Consideraciones importantes.

El presente documento desarrolla la implementación de una función de transferencia en tiempo discreto en la plataforma de desarrollo Ophyra diseñada por Intesc Electronics & Embedded. Tome en cuenta que por la extensión del documento solo se explica la información importante o esencial para la práctica y no se muestran los pasos intermedios o detalles de desarrollo triviales. Por lo tanto este documento supone que el lector tiene conocimientos previos de programación y de procesamiento digital de señales.
Para cualquier duda referente a la práctica o información adicional, por favor diríjase a la siguiente dirección www.intesc.mx y consulte la sección de ***Soporte->Contenido->Ophyra*** o también diríjase a la sección de ***Soporte->Foro***, en donde con gusto responderemos a tus preguntas.

Para información especializada sobre el microcontrolador ***STM32F407VG*** diríjase a la siguiente dirección www.st.com e introduzca en el buscador de la página el modelo del microcontrolador.

1. Objetivo

Implementar en el tiempo discreto una función de transferencia que representa el comportamiento de un motor. Utilizar la plataforma de desarrollo Ophyra para visualizar una señal cuadrada de entrada y la respuesta de salida en el osciloscopio de la función de transferencia.

2. Antecedentes matemáticos y desarrollo de los cálculos.

En la figura 1 se observa el modelo a bloques (implementado en Simulink de MATLAB) que representa el comportamiento de un motor. El primer bloque muestra el modelo matemático en el dominio de Laplace, el segundo bloque, representa el modelo en el dominio Z (tiempo discreto). En la figura 2, se observa la representación gráfica del comportamiento de ambos sistemas cuando se tiene a la entrada un escalón unitario (línea de color azul). La línea de color verde representa la respuesta de salida del primer bloque (dominio Laplace). La línea de color rojo representa la respuesta de salida del sistema en el dominio Z.

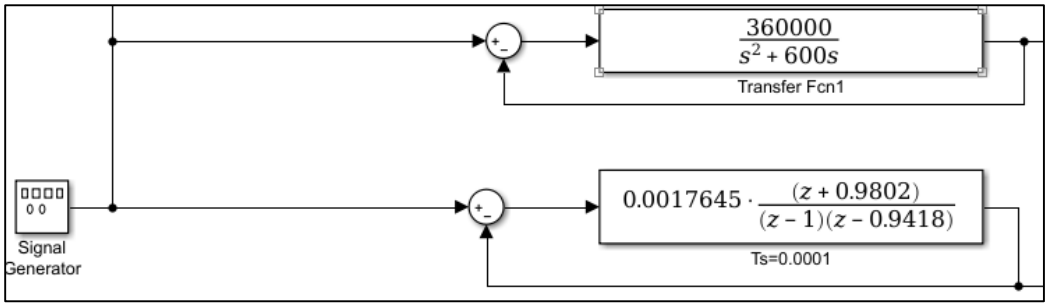

Figura 1. Funciones de transferencia.

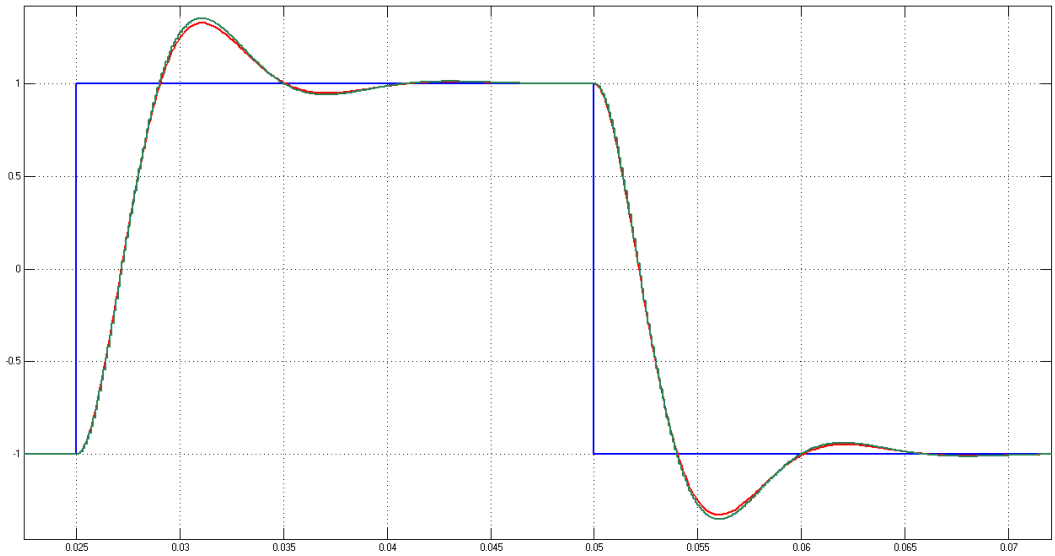

Figura 2. Respuesta de los bloques en Simulink.

