Guía práctica ADC:

El objetivo de esta práctica es convertir cuatro señales analógicas a digitales con la etiqueta de Voltaje de batería, Temperatura en grados Celsius, Velocidad en revoluciones por minuto y Peso en kilos, de un sistema que tiene las señales unidas a los canales 0, 2, 4 y 6, respectivamente. Cada toma debe enviarse a un terminal serial con la información completa de cada muestra. En cada una de las entradas analógicas debe haber un potenciómetro que simule las señales. El procedimiento es el siguiente:

1. Utilice la siguiente función para hacer la inicialización del puerto serial 1 a una velocidad de 9.600 bps:

```
/*********************************************
* Inicialización del puerto Serial1
*********************************************/
void UART1_Init(void)
{
    // *Habilitar el reloj del puerto serial, del puerto Gpio donde está ubicado y de la función alterna.
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_AFIO | RCC_APB2Periph_GPIOA, ENABLE);

    // *Configuración del modo de función alterna de cada pin del puerto serial.
    GPIO_InitTypeDef GPIO_Struct;
    // TX: GPIOA PIN9 funcion alterna
    GPIO_Struct.GPIO_Pin = GPIO_Pin_9;
    GPIO_Struct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Struct.GPIO_Mode = GPIO_Mode_AF_PP;
    GPIO_Init(GPIOA, &GPIO_Struct);
    // RX: GPIOA PIN10 funcion alterna
    GPIO_Struct.GPIO_Pin = GPIO_Pin_10;
    GPIO_Struct.GPIO_Speed = GPIO_Speed_50MHz;
    GPIO_Struct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
    GPIO_Init(GPIOA, &GPIO_Struct);

    // Inicializa parametros de Uart 9600 bps, 8 bits, 1 stop, no paridad
    USART_InitTypeDef UART_Struct;
    USART_StructInit(&UART_Struct);
    UART_Struct.USART_BaudRate= 9600;
    //*Habilitación del puerto serial. TX Y RX
    USART_Init(USART1, &UART_Struct);
    USART_Cmd(USART1, ENABLE);
}
```

2. Use la siguiente función de inicialización del reloj para tener un SysClk de 56 Mhz.

```
/*************************************************
 *  CONFIGURA EL SYSCLOCK EN 56Mh CON CONEXION EN LOS PERIFERICOS: HCLK, PCLK1, PCLK2.
 *************************************************/
void reloj_56M(void)
{
    // CONFIGURAR EL RELOJ
    RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI); // Asegura PLL no esté en uso
    RCC_PLLCmd(DISABLE);// para cambiar multiplicador, debe deshabilitarse el PLL
    RCC_PLLConfig(RCC_PLLSource_HSE_Div2, RCC_PLLMul_14); //8Mhz/2*14= 56Mhz
    RCC_PLLCmd(ENABLE); // habilita el PLL
    while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY)== RESET)
    {
    }
    // ESTAS SON LAS CONFIGURACIONES DE CADA PREESCALER DE LOS PUERTOS
    RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK); // PREESCALER DEL INICIO.
    RCC_HCLKConfig( RCC_SYSCLK_Div1); // HCLK = SYSCLK
    RCC_PCLK2Config( RCC_HCLK_Div1);  // PCLK2 = HCLK
    RCC_PCLK1Config( RCC_HCLK_Div2);  // PCLK1 = HCLK/2
}
```

3. Defina la estructura del puerto donde están los canales 0, 2, 4 y 6 y la estructura del ADC.

5. Configure el puerto y los pines de entrada y el reloj del puerto.

Estructura de los puertos GPIO:

```
GPIO_InitTypeDef GPIO_InitStruct; // estructura para configurar los pines
// configuración de los pines del ADC (PA0 -> canal 0 a PA7 -> canal 7) como entradas analógicas
GPIO_StructInit(&GPIO_InitStruct); // inicialización de la estructura
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_2| GPIO_Pin_4| GPIO_Pin_6; // canales 0, 2, 4, 6
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
GPIO_Init(GPIOA, &GPIO_InitStruct);
```

6. Inicialice la estructura del ADC y habilite el modo Scan, deshabilite el disparo externo, fije el número de canales, habilite una sola muestra e inicialice el ADC1.

