DAC，COMP，HRTIM Fault 功能的使用

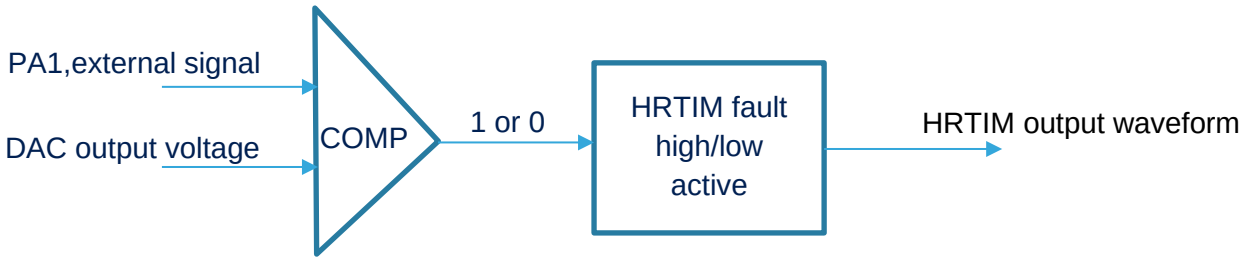
关键字：HRTIM，Fault，DAC，COMP

1. 引言

这个例程是使用 STM32G474 NUCLEO 进行测试的，集合了 DAC， COMP， HRTIM 的功能模块。

2. 信号路径

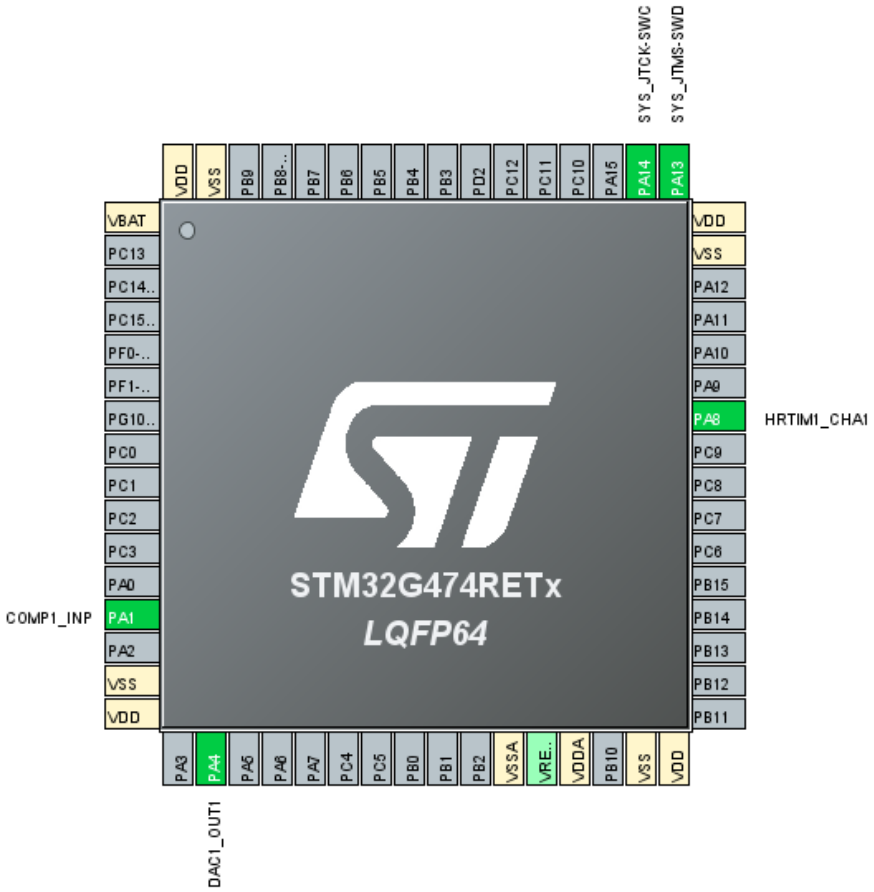
PA1 的输入信号同 DAC 的输出信号进入 COMP 比较器进行比较，经 COMP 比较后的输出信号连接到 HRTIM 的 Fault4 信号，来控制 HRTIM 的输出信号的停止。



当 PA1 > DAC value,比较器输出高“1”， 这个信号为 HRTIM 的 fault4 信号，当 HRTIM 配置为 fault4 高有效，则当这个高电平出现时，HRTIM 输出波形停止，具体输出的电平可以通过寄存器进行配置。

3. STM32CubeMX 配置

PA1 的输入信号同 DAC 的输出信号进入 COMP 比较器进行比较，经 COMP 比较后的输出信号连接到 HRTIM 的 Fault4 信号，来控制 HRTIM 的输出信号的停止。