Una forma de representar sistemas lineales discretos, como la que se muestra en la ecuación 1, es mediante la forma de ecuación de diferencia lineales, como la mostrada en la ecuación 2. Esta forma representa la función en términos de datos discretos que se van almacenando y recorriendo con forme el sistema de procesamiento está muestreando una entrada.

$$H(z) = \frac{Y(z)}{X(x)} = \frac{b_0+(b_1*z^{-1})+(b_2*z^{-2})+\ldots+(b_{m-1}*z^{(m-1)})+(b_m*z^{-m})}{a_0+(a_1*z^{-1})+(a_2*z^{-2})+\ldots+(a_{n-1}*z^{(n-1)})+(a_n*z^{-n})} \quad (1)$$

$$Y(k) = (a_1 * Y(k-1)) + (a_2 * Y(k-2)) + (a_n * Y(k-n)) + (b_0 * X(k)) + (b_1 * X(k-1)) + (b_m * X(k-m)) \quad (2)$$

A continuación se desarrollará la función de transferencia del segundo bloque mostrado en la figura 1, para obtener la forma de ecuación de diferencias lineales. El desarrollo comienza multiplicando ambos lados de la ecuación por el denominador.

$$[Y(z)] * [z^2 - 1.5488 + 0.5488] = [0.14881z + 0.121905152] * [X(z)] \quad (3)$$

Se divide ambos lados entre z².

$$\frac{1}{(z^2)} * [Y(z)] * [z^2 - 1.5488 + 0.5488] = \frac{1}{z^2}[0.14881z + 0.121905152] * [X(z)] \quad (4)$$

$$[Y(z)] * [1 - 1.5488z^{-1} + 0.5488z^{-2}] = [X(z)] * [0.14881z^{-1} + 0.121905152z^{-2}] \quad (5)$$

Se expande la función por ambos lados y se despeja *Y(z)*:

$$[Y(z) - 1.5488z^{-1}Y(z) + 0.5488z^{-2}Y(z)] = [0.14881z^{-1}X(z) + 0.121905152z^{-2}X(z)] \quad (6)$$

$$Y(z) = 1.5488z^{-1}Y(z) - 0.5488z^{-2}Y(z) + 0.14881z^{-1}X(z) + 0.121905152z^{-2}X(z) \quad (7)$$

Se reemplaza los términos discretos $z^{-n}Y(z)$ y $z^{-n}X(z)$ por sus equivalentes *Y(k-n)* y *X(k-n)*. De esta forma la ecuación 3 queda en términos de las muestras discretas.

$$Y(k) = 1.5488Y(k-1) - 0.5488Y(k-2) + 0.14881X(k-1) + 0.121905152X(k-2) \quad \textbf{(8)}$$

La ecuación 8 es el algoritmo que se implementará en la tarjeta de desarrollo Ophyra, para simular el comportamiento del motor.

3. Frecuencia de muestreo.

Todo sistema de procesamiento digital de señales emplea un tiempo para tomar una muestra, procesarla y producir una salida. En general al tiempo que requiere el sistema de cómputo para adquirir una muestra se le llama **Tiempo de Muestreo** (dado en unidades de segundo). El tiempo que requiere para procesar el dato entrante y generar la salida, se le llama **Tiempo de Procesamiento**. En muchas ocasiones el Tiempo de Procesamiento suele incluirse dentro del Tiempo de Muestreo, sin hacer una distinción clara entre uno y otro. En este documento se empleará el término *Tiempo de Muestreo* de manera general incluyendo ambos procesos. Si al tiempo de muestreo se le calcula su inversa se obtiene el concepto de **Frecuencia de Muestreo** (dado en unidades de Hz). La Frecuencia de Muestreo indica las veces por segundo que el sistema es capaz de tomar y procesar datos. Para está práctica se utilizará una frecuencia de muestreo igual a 10KHz, lo que quiere decir, que se procesarán hasta 10,000 muestras por segundo.