7. Configure cada canal con el orden que debe hacerse el muestreo y la velocidad de conversion.

Estructura del ADC:

```
ADC_InitTypeDef ADC_InitStruct;
ADC_InitStruct.ADC_Mode = ADC_Mode_Independent; // configuración del ADC1
ADC_InitStruct.ADC_ScanConvMode = ENABLE;    // multiples canales
ADC_InitStruct.ADC_ContinuousConvMode = DISABLE; // modo de conversión contínuo
ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;  // sin inicio de conversión externo
ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right; // alineamiento de presentación de datos hacia la derecha
ADC_InitStruct.ADC_NbrOfChannel = num_canales; // 8 canales de conversión
ADC_Init(ADC1, &ADC_InitStruct); // carga información de configuración
// configuración de cada canal
ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_239Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_2, 2, ADC_SampleTime_239Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_4, 3, ADC_SampleTime_239Cycles5);
ADC_RegularChannelConfig(ADC1, ADC_Channel_6, 4, ADC_SampleTime_239Cycles5);
```

4. Fije la frecuencia del ADC a no más de 14 MHz y habilite el reloj del ADC.

```
// habilitación del ADC1 y GPIOA
RCC_ADCCLKConfig(RCC_PCLK2_Div4);// 14M
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOA, ENABLE);
```

8. Habilite el ADC1.

```
// habilitación de ADC1
ADC_Cmd(ADC1, ENABLE);

// calibración
ADC_ResetCalibration(ADC1);
while(ADC_GetResetCalibrationStatus(ADC1));
ADC_StartCalibration(ADC1);
while(ADC_GetCalibrationStatus(ADC1));
```

9. Configure el DMA, canal 1 y lo habilita

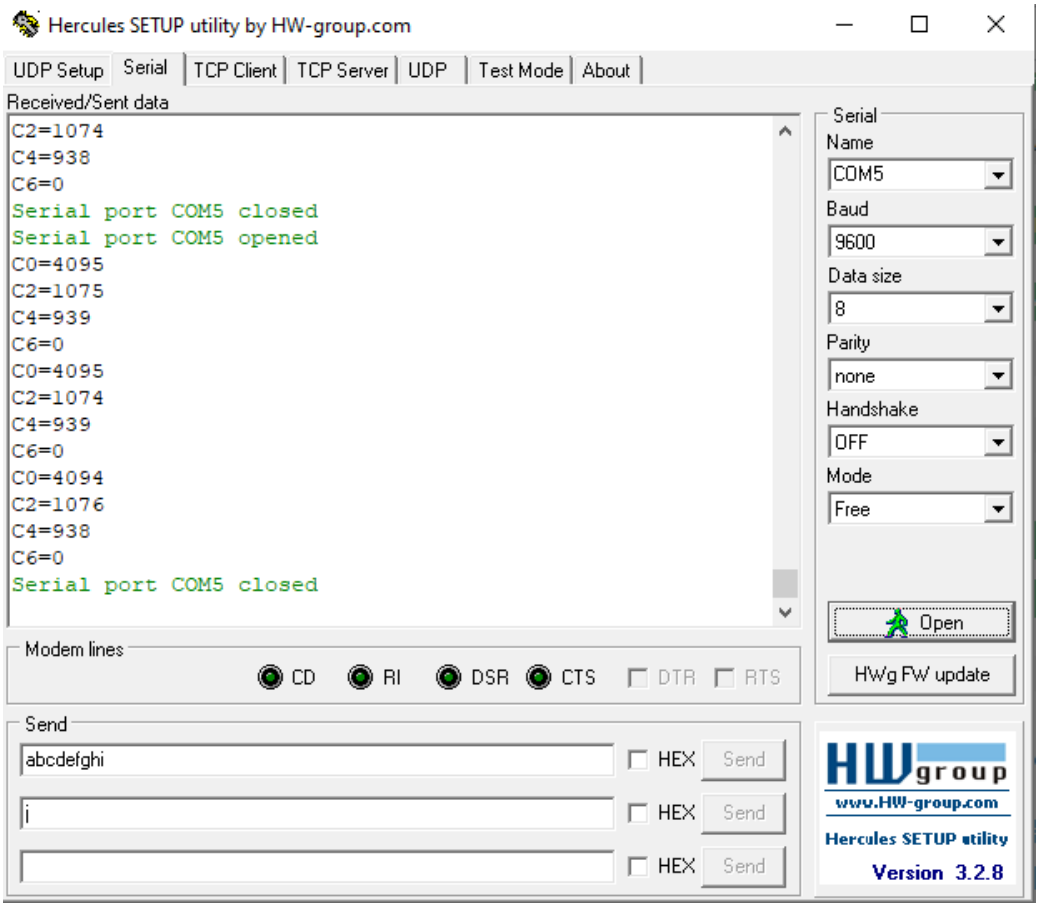
```
/*********************************************
 * Configuracion del DMA PARA DMA.
 *********************************************/
void DMA_ADC_Init(uint32_t *destination, uint16_t num_canales)
 {
    RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE); // activación del reloj del DMA
    DMA_DeInit(DMA1_Channel1); // DESINICIALIZA EL CANAL QUE VAMOS A CONFIGURAR.
    // CONFIGURACION DEL DMA POR DAFAULT PRIMERO
    DMA_InitTypeDef DMA_InitStruct; // CREA LA ESTRUCTURA PARA DMA
    DMA_StructInit(&DMA_InitStruct); // INICIALIZA POR DEFAULT, EL & DICE TOME EL VALOR Y AHI TOMA LO Q TENGA A LA DERECHA.
    // CONFIGURACION DEL DMA REQUERIDA
    DMA_InitStruct.DMA_M2M = DMA_M2M_Disable; // este canal se va a usar para transferencia desde periférico a memoria
    DMA_InitStruct.DMA_Mode = DMA_Mode_Circular; //selección de modo circular
    DMA_InitStruct.DMA_Priority = DMA_Priority_Medium; //prioridad media
    DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord; //tamaño del dato de la fuente y el destino= 16bit
    DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Enable; //habilitación de incremento automático en destino
    DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralSRC; //La posición asignada al registro periférico será la fuente
    DMA_InitStruct.DMA_BufferSize = num_canales; //tamaño de los datos que se transfieren
    DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR; //dirección de inicio de la fuente y el destino
    DMA_InitStruct.DMA_MemoryBaseAddr = (uint32_t)destination;
    // INICIALIZA EL DMA Y EL USAR.
    DMA_Init(DMA1_Channel1, &DMA_InitStruct); //programa registros del DMA

    DMA_Cmd(DMA1_Channel1, ENABLE); //habilita transferencia en el canal de DMA1
    ADC_DMACmd(ADC1, ENABLE); // habilitación de DMA para ADC
 }
```

