CONFIGURACIÓN E IMPLEMENTACIÓN DE MÓDULOS INTERNOS DE LA STM32 - LABORATORIO III

Joaquin Jose Avila Pallares, Erick David Daleman Amaya, Fanhor Esteban Navia Forero
joaquinj.avilap@ecci.edu.co, erickd.dalemana@ecci.edu.co, fanhore.naviaf@ecci.edu.co

***Abstract* — Due to all the peripherals that the STM32 handles, Laboratory III consists of implementing the control of a geared motor by means of a matrix keyboard and that each key contains a defined condition so that the geared motor acts on a potentiometer varying its voltage value, allowing to define the Angle of a servomotor. For the control process of the laboratory proposal, the USART, ADC and TIMER will be configured, these being the fundamental objective of the report which seeks to configure and implement the internal modules of the STM32F103C8T6.**

Index Terms* — *Prescaler, Periodo, Microcontrolador, ADC, STM32, KEYPAD, PWM, TIMER, USART, GPIO, RX, TX.

I. INTRODUCCIÓN

El proceso de configuración e implementación de una STM32 se puede desarrollar desde diferentes herramientas de programación en la cual una u otra puede brindar un nivel de complejidad a la hora de desarrollar el código que se implementara en cualquier actividad o proceso que el programador desee ejecutar.

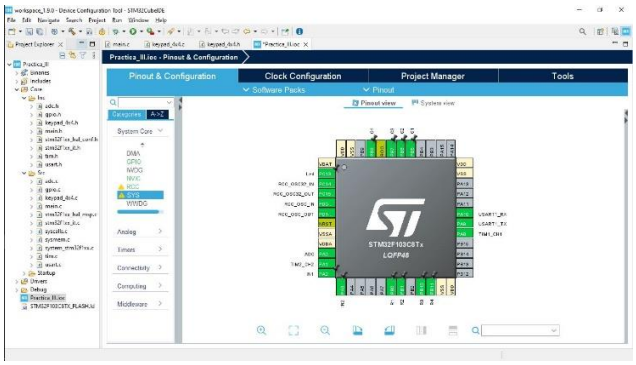

Fig. 1. Entorno de configuración de STM32CubeIDE.

El entorno de programación oficial que brinda STM32 a sus tarjetas de desarrollo y/o sistemas embebidos es el entorno de STM32CubeIDE, esta es una herramienta de desarrollo multi-OS todo en uno que permite la configuración de periférica, generación de código, compilación de código y funciones de depuración para microcontroladores y microprocesadores STM32 [1], [2].

En la figura 1 se puede observar el entorno de programación de STM32CubeIDE.

El laboratorio contará con el modo de operación del ADC quien permitirá ver los valores mínimos y máximos del ADC trabajando una conversión de voltaje de 3.3V (voltios), este se utilizará con DMA (Direct Memory Access). Con DMA, los resultados de conversión se pueden almacenar directamente en la RAM sin la participación de la CPU[3].

El otro punto fundaméntala del laboratorio se debe al manejo del PWM el cual se configura bajo el uso del TIMER que maneja la STM32, además también se implementara lectura de un teclado matricial 4x4 cuya lectura se tomara con la utilización de la librería KEYPAD 4x4 la cual es implementada por electrónica y circuitos[4] en el curso de programación de microcontroladores ARM en lenguaje C.

El laboratorio como anteriormente se ha nombrado se trabajara bajo la configuración de diferentes módulos internos de la STM32, con los que se permitirá tener el control de un motorreductor quien será el encargado de cambiar el ángulo de giro de un potenciómetro y este a su vez indicara de acuerdo al valor del ADC el ángulo que debe llegar el servo motor, cabe resaltar que para que se de ejecución de lo mencionado anteriormente el usuario debe de indicar por medio de una de las teclas del KEYPAD las distintas condiciones que se hallan programado en el microcontrolador.

II. MATERIALES

Los materiales del laboratorio que darán desarrollo al mismo son los siguientes:

- ✓ STM32F103C8T6.
- ✓ Programador STLINK V2.
- ✓ Convertidor USB a TTL CP2102.
- ✓ Motorreductor.
- ✓ Servomotor.
- ✓ Teclado matricial 4x4 (Membrana).
- ✓ Protoboard.