首先配置 DAC 模块，DAC out1 选择 connected to on chip-peripherals, 这里为了测试方便，选择了“Connected to external pin and to on chip-peripherals”，DAC 输出信号连接到 IC 内部，同时输出到 PA4 GPIO 口。这样就客户测量 DAC 的具体输出值。

而在 IC 内部，DAC 连接到了 COMP 的负端。

COMP 的配置如下：

PA1：COPM 的正向输入端。

DAC1 OUT1 在 IC 内部连接到了 COMP 的负向输入端。

具体请看下面两种图：

图 1. DAC 配置

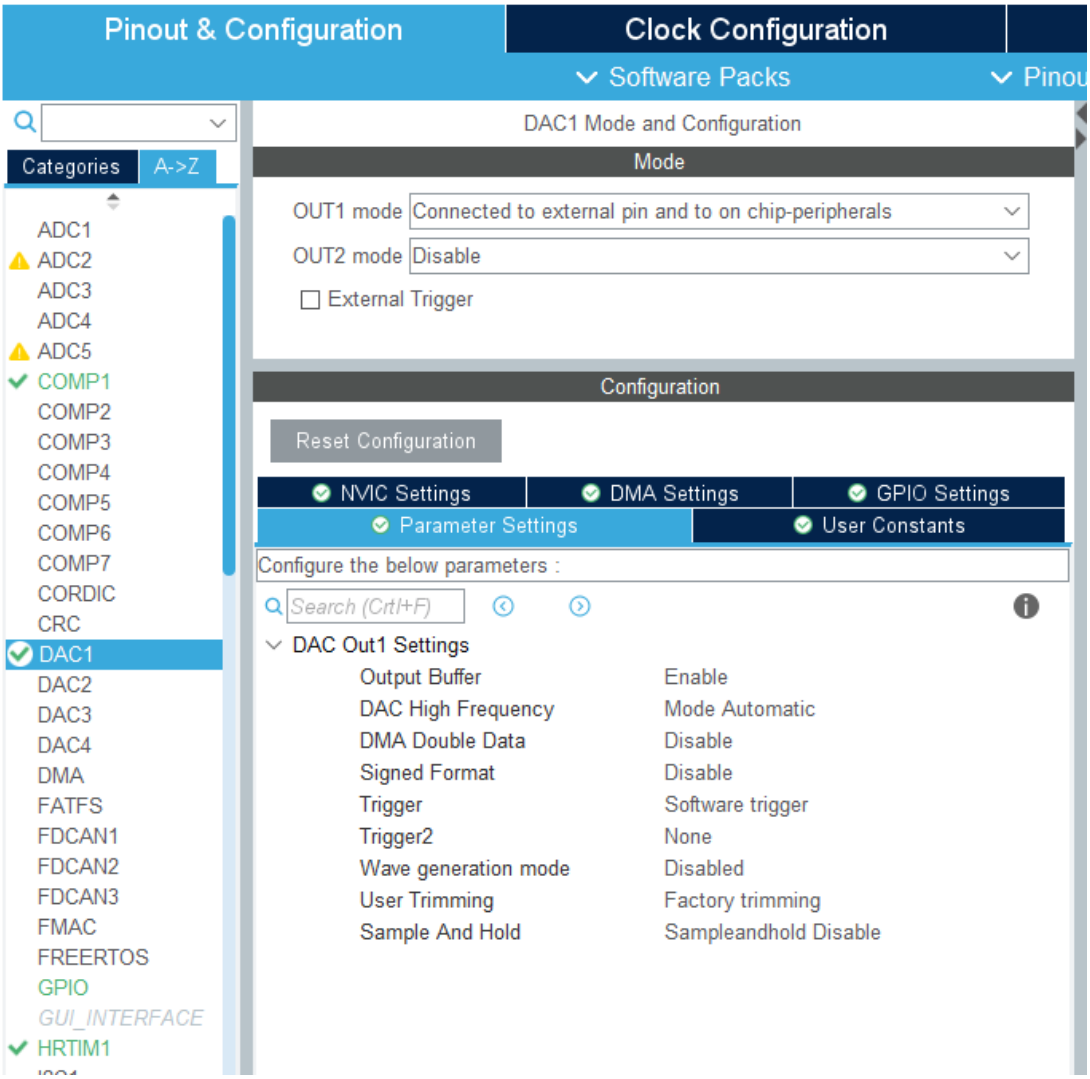
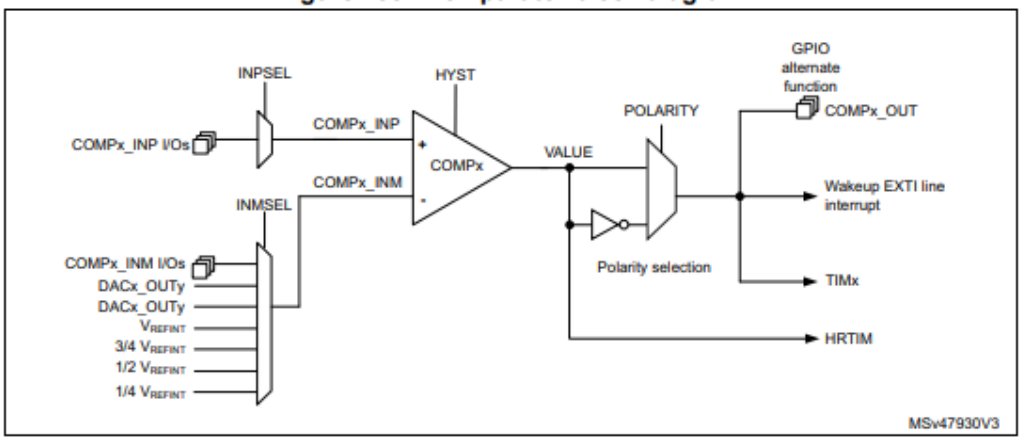
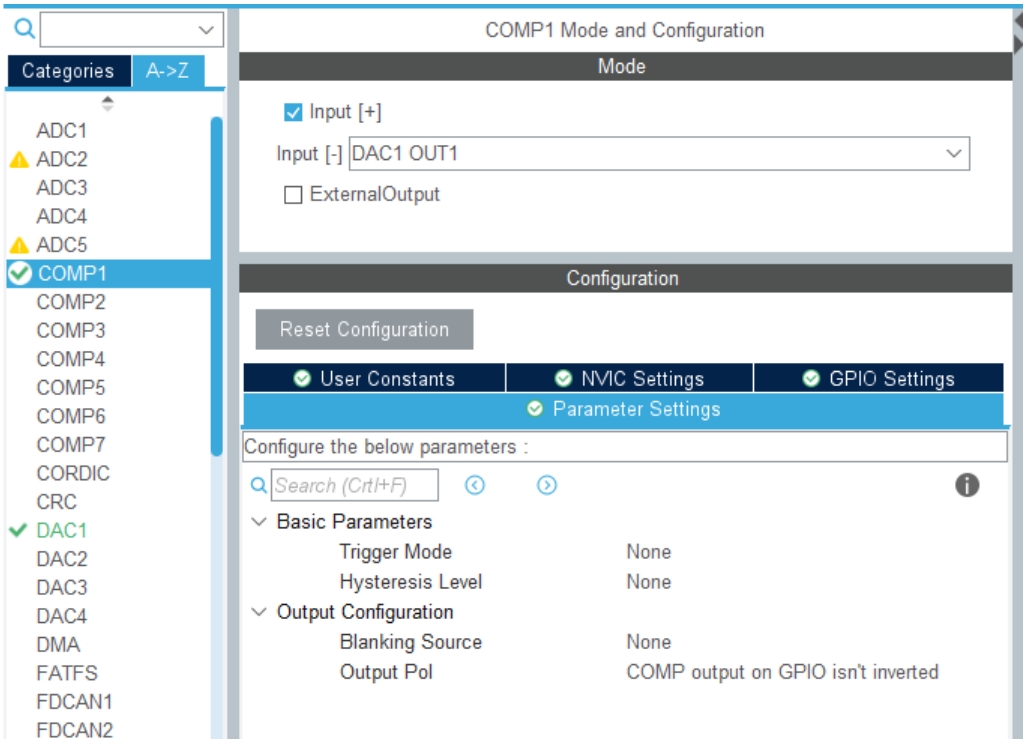