10. Dentro del while(1), inicie la conversión y espere a que termine.

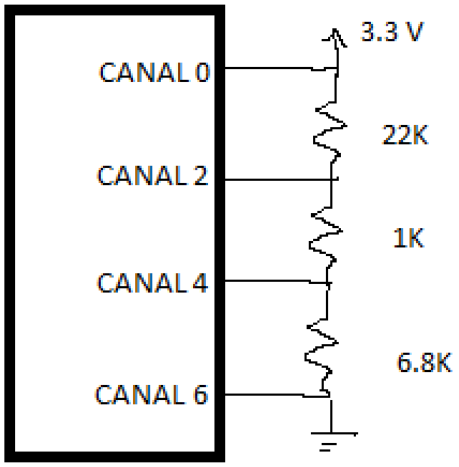
11. Lea y envíe cada dato al terminal serial y espere aproximadamente un segundo para volver a hacer la siguiente toma de muestras.

```
while(1)
{
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, 'C');
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, '0');
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, '=');
    UART_numeroADC(destination[0]);
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, 'C');
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, '2');
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, '=');
    UART_numeroADC(destination[1]);
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, 'C');
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, '4');
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, '=');
    UART_numeroADC(destination[2]);
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, 'C');
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, '6');
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
    USART_SendData(USART1, '=');
    UART_numeroADC(destination[3]);
    for (int i = 0; i < 2000000; ++i) asm("nop");// retardo
    ADC_SoftwareStartConvCmd(ADC1 , ENABLE);// inicia conversión
}
```

RESULTADO:



CONFIGURACION DE LOS CANALES:



SE REALIZO MEDICION DE VOLTAJE CON MULTIMETRO Y SE CORROBORO LOS VALORES CON LOS VALORES DE HERCULES Y DIERON EXACTOS.