- ✓ Potenciómetro 50 kΩ.
- ✓ Modulo L298N.
- ✓ Fuente 12V (voltios).
- ✓ Engranaje de 18 dientes.
- ✓ Engranaje de 10 dietes.
- ✓ Jumpers.

III. METODOLOGÍA

Inicialmente para el desarrollo del laboratorio se procede a realizar la configuración de la STM32F103C8T6 en el espacio de configuración de pines para la tarjeta seleccionada, así como se puede observar en la figura 1 mencionada anteriormente. La configuración de los pines se implementa de la siguiente manera:

Teclado matricial: Se configura 4 pines como salida y 4 pines de entrada en modo Pull-up, dichos pines se nombrarán como dice la librería KEYPAD a utilizar, la cual indica que los pines se salida se nombren como R1, R2, R3, R4 y lo pines de entrada como C1, C2, C3, C4, los GPIO de salida representan las filas y los GPIO de entrada representan las columnas del teclado matricial [4], parte de la configuración de los pines la podemos observar en la figura 2.

Pin ...	Signal ...	GPIO o...	GPIO ...	GPIO P...	Maximu...	User La...	Modified
PA2	n/a	Low	Output ...	No pull-...	Low	IN1	☑
PA3	n/a	Low	Output ...	No pull-...	Low	IN2	☑
PB0	n/a	Low	Output ...	No pull-...	Low	R1	☑
PB1	n/a	Low	Output ...	No pull-...	Low	R2	☑
PB5	n/a	n/a	Input m...	Pull-up	n/a	C1	☑
PB6	n/a	n/a	Input m...	Pull-up	n/a	C2	☑
PB7	n/a	n/a	Input m...	Pull-up	n/a	C3	☑

Fig. 2. Configuración de GPIO teclado matricial.

Los pines del teclado matricial utilizados para el desarrollo del código se indican en la tabla 1.

Tabla 1. Definición de GPIO.

GPIO	Nombre del pin
PB0	R1
PB1	R2
PB10	R3
PB11	R4
PB5	C1
PB6	C2
PB7	C3
PB8	C4

Timer 1: Para la habilitación de este se configura el Clock Source como Internal Clock, seguidamente en los parámetros de configuración se procede a indicar el Prescaler y el periodo en el que el Timer 1 va trabajar asi como se puede observar en la figura 3. Cabe resaltar que este Timer será el encargado de enviar el acho de pulso al servo motor cuyo periférico está configurado como salida en el pin PA8 de la STM32.

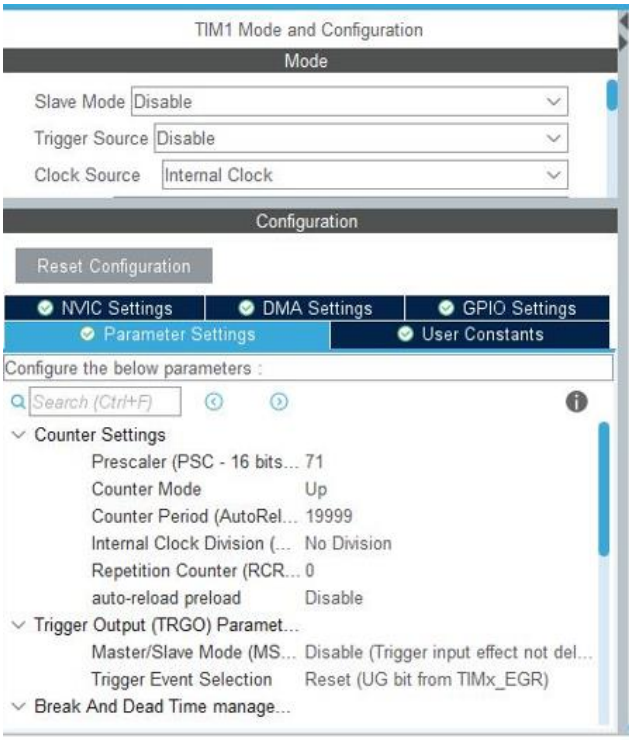

Fig. 3. Configuración TIMER 1 PWM_SERVO.