图 2. COMP 框图和配置

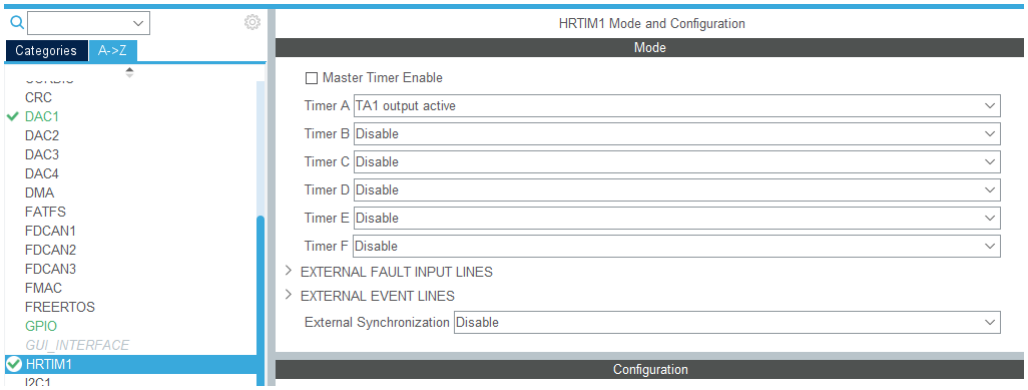
Figure 168. Comparator block diagram



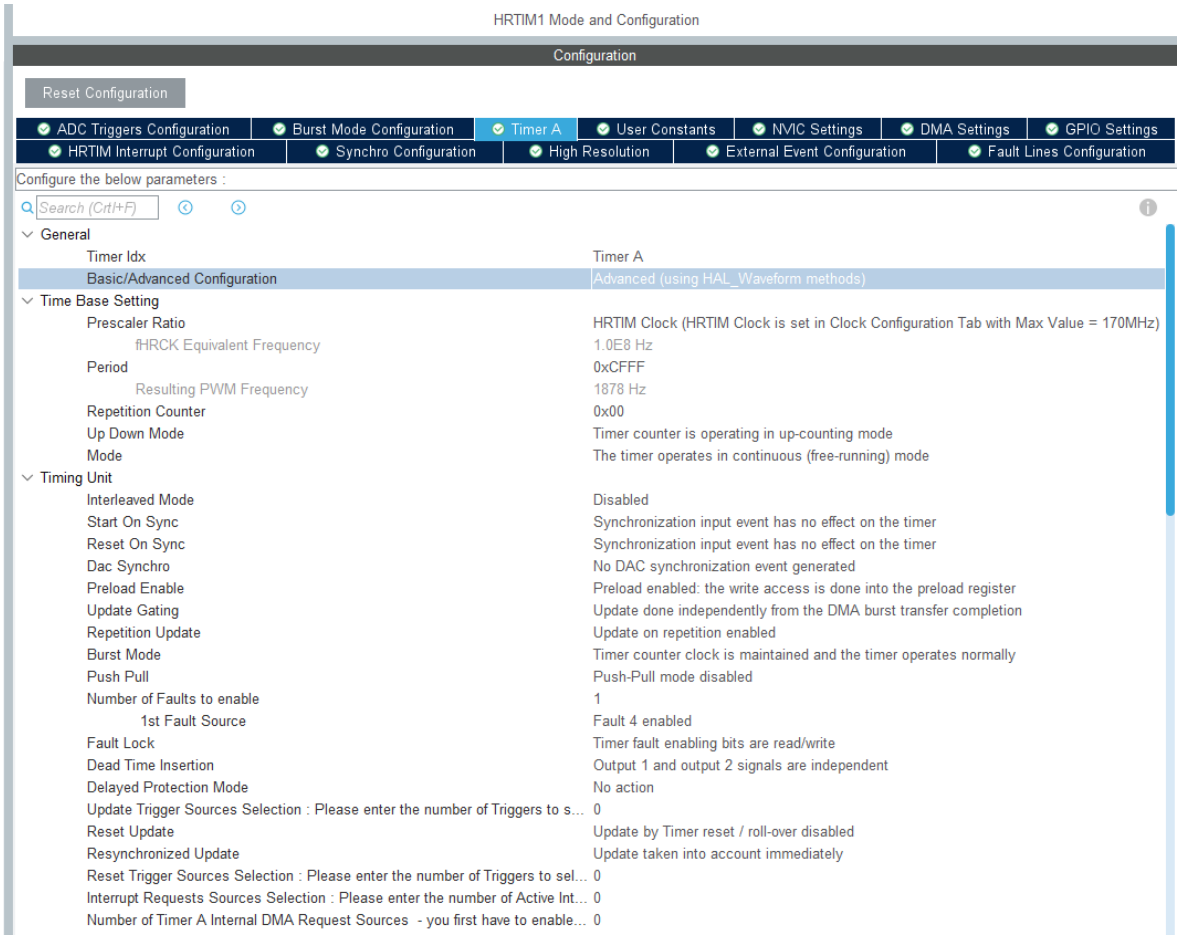


最后配置 HRTIM 模块：
使能 TimerA 的 TA1 输出。

图 3. HRTIM 配置

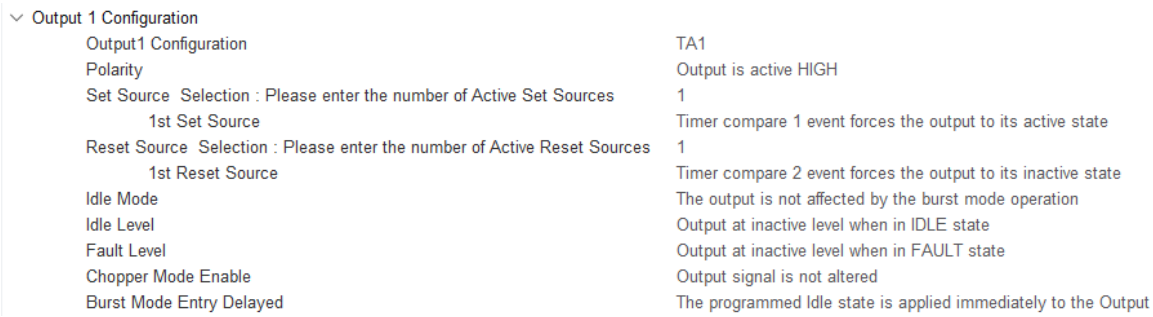


配置 TIMA 周期值为 0XCFFF，向上计数模式，持续计数方式,Fault4 作 fault 源。

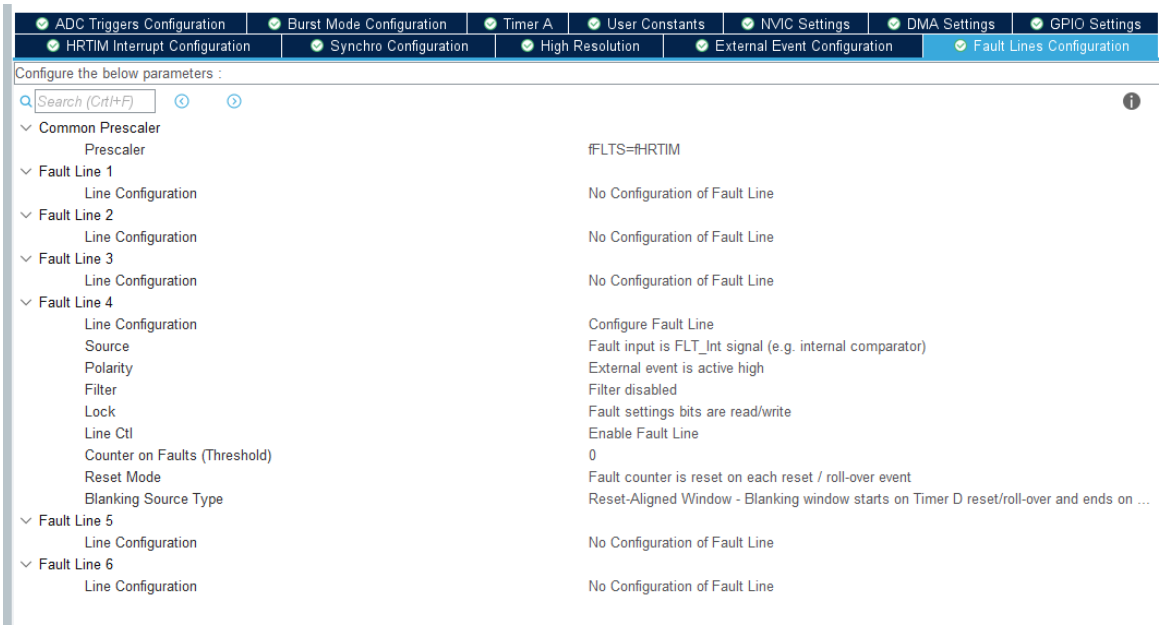


再使能两个比较器 compare1 和 compare2，分别配置为 0x2FFF 和 0x8FFF 以生成比较事件：

Compare1 事件之后 HRTIM 输出高电平，compare2 事件之后输出低电平：

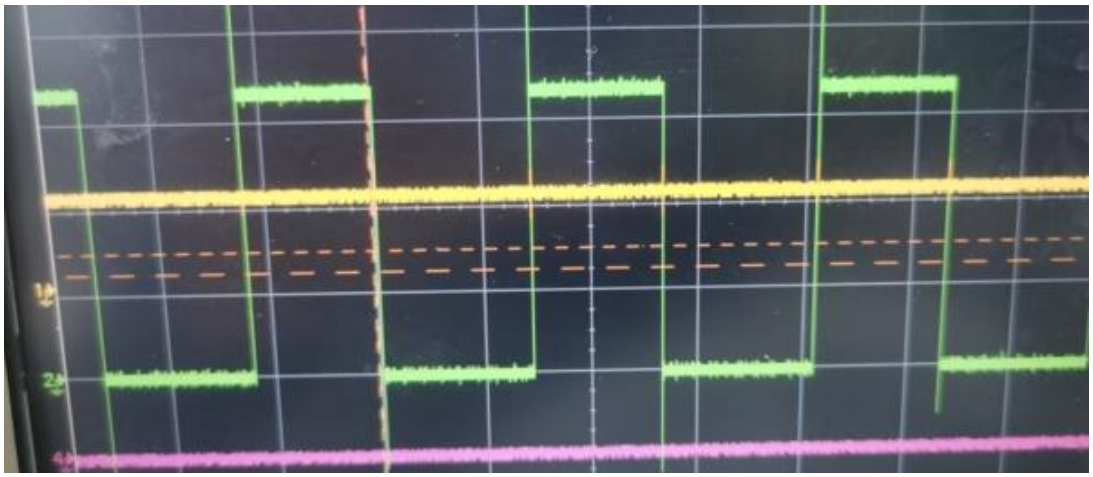


下面是 Fault4 的配置，事件配置成高有效：



HRTIM 输出波形:

下面的方波信号为 HRTIM 的输出波形，由于 compare1 和 compare2 的比较值是固定的，并没有在 Timer、 DMA 等模块的控制下进行更改，因此下面的波形周期和占空比是一定的。



4. 代码实现

下面是具体实现代码，先初始化 DAC，然后是 COMP，等 COMP 输出信号稳定后再使能 HRTIM 模块。

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_DAC1_Init();
  if (HAL_DAC_SetValue(&hdac1, DAC_CHANNEL_1, DAC_ALIGN_12B_R,0x500) != HAL_OK
  { Error_Handler();  }
  if (HAL_DAC_Start(&hdac1, DAC_CHANNEL_1) != HAL_OK)
  {    Error_Handler();  }
  MX_COMP1_Init();
  HAL_Delay(1000);
  MX_HRTIM1_Init();
  if(HAL_COMP_Start(&hcomp1) != HAL_OK)
  {    Error_Handler();   }
  HAL_HRTIM_WaveformOutputStart(&hhrtim1, HRTIM_OUTPUT_TA1 );
  HAL_HRTIM_WaveformCounterStart(&hhrtim1, HRTIM_TIMERID_TIMER_A);
  while (1)
  {  }
}
```

