Sistema de Riego por Medio de IOT y con LM35

Leo Danny Sánchez Arcila y Alejandro Bañol Escobar.
Universidad Tecnológica de Pereira.
Sistemas Digitales IV
22/07/2021.

***Resumen*—La aplicación IOT consiste en controlar un sistema de riego conformado por dos motobombas las cuales, a partir de las lecturas de dos sensores de temperatura se encienden y apagan. Por lo tanto, al sobrepasar cierta temperatura se activa la motobomba correspondiente. De igual manera, se tiene un modo manual para encender y apagar las motobombas cuando se requiera, ignorando las lecturas de los sensores.**

Abstract - The IOT application consists of controlling an irrigation system made up of two motor pumps which, based on the readings of two temperature sensors, turn on and off. Therefore, when a certain temperature is exceeded, the corresponding motor pump is activated. Similarly, there is a manual mode to turn the motor pumps on and off when required, ignoring the sensor readings.

Palabras clave- IOT, LM35, Python, Temperatura, sensores.
***Index Terms*—IOT, LM35, Python, Temperature, sensors.**

I. Introducción

El proyecto consta de dos sensores de temperatura (LM35), cada uno asociado a una motobomba que sirven para monitorear y controlar un sistema de riego. Para controlar el sistema se hace uso de una aplicación móvil llamada MqttDash que permite crear una conexión con el bróker. Además, se puede suscribir uno a un tópico y crear botones, textos y demás para mandar o recibir datos. La aplicación recrea el funcionamiento de MqttLens.

EL modo automático consiste en encender la motobomba correspondiente cuando alguno de los sensores detecte una temperatura mayor o igual a 35 grados y también permite apagar la motobomba correspondiente cuando alguno de los sensores detecte una temperatura menor a 35 grados.

El modo manual permite ignorar la lectura de los sensores y de esta forma poder encender cualquiera de las motobombas de manera remota. Además, el dato se va a almacenar en la base de datos, una vez haya un cambio de 1 grado en la temperatura, tanto si está en el modo manual como en el automático.

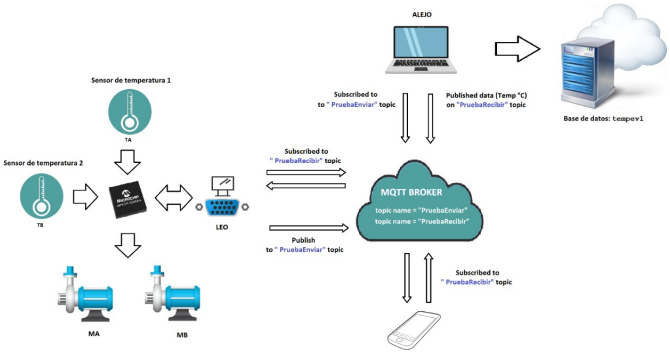


Figura 1. Diagrama de bloques del sistema de riego.

II. Procedimiento.

En esta sección se especifican los códigos dispuestos en Mplab y en Python para enviar y recibir datos a través del serial y de Mqtt. Además, se especifican los cálculos para el acondicionamiento de los sensores y las motobombas, el diseño dela interfaz por medio de la aplicación MqttDash y la creación de la base de datos.

II-A. Acondicionamiento de los sensores:

Se tiene un rango de temperatura admisible de $0°$ a $50°$ y se quiere llegar a un rango de 0 a 5 V. Como la temperatura esta en grados, toca hacer la conversión a voltajes, para así poder hacer la relación y acondicionar.

Según el datasheet del LM35:

$$Vo = \frac{(10mV)}{(°C)} * Tin$$

Por consiguiente, el voltaje de salida para una temperatura de entrada de $0°es$:

$$Vo = \frac{(10mV)}{(°C)} * 0° = 0V$$

El voltaje de salida para una temperatura de entrada de $50°es$:

$$Vo = \frac{(10mV)}{(°C)} * 50° = 0{,}5V$$

Teniendo esto, se hace el acondicionamiento, cambiando entre los siguientes rangos:

$$[0V \quad 0{,}5V] \Rightarrow [0V \quad 5V]$$

Ahora, calculando la ecuación de la recta se tienen los valores de los elementos que conforman al circuito sumador restador, como se muestra en la figura 2.

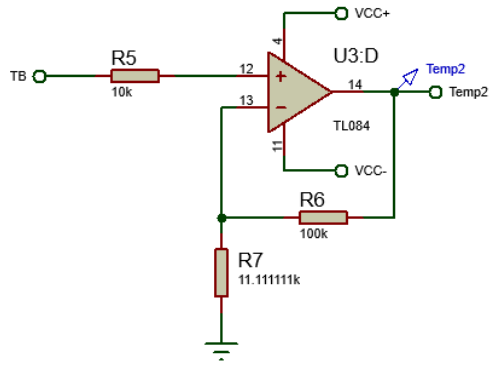


Figura 2. Esquemático del acondicionamiento de los sensores.

II-B. Acondicionamiento de las motobombas:

Para este caso, se hace un aislamiento eléctrico por medio del optoacoplador 4N25, para de esta forma separar la parte de control de la parte de potencia. Luego, por medio del puente H L293D se alimentan ambos motores, los cuales son motores dc de 12V. La figura 3 muestra el esquemático de esta parte de potencia.