Por otra parte, el microcontrolador de Ophyra cuenta con 14 Temporizadores (Timers), de los cuales, 12 son de 16 bits y los restantes 2 son de 32 Bits. Así mismo son capaces de operar a frecuencias de hasta 84MHz o 42MHz. En el presente documento se utilizará un Temporizador de Ophyra para generar un Tiempo de Muestreo igual a 0.0001seg (reciproco de 10KHz). La idea general, es que el temporizador indique el momento exacto en que el microcontrolador debe tomar y procesar una muestra. Para ello se deben ajustar parámetros del Temporizador para que funcione a la frecuencia que se requiere. De lo anterior, los parámetros a ajustar son los que se muestran en las ecuaciones 9 y 10.

$$FrecuenciaTimer = \frac{FrecuenciaReloj}{(Prescaler+1)} \tag{9}$$

$$Tiempo = (\frac{1}{FrecuenciaTimer})(CounterPeriod + 1) \tag{10}$$

Donde:

Frecuencia de Reloj: Por defecto es 16MHz (puede ser modificada si es necesario).
Prescaler: Es el divisor de frecuencia y es un número entero de 16bits.
Tiempo: Inverso de la frecuencia a la que deseamos que el Timer opere.
CounterPeriod: La cuenta a la que debe llegar el Timer para generar un desborde.

Proponiendo un Prescaler con valor cero y despejando CounterPeriod tenemos:

$$CounterPeriod = [(Tiempo)(FrecuenciaTimer)] - 1 = [(\frac{1}{1KHz})(16 * MHz)] - 1$$

$$CounterPeriod = [(100x10^{-6})(16x10^{6})] - 1 = 1599 \tag{11}$$

Estos datos se utilizarán posteriormente durante el desarrollo de esta práctica.

3. Procedimiento: Implementación en Ophyra.

A partir de la ecuación 8 (que se reescribe a continuación) se puede analizar el algoritmo que se implementará en Ophyra.

$$Y(k) = 1.5488Y(k-1) - 0.5488Y(k-2) + 0.14881X(k-1) + 0.121905152X(k-2)$$

Los términos X(k) e *Y(k)* representan los datos actuales entrante y saliente respectivamente del sistema. Los términos *X(K-1), X(k-2)*, *Y(K-1), Y(K-2)* representan los datos acumulados o *retrasados* en un *(k-n)* tiempo anterior respecto del actual. Por ejemplo, *X(k-2)* representa un dato de entrada que fue muestreado y procesado 2 turnos anteriores con respecto del dato actual. Por lo tanto, es necesario guardar estos datos para que puedan ser procesados junto con la muestra actual *X(k)* (que se obtiene del ADC de Ophyra). Aunado a lo anterior, en el sistema original (mostrado en la figura 1) se observa una retro alimentación de la ecuación, es decir, la salida del sistema es retornada hacia atrás y sumada algebraicamente con la entrada. En términos de procesamiento en un microcontrolador, este procedimiento se realiza mediante la resta del dato actual (dato del ADC), menos la salida del sistema en un tiempo anterior o *Y(k-1)*. El resultado de la operación anterior recibe entonces el nombre de ***"error".*** El error es en realidad el dato numérico que ingresa a la ecuación del sistema como *X(k)*. Y los términos *X(k-1)* y *X(k-2)* son los errores calculados anteriormente. Por lo tanto, para guardar los datos como lo indica el algoritmo se realiza de la siguiente manera y en el orden estricto:

$$X(k)\ o\ error = LecturaADC - Y(k-1)$$
$$X(k-2) = X(k-1)$$
$$X(k-1) = error$$
$$Y(k-2) = Y(k-1)$$
$$Y(k-1) = Y(k)$$

Al procedimiento anterior se le llama corrimiento de datos en la memoria y es el procedimiento que se realizará en Ophyra.

La figura 3 muestra en forma de diagrama de flujo el algoritmo completo a implementar en Ophyra. Como se mencionó anteriormente, la idea global es que el Temporizador (ajustado a 10KHz) indique el momento exacto en el que se deben procesar los datos, mediante una llamada de interrupción.