Como el servo motor trabaja a una frecuencia de 50Hz y la STM32F103C8T6 trabaja a una velocidad máxima de 72MHz con la que se estará desarrollando este laboratorio. Por lo tanto si definimos que la frecuencia del Timer será de 1MHz el Prescaler para la configuración del Timer 1 será la siguiente:

$$Prescaler = \left(\frac{Fc}{Ft}\right) - 1$$

$$Prescaler = \left(\frac{72MHz}{1MHz}\right) - 1$$

$$Prescaler = 71$$

Para la configuración del periodo como esta solo acepta el tiempo en segundo se procede a dividir 1 sobre los 50Hz el cual es la frecuencia del servo motor.

$$T = \left(\frac{1}{50Hz}\right) = 0.02\ seg$$

Obteniendo como resultado 0.02 segundos, teniendo la frecuencia en del motor en segundos se determina el periodo de Timer este se obtiene multiplicando en tiempo por la frecuencia del Timer la cual se definió de 1MHz y a esto se le resta 1 obteniendo como resultado el periodo máximo del Timer 1.

$$Periodo = (0.02seg * 1MHz) - 1$$

$$Periodo = 19999$$

Como 19999 es el 100% de ciclo del trabajo el PWM del servo motor tomara valores entre 0 a 19999, la relación de estos cálculos se tomó con respecto a la guía de configuración del PWM con Timer del B105 LAB [5] .

Timer 2 & ADC: Al igual que el Timer 1 su configuración es muy parecida la diferencia está en el que el ADC es el encargado de definir el periodo de trabajo del Timer para ello se realiza la lectura del ADC esta lectura del ADC tendrá un ciclo de entre 0 a 4096 este valor es debido a los 2^12 bits que tiene el ADC. Teniendo en cuenta lo anterior se configura el ADC en modo de conversión continua para la lectura de la muestra [6], en la configuración del Timer el Prescaler se define un valor de 2 y para el periodo el 100% del ciclo del trabajo que en este caso es la mayor muestra del ADC la cual de 4096, Cuya configuración se puede observar en la figura 4.

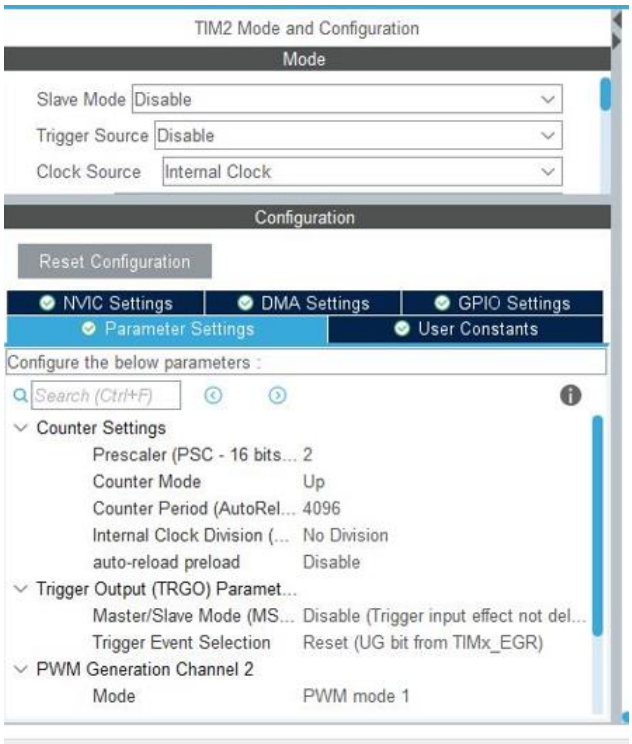

Fig. 4. Configuración Timer 2 PWM_MOTORREDUCTOR.

La salida del Timer 2 es un periférico el cual está en el pin PA1, este controla el PWM del motorreductor su parámetro será definido de acuerdo al comportamiento que se quiera para el motor.

USART1: La STM32F103C8T6 maneja 3 USART para el laboratorio se utilizará el USART1 cuya configuración consiste en definir el modo de comunicación asíncrona e indicar el Baud Rate del puerto serial la cual será de 115200 Bits/s asi como se puede ver en la figura 5.