```
static void MX_COMP1_Init(void)
{
  hcomp1.Instance = COMP1;
  hcomp1.Init.InputPlus = COMP_INPUT_PLUS_IO1;
  hcomp1.Init.InputMinus = COMP_INPUT_MINUS_DAC1_CH1;
  hcomp1.Init.OutputPol = COMP_OUTPUTPOL_NONINVERTED;
  hcomp1.Init.Hysteresis = COMP_HYSTERESIS_NONE;
  hcomp1.Init.BlankingSrce = COMP_BLANKINGSRC_NONE;
  hcomp1.Init.TriggerMode = COMP_TRIGGERMODE_NONE;
  if (HAL_COMP_Init(&hcomp1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

```
static void MX_DAC1_Init(void)
{
  DAC_ChannelConfTypeDef sConfig = {0};
  hdac1.Instance = DAC1;
  if (HAL_DAC_Init(&hdac1) != HAL_OK)
  {
    Error_Handler();
  }
  sConfig.DAC_HighFrequency = DAC_HIGH_FREQUENCY_INTERFACE_MODE_AUTOMATIC;
  sConfig.DAC_DMADoubleDataMode = DISABLE;
  sConfig.DAC_SignedFormat = DISABLE;
  sConfig.DAC_SampleAndHold = DAC_SAMPLEANDHOLD_DISABLE;
  sConfig.DAC_Trigger = DAC_TRIGGER_SOFTWARE;
  sConfig.DAC_Trigger2 = DAC_TRIGGER_NONE;
  sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
  sConfig.DAC_ConnectOnChipPeripheral = DAC_CHIPCONNECT_BOTH;
  sConfig.DAC_UserTrimming = DAC_TRIMMING_FACTORY;
  if (HAL_DAC_ConfigChannel(&hdac1, &sConfig, DAC_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
}
```

```
static void MX_HRTIM1_Init(void)
{
  HRTIM_FaultBlankingCfgTypeDef pFaultBlkCfg = {0};
  HRTIM_FaultCfgTypeDef pFaultCfg = {0};
  HRTIM_TimeBaseCfgTypeDef pTimeBaseCfg = {0};
  HRTIM_TimerCtlTypeDef pTimerCtl = {0};
  HRTIM_TimerCfgTypeDef pTimerCfg = {0};
  HRTIM_CompareCfgTypeDef pCompareCfg = {0};
  HRTIM_OutputCfgTypeDef pOutputCfg = {0};

  hhrtim1.Instance = HRTIM1;
  hhrtim1.Init.HRTIMInterruptResquests = HRTIM_IT_FLT4;
  hhrtim1.Init.SyncOptions = HRTIM_SYNCOPTION_NONE;
  if (HAL_HRTIM_Init(&hhrtim1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_HRTIM_DLLCalibrationStart(&hhrtim1, HRTIM_CALIBRATIONRATE_3) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_HRTIM_PollForDLLCalibration(&hhrtim1, 10) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_HRTIM_FaultPrescalerConfig(&hhrtim1, HRTIM_FAULTPRESCALER_DIV1) != HAL_OK)
  {
    Error_Handler();
  }
  pFaultBlkCfg.Threshold = 0;
  pFaultBlkCfg.ResetMode = HRTIM_FAULTCOUNTERRST_UNCONDITIONAL;
  pFaultBlkCfg.BlankingSource = HRTIM_FAULTBLANKINGMODE_RSTALIGNED;
  if (HAL_HRTIM_FaultCounterConfig(&hhrtim1, HRTIM_FAULT_4, &pFaultBlkCfg) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_HRTIM_FaultBlankingConfigAndEnable(&hhrtim1, HRTIM_FAULT_4, &pFaultBlkCfg) != HAL_OK)
  {
    Error_Handler();
  }
  pFaultCfg.Source = HRTIM_FAULTSOURCE_INTERNAL;
  pFaultCfg.Polarity = HRTIM_FAULTPOLARITY_LOW;
  pFaultCfg.Filter = HRTIM_FAULTFILTER_NONE;
  pFaultCfg.Lock = HRTIM_FAULTLOCK_READWRITE;
  if (HAL_HRTIM_FaultConfig(&hhrtim1, HRTIM_FAULT_4, &pFaultCfg) != HAL_OK)
  {
    Error_Handler();
  }

  HAL_HRTIM_FaultModeCtl(&hhrtim1, HRTIM_FAULT_4, HRTIM_FAULTMODECTL_ENABLED);
  pTimeBaseCfg.Period = 0xCFFF;
  pTimeBaseCfg.RepetitionCounter = 0x00;
  pTimeBaseCfg.PrescalerRatio = HRTIM_PRESCALERRATIO_DIV1;