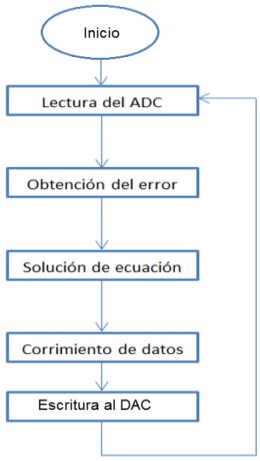

Figura 3. Algoritmo para Ophyra.

Ophyra cuenta con puertos de expansión donde tenemos accesibilidad a recursos como ADC, DAC, PWM, TFT, SD-CARD, BUS-CAN, entre otros, haciendo que su uso sea más flexible comparado con otras tarjetas. Para está práctica se utilizará el Canal 0 del ADC1, el Timer 3 y el canal 2 del DAC.

Para implementar el algoritmo primero se genera un proyecto nuevo en STM32CubeMx. Seguidamente se selecciona los recursos a usar: ADC, DAC y el Timer 3, como se muestra en la figura 4.

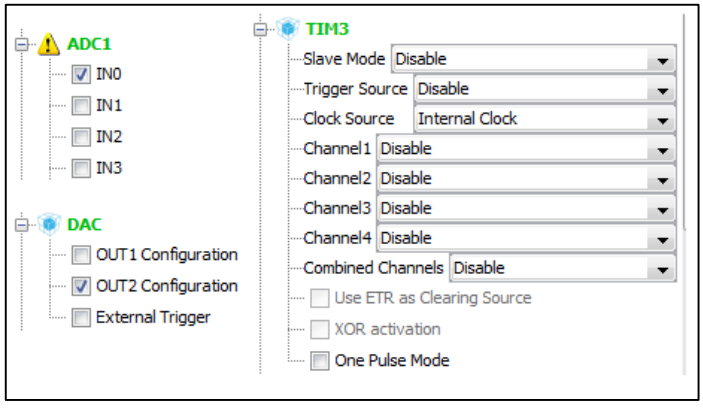

Figura 4. Selección de recursos.

En la sección 2, con la ecuación 11, se calcularon los datos que se deben ingresar a la configuración de recursos del Timer3 para ajustarlo a 10KHz. En la figura 5 se muestra este procedimiento.

Nota: El alumno puede modificar y recalcular los valores del Prescaler y Counter Period según convenga.

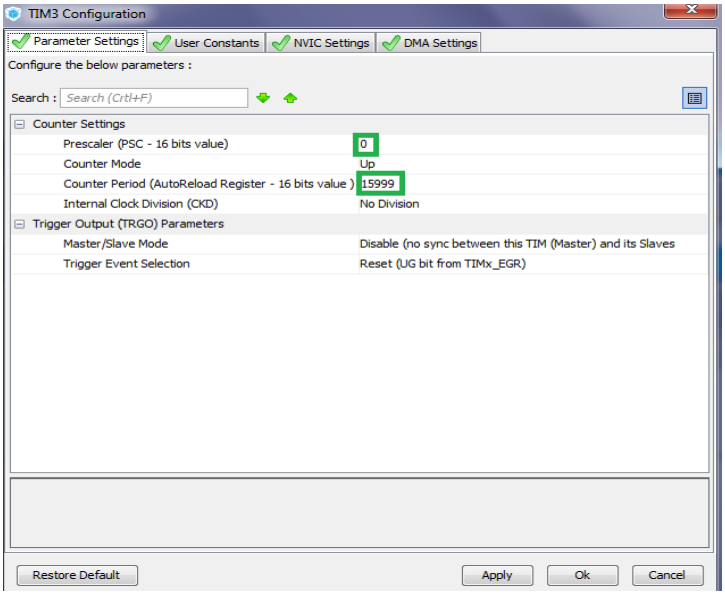

Figura 5. Configuración del Timer3.

Como el Timer3 generará una interrupción, es necesario habilitar la interrupción global del mismo. Esta configuración se encuentra en la pestaña **NVIC** de la misma ventana, como se muestra en la figura 6.

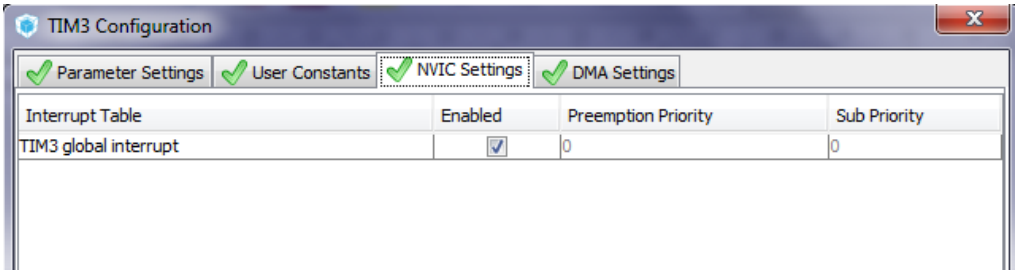

Figura 6. Habilitación de interrupción global del Timer3.