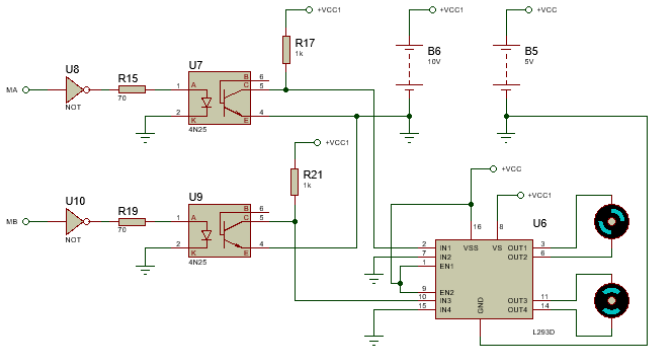


Figura 3. Esquemático del acondicionamiento de los motores.

II-C. Programa en Mplab:

Para esto, se crea el programa en Mplab que permita adquirir los datos que se ingresan por el puerto análogo del microcontrolador. Se crea una cabecera que contiene la función para la configuración del ADC y la función para hacer la conversión del ADC. Se crea una cabecera para la configuración de UART y se anexan las cabeceras para los retardos, la configuración del reloj, la configuración de los pragmas, para los registros del micro y para la biblioteca estándar del lenguaje de programación C.

En la función principal se hace el llamado a las funciones de configuración del reloj, del UART y del ADC. Posteriormente, se declaran las variables que intervendrán en el programa como se muestra en la siguiente tabla.

Variable	Tipo de dato	Descripción
dato	char	Guarda el valor que reciba del búfer de recepción del UART
dato2	int	Sirve para diferenciar entre el modo automático y el manual
Conversion	long	Guarda el valor que se obtiene de la conversión del ADC.
Temp1	float	Guarda el valor que se obtiene al hacer la conversión a temperatura del sensor A
Temp2	float	Guarda el valor que se obtiene al hacer la conversión a temperatura del sensor B
T1	float	Sirve para evaluar un condicional que permite mostrar los datos solo cuando ocurra un cambio en el sensor A.
T2	float	Sirve para evaluar un condicional que permite mostrar los datos solo cuando ocurra un cambio en el sensor B.
C1	int	Permite identificar cuando esta encendido el motor A
A1	int	Permite identificar cuando esta encendido el motor B

Cuadro I
VARIABLES QUE INTERVIENEN EN EL PROGRAMA.

Luego, se hace la configuración de los puertos del micro, para lo cual a partir del registro TRIS se declara todo el puerto B como salida.

Ahora, dentro del ciclo infinito se empieza por almacenar en la variable dato el valor que retorna la función UART_ENVIAR, la cual pregunta si hubo desborde en la recepción y de cumplirse este caso se restablecerá el búfer de recepción al estado vacío. Por último, pregunta si hay datos en el búfer de recepción y de cumplirse este caso retorna el valor que se encuentre en el registro receptor del UART (U1RXREG).

La siguiente tabla muestra los caracteres que se eligieron para los dos modos de funcionamiento y para el encendido y apagado de los motores.

Carácter	Funcionamiento
E	Activar modo automático
M	Activar modo manual
C	Encender motor A si está en modo manual
P	Apagar motor A si está en modo manual
A	Encender motor B si está en modo manual
B	Apagar motor B si está en modo manual

Cuadro II
CARACTERES QUE DESCRIBEN EL FUNCIONAMIENTO DEL PROGRAMA.

Después, se crea un switch case que lee la variable dato y sirve para distinguir si se seleccionó el modo automático o el manual. Por lo tanto, si el caso es una 'E' pone la variable dato2 en cero y si el caso es una 'M' pone la variable dato2

en 1. Si dato2 es igual a cero, indica que se activó el modo automático y si dato2 es igual a 1, indica que se activo el modo manual.

Luego lo que se hace es leer los canales análogos del ADC que para este caso serian 2, ya que se van a adquirir los datos proporcionados por dos sensores. El sensor A se conecta al pin AN0 del ADC y el sensor B se conecta al pin AN1 del ADC. Por lo tanto, teniendo esto en cuenta se deben muestrear los sensores por separado, por lo tanto, primero se muestreo el pin ANO y se guarda la conversión en la variable Temp1 y luego se muestrea el pin AN1 y se guarda la conversión en la variable Temp2.

Entrando más en detalle, lo que se hace en cada muestreo es primero activar el ADC a través del registro ADON, luego a través del registro CHOSA se selecciona el pin a muestrear, por consiguiente, para muestrear el pin AN0 se pone el registro CHOSA en 0 y para el pin AN1 se pone el registro en 1. Luego, se almacena en la variable conversión el valor que retorna la función ConverADC y en la variable Temperatura1 o 2 se almacena el valor que resulta de la multiplicación entre la conversión y 0.0122 y finalmente, se esperan 600 milisegundos para que se complete bien la conversión.

El calculo de la resolución del ADC para lograr convertir los datos de voltaje a temperatura, es la siguiente:

$$\frac{V}{T} = 10x10^{-3}$$

$$T = 100 * V$$

$$V = \%resolución * Conversión$$

$$V = \frac{(V_{ref}^{+} - V_{ref}^{-})}{2^n - 1}$$

$$V = \frac{(0{,}5 - 0)}{2^{12} - 1} = 0{,}000122$$

$$V = 0{,}000122 * Conversión$$

$$T = 100 * (0{,}000122 * Conversión) = 0{,}0122 * Conversión$$

Ahora, se crea un condicional que permite evaluar si la temperatura esta dentro del rango permitido, que es entre 0 y 50 grados; si se cumple dicha condición, se crea un condicional que pregunta si Temp1 es diferente a T1 y si se cumple, imprime por el serial los valores de Temp1, Temp2, estado de los dos motores y el carácter que indica el modo en el que se encuentra. Después, se iguala la variable T1 a la variable Temp1. Por último, se desactiva el ADC igualando el ADON a cero.