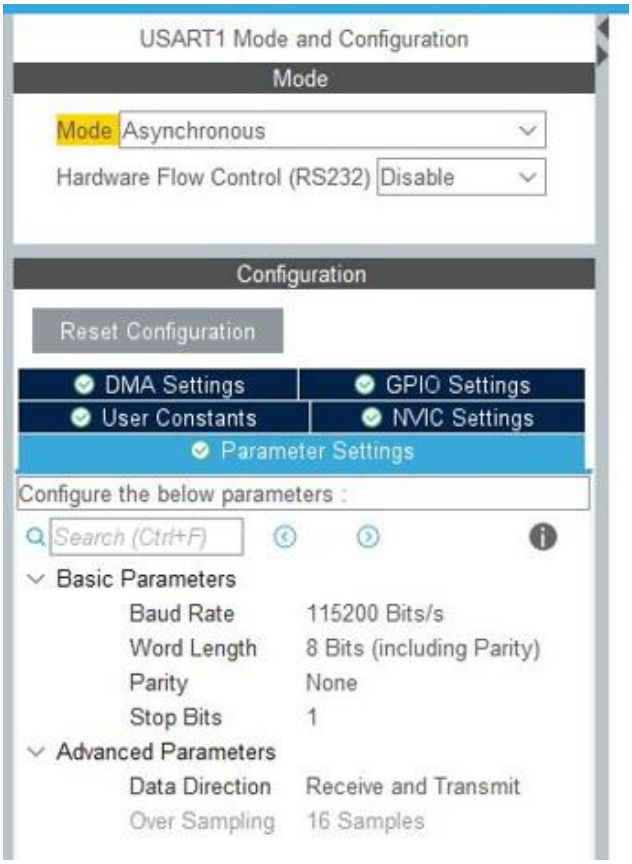

Fig. 5. Configuración USART1.

La comunicación serial de la STM32 se realizará con un convertidor de USB a TTL por medio de los pines TX y RX del convertidor y la tarjeta STM32 cuyos periféricos a utilizar son el pin PA9 y PA10 del microcontrolador.

Circuito: En la figura 6 se puede observar las conexiones del circuito a implementar para el laboratorio III.

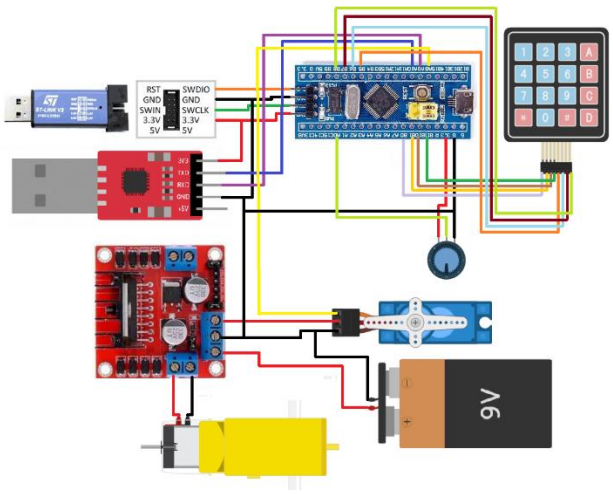

Fig. 6. Circuito Laboratorio III.

Teniendo en cuenta todas las configuraciones planteadas anteriormente, se debe indicar que para el desarrollo de este laboratorio se tuvieron en cuenta aspectos mecánicos como la transferencia de movimiento del motor para el cambio de voltaje del potenciómetro. Para ello se utilizo un engranaje de 10 dientes acoplado al motor y otro engranaje de 18 dientes acoplado al potenciómetro esto con el fin de tener un mejor control del potenciómetro de acuerdo a la relación de los piñones.

En cuanto al comportamiento del motorreductor se determinó el mínimo PWM que se le puede definir al motorreductor para poder mover la relación de los engranajes y trasmitir el movimiento.

Programación: Una vez se configuran los pines de la tarjeta se procede a generar el código base del programa según las configuraciones planteadas anteriormente, una vez generado el código se agregan o incluyen las librerías necesarias para el desarrollo de la programación.

```
19 /* Includes ------------------------
20 #include "main.h"
21 #include "adc.h"
22 #include "tim.h"
23 #include "usart.h"
24 #include "gpio.h"
25
26 /* Private includes ---------------
27 /* USER CODE BEGIN Includes */
28 #include <stdio.h>
29 #include <string.h>
30 #include "keypad_4x4.h"
31 /* USER CODE END Includes */
32
```

Fig. 7. Librerías Agregadas.