Para el ADC y DAC no se requiere de ninguna configuración especial, debido a que no se usarán recursos extras del microcontrolador.

A continuación se genera el proyecto en *Atollic TrueSTUDIO* mediante *STM32CubeMx*. Al generarlo, se muestra una ventana indicando que la compilación ha sido exitosa con la opción de abrir el código en el *IDE Atollic TrueSTUDIO,* como se observa en la figura 7.

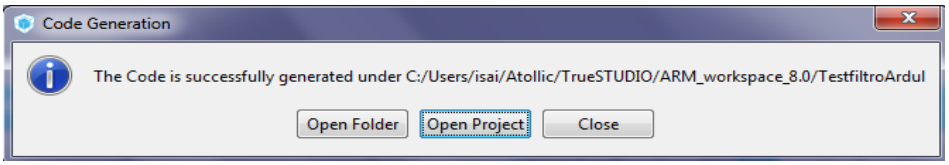

Figura 7. Generación del código en Atollic TrueSTUDIO.

Con Atollic TrueSTUDIO abierto se accede al archivo main.c. En este archivo se activa la interrupción del Timer3 y el funcionamiento del DAC. Esto se realiza ingresando las líneas de código que se muestran en la figura 8, en una sección (*USER CODE BEGIN*) antes del ciclo infinito *while(true)* del archivo main.

```
 96   /* Initialize all configured peripherals */
 97   MX_GPIO_Init();
 98   MX_ADC1_Init();
 99   MX_DAC_Init();
100   MX_TIM3_Init();
101
102   /* USER CODE BEGIN 2 */
103   HAL_TIM_Base_Start_IT(&htim3);
104   HAL_DAC_Start(&hdac, DAC_CHANNEL_2);
105   /* USER CODE END 2 */
```

Figura 8. Instrucciones para activar el Timer3 y DAC.

El siguiente paso es entrar al archivo stm32f4xx_it.c; en este archivo se encuentra el vector de interrupción del Timer3 (aproximadamente en la línea 190) como se observa en la figura 9.

```
187 /**
188  * @brief This function handles TIM3 global interrupt.
189  */
190 void TIM3_IRQHandler(void)
191 {
192   /* USER CODE BEGIN TIM3_IRQn 0 */
193
194   /* USER CODE END TIM3_IRQn 0 */
195   HAL_TIM_IRQHandler(&htim3);
196   /* USER CODE BEGIN TIM3_IRQn 1 */
197
198   /* USER CODE END TIM3_IRQn 1 */
199 }
200
201 /* USER CODE BEGIN 1 */
202
203 /* USER CODE END 1 */
```

Figura 9. Función que maneja la interrupción del Timer3.

En esta sección se escribe el algoritmo del sistema, como se muestra en figura 10.

```
/**
* @brief This function handles TIM3 global interrupt.
*/
void TIM3_IRQHandler(void)
{
  /* USER CODE BEGIN TIM3_IRQn 0 */

  /* USER CODE END TIM3_IRQn 0 */
  HAL_TIM_IRQHandler(&htim3);
  /* USER CODE BEGIN TIM3_IRQn 1 */
  uint16_t lectura;
  float yk=0, error;

  //Leemos del ADC
  HAL_ADC_Start(&hadc1);
  lectura=HAL_ADC_GetValue(&hadc1);
  HAL_ADC_Stop(&hadc1);
  //Realizamos la ecuacion
  error=(lectura-y[0]);
  yk=( (a[0]*y[0])+(a[1]*y[1])+(b[0]*error)+(b[1]*x[0])+(b[2]*x[1]) );
  //Escribimos en el dac
  HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R, (uint32_t) yk);
  //Hacemos el corrimiento
  y[1]=y[0];
  y[0]=yk;
  x[1]=x[0];
  x[0]=error;
  //Cambio del estado en el PIN para observar su señal
  HAL_GPIO_TogglePin(GPIOE, GPIO_PIN_9);
  /* USER CODE END TIM3_IRQn 1 */
}

/* USER CODE BEGIN 1 */

/* USER CODE END 1 */
```

Figura 10. Implementación del algoritmo.