Luego, se hace el mismo proceso para muestrear los datos del sensor B y ahora que ya se tienen los valores de Temp1 y Temp2, si dato2 es igual a cero, indica que se activó el modo automático. Por lo tanto, lo que se hace en este caso es hacer condicionales para definir a que temperatura se prenden y apagan los motores. LA siguiente tabla muestra este proceso.

Condición	Acción	Descripción
Temp1 $\geq 35{,}0 \&\& Temp2 \geq 35{,}0$	PORTB = 0x0003 Delay (300ms)	Encender ambos motores y esperar 300 milisegundos
Temp1 $\geq 35{,}0 \&\& Temp2 \prec 35{,}0$	PORTB = 0x0001 Delay (300ms)	Encender motor A, apagar motor B y esperar 300 ms
Temp1 $\prec 35{,}0 \&\& Temp2 \geq 35{,}0$	PORTB = 0x0002 Delay (300ms)	Encender motor B, apagar motor A y esperar 300 ms
Temp1 $\prec 35{,}0 \&\& Temp2 \prec 35{,}0$	PORTB = 0x0000 Delay (300ms)	Apagar ambos motores y esperar 300 milisegundos

Cuadro III
ESTADO DE LOS MOTORES DEPENDIENDO DE LA TEMPERATURA.

Terminando el modo automático, vuelve y se lee el valor dispuesto en serial y, por consiguiente, si dato es igual a 'M' quiere decir que entramos en modo manual, por lo cual vamos a poder encender los motores de forma independiente sin necesidad de tener en cuenta la temperatura. Por lo tanto, se crean condicionales que evalúen si se encuentra en modo manual y si se solicito encender o apagar algunos de los motores. Este proceso se muestra en la siguiente tabla.

Condición	Acción	Descripción
dato == 'C' && dato2 == 1	LATBbits.LATB0 = 1 delay (20ms), imprimir mensaje y C1 = 1	Encender el motor A, esperar 20 ms e imprimir por serial el mensaje
dato == 'P' && dato2 == 1	LATBbits.LATB0 = 0, delay (20ms), imprimir mensaje y C1 = 0	Apagar el motor A, esperar 20 ms e imprimir por serial el mensaje
dato == 'A' && dato2 == 1	LATBbits.LATB1 = 1, delay (20ms), imprimir mensaje y A1 = 1	Encender el motor B, esperar 20 ms e imprimir por serial el mensaje
dato == 'B' && dato2 == 1	LATBbits.LATB1 = 0, delay (20ms), imprimir mensaje y A1 = 0	Apagar el motor B, esperar 20 ms e imprimir por serial el mensaje

Cuadro IV
ESTADO DE LOS MOTORES ESTANDO EN MODO MANUAL.

Con esto se da por terminado el programa implementado en Mplab para la adquisición de las lecturas de los sensores de temperatura y el accionamiento de las motobombas.

II-D. Programa en Python:

En Python, se implementa un código que permita leer el puerto serial y publicar esto a través de un tópico con fin de

que se pueda guardar esta información de forma remota en una base de datos. Además, se suscribe a otro tópico para leer la información allí publicada por la aplicación y escribir esta información en el puerto serial.

El código consiste en importar primero las librería paho.mqtt.client para conectarse a mqtt, la librería serial para comunicarnos por vía serial y la librería threading para poder hacer otro subproceso. Luego, se crea el objeto puerto con el cual se abre el puerto serial y se define el puerto que para este caso es el COM2 ya que en Proteus esta definido el COM1 y luego se le da la velocidad de 9600 Baudios. Luego se crea la función recibirUART que se encarga de leer el puerto serial, guarda la información en la variable data y siempre y cuando haya información convierte el data a string y lo publica en el tópico **PruebaEnviar**.

Ahora, se crean dos funciones predefinidas, con on_connect se suscribe a un tópico que para este caso se nombro **PruebaRecibir** y con on_message se leen los mensajes publicados en el tópico al que se esta suscrito y se escribe este dato en el puerto serial. Por otro lado, se crea una función llamada tarea2 que se encarga de generar un bucle infinito para que el programa siga leyendo el tópico aunque se este ejecutando otro proceso. Luego, se crea el cliente, me conecto al tópico y mando por serial lo que llegue a ese tópico, luego le digo cual es el servidor, el puerto y el tiempo de espera.

Luego se crea el hilo que sirve para poder ejecutar dos procesos automáticamente. Por lo tanto, se define una variable t que crea o ejecuta la tarea2 y luego se inicia ese subproceso declarando t.start(). Finalmente, se llama a la función recibirUART. La figura 4 muestra el diagrama de flujo del programa principal.

