Embedded Systems Bare-Metal Programming Ground Up (STM 32)

AIM: To set up a timer with a frequency of 100Hz. When the timer overflows, trigger the ADC to read an analog voltage value on a GPIO pin. Then, transfer the ADC results via DMA to RAM and trigger an interrupt. In the IRQ handler, send the above result to UART.

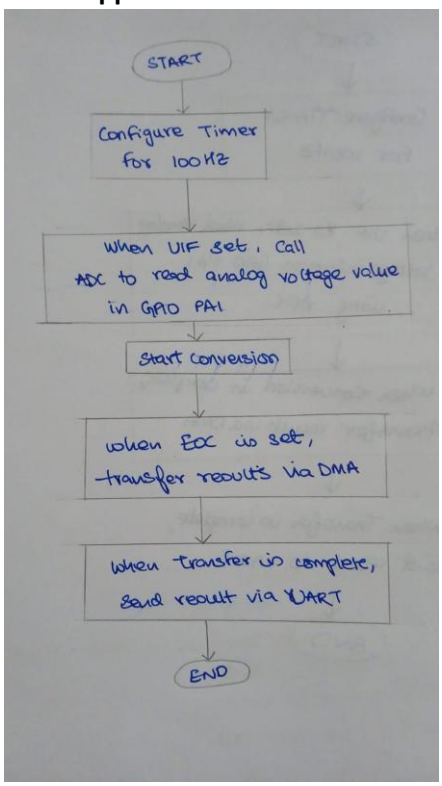
Hardware Required:

STM32F411CEU6 BlackPill Board, STLink v2 Debugger, CP2102 TTL to USB Converter

Software Required:

STM32CubeIDE, RealTerm

Initial Approach:



What I was doing wrong:

-Start ADC in timer ISR
-Wait for ADC inside ISR
-Configure DMA inside ISR
-Poll EOC
-Block inside interrupt

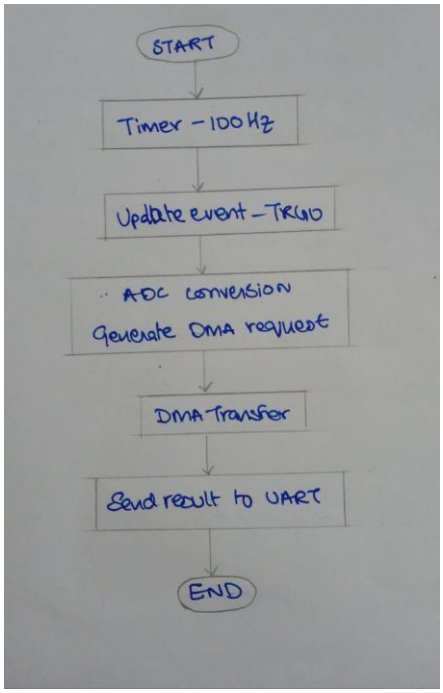
Mainly using multiple software interrupts and while loops inside then and also using SWSTART to start ADC, which is not what is exactly required by the problem statement.

The Fix:

After a little more research and referencing the registers used, decided to try using TRGO to create hardware interrupt to trigger ADC. Then in ADC->CR2, use EXTEN, DMA_EN, DDS to generate DMA

requests everytime a conversion is complete. Use DMA_initialisation function to transfer data to memory.
Finally, use only one DMA_IRQHandler(), to wait for transfer to memory complete and then display the results using UART.

The modified approach:



CODE:

Main.c

```
//FOLLOWING CODE HAS BEEN WRITTEN FOR STM32F411CEU6
#include "stm32f4xx.h"
#include <stdint.h>
#include <stdio.h>
#include "adc.h"
#include "uart.h"
#include "systick.h"
#include "tim.h"

extern volatile uint16_t data;
volatile uint16_t adc_latest;
volatile uint8_t dma_done = 0;

int main(void){
      uart2_tx_init();
      pa1_adc_init();
      dma2_stream0_init();
      tim2_100hz_interrupt_init();
      while(1){
            if (dma_done){
                  dma_done=0;
```

```
                printf("ADC = %u\r\n", adc_latest);
            }
      }
}
void DMA2_Stream0_IRQHandler(void)
{
      /*CHECK FOR TRASNFER COMPLETE INTERRUPT*/
      if (DMA2->LISR & LISR_TCIF0){
            /*CLEAR FLAG*/
            DMA2->LIFCR |= LIFCR_CTCIF0;
            //SEND RESULT TO UART
            adc_latest=data;
            dma_done=1;
      }
}
```