La ecuación a implementar (ecuación 8), contiene factores que multiplican a cada termino *X(k-n)*. Estos factores corresponden a los coeficientes ***a1***, ***a2***, ***a3…aN*** que aparecen en la ecuación de diferencias lineales (ecuación 2). En el lenguaje C++, estos coeficientes se representan como un vector o arreglo de números flotantes llamado: ***a***. Donde ***a[0]*** guarda el valor del coeficiente *a1*, ***a[1]*** es igual al coeficiente *a2* y así sucesivamente. De igual forma esto se aplica para el arreglo los coeficientes ***bN*** que multiplican a los términos *Y(k-n)*.

Este arreglo de variables se declara de forma global, como se muestra en la figura 11, para poder acceder y guardar los datos de forma general.

```
/* USER CODE BEGIN 0 */
extern ADC_HandleTypeDef hadc1;
extern DAC_HandleTypeDef hdac;
const float a[2]={1.941764533584249, -0.941764533584249}, b[3]={0.0, 0.001764533584249, 0.001729594400696};
float y[2]={0.0, 0.0}, x[2]={0.0, 0.0};
/* USER CODE END 0 */
```

Figura 11. Arreglos y variables declaradas como globales.

Ahora ya se puede compilar el proyecto en Atollic TrueSTUDIO y generar el archivo *.HEX.

Como consejo adicional que ayudará a visualizar si el procesamiento de la ecuación se realiza con una frecuencia igual a la de frecuencia de muestreo propuesta (10KHz). Se recomienda incluir dentro del vector de interrupción una sentencia adicional que cambie el estado lógico de un pin cada vez que se ejecute la sección de código de la interrupción. Esto generará una señal cuadrada de la mitad de la frecuencia de muestreo. Si esto ocurre, entonces se comprueba que el microcontrolador realiza el procesamiento dentro de lo establecido (tiempo real).

Tome también en cuenta que la resolución del ADC y el DAC es de 12 bits, por lo que el valor numérico de los datos procesados se encontrará en el rango de entre 0 a 4095.

4. Resultados.

El ADC de Ophyra no soporta voltajes negativos de entrada, por lo tanto, se debe inyectar una señal de voltaje positivo para no dañarlo. El rango de voltaje que acepta el ADC es de entre 0 a 3.3 volts. Le figura 12 muestra la configuración del generador de funciones con los siguientes parámetros:

- Tipo de señal = cuadrada.
- Frecuencia = 20Hz.
- Voltaje de pico=0.6v.
- Voltaje de offset=0.8v.

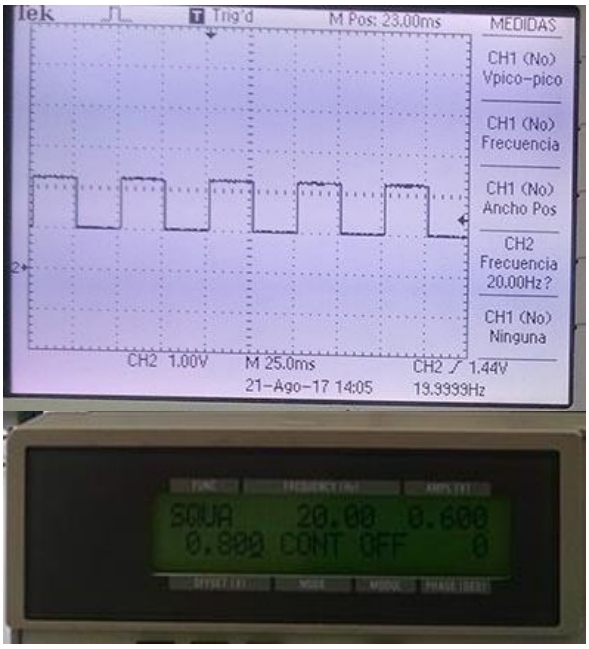

Figura 12. Ajuste del generador de funciones.

Conectando la seña cuadrada a Ophyra, como se muestra en la figura 13, y la salida del canal 2 del DAC a un osciloscopio, se puede observar la respuesta de salida (ecuación implementada) que simula el comportamiento de un motor de DC. En las figuras 14 y 15 se muestran a detalle las señales a visualizar en la práctica.