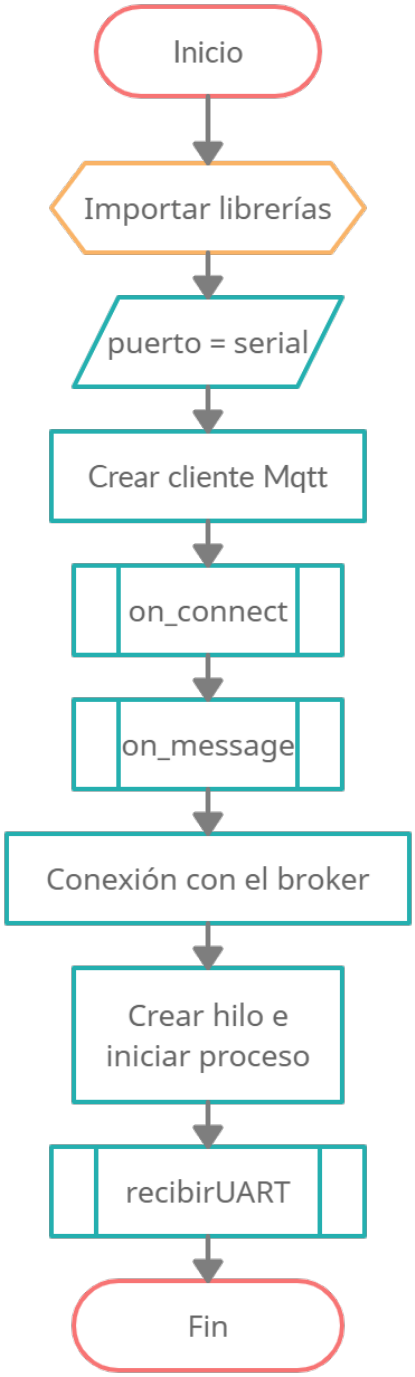


Figura 4. Programa principal para envió de información al Mqtt y al serial.

La figura 5 muestra el diagrama de flujo de la función **recibirUART**

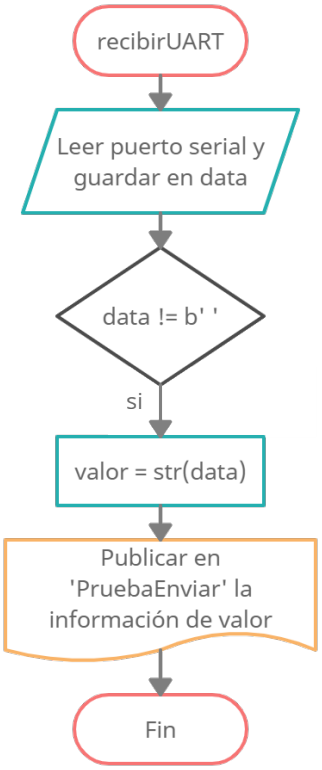


Figura 5. Diagrama de flujo de la función recibirUART.

La figura 6 muestra el diagrama de flujo de las funciones **on_connect**, **on_message** y **tarea2**

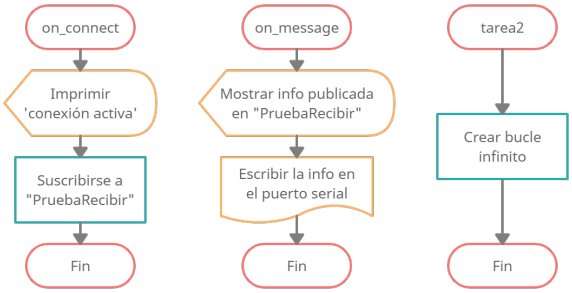


Figura 6. Diagramas de flujo de las funciones on_connect, on_message y tarea2.

II-E. Programa en Python para guardar la información en DB:

Este programa se implementa para guardar la información enviada a través del tópico **PruebaEnviar** en la base de datos generada en PhpMyAdmin. Además, también se publica en el tópico **DataT1** el valor de la temperatura del sensor A y en el topico **DataT2** el valor de la temperatura del sensor B.

En general, se empieza importando las librerías para la conexión con Mqtt y con Mysql. Luego, se define la base de datos creada previamente en MySQL y se crea el objeto para ejecutar los comandos SQL. Ahora, se suscribe al tópico **PruebaEnviar** y se lee la información publicada en este, la cual se decodifica por medio de la función split. Por lo tanto, se obtiene una lista con la información separada y, por consiguiente, se guarda dicha información en variables diferentes que corresponden a la temperatura 1 y 2, el modo de funcionamiento en el que se encuentra y los estados de los motores A y B. Luego se publica la información de la temperatura en otros dos tópicos para poderla visualizar en la APP y finalmente, se guarda la información en la base de datos por medio de los comandos SQL. La figura 7 muestra el diagrama de flujo del programa principal.

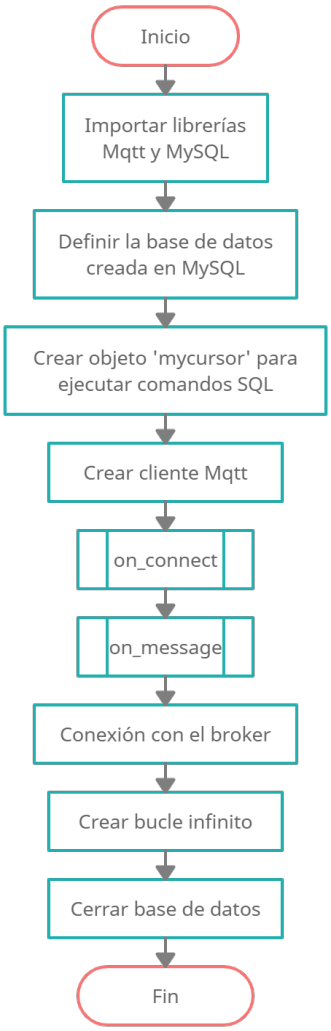


Figura 7. Programa principal para almacenar información en la base de datos.