Timer.c

```
#include "stm32f4xx.h"

#define tim2_en (1U<<0)
#define CR1_CEN (1U<<0)
#define DIER_UIE (1U<<0)
#define CR2_MMS (1U<<5)

void tim2_100hz_interrupt_init(void)
{
      /*enable clock access to tim2 thru apb1*/
      RCC->APB1ENR |= tim2_en;
      /*set PSC value: configuring clock tree, essentially, so that
timer is 1hz or 1s
       * so every 1s there is a timeout*/
      TIM2->PSC =1600-1;  //16Mhz /1600 = 10 000
      /*set ARR value   */
      TIM2->ARR = 100 -1;  // 10000/100 = 100
      /*clear counter*/
      TIM2->CNT = 0;
      /*enable timer*/
      TIM2->CR1 = CR1_CEN;

      /*ENABLE UPDATE EVENT - TRGO */
      TIM2->CR2 &=~(7U<<4);
      TIM2->CR2 |= CR2_MMS;

}
```

ADC.c

```
#include "stm32f4xx.h"
#include "adc.h"

#define ADC1EN (1U<<8)
#define ADC_CH1 (1U<<0)
#define GPIOAEN (1U<<0)
#define ADC_SEQ_LEN_1 0x00
#define CR2_ADON (1U<<0)
#define CR2_SWSTART (1U<<30)
#define SR_EOC (1U<<1)

/*---------- DMA2 CHANNEL 0 STREAM 0 --------------------*/
#define DMA2EN (1U<<22)
#define DMA_S0_EN (1U<<0)
#define CHSEL0_MASK (7U<<25)
#define DMA_MEM_INC (1U<<10)
#define DMA_DIR_PERIPH_TO_MEM_MASK (3U<<6)
#define DMA_CR_TCIE (1U<<4)

#define DMA_PSIZE_16BIT (1U << 11)
#define DMA_MSIZE_16BIT (1U << 13)

#define ADC_DMA_EN (1U<<8)

#define ADC_DMA_EXTEN (1U<<28)
#define ADC_DDS (1U<<9)

/*for DMA to start automatically = circular buffer (no need to press
nrst everytime)*/
//#define DMA_CR_CIRC (1U << 8)

/*----------------------------------------*/
volatile uint16_t data;

void dma2_stream0_init()
{
	/*enable clock access to dma - connect to ahb1*/
	RCC->AHB1ENR |= DMA2EN;

	/*disable dma2 stream0*/
	DMA2_Stream0->CR &=~ DMA_S0_EN;

	/*wait until DMA2 stream0 is disabled*/
	while (DMA2_Stream0->CR & DMA_S0_EN){};

	/*clear all interrupt flags of stream0*/
	DMA2->LIFCR |= (1U<<0);
	DMA2->LIFCR |= (1U<<2);
	DMA2->LIFCR |= (1U<<3);
	DMA2->LIFCR |= (1U<<4);
	DMA2->LIFCR |= (1U<<5);
```

```
	/*set the destination buffer*/