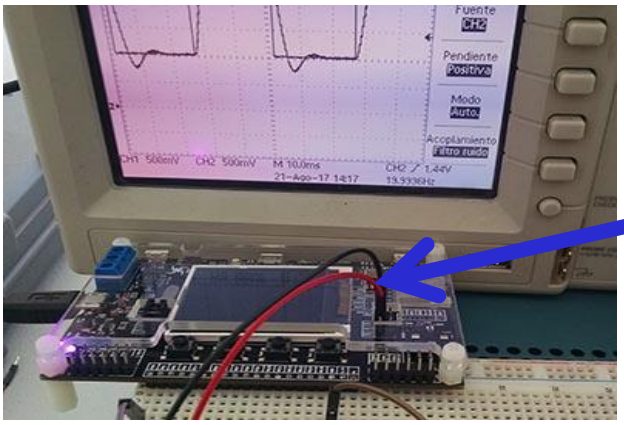

Figura 13. Conexión hacia Ophyra.

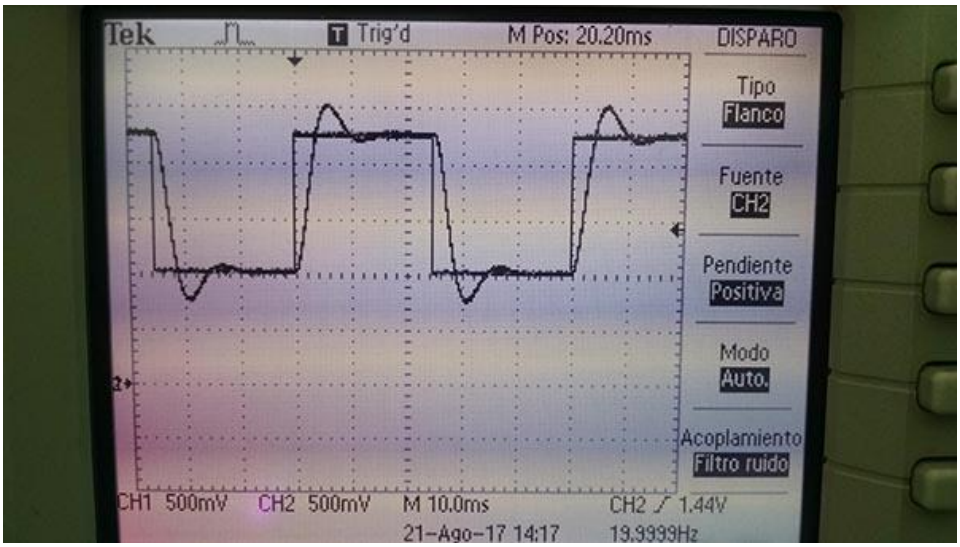

Figura 14. Salida del DAC.

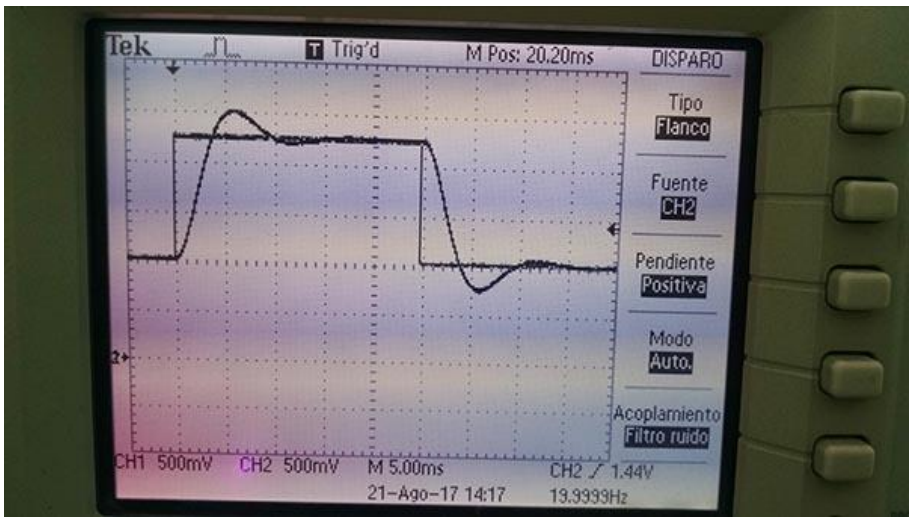

Figura 15. Salida del DAC.