La figura 8 muestra el diagrama de flujo de la función **on_message**

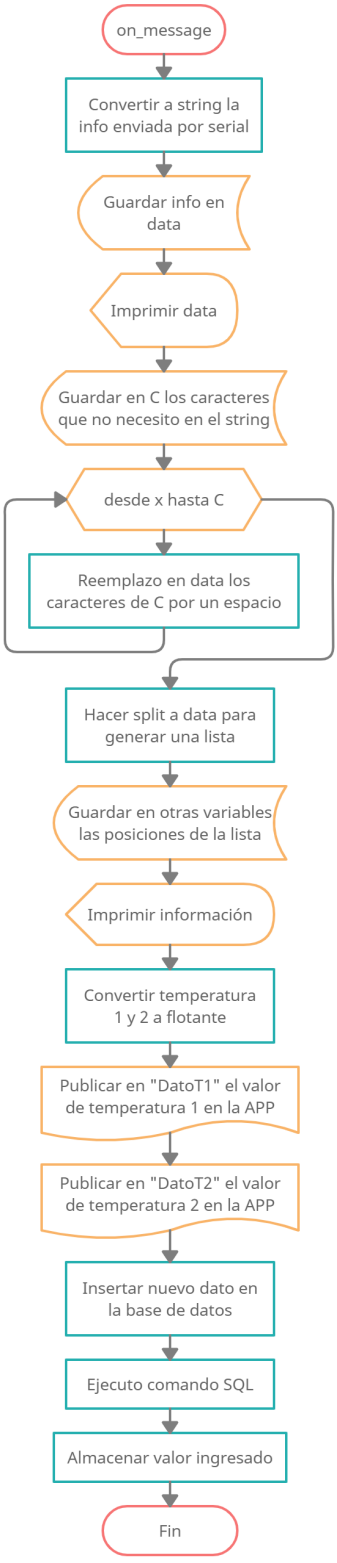


Figura 8. Diagrama de flujo de la función on_message.

La figura 9 muestra el diagrama de flujo de la función **on_connect**

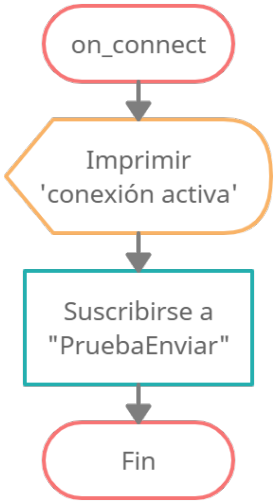


Figura 9. Diagrama de flujo de la función on_connect.

II-F. Diseño de la interfaz en la aplicación:

Para usar la aplicación, primero se define el nombre de la conexión y la dirección del broker. Luego, se crean tres Switch/button, uno para controlar el encendido y apagado del motor A, el otro para controlar el encendido y apagado del motor B y el ultimo para seleccionar entre modo automático y manual. Por último, se crean 2 Range/progress, uno para visualizar la temperatura del sensor A suscrito al tópico **DataT1** y el otro para visualizar la temperatura del sensor Bsuscrito al tópico **DataT2**. Los tres Switch/button se suscriben al topico **PruebaRecibir**. La figura 10 muestra el diseño de esta interfaz.

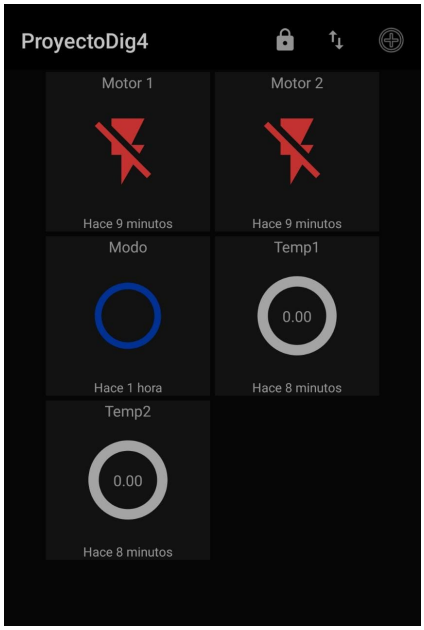


Figura 10. Interfaz de la aplicación.

II-G. Creación de la base de datos:

Para la creación de la base de datos, se hace a través de PhpMyAdmin, la cual es una herramienta con la que se puede manejar y administrar bases de datos MySQL. La figura 11, muestra la estructura de la base de datos.

Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra
id	int(200)			No	*Ninguna*		AUTO_INCREMENT
TemppSA	varchar(15)	latin1_spanish_ci		No	*Ninguna*		
TemppSB	varchar(15)	latin1_spanish_ci		No	*Ninguna*		
StateMA	varchar(15)	latin1_spanish_ci		No	*Ninguna*		
StateMB	varchar(15)	latin1_spanish_ci		No	*Ninguna*		
MODO	varchar(15)	latin1_spanish_ci		No	*Ninguna*		
Hora	time			No	current_timestamp()		
Fecha	date			No	current_timestamp()		

Figura 11. Estructura de la base de datos.

Como se observa en la figura 11, se van a guardar los datos de temperatura, los estados de las motobombas, el modo de funcionamiento, la hora y la fecha en la que se subió la información.