En la figura 7 se observa como se incluye la librería de interés keypad_4x4 esta librería se incluye de manera manual agregando el archivo punto h y c de la librería keypad_4x4 asi como se puede observar en la figura 8 señalando cada archivo con la flecha roja, las librerías que están desde la línea de código 20 a la 24 son agregadas automáticamente al generar el código base asi como se observa en la figura 7.

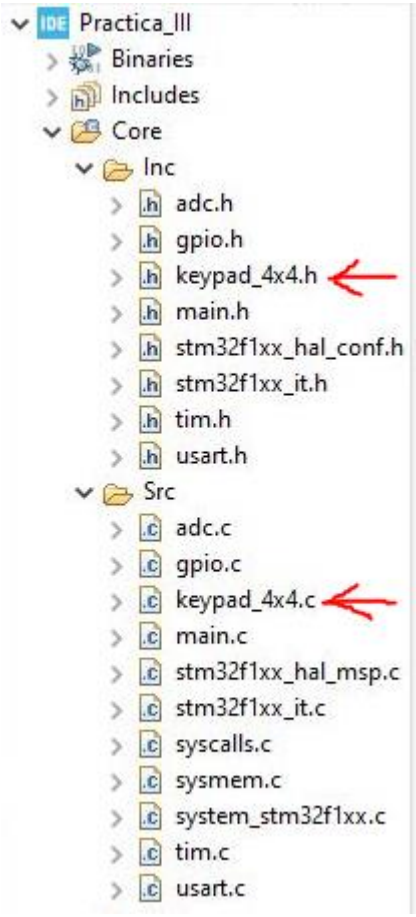

Fig. 8. Librería keypad_4x4 agregada manualmente.

Después de incluir la librería definimos las variables que utilizaremos en el código estas se pueden observar en la figura 9.

```
51 /* USER CODE BEGIN PV */
52 char tecla, dato[500];
53 int PWM_MOTOR = 2400, ADC = 0, t = 0, angulo = 0, ag = 0;
54 float Potenciometro = 0;
55 /* USER CODE END PV */
56
```

Fig. 9. Variables.

Luego se definen las funciones de control para el control y configuración de los pines asi como se puede ver en la figura 10.

```
/* Private function prototypes -----
void SystemClock_Config(void);
/* USER CODE BEGIN PFP */
void Configuraciones_Iniciales();
void keypad();
void condiciones();
void conversion_Pot_Ang();
void servo_angulo(uint8_t ang);
/* USER CODE END PFP */
```

Fig. 10.Definición de funciones.

Seguidamente en la figura 11 se puede ver que en el main inicializamos las funciones de las configuraciones iniciales del programa.

```
main.c  keypad_4x4.c  keypad_4x4
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------

  /* Reset of all peripherals, Init
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peri
  MX_GPIO_Init();
  MX_ADC1_Init();
  MX_TIM1_Init();
  MX_TIM2_Init();
  MX_USART1_UART_Init();
  /* USER CODE BEGIN 2 */
  Configuraciones_Iniciales();
  /* USER CODE END 2 */
```

Fig. 11. Inicialización de funciones

De allí se procede a configurar la función de Configuraciones_Iniciales asi como se puede ver en la figura 12, en esta función inicializamos las entradas y salidas en estado bajo, también en esta inicialización se llama a la función conversion_Pot_Ang para determinar cuál es la señal de voltaje tiene este y de acuerdo con ello poder devolver el potenciómetro a estado bajo.

```
main.c  keypad_4x4.c  keypad_4x4.h  *Practica_III.ioc
/* USER CODE BEGIN 4 */
void Configuraciones_Iniciales(){
    sprintf(dato,"**********El mejor control del mundo**********\r\n");
     HAL_UART_Transmit(&huart1, (uint8_t*)dato, strlen(dato), HAL_MAX_DELAY);
     HAL_GPIO_WritePin(Led_GPIO_Port, Led_Pin, 1);
     HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 0);
     HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 0);
     HAL_TIM_Base_Start(&htim1);
     HAL_TIM_Base_Start(&htim2);
     HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
     HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_2);
     TIM1->CCR1 = 0;
     TIM2->CCR2 = 0;
     conversion_Pot_Ang();
     t = (900/3.16)*Potenciometro;
     TIM2->CCR2 = PWM_MOTOR;
     HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 1);
     HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 0);
     HAL_Delay(t);
     HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 0);
     HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 0);
     HAL_Delay(10);
```

Fig. 12. Configuraciones_Iniciales.

