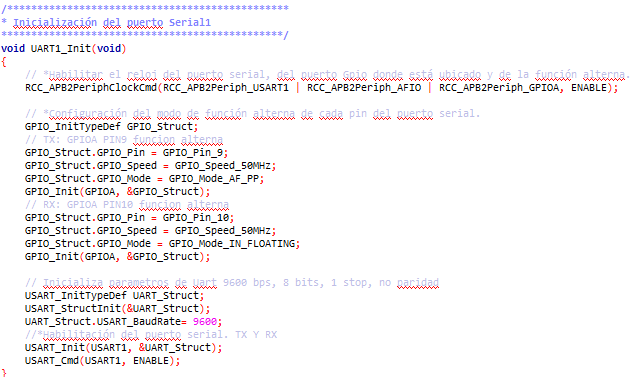
**Guía práctica ADC:**

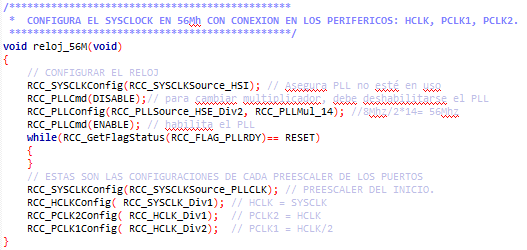
El objetivo de esta práctica es convertir cuatro señales analógicas a digitales con la etiqueta de

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:**



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

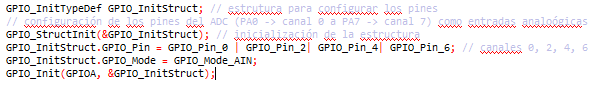


**3. Defina la estructura del puerto donde están los canales 0, 2, 4 y 6 y la estructura del**

**ADC.**

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

Estructura de los puertos GPIO:



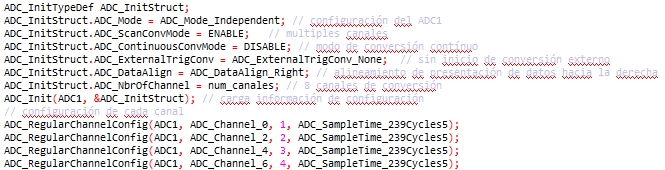
**6. Inicialice la estructura del ADC y habilite el modo Scan, deshabilite el disparo externo,**

**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.**

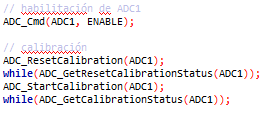
Estructura del ADC:



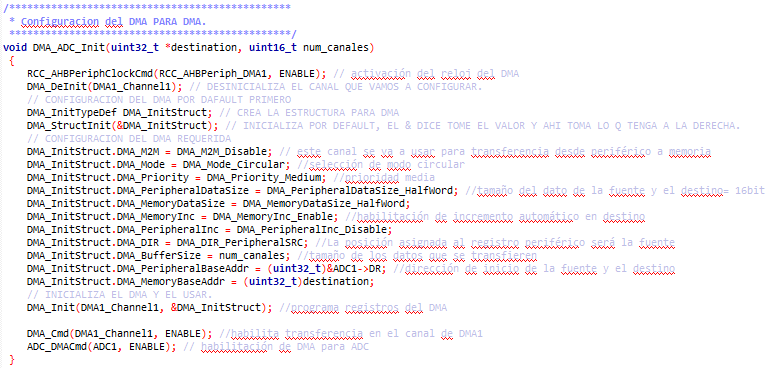
**4. Fije la frecuencia del ADC a no más de 14 MHz y habilite el reloj del ADC.**



**8. Habilite el ADC1.**



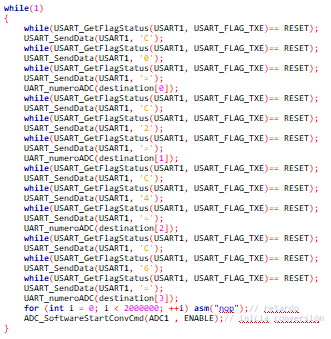
**9. Configure el DMA, canal 1 y lo habilita**



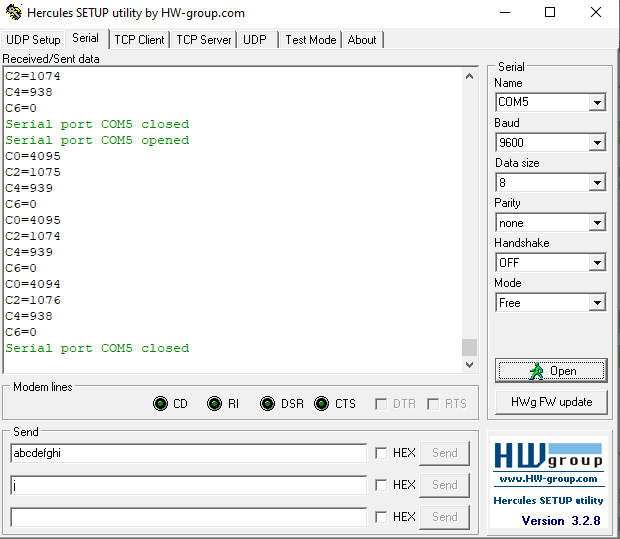
**10. Dentro del while(1), inicie la conversión y espere a que termine.**

**11. Lea y envíe cada dato al terminal serial y espere aproximadamente un segundo para**

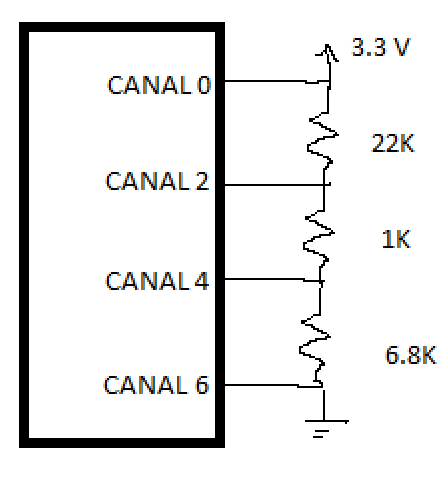
**volver a hacer la siguiente toma de muestras.**



**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

}