III. Pruebas

Para la realización de las pruebas, se diseña el esquemático que se muestra en la figura 12. La activación y desactivación de los motores se simulo a partir de leds para poder visualizar si se encuentran encendidos o apagados.

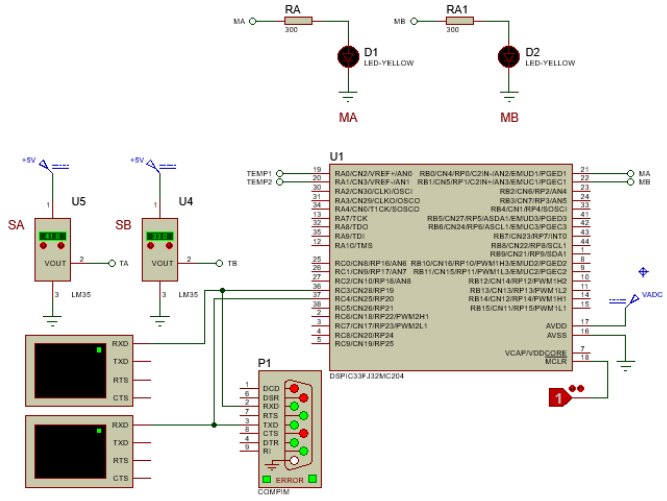


Figura 12. Esquemático general del proyecto

la siguiente figura muestra que apenas se inicia la simulación, el sistema empieza en modo automático y como se observa para una temperatura en el sensor A de 5.12 grados el motor A se encuentra apagado y para una temperatura en el sensor B de 36.16 grados, el motor B se encuentra encendido.

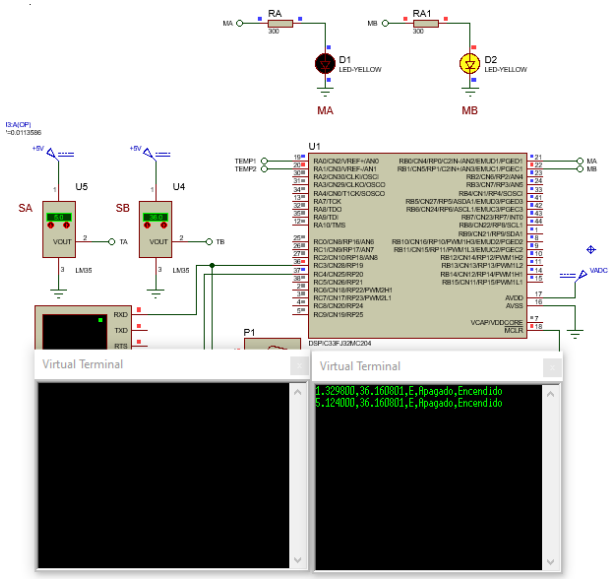


Figura 13. Modo Automático, MA Apagado y MB Encendido.

Ahora, la siguiente figura muestra el almacenamiento de esta información en la base de datos.

+ Opciones

	ID	TemppSA	TemppSB	StateMA	StateMB	MODO	Hora	FECHA
Editar Copiar Borrar	84	1.329800	36.160801	Apagado	Encendido	E	21:51:06	2021-07-20
Editar Copiar Borrar	85	5.124000	36.160801	Apagado	Encendido	E	21:51:08	2021-07-20

Figura 14. Base de datos, Automático, MA Apagado y MB Encendido.

la siguiente figura que para una temperatura en el sensor A de 37.17 grados, el motor A se encuentra Encendido y para una temperatura en el sensor B de 38.17 grados, el motor B se encuentra Encendido.

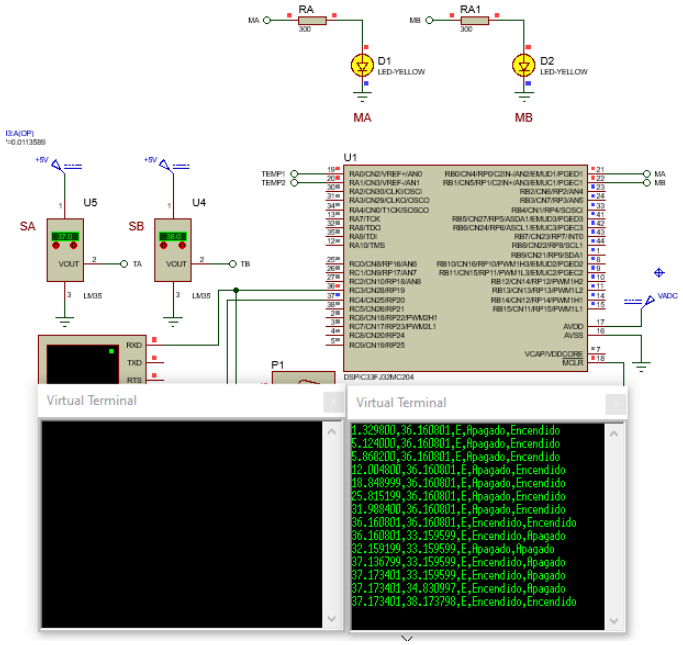


Figura 15. Modo Automático, MA Encendido y MB Encendido.

Ahora, la siguiente figura muestra el almacenamiento de esta información en la base de datos.