  pTimeBaseCfg.Mode = HRTIM_MODE_CONTINUOUS;
  if (HAL_HRTIM_TimeBaseConfig(&hhrtim1, HRTIM_TIMERINDEX_TIMER_A, &pTimeBaseCfg) != HAL_OK)
  {
    Error_Handler();
  }
  pTimerCtl.UpDownMode = HRTIM_TIMERUPDOWNMODE_UP;
  pTimerCtl.TrigHalf = HRTIM_TIMERTRIGHALF_DISABLED;
  pTimerCtl.GreaterCMP1 = HRTIM_TIMERGTCMP1_EQUAL;
  pTimerCtl.DualChannelDacEnable = HRTIM_TIMER_DCDE_DISABLED;
  if (HAL_HRTIM_WaveformTimerControl(&hhrtim1, HRTIM_TIMERINDEX_TIMER_A, &pTimerCtl) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  pTimerCfg.InterruptRequests = HRTIM_TIM_IT_NONE;
  pTimerCfg.DMARequests = HRTIM_TIM_DMA_NONE;
  pTimerCfg.DMASrcAddress = 0x0000;
  pTimerCfg.DMADstAddress = 0x0000;
  pTimerCfg.DMASize = 0x1;
  pTimerCfg.HalfModeEnable = HRTIM_HALFMODE_DISABLED;
  pTimerCfg.InterleavedMode = HRTIM_INTERLEAVED_MODE_DISABLED;
  pTimerCfg.StartOnSync = HRTIM_SYNCSTART_DISABLED;
  pTimerCfg.ResetOnSync = HRTIM_SYNCRESET_DISABLED;
  pTimerCfg.DACSynchro = HRTIM_DACSYNC_NONE;
  pTimerCfg.PreloadEnable = HRTIM_PRELOAD_ENABLED;
  pTimerCfg.UpdateGating = HRTIM_UPDATEGATING_INDEPENDENT;
  pTimerCfg.BurstMode = HRTIM_TIMERBURSTMODE_MAINTAINCLOCK;
  pTimerCfg.RepetitionUpdate = HRTIM_UPDATEONREPETITION_ENABLED;
  pTimerCfg.PushPull = HRTIM_TIMPUSHPULLMODE_DISABLED;
 pTimerCfg.FaultEnable = HRTIM_TIMFAULTENABLE_FAULT4;
  pTimerCfg.FaultLock = HRTIM_TIMFAULTLOCK_READWRITE;
  pTimerCfg.DeadTimeInsertion = HRTIM_TIMDEADTIMEINSERTION_DISABLED;
  pTimerCfg.DelayedProtectionMode = HRTIM_TIMER_A_B_C_DELAYEDPROTECTION_DISABLED;
  pTimerCfg.UpdateTrigger = HRTIM_TIMUPDATETRIGGER_NONE;
  pTimerCfg.ResetTrigger = HRTIM_TIMRESETTRIGGER_NONE;
  pTimerCfg.ResetUpdate = HRTIM_TIMUPDATEONRESET_ENABLED;
  pTimerCfg.ReSyncUpdate = HRTIM_TIMERESYNC_UPDATE_UNCONDITIONAL;
  if (HAL_HRTIM_WaveformTimerConfig(&hhrtim1, HRTIM_TIMERINDEX_TIMER_A, &pTimerCfg) != HAL_OK)
  {
    Error_Handler();
  }
  pCompareCfg.CompareValue = 0x2FFF;
  if (HAL_HRTIM_WaveformCompareConfig(&hhrtim1, HRTIM_TIMERINDEX_TIMER_A, HRTIM_COMPAREUNIT_1, &pCompareCfg) != HAL_OK)
  {
    Error_Handler();
  }
  pCompareCfg.CompareValue = 0x8FFD;
  pCompareCfg.AutoDelayedMode = HRTIM_AUTODELAYEDMODE_REGULAR;
  pCompareCfg.AutoDelayedTimeout = 0x0000;

  if (HAL_HRTIM_WaveformCompareConfig(&hhrtim1, HRTIM_TIMERINDEX_TIMER_A, HRTIM_COMPAREUNIT_2, &pCompareCfg) != HAL_OK)
  {
    Error_Handler();
  }
  pOutputCfg.Polarity = HRTIM_OUTPUTPOLARITY_HIGH;
  pOutputCfg.SetSource = HRTIM_OUTPUTSET_TIMCMP1;
  pOutputCfg.ResetSource = HRTIM_OUTPUTRESET_TIMCMP2;
  pOutputCfg.IdleMode = HRTIM_OUTPUTIDLEMODE_NONE;
  pOutputCfg.IdleLevel = HRTIM_OUTPUTIDLELEVEL_INACTIVE;
  pOutputCfg.FaultLevel = HRTIM_OUTPUTFAULTLEVEL_INACTIVE;
  pOutputCfg.ChopperModeEnable = HRTIM_OUTPUTCHOPPERMODE_DISABLED;
  pOutputCfg.BurstModeEntryDelayed = HRTIM_OUTPUTBURSTMODEENTRY_REGULAR;
  if (HAL_HRTIM_WaveformOutputConfig(&hhrtim1, HRTIM_TIMERINDEX_TIMER_A, HRTIM_OUTPUT_TA1, &pOutputCfg) != HAL_OK)
  {
    Error_Handler();
  }
  HAL_HRTIM_MspPostInit(&hhrtim1);

}
```