CODIGO COMPLETO:

```
#include "stm32f10x_conf.h"

void reloj_56M(void);
void UART1_Init(void);
void UART_numeroADC(uint32_t adc_value);
void DMA_ADC_Init(uint32_t *destination, uint16_t num_canales);

int main(void)
{
   reloj_56M();
   UART1_Init();

   uint16_t destination[]={0,0,0,0};// arreglo donde van a quedar los datos de los canales
convertidos
   uint16_t num_canales = 4;

   // configuracion de los pines de los canales del adc
   // habilitación del ADC1 y GPIOA
   RCC_ADCCLKConfig(RCC_PCLK2_Div4);// 14M
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1 | RCC_APB2Periph_GPIOA,
ENABLE);

   GPIO_InitTypeDef GPIO_InitStruct; // estrutura para configurar los pines
   // configuración de los pines del ADC (PA0 -> canal 0 a PA7 -> canal 7) como entradas
analoógicas
   GPIO_StructInit(&GPIO_InitStruct); // inicialización de la estructura
   GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0 | GPIO_Pin_2| GPIO_Pin_4| GPIO_Pin_6; //
canales 0, 2, 4, 6
   GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN;
   GPIO_Init(GPIOA, &GPIO_InitStruct);

   ADC_InitTypeDef ADC_InitStruct;
   ADC_InitStruct.ADC_Mode = ADC_Mode_Independent; // configuración del ADC1
   ADC_InitStruct.ADC_ScanConvMode = ENABLE;   // multiples canales
   ADC_InitStruct.ADC_ContinuousConvMode = DISABLE; // modo de conversión contínuo
   ADC_InitStruct.ADC_ExternalTrigConv = ADC_ExternalTrigConv_None;  // sin inicio de
conversión externo
   ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right; // alineamiento de presentación de
datos hacia la derecha
   ADC_InitStruct.ADC_NbrOfChannel = num_canales; // 8 canales de conversión
   ADC_Init(ADC1, &ADC_InitStruct); // carga información de configuración
   // configuración de cada canal
   ADC_RegularChannelConfig(ADC1, ADC_Channel_0, 1, ADC_SampleTime_239Cycles5);
   ADC_RegularChannelConfig(ADC1, ADC_Channel_2, 2, ADC_SampleTime_239Cycles5);
   ADC_RegularChannelConfig(ADC1, ADC_Channel_4, 3, ADC_SampleTime_239Cycles5);
   ADC_RegularChannelConfig(ADC1, ADC_Channel_6, 4, ADC_SampleTime_239Cycles5);


   // habilitación de ADC1
```

```
   ADC_Cmd(ADC1, ENABLE);

   // calibración
   ADC_ResetCalibration(ADC1);
   while(ADC_GetResetCalibrationStatus(ADC1));
   ADC_StartCalibration(ADC1);
   while(ADC_GetCalibrationStatus(ADC1));

   DMA_ADC_Init(&destination, num_canales);

   ADC_SoftwareStartConvCmd(ADC1 , ENABLE);// inicia conversión

  while(1)
  {
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, 'C');
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, '0');
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, '=');
     UART_numeroADC(destination[0]);
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, 'C');
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, '2');
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, '=');
     UART_numeroADC(destination[1]);
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, 'C');
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, '4');
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, '=');
     UART_numeroADC(destination[2]);
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, 'C');
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, '6');
     while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
     USART_SendData(USART1, '=');
     UART_numeroADC(destination[3]);
     for (int i = 0; i < 2000000; ++i) asm("nop");// retardo
     ADC_SoftwareStartConvCmd(ADC1 , ENABLE);// inicia conversión
  }
}



/*********************************************
```

```
 *  CONFIGURA EL SYSCLOCK EN 56Mh CON CONEXION EN LOS PERIFERICOS: HCLK,
PCLK1, PCLK2.
 **********************************************/
void reloj_56M(void)
{
   // CONFIGURAR EL RELOJ
   RCC_SYSCLKConfig(RCC_SYSCLKSource_HSI); // Asegura PLL no esté en uso
   RCC_PLLCmd(DISABLE);// para cambiar multiplicador, debe deshabilitarse el PLL
   RCC_PLLConfig(RCC_PLLSource_HSE_Div2, RCC_PLLMul_14); //8Mhz/2*14= 56Mhz
   RCC_PLLCmd(ENABLE); // habilita el PLL
   while(RCC_GetFlagStatus(RCC_FLAG_PLLRDY)== RESET)
   {
   }
   // ESTAS SON LAS CONFIGURACIONES DE CADA PREESCALER DE LOS PUERTOS
   RCC_SYSCLKConfig(RCC_SYSCLKSource_PLLCLK); // PREESCALER DEL INICIO.
   RCC_HCLKConfig( RCC_SYSCLK_Div1); // HCLK = SYSCLK
   RCC_PCLK2Config( RCC_HCLK_Div1);  // PCLK2 = HCLK