Después de inicializar las entradas y salidas, en la función Keypad se ejecuta la lectura del teclado, las condiciones que se generan de acuerdo a la tecla presionada se realiza la conversión del potenciómetro a ángulo y finalmente se posiciona el ángulo al que debe llegar el servo motor, esta configuración del keypad se puede ver en la figura 13. Cabe resaltar que en la función keypad se llama a la función keypad_Get_Char la cual determina cual tecla fue presionada, esta función está atada a la librería keypad_4x4.h quien hace un barrido de las filas y columnas y asi identificar la letra presionada.

```
void keypad()
{                    Practica_III/Core/Src/keypad_4x4.c
    tecla = Keypad_Get_Char();
    if (tecla !=0){
        sprintf(dato,"Tecla: %c, ",tecla);
        HAL_UART_Transmit(&huart1, (uint8_t*)dato, strlen(dato), HAL_MAX_DELAY);
    }
    condiciones();
    conversion_Pot_Ang();
    servo_angulo(angulo);
}
```

Fig. 13. Función Keypad.

En la función de condiciones se tiene estado que se generara al presionar cada tecla asi como se observa en la figura 14 la cual muestra que si se presiona la tecla “C” el motor va girar un tiempo de 900 milisegundos con un PWM definido en 2400 cuyo parámetro es el mínimo requerido para poder mover la relación de los engranajes y asi variar el potenciómetro.

```
202 void condiciones(){
203     switch (tecla){
204     case 'A':
205         HAL_GPIO_WritePin(Led_GPIO_Port, Led_Pin, 0);
206         sprintf(dato,"Led encendido, ");
207         HAL_UART_Transmit(&huart1, (uint8_t*)dato, strlen(dato), HAL_MAX_DELAY);
208         break;
209     case 'B':
210         HAL_GPIO_WritePin(Led_GPIO_Port, Led_Pin, 1);
211         sprintf(dato,"Led apagado, ");
212         HAL_UART_Transmit(&huart1, (uint8_t*)dato, strlen(dato), HAL_MAX_DELAY);
213         break;
214     case 'C':
215         t =(900/3.19)*3.19;
216         TIM2->CCR2 = PWM_MOTOR;
217         HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 0);
218         HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 1);
219         HAL_Delay(t);
220         HAL_GPIO_WritePin(IN1_GPIO_Port, IN1_Pin, 0);
221         HAL_GPIO_WritePin(IN2_GPIO_Port, IN2_Pin, 0);
222         HAL_Delay(10);
223         break;
```

Fig. 14. Función de condiciones.

La línea de código 215 es t almacena la función de la linealización del tiempo respecto al voltaje indicado, en este caso como el voltaje es de 3,19V el motor debe girar 900 milisegundos que es el máximo tiempo para llevar el voltaje del potenciómetro a los 3,19V y este valor determinara que el ángulo del servo motor este a 180 grados.

La tabla 2 muestra el estado que se genera en el servo motor al momento de presionar una tecla.

Tabla 2. Condiciones según la tecla presionada.