	ID	TemppSA	TemppSB	StateMA	StateMB	MODO	Hora	FECHA
Editar Copiar Borrar	93	32.159199	33.159599	Apagado	Apagado	E	22:09:20	2021-07-20
Editar Copiar Borrar	94	37.136799	33.159599	Encendido	Apagado	E	22:10:31	2021-07-20
Editar Copiar Borrar	95	37.173401	33.159599	Encendido	Apagado	E	22:10:36	2021-07-20
Editar Copiar Borrar	96	37.173401	34.830997	Encendido	Apagado	E	22:14:05	2021-07-20
Editar Copiar Borrar	97	37.173401	38.173798	Encendido	Encendido	E	22:14:09	2021-07-20

Figura 16. Base de datos, Automático, MA Encendido y MB Encendido.

La siguiente imagen muestra que al mandar una M el sistema cambia a modo manual y los motores se inicializan apagados, sin importar el valor de temperatura que se tenga.

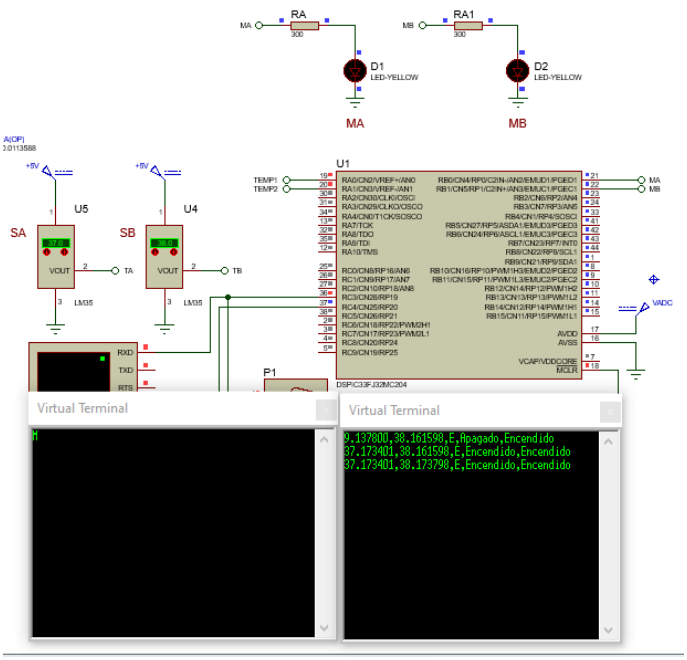


Figura 17. Activación del modo manual.

Ahora, la siguiente imagen muestra que cuando se envía una C El motor A se enciende.

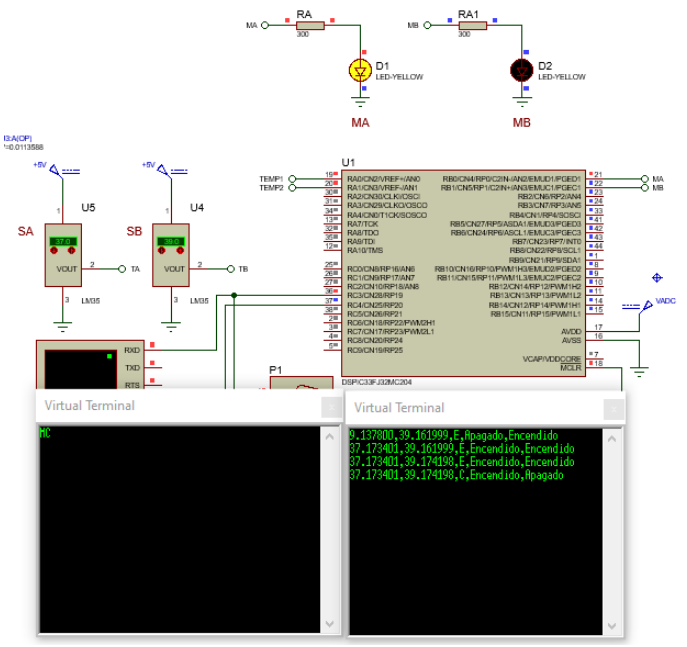


Figura 18. Manual, Encender MA con C.

La siguiente imagen muestra el almacenamiento en la base de datos de esta información.

	ID	TemppSA	TemppSB	StateMA	StateMB	MODO	Hora	FECHA
Editar Copiar Borrar	115	9.137800	39.161999	Apagado	Encendido	E	22:32:59	2021-07-20
Editar Copiar Borrar	116	37.173401	39.161999	Encendido	Encendido	E	22:33:01	2021-07-20
Editar Copiar Borrar	117	37.173401	39.174198	Encendido	Encendido	E	22:33:03	2021-07-20
Editar Copiar Borrar	118	37.173401	39.174198	Encendido	Apagado	C	22:33:26	2021-07-20

Figura 19. Base de datos, Manual, Encender MA con C.

Ahora, la siguiente imagen muestra que cuando se envía una A El motor B se enciende y como anteriormente se había encendido el motor A, este permanece encendido.

