LL\_DAC\_DIGITAL\_SCALE(LL\_DAC\_RESOLUTION\_12B) \  )  #ifdef CIRCLE  //存储正弦信号（作为圆的纵坐标）  const uint16\_t WaveformSine\_12bits\_32samples[] =  {  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2048),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2447),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2831),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3185),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3495),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3750),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3939),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(4056),    \_\_WAVEFORM\_AMPLITUDE\_SCALING(4095),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(4056),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3939),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3750),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3495),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3185),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2831),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2447),    \_\_WAVEFORM\_AMPLITUDE\_SCALING(2048),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1649),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1265),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(911),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(601),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(346),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(157),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(40),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(40),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(157),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(346),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(601),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(911),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1265),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1649)  };  //存储正弦信号（作为圆的横坐标） 数组顺序控制90°相位差  const uint16\_t WaveformSine\_12bits\_32samples1[] =  {  \_\_WAVEFORM\_AMPLITUDE\_SCALING(4095),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(4056),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3939),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3750),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3495),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3185),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2831),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2447),    \_\_WAVEFORM\_AMPLITUDE\_SCALING(2048),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1649),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1265),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(911),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(601),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(346),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(157),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(40),    \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(40),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(157),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(346),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(601),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(911),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1265),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1649),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2048),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2447),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2831),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3185),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3495),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3750),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3939),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(4056)  };  #else  //0-3500 间隔500绘制正方形  const uint16\_t WaveformSine\_12bits\_32samples[] =  {  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),    \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0)  };  const uint16\_t WaveformSine\_12bits\_32samples1[] =  {  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),    \_\_WAVEFORM\_AMPLITUDE\_SCALING(0),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),    \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),    \_\_WAVEFORM\_AMPLITUDE\_SCALING(3500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(3000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(2000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(1000),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(500),  \_\_WAVEFORM\_AMPLITUDE\_SCALING(0)  };  #endif  void SystemClock\_Config(void);  void Configure\_USER\_Interrupt(void);  void Configure\_DMA(void);  void Configure\_TIM\_TimeBase\_DAC\_trigger(void);  void Configure\_DAC(void);  void Activate\_DAC(void);  int main(void)  {  SystemClock\_Config();  Configure\_USER\_Interrupt();  Configure\_DMA(); /\* 为从DAC传输数据配置DMA \*/  Configure\_TIM\_TimeBase\_DAC\_trigger(); /\* 配置定时器作为一个时间基准，用于触发DAC转换的启动 \*/  Configure\_DAC(); /\* 配置DAC通道 \*/  Activate\_DAC(); /\* 使能DAC通道 \*/  while (1)  {  }  }  void SystemClock\_Config(void)  {  LL\_FLASH\_SetLatency(LL\_FLASH\_LATENCY\_2);  /\* HSI configuration and activation \*/  LL\_RCC\_HSI\_Enable();  while(LL\_RCC\_HSI\_IsReady() != 1)  {  }  /\* Main PLL configuration and activation \*/  LL\_RCC\_PLL\_ConfigDomain\_SYS(LL\_RCC\_PLLSOURCE\_HSI, LL\_RCC\_PLLM\_DIV\_4, 70, LL\_RCC\_PLLR\_DIV\_5);  LL\_RCC\_PLL\_Enable();  LL\_RCC\_PLL\_EnableDomain\_SYS();  while(LL\_RCC\_PLL\_IsReady() != 1)  {  }  /\* Sysclk activation on the main PLL \*/  LL\_RCC\_SetSysClkSource(LL\_RCC\_SYS\_CLKSOURCE\_PLL);  while(LL\_RCC\_GetSysClkSource() != LL\_RCC\_SYS\_CLKSOURCE\_STATUS\_PLL)  {  }  /\* Set AHB prescaler\*/  LL\_RCC\_SetAHBPrescaler(LL\_RCC\_SYSCLK\_DIV\_1);  /\* Set APB1 prescaler\*/  LL\_RCC\_SetAPB1Prescaler(LL\_RCC\_APB1\_DIV\_1);  /\* Set systick to 1ms in using frequency set to 56MHz \*/  /\* This frequency can be calculated through LL RCC macro \*/  /\* ex: \_\_LL\_RCC\_CALC\_PLLCLK\_FREQ(\_\_LL\_RCC\_CALC\_HSI\_FREQ(),  LL\_RCC\_PLLM\_DIV\_4, 70, LL\_RCC\_PLLR\_DIV\_5)\*/  LL\_Init1msTick(56000000);  /\* Update CMSIS variable (which can be updated also through SystemCoreClockUpdate function) \*/  LL\_SetSystemCoreClock(56000000);  }  void Configure\_USER\_Interrupt(void)  {  LL\_EXTI\_InitTypeDef EXTI\_InitStruct = {0};  LL\_IOP\_GRP1\_EnableClock(LL\_IOP\_GRP1\_PERIPH\_GPIOC);  LL\_EXTI\_SetEXTISource(LL\_EXTI\_CONFIG\_PORTC, LL\_EXTI\_CONFIG\_LINE13);  EXTI\_InitStruct.Line\_0\_31 = LL\_EXTI\_LINE\_13;  EXTI\_InitStruct.LineCommand = ENABLE;  EXTI\_InitStruct.Mode = LL\_EXTI\_MODE\_IT;  