5. 小结

STM32G474 在 IC 内部有很多内部连接通路，可以根据具体应用灵活进行配置。在上面的应用中 DAC 的输出信号连接到 COMP 比较器的 INM 输入口，COMP 的输出信号连接到 HRTIM 的 Fault4 信号，来控制 HRTIM 的输出停止。

版本历史

日期	版本	变更
2022 年 10 月 17 日	1.0	首版发布

重要通知 – 请仔细阅读

意法半导体公司及其子公司 （“ST”）保留随时对 ST 产品和 / 或本文档进行变更的权利，恕不另行通知。买方在订货之前应获取关于 ST 产品的最新信息。 ST 产品的销售依照订单确认时的相关 ST 销售条款。

买方自行负责对 ST 产品的选择和使用， ST 概不承担与应用协助或买方产品设计相关的任何责任。

ST 不对任何知识产权进行任何明示或默示的授权或许可。

转售的 ST 产品如有不同于此处提供的信息的规定，将导致 ST 针对该产品授予的任何保证失效。

ST 和 ST 徽标是 ST 的商标。若需 ST 商标的更多信息，请参考 *www.st.com/trademarks*。所有其他产品或服务名称均为其 各自所有者的财产。

本文档是 ST 中国本地团队的技术性文章，旨在交流与分享，并期望借此给予客户产品应用上足够的帮助或提醒。若文中内容存有局限或与 ST 官网资料不一致，请以实际应用验证结果和 ST 官网最新发布的内容为准。您拥有完全自主权是否采纳本文档（包括代码，电路图等）信息，我们也不承担因使用或采纳本文档内容而导致的任何风险。

本文档中的信息取代本文档所有早期版本中提供的信息。