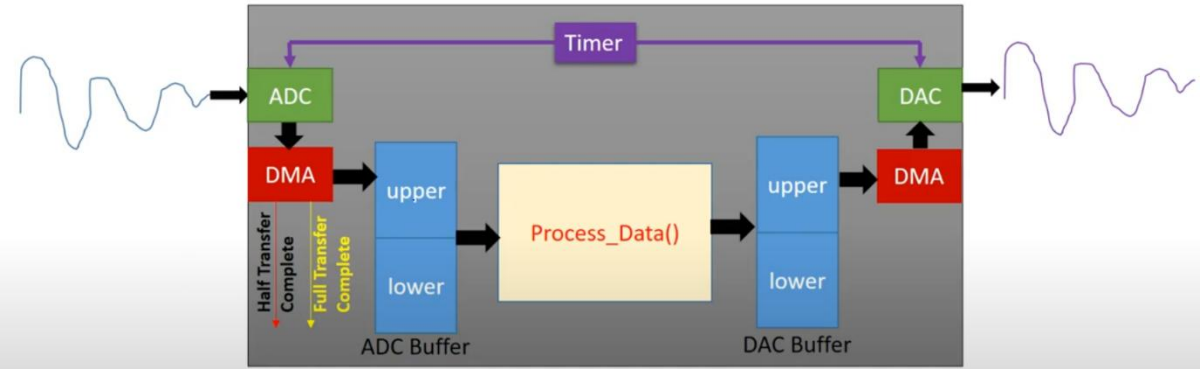
ADC to DAC Via DMA:

Discovery Board: STM32F407G-DISC1

Framework: STM32CubeIDE

Connection: Connection Channel 1 of Oscilloscope to DAC pin(PA4) of board, GND also.
Give a Square wave of 3v and 1kHz to ADC pin(PA2) of board.

Block Diagram:



https://www.youtube.com/watch?v=ThFfI-JSv2Y&ab_channel=ENGRTUTOR

Configuration in Cube IDE:

1. In a new project, first configure “SYS” pins for debugging mode on the chip, “Serial Wire”
2. Synchronous Timer Configuration for both DAC and ADC: TIM2
   a) Clock Source: Internal Clock
   b) Parameter Settings:
      i. Prescaler = 0
      ii. Auto-reload Register: TIMclk / ((ARR + 1)*(prescalar + 1)) = SamplingFreq(like 16kHz in our case) = 5249

TIMclk(APBx Timer Clock(MHz)): as tim2 on APB1 bus, Ref: DS8626_STM32F40xxx.book data Sheet of chip.

iii. Trigger out set to “Update Event”, as trigger event every time the timer counter reaches ARR.

iv. Why Tim2: because its option availability on the ADC as “External Trigger Conversion Source”.

3. ADC1: Channel 2 on PA2 on board

a) Parameter Settings:

i. Scan Conversion : Disabled because of only 1 channel, if more than 1 it would set automatically enabled.

ii. DMA Continuous Request: enabled , because if not then it only run the ADC runs only 1 time and when call back function for DMA call only 1 time then it execute only 1 time and stop getting values on ADC.

iii. External Trigger Source as Timer 2 Trig Out Event.

iv. No of Conversion : 1 because of using only 1 ADC.

v. Rank is user for multiple ADCs and there respective priorities with the Sampling Time in Cycles(low cycle is use for fast Data conversion Time b/c of low impedance where input signal can quickly store the internal Capacitor of ADC.

b) DMA Settings:

i. Adds a new to ADC1 as Peripheral to Memory.

ii. Mode: Circular

iii. Data Width: Half Word(16 bits / 2 bytes) which is suitable for Audio Data processing.

c) NYIC Settings:

i. Check the DMA global Interrupt which will be automatically marked.

4. DAC: Channel 1 Configuration

a) Parameter Settings:

i. Output Buffer: Enable

ii. Trigger: Again Timer 2 Trig Out Event

b) NYIC & DMA Settings:

i. Same as above for ADC.

Main.c File:

Initializing the Tim2, and ADC and DAC starting the DMA Function before entering the main loop, which connects with gets the data and store it in memory using DMA.

```
  /* USER CODE BEGIN 2 */
  //Start time base and DMA channels ================================================
//HAL_ADCEx_Calibration_Start(&hadc1);
  HAL_TIM_Base_Start(&htim2);
  HAL_ADC_Start_DMA(&hadc1, (uint32_t*) adc_val, FULLBUFFERSIZE);
  HAL_DAC_Start_DMA(&hdac, DAC_CHANNEL_1, (uint32_t*) dac_val, FULLBUFFERSIZE, DAC_ALIGN_12B_R);

  /* USER CODE END 2 */
```

Also, defining the buffers for ADC and DAC using double buffering methods, which explains well in the project: https://www.youtube.com/watch?v=zlGSxZGwj-E&ab_channel=Phil%E2%80%99sLab

```
/* USER CODE BEGIN PV */
// # of samples user accesses per data block
#define DATASIZE 128
//#define DATASIZE 4
// full adc and dac buffer size
#define FULLBUFFERSIZE 256
//#define FULLBUFFERSIZE 8
// create buffers for adc and dac
volatile uint32_t adc_val[FULLBUFFERSIZE];
volatile uint32_t dac_val[FULLBUFFERSIZE];

// Pointers to half buffer
static volatile uint32_t* inbufptr;
static volatile uint32_t* outbufptr = &dac_val[0];

uint8_t dataReadyFlag;

/* USER CODE END PV */
```

Under the Driver(HAL), ..hal_adc.c gives two function of half and complete call back function of ADC buffer of two interrupts(Half transfer completer and Full Transfer complete), where on we points to the memory location of buffers of adc and dac for sending in the data and moving out respectively.

As we have started the adc and dac at the same time by using the trigger from the same clock source, so we assume that data transfer will be on almost same location that is why the pointer are either pointing to the start or midpoint in the same time, using half or full complete callBack functions, as:

```
void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef *hadc)
{
  // first half of adc buffer is now full
    inbufptr = &adc_val[0];
    outbufptr = &dac_val[0];
    processDSP();
    dataReadyFlag = 1;
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
    inbufptr = &adc_val[FULLBUFFERSIZE/2];
    outbufptr = &dac_val[FULLBUFFERSIZE/2];
    processDSP();
    dataReadyFlag = 1;
}
```

One more thing here, using of flags to start the process function(which just giving data from one memory location to another or in future it could be like filtration or FAT File handling System) which needs to be occurred in the main processor,

```
// placeholder for a dsp function block
void processDSP()
{
    //static int in = 0;
    //static int out = 0;
    for (int n = 0; n < (FULLBUFFERSIZE/2); n++)
    {
        outbufptr[n] = inbufptr[n];
        //outbufptr[n] = n + 1;

    }

    dataReadyFlag = 0;
}
```

But using it in the main processor, it don't just provide our desired signal to process, that is why we used it in out call back functions which emerges from interrupts, as shown above.

```
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
//    if(dataReadyFlag)
      {
//        processDSP();
      }
  }
  /* USER CODE END 3 */
}
```

.