EXTI\_InitStruct.Trigger = LL\_EXTI\_TRIGGER\_RISING;  LL\_EXTI\_Init(&EXTI\_InitStruct);  LL\_GPIO\_SetPinPull(GPIOC, LL\_GPIO\_PIN\_13, LL\_GPIO\_PULL\_NO);  LL\_GPIO\_SetPinMode(GPIOC, LL\_GPIO\_PIN\_13, LL\_GPIO\_MODE\_INPUT);  NVIC\_SetPriority(EXTI4\_15\_IRQn,0);  NVIC\_EnableIRQ(EXTI4\_15\_IRQn);  }  void Configure\_DMA(void)  {  /\* 配置NVIC 以使能DMA中断 \*/  NVIC\_SetPriority(DMA1\_Channel1\_IRQn,2); /\* DMA IRQ优先级低于DAC IRQ \*/  NVIC\_EnableIRQ(DMA1\_Channel1\_IRQn);  NVIC\_SetPriority(DMA1\_Channel2\_3\_IRQn,3); /\* DMA IRQ优先级低于DAC IRQ \*/  NVIC\_EnableIRQ(DMA1\_Channel2\_3\_IRQn);    /\* 使能送给DMA的外部时钟 \*/  LL\_AHB1\_GRP1\_EnableClock(LL\_AHB1\_GRP1\_PERIPH\_DMA1);  /\* 配置DMA传输 \*/  /\* - DMA传输在循环模式下有无限的DAC信号产生。 \*/  /\* - 直接存储器存取转移到DAC没有地址增量。 \*/  /\* - 直接存储器存取转移与地址增量。 \*/  /\* - DMA传输到DAC以半字匹配DAC分辨率12位 \*/  /\* - DMA从存储器通过半字转移到与DAC数据缓冲区匹配的变量类型:半字。 \*/  LL\_DMA\_ConfigTransfer(DMA1,LL\_DMA\_CHANNEL\_1,LL\_DMA\_DIRECTION\_MEMORY\_TO\_PERIPH | LL\_DMA\_MODE\_CIRCULAR |  LL\_DMA\_PERIPH\_NOINCREMENT | LL\_DMA\_MEMORY\_INCREMENT | LL\_DMA\_PDATAALIGN\_HALFWORD |  LL\_DMA\_MDATAALIGN\_HALFWORD | LL\_DMA\_PRIORITY\_HIGH );  LL\_DMA\_ConfigTransfer(DMA1,LL\_DMA\_CHANNEL\_2,LL\_DMA\_DIRECTION\_MEMORY\_TO\_PERIPH | LL\_DMA\_MODE\_CIRCULAR |  LL\_DMA\_PERIPH\_NOINCREMENT | LL\_DMA\_MEMORY\_INCREMENT | LL\_DMA\_PDATAALIGN\_HALFWORD |  LL\_DMA\_MDATAALIGN\_HALFWORD | LL\_DMA\_PRIORITY\_HIGH );    /\* 选择DAC作为DMA传输请求 \*/  LL\_DMA\_SetPeriphRequest(DMA1,LL\_DMA\_CHANNEL\_1,LL\_DMAMUX\_REQ\_DAC1\_CH1);  LL\_DMA\_SetPeriphRequest(DMA1,LL\_DMA\_CHANNEL\_2,LL\_DMAMUX\_REQ\_DAC1\_CH2);    /\* 设置源和目的的DMA传输地址 \*/  LL\_DMA\_ConfigAddresses(DMA1,LL\_DMA\_CHANNEL\_1,(uint32\_t)&WaveformSine\_12bits\_32samples,LL\_DAC\_DMA\_GetRegAddr(DAC1, LL\_DAC\_CHANNEL\_1,  LL\_DAC\_DMA\_REG\_DATA\_12BITS\_RIGHT\_ALIGNED),LL\_DMA\_DIRECTION\_MEMORY\_TO\_PERIPH);  LL\_DMA\_ConfigAddresses(DMA1,LL\_DMA\_CHANNEL\_2,(uint32\_t)&WaveformSine\_12bits\_32samples1,LL\_DAC\_DMA\_GetRegAddr(DAC1, LL\_DAC\_CHANNEL\_2,  LL\_DAC\_DMA\_REG\_DATA\_12BITS\_RIGHT\_ALIGNED),LL\_DMA\_DIRECTION\_MEMORY\_TO\_PERIPH);  /\* 设置DMA传输大小 \*/  LL\_DMA\_SetDataLength(DMA1,LL\_DMA\_CHANNEL\_1,WAVEFORM\_SAMPLES\_SIZE);  LL\_DMA\_SetDataLength(DMA1,LL\_DMA\_CHANNEL\_2,WAVEFORM\_SAMPLES\_SIZE);    /\* 使能DMA传输中断: 传输错误 \*/  LL\_DMA\_EnableIT\_TE(DMA1,LL\_DMA\_CHANNEL\_1);  LL\_DMA\_EnableIT\_TE(DMA1,LL\_DMA\_CHANNEL\_2);    /\* 注: 在本例中，唯一激活的DMA中断是传输错误。如果需要，DMA中断传输的一半和传输完成可以被激活。参考DMA示例。 \*/  /\* 启用DMA传输 \*/  LL\_DMA\_EnableChannel(DMA1,LL\_DMA\_CHANNEL\_1);  LL\_DMA\_EnableChannel(DMA1,LL\_DMA\_CHANNEL\_2);  }  //动态显示  void Configure\_TIM\_TimeBase\_DAC\_trigger(void)  {  uint32\_t timer\_clock\_frequency = 0; /\* 定时器的时钟频率 \*/  uint32\_t timer\_prescaler = 0; /\* 时基预分频器，使时基按可能的最小频率对齐 \*/  uint32\_t timer\_reload = 0; /\* 定时器预分频器功能中的定时器重新加载值，以实现时间基准周期 \*/  if (LL\_RCC\_GetAPB1Prescaler() == LL\_RCC\_APB1\_DIV\_1)  {  timer\_clock\_frequency = \_\_LL\_RCC\_CALC\_PCLK1\_FREQ(SystemCoreClock, LL\_RCC\_GetAPB1Prescaler());  }  else  {  timer\_clock\_frequency = (\_\_LL\_RCC\_CALC\_PCLK1\_FREQ(SystemCoreClock, LL\_RCC\_GetAPB1Prescaler()) \* 2);  }  /\* 计时器预分频器计算 \*/  /\* (computation for timer 16 bits, additional + 1 to round the prescaler up) \*/  timer\_prescaler = ((timer\_clock\_frequency / (WAVEFORM\_TIMER\_PRESCALER\_MAX\_VALUE \* WAVEFORM\_TIMER\_FREQUENCY\_RANGE\_MIN)) +1);  /\* 定时器重载计算 \*/  timer\_reload = (timer\_clock\_frequency / (timer\_prescaler \* WAVEFORM\_TIMER\_FREQUENCY));  /\* 使能定时器外围时钟 \*/  LL\_APB1\_GRP1\_EnableClock(LL\_APB1\_GRP1\_PERIPH\_TIM6);  LL\_APB1\_GRP1\_EnableClock(LL\_APB1\_GRP1\_PERIPH\_TIM7);  /\* 设置定时器预分频器值 \*/  LL\_TIM\_SetPrescaler(TIM6, (timer\_prescaler - 1));  LL\_TIM\_SetPrescaler(TIM7, (timer\_prescaler - 1)+MOVE);//频率差造成动态图像  /\* 设置定时器自动重载值 \*/  LL\_TIM\_SetAutoReload(TIM6, (timer\_reload - 1));  LL\_TIM\_SetAutoReload(TIM7, (timer\_reload - 1));  /\* 设置当前模式 \*/  LL\_TIM\_SetCounterMode(TIM6, LL\_TIM\_COUNTERMODE\_UP);  LL\_TIM\_SetCounterMode(TIM7, LL\_TIM\_COUNTERMODE\_UP);  /\* 注: 在本例中，计时器中断没有被激活。如果需要，在每个时间基准周期上定时器中断是可能的。 \*/  /\* 设置定时器触发输出 (TRGO) \*/  LL\_TIM\_SetTriggerOutput(TIM6, LL\_TIM\_TRGO\_UPDATE);  LL\_TIM\_SetTriggerOutput(TIM7, LL\_TIM\_TRGO\_UPDATE);  /\* 使能计数器 \*/  LL\_TIM\_EnableCounter(TIM6);  LL\_TIM\_EnableCounter(TIM7);  }  void Configure\_DAC(void)  {  /\* 使能DAC相关的GPIO时钟\*/  LL\_IOP\_GRP1\_EnableClock(LL\_IOP\_GRP1\_PERIPH\_GPIOA);  /\* 将GPIO配置在模拟模式作为DAC的输出端 \*/  LL\_GPIO\_SetPinMode(GPIOA, LL\_GPIO\_PIN\_4, LL\_GPIO\_MODE\_ANALOG);  LL\_GPIO\_SetPinMode(GPIOA, LL\_GPIO\_PIN\_5, LL\_GPIO\_MODE\_ANALOG);  /\* 配置NVIC使能DAC1中断 \*/  NVIC\_SetPriority(TIM6\_DAC\_LPTIM1\_IRQn, 0);  NVIC\_EnableIRQ(TIM6\_DAC\_LPTIM1\_IRQn);  NVIC\_SetPriority(TIM7\_LPTIM2\_IRQn, 1);  NVIC\_EnableIRQ(TIM7\_LPTIM2\_IRQn);  /\* 使能DAC时钟 \*/  LL\_APB1\_GRP1\_EnableClock(LL\_APB1\_GRP1\_PERIPH\_DAC1);  /\* 选择触发源 \*/  LL\_DAC\_SetTriggerSource(DAC1, LL\_DAC\_CHANNEL\_1, LL\_DAC\_TRIG\_EXT\_TIM6\_TRGO);  LL\_DAC\_SetTriggerSource(DAC1, LL\_DAC\_CHANNEL\_2, LL\_DAC\_TRIG\_EXT\_TIM7\_TRGO);  /\* 设置DAC通道1的输出 \*/  LL\_DAC\_ConfigOutput(DAC1, LL\_DAC\_CHANNEL\_1, LL\_DAC\_OUTPUT\_MODE\_NORMAL, LL\_DAC\_OUTPUT\_BUFFER\_ENABLE, LL\_DAC\_OUTPUT\_CONNECT\_GPIO);  LL\_DAC\_ConfigOutput(DAC1, LL\_DAC\_CHANNEL\_2, LL\_DAC\_OUTPUT\_MODE\_NORMAL, LL\_DAC\_OUTPUT\_BUFFER\_ENABLE, LL\_DAC\_OUTPUT\_CONNECT\_GPIO);  /\* 启用DAC通道DMA请求 \*/  LL\_DAC\_EnableDMAReq(DAC1, LL\_DAC\_CHANNEL\_1);  LL\_DAC\_EnableDMAReq(DAC1, LL\_DAC\_CHANNEL\_2);  /\* 使能DAC通道一中断 \*/  LL\_DAC\_EnableIT\_DMAUDR1(DAC1);  }  void Activate\_DAC(void)  {  \_\_IO uint32\_t wait\_loop\_index = 0;  /\* 使能DAC通道1 \*/  LL\_DAC\_Enable(DAC1, LL\_DAC\_CHANNEL\_1);  LL\_DAC\_Enable(DAC1, LL\_DAC\_CHANNEL\_2);    wait\_loop\_index = ((LL\_DAC\_DELAY\_STARTUP\_VOLTAGE\_SETTLING\_US \* (SystemCoreClock / (100000 \* 2))) / 10);  while(wait\_loop\_index != 0)  {  wait\_loop\_index--;  }  LL\_DAC\_EnableTrigger(DAC1, LL\_DAC\_CHANNEL\_1);  LL\_DAC\_EnableTrigger(DAC1, LL\_DAC\_CHANNEL\_2);  }    void USER\_Interrupt\_CallBack(void)  {  MOVE = ( MOVE + 1 ) % 2;  Configure\_TIM\_TimeBase\_DAC\_trigger();  }  #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 can add his own implementation to report the file name and line number,  ex: printf("Wrong parameters value: file %s on line %d", file, line) \*/  /\* Infinite loop \*/  while (1)  {  }  }  #endif |

在stm32g0xx\_it.c文件下，在与外部中断相关的中断服务程序中调用自定义中断回调函数。

|  |
| --- |
| void EXTI4\_15\_IRQHandler(void)  {  if (LL\_EXTI\_IsActiveRisingFlag\_0\_31(LL\_EXTI\_LINE\_13) != RESET)  {  LL\_EXTI\_ClearRisingFlag\_0\_31(LL\_EXTI\_LINE\_13);  USER\_Interrupt\_CallBack();  }  } |

最后编译运行即可

## 结果展示

测量PA4与PA5引脚信号可得出以下结论。

