EED308 **Embedded System Hardware** **Spring 2021**

Experiment 7 Interrupts, Timers, DAC and ADC in STM32

The objective of this experiment is to get familiar with the use of the following features of a microcontroller: **Interrupt**, **Timer**, **DAC** and **ADC**. A triangular waveform having frequency adjustable through a Timer will be generated at the output of a DAC through a programme triggered by an Interrupt from a Timer. Another Timer will then be used to sample the triangular waveform and an ADC used to convert these samples back to Digital. Configurable interconnections among the microcontroller peripherals will be utilised to set up the whole system.

Part A: Generation of Triangular Waveform by DAC triggered by a Timer

A triangular waveform can be generated by a **DAC** by applying to the input of the **DAC** a repetitive sequence of numbers that increase linearly from zero up to a prescribed maximum value and then decreasing linearly back to zero. As the microcontroller has a built-in reference voltage V_{REF+} = 3.3V, the analog output voltage V_A is given in terms of the digital 12-bit input N_D by the relationship $V_A = 3.3N_D / 4096$. So if N_D is incremented in 16 steps of 256 (=100_H) each from an initial value 000_H, and then decremented in 16 steps of the same step size, V_A will go up linearly from 0V to 3.3V in 16 steps, and then go down linearly to 0V in 16 steps. This cycle can be repeated endlessly to generate a triangular wave having a peak value 3.3V. The time period of the triangular wave will then be 32 times the time taken for each step in the **DAC** input sequence, which is given by the time interval between successive Update Event (UEV) outputs from the Timer configured in the basic Time Base mode. The Frequency of UEV output is given by:

$$f_{UEV} = f_{PSC} /\{(\text{PSC} + 1)\ (\text{ARR} + 1)\ (\text{RCR} + 1)\},$$

where PSC, ARR and RCR denote the values stored in the Pre-Scaler Counter, Auto-Reload Register and Repeat Counter Register respectively.

1. Set up the Timer **TIM3** in the basic Time Base mode with an 8MHz Clock input and with PSC = 79 and RCR = 0 to generate Interrupts. The Timer would then generate **Interrupts** at time intervals $T_I = 10(N_A + 1)$ μs, where N_A is the ARR setting.
2. Write an **ISR** to generate a periodic **DAC** input going up from 0 to 4096 in 16 equal steps, and then coming down from 4096 to 0 in 16 equal steps again, with step size of 256. [Ref. Sections 16.1 – 16.5 of RM0316 for the STM32F303 MCU]
3. Display, using the built-in Serial Wire Viewer (SWV), the **DAC** output as a **Curve** on your laptop screen for N_A = 4, 9, 19 and 39. Note how the shape of the displayed waveform is affected by the time interval between consecutive Interrupts, as decided by the value of N_A. How would you estimate the time taken by the ISR to generate successive points on the DAC output?

Part B: Sampling the Triangular Waveform by ADC triggered by a Timer

1. Set the **DAC** output for a time period of 6.4ms (N_A = 19), and connect it to an **ADC** input. Use **TIM1** to trigger the **ADC** at intervals of 0.8ms by suitably choosing the Timer parameters, using the internal connection between **TIM1** channel outputs and **ADC** trigger inputs, as given in the interconnections among peripherals.

   [Ref. Sections 15.3.13 and 15.3.18 of RM0316 for the STM32F303 MCU.]
2. Tabulate the values of the **ADC** output in Hex format for two complete cycles of the triangular waveform, along with the values given by the triangular **DAC** output waveform. Compare the corresponding values and comment on the error.
3. Repeat step **B.2** for N_A = 9 and 39. Comment on the effect on the error.

STM32CubeIDE Configuration Steps for DAC with TIM3

1. Select RCC from the pinout pane, and choose
   a. High Speed Clock (HSE): Crystal/Ceramic Resonator.
2. Select TIM3 and TIM1 from the pinout pane, and make the following choice:
   a. Clock Source: Internal Clock.
3. Select DAC1 from the pinout pane:
   a. Select OUT1 Configuration.
4. Select ADC1 from the pinout pane, and select the following choice:
   a. IN1: Single-ended
5. From the Clock Configuration Set the Timer Clock to the 8 MHz.
   a. Set automatically by setting the Timer peripheral clock 8Mhz and enter
6. Set the following parameters for TIM3 from the configuration pane:
   Parameter Settings: Pre-scalar = 79, Counter Mode UP, Counter Period (ARR) = 19.
   a. NVIC Settings: Check the TIM3 Global Interrupt enable.
7. Set the following parameters for DAC1 from the configuration pane:
   a. Parameter Settings: Check the output buffer to enable.

STM32CubeIDE Configuration Steps for ADC with TIM1

8. Set the following parameters for TIM1 from the configuration pane:
   Parameter Settings: Pre-scalar = 79, Counter Mode UP, Counter Period (ARR) = 79.
   Trigger Event Selection TRGO to Update Event
9. Set the following parameters for ADC1 from the configuration pane:
   Under ADC Regular Conversion Mode
   a. External Trigger Conversion Source: Timer 1 Trigger out event
   b. NVIC Settings: Enable ADC1 and ADC2 interrupts

****

10. Generate the C code corresponds to the above configurations and insert User codes as follows.
    (The parameter values have to be entered as required).
    a. Define the required Header files.
    ```
    #include "main.h"
    #include "stm32f3xx_hal.h"
    ```
    b. Define Private Variables for DAC output, ADC output.
    ```
        // write your Private variables ---------------------------------------------------------*/
    ```
    c. On top of the Main function i.e., under User code **Begin 0**.
    ```
    /* USER CODE BEGIN 0 */
    //Code for DAC
    void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
    {
                    if(htim->Instance == TIM3)
                    {
                    // Write your Triangular wave Generation code here by using appropriate variables
                    }
    HAL_DAC_SetValue(&hdac1,DAC_CHANNEL_1,DAC_ALIGN_12B_R,dacInput);
    HAL_DAC_Start(&hdac1,DAC_CHANNEL_1);
    dacOutput = (3.3*dacInput)/4096; // This is the DAC output in terms of voltages: 0 to 3.3v
    /*Since the datatype of last argument of HAL_DAC_SetValue is "uint32_t", we have to convert the analog
       value (0 to 3.3v) into digital value (0 to 4096) */
    }
    }

    //Code for ADC
    void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
    {
    if (hadc->Instance==ADC1)
    {
    HAL_ADC_PollForConversion(&hadc1,100);
    adcOutput = HAL_ADC_GetValue(&hadc1);
    }
    }
     /* USER CODE END 0 */
    ```
    d. Under user code **Begin 2** of Main function
    ```
    HAL_TIM_Base_Start_IT(&htim3); // This will create the events periodically for DAC
    HAL_TIM_Base_Start(&htim1); // This will trigger the ADC
    HAL_ADC_Start_IT(&hadc1); // Starting the ADC
    ```
    e. Debug the written program and show the generated triangular waveform on "SWV data trace timeline graph".
11. Now compile the code and connect the header wire between the DAC output and ADC output to generate the ADC output waveform. Compare the two waveforms and comment.