	DMA2_Stream0->PAR = (uint32_t)&ADC1->DR;

	/*set the source buffer*/
	DMA2_Stream0->M0AR =(uint32_t) &data;

	/*set the length*/
	DMA2_Stream0->NDTR = 1;

	/*SELECT STREAM0 AND CHANNEL 0*/
	DMA2_Stream0->CR &=~(CHSEL0_MASK);

	/*enable memory to increment*/
	DMA2_Stream0->CR &=~ DMA_MEM_INC;

	/*configure the transfer direction (RX) - here, data moving
from PERIPHERAL TO MEMORY*/
	DMA2_Stream0->CR &=~(DMA_DIR_PERIPH_TO_MEM_MASK);

	/*ENABLE DMA TX COMPLETE INTERRUPT*/
	DMA2_Stream0->CR |= DMA_CR_TCIE;

	/*enable direct mode, disable FIFO*/
	DMA2_Stream0->FCR = 0;

	// Clear PSIZE and MSIZE
	DMA2_Stream0->CR &= ~((3U << 11) | (3U << 13));

	// Set peripheral size = 16-bit
	DMA2_Stream0->CR |= DMA_PSIZE_16BIT;

	// Set memory size = 16-bit
	DMA2_Stream0->CR |= DMA_MSIZE_16BIT;

	/*enable circular mode*/
	//DMA2_Stream0->CR |= DMA_CR_CIRC;

	/*ENABLE DMA2 STREAM0*/
	DMA2_Stream0->CR |= DMA_S0_EN;

	/*ENABLE ADC1 DMA*/
	ADC1->CR2 |= ADC_DMA_EN;

	/*ENABLE EXTERNAL TRIGGER AT RISING EDGE*/
	ADC1->CR2 &= ~(3U << 28);  // clear EXTEN
	ADC1->CR2 |= ADC_DMA_EXTEN;

	/*select the external event used to trigger the start of
conversion: TIM2 TRGO */
	ADC1->CR2 &=~ (1U<<24);
```

```
	ADC1->CR2 |= (1U<<25);
	ADC1->CR2 |= (1U<<26);
	ADC1->CR2 &=~ (1U<<27);

	/*SET DDS TO ENABLE CONTINOUS DMA REQUESTS*/
	ADC1->CR2 |= ADC_DDS;

	NVIC_EnableIRQ(DMA2_Stream0_IRQn);
}

void pa1_adc_init(void){
	/**Configure the ADC GPIO pin****/
	/*Enable clock access to gpioA*/
	RCC->AHB1ENR|=GPIOAEN;
	/*Set the mode of PA1 to analog*/
	GPIOA->MODER |=(1U<<2);
	GPIOA->MODER |=(1U<<3);
	/*Configure the ADC module*/
	/*Enable clock acces to ADC*/
	RCC->APB2ENR |= ADC1EN;
	/*Configure adc parameters*/
	/*CONVERSION seqn start*/
	ADC1->SQR3 = ADC_CH1;
	/*COnversion seqn length*/
	ADC1->SQR1 = ADC_SEQ_LEN_1;

	/*set sampling time*/
	//ADC1->SMPR2 &= ~(7U << 3);
	//ADC1->SMPR2 |=  (7U << 3);

	/*Enable ADC module*/
	ADC1->CR2 |= CR2_ADON;

	for(volatile int i = 0; i < 1000; i++);
}
```