   RCC_PCLK1Config( RCC_HCLK_Div2);  // PCLK1 = HCLK/2
}

/**********************************************
* Inicialización del puerto Serial1
**********************************************/
void UART1_Init(void)
{
   // *Habilitar el reloj del puerto serial, del puerto Gpio donde está ubicado y de la función
alterna.
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1 | RCC_APB2Periph_AFIO |
RCC_APB2Periph_GPIOA, ENABLE);

   // *Configuración del modo de función alterna de cada pin del puerto serial.
   GPIO_InitTypeDef GPIO_Struct;
   // TX: GPIOA PIN9 funcion alterna
   GPIO_Struct.GPIO_Pin = GPIO_Pin_9;
   GPIO_Struct.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Struct.GPIO_Mode = GPIO_Mode_AF_PP;
   GPIO_Init(GPIOA, &GPIO_Struct);
   // RX: GPIOA PIN10 funcion alterna
   GPIO_Struct.GPIO_Pin = GPIO_Pin_10;
   GPIO_Struct.GPIO_Speed = GPIO_Speed_50MHz;
   GPIO_Struct.GPIO_Mode = GPIO_Mode_IN_FLOATING;
   GPIO_Init(GPIOA, &GPIO_Struct);
   // Inicializa parametros de Uart 9600 bps, 8 bits, 1 stop, no paridad
   USART_InitTypeDef UART_Struct;
   USART_StructInit(&UART_Struct);
   UART_Struct.USART_BaudRate= 9600;
   //*Habilitación del puerto serial. TX Y RX
   USART_Init(USART1, &UART_Struct);
   USART_Cmd(USART1, ENABLE);
}
/**********************************************
```

```
 *  FUNCION PARA ENVIAR EL NUMERO ADC (ENTRE 0 Y 4095) DE CUALQUIER TAMAÑO POR UART CONVERTIDO A ASCII.
 ***********************************************/
void UART_numeroADC(uint32_t adc_value)
{
   uint16_t arreglo[]={'0', '0', '0', '0', '0', '0', '0', '0', '0', '0'}; // arreglo donde se va a guardar el dato subdividido
   uint32_t a= adc_value;
   uint32_t b, c;
   uint8_t apuntador= 0;

   /// Subdividir el numero (adc_value) por cada entero.
   if(a== 0)
   {
      while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
      USART_SendData(USART1, '0');
   }
   while(a>0)
   {
      b= a/10;
      c= (uint32_t) (a- b*10);
      arreglo[apuntador]= (uint16_t) c + 48;
      apuntador++;
      a= b;
   }

   /// Mostrar en pantalla el vector.
   for(uint8_t i=0; i<apuntador; i++)
   {
      while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
      USART_SendData(USART1, arreglo[apuntador-i-1]);
   }
   while(USART_GetFlagStatus(USART1, USART_FLAG_TXE)== RESET);
   USART_SendData(USART1, '\n');
}



/***********************************************
 * Configuracion del DMA PARA DMA.
 ***********************************************/
void DMA_ADC_Init(uint32_t *destination, uint16_t num_canales)
 {
   RCC_AHBPeriphClockCmd(RCC_AHBPeriph_DMA1, ENABLE); // activación del reloj del DMA
   DMA_DeInit(DMA1_Channel1); // DESINICIALIZA EL CANAL QUE VAMOS A CONFIGURAR.
   // CONFIGURACION DEL DMA POR DAFAULT PRIMERO
   DMA_InitTypeDef DMA_InitStruct; // CREA LA ESTRUCTURA PARA DMA
   DMA_StructInit(&DMA_InitStruct); // INICIALIZA POR DEFAULT, EL & DICE TOME EL VALOR Y AHI TOMA LO Q TENGA A LA DERECHA.
```

```
    // CONFIGURACION DEL DMA REQUERIDA
    DMA_InitStruct.DMA_M2M = DMA_M2M_Disable; // este canal se va a usar para
transferencia desde periférico a memoria
    DMA_InitStruct.DMA_Mode = DMA_Mode_Circular; //selección de modo circular
    DMA_InitStruct.DMA_Priority = DMA_Priority_Medium; //prioridad media
    DMA_InitStruct.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord; //tamaño
del dato de la fuente y el destino= 16bit
    DMA_InitStruct.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
    DMA_InitStruct.DMA_MemoryInc = DMA_MemoryInc_Enable; //habilitación de incremento
automático en destino
    DMA_InitStruct.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
    DMA_InitStruct.DMA_DIR = DMA_DIR_PeripheralSRC; //La posición asignada al registro
periférico será la fuente
    DMA_InitStruct.DMA_BufferSize = num_canales; //tamaño de los datos que se transfieren
    DMA_InitStruct.DMA_PeripheralBaseAddr = (uint32_t)&ADC1->DR; //dirección de inicio de la
fuente y el destino
    DMA_InitStruct.DMA_MemoryBaseAddr = (uint32_t)destination;
    // INICIALIZA EL DMA Y EL USAR.
    DMA_Init(DMA1_Channel1, &DMA_InitStruct); //programa registros del DMA

    DMA_Cmd(DMA1_Channel1, ENABLE); //habilita transferencia en el canal de DMA1
    ADC_DMACmd(ADC1, ENABLE); // habilitación de DMA para ADC
 }
```