Tecla	Voltaje	Posición servo motor (Grados)
0	0	0
C	3,19	180
D	1,595	90

En cuanto a la función conversión_Pot_Ang esta como su nombre lo indica se realiza la conversión del voltaje medido con respecto a la conversión del ADC, este en la línea de código 247 es encendido el **&hadc1** luego en la línea 249 se lee y se al almacena en la variable ADC el valor leído del **&hadc1.**

Luego de realizar la lectura del ADC se realiza la conversión a voltaje y este a su vez es almacenado en la variable potenciómetro y luego pasar ese voltaje a ángulo asi como se puede observar en la figura 15 las líneas de código 250 y 251 realizan la linealización tanto para el voltaje con respecto al ADC leído y a su vez para el ángulo con respecto a la variación del potenciómetro.

main.c | keypad_4x4.c | keypad_4x4.h | *Practica_III.ioc

```
246 void conversion_Pot_Ang(){
247     HAL_ADC_Start(&hadc1);
248     HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
249     ADC = HAL_ADC_GetValue(&hadc1);
250     Potenciometro = (float)((ADC*3.3)/4095.0);
251     angulo = (180/3.19)*Potenciometro;
252     sprintf(dato,"ADC: %i, Potenciometro: %0.2f, Angulo: %i ",ADC, Potenciometro, angulo);
253     HAL_UART_Transmit(&huart1, (uint8_t*)dato, strlen(dato), HAL_MAX_DELAY);
254 }
```

Fig. 15.Función de conversion_Pot_Ang.

La función servo_angulo recibe el valor del ángulo y este valor en linealizado de acuerdo al pulso mínimo y máximo que puede recibir el servo motor almacenando pulso en la variable PWM_SERVO luego este pulso es apuntado al Timer 1 quien es el que envía el ancho de pulso al servo motor, lo mencionado anteriormente se puede ver en la figura 16.

```
255 void servo_angulo(uint8_t ang){
256     uint16_t PWM_SERVO;
257     PWM_SERVO =(uint16_t)((ang-0)*(Pulso_Max - Pulso_Min)/(180-0) + Pulso_Min);
258     TIM1->CCR1 = PWM_SERVO;
259     sprintf(dato,", PWM_SERVO: %i \r\n",PWM_SERVO);
260     HAL_UART_Transmit(&huart1, (uint8_t*)dato, strlen(dato), HAL_MAX_DELAY);
261 }
262 /* USER CODE END 4 */
```

Fig. 16. Función servo_angulo.

Para finalizar cabe resaltar que el código que genera el programa STM32CubeIDE presenta una estructura de código en la que cada parte del código que se realice debe ser especificada en el espacio de código asignado, de lo contrario si al realizar una nueva configuración en los pines y se vuelve actualizar la estructura el código que estuviese fuera del espacio asignado se borrara.

IV. RESULTADOS

A continuación, el a figura 17 se puede observar el circuito realizado físicamente este funciona de manera adecuada sin embargo se recomienda tener en cuenta las referencias de voltajes a utilizar para cada unos de los componentes, trabajar el motorreductor y el servo motor con una alimentación externa es la manera adecuada para tener una eficiencia del comportamiento del circuito desarrollado para el laboratorio.

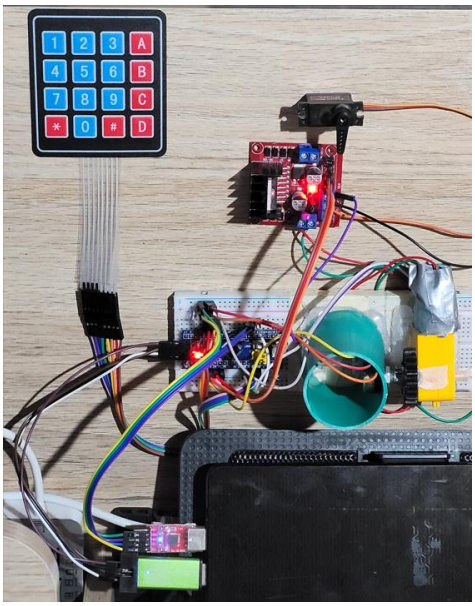

Fig. 17. Montaje físico del Circuito Laboratorio III.

El voltaje de alimentación del motorreductor es de 12V (voltios) este voltaje es suministrado al Driver (L298N) de control de giro del motorreductor por medio de una fuente de alimentación, el Driver cuenta con una salida de 5V (voltios), con esta salida de voltaje se alimenta al servo motor.

La visualización de los datos tomados por el ADC y la linealización del este para determinar el valor de voltaje del potenciómetro, el Angulo y el valor de PWM del servo motor se puede ver por medio del monitor serial de Hércules asi como se observa en la figura 18.