UART.c

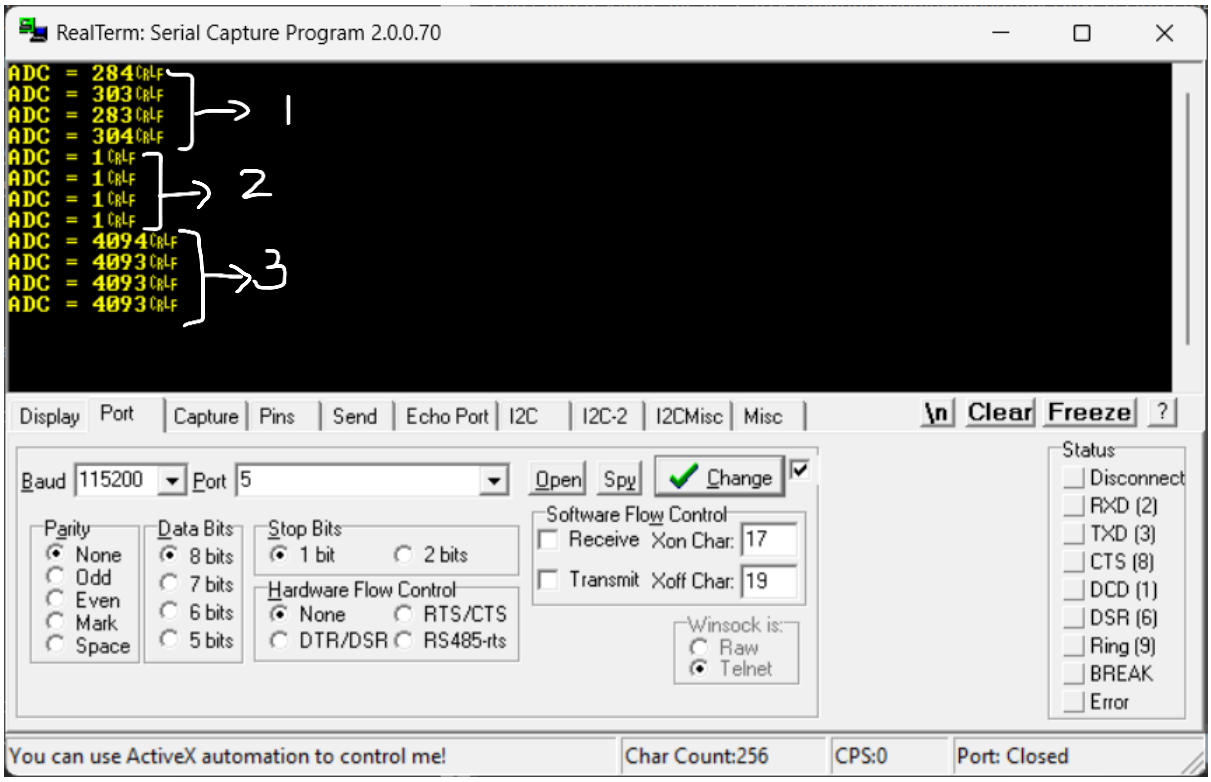
```
#include "uart.h"

#define GPIOAEN (1U<<0)
#define UART2EN (1U<<17)

#define CR1_TE (1U<<3)
#define CR1_RE (1U<<2)
#define CR1_UE (1U<<13)

#define SR_TXE (1U<<7)
#define SR_RXNE (1U<<5)

#define SYSFREQ 16000000
#define APB1_CLK SYSFREQ
```

```
#define UART_BAUDRATE 115200
/*-----------------------------------*/

static void uart_set_baudrate(USART_TypeDef *USARTx,uint32_t
PeriphClk, uint32_t BaudRate);
static uint16_t compute_uart_bd(uint32_t PeriphClk, uint32_t
BaudRate);
void uart2_tx_init(void);
void uart2_write(int ch);
/*-------------------------------------*/

int __io_putchar(int ch){
      uart2_write(ch);
      return ch;
}


void uart2_tx_init(void){
      /***Configure uart gpio pin****/

      /**1.Enable clock access to gpioa***/
      RCC->AHB1ENR|=GPIOAEN;

      /***2.Set PA2 mode to AF mode ****/
      GPIOA->MODER&=~(1U<<4);
      GPIOA->MODER|=(1U<<5);

      /***2.Set PA2 AF type to UART_TX(AF07)****/

      /**GPIO AFLR - ref manual - Pin2 -bit 8 to 11**/
      GPIOA->AFR[0]|=(1U<<8);
      GPIOA->AFR[0]|=(1U<<9);
      GPIOA->AFR[0]|=(1U<<10);
      GPIOA->AFR[0]&=~(1U<<11);

      /*****Configure UART module ********/

      /**Enable clock access to uart2****/
      RCC->APB1ENR|=UART2EN;
      /***configure baudrate***/
      uart_set_baudrate(USART2,APB1_CLK,UART_BAUDRATE);
      /***configure the transfer direction***/
      USART2->CR1 = CR1_TE; //Not using | operate deliberately,
clearing all bits except bit 3
      /***enable the UART module***/
      USART2->CR1|= CR1_UE;
}


void uart2_write(int ch){
```

```
	/***MAke sure TX register is empty b4 putting data in it***/
	while (!(USART2->SR & SR_TXE)){}
	/**Write to transmit data register***/
	USART2->DR=(ch & 0xFF);
}
```

```
static void uart_set_baudrate(USART_TypeDef *USARTx,uint32_t
PeriphClk, uint32_t BaudRate){
	USARTx->BRR=compute_uart_bd(PeriphClk,BaudRate);
}

static uint16_t compute_uart_bd(uint32_t PeriphClk, uint32_t
BaudRate){
	return ((PeriphClk +(BaudRate/2U))/BaudRate);
}
```

OUTPUT:



1- **when pin PA1 is floating**
2- **When pin PA1 is connected to GND**
3- **When pin PA1 is connected 3V3**

Result:

Hence, we have achieved our aim to transfer an analog voltage value of GPIO pin every 100Hz through DMA to store it and use UART to view the value read.