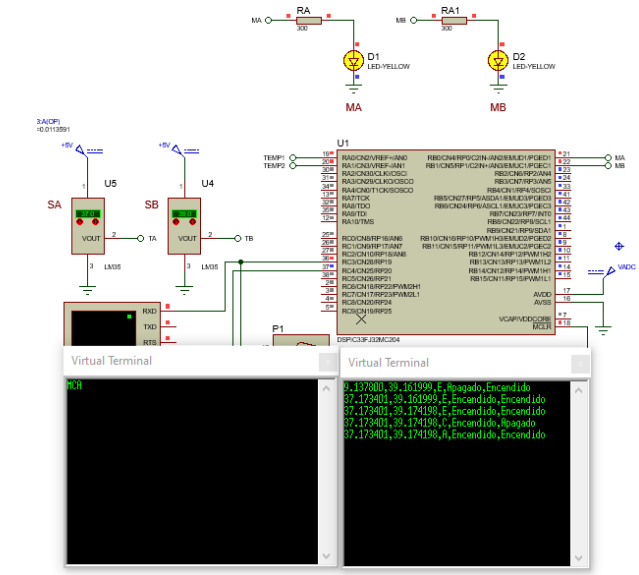
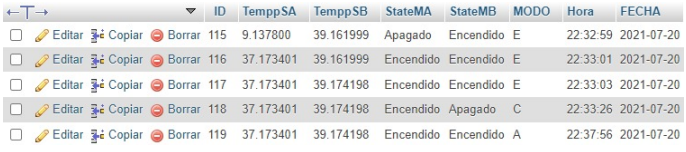


Figura 20. Manual, Encender MB con A.

La siguiente imagen muestra el almacenamiento en la base de datos de esta información.



	ID	TemppSA	TemppSB	StateMA	StateMB	MODO	Hora	FECHA
Editar Copiar Borrar	115	9.137800	39.161999	Apagado	Encendido	E	22:32:59	2021-07-20
Editar Copiar Borrar	116	37.173401	39.161999	Encendido	Encendido	E	22:33:01	2021-07-20
Editar Copiar Borrar	117	37.173401	39.174198	Encendido	Encendido	E	22:33:03	2021-07-20
Editar Copiar Borrar	118	37.173401	39.174198	Encendido	Apagado	C	22:33:26	2021-07-20
Editar Copiar Borrar	119	37.173401	39.174198	Encendido	Encendido	A	22:37:56	2021-07-20

Figura 21. Base de datos, Manual, Encender MB con A.

IV. Conclusiones

- Es importante contar con adecuada metodología de evaluación del desempeño de todo el proyecto, ya que si se realiza de forma adecuada, se pueden encontrar fallas o aspectos que se pueden mejorar. En el caso de encontrarse errores también se debe contar con una adecuada forma de depuración, ya que por ser un proyecto el cual combina diferentes elementos como hardware, gestión de información, protocolos de comunicación, conexión a un servidor y un aplicativo. Por lo tanto, el poder hallar una solución puede ser algo difícil, tal como sucedió en algunas ocasiones, cuando por ejemplo ciertos problemas que se presentaban desde el código en python se lograron solucionar desde el código del dsPIC.

- Antes de dar inicio al desarrollo de los códigos que permitan el funcionamiento del sistema, es de vital importancia tener claridad sobre todos los elementos que componen el sistema, donde se debe definir que dispositivos envían o reciben información, como aquellos que cumplen ambas funciones y como se maneja y almacena la información. En el caso del proyecto esto ayudo a determinar la cantidad de tópicos requeridos, a cuales se debía realizar la subscripción y a cuales se publicaba la información. Después como se tenia información que combinaba caracteres y números, se busco la forma de enviar todo junto y por ultimo una forma de decodificar la información.

- Para encender ambas motobombas, fue necesario dejar todo el puerto B como salida y a la hora de mostrar por el puerto, se declara el pin de forma hexadecimal. Por lo tanto, para el caso en que se dé que ambas motobombas se deben encender, entonces se manda en hexadecimal 0003 lo cual indica que se van a encender los pines RB0 y RB1. Se hizo de esta manera, ya que al encender cada pin por separado presento el problema de que solo se encendía el ultimo pin RB1 y el otro pin solo se veía titilar en determinado tiempo.

- Para encender los motores de forma manual toco enviar caracteres diferentes para cada acción, por ejemplo, para encender el motor 1 se manda la C y para apagarlo la P y para encender el motor 2 se manda la A y para apagarlo la B. De esta manera funciona el encendido de ambos sensores por separado, pero no encienden ambos motores a la misma vez, cuando se enciende el motor B se apaga el motor A sin indicarle que se apague e igual sucede cuando se enciende el otro motor. Para solucionar esto, se usó el registro LAT para el puerto B haciendo que ya no se lea el puerto, sino que se escriba en cada pin el valor que se desea.

- A la hora de subir la información a la base de datos se complico un poco ya que se estuvo mandando solo los datos de temperatura, por consiguiente a la hora de definir el estado de los motores se complico un hacerlo desde python. Por lo tanto, se opto por cambiar en Mplab la forma en la que se imprimía la información, para lo cual se mando por el serial la información de la temperatura, el estado de las motobombas y el modo de funcionamiento.

Referencias

[1] Nathanial, ”Python/MySQL query error: ‘Unknown column”, Stack overflow. [Online]. Available: https://stackoverflow.com/questions/19544015/python-mysql-query-error-unknown-column

[2] Luis Llamas, ”¿QUÉ ES MQTT? SU IMPORTANCIA COMO PROTOCOLO IOT”, luisllamas.es. [Online]. Available: https://www.luisllamas.es/que-es-mqtt-su-importancia-como-protocolo-iot/

[3] MakerElectronico, ÏO puertos digitales TRIS, PORT, LAT”, makerelectronico.com. [Online]. Available: https://www.makerelectronico.com/io-puertos-digitales-tris-port-lat-con-pic18f4550/

[4] Gabriel Canepa, Üso de las funciones split y join en Python”, blog.carreralinux.com.ar. [Online]. Available: https://blog.carreralinux.com.ar/2017/07/uso-split-y-join-python/