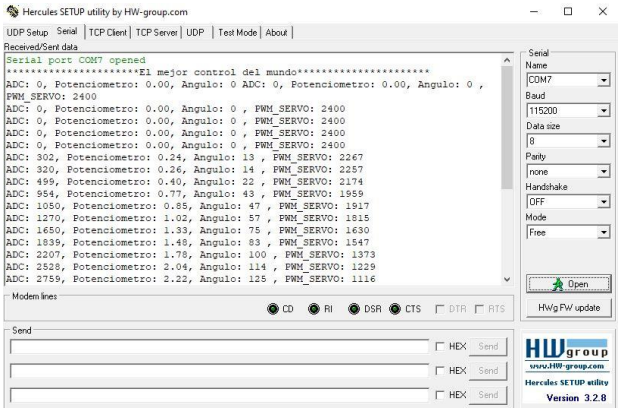

Fig. 18. Visualización de datos por monitor serial Hércules.

En la figura 18 se puede ver la visualización de los datos enviados desde el microcontrolador por medio del monitor serial, en esta figura se observa que a medida que se varía el dato del ADC se varía los datos del voltaje del potenciómetro, ángulo, y PWM servo motor, Cabe resaltar que, por cuestiones de ejemplo, el potenciómetro fue variado manualmente y poder tener la guía de explicación de los datos visualizados en la figura 18.

El estado de conversión o linealización de las funciones se comporta de manera adecuada sin embargo se debe realizar algunos ajustes debido a una variación del 0,01 grado que no permite visualizar por monitor serial el estado correcto en el que el servo motor se encuentra, físicamente este sitúa en la medida asignada por el usuario.

V. CONCLUSIONES

La relación mecánica de los engranajes se debe acoplar de manera que no queda demasiado juego entre los dientes de los engranajes esto permite generar mayor exactitud a la medida o el valor al que se quiere llegar.

La referencia a tierra debe ser la misma para todo el circuito esto evita que se generen fallos o no se ejecute el funcionamiento del servo motor u otro actuador.

Con solo la lectura del ADC se puede dar configuración a los demás periféricos que se esté utilizando, las funciones de linealización utilizadas son de parte fundamental para la ejecución adecuada del código y poder tener una conversión adecuada en relación a la medida que se quiera obtener.

Para finalizar se debe tener en cuenta la velocidad de transmisión serial que se este utilizando en la tarjeta ya que esta debe ser la misma velocidad que se debe configurar en el monitor serial para la correcta visualización de los datos mor medio del monitor serial.

VI. REFERENCIAS

[1] ST, “Introduction to STM32CubeIDE - stm32mcu,” *www.st.com*, 2023. https://wiki.stmicroelectronics.cn/stm32mcu/wiki/STM32CubeIDE:Introduction_to_STM32CubeIDE#What_is_STM32CubeIDE-3F (accessed Apr. 02, 2023).

[2] ST, “STM32CubeIDE - Integrated Development Environment for STM32 - STMicroelectronics,” *www.st.com*, 2023. https://www.st.com/en/development-tools/stm32cubeide.html (accessed Apr. 02, 2023).

[3] Embeddedexplorer, “Using STM32 ADC with STM32CubeIDE and HAL driver,” *embeddedexplorer.com*, 2022. https://embeddedexplorer.com/using-stm32-adc-with-stm32cubeide-and-hal-driver/ (accessed Apr. 02, 2023).

[4] Electronica y circuitos, “(73) 19.- MANEJO DE TECLADO MATRICIAL 4X4 - CURSO MICROCONTROLADORES ARM EN C - YouTube,” *YouTube*, 2022. https://www.youtube.com/watch?v=nqqQV_OqmTo&t=176s (accessed Apr. 02, 2023).

[5] B105 LAB, “STM32F4: Configurar el PWM con Timers | B105 lab,” *elb105.com*, 2018. https://elb105.com/stm32f4-configurar-el-pwm-con-timers/ (accessed Apr. 02, 2023).

[6] B105 LAB, “STM32F4: Configuración y uso básico del ADC | B105 lab,” *elb105*, 2018. https://elb105.com/stm32f4-configuracion-y-uso-basico-del-adc/ (accessed Apr